An Example VHDL Application for the TM-3A

Dave Galloway

Department of Electrical and Computer Engineering
University of Toronto

August 2002

Introduction

This document describes a simple application circuit written in VHDL for the TM-3A, and explains how to create, compile and run it.

The circuit implements a 32-bit counter. A program running on a workstation will read the output of the counter over the network.

Getting the Source for the Circuit

The source for this example circuit is on the EECG machines in the directory ˜tm3/examples/countervhd.

Sign on to one of the EECG Linux workstations. Make a new directory, and copy some of the files from the example directory:

```
mkdir counter
cd counter
set mydir=$cwd
pushd ˜tm3/examples/countervhd
cp makefile counter.vhd counter.ports suncounter.c $mydir
popd
```

Add the following directory to your path, so that the Transmogrifier commands are available to you:

```
set path=(˜tm3/bin $path)
```

You should probably add that line to your .cshrc file so that the change will be permanent.

The four files that you copied are:

`counter.vhd`	the circuit that will run on the Transmogrifier, described in VHDL
`counter.ports`	a description of the inputs and outputs of the circuit
`makefile`	used to compile the circuit and the program
`suncounter.c`	a C program that runs on a UNIX workstation and talks to the circuit

Compiling the Example

To compile the circuit into a form that can be used to program the TM-3A, type:

```
make tm3abits
```

The counter.vhd source will be compiled by the VHDL compiler, and the resulting circuit description will be passed through a series of scripts and programs that will produce the programming bits for the chips on the board. It will place the results, along with a number of intermediate files, into a new subdirectory called `tm3a` in your current directory. Any existing `tm3a` subdirectory will be removed. The whole process will take about 2 minutes (on a 2 GHz Linux workstation).

To compile the program that runs on a Linux workstation and talks to the circuit, type:

```
make my_linux_counter
```

Downloading the Example to the Transmogrifier

First, you must decide which of the Transmogrifiers you want to use. By default, the commands will use the TM-3A attached to tiger.eecg. If you want any of the other Transmogrifiers, you must set the TM3_SERVER environment variable to the name of the workstation that is attached to the Transmogrifier you want, something like this:

```
setenv TM3_SERVER varan.eecg
```

At this point, you may want to run the Transmogrifier status monitor display on your workstation. Run:

```
tm3status -t &
```

This will produce a new window on your screen which displays the name of the design currently running in the machine, and a column of green and red lights. A circuit is ready when all of the lights are green.

Now use the `tm3get` command to reserve the Transmogrifier for 10 minutes:

```
tm3get
```

If someone else is using the machine, you will get a message telling you how long you will have to wait:

```
sorry, drg has it (priority 100) for 597 more seconds
```

To download your circuit into the TM-3A, type:

```
tm3run
```

If you are running tm3status, you will see the lights go red as the previous design is stopped, and then change to green as your design is loaded. The