XILINX®

XAPP192 (v1.0) December 15, 2000

Interfacing a Virtex-E Device to a MIPS Processor

Summary

This application note describes a reference design for a Virtex™-E FPGA interface to a MIPS processor. The interface connections are shown while discussing techniques for running the design at the fastest data throughput speed available from a MIPS processor.

Design Discussion

General

The design was synthesized using Synplify Pro 6.0.0 and implemented using Xilinx Alliance Software Version 3.1i SP1.

Presently, the MIPS I/O clock speed is the limiting factor for I/O throughput. The maximum I/O speed for a MIPS device (QED RM7000) is currently 125 MHz, half the speed of the internal clock. With the release of QED RM 7000A, the maximum MIPS I/O speed will increase to 133 MHz, thereby increasing the maximum system speed to the same level. The top system interface speed for this design could therefore be as high as 133 MHz.

This reference design assumes a direct interface between a Virtex-E BG432 –8 FPGA, and a 250 MHz QED RM7000 MIPS processor (http://www.qedinc.com). Figure 1 shows the block/flow diagram used and the design interface connections.

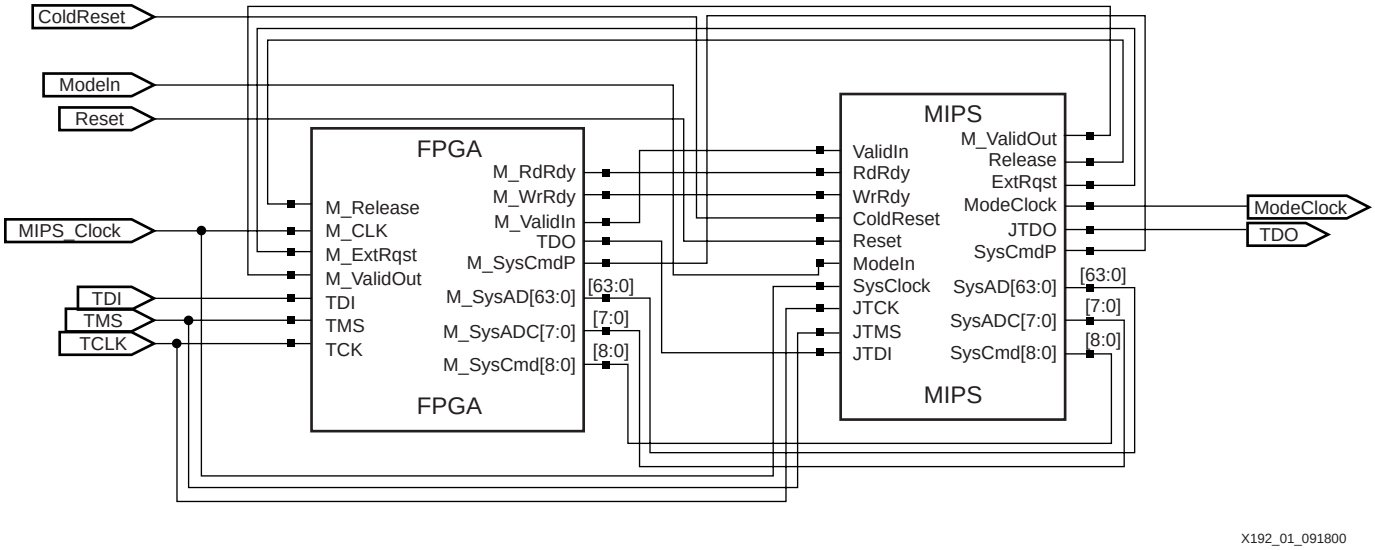


Figure 1: **Virtex-E FPGA to MIPS Processor Interface**

Design Hierarchy

The design utilizes an architecture similar to that of the PCI core from Xilinx. From the top level, the design breaks into two parts: the user application and an example MIPS interface (see Figure 2). From there, the MIPS interface design is again broken into smaller modules (see Figure 3). Both illustrations appear on the next page.

