

System Generator for DSP

リファレンス ガイド

UG638 (v11.4) 2009 年 12 月 2 日



Xilinx is disclosing this user guide, manual, release note, and/or specification (the "Documentation") to you solely for use in the development of designs to operate with Xilinx hardware devices. You may not reproduce, distribute, republish, download, display, post, or transmit the Documentation in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Xilinx expressly disclaims any liability arising out of your use of the Documentation. Xilinx reserves the right, at its sole discretion, to change the Documentation without notice at any time. Xilinx assumes no obligation to correct any errors contained in the Documentation, or to advise you of any corrections or updates. Xilinx expressly disclaims any liability in connection with technical support or assistance that may be provided to you in connection with the Information.

THE DOCUMENTATION IS DISCLOSED TO YOU "AS-IS" WITH NO WARRANTY OF ANY KIND. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE DOCUMENTATION, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT OF THIRD-PARTY RIGHTS. IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOSS OF DATA OR LOST PROFITS, ARISING FROM YOUR USE OF THE DOCUMENTATION.

© Copyright 2002-2009 Xilinx, Inc. All rights reserved.

Xilinx, the Xilinx logo, the Brand Window, Virtex, Spartan, CoolRunner, ISE, and other designated brands included herein are trademarks of Xilinx, Inc. Certain other third-party trademarks are used under license, for further information, see <http://japan.xilinx.com/legal.htm>. All other trademarks are the property of their respective owners.

本資料は英語版 (v.11.4) を翻訳したもので、内容に相違が生じる場合には原文を優先します。

資料によっては英語版の更新に対応していないものがあります。

日本語版は参考用としてご使用の上、最新情報につきましては、必ず最新英語版をご参照ください。

目次

このマニュアルについて

マニュアルの内容	17
System Generator の PDF マニュアル セット	17
その他のリソース	17
表記規則.....	18
書体.....	18
オンライン マニュアル.....	19

第 1 章：ザイリンクス ブロックセット

ブロックセット ライブラリの構造	22
Basic Elements ブロック	22
Communication ブロック	25
Control Logic ブロック	26
Data Types ブロック	28
DSP ブロック	29
Index ブロック	31
Math ブロック	39
Memory ブロック	41
Shared Memory ブロック	42
Tools ブロック	43
System Generator でサポートされる Simulink ブロック	44
ブロックのパラメータ ダイアログ ボックスの共通オプション	45
[Precision]	45
[Type]	45
[Number of bits]	45
[Binary point]	45
[Overflow] と [Quantization]	45
[Latency]	46
[Provide synchronous reset port]	46
[Provide enable port]	47
[Sample Period]	47
[Use behavioral HDL (otherwise use core)]	47
[Use core placement information]	47
[Use XtremeDSP Slice]	47
[Placement]	48
[FPGA Area (slices, FFs, BRAMs, LUTs, IOBs, emb. mults, TBUFs)] / [Use area above for estimation]	48
ブロックのリファレンス ページ	49
Accumulator	50
ブロック インターフェイス	50
ブロック パラメータ	50
ザイリンクス LogiCORE	51
Addressable Shift Register	52
ブロック インターフェイス	52
ブロック パラメータ	53
ザイリンクス LogiCORE	53
AddSub	54
ブロック パラメータ	54
ザイリンクス LogiCORE	54
Assert	56
ブロック パラメータ	56

Assert ブロックを使用したレートおよびデータ型の改善方法	56
BitBasher	58
ブロック パラメータ	58
サポートされる Verilog 構文	58
制限	60
Black Box	61
ブラック ボックスの HDL 要件	61
Black Box ブロックのコンフィギュレーション ウィザード	63
Black Box ブロックのコンフィギュレーション M 関数	63
サンプル周期	65
ブロック パラメータ	66
HDL 協調シミュレーションのためのデータ型変換	67
例	68
関連項目	68
ChipScope	69
ハードウェアとソフトウェア条件	69
ブロック パラメータ	70
ChipScope プロジェクト ファイル	71
ChipScope から MATLAB ワークスペースへのデータのインポート	71
既知の問題	72
詳細情報	72
CIC Compiler 1.2	73
ブロック パラメータのダイアログ ボックス	73
ザイリンクス LogiCORE	74
CIC Compiler 1.3	75
ブロック パラメータ ダイアログ ボックス	75
ザイリンクス LogiCORE	76
Clock Enable Probe	77
Clock Probe	79
CMult	80
ブロック パラメータ	80
ザイリンクス LogiCORE	81
Complex Multiplier 3.0	82
ブロック パラメータ ダイアログ ボックス	82
ザイリンクス LogiCORE	83
Complex Multiplier 3.1	84
ブロック パラメータ ダイアログ ボックス	84
ザイリンクス LogiCORE	85
Concat	86
ブロック インターフェイス	86
ブロック パラメータ	86
Configurable Subsystem Manager	87
ブロック パラメータ	88
Constant	89
ブロック パラメータ	89
付録: DSP48 の制御命令形式	90
Convert	92
ブロック パラメータ	92
Convolution Encoder 6.1	94
ブロック インターフェイス	94
ブロック パラメータ ダイアログ ボックス	94
ザイリンクス LogiCORE	95
Convolution Encoder 7.0	96
ブロック パラメータ ダイアログ ボックス	96
ザイリンクス LogiCORE	97
CORDIC 4.0	98

ブロック パラメータ ダイアログ ボックス	98
Xilinx LogiCORE	100
Counter	101
ブロック パラメータ	102
ザイリンクス LogiCORE	103
DAFIR v9_0	104
ブロック インターフェイス	104
係数のリロード	105
係数のリロード用のオプション ポート	105
ブロック パラメータ ダイアログ ボックス	106
ザイリンクス LogiCORE	107
DDS Compiler 2.1	108
ブロック インターフェイス	109
ブロック パラメータ	110
ザイリンクス LogiCORE	111
DDS Compiler 3.0	112
ブロック インターフェイス	113
ブロック パラメータ	114
ザイリンクス LogiCORE	115
DDS Compiler 4.0	116
アーキテクチャの概要	116
ブロック インターフェイス	117
ブロック パラメータ	118
ザイリンクス LogiCORE	121
Delay	122
ブロック パラメータ	122
ビヘイビア レベルの HDL を使用したロジック合成	123
構造レベルの HDL を使用したロジック合成	123
大きい遅延のインプリメント	125
リセット可能な遅延と初期値	125
ザイリンクス LogiCORE	126
Depuncture	127
ブロック パラメータ	128
Disregard Subsystem	129
Divider Generator 2.0	130
ブロック パラメータ	130
ザイリンクス LogiCORE	131
Divider Generator 3.0	132
ブロック パラメータ	132
ザイリンクス LogiCORE	133
Down Sample	134
レイテンシ 0 の場合	135
レイテンシがある場合	135
ブロック パラメータ	136
ザイリンクス LogiCORE	136
DSP48	137
ブロック パラメータ	137
関連項目	139
DSP48 Macro	140
ブロック インターフェイス	140
ブロック パラメータ	140
DSP48 Macro ブロックの opmode の入力	141
パイプライン オプションの入力とカスタム パイプライン オプションの変更	148
DSP48 Macro の制限	148
関連項目	148
DSP48 Macro 2.0	149

ブロック パラメータ	149
ザイリンクス LogiCORE	153
関連項目	153
DSP48A	154
ブロック パラメータ	154
関連項目	156
DSP48E	157
ブロック パラメータ	157
関連項目	161
Dual Port RAM	162
ブロック インターフェイス	162
ブロック パラメータ	164
ザイリンクス LogiCORE	166
EDK Processor	168
メモリ マップ インターフェイス	168
ブロック パラメータ	169
既知の問題	171
MicroBlaze プロセッサのオンライン マニュアル	171
Expression	172
ブロック パラメータ	172
Fast Fourier Transform 6.0	173
計算方法	173
ブロック インターフェイス	173
ブロック パラメータ	176
ブロックのタイミング	177
ザイリンクス LogiCORE	178
Fast Fourier Transform 7.0	179
計算方法	179
ブロック インターフェイス	179
ブロック パラメータ	182
ブロックのタイミング	184
ザイリンクス LogiCORE	184
FDATool	185
使用例	185
FDATool のインターフェイス	185
FIFO	187
ブロック パラメータ	187
ザイリンクス LogiCORE	187
	187
FIR Compiler 4.0	188
ブロック インターフェイス	188
ブロック パラメータ	189
ザイリンクス LogiCORE	194
既知の問題	195
FIR Compiler 5.0	196
ブロック インターフェイス	196
ブロック パラメータ	197
ザイリンクス LogiCORE	202
From FIFO	203
ブロック パラメータ	203
ザイリンクス LogiCORE	204
関連項目	204
From Register	205
ブロック パラメータ	205
クロック ドメインの切り替え	205
関連項目	206

Gateway In	207
Gateway ブロック	207
ブロック パラメータ	207
Gateway Out	209
Gateway ブロック	209
ブロック パラメータ	209
Indeterminate Probe	211
Interleaver Deinterleaver 5.0	212
ブロック インターフェイス	213
ブロック パラメータ	213
ザイリンクス LogiCORE	214
Interleaver Deinterleaver 5.1	215
ブロック インターフェイス	216
ブロック パラメータ	216
ザイリンクス LogiCORE	217
Inverter	218
ブロック パラメータ	218
JTAG Co-Simulation	219
ブロック パラメータ	219
LFSR	221
ブロック インターフェイス	221
ブロック パラメータ	221
Logical	223
ブロック パラメータ	223
ザイリンクス LogiCORE	223
MCode	224
MCode ブロックのコンフィギュレーション	224
MATLAB 言語のサポート	225
ブロック パラメータ ダイアログ ボックス	244
MicroBlaze Processor	245
ブロック インターフェイス	245
ブロック パラメータ	248
MicroBlaze ソフトウェアの問題	250
既知の問題	252
MicroBlaze プロセッサのオンライン マニュアル	253
関連項目	253
ModelSim	254
ブロック パラメータ	254
タイム スケール	256
Mult	259
ブロック パラメータ	259
ザイリンクス LogiCORE	260
Multiple Subsystem Generator	261
ブロック パラメータ	261
デザインの生成	261
複数クロックのサポート	264
生成ファイル	265
Mux	266
ブロック パラメータ	266
Negate	267
ブロック パラメータ	267
Network-based Ethernet Co-Simulation	268
ブロック パラメータ	268
関連項目	269
Opmode	270
ブロック パラメータ	270

ザイリンクス LogiCORE	271
DSP48 の制御命令形式	271
DSP48E の制御命令形式	272
Parallel to Serial	273
ブロック インターフェイス	273
ブロック パラメータ	273
Pause Simulation	274
ブロック パラメータ	274
PicoBlaze Instruction Display	275
ブロック インターフェイス	275
ブロック パラメータ	275
ザイリンクス LogiCORE	275
PicoBlaze Microcontroller	276
ブロック インターフェイス	276
ブロック パラメータ	276
PicoBlaze アセンブラの使用法	277
既知の問題	277
PicoBlaze Microprocessor のオンライン マニュアル	277
Point-to-point Ethernet Co-Simulation	278
ブロック パラメータ	278
関連項目	279
Puncture	280
ブロック パラメータ	280
Reed-Solomon Decoder 6.1	282
ブロック インターフェイス	282
ブロック パラメータ	283
ザイリンクス LogiCORE	285
Reed-Solomon Decoder 7.0	286
ブロック インターフェイス	286
ブロック パラメータ	288
ザイリンクス LogiCORE	290
Reed-Solomon Encoder 6.1	291
ブロック インターフェイス	292
ブロック パラメータ	292
ザイリンクス LogiCORE	294
Reed-Solomon Encoder 7.0	295
ブロック インターフェイス	296
ブロック パラメータ	296
ザイリンクス LogiCORE	299
Register	300
ブロック インターフェイス	300
ブロック パラメータ	300
ザイリンクス LogiCORE	300
Reinterpret	301
ブロック パラメータ	301
Relational	302
ブロック パラメータ	302
ザイリンクス LogiCORE	302
Reset Generator	303
ブロック パラメータ	303
Resource Estimator	304
ブロック パラメータ	304
リソース概算の実行	305
Resource Estimation ブロックでサポートされるブロック	305
ISE レポートの表示	306
既知の問題	307

ROM	308
ブロック パラメータ	308
ザイリンクス LogiCORE	309
Sample Time	311
Scale	312
ブロック パラメータ	312
ザイリンクス LogiCORE	312
Serial to Parallel	313
ブロック インターフェイス	313
ブロック パラメータ	313
Shared Memory	314
ブロック インターフェイス	314
ブロック パラメータ	316
ザイリンクス LogiCORE	317
関連項目	317
Shared Memory Read	318
FIFO のトランザクション	318
ロック可能なメモリのトランザクション	318
ブロック パラメータ	319
関連項目	319
Shared Memory Write	320
FIFO のトランザクション	320
ロック可能なメモリのトランザクション	320
ブロック パラメータ	321
関連項目	321
Shift	322
ブロック パラメータ	322
ザイリンクス LogiCORE	322
Simulation Multiplexer	323
シミュレーションにサブシステム、ハードウェアにブラックボックスを使用する場合 ...	323
ブロック パラメータ	324
Single Port RAM	325
ブロック インターフェイス	325
ブロック パラメータ	325
書き込みモード	327
ハードウェアの注意点	328
ザイリンクス LogiCORE	328
Single-Step Simulation	331
ブロック パラメータ	331
Slice	332
ブロック パラメータ	332
System Generator	333
ブロック パラメータ	333
Threshold	339
ブロック パラメータ	339
ザイリンクス LogiCORE	339
Time Division Demultiplexer	340
ブロック インターフェイス	340
ブロック パラメータ	341
Time Division Multiplexer	342
ブロック インターフェイス	342
ブロック パラメータ	342
To FIFO	343
ブロック パラメータ	343
ザイリンクス LogiCORE	344
関連項目	344

To Register	345
ブロック パラメータ	345
ザイリンクス LogiCORE	345
クロック ドメインの切り替え	346
関連項目	346
Toolbar	347
ブロック インターフェイス	347
ツールバー メニュー	348
リファレンス	348
関連項目	349
Up Sample	350
ブロック インターフェイス	350
ブロック パラメータ	351
Viterbi Decoder 6_1	352
ブロック インターフェイス	352
ブロック パラメータ	353
ザイリンクス LogiCORE	354
Viterbi Decoder 7.0	355
ブロック インターフェイス	355
ブロック パラメータ	356
ザイリンクス LogiCORE	359
WaveScope	360
クイック チュートリアル	360
ブロック インターフェイス	363
ザイリンクス LogiCORE バージョン	371

第 2 章：ザイリンクス リファレンス ブロックセット

通信	375
制御ロジック	375
DSP	375
画像	376
演算	376
2 Channel Decimate by 2 MAC FIR Filter	377
ブロック パラメータ	377
参考資料	377
2n+1-tap Linear Phase MAC FIR Filter	378
ブロック パラメータ	378
参考資料	378
2n-tap Linear Phase MAC FIR Filter	379
ブロック パラメータ	379
参考資料	379
2n-tap MAC FIR Filter	380
ブロック パラメータ	380
参考資料	380
4-channel 8-tap Transpose FIR Filter	381
ブロック パラメータ	381
4n-tap MAC FIR Filter	382
ブロック パラメータ	382
参考資料	382
5x5Filter	383
ブロック パラメータ	384
BPSK AWGN Channel	385
ブロック パラメータ	385
参考資料	385
CIC Filter	386
ブロック インターフェイス	386

ブロック パラメータ	387
参考資料	387
Convolutional Encoder	388
インプリメンテーション	388
ブロック インターフェイス	389
ブロック パラメータ	389
CORDIC ATAN	390
ブロック パラメータ	390
参考資料	390
CORDIC DIVIDER	391
ブロック パラメータ	391
参考資料	391
CORDIC LOG	392
ブロック パラメータ	392
参考資料	393
CORDIC SINCOS	394
ブロック パラメータ	394
参考資料	394
CORDIC SQRT	395
ブロック パラメータ	395
参考資料	396
Dual Port Memory Interpolation MAC FIR Filter	397
ブロック パラメータ	397
参考資料	397
Interpolation Filter	398
ブロック パラメータ	398
参考資料	398
m-channel n-tap Transpose FIR Filter	399
ブロック パラメータ	399
Mealy State Machine	400
例	401
ブロック パラメータ	402
Moore State Machine	403
例	404
ブロック パラメータ	405
Multipath Fading Channel Model	406
概念	406
インプリメンテーション	407
ブロック パラメータ	407
関数	408
データ形式	409
入力	410
出力	410
タイミング制約	411
初期化	411
デモンストレーション	411
ハードウェア協調シミュレーションの例	411
参考資料	412
n-tap Dual Port Memory MAC FIR Filter	413
ブロック パラメータ	413
参考資料	413
n-tap MAC FIR Filter	414
ブロック パラメータ	414
参考資料	414
Registered Mealy State Machine	415
例	416

ブロック パラメータ	417
Registered Moore State Machine	418
例	419
ブロック パラメータ	420
Virtex Line Buffer	421
ブロック パラメータ	421
Virtex2 Line Buffer	422
ブロック パラメータ	422
Virtex2 5 Line Buffer	423
ブロック パラメータ	423
White Gaussian Noise Generator	424
4 ビット Leap-Forward LFSR	424
Box-Muller 法	425
ブロック パラメータ	425
参考資料	425

第 3 章：ザイリンクス XtremeDSP キット ブロックセット

XtremeDSP ADC	428
ブロック パラメータ	428
データシート	428
XtremeDSP Co-simulation	429
ブロック パラメータ	429
XtremeDSP DAC	431
ブロック パラメータ	431
データシート	431
XtremeDSP External RAM	432
ブロック パラメータ	432
XtremeDSP LED Flasher	433
ブロック パラメータ	433

第 4 章：System Generator ユーティリティ

xlAddTerms	436
構文	436
説明	436
例	438
メモ	438
関連項目	439
xlCache	440
構文	440
説明	440
関連項目	441
xlConfigureSolver	442
構文	442
説明	442
例	442
xlfa_denominator	443
構文	443
説明	443
関連項目	443
xlfa_numerator	444
構文	444
説明	444
関連項目	444
xlGenerateButton	445
構文	445

説明.....	445
関連項目	445
xlgetParam および xlsetparam.....	446
構文.....	446
説明.....	446
例	447
関連項目	447
xlgetparams	448
構文.....	448
説明.....	448
例	448
関連項目	449
xlGetReloadOrder	450
構文.....	450
説明.....	450
関連項目	451
xlInstallPlugin	452
構文.....	452
説明.....	452
例	452
関連項目	452
xlLoadChipScopeData.....	453
構文	453
説明.....	453
例	453
関連項目	453
xlSBDBuilder	454
構文.....	454
説明.....	454
関連項目	456
xlSetNonMemMap	457
構文.....	457
説明.....	457
例	457
関連項目	457
xlSetUseHDL	458
構文.....	458
説明.....	458
例	458
関連項目	458
xlSwitchLibrary.....	459
構文.....	459
説明.....	459
例	459
xlTBUtills.....	460
構文.....	460
説明.....	460
例	461
メモ.....	463
関連項目	463
xlTimingAnalysis	464
構文.....	464
説明.....	464
例	464
xlUpdateModel	465
構文.....	465

説明	465
例	467
xlVersion	468
構文	468
説明	468
関連項目	468

第 5 章：プログラムを使用したアクセス

プログラム生成用 System Generator API	469
はじめに	469
xBlock	470
xInport	471
xOutput	471
xSignal	472
xlsub2script	472
xBlock ヘルプ	474
PG API の例	475
Hello World	475
MACC	476
マスクされたサブシステム内の MACC	477
PG API エラー / 警告の状況およびメッセージ	481
xBlock エラー メッセージ	481
xInport エラー メッセージ	482
xOutput エラー メッセージ	482
xSignal エラー メッセージ	482
xsub2script エラー メッセージ	482
Shared Memory ブロックへの C++ アクセス	483
ハードウェア協調シミュレーションへの M コード アクセス	483
M コード/ハードウェア協調シミュレーションで使用するハードウェアのコンパイル	484
M コード/ハードウェア協調シミュレーション セマンティクス	484
データ表現	484
M コードからハードウェアへのインターフェイス	485
M コード/ハードウェア協調シミュレーションの例	485
M コード/ハードウェア協調シミュレーション テストベンチの自動生成	490
リソースの管理	493
M コード/ハードウェア協調シミュレーションの MATLAB クラス	493
M コード/ハードウェア協調シミュレーションの Shared Memory MATLAB クラス	499
M コード/ハードウェア協調シミュレーションの Shared FIFO MATLAB クラス	501
M コード/ハードウェア協調シミュレーション ユーティリティ関数	502
SharedMemory	505
Public タイプ	505
Public メソッド	505
Static Public 属性	505
Protected タイプ	505
Protected メソッド	505
Protected 属性	505
メンバー列挙	506
コンストラクタおよびデストラクタ	506
メンバー関数	507
メンバー データ	510
LockableSharedMemory	511
Public タイプ	511
Public メソッド	511
Static Public 属性	511
メンバー Typedefs	511
コンストラクタおよびデストラクタ	511
メンバー関数	512

メンバー データ	513
SharedMemoryProxy	515
Public タイプ	515
Public メソッド	515
Static Public 属性	515
メンバー Typedefs	515
コンストラクタおよびデストラクタ	515
メンバー関数	516
メンバー データ	517
Request Struct	518
Public タイプ	518
Static Public 属性	518
メンバー列挙	518
メンバー データ	518
NamedPipeReader	519
Public メソッド	519
Static Public 属性	519
コンストラクタおよびデストラクタ	519
メンバー関数	520
メンバー データ	521
NamedPipeWriter	522
Public メソッド	522
Static Public 属性	522
コンストラクタおよびデストラクタ	522
メンバー関数	523
メンバー データ	524
索引	525

このマニュアルについて

このマニュアルには、System Generator で使用されているブロックの詳細な情報が記載されています。また、System Generator のユーティリティおよびプログラムのアクセスについても説明されています。

マニュアルの内容

このマニュアルには、次の内容が含まれています。

- ザイリンクス ブロックセット
- ザイリンクス リファレンス ブロックセット
- XtremeDSP キット
- System Generator ユーティリティ
- プログラム アクセス

System Generator の PDF マニュアル セット

このマニュアルは System Generator の Help システムから参照でき、また System Generator の PDF マニュアル セット の一部です。この PDF マニュアル セット には、次のマニュアルが含まれています。

- 『System Generator for DSP 入門ガイド』
- 『System Generator for DSP ユーザー ガイド』
- 『System Generator for DSP リファレンス ガイド』

メモ：これらのマニュアル間のハイパーリンクは、PDF ファイルが同じフォルダにある場合にのみ機能します。Adobe Reader でハイパーリンクをクリックした場合、Alt キーと左方向キー (←) を同時に押すと、前に参照していたページに戻ることができます。

その他のリソース

その他の資料は、次のザイリンクス Web サイトを参照してください。

<http://japan.xilinx.com/literature>

シリコン、ソフトウェア、IP に関する質問および解答をアンサー データベースで検索したり、テクニカル サポートのウェブ ケースを開くには、次のザイリンクス Web サイトにアクセスしてください。

<http://japan.xilinx.com/support>

表記規則

このマニュアルでは、次の表記規則を使用しています。各規則について、例を挙げて説明します。

書体

次の規則は、すべてのマニュアルで使用されています。

表記規則	使用箇所	例
Courier フォント	システムが表示するメッセージ、プロンプト、プログラム ファイルを表示します。	speed grade: - 100
Courier フォント (太字)	構文内で入力するコマンドを示します。	ngdbuild design_name
イタリック フォント	ユーザーが値を入力する必要がある構文内の変数に使用します。	<i>ngdbuild</i> design_name
二重/一重かぎカッコ『』、『』	『』はマニュアル名を、『』はセクション名を示します。	詳細については、『開発システムリファレンス ガイド』の「PAR」を参照してください。
角カッコ []	オプションの入力またはパラメータを示しますが、 bus[7:0] のようなバス仕様では必ず使用します。また、GUI 表記にも使用します。	ngdbuild [option_name] design_name [File] → [Open] をクリックします。
中カッコ { }	1 つ以上の項目を選択するためのリストを示します。	lowpwr = {on off}
縦棒	選択するリストの項目を分離します。	lowpwr = {on off}
縦の省略記号 . . .	繰り返し項目が省略されていることを示します。	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
横の省略記号 ...	繰り返し項目が省略されていることを示します。	allow block block_name loc1 loc2 ... locn;

オンライン マニュアル

このマニュアルでは、次の規則が使用されています。

表記規則	使用箇所	例
青色の文字	マニュアル内の相互参照を示します。	詳細については、「 その他のリソース 」を参照してください。 詳細については、第 1 章「 タイトルフォーマット 」を参照してください。
赤色の文字	ほかのマニュアルへの相互参照を示します。	詳細については、『Virtex-II Platform FPGA ユーザーガイド』の 図 2-5 を参照してください。
青色の下線付き文字	Web サイト (URL) へのハイパーリンクです。	最新のスピード ファイルは、 http://japan.xilinx.com から入手できます。

ザイリンクス ブロックセット

ブロックセット ライブラリの構造	ザイリンクス ブロックがライブラリにどのように分類されているかを示します。
System Generator でサポートされる Simulink ブロック	ザイリンクス ブロックセットのほとんどのブロックに共通するブロック パラメータについて説明します。
ブロックのリファレンス ページ	ザイリンクス ブロックセットをアルファベット順に並べ、各ブロックの詳細を説明します。
ザイリンクス LogiCORE バージョン	ザイリンクス ブロックセットで使用するザイリンクス LogiCORE™ のバージョン番号をリストします。

ブロックセット ライブラリの構造

[Xilinx Blockset] には、Simulink を使用して FPGA に DSP およびその他のデジタル システムを構築するためのブロックが含まれます。ブロックは、ファンクション別にライブラリに分類されています。Gateway I/O ブロックなど幅広く応用可能なブロックは、複数のライブラリに含まれています。含まれるライブラリは次のとおりです。

ライブラリ	説明
Index ブロック	ザイリンクス ブロックセットのすべてのブロック
Basic Elements ブロック	デジタル ロジックの基本ブロック
Communication ブロック	デジタル通信システムでよく使用される順方向誤り訂正ブロックおよびモジュレータ ブロック
Control Logic ブロック	制御回路およびステート マシン用のブロック
Data Types ブロック	データ型を変換するブロック (Gateway ブロック を含む)
DSP ブロック	DSP (デジタル信号処理) ブロック
Math ブロック	演算ファンクションをインプリメントするブロック
Memory ブロック	メモリをインプリメントするブロックおよびメモリにアクセスするためのブロック
Shared Memory ブロック	ザイリンクス共有メモリをインプリメントするブロックおよび共有メモリにアクセスするためのブロック
Tools ブロック	コード生成 (System Generator ブロック)、リソース予測、HDL 協調シミュレーションなどを実行するユーティリティ ブロック

Basic Elements ブロック

表 1-1 : Basic Elements ブロック

ブロック	説明
Addressable Shift Register	ザイリンクスの Addressable Shift Register ブロックは、可変長シフトレジスタで、遅延チェーンのどのレジスタでも呼び出して、出力データポートに駆動できます。
Assert	Assert ブロックは、信号にレートやデータ型をアサートするために使用します。このブロックには、ハードウェア コストがかかりません。このブロックを使用すると、設計者が手を入れる必要のある場合に、レートやデータ型の問題を改善できます。
BitBasher	ザイリンクスの BitBasher ブロックでは、ブロックに接続された入力値がスライスされた後、連結されて追加されます。
Black Box	System Generator の Black Box ブロックでは、HDL モデルを System Generator 用に変換できます。
Clock Enable Probe	ザイリンクスの Clock Enable Probe ブロックでは、 System Generator モデルのザイリンクス信号から派生したクロック イネーブル信号を抽出できます。

表 1-1 : Basic Elements ブロック

ブロック	説明
Concat	ザイリンクスの Concat ブロックでは、符号なしの整数で表される n ビット ベクタ (2 進小数点の位置が 0 にある、 n 個の符号なしの値) が連結されます。
Constant	ザイリンクスの Constant ブロックでは、固定小数点値、ブール値、 DSP48 命令のいずれかの定数が生成されます。このブロックは、 Simulink の Constant ブロックと類似していますが、ザイリンクスブロックの入力を直接駆動できる点が異なります。
Convert	ザイリンクスの Convert ブロックでは、各入力サンプルが指定した演算タイプの値に変換されます。たとえば、ある数値を符号付き (2 の補数) または符号なしの値にできます。
Counter	ザイリンクスの Counter ブロックは、フリー ランニング カウンタ、またはアップ カウンタ、ダウン カウンタ、アップ/ダウン カウンタなどのカウント制限のあるカウンタをインプリメントします。カウンタの出力は、符号付きまたは符号なしの固定小数点の値で指定できます。
Divider Generator 2.0	ザイリンクスの Divider Generator 2.0 ブロックは、 Radix-2 の非回復型除算、またはプリスケールリングを用いた高基数除算に基づいて整数除算を行う回路を作成します。
Expression	ザイリンクスの Expression ブロックは、ビット単位の論理式を実行します。
Gateway In	ザイリンクスの Gateway In ブロックは、 Simulink デザインのザイリンクス部分への入力に使用します。このブロックは、 Simulink の整数、ダブル データ、固定小数点などのデータ型を System Generator の固定小数点型に変換します。各ブロックでは、 System Generator で生成された HDL デザインの最上位入力ポートを定義します。
Gateway Out	ザイリンクスの Gateway Out ブロックは、 Simulink デザインのザイリンクス部からの出力に使用します。このブロックでは、 System Generator の固定小数点型のデータが Simulink のダブル データ型に変換されます。
LFSR	ザイリンクスの LFSR ブロックは、 LFSR (Linear Feedback Shift Register) をインプリメントします。このブロックでは、 XOR または XNOR のいずれかを使用してガロアおよびフィボナッチ構造の両方がサポートされるほか、リロード可能な入力を使用して、レジスタの値をいつでも変更できます。 LFSR の出力とリロード可能な入力は、シリアル ポートまたはパラレル ポートのいずれかでコンフィギュレーションできます。
Logical	ザイリンクスの Logical ブロックでは、2、3、または 4 つの固定小数点の値のビット単位の論理演算が実行されます。オペランドには、2 進小数点の位置を揃えるため、必要に応じて 0 がパディングされたり、符号が拡張されたりします。これらの値で論理演算が実行され、結果が出力ポートに送信されます。

表 1-1 : Basic Elements ブロック

ブロック	説明
Mux	ザイリンクスの Mux ブロックは、マルチプレクサをインプリメントします。ブロックには、セレクト入力 (符号なし) が 1 つと、ユーザーがコンフィギュレーション可能なデータ バス入力 (0 ~ 1024 の範囲) がいくつか含まれます。
Parallel to Serial	Parallel to Serial ブロックは、入力ワードを読み込み、それを分割してマルチプレクサで N 倍多重化した出力ワードにします。N は、入力ビットから出力ビットの数の比率です。出力順は、 LSB からか MSB からかのどちらかになります。
Register	ザイリンクスの Register ブロックは、1 サンプル周期のレイテンシを含む D フリップフロップ ベースのレジスタになります。
Reinterpret	ザイリンクスの Reinterpret ブロックは、入力値に関係なく、出力を強制的に新しいデータ型にします。
Relational	ザイリンクスの Relational ブロックは、コンパレータをインプリメントします。
Reset Generator	Reset Generator ブロックは、システム サンプル レートで実行されるユーザー リセット信号をキャプチャし、ブロックで指定されたレートで実行される 1 つまたは複数のダウンサンプルされたリセット信号を生成します。
Serial to Parallel	Serial to Parallel ブロックでは、連続して入力されたデータが指定した倍数のサイズの出力 1 つにまとめられて、出力されます。この連続入力は、最上位ワードまたは最下位ワードのいずれかを先頭にした順序になります。
Slice	ザイリンクスの Slice ブロックを使用すると、入力データからビットのシーケンスを切り取り、新しいデータ値を作成できます。このデータ値がブロックから出力されます。出力データ型は、2 進小数点が 0 の位置の符合なしになります。
System Generator	System Generator ブロックは、システム制御およびシミュレーション パラメータを提供し、コード ジェネレータを起動するために使用されます。 System Generator ブロックは、デザインにおける特異な役割りのため、 System Generator トークンと呼ばれることもあります。ザイリンクス ブロックセットからのエレメントを含む Simulink モデルには、 System Generator ブロック (トークン) が最低 1 つは含まれます。 System Generator ブロックをモデルに追加すると、コードの生成およびシミュレーションの処理方法を指定できるようになります。
Time Division Demultiplexer	ザイリンクスの Time Division Demultiplexer ブロックは、シリアル入力をそれより遅いレートの複数の出力にします。

表 1-1 : Basic Elements ブロック

ブロック	説明
Time Division Multiplexer	ザイリンクスの Time Division Multiplexer ブロックは、入力ポートの値を 1 つの高速レートの出力ストリームに多重化します。
Up Sample	ザイリンクスの Up Sample ブロックは、ブロックが配置された箇所のサンプル レートを増加するために使用します。出力サンプル周期は、 $1/n$ になります (1 が入力サンプル周期、 n はサンプリングレート)。

Communication ブロック

表 1-2 : Communication ブロック - FEC

Communication ブロック	説明
Convolution Encoder 7.0	ザイリンクスの Convolution Encoder ブロックは、たたみ込み符号のエンコーダをインプリメントします。通常 Viterbi デコーダと併用され、デジタル通信システムで順方向誤り訂正 (FEC) を実行します。
Depuncture	ザイリンクスの Depuncture ブロックを使用すると、デパンクチャ符号で指定した位置の入力データにシンボルを挿入できます。
Interleaver Deinterleaver 5.0	ザイリンクスの Interleaver/Deinterleaver ブロックは、インターリーバまたはデインターリーバをインプリメントします。インターリーバは、1:1 の確定的な方法でシンボルのシーケンス順序を並び替えるデバイスです。デインターリーバは、この並び替えられたシーケンスを再び格納するデバイスです。
Interleaver Deinterleaver 5.1	ザイリンクスの Interleaver/Deinterleaver ブロックは、インターリーバまたはデインターリーバをインプリメントします。インターリーバは、1:1 の確定的な方法でシンボルのシーケンス順序を並び替えるデバイスです。デインターリーバは、この並び替えられたシーケンスを再び格納するデバイスです。
Puncture	ザイリンクスの Puncture ブロックでは、データストリームの入力ワードからユーザーが指定したビットのセットが削除されます。
Reed-Solomon Decoder 6.1	Reed-Solomon (RS) コードは、ブロック ベースの誤り訂正符号 (ECC) で、デジタル通信およびストレージ分野のさまざまなアプリケーションに使用されています。
Reed-Solomon Decoder 7.0	Reed-Solomon (RS) コードは、ブロック ベースの誤り訂正符号 (ECC) で、デジタル通信およびストレージ分野のさまざまなアプリケーションに使用されています。

表 1-2 : Communication ブロック - FEC

Communication ブロック	説明
Reed-Solomon Encoder 6.1	Reed-Solomon (RS) コードは、ブロック ベースの誤り訂正符号 (ECC) で、デジタル通信およびストレージ分野のさまざまなアプリケーションに使用されています。
Reed-Solomon Encoder 7.0	Reed-Solomon (RS) コードは、ブロック ベースの誤り訂正符号 (ECC) で、デジタル通信およびストレージ分野のさまざまなアプリケーションに使用されています。
Viterbi Decoder 7.0	たたみ込みエンコーダでエンコードされたデータは、ザイリンクスの Viterbi Decoder ブロックを使用してデコードできます。

Control Logic ブロック

表 1-3 : Control Logic ブロック

Control Logic ブロック	説明
Black Box	System Generator の Black Box ブロックでは、HDL モデルを System Generator 用に変換できます。
Constant	ザイリンクスの Constant ブロックでは、固定小数点値、ブール値、DSP48 命令のいずれかの定数が生成されます。このブロックは、Simulink の Constant ブロックと類似していますが、ザイリンクス ブロックの入力を直接駆動できる点が異なります。
Counter	ザイリンクスの Counter ブロックは、フリー ランニング カウンタ、またはアップ カウンタ、ダウン カウンタ、アップ/ダウン カウンタなどのカウント制限のあるカウンタをインプリメントします。カウンタの出力は、符号付きまたは符号なしの固定小数点の値で指定できます。
Dual Port RAM	ザイリンクスの Dual Port RAM ブロックは、ランダム アクセス メモリ (RAM) をインプリメントします。デュアルポートを使用すると、複数のデータ幅を使用して異なるサンプル レートでメモリ空間に同時にアクセスできます。
EDK Processor	EDK Processor ブロックは、System Generator で開発したユーザー ロジックをザイリンクスの EDK (エンベデッド開発システム) を使用して作成したエンベデッド プロセッサ システムに接続するために使用します。
Expression	ザイリンクスの Expression ブロックは、ビット単位の論理式を実行します。
FIFO	ザイリンクスの FIFO ブロックは、FIFO メモリ キューをインプリメントします。
Inverter	ザイリンクスの Inverter ブロックでは、固定小数点型の論理コンポーネントがビット単位で計算されます。このブロックは、合成可能な VHDL モジュールとしてインプリメントされます。

表 1-3 : Control Logic ブロック

Control Logic ブロック	説明
Logical	ザイリンクスの Logical ブロックでは、2、3、または 4 つの固定小数点の値のビット単位の論理演算が実行されます。オペランドには、2 進小数点の位置を揃えるため、必要に応じて 0 がパディングされたり、符号が拡張されたりします。これらの値で論理演算が実行され、結果が出力ポートに送信されます。
MCode	ザイリンクスの MCode ブロックは、 Simulink でユーザーの提供する MATLAB 関数を実行するためのコンテナです。M 関数の名前はブロックのパラメータから指定します。このブロックは、M コードを実行して Simulink のシミュレーション中にブロックの出力を計算します。同じコードは、ハードウェアが生成されるときに、同等のビヘイビアレベルの VHDL/Verilog に直接変換されます。
Mux	ザイリンクスの Mux ブロックは、マルチプレクサをインプリメントします。ブロックには、セレクト入力 (符号なし) が 1 つと、ユーザーがコンフィギュレーション可能なデータ バス入力 (0 ~ 1024 の範囲) がいくつか含まれます。
PicoBlaze Microcontroller	PicoBlaze Microcontroller ブロックは、 PicoBlaze マクロを使用して 8 ビットのエンベデッド マイクロコントローラをインプリメントします。
Register	ザイリンクスの Register ブロックは、1 サンプル周期のレイテンシを含む D フリップフロップ ベースのレジスタになります。
Relational	ザイリンクスの Relational ブロックは、コンパレータをインプリメントします。
ROM	ザイリンクスの ROM ブロックは、シングル ポートの読み出し専用メモリ (ROM) です。
Shift	ザイリンクスの Shift ブロックは、入力信号を左または右にシフトします。出力には、入力と同じ固定小数点のコンテナが含まれます。
Single Port RAM	ザイリンクスの Single Port RAM ブロックは、データ入力ポートとデータ出力ポートが 1 つずつ付いたランダム アクセス メモリ (RAM) をインプリメントします。
Slice	ザイリンクスの Slice ブロックを使用すると、入力データからビットのシーケンスを切り取り、新しいデータ値を作成できます。このデータ値がブロックから出力されます。出力データ型は、2 進小数点が 0 の位置の符合なしになります。

Data Types ブロック

表 1-4 : Data Types ブロック

Data Types ブロック	説明
Concat	ザイリンクスの Concat ブロックでは、符号なしの整数で表される n ビット ベクタ (2 進小数点の位置が 0 にある、 n 個の符号なしの値) が連結されます。
Convert	ザイリンクスの Convert ブロックでは、各入力サンプルが指定した演算タイプの値に変換されます。たとえば、ある数値を符号付き (2 の補数) または符号なしの値にできます。
Gateway In	ザイリンクスの Gateway In ブロックは、 Simulink デザインのザイリンクス部分への入力に使用します。このブロックは、 Simulink の整数、ダブルデータ、固定小数点などのデータ型を System Generator の固定小数点型に変換します。各ブロックでは、 System Generator で生成された HDL デザインの最上位入力ポートを定義します。
Gateway Out	ザイリンクスの Gateway Out ブロックは、 Simulink デザインのザイリンクス部からの出力に使用します。このブロックでは、 System Generator の固定小数点型のデータが Simulink のダブル データ型に変換されます。
Parallel to Serial	Parallel to Serial ブロックは、入力ワードを読み込み、それを分割してマルチプレクサで N 倍多重化した出力ワードにします。 N は、入力ビットから出力ビットの数の比率です。出力順は、 LSB からか MSB からのどちらかになります。
Reinterpret	ザイリンクスの Reinterpret ブロックは、入力値に関係なく、出力を強制的に新しいデータ型にします。
Scale	ザイリンクスの Scale ブロックでは、入力が 2 のべき乗でスケール変換されます。べき乗の値は、正にでも負にでもできます。このブロックには、入力が 1 つと出力が 1 つあります。スケール変換では、コンテナ内のビットを変更せずに 2 進小数点の位置が移動されます。
Serial to Parallel	Serial to Parallel ブロックでは、連続して入力されたデータが指定した倍数のサイズの出力 1 つにまとめられて、出力されます。この連続入力は、最上位ワードまたは最下位ワードのいずれかを先頭にした順序になります。
Shift	ザイリンクスの Shift ブロックは、入力信号を左または右にシフトします。出力には、入力と同じ固定小数点のコンテナが含まれます。
Slice	ザイリンクスの Slice ブロックを使用すると、入力データからビットのシーケンスを切り取り、新しいデータ値を作成できます。このデータ値がブロックから出力されます。出力データ型は、2 進小数点が 0 の位置の符号なしになります。

DSP ブロック

表 1-5 : DSP ブロック

DSP ブロック	説明
CIC Compiler 1.2	ザイリンクスの CIC Compiler には、さまざまなザイリンクス FPGA デバイスに対して CIC (Cascaded Integrator-Comb) フィルタをデザインおよびインプリメントする機能があります。
Complex Multiplier 3.0	ザイリンクスの Complex Multiplier ブロックは、2 つの複素数を乗算します。
Complex Multiplier 3.1	ザイリンクスの Complex Multiplier ブロックは、2 つの複素数を乗算します。
CORDIC 4.0	ザイリンクスの CORDIC 4.0 ブロックは、CORDIC (Coordinate Rotational Digital Computer) アルゴリズムをインプリメントします。
DAFIR v9_0	ザイリンクスの DAFIR ブロックは、分散演算タイプの FIR (Finite Impulse Response) デジタル フィルタ、または同一の FIR フィルタ (マルチチャネル モード) のバンクをインプリメントします。
DDS Compiler 2.1	ザイリンクスの DDS Compiler v2_1 ブロックは、ダイレクト デジタル シンセサイザで、NCO (Numerically Controlled Oscillator) と呼ばれます。このブロックでは、ルックアップ テーブルを使用してサイン波が生成されます。ルックアップ テーブルでマップされた位相は、デジタル積分器 (アキュムレータ) で出力正弦波形にされます。
DDS Compiler 4.0	ザイリンクス DDS Compiler 4.0 ブロックの機能は DDS Compiler 3.0 と同じですが、次の新機能のサポートされている点が異なります。
Divider Generator 2.0	ザイリンクスの Divider Generator 2.0 ブロックは、Radix-2 の非回復型除算、またはプリスケールリングを用いた高基数除算に基づいて整数除算を行う回路を作成します。
Divider Generator 3.0	ザイリンクスの Divider Generator 3.0 ブロックは、Radix-2 の非回復型除算、またはプリスケールリングを用いた高基数除算に基づいて整数除算を行う回路を作成します。
DSP48	ザイリンクスの DSP48 ブロックは、ザイリンクスの Virtex®-4 デバイスを使用する DSP アプリケーションを効率的に構築するためのブロックです。DSP48 では、18X18 ビットの符号付き乗算器と 48 ビット加算器が組み合わされており、加算器の入力はプログラマブル マルチプレクサで選択されます。
DSP48 Macro 2.0	DSP48 Macro 2.0 ブロックは、DSP48、DSP48A、および DSP48E ブロックをデバイスに依存せずに抽象化したものです。デバイス別に DSP スライスを使用する代わりに、このブロックを使用すると、ザイリンクス デバイス間でデザインを動かしやすくなります。

表 1-5：DSP ブロック

DSP ブロック	説明
DSP48A	ザイリンクスの DSP48A ブロックは、ザイリンクスの Spartan-3A DSP デバイスを使用する DSP アプリケーションを効率的に構築するためのブロックです。DSP48A は、DSP48 および DSP48E の軽量版です。
DSP48E	ザイリンクスの DSP48E ブロックは、ザイリンクスの Virtex-5 デバイスを使用する DSP アプリケーションを効率的に構築するためのブロックです。DSP48E では、18X25 ビットの符号付き乗算器と 48 ビット加算器が組み合わされており、加算器の入力はプログラマブルマルチプレクサで選択されます。
Fast Fourier Transform 6.0	ザイリンクス Fast Fourier Transform 6.0 ブロックは、離散フーリエ変換 (DFT) を計算する効率的なアルゴリズムをインプリメントします。
Fast Fourier Transform 7.0	ザイリンクス Fast Fourier Transform 7.0 ブロックは、離散フーリエ変換 (DFT) を計算する効率的なアルゴリズムをインプリメントします。
FDATool	ザイリンクスの FDATool ブロックは、MATLAB の Signal Processing Toolbox の一部である FDATool ソフトウェアへのインターフェイスになります。
FIFO	ザイリンクスの FIFO ブロックは、FIFO メモリ キューをインプリメントします。
FIR Compiler 4.0	ザイリンクス FIR Compiler 4.0 ブロックは、高速 MAC ベースの FIR フィルタをインプリメントします。このブロックは、入力データのストリームを受信し、フィルタのコンフィギュレーションに応じてフィルタ処理した結果を、固定の遅延で出力します。
FIR Compiler 5.0	ザイリンクスの FIR Compiler 5.0 ブロックは、積和演算 (MAC) ベースまたは分散演算の FIR フィルタをインプリメントします。このブロックは、入力データのストリームを受信し、フィルタのコンフィギュレーションに応じてフィルタ処理した結果を、固定の遅延で出力します。MAC ベースのフィルタは、次の図のように、カスケード接続された Xtreme DSP スライスを使用してインプリメントされます。
LFSR	ザイリンクスの LFSR ブロックは、LFSR (Linear Feedback Shift Register) をインプリメントします。このブロックでは、XOR または XNOR のいずれかを使用してガロアおよびフィボナッチ構造の両方がサポートされるほか、リロード可能な入力を使用して、レジスタの値をいつでも変更できます。LFSR の出力とリロード可能な入力は、シリアルポートまたはパラレルポートのいずれかでコンフィギュレーションできます。
Opmode	ザイリンクスの Opmode ブロックでは、DSP48 または DSP48E 命令の定数が生成されます。この命令は、DSP48 の場合が 11 ビット値、DSP48E の場合が 15 ビット値になります。命令には、opmode、carry-in、carry-in select が含まれるほか、DSP48 か DSP48E かによって subtract または alumode ビットのいずれかが含まれます。

Index ブロック

表 1-6 : Index ブロック

Index ブロック	説明
Accumulator	ザイリンクスの Accumulator ブロックは、加算器または減算器ベースのスケールリング アキュムレータです。
Addressable Shift Register	ザイリンクスの Addressable Shift Register ブロックは、可変長シフトレジスタで、遅延チェーンのどのレジスタでも呼び出して、出力データポートに駆動できます。
AddSub	ザイリンクスの AddSub ブロックは、加算器/減算器です。処理方法は、加算か減算のどちらかに指定することもできますが、sub 信号の制御に従ってダイナミックに変更することもできます。
Assert	Assert ブロックは、信号にレートやデータ型をアサートするために使用します。このブロックには、ハードウェア コストがかかりません。このブロックを使用すると、設計者が手を入れる必要のある場合に、レートやデータ型の問題を改善できます。
BitBasher	ザイリンクスの BitBasher ブロックでは、ブロックに接続された入力の値がスライスされた後、連結されて追加されます。
Black Box	System Generator の Black Box ブロックでは、HDL モデルを System Generator 用に変換できます。
ChipScope	ザイリンクス ChipScop™ ブロックでは、ランタイム デバッグおよび FPGA 内の信号検証などを実行できます。
CIC Compiler 1.2	ザイリンクスの CIC Compiler には、さまざまなザイリンクス FPGA デバイスに対して CIC (Cascaded Integrator-Comb) フィルタをデザインおよびインプリメントする機能があります。
Clock Enable Probe	ザイリンクスの Clock Enable Probe ブロックでは、System Generator モデルのザイリンクス信号から派生したクロック イネーブル信号を抽出できます。
Clock Probe	ザイリンクスの Clock Probe ブロックでは、倍精度のクロック信号が Simulink システム周期と同じ周期で出力されます。
CMult	ザイリンクスの CMult ブロックは、gain 演算子をインプリメントします。出力は、入力と定数値の積になります。この値は、定数を求める MATLAB の論理式で表すことができます。
Complex Multiplier 3.0	ザイリンクスの Complex Multiplier ブロックは、2 つの複素数を乗算します。
Complex Multiplier 3.1	ザイリンクスの Complex Multiplier ブロックは、2 つの複素数を乗算します。
Concat	ザイリンクスの Concat ブロックでは、符号なしの整数で表される n ビット ベクタ (2 進小数点の位置が 0 にある、n 個の符号なしの値) が連結されます。

表 1-6 : Index ブロック

Index ブロック	説明
Configurable Subsystem Manager	ザイリンクスの Configurable Subsystem Manager ブロックは、 Simulink のコンフィギャブル システムの機能を拡張したブロックで、シミュレーションと同様、ハードウェアの生成にもサブシステム コンフィギュレーションが選択できるようになります。
Constant	ザイリンクスの Constant ブロックでは、固定小数点値、ブール値、 DSP48 命令のいずれかの定数が生成されます。このブロックは、 Simulink の Constant ブロックと類似していますが、ザイリンクス ブロックの入力を直接駆動できる点が異なります。
Convert	ザイリンクスの Convert ブロックでは、各入力サンプルが指定した演算タイプの値に変換されます。たとえば、ある数値を符号付き (2 の補数) または符号なしの値にできます。
Convolution Encoder 6.1	ザイリンクスの Convolution Encoder ブロックは、たたみ込み符号のエンコーダをインプリメントします。通常 Viterbi デコーダと併用され、デジタル通信システムで順方向誤り訂正 (FEC) を実行します。
Convolution Encoder 7.0	ザイリンクスの Convolution Encoder ブロックは、たたみ込み符号のエンコーダをインプリメントします。通常 Viterbi デコーダと併用され、デジタル通信システムで順方向誤り訂正 (FEC) を実行します。
CORDIC 4.0	CORDIC コアは、次の論理式タイプをインプリメントします。
Counter	ザイリンクスの Counter ブロックは、フリー ランニング カウンタ、またはアップ カウンタ、ダウン カウンタ、アップ/ダウン カウンタなどのカウント制限のあるカウンタをインプリメントします。カウンタの出力は、符号付きまたは符号なしの固定小数点の値で指定できます。
DAFIR v9_0	ザイリンクスの DAFIR ブロックは、分散演算タイプの FIR (Finite Impulse Response) デジタル フィルタ、または同一の FIR フィルタ (マルチチャネル モード) のバンクをインプリメントします。
DDS Compiler 2.1	ザイリンクスの DDS Compiler v2_1 ブロックは、ダイレクト デジタル シンセサイザで、 NCO (Numerically Controlled Oscillator) と呼ばれます。このブロックでは、ルックアップ テーブルを使用してサイン波が生成されます。ルックアップ テーブルでマップされた位相は、デジタル積分器 (アキュムレータ) で出力正弦波形にされます。
DDS Compiler 4.0	ザイリンクス DDS Compiler 4.0 ブロックの機能は DDS Compiler 3.0 と同じですが、次の新機能のサポートされている点が異なります。
Delay	Delay ブロックは、 L サイクルの固定遅延をインプリメントします。
Depuncture	ザイリンクスの Depuncture ブロックを使用すると、デパンクチャ符号で指定した位置の入力データにシンボルを挿入できます。
Divider Generator 2.0	ザイリンクスの Divider Generator 2.0 ブロックは、 Radix-2 の非回復型除算、またはプリスケールリングを用いた高基数除算に基づいて整数除算を行う回路を作成します。

表 1-6 : Index ブロック

Index ブロック	説明
Divider Generator 3.0	ザイリンクスの Divider Generator 3.0 ブロックは、Radix-2 の非回復型除算、またはプリスケールを用いた高基数除算に基づいて整数除算を行う回路を作成します。
Down Sample	ザイリンクスの Down Sample ブロックは、ブロックが配置された箇所のサンプル レートを削減するために使用します。
DSP48	ザイリンクスの DSP48 ブロックは、ザイリンクスの Virtex®-4 デバイスを使用する DSP アプリケーションを効率的に構築するためのブロックです。DSP48 では、18X18 ビットの符号付き乗算器と 48 ビット加算器が組み合わされており、加算器の入力はプログラマブル マルチプレクサで選択されます。
DSP48 Macro	DSP48 Macro ブロックは、DSP48、DSP48A、および DSP48E ブロックをデバイスに依存せずに抽象化したものです。デバイス別に DSP スライスを使用する代わりに、このブロックを使用すると、ザイリンクスデバイス間でデザインを動かしやすくなります。
DSP48 Macro 2.0	DSP48 Macro 2.0 ブロックは、DSP48、DSP48A、および DSP48E ブロックをデバイスに依存せずに抽象化したものです。 デバイス別に DSP スライスを使用する代わりに、このブロックを使用すると、ザイリンクス デバイス間でデザインを動かしやすくなります。
DSP48A	ザイリンクスの DSP48A ブロックは、ザイリンクスの Spartan-3A DSP デバイスを使用する DSP アプリケーションを効率的に構築するためのブロックです。DSP48A は、DSP48 および DSP48E の軽量版です。
DSP48E	ザイリンクスの DSP48E ブロックは、ザイリンクスの Virtex-5 デバイスを使用する DSP アプリケーションを効率的に構築するためのブロックです。DSP48E では、18X25 ビットの符号付き乗算器と 48 ビット加算器が組み合わされており、加算器の入力はプログラマブル マルチプレクサで選択されます。
Dual Port RAM	ザイリンクスの Dual Port RAM ブロックは、ランダム アクセス メモリ (RAM) をインプリメントします。デュアルポートを使用すると、複数のデータ幅を使用して異なるサンプル レートでメモリ空間に同時にアクセスできます。
EDK Processor	EDK Processor ブロックは、System Generator で開発したユーザー ロジックをザイリンクスの EDK (エンベデッド開発システム) を使用して作成したエンベデッド プロセッサ システムに接続するために使用します。
Expression	ザイリンクスの Expression ブロックは、ビット単位の論理式を実行します。
Fast Fourier Transform 6.0	ザイリンクス Fast Fourier Transform 6.0 ブロックは、離散フーリエ変換 (DFT) を計算する効率的なアルゴリズムをインプリメントします。
Fast Fourier Transform 7.0	ザイリンクス Fast Fourier Transform 7.0 ブロックは、離散フーリエ変換 (DFT) を計算する効率的なアルゴリズムをインプリメントします。

表 1-6 : Index ブロック

Index ブロック	説明
FDATool	ザイリンクスの FDATool ブロックは、 MATLAB の Signal Processing Toolbox の一部である FDATool ソフトウェアへのインターフェイスになります。
FIFO	ザイリンクスの FIFO ブロックは、 FIFO メモリ キューをインプリメントします。
FIR Compiler 4.0	ザイリンクスの FIFO ブロックは、 FIFO メモリ キューをインプリメントします。
FIR Compiler 5.0	ザイリンクスの FIR Compiler 5.0 ブロックは、積和演算 (MAC) ベースまたは分散演算の FIR フィルタをインプリメントします。このブロックは、入力データのストリームを受信し、フィルタのコンフィギュレーションに応じてフィルタ処理した結果を、固定の遅延で出力します。 MAC ベースのフィルタは、次の図のように、カスケード接続された Xtreme DSP スライスを使用してインプリメントされます。
From FIFO	ザイリンクス From FIFO ブロックは、 First-In First-Out (FIFO) のメモリ キューの後半分をインプリメントします。
From Register	ザイリンクス From Register ブロックは、 D フリップフロップ ベースのレジスタの後半分をインプリメントします。物理的なレジスタは、2 つのデザインまたは同じデザインの 2 箇所でも共有できます。
Gateway In	ザイリンクスの Gateway In ブロックは、 Simulink デザインのザイリンクス部分への入力に使用します。このブロックは、 Simulink の整数、ダブル データ、固定小数点などのデータ型を System Generator の固定小数点型に変換します。各ブロックでは、 System Generator で生成された HDL デザインの最上位入力ポートを定義します。
Gateway Out	ザイリンクスの Gateway Out ブロックは、 Simulink デザインのザイリンクス部からの出力に使用します。このブロックでは、 System Generator の固定小数点型のデータが Simulink のダブル データ型に変換されます。
Indeterminate Probe	ザイリンクスの Indeterminate Probe ブロックの出力は、入力データが不定値 (MATLAB 値が NaN) かどうかを示します。不定データは、 VHDL の不定データ値を表す X に相当します。
Interleaver Deinterleaver 5.0	ザイリンクスの Interleaver/Deinterleaver ブロックは、インターリーブまたはデインターリーブをインプリメントします。インターリーブは、1:1 の確定的な方法でシンボルのシーケンス順序を並び替えるデバイスです。デインターリーブは、この並び替えられたシーケンスを再び格納するデバイスです。
Interleaver Deinterleaver 5.1	ザイリンクスの Interleaver/Deinterleaver ブロックは、インターリーブまたはデインターリーブをインプリメントします。インターリーブは、1:1 の確定的な方法でシンボルのシーケンス順序を並び替えるデバイスです。デインターリーブは、この並び替えられたシーケンスを再び格納するデバイスです。

表 1-6 : Index ブロック

Index ブロック	説明
Inverter	ザイリンクスの Inverter ブロックでは、固定小数点型の論理コンポーネントがビット単位で計算されます。このブロックは、合成可能な VHDL モジュールとしてインプリメントされます。
JTAG Co-Simulation	JTAG Co-Simulation ブロックを使用すると、 JTAG とパラレル ケーブル IV またはプラットフォーム USB を使用したハードウェア協調シミュレーションを実行できます。このブロックのインターフェイスは、広く普及している JTAG を活用して System Generator の Hardware in the Loop シミュレーションをほかのさまざまな FPGA プラットフォームに拡張しています。
LFSR	ザイリンクスの LFSR ブロックは、 LFSR (Linear Feedback Shift Register) をインプリメントします。このブロックでは、 XOR または XNOR のいずれかを使用してガロアおよびフィボナッチ構造の両方がサポートされるほか、リロード可能な入力を使用して、レジスタの値をいつでも変更できます。 LFSR の出力とリロード可能な入力は、シリアル ポートまたはパラレル ポートのいずれかでコンフィギュレーションできます。
Logical	ザイリンクスの Logical ブロックでは、2、3、または 4 つの固定小数点の値のビット単位の論理演算が実行されます。オペランドには、2 進小数点の位置を揃えるため、必要に応じて 0 がパディングされたり、符号が拡張されたりします。これらの値で論理演算が実行され、結果が出力ポートに送信されます。
MCode	ザイリンクスの MCode ブロックは、 Simulink でユーザーの提供する MATLAB 関数を実行するためのコンテナです。 M 関数の名前はブロックのパラメータから指定します。このブロックは、 M コードを実行して Simulink のシミュレーション中にブロックの出力を計算します。同じコードは、ハードウェアが生成されるときに、同等のビヘイビアレベルの VHDL/Verilog に直接変換されます。
ModelSim	System Generator の Black Box ブロックを使用すると、既存の HDL ファイルをモデルに組み込むことができます。モデルがシミュレーションされると、協調シミュレーションを使用してブラック ボックスがシミュレーションされるようになります。この ModelSim HDL 協調シミュレーション ブロックでは、1 つまたは複数のブラック ボックスの協調シミュレーションがコンフィギュレーションおよび制御できます。
Mult	ザイリンクスの Mult ブロックは、乗算器をインプリメントします。このブロックでは、2 つの入力ポートのデータの積が出力ポートに出力されます。
Multiple Subsystem Generator	ザイリンクスの Multiple Subsystem Generator ブロックは、複数の System Generator デザインを複数のクロック ドメインを使用する 1 つの最上位レベルの HDL コンポーネントに接続します。この最上位レベルのコンポーネントには、各 System Generator デザインに接続されたロジックと、デザインが互いに通信できるようにするロジックが含まれます。

表 1-6 : Index ブロック

Index ブロック	説明
Mux	ザイリンクスの Mux ブロックは、マルチプレクサをインプリメントします。ブロックには、セレクト入力 (符号なし) が 1 つと、ユーザーがコンフィギュレーション可能なデータ バス入力 (0 ~ 1024 の範囲) がいくつか含まれます。
Negate	ザイリンクスの Negate ブロックでは、入力の論理否定 (2 の補数) が実行されます。このブロックは、ザイリンクス LogiCORE または合成可能な VHDL モジュールとしてインプリメントできます。
Network-based Ethernet Co-Simulation	ザイリンクスの Network-based Ethernet Co-Simulation ブロックは、 IPv4 ネットワーク インフラストラクチャのイーサネット接続を介して、ハードウェア協調シミュレーションを実行するインターフェイスを提供します。
Opmode	ザイリンクスの Opmode ブロックでは、 DSP48 または DSP48E 命令の定数が生成されます。この命令は、 DSP48 の場合が 11 ビット値、 DSP48E の場合が 15 ビット値になります。命令には、 opmode 、 carry-in 、 carry-in select が含まれるほか、 DSP48 か DSP48E にかよって subtract または alumode ビットのいずれかが含まれます。
Parallel to Serial	Parallel to Serial ブロックは、入力ワードを読み込み、それを分割してマルチプレクサで N 倍多重化した出力ワードにします。 N は、入力ビットから出力ビットの数の比率です。出力順は、 LSB からか MSB からのどちらかになります。
Pause Simulation	ザイリンクスの Pause Simulation ブロックでは、入力が 0 以外の場合にシミュレーションを一時停止します。このブロックには、ザイリンクスの信号タイプでも入力として使用できます。
PicoBlaze Instruction Display	PicoBlaze Instruction Display ブロックは、エンコードされた 18 ビットの PicoBlaze 命令と 10 ビットのアドレスを読み込み、デコードされた命令とプログラム カウンタをブロックのアイコンに表示します。このブロックを使用すると、 PicoBlaze デザインをデバッグするのに便利です。また、このブロックを Single-Step Simulation ブロックと共に使用して、各命令を実行することもできます。
PicoBlaze Microcontroller	PicoBlaze Microcontroller ブロックは、 PicoBlaze マクロを使用して 8 ビットのエンベデッド マイクロコントローラをインプリメントします。
Point-to-point Ethernet Co-Simulation	ザイリンクスの Point-to-point Ethernet Co-Simulation ブロックは、イーサネット接続を介して、ハードウェア協調シミュレーションを実行するインターフェイスを提供します。
Puncture	ザイリンクスの Puncture ブロックでは、データ ストリームの入力ワードからユーザーが指定したビットのセットが削除されます。
Reed-Solomon Decoder 6.1	Reed-Solomon (RS) コードは、ブロック ベースの誤り訂正符号 (ECC) で、デジタル通信およびストレージ分野のさまざまなアプリケーションに使用されています。

表 1-6 : Index ブロック

Index ブロック	説明
Reed-Solomon Decoder 7.0	Reed-Solomon (RS) コードは、ブロック ベースの誤り訂正符号 (ECC) で、デジタル通信およびストレージ分野のさまざまなアプリケーションに使用されています。
Reed-Solomon Encoder 6.1	Reed-Solomon (RS) コードは、ブロック ベースの誤り訂正符号 (ECC) で、デジタル通信およびストレージ分野のさまざまなアプリケーションに使用されています。
Reed-Solomon Encoder 7.0	Reed-Solomon (RS) コードは、ブロック ベースの誤り訂正符号 (ECC) で、デジタル通信およびストレージ分野のさまざまなアプリケーションに使用されています。
Register	ザイリンクスの Register ブロックは、1 サンプル周期のレイテンシを含む D フリップフロップ ベースのレジスタになります。
Reinterpret	ザイリンクスの Reinterpret ブロックは、入力値に関係なく、出力を強制的に新しいデータ型にします。
Relational	ザイリンクスの Relational ブロックは、コンパレータをインプリメントします。
Reset Generator	Reset Generator ブロックは、システム サンプル レートで実行される ユーザー リセット信号をキャプチャし、ブロックで指定されたレートで実行される 1 つまたは複数のダウンサンプルされたリセット信号を生成します。
Resource Estimator	ザイリンクスの Resource Estimator ブロックを使用すると、System Generator サブシステムまたはモデルをインプリメントするのに必要な FPGA リソースが高速に概算できます。
ROM	ザイリンクスの ROM ブロックは、シングル ポートの読み出し専用メモリ (ROM) です。
Sample Time	Sample Time ブロックでは、標準化された入力値のサンプル周期がレポートされます。標準化されたサンプル周期は、Simulink の絶対サンプル周期とは異なります。ハードウェアでは、定数としてインプリメントされます。
Scale	ザイリンクスの Scale ブロックでは、入力が 2 のべき乗でスケール変換されます。べき乗の値は、正にでも負にでもできます。このブロックには、入力が 1 つと出力が 1 つあります。スケール変換では、コンテナ内のビットを変更せずに 2 進小数点の位置が移動されます。
Serial to Parallel	Serial to Parallel ブロックでは、連続して入力されたデータが指定した倍数のサイズの出力 1 つにまとめられて、出力されます。この連続入力は、最上位ワードまたは最下位ワードのいずれかを先頭にした順序になります。
Shared Memory	ザイリンクスの Shared Memory ブロックは、複数のデザインや 1 つのデザインの選択した箇所で共有できるランダム アクセス メモリ (RAM) をインプリメントします。

表 1-6 : Index ブロック

Index ブロック	説明
Shared Memory Read	ザイリンクスの Shared Memory Read ブロックは、ザイリンクスの共有メモリ オブジェクトからデータを読み出すための高速インターフェイスです。このブロックでは、FIFO オブジェクトとロック可能な共有メモリ オブジェクトの両方がサポートされます。
Shared Memory Write	ザイリンクスの Shared Memory Read ブロックは、ザイリンクスの共有メモリ オブジェクトにデータを書き込むための高速インターフェイスです。このブロックでは、FIFO オブジェクトとロック可能な共有メモリ オブジェクトの両方がサポートされます。
Shift	ザイリンクスの Shift ブロックは、入力信号を左または右にシフトします。出力には、入力と同じ固定小数点のコンテナが含まれます。
Simulation Multiplexer	Simulation Multiplexer ブロックは、System Generator で廃止されています。
Single Port RAM	ザイリンクスの Single Port RAM ブロックは、データ入力ポートとデータ出力ポートが 1 つずつ付いたランダム アクセス メモリ (RAM) をインプリメントします。
Single-Step Simulation	ザイリンクスの Single-Step Simulation ブロックは、シングル ステップ モードの場合にクロック サイクルごとにシミュレーションを一時停止します。
Slice	ザイリンクスの Slice ブロックを使用すると、入力データからビットのシーケンスを切り取り、新しいデータ値を作成できます。このデータ値がブロックから出力されます。出力データ型は、2 進小数点が 0 の位置の符合なしになります。
System Generator	System Generator ブロックは、システム制御およびシミュレーションパラメータを提供し、コード ジェネレータを起動するために使用されます。System Generator ブロックは、デザインにおける特異な役割のため、System Generator トークンと呼ばれることもあります。ザイリンクス ブロックセットからのエレメントを含む Simulink モデルには、System Generator ブロック (トークン) が最低 1 つは含まれます。System Generator ブロックをモデルに追加すると、コードの生成およびシミュレーションの処理方法を指定できるようになります。
Threshold	ザイリンクスの Threshold ブロックでは、入力数の符号がテストされます。入力数が負の場合、ブロックの出力は -1 になり、それ以外の場合は出力は 1 になります。出力は 2 ビットの長さの符号付き固定小数点整数です。ブロックには、入力と出力が 1 つずつ含まれます。
Time Division Demultiplexer	ザイリンクスの Time Division Demultiplexer ブロックは、シリアル入力をそれより遅いレート複数の出力にします。
Time Division Multiplexer	ザイリンクスの Time Division Multiplexer ブロックは、入力ポートの値を 1 つの高速レートの出力ストリームに多重化します。
To FIFO	ザイリンクス To FIFO ブロックは、First-In First-Out (FIFO) のメモリキューの前半分をインプリメントします。

表 1-6 : Index ブロック

Index ブロック	説明
To Register	ザイリンクスの To Register ブロックは、1 サンプル周期のレイテンシを含む D フリップフロップ ベースのレジスタの前半部分をインプリメントします。レジスタは、複数デザインまたは 1 つのデザインの複数セクションで共有できます。
Toolbar	ザイリンクスの Toolbar ブロックを使用すると、System Generator の複数の便利なユーティリティに素早くアクセスできます。Toolbar ブロックは、Simulink の拡大/縮小表示を簡単にし、新しい自動レイアウトや Simulink モデルへの配線機能を追加します。
Up Sample	ザイリンクスの Up Sample ブロックは、ブロックが配置された箇所のサンプル レートを増加するために使用します。出力サンプル周期は、 $1/n$ になります (1 が入力サンプル周期、n はサンプリング レート)。
Viterbi Decoder 6_1	たたみ込みエンコーダでエンコードされたデータは、ザイリンクスの Viterbi Decoder ブロックを使用してデコードできます。
WaveScope	System Generator の WaveScope ブロックは、高度な機能を備えた使いやすい波形ビューアで、System Generator デザインの解析およびデバッグに使用します。

Math ブロック

表 1-7 : Math ブロック

Math ブロック	説明
Accumulator	ザイリンクスの Accumulator ブロックは、加算器または減算器ベースのスケールリング アキュムレータです。
AddSub	ザイリンクスの AddSub ブロックは、加算器/減算器です。処理方法は、加算か減算のどちらかに指定することもできますが、sub 信号の制御に従ってダイナミックに変更することもできます。
CMult	ザイリンクスの CMult ブロックは、gain 演算子をインプリメントします。出力は、入力と定数値の積になります。この値は、定数を求める MATLAB の論理式で表すことができます。
Complex Multiplier 3.0	ザイリンクスの Complex Multiplier ブロックは、2 つの複素数を乗算します。
Complex Multiplier 3.1	ザイリンクスの Complex Multiplier ブロックは、2 つの複素数を乗算します。
Constant	ザイリンクスの Constant ブロックでは、固定小数点値、ブール値、DSP48 命令のいずれかの定数が生成されます。このブロックは、Simulink の Constant ブロックと類似していますが、ザイリンクス ブロックの入力を直接駆動できる点が異なります。

表 1-7 : Math ブロック

Math ブロック	説明
Convert	ザイリンクスの Convert ブロックでは、各入力サンプルが指定した演算タイプの値に変換されます。たとえば、ある数値を符号付き (2 の補数) または符号なしの値にできます。
CORDIC 4.0	ザイリンクスの CORDIC 4.0 ブロックは、 CORDIC (Coordinate Rotational Digital Computer) アルゴリズムをインプリメントします。
Counter	ザイリンクスの Counter ブロックは、フリー ランニング カウンタ、またはアップ カウンタ、ダウン カウンタ、アップ/ダウン カウンタなどのカウント制限のあるカウンタをインプリメントします。カウンタの出力は、符号付きまたは符号なしの固定小数点の値で指定できます。
Divider Generator 2.0	ザイリンクスの Divider Generator 2.0 ブロックは、 Radix-2 の非回復型除算、またはプリスケールリングを用いた高基数除算に基づいて整数除算を行う回路を作成します。
Expression	ザイリンクスの Expression ブロックは、ビット単位の論理式を実行します。
Inverter	ザイリンクスの Inverter ブロックでは、固定小数点型の論理コンポーネントがビット単位で計算されます。このブロックは、合成可能な VHDL モジュールとしてインプリメントされます。
Logical	ザイリンクスの Logical ブロックでは、2、3、または 4 つの固定小数点の値のビット単位の論理演算が実行されます。オペランドには、2 進小数点の位置を揃えるため、必要に応じて 0 がパディングされたり、符号が拡張されたりします。これらの値で論理演算が実行され、結果が出力ポートに送信されます。
MCode	ザイリンクスの MCode ブロックは、 Simulink でユーザーの提供する MATLAB 関数を実行するためのコンテナです。M 関数の名前はブロックのパラメータから指定します。このブロックは、M コードを実行して Simulink のシミュレーション中にブロックの出力を計算します。同じコードは、ハードウェアが生成されるときに、同等のビヘイビアレベルの VHDL/Verilog に直接変換されます。
Mult	ザイリンクスの Mult ブロックは、乗算器をインプリメントします。このブロックでは、2 つの入力ポートのデータの積が出力ポートに出力されます。
Negate	ザイリンクスの Negate ブロックでは、入力の論理否定 (2 の補数) が実行されます。このブロックは、ザイリンクス LogiCORE または合成可能な VHDL モジュールとしてインプリメントできます。
Reinterpret	ザイリンクスの Reinterpret ブロックは、入力値に関係なく、出力を強制的に新しいデータ型にします。
Relational	ザイリンクスの Relational ブロックは、コンパレータをインプリメントします。

表 1-7 : Math ブロック

Math ブロック	説明
Scale	ザイリンクスの Scale ブロックでは、入力が 2 のべき乗でスケール変換されます。べき乗の値は、正にでも負にでもできます。このブロックには、入力が 1 つと出力が 1 つあります。スケール変換では、コンテナ内のビットを変更せずに 2 進小数点の位置が移動されます。
Shift	ザイリンクスの Shift ブロックは、入力信号を左または右にシフトします。出力には、入力と同じ固定小数点のコンテナが含まれます。
Threshold	ザイリンクスの Threshold ブロックでは、入力数の符号がテストされます。入力数が負の場合、ブロックの出力は -1 になり、それ以外の場合は出力は 1 になります。出力は 2 ビットの長さの符号付き固定小数点整数です。ブロックには、入力と出力が 1 つずつ含まれます。

Memory ブロック

表 1-8 : Memory ブロック

Memory ブロック	説明
Addressable Shift Register	ザイリンクスの Addressable Shift Register ブロックは、可変長シフトレジスタで、遅延チェーンのどのレジスタでも呼び出して、出力データポートに駆動できます。
Delay	Delay ブロックは、L サイクルの固定遅延をインプリメントします。
Dual Port RAM	ザイリンクスの Dual Port RAM ブロックは、ランダム アクセス メモリ (RAM) をインプリメントします。デュアルポートを使用すると、複数のデータ幅を使用して異なるサンプル レートでメモリ空間に同時にアクセスできます。
LFSR	ザイリンクスの LFSR ブロックは、 LFSR (Linear Feedback Shift Register) をインプリメントします。このブロックでは、 XOR または XNOR のいずれかを使用してガロアおよびフィボナッチ構造の両方がサポートされるほか、リロード可能な入力を使用して、レジスタの値をいつでも変更できます。 LFSR の出力とリロード可能な入力は、シリアルポートまたはパラレルポートのいずれかでコンフィギュレーションできます。
Register	ザイリンクスの Register ブロックは、1 サンプル周期のレイテンシを含む D フリップフロップベースのレジスタになります。
ROM	ザイリンクスの ROM ブロックは、シングルポートの読み出し専用メモリ (ROM) です。

表 1-8 : Memory ブロック

Memory ブロック	説明
Shared Memory	ザイリンクスの Shared Memory ブロックは、複数のデザインや 1 つのデザインの選択した箇所で共有できるランダム アクセス メモリ (RAM) をインプリメントします。
Single Port RAM	ザイリンクスの Single Port RAM ブロックは、データ入力ポートとデータ出力ポートが 1 つずつ付いたランダム アクセス メモリ (RAM) をインプリメントします。

Shared Memory ブロック

表 1-9 : Shared Memory ブロック

Shared Memory ブロック	説明
From FIFO	ザイリンクス From FIFO ブロックは、First-In First-Out (FIFO) のメモリ キューの後半分をインプリメントします。
From Register	ザイリンクス From Register ブロックは、D フリップフロップ ベースのレジスタの後半分をインプリメントします。物理的なレジスタは、2 つのデザインまたは同じデザインの 2 箇所で共有できます。
Multiple Subsystem Generator	ザイリンクスの Multiple Subsystem Generator ブロックは、複数の System Generator デザインを複数のクロック ドメインを使用する 1 つの最上位レベルの HDL コンポーネントに接続します。この最上位レベルのコンポーネントには、各 System Generator デザインに接続されたロジックと、デザインが互いに通信できるようにするロジックが含まれます。
Shared Memory	ザイリンクスの Shared Memory ブロックは、複数のデザインや 1 つのデザインの選択した箇所で共有できるランダム アクセス メモリ (RAM) をインプリメントします。
Shared Memory Read	ザイリンクスの Shared Memory Read ブロックは、ザイリンクスの共有メモリ オブジェクトからデータを読み出すための高速インターフェイスです。このブロックでは、FIFO オブジェクトとロック可能な共有メモリ オブジェクトの両方がサポートされます。
Shared Memory Write	ザイリンクスの Shared Memory Read ブロックは、ザイリンクスの共有メモリ オブジェクトにデータを書き込むための高速インターフェイスです。このブロックでは、FIFO オブジェクトとロック可能な共有メモリ オブジェクトの両方がサポートされます。
To FIFO	ザイリンクス To FIFO ブロックは、First-In First-Out (FIFO) のメモリ キューの前半分をインプリメントします。
To Register	ザイリンクスの To Register ブロックは、1 サンプル周期のレイテンシを含む D フリップフロップ ベースのレジスタの前半部分をインプリメントします。レジスタは、複数デザインまたは 1 つのデザインの複数セクションで共有できます。

Tools ブロック

表 1-10 : Tools ブロック

Tools ブロック	説明
ChipScope	ザイリンクス ChipScop™ ブロックでは、ランタイム デバッグおよび FPGA 内の信号検証などを実行できます。
Clock Probe	ザイリンクスの Clock Probe ブロックでは、倍精度のクロック信号が Simulink システム周期と同じ周期で出力されます。
Configurable Subsystem Manager	ザイリンクスの Configurable Subsystem Manager ブロックは、Simulink のコンフィギャブル システムの機能を拡張したブロックで、シミュレーションと同様、ハードウェアの生成にもサブシステム コンフィギュレーションが選択できるようになります。
FDATool	ザイリンクスの FDATool ブロックは、MATLAB の Signal Processing Toolbox の一部である FDATool ソフトウェアへのインターフェイスになります。
Indeterminate Probe	ザイリンクスの Indeterminate Probe ブロックの出力は、入力データが不定値 (MATLAB 値が NaN) かどうかを示します。不定データは、VHDL の不定データ値を表す X に相当します。
ModelSim	System Generator の Black Box ブロックを使用すると、既存の HDL ファイルをモデルに組み込むことができます。モデルがシミュレーションされると、協調シミュレーションを使用してブラック ボックスがシミュレーションされるようになります。この ModelSim HDL 協調シミュレーション ブロックでは、1 つまたは複数のブラック ボックスの協調シミュレーションがコンフィギュレーションおよび制御できます。
Pause Simulation	ザイリンクスの Pause Simulation ブロックでは、入力が 0 以外の場合にシミュレーションを一時停止します。このブロックには、ザイリンクスの信号タイプでも入力として使用できます。
PicoBlaze Instruction Display	PicoBlaze Instruction Display ブロックは、エンコードされた 18 ビットの PicoBlaze 命令と 10 ビットのアドレスを読み込み、デコードされた命令とプログラム カウンタをブロックのアイコンに表示します。このブロックを使用すると、PicoBlaze デザインをデバッグするのに便利です。また、このブロックを Single-Step Simulation ブロックと共に使用して、各命令を実行することもできます。
Resource Estimator	ザイリンクスの Resource Estimator ブロックを使用すると、System Generator サブシステムまたはモデルをインプリメントするのに必要な FPGA リソースが高速に概算できます。
Simulation Multiplexer	Simulation Multiplexer ブロックは、System Generator で廃止されています。
Single-Step Simulation	ザイリンクスの Single-Step Simulation ブロックは、シングル ステップ モードの場合にクロック サイクルごとにシミュレーションを一時停止します。

表 1-10 : Tools ブロック

Tools ブロック	説明
System Generator	System Generator ブロックは、システム制御およびシミュレーションパラメータを提供し、コード ジェネレータを起動するために使用されます。 System Generator ブロックは、デザインにおける特異な役割のため、 System Generator トークンと呼ばれることもあります。ザイリンクス ブロックセットからのエレメントを含む Simulink モデルには、 System Generator ブロック (トークン) が最低 1 つは含まれます。 System Generator ブロックをモデルに追加すると、コードの生成およびシミュレーションの処理方法を指定できるようになります。
Toolbar	ザイリンクスの Toolbar ブロックを使用すると、 System Generator の複数の便利なユーティリティに素早くアクセスできます。 Toolbar ブロックは、 Simulink の拡大/縮小表示を簡単にし、新しい自動レイアウトや Simulink モデルへの配線機能を追加します。
WaveScope	System Generator の WaveScope ブロックは、高度な機能を備えた使いやすい波形ビューアで、 System Generator デザインの解析およびデバッグに使用します。

System Generator でサポートされる Simulink ブロック

通常、[Simulink](#) ブロックはシミュレーション用にザイリンクス デザインに含まれますが、ザイリンクス ハードウェアにはマップされません。ただし、次の [Simulink](#) ブロックはで完全にサポートされ、ザイリンクス ハードウェアにマップされます。

表 1-11 : System Generator でサポートされる Simulink ブロック

Simulink ブロック	説明
Demux	Demux ブロックは、入力信号のコンポーネントを抽出し、それを別々の信号として出力します。
From	From ブロックは、対応する Goto ブロックからの信号を受信し、それを出力に渡します。
Goto	Goto ブロックはその入力を対応する From ブロックに渡します。
Mux	Mux ブロックは、入力を 1 つのベクタ出力にまとめます。

このブロックの詳細は、[Simulink](#) のマニュアルを参照してください。

ブロックのパラメータ ダイアログ ボックスの共通オプション

各ザイリンクス ブロックの制御やパラメータは、そのブロックのパラメータ ダイアログ ボックスから設定できます。このダイアログ ボックスは、ブロックをダブルクリックすると表示されます。これらのパラメータの多くがそのブロックに特有のものです。ブロック特有のパラメータの詳細については、そのブロックの説明を参照してください。

その他の制御およびパラメータは、ほとんどのブロックで共通しています。次は、共通の制御ボタンおよびパラメータです。

このダイアログ ボックスには、[OK]、[Cancel]、[Help]、[Apply] の 4 つのボタンがあります。[Apply] は、ダイアログ ボックスを開いたままブロックに対する設定変更を適用し、[Help] は、そのブロックの HTML ヘルプを表示します。[Cancel] は、変更を保存せずにダイアログ ボックスを閉じ、[OK] は、変更を適用してからダイアログ ボックスを閉じます。

[Precision]

ザイリンクス ブロックセットの基本的な計算モードは、任意の精度 (Precision) の固定小数点になっています。ほとんどのブロックに、ビット数や 2 進小数点などの精度を選択できるオプションがあります。

デフォルトでは、ザイリンクス ブロックの出力は完全精度 (Full) に設定されています。これは、エラーなく出力結果を表示するのに十分な精度です。ほとんどのブロックで、ユーザー定義 (User-defined) の精度を選択できます。この精度の場合、合計ビットと小数点以下のビット数が固定されます。

[Type]

ブロックのパラメータ ダイアログ ボックスの [Type] フィールドから、出力信号のデータ型に [Unsigned] (符号なし) か [Signed (2's comp)] (符号付き (2 の補数)) を選択できます。

[Number of bits]

固定小数点は、ビット数、2 進小数点、演算タイプなどのパラメータで指定したとおり、ワード長のデータ型に格納されます。サポートされる最大ビット数は、4096 です。

[Binary point]

2 進小数点 (Binary point) は、固定小数点のスケールリングに使用されます。このパラメータは、出力ポートの 2 進数小数点の右側のビット数 (小数点以下のビットの大きさ) を示します。2 進小数点の位置は、0 ～ 指定ビット数の間にする必要があります。

[Overflow] と [Quantization]

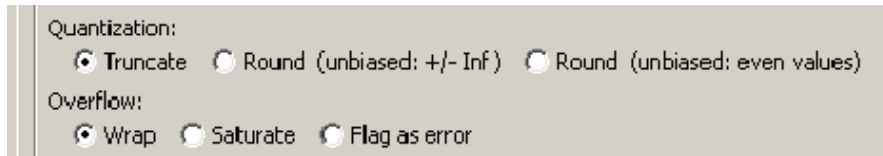
ユーザー定義の精度が選択されている場合、オーバーフロー (Overflow) または量子化 (Quantization) が原因でエラーが発生することがあります。オーバーフローのエラーは、データが表現可能な値の範囲外である場合に発生します。量子化のエラーは、小数点以下のビット数が値のそれを表現するのに不十分な場合に発生します。

ザイリンクスの固定小数点型では、ユーザー定義の精度に対して複数のオプションがサポートされます。オーバーフローの場合、[Saturate] (上限が最大の正の値、下限が最小の負の値に飽和演算)、[Wrap] (MSB よりも左のビットを削除)、[Flag as error] (シミュレーション中に Simulink エラーと

してオーバーフロー) というオプションがあります。[Flag as error] は、シミュレーションのみに使用できます。生成されたハードウェアは、[Wrap] をオンにしたときと同じになります。

量子化の場合、[Round] と [Truncate] というオプションがあります。[Round] は、四捨五入 (最近似値への四捨五入。同等の近似値が 2 つある場合は、0 から遠い方の値に四捨五入)、[Truncate] は切り捨て (LSB よりも右のビットを削除) を実行します。

次は、量子化とオーバーフローのオプションを設定するインターフェイスです。



[Round(unbiased: +/- inf)] では、正または負の無限大への対称丸めが実行されます。これは、MATLAB の round() 関数と類似しており、0 から遠い方の最も近いビットに値が丸められます。2 つの値の調度真ん中に値がある場合は、2 つの値のうちの絶対値が大きい方の値が選択されます。たとえば、01.0110 を Fix_4_2 に丸めると、01.10 になります。これは、01.0110 が 01.01 と 01.10 の調度真ん中の値で、01.10 の方が 0 より遠いからです。

[Round (unbiased: even values)] では、偶数丸めまたは不偏丸めが実行されます。対称丸めはバイアスされます。これは、対称丸めが 0 から遠いすべての曖昧な中間値を丸めるからで、丸められた値の平均的な絶対値は、処理前の値の平均的な絶対値よりも大きくなります。偶数丸めでは、0 に近い方への丸めと遠い方への丸めを切り替えることで、これが削除されます。つまり、中間値は一番近い偶数値に丸められます。たとえば、01.0110 を Fix_4_2 に丸めると、01.10 になります。これは、01.0110 が 01.01 と 01.10 の調度真ん中の値で、01.10 が偶数だからです。01.1010 を Fix_4_2 に丸めると、01.10 になります。これは、01.1010 が 01.10 と 01.11 の調度真ん中の値で、01.10 が偶数だからです。

どのオプションを選択しても、生成される HDL モデルと Simulink モデルの動作は同一になります。

[Latency]

ザイリンクス ブロックセットのエレメントの多くに、レイテンシ オプションがあり、ブロックの出力を遅延させるサンプル周期が定義できるようになっています。1 サンプル周期は FPGA インプリメンテーションの複数クロック サイクルに相当することがあります (ハードウェアが Simulink モデルに対してオーバークロックになる場合など)。System Generator では、余分なパイプラインを実行しません。レイテンシを追加する場合は、通常ブロックの出力にシフト レジスタをインプリメントします。

[Provide synchronous reset port]

[Provide synchronous reset port] をオンにすると、ブロックのオプションのリセット (rst) ピンが有効になります。

このリセット信号がアサートされると、ブロックが最初のステートに戻ります。リセット信号は、ブロックで使用可能なオプションのイネーブル信号よりも優先されます。この信号は、ブロックのサンプル レートの倍数のレートで実行する必要があり、リセット ポートを駆動する信号は、ブール型にする必要があります。

[Provide enable port]

[Provide enable port] をオンにすると、ブロックのオプションのイネーブル (en) ピンが有効になります。イネーブル信号がアサートされないと、イネーブル信号が再びアサートされるまで (またはリセット信号がアサートされるまで)、ブロックが現在のステートを保持したままになります。リセット信号はイネーブル信号よりも優先されます。イネーブル信号は、ブロックのサンプルレートの倍数のレートで実行する必要があり、イネーブル ポートを駆動する信号は、ブール型にする必要があります。

[Sample Period]

データ ストリームは、特定のサンプル レートで **Simulink** を通ります。通常は、各ブロックで入力 サンプル レートが検出され、正しいサンプル レートが出力に送られますが、ザイリンクスの **Up Sample** ブロックと **Down Sample** ブロックを使用すると、サンプル レートを増加したり、減少したりできます。

[Specify explicit sample period]

デフォルトのままではなく、[Specify explicit sample period] をオンにすると、すべてのブロック出力に必要なサンプル周期を設定できます。このオプションは、フィードバック ループなどの機能をデザインにインプリメントする際に便利です。 フィードバック ループを使用する場合、**System Generator** でデフォルトのサンプル レートを決定できません。これは、フィードバック ループでは、まだ決まっていない出力サンプル レートによって入力サンプル レートが作成されるからです。**System Generator** には、このような場合、ループ全体のサンプル周期を構築するヒントを入力する必要があります。

[Use behavioral HDL (otherwise use core)]

オンにすると、コアからの構造レベルの **HDL** 記述ではなく、**M** コード シミュレーションで生成されたビヘイビア レベルの **HDL** 記述が使用されます。

M コード シミュレーションでは、**C** シミュレーションが作成され、この **C** シミュレーションによりビヘイビア レベルの **HDL** が作成されます。このオプションをオンにすると、後の合成でこのビヘイビア レベルの **HDL** 記述が使用されます。オフにすると、コアおよび **HDL** テンプレート (モデル内のブロックにそれぞれ対応) から生成された構造記述の **HDL** が合成に使用されます。デザインの各ブロック用に生成されたコアは、キャッシュに格納されて、後のネットリストで使用されます。これは、一番速いネットリスト生成方法で、そのコアが後の合成ツールと配置配線ツールで使用できるという利点もあります。

[Use core placement information]

オンにすると、生成されるコアに相対配置情報が含まれ、通常はインプリメンテーションが高速になります。配置ではこの情報に基づいて制約が付くので、配置配線ソフトウェアの速度が遅くなることもあります。

[Use XtremeDSP Slice]

オンにすると、可能な場合に **XtremeDSP** スライス (**DSP48** タイプのエレメント) がターゲット デバイスで使用されます。可能でなければ、乗算器には **CLB** ロジックが使用されます。

[Placement]

乗算器コアの場合に [Use core placement information] をオンにすると、このオプションが表示され、ハードウェアに配置される乗算器コアの形状を指定できます。[Rectangular] を選択すると、配置される LUT の密度が低い四角形のコアが生成されます。[Triangular] を選択すると、高密度配置の LUT を使用して、コンパクトな形状のコアが作成されます。

[FPGA Area (slices, FFs, BRAMs, LUTs, IOBs, emb. mults, TBUFs)] / [Use area above for estimation]

これらのフィールドは、Resource Estimator ブロックで使用されます。Resource Estimator を使用すると、System Generator デザインに必要なハードウェア リソースを計算できます。

デザインに Resource Estimator ブロックを配置した場合、[FPGA area] ボックスにそのブロックの FPGA エリア使用率を手動で入力できます。この値を入力しない場合は、Resource Estimator ブロックで自動的に値が計算されて入力されます。

そのブロック用の値を直接入力する場合は、[Define FPGA area for resource estimation] をオンにして、Resource Estimator ブロックでその値が使用されるようにする必要があります。オンにしておかないと、Resource Estimator ブロックで FPGA エリアが計測され、このフィールドに入力した値が上書きされてしまいます。

[FPGA area] ボックスに入力できる値は、次の 7 つです。値は、それぞれ正しい位置に入力する必要があります。この値が [1,2,3,4,5,6,7] の場合、各値は次を示します。

- 1 = ブロックで使用されるスライス。1 つの FPGA スライスには、通常フリップフロップが 2 つ、LUT が 2 つ、関連する MUX、キャリー ロジック、制御ロジックが含まれます。
- 2 = ブロックで使用されるフリップフロップ
- 3 = ブロックで使用されるブロック RAM (BRAM)
- 4 = ブロックで使用される LUT
- 5 = ブロックで消費される IOB
- 6 = ブロックで使用されるエンベデッド (Emb.) 乗算器。
- 7 = ブロックで使用されるトライステート バッファ (TBUF)

Resource Estimator ブロックで考慮されるのは、ハードウェア コストのあるザイリンクス ブロック (物理的なハードウェア リソースが必要なブロック) のみです。[FPGA area] ボックスは、関連するハードウェアがないブロックでは表示されません。

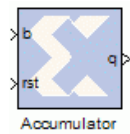
スライスは LUT とフリップフロップに関連していても (各スライスには LUT が 1 つとフリップフロップが 1 つ含まれる)、パックされるスライス数はデザインによって異なるため、別々に入力されます。

Resource Estimator ブロックの説明にあるように、ザイリンクス ブロックの中には自動的なリソースの概算がサポートされないものがあります。こういったブロックの [FPGA area] ボックスは自動的にアップデートされないため、MATLAB のコンソール ウィンドウに警告メッセージが表示されます。

ブロックのリファレンス ページ

Accumulator

このブロックは、[Xilinx Blockset] の [Math] および [Index] ライブラリにリストされています。



ザイリンクスの Accumulator ブロックは、加算器または減算器ベースのスケールリング アキュムレータです。

このブロックの入力値は、スケールされた格納値に累積されていきます。このスケール係数は、パラメータ ダイアログ ボックスから指定できます。

ブロック インターフェイス

このブロックには、**b** 入力と **q** 出力があり、出力データ幅は、入力データ幅と同じにする必要があります。また、出力と入力には、同じ演算タイプと 2 進小数点の位置が使用されます。この **q** 出力は、次のように計算されます。

$$q(n) = \begin{cases} 0 & \text{if } rst=1 \\ q(n-1) \times FeedbackScaling + b(n-1) & \text{otherwise} \end{cases}$$

減算器ベースのアキュムレータの場合は、**b(n)** 入力の加算が減算になります。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Operation] : ブロックを加算器ベースにするか減算器ベースにするかを指定します。
- [Feedback scaling] : フィードバックのスケール係数を次のいずれかに指定します。
1、1/2、1/4、1/8、1/16、1/32、1/64、1/128、1/256
- [Reinitialize with input 'b' on reset] : オンにすると、アキュムレータの出力が入力ポート **b** のデータにリセットされます。オフにすると、アキュムレータの出力は **0** にリセットされます。このオプションはブロックにリセット ポートがある場合にのみ使用できます。アキュムレータがマルチレート システムに含まれる場合にオンにすると、クロック速度に影響が出ます。この場合、アキュムレータがシステム レートで強制的に実行されます。これは、アキュムレータを駆動するクロック イネーブル信号 (CE) がシステム レートで実行され、入力に対するリセット動作がこの CE 信号で指定されるからです。

[Implementation] タブ

[Implementation] タブからは、次のようなパラメータを設定できます。

- [Use behavioral HDL (otherwise use core)] : ビヘイビア レベルの HDL 記述が使用されます。これで、最適化をパフォーマンス重視にするか、エリア重視にするかをダウンストリームのロジック合成ツールで柔軟に決定できます。
- [Implement using] : コア ロジックはファブリックか、DSP48 がターゲット デバイスで使用可能な場合は DSP48 にインプリメントできます。デフォルトは [Fabric] です。

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

Accumulator ブロックのレイテンシの値は常に 1 です。

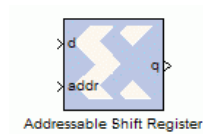
ザイリンクス LogiCORE

ビヘイビア レベルの HDL 記述のオプションが使用されない場合、このブロックはザイリンクス LogiCORE を使用します。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
Accumulator	Accumulator	v11.0	•	•	•	•	•	•	•	•	•	•

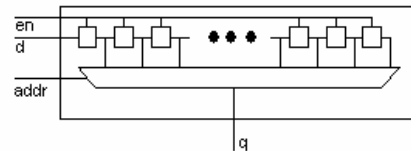
Addressable Shift Register

このブロックは、[Xilinx Blockset] の [Basic Elements]、[Memory]、[Index] ライブラリにリストされています。



ザイリンクスの Addressable Shift Register ブロックは、可変長シフトレジスタで、遅延チェーンのどのレジスタでも呼び出して、出力データポートに駆動できます。

このブロックは、チェーン接続されたレジスタと考えるとわかりやすく、次の図に示すように、各レジスタの出力はマルチプレクサの入力に接続されます。このマルチプレクサのセレクトラインは、アドレスポート (addr) で駆動されます。出力データポートは、次の図では q と表示されています。



Addressable Shift Register ブロックの深さは最大 1024 ビットで、最小 2 ビットです。このため、アドレス入力ポートは、1 ～ 10 ビットになります。データ入力ポートの幅は、このブロックがザイリンクス LogiCORE を使用してインプリメントされる場合 ([Use behavioral HDL (otherwise use core)] がオフの場合)、1 ～ 255 ビットにする必要があります。

ハードウェアでは、アドレスポートは出力ポートに対して非同期です。このため、ブロックの S 関数では、アドレスポートが入力データポートよりも優先されます。つまり、次に続くサイクルでは、レジスタから読み出されたアドレスデータ値が、シフト動作の実行される前に出力に駆動されます。Simulink ソフトウェアモデルでは、この順序でないと、データポートと遅延チェーンの最初のレジスタ間に 1 クロックサイクルのレイテンシが確保されません (シフト動作が最初で読み出しがその後続く場合、遅延がなくなるので、ハードウェアは不正な状態になります)。

ブロック インターフェイス

ブロック インターフェイス (Addressable Shift Register のアイコンの入力と出力) は、次のとおりです。

入力信号

d	データ入力
addr	アドレス
en	イネーブル信号 (オプション)

出力信号

q	データ出力
---	-------

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

このブロックでは、次のパラメータが設定できます。

- **[Infer maximum latency (depth) using address port width]** : アドレス ポートのビット幅に基づいて、シフト レジスタの深さまたは最大レイテンシが自動的に決定されるようにできます。
- **[Maximum latency (depth)]** : 上のオプションで最大レイテンシが決定されない場合、手動で最大レイテンシを入力します。
- **[Initial value vector]** : 初期レジスタ値を指定します。ベクタ長がシフト レジスタの深さよりも大きい場合、このベクタの後に続くエレメントは削除されます。シフト レジスタの深さがベクタ長よりも大きい場合、このシフト レジスタの後に続くレジスタが 0 に初期化されます。

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

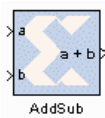
ザイリンクス LogiCORE

このブロックでは、ビヘイビア レベルの HDL 記述を使用しない場合、ザイリンクス LogiCORE の RAM-based Shift Register が使用されます。LogiCORE を使用する場合、データ入力ポートの幅は、1 ～ 255 ビットです。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
Addressable Shift Register	RAM-based Shift Register	V11.0	•	•	•	•	•	•	•	•	•	•

AddSub

このブロックは、[Xilinx Blockset] の [Math] および [Index] ライブラリにリストされています。



ザイリンクスの AddSub ブロックは、加算器 / 減算器です。処理方法は、加算か減算のどちらかに指定することもできますが、sub 信号の制御に従ってダイナミックに変更することもできます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Operation] : ブロックを [Addition]、[Subtraction]、[Addition or subtraction] に指定します。[Addition or subtraction] をオンにすると、その sub 入力ポートの値によって加算器か減算器になります。この入力ポートは必ずブール信号で駆動されます。sub 入力が 1 の場合は減算器になり、それ以外の場合は加算器になります。
- [Provide carry-in port] : オンにすると、キャリーイン ポート (cin) が含まれます。キャリーイン ポートは、精度に [User defined] を選択し、入力の 2 進小数点を 0 に設定した場合にのみ使用できます。
- [Provide carry-out port] : オンにすると、キャリーアウト ポート (cout) が含まれます。キャリーアウト ポートは、精度に [User defined] を選択し、入力と出力が符号なし (Unsigned) で、出力整数ビットが x ($x = \max(\text{整数ビット } a, \text{整数ビット } b)$) の場合にのみ使用できます。

[Implementation] タブ

[Implementation] タブからは、次のようなパラメータを設定できます。

- [Use behavioral HDL (otherwise use core)] : ビヘイビア レベルの HDL 記述が使用されます。これで、最適化をパフォーマンス重視にするか、エリア重視にするかをダウンストリームのロジック合成ツールで柔軟に決定できます。

コア パラメータ

- [Pipeline for maximum performance] : ザイリンクス LogiCORE は、内部でパイプライン接続して、エリアではなくスピードを改善できます。このオプションをオンにすると、最大許容レイテンシに到達するまで、すべてのユーザー定義のレイテンシがコアに含まれるようになります。このオプションをオフにした場合にレイテンシが 0 を超えると、コアには 1 つの出力レジスタが含まれ、それ以上のレイテンシはコアの外に追加されていきます。
- [Implement using] : コア ロジックはファブリックか、DSP48 がターゲット デバイスで可能な場合は DSP48 にインプリメントできます。デフォルトは [Fabric] です。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

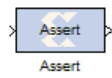
ザイリンクス LogiCORE

このブロックでは、ビヘイビア レベルの HDL 記述を使用しない場合、このブロックは、次のザイリンクス LogiCORE コアを使用します。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
AddSub	Adder Subtractor	V11.0	•	•	•	•	•	•	•	•	•	•

Assert

このブロックは、[Xilinx Blockset] の [Index] ライブラリにリストされています。



Assert ブロックは、信号にレートやデータ型をアサートするために使用します。このブロックには、ハードウェア コストがかかりません。このブロックを使用すると、設計者が手を入れる必要のある場合に、レートやデータ型の問題を改善できます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

このブロックでは、次のパラメータが設定できます。

- **[Assert type]** : 入力に指定したデータ型と同じ型をアサートするかどうかを指定します。データ型が同じでない場合は、エラー メッセージが表示されます。
- **[Specify type]** : アサートするデータ型を **type** という入力ポートに接続された信号から決定するか、このダイアログ ボックスの **[Explicitly]** をオンにして決定するかを指定します。
- **[Assert rate]** : 入力に指定したレートと同じレートをアサートするかどうかを指定します。レートが同じでない場合は、エラー メッセージが表示されます。
- **[Specify rate]** : アサートする最初のレートを **rate** という入力ポートに接続された信号から決定するか、このダイアログ ボックスの **[Explicitly]** をオンにして決定するかを指定します。
- **[Provide output port]** : ブロックに出力ポートを含めるかどうかを指定します。出力ポートの信号のタイプやデータ型には、アサートに指定したものが使用されます。

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

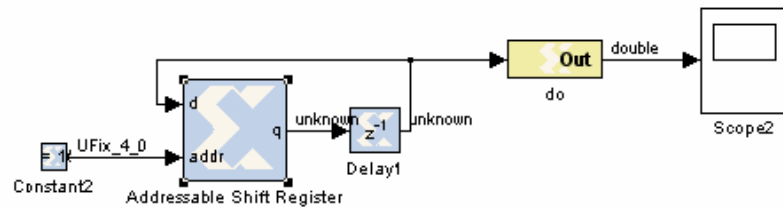
このブロックの **[Output type]** のオプションは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明される演算タイプと同じです。

このブロックでは、ザイリンクス LogiCORE は使用されず、ハードウェアにインプリメントされてもリソースが使用されません。

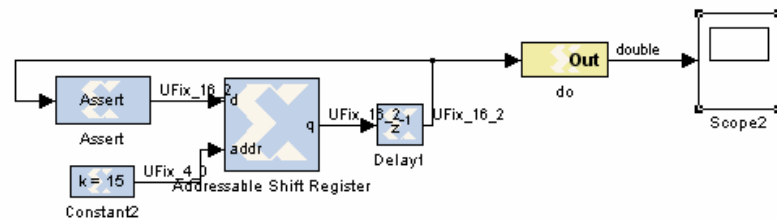
Assert ブロックを使用したレートおよびデータ型の改善方法

シミュレーション エンジンでレートやデータ型の問題が回避できなかった場合は、Assert ブロックを使用して特定レートやデータ型を強制的に設定します。通常、このブロックを使用するのは、フィードバックを使用し、信号ソースとして動作するコンポーネントを使用した場合です。たとえば、次の回路図では、SRL16 のレートとデータ型を設定するために Assert ブロックが必要になります。この場合、Assert ブロックを使用すると、最初のレートが設定できます。これが SRL16 を通った後、Assert ブロックに戻り、SRL16 の入力に戻ります。次の回路図の場合、Assert ブロックが使用されていないので、次のようなエラー メッセージが表示されます。

“The data types could not be established for the feedback paths through this block. You may need to add **Assert** blocks to instruct the system how to resolve types.



この問題を回避するには、次の図のようにフィードバックパスに **Assert** ブロックを含めます。:

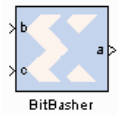


この例の場合、データ型を変更するのに **Assert** ブロックが必要ですが、レートは **Constant** ブロックに割り当てて設定することもできます。レートを強制的に設定するのに **Constant** ブロックを使用するか **Assert** ブロックを使用するかは、ユーザーの任意です。状況によって、どちらを使用するか決めてください。

System Generator 8.1 以降のバージョンでは、レートとデータ型を確定して問題を回避できますが、System Generator のコンポーネントの中には、問題が回避できそうな場合でも **Assert** ブロックを必要とするものがあります。必要とするのは、ブラック ボックス コンポーネントや一部の IP ブロックなどです。

BitBasher

このブロックは、[Xilinx Blockset] の [Basic Elements]、[Data Types]、[Index] ライブラリにリストされています。



ザイリンクスの **BitBasher** ブロックでは、ブロックに接続された入力の値がスライスされた後、連結されて追加されます。

これらの操作は、**Verilog** 構文を使用して記述されます (詳細は、このマニュアルに記述されています)。このブロックでは、最大で 4 つの出力ポートが使用可能です。出力ポートの数は、論理式の数と同じになります。このブロックには、ハードウェア コストがかかりません。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- **[BitBasher Expression]** : Verilog 構文に基づいたビット単位の論理式です。論理式は、改行して入力すると、最大 4 つまで指定できます。

[Output Type] タブ

- **[Output]** : データ型の指定されたポートが表示されます。
- **[Output type]** : 出力の演算タイプを設定します。
- **[Binary point]** : 出力の 2 進小数点の位置を設定します。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

サポートされる Verilog 構文

BitBasher ブロックでサポートされる **Verilog** 論理文は、スライスや連結、反復などのビット単位の操作のみで、すべて次のテンプレート文を使用する必要があります。

```
output_var = {bitbasher_expr}
```

bitbasher_expr : Verilog 構文に基づいたスライス、連結、反復文、または単に入力ポートの識別子。

output_var : 出力ポートの識別子。 **output_var** という名前の出力ポートがブロックに表示され、**bitbasher_expr** の結果が維持されます。

連結文

```
output_var = {bitbasher_expr1, bitbasher_expr2, bitbasher_expr3}
```

前述したように、連結文はサポートされます。**bitbasher_exprN** は、それぞれ論理文、または単に入力ポートの識別子のいずれかになります。

次は、具体例です。

```
a1 = {b,c,d,e,f,g}
a2 = {e}
a3 = {b,{f,c,d},e}
```

スライス文

```
output_var = {port_identifier[bound1:bound2]}...(1)
output_var = {port_identifier[bitN]}...(2)
```

port_identifier: ビットが抽出される入力ポート

bound1、bound2: 0 ～ (port_identifier のビット幅 - 1) の負ではない整数

bitN: 0 ～ (port_identifier のビット幅 - 1) の負ではない整数

上記に示すとおり、入力ポートからビットを抽出するには方法が 2 つあります。連続するビットの範囲が抽出される必要のある場合、次の式を使用します。

```
output_var = {port_identifier[bound1:bound2]}...(1)
```

抽出されるのが 1 ビットのみの場合は、次の式を使用します。

```
output_var = {port_identifier[bitN]}...(2)
```

次は、具体例です。

```
a1 = {b[7:3]}
```

a1 には、b 入力の 7 ビット ～ 3 ビットまでが同じ順序で保持され、それらが b ビットに出力されます。たとえば、b が 110110110 の場合、a1 は 10110 になります。

```
a2 = {b[3:7]}
```

a2 には、b 入力の 7 ビット ～ 3 ビットまでが逆の順序で保持され、それらが b ビットに出力されます。たとえば、b が 110100110 の場合、a2 は 00101 になります。

```
a3 = {b[5]}
```

a3 は、b 入力の 5 ビットを保持します。

```
a4 = {b[7:5], c[3:9], {d, e}}
```

この式は、スライス文と連結文を組み合わせたものです。b 入力の 7 ～ 5 ビット、c 入力の 3 ～ 9 ビット、d 入力と e 入力のビットがすべて連結されます。

反復文

```
output_var = {N{bitbasher_expr}}
```

N: 反復回数を表す正の整数。

次は、具体例です。

```
a1 = {4{b[7:3]}}
```

上記の式は、a1 = {b[7:3], b[7:3], b[7:3], b[7:3]} と同じです。

```
a2 = {b[7:3], 2{c, d}}
```

上記の式は、a2 = {b[7:3], c, d, c, d} と同じです。

定数文

2 進定数: N'bbin_const

8 進定数: N'oocal_const

10 進定数: N'doactal_const

16 進定数: N'hocal_const

N: 定数を表すのに使用されるビット数を示す正の整数

bin_const: 0 と 1 で表される 2 進数の文字列

octal_const: 0、1、2、3、4、5、6、7 で表される 8 進数の文字列

decimal_const: 0、1、2、3、4、5、6、7、8、9 で表される 10 進数の文字列

hexadecimal_const: 0、1、2、3、4、5、6、7、8、9、a、b、c、d、e、f で表される 16 進の文字列

定数は、入力ポートから派生した追加論理式にのみ使用できます。つまり、**BitBasher** ブロックでは、**Constant** ブロックのようなブロックからの定数をソースとして使用することはできません。

次は、具体例です。

```
a1 = {4'b1100, e}
```

e が 110110110 の場合、a1 は 1100110110110 になります。

```
a1 = {4'hb, e}
```

e が 110110110 の場合、a1 は 1101110110110 になります。

```
a1 = {4'o10, e}
```

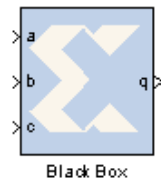
e が 110110110 の場合、a1 は 1000110110110 になります。

制限

- [BitBasher Expression] フィールドには、マスクされたパラメータは指定できません。
- 論理文に定数だけを含めることはできません。必ず少なくとも入力ポートを 1 つ含める必要があります。

Black Box

このブロックは、[Xilinx Blockset] の [Basic Elements]、[Control Logic]、[Index] ライブラリにリストされています。



System Generator の Black Box ブロックでは、HDL モデルを System Generator 用に変換できます。

このブロックでは、System Generator でコード生成中に、Simulink のシミュレーション ビヘイビアとインプリメンテーション ファイルの両方が使用されるように指定できます。ブラック ボックスのポートは、ほかの System Generator ブロックと同類の信号を入出力します。ブラック ボックスがハードウェアに変換されると、関連する HDL のエンティティが自動的に取り込まれ、ほかのブロックに接続されます。

ブラック ボックスは、VHDL または Verilog を Simulink モデルに変換するために使用できます。このブラック ボックスの HDL は、ISE Simulator か Model Technology 社の ModelSim シミュレーション ソフトウェアのどちらかの System Generator インターフェイスを使用した Simulink で協調シミュレーションされます。詳細は、ModelSim ブロックおよび「HDL 協調シミュレーション」を参照してください。

ブラック ボックスは、HDL を System Generator モデルに変換するだけでなく、外部シミュレーション モデルに関連付けられたインプリメンテーションを定義するためにも使用できます (例: ハードウェア協調シミュレーション ブロック)。System Generator には、ブラック ボックスの機能や使用方法を示す例 (ブラック ボックスの例) も含まれています。

ブラック ボックスの HDL 要件

ブラック ボックスに関連する HDL コンポーネントは、すべての次の System Generator の要件や命名規則に従っている必要があります。

- エンティティ名は、xlfir や xlregister のような System Generator の予約されているエンティティ名と重ならないようにします。
- 双方向ポートは HDL ブラック ボックスでサポートされますが、System Generator ではポートとして表示されず、ネットリストの後に生成された HDL でのみ表示されます。詳細は、「ブラック ボックスのコンフィギュレーション M 関数」を参照してください。
- たとえば、std_logic_vector(0 to 7) ではなく std_logic_vector(7 downto 0) のように、最上位レベルのポートは、MSB から LSB の順に記述する必要があります。
- Verilog のブラック ボックスの場合、モジュール名やポート名は必ず小文字にし、標準的な VHDL の命名規則に従うようにします。
- クロック ポートやクロック イネーブル ポートは、下記の規則に従って命名する必要があります。
- クロック ポートまたはクロック イネーブル ポートのタイプは必ず std_logic にします。Verilog のブラック ボックスの場合、このようなポートはベクタのない入力にします (例: input clk)。
- ブラック ボックスのクロック ポートとクロック イネーブル ポートの取り扱いは、ほかのポートと異なります。ブラック ボックスがハードウェアに変換されると、クロック ポートとクロック イネーブル ポートが入力される信号で駆動されます。信号のレートは、ブロックの設定と Simulink に送られるサンプル レートに従って指定できます。
- 立ち下がりエッジでトリガされる出力データは使用できません。

ブラック ボックスのクロックの動作を理解するには、**System Generator** での一般的な**タイミングとクロック**の処理を参照してください。**System Generator** では、ハードウェアで複数のクロック レートを生成するために、1 つのクロックに対して複数のクロック イネーブル (1 つのレートに 1 つのイネーブル) が使用されます。このクロック イネーブルが、最適なタイミングでハードウェアの異なる部分をそれぞれ有効にします。各クロック イネーブルのレートは、**Simulink** のサンプル周期によって異なります。クロックの必要な **System Generator** ブロックには、**HDL** に少なくともクロックとクロック イネーブル ポートが 1 つずつ記述されます。クロック レートが複数あるブロックの場合は、さらに多くのクロックおよびクロック イネーブル ポートが含まれます。

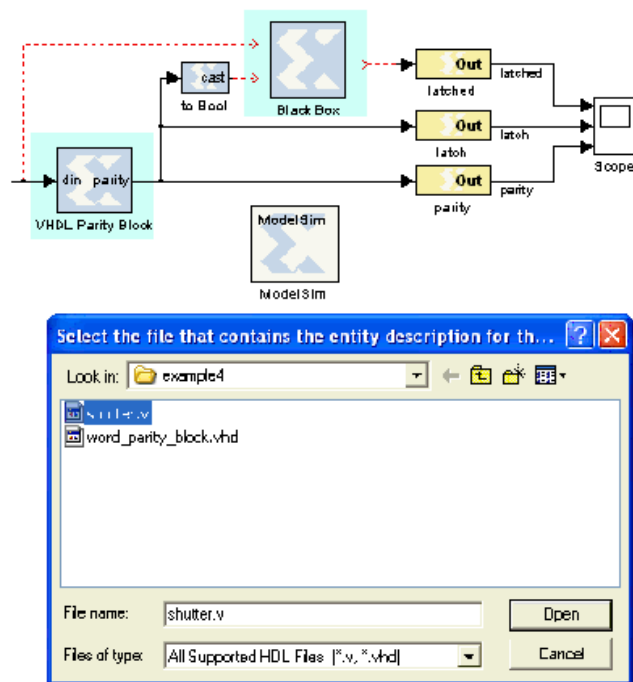
ブラック ボックスのクロックは、ほかの **System Generator** ブロックのクロックと同じように動作します。ブラック ボックスの **HDL** には、**Simulink** のサンプルレートごとに別々のクロック ポートとクロック イネーブルを含める必要があります。ブラック ボックスの **HDL** のクロック ポートとクロック イネーブル ポートは、次のように記述します。

- クロックとクロック イネーブルはペアで記述する必要があります。つまり、クロックごとに対応するクロック イネーブルが必要です。ブラック ボックスにクロック ポートが 1 つ以上含まれることもありますが、各クロック ポートを駆動するために使用されるクロック ソースは 1 つだけです。異なるのは、クロック イネーブルのレートだけです。
- クロック名には **clk** を、クロック イネーブル名には **ce** を含めます。
- クロック イネーブルの名前は、クロックと同じにする必要がありますが、**clk** の部分は **ce** にします。たとえば、クロック名が **src_clk_1** の場合、クロック イネーブルは **src_ce_1** にする必要があります。

クロック ポートとクロック イネーブルポートは、**Black Box** ブロックのアイコンには表示されません。最上位レベルの **HDL** のクロック イネーブル ポートを **System Generator** で表示させるには、別のイネーブル ポートを最上位レベルの **HDL** に追加し、この信号とクロック イネーブル信号が **AND** ゲートを通るように設定します。

Black Box ブロックのコンフィギュレーション ウィザード

このウィザードを使用すると、簡単に Verilog または VHDL コンポーネントをブラック ボックスに関連付けできます。起動するには、ブラック ボックスをモデルに追加します。ウィザードを使用するには、ブラック ボックスの HDL コンポーネントを定義するファイルをモデルが含まれるディレクトリにコピーする必要があります。新しいブラック ボックスがモデルに追加されると、次の図のようにウィザードが自動的に開きます。



このウィザードで、ブラック ボックスに関連付ける HDL ファイルを選択し、[開く] をクリックすると、次に説明するようなブラック ボックスのコンフィギュレーション M 関数が生成され、ブロックと関連付けられます。ウィザードで生成されるコンフィギュレーション M 関数は、通常は変更せずにそのまま使用できますが、手動で修正を入れる必要のあることもあります。修正が必要かどうかは、HDL がどれだけ複雑化によって異なります。

Black Box ブロックのコンフィギュレーション M 関数

ブラック ボックスでは、そのインターフェイス (ポート、ジェネリックなど) とインプリメンテーションを記述する必要があります。これは MATLAB の M 関数 (または P ファンクション) の定義 (ブロックのコンフィギュレーション) を使用して記述されます。この関数名は、ブロック パラメータ ダイアログ ボックスの [Block configuration m-function] で指定する必要があります。

コンフィギュレーション M 関数では、次が実行されます。

- ブラック ボックスに関連付ける HDL コンポーネントの最上位レベルのエンティティ名を指定します。
- HDL 言語を VHDL または Verilog に指定します。
- ポートのタイプ、方向、ビット幅、2 進小数点、名前、サンプル レートを記述します。ポートは、スタティックまたはダイナミックに指定できます。スタティック ポートは変更せず、ダイナミック ポートはデザインに変更があると変更されます。たとえば、ダイナミック ポートを駆動する信号が変わると、幅とタイプも変わります。

- 必要なポート タイプとデータ レート チェックを定義します。
- ブラック ボックスの HDL で必要とされるジェネリックをすべて定義します。
- ブラック ボックスの HDL と関連付ける EDIF などのほかのファイルを指定します。
- ブロックのクロックとクロック イネーブルを定義します (次のクロックの命名規則を参照してください)。
- HDL に組み合わせフィードスルー パスを含めるかどうかを宣言します。

System Generator には、ブラック ボックスをコンフィギュレーションするオブジェクト ベースのインターフェイスが 2 タイプ含まれます。1 つは SysgenBlockDescriptors で、エンティティ特性の定義に使用され、もう 1 つは SysgenPortDescriptors で、ポート特性の定義に使用されます。このインターフェイスは、コンフィギュレーション M 関数で System Generator にブロックのインターフェイス、シミュレーション モデル、インプリメンテーションについての方法を提供するために使用されます。

ブラック ボックスの HDL に組み合わせパス (入力から出力ポートまで直接接続されたフィードスルー パス) が少なくとも 1 つ含まれている場合は、コンフィギュレーション M 関数で tagAsCombinational を使用してそのブロックに組み合わせタグをつけます。ブラック ボックスでは、一部のパスだけを組み合わせパスにすることができます。組み合わせパスを含むブロックには、必ずこのタグを付けてください。これにより、System Generator でこのようなブロックが識別され、Simulink シミュレータに伝えられるようになります。タグを使用しないと、正確なシミュレーション結果が出ません。

ブラック ボックスのコンフィギュレーション M 関数は、モデルがコンパイルされたときに何度か実行されます。この関数には、通常ブロックの入力ポートによって異なるコードが含まれます。たとえば、入力ポートの属性によって、出力ポートのデータ型やレートを設定する必要のあることもあれば、入力ポートのデータ型とレートを確認する必要のあることもあります。関数が実行されるときに、実行されるコードを Simulink が完全に認識しないこともあります。

情報が不明な場合 (特に、例外などの条件が不明な場合) は、BlockDescriptor の inputTypesKnown および inputRatesKnown を使用して問題を回避します。これらは、それぞれ入力ポート タイプとレートの情報を提供します。これらを使用したコード例は、次のようになります。

```
if (this_block.inputTypesKnown)
% set dynamic output port types
% set generics that depend on input port types
% check types of input ports
end
```

入力レートがすべてわかっている場合、このコードではダイナミック出力ポートのタイプが設定されるほか、入力ポートのタイプによってジェネリックが設定され、入力タイプが適しているかどうかを検証されます。変数定義など、これらの条件ブロックの外に記述する必要のあるコードは、中に含めないように注意してください。

上記のコードでは、this_block というオブジェクト名が使用されています。ブラック ボックスのコンフィギュレーション M 関数では、入力引数を使用し、this_block が自動的に使用できるようになっています。MATLAB では、this_block はブラック ボックスを表すオブジェクトで、ブラック ボックスをテストして設定するために、コンフィギュレーション M 関数内で使用されます。this_block オブジェクトは、MATLAB クラスの「SysgenBlockDescriptor」のインスタンスです。this_block への適用方法は、付録 A を参照してください。サンプルのコンフィギュレーション M 関数を生成するには、単純な VHDL エンティティでコンフィギュレーション ウィザードを実行します (詳細は後述)。

ブラック ボックスのコンフィギュレーション オプションについては、[「ブラック ボックスの例」](#)を参照してください。

サンプル周期

ブラック ボックスの出力ポート、クロック、クロック イネーブルは、コンフィギュレーション **M** 関数のサンプル周期で割り当てする必要があります。サンプル周期がダイナミックであるか、ブラック ボックスでレートをチェックする必要がある場合、この関数で入力ポートのサンプル周期を取得する必要があります。ブラック ボックスのサンプル周期は、**System Generator** のマスタ ブロックの **[Simulink system period]** フィールドで指定したシステム レートの整数倍で記述します。たとえば、**[Simulink system period]** が $1/8$ の場合、ブラック ボックスの入力ポートはそのシステム レート ($1/8$) で実行され、コンフィギュレーション **M** 関数でポートのレートとしてレポートされた **1** が認識されます。同様に、**[Simulink system period]** が π の場合、出力ポートはそのシステム レートの 4 倍の速さ (4π) にする必要がありますので、コンフィギュレーション **M** 関数で出力ポートのレートを 4 に設定する必要があります。定数ポートに最適なレートは **Inf** です。

各出力ポートの出力レートを設定する方法については、次のコード例を参照してください。

```
block.outport(1).setRate(theInputRate);  
block.outport(2).setRate(theInputRate*5);  
block.outport(3).setRate(theInputRate*5);
```

最初の行では、出力ポートを入力ポートと同じレートに設定し、次の 2 行では、出力レートを入力ポートの 5 倍に設定しています。

ブロック パラメータ

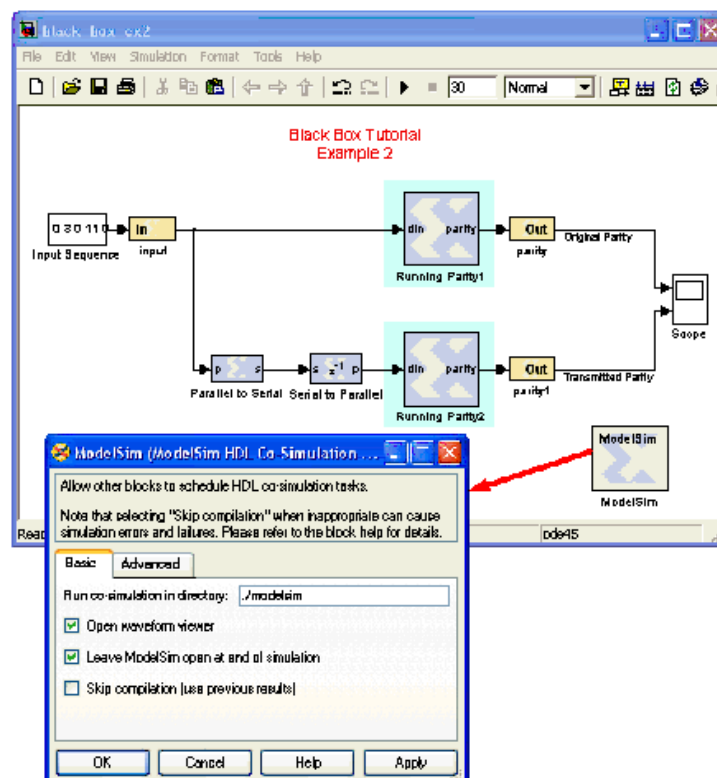
ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- **[Block configuration m-function]** : ブラック ボックスに関連付けたコンフィギュレーション M 関数の名前を指定します。この関数を含むファイルは、通常モデルを含むディレクトリに保存されますが、MATLAB パスのいずれかに保存することもできます。MATLAB では、コンフィギュレーション M 関数を含め、関数名はすべて 63 文字以内にする必要があります。このテキスト ボックスには、ファイルの拡張子 (.m または .p) は含めないでください。
- **[Simulation mode]** : シミュレーション モード (Inactive、ISE Simulator、External co-simulator) を指定します。モードが [Inactive] の場合、ブラック ボックスはすべての入力データを無視し、出力ポートに 0 を出力します。このモードの場合は、「[コンフィギャブル サブシステムと System Generator](#)」の説明にあるコンフィギャブル サブシステムを使用して、ブラック ボックスをカップリングする必要があります。

System Generator でコンフィギャブル サブシステムを使用すると、シミュレーション結果用とハードウェア用の 2 つのパスを同じにできます。この方法では、シミュレーション速度は最速になりますが、シミュレーション モデルを構築する必要があります。モードを [ISE Simulator] または [External co-simulator] にすると、ブラック ボックスのシミュレーション結果はブラック ボックスに関連付けられた HDL で協調シミュレーションを使用して作成されます。[External co-simulator] を使用する場合は、ModelSim の HDL 協調シミュレーションブロックをデザインに追加し、[HDL co-simulator to use] フィールドでその ModelSim ブロックの名前を指定する必要があります。次は、その例です。



System Generator では、HDL 協調シミュレーション用に Mentor Graphics 社の ModelSim シミュレータがサポートされます。Verilog のブラック ボックスの協調シミュレーションには、混合モードのライセンスが必要です。これは、System Generator で記述されるデザイン部分が VHDL だからです。

ブラック ボックスの協調シミュレータ ブロックは、通常はブラック ボックスと同じサブシステムに保存されますが、どこにでも保存可能です。協調シミュレーション ブロックへのパスは、絶対パスにも、ブラック ボックスを含むサブシステムへの相対パス (例: ../ModelSim) にもできます。シミュレーション中は、協調シミュレータ ブロックごとにライセンスが 1 つ使用されます。複数のブラック ボックスで同じ協調シミュレーション ブロックを共有しておくと、ライセンスが不足する問題を回避できます。System Generator では、複数のブロックを 1 つの ModelSim シミュレーションにまとめるのに必要な VHDL が自動的に生成され、追加されます。

HDL 協調シミュレーションのためのデータ型変換

協調シミュレーション中は、System Generator のポートが HDL シミュレータのポートを駆動したり、HDL シミュレータのポートが System Generator のポートを駆動しますが、これらのツールの信号のデータ型は同一ではないので、変換する必要があります。変換に使用される規則は、次のとおりです。

- System Generator の信号はブール型、符号なし固定小数点、または符号付き固定小数点のいずれかにできます。固定小数点の信号の値は決定する必要はありませんが、ブール信号は未決定にはできません。System Generator で決定しなかった場合は、HDL 信号のビットすべてが X になります。これ以外の場合は、その信号の値を表す 0 と 1 を使用した値になります。
- HDL 信号を System Generator に戻す場合は、ブラック ボックスのコンフィギュレーション M 関数で指定したとおりに、標準ロジック タイプがブール型と固定小数点型の値に変換されます。幅が違う場合は、エラー メッセージが表示されます。不確定な信号 (Weak High、Weak Low など) はすべて System Generator の不確定値に変換されます。HDL シミュレーションで一部だけ不確定な信号 (一番上のビットだけが不確定なビット ベクタなど) は、System Generator ではすべて不確定になります。
- HDL から System Generator への変換は、カスタムのシミュレーション専用の最上位レベル ラップを VHDL に追加すると詳細に設定できます。ラップは、たとえば Weak Low 信号すべてを 0 に変換したり、不確定な信号すべてを 0 か 1 にしたりしてから、System Generator に戻します。

例

次は、System Generator ブラック ボックスに関連付けが可能な VHDL エンティティの例です (この例は、ブラック ボックスの例：[VHDL モジュールのインポート](#)からのものです)。

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity word_parity_block is
    generic (width : integer := 8);
port (din : in std_logic_vector(width-1 downto 0);
      parity : out std_logic);
end word_parity_block;
architecture behavior of word_parity_block is
begin
    WORD_PARITY_Process : process (din)
        variable partial_parity : std_logic := '0';
    begin
        partial_parity := '0';
        XOR_BIT_LOOP: for N in din'range loop
            partial_parity := partial_parity xor din(N);
        end loop; -- N
        parity <= partial_parity after 1 ns ;
    end process WORD_PARITY_Process;
end behavior;
```

次は、コンフィギュレーション M 関数の例です。上記の VHDL を System Generator ブラック ボックス内で使用可能にします。

```
function word_parity_block_config(this_block)
this_block.setTopLevelLanguage('VHDL');
    this_block.setEntityName('word_parity_block');
    this_block.tagAsCombinational;
    this_block.addSimulinkInport('din');
    this_block.addSimulinkOutport('parity');
    parity = this_block.port('parity');
    parity.setWidth(1);
    parity.useHDLVector(false);
    % -----
    if (this_block.inputTypesKnown)
        this_block.addGeneric('width',
            this_block.port('din').width);
    end % if(inputTypesKnown)
    % -----
    % -----
    if (this_block.inputRatesKnown)
        din = this_block.port('din');
        parity.setRate(din.rate);
    end % if(inputRatesKnown)
    % -----
    this_block.addFile('word_parity_block.vhd');
    return;
```

関連項目

[HDL モジュールのインポート](#)

ChipScope

このブロックは、[Xilinx Blockset] の [Tools] および [Index] ライブラリにリストされています。



ザイリンクス ChipScop™ ブロックでは、ランタイム デバッグおよび FPGA 内の信号検証などを実行できます。

キャプチャ メモリは大容量で、トリガ オプションが複数提供されています。データは、ユーザー定義のトリガ条件に基づいてキャプチャされ、内部ブロック メモリに保存されます。

ChipScope ブロックには、ChipScope Pro Analyzer を使用してランタイム時にアクセスできます。ChipScope Pro Analyzer は、FPGA のコンフィギュレーション、トリガ条件の設定、ランタイム時にキャプチャされたデータの表示に使用します。 制御信号およびデータ信号はすべて JTAG ポートを介して転送されるので、I/O ピンを使用してデータをオフチップに駆動する必要はありません。データは、ChipScope Pro Analyzer からエクスポートして、MATLAB ワークスペースに読み込み直すことができます。

ハードウェアとソフトウェア条件

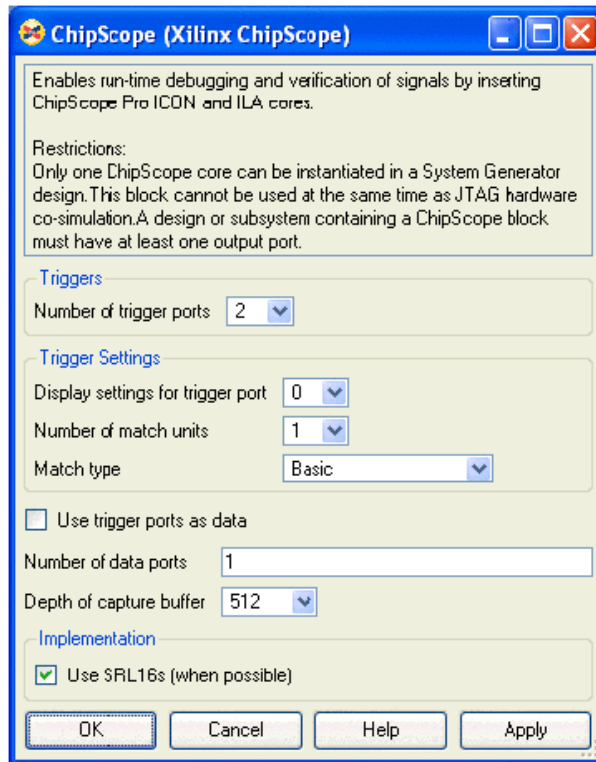
ChipScope Pro ソフトウェア (このブロックを使用するためのソフトウェア インストール情報は、「ソフトウェア要件」から入手可能)、ダウンロード ケーブル、JTAG コネクタ付きの FPGA ボードが必要です。ChipScope Pro の購入については、http://japan.xilinx.com/ise/optional_prod/cspro.htm を参照してください。

ChipScope Pro Analyzer では、PC と JTAG バウンダリ スキャン チェーンのデバイス間の通信用に次のダウンロード ケーブルがサポートされます。

- パラレル ケーブル III
- パラレル ケーブル IV
- MultiLINX (JTAG モードのみ)
- Agilent E5904B Option 500、FPGA Trace Port Analyzer (Agilent E5904B TPA)

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。



このブロックでは、次のパラメータが設定できます。

- **[Number of trigger ports]** : 複数のトリガ ポートを使用すると、検出されるイベントの範囲が拡大でき、格納されるデータ容量を削減できます。選択できるトリガ ポートは、16 個までです。トリガ ポートの番号は 0 から開始され、Trig0、Trig1 というように名前が付きます。デフォルトは TrigN-1 です。トリガ ポートの名前は、ポートに接続された信号で指定すると変更できます。
- **[Display settings for trigger port]** : 各トリガ ポートに対し、一致ユニットの数と、一致タイプを設定する必要があります。このオプションでは、表示オプションを設定するトリガ ポートをドロップダウン リストから指定します。N 個のポートに対し、トリガ ポート 0 ～ N-1 のトリガポートの表示オプションが表示されます。
- **[Number of match units]** : トリガ ポートごとに複数の一致ユニットを使用すると、イベント検出の柔軟性が向上します。トリガ イベントをテストするため、1 ～ 4 つの一致ユニットを使用できます。トリガ値は、ChipScope Pro Analyzer の実行時に設定します。
- **[Match type]** : このオプションは、次の 6 つのいずれかに設定できます。
 - a. **[Basic]** : = または <> 比較を実行します。
 - b. **[Basic with edges]** : [Basic] の操作に加え、High から Low、Low から High への遷移も検出します。
 - c. **[Extended]** : = または <>、>、<、<=、>= 比較を実行します。
 - d. **[Extended with edges]** : [Extended] の操作に加え、High から Low、Low から High への遷移も検出します。

- e. [Rnage]: =、<>、>、>=、<、<=、範囲内、範囲外比較を実行します。
- f. [Range with edges]: [Range] の操作に加え、High から Low、Low から High への遷移も検出します。

メモ: [Basic] が最もエリア効率が高く、各 FPGA スライスで 8 ビット比較できます。[Basic with edges] では、各スライスで 4 ビット、[Extended] と [Extended with edges] では 2 ビット、[Range] と [Range with edges] では 1 ビット比較できます。

- [Use trigger ports as data]: オンにすると、データ ポートとトリガ ポートが同一になり、trig0/data0、trig1/data1、... trigN-1/dataN-1 という名前になります (N はトリガ ポート数を表します)。このモードは、ChipScope ブロックでキャプチャおよび収集されるデータをイネーブルにするため、ほとんどのロジック アナライザで使用されています。このモードを使用すると、キャプチャされるデータ量が制限されるので、ハードウェア リソースが節約できます。

オフにすると、データ ポートはトリガ ポートから完全に独立します。トリガ ポートの名前は、trig0、trig1、...trigN-1、データ ポートの名前は data0、data1、...dataN-1 になります。ポートの名前は、ポートに接続される信号で指定すると変更できます。

- [Number of data ports]: サンプルごとにデータ 256 ビットまでをキャプチャできます。つまり、データ ポート数とポートごとのビット数を乗算した値は 256 ビット以下である必要があります。System Generator ではデータ幅が自動的に伝搬されるので、データ ポートの数のみを指定する必要があります。
- [Depth of capture buffer]: このオプションは、2 のべき乗で設定します。

ChipScope プロジェクト ファイル

System Generator では、ブロックに接続されたデータ信号をバスにグループ化するために ChipScope のプロジェクト ファイルが作成されます。バスは各データ ポートに対して作成されるので、ChipScope Pro Analyzer の [Bus Plot] ウィンドウでアナログ波形として表示できます。各データ バスは、Simulink モデルで使用される 2 進小数点に基づいて測定されます。ChipScope ブロックに接続された信号に名前を付けると、それが ChipScope プロジェクト ファイルでバス名として使用されます。

プロジェクトは、[File] → [Import] → [Select New File] をクリックしてプロジェクト ファイルを選択すると、ChipScope Pro Analyzer に読み込むことができます。プロジェクトは、<block name>.cdc として保存されます。<block name> は、モデルのターゲット ディレクトリでコンパイルされる Chipscope ブロックの名前です。

ChipScope から MATLAB ワークスペースへのデータのインポート

ChipScope Pro Analyzer からデータをエクスポートするには、まず [Bus Plot] ウィンドウでエクスポートするバスを選択します。[File] → [Export option] で [ASCII] と [Bus Plot Buses] を選択します。[Export] をクリックし、ファイルを .prn という拡張子で保存します。MATLAB で現在の作業ディレクトリを PRN ファイルを保存したディレクトリに変更し、次をタイプします。

