The TM-3 Ports Package

Dave Galloway

Department of Electrical and Computer Engineering
University of Toronto

August 2002

Introduction

The TM-3 ports package allows a program running on a UNIX workstation to communicate with a user's circuit in the TM-3 or TM-3A. The user's program may run on any machine, and does not have to run on the machine that is physically attached to the TM-3.

View from the User's Circuit

A circuit for the TM-3 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 purposes of this document, each named set of signals is called a port. For example, a circuit might have an 8-bit input port called `data_in`, a 16-bit output port called `result`, a 3-bit input port called `mode` and a 1-bit input port called `go`.

Create the appropriate input and output signals in your circuit, and leave them dangling. Do not attach them to physical pins on the Virtex FPGA. If you are using tmcc to create your circuit, you would create some of the above signals like so:

```
inputport(data_in)
portflags(data_in, PORT_WIRE);
outputport(result);
portflags(result, PORT_REGISTERED);
```

The ports support software will add a wrapper circuit to your 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.

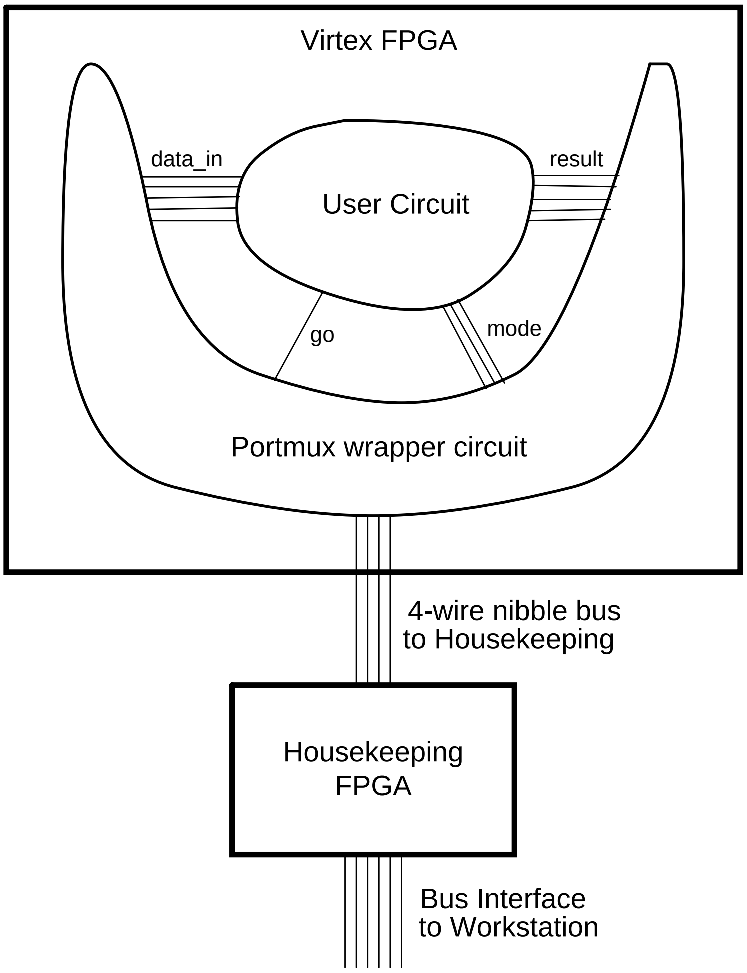

Figure 1: User's circuit wrapped by portmux circuit

View from the User's Program

A program on a workstation can communicate with a circuit in the TM-3 by calling a library of port routines. For example, this C code will set an 8-bit `data_in` port on the circuit to the value 42:

```
char 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 to the clock used by the circuit.

The available routines are:

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

initialize the package, and talk to the TM-3 that is physically attached to the given hostname. If hostname is an empty string, the value of the environment variable `TM3_SERVER` will be used as the hostname. If that variable does not exist, 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, create a port description file with the same name as your design file, but with a .ports extension. For example, if your circuit is called bitfilter.xnf, create a file called bitfilter.ports in the same directory.

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

```
  Name     direction   bits   Handshake_from_circuit   Handshake_from_workstation

data_in    i              8   want_data                data_ready
result     o             16   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 you do not specify flow control signals for a port, then data will be transferred to or from the circuit port when ever the workstation asks for it. If the circuit can not take inputs as fast as the workstation can provide them, or may produce outputs faster than the workstation can take them, then you should add flow control circuitry to your circuit.

Flow Control for Output Ports

Flow control for an output port uses two 1-bit signal wires. Your circuit sets the first signal to 1 when the value on the output port is stable. Your circuit should then wait for the second signal to become 1, then it should lower the first signal to 0, then wait for the second signal to become 0. In tmcc:

```
result = something;
result_ready = 1;
while(!want_result)
        ;
result_ready = 0;
while(want_result)
        ;
```

Flow Control for Input Ports

Flow control for an input port also uses two 1-bit signal wires. Your circuit sets the first signal to 1 when it is ready for more input. Your circuit should then wait for the second signal to become 1, take the new input from the port, then lower the first signal to 0, then wait for the second signal to become 0. In tmcc:

```
data_wanted = 1;
while(!data_ready)
           ;
temp = data_in;
data_wanted = 0;
while(data_ready)
           ;
```

Implementation Internals

Tm3mon

The TM-3 is physically connected to a workstation through a bus interface. A program called tm3mon runs on the workstation, and handles the communication with the TM-3. Other programs talk to the TM-3 by communicating with tm3mon over the network. See *The tm3mon Program* document for details.

Communication From Workstation to Circuit

The ports library routines talk to the tm3mon program over the network. Tm3run tells tm3mon the location of the port description files at the same time that it programs the design into the TM-3. The ports routines ask tm3mon where the port description files are, and use them to translate a port name into a specific chip and port number.

The tm_read and tm_write routines create packets with a 32-bit header. The header contains the chip number, the port number, a read/write bit and a byte count. The packet with header is sent to tm3mon over a socket. Tm3mon uses the header to ship the packet (with header intact) over the bus interface to the TM-3's housekeeping chip, which then passes it over a 4-bit nibble bus to the correct Virtex chip.

The user's circuit is modified by tm3xnf or tm3vhd. It adds a multiplexor/demultiplexor circuit that can talk to the 4-bit nibble bus, receive packets from the workstation, and transfer data to or from numbered ports on the user's circuit. This additional wrapper circuit is written in tmcc, and generated automatically for each user's circuit. The source can be found in the tm3/netlist directory.

On a read from the circuit to the workstation program, the wrapper circuit will transfer the requested number of bytes from the user's circuit (with optional handshaking) over the nibble bus and bus interface to tm3mon, 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 SUN4, 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 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 read or written to a port will be treated as signed twos complement numbers and will be sign-extended. If the most significant bit of the value is a 0, then the padding bits will be 0. If the most significant bit of the value is a 1, then the padding bits will be 1.

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.