The TME Ports Package

Dave Galloway

Edwards S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto

January 2014

Introduction

The TME ports package allows a program running on a Linux or UNIX workstation to communicate with a user's circuit in an FPGA development board. The program may run on any machine, and does not have to run on the machine that is physically attached to the board.

The ports package currently supports Altera's DE-4 and DE-5 development boards. It communicates with the board over a PCIe bus.

View from the User's Circuit

A user circuit will have a number of wires sticking out of it, intended for communication with the outside world. These wires may be single 1-bit signals, or they may be multi-bit buses. For the purpose of this document, each named set of signals is called a port. For example, a circuit might have an 32-bit input port called `data_in`, a 32-bit output port called `result`, a 3-bit input port called `mode` and a 1-bit input port called `go`.

The user creates the appropriate input and output signals in the circuit and then declares a component called `pcie_portmux` that is connected to these signals. The component also needs some signals that allow it to talk to the PCIe bus, along with clock and reset signals. See the *An Example Application for the TME Package* document for details.

The ports support software will create the `pcie_portmux` component automatically. For details on the ports support software, see the *Creating and Running Circuits with the TME Ports Package* document. The `pcie_portmux` component acts as a wrapper circuit around the user's circuit that connects to the dangling input and output ports. The wrapper circuit will transfer data between the ports and the workstation. See Figure 1.

Current Implementation Restrictions

Bi-directional ports are not supported.

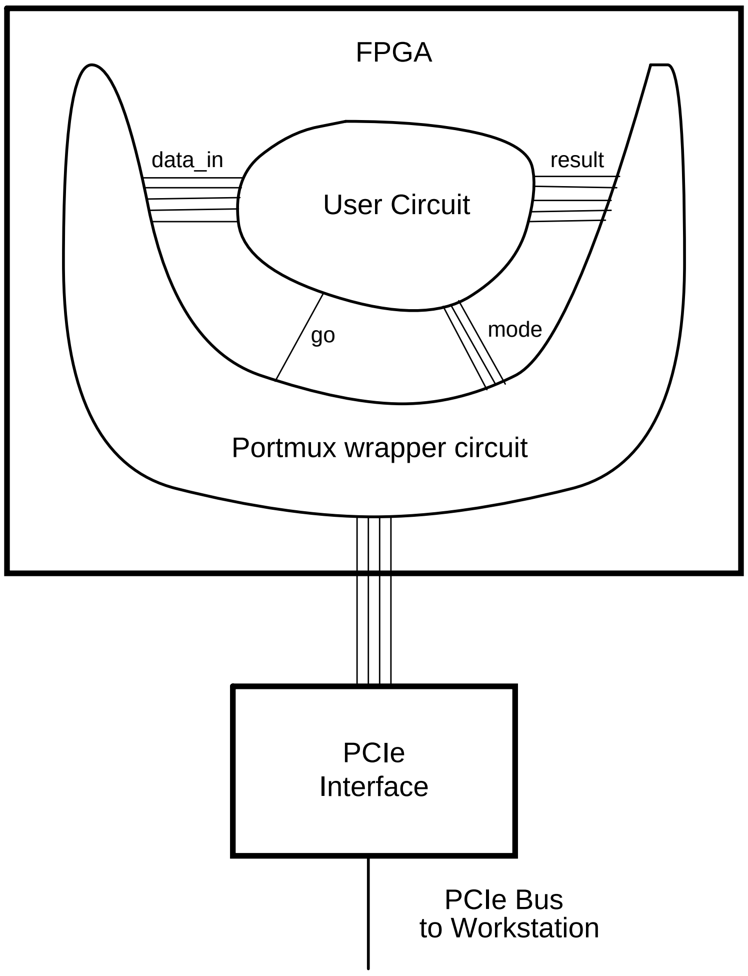

Figure 1: User’s circuit wrapped by portmux circuit

View from the User’s Program

A program on a workstation can communicate with a circuit in the board by calling a library of port routines. For example, this C code will set an 32-bit `data_in` port on the circuit to the value 42:

```
int temp = 42;

p = tm_open("data_in", "w");
tm_write(p, &temp, sizeof(temp));
```

All of the bits of the port will change at the same time, and the change will be synchronized with the circuit’s clock signal.

The available routines are:

```
tm_init(char *hostname)
```

initialize the package, and talk to the board that is physically attached to the given hostname. If hostname is an empty string, the value of the environment variable `TM_SERVER` will be used as the hostname. If that variable is not set, a default host will be contacted.

```
int tm_open(char *portname, char *mode)
```

open a particular port with a mode of "r" for reading (data flows from the circuit to the workstation) or "w" for writing (data flows from the workstation to the circuit). Returns an integer port descriptor to be given to the other routines in this package, or -1 on error.

```
int tm_write(int p, char *buf, int nbytes)
```

write nbytes of data starting from the given memory location to the given port. Returns the number of bytes written, or -1 on error.

```
int tm_read(int p, char *buf, int nbytes)
```

read nbytes of data from the given port into memory. Returns the number of bytes read, or -1 on error.

```
tm_close(int p)
```

close the given port.