© 2000 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and disclaimers are as listed at http://www.xilinx.com/legal.htm.
All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

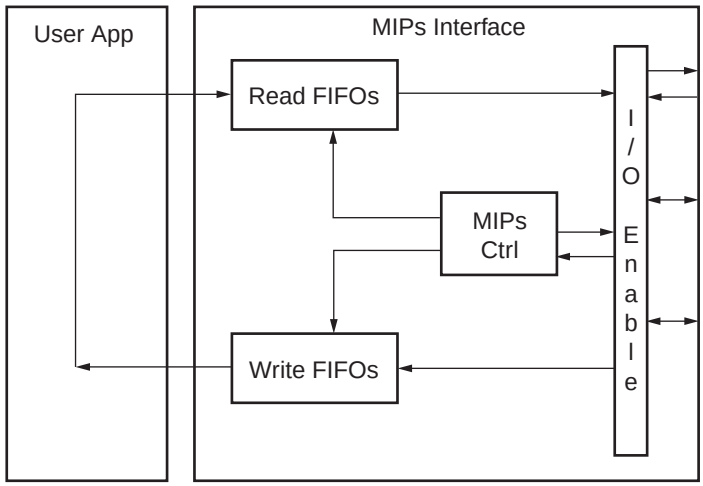


Figure 2: **Top-Level Layout of the MIPS Interface Design**

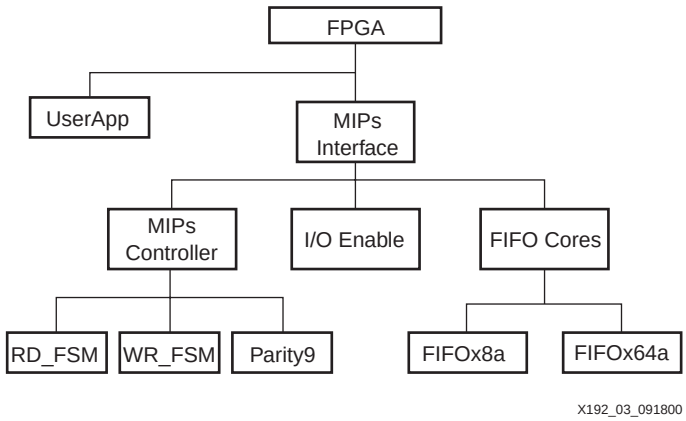


Figure 3: **Interface Functional Flow Hierarchy**

MIPS Interface

The MIPS interface is the top-level module for a self-contained design that allows interfacing to the MIPS processor. The MIPS interface uses FIFOs to control the flow of data between the user application and the MIPS processor.

MIPS Controller

The MIPS controller is where all the control signals for the MIPS interface are generated.

RD_FSM

RD_FSM is a state machine that monitors the system commands. If a read command is on the system command bus, it generates the appropriate control signals in the proper sequence.

WR_FSM

WR_FSM is a state machine that monitors the system commands. If a write command is on the system command bus, it generates the appropriate control signals in the proper sequence.

Parity9

Parity9 is a module that performs a simple parity check on data, and provides an error signal if there is a parity error.

I/O Enable

I/O Enable is a module in the MIPS interface that is the gatekeeper of all inputs and outputs to the MIPS interface module. This provides a central location for controlling the I/O signals and their output enables, as well as adding latency delays.

FIFO Cores

FIFO Cores is the module that contains all of the CoreGen-generated FIFO cores that are used in the design.

FIFOx8a

FIFOx8a is a 64 bit deep by 8 bit wide Asynchronous FIFO generated by CoreGen.

FIFOx64a

FIFOx64a is a 64 bit deep by 64 bit wide Asynchronous FIFO generated by CoreGen.

Async_FIFO

Aysnc_FIFO (not shown in hierarchy) is a module produced by CoreGen and used by both FIFOx64a and FIFOx8a.

User_App

User_App is where the user places the design.

S_Ctrl

S_Ctrl (not shown in hierarchy) is an example module used in this design inside the User_App.

S_Interface

S-Interface (not shown in hierarchy) is an example module used in this design inside the User_App.

State Machines

RD_FSM

Figure 4 shows the Finite State Machine used in RD_FSM.

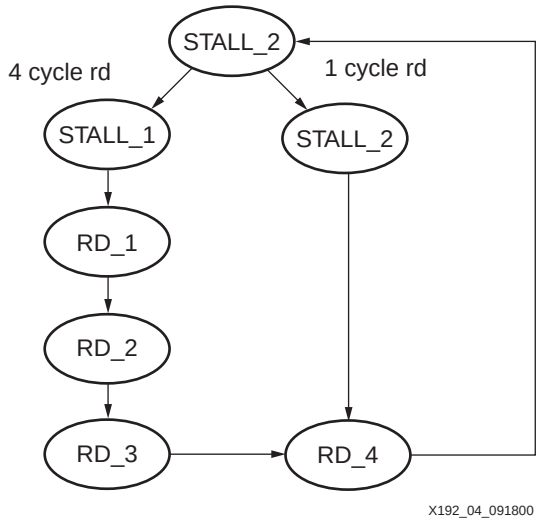


Figure 4: **Read Command Finite State Machine**

WR_FSM

Figure 5 shows the Finite State Machine used in WR_FSM.

