



XAPP1180 (v1.1) July 28, 2015

Reference System: Kintex-7 MicroBlaze System Simulation Using IP Integrator

James Lucero

Summary

This reference system demonstrates the functionality of a MicroBlaze™ processor system on the Kintex® -7 device architecture using IP Integrator in simulation and in hardware. This system includes common peripherals like main memory, as well as RS232, which are critical in system design.

Several stand-alone software applications are provided to verify functionality of the peripherals in the reference system. Applications include `hello_uart` and `hello_mem`.

This application note describes how to set up the simulation environment for the system, execute the simulation using either Vivado® simulator or ModelSim® environments, and run the design on hardware. This application note describes running the design in simulation and hardware, targeting the KC705 board that contains the Kintex-7 XC7K410TFFG900-2 FPGA.

Reference Design

You can download the [Reference Design Files](#) for this application note from the Xilinx® website. The following directory structure is used after unzipping the file on the local machine:

```
mb_dds_simulation/
```

<code>project_1.sdk/</code>	Contains the software projects for the design
<code>project_1.srscs/</code>	Contains files for the embedded design
<code>sim/</code>	Contains files necessary for simulation
<code>ready_for_download/</code>	Contains the precompiled bitstream and software applications for this application note.

[Table 1](#) shows the reference design matrix.

Table 1: Reference Design Matrix

Parameter	Description
General	
Developer name	James Lucero
Target devices	Kintex-7 FPGAs MicroBlaze Processor
Source code provided	Yes

Table 1: Reference Design Matrix (Cont'd)

Parameter	Description
Source code format	Verilog
Design uses code and IP from existing Xilinx application note and reference designs or third party	No
Software Requirements	
Vivado Design Suite	System Edition 2015.2
SDK	Version 2015.2
Simulation	
Functional simulation performed	Yes
Timing simulation performed	No
Test bench used for functional and timing simulations	Yes
Test bench format	Verilog
Simulator software/version used	ModelSim SE/QuartaSim [®] Advanced Simulator 10.3d with VHDL and Verilog co-simulation capability or Vivado simulator
SPICE/IBIS simulations	N/A
Implementation	
Synthesis software tools/versions used	Vivado synthesis
Implementation software tools/versions used	Vivado implementation
Static timing analysis performed	Yes
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	KC705 evaluation board (revision C or D)
Board connection requirements	Two USB Type-A to Mini-B 5-pin cables

Hardware System Specifics

The high performance portion of the design uses:

- MicroBlaze processor at 200 MHz
- AXI Interconnect instance at 200 MHz
- AXI_7SERIES_DDRx memory controller at 800 MHz with a DDR3 data width of 64 bits

The lower performance portion of the design uses:

- AXI Peripheral IP (AXI4LITE) at 100 MHz
- AXI Interconnect instance at 100 MHz

The reference system includes:

- MicroBlaze
- AXI_INTERCONNECT
- MIG_7SERIES
- AXI_GPIO
- AXI_UART16550
- LMB BRAM

Refer to [Figure 1](#) for the block diagram and [Table 2](#) for the address map of the system.

Concepts and Setup

The sections below provide technical explanations and setup instructions for simulation and hardware generation of a MicroBlaze processor system with a Kintex-7 device using an IP Integrator design.

Block Diagram

[Figure 1](#) shows the reference system block diagram.

