LogiCORE IP MicroBlaze Micro Controller System (v1.1)

DS865 April 24, 2012 **Product Specification**

Introduction

The LogiCORE™ MicroBlaze™ Micro Controller System (MCS) is a complete standalone processor system intended for controller applications. It is highly integrated and includes the MicroBlaze processor, local memory for program and data storage as well as a tightly coupled I/O module implementing a standard set of peripherals.

The MicroBlaze processor included in the MCS has a fixed configuration, optimized for minimal area. The full-featured MicroBlaze processor is available in the ISE® Design Suite Embedded Edition.

Features

- MicroBlaze processor
- Local Memory
- MicroBlaze Debug Module (MDM)
- Tightly Coupled I/O Module including
  - IO Bus
  - Interrupt Controller using fast interrupt mode
  - UART
  - Fixed Interval Timers
  - Programmable Interval Timers
  - General Purpose Inputs
  - General Purpose Outputs

LogiCORE Facts				
Core Specifics				
Supported Device Family[1]	Spartan®-3, Spartan-6, Virtex®-4, Virtex-5, Virtex-6, Virtex-7, Kintex™-7, Artix™-7, Zynq™-7000			
Supported User Interfaces	Local Memory Bus (LMB), Dynamic Reconfiguration Port (DRP)			
Resources[2]				
Configuration	**LUTs**	**FFs**	**DSP Slices**	**Block RAMs**
Minimum	546	276	0	1
Maximum	1201	1619	0	32
Provided with Core				
Documentation	Product Specification			
Design Files	VHDL			
Example Design	Not Provided			
Test Bench	Not Provided			
Constraints File	Not Provided			
Supported SW Driver[3]	Standalone			
Tested Design Tools				
Design Entry Tools	ISE 14.1			
Simulation[4]	Mentor Graphics ModelSim			
Synthesis Tools	ISE 14.1			
Support				
Provided by Xilinx @ www.xilinx.com/support				

1. For a complete listing of supported families, see the release notes for this core.
2. Resources listed here are for Virtex-6 devices. For more complete device performance numbers, see Table 43.
3. Standalone I/O Module driver details can be found in the SDK directory (<*install_directory*>/doc/usenglish/xilinx_drivers.htm).
4. For the supported versions of the tools, see the ISE Design Suite 14: Release Notes Guide.

Functional Description

The MicroBlaze Micro Controller System (MCS) is highly integrated standalone processor system intended for controller applications. Data and program is stored in a local memory, debug is facilitated by the MicroBlaze Debug Module, MDM. A standard set of peripherals is also included, providing basic functionality like interrupt controller, UART, timers and general purpose input and outputs.

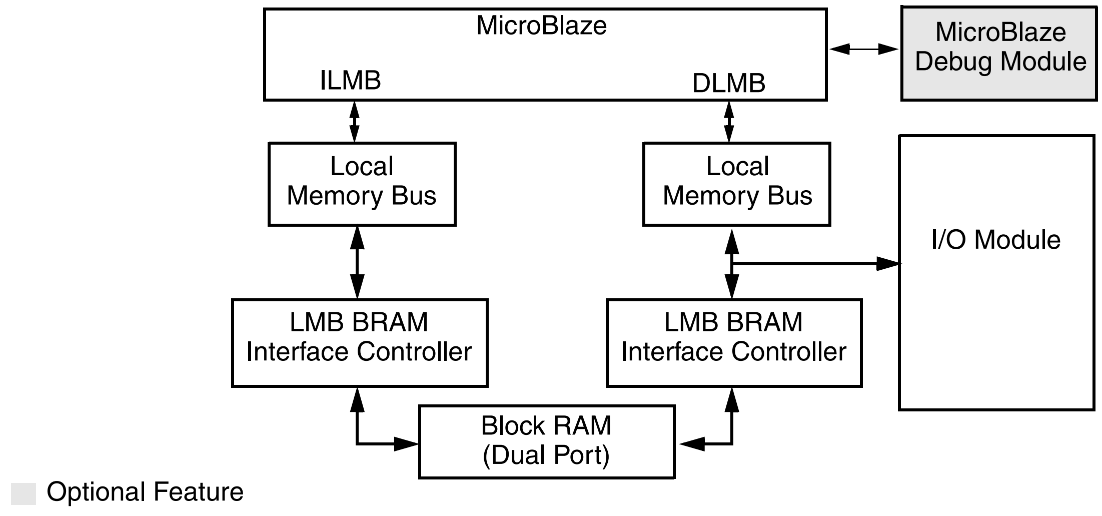

Figure 1: **MicroBlaze Micro Controller System (MCS)**

MicroBlaze

The MicroBlaze embedded processor soft core is a reduced instruction set computer (RISC) optimized for implementation in Xilinx® Field Programmable Gate Arrays (FPGAs). Detailed information on the MicroBlaze processor can be found in the *MicroBlaze Processor Reference Guide* [Ref 1].

The MicroBlaze parameters in MicroBlaze MCS are fixed except for the possibility to enable/disable the debug functionality. The values of all MicroBlaze parameters in MicroBlaze MCS can be found in Table 6. These values correspond to the MicroBlaze Configuration Wizard *Minimum Area* configuration.

Local Memory

Local memory is used for data and program storage and is implemented using Block RAM. The size of the local memory is parameterized and can be between 4kB and 64kB. The local memory is connected to MicroBlaze through the Local Memory Bus, LMB, and the LMB BRAM Interface Controllers. Detailed information on LMB can be found in Local Memory Bus (LMB) V10 (DS445) and detailed information on the LMB BRAM Interface Controller can be found in IP Processor LMB BRAM Interface Controller (DS452).

The LMB Bus and the LMB BRAM Interface Controller parameters are fixed except for the memory size. The value of the parameters can be found in Table 8, Table 9, Table 10 and Table 11.

Debug

The MicroBlaze Debug Module, MDM, connects MicroBlaze debug logic with the XMD low level debugger. XMD can be used for downloading software, to set break points, view register and memory contents etc. Detailed information about MDM can be found in MicroBlaze Debug Module (MDM) (DS641).

The MDM parameters are fixed and their values can be found in Table 12.

I/O Module

The I/O Module is a light-weight implementation of a set of standard IO functions commonly used in a MicroBlaze processor sub-system. The input/output signals of the I/O Module are shown in Figure 2. The detailed list of signals is listed and described in Table 4.

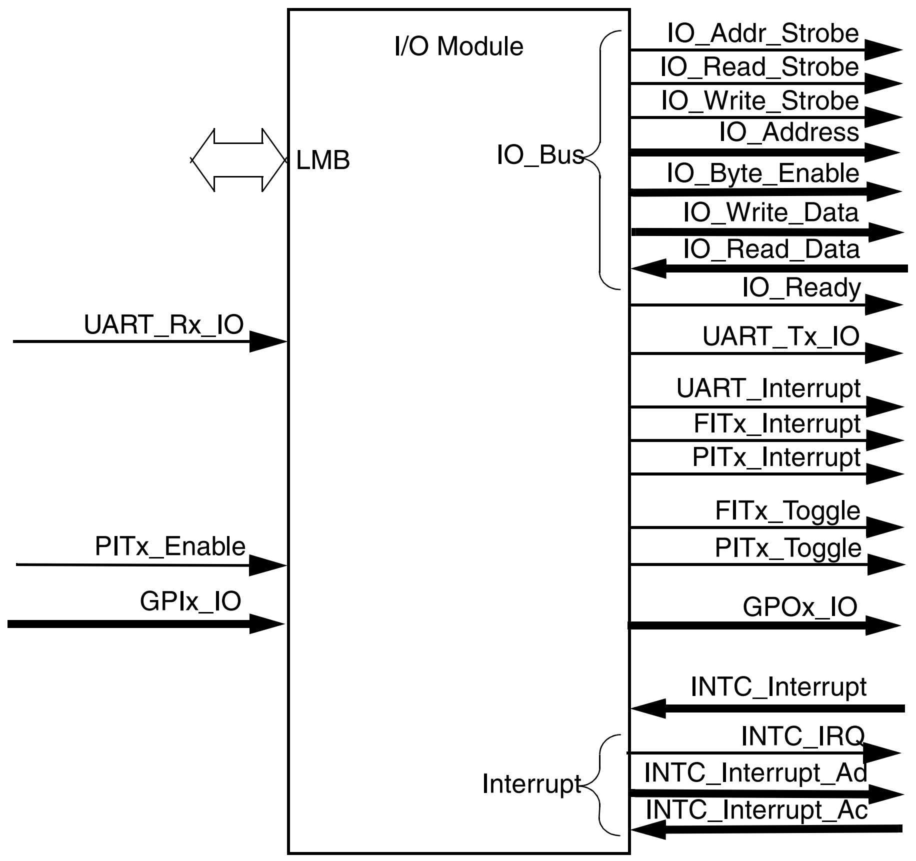

Figure 2: **I/O Module Block Diagram**

IO Bus

The IO Bus provides a simple bus for accessing to external modules using MicroBlaze Load/Store instructions. The IO Bus is mapped at address 0xC0000000-0xFFFFFFFF in the MicroBlaze memory space, with the IO Bus address directly reflecting the byte address used by MicroBlaze Load/Store instructions. IO Bus data is 32-bit wide, with byte enables to write byte and half-word data.

The IO Bus has a ready handshake to handle different waitstate needs, from IO_Ready asserted the cycle after the IO_Addr_Strobe is asserted to as many cycles as needed. There is no time-out on the IO Bus and MicroBlaze is stalled until IO_Ready is asserted. IO_Address, IO_Byte_Enable, IO_Write_Data, IO_Read_Strobe, IO_Write_Strobe are only valid when IO_Addr_Strobe is asserted. For read access IO_Read_Data is sampled at the rising Clk edge, when the slave has asserted IO_Ready.

IO Bus read and write transactions can be found in the two following timing diagrams in Figure 3 and Figure 4.