```
xlLoadChipScopeData('<your file name>.prn');
```

このコマンドを実行すると、PRN ファイルからのデータが MATLAB ワークスペースに読み込まれます。新しいワークスペースの変数の名前は ChipScope ブロックのポート名になります。ChipScope ブロックに接続された信号に名前を付けると、それが MATLAB ワークスペースの変数として使用されます。信号名を付けなかった場合、ポート名は [Use trigger ports as data] オプションの設定によって異なります。オンの場合は、デフォルトで trig0_data0、trig1_data1、... trigN-1_dataN-1 という名前になり、オフの場合は、デフォルトで data0、data1、... dataN という名前になります。

既知の問題

- このブロックを使用するためのソフトウェア インストール情報は、「[ソフトウェア要件](#)」を参照してください。
- System Generator デザインにインスタンス化できる ChipScope コアは 1 つだけです。Simulink の Goto ブロックと From ブロックを使用すると、信号を簡単に ChipScope ブロックに配線できます。
- ChipScope ブロックと JTAG ハードウェア協調シミュレーションは両方とも JTAG ポートを使用するため、同時に使用できません。
- ChipScope ブロックを含むデザインまたはサブシステムには、出力ポートが最低 1 つは必要です。出力ポートがない場合、ChipScope ブロックは VHDL 合成中に最適化されて削除されます。

詳細情報

ChipScope Pro ソフトウェアの詳細は、<http://japan.xilinx.com/chipscope> を参照してください。

このブロックの使用方法を詳細に示したチュートリアルについては、「[ChipScope Pro Analyzer を使用したリアルタイム ハードウェア デバッグ](#)」を参照してください。

CIC Compiler 1.2

このブロックは、[Xilinx Blockset] の [DSP] および [Index] ライブラリにリストされています。



ザイリンクスの CIC Compiler には、さまざまなザイリンクス FPGA デバイスに対して CIC (Cascaded Integrator-Comb) フィルタをデザインおよびインプリメントする機能があります。

CIC フィルタ (別名 Hogenauer フィルタ) はマルチレート フィルタで、デジタルシステムの大規模なサンプル レートの変更をインプリメントするためによく使用されます。通常は、余剰のサンプル レートが大量にあるようなアプリケーションで使用されるので、デジタル ダウン コンバータ (DDC) およびデジタル アップ コンバータ (DUC) と同様、処理された信号で占められるバンド幅よりもシステム レートがかなり大きくなります。CIC フィルタのインプリメンテーションには、加算器、減算器、遅延エレメントのみが使用されます。このため、CIC フィルタがマルチレート フィルタをハードウェア効率良くインプリメンテーションされます。

ブロック パラメータのダイアログ ボックス

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

フィルタの指定

- **[Filter type]** : このコアでは、補間フィルタと分割フィルタのいずれのファンクションもサポートされています。フィルタ タイプを分割フィルタにすると、入力サンプル ストリームが係数 R でダウン サンプリングされ、補間フィルタにすると、係数 R でアップ サンプリングされます。
- **[Number of Stages]** : 補間およびくし型のステージ数。 N ステージに指定すると、フィルタには N 個の積分器と N 個のくし型ステージが含まれます。指定できる値は 3 ~ 6 です。
- **[Differential delay]** : 分割フィルタと補間フィルタのいずれかのくし型セクションにある各くし型フィルタで使用されるユニット遅延の数。指定できる値は 1 または 2 です。
- **[Number of channels]** : インプリメンテーションでサポートされるチャネル数。指定できる値は 1 ~ 16 です。

精度の指定

- **[Input data width]** : 2 ~ 20 ビットに指定できます。
- **[Output data width]** : 最大 48 ビットまで指定できます。

サンプル レートの変更

- **[Sample rate change]** : [Fixed] または [Programmable] のいずれかを選択できます。
- **[Fixed or initial rate(ir)]** : CIC フィルタの最初の、または固定されたサンプル レートの変更値を指定します。指定できる値は 4 ~ 8192 です。
- **[Minimum rate (Range: 4..ir)]** : プログラマブル レート変更用の最小レート。指定できる値は 4 ~ 固定レート (ir) です。
- **[Maximum rate (Range: ir..8192)]** : プログラマブル レート変更用の最大レート。指定できる値は 固定レート (ir) ~ 8192 です。

メモ : CIC Compiler ブロックは入力をサンプリングし、Simulink システム周期で出力するように設定されています。[Decimation] (間引き) の場合、RDY 出力ポートを使用して dout 信号がレジスタに取り込まれ、その後、Down Sample ブロックに取り込まれます。このブロック

では、サンプリング レートがストリーミング入力用に固定 (Fixed) サンプル レートに設定されます。Simulink システム周期が補間レートと同じの [Interpolation] (補間) を使用する場合、ND と RFD 信号を組み合わせ、より遅いレートでの CIC Compiler への入力に使用してください。その他すべての固定 (Fixed) およびプログラマブル (Programmable) レートの補間と間引き CIC フィルタでは、ND、RFD、RDY ポートを Down Sample および Up Sample ブロックと合わせて使用し、CIC Compiler へのデータの入出力を管理します。ND、RFD、RDY 信号の詳細については、CIC Compiler の LogiCORE データシートを参照してください。

オプション ポート

- **CE** : クロック イネーブル - コア クロック イネーブル (アクティブ High)。この信号が High になると、フィルタは入力データを通常どおり処理します。Low の場合は、フィルタはデータの処理を停止し、その状態を保ちます。
- **SCLR** : 同期クリア - 同期リセット (アクティブ High)。SCLR を CLK と同時にアサートすると、フィルタの内部ステートがリセットされます。

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

[Implementation] タブ

- [Use Xtreme DSP slice] : このフィールドでは、可能であればターゲット デバイスで XtremeDSP スライス (DSP48 タイプのエレメント) を使用するように指定できます。

メモ : このブロックへのインターフェイスがある場合、CIC Compiler v1.2 のデータシート (LogiCORE 製品仕様) の「Interface, Control, and Timing」のガイドラインに必ず従ってください。

ザイリンクス LogiCORE

このブロックは、次のザイリンクス LogiCORE コアを使用します。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex			
			3、 3E	3A	3A DSP	6	6 -1L	4	5	6	6 -1L
CIC Compiler 1.2	CIC Compiler	V1.2	•	•	•			•	•		

CIC Compiler 1.3

このブロックは、[Xilinx Blockset] の [DSP] および [Index] ライブラリにリストされています。



ザイリンクスの CIC Compiler には、さまざまなザイリンクス FPGA デバイスに対して CIC (Cascaded Integrator-Comb) フィルタをデザインおよびインプリメントする機能があります。

CIC フィルタ (別名 Hogenauer フィルタ) はマルチレート フィルタで、デジタルシステムの大規模なサンプル レートの変更をインプリメントするためによく使用されます。通常は、余剰のサンプル レートが大量にあるようなアプリケーションで使用されるので、デジタル ダウン コンバータ (DDC) およびデジタル アップ コンバータ (DUC) と同様、処理された信号で占められるバンド幅よりもシステム レートがかなり大きくなります。CIC フィルタのインプリメンテーションには、加算器、減算器、遅延エレメントのみが使用されます。このため、CIC フィルタがマルチレート フィルタをハードウェア効率良くインプリメンテーションされます。

ブロック パラメータ ダイアログ ボックス

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

フィルタの指定

- **[Filter type]** : このコアでは、補間フィルタと分割フィルタのいずれのファンクションもサポートされています。フィルタ タイプを分割フィルタにすると、入力サンプル ストリームが係数 R でダウン サンプリングされ、補間フィルタにすると、係数 R でアップ サンプリングされます。
- **[Number of Stages]** : 補間およびくし型のステージ数。N ステージに指定すると、フィルタには N 個の積分器と N 個のくし型ステージが含まれます。指定できる値は 3 ~ 6 です。
- **[Differential delay]** : 分割フィルタと補間フィルタのいずれかのくし型セクションにある各くし型フィルタで使用されるユニット遅延の数。指定できる値は 1 または 2 です。
- **[Number of channels]** : インプリメンテーションでサポートされるチャネル数。指定できる値は 1 ~ 16 です。

サンプル レートの変更指定

- **[Sample rate change]** : [Fixed] または [Programmable] のいずれかを選択できます。
- **[Fixed or Initial Rate(ir)]** : CIC フィルタの最初の、または固定されたサンプル レートの変更値を指定します。指定できる値は 4 ~ 8192 です。
- **[Minimum Rate (Range: 4..ir)]** : プログラマブル レート変更用の最小レート。指定できる値は 4 ~ 固定レート (ir) です。
- **[Maximum Rate (Range: ir..8192)]** : プログラマブル レート変更用の最大レート。指定できる値は 固定レート (ir) ~ 8192 です。

ハードウェア オーバーサンプリングの指定

[Select format] : [Maximum_Possible]、[Sample_Period]、または [Hardware_Oversampling_Rate] を選択します。

[Sample period] : このオプションがオンになると、nd (新規データ - アクティブ High) 入力ポートがブロックに配置されます。nd がアサートされると、din ポートのデータ サンプルがフィルタに読み込まれます。

[Hardware Oversampling Rate] : フォーマットに [Hardware_Oversampling_Rate] を選択した場合、ハードウェア オーバーサンプリング レートをここに入力します。

[Implementation] タブ

精度の指定

- [Quantization] : [Full_Precision] または [Trunction] に指定できます。
- [Output data width] : 上記の [Trunction] オプションの場合、最大 48 ビットまで指定できます。

オプション

- [Use Xtreme DSP slice] : オンにすると、可能な場合に XtremeDSP スライス (DSP48 タイプのエレメント) がターゲット デバイスで使用されます。

制御オプション

- rst (同期リセット アクティブ High)
- en (クロック イネーブル アクティブ High) ポート
- nd (新規データ - アクティブ High) この信号がアサートされると、din ポートのデータ サンプルがフィルタに読み込まれます。この制御ポートは、フォーマットに [Sample Period] を選択した場合にのみブロックに配置できます。詳細は、[Basic] タブの「ハードウェア オーバーサンプリングの指定」を参照してください。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

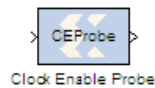
ザイリンクス LogiCORE

このブロックでは、次のザイリンクス LogiCORE コアが使用されます。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
CIC Compiler 1.3	CIC Compiler	V1.3	•	•	•	•	•	•	•	•	•	•

Clock Enable Probe

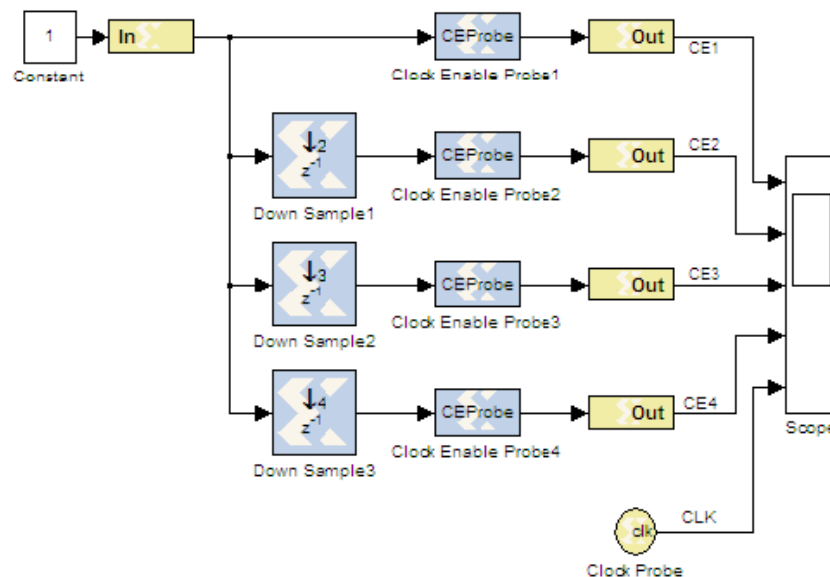
このブロックは、[Xilinx Blockset] の [Basic Elements] ライブラリと [Index] ライブラリにリストされています。

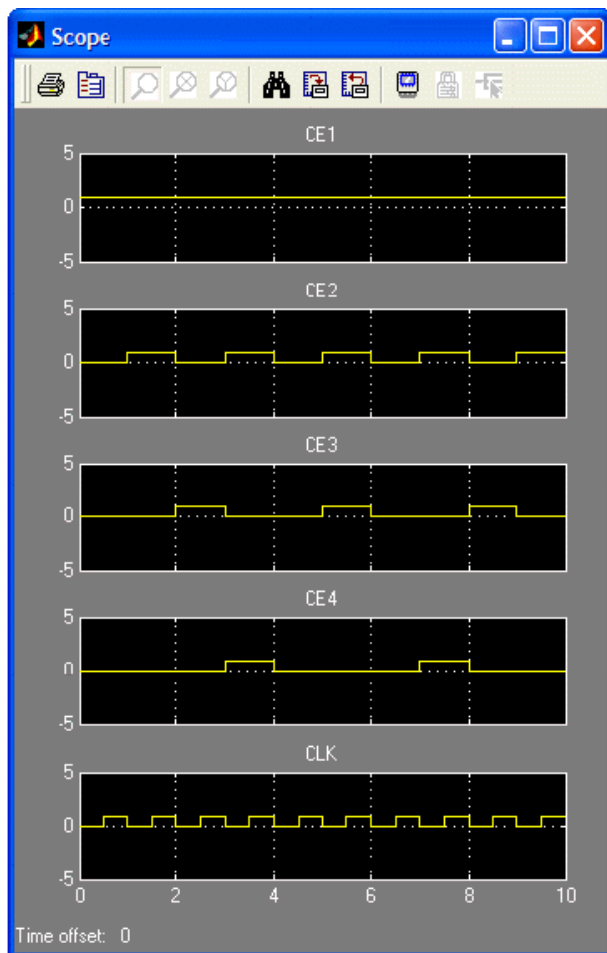


ザイリンクスの **Clock Enable Probe** ブロックでは、**System Generator** モデルのザイリンクス信号から派生したクロック イネーブル信号を抽出できます。

このブロックは、どのデータ型のザイリンクス信号も入力として受信し、ブール信号を出力します。この出力は、ブール信号が使用可能であれば、デザインのどこにでも使用できます。プローブ出力は、マルチレート回路のハードウェア インプリメンテーションで使用する理想的なクロック イネーブル信号の動作を模倣した循環パルスです。パルスの周波数は、入力信号のサンプル周期から派生します。**Simulink** の 1 サンプル周期の間、イネーブル パルスが入力信号のサンプル周期の終わりでアサートされます。**Simulink** のシステム周期と同じサンプル周期の信号の場合は、ブロックの出力は常に 1 になります。

次の図は、解析スコープを付けた **Clock Enable Probe** ブロックの使用方法和動作を示した例です。このモデルの **Simulink** システムのサンプル周期は、**System Generator** ブロックで 1.0 秒と指定されています。このモデルには、**Simulink** のシステム周期だけでなく、**Down Sample** ブロックで定義されたサンプル周期が 3 つ使用されています。**Clock Enable Probe (CEProbe)** は、各 **Down Sample** ブロックの後に配置され、派生したクロック イネーブル信号を抽出します。プローブの出力は出力ゲートウェイと解析用のスコープに送信されます。この例には、ハードウェアシステム クロックを倍精度で出力する **CLK** プローブも含まれます。スコープの出力は、この **CLK** プローブ出力と 4 つのクロック イネーブルプローブからの出力を示しています。





Clock Enable Probe ブロックには、パラメータがありません。

Clock Probe

このブロックは、[Xilinx Blockset] の [Tools] および [Index] ライブラリにリストされています。



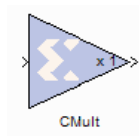
ザイリンクスの **Clock Probe** ブロックでは、倍精度のクロック信号が **Simulink** システム周期と同じ周期で出力されます。

クロック信号は、**Simulink** サンプル周期の開始時にアサートされるクロックを使用し、**50/50** のデューティ サイクルで出力されます。**Clock Probe** ブロックの倍精度出力は、解析専用なのでハードウェアには変換できません。

このブロックにはパラメータがありません。

CMult

このブロックは、[Xilinx Blockset] の [Math] および [Index] ライブラリにリストされています。



ザイリンクスの CMult ブロックは、gain 演算子をインプリメントします。出力は、入力と定数値の積になります。この値は、定数を求める MATLAB の論理式で表すことができます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Constant] : 定数または式が入ります。定数が指定された固定小数点型で正確に表すことができない場合、必要に応じて値は近似値に丸められるかサチュレート（飽和演算）が実行されます。正の値は符号なし、負の値は符号ありとしてインプリメントされます。
- [Number of bits] : 定数の 2 進小数点のビット位置を指定します。ビット 0 は LSB を表します。
- [Binary point] : 2 進小数点の位置を指定します。

[Output Type] タブ

このタブのパラメータでは、CMult ブロックの出力精度を定義します。詳細については、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」を参照してください。

[Implementation] タブ

[Implementation] タブからは、次のようなパラメータを設定できます。

- [Use behavioral HDL description (otherwise use core)] : オンにすると、ビヘイビア レベルの HDL が使用され、オフにするとザイリンクスの LogiCORE Multiplier コアが使用されます。
メモ：これがオフにすると、次のいずれかの場合に、System Generator では内部でビヘイビア HDL モデルを使ってシミュレーションされます。
 - a. 定数値が 0 である（または 0 に切り捨てられた）場合
 - b. 定数値が 0 未満で、そのビット幅が 1 の場合
 - c. 定数のビット幅または入力が 1 未満、または 64 を超える場合
 - d. 入力データのビット幅が 1 で、そのデータ型が x1Fix の場合これは、すべての Virtex および Spartan デバイス ファミリに適用されます。
- [Implement using] : 分散 RAM かブロック RAM を指定します。
- [Test for optimum pipelining] : [Basic] タブで指定した [Latency] が少なくともブロックのコンフィギュレーションでサポートされる最適なパイプライン長と等しいかどうかチェックされます。レイテンシの値がこのテストをパスした場合、コアはスピードを重視して最適化されます。

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

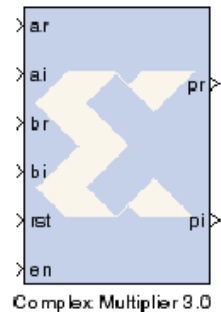
ザイリンクス LogiCORE

設定に応じて、このブロックは、次のザイリンクス LogiCORE コアを使用します。

System Generator Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
CMult	Multiplier	V11.2	•	•	•	•	•	•	•	•	•	•

Complex Multiplier 3.0

このブロックは、[Xilinx Blockset] の [DSP]、[Math] および [Index] ライブラリにリストされています。



ザイリンクスの **Complex Multiplier** ブロックは、2 つの複素数を乗算します。

すべてのオペランドおよびその結果は、符号付きの 2 の補数形式で示されます。オペランドの幅とその結果の幅は、パラメータ指定可能です。

ブロック パラメータ ダイアログ ボックス

[Page 1] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

乗算器構築オプション

- [Use_LUTs] : ファブリックで LUT を使用します。
- [Use_Mults] : エンベデッド乗算器/XtremeDSP スライスを使用します。

最適化目標

[Use_Mults] が選択されている場合にのみ使用できます。

- [Resources] : 3 実数乗数構造が使用されますが、3:1 乗算器構造でさらに多くの乗算器リソースが使用される場合は、4 実数乗数構造が使用されます。
- [Performance] : 4 実数乗数構造が常に使用され、最適な周波数パフォーマンスが達成できるようになります。

出力積の範囲

必要な出力積の **MSB** と **LSB** を選択します。この値は、**A** と **B** のオペランドの幅が設定されると、自動的に完全精度の積を提供するように設定されます。出力は、必要であれば符号拡張されます。丸めが必要な場合は、[Output LSB] の値を 0 よりも大きい値にして、丸めオプションを設定できるようにします。

[Page 2] タブ

出力の丸め

丸めが必要な場合は、[Output LSB] を 0 より大きい値にしておく必要があります。

- [Truncate] : 出力が切り捨てられます。
- [Random_Rounding] : 詳細は、Complex Multiplier 3.0 の製品仕様の「Rounding」セクションを参照してください。

オプション ポート

- **en** : クロック イネーブル - ブロックでオプションのイネーブル (**en**) ピンが有効になります。イネーブル信号がアサートされないと、イネーブル信号が再びアサートされるまで (またはリ

セット信号がアサートされるまで)、ブロックが現在のステートを保持したままになります。リセット信号はイネーブル信号よりも優先されます。イネーブル信号は、ブロックのサンプルレートの倍数のレートで実行する必要があり、イネーブルポートを駆動する信号は、ブール型にする必要があります。

- **rst** : リセット - ブロックでオプションのリセット (**rst**) ピンが有効になります。このリセット信号がアサートされると、ブロックが最初のステートに戻ります。リセット信号は、ブロックで使用可能なオプションのイネーブル信号よりも優先されます。この信号は、ブロックのサンプルレートの倍数のレートで実行する必要があり、リセットポートを駆動する信号は、ブール型にする必要があります。

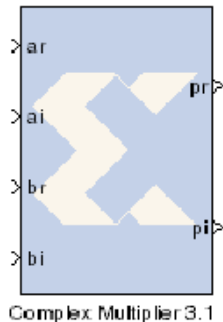
ザイリンクス LogiCORE

このブロックは、次のザイリンクス LogiCORE コアを使用します。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
Complex Multiplier 3.0	Complex Multiplier	V3.0	•	•	•	•		•	•	•	•	•

Complex Multiplier 3.1

このブロックは、[Xilinx Blockset] の [DSP]、[Math] および [Index] ライブラリにリストされています。



ザイリンクスの **Complex Multiplier** ブロックは、2 つの複素数を乗算します。

すべてのオペランドおよびその結果は、符号付きの 2 の補数形式で示されます。オペランドの幅とその結果の幅は、パラメータ指定可能です。

ブロック パラメータ ダイアログ ボックス

[Page 1] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

乗算器構築オプション

- [Use_LUTs] : ファブリックで LUT を使用します。
- [Use_Mults] : エンベデッド乗算器/XtremeDSP スライスを使用します。

最適化目標

[Use_Mults] が選択されている場合にのみ使用できます。

- [Resources] : 3 実数乗数構造が使用されますが、3:1 乗算器構造でさらに多くの乗算器リソースが使用される場合は、4 実数乗数構造が使用されます。
- [Performance] : 4 実数乗数構造が常に使用され、最適な周波数パフォーマンスが達成できるようになります。

出力積の範囲

必要な出力積の **MSB** と **LSB** を選択します。この値は、**A** と **B** のオペランドの幅が設定されると、自動的に完全精度の積を提供するように設定されます。出力は、必要であれば符号拡張されます。丸めが必要な場合は、[Output LSB] の値を 0 よりも大きい値にして、丸めオプションを設定できるようにします。

[Page 2] タブ

コア レイテンシ

ブロック レイテンシは必要に応じて調整できます。デフォルトは -1 で、下位の LogiCORE をパイプライン接続してパフォーマンスを最大にします。

出力の丸め

丸めが必要な場合は、[Output LSB] を 0 より大きい値にしておく必要があります。

- [Truncate] : 出力が切り捨てられます。

- **[Random_Rounding]** :このオプションをオンにすると、**round_cy** 入力ポートがブロックに追加され、キャリーイン ビットを入力にできます。詳細は、**Complex Multiplier 3.1** の製品仕様の「**Rounding**」セクションを参照してください。

オプション ポート

- **en** : クロック イネーブル - ブロックでオプションのイネーブル (**en**) ピンが有効になります。イネーブル信号がアサートされないと、イネーブル信号が再びアサートされるまで (またはリセット信号がアサートされるまで)、ブロックが現在のステートを保持したままになります。リセット信号はイネーブル信号よりも優先されます。 イネーブル信号は、ブロックのサンプルレートの倍数のレートで実行する必要があり、イネーブル ポートを駆動する信号は、ブール型にする必要があります。
- **rst** : リセット - ブロックでオプションのリセット (**rst**) ピンが有効になります。このリセット信号がアサートされると、ブロックが最初のステートに戻ります。リセット信号は、ブロックで使用可能なオプションのイネーブル信号よりも優先されます。この信号は、ブロックのサンプルレートの倍数のレートで実行する必要があり、リセット ポートを駆動する信号は、ブール型にする必要があります。

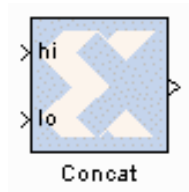
ザイリンクス LogiCORE

このブロックは、次のザイリンクス LogiCORE コアを使用します。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
Complex Multiplier 3.1	Complex Multiplier	V3.1	•	•	•	•	•	•	•	•	•	•

Concat

このブロックは、[Xilinx Blockset] の [Basic Elements]、[Data Types]、[Index] ライブラリにリストされています。



ザイリンクスの Concat ブロックでは、符号なしの整数で表される n ビット ベクタ (2 進小数点の位置が 0 にある、 n 個の符号なしの値) が連結されます。

ザイリンクスの Reinterpret ブロックでは、この Concat ブロックの機能がさらに拡張されています。

ブロック インターフェイス

このブロックには、入力ポートが n 個と出力ポートが 1 つ含まれます。 n は 2 ~ 1024 までの値になります。最初と最後の入力ポートには、それぞれ **hi** および **lo** という名前が付いています。この 2 つのポート間の入力ポートには、何も付いていません。**hi** ポートへの入力が出力の **MSB** に、**lo** ポートへの入力が出力の **LSB** になります。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

このブロックでは、次のパラメータが設定できます。

- [Number of inputs] : 連結する入力数を 2 ~ 1024 までの範囲で指定します。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

Concat ブロックでは、ザイリンクス LogiCORE は使用されません。

Configurable Subsystem Manager

このブロックは、[Xilinx Blockset] の [Tools] および [Index] ライブラリにリストされています。



Configurable Subsystem Manager

ザイリンクスの Configurable Subsystem Manager ブロックは、Simulink のコンフィギュラブルシステムの機能を拡張したブロックで、シミュレーションと同様、ハードウェアの生成にもサブシステム コンフィギュレーションが選択できるようになります。

このブロックを使用すると、System Generator で使用された場合、特定機能の Simulink ライブラリ ブロック (サブシステム) を作成できます。詳細は、「[コンフィギュラブル サブシステムと System Generator](#)」を参照してください。

System Generator では、コンフィギュラブル サブシステムとしてインポートしたライブラリ サブシステムに Configurable Subsystem Manager ブロックが自動的に挿入されます。また、Simulink および System Generator のコンフィギュラブル サブシステムの機能を使用して、ライブラリ サブシステムを手動で作成することもできます。

コンフィギュラブル サブシステムにはサブブロックのコレクションが含まれ、そのサブブロック 1 つ 1 つがサブシステムを表しています (サブシステムのブロックを選択して、どのサブブロックを使用するかを指定できます)。このサブブロックを使用して、サブシステムのシミュレーション結果が取得されます。

System Generator デザインはシミュレーションできるだけでなく、ハードウェアに変換もできるので、2 つ目のブロックはコンフィギュラブル サブシステムのハードウェア表記として指定しておく、便利です。このハードウェア表記は、コンフィギュラブル サブシステムをハードウェアに変換するために使用されるサブブロックです。たとえば、コンフィギュラブル サブシステムにフィルタをインプリメントする HDL が記述されたブラック ボックスと、通常の System Generator ブロックを使用して同じフィルタをインプリメントするサブシステムの 2 つのサブブロックが含まれるとします。この場合、サブシステムの方をサブシステムとして、ブラック ボックスの方をハードウェア表記として使用します。つまり、サブシステムをシミュレーションに、ブラック ボックスの HDL をハードウェア生成に使用します。

Configurable Subsystem Manager ブロックでは、System Generator コンフィギュラブル サブシステムのサブブロックのどれをハードウェア表記に指定するかを決めることができます。指定するには、次の手順に従ってください。

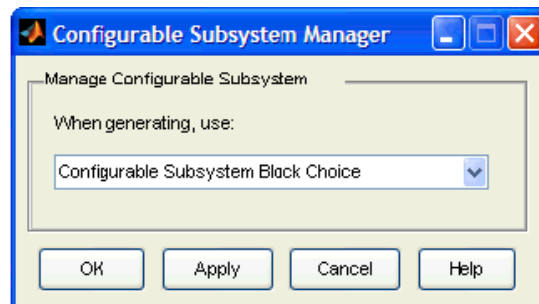
- 1) サブブロックの 1 つの中に Configurable Subsystem Manager ブロックを配置します。
- 2) パラメータ設定の [When generating, use] でハードウェア表記を選択します。

メモ： Configurable Subsystem Manager ブロックを使用できるのは、コンフィギュラブル サブシステムのサブブロック内に配置した場合のみなので、サブシステム内には最低でもサブブロックが 1 つは必要です。

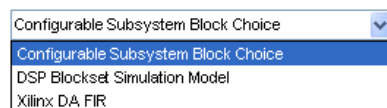
メモ： 複数のサブブロックにこのブロックがそれぞれ含まれる場合、ブロック同士が自動的に同期し、同じハードウェア表記が選択されます。

ブロック パラメータ

次の図に、このブロックのパラメータ ダイアログ ボックスを示します。



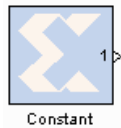
このブロックで設定できるパラメータは、[When generating, use] だけです。このパラメータでは、ハードウェアの生成にどのサブブロックを使用するかを指定します。次は、選択肢の例です。



[Configurable Subsystem Block Choice] を選択すると、コンフィギャブル サブシステムとして指定されたサブブロックがハードウェア生成にも使用されます。これ以外をリストから選択すると、それがハードウェア生成に使用されます。

Constant

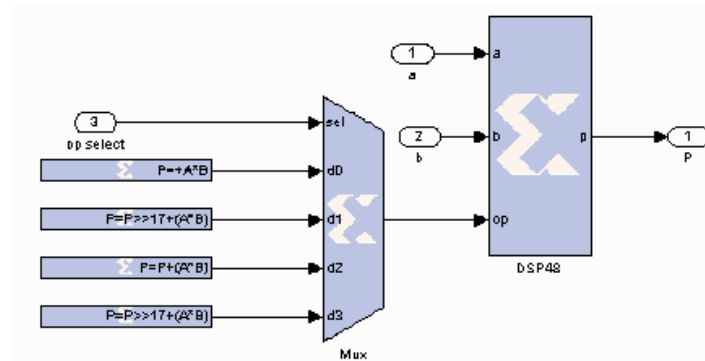
このブロックは、[Xilinx Blockset] の [Basic Elements]、[Control Logic]、[Math]、[Index] ライブラリにリストされています。



ザイリンクスの Constant ブロックでは、固定小数点値、ブール値、DSP48 命令のいずれかの定数が生成されます。このブロックは、Simulink の Constant ブロックと類似していますが、ザイリンクス ブロックの入力を直接駆動できる点が異なります。

DSP48 命令モード

DSP48 命令を作成するように設定した場合に Constant ブロックを使用すると、DSP48 の制御シーケンスを生成しやすくなります。次の図に例を示します。この例では、DSP48 ブロックに 4 つの命令を含むシーケンスが使用され、35X35 ビットの乗算器がインプリメントされています。Constant ブロックは命令をマルチプレクサに伝え、マルチプレクサがシーケンスの各命令を選択しています。



ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Type]: 定数のタイプを指定します。指定できるのは、ブール型、符号付き固定小数点、符号なし固定小数点、DSP48 命令のいずれかです。
- [Constant value]: 定数の値を指定します。変更すると、新しい値がブロック アイコンに表示されます。定数が指定された固定小数点型で正確に表すことができない場合、必要に応じて値は近似値に丸められるかサチュレート (飽和演算) が実行されます。正の値は符号なし、負の値は符号ありとしてインプリメントされます。
- [Sampled constant]: サンプル周期を出力に関連付け、Constant ブロックの駆動するブロックに伝播できます (これが便利なのは、ブロックは最後にはハードウェアをターゲットにし、Simulink のサンプル周期がハードウェアのクロック周期を作成するのに使用されるためです)。

[DSP48] タブ

[Type] フィールドで [DSP48 Instruction] をオンにすると、このタブが設定できるようになります。DSP48 の詳細は、[DSP48](#) ブロックの説明を参照してください。

- [DSP48 operation]: 選択した DSP48 命令を表示します。

- [Operation select] : DSP48 命令を選択できます。カスタムを選択すると、z_mux +/- (yx_mux + carry) フォーマットの命令形式が可能なマスク パラメータが表示されます。
- [Z mux] : DSP48 の加算器に対する Z ソースを {'0', 'C', 'PCIN', 'P', 'C', 'PCIN>>17', 'P>>17'} のいずれかから指定します。
- [Operand] : DSP48 の加算器で加算を実行するか、減算を実行するか指定します。
- [YX muxes] : DSP48 の加算器に対する YX ソースを {'0', 'P', 'A:B', 'A*B', 'C', 'P+C', 'A:B+C'} のいずれかから指定します。[A:B] にすると、A[17:0] が B[17:0] と連結されて 36 ビットの値が出力され、DSP48 加算器の入力として使用されます。
- [Carry input] : DSP48 の加算器に対する YX ソースを {'0', '1', 'CIN', '~SIGN(P or PCIN)', '~SIGN(A:B or A*B)', '~SIGND(A:B or A*B)'} のいずれかから指定します。[~SIGN (P or PCIN)] の場合は、キャリー ソースが [Z mux] の設定によって P か PCIN のいずれかになり、[~SIGN (P or PCIN)] の場合は、キャリー ソースが [Z mux] の設定によって P か PCIN のいずれかになり、[~SIGND (A*B or A:B)] 場合は、[~SIGN(A*B or A:B)] に遅延が追加されます。

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

このブロックでは、ザイリンクス LogiCORE は使用されません。

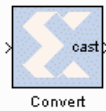
付録：DSP48 の制御命令形式

命令フィールド名	ロケーション	ニーモニック	説明
YX Mux	op[3:0]	0	0
		P	DSP48 出力レジスタ
		A:B	入力 A と B を連結 (A は MSB)
		A*B	入力 A と B を乗算
		C	DSP48 の入力 C
		P+C	DSP48 の入力 C + P
		A:B+C	入力 A と B の連結 + C レジスタ
Z Mux	op[6:4]	0	0
		PCIN	PCOUT からの DSP48 カスケード入力
		P	DSP48 出力レジスタ
		C	DSP48 の入力 C
		PCIN>>17	17 個ダウンシフトされたカスケード入力
		P>>17	17 個ダウンシフトされた DSP48 出力レジスタ
Operand	op[7]	+	加算
		-	減算

命令フィールド名	ロケーション	ニーモニック	説明
Carry In	op[8]	0 or 1	キャリーインを 0 か 1 に設定
		CIN	ソースに cin を選択
		'~SIGN(P or PCIN)	P または PCIN の対称丸め
		'~SIGN(A:B or A*B)	A:B または A*B を対称丸め
		'~SIGND(A:B or A*B)	A:B または A*B を対称丸め (遅延付き)

Convert

このブロックは、[Xilinx Blockset] の [Basic Elements]、[Data Types]、[Math]、[Index] ライブラリにリストされています。



ザイリンクスの **Convert** ブロックでは、各入力サンプルが指定した演算タイプの値に変換されます。たとえば、ある数値を符号付き (2 の補数) または符号なしの値にできます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

このブロックの [Type] は、「**System Generator** でサポートされる **Simulink ブロック**」で説明される演算タイプと同じです。

[Quantization]

量子化のエラーは、小数点以下のビット数とその部分を表現するのに不十分な場合に発生します。オプションには、[Truncate] (LSB よりも右のビットを削除)、[Round(unbiased: +/- inf)] [Round(unbiased: even values)] があります。

[Round(unbiased: +/- inf)] では、正または負の無限大への対称丸めが実行されます。これは、MATLAB の round() 関数と類似しており、0 から遠い方の最も近いビットに値が丸められます。2 つの値の調度真ん中に値がある場合は、2 つの値のうちの絶対値が大きい方の値が選択されます。たとえば、01.0110 を Fix_4_2 に丸めると、01.10 になります。これは、01.0110 が 01.01 と 01.10 の調度真ん中の値で、01.10 の方が 0 より遠いからです。

[Round (unbiased: even values)] では、偶数丸めまたは不偏丸めが実行されます。対称丸めはバイアスされます。これは、対称丸めが 0 から遠いすべての曖昧な中間値を丸めるからで、丸められた値の平均的な絶対値は、処理前の値の平均的な絶対値よりも大きくなります。偶数丸めでは、0 に近い方への丸めと遠い方への丸めを切り替えることで、これが削除されます。つまり、中間値は一番近い偶数値に丸められます。たとえば、01.0110 を Fix_4_2 に丸めると、01.10 になります。これは、01.0110 が 01.01 と 01.10 の調度真ん中の値で、01.10 が偶数だからです。01.1010 を Fix_4_2 に丸めると、01.10 になります。これは、01.1010 が 01.10 と 01.11 の調度真ん中の値で、01.10 が偶数だからです。

[Overflow]

オーバーフローのエラーは、データが表現可能な値の範囲外である場合に発生します。オーバーフローの場合、[Saturate] (上限が最大の正の値、下限が最小の負の値に飽和演算)、[Wrap] (MSB よりも左のビットを削除)、[Flag as error] (シミュレーション中に **Simulink** エラーとしてオーバーフロー) というオプションがあります。[Flag as error] は、シミュレーションのみに使用できます。生成されるハードウェアは、[Wrap] をオンにしたときと同じになります。

[Implementation] タブ

[Implementation] タブからは、次のようなパラメータを設定できます。

- [Pipeline for maximum performance] : 最大パフォーマンスを達成するためにパイプライン レジスタを使用するようブロックに命令します。これにより、ブロック レイテンシは上昇する可能性があります。

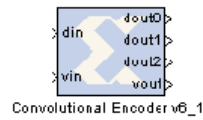
レイテンシ (パイプライン = 0) : 出力の遅延パイプライン

レイテンシ (パイプライン = 1) : 出力レジスタ、サチュレート前のレジスタ、量子化前のレジスタ、出力の遅延パイプラインとしての余剰レイテンシ

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

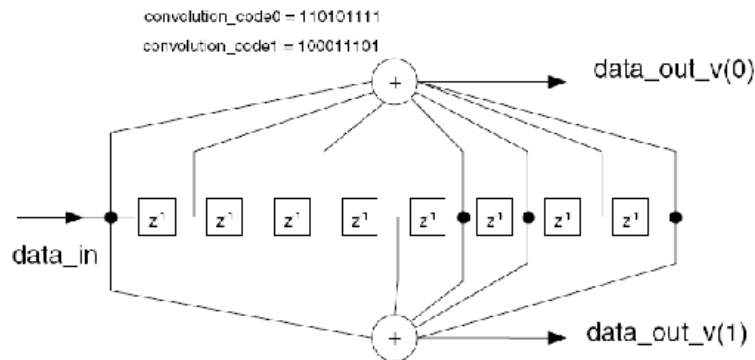
Convolution Encoder 6.1

このブロックは、[Xilinx Blockset] の [Communication] および [Index] ライブラリにリストされています。



ザイリンクスの Convolution Encoder ブロックは、たたみ込み符号のエンコーダをインプリメントします。通常 Viterbi デコーダと併用され、デジタル通信システムで順方向誤り訂正 (FEC) を実行します。

値はリニア フィード フォワード シフト レジスタで、次の図に示すように、各ビットの入力データのモジュロ 2 の和を計算してエンコードされます。シフト レジスタ長は制約長で指定されます。たたみ込み符号では、データ枠のどのビットをモジュロ 2 に加算するかが指定されます。ブロックをリセットすると、シフト レジスタが 0 に設定されます。エンコーダのレートは、入力ビット長の出力ビット長に対する比率です。レートが 1/2 のエンコーダからは、各入力ビットごとに 2 ビットが出力されます。同様に、レートが 1/3 のエンコーダからは、各入力ビットごとに 3 ビットが出力されます。



ブロック インターフェイス

このブロックには 2 ～ 4 個の入力ポートと 3 ～ 8 個の出力ポートがあります。din ポートのデータ型は UFix1_0 である必要があります。このデータ型を使用すると、エンコードされる値が許可されます。vin ポートは、din の値が有効かどうかを示し、有効な値のみがエンコードされます。出力ポート dout1 ～ dout7 からは、エンコードされたデータが出力されます。ポート dout1 は配列の最初の符号、dout2 は 2 番目の符号を出力します。配列の符号の数によってエンコーダの出力レートが設定され、データ出力ポートの数が設定されます。出力ポート vout は、出力値が有効かどうかを示します。

ブロック パラメータ ダイアログ ボックス

次の図は、ブロックのパラメータ ダイアログ ボックスを示しています。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Constraint length] : 制約長。n+1 になります (n はエンコーダに含まれる制約レジスタ長)。
- [Convolutional code array (octal)] : 8 進数のたたみ込み符号の配列。出力レートは配列長で決まります。2 ～ 7 の符号を指定します。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

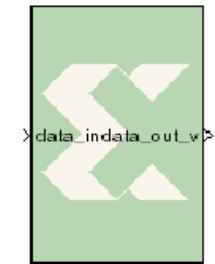
ザイリンクス LogiCORE

このブロックは、次のザイリンクス LogiCORE コアを使用します。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex		
			3、 3E	3A	3A DSP	6	6	4	5	6
Convolution Encoder 6.1	Convolution Encoder	V6.1	•	•	•			•	•	

Convolution Encoder 7.0

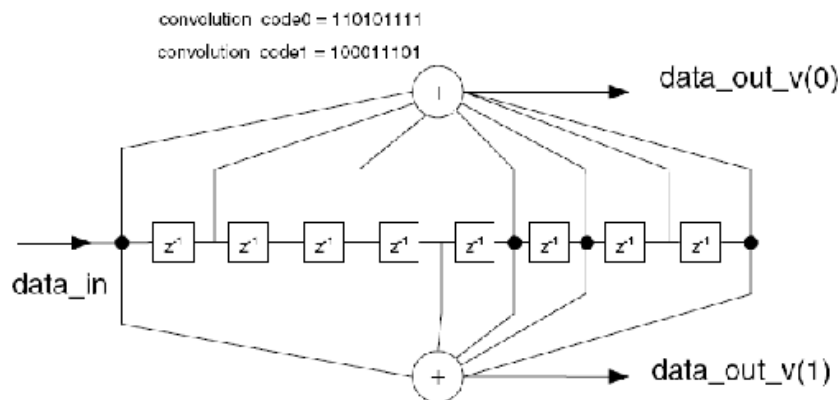
このブロックは、[Xilinx Blockset] の [Communication] および [Index] ライブラリにリストされています。



Convolution Encoder 7.0

ザイリンクスの **Convolution Encoder** ブロックは、たたみ込み符号のエンコーダをインプリメントします。通常 **Viterbi** デコーダと併用され、デジタル通信システムで順方向誤り訂正 (FEC) を実行します。

値のエンコードにはリニア フィード フォワード シフト レジスタが使用され、次の図に示すように、入力データをスライドさせながらモジュロ 2 の和を算出します。シフトレジスタ長は制約長で指定されます。たたみ込み符号では、データ枠のどのビットをモジュロ 2 に加算するかが指定されます。ブロックをリセットすると、シフトレジスタが 0 に設定されます。エンコーダのレートは、入力ビット長の出力ビット長に対する比率です。レートが 1/2 のエンコーダからは、各入力ビットごとに 2 ビットが出力されます。同様に、レートが 1/3 のエンコーダからは、各入力ビットごとに 3 ビットが出力されます。



ブロック パラメータ ダイアログ ボックス

次の図は、ブロックのパラメータ ダイアログ ボックスを示しています。

[Page_0] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

データ レート

- **[Input Rate]** : パンクチャあり。入力レートのみが変更でき、その値の範囲は 2 ~ 12 になり、 n/m のレートのエンコーダになります (n は入力レート、 $n < m < 2n$)。
- **[Output Rate]** : パンクチャなし。出力レートのみが変更でき、その値の範囲は 2 ~ 7 になり、2 の場合は 1/2、7 の場合は 1/7 のレートになります。

パンクチャ

- **[Punctured]** : ブロックをパンクチャするかどうか指定します。
- **[Dual Output]** : デュアル チャネルのパンクチャ ブロックを指定します。
- **[Puncture Code0 and Code1]** : 出力より前にエンコード データからビットを削除するため、2 つのパンクチャ パターン符号が使用されます。各パンクチャ符号の長さはパンクチャ入力レー

トと同じであり、2つの符号で1に設定されるビット総数はパンクチャ出力レートと同じでない、これらの符号は有効になりません。エンコーダからの出力ビットを示す位置の0は送信されません。具体例については、関連する LogiCORE データシートを参照してください。

[Page_1] タブ

たたみ込み

- [Constraint length] : 制約長。n+1 になります (n はエンコーダに含まれる制約レジスタ長)。
- [Convolution code] : 2進数のたたみ込み符号の配列。出力レートは配列長で決まります。2 ～ 7 の符号を指定します。

オプションのピン

- ND : ND (New Data) 入力がロジック High にサンプリングされる場合、DATA_IN の新しいシンボルが同じ立ち上がりクロック エッジでサンプリングされることを示します。
- RFD : RFD (Ready for Data) は、コアが DIN の新しいデータをサンプリングする準備ができたことを示します。
- FD_IN : FD_IN (First Data) 入力はパンクチャブロックにのみあり、新しいパンクチャグループの始まりを示します。
- RFFD : RFFD (Ready for First Data) が High の場合、FD_IN がアサートできることを示します。
- RDY : RDY (Ready) 出力は DATA_OUT_V の有効なデータを示します。
- SCLR : SCLR がアサートされると (High になると)、コアのフリップフロップが同時に初期化されます。
- CE : CE がディアサートされると (Low になると)、同期入力のすべてが無視され、ブロックには現在のステートがそのまま維持されます。

このブロックで使用するその他のパラメータは、[ブロックのパラメータ ダイアログ ボックスの共通オプション](#) で説明されています。

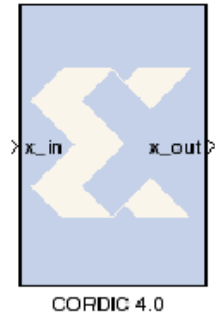
ザイリンクス LogiCORE

このブロックは、次のザイリンクス LogiCORE コアを使用します。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex			
			3、 3E	3A	3A DSP	6	6 -1L	4	5	6	6 -1L
Convolution Encoder 7.0	Convolution Encoder	V7.0	•	•	•	•		•	•	•	

CORDIC 4.0

このブロックは、[Xilinx Blockset] の [DSP]、[Math] および [Index] ライブラリにリストされています。



ザイリンクスの CORDIC 4.0 ブロックは、CORDIC (Coordinate Rotational Digital Computer) アルゴリズムをインプリメントします。

CORDIC コアは、次の論理式タイプをインプリメントします。

- 直交座標 ⇄ 極座標の変換
- 三角関数
- 双曲線
- 平方根

CORDIC コアには、次の 2 つのアーキテクチャ コンフィギュレーションが使用できます。

- シングル サイクルのデータ スループットを使用した完全なパラレル コンフィギュレーション (シリコン エリアが大量に使用される)
- マルチサイクル スループットを使用したワード シリアル インプリメンテーション (シリコン エリアは少ししか使用されない)

粗回転 (Coarse Rotation) は、入力サンプルを全円から第一象限に回転するために実行されます (CORDIC アルゴリズムは第一象限でしか有効にならないので、粗回転段階は必ず必要です)。反粗回転段階では、出力サンプルが正しい象限に回転されます。

CORDIC アルゴリズムではこの結果の振幅にスケール係数を使用します。CORDIC コアには CORDIC スケール係数を自動的に補うオプションがあります。

ブロック パラメータ ダイアログ ボックス

[Page 1] タブ

ファンクション選択

- [Square_Root]: オンにすると、簡素化された CORDIC アルゴリズムを使用して入力の正の平方根が計算されます。
- [Rotate]: オンにすると、入力ベクタの (X,Y) が CORDIC アルゴリズムを使用して入力角度分回転されます。これにより、スケール出力ベクタ $Z_i * (X', Y')$ が生成されます。
- [Translate]: オンにすると、入力ベクタの (X,Y) は Y コンポーネントが 0 になるまで CORDIC アルゴリズムを使用して回転されます。これにより、スケール出力振幅 $Z_i * \text{Mag}(X,Y)$ と出力位相 $\text{Atan}(Y/X)$ が生成されます。
- [Sin_and_Cos]: オンにすると、ユニット ベクタが CORDIC アルゴリズムを使用して入力角度分回転されます。これにより、出力ベクタ ($\text{Cos}()$, $\text{Sin}()$) が生成されます。
- [Sinh_and_Cosh]: オンにすると、CORDIC アルゴリズムが使用され、ベクタ (1,0) が双曲線カーブに沿って双曲線角度 p まで移動します。双曲線角度は、ベクタ (X, Y) の下のエリアの対数を表し、三角関数とは関係ありません。これにより、出力ベクタ ($\text{Cosh}(p)$, $\text{Sinh}(p)$) が生成されます。
- [Arc_Tan]: オンにすると、入力ベクタの (X,Y) は Y コンポーネントが 0 になるまで CORDIC アルゴリズムを使用して回転されます。これにより、出力角度 $\text{Atan}(Y/X)$ が生成されます。

- **[Arc_Tanh]** : オンにすると、CORDIC アルゴリズムが使用されて、Y コンポーネントが 0 になるまで入力ベクタ (X,Y) が双曲線カーブに沿って移動します。これにより、双曲線の角度 $\text{Atanh}(Y/X)$ が生成されます。双曲線角度は、ベクタ (X,Y) の下のエリアの対数を表し、三角関数とは関係ありません。

アーキテクチャ コンフィギュレーション

- **[Word_Serial]** : 小さいエリアのハードウェア結果が選択されます。
- **[Parallel]** : 高スループットのハードウェア結果が選択されます。

パイプライン モード

- **[No_Pipelining]** : CORDIC コアはパイプラインなしにインプリメントされます。
- **[Optimal]** : CORDIC コアは、LUT を追加で使わずに、できるだけ多くのパイプライン ステージを使用してインプリメントされます。
- **[Maximum]** : CORDIC コアは各シフト加算減算の後にパイプラインを使用してインプリメントされます。

[Page 2] タブ

データ フォーマット

- **[SignedFraction]** : デフォルト設定。X および Y 入力と出力は、2 ビットの整数幅の固定小数点の 2 の補数値として記述されます。
- **[UnsignedFraction]** : 平方根のファンクション コンフィギュレーションにのみ使用できます。X および Y 入力と出力は、1 ビットの整数幅の符号なし固定小数点として記述されます。
- **[UnsignedInteger]** : 平方根のファンクション コンフィギュレーションにのみ使用できます。X および Y 入力と出力は、符号なしの整数として記述されます。

位相フォーマット

- **[Radians]** : 位相は、ラジアン単位で、3 ビットの整数幅の固定小数点の 2 の補数値として記述されます。
- **[Scaled_Radians]** : 位相は、 π ラジアン単位で、3 ビットの整数幅の固定小数点の 2 の補数値として記述されます。1 スケール ラジアン = $\pi * 1$ ラジアンです。

出力オプション

- **[Output width]** : 出力ポート、X_OUT、Y_OUT、PHASE_OUT の幅を制御します。出力幅は、8 ~ 48 ビットでコンフィギュレーションできます。

丸めモード

- **[Truncate]** : X_OUT、Y_OUT、PHASE_OUT 出力は切り捨て処理されます。
- **[Round_Pos_Inf]** : X_OUT、Y_OUT、PHASE_OUT 出力は丸め処理されます (1/2 の場合は上に切り上げ)。
- **[Round_Pos_Neg_Inf]** : X_OUT、Y_OUT、PHASE_OUT 出力は丸め処理されます (1/2 の場合は上に切り上げ、-1/2 の場合は下に切り下げ)。
- **[Nearest_Even]** : X_OUT、Y_OUT、PHASE_OUT 出力は一番近い偶数値に丸め処理されます (1/2 の場合は下に切り下げ、3/2 の場合は上に切り上げ)。

[Page 3] タブ

アドバンス コンフィギュレーション パラメータ

- **[Iterations]** : 内部加算減算の繰り返し回数を制御します。0 に設定すると、実行される繰り返し回数が、出力に必要な精度に基づいて自動的に決定されます。
- **[Precision]** : 内部加算減算の繰り返し精度を設定します。0 に設定すると、内部精度が出力に必要な精度と内部繰り返し回数に基づいて自動的に決定されます。
- **[Coarse rotation]** : **Coarse Rotation** モジュールのインスタンスエーションを制御します。**Coarse Rotation** モジュールのインスタンスエーションは、**Vector Rotation**、**Vector Translation**、**Sin and Cos**、**Arc Tan** コンフィギュレーションのデフォルトです。**Coarse Rotation** がこれらのファンクションに対してオフになっていると、入力/出力範囲が第一象限 ($-\pi/4 \sim +\pi/4$) に制限されます。
- **Coarse Rotation** は、**Sinh/Cosh**、**Arctanh**、**Square Root** コンフィギュレーションには必要ありません。標準の **CORDIC** アルゴリズムは、第一象限で動作します。**Coarse Rotation** を使用すると、入力サンプルが最初の象限に回転され、出力サンプルが正しい象限に反回転で戻されるので、**CORDIC** の動作範囲がすべての象限に拡張されます。
- **[Compensation scaling]** : **CORDIC** 振幅スケールを補うために使用される **Compensation Scaling** モジュールを制御します。**CORDIC** の振幅スケールは、**Vector Rotation** および **Vector Translation** ファンクション コンフィギュレーションに影響しますが、**SinCos**、**SinhCosh**、**ArcTan**、**ArcTanh**、**Square Root** ファンクション コンフィギュレーションには影響しません。後述のコンフィギュレーションでは、**[Compensation scaling]** は **[No Scale Compensation]** に設定しておく必要があります。

オプションのピン

- **en** : イネーブル信号がアサートされないと、イネーブル信号が再びアサートされるまで (またはリセット信号がアサートされるまで)、ブロックが現在のステートを保持したままになります。リセット信号はイネーブル信号よりも優先されます。イネーブル信号は、ブロックのサンプルレートの倍数のレートで実行する必要があり、イネーブル ポートを駆動する信号は、ブール型にする必要があります。
- **rst** : このリセット信号がアサートされると、ブロックが最初のステートに戻ります。リセット信号は、ブロックで使用可能なオプションのイネーブル信号よりも優先されます。この信号は、ブロックのサンプル レートの倍数のレートで実行する必要があり、リセット ポートを駆動する信号は、ブール型にする必要があります。
- **nd** : 入力ポートに新しいサンプルがあります。
- **rdy** : 新しい出力データが準備できています。
- **X out** : データ出力ポート
- **Y out** : データ出力ポート
- **Phase output** : データ出力ポート

Xilinx LogiCORE

このブロックは、次のザイリンクス LogiCORE コアを使用します。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
CORDIC 4.0	CORDIC	V4.0	•	•	•	•	•	•	•	•	•	•

Counter

このブロックは、[Xilinx Blockset] の [Basic Elements]、[Control Logic]、[Math]、[Index] ライブラリにリストされています。



ザイリンクスの Counter ブロックは、フリー ランニング カウンタ、またはアップ カウンタ、ダウン カウンタ、アップ/ダウン カウンタなどのカウント制限のあるカウンタをインプリメントします。カウンタの出力は、符号付きまたは符号なしの固定小数点の値で指定できます。

フリー ランニング カウンタは、FPGA ハードウェアの中でもコストが一番安価です。フリー ランニングのアップ カウンタ、ダウン カウンタ、アップ/ダウン カウンタは、このブロックの [Provide load port] をオンにして、その出力に *din* 入力ポートの値を読み込むようにも設定できます。

$$out(n) = \begin{cases} InitialValue & \text{if } n = 0 \\ (out(n-1) + Step) \bmod 2^N & \text{otherwise} \end{cases}$$

フリー ランニングのアップ カウンタの出力は、次のように計算されます。

$$out(n) = \begin{cases} InitialValue & \text{if } n = 0 \\ din(n-1) & \text{if } load(n-1) = 1 \\ (out(n-1) + Step) \bmod 2^N & \text{otherwise} \end{cases}$$

n は、カウンタのビット数を示しています。フリー ランニングのダウン カウンタの場合は、加算を減算に置き換えて計算します。

フリー ランニングのアップ/ダウン カウンタの場合、入力アップ ポートが 1 の場合は加算、0 の場合は減算が実行されます。

カウント制限のあるカウンタは、フリー ランニング カウンタとコンパレータを組み合わせでインプリメントします。カウント制限のあるカウンタの出力精度は、64 ビットにのみ制限されています。このタイプのカウンタは、最初と最後の値の差を調度半分に分ける [Step] 値を設定すると、最初と最後の値の間に割り込むように設定できます。

カウント制限のあるアップ カウンタの出力は、次のように計算されます。

$$out(n) = \begin{cases} InitialValue & \text{if } n = 0 \text{ or } out(n-1) = CountLimit \\ (out(n-1) + Step) \bmod 2^N & \text{otherwise} \end{cases}$$

カウント制限のあるダウン カウンタの場合は、加算を減算に置き換えて計算します。カウント制限のあるアップ/ダウン カウンタの場合、入力ポート *up* が 1 の場合は加算、0 の場合は減算が実行されます。

load ポートが付いたフリー ランニングのアップ カウンタの出力は、次のように計算されます。

$$out(n) = \begin{cases} StartCount & \text{if } n = 0 \text{ or } rst(n) = 1 \\ din & \text{if } rst(n) = 0 \text{ and } load(n) = 1 \\ (out(n-1) + CountByValue) \bmod 2^N & \text{otherwise} \end{cases}$$

n は、カウンタのビット数を示しています。ダウン カウンタの場合は、加算を減算に置き換えて計算します。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- **[Counter type]** : カウント制限のあるカウンタまたはフリー ランニングのカウンタのどちらかを指定します。
- **[Number of bits]** : ブロックの出力のビット数を指定します。
- **[Binary point]** : ブロックの出力の 2 進小数点の位置を指定します。
- **[Output type]** : ブロックの出力を符号付きにするか符号なしにするか指定します。
- **[Initial value]** : 最初に出力される値を指定します。
- **[Count to value]** : 最後の値を指定します。この値により、カウント制限のあるカウンタはリセットされます。**Inf** は、指定された精度で表現可能な最大出力を示します。この値は、最初の値と同じにはできません。
- **[Step]** : インクリメント値またはデクリメント値を指定します。
- **[Count direction]** : カウント方向 (アップまたはダウン) を指定するか、オプションの入力ポート **up** でカウンタの方向を指定します (アップ/ダウンを選択した場合)。
- **[Provide load port]** : オンにすると、ブロックが **load** と **din** ポートの付いたフリー ランニングのロード カウンタとして動作します。**load** ポートは、フリー ランニング カウンタの場合にのみ使用できます。

[Implementation] タブ

[Implementation] タブからは、次のようなパラメータを設定できます。

インプリメンテーションの詳細

- **[Use behavioral HDL (otherwise use core)]** : ビヘイビア レベルの HDL 記述が使用されます。これで、最適化をパフォーマンス重視にするか、エリア重視にするかをダウンストリームのロジック合成ツールで柔軟に決定できます。

コアのパラメータ

- **[Implement using]** : コア ロジックはファブリックか、**DSP48** がターゲット デバイスで可能な場合は **DSP48** にインプリメントできます。デフォルトは **[Fabric]** です。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

このブロックは、次のザイリンクス LogiCORE コアを使用します。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
Counter	Binary Counter	V11.0	•	•	•	•	•	•	•	•	•	•

DAFIR v9_0

このブロックは、[Xilinx Blockset] の [DSP] および [Index] ライブラリにリストされています。



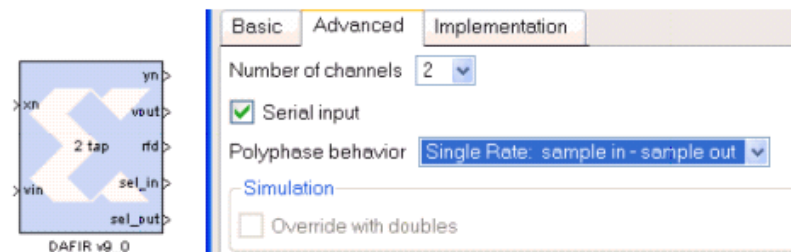
ザイリンクスの DAFIR ブロックは、分散演算タイプの FIR (Finite Impulse Response) デジタル フィルタ、または同一の FIR フィルタ (マルチチャネル モード) のバンクをインプリメントします。

N タップ フィルタは、 $h(0)$ 、 $h(1)$ 、....、 $h(n-1)$ のように N 個のフィルタ係数 (またはタップ) で定義されます。 $h(i)$ はそれぞれザイリンクスの固定小数点の値になります。このフィルタ ブロックでは、ザイリンクスの固定小数点のデータ サンプルのストリーム ($x(0)$ 、 $x(1)$ 、...) を受信し、n で出力を計算します。

ブロック インターフェイス

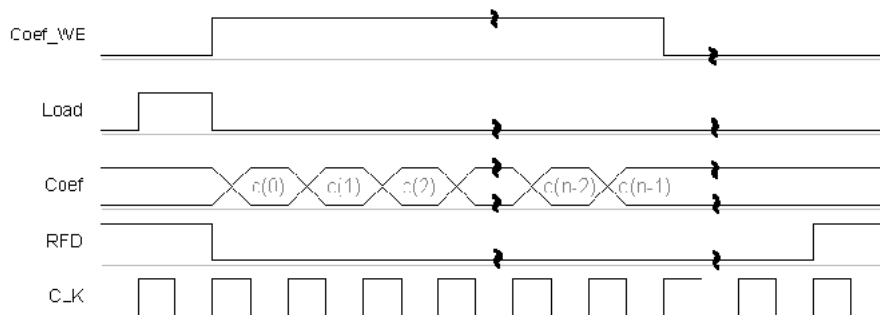
FIR ブロックには、1 ~ 8 つのデータ チャネルと 複数のオプションのポート を含めることができます。

- **vin : xn** シンボルが有効か無効かを示します。デシメーション FIR フィルタの場合、vin ポートのステートは間引きされるサンプル グループすべてに合わせる必要があります。つまり、N ずつの vin のサンプル グループ (N はデシメーション係数) では、すべて 1 かすべて 0 のいずれかにする必要があります。サンプル グループは、シミュレーションの開始時点 ($t=0$) から揃えられます。
- **vout : yn** シンボルが有効か無効かを示します。
- **rfd** : ブロックが新しいデータを受信できる状態かどうかを示します。このポートは、入力ポート xn と同じデータ レートでブール信号を駆動します。この信号は、シミュレーションの開始時にアサートされ、リロード サイクルが開始されるまでアサートされたまま (true) です。rfd 信号はロード信号がアサートされると、すぐ後のサイクルで Low になります。rfd 信号はブロックがリロード シーケンスを終了すると、再びアサートされます。使用できるのは、係数をリロードしているときか、シリアル入力を選択されたときです。
- **sel_in** : シリアル入力を選択された場合、現在のフィルタの入力チャネル数を示します。
- **sel_out** : シリアル入力を選択された場合、現在のフィルタの出力チャネル数を示します。



係数のリロード

DA FIR フィルタには、係数をリロードするためのポートをオプションで含めることができます。リロード シーケンスが開始されると、フィルタは新しいデータ入力サンプルを受信するのを停止し、新しいフィルタ係数を受信し始めます。この新しい係数がすべて書き込まれると、フィルタは係数処理し、必要な内部データ構造を初期化します。リロードに必要な時間は、フィルタの長さタイプによって異なります。リロード シーケンスが終了すると、フィルタはオンライン状態に戻り、新しい入力データ サンプルの受信を続行します。リロード シーケンスとフィルタのリロード時間の詳細は、**FIR コア**のデータシートを参照してください。次の図は、リロード シーケンスのタイミング例です。

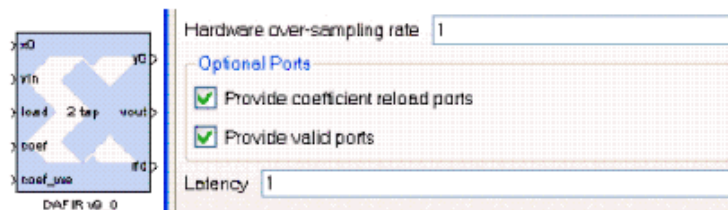


係数のリロード用のオプション ポート

coef : 新しいフィルタ係数はこのポートを介してブロックに書き込まれます。**coef** ポートのビット数や 2 進小数点は、指定した係数ビット数と係数 2 進小数点と同じになります。このポートは、入力ポート **xn** と同じデータ レートで駆動される必要があります。

coef_we : **coef** ポートのデータがブロックに書き込まれるタイミングを制御する書き込みイネーブル信号です。このポートは、リロード サイクルが開始した後に、新しい係数がフィルタに書き込まれるタイミングを決定します。最初の新しい係数は、ロード信号がアサートされた後のサイクルでフィルタに書き込めるようになります。このポートは、入力ポート **xn** と同じデータ レートのブール信号で駆動される必要があります。

load : **load** ポートがアサートされると (**true** になると)、係数のリロード シーケンスが開始されます。**load** 信号は 1 サイクル間送信される必要があります、リロード シーケンス中の次のアサートで、リロード プロセスが再開されます。このポートは、入力ポート **xn** と同じデータ レートのブール信号で駆動される必要があります。



ブロック パラメータ ダイアログ ボックス

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- **[Coefficients]** : フィルタ係数のベクタで、MATLAB ワークスペースの変数から算出したり、MATLAB で代わりに計算することもできます。System Generator のチュートリアル の例も参照してください。
- **[Structure]** : ザイリンクス Smart-IP FIR コアの推奨インプリメンテーションは、フィルタ タップ シーケンスの構造によって異なります。[Inferred from Coefficients]、[None]、[Symmetric]、[Negative Symmetric]、[Half-Band Filter]、[Interpolate FIR] のいずれかを選択できます。
- **[Number of bits (always signed)]** : フィルタの係数を表すのに使用されるビット数を入力します。
- **[Binary point]** : フィルタの係数を表すのに使用される小数点以下のビット数を入力します。
- **[Hardware over-sampling rate]** : 並列処理の度合いを決定します。レートを 1 にすると、完全なパラレル フィルタが生成され、 n ビット信号のレートを n にすると非対称の、 $n+1$ にすると対称のインパルス レスポンスの完全なシリアル インプリメンテーションになります。中間の値にすると、中レベルの並列処理を使用してインプリメンテーションされます。

オプション ポート

- **[Provide coefficient reload ports]** : ブロックに係数をリロードするインターフェイスを追加します。
- **[Provide valid ports]** : 入力有効ポート (vin) と出力有効ポート (vout) をブロックに追加します。
- **[Provide reset port]** : ブロックに rst ポートが追加されます。

[Latency]

[Advanced] タブ

[Advanced] タブからは、次のようなパラメータを設定できます。

- **[Number of channels]** : 1 ~ 8 を選択します。マルチチャネル フィルタの場合は、多相機能がサポートされていないので、フィルタはシングル レートにする必要があります。チャンネルをシリアルで処理するコアの場合、System Generator は必要なスループットにするため、チャンネル数と同じ係数でコアをオーバークロックにします。制御ロジックのオーバーヘッドを削減するには、ブロックの有効ビットがすべての入力で一貫している必要があります。
- **[Serial input]** : チャンネル数が複数ある場合、フィルタへの入力をシリアル (時分割多重) またはパラレルのいずれかにできます。
- **[Polyphase behavior]** : デシメーション、インターポレーション、シングル レートから該当するものを選択します。

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

このブロックでは、常にザイリンクス LogiCORE Distributed Arithmetic FIR Filter コアが使用されます。

Simulink モデルはサンプル入力/サンプル出力ベースで動作しますが、このコアではオーバークロックによるシリアル演算を使用できます。この場合、レイテンシは追加されますが、フィルタに必要なハードウェアは削減できます。フィルタ モードとパラメータの詳細は、このコアのデータシートを参照してください。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan				Virtex		
			3、 3E	3A	3A DSP	6	4	5	6
DAFIR v9_0	Distributed Arithmetic FIR Filter	V9.0	•				•		

DDS Compiler 2.1

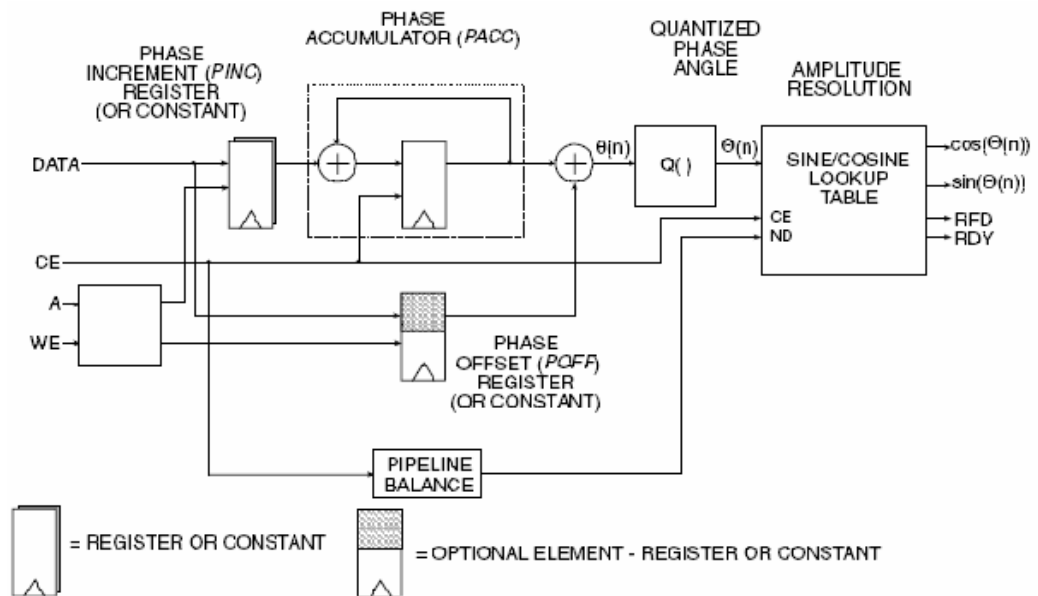
メモ：このブロックは、**DDS Compiler 4.0** ブロックに置き換わっています。

このブロックは、[Xilinx Blockset] の [DSP] および [Index] ライブラリにリストされています。

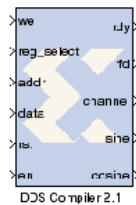


ザイリンクスの DDS Compiler v2_1 ブロックは、ダイレクト デジタル シンセサイザで、NCO (Numerically Controlled Oscillator) とも呼ばれます。このブロックでは、ルックアップ テーブルを使用してサイン波が生成されます。ルックアップ テーブルでマップされた位相は、デジタル積分器 (アキュムレータ) で出力正弦波形にされます。

DDS を理解するには、ブロックがどのように FPGA ハードウェアにインプリメントされるかを知る必要があります。次の図は、ザイリンクス LogiCORE の高位レベルを示したものです。位相増分と位相オフセットは定数として定義できるほか、オプションの入力ポートを介してダイナミックに設定することもできます。これらの値は、サンプルごとのサイクルで定義されます。たとえば、位相増分が $1/10$ の場合は、サンプル 10 サイクルで 1 つのサイン波が終了することを示します。位相増分が累積されると、位相オフセットが結果に追加されます。ディザリングが使用されると、量子化の前にディザリング シーケンス (量子化器からの位相エラーを回避) が追加されます。この後、量子化された値を使用してサイン/コサイン ルックアップ テーブルにインデックスが付けられ、位相空間が時間にマップされます。



ブロック インターフェイス



DDS Compiler v2_0 ブロックには、次のようなポートがあります。

入力ポート

- **we** : 書き込みイネーブルです。オフセット周波数メモリとプログラマブル周波数メモリのいずれか、または両方に対して書き込みをイネーブルにします。どちらのメモリに書き込まれるかは、**reg_select** ポートの値によって決まります。下位の **LogiCORE** の **WE** ポートにマップされます。
- **reg_select** : 位相増分メモリ (**PINC**) と位相オフセット (**POFF**) メモリへの書き込みに使用するアドレス セレクトです。**reg_select=0** で **PINC** メモリが、**reg_select=1** で **POFF** メモリが選択されます。
- **addr** : このバスは、現在選択しているメモリに対して最大 16 チャンネルまでアドレス指定するためのもので、ビット数は 2 チャンネルの場合が 1、3 または 4 チャンネルの場合が 2、5 ~ 8 チャンネルの場合が 3、9 ~ 16 チャンネルの場合が 4 になります。
- **rst** : 同期リセットです。1 の場合、ブロックの内部メモリがリセットされます。下位の **LogiCORE** の **SCLR** (同期クリア) 入力にマップされます。
- **en** : ユーザー イネーブルです。1 の場合、ブロックがアクティブになります。下位の **LogiCORE** の **CE** ポートにマップされます。

出力ポート

- **rdy** : アクティブ **High** で出力データが準備できているかどうかを示します。出力サンプルが有効になったことを示します。
- **rfd** : アクティブ **High** でデータが準備できているかどうかを示します。多くのザイリンクス **LogiCORE** に含まれるデータ フロー制御信号です。DDS のコンテンツは、ほかの **LogiCORE** コアと合わせるためだけに提供されます。このオプションのポートは、常に **VCC** に接続されます。
- **channel** : チャンネルのインデックスです。下位のコアがマルチチャンネル用にコンフィギュレーションされている場合に、どのチャンネルが出力で使用可能なのかを示します。これは、符号なしの値になります。幅は **[Basic]** タブの **[Output frequency array (MHZ)]** で指定したチャンネル数で決まります。
- **sine** : サイン出力の値です。下位の **LogiCORE** の **SINE** 出力にマップされます。
- **cosine** : コサイン出力の値です。下位の **LogiCORE** の **COSINE** 出力にマップされます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

ファンクションの選択

- [Output selection] : サイン、コサイン、またはその両方など、ブロックが計算する関数を指定します。

極性

- ◆ [Negative sine] : サイン出力をネゲートします。
- ◆ [Negative cosine] : コサイン出力をネゲートします。

パフォーマンス オプション

- [DDS clock rate (MHz)] : DDS コアに提供されるクロックの周波数を指定します。
- [Spurious free dynamic range (dB)] : DDS 出力で生成される 帯域外ノイズに必要な周波数ドメインを定義します。範囲は、スプリ アス抑制の 16dB ~ 115dB です。SFDR 値が 102dB 以上の場合、インプリ メンテーションでエンベデッド 乗算器が必要になるテイラー展開が使用されます。
- [Frequency resolution (Hz)] : 同調周波数の粒度を指定します。
- [Output Function] : サイン、コサイン、またはその両方など、ブロックが計算する関数を指定します。
- [Negative sine] : サイン出力をネゲートします。
- [Negative cosine] : コサイン出力をネゲートします。
- [Output frequency] の [Type] : 出力周波数を [Fixed] にするか [Programmable] にするか指定します。[Programmable] を選択すると、channel、data、we 入力ポートがブロックに追加されます。

オプション ポート

- rfd : アクティブ High でデータが準備できているかどうかを示します。rfd は、多くのザイリンクス LogiCORE コアに含まれるデータ フロー制御信号です。DDS のコンテンツは、ほかの LogiCORE ベースのコアと合わせるためだけに提供されます。このオプションのポートは、常に VCC に接続されます。
- rdy : アクティブ High で出力データが準備できているかどうかを示します。出力サンプルが有効になったことを示します。
- channel : 現在の出力サンプルが対応するチャンネルがどれかを示す出力チャンネル ポートが含まれます。

[Output Frequency] タブ

- [Output frequency array (MHz)] : チャンネルごとに独立した周波数を配列に入力できます。

[Output Phase Offset] タブ

- [Phase Offset] : 位相オフセットを [Fixed]、[Programmable]、[None] のいずれかに指定します。[Programmable] を選択すると、channel、data、we 入力ポートがブロックに追加されます。

- **[Phase Offset Angles (x2pi radians)]** : チャンネルごとに独立したオフセットを配列に入力できます。入力した値は、 2π ラジアンで乗算されます。**[Phase Offset]** の **[Type]** を **[Fixed]** にした場合にのみ設定できます。

[Implementation] タブ

[Implementation] タブからは、次のようなパラメータを設定できます。

インプリメンテーション オプション

- **[Memory type]** : ブロックをインプリメントするのに分散メモリを使用するか、ブロック RAM メモリを使用するかを指定します。デフォルトではブロック RAM がインプリメントされます。
- **[DSP48 Use]** : **[Maximal]** に設定すると、DSP スライスが使用され、最大のパフォーマンスが達成されます。

パフォーマンス オプション

- **[Latency configuration]** : **[Auto]** にすると、最大パフォーマンスにするためにコアがパイプライン接続され、コアには高いレートのカロックが供給できるようになります。
- **[Accumulator latency]** : 位相アキュムレータのレイテンシを 0 か 1 に指定します。

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

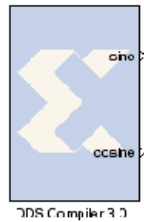
ザイリンクス LogiCORE

このブロックは、次のザイリンクス LogiCORE コアを使用します。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan			Virtex	
			3, 3E	3A	3A DSP	4	5
DDS Compiler 2.1	DDS Compiler v2_1	V2.1	•	•	•	•	•

DDS Compiler 3.0

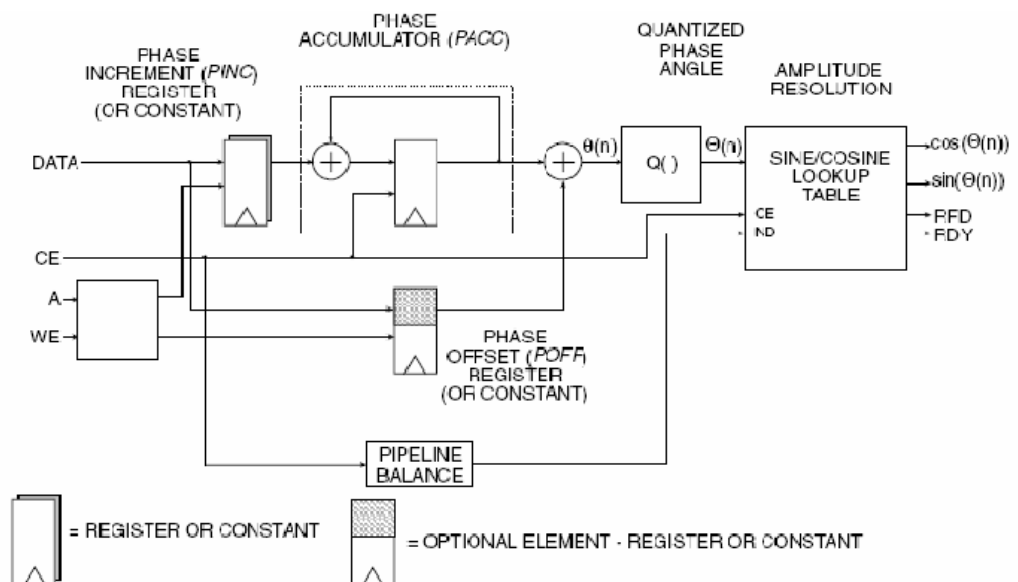
このブロックは、[Xilinx Blockset] の [DSP] および [Index] ライブラリにリストされています。



ザイリンクス DDS Compiler 3.0 ブロックの機能は DDS Compiler 2.1 と同じですが、Virtex-6 と Spartan-6 がサポートされている点が異なります。

ザイリンクスの DDS Compiler ブロックは、ダイレクト デジタル シンセサイザで、NCO (Numerically Controlled Oscillator) とも呼ばれます。このブロックでは、ルックアップ テーブルを使用してサイン波が生成されます。ルックアップ テーブルでマップされた位相は、デジタル積分器 (アキュムレータ) で出力正弦波形にされます。

DDS を理解するには、ブロックがどのように FPGA ハードウェアにインプリメントされるかを知らする必要があります。次の図は、ザイリンクス LogiCORE の高位レベルを示したものです。位相増分と位相オフセットは定数として定義できるほか、オプションの入力ポートを介してダイナミックに設定することもできます。これらの値は、サンプルごとのサイクルで定義されます。たとえば、位相増分が 1/10 の場合は、サンプル 10 サイクルで 1 つのサイン波が終了することを示します。位相増分が累積されると、位相オフセットが結果に追加されます。ディザリングが使用されると、量子化の前にディザリング シーケンス (量子化器からの位相エラーを回避) が追加されます。この後、量子化された値を使用してサイン/コサイン ルックアップ テーブルにインデックスが付けられ、位相空間が時間にマップされます。



ブロック インターフェイス



DDS Compiler v2_0 ブロックには、次のようなポートがあります。

入力ポート

- **we** : 書き込みイネーブル (アクティブ High) です。PINC または POFF メモリへの書き込み動作をイネーブルにします。下位の LogiCORE の WE ポートにマップされます。
- **reg_select** : 位相増分 メモリ (PINC) と位相オフセット (POFF) メモリへの書き込みに使用するアドレス セレクトです。reg_select=0 で PINC メモリが、reg_select=1 で POFF メモリが選択されます。
- **addr** : このバスは、現在選択しているメモリに対して最大 16 チャンネルまでアドレス指定するためのもので、ビット数は 2 チャンネルの場合が 1、3 または 4 チャンネルの場合が 2、5 ～ 8 チャンネルの場合が 3、9 ～ 16 チャンネルの場合が 4 になります。
- **data** : 時分割のデータ バスです。このポートは、プログラマブルなオフセット周波数メモリおよびプログラマブルな位相オフセット メモリに値を送信するために使用されます。 下位の LogiCORE の DATA バスにマップされます。
- **rst** : 同期リセットです。 1 の場合、ブロックの内部メモリがリセットされます。 下位の LogiCORE の SCLR (同期クリア) 入力にマップされます。
- **en** : ユーザー イネーブルです。 1 の場合、ブロックがアクティブになります。 下位の LogiCORE の CE ポートにマップされます。

出力ポート

- **rdy** : アクティブ High で出力データが準備できているかどうかを示します。出力サンプルが有効になったことを示します。
- **rfd** : アクティブ High でデータが準備できているかどうかを示します。多くのザイリンクス LogiCORE に含まれるデータ フロー制御信号です。DDS のコンテンツは、ほかの LogiCORE コアと合わせるためだけに提供されます。このオプションのポートは、常に VCC に接続されます。
- **channel** : チャンネルのインデックスです。下位のコアがマルチチャンネル用にコンフィギュレーションされている場合に、どのチャンネルが出力で使用可能なのかを示します。これは、符号なしの値になります。幅は [Basic] タブの [Output frequency array (MHz)] で指定したチャンネル数で決まります。
- **sine** : サイン出力の値です。下位の LogiCORE の SINE 出力にマップされます。
- **cosine** : コサイン出力の値です。下位の LogiCORE の COSINE 出力にマップされます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

ファンクションの選択

- **[Output selection]** : サイン、コサイン、またはその両方など、ブロックが計算する関数を指定します。

極性

- ◆ **[Negative sine]** : サイン出力をネゲートします。
- ◆ **[Negative cosine]** : コサイン出力をネゲートします。

パフォーマンス オプション

- **[Number of channels]** : チャネルは、**DDS** で時分割され、各チャネルで有効なクロックに影響を与えます。**DDS** では、1 ~ 16 の時分割チャネルまでサポートされます。
- **[DDS clock rate (MHz)]** : **DDS** コアに提供されるクロックの周波数を指定します。
- **[Spurious free dynamic range (dB)]** : **DDS** 出力で生成される帯域外ノイズに必要な周波数ドメインを定義します。範囲は、スプリアス抑制の 16dB ~ 115dB です。**SFDR** 値が 102dB 以上の場合、インプリメンテーションでエンベデッド乗算器が必要になるテイラー展開が使用されます。
- **[Frequency resolution (Hz)]** : 同調周波数の粒度を指定します。

オプションのポート

- **rfd** : アクティブ **High** でデータが準備できているかどうかを示します。多くのザイリンクス **LogiCORE** に含まれるデータフロー制御信号です。**DDS** のコンテンツは、ほかの **LogiCORE** コアと合わせるためだけに提供されます。このオプションのポートは、常に **VCC** に接続されます。
- **rdy** : アクティブ **High** で出力データが準備できているかどうかを示します。出力サンプルが有効になったことを示します。
- **channel** : チャネルのインデックスです。下位のコアがマルチチャネル用にコンフィギュレーションされている場合に、どのチャネルが出力で使用可能なのかを示します。これは、符号なしの値になります。幅は [Basic] タブの **[Output frequency array (MHz)]** で指定したチャネル数で決まります。

[Output Frequency] タブ

- **[Phase Increment]** : 位相増分を **[Fixed]** または **[Programmable]** に指定します。**[Programmable]** を選択すると、**data** および **we** 入力ポートがブロックに追加されます。データバス入力を介して位相増分の値をインポートする方法については、関連する **LogiCORE** データシートを参照してください。
- **[Output frequencies (MHz)]** : チャネルごとに独立した周波数を配列に入力できます。

[Output Phase Offset] タブ

- **[Phase Offset]** の **[Type]** : 位相オフセットを **[Fixed]**、**[Programmable]**、**[None]** のいずれかに指定します。**[Programmable]** を選択すると、**data** および **we** 入力ポートがブロックに追加されます。

- **[Phase Offset Angles (x2pi radians)]** : チャンネルごとに独立したオフセットを配列に入力できます。入力した値は、 2π ラジアンで乗算されます。**[Phase Offset]** の **[Type]** を **[Fixed]** にした場合にのみ設定できます。

[Implementation] タブ

[Implementation] タブからは、次のようなパラメータを設定できます。

インプリメンテーション オプション

- **[Memory type]** : ブロックをインプリメントするのに分散メモリを使用するか、ブロック RAM メモリを使用するかを指定します。デフォルトは **[Auto]** で、ツールによって最適なインプリメンテーションが自動的に決定されます。
- **[Optimization Goal]** : ブロックを **[Area]** または **[Speed]** のどちらかを重要視して最適化するか指定できます。デフォルトは **[Auto]** で、ツールによって最適なインプリメンテーションが自動的に決定されます。
- **[DSP48 Use]** : **[Maximal]** に設定すると、DSP スライスが使用され、最大のパフォーマンスが達成されます。

パフォーマンス オプション

- **[Noise Shaping]** : **[None]**、**[Phase Dithering]**、**[Taylor Series Corrected]**、**[Auto]** のいずれかを選択します。**[Auto]** を選択すると、ノイズシェープのタイプがほかの LogiCORE パラメータに基づいて選択されます。
- **[Latency configuration]** : **[Auto]** に設定すると、最大のパフォーマンスが達成されるように、LogiCORE コアがパイプライン処理されます。これにより、コアのクロックを最高速のレートにできます。このコンフィギュレーション オプションを使用すると、LogiCORE のレイテンシを選択できます。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

このブロックは、次のザイリンクス LogiCORE コアを使用します。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex			
			3、 3E	3A	3A DSP	6	6-1L	4	5	6	6-1L
DDS Compiler 3.0	DDS Compiler v3_0	V3.0	•	•	•	•		•	•	•	•

DDS Compiler 4.0

このブロックは、[Xilinx Blockset] の [DSP] および [Index] ライブラリにリストされています。



DDS Compiler 4.0

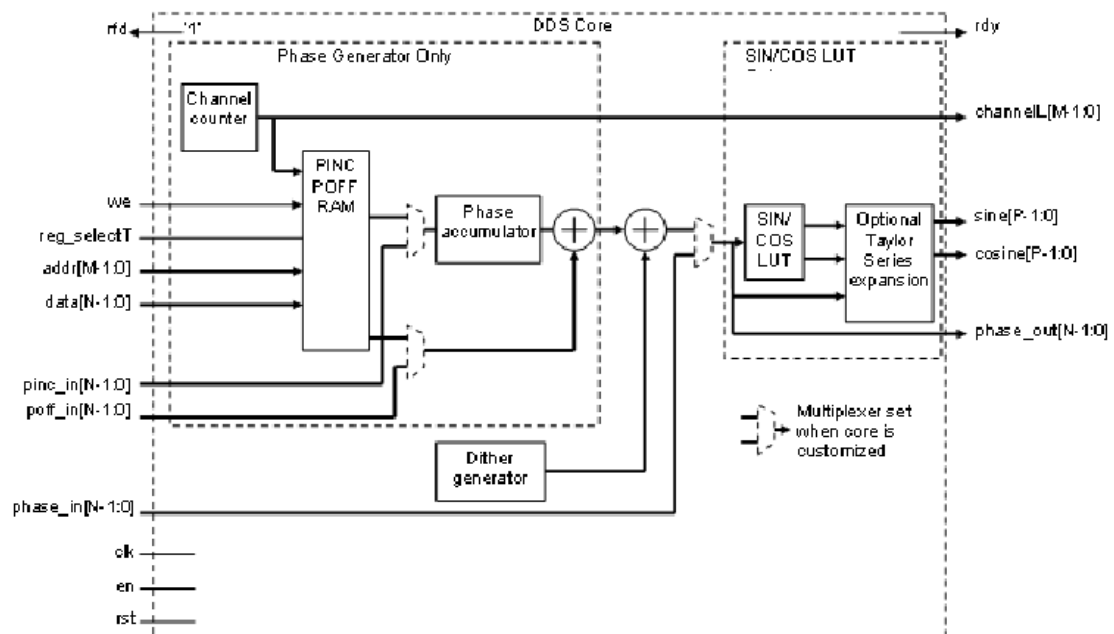
ザイリンクス DDS Compiler 4.0 ブロックの機能は DDS Compiler 3.0 と同じですが、次の新機能のサポートされている点が異なります。

- ブロックを Phase Generator または SIN/COS LUT のみとして使用するオプションを新しく追加。この機能により、Direct Digital Synthesizer を各アプリケーションの必要性に合わせてカスタマイズして構築できます。
- SFDR (Spurious Free Dynamic Range) を 120dB から 150 dB に増加
- システム レベルのパラメータ (SFDR、周波数解像度) またはハードウェア パラメータ (位相および出力幅) を使用した DDS をコンフィギュレーションするオプション
- 最大のパフォーマンスを得るために XtremeDSPTM スライスの使用をトレードオフするオプション
- 位相増分および位相オフセットを定数、プログラマブル、またはダイナミックとしてコンフィギュレーションするオプション

メモ：このブロックは、DDS Compiler 3.0 ブロックに置き換わるものです。DDS Compiler 4.0 は、前のバージョンに対してビット精度が高くありません。また、位相オフセットのレイテンシは、ストリーミング モードで使用しやすいように、位相増分のレイテンシに合わせて調整されます。この変更は、既存の [Programmable] モードおよび [Fixed] モードにも適用されます。

アーキテクチャの概要

DDS Compiler を理解するには、ブロックがどのように FPGA ハードウェアにインプリメントされるかを知る必要があります。次に DDS Compiler コアのブロック図を示します。このコアには、Phase Generator と SIN/COS LUT の主に 2 つのパーツが含まれます。これらのパーツは、個別に使用したり、オプションのデザイナー ジェネレータと一緒に使用して、DDS 機能を作成できます。時分割マルチチャネル機能は、コンフィギュラブルな位相増分パラメータとオフセット パラメータでサポートされます。



Phase Generator

Phase Generator にはアキュムレータとその後にオプションの加算器が含まれ、位相オフセットを追加します。コアがカスタマイズされると、位相増分とオフセットは、[Fixed]、[Programmable]、または `pinc_in` と `poff_in` 入力ポートでそれぞれ提供されるよう、個別にコンフィギュレーションできます。

[Programmable] に設定すると、レジスタが `addr`、`reg_select`、`we`、`data` 信号を含むバス インターフェイスを使用してインプリメントされます。アドレス入力の `addr` では、マルチチャネル モードの場合にデータを書き込むチャネルを指定します。`reg_select` では、データが位相増分かオフセットかを指定します。

[Fixed] に設定した場合、コアがカスタマイズされたときに DDS 出力周波数が設定されます。コアがデザインに埋め込まれてしまうと、この周波数は変更できません。

SIN/COS LUT と一緒に使用すると、オプションのディザ ジェネレータを増加した SFDR を提供するようにコンフィギュレーションできます。この際、増加されたノイズフロアが使用されます。

SIN/COS LUT

SIN/COS LUT は、Phase Generator の出力をサインおよびコサイン出力に変換します。半波長および 1/4 波長のストレージ方法を使用すると、効率的なメモリ使用率になります。両方の出力と論理否定の存在は、コアをカスタマイズするときにコンフィギュレーション可能です。精度は、オプションの Taylor Series Correction を使用して増加できます。これにより、高速動作で高い SFDR を達成するために、XtremeDSP をサポートする FPGA ファミリに XtremeDSP スライスが置かれます。

ブロック インターフェイス



DDS Compiler 4.0 ブロックには、次のようなポートがあります。

入力ポート

- **we** : 書き込みイネーブル (アクティブ High)。オフセット周波数メモリとプログラマブル周波数メモリのいずれか、または両方に対して書き込みをイネーブルにします。どちらのメモリに書き込まれるかは、`reg_select` ポートの値によって決まります。下位の LogiCORE の `we` ポートにマップされます。
- **reg_select** : 位相増分 メモリ (PINC) と位相オフセット (POFF) メモリへの書き込みに使用するアドレス セレクトです。`reg_select=0` で PINC メモリが選択されます。`reg_select=1` の場合、POFF メモリが選択されます。このポートは、[Phase Increment] と [Phase Offset] が [Programmable] の場合にのみ現れます。
- **addr** : このバスは、現在選択しているメモリに対して最大 16 チャネルまでアドレス指定するためのもので、ビット数は 2 チャネルの場合が 1、3 または 4 チャネルの場合が 2、5 ~ 8 チャネルの場合が 3、9 ~ 16 チャネルの場合が 4 になります。

- **data** : 時分割のデータ バスです。このポートは、プログラマブルな位相増分メモリおよびプログラマブルな位相オフセット メモリに値を送信するために使用されます。入力値は、位相角度を表します。この入力、符号なしまたは符号付きの純粋な少数の量になることがあります。位相増分または位相オフセットを提供する場合、位相はサイクルの少数部分として入力されます。つまり、18 ビット位相の場合、タイプは **UFix 18.18** または **Fix 18.18** で、それぞれ範囲は $0 \leq \text{phase} < 1.0$ または $-0.5 \leq \text{phase} < 0.5$ になります。位相増分の場合、提供される少数は、コアにチャンネルごとに送信されるクロックのレートに対する出力周波数でもあり、コアに送信されるクロックのレートは、チャンネル数で割られます。
- **rst** : 同期リセットです。1 の場合、ブロックの内部メモリがリセットされます。**POFF** および **PINC** メモリはリセットされません。下位の **LogiCORE** の **SCLR** (同期クリア) 入力にマップされます。
- **en** : ユーザー イネーブルです。1 の場合、ブロックがアクティブになります。下位の **LogiCORE** の **CE** ポートにマップされます。**POFF** および **PINC** メモリの書き込みには適用されません。
- **phase_in** : **DDS Compiler** が **SIN_COS_LUT_only** としてコンフィギュレーションされると使用されます。これは、**Phase Generator** で作成される位相信号を置き換える位相入力です。この入力、符号なしまたは符号付きの純粋な少数量のどちらかになり、サイクルの少数部分として位相を提供します。
- **pinc_in** : 位相増分のストリーミング入力です。この入力を使用すると、**DDS** 出力周波数が簡単に変更できます。この入力、符号なしまたは符号付きの純粋な少数量のどちらかになり、サイクルの少数部分として位相増分を提供します。これは、チャンネルごとにコアに送信されるクロック レートの小数部分と同じように、出力周波数でもあります。
- **poff_in** : 位相オフセットのストリーミング入力です。この入力を使用すると、**DDS** 出力位相が簡単に変更できます。この入力、符号なしまたは符号付きの少数量のどちらかになり、サイクルの少数部分として位相を提供します。

出力ポート

- **rdy** : アクティブ **High** で出力データが準備できているかどうかを示します。出力サンプルが有効になったことを示します。
- **rfd** : アクティブ **High** でデータが準備できているかどうかを示します。多くのザイリンクス **LogiCORE** に含まれるデータ フロー制御信号です。**DDS** のコンテンツは、ほかの **LogiCORE** コアと合わせるためだけに提供されます。このオプションのポートは、常に **VCC** に接続されます。
- **channel** : チャンネルのインデックスです。下位のコアがマルチチャンネル用にコンフィギュレーションされている場合に、どのチャンネルが出力で使用可能なのかを示します。これは、符号なしの値になります。幅は **[Basic]** タブの **[Number of Channels]** で指定したチャンネル数で決まります。
- **sine** : サイン出力の値です。下位の **LogiCORE** の **SINE** 出力にマップされます。
- **cosine** : コサイン出力の値です。下位の **LogiCORE** の **COSINE** 出力にマップされます。
- **phase_out** : **[Phase_Generator_only]** オプションをオンにすると、表示されます。この出力は、その他すべての場合でオプションになります。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、**Simulink** モデル内でブロックをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

[Configuration Options] : このパラメータを使用すると、DDS の 2 パーツが個別に、または一緒にインスタンス化できます。次のいずれかを選択します。

- Phase_Generator_and_SIN_COS_LUT
- SIN_COS_LUT_only
- Phase_Generator_only

システム要件

- ◆ [System Clock (Mhz)] : アーキテクチャを決定し、指定した出力周波数から位相増分を計算するために、ブロックに供給するクロックの周波数を指定します。この値は、システムクロックの固定比率になります。
- ◆ [Number of Channels] : チャンネルは、DDS で時分割され、各チャンネルで有効なクロックに影響を与えます。DDS では、1 ~ 16 の時分割チャンネルまでサポートされます。

[Parameter Selection] : [Choose System_Parameters] または [Hardware_Parameters] を選択します。

システム パラメータ

- ◆ [Spurious Free Dynamic Range (dB)] : DDS で生成されるトーンのターゲット純度。これにより、出力幅、内部バス幅、さまざまなインプリメンテーションが設定されます。
- ◆ [Frequency Resolution (Hz)] : PINC および POFF 値の精度を設定します。精度が高いほど、大きいアキュムレータが必要になり、精度が低いほど、ハードウェア リソースのコストが低くて済みます。

[Noise Shaping] : [None]、[Phase_Dithering]、[Taylor_Series_Corrected]、[Auto] のいずれかを選択します。

[Configuration Options] に [SIN_COS_LUT_only] を選択した場合は、[None] か [Taylor_Series_Corrected] しか設定できません。[Phase_Generator_Only] をオンにした場合は、[None] しか選択できません。

ハードウェア パラメータ

- ◆ [Phase Width] : 周波数解像度と同じで、内部位相計算の幅を設定します。
- ◆ [Output Width] : SFDR と大体同じで、出力精度と使用可能な最小位相幅を設定します。ただし、出力の正確さは [Noise Shaping] の設定にも影響を受けます。

[Output Selection] : サイン、コサイン、またはその両方 (Sine_and_Cosine) など、ブロックが計算する関数を指定します。

極性

- ◆ [Negative sine] : サイン出力をネゲートします。
- ◆ [Negative cosine] : コサイン出力をネゲートします。

振幅モード

- [Full_Range] : 使用可能な最大振幅が選択されます。
- [Unit_Circle] : 調度 2 のべき乗の振幅 (Full_Range 振幅の約半分) が選択されます。

[Use explicit period] : オンにすると、DDS Compiler 4.0 は下のダイアログ ボックスで指定した明示的なサンプル周期を使用します。

[Implementation] タブ

インプリメンテーション オプション

- ◆ [Memory Type] : [Auto]、[Distributed_ROM]、または [Block_ROM] のいずれかを選択します。
- ◆ [Optimization Goal] : [Auto]、[Area]、または [Speed] のいずれかを選択します。
- [DSP48 Use] : [Minimal] か [Maximal] を選択します。[Maximal] に設定すると、XtremeDSP スライスが使用され、最大のパフォーマンスが達成されます。

レイテンシ オプション

- ◆ [Auto] : DDS は、最適なパフォーマンスになるように完全にパイプライン処理されます。
- ◆ [Configurable] : 下の [Latency] プルダウン メニューで、より少ないパイプライン ステージを指定できるようになります。これにより、消費されるリソースが少なくなります。

オプションのピン

- ◆ [Has phase out] : オンにすると、DDS に phase_output ポートが含まれるようになります。これは、DDS の Phase_Generator 半分の出力なので、サイン/コサイン ルックアップ テーブルのレイテンシ分、サインおよびコサイン出力よりも先行します。
- ◆ rfd : オンにすると、DDS に rfd ポートが含まれるようになります。This is for completeness. DDS は常に data、pinc_in、poff_in 用に準備されています。
- ◆ rdy : オンにすると、DDS にサインおよびコサイン出力を有効にする rdy 出力ポートが含まれます。
- ◆ [Channel Pin] : オンにすると、DDS Compiler に sine および cosine ポート出力の属するチャンネルを許可するチャンネル (出力) ポートが含まれます。

[Output Frequency] タブ

- [Phase Increment Programmability] : 位相増分を [Fixed]、[Programmable]、または [Streaming] に指定します。[Programmable] を選択すると、channel、data、we 入力ポートがブロックに追加されます。

[Basic] タブの [Configuration Options] フィールドの [Phase_Generator_and_SIN_COS_LUT] をオンにし、[Basic] タブの [Parameter Selection] を [Hardware Parameters] に、[Phase Offset Angles] タブの [Phase Increment Programmability] フィールドを [Fixed] または [Programmable] に指定すると、次のフィールドが表示されます。

- ◆ [Output frequencies (Mhz)] : チャンネルごとに独立した周波数を配列に入力できます。このフィールドは、[Basic] タブの [Parameter Selection] を [System Parameters] に、[Phase Increment Programmability] を [Fixed] または [Programmable] に設定すると表示されます。
- ◆ [Phase Angle Increment Values] : [Basic] タブの [Configuration Options] フィールドの [Phase_Generator_and_SIN_COS_LUT] をオンにし、[Basic] タブの [Parameter Selection] を [Hardware Parameters] に、[Phase Offset Angles] タブの [Phase Increment Programmability] フィールドを [Fixed] または [Programmable] に指定すると、このフィールドが表示されます。入力する値はバイナリ (2 進数値) にする必要があります。範囲は、0 からアキュムレータの重み (例 : $2^{\text{Phase_Width}-1}$) の間になります。

[Phase Offset Angles] タブ

- [Phase Offset Programmability] : 位相オフセットを [None]、[Fixed]、[Programmable]、または [Streaming] に指定します。[Fixed] または [Programmable] を選択すると、channel、data、we 入力ポートがブロックに追加されます。
- ◆ [Phase Offset Angles (x2pi radians)] : チャネルごとに独立したオフセットを配列に入力できます。入力した値は、2p ラジアンで乗算されます。このフィールドは、[Basic] タブの [Parameter Selection] を [System Parameters] に、[Phase Increment Programmability] を [Fixed] または [Programmable] に設定すると表示されます。
- ◆ [Phase Angle Offset Values] : チャネルごとに独立したオフセットを配列に入力できます。入力した値は、2p ラジアンで乗算されます。このフィールドは、[Basic] タブの [Parameter Selection] を [Hardware Parameters] に、[Phase Increment Programmability] を [Fixed] または [Programmable] に設定すると表示されます。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

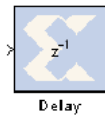
ザイリンクス LogiCORE

このブロックでは、ザイリンクス LogiCORE DDS Compiler v4.0 コアが使用されます。

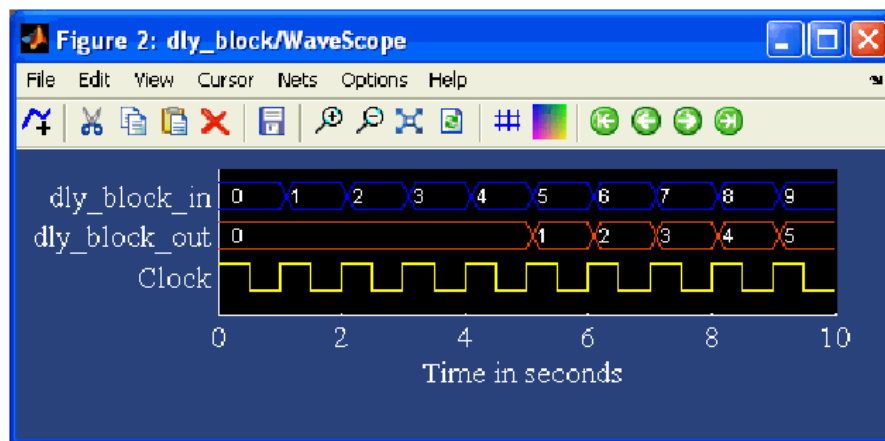
System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
DDS Compiler 4.0	DDS Compiler	V4.0	•	•	•	•	•	•	•	•	•	•

Delay

このブロックは、[Xilinx Blockset] の [Basic Elements]、[Memory]、[Index] ライブラリにリストされています。



Delay ブロックは、L サイクルの固定遅延をインプリメントします。ブロックには、遅延の値が Z 変換を表す z^{-L} で表示されます。ブロックの入力へのデータは、L サイクル後の出力に送信されます。出力データのレートとデータ型は、入力から伝送される値になります。このブロックは、主に回路のほかの部分にある同一のパイプライン遅延に使用されます。Delay ブロックは、Register ブロックとは異なります。Register ブロックの場合は、レイテンシは 1 サイクルしか使用できず、初期値パラメータが含まれます。Delay ブロックでは、レイテンシを指定できますが、0 以外の初期値は使用できません。次の図は、L=4 および 周期=1 秒の場合の Delay ブロックを示しています。



ランタイム中に調整する必要がある遅延がある場合は、Addressable Shift Register ブロックを使用してください。整数値のクロック サイクル以外の遅延タイプはサポートされていないので、少数の例外を除いて、同期デザインでは使用できません。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Latency]: 遅延のサイクル数を指定します。[Provide enable port] をオフにする場合は、レイテンシを 0 にする必要があります。レイテンシには、負の値は使用できません。レイテンシが 0 の場合、Delay ブロックはロジック合成中にワイヤにまとめられます。レイテンシが L=1 に設定されると、合成でブロックは通常フリップフロップになります (データ幅が 1 より大きい場合は、複数のフリップフロップになります)。

[Implementation] タブ

[Implementation] タブからは、次のようなパラメータを設定できます。

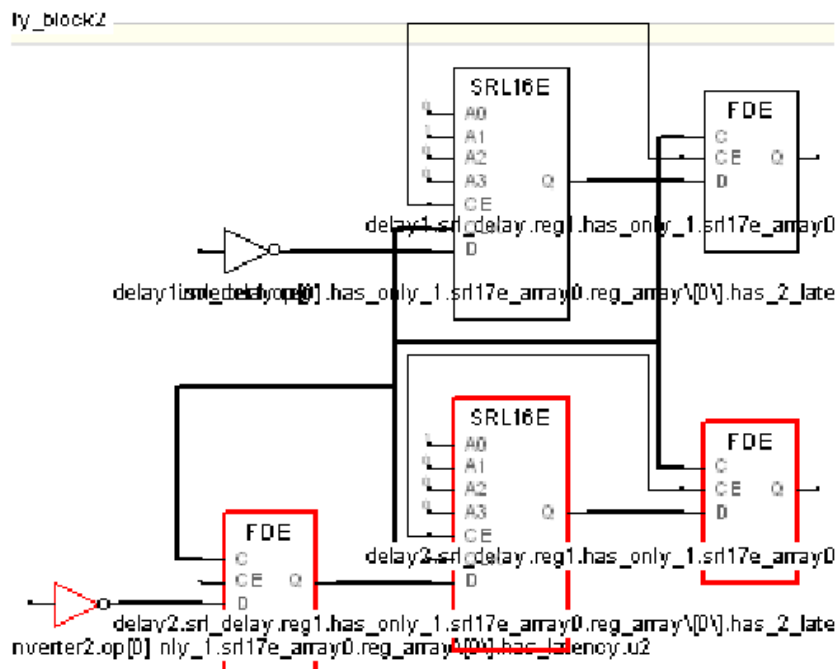
- [Implement using behavioral HDL] : ビヘイビア レベルの HDL がインプリメンテーションに使用されます。 これにより、ダウンストリームのロジック合成ツールで最適なインプリメンテーションが選択されるようになります。 このブロックで使用されるその他のパラメータは、「ブロックのパラメータ ダイアログ ボックスの共通オプション」で説明されています。

ビヘイビア レベルの HDL を使用したロジック合成

ダウンストリームのロジック合成ツールに Synplify Pro を使用する場合は、こちらの設定が推奨されます。 ロジック合成ツールは、指定どおりに遅延をインプリメントし、遅延ラインをブロック RAM、DSP48、またはエンベデッド IOB のフリップフロップなどの前後に移動させるような最適化を実行します。また、選択したアーキテクチャに基づいて、ロング遅延ラインに専用 SRL のカスケード出力を使用したり、パス遅延に基づいて遅延ラインの片方または両方の終端にフリップフロップを使用したりといった最適化も実行します。この設定を使用すると、ロジック合成ツールが高度であれば、遅延ラインの一部を組み合わせてロジック部分に戻して、タイミングを変更することもできます。

構造レベルの HDL を使用したロジック合成

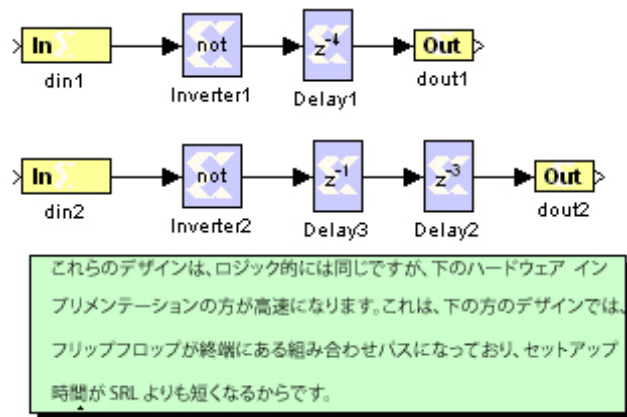
[Implement using behavioral HDL] をオフにすると、構造レベルの HDL が使用されます。これはデフォルト設定で、既知の柔軟性の少ないインプリメンテーションが実行されます。こちらの設定は、XST を使用する場合に向いています。通常は、この設定にすると、L-1 サイクルの SRL (シフトレジスタ LUT) 遅延の後にフリップフロップを付けた (SRL とフリップフロップは同一スライスにパッキング) 構造レベルの HDL が作成されます。レイテンシが L=17 よりも大きい場合、専用のカスケード配線を使用していなくても、複数の SRL/フリップフロップがカスケード接続されます。次の例は、レイテンシが L=32 の場合の 1 ビット幅の Delay ブロックの合成結果です。



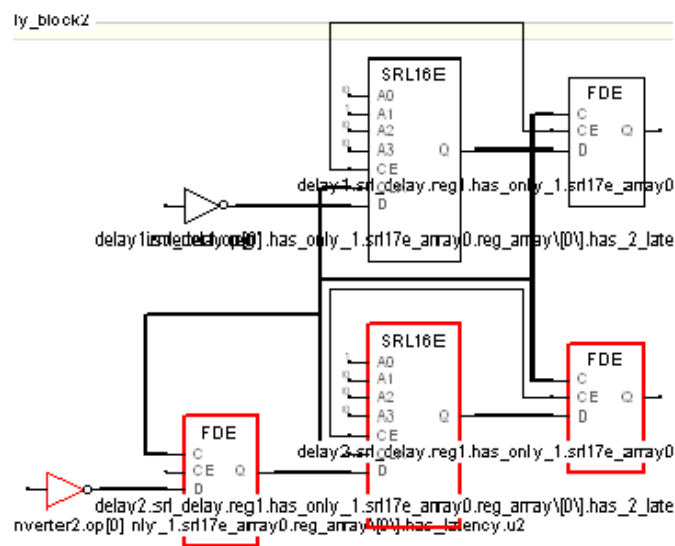
最初の SRL では 16 サイクルの遅延が追加され、接続されたフリップフロップで遅延がさらに 1 サイクル追加されます。2 つ目の SRL では、14 サイクルの遅延が追加されます。14 サイクルなのは、アドレスが {A3,A2,A1,A0}=1101 (2 進数) = 13 に設定されており、SRL を通った後のレイテンシはアドレスに 1 サイクル足した値になるからです。最後のフリップフロップで遅延が 1 サイクル追加されるので、トータルで $L=16+1+14+1=32$ サイクルの遅延になります。

SRL を使用すると、ザイリンクス アーキテクチャで遅延を効率的にインプリメントできます。SRL とそれに接続された単一のロジックセルを含むフリップフロップは、17 サイクルの遅延をインプリメントできますが、フリップフロップのみを含む遅延ラインの場合は、ロジックセルごとに 1 サイクルの遅延しかインプリメントできません。

SRL のセットアップ タイムはフリップフロップのセットアップ タイムより長いので、遅延ブロックの前に組み合わせパスがある高速デザインでは、構造レベルの HDL 設定を使用する場合、Delay ブロックの前にレイテンシ $L=1$ の付いた遅延ブロックを 1 つ追加すると効率的です。これにより、クリティカルパスが SRL の長いセットアップ時間が原因で問題になることはなくなります。次の図は、その例です。



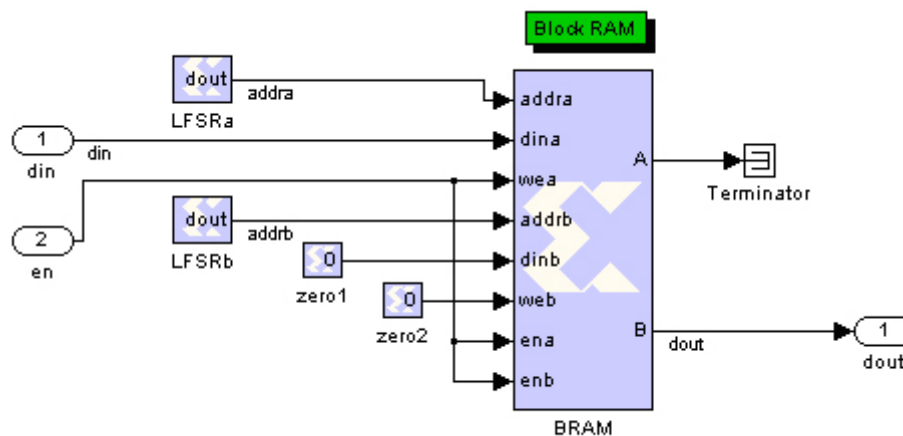
どちらのデザインも合成すると、次の図のようになります。高速なデザインの方を赤でハイライトしています。



高速な方のデザインでは、Inverter2 のレイテンシを 1 に設定し、Delay 3 を削除したものと同等ですが、Inverter2 のレイテンシを 4 に設定し、Delay ブロックを削除したものと同等ではありません。この場合、合成結果は、上の遅い方のデザインと同じになります。

大きい遅延のインプリメント

たとえば、128 サイクルを超えるような大きい遅延には、特に幅の大きいバスを使用した場合など、ブロック RAM ベースの遅延ブロックを使用した方がよいことがあります。このような遅延ブロックは、ザイリンクスの一般的なファブリックの一部である SRL を使用してインプリメントされます。遅延が非常に大きい場合は、エンベデッド ブロック RAM に遅延をインプリメントすると、ファブリック スペースを節約できます。このように遅延を付けると、ブロック RAM のデュアル ポートの特性を生かすことができ、固定またはランタイム変数遅延を使用してインプリメントすることができます。このようなブロックは、基本的に複数のアドレス カウンタが接続されたブロック RAM です。次の図は、デザインを高速にするため、アドレス カウンタに LFSR (Linear Feedback Shift Register) を使用して大きい遅延をインプリメントした新しい方法ですが、従来のカウンタも使用することができます。カウンタ間の値の違い (RAM レイテンシを減算した値) が、遅延ラインのレイテンシ L です。



この遅延エレメントでは、ブロック RAM を使用して大きな遅延をインプリメントしています。アドレスは、ガロアフィールドでカウンタとして使用される LFSR から指定されます。B ポートのアドレスは、 $\alpha^0 (=1)$ で開始するカウンタで生成されます (α はガロアフィールドのプリミティブ エレメント)。A ポートのアドレスは、 $\alpha^L (L-R)$ で開始するカウンタで生成されます (L はレイテンシ合計、 R は RAM のレイテンシ)。

リセット可能な遅延と初期値

遅延ラインを完全に 0 にリセット可能にする必要がある場合は、 L レジスタの文字列を使用して遅延をインプリメントするか、遅延ラインの内容が一気に消去される間に強制的に出力を 0 にするような回路を作成します。

Delay ブロックでは、初期値は設定できませんが、Addressable Shift Register ブロックではサポートされます。このブロックは、通常固定アドレスと使用されると Delay ブロックと同じ働きをし、SRL ベースの遅延ラインに同期します。初期値は、初期化にのみ使用され、リセットには使用されません。Addressable Shift Register ブロックを構造レベルの HDL モードで使用すると ([Use behavioral HDL] チェック ボックスをオフにすると)、遅延ラインの終端がフリップフロップにはな

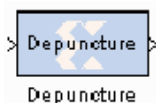
らないので、速度がかなり遅くなってしまいます。この場合、ビヘイビア記述にするか、Addressable Shift Register の前に Register ブロックまたは Delay ブロックを配置すると問題を回避できます。

ザイリンクス LogiCORE

Delay ブロックではザイリンクス LogiCORE は使用されませんが、ザイリンクス デバイスの SRL16 の機能を使用すると、効率的にマップできます。

Depuncture

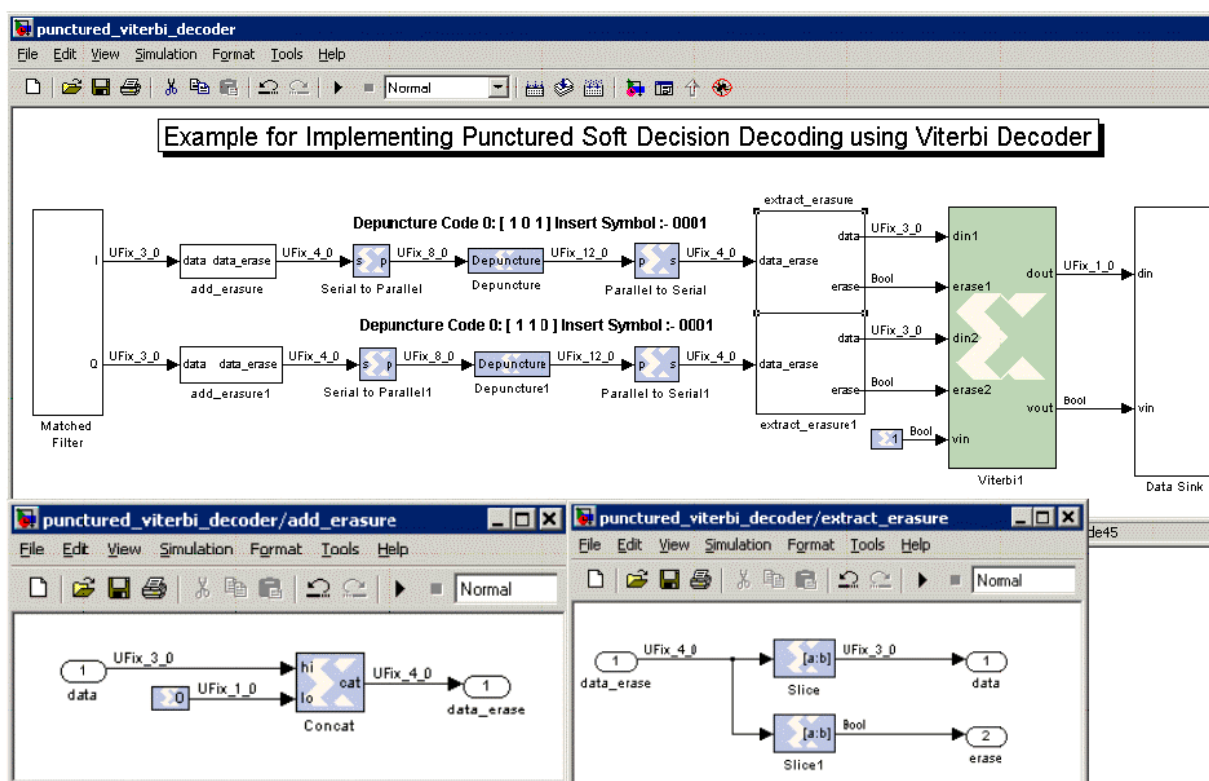
このブロックは、[Xilinx Blockset] の [Communication] および [Index] ライブラリにリストされています。



ザイリンクスの Depuncture ブロックを使用すると、デパンクチャ符号で指定した位置の入力データにシンボルを挿入できます。

このブロックは、UFixN_0 型 (N は挿入文字列 x の長さ (デパンクチャ符号の 1 の数) のデータを入力し、UFixK_0 型 (N はデパンクチャ符号の長さで乗算された挿入文字列の長さ) のデータを出力します。

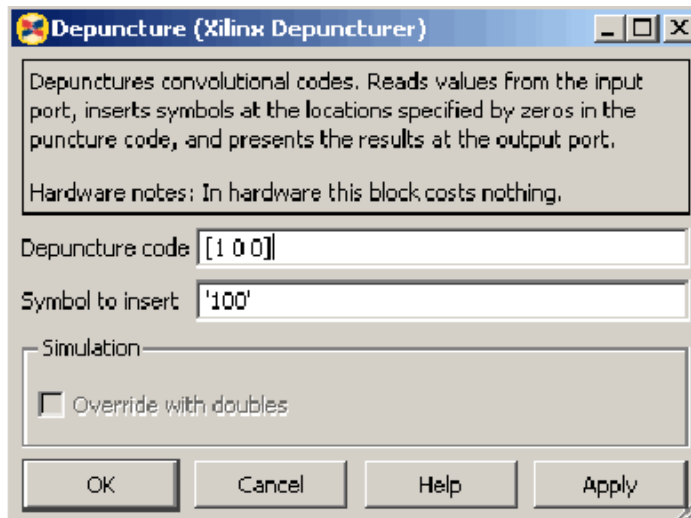
Depuncture ブロックは、パンクチャドたたみ込み符号をデコードするために使用します。次の図は、このブロックを使用してパンクチャドたたみ込み符号の軟判定ビット (Viterbi) 復号をインプリメントした例を示しています。



上図は、add_erasure サブシステムに接続された Matched Filter ブロックを示しています。add_erasure サブシステムでは、入力データに 0 が付けられ、非消去信号としてマークされます。この add_erasure サブシステムからの出力が Serial to Parallel ブロックに送信され、Serial to Parallel ブロックで 2 つの連続する軟入力に連結され、8 ビット ワードで Depuncture ブロックに送信されます。Depuncture ブロックは、符号 0 ([1 0 1]) の場合 MSB から 4 ビット後にシンボル 0001 を、符号 1 ([1 1 0]) の場合 MSB から 8 ビット後に挿入し、12 ビット ワードを形成します。この後、Depuncture ブロックの出力は、Parallel to Serial ブロックを使用して 4 ビット ワードにシリアルライズされます。extract_erasure サブシステムは、入力された 4 ビット ワードの 3 ビットを MSB から抽出し、軟判定の入力データ ワードを形成し、LSB から 1 ビットを抽出して Viterbi デコーダの消去信号を形成します。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。



Puncture ブロックのパラメータを次に示します。

- [Depuncture code] : 入力に挿入する文字列のデパンクチャ パターンを指定します。
- [Symbol to insert] : デパンクチャ符号に挿入する 2 進数ワードを指定します。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

Disregard Subsystem

このブロックは、[Xilinx Blockset] の [Tools] および [Index] ライブラリにリストされています。



Disregard Subsystem
For Generation

このブロックは、**System Generator** バージョン 6.2 から廃止されています。

このブロックは、今後のバージョンで削除される可能性があります。このブロックの機能は、**Simulink** のコンフィギャブル サブシステムから使用できます。詳細は、「[コンフィギャブル サブシステムと System Generator](#)」を参照してください。コンフィギャブル サブシステムの方が **Disregard Subsystem** ブロックよりも多くの利点があります。

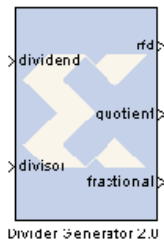
Disregard Subsystem ブロックは、モデルのサブシステムに配置して、**System Generator** でそのサブシステムのハードウェアが生成されないようにするために使用できます。このブロックを **Simulation Multiplexer** ブロックと共に使用すると、デザインの一部に対して代替シミュレーション モデルを構築できます。たとえば、ブラック ボックスのシミュレーション モデルを提供することができます。

このブロックには、パラメータはありません。

Divider Generator 2.0

メモ：このブロックは、Divider Generator 3.0 ブロックに置き換わっています。

このブロックは、[Xilinx Blockset] の [DSP]、[Math] および [Index] ライブラリにリストされています。



ザイリンクスの Divider Generator 2.0 ブロックは、Radix-2 の非回復型除算、またはプリスケールリングを用いた高基数除算に基づいて整数除算を行う回路を作成します。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

共通オプション

- [Dividend and quotient width] : (整数) 2 ～ 32 (基数-2)。2 ～ 54 (高基数)。被除数と商の両方の幅を指定します。
- [Divisor width] : (整数) : 2 ～ 32 (基数-2)。2 ～ 54 (高基数)。
- [Algorithm Type] :
 - ◆ [Radix-2] : 整数値のオペランドを使用した Radix-2 の非回復型除算で、余りが生成できるようになります。この方法は、オペランドの幅が約 16 ビット未満の場合に推奨されます。符号なしと符号付き (2 の補数) 両方の除数および被除数入力サポートされます。
 - ◆ [High Radix] : プリスケールリングを用いた高基数除算です。オペランドの幅が 16 ビットを超える場合に推奨されます。ただし、インプリメンテーションでは DSP48 (または同種の DSP48) プリミティブが必要となります。サポートされる除数および被除数入力は符号付き (2 の補数) のみになります。
- [Remainder type] :
 - ◆ [Remainder] : Radix-2 でのみサポートされます。
 - ◆ [Fractional] : 分数ポート出力のビット数を指定します。
- [Fractional Width] : 余りのタイプに [Fractional] を選択した場合に、分数ポート出力のビット数を指定します。

Radix-2 のオプション

- [Clocks per division] : 新規データが入力されて出力されるまでのクロックの間隔を指定します。

高基数のオプション

- [Detect divide by zero] : コアに 0 での除算を示す出力ポートを含めるかどうかを指定します。

- **[Latency configuration] : [Automatic]** (完全にパイプライン) または **[Manual]** (次のフィールドで指定) のいずれかを指定します。
- **[latency]** : クロック イネーブルの付いたクロック サイクルに対し、入力から出力までの正確なレイテンシを指定します。

オプション ポート

- **en** : イネーブル ポートを追加します。
- **rst** : リセット ポートを追加します。

このブロックで使用されるその他のパラメータは、「[System Generator](#) でサポートされる [Simulink ブロック](#)」で説明されています。

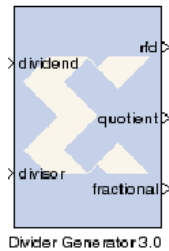
ザイリンクス LogiCORE

このブロックは、次のザイリンクス LogiCORE コアを使用します。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan			Virtex	
			3、3E	3A	3A DSP	4	5
Divider Generator 2.0	Divider v2.0	V2.0	•	•	•	•	•

Divider Generator 3.0

このブロックは、[Xilinx Blockset] の [DSP]、[Math] および [Index] ライブラリにリストされています。



ザイリンクスの Divider Generator 3.0 ブロックは、Radix-2 の非回復型除算、またはプリスケールリングを用いた高基数除算に基づいて整数除算を行う回路を作成します。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

共通オプション

- [Dividend and quotient width] : (整数) 2 ~ 32 (基数-2)。2 ~ 54 (高基数)。被除数と商の両方の幅を指定します。
- [Divisor width] : (整数) : 2 ~ 32 (基数-2)。2 ~ 54 (高基数)。
- [Algorithm Type] :
 - ◆ [Radix-2] : 整数値のオペランドを使用した Radix-2 の非回復型除算で、余りが生成できるようになります。この方法は、オペランドの幅が約 16 ビット未満の場合に推奨されます。符号なしと符号付き (2 の補数) 両方の除数および被除数入力サポートされます。
 - ◆ [High Radix] : プリスケールリングを用いた高基数除算です。オペランドの幅が 16 ビットを超える場合に推奨されます。ただし、インプリメンテーションでは DSP48 (または同種の DSP48) プリミティブが必要となります。サポートされる除数および被除数入力は符号付き (2 の補数) のみになります。
- [Reminder type] :
 - ◆ [Remainder] : Radix-2 でのみサポートされます。
 - ◆ [Fractional] : 分数ポート出力のビット数を指定します。
- [Fractional Width] : 余りのタイプに [Fractional] を選択した場合に、分数ポート出力のビット数を指定します。

Radix2 のオプション

- [Clocks per division] : 新規データが入力されて出力されるまでのクロックの間隔を指定します。

高基数のオプション

- [Detect divide by zero] : コアに 0 での除算を示す出力ポートを含めるかどうかを指定します。
- [Latency configuration] : [Automatic] (完全にパイプライン) または [Manual] (次のフィールドで指定) のいずれかを指定します。

- **[latency]** : クロック イネーブルの付いたクロック サイクルに対し、入力から出力までの正確なレイテンシを指定します。

オプション ポート

- **en** : イネーブル ポートを追加します。
- **rst** : リセット ポートを追加します。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

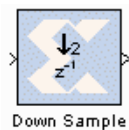
ザイリンクス LogiCORE

このブロックは、次のザイリンクス LogiCORE コアを使用します。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
Divider Generator 3.0	Divider Generator	V3.0	•	•	•	•	•	•	•	•	•	•

Down Sample

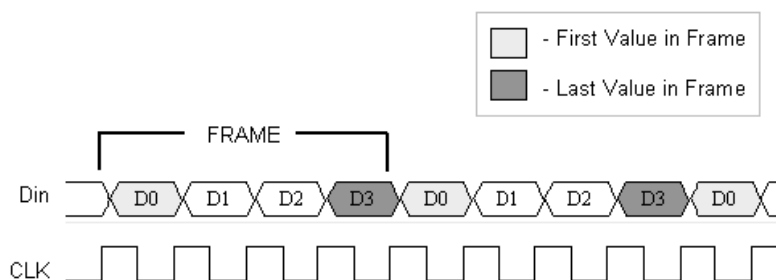
このブロックは、[Xilinx Blockset] の [Basic Elements] ライブラリと [Index] ライブラリにリストされています。



ザイリンクスの **Down Sample** ブロックは、ブロックが配置された箇所のサンプルレートを削減するために使用します。

入力信号は、フレームの最初か最後のいずれかの値で等区間でサンプリングされます。サンプリングされた値は出力ポートに送られると、次のサンプルが取り出されるまで保持されます。

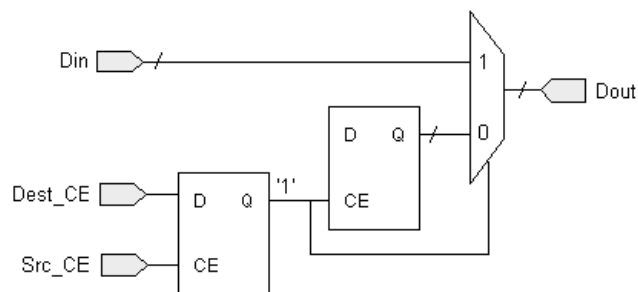
1 つの **Down Sample** フレームには、1 入力サンプル (サンプリング レート 1) が含まれます。次の図は、サンプリング レート 4 でコンフィギュレーションされた **Down Sample** ブロックのフレーム例を示しています。



Down Sample ブロックは、後で説明する 3 つのインプリメンテーション方法のいずれかを使用してハードウェアで認識されます。インプリメンテーションの効率は、方法によって異なります。このブロックは、**Src_CE** と **Dest_CE** の 2 つのクロック イネーブル信号をハードウェアで受信します。**Src_CE** の方が速く、入力データ ストリーム レートに対応します。**Dest_CE** はそれよりも遅く、出力ストリーム レート (ダウン サンプル データ) に対応します。これらのイネーブル信号でハードウェアのレジスタ サンプリングが制御されます。

レイテンシ 0 の場合

レイテンシ 0 の Down Sample ブロックは、フレームの最初の値をサンプリングするようにコンフィギュレーションされる必要があります。入力フレームの最初のサンプルは、MUX を通って出力ポートに渡されます。レジスタは、最初のサンプルの間にこの値をサンプリングし、MUX はフレームの 2 つ目のサンプルの開始時にレジスタ出力に切り替わります。この結果、フレームの最初のサンプルがフレーム全体が処理される間に出力ポートに出力されます。MUX には Din から Dout までの組み合わせパスが使用されるので、このハードウェア インプリメンテーションが一番効率の悪い方法になります。デスティネーション クロック イネーブル (Dest_CE) のタイミングは、サンプル周期の最後ではなく、最初にアサートされるように、1 ビット レジスタで調整されます。このハードウェア インプリメンテーションは、次の図のようになります。

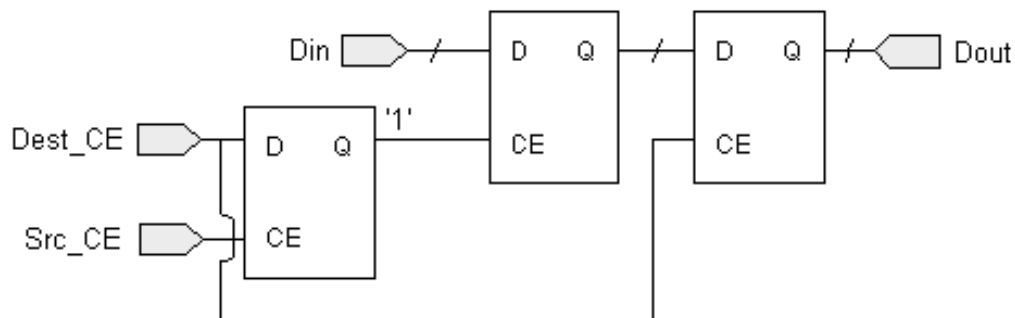


レイテンシがある場合

Down Sample ブロックが 0 より大きいレイテンシでコンフィギュレーションされる場合は、もっと効率の良いインプリメンテーションが使用されます。この場合、インプリメンテーション方法は 2 つあり、Down Sample ブロックがフレームの最初の値か最後の値のどちらをサンプリングするように設定されているかによって方法は異なります。

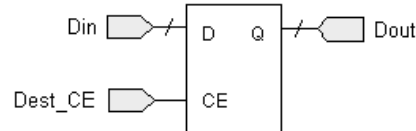
フレームの最初の値をサンプリングする場合

この場合、入力ストリームを正しくサンプリングするためにレジスタが 2 つ必要です。最初のレジスタは、入力フレームの開始時の入力をサンプリングするように調整されたクロック イネーブル信号でイネーブルになります。2 つ目のレジスタはサンプル周期の最後に最初のレジスタの内容をサンプリングし、出力データを正しく揃っているかどうかを確認します。



フレームの最後の値をサンプリングする場合

Down Sample ブロックがフレームの最後の値をサンプリングするようにコンフィギュレーションされる場合のインプリメンテーションは、最も効率の良いものになります。この場合、レジスタはフレームの最後で入力データをサンプリングし、その値を次のフレーム中に出力します。



ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Sampling rate (number of input samples per output sample)] : 2 以上の整数にする必要があります。これは、入力に対する出力サンプル周期の比率で、サンプル レート分周器の値になります。たとえば、2 と指定した場合は、入力サンプル レートは 2:1 の比率で分周されます。整数以外の比率を指定する場合は、Down Sample ブロックと Up Sample ブロックを組み合わせて使用してください。
- [Sample] : Down Sample ブロックは、フレームの最初の値か最後の値のいずれかでサンプリングを実行します。このパラメータでは、どちらの値がサンプリングされるかを指定します。

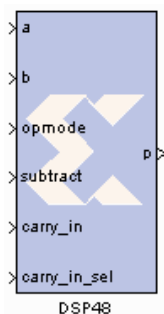
このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

Down Sample ブロックでは、ザイリンクス LogiCORE は使用されません。

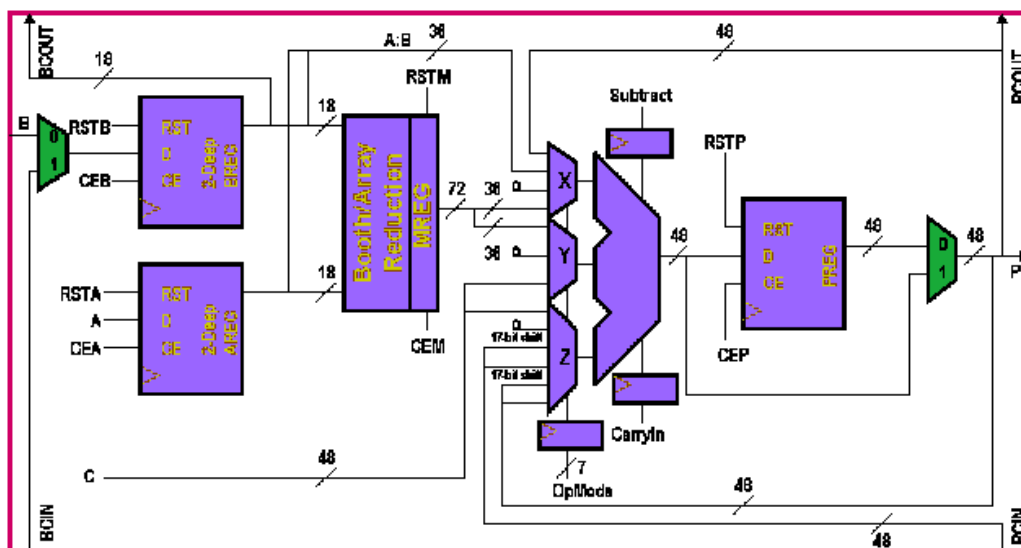
DSP48

このブロックは、[Xilinx Blockset] の [Index] および [DSP] ライブラリにリストされています。



ザイリンクスの DSP48 ブロックは、ザイリンクスの Virtex®-4 デバイスを使用する DSP アプリケーションを効率的に構築するためのブロックです。DSP48 では、18X18 ビットの符号付き乗算器と 48 ビット加算器が組み合わされており、加算器の入力はプログラマブル マルチプレクサで選択されます。

これらの演算は、ダイナミックに選択できます。オプションの入力と乗算器パイプライン レジスタだけでなく、subtract、carry_in、opmode ポートのレジスタも選択できます。DSP48 ブロックは、[Implementation] タブの [Use synthesizable model] をオンにした場合、DSP48 ハードウェア プリミティブを含まないデバイスをターゲットにすることもできます。



ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- **[B or BCIN input]** : B 入力を b ポートから直接取り込むか、カスケードされた bcin ポートから取り込むかを指定します。bcin ポートは、別の DSP48 ブロックにのみ接続できます。
- **[Consolidate control port]** : オンにすると、opmode、subtract、carry_in、carry_in_sel ポートを 1 つの 11 ビット ポートに組み合わせます。ビット 0 ～ 6 が opmode、ビット 7 が subtract ポート、ビット 8 が carry_in ポート、ビット 9 と 10 が carry_in_sel ポートになります。このオプションは、DSP48 命令を生成するのに Constant ブロックを使用した場合にのみ使用してください。
- **[Provide C port]** : オンにすると、c ポートが使用可能になります。オフの場合は、c ポートは 0 に接続されます。

- [Provide PCIN port] : オンにすると、pcin ポートが使用可能になります。pcin ポートは、別の DSP48 ブロックの pcout にのみ接続できます。
- [Provide PCOUT port] : オンにすると、pcout ポートが使用可能になります。pcout ポートは、別の DSP48 ブロックの pcin にのみ接続できます。
- [Provide BCOUT port] : オンにすると、bcout ポートが使用可能になります。bcout ポートは、別の DSP48 ブロックの bcin にのみ接続できます。
- [Provide global reset port] : オンにすると、rst ポートが使用可能になります。このポートは、パイプライン選択に基づいて、使用可能なリセット ポートすべてに接続されます。
- [Provide global enable port] : オンにすると、オプションの en ポート が使用可能になります。このポート は、パイプライン選択に基づいて、使用可能なイネーブル ポート すべてに接続されます。

[Pipelining] タブ

[Pipelining] タブからは、次のようなパラメータを設定できます。

- [Length of A pipeline] : 入力レジスタ A のパイプライン長を指定します。パイプラインの長さを 0 にすると、入力のレジスタが削除されます。
- [Length of B/BCIN pipeline] : b 入力のパイプライン長を b から読み込むか、bcin から読み込むかを指定します。
- [Pipeline C] : c ポートからの入力にレジスタを付けるかどうかを指定します。
- [Pipeline P] : p 出力と pout 出力にレジスタを付けるかどうかを指定します。
- [Pipeline multiplier] : 内部乗算器の出力にレジスタを付けるかどうか指定します。
- [Pipeline opmode] : opmode ポートにレジスタを付けるかどうかを指定します。
- [Pipeline subtract] : subtract ポートにレジスタを付けるかどうかを指定します。
- [Pipeline carry in] : carry_in ポートにレジスタを付けるかどうかを指定します。
- [Pipeline carry in sel] : carry_in_sel ポートにレジスタを付けるかどうかを指定します。

[Ports] タブ

[Ports] タブからは、次のようなパラメータを設定できます。

- [Reset port for A] : オンにすると、rst_a ポートが使用可能になります。1 に設定されると、ポート a のパイプライン レジスタがリセットされます。
- [Reset port for B] : オンにすると、rst_b ポートが使用可能になります。1 に設定されると、ポート b のパイプライン レジスタがリセットされます。
- [Reset port for C] : オンにすると、rst_c ポートが使用可能になります。1 に設定されると、ポート c のパイプライン レジスタがリセットされます。
- [Reset port for multiplier] : オンにすると、rst_m ポートが使用可能になります。1 に設定されると、内部乗算器のパイプライン レジスタがリセットされます。
- [Reset port for P] : オンにすると、rst_p ポートが使用可能になります。1 に設定されると、出力レジスタがリセットされます。
- [Reset port for carry in] : オンにすると、rst_carryin ポートが使用可能になります。1 に設定されると、carry_in のパイプライン レジスタがリセットされます。
- [Reset port for controls (opmode, subtract, carry_in, carry_in_sel)] : オンにすると、rst_ctrl ポートが使用可能になります。1 に設定されると、subtract、opmode、carry_in_sel レジスタのパイプライン レジスタがリセットされます。

- [Enable port for A] : オンにすると、ポート A のパイプライン レジスタのイネーブル ポート `ce_a` が使用可能になります。
- [Enable port for B] : オンにすると、ポート B のパイプライン レジスタのイネーブル ポート `ce_b` が使用可能になります。
- [Enable port for C] : オンにすると、ポート C のパイプライン レジスタのイネーブル ポート `ce_c` が使用可能になります。
- [Enable port for multiplier] : オンにすると、乗算器レジスタのイネーブル ポート `ce_m` が使用可能になります。
- [Enable port for P] : オンにすると、ポート P のパイプライン レジスタのイネーブル ポート `ce_p` が使用可能になります。
- [Enable port for carry in] : オンにすると、`carry_in` レジスタのイネーブル ポート `ce_carry_in` が使用可能になります。
- [Enable port for controls (opmode, subtract, carry_in, carry_in_sel)] : オンにすると、イネーブル ポート `ce_ctrl` と `ce_cinsub` が使用可能になります。ポート `ce_ctrl` は `opmode` と `carry_in_sel` レジスタを、`ce_cinsub` は `subtract` レジスタを制御します。

[Implementation] タブ

- [Use synthesizable model] : オンにすると、DSP48 が RTL 記述からインプリメントされ、DSP48 ハードウェアには直接マップされません。このオプションは、DSP48 ハードウェア プリミティブを含まないデバイス ファミリーをターゲットにした場合に使用すると便利です。
- [Use adder only] : オンにすると、ブロックは乗算器を使用せずに、最速のパフォーマンスを達成できるようにハードウェアで最適化されます。乗算器を使用する命令があると、シミュレーションでエラーが発生します。

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

関連項目

DSP48 ブロックの詳細については、次のトピックを参照してください。

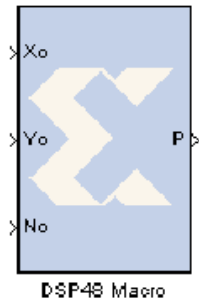
[DSP48 Macro](#)

[複数クロックのサイクル単位アイランドの生成](#)

[ザイリンクス XtremeDSP](#)

DSP48 Macro

このブロックは、[Xilinx Blockset] の [Index] および [Control Logic] ライブラリにリストされています。



DSP48 Macro ブロックは、DSP48、DSP48A、および DSP48E ブロックをデバイスに依存せずに抽象化したものです。デバイス別に DSP スライスを使用する代わりに、このブロックを使用すると、ザイリンクス デバイス間でデザインを動かしやすくなります。

このブロックは、コンパイル時に指定したターゲット デバイスによって、データ型の調整用に **Reinterpret** ブロックおよび **Convert** ブロックを使用したり、複数の **opmode** や入力の処理用にマルチプレクサを使用したり、レジスタを使用したりして 1 つの DSP48/DSP48A/DSP48E ブロックをラップします。

メモ：この後の説明では、DSP/DSP48A/DSP48E をまとめて DSP48 と記述します。

ブロック インターフェイス

DSP48 Macro ブロックの入力数と出力数は、ユーザーが指定したパラメータ値に従います。入力データ ポートは DSP48 Macro の [Instructions] フィールドに入力された **opmode** によって決まります。[Instructions] フィールドに **opmode** が複数指定される場合は、入力ポート **Sel** が表示されます。[Instructions] フィールドの詳細は、「DSP48 Macro ブロックの **opmode** の入力」のセクションを参照してください。

DSP48 Macro のすべてのコンフィギュレーションで表示されるポートは、出力データ ポート **P** だけです。出力ポート **PCOUT**、**BCOUT**、**ACOUT**、**CARRYOUT**、**CARRYCASCOUT** は、ユーザーの選択によって表示されます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Inputs to Port A] : XtremeDSP スライスのポート A またはポート A:B に接続される記号ポートの識別子 (オペランド) を指定します。指定したものは、[Instructions] フィールドに表示されます。
- [Inputs to port B] : ポート B に接続される記号ポートの識別子 (オペランド) を指定します。指定したものは、[Instructions] フィールドに表示されます。
- [Inputs to port C] : ポート C に接続される記号ポートの識別子 (オペランド) を指定します。指定したものは、[Instructions] フィールドに表示されます。
- [Instructions] : DSP48 Macro ブロックの命令を指定します。詳細は、「[DSP48 Macro ブロックの **opmode** の入力](#)」を参照してください。

[Pipelining] タブ

- [Pipeline Options] : XtremeDSP スライスのパイプライン オプションと、XtremeDSP スライスの各ポートのデータに付けるレイテンシを指定します。選択できるのは、[External Registers]、[No External Registers]、[Custom] のいずれかです。[External Registers] を選択すると、DSP48

Macro の下にあるマルチプレクサの出力にレジスタが付きます (これにより、高速処理が可能になります)。DSP48 Macro が DSP48 を加算器としてのみコンフィギュレーションする場合 ([Instructions] フィールドに入力された論理式から推論されます)、レイテンシは 3 です。それ以外の場合は、4 になります。[No External Registers] を選択した場合は、マルチプレクサの出力にはレジスタが付かず、DSP48 Macro のレイテンシは 2 になります。[Custom] を選択すると、XtremeDSP スライスの内外のレジスタ インスタンスすべてが [Custom Pipeline Options] フィールドの設定に基づいて推論されます。XtremeDSP スライスを加算器および乗算器として使用する命令がある場合は、このフィールドを必ず使用してください。

- [Custom Pipeline Options] : [Pipeline options] を [Custom] に設定した場合にのみ表示されるフィールドで、XtremeDSP スライスおよびマルチプレクサ レジスタ インスタンスを個別に制御できます。
- [Custom Pipeline Options ([A,B,C,P,Ctrl,M,MuxA,MuxB,MuxC,MuxCtrl])] : [Custom Pipeline Options] を 1 つの整数配列として指定できます。

[Ports] タブ

[Ports] タブでは、BCOUT、ACOUT、CARRYOUT、CARRYCASCOUT、PCOUT に加えて、さまざまな XtremeDSP スライスのリセット ポートおよびイネーブル ポートを使用するかどうかを指定できます。

[Implementation] タブ

- [Use DSP48] : Virtex-4、Virtex-5、または Spartan-3A DSP で XtremeDSP スライスを使用するかどうかを指定します。オフにすると、それ以外のデバイスで使用できる合成可能な XtremeDSP スライスのモデルが使用されます。

DSP48 Macro ブロックの opmode の入力

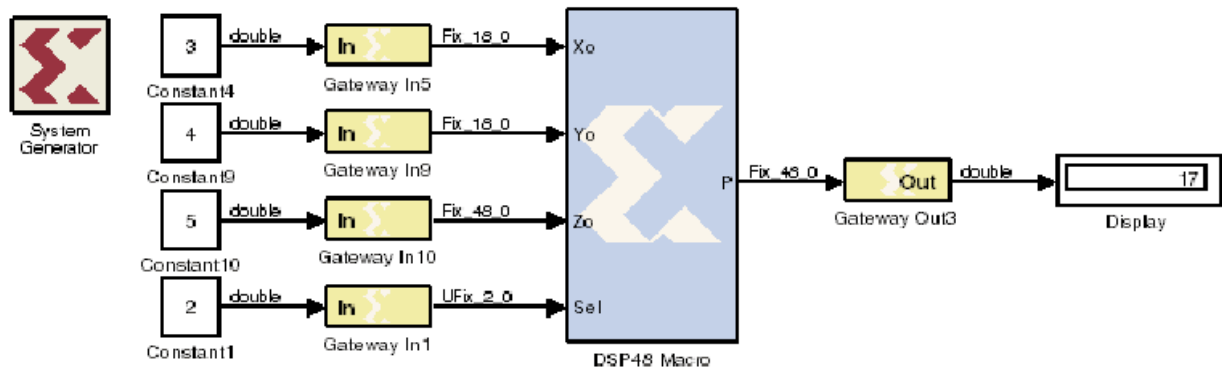
DSP48 では、opmode ポートへの入力によって異なる演算操作が可能です。このため、DSP48 はダイナミックな演算ユニットとして動作させることができます。DSP48 Macro を使用すると、DSP48 を簡単にダイナミックな演算ユニットとして使用できます。DSP48 Macro は、マルチプレクサを使用して複数のオペランドおよび opmode を順序付けし、データ ポートで信号を揃えます。オペランドと opmode の実行順序は、[Instructions] フィールドに入力された opmode の順序になります。[Instructions] フィールドには、最低で 1 つから最高で 8 つまでの opmode を含めることができます。ここでは、DSP48 Macro の [Instructions] フィールドに opmode を入力する際に発生する問題について説明します。

opmode の形式

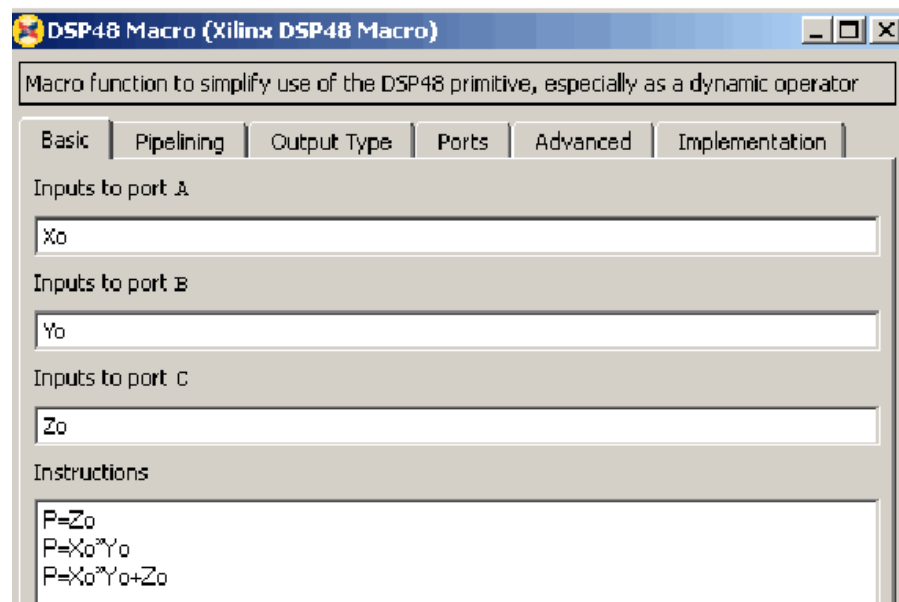
opmode を分割するには、改行文字を使用します。opmode はそれぞれ次の規則に従って入力する必要があります。

- 各 opmode は P に割り当てられ、P= で開始される必要があります。
- 演算子 (=) の後の式には、演算子 +/=/* を使用し、オペランドには記号ポート識別子 (「オペランドの形式」を参照) を使用する必要があります。
- DSP48 にインプリメントできる opmode のみが有効です。DSP48 Macro でサポートされる opmode は、この後の表にリストされています。

次の図に示すような単純なモデルがあるとします。DSP48 Macro には、Xo、Yo、Zo という 3 つの入力があります。ブロック ダイアログ ボックスで複数の命令 `opmode` が指定されるので、Sel 入力ポートが自動的に追加されます。



次は、上の図の DSP48 Macro のダイアログ ボックスを示しています。[Instructions] フィールドに 3 つの `opmode` が入力されています。



Sel 入力に 0 が指定されると、最初の命令 `opmode` がインプリメントされます。Zo の値は P 出力に直接読み込まれます。この例では、出力が 5 になります。

Sel 入力に 1 が指定されると、2 つ目の命令 `opmode` ($Xo * Yo$) がインプリメントされます。この例では、出力が 12 になります。

Sel に 2 が指定されると、3 つ目の命令 ($Xo * Yo + Zo$) がインプリメントされ、出力が 17 になります。

デザインがコンパイルされると、ターゲット デバイスが Virtex®-4 の場合は DSP48 スライスが、Virtex-5 の場合は DSP48E スライスが、Spartan®-3A DSP の場合は DSP48A スライスがインプリメンテーションで使用されます。

オペランドの形式

opmode に使用される各オペランド (記号ポート識別子) は、次の規則に従って記述される必要があります。

- 各オペランドは、アルファベット (a ~ z、A ~ Z) で始まり、その後はアルファベット/数字またはアンダースコア (_) を使用する必要があります。
- オペランドには、次の表の予約ポート名は使用できません。
- オペランドは、[Inputs to port A]、[Inputs to port B]、[Inputs to port C] フィールドにそれぞれ 1 度しか入力できません。同じリストにオペランドを複数入れる場合は、スペースかカンマ (,) で区切る必要があります。

上図では、Xo、Yo、Zo 記号ポート識別子です。記号ポート識別子/オペランドは、a1、signal_1、delayed_signal や Cin、_port1、delay\$%、12signal のように記述できます。

予約ポート名

予約ポート名	ポート タイプ	メモリ タイプ
PCIN	入力。DSP48 の PCIN ポートに接続されます。	使用される opmode によって表示されるポートです。この後の 2 つ目の表を参照してください。opmode が 0x10-0xf の場合、PCIN 入力ポートが使用されます。PCIN ポートは、別の DSP48/DSP48 Macro ブロックの PCOUT ポートに接続する必要があります。
BCIN	入力。DSP48 の BCIN ポートに接続されます。	2 つ目の表にリストされる opmode で B が BCIN に置換されると表示されるポートです。別の DSP48/DSP48 Macro ブロックの BCOUT ポートに接続する必要があります。
PCIN>>17	入力。DSP48 の PCIN ポートに接続されます。	この後の 2 つ目の表を参照してください。opmode が 0x50-0x5f の場合、このポートが使用されます。これは、PCIN が 17 ビット右にシフトされたポートで、DSP48 の z マルチプレクサを通った DSP48 加算器への入力になります。

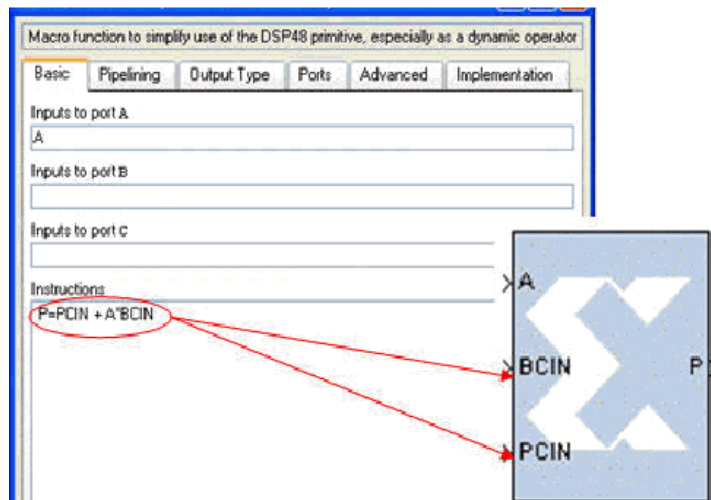
予約ポート名	ポート タイプ	メモリ タイプ
CIN	入力。DSP48 の carry_in ポートに接続されます。	opmode に Cin が含まれると表示されるポートです。この後の 2 つ目の表を参照してください。0x00 以外の opmode すべてでオプションです。
PCOUT	出力。DSP48 の PCOUT ポートに接続されます。	[Ports] タブの [PCOUT] をオンにすると、このポートが表示されます。
ACOUT	出力。DSP48 の ACOUT ポートに接続されます。	[Ports] タブの [ACOUT] をオンにすると、このポートが表示されます。
BCOUT	出力。DSP48 の BCOUT ポートに接続されます。	[Ports] タブの [BCOUT] をオンにすると、このポートが表示されます。
rst_all	入力。DSP48 の rst とすべてのレジスタのリセット ポートに接続されます。	[Ports] タブの [Global Reset] をオンにすると、このポートが表示されます。
ce_all	入力。DSP48 の en とすべてのレジスタのイネーブル ポートに接続されます。	[Ports] タブの [Global Enable] をオンにすると、このポートが表示されます。
Sel	入力	[Instructions] フィールドに命令文が複数あると表示され、[Instructions] フィールドの opmode から 1 つを選択するために使用されます。
P	出力	常にあります。
P>>17	-	この後の 2 つ目の表を参照してください。opmode は 0x60-0x6f です。これは、P が 17 ビット右にシフトされたポートで、DSP48 の z マルチプレクサを通った DSP48 加算器への入力になります。

opmode の選択

前述したように、[Instructions] フィールドに opmode が複数ある場合、opmode を選択するポートをブロックに必要です。この場合は、選択するための Sel ポートが表示されます。Sel ポートはマスクの下のマルチプレクサに接続されます。Sel ポートに接続される信号はすべて適切なデータ型にする必要があります。[Instructions] フィールドにリストされる各 opmode の Sel 信号の値は opmode の位置に対応します。最初の位置が 0 の場合、2 つ目の位置は 1 になり、この後数値が上がっていきます。

予約識別子の使用

予約識別子には、DSP48 Macro ブロックでポートとして明示されるものとそうでないものの 2 種類があります。各予約語の識別子の説明と使用方法は、上の表に記述しています。次の図は、予約語 PCIN と BCIN を使用した例です。[Instructions] フィールドには $P=PCIN + A*BCIN$ と入力されています。



モードの選択

DSP48 Macro は、加算器モードと乗算器モードの 2 つのモードで使用できます。モードは、使用される DSP48 Macro の **opcode** によって選択されます。各モードでサポートされる **opcode** については、2 つ目の表を参照してください。A ポートと B ポートが DSP48 の加算器への入力として接続されると、1 つの符号付き 36 ビット入力に連結されます (DSP48 のセクションを参照してください)。DSP48 の乗算器では、これらのポートが別々の 2 の補数の符号付き 18 ビット入力として解釈されます。

DSP48 の opcode

次の表では、Cin はすべての **opcode** でオプションになります。A:B は、DSP48 Macro ブロックの [Inputs to port A] フィールドのオペランドすべてを表し、DSP48 ブロックの加算器への入力になります。A、B、C はそれぞれ [Inputs to port A]、[Inputs to port B]、[Inputs to port C] フィールドのオペランドを示します。その他すべての記号は予約されています (詳細は、上のオペランドの表を参照してください)。

DSP48 Macro の擬似 opcode	DSP48 Macro モード	DSP48 のサポート	DSP48E のサポート	DSP48A のサポート
$P=Cin$ $P=+Cin$ $P=-Cin$	----	○	○	○
$P=P+Cin$ $P=-P-Cin$	----	○	○	○
$P = A:B + Cin$	加算器	○	○	○
$P = A*B + Cin$ $P = -A*B - Cin$	乗算器	○	○	○

DSP48 Macro の擬似 opcode	DSP48 Macro モード	DSP48 のサ ポート	DSP48E の サポート	DSP48A の サポート
$P=C+Cin$ $P=-C-Cin$	----	○	○	○
$P=+C+P+Cin$ $P=-C-P-Cin$	----	○	○	○
$P=A:B + C + Cin$ $P = -A:B-C-Cin$	加算器	○	○	○
$P = PCIN + Cin$ $P = PCIN -Cin$	----	○	○	○ ($A:B + C + Cin$ のみ)
$P=PCIN+P+Cin$ $P=PCIN-P-Cin$	----	○	○	○
$P=PCIN+A:B+Cin$ $P=PCIN-A:B-Cin$	加算器	○	○	○
$P=PCIN+A*B+Cin$ $P=PCIN-A*B-Cin$	乗算器	○	○	○
$P=PCIN+C +Cin$ $P=PCIN-C -Cin$	----	○	○	○
$P=PCIN+C+P+Cin$ $P=PCIN-P-C-Cin$	----	○	○	○
$P=PCIN+A:B+C+Cin$ $P=PCIN-A:B-C-Cin$	加算器	○	○	○
$P=P-Cin$	----	○	○	○
$P=P+P+Cin$ $P=P-P-Cin$	----	○	○	○
$P=P-A:B-Cin$ $P=P+A:B+Cin$	加算器	○	○	○
$P=P+A*B+Cin$	乗算器	○	○	○
$P=P+C+Cin$ $P=P-C-Cin$	----	○	○	×
$P=P+C+P+Cin$ $P=P-C-P-Cin$	----	○	○	×
$P=P+C+P+Cin$ $P=P-C-P-Cin$	加算器	○	○	×
$P=C-Cin$	----	○	○	○
$P=C-P-Cin$	----	○	○	○

DSP48 Macro の擬似 opmode	DSP48 Macro モード	DSP48 のサポート	DSP48E のサポート	DSP48A のサポート
$P=C-A:B-Cin$	加算器	○	○	○
$P=C-A*B-Cin$	乗算器	○	○	○
$P=C+C+Cin$ $P=C-C-Cin$	----	○	○	×
$P=C+C+P+Cin$ $P=C-C-P-Cin$	----	○	○	×
$P=PCIN>>17+Cin$, $P=PCIN>>17Cin$	----	○	○	×
$P=PCIN>>17+P+Cin$ $P=PCIN>>17-P-Cin$	----	○	○	×
$P=PCIN>>17+A:B+Cin$ $P=PCIN>>17-A:B-Cin$	加算器	○	○	×
$P=PCIN>>17+A*B+Cin$ $P=PCIN>>17-A*B-Cin$	乗算器	○	○	×
$P=PCIN>>17+C+Cin$ $P=PCIN>>17-C-Cin$	----	○	○	×
$P=PCIN>>17+P+C+Cin$ $P=PCIN>>17-P-C-Cin$	----	○	○	×
$P=PCIN>>17+C+A:B+Cin$ $P=PCIN>>17-C-A:B-Cin$	加算器	○	○	×
$P=P>>17+Cin$ $P=P>>17-Cin$	----	○	○	×
$P=P>>17+P+Cin$ $P=P>>17-P-Cin$	----	○	○	×
$P=P>>17+A:B+Cin$ $P=P>>17-A:B-Cin$	加算器	○	○	×
$P=P>>17+A*B+Cin$ $P=P>>17-A*B-Cin$	乗算器	○	○	×
$P=P>>17+C+Cin$ $P=P>>17-C-Cin$	----	○	○	×
$P=P>>17+P+C+Cin$ $P=P>>17-P-C-Cin$	----	○	○	×
$P=P>>17+C+A:B+Cin$ $P=P>>17-C-A:B-Cin$	加算器	○	○	×

パイプライン オプションの入力とカスタム パイプライン オプションの変更

A、B、C ポートのデータパスはそれぞれ異なっており、含まれるレジスタの数も違うことがあります。このため、アライメントの問題が発生します。制御信号でも同じ問題があります。このため、パイプライン モデルが非常に重要になります。DSP48 Macro ブロックには、[External Registers]、[No External Registers]、[Custom] の 3 つのパイプライン オプションが使用できます。

[External Registers]

このオプションを使用すると、DSP48 ブロックの外部にもレジスタが追加され、すべての制御信号とデータ信号がアライメントされます。高速処理のためには、マルチプレクサの出力にこれらの外部レジスタを付ける必要があります。DSP48 Macro の [Instructions] フィールドのすべての opmode に乗算器が必要な場合は、DSP48 Macro のレイテンシは 4 になり、どの命令でも乗算器が必要とされない場合は、レイテンシが 3 になります。

[No External Registers]

DSP48 ブロックの外部にレジスタを使用せず、すべての制御信号とデータ信号がアライメントされます。このモードでは、MREG は選択されません。DSP48 Macro のレイテンシは 2 になります。

[Custom]

このオプションでは、ブロックの各レジスタを制御できます。このオプションを選択すると、[Custom Pipeline Options] フィールドで各レジスタを選択できるようになります。DSP48 で乗算器を使用し、入力の 1 つに A:B を使用した加算器を使用する必要のある命令が DSP48 Macro に含まれる場合は、パイプライン オプションには [Custom] しか使用できません。

DSP48 Macro の制限

DSP48 Macro を使用すると、DSP 48 ブロックを使用する必要はなくなりますが、次のような制限があります。

- DSP48 の丸め機能はサポートされません。
- ファブリックからのキャリーインのみがサポートされます。
- 入力データ型はすべてがサポートされているわけではありません。1 つの DSP48 のデータ型の制限を越える入力データ型は現在のところサポートされていません。たとえば、入力をアライメントした後、DSP48 のポート A の入力が 18 ビットを超えると、エラーになります。

関連項目

DSP48 ブロックの詳細については、次のトピックを参照してください。

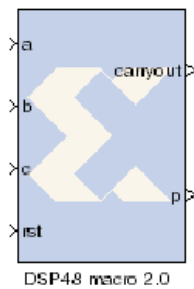
[DSP48 ブロック](#)

[複数クロックのサイクル単位アイランドの生成](#)

[ザイリンクス XtremeDSP](#)

DSP48 Macro 2.0

このブロックは、[Xilinx Blockset] の [Index] および [DSP] ライブラリにリストされています。



DSP48 Macro 2.0 ブロックは、DSP48、DSP48A、および DSP48E ブロックをデバイスに依存せずに抽象化したものです。デバイス別に DSP スライスを使用する代わりに、このブロックを使用すると、ザイリンクス デバイス間でデザインを動かしやすくなります。

DSP48 Macro は、opmode、subtract、alumode、inmode 制御すべてを 1 つの SEL ポートに抽象化し、XtremeDSP スライスに対して単純化されたインターフェイスを提供します。また、すべての CE を 1 つの CE に、RST を SCLR ポートにまとめます。この抽象化により、デバイス ファミリ間で HDL を移行しやすくなります。

命令は 1 ～ 64 まで指定することができ、ターゲット デバイスの XtremeDSP スライスのさまざまな制御信号に変換されます。この命令は ROM に格納され、ROM からは SEL ポートで最適な命令が選択されます。

ブロック パラメータ

[Instructions] タブ

[Instructions] タブは、LogiCORE にインプリメントする動作を定義するために使用します。命令はそれぞれ改行して記述するか、カンマで区切ったリストにし、上から順に列挙します。最大で 64 個まで命令を指定できます。

このタブのパラメータすべての詳細は、LogiCORE IP DSP48 Macro v2.0 の製品仕様の「Instructions Page」(3 ページ目) を参照してください。

[Pipeline Options] タブ

[Pipeline Options] タブは、さまざまな入力パスのパイプラインの深さを定義するために使用します。

[Pipeline Options]

[Automatic]、[By Tier]、[Expert] から使用するパイプライン手法を指定します。

[Custom Pipeline options]

さまざまな入力パスのパイプラインの深さを指定します。

[Tier 1 to 6]

[Pipeline Options] で [By Tier] を選択した場合、指定したパイプライン ステージの入力パスすべてのレジスタをイネーブル/ディスエーブルするために使用されるパラメータです。ただし、次のような制限があります。

- ◆ P が式で指定されると、階層 6 が強制的に使用されます。非同期フィードバックはサポートされません。
- ◆ Spartan-3ADSP/6 の場合、階層 5 と前置加算器を指定すると階層 3 が強制的に使用されます。前置加算器制御信号のレジスタ入力、2 つ目のステージの加算器制御信号と分けることはできません。

[Individual registers]

[Pipeline Options] で [Expert] を選択した場合、このパラメータで各レジスタ ステージをイネーブル/ディスエーブルにできます。ただし、次のような制限があります。

- ◆ P が式で指定されると、P レジスタが強制的に使用されます。非同期フィードバックはサポートされません。
- ◆ Spartan-3ADSP/6 のパイプライン ステージ 5 では、CARRYIN レジスタがステージ 5 の SEL レジスタに接続されます。ステージ 5 の SEL レジスタと前置加算器が指定される場合、ステージ 3 の SEL レジスタが強制的に使用されます。
- ◆ Virtex-4 のパイプライン ステージ 5 では、CARRYIN レジスタがいずれかの乗算器入力で丸め関数を指定すると強制的に使用されます。

このタブのパラメータすべての詳細は、LogiCORE IP DSP48 Macro v2.0 の製品仕様の「Detailed Pipe Implementaton」(9 ページ目) を参照してください。

[Implementation] タブ

[Implementation] タブは、インプリメンテーション オプションを定義するために使用します。

出力ポートのプロパティ

- [Precision] : P 出力ポートの精度を指定します。
 - ◆ [Full] : 出力ポート P のビット幅がフルの XtremeDSP Slide 幅の 48 ビットに設定されます。
 - ◆ [User_Defined] : P の出力幅が 48 ビットまでのどの値にでも設定できます。48 ビット未満に設定すると、出力では切り捨て処理が実行されます (LSB が削除されます)。
- [Width] : P 出力ポートの [User_Defined] 出力幅を指定します。
- [Binary Point] : P 出力ポートの 2 進小数点の位置を指定します。

特別ポート

- [Use ACOUT] : オプションのカスケード A 出力ポートを使用します。
- [Use BCOUT] : オプションのカスケード B 出力ポートを使用します。
- [Use CARRYCASCOUT] : オプションのカスケード キャリアアウト出力ポートを使用します。
- [Use PCOUT] : オプションのカスケード Poutput ポートを使用します。

制御ポート

このタブのパラメータすべての詳細は、LogiCORE IP DSP48 Macro v2.0 の製品仕様の「Implementaton Page」(4 ページ目) を参照してください。

DSP48 Macro ブロック デザインの DSP48 Macro 2.0 へのアップグレード

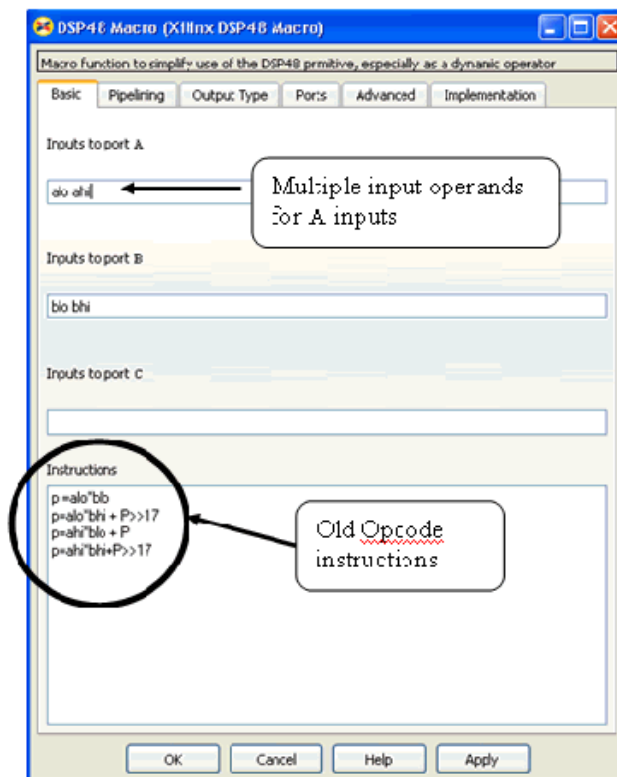
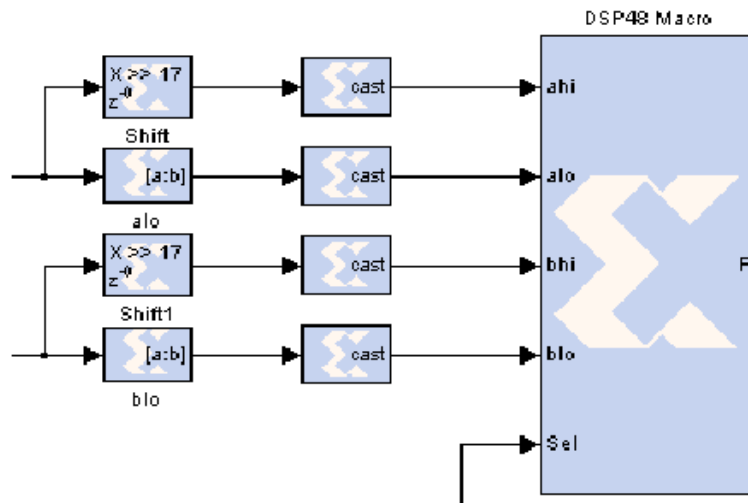
11.4 リリースから、DSP Macro ブロックのバージョン 2.0 がリリースされています。ここでは、既存の DSP Macro ブロック デザインを DSP Macro 2.0 にアップグレードする方法について説明します。

DSP48 Macro と DSP48 Macro 2.0 の主な違いは、この IP のロジック容量を抑えるため、DSP48 Macro 2.0 では内部入力マルチプレクサ回路が削除されていることにあります。これにより、DSP48 Macro を使用した既存デザインで DSP48 Macro 2.0 にアップグレードする場合は、次に注意する必要があります。複数の入力オペランド (A1、A2、B1、B2、etc) を指定する必要はなくなっ

ています。このため、新しい DSP48 Macro 2.0 を使用する場合に次の図に示すような特異な入力オペランドが複数あると、単純な MUX 回路を追加する必要があります。

DSP48 Macro-ベースの符号付き 35x35 乗算器

次の DSP48 Macro には、ポート A への入力に `alo`、`ahi`、ポート B への入力に `blo`、`bhi` といった複数の 18 ビットの入力オペランドが含まれています。この入力オペランドと **Opcode** 命令は次のように指定されます。複数の入力オペランドが DSP48 Macro ブロックにより内部で処理されていることがわかります。



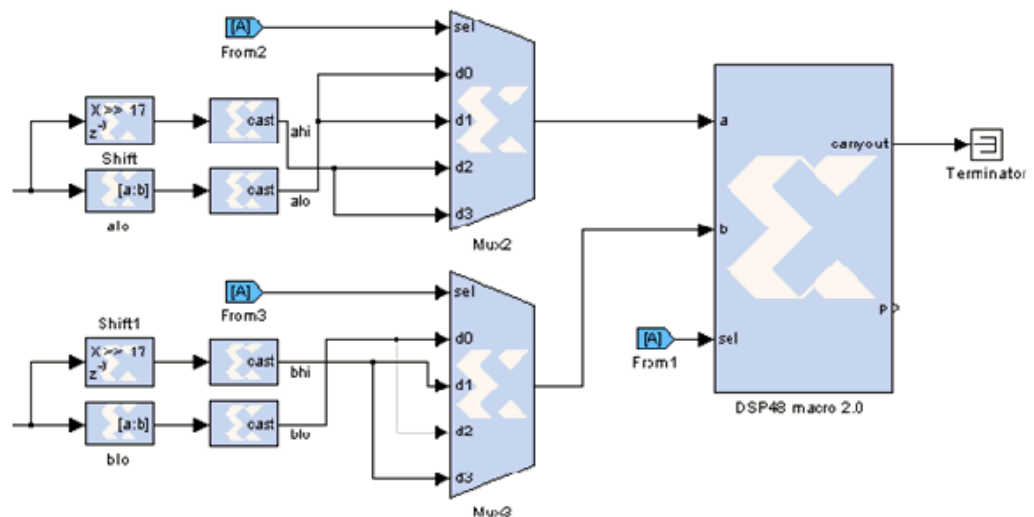
DSP48 Macro-2.0 ベースの符号付き 35x35 乗算器

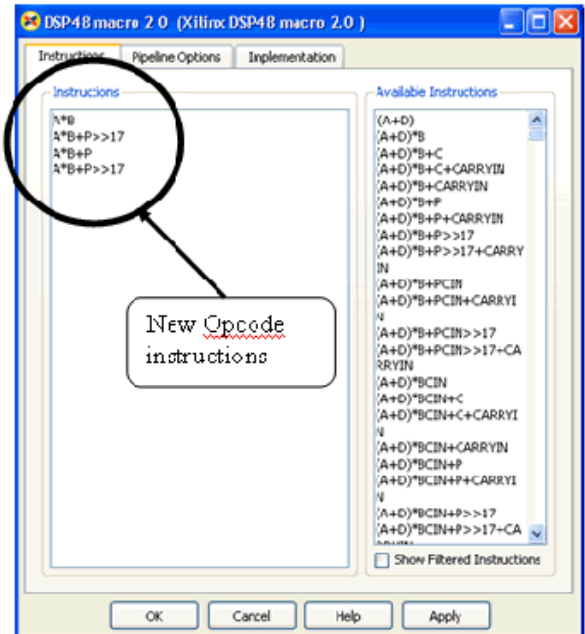
上記と同じモデルは新しい DSP48 Macro 2.0 ブロックにアップグレードできます。デザインをアップデートする場合は、次の簡単な手順とデザイン ガイドラインに従う必要があります。

1. 古いブロックと新しいブロックの入力および出力パイプライン レジスタの選択が同じであることを確認してください。これは [Pipeline Options] の設定から確認できます。
2. 必要とされる特異なオペランドが複数ある場合、次の図のように MUX 回路を使用する必要があります。
3. 新しいデザインに古いバージョンと同じ機能があり、結果が同じになることを確認してください。確認するには、簡単な Simulink シミュレーションを実行して、デザインをインプリメントする必要があります。
4. System Generator の DSP48 Macro 2.0 ブロックを使用して前置加算器をコンフィギュレーションおよび指定する場合、データ幅の入力オペランドのようなデザイン パラメータの一部はデバイスによって異なることに注意してください。この LogicCore IP のパラメータすべての詳細は、LogiCORE IP DSP48 Macro v2.0 の製品仕様を参照してください。

4 入力および 2 出力の MUX 回路は、次のようにデコードできます。

<u>sel</u>	<u>A inputs</u>	<u>B inputs</u>	<u>Opcode</u>
0	<u>alo</u>	<u>blo</u>	$A * B$
1	<u>alo</u>	<u>bhi</u>	$A * B + P \gg 17$
2	<u>ahi</u>	<u>blo</u>	$A * B + P$
3	<u>ahi</u>	<u>bhi</u>	$A * B + P \gg 17$





上記の完全モデルは、次のパスから入手できます。
 <sysgen_path>/examples/dsp48/mult35x35/dsp48macro_mult35x35.mdl

ザイリンクス LogiCORE

このブロックでは、次のザイリンクス LogiCORE コアが使用されます。

System Generator Block	Xilinx LogiCORE™	LogiCORE バージョン / データシート	Spartan® Device					Virtex® Device				
			3, 3E	3A	3A DSP	6	6 -1L	4	5	5Q	6	6 -1L
DSP48 Macro 2.0	DSP48 macro 2.0	V2.0	•	•	•	•	•	•	•	•	•	•

関連項目

DSP48 ブロックの詳細については、次のトピックを参照してください。

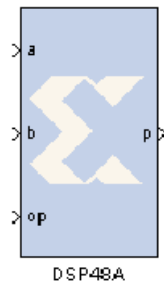
[DSP48 ブロック](#)

[複数クロックのサイクル単位アイランドの生成](#)

[ザイリンクス XtremeDSP™](#)

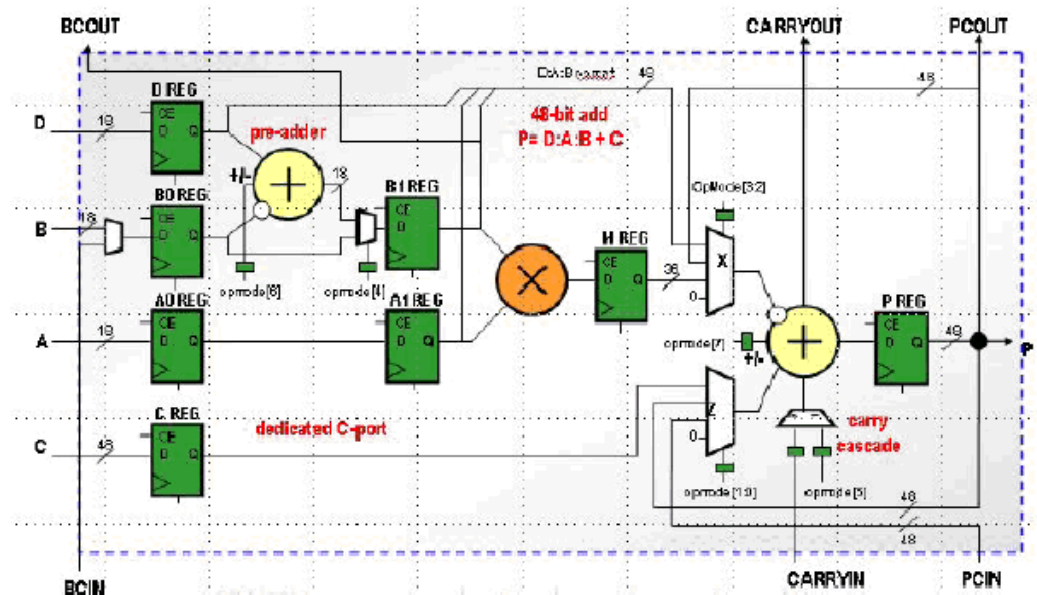
DSP48A

このブロックは、[Xilinx Blockset] の [Index] および [DSP] ライブラリにリストされています。



ザイリンクスの DSP48A ブロックは、ザイリンクスの Spartan-3A DSP デバイスを使用する DSP アプリケーションを効率的に構築するためのブロックです。DSP48A は、DSP48 および DSP48E の軽量版です。

このブロックの特徴は、専用の C ポートと加算器の前にさらに加算器 (図の pre-adder) が追加されているところにあります。DSP48A では、18X18 ビットの符号付き乗算器と 48 ビット加算器とが組み合わされており、加算器の入力はプログラマブル マルチプレクサで選択されます。これらの演算は、ダイナミックに選択できます。オプションの入力と乗算器パイプライン レジスタだけでなく、subtract、carry_in、opcode ポートのレジスタも選択できます。DSP48A ブロックは、[Implementation] タブの [Use synthesizable model] をオンにした場合、DSP48A ハードウェア プリミティブを含まないデバイスをターゲットにすることもできます。



ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Consolidate control port (opcode, carry_in, preadd select, preadd subtract)] : オンにすると、opcode、subtract、preadd select、preadd subtract ポートを 1 つの 8 ビット ポートに組み合わせます。ビット 0 ~ 3 が opcode ポート、ビット 4 が preadd select ポート、ビット 5 が carry_in ポート (carry_in が direct に設定されている場合)、ビット 6 が preadder subtract ポート、ビット 7 が subtract ポートになります。このオプションは、DSP48 命令を生成するのに Opmode ブロックを使用した場合にのみ使用してください。
- [Provide C port] : オンにすると、c ポートが使用可能になります。オフの場合は、c ポートは 0 に接続されます。

- [Provide D port] : オンにすると、d ポートが使用可能になります。オフの場合は、d ポートは 0 に接続されます。
- [Provide PCIN port] : オンにすると、pcin ポートが使用可能になります。pcin ポートは、別の DSP48A ブロックの pcout にのみ接続できます。
- [Provide PCOUT port] : オンにすると、pcout ポートが使用可能になります。pcout ポートは、別の DSP48A ブロックの pcin にのみ接続できます。
- [Provide BCOUT port] : オンにすると、bcout ポートが使用可能になります。bcout ポートは、別の DSP48A ブロックの b ポートにのみ接続できます。
- [Provide CARRYIN port] : オンにすると、carryin ポートが使用可能になります。
- [Provide CARRYOUT port] : オンにすると、carryout ポートが使用可能になります。carryout ポートは、別の DSP48A ブロックの carryin ポートにのみ接続できます。
- [Provide global reset port] : オンにすると、rst ポートが使用可能になります。このポートは、パイプライン選択に基づいて、使用可能なリセット ポートすべてに接続されます。
- [Provide global enable port] : オンにすると、en ポートが使用可能になります。このポートは、パイプライン選択に基づいて、使用可能なイネーブル ポートすべてに接続されます。

[Pipelining] タブ

[Pipelining] タブからは、次のようなパラメータを設定できます。

- [Use A0 reg] : A0 reg を使用するかどうかを指定します。
- [Use A1 reg] : A1 reg を使用するかどうかを指定します。
- [Use B0 reg] : B0 reg を使用するかどうかを指定します。
- [Use B1 reg] : B1 reg を使用するかどうかを指定します。
- [Pipeline C] : c ポートからの入力にレジスタを付けるかどうかを指定します。
- [Pipeline D] : d ポートからの入力にレジスタを付けるかどうかを指定します。
- [Pipeline multiplier] : 内部乗算器の出力にレジスタを付けるかどうかを指定します。
- [Pipeline P] : p 出力と pcout 出力にレジスタを付けるかどうかを指定します。
- [Pipeline opmode] : opmode ポートにレジスタを付けるかどうかを指定します。
- [Pipeline carry in] : carry_in ポートにレジスタを付けるかどうかを指定します。

[Ports] タブ

[Ports] タブからは、次のようなパラメータを設定できます。

- [Reset port for A] : オンにすると、rst_a ポートが使用可能になります。1 に設定されると、ポート a のパイプライン レジスタがリセットされます。
- [Reset port for B] : オンにすると、rst_b ポートが使用可能になります。1 に設定されると、ポート b のパイプライン レジスタがリセットされます。
- [Reset port for D] : オンにすると、rst_d ポートが使用可能になります。1 に設定されると、ポート c のパイプライン レジスタがリセットされます。
- [Reset port for C] : オンにすると、rst_c ポートが使用可能になります。1 に設定されると、ポート c のパイプライン レジスタがリセットされます。
- [Reset port for multiplier] : オンにすると、rst_m ポートが使用可能になります。1 に設定されると、内部乗算器のパイプライン レジスタがリセットされます。

- [Reset port for P] : オンにすると、rst_p ポートが使用可能になります。1 に設定されると、出力レジスタがリセットされます。
- [Reset port for opmode] : オンにすると、rst_opmode ポートが使用可能になります。1 に設定されると、opmode ポートのパイプライン レジスタがリセットされます。
- [Reset port for carry in] : オンにすると、rst_carryin ポートが使用可能になります。1 に設定されると、carry_in のパイプライン レジスタがリセットされます。
- [Enable port for A] : オンにすると、ポート A のパイプライン レジスタのイネーブル ポート ce_a が使用可能になります。
- [Enable port for B] : オンにすると、ポート B のパイプライン レジスタのイネーブル ポート ce_b が使用可能になります。
- [Enable port for C] : オンにすると、ポート C のパイプライン レジスタのイネーブル ポート ce_c が使用可能になります。
- [Enable port for D] : オンにすると、ポート D のパイプライン レジスタのイネーブル ポート ce_d が使用可能になります。
- [Enable port for multiplier] : オンにすると、乗算器レジスタのイネーブル ポート ce_m が使用可能になります。
- [Enable port for P] : オンにすると、ポート P のパイプライン レジスタのイネーブル ポート ce_p が使用可能になります。
- [Enableport for opmode] : オンにすると、ce_opmode ポートが使用可能になります。
- [Enable port for carry in] : オンにすると、carry_in レジスタのイネーブル ポート ce_carry_in が使用可能になります。

[Implementation] タブ

[Implementation] タブからは、次のようなパラメータを設定できます。

- [Use synthesizable model] : オンにすると、DSP48A が RTL 記述からインプリメントされ、DSP48A ハードウェアには直接マップされません。このオプションは、DSP48A ハードウェアプリミティブを含まないデバイス ファミ리를ターゲットにした場合に使用すると便利です。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

関連項目

DSP48 ブロックの詳細については、次のトピックを参照してください。

[DSP48 Macro](#)

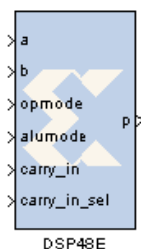
[複数クロックのサイクル単位アイランドの生成](#)

[『Virtex-5 XtremeDSP Design Considerations User Guide』](#)

[ザイリンクス XtremeDSP](#)

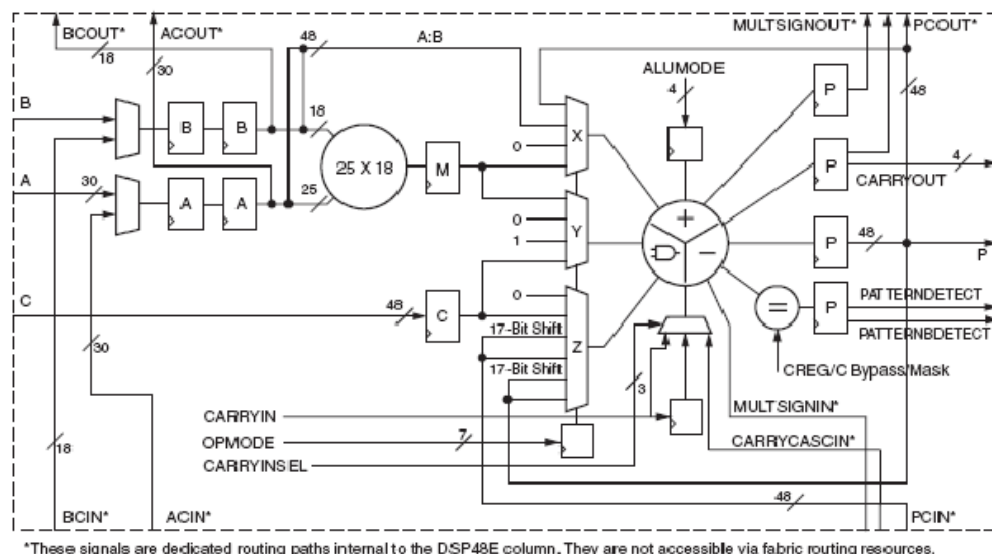
DSP48E

このブロックは、[Xilinx Blockset] の [Index] および [DSP] ライブラリにリストされています。



ザイリンクスの DSP48E ブロックは、ザイリンクスの Virtex-5 デバイスを使用する DSP アプリケーションを効率的に構築するためのブロックです。DSP48E では、18X25 ビットの符号付き乗算器と 48 ビット加算器が組み合わされており、加算器の入力はプログラマブル マルチプレクサで選択されます。

これらの演算は、ダイナミックに選択できます。オプションの入力と乗算器パイプラインレジスタだけでなく、alu mode、carryin、opmode ポートのレジスタも選択できます。DSP48E ブロックは、[Implementation] タブの [Use synthesizable model] をオンにした場合、DSP48E ハードウェア プリミティブを含まないデバイスをターゲットにすることもできます。



ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [A or ACIN input] : A 入力を a ポートから直接取り込むか、カスケードされた acin ポートから取り込むかを指定します。acin ポートは、別の DSP48 ブロックにのみ接続できます。
- [B or BCIN input] : B 入力を b ポートから直接取り込むか、カスケードされた bcin ポートから取り込むかを指定します。bcin ポートは、別の DSP48 ブロックにのみ接続できます。
- [Read dynamic pattern from c register] : オンにすると、パターン検出で使されたパターンが c ポートから読み込まれます。
- [Pattern (48 bit hex value)] : 指定した値がパターン検出ロジックで使されます。パターン検出ロジックでは、加算器、減算器、ロジックユニットの出力の等価チェックが実行されます。
- [Read dynamic mask from c register] : オンにすると、パターン検出で使されたマスクが c ポートから読み込まれます。

- [Pattern mask (48 bit hex value)] : パターン検出中に一部のビットをマスクして除去するために使用する 48 ビットの値です。
- [Reset p register on pattern detection] : オンにしてパターンが検出されると、次のサイクルの p レジスタがリセットされます。

[Optional Ports] タブ

[Optional Ports] タブからは、次のようなパラメータを設定できます。

[Consolidate control port] : オンにすると、opmode、alumode、carry_in、carry_in_sel ポートを 1 つの 15 ビット ポートに組み合わせます。ビット 0 ~ 6 が opmode ポート、ビット 7 ~ 10 が alumode ポート、ビット 11 が carry_in ポート、ビット 12 ~ 14 が carry_in_sel ポートになります。このオプションは、DSP48E 命令を生成するのに Opmode ブロックを使用した場合にのみ使用してください。

[Provide c port] : オンにすると、c ポートが使用可能になります。オフの場合は、c ポートは 0 に接続されます。

[Provide global reset port] : オンにすると、rst ポートが使用可能になります。このポートは、パイプライン選択に基づいて、使用可能なリセット ポートすべてに接続されます。

[Provide global enable port] : オンにすると、オプションの en ポートが使用可能になります。このポートは、パイプライン選択に基づいて、使用可能なイネーブル ポートすべてに接続されます。

[Provide pcin port] : オンにすると、pcin ポートが使用可能になります。pcin ポートは、別の DSP48 ブロックの pcout にのみ接続できます。

[Provide carry cascade in port] : オンにすると、キャリー カスケード入力ポートが使用可能になります。このポートが接続できるのは、別の DSP48E ブロックのキャリー カスケード出力ポートのみです。

[Provide multiplier sign cascade in port] : オンにすると、乗算符号カスケード入力ポート (multsigncascin) が使用可能になります。このポートが接続できるのは、別の DSP48E ブロックの乗算符号カスケード出力ポートのみです。

[Provide carryout port] : オンにすると、carryout 出力ポートが使用可能になります。加算器/減算器の演算モードが 1 つの 48 ビット加算器に設定されると、carryout ポートの幅は 1 ビットになり、2 つの 24 ビット加算器に設定されると、2 ビットになります。MSB は 2 つ目の加算器の carryout に、LSB は最初の加算器の carryout になります。4 つの 12 ビット加算器に設定されると、carryout ポートの幅は 4 ビットになり、加算された 48 ビット入力が 4 つの 12 ビットに分割されます。

[Provide pattern detect port] : オンにすると、パターン検出出力ポートが使用できます。マスクまたは c レジスタのいずれかからのパターンが一致すると、パターン検出ポートは 1 に設定されます。

[Provide pattern bar detect port] : オンにすると、パターン バー検出 (patternbdetect) 出力ポートが使用できます。マスクまたは c レジスタのいずれかからの反転パターンが一致すると、パターン バー検出ポートは 1 に設定されます。

[Provide overflow port] : オンにすると、オーバーフロー出力ポートが使用可能になります。このポートは、DSP48E が P[N] ビットを超えてオーバーフローになることを示します。N は 1 ~ 46 の値になり、GUI のマスク フィールドか c ポート入力で設定されるマスクの 1 の数によって決まります。

[Provide underflow port] : オンにすると、アンダーフロー出力ポートが使用可能になります。DSP48E がアンダーフローになることを示します。アンダーフローは、P[N] ビットを下回ると発生します。N は GUI のマスク フィールドか c ポート入力で設定されるマスクの 1 の数によって決まります。

[Provide ACOUT port] : オンにすると、acout 出力ポートが使用可能になります。acout ポートは、別の DSP48E ブロックの acin にのみ接続できます。

[Provide BCOUT port] : オンにすると、bcout ポートが使用可能になります。bcout ポートは、別の DSP48E ブロックの bcin にのみ接続できます。

[Provide PCOUT port] : オンにすると、pcout ポートが使用可能になります。pcout ポートは、別の DSP48E ブロックの pcin にのみ接続できます。

[Provide multiplier sign cascade out port] : オンにすると、乗算器符号カスケード出力 (multsigncascout) ポートが使用可能になります。このポートは、別の DSP48E ブロックの乗算器符号カスケード入力ポートにのみ接続でき、2 つの DSP48E から構築される 92 ビットのアキュムレータ/加算器および減算器をサポートするために使用されます。

[Provide carry cascade out port] : オンにすると、キャリー カスケード出力ポート (carrycascout) が使用可能になります。このポートが接続できるのは、別の DSP48E ブロックのキャリー カスケード入力ポートのみです。

[Pipelining] タブ

[Pipelining] タブからは、次のようなパラメータを設定できます。

- [Length of a/acin pipeline] : 入力レジスタ A のパイプライン長を指定します。パイプラインの長さを 0 にすると、入力のレジスタが削除されます。
- [Length of b/bcin pipeline] : b 入力のパイプライン長を b から読み込むか、bcin から読み込むかを指定します。
- [Length of acout pipeline] : a/acin 入力ポートと acout 出力ポート間のパイプラインの長さを指定します。0 にすると、acout パイプラインからレジスタが削除されます。値は a/acin パイプラインの長さと同じまたはそれ以下に指定する必要があります。
- [Length of bcout pipeline] : b/bcin 入力ポートと bcout 出力ポート間のパイプラインの長さを指定します。0 にすると、bcout パイプラインからレジスタが削除されます。値は b/bcin パイプラインの長さと同じまたはそれ以下に指定する必要があります。
- [Pipeline c] : c ポートからの入力にレジスタを付けるかどうかを指定します。
- [Pipeline p] : p 出力と pcout 出力にレジスタを付けるかどうかを指定します。
- [Pipeline multiplier] : 内部乗算器の出力にレジスタを付けるかどうかを指定します。
- [Pipeline opmode] : opmode ポートにレジスタを付けるかどうかを指定します。
- [Pipeline alumode] : alumode ポートにレジスタを付けるかどうかを指定します。
- [Pipeline carry in] : carry_in ポートにレジスタを付けるかどうかを指定します。
- [Pipeline carry in select] : carry_in_sel ポートにレジスタを付けるかどうかを指定します。

リセット ポートとイネーブル ポート

[Reset/Enable Ports] タブからは、次のようなパラメータを設定できます。

- [Reset port for a/acin] : オンにすると、rst_a ポートが使用可能になります。1 に設定されると、ポート a のパイプライン レジスタがリセットされます。
- [Reset port for b/bcin] : オンにすると、rst_b ポートが使用可能になります。1 に設定されると、ポート b のパイプライン レジスタがリセットされます。
- [Reset port for c] : オンにすると、rst_c ポートが使用可能になります。1 に設定されると、ポート c のパイプライン レジスタがリセットされます。

- [Reset port for multiplier] : オンにすると、rst_m ポートが使用可能になります。1 に設定されると、内部乗算器のパイプライン レジスタがリセットされます。
- [Reset port for P] : オンにすると、rst_p ポートが使用可能になります。1 に設定されると、出力レジスタがリセットされます。
- [Reset port for carry in] : オンにすると、rst_carryin ポートが使用可能になります。1 に設定されると、carry_in のパイプライン レジスタがリセットされます。
- [Reset port for alumode] : オンにすると、rst_alumode ポートが使用可能になります。1 に設定されると、alumode ポートのパイプライン レジスタがリセットされます。
- [Reset port for controls (opmode and carry_in_sel)] : オンにすると、rst_ctrl ポートが使用可能になります。1 に設定されると、opmode と carry_in_sel レジスタが使用可能な場合、そのパイプライン レジスタがリセットされます。
- [Enable port for first a/acin register] : オンにすると、最初の a パイプライン レジスタのイネーブル ポート ce_a1 が使用可能になります。
- [Enable port for second a/acin register] : オンにすると、2 つ目の a パイプライン レジスタのイネーブル ポート ce_a2 が使用可能になります。
- [Enable port for first b/bcin register] : オンにすると、最初の b パイプライン レジスタのイネーブル ポート ce_b1 が使用可能になります。
- [Enable port for second b/bcin register] : オンにすると、2 つ目の b パイプライン レジスタのイネーブル ポート ce_b2 が使用可能になります。
- [Enable port for c] : オンにすると、ポート C レジスタのイネーブル ポート ce_c が使用可能になります。
- [Enable port for multiplier] : オンにすると、乗算器レジスタのイネーブル ポート ce_m が使用可能になります。
- [Enable port for p] : オンにすると、ポート P 出力レジスタのイネーブル ポート ce_p が使用可能になります。
- [Enable port for carry in] : オンにすると、carry_in レジスタのイネーブル ポート ce_carry_in が使用可能になります。
- [Enable port for alumode] : オンにすると、alumode レジスタのイネーブル ポート ce_alumode が使用可能になります。
- [Enable port for multiplier carry in] : オンにすると、乗算器レジスタのイネーブル ポート mult_carry_in が使用可能になります。
- [Enable port for controls (opmode and carry_in_sel)] : オンにすると、ce_ctrl ポートが使用可能になります。ce_ctrl ポートでは、opmode と carry_in_sel レジスタが制御されます。

[Implementation] タブ

[Implementation] タブからは、次のようなパラメータを設定できます。

- [Use synthesizable model] : オンにすると、DSP48E が RTL 記述からインプリメントされ、DSP48E ハードウェアには直接マップされません。このオプションは、DSP48E ハードウェアプリミティブを含まないデバイス ファミリーをターゲットにした場合に使用すると便利です。
- [Mode of operation for the adder/subtractor] : このモードは、小型の加減算ファンクションを高速で低電力、低ロジック使用率でインプリメントする際に使用します。加算/減算/ロジック ユニットの加算器と減算器は、2 つの 24 ビット フィールドか 4 つの 12 ビット フィールドに分割することもできます。分割する場合は、モードを [Two 24-bit adders] または [Four 12-bit

adders] に設定します。詳細は、『Virtex-5 XtremeDSP Design Considerations User Guide』を参照してください。

- [Use adder only] : オンにすると、ブロックは乗算器を使用せずに、最速のパフォーマンスを達成できるようにハードウェアで最適化されます。乗算器を使用する命令があると、シミュレーションでエラーが発生します。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

関連項目

DSP48 ブロックの詳細については、次のトピックを参照してください。

[DSP48 Macro](#)

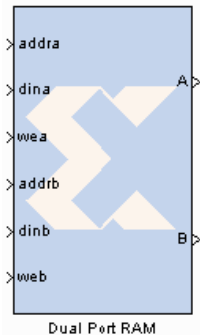
[複数クロックのサイクル単位アイランドの生成](#)

[『Virtex-5 XtremeDSP Design Considerations User Guide』](#)

[ザイリンクス XtremeDSP](#)

Dual Port RAM

このブロックは、[Xilinx Blockset] の [Control Logic]、[Memory]、[Index] ライブラリにリストされています。



ザイリンクスの Dual Port RAM ブロックは、ランダム アクセス メモリ (RAM) をインプリメントします。デュアルポートを使用すると、複数のデータ幅を使用して異なるサンプルレートでメモリ空間に同時にアクセスできます。

ブロック インターフェイス

このブロックには、読み出しと書き込みを同時にするためのポート セットが 2 組含まれます。アドレスポート、データポート、書き込みイネーブルポートを使用することで、単一のメモリ空間へのアクセスが共有できるようになります。デフォルトでは、各ポート セットに出力ポートが 1 つとアドレス用、入力データ用、書き込みイネーブル用に入力ポートが 3 つ含まれます。入力ポート セットには、それぞれポート イネーブルと同期リセット信号をオプションで追加することもできます。

フォーム ファクタ

Dual Port RAM ブロックでは、さまざまなフォーム ファクタ (FF) がサポートされます。フォーム ファクタは、次のように定義されます。

$$FF = W_B / W_A \quad (W_B \text{ はポート B のデータ幅、} W_A \text{ はポート A のデータ幅})$$

ポート B の深さ (D_B) は、次のように指定したフォーム ファクタから推論されます。

$$D_B = D_A / FF.$$

フォーム ファクタ 1 の場合、ポート A とポート B のデータ入力ポートの演算タイプと 2 進小数点はそれぞれ異なります。フォーム ファクタが 1 より大きい場合、ポート A とポート B の演算タイプは符号なしになり、2 進小数点は 0 の位置になります。出力ポート A と B からは、それぞれの入力データポートと同じデータ型が出力されます。

このメモリ ブロックのロケーションは、各アドレスポートに有効なアドレスを指定するとアクセスでき、読み出しまたは書き込みできるようになります。有効なアドレスは、0 ~ $d1$ の符号なしの整数です。 d は、RAM の深さ (RAM のワード数) を表します。メモリの最後よりも先へ読み飛ばそうとすると、シミュレーションでエラーになります。初期の RAM コンテンツは、ブロック パラメータから指定できます。書き込みイネーブルポートは、それぞれブール値にする必要があります。データ入力の値は、WE ポートが 1 の場合にアドレスラインで指定したロケーションに書き込まれます。

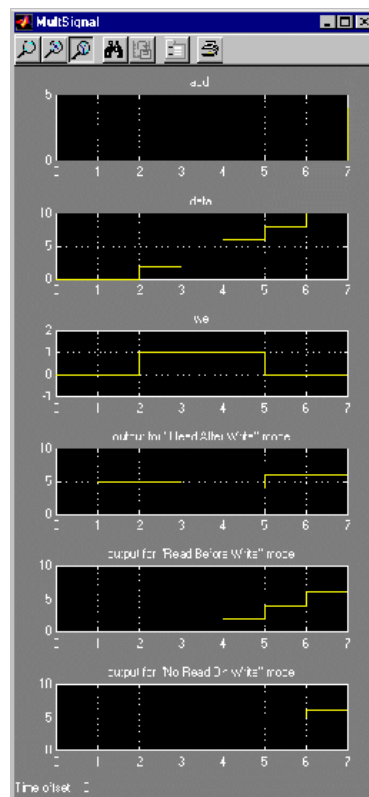
書き込みモード

書き込み中の出力は、書き込みモードによって異なります。WE ポートが 0 の場合、出力ポートはアドレスラインで指定したロケーションの値になります。書き込み中 (WE がアサートされている間)、入力データポートのデータはポートのアドレス入力で選択したメモリ ロケーションに格納され

ます。書き込みサイクル中は、データ出力ポート A と B のビヘイビアは次のいずれかに設定できます。

- Read after write
- Read before write
- No read on write

次の図では、書き込みモードの詳細を示しています。この図では、メモリの初期値は 5 に、アドレスビットは 4 に指定されています。[No read on write] モードを使用した場合、出力はアドレス ラインの影響を受けず、出力は WE が 0 のときの最後の出力と同じになります。ほかの 2 つのモードにすると、出力はアドレス ラインで指定したロケーションから取得されるので、ロケーションの値が書き込まれます。この場合、出力は書き込み前の値 ([Read before write] モード) になります。



コリジョン ビヘイビア

両方のポートに同時にアクセスすると、次のようなビヘイビアになります。

Read-Read コリジョン

両方のポートで同じメモリ セルから同時に読み出しが行われた場合は、問題なく読み出されます。

Write-Write コリジョン

両方のポートが同じメモリ セルに同時に書き込むと、両方の出力に無効であることを示す **nan** というマークが付きます。

Write-Read コリジョン

1 つのポートが書き込みを、もう 1 つが同じメモリ セルから読み出しをすると発生するコリジョンです。メモリ コンテンツは破損しませんが、読み出しポートの出力データは、書き込みポートの書き込みモードによっては破損することがあります。

- 書き込みポートが **[Read before write]** モードの場合、もう 1 つのポートは古いメモリ コンテンツを問題なく読み出すことができます。
- 書き込みポートが **[Read after write]** または **[No read on write]** モードの場合、読み出しポートの出力は無効 (**nan**) になります。

各ポートの書き込みモードは、ブロック パラメータ ダイアログ ボックスの **[Advance]** タブで設定できます。

タイミング パフォーマンスの向上

Virtex-4、Virtex-5、Virtex-6、Spartan-3A DSP デバイスにデュアル ポート RAM ブロックをインプリメントする場合は、次の設定をしておく、最速のタイミング パフォーマンスを達成できます。

- **[Provide synchronous reset port for port A output register]** をオフにします。
- **[Provide synchronous reset port for port B]** をオフにします。
- **[Depth]** には、16,384 未満の値を指定します。
- **[Latency]** は 2 またはそれ以上の値に設定します。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- **[Depth]** : ポート A のメモリのワード数を指定します。値は正の整数にする必要があります。ポート B の深さは、入力データ幅で指定したフォーム ファクタから推論されます。
- **[Initial value vector]** : 初期メモリ コンテンツを指定します。初期値ベクタ エレメントの容量と精度は、ポート A に指定したデータ形式に基づきます。ベクタが RAM よりも長い場合、ベクタの後続エレメントは消去されます。RAM がベクタよりも長い場合、この RAM の後続ワードは 0 に設定されます。初期値ベクタは、RAM のデータ ポート A に指定された精度に従って、サチュレートされるか丸められます。

- [Memory Type] : ブロック RAM か分散 RAM を選択します。分散デュアル ポート RAM では、常にポート A が [Read before write] モードに、ポート B が読み出し専用を設定されます。
- [Write Modes] : メモリを [Read after write]、[Read before write]、または [No read on write] に指定します。これらのモードは、デバイスによって使用できるものとできないものがあります。
- [Initial value for port A output register] : ポート A 出力レジスタの初期値を指定します。初期値は、RAM のデータ ポート A に指定された精度に従って、サチュレート演算されるか丸められます。この初期値を設定するオプションは、Spartan-3、Virtex-4、Virtex-5、Virtex-6、Spartan-3A DSP デバイスでのみ使用できます。
- [Initial value for port B output register] : ポート B 出力レジスタの初期値を指定します。初期値は、RAM のデータ ポート B に指定された精度に従って、サチュレートされるか丸められます。この初期値を設定するオプションは、Spartan-3、Virtex-4、Virtex-5、Virtex-6、Spartan-3A DSP デバイスでのみ使用できます。
- [Provide synchronous reset port for port A output register] : オンにすると、ブロック RAM のポート A 出力レジスタのリセット ポートにアクセスできるようになります。リセット ポートは、ブロック RAM のレイテンシが 1 に設定されている場合にのみ使用できます。
- [Provide synchronous reset port for port B output register] : オンにすると、ブロック RAM のポート B 出力レジスタのリセット ポートにアクセスできるようになります。リセット ポートは、ブロック RAM のレイテンシが 1 に設定されている場合にのみ使用できます。
- [Provide enable port for port A] : オンにすると、ポート A のイネーブル ポートにアクセスできます。イネーブル ポートは、このブロックのレイテンシが 1 以上の場合にのみ使用できます。
- [Provide enable port for port B] : オンにすると、ポート B のイネーブル ポートにアクセスできます。イネーブル ポートは、このブロックのレイテンシが 1 以上の場合にのみ使用できます。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

このブロックでは、常にザイリンクス LogiCORE の Dual Port Block Memory または Distributed Memory が使用されます。デュアルポート ブロック メモリの場合、アドレス幅は $\text{ceil}(\log_2(d))$ と同じにする必要があります。 d は、メモリの深さを示しています。このブロック メモリの最大のデータワード幅は、指定した深さになります。指定できる最大の深さは、ターゲットにしたデバイスファミリーによって異なります。この後の表は、ブロック メモリの深さ別に、最大データワード幅をそれぞれ示しています。

このブロックでは、次のザイリンクス LogiCORE コアが使用されます。

System Generator ブロック	ザイリンクス LogiCORE™	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
Dual Port RAM	Block Memory Generator	V3.3	•	•	•	•	•	•	•	•	•	•
	Distributed Memory Generator	V4.3	•	•	•	•	•	•	•	•	•	•

深さ別の最大データ幅 (Virtex/Virtex-E/Spartan-3)

深さ	幅
2 ~ 2048	256
2049 ~ 4096	192
4097 ~ 8192	96
8193 ~ 16K	48
16K+1 ~ 32K	24
32K+1 ~ 64K	12
64K+1 ~ 128K	6
128K+1 ~ 256K	3

深さ別の最大データ幅 (Virtex-4/Virtex-5/Spartan-3A DSP)

深さ	幅
2 ~ 8192	256
8193 ~ 16K	192
16K+1 ~ 32K	96
32K+1 ~ 64K	48
64K+1 ~ 128K	24
128K+1 ~ 256K	12
256K+1 ~ 512K	6
512K+1 ~ 1024K	3

分散メモリのパラメータを選択した場合は、LogiCORE Distributed Memory が使用されます。深さは、Spartan-3、Virtex-4、Virtex-5、Virtex-6、Spartan-3A DSP デバイスの場合は 16 ~ 65536、その他の FPGA ファミリの場合は 16 ~ 4096 の範囲で指定する必要があります。ワード幅は、1 ~ 1024 の範囲で指定する必要があります。

EDK Processor

このブロックは、[Xilinx Blockset] の [Index] および [Control Logic] ライブラリにリストされています。

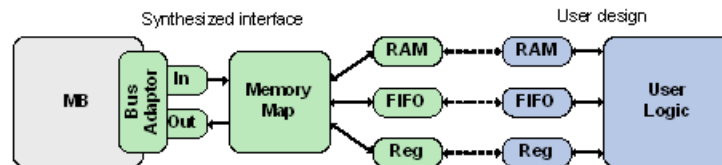


EDK Processor ブロックは、System Generator で開発したユーザー ロジックをザイリンクスの EDK (エンベデッド開発システム) を使用して作成したエンベデッド プロセッサ システムに接続するために使用します。

EDK Processor ブロックでは、EDK pcore 生成 ([EDK pcore generation]) と HDL ネットリスト生成 ([HDL netlisting]) の 2 つのデザイン フローがサポートされています。[HDL netlisting] を使用すると、EDK を使用して作成したエンベデッド プロセッサ システムが System Generator モデルにインポートされ、[EDK pcore generation] を使用すると、System Generator モデルが pcore としてエクスポートされます。この pcore は後で EDK プロジェクトにインポートして、エンベデッド プロセッサに接続できます。

メモリ マップ インターフェイス

EDK Processor ブロックでは、エンベデッド プロセッサと System Generator で開発したユーザー ロジックの通信用に、自動的に Shared Memory ベースのメモリ マップ インターフェイスが生成されます。EDK Processor ブロックでは、エンベデッド プロセッサが接続された共有メモリにアクセスできるように C デバイス ドライバも自動的に生成されます。



上の図は、EDK Processor ブロックで生成されたメモリ マップ インターフェイスを示しています。System Generator で開発されたユーザー ロジックが、共有メモリの設定に接続されています。これらの共有メモリは、後に説明するダイアログ ボックスを使用すると、EDK Processor ブロックに追加できます。EDK Processor ブロックは、自動的に残り半分の共有メモリとメモリ マップ インターフェイスを生成します。メモリ マップ インターフェイスは、ユーザーの選択により、スレーブ PLB v4.6 インターフェイスか FSL (Fast Simplex Link) バスのペアのいずれかを介してこれらの共有メモリと MicroBlaze プロセッサを接続します。デフォルトでは、PLB v4.6 インターフェイスが選択されています。MicroBlaze プロセッサがメモリ マップ インターフェイスにある名前やロケーション情報に基づいて、これらの共有メモリにアクセスできるように、C デバイス ドライバも自動的に生成されます。

メモリ マップ インターフェイスは、[EDK pcore generation] または [HDL netlisting] フローのいずれかで EDK Processor ブロックによって生成されます。[EDK pcore generation] フローの場合、Bus Adaptor の右側のハードウェアのネットリストが pcore としてエクスポートされ、[HDL netlisting] フローの場合、図のすべてのハードウェア (MicroBlaze プロセッサ、メモリ マップ インターフェイス、共有メモリ、ユーザー ロジックなど) がほかの System Generator デザインと同じように、一緒にネットリストにまとめられます。

EDK Processor ブロックのデザインおよびシミュレーション手法については、「ハードウェア/ソフトウェア協調設計」を参照してください。

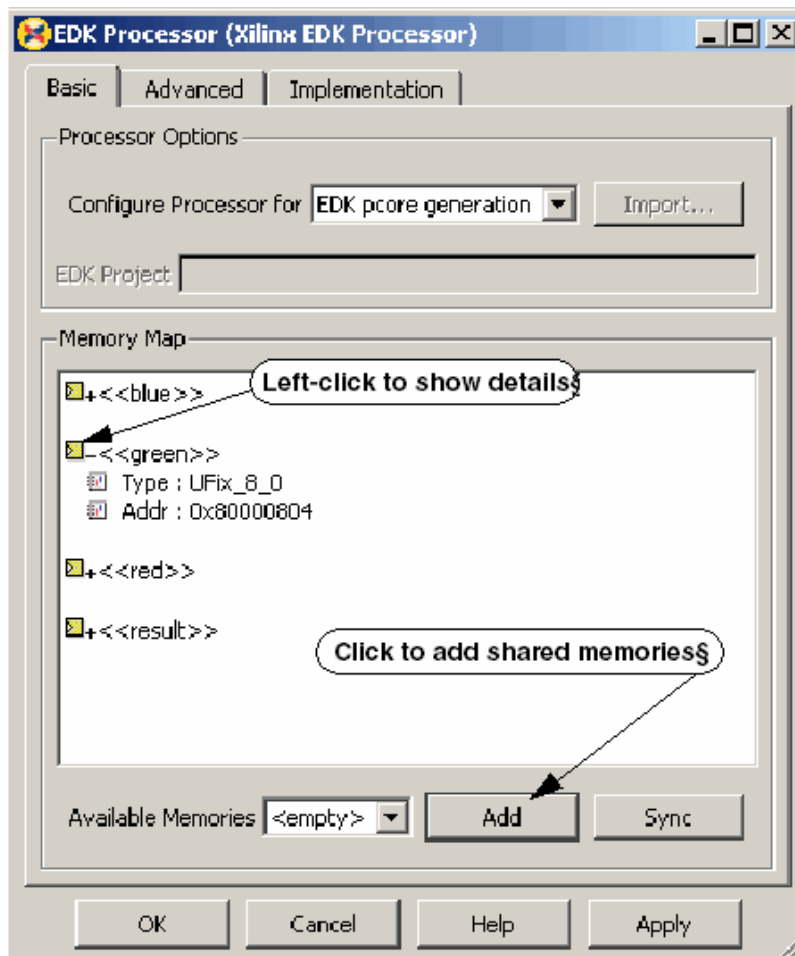
ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Configure processor for] : EDK Processor ブロックは、EDK pcore 生成 ([EDK pcore generation]) または HDL ネットリスト生成 ([HDL netlisting]) のいずれかに設定できます。[HDL netlisting] を選択すると、EDK Import Wizard が自動的に起動されます。
- [Import] : EDK Import Wizard が起動されます。
- [XPS project] : インポートする XMP プロジェクト ファイルの名前を指定します。XMP プロジェクト ファイルを参照するには、[Import] をクリックします。
- [Memory Map] : プロセッサに関連付けられた共有メモリが表示されます。アイテムを右クリックすると、設定、削除、再同期、ツリー ビューの更新など、共有メモリで実行可能な操作メニューが表示されます。共有メモリを再同期すると、ユーザー ロジックで使用された共有メモリと EDK Processor ブロックで自動的に生成された共有メモリに矛盾がないようにできます。



[Advanced] タブ

[Advanced] タブからは、次のようなパラメータを設定できます。

ポート インターフェイス

詳細は、「[System Generator へのプロセッサ ポートの追加](#)」を参照してください。

ソフトウェア

- ◆ [Initial Program] : 初期プログラム (ELF ファイル) を EDK Processor ブロックに設定できます。[Bitstream] または [Hardware Co-simulation] コンパイル ターゲットを使用して EDK Processor ブロックを含むビットストリームを作成する場合、このフィールドで指定された初期プログラム ファイルがビットストリームの作成後にプロセッサのプログラムメモリに読み込まれます。

[Implementation] タブ

メモリ マップ インターフェイス

[Implementation] タブからは、次のようなパラメータを設定できます。

メモ : 11.3 リリースから、System Generator の FSL サポート は開発停止になりました。ISE Design Suite 11 では FSL を続行して使用はできますが、ISE Design Suite 12 からは含まれなくなる予定です。

メモリ マップ インターフェイス

- ◆ [Bus Type] : ペリフェラル バスに PLB (Processor Local Bus) v4.6 または FSL (Fast Simplex Link) を選択します。デフォルトはパフォーマンスの良い PLB v4.6 になっています。
- ◆ pcore バスに PLB が選択されている場合、ターゲットの MicroBlaze プロセッサには DPLB インターフェイスに正しく接続された PLB v4.6 と、システム リセット ピンに接続された proc_sys_reset モジュールが含まれている必要があります。また、pcore の PLB メモリ マップと PLB バスの両方が同じ動作周波数で実行される必要があります。XPS Base System Builder を使用して MicroBlaze プロセッサを構築した場合は、これらの要件が満たされます。
- ◆ [Base Address] : [PLB v4.6 (Processor Local Bus)] を選択すると、自動的にバス アドレス空間が調整され、最小化されます。ベース アドレスがわかっている場合は、ボックスに直接入力し、[lock] をクリックします。それ以外は、自動的に決定されます。[Base Address] オプションは、FSL をバスに選択した場合は使用できません。
- ◆ [Dual Clocks] : PLB v4.6 を選択したときのみ表示されます。EDK インポート フローでは、最上位レベルのネットリストに plb_clk という余分のクロックが表示されます。プロセッサと PLB v4.6 バスのアダプタはこの plb_clk で駆動され、残りの System Generator デザインは sysgen_clk で駆動されます。

ハードウェア協調シミュレーションのネットリストを生成する場合、plb_clk はボードの入力クロックで直接駆動され、sysgen_clk はハードウェア協調シミュレーション モジュールで制御されます。

pcore としてエクスポートする場合、生成される pcore には、System Generator デザインを駆動するため、XPS で別のクロック ポートを接続しておく必要があります。詳細は、「[EDK プロセッサでの非同期のサポート](#)」を参照してください。

- ◆ [Register Read-Back] : メモリ マップのインターフェイスは、通常一方向で、レジスタにはプロセッサからの読み出しまたはプロセッサへの書き込みのいずれかを実行できます。

[Register Read-Back] をイネーブルにすると、書き込みと読み出しを実行できます。このオプションをオンにすると、メモリ マップへの入力が増加し、スピードが低下しエリア使用率が増えます。

制約

- ◆ [Constraint file] : System Generator で自動的に生成される変更済みの UCF ファイルへのパス名。XPS プロジェクトを System Generator に問題なくインポートした後、XPS プロジェクトにユーザー制約ファイル (UCF) が含まれる場合、System Generator でその UCF ファイルが解析され、EDK プロセッサ ブロックの設定に基づいてその UCF ファイルに変更が加えられます。System Generator で加えられた変更を確認するには、[Constraints file] テキスト フィールドの横の [View] ボタンをクリックしてください。変更内容が適切でない場合は、元の UCF ファイルを変更してから、XPS プロジェクトに再インポートします。
- ◆ [Inherit Device Type from System Generator] : EDK プロセッサ ブロックが HDL インポート モードで設定されている場合にのみ使用できます。オンにすると、ネットリスト生成中に、System Generator は System Generator ブロックで選択されたデバイス タイプを XPS に伝え、新規プロセッサ サブシステムを合成します。このオプションをオンにすると、インポートされた XPS システムでボード特有のリソースを使用していたり、特定ボードまたはデバイスにシステムを接続するような制約が含まれていると、ネットリストでエラーが発生することもあります。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

既知の問題

- 各デザインで使用できる EDK Processor ブロックは 1 つだけです。
- 各デザインで使用できる MicroBlaze プロセッサは 1 つだけです。1 つのデザインに複数の MicroBlaze プロセッサおよびエンベデッド PowerPC プロセッサは使用できません。
- Multiple Subsystem Generator ブロックでは、EDK Processor ブロックを含むデザインがサポートされません。

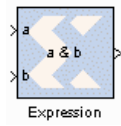
MicroBlaze プロセッサのオンライン マニュアル

MicroBlaze の詳細は、次を参照してください。

http://japan.xilinx.com/products/design_resources/proc_central/microblaze.htm

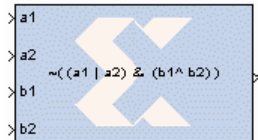
Expression

このブロックは、[Xilinx Blockset] の [Basic Elements]、[Control Logic]、[Math]、[Index] ライブラリにリストされています。



ザイリンクスの Expression ブロックは、ビット単位の論理式を実行します。

論理式は、次の表に記述する演算子で指定します。入力ポートの数は、論理式から推論されます。入力ポートのラベルは論理式から認識され、ブロックにはそのとおりの名前が付きまます。たとえば、 $\sim((a1 \mid a2) \& (b1 \wedge b2))$ では、図のように a1、a2、b1、b2 というラベルの付いた入力ポートが 4 つできます。



論理式が解析されると、それが VHDL (または Verilog) で記述されます。次は、ブロックで使用できる演算子を優先度の高いものから順に示しています。

演算子	シンボル
優先	()
NOT	~
AND	&
OR	
XOR	^

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Expression] : ビット単位の論理式を指定します。
- [Align binary point] : 2 進小数点で自動的に揃えるかどうかを指定します。オフの場合、すべての入力の 2 進小数点の位置が同じになります。

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

Fast Fourier Transform 6.0

メモ：このブロックは、Fast Fourier Transform 7.0 ブロックに置き換わっています。

このブロックは、[Xilinx Blockset] の [DSP] および [Index] ライブラリにリストされています。



Fast Fourier Transform 6.0

ザイリンクス Fast Fourier Transform 6.0 ブロックは、離散フーリエ変換 (DFT) を計算する効率的なアルゴリズムをインプリメントします。

ザイリンクスの Fast Fourier Transform 6.0 ブロックは、Virtex-5、Virtex4、Spartan-3、Spartan-3E、Spartan-3A、Spartan-3A DSP デバイスでのみサポートされます。

N ポイント ($N = 2m, m = 3 - 16$) の順方向または逆方向 DFT (IDFT) は、データ幅 8 ~ 34 を使用して表される N 複素数値のベクタで計算されます。変換の計算には、Cooley-Tukey 型アルゴリズムが使用されます。FFT の一般的な

計算式については、次で説明します。

計算方法

FFT は、正の整数の 2 のべき乗であるサンプル サイズの離散フーリエ変換 (DFT) を計算する効率的なアルゴリズムです。1 シーケンスの DFT は次のように定義されます。

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-jn k 2\pi / N} \quad k = 0, \dots, N-1$$

N は変換を、j は -1 の平方根を表します。逆方向 DFT (IDFT) は、次のように計算されます。

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)e^{jn k 2\pi / N} \quad n = 0, \dots, N-1$$

ブロック インターフェイス

入力信号

- | | |
|-------|--|
| xn_re | <ul style="list-style-type: none"> 入力データ ストリームの実数。xn_re を駆動する信号は、S-1 に 2 進小数点が付いた S 幅の符号付きデータ型にできます (S は 8 ~ 34 の値です)。
例: Fix_8_7、Fix_34_33 |
| xn_im | <ul style="list-style-type: none"> 入力データ ストリームの虚数。xn_im を駆動する信号は、S-1 に 2 進小数点が付いた S 幅の符号付きデータ型にできます (S は 8 ~ 34 の値です)。
例: Fix_8_7、Fix_34_33 |
| start | <ul style="list-style-type: none"> 各ベクタ フレームの開始を示します。start 信号はパルスとしてアサートするか、High に接続して、入力データ フレームの処理を開始することができます。start を駆動する信号は、ブール型にする必要があります。 |

unload	出力を自然な順序で読み出すために使用します。unload ポートは、アーキテクチャに [Radix-4 Burst I/O]、[Radix-2 Burst I/O] または [Radix-2 Lite Burst I/O]、出力順に [Natural Order] を選択してインプリメントした場合にのみ使用できます。unload 信号は、ブロックが入力フレームの処理を終了した後にサンプリングされます。ブロックは、unload 信号が High にアサートされると自然な順序でデータを出力します。start 信号が unload 信号よりも前にアサートされると、ブロックはデータをビット反転順に出力します。unload を駆動する信号は、ブール型にする必要があります。
fwd_inv	逆方向変換の場合は 0、順方向変換の場合は 1 です。fwd を駆動する信号は、ブール型にする必要があります。デフォルトでは FFT は順方向で変換されます。
fwd_inv_we	アサートされると、次の入力データ フレームのために、入力ポート fwd から変換タイプを読み込みます。fwd_inv_we を駆動する信号は、ブール型にする必要があります。
nfft	次の入力データ フレームのポイント サイズを提供します。nfft ポートは、[Enable dynamic transform size] がオンの場合にのみ使用できます。nfft を駆動する信号は、0 に 2 進小数点が付いた幅 5 の符号なし信号、UFix_5_0 にする必要があります。 [Point size of the transform] : NFFT は、変換のサイズまたはそれより小さいポイント サイズのいずれかにできます。たとえば、1024 ポイントの FFT は 1024、512、256... といったポイント サイズを計算できます。NFFT の値は log2 (ポイント サイズ) です。このポートは、[Run time configurable transform length] と一緒にのみ使用されます。
nfft_we	アサートされると、ブロックの現在の操作がリセットされ、次の入力データ フレームのために入力ポート nfft からポイント サイズを読み込みます。nfft_we ポートは、[Enable dynamic transform size] がオンの場合にのみ使用できます。nfft_we を駆動する信号は、ブール型にする必要があります。
cp_len	次の入力データ フレームの CP (Cyclic Prefix) 長サイズを提供します。cp_len ポートは、CP 長の挿入を行うチェック ボックスがオンになっている、出力順が [Natural Order] に設定されている場合にのみ使用できます。cp_len を駆動する信号は、バイナリ ポイントが 0 の N 幅の符号なしの信号にする必要があります、N はサンプル ポイント UFix_N_0 の最大数の log2 です。cp_len は、ポイント サイズより少ない 0 ~ 1 のいずれかの数値になります。
cp_len_we	アサートされると、次の入力データ フレームのために、入力ポート cp_len から CP (Cyclic Prefix) 長を読み込みます。cp_len_we ポートは、CP 長の挿入を行うチェック ボックスがオンになっている、出力順が [Natural Order] に設定されている場合にのみ使用できます。cp_len_we を駆動する信号は、ブール型にする必要があります。
scale_sch	スケーリング スケジュールを入力データ フレームに使用するためのポートです。scale_sch ポートは、固定小数点をスケーリングするモードの場合にのみ使用できます。このポートの詳細は、LogiCORE データシートの 11 ページ目を参照してください。

scale_sch_we	アサートされると、次の入力データ フレームのために、入力ポート scale_sch からスケーリング スケジュールを読み込みます。 scale_sch_we ポートは、固定小数点をスケーリングするモードの場合にのみ使用できます。 scale_sch_we を駆動する信号は、ブール型にする必要があります。
出力信号	
xk_re	出力データ ストリームの実数。[Scaled] および [Block floating point] モードの場合、 xk_re は xn_re 入力と同じになります。 xk_re 信号の幅は、[Unscaled] モードにすると、2 進小数点 xn_re から左に $(1+\log 2N)$ ビット拡張します。 N は最大ポイント サイズです。この信号は dv 信号が High のときのみ有効です。 メモ : xn_re および xn_im 信号は、同じデータ型にする必要があります。
xk_im	出力データ ストリームの虚数。[Scaled] および [Block floating point] モードの場合、 xk_im は xn_im 入力と同じになります。 xk_im 信号の幅は、[Unscaled] モードにすると、2 進小数点 xn_im から左に $(1+\log 2N)$ ビット拡張します。 N は最大ポイント サイズです。この信号は dv 信号が High のときのみ有効です。 メモ : xn_re および xn_im 信号は、同じデータ型にする必要があります。
xn_index	入力データのインデックスを示します。 xn_index 信号は、0 に 2 進小数点が付いた幅 $\log 2N$ の符号なし信号としてマークされます (N は最大ポイント サイズです)。この信号は dv 信号が High のときのみ有効です。
xk_index	出力データのインデックスを示します。 xk_index 信号は、0 に 2 進小数点が付いた幅 $\log 2N$ の符号なし信号としてマークされます (N は最大ポイント サイズです)。
rfd	start 信号がアサートされてから xn_index のカウントが N-1 に到達するまでアクティブ High です (N は最大ポイント サイズです)。 rfd 信号は、ブール信号としてマークされます。
busy	ブロックが現在の入力データ フレームを処理するときにアクティブ High になります。 busy 信号は、ブール信号としてマークされます。
dv	High になると、出力データが有効であることを示します。 dv 信号はブール型です。
edone	ブロックが処理されたデータ フレームを出力する 1 サンプル周期前にアクティブ High になります。 edone は、ブール信号としてマークされます。
done	ブロックが処理されたデータ フレームを出力するときにアクティブ High になります。 done は、ブール信号としてマークされます。
cpv	CP (Cyclic Prefix) データが出力にある場合に、その出力データが有効かどうかを示します。 cpv ポートは、CP 長の挿入を行うチェック ボックスがオンになっていて、出力順が [Natural Order] に設定されている場合にのみ使用できます。 cpv 信号はブール信号としてマークされます。

rfs	ブロックが start 入力を処理してデータを読み込み始める準備ができると、アクティブ High になります。rfs ポートは、CP 長の挿入を行うチェックボックスがオンになっていて、出力順が [Natural Order] に設定されている場合に、[Pipelined Streaming I/O implementation] に対してのみ使用できます。rfs 信号はブール信号としてマークされます。
ovflo	[Scaled] モードで入力データ フレームを処理する間にオーバーフローが検出されると、出力データ フレームがアクティブ High 信号でマークされます。この信号は dv 信号が High のときのみ有効です。ovflo 信号は、ブール信号としてマークされます。
blk_exp	[Block floating point] モードの場合、出力データ フレームの指数を指定します。blk_exp は、2 進小数点が 0 のビット幅 5 の符号なし信号としてマークされます。この信号は dv 信号が High のときのみ有効です。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Transform length] : $N = 2(3..16) = 8 - 65536$ の 1 つ
- [Implementation Options] : [Pipelined]、[Streaming I/O; Radix-4]、[Burst I/O; Radix-2]、[Burst I/O; Radix-2 Lite]、[Burst I/O] のいずれかを選択します。

変換長オプション

- [Run time configurable transform length] : 変換ポイント サイズは、このオプションをオンにすると、nfft ポートから設定できます。有効な設定とそれに対応する変換サイズについては、LogiCORE データシート v.6.0 の「Transform Size」のセクションに記述されています。

[Advanced] タブ

[Advanced] タブからは、次のようなパラメータを設定できます。

精度オプション

- [Phase factor width] : 位相係数のビット幅に 8 ～ 34 のいずれかを選択します。

スケーリング オプション

出力データ型を [Unscaled]、[Scaled]、[Block floating point] のいずれかに選択します。[Block floating point] は、インプリメンテーションに [Pipelined Streaming I/O] を選択した場合にのみ使用できます。

丸めモード

- [Rounding mode] : [Truncation] または [Convergent rounding] を選択します。

出力順序

- [Output ordering] : 出力をビット反転順 (Bit/Digit) か自然な順序 (Natural Order) のどちらかに設定します。

- **[Cyclic prefix insertion]** : 変換出力フレームの CP (Cyclic Prefix) の挿入をダイナミックに指定するオプションの入力ポート **cp_len** および **cp_len_we** を付けることができます。CP の挿入には、FFT の出力のセクションが 1 つ使用され、変換の最初には接頭辞が付きます。CP を含む冗長出力データ (出力データの終わりのコピー) 後に、出力データがすべて通常の順番どおりに続きます。CP の挿入は、出力順が **[Natural Order]** の場合にのみ使用できます。

オプションのピン

- **en** :
- **rst** :
- **ovflo** : **[Scaling]** で **[Scaled]** を選択すると、オプションの出力ポート **ovflo** を付けることができます。

[Implementation] タブ

[Implementation] タブからは、次のようなパラメータを設定できます。

メモリ オプション

- **[Data]** : 分散 RAM かブロック RAM を選択します。このオプションは、サンプル ポイント 16 ~ 1024 の場合にのみ使用できます。このオプションは、インプリメンテーションが **[Pipelined Streaming I/O]** の場合は使用できません。
- **[Phase factors]** : 分散 RAM かブロック RAM を選択します。このオプションは、サンプル ポイント 16 ~ 1024 の場合にのみ使用できます。このオプションは、インプリメンテーションが **[Pipelined Streaming I/O]** の場合は使用できません。
- **[Number of stages using Block RAM]** : ブロック RAM と分散 RAM (一部のみ) にデータおよび位相係数を格納します。このオプションは、インプリメンテーションに **[Pipelined Streaming I/O]** を選択した場合にのみ使用できます。
- **[Reorder buffer]** : 分散 RAM かブロック RAM を選択します。

最適化オプション

- **[Optimize Block RAM count using hybrid memories]** : ハイブリッド メモリを使用してブロック RAM カウントを最適化します。
- **[Optimize for speed using Xtreme DSP slices]** : Virtex-4 および Virtex-5 の場合、Xtreme DSP スライスで複素乗算、バタフライ演算の加算/減算を計算できます。
- **[Complex multiplication]** : 実質乗数を 3 ではなく 4 で構築した複素乗算器をインプリメントします。これにより、複素乗算すべてが Xtreme DSP スライス内で計算されるので、クロック速度が高速になります。このオプションを使用すると、DSP48 の追加使用を最小限に抑えたまま、クロック速度を最大限に上げることができます。このオプションは、Virtex-4 でのみ使用できます。Virtex-5 の場合は、常に選択されます。
- **[Butterfly arithmetic]** : DSP48 を使用して、バタフライ演算の加算と減算をインプリメントします。このオプションは、出力幅が 30 以下の場合に、Virtex-4 でのみ使用できます。Virtex-5 の場合、この機能はすべての出力幅で使用できます。

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ブロックのタイミング

FFT ブロックの制御ビヘイビアやタイミングの詳細は、コアのデータシートを参照してください。

ザイリンクス LogiCORE

このブロックは、次のザイリンクス LogiCORE コアを使用します。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan			Virtex	
			3、3E	3A	3A DSP	4	5
Fast Fourier Transform 6.0	Fast Fourier Transform	V6.0	•	•	•	•	•

Fast Fourier Transform 7.0

このブロックは、[Xilinx Blockset] の [DSP] および [Index] ライブラリにリストされています。



Fast Fourier Transform 7.0

ザイリンクス Fast Fourier Transform 7.0 ブロックは、離散フーリエ変換 (DFT) を計算する効率的なアルゴリズムをインプリメントします。

N ポイント ($N = 2m, m = 3 - 16$) の順方向または逆方向 DFT (IDFT) は、データ幅 8 ~ 34 を使用して表される N 複素数値のベクタで計算されます。変換の計算には、[Burst I/O] アーキテクチャの場合 Cooley-Tukey 型アルゴリズムが、[Pipelined]、[Streaming I/O] アーキテクチャの場合 [Decimation In Frequency] が使用されます。FFT の一般的な計算式については、次で説明します。

計算方法

FFT は、正の整数の 2 のべき乗であるサンプル サイズの離散フーリエ変換 (DFT) を計算する効率的なアルゴリズムです。1 シーケンスの DFT は次のように定義されます。

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-jnk2\pi/N} \quad k = 0, \dots, N-1$$

N は変換長を、j は -1 の平方根を表します。逆方向 DFT (IDFT) は、次のように計算されます。

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)e^{jnk2\pi/N} \quad n = 0, \dots, N-1$$

ブロック インターフェイス

入力信号

xn_re	入力データ ストリームの実数。xn_re を駆動する信号は、S-1 に 2 進小数点が付いた S 幅の符号付きデータ型にできます (S は 8 ~ 34 の値です)。例: Fix_8_7、Fix_34_33
xn_im	入力データ ストリームの虚数。xn_im を駆動する信号は、S-1 に 2 進小数点が付いた S 幅の符号付きデータ型にできます (S は 8 ~ 34 の値です)。例: Fix_8_7、Fix_34_33
start	各ベクタ フレームの開始を示します。start 信号はパルスとしてアサートするか、High に接続して、入力データ フレームの処理を開始することができます。start を駆動する信号は、ブール型にする必要があります。

unload	出力を自然な順序で読み出すために使用します。unload ポートは、アーキテクチャに [Radix-4 Burst I/O]、[Radix-2 Burst I/O] または [Radix-2 Lite Burst I/O]、出力順に [Natural Order] を選択してインプリメントした場合にのみ使用できます。unload 信号は、ブロックが入力フレームの処理を終了した後にサンプリングされます。ブロックは、unload 信号が High にアサートされると自然な順序でデータを出力します。データは、unload がアサートされてから数サイクル後に出力されます (直後ではありません)。出力順が [Natural Order] の場合は、常に unload を使用してデータをアンロードする必要があります。start が unload のアサート前にアサートされると、新しい変換が開始され、最後のフレームが上書きされます。出力がビット反転順の場合、unload ピンはなく、start が前のフレームをアンロードして新しいフレームを同時にロードするためにアサートされる必要があります。
fwd_inv	逆方向変換の場合は 0、順方向変換の場合は 1 です。fwd_inv を駆動する信号は、ブール型にする必要があります。デフォルトでは FFT は順方向で変換されます。
fwd_inv_we	アサートされると、次の入力データ フレームのために、入力ポート fwd から変換タイプを読み込みます。fwd_inv_we を駆動する信号は、ブール型にする必要があります。
nfft	次の入力データ フレームのポイント サイズを提供します。nfft ポートは、[Run time configurable transform length] がオンの場合にのみ使用できます。nfft を駆動する信号は、0 に 2 進小数点が付いた幅 5 の符号なし信号、UFix_5_0 にする必要があります。 [Point size of the transform] : NFFT は、変換のサイズまたはそれより小さいポイント サイズのいずれかにできます。たとえば、1024 ポイントの FFT は 1024、512、256... といったポイント サイズを計算できます。NFFT の値は \log_2 (ポイント サイズ) です。このポートは、[Run time configurable transform length] と一緒にのみ使用されます。
nfft_we	アサートされると、ブロックの現在の操作がリセットされ、次の入力データ フレームのために入力ポート nfft からポイント サイズを読み込みます。nfft_we ポートは、[Run time configurable transform length] がオンの場合にのみ使用できます。nfft_we を駆動する信号は、ブール型にする必要があります。
cp_len	次の入力データ フレームの CP (Cyclic Prefix) 長サイズを提供します。cp_len ポートは、CP 長の挿入を行うチェック ボックスがオンになっていて、出力順が [Natural Order] に設定されている場合にのみ使用できます。cp_len を駆動する信号は、バイナリ ポイントが 0 の N 幅の符号なしの信号にする必要があり、N はサンプル ポイント UFix_N_0 の最大数の \log_2 です。cp_len は、ポイント サイズより少ない 0 ~ 1 のいずれかの数値になります。
cp_len_we	アサートされると、次の入力データ フレームのために、入力ポート cp_len から CP (Cyclic Prefix) 長を読み込みます。cp_len_we ポートは、CP 長の挿入を行うチェック ボックスがオンになっていて、出力順が [Natural Order] に設定されている場合にのみ使用できます。cp_len_we を駆動する信号は、ブール型にする必要があります。

scale_sch	スケーリング スケジュールを入力データ フレームに使用するためのポートです。scale_sch ポートは、固定小数点をスケーリングするモードの場合にのみ使用できます。このポートの詳細は、LogiCORE データシートの 12 ページ目を参照してください。
scale_sch_we	アサートされると、次の入力データ フレームのために、入力ポート scale_sch からスケーリング スケジュールを読み込みます。 scale_sch_we ポートは、固定小数点をスケーリングするモードの場合にのみ使用できます。scale_sch_we を駆動する信号は、ブール型にする必要があります。

出力信号

xk_re	出力データ ストリームの実数。xk_re は xn_re 入力と同じ幅とモードになります。xk_re 信号の幅は、[Unscaled] モードにすると、2 進小数点 xn_re から左に $(1+\log_2 N)$ ビット拡張します。N は最大ポイント サイズです。この信号は dv 信号が High のときのみ有効です。
xk_im	出力データ ストリームの虚数。[Scaled] および [Block floating point] モードの場合、xk_im は xn_im 入力と同じになります。xk_im 信号の幅は、[Unscaled] モードにすると、2 進小数点 xn_im から左に $(1+\log_2 N)$ ビット拡張します。N は最大ポイント サイズです。この信号は dv 信号が High のときのみ有効です。 メモ : xk_re および xk_im 信号は、同じデータ型にする必要があります。
xn_index	入力データのインデックスを示します。xn_index 信号は、0 に 2 進小数点が付いた幅 $\log_2 N$ の符号なし信号としてマークされます (N は最大ポイント サイズです)。
xk_index	出力データのインデックスを示します。xk_index 信号は、0 に 2 進小数点が付いた幅 $\log_2 N$ の符号なし信号としてマークされます (N は最大ポイント サイズです)。
rfd	start 信号がアサートされてから xn_index のカウントが N-1 に到達するまでアクティブ High です (N は最大ポイント サイズです)。rfd 信号は、ブール信号としてマークされます。
busy	ブロックが現在の入力データ フレームを処理するときにアクティブ High になります。busy 信号は、ブール信号としてマークされます。
dv	High になると、出力データが有効であることを示します。dv 信号はブール型です。
edone	ブロックが処理されたデータ フレームを出力する 1 サンプル周期前にアクティブ High になります。edone は、ブール信号としてマークされます。
done	ブロックが処理されたデータ フレームを出力するときにアクティブ High になります。done は、ブール信号としてマークされます。
cpv	CP (Cyclic Prefix) データが出力にある場合に、その出力データが有効かどうかを示します。cpv ポートは、CP 長の挿入を行うチェック ボックスがオンになっていて、出力順が [Natural Order] に設定されている場合にのみ使用できます。cpv 信号はブール信号としてマークされます。

rfs	ブロックが start 入力を処理してデータを読み込み始める準備ができると、アクティブ High になります。rfs ポートは、CP 長の挿入を行うチェックボックスがオンになっていて、出力順が [Natural Order] に設定されている場合に、[Pipelined Streaming I/O implementation] に対してのみ使用できます。rfs 信号はブール信号としてマークされます。
ovflo	[Scaled] モードで入力データ フレームを処理する間にオーバーフローが検出されると、出力データ フレームがアクティブ High 信号でマークされます。この信号は dv 信号が High のときのみ有効です。ovflo 信号は、ブール信号としてマークされます。
blk_exp	[Block floating point] モードの場合、出力データ フレームの指数を指定します。blk_exp は、2 進小数点が 0 のビット幅 5 の符号なし信号としてマークされます。この信号は dv 信号が High のときのみ有効です。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Transform length] : $N = 2(3..16) = 8 - 65536$ の 1 つ
- [Implementation Options] : pipelined_streaming_io、radix_4_burst_io、radix_2_burst_io、radix_2_lite_burst_io のいずれかを選択します。
- [Target clock frequency(MHz)] : ターゲット クロック周波数を入力します。
- [Target data throughput(MSPS)] : ターゲット スループットを入力します。

変換長オプション

- [Run time configurable transform length] : 変換長は、このオプションをオンにすると、nfft ポートから設定できます。有効な設定とそれに対応する変換サイズについては、LogiCORE データシート v.7.0 の「Transform Size」のセクションに記述されています。

[Advanced] タブ

[Advanced] タブからは、次のようなパラメータを設定できます。

精度オプション

- [Phase factor width] : 位相係数のビット幅に 8 ～ 34 のいずれかを選択します。

スケーリング オプション

出力データ型を [Unscaled]、[Scaled]、[Block floating point] のいずれかを選択します。

丸めモード

- [Rounding mode] : [Truncation] または [Convergent rounding] を選択します。

出力順序

- [Output ordering] : 出力をビット反転順 (Bit/Digit) か自然な順序 (Natural Order) のどちらかに設定します。

- **[Cyclic prefix insertion]** : 変換出力フレームの CP (Cyclic Prefix) の挿入をダイナミックに指定するオプションの入力ポート `cp_len` および `cp_len_we` を付けることができます。CP の挿入には、FFT の出力のセクションが 1 つ使用され、変換の最初には接頭辞が付きます。CP を含む冗長出力データ (出力データの終わりのコピー) 後に、出力データがすべて通常の順番どおりに続きます。CP の挿入は、出力順が **[Natural Order]** の場合にのみ使用できます。

オプションのピン

- **en** : クロック イネーブル - ブロックでオプションのイネーブル (**en**) ピンが有効になります。イネーブル信号がアサートされないと、イネーブル信号が再びアサートされるまで (またはリセット信号がアサートされるまで)、ブロックが現在のステートを保持したままになります。リセット信号はイネーブル信号よりも優先されます。イネーブル信号は、ブロックのサンプルレートの倍数のレートで実行する必要があり、イネーブル ポートを駆動する信号は、ブール型にする必要があります。
- **rst** : リセット - ブロックでオプションのリセット (**rst**) ピンが有効になります。このリセット信号がアサートされると、ブロックが最初のステートに戻ります。リセット信号は、ブロックで使用可能なオプションのイネーブル信号よりも優先されます。この信号は、ブロックのサンプルレートの倍数のレートで実行する必要があり、リセット ポートを駆動する信号は、ブール型にする必要があります。
- **ovflo** : **[Scaling]** で **[Scaled]** を選択すると、オプションの出力ポート `ovflo` を付けることができます。

入力データのタイミング

- **[No offset]** : 最初のサンプル ペアが開始パルスを使用して適用され、`xn_index=0` から `xn_index=1` に変わると、読み込まれます。
- **[3 clock cycle offset (pre-v7.0 behavior)]** : 最初のサンプル ペアが `xn_index=3` から `xn_index=4` (`start` が適用されてから 3 サイクル後) に変わると読み込まれます。

[Implementation] タブ

[Implementation] タブからは、次のようなパラメータを設定できます。

メモリ オプション

- **[Data]** : 分散 RAM かブロック RAM を選択します。このオプションは、サンプル ポイント 8 ~ 1024 の場合にのみ使用できます。このオプションは、インプリメンテーションが **[Pipelined Streaming I/O]** の場合は使用できません。
- **[Phase factors]** : 分散 RAM かブロック RAM を選択します。このオプションは、サンプル ポイント 8 ~ 1024 の場合にのみ使用できます。このオプションは、インプリメンテーションが **[Pipelined Streaming I/O]** の場合は使用できません。
- **[Number of stages using Block RAM]** : ブロック RAM と分散 RAM (一部のみ) にデータおよび位相係数を格納します。このオプションは、インプリメンテーションに **[Pipelined Streaming I/O]** を選択した場合にのみ使用できます。
- **[Reorder buffer]** : 最大 1024 ポイントの変換サイズまでの分散 RAM かブロック RAM を選択します。
- **[Hybrid Memories]** : ハイブリッド メモリを使用してブロック RAM カウントを最適化する場合はオンにします。

最適化オプション

- **[Complex Multipliers]** : 次のいずれかを選択します。

- ◆ Use CLB logic
- ◆ Use 3-multiplier structure (resource optimization)
- ◆ Use 4-multiplier structure (performance optimization)
- [Butterfly arithmetic] : 次のいずれかを選択します。
 - ◆ Use CLB logic
 - ◆ Use XTremeDSP slices

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ブロックのタイミング

FFT ブロックの制御ビヘイビアやタイミングの詳細は、コアのデータシートを参照してください。

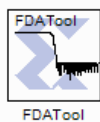
ザイリンクス LogiCORE

このブロックは、次のザイリンクス LogiCORE コアを使用します。

System Generator ブロック	ザイリンクス LogiCORE™	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
Fast Fourier Transform 7.0	Fast Fourier Transform	V7.0	•	•	•	•	•	•	•	•	•	•

FDATool

このブロックは、[Xilinx Blockset] の [DSP]、[Tools] および [Index] ライブラリにリストされています。



ザイリンクスの FDATool ブロックは、MATLAB の Signal Processing Toolbox の一部である FDATool ソフトウェアへのインターフェイスになります。

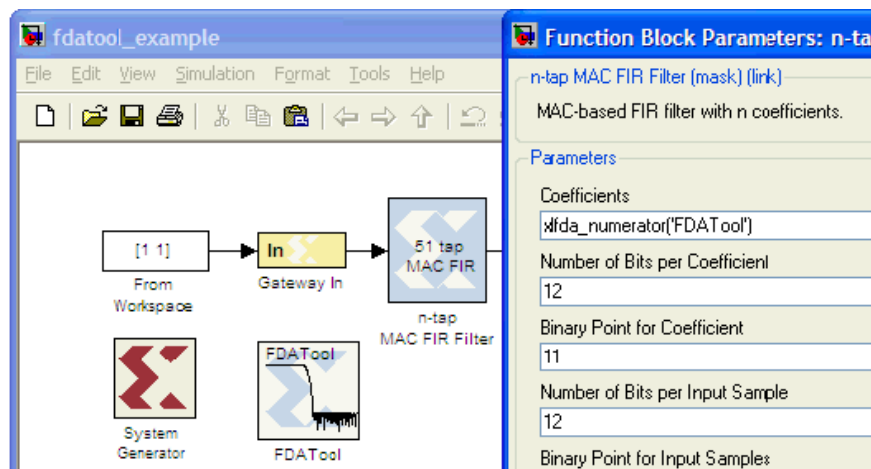
Signal Processing Toolbox がインストールされていない場合、このブロックは正しく機能しないので、使用しないでください。このブロックを使用すると、FDATool オブジェクトを定義し、System Generator モデルの一部として格納することができます。

FDATool では、デジタル フィルタをグラフィック ユーザー インターフェイスを使用して定義できます。

使用例

フィルタを定義するサブシステムに FDATool ブロックをコピーします。ブロックのアイコンをダブルクリックすると、FDATool セッションが開始され、グラフィック ユーザー インターフェイスが表示されます。フィルタは、インターフェイスブロックの内部のデータ構造に格納されます。フィルタの係数は、System Generator の一部として提供される MATLAB のヘルパー関数を使用して抽出できます。xlfda_numerator('fdablk') 関数を呼び出すと、fdablk という名前の FDATool ブロックの伝達関数の分子 (FIR フィルタのインパルス応答) が返されます。同様に、ヘルパー関数 xlfda_denominator('fdablk') を実行すると、非 FIR フィルタの分母が取り出されます。

FDATool ブロックは、通常係数が xlfda_numerator('fdablk') に設定された FIR フィルタブロックと共に使用されます。次は、その関数を指定した FIR フィルタブロックのダイアログボックスを示しています。



xlfda_numerator() は、メモリ ブロックを初期化、または FIR フィルタを含むサブシステムの coefficient 変数を初期化することができます。

このブロックでは、ハードウェア リソースは使用されません。

FDATool のインターフェイス

Simulink モデルでブロックのアイコンをダブルクリックすると、FDATool セッションが開始され、グラフィック ユーザー インターフェイスが表示されます。セッションを閉じると、FDATool オブ

ジェクトが FDATool ブロックの UserData パラメータに格納されます。必要であれば、ヘルパー関数 `xlfa_numerator()` および `get_param()` を使用し、オブジェクトから情報を抽出してください。

FIFO

このブロックは、[Xilinx Blockset] の [Control Logic]、[Memory]、[Index] ライブラリにリストされています。



ザイリンクスの FIFO ブロックは、FIFO メモリ キューをインプリメントします。書き込みイネーブル入力 (we) が 1 になると、このモジュールのデータ入力ポート (din) の値が次の使用可能な空のメモリ ロケーションに書き込まれます。読み出しイネーブル入力 (re) をアサートすると、データが書き込まれる順番にデータ出力ポート (dout) を通って FIFO から読み出されます。FIFO はブロックまたは分散 RAM を使用してインプリメントできます。

未使用のロケーションがモジュールの内部メモリに残っていない場合、出力ポート full が 1 にアサートされます。出力ポート percent_full は、フルの FIFO のパーセントがユーザー指定の精度で表示されます。empty 出力ポートは、FIFO が空になるとアサートされます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Depth] : 格納できるワード数。
- [Bits of precision to use for %full signal] : %full ポートのビット幅を指定します。この符号なし出力の 2 進小数点は、常にワードの最上部にあります。このため、たとえば精度が 1 に設定されると、出力は 0.0 と 0.5 の 2 つの値を取り込みます。この場合、0.5 は FIFO が少なくとも 50% フルであることを示しています。

[Implementation] タブ

- [Memory Type] : FIFO の FPGA へのインプリメンテーション方法を指定します。ブロック RAM か分散 RAM のいずれかを選択できます。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

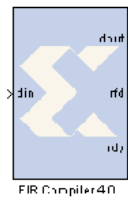
このブロックでは、ザイリンクス LogiCORE FIFO Generator コアを使用してインプリメントされます。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/データシート	Spartan					Virtex				
			3、3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
FIFO	FIFO Generator	V5.3	•	•	•	•	•	•	•	•	•	•

FIR Compiler 4.0

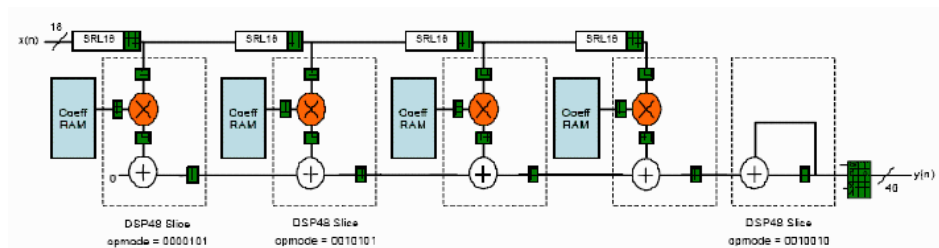
メモ：このブロックは、FIR Compiler 5.0 ブロックに置き換わっています。

このブロックは、[Xilinx Blockset] の [DSP] および [Index] ライブラリにリストされています。



ザイリンクス FIR Compiler 4.0 ブロックは、高速 MAC ベースの FIR フィルタをインプリメントします。このブロックは、入力データのストリームを受信し、フィルタのコンフィギュレーションに応じてフィルタ処理した結果を、固定の遅延で出力します。

このフィルタは、カスケード接続された DSP48/DSP48E/DSP48A スライスを次の図のように使用してインプリメントされます。



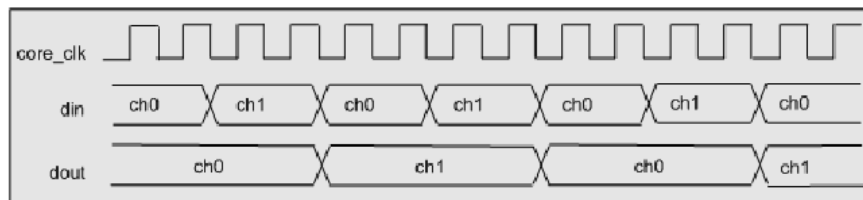
メモ：この後の説明では、DSP/DSP48A/DSP48E をまとめて DSP48 と記述します。

ブロック インターフェイス

FIR Compiler v3_1 ブロックには、すべてのフィルタ コンフィギュレーションで表示される **din** および **dout** に加え、さまざまなオプションのポートを含めることができます。

入力ポート

- **din** : FIR Compiler のデータ入力ポート。このポートを介して、すべてのチャンネルのデータがマルチプレクサと同じように、1 度に FIR Compiler に送信されます。



- **rst** : 同期リセット ポート
- **en** : 同期イネーブル ポート
- **nd** : この信号がアサートされると、**din** ポートのデータ サンプルがフィルタ コアに読み込まれます。**nd** は **rfd** が Low の間はアサートされるべきではありません。これは、**rfd** が Low のときのサンプルがコアですべて無視されるからです。
- **filt_sel** : 入力信号の F ビット幅のフィルタ選択信号で、 $F = \text{ceil}(\log_2(\text{filter sets}))$ です。複数のフィルタ セットを使用したときだけ、使用できます。
- **coef_ld** : 係数読み込みポート。**coef_ld** を駆動する信号は、ブール型にする必要があります。**coef_ld** ポートの詳細は、この後の説明を参照してください。

- **coef_we** : 係数書き込みイネーブルポートです。**coef_we** を駆動する信号は、ブール型にする必要があります。**coef_we** ポートの詳細は、この後の説明を参照してください。
- **coef_din** : 係数データ入力ポート。**coef_din** を駆動する信号は、このブロックの係数に指定した信号と同じタイプにする必要があります。**coef_din** ポートの詳細は、この後の説明を参照してください。
- **coef_filt_sel** : 係数を読み込むための入力信号の F ビット幅のフィルタ選択信号で、 $F = \text{ceil}(\log_2(\text{filter sets}))$ です。複数のフィルタ セットとリローダブル係数を使用したときだけ、使用できます。

出力ポート

- **dout** : FIR Compiler のデータ出力ポート。このポートを介して、フィルタ処理されたすべてのチャンネルの出力データがマルチプレクサのように 1 度に出力されます。
- **rdy** : 新しいフィルタ出力サンプルが **dout** ポートに出力されたことを示します。
- **rfd** : コアが新しいデータを受信できる状態かどうかを示します。
- **chan_in** : どのチャンネルの入力データ サンプルを次の **din** ポートに駆動するかを示します。**din** の入力サンプルがコアに送信されると、即座にこの出力ポートの値が変更され、データ入力の必要な次のチャンネルを示します。

メモ : FIR Compiler の **din** ポートが [Simulink System Period] と異なるレートでサンプリングされると、入力ポートがレジスタに取り込まれ、正確なビットおよびサイクルで System Generator シミュレーションが実行されるようになります。この結果、**chan_in** 出力が **din** ポートのチャンネル データよりも 1 サイクル遅くなります。たとえば、**chan_in** 出力の値が 1 の場合、**din** でサンプリングされたデータはチャンネル 2 に対応します。このビヘイビアは、[Detailed Implementation] タブの [Chan In] オプション エリアで、[Generate chan_in value in advance] をオンにし、[Number of samples] を 1 に設定すると修正できます。

- **chan_out** : **dout** ポートの出力データがあるチャンネルがどれかを示します。
- **dout_i** : ヒルベルト変換を使用した場合の同相部 (I) データの出力を示します。
- **dout_q** : ヒルベルト変換を使用した場合の直交部 (Q) データの出力を示します。
- **data_valid** : 出力データが有効かどうかを示す信号で、**rdy** と共に、または **rdy** よりも優先して使用できます。新しいフィルタ出力サンプルが **dout** ポート (完了したデータ ベクタから生成) に出力されたことを示します。MAC Based FIR ブロックのインプリメンテーションに使用できます。

詳細は、[FIR Compiler のデータシート](#)を参照してください。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Filter Specification] タブ

[Filter Specification] タブからは、次のようなパラメータを設定できます。

フィルタ係数

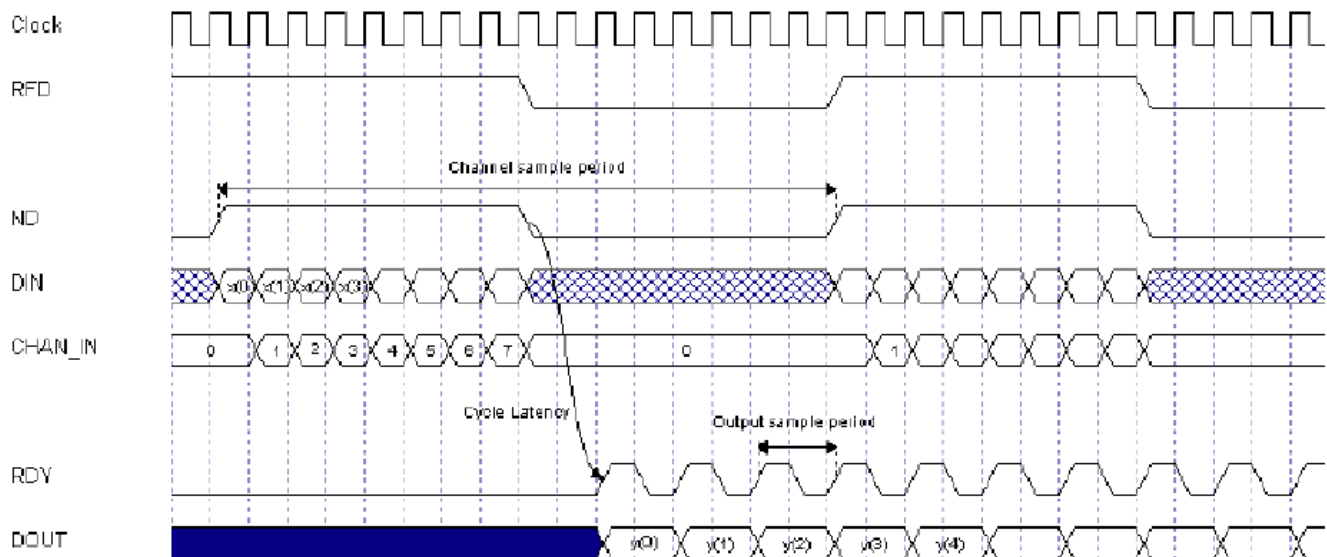
- **[Coefficients]** : 指定した係数ベクタが MATLAB 行のベクタになります。タップ数は、MATLAB 行のベクタ長から推論されます。これらの係数は、[FDATool](#) ブロックを使用しても入力できます。

- [Number of coefficients sets] : 使用される係数セットの数を指定します。1 より大きい場合、複数の係数セット モードの FIR コンパイラが使用されることを示します。

フィルタ仕様

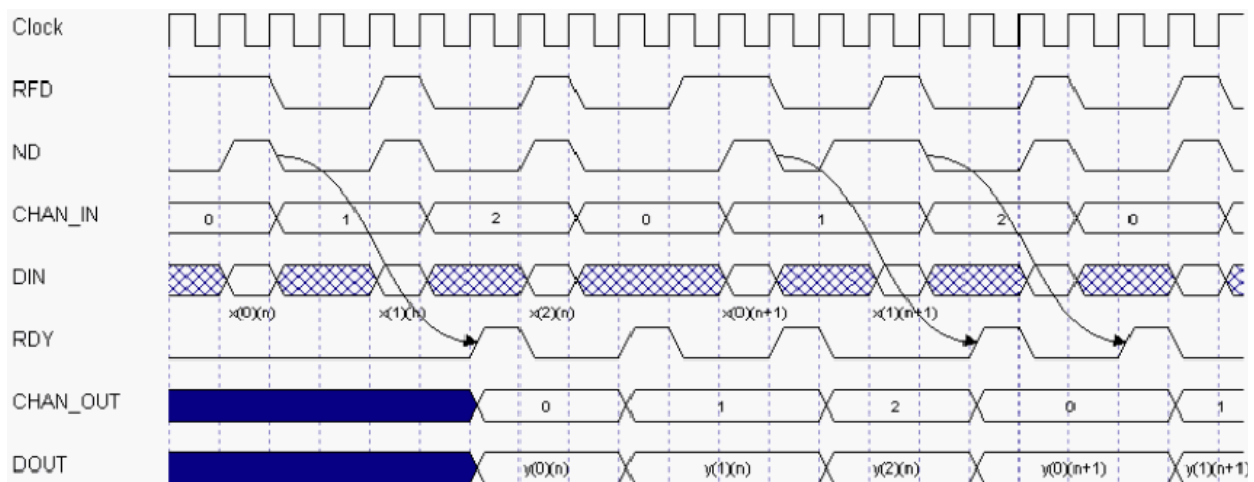
- [Filter type] : フィルタを次のいずれかに設定することができます。
 - ◆ [SingleRate] : 入力と出力のデータ レートが同じになります。
 - ◆ [Decimation] : 出力のデータ レートが入力よりも [Sample rate change] で指定した係数分だけ遅くなります。
 - ◆ [Interpolation] : 出力のデータ レートが入力よりも [Sample rate change] で指定した係数分だけ速くなります。
 - ◆ [Interpolated] : この FIR フィルタのアーキテクチャは、これまでの FIR フィルタと類似していますが、k-1 ユニットの遅延に置き換えられたユニット遅延演算子が付いているところが違います。k はゼロ パック ファクタを示します。このタイプのフィルタは上記の [Interpolation] と異なり、狭帯域フィルタを効率的に実現し、少し変更するだけで広帯域も [Polyphase_Filter_Bank_Transmitter] : ザイリンクスの FFT コアから出力されるデータを効率的に読み込んでフィルタをかける多相フィルタ バンクがインプリメントされます。
 - ◆ [Polyphase_Filter_Bank_Receiver] : ザイリンクスの FFT コアに入力されるデータを効率的に読み込んでフィルタをかける多相フィルタ バンクがインプリメントされます。
- [Rate change type] :
 - ◆ [Integer] : レート変更の係数を整数で指定します。
 - ◆ [Fixed_Fractional] : レート変更の係数を分数で指定します。
- [Interpolation rate value] : 出力のデータ レートが入力よりも [Sample rate change] で指定した係数分だけ速くなります。
- [Decimation rate value] : 出力のデータ レートが入力よりも [Sample rate change] で指定した係数分だけ遅くなります。
- [Zero packing factor] : 係数ベクタで指定した係数の間にここで指定した数の 0 を挿入できます。ゼロ パック ファクタの k は、提供された係数値の間に k-1 個の 0 を挿入します。このパラメータは、[Filter type] が [Interpolation] に設定されると使用できます。
- [Number of channels] : FIR Compiler ブロックで処理されるデータ チャネルの数を指定できます。複数のチャネル データがマルチプレクサと同じように 1 度にコアに渡されます。サポートされるチャネル数は、最大で 64 です。
- [Effective input sample period] : nd が選択されているか、フィルタ タイプに [Polyphase_Filter_Bank_Transmitter] が選択されている場合に、効率的な入力サンプル周期を指定するために使用します。その他すべてのフィルタ タイプおよび制御オプションでは、サンプル周期はそのフィルタの System Generator ブロックの入力ポート プロパティから読み込まれます。効率的な入力サンプル周期は、チャネル サンプル周期を処理されるチャネル数で割ると 算出されます。CORE Generator では、 $\text{Clock_Frequency} / (\text{Sample_Frequency} * \text{Number_Channels})$ と同じになります。このパラメータによって、Systolic_Multiple_Accumulate FIR フィルタのフォールディング ファクタが決まります。

次の図は、Polyphase_Filter_Bank_Transmitter のタイミング図とその効率的な入力サンプル周波数の計算例を示しています。



この図では、8 チャンネルが含まれ、チャンネル サンプル周期は 16 です。これにより、効率的なサンプル周期は 2 になります。効率的な入力サンプル周期と出力サンプル周期は、Polyphase Filter Bank Transmitter フィルタ タイプでは同じ値になります。

次の図は、ND (New Data) ポートが選択された状態の 3-Channel フィルタの入力タイミングを示しています。この例では、チャンネル サンプル周期は 9 で、効率的なサンプル周期は 3 になっています。データ サンプルの処理に使用可能なクロック サイクルの数は、入力サンプル周期、システム周期、インターポーレーション (補間) レート、デシメーション (間引き) レートによって決まり、コアのインプリメンテーションの並列処理レベルに直接影響します。



[Implementation] タブ

[Implementation] タブからは、次のようなパラメータを設定できます。

- [Filter architecture] には、次のいずれかを選択します。
 - ◆ [Systolic_Multiply_Accumulate] : これは 乗算器/DSP48 をカスケード接続した MAC ベースのアーキテクチャで、System Generator ブロックの FIR Compiler v3.0/3.1/3.2 でサポートされるアーキテクチャと類似しています。
 - ◆ [Transpose_Multiply_Accumulate] : Transpose Multiply-Accumulate アーキテクチャでは、Transposed Direct-Form フィルタがインプリメントされます。
 - ◆ [Distributed_Arithmetic] : Distributed Arithmetic FIR がインプリメントされます。

係数オプション

- [Use reloadable coefficients] : ブロックに係数をリロードするポートを追加するかどうかを指定します。
- [Coefficients Structure] : 係数のストラクチャを指定します。係数のストラクチャによって、コアが最適化され、特定のハードウェア コンフィギュレーションをインプリメントするのに必要なハードウェア容量が削減されます。ストラクチャには、次のいずれかを選択できます。
 - ◆ Inferred
 - ◆ Non-Symmetric
 - ◆ Symmetric
 - ◆ Negative_Symmetric
 - ◆ Half_Band
 - ◆ Hilbert

係数のベクタは、ここで指定したストラクチャと一致している必要があります。ただし、[Inferred from Coefficients] を選択した場合、ストラクチャはこれらの係数から自動的に決定されるので、その必要はありません。

- [Coefficient type] : 符号付きか符号なしかを指定します。
- [Quantization] : 係数の量子化方法を次のいずれかに指定します。
 - ◆ Integer_Coefficients
 - ◆ Quantize_Only
 - ◆ Maximize_Dynamic_Range
- [Coefficient width] : 係数を表すために使用するビット数を指定します。
- [Coefficient fractional bits] : 係数のデータパス オプションのバイナリの小数点位置を指定します。
- [Number of paths] : フィルタが処理するパラレル データ パスの数を指定します。
- [Output rounding mode] : 次のいずれかを選択します。
 - ◆ Full_Precision
 - ◆ Truncate_LSBs
 - ◆ Non_Symmetric_Rounding_Down
 - ◆ Non_Symmetric_Rounding_Up
 - ◆ Symmetric_Rounding_to_Zero
 - ◆ Symmetric_Rounding_to_Infinity

- ◆ Convergent_Rounding_to_Even
- ◆ Convergent_Rounding_to_Odd
- [Output width] : 出力幅を指定します。[Rounding Mode] で [Full_Precision] 以外を設定している場合にのみ有効になります。
- [Allow Rounding Approximation] : オンにすると、対称丸め (Symmetric Rounding) を使用している場合に、近似を有効にしてリソースを節約します。

[Detailed Implementation] タブ

[Detailed Implementation] タブからは、次のようなパラメータを設定できます。

- [Optimization Goal] : コアが一番速い速度 ([Speed] オプション)、一番小さいエリア ([Area] オプション) で動作する必要があるかどうかを指定します。デフォルトでは [Area] オプション (推奨) が使用され、通常はそのデザインに最適な速度とエリアを達成されますが、コンフィギュレーションによっては、全体のリソース使用率を改善するために [Speed] 設定が必要なこともあります。[Speed] 設定では、通常クリティカルパスにパイプライン レジスタが追加されます。
 - ◆ Area
 - ◆ Speed

制御オプション

- [rst] : ブロックで rst ポートが使用されます。このコアでは、rst が使用されると、常に sclr_deterministic オプションが使用されます。sclr_deterministic オプションの詳細は、LogiCORE データシートを参照してください。
- [Use deterministic rst behavior] : オンにすると、シミュレーション速度を高速にできます。このフローでは、リセット後、dout はすべてのデータ バッファがポスト リセット データで一杯になるまで、0 に維持されます。この機能を使用すると、少量の制御ロジックおよびデータ ベクタが完了したかどうかを決めるのに使用されるカウンタなどのコア リソースが追加されます。[rst] がオンでこのオプションがオフの場合、シミュレーション速度はかなり遅くなりますが、dout を 0 に保つための余分なロジックは使用されません。
- [en] : ブロックで en ポートが使用されます。
- [nd] : nd (new data) 入力ポートが使用されます。
- [data_valid] : データ有効 (data valid) 出力ポートが使用されます。

Chan In オプション

- [Generate chan_in value in advance] : フィルタで CHAN_IN 値を前もって生成するかどうか指定します。
- [Number of samples] : 前もって計算される CHAN_IN 値の入力サンプル数を指定します。

メモリ オプション

MAC インプリメンテーションのメモリ タイプは、ユーザーが選択することもできますが、最適なインプリメンテーション オプションになるように自動的に選択させることもできます。このオプションは、DA ベースのアーキテクチャでは選択できないようになっています。DSP スライスやエンベデッド乗算器の使用できないファミリの場合は、データおよび係数バッファに制限されており、[Automatic] は選択できないようになっています。[Distributed] を選択すると、フィルタ構造に対して適切なシフト レジスタがインプリメンテーションされます。RAM を [Block] または [Distributed] に選択する場合は、不適切な方を選ぶと、効率の悪いリソース使用になる可能性があるため注意が必要です。ほとんどのアプリケーションで、デフォルトの [Automatic] モードの使用をお勧めします。

- **[Data buffer type]** : データ サンプルを格納するために使用するメモリのタイプを指定します。
- **[Coefficient buffer type]** : 係数を格納するために使用するメモリのタイプを指定します。
- **[Input buffer type]** : データ入力バッファをインプリメントするのに使用されるメモリ タイプを指定します。
- **[Output buffer type]** : データ出力バッファをインプリメントするのに使用されるメモリ タイプを指定します。
- **[Preference for other storage]** : データパスに一般的なストレージをインプリメントするのに使用されるメモリ タイプを指定します。

DSP スライス列オプション

- **[Multi column support]** : DSP スライスを含むデバイス ファミリの場合、高速の大型フィルタをインプリメントすると、複数列にまたがって DSP スライスをチェーン接続する必要のあることもあります。このような場合は、複数れ手うにまたがってフィルタ構造をフォールディングする方法を選択できます。この方法には、そのプロジェクトで選択したデバイスに基づいて自動的に行われる **[Automatic]** と、最初とその後続く列の長さをユーザーが選択する **[Custom]** があります。
 - ◆ **Disable**
 - ◆ **Automatic**
 - ◆ **Custom**
- **[First column length]** : 複数列用 DSP48 フィルタをインプリメントする場合に、最初の列の長さを指定します。これは、**[Multiple DSP48 column support]** を **[Custom]** にした場合にのみ設定できます。
- **[Column wrap length]** : 複数列用 DSP48 フィルタをインプリメントする場合に、後続の列の長さを指定します。この値は、指定した最初の列の長さと同じか、それよりも大きい値にする必要があります。これは、**[Multiple DSP48 column support]** を **[Custom]** にした場合にのみ設定できます。
- **[Inter column pipe length]** : 複数列用 DSP48 フィルタをインプリメントする場合に、列の間のパイプラインの長さを指定します。これは、**[Multiple DSP48 column support]** を **[Custom]** にした場合にのみ設定できます。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

このブロックでは、次のザイリンクス LogiCORE FIR Compiler コアが使用されます。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan			Virtex	
			3、3E	3A	3A DSP	4	5
FIR Compiler 4.0	FIR Compiler	V4.0	•	•	•	•	•

既知の問題

次は、System Generator の FIR Compiler に関する既知の問題です。

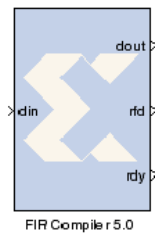
- 現在のところ、FIR Compiler ブロックでは Virtex-4 デバイスの MAC FIR しかサポートされません。
- リロード可能な係数や複数の係数セットは、サポートされません。
- 複数列の自動インプリメンテーションはサポートされません。
- 複数列の自動インプリメンテーションはサポートされません。
- サポートされるタップは、最大 512 タップのみです。
- このブロックを含むデザインをシミュレーションし、ネットリストの段階で次のようなエラーメッセージが表示された場合は、`$target_directory/sysgen/coregen_XXXX/coregen_tmp` ディレクトリの `coregen.log` ファイルを参照してください。