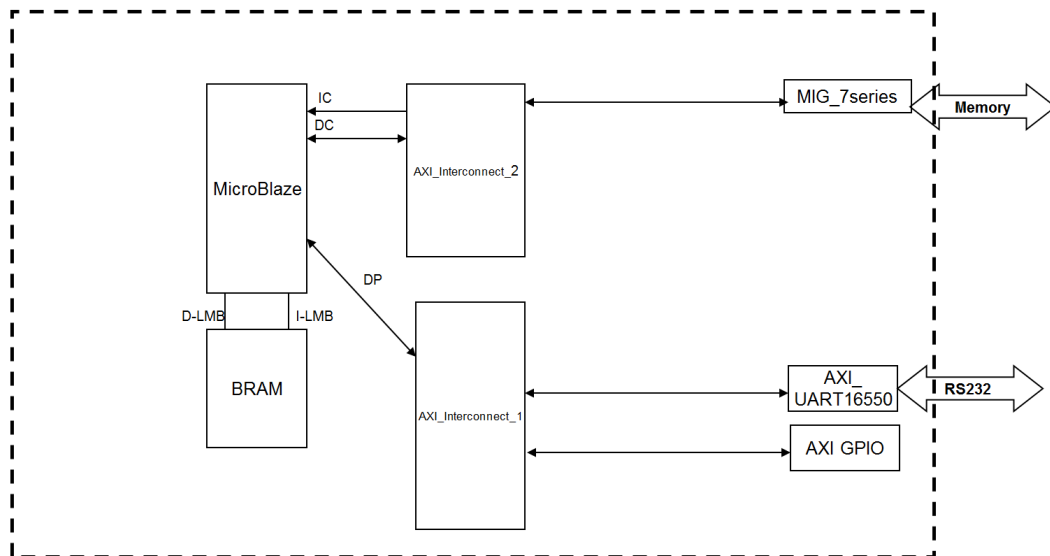


Figure 1: Reference System Block Diagram

Address Map

Table 2 displays the reference system address map.

Table 2: Reference System Address Map

Peripheral	Instance	Base Address	High Address
mig_7series	mig_7series_1	0x80000000	0xBFFFFFFF
axi_uart16550	axi_uart16550_1	0x40400000	0x4040FFFF
axi_gpio	axi_gpio_1	0x42000000	0x4200FFFF
lmb_bram_if_cntlr	lmb_bram_if_cntlr_1	0x00000000	0x0000FFFF
lmb_bram_if_cntlr	lmb_bram_if_cntlr_2	0x00000000	0x0000FFFF

Software Applications

Processor instructions are run in either Local Memory Bus (LMB) Block RAM (BRAM) or MIG_7SERIES by the linker script in the SDK project.

Hello Uart Software Application

This software application tests functionality of the AXI UART16550. It prints `Hello World!` to the simulated RS232 terminal where the output is seen in the TCL Console in Vivado or the Transcript window in ModelSim.

This software application is run out of DDR3 external memory using the MIG_7SERIES memory controller.

Hello Mem Software Application

This software application tests the functionality of the MIG_7SERIES memory controller by means of a memory test. The software application polls a GPIO register to determine when the memory controller is calibrated since the GPIO input is connected to the `init_done` signal from the memory controller. After the memory is calibrated, a short memory test (XIL_TESTMEM_INCREMENT with 32-words in a test pattern) is executed on the memory controller. The `PASSED!` or `FAILED!` message appears on the simulation RS232 terminal based upon results of the memory test.

This software application is run out of internal memory using LMB BRAM.

Setting the Simulation Environment for ModelSim/Quarta Simulator

Compiling Libraries

1. Ensure that the `MODELSIM` (for Linux) or `MODELTECH` (for Windows) variables point to a valid `modelsim.ini` file that is writable.
2. Open Vivado.
3. In the Tcl Console, type in the following command to compile all Xilinx libraries except EDK libraries.

```
compile_simlib -simulator modelsim -library unisim -library simprim -library  
xilinxcorelib -language all -directory <PATH_TO_COMPILED_LIBRARIES>
```

Note: Compile time depends on the local machine.

Within any Vivado Design Suite project, the compiled libraries directory points to the Compiled library directory in the Simulation settings which is discussed later in this application note.

Simulation Directories and Files

One simulation directory is used for both Linux and Windows environment. The directory is `sim/` located in the main project directory.

Scripts in this area can convert an ELF file when the linker script is set for DDR address space into files that load up memory models with processor instructions. During this process, Data2MEM is used to generate the initial MEM files and Perl scripts are used to modify MEM files into a format that loads the memory models properly with processor instructions. The simulation testbench loads the MEM files into the memory when the memory controller is calibrated.

Modifying the Testbench/Scripts for Different Sized DDR3 Components

Each MEM file is associated with one memory model, which is 16 bits (for example, 4x16-bit memory models are necessary for a 64-bit DDR3).

The testbench (`system_tb.v`) is modified depending the amount of memory models needed for the DDR3 component (1x for 16-bits, 2x for 32-bits, 4x for 64 bits).

The `gen_memfiles.sh/gen_memfiles.bat` commands are modified based on the MEM files used (1x `sim/ddr3_0.mem`, 2x `sim/ddr3_0.mem` `sim/ddr3_1.mem`, 4x `sim/ddr3_0.mem` `sim/ddr3_1.mem` `sim/ddr3_2.mem` `sim/ddr3_3.mem`).

The `memory_init.bmm` file is modified based upon the address space of the memory, width of the memory, and the MEM files used.

A 16-bit DDR3 example for these modifications is located in the `sim/example_16` directory.