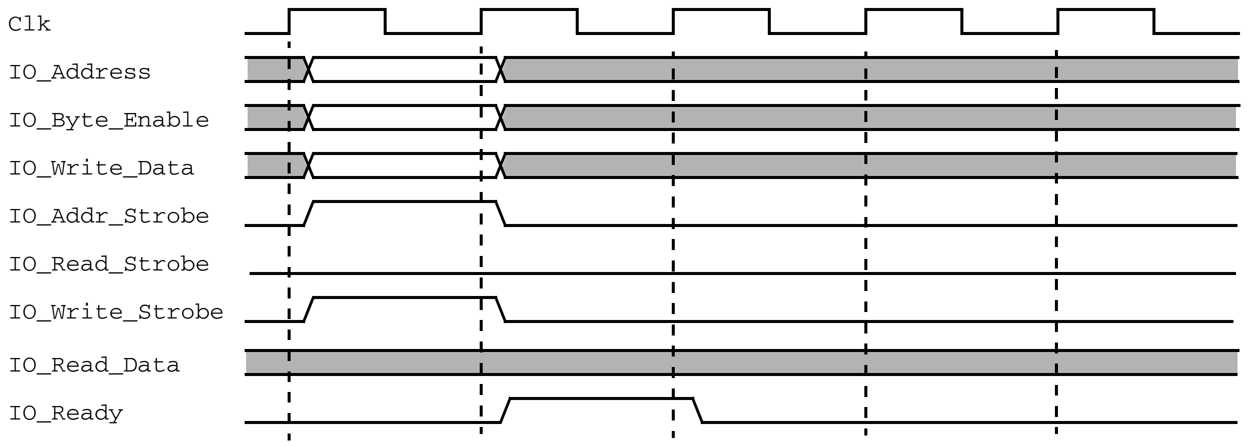

Figure 3: **IO Bus Write**

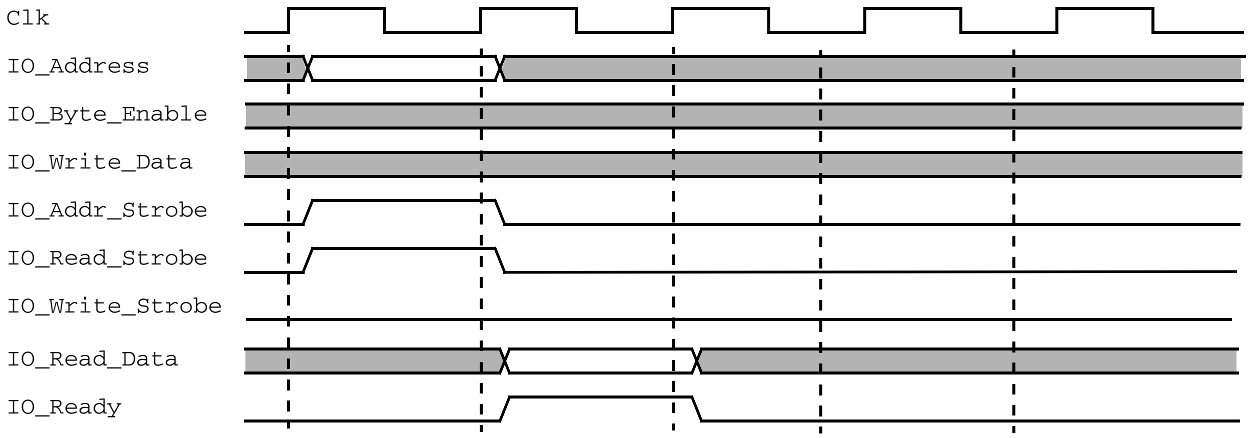

Figure 4: **IO Bus Read**

The byte enable signals indicate which byte lanes of the data bus contain valid data. Valid values for `IO_Byte_Enable` are shown in Table 1 below.

Table 1: **Valid Values for IO_Byte_Enable[3:0]**

IO_Byte_Enable	IO_Address	IO_Data_Write and IO_Data_Read Byte Lanes Used			
[3:0]	[1:0]	[31:24]	[23:16]	[15:8]	[7:0]
0001	00				●
0010	01			●	
0100	10		●		
1000	11	●			
0011	00			●	●
1100	10	●	●		
1111	00	●	●	●	●

The IO Bus is fully compatible with the Xilinx Dynamic Reconfiguration Port.(DRP). This configuration port supports partial dynamic reconfiguration of functional blocks, such as CMTs, clock management, XADC, serial transceivers, and the PCIe® block.

The nominal connection of the IO Bus to the DRP is illustrated in Table 2.

Table 2: **Mapping of the IO Bus to the Dynamic Reconfiguration Port**

MicroBlaze MCS Signal	DRP Signal	Note
Clk	DCLK	
IO_Addr_Strobe	DEN	
IO_Read_Strobe	-	Not used by DRP
IO_Write_Strobe	DWE	
IO_Address[m+2:2]	DADDR[m:0]	Uses 32-bit word access for DRP
IO_Byte_Enable	-	Only 32-bit word accesses used for DRP
IO_Write_Data[n:0]	DI[n:0]	Data width depends on DRP (n < 32)
IO_Read_Data[n:0]	DO[n:0]	Data width depends on DRP (n < 32)
IO_Ready	DRDY	

For a detailed description of the DRP, see *7 Series FPGAs Configuration User Guide* [Ref 5].

UART

The Universal Asynchronous Receiver Transmitter (UART) interface provides the controller interface for asynchronous serial data transfers. Features supported include:

- One transmit and one receive channel (full duplex)
- Configurable number of data bits in a character (5-8)
- Configurable parity bit (odd or even)
- Configurable baud rate

The UART performs parallel-to-serial conversion on characters received through LMB and serial-to-parallel conversion on characters received from a serial peripheral.

The UART is capable of transmitting and receiving 8, 7, 6 or 5-bit characters, with 1-stop bit and odd, even or no parity. The UART can transmit and receive independently.

The device can be configured and its status can be monitored via the internal register set. The UART also asserts the UART_Interrupt output when the receiver becomes non-empty, when the transmitter becomes empty or when an error condition has occurred. The individual interrupt events are connected to the Interrupt Controller of the I/O Module and can be used to assert the INTC_IRQ output signal.

Fixed Interval Timer, FIT

The Fixed Interval Timer generates a strobe (interrupt) signal at fixed intervals. The Fixed Interval Timer asserts the output signal FITx_Interrupt one clock cycle every C_FITx_No_CLOCKS. Operation begins immediately after FPGA configuration and the clock is running. The FITx_Toggle output signal is toggled each time FITx_Interrupt is asserted, creating a 50% duty cycle output with twice the FITx_Interrupt period.

Using the parameter C_FITx_INTERRUPT, the FIT can be connected to the Interrupt Controller of the I/O Module and used for generating interrupts every time the strobe occurs.

Programmable Interval Timer, PIT

The Programmable Interval Timer, PIT, has a configurable width from 1 to 32. The PIT operation and period are controlled by software. The timer starts counting when it is enabled by setting the EN bit in the PITx Control Register.

The PITx_Interrupt output signal is asserted one clock cycle when the timer lapses. The timer can be used in continuous mode, where the timer reloads automatically when it lapses. In continuous mode, the period between two PITx_Interrupt assertions is the value in PITx Preload Register + 2 count events.

The PIT can also be used in one-shot mode, where the timer stops when it has reached zero. The timer is implemented by means of a counter that is pre-loaded with the timer value and then decremented. When the counter reaches zero, the timer lapses, and the interrupt signal is generated.

The PITx_Toggle output signal is toggled each time PITx_Interrupt is asserted, creating a 50% duty cycle output with twice the PITx_Interrupt period when the timer is operated in continuous mode.

The value of the counter that implements the timer can be read by software if the C_PITx_Readable parameter is enabled.

The PIT can have a pre-scaler connected from any FITx, PITx, or External. The pre-scaler is selected by the C_PITx_PRESCALER parameter. The PIT has no pre-scaler by default. If External is selected the input signal PITx_Enable is used as pre-scaler. Selecting External as pre-scaler can also be used to measure the width in clock cycles of a signal connected to the PITx_Enable input.

Using the parameter C_PITx_INTERRUPT, the PIT can be connected to the Interrupt Controller of the I/O Module and used for generating interrupts every time it lapses.

General Purpose Output, GPO

The General Purpose Output, GPO, drives I/O Module GPO output signals defined by the value of the GPOx register, programmable from software. The width of the GPOx is defined by the C_GPOx_SIZE and the initial value is defined by the parameter C_GPOx_INIT. When the GPOx register is written, the value of the GPOx output signals will change accordingly.

General Purpose Input, GPI

The General Purpose Input, GPI, makes it possible for software to sample the value of the I/O Module GPI input signals by reading the GPIx register. The width of GPIx is defined by the parameter C_GPIx_SIZE.

Interrupt Controller, INTC

The Interrupt Controller handles both I/O module internal interrupt events and external ones. The internal interrupt events originate from the UART and the Fixed or Programmable Interval Timers. For an internal interrupt to be generated on the INTC_IRQ output, the corresponding I/O Module parameter needs to be set, e.g. C_UART_RX_INTERRUPT=1, and that particular interrupt needs to be enabled in the Interrupt Enable Register.

The Interrupt Controller supports up to 16 external interrupts using the INTC_Interrupt inputs. The number of external interrupts is defined by the parameter C_INTC_INTR_SIZE. The external interrupt signals can be individually configured as either edge or level sensitive by the C_INTC_LEVEL_EDGE parameter. The polarity of the external interrupt signals can be individually configured to be either active-High (rising edge) or active-Low (falling edge) by the C_INTC_POSITIVE parameter. Interrupt events for external interrupt sources are generated according to Table 3.

Table 3: **Interrupt Event Generation**

C_INTC_LEVEL_EDGE(x)	C_INTC_POSITIVE(x)	INTC_Interrupt(x) Input
0	0	0
0	1	1
1	0	1 -> 0
1	1	0 -> 1
0	0	0

The current status of all interrupt sources can be read from the Interrupt Status Register. The current status of all enabled interrupts can be read from the Interrupt Pending Register.

An interrupt is cleared in both the Interrupt Status and Interrupt Pending Registers by writing to the Interrupt Acknowledge Register, with bits set corresponding to the interrupts that should be cleared.