```
standard exception: XNetlistEngine: An exception  
was raised: com.xilinx.sysgen.netlist.f: ERROR:  
coreutil - Failure to generate output products  
at C:/MATLAB701/toolbox/xilinx/sysgen/scripts/  
SgGenerateCores.pm line 590
```

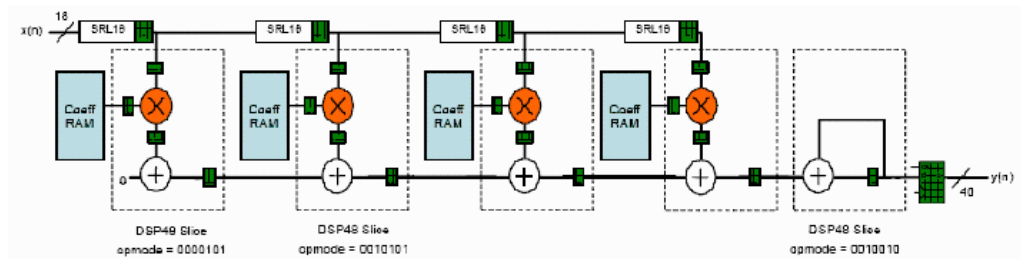
`$target_directory` は、System Generator ブロックで指定されたネットリスト ターゲット ディレクトリを示しています。

FIR Compiler 5.0

このブロックは、[Xilinx Blockset] の [DSP] および [Index] ライブラリにリストされています。



ザイリンクスの FIR Compiler 5.0 ブロックは、積和演算 (MAC) ベースまたは分散演算の FIR フィルタをインプリメントします。このブロックは、入力データのストリームを受信し、フィルタのコンフィギュレーションに応じてフィルタ処理した結果を、固定の遅延で出力します。MAC ベースのフィルタは、次の図のように、カスケード接続された Xtreme DSP スライスを使用してインプリメントされます。



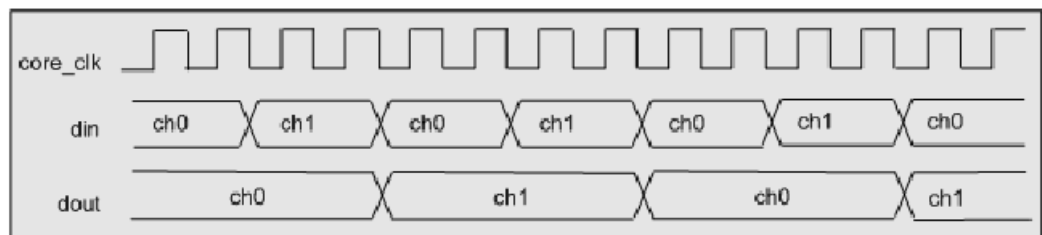
メモ：この後の説明では、DSP48/DSP48E/DSP48A をまとめて「Xtreme DSP スライス」と記述します。

ブロック インターフェイス

FIR Compiler 5.0 ブロックには、すべてのフィルタ コンフィギュレーションで表示される **din** および **dout** に加え、さまざまなオプションのポートを含めることができます。

入力ポート

- **din** : FIR Compiler のデータ入力ポート。このポートを介して、すべてのチャネルのデータがマルチプレクサと同じように、1 度に FIR Compiler に送信されます。



- **rst** : 同期リセット ポート
- **en** : 同期イネーブル ポート
- **nd** : この信号がアサートされると、**din** ポートのデータ サンプルがこのフィルタ コアで受信されます。nd は rfd が Low の間はアサートされるべきではないので、rfd が Low のときのサンプルはこのコアでは無視されます。
- **filt_sel** : 入力信号の F ビット幅のフィルタ選択信号で、 $F = \text{ceil}(\log_2(\text{filter sets}))$ です。複数のフィルタ セットを使用したときだけ、使用できます。

- **coef_ld** : 新しい係数リロード サイクルの開始を示します。
- **coef_we** : 係数をフィルタにロードするために使用され、ホストがインターフェイス上で送信準備できるまでロードを一時停止することができます。
- **coef_din** : 係数をリロードするための入力データ バスです。K は、ほとんどのフィルタ タイプではコアの係数幅で、対称係数構造のない補間フィルタの場合は係数幅 + 2 です。
- **coef_filt_sel** : 係数を読み込むための入力信号の F ビット幅のフィルタ選択信号で、 $F = \text{ceil}(\log_2(\text{filter sets}))$ です。複数のフィルタ セットとリローダブル係数を使用したときだけ、使用できます。

出力ポート

- **dout** : マルチチャネル インプリメンテーションの場合、この出力はすべてのチャネルで時分割されます。
- **rdy** : 新しいフィルタ出力サンプルが **dout** ポートに出力されたことを示します。
- **rfd** : コアが新しいデータ サンプルを受信できる状態かどうかを示します。
- **chan_in** : マルチチャネル インプリメンテーションで、現在の入力があるどのチャネルに向かっているかを示します。
- **FIR Compiler** の **din** ポートが [Simulink System Period] と異なるレートでサンプリングされると、入力ポートがレジスタに取り込まれ、正確なビットおよびサイクルで **System Generator** シミュレーションが実行されるようになります。この結果、**chan_in** 出力が **din** ポートのチャネルデータよりも 1 サイクル遅くなります。たとえば、**chan_in** 出力の値が 1 の場合、**din** でサンプリングされたデータはチャネル 2 に対応します。このビヘイビアは、[Detailed Implementation] タブの [Chan In] オプション エリアで、[Generate chan_in value in advance] をオンにし、[Number of samples] を 1 に設定すると修正できます。
- **chan_out** : コアで生成される標準バイナリ カウントで、現在のフィルタの出力チャネル数を示します。
- **dout_i** : ヒルベルト変換を使用した場合の同相部 (I) データの出力を示します。
- **dout_q** : ヒルベルト変換を使用した場合の直交部 (Q) データの出力を示します。
- **data_valid** : **rdy** と共に、または **rdy** よりも優先して使用できるインジケータ信号です。新しいフィルタ出力サンプルが **dout** ポート (完了したデータ ベクタから生成) に出力されたことを示します。**MAC Based FIR** ブロックのインプリメンテーションに使用できます。

有効な設定とそれに対応する変換サイズについては、LogiCORE データシート v5.0 を参照してください。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、**Simulink** モデル内でブロックをダブルクリックすると表示されます。

[Filter Specification] タブ

[Filter Specification] タブからは、次のようなパラメータを設定できます。

フィルタの係数

- **[Coefficient Vector]** : 指定した係数ベクタが **MATLAB** 行のベクタになります。タップ数は、**MATLAB** 行のベクタ長から推論されます。これらの係数は、**FDATool** ブロックを使用しても入力できます。

- [Number of coefficients sets] : インプリメントされるフィルタ係数のセット数を指定します。指定する値は、余りなしで係数の数に割り切れるようにする必要があります。

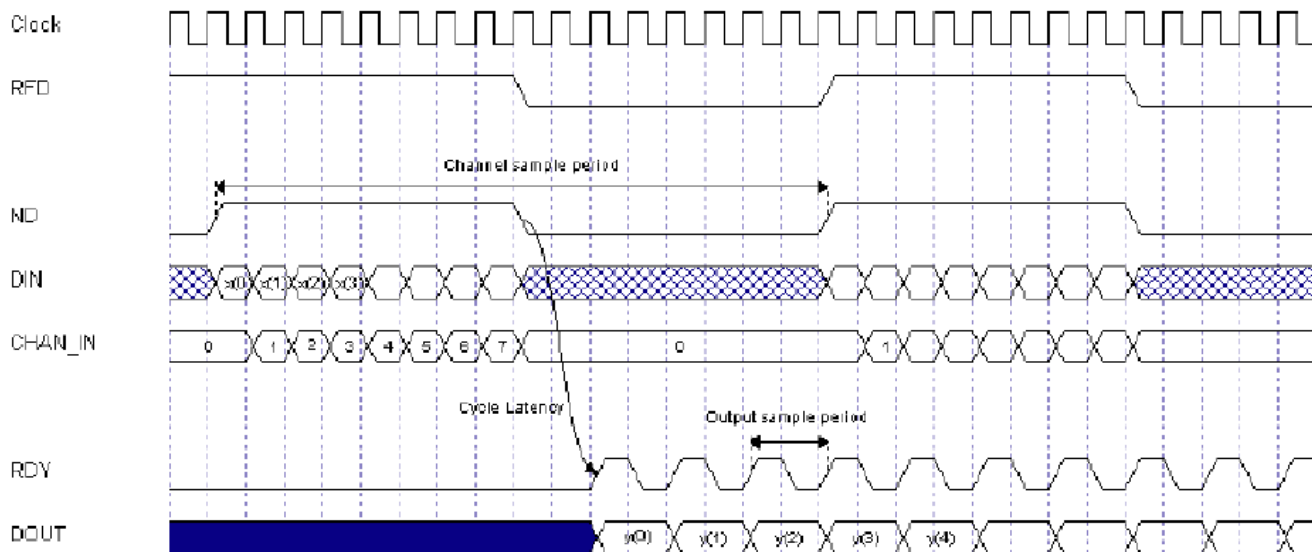
フィルタの指定

- [Filter type] :
 - ◆ [Single_Rate] : 入力と出力のデータ レートが同じになります。
 - ◆ [Interpolation] : 出力のデータ レートが入力よりも [Interpolation Rate Value] で指定した係数分だけ速くなります。
 - ◆ [Decimation] : 出力のデータ レートが入力よりも [Decimation Rate Value] で指定した係数分だけ遅くなります。
 - ◆ [Interpolated] : この FIR フィルタのアーキテクチャは、これまでの FIR フィルタと類似していますが、 $k-1$ ユニットの遅延に置き換えられたユニット遅延演算子が付いているところが違います。 k はゼロ パック ファクタを示します。このタイプのフィルタは上記の [Interpolation] と異なり、狭帯域フィルタを効率的に実現し、少し変更するだけで広帯域も実現可能なシングル レート システムです。入力と出力のデータ レートが同じになります。
 - ◆ [Polyphase_Filter_Bank_Transmitter] : これは、マルチチャネルの FDM (周波数分割多重) デジタル トランスミッタを効率的にインプリメントするために、ザイリンクス FFT コアと共に使用します。
 - ◆ [Polyphase_Filter_Bank_Receiver] : これは、マルチチャネルの FDM (周波数分割多重) デジタル トランスミッタを効率的にインプリメントするために、ザイリンクス FFT コアと共に使用します。
- [Rate change type] :
 - ◆ [Integer] : レート変更の係数を整数で指定します。
 - ◆ [Fixed_Fractional] : レート変更の係数を分数で指定します。
- [Zero packing factor] : 係数ベクタで指定した係数の間にここで指定した数の 0 を挿入できます。ゼロ パック ファクタの k は、提供された係数値の間に $k-1$ 個の 0 を挿入します。このパラメータは、[Filter type] が [Interpolation] に設定されると使用できます。
- [Number of channels] : FIR Compiler ブロックで処理されるデータ チャネルの数を指定できます。複数のチャネル データがマルチプレクサと同じように 1 度にコアに渡されます。サポートされるチャネル数は、最大で 64 です。

ハードウェア オーバーサンプリングの指定

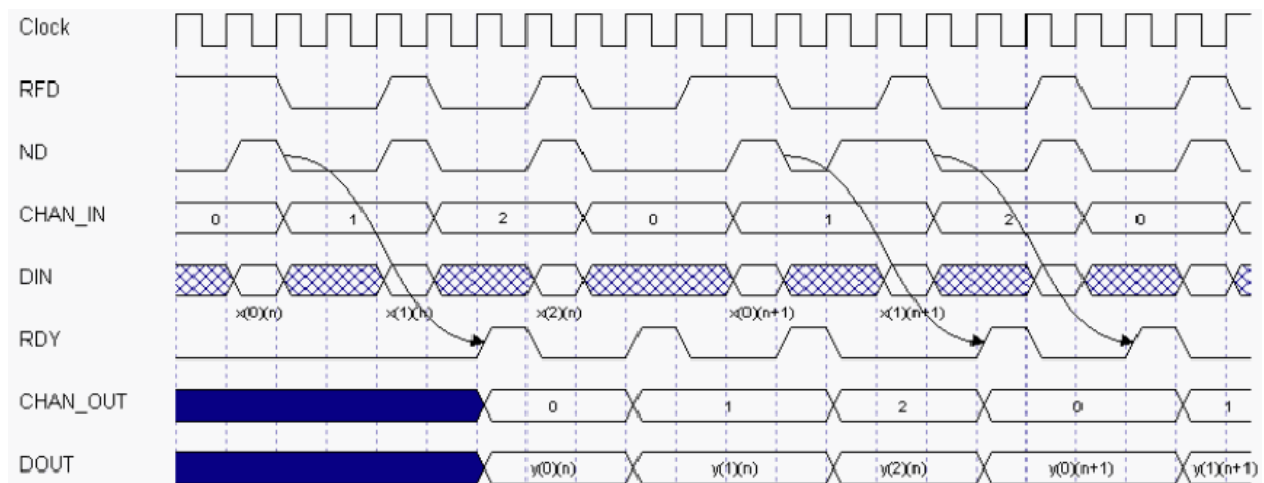
- [Select format] :
 - ◆ [Maximum_Possible] : オーバーサンプリングが din サンプル レートに基づいて自動的に決定されるように指定します。
 - ◆ [Sample_Period] : 下のサンプル周期のダイアログ ボックスが使用できるようになります。サンプル周期を入力します。
 - ◆ [Hardware Oversampling Rate] : [Hardware Oversampling Rate] ダイアログ ボックスが使用できるようになります。[Hardware Oversampling Rate] を入力します。
- [Hardware Oversampling Rate] : 並列処理の度合いを決定します。レートを 1 にすると、完全なパラレル フィルタが生成され、 n ビット信号のレートを n にすると非対称の、 $n+1$ にすると対称のインパルス レスポンスの完全なシリアル インプリメンテーションになります。中間の値にすると、中レベルの並列処理を使用してインプリメンテーションされます。

次の図は、Polyphase_Filter_Bank_Transmitter のタイミング図とその効率的な入力サンプル周波数の計算例を示しています。



この図では、8 チャンネルが含まれ、チャンネル サンプル周期は 16 です。これにより、効率的なサンプル周期は 2 になります。効率的な入力サンプル周期と出力サンプル周期は、Polyphase Filter Bank Transmitter フィルタ タイプでは同じ値になります。

次の図は、ND (New Data) ポートが選択された状態の 3-Channel フィルタの入力タイミングを示しています。この例では、チャンネル サンプル周期は 9 で、効率的なサンプル周期は 3 になっています。データ サンプルの処理に使用可能なクロック サイクルの数は、入力サンプル周期、システム周期、インターポーレーション (補間) レート、デシメーション (間引き) レートによって決まり、コアのインプリメンテーションの並列処理レベルに直接影響します。



[Implementation] タブ

[Implementation] タブからは、次のようなパラメータを設定できます。

- [Filter architecture] には、次のいずれかを選択します。
 - ◆ [Systolic_Multiply_Accumulate] : これは 乗算器/Xtreme DSP スライスをカスケード接続した MAC ベースのアーキテクチャで、
 - ◆ [Transpose_Multiply_Accumulate] : Transpose Multiply-Accumulate アーキテクチャでは、Transposed Direct-Form フィルタがインプリメントされます。
 - ◆ [Distributed_Arithmetic] : Distributed Arithmetic FIR がインプリメントされます。

係数オプション

- [Use reloadable coefficients] : ブロックに係数をリロードするポートを追加するかどうかを指定します。

メモ : このブロックでは `xlGetReloadOrder` 関数がサポートされます。詳細は、`xlGetReloadOrder` を参照してください。

- [Coefficients Structure] : 係数のストラクチャを指定します。係数のストラクチャによって、コアが最適化され、特定のハードウェア コンフィギュレーションをインプリメントするのに必要なハードウェア容量が削減されます。ストラクチャには、次のいずれかを選択できます。
 - ◆ Inferred
 - ◆ Non-Symmetric
 - ◆ Symmetric
 - ◆ Negative_Symmetric
 - ◆ Half_Band
 - ◆ Hilbert

係数のベクタは、ここで指定したストラクチャと一致している必要があります。ただし、[Inferred from Coefficients] を選択した場合、ストラクチャはこれらの係数から自動的に決定されるので、その必要はありません。

- [Coefficient type] : 符号付きか符号なしかを指定します。
- [Quantization] : 係数の量子化方法を次のいずれかに指定します。
 - ◆ Integer_Coefficients
 - ◆ Quantize_Only
 - ◆ Maximize_Dynamic_Range
- [Coefficient width] : 係数を表すために使用するビット数を指定します。
- [Best Precision Fractional Bits] :
- [Coefficient fractional bits] : 係数のデータパス オプションのバイナリの小数点位置を指定します。
- [Number of paths] : フィルタが処理するパラレル データ パスの数を指定します。
- [Output rounding mode] : 次のいずれかを選択します。
 - ◆ Full_Precision
 - ◆ Truncate_LSBs
 - ◆ Non_Symmetric_Rounding_Down
 - ◆ Non_Symmetric_Rounding_Up

- ◆ Symmetric_Rounding_to_Zero
- ◆ Symmetric_Rounding_to_Infinity
- ◆ Convergent_Rounding_to_Even
- ◆ Convergent_Rounding_to_Odd
- [Output width]: 出力幅を指定します。[Rounding Mode] で [Full_Precision] 以外を設定している場合にのみ有効になります。
- [Allow Rounding Approximation]: オンにすると、対称丸め (Symmetric Rounding) を使用している場合に、近似を有効にしてリソースを節約します。

[Detailed Implementation] タブ

[Detailed Implementation] タブからは、次のようなパラメータを設定できます。

- [Optimization Goal]: コアが一番速い速度 ([Speed] オプション)、一番小さいエリア ([Area] オプション) で動作する必要があるかどうかを指定します。デフォルトでは [Area] オプション (推奨) が使用され、通常はそのデザインに最適な速度とエリアを達成されますが、コンフィギュレーションによっては、全体のリソース使用率を改善するために [Speed] 設定が必要なこともあります。[Speed] 設定では、通常クリティカルパスにパイプラインレジスタが追加されます。
 - ◆ Area
 - ◆ Speed

制御オプション

- rst: ブロックで rst ポートが使用されます。このコアでは、rst が使用されると、常に sclr_deterministic オプションが使用されます。sclr_deterministic オプションの詳細は、LogiCORE データシートを参照してください。
- data_valid: データ有効 (data valid) 出力ポートが使用されます。
- nd: nd (new data) 入力ポートが使用されます。
- ce: ブロックでクロック イネーブル ポートが使用されます。

Chan In オプション

- [Generate chan_in value in advance]: フィルタで CHAN_IN 値を前もって生成するかどうか指定します。
- [Number of samples]: 前もって計算される CHAN_IN 値の入力サンプル数を指定します。

メモリ オプション

MAC インプリメンテーションのメモリ タイプは、ユーザーが選択することもできますが、最適なインプリメンテーション オプションになるように自動的に選択させることもできます。

[Distributed] を選択すると、フィルタ構造に対して適切なシフトレジスタがインプリメンテーションされます。RAM を [Block] または [Distributed] に選択する場合は、不適切な方を選ぶと、効率の悪いリソース使用になる可能性があるため注意が必要です。ほとんどのアプリケーションで、デフォルトの [Automatic] モードの使用をお勧めします。

- [Data buffer type]: データサンプルを格納するために使用するメモリのタイプを指定します。
- [Coefficient buffer type]: 係数を格納するために使用するメモリのタイプを指定します。
- [Input buffer type]: データ入力バッファをインプリメントするのに使用されるメモリタイプを指定します。

- **[Output buffer type]** : データ出力バッファをインプリメントするのに使用されるメモリ タイプを指定します。
- **[Preference for other storage]** : データパスに一般的なストレージをインプリメントするのに使用されるメモリ タイプを指定します。

DSP スライス列オプション

- **[Multi column support]** : DSP スライスを含むデバイス ファミリの場合、高速の大型フィルタをインプリメントすると、複数列にまたがって DSP スライスをチェーン接続する必要のあることもあります。このような場合は、複数れ手うにまたがってフィルタ構造をフォールディングする方法を選択できます。この方法には、そのプロジェクトで選択したデバイスに基づいて自動的に行われる **[Automatic]** と、最初とその後に続く列の長さをユーザーが選択する **[Custom]** があります。
- **[Column Configuration]** : カンマ区切りのリストで各列の長さを指定します (詳細は、データシートを参照してください)。
- **[Inter-Column Pipe Length]** : 列同士を接続するためにはパイプライン ステージが必要です。必要なパイプラインのレベルは、必要なシステム クロック レート、選択デバイス、その他システム レベルのパラメータによって異なります。このパラメータは、常にユーザーが指定できるようになっています。

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

このブロックは、次のザイリンクス LogiCORE を使用します。

System Generator ブロック	ザイリンクス LogiCORE™	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
FIR Compiler 5.0	FIR Compiler	V5.0	•	•	•	•	•	•	•	•	•	•

From FIFO

このブロックは、[Xilinx Blockset] の [Shared Memory] および [Index] ライブラリにリストされています。



ザイリンクス From FIFO ブロックは、First-In First-Out (FIFO) のメモリ キューの後半分をインプリメントします。

読み出しイネーブル入力ポートをアサートすると、データはデータ出力ポート (dout) を通って FIFO から読み出されます。FIFO が空の場合は、empty 出力ポートがアサートされます。出力ポート percent_full には、フルの FIFO のパーセントがユーザー指定の精度で表示されます。

From FIFO は、FIFO Generator v2.1 コアを使用してハードウェアにインプリメントされます。System Generator のハードウェア協調シミュレーション インターフェイスでは、From FIFO ブロックをコンパイルし、FPGA ハードウェアで協調シミュレーションできます。共有 FIFO を System Generator 協調シミュレーション ハードウェアで使用すると、ホスト PC と FPGA 間でデータを高速に転送でき、リアルタイム ハードウェア協調シミュレーション機能が強化されます。

9.2 リリースからは、同じ名前の From FIFO ブロックと To FIFO ブロックがペアになり、ネットリストで 1 つの BRAM ベースの FIFO ブロックになっています。From FIFO または To FIFO ブロックが別のブロックとペアにならない場合は、その入力ポートと出力ポートが最上位レベルの System Generator デザインに含まれます。ペアになったブロックはデザインの中の階層にでも配置できますが、同じ名前の From FIFO または To FIFO ブロックが複数ある場合は、エラーになります。

9.2 リリース以前のバージョンとの互換性を保持するには、MATLAB グローバル変数 xISgSharedMemoryStitch を off に設定してください。これには、MATLAB コマンド ラインに次のように入力します。