Modifying the Testbench for Different UART Frequency

Instructions to modify the UART core clock frequency/AXI frequency are located in the comments in the testbench (`system_tb.v`).

Simulation Files

The `sim/` directory contains the following pertinent files and directories:

- `512Mb_ddr3/` - Micron DDR3 memory model.
- `gen_memfiles.sh` - Linux main script to convert ELF to MEM files.
- `gen_memfiles.bat` - Windows main script to convert ELF to MEM files.
- `memory_init.bmm` - Block Memory Map file used by `data2mem` to generate the initial MEM files from an ELF file.
- `ddr3_x.mem` - Memory files to load external memory models. Provided files are using `hello_uart` external memory files.
- `system_tb.v` - Verilog testbench for the design.
- `uart_rcvr.v` - For simulated RS232 terminal.
- `uart_rcvr_wrapper.v` - For simulated RS232 terminal.

Note: Output from the software application is monitored inside the ModelSim/Quarta Simulator terminal or Vivado TCL console while the simulation is running.

Executing the Simulation

Executing the Simulation from IP Integrator

To generate and run the system simulation from IP Integrator, follow these steps:

Opening Project/Setting Simulation Mode and Generating Design Files

1. Open `<unzip dir>/mb_ddr_simulation/project_1.xpr` inside Vivado.
Note: By default, Vivado simulator is selected for the Target Simulator.
2. In Flow Navigator, expand **IP Integrator** and select **Open Block Design**.
3. Select `design_1.bd`, if necessary. The block diagram is shown in [Figure 2](#).