Either normal or fast interrupt mode can be used, based on latency requirement. Fast interrupt mode is enabled for an interrupt by setting the corresponding bit in the Interrupt Mode Register (IRQ_MODE). In this case the Interrupt Controller drives the interrupt vector address of the highest priority interrupt on the INTC_Interrupt_Address port, along with INTC_IRQ. The generated interrupt is cleared based on acknowledge received from the processor via the INTC_Interrupt_Ack port. The processor sends 0b01 on this port when the interrupt is being acknowledged by the processor (i.e. when branching to the interrupt service routine), sends 0b10 when executing a return from interrupt instruction in the interrupt service routine, and sends 0b11 when interrupts are re-enabled. The bit in IRQ_STATUS corresponding to the interrupt is cleared when 0b10 or 0b11 is seen on the port. The interrupt vector address for each interrupt is stored in the corresponding IRQ_VECTOR register.

Tool Flow

The MicroBlaze MCS utilizes the generic tool flow of all LogiCORE IP. This flow requires some manual steps in PlanAhead™ and Project Navigator primarily to support software development. The SDK software development flow is also briefly described here.

Generic PlanAhead and Project Navigator Tool Flow

The generic tool flow in PlanAhead and Project Navigator is illustrated by the flow chart in Figure 5.

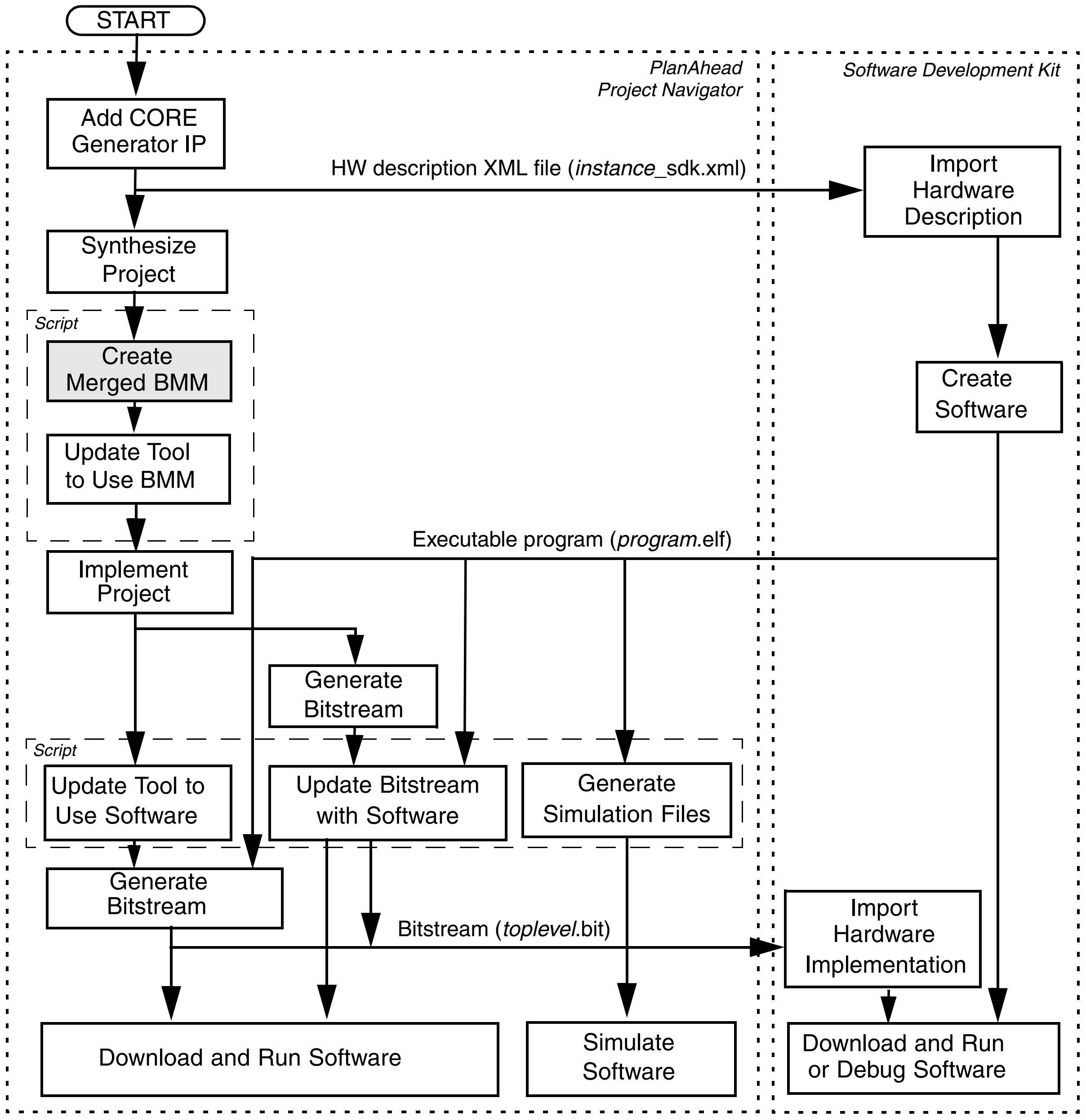
START
PlanAhead
Project Navigator
Software Development Kit
Add CORE Generator IP
HW description XML file (*instance*_sdk.xml)
Import Hardware Description
Synthesize Project
Script
Create Merged BMM
Update Tool to Use BMM
Create Software
Executable program (*program*.elf)
Implement Project
Generate Bitstream
Script
Update Tool to Use Software
Update Bitstream with Software
Generate Simulation Files
Generate Bitstream
Bitstream (*toplevel*.bit)
Import Hardware Implementation
Download and Run Software
Simulate Software
Download and Run or Debug Software

Figure 5: **Generic PlanAhead and Project Navigator Tool Flow**

This flow illustrates the specific steps required to implement a project with the MicroBlaze MCS in PlanAhead or Project Navigator, and the relationship between the hardware and software tools.

Each of the steps are described in general here. Specific commands used in PlanAhead, ISE Project Navigator and Xilinx Software Development Kit (SDK) are covered in the following sections.

- **Add CORE Generator™ IP**: In this step the specific MicroBlaze MCS component parameters are defined using the configuration dialog, and the component is generated and synthesized. Several files are created during this step:
  - `component-name_sdk.xml` - Hardware description of the specific component, imported into SDK.
  - `component-name.bmm` - The BMM file of the specific component, which defines the configuration of the Block RAMs used by the component. This file is necessary to update the bitstream with the software to be executed by MicroBlaze.
  - `microblaze_mcs_setup.tcl` - A script that is available to automate certain steps in the flow.
  - `mb_bootloop_le.elf` - An infinite loop, which is the default program used to update the bitstream.

  Note: The full hierarchical name of the component in the design as well as the input clock frequency must be decided in this step, and adhered to when the component is later instantiated.
- **Create Merged BMM**: This step is optional, and is only required when the project contains more than one MicroBlaze MCS core.

  The step can be performed by executing the script `microblaze_mcs_setup.tcl` in the tool Tcl Console. The script creates a merged BMM file, called `microblaze_mcs_merged.bmm`, which includes all MicroBlaze MCS components in the project.

  To perform the step manually, find all the MicroBlaze MCS core BMM files in the project, and merge them using a text editor. The contents of the files can simply be concatenated in any order, except that the id number at the end of each ADDRESS_MAP line (100 in the input files) must be changed to a unique number for each ADDRESS MAP line. It is suggested to use the numbers 100, 200, etc.
- Update Tool to Use BMM: This step informs the tool about the BMM file to use, either the component BMM file, `component-name.bmm`, or he merged file from the previous step when the project contains more than one MicroBlaze MCS core.

  The step is also performed by executing the script `microblaze_mcs_setup.tcl` in the tool Tcl Console. Project properties are updated to use the appropriate BMM file, by adding a command line option to the **`ngdbuild`** command.

  To perform the step manually, see the specific commands for PlanAhead or ISE Project Navigator below.
- **Implement Project**: This is the normal step to create the implemented netlist.
- **Update Tool to Use Software**: This step informs the tool about the software executable files to use, one for each MicroBlaze MCS component in the project. After this step, whenever the bitstream is generated, it is updated with the contents of the software executable files.

  The step can be performed by invoking the `microblaze_mcs_data2mem` Tcl procedure, with one argument for each MicroBlaze MCS component in the project, indicating the corresponding software executable ELF file. Project properties are updated to use the appropriate files, by adding a command line option to the **`bitgen`** command.

  To perform the step manually, see the specific commands for PlanAhead or ISE Project Navigator below.

  Note: With more than one MicroBlaze MCS component in the project, the order in which to enter the ELF file arguments can be determined by first invoking the Tcl procedure without arguments.
- **Generate Bitstream**: This is the normal step to generate the bitstream, which creates two hardware implementation files that can be imported into SDK, for running or debugging software:
  - `toplevel.bit` - The bitstream created by the tools.
  - `component-name_bd.bmm` or `microblaze_mcs_merged_bd.bmm` - The BMM file updated with Block RAM placement. This file is used when updating the bitstream with the software created in SDK.

  If this step is performed after the tool has been updated to use software, the bitstream will be updated with the contents of the software executable files. If not, the bitstream can be updated with software after it has been generated.

- **Update Bitstream with Software**: This step is used to update the previously generated bitstream with all software executable files. If the software has been changed, this is the only step necessary to modify the bitstream. It is not necessary to regenerate the bitstream in this case.

  The step is also performed by invoking the `microblaze_mcs_data2mem` Tcl procedure. The procedure invokes **`data2mem`** to update the bitstream.

  To perform the step manually, see the specific commands for PlanAhead or ISE Project Navigator below.
- **Generate Simulation Files**: This step is used to generate MEM files used when simulating the project. These files contain the memory content of all Block RAMs used when simulating the project. When behavioral simulation is started, the files are automatically read by the simulator when elaborating the design.

  The step is also performed by invoking the microblaze_mcs_data2mem Tcl procedure. The procedure invokes **`data2mem`** to create the files *`component-name`*`.lmb_bram_`*`n`*`.mem` for each MicroBlaze MCS component.

  To perform the step manually, see the specific commands for PlanAhead or ISE Project Navigator below.
- **Download and Run Software**: When downloading the updated bitstream to the FPGA with **`impact`**, the software immediately starts to run as soon as reset is deactivated.
- **Import Hardware Description**: This step is performed in SDK, using the hardware description file *`component-name`*`_sdk.xml` created when the MicroBlaze MCS component was generated. If there are more than one component in the project, a hardware platform specification must be imported for each component.
- **Import Hardware Implementation**: This step is performed in SDK, using the *`toplevel`*`.bit` bitstream and the *`component-name`*`_bd.bmm` or `microblaze_mcs_merged_bd.bmm` BMM file.
- **Download and Run or Debug Software**: Once the FPGA has been programmed, software can be run or debugged as usual in SDK.