```
global xISgSharedMemoryStitch;
xISgSharedMemoryStitch = 'off';
```

ブロック パラメータ

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Shared memory name] : 共有 FIFO の名前を付けます。同じ名前の FIFO はすべて同じ物理的 FIFO を共有します。
- [Ownership] : メモリが [Locally owned] か [Owned elsewhere] かを指定します。[Locally owned] の場合、ブロックで FIFO のインスタンスが作成され、[Owned elsewhere] の場合、ブロックが既に作成された FIFO インスタンスに接続されます。
- [Depth] : メモリ ブロックのワード数を指定します。ワード サイズは、din ポートのビット幅から推論されます。
- [Bits of precision to use for %full port] : %full ポートのビット幅を指定します。この符号なし出力の 2 進小数点は、常にワードの最上部にあります。このため、たとえば精度が 1 に設定されると、出力は 0.0 と 0.5 の 2 つの値を取り込みます。この場合、0.5 は FIFO が少なくとも 50% フルであることを示しています。
- [Provide asynchronous reset port] : オプションの非同期のエッジトリガ リセット ポート (rst) が使用されます。11.2 よりも前のリリースでは、このリセットはレベルトリガで、ブロックはこれが High になってもリセット モードのままでした。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

このブロックでは、ザイリンクス LogiCORE FIFO Generator コアを使用してインプリメントされます。

System Generator ブロック	ザイリンクス LogiCORE™	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
FIFO	FIFO Generator	V5.3	•	•	•	•	•	•	•	•	•	•

関連項目

From FIFO ブロックの詳細については、次のトピックを参照してください。

[To FIFO](#)

[Multiple Subsystem Generator](#)

[共有 FIFO の協調シミュレーション](#)

From Register

このブロックは、[Xilinx Blockset] の [Index] ライブラリにリストされています。



ザイリンクス From Register ブロックは、D フリップフロップ ベースのレジスタの後半分をインプリメントします。物理的なレジスタは、2 つのデザインまたは同じデザインの 2 箇所まで共有できます。

このブロックは、レジスタから対応する To Register ブロックで書き込まれたデータを読み出します。dout ポートは、レジスタの出力ポートを表し、ビット幅は対応する To Register ブロックの幅と同じである必要があります。

9.2 リリースからは、同じ名前の From Register ブロックと To Register ブロックがペアになり、ネットリストで 1 つの Register ブロックになっています。From Register または To Register ブロックが別のブロックとペアにならない場合は、その入力ポートと出力ポートが最上位レベルの System Generator デザインに含まれます。ペアになったブロックはデザインの中のどの階層にでも配置できますが、同じ名前の From Register または To Register ブロックが複数ある場合は、エラーになります。

9.2 リリース以前のバージョンとの互換性を保持するには、MATLAB グローバル変数 `xlSgSharedMemoryStitch` を `off` に設定してください。これには、MATLAB コマンド ラインに次のように入力します。

```
global xlSgSharedMemoryStitch;
xlSgSharedMemoryStitch = 'off';
```

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブからは、次のようなパラメータを設定できます。

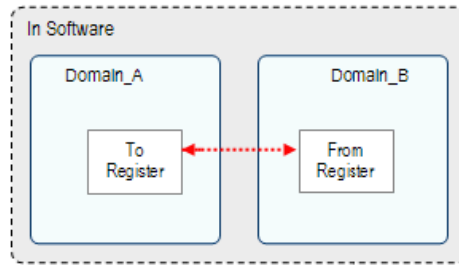
- [Shared memory name] : 共有レジスタの名前を付けます。1 つのレジスタには、同じ名前の To Register ブロックと From Register ブロックがそれぞれ 1 つずつ必要です。この名前は、デザインのほかの共有メモリの名前とは異なるものにしてください。
- [Initial value] : レジスタの初期値を指定します。
- [Ownership and initialization] : メモリが [Locally owned and initialized] か [Owned and initialized elsewhere] かを指定します。[Locally owned and initialized] の場合、ブロックでレジスタのインスタンスが作成され、[Owned and initialized elsewhere] の場合、ブロックが既に作成されたレジスタ インスタンスに接続されます。この結果、シミュレーション中に 2 つの共有レジスタ ブロックが 2 つの異なるモデルで使用される場合、[Locally owned and initialized] のブロックを含むモデルが最初に開始される必要があります。

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

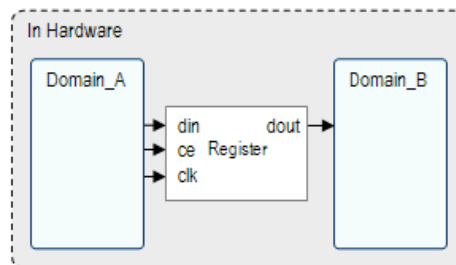
クロック ドメインの切り替え

To Register と From Register ブロックのペアがクロック ドメインの境界を切り替えるために使用されると、ハードウェアには 1 つのレジスタがインプリメントされます。このレジスタには、To Register ブロックのクロック ドメインが使用されます。たとえば、デザインに Domain_A と

Domain_B の 2 つのクロック ドメインがあるとし、次の図のように、2 つのクロック ドメインを切り替えるために 1 組の共有レジスタが使用されるとします。



Multiple Subsystem Generator ブロックを使用してデザインが生成されると、含まれるレジスタは 1 つだけになります。レジスタのクロック信号とクロック ネーブル信号は Domain_A ドメインから駆動されます。



この方法でクロック ドメインを切り替えると、問題になることがあります。安定させるには、2 つの Register ブロックを From Register ブロックの直後に追加し、データを From Register のクロック ドメインに再同期させます。

関連項目

From Register ブロックの詳細については、次のトピックを参照してください。

[To Register](#)

[Multiple Subsystem Generator](#)

[共有レジスタの協調シミュレーション](#)

Gateway In

このブロックは、[Xilinx Blockset] の [Basic Elements]、[Data Types]、[Index] ライブラリにリストされています。



ザイリンクスの **Gateway In** ブロックは、**Simulink** デザインのザイリンクス部分への入力に使用します。このブロックは、**Simulink** の整数、ダブルデータ、固定小数点などのデータ型を **System Generator** の固定小数点型に変換します。各ブロックでは、**System Generator** で生成された HDL デザインの最上位入力ポートを定義します。

ダブルデータ型を **System Generator** の固定小数点型に変換する場合、**Gateway In** ブロックでは選択されたオーバーフローおよび量子化オプションが使用されます。オーバーフローの場合、最大の正の値か最小の負の値にサチュレート (飽和演算) するか、ラップするか (**MSB** の左側のビットを削除)、シミュレーション中に **Simulink** エラーとしてオーバーフローをフラグするオプションがあります。量子化の場合、最近似値 (同等の近似値が 2 つある場合は、0 から遠い方の値) に丸めるか、切り捨てる (**LSB** の右側にビットを削除) オプションがあります。

オーバーフローと量子化は、ハードウェアではなく、ハードウェアの前のブロックのソフトウェア段階で実行されます。

Gateway ブロック

Gateway In ブロックには、次のような機能があります。

- このブロックは、**Simulink** の整数、ダブルデータ、固定小数点などのデータ型を **Simulink** でのシミュレーション中に **System Generator** の固定小数点型に変換します。
- **System Generator** で生成された HDL デザインの最上位入力ポートを定義します。
- **System Generator** ブロックで [Create testbench] をオンにすると、テストベンチのスティミュラスを定義します。この場合、HDL コードの生成中に、**Simulink** のシミュレーション中に発生するブロックの入力が、データファイルでは論理ベクタとして記録されます。HDL シミュレーション中は、最上位レベルのテストベンチに挿入されるエンティティにより、このベクタと **Gateway Out** ブロックから出力されたベクタが予測結果に対してチェックされます。
- 最上位レベルの HDL エンティティで対応ポートに名前を付けます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

[Implementation] タブからは、次のようなパラメータを設定できます。

- **[IOB timing constraint]** : ハードウェアでは、**Gateway In** は入力/出力バッファ (**IOB**) のセットとして認識されます。**IOB** のタイミングに制約を付けるには、[None] と [Data rate] の 2 通りの方法があります。

[None] をオンにすると、**System Generator** で生成される制約ファイル (合成ツールに **XST** を使用する場合は **XCF**、それ以外は **NCF**) に **IOB** のタイミング制約は含まれません。このため、**IOB** から同期エレメントへのバスには制約が付きません。

[Data rate] をオンにすると、**IOB** が動作するデータ レートで制約されます。このレートは、**System Generator** ブロックの [System clock period] の値、およびデザイン内のその他のサンプリング周期に対する **Gateway** ブロックのサンプリング レートによって決定されます。たとえ

ば、次の OFFSET = IN 制約は、10ns のシステム周期で実行される Din という名前の Gateway In ブロック用に生成されています。

```
# Offset in constraints
NET "Din(0)" OFFSET = IN : 10.0 : BEFORE "clk";
NET "Din(1)" OFFSET = IN : 10.0 : BEFORE "clk";
NET "Din(2)" OFFSET = IN : 10.0 : BEFORE "clk";
```

- [Specify IOB location constraints] : オンにすると、IOB ロケーション制約を指定できるテキスト ボックスが表示されるようになります。
- [IOB pad locations] : IOB ピン ロケーションをセル配列として指定できます。ロケーションは、パッケージによって異なります。たとえば、上の例の場合に FG パッケージで Virtex-E 2000 を使用するとすると、Din バスのロケーション制約は C36、B36、D35 に指定できます。これは、XCF (または NCF) ファイルでは次のような制約に変換されます。

```
# Loc constraints
NET "Din(0)" LOC = "D35";
NET "Din(1)" LOC = "B36";
NET "Din(2)" LOC = "C35";
```

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。 Gateway In ブロックでは、ほかのブロックと違い、[Quantization] フィールドで [Round] または [Overflow] フィールドで [Saturate] を選択した場合に、ハードウェア リソースが追加が必要にはなりません。

Gateway Out

このブロックは、[Xilinx Blockset] の [Basic Elements]、[Data Types]、[Index] ライブラリにリストされています。



ザイリンクスの Gateway Out ブロックは、Simulink デザインのザイリンクス部からの出力に使用します。このブロックでは、System Generator の固定小数点型のデータが Simulink のダブルデータ型に変換されます。

Gateway Out ブロックは、コンフィギュレーションによって、System Generator で生成された最上位レベルの HDL デザインの出力ポートを定義できたり、単にハードウェア記述で削除されるテストポイントとして使用したりできます。

Gateway ブロック

Gateway Out ブロックには、次のような機能があります。

- このブロックでは、System Generator の固定小数点型のデータが Simulink のダブルデータ型に変換されます。
- System Generator で生成された最上位レベルの HDL デザインの I/O ポートを定義します。Gateway Out ブロックは、最上位レベルの出力ポートを定義します。
- System Generator ブロックで [Create testbench] をオンにすると、テストベンチ結果のベクタを定義します。この場合、HDL コードの生成中に、Simulink のシミュレーション中に発生するブロックからの出力が、データファイルでは論理ベクタとして記録されます。最上位レベルのポートには、それぞれ HDL コンポーネントが最上位レベルのテストベンチに挿入され、HDL シミュレーション中に予測結果に対してこのベクタがチェックされます。
- 最上位レベルの HDL エンティティで対応出力ポートに名前を付けます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブからは、次のようなパラメータを設定できます。

- [Translate into output port] : オフにすると、このブロックがハードウェアに変換されるときに実際の出力ポートにならないようにできます。デフォルトはオンで、出力ポートがイネーブルになります。オフにすると、Gateway Out ブロックはデバッグ中のみ使用され、デザインを部分的にプローブする際に Simulink の Sink ブロックとの通信に使用されます。この場合、Gateway Out ブロックはグレーで表示され、Gateway ブロックが出力ポートに変換されないことを示します。
- [IOB timing constraint] : ハードウェアでは、Gateway Out ブロックは入力/出力バッファ (IOB) のセットとして認識されます。IOB のタイミングに制約を付けるには、[None] と [Data rate] と [Data rate, set 'FAST' attribute] の 3 通りの方法があります。

[None] をオンにすると、System Generator で生成される制約ファイル (合成ツールに XST を使用する場合は XCF、それ以外は NCF) に IOB のタイミング制約は含まれません。このため、IOB から同期エレメントへのパスには制約が付きません。

[Data rate] をオンにすると、IOB が動作するデータ レートで制約されます。このレートは、System Generator ブロックの [System clock period] の値、およびデザイン内のその他のサンプリング周期に対する Gateway Out ブロックのサンプリング レートによって決定されます。た

たとえば、次の **OFFSET = OUT** 制約は、10ns のシステム周期で実行される **Dout** という名前の **Gateway Out** ブロック用に生成されています。

```
# Offset out constraints
NET "Dout(0)" OFFSET = OUT : 10.0 : AFTER "clk";
NET "Dout(1)" OFFSET = OUT : 10.0 : AFTER "clk";
NET "Dout(2)" OFFSET = OUT : 10.0 : AFTER "clk";
```

[Data rate, set 'FAST' attribute] をオンにすると、上の **OFFSET = OUT** 制約が出力され、各 **IOB** に対して、**FAST** スルー レート属性がさらに生成されます。これにより、遅延は削減されますが、ノイズと消費電力は増加します。前の例の場合、次の属性が **XCF** (または **NCF**) ファイルに追加されます。

```
NET "Dout(0)" FAST;
NET "Dout(1)" FAST;
NET "Dout(2)" FAST;
```

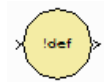
- [Specify IOB location constraints] : オンにすると、**IOB** ロケーション制約を指定できるようになります。
- [IOB pad locations] : **IOB** ピン ロケーションをセル配列として指定できます。ロケーションは、パッケージによって異なります。たとえば、上の例の場合に **FG** パッケージで **Virtex-E 2000** を使用すると、**Dout** バスのロケーション制約は **B34**、**D33**、**B35**、に指定できます。これは、**XCF** (または **NCF**) ファイルでは次のような制約に変換されます。

```
# Loc constraints
NET "Dout(0)" LOC = "B35";
NET "Dout(1)" LOC = "D33";
NET "Dout(2)" LOC = "B34";
```

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

Indeterminate Probe

このブロックは、[Xilinx Blockset] の [Tools] および [Index] ライブラリにリストされています。

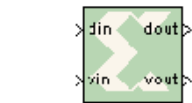


ザイリンクスの **Indeterminate Probe** ブロックの出力は、入力データが不定値 (MATLAB 値が NaN) かどうかを示します。不定データは、VHDL の不定データ値を表す X に相当します。

このブロックは、どのタイプのザイリンクス信号も入力として受信し、ダブル信号を出力します。このブロックに不定データが入力されると、出力信号が 1 にアサートされます。それ以外の場合は、出力は 0 になります。

Interleaver Deinterleaver 5.0

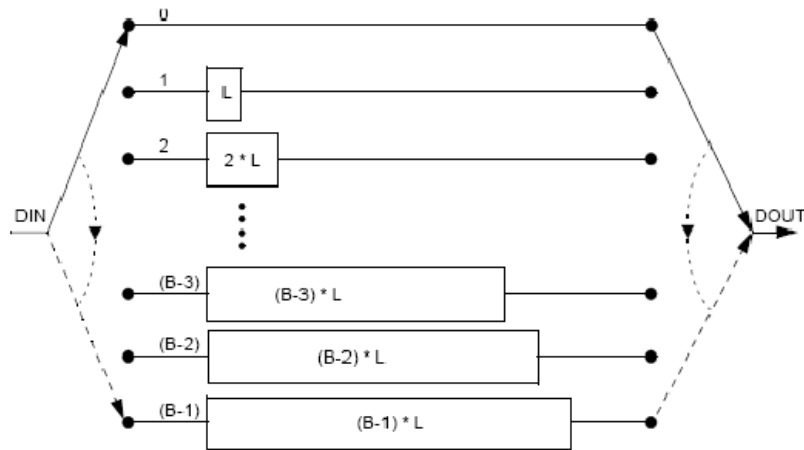
このブロックは、[Xilinx Blockset] の [Communication] および [Index] ライブラリにリストされています。



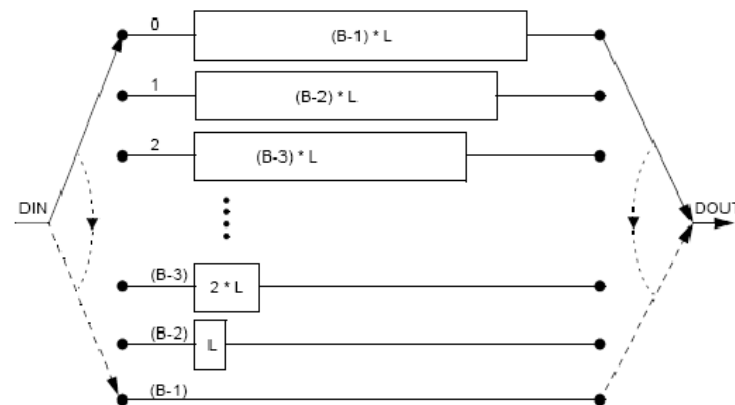
Interleaver Deinterleaver v5_0

ザイリンクスの **Interleaver/Deinterleaver** ブロックは、インターリーバまたはデインターリーバをインプリメントします。インターリーバは、1:1 の確定的な方法でシンボルのシーケンス順序を並び替えるデバイスです。デインターリーバは、この並び替えられたシーケンスを再び格納するデバイスです。

ブロックがインターリーバモードの場合、2つの同期された整流器アーム (図の傍線矢印部分) を使用して、**din** ポートでサンプリングされた入力データが **B** シフトレジスタで多重化され、**dout** ポートへ出力されます。**B** は、ブロックのパラメータ ダイアログ ボックスで入力されたブランチ数です。ブランチ 0 には長さ 0 の、ブランチ 1 には長さ L の、ブランチ 2 には長さ $2L$ のシフトレジスタが含まれます。ブランチ $(B-1)$ には、長さ $(B-1)L$ のシフトレジスタが含まれます。 L は、長さ 1 の場合に配列として入力されたブランチ長を示す定数です。



ブロックがデインターリーバモードの場合、2つの同期された整流器アーム (図の傍線矢印部分) を使用して、**din** ポートでサンプリングされた入力データが **B** シフトレジスタで多重化され、**dout** ポートへ出力されます。ブランチ 0 には長さ $(B-1)L$ の、ブランチ $(B-1)$ には長さ 0 のシフトレジスタが含まれます。



ブランチ長が配列として指定されると、ブロックはインターリーブ モードでもデインターリーブ モードでも同じように動作します。これは、配列ですべてのブランチの長さが完全に定義されるためです。配列には、ブランチ数と同じ長さ **B** が使用される必要があります。

リセット ピン (rst) は、整流器アームをブランチ 0 に設定しますが、データのブランチはクリアされません。

ブロック インターフェイス

Interleaver/Deinterleaver ブロックには 2 ～ 4 個の入力ポートと 2 個の出力ポートがあります。din 入力ポートは 1 ～ 256 ビットにする必要があります。vin ポートは、din ポートの値が有効かどうかを示し、有効なデータのみを多重化し、シフト レジスタから出力します。vout ポートは、dout ポートの値が有効かどうかを示します。出力ポート dout のサイズは入力ポート din のサイズと同じになります。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Mode] : [Interleaver] または [Deinterleaver] を選択します。
- [Number of branches] : 1 ～ 256
- [Lengths of branches] : 1 ～ MAX。MAX の値は、ブランチ数とコア入力のサイズによって異なります。ブランチ長は、長さ 1 またはブランチ数の配列になります。配列サイズが 1 の場合、その値が続くブランチとの定数差として使用されます。それ以外の場合、ブランチはそれぞれ独自の長さになります。

[Implementation] タブ

[Implementation] タブからは、次のようなパラメータを設定できます。

- [Memory type] : [Automatically chosen]、[Block RAM]、[Distributed RAM] のいずれかを選択します。
- [Pipeline for maximum performance] : コアをパイプライン接続します。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

このブロックでは、次のザイリンクス LogiCORE の Interleaver/De-interleaver コアが使用されます。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan				Virtex		
			3、 3E	3A	3A DSP	6	4	5	6
Interleaver Deinterleaver 5.0	Interleaver/ De-Interleaver	V5.0	•				•		

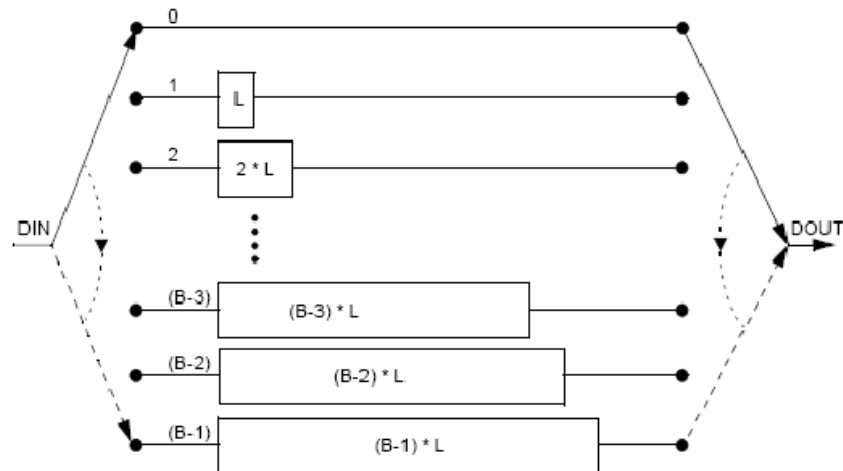
Interleaver Deinterleaver 5.1

このブロックは、[Xilinx Blockset] の [Communication] および [Index] ライブラリにリストされています。

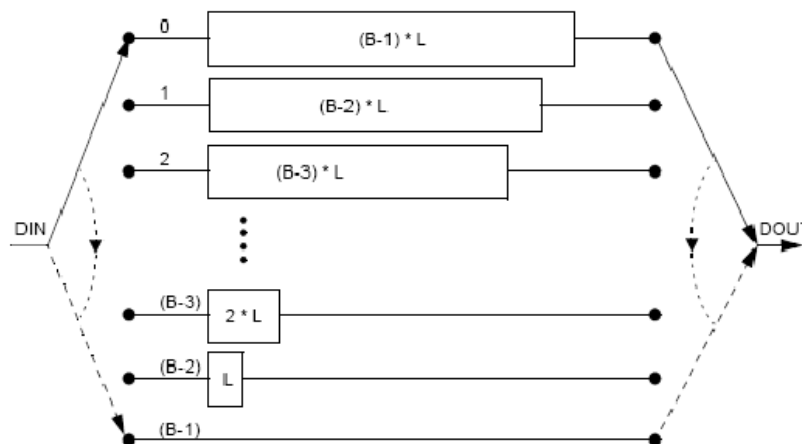


ザイリンクスの Interleaver/Deinterleaver ブロックは、インターリーバまたはデインターリーバをインプリメントします。インターリーバは、1:1 の確定的な方法でシンボルのシーケンス順序を並び替えるデバイスです。デインターリーバは、この並び替えられたシーケンスを再び格納するデバイスです。

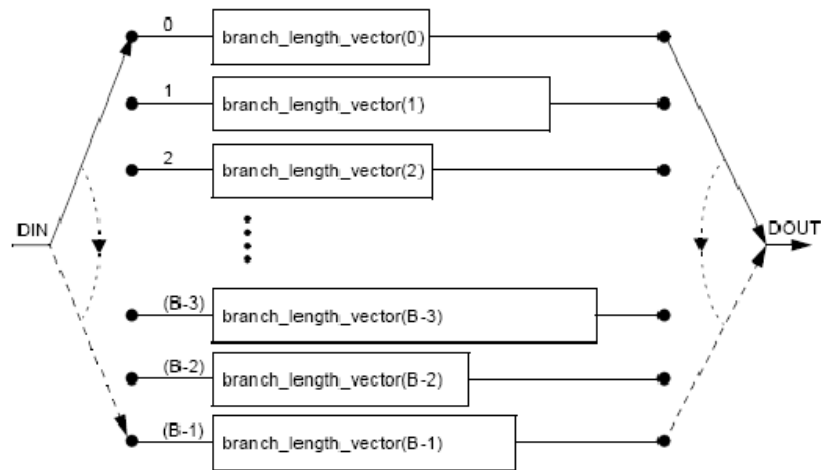
ブロックがインターリーバ モードの場合、2 つの同期された整流器アーム (図の傍線矢印部分) を使用して、din ポートでサンプリングされた入力データが B シフトレジスタで多重化され、dout ポートへ出力されます。 B は、ブロックのパラメータダイアログボックスで入力されたブランチ数です。ブランチ 0 には長さ 0 の、ブランチ 1 には長さ L の、ブランチ 2 には長さ $2L$ のシフトレジスタが含まれます。ブランチ $(B-1)$ には、長さ $(B-1)L$ のシフトレジスタが含まれます。 L は、長さ 1 の場合に配列として入力されたブランチ長を示す定数です。



ブロックがデインターリーバ モードの場合、2 つの同期された整流器アーム (図の傍線矢印部分) を使用して、din ポートでサンプリングされた入力データが B シフトレジスタで多重化され、dout ポートへ出力されます。ブランチ 0 には長さ $(B-1)*L$ の、ブランチ $(B-1)$ には長さ 0 のシフトレジスタが含まれます。



次のようにファイルをブランチ長を指定するために使用した場合、出力されるコアをインターリーバと呼ぶかデインターリーバと呼ぶかは任意です。重要なのは、1 つがもう片方の逆になっているかどうかです。ファイルを使用する場合、各ブランチ長は個別に指定できます。



リセット ピン (rst) は、整流器アームをブランチ 0 に設定しますが、データのブランチはクリアされません。

ブロック インターフェイス

Interleaver/Deinterleaver ブロックには 2 ～ 4 個の入力ポートと 2 個の出力ポートがあります。din 入力ポートは 1 ～ 256 ビットにする必要があります。vin ポートは、din ポートの値が有効であることを示します。有効なデータのみを多重化し、シフト レジスタから出力します。vout ポートは、dout ポートの値が有効であることを示します。出力ポート dout のサイズは入力ポート din のサイズと同じになります。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Mode] : [Interleaver] または [Deinterleaver] を選択します。
- [Number of branches] : 1 ～ 256
- [Lengths of branches] : 1 ～ MAX。MAX の値は、ブランチ数とコア入力のサイズによって異なります。ブランチ長は、長さ 1 またはブランチ数の配列になります。配列サイズが 1 の場合、その値が続くブランチとの定数差として使用されます。それ以外の場合、ブランチはそれぞれ独自の長さになります。

[Implementation] タブ

[Implementation] タブからは、次のようなパラメータを設定できます。

- [Memory type] : [Automatically chosen]、[Block RAM]、[Distributed RAM] のいずれかを選択します。
- [Pipeline for maximum performance] : 最大のパフォーマンスにするために LogiCORE をパイプライン接続します。:
- [Use core placement information] : オンにすると、生成するコアに相対配置情報が含まれます。このため、通常はインプリメンテーションが高速になります。配置ではこの情報に基づいて制約が付くので、配置配線ソフトウェアの速度が遅くなることもあります。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

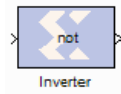
ザイリンクス LogiCORE

このブロックでは、次のザイリンクス LogiCORE の Interleaver/De-interleaver コアが使用されます。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan				Virtex		
			3、 3E	3A	3A DSP	6	4	5	6
Interleaver Deinterleaver 5.1	Interleaver/ De-Interleaver	V5.1	•				•		

Inverter

このブロックは、[Xilinx Blockset] の [Basic Elements]、[Control Logic]、[Math]、[Index] ライブラリにリストされています。



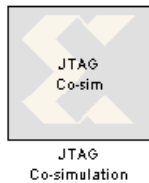
ザイリンクスの **Inverter** ブロックでは、固定小数点型の論理コンポーネントがビット単位で計算されます。このブロックは、合成可能な VHDL モジュールとしてインプリメントされます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

JTAG Co-Simulation



JTAG Co-Simulation ブロックを使用すると、JTAG とパラレル ケーブル IV またはプラットフォーム USB を使用したハードウェア協調シミュレーションを実行できます。このブロックのインターフェイスは、広く普及している JTAG を活用して System Generator の Hardware in the Loop シミュレーションをほかのさまざまな FPGA プラットフォームに拡張しています。

協調シミュレーション ブロックには、さまざまなポート インターフェイスが使用されます。JTAG ハードウェア協調シミュレーションのためにモデルがインプリメントされると、新しいライブラリが作成され、このライブラリに元のモデルのゲートウェイ名 (サブシステムが最上位レベルでない場合はポート名) と同じポート名の付いたカスタムの JTAG Co-Simulation ブロックが含まれます。このブロックは、Simulink のシミュレーション中に、FPGA ハードウェア プラットフォームとの通信に使用されます。このブロックの入力ポートに書き込まれたシミュレーション データがハードウェアに渡されます。データがブロックの出力ポートから読み出されると、ブロックはハードウェアから適切な値を読み込んで、出力ポートに駆動します。これで、データが Simulink で解釈できるようになります。また、ブロックではプラットフォームの開始、コンフィギュレーション、ステップ、終了が自動的に実行されます。

JTAG ハードウェアの要件と新規プラットフォームのサポート方法については、「イーサネットハードウェア協調シミュレーション」を参照してください。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- **[Clock source]** : シングル ステップまたはフリーランニング クロック ソースを選択できます。シングル ステップ クロックを選択すると、1 度に 1 クロック サイクルずつボードをステップできます。各クロック サイクル ステップは、Simulink での一定の時間に相当します。このクロック ソースを使用すると、シミュレーション中の協調シミュレーション ハードウェアのビヘイビアのビット精度および周期精度が、そのサブシステムのシミュレーション ビヘイビアよりも正確になります。シングル ステップが不要で、フリーランニング クロックでボードを実行できる場合もあります。この場合、ボードは Simulink シミュレーションと非同期で動作します。
- **[Has combinational path]** : ハードウェア協調シミュレーション ブロックの出力ポートから同じブロックの入力ポートへの直接組み合わせフィードバック パス (同じブロックの出力ポートから入力ポートへの接続) が必要な場合もあります。出力ポートから入力ポートへの直接フィードバック パスが必要で、デザインに入力ポートから出力ポートへの組み合わせパスが含まれない場合は、このチェック ボックスをオフにすると、デザインでフィードバック パスを使用できるようになります。
- **[Bitstream name]** : JTAG 協調シミュレーション プラットフォーム用の協調シミュレーション FPGA コンフィギュレーション ファイルを指定します。新規の協調シミュレーション ブロックがコンパイル中に作成されると、このパラメータが自動的に設定され、正しいコンフィギュレーション ファイルが使用されます。このパラメータは、コンフィギュレーション ファイルのディレクトリを変更した場合にのみ修正します。

[Advanced] タブ

- **[Skip device configuration]** : オンにすると、シミュレーションの最初のデバイス コンフィギュレーション段階を飛ばします。シミュレーションの最後にリセット (または再プログラム) する必要がない協調シミュレーション デザインでは、オンにした方が効率的です。オンにすると、協調シミュレーション プラットフォームはプログラムされないで、注意が必要です。ハードウェアプラットフォームに協調シミュレーション ビットストリームが読み込まれないで、ハードウェア協調シミュレーションが実行される可能性があります。

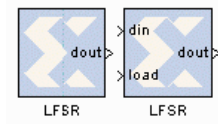
[Cable] タブ

- **[Download cable]** : パラレル ケーブル IV およびプラットフォーム USB ケーブルから、JTAG ハードウェア協調シミュレーション用のケーブルを選択できます。
- **[Cable speed]** : ハードウェア協調シミュレーションでは、プログラム ケーブルをデフォルトの最大速度よりも少ない周波数で動作させる必要があることがあります。このメニューを使用すると、ハードウェア設定にあったケーブル速度を選択できます。通常、デフォルトの速度で問題ありませんが、**System Generator** で協調シミュレーション用にデバイスがコンフィギュレーションできなかった場合は、速度の遅いケーブルを使用してみてください。
- **[Shared cable for concurrent access]** : このオプションを使用すると、JTAG 協調シミュレーション中に JTAG ケーブルが EDK XMD および ChipScope Pro Analyzer と共有されます。このオプションがオンのとき、JTAG 協調シミュレーション エンジンで必要になるのはケーブル アクセス時のロックのみで、アクセスが完了するとこのロックがすぐに解除されます。オフのときは、ロックがシミュレーション中保持されます。ケーブルのロックとロック解除ではオーバーヘッドが大きくなるため、このオプションはデフォルトではディスエーブルにされています。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

LFSR

このブロックは、[Xilinx Blockset] の [Basic Elements]、[DSP]、[Memory]、[Index] ライブラリにリストされています。



ザイリンクスの LFSR ブロックは、LFSR (Linear Feedback Shift Register) をインプリメントします。このブロックでは、XOR または XNOR のいずれかを使用してガロアおよびフィボナッチ構造の両方がサポートされるほか、リロード可能な入力を使用して、レジスタの値をいつでも変更できます。LFSR の出力とリロード可能な入力は、シリアルポートまたはパラレルポートのいずれかでコンフィギュレーションできます。

ブロック インターフェイス

ポート名	ポートの説明	ポート タイプ
din	リロード可能なシード値のデータ入力	オプションのシリアルまたはパラレル入力
load	din のロード信号	オプションのブール入力
rst	リセット信号	オプションのブール入力
en	イネーブル信号	オプションのブール入力
dout	LFSR のデータ出力	必須のシリアルまたはパラレル出力

上の表にあるように、このブロックには 0 ～ 4 の入力ポートと 1 つの出力ポートがあります。選択したコンフィギュレーションで 0 入力が必要とされると、LFSR は指定した初期シード値で開始されるように設定され、LFSR のストラクチャタイプ、ゲートタイプ、初期シード値によって決まる反復可能なステートのシーケンスを実行します。

オプションの din および load ポートを使用すると、ランタイム中に LFSR の現在の値を変更できます。load が読み込みを終了すると、LFSR は 0 入力の場合のように動作し、新しいシーケンスを開始します。このシーケンスは、新しく読み込まれたシード値、およびストラクチャタイプ、ゲートタイプに対して統計的にコンフィギュレーションされた LFSR を基に作成されます。

オプションの rst ポートは、統計的に指定された LFSR の初期シード値をリロードし、rst 信号が Low になるまでリロードし続けます。オプションの en ポートが Low になると、LFSR は en が次に High になるまでその値のままになります。

ブロック パラメータ

ブロック パラメータのダイアログボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Type] : [Fibonacci] または [Galois] を選択できます。このフィールドでは、フィードバックのストラクチャを指定します。[Fibonacci] を選択すると、レジスタチェーンの始まりに XOR (または XNOR) ゲートが 1 つ含まれ、タップと最初のレジスタに送信される出力がこのゲートを介します。[Galois] を選択すると、各タップに XOR (または XNOR) ゲートが 1 つ含まれ、チェーンの最後のレジスタからの出力とそのタップのレジスタへの入力がこのゲートを介します。

- **[Gate type]** : [XOR] または [XNOR] を選択します。このフィールドでは、フィードバック信号で使用するゲートを指定します。
- **[Number of bits in LFSR]** : このフィールドでは、**LFSR** チェーンのレジスタ数を指定します。この値によって、パラレルに選択された場合の入力と出力のサイズが決まります。
- **[Feedback polynomial]** : フィードバック チェーンのタップ ポイントを指定します。入力する値は、16 進数を一重引用符で囲んだ値にする必要があります。この多項式の **LSB** は常に 1 に設定する必要があり、**MSB** は暗示的に 1 になりますが、16 進数入力では指定されません。この論理式の指定方法と反復シーケンスを最大数に設定する方法については、**LFSR** コアのデータシートを参照してください。
- **[Initial value]** : **LFSR** が反復シーケンスを開始するときの初期シード値を指定します。この初期値は、ゲート タイプに **XOR** を選択した場合はすべてを 0 にせず、**XNOR** を選択した場合はすべてを 1 にしないようにしてください。これらの値は、**LFSR** を停止してしまいます。

[Advanced] タブ

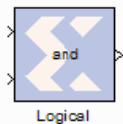
[Advanced] タブからは、次のようなパラメータを設定できます。

- **[Parallel output]** : **LFSR** チェーンのビットすべてが出力に接続されるか、チェーン (シリアルまたはパラレル) の最後のレジスタにのみ接続されるかを指定します。
- **[Use reloadable seed values]** : ランタイム中にダイナミックな **LFSR** シード値をリロードするのに、入力が必要かどうかを指定します。
- **[Parallel input]** : リロード可能な入力シード値を一度に 1 ビット シフトさせるか、パラレルで入力されるかを指定します。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

Logical

このブロックは、[Xilinx Blockset] の [Basic Elements]、[Control Logic]、[Math]、[Index] ライブラリにリストされています。



ザイリンクスの **Logical** ブロックでは、2、3、または 4 つの固定小数点の値のビット単位の論理演算が実行されます。オペランドには、2 進小数点の位置を揃えるため、必要に応じて 0 がパディングされたり、符号が拡張されたりします。これらの値で論理演算が実行され、結果が出力ポートに送信されます。

このブロックは、ハードウェアでは合成可能な VHDL としてインプリメントされます。論理ゲートのツリー構造を作成する場合は、ロジックが合成およびマップでコラプスされるので、この合成可能なインプリメンテーションが最適な方法になります。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Logical function] : ビット単位の論理演算の AND、NAND、OR、NOR、XOR、XNOR のいずれかを選択します。
- [Number of inputs] : ブロックの出力のビット数 (1 ~ 1024) を指定します。

[Output Type] タブ

[Output Type] タブからは、次のようなパラメータを設定できます。

- [Align binary point] : 2 進小数点で自動的に揃えるかどうかを指定します。オフの場合、すべての入力の 2 進小数点の位置が同じになります。

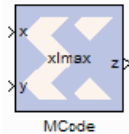
このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

このブロックでは、ザイリンクス LogiCORE は使用されません。

MCode

このブロックは、[Xilinx Blockset] の [Control Logic]、[Math]、[Index] ライブラリにリストされています。



ザイリンクスの MCode ブロックは、Simulink でユーザーの提供する MATLAB 関数を実行するためのコンテナです。M 関数の名前はブロックのパラメータから指定します。このブロックは、M コードを実行して Simulink のシミュレーション中にブロックの出力を計算します。同じコードは、ハードウェアが生成されるときに、同等のビヘイビアレベルの VHDL/Verilog に直接変換されます。

ブロックの Simulink インターフェイスは MATLAB 関数とブロック パラメータを基に決定されます。パラメータごとに関数への入力ポートが 1 つと、関数の返す値それぞれに出力ポートが 1 つ使用されます。ポート名と順序は、パラメータと返し値の名前と順序に対応します。

MCode ブロックでは、限られた MATLAB 言語のサブセットがサポートされています。このサブセットは、演算ファンクション、有限ステート マシン、制御ロジックをインプリメントする場合に便利です。完全な MATLAB アルゴリズムを固定小数点型の FPGA ハードウェアにインプリメントする場合は、ザイリンクスの AccelDSP™ 合成ツールの使用を考慮してください。AccelDSP は、カスタム IP ブロックを高レベルの浮動小数点型 MATLAB から作成するのに使用でき、このブロックはザイリンクス DSP ブロックセットと組み合わせて使用できます。

MCode ブロックを作成する場合、次の 3 つの主なコーディング規則に必ず従ってください。

- すべてのブロックの入力と出力はザイリンクス固定小数点型にする必要があります。
- ブロックには、出力ポートが少なくとも 1 つは必要です。
- ブロックのコードは、MATLAB パスカブロックを使用するモデルと同じディレクトリにある必要があります。

「[MATLAB の FPGA へのコンパイル](#)」には、MCode の関数例が 3 つ記述されています。最初の例には、入力の最大値を返す xymax 関数が含まれ、2 つ目の例は、単純な演算を実行する方法を示しています。3 つ目の例は、有限ステート マシンの構築方法を示しています。詳細については、「[その他の例とチュートリアル](#)」を参照してください。

MCode ブロックのコンフィギュレーション

ブロックの M コード関数の名前は [MATLAB function] で設定します。パラメータを設定するときに、次のいずれかのディレクトリの 1 つにその関数がある必要があります。

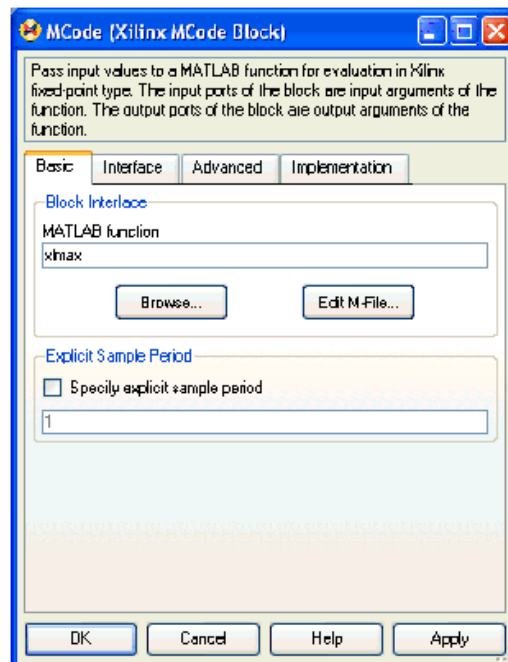
- モデル ファイルがあるディレクトリ
- private という名前のモデル ディレクトリのサブディレクトリ
- MATLAB パスのディレクトリ

ブロックのアイコンには、M 関数の名前が表示されます。たとえば、xymax 関数を含む xymax.m ファイルがあるとします。

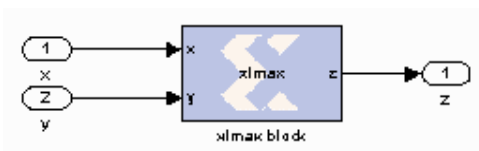
```
function z = xymax(x, y)
    if x > y
        z = x;
    else
        z = y;
    end
```

xymax 関数に基づいた MCode ブロックには、x と y という入力ポートと、z という出力ポートが含まれます。

次の図は、ximax 関数を使用するために MCode ブロックを設定したところを示しています。



このモデルがコンパイルされると、ximax の MCode ブロックは次の図のように表示されます。



MATLAB 言語のサポート

MCode ブロックでは、次の MATLAB 言語構文がサポートされます。

- 代入文
- if/else/elseif と end 文 (そのみでも、組み合わせても使用可能)
- switch 文
- 加算と減算のみを行う演算式
- 加算
- 減算
- 乗算
- 2 のべき乗での除算

- 比較演算子

<	より小さい
<=	それ以下
>	より大きい
>=	それ以上
==	等式
~=	不等式

- 論理演算子

&	And
	Or
~	Not

MCode ブロックでは、次の MATLAB 関数がサポートされます。

- データ型変換。サポートされるデータ型は、ザイリンクスの固定小数点型 `xfix` のみです。`xfix()` 関数は、このデータ型にデータを変換します。変換は整数の場合は陰関数で、固定小数点の定数の場合は陽関数で実行されます。すべての値は、スカラ値にする必要があります。アレイ値はサポートされていません。
- `xfix` プロパティを返す関数：

<code>xl_nbits()</code>	ビット数を返します
<code>xl_binpt()</code>	2 進小数点の位置を返します
<code>xl_arith()</code>	演算タイプを返します
- ビット単位の論理関数

<code>xl_and()</code>	ビット単位の AND
<code>xl_or()</code>	ビット単位の OR
<code>xl_xor()</code>	ビット単位の XOR
<code>xl_not()</code>	ビット単位の NOT
- シフト関数：`xl_lsh()` および `xl_rsh()`
- スライス関数：`xl_slice()`
- 連結関数：`xl_concat()`
- 再解釈関数：`xl_force()`
- 内部トライステート変数：`xl_state()`

- MATLAB 関数

<code>disp()</code>	さまざまな値を表示
<code>error()</code>	メッセージおよび停止を表示
<code>isnan()</code>	数字が NaN かどうかをテスト
<code>NaN()</code>	NaN (Not-a-Number) を返却
<code>num2str()</code>	数値を文字列に変換
<code>ones(1,N)</code>	1 行 N 列の要素がすべて 1 の行列を出力します。
<code>pi()</code>	円周率 (π) を返します
<code>zeros(1,N)</code>	1 行 N 列の要素がすべて 0 の行列を出力します。

データ型

`xfix` データ型には、符号なし固定小数点型 (`xlUnsigned`)、符号付き固定小数点型 (`xlSigned`)、ブール型 (`xlBoolean`) の 3 つがあります。これらのデータ型を演算すると、符号付きまたは符号なしの固定小数点の値になります。比較演算子の結果は、ブール型になります。比較演算子は、データ型を混ぜて使用できる場合は、どの `xfix` 型にでもできます。ブール変数はブール変数と比較されますが、固定小数点型の値とは比較されず、演算子は互換性がありません。論理演算子は、ブール変数にのみ使用できます。演算はすべて完全精度で実行されます。この場合、情報を失わないだけの最小限の精度が使用されます。

リテラル定数

整数型、浮動小数点型、ブール型のリテラル定数がサポートされます。整数のリテラル定数は、2 進小数点の位置が 0 の適切な幅で自動的に `xfix` 値に変換されます。浮動小数点型のリテラル定数は、`xfix()` 変換関数を使用して `xfix` に変換する必要があります。定義済みの MATLAB 値の `true` と `false` は、自動的にブール型のリテラル定数に変換されます。

代入文

代入文の左側に含めることができる変数は 1 つだけです。変数は、何度でも割り当てることができます。

制御フロー

`if` 文を使用した条件文は、ブール型の値を求めるようにする必要があります。`switch` 文には、`case` と `otherwise` 節を含めることができます。`switch` セレクタとその `case` 文のデータ型は互換性があるようにする必要がありますので、`switch` セレクタは `case` 文がそうであれば、ブール型にできます。`switch` 文の `case` はすべて定数にする必要があります。この `case` は入力値には依存できません。

同じ変数を制御文の複数の分岐に代入する場合、代入されたデータ型は互換性があるようにする必要があります。たとえば、次のように指定します。

```
if (u > v)
    x = a;
else
    x = b;
end
```

これが使用できるのは、`a` と `b` の両方がブール型または演算型になっている場合のみです。

定数式

定数式とは、その値を表す定数であり、入力引数の値には依存しません。たとえば、変数 `c` を次のように定義したとします。

```
a = 1;
b = a + 2;
c = xfix({xlSigned, 10, 2}, b + 3.345);
```

これは、定数を要求するどの文でも使用できます。

xfix() Conversion

`xfix()` 変換関数では、`double` が `xfix` に変換されるか、1 つの `xfix` が異なる特性を持つ別のデータ型に変更されます。この変換関数の呼び出し文は、次のようになります。

```
x = xfix(type_spec, value)
```

この場合、`x` は `xfix` を受け取る変数で、`type_spec` は作成する `xfix` のデータ型を指定するセル配列で、`value` は演算される値です。`value` は、浮動小数点型または `xfix` 型にできます。セル配列を示す `type_spec` は、通常の MATLAB 手法では次のように `{ }` を使用して定義します。

```
xfix({xlSigned, 20, 16, xlRound, xlWrap}, 3.1415926)
```

この場合、`xfix` で π の近似値が返ります。この近似値は、符号付きで、20 ビット (小数点以下のビット 16) を占め、丸めで量子化され、オーバーフローでラップされた値になります。

`type_spec` には、1、3、または 5 個のエレメントが含まれます。エレメントの中には、削除できるものもあります。エレメントが削除されると、デフォルトのエレメント設定が使用されます。エレメントでは、データ型、データ幅、2 進小数点の位置、量子化モード、オーバーフロー モードなどのプロパティが指定できます (表示順どおり)。データ型は、`xlBoolean`、`xlUnsigned` または `xlSigned` にできます。`xlBoolean` の場合、追加エレメントは必要なく、追加もできません。ほかのデータ型の場合は、データ幅と 2 進小数点を入力する必要があります。量子化モードとオーバーフロー モードは、オプションですが、どちらか 1 つを指定した場合は、もう一方も必ず指定する必要があります。量子化の値には、`xlTruncate`、`xlRound`、`xlRoundBanker` を使用できます。デフォルトは `xlTruncate` です。オーバーフローの値には、`xlWrap`、`xlSaturate`、`xlThrowOverflow` を使用できます。`xlThrowOverflow` の場合、オーバーフローがシミュレーション中に発生すると、例外が発生します。

`type_spec` の値はすべてコンパイル時に既知の値である必要があります。この `type_spec` 値は関数への入力値には依存できません。

次は、`xfix()` 変換の別の例です。

```
width = 10, binpt = 4;
z = xfix({xlUnsigned, width, binpt}, x + y);
```

この `z` へは、量子化に `xlTruncate`、オーバーフローに `xlWrap` を使用した 10 ビット幅 (小数点以下のビット 4) の符号なし固定小数点型に `x + y` を変換した結果が代入されます。

複数の `xfix()` 呼び出しで同じ `type_spec` 値が必要な場合は、`type_spec` を変数に代入し、`xfix()` 呼び出しでこの変数を使用します。たとえば、次のように使用できます。

```
proto = {xlSigned, 10, 4};
x = xfix(proto, a);
y = xfix(proto, b);
```


xfix プロパティ : xl_arith、xl_nbits、xl_binpt

xfix には、それぞれ演算タイプ、ビット幅、2 進小数点を示す 3 つのプロパティが含まれます。MCode ブロックには、これらの 3 つのプロパティを固定小数点の値で取得するための関数が 3 つ含まれます。この 3 つの関数の結果は定数で、Simulink がモデルをコンパイルしたときに評価されます。

`a = xl_arith(x)` 関数は、入力数 `x` の演算タイプを返します。返される値は、`xlUnsigned` の場合が 1、`xlSigned` の場合が 2、`xlBoolean` の場合が 3 になります。

`n = xl_nbits(x)` 関数は、入力数 `x` の幅を返します。

`b = xl_binpt(x)` 関数は、入力数 `x` の 2 進小数点を返します。

ビット単位の演算子 : xl_or、xl_and、xl_xor、xl_not

MCode ブロックには、`xl_or`、`xl_and`、`xl_xor`、`xl_not` というビット単位の論理演算を行う 4 つのビルトイン関数が含まれています。

関数 `xl_or`、`xl_and`、`xl_xor` は、ビット単位の論理 OR、AND、XOR をそれぞれ実行します。関数はそれぞれ次の形式で記述します。

```
x = xl_op(a, b, Ö).
```

関数はそれぞれ、少なくとも 2 つの固定小数点が入力され、1 つの固定小数点が返されます。入力引数はすべて 2 進小数点で揃えられます。

関数 `xl_not` では、ビット単位の論理 NOT が実行されます。形式は、`x = xl_not(a)` のようになります。入力引数に使用されるのは `xfix` 数 1 つだけで、1 つの固定小数点の値が返されます。

次は、これらの関数を使用した例です。

```
X = xl_and(a, b);
Y = xl_or(a, b, c);
Z = xl_xor(a, b, c, d);
N = xl_not(x);
```

シフト演算子 : xl_rsh および xl_lsh

`xl_lsh` および `xl_rsh` 関数では、固定小数点型ビットのシーケンスをシフトできます。関数は次の形式で記述します。

`x = xl_lsh(a, n)` および `x = xl_rsh(a, n)` (`a` は `xfix` 値で、`n` はシフトするビット数)

固定小数点の値が `n` ビット分左または右にシフトされます。右にシフト (`xl_rsh`) すると、固定小数点が **LSB** に向かって動きます。左にシフト (`xl_lsh`) すると、固定小数点が **MSB** に向かって動きます。どちらのシフト関数も完全精度でシフトします。ビットはどれも削除されず、出力精度は必要であればシフトされた 2 進小数点の位置に合わせて調整されます。

次はその例です。

```
% left shift a 5 bits
a = xfix({xlSigned, 20, 16, xlRound, xlWrap}, 3.1415926)
b = xl_rsh(a, 5);
```

出力 `b` は、21 ビットで、2 進小数点の位置がビット 21 にある `xlSigned` 型です。

スライス関数：xl_slice

xl_slice 関数を使用すると、固定小数点型ビットのシーケンスにアクセスできます。関数は次の形式で記述します。

```
x = xl_slice(a, from_bit, to_bit)
```

固定小数点の各ビットは、LSB の 0 から MSB に向かって、連続した数字でインデックスが付きまゝ。たとえば、8 ビット幅で 2 進小数点の位置が 0 にある場合、LSB のインデックスは 0 に、MSB のインデックスは 7 になります。from_bit または to_bit 引数が入力数のインデックス範囲を超えると、エラーになります。関数を実行した結果は、0 の位置に 2 進小数点がある符合なし固定小数点になります。

次はその例です。

```
% slice 7 bits from bit 10 to bit 4
b = xl_slice(a, 10, 4);
% to get MSB
c = xl_slice(a, xl_nbits(a)-1, xl_nbits(a)-1);
```

連結関数：xl_concat

x = xl_concat(hi, mid, ..., low) 関数では、複数の固定小数点の値が連結され、1 つの固定小数点値が作成されます。最初の実引数が MSB に、最後の引数が LSB になります。出力は、0 の位置に 2 進小数点がある符合なし固定小数点になります。

再解釈関数：xl_force

x = xl_force(a, arith, binpt) 関数は、出力を新しいデータ型にします。arith はその新しい演算型に、binpt はその新しい 2 進小数点の位置になります。arith は、xlUnsigned、xlSigned または xlBoolean のいずれかにできます。binpt は、0 ～ ビット幅にしないと、エラーになります。

ステート変数：xl_state

MCode ブロックには、1 つのシミュレーション段階から次の段階になるときに値を保持する内部ステート変数を含めることができます。ステート変数は、MATLAB キーワードの persistent で宣言でき、最初に xl_state 関数呼び出しで代入する必要があります。

次のコードは、4 ビット アキュムレータの例です。

```
function q = accum(din, rst)
    init = 0;
    persistent s, s = xl_state(init, {xlSigned, 4, 0});
    q = s;
    if rst
        s = init;
    else
        s = s + din;
    end
```

ステート変数 s は、persistent に宣言され、s への最初の代入は xl_state を起動した結果になっています。xl_state 関数では、2 つの引数を使用されます。1 つは初期値で、定数にする必要があります、もう 1 つはステート変数の精度で、xfix 関数呼び出しで説明されているようなセル配列型か、xfix 型にできます。上記のコードの場合、s = xl_state(init, din) では、ステート変数の s に din が精度として使用されます。xl_state 関数は、persistent 変数に代入する必要があります。

xl_state 関数は、次のように動作します。

1. 最初のシミュレーション サイクルで、`xl_state` 関数は指定した精度でステート変数を初期化します。
2. 次のシミュレーション サイクルで、`xl_state` 関数は最後のクロック サイクルから残っているステート値を取り出し、その値を指定した精度で対応する変数に代入します。

ステート変数の値は、`v = xl_state(init, precision)` で返されます。最初の入力引数 `init` は初期値、2 つ目の引数 `precision` はこのステート変数の精度を表します。`precision` は、`{type, nbits, binpt}` または `{type, nbits, binpt, quantization, overflow}` のセル配列型にできます。`precision` は、`xfix` 型の数にもできます。

ベクタのオブジェクトは、`v = xl_state(init, precision, maxlen)` で返されます。ベクタは、`init` で初期化され、最大 `maxlen` までの長さになります。ベクタは、`init` で初期化されるので、たとえば、`v = xl_state(zeros(1, 8), prec, 8)` では 0 が 8 つのベクタが作成され、`v = xl_state([], prec, 8)` では最大長 8 の空のベクタが作成され、`v = xl_state(0, prec, 8)` では最大長 8 の 0 が 1 つのベクタが作成されます。

ベクタ型のステート変数は、両端 (フロント エンドがアドレス 0 のエレメント、バック エンドが長さ - 1 のエレメント) で終わるキューです。

ベクタには、次の計算式が使用できます。

<code>val = v(idx);</code>	アドレス <code>idx</code> のエレメントの値を返します。
<code>v(idx) = val;</code>	アドレス <code>idx</code> のエレメントに <code>val</code> を代入します。
<code>f = v.front;</code>	フロント エンドの値を返します。ベクタが空の場合は、エラーになります。
<code>v.push_front(val);</code>	<code>val</code> をフロント エンドまで押し出し、ベクタの長さを 1 増加します。ベクタがフルの場合は、エラーになります。
<code>v.pop_front;</code>	フロント エンドから 1 つエレメントを取り出し、ベクタの長さを 1 削減します。ベクタが空の場合は、エラーになります。
<code>b = v.back;</code>	バック エンドの値を返します。ベクタが空の場合は、エラーになります。
<code>v.push_back(val);</code>	<code>val</code> をバック エンドまで押し出し、ベクタの長さを 1 増加します。ベクタがフルの場合は、エラーになります。
<code>v.pop_back;</code>	バック エンドから 1 つエレメントを取り出し、ベクタの長さを 1 削減します。ベクタが空の場合は、エラーになります。
<code>v.push_front_pop_back(val);</code>	<code>val</code> をフロント エンドまで押し出し、バック エンドからエレメントを 1 つ取り出します。シフト演算で、ベクタの長さは変わりません。これを実行する場合は、ベクタを空にはできません。
<code>full = v.full;</code>	ベクタがフルの場合は <code>true</code> を、それ以外の場合は <code>false</code> を返します。
<code>empty = v.empty;</code>	ベクタが空の場合は <code>true</code> を、それ以外の場合は <code>false</code> を返します。
<code>len = v.length;</code>	ベクタのエレメント数を返します。

ステート変数を問い合わせるベクタのメソッドは「クエリー メソッド」と呼ばれます。クエリー メソッドは、値を返します。クエリー メソッドには、`v(idx)`、`v.front`、`v.back`、`v.full`、`v.empty`、`v.length`、`v.maxlen` などがあります。ステート変数を変更するベクタのメソッドは、「アップデート メソッド」と呼ばれます。アップデート メソッドは値を返しません。アップデート メソッドには、`v(idx) = val`、`v.push_front(val)`、`v.pop_front`、`v.push_back(val)`、`v.pop_back`、`v.push_front_pop_back(val)` などがあります。ベクタのクエリー メソッドは、すべてシミュレーション サイクル中に、アップデート メソッドよりも前に起動される必要があります。この順序が違うと、モデルのコンパイル中にエラーになります。

MCode ブロックでは、ベクタ型のステート変数がある使用法に基づいて、レジスタのベクタ、遅延ライン、アドレス指定可能なシフト レジスタ、シングル ポート ROM、シングル ポート RAM などにマップされることがあります。`xl_state` 関数は、MATLAB の一次元配列をインデックス 0 の定数配列に変換するために使用することもできます。MCode ブロックでベクタ型のステート変数を FPGA デバイスにマップできない場合は、モデルのネットリスト生成中にエラー メッセージが表示されます。次は、ベクタ型のステート変数を使用した例です。

遅延ライン

次の関数のステート変数は、遅延ラインにマップされます。

```
function q = delay(d, lat)
persistent r, r = xl_state(zeros(1, lat), d, lat);
q = r.back;
r.push_front_pop_back(d);
```

レジスタのライン

次の関数のステート変数は、レジスタのラインにマップされます。

```
function s = sum4(d)
persistent r, r = xl_state(zeros(1, 4), d);
S = r(0) + r(1) + r(2) + r(3);
r.push_front_pop_back(d);
```

定数のベクタ

次の関数のステート変数は、定数のベクタにマップされます。

```
function s = myadd(a, b, c, d, nbits, binpt)
p = {xlSigned, nbits, binpt, xlRound, xlSaturate};
persistent coef, coef = xl_state([3, 7, 3.5, 6.7], p);
s = a*coef(0) + b*coef(1) + c*coef(2) + c*coef(3);
```

アドレス指定可能なレジスタ

次の関数のステート変数は、アドレス指定可能なシフト レジスタにマップされます。

```
function q = addrsr(d, addr, en, depth)
persistent r, r = xl_state(zeros(1, depth), d);
q = r(addr);
if en
    r.push_front_pop_back(d);
end
```

シングル ポート ROM

次の関数のステート変数は、シングル ポート ROM にマップされます。

```
function q = addrsr(contents, addr, arith, nbits, binpt)
proto = {arith, nbits, binpt};
```

```
persistent mem, mem = xl_state(contents, proto);
q = mem(addr);
```

シングル ポート RAM

次の関数のステート変数は、シングル ポート RAM にマップされます。

```
function dout = ram(addr, we, din, depth, nbits, binpt, ram_enable)
    proto = {xlSigned, nbits, binpt};
    persistent mem, mem = xl_state(zeros(1, depth), proto);
    persistent dout_temp, dout_temp = xl_state(0, proto);
    dout = dout_temp;
    dout_temp = mem(addr);
    if we
        mem(addr) = din;
    end
```

次の関数のステート変数は、ブロック RAM にシングル ポート RAM としてマップされます。

```
function dout = ram(addr, we, din, depth, nbits, binpt, ram_enable)
    proto = {xlSigned, nbits, binpt};
    persistent mem, mem = xl_state(zeros(1, depth), proto);
    persistent dout_temp, dout_temp = xl_state(0, proto);
    dout = dout_temp;
    dout_temp = mem(addr);
    if we
        mem(addr) = din;
    end
```

MATLAB 関数

disp()

論理式の値を表示します。MATLAB コンソール ウィンドウに表示させるには、MCode ブロックのパラメータ ダイアログ ボックスの [Advanced] タブで [Enable printing with disp] をオンにしておく必要があります。この引数は、文字列、xfix 型の数字、MCode のステート変数のいずれかにできます。xfix 型の数字の場合は、データ型、2 進数値、倍精度値が表示されます。たとえば、変数 x に xfix({xlSigned, 10, 7}, 2.75) が代入される場合、disp(x) では次が表示されます。

```
type: Fix_10_7, binary: 010.1100000, double: 2.75
```

この引数がベクタ型のステート変数の場合、disp() ではすべてのエレメントの最大長、現在の長さ、2 進数値およびダブル型の値が表示されます。[Enable printing with disp] をオンにしておくと、disp() 関数が起動されたときに、各シミュレーション段階でそのブロックのタイトル ラインが表示されます。タイトル ラインには、ブロック名、Simulink のシミュレーション時間、FPGA のクロック数などが含まれます。

次の MCode の関数には、disp() 関数を使用した例が複数含まれています。

```
function x = testdisp(a, b)
persistent dly, dly = xl_state(zeros(1, 8), a);
persistent rom, rom = xl_state([3, 2, 1, 0], a);
disp('Hello World!');
disp(['num2str(dly) is ', num2str(dly)]);
disp('disp(dly) is ');
disp(dly);
disp('disp(rom) is ');
disp(rom);
```

```

a2 = dly.back;
dly.push_front_pop_back(a);
x = a + b;
disp(['a = ', num2str(a), ', ', ' ', ...
'b = ', num2str(b), ', ', ' ', ...
'x = ', num2str(x)]);
disp(num2str(true));
disp('disp(10) is');
disp(10);
disp('disp(-10) is');
disp(-10);
disp('disp(a) is ');
disp(a);
disp('disp(a == b)');
disp(a==b);

```

次は、最初のシミュレーション段階の結果です。

```

xlmcode_testdisp/MCode (Simulink time: 0.000000, FPGA clock: 0)
Hello World!
num2str(dly) is [0.000000, 0.000000, 0.000000, 0.000000, 0.000000,
0.000000, 0.000000, 0.000000]
disp(dly) is
type: Fix_11_7,
maxlen: 8,
length: 8,
0: binary 0000.0000000, double 0.000000,
1: binary 0000.0000000, double 0.000000,
2: binary 0000.0000000, double 0.000000,
3: binary 0000.0000000, double 0.000000,
4: binary 0000.0000000, double 0.000000,
5: binary 0000.0000000, double 0.000000,
6: binary 0000.0000000, double 0.000000,
7: binary 0000.0000000, double 0.000000,
disp(rom) is
type: Fix_11_7,
maxlen: 4,
length: 4,
0: binary 0011.0000000, double 3.0,
1: binary 0010.0000000, double 2.0,
2: binary 0001.0000000, double 1.0,
3: binary 0000.0000000, double 0.0,
a = 0.000000, b = 0.000000, x = 0.000000
1
disp(10) is
type: UFix_4_0, binary: 1010, double: 10.0
disp(-10) is
type: Fix_5_0, binary: 10110, double: -10.0
disp(a) is
type: Fix_11_7, binary: 0000.0000000, double: 0.000000
disp(a == b)
type: Bool, binary: 1, double: 1

```

この例は、「[MATLAB の FPGA へのコンパイル](#)」に記述されています。

error()

メッセージおよび停止関数を表示します。詳細は、この関数に関する **MATLAB** のヘルプを参照してください。メッセージのフォーマットは、**MCode** ブロックではサポートされていません。次は、この関数の使用例です。

```

if latency <=0
    error('latency must be a positive');
end

```

isnan()

NaN (Not-a-Number) かどうかを返します。isnan(X) の場合、X が Not-a-Number だと true を返します。X は、ダブル型またはザイリンクス固定小数点型のスカラ値にする必要があります。この関数は、ベクタ型または二次元配列型ではサポートされません。次は、この関数の使用例です。

```

if isnan(incr) & incr == 1
    cnt = cnt + 1;
end

```

NaN()

NaN() 関数は、の IEEE 規格の演算式を生成します。NaN は、0.0/0.0 や inf-inf のような数学的に定義されていない演算の結果を表します。NaN(1,N) は、1 行 N 列の要素がすべて NaN 値の行列を出力します。次は、NaN を使用した例です。

```

if x < 0
    z = NaN;
else
    z = x + y;
end

```

num2Str()

数値を文字列に変換します。たとえば、num2str(X) は X を文字列に変換します。X は、ダブル型またはザイリンクス固定小数点型、ベクタ型ステート変数のスカラ値にする必要があります。デフォルトの桁数は、X のエレメントの大きさによって異なります。使用例は次のとおりです。

```

if opcode <=0 | opcode >= 10
    error(['opcode is out of range: ', num2str(opcode)]);
end

```

ones()

ones() 関数は、指定した数の 1 の値を生成します。たとえば、ones(1,N) は、1 行 N 列の要素がすべて 1 の行列を出力します。ones(M,N) の場合、M は 1 である必要があります。この関数は、通常 xl_state() 関数呼び出しと共に使用されます。たとえば、次の例では、[1, 1, 1, 1] に初期化される 1 行 4 列のステート変数が作成されます。

```

persistent m, m = xl_state(ones(1, 4), proto)

```

zeros()

zeros() 関数は、指定した数の 0 の値を生成します。たとえば、zeros(1,N) は、1 行 N 列の要素がすべて 0 の行列を出力します。zero(M,N) の場合、M は 1 である必要があります。この関数は、通常 xl_state() 関数呼び出しと共に使用されます。たとえば、次の例では、[0, 0, 0, 0] に初期化される 1 行 4 列のステート変数が作成されます。

```

persistent m, m = xl_state(zeros(1, 4), proto)

```

FOR ループ

FOR 文は、完全にループ展開されています。次の関数では、n サンプルが合計されます。

```

function q = sum(din, n)
    persistent regs, regs = xl_state(zeros(1, 4), din);

```

```

q = reg(0);
for i = 1:n-1
    q = q + reg(i);
end
regs.push_front_pop_back(din);

```

次の関数では、ビットを反転します。

```

function q = bitreverse(d)
    q = xl_slice(d, 0, 0);
    for i = 1:xl_nbits(d)-1
        q = xl_concat(q, xl_slice(d, i, i));
    end

```

使用可能な変数

MATLAB コードでは、コードが記述順に実行されます。MCode ブロックでは、実行可能なパスが使用される前にそれぞれ変数に値を代入しておく必要があります（代入文の左側は例外）。これで、変数が使用可能な状態になります。使用不可能な変数があると、エラー メッセージが表示されます。

たとえば、次のような M コードがあるとします。

```

function [x, y, z] = test1(a, b)
    x = a;
    if a>b
        x = a + b; y = a;
    end
    switch a
    case 0
        z = a + b;
    case 1
        z = a ñ b;
    end

```

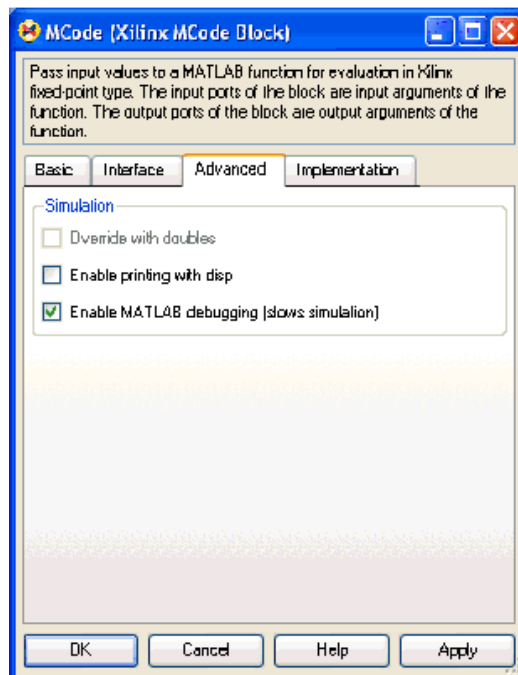
この場合、a、b、x は使用可能ですが、y と z は使用可能ではありません。y が使用できないのは、if 文に else 文がないためで、z が使用できないのは、switch 文に otherwise 部分がないからです。

MCode のデバッグ

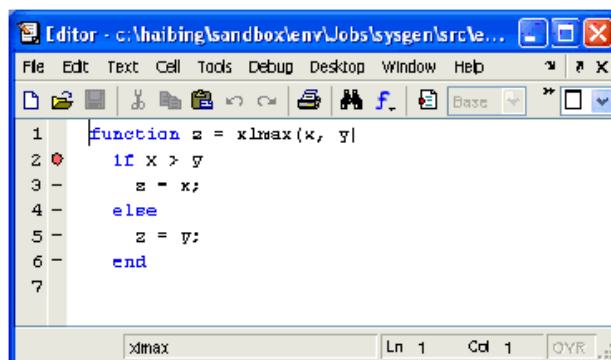
MCode をデバッグするには、コードに disp() 関数を挿入して表示させる方法と、MATLAB デバッガを使用する方法の 2 つがあります。関数の使用方法は、disp() を参照してください。

MATLAB デバッガを使用する場合は、MCode ブロックのパラメータ ダイアログ ボックスの [Advanced] タブで [Enable printing with disp] をオンにしておく必要があります。これで、MATLAB 関数を MATLAB エディタで開いて、ブレークポイントを設定したり、M 関数をデバッグしたりできるようになります。ただし、スクリプトを変更した場合は、必ず MATLAB コンソール ウィンドウで clear 関数コマンドを実行してください。

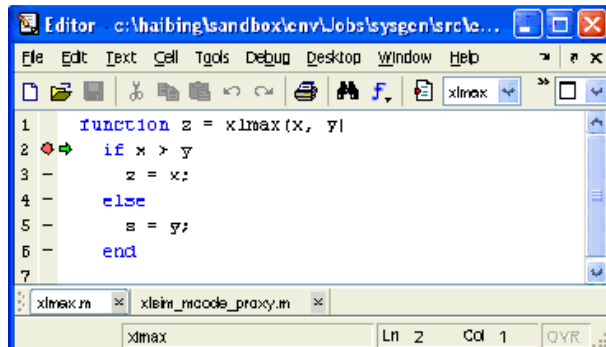
M 関数のデバッグを開始するには、まず MCode ブロックのパラメータ ダイアログ ボックスの [Advanced] タブで [Enable printing with disp] をオンにし、[OK] または [Apply] をクリックする必要があります。



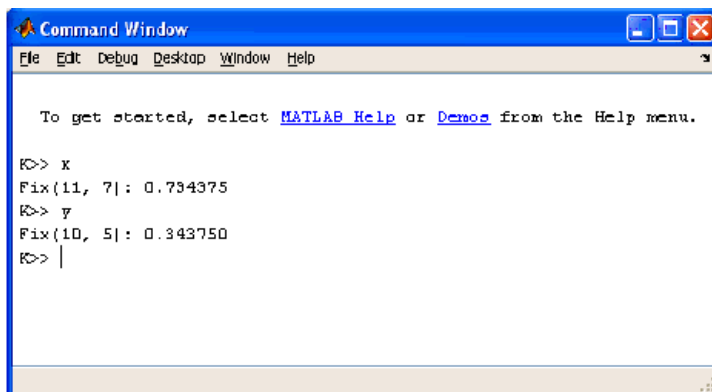
これで、必要に応じて MATLAB エディタで M ファイルを編集したり、ブレークポイントを設定したりできるようになります。



Simulink のシミュレーション中、MATLAB デバッガは設定したブレークポイントで停止します。



デバッグ中は、MATLAB コンソール ウィンドウに変数名を入力すると、その変数の値を確認することもできます。



MCode ブロックの関数が MATLAB デバッガから実行される場合は、次の点に注意してください。MCode 内の switch/case 文は xfix 型にする必要がありますが、MATLAB コンソール ウィンドウから switch/case 文を実行する場合は、double 型または char 型にする必要があります。このため、MATLAB コンソール ウィンドウで実行する場合は、double() を追加する必要があります。たとえば、次のようなコードがあるとします。

```
switch i
case 0
    x = 1
case 1
    x = 2
end
```

i は xfix 型です。コンソール ウィンドウから実行するには、次のコードに変更する必要があります。

```
switch double(i)
case 0
    x = 1
case 1
    x = 2
end
```

double() 関数呼び出しは、M コードがコンソール ウィンドウから実行された場合にのみ使用されます。MCode ブロックでは、double() が無視されます。

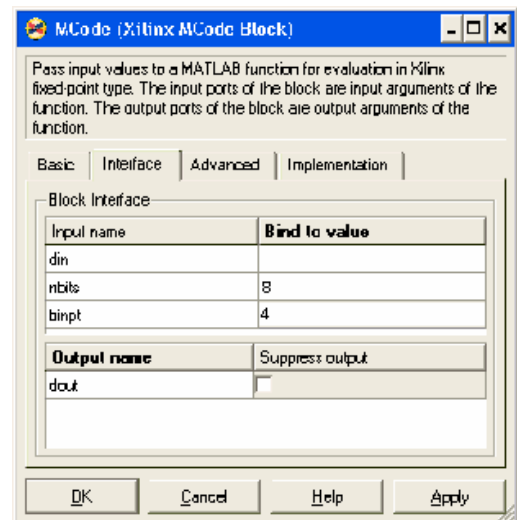
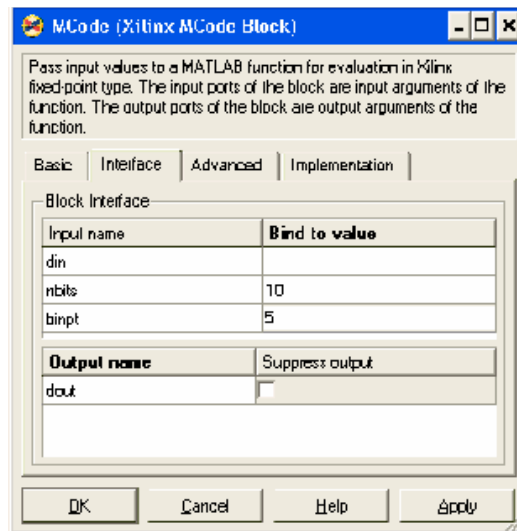
パラメータの伝達

各ブロックの動作が異なるように M 関数に違うパラメータを渡せば、同じ M 関数を別の MCode ブロックで使用することができます。これには、入力引数を複数の値に結合する必要があります。入力引数の結合は、ブロックのパラメータ ダイアログ ボックスの [Interface] タブから設定できます。引数を複数の値に結合すると、これらの M 関数の引数が MCode ブロックの入力ポートとしては表示されなくなります。

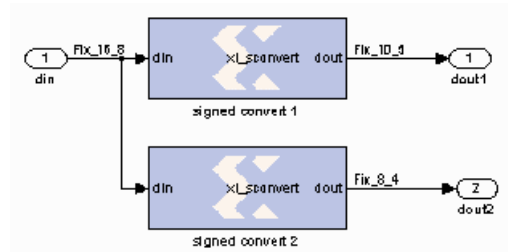
たとえば、次のような M 関数があるとします。

```
function dout = xl_sconvert(din, nbits, binpt)
proto = {xlSigned, nbits, binpt};
dout = xfix(proto, din);
```

次の図では、2 つの異なる xl_sconvert ブロックの din 入力に対して結合を設定しています。



次の図は、モデルをコンパイルした後のブロックを示しています。



パラメータは、ダブル型か論理数にできます。

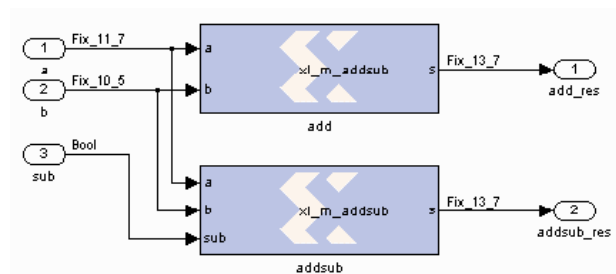
オプションの入力ポート

メカニズムを渡すパラメータを使用すると、MCode ブロックにオプションの入力ポートを付けることができます。たとえば、次のような M 関数があるとします。

```
function s = xl_m_addsub(a, b, sub)
    if sub
        s = a ñ b;
    else
        s = a + b;
    end
```

sub が false に設定されると、この M 関数を使用する MCode ブロックには a と b の 2 つの入力ポートが付き、完全精度の加算が実行されます。空のセル配列 { } に設定されると、a、b、および sub の 3 つの入力ポートが使用され、入力ポート sub の値に応じて完全精度の加算または減算が実行されます。

次の図は、同じ xl_m_addsub 関数を使用した 2 つのブロック (入力ポートが 2 つのブロックと 3 つのブロック) を示しています。



ステート マシンの構築

MCode ブロックを使用してステート マシンを構築する方法は、2 つあります。1 つは、MATLAB 関数を使用して処理状態を把握しない遷移関数を指定し、MCode ブロックと 1 つまたは複数のステート レジスタ ブロックを組み合わせる方法です。通常、MCode ブロックは次のステートを表す値でレジスタを駆動し、レジスタは現在のステートを MCode ブロックに送信します。これには、MCode ブロックからのステート出力の精度をスタティックにする (ブロックへのどの入力からも独立させる) 必要があります。場合によっては、`xfix()` 変換関数を使用して、スタティックな精度にする必要のあることもあります。次のコードは、これを実行する例です。

```
function nextstate = fsm1(currentstate, din)
    % some other code
    nextstate = currentstate;
    switch currentstate
        case 0, if din==1, nextstate = 1; end
    end
    % a xfix call should be used at the end
    nextstate = xfix({xlUnsigned, 2, 0}, nextstate);
```

もう 1 つの方法では、ステート変数を使用します。上記の関数は、次のように書き直すことができます。

```
function currentstate = fsm1(din)
    persistent state, state=xl_state(0,{xlUnsigned,2,0});
    currentstate = state;
    switch double(state)
        case 0, if din==1; state = 1; end
    end
```

ステート変数のリセット信号とイネーブル信号

MCode ブロックでは、ステート変数への条件代入文に含まれる分岐が 2 つ以下の場合、自動的にレジスタのリセット信号およびイネーブル信号を推論させることができます。

たとえば、次の M コードでは `persistent` ステート変数 `r1` の条件代入文でイネーブル信号が推論されます。

```
function myFn = aFn(en, a)
    persistent r1, r1 = xl_state(0, {xlUnsigned, 2, 0});
    myFn = r1;
    if en
        r1 = r1 + a
    else
        r1 = r1
    end
```

この場合、`persistent` ステート変数の `r1` への条件代入文に 2 つの分岐があります。この条件代入を実行するために、レジスタが 1 つ使用されています。レジスタの入力は `r1 + a` に、出力は `r1` に接続されます。このレジスタのイネーブル信号は推論され、`en` がアサートされたときに `en` ポートに接続されます。`persistent` ステート変数の `r1` は `en` が `false` になると `r1 + a` に代入され、レジスタのイネーブル信号がデリアサートされると `r1` が `r1` に代入されます。

次の M コードでも、イネーブル信号が推論されます。

```
function myFn = aFn(en, a)
    persistent r1, r1 = xl_state(0, {xlUnsigned, 2, 0});
    myFn = r1;
    if en
        r1 = r1 + a
```

end

persistent ステート変数の **r1** が定数以外の値 **r1 + a** に条件代入されているので、リセット信号ではなく、イネーブル信号が推論されます。

persistent ステート変数の **r1** への条件代入文に分岐が 3 つあると、イネーブル信号は推論されません。次の **M** コードでは、**persistent** ステート変数の **r1** への条件代入文に分岐が 3 つあるので、イネーブル信号が推論されません。

```
function myFn = aFn(en, en2, a, b)
persistent r1, r1 = xl_state(0, {xlUnsigned, 2, 0});
if en
    r1 = r1 + a
elseif en2
    r1 = r1 + b
else
    r1 = r1
v
```

persistent ステート変数が条件的に定数に代入されると、リセット信号を推論可能です。リセットは同期です。次の **M** コード例では、**rst** が **true** の場合に **persistent** ステート変数の **r1** に定数 **init** が代入され、それ以外の場合は **r1 + 1** が代入され、リセット信号が推論されています。

```
function myFn = aFn(rst)
persistent r1, r1 = xl_state(0, {xlUnsigned, 4, 0});
myFn = r1;
init = 7;
if (rst)
    r1 = init
else
    r1 = r1 + 1
end
```

上のリセットを推論する **M** コード例は、次のように記述することもできます。

```
function myFn = aFn(rst)
persistent r1, r1 = xl_state(0, {xlUnsigned, 4, 0});
init = 1;
myFn = r1;
r1 = r1 + 1
if (rst)
    r1 = init
end
```

上記のどちらのコード例でも、**persistent** ステート変数の **r1** を含むレジスタのリセット信号は **rst** に代入されます。**rst** が **true** になると、レジスタのリセット入力のアサートされ、**persistent** ステート変数が **init** 定数に代入されます。**rst** が **false** になると、レジスタのリセット入力がデアサートされ、**persistent** ステート変数の **r1** が **r1 + 1** に代入されます。**persistent** ステート変数の条件代入に 3 つ以上の分岐が含まれると、**persistent** ステート変数のレジスタでリセット信号は推論されません。

1 つの **persistent** ステート変数のレジスタでリセット信号とイネーブル信号を同時に推論することができます。たとえば、次の **M** コードでは **persistent** ステート変数 **r1** に対し、リセット信号とイネーブル信号が同時に推論されています。

```
function myFn = aFn(rst,en)
persistent r1, r1 = xl_state(0, {xlUnsigned, 4, 0});
myFn = r1;
init = 0;
if rst
    r1 = init
```

```
else
  if en
    r1 = r1 + 1
  end
end
```

persistent ステート変数の **r1** のレジスタのリセット入力、**rst** に接続されます。**rst** が **true** になると、レジスタのリセット入力のアサートされ、**r1** が **init** に代入されます。レジスタのイネーブル入力は **en** に接続されます。**en** が **true** になると、レジスタのイネーブル入力のアサートされ、**r1** が **r1 + 1** に代入されます。推論されたリセット信号は、条件代入文の順序に関係なく、推論されたイネーブル信号よりも優先されます。上の 2 つ目のコード例で、**rst** と **en** の両方が **true** であれば、**persistent** ステート変数の **r1** が **init** に代入されます。

リセット信号とイネーブル信号は、**switch** 文の分岐が 2 つ以下であれば、**switch** 文を使用した **persistent** ステート変数の条件代入でも推論できます。

MCode ブロックでは、FPGA のコードを生成する際に、実行されないコードは削除され、定数伝搬コンパイラで最適化が実行されます。これにより、条件文の分岐の 1 つが 1 度も実行されない場合、**persistent** ステート変数の条件代入文でリセット信号とイネーブル信号のどちらか、または両方が推論されます。これには、条件文に 2 つの分岐が含まれ、それらが実行されないコードが削除されて定数伝搬が実行された後に実行されている必要があります。

組み合わせロジックのパイプライン接続

生成された MCode ブロックの FPGA ビットストリームには、さまざまなレベルの組み合わせロジックが含まれるので、大きなクリティカルパス遅延が発生することがあります。この組み合わせロジックをダウンストリームのロジック合成ツールで自動的にパイプライン接続させるには、MCode ブロックの入力の前か出力の後に遅延ブロックを追加します。これらの遅延ブロックでは、**[Implement using behavioral HDL]** パラメータをオンにして、コードジェネレータで合成可能な HDL を使用した遅延がインプリメントされるようにします。この後、ダウンストリームのロジック合成ツールでレジスタのリタイミングまたはレジスタ自動調整をインプリメントするように命令します。または、ベクタ型のステート変数を使用して遅延を指定する方法もあります。

乗算と除算を使用したシフト演算

MCode ブロックでは、数値が 2 のべき乗の定数で乗算または除算されるタイミングを検出できます。検出すると、MCode ブロックはシフト演算を実行します。たとえば、4 で乗算することは 2 ビット左にシフトすることと同じであり、8 で除算することは 3 ビット右にシフトすることと同じです。シフトは、2 進小数点を調整して、**xfix** データ型のコンテナを必要に応じて拡張するとインプリメントされます。たとえば、**Fix_8_4** 値を 4 で乗算すると **Fix_8_2** 値になり、**Fix_8_4** を 64 で乗算すると **Fix_10_0** 値になります。

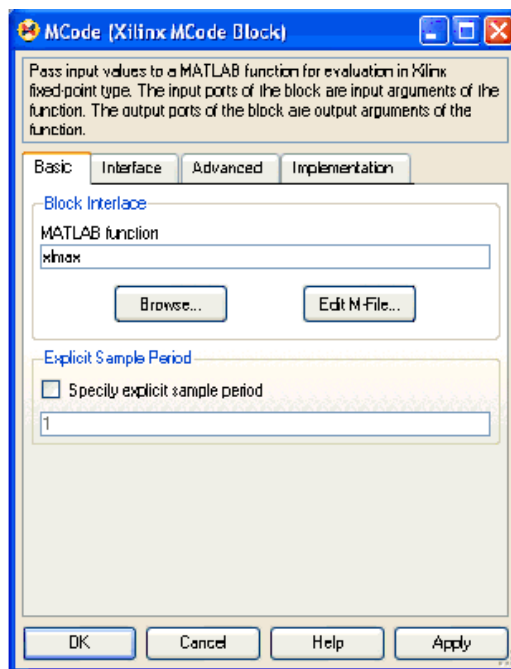
丸めを使用した xl_state 関数

xl_state 関数では、ステート変数用に xfix 型のコンテナが作成されます。このコンテナの精度は、xl_state 関数に渡される 2 つ目の引数で指定されます。精度が丸めモードの場合は xlRound が使用され、丸めに必要なハードウェア リソースが追加されます。初期値の丸めのみが必要とされる場合は、ハードウェア リソースを追加する必要がありません。この後、丸め値が xl_state 関数に渡されます。次は、その例です。

```
init = xfix({xlSigned,8,5,xlRound,xlWrap}, 3.14159);
persistent s, s = xl_state(init, {xlSigned, 8, 5});
```

ブロック パラメータ ダイアログ ボックス

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

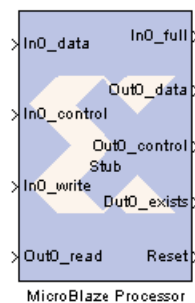


前述したように、MCode ブロックの関数名は [MATLAB function] で指定し、定数入力とその値のリストは [Interface] タブで指定します。

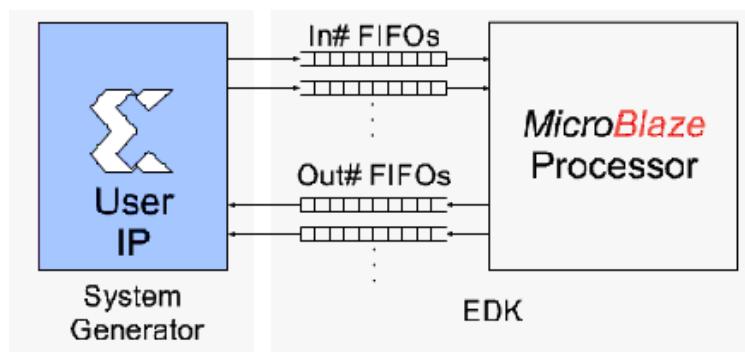
このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

MicroBlaze Processor

このブロックは廃止されています。EDK Processor ブロックを使用してください。



MicroBlaze™ Processor ブロックを使用すると、EDK の MicroBlaze プロセッサ用に作成されたペリフェラルをデザインおよびシミュレーションできます。次の図に示すように、プロセッサで使用可能な FSL (Fast Simplex Link) を介して、カスタマイズした IP を MicroBlaze に接続できます。FSL は、一方向の FIFO と考えることができます。MicroBlaze には、最大で 8 つの入力 FSL と 8 つの出力 FSL の合計 16 個の FSL を含めることができます。同期 FIFO と非同期 FIFO のどちらも使用できますので、MicroBlaze プロセッサに対して同期または非同期に動作する System Generator ペリフェラルを作成可能です。次の図に示すとおり、FSL FIFO は [EDK \(エンベデッド開発キット\)](#) ツールの左側に配置されます。このようなシステムのシミュレーションには、ハードウェア協調シミュレーションを使用します。シミュレーション モデルの作成、管理、設定は、ブロックのパラメータ ダイアログ ボックスから指定できます。詳細は、この後の説明を参照してください。



ブロック インターフェイス

この説明で、シンボル#の番号は0～7を表します。このブロックの入力および出力の数はユーザーがコンフィギュレーションできます。それぞれの入力インターフェイスで、入力ポート 3 つ (In#_data、In#_control、In#_write) と出力ポート 1 つ (In#_write) の合計 4 つのポートが作成されます。同様に、それぞれの出力インターフェイスで、入力ポート 1 つ (Out#_read) と出力ポート 3 つ (Out#_data、Out#_control、Out#_exists) の合計 4 つのポートが作成されます。最大で 8 入力と 8 出力のインターフェイスがサポートされます。オプションの Rst ポートは [Provide reset port] をオンにすると使用できます。Rst ポートは、MicroBlaze Processor ブロックで出力ポートとして表示されます。このポートは、MicroBlaze のどの非同期リセット ポート/ピンにでも接続でき、MicroBlaze から接続された System Generator デザインをリセットできます。

ポート

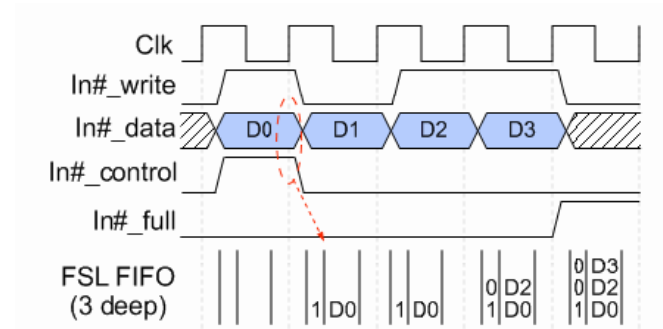
各ポートの説明を、次の表に示します。

ポート名	ポートタイプ	ポート幅 (ビット)	ポートの説明	FSL 接続
In#data	入力	32	FSL FIFO にデータを書き込みます。	MFSL (Master FSL) 接続
In#_control	入力	1	ビットをフラグします。High になると、FIFO に書き込まれたデータが制御ワードになります。	
In#_write	入力	1	High になると、FSL FIFO への書き込みがイネーブルになります。EDK の MicroBlaze プロセッサはスレーブ ペリフェラルとして接続されます。	
In#_full	出力	1	High になると、FSL FIFO がフルであることを示します。	
Out#_data	出力	32	FSL FIFO からデータを読み出します。	SFSL (Slave FSL) 接続
Out#_control	出力	1	ビットをフラグします。High になると、FIFO から読み出されたデータが制御ワードになります。	
Out#_read	入力	1	High になると、FSL FIFO からの読み出しがイネーブルになります。EDK の MicroBlaze プロセッサはマスタ ペリフェラルとして接続されます。	
Out#exists	出力	1	High になると、FIFO が空ではないことを示します。	
Rst	出力	1	非同期リセット ポート/ピンのステータスを示します。	なし

次の図は、それぞれ読み出しおよび書き込み操作が問題なく実行されるための、ポートのさまざまな信号間のタイミング関係を示しています。

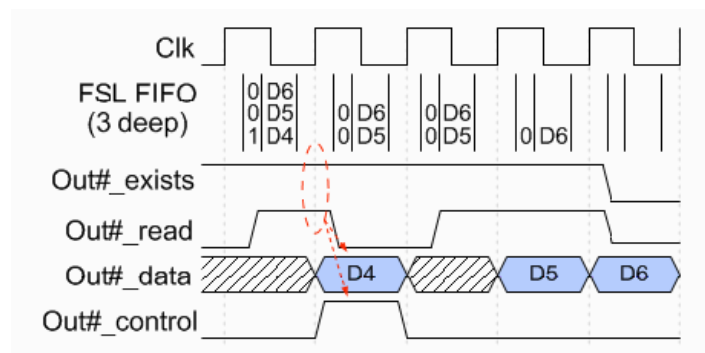
書き込み操作

次のタイミング図に示すように、In#_full が Low の場合、In#_write が High になり、In#_data の値が FIFO に書き込まれます。FIFO がフルになると、In#_full が High になり、FIFO への書き込みは無視されます。In# ports は、接続される MFSL を制御するので、EDK プロジェクトでマスタ ペリフェラルとしてコンフィギュレーションされる必要があります。



読み出し操作

次のタイミング図に示すように、Out#_exists が High のときに Out#_read を High にすると、FIFO の最初のデータが読み出されます。データは、Out#_data ポートから読み出されます。FIFO が空になると Out#_exists が Low になります。空の FIFO から読み出すと、定義されていない値が返されます。Out# ports は、接続される FSL に対してスレーブになるので、EDK プロジェクトでスレーブ ペリフェラルとしてコンフィギュレーションされる必要があります。



System Generator で FSL ペリフェラルをデザインしたら、そのモデルを EDK のエクスポート ツールを使用して EDK 環境に抽出する必要があります。System Generator と VHDL ソース コードを使用すると、このブロックの MPD (Microprocessor Peripheral Definition)、PAO (Peripheral Analyze Order)、BBD (Black Box Definition) ファイルが作成されます。詳細については、「[EDK Export Tool](#)」を参照してください。

メモ：In# と Out# の FSL へのマップは、EDK 環境から制御できます。たとえば、In0 を必ずしも SFSL0 に接続する必要はなく、同様に、Out0 も MSFL0 に接続する必要はありません。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

[General] タブ

このタブでは、**MicroBlaze Processor** ブロックをカスタマイズし、ハードウェア協調シミュレーション機能を使用できるようにします。

[General] タブからは、次のようなパラメータを設定できます。

- **[Number of input interfaces]** : **System Generator** から **MicroBlaze** への **FSL** インターフェイスの数を指定します。このブロックのインターフェイスは、入力ポートと出力ポートの数でコンフィギュレーションされます。入力インターフェイスの数は、8 以下にする必要があります。
- **[Number of output interfaces]** : **MicroBlaze** から **System Generator** への **FSL** インターフェイスの数を指定します。このブロックのインターフェイスは、入力ポートと出力ポートの数でコンフィギュレーションされます。出力インターフェイスの数は、8 以下にする必要があります。
- **[Provide Reset Port]** : ブロック インターフェイスに **Rst** 出力ポートが追加されます。これで **System Generator** デザインを **MicroBlaze** プロセッサからリセットできるようになります。
- **[Provide processor model]** : **MicroBlaze** のハードウェア協調シミュレーション機能を使用できるようにします。オンにすると、**[Hardware]** タブと **[Software]** タブが使用できるようになります。

[Hardware] タブ :

このタブでは、**MicroBlaze Processor** ブロックをカスタマイズし、ハードウェア協調シミュレーション機能を使用できるようにします。

[Hardware] タブからは、次のようなパラメータを設定できます。

- **[Simulation model]** : ブロックが最初に使用されるときは **System Generator** プロセッサ コアのキャッシュは空なので、**[Simulation Model]** プルダウン メニューにはシミュレーション モデルが何も表示されません。
シミュレーション モデルは、ハードウェア プラットフォーム (ボード) に接続し、**RS232** ポートが使用可能かどうかのどのボード特有の情報を **MicroBlaze** プロセッサに認識させる必要があります。シミュレーション モデルがコアのキャッシュにあると、このプルダウン メニューにリストされます (例 : `companyxyz_boardnm_partnm_packagenum_rev_1`)。
- **[Add simulation model]** : このボタンをクリックすると、**[Hardware Co-Simulation Targets]** ダイアログ ボックスが表示され、コンパイルするターゲットが指定できるようになります (詳細は、「[System Generator のコンパイル タイプ](#)」を参照してください)。**[Generate]** ボタンをクリックすると、次のようにコンパイルされます。コンパイルには、多少時間がかかります。
 - a. **EDK** プロジェクトが **System Generator** のブロックで指定したターゲット ディレクトリに作成されます (**XPS**)。
 - b. **EDK** プロジェクトのネットリストが生成されます (**EDK+XFlow**)。
 - c. **EDK** ネットリストからハードウェア協調シミュレーションが作成され、**System Generator** コアのキャッシュに保存されます (**System Generator**、**XFlow**)。
 - d. **EDK** ソフトウェア ライブラリがコンパイルされます (**EDK**)。

メモ : 新しいボード サポート パッケージを作成する場合 ([新規プラットフォームのサポート](#))、そのボードのシステムのリセット ポートと **RS232** ポートをメモリ マップされないポートとして指定しておかないと、正しく配線がされません。また、リセット ポートの名前は **Reset** に、**RS232** ポート

の名前は **RXD** に、送信ポートの名前は **TXD** にしておく必要があります。このように命名しないと、**System Generator** で信号が正しく検出されず、関連するピンに配線されません。

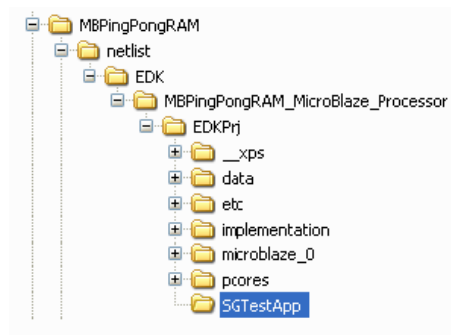
RS232 ポートが使用できる場合は、**MicroBlaze** プロセッサに **UART** が次のパラメータで作成されます。

[Baud rate (bits per second)] : 115200、[Data bits] : 8、[Parity] : None、[Stop bits] : 1、[Flow control] : none。stdin および stdout チャンネルは、**UART** にマップされ、**RS232** を介して入力と出力ができるようになります。

[Software] タブ

シミュレーション モデルで実行されるソース コードの編集およびコンパイルをするボタンが含まれます。

- **[Edit Source Code]** : この **MicroBlaze** ブロックに関連するソース コードが開きます。このコードは、シミュレーション モデルでのみ使用されます。



上の図は、**System Generator** で作成されるディレクトリ構造を示しています。この場合、**MBPingPongRAM** が **Simulink** モデルの名前で、**netlist** フォルダがユーザーの指定したターゲット ディレクトリになります。**EDK** プロジェクトは、モデル名を **MicoBlaze** の完全パスに変更して作成されたディレクトリの下に生成されます。このブロックに関連付けられたソースコードは、ハイライトされたディレクトリ (**SGTestApp**) に **MainProg.c** という名前で保存されています。

ソース ファイルの編集には、デフォルトの **MATLAB** エディタが使用されます。これは、ユーザーが **MATLAB** から設定できるオプションで、**MATLAB** で [ファイル] → [設定] → [エディタ/デバッガ] をクリックすると変更できます。

- **[Compile Source Code]** : クリックすると、ソース コードがコンパイルされ、ハードウェア協調シミュレーションのビット ファイルが作成した 2 進数コードでアップデートされます。

MicroBlaze ソフトウェアの問題

ソフトウェアからの FSL ペリフェラルへのアクセス

System Generator ペリフェラルには、接続された FSL のアセンブリ命令を介して MicroBlaze プロセッサからアクセスできます。EDK には、FSL からの読み出しおよび FSL への書き込みを単純化するための C マクロが 8 つ提供されています。詳細については、[EDK マニュアル](#)を参照してください。

ノンブロッキング データ リードとデータ ライト

<code>microblaze_nbread_datafsl(val,id);</code>

<code>microblaze_nbwrite_datafsl(val,id);</code>
--

ノンブロッキング コントロール リードとコントロール ライト

<code>microblaze_nbread_cntlfsl(val,id);</code>

<code>microblaze_nbwrite_cntlfsl(val,id);</code>
--

ブロッキング データ リードとデータ ライト

<code>microblaze_bread_datafsl(val,id);</code>
--

<code>microblaze_bwrite_datafsl(val,id);</code>

ブロッキング コントロール リードとコントロール ライト

<code>microblaze_bread_cntlfsl(val,id);</code>
--

<code>microblaze_bwrite_cntlfsl(val,id);</code>

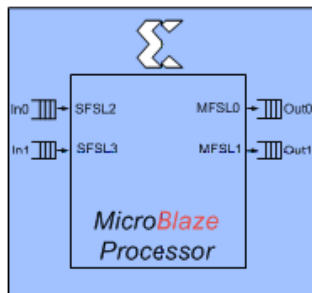
上のマクロでは、`val` は FSL から読み出される、または FSL に書き込まれる 32 ビットのデータ値を示し、`id` はアクセスされた FSL を示します。ブロッキング リードまたはライトを実行すると、MicroBlaze は読み出しまたは書き込みがあるまで停止します。ノンブロッキング リードまたはライトを実行すると、読み出しまたは書き込みが終了できない状態であっても、MicroBlaze は停止しません。データ リードは、`val` を FSL のデータ ポートに、0 を FSL の制御ポートに書き込み、コントロール ライトは、`val` を FSL のデータ ポートに、1 を FSL の制御ポートに書き込みます。詳細については、[EDK の MicroBlaze のマニュアル](#)と、次の System Generator のチュートリアルを参照してください。

- [MicroBlaze プロセッサ ペリフェラルの設計とエクスポート](#)
- [チュートリアル : MicroBlaze プロセッサ システムの設計とシミュレーション](#)

EDK の FSL バスと System Generator ポート間の通信

FSL インスタンスと EDK の FSL バス接続は異なります。MicroBlaze プロセッサには、FSL (もしくは FSL インターフェイスを模倣したペリフェラル) に接続できるバス接続が 16 個あります。このうち 8 つが入力で、EDK では SFSL (Slave FSL) と呼ばれ、残りの 8 つが出力で、EDK では MFSL (Master FSL) と呼ばれます。FSL インスタンスは、FIFO ハードウェアの物理的なインプリメンテーションにより作成されます。

EDK では、SFSL と MFSL バス接続の番号は FSL インスタンスに依存しません。たとえば、SFSL0 は必ずしも FSL のインスタンス 0 に接続されるわけではなく、同様に、MFSL1 は FSL のインスタンス 1 に接続されるわけではありません。つまり、System Generator ブロックの In0_* ポートは SFSL0 に接続する必要はありません。このため、ユーザーが FSL 接続を把握しておかないと、正しいソフトウェアコードが記述できません。たとえば、FSL0 に対する `microblaze_nbwrite_datafsl` 命令がある場合に、System Generator へ Out0_* から出力されるとは限りません。



EDK で提供される FSL へアクセスする関数では、FSL インスタンスではなく、FSL バス接続の方を参照しています。上の図は、System Generator の MicroBlaze ブロックを示しています。内側のボックスが手動でコンフィギュレーションされた MicroBlaze を、外側のボックスが System Generator ブロックで表示される MicroBlaze プロセッサを示しています。この場合、In0 はバス接続 SFSL2 に、In1 は SFSL3 に接続されています。

上の図では、In0 からデータにアクセスするために記述されたソフトウェアコードを使用すると、SFSL2 にアクセスされます。In0 からのノンブロッキングデータリードは、次のように記述されます。

```
int val;
microblaze_nbread_datafsl(val, 2);
```

同様に、Out0 へのノンブロッキングデータライトは、次のように記述されます。

```
microblaze_nbwrite_datafsl(val, 0);
```

シミュレーション中の FSL の読み出し/書き込み

通常は、EDK にエクスポートした場合、バス接続をワイヤにまとめる必要があります。System Generator からはシミュレーション中にコンフィギュレーション済みの MicroBlaze が提供されるので、In0 ~ In7 は SFSL0 ~ SFSL7 に、Out0 ~ Out7 は MFSL0 ~ MFSL7 に接続されます。

FSL の id 0 からの読み出しは In0 に、FSL の id 0 への書き出しは Out0 に接続されます。

FSL の読み出しおよび書き込みエラー

MicroBlaze のマニュアルでは、バスがビッグ エンディアン命名規則を使用して表記されています。次の説明はそれに合わせてビッグ エンディアン命名規則を使用しているので、ビット 0 が MSB に相当します。

MicroBlaze Status Register (MSR) は、MicroBlaze にステータス状況を格納し、エラーが発生したかどうかを確認するために使用されます。

MSR の読み出しによるエラー確認

FSL の読み出しおよび書き込みでエラーが発生すると、そのエラーが **MSR (MicroBlaze Status Register)** に返されます。MSR のエラーは、アセンブリ命令の **mfs** を使用すると読み出すことができます。MSR を読み出すには、次のマクロを使用してください。

```
#define readmsr(val, dep) asm("mfs %0,rmsr" : "=d" (##val##) : "d" (dep))
```

C プログラムの一番上でこのマクロを定義してください。具体的には次のように使用します。