The Port Description File

To use the ports package with a circuit, the user creates a port description file with the same name as the design file, but with a .ports extension. For example, if the circuit is called bitfilter.vhd, the port description file will be in the same directory, with the name bitfilter.ports.

The .ports file should describe the ports on the circuit. For example:

```
  Name    direction   bits   Handshake_from_circuit   Handshake_from_workstation

data_in   i             32   want_data                data_ready
result    o             32   result_ready             want_result
mode      i              3
go        i              1
```

Each line in the file describes a single port. The first three fields on the line are the name of the port, its direction (i for an input to the circuit, o for an output), and the number of bits in the port. The other two optional fields contain the names of 1-bit signals used by the circuit to provide flow control on the port. The first handshaking signal is an output from the circuit, the second is an input to the circuit.

Flow Control

If flow control signals are not specified for a port, then data will be transferred to or from the circuit port whenever the workstation asks for it. If the circuit can not take inputs as fast as the workstation can provide them, or might produce outputs faster than the workstation can take them, then flow control circuitry should be added to the circuit.

Flow Control for Output Ports

Flow control for an output port uses two 1-bit signal wires. The circuit sets the first signal to 1 when the value on the output port is stable. The portmux circuit will set the second signal to 1 when it is willing to receive the data from the output port. When both signals are 1, data is transferred from the circuit to the portmux. One data item is transferred in every clock cycle that has both signals set to 1.

Flow Control for Input Ports

Flow control for an input port also uses two 1-bit signal wires. The circuit sets the first signal to 1 when it is ready for more input. The portmux circuit sets the second signal to 1 when it has data available. When both signals are 1, data is transferred from the portmux to the circuit. One data item is transferred in every clock cycle that has both signals set to 1.

Implementation Internals

Tmemon

The board is physically connected to a workstation through the PCIe bus. A program called tmemon runs on the workstation, and talks to the board over the bus. Other programs talk to the board by communicating with tmemon over the network using TCP/IP. See *The tmemon Program* document for details.

Communication From Workstation to Circuit

The ports library routines talk to the tmemon program over the network. The ports routines look for the port description file in the current directory, and use it to translate port names into specific port numbers.

The tm_read() and tm_write() routines create packets with a 32-bit header. The header contains a board identifier, the port number, a read/write bit and a byte count. The packet along with its header is sent to tmemon over a network socket.

Tme creates the `pcie_portmux` component and adds it to the user’s circuit. The component implements a multiplexor/demultiplexor circuit that can talk to the PCIe interface on the board, receive packets from tmemon over the bus, and transfer data to or from numbered ports on the user’s circuit. This additional wrapper circuit is written in Verilog, and is generated automatically for each user’s circuit. The generated source is placed in a sub-directory called pcie_portmux.

On a read from the circuit to the workstation program, the `pcie_portmux` component will transfer the requested number of bytes from the user’s circuit (with optional handshaking) over the PCIe bus to tmemon, which will return it to the user’s program over the network.

Storage of Port Values on the Workstation

The tm_read() function reads one or more values from a port and stores them in memory on the workstation, starting at the address given as the second argument. Each value will be stored in the smallest number of bytes needed. For example, a 24-bit port value will be stored in 3 bytes.

Each value is stored in memory in what is called "host byte order". On an Intel processor (like the EECG Linux machines) this means that the byte with the lowest address will hold the least significant bits of the value, and the byte with the highest address will hold the most significant bits. On a big-endian architecture like the PowerPC, it means that the byte with the lowest address will hold the most significant bits of the value, and the byte with the highest address will hold the least significant bits.

Alignment of Data with Unusual Widths

Ports on a circuit may be any number of bits. When a port value is stored in memory on the workstation, it will be stored in a series of 8-bit bytes. If the number of bits in the port is not evenly divisible by 8, the value will be padded with 0 to fill up the partial byte. For example, a 21-bit port value will be stored in 3 bytes, and a series of values read from a 21-bit port will each take 3 bytes.

The padding will be added to the most significant bits of the most significant byte (the byte with the highest address on an Intel machine).

Values written to a port by tm_write() will be treated in the same manner. It will be assumed that each value takes up an integral number of bytes. The values will be read in host byte order, with the least significant byte having the lowest address in memory (for an Intel machine). If the width of the port is not a multiple of 8 bits, the most significant bits will be ignored.