PlanAhead

The PlanAhead commands to achieve the MicroBlaze MCS specific steps above are detailed here.

Using the provided script to perform the steps:

- **Create Merged BMM** and **Update Tool to Use BMM**:

  In the Tcl Console type the following commands:

```
cd project-path
source project-name.srcs/sources_1/ip/component-name/microblaze_mcs_setup.tcl
```

- **Update Tool to Use Software**, **Update Bitstream with Software** and **Generate Simulation Files**:

  Type the following command in the Tcl Console, to perform this with one MicroBlaze MCS component:

  `microblaze_mcs_data2mem /`*`sdk-workspace-path`*`/`*`sdk-program`*`/Debug/`*`sdk-program`*`.elf`

  For each additional MicroBlaze MCS component, add an additional executable ELF file to the command line.

Performing the steps manually:

- **Update Tool to Use BMM**:

  With one MicroBlaze MCS component, type the following command in the Tcl Console, using the appropriate absolute directory path:

```
config_run [current_run] -program ngdbuild -option {More Options} -value \
  {-bm /project-path/project-name.srcs/sources_1/ip/component-name/component-name_bd.bmm}
```

  With more than one MicroBlaze MCS component, the -bm option must indicate the merged BMM file instead.

- **Update Tool to Use Software**:

  With one MicroBlaze MCS component, type the following command in the Tcl Console, using the appropriate absolute directory path:

```
config_run [current_run] -program bitgen -option {More Options} -value \
  {-bd /sdk-workspace-path/sdk-program/Debug/sdk-program.elf tag component-name}
```

  With more than one MicroBlaze MCS component, the -bd option must be repeated for each component.

- **Update Bitstream with Software**:

  To perform this step with one MicroBlaze MCS component, invoke **`data2mem`** with e.g. the following command line options, using the appropriate directory paths to the indicated files:

```
cd project-path
data2mem -p part \
  -bm project-name.srcs/sources_1/ip/component-name/component-name_bd.bmm \
  -bd /sdk-workspace-path/sdk-program/Debug/sdk-program.elf tag component-name \
  -bt project-name.runs/impl_1/toplevel.bit \
  -o b project-name.runs/impl_1/download.bit
```

  Here *`part`* is the complete part name, consisting of device, package, and speed concatenated.

  With more than one MicroBlaze MCS component, the -bm option must indicate the merged BMM file.updated with Block RAM placement.

  For each additional MicroBlaze MCS component, the -bd option has to be repeated, followed by the appropriate executable ELF file, the keyword tag, and the component name.

- **Generate Simulation Files**:

  To perform this step manually with one MicroBlaze MCS component, invoke **`data2mem`** with e.g. the following command line options, using the appropriate directory paths for the indicated files:

```
cd project-path
data2mem -p part \
  -bm project-name.srcs/sources_1/ip/component-name/component-name.bmm \
  -bd /sdk-workspace-path/sdk-program/Debug/sdk-program.elf tag component-name \
  -bx project-name.sim/sim_1 -u
```

  Here *`part`* is the complete part name, consisting of device, package, and speed concatenated.

  For each additional MicroBlaze MCS component, the -bd option has to be repeated, followed by the appropriate executable ELF file, the keyword tag, and the component name.

  If the output directory indicated by the -bx option does not exist, it has to be created manually.

For additional information, see *Xilinx PlanAhead Manuals* [Ref 4].

Project Navigator

The Project Navigator commands to achieve the MicroBlaze MCS specific steps above are detailed here.

Using the provided script to perform the steps:

- **Create Merged BMM** and **Update Tool to Use BMM**:
  - If the Tcl Console is not visible, select **View** → **Panels** → **Tcl Console** in the menu.
  - In the Tcl Console type the following command:

```
source ipcore_dir/microblaze_mcs_setup.tcl
```

- **Update Tool to Use Software**, **Update Bitstream with Software** and **Generate Simulation Files**:

  Type the following command in the Tcl Console, to perform this with one MicroBlaze MCS component:

```
microblaze_mcs_data2mem /sdk-workspace-path/sdk-program/Debug/sdk-program.elf
```

  For each additional MicroBlaze MCS component, add an additional executable ELF file to the command line.

Performing the steps manually:

- **Update Tool to Use BMM**:

  With one MicroBlaze MCS component, type the following command in the Tcl Console:

  ```
  project set {Other Ngdbuild Command Line Options} {-bm ipcore_dir/component-name_bd.bmm}
  ```

  With more than one MicroBlaze MCS component, the -bm option must indicate the merged BMM file instead.

- **Update Tool to Use Software**:

  With one MicroBlaze MCS component, type the following command in the Tcl Console, using the appropriate absolute directory path:

  ```
  project set {Other Bitgen Command Line Options} \
    {-bd /sdk-workspace-path/sdk-program/Debug/sdk-program.elf tag component-name}
  ```

  With more than one MicroBlaze MCS component, the -bd option must be repeated for each component.

- **Update Bitstream with Software**:

  To perform this step with one MicroBlaze MCS component, invoke **`data2mem`** with e.g. the following command line options, using the appropriate directory paths to the indicated files:

  ```
  cd project-path
  data2mem -p part \
    -bm ipcore_dir/component-name_bd.bmm \
    -bd /sdk-workspace-path/sdk-program/Debug/sdk-program.elf tag component-name \
    -bt project-name.runs/impl_1/toplevel.bit \
    -o b project-name.runs/impl_1/download.bit
  ```

  Here *`part`* is the complete part name, consisting of device, package, and speed concatenated.

  With more than one MicroBlaze MCS component, the -bm option must indicate the merged BMM file.updated with Block RAM placement.

  For each additional MicroBlaze MCS component, the -bd option has to be repeated, followed by the appropriate executable ELF file, the keyword tag, and the component name.

- **Generate Simulation Files**:

  To perform this step manually with one MicroBlaze MCS component, invoke **`data2mem`** with e.g. the following command line options, using the appropriate directory paths for the indicated files:

  ```
  cd project-path
  data2mem -p part \
    -bm ipcore_dir/component-name.bmm \
    -bd /sdk-workspace-path/sdk-program/Debug/sdk-program.elf tag component-name \
    -bx . -u
  ```

  Here *`part`* is the complete part name, consisting of device, package, and speed concatenated.

  For each additional MicroBlaze MCS component, the -bd option has to be repeated, followed by the appropriate executable ELF file, the keyword tag, and the component name.

For additional information, see *Xilinx ISE Manuals* [Ref 3].

SDK

The SDK commands to achieve the MicroBlaze MCS specific steps above are detailed here:

- Import Hardware Description - For each MicroBlaze MCS component to import:
  - Select **File** → **New** → **Xilinx Hardware Platform Specification** in the menu.
  - Click on **Browse**, and navigate to the hardware description file:
    - In PlanAhead this file is typically called `*project-name*.srcs/sources_1/ip/*component-name*/*component-name*_sdk.xml`.
    - In Project Navigator this file is typically called `ipcore_dir/*component-name*_sdk.xml`.
  - Click on **Finish** to perform the import.

  After the hardware description has been imported, a `standalone` board support package can be created, which provides MicroBlaze processor-specific code, and the I/O Module software driver. The MicroBlaze MCS configuration is available in the generated file `microblaze_0/include/xparameters.h`.
- Import Hardware Implementation:
  - Select **Xilinx Tools** → **Program FPGA** in the menu.
  - Click on the first **Browse** button, and navigate to the bitstream:
    - In PlanAhead this file is typically called `*project-name*.runs/impl_1/*toplevel*.bit`.
    - In Project Navigator this file is typically called `*toplevel*.bit`.
  - Click on the first **Browse** button, and navigate to the BMM file updated with Block RAM placement.
    - In PlanAhead with one MicroBlaze MCS component, this file is typically called `*project-name*.srcs/sources_1/ip/*component-name*/*component_name*_bd.bmm`
    - In Project Navigator with one MicroBlaze MCS component, this file is typically called `ipcore_dir/*component_name*_bd.bmm`

    With more than one MicroBlaze MCS component, the merged BMM file updated with Block RAM placement must be selected instead.
  - Click on **Program** to perform the import and program the FPGA.

For additional information, see *Xilinx SDK Help* [Ref 2].

Troubleshooting

This section provides help in diagnosing and correcting issues that may occur with the MicroBlaze MCS specific tool flow above. If an error not listed here is encountered, please refer to the corresponding tool documentation.

For each listed error message, the probable cause of the error, and the suggested corrective action is provided.

Step:	**Create Merged BMM, Update Tool to Use BMM**
Tcl Command:	microblaze_mcs_setup
Error message:	ERROR: Could not find a BMM file for *instances*. Please regenerate the MicroBlaze MCS instances.
Possible causes:	• With PlanAhead, the BMM file has not been generated after customizing a MicroBlaze MCS instance, or after adding an existing IP. • The BMM file has inadvertently been deleted.
Corrective actions:	• With PlanAhead, select each instance and use Generate IP in the context menu, or synthesize the project, and then invoke the command again. • With Project Navigator, double-click on each MicroBlaze MCS instance to regenerate it, and then invoke the command again.

Step:	**Implement Project**
Tool:	Ngdbuild
Error message:	NgdBuild:989 - Failed to process BMM information
Possible causes:	• The parameter “Instance Hierarchical Design Name” set in the MicroBlaze MCS configuration dialog does not match the actual instantiation name or place in the instantiation hierarchy. Note that this is case sensitive in the tools. • The parameter “Memory Size” set in the MicroBlaze MCS configuration dialog has changed, but the corresponding BMM file has not been updated.
Corrective action:	• Change “Instance Hierarchical Design Name” to the correct value in the MicroBlaze MCS configuration dialog. This is the actual name used in the instantiation, prefixed with all hierarchical levels below the top instance, separated with /. • Regenerate the BMM file according to the previous item.