```
int val, mymsr;  
microblaze_nbread_cntlfs1(val, 0);  
readmsr(mymsr, val);
```

このマクロにより、**val** レジスタの書き込みと **MSR** の読み出しの間に依存関係が築かれます。これは、特にループ内で使用します。依存関係がないと、コンパイラはソフトウェア コードの最適化を実行する際に **mfs** 命令をループの外部に移動してしまいます。

FSL 読み出しエラー

FSL からの読み出しでエラーが発生する原因は、「**data invalid**」(データが無効) か、「**FSL error**」(FSL エラー) にあります。**data invalid** の場合は、FSL にデータがないので、ブロッキングまたはノンブロッキング データ リードが実行できません。**FSL error** は、データ値を読み出すのに、ブロッキングまたはノンブロッキング コントロール リードが使用された場合に、制御フラグが 1 に設定されていないと発生します。また、データ値を読み出すのに、ブロッキングまたはノンブロッキング データ リードが使用された場合に、制御フラグが 1 に設定されていなくても発生します。

data invalid エラーが発生すると、MSR はキャリー フラグを **High** に設定します。このキャリー フラグのマスクは **0X4** で、ビット 29 に対応します。キャリー フラグは、MSR の **MSB (ビット 0)** にも複製されます。

FSL error が発生すると、MSR は **FSL** フラグを **High** に設定します。この **FSL** フラグのマスクは **0X10** で、ビット 27 に対応します。

FSL 書き込みエラー

書き込まれる **FSL** がフルの場合、FSL への書き込みでエラーが発生します。フルの **FSL** に書き込もうとすると、ブロッキングおよびノンブロッキング ライトの両方から **data invalid** エラーが返されます。

data invalid エラーが発生すると、MSR はキャリー フラグを **High** に設定します。このキャリー フラグのマスクは **0X4** で、ビット 29 に対応します。キャリー フラグは、MSR の **MSB (ビット 0)** にも複製されます。

既知の問題

- 各デザインで使用できる **MicroBlaze** プロセッサは 1 つだけです。ただし、複数の **MicroBlaze** ブロックを EDK 側から作成された **FSL** インターフェイスに接続することはできます。
- MicroBlaze** は EDK にエクスポートするとき、最上位レベルにインスタンスシートする必要があります。ハードウェア協調シミュレーション モデルを使用する場合は、FSL の読み出しおよび書き込みでエラーが発生しないかどうかを確認してください。シミュレーションが開始されると、**MicroBlaze** プログラムがハードウェアで実行され、モデルが実行される前に **FSL** から読み出しまたは書き込みが行われることがあります。詳細は、「[チュートリアル: MicroBlaze プロセッサ システムの設計とシミュレーション](#)」および「[EDK Export Tool](#)」を参照してください。
- このブロックでは、Verilog ネットリストはサポートされません。

MicroBlaze プロセッサのオンライン マニュアル

MicroBlaze の詳細は、次のサイトを参照してください。

http://japan.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=micro_blaze

関連項目

[MicroBlaze プロセッサ ペリフェラルの設計とエクスポート](#)

[チュートリアル : MicroBlaze プロセッサ システムの設計とシミュレーション](#)

[EDK Export Tool](#)

ModelSim

このブロックは、[Xilinx Blockset] の [Tools] および [Index] ライブラリにリストされています。



System Generator の **Black Box** ブロックを使用すると、既存の HDL ファイルをモデルに組み込むことができます。モデルがシミュレーションされると、協調シミュレーションを使用してブラック ボックスがシミュレーションされるようにできます。この ModelSim HDL 協調シミュレーションブロックでは、1 つまたは複数のブラック ボックスの協調シミュレーションがコンフィギュレーションおよび制御できます。

シミュレーション中は、各 ModelSim ブロックで ModelSim のコピーが作成され、ModelSim ライセンスが 1 つずつ使用されます。ライセンスが少ない場合は、複数のブラック ボックスで同じブロックを共有できます。

ModelSim ブロックでは、次が実行されます。

- ブラック ボックス HDL が ModelSim 内でシミュレーションされるようにするために必要な VHDL および Verilog を追加します。
- Simulink のシミュレーションが開始されると、ModelSim セッションを開始します。
- Simulink と ModelSim 間の通信を媒介します。
- ブラック ボックス HDL がコンパイルされたときにエラーがあれば、それをレポートします。
- シミュレーションが終了すると、ModelSim を終了します。

メモ：ModelSim ブロックでサポートされる基数タイプは、ModelSim ツールの記号タイプのみです。記号タイプの基数の場合、ModelSim ブロックには列挙型の実際の値が表示され、オブジェクトの値を該当するその他の基数タイプに変換します。記号タイプの基数については、ModelSim のマニュアルを参照してください。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

[Run co-simulation in directory] : ModelSim がここで指定したディレクトリで開始されます。ディレクトリが存在しない場合は作成されます。協調シミュレーション用に System Generator で生成される補助ファイルも含め、すべてのブラック ボックス ファイルがこのディレクトリにコピーされます。既存のファイルは自動的に上書きされます。ディレクトリは、相対パスまたは絶対パスのどちらでも指定できます。相対パスは、Simulink の MDL ファイルがあるディレクトリを基準に解釈されます。

[Open waveform viewer] : オンにすると、ModelSim 波形ウィンドウが自動的に開き、標準的な信号セットが表示されます。この信号には、すべてのブラック ボックスの入力と出力、および System Generator から提供されるクロック信号とクロック イネーブル信号の入力と出力がすべて含まれます。信号表示は、補助的な Tcl スクリプトでカスタマイズできます。スクリプトを指定するには、[Advanced] タブで [Add custom scripts] をオンにし、[Script to run after "vsim"] にスクリプト名 (例 : myscript.do) を入力します。カスタマイズされた波形ビューアの例は、
<sysgen_tree>/examples/black_box/example5 ディレクトリに含まれています。この例については、「[ModelSim を使用したアドバンス ブラック ボックスの例](#)」を参照してください。

[Leave ModelSim open at end of simulation] : オンにすると、Simulink シミュレーションが終了した後も ModelSim セッションが開いたままになります。

[Skip compilation (use previous results)] : オンにすると、HDL 協調シミュレーション用の ModelSim ブロックを使用するブラック ボックスすべての ModelSim コンパイルが実行されなくなります。このオプションは、ModelSim が実行されるディレクトリの下に ModelSim の作業ディレクトリがあり、作業ディレクトリにすべてのブラック ボックス HDL の最新の ModelSim コンパイル結果が含まれている場合に使用します。オンにすると、シミュレーションの開始に必要な時間を大幅に削減できますが、条件に合わないときにオンにすると、シミュレーションでエラーになるか、シミュレーションが実行されても間違った結果が出力されてしまいます。

[Advanced] タブ

[Advanced] タブからは、次のようなパラメータを設定できます。

[Include Verilog unisim library] : オンにすると、シミュレーション中、ModelSim に Verilog の UniSim ライブラリを含めるようにできます。ただし、Verilog の UniSim ライブラリは、ModelSim で UNISIMS_VER にマップしておく必要があります。また、このオプションをオンにすると、glbl.v モジュールがコンパイルされ、シミュレーション中に起動されます。

[Add custom scripts] : script は、ModelSim で実行される Tcl マクロ ファイル (DO ファイル) のことです。オンにすると、[Script to run before starting compilation]、[script to run in place of "vsim"]、[Script to run after "vsim"] フィールドが設定できるようになります。[Add custom scripts] をオンにしておかないと、これらのフィールドで名前を付けた DO ファイルは実行されません。

[Script to run before starting compilation] : ブラック ボックス HDL ファイルをコンパイルする前に ModelSim で実行される Tcl マクロ ファイル (DO ファイル) の名前を入力します。

メモ : ModelSim マクロ ファイル (DO ファイル) の記述方法は、ModelSim のユーザー ガイドの「Tcl and macros (DO files)」の章を参照してください。

[Script to run in place of "vsim"] : ModelSim では、Tcl (Tool Command Language) をスクリプト言語として使用して、ツールを制御および拡張します。このフィールドには、ModelSim の Tcl マクロ ファイル (DO ファイル) を入力します。このファイルが、System Generator で ModelSim のシミュレーションを開始する命令があると、ModelSim の do コマンドで実行されます。マクロ ファイルの実行が開始されてからシミュレーションを開始する場合は、このマクロ ファイルに vsim コマンドを含める必要があります。

通常このパラメータが空白の場合、または [Add custom scripts] がオフの場合、System Generator は ModelSim にデフォルトのコマンド vsim \$stoplevel -title {System Generator Co-Simulation (from block \$blockname)} を実行するように命令します。この場合、\$stoplevel はシミュレーションの最上位レベルのエンティティ名 (例 : work.my_model_mti_block)、\$blockname は現在の協調シミュレーションに関連する Simulink モデルの中の ModelSim ブロック名を示しています。問題を回避するために、ブロック名に改行などの文字は使用しないでください。

[Add custom scripts] をオンにして、このパラメータを指定すると、System Generator が do \$* \$stoplevel \$blockname を実行するように ModelSim に命令をします。この場合、\$stoplevel と \$blockname は上記の説明と同じで、\$* はこのフィールドに入力された文字になります。たとえば、このフィールドに foo.do という文字を入力すると、ModelSim は foo.do を実行します。次に、このマクロ ファイルで \$stoplevel に \$1、\$blockname に \$2 を適用すると、マクロ ファイル foo.do の中の vsim \$1 コマンドは、\$stoplevel で vsim を実行します。

[Script to run after "vsim"] : このフィールドには、ブラック ボックスのすべての HDL が問題なくコンパイルされ、ModelSim シミュレーションが問題なく終了した後に、ModelSim で実行される Tcl マクロ ファイル (DO ファイル) を入力します。[Basic] タブで [Open waveform viewer] をオン

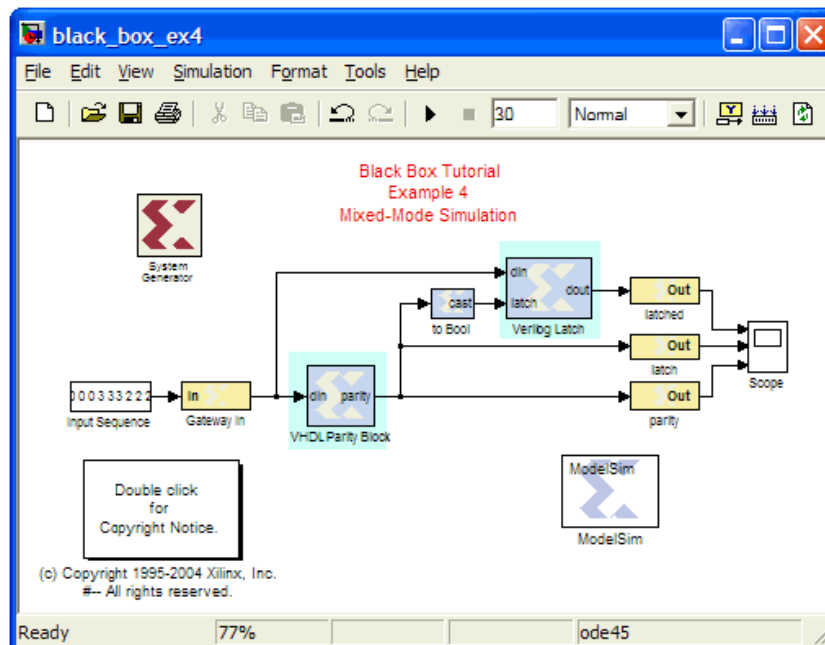
にしていると、このスクリプトを実行する前に、波形ビューアを開いてカスタマイズするすべてのコマンドが **System Generator** で実行されます。これで、信号をデフォルトのビューアに追加するか、完全なカスタム ビューアを作成して、波形ビューアをカスタマイズできるようになります。ブラック ボックスのチュートリアルには、波形ビューアのカスタマイズ例が含まれています。

カスタム スクリプトでは相対パスを使用すると便利ことが多くあります。相対パスは、モデルの MDL ファイルがあるディレクトリを基準に解釈されます。**[Run co-simulation in directory]** フィールドの相対パスも、モデルの MDL ファイルがあるディレクトリを基準に解釈されます。このため、たとえば **[Run co-simulation in directory]** で ModelSim が実行される ./modelsim ディレクトリが指定されている場合に、スクリプトを定義するフィールドに相対パス ../foo.do を指定すると、MDL を含むディレクトリの foo.do という名前のファイルが使用されます。

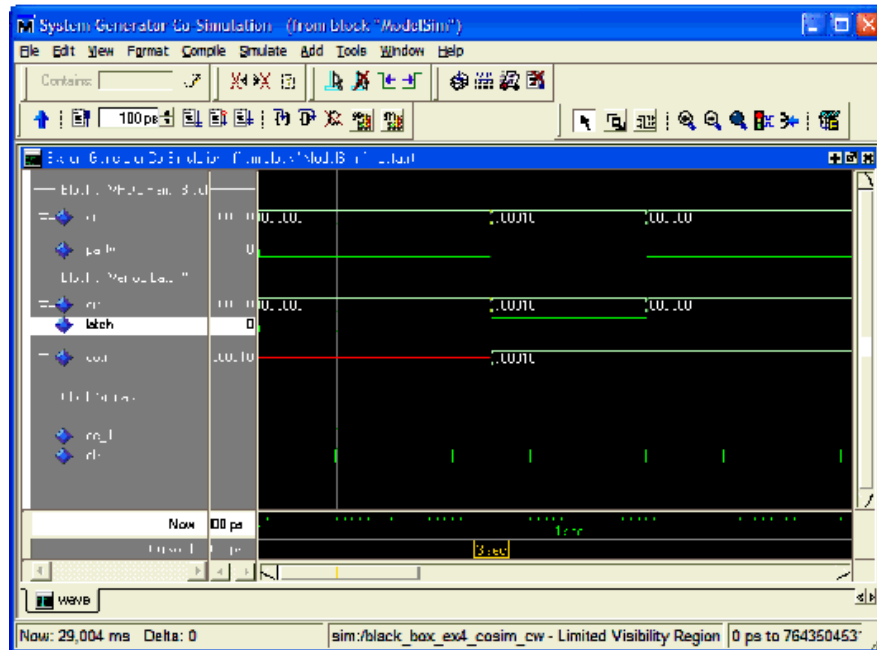
タイム スケール

ModelSim と Simulink のタイム スケールは同じです。Simulink シミュレーション時間の 1 秒は、ModelSim シミュレーション時間の 1 秒と同じ長さです。これにより、2 つの設定で発生するイベントの時間が比較しやすくなります。通常は、Simulink のタイム スケールが大きくても便利です。これは、タイム スケールが大きいと、System Generator で HDL モデルのタイミング特性に関する問題なしに、イベントをスケジュールできるからです。この場合、協調シミュレーション モデルで System Generator のイベント スケジュールをユーザーが注意する必要がありません。

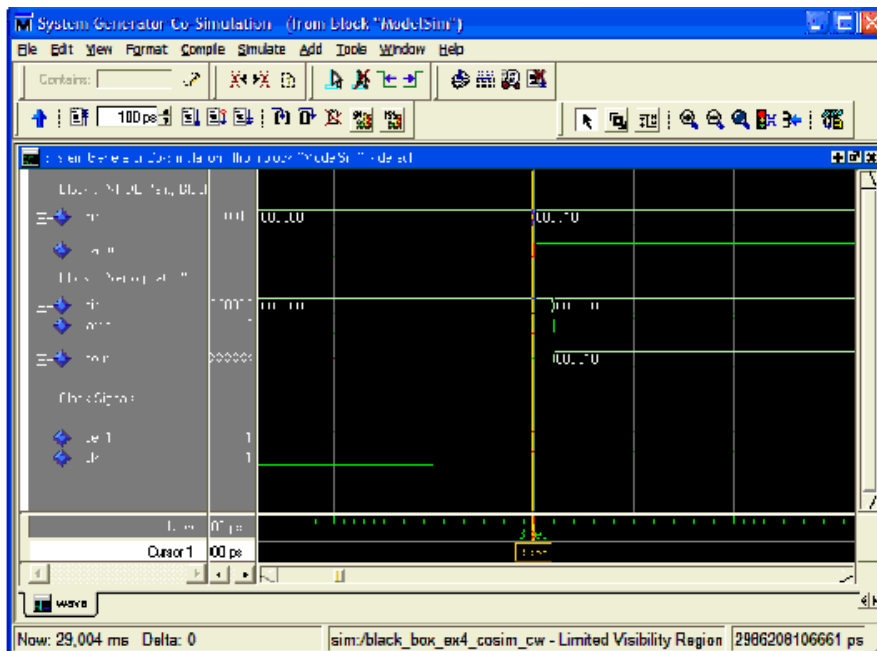
この例は、System Generator のディレクトリ、<sysgen_tree>/example/black_box/example4 に含まれています。この例については、「Verilog モジュールのインポート」も参照してください。



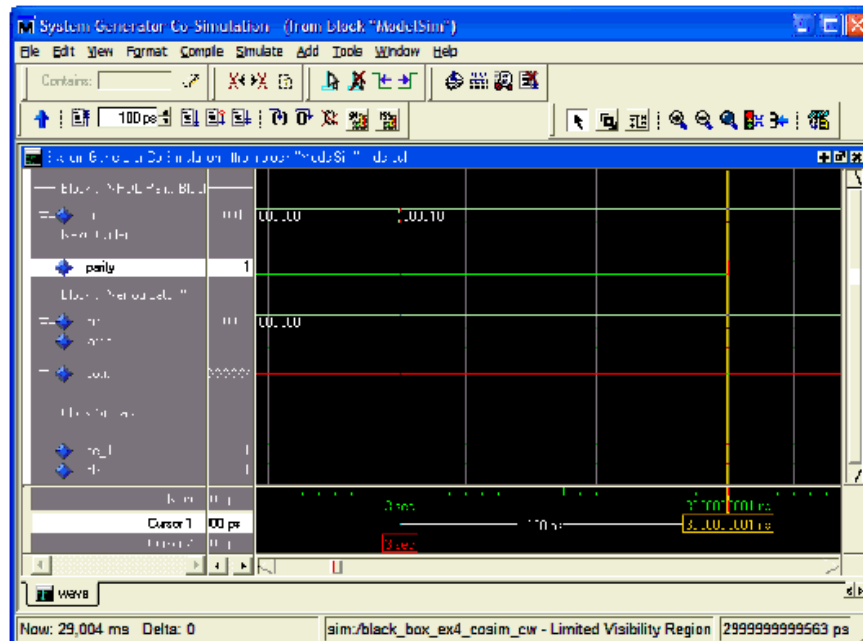
上記のモデルが実行されると、ModelSim で次の波形が表示されます。



このタイミング スケールでは (上の図では、6 秒間隔)、3 秒目のクロックの立ち上がりエッジとそれに対応する 2 つのブラック ボックスそれぞれからの送信データが同時に表れています。これは、**Simulink** のシミュレーションと同じです。ただし、モデルを見ると、明らかに最初のブラック ボックスの出力が 2 つ目のブラック ボックスに入力されています。どちらのブラック ボックスも組み合わせフィードスルーを含むので、入力の変更がすぐに出力の変更に反映されます。3 秒目のイベントを拡大すると、**System Generator** がこの依存性をどのように回避したのかがわかります。表示される間隔は、20 ms に短縮されています。 .



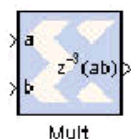
上の図は、System Generator がクロックの立ち上がりエッジをシフトしたところを示しています。これで、入力値が Simulink から集められ、最初のブラック ボックスに渡される前にクロックが立ち上がるようになっていきます。この後、値が最初のブラック ボックスに渡され、その結果がわずかに遅れた時間で 2 つ目のブラック ボックスに渡されます。さらに少し拡大すると、最初のブラック ボックスの HDL モデルに伝搬遅延が含まれているのがわかります。この遅延は、System Generator で大きなタイム スケールを使用することによって、効率的に取り除かれています。最初のブラック ボックスからの実際の遅延 (調度 1 ns) は、次の図に表示されています。



ブラック ボックス コンポーネントからデータを伝搬する場合、System Generator ではシステム クロック周期の 1/1000 が 1us に分配され、それがシステム クロック周期の 1/100 の 5ns に縮小されて分配され、そのしきい値よりも低くなると、デルタ遅延ステップが使用されます (ModelSim に対して run 0 ns コマンドが実行されます)。HDL に転送遅延などのタイミング情報が含まれ、Simulink のシステム周期の設定が低すぎる場合は、正しいシミュレーション結果が出ません。上記のモデルの場合、Simulink のシステム周期設定が 5e-7 よりも削減されると、エラーになります。このポイントは、System Generator がデータ転送用にブラック ボックスのデルタ遅延ステップを実行する箇所です。

Mult

このブロックは、[Xilinx Blockset] の [Math] および [Index] ライブラリにリストされています。



ザイリンクスの **Mult** ブロックは、乗算器をインプリメントします。このブロックでは、2つの入力ポートのデータの積が出力ポートに出力されます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Latency] : ブロックの出力を遅延させるサンプル周期が定義できるようになっています。
- 乗算器のユーザー データ型のサチュレートおよび丸め設定 :
乗算器のユーザーのデータ型にサチュレートまたは丸めが選択されている場合、サチュレート/丸めロジックがまずパイプライン接続されてから、別のレジスタがコアに追加されるように、レイテンシが分散されます。たとえば、レイテンシを3にし、サチュレート/丸めを選択した場合、最初のレジスタはサチュレートまたは丸めロジックの後に配置され、コアへのパイプライン用にレジスタが2つ追加されます。レジスタは最適なパイプラインになるまでコアに追加され、サチュレート/丸めロジックの後にさらにレジスタが配置されますが、選択したデータ型で追加のサチュレート/丸めロジックが必要ない場合は、すべてのレジスタがコアのパイプラインに使用されます。

[Implementation] タブ

[Implementation] タブからは、次のようなパラメータを設定できます。

- [Use behavioral HDL (otherwise use core)] ビヘイビアレベルのHDL記述が使用されます。これで、最適化をパフォーマンス重視にするか、エリア重視にするかをダウンストリームのロジック合成ツールで柔軟に決定できます。

コアのパラメータ

- [Optimize for Speed|Area] : ブロックを [Speed] または [Area] のどちらを重要視して最適化するか指定できます。
- [Use embedded multipliers] : オンにすると、可能な場合に XtremeDSP スライス (DSP48 タイプのエンベデッド乗算器) がターゲット デバイスで使用されます。
- [Test for optimum pipelining] : [Basic] タブで指定した [Latency] が少なくとも最適なパイプライン長と等しいかどうかチェックされます。レイテンシの値がこのテストをパスした場合、コアはスピードを重視して最適化されます。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

Mult ブロックでは、[Implement from behavioral HDL description (otherwise use core)] がオンになっている場合を除いて、次のザイリンクス Multiplier Generator コアが使用されます。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6 -1L
Mult	Multiplier	V11.2	•	•	•	•	•	•	•	•	•	•

Multiple Subsystem Generator

このブロックは、[Xilinx Blockset] の [Shared Memory] および [Index] ライブラリにリストされています。



ザイリンクスの **Multiple Subsystem Generator** ブロックは、複数の **System Generator** デザインを複数のクロックドメインを使用する 1 つの最上位レベルの HDL コンポーネントに接続します。この最上位レベルのコンポーネントには、各 **System Generator** デザインに接続されたロジックと、デザインが互いに通信できるようにするロジックが含まれます。

ソフトウェアでは、この通信は共有メモリと共有メモリ関連のブロック (**Shared Memory**、**To/From FIFO**、**To/From Register** ブロックなど) を使用して処理されます。ハードウェアでは、デザインが共有メモリに対応するハードウェア インプリメンテーション (デュアルポートメモリ、非同期 FIFO、レジスタなど) に関連付けられ、システムをパーティションしたり、複数のクロックドメインでインプリメントしたりできます。

メモ : Multiple Subsystem Generator ブロックでは、EDK Processor ブロックを含むデザインがサポートされません。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

Multiple Subsystem Generator ブロックのパラメータには、次のようなものがあります。

- **[Part]** : 使用する FPGA デバイスを指定します。
- **[Target directory]** : コンパイル結果を記述するディレクトリを指定します。**System Generator** および **FPGA インプリメンテーション ツール** では多数のファイルが生成されるので、個別のディレクトリ (**Simulink** モデル ファイルが含まれるディレクトリとは別のディレクトリ) を指定することをお勧めします。
- **[Synthesis tool]** : デザインの合成に使用するツールを指定します。**Synplicity** 社の **Synplify Pro** または **Synplify**、およびザイリンクスの **XST** を選択できます。
- **[Hardware description language]** : 生成する HDL 言語のタイプ (**Verilog** または **VHDL**) を選択します。

デザインの生成

Multiple Subsystem Generator ブロックでは、パラメータ ダイアログ ボックスで **[Generate]** ボタンをクリックすると、次が実行されます。

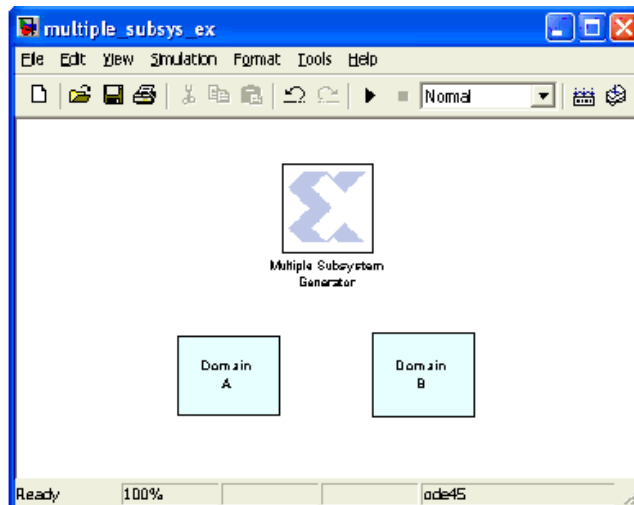
1. 生成して接続する **System Generator** デザインを決定します。
2. **System Generator** デザインを適切な設定でコンフィギュレーションし、デザインを個別に生成します。
3. 共有メモリ ブロックのハードウェア インプリメンテーション (コアのネットリストなど) を出力します。
4. 対応する共有メモリのハードウェア インプリメンテーションと接続された **System Generator** デザインを含む最上位レベルの HDL ファイルを生成します。

Multiple Subsystem Generator ブロックでは、ブロックと同じ階層レベルにある **System Generator** ブロックを含むサブシステムが検索され、どのサブシステムをインプリメントして接続するかが決

定されます。Multiple Subsystem Generator ブロックを Simulink デザインに含める場合は、次の規則に従う必要があります。

- System Generator ブロックは、通常 Multiple Subsystem Generator ブロックと同じ階層レベルには含めません。
- サブシステム内の少なくとも 2 つのマスタ System Generator ブロックを Multiple Subsystem Generator ブロックと同じ階層レベルに含める必要があります。
- 1 つの階層レベルに含めることができる Multiple Subsystem Generator ブロックは 1 つだけです。

たとえば、次のようなブロック図があるとします。この図には、サブシステムが 2 つ含まれています。各サブシステムには、System Generator ブロックが 1 つと System Generator ロジックが複数含まれます。この図に表示されているサブシステムは 2 つだけですが、Multiple Subsystem Generator ブロックにはそれ以上の数のサブシステムを対応させることができます。Multiple Subsystem Generator ブロックは、この 2 つのサブシステムと同じ階層レベルに含まれます。Multiple Subsystem Generator ブロックを使用してデザイン全体を生成するように選択すると、サブシステムが生成されて接続されます。



1 つのマスタの System Generator ブロックを含むサブシステムは、Multiple Subsystem Generator ブロックのダイアログ ボックスで [Generate] ボタンをクリックすると、NGC コンパイル ターゲットを使用してインプリメントされます。NGC コンパイル ターゲットを使用すると、結果の HDL ネットリスト、コア、制約が 1 つのネットリスト ファイルとして作成できるという利点があります。デザインをまとめる HDL コンポーネントでは、System Generator デザインがブラック ボックスとしてインスタンス化されます。NGC ファイルにはブラック ボックスのインプリメンテーションが含まれます。上の例の場合、それぞれが各サブシステムに対応した 3 つの NGC ファイルが生成されます。

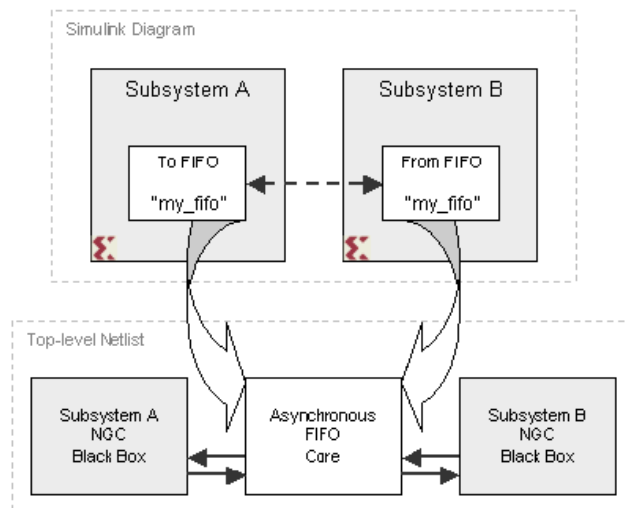
デザインの生成前に、Multiple Subsystem Generator ブロックのダイアログ ボックスで [Part]、[Synthesis tool]、[Hardware description language] を設定しておきます。これらの設定は、マスタの System Generator ブロックの設定よりも優先されます。元の System Generator 設定は、生成が完了すると回復されます。

Multiple Subsystem Generator ブロックを使用して接続されたサブシステム同士の通信には、To/From FIFO ブロックまたは To/From Register ブロックのいずれかの共有メモリ ブロックのペアが使用されます。このブロック ペアは、1 つのブロック (例: To FIFO ブロック) が 1 つのサブシステムに含まれる場合、ペアのもう 1 つのブロック (From FIFO ブロック) が別のサブシステムに含まれるようにパーティションする必要があります。

デザイン全体がハードウェアに変換されると、FIFO の半分が 2 つ、対応するサブシステムから読み出されます。この後、前に共有メモリ ポート (data in, data out など) に接続された System Generator ロジックがそのデザインの最上位レベルのポートに接続されます。つまり、1 つのサブシステムの HDL コンポーネントには、共有メモリの半分用のポートが含まれます。残りの半分には、共有メモリの反対側用のポートが含まれます。共有メモリのハードウェア インプリメンテーションが作成され、最上位レベルの共有メモリ ポートに接続されます。

メモ： Multiple Subsystem Generator ブロックでは、現在のところ、複数の共有メモリ ブロックが同じサブシステム内の同じ共有メモリ オブジェクトを参照できないようになっています。たとえば、To FIFO ブロックはほかのサブシステムにある 2 つの From FIFO ブロックとの通信には使用できません。

たとえば、A と B の 2 つのサブシステムがあるとします。サブシステム A には To FIFO ブロックが、サブシステム B には From FIFO ブロックが含まれます。FIFO の反対半分では、my_fifo という同じ共有メモリが指定されます。Multiple Subsystem Generator ブロックを使用してネットリストが作成されると、To FIFO ブロックと From FIFO ブロックが対応するサブシステムからそれぞれ削除され、1 つのコア インプリメンテーション (例：非同期 FIFO コアなど) に統合されます。次の図は、このプロセスを示しています。



次の表は、共有メモリおよび共有メモリ関連のブロックをインプリメントに使用されるコアまたは HDL コンポーネントのインプリメンテーションを示しています。

To ブロック	From ブロック	ハードウェア インプリメンテーション
共有メモリ	共有メモリ	Dual Port Block Memory 6.1
To FIFO	To FIFO	Fifo Generator 2.1
To Register	To Register	synth_reg_w_init.(vhd,v)

メモ： 共有メモリ ブロックは、サブシステム間の通信手段としてのみ使用されます。サブシステム間の通信に System Generator 信号は使用しないでください。これは、これらの信号が Multiple Subsystem Generator ブロックで作成される最上位レベルの HDL コンポーネントの最上位レベルのポートに最終的に変換されるからです。

Multiple Subsystem Generator ブロックを使用した System Generator デザインのゲートウェイポートはすべて最上位レベルの HDL コンポーネントのポート インターフェイスに含まれます。ま

た、クロック ポートとクロック イネーブル ポートは、それぞれ各 **System Generator** サブシステムのポート インターフェイスに含まれます。クロック ポート名とクロック イネーブル名は、ポート名にデザイン名が付いた名前では区別されます。たとえば、**Domain A** というサブシステムに **inport_a** という入力ポート 1 つと **outport_a** という出力ポート 1 つが含まれるとします。また、**Domain B** というサブシステムには、**inport_b** という入力ポート 1 つと **outport_b** という出力ポート 1 つが含まれるとします。結果の最上位レベルのエンティティの **VHDL** ポート インターフェイスは、次のようになります。

```
entity multiple_subsys_ex is
port (
  domain_a_ce: in std_logic := '1';
  domain_a_clk: in std_logic;
  domain_b_ce: in std_logic := '1';
  domain_b_clk: in std_logic;
  inport_a: in std_logic_vector(17 downto 0);
  inport_b: in std_logic_vector(17 downto 0);
  outport_a: out std_logic_vector(17 downto 0);
  outport_b: out std_logic_vector(17 downto 0)
);
end multiple_subsys_ex;
```

複数クロックのサポート

Multiple Subsystem Generator ブロックを使用するサブシステムには、それぞれマスタの **System Generator** ブロックが 1 つ含まれるので、**Simulink** システム周期や **FPGA** クロック周期などのクロック情報をブロックごとに指定することができます。異なる **Simulink** システム周期を指定すると、各 **System Generator** デザインを別のレートでシミュレーションでき、非同期のクロック ドメインを使用するモデル システムを効率的に作成できます。

Multiple Subsystem Generator ブロックは、生成されるサブシステムごとに別のクロック ポートを作成します。このクロック ポートが **System Generator** デザインの対応するクロック ポートに配線されます。複数のクロックを使用するデザインのネットリストが生成されると（高位レベルのモデルから下位レベルの **HDL** 記述に変換されると）、共有メモリの半分 2 つが対応するサブシステムから上位の階層レベルに移動されます。この共有メモリ ペアの 2 つの半分がクロック ドメイン ブリッジ（デュアル ポート メモリなど）をインプリメントする 1 つの **HDL** コンポーネントに置換されます。この後、2 つのクロック ドメインからのクロックがブリッジ コンポーネントの反対側に、必要なデータおよび制御信号と共に接続されます。

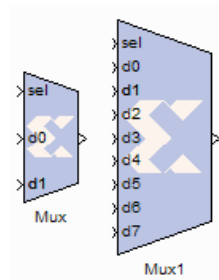
生成ファイル

Multiple Subsystem Generator ブロックでは、[Generate] ボタンをクリックすると、下位レベルのファイルが複数生成されます。これらのファイルは、ブロックのダイアログ ボックスで指定したターゲット ディレクトリに書き込まれます。このブロックで生成される主なファイルは、次のとおりです。

ファイル タイプ	説明
<design>.vhd (または .v)	System Generator デザインを含む最上位レベルの HDL コンポーネントがまとめられています。
.edn ファイル	Multiple Subsystem Generator ブロックは、HDL を書き込むだけでなく、CORE Generator から共有メモリのハードウェア インプリメンテーションを実行します。CORE Generator では、multiplier_virtex2_6_0_83438798287b830b.edn というような名前の EDIF ファイルが生成されます。
globals	デザインを記述するキーと値のペアが含まれます。このファイルは Perl ハッシュ テーブルとして構成されており、Perl eval 関数を使用することにより Perl スクリプトでキーと値を使用できます。
<design>.xcf (または .ncf)	タイミング制約およびポートのロケーション制約が含まれます。これらの制約は、ザイリンクスの合成ツール XST およびザイリンクス インプリメンテーション ツールで使用されます。XST 以外の合成ツールを指定した場合は、拡張子は .ncf となります。
hdlFiles	Multiple Subsystem Generator ブロックで生成される HDL ファイルのリストが含まれます。ファイルは、通常の HDL 依存順でリストされています。
<design>.npl	ザイリンクスのプロジェクト管理ツール Project Navigator で HDL および EDIF ファイルを処理するためのプロジェクト ファイルです。

Mux

このブロックは、[Xilinx Blockset] の [Basic Elements]、[Control Logic]、[Index] ライブラリにリストされています。



ザイリンクスの **Mux** ブロックは、マルチプレクサをインプリメントします。ブロックには、セレクト入力 (符号なし) が 1 つと、ユーザーがコンフィギュレーション可能なデータ バス入力 (0 ～ 1024 の範囲) がいくつか含まれます。

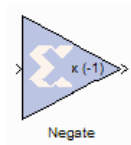
ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

このブロックで使用するパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

Negate

このブロックは、[Xilinx Blockset] の [Math] および [Index] ライブラリにリストされています。



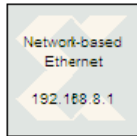
ザイリンクスの **Negate** ブロックでは、入力の論理否定 (2 の補数) が実行されます。このブロックは、ザイリンクス **LogiCORE** または合成可能な **VHDL** モジュールとしてインプリメントできます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

このブロックで使用するパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

Network-based Ethernet Co-Simulation



ザイリンクスの Network-based Ethernet Co-Simulation ブロックは、IPv4 ネットワーク インフラストラクチャのイーサネット接続を介して、ハードウェア協調シミュレーションを実行するインターフェイスを提供します。

インターフェイス、必要条件、セットアップ手順などの詳細は、「[ネットワーク ベースのイーサネット ハードウェア協調シミュレーション](#)」を参照してください。

このブロックには、さまざまなポート インターフェイスが使用されます。ネットワーク ベースのイーサネット ハードウェア協調シミュレーションのためにモデルがインプリメントされると、新しいライブラリが作成され、このライブラリに元のモデルのゲートウェイ名（サブシステムが最上位レベルでない場合はポート名）と同じポートの付いたカスタムの Network-based Ethernet Co-simulation ブロックが含まれます。このブロックは、Simulink のシミュレーション中に、FPGA ハードウェア プラットフォームとの通信に使用されます。ブロックの入力ポートに書き込まれたシミュレーション データはブロックによりハードウェアに渡されます。データがブロックの出力ポートから読み出されると、ブロックはハードウェアから適切な値を読み込んで、出力ポートに駆動します。これで、データが Simulink で解釈できるようになります。また、ブロックではプラットフォームの開始、コンフィギュレーション、ステップ、終了が自動的に実行されます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- **[Clock source]** : シングル ステップまたはフリーランニング クロック ソースを選択できます。シングル ステップ クロックを選択すると、1 度に 1 クロック サイクルずつボードをステップできます。各クロック サイクル ステップは、Simulink での一定の時間に相当します。このクロック ソースを使用すると、シミュレーション中の協調シミュレーション ハードウェアのビヘイビアのビット精度および周期精度が、そのサブシステムのシミュレーション ビヘイビアよりも正確になります。シングル ステップが不要で、フリーランニング クロックでボードを実行できる場合もあります。この場合、ボードは Simulink シミュレーションと非同期で動作します。
- **[Has combinational path]** : ハードウェア協調シミュレーション ブロックの出力ポートから同じブロックの入力ポートへの直接組み合わせフィードバック パス (同じブロックの出力ポートから入力ポートへの接続) が必要な場合もあります。出力ポートから入力ポートへの直接フィードバック パスが必要で、デザインに入力ポートから出力ポートへの組み合わせパスが含まれない場合は、このチェック ボックスをオフにすると、デザインでフィードバック パスを使用できるようになります。
- **[Bitstream name]** : ネットワーク ベースのイーサネット ハードウェア協調シミュレーション プラットフォーム用の協調シミュレーション FPGA コンフィギュレーション ファイルを指定します。新規の協調シミュレーション ブロックがコンパイル中に作成されると、このパラメータが自動的に設定され、正しいコンフィギュレーション ファイルが使用されます。このパラメータは、コンフィギュレーション ファイルのディレクトリを変更した場合にのみ修正します。

[Network] タブ

[Network] タブからは、次のようなパラメータを設定できます。

- **[FPGA IP address]** : ターゲット FPGA プラットフォームに関連付けられた IPv4 アドレスを指定します。IP アドレスは、10 進数をピリオドで区切った表記方法 (192.168.8.1 など) で指定します。IP アドレスのコンフィギュレーションについては、「[ハードウェア プラットフォームのインストール](#)」を参照してください。
- **[Timeout]** : コンフィギュレーション プロセスと協調シミュレーション プロセス中にパケット損失があった場合に、パケット再送信に使用されるタイムアウト値をミリ秒で指定します。通常はデフォルトの値で問題ありませんが、レイテンシが大きかったり、混線のためにネットワーク接続が遅い場合は、より大きな値を指定してみてください。
- **[Number of retries]** : コンフィギュレーション プロセスと協調シミュレーション プロセス中にパケット損失があった場合に、パケット再送信に使用されるリトライ回数を指定します。通常はデフォルトの値で問題ありませんが、ネットワーク接続でかなりの量のパケット損失がある場合は、より大きな値を指定してみてください。

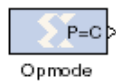
関連項目

[イーサネット ハードウェア協調シミュレーション](#)

[ネットワーク ベースのイーサネット ハードウェア協調シミュレーション](#)

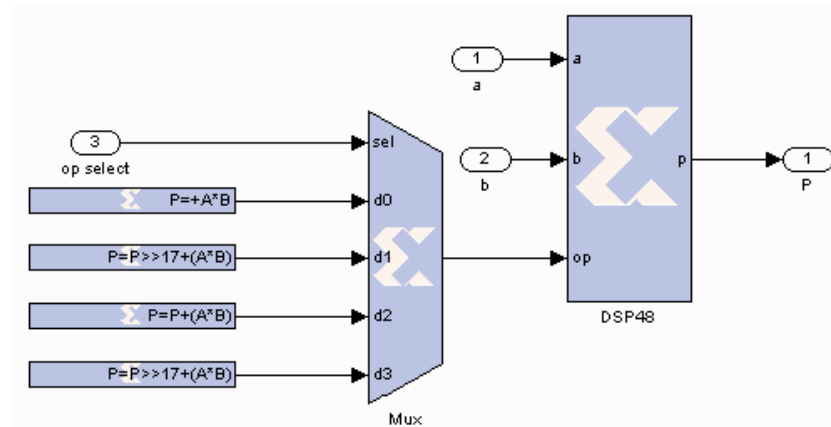
Opmode

このブロックは、[Xilinx Blockset] の [DSP] および [Index] ライブラリにリストされています。



ザイリンクスの Opmode ブロックでは、DSP48 または DSP48E 命令の定数が生成されます。この命令は、DSP48 の場合が 11 ビット値、DSP48E の場合が 15 ビット値になります。命令には、opmode、carry-in、carry-in select が含まれるほか、DSP48 か DSP48E かによって subtract または alumode ビットのいずれかが含まれます。

Opmode ブロックは、DSP48 または DSP48E の制御シーケンスを生成するのに使用すると便利です。次の図に例を示します。この例では、DSP48 ブロックに 4 つの命令を含むシーケンスが使用され、35X35 ビットの乗算器がインプリメントされています。Opmode ブロックは命令をマルチプレクサに伝え、マルチプレクサがシーケンスの各命令を選択しています。



ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Opmode] タブ

[Opmode] タブからは、次のようなパラメータを設定できます。

- [Device] : DSP48 または DSP48E デバイスのどちらの命令を生成するか指定します。
- [Operation] : ブロックで生成される命令を表示します。命令は、ブロック表面にも表示されます。
- [DSP48 Instruction] または [Custom Instruction] : DSP48 または DSP48E 命令を選択できます。カスタムを選択すると、z_mux +/- (yx_mux + carry) フォーマットの命令形式が可能なマスク パラメータが表示されます。
- [Z mux] : DSP48(E) の加算器に対する Z ソースを {'0', 'C', 'PCIN', 'P','C', 'PCIN>>17', 'P>>17'} のいずれかから指定します。
- [Operand] : DSP48 の加算器で加算を実行するか、減算を実行するか指定します。DSP48E の場合、演算は [Instruction] プルダウンから選択します。
- [YX muxes] : DSP48 の加算器に対する YX ソースを {'0','P', 'A:B', 'A*B', 'C', 'P+C', 'A:B+C'} のいずれかから指定します。[A:B] にすると、A が B と連結されて 1 つの値が出力され、加算器の入力として使用されます。

- [Carry input] : DSP48 の加算器に対する YX ソースを {'0', '1', 'CIN', '~SIGN(P or PCIN)', '~SIGN(A:B or A*B)', '~SIGND(A:B or A*B)'} のいずれかから指定します。[~SIGN (P or PCIN)] の場合は、キャリー ソースが [Z mux] の設定によって P か PCIN のいずれかになり、[~SIGN (P or PCIN)] の場合は、キャリー ソースが [Z mux] の設定によって P か PCIN のいずれかになり、[~SIGND (A*B or A:B)] 場合は、[~SIGN(A*B or A:B)] に遅延が追加されます。詳細は、「DSP48 の制御命令形式」を参照してください。

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

Opmode ブロックでは、ザイリンクス LogiCORE は使用されません。

DSP48 の制御命令形式

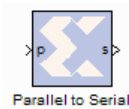
命令フィールド名	ロケーション	ニーモニック	説明
YX Mux	op[3:0]	0	0
		P	DSP48 出力レジスタ
		A:B	入力 A と B を連結 (A は MSB)
		A*B	入力 A と B を乗算
		C	DSP48 の入力 C
		P+C	DSP48 の入力 C + P
		A:B+C	入力 A と B の連結 + C レジスタ
Z Mux	op[6:4]	0	0
		PCIN	PCOUT からの DSP48 カスケード入力
		P	DSP48 出力レジスタ
		C	DSP48 の入力 C
		PCIN>>17	17 個ダウンシフトされたカスケード入力
		P>>17	17 個ダウンシフトされた DSP48 出力レジスタ
Operand	op[7]	+	加算
		-	減算
Carry In	op[8]	0 または 1	キャリーインを 0 か 1 に設定
		CIN	ソースに cin を選択
		'~SIGN(P or PCIN)	P または PCIN の対称丸め
		'~SIGN(A:B or A*B)	A:B または A*B を対称丸め
		'~SIGND(A:B or A*B)	A:B または A*B を対称丸め (遅延付き)

DSP48E の制御命令形式

命令フィールド名	ロケーション	ニーモニック	説明
YX Mux	op[3:0]	0	0
		P	DSP48 出力レジスタ
		A:B	入力 A と B を連結 (A は MSB)
		A*B	入力 A と B を乗算
		C	DSP48 の入力 C
		P+C	DSP48 の入力 C + P
		A:B+C	入力 A と B の連結 + C レジスタ
Z Mux	op[6:4]	0	0
		PCIN	PCOUT からの DSP48 カスケード入力
		P	DSP48 出力レジスタ
		C	DSP48 の入力 C
		PCIN>>17	17 個ダウンシフトされたカスケード入力
		P>>17	17 個ダウンシフトされた DSP48 出力レジスタ
Alumode	op[10:7]	X+Z	加算
		Z-X	減算
Carry InSelect op[14:12]		0 または 1	キャリーインを 0 か 1 に設定
		CIN	ソースに cin を選択します。Opmode ブロックに CIN ポートが追加され、その値がビット ロケーション 11 のニーモニックに挿入されます。

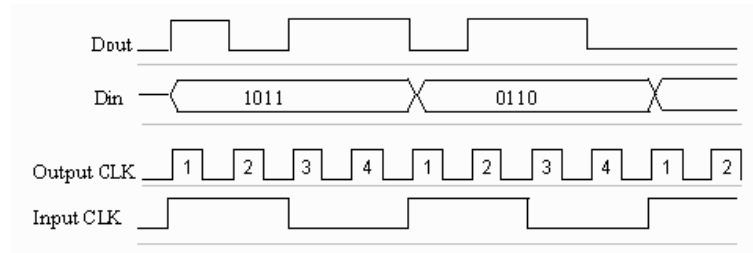
Parallel to Serial

このブロックは、[Xilinx Blockset] の [Basic Elements]、[Data Types]、[Index] ライブラリにリストされています。



Parallel to Serial ブロックは、入力ワードを読み込み、それを分割してマルチプレクサで N 倍多重化した出力ワードにします。 N は、入力ビットから出力ビットの数の比率です。出力順は、LSB からか MSB からかのどちらかになります。

次の波形図は、このブロックの動作を示しています。



この例では、入力幅 4、出力ワード 1 の場合に、最上位ワードから出力されるように設定されています。

ブロック インターフェイス

Parallel to Serial ブロックには、入力が 1 つと出力が 1 つあります。入力ポートは、どのサイズにでもできます。出力ポート サイズは、ブロックのパラメータ ダイアログ ボックスで指定できます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Output order] : 最上位ワードまたは最下位ワードのどちらを最初に出力するか指定します。
- [Type] : 符号付きか符号なしかを指定します。
- [Number of bits] : 出力幅を指定します。入力ビット数を均等に除算できる値にする必要があります。
- [Binary point] : 2 進小数点の位置を指定します。

このブロックの最小レイテンシは 1 です。


このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

Pause Simulation

このブロックは、[Xilinx Blockset] の [Tools] および [Index] ライブラリにリストされています。



ザイリンクスの **Pause Simulation** ブロックでは、入力が 0 以外の場合にシミュレーションを一時停止します。このブロックには、ザイリンクスの信号タイプでも入力として使用できます。

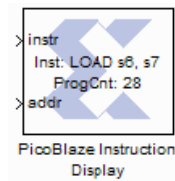
シミュレーションが一時停止した場合は、ツールバーの [Start] ボタン () をクリックすると再開されます。

ブロック パラメータ

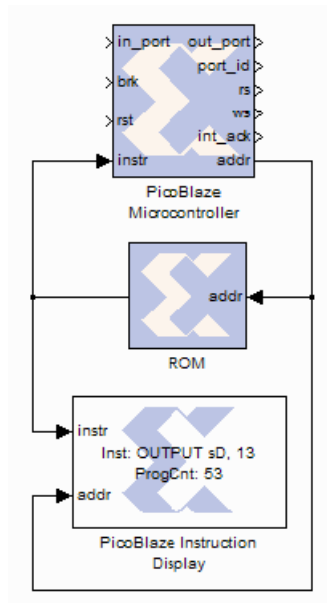
このブロックにはパラメータがありません。

PicoBlaze Instruction Display

このブロックは、[Xilinx Blockset] の [Tools] および [Index] ライブラリにリストされています。



PicoBlaze Instruction Display ブロックは、エンコードされた 18 ビットの PicoBlaze 命令と 10 ビットのアドレスを読み込み、デコードされた命令とプログラム カウンタをブロックのアイコンに表示します。このブロックを使用すると、PicoBlaze デザインをデバッグするのに便利です。また、このブロックを [Single-Step Simulation](#) ブロックと共に使用して、各命令を実行することもできます。



ブロック インターフェイス

PicoBlaze Instruction Display ブロックには、入力ポートが 2 つ含まれます。instr ポートには、18 ビットのエンコード命令が、addr ポートにはプログラム カウンタである 10 ビットのアドレス値が入力されます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

このブロックでは、次のパラメータが設定できます。

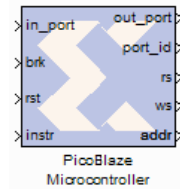
- [Version] : [PicoBlaze 2] または [PicoBlaze 3] を指定します。
- [Disable Display] : オンにすると、表示がアップデートされなくなるので、デバッグ モード以外の場合にシミュレーション速度を上げることができます。

ザイリンクス LogiCORE

PicoBlaze Instruction Display ブロックでは、ザイリンクス LogiCORE が使用されません。

PicoBlaze Microcontroller

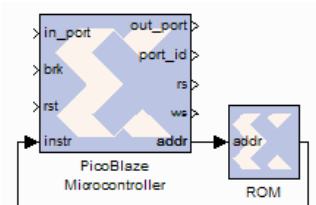
このブロックは、[Xilinx Blockset] の [Control Logic] および [Index] ライブラリにリストされています。



PicoBlaze Microcontroller ブロックは、PicoBlaze マクロを使用して 8 ビットのエンベデッド マイクロコントローラをインプリメントします。

このブロックでは、PicoBlaze 2 と PicoBlaze 3 の 2 バージョンがサポートされます。PicoBlaze 2 では Virtex-II が、PicoBlaze 3 では Spartan-3、Virtex-4 がサポートされます。PicoBlaze 2 マクロでは、42 の命令と 32 個の 8 ビット 汎用レジスタ、256 個の直接または間接的なアドレス ポートと 1 つのマスク 可能割り 込みが提供されます。PicoBlaze 3 では、53 の命令、16 個の 8 ビット 汎用レジスタ、256 個の直接または間接的なアドレス ポートと 1 つのマスク 可能割り 込みに加え、STORE および FETCH 命令を使用してアクセス可能な 64 バイトの内部スクラッチ パッド メモリも提供されます。PicoBlaze2 エンベデッド コントローラとその命令セットに関する詳細は、アプリケーション ノート XAPP627 (<http://japan.xilinx.com/bvdocs/appnotes/xapp627.pdf>) を参照してください。

通常は、1024 X 8 ビットのブロック ROM 1 つにプログラムを保存できます。このブロックと ROM は次の図のように接続されます。



ブロック インターフェイス

どちらのバージョンのブロックにも入力ポートが 4 つ含まれます。8 ビットのデータ ポート in_port が INPUT 命令中に読み出しを行います。この値は 32 個のいずれのレジスタにも転送できます。プログラムは、brk ポートを 1 に設定すると中断されます。プロセッサは、rst を 1 に設定するとリセットされます。リセットをすると、レジスタの内容が一掃され、プロセッサがアドレス 0 で命令を実行し始めるようになります。8 ビットの入力ポート instr には PicoBlaze 命令が入力されます。

PicoBlaze 2 ブロックには、出力ポートが 5 つ含まれます。PicoBlaze 3 ブロックには、出力ポートが 6 つ含まれます。8 ビットの出力ポート out_port が OUTPUT 命令中に書き込みを行います。port_id 出力は、読み出し/書き込み中に読み出される (書き込まれる) 値のロケーションを識別します。出力ポート rs (read strobe) および ws (write strobe) は、読み出し (INPUT) または書き込み (OUTPUT) のどちらが実行されるかを示します。addr はプロセッサで実行される次の命令のアドレスを示します。プロセッサは、内部にプログラムを保存できません。出力ポート addr は、次に実行される命令のロケーションを指定します。ack ポート (PicoBlaze 3 のみ) は、いつ割り込みサービスルーチンを開始するかを示します (プログラム カウンタを 0x3FF に設定します)。

ブロック パラメータ

次は、PicoBlaze Microcontroller ブロックに特有のパラメータです。

- [Version] : [PicoBlaze 2] または [PicoBlaze 3] を指定します。

- [Display internal state] : オンにすると、MATLAB ワークスペースにレジスタ フラグおよび制御フラグを使用できるようになります。この情報は、次の命名規則に従った構造で表示されます。

< design name >_< subsystem name >_< PicoBlaze block name >.

この構造には、各レジスタ (s00、s01 など) のフィールドと制御フラグ CARRY と ZERO が含まれます。

- [Display values as] : 値を表示するために使用する基数を指定します。

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

PicoBlaze アセンブラの使用法

メモ : ザイリンクス PicoBlaze Assembler は、Windows でのみ使用可能です。サードパーティの PicoBlaze Assembler は、Linux で使用できますが、ザイリンクスからは提供していません。

1. PicoBlaze プログラムを記述します。プログラムを拡張子 .psm で保存します。
2. MATLAB コマンド プロンプトで次のコマンドを入力し、アセンブラを実行します。

```
xlpb_as -p <your_psm_file>.
```

デフォルトでは、PicoBlaze 3 のプログラムがアセンブルされます。PicoBlaze 2 のプログラムをアセンブルする場合は、-v 2 オプションを使用します。このスクリプトにより、PicoBlaze アセンブラが実行され、M コード プログラムが生成されます。このプログラムを使用し、プログラムを保存するための ROM または RAM を生成します。

既知の問題

- PicoBlaze アセンブラの xlpb_as では、パス全体の文字数が 52 文字を超えるディレクトリにアセンブリ コード ファイルがあると、エラーになります。
- このブロックでは、Verilog ネットリストはサポートされません。

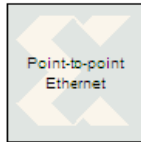
PicoBlaze Microprocessor のオンライン マニュアル

詳細は、

http://japan.xilinx.com/ipcenter/processor_central/picoblaze/picoblaze_user_resources.htm

を参照してください。

Point-to-point Ethernet Co-Simulation



ザイリンクスの Point-to-point Ethernet Co-Simulation ブロックは、イーサネット接続を介して、ハードウェア協調シミュレーションを実行するインターフェイスを提供します。

インターフェイス、必要条件、セットアップ手順などの詳細は、「[イーサネット ハードウェア協調シミュレーション](#)」を参照してください。

Point-to-Point Ethernet Co-Simulation ブロックは、System Generator ブロックの [Compilation] で [Ethernet] → [Point-to-point] を選択すると作成されます。作成されたブロックには、元のゲートウェイに対応するポート (またはサブシステム ポート) が含まれ、ほかの System Generator ブロックと同じように使用できます。このブロックは、Simulink のシミュレーション中に、FPGA ハードウェア プラットフォームとの通信に使用されます。ブロックの入力ポートに書き込まれたシミュレーション データは、ハードウェアに渡されます。データがブロックの出力ポートから読み出されると、ブロックはハードウェアから適切な値を読み込んで、出力ポートに駆動します。これで、データが Simulink で解釈できるようになります。また、ブロックではプラットフォームの開始、コンフィギュレーション、ステップ、終了が自動的に実行されます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- **[Clock source]** : シングル ステップまたはフリーランニング クロック ソースを選択できます。シングル ステップ クロックを選択すると、1 度に 1 クロック サイクルずつボードをステップできます。各クロック サイクル ステップは、Simulink での一定の時間に相当します。このクロック ソースを使用すると、シミュレーション中の協調シミュレーション ハードウェアのビヘイビアのビット精度および周期精度が、そのサブシステムのシミュレーション ビヘイビアよりも正確になります。シングル ステッピングは不要で、フリーランニング クロックでボードを実行できる場合もあります。この場合、ボードは Simulink シミュレーションと非同期で動作します。
- **[Has combinational path]** : ハードウェア協調シミュレーション ブロックの出力ポートから同じブロックの入力ポートへの直接組み合わせフィードバック パス (同じブロックの出力ポートから入力ポートへの接続) が必要な場合もあります。出力ポートから入力ポートへの直接フィードバック パスが必要で、デザインに入力ポートから出力ポートへの組み合わせパスが含まれない場合は、このチェック ボックスをオフにすると、デザインでフィードバック パスを使用できるようになります。
- **[Bitstream name]** : Specifies the co-simulation FPGA configuration file for the Point-to-point Ethernet hardware co-simulation platform. 新規の協調シミュレーション ブロックがコンパイル中に作成されるとこのパラメータが自動的に設定され、正しいコンフィギュレーション ファイルが使用されます。このパラメータは、コンフィギュレーション ファイルのディレクトリを変更した場合にのみ修正します。

[Ethernet] タブ

[Ethernet] タブからは、次のようなパラメータを設定できます。

- **[Host interface]** : ホスト ネットワーク インターフェイス カードを指定します。このカードは、協調シミュレーションでターゲット **FPGA** プラットフォームに接続するために使用されます。リストには、**point-to-point** のイーサネットの協調シミュレーションで使用されるネットワーク インターフェイス カードがすべて表示されます。情報パネルには、選択したインターフェイスの **MAC** アドレス、リンク速度、最大フレーム サイズと、**Windows** 環境での対応する接続名が表示されます。使用可能なインターフェイスのリストと情報パネルは、最新の状態ではないことがありますので、**[Refresh]** ボタンをクリックしてアップデートしてください。
- **[FPGA interface MAC address]** : ターゲット **FPGA** プラットフォームに関連付けられたイーサネット **MAC** アドレスを指定します。**MAC** アドレスは、2 桁の 16 進数 6 個をコロンで区切って指定します (**00:0a:35:11:22:33** など)。JTAG ベースのコンフィギュレーションの場合、イーサネット LAN のアドレスと競合していなければ、**MAC** アドレスは各 **FPGA** プラットフォームに任意で割り当てることができます。**point-to-point** のイーサネット接続のコンフィギュレーションの詳細は、**System ACE** を使用したコンフィギュレーションについて記述されたマニュアルを参照してください。**FPGA** プラットフォームの **MAC** アドレスのコンフィギュレーション方法の詳細が記述されています。

[Configuration] タブ

- **[Download cable]** : JTAG ベースのデバイス コンフィギュレーションの場合は、プログラム ケーブルにパラレル ケーブル **IV** かプラットフォーム **USB** を選択し、**point-to-point** のイーサネットの場合は、イーサネット接続でデバイス コンフィギュレーションを実行します。この設定方法は、「[ハードウェア プラットフォームのインストール](#)」を参照してください。
- **[Cable speed]** : JTAG ベースのコンフィギュレーションの場合にのみ使用します。ハードウェア協調シミュレーションでは、プログラム ケーブルをデフォルトの最大速度よりも、周波数で動作させる必要があることがあります。このメニューを使用すると、ハードウェア設定にあったケーブル速度を選択できます。通常、デフォルトの速度で問題ありませんが、**System Generator** で協調シミュレーション用にデバイスがコンフィギュレーションできなかった場合は、速度の遅いケーブルを使用してみてください。
- **[Configuration timeout]** : コンフィギュレーション プロセスに必要なタイムアウト値をミリ秒で指定します。通常はデフォルトの値で問題ありませんが、デバイス コンフィギュレーション後の接続の再構築に時間がかかりすぎる場合は、より大きな値を指定してみてください。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

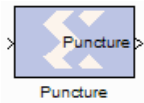
関連項目

[イーサネット ハードウェア協調シミュレーション](#)

[ポイントツーポイント イーサネット ハードウェア協調シミュレーション](#)

Puncture

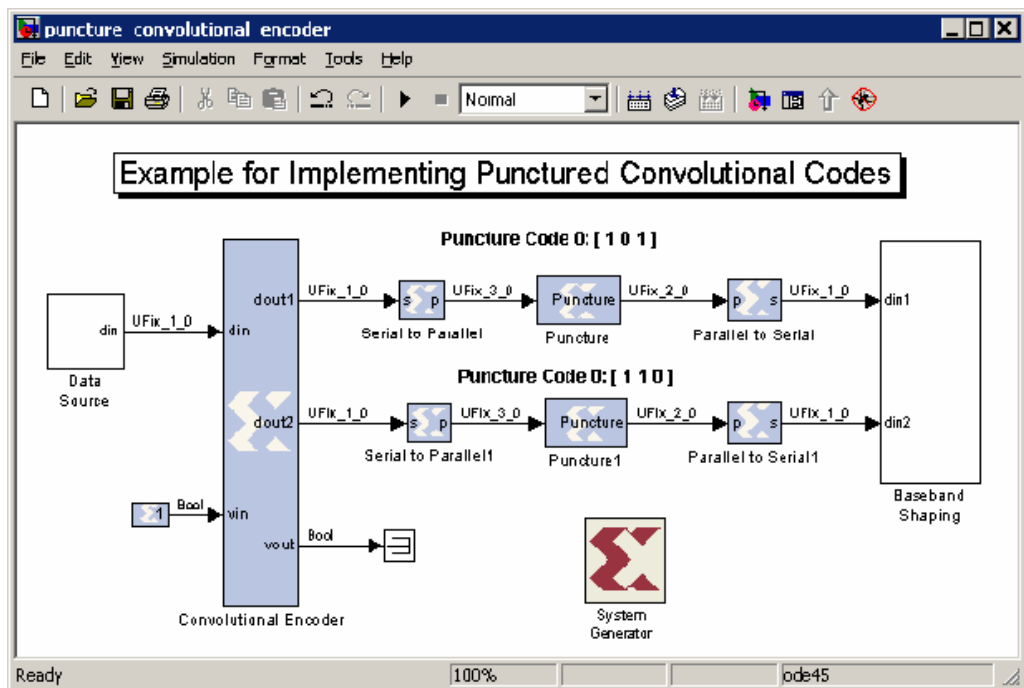
このブロックは、[Xilinx Blockset] の [Communication] および [Index] ライブラリにリストされています。



ザイリンクスの Puncture ブロックでは、データ ストリームの入力ワードからユーザーが指定したビットのセットが削除されます。

このブロックでは、パンクチャド符号パラメータ (削除するビットを指定する 2 進数ベクタ) に基づいて、UFixN_0 型 (N はパンクチャド符号の長さ) の入力データが UFixK_0 型 (K はパンクチャド符号の 1 の数) の出力データに変換されます。出力レートは、入力レートと同じになります。

このブロックは、よく Convolution Encoder ブロックと共に使用され、次の図に示すように、パンクチャドたたみ込み符号をインプリメントします。



図のシステムでは、Convolution Encoder ブロックが 1/2 のレートでインプリメントされており、出力がパンクチャド処理されて、3 入力ビットそれぞれに対して 4 出力ビットが出力されています。上の Puncture ブロックでは符号 0 ([1 0 1]) の真ん中のビットが削除され、下の Puncture ブロックでは符号 1 ([1 1 0]) の LSB が削除され、2 ビットにパンクチャド処理された値が出力されます。これらのデータストリームは、ベースバンド形成のために 1 ビットの同相の直交データストリームにシリアルライズされます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

このブロックでは、次のパラメータが設定できます。

- [Puncture code]: ビット ベクタとして表記されるパンクチャド パターンで、i の箇所が 0 になると、ビット i が削除されることを示します。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

Reed-Solomon Decoder 6.1

このブロックは、[Xilinx Blockset] の [Communication] および [Index] ライブラリにリストされています。



Reed-Solomon Decoder 6.1

Reed-Solomon (RS) コードは、ブロック ベースの誤り訂正符号 (ECC) で、デジタル通信およびストレージ分野のさまざまなアプリケーションに使用されています。

このコードは、デジタル ストレージ デバイス、ワイヤレス/モバイル通信、デジタル ビデオ放送などの多くのシステムでエラーを訂正するために使用されます。

RS Decoder では、RS Encoder で生成されたブロックを処理し、エラーを訂正して情報シンボルを復元しようとします。エラーの数とタイプは、コードの特性によって訂正することができます。

このブロックは、Spartan-3A DSP のほか、以前からサポートされていた Virtex-4、Virtex-5、Spartan-3、Spartan-3A/3AN、Spartan-3E でもサポートされます。

Reed-Solomon コードは、BCH (Bose-Chaudhuri-Hocquenghem) コード、つまり線形ブロック コードです。たとえば、線形コード (n, k) は、有限フィールドの n 次元ベクトル空間の k 次元部分空間を表します。このフィールドの要素は、「シンボル」と呼ばれます。Reed-Solomon コードでは、通常 n は論理式で、 s は各シンボルのビット幅です。コードが短縮されると、 n の値は小さくなります。デコーダは、全コードも短縮されたコードも処理します。また、エラーを含む可能性の高いシンボルである **erasure** も処理します。

デコーダでブロックが処理されると、次のいずれかが発生します。

1. 情報シンボルが復元されます。復元されるのは $2p+r < n-k$ の場合です。 p はエラー数、 r は **erasure** の数を示します。
2. デコーダから、情報シンボルを復元できないことがレポートされます。
3. デコーダでは、情報シンボルを復元できなかったのに、エラーがレポートされません。

どの状況になるかは、コードと通信チャネルの特性によって決まります。Simulink からは、チャネルのモデルおよびこれらの状況の予測が可能なツールが提供されています。

ブロック インターフェイス

ザイリンクスの RS Decoder ブロックには、**data_in**、**sync** という入力と、**data_out**、**blk_strt**、**blk_end**、**err_found**、**err_cnt**、**fail**、**ready**、**rfd** という出力があります。また、**n_in**、**erase**、**rst**、**en** というオプションの入力ポートと、**erase_cnt**、**data_del** というオプションの出力ポートも含めることができます。

次に、これらのポートについて説明します。

- **data_in** : デコードする n 個のシンボルのブロックが入力されます。**din** 信号は **UFI_Xs_0** 型にする必要があります。 s は各シンボルのビット幅です。
- **sync** : デコーダに **data_in** からのシンボルの処理をいつ開始するか伝えます。デコーダは、最初に **sync** がアサートされるまで入力シンボルを消去します。**sync** がアサートされたときのシンボルが、デコーダで処理される最初の n シンボルブロックの開始を表します。**sync** 信号は、デコーダが別のコードのブロックを受信する準備ができるまで無視されます。**sync** を駆動する信号は、ブール型にする必要があります。
- **reset** : デコーダが非同期にリセットされます。**reset** を駆動する信号は、ブール型にする必要があります。

- **reset** : デコーダがコード シンボルをデコードし始める前に少なくとも 1 サンプル周期間 High にする必要があります。
- **erase : din** : 現在ある信号を **erasure** として処理するかどうかを指定します。**erase** を駆動する信号は、ブール型にする必要があります。
- **n_in : n_in** : 各ブロックの開始時にサンプリングされます。新しいブロックの長さ **n_block** はサンプリングされた **n_in** に設定します。**din** 信号は **UFI_X_s_0** 型にする必要があります。**s** は各シンボルのビット幅です。
- **rst** : デコーダが同期にリセットされます。このポートは、[Provide synchronous reset port] をオンにするとブロックに追加されます。**rst** を駆動する信号は、ブール型にする必要があります。
- **en** : デコーダのイネーブル信号を伝達します。**en** を駆動する信号は、ブール型にする必要があります。
- **data_out** : デコードされた情報シンボルとパリティ シンボルが出力されます。**data_out** のデータ型は **data_in** と同じです。
- **blk_strt** : **data_out** にブロックの最初のシンボルが出力されると、1 になります。**blk_strt** からは、**UFI_X_1_0** 型の信号が出力されます。
- **blk_end** : **data_out** にブロックの最後のシンボルが出力されると、0 になります。**blk_end** からは、**UFI_X_1_0** 型の信号が出力されます。
- **err_found** : **data_out** にブロックの最後のシンボルが出力されたときの値になります。デコーダでエラーが検出され、デコード中に消去されると、値は 1 になります。**err_found** の値は、必ず **UFI_X_1_0** 型にします。
- **err_cnt** : **data_out** にブロックの最後のシンボルが出力されたときの値になります。値は、訂正されたエラーの数になります。**err_cnt** は、**UFI_X_b_0** 型にする必要があります。**b** は **n-k** を表すのに必要なビット数です。
- **fail** : **dout** にブロックの最後のシンボルが出力されたときの値になります。デコーダが情報シンボルを復元できなかった場合は 1 に、それ以外の場合は 0 になります。**fail** は、**UFI_X_1_0** 型にする必要があります。
- **ready** : デコーダが **data_in** 入力をサンプリングできるようになると 1 に、それ以外の場合は 0 になります。**ready** は、**UFI_X_1_0** 型にする必要があります。
- **rffd** : デコーダが **data_in** 入力のコード ブロックの最初のシンボルをサンプリングできるようになると 1 に、それ以外の場合は 0 になります。**rffd** は、**UFI_X_1_0** 型にする必要があります。
- **data_del** : デコードされたシンボルは **data_out** に出力され、デコードされていないシンボルは **data_del** に出力されます。**data_del** のデータ型は **data_in** と同じです。
- **erase_cnt** : 消去デコードがイネーブルになっているときのみ使用できます。**dout** にブロックの最後のシンボルが出力されたときの値になります。値は、訂正された削除の数になります。**erase_cnt** は、**UFI_X_b_0** 型にする必要があります。**b** は **n-k** を表すのに必要なビット数です。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Code specification] : 次の中から RS Decoder のタイプを指定します。

- ◆ [Custom] : ブロック パラメータをすべて設定できます。
- ◆ [ATSC] : ATSC (Advanced Television Systems Committee) 規格の短縮された RS コード (207, 187) をインプリメントします。
- ◆ [CCSDS] : CCSDS (Consultative Committee for Space Data Systems) 規格の短縮された RS コード (255, 223) をインプリメントします。
- ◆ [DVB] : DVB (Digital Video Broadcasting) 規格の短縮された RS コード (204, 188) をインプリメントします。
- ◆ IESS-308(126) IESS-308 (INTELSAT Earth Station Standard) 規格の短縮された RS コード (126, 112) をインプリメントします。
- ◆ IESS-308(194) IESS-308 規格の短縮された RS コード (194, 178) をインプリメントします。
- ◆ IESS-308(208) IESS-308 規格の短縮された RS コード (208, 192) をインプリメントします。
- ◆ IESS-308(219) IESS-308 規格の短縮された RS コード (219, 201) をインプリメントします。
- ◆ IESS-308(225) IESS-308 規格の短縮された RS コード (225, 205) をインプリメントします。
- ◆ [IEEE-802.16d] : IEEE-802.16d 規格の RS コード (255, 239) をすべてをインプリメントし
[Number of channels] : エンコーダで処理される時分割多重チャネルの数を指定します。
エンコーダでは、最大で 128 チャネルまでがサポートされます。
- [Clocks per symbol] : 入力データ シンボルごとに使用するサンプル周期数を指定します。この値を増加すると、プロセス遅延が削減され、コード ワードの連続デコードがサポートされます。入力データには、指定したクロック シンボル数が保持されます。
- [Provide erase port] : オンにすると、ブロックに erase 入力が増加されます。
- [Provide variable block length port (n_in)] : オンにすると、ブロックに n_in 入力が増加されます。
- [Provide original delayed data port (data_del)] : オンにすると、ブロックに data_del 出力が増加されます。
- [Symbol width] : コードのシンボルの幅をビットで示します。エンコーダでサポートされる幅は、3 ~ 12 までです。
- [Number of symbols per code block (n)] : エンコーダが出力するシンボルをブロック数で示します。使用できる値は、 $3 \sim 2S - 1$ で、s はシンボル幅です。
- [Number of information symbols per code block (k)] : 各ブロックに含まれる情報シンボルの数を示します。使用可能な値は、 $\max(n - 256, 1) \sim n - 2$ です。
- [Field polynomial] : 多項式を指定します。この多項式からシンボル フィールドが決まります。この値は 10 進数で指定します。この多項式は、プリミティブにする必要があります。値が 0 の場合は、デフォルトの多項式が使用されます。次の表は、デフォルトの多項式を示しています。

シンボル幅	多項式 (デフォルト)	アレイ
3	$x^3 + x + 1$	11
4	$x^4 + x + 1$	19

シンボル幅	多項式 (デフォルト)	アレイ
5	$x^5 + x^2 + 1$	37
6	$x^6 + x + 1$	67
7	$x^7 + x^3 + 1$	137
8	$x^8 + x^4 + x^3 + x^2 + 1$	285
9	$x^9 + x^4 + 1$	529
10	$x^{10} + x^3 + 1$	1033
11	$x^{11} + x^2 + 1$	2053
12	$x^{12} + x^6 + x^4 + x + 1$	4179

- [Generator start] : 生成多項式の最初のルート r を指定します。生成多項式 $g(x)$ は、次のように計算されます。

$$g(x) = \prod_{i=0}^{n-k-1} (x - \alpha^{h(r+i)})$$

α は、シンボル フィールドのプリミティブ エレメントです。スケール係数は次のパラメータで指定します。

- [Scaling factor for generator polynomial] : 上記の式では h で記述される部分で、コードのスケール係数を指定します。通常、 h は 1 ですが、 $2^S - 1$ の値まで設定できます。 s はシンボル幅です。この値を指定しないと α^h がプリミティブになりません。 h は $2^S - 1$ と互いに素な数にする必要があります。
- [Memory type] : [Automatic]、[Block RAM]、[Distributed memory] のいずれかを選択できます。
- [Optimization] : エリアとスピードのどちらを優先させるか選択します。
- [Self recovering state machine] : オンにすると、不正なステートになったときにブロックが同期でリセットされます。

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

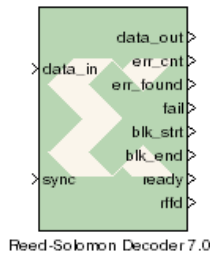
このブロックでは、次のザイリンクス LogiCORE Reed-Solomon Decoder コアが使用されます。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/データシート	Spartan				Virtex		
			3、3E	3A	3A DSP	6	4	5	6
Reed-Solomon Decoder 6.1	Reed-Solomon Decoder	V6.1	•	•	•		•	•	

このコアにはライセンスが必要です。購入は、次のザイリンクス Web サイトからできます。
http://japan.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=DO-DI-RSD

Reed-Solomon Decoder 7.0

このブロックは、[Xilinx Blockset] の [Communication] および [Index] ライブラリにリストされています。



Reed-Solomon Decoder 7.0

Reed-Solomon (RS) コードは、ブロック ベースの誤り訂正符号 (ECC) で、デジタル通信およびストレージ分野のさまざまなアプリケーションに使用されています。

このコードは、デジタル ストレージ デバイス、ワイヤレス/モバイル通信、デジタル ビデオ放送などの多くのシステムでエラーを訂正するために使用されます。

Reed-Solomon Decoder では、Reed-Solomon Encoder で生成されたブロックを処理し、エラーを訂正して情報シンボルを復元しようとします。エラーの数とタイプは、コードの特性によって訂正することができます。

このブロックは、Spartan-3A DSP のほか、以前からサポートされていた Virtex-4、Virtex-5、Spartan-3、Spartan-3A/3AN、Spartan-3E でもサポートされます。

Reed-Solomon コードは、BCH (Bose-Chaudhuri-Hocquenghem) コード、つまり線形ブロック コードです。たとえば、線形コード (n, k) は、有限フィールドの n 次元ベクトル空間の k 次元部分空間を表します。このフィールドの要素は、「シンボル」と呼ばれます。Reed-Solomon コードでは、通常 n は論理式で、 s は各シンボルのビット幅です。コードが短縮されると、 n の値は小さくなります。デコーダは、全コードも短縮されたコードも処理します。また、エラーを含む可能性の高いシンボルである **erasure** も処理します。

デコーダでブロックが処理されると、次のいずれかが発生します。

1. 情報シンボルが復元されます。復元されるのは $2p+r < n-k$ の場合です。 p はエラー数、 r は **erasure** の数を示します。
2. デコーダから、情報シンボルを復元できないことがレポートされます。
3. デコーダでは、情報シンボルを復元できなかったのに、エラーがレポートされません。

どの状況になるかは、コードと通信チャネルの特性によって決まります。Simulink からは、チャネルのモデルおよびこれらの状況の予測が可能なツールが提供されています。

ブロック インターフェイス

ザイリンクスの RS Decoder ブロックには、**data_in**、**sync** という入力と、**data_out**、**blk_strt**、**blk_end**、**err_found**、**err_cnt**、**fail**、**ready**、**rfd** という出力があります。また、**n_in**、**erase**、**rst**、**en** というオプションの入力ポートと、**erase_cnt**、**data_del** というオプションの出力ポートも含めることができます。

次に、これらのポートについて説明します。

- **data_in** : デコードする n 個のシンボルのブロックが入力されます。**din** 信号は UFIX_s_0 型にする必要があります。 s は各シンボルのビット幅です。
- **sync** : デコーダに **data_in** からのシンボルの処理をいつ開始するか伝えます。デコーダは、最初に **sync** がアサートされるまで入力シンボルを消去します。**sync** がアサートされたときのシンボルが、デコーダで処理される最初の n シンボルブロックの開始を表します。**sync** 信号は、デコーダが別のコードのブロックを受信する準備ができるまで無視されます。**sync** を駆動する信号は、ブール型にする必要があります。
- **erase** : **din** に現在ある信号を **erasure** として処理するかどうかを指定します。**erase** を駆動する信号は、ブール型にする必要があります。

- **n_in** : **n_in** は各ブロックの開始時にサンプリングされます。新しいブロックの長さ **n_block** はサンプリングされた **n_in** に設定します。**din** 信号は **UFIX_s_0** 型にする必要があります。**s** は各シンボルのビット幅です。**[Variable Block Length]** を選択すると、ブロックに追加されます。
- **rst** : デコーダがリセットされます。このポートは、**[Synchronous Reset]** をオンにするとブロックに追加されます。**rst** を駆動する信号は、ブール型にする必要があります。
- **reset** は、デコーダがコード シンボルをデコードし始める前に少なくとも 1 サンプル周期間 **High** にする必要があります。
- **en** : デコーダのクロック イネーブル信号を伝達します。**en** を駆動する信号は、ブール型にする必要があります。オプションのクロック イネーブル ピンを選択すると、ブロックに追加されます。
- **data_out** : デコードされた情報シンボルとパリティ シンボルが出力されます。**data_out** のデータ型は **data_in** と同じです。
- **blk_strt** : **data_out** にブロックの最初のシンボルが出力されると、1 になります。**blk_strt** からは、**UFIX_1_0** 型の信号が出力されます。
- **blk_end** : **data_out** にブロックの最後のシンボルが出力されると、0 になります。**blk_end** からは、**UFIX_1_0** 型の信号が出力されます。
- **err_found** : **data_out** にブロックの最後のシンボルが出力されたときの値になります。デコーダでエラーが検出され、デコード中に消去されると、値は 1 になります。**err_found** の値は、必ず **UFIX_1_0** 型にします。
- **err_cnt** : **data_out** にブロックの最後のシンボルが出力されたときの値になります。値は、訂正されたエラーの数になります。**err_cnt** は、**UFIX_b_0** 型にする必要があります。**b** は **n-k** を表すのに必要なビット数です。
- **fail** : **dout** にブロックの最後のシンボルが出力されたときの値になります。デコーダが情報シンボルを復元できなかった場合は 1 に、それ以外の場合は 0 になります。**fail** は、**UFIX_1_0** 型にする必要があります。
- **ready** : デコーダが **data_in** 入力をサンプリングできるようになると 1 に、それ以外の場合は 0 になります。**ready** は、**UFIX_1_0** 型にする必要があります。
- **rffd** : デコーダが **data_in** 入力のコード ブロックの最初のシンボルをサンプリングできるようになると 1 に、それ以外の場合は 0 になります。**rffd** は、**UFIX_1_0** 型にする必要があります。
- **info end** : **data_out** のブロック最後の情報シンボルを示します。
- **data_del** : デコードされたシンボルは **data_out** に出力され、デコードされていないシンボルは **data_del** に出力されます。**data_del** のデータ型は **data_in** と同じです。
- **erase_cnt** : 消去デコードがイネーブルになっているときのみ使用できます。**dout** にブロックの最後のシンボルが出力されたときの値になります。値は、訂正された削除の数になります。**erase_cnt** は、**UFIX_b_0** 型にする必要があります。**b** は **n-k** を表すのに必要なビット数です。
- **bit_err_0_to_1** : 0 として受信されて 1 に直されたビット数です。
- **bit_err_1_to_0** : 1 として受信されて 0 に直されたビット数です。
- **bit_err_rdy** : **bit_err_0_to_1** および **bit_err_1_to_0** が有効であることを示します。
- **mark_in** : **data_in** に印を付けるためのマーカー ビットです。
- **mark_out** : ブロック レイテンシによって遅延された **mark_in** です。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

[Page_0] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

コード ブロック指定

- [Code specification] : 次の中から RS Decoder のタイプを指定します。
 - ◆ [Custom] : ブロック パラメータをすべて設定できます。
 - ◆ [DVB] : DVB (Digital Video Broadcasting) 規格の短縮された RS コード (204, 188) をインプリメントします。
 - ◆ [ATSC] : ATSC (Advanced Television Systems Committee) 規格の短縮された RS コード (207, 187) をインプリメントします。
 - ◆ [CCSDS] : CCSDS (Consultative Committee for Space Data Systems) 規格の短縮された RS コード (255, 223) をインプリメントします。
 - ◆ IESS-308 (All) : IESS-308 (INTELSAT Earth Station Standard) 規格の短縮された RS コード (すべて) をインプリメントします。
 - ◆ IESS-308(126) IESS-308 (INTELSAT Earth Station Standard) 規格の短縮された RS コード (126, 112) をインプリメントします。
 - ◆ IESS-308(194) IESS-308 規格の短縮された RS コード (194, 178) をインプリメントします。
 - ◆ IESS-308(208) IESS-308 規格の短縮された RS コード (208, 192) をインプリメントします。
 - ◆ IESS-308(219) IESS-308 規格の短縮された RS コード (219, 201) をインプリメントします。
 - ◆ IESS-308(225) IESS-308 規格の短縮された RS コード (225, 205) をインプリメントします。
 - ◆ [IEEE-802.16d] : IEEE-802.16d 規格の RS コード (255, 239) すべてをインプリメントします。
- [Symbol width] : コードのシンボルの幅をビットで示します。エンコーダでサポートされる幅は、3 ～ 12 までです。
- [Field polynomial] : 多項式を指定します。この多項式からシンボル フィールドが決まります。この値は 10 進数で指定します。この多項式は、プリミティブにする必要があります。値が 0 の場合は、デフォルトの多項式が使用されます。次の表は、デフォルトの多項式を示しています。

シンボル幅	デフォルトの多項式	配列
3	$x^3 + x + 1$	11
4	$x^4 + x + 1$	19
5	$x^5 + x^2 + 1$	37
6	$x^6 + x + 1$	67
7	$x^7 + x^3 + 1$	137
8	$x^8 + x^4 + x^3 + x^2 + 1$	285
9	$x^9 + x^4 + 1$	529
10	$x^{10} + x^3 + 1$	1033

シンボル幅	デフォルトの多項式	配列
11	$x^{11} + x^2 + 1$	2053
12	$x^{12} + x^6 + x^4 + x + 1$	4179

- [Scaling Factor (h)] : 上記の式では h で記述される部分で、コードのスケール係数を指定します。通常、 h は 1 ですが、 $2S - 1$ の値まで設定できます。 s はシンボル幅です。この値を指定しないと α^h がプリミティブになりません。 h は $2S - 1$ と互いに素な数にする必要があります。
- [Generator Start] : 生成多項式の最初のルート r を指定します。生成多項式 $g(x)$ は、次のように計算されます。

$$g(x) = \prod_{i=0}^{n-k-1} (x - \alpha^{s(i+r)})$$

α は、シンボル フィールドのプリミティブ エLEMENT です。スケール係数は次のパラメータで指定します。

- [Variable Block Length] : オンにすると、ブロックに **n_in** 入力追加されます。
- [Symbols Per Block(n)] : エンコーダが出力するシンボルをブロック数で示します。使用できる値は、 $3 \sim 2S - 1$ で、 s はシンボル幅です。
- [Data Symbols(k)] : 各ブロックに含まれる情報シンボルの数を示します。使用可能な値は、 $\max(n - 256, 1) \sim n - 2$ です。

さまざまなチェック シンボル オプション

- [Variable Number of Check Symbols (r)]
- [Define Supported R_IN Values]
- **R_IN** でサンプリングされる可能性のある値の一部のみが実際に必要な場合は、コアの容量を少し減少することができます。たとえば、インテルサット規格の場合、**R_IN** 入力は 5 ビット幅になりますが、必要とされる r 値は 14、16、18、20 のみです。この場合、これら 4 つの値のみがサポートされるように定義すると、コアの容量を削減可能です。これ以外の値が **R_IN** でサンプリングされる場合、コアではデータが正しくデコードされません。
 - ◆ [Number of Supported R_IN Values] : サポートされる **R_IN** 値の数を指定します。
 - ◆ [Supported R_IN Definition File] : サポートされる **R** 値を定義する COE ファイルを指定します。COE ファイルのフォーマットは、`radix=10; legal_r_vector=14,16,18,20;` のようになります。`legal_r_vector` のエレメント数は指定した [Number of Supported R_IN Values] の数と同じにする必要があります。

[Page_1] タブ

インプリメンテーション

- [Optimization] : [Area] または [Speed] のいずれかを選択します。
- ステート マシン
- [Self Recovering] : オンにすると、不正なステートになったときにブロックが同期でリセットされます。
- [Memory Style] : [Distributed] (分散)、[Block] (ブロック)、[Automatic] (自動) のいずれかを選択します。

- [Clocks Per Symbol] : 入力データ シンボルごとに使用するサンプル周期数を指定します。この値を増加すると、プロセス遅延が削減され、コード ワードの連続デコードがサポートされます。入力データには、指定したクロック シンボル数が保持されます。
- [Number Of Channels] : エンコーダで処理される時分割多重チャネルの数を指定します。エンコーダでは、最大で 128 チャネルまでがサポートされます。
- パンクチャ オプション
- [Number of Puncture Patterns] : LogiCORE で処理するパンクチャ パターン数を指定します。パンクチャの必要ない場合は 0 に設定されます。
- [Puncture Definition File] : パンクチャ パターンを定義するのに使用するパンクチャ定義ファイル名を指定します。

[Page_2] タブ

- [Info End] : data_out のブロック最後の情報シンボルを示します。
- [Original Delayed Data] : オンにすると、ブロックに data_del 出力が追加されます。
- [Erase] : オンにすると、ブロックに erase 入力が追加されます。
- [Error Statistics] : 次の 3 つのエラー統計出力が追加されます。
 - ◆ bit_err_0_to_1 : 1 として受信されて 0 に直されたビット数です。
 - ◆ bit_err_1_to_0 : 0 として受信されて 1 に直されたビット数です。
 - ◆ bit_err_rdy : bit_err_0_to_1 および bit_err_1_to_0 が有効であることを示します。
- [Marker Bits] : ブロックに次のポートが追加されます。
 - ◆ mark_in : data_in に印を付けるためのマーカー ビットです。
 - ◆ mark_out : LogiCORE レイテンシによって遅延された mark_in です。
- [Number of Marker Bits] : マーカー ビットの数を指定します。

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

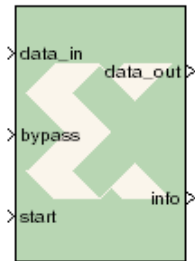
このブロックでは、次のザイリンクス LogiCORE が使用されます。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan				Virtex		
			3、 3E	3A	3A DSP	6	4	5	6
Reed-Solomon Decoder 7.0	Reed- Solomon Decoder	V7.0	•	•	•	•	•	•	•

このコアにはライセンスが必要です。購入は、次のザイリンクス Web サイトからできます。
http://japan.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=DO-DI-RSD

Reed-Solomon Encoder 6.1

このブロックは、[Xilinx Blockset] の [Communication] および [Index] ライブラリにリストされています。



Reed-Solomon Encoder 6.1

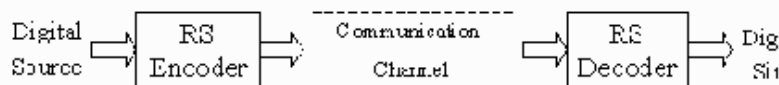
Reed-Solomon (RS) コードは、ブロック ベースの誤り訂正符号 (ECC) で、デジタル通信およびストレージ分野のさまざまなアプリケーションに使用されています。

このコードは、デジタル ストレージ デバイス、ワイヤレス/モバイル通信、デジタル ビデオ放送などの多くのシステムでエラーを訂正するために使用されます。

Reed-Solomon Encoder では、集まったデータ ブロックに冗長シンボルが追加され、転送中のエラーが訂正されます。エラーの原因は、ノイズやインターフェイス、CD の傷などさまざまです。RS Decoder は、エラーを修正し、元のデータを復元しようとします。エラーの数とタイプは、コードの特性によって訂正することができます。

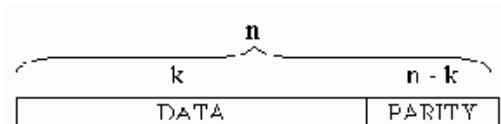
このブロックは、Spartan-3A DSP のほか、以前からサポートされていた Virtex-4、Virtex-5、Spartan-3、Spartan-3A/3AN、Spartan-3E でもサポートされます。

次の図は、典型的なシステムを示しています。



Reed-Solomon コードは、BCH (Bose-Chaudhuri-Hocquenghem) コード、つまり線形ブロック コードです。たとえば、線形コード (n, k) は、有限フィールドの n 次元ベクタ空間の k 次元部分空間を表します。このフィールドの要素は、「シンボル」と呼ばれます。Reed-Solomon コードでは、通常 n は論理式で $2^s - 1$ 、 s は各シンボルのビット幅です。コードが短縮されると、 n の値は小さくなります。エンコーダは、全コードも短縮されたコードも処理します。

エンコーダは、 $n-k$ パリティ シンボルを追加することで、長さ k の情報ブロックから長さ n のコードブロックを構築します。



Reed-Solomon コードは、フィールド多項式と生成多項式で指定されます。フィールド多項式はシンボル フィールドを構築するために、生成多項式はパリティシンボルを計算するために使用されます。エンコーダでは、どちらの多項式もコンフィギュレーションできます。生成多項式は、次のようになります。

$$g(x) = (x - \alpha^j)(x - \alpha^{j+1}) \dots (x - \alpha^{i+n-k-1})$$

α は、 $n + 1$ 個の要素を含む有限フィールドのプリミティブ エレメントです。

ブロック インターフェイス

ザイリンクスの RS Encoder ブロックには、`data_in`、`bypass`、`start` という入力ポートと、`data_out`、`info` という出力ポートが含まれます。また、`n_in`、`r_in`、`nd`、`rst`、`en` というオプションの入力ポートと、`rdy`、`rfd`、`rffd` というオプションの出力ポートも含まれます。

次に、これらのポートについて説明します。

- **data_in** : エンコードする n 個のシンボルのブロックが入力されます。各ブロックには、情報シンボル k とその後の未解釈のフィルタ シンボル $n - k$ で形成されます。`din` 信号は `UFX_s_0` 型にする必要があります。 s は各シンボルのビット幅です。
- **start** : エンコーダに `din` からのシンボルの処理をいつ開始するか伝えます。エンコーダは、最初に `start` がアサートされるまで入力シンボルを消去します。`start` がアサートされたときのシンボルが、エンコーダで処理される最初の n シンボル ブロックの開始を表します。`start` が 1 サンプル周期よりも長い間アサートされると、最後の周期の値がブロックの始めに使用されます。`start` 信号は、`bypass` が同時にアサートされると無視されます。`start` を駆動する信号は、ブール型にする必要があります。
- **bypass** : `bypass` がアサートされると、4 サンプル周期 (CCSDS の場合は 6 サンプル周期) の遅延が追加され、`din` の値が変更されずにそのまま `dout` に出力されます。`bypass` 信号は、エンコーダのステートに影響を与えません。`bypass` を駆動する信号は、ブール型にする必要があります。
- **n_in** : `n_in` は各ブロックの開始時にサンプリングされます。新しいブロックの長さ `n_block` はサンプリングされた `n_in` に設定します。`din` 信号は `UFX_s_0` 型にする必要があります。 s は各シンボルのビット幅です。
- **r_in** : `r_in` は各ブロックの開始時にサンプリングされます。新しいブロックの長さ `r_block` はサンプリングされた `r_in` に設定します。`r_in` 信号は `UFX_p_0` 型にする必要があります。 p はデフォルトのコード ワードでパリティ ビット ($n - k$) を表すのに必要なビット数です。
- **nd** : 各 `data_in` シンボルがパリティ シンボルを処理するための情報シンボルの一部であることを示します。`nd` を駆動する信号は、ブール型にする必要があります。
- **rst** : reset 信号を転送します。`rst` を駆動する信号は、ブール型にする必要があります。
- **en** : イネーブル信号を転送します。`en` を駆動する信号は、ブール型にする必要があります。
- **data_out** : n 個のシンボルを含むブロックを出力します。シンボルは、`data_in` から読み出された情報シンボル k がエンコードされたものです。
- **data_out** のデータ型は `data_in` と同じです。
- **info** : `data_out` の値が情報シンボルの場合は 1 に、パリティ シンボルの場合は 0 になります。`info` は、`UFX_1_0` 型にする必要があります。`rdy` : `data_out` の各シンボルが有効か無効かを示します。`rdy` は `UFX_1_0` 型にする必要があります。
- **rfd** : エンコーダが情報シンボルを入出力すると 1 に、パリティ シンボルを入出力すると 0 になります。`rfd` は、`UFX_1_0` 型にする必要があります。
- **rffd** : エンコーダが新しい `start` パルスを受信できるようになると 1 になります。`rffd` は、`UFX_1_0` 型にする必要があります。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

このブロックでは、次のパラメータが設定できます。

- [Code specification] : 次のいずれかからエンコーダのタイプを指定します。
 - ◆ [Custom] : ブロック パラメータをすべて設定できます。
 - ◆ [ATSC] : ATSC (Advanced Television Systems Committee) 規格の短縮された RS コード (207, 187) をインプリメントします。
 - ◆ [CCSDS] : CCSDS (Consultative Committee for Space Data Systems) 規格の短縮された RS コード (255, 223) をインプリメントします。
 - ◆ [DVB] : DVB (Digital Video Broadcasting) 規格の短縮された RS コード (204, 188) をインプリメントします。
 - ◆ IESS-308(126) IESS-308 (INTELSAT Earth Station Standard) 規格の短縮された RS コード (126, 112) をインプリメントします。
 - ◆ IESS-308(194) IESS-308 規格の短縮された RS コード (194, 178) をインプリメントします。
 - ◆ IESS-308(208) IESS-308 規格の短縮された RS コード (208, 192) をインプリメントします。
 - ◆ IESS-308(219) IESS-308 規格の短縮された RS コード (219, 201) をインプリメントします。
 - ◆ IESS-308(225) IESS-308 規格の短縮された RS コード (225, 205) をインプリメントします。
 - ◆ [ITU-J.83 Annex B] : ITU-J.83 Annex B 規格の拡張された RS コード (128, 122) をインプリメントします。
- [Number of channels] : エンコーダで処理される時分割多重チャネルの数を指定します。エンコーダでは、最大で 128 チャネルまでがサポートされます。
- [Provide variable number of check symbols (r_in)] : オンにすると、ブロックに r_in と n_in 入力が増加されます。
- [Provide variable block length port (n_in)] : オンにすると、ブロックに n_in 入力が増加されます。
- [Provide new data port (nd)] : オンにすると、ブロックに nd 入力が増加されます。
- [Provide ready port (rdy)] : オンにすると、ブロックに rdy 出力が増加されます。
- [Provide ready for data port (rfd)] : オンにすると、ブロックに rdy 出力が増加されます。
- [Provide ready for first data port (rffd)] : オンにすると、ブロックに rffd 出力が増加されます。
- [Symbol width] : コードのシンボルの幅をビットで示します。エンコーダでサポートされる幅は、3 ~ 12 までです。
- [n (number of symbols per code block)] : エンコーダが出力するシンボルをブロック数で示します。使用できる値は、 $3 \sim 2S - 1$ で、s はシンボル幅です。
- [k (number of information symbols per code block)] : 各ブロックに含まれる情報シンボルの数を示します。使用可能な値は、 $\max(n - 256, 1) \sim n - 2$ です。
- [Field polynomial] : 多項式を指定します。この多項式からシンボル フィールドが決まります。この値は 10 進数で指定します。この多項式は、プリミティブにする必要があります。値が 0 の場合は、デフォルトの多項式が使用されます。次の表は、デフォルトの多項式を示しています。

シンボル幅	多項式 (デフォルト)	アレイ
3	$x^3 + x + 1$	11
4	$x^4 + x + 1$	19
5	$x^5 + x^2 + 1$	37

シンボル幅	多項式 (デフォルト)	アレイ
6	$x^6 + x + 1$	67
7	$x^7 + x^3 + 1$	137
8	$x^8 + x^4 + x^3 + x^2 + 1$	285
9	$x^9 + x^4 + 1$	529
10	$x^{10} + x^3 + 1$	1033
11	$x^{11} + x^2 + 1$	2053
12	$x^{12} + x^6 + x^4 + x + 1$	4179

- **[Generator start]** : 生成多項式の最初のルート r を指定します。生成多項式 $g(x)$ は、次のように計算されます。

$$g(x) = \prod_{i=0}^{n-k-1} (x - \alpha^{h(r+i)})$$

- α は、シンボル フィールドのプリミティブ エレメントです。スケール係数は次のパラメータで指定します。
- **[Scaling factor for generator polynomial]** : コードのスケール係数を指定します。通常、スケール係数は 1 ですが、 $2^s - 1$ の値まで設定できます。s はシンボル幅です。この値を指定しないと α^h がプリミティブにならないので、h は $2^s - 1$ と互いに素な数にする必要があります。
- **[Memory style]** : **[Automatic]**、**[Block RAM]**、**[Distributed memory]** のいずれかを選択できます。このオプションは、CCSDS コードの場合にのみ選択できます。
- **[Check Symbol Generator]** : エリアと柔軟性のどちらを優先して最適化するか選択します。このオプションは、エンコーダにさまざまな数のチェック シンボルが入力された場合にのみ選択できます。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

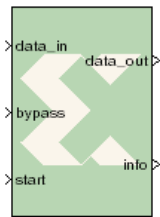
このブロックでは、次のザイリンクス LogiCORE Reed-Solomon Encoder コアが使用されます。:

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan				Virtex		
			3、3E	3A	3A DSP	6	4	5	6
Reed-Solomon Encoder 6.1	Reed-SolomonS Encoder	V6.1	•	•	•		•	•	

このコアにはライセンスが必要です。購入は、次のザイリンクス Web サイトからできます。
http://japan.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=DO-DI-RSD

Reed-Solomon Encoder 7.0

このブロックは、[Xilinx Blockset] の [Communication] および [Index] ライブラリにリストされています。



Reed-Solomon Encoder 7.0

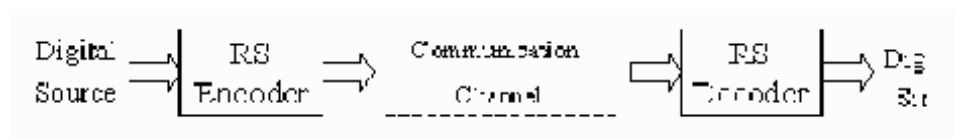
Reed-Solomon (RS) コードは、ブロックベースの誤り訂正符号 (ECC) で、デジタル通信およびストレージ分野のさまざまなアプリケーションに使用されています。

このコードは、デジタルストレージデバイス、ワイヤレス/モバイル通信、デジタルビデオ放送などの多くのシステムでエラーを訂正するために使用されます。

Reed-Solomon Encoder では、集まったデータブロックに冗長シンボルが追加され、転送中のエラーが訂正されます。エラーの原因は、ノイズやインターフェイス、CD の傷などさまざまです。RS Decoder は、エラーを修正し、元のデータを復元しようとしています。エラーの数とタイプは、コードの特性によって訂正することができます。

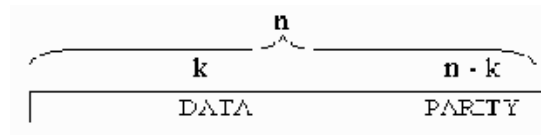
このブロックは、Spartan-3A DSP のほか、以前からサポートされていた Virtex-4、Virtex-5、Spartan-3、Spartan-3A/3AN、Spartan-3E でもサポートされます。

次の図は、典型的なシステムを示しています。



Reed-Solomon コードは、BCH (Bose-Chaudhuri-Hocquenghem) コード、つまり線形ブロックコードです。たとえば、線形コード (n, k) は、有限フィールドの n 次元ベクタ空間の k 次元部分空間を表します。このフィールドのエレメントは、「シンボル」と呼ばれます。Reed-Solomon コードでは、通常 n は論理式で $2^s - 1$ 、 s は各シンボルのビット幅です。コードが短縮されると、 n の値は小さくなります。エンコーダは、全コードも短縮されたコードも処理します。

エンコーダは、 $n-k$ パリティシンボルを追加することで、長さ k の情報ブロックから長さ n のコードブロックを構築します。



Reed-Solomon コードは、フィールド多項式と生成多項式で指定されます。フィールド多項式はシンボルフィールドを構築するために、生成多項式はパリティシンボルを計算するために使用されます。エンコーダでは、どちらの多項式もコンフィギュレーションできます。生成多項式は、次のようになります。

$$g(x) = (x - \alpha^j)(x - \alpha^{j+1}) \cdots (x - \alpha^{j+n-k-1})$$

α は、 $n+1$ 個のエレメントを含む有限フィールドのプリミティブエレメントです。

ブロック インターフェイス

ザイリンクスの RS Encoder ブロックには、`data_in`、`bypass`、`start` という入力ポートと、`data_out`、`info` という出力ポートが含まれます。また、`n_in`、`r_in`、`nd`、`rst`、`en` というオプションの入力ポートと、`rdy`、`rfd`、`rffd` というオプションの出力ポートも含まれます。

次に、これらのポートについて説明します。

- **data_in** : エンコードする n 個のシンボルのブロックが入力されます。各ブロックには、情報シンボル k とその後の未解釈のフィルタ シンボル $n - k$ で形成されます。`din` 信号は `UFX_s_0` 型にする必要があります。 s は各シンボルのビット幅です。
- **start** : エンコーダに `din` からのシンボルの処理をいつ開始するか伝えます。エンコーダは、最初に `start` がアサートされるまで入力シンボルを消去します。`start` がアサートされたときのシンボルが、エンコーダで処理される最初の n シンボル ブロックの開始を表します。`start` が 1 サンプル周期よりも長い間アサートされると、最後の周期の値がブロックの始めに使用されます。`start` 信号は、`bypass` が同時にアサートされると無視されます。`start` を駆動する信号は、ブール型にする必要があります。
- **bypass** : `bypass` がアサートされると、4 サンプル周期 (CCSDS の場合は 6 サンプル周期) の遅延が追加され、`din` の値が変更されずにそのまま `dout` に出力されます。`bypass` 信号は、エンコーダのステートに影響を与えません。`bypass` を駆動する信号は、ブール型にする必要があります。
- **n_in** : この信号は、ブロック サイズが変数の場合に使用されます。`n_in` は各ブロックの開始時にサンプリングされます。新しいブロックの長さ `n_block` はサンプリングされた `n_in` に設定します。`din` 信号は `UFX_s_0` 型にする必要があります。 s は各シンボルのビット幅です。
- **r_in** : この信号は、チェック シンボルの数を変数の場合に使用されます。`n_in` は各ブロックの開始時にサンプリングされます。新しいブロックの長さ `r_block` はサンプリングされた `r_in` に設定します。`r_in` 信号は `UFX_p_0` 型にする必要があります。 p はデフォルトのコード ワードでパリティ ビット ($n - k$) を表すのに必要なビット数です。
- **nd** : 各 `data_in` シンボルがパリティ シンボルを処理するための情報シンボルの一部であることを示します。`nd` を駆動する信号は、ブール型にする必要があります。
- **rst** : reset 信号を転送します。`rst` を駆動する信号は、ブール型にする必要があります。
- **en** : イネーブル信号を転送します。`en` を駆動する信号は、ブール型にする必要があります。
- **data_out** : n 個のシンボルを含むブロックを出力します。シンボルは、`data_in` から読み出された情報シンボル k がエンコードされたものです。`data_out` のデータ型は `data_in` と同じです。
- **info** : `data_out` の値が情報シンボルの場合は 1 に、パリティ シンボルの場合は 0 になります。`info` は、`UFX_1_0` 型にする必要があります。
- **rdy** : `data_out` の各シンボルが有効か無効かを示します。`rdy` は `UFX_1_0` 型にする必要があります。
- **rfd** : エンコーダが情報シンボルを入出力すると 1 に、パリティ シンボルを入出力すると 0 になります。`rfd` は、`UFX_1_0` 型にする必要があります。
- **rffd** : エンコーダが新しい `start` パルスを受信できるようになると 1 になります。`rffd` は、`UFX_1_0` 型にする必要があります。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

このブロックでは、次のパラメータが設定できます。

[Page_0] タブ

コード ブロック指定

- [Code specification] : 次のいずれかからエンコーダのタイプを指定します。
- [Custom] : ブロック パラメータをすべて設定できます。
- [DVB] : DVB (Digital Video Broadcasting) 規格の短縮された RS コード (204, 188) をインプリメントします。
- [ATSC] : ATSC (Advanced Television Systems Committee) 規格の短縮された RS コード (207, 187) をインプリメントします。
- [G_709] : 光ネットワークを使用したデータ通信用の G.709 規格をインプリメントします。
- [ETSI_BRAN] : BRAN (Broadband Radio Access Networks) 用の ETSI (欧州電気通信標準化機構) 規格をインプリメントします。
- [CCSDS] : CCSDS (Consultative Committee for Space Data Systems) 規格の短縮された RS コード (255, 223) をインプリメントします。
- [ITU-J.83 Annex B] : ITU-J.83 Annex B 規格の拡張された RS コード (128, 122) をインプリメントします。
- IESS-308(126) IESS-308 (INTELSAT Earth Station Standard) 規格の短縮された RS コード (126, 112) をインプリメントします。
- IESS-308(194) IESS-308 規格の短縮された RS コード (194, 178) をインプリメントします。
- IESS-308(208) IESS-308 規格の短縮された RS コード (208, 192) をインプリメントします。
- IESS-308(219) IESS-308 規格の短縮された RS コード (219, 201) をインプリメントします。
- IESS-308(225) IESS-308 規格の短縮された RS コード (225, 205) をインプリメントします。
- [Variable Number of Check Symbols (r)] : オンにすると、ブロックに r_{in} と n_{in} 入力が増加されます。
- [Variable Block Length] : オンにすると、ブロックに n_{in} 入力が増加されます。
- [Symbol width] : コードのシンボルの幅をビットで示します。エンコーダでサポートされる幅は、3 ~ 12 までです。
- [Field polynomial] : 多項式を指定します。この多項式からシンボル フィールドが決まります。この値は 10 進数で指定します。この多項式は、プリミティブにする必要があります。値が 0 の場合は、デフォルトの多項式が使用されます。次の表は、デフォルトの多項式を示しています。

シンボル幅	デフォルトの多項式	配列
3	$x^3 + x + 1$	11
4	$x^4 + x + 1$	19
5	$x^5 + x^2 + 1$	37
6	$x^6 + x + 1$	67
7	$x^7 + x^3 + 1$	137
8	$x^8 + x^4 + x^3 + x^2 + 1$	285
9	$x^9 + x^4 + 1$	529
10	$x^{10} + x^3 + 1$	1033

シンボル幅	デフォルトの多項式	配列
11	$x^{11} + x^2 + 1$	2053
12	$x^{12} + x^6 + x^4 + x + 1$	4179

- **[Scaling Factor (h)]** : コードのスケール係数を指定します。通常、スケール係数は 1 ですが、 $2S - 1$ の値まで設定できます。s はシンボル幅です。この値を指定しないと α^h がプリミティブにならないので、h は $2S - 1$ と互いに素な数にする必要があります。
- **[Generator start]** : 生成多項式の最初のルート r を指定します。生成多項式 $g(x)$ は、次のように計算されます。

$$g(x) = \prod_{i=0}^{n-k-1} (x - \alpha^{(ir + j)})$$

- **[Symbols Per Block(n)]** : エンコーダが出力するシンボルをブロック数で示します。使用できる値は、 $3 \sim 2S - 1$ で、s はシンボル幅です。
- **[Data Symbols(k)]** : 各ブロックに含まれる情報シンボルの数を示します。使用可能な値は、 $\max(n - 256, 1) \sim n - 2$ です。

[Page_1] タブ

インプリメンテーション

- **[Check Symbol Generator Optimization]** : 次のいずれかを選択できます。
 - ◆ **[Fixed Architecture]** : 効率の良い固定アーキテクチャを使用してチェック シンボル ジェネレータをインプリメントします。
 - ◆ **[Area]** : エリアおよび速度効率を目標にチェック シンボル ジェネレータのインプリメンテーションが最適化されます。入力範囲である **N_IN** は削減されます。
 - ◆ **[Flexibility]** : 入力範囲 **N_IN** を最大限にすることを目標にチェック シンボル ジェネレータのインプリメンテーションが最適化されます。
- **[Memory Style]** : **[Distributed]**、**[Block]**、**[Automatic]** メモリのいずれかを選択できます。このオプションは、CCSDS コードの場合にのみ選択できます。
- **[Number of Channels]** : エンコーダで処理される時分割多重チャネルの数を指定します。エンコーダでは、最大で 128 チャネルまでがサポートされます。

オプションのピン

- **CE** : オンにすると、ブロックに **ce** (クロック イネーブル) 入力が増加されます。
- **RDY** : オンにすると、ブロックに **rdy** (ready) 出力が増加されます。
- **ND** : オンにすると、ブロックに **nd** (new data) 入力が増加されます。
- **RFD** : オンにすると、ブロックに **rfd** (ready for data) 出力が増加されます。
- **SCLR** : オンにすると、ブロックに **sclr** (synchronous clear) 入力が増加されます。
- **RFFD** : オンにすると、ブロックに **rffd** (ready for first data) 出力が増加されます。
- このブロックで使用するその他のパラメータは、で説明されています。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

このブロックでは、次のザイリンクス LogiCORE が使用されます。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan				Virtex		
			3、 3E	3A	3A DSP	6	4	5	6
Reed-Solomon Encoder 7.0	Reed-Solomon Encoder	V7.0	•	•	•	•	•	•	•

このコアにはライセンスが必要です。購入は、次のザイリンクス Web サイトからできます。

http://japan.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=DO-DI-RSD

Register

このブロックは、[Xilinx Blockset] の [Basic Elements]、[Control Logic]、[Memory]、[Index] ライブラリにリストされています。



ザイリンクスの **Register** ブロックは、1 サンプル周期のレイテンシを含む D フリップフロップ ベースのレジスタになります。

ブロック インターフェイス

このブロックには、データ用の入力ポートが 1 つと入力リセット ポート (オプション) が 1 つ含まれます。初期出力値は、ブロックのパラメータ ダイアログ ボックスで指定できます。入力されたデータは、1 サンプル周期後に出力されます。リセットされると、レジスタはパラメータ ダイアログ ボックスで指定した初期値に戻ります。

Register ブロックは、オプションのリセット ポートが含まれている点と初期値をユーザーが指定できる点が **Delay** ブロックとは異なります。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- **[Initial value]** : レジスタの初期値を指定します。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

このブロックは、合成可能な **VHDL** モジュールとしてインプリメントされます。このブロックでは、ザイリンクス **LogiCORE** は使用されません。

Reinterpret

このブロックは、[Xilinx Blockset] の [Basic Elements]、[Math]、[Index] ライブラリにリストされています。



ザイリンクスの **Reinterpret** ブロックは、入力値に関係なく、出力を強制的に新しいデータ型にします。

2 進数に変更されずに渡されるので、このブロックはハードウェアでリソースを消費しません。出力のビット数は、入力のビット数と常に同じになります。

このブロックを使用すると、符号なしデータを符号付きデータとして、または符号付きデータを符号なしデータとして再解釈させることができます。また、データ内の 2 進小数点の位置を変更することで、データのスケールリングも再解釈させることができます。ザイリンクスの **Scale** ブロックにも同じような機能があります。

このブロックは、たとえば 6 ビット幅の符号付き、小数点以下の 2 ビットの入力を、符号なしの小数点以下の 0 ビットの出力にした場合、-2.0 (2 進数で 1110.00、2 の補数) の入力を 56 (2 進数で 111000) の出力に変換します。

このブロックは、ザイリンクスの **Slice** ブロックまたは **Concat** ブロックと組み合わせて使用すると特に便利で、次のような場合に使用します。

たとえば、符号付きデータ 1 つと符号なしの 2 ビット (UFix_2_0) を運ぶ 2 つの信号がある場合に、2 つ目の信号からの 2 ビットを連結して、符号付信号の最後尾 (LSB) に渡すシステムを設計するとします。

設計するには、**Reinterpret** ブロック 2 つと **Concat** ブロック 1 つを使用します。まず、最初の **Reinterpret** ブロックで符号付き入力信号を 2 進小数点の位置が 0 の符号なしの値にします。この値と別の信号の UFix_2_0 を **Concat** ブロックに入力します。**Concat** ブロックの後には、2 つ目の **Reinterpret** ブロックを配置し、Concat ブロックの出力値を正しい位置に 2 進小数点が付いた符号付きの値に戻します。

このシステムには、ブロックが 3 つ必要ですが、ハードウェア インプリメンテーションでは単にバスの連結として認識されるので、ハードウェア コストはかかりません。

ブロック パラメータ

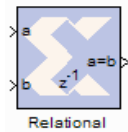
このブロックのパラメータを次に示します。

- **[Force Arithmetic Type]** : オンにすると、**[Output Arithmetic Type]** パラメータが設定できるようになり、その設定によって出力値の演算タイプが決まります。オフにすると、出力値の演算タイプは入力値と同じになります。
- **[Output Arithmetic Type]** : 出力を符号なしか、符号付き (2 の補数) のどちらにするか指定できます。
- **[Force Binary Point]** : オンにすると、**[Output Binary Point]** パラメータが設定できるようになり、その設定によって出力値の 2 進小数点の位置が決まります。オフにすると、出力値の演算タイプは入力値と同じになります。
- **[Output Binary Point]** : 出力の 2 進小数点の位置を指定できます。使用できる値は、0 ~ 入力のビット数の間の整数です。

このブロックでは、ハードウェア リソースは使用されません。

Relational

このブロックは、[Xilinx Blockset] の [Basic Elements]、[Control Logic]、[Math]、[Index] ライブラリにリストされています。



ザイリンクスの Relational ブロックは、コンパレータをインプリメントします。

サポートされている比較演算は次のとおりです。

- equal-to ($a = b$)
- not-equal-to ($a \neq b$)
- less-than ($a < b$)
- greater-than ($a > b$)
- less-than-or-equal-to ($a \leq b$)
- greater-than-or-equal-to ($a \geq b$)
- ブロックの出力はブール型になります。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

ブロックに特有のパラメータは、次のパラメータのみです。

- [Comparison] : ブロックで使用される比較演算子を指定します。

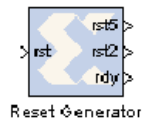
このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

Relational ブロックでは、ザイリンクス LogiCORE は使用されません。

Reset Generator

このブロックは、[Xilinx Blockset] の [Basic Elements] ライブラリと [Index] ライブラリにリストされています。

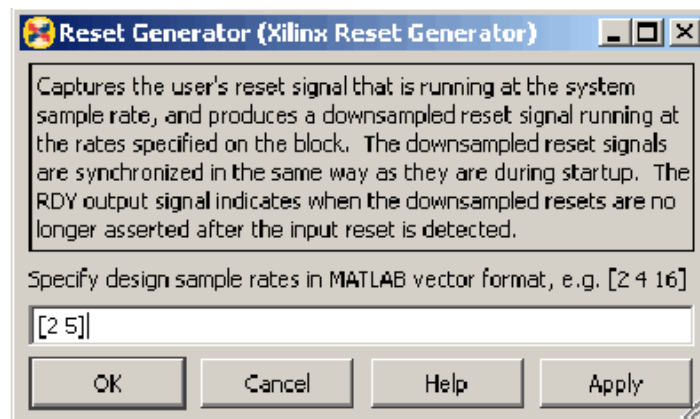


Reset Generator ブロックは、システム サンプル レートで実行されるユーザー リセット信号をキャプチャし、ブロックで指定されたレートで実行される 1 つまたは複数のダウンサンプルされたリセット信号を生成します。

ダウンサンプルされたリセット信号は起動時と同じように同期されます。RDY 出力信号は、入力リセットが検出された後に、ダウンサンプルされたリセットがアサートされなくなることを示します。

ブロック パラメータ

ブロック パラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。



デザインのサンプル レートは図のように MATLAB ベクタ形式で指定します。出力の数はいくつにでも指定できます。

Resource Estimator

このブロックは、[Xilinx Blockset] の [Tools] および [Index] ライブラリにリストされています。



ザイリンクスの **Resource Estimator** ブロックを使用すると、**System Generator** サブシステムまたはモデルをインプリメントするのに必要な **FPGA** リソースが高速に概算できます。

このブロックは、ザイリンクス ブロック専用の概算ツールを起動し、値を合計して、ルックアップ テーブル (**LUT**)、フリップフロップ (**FF**)、ブロック メモリ (**BRAM**)、**18X18** 乗算器、トライステート バッファ、**I/O** などの大まかな値を計算します。

FPGA リソースを必要とするザイリンクス ブロックには、すべてそのリソース必要条件を含むベクタを格納するパラメータが含まれます。**Resource Estimator** ブロックは、下位の関数を起動できます。これらの関数では、これらのベクタを生成したり (パラメータまたはデータ型が変更された後)、またはベクタに格納されていた前に計算済みの値を合計したりできます。各ブロックに含まれる **[Define FPGA area for resource estimation]** チェック ボックスをオンにすると、概算ツール関数を起動せずに、ベクタに格納された概算値が使用されます。

Resource Estimator ブロックは、モデルのサブシステム内のどこにでも配置できます。別の **Resource Estimator** ブロックがこのブロックの下位階層に配置される場合は、次の説明を参照してください。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

Resource Estimator ブロックのダイアログ ボックスには、次のパラメータが含まれます。

- **[Slices]** : ブロックで使用されるスライス (1 つのスライスには、通常フリップフロップが 2 つ、**LUT** が 2 つ、関連する **MUX**、キャリー ロジック、制御ロジックが含まれます)。
- **[FFs]** : ブロックで使用されるフリップフロップ
- **[BRAMs]** : ブロックで使用されるブロック RAM (**BRAM**)
- **[LUTs]** : ブロックで使用されるルックアップ テーブル
- **[IOBs]** : ブロックに使用される入力/出力ブロック
- **[Emb. Mults]** : ブロックで使用されるエンベデッド乗算器 (たとえば、**Virtex-II** デバイスにはエンベデッド **18X18** 乗算器が含まれます)。
- **[TBUFs]** : ブロックで使用されるトライステート バッファ
- **[Use area above]** : オンにすると、このサブシステムで実行されるリソース概算の結果が、ダイアログ ボックスで指定した値になります (ダイアログ ボックスで指定した値は、各 **System Generator** ブロックの **[FPGA Area Estimation]** フィールドと同じになります)。オンにすると、このブロックのサブシステムと同じレベルにあるブロック、および下位のブロックのリソースは自動的に概算されません。
- **[Estimate options]** : 概算方法を **[Estimate]**、**[Quick]**、**[Post Map]**、**[Read Mrp]** のいずれかにします。これらのオプションの詳細は、次のセクションで説明します。
- **[Estimate]** ボタン : リソース概算が開始されます。

リソース概算の実行

上で説明した FPGA エリアのフィールドは、手動で入力することもできますが、次のいずれかの [Estimate options] を選択してリソース概算を実行すると、自動的に入力されます。

- **[Estimate]** : 各ブロックとサブシステムでブロックの概算関数をトップダウンで再帰的に起動します。概算関数を含まず、ハードウェアにインプリメント可能なブロック (共有メモリ ブロック以外) の場合は、ポスト マップ エリアを使用して自動的に概算されます。[Define FPGA area for resource estimation] がオンになっているブロックでは、その概算関数は実行されず、格納された概算が使用されます。Resource Estimator ブロックで [Use area above] をオンにすると、このブロックの概算がそれを含むサブシステム全体に使用されます。そのモデルの階層部分に対しては、ほかの概算関数は実行されません。
- **[Quick]** : [Estimate] ボタンをクリックすると、現在のサブシステムと同じ階層、またはそれより下位のブロックおよびサブシステムで指定した FPGA エリアの値がすべて合計されます。下位の概算関数は起動されません。
- **[Post-Map]** : [Estimate] ボタンをクリックすると、サブシステム全体に対してザイリンクスのマップ ツールが自動的に起動され、作成された MRP ファイル (Map Report File) から結果を読み戻します。このオプションを使用するには、System Generator ブロックと Resource Estimator ブロックを概算されるサブシステムと一緒にインスタンス化する必要があります。
- **[Read Mrp]** : [Estimate] ボタンをクリックすると、ファイル ブラウザが開き、選択した MRP ファイルからの結果が Resource Estimator に読み込まれます。リソース情報を入手する方法は、合成、変換、マップ 済みのサブシステムでのみ使用できます。このオプションは、概算関数を含まず、デザインを変更しない複雑なザイリンクス ブロックで使用すると便利です。

MRP ファイルからの数と Resource Estimator のダイアログ ボックスから挿入された数がわずかに異なることがあります ([Post-Map] オプションを使用した場合にも異なることがあります)。MPR ファイルの IOB FF リソースは、すべてダイアログ ボックスの [FFs] フィールドに追加されます。また、MPR の同じ行の IOB FF リソースの半分は、[Slices] に追加されます。[IOBs] は [Post-Map] または [Read MRP] オプションで概算を実行すると、常に 0 に設定されます。通常、この機能はサブシステムを概算するときに便利なので、IOB リソースは CLB 使用率に含めておき、最終デザインで使用されない IOB リソースが間違ってレポートされないようにします。

Resource Estimation ブロックでサポートされるブロック

高速なリソース概算関数を含むブロック

Accumulator、Addressable Shift Register、AddSub、CMult (シーケンシャル版はサポートなし)、Convert、Counter、Delay、Down Sample、Dual Port RAM、FIFO、FFT、FFTx、Gateway In、Gateway Out、Inverter、LFSR、Logical、Mult (シーケンシャル版はサポートなし)、Mux (トライステート版はサポートなし)、Negate、Parallel to Serial、PicoBlaze Processor、Register、Relational、ROM、Serial To Parallel、Shift、Sine Cosine、Single Port RAM、Threshold、Up Sample

ポスト マップ 概算を使用するブロック

高速なリソース概算の関数を含まないでハードウェアを使用する System Generator ブロックは、ポスト マップ エリアを使用して概算されます。この方法が使用されないようにするには、定数またはユーザー作成の概算関数を [FPGA area] フィールドに入力し、[Define FPGA area for resource estimation] をオンにします。

ハードウェアを使用しないブロック

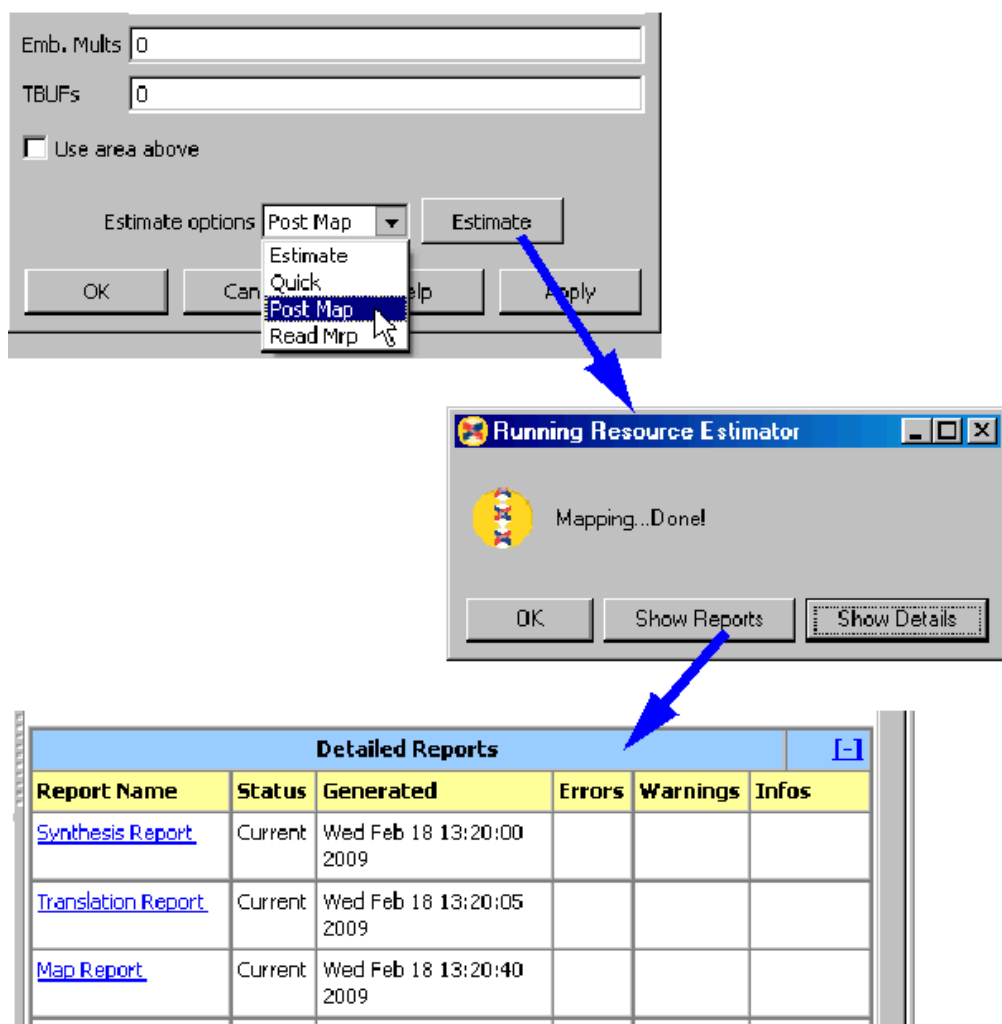
System Generator、Clear Quantization Error、Clock Enable Probe、Clock Probe、Concat、Constant、Discard Subsystem、FDATool、Indeterminate Probe、ModelSim、Pause Simulation、PicoBlaze Instruction Display、Quantization Error、Reinterpret、Sample Time、Scale、Simulation Multiplexer、Single-Step Simulation、Slice、BitBasher

特別な処理が必要なブロック

Discard Subsystem ブロックの場合、Resource Estimator ブロックでこのブロックを含むサブシステムに含まれるリソースがすべて無視されます。Shared Memory ブロックは概算されません。Shared Memory ブロックを含むデザインでは、Multiple Subsystem Generator ブロックを使用して HDL ネットリスト ファイルを生成します。ISE を使用してデザインの MPR ファイル (Map Report File) を作成し、[Read MRP] オプションを使用してこの MRP ファイルに含まれる結果を取得します。

ISE レポートの表示

[Estimate option] で [Post Map] を選択して [Estimate] ボタンをクリックすると、次のような [Running Resource Estimator] ダイアログ ボックスが表示されます。[Show Reports] ボタンをクリックすると、関連する ISE レポートが表示されます。



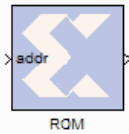
既知の問題

System Generator の Resource Estimation ブロックには、次のような既知の問題があります。

- 概算は、Simulink がコンパイル中に計算する各ブロックの入力と出力のデータ型に基づいて計算されます。ブロック レベルではわからないトリミングが実行されると、これらのトリミングされたリソース分が余分に概算されます。
- 合成ツールでまとめられた別々のブロックのロジックは、余分に概算されます。たとえば、レイテンシのないブロックを使用する場合、ブロックの境界を超えて組み合わせロジックが最適化されることがよくあります。
- マルチレート デザインのクロック イネーブル生成ロジックは、実際よりも低く概算されます。System Generator では、1 つのクロックを使用して、レートごとに別のクロック イネーブルが生成されます。クロック イネーブル ドライバのロジック量を正しく概算するためには、Resource Estimator でブロック レベルではなく、システム全体を認識させる必要があります。実際よりも低く概算されるリソースには、この追加のクロック イネーブル接続に関連したものも含まれます。クロック イネーブル接続は、ブロック概算関数からは認識されない各ブロックに対して追加されます。
- Shared Memory ブロックは概算されません。Shared Memory ブロックを含むデザインでは、レポートされる概算に Shared Memory ブロックで使用するリソースが含まれません。

ROM

このブロックは、[Xilinx Blockset] の [Control Logic]、[Memory]、[Index] ライブラリにリストされています。



ザイリンクスの ROM ブロックは、シングル ポートの読み出し専用メモリ (ROM) です。

値はワードで格納され、すべてのワードが同じ演算タイプ、幅、2 進小数点になります。各ワードは、調度 1 つのアドレスに該当します。アドレスは、0 ~ d1 の符号なしの固定小数点型の値になります。d は、ROM の深さ (ワード数) を表します。メモリ コンテンツは、ブロック パラメータから指定できます。このブロックには、メモリ アドレス用の入力ポートが 1 つと、データ出力用の出力ポートが 1 つ含まれます。アドレス ポートは、符号なしの固定小数点型になります。このブロックは、分散メモリかブロック メモリのどちらかのザイリンクス LogiCORE を使用してインプリメントされます。

Virtex-4、Virtex-5、Virtex-6、Spartan-6、Spartan3A DSP デバイスにシングル ポート ROM ブロックをインプリメントする場合は、次の設定をしておく、最速のタイミング パフォーマンスを達成できます。

- [Provide reset port for output register] をオフにします。
- [Depth] には、16,384 未満の値を指定します。
- [Latency] は 2 またはそれ以上の値に設定します。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Depth] : 格納できるワード数を指定します。正の値しか指定できません。
- [Initial value vector] : 初期値を指定します。ベクタ長が ROM の深さよりも大きい場合、このベクタの後に続くエレメントは削除されます。ROM の深さがベクタ長よりも大きい場合、この ROM の後続ワードは 0 に設定されます。初期値ベクタは、ROM の指定されたデータ精度に従ってサチュレートされるか、丸められます。
- [Memory Type] : 分散 RAM またはブロック RAM を使用するかどうかを指定します。
- [Provide reset port for output register] : オンにすると、ブロック ROM の出力レジスタのリセット ポートにアクセスできるようになります。リセット ポートは、ブロック ROM のレイテンシが 1 に設定されている場合のみ使用できます。
- [Initial value for output register] : 出力レジスタの初期値を指定します。初期値は、ROM の指定されたデータ精度に従ってサチュレートされるか、丸められます。この初期値を設定するオプションは、Spartan-3、Virtex-4、Virtex-5、Virtex-6、Spartan-6、Spartan-3A DSP デバイスでのみ使用できます。

[Output Type] タブ

[Output Type] タブからは、次のようなパラメータを設定できます。

- [Word type] : データを符号付きにするか符号なしにするか指定します。
- [Number of bits] : メモリ ワードのビット数を指定します。

- [Binary point]: メモリ ワードの 2 進小数点の位置を指定します。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

このブロックでは、常にザイリンクス LogiCORE の Single Port Block Memory または Distributed Memory が使用されます。

ブロック メモリの場合、アドレス幅は $\text{ceil}(\log_2(d))$ と同じにする必要があります。d は、メモリの深さを示しています。このブロック メモリの最大のデータ ワード幅は、指定した深さになり、最大の深さは、ターゲットにしたデバイス ファミリによって異なります。次の表は、ブロック メモリの深さ別に、最大データ ワード幅をそれぞれ示しています。

深さ別の最大データ幅 (Spartan-3)

深さ	幅
2 ~ 2048	256
2049 ~ 4096	192
4097 ~ 8192	96
8193 ~ 16K	48
16K+1 ~ 32K	24
32K+1 ~ 64K	12
64K+1 ~ 128K	6
128K+1 ~ 256K	3

深さ別の最大データ幅 (Virtex-4/Virtex-5/Spartan-3A DSP)

深さ	幅
2 ~ 8192	256
8193 ~ 16K	192
16K+1 ~ 32K	96
32K+1 ~ 64K	48
64K+1 ~ 128K	24
128K+1 ~ 256K	12
256K+1 ~ 512K	6
512K+1 ~ 1024K	3

分散メモリのパラメータを選択した場合は、LogiCORE Distributed Memory が使用されます。深さは、Spartan-3、Virtex-4、Virtex-5、Spartan-3A DSP デバイスの場合は 16 ~ 65536、その他の FPGA ファミリの場合は 16 ~ 4096 の範囲で指定する必要があります。ワード幅は、1 ~ 1024 の範囲で指定する必要があります。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
ROM	Block Memory Generator	V3.3		•	•	•	•	•	•	•	•	•
	Distributed Memory Generator	V4.3	•	•	•	•	•	•	•	•	•	•

Sample Time

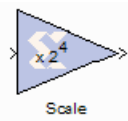
このブロックは、[Xilinx Blockset] の [Tools] および [Index] ライブラリにリストされています。



Sample Time ブロックでは、標準化された入力値のサンプル周期がレポートされます。標準化されたサンプル周期は、**Simulink** の絶対サンプル周期とは異なります。ハードウェアでは、定数としてインプリメントされます。

Scale

このブロックは、[Xilinx Blockset] の [Data Types]、[Math]、[Index] ライブラリにリストされています。



ザイリンクスの **Scale** ブロックでは、入力が 2 のべき乗でスケール変換されます。べき乗の値は、正にでも負にでもできます。このブロックには、入力が 1 つと出力が 1 つあります。スケール変換では、コンテナ内のビットを変更せずに 2 進小数点の位置が移動されます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

ブロックに特有のパラメータは、スケール係数を指定する [Scale factor *s*] だけで、正または負の整数を指定できます。ブロックの出力は $i \cdot 2^k$ になります。*i* は入力値、*k* はスケール係数を示します。スケール変換をすると、2 進小数点の位置が移動し、ハードウェアでコストがかからなくなります (ただし、シフトにより、ロジックが追加されることがあります)。

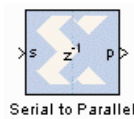
このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

このブロックでは、ザイリンクス **LogiCORE** は使用されません。

Serial to Parallel

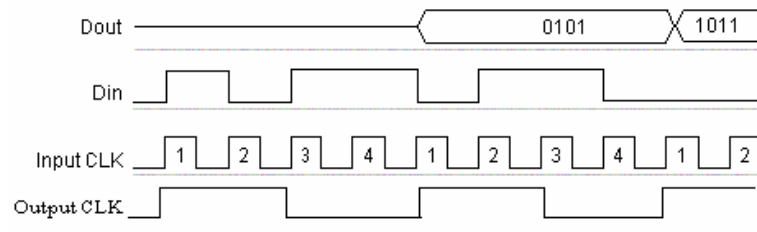
このブロックは、[Xilinx Blockset] の [Basic Elements]、[Data Types]、[Index] ライブラリにリストされています。



Serial to Parallel

Serial to Parallel ブロックでは、連続して入力されたデータが指定した倍数のサイズの出力 1 つにまとめられて、出力されます。この連続入力は、最上位ワードまたは最下位ワードのいずれかを先頭にした順序になります。

次の波形図は、このブロックの動作を示しています。



この例では、入力幅 1、出力幅 4 で、ワード サイズが 1 ビットの場合に、最上位ワードから入力されるように設定されています。

ブロック インターフェイス

Serial to Parallel ブロックには、入力が 1 つと出力が 1 つあります。入力ポートは、どのサイズにでもできます。出力ポート サイズは、ブロックのパラメータ ダイアログ ボックスで指定できます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

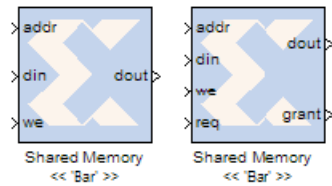
- [Input order] : 最下位ワードか最上位ワードのどちらを最初に入力するか指定します。
- [Arithmetic type] : 出力を符号付きか符号なしのいずれかに指定します。
- [Number of bits] : 出力幅は、入力ビット数の倍数である必要があります。
- [Binary point] : 出力の 2 進小数点の位置を指定します。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

入力ビット数で出力ビット数が割れない場合は、エラー メッセージが表示されます。このブロックの最小レイテンシは 0 です。

Shared Memory

このブロックは、[Xilinx Blockset] の [Shared Memory] および [Index] ライブラリにリストされています。



ザイリンクスの Shared Memory ブロックは、複数のデザインや 1 つのデザインの選択した箇所で共有できるランダム アクセス メモリ (RAM) をインプリメントします。

Shared Memory ブロックは、その名前で識別されます。図の Shared Memory ブロックには、**Bar** という名前が付いています。この Bar のインスタンス同士は、同じモデル内にあろうが、別のモデルにあろうが、MATLAB の別のインスタンスであっても、同

じメモリ空間を共有します。System Generator のハードウェア協調シミュレーション インターフェイスでは、Shared Memory ブロックを FPGA ハードウェアにコンパイルし、協調シミュレーションできます。これらのインターフェイスにより、ハードウェア ベースの共有メモリ リソースを、ホスト PC の共通アドレス空間にマップすることが可能です。Shared Memory を System Generator 協調シミュレーション ハードウェアで使用すると、ホスト PC と FPGA 間でデータを高速に転送でき、リアルタイム ハードウェア協調シミュレーション機能が強化されます。

9.2 リリースからは、同じ名前の Shared Memory ブロック同士がペアになり、ネットリストでは 1 つの BRAM ベースの Dual Port RAM ブロックになっています。Shared Memory ブロックがペアになっていない場合は、ブロックの入力ポートおよび出力ポートは System Generator デザインの最上位に配置されます。ペアになったブロックはデザインのどの階層にでも配置できますが、同じ名前の Shared Memory ブロックが 2 つより多いと、エラーになります。

以前のバージョンとの互換性を保持するには、MATLAB グローバル変数 `xlSgSharedMemoryStitch` を `off` に設定してください。これには、MATLAB コマンド ラインに次のように入力します。