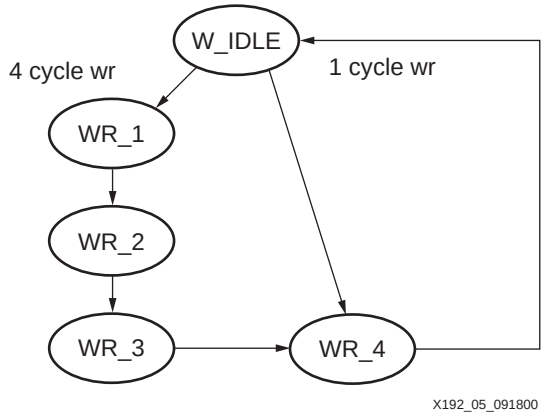


Figure 5: **Write Finite State Machine**

Achieving Speed

The following techniques were used to achieve the maximum system clock speed in the design.

Delay-Locked Loops (DLL)

A DLL is used to eliminate clock delay amongst registers communicating with I/O. This gives the user more of the clock period to utilize because there is less clock skew.

A DLL is used in the design by inferring a BUFGDLL, by using the following attribute in Synplify.

```
/* synthesis xc_clockbuftype = "BUFGDLL" */
```

For more detailed information, refer to **XAPP 132**, ***Using the Virtex Delay-Locked Loop***. For more information on Synplify, refer to **http://www.synplicity.com**.

Registering I/Os

Registered I/Os are registers placed on the edge of the die as close to the I/O pad as possible. These registers are inside the IOB (Input/Output Block). By using I/O registers, one guarantees the shortest path between an I/O and a register.

To use the registered I/O with a device based on the Virtex architecture (Virtex, Spartan®-II, or Virtex-E), all of the flip-flops in the same IOB must use the same clock and reset signals. No logic is permitted between the flip flop and the I/O pad, because there is no logic in the IOB and all logic must be implemented via a Configuration Logic Block (CLB) outside of the IOB. To pull the registers into the IOBs, use the map option:

`-pr [ i | o | b ]`

where **`i`** = input, **`o`** = output, and **`b`** = both.

For more information regarding map options, refer to *Development System Reference Guide* at **http://toolbox.xilinx.com/docsan/3_1i**.

SelectI/O™ Resource

The SelectI/O resource allows one to specify the use of different I/O standards with Virtex-based families. For this design, to achieve the desired I/O throughput, the HSTL IV standard was used. HSTL IV matches the voltage requirements of the QED RM7000 MIPS processor and is also able to be run at system speeds exceeding 133 MHz, the top speed of QED RM7000A, which is not yet in production.

For this design, the following Synplify attributes were used on the top level to infer HSTL IV I/O:

```
/* synthesis xc_padtype = "IBUF_HSTL_IV" */
/* synthesis xc_padtype = "IOBUF_HSTL_IV" */
/* synthesis xc_padtype = "OBUF_HSTL_IV" */
```

For more information on how to infer I/O standards in Synplify, refer to Synplify's help menu or to **http://www.synplicity.com**.

For more information on using the SelectI/O resource refer to **XAPP133**, ***Using the Virtex SelectI/O Resource***.

Area Constraints and Pin Constraints

To aid the placer in achieving optimal speed, it helps to place area and pin constraints on your design. It also helps to make it more modular in nature and easier to integrate into a system.

For example all of the block RAM used for the asynchronous FIFOs is locked to the leftmost column. Area constrain the rest of the design to a location on the edge of the die, as close to the block RAM as possible.

In this case, locking the pins to locations close to the block RAM was not necessary, as the area constraint on the logic and the timing constraints pulled the pins around the block RAM and area constraint.

If pin constraints are required, lock the data/address lines to the horizontal side and the control and clock lines to the vertical side of the die, close to the block RAM. This utilizes the architecture of the FPGA. The CLBs have horizontal direct connects to the adjacent CLB, which is good for data paths and data manipulation. The FPGA also has long lines that run vertically along the columns of the device. These long lines are well suited for control signals to the data paths.