Step:	**Implement Project**
Tool:	Ngdbuild
Error message:	NgdBuild:634 - Cannot open input BMM file
Possible causes:	The Ngdbuild -bm option does not indicate the correct BMM file.
Corrective action:	Change the Ngdbuild option, either manually, or by invoking `microblaze_mcs_setup` in the Tcl Console.

Step:	**Implement Project**
Tool:	Ngdbuild
Error message:	NgdBuild:76 - File "*path*/*component-name*.ngc" cannot be merged into block "*instance-name*" (TYPE="*component-name*") because one or more pins on the block, including pin "*pin-name*", were not found in the file. Please make sure that all pins on the instantiated component match pins in the lower-level design block (irrespective of case). If there are bussed pins on this block, make sure that the upper-level and lower-level netlists use the same bus-naming convention.
Possible causes:	The instantiation does not match the MicroBlaze MCS component, with one or more different input pins.
Corrective action:	Change the instantiation to match the template in *component-name*.vho (VHDL project) or *component_name*.veo (Verilog project).

Step:	**Update Tool to Use Software**
Tcl Command:	microblaze_mcs_data2mem
Error message:	ERROR: Too many arguments. At most *instance-count* ELF files should be given.
Possible causes:	• The command has not been invoked with the correct number of arguments. There should be at most one argument per MicroBlaze MCS core. • The paths to the ELF files include space characters.
Corrective action:	• Invoke the command with the correct number of arguments. To check the number of arguments, and their order, invoke the command without arguments. This will update the project with the boot loop, and list the detected cores in the order the arguments should be given. • Ensure that each path is enclosed in double quotes if it includes space characters.
Step:	**Update Bitstream with Software**
Tcl Command:	microblaze_mcs_data2mem
Error messages:	• ERROR: Could not find *BMM-filename*. Please regenerate the MicroBlaze MCS instance. • ERROR: Could not find *BMM-filename*. Please invoke "microblaze_mcs_setup" and implement the design.
Possible causes:	• With PlanAhead, the BMM file has not been generated after customizing a MicroBlaze MCS instance, or after adding an existing IP. • The BMM file has inadvertently been deleted.
Corrective action:	• With PlanAhead, select each instance and use Generate IP in the context menu, or synthesize the project, invoke the `microblaze_mcs_setup` command again, and then implement the design. • With Project Navigator, double-click on each MicroBlaze MCS instance to regenerate it, invoke the `microblaze_mcs_setup` command again, and then implement the design.
Step:	**Update Bitstream with Software**
Tcl Command:	microblaze_mcs_data2mem
Error messages:	• ERROR: Could not find *ELF-filename*. Please make sure the file exists. • ERROR: *filename* is not an ELF file.
Possible causes:	• The command has not been invoked with the correct file names or paths. • The executable file extension must be .elf. • The paths to the ELF files include space characters. • The paths to the ELF files do not follow Tcl syntax.
Corrective action:	• Invoke the command with the correct file names and paths. • Ensure that the file extension is correct. • Ensure that each path is enclosed in double quotes if it includes space characters. • The path separator character must be `/`.
Step:	**Update Bitstream with Software**
Tool:	Data2MEM
Error messages:	ERROR:Data2MEM:31 - Out of bounds code segment for ram space in '*BMM-filename*'. Memory space '*component-name*.lmb_bram' occupies [*address-range*] Code segment *index* occupies [*address-range*]
Possible causes:	The MicroBlaze MCS core memory size is smaller than the size used when creating the software application.
Corrective action:	• Increase the memory size in the MicroBlaze MCS configuration dialog. • Open SDK to automatically detect the changed hardware configuration, and build the program for the available memory size. Should the program not fit in available memory, an error will occur. In this case, increase the memory size in the MicroBlaze MCS configuration dialog.

Step:	**Generate Bitstream**
Tool:	Bitgen
Error message:	The design '*toplevel*.ncd' is missing any BMM information for given BRAM data files. BRAMs can't be initialized with the given data without BMM information. Either BMM information must be given to NGDBuild with a '-bm' option, or embedded BMM information must be included in the source HDL.
Possible causes:	The design has been implemented without the Ngdbuild -bm option to define the BMM file, but with the Bitgen -bd option to define the used ELF files.
Corrective action:	Add the Ngdbuild option, either manually, or by invoking the `microblaze_mcs_setup` command, and then implement the design again.
Step:	**Simulate Software**
Tool:	ISIM
Error message:	ERROR:HDLCompiler:1030 - "*path*/vhdl/src/unisims/primitive/RAMB16BWER.vhd" Line 681: Cannot open file 'int_infile'.
Possible causes:	• The MEM files have not been generated, or are not located in the correct place.
Corrective actions:	• Run Data2MEM manually to create simulation files, or invoke the `microblaze_mcs_data2mem` command with the appropriate ELF files as arguments. • Move the MEM files to the correct place. In PlanAhead, the files are placed in the sim_1 simulation set directory by default. If another simulation set is used, they must be moved to that directory.
Step:	**Simulate Software**
Tool:	ModelSim
Error message:	ERROR:Simulator:777 - Static elaboration of top level VHDL design unit tb in library work failed ** Fatal: (vsim-7) Failed to open VHDL file "*component-name*.lmb_bram_*index*.mem" in r mode.
Possible causes:	The MEM files have not been generated.
Corrective actions:	Run Data2MEM manually to create simulation files, or invoke the `microblaze_mcs_data2mem` command with the appropriate ELF files as arguments.
Step:	**Download and Run Software**
Tool:	Impact
Problem:	No output or mangled output on the UART console.
Possible causes:	• The bitstream has not been configured with software. • Frequency defined in the MicroBlaze MCS settings does not match actual frequency of the connected clock input. • Baud rate and/or other UART setting, defined in the MicroBlaze MCS settings, do not match the terminal program settings.
Corrective actions:	• Run data2mem manually to configure the bitstream with software, or invoke the `microblaze_mcs_data2mem` command with the appropriate ELF files as arguments. • Correct the frequency in the MicroBlaze MCS configuration dialog. • Change the terminal program settings to match the MicroBlaze MCS configuration.

MicroBlaze MCS Signals

The I/O ports and signals for MicroBlaze MCS are listed and described in Table 4.

Table 4: **MicroBlaze MCS Signals**

Port Name	MSB:LSB	I/O	Description
System Signals			
Clk		I	System clock
Reset		I	System reset
MicroBlaze Signals			
Trace_Valid_Instr		O	Valid instruction on trace port
Trace_Instruction	0:31	O	Instruction code
Trace_PC	0:31	O	Program counter
Trace_Reg_Write		O	Instruction writes to the register file
Trace_Reg_Addr	0:4	O	Destination register address
Trace_MSR_Reg	0:14	O	Machine status register
Trace_New_Reg_Value	0:31	O	Destination register update value
Trace_Jump_Taken		O	Branch instruction evaluated true, i.e taken
Trace_Delay_Slot		O	Instruction is in delay slot of a taken branch
Trace_Data_AccessT		O	Valid D-side memory access
Trace_Data_Address	0:31	O	Address for D-side memory access
Trace_Data_Write_Value	0:31	O	Value for D-side memory write access
Trace_Data_Byte_Enable	0:3	O	Byte enables for D-side memory access
Trace_Data_Read		O	D-side memory access is a read
Trace_Data_Write		O	D-side memory access is a write
IO Bus Signals			
IO_Addr_Strobe		O	Address strobe signals valid IO Bus output signals
IO_Read_Strobe		O	IO Bus access is a read
IO_Write_Strobe		O	IO Bus access is a write
IO_Address	31:0	O	Address for access
IO_Byte_Enable	3:0	O	Byte enables for access
IO_Write_Data	31:0	O	Data to write for IO Bus write access
IO_Read_Data	31:0	I	Read data for IO Bus read access
IO_Ready		I	Ready handshake to end IO Bus access
UART Signals			
UART_Rx_IO		I	Receive Data
UART_Tx_IO		O	Transmit Data
UART_Interrupt		O	UART Interrupt
FIT Signals			
FITx_Interrupt(1)		O	FITx timer lapsed
FITx_Toggle(1)		O	Inverted FITx_Toggle when FITx timer lapses

Table 4: **MicroBlaze MCS Signals *(Cont'd)***

Port Name	MSB:LSB	I/O	Description
PIT Signals			
PITx_Enable(1)		I	PITx count enable when C_PITx_PRESCALER = External
PITx_Interrupt(1)		O	PITx timer lapsed
PITx_Toggle(1)		O	Inverted PITx_Toggle when PITx lapses
GPO Signals			
GPOx(1)	[C_GPOx_SIZE - 1]:0	O	GPOx Output
GPI Signals			
GPIx(1)	[C_GPIx_SIZE - 1]:0	I	GPIx Input
INTC Signals			
INTC_Interrupt	0:[C_INTC_INTR_SIZE - 1]	I	External interrupt inputs

1. x = 1, 2, 3 or 4

MicroBlaze MCS Parameters

To allow the user to create an MicroBlaze MCS that is uniquely tailored for a specific system, certain features can be parameterized. This makes it possible for the user to configure a component that only utilizes the resources required by the system, and operates with the best possible performance. The features that can be parameterized in MicroBlaze MCS are shown in Table 5.