```
global xlSgSharedMemoryStitch;
xlSgSharedMemoryStitch = 'off';
```

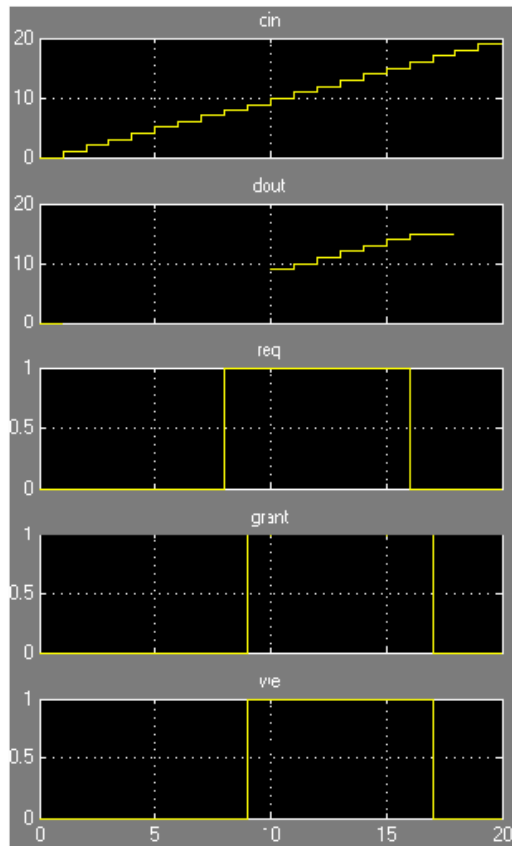
ブロック インターフェイス

デフォルトでは、Shared Memory ブロックには 3 つの入力 (`addr`、`din`、`we`) と、1 つの出力 (`dout`) が含まれます。Shared Memory ブロックへのアクセスは、[Access protection] を [Lockable] に設定すると保護できます。[Lockable] にすると、入力ポート `req` と出力ポート `grant` の 2 つのポートが追加されます。

`addr` は `UFIX_N_0` 型の信号で駆動する必要があります。この場合、`N` は `ceil(log2(depth))` です。メモリのワード サイズは、コンパイル時に、`din` ポートを駆動する信号のビット幅で決まります。書き込みイネーブル (`we`) ポートが 1 で駆動されると、`din` ポートの値が `addr` ポートで指定したメモリ アドレスに書き込まれます。

[Access protection] を [Lockable] に設定すると、メモリへのアクセス制御に `req` ポートと `grant` ポートが使用されます。読み出しや書き込みの前に、まず `req` を 1 に設定してリクエストをする必要があります。 `grant` が 1 になると、アクセス リクエストが許可され、読み出しまたは書き込みが

実行できるようになります。次の図は、req、grant、we ポートの関係を示しています。また、この図から、メモリへのアクセスが許可されるまでブロック出力がないことがわかります。

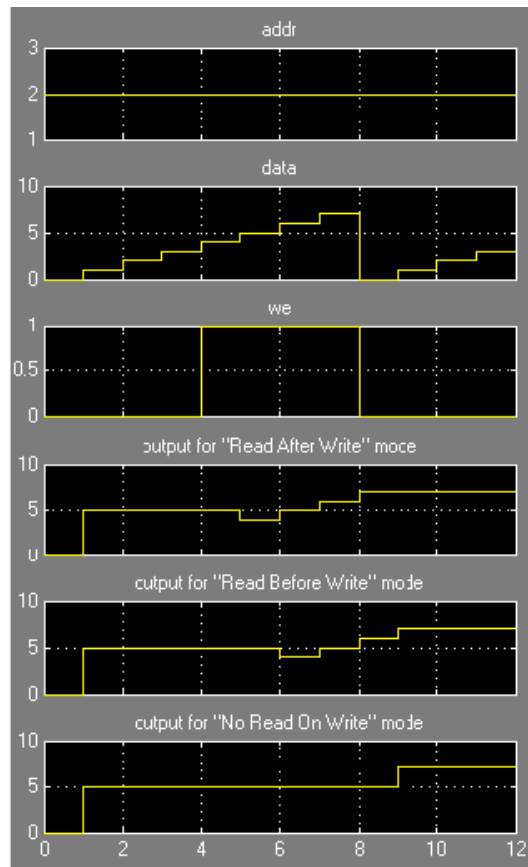


書き込み中の出力は、書き込みモードによって異なります。we ポートが 0 の場合、出力ポートはアドレス ラインで指定したロケーションの値になります。書き込み中 (we がアサートされている間)、入力データ ポートのデータはポートのアドレス入力で選択したメモリ ロケーションに格納されます。書き込みサイクル中は、データ出力ポートのビヘイビアは次のいずれかに設定できます。

- Read After Write
- Read Before Write
- No Read On Write

次の図では、書き込みモードの詳細を示しています。次の図では、メモリの初期値は 5 に、アドレス ビットは 2 に指定されています。[No read on write] モードを使用した場合、出力はアドレス ラインの影響を受けず、we が 0 のときの最後の出力と同じになります。we が 1 の場合は、次に we が 1 になるまで dout が前の値を保持します。図では、dout は、we が 1 に設定された 1 サイクル後に addr の 2 の値を反映しています。

ほかの 2 つのモードにすると、出力はアドレス ラインで指定したロケーションから取得されるので、ロケーションの値が書き込まれます。この場合、出力は書き込み前の値 ([Read before write] モード) になります。



ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- **[Shared memory name]** : Shared Memory ブロックの名前を付けます。同じ名前のメモリはすべて同じ物理的メモリを共有します。
- **[Depth]** : メモリ ブロックのワード数を指定します。ワード サイズは、din データ ポートのビット幅から推論されます。
- **[Ownership and initialization]** : メモリが [Locally owned and initialized] か [Owned and initialized elsewhere] かを指定します。[Locally owned and initialized] の場合は、[Initial value vector] が設定できるようになります。[Locally owned and initialized] の場合、ブロックでメモリのインスタンスが作成され、[Owned and initialized elsewhere] の場合、ブロックが既に作成されたメモリ インスタンスに接続されます。この結果、シミュレーション中に 2 つの Shared Memory ブロックが 2 つの異なるモデルで使用される場合、[Locally owned and initialized] のブロックを含むモデルが最初に開始される必要があります。

- **[Initial value vector]** : 初期メモリ コンテンツを指定します。初期値ベクタ エLEMENTの容量と精度は、**din** を駆動するデータ サンプルの型から推論されます。ベクタ長が **RAM** よりも大きい場合、このベクタの後に続くELEMENTは削除されます。**RAM** がベクタよりも長い場合、この **RAM** の後続ワードは **0** に設定されます。初期値ベクタは、**RAM** のデータ ポート **din** に指定された精度に従ってサチュレートされるか、丸められます。
- **[Access protection]** : **[Lockable]** か **[Unprotected]** のいずれかを指定します。保護なしのメモリにすると、読み出しまたは書き込み中に制限はありませんが、ロックされると、ブロックはメモリにアクセスが許可 (**grant**) されたときのみ書き込めるようになります。**grant** ポートから **1** が出力されると、メモリへのアクセスが許可され、書き込みリクエストが送られます。
- **[Access mode]** : メモリをデザインでどのように使用するか指定します。**[Read and write]** をオンにすると、ブロックには **din** ポートと **dout** ポートが付き、**[Read only]** をオンにすると、メモリの読み出しアクセス用の **dout** ポートが付き、**[Write only]** をオンにすると、メモリの書き込みアクセス用の **din** ポートが付き。
- **[Write mode]** : メモリを **[Read after write]**、**[Read before write]**、または **[No read on write]** に指定します。これらのモードは、デバイスによって使用できるものとできないものがあります。
- **[Memory access timeout (sec)]** : メモリがハードウェアで 사용되는場合に、リクエストに反応するまでの最大待機時間を指定します。
- **[Latency]** : **1** または **2** に設定します。

このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

このブロックでは、ザイリンクス LogiCORE Dual Port Block Memory Generator コア v2.6 が使用されます。

関連項目

詳細は、Shared Memory ブロックの使用方法与詳細は、次の説明を参照してください。

[Multiple Subsystem Generator](#)

[共有レジスタの協調シミュレーション](#)

Shared Memory Read

このブロックは、[Xilinx Blockset] の [Shared Memory] および [Index] ライブラリにリストされています。



ザイリンクスの Shared Memory Read ブロックは、ザイリンクスの共有メモリ オブジェクトからデータを読み出すための高速インターフェイスです。このブロックでは、FIFO オブジェクトとロック可能な共有メモリ オブジェクトの両方がサポートされます。

リクエストされたデータは、共有メモリから読み出され、Simulink のスカラ型、ベクタ型、マトリックス型の信号で出力ポートに読み込まれます。ブロックの下に括弧付きテキストは、このブロック インターフェイスと共有されるメモリを示します。ブロックに表示される深さと幅から、共有メモリのサイズがわかります。これらの値は、ブロックが共有メモリ オブジェクトに接続されると、ランタイム中にアップデートされます。

Shared Memory Read ブロックは、シミュレーション中に起動され、接続された共有メモリ オブジェクトと共に複数のトランザクションを実行します。ブロックが起動される周波数は、[Sample time] で設定します。実行されるトランザクションは、ブロックが FIFO に接続されているか、ロック可能な共有メモリ オブジェクトに接続されているかによって異なります。

FIFO のトランザクション

共有 FIFO オブジェクトのトランザクションは、シミュレーション サイクル中に次の順序で発生します。

- データ待機：Shared Memory Read ブロックは、[Output dimensions] で指定したワード数が共有 FIFO オブジェクトで使用可能になるのを待ちます。ワード数が 15 秒後も FIFO で使用可能にならない場合は、時間切れになり、シミュレーションは停止されます。
- データ読み出し：十分なワード数が使用可能になると、Shared Memory Read は共有 FIFO オブジェクトからデータを読み出します。

ロック可能なメモリのトランザクション

ロック可能なメモリのトランザクションは、シミュレーション サイクル中に次の順序で発生します。

- ロックの獲得：ブロックが共有メモリのコンテンツを読み出す前に、共有メモリ オブジェクトのロックを獲得する必要があります。15 秒後もロックが獲得できない場合は、時間切れになり、シミュレーションは停止されます。
- データ読み出し：ロックが獲得されると、Shared Memory Read ブロックは共有メモリ オブジェクトからデータを読み出します。
- ロックの解除：Shared Memory Read ブロックは共有メモリ オブジェクトからデータを読み出すと、ロックを解除します。

Shared Memory Read ブロックは、シミュレーションでのみ使用され、ネットリストでは無視されます。Shared Memory Read ブロックは、特に高スループット要件のあるハードウェア協調シミュレーション デザインに使用されます。詳細は、「[ハードウェア協調シミュレーションを使用したリアルタイム信号処理](#)」を参照してください。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- **[Shared memory name]** : データが読み出されるザイリンクスの共有メモリ オブジェクトの名前を指定できます。この共有メモリは、ほかで作成され、初期化される共有 **FIFO** かロック可能メモリにする必要があります (**Shared Memory Read** ブロックは、指定した共有メモリ オブジェクトを作成しません)。
- **[Type]** : ザイリンクス共有 **FIFO** かロック可能メモリ オブジェクトのどちらから読み出すか指定します。
- **[Sample time]** : 共有メモリから読み出しを実行する頻度を指定します。

[Output Type] タブ

[Output Type] タブからは、次のようなパラメータを設定できます。

- **[Data type]** : **Shared Memory Read** ブロックで共有メモリのデータ ワードがどのように解釈されるかを指定します。生成されるデータ型には、**Simulink** のスカラ型、ベクタ型、マトリックス型信号を選択できます。サポートされるデータ型は、**int8**、**uint8**、**int16**、**uint16**、**int32**、**uint32** です。選択したデータ型の幅は、共有メモリ オブジェクトに格納されたデータの幅と同じにする必要があります。たとえば、共有メモリのデータ幅が 16 ビットの場合は、**int16** または **uint16** を選択できます。
- **[Output dimensions]** : 使用可能な次元のサイズを入力することで、共有メモリのデータ イメージがどのように解釈されるかを指定します。ベクタ型出力の場合、一次元 (**N**) を、マトリックス型出力の場合、2次元配列 [**M**, **N**] を指定する必要があります。**M** は行数、**N** は列数を表します。出力エレメントの合計 (**N**、または **M*N**) は、共有メモリの深さを超えないようにしてください。
- **[Use frame-based output]** : **Shared Memory Read** ブロックからの出力信号をフレーム ベースの信号にするか、サンプル ベースの信号にするかを指定します。フレーム ベースの信号とは、連続したサンプル ベースの信号がバッファでまとめられたものです。たとえば、フレーム ベースの出力は **Simulink** の **Unbuffer** ブロックを駆動するために使用されます。フレーム ベースの出力をイネーブルにするには、[**Output dimensions**] で出力を 2次元に指定する必要があります。

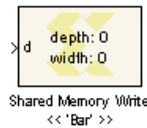
関連項目

[Shared Memory Write](#)

[ハードウェア協調シミュレーションを使用したリアルタイム信号処理](#)

Shared Memory Write

このブロックは、[Xilinx Blockset] の [Shared Memory] および [Index] ライブラリにリストされています。



ザイリンクスの Shared Memory Read ブロックは、ザイリンクスの共有メモリ オブジェクトにデータを書き込むための高速インターフェイスです。このブロックでは、FIFO オブジェクトとロック可能な共有メモリ オブジェクトの両方がサポートされます。

Shared Memory Write ブロックの入力ポートは、共有メモリ オブジェクトに書き込むデータを含む Simulink のスカラ型、ベクタ型、マトリックス型の信号で駆動します。ブロックの下括弧付きテキストは、このブロック インターフェイスと共有されるメモリを示します。ブロックに表示される深さと幅から、共有メモリのサイズがわかります。これらの値は、ブロックが共有メモリ オブジェクトに接続されると、ランタイム中にアップデートされます。入力データの幅は、共有メモリの幅と同じにする必要があります。入力のエレメントの合計は、共有メモリの深さを超えないようにしてください。

Shared Memory Write ブロックは、シミュレーション中に起動され、接続された共有メモリ オブジェクトと共に複数のトランザクションを実行します。ブロックが起動される周波数は、サンプル周期で決まります。サンプル周期は、入力ポートを駆動する信号から取得されます。実行されるトランザクションは、ブロックが FIFO に接続されているか、ロック可能な共有メモリ オブジェクトに接続されているかによって異なります。

FIFO のトランザクション

共有 FIFO オブジェクトのトランザクションは、シミュレーション サイクル中に次の順序で発生します。

- 使用可能ストレージの待機：Shared Memory Write ブロックは、共有 FIFO オブジェクトのストレージが使用可能になるのを待ちます。ストレージの容量は、データ入力ポートを駆動する信号のサイズ（ワード数）によって異なります。たとえば入力信号が 256 ワード幅の場合、Shared Memory Write ブロックは 256 ワードが共有 FIFO で使用できるようになるまで待ちます。ストレージが 15 秒後も使用可能にならない場合は、時間切れになり、シミュレーションは停止されます。
- データ書き込み：十分なワード数が使用可能になると、Shared Memory Write は共有 FIFO オブジェクトにデータを書き込みます。

ロック可能なメモリのトランザクション

- ロックの獲得：ブロックが共有メモリのコンテンツを書き込む前に、共有メモリ オブジェクトのロックを獲得する必要があります。15 秒後もロックが獲得できない場合は、時間切れになり、シミュレーションは停止されます。
- データ書き込み：ロックが獲得されると、Shared Memory Write ブロックは共有メモリ オブジェクトにデータを書き込みます。
- ロックの解除：Shared Memory Write ブロックは共有メモリ オブジェクトにデータを書き込むと、ロックを解除します。

Shared Memory Write ブロックは、シミュレーションでのみ使用され、ネットリストでは無視されます。Shared Memory Write ブロックは、特に高スループット要件のあるハードウェア協調シミュレーション デザインに使用されます。詳細は、「[ハードウェア協調シミュレーションを使用したリアルタイム信号処理](#)」を参照してください。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

Shared Memory Write ブロック特有のパラメータは、次のとおりです。

[Shared memory name]: データが書き込まれる共有メモリ オブジェクトの名前を指定できます。このメモリは、ほかで作成され、初期化されるロック可能メモリにする必要があります (**Shared Memory Write** ブロックは、指定した共有メモリ オブジェクトを作成しません)。

[Type]: ザイリンクス共有 **FIFO** かロック可能メモリ オブジェクト のどちらに書き込むか指定します。

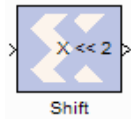
関連項目

[Shared Memory Read](#)

[ハードウェア協調シミュレーションを使用したリアルタイム信号処理](#)

Shift

このブロックは、[Xilinx Blockset] の [Control Logic]、[Data Types]、[Math]、[Index] ライブラリにリストされています。



ザイリンクスの **Shift** ブロックは、入力信号を左または右にシフトします。出力には、入力と同じ固定小数点のコンテナが含まれます。

ブロック パラメータ

Shift ブロック特有のパラメータは、次のとおりです。

- **[Shift direction]** : 左か右かを指定します。**[Right]** をオンにすると、入力が符号拡張され、コンテナ内の **LSB** 方向に移動されます。シフトされたビットがコンテナからはみ出す場合は、それが削除されます。**[Left]** をオンにすると、**LSB** に **0** がパディングされてコンテナ内の **MSB** 方向に移動されます。シフトされたビットがコンテナからはみ出す場合は、それが削除されます。
- **[Number of bits]** : シフトするビット数を指定します。負の値にすると、**[Shift direction]** で選択した方向が逆になります。

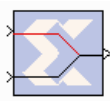
このブロックで使用するその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

このブロックでは、ザイリンクス LogiCORE は使用されません。

Simulation Multiplexer

このブロックは、[Xilinx Blockset] の [Index] ライブラリにのみリストされています。

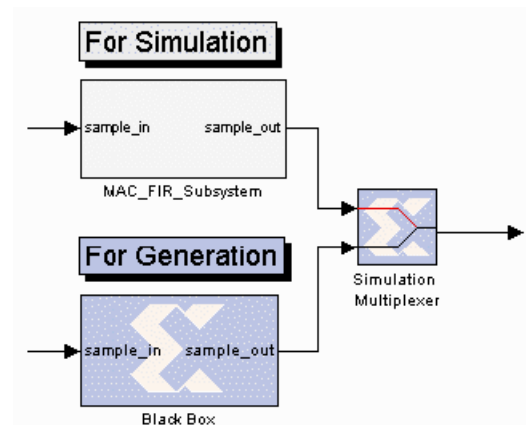


Simulation Multiplexer ブロックは、System Generator で廃止されています。

このブロックは、今後ザイリンクス ブロックセットから削除される予定です。このブロックの機能は、Simulink のコンフィギャブル サブシステムから使用できます。コンフィギャブル サブシステムには、Simulation Multiplexer ブロックを使用するよりも多くの利点があります。

Simulation Multiplexer は、デザインの 2 つの部分を実行させることができる System Generator ブロックで、最初の部分からのシミュレーション結果と 2 つ目の部分からのハードウェアが使用されます。

これは、たとえば、サブシステムが Simulink ブロックを使用して通常どおりに定義されたのに、ハードウェアではサブシステムをインプリメントするのにブラックボックス HDL が使用される場合などに便利です。次の図は、その例です。



シミュレーションにサブシステム、ハードウェアにブラックボックスを使用する場合

Simulation Multiplexer ブロックには、入力ポートが 2 つ含まれます。ブロックのパラメータダイアログ ボックスでは、1 つは For simulation で、もう 1 つが For generation になっています。For simulation ポートを駆動するデザイン部分は、シミュレーション モデルとして使用され、For generation ポートを駆動するデザイン部分は、ハードウェアを生成するために使用されます。同じポートを両方の目的に使用することもできます。この場合、組み合わせポート For simulation/For generation で駆動されるデザイン部分がシミュレーションとハードウェア生成の両方に使用され、残りの部分は無視されます。また、Simulation Multiplexer ブロックを含むデザインからのシミュレーション結果には、ビット精度およびサイクル精度は必要ありません。

Simulation Multiplexer は、シミュレーションとハードウェアで使用するものが違うときに便利なブロックです。たとえば、FPGA ビットストリームを含むハードウェア協調シミュレーション トークンは、シミュレーションできますが、ハードウェアには変換できません。ビットストリームを生成するために使用した HDL が使用できる場合は、ブラックボックスにその HDL が含まれます。Simulation Multiplexer の For simulation ポートをトークンで駆動し、For generation ポートをブラックボックスで駆動すると、デザインのシミュレーションとハードウェアの生成の両方ができるようになります。また、Simulation Multiplexer ブロックは、異なるタイプの HDL を含むブラックボックス同士を切り

替えるためにも使用します。ビヘイビア レベルの HDL をシミュレーションに使用することもできれば、RTL レベルの HDL をインプリメンテーションに使用することもできます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

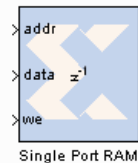
このブロックのパラメータを次に示します。

[For simulation, pass through data from input port] : 1 か 2 のどちらの入力ポートをシミュレーションで使用するか指定します。

[For generation, pass through data from input port] : 1 か 2 のどちらの入力ポートを生成で使用するか指定します。

Single Port RAM

このブロックは、[Xilinx Blockset] の [Control Logic]、[Memory]、[Index] ライブラリにリストされています。



ザイリンクスの Single Port RAM ブロックは、データ入力ポートとデータ出力ポートが 1 つずつ付いたランダム アクセス メモリ (RAM) をインプリメントします。

ブロック インターフェイス

このブロックには、出力ポートが 1 つとアドレス用、入力データ用、書き込みイネーブル (WE) 用の入力ポートが 3 つ含まれます。値はワードで格納され、すべてのワードが同じ演算タイプ、幅、2 進小数点になります。

Single Port RAM ブロックは、ブロック メモリか分散メモリ リソースのいずれかを使用して FPGA にインプリメントできます。データ ワードはそれぞれ 1 つのアドレスに関連付けられ、アドレスは 0 ～ d-1 の範囲の符号なしの整数にする必要があります。d は、RAM の深さ (RAM のワード数) を表します。メモリの最後を読み飛ばすと、シミュレーションでエラーになりますが、ブロック メモリをインプリメントする場合は、指定したアドレス範囲を超えていても、ハードウェアで読み出すことができます。初期の RAM コンテンツは、ブロック パラメータから指定できます。

書き込みイネーブル信号はブール型にする必要があります。データ入力の値は、この書き込みイネーブル ポートが 1 の場合にアドレス入力で指定したメモリ ロケーションに書き込まれます。書き込み中の出力は、インプリメントするメモリによって異なります。

出力ポートのビヘイビアは、選択した書き込みモードによって異なります。WE ポートが 0 の場合、出力ポートはアドレス ラインで指定したロケーションの値になります。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。

このブロックのパラメータを次に示します。

- **[Depth]** : メモリに格納できるワード数を指定します。正の値しか指定できません。
- **[Initial value vector]** : メモリの初期コンテンツを指定します。ベクタ長がメモリの深さよりも大きい場合、深さよりも大きいインデックスが付いた値が無視されます。深さがベクタ長よりも大きい場合は、ベクタ長よりも大きい値のアドレスのメモリ ロケーションが 0 に初期化されます。初期値は、データ ポートで指定された精度に従って、必要であればサチュレートされ、丸められます。
- **[Write Mode]** : WE がアサートされたときのメモリ ビヘイビアを指定します。サポートされるモードは [Read before write]、[Read after write]、[No read on write] です。[Read before write] にすると、出力値には書き込み前のメモリのステートが反映されます。[Read after write] にすると、出力値には書き込み後のメモリのステートが反映されます。[No read on write] にすると、出力値は、アドレスやメモリのステートが変わっても同じ値のままになります。これらのモードは、デバイスによって使用できるものとできないものがあります。詳細は、このセクションの「書き込みモード」および「ハードウェアの注意点」を参照してください。
- **[Memory Type]** : ブロック RAM か分散 RAM を選択します。

- [Provide reset port for output register] : ブロック RAM の出力レジスタを制御するリセットポートが使用されます。メモリ コンテンツはこのポートでは初期値にリセットされません。リセット ポートは、ブロック RAM のレイテンシが 1 に設定されている場合にのみ使用できます。
- [Initial value for output register] : 出力レジスタの初期値を指定します。初期値は、必要に応じて、RAM のデータ ポート B に指定された精度に従ってサチュレートされ、丸められます。

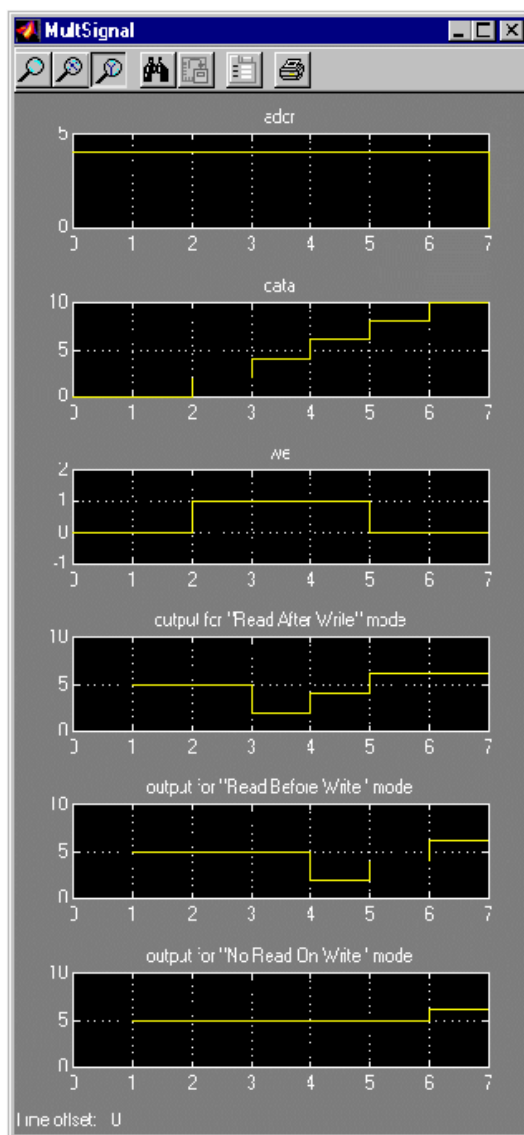
このブロックで使用するその他のパラメータは、この章の最初に説明されています。

書き込みモード

書き込み中 (WE がアサートされている間)、データ入力ポートのデータはポートのアドレス入力で選択したメモリ ロケーションに格納されます。書き込み中のデータ出力ポート **A** のビヘイビアは次のいずれかに設定できます。

- Read after write
- Read before write
- No read On write

次の図では、これらの書き込みモードの詳細を示しています。この図では、メモリの初期値は 5 に、アドレス ビットは 4 に指定されています。[No read on write] モードを使用した場合、出力はアドレス ラインの影響を受けず、出力は WE が 0 のときの最後の出力と同じになります。ほかの 2 つのモードにすると、出力はアドレス ラインで指定したロケーションから取得されるので、ロケーションの値が書き込まれます。つまり、出力は書き込み前の値 ([Read before write] モード) か書き込み後の新しい値 ([Read after write] モード) のどちらかになります。



ハードウェアの注意点

LogiCORE の分散メモリでサポートされるのは、[Read before write] モードのみです。Single Port RAM ブロックでは、指定したレイテンシが 0 より大きい場合、書き込みモードを [Read after write] に設定した分散メモリもサポートされます。ただし、分散メモリで [Read after write] モードにするには、余分なハードウェア リソース (書き込み中にデータをラッチするために、分散メモリの出力に MUX を付けるなど) が必要になります。

Virtex-4、Virtex-5、Virtex-6、Spartan-6、Spartan3A DSP デバイスにシングル ポート ROM ブロックをインプリメントする場合は、次の設定をしておく、最速のタイミング パフォーマンスを達成できます。

- [Provide reset port for output register] をオフにします。
- [Depth] には、16,384 未満の値を指定します。
- [Latency] は 2 またはそれ以上の値に設定します。

ザイリンクス LogiCORE

このブロックでは、常にザイリンクス LogiCORE の Single Port Block Memory または Distributed Memory が使用されます。

ブロック メモリの場合、アドレス幅は $\text{ceil}(\log_2(d))$ と同じにする必要があります。d は、メモリの深さを示しています。このブロック メモリの最大のデータ ワード幅は、指定した深さになり、最大の深さは、ターゲットにしたデバイス ファミリによって異なります。次の表は、ブロック メモリの深さ別に、最大データ ワード幅をそれぞれ示しています。

深さ別の最大データ幅 (Virtex/Virtex-E/Spartan-3)

深さ	幅
2 ~ 2048	256
2049 ~ 4096	192
4097 ~ 8192	96
8193 ~ 16K	48
16K+1 ~ 32K	24
32K+1 ~ 64K	12
64K+1 ~ 128K	6
128K+1 ~ 256K	3

深さ別の最大データ幅 (Virtex-4/Virtex-5/Spartan-3A DSP)

深さ	幅
2 ~ 8192	256
8193 ~ 16K	192
16K+1 ~ 32K	96
32K+1 ~ 64K	48
64K+1 ~ 128K	24
128K+1 ~ 256K	12
256K+1 ~ 512K	6
512K+1 ~ 1024K	3

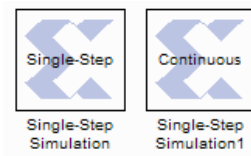
分散メモリのパラメータを選択した場合は、LogiCORE Distributed Memory が使用されます。深さは、Spartan-3、Virtex-4、Virtex-5、Virtex-6、Spartan-6、Spartan-3A DSP デバイスの場合は 16 ~ 65536、その他の FPGA ファミリの場合は 16 ~ 4096 の範囲で指定する必要があります。ワード幅は、1 ~ 1024 の範囲で指定する必要があります。

このブロックでは、次のザイリンクス LogiCORE コアが使用されます。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6 -1L
Single Port RAM	Block Memory Generator	V3.3	•	•	•	•	•	•	•	•	•	•
	Distributed Memory Generator	V4.3	•	•	•	•	•	•	•	•	•	•

Single-Step Simulation

このブロックは、[Xilinx Blockset] の [Tools] および [Index] ライブラリにリストされています。



ザイリンクスの **Single-Step Simulation** ブロックは、シングル ステップ モードの場合にクロック サイクルごとにシミュレーションを一時停止 します。

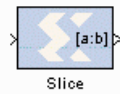
アイコンをダブルクリックすると、シングル ステップから連続モードに 切り替わります。シミュレーションが一時停止した場合は、ツールバー の [Start] ボタン (▶) をクリックすると再開されます。

ブロック パラメータ

このブロックにはパラメータがありません。

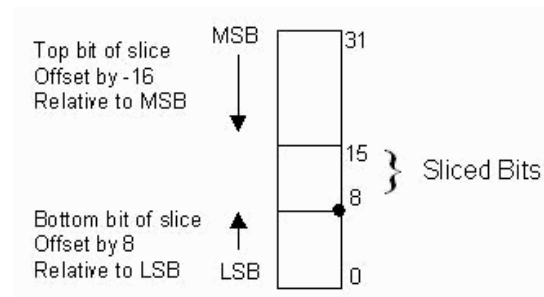
Slice

このブロックは、[Xilinx Blockset] の [Basic Elements]、[Control Logic]、[Data Types]、[Index] ライブラリにリストされています。



ザイリンクスの **Slice** ブロックを使用すると、入力データからビットのシーケンスを切り取り、新しいデータ値を作成できます。このデータ値がブロックから出力されます。出力データ型は、2 進小数点が 0 の位置の符合なしになります。

このブロックでは、ビットのシーケンスを指定するのにいくつかの手法が提供されています。パラメータ化の際に入力のタイプがわかっている場合は、これらの手法を使用することで機能的な利点はありませんが、入力データの幅および 2 進小数点の位置が変化するようなデザインでは、これらの手法が有益になります。たとえば、入力の最上位ビットのみ、整数ビットのみ、または小数点部分の上位 3 ビットのみを抽出するよう設定できます。次の図は、入力の上位 16 ビットと下位 8 ビットを除いてすべてを抽出するところを示しています。



ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

このブロックでは、次のパラメータが設定できます。

- **[Width of slice (number of bits)]** : 抽出するビット数を指定します。
- **[Boolean output]** : シングルビット スライスのデータ型をブール型にするかどうか指定します。
- **[Specify range as]** : **[Two bit locations]**、**[Upper bit location + width]**、**[Lower bit location + width]** のいずれかをオンにします。スライスの両端のエンドポイントのビット ロケーションを指定するか、片方のエンドポイント + ビット数を使用して指定できます。
- **[Offset of top bit]** : **LSB**、**MSB**、または 2 進小数点からその範囲最後のビットまでに使用するオフセットを指定します。
- **[Offset of bottom bit]** : **LSB**、**MSB**、または 2 進小数点からその範囲最後のビットまでに使用するオフセットを指定します。
- **[Relative to]** : ビット スライスの位置を **MSB** (最上位ビット)、**LSB** (最下位ビット)、スライスの上部または下部の 2 進小数点のいずれかが基準になるように指定します。

このブロックで使用されるその他のパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

System Generator

このブロックは、[Xilinx Blockset] の [Basic Elements]、[Tools]、[Index] ライブラリにリストされています。

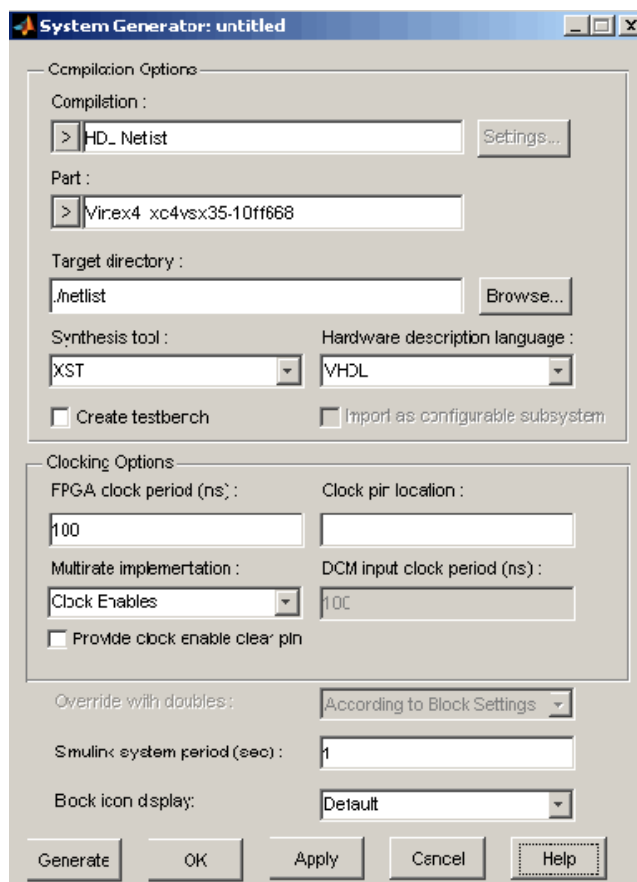


System Generator ブロックは、システム制御およびシミュレーション パラメータを提供し、コード ジェネレータを起動するために使用されます。System Generator ブロックは、デザインにおける特異な役割りのため、System Generator トークンと呼ばれることもあります。ザイリンクス ブロックセットからのエレメントを含む Simulink モデルには、System Generator ブロック (トークン) が最低 1 つは含まれます。System Generator ブロックをモデルに追加すると、コードの生成およびシミュレーションの処理方法を指定できるようになります。

System Generator ブロックの使用方法は、「[System Generator トークンを使用したコンパイルとシミュレーション](#)」を参照してください。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、Simulink モデル内のアイコンをダブルクリックすると表示されます。



このブロックでは、次のパラメータが設定できます。

コンパイル オプション

- [Compilation] : コード ジェネレータが起動されたときに出力されるコンパイル結果のタイプを指定します。詳細は、「[System Generator のコンパイル タイプ](#)」を参照してください。
- [Part] : 使用する FPGA デバイスを指定します。
- [Target directory] : コンパイル結果を保存するディレクトリを指定します。System Generator および FPGA インプリメンテーション ツールでは多数のファイルが生成されるので、個別のディレクトリ (Simulink モデル ファイルが含まれるディレクトリとは別のディレクトリ) を作成することをお勧めします。
- [Synthesis tool] : デザインの合成に使用するツールを指定します。Synplicity 社の Synplify Pro または Synplify、およびザイリンクスの XST を選択できます。
- [Hardware description language] : デザインのコンパイルに使用する HDL 言語を指定します。[VHDL] または [Verilog] を選択できます。
- [Create testbench] : HDL テストベンチを作成するよう指定します。HDL シミュレータでテストベンチをシミュレーションし、コンパイルされたデザインのシミュレーション結果を Simulink シミュレーション結果と比較します。System Generator では、デザインを Simulink でシミュレーションし、Gateway ブロックで検出される値を保存することにより、テスト ベクタを作成します。 テストベンチの最上位 HDL ファイルの名前は、<name>_testbench.vhd/.v となります。<name> はテストするデザイン部分名になります。

メモ：このオプションは、Shared Memory ブロックがデザインに含まれている場合はサポートされません。

クロック オプション

- [FPGA clock period (ns)] : システム クロックの周期を ns で指定します。値は整数である必要はありません。ここで指定した周期は、制約ファイルでグローバル PERIOD 制約として設定され、ザイリンクス インプリメンテーション ツールに渡されます。複数サイクル パスは、この値の整数倍で制約されます。
- [Clock pin location] : ハードウェア クロックのピン ロケーションを指定します。この情報は、制約ファイルを介してザイリンクス インプリメンテーション ツールに渡されます。System Generator デザインをより大規模な HDL デザインの一部として含める場合は、このオプションは指定しないでください。
- [Multirate implementation] :
 - ◆ [Clock Enables] (デフォルト) : マルチレート デザインを駆動するクロック イネーブル ジェネレータ回路が作成されます。
 - ◆ [Hybrid DCM-CE] : Virtex-4 および Virtex-5 で最高 3 つまで、Spartan-3A DSP で 2 つまでのクロック ポートをさまざまなレートで駆動できる DCM を含んだクロック ラップが作成されます。DCM 出力ポートへのレートのマッピングは、CLK0 > CLK2x > CLKdv > CLKfx の順で優先されます。デザインに含まれるクロックが DCM で処理可能なクロックよりも多い場合、残りのクロックはクロック イネーブル コンフィギュレーションでサポートされます。

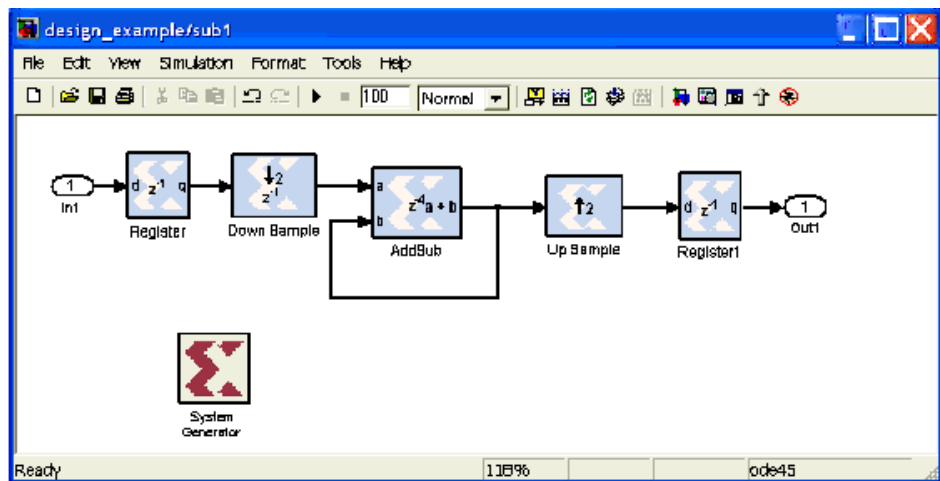
reset 入力ポートが DCM クロック ラップにあると、DCM をリセットできるようになり、locked 出力ポートがあると、外部デザインで入力データを 1 つの clk 入力ピンに同期させやすくなります。
 - ◆ [Expose Clock Ports] : System Generator デザインの最上位レベルに複数のクロック ポートを含めることで、デザイン外部からの同期クロック入力を複数使用できるようになります。

詳細は、「[タイミングとクロック](#)」を参照してください。

- [DCM input clock period(ns)] : DCM 入力クロック周期が [FPGA clock period(ns)] オプション (システム クロック) と異なる場合に指定します。これで FPGA クロック周期 (システム クロック) がこのハードウェア定義の入力から派生するようになります。
- [Provide clock enable clear pin] : 最上位レベルのクロック ラップに `ce_clr` ポートを付けるかどうか指定します。`ce_clr` 信号は、クロック イネーブル生成ロジックをリセットするために使用されます。クロック イネーブル生成ロジックをリセットできるようにすると、ダイナミックな制御が可能になり、データ パス サンプリングの開始が指定できます。詳細は、「[自動生成されたクロック イネーブル ロジックのリセット](#)」を参照してください。

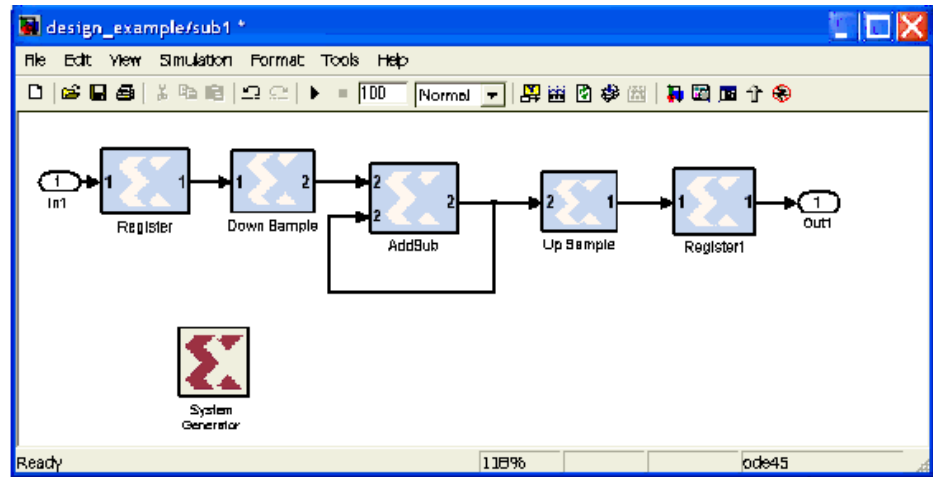
その他のオプション

- [Simulink system period (sec)] Simulink のシステム周期を秒単位で指定します。Simulink のシステム周期は、モデルのサンプル周期の最大公約数になります。サンプル周期は、ブロックのダイアログ ボックスで設定するか、Simulink の伝搬規則に従って決められるか、このオプションを使用したブロックのハードウェア オーバーサンプリング レートを基に算出されます。ハードウェア オーバーサンプリング レートを基に算出された場合、実際のサンプル時間は Simulink のブロックの観測可能なシミュレーション サンプル時間よりも速くなります。ハードウェアでは、オーバーサンプリング レートが 1 より大きいブロックは、入力をデータよりも速いレートで処理されます。たとえば、オーバーサンプリング レートが 8 の配列型乗算器は、Simulink ではその乗算器ブロックの実際のサンプル時間の 1/8 のサンプル周期になります。このパラメータは、マスタ ブロックでのみ変更できます。
- [Block icon display] : ブロックのアイコンに表示する情報の種類を指定します。ブロックのアイコンは、デザインがコンパイルされてから、選択された表示オプションでアップデートされます。表示オプションは、次から選択できます。
 - ◆ [Default] : デフォルト表示になります。ブロックのデフォルト アイコンは、xbsIndex ライブラリから使用されます。

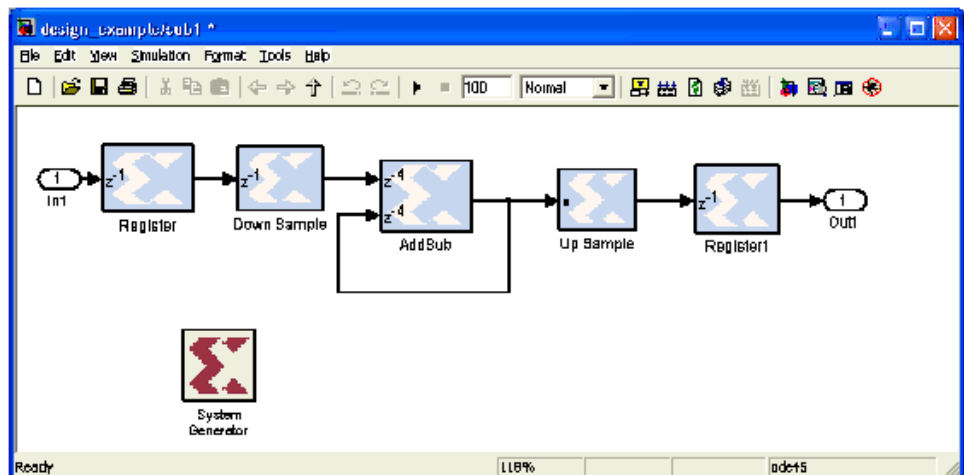


- ◆ [Normalized Sample Periods] : ブロックのすべての入力ポートおよび出力ポートの標準化されたサンプル周期が表示されます。たとえば、Simulink のシステム周期が 4 に設定される場合、ブロック ポートに伝搬されるサンプル周期は 4 です。この場合、表示される標準

化されたサンプル周期は 1 になります。ブロック ポートに伝搬されるサンプル周期が 8 の場合は、表示は 2 になります。数値が大きいほど、レートが遅いことを示します。

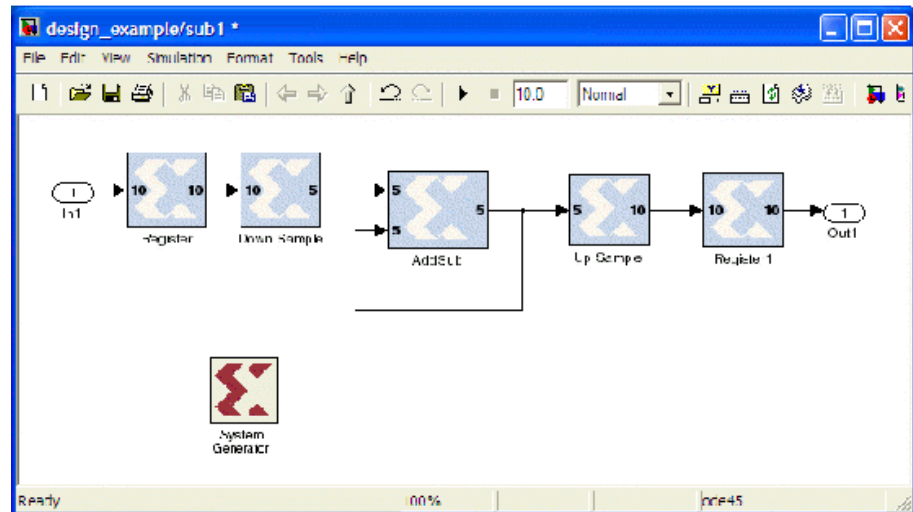


- ◆ [Pipeline stages] : ブロックの入力ポートからのレイテンシ情報が表示されます。表示されたパイプライン段数は、FFT、RS Encoder/Decoder、Viterbi Decoder などの一部のハイレベルなブロックでは正確でないことがあります。この場合、表示されるパイプライン段数は、ブロックに入力から出力への組み合わせパスを含めるかどうかを決定するために使用できます。たとえば、次の図の Up Sample ブロックは、入力から出力に組み合わせパスが含まれることを示しています。

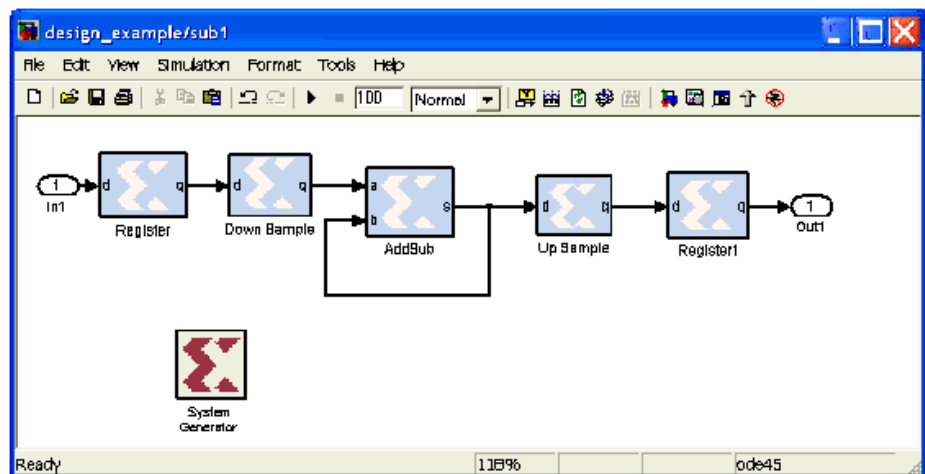


- ◆ [Sample frequencies (MHz)] : ブロックのすべての入力ポートおよび出力ポートのサンプル周波数が表示されます。 周波数は、ポートの標準化されたサンプル周期と System

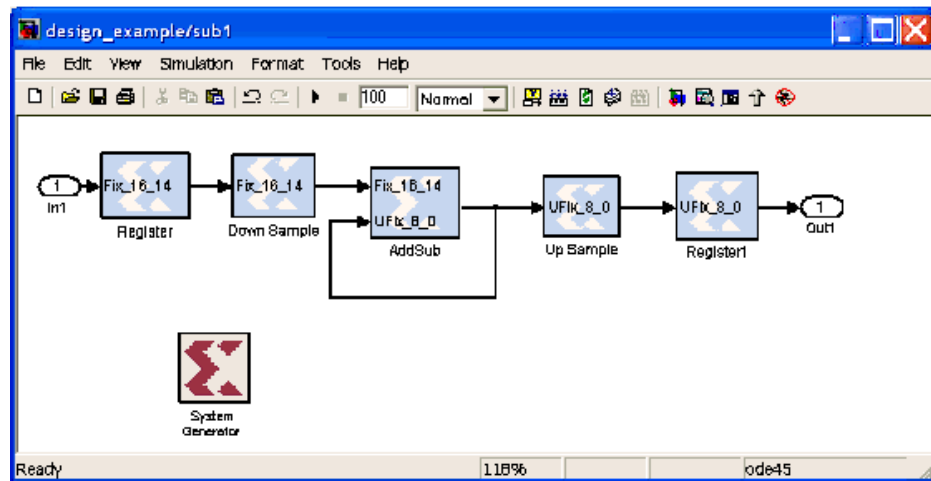
Generator ブロックで指定された FPGA クロック周期を乗算すると算出できます。サンプル周波数の単位は MHz です。



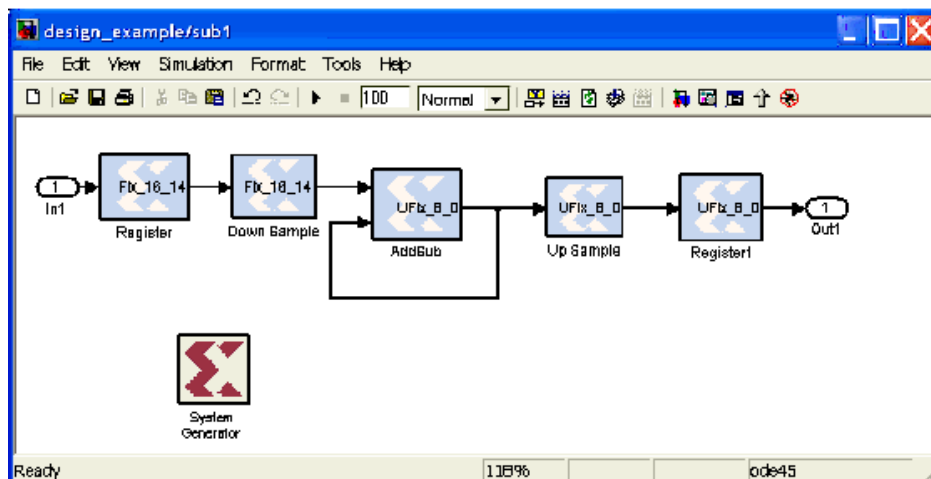
- ◆ [HDL port names] : ブロックのネットリスト エンティティに対応する HDL の入力ポート名と出力ポート名を表示します。



- ◆ [Input data types] : ブロックの入力ポートを駆動する信号のデータ型を表示します。



- ◆ [Output data types] : ブロックの出力ポートのデータ型を表示します。



Threshold

このブロックは、[Xilinx Blockset] の [Math] および [Index] ライブラリにリストされています。



ザイリンクスの **Threshold** ブロックでは、入力数の符号がテストされます。入力数が負の場合、ブロックの出力は -1 になり、それ以外の場合は出力は 1 になります。出力は 2 ビットの長さの符号付き固定小数点整数です。ブロックには、入力と出力が 1 つずつ含まれます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

このブロックで使用するパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

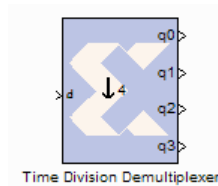
ブロック パラメータでは出力のデータ型は制御されません。これは、出力が常に 2 ビット長の符号付き固定小数点整数であるからです。

ザイリンクス LogiCORE

このブロックでは、ザイリンクス **LogiCORE** は使用されません。

Time Division Demultiplexer

このブロックは、[Xilinx Blockset] の [Basic Elements] ライブラリと [Index] ライブラリにリストされています。



ザイリンクスの Time Division Demultiplexer ブロックは、シリアル入力をそれより遅いレート複数の出力にします。

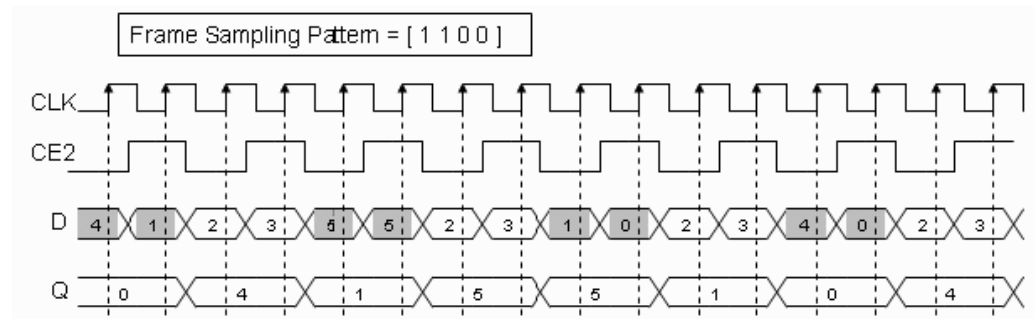
ブロック インターフェイス

このブロックには、1 つのデータ入力ポートと複数 (1 ~ 32) のユーザーが設定できるデータ出力が含まれます。データ出力ポートの演算タイプと精度は入力データポートと同じになります。また、このブロックには、入力有効ポート (vin) と出力有効ポート (vout) もオプションで含まれます。どちらの有効ポートもブール型です。ブロックのインプリメンテーションは、単一チャネルか複数チャネルのいずれかにできます。

単一チャネルのインプリメンテーション

単一チャネルのインプリメンテーションの場合、Time Division Demultiplexer ブロックにはデータ入力ポートとデータ出力ポートがそれぞれ 1 つ含まれます。データ有効入力ポートとデータ有効出力ポートもオプションで使用できます。入力データフレームの長さは、フレームサンプリングパターンの長さによって決まります。1 の位置が、ダウンサンプリングされる入力値と 1 の数 (ダウンサンプリング係数) を示します。単一チャネルモードの場合、入力したフレームサンプリングパターンに基づいて、すべての入力データフレームの最初と 2 つ目の入力値がレート 2 で出力されます。

このインプリメンテーションでは、入力フレームのサイズがデータフレームからサンプリングされる値の数で除算されます。入力データフレームの値は、すべてオプションの有効ポートで使用可能にすることもできます。



複数チャネルのインプリメンテーション

複数チャネルのインプリメンテーションの場合、Time Division Demultiplexer ブロックには、1 つのデータ入力ポートと複数の出力ポートが含まれます。出力ポートの数は、フレームサンプリングパターンの 1 の数と同じになります。データ有効入力ポートとデータ有効出力ポートもオプションで使用できます。入力データフレームの長さは、フレームサンプリングパターンの長さによって決まります。1 の位置が、入力値がダウンサンプリングされ、対応するデータチャネルに出力されることを示します。複数チャネルモードの場合、入力したフレームサンプリングパターンに基づいて、

すべての入力データ フレームの最初と 2 つ目の入力値がレート 4 で対応する出力チャンネルに転送されます。

複数チャンネルのインプリメンテーションの場合、ダウンサンプリング係数は常に入力フレームのサイズと同じになります。入力データ フレームの値は、すべてオプションの有効ポートで使用可能にすることもできます。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

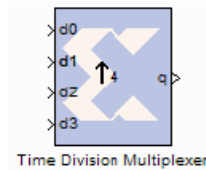
このブロックのパラメータを次に示します。

- **[Frame sampling pattern]** : シリアル入力データ フレームのサイズを指定します。フレームのサンプリング パターンは 1 と 0 のみを含む **MATLAB** ベクタにする必要があります。
- **[Implementation]** : デマルチプレクサのビヘイビアを単一チャンネル モードにするか複数チャンネル モードにするか指定します。各モードのビヘイビアは、前述の説明を参照してください。
- **[Provide valid ports]** : オンにすると、オプションの入力有効ポート (**vin**) と出力有効ポート (**vout**) を含めることができます。**vin** ポートでは、すべての入力データ値をシリアル入力データ フレームの一部として有効にできます。**vout** ポートは、出力ポートが有効か無効かを示します。

このブロックで使用されるパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

Time Division Multiplexer

このブロックは、[Xilinx Blockset] の [Basic Elements] ライブラリと [Index] ライブラリにリストされています。



ザイリンクスの **Time Division Multiplexer** ブロックは、入力ポートの値を 1 つの高速レート of 出力ストリームに多重化します。

ブロック インターフェイス

このブロックには、32 個の入力ポートと 1 つの出力ポートがあります。入力ポートの演算タイプ、精度、レートはすべて同じである必要があります。出力ポートの演算タイプと精度は入力ポートと同じになります。出力レートは **nr** になります (**n** は入力ポート数、**r** は共通レート)。また、このブロックには、入力と出力がそれぞれ有効か無効を示す **vin** ポートと **vout** ポートがオプションで含まれます。どちらの有効ポートもブール型です。

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

このブロックでは、次のパラメータが設定できます。

- [Number of inputs] : 入力のビット数 (2 ~ 32) を指定します。
- [Provide valid port] : オンにすると、オプションの入力有効ポート (**vin**) と出力有効ポート (**vout**) を含めることができます。**vin** ポートは入力の値が無効なことを示し、**vout** ポートは出力フレームが無効なことを示します。

このブロックで使用されるパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

To FIFO

このブロックは、[Xilinx Blockset] の [Index] ライブラリにリストされています。



ザイリンクス To FIFO ブロックは、First-In First-Out (FIFO) のメモリ キューの前半分をインプリメントします。

書き込みイネーブル (we) 入力が 1 になると、このモジュールのデータ ポートの値が次の使用可能な空のメモリ ロケーションに書き込まれます。FIFO がフルの場合は、full 出力ポートがアサートされます。出力ポート percent_full には、フルの FIFO のパーセントがユーザー指定の精度で表示されます。

To FIFO は、FIFO Generator v2.1 コアを使用してハードウェアにインプリメントされます。System Generator のハードウェア協調シミュレーション インターフェイスでは、To FIFO ブロックを FPGA ハードウェアにコンパイルし、協調シミュレーションできます。共有 FIFO を System Generator 協調シミュレーション ハードウェアで使用すると、ホスト PC と FPGA 間でデータを高速に転送でき、リアルタイム ハードウェア協調シミュレーション機能が強化されます。

9.2 リリースからは、同じ名前の From FIFO ブロックと To FIFO ブロックがペアになり、ネットリストで 1 つの BRAM ベースの FIFO ブロックになっています。From FIFO または To FIFO ブロックが別のブロックとペアにならない場合は、その入力ポートと出力ポートが最上位レベルの System Generator デザインに含まれます。ペアになったブロックはデザインの中のどの階層にでも配置できますが、同じ名前の From FIFO または To FIFO ブロックが複数ある場合は、エラーになります。

以前のバージョンとの互換性を保持するには、MATLAB グローバル変数 xISgSharedMemoryStitch を off に設定してください。これには、MATLAB コマンド ラインに次のように入力します。

```
global xISgSharedMemoryStitch;
xISgSharedMemoryStitch = 'off';
```

ブロック パラメータ

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Shared memory name] : 共有 FIFO の名前を付けます。同じ名前の FIFO はすべて同じ物理的 FIFO を共有します。
- [Ownership] : メモリが [Locally owned] か [Owned elsewhere] かを指定します。[Locally owned] の場合、ブロックで FIFO のインスタンスが作成され、[Owned elsewhere] の場合、ブロックが既に作成された FIFO インスタンスに接続されます。
- [Depth] : メモリ ブロックのワード数を指定します。ワード サイズは、din ポートのビット幅から推論されます。
- [Bits of precision to use for %full port] : %full ポートのビット幅を指定します。この符号なし出力の 2 進小数点は、常にワードの最上部にあります。このため、たとえば精度が 1 に設定されると、出力は 0.0 と 0.5 の 2 つの値を取り込みます。この場合、0.5 は FIFO が少なくとも 50% フルであることを示しています。
- [Provide asynchronous reset port] : オプションの非同期のエッジトリガ リセット ポート (rst) が使用されます。11.2 よりも前のリリースでは、このリセットはレベルトリガで、ブロックはこれが High になってもリセット モードのままでした。

このブロックで使用されるパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

このブロックは、次のザイリンクス LogiCORE を使用します。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6 -1L
To FIFO	FIFO Generator	V5.3	•	•	•	•	•	•	•	•	•	•

関連項目

From FIFO ブロックの詳細については、次のトピックを参照してください。

[FIR Compiler 4.0](#)

[Multiple Subsystem Generator](#)

[共有 FIFO の協調シミュレーション](#)

To Register

このブロックは、[Xilinx Blockset] の [Index] ライブラリにリストされています。



ザイリンクスの **To Register** ブロックは、1 サンプル周期のレイテンシを含む D フリップフロップ ベースのレジスタの前半部分をインプリメントします。レジスタは、複数デザインまたは 1 つのデザインの複数セクションで共有できます。

このブロックには、入力ポートが 2 つ含まれます。**din** ポートは、入力データを受け取り、レジスタのビット幅を設定します。初期出力値は、ブロックのパラメータ ダイアログ ボックスで指定できます。イネーブル ポート (**en**) がアサートされると、入力されたデータは、1 サンプル周期後に **dout** から出力されます。**en** がアサートされないと、レジスタに書き込まれた最後の値が **dout** に出力されます。

9.2 リリースからは、同じ名前の **To Register** ブロックと **From Register** ブロックがペアになり、ネットリストで 1 つの **Register** ブロックになっています。**To Register** または **From Register** ブロックが別のブロックとペアにならない場合は、その入力ポートと出力ポートが最上位レベルの **System Generator** デザインに含まれます。ペアになったブロックはデザインのどの階層にでも配置できますが、同じ名前の **To Register** または **From Register** ブロックが複数ある場合は、エラーになります。

以前のバージョンとの互換性を保持するには、MATLAB グローバル変数 `xlSgSharedMemoryStitch` を `off` に設定してください。これには、MATLAB コマンド ラインに次のように入力します。

```
global xlSgSharedMemoryStitch;  
xlSgSharedMemoryStitch = 'off';
```

ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- **[Shared memory name]** : 共有レジスタの名前を付けます。1 つのレジスタに **To Register** ブロックが必ず 1 つ必要です。この名前は、デザインのほかの共有メモリの名前とは異なるものにしてください。
- **[Initial value]** : レジスタの初期値を指定します。
- **[Ownership and initialization]** : メモリが **[Locally owned and initialized]** か **[Owned and initialized elsewhere]** かを指定します。**[Locally owned and initialized]** の場合、ブロックでレジスタのインスタンスが作成され、**[Owned and initialized elsewhere]** の場合、ブロックが既に作成されたレジスタ インスタンスに接続されます。この結果、シミュレーション中に 2 つの共有レジスタ ブロックが 2 つの異なるモデルで使用される場合、**[Locally owned and initialized]** のブロックを含むモデルが最初に開始される必要があります。

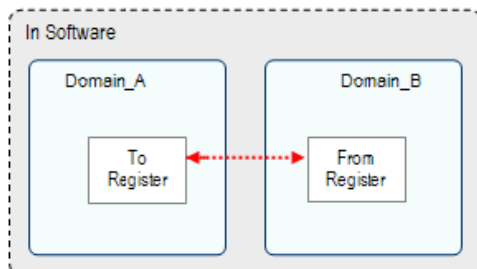
このブロックで使用されるパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

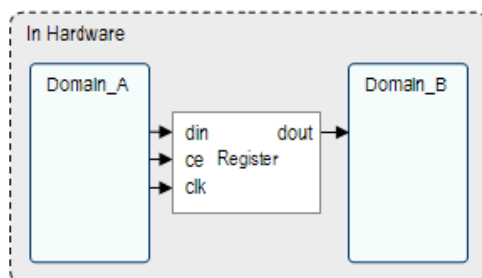
このブロックは、合成可能な **VHDL** モジュールとしてインプリメントされます。このブロックでは、ザイリンクス **LogiCORE** は使用されません。

クロック ドメインの切り替え

To Register と From Register ブロックのペアがクロック ドメインの境界を切り替えるために使用されると、ハードウェアには 1 つのレジスタがインプリメントされます。このレジスタには、To Register ブロックのクロック ドメインが使用されます。たとえば、デザインに Domain_A と Domain_B の 2 つのクロック ドメインがあると、次の図のように、2 つのクロック ドメインを切り替えるために、1 組の共有レジスタが使用されるとします。



Multiple Subsystem Generator ブロックを使用してデザインが生成されると、含まれるレジスタは 1 つだけになります。レジスタのクロック信号とクロック ネーブル信号は Domain_A ドメインから駆動されます。



この方法でクロック ドメインを切り替えると、問題になることがあります。安定させるには、2 つの Register ブロックを From Register ブロックの直後に追加し、データを From Register のクロック ドメインに再同期させます。

関連項目

To Register ブロックの詳細については、次のトピックを参照してください。

[From Register](#)

[Multiple Subsystem Generator](#)

[共有レジスタの協調シミュレーション](#)

Toolbar

このブロックは、[Xilinx Blockset] の [Tools] および [Index] ライブラリにリストされています。

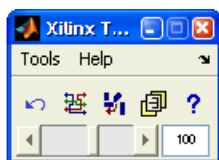


ザイリンクスの **Toolbar** ブロックを使用すると、**System Generator** の複数の便利なユーティリティに素早くアクセスできます。**Toolbar** ブロックは、**Simulink** の拡大/縮小表示を簡単にし、新しい自動レイアウトや **Simulink** モデルへの配線機能を追加します。

また、次に説明するようなツールも含まれます。

ブロック インターフェイス

Toolbar ブロックをダブルクリックすると、次のグラフィック ユーザー インターフェイスが起動されます。





これは、コマンド ラインで **xlTBUtils** (**Toolbar** ブロックで使用される関数集合) を使用しても起動できます。

```
xlTBUtils('Toolbar');
```

このインターフェイスは、1 度に 1 つしか開くことができません。**Toolbar** ブロックは、配置箇所には関係なく、常に作業中の **Simulink** モデルに対してのみアクションを実行します。たとえば、モデル **A** から **Toolbar** ブロックを起動した場合でも、モデル **B** で作業中であれば、このブロックはモデル **B** に対して使用できます。

ツールバー ボタン

ツールバー ボタン	説明
	[Undo] : Toolbar で実行したモデル レイアウトへの一番最近の変更を取り消すことができ、レイアウトをその変更の前の状態に戻します。最大で 3 つ前までの変更を取り消すことができます。
	[Reroute] : ラインを配線し直して、モデルのリーダビリティを向上します。選択したラインのみを配線し直すことができます。選択しないと、モデル内のラインがすべて配線し直されます。
	[Auto layout] : ブロックを配置し直し、ラインを配線し直して、モデルのリーダビリティを向上します。
	[Add terminators] : xlAddTerms 関数を呼び出して、作業中モデルにソースとシンクを追加します。 System Generator ブロックには System Generator の Constant ブロックと同じ、 Simulink ブロックには Simulink の Constant ブロックと同じソースが追加されます。ターミネータがシンクとして使用されます。

ツールバー ボタン	説明
	[Help] : このドキュメントを開きます。
	[Zoom] : スライドを動かすか、[Zoom Factor] の値を指定して、Simulink モデルを拡大表示するか、縮小表示します。スライドを左右に動かすか、[Zoom Factor] の値を変えてください。[Zoom Factor] には、5 ～ 1000 までの値を入力できます。

ツールバー メニュー

ツールバー ボタン	説明
Tools	
Create Plugins	System Generator Board Description Builder ツールを起動します。
Inspect Selected	Simulink Inspector を開きます。これには、選択しているブロックのプロパティが含まれます。このツールは、複数ブロックのサイズやブロックの水平方向の配置を設定する場合に使用すると便利です。
Toolbar Properties	<p>次の図のような [プロパティ] ダイアログ ボックスが開きます。[Auto layout] および [Reroute] のパラメータを設定できます。[Layout X pitch] と [Layout Y pitch] は、配置されたブロック間の X 軸方向と Y 軸方向の距離をそれぞれ表します。</p> <p>[Use simulink autorouter] がオンになっていると、Toolbar ブロックで Simulink の autorouter が使用されます。オフの場合は、ソースからディスティネーションへダイレクト ラインがひかれます。</p> 
Help	このドキュメントを開きます。

リファレンス

- 1) E.R.Gansner, E.Koutsofios, S.C.North, KVo, 『A Technique for Drawing Directed Graphs』
<http://www.graphviz.org/Documentation/TSE93.pdf>
- 2) [Reroute] および [Auto layout] ボタンをクリックすると、Graphviz というオープン ソース パッケージが起動されます。このパッケージの詳細は、<http://www.graphviz.org/> を参照してください。

関連項目

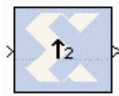
[xlAddTerms](#)

[xlSBDBuilder](#)

[xlTBUtils](#)

Up Sample

このブロックは、[Xilinx Blockset] の [Basic Elements] ライブラリと [Index] ライブラリにリストされています。

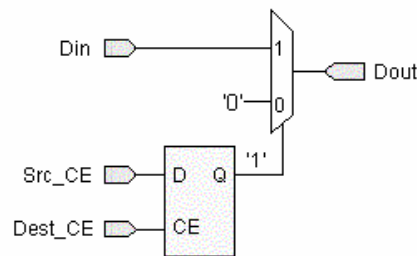


Up Sample

ザイリンクスの Up Sample ブロックは、ブロックが配置された箇所のサンプル レートを増加するために使用します。出力サンプル周期は、 $1/n$ になります (1 が入力サンプル周期、 n はサンプリング レート)。

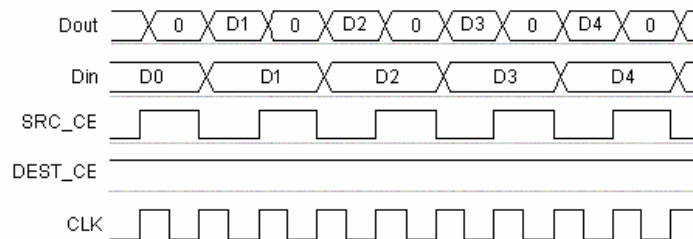
入力信号は、アップサンプリングされます。これにより、入力サンプル フレーム内で、1 入力サンプルは n 回出力されるか (サンプルがコピーされた場合)、0 の数 ($n-1$) 個で 1 回出力されます (0 がパディングされた場合)。

ハードウェアには、Up Sample ブロックは 2 つのいずれかの方法でインプリメントされます。ブロックのパラメータ ダイアログ ボックスで [Copy samples] をオンにすると、Din ポートが直接 Dout に接続され、ハードウェアは拡張されません。0 がパディングされるようにすると、入力サンプルと挿入された 0 を切り替えるのに MUX が使用されます。次の図は、0 がパディングされた Up Sample ブロックの回路図を示しています。



ブロック インターフェイス

このブロックは、Src_CE と Dest_CE の 2 つのクロック イネーブル信号を受信します。Src_CE は入力データ ストリーム レートに対応したクロック イネーブル信号で、Dest_CE は出力データ ストリーム レートに対応したより高速なクロック イネーブル信号です。回路図では、MUX にフリップフロップを 1 つ追加しています。フリップフロップを付けると Src_CE のタイミングが調整できるので、MUX は入力サンプル周期の開始時にデータ入力サンプルに切り替え、最初の入力サンプル後に定数 0 に切り替えます。回路図には、Din から Dout までの組み合わせパスが含まれているので、0 をパディングするように設定された Up Sample ブロックの後には、可能であれば、レジスタを付けるようにしてください。



ブロック パラメータ

ブロック パラメータのダイアログ ボックスは、**Simulink** モデル内のアイコンをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- **[Sampling rate (number of output samples per input sample)]** : 2 以上の整数にする必要があります。これは、入力に対する出力サンプル周期の比率で、サンプル レートを乗算する値になります。たとえば、2 と指定した場合は、入力サンプル レートが 2 倍になります。整数以外の比率を指定する場合は、**Down Sample** ブロックと **Up Sample** ブロックを組み合わせ使用してください。
- **[Copy samples (otherwise zeros are inserted)]** : 増加されたクロック レートによって生成される追加のサンプルをどうするか指定します。オンにすると、その追加のサンプル時間で同じサンプルがコピーされます。オフにすると、追加のサンプルは 0 になります。

オプション ポート

- ◆ **[Provide enable port]** : オンにすると、レイテンシが 0 より大きい正の整数に指定された場合に、**en** (イネーブル) 入力ポートが追加されます。
- **[Latency]** : ブロックの出力を遅延させるサンプル周期が定義できるようになっています。1 サンプル周期は **FPGA** インプリメンテーションの複数クロック サイクルに相当することがあります (ハードウェアが **Simulink** モデルに対してオーバークロックになる場合など)。ユーザー定義のサンプル レイテンシは、入力サンプル レートでクロック イネーブルされるシフト レジスタを配置することで、**Upsample** ブロックで処理されます。レイテンシが 0 以外の **Upsample** ブロックのビヘイビアは、レイテンシ 0 の **Upsample** ブロックの入力に同等のレイテンシを含む遅延ブロックを配置した場合と同様になります。

このブロックで使用するパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

Viterbi Decoder 6_1

このブロックは、[Xilinx Blockset] の [Communication] および [Index] ライブラリにリストされています。



たたみ込みエンコーダでエンコードされたデータは、ザイリンクスの Viterbi Decoder ブロックを使用してデコードできます。

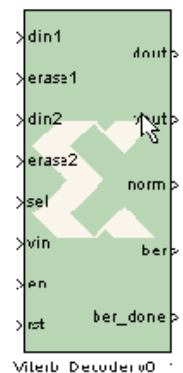
デコード プロセスには、2 段階あります。最初の段階では、すべてのデータ入力の組み合わせに対して入力データのコストを計算します。コストの決定には、ハミング手法またはユークリッド手法を使用できます。2 つ目の段階では、トレリス線図をトレースバックし、最適なパスを決定します。トレリス線図のトレース長は、[Traceback length] パラメータで指定できます。

このデコーダでは、最適なたたみ込み符号を使用した場合、エラー レートが最小になります。エンコードするたたみ込み符号は、デコードするたたみ込み符号と同じにする必要があります。

制約長	1/2 レート (8 進数) の場合 の最適なたたみ込み符号	1/3 レート (8 進数) の場合の 最適なたたみ込み符号
3	[7 5]	[7 7 5]
4	[17 13]	[17 13 15]
5	[37 33]	[37 33 25]
6	[57 65]	[57 65 71]
7	[117 127]	[117 127 155]
8	[357 233]	[357 233 251]
9	[755 633]	[755 633 447]

このブロックは、Spartan-3A DSP のほか、以前からサポートされていた Virtex-4、Virtex-5、Spartan-3、Spartan-3A/3AN、Spartan-3E でもサポートされます。

ブロック インターフェイス



Viterbi Decoder ブロックでは、1/2 ～ 1/7 のレートがサポートされ、din1 ～ din7 という名前の入力ポートが 2 ～ 7 個表示されます。ハード コーディングするには、各データ入力を 1 ビット幅にする必要があります。ソフト コーディングの場合は、ビット幅は 3 ～ 8 ビットにできます。vin ポートは、din ポートの値が有効であることを示します。外部パンクチャリングを使用する場合は、デコーダ

のレートによって、最大で 7 個の erase ポートが使用できるようになります。erase ピンが High になると、対応するデータ ピンがゼロ記号として処理されます。制約長とトレースバック長を指定すると、ブロックをデュアルデコーダ (たたみ込み符号 2 つで、出力レート 2 つ) として使用できます。sel という入力ポートでは、たたみ込み符号がどちらの入力データを使用するか指定されます。sel が 0 の場合、データは [Convolution code array 1 (octal)] を使用してデコードされ、1 の場合は [Convolution code array 2 (octal)] を使用してデコードされます。

Viterbi Decoder ブロックには、2 ～ 5 個の出力ポートが含まれます。dout ポートは 1 ビットのデコード結果を出力し、vout ポートは値が無効なことを示します。ber ポートはエンコードし直された dout の値と遅延の din の値の差をカウントし、チャンネルのビット エラー レートを出力します。検出されたエラー数は 8 で除算され、ber ポートに出力されます。ber_done ポートは、エラー カウントの入力サンプル数が処理されたかどうかを示します。norm 信号は、ブロック内で標準化が実行されたことを示します。norm ポートでは、チャンネル上のエラーがすぐに監視され、その結果が出力されます。標準化がさらに頻繁になると (norm ポートが High になると)、出力されるエラー レートも高くなります。

ブロック パラメータ

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Constraint length] : $n+1$ になります (n はエンコーダに含まれる制約レジスタ長)。
- [Use dual decoder] : オンにすると、ブロックがデュアル デコーダとして動作し、sel 入力ポートが使用できるようになります。
- [Convolution code array 1 (octal)] : 8 進数のたたみ込み符号の 1 つ目の配列を指定します。出力レートは配列の長さで決まります。2 ～ 7 の符号を指定します。デュアル デコードが使用される場合に sel ポートの値が 0 だと、この配列になります。
- [Convolution code array 2 (octal)] : 8 進数のたたみ込み符号の 2 つ目の配列を指定します。出力レートは配列の長さで決まります。2 ～ 7 の符号を指定します。デュアル デコードが使用される場合に sel ポートの値が 1 だと、この配列になります。この場合、出力レートは 1 つ目の配列からの出力レートと同じにする必要はありません。
- [Traceback length] : Viterbi Decoder のトレリス線図のトレースバック長を指定します。最適な長さは、制約長の 5 ～ 7 倍です。
- [Coding] : [Hard] か [Soft] かを指定します。ハード エンコーディングではハミング手法、ソフト エンコーディングではユークリッド手法を使用して、コストが計算されます。ハード コーディングでは入力データを 1 ビット幅に、ソフト コーディングでは 2 ～ 8 ビット幅にする必要があります。デュアル デコーディング、外部バンクチャリング、シリアル アーキテクチャを使用する場合はソフト コーディングを使用する必要があります。
- [Data format] : [Signed magnitude] または [Offset binary] (ソフト コーディングの場合のみ選択可能) を指定します。
- [Provide bit error rate port] : オンにすると、ber と ber_done ポートがブロックに追加されます。
- [Number of input samples for error count] : 計算されたビット エラー レートに対する入力サンプル数を指定します。
- [Provide normalization port] : オンにすると、norm ポートがブロックに追加されます。

[Advanced] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- [Use external puncturing] : オンにすると、erase ポートがブロックに追加されます。
- [Use best state] : オンにすると、トレースバックが最適なステートから開始されます。
- [Width reduction] : コストを節約して最適な状態を決定するために、何個の LSB を無視するか指定します。

[Implementation] タブ

[Implementation] タブからは、次のようなパラメータを設定できます。

- [Architecture type] : [Parallel] または [Serial] を選択します。
- [Optimization] : [Area] または [Speed] を選択します(パラレルアーキテクチャの場合にのみ指定できます)。
- [Reduce latency] : オンにすると、ブロックのレイテンシが約 50% 減少します。

このブロックで使用するパラメータは、「[ブロックのパラメータダイアログボックスの共通オプション](#)」で説明されています。

ザイリンクス LogiCORE

このブロックでは、次のザイリンクス LogiCORE Viterbi Decoder コアが使用されます。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan				Virtex		
			3、 3E	3A	3A DSP	6	4	5	6
Viterbi Decoder 6_1	Viterbi Decoder	V6.1	•	•	•		•	•	

Viterbi Decoder 7.0

このブロックは、[Xilinx Blockset] の [Communication] および [Index] ライブラリにリストされています。



たたみ込みエンコーダでエンコードされたデータは、ザイリンクスの **Viterbi Decoder** ブロックを使用してデコードできます。

デコード プロセスには、2 段階あります。最初の段階では、すべてのデータ入力の組み合わせに対して入力データのコストを計算します。コストの決定には、ハミング手法またはユークリッド手法を使用できます。2 つ目の段階では、トレリス線図をトレースバックし、最適なパスを決定します。トレリス線図のトレース長は、[Traceback length] パラメータで指定できます。

このデコーダでは、最適なたたみ込み符号を使用した場合、エラー レートが最小になります。エンコードするたたみ込み符号は、デコードするたたみ込み符号と同じにする必要があります。

制約長	1/2 レート (8 進数) の場合 の最適なたたみ込み符号	1/3 レート (8 進数) の場合の 最適なたたみ込み符号
3	[7 5]	[7 7 5]
4	[17 13]	[17 13 15]
5	[37 33]	[37 33 25]
6	57 65]	[57 65 71]
7	[117 127]	[117 127 155]
8	[357 233]	[357 233 251]
9	[755 633]	[755 633 447]

このブロックは、Spartan-3A DSP のほか、以前からサポートされていた Virtex-4、Virtex-5、Spartan-3、Spartan-3A/3AN、Spartan-3E でもサポートされます。

ブロック インターフェイス



Viterbi Decoder ブロックでは、1/2 ～ 1/7 のレートがサポートされ、din1 ～ din7 という名前の入力ポートが 2 ～ 7 個表示されます。ハード コーディングするには、各データ入力を 1 ビット幅にする必要があります。ソフト コーディングの場合は、ビット幅は 3 ～ 8 ビットにできます。vin ポートは、din ポートの値が有効であることを示します。外部パンクチャリングを使用する場合は、デコーダ

のレートによって、最大で 7 個の erase ポートが使用できるようになります。erase ピンが High になると、対応するデータ ピンがゼロ記号として処理されます。制約長とトレースバック長を指定すると、ブロックをデュアルデコーダ (たたみ込み符号 2 つで、出力レート 2 つ) として使用できます。sel という入力ポートでは、たたみ込み符号がどちらの入力データを使用するか指定されます。sel が 0 の場合、データは [Convolution code array 1 (octal)] を使用してデコードされ、1 の場合は [Convolution code array 2 (octal)] を使用してデコードされます。

Viterbi Decoder ブロックには、2 ～ 5 個の出力ポートが含まれます。dout ポートは 1 ビットのデコード結果を出力し、vout ポートは値が無効なことを示します。ber ポートはエンコードし直された dout の値と遅延の din の値の差をカウントし、チャンネルのビット エラー レートを出力します。検出されたエラー数は 8 で除算され、ber ポートに出力されます。ber_done ポートは、エラー カウントの入力サンプル数が処理されたかどうかを示します。norm 信号は、ブロック内で標準化が実行されたことを示します。norm ポートでは、チャンネル上のエラーがすぐに監視され、その結果が出力されます。標準化がさらに頻繁になると (norm ポートが High になると)、出力されるエラー レートも高くなります。

ブロック パラメータ

[Page1] タブ

[Page1] タブからは、次のようなパラメータを設定できます。

Viterbi タイプ

- [Standard] : このタイプは基本的な Viterbi Decoder です。
- [Multi-Channel] : このタイプでは 1 つの Viterbi Decoder を使用して、間引きされた多数のデータ チャンネルがデコードできます。
- [Number of Channels] : デコードされるチャンネル数を 2 ～ 32 のいずれかの値に指定できます。これは、[Multi-Channel] をオンにした場合に使用します。
- [Trellis Mode] : このタイプは、TCM および SECTOR_IN 入力を使用したトレリス モードのデコーダです。
- [Dual Decoder] : オンにすると、2 セットのたたみ込み符号でブロックがデュアル デコーダとして動作し、sel 入力ポートが使用できるようになります。

デコーダ オプション

- [Use Reduced Latency] : ブロックのレイテンシは、トレースバック長と制約長で決まります。このオプションをオンにすると、ブロックは約半分になり、レイテンシはトレースバック長の 2 倍にしかありません。
- [Constraint length] : $n+1$ になります (n はエンコーダに含まれる制約レジスタ長)。
- [Traceback length] : Viterbi Decoder のトレリス線図のトレースバック長を指定します。最適な長さは、制約長の 5 ～ 7 倍です。

[Page2] タブ

アーキテクチャ

- [Parallel] : 大型で高速な Viterbi Decoder です。
- [Serial] : 小型で、入力データを直列方式で処理します。各入力シンボル セットを処理するために必要なクロック サイクル数は、出力レートとデータのソフト幅によって決まります。

ベスト ステート

- **[Use Best State]** : 高度にパンクチャされたデータの BER パフォーマンスを向上させます。
- **[Best State Width]** : コストを節約して最適な状態を決定するために、何個の **LSB** を無視するか指定します。

コーディング

- **[Soft Width]** : ソフト コード データの入力幅は 3 ~ 8 の範囲のいずれかの数値になり、幅が大きいほど、ロジックが必要です。ブロックが直列モードでインプリメントされる場合にソフト幅を大きくすると、直列の処理時間も増加します。
- **[Soft Coding]** : ユークリッド手法を使用して Viterbi トレリスの分岐に対する入力データのコストが計算されます。
- **[Hard Coding]** : 入力データ ビットと Viterbi トレリス分岐間のハミング符号の差異が使用されます。ハード コーディングは、標準的なパラレル ブロックでのみ使用できます。

データ形式

- **[Signed magnitude]** :
- **[Offset Binary]** (ソフト コーディングの場合のみ)

ソフト幅 3 の場合の **[Signed magnitude]** と **[Offset-Binary]** のデータ形式については、該当する LogiCORE 製品仕様の表 1 を参照してください。

デュアル レート デコーダ

制約長とトレースバック長を指定すると、ブロックをデュアル デコーダとして使用できます。たたみ込み符号と出力レートの組み合わせ 2 つがデコーダに対して内部で使用可能です。デュアル デコーダを使用すると、同じ制約帳の 2 つの異なるデコーダが必要な場合に、チップのエリア使用率を大幅に削減できます。次の 2 つのタブでは、このデュアル デコーダ機能のたたみ込み符号を指定できます。

[Page3] タブ

たたみ込み 0

- **[Output Rate 0]** : 2 ~ 7 のいずれかの値を使用できます。
- **[Convolution Code 0 Radix]** : たたみ込み符号は、2 進数、8 進数、10 進数で入力および表示できます。
- **[Convolution Code Array (0-6)]** : たたみ込み符号の 1 つ目の配列を指定します。出力レートは配列長で決まります。2 ~ 7 の符号を指定します。デュアル デコードが使用される場合に **sel** ポートの値が 0 (low) だと、この配列になります。

[Page4] タブ

このタブのオプションは、[Page1] タブの Viterbi タイプにデュアル デコーダを選択した場合にのみ選択できます。

たたみ込み 1

- **[Output Rate 1]** : 2 ~ 7 のいずれかの値を使用できます。これはデコーダがデュアル デコーダの場合に 2 つ目の出力レートになります。入力データは **SEL** 入力が高になるとこのレートでデコードされます。このオプションはデュアル デコーダ以外には使用されません。
- **[Convolution Code 1 Radix]** : たたみ込み符号は、2 進数、8 進数、10 進数で入力および表示できます。

- [Convolution Code Array (0-6)] : たたみ込み符号の 1 つ目の配列を指定します。出力レートは配列長で決まります。2 ~ 7 の符号を指定します。デュアル デコードが使用される場合に sel ポートの値が 1 (high) だと、この配列になります。

[Page5] タブ

パケット オプション

トレリスの初期化

- [None] : トレリスの初期化は実行されません。
- [State Zero] : PACKET_START 信号がアサートされる (High になる) と、トレリスがステート 0 に初期化されます。ステートのコストはすべて ACS モジュールで最大値に初期化されます (ステート 0 を除く)。
- [Equal States] : PACKET_START 信号がアサートされる (High になる) と、トレリス内のすべてのステートが同じ値に初期化されます。
- [User Input] : PACKET_START 信号がアサートされる (High になる) と、トレリスが PS_STATE のステートに初期化されます。ステートのコストはすべて ACS モジュールで最大値に初期化されます (PACKET_START 入力が High の場合に 0 に初期化されるダイナミックな入力ステートを除く)。

ダイレクト トレースバック

トレースバックとパケットの最終ステートの処理方法を指定できます。

- [Maximum Direct] : エンコード ビットの数を直接トレースされるように指定します。範囲は 10 ~ 42 です。
- [None] : ダイレクト トレースバックは実行されません。
- [State Zero] : TB_BLOCK 信号が High になると、入力データがステート 0 からトレーニング シーケンスなしに直接トレースバックされます。
- [User Input] : TB_BLOCK 信号が High になると、入力データが TB_STATE ユーザー入力からトレーニング シーケンスなしに直接トレースバックされます。TB_STATE の値は、TB_BLOCK 信号 High の最後のクロック エッジで選択されます。
- [Best State] : TB_BLOCK 信号が High になると、入力データがベスト ステートからトレーニング シーケンスなしに直接トレースバックされます。ベスト ステートは ACS モジュールのコストを基にデコーダに対して内部で生成されます。

[Page6] タブ

パンクチャ

- [None] : 入力データはパンクチャされません。
- [External (Erased Symbols)] : オンにすると、erase ポートがブロックに追加されます。NULL シンボル (チャネル間送信の前に削除されたシンボル) があると、消去入力の erase が使用されていることを示します。

BER オプション

- [Use BER Symbol Count] : このビット エラー レート (BER) オプションを使用すると、送信チャネルのエラー レートが監視されます。
- [Number of BER Symbols] : エラー カウントが行われる入力シンボル数を指定します。

[Page7] タブ

同期化オプション

- [Use Synchronization] : 同期されないことを示す出力 (OUT_OF_SYNC) が必要な場合、オンにします。
- [Use Dynamic Thresholds] : オンにすると、同期入力バスである NORM_THRESH および BER_THRESH がブロックに追加されます。これらの 16 ビットの入力バスは BER (ビット エラー) しきい値と NORM (正規化) しきい値にそれぞれ該当し、同期化を評価するためのしきい値がダイナミックに修正できます。

スタティックなしきい値

- [BER Thresh] : これは同期化を評価するためのプリセットしきい値です。正規化のしきい値が取得される前にビット エラー カウントがこのしきい値に到達すると、ブロックは同期されず、OUT_OF_SYNC 出力がアサートされます。
- [Norm Thresh] : これは同期化を評価するためのプリセットしきい値です。ビット エラー カウントのしきい値が取得される前に正規化カウントがこのしきい値に到達すると、ブロックは同期され、OUT_OF_SYNC 出力がディアサートされます。

[Page8] タブ

オプションのピン

- CE : クロック イネーブル - コア クロック イネーブル (アクティブ High)。この信号が High になると、デコーダは入力データを通常どおり処理します。Low の場合は、デコーダはデータの処理を停止し、その状態を保ちます。
- RDY : DATA_OUT 出力ポートの有効なデータを示します。この出力は、直列接続の場合に必須になります。
- SCLR : 同期クリア - 同期リセット (アクティブ High)。SCLR を CLK と同時にアサートすると、デコーダの内部ステートがリセットされます。
- NORM : Add Compare Select モジュールに対して内部で正規化が行われたかどうかを示します。
- [Block Valid] : BLOCK_IN および BLOCK_OUT 信号が必要な場合、オンにします。これらの信号は、デコーダを介したデータのブロックの動きを監視します。BLOCK_IN に対する BLOCK_OUT は、デコーダのレイテンシ分遅く出力されます。

このブロックで使用されるパラメータは、「[ブロックのパラメータ ダイアログ ボックスの共通オプション](#)」で説明されています。

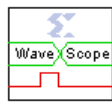
ザイリンクス LogiCORE

このブロックでは、次のザイリンクス LogiCORE Viterbi Decoder コアが使用されます。

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/データシート	Spartan				Virtex		
			3、3E	3A	3A DSP	6	4	5	6
Viterbi Decoder 7.0	Viterbi Decoder	V7.0	•	•	•	•	•	•	•

WaveScope

このブロックは、[Xilinx Blockset] の [Tools] および [Index] ライブラリにリストされています。

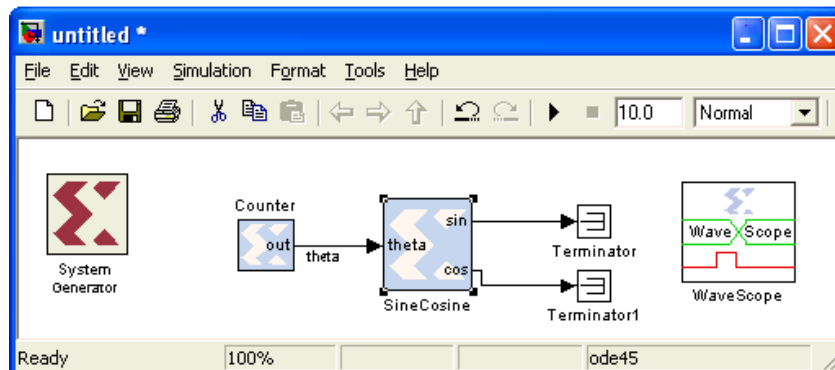


System Generator の WaveScope ブロックは、高度な機能を備えた使いやすい波形ビューアで、System Generator デザインの解析およびデバッグに使用します。

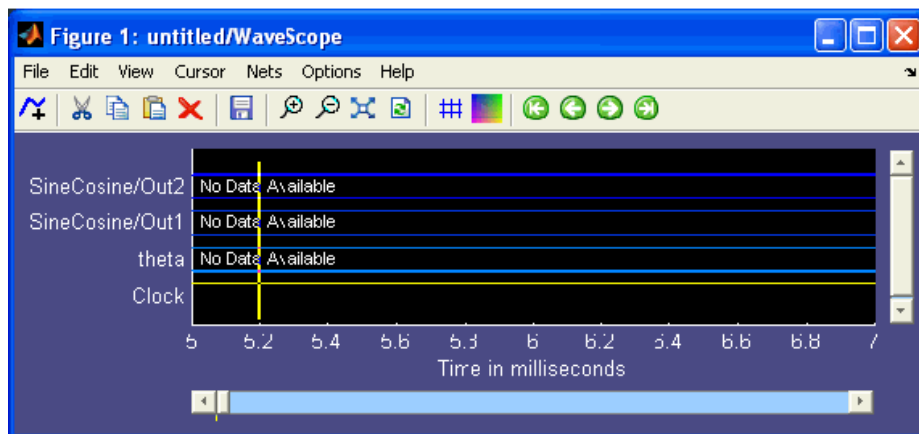
このビューアでは、シミュレーションが完了した後のすべてのワイヤの時間変化を確認できます。信号はロジックまたはアナログ形式でフォーマットできるほか、2 進数、16 進数、10 進数などでも表示できます。

クイック チュートリアル

次に、単純な例を使って、WaveScope の使用方法を説明します。

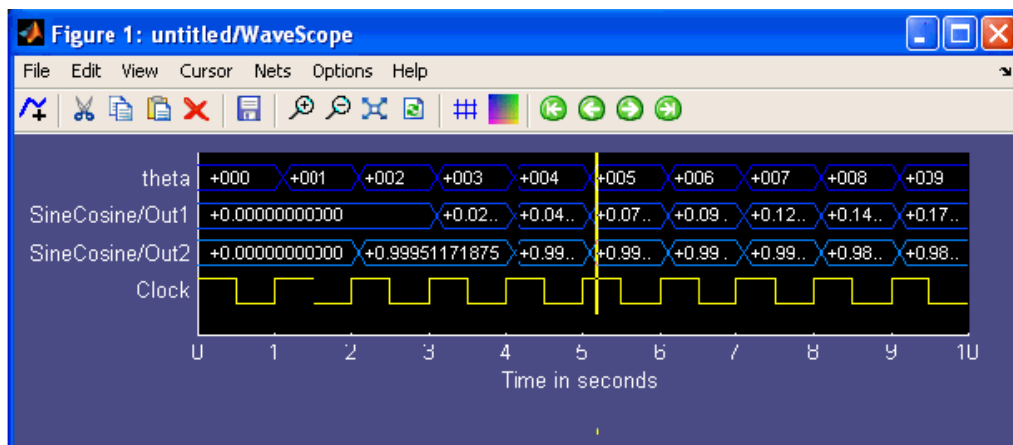


WaveScope ブロックをモデルにドロップし、ダブルクリックすると、何も表示されていない波形ビューアが起動されます。3 つのワイヤを **Shift** キーを押しながら選択し、波形ビューアで [Add Selected Nets] ボタン (+) をクリックして、これらのワイヤをビューアに追加します。波形ビューアは、次のようになります。

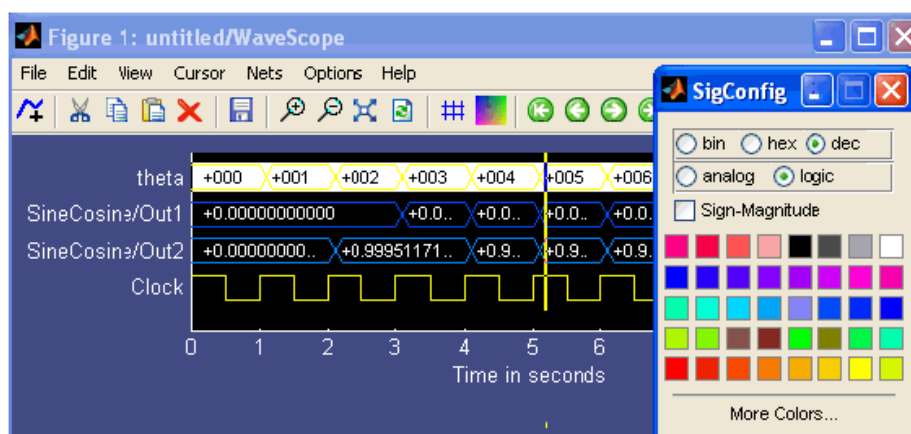


ビューアには、3 つの信号が表示されています。このうち 2 つには、モデル内で命名していなかったため、自動的に名前が付いています。モデルのウィンドウに戻り [シミュレーションの開始] ボタン (▶) をクリックします。このシミュレーションの周期は 1 秒で、10 秒間実行されます。波形ビュー

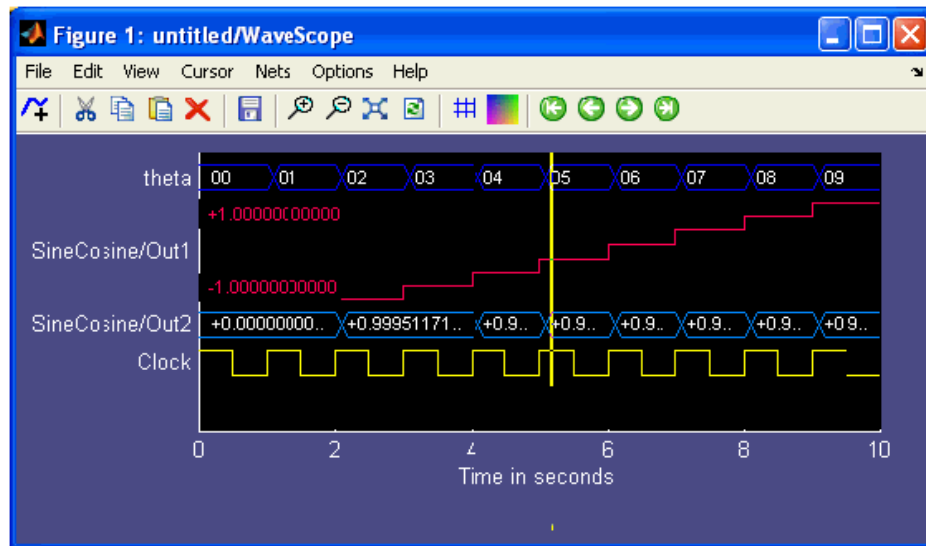
アは自動的にアップデートされます。🔍 ボタンをクリックし、図のように波形全体が表示されるようにしてください。



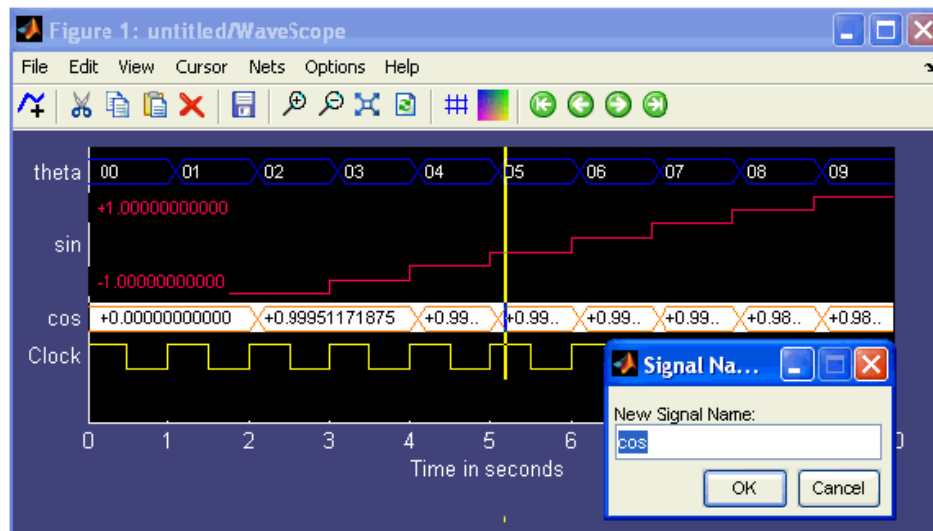
theta 信号の基数は 16 進数に変更できます。theta という信号名をクリックするか、その信号波形をクリックしてハイライトし、そのハイライトされた信号をダブルクリックし (名前の方はダブルクリックしません)、次のフォーマット ダイアログ ボックスを起動します。




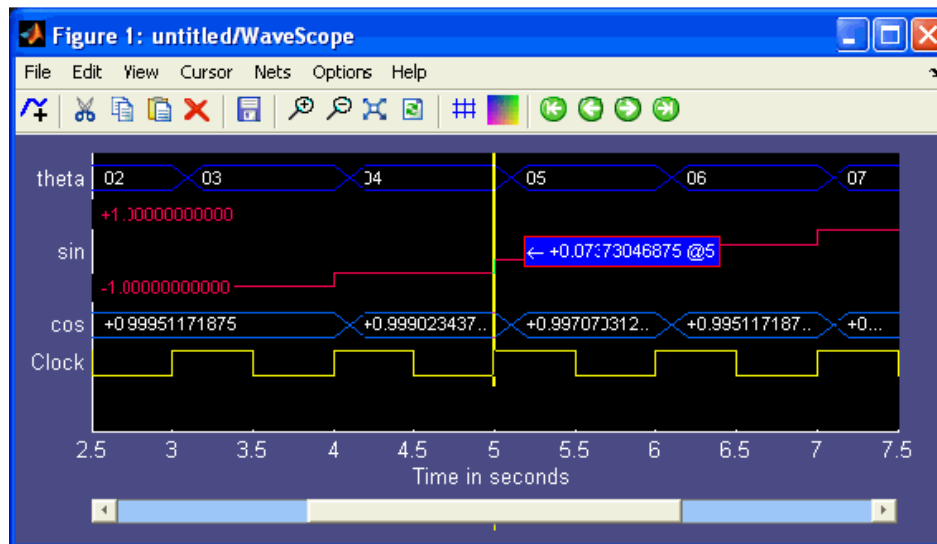
[hex] をオンにし、theta のフォーマットを 16 進数に変更します。同様に、SineCosine/Out1 信号を [analog] に変更し、色を赤に変更してください。



信号名をダブルクリックして、テキスト ボックスに名前を入力して、信号名を変更してください。



新しい信号名がモデルに表示されます。 ボタンを使用すると、シミュレーションの一部を拡大できます。**[Cursor] → [Center Cursor]** をクリックし、黄色のカーソルを画面の中央に配置し、マウスポインタをそのカーソルに合わせ、表示される信号の値を確認してください。



ブロック インターフェイス


WaveScope のアイコンをダブルクリックすると、波形ビューアが開きます。このビューアを閉じていても、シミュレーションの最後に自動的に起動されます。このビューアを使用すると、シミュレーション結果をさまざまな方法で確認できます。

WaveScope では、指定したネットの信号が表示されます。この信号はさまざまな方法で同時に確認できます。たとえば、ロジック形式またはアナログ形式の両方で確認したりできます。信号はロジックまたはアナログ形式でフォーマットできるほか、2 進数、16 進数、10 進数などでも表示できます。一番下には、参考のためにクロック信号が表示されます。

WaveScope のウィンドウは、次のように使用できます。

- どのネットの信号を表示するか選択します。
- 信号の表示を設定します。
- 信号を表示します。

ネットの選択

WaveScope のウィンドウで確認するネットを選択するには、次の 2 通りの方法があります。1 つ目は、Simulink のウィンドウでザイリンクス ブロックの出力ネットを選択し、ツールバーの **[Add Selected Nets]** ボタン () をクリックします。複数のブロック/ネットを選択する場合は、Shift キーを押しながらクリックします。選択したネットの信号が WaveScope のウィンドウに表示されます。選択したブロックの入力と出力が、すべてこのウィンドウに追加されます。モデルがシミュレーションされるまでは、WaveScope に表示できるデータはありません。データは、シミュレーション後に表示されます。

ツールバーで **[Add Selected Nets]** ボタンを何度かクリックすると、WaveScope のウィンドウにその信号がその回数分表示されます。

2 つ目の選択方法では、[Nets] メニューを使用します。このメニューには、モデル内のブロックとネットの階層リストが含まれます。図が複雑な場合は、このメニューを使用した方が特定のネットを選択しやすくなります。

一番下には、デザイン内で一番速いレートのクロック信号が表示されます。このクロック信号は、どの信号を表示しても表示されます。

信号の選択と移動

信号または対応するネット名は、信号をクリックすると選択できます。信号を選択すると、別の場所にドラッグして移動できます。複数の信号を 1 度を選択する場合は、ネット名を **Shift** キーまたは **Ctrl** キーを使用します。信号をクリックしても選択できません。

複数の信号を選択して移動させると、すべてが 1 度に移動されます。この方法は、複数の関連する信号を一度に表示されるようにする場合に便利です。

WaveScope の画面上でクロック信号を選択して移動することはできません。クロック信号は、常に一番下に表示されます。

信号の削除

WaveScope のウィンドウに信号を追加した後に削除するには、その信号を選択し、ツールバーの [Delete Signals] ボタンをクリックします。キーボードの **Delete** キーを押したり、[編集] → [Delete] をクリックしても削除できます。

また、カット、コピー、貼り付けなどの基本的な操作もできます。これらは、ツールバーの [Copy Signals] ボタンや [Paste Signals] ボタンを使用できるほか、キーボード ショートカット (カットの場合は **Ctrl** キー + **X**、コピーの場合は **Ctrl** キー + **C**、貼り付けの場合は **Ctrl** キー + **V**) を使用したり、[編集] メニューからそれぞれのコマンドをクリックしても実行できます。

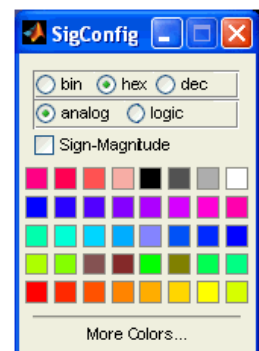
WaveScope の画面上でクロック信号をコピー、貼り付け、削除することはできません。

信号表示の設定

信号によっては、何もしなくても数値が重要な場合は数値で表示されたり、ステートの遷移が重要な場合はロジック ステートで表示されたりすることがあります。WaveScope を使用すると、表示方法を選択できます。

まず信号を選択し、ダブルクリックします (信号名ではなく、信号自体をダブルクリックします)。図のようなダイアログ ボックスが表示されます。

- フォーマット - トランザクションを強調したロジック信号として信号を表示させる場合は [logic] をオンにします。この値は各トランザクション後に書き込まれます。信号の値をグラフのように表示させる場合は [analog] をオンにします。信号の High と Low の値も、グラフの左側に表示されます。アナログ信号のサイズは、選択したアナログ信号の一番下をドラッグすると変更できます。
- 基数 - [bin]、[hex]、[dec] から選択できます。値は、最適な基数点で表示されます。たとえば、10 進数の 10.5 は 16 進数では A.8 と表示されます。
- 符号絶対値 - [Sign-Magnitude] をオンにすると、WaveScope で値が 2 の補数値ではなく、符号絶対値として解釈されます。このフォーマットでは、値は常に 10 進数で表示されます。
- カラー - WaveScope ではデフォルトの色が選択されています。カラー ボタンで色を選ぶと、選択した信号すべてにその色が反映されます。

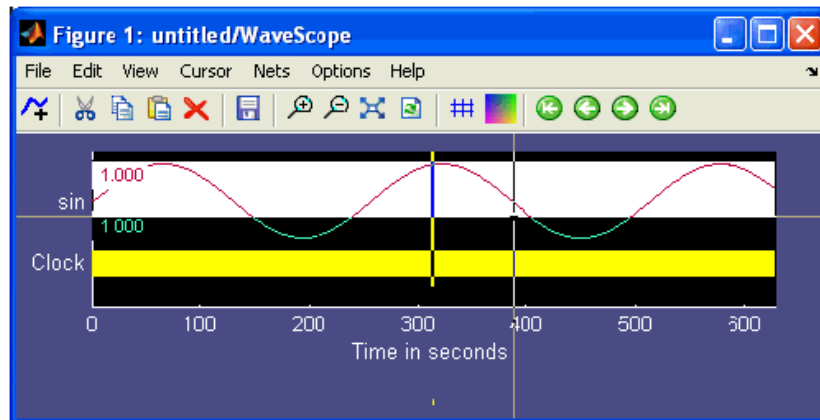


デフォルトでは、ロジック信号の値はグラフで表示されます。表示されないようにするには、[オプション] → [Show Values] をクリックして、オフにします。


クロック信号の表示は変更できません。

アナログ信号の高さの変更

アナログ信号のサイズを変更するには、選択したアナログ信号の下部分を選択して (図参照)、上下に動かすと、アナログ信号を小さくしたり、大きくしたりできます。



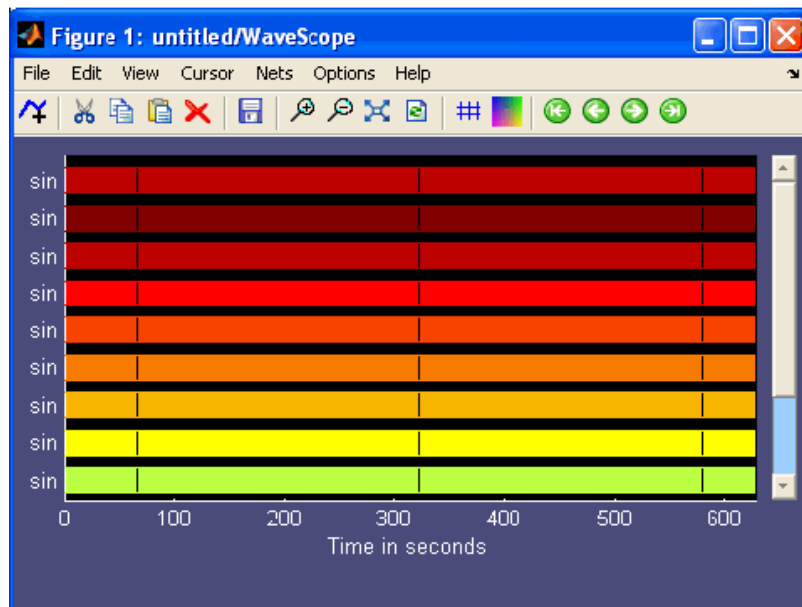
信号名の変更

変更する信号の名前をダブルクリックします。また、モデルのワイヤ名も変更できます。この場合、シミュレーションを再実行するか、 ボタンで WaveScope のウィンドウを更新すると、信号名がアップデートされます。

信号のレインボー表示

信号は、それぞれ別の色を使用すると見やすくなります。信号が追加されると、新しい色がレインボーパレットから選択されます。信号グループを選択して、[Rainbow Selected Signals] ボタンをク

リックすると、レインボー表示されます。すべての信号のレインボー表示を一度にやり直すには、Ctrl キーを押しながらすべてを選択し、[Rainbow Selected Signals] ボタンをクリックします。



信号の表示

信号を選択し、モデルをシミュレーションすると、WaveScope で信号が表示されるようになります。

拡大/縮小とスクロール

虫眼鏡のアイコン、[View] メニュー、キーボードの I および O ボタンのいずれかを使用すると、拡大/縮小ができます。また、WaveScope のウィンドウで四角を描くようにマウスをドラッグすると、その部分を拡大できます。スクロールする場合は、矢印キーを使用するか、画面右側か下側にあるスクロールバーを動かします。すべてが表示内に収まる場合は、スクロールバーは表示されません。

Ctrl キーを使用すると、さらに細かく拡大/縮小ができます。Ctrl キーを押しつつ、左向きまたは右向き矢印キーを押すと、1 クロック サイクル単位で表示が切り替わります。Ctrl キーを押しつつ、I または O キーを押すと、細かい単位で拡大/縮小をすることができます。

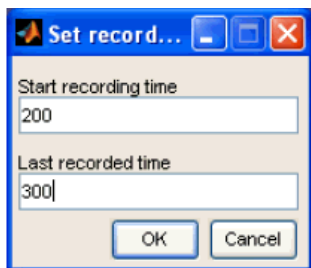
レコーディング制限の変更

データの一部しか表示する必要のないことがあります。たとえば、シミュレーションが長時間実行される場合に、最後の 1000 ステップのみ確認したいときなどです。

WaveScope で表示されるデータが多くなるほど、WaveScope の動きが遅くなります。該当するデータだけを拡大することもできますが、WaveScope に含まれるデータが多ければ、動きは遅いままです。このような場合は、WaveScope でレコーディング制限を削減してください。

デフォルトでは、WaveScope はシミュレーションの開始から終了までの信号の値を記録します。これらを変更するには、[オプション] → [Recording Limits] をクリックし、ダイアログボックスでレコーディングの開始時間と終了時間を指定します。次の図のように、ダイアログボックスには現在

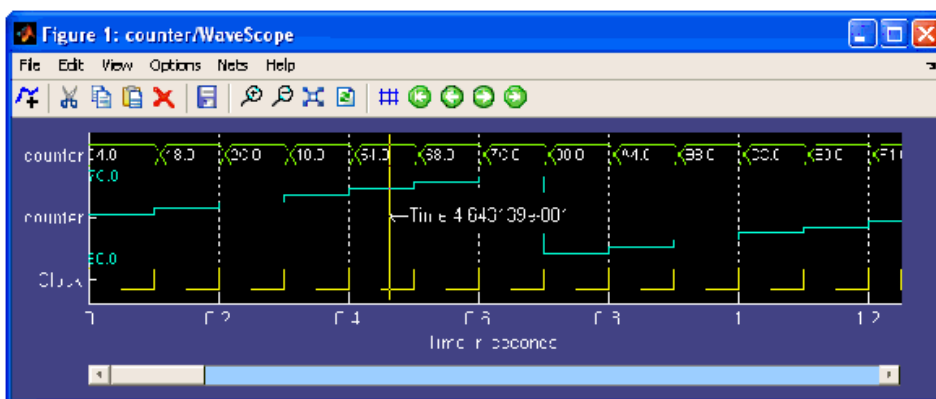
の最小値と最大値が自動的に入力されています。ここに値を入力してください。終了時間に **Inf** と入力すると、時間の上限がなくなります。



レコーディング制限を設定すると、WaveScope ではその時間の範囲内の値しか表示されなくなります。拡大表示しても、この範囲外の部分は表示されません。シミュレーションに戻ると、その範囲時間の値のみが記録されます。

グリッドの表示

ツールバーの [Toggle Grid] ボタン (##) をクリックすると、X 軸の値が付いた縦のラインが表示されるようになります。

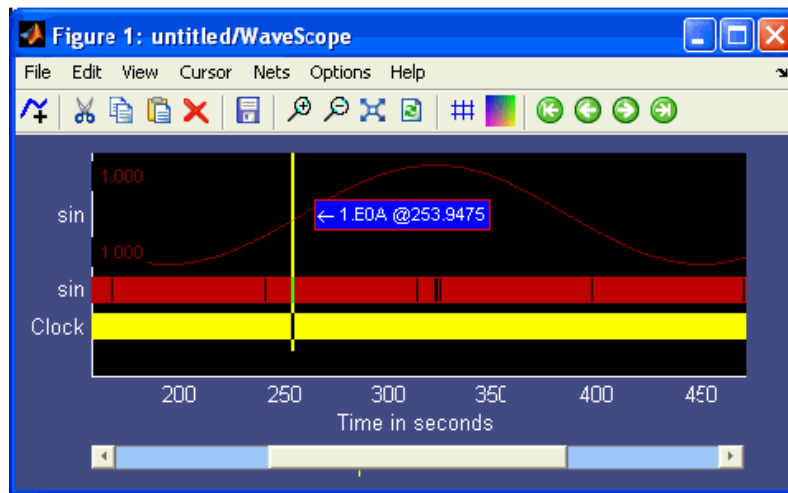


カーソルの使用

カーソルを使用すると、信号を揃えて並べたり、ポイントをマークしたりできます。カーソルは、時間軸の下をクリックすると、現在表示している時間幅に表示されます。時間軸の下にポインタを置くと、マウス ポインタが十字に変わり、カーソルがその位置に移動され、現在の表示内で動かせるようになったことを示します。

C キーを押すか、[Cursor] → [Center Cursor] をクリックすると、カーソルを画面の中央に表示させることもできます。画面に一度表示されたカーソルは、ドラッグすると動かすことができます。マウス ポインタをカーソルの上に置くと、ポインタが十字に変わり、カーソルがドラッグできるようになります。

マウス ポインタをカーソルの上に置くと、そのポインタの下にある信号の値がツール情報として表示されます。ツール情報を使用すると、値がすべて表示されないような拡大比率の場合、アナログ信号の値やロジック信号の値を表示するのに便利です。

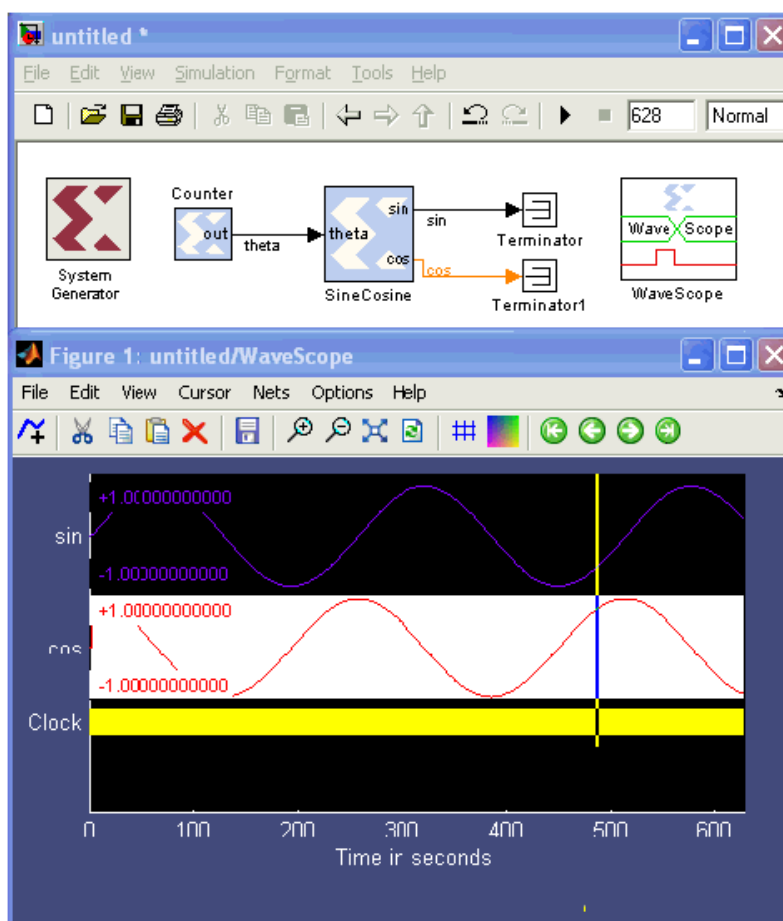


カーソルをドラッグすると、ツール情報は更新されます。スクロールバーの下に、小さい黄色のカーソルが表示されます。選択した表示にカーソルが見当たらない場合は、この小さいカーソルの位置から、実際のカーソルが時間軸のどこに置かれているかがわかります。カーソルを現在の位置に移動するには、J キーを押すか、[Cursor] → [Jump to Cursor] をクリックします。

表示を合わせたり、検索したりせずに、次の信号遷移を表示することもできます。次の遷移を表示するには、カーソルを画面上に置き、遷移を表示する信号を選択します。Enter キーを押すか、[Cursor] → [Move Cursor Next] をクリックすると、カーソルが次の信号遷移に移動します。カーソルが画面からはみ出している場合は、それに合わせて表示されるようになります。

クロスプローブ

信号を WaveScope のウィンドウ内で選択すると、次の図のようにクロスプローブされ、対応するワイヤがモデルのウィンドウでハイライトされます (図のオレンジ色の部分)。



ハイライトされた信号が階層の下の方にある場合は、その上位ブロックがオレンジでハイライトされます。

[Cursor] メニュー

[Cursor] メニューには、次の 4 つのコマンドが含まれます。

[Center Cursor]

現在表示している時間幅の画面の中央にカーソルが配置されます。C キーを押しても、同じように配置されます。

[Jump to Cursor]

カーソルがある位置に現在の表示が移動されます。J キーを押しても、同じように移動されます。

[Move Cursor Next]

一番最近選択した信号の次の遷移にカーソルが移動されます。Enter キーを押しても、同じように移動されます。

[Move Cursor Last]

一番最近選択した信号の前の遷移にカーソルが移動されます。**Shift** キーを押しながら **Enter** キーを押しても、同じように移動されます。

[オプション] メニュー

[オプション] メニューには、次の 4 つのコマンドが含まれます。

[Grid Lines]

時間軸のグリッドが表示されます。

[Show Values]

数値が表示されます。デフォルトでは、オンになっています。クリックすると、表示をオフにできます。

[Run at End of Sim]

WaveScope がシミュレーションの終了時に実行されます。デフォルトでは、オンになっています。シミュレーションの終了時に **WaveScope** が開かないようにするには、クリックして、オフにします。

[Recording Limits]

前述の「レコーディング制限の変更」に記述したとおり、**WaveScope** で表示されるシミュレーション時間を制限するために使用します。

ザイリンクス LogiCORE バージョン

System Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
Accumulator	Accumulator	V11.0	•	•	•	•	•	•	•	•	•	•
Addressable Shift Register	RAM-based Shift Register	V11.0	•	•	•	•	•	•	•	•	•	•
AddSub	Adder Subtractor	V11.0	•	•	•	•	•	•	•	•	•	•
CIC Compiler 1.3	CIC Compiler	V1.3	•	•	•	•	•	•	•	•	•	•
CMult	Multiplier	V11.2	•	•	•	•	•	•	•	•	•	•
Complex Multiplier 3.0	Complex Multiplier	V3.0	•	•	•	•	•	•	•	•	•	•
Complex Multiplier 3.1	Complex Multiplier	V3.1	•	•	•	•	•	•	•	•	•	•
Convolution Encoder 6.1	Convolution Encoder	V6.1	•	•	•			•	•			
Convolution Encoder 7.0	Convolution Encoder	V7.0	•	•	•	•		•	•		•	
CORDIC 4.0	CORDIC	V4.0	•	•	•	•	•	•	•	•	•	•
Counter	Binary Counter	V11.0	•	•	•	•	•	•	•	•	•	•
DAFIR v9_0	Distributed Arithmetic FIR Filter	V9.0	•					•				
DDS Compiler 4.0	DDS Compiler	V4.0	•	•	•	•	•	•	•	•	•	•
Divider Generator 3.0	Divider Generator	V3.0	•	•	•	•	•	•	•	•	•	•
DSP48 Macro 2.0	DSP48 Macro 2.0	V2.0	•	•	•	•	•	•	•	•	•	•
Dual Port RAM	Block Memory Generator	V3.3	•	•	•	•	•	•	•	•	•	•
	Distributed Memory Generator	V4.3	•	•	•	•	•	•	•	•	•	•

System Generator Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
Fast Fourier Transform 7.0	Fast Fourier Transform	V7.0	•	•	•	•	•	•	•	•	•	•
FIFO	FIFO Generator	V5.3	•	•	•	•	•	•	•	•	•	•
FIR Compiler 5.0	FIR Compiler	V5.0	•	•	•	•	•	•	•	•	•	•
From FIFO	FIFO Generator	V5.3	•	•	•	•	•	•	•	•	•	•
Interleaver Deinterleaver 5.0	Interleaver/ De-Interleaver	V5.0	•					•				
Interleaver Deinterleaver 5.1	Interleaver/ De-Interleaver	V5.1	•	•	•			•	•			
Mult	Multiplier	11.2	•	•	•	•	•	•	•	•	•	•
Reed-Solomon Decoder 6.1	Reed-Solomon Decoder	V6.1	•	•	•			•	•			
Reed-Solomon Decoder 7.0	Reed-Solomon Decoder	V7.0	•	•	•			•	•			
Reed-Solomon Encoder 6.1	Reed-SolomonS Encoder	V6.1	•	•	•			•	•			
Reed-Solomon Encoder 7.0	Reed-SolomonS Encoder	V7.0	•	•	•	•		•	•		•	
ROM	Block Memory Generator	V3.3	•	•	•	•	•	•	•	•	•	•
	Distributed Memory Generator	V4.3	•	•	•	•	•	•	•	•	•	•
Single Port RAM	Block Memory Generator	V3.3	•	•	•	•	•	•	•	•	•	•
	Distributed Memory Generator	V4.3	•	•	•	•	•	•	•	•	•	•

System Generator Generator ブロック	ザイリンクス LogiCORE	LogiCORE バージョン/ データシート	Spartan					Virtex				
			3、 3E	3A	3A DSP	6	6-1L	4	5	5Q	6	6-1L
To FIFO	FIFO Generator	V5.3	•	•	•	•	•	•	•	•	•	•
Viterbi Decoder 6_1	Viterbi Decoder	V6.1	•	•	•			•	•			
Viterbi Decoder 7.0	Viterbi Decoder	V7.0	•	•	•	•		•	•	•	•	

ザイリンクス リファレンス ブロックセット

含まれるリファレンス ライブラリは次のとおりです。

通信

[Communication] ライブラリ

[BPSK AWGN Channel](#)

[Convolutional Encoder](#)

[Multipath Fading Channel Model](#)

[White Gaussian Noise Generator](#)

制御ロジック

[Control Logic] ライブラリ

[Mealy State Machine](#)

[Moore State Machine](#)

[Registered Mealy State Machine](#)

[Registered Moore State Machine](#)

DSP

[DSP] ライブラリ

[2 Channel Decimate by 2 MAC FIR Filter](#)

[2n+1-tap Linear Phase MAC FIR Filter](#)

[2n-tap Linear Phase MAC FIR Filter](#)

[2n-tap MAC FIR Filter](#)

[4-channel 8-tap Transpose FIR Filter](#)

[4n-tap MAC FIR Filter](#)

[CIC Filter](#)

[Dual Port Memory Interpolation MAC FIR Filter](#)

[Interpolation Filter](#)

[DSP] ライブラリ

[m-channel n-tap Transpose FIR Filter](#)
[n-tap Dual Port Memory MAC FIR Filter](#)
[n-tap MAC FIR Filter](#)

画像

[Imaging] ライブラリ

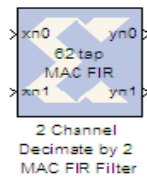
[5x5Filter](#)
[Virtex Line Buffer](#)
[Virtex2 5 Line Buffer](#)
[Virtex2 Line Buffer](#)

演算

[Math] ライブラリ

[CORDIC ATAN](#)
[CORDIC DIVIDER](#)
[CORDIC LOG](#)
[CORDIC SINCOS](#)
[CORDIC SQRT](#)

2 Channel Decimate by 2 MAC FIR Filter



積和ベースの FIR フィルタをインプリメントします。**n-tap** フィルタでは、専用乗算器 1 つとデュアルポートブロック RAM 1 つが使用されます。時分割多重 (TDM) の 2 つのチャンネルが同じ MAC エンジンと一緒に処理されます。各チャンネルに指定する係数のセットは、含まれる係数の数が同じであれば完全に異なっていてもかまいません。フィルタでは、多相フィルタ手法を使用して、2 に固定されたデシメーションが実行されます。フィルタの設定には、複数の係数セットおよびデータサンプルをフィルタデザインに格納する手法が示されます。Virtex™ FPGA ファミリ (および Virtex ファミリの派生デバイス) では、高速で小型の加算器、乗算器、および柔軟なメモリアーキテクチャを構築するための専用回路が提供されます。フィルタデザインではこれらのシリコンの特性が活用され、小型でリソース効率のよいデザインがインプリメントされます。

このフィルタリファレンスブロックには、インプリメンテーションの説明が提供されています。説明を読むには、ブロックをモデル内に配置し、そのブロックを右クリックして、[Explore] を選択します。サブブロックの 1 つをダブルクリックすると、そのサブブロックのモデルが開き、説明が読めるようになります。

ブロックパラメータ

ブロックのパラメータダイアログボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

このリファレンスブロックのパラメータは次のとおりです。

- [Data Input Bit Width] : 入力サンプルの幅
- [Data Input Binary Point] : 入力の 2 進小数点の位置
- [Coefficient Vector (Ch.1)] : フィルタのチャンネル 1 の係数を指定します。タップ数は、係数ベクタのサイズから推論されます。
- [Coefficient Vector (Ch.2)] : フィルタのチャンネル 2 の係数を指定します。タップ数は、係数ベクタのサイズから推論されます。
 メモ : 係数ベクタのサイズは同じである必要があります。必要に応じてパディングを追加し、サイズを同じにします。
- [Number of Bits per Coefficient] : 各係数のビット幅
- [Binary Point per Coefficient] : 各係数の 2 進小数点の位置
 メモ : 係数ベクタのサイズは同じである必要があります。必要に応じてパディングを追加し、サイズを同じにします。
- [Sample Period] : 入力のサンプル周期

参考資料

J. Hwang, J. Ballagh, 『Building Custom FIR Filters Using System Generator』、12th International Field-Programmable Logic and Applications Conference (FPL) (2002 年 9 月、モンペリエ (フランス) にて開催)、LNCS (Lecture Notes in Computer Science) Vol. 2438

2n+1-tap Linear Phase MAC FIR Filter



積和ベースの **FIR** フィルタをインプリメントします。このフィルタでは、奇数個の係数の係数対称を利用して、フィルタのスループットを向上させます。フィルタ デザインでは、**Virtex** ファミリー **FPGA** のシリコンの特性 (高速で小型の加算器、乗算器、および柔軟なメモリ アーキテクチャを構築するための専用回路) が利用されます。

このフィルタ リファレンス ブロックには、インプリメンテーションの説明が提供されています。説明を読むには、ブロックをモデル内に配置し、そのブロックを右クリックして、**[Explore]** を選択します。サブブロックの 1 つをダブルクリックすると、そのサブブロックのモデルが開き、説明が読めるようになります。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、**Simulink** モデル内でブロックをダブルクリックすると表示されます。

パラメータは次のとおりです。

- **[Coefficients]** : フィルタの係数を指定します。タップ数は、係数ベクタのサイズから推論されます。
- **[Number of Bits per Coefficient]** : 各係数のビット幅
- **[Binary Point for Coefficient]** : 各係数の 2 進小数点の位置
- **[Number of Bits per Input Sample]** : 入力サンプルの幅
- **[Binary Point for Input Samples]** : 入力の 2 進小数点の位置
- **[Input Sample Period]** : 入力のサンプル周期

参考資料

J. Hwang, J. Ballagh, 『Building Custom FIR Filters Using System Generator』、12th International Field-Programmable Logic and Applications Conference (FPL) (2002 年 9 月、モンペリエ (フランス) にて開催)、LNCS (Lecture Notes in Computer Science) Vol. 2438

2n-tap Linear Phase MAC FIR Filter



積和ベースの FIR フィルタをインプリメントします。このブロックでは、偶数個の係数の係数対称を利用して、フィルタのスループットを向上させます。フィルタデザインでは、Virtex ファミリ FPGA のシリコンの特性 (高速で小型の加算器、乗算器、および柔軟なメモリ アーキテクチャを構築するための専用回路) が利用されます。

このフィルタ リファレンス ブロックには、インプリメンテーションの説明が提供されています。説明を読むには、ブロックをモデル内に配置し、そのブロックを右クリックして、[Explore] を選択します。サブブロックの 1 つをダブルクリックすると、そのサブブロックのモデルが開き、説明が読めるようになります。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

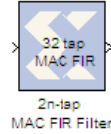
パラメータは次のとおりです。

- [Coefficients] : フィルタの係数を指定します。タップ数は、係数ベクタのサイズから推論されます。
- [Number of Bits per Coefficient] : 各係数のビット幅
- [Binary Point for Coefficient] : 各係数の 2 進小数点の位置
- [Number of Bits per Input Sample] : 入力サンプルの幅
- [Binary Point for Input Samples] : 入力の 2 進小数点の位置
- [Input Sample Period] : 入力のサンプル周期

参考資料

J. Hwang, J. Ballagh, 『Building Custom FIR Filters Using System Generator』、12th International Field-Programmable Logic and Applications Conference (FPL) (2002 年 9 月、モンペリエ (フランス) にて開催)、LNCS (Lecture Notes in Computer Science) Vol. 2438

2n-tap MAC FIR Filter



積和ベースの **FIR** フィルタをインプリメントします。3 とおりのフィルタ コンフィギュレーションにより、フィルタのスループットとデバイスのリソース使用のトレードオフが示されます。**Virtex FPGA** ファミリ (および **Virtex** ファミリの派生デバイス) では、高速で小型の加算器、乗算器、および柔軟なメモリ アーキテクチャを構築するための専用回路が提供されます。フィルタ デザインではこれらのシリコンの特性が活用され、小型でリソース効率のよいデザインがインプリメントされます。

このフィルタ リファレンス ブロックには、インプリメンテーションの説明が提供されています。説明を読むには、ブロックをモデル内に配置し、そのブロックを右クリックして、**[Explore]** を選択します。サブブロックの 1 つをダブルクリックすると、そのサブブロックのモデルが開き、説明が読めるようになります。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、**Simulink** モデル内でブロックをダブルクリックすると表示されます。

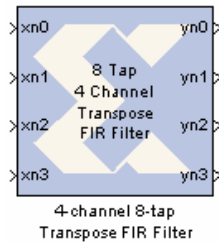
パラメータは次のとおりです。

- **[Coefficients]** : フィルタの係数を指定します。タップ数は、係数ベクタのサイズから推論されます。
- **[Number of Bits per Coefficient]** : 各係数のビット幅
- **[Binary Point for Coefficient]** : 各係数の 2 進小数点の位置
- **[Number of Bits per Input Sample]** : 入力サンプルの幅
- **[Binary Point for Input Samples]** : 入力の 2 進小数点の位置
- **[Input Sample Period]** : 入力のサンプル周期

参考資料

J. Hwang, J. Ballagh, 『Building Custom FIR Filters Using System Generator』、12th International Field-Programmable Logic and Applications Conference (FPL) (2002 年 9 月、モンペリエ (フランス) にて開催)、LNCS (Lecture Notes in Computer Science) Vol. 2438

4-channel 8-tap Transpose FIR Filter



4 チャンネル 8 タップの転置型 FIR フィルタをインプリメントします。転置構造はザイリンクス **FPGA** でのデータ パス処理に適しており、より大型のフィルタに容易に拡張できます。フィルタでは、**Virtex** ファミリー **FPGA** のシリコンの特性 (高速で小型の加算器、乗算器、および柔軟なメモリーアーキテクチャを構築するための専用回路) が利用されます。

このフィルタ リファレンス ブロックには、インプリメンテーションの説明が提供されています。説明を読むには、ブロックをモデル内に配置し、そのブロックを右クリックして、**[Explore]** を選択します。サブブロックの 1 つをダブルクリックすると、そのサブブロックのモデルが開き、説明が読めるようになります。

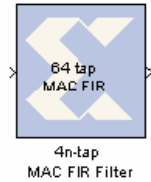
ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、**Simulink** モデル内でブロックをダブルクリックすると表示されます。

パラメータは次のとおりです。

- **[Coefficients]** : フィルタの係数を指定します。タップ数は、係数ベクタのサイズから推論されます。
- **[Number of Bits per Coefficient]** : 各係数のビット幅
- **[Binary Point for Coefficient]** : 各係数の 2 進小数点の位置
- **[Number of Bits per Input Sample]** : 入力サンプルの幅
- **[Binary Point for Input Samples]** : 入力の 2 進小数点の位置
- **[Input Sample Period]** : 入力のサンプル周期

4n-tap MAC FIR Filter



積和ベースの **FIR** フィルタをインプリメントします。3 とおりのフィルタ コンフィギュレーションにより、フィルタのスループットとデバイスのリソース使用のトレードオフが示されます。**Virtex FPGA** ファミリ (および **Virtex** ファミリの派生デバイス) では、高速で小型の加算器、乗算器、および柔軟なメモリ アーキテクチャを構築するための専用回路が提供されます。フィルタ デザインではこれらのシリコンの特性が活用され、小型でリソース効率のよいデザインがインプリメントされます。

このフィルタ リファレンス ブロックには、インプリメンテーションの説明が提供されています。説明を読むには、ブロックをモデル内に配置し、そのブロックを右クリックして、**[Explore]** を選択します。サブブロックの 1 つをダブルクリックすると、そのサブブロックのモデルが開き、説明が読めるようになります。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、**Simulink** モデル内でブロックをダブルクリックすると表示されます。

パラメータは次のとおりです。

- **[Coefficients]** : フィルタの係数を指定します。タップ数は、係数ベクタのサイズから推論されます。
- **[Number of Bits per Coefficient]** : 各係数のビット幅
- **[Binary Point for Coefficient]** : 各係数の 2 進小数点の位置
- **[Number of Bits per Input Sample]** : 入力サンプルの幅
- **[Binary Point for Input Samples]** : 入力の 2 進小数点の位置
- **[Input Sample Period]** : 入力のサンプル周期

参考資料

J. Hwang, J. Ballagh, 『Building Custom FIR Filters Using System Generator』、12th International Field-Programmable Logic and Applications Conference (FPL) (2002 年 9 月、モンペリエ (フランス) にて開催)、LNCS (Lecture Notes in Computer Science) Vol. 2438

5x5Filter



5 つの n-tap MAC FIR フィルタを使用してインプリメントされます。n-tap MAC FIR フィルタは、[Xilinx Reference Blockset] → [DSP] ライブラリに含まれています。

グレースケールの画像をフィルタする 9 つの異なる 2-D フィルタがあります。フィルタを選択するには、5x5 フィルタブロックのマスクパラメータを変更します。2-D フィルタ係数はブロック RAM に格納され、モデルでは係数は最適化されません。[Initialization] タブで 5x5 フィルタブロックのマスクを変更すると、代わりに独自の係数とスケール係数を使用できます。

9 つのフィルタで使用されている係数を次に示します。フィルタの出力は、<filter name>Div と呼ばれるスケール係数で乗じられます。

```
edge = [ 0  0  0  0 0; ...
        0 -1 -1 -1 0; ...
        0 -1 -1 -1 0; ...
        0  0  0  0 0];
edgeDiv = 1;

sobelX = [ 0  0  0  0 0; ...
          0 -1  0  1 0; ...
          0 -2  0  2 0; ...
          0 -1  0  1 0; ...
          0  0  0  0 0];
sobelXDiv = 1;

sobelY = [ 0  0  0  0 0; ...
          0  1  2  1 0; ...
          0  0  0  0 0; ...
          0 -1 -2 -1 0; ...
          0  0  0  0 0];
sobelYDiv = 1;

sobelXY = [ 0  0  0  0 0; ...
            0  0 -1 -1 0; ...
            0  1  0 -1 0; ...
            0  1  1  0 0; ...
            0  0  0  0 0];
sobelXYDiv = 1;

blur = [ 1  1  1  1 1; ...
        1  0  0  0 1; ...
        1  0  0  0 1; ...
        1  0  0  0 1; ...
        1  1  1  1 1];
blurDiv = 1/16;

smooth = [ 1  1  1  1 1; ...
           1  5  5  5 1; ...
           1  5 44  5 1; ...
           1  5  5  5 1; ...
           1  1  1  1 1];
smoothDiv = 1/100;
```