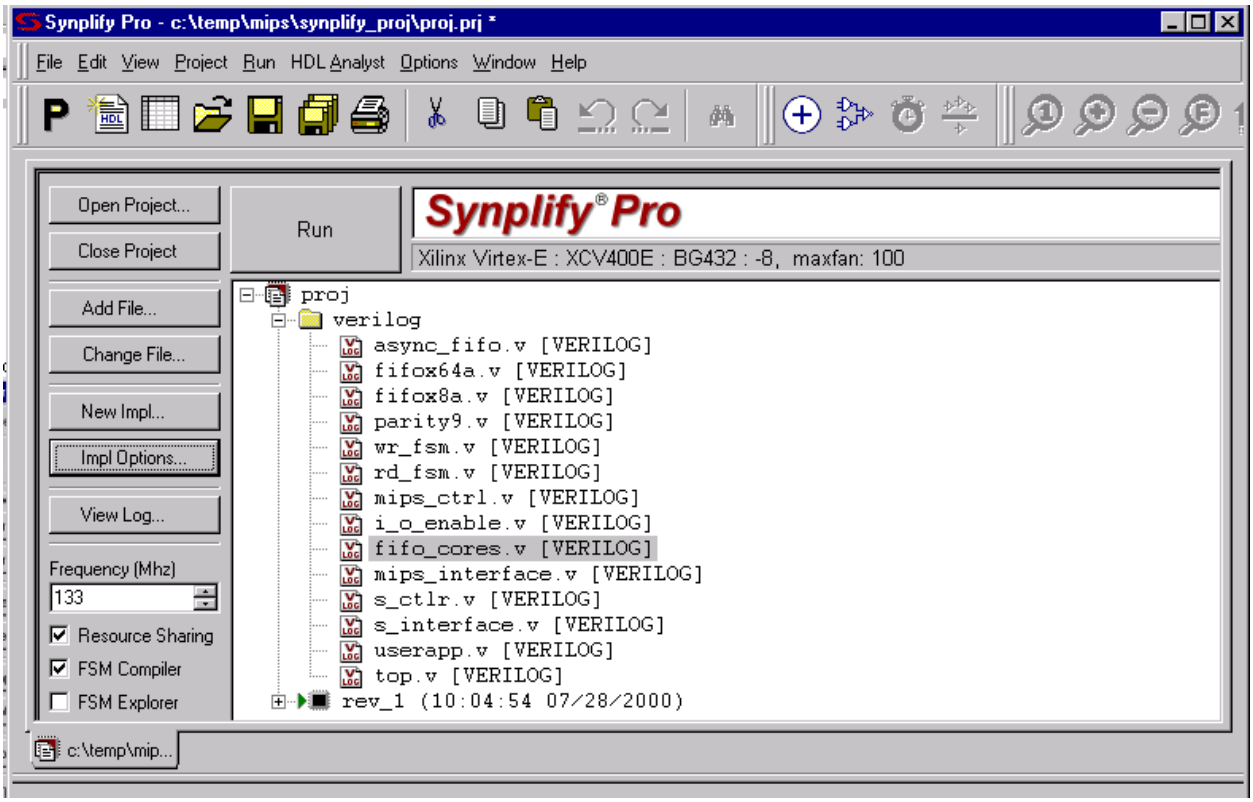
Synthesis Settings

In Synplify Pro 6.0.0, the frequency was set to 133 MHz to inform the tool to synthesize for speed. The implementation device was set for a Virtex-E design to allow the tool to use and instantiate architecture-specific features, like block RAM.

Files Synthesized

Listed below are the design files. Figure 6 is a screen capture of Synplify showing the file ordering that was used.

- **`top.v`**
- **`user_app.v`**
- **`s_ctrl.v`**
- **`s_interface.v`**
- **`mips_interface.v`**
- **`mips_ctrl.v`**
- **`rd_fsm.v`**
- **`wr_fsm.v`**
- **`parity9.v`**
- **`i_o_enable.v`**
- **`fifo_cores.v`**
- **`fifox64a.v`**
- **`fifox8a.v`**
- **`asynch_fifo.v`**



X192_06_100200

Figure 6: **Example Ordering of Files in Synplify**

Implementation Options

The two implementation options used to achieve system speeds of 133 MHz or greater on Virtex-E devices are:

`Map -pr b` (pack *both* input and output registers into the IOB)

`Par -ol 5` (overall Effort Level = 5)

Design Notes

1. The reference design **XAPP192.zip** is provided "as-is."
2. This interface is designed to demonstrate a high-speed interaction between the MIPS processor and an XCV400E –8 BG432. It does not cover command return latency. To compensate for read command latency, incorporate the I/O enable registers as part of the Read FIFO.
3. This design is a one-to-one connection, so it does not cover the NULL system command.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/15/00	1.0	Initial Xilinx release.