The internal modules of the MicroBlaze MCS have fixed configurations detailed in:

- Table 6 - MicroBlaze
- Table 7 - I/O Module
- Table 8 and Table 9 - LMB v10
- Table 10 and Table 11 - LMB BRAM IF Controller
- Table 12 - MicroBlaze Debug Module

Table 5: **MicroBlaze MCS Parameters**

Parameter Name	Feature/Description	Allowable Values	Default Value	VHDL Type
MCS Parameters				
C_FAMILY(1)	FPGA Architecture	Supported architectures	virtex5	string
C_XDEVICE(1)	FPGA Device Name	Supported devices	xc5vlx50t	string
C_XPACKAGE(1)	FPGA Package Name	Supported packages	ff1136	string
C_XSPEEDGRADE(1)	FPGA Speed Grade	Supported speed grades	-1	string
C_MICROBLAZE_INSTANCE(1)	Instance Name		microblaze_0	string
C_PATH	Hierarchical path from top of design to MCS instance		mb/UO	
C_FREQ	Frequency of CLK input		100000000	integer

Table 5: **MicroBlaze MCS Parameters *(Cont'd)***

Parameter Name	Feature/Description	Allowable Values	Default Value	VHDL Type
C_MEMSIZE	Local Memory Size in bytes	4096, 8192, 16384, 32768, 65536	8192	integer
C_DEBUG_ENABLE	Enable implementation of debug	0 = Not Used 1 = Used	0	Integer
IO Bus Parameter				
C_USE_IO_BUS	Use IO Bus	0 = Not Used 1 = Used	0	integer
UART Parameters				
C_USE_UART_RX	Use UART Receive	0 = Not Used 1 = Used	0	integer
C_USE_UART_TX	Use UART Transmit	0 = Not Used 1 = Used	0	integer
C_UART_BAUDRATE	Baud rate of the UART in bits per second	integer (e.g. 115200)	9600	integer
C_UART_DATA_BITS	The number of data bits in the serial frame	5 - 8	8	integer
C_UART_USE_PARITY	Determines whether parity is used or not	0 = No Parity 1 = Use Parity	0	integer
C_UART_ODD_PARITY	If parity is used, determines whether parity is odd or even	0 = Even Parity 1 = Odd Parity	0	integer
C_UART_RX_INTERRUPT	Use UART RX Interrupt in INTC	0 = Not Used 1 = Used	0	integer
C_UART_TX_INTERRUPT	Use UART TX Interrupt in INTC	0 = Not Used 1 = Used	0	integer
C_UART_ERROR_INTERRUPT	Use UART ERROR Interrupt in INTC	0 = Not Used 1 = Used	0	integer
FIT Parameters				
C_USE_FITx(1)	Enable implementation of FIT	0 = Not Used 1 = Used	0	integer
C_FITx_No_CLOCKS(2)	The number of clock cycles between strobes	>2	6216	integer
C_FITx_INTERRUPT(2)	Use FITx_Interrupt in INTC	0 = Not Used 1 = Used	0	integer
PIT Parameters				
C_USE_PITx(2)	Enable implementation of PIT	0 = Not Used 1 = Used	0	integer
C_PITx_SIZE(2)	Size of PITx counter	1 - 32	1	integer
C_PITx_READABLE(2)	Make PITx counter software readable	0 = Not SW readable 1 = SW readable	1	integer

Table 5: **MicroBlaze MCS Parameters *(Cont'd)***

Parameter Name	Feature/Description	Allowable Values	Default Value	VHDL Type
C_PITx_PRESCALER[2][3]	Select PITx prescaler	0 = No prescaler 1 = FIT1 2 = FIT2 3 = FIT3 4 = FIT4 5 = PIT1 6 = PIT2 7 = PIT3 8 = PIT4 9 = External	0	integer
C_PITx_INTERRUPT[2]	Use PITx_Interrupt in INTC	0 = Not Used 1 = Used	0	integer
GPO Parameters				
C_USE_GPOx[2]	Use GPOx	0 = Not Used 1 = Used	0	integer
C_GPOx_SIZE[2]	Size of GPOx	1 - 32	32	integer
C_GPOx_INIT[2]	Initial value for GPOx	Fit Range (31:0)	all zeros	std_logic_vector
GPI Parameters				
C_USE_GPIx[2]	Use GPIx	0 = Not Used 1 = Used	0	integer
C_GPIx_SIZE[2]	Size of GPIx	1 - 32	32	integer
INTC Parameters				
C_INTC_USE_EXT_INTR	Use I/O Module external interrupt inputs	0 = Not Used 1 = Used	0	integer
C_INTC_INTR_SIZE	Number of external interrupt inputs used	1 - 16	1	integer
C_INTC_LEVEL_EDGE	Level or edge triggered for each external interrupt	For each bit: 0 = Level 1 = Edge	level	std_logic_vector
C_INTC_POSITIVE	Polarity for each external interrupt	For each bit: 0 = active-Low 1 = active-High	active-High	std_logic_vector

1. Values automatically populated by tool.
2. x=1, 2, 3 or 4.
3. Selecting PIT prescaler the same as PITx is illegal, e.g. PIT2 cannot be prescaler to itself.

Table 6: **Internal MicroBlaze Parameters Settings**

Parameter Name	Feature/Description	Value
C_FAMILY	Target Family	Value of MicroBlaze MCS parameter C_FAMILY
C_AREA_OPTIMIZED	Select implementation to optimize area with lower instruction throughput	1
C_INTERCONNECT	Select interconnect 1 = PLBv46	1
C_ENDIANNESS	Select endianness (1 = Little endian)	1
C_FAULT_TOLERANT	Implement fault tolerance	0

Table 6: **Internal MicroBlaze Parameters Settings *(Cont'd)***

Parameter Name	Feature/Description	Value
C_LOCKSTEP_SLAVE	Lockstep Slave	0
C_AVOID_PRIMITIVES	Disallow FPGA primitives	0
C_PVR	Processor version register mode selection All other PVR parameters are don't care.	0
C_RESET_MSR	Reset value for MSR register	0x00
C_INSTANCE	Instance Name	Value of MicroBlaze MCS parameter C_MICROBLAZE_INSTANCE
C_D_PLB	Data side PLB interface. All other DPLB parameters are don't care.	0
C_D_AXI	Data side AXI interface All other DAXI parameters are don't care.	0
C_D_LMB	Data side LMB interface	1
C_I_PLB	Instruction side PLB interface. All other IPLB parameters are don't care.	1
C_I_AXI	Instruction side AXI interface. All other IAXI parameters are don't care.	0
C_I_LMB	Instruction side LMB interface	1
C_USE_BARREL	Include barrel shifter	0
C_USE_DIV	Include hardware divider	0
C_USE_HW_MUL	Include hardware multiplier	0
C_USE_FPU	Include hardware floating point unit	0
C_USE_MSR_INSTR	Enable use of instructions: MSRSET and MSRCLR	0
C_USE_PCMP_INSTR	Enable use of instructions: CLZ, PCMPBF, PCMPEQ, and PCMPNE	0
C_USE_REORDER_INSTR	Enable use of instructions: LBUR, LHUR, LWR, SBR,SHR, SWR, SWAPB, and SWAPH	0
C_*EXCEPTION*(1) C_OPCODE_0x0_ILLEGAL C_USE_STACK_PROTECTION	No exceptions are used	0
C_DEBUG_ENABLED	MDM Debug interface	Value of MicroBlaze MCS parameter C_DEBUG_ENABLED
C_NUMBER_OF_PC_BRK	Number of hardware breakpoints	Value of MicroBlaze MCS parameter C_DEBUG_ENABLED
C_NUMBER_OF_RD_ADDR_BRK	Number of read address watchpoints	0
C_NUMBER_OF_WR_ADDR_BRK	Number of write address watchpoints	0
C_INTERRUPT_IS_EDGE	Level/Edge Interrupt	0
C_EDGE_IS_POSITIVE	Negative/Positive Edge Interrupt	1
C_FSL_LINKS	Number of stream interfaces (FSL or AXI) All other stream parameters are don't care	0
C_USE_ICACHE	Instruction cache All other Instruction Cache parameters are don't care	0

Table 6: **Internal MicroBlaze Parameters Settings *(Cont'd)***

Parameter Name	Feature/Description	Value
C_USE_DCACHE	Data cache All other Data Cache parameters are don't care	0
C_USE_MMU	Memory Management All other MMU parameters are don't care	0
C_USE_INTERRUPT	Enable interrupt handling	2
C_USE_EXT_BRK	Enable external break handling	Value of MicroBlaze MCS parameter C_DEBUG_ENABLED
C_USE_EXT_NM_BRK	Enable external non-maskable break handling	Value of MicroBlaze MCS parameter C_DEBUG_ENABLED
C_USE_BRANCH_TARGET_CACHE	Enable Branch Target Cache All other BTC parameters are don't care	0

1. * denotes wildcard and represents any number of characters or numbers.

Table 7: **Internal I/O Module Parameters Settings**

Parameter Name	Feature/Description	Value
C_BASEADDR	LMB I/O Module Register Base Address	0x80000000
C_HIGHADDR	LMB I/O Module Register High Address	0x8000FFFF
C_MASK	LMB I/O Module Register Address Space Decode Mask	0xC0000000
C_IO_HIGHADDR	LMB I/O Module IO Bus Base Address	0xC0000000
C_IO_LOWADDR	LMB I/O Module IO Bus Address	0xFFFFFFFF
C_IO_MASK	LMB I/O Module IO Bus Address Space Decode Mask	0xC0000000
C_LMB_AWIDTH	LMB Address Bus Width	32
C_LMB_DWIDTH	LMB Data Bus Width	32
C_INTC_HAS_FAST	Use fast interrupt mode	1
C_INTC_ADDR_WIDTH	Interrupt Address width	12 - 16(1)

1. Value depends on C_MEMSIZE: 12 for 4096, 13 for 8192, 14 for 16384, 15 for 32768, and 16 for 65536.

Table 8: **Internal LMB_v10 Parameters Settings (ILMB)**

Parameter Name	Feature/Description	Value
C_LMB_NUM_SLAVES	Number of LMB Slaves	1
C_LMB_AWIDTH	LMB Address Bus Width	32
C_LMB_DWIDTH	LMB Data Bus Width	32
C_EXT_RESET_HIGH	Level of external reset	1 = active-High reset

Table 9: **Internal LMB_v10 Parameters Settings (DLMB)**

Parameter Name	Feature/Description	Value
C_LMB_NUM_SLAVES	Number of LMB Slaves	2
C_LMB_AWIDTH	LMB Address Bus Width	32
C_LMB_DWIDTH	LMB Data Bus Width	32
C_EXT_RESET_HIGH	Level of external reset	1 = active-High reset

Table 10: **Internal LMB BRAM IF Controller Parameters Settings (ILMB Controller)**

Parameter Name	Feature/Description	Value
C_BASEADDR	LMB BRAM Base Address	0
C_HIGHADDR	LMB BRAM HIGH Address	Value of MicroBlaze MCS Parameter C_MEMSIZE
C_MASK	LMB Decode Mask	0x80000000
C_LMB_AWIDTH	LMB Address Bus Width	32
C_LMB_DWIDTH	LMB Data Bus Width	32
C_ECC	Implement Error Correction and Detection All other ECC as well AXI and PLB parameters are don't care	0 = No ECC

Table 11: **Internal LMB BRAM IF Controller Parameters Settings (DLMB Controller)**

Parameter Name	Feature/Description	Value
C_BASEADDR	LMB BRAM Base Address	0
C_HIGHADDR	LMB BRAM HIGH Address	Value of MicroBlaze MCS Parameter C_MEMSIZE
C_MASK	LMB Decode Mask	0x80000000
C_LMB_AWIDTH	LMB Address Bus Width	32
C_LMB_DWIDTH	LMB Data Bus Width	32
C_ECC	Implement Error Correction and Detection All other ECC as well as AXI and PLB parameters are don't care	0 = No ECC

Table 12: **MicroBlaze Debug Module Parameters Settings**

Parameter Name	Feature/Description	Value
C_FAMILY	FPGA Architecture	Value of MicroBlaze MCS Parameter C_FAMILY
C_MB_DBG_PORTS	Number of MicroBlaze debug ports	1
C_JTAG_CHAIN	Position in the FPGA JTAG chain	2
C_USE_UART	Enables the UART interface. All other UART as well as AXI and PLB parameters are don't care	0

Parameter - Port Dependencies

The width of many of the MicroBlaze MCS signals depends on design parameters. The dependencies between the design parameters and I/O signals are shown in Table 13.