```

sharpen = [ 0  0  0  0 0; ...
0 -2 -2 -2 0; ...
0 -2 32 -2 0; ...
0 -2 -2 -2 0; ...
0  0  0  0 0];
sharpenDiv = 1/16;

gaussian = [1 1 2 1 1; ...
1 2 4 2 1; ...
2 4 8 4 2; ...
1 2 4 2 1; ...
1 1 2 1 1];
gaussianDiv = 1/52;

identity = [ 0  0  0  0 0; ...
0  0  0  0 0; ...
0  0  1  0 0; ...
0  0  0  0 0; ...
0  0  0  0 0];
identityDiv = 1;

```

このフィルタでは、スライス 309 個、専用乗算器 5 個、ザイリンクス XC2V250-6 デバイスの RAM ブロック 5 個が使用され、213MHz (Advanced スピード ファイル 1.96、ISE 4.2.01i ソフトウェア) で動作します。

基礎となる 5-tap MAC FIR フィルタは、入力レートの 5 倍の速さで動作します。このため、デザインのスループットは $213\text{MHz} / 5 = 426,000,000$ ピクセル/秒となります。 64×64 の画像では、これは $42.6 \times 10^6 / (64 \times 64) = 10,400$ フレーム/秒、 256×256 の画像ではスループットは 650 フレーム/秒、 512×512 の画像では 162 フレーム/秒となります。

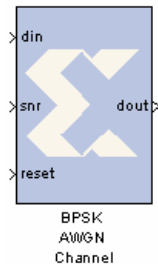
ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

パラメータは次のとおりです。

- [5x5 Mask] : [Edge]、[Sobel X]、[Sobel Y]、[Sobel X-Y]、[Blur]、[Smooth]、[Sharpen]、[Gaussian]、または [Identity] フィルタの係数が選択できます。
- [Sample Period] : 入力信号のサンプル周期を必ず指定します。

BPSK AWGN Channel

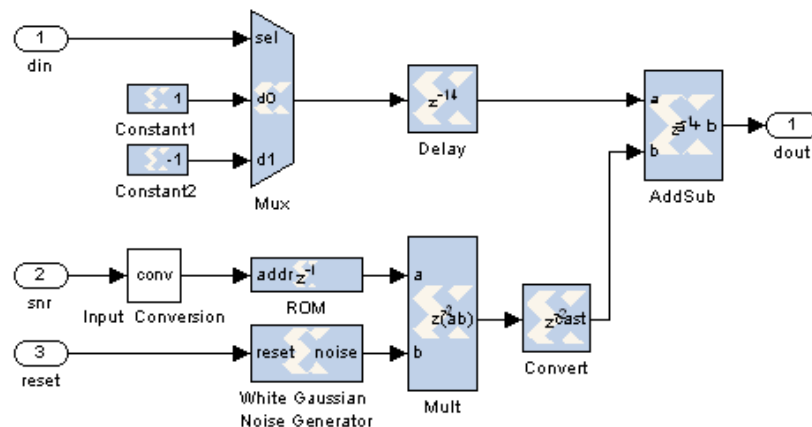


入力信号にスケールされた白色ガウス雑音を追加します。白色ガウス雑音は、**White Gaussian Noise Generator** リファレンス ブロックで生成されます。

このノイズは、ノイズのばらつきを調整するため、次に示すように **SNR** に基づいてスケールされます。ユニット シンボル エネルギー ($E_s = 1$) を使用する符号化されていない **BPSK** では、**SNR** は E_b/N_0 で、dB の単位で表されます。**SNR** 入力は **UFix8_4** で、有効な範囲は 0.0625 刻みで 0.0 ~ 15.9375dB です。

符号付きシステムで **AWGN** を使用する場合、またはコアを異なる変調形式で使用する場合は、**SNR** を調整してスペクトル効率の違いに対応する必要があります。コード レートが 1/2 の **BPSK** で、 $E_s = 1$ および N_0 が定数である場合は、 $E_b = 2$ および $E_b/N_0 = \text{SNR} + 3 \text{ dB}$ となります。符号化されていない **QPSK** で、 $I = \pm 1$ および $Q = \pm 1$ 、独立したノイズシーケンスが加わる場合は、各チャネルは独立した **BPSK** チャネルのようになり、 $E_b/N_0 = \text{SNR}$ となります。この **QPSK** にコード レート 1/2 を追加すると、 $E_b/N_0 = \text{SNR} + 3 \text{ dB}$ となります。

AWGN チャネルの全体的なレイテンシは、15 クロック サイクルです。チャネル出力は 17 ビット (2 進小数点以下が 11 ビット) の符号付きの値です。入力ポート **SNR** のデータ型は任意です。リセットポートはブール型、**din** 入力ポートは 2 進小数点の位置が 0 である符号なし 1 ビット値である必要があります。



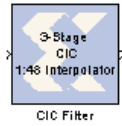
ブロック パラメータ

[Decimal Seed Value] : 10 進開始シード値

参考資料

1. A. Ghazel, E. Boutillon, J. L. Danger, G. Gulak and H. Laamari, 『Design and Performance Analysis of a High Speed AWGN Communication Channel Emulator』、IEEE PACRIM Conference (2001 年 8 月 プリティッシュ コロンビア州ビクトリア (カナダ) にて開催)
2. ザイリンクス データシート、『Additive White Gaussian Noise (AWGN) Core v1.0』、Xilinx, Inc., 2002 年 10 月

CIC Filter



このフィルタはマルチレート フィルタで、デジタル システムにおけるサンプル レートの大幅な変化の検出に使用されます。デシメーション構造、インターポレーション構造の両方がサポートされます。**CIC** フィルタには乗算器が含まれず、加算器、減算器、およびレジスタのみが含まれます。通常は、信号の帯域幅よりもシステムのサンプルレートがかなり大きい場合のように、サンプル レートが非常に大きなアプリケーションで使用されます。このフィルタは、デジタル ダウン コンバータおよびデジタル アップ コンバータで頻繁に使用されます。

このフィルタ リファレンス ブロックには、インプリメンテーションの説明が提供されています。説明を読むには、ブロックをモデル内に配置し、そのブロックを右クリックして、[Explore] を選択します。サブブロックの 1 つをダブルクリックすると、そのサブブロックのモデルが開き、説明が読めるようになります。

ブロック インターフェイス

CIC ブロックには次のシングル データ入力ポートおよびデータ出力ポートがあります。

- **xn** : データ入力ポート。1 ～ 128 ビット。
- **yn** : データ出力ポート

CIC フィルタの基本的な構成要素は、積分器とくし形フィルタです。1 つの積分器は、次の伝達関数を使用するシングル ポールの IIR フィルタです。

$$H(z) = (1 - z^{-1})^{-1}$$

積分器の直結フィードバック係数は、 $y[n] = y[n-1] + x[n]$ です。

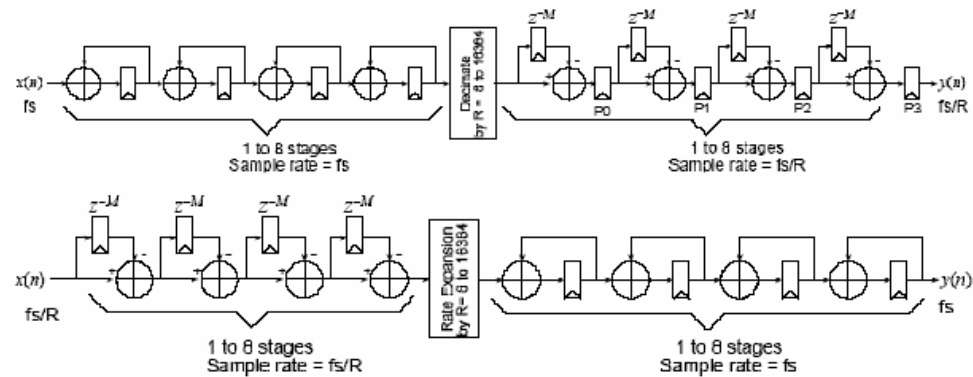
1 つのくし形フィルタは、次の式で表現される奇数対称の FIR フィルタです。

$$y[n] = x[n] - x[n - RM]$$

ここで **M** は、ブロック パラメータで選択された差動遅延、**R** は選択された整数レートの変更係数です。くし形フィルタ 1 段の伝達関数は次のとおりです。

$$H(z) = 1 - z^{-RM}$$

次の 2 つの図に見られるように、CIC フィルタでは **N** 個の積分器セクションが **N** 個のくし形セクションにカスケード接続されます。積分器構造およびくし形構造がレートの変化の影響を受けないように、セクション間には **Up Sample** や **Down Sample** などのレート変更ブロックが挿入されます。インターポレータでは、**Up Sample** ブロックにより、値が 0 のサンプルが **R-1** 個、くし形セクションの出力の連続したサンプルの間に挿入され、レートが **R** 倍になります。デシメータでは、**Down Sample** ブロックにより、最後の積分器段から出力のサブサンプルが削除され、サンプル レートが **1/R** になります。



ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

パラメータは次のとおりです。

- [Input Bit Width] : 入力サンプルの幅
- [Input Binary Point] : 入力の 2 進小数点の位置
- [Filter Type] : [Interpolator] または [Decimator]
- [Sample Rate Change] : 8 ~ 16384
- [Number of Stages] : 1 ~ 32
- [Differential Delay] : 1 ~ 4
- [Pipeline Differentiators] : オンまたはオフ

参考資料

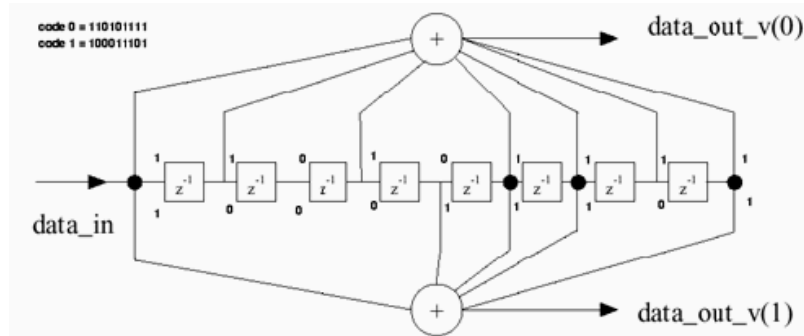
E. B. Hogenauer, 『An economical class of digital filters for decimation and interpolation』、IEEE トランザクション「Acoustics, Speech and Signal Processing」vol. ASSP- 29(2)、1981 年、pp. 155 ~ 162

Convolutional Encoder



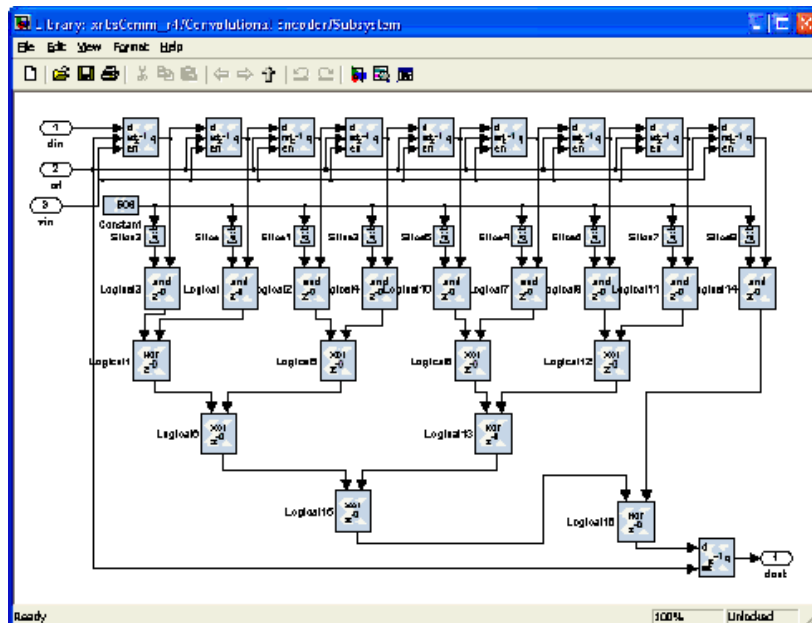
たたみ込み符号のエンコーダをインプリメントします。通常 Viterbi デコーダと併用され、デジタル通信システムで順方向誤り訂正 (FEC) を実行します。

値のエンコードにはリニア フィード フォワード シフト レジスタが使用され、次の図に示すように、入力データをスライドさせながらモジュロ 2 の和を算出します。シフトレジスタ長は制約長で指定されます。たたみ込み符号では、データ枠のどのビットをモジュロ 2 に加算するかが指定されます。ブロックをリセットすると、シフトレジスタが 0 に設定されます。エンコーダのレートは、入力ビット長の出力ビット長に対する比率です。レートが 1/2 のエンコーダからは、各入力ビットごとに 2 ビットが出力されます。同様に、レートが 1/3 のエンコーダからは、各入力ビットごとに 3 ビットが出力されます。



インプリメンテーション

ブロックは、パラメータ指定可能なマルチプレクサ ベースのコラプスの一種を使用してインプリメントされます。この方法では、ロジック ブロックが定数で駆動されます。ここでは定数はたたみ込み符号で、リニア フィード フォワード シフト レジスタのどのレジスタを出力の計算に使用するかを決定します。定数で駆動されるロジックはすべて、ダウンストリームの合成ツールで最適化されて削除されます。



ブロック インターフェイス

現在このブロックには 3 つの入力ポートと 3 つの出力ポートがあります。**din** ポートのタイプは **UFix1_0** である必要があります。ここでエンコードされる値が許可されます。**vin** ポートは、**din** に示された値が有効であることを表します。有効な値のみがエンコードされます。**rst** ポートでは、**High** の場合にたたみ込みエンコーダがリセットされます。イネーブル ポートを追加するには、サブシステムを開き、定数 **Enable** を入力ポートに変更します。出力ポート **dout1** および **dout2** からは、エンコードされたデータが出力されます。ポート **dout1** は配列の最初の符号、**dout2** は 2 番目、というように対応します。出力ポートを追加するには、サブシステムを開き、モデルに示される手順に従います。出力ポート **vout** は、出力値が有効かどうかを示します。

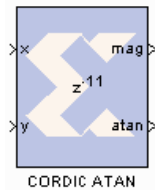
ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、**Simulink** モデル内でブロックをダブルクリックすると表示されます。

パラメータは次のとおりです。

- **[Constraint Length]** : 制約長。n+1 で、n はエンコーダに含まれる制約レジスタ長。
- **[Convolutional code array (octal)]** : 8 進数のたたみ込み符号の配列。出力レートは配列長から派生します。2 ~ 7 の符号を指定します。

CORDIC ATAN



Circular Vectoring モードで完全にパラレルな CORDIC (COordinate Rotation DIgital Computer) アルゴリズムを使用して、直交座標から極座標への変換をインプリメントします。

入力を $\langle x, y \rangle$ とすると、絶対値 $m = K \times \sqrt{x^2 + y^2}$ および角度 $a = \arctan(y/x)$ より新しいベクタ $\langle m, a \rangle$ が算出されます。通常どおり、絶対値のスケール係数 $K = 1.646760...$ はプロセッサでは補正されないため、絶対値出力をこの係数でスケールする必要があります。CORDIC プロセッサのインプリメントには、ザイリンクス ブロックセットのブロックが使用されます。

このアルゴリズムは、次の 3 段階でインプリメントされます。

1. 大まかな角度の回転：このアルゴリズムでは、 $-\pi/2 \sim \pi/2$ の角度のみが収束されます。 $x < 0$ の場合、入力ベクタを対称移動して X 座標を負でない値にし、第 1 象限および第 3 象限に移動します。
2. 詳細な角度の回転：直交座標から極座標に変換するため、結果のベクタを y が 0 に近づくように少しずつ回転します。 i 段目での回転角度は、 $\pm \arctan(1/2^i)$ (正負は、入力 y が負の場合は正で、正の場合は負) になります。
3. 角度修正：手順 1 で対称移動が実行された場合は、 $\pm\pi$ から減算して修正します。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

パラメータは次のとおりです。

- [Number of Processing Elements]：詳細な角度の回転での段の反復回数を指定します。
- [X,Y Data Width]：入力 x および y の幅を指定します。入力 x および y はデータ幅が同じで、符号付きである必要があります。
- [X,Y Binary Point Position]：入力 x および y の 2 進小数点の位置を指定します。入力 x および y は 2 進小数点の位置が同じで、符号付きである必要があります。
- [Latency for each Processing Element]：各循環回転段の後のパイプライン レイテンシを設定します。

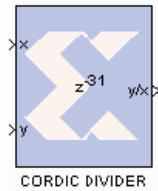
このブロックのレイテンシは、次の式に基づいて算出されます。

$$\text{Latency} = 3 + \text{sum (処理エレメントのレイテンシ)}$$

参考資料

1. J. E. Volder, 『The CORDIC Trigonometric Computing Technique』、IRE トランザクション、Electronic Computers、Vol. EC-8、1959 年、pp. 330 ~ 334
2. J. S. Walther, 『A Unified Algorithm for Elementary Functions』、Spring Joint Computer Conference、1971 年、pp. 379 ~ 385
3. Yu Hen Hu, 『CORDIC-Based VLSI Architectures for Digital Signal Processing』、IEEE Signal Processing Magazine、1992 年 7 月、pp. 17 ~ 34

CORDIC DIVIDER



Linear Vectoring モードで完全にパラレルな CORDIC (COordinate Rotation Digital Computer) アルゴリズムを使用して、除算回路をインプリメントします。

入力を $\langle x, y \rangle$ とすると、出力 y/x が算出されます。CORDIC プロセッサのインプリメントには、ザイリンクス ブロックセットのブロックが使用されます。

このアルゴリズムは、次の 4 段階でインプリメントされます。

1. 座標回転: CORDIC アルゴリズムでは、 x の正の値のみが収束されます。 x および y 座標を負でない値にし、入力ベクトルを第 1 象限に移動します。この回路は、最小の負の値を除く X および Y のすべての値を収束させるように設計されています。
2. 正規化: CORDIC アルゴリズムでは、 $2x$ 以下の y のみが収束されます。入力 x および y を、最上位ビット (MSB) が 1 になるまで左にシフトします。 y の x に相対するシフトは記録され、座標修正段に渡されます。
3. リニア回転: 比を算出するため、結果のベクトルを y が 0 に近づくように少しずつ回転します。回転の最終段で、 y/x が得られます。
4. 座標修正: 座標軸の移動および x に相対する y のシフトに基づき、結果の比に正しい符号を割り当て、 2^n (x に相対する y のシフト) で乗算します。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

パラメータは次のとおりです。

- [Number of Processing Elements]: リニア回転での段の反復回数を指定します。
- [X,Y Data Width]: 入力 x および y の幅を指定します。入力 x および y はデータ幅が同じで、符号付きである必要があります。
- [X,Y Binary Point Position]: 入力 x および y の 2 進小数点の位置を指定します。入力 x および y は 2 進小数点の位置が同じで、符号付きである必要があります。
- [Latency for each Processing Element]: 各リニア回転段の後のパイプライン レイテンシを設定します。

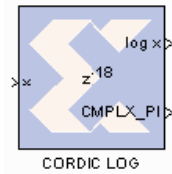
このブロックのレイテンシは、次の式に基づいて算出されます。

$$\text{Latency} = 4 + \text{data width} + \text{sum (処理エレメントのレイテンシ)}$$

参考資料

1. J. E. Volder, 『The CORDIC Trigonometric Computing Technique』、IRE トランザクション、Electronic Computers、Vol. EC-8、1959 年、pp. 330 ~ 334
2. J. S. Walther, 『A Unified Algorithm for Elementary Functions』、Spring Joint Computer Conference、1971 年、pp. 379 ~ 385
3. Yu Hen Hu, 『CORDIC-Based VLSI Architectures for Digital Signal Processing』、IEEE Signal Processing Magazine、1992 年 7 月、pp. 17 ~ 34

CORDIC LOG



Hyperbolic Vectoring モードで完全にパラレルな CORDIC (COordinate Rotation Digital Computer) アルゴリズムを使用して、自然対数回路をインプリメントします。

入力を x とすると、 $\log(x)$ が算出され、複素出力が必要な場合は、複素 π 値を追加するためのフラグも出力されます。CORDIC プロセッサのインプリメントには、ザイリンクス ブロックセットのブロックが使用されます。

自然対数は、CORDIC アルゴリズムで次の式を使用して間接的に計算されます。

$$\log(w) = 2 \times \tanh^{-1}[(w-1)/(w+1)]$$

$$\log(w \times 2^E) = \log(w) + E \times \log(2)$$

このアルゴリズムは、次の 4 段階でインプリメントされます。

1. 座標回転: CORDIC アルゴリズムでは、正の x 値のみが収束されます。 $x < 0$ の場合、入力データを負でない値に変換します。 $x = 0$ の場合、0 が検出されたことを示すフラグを出力段に接続されている最終段に伝搬します。この回路は、最小の負の値を除く x のすべての値を収束させるように設計されています。
2. 正規化: CORDIC アルゴリズムでは、値が $0.5 \sim 1$ である x のみが収束されます。正規化では、入力 x を最上位ビットが 1 になるまで左にシフトします。対数出力は、 $\log(w) = 2 \times \tanh^{-1}\{(w-1)/(w+1)\}$ の式より得られます。この式に基づき、入力 w は $x = w + 1$ および $y = w - 1$ に対応付けられます。
3. リニア回転: $\tanh^{-1}\{(w-1)/(w+1)\}$ を算出するため、結果のベクトルを y が 0 に近づくように少しずつ回転します。
4. 座標修正: 入力が負で複素出力が必要な場合、 π を追加するために、CMPLX_PI フラグを出力します。左シフトが x に適用されている場合、 $\log(w \times 2^E) = \log(w) + E \times \log(2)$ の式で出力を調整します。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

パラメータは次のとおりです。

- [Number of Processing Elements (integer value starting from 1)]: 双曲線回転での段の反復回数を指定します。
- [Input Data Width]: 入力 x の幅を指定します。入力 x は指定されたデータ幅で、符号付きである必要があります。
- [Input Binary Point Position]: 入力 x の 2 進小数点の位置を指定します。入力 x は 2 進小数点指定された位置にあり、符号付きである必要があります。
- [Latency for each Processing Element [1001]]: 各双曲線回転段の後のパイプライン レイテンシを設定します。

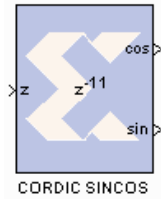
このブロックのレイテンシは、次の式に基づいて算出されます。

$$\text{Latency} = 2 + \text{Data Width} + \text{sum (処理エレメントのレイテンシ)}$$

参考資料

1. J. E. Volder、『The CORDIC Trigonometric Computing Technique』、IRE トランザクション、Electronic Computers、Vol. EC-8、1959 年、pp. 330 ~ 334
2. J. S. Walther、『A Unified Algorithm for Elementary Functions』、Spring Joint Computer Conference、1971 年、pp. 379 ~ 385
3. Yu Hen Hu、『CORDIC-Based VLSI Architectures for Digital Signal Processing』、IEEE Signal Processing Magazine、1992 年 7 月、pp. 17 ~ 34

CORDIC SINCOS



Circular Rotation モードで完全にパラレルな CORDIC (COordinate Rotation Digital Computer) アルゴリズムを使用して、サイン波およびコサイン波生成回路をインプリメントします。

入力角度を z とすると、出力 $\cos(z)$ および $\sin(z)$ が算出されます。CORDIC プロセッサのインプリメントには、ザイリンクス ブロックセットのブロックが使用されます。このアルゴリズムは、次の 3 段階でインプリメントされます。

1. 大まかな角度の回転: $-\pi/2 \sim \pi/2$ の角度のみが収束されます。 $z > \pi/2$ の場合、 $\pi/2$ を減算して第 1 象限に対称移動します。 $z < -\pi/2$ の場合、 $\pi/2$ を加算して第 3 象限に対称移動します。この回路は、最小の負の値を除く z のすべての値を収束させるように設計されています。
2. 詳細な角度の回転: x を $1/1.646760$ に、 y を 0 に設定し、回転モードの CORDIC プロセッサで、入力角度 z のコサイン波およびサイン波を生成します。
3. 座標修正: 手順 1 で対称移動が実行された場合は、この段階で修正します。

$z > \pi/2$ の場合、 $z = t + \pi/2$ を使用

$$\sin(z) = \sin(t) \cdot \cos(\pi/2) + \cos(t) \cdot \sin(\pi/2) = \cos(t)$$

$$\cos(z) = \cos(t) \cdot \cos(\pi/2) - \sin(t) \cdot \sin(\pi/2) = -\sin(t)$$

$z < \pi/2$ の場合、 $z = t - \pi/2$ を使用

$$\sin(z) = \sin(t) \cdot \cos(-\pi/2) + \cos(t) \cdot \sin(-\pi/2) = -\cos(t)$$

$$\cos(z) = \cos(t) \cdot \cos(-\pi/2) - \sin(t) \cdot \sin(-\pi/2) = \sin(t)$$

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

パラメータは次のとおりです。

- [Number of Processing Elements]: リニア回転での段の反復回数を指定します。
- [Input Data Width]: 入力 z の幅を指定します。入力 z は指定されたデータ幅で、符号付きである必要があります。
- [Input Binary Point Position]: 入力 z の 2 進小数点の位置を指定します。入力 z は 2 進小数点の位置が同じで、符号付きである必要があります。2 進小数点は、 $\pi/2$ を表すのにビット数が十分になる位置に指定する必要があります。
- [Latency for each Processing Element]: 各循環回転段の後のパイプライン レイテンシを設定します。

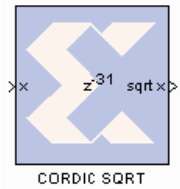
このブロックのレイテンシは、次の式に基づいて算出されます。

$$\text{Latency} = 3 + \text{sum (処理エレメントのレイテンシ)}$$

参考資料

1. J. E. Volder, 『The CORDIC Trigonometric Computing Technique』、IRE トランザクション、Electronic Computers、Vol. EC-8、1959 年、pp. 330 ~ 334
2. J. S. Walther, 『A Unified Algorithm for Elementary Functions』、Spring Joint Computer Conference、1971 年、pp. 379 ~ 385
3. Yu Hen Hu, 『CORDIC-Based VLSI Architectures for Digital Signal Processing』、IEEE Signal Processing Magazine、1992 年 7 月、pp. 17 ~ 34

CORDIC SQRT



Hyperbolic Vectoring モードで完全にパラレルな CORDIC (COordinate Rotation DIgital Computer) アルゴリズムを使用して、平方根回路をインプリメントします。

入力 x から、出力 $\text{sqrt}(x)$ が算出されます。CORDIC プロセッサのインプリメントには、ザイリンクス ブロックセットのブロックが使用されます。

平方根は、CORDIC アルゴリズムで次の式を使用して間接的に計算されます。

$$\text{sqrt}(w) = \text{sqrt} \{ (w + 0.25)^2 - (w - 0.25)^2 \}$$

このアルゴリズムは、次の 4 段階でインプリメントされます。

1. 座標回転: CORDIC アルゴリズムでは、正の x 値のみが収束されます。 $x < 0$ の場合、入力データを負でない値に変換します。 $x = 0$ の場合、0 が検出されたことを示すフラグを座標修正段に伝搬します。この回路は、最小の負の値を除く x のすべての値を収束させるように設計されています。
2. 正規化: CORDIC アルゴリズムでは、0.25 ~ 1 である x 値のみが収束されます。正規化では、入力 x を最上位の符号なしビットが 1 になるまで左にシフトします。左シフトしてシフト値が奇数になった場合は、偶数になるように右に 1 つシフトします。シフト値は 2 で除算され、座標修正段に渡します。平方根は、 $\text{sqrt}(w) = \text{sqrt} \{ (w + 0.25)^2 - (w - 0.25)^2 \}$ の式より得られます。この式に基づき、入力 x は $X = x + 0.25$ および $Y = x - 0.25$ に対応付けられます。
3. 双曲線回転: $\text{sqrt}(X^2 - Y^2)$ を算出するため、結果のベクトルを y が 0 に近づくように少しずつ回転します。
4. 座標修正: 入力 x が負で左シフトが適用されている場合、出力に適切な符号を割り当て、 $2^{-\text{shift}}$ で乗算します。入力が 0 の場合は、0 が検出されたことを示すフラグを使用して、出力を 0 に設定します。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

パラメータは次のとおりです。

- [Number of Processing Elements (integer value starting from 1)]: リニア回転での段の反復回数を指定します。
- [Input Data Width]: 入力 x の幅を指定します。入力 x は指定されたデータ幅で、符号付きである必要があります。
- [Input Binary Point Position]: 入力 x の 2 進小数点の位置を指定します。入力 x は 2 進小数点指定された位置にあり、符号付きである必要があります。
- [Latency for each Processing Element [1001]]: 各双曲線回転段の後のパイプライン レイテンシを設定します。

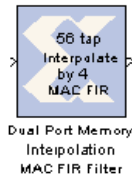
このブロックのレイテンシは、次の式に基づいて算出されます。

$$\begin{aligned} \text{Latency} &= 7 + (\text{data width} - \text{binary point}) \\ &+ \text{mod} \{ (\text{data width} - \text{binary point}), 2 \} \\ &+ \text{sum} (\text{latency of Processing Elements}) \end{aligned}$$

参考資料

1. J. E. Volder、『The CORDIC Trigonometric Computing Technique』、IRE トランザクション、Electronic Computers、Vol. EC-8、1959 年、pp. 330 ～ 334
2. J. S. Walther、『A Unified Algorithm for Elementary Functions』、Spring Joint Computer Conference、1971 年、pp. 379 ～ 385
3. Yu Hen Hu、『CORDIC-Based VLSI Architectures for Digital Signal Processing』、IEEE Signal Processing Magazine、1992 年 7 月、pp. 17 ～ 34

Dual Port Memory Interpolation MAC FIR Filter



積和ベースの FIR フィルタをインプリメントし、ユーザーが選択可能なインターポレーションを実行します。n-tap フィルタには、専用乗算器 1 つとデュアルポートブロック RAM 1 つが使用されます。このフィルタ コンフィギュレーションでは、1 つのブロック RAM に係数およびデータ サンプルを格納する循環 RAM バッファの手法が示されます。このフィルタを使用すると、インターポレーション係数をユーザーが選択できます。Virtex FPGA ファミリ (および Virtex ファミリの派生デバイス) では、高速で小型の加算器、乗算器、および柔軟なメモリ アーキテクチャを構築するための専用回路が提供されます。フィルタ デザインではこれらのシリコンの特性が活用され、小型でリソース効率のよいデザインがインプリメントされます。

このフィルタ リファレンス ブロックには、インプリメンテーションの説明が提供されています。説明を読むには、ブロックをモデル内に配置し、そのブロックを右クリックして、[Explore] を選択します。サブブロックの 1 つをダブルクリックすると、そのサブブロックのモデルが開き、説明が読めるようになります。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

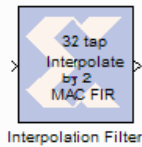
パラメータは次のとおりです。

- [Data Input Bit Width] : 入力サンプルの幅
- [Data input Binary Point] : 入力の 2 進小数点の位置
- [Coefficients] : フィルタの係数を指定します。タップ数は、係数ベクタのサイズから推論されます。
- [Number of Bits per Coefficient] : 各係数のビット幅
- [Binary Point Per Coefficient] : 各係数の 2 進小数点の位置
- [Interpolation Ratio] : フィルタのインターポレーション レートを選択します (2 ~ 10)。
- [Sample Period] : 入力のサンプル周期

参考資料

J. Hwang, J. Ballagh, 『Building Custom FIR Filters Using System Generator』、12th International Field-Programmable Logic and Applications Conference (FPL) (2002 年 9 月、モンペリエ (フランス) にて開催)、LNCS (Lecture Notes in Computer Science) Vol. 2438

Interpolation Filter



積和ベースの FIR フィルタをインプリメントし、ユーザーが選択可能なインターポレーションを実行します。n-tap フィルタには、専用乗算器 1 つとデュアルポートブロック RAM 1 つが使用されます。このフィルタ コンフィギュレーションでは、1 つのブロック RAM に係数およびデータ サンプルを格納する循環 RAM バッファの手法が示されます。このフィルタを使用すると、インターポレーション係数をユーザーが選択できます。Virtex FPGA ファミリ (および Virtex ファミリの派生デバイス) では、高速で小型の加算器、乗算器、および柔軟なメモリ アーキテクチャを構築するための専用回路が提供されます。フィルタ デザインではこれらのシリコンの特性が活用され、小型でリソース効率のよいデザインがインプリメントされます。

このフィルタ リファレンス ブロックには、インプリメンテーションの説明が提供されています。説明を読むには、ブロックをモデル内に配置し、そのブロックを右クリックして、[Explore] を選択します。サブブロックの 1 つをダブルクリックすると、そのサブブロックのモデルが開き、説明が読めるようになります。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

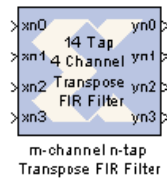
パラメータは次のとおりです。

- [Input Data Bit Width] : 入力サンプルの幅
- [Input Data Binary Point] : 入力の 2 進小数点の位置
- [Coefficients] : フィルタの係数を指定します。タップ数は、係数ベクタのサイズから推論されます。
- [Number of bits per Coefficient] : 各係数のビット幅
- [Binary Point per Coefficient] : 各係数の 2 進小数点の位置
- [Interpolation Factor] : フィルタのインターポレーション レートを選択します (2 ~ 10)。
- [Sample Period] : 入力のサンプル周期

参考資料

J. Hwang, J. Ballagh, 『Building Custom FIR Filters Using System Generator』、12th International Field-Programmable Logic and Applications Conference (FPL) (2002 年 9 月、モンペリエ (フランス) にて開催)、LNCS (Lecture Notes in Computer Science) Vol. 2438

m-channel n-tap Transpose FIR Filter



このフィルタでは、時分割多重化された、完全にパラレルなアーキテクチャが使用されます。Virtex FPGA ファミリ (および Virtex ファミリの派生デバイス) には SRL16E と呼ばれる専用シフトレジスタ回路が含まれており、マルチチャネルアーキテクチャを最適にインプリメントするために使用されます。時分割のマルチプレクサおよびデマルチプレクサを、インプリメントするかしないかを指定できます。乗算器には、エンベデッド乗算器が使用されます。

モデルはダイナミックに構築されるため、係数の数が変化することによって、下位の構造も変化します。

このフィルタリファレンスブロックには、インプリメンテーションの説明が提供されています。説明を読むには、ブロックをモデル内に配置し、そのブロックを右クリックして、[Explore] を選択します。サブブロックの 1 つをダブルクリックすると、そのサブブロックのモデルが開き、説明が読めるようになります。

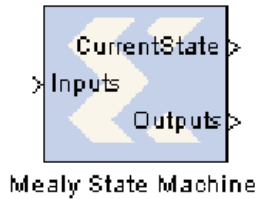
ブロック パラメータ

ブロックのパラメータダイアログボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

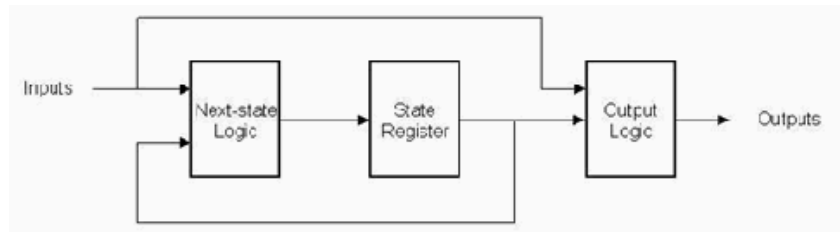
パラメータは次のとおりです。

- [Input Bit Width] : 入力サンプルの幅
- [Input Binary Point] : 入力の 2 進小数点の位置
- [Coefficients] : フィルタの係数を指定します。タップ数は、係数ベクタのサイズから推論されます。
- [Coefficients Bit Width] : 各係数のビット幅
- [Coefficients Binary Point] : 各係数の 2 進小数点の位置
- [Number of Channels] : チャネル数を指定します。サポートされる数には制限はありません。
- [Time Division Multiplexer Front End] : TDM フロントエンド回路をインプリメントするかしないかを指定します (入力データが時分割多重化されている場合)。
- [Time Division DeMultiplexer Back End] : TDD バックエンド回路をインプリメントするかしないかを指定します (出力を時分割多重化する場合)。これは、フィルタが別のマルチチャネル構造に接続している場合に便利です。
- [Input Sample Period] : 入力のサンプル周期

Mealy State Machine



ミーリ マシンは、出力がステート遷移の関数 (マシンの現在のステートと入力の関数) である有限ステート マシンです。次のブロック図で表されます。



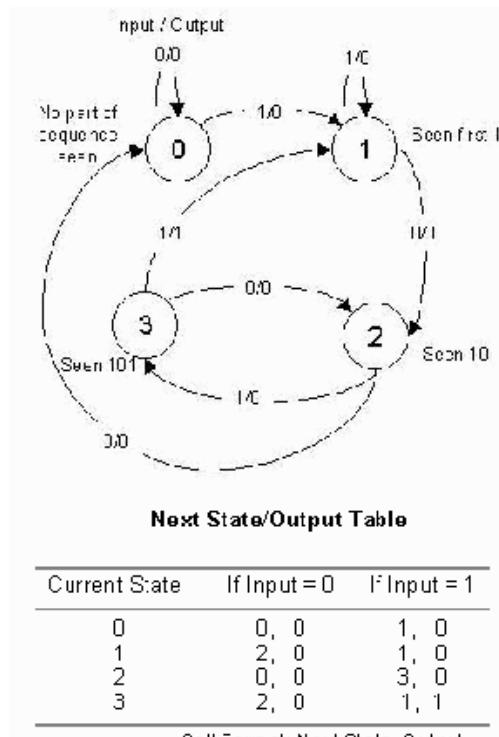
System Generator では、さまざまな方法でこのようなステート マシンをインプリメントできます。一例としては、MCode ブロックを使用して遷移関数をインプリメントし、レジスタを使用してステート変数をインプリメントする方法があります。このリファレンスブロックでは、ブロック メモリおよび分散メモリを使用してミーリ マシンをインプリメントします。インプリメンテーションは、高速で効率的です。たとえばザイリンクスの Virtex デバイスでは、150MHz 以上で動作する1つのブロック RAM を使用して、8 ステート、1 入力、2 出力のレジスタ付きステート マシンを実現できます。

遷移関数と出力の対応は、それぞれ $N \times M$ マトリックスで表されます。ここで N はステート数、 M は入力英字のサイズ (2 進入力の場合は $M = 2$) です。行数を $0 \sim N - 1$ 、列数を $0 \sim M - 1$ にすると便利です。各ステートは $0 \sim N - 1$ の符号なし整数として、各英字は $0 \sim M - 1$ の符号なし整数として表されます。各マトリックスの行インデックスは現在のステートを示し、列インデックスは入力文字を示します。

F を $N \times M$ 遷移関数マトリックス、 O を $N \times M$ 出力関数マトリックス、現在のステートを i 、現在の入力文字を j とすると、 $F(i,j)$ は次のステート、 $O(i,j)$ はミーリ マシンの対応する出力です。

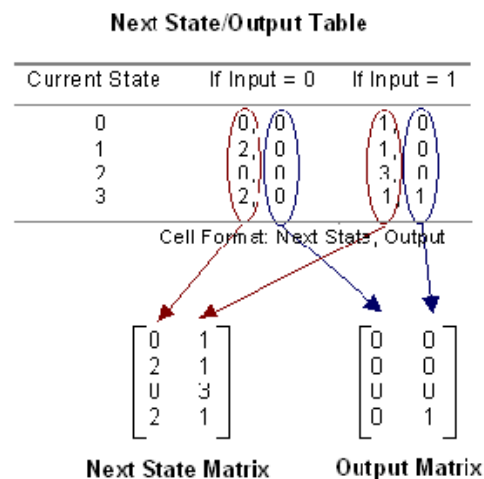
例

ミーリ マシンを設計して、シリアル ストリーム ビットの 1011 というパターンを認識させる場合を考えてみます。状態遷移図およびこれに対応する遷移表は次のようになります。



表には、現在のステートおよび入力を基にした、次のステートおよび出力がリストされています。たとえば、現在のステートが 3 で 入力 が 1 の場合は、次のステートは 2 および出力は 1 となり、正しいシーケンスが示されています。

Mealy State Machine ブロックは、上記の次のステート/出力の表より得られる、次のステートと出力マトリックスでコンフィギュレーションされます。これらのマトリックスは、次のように構成されています。



ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、**Simulink** モデル内でブロックをダブルクリックすると表示されます。

次のステート ロジック、ステート レジスタ、および出力ロジックは、高速専用ブロック **RAM** を使用してインプリメントされます。出力ロジックはルックアップ テーブルとしてコンフィギュレーションされた専用 **RAM** を使用してインプリメントされるため、レイテンシは **0** です。

ミーリ ステート マシンのインプリメントに使用されるビット数は、次の式で算出されます。

$$\text{depth} = (2k)(2i) = 2k+i$$

$$\text{width} = k+o$$

$$N = \text{depth} * \text{width} = (k+o)(2k+i)$$

この場合、各値の定義は次のとおりです。

N = ブロック **RAM** ビット数の合計

s = ステート数

$$k = \text{ceil}(\log_2(s))$$

i = 入力ビット数

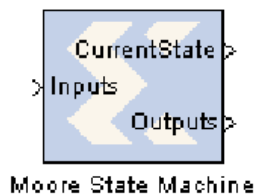
o = 出力ビット数

次の表に、さまざまなステート マシンに必要なブロック **RAM** サイズの例を示します。

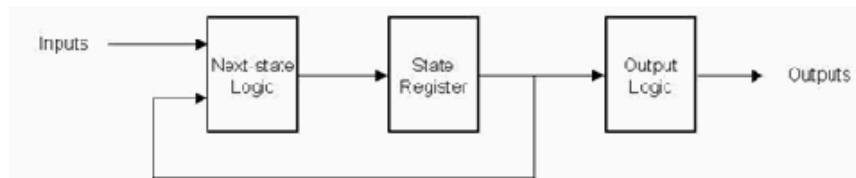
ステート数	入力ビット数	出力ビット数	必要なブロック RAM のビット数
2	5	10	704
4	1	2	32
8	6	7	5120
16	5	4	4096
32	4	3	4096
52	1	11	2176
100	4	5	24576

ブロック **RAM** の幅およびワード数の制限は、シングル ポート **RAM** ブロックのオンライン ヘルプ で説明されています。

Moore State Machine



ムーアマシンは、出力がマシンの現在のステートの関数である有限ステートマシンです。次のブロック図で表されます。



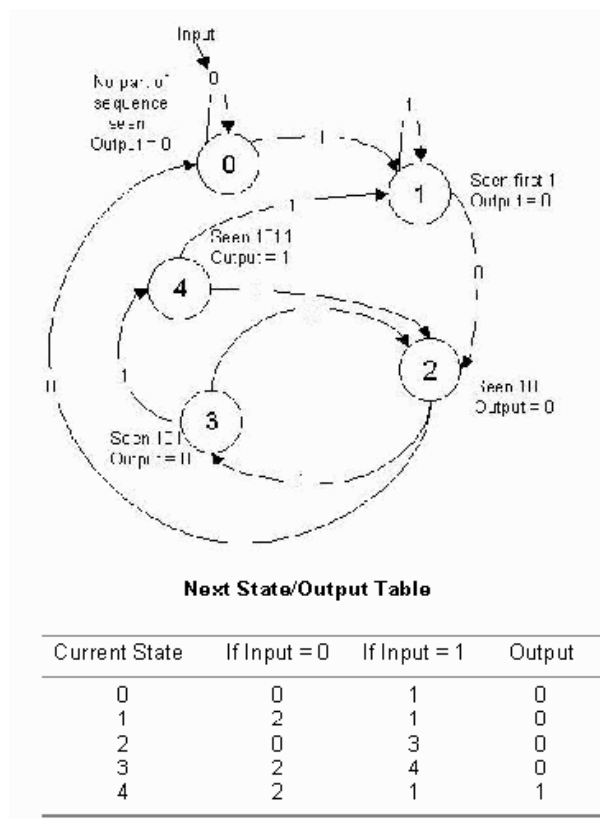
System Generator では、さまざまな方法でこのようなステートマシンをインプリメントできます。一例としては、MCode ブロックを使用して遷移関数をインプリメントし、レジスタを使用してステート変数をインプリメントする方法があります。このリファレンスブロックでは、ブロックメモリおよび分散メモリを使用して Moore マシンをインプリメントします。インプリメンテーションは、高速で効率的です。たとえばザイリンクスの Virtex デバイスでは、150MHz 以上で動作する 1 つのブロック RAM を使用して、8 ステート、1 入力、2 出力のレジスタ付きステートマシンを実現できます。

遷移関数と出力の対応は、それぞれ $N \times M$ マトリックスで表されます。ここで N はステート数、 M は入力値の数 (1 ビット入力の場合は $M = 2$) です。行数を $0 \sim N - 1$ 、列数を $0 \sim M - 1$ にすると便利です。各ステートは $0 \sim N - 1$ の符号なし整数として、各英字は $0 \sim M - 1$ の符号なし整数として表されます。各マトリックスの行インデックスは現在のステートを示し、列インデックスは入力文字を示します。

F を $N \times M$ 遷移関数マトリックス、 O を $N \times M$ 出力関数マトリックス、現在のステートを i 、現在の入力文字を j とすると、 $F(i,j)$ は次のステート、 $O(i,j)$ は Moore マシンの対応する出力です。

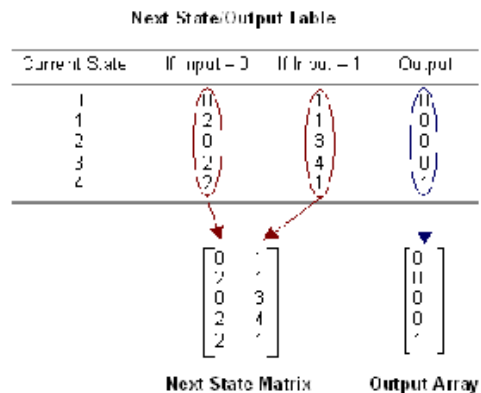
例

Moore マシンを設計して、シリアル ストリームのビットの 1011 というパターンを認識させる場合を考えてみます。状態遷移図およびこれに対応する遷移表は次のようになります。



表には、現在のステートおよび入力を基にした、次のステートおよび出力がリストされています。たとえば、現在のステートが 4 の場合は 出力が 1 となり、正しいシーケンスが示されています。入力 が 1 の場合、次のステートは 1 となります。

レジスタ付きの **Moore State Machine** ブロックは、上記の次のステート/出力の表より得られる、次のステート マトリックスと出力配列でコンフィギュレーションされます。これらのマトリックスは、次のように構成されています。



マトリックスの行は、現在のステートに対応します。次のステートのマトリックスでは、各入力値が 1 つの列に示されています。入力値はステート マシンの出力に影響しないため、出力配列は 1 列のみです。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、**Simulink** モデル内でブロックをダブルクリックすると表示されます。

このブロックの次のステート ロジックおよびステート レジスタは、高速専用ブロック **RAM** を使用してインプリメントされます。出力ロジックはルックアップ テーブルとしてコンフィギュレーションされた専用 **RAM** を使用してインプリメントされるため、レイテンシは **0** です。

Moore ステート マシンのインプリメントに使用されるビット数は、次の式で算出されます。

$$d_s = (2^k)(2^i) = 2^{k+i}$$

$$w_s = k$$

$$N_s = d_s * w_s = (k)(2^{k+i})$$

この場合、各値の定義は次のとおりです。

N_s = 次のステート ロジック ブロック **RAM** ビット数の合計

s = ステート数

$k = \text{ceil}(\log_2(s))$

i = 入力ビット数

d_s = ステート ロジック ブロック **RAM** のワード数

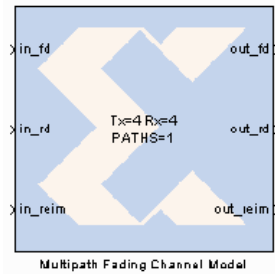
w_s = ステート ロジック ブロック **RAM** の幅

次の表に、さまざまなステート マシンに必要なブロック **RAM** サイズの例を示します。

ステート数	入力ビット数	必要なブロック RAM のビット数
2	5	64
4	1	8
8	6	1536
16	5	2048
32	4	2560
52	1	768
100	4	14336

ブロック **RAM** の幅およびワード数の制限は、シングル ポート ブロック メモリのコア データシートで説明されています。

Multipath Fading Channel Model



フェージング通信チャネルのモデルをインプリメントします。Single Input/Single Output (SISO) と Multiple Input/Multiple Output (MIMO) の両方のチャネルをサポートします。Simulink の Multipath Rayleigh Fading Channel ブロックに似たファンクションがハードウェアで実現できる形で提供されるため、通信リンク全体のハードウェア協調シミュレーションが高速に実行できます。

概念

このブロックでは、Kronecker モデルがインプリメントされます。このモデルはエレメント数が 4 個以内のアンテナ配列を使用するシステムに適しています。モデルの基本的なパラメータは次のとおりです。

- **MT**: 送信配列内のアンテナ数。SISO システムでは 1 です。
- **MR**: 受信配列内のアンテナ数。SISO システムでは 1 です。
- **N**: 配列間の離散パスの数。周波数平面流路では、1 です。

このモデルは、次の離散時間の論理式で表されます。

$$y(nT) = \sum_{k=1}^N g_k H_k(nT) x(nT - d_k)$$

この場合、各値の定義は次のとおりです。

- **x(.)**: 送信シンボル列ベクタ (MT 複合エレメント、時間で変化)
- **T**: サンプル間隔
- **n**: サンプル インデックス
- **d_k**: パス k の遅延
- **H_k(.)**: チャネル係数マトリックス (MR × MT 複合エレメント、時間で変化)
- **g_k**: パス k の利得
- **y(.)**: 受信シンボル列ベクタ (MR 複合エレメント、時間で変化)

チャネル係数マトリックスではアンテナ配列の空間的な共分散マトリックスをさらに詳細に定義できます。

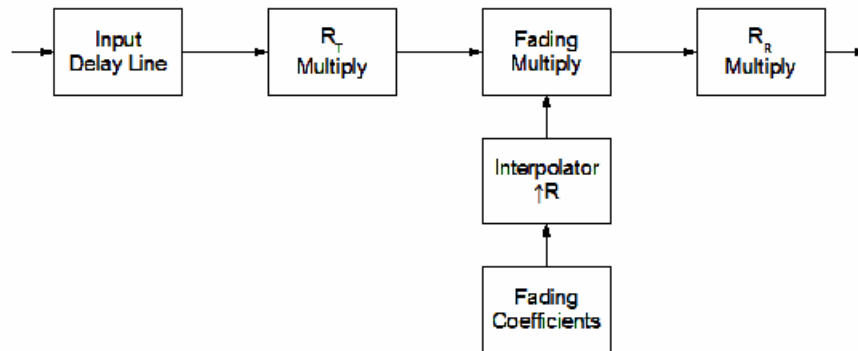
$$H_k(nT) = R_{T,k}^{1/2} H_{U,k}(nT) R_{R,k}^{1/2}$$

この場合、各値の定義は次のとおりです。

- **R_{T,k}**: パス k の送信配列空間的共分散マトリックス
- **H_{U,k}(.)**: パス k の無相関チャネル係数マトリックス (M_R × M_T エレメント、時間で変化)
- **R_{R,k}**: パス k の受信配列空間的共分散マトリックス

インプリメンテーション

上記の式は、疎行列操作であると言えるので、パス合計を削除できます。その後、モデルは次のようにインプリメントされます。



ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

[Paths] タブ

[Path] タブからは、次のようなパラメータを設定できます。

- **[Path Delay Vector]** : モデルの各パスの遅延拡散を指定します。各エレメントは、パスを遅延させるサンプル数を表します。値は N エレメント ベクタである必要があります。
- **[Path Gain Vector]** : モデルの各パスの利得を指定します。各エレメントは、パスのリニア利得を表します。値は N エレメント ベクタである必要があります。

[Covariance] タブ

周波数選択性チャネル ($N > 1$) をサポートするために、次のパラメータを 3 次元配列として指定します。最初の 2 つの次元では方形の共分散マトリックスが、3 番目の次元ではパスが指定されます。周波数選択性チャネルに 2 次元配列が指定された場合は、自動的に複製されて 3 次元配列が生成されます。周波数平面流路 ($N = 1$) では、3 番目の次元はオプションです。

- **[Transmit Array Spatial Covariance Matrices]** : 各パスの送信アンテナ配列共分散マトリックスを指定します。値は $M_T \times M_T$ マトリックスまたは $M_T \times M_T \times N$ 配列です。
- **[Receive Array Spatial Covariance Matrices]** : 各パスの受信アンテナ配列共分散マトリックスを指定します。値は $M_R \times M_R$ マトリックスまたは $M_R \times M_R \times N$ 配列です。

[Fading] タブ

- **[Spectrum Data]** : 各物理パスの、フェージング位相および周波数応答を指定します。物理パス数は離散パス数 (N) と、送信/受信アンテナ配列 ($M_T \times M_R$) のエレメント間のパス数の積です。 $M_R \times M_T \times N$ の次元の多次元構造である必要があります。

- [Rate] : 最大ドップラー周波数 (FDMAX) からチャンネル サンプル周波数 (FS) までのインターポレーション レートを指定します。次のように算出されます。

$$R = \left\lceil \frac{FS}{(256 \cdot F_{DMAX})} \right\rceil$$

[Internal] タブ

- [Datapath Width in Bits] : すべての内部データパスの幅を指定します。
- [Transmit Multiply Binary Point] : RT 乗算ブロックの出力での 2 進小数点の位置を指定します。
- [Fading Multiply Binary Point] : フェージング乗算ブロックの出力での 2 進小数点の位置を指定します。
- [Receive Multiply Binary Point] : RR 乗算ブロックの出力での 2 進小数点の位置を指定します。
- [Covariance Matrix Binary Point] : 共分散マトリックス係数の 2 進小数点の位置を指定します。
- [Random Seed] : 位相ノイズ乱数発生器の 61 ビット (16 桁の 16 進数) シードを指定します。

関数

モデルには、パラメータ生成用に、次の 2 つの MATLAB 関数が含まれています。

create_r_la

create_r_la(M,P,phi0,d,lambda,AS) 関数では、参考資料 1 に記述されているように、操作ベクタから共分散マトリックスが生成されます。

- M : 配列内のアンテナ数を指定します (送信または受信)。
- P : マトリックスの生成用に統合するランダム パスの数を指定します (50,000 で良好な結果が得られます)。
- phi0 : 送信 (送信配列) または 受信 (受信配列) の平均角度を、ラジアンで指定します。
- d : アンテナ間隔を、ベースラインに沿ったアンテナ位置のベクタとして指定します。この値がスカラ値として指定された場合は、エレメントがベースラインの起点の周囲に等間隔に分散された等間隔リニア アレイ (ULA) と見なされます。
- lambda : 波長をメートルで指定します。
- AS : 平均角度の角拡散をラジアンで指定します。

たとえばエレメント間隔が 2GHz で $\lambda/2$ 、角拡散が 15° の 3 つのエレメントの ULA にマトリックスを作成する場合は、次のようになります。

```
lambda=2.0e9/2.99e8;
create_r_la(3,50000,0,lambda/2,lambda,15*(2*pi/360))
```

calc_path_data

calc_path_data(spec_type,spec_fd) 関数では、モデルのスペクトル データが生成されます。

- spec_type : モデルの各物理パスのスペクトル タイプを指定します。この値は、MR × MT × N の多次元配列である必要があります。各エレメントで、物理パスのスペクトル タイプが指定されます。

- **spec_fd** : 各物理パスに、最大ドップラー周波数 (FDMAX) に正規化された、スペクトルドップラー周波数を指定します。この値には $MR \times MT \times N$ の多次元配列またはスカラー値を指定でき、スカラー値の場合はすべての物理パスに値が適用されます。省略された場合は、1 と見なされます。

各スペクトルタイプエレメントの値により、その物理パスに使用されるスペクトル形状が指定されます。次の4つのスペクトルタイプがサポートされています。

- **Type 0** : 空の物理パス。パス係数は 0 で、パスでは送信が実行されません。
- **Type 1** : インパルス物理パス。インパルスパスのスペクトルには 1 つのインパルスがあります。これらを使用して、Rician チャンネルの要件とされるようなチャンネルモデルで LOS (line-of-sight) パスを表すことができます。
- **Type 2** : 典型的なスペクトルの物理パス。典型的なスペクトルは、Jakes スペクトルまたは Clarke スペクトルとしても知られます。これは参考資料 2、3、4 の移動局とのワイヤレスリンクのモデル化に使用され、次のように定義されます。

$$S(f) \propto \frac{1}{\sqrt{1 - \left(\frac{f}{f_d}\right)^2}} \quad |f| \leq f_d$$

$$S(f) = 0 \quad \text{elsewhere}$$

- **Type 3** : 丸められたスペクトルの物理パス。参考資料 5 の固定局とのワイヤレスリンクのモデル化に使用され、次のように定義されます。

$$S(f) \propto 1 - 1.72 \left(\frac{f}{f_d}\right)^2 + 0.785 \left(\frac{f}{f_d}\right)^4 \quad |f| \leq f_d$$

$$S(f) = 0 \quad \text{elsewhere}$$

生成されると、各スペクトルは単一の指数に正規化されます。

たとえば、 $M_T=4$ のスペクトルデータを作成してプロットするには、 $M_R=3$ および $N=2$ チャンネルとなります。この場合は 2 つのパスが組み合わさり、インパルスおよび典型などの Rician フェージングが発生します。移動局 (MS) は基地局から $0.707 \times v_{MS}$ (LOS 物理パスでは $fd=0.707$) で遠ざかるものとします。

```
Mt=4; Mr=3; N=2;
spec_type=cat(3,ones(Mr,Mt)*1,ones(Mr,Mt)*2);
spec_fd =cat(3,ones(Mr,Mt)*0.707,ones(Mr,Mt)*1);
spec_data=calc_path_data(spec_type,spec_fd);
plot([spec_data.spectrum]);
```

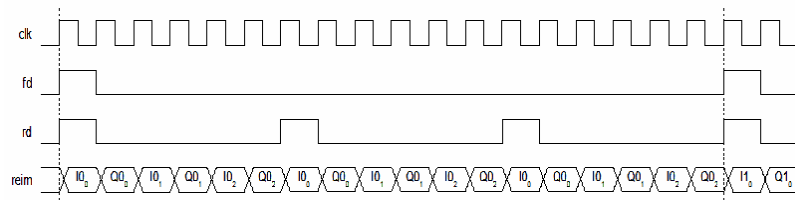
データ形式

内部的には、複素ベクタのブロック間での伝送に、3 信号のインターフェイスがモデルで使用されます。このインターフェイスを使用すると、マトリックス/ベクタ操作がチェーン接続されます。ベクタは、フレームおよび反復ハンドシェイク信号が設定された、インターリーブされた実サンプルおよび虚サンプルのストリームとして伝送されます。このインターフェイスを使用すると、フレーム当たり複数回ベクタを繰り返すことができます。この機能を使用すると、マトリックス行 1 行当たり 1 度だけ使用されるベクタ値が何度も必要となる場合に、マトリックスとベクタの乗算を単純化できます。

3つの信号インターフェイスを次に示します。

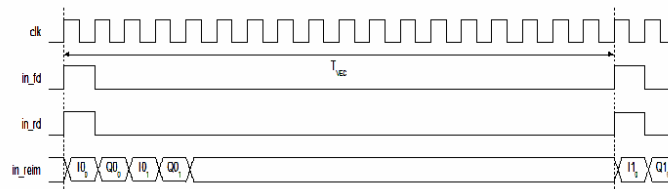
- **reim** : 各ベクタの、インターリーブされた実サンプル (**I**) および虚サンプル (**Q**) のストリーム。信号で示されるように、各ベクタは複数回伝送される可能性があります。
- **fd** : 各ベクタ フレームの開始を示します。
- **rd** : 各ベクタ反復の開始を示します。

次の図に、3 エLEMENTのベクタが 3×3 マトリックスでの乗算の前はどのように表されるかを示します。ベクタは3回(マトリックス行ごとに1回)反復するため、乗算ロジックが簡略化されます。



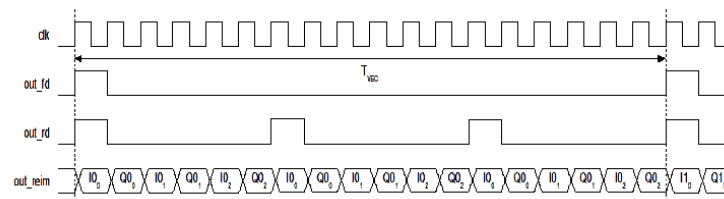
入力

入力データは in_fd、in_rd、および in_reim ポートに示されます。入力時にはベクタを反復する必要はないため、in_rd signal は無視され、最初の 2 つの MT サンプルのみが使用されます。たとえば、MT=2 チャネルでは次のようになります。



出力

出力データは `out_fd`、`out_rd`、および `out_reim` ポートに示されます。データは、フレーム全体で反復されます。たとえば、**MR=3** チャンネルでは次のようになります。



タイミング制約

一連の fd パルス (TVEC) 間には、内部ブロックでのデータの処理に十分なサンプル数が必要です。各ブロックに必要なサイクル数は、次に示す、 MT 、 MR 、 N 、および $RATE$ パラメータの関数です。

RT 乗算 : $2 \times MT \times MT \times N$ サイクル必要

フェージング乗算 : $2 \times MT \times MR \times N \times \text{ceil}(64/RATE)$ サイクル必要

RR 乗算 : $2 \times MR \times MR \times N$ サイクル必要

TVEC の最小値は次のようになります。

$$T_{VEC} = 2N \max(\max(M_T, M_R)^2, M_T M_R \lceil 64/(RATE) \rceil)$$

この制約が満たされない場合、シミュレーション中にモデルでエラーが発生します。

初期化

モデルでは、フェージング係数ジェネレータの初期化におよそ $3 \times R$ の入力フレームを必要とします。初期化中は、チャネル係数および出力データは 0 になります。

デモンストレーション

モデルの使用方法を示す次の 2 つのデモがあります。どちらにも、パラメータの計算方法が示されています。

- SISO チャネル モデル : 3GPP TS 25.104、Annex B.2、Case 4 の SISO チャネルのデモ
- MIMO チャネル モデル : 周波数平面 MIMO 流路のデモ

ハードウェア協調シミュレーションの例

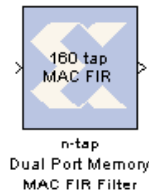
ハードウェア協調シミュレーションでモデルを使用する方法の例が、<sysgen_tree>/examples/mfcm_hwcossim ディレクトリに含まれています。このディレクトリには次の 3 つのファイルが含まれています。

- mfcm_hw.mdl : 協調シミュレーション デザインのハードウェア コンポーネントを指定するモデルです。デザインは、データ入力用の共有メモリ、チャネル モデル、およびデータ出力用の共有メモリより構成されています。
- mfcm_hw_cw.bit : mfcm_hw.mdl デザインを XtremeDSP キット用にコンパイルしたものです。
- mfcm_cossim.mdl : 協調シミュレーションのソフトウェア コンポーネントを指定するモデルです。共有メモリ ブロックを使用して、データの packets がハードウェアで処理するために渡され、処理済みのデータの packets が受信されます。このデザインでは、生成済みの mfcm_hw_cw.bit がデフォルトで使用されます。ハードウェア ターゲット が異なる場合は、生成し直す必要があります。

参考資料

1. A. Forenza, R.W. Heath Jr., 『Impact of Antenna Geometry on MIMO Communication in Indoor Clustered Channels』, Wireless Networking and Communications Group, ECE Department, The University of Texas at Austin
2. 3GPP TS 25.101 V6.7.0 (2005-03), 『Annex B, User Equipment (UE) radio transmission and reception (FDD)』, Technical Specification Group Radio Access Network, 3rd Generation Partnership Project
3. 3GPP TS 25.104 V6.8.0 (2004-12), 『Annex B, Base Station (BS) radio transmission and reception (FDD)』, Technical Specification Group Radio Access Network, 3rd Generation Partnership Project
4. 3GPP TR 25.943 V6.0.0 (2004-12), 『Deployment aspects』, Technical Specification Group Radio Access Network, 3rd Generation Partnership Project
5. IEEE 802.16.3c-01/29r4 (2001-07-16), 『Channel Models for Fixed Wireless Applications』, IEEE 802.16 Broadband Wireless Access Working Group

n-tap Dual Port Memory MAC FIR Filter



積和ベースの FIR フィルタをインプリメントします。フィルタでは、専用乗算器 1 つとデュアルポートブロック RAM 1 つが使用されます。このフィルタ コンフィギュレーションでは、係数およびデータ サンプルをフィルタ デザインに格納する方法が示されます。Virtex FPGA ファミリ (および Virtex ファミリの派生デバイス) では、高速で小型の加算器、乗算器、および柔軟なメモリ アーキテクチャを構築するための専用回路が提供されます。フィルタ デザインではこれらのシリコンの特性が活用され、小型でリソース効率のよいデザインがインプリメントされます。

このフィルタ リファレンス ブロックには、インプリメンテーションの説明が提供されています。説明を読むには、ブロックをモデル内に配置し、そのブロックを右クリックして、[Explore] を選択します。サブブロックの 1 つをダブルクリックすると、そのサブブロックのモデルが開き、説明が読めるようになります。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

パラメータは次のとおりです。

- [Data Input Bit Width] : 入力サンプルの幅
- [Data input Binary Point] : 入力の 2 進小数点の位置
- [Coefficients] : フィルタの係数を指定します。タップ数は、係数ベクタのサイズから推論されます。
- [Number of Bits per Coefficient] : 各係数のビット幅
- [Binary Point Per Coefficient] : 各係数の 2 進小数点の位置
- [Sample Period] : 入力のサンプル周期

参考資料

J. Hwang, J. Ballagh, 『Building Custom FIR Filters Using System Generator』、12th International Field-Programmable Logic and Applications Conference (FPL) (2002 年 9 月、モンペリエ (フランス) にて開催)、LNCS (Lecture Notes in Computer Science) Vol. 2438

n-tap MAC FIR Filter



積和ベースの **FIR** フィルタをインプリメントします。3 とおりのフィルタ コンフィギュレーションにより、フィルタのスループットとデバイスのリソース使用のトレードオフが示されます。**Virtex FPGA** ファミリ (および **Virtex** ファミリの派生デバイス) では、高速で小型の加算器、乗算器、および柔軟なメモリ アーキテクチャを構築するための専用回路が提供されます。フィルタ デザインではこれらのシリコンの特性が活用され、小型でリソース効率のよいデザインがインプリメントされます。

このフィルタ リファレンス ブロックには、インプリメンテーションの説明が提供されています。説明を読むには、ブロックをモデル内に配置し、そのブロックを右クリックして、**[Explore]** を選択します。サブブロックの 1 つをダブルクリックすると、そのサブブロックのモデルが開き、説明が読めるようになります。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、**Simulink** モデル内でブロックをダブルクリックすると表示されます。

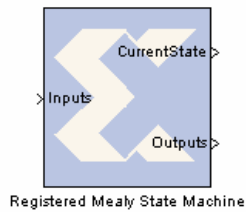
パラメータは次のとおりです。

- **[Coefficients]** : フィルタの係数を指定します。タップ数は、係数ベクタのサイズから推論されます。
- **[Number of Bits per Coefficient]** : 各係数のビット幅
- **[Binary Point for Coefficient]** : 各係数の 2 進小数点の位置
- **[Number of Bits per Input Sample]** : 入力サンプルの幅
- **[Binary Point for Input Samples]** : 入力の 2 進小数点の位置
- **[Input Sample Period]** : 入力のサンプル周期

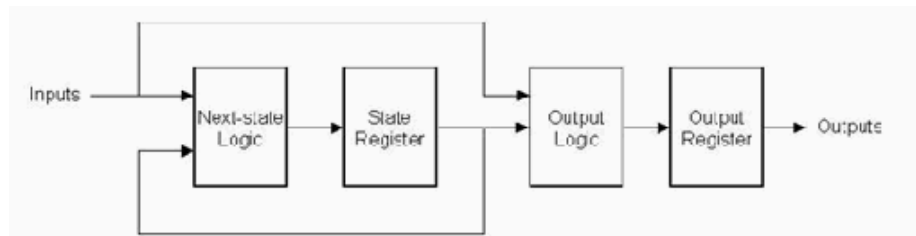
参考資料

J. Hwang, J. Ballagh, 『Building Custom FIR Filters Using System Generator』、12th International Field-Programmable Logic and Applications Conference (FPL) (2002 年 9 月、モンペリエ (フランス) にて開催)、LNCS (Lecture Notes in Computer Science) Vol. 2438

Registered Mealy State Machine



ミーリ マシンは、出力がステート遷移の関数 (マシンの現在のステートと入力の関数) である有限ステート マシンです。レジスタ付きミーリ マシンはレジスタ付き出力を持つステート マシンで、次のブロック図で示されます。



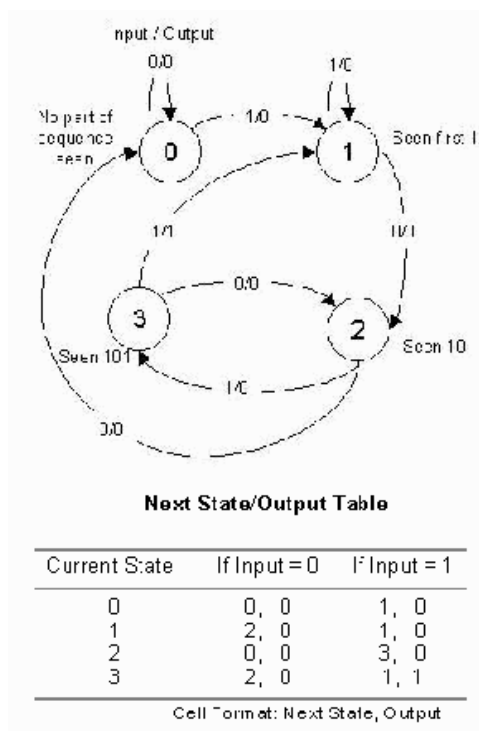
System Generator では、さまざまな方法でこのようなステート マシンをインプリメントできます。一例としては、MCode ブロックを使用して遷移関数をインプリメントし、レジスタを使用してステート変数をインプリメントする方法があります。このリファレンスブロックでは、ブロック メモリおよび分散メモリを使用してミーリ マシンをインプリメントします。インプリメンテーションは、高速で効率的です。たとえばザイリンクスの Virtex デバイスでは、150MHz 以上で動作する 1 つのブロック RAM を使用して、8 ステート、1 入力、2 出力のレジスタ付きステート マシンを実現できます。

遷移関数と出力の対応は、それぞれ $N \times M$ マトリックスで表されます。ここで N はステート数、 M は入力英字のサイズ (2 進入力の場合は $M = 2$) です。行数を $0 \sim N - 1$ 、列数を $0 \sim M - 1$ にすると便利です。各ステートは $0 \sim N - 1$ の符号なし整数として、各英字は $0 \sim M - 1$ の符号なし整数として表されます。各マトリックスの行インデックスは現在のステートを示し、列インデックスは入力文字を示します。

F を $N \times M$ 遷移関数マトリックス、 O を $N \times M$ 出力関数マトリックス、現在のステートを i 、現在の入力文字を j とすると、 $F(i,j)$ は次のステート、 $O(i,j)$ はミーリ マシンの対応する出力です。

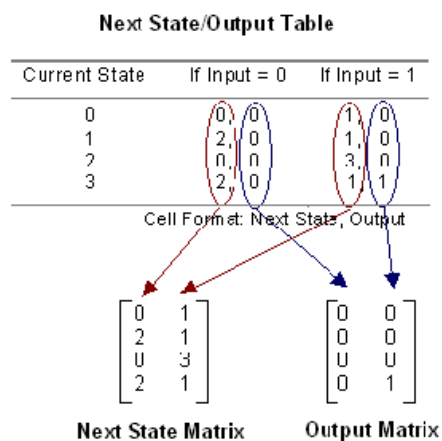
例

ミーリ マシンを設計して、シリアル ストリーム ビットの 1011 というパターンを認識させる場合を考えてみます。状態遷移図およびこれに対応する遷移表は次のようになります。



表には、現在の状態および入力を基にした、次の状態および出力がリストされています。たとえば、現在の状態が 3 で 入力 が 1 の場合は、次の状態は 0 および出力は 1 となり、正しいシーケンスが示されています。

Registered Mealy State Machine ブロックは、上記の次の状態/出力の表より得られる、次の状態と出力マトリックスでコンフィギュレーションされます。これらのマトリックスは、次のように構成されています。



マトリックスの行は状態に、列は入力値に対応しています。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、**Simulink** モデル内でブロックをダブルクリックすると表示されます。

次のステート ロジック、ステート レジスタ、出力ロジック、および出力レジスタは、高速専用ブロック **RAM** を使用してインプリメントされます。ステート マシン ライブラリの 4 つのブロックの中でこれが最も高速で効率的ですが、出力にはレジスタが付けられているため、入力出力には即座には反映されません。

ミーリ ステート マシンのインプリメントに使用されるビット数は、次の式で算出されます。

$$\text{depth} = (2k)(2i) = 2k+i$$

$$\text{width} = k+o$$

$$N = \text{depth} * \text{width} = (k+o)(2k+i)$$

この場合、各値の定義は次のとおりです。

N = ブロック RAM ビット数の合計

s = ステート数

$k = \text{ceil}(\log_2(s))$

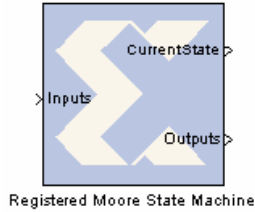
i = 入力ビット数

o = 出力ビット数

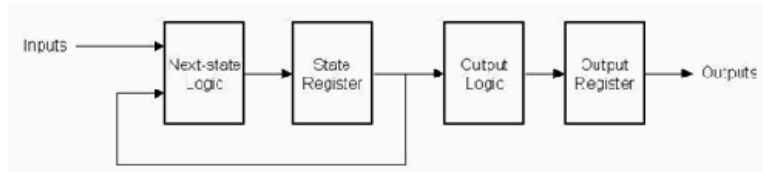
次の表に、さまざまなステート マシンに必要なブロック RAM サイズの例を示します。

ステート数	入力ビット数	出力ビット数	必要なブロック RAM のビット数
2	5	10	704
4	1	2	32
8	6	7	5120
16	5	4	4096
32	4	3	4096
52	1	11	2176
100	4	5	24576

Registered Moore State Machine



ムーア マシンは、出力がマシンの現在のステートの関数である有限ステート マシンです。レジスタ付きムーア マシンはレジスタ付き出力を持つステート マシンで、次のブロック図で示されます。



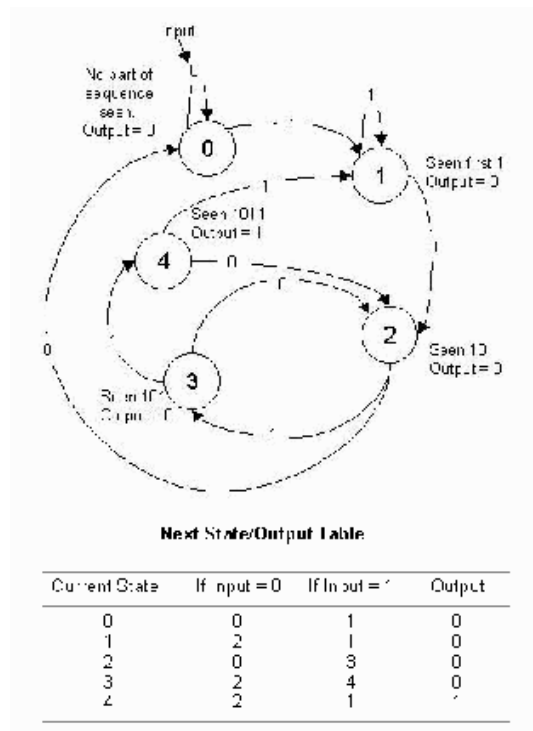
System Generator では、Mcode ブロックを使用するなど、さまざまな方法でこのようなステート マシンをインプリメントできます。このリファレンスブロックでは、ブロック メモリおよび分散メモリを使用して Moore マシンをインプリメントします。インプリメンテーションは、高速で効率的です。たとえばザイリンクスの Virtex デバイスでは、150MHz 以上で動作する 1 つのブロック RAM を使用して、8 ステート、1 入力、2 出力のレジスタ付きステート マシンを実現できます。

遷移関数と出力の対応は、それぞれ $N \times M$ マトリックスで表されます。ここで N はステート数、 M は入力英字のサイズ (2 進入力の場合は $M = 2$) です。行数を $0 \sim N - 1$ 、列数を $0 \sim M - 1$ にすると便利です。各ステートは $0 \sim N - 1$ の符号なし整数として、各英字は $0 \sim M - 1$ の符号なし整数として表されます。各マトリックスの行インデックスは現在のステートを示し、列インデックスは入力文字を示します。

F を $N \times M$ 遷移関数マトリックス、 O を $N \times M$ 出力関数マトリックス、現在のステートを i 、現在の入力文字を j とすると、 $F(i,j)$ は次のステート、 $O(i,j)$ は Moore マシンの対応する出力です。

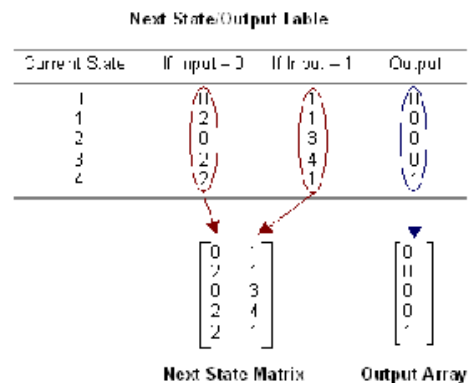
例

Moore マシンを設計して、シリアル ストリーム ビットの 1011 というパターンを認識させる場合を考えてみます。状態遷移図およびこれに対応する遷移表は次のようになります。



表には、現在の状態および入力を基にした、次の状態および出力がリストされています。たとえば、現在の状態が 4 の場合は 出力が 1 となり、正しいシーケンスが示されています。入力が 1 の場合、次の状態は 1 となります。

レジスタ付きの **Moore State Machine** ブロックは、上記の次の状態/出力の表より得られる、次の状態 マトリックスと出力配列でコンフィギュレーションされます。これらのマトリックスは、次のように構成されています。



マトリックスの行は、現在の状態に対応します。次の状態のマトリックスでは、各入力値が 1 つの列に示されています。入力値は状態 マシンの出力に影響しないため、出力配列は 1 列のみです。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、**Simulink** モデル内でブロックをダブルクリックすると表示されます。

このブロックの次のステート ロジックおよびステート レジスタは、高速専用ブロック **RAM** を使用してインプリメントされます。

Moore ステート マシンのインプリメントに使用されるビット数は、次の式で算出されます。

$$d_s = (2^k)(2^i) = 2^{k+i}$$

$$w_s = k$$

$$N_s = d_s * w_s = (k)(2^{k+i})$$

この場合、各値の定義は次のとおりです。

N_s = 次のステート ロジック ブロック **RAM** ビット数の合計

s = ステート数

$k = \text{ceil}(\log_2(s))$

i = 入力ビット数

d_s = ステート ロジック ブロック **RAM** のワード数

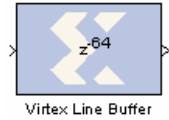
w_s = ステート ロジック ブロック **RAM** の幅

次の表に、さまざまなステート マシンに必要なブロック **RAM** サイズの例を示します。

ステート数	入力ビット数	必要なブロック RAM のビット数
2	5	64
4	1	8
8	6	1536
16	5	2048
32	4	2560
52	1	768
100	4	14336

ブロック **RAM** の幅およびワード数の制限は、シングル ポート ブロック メモリのコア データシートで説明されています。

Virtex Line Buffer



連続したピクセルのストリームを、指定したワード数のバッファ分遅延します。

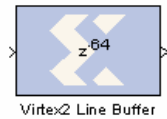
ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、**Simulink** モデル内でブロックをダブルクリックすると表示されます。

パラメータは次のとおりです。

- [Buffer Depth] : ピクセルのストリームを遅延するサンプル数
- [Sample Period] : ブロックが実行されるサンプル レート

Virtex2 Line Buffer



連続したピクセルのストリームを、指定したワード数のバッファ分遅延します。
Virtex-II ファミリに最適化されており、シングルポート RAM ブロックの Read Before Write オプションが使用されます。

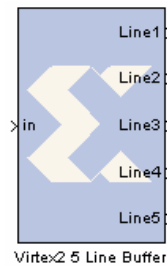
ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

パラメータは次のとおりです。

- [Buffer Depth] : ピクセルのストリームを遅延するサンプル数
- [Sample Period] : ブロックが実行されるサンプル レート

Virtex2 5 Line Buffer



連続したピクセルのストリームをバッファに保管し、5ラインの出力を生成します。各ラインは N サンプル分遅延されます。 N はラインの長さです。ライン 1 は $4*N$ サンプル分遅延され、後続の各ラインの遅延は N サンプル分ずつ短くなっており、ライン 5 は入力のコピーとなっています。

このブロックでは、[Xilinx Reference Blockset] → [Imaging] ライブラリにある Virtex2 Line Buffer ブロックが使用されます。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

パラメータは次のとおりです。

- [Line Size] : 各ラインが遅延されるサンプル数
- [Sample Period] : ブロックが実行されるサンプル レート

White Gaussian Noise Generator

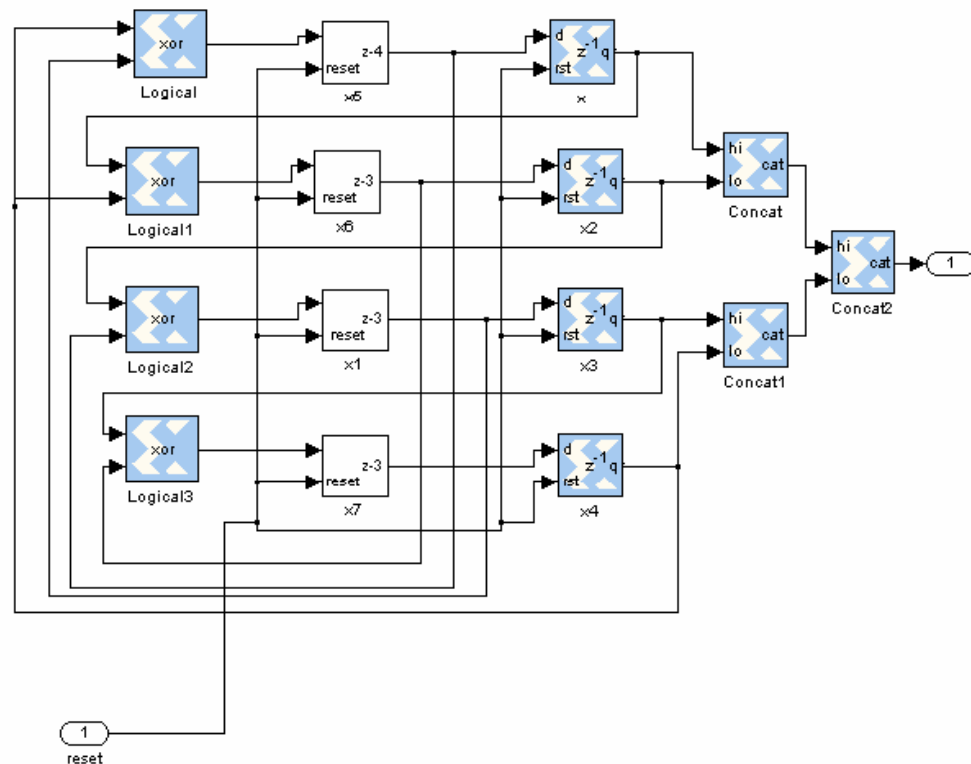


「参考資料」の文献に示されている一般的な方法で、Box-Muller アルゴリズムと中心極限定理を組み合わせて使用し、白色ガウス雑音 (WGNG) を生成します。

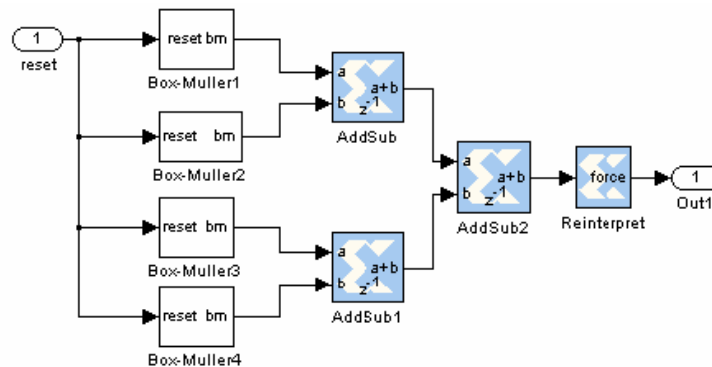
Box-Muller アルゴリズムでは、 $[0,1]$ に一様に分散された、2 つの独立した確率変数の変換を介して単位法線確率変数が生成されます。この生成は、Box-Muller 関数値を ROM に格納し、一様確率変数でアドレスを付けることで実行されます。

一様確率変数は、複数ビットの leap-forward LFSR で生成されます。標準の LFSR では、クロックサイクルごとに 1 出力が生成されますが、K ビット leap-forward LFSR では、1 つのサイクルで k 個の出力を生成できます。たとえば、4 ビットの leap-forward LFSR からは 0 ～ 15 の 離散一様確率変数が出力されます。48 ビット ブロック パラメータ シードの一部により、各 LFSR が初期化されるので、カスタマイズが可能です。パラレル接続された 4 つの Box-Muller サブシステムからの出力は平均化され、確率密度関数 (PDF) が取得されます。これは、0.2% 以内で 4.8σ までのガウス分布です。WGNG の全体的なレイテンシは、10 クロック サイクルとなります。出力ポート ノイズは 12 ビット (2 進小数点以下が 7 ビット) の符号付数値です。

4 ビット Leap-Forward LFSR



Box-Muller 法



ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

[Seed] : 10 進開始シード値です。

参考資料

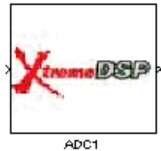
A. Ghazel, E. Boutillon, J. L. Danger, G. Gulak and H. Laamari, 『Design and Performance Analysis of a High Speed AWGN Communication Channel Emulator』, IEEE PACRIM Conference (2001 年 8 月 ブリティッシュ コロンビア州ビクトリア (カナダ) にて開催)

ザイリンクス XtremeDSP キット ブロックセット

XtremeDSP キット ブロックセットには、次のブロックが含まれます。

ライブラリ	説明
XtremeDSP ADC	モデルでハードウェア協調シミュレーションを実行する場合に、System Generator コンポーネントを Nallatech 社製 BenAdda ボードの 2 つのアナログ入力チャンネルに接続できるようにします。
XtremeDSP Co-simulation	XtremeDSP の協調シミュレーション用にコンパイルされた Simulink サブシステムの代わりに使用できます。
XtremeDSP DAC	モデルでハードウェア協調シミュレーションを実行する場合に、System Generator コンポーネントを Nallatech 社製 BenAdda ボードの 2 つのアナログ出力チャンネルに接続できるようにします。
XtremeDSP External RAM	モデルでハードウェア協調シミュレーションを実行する場合に、System Generator コンポーネントを Nallatech 社製 BenAdda ボードの外部 256K × 16 ZBT SRAM に接続できるようにします。
XtremeDSP LED Flasher	モデルでハードウェア協調シミュレーションを実行する場合に、System Generator モデルで BenADDA ボード上の 3 色 LED を使用可能にします。

XtremeDSP ADC



モデルでハードウェア協調シミュレーションを実行する場合に、System Generator コンポーネントを Nallatech 社製 BenAdda ボードの 2 つのアナログ入力チャンネルに接続できるようにします。ADC1 と ADC2 があり、それぞれアナログ入力チャンネル 1 と 2 に接続します。

Simulink では、このブロックは Register ブロックを駆動する Gateway In ブロックを使用してモデル化されています。倍精度信号が入力として受信され、2 進小数点位置が 13 の符号付き 14 ビットのザイリンクス固定小数点信号が出力として生成されます。

ハードウェアでは、ADC ブロックの出力で駆動されるコンポーネントは、BenAdda ボード上にある 2 つの 14 ビット AD6644 AD コンバータ デバイスのいずれかで駆動されます。ADC ブロックを使用する System Generator モデルがハードウェアに変換されると、ADC ブロックはモデル HDL の最上位の入力ポートに変換されます。ポートが ADC コンポーネントにより正しく駆動されるように、適切なピン配置制約が BenAdda 制約ファイルに追加されます。

ハードウェア協調シミュレーション モデルに ADC ブロックが含まれる場合は、フリーランニングクロックを使用する必要があります。また、プログラム可能なクロック スピードは 64MHz 以下に設定する必要があります。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

ADC ブロックのパラメータを次に示します。

- [Sample Period] : ブロックのサンプル周期を指定します。

データシート

AD6644 デバイスのデータシートは、XtremeDSP 開発キットのインストール ディレクトリに含まれています。Nallatech 社製 FUSE ソフトウェアが含まれる FUSE ディレクトリが存在する場合は、データシートは次の場所にあります。

FUSE\XtremeDSP Development Kit\Docs\Datasheets\ADC ad6644.pdf

XtremeDSP Co-simulation



XtremeDSP 協調シミュレーション用にコンパイルされた **Simulink** サブシステムの代わりに使用できます。このブロックは、シミュレーションの実行中はこのブロックの基となるサブシステムとまったく同じ動作をします。唯一の相違点は、シミュレーション データがソフトウェアではなく、ハードウェアで処理されることです。

協調シミュレーション ブロックのポート インターフェイスは多様です。モデルを協調シミュレーション用にコンパイルすると、カスタムの **XtremeDSP** ハードウェア協調シミュレーション ブロックを含む新しいライブラリが作成されます。このブロックには入力および出力ポートがあり、オリジナルのモデルの **Gateway** 名 (サブシステムが最上位レベルでない場合は、ポート名) に対応しています。

ハードウェア協調シミュレーション ブロックは、**Simulink** シミュレーション中は、**XtremeDSP** 開発キット ボードと通信します。ブロックの入力ポートに書き込まれたシミュレーション データはブロックによりハードウェアに渡されます。データが協調シミュレーション ブロックの出力ポートから読み出される場合は、ブロックでハードウェアから適切な値が読み出され、**Simulink** で解釈できるように出力ポートに駆動されます。また、ブロックでは開発キットの開始、コンフィギュレーション、ステップ、終了が自動的に実行されます。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、**Simulink** モデル内でブロックをダブルクリックすると表示されます。

[Basic] タブ

[Basic] タブからは、次のようなパラメータを設定できます。

- **[Clock source]** : シングル ステップまたはフリーランニング クロック ソースを選択できます。シングル ステップ クロックを選択すると、1 度に 1 クロック サイクルずつボードをステップできます。各クロック サイクル ステップは、**Simulink** での一定の時間に相当します。このクロック ソースを使用すると、シミュレーション中の協調シミュレーション ハードウェアのビヘイビアが、そのサブシステムのシミュレーション ビヘイビアに比較して、ビット精度およびサイクル精度となります。シングル ステッピングは不要で、フリーランニング クロックでボードを実行できる場合もあります。この場合、ボードは **Simulink** シミュレーションと非同期で動作します。
- **[Frequency (MHz)]** : フリーランニング クロック モードを選択した場合は、シミュレーション実行中のフリーランニング クロックの動作周波数を指定します。指定されたクロックの周波数は、プログラム可能なオシレータで使用可能な周波数に近い値に丸められます。
メモ : モデルの **FPGA** インプリメンテーションの最大動作周波数を超えないように注意してください。プログラム可能なオシレータの有効な動作周波数を次に示します。
20 MHz、25 MHz、30 MHz、33.33 MHz、40 MHz、45 MHz、50 MHz、60 MHz、66.66 MHz、70 MHz、75 MHz、80 MHz、90 MHz、100 MHz、120 MHz
- **[Card number]** : ハードウェア協調シミュレーションで使用する、**XtremeDSP** 開発キット カードのインデックスを指定します。複数の **XtremeDSP** キット ボードがインストールされている場合を除き、デフォルト値の 1 を使用してください。
- **[Bus]** : **Simulink** シミュレーションの実行中に、協調シミュレーション ブロックと **XtremeDSP** 開発キット ボードの通信で使用するインターフェイスを選択します。**PCI** または **USB** インターフェイスのいずれかを選択します。

- **[Has combinational path]** : ハードウェア協調シミュレーション ブロックの出力ポートから同じブロックの入力ポートへの直接組み合わせフィードバック パス (同じブロックで出力ポートから入力ポートにワイヤで接続されているなど) が必要な場合もあります。出力ポートから入力ポートへの直接フィードバック パスが必要で、デザインに入力ポートから出力ポートへの組み合わせパスが含まれない場合は、このチェック ボックスをオフにすると、デザインでフィードバック パスを使用できるようになります。
- **[Bitstream name]** : XtremeDSP 開発キット ボード用の協調シミュレーション **FPGA** コンフィギュレーション ファイルを指定します。新規の協調シミュレーション ブロックがコンパイル中にインスタンス化されると、このパラメータが自動的に設定され、正しいコンフィギュレーション ファイルが使用されます。ファイルの場所が変わった場合にのみこのパラメータを修正します。

XtremeDSP DAC



モデルでハードウェア協調シミュレーションを実行する場合に、System Generator コンポーネントを Nallatech 社製 BenAdda ボードの 2 つのアナログ出力チャンネルに接続できるようにします。DAC1 と DAC2 があり、それぞれアナログ出力チャンネル 1 と 2 に接続します。

Simulink では、このブロックは Gateway Out ブロックを駆動する Register ブロックによりモデル化されています。DAC 制御信号はすべて、正しく定数に接続されています。DAC ブロックは、2 進小数点の位置が 13 である 14 ビット ザイリンクス固定小数点信号で駆動される必要があります。DAC ブロックでは、倍精度型の信号が出力されます。

ハードウェアでは、DAC ブロック入力を駆動するコンポーネントにより、BenAdda ボードにある 2 つの 14 ビット AD9772A DA コンバータ デバイスのいずれかが駆動されます。DAC ブロックを使用する System Generator モデルがハードウェアに変換されると、DAC ブロックはモデル HDL の最上位の出力ポートに変換されます。出力ポートが DAC ピンを正しく駆動するように、適切なピン配置制約が BenAdda 制約ファイルに追加されます。

ハードウェア協調シミュレーション モデルに DAC ブロックが含まれる場合は、フリーランニング クロックを使用する必要があります。また、プログラム可能なクロック スピードは 64MHz 以下に設定する必要があります。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

DAC ブロックのパラメータを次に示します。

- [Sample Period] : ブロックのサンプル周期を指定します。

データシート

AD9772A デバイスのデータシートは、XtremeDSP 開発キットのインストール ディレクトリに含まれています。FUSE ソフトウェアが含まれる FUSE ディレクトリが存在する場合は、データシートは次の場所にあります。

FUSE\XtremeDSP Development Kit\Docs\Datasheets\DAC AD9772A.pdf

XtremeDSP External RAM



モデルでハードウェア協調シミュレーションを実行する場合に、System Generator コンポーネントを Nallatech 社製 BenAdda ボードの外部 256K × 16 ZBT SRAM に接続できるようにします。

このブロックは、メモリ デバイス用の Simulink シミュレーション モデルです。ブロックのポートは、通常の同期 RAM デバイスのポートと同様に動作します。アドレス ポートは、2 進小数点の位置が 0 である符号なし 18 ビット ザイリンクス固定小数点信号で駆動される必要があります。we ポートはザイリンクス プール信号で駆動される必要があります。data ポートは 16 ビット ザイリンクス固定小数点信号で駆動される必要があります。ブロックは、16 ビット ザイリンクス固定小数点データ値を出力ポートに駆動します。

ハードウェアでは、Simulink のブロックから読み出しおよび書き込みを実行するコンポーネントにより、BenAdda ボード上の Micron 社製 ZBT SRAM デバイスに対する読み出しおよび書き込みが実行されます。外部 RAM ブロックを使用する System Generator モデルがハードウェアに変換されると、RAM ブロックのポートはモデル HDL の最上位の入力および出力ポートに変換されます。これらのポートの適切なピン配置制約が、BenAdda 制約ファイルに含まれます。ZBT SRAM デバイスでは、ハードウェア協調シミュレーションのインプリメンテーションの System Generator 部分と同じクロックが使用されます。

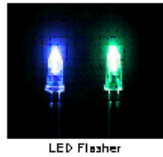
ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

このブロックでは、次のパラメータが設定できます。

- [Output Data Type] : RAM の出力データ型を選択します。[Unsigned] または [Signed] (2 の補数) を指定します。
- [Data Width] : 入力データ幅を指定します。
- [Data Binary Point] : メモリの内容として格納されているデータ値の 2 進小数点の位置を指定します。2 進小数点の位置は、0 ~ 16 (データ幅) です。

XtremeDSP LED Flasher



モデルでハードウェア協調シミュレーションを実行する場合に、System Generator モデルでBenADDA ボード上の 3 色 LED を使用可能にします。モデルで協調シミュレーションを実行すると、LED は赤、緑、黄色と変化します。LED は、27 ビットのフリーランニング カウンタの上位 2 ビットで駆動されます。LED の 3 色の変化を確認するには、モデル シミュレーションの実行中にフリーランニング クロックを選択する必要があります。

ブロック パラメータ

ブロックのパラメータ ダイアログ ボックスは、Simulink モデル内でブロックをダブルクリックすると表示されます。

System Generator ユーティリティ

xlAddTerms	シンクおよびソースを自動的に System Generator モデルに追加します。
xlfa_denominator	FDATool ブロックのフィルタ オブジェクトの分母を返します。
xlfa_numerator	FDATool ブロックのフィルタ オブジェクトの分子を返します。
xlGenerateButton	System Generator のコード ジェネレータを起動します。
xlgetparam および xlsetparam	System Generator ブロックのパラメータを取得し、設定するために使用します。
xlgetparams	System Generator ブロックのすべてのパラメータ値を取得するために使用します。
xlInstallPlugin	System Generator ハードウェア協調シミュレーション プラグイン のインストールに使用します。
xlLoadChipScopeData	ChipScope データの PM ファイルをワークスペースに読み込みます。
xlSBDBuilder	System Generator Board Description Builder (SBDBuilder) を起動します。
xlSetNonMemMap	Gateway ブロックを、メモリ マップされないポートとして使用するよう設定します。
xlSetUseHDL	サブシステムのモデルのブロックの [Use behavioral HDL] オプションをオンにします。
xlSwitchLibrary	ターゲット ディレクトリの HDL ライブラリ参照を指定したライブラリ名に置き換えます。
xlTBUtils	Xilinx Toolbar ブロックで使用可能な、layout、redrawlines、getselected などの便利なプロシージャへのアクセスを提供します。
xlTimingAnalysis	System Generator のタイミング解析ツールを起動し、指定されたタイミング データを表示します。
xlUpdateModel	System Generator のバージョンを管理します。

xlAddTerms

この関数は、モデルで接続されていないポートを終端または駆動するためのブロックを追加すると言う点で、**Simulink** の **addterms** コマンドに似ています。この関数を使用すると、出力ポートは **Simulink** 終端ブロックで終端され、入力ポートは **Simulink** または **System Generator** の **Constant** ブロックで正しく駆動されます。また、**System Generator** の **Gateway** ブロックも条件付きで追加できます。

optionStruct 引数を使用すると、ブロックのプロパティを設定したり (**Constant** ブロックの値を 5 にするなど)、異なるソースまたは終端ブロックが使用されるように設定できます。

構文

```
xlAddTerms (arg1,optionStruct)
```

説明

次の説明で、「ソース ブロック」は接続されていないポートを駆動するブロックを意味し、「終端ブロック」は接続されていないポートを終端するブロックを意味します。

```
xlAddTerms (arg1,optionStruct)
```

xlAddTerms の引数は 1 つまたは 2 つです。2 番目の引数、**optionStruct** はオプションです。最初の引数は、システム名またはブロックのリストです。

arg1	説明
gcs	現在のシステムの文字列ハンドルです。
'top/test1'	test1 というシステムの文字列ハンドル。この場合、 xlAddTerms がシステムに渡され、 test1 の下のすべてのブロック (サブシステムの子ブロックもすべて) で実行されます。
{ 'top/test1' }	ブロックの文字列ハンドル。この場合、 xlAddTerms はブロックに渡され、 test1 ブロックでのみ実行されます。子ブロックは処理されません。
{ 't/b1'; 't/b2'; 't/b3' }	ブロックの文字列ハンドルのリスト。
[1;2;3]	ブロックの数値ハンドルのリスト。

`optionStruct` 引数はオプションですが、使用の場合は MATLAB 構造にする必要があります。次の表に、構造で使用可能な値を示します。構造フィールド名では、MATLAB の場合と同様、大文字と小文字が区別されます。

optionStruct	説明
Source	<p><code>xlAddTerms</code> では、任意のソース ブロックを使用して入力ポートを終端できます (<code>SourceWith</code> フィールド参照)。ソース ブロックのパラメータは、<code>optionStruct</code> の <code>Source</code> フィールドを使用し、パラメータを <code>Source</code> フィールドのサブフィールドとして渡すことにより指定できます。<code>Source</code> フィールドを使用すると、ソース ブロックに対して <code>set_params</code> が複数回実行されます。ソース ブロックのタイプは変更可能であるため、指定されたパラメータが使用されるソース ブロックに対応しているように注意する必要があります。</p> <p>Simulink の <code>Constant</code> ブロックが <code>Source Block</code> として使用されている場合に、ブロックの値を 10 にする例を示します。</p> <pre>Source.value = '10'</pre> <p>System Generator の <code>Constant</code> ブロックがソース ブロックとして使用されている場合に、<code>Constant</code> ブロックの値を 10 に、データ型を <code>UFIX_32_0</code> に設定する方法は次のとおりです。</p> <pre>Source.const = '10'; Source.arith_type='Unsigned'; Source.bin_pt=0; Source.n_bits=32;</pre>
SourceWith	<p>ソース ブロックを指定します。デフォルトでは、<code>Constant</code> ブロックが使用されます。<code>SourceWith</code> にはサブフィールドが 2 つあり、両方とも指定する必要があります。</p> <p>SourceWithBlock : フル パスおよび使用されるブロック名を指定する文字列です (<code>built-in/Constant</code>、<code>xbsIndex_r3/AddSub</code>、など)。</p> <p>SourceWithPort : 接続に使用されるポート番号を指定する文字列です。1 または 3 などのように指定します。1 を指定した場合は、接続にポート 1 が使用されます。</p>
TermWith	<p>終端ブロックを指定します。デフォルトでは、Simulink の <code>Terminator</code> ブロックが使用されます。<code>TermWith</code> にはサブフィールドが 2 つあり、両方とも指定する必要があります。</p> <p>TermWithBlock : フル パスおよび使用されるブロック名を指定する文字列です (<code>built-in/Terminator</code>、<code>xbsIndex_r3/AddSub</code>、など)。</p> <p>TermWithPort : 接続に使用されるポート番号を指定する文字列です。1 または 3 などのように指定します。1 を指定した場合は、接続にポート 1 が使用されます。</p>
UseGatewayIns	<p>必要に応じて、System Generator の <code>Gateway In</code> ブロックが挿入されるようにします。このフィールドを使用すると、<code>Gateway In</code> が挿入されます。<code>Gateway In</code> を使用しない場合は、このフィールドは使用しないでください。</p>

optionStruct	説明
GatewayIn	<p>Gateway In が挿入される場合は、Source と Term の場合のように、このフィールドを使用してパラメータを設定できます。</p> <p>例 :</p> <pre>GatewayIn.arith_type='Unsigned'; GatewayIn.n_bits='32' GatewayIn.bin_pt='0'</pre> <p>Gateway In で ufix_32_0 が出力されるように設定されます。</p>
UseGatewayOuts	<p>必要に応じて、System Generator の Gateway Out が挿入されるようにします。このフィールドを使用すると、Gateway Out が挿入されます。Gateway Out を使用しない場合は、このフィールドは使用しないでください。</p>
GatewayOut	<p>Gateway Out が挿入される場合は、Source と Term の場合のように、このフィールドを使用してパラメータを設定できます。</p> <p>例 :</p> <pre>GatewayOut.arith_type='Unsigned'; GatewayOut.n_bits='32' GatewayOut.bin_pt='0'</pre> <p>Gateway Out に ufix_32_0 が入力されるように設定されます。</p>
RecurseSubSystems	<p>xlAddTerm がすべての子サブシステムで再帰的に実行されるようにします。スカラ値番号 1 または 0 を指定します。</p>

例

例 1 : 現在のシステムでデフォルトパラメータを使用して xlAddTerms を実行します。Constant ソース ブロックが使用され、Gateway は追加されません。サブシステムは再帰的に終端されます。

```
xlAddTerms(gcs);
```

例 2 : サブシステム tt/mySubsystem のすべてのブロックで xlAddTerms を実行します。

```
xlAddTerms(find_system('tt/mySubsystem', 'SearchDepth', 1));
```

例 3 : 現在のシステムで、ソース ブロックの定数値を 1 に設定し、Gateway Out を使用し、終端ブロックとして Simulink の display ブロックが使用されるように xlAddTerms を実行します。

```
s.Source.const = '10';
s.UseGatewayOuts = 1;
s.TermWith.Block = 'built-in/Display';
s.TermWith.Port = '1';
s.RecurseSubSystem = 1;
xlAddTerms(gcs,s);
```

メモ

フィールド名では大文字と小文字が区別されることに注意してください。Source、GatewayIn、および GatewayOut のフィールドを使用する場合、設定されるパラメータ名が有効であることを確認してください。

関連項目

[「Toolbar」](#)、[「xlTBUtils」](#)

xlCache

System Generator キャッシュを管理するために使用します。

構文

```
[core, sg, usertemp] = xlCache ('getpath')
xlCache ('clearall')
xlCache ('clearcorecache')
xlCache ('cleardiskcache')
xlCache ('cleartargetcache')
xlCache ('clearusertemp')
[maxsize] = xlCache ('getdiskcachesize')
[maxentries] = xlCache ('getdiskcacheentries')
```

説明

この関数は、System Generator キャッシュを管理するために使用します。この関数は、次のような形式で記述することもできます。

```
[core, sg, usertemp] = xlCache ('getpath')
```

System Generator コアのキャッシュ、ディスク キャッシュ、および usertemp ディレクトリのローケーションを返します。

```
xlCache ('clearall')
```

System Generator コアのキャッシュ、ディスク キャッシュ、usertemp ロケーションを削除し、ディスクからコンパイル ターゲット プラグイン キャッシュをリロードします。

```
xlCache ('clearcorecache')
```

コアのキャッシュを削除します。コアのキャッシュは、ザイリンクス CORE Generator から生成されたコアを格納することで処理時間を短縮し、再利用が可能な場合はこれらのファイル呼び出します。

```
xlCache ('cleardiskcache')
```

ディスク キャッシュを削除します。ディスク キャッシュは、シミュレーションおよび生成に関連するファイルにタグを付けて保存することで処理時間を短縮し、シミュレーションまたは生成を次に実行するときに、これらのファイルを再生成するのではなく、キャッシュにあるファイル呼び出します。

```
xlCache ('cleartargetcache')
```

コンパイル ターゲット プラグイン キャッシュを作り直します。コンパイル ターゲット プラグイン キャッシュは、新しいコンパイル ターゲット プラグインが追加されるか、既存のターゲットが変更される場合に作成し直す必要があります。

```
xlCache ('clearusertemp')
```

usertemp ディレクトリの内容を削除します。usertemp ディレクトリは、System Generator でシミュレーションまたはネットリスト作成中に使用される一時ファイルの保管場所です。これらの一時ファイルは、デバッグ目的でディスクに保持されるので、削除しても問題ありません。パフォーマンスに影響を与えることもありません。

```
[maxsize] = xlCache ('getdiskcachesize')
```

ディスク キャッシュで使用される最大ディスク容量を返します。デフォルトでは、ディスク キャッシュでファイルを保存するのに 500MB のディスク容量が使用されます。SYSGEN_CACHE_SIZE

環境変数をこのキャッシュの容量に **MB** で設定する必要があります。複数個の大型デザインを設計している場合は、デフォルトより大きな値を設定してください。

`[maxentries] = xlCache ('getdiskcacheentries')`

キャッシュの最大エントリ数を返します。デフォルトでは、**20,000** 個のエントリが保存されます。キャッシュ エントリ データベースのエントリ数を指定するには、**SYSGEN_CACHE_ENTRIES** 環境変数を設定します。エントリ数を小さくしすぎると、キャッシュのパフォーマンスが低下する可能性があります。複数個の大型デザインを設計している場合は、デフォルトより大きな値を設定してください。

関連項目

[System Generator キャッシュの設定](#)

xlConfigureSolver

xlConfigureSolver 関数は、シミュレーション中に最適なパフォーマンスになるように、モデルの Simulink ソルバー設定をします。

構文

```
xlConfigureSolver(<model_handle>);
```

説明

xlConfigureSolver 関数は、<model_handle> に記述されるモデルをコンフィギュレーションします。<model_handle> は、Simulink モデルに対するストリングまたは数値ハンドルにできます。ライブラリ モデルにはコンフィギュレーションするシミュレーション ソルバー パラメータがないので、この関数ではサポートされていません。

System Generator シミュレーション中にパフォーマンスが最適になるように、次の Simulink シミュレーションのコンフィギュレーション パラメータが設定されます。

```
'SolverType' = 'Variable-step'  
'Solver' = 'VariableStepDiscrete'  
'SolverMode' = 'SingleTasking'
```

例

xlConfigureSolver 関数は、次のように設定します。

1. MDL ファイル (sysgen/examples/chipscope/chip.mdl) を開きます。
2. MATLAB コマンド ラインに次を入力します。
gcs
ans = chip
これは、モデルのストリング ハンドルです。
3. ここで、MATLAB コマンド ラインで次を入力します。

```
>> xlConfigureSolver(gcs)  
Set 'SolverType' to 'Variable-step'  
Set 'Solver' to 'VariableStepDiscrete'  
Set 'SolverMode' to 'SingleTasking'  
Set 'SingleTaskRateTransMsg' to 'None'  
Set 'InlineParams' to 'on'
```

xlfda_denominator

ザイリンクス FDATool ブロックに格納されているフィルタ オブジェクトの分母を返します。

構文

```
[den] = xlfda_denominator(fdablk_name);
```

説明

ザイリンクス FDATool ブロック `fdablk_name` に格納されているフィルタ オブジェクトの分母を返します。ブロックが存在しない場合は、エラーが返されます。ブロック名は、ローカル (FDATool など)、相対パス (../FDATool など)、または絶対パス (untitled/foo/bar/FDATool など) で指定します。

関連項目

[「xlfda_numerator」](#)、[「FDATool」](#)

xlfd_a_nu_merator

ザイリンクス FDATool ブロックに格納されているフィルタ オブジェクトの分子を返します。

構文

```
[num] = xlfd_a_nu_merator(fdablk_name);
```

説明

ザイリンクス FDATool ブロック `fdablk_name` に格納されているフィルタ オブジェクトの分子を返します。ブロックが存在しない場合は、エラーが返されます。ブロック名は、ローカル (FDATool など)、相対パス (../FDATool など)、または絶対パス (untitled/foo/bar/FDATool など) で指定します。

関連項目

[「xlfd_a_denominator」](#)、[「FDATool」](#)

xlGenerateButton

System Generator のコード ジェネレータを起動します。

構文

```
status = xlGenerateButton(sysgenblock)
```

説明

System Generator のコード ジェネレータを起動し、ステータス コードを返します。 System Generator ブロックを引数としてこの関数を実行すると、System Generator ブロックのパラメータ ダイアログ ボックスを開き、[Generate] ボタンをクリックするのと同じ操作が実行されます。次に、xlGenerateButton より返されるステータス コードを示します。

ステータス	説明
1	キャンセル
2	シミュレーションの実行中
3	チェック パラメータ エラー
4	コンパイル/生成ネットリスト エラー
5	ネットリスタ エラー
6	ネットリスタ後スクリプト エラー
7	ネットリスト後エラー
8	生成後エラー
9	コンフィギャブル サブシステムとしてインポートしている場合の、外部ビューの不一致

関連項目

[「xlgetparam および xlsetparam」](#)、[「xlgetparams」](#)、[「System Generator」](#)

xlgetparam および xlsetparam

System Generator ブロックのパラメータを取得し、設定するために使用します。どちらの関数も Simulink の `get_param` および `set_param` コマンドに似ており、**Simulink** 関数の代わりに使用する必要があります。

構文

```
value1, value2, ...] = xlgetparam(sysgenblock, param1, param2, ...)  
  
xlsetparam(sysgenblock, param1, value1, param2, value2, ...)
```

説明

System Generator ブロックは、1 つのインスタンスに複数のセットのパラメータが格納されるという点で、ほかのブロックと異なります。パラメータの各セットは、**System Generator** ブロックで使用可能な異なるコンパイル ターゲットに対応します。**System Generator** ブロックに格納された異なるコンパイル ターゲットの切り替えには、`'compilation'` パラメータを使用します。特定のコンパイル タイプに対応するパラメータを取得または設定するには、最初に `xlsetparam` を使用して `'compilation'` パラメータを正しいコンパイル ターゲットに設定してから、その他の値を指定する必要があります。

```
[value1, value2, ...] = xlgetparam(sysgenblock, param1, param2, ...)
```

`xlgetparam` の最初の入力引数は、**System Generator** ブロック へのハンドルである必要があります。その後に続く 引数は、パラメータ名です。返される出力は、入力パラメータと同じ数の配列です。要求されたパラメータが存在しない場合は、`xlgetparam` から 空の値が返されます。`xlgetparams` 関数は、現在のコンパイル ターゲット のすべてのパラメータの取得に使用できます。

```
xlsetparam(sysgenblock, param1, value1, param2, value2, ...)
```

`xlsetparam` 関数でも、**System Generator** ブロックのハンドルが最初の引数として使用されます。その後に続く 引数は、パラメータ名とパラメータ値のペアで指定する必要があります。

コンフィギュレーション パラメータの指定

System Generator トークンの `compilation` パラメータでは、HDL Netlist や NGC Netlist のようなコンパイル タイプが取り込まれます。前述したように、コンパイル タイプが変更されると、**System Generator** トークンにはその特定のコンパイル タイプ用に選択したオプションすべてが記録されます。たとえば、HDL Netlist を選択して対応するターゲット ディレクトリが `hdl_dir` に設定された場合に、NGC Netlist が選択されると、ターゲット ディレクトリは `ngc_dir` のような別のディレクトリをポイントします。コンパイル タイプを変更すると、**System Generator** はそのコンパイル タイプで以前に設定したオプションを使用します。コンパイル タイプを初めて選択した場合、デフォルト値が使用され、**System Generator** トークンの残りのオプションが設定されます。

System Generator ブロックのコンパイル タイプを設定するために `xlsetparam` を使用する場合は、パラメータの設定順が重要になるので、上記に注意し、最初にどのパラメータを設定するよりも前に、ブロックの `compilation` タイプを設定するようにしてください。

`compilation` パラメータを設定するために `xlsetparam` を使用する場合は、必ずそれ以外のパラメータがそのコマンドに対して設定されないようにします。たとえば、次は使用できません。

```
xlsetparam(sysgenblock, 'compilation', 'HDL Netlist', 'synthesis_tool', 'XST')
```

例

例 1: HDL ネットリストに使用される合成ツールの変更

```
xlsetparam(sysgenblock, 'compilation', 'HDL Netlist');  
xlsetparam(sysgenblock, 'synthesis_tool', 'XST')
```

1 番目の xlsetparam では、コンパイル ターゲットが **HDL Netlist** に設定されています。2 番目の xlsetparam では、合成ツールが **XST** に変更されています。

例 2: ファミリーおよびデバイス情報の取得

```
[fam,part]=xlgetparam(sysgenblock,'xilinxfamily','part')  
fam =  
Virtex2  
part =  
xc2v1000
```

関連項目

[「xlGenerateButton」](#)、[「xlgetparams」](#)

xlgetparams

System Generator ブロックの現在のコンパイル タイプに設定されているパラメータをすべて取得します。xlgetparam コマンドおよび xlsetparam コマンドと併用して、System Generator ブロックのパラメータを変更または取得できます。

構文

```
paramstruct = xlgetparams(sysgenblock_handle);
```

sysgenblock_handle を取得するには、MATLAB コマンド ラインに gbc または gcbh を入力します。

```
paramstruct = xlgetparams('chip/ System Generator');
paramstruct = xlgetparams(gcb);
paramstruct = xlgetparams(gcbh);
```

説明

System Generator ブロックで使用可能なパラメータすべてをこのコマンドで取得できます。パラメータの詳細は、System Generator ブロックのマニュアルを参照してください。

```
paramstruct = xlgetparams(sysgenblock);
```

xlgetparams の最初の入力引数は、System Generator ブロックへのハンドルである必要があります。この関数では、パラメータ値のペアをリストする MATLAB 構造が返されます。

例

xlparams 関数は、次のように設定します。

1. MDL ファイル (sysgen/examples/chipscope/chip.mdl) を開きます。

```
sysgen/examples/chipscope/chip.mdl
```

2. System Generator ブロックを選択します。

3. MATLAB コマンド ラインに次を入力します。

```
gcb
```

```
ans = chip/ System Generator
```

これは、System Generator ブロックのストリング ハンドルです。

4. ここで、MATLAB コマンド ラインで次を入力します。

```
gcbh
```

```
ans = 4.3431
```

これは、System Generator ブロックの数値ハンドルです。

5. ここで、MATLAB コマンド ラインで次を入力します

```
xlgetparams(gcb)
```

この関数は、ビットストリームのコンパイル タイプに関連するパラメータをすべて返します。

```
compilation: 'Bitstream'
```

```
compilation_lut: [1x1 struct]
```

```
simulink_period: '1'
```

```
incr_netlist: 'off'
```

```
trim_vbits: 'Everywhere in SubSystem'
```

```
dbl_ovrd: 'According to Block Masks'
```

```
deprecated_control: 'off'
```

```
block_icon_display: 'Default'
```

```
xilinxfamily: 'virtex5'
```

```
part: 'xc5vsx50t'
speed: '-1'
package: 'ff1136'
synthesis_tool: 'XST'
directory: './bitstream'
testbench: 'off'
sysclk_period: '10'
core_generation: 'According to Block Masks'
run_coregen: 'off'
eval_field: '0'
clock_loc: 'AH15'
clock_wrapper: 'Clock Enables'
dcm_input_clock_period: '100'
synthesis_language: 'VHDL'
ce_clr: 0
preserve_hierarchy: 0
postgeneration_fcn: 'xlBitstreamPostGeneration'
settings_fcn: 'xlTopLevelNetlistGUI'
```

compilation_lut パラメータは、この System Generator ブロックに格納されたほかのコンパイルタイプをリストする別の構造です。xlsetparam を使用してコンパイルタイプを設定すると、そのコンパイルタイプに対応するパラメータが xlgetparams または xlgetparam で認識されるようになります。

関連項目

[「xlGenerateButton」](#)、[「xlgetparam および xlsetparam」](#)

xlGetReloadOrder

xlGetReloadOrder は、FIR Compiler ブロック (バージョン 5.0 以降) のリロード順を取得するための関数です。

構文

```
A = xlGetReloadOrder(block_handle, paramStruct, returnType)
```

説明

block_handle

デザインに含まれる FIR Compiler ブロックのハンドルです。FIR Compiler ブロックを選択したら、この関数は次のように起動できます。

```
xlGetReloadOrder(gcbh)
```

この関数の必須パラメータはこれだけです。

paramStruct

抽象化したパラメータの名前と値の組み合わせ。たとえば、[Hardware Oversampling Specification] が [Maximum_Possible] に設定される場合、返されるリロード順は hardwareoversamplingrate が 4. e.g >>options = ... のように明示的に指定されていない限り、不正になる可能性があります。

```
struct('ratespecification','Hardware_Oversampling_Rate','hardwareoversamplingrate',4)
```

```
>> xlGetReloadOrder(gcbh, options)
```

このパラメータはオプションのパラメータで、デフォルト値は struct() です。

returnType

リロード順序の情報形式を指定します。値は address_vector または transform_matrix のいずれかになります。たとえば、A が係数の行ベクタの場合、リロード順に分類された係数は次のように取得できます。

```
reload_order_coefficients = ...
```

```
A(xlGetReloadOrder(gcbh, struct(), 'address_vector'))
```

reload_order_coefficients は、A に含まれる係数がリロード チャネルを介して FIR Compiler に渡される順序を指定しています。

また、transform_matrix 形式は次のように指定できます。

```
reload_order_coefficients = xlGetReloadOrder(gcbh,...  
struct(),'transform_matrix')*A'
```

このパラメータはオプションのパラメータで、デフォルト値は 'transform_matrix' です。

例

xlGetReloadOrder 関数は、次のように使用できます。

1. 次のパスのモデルを開きます。
`<sysgen_path>/examples/demos/sysgenReloadable.mdl`
2. FIR Compiler ブロックを選択します。

3. MATLAB コマンド ラインにと `xlGetReloadOrder(gcbh)` 入力します。

次のような係数のリロード順が表示されます。

```
0  0  1
0  1  0
1  0  0
```

メモ：Return タイプは指定されていないので、デフォルトの `'transform_matrix'` になっています。指定された係数は、1 2 3 2 1 です。フィルタは対称フィルタとして推論されるので、5 個の係数のうちの 3 個のみを読み込む必要があります。読み込む順序は、3 つ目のエレメント、2 つ目のエレメント、最初のエレメント (3 2 1) の順にする必要があります。

4. 同じ FIR Compiler 設定で次のように Return タイプを `'transform_matrix'` から `'address_vector'` に変更します。

```
xlGetReloadOrder(gcbh, struct(), 'address_vector'),
```

同じ係数のリロード順が表示されますが、形式が異なります。

```
ans =
```

```
3
2
1
```

5. 次にフィルタの係数構造を変更してみます。FIR Compiler ブロックをダブルクリックし、[Implementation] タブの [Coefficient Structure] で [Non_Symmetric] を選択し、[OK] をクリックします。

6. FIR Compiler b が選択されていることを確認し、前の手順と同じコマンドを入力します。読み込み順序と読み込まれた係数の数の違いに注目してください。

```
ans =
```

```
5
4
3
2
1
```

メモ：指定された係数は、1 2 3 2 1 です。フィルタは明示的に `non_symmetric` フィルタに設定されたので、5 個すべての係数が上記のようなリロード順 (5 つ目 (1)、4 つ目 (2)、3 つ目 (3)、2 つ目 (2)、1 つ目 (1)) で読み込まれます。

関連項目

[FIR Compiler 5.0 ブロック](#)

xlInstallPlugin

指定された System Generator のハードウェア協調シミュレーション プラグインをインストールします。インストールが完了すると、新しいコンパイル ターゲットを System Generator ブロックのパラメータ ダイアログ ボックスで選択できるようになります。

構文

```
xlInstallPlugin('<plugin_name>')
```

説明

インストールするプラグイン ファイルの名前を指定する、**plugin** というパラメータのみが使用されます。このパラメータには、パス情報を含めることができます。**.zip** の拡張子はオプションです。

例

例 1：

```
xlInstallPlugin('plugin.zip')
```

例 2：

```
xlInstallPlugin('plugin')
```

関連項目

[「Linux OS への System Generator のインストール」](#)、[「新規プラットフォームのサポート」](#)
[「xISBDBuilder」](#)

xlLoadChipScopeData

ChipScope Pro の PRN ファイルを読み込み、ワークスペース変数を作成し、条件付きで結果をプロットします。

構文

```
status = xlLoadChipScopeData(filename, plotResults);
```

説明

filename に指定された PRN ファイルを読み込み、plotResults == 1 に指定されている場合は結果をプロットします。filename に指定されたファイルが見つからない場合は、-1 のステータスが返されます。正常に終了した場合は 0 のステータスが返されます。

メモ：符号付きおよび符号なしの 10 進数のみがサポートされます。

例

例 1：

```
xlLoadChipScopeData('SineWave.prn',0);
```

関連項目

[「ChipScope」](#)

xlSBDBuilder

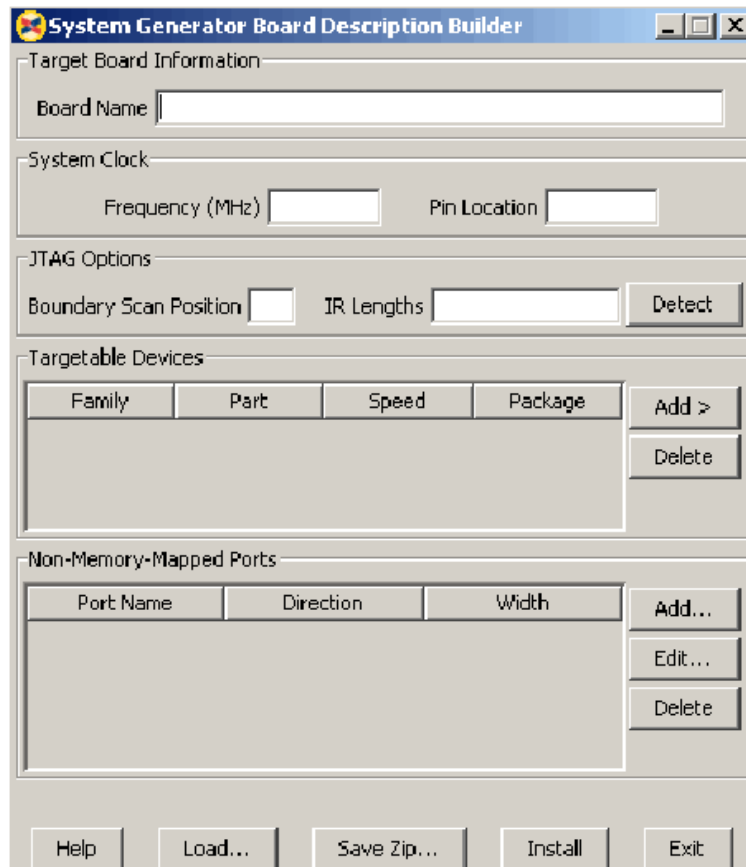
System Generator Board Description (SBD) Builder アプリケーションは、協調シミュレーションプラットフォームに関連した情報を入力可能なグラフィカル ユーザー インターフェイスで、新規の JTAG ハードウェア協調シミュレーションプラグインの設計に役立ちます。

構文

```
xlSBDBuilder;
```

説明

SBDBuilder を起動すると、次のようなダイアログ ボックスが開きます。



The dialog box is titled "System Generator Board Description Builder". It contains several sections for configuring board information:

- Target Board Information:** A text field for "Board Name".
- System Clock:** Two text fields for "Frequency (MHz)" and "Pin Location".
- JTAG Options:** Two text fields for "Boundary Scan Position" and "IR Lengths", followed by a "Detect" button.
- Targetable Devices:** A table with columns "Family", "Part", "Speed", and "Package". To the right are "Add >" and "Delete" buttons.
- Non-Memory-Mapped Ports:** A table with columns "Port Name", "Direction", and "Width". To the right are "Add...", "Edit...", and "Delete" buttons.

At the bottom, there are five buttons: "Help", "Load...", "Save Zip...", "Install", and "Exit".

このダイアログ ボックスで次に説明するようにパラメータを指定して、ボード サポート パッケージを作成します。

[Board Name]：ボード名を指定します。この名前が、System Generator トークンの [Compilation] で JTAG ハードウェア協調シミュレーションプラットフォームとしてリストされます。

[System Clock]：JTAG ハードウェア協調シミュレーションでは、System Generator デザインを駆動するオンボード クロックが必要です。ボードのシステム クロックについて、次の情報を指定します。

- [Frequency (MHz)]：オンボードのシステム クロックの周波数を MHz で指定します。
- [Pin Location]：システム クロックを接続する FPGA 入力ピンを指定します。

[JTAG Options] : System Generator でハードウェア協調シミュレーション用に FPGA をプログラムするには、FPGA ボードの JTAG チェーンに関する情報が必要です。これらの情報を見つける方法は、「プラットフォームの情報の取得」を参照してください。ボードの仕様がはっきりわからない場合は、製造業者のマニュアルを参照してください。[JTAG Options] では、次の情報を入力します。

- [Boundary Scan Position] : JTAG チェーンでのターゲット FPGA の位置を指定します。チェーンのデバイスには 1 から番号が付けられています (最初のデバイスは 1、2 番目のデバイスは 2 など)。
- [IR Lengths] : JTAG チェーン上のすべてのデバイスに対して命令レジスタの幅を指定します。値の間は、スペース、カンマ、またはセミコロンで区切ります。
- [Detect] : このボタンをクリックすると、FPGA ボードにアクセスして自動的に IR 幅が検出されます。この機能を使用するには、ボードの電源をオンにし、パラレル ケーブル IV に接続しておく必要があります。JTAG チェーン上の不明のデバイスの IR 幅はクエスチョン マーク (?) で示されるので、手動で指定する必要があります。

[Targetable Devices] : この表には、ボード上の FPGA でプログラム可能なものがリストされます。JTAG チェーンのすべてのデバイスがリストされるわけではなく、指定したバウンダリ スキャン位置に存在する可能なデバイスがリストされます。ほとんどのボードでは、指定する必要があるデバイスは 1 のみですが、ボードによっては複数のデバイスから選択可能です (同じソケットにある XCV1000 または XCV2000 など)。次のように [Add] および [Delete] ボタンを使用して、デバイスリストを作成します。

- [Add] : ボード上のデバイスを選択するメニューが表示されます。次の図に示すように、デバイスはファミリ、パーツ名、スピード、パッケージタイプで分類されています。
- [Delete] : リストから選択したデバイスを削除します。

spartan2	▶			
spartan2e	▶	xc2s50e	▶	
spartan3	▶	xc2s100e	▶	
virtex	▶	xc2s150e	▶	-7 ▶
virtexe	▶	xc2s200e	▶	-6 ▶ ft256
virtex2	▶	xc2s300e	▶	fg456
virtex2p	▶	xc2s400e	▶	pq208
virtex4	▶	xc2s600e	▶	

[Non-Memory-Mapped Ports] : ボード専用ポートのサポートを追加できます。ボード専用ポートは、ハードウェア協調シミュレーション中に FPGA でアクセスする必要のあるオンボード コンポーネント (外部メモリ、DAC、ADC など) がある場合に便利です。ボード専用ポートは、ハードウェア協調シミュレーション用にデザインをコンパイルしたときに、Simulink ポートが作成されるのではなく物理的な場所にマップされるので、[Non-Memory-Mapped Ports] (メモリ マップされないポート) と示されています。詳細は、「メモリ マップされないポートの指定」を参照してください。[Add]、[Edit]、および [Delete] ボタンを使用して、ボード専用ポートを設定します。

- [Add] : ポートの情報を入力するダイアログ ボックスが表示されます。
- [Edit] : 選択したポートに変更を加えます。
- [Delete] : リストから選択したポートを削除します。

[Help] : このマニュアルを表示します。

[Load]：読み込む SBDBuilder Saved Description (保存された記述) XML ファイルを選択します。このファイルは、プラグインを作成するたびに保存されます。以前のプラグイン ファイルを読み込んで、編集する場合に便利です。

[Save Zip]：System Generator のプラグイン ファイルすべてを含む ZIP ファイルを作成します。開いたダイアログ ボックスで、ファイル名とパス名を指定します。作成される ZIP ファイルは、System Generator の [xlInstallPlugin](#) 関数に渡すのに適切なフォーマットです。

[Exit]：アプリケーションを終了します。

関連項目

「[Linux OS への System Generator のインストール](#)」、「[新規プラットフォームのサポート](#)」、
「[xlInstallPlugin](#)」

xlSetNonMemMap

ハードウェア協調シミュレーションの実行時に、Gateway In または Gateway Out ブロックがメモリマップされないポートとして使用されるように設定します。このオプションは、ハードウェア協調シミュレーション メモリ マップではなく、FPGA 外部のハードウェアに Gateway が配線される場合に頻繁に使用されます。

構文

```
xlSetNonMemMap(block, company, project)
```

説明

xlSetNonMemMap の呼び出しには、最低 3 個のパラメータが必要です。1 番目は、メモリ マップされないよう設定する Gateway の名前またはハンドルです。メモリ マップされないよう設定するかどうかは、社名およびプロジェクト名に基づきます。2 番目および 3 番目のパラメータは、社名およびプロジェクト名を表す文字列です。

例

例 1:

```
xlSetNonMemMap(gcbh, 'Xilinx', 'jtaghwcosim');
```

例の 1 番目のパラメータでは、現在選択されているブロックのハンドルが返されます。この Gateway は、サイリンクスの JTAG ハードウェア協調シミュレーション用に生成される場合は、メモリ マップされないポートとして使用されます。

例 2:

```
xlSetNonMemMap(gcbh, 'Nallatech', 'xdspkit');
```

例の 1 番目のパラメータでは、現在選択されているブロックのハンドルが返されます。この Gateway は、Nallatech 社製 XtremeDSP 開発キット用に生成される場合は、メモリ マップされないポートとして使用されます。

関連項目

[「ハードウェア協調シミュレーションの使用」](#)、[「新規プラットフォームのサポート」](#)

xlSetUseHDL

サブシステムのモデルのブロックの [Use behavioral HDL] オプションをオンにします。

構文

```
xlSetUseHDL(system, mode)
```

説明

system パラメータで指定されたモデルまたはシステムで、コアまたはビヘイビア HDL (モードによりいずれかを決定) の使用が設定されます。モードは数値で、0 の場合はコア、1 の場合はビヘイビア HDL が使用されます。

例

例 1：

```
xlSetUseHDL(gcs, 0)
```

現在選択されたシステムでコアを使用するように設定します。

関連項目

[「xlSetNonMemMap」](#)、[「xlSBDBuilder」](#)

xlSwitchLibrary

ターゲット ディレクトリ の HDL ライブラリ 参照を指定したライブラリ 名に置き換えます。

構文

```
xlSwitchLibrary(<target_directory>, <from_library_name>,  
<to_library_name>)
```

説明

<target_directory> ディレクトリにある System Generator デザインの <from_library_name> への HDL ライブラリ参照を <to_library_name> に置き換えます。

例

例 1:

次のコマンドは、System Generator で作成された .\netlist というターゲット ディレクトリで xlSwitchLibrary を実行し、デフォルト ライブラリを work から design1 に変更します。

```
>> xlSwitchLibrary('.\netlist_w_dcm', 'work', 'design1')  
INFO: Switching HDL library references in design 'basicmult_dcm_mcw'  
...  
INFO: A backup of the original files can be found at  
'D:\Matlab\work\Basic\netlist_w_dcm\switch_lib_backup.TlOy'.  
INFO: Processing file 'basicmult.vhd' ...  
INFO: Processing file 'basicmult_mcw.vhd' ...  
INFO: Processing file 'basicmult_dcm_mcw.vhd' ...  
INFO: Processing file 'xst_basicmult.prj' ...  
INFO: Processing file 'vcom.do' ...  
INFO: Processing file 'vsim.do' ...  
INFO: Processing file 'pn_behavioral.do' ...  
INFO: Processing file 'pn_posttranslate.do' ...  
INFO: Processing file 'pn_postmap.do' ...  
INFO: Processing file 'pn_postpar.do' ...  
INFO: Processing file 'basicmult_dcm_mcw.ise' ...
```

xlTBUtills

ザイリンクス **Toolbar** ブロックの一部へのアクセスを提供します。layout、rerouting 関数および選択されたブロックおよび行を返す関数にアクセスできるようになります。

構文

```
xlTBUtills(function, args)
```

例：

```
xlTBUtills('ToolBar')
xlTBUtills('Layout',struct('verbose',1,'autoroute',0))
xlTBUtills('Layout',optionStruct)
xlTBUtills('Redrawlines',struct('autoroute',0))
xlTBUtills('RedrawLines',optionStruct)
[lines,blks]=xlTBUtills('GetSelected','All')
```

説明

xlTBUtills(function [,args])

xlTBUtills は、ザイリンクス **Toolbar** ブロックで使用される関数の集合です。引数には、実行する関数名を指定します。引数として指定した関数に引数が必要な場合は、その関数に引数を指定します。引数では大文字と小文字は区別されません。使用できる値を次に示します。詳細な説明は、表の後に記述します。

関数	説明
'ToolBar'	ザイリンクスの Toolbar GUI を起動します。既に起動している場合は、前面に表示されます。
'Layout'	モデルでレイアウト アルゴリズムを実行し、配置および再配線を実行します。次に説明するオプション構造を使用して、カスタマイズできます。
'RedrawLines'	モデルで配線アルゴリズムを実行し、再配線を実行します。次に説明するオプション構造を使用して、カスタマイズできます。
'GetSelected'	フォーカスされたシステムで選択されているブロックおよびラインへの MATLAB Simulink ハンドルを返します。

xlTBUtills('Layout',optionStruct)

Simulink モデルを自動的に配置配線します。optionStruct は MATLAB 構造のデータ型で、Layout のパラメータを含みます。optionStruct 引数はオプションです。

Layout では、回路は左から右に配置されると想定されています。配置の後、Simulink を使用してワイヤ接続が自動的に配線されます。Simulink では、ブロック ラベルなど、画面に表示されるものはすべて避けるように配線されます。[ignore_labels] をオンにすると、ラベルを通過して配線されます。ラベルは配線後に手動で見やすい場所に移動できます。フィールド名では大文字と小文字が区別されることに注意してください。

フィールド名	説明 [デフォルト値]
x_pitch、 y_pitch	ブロックの間隔 (ピッチ) のピクセル数。x_pitch ではブロックの左右の間隔、y_pitch ではブロックの上下の間隔を指定します。[30]
x_start、 y_start	左余白 (x_start) および上余白 (y_start) をピクセル数で指定します。モデルの左および上のスペースです。[10]
autoroute	Simulink のラインの自動配線をオンにします。値は 1 または 0 です。[1]
ignore_labels	ラインを自動配線する場合、Simulink ではテキスト ラベルを避けて配線します。このフィールドを 1 に設定すると、配線実行中はテキスト ラベルのサイズが最小化されます。
sys	レイアウトするシステムの名前。[gcs]
verbose	1 に設定すると、レイアウト処理の実行中に待機バーが表示されます。

xITBUtils('RedrawLines',optionStruct)

ラインが選択されていない場合は、Simulink モデル内のラインをすべて再描画します。ラインを選択した場合は、選択されたラインのみが再描画されます。ブランチを選択した場合は、ライン全体 (メインのトランクおよびその他のサブ ブランチすべて) が再描画されます。

フィールド名	説明 [デフォルト値]
autoroute	Simulink のラインの自動配線をオンにします。値は 1 または 0 です。[1]
sys	レイアウトするシステムの名前。[gcs]

[lines,blks]=xITBUtils('GetSelected',arg)

選択されたブロックおよびラインへのハンドルを返します。引数 arg はオプションです。値は、次の表で説明されている文字列のいずれかである必要があります。

フィールド名	説明 [デフォルト値]
'all'	選択されたラインとブロックの両方を取得します (デフォルト)。
'lines'	選択されたラインのみを取得します。
'blocks'	選択されたブロックのみを取得します。

2 つの項目 (ライン情報 (lines) およびブロック ハンドルの配列 (blks) を含む構造の配列) を持つ配列を返します。'lines' 引数を使用される場合は、blks は空の配列となり、'blocks' 引数を使用される場合は、lines が空の配列となります。

例

例 1a : レイアウトの実行