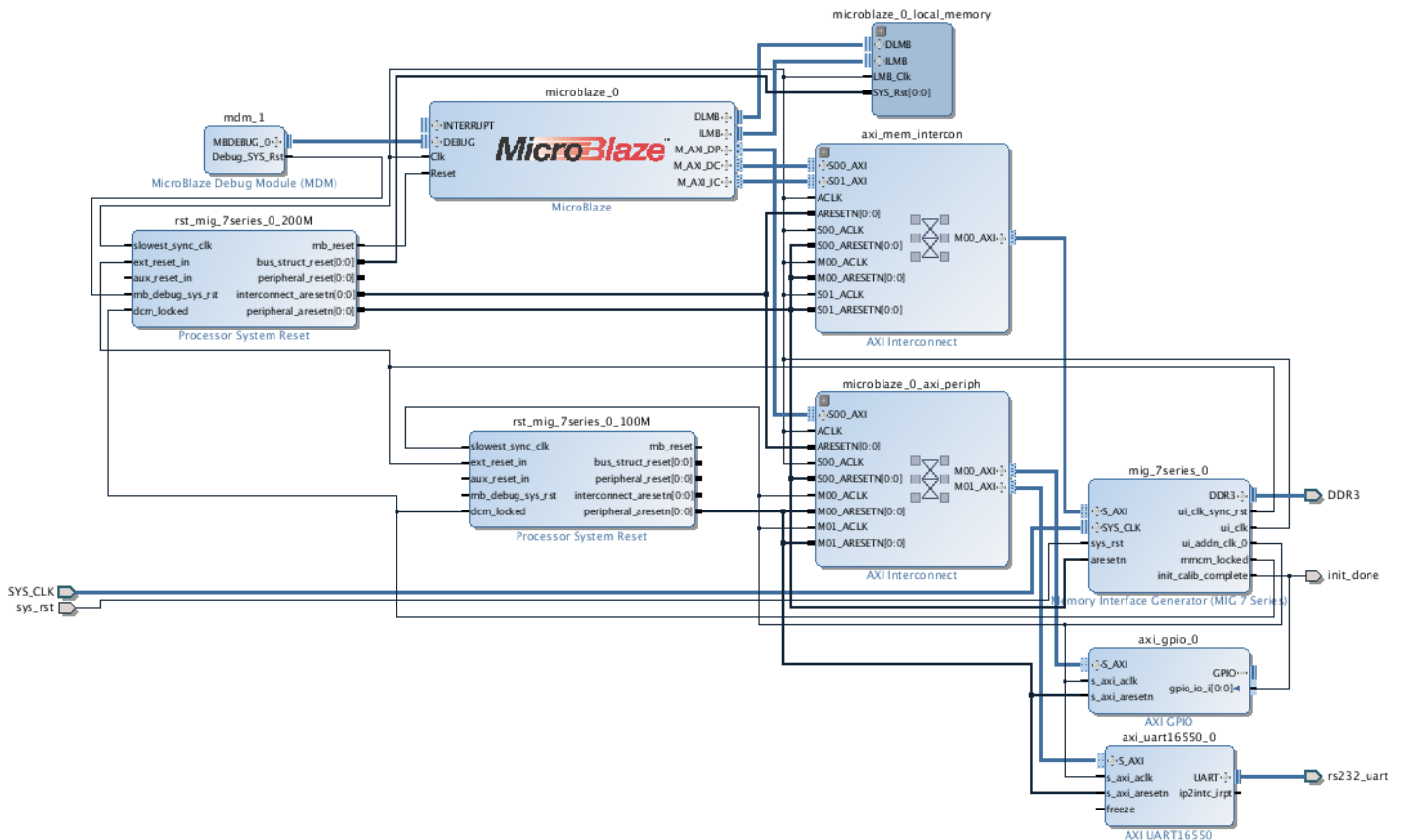


Figure 2: IP Integrator Block Diagram

4. In the Sources tab, expand `design_1_wrapper`.
5. Right-click `design_1_i` and click on **Generate Output Products**.
6. Click **Generate** to generate the design files and wrappers.
7. ModelSim/QuartaSim users only:
 - a. In the Flow Navigator, expand **Simulation** and click on **Simulation Settings**.
The Project Settings > Simulation tab opens.
 - b. Click **Target simulator** and select **ModelSim Simulator** or **Quarta Advanced Simulator** then click **YES** in the **Target Simulator** dialog box.
 - c. For **Compiled library location**: select the location of `<PATH_TO_COMPILED_LIBRARIES>`.
 - d. Click the **Simulation** tab and ensure **-novopt** is included for **vsim.more_options**.
8. Select **Project Manager** in the Flow Navigator tab.
9. Under Sources, expand the **Simulation Sources**, and then expand `sim_1 > system_tb > design_1_wrapper`.
10. Right-click `design_1_i` and select **Associate ELF Files**. For `microblaze_0` under Simulation Sources, click the **Browse** button under the Associated ELF File column.

11. Select the ELF file in the project that will be used for simulation.
12. If code exists in the DDR3 memory space, follow the steps in the following sections: [Generating External Memory Files For Windows](#) or [Generating External Memory Files For Linux](#).

Note: To add another ELF file to the project, in Project Manager select **Add Sources > Add or Create Simulation Sources > Add Files** and select the ELF file(s) to add. By default, the `hello_uart` application is selected.

Generating External Memory Files For Windows

1. Open a Command Prompt. Run `settings32.bat` or `settings64.bat` from the Xilinx tool install area. Typical install area is `C:\Xilinx\Vivado\2015.2`.

2. Set the following path in same Command Prompt:

Note: Paths and version of Windows depends on the local installation which can be different than below. An example setup file for this path is in `<unzip_dir>/mb_ddr_simulation/path64.bat` or `<unzip_dir>/mb_ddr_simulation/path32.bat`.

```
set PATH=C:\Xilinx\Vivado\2015.2\bin;C:\Xilinx\Vivado\2015.2\lib\win64.o;
C:\Xilinx\Vivado\2015.2\tps\win64\jre\bin\server;C:\Xilinx\Vivado\2015.2\
tps\win64\jre\bin;C:\Xilinx\SDK\2015.2\bin\nt64;C:\Xilinx\Vivado\2015.2\
ids_lite\EDK\bin\nt64;C:\Xilinx\Vivado\2015.2\ids_lite\EDK\lib\nt64;C:\
Xilinx\Vivado\2015.2\ids_lite\ISE\bin\nt64;C:\Xilinx\Vivado\2015.2\ids_
lite\ISE\lib\nt64;C:\Xilinx\SDK\2015.2\gnuwin\bin;%PATH%
```

3. In the same Command Prompt window, change directories into the main Vivado project directory `<unzip_dir>\mb_ddr_simulation\`
4. Run the following command:

```
sim\gen_memfiles.bat <location of ELF file>
```

Note: This will take about 20 seconds to run and a few commands windows will open and close. The memory files (`.mem`) are found in the `sim/` directory.

Generating External Memory Files For Linux

1. Ensure the current directory in the Vivado project is `mb_ddr_simulation/`.
2. Ensure that the `sim/` directory has executable permissions:

```
exec sim/gen_memfiles.sh <location of ELF file>
```

Note: The memory files (`.mem`) are located in the `sim/` directory.