Table 13: **Parameter-Port Dependencies**

Parameter Name	Ports (Port width depends on parameter)
C_INTC_INTR_SIZE	INTC_Interrupt
C_GPO1_SIZE	GPO1
C_GPO2_SIZE	GPO2
C_GPO3_SIZE	GPO3
C_GPO4_SIZE	GPO4
C_GPI1_SIZE	GPI1
C_GPI2_SIZE	GPI2
C_GPI3_SIZE	GPI3
C_GPI4_SIZE	GPI4

MicroBlaze MCS Register Descriptions

Table 14: **MicroBlaze MCS Address Map**

Address (hex)	Name	Access Type	Description
0x0 - C_MEMSIZE-1	Local Memory	RW	Local Memory for MicroBlaze software
C_MEMSIZE - 0x7FFFFFFF	Reserved		
0x80000000	UART_RX	R	UART Receive Data Register
0x80000004	UART_TX	W	UART Transmit Data Register
0x80000008	UART_STATUS	R	UART Status Register
0x8000000C	IRQ_MODE	W	Interrupt Mode Register
0x80000010	GPO1	W	General Purpose Output 1 Register
0x80000014	GPO2	W	General Purpose Output 2 Register
0x80000018	GPO3	W	General Purpose Output 3 Register
0x8000001C	GPO4	W	General Purpose Output 4 Register
0x80000020	GPI1	R	General Purpose Input 1 Register
0x80000024	GPI2	R	General Purpose Input 2 Register
0x80000028	GPI3	R	General Purpose Input 3 Register
0x8000002C	GPI4	R	General Purpose Input 4 Register
0x80000030	IRQ_STATUS	R	Interrupt Status Register
0x80000034	IRQ_PENDING	R	Pending Interrupt Register
0x80000038	IRQ_ENABLE	W	Interrupt Enable Register
0x8000003C	IRQ_ACK	W	Interrupt Acknowledge Register
0x80000040	PIT1_PRELOAD	W	PIT1 Preload Register
0x80000044	PIT1_COUNTER	R	PIT1 Counter Register
0x80000048	PIT1_CONTROL	W	PIT1 Control Register
0x8000004C	Reserved		
0x80000050	PIT2_PRELOAD	W	PIT2 Preload Register
0x80000054	PIT2_COUNTER	R	PIT2 Counter Register
0x80000058	PIT2_CONTROL	W	PIT2 Control Register
0x8000005C	Reserved		
0x80000060	PIT3_PRELOAD	W	PIT3 Preload Register
0x80000064	PIT3_COUNTER	R	PIT3 Counter Register
0x80000068	PIT3_CONTROL	W	PIT3 Control Register
0x8000006C	Reserved		
0x80000070	PIT4_PRELOAD	W	PIT4 Preload Register
0x80000074	PIT4_COUNTER	R	PIT4 Counter Register
0x80000078	PIT4_CONTROL	W	PIT4 Control Register
0x8000007C	Reserved		
0x80000080 - 0x800000FC	IRQ_VECTOR_0 - IRQ_VECTOR_31	W	Interrupt Address Vector Registers
0x80000100 - 0xBFFFFFFF	Reserved		
0xC0000000 - 0xFFFFFFFF	IO Bus	RW	Mapped to IO Bus address output IO_Address

UART Receive Data Register (UART_RX)

A register contains data received by the UART. Reading of this location will result in reading the current word from the register. When a read request is issued without having received a new character, the previously read data will be read again. This register is a read-only register. Issuing a write request to the register will do nothing but generate the write acknowledgement.

The register is implemented if C_USE_UART_RX is set to 1.

Table 15: **UART Receive Data Register (UART_RX) (C_DATA_BITS=8)**

Reserved		UART_RX	
31	8	7	0

Table 16: **UART Receive Data Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
31:C_UART_DATA_BITS	-	R	0	Reserved
[C_UART_DATA_BITS-1]:0	UART_RX	R	0	UART Receive Data

UART Transmit Data Register (UART_TX)

A register contains data to be output by the UART. Data to be transmitted is written into this register. This is write only location. Issuing a read request to this register generates the read acknowledgement with zero data. Writing this register when the character has not been transmitted will overwrite previously written data, resulting in loss of data.

The register is implemented if C_USE_UART_TX is set to 1.

Table 17: **UART Transmit Data Register (UART_TX) (C_DATA_BITS=8)**

Reserved		UART_TX	
31	8	7	0

Table 18: **UART Transmit Data Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
31:C_UART_DATA_BITS	-	R	0	Reserved
[C_UART_DATA_BITS-1]:0	UART_TX	R	0	UART Transmit Data

UART Status Register (UART_Status)

The UART Status Register contains the status of the receive and transmit registers, and if there are any errors. This is read only register. If a write request is issued to status register it will do nothing but generate write acknowledgement.

The register is implemented if C_USE_UART_RX or C_USE_UART_TX is set to 1.

Table 19: **UART Status Register (UART_Status)**

Reserved		UART_Status	
31	8	7	0

Table 20: **UART Status Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
7	Parity Error	R	0	Indicates that a parity error has occurred after the last time the status register was read. If the UART is configured without any parity handling, this bit is always '0'. The received character is written into the receive register. This bit is cleared when the status register is read. 0 = No parity error has occurred 1 = A parity error has occurred
6	Frame Error	R	0	Indicates that a frame error has occurred after the last time the status register was read. Frame Error is defined as detection of a stop bit with the value 0. The receive character is ignored and not written to the receive register. This bit is cleared when the status register is read. 0 = No Frame error has occurred 1 = A frame error has occurred
5	Overrun Error	R	0	Indicates that a overrun error has occurred since the last time the status register was read. Overrun occurs when a new character has been received but the receive register has not been read. The received character is ignored and not written into the receive register. This bit is cleared when the status register is read. 0 = No interrupt has occurred 1 = Interrupt has occurred
4	-	R	0	Reserved
3	Tx Used	R	0	Indicates if the transmit register is in use 0 = Transmit register is not in use 1 = Transmit register is in use
2	-	R	0	Reserved
1	-	R	0	Reserved
0	Rx Valid Data	R	0	Indicates if the receive register has valid data 0 = Receive register is empty 1 = Receive register has valid data

General Purpose Output x Register (GPOx) (x = 1, 2, 3 or 4)

This register holds the value that will be driven to the corresponding bits in the I/O Module GPOx port output signals. All bits in the register are updated when the register is written.

This register is not implemented if the value of C_USE_GPOx is 0.

Table 21: **General Purpose Output x Register (GPOx)**

Reserved		GPOx	
31	C_GPOx_SIZE	C_GPOx_SIZE-1	0

Table 22: **General Purpose Output x Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
31:C_GPOx_SIZE	-	-	-	Reserved
[C_GPOx_SIZE-1]:0	GPOx	W	0	Register holds data driven to corresponding bits in the GPO port

General Purpose Input x Register (GPIx) (x=1, 2, 3 or 4)

This register reads the value that is input on the corresponding I/O Module GPIx port input signal bits.

This register is not implemented if the value of C_USE_GPIx is 0.

Table 23: **General Purpose Input x Register (GPIx)**

Reserved		GPIx	
31	C_GPIx_SIZE	C_GPIx_SIZE-1	0

Table 24: **General Purpose Input x Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
31:C_GPIx_SIZE	-	R	0	Reserved
[C_GPIx_SIZE-1]:0	GPIx	R	0	Register reads value input on the I/O Module GPIx port input signals

Interrupt Status Register (IRQ_STATUS)

The Interrupt Status Register holds information on interrupt events that have occurred. The register is read-only and the IRQ_ACK register should be used to clear individual interrupts.

Table 25: **Interrupt Status Register (IRQ_STATUS)**

Reserved		INTC_Interrupt		Reserved		Internal Interrupts	
31	C_INTC_EXT_INTR+16	C_INTC_EXT_INTR+15	16	15	11	10	0

Table 26: **Interrupt Status Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
31:[C_INTC_EXT_INTR + 16]	-	R	0	Reserved
[C_INTC_EXT_INTR+15]:16	INTC_Interrupt	R	0	I/O Module external interrupt input signal INTC_Interrupt [C_INTC_EXT_INTR-1:0] mapped to corresponding bit positions in IRQ_STATUS
15:11	-	R	0	Reserved
10	FIT4	R	0	FIT4 strobe
9	FIT3	R	0	FIT3 strobe
8	FIT2	R	0	FIT2 strobe
7	FIT1	R	0	FIT1 strobe
6	PIT4	R	0	PIT4 lapsed
5	PIT3	R	0	PIT3 lapsed
4	PIT2	R	0	PIT2 lapsed
3	PIT1	R	0	PIT1 lapsed
2	UART_RX	R	0	UART Received Data
1	UART_TX	R	0	UART Transmitted Data
0	UART_ERR	R	0	UART Error

Interrupt Pending Register (IRQ_PENDING)

The Interrupt Pending Register holds information on enabled interrupt events that have occurred. IRQ_PENDING is the contents of IRQ_STATUS bit-wised masked with the IRQ_ENABLE register. The register is read-only and the IRQ_ACK register should be used to clear individual interrupts.