```
a.verbose = 1;
a.autoroute = 0;
xITBUtils('Layout',a);
```

レイアウト ツールが、verbose をオン、autoroute をオフにして実行されます。

例 1b : レイアウトの実行

```
xITBUtils('Layout',struct('verbose',1,'autoroute',0));
```

レイアウト ツールが、**verbose** をオン、**autoroute** をオフにして実行されます。

例 2 : ラインの再描画

```
xlTBUtils('Redrawlines',struct('autoroute',0));
```

現在のシステムのラインを、**autoroute** をオフにして再描画します。

例 3 : 選択されたラインおよびブロックの取得

```
[lines,blks]=xlTBUtils('GetSelected')  
lines =
```

```
1x3 struct array with fields:
```

```
Handle  
Name  
Parent  
SrcBlock  
SrcPort  
DstBlock  
DstPort  
Points  
Branch
```

```
blks =
```

```
1.0e+003 *  
  
3.0320  
3.0480
```

現在のシステムで選択されたすべてのラインとブロックが返されます。この例では、ライン 3 つとブロック 2 つが選択されています。最初のライン ハンドルには、次のコマンドを使用してアクセスできます。

```
lines(1).Handle
```

```
ans =
```

```
3.0740e+003
```

最初ブロックへのハンドルには、次のコマンドを使用してアクセスできます。

```
blks(1)  
ans =  
3.0320e+003
```

メモ

Layout および **RedrawLines** の実行内容は元に戻すことができません。元に戻す必要がある場合に備えて、実行前にモデルのコピーを保存してください。

* このユーティリティを使用する場合、AT&T 社の規約に従う必要があります。次の内容を参照してください。

This product contains certain software code or other information ("AT&T Software") proprietary to AT&T Corp. ("AT&T"). The AT&T Software is provided to you "AS IS". YOU ASSUME TOTAL RESPONSIBILITY AND RISK FOR USE OF THE AT&T SOFTWARE. AT&T DOES NOT MAKE, AND EXPRESSLY DISCLAIMS, ANY EXPRESS OR IMPLIED WARRANTIES OF ANY KIND WHATSOEVER, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, WARRANTIES OF TITLE OR NON-INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHTS, ANY WARRANTIES ARISING BY USAGE OF TRADE, COURSE OF DEALING OR COURSE OF PERFORMANCE, OR ANY WARRANTY THAT THE AT&T SOFTWARE IS "ERROR FREE" OR WILL MEET YOUR REQUIREMENTS.

Unless you accept a license to use the AT&T Software, you shall not reverse compile, disassemble or otherwise reverse engineer this product to ascertain the source code for any AT&T Software.

© AT&T Corp. All rights reserved. AT&T is a registered trademark of AT&T Corp.

関連項目

[「Toolbar」](#)、[「xlAddTerms」](#)

xlTimingAnalysis

System Generator のタイミングおよび電力解析ツールの GUI は通常、MATLAB の System Generator ブロックのパラメータ ダイアログ ボックスで [Timing Analysis] コンパイル ターゲットを使用すると起動しますが、このコマンドを使用しても起動できます。[Timing Analysis] コンパイル ターゲットを設定すると、ツールでデザインがコンパイルされ、配置配線が実行され、その他の処理が実行されてからタイミングおよび電力解析ツールの GUI が表示されます。このコマンドを使用すると、コンパイル ターゲットを新たに処理せずに、前回生成されたタイミング データを GUI で開くことができます。

構文

```
xlTimingAnalysis(target_directory);
```

説明

タイミング データが保存されているディレクトリを指定してこのコマンドを呼び出すと、System Generate タイミング解析ツールの GUI が起動します。

タイミング解析ツールの GUI には、指定されたターゲット ディレクトリに保存されている timing.twx および name_translations データ ファイルの内容が表示されます。

ターゲット ディレクトリ名は、相対パスでも絶対パスでもかまいません。

例

```
>> xlTimingAnalysis('timing')
```

ここで 'timing' は前回のタイミング解析を実行したターゲット ディレクトリ名です。

xlUpdateModel

System Generator v7.1 以前で作成されたモデルを v9.1.01 以降で使用するには、モデルをアップデートする必要があります。モデルをアップデートするには、変換スクリプトを起動する MATLAB コマンド `xlUpdateModel` を実行します。

変換スクリプトでは、モデルの変換前のバージョンおよび変換後の新しいバージョンは自動的に保存されません。変換スクリプトを実行する前にモデルのバックアップ コピーを作成するか、アップデートしたモデルを別の名前で保存してください。

モデルによっては、`xlUpdateModel` を実行した後に手動で変更を加える必要があるものもあります。この関数を実行すると、手動の変更が必要な箇所が示されます。

構文

```
xlUpdateModel('my_model_name');  
xlUpdateModel('my_model_name', 'lib');  
xlUpdateModel('my_model_name', 'assert');
```

説明

V2.x 以前のモデルのアップデート

v3.1 より前のバージョンのモデルをアップデートする場合は、System Generator v7.x を入手してモデルを v7.x にアップデートしてから v9.1.01 にアップデートする必要があります。

v3.x、v6.x、および v7.x モデルのアップデート

このセクションでは、System Generator v3.x、v6.x、および v7.x モデルを v9.1.01 で機能するようにアップデートする手順を示します。

メモ：このセクションの v7.x に関する手順は、v3.x または v6.x にも適用できます。

v7.x モデルを v9.1.01 にアップデートする基本的な手順は、次のとおりです。

- 1) v7.1 モデルおよびモデルで使用されるユーザー定義ライブラリのバックアップ コピーを作成します。
- 2) `xlUpdateModel` をまずライブラリに対して実行し、その後モデルに対して実行します。
- 3) `xlUpdateModel` のレポートを参照し、指示に従います。
- 4) モデルが v9.1.01 で動作するかどうかを確認します。

これらの手順を、次に詳しく説明します。

1. v7.1 モデルおよびモデルで使用されるユーザー定義ライブラリのバックアップ コピーを作成します。
2. `xlUpdateModel` を実行します。

MATLAB の [Command Window] で `cd` コマンドを使用し、モデルを含むディレクトリに移動します。モデル名が `designName.mdl` の場合は、「`xlUpdateModel('designName')`」と入力します。

`xlUpdateModel` は、次のタスクを実行します。

- ◆ v7.x デザインの各ブロックを、同じ設定を使用した対応する v9.1.01 ブロックにアップデートします。

- ◆ 加えた変更を説明するレポートを作成します。このレポートに、ユーザーが手動で加える必要のある変更も記述されます。

ほとんどの場合、xlUpdateModel で等価の v9.1.01 モデルが生成されますが、変更が必要な構文が含まれている可能性があります。レポートを参照し、このセクションの残りの手順に従うことが重要です。

3. xlUpdateModel レポートを参照し、その指示に従います。

レポートに次の問題が記述されている場合は、手動の変更が必要です。

- a. System Generator v7.x モデルに削除されたブロックが含まれている。

次のブロックは、System Generator から削除されています。

CIC、Clear Quantization Error、Digital Up Converter、J.83 Modulator、Quantization Error、Sync

- b. System Generator v7.x モデルに廃止予定のブロックが含まれている。

DDSV4.0 ブロックはまだ System Generator に含まれていますが、廃止予定です。

- c. System Generator v7.x モデルでサンプリング周期を明示的に指定するフィールドが使用されている。

サンプリング周期を明示的に指定するフィールドは、System Generator v9.1.01 のソース以外のほとんどのブロックで削除されています。Counter ブロックなどのソース ブロックでは、サンプリング周期の明示的な指定が可能です。フィードバック ループを含むモデルをアップデートする場合は、System Generator でパスの適切なレートとタイプを判断できるようにするため、通常 xlUpdateModel を実行した後に Assert ブロックを追加する必要があります。次のメッセージは、Assert ブロックが必要であることを示しています。

The data rates could not be established for the feedback paths through this block. You may need to add Assert blocks to instruct the system

この場合、各フィードバック ループに Assert ブロックを追加し、このブロックでレートとタイプを指定します。

変換スクリプトでは、v7.1 モデルの周期が明示的に指定された部分すべてに対して、モデルが変換されたことが示されます。変換後のモデルでは、ほとんどの場合 Assert ブロックを追加する必要はありません。Assert ブロックが必要かどうかを判断するには、ダイアグラムをアップデートします ([編集] → [モデルの更新] をクリック)。レートが決定しない場合は、1 つ以上の Assert ブロックを挿入する必要があります。

明示的なサンプリング周期の設定を使用するブロックの後に自動的に Assert ブロックを追加するように、変換スクリプトを設定できます。このオプションを使用するには、次のコマンドを実行します。

```
xlUpdateModel (designName, 'assert')
```

4. アップデートされたモデルを保存し、閉じます。

アップデート前のモデルのバックアップ コピーを作成していない場合は、アップデート後のモデルを別の名前で作成します。

5. モデルが System Generator v9.1.01 で動作することを確認します。

上記の手順に従っていれば、モデルは System Generator v9.1.01 で動作するはずです。モデルを System Generator v9.1.01 で開き、実行してください。

例

例 1 :

```
>> xlUpdateModel('my_model_name');
```

現在の作業ディレクトリにある my_model_name.mdl ファイルをアップデートします。

例 2 :

```
>> xlUpdateModel('my_model_name','lib');
```

現在の作業ディレクトリにある my_model_name.mdl ファイルと関連するライブラリをアップデートします。

例 3 :

```
>> xlUpdateModel('my_model_name','assert');
```

現在の作業ディレクトリにある my_model_name.mdl ファイルをアップデートします。必要に応じて Assert ブロックを追加します。

xlVersion

複数のバージョンの **System Generator** をインストールできます。**MATLAB** コマンド **xlVersion** を使用するとインストールされているバージョンが表示され、バージョンを切り替えることができます。バージョンの切り替えには、**MATLAB** の再起動が必要な場合があります。その場合は、コマンドによりその指示が表示されます。

System Generator 8.2 をインストールした後に **8.1** をインストールした場合、**xlVersion** を機能させるには **8.2** を再インストールする必要があります。

構文

```
xlVersion;  
xlVersion ver;  
xlVersion -add directory;
```

説明

パラメータを指定せずにこのコマンドを呼び出すと、インストールされた現在のバージョンの **System Generator** および使用可能なバージョンすべてが表示されます。

ver オプションで、切り替え先の **System Generator** のバージョンを指定します。

-add オプションを使用すると、ディレクトリを指定できます。**System Generator** がインストールされているディレクトリを指定します。指定されたディレクトリにインストールされた **System Generator** が、使用する **System Generator** として起動します。

関連項目

[「ハードウェア協調シミュレーションを使用したリアルタイム信号処理」](#)

プログラムを使用したアクセス

プログラム生成用 System Generator API

はじめに

プログラム生成用の System Generator のスクリプト (PG API スクリプト) は、MATLAB の M 関数ファイルで、xBlock、xSignal、xInport、および xOutport オブジェクトをインスタンス化およびインターコネクトすることで、System Generator のサブシステムを構築します。これは、System Generator ダイアグラム (サブシステム) を構築する方法です。この章の例で示すように、System Generator のプログラム スクリプトの最上位関数はエントリ ポイントで、xBlock コンストラクタから起動する必要があります。コンストラクタが終了すると、MATLAB で対応する System Generator のサブシステムが対応するモデルに追加されます。モデルが開いていない場合は、「untitled」という名前の新規モデルが作成され、このモデルに System Generator サブシステムが挿入されます。

xBlock コンストラクタでは、xBlock オブジェクトが作成されます。このオブジェクトは、ライブラリ ブロックから作成するか、またはサブシステムにすることができます。xSignal オブジェクトは、ターゲットにソース ブロックを接続するワイヤに対応します。xInport オブジェクトでは、Simulink の Inport オブジェクトがインスタンス化され、xOutport オブジェクトでは Simulink の Outport オブジェクトがインスタンス化されます。

また、API には、Simulink ダイアグラムをプログラム生成スクリプトに変換するヘルプ関数 `xlsub2script` があります。

API は、learning モード、production モード、および debugging モードの 3 つのモードで動作します。learning モードでは、物理的なスクリプト ファイルなしでコマンドを入力できます。このモードは、API を習得するのに役立ちます。このモードでは、すべてのブロック、ポート、およびサブシステムが「untitled」という名前の Simulink モデルに追加されます。引数を使用せずに xBlock を実行すること、および別の learning セッションを開始する前に「untitled」モデルを終了することを忘れないでください。production モードには M 関数ファイルがあり、xBlock コンストラクタから起動されて、サブシステムが生成されます。サブシステムは、既存のモデルまたは新しいモデルに含めることができます。debugging モードは、production モードの動作とほぼ同一ですが、新しいオブジェクトが作成されたり、または新しい接続が確立されるたびに Simulink ダイアグラムが再配線される点で異なります。このモードは、ブレーク ポイントを設定したり、シングル ステップするスクリプトをデバッグするときに便利です。

xBlock

xBlock コンストラクタでは、xBlock オブジェクトが作成されます。このオブジェクトは、ライブラリ ブロックから作成するか、またはサブシステムにすることができます。xBlock コンストラクタは、次の 3 つの方法で使用できます。

- 現在のサブシステムにリーフ ブロックを追加する
- 現在のサブシステムにサブシステムを追加する
- モデルに最上位のサブシステムを接続する

xBlock では、4 つの引数を使用され、次のように実行します。

```
block = xBlock(source, params, inports, outports);
```

source 引数は文字列で、ライブラリ ブロック名が指定されることが想定されます。ソース ブロックが xbsIndex_r4 ライブラリまたは Simulink ビルトイン ライブラリに含まれる場合は、ライブラリ名なしでブロック名を使用できます。たとえば xBlock('AddSub', ...) の呼び出しは、xBlock('xbsIndex_r4/AddSub', ...) と同様に処理されます。これらのライブラリに含まれないソース ブロックでは、xBlock('xbsTest_r4/Assert Relation', ...) のように、完全パスを使用する必要があります。source 引数が関数ハンドルの場合、PG API 関数として解釈されます。この関数ハンドルが MATLAB 構造体の場合、最上位をモデルに接続する方法を指定するコンフィギュレーション構造体として扱われます。

params 引数では、パラメータが設定されます。配列に基づいて結合するセル配列、名前に基づいて結合する MATLAB 構造体のいずれかを指定できます。source のパラメータがライブラリ内のブロックの場合は、この引数をセル配列にする必要があります。関数ポインタの場合は、この引数をセル配列にする必要があります。

inports および outports 引数では、サブシステムの入力および出力ポートの結合方法を指定します。配列に基づいて結合する場合はセル配列、名前に基づいて結合する場合は MATLAB 構造体を指定できます。入力および出力の結合を指定する場合、セル配列の要素に xSignal、xInport、または xOutport オブジェクトのいずれかを指定できます。ポート結合の引数が MATLAB 構造体の場合は、構造体のフィールドにブロックのポート名、構造体の値にポートが結合されるオブジェクトを指定します。

2 個のポートを結合する引数は、オプションです。xBlock オブジェクトの構築時に引数がない場合は、xBlock オブジェクトの bindPort を使用してポートの結合を指定できます。bindPort を実行するには、コマンド ラインに次のように入力します。

```
block.bindPort(inports, outports)
```

inports および outports 引数では、入力ポートと出力ポートの結合を指定します。この場合、xBlock で source および params の 2 個の引数のみを使用してオブジェクト ブロックが作成されます。

次に、その他の xBlock メソッドを示します。

- names = block.getOutportNames では、出力ポート名のセル配列を返します。
- names = block.getInportNames では、入力ポート名のセル配列を返します。
- nin = block.getNumInports では、入力ポート数を返します。
- nout = block.getNumoutports では、出力ポート数を返します。
- insigs = block.getInSignals では、入力信号のセル配列を返します。
- outsigs = block.getOutSignals では、出力信号のセル配列を返します。

xInport

xInport オブジェクトは、サブシステムの入力ポートを表します。

次に、各種コンストラクタを示します。

```
port = xInport(port_name)
```

port_name という名前の xInport オブジェクトを作成します。

```
[port1, port2, port3, ...] = xInport(name1, name2, name2, ...)
```

入力ポートとその名前のリストを作成します。

```
port = xInport
```

自動生成された名前の入力ポートを作成します。

xInport オブジェクトは、ポート結合に渡すことができます。

メソッド

```
outsigs = port.getOutSignals
```

出力信号のセル配列を返します。

xOutputport

xOutputport オブジェクトは、サブシステムの出力ポートを表します。

次に、各種コンストラクタを示します。

```
port = xOutputport(port_name)
```

port_name という名前の xOutputport オブジェクトを作成します。

```
[port1, port2, port3, ...] = xOutputport(name1, name2, name2, ...)
```

出力ポートとその名前のリストを作成します。

```
port = xOutputport
```

自動生成された名前の出力ポートを作成します。

xOutputport オブジェクトは、ポート結合に渡すことができます。

メソッド

```
port.bind(obj)
```

オブジェクトをポートに接続します。port は xOutputport オブジェクトで、obj は xSignal または xInport のいずれかです。

```
insigs = port.getInSignals
```

入力信号のセル配列を返します。

xSignal

`xSignal` は、ソースをターゲットに接続する信号オブジェクトを表します。

次に、各種コンストラクタを示します。

```
sig = xSignal(sig_name)
```

`sig_name` という名前の `xSignal` オブジェクトを作成します。

```
[sig1, sig2, sig3, ...] = xSignal(name1, name2, name2, ...)
```

信号とその名前のリストを作成します。

```
sig = xSignal
```

自動生成された名前の `xSignal` を作成します。

`xSignal` オブジェクトは、ポート結合に渡すことができます。

メソッド

```
sig.bind(obj)
```

オブジェクトを信号に接続します。`sig` は `xSignal` オブジェクトで `obj` は `xSignal` または `xInport` オブジェクトです。

```
src = sig.getSrc
```

`xSignal` オブジェクトを駆動するソース オブジェクトのセル配列を返します。セル配列には、要素を 1 個まで持たせることができます。ソースが入力ポートの場合、ソース オブジェクトはオブジェクトになります。ソースがブロックの出力ポートの場合、ソース オブジェクトはフィールドとポートから構成される構造体になります。ブロック フィールドは `xBlock` オブジェクトで、ポート フィールドはポート インデックスです。

```
dst = sig.getDst
```

`xSignal` オブジェクトが駆動するデスティネーション オブジェクトのセル配列を返します。各要素には、構造体または `xOutput` オブジェクトのいずれかを指定できます。`getSrc` メソッドの返し値と同様に定義されます。

xlsub2script

`xlsub2script` は、サブシステムを **System Generator** スクリプトの最上位に変換するヘルパー関数です。

`xlsub2script(subsystem)` は、サブシステムを最上位スクリプトに変換します。引数には、モデルも指定できます。

デフォルトでは、生成される **M** 関数ファイルの名前は、サブシステム名に含まれるスペースがアンダースコアに置き換えられた名前になります。`xlsub2script` が完了すると、生成したスクリプトの使用法を示したヘルプ メッセージが表示されます。`xlsub2script` 関数の主な目的は、**System Generator** スクリプトを簡単に習得できるようにすることです。また、このユーティリティを使用すると、グラフィックを使用しながらサブシステムを構築し、**PG API** の **M** 関数に変換できます。

`xlsub2script(block)` の `block` はリーフ ブロックで、ブロックを作成する `xBlock` 呼び出しを表示します。

次に、`xlsub2script` の制限事項を示します。

- サブシステムに `gcb`、`set_param`、`get_param`、`add_block` などの関数呼び出しを含むマスク初期化コードがある場合、関数がエラーになるので、コードからこれらの呼び出しを削除する必要があります。
- サブシステム内のグローバル変数にアクセスできる場合、`xlsub2script` を実行する最上位サブシステムに対応するマスク パラメータを追加する必要があります。
- ブロックのリンクが壊れている場合は、そのブロックがスキップされます。

`xlsub2script` は、次の方法でも実行できます。

```
xlsub2script(subsystem, options)
```

`options` には、**MATLAB** 構造体を指定します。 `options` 構造体には、`forcewrite` および `basevars` という 2 つのフィールドがあります。

`xlsub2script` が同じサブシステムで 2 回実行されると、既存の **M** 関数ファイルへの上書きが試行されます。デフォルトでは、このファイルに上書きするかを尋ねるポップアップ ダイアログ ボックスが表示されます。 `options` 引数の `forcewrite` フィールドを `true` または `1` に設定すると、このダイアログ ボックスが表示されずに **M** 関数ファイルが上書きされます。

サブシステムは、**MATLAB** の基本ワークスペースに含まれる一部の変数に基づいている場合があります。この場合、`xlsub2script` を実行するときに、これらの変数が選択されて、基本ワークスペース変数を処理するコードが生成されるようにする必要があります。このために `options` 引数の `basevars` フィールドを使用します。基本ワークスペースの変数すべてが選択されるようにするには、`basevars` フィールドを `all` に設定する必要があります。特定の変数のみが選択されるようにするには、`basevars` フィールドを文字列のセル配列にして、各文字列に変数名を指定します。

次に、`options` 引数を使用して `xlsub2script` を呼び出す例を示します。

```
xlsub2script(subsystem, struct('forcewrite', true));  
xlsub2script(subsystem, struct('forcewrite', true, 'basevars',  
                                'all'));  
  
options.basevars = {'var1', 'var2', 'var3'};  
xlsub2script(subsystem, options);  
xlsub2script(subsystem, struct('basevars', {{'var1', 'var2',  
                                              'var3'}}));
```

メモ: **MATLAB** では、構造体のフィールドがセル配列の場合、`struct()` 関数呼び出しを呼び出すときに `{}` がもう 1 つ必要になります。

xBlock ヘルプ

`xBlockHelp(<block_name>)` は、パラメータ名およびそのパラメータで許容される値を表示します。パラメータなしで実行する場合、`xbsIndex_r4` ライブラリで使用可能なブロックが表示されます。

たとえば、**MATLAB** コマンド ラインで次を実行したとします。

```
xBlockHelp('AddSub')
```

次の表が表示されます。

'xbsIndex_r4/AddSub' Parameter Table

Parameter	Acceptable value	Type
mode	'Addition' 'Subtraction' 'Addition or Subtraction'	String
use_carryin	'off' 'on'	String
use_carryout	'off' 'on'	String
en	'off' 'on'	String
latency	An Int value	Int
precision	'Full' 'User Defined'	String
arith_type	'Signed (2's comp)' 'Unsigned'	String
n_bits	An Int value	Int
bin_pt	An Int value	Int
quantization	'Truncate' 'Round (unbiased: +/- Inf)'	String
overflow	'Wrap' 'Saturate' 'Flag as error'	String
use_behavioral_HDL	'off' 'on'	String
pipelined	'off' 'on'	String
use_rpm	'off' 'on'	String

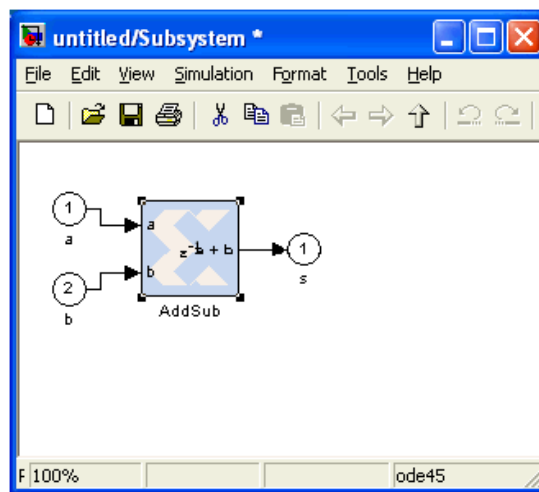
PG API の例

Hello World

この例では、MATLAB コマンド シェルにコマンドを入力できる learning モードで PG API を実行します。

1. 新しい learning セッションを開始するには、MATLAB コマンド コンソールに次を入力します。
xBlock
2. 次の 3 つのコマンドを入力して、Subsystem という名前のサブシステムを untitled という名前のモデル内に作成します。

```
[a, b] = xInport('a', 'b');  
s = xOutport('s');  
adder = xBlock('AddSub', struct('latency', 1), {a, b}, {s});
```



上記のコマンドでは、Simulink 入力ポート (a と b) の 2 個、レイテンシが 1 の加算ブロック、および Simulink 出力ポート (s) があるサブシステムが作成されます。2 個の入力ポートは加算器に接続され、加算器はサブシステムの出力ポートに接続されます。AddSub パラメータでは、xbsIndex_r4 ライブラリに含まれる AddSub が参照されます。ブロックの完全パスが指定されていない場合、デフォルトでは xBlock で xbsIndex_r4 およびビルトイン ライブラリが代わりに検索されます。ライブラリは、xBlock を使用する前に読み込まれる必要があります。load_system を使用して、ライブラリを読み込んでから xBlock を実行してください。

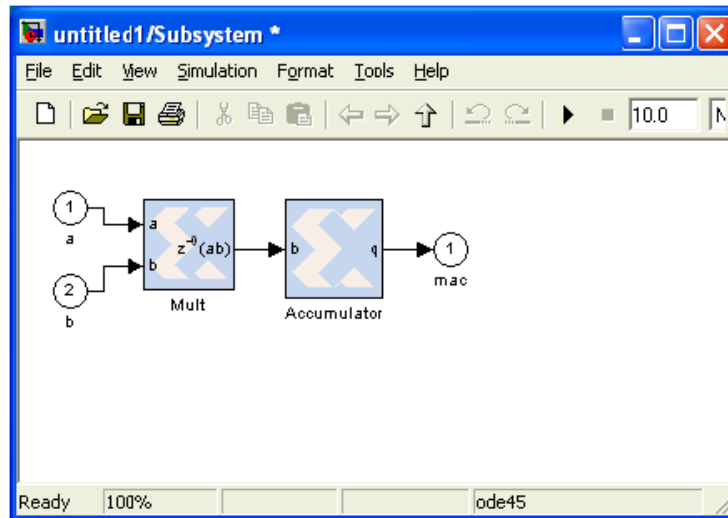
デバッグのヒント：MATLAB コンソールで「adder」と入力すると、System Generator で加算器ブロックの概要が表示され、Simulink ダイアグラムでこのブロックがハイライトされます。同様に「a」、「b」、および「s」と入力して、サブシステムの入力ポートと出力ポートをハイライトできます。

MACC

1. この例を **learning** モードで実行します。新規の **learning** セッションを開始するには、**xBlock** を実行します。
2. **MATLAB** コンソール ウィンドウで次のコマンドを入力して、新しいサブシステムで **MAC** ファンクションを作成します。

```
[a, b] = xInport('a', 'b');
mac = xOutput('mac');
m = xSignal;
mult = xBlock('Mult', struct('latency', 0, 'use_behavioral_HDL', 'on'),
{a, b}, {m});
acc = xBlock('Accumulator', struct('rst', 'off', 'use_behavioral_HDL',
'on'), {m}, {mac});
```

System Generator でビヘイビア HDL を生成することで、2 個のブロックが **DSP48** ブロック 1 個にパックされます。現段階では、**XST** では乗算器ブロックを強制的に組み合わせブロックにしない限り、**DSP48** ブロック 1 個にパックされません。



メモ：例 1 で作成されたモデルを閉じない場合は、例 2 が **untitled1** という名前のモデル内に作成されます。それ以外は、新しいモデル **untitled** が作成されます。

デバッグのヒント：PG API では、生成されたサブシステムに含まれるブロックおよび信号の情報を得る関数が提供されています。次のコマンドをそれぞれ実行してから、**MATLAB** コンソールに表示される内容と **Simulink** ダイアグラムでの変化を確認してください。

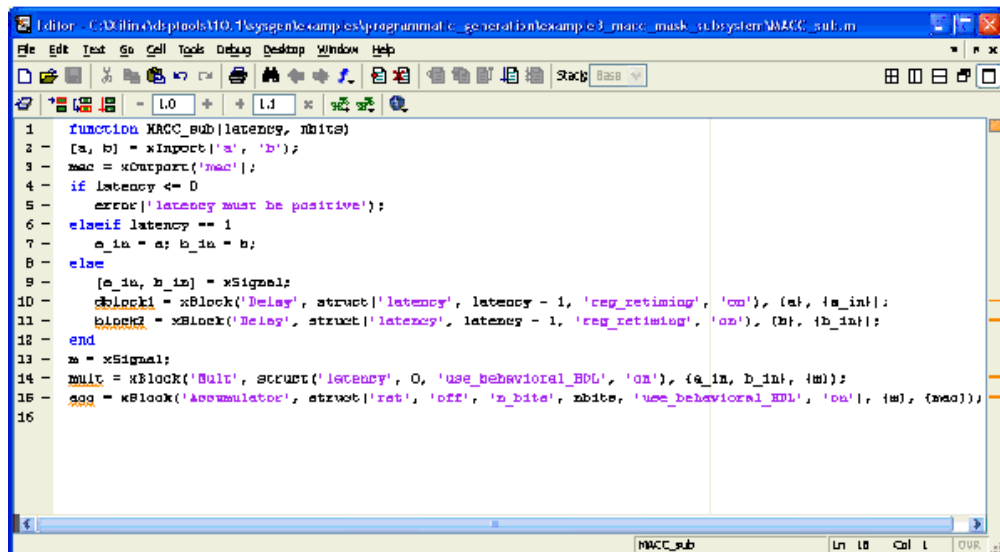
```
mult_ins = mult.getInSignals
mult_ins{1}
mult_ins{2}
src_a = mult_ins{1}.getSrc
src_a{1}
m_dst = m.getDst
m_dst{1}
m_dst{1}.block
```


マスクされたサブシステム内の MACC

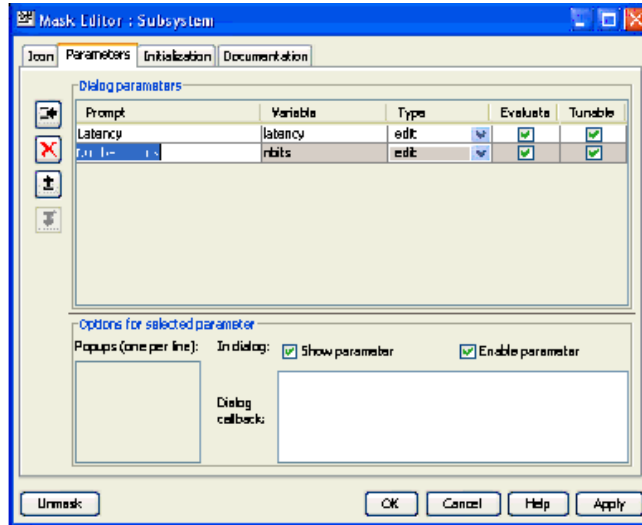
PG API で特定のサブシステムを作成してそのサブシステムのマスク パラメータから PG API にパラメータを渡す場合、PG API を production モードで実行する必要があります。このモードでは、物理的な M 関数ファイルが必要で、関数を xBlock コンストラクタに渡す必要があります。

1. まず、最上位 PG API の M 関数ファイル MACC_sub.m を次のコードを使用して作成します。

```
function MACC_sub(latency, nbits)
[a, b] = xInport('a', 'b');
mac = xOutport('mac');
if latency <= 0
    error('latency must be positive');
elseif latency == 1
    a_in = a; b_in = b;
else
    [a_in, b_in] = xSignal;
    dblock1 = xBlock('Delay', struct('latency', latency - 1,
    'reg_retiming', 'on'), {a}, {a_in});
    block2 = xBlock('Delay', struct('latency', latency - 1,
    'reg_retiming', 'on'), {b}, {b_in});
end
m = xSignal;
mult = xBlock('Mult', struct('latency', 0, 'use_behavioral_HDL', 'on'),
{a_in, b_in}, {m});
acc = xBlock('Accumulator', struct('rst', 'off', 'n_bits', nbits,
'use_behavioral_HDL', 'on'), {m}, {mac});
```



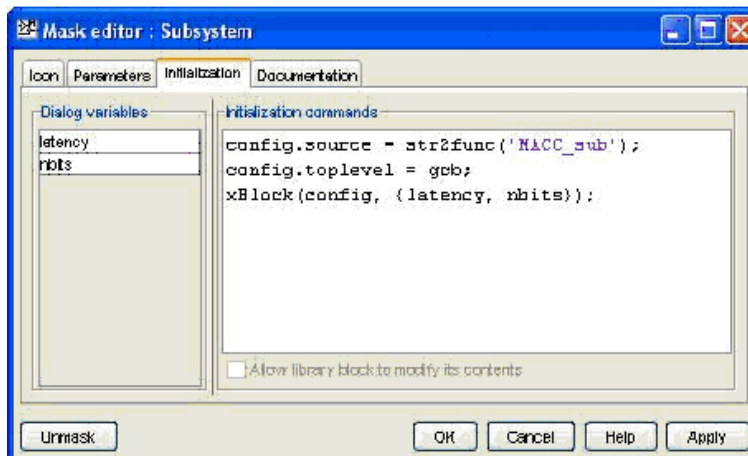
2. このスクリプトで定義されたサブシステムをマスクするには、マスク パラメータである `latency` および `nbits` を追加します。



3. 次に、次のコード行をサブシステムのマスク初期化コマンドに含めます。

```
config.source = str2func('MACC_sub');
config.toplevel = gcb;
xBlock(config, {latency, nbits});
```

production モードでは、`xBlock` コンストラクタの最初の引数がコンフィギュレーション用の MATLAB 構造体です。この MATLAB 構造体には `source` フィールドと `toplevel` フィールドが必要です。`source` フィールドは、M 関数への関数ポインタで、`toplevel` フィールドは Simulink サブシステムを指定する文字列です。`toplevel` フィールドが 1 の場合、名前が付けられていないモデルが作成され、そのモデル内にサブシステムが作成されます。

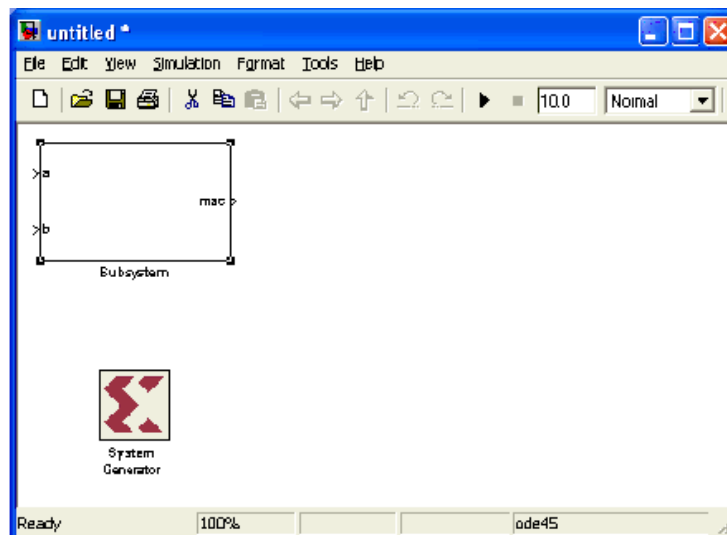


次のように MATLAB 構造体呼び出しを使用して、最上位コンフィギュレーションを作成することもできます。

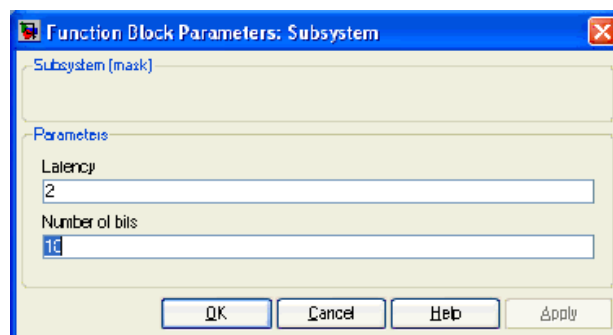
```
xBlock(struct('source', str2func(MACC_sub), 'toplevel', gcb), {latency,
                                                                nbits});
```

[OK] をクリックします。

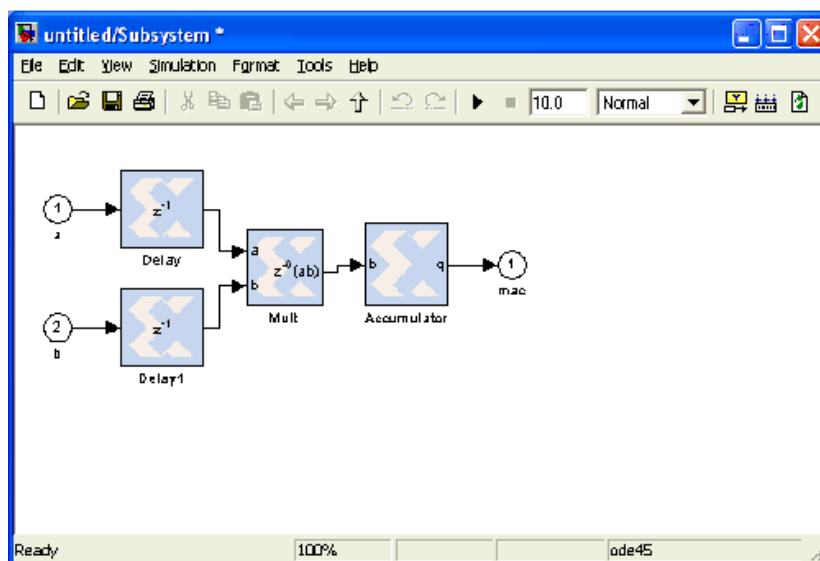
次のサブシステムが作成されます。



4. 次の図に示すようにマスク パラメータを設定してから、[OK] をクリックします。



次のダイアグラムが生成されます。



デバッグのヒント：MATLAB エディタで `MACC_sub.m` を開いて関数をデバッグします。デフォルトでは、`xBlock` コンストラクタにより最後に自動レイアウトが実行されます。ブロックの追加ごとに自動レイアウトを確認する場合は、次のように入力して最上位の `xBlock` を起動します。

```
config.source = str2func('MACC_sub');  
config.toplevel = gcb;  
config.debug = 1;  
xBlock(config, {latency, nbits});
```

コンフィギュレーション構造体の `debug` フィールドを 1 に設定すると、PG API を `debugging` モードで実行することになり、アクションごとに自動レイアウトが実行されます。

キャッシュのヒント：ほとんどの場合で、必要な場合のみにサブシステムを再生成することが望めます。`xBlock` コンストラクタには、キャッシュ メカニズムが備わっています。セル配列に依存ファイルのリストを指定し、最上位コンフィギュレーションの `depend` フィールドをこのリストを使用して設定できます。`depend` リストに含まれるファイルが変更されたり、最上位の関数に渡される引数リストが変更されると、サブシステムが再生成されます。`MACC_sub` でキャッシュ機能を使用する場合は、最上位の `xBlock` を次のように実行します。

```
config.source = str2func('MACC_sub');  
config.toplevel = gcb;  
config.depend = {'MACC_sub.m'};  
xBlock(config, {latency, nbits});
```

コンフィギュレーション構造体の `depend` フィールドは、セル配列です。配列の各要素はファイル名で、**P** ファイルの名前または **M** ファイルの名前を入力できます。また、拡張子なしでも使用できます。`xBlock` では、パスで最初に検出されるファイルが使用されます。

PG API エラー / 警告の状況およびメッセージ

xBlock エラー メッセージ

状況	エラー メッセージ
<code>xBlock(NoSubSourceBlock, ...)</code> を呼び出すとき、ソース ブロックが見つかりません。	Source block NoSubSourceBlock cannot be found.
<code>xBlock(sourceBlock, parameterBinding)</code> を呼び出すときにパラメータの値が不正な場合、 <code>xBlock</code> でパラメータエラーがレポートされます。例： <code>xBlock('AddSub', struct('latency', -1));</code>	Illegal parameterization: Latency Latency is set to a value of -1, but the value must be greater than or equal to 0
入力ポートの結合リストに <code>xSignal</code> または <code>xInport</code> 以外のオブジェクトが含まれています。	Only objects of <code>xInport</code> or <code>xSignal</code> can appear in inport binding list.
出力ポートの結合リストに <code>xSignal</code> または <code>xOutport</code> 以外のオブジェクトが含まれています。	Only objects of <code>xOutport</code> or <code>xSignal</code> can appear in outport binding list.
<code>xBlock</code> の最初の引数が関数ポインタの場合、2 番目の引数はセル配列であることが想定されます。それ以外が引数に指定されると、エラーが発生します。	Cell array is expected for the second argument of the <code>xBlock</code> call
ソースのコンフィギュレーション構造体の最上位が定義されている場合、 Simulink サブシステムを指定し、 char 型の配列にする必要があります。 char 型以外の配列の場合は、エラーが発生します。	Top level must be a char array
出力ポートの結合リストに含まれるオブジェクトがすでに駆動されている場合、つまりは 2 つの駆動ソースを持たせるような場合に、エラーが発生します。(メモ: このエラーメッセージは分かりづらいので、修正される予定です。)	Source of <code>xSignal</code> object already exists

xInport エラー メッセージ

状況	エラー メッセージ
同じ名前で xInport オブジェクトを再度作成しようとする、エラーが発生します。たとえば、 <code>p = xInport('a', 'a')</code> を呼び出すとエラーになります。	A new block named 'untitled/Subsystem/a' cannot be added.

xOutput エラー メッセージ

状況	エラー メッセージ
同じ名前で xOutput オブジェクトを再度作成しようとする、エラーが発生します。たとえば、 <code>p = xOutput('a', 'a')</code> を呼び出すとエラーになります。	A new block named 'untitled/Subsystem/a' cannot be added.
xOutput オブジェクトを 2 回結合しようとする、エラーが発生します。たとえば、次の呼び出しシーケンスはエラーになります。 <code>[a, b] = xInport('a', 'b'); c = xOutput('c'); c.bind(a); c.bind(b);</code>	The destination port already has a line connection.

xSignal エラー メッセージ

状況	エラー メッセージ
xSignal オブジェクトを 2 つのソースを使用して結合しようとする、エラーが発生します。たとえば、次の呼び出しシーケンスはエラーになります。 <code>[a, b] = xInport('a', 'b'); sig = xSignal; sig.bind(a); sig.bind(b);</code>	Source of xSignal object already exists.

xsub2script エラー メッセージ

状況	エラー メッセージ
xsub2script が引数なしで起動されます。	An argument is expected for xsub2script
最初の引数がサブシステムではないか、またはモデルが開いていません。	The first argument must be a model, subsystem, or a block. Please make sure the model is opened or the argument is a valid string for a model or a block.

状況	エラー メッセージ
サブシステムの初期化コードに Simulink 関数呼び出しが含まれています。	Subsystem has Simulink function calls, such as gcb, get_param, set_param, add_block. Please remove these calls and run xlsb2script again or you can pick a different subsystem to run xlsb2script.
サブシステムに Goto ブロックがあります。	You have the following Goto blocks, please modify the model to remove them and run xlsb2script again.

Shared Memory ブロックへの C++ アクセス

System Generator API では、ファイル `SharedMemory.h` で定義されているクラスが主に使用されています。各クラス内には、System Generator データへのアクセスを可能にする関数があります。この API には、`SharedMemory.h` (`sysgen/include` 内) および `sysgen.dll` (`sysgen/lib` 内) へのリンクを含める必要があります。

次のオブジェクトが、System Generator の名前空間で定義されています。

「[SharedMemory](#)」クラス

「[LockableSharedMemory](#)」クラス

「[SharedMemoryProxy](#)」クラス

「[Request Struct](#)」

「[NamedPipeReader](#)」クラス

「[NamedPipeWriter](#)」クラス

ハードウェア協調シミュレーションへの M コード アクセス

System Generator のハードウェア協調シミュレーションを使用すると、Simulink シミュレーション環境でオンチップ高速シミュレーションおよび検証機能が使用できます。通常の System Generator フローでは、まず System Generator モデルがハードウェア協調プラットフォーム用にコンパイルされ、この間にデザインのハードウェア インプリメンテーション (ビットストリーム) が生成され、ハードウェア協調シミュレーション ブロックに関連付けられます。このブロックが Simulink モデルに挿入されて、ポートがソースおよびシンク ブロックに接続されます。コンパイルされた System Generator デザインが FPGA デバイスで実行される間、モデル全体がシミュレーションされます。

このような動作の代わりに、System Generator のハードウェア協調シミュレーション フローで作成されたハードウェアを MATLAB の M コード を使用してプログラム制御できます (M コード/ハードウェア協調シミュレーション)。M コード/ハードウェア協調シミュレーション インターフェイスを使用すると、Simulink のフレームワークと関係なく、ハードウェアに対応する MATLAB オブジェクトを純粋な M コード内に生成できます。これらのオブジェクトは、ハードウェアからデータを読み出したり、ハードウェアにデータを書き込んだりするのに使用できます。

この機能は、スクリプト インターフェイスをハードウェア協調シミュレーションに提供するときにより、ハードウェアをスクリプト テストベンチで使用したり、M コード内にハードウェア高速シミュレーションとして投入できます。System Generator によるシミュレーションでのスケジューリング セマンティクスと異なり、M コード/ハードウェア協調シミュレーションでは、柔軟に任意のスケジュールを使用できます。デザインの動作が分かっている場合は、この融通性を使用してシミュ

レーション時間を短縮できます。また、M コード/ハードウェア協調シミュレーション オブジェクトでは、MATLAB コンソールからハードウェアにアクセスが可能で、ハードウェアの内部ステートを対話的に観察できます。

M コード/ハードウェア協調シミュレーションで使用するハードウェアのコンパイル

M コード/ハードウェア協調シミュレーションで使用するハードウェアをコンパイルするには、System Generator による典型的なハードウェア協調シミュレーション フローと同様のフローに従います。最初に、Simulink で System Generator モデルから開始して、System Generator トークンでハードウェア協調シミュレーションを選択してから [Generate] をクリックします。生成プロセスの最後に、ハードウェア協調シミュレーション ライブラリが作成されます。

ネットリスト ディレクトリに、BIT ファイルと HWC ファイルが含まれます。BIT ファイルは FPGA インプリメンテーションに対応し、HWC ファイルは M コード/ハードウェア協調シミュレーションに必要な情報を含んでいます。BIT ファイルおよび HWC ファイルの名前は、mydesign_cw.bit および mydesign_cw.hwc のように対になっています。

HWC ファイルには、デザインおよび選択されたハードウェア協調シミュレーション インターフェイスを記述したメタ情報が指定されています。このメタ情報と共に、M コード/ハードウェア協調シミュレーションを使用してハードウェア協調シミュレーション インスタンスをインスタンスエートできます。このインスタンスを使用すると、協調シミュレーション エンジンと通信できます。

M コード/ハードウェア協調シミュレーションでは、既存の協調シミュレーション ブロックに含まれるポート、共有メモリ、固定小数点の表記法と同じ表記法が使用されます。すべてのデザインで最上位ポートおよびエンベデッドされた共有メモリが外部からアクセスできるようになっています。

M コード/ハードウェア協調シミュレーション セマンティクス

M コード/ハードウェア協調シミュレーションのシミュレーション セマンティクスは、System Generator ブロック ダイアグラムでのハードウェア協調シミュレーションで 사용되는ものと異なり柔軟性が高く、ブロック ベースのハードウェア協調シミュレーションで 사용되는シミュレーション セマンティクスをエミュレートできます。

ブロック ベースのハードウェア協調シミュレーションでは、厳格なシミュレーション セマンティクスが強いられ、クロック サイクルを進める前に、すべての入力ポートが記述されます。次に、すべての出力ポートが読み出されて、クロックが進みます。M コード/ハードウェア協調シミュレーションでは、ポートの読み出しおよび書き込みのスケジューリングはユーザーによって決定されます。たとえば、1 クロック サイクルおきに特定のポートにデータを書き込んだり、特定のクロック サイクル間出力を読み出すプログラムを作成できます。この柔軟性により、データ転送を最適化してシミュレーション時間を短縮できます。

データ表現

M コード/ハードウェア協調シミュレーションでは、同時に外部エンティティに対して倍精度の浮動小数点の値が生成されますが、内部では固定小数点データ型が使用されます。共有メモリのポートまたはメモリ ロケーションに渡されるすべてのデータ サンプルは、固定小数点の値です。各サンプルには、現在のデータ幅とコンパイル時に固定される明示的な 2 進小数点位置があります。データの倍精度から固定小数点への変換は、M コード/ハードウェア協調シミュレーションの境界で発生します。現段階のインプリメンテーションでは、入力データの量子化は丸め処理され、オーバーフローは飽和処理されます。

M コードからハードウェアへのインターフェイス

モデルがハードウェア協調シミュレーション用にコンパイルされている場合、生成されたビットストリームはモデルベースの Simulink フローと MATLAB で実行される M コードの両方で使用できます。ハードウェアのビットストリームにアクセスする一般的な動作のシーケンスは、通常次のようになります。

1. ハードウェア協調シミュレーション インターフェイスをコンフィギュレーションします。ハードウェア協調シミュレーションのコンフィギュレーションは永続で、HWC ファイルに保存されることに注意してください。協調シミュレーション インターフェイスを変更しない限り、この手順を再実行する必要はありません。
2. デザインの M コード/ハードウェア協調シミュレーション インスタンスを作成します。
3. M コード/ハードウェア協調シミュレーション インターフェイスを開きます。
4. シミュレーションが完了するまで次の手順を繰り返し実行します。
 - a. シミュレーション データを入力ポートに書き込みます。
 - b. 出力ポートからシミュレーション データを読み出します。
 - c. デザイン クロックを 1 サイクルごとに進めます。
5. M コード/ハードウェア協調シミュレーション インターフェイスを閉じます。
6. M コード/ハードウェア協調シミュレーション インスタンスを解放します。

M コード/ハードウェア協調シミュレーションの例

チュートリアル例 : MATLAB ハードウェア協調シミュレーションの使用 (M-Hwcosim)

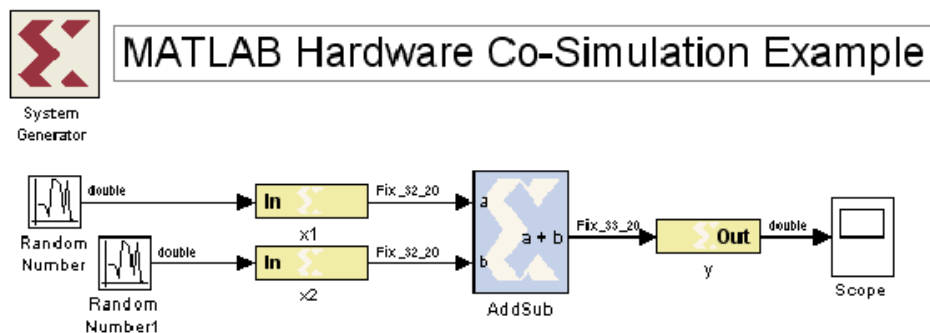
このチュートリアルでは、単純な AddSub モデルを使用した MATLAB ハードウェア協調シミュレーションの実行方法について説明します。AddSub モデルは入力 2 つの出力 1 つ (オペランド 2 つ (x1 と x2)、および合計出力 1 つ (y)) で構成されています。

メモ : このチュートリアルの手順では、イーサネット ハードウェア協調シミュレーション用に、ML506 プラットフォームでの実行に必要なハードウェアとソフトウェアを両方ともインストールおよびコンフィギュレーション済みであることを前提にして説明しています。このプラットフォームのインストール方法とコンフィギュレーション方法については、「イーサネット ハードウェア協調シミュレーション用の ML506 ボードのインストール」を参照してください。

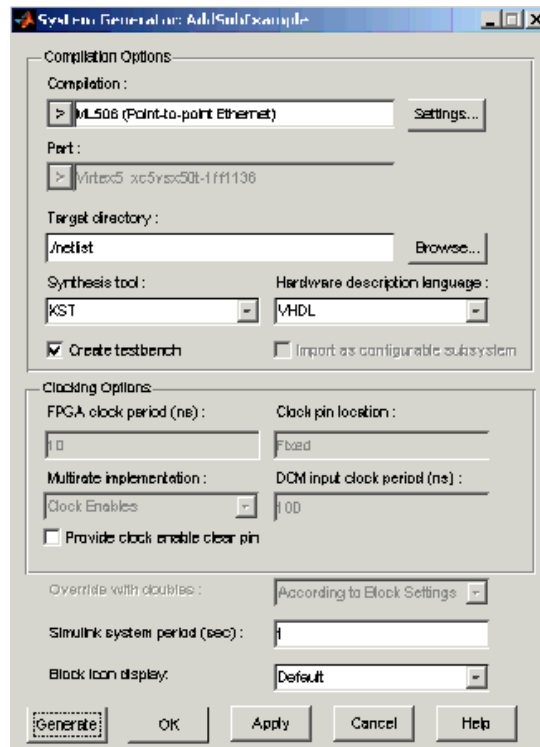
AddSubExample デザインは、次のディレクトリにあります。

```
<sysgen_tree>/examples/mhwcosim/AddSubExample.mdl
```

1. MATLAB でモデルを開いて、次のブロックを確認します。

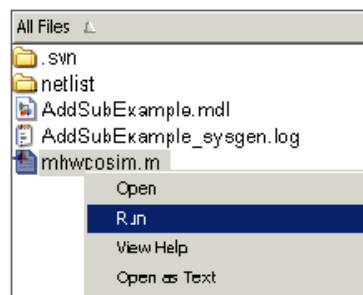


2. System Generator ブロックをダブルクリックして、次のダイアログ ボックスを開きます。.



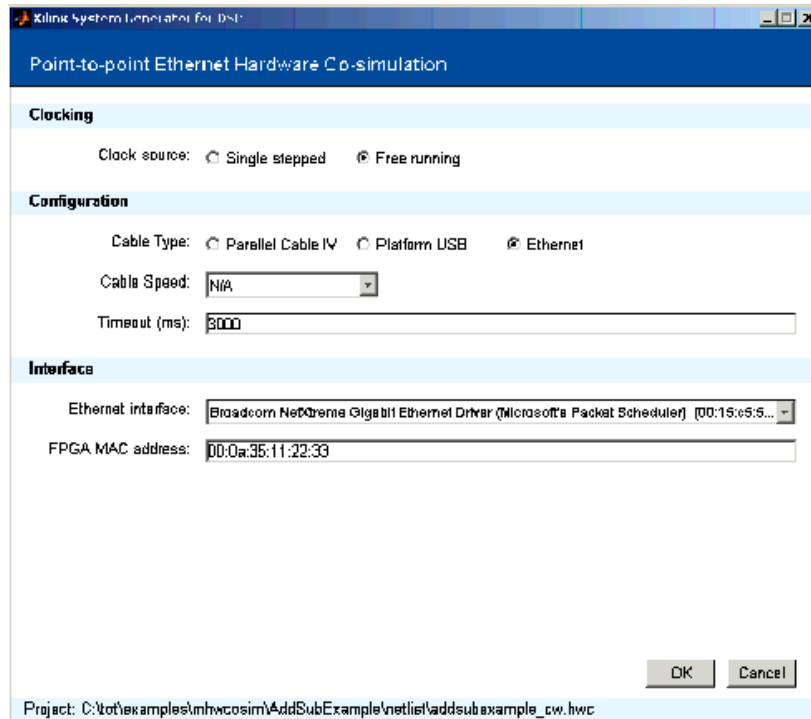
上記のように、[Create testbench] チェックボックスをオンにすると、ハードウェア協調シミュレーションのコンパイル フローに M コード スクリプトの <design>_hwcosim_test.m とゴールデン テスト データ ファイルの <design>_<port>_hwcosim_test.dat が Simulink シミュレーションに基づいて各ゲートウェイごとに自動生成されます。この後、ネットリストの名前が付いたサブディレクトリが現在の作業ディレクトリに作成され、生成されたファイルが含まれます。テストベンチの自動生成については、「[M コード/ハードウェア協調シミュレーション テストベンチの自動生成](#)」を参照してください。

3. [Generate] をクリックし、ハードウェア協調シミュレーションのネットリスト生成を開始します。ここまでは、デザイン フローは、Simulink ハードウェアのフローと全く同じです。
4. ネットリストが生成されたら、MATLAB のハードウェア協調シミュレーションを実行します。MATLAB コンソールに mhwcosim と入力するか、そのファイルを右クリックして [Run] を選択し、提供されている M コード スクリプトのパス (./examples/mhwcosim/AddSubExample/AddSubExample.mdl/mhwcosim.m) を入力します。



メモ：この M コード スクリプトは、自動生成された M コード スクリプトを少し修正して作成され、シミュレーション結果の一部を表示します。

5. 最初にモデルをシミュレーションするには、次のコンフィギュレーション ダイアログ ボックスが表示されます。コンピュータやケーブル タイプに従ってパラメータを設定し、[OK] をクリックします。



6. 約 30 秒後、MATLAB コンソールに次のようなシミュレーション結果が表示されます。

Simulation OK

Cycle	Expected values	Actual values
0	2.3299064636230469	2.3299064636230469
1	3.3922843933105469	3.3922843933105469
2	-2.8923435211181641	-2.8923435211181641
3	-0.7200584411621094	-0.7200584411621094
4	-0.0897617340087891	-0.0897617340087891
5	1.0269565582275391	1.0269565582275391
6	0.7500820159912109	0.7500820159912109
7	-0.6458797454833984	-0.6458797454833984
8	1.6952972412109375	1.6952972412109375
9	-1.1141872406005859	-1.1141872406005859

まとめ

System Generator には、Simulink ハードウェア協調シミュレーションだけでなく、MATLAB ハードウェア協調シミュレーションにより、ハードウェア協調シミュレーションを実行する方法もあります。これにより、System Generator のハードウェア協調シミュレーション フローで作成されたハードウェアを MATLAB の M コード を使用してプログラム制御できます (M コード/ハードウェア協調シミュレーション)。M コード/ハードウェア協調シミュレーション インターフェイスを使用すると、Simulink のフレームワークと関係なく、ハードウェアに対応する MATLAB オブジェクトを純粋な M コード内に生成できます。これらのオブジェクトは、ハードウェアからデータを読み出したり、ハードウェアにデータを書き込んだりするのに使用できます。読み出し、書き込み、その他サポートされる機能の詳細については、「[M コード/ハードウェア協調シミュレーションの MATLAB クラス](#)」を参照してください。

この機能は、スクリプト インターフェイスをハードウェア協調シミュレーションに提供するときに便利で、ハードウェアをスクリプト テストベンチで使用したり、M コード内にハードウェア高速シミュレーションとして投入できます。アプリケーションの中には、このハードウェア協調シミュレーション方法を使用することで、パフォーマンスが一部改善されるものもあります。

例 2

M コードでは、例 1 で記述されたシミュレーションを実行するのに別の構文形式が使用されます。この形式で `exec` 命令を使用すると、シミュレーション時間が短縮されます。これは、ブロックのポートを識別するために必要な名前に基づくルックアップ数が削減され、また M コードで繰り返し実行される実行コードが 1 つの命令にまとめられることにより、M コードの割り込みで発生するオーバーヘッドが削減されるためです。

```
% Configure the co-simulation interface. Note: This needs only to be
% done once, since the configuration is stored back into the hwc file
% This will launch a configuration GUI.
xlHwcosimConfig('mydesign.hwc');

% Define the number of simulation cycles.
nCycles = 1000;

% Creates a hardware co-simulation instance from the project
% 'mydesign.hwc'.
h = Hwcosim('mydesign.hwc');

% Opens and configures the hardware co-simulation interface.
open(h);

% Initializes the 'op' input port with a constant value zero.
write(h, 'op', 0);

% Initializes an execution definition that covers the input ports,
% x1 and x2, and the output ports y. It returns an execution
% identifier for use in subsequent exec instructions.
execId = initExec(h, {'x1', 'x2'}, {'y'});

% Simulate the design using the exec instruction.
% The input data are given as a 2-D matrix. Each row of the matrix
% gives the simulation data of an input port for all the cycles.
% For example, row i column j stores the data for the i-th port at
% (j-1)th cycle.
result = exec(h, execId, nCycles, rand(2, nCycles));
```

```
% Releases the hardware co-simulation instance.  
% The hardware co-simulation interface is closed implicitly.  
release(h);
```

例 3

この例では、M コード/ハードウェア協調シミュレーション フローで M コードを使用して Shared Memory にアクセスする方法を示します。この例では、MyMem という名前の Shared Memory、WriteFifo および ReadFifo という名前の SharedFifo がある System Generator モデルがハードウェア協調シミュレーション ブロックにコンパイルされていると想定します。

メモ：この例のモデルおよびソース コードは、<path_to_sysgen>/examples/mhwcossim/ShMemExample にあります。

```
% Creates a hardware co-simulation instance from the project  
'shmem.hwc'.  
h = Hwcossim('shmem.hwc');  
  
% Opens and configures the hardware co-simulation interface.  
open(h);  
  
% Creates a shared memory instance 'MyMem'. It connects the  
corresponding  
% shared memory running in hardware.  
m = Shmem('MyMem');  
  
% Creates a shared FIFO instance 'WriteFifo' for writing data to the  
% hardware. Similarly, creates another shared FIFO instance 'ReadFifo'  
for  
% reading data from the hardware.  
wf = Shfifo('WriteFifo');  
rf = Shfifo('ReadFifo');  
  
% Writes random numbers to memory address 0 to 49 of MyMem.  
m(0:49) = rand(1, 50);  
  
% Read the value at memory address 100 of MyMem.  
y = m(100);  
  
% Writes 10 random numbers to WriteFifo if it has 10 or more empty  
space.  
if wf.Available >= 10  
    write(wf, 10, rand(1, 10));  
end  
  
% Reads 5 values from ReadFifo if it has 5 or more data.  
if rf.Available >= 5  
    d = read(rf, 5);  
end  
  
% Releases the shared memory instances.  
release(m);  
release(wf);  
release(rf);  
  
% Releases the hardware co-simulation instance.  
release(h);
```

M コード/ハードウェア協調シミュレーション テストベンチの自動生成

M コード/ハードウェア協調シミュレーションでは、ハードウェア協調シミュレーション用のテストベンチを生成できます。System Generator で [Create testbench] がオンのとき、Simulink シミュレーションに基づく各ゲートウェイに対して、ハードウェア協調シミュレーションのコンパイルフローにより M コード スクリプト (<design>_hwcosim_test.m) およびゴールデン テスト データ ファイル (<design>_<port>_hwcosim_test.dat) が生成されます。M コード スクリプトでは、M コード/ハードウェア協調シミュレーション API を使用してハードウェアに含まれるデザインをシミュレーションして、結果をゴールデン テスト データと比較して検証するテストベンチがインプリメントされます。シミュレーションで不一致が検出された場合は、<design>_hwcosim_test.results にレポートされます。

例 4 で示すように、テストベンチ コードは読みやすく、独自のシミュレーション コードの基礎として使用できます。

メモ：この例のモデルおよびソース コードは、<path_to_sysgen>/examplesmhwcosim/MultiRatesExample にあります。

例 4

```
function multi_rates_cw_hwcosim_test
try
    % Define the number of hardware cycles for the simulation.
    ncycles = 10;

    % Load input and output test reference data.
    testdata_in2 = load('multi_rates_cw_in2_hwcosim_test.dat');
    testdata_in3 = load('multi_rates_cw_in3_hwcosim_test.dat');
    testdata_in7 = load('multi_rates_cw_in7_hwcosim_test.dat');
    testdata_pb00 = load('multi_rates_cw_pb00_hwcosim_test.dat');
    testdata_pb01 = load('multi_rates_cw_pb01_hwcosim_test.dat');
    testdata_pb02 = load('multi_rates_cw_pb02_hwcosim_test.dat');
    testdata_pb03 = load('multi_rates_cw_pb03_hwcosim_test.dat');
    testdata_pb04 = load('multi_rates_cw_pb04_hwcosim_test.dat');

    % Pre-allocate memory for test results.
    result_pb00 = zeros(size(testdata_pb00));
    result_pb01 = zeros(size(testdata_pb01));
    result_pb02 = zeros(size(testdata_pb02));
    result_pb03 = zeros(size(testdata_pb03));
    result_pb04 = zeros(size(testdata_pb04));

    % Initialize sample index counter for each sample period to be
    % scheduled.
    insp_2 = 1;
    insp_3 = 1;
    insp_7 = 1;
    outsp_1 = 1;
    outsp_2 = 1;
    outsp_3 = 1;
    outsp_7 = 1;

    % Define hardware co-simulation project file.
    project = 'multi_rates_cw.hwc';

    % Create a hardware co-simulation instance.
    h = Hwcosim(project);
```

```

% Open the co-simulation interface and configure the hardware.
try
    open(h);
catch
    % If an error occurs, launch the configuration GUI for the user
    % to change interface settings, and then retry the process again.
    release(h);
    xlHwcosimConfig(project, true);
    drawnow;
    h = Hwcosim(project);
    open(h);
end

% Simulate for the specified number of cycles.
for i = 0:(ncycles-1)

    % Write data to input ports based their sample period.
    if mod(i, 2) == 0
        h('in2') = testdata_in2(insp_2);
        insp_2 = insp_2 + 1;
    end
    if mod(i, 3) == 0
        h('in3') = testdata_in3(insp_3);
        insp_3 = insp_3 + 1;
    end
    if mod(i, 7) == 0
        h('in7') = testdata_in7(insp_7);
        insp_7 = insp_7 + 1;
    end

    % Read data from output ports based their sample period.
    result_pb00(outsp_1) = h('pb00');
    result_pb04(outsp_1) = h('pb04');
    outsp_1 = outsp_1 + 1;
    if mod(i, 2) == 0
        result_pb01(outsp_2) = h('pb01');
        outsp_2 = outsp_2 + 1;
    end
    if mod(i, 3) == 0
        result_pb02(outsp_3) = h('pb02');
        outsp_3 = outsp_3 + 1;
    end
    if mod(i, 7) == 0
        result_pb03(outsp_7) = h('pb03');
        outsp_7 = outsp_7 + 1;
    end

    % Advance the hardware clock for one cycle.
    run(h);

end

% Release the hardware co-simulation instance.
release(h);

% Check simulation result for each output port.
logfile = 'multi_rates_cw_hwcosim_test.results';
logfd = fopen(logfile, 'w');
sim_ok = true;

```

```

        sim_ok = sim_ok & check_result(logfd, 'pb00', testdata_pb00,
result_pb00);
        sim_ok = sim_ok & check_result(logfd, 'pb01', testdata_pb01,
result_pb01);
        sim_ok = sim_ok & check_result(logfd, 'pb02', testdata_pb02,
result_pb02);
        sim_ok = sim_ok & check_result(logfd, 'pb03', testdata_pb03,
result_pb03);
        sim_ok = sim_ok & check_result(logfd, 'pb04', testdata_pb04,
result_pb04);
        fclose(logfd);
        if ~sim_ok
            error('Found errors in simulation results. Please refer to ''%s''
for details.', logfile);
        end

    catch
        err = lasterr;
        try release(h); end
        error('Error running hardware co-simulation testbench. %s', err);
    end

%-----

function ok = check_result(fd, portname, expected, actual)
    ok = false;

    fprintf(fd, ['\n' repmat('=', 1, 95), '\n']);
    fprintf(fd, 'Output: %s\n\n', portname);

    % Check the number of data values.
    nvals_expected = numel(expected);
    nvals_actual = numel(actual);
    if nvals_expected ~= nvals_actual
        fprintf(fd, ['The number of simulation output values (%d) differs '
...
                    'from the number of reference values (%d).\n'], ...
                    nvals_actual, nvals_expected);
        return;
    end

    % Check for simulation mismatches.
    mismatches = find(expected ~= actual);
    num_mismatches = numel(mismatches);
    if num_mismatches > 0
        fprintf(fd, 'Number of simulation mismatches = %d\n',
num_mismatches);
        fprintf(fd, '\n');
        fprintf(fd, 'Simulation mismatches:\n');
        fprintf(fd, '-----\n');
        fprintf(fd, '%10s %40s %40s\n', 'Cycle', 'Expected values', 'Actual
values');
        fprintf(fd, '%10d %40.16f %40.16f\n', ...
[mismatches-1; expected(mismatches); actual(mismatches)]);
        return;
    end

    ok = true;
    fprintf(fd, 'Simulation OK\n');

```


リソースの管理

M コード/ハードウェア協調シミュレーションでは、ハードウェア協調シミュレーション インスタンスに対して保持するリソースを管理します。解放命令を実行したときまたは MATLAB を終了したときにこれらのリソースが解放されます。ただし、シミュレーションが終了した場合やエラーが発生したときは、必ずリソースをクリーンアップしてください。エラーが発生した場合にクリーンアップを正しく実行できるよう、次に示すように MATLAB の try-catch ブロックに M コード/ハードウェア協調シミュレーション命令を含めておいてください。

```
try
    % M-Hwcosim instructions here
catch
    err = lasterror;
    % Release any Hwcosim, Shmem, or Shfifo instances
    try release(hwcosim_instance); end
    try release(shmem_instance); end
    try release(shfifo_instance); end
    rethrow(err);
end
```

次のコマンドを使用すると、すべてのハードウェア協調シミュレーション インスタンスまたは共有メモリ インスタンスを解放できます。

```
xlHwcosim('release');      % Release all Hwcosim instances
xlHwcosim('releaseMem');   % Release all Shmem instances
xlHwcosim('releaseFifo');  % Release all Shfifo instances
```

M コード/ハードウェア協調シミュレーションの MATLAB クラス

ハードウェア協調シミュレーション

ハードウェア協調シミュレーションの MATLAB クラスでは、ハードウェア協調シミュレーション エンジンの高度な抽象化が提供されます。インスタンス化されたハードウェア協調シミュレーションのオブジェクトは、それぞれハードウェア協調シミュレーション インスタンスを表します。オブジェクトには、そのインスタンスに関連する固有の識別子などのプロパティが組み込まれます。ほとんどの命令の実行で、ハードウェア協調シミュレーション オブジェクトは入力引数として取り込まれます。一部の操作では、簡略化のため代替の省略表現が用意されています。同様に、共有メモリや共有 FIFO に関連する操作にアクセスするための Shmem および Shfifo クラスが用意されています。

アクション	構文
コンストラクタ	<code>h = Hwcosim(project)</code>
デストラクタ	<code>release(h)</code>
ハードウェアを開く	<code>open(h)</code>
ハードウェアを閉じる	<code>close(h)</code>

アクション	構文
データを書き込む	<code>write(h, inPorts, inData)</code> <code>h(inPorts) = inData</code>
データを読み出す	<code>outData = read(h, outPorts)</code> <code>outData = h(outPorts)</code>
実行	<code>run(h)</code> <code>run(h, n)</code>
ベクトル化実行	<code>outData = exec(h, execId, nCycles, inData)</code>
プロパティの取得	<code>data = get(h, prop)</code>

コンストラクタ

構文

```
h = Hwcosim(project);
```

説明

ハードウェア協調シミュレーション インスタンスを作成します。インスタンスは、ハードウェア協調シミュレーション プロジェクトへのリファレンスで、ハードウェアへの明示的なリンクは指定しません。ハードウェア協調シミュレーション オブジェクトを作成すると、ハードウェア協調シミュレーション エンジンに **FPGA** のビットストリームの格納場所が通知されますが、**FPGA** にビットストリームはダウンロードされません。ビットストリームは、**open** コマンドが発行された後にのみハードウェアにダウンロードされます。

プロジェクトの引数は、ハードウェア協調シミュレーションが記述されている **HWC** ファイルを指定する必要があります。

デストラクタ

構文

```
release(h);
```

説明

ハードウェア協調シミュレーション オブジェクトによって使用されるリソースを解放します。ハードウェアへのリンクが開いたままの場合は、最初にハードウェアを閉じてからリソースが解放されます。

ハードウェアを開く

構文

```
open(h);
```

説明

ホスト PC および **FPGA** 間の接続を開きます。この関数が呼び出される前に、ハードウェア協調シミュレーションのインターフェイスをコンフィギュレーションする必要があります。**xlHwcosimConfig** ユーティリティを使用して、ハードウェア協調シミュレーションのインターフェイスをコンフィギュレーションします。引数 **h** には、ハードウェア協調シミュレーションのオブジェクトを指定します。

ハードウェアを閉じる

構文

```
close(h);
```

説明

ホスト PC および FPGA 間の接続を閉じます。引数 **h** には、ハードウェア協調シミュレーションのオブジェクトを指定します。

データを書き込む

構文

```
h('portName') = inData;  
  
h({inPortNames}) = [inData];  
  
h([inPortIndices]) = [inData];  
  
write(h, 'portName', inData);  
  
write(h, {inPortNames}, [inData]);  
  
write(h, [inPortIndices], [inData]);
```

説明

ポートには、名前またはインデックスを使用してアクセスできます。ポート名およびインデックスは、ハードウェア協調シミュレーション オブジェクトの **Inport** プロパティを取得することにより、ハードウェア協調シミュレーションのインスタンスから抽出できます。ポートが名前により参照される場合、ポート名のセル配列の後ろにポートに対応するデータ配列が続くことが想定されます。同様に、ポートがインデックスにより参照される場合、ポートの配列にデータ配列が続くことが想定されます。

メモ：読み出し/書き込み操作を多数実行する場合は、パフォーマンスが低下するので複数のポートを名前前で指定しないでください。 **get** 命令を使用してポート名のシーケンスを 1 つの同等なインデックス シーケンスにし、このシーケンスを後続の読み出し/書き込み操作に使用することを推奨します。

データを読み出す

構文

```
outData = h('portName');  
  
[outData] = h({outPortNames});  
  
[outData] = h([outPortIndices]);  
  
outData = read(h, 'portName');  
  
[outData] = read(h, {outPortNames});  
  
[outData] = read(h, [outPortIndices]);
```

説明

ポートには、名前またはインデックスを使用してアクセスできます。ポート名およびインデックスは、ハードウェア協調シミュレーション オブジェクトの **Output** プロパティを取得することにより、ハードウェア協調シミュレーションのインスタンスから抽出できます。ポートが名前により参照される場合、ポート名のセル配列の後ろにポートに対応するデータ配列が続くことが想定されま

す。同様に、ポートがインデックスにより参照される場合、ポートの配列にデータ配列が続くことが想定されます。

メモ：読み出し/書き込み操作を多数実行する場合は、パフォーマンスが低下するので複数のポートを名前で指定しないでください。 `get` 命令を使用してポート名のシーケンスを 1 つの同等なインデックス シーケンスにし、このシーケンスを後続の読み出し/書き込み操作に使用することを推奨します。

実行

構文

```
run(h);

run(h, n);
```

説明

ハードウェア協調シミュレーションのオブジェクトがシングル ステップ モードで実行されるようにコンフィギュレーションされている場合、`run` コマンドを使用してクロックを進めます。`run(h)` では、1 サイクルごとにクロックを進め、`run(h,n)` では `n` サイクルごとにクロックを進めます。

ハードウェア協調シミュレーションのオブジェクトがフリーランニング モードでコンフィギュレーションされている場合、`run` コマンドはハードウェア協調シミュレーションのクロックに影響しません。ただし、`JTAG` ハードウェア協調シミュレーションでは、`write` コマンドは効率性の理由でバッファに格納され、`run` コマンドを使用すると `write` バッファをフラッシュできます。

メモ：現段階では、`run` コマンドはフリーランニング モードのイーサネット ハードウェア協調シミュレーションには影響はありませんが、この動作は将来変更される可能性があります。

プロパティの取得

構文

```
get(h);

getrun(h, prop);
```

説明

ハードウェア協調シミュレーションのオブジェクト `h` に関連するプロパティを返します。プロパティは、`MATLAB` 構造体として次のフィールドと共に返されます。

prop	説明
Id	内部使用
Inport	入力ポートすべてを記述する構造体
Outport	出力ポートすべてを記述する構造体
Execution	実行スケジュールを記述する構造体
SharedMemory	オブジェクト内で使用可能な共有メモリを記述する構造体

Exec Id の作成

構文

```
execId = initExec(h, inPorts, outPorts);

getrun(h, prop);
```

説明

exec 命令は、MATLAB 環境で発生したオーバーヘッドを最小限に抑えるために使用します。この命令では、操作のシーケンスを下位のハードウェア協調シミュレーション エンジン 1 つにまとめて起動することで、M コードの解釈で発生するオーバーヘッドを削減し、M コードとエンジン間を切り換えます。これにより、**write**、**read**、および **run** 命令のシーケンスを繰り返し実行するのとは比べてシミュレーション速度が著しく向上します。

実行の定義は、その定義が後続の実行で実行される前に **initExec** を使用して初期化されます。実行の定義では、実行に関わる入力および出力ポートを指定します。実行は、入力および出力ポートのサブセットで定義できます。実行中には関連するポートのみで読み出し/書き込みが実行されますが、その他の入力ポートは同じ値で駆動されることが想定され、その他の出力ポートは無視されます。

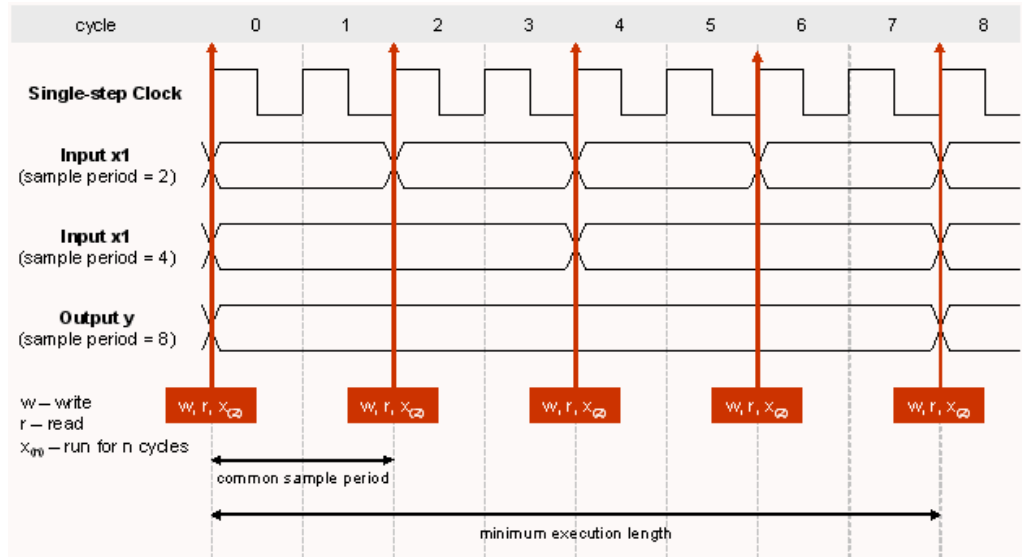
initExec の **inPorts** および **outPorts** 引数には、ポート名のセル配列または配列/ポート インデックスを指定できます。

メモ : **initExec** 命令と **exec** 命令が分けられているのは、パフォーマンス考慮のためのみです。初期化プロセスは、後続の実行の前に 1 回限りオーバーヘッドになるように実行されます。メモリがすべてのシミュレーション サイクルに対する入力データを保持できないときなど、複数の実行にシミュレーションを分割する必要があるような状況で、これは特に重要です。

実行は、サイクル ベースで実行され、入力および出力データがサイクルごとに渡されます。複数レート的设计では、関連するポートの **GCD** レートの周期 (共通のサンプル周期) で内部操作がスケジュールされます。サイクル数は、関連するポートの **LCM** レート (最短実行時間) の倍数にする必要があります。

exec 命令と **read**、**write**、および **run** 命令を混合して使用するときは、最新の注意を払う必要があります。実行の前に、すべての関連する入力ポートおよび出力ポートのサンプルを共通のサンプル周期境界で揃える必要があります。つまり、最初の実行サイクルで関連するポートをサンプルすることになります。この状況で実行が完了すると、関連ポートに対するサンプルのアライメントが保障されます。これは、実行時間が **LCM** レートの倍数であるためです。

次の図に、サンプル周期 2 サイクルと 4 サイクルで動作する 2 個の入力ポートとサンプル周期 8 サイクルで動作する出力ポート 1 個に関連する実行を示します。共通のサンプル周期は $GCD(2, 4, 8) = 2$ サイクルに設定されているので、**write**、**read**、および **run** 操作のシーケンスは、実行の最初のサイクルから開始して 2 サイクルごとに実行されます。実行最短時間は、 $LCM(2, 4, 8) = 8$ サイクルで、実行は 8 サイクルの倍数で実行する必要があります。



ベクトル化実行

構文

```
outData = exec(h, execId, nCycles, inData);
```

説明

exec 命令は、MATLAB 環境で発生したオーバーヘッドを最小限に抑えるために使用します。この命令では、操作のシーケンスを下位のハードウェア協調シミュレーションエンジン 1 つにまとめて起動することで、M コードの解釈で発生するオーバーヘッドを削減し、M コードとエンジン間を切り換えます。これにより、**write**、**read**、および **run** 命令のシーケンスを繰り返し実行するのとは比べてシミュレーション速度が著しく向上します。

execId の引数は、**initExec** への呼び出しで構成されます。**nCycles** では実行するシミュレーションサイクル数が指定され、**inData** には各サイクルでポートを駆動するのに使用されるデータが含まれています。**inData** は、2D マトリックス **[M,N]** で、**M** は **initExec** で指定されている **inPorts** 数、**N** は **nCycles** に相当します。同じ実行サイクルでのすべてのポート データは、同じ列に格納されます。たとえば、**inData** マトリックスの **[M,N]** 要素は、実行で指定された **M** 番目の入力ポートに対する (**N-1**) 目のサイクル データ サンプルに対応します。

M コード/ハードウェア協調シミュレーションの Shared Memory MATLAB クラス

Shmem

Shmem MATLAB クラスでは、ハードウェア協調シミュレーション オブジェクトに組み込まれている共有メモリへのインターフェイスが提供されます。

アクション	構文
コンストラクタ	<code>m = Shmem(memName)</code>
デストラクタ	<code>release(m)</code>
データを書き込む	<code>write(m, addresses, inData)</code> <code>m(addresses) = inData</code>
データを読み出す	<code>outData = read(m, addresses)</code> <code>outData = m(addresses)</code>
プロパティの設定	<code>set(m, prop, data)</code>
プロパティの取得	<code>data = get(m, prop)</code>

コンストラクタ

構文

```
m = Shmem(memName);
```

説明

Shared Memory また Shared Register へのオブジェクト ハンドルを作成します。この引数には、System Generator モデルで定義されている共有メモリの名前を指定します。これはブロック オブジェクトで、特定の名前の共有メモリ 1 個のみを一度に使用できます。

デストラクタ

構文

```
release(m);
```

説明

Shmen オブジェクトにより使用されるリソースを解放します。

データを書き込む

構文

```
write(m, addresses, inData);
```

```
m(addresses) = inData;
```

説明

共有メモリへの書き込み時には、アドレスに整数また書き込み先アドレスを指定した整数列を指定できます。

共有レジスタへの書き込み時には、アドレスを 0 に設定する必要があります。

データを読み出す

構文

```
outData = read(m, addresses);
```

```
outData = m(addresses);
```

説明

共有メモリからの読み出し時には、アドレスに整数また読み出し先アドレスを指定した整数列を指定できます。

共有レジスタからの読み出し時には、アドレスを 0 に設定する必要があります。

プロパティの設定

構文

```
set(m, prop, data);
```

説明

Shmem オブジェクトのプロパティ設定に使用します。

プロパティの取得

構文

```
data=get(m);
```

```
data=get(m, prop);
```


説明

Shmem オブジェクトのプロパティ取得に使用します。

M コード/ハードウェア協調シミュレーションの Shared FIFO MATLAB クラス

Shfifo

Shfifo MATLAB クラスでは、ハードウェア協調シミュレーション オブジェクトに組み込まれている共有 FIFO へのインターフェイスが提供されます。

アクション	構文
コンストラクタ	<code>m = Shfifo(memName)</code>
デストラクタ	<code>release(m)</code>
データを書き込む	<code>write(m, numValues, inData)</code>
データを読み出す	<code>outData = read(m, numValues)</code>
プロパティの設定	<code>set(m, prop, data)</code>
プロパティの取得	<code>data = get(m, prop)</code>

コンストラクタ

構文

```
m = Shfifo(fifoName);
```

説明

Shared FIFO オブジェクトへのオブジェクト ハンドルを作成します。この引数には、System Generator モデルで定義されている共有 FIFO の名前を指定します。これはブロック オブジェクトで、特定の名称の共有メモリ 1 個のみを一度に使用できます。

デストラクタ

構文

```
release(m);
```

説明

Shfifo オブジェクトにより使用されるリソースを解放します。

データを書き込む

構文

```
write(m, numValues, inData);
```

説明

Shared FIFO への書き込み時には、numValues に FIFO に書き込むデータ数を整数で指定し、inData には書き込まれるデータの格納先の配列を指定します。

データを読み出す

構文

```
outData = read(m, numValues);
```

説明

Shared FIFO からの読み出し時には、**numValues** に FIFO から読み出すデータ数を整数で指定し、**outData** には読み出されるデータの格納先の配列を指定します。

プロパティの設定

構文

```
set(m, prop, data);
```

説明

Shfifo オブジェクトのプロパティ設定に使用します。

プロパティの取得

構文

```
data=get(m);
```

```
data=get(m, prop);
```

説明

FIFO の FULL フラグなど、Shfifo オブジェクトのプロパティを入手します。

M コード/ハードウェア協調シミュレーション ユーティリティ関数

xlHwcosim

構文

```
xlHwcosim('release');
```

```
xlHwcosim('releaseMem');
```

```
xlHwcosim('releaseFifo');
```

説明

M コード/ハードウェア協調シミュレーション、Shared Memory または Shared FIFO オブジェクトが作成されるとき、グローバルシステム リソースを使用してこれらのオブジェクトにレジスタが付けられます。これらのオブジェクトは通常そのオブジェクトに対して解放コマンドが呼び出されたときに、解放されます。xlHwcosim では、予期しないエラーが発生したときなど、M コード/ハードウェア協調シミュレーションで使用されるすべてのリソースを簡単に解放できます。xlHwcosim 呼び出しでは特定のタイプのオブジェクトのインスタンスすべてに対するリソースを解放するので、各オブジェクトに対する解放関数は、可能ならば使用する必要があります。

xlHwcosim('release') では、ハードウェア協調シミュレーション オブジェクトのすべてのインスタンスを解放します。

xlHwcosim('releaseMem') では、Shmem オブジェクトのすべてのインスタンスを解放します。

xlHwcosim('releaseFifo') では、Shfifo オブジェクトのすべてのインスタンスを解放します。

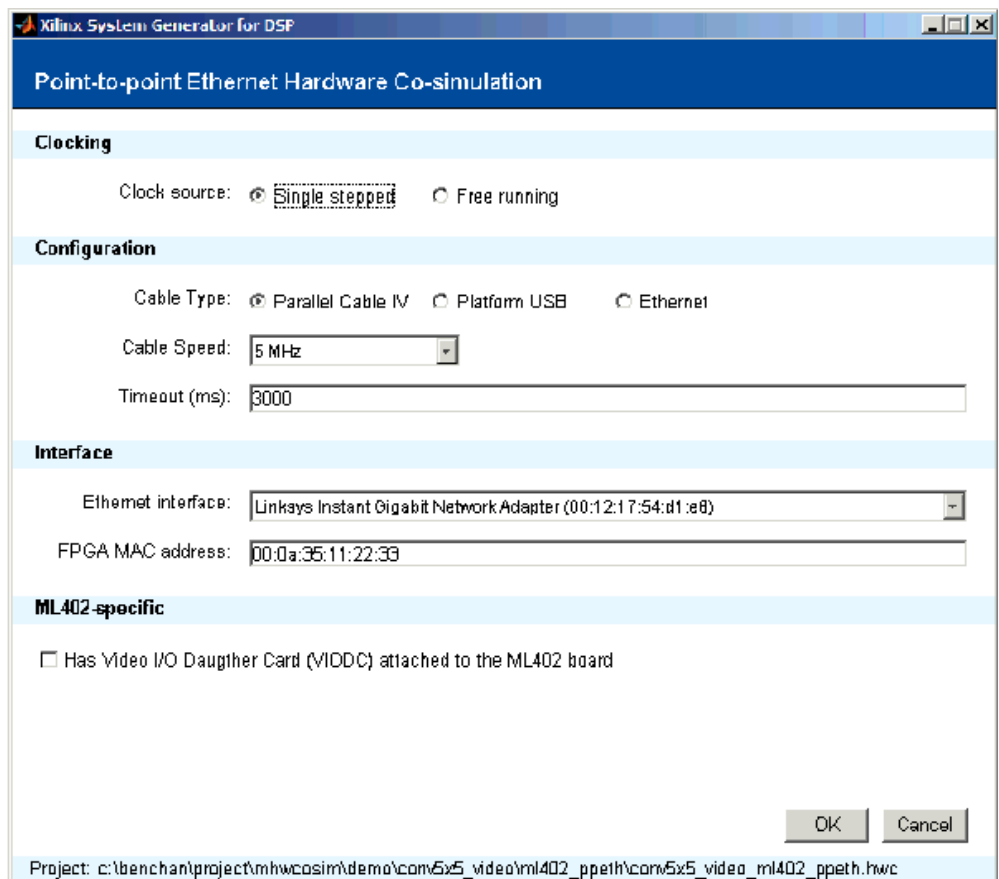
xlHwcosimConfig

構文

```
xlHwcosimGetDesignInfo;  
  
xlHwcosimGetDesignInfo('netlist')  
  
xlHwcosimGetDesignInfo('c:/design/macfir_cw.hwc')
```

説明

xlHwcosimConfig では、次に示すグラフィカル フロントエンドが起動され、ハードウェア協調シミュレーション インターフェイスの設定を設定できます。Simulink で Hardware Co-simulation ブロックをダブルクリックして表示されるブロック GUI と同等です。この起動方法は、xlHwcosimGetDesignInfo と類似しています。



xlHwcosimGetDesignInfo

構文

```
xlHwcosimGetDesignInfo;  
  
xlHwcosimGetDesignInfo('netlist')  
  
xlHwcosimGetDesignInfo('c:/design/macfir_cw.hwc')
```

説明

`xlHwcosimGetDesignInfo` は、HWC ファイルに含まれるデザイン情報を読み出すときに使用します。デフォルトでは、HWC ファイルを入力として処理して、MATLAB 構造体配列にデザイン情報を戻します。HWC ファイルが指定されていない場合は、現在のディレクトリに含まれるプロジェクト ファイルが検索されます。ディレクトリが指定されている場合は、そのディレクトリ内の HWC ファイルが検索されます。

xlHwcosimSimulate

構文

```
outData = xlHwcosimSimulate(project, nCycles, inData)

[o1, o2, ...] = xlHwcosimSimulate(project, nCycles, i1, i2, ...)

outData = xlHwcosimSimulate(project, nCycles, struct('Inport',
    inPorts, 'Outport', outPorts, inData))
```

説明

`xlHwcosimSimulate` では、規定の入力値でデザインをシミュレーションする関数呼び出しを提供します。シミュレーションは、サイクル ベースで実行されます。この関数では、データ値のシーケンスがサイクルごとに各入力ポートに対して 1 つ取り込まれ、結果のシーケンスがサイクルごとに各出力ポートに対して 1 つ返されます。デフォルトでは、すべての入力および出力ポートが使用され、データの値はポート インデックスの昇順でポートにマップされます。

`xlHwcosimSimulate` は、単純なシミュレーションや汎用シミュレーション目的には便利ですが、一部制限があります。

- ユーザー定義のシミュレーション セマンティクスを含まない
- すべてのシミュレーション サイクルが総括的に実行されるため、シミュレーション サイクルでブレイクポイントを設定できない
- 共有メモリアクセスがない

SharedMemory

「[LockableSharedMemory](#)」および「[SharedMemoryProxy](#)」に継承されます。

Public タイプ

- enum creation_tag_dispatch { creation_tag }
- enum owner_type { base, lockable, proxy }

Public メソッド

- SharedMemory (const std::string &name, int nwords, int word_size, creation_tag_dispatch)
- SharedMemory (const std::string &name, unsigned start_address=0, int nwords=INHERIT, int word_size=INHERIT, double timeout_sec=NEVER)
- virtual ~SharedMemory ()
- std::string getName () const
- unsigned getNWords () const
- unsigned getWordSize () const
- owner_type getOwnerType () const
- virtual bool couldBlockOnReadOrWrite () const
- virtual bool read (unsigned addr, StdLogicVector &value, double timeout_sec=NEVER) const
- virtual bool write (unsigned addr, const StdLogicVector &value, double timeout_sec=NEVER)
- virtual bool readArray (unsigned addr, unsigned nwords, StdLogicVectorVector &buffer, double timeout_sec=NEVER) const
- virtual bool writeArray (unsigned addr, unsigned nwords, const StdLogicVectorVector &buffer, double timeout_sec=NEVER)

Static Public 属性

- const int NEVER = -1
- const int INHERIT = -1

Protected タイプ

- enum protected_constructor_tag_dispatch { protected_constructor_tag }

Protected メソッド

- SharedMemory (const std::string &name, int nwords, int word_size, protected_constructor_tag_dispatch)
- SharedMemory ()

Protected 属性

- SharedMemoryImpl * _impl

メンバー列挙

enum creation_tag_dispatch

列挙値：

creation_tag : 既存の物理メモリにアクセスするコンストラクタから、物理メモリを作成するコンストラクタを区別するためにのみ使用されます。

enum owner_type

列挙値：

base : SharedMemory として作成された物理メモリ。

lockable : LockableSharedMemory として作成された物理メモリ。

proxy : SharedMemoryProxy として作成された物理メモリ。

enum protected_constructor_tag_dispatch [protected]

列挙値：

protected_constructor_tag

コンストラクタおよびデストラクタ

SharedMemory (const std::string & name, int nwords, int word_size, creation_tag_dispatch T)

このタグ ディスパッチされたコンストラクタでは、オブジェクトの基礎となる物理メモリ (OS と共有) が作成されます。呼び出す際に、メモリが格納するワード数およびワード当たりのビット数を指定する必要があります。コンストラクタの最終的な引数は、列挙型定数 SharedMemory::creation_tag である必要があります。

パラメータ

name : OS への発行に使用された共有メモリの名前。ほかのスレッドによる作成されたメモリの検出にも使用されます。

nwords : メモリが格納するワード数。

word_size : ワード当たりのビット数。

T : 使用できる値は SharedMemory::creation_tag のみです。このパラメータは、このコンストラクタが要求されたものであり、指定された名前を持つ既存の共有メモリを検出するものではないことをコンパイラに明確に示すために使用されます。

SharedMemory (const std::string & name, unsigned start_address = 0, int nwords = INHERIT, int word_size = INHERIT, double timeout_sec = NEVER)

このコンストラクタでは、既に作成された物理メモリ ストアを使用する SharedMemory インスタンスが作成されます。既存のメモリは指定された名前を使用して OS により検出されますが、指定した名前のメモリが見つからず、タイムアウトが発生した場合は、Sysgen::Error というエラーが発生します。

パラメータ

name : OS への発行に使用された共有メモリの名前。

start_address : 物理メモリのアドレス空間へのオフセット (デフォルトは 0)。

nwords : イメージ化されたメモリのサイズ。物理メモリのサイズよりも小さい値にします。大きい場合 (**start_address** が設定されている場合は物理メモリの終点を越えた場合) は、**Sysgen::Error** というエラーが発生します。デフォルト値は **INHERIT** で、イメージ化されたメモリは物理メモリの終点まで拡張します。

word_size : ワード当たりのビット数。物理メモリに一致する必要があります。一致しない場合は **Sysgen::Error** というエラーが発生します。デフォルト値は **INHERIT** です。

timeout_sec : 物理共有メモリが OS を介して使用可能になるまでコンストラクタが待機する時間を秒単位で指定します。デフォルト値は 15 秒です。**NEVER** にも設定できます。

~SharedMemory () [virtual]

SharedMemory オブジェクトが破棄されると、そのハンドルが OS を介して共有されている物理メモリに解放されます。物理メモリはリファレンスを考慮したリソースで、リソースへのハンドルがすべて解放されるとメモリも解放されます。メモリが 1 つのスレッドで作成されてから 2 番目のスレッドでアクセスされる場合、物理メモリを作成および初期化したオブジェクトが作成スレッドで破棄された後も、2 番目のスレッドはメモリストアにそのままアクセスし続けることができます。

SharedMemory (const std::string & name, int nwords, int word_size, protected_constructor_tag_dispatch T) [protected]

派生クラスで使用する **protected** コンストラクタで、**public** クラス API の一部ではありません。

パラメータ

nwords : メモリが格納するワード数。

word_size : ワード当たりのビット数。

T : 使用できる値は **SharedMemory::protected_constructor_tag** のみです。このパラメータは、このコンストラクタが要求されたものであり、**public** コンストラクタではないことをコンパイラに明確に示すために使用されます。

SharedMemory () [protected]

デフォルトのコンストラクタでは基礎になるインプリメンテーションがない (**_impl** ポインタが **NULL**) **SharedMemory** が作成されます。この場合、このコンストラクタは **private** として宣言され、親のインプリメンテーションを確立する必要がある派生クラスによってのみ使用されます。

メンバー関数

std::string getName () const

戻り値

メモリの作成に使用された名前が返されます。この名前は、ほかの **SharedMemory** インスタンスを同じメモリに割り当てる場合にも使用できます。

unsigned getNWords () const

戻り値

メモリに格納されるワード数が返されます。メモリのインデックスは、アドレス 0 ~ **getNWords()-1** になります。特定の **SharedMemory** インスタンスでは、この値は定数です。

unsigned getWordSize () const

戻り値

メモリ のワード 当たり のビット 数が返されます。特定の SharedMemory インスタンスでは、この値は定数です。

Sysgen::SharedMemory::owner_type getOwnerType () const

戻り値

次の列挙定数値のいずれか 1 つが返されます。

- ◆ SharedMemory::base : 物理メモリがベース SharedMemory コンストラクタを介して作成されている場合
- ◆ SharedMemory::lockable : 物理メモリが派生 LockableSharedMemory コンストラクタを介して作成されている場合
- ◆ SharedMemory::proxy : 物理メモリが派生 SharedMemoryProxy コンストラクタを介して作成されている場合

bool couldBlockOnReadOrWrite () const [virtual]

戻り値

read() または write() への呼び出しがブロック (read/write 呼び出しへの timeout_sec パラメータが NEVER に設定されている場合) またはタイムアウトする可能性のある場合は True が返されます。メモリを作成していない SharedMemory オブジェクトが、反対側にある LockableSharedMemory または SharedMemoryProxy として作成されたメモリを参照する場合があります。いずれの場合も、read/write 呼び出しがブロックする可能性があります。SharedMemory オブジェクトが LockableSharedMemory に接続されている場合は、読み出し操作および書き込み操作で acquireLock および releaseLock セマンティクスが暗示的に強制されます。

LockableSharedMemory および SharedMemoryProxy で再インプリメントされます。

bool read (unsigned addr, StdLogicVector & value, double timeout_sec = NEVER) const [virtual]

パラメータ

addr : 読み出されるアドレス。範囲は [0, getNWords()-1] である必要があります。範囲外の場合は、Sysgen::Error 例外が発生します。

value : メモリ から 読み出された値で内容が上書きされる StdLogicVector への参照。StdLogicVector の型およびサイズを適切にするには、呼び出し側でコンストラクトする必要があります。

timeout_sec : read 操作を試みる時間を秒単位で指定します。

戻り値

読み出しが正しく実行された場合は True が返されます。timeout_sec が NEVER に設定されている場合は Ttrue が返されるか、何も返されません。False が返された場合は、タイムアウトです。

couldBlockOnReadOrWrite()、readArray() も参照してください。

bool write (unsigned addr, const StdLogicVector & value, double timeout_sec = NEVER) [virtual]

パラメータ

addr : 書き込まれるアドレス。範囲は [0, getNWords()-1] である必要があります。範囲外の場合は、Sysgen::Error 例外が発生します。

value : 内容が物理共有メモリにコピーされる StdLogicVector への参照。StdLogicVector のビット数をメモリに一致させるには、呼び出し側でコンストラクトする必要があります。

timeout_sec : write 操作を試みる時間を秒単位で指定します。

戻り値

書き込みが正しく実行された場合は True が返されます。timeout_sec が NEVER に設定されている場合は Ttrue が返されるか、何も返されません。False が返された場合は、タイムアウトです。

couldBlockOnReadOrWrite(), writeArray() も参照してください。

bool readArray (unsigned addr, unsigned nwords, StdLogicVectorVector & buffer, double timeout_sec = NEVER) const [virtual]

パラメータ

addr : 読み出される 1 番目のアドレス。範囲は [0, getNWords()-1] である必要があります。範囲外の場合は、Sysgen::Error 例外が発生します。

nword : 読み出されるワード数。

buffer : メモリから読み出された値で内容が上書きされる StdLogicVectorVector への参照。StdLogicVectorVector の型、ワード数 (nwords 以上)、およびワード当たりのビット数を適切にするには、呼び出し側でコンストラクトする必要があります。

addr+nwords > getNWords() の場合は、Sysgen::Error 例外が発生します。

timeout_sec : readArray 操作を試みる時間を秒単位で指定します。

戻り値

読み出しが正しく実行された場合は True が返されます。timeout_sec が NEVER に設定されている場合は Ttrue が返されるか、何も返されません。False が返された場合は、タイムアウトです。

couldBlockOnReadOrWrite(), read() も参照してください。

bool writeArray (unsigned addr, unsigned nwords, const StdLogicVectorVector & buffer, double timeout_sec = NEVER) [virtual]

パラメータ

addr : 書き込まれる 1 番目のアドレス。範囲は [0, getNWords()-1] である必要があります。範囲外の場合は、Sysgen::Error 例外が発生します。

nword : 書き込まれるワード数。

addr+nwords > getNWords() の場合は、Sysgen::Error 例外が発生します。

buffer : 内容が物理共有メモリに移動される StdLogicVectorVector への参照。StdLogicVectorVector の型、ワード数 (nwords 以上)、およびワード当たりのビット数を適切にするには、呼び出し側でコンストラクトする必要があります。

timeout_sec : writeArray 操作を試みる時間を秒単位で指定します。

戻り値

書き込みが正しく実行された場合は **True** が返されます。**timeout_sec** が **NEVER** に設定されている場合は **Ttrue** が返されるか、何も返されません。**False** が返された場合は、タイムアウトです。

couldBlockOnReadOrWrite()、**write()** も参照してください。

メンバー データ

const int NEVER = -1 [static]

タイムアウトが発生しないように、タイムアウト設定でメソッドをパラメータ化するために使用します。

LockableSharedMemory および **SharedMemoryProxy** で再インプリメントされます。

const int INHERIT = -1 [static]

既に作成されている共有メモリの特性を継承するために使用されます。

LockableSharedMemory で再インプリメントされます。

SharedMemoryImpl* _impl [protected]

LockableSharedMemory

「SharedMemory」の特性を継承します。

Public タイプ

- typedef void(* callback)(LockableSharedMemory &, void *)

Public メソッド

- LockableSharedMemory (const std::string &name, int nwords, int word_size, creation_tag_dispatch)
- LockableSharedMemory (const std::string &name, unsigned start_address=0, int nwords=INHERIT, int word_size=INHERIT, double timeout_sec=15.0)
- virtual ~LockableSharedMemory ()
- virtual bool couldBlockOnReadOrWrite () const
- virtual bool acquireLock (double timeout_sec=NEVER)
- virtual bool acquireLock (callback function, void *arg, double timeout_sec=NEVER)
- virtual bool lockedByMe () const
- virtual void releaseLock ()
- virtual const StdLogicVectorVector & viewAsStdLogicVectorVector () const
- virtual StdLogicVectorVector & viewAsStdLogicVectorVector ()
- const uint32 * getRawDataPtr () const
- uint32 * getRawDataPtr ()

Static Public 属性

- const int NEVER = -1
- const int INHERIT = -1

メンバー Typedefs

- typedef void(* callback)(LockableSharedMemory&, void*)

コンストラクタおよびデストラクタ

LockableSharedMemory (const std::string & name, int nwords, int word_size, creation_tag_dispatch T)

マッチング ベース クラス (SharedMemory) コンストラクタに似ていますが、ロック (mutex) セマンティクス付きの共有メモリが作成されます。LockableSharedMemory クラスは、SharedMemory クラスを acquireLock() および releaseLock() メソッドを使用して拡張します。

LockableSharedMemory (const std::string & name, unsigned start_address = 0, int nwords = INHERIT, int word_size = INHERIT, double timeout_sec = 15.0)

マッピング ベース クラス (SharedMemory) コンストラクタに似ていますが、ロック (mutex) セマンティクス付きの共有メモリが作成されます。LockableSharedMemory クラスは、SharedMemory クラスを acquireLock() および releaseLock() メソッドを使用して拡張します。

~LockableSharedMemory () [virtual]

使用法はベース クラス (SharedMemory) デストラクタとほぼ同じですが、LockableSharedMemory デストラクタではロックが保持されていれば解放されます。

メンバー関数

virtual bool couldBlockOnReadOrWrite () const [inline, virtual]

戻り値

read() または write() への呼び出しがブロック (read/write 呼び出しへの timeout_sec パラメータが NEVER に設定されている場合) またはタイムアウトする可能性のある場合は True が返されます。メモリを作成していない SharedMemory オブジェクトが、反対側にある LockableSharedMemory または SharedMemoryProxy として作成されたメモリを参照する場合があります。いずれの場合も、read/write 呼び出しがブロックする可能性があります。SharedMemory オブジェクトが LockableSharedMemory に接続されている場合は、読み出し操作および書き込み操作で acquireLock および releaseLock セマンティクスが暗示的に強制されます。

SharedMemory から再インプリメントされます。

bool acquireLock (double timeout_sec = NEVER) [virtual]

ロックの取得を試みます。

パラメータ

timeout_sec : acquireLock 操作を試みる時間を秒単位で指定します。

返される内容

timeout_sec 秒以内にロックが取得できた場合は True、それ以外は False が返されます。timeout_sec が NEVER に設定されている場合は True が返されるか、何も返されません。

bool acquireLock (callback function, void * arg, double timeout_sec = NEVER) [virtual]

ロックの取得を試み、取得された場合は、処理中のほかのユーザーがロックを解放するために使用できるコールバック関数を設定します。ユーザー アプリケーションでは、通常はこのメソッドを使用しないでください。このメソッドは、複数のメモリ クライアントが 1 つのスレッドにあり、このメソッドを使用しなければデッドロックになる可能性がある、System Generator の内部アプリケーションで使用します。

パラメータ

function : ロックが必要なほかの共有メモリ クライアントによって起動されるコールバック関数。

arg : コールバック関数に渡される void* 引数。

`timeout_sec` : `acquireLock` 操作を試みる時間を秒単位で指定します。

戻り値

`timeout_sec` 秒以内にロックが取得できた場合は `True`、それ以外は `False` が返されます。
`timeout_sec` が `NEVER` に設定されている場合は `True` が返されるか、何も返されません。

`bool lockedByMe () const [virtual]`

戻り値

呼び出しているインスタンスがロックを保持している場合は `True` が返されます。

`void releaseLock () [virtual]`

呼び出しているインスタンスがロックを保持している場合は解放します。ロックを保持していない場合は、呼び出しは `no-op` となります。

`const Sysgen::StdLogicVectorVector & viewAsStdLogicVectorVector () const [virtual]`

戻り値

内部データストアが物理共有メモリに対応付けられている `const StdLogicVectorVector` 参照が返されます。このメソッドは、高性能のアプリケーションでのみ使用してください。アクセスは高速ですが、チェックが実行されないため安全性が低くなります。

`Sysgen::StdLogicVectorVector & viewAsStdLogicVectorVector () [virtual]`

戻り値

内部データストアが物理共有メモリに対応付けられている `StdLogicVectorVector` 参照が返されます。このメソッドは、高性能のアプリケーションでのみ使用してください。アクセスは高速ですが、チェックが実行されないため安全性が低くなります。

`const Sysgen::uint32 * getRawDataPtr () const`

戻り値

物理共有メモリの内部データストアへの `const raw` データポインタが返されます。このメソッドは、高性能のアプリケーションでのみ使用してください。アクセスは高速ですが、チェックが実行されないため安全性が低くなります。

`Sysgen::uint32 * getRawDataPtr ()`

戻り値

物理共有メモリの内部データストアへの `raw` データポインタが返されます。このメソッドは、高性能のアプリケーションでのみ使用してください。アクセスは高速ですが、チェックが実行されないため安全性が低くなります。

メンバー データ

`const int NEVER = -1 [static]`

タイムアウトが発生しないように、タイムアウト設定でメソッドをパラメータ化するために使用します。

SharedMemory から再インプリメントされます。

const int INHERIT = -1 [static]

既に作成されている共有メモリの特性を継承するために使用されます。

SharedMemory から再インプリメントされます。

SharedMemoryProxy

「SharedMemory」の特性を継承します。

Public タイプ

- typedef void(* requestServicer)(const Request &, SharedMemoryProxy &, void *arg)

Public メソッド

- SharedMemoryProxy (const std::string &name, int nwords, int word_size, requestServicer rs, void *rs_arg=NULL)
- ~SharedMemoryProxy ()
- virtual bool couldBlockOnReadOrWrite () const
- void service ()
- virtual const StdLogicVectorVector & viewAsStdLogicVectorVector () const
- virtual StdLogicVectorVector & viewAsStdLogicVectorVector ()
- const uint32 * getRawDataPtr () const
- uint32 * getRawDataPtr ()

Static Public 属性

- const int NEVER = -1

メンバー Typedefs

```
typedef void(* requestServicer)(const Request&, SharedMemoryProxy&, void *arg)
```

この typedef で宣言されたタイプの関数ポインタは、SharedMemoryProxy コンストラクタのコンストラクタに渡されます。詳細は、コンストラクタのマニュアルを参照してください。

コンストラクタおよびデストラクタ

SharedMemoryProxy (const std::string & name, int nwords, int word_size, requestServicer rs, void * rs_arg = NULL)

このコンストラクタでは、オブジェクトの基礎となる物理メモリ (OS と共有) が作成されます。呼び出す際に、メモリが格納するワード数およびワード当たりのビット数を指定する必要があります。SharedMemoryProxy で作成される物理メモリには、ほかのクライアントでサービス (読み出し/書き込み) 要求はできますが、格納されたデータに直接アクセスできません。クライアントは、ベース クラス SharedMemory オブジェクトを介してメモリにアクセスします。サービス要求は、物理メモリを作成した SharedMemoryProxy オブジェクトに渡されます。この構造により、SharedMemoryProxy でリモート記憶域にデータが格納されます (実際に保管されたデータがハードウェア プラットフォームまたはリモート マシンに格納される場合など)。

関数ポインタは、ほかのメモリ クライアント による `SharedMemoryProxy` への要求を処理する関数を指定する必要があります。これらの要求は、渡された `SharedMemoryProxy::Request` オブジェクトによりエンコードされた読み出し 要求または書き込み要求の形を取ることができます。 またこのオブジェクト には、読み出し/書き込みのワード 数および開始アドレスも 含まれます。`SharedMemoryProxy` への参照も、コンストラクタで使用するポイド ポインタと共にサービス提供関数に渡されます。

パラメータ

name : OS への発行に使用された共有メモリの名前。ほかのスレッドによる作成されたメモリの検出にも使用されます。

nwords : メモリが格納するワード数。

word_size : ワード当たりのビット数。

rs : メモリ要求を提供するコールバック関数。

rs_arg : requestServicer コールバックを渡す void* 引数。

~SharedMemoryProxy ()

サービス要求を発行した共有 OS リソースにインスタンスのハンドルを解放します。これは参照を考慮したリソースで、リソースを確立した `SharedMemoryProxy` が破棄された後も維持されることがあります。これは、`SharedMemoryProxy` デストラクタが呼び出された場合は、メモリの残りのクライアントで問題が発生することを意味します。特に読み出し/書き込み呼び出しは処理されず、停止するかタイムアウトになります。

メンバー関数

virtual bool couldBlockOnReadOrWrite () const [inline, virtual]

戻り値

`read()` または `write()` への呼び出しがブロック (`read/write` 呼び出しへの `timeout_sec` パラメータが `NEVER` に設定されている場合) またはタイムアウトする可能性のある場合は `True` が返されます。メモリを作成していない `SharedMemory` オブジェクトが、反対側にある `LockableSharedMemory` または `SharedMemoryProxy` として作成されたメモリを参照する場合があります。いずれの場合も、`read/write` 呼び出しがブロックする可能性があります。`SharedMemory` オブジェクトが `LockableSharedMemory` にインターフェイスする場合は、読み出し操作および書き込み操作で `acquireLock` および `releaseLock` セマンティクスが暗示的に強制されます。

`SharedMemory` から再インプリメントされます。

void service ()

クライアントでサービス要求が発行されたかどうかを確認し、発行されている場合は、`requestServicer callback` (`SharedMemoryProxy` コンストラクタにより確立) が呼び出されます。

const Sysgen::StdLogicVectorVector & viewAsStdLogicVectorVector () const [virtual]

戻り値

内部データストアが物理共有メモリに対応付けられている `const StdLogicVectorVector` 参照が返されます。このメソッドは、高性能のアプリケーションでのみ使用してください。アクセスは高速ですが、チェックが実行されないため安全性が低くなります。

Sysgen::StdLogicVectorVector & viewAsStdLogicVectorVector () [virtual]

戻り値

内部データ ストアが物理共有メモリに対応付けられている **StdLogicVectorVector** 参照が返されます。このメソッドは、高性能のアプリケーションでのみ使用してください。アクセスは高速ですが、チェックが実行されないため安全性が低くなります。

const Sysgen::uint32 * getRawDataPtr () const

戻り値

物理共有メモリの内部データ ストアへの **const raw** データ ポインタが返されます。このメソッドは、高性能のアプリケーションでのみ使用してください。アクセスは高速ですが、チェックが実行されないため安全性が低くなります。

Sysgen::uint32 * getRawDataPtr ()

戻り値

物理共有メモリの内部データ ストアへの **raw** データ ポインタが返されます。このメソッドは、高性能のアプリケーションでのみ使用してください。アクセスは高速ですが、チェックが実行されないため安全性が低くなります。

メンバー データ

const int NEVER = -1 [static]

タイムアウトが発生しないように、タイムアウト設定でメソッドをパラメータ化するために使用します。

SharedMemory から再インプリメントされます。

Request Struct

Public タイプ

- enum Type { read_request, write_request }

Static Public 属性

- enum Sysgen::SharedMemoryProxy::Request::Type type
- unsigned start_address
- unsigned nwords

SharedMemoryProxy::requestServicer コールバック関数に渡された情報のエンコードに使用します。詳細は、SharedMemoryProxy コンストラクタのマニュアルを参照してください。

メンバー列挙

enum Type

列挙値：

read_request : クライアントでメモリを読み出します。

write_request : クライアントでメモリに書き込みます。

メンバー データ

enum Sysgen::SharedMemoryProxy::Request::Type : type

unsigned start_address

読み出し/書き込みを開始するアドレス。

unsigned nwords

読み出し/書き込みのワード数。

NamedPipeReader

Public メソッド

- NamedPipeReader (const std::string &name, int nwords=INHERIT, int word_size=INHERIT, double timeout_sec=15.0)
- ~NamedPipeReader ()
- void peek (StdLogicVector &value) const
- bool read (StdLogicVector &value, double timeout_sec=NEVER)
- bool readArray (unsigned nwords, StdLogicVectorVector &buffer, double timeout_sec=NEVER)
- unsigned getNWords () const
- unsigned getWordSize () const
- bool isEmpty () const
- unsigned numAvailable () const

Static Public 属性

- const int NEVER = -1
- const int INHERIT = -1

コンストラクタおよびデストラクタ

NamedPipeReader (const std::string & name, int nwords = INHERIT, int word_size = INHERIT, double timeout_sec = 15.0)

NamedPipeWriter で既に作成されている名前付きパイプからデータを読み出す NamedPipeReader インスタンスを作成します。名前付きパイプは指定された名前を使用して OS より検出されますが、指定した名前付きパイプが見つからず、タイムアウトが発生した場合は、Sysgen::Error というエラーが発生します。

パラメータ

name : NamedPipeWriter でパイプ作成時に使用された名前。

nwords : NamedPipeWriter で作成されたパイプのワード数よりも小さい値にします。大きい場合は、Sysgen::Error というエラーが発生します。デフォルト値は INHERIT です。

word_size : ワード当たりのビット数。NamedPipeWriter で指定されたワード数に一致する必要があります。一致しない場合は Sysgen::Error というエラーが発生します。デフォルト値は INHERIT です。

timeout_sec : 名前付きパイプが OS を介して使用可能になるまでコンストラクタが待機する時間を秒単位で指定します。デフォルト値は 15 秒です。NEVER にも設定できます。

~NamedPipeReader ()

メンバー関数

void peek (StdLogicVector & value) const

パイプの末端の値を取得します。取得される値は `read()` の場合と同じですが、パイプの状態は変更されず、`peek()` で確認したワードがパイプから削除されることはありません。`peek()` では、`read()` と異なり、パイプの状態が変更されないため、暗示的な `mutex` の要件がなく、パイプが空でない限り、処理は必ず正しく実行されます。

パイプが空の場合は、`Sysgen::Error` 例外が発生します。

パラメータ

value : パイプの値で内容が上書きされる `StdLogicVector` への参照。`StdLogicVector` の型およびサイズを適切にするには、呼び出し側でコンストラクトする必要があります。

`read()` も参照してください。

bool read (StdLogicVector & value, double timeout_sec = NEVER)

パイプが空の場合は、`Sysgen::Error` 例外が発生します。

`peek()`、`readArray()` も参照してください。

パラメータ

value : パイプから読み出された値で内容が上書きされる `StdLogicVector` への参照。`StdLogicVector` の型およびサイズを適切にするには、呼び出し側でコンストラクトする必要があります。

timeout_sec : `read` 操作を試みる時間を秒単位で指定します。特定のパイプへのアクセスでは、`NamedPipeWriter` と `NamedPipeReader` 間に暗示的な `mutex` があります。

戻り値

読み出しが正しく実行された場合は `True`。`timeout_sec` が `NEVER` に設定されている場合は `True` が返されるか、何も返されません。`False` が返された場合は、タイムアウトです。

bool readArray (unsigned nwords, StdLogicVectorVector & buffer, double timeout_sec = NEVER)

`nwords` で指定したワード数がパイプに含まれていない場合は、`Sysgen::Error` 例外が発生します。呼び出しでは、`nwords < numAvailable()` であることを確認する必要があります。

パラメータ

nword : 書き込まれるワード数。

buffer : 内容がパイプにコピーされる `StdLogicVectorVector` への参照。`StdLogicVectorVector` の型、ワード数 (`nwords` 以上)、およびワード当たりのビット数を適切にするには、呼び出し側でコンストラクトする必要があります。

timeout_sec : `read` 操作を試みる時間を秒単位で指定します。特定のパイプへのアクセスでは、`NamedPipeWriter` と `NamedPipeReader` 間に暗示的な `mutex` があります。

戻り値

読み出しが正しく実行された場合は `True` が返されます。`timeout_sec` が `NEVER` に設定されている場合は `True` が返されるか、何も返されません。`False` が返された場合は、タイムアウトです。

`read()` も参照してください。

unsigned getNWords () const

戻り値

パイプで保持できるワード数 (現在保持されていない分のみ) が返されます。

`numAvailable()` も参照してください。

unsigned getWordSize () const

戻り値

パイプで伝送されるデータのワード当たりのビット数が返されます。特定の `NamedPipe` インスタンスでは、この値は定数です。

bool isEmpty () const [inline]

unsigned numAvailable () const

戻り値

パイプにあり、読み出し可能なワード数が返されます。

`getNWords()` も参照してください。

メンバー データ

const int NEVER = -1 [static]

const int INHERIT = -1 [static]

NamedPipeWriter

Public メソッド

- NamedPipeWriter (const std::string &name, int nwords, int word_size)
- ~NamedPipeWriter ()
- bool write (const StdLogicVector &value, double timeout_sec=NEVER)
- bool writeArray (unsigned nwords, const StdLogicVectorVector &buffer, double timeout_sec=NEVER)
- unsigned getNWords () const
- unsigned getWordSize () const
- bool isFull () const
- unsigned numAvailable () const

Static Public 属性

- const int NEVER = -1

コンストラクタおよびデストラクタ

NamedPipeWriter (const std::string & name, int nwords, int word_size)

このコンストラクタでは、名前付きパイプ オブジェクトの基礎となる物理メモリ (OS と共有) が作成されます。呼び出す際に、パイプが保持できるワード数およびワード当たりのビット数を指定する必要があります。

名前付きパイプには、ライタは 1 つしかありません。リーダは複数持つことができ、使用および破棄を繰り返すことができます。

パラメータ

name : OS への発行に使用された共有された名前付きパイプの名前。ほかのスレッドによる、このパイプの検出にも使用されます。

nwords : パイプが保持するワード数。

word_size : ワード当たりのビット数。

~NamedPipeWriter ()

名前付きパイプを表す共有された OS リソースへのインスタンスのハンドルを解放します。これはリファレンスを考慮したリソースで、リソースを確立した **NamedPipeWriter** が破棄された後も維持されることがあります。同じリソースを使用している **NamedPipeReader** インスタンスでは継続してリソースにアクセスし、データを読み出すことができます。リソースが空になるまでは、パイプに新しいデータを追加できません。

メンバー関数

bool write (const StdLogicVector & value, double timeout_sec = NEVER)

パイプがフルの場合は、Sysgen::Error 例外が発生します。

パラメータ

value : 内容が名前付きパイプ ストレージにコピーされる StdLogicVector への参照。
StdLogicVector のビット数を適切にパイプに一致させるには、呼び出し側でコンストラクトする必要があります。

timeout_sec : write 操作を試みる時間を秒単位で指定します。特定のパイプへのアクセスでは、NamedPipeWriter と NamedPipeReader 間に暗示的な mutex があります。

戻り値

書き込みが正しく実行された場合は True が返されます。timeout_sec が NEVER に設定されている場合は Ttrue が返されるか、何も返されません。False が返された場合は、タイムアウトです。

writeArray() も参照してください。

bool writeArray (unsigned nwords, const StdLogicVectorVector & buffer, double timeout_sec = NEVER)

パイプに十分な nwords の領域がない場合は、Sysgen::Error 例外が発生します。

パラメータ

nword : 書き込まれるワード数。

buffer : 内容が名前付きパイプに移動される StdLogicVectorVector への参照。
StdLogicVectorVector の型、ワード数 (nwords 以上)、およびワード当たりのビット数を適切にするには、呼び出し側でコンストラクトする必要があります。

timeout_sec : write 操作を試みる時間を秒単位で指定します。特定のパイプへのアクセスでは、NamedPipeWriter と NamedPipeReader 間に暗示的な mutex があります。

戻り値

書き込みが正しく実行された場合は True が返されます。timeout_sec が NEVER に設定されている場合は Ttrue が返されるか、何も返されません。False が返された場合は、タイムアウトです。

write() も参照してください。

unsigned getNWords () const

戻り値

パイプで保持できるワード数 (現在保持されていない分) が返されます。

numAvailable() も参照してください。

unsigned getWordSize () const

戻り値

パイプで伝送されるデータのワード当たりのビット数が返されます。特定の NamedPipe インスタンスでは、この値は定数です。

```
bool isFull () const [inline]
```

```
unsigned numAvailable () const
```

メンバー データ

```
const int NEVER = -1 [static]
```


索引

数字

2 Channel Decimate by 2 MAC FIR Filter リファレンス デザイン 377
2n+1-tap Linear Phase MAC FIR Filter リファレンス デザイン 378
2n-tap Linear Phase MAC FIR Filter リファレンス デザイン 379
2n-tap MAC FIR Filter リファレンス デザイン 380
4-channel 8-tap Transpose FIR Filter リファレンス デザイン 381
4n-tap MAC FIR Filter リファレンス デザイン 382
5x5Filter リファレンス デザイン 383

A

Accumulator ブロック 50
Addressable Shift Register ブロック 52
AddSub ブロック 54
Assert ブロック 56

B

Basic Elements ブロック 22
BitBasher ブロック 58
Black Box ブロック 61
BPSK AWGN Channel リファレンス デザイン 385

C

C++ アクセス 483
ChipScope Pro Analyzer
 MATLAB ワークスペースへのデータのインポート 71
 既知の問題 72
 ハードウェアとソフトウェア条件 69
 プロジェクト ファイル 71
ChipScope ブロック 69
CIC Compiler 1.1 ブロック 73
CIC Compiler 1.3 ブロック 75
CIC Filter リファレンス デザイン 386
Clock Enable Probe ブロック 77
Clock Probe ブロック 79
CMult ブロック 80

Communication ブロック 25
Complex Multiplier 3.0 ブロック 82
Complex Multiplier 3.1 ブロック 84
Concat ブロック 86
Configurable Subsystem Manager ブロック 87
Constant ブロック 89
Control Logic ブロック 26
Convert ブロック 92
Convolution Encoder 7.0 ブロック 96
Convolutional Encoder 6.1 ブロック 94
Convolutional Encoder リファレンス デザイン 388
CORDIC 4.0 ブロック 98
CORDIC ATAN リファレンス デザイン 390
CORDIC DIVIDER リファレンス デザイン 391
CORDIC LOG リファレンス デザイン 392
CORDIC SINCOS リファレンス デザイン 394
CORDIC SQRT リファレンス デザイン 395
Counter ブロック 101

D

DAFIR v9_0 ブロック 104
Data Types ブロック 28
DDS Compiler 3.1 ブロック 112
DDS Compiler 4.0 ブロック 116
DDS Compiler v2_0 ブロック 108
Delay ブロック 122
Depuncture ブロック 127
Disregard Subsystem ブロック 129
Divider Generator 2.0 ブロック 130
Divider Generator 3.0 ブロック 132
Down Sample ブロック 134
DSP ブロック 29
DSP48 Macro 2.0 ブロック 149
DSP48 Macro ブロック 140
DSP48 ブロック 137
DSP48A ブロック 154
DSP48E ブロック 157
Dual Port Memory Interpolation MAC FIR Filter リファレンス デザイン 397

Dual Port RAM ブロック 162

E

EDK Processor ブロック 168
Expression ブロック 172

F

Fast Fourier Transform 6.0 ブロック 173
Fast Fourier Transform 7.0 ブロック 179
FDATool ブロック 185
FIFO ブロック 187
FIR Compiler 4.0 ブロック 188
FIR Compiler 5.0 ブロック 196
From FIFO ブロック 203
From Register ブロック 205

G

Gateway In ブロック 207
Gateway Out ブロック 209

I

Indeterminate Probe ブロック 211
Index ブロック 31
Interleaver Deinterleaver 5.0 ブロック 212
Interleaver Deinterleaver 5.1 ブロック 215
Interpolation Filter リファレンス デザイン 398
Inverter ブロック 218

J

JTAG Co-Simulation ブロック 219

L

LFSR ブロック 221
Lockable SharedMemory クラス 511
Logical ブロック 223

M

M コード

ハードウェア協調シミュレーションへのアクセス 483

ハードウェアへのインターフェイス 485

M コード/ハードウェア協調シミュレーション

MATLAB クラス 493

Shared FIFO MATLAB クラス 501

Shared Memory MATLAB クラス 499

シミュレーションセマンティクス 484

データ表現 484

テストベンチの自動生成 490

ハードウェアのコンパイル 484

ユーティリティ関数 502

例 485

Math ブロック 39

MATLAB クラス

Shfifo 501

Shmem 499

ハードウェア協調シミュレーション 493

m-channel n-tap Transpose FIR Filter
リファレンス デザイン 399

MCode ブロック 224

Mealy State Machine リファレンス
デザイン 400

Memory ブロック 41

MicroBlaze Processor ブロック 245

ModelSim ブロック 254

Moore State Machine リファレンス
デザイン 403

Mult ブロック 259

Multipath Fading Channel Model
リファレンス デザイン 406

Multiple Subsystem Generator
ブロック 261

Mux ブロック 266

N

NamedPipeReader 519

NamedPipeWriter 522

Negate ブロック 267

Network Ethernet Co-simulation
ブロック 268

n-tap Dual Port Memory MAC FIR
Filter リファレンス デザイン 413

n-tap MAC FIR Filter リファレンス
デザイン 414

O

Opmode ブロック 270

P

Parallel to Serial ブロック 273

Pause Simulation ブロック 274

PG API 469

xBlock 470

xBlockHelp 474

xInput 471

xlsub2script 472

xOutput 471

xSignal 472

エラーおよび警告メッセージ 481

はじめに 469

PG API の例

Hello World 475

MACC 476

マスクされたサブシステム内の
MACC 477

PicoBlaze Instruction Display ブロック
275

PicoBlaze Microcontroller ブロック
276

Point-to-Point Ethernet Co-Simulation
ブロック 278

Puncture ブロック 280

R

Reed-Solomon Decoder 6.1 ブロック
282

Reed-Solomon Decoder 7.0 ブロック
286

Reed-Solomon Encoder 6.1 ブロック
291

Reed-Solomon Encoder 7.0 ブロック
295

Register ブロック 300

Registered Mealy State Machine
リファレンス デザイン 415

Registered Moore State Machine
リファレンス デザイン 418

Reinterpret ブロック 301

Relational ブロック 302

Request Struct 518

Reset Generator ブロック 303

Resource Estimator ブロック 304

ROM ブロック 308

S

Sample Time ブロック 311

Scale ブロック 312

Serial to Parallel ブロック 313

Shared Memory Read ブロック 318

Shared Memory Write ブロック 320

Shared Memory ブロック 42, 314, 483
C++ アクセス 483

SharedMemory クラス 505

SharedMemory.h 483

SharedMemoryProxy クラス 515

Shift ブロック 322

Simulation Multiplexer ブロック 323

Single Port RAM ブロック 325

Single-Step Simulation ブロック 331

Slice ブロック 332

Sysgen Generator

NamedPipeReader クラス 519

NamedPipeWriter クラス 522

Sysgen Namespace

Lockable SharedMemory クラス
511

SharedMemory クラス 505

sysgen.dll 483

System Generator 名前空間

Request Struct 518

SharedMemoryProxy クラス 515

System Generator ブロック 333

System Generator ユーティリティ

xlAddTerms 436

xlCache 440

xlconfiguresolver 442

xlfa_denominator 443

xlfa_numerator 444

xlGenerateButton 445

xlgetparam 446, 448

xlGetReloadOrder 450

xlInstallPlugin 452

xlLoadChipScopeData 453

xlSDBBuilder 454

xlSetNonMemMap 457

xlsetparam 446

xlSetUseHDL 458

xlSwitchLibrary 459

xlTBUtils 460

xlTimingAnalysis 464

xlUpdateModel 465
xlVersion 468

T

Threshold ブロック 339
Time Division Demultiplexer ブロック 340
Time Division Multiplexer ブロック 342
To FIFO ブロック 343
To Register ブロック 345
Toolbar ブロック 347
Tools ブロック 43

U

Up Sample ブロック 350

V

Virtex Line Buffer (画像) リファレンス デザイン 421
Virtex2 5 Line Buffer (画像) リファレンス デザイン 423
Virtex2 Line Buffer (画像) リファレンス デザイン 422
Viterbi Decoder 7.0 ブロック 355
Viterbi Decoder v6_0 ブロック 352

W

WaveScope ブロック 360
White Gaussian Noise Generator (通信) リファレンス デザイン 424

X

xBlock 470
xInput 471
xlAddTerms 436
xlBlockHelp 474
xlCache 440
xlconfiguresolver 442
xlfa_denominator 443
xlfa_numerator 444
xlGenerateButton 445
xlgetparam 446, 448
xlGetReloadOrder 450
xlInstallPlugin 452
xlLoadChipScopeData 453

xlSBDBuilder 454
xlSetNonMemMap 457
xlsetparam 446
xlSetUseHDL 458
xlsub2script 472
xlSwitchLibrary 459
xlTBUtills 460
xlTimingAnalysis 464
xlUpdateModel 465
xlVersion 468
xOutput 471
xSignal 472
XtremeDSP ADC ブロック 428
XtremeDSP Co-simulation ブロック 429
XtremeDSP DAC ブロック 431
XtremeDSP External RAM ブロック 432
XtremeDSP LED Flasher ブロック 433

き

共通オプション
 ブロック パラメータ 45
共有メモリの結合
 From FIFO ブロック 203
 From Register ブロック 205
 Shared Memory ブロック 314
 To FIFO ブロック 343
 To Register ブロック 345

こ

コンパイル
 M コード/ハードウェア協調シミュレーション 484

さ

ザイリンクス ブロック ライブラリ
 Basic Elements ブロック 22
 Communication ブロック 25
 Control Logic ブロック 26
 Data Types ブロック 28
 DSP ブロック 29
 Index ブロック 31
 Math ブロック 39
 Memory ブロック 41
 Shared Memory ブロック 42
 Tools ブロック 43

ザイリンクス ブロック セット
 Accumulator 50
 Addressable Shift Register 52
 AddSub 54
 Assert 56
 BitBasher 58
 Black Box 61
 ChipScope 69
 CIC Compiler 1.1 73
 CIC Compiler 1.3 75
 Clock Enable Probe 77
 Clock Probe 79
 CMult 80
 Complex Multiplier 3.0 82
 Complex Multiplier 3.1 84
 Concat 86
 Configurable Subsystem Manager 87
 Constant 89
 Convert 92
 Convolution Encoder 7.0 96
 Convolutional Encoder 6.1 94
 CORDIC 4.0 98
 Counter 101
 DAFIR v9_0 104
 DDS Compiler 3.1 112
 DDS Compiler 4.0 116
 DDS Compiler v2_0 108
 Delay 122
 Depuncture 127
 Disregard Subsystem 129
 Divider Generator 2.0 130
 Divider Generator 3.0 132
 Down Sample 134
 DSP48 137
 DSP48 Macro 140
 DSP48 Macro 2.0 149
 DSP48A 154
 DSP48E 157
 Dual Port RAM 162
 EDK Processor 168
 Expression 172
 Fast Fourier Transform 6.0 173
 Fast Fourier Transform 7.0 179
 FDATool 185
 FIFO 187
 FIR Compiler 4.0 188
 FIR Compiler 5.0 196
 From FIFO 203
 From Register 205

Gateway In 207
 Gateway Out 209
 Indeterminate Probe 211
 Interleaver Deinterleaver 5.1 215
 Interleaver Deinterleaver v5_0 212
 Inverter 218
 JTAG Co-Simulation 219
 LFSR 221
 Logical 223
 MCode 224
 MicroBlaze Processor 245
 ModelSim 254
 Mult 259
 Multiple Subsystem Generator 261
 Mux 266
 Negate 267
 Network Ethernet Co-simulation 268
 Opmode 270
 Parallel to Serial 273
 Pause Simulation 274
 PicoBlaze Instruction Display 275
 PicoBlaze Microcontroller 276
 Point-to-Point Ethernet Co-Simulation 278
 Puncture 280
 Reed-Solomon Decoder 6.1 282
 Reed-Solomon Decoder 7.0 286
 Reed-Solomon Encoder 6.1 291
 Reed-Solomon Encoder 7.0 295
 Register 300
 Reinterpret 301
 Relational 302
 Reset Generator 303
 Resource Estimator 304
 ROM 308
 Sample Time 311
 Scale 312
 Serial to Parallel 313
 Shared Memory 314
 Shared Memory Read 318
 Shared Memory Write 320
 Shift 322
 Simulation Multiplexer 323
 Single Port RAM 325
 Single-Step Simulation 331
 Slice 332
 System Generator 333
 Threshold 339

Time Division Demultiplexer 340
 Time Division Multiplexer 342
 To FIFO 343
 To Register 345
 Toolbar 347
 Up Sample 350
 Viterbi Decoder 7.0 355
 Viterbi Decoder v6_0 352
 WaveScope 360
 XtremeDSP ADC 428
 XtremeDSP Co-simulation 429
 XtremeDSP DAC 431
 XtremeDSP External RAM 432
 XtremeDSP LED Flasher 433
 ザイリンクス ブロックセット ライブラリ
 ブロックの構造 22
 ザイリンクス リファレンス デザイン ライブラリ
 2 Channel Decimate by 2 MAC FIR Filter 377
 2n+1-tap Linear Phase MAC FIR Filter 378
 2n-tap Linear Phase MAC FIR Filter 379
 2n-tap MAC FIR Filter 380
 4-channel 8-tap Transpose FIR Filter 381
 4n-tap MAC FIR Filter 382
 5x5Filter 383
 BPSK AWGN Channel 385
 CIC Filter 386
 Convolutional Encoder 388
 CORDIC ATAN 390
 CORDIC DIVIDER 391
 CORDIC LOG 392
 CORDIC SINCOS 394
 CORDIC SQRT 395
 DSP デザイン 375
 Dual Port Memory Interpolation MAC FIR Filter 397
 Interpolation Filter 398
 m-channel n-tap Transpose FIR Filter 399
 Mealy State Machine 400
 Moore State Machine 403
 Multipath Fading Channel Model 406
 n-tap Dual Port Memory MAC FIR Filter 413
 n-tap MAC FIR Filter 414

Registered Mealy State Machine 415
 Registered Moore State Machine 418
 Virtex Line Buffer (画像) 421
 Virtex2 5 Line Buffer (画像) 423
 Virtex2 Line Buffer (画像) 422
 White Gaussian Noise Generator (通信) 424
 演算デザイン 376
 画像デザイン 376
 制御ロジック デザイン 375
 通信デザイン 375
 サチュレートおよび丸めロジック 259

ち

チュートリアル
 M コード/ハードウェア協調シミュレーション
 MATLAB ハードウェア協調シミュレーションの使用 485

は

ハードウェア協調シミュレーション 483
 M コード アクセス 483
 パイプライン
 サチュレートおよび丸めロジック 259
 パラメータ
 共通オプション 45

ふ

プログラム生成
 System Generator ブロック図 469
 ブロックセット ライブラリ
 構造 22
 ブロック パラメータ
 共通オプション 45

め

メモリの結合
 From FIFO ブロック 203
 From Register ブロック 205
 Shared Memory ブロック 314
 To FIFO ブロック 343
 To Register ブロック 345

ゆ

ユーティリティ関数

M コード/ハードウェア協調シ
ミュレーション 502

れ

例

M コード/ハードウェア協調シ
ミュレーション 485