Running the Simulation

1. Launch the simulator with **Simulation > Run Simulation > Run Behavioral Simulation** in the Flow Navigator tab.
2. Once Vivado simulator or Modelsim/Quarta is done compiling the design, add the required signals to the design and execute the run command with the specified run time for the selected software application. For example, `run 200us`.

Running the Software Applications

Running the Hello Uart Software Application

This simulation should run for ~271 μ s. Vivado simulator run times can be around 1 to 2 hours.

The AXI UART16550 starts transmitting output to the simulated RS232 terminal and/or the ModelSim transcript window or Vivado simulator Tcl console. The output is shown below.

```

UART    0 Transmitted Out Char = 0x48 = "H" @ Time = 169215000.0 ps
UART    0 Transmitted Out Char = 0x65 = "e" @ Time = 178815000.0 ps
UART    0 Transmitted Out Char = 0x6c = "l" @ Time = 187935000.0 ps
UART    0 Transmitted Out Char = 0x6c = "l" @ Time = 197055000.0 ps
UART    0 Transmitted Out Char = 0x6f = "o" @ Time = 206175000.0 ps
UART    0 Transmitted Out Char = 0x20 = " " @ Time = 215295000.0 ps
UART    0 Transmitted Out Char = 0x55 = "U" @ Time = 224415000.0 ps
UART    0 Transmitted Out Char = 0x61 = "a" @ Time = 233535000.0 ps
UART    0 Transmitted Out Char = 0x72 = "r" @ Time = 243135000.0 ps
UART    0 Transmitted Out Char = 0x74 = "t" @ Time = 252255000.0 ps
UART    0 Transmitted Out Char = 0x0d = <special char> @ Time = 261375000.0 ps
UART    0 Transmitted Out Char = 0x0a = <special char> @ Time = 270495000.0 ps

```

Simulating the Hello Mem Software Application

This simulation should run for ~140 μ s. Vivado simulator runtime is around 1 hour.

The output is seen in the ModelSim/Quarta transcript window or Vivado simulator Tcl Console after the memory test is completed successfully. The output is shown below.

```

UART    0 Transmitted Out Char = 0x50 = "P" @ Time = 95865000.0 ps
UART    0 Transmitted Out Char = 0x41 = "A" @ Time = 100665000.0 ps
UART    0 Transmitted Out Char = 0x53 = "S" @ Time = 105465000.0 ps
UART    0 Transmitted Out Char = 0x53 = "S" @ Time = 110265000.0 ps
UART    0 Transmitted Out Char = 0x45 = "E" @ Time = 115065000.0 ps
UART    0 Transmitted Out Char = 0x44 = "D" @ Time = 119865000.0 ps
UART    0 Transmitted Out Char = 0x21 = "!" @ Time = 124665000.0 ps
UART    0 Transmitted Out Char = 0x0d = <special char> @ Time = 129465000.0 ps
UART    0 Transmitted Out Char = 0x0a = <special char> @ Time = 134265000.0 ps

```

Executing the System in Hardware

This section provides instructions to execute the reference design in hardware.

1. Connect a USB cable from the host PC to the USB JTAG port. Ensure the appropriate device drivers are installed.
2. Connect a second USB cable from the host PC to the USB UART port. Ensure that the USB UART drivers have been installed.
3. Set power ON.
4. Start a terminal program (such as HyperTerminal) on the host PC with these settings:
 - Baud Rate: 9600
 - Data Bits: 8
 - Parity: None
 - Stop Bits: 1
 - Flow Control: None

Executing the Reference System Using the Pre-Built Bitstream and the Compiled Software Application

These are the steps to execute the system using files in the `ready_for_download/` directory of the `<unzip_dir>/mb_ddr_simulation/` directory:

1. Invoke Xilinx Microprocessor Debugger (XMD).
2. Change directories to the `ready_for_download` directory.
3. Download the bitstream inside XMD:

```
XMD% fpga -f design_1_wrapper.bit
```

4. Connect to the processor inside XMD:

```
XMD% connect mb mdm
```

5. Download the processor code (ELF) file:

```
XMD% dow hello_uart.elf
```

or:

```
XMD% dow hello_mem.elf
```

6. Run the software:

```
XMD% run
```

Either `Hello Uart` or `PASSED!` displays on the HyperTerminal screen.

Building the Hardware

This section covers rebuilding the hardware design with Vivado.