Table 27: **Interrupt Pending Register (IRQ_PENDING)**

Reserved		INTC_Interrupt		Reserved		Internal Interrupts	
31	C_INTC_EXT_INTR+16	C_INTC_EXT_INTR+15	16	15	11	10	0

Table 28: **Interrupt Pending Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
31:[C_INTC_EXT_INTR+16]	-	R	0	Reserved
[C_INTC_EXT_INTR+15]:16	INTC_Interrupt	R	0	I/O Module external interrupt input signal INTC_Interrupt [C_INTC_EXT_INTR-1:0] mapped to corresponding bit positions in IRQ_STATUS
15:11	-	R	0	Reserved
10	FIT4	R	0	FIT4 strobe
9	FIT3	R	0	FIT3 strobe
8	FIT2	R	0	FIT2 strobe
7	FIT1	R	0	FIT1 strobe
6	PIT4	R	0	PIT4 lapsed
5	PIT3	R	0	PIT3 lapsed
4	PIT2	R	0	PIT2 lapsed
3	PIT1	R	0	PIT1 lapsed
2	UART_RX	R	0	UART Received Data
1	UART_TX	R	0	UART Transmitted Data
0	UART_ERR	R	0	UART Error

Interrupt Enable Register (IRQ_ENABLE)

The Interrupt Enable Register enables assertion of the I/O Module interrupt output signal INTC_IRQ by individual interrupt sources. The contents of this register is also used to mask the value of the IRQ_STATUS register when registering enabled interrupts in the IRQ_PENDING register.

Table 29: **Interrupt Enable Register (IRQ_ENABLE)**

Reserved	INTC_Interrupt	Reserved	Internal Interrupts
31 C_INTC_EXT_INTR+16	C_INTC_EXT_INTR+15 16	15 11	10 0

Table 30: **Interrupt Enable Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
31:[C_INTC_EXT_INTR+16]	-	-	0	Reserved
[C_INTC_EXT_INTR+15]:16	INTC_Interrupt	W	0	Enable I/O Module external interrupt input signal INTC_Interrupt(16-C_INTC_EXT_INTR)
15 - 11	-	-	0	Reserved
10	FIT4	W	0	FIT4 interrupt enabled
9	FIT3	W	0	FIT3 interrupt enabled
8	FIT2	W	0	FIT2 interrupt enabled
7	FIT1	W	0	FIT1 interrupt enabled
6	PIT4	W	0	PIT4 interrupt enabled
5	PIT3	W	0	PIT3 interrupt enabled
4	PIT2	W	0	PIT2 interrupt enabled
3	PIT1	W	0	PIT1 interrupt enabled
2	UART_RX	W	0	UART Received Data interrupt enabled
1	UART_TX	W	0	UART Transmitted Data interrupt enabled
0	UART_ERR	W	0	UART Error interrupt enabled

Interrupt Acknowledge Register (IRQ_ACK)

This register is used as a command register for clearing individual interrupts in IRQ_STATUS and IRQ_PENDING registers. All bits written '1' clear the corresponding bits in the IRQ_STATUS and IRQ_PENDING registers. The register is write-only.

Table 31: **Interrupt Acknowledge Register (IRQ_ACK)**

IRQ_ACK
31 0

Table 32: **Interrupt Acknowledge Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
31:0	IRQ_ACK	W	0	All bit position written with 1 will clear corresponding bits in both the IRQ_STATUS and the IRQ_PENDING registers

Interrupt Mode Register (IRQ_MODE)

This register is used to define which interrupts use fast interrupt mode. All bits written '1' use fast interrupt mode. The register is write-only.

The register is only implemented when fast interrupt mode is enabled, by setting C_INTC_HAS_FAST to 1.

Table 33: **Interrupt Mode Register (IRQ_MODE)**

IRQ_MODE	
31	0

Table 34: **Interrupt Mode Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
31:0	IRQ_MODE	W	0	All bit position written with 1 will use fast interrupt mode

Interrupt Address Vector Registers (IRQ_VECTOR_0 - IRQ_VECTOR_31)

These 32 registers are used as Interrupt Address Vector for the corresponding interrupt bit. The content is sent to the processor using an internal signal when the interrupt occurs. The registers are write-only.
The two least significant bits and the most significant bits of each register are fixed to 0. The range of most significant fixed bits depends on C_MEMSIZE: 12-31 for 4096, 13-31 for 8192, 14-31 for 16384, 15-31 for 32768, and 16-31 for 65536.
For reserved interrupt bits (11-15), and unused external interrupts (greater than C_INTC_EXT_INTR+15), writing to the corresponding register has no effect.

Table 35: **Interrupt Address Vector Register (IRQ_VECTOR_x)**

0		IRQ_VECTOR_x		0	
31	12-16	11-15	2	1	0

Table 36: **Interrupt Address Vector Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
31:0	IRQ_VECTOR	W	0x10	The Interrupt Address Vector for the corresponding interrupt.

PITx Preload Register (PITx_PRELOAD) (x = 1, 2, 3 or 4)

The value written to this register determines the period between two consecutive PITx_Interrupt events. The period will be the value written to the register + 2 count events.

The register is implemented if C_USE_PITx is 1.

Table 37: **PITx Preload Register (PITx_PRELOAD)**

Reserved		PITx_PRELOAD	
31	C_PITx_SIZE	C_PITx_SIZE-1	0

Table 38: **PITx Preload Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
31:C_PITx_SIZE	-	-	-	Reserved
[C_PITx_SIZE-1]:0	PITx_PRELOAD	W	0	Register holds the timer period

PITx Counter Register (PITx_COUNTER) (x = 1, 2, 3 or 4)

When reading this register the obtained data will be a sample of the current counter value.

The register is implemented if C_USE_PITx is 1 and C_PITx_READABLE is 1.

Table 39: **PITx Counter Register (PITx_COUNTER)**

Reserved		PITx_PRELOAD	
31	C_PITx_SIZE	C_PITx_SIZE-1	31

Table 40: **PITx Counter Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
31:C_PITx_SIZE	-	-	-	Reserved
[C_PITx_SIZE-1]:0	PITx_COUNTER	R	0	PITx counter value at time of read

PITx Control Register (PITx_CONTROL) (x=1, 2, 3 or 4)

The EN bit in this register enables/disables counting. The PRELOAD bit determines if the counting is continuous with automatic reload of the PITx_PRELOAD value when lapsing (PITx_COUNTER = 0) or if the counting is stopped after counting the number of cycles defined in PITx_PRELOAD.

The register is implemented if C_USE_PITx is 1.

Table 41: **PITx Control Register (PITx_CONTROL)**

Reserved		RELOAD	EN
31	2	1	0

Table 42: **PITx Control Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
31:2	-	-	0	Reserved
1	PRELOAD	W	0	0 = Counter counts PITx_PRELOAD value cycles and then stops 1 = Counter value is automatically reloaded with the PITx_PRELOAD value when counter lapses
0	EN	W	0	0 = Counting Disabled 1 = Counter Enabled

Design Implementation

Design Tools

See the Tool Flow chapter.

Target Technology

The target technology is an FPGA listed in the Supported Device Family(1) field of the LogiCORE Facts table.

Device Utilization and Performance Benchmarks

Because the MicroBlaze MCS is a module that is used together with other parts of the design in the FPGA, the utilization and timing numbers reported in this section are just estimates, and the actual utilization of FPGA resources and timing of the MicroBlaze MCS design will vary from the results reported here. All parameters not given in the table below have their default values.

Table 43: **Performance and Resource Utilization Benchmarks on Virtex-6 (xc6vlx240t-1-ff1156)**

Parameter Values (other parameters at default value)														Device Resources	
C_USE_UART_RX	C_USE_UART_TX	C_INTC_USE_EXT_INTR	C_INTC_INTR_SIZE	C_USE_FIT1	C_FIT1_No_CLOCKS	C_USE_PIT1	C_PIT1_SIZE	C_USE_GPI1	C_GPI1_SIZE	C_USE_GPO1	C_GPO1_SIZE	C_USE_IO_BUS	C_DEBUG_ENABLE	LUTs	Flip-Flops
1	1	0	0	0	0	0	0	0	0	0	0	0	0	546	276
1	1	1	5	0	0	0	0	0	0	0	0	0	0	606	340
1	1	1	5	1	65000	0	0	0	0	0	0	0	0	620	353
1	1	1	5	1	65000	1	32	0	0	0	0	0	0	656	441
1	1	1	5	1	65000	1	32	1	32	0	0	0	0	658	473
1	1	1	5	1	65000	1	32	1	32	1	32	0	0	659	505
1	1	1	5	1	65000	1	32	1	32	1	32	1	0	675	610
1	1	1	5	1	65000	1	32	1	32	1	32	1	1	882	946

Support

Xilinx provides technical support for this LogiCORE product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

Ordering Information

The MicroBlaze MCS core is provided under the Xilinx End User License Agreement and can be generated using the Xilinx CORE Generator™ system. The CORE Generator system is shipped with the Xilinx ISE Design Suite software.

The MicroBlaze MCS core does not require a License Key. Contact your local Xilinx sales representative for pricing and availability of additional Xilinx LogiCORE IP modules and software. Information about additional Xilinx LogiCORE IP modules is available on the Xilinx IP Center.

Reference Documents

The following reference documents are available online:

1. MicroBlaze Processor Reference Guide (UG081)
2. Xilinx SDK Help
3. Xilinx ISE Manuals
4. Xilinx PlanAhead Manuals
5. 7 Series FPGAs Configuration User Guide (UG470)

Additional Resources

The following additional resources are available online:

- The entire set of GNU manuals:
  www.gnu.org/manual
- Xilinx Data Sheets:
  www.xilinx.com/support/documentation/data_sheets.htm
- Xilinx Problem Solvers:
  www.xilinx.com/support/troubleshoot.htm
- Additional Xilinx Documentation:
  www.xilinx.com/support/library.htm
- Xilinx Glossary:
  www.xilinx.com/company/terms.htm
- Xilinx Support:
  www.xilinx.com/support

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
01/18/12	1.0	Initial release.
04/24/12	1.1	Updated for 14.1, adding support for low-latency interrupt.

Notice of Disclaimer

Automotive Applications Disclaimer