Note: The generated bitstream is located at:

```
<unzip_dir>/mb_ddr_simulation/project_1.runs/impl_1/design_1_wrapper.bit
```

1. Open `<unzip_dir>/mb_ddr_simulation/project_1.xpr` inside Vivado.
2. In Flow Navigator:
 - a. Expand **IP Integrator** and select **Open Block Design**.
 - b. Select `design_1.bd`, if necessary.
3. In the Sources tab:
 - a. Expand `design_1_wrapper`.
 - b. Right-click `design_1_i` and click **Generate Output Products**.
 - c. Click **Generate** to generate the design files and wrappers.
4. In the Flow Navigator tab, click **Project Manager**.
5. In the Log window, select the **Design Runs** tab.
6. Right-click `impl_1`, select **Launch Run**, and click **OK** in the Launch Selected Runs dialog box.
7. Click **OK** on the Missing Synthesis Results box to generate the netlist.

Note: This step takes between 30 mins - 60 mins depending on the machine.
8. When completed, click **Generate Bitstream** in the Implementation Completed dialog box.

Note: Make sure the implementation met timing, especially if the design was modified.
9. Click **File > Export > Export Hardware**. Check **Include bitstream** and click **OK**.

Compiling Software and Running Design Through the SDK Tool

This section describes compiling software with the SDK tool and then running the hardware and software through SDK.

Compiling Software with the SDK Tool

1. Start SDK. In Linux, type in `xsdk` to start SDK.
2. In the Workspace Launcher, select the following **Workspace:**

```
<unzip_dir>\project_1.sdk\SDK_Workspace
```
3. Click **OK**.

Note: Steps 4 through 7 are only needed with an empty SDK_Workspace.

4. The BSP, hardware platform, and software applications need to be imported. Select **File > Import > General > Existing Projects into Workspace**.
5. Click **Next**.
6. Browse to the `<unzip_dir>\project_1.sdk` and click **OK**.
7. Make sure that all check boxes are selected, including `standalone_bsp_0`, `hw_platform_0`, and associated software applications are selected.
8. Click **Finish**.

Note: At this point the BSP and software applications are compiled. This should take 2-5 minutes. If an error message displays about any other software application, right-click the software application project in the Project Explorer tab, select **Change Referenced BSP**, and click `standalone_bsp_0`. Select **OK**.

9. At this point, you can modify existing software applications and create new ones in SDK.

Note: By default, the included software applications are set for `-DSIM=1`. You can change this option by right-clicking on the software application **C/C++ Build Settings > MicroBlaze gcc compiler > Miscellaneous**. For running software in hardware, remove this option.

Running the Hardware and Software Through the SDK Tool

1. With the SDK_Workspace project opened, Click **Xilinx Tools > Program FPGA**.
2. Click **Program**.
3. In the Project Explorer window, right-click (**Software_Application_Name**) > **Run As > Launch on Hardware**.

Modifying the System

To modify the system:

1. Connect the instances as follows:

Instances	Connect to
AXI masters/slaves instances with AXI4Lite protocol	microblaze_0_axi_periph
AXI masters/slaves instances with AXI4 protocol	axi_mem_intercon

2. After the block has been modified and saved, generate a new SDK_Export directory for the existing SDK_Workspace and regenerate the design.
3. In Sources in the Project Manager tab, expand `design_1_wrapper`.

Note: If a new bitstream must be generated, refer to [Building the Hardware, page 11](#).
4. Right-click `design_1_i` and click **Generate Output Products**.
5. Click **Generate** to generate the design files and wrappers.

6. Click **File > Export > Export Hardware**. Check **Include bitstream** and click **OK**.
7. Click **YES** in the Module Already Exported dialog box.
8. In the SDK project, right-click **hw_platform_0** and select **Change Hardware Platform Specification**.
9. Click **Yes** on the warning dialog box that opens.
10. Provide a link to the newly created `design_1_wrapper.hdf`, which is in `<unzip_dir>\project_1.sdk\`.
11. Click **OK**.
12. Rerun the Executing the Simulation section if needed.

References

The design files for this document are located at <https://secure.xilinx.com/webreg/clickthrough.do?cid=344919>.

The following documents provide additional information:

- [UG081](#), *MicroBlaze Processor Reference Guide*
- [UG900](#), *Vivado Design Suite User Guide: Logic Simulation*
- [UG994](#), *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator*
- [UG898](#), *Vivado Design Suite User Guide: Embedded Hardware Design*
- [UG810](#), *KC705 Evaluation Board for the Kintex-7 FPGA User Guide*

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
07/28/2015	1.1	Updated reference design for 2015.2. Made corresponding revisions to instructions in this application note.
10/02/2013	1.0	Initial Xilinx release.

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>.

© Copyright 2013-2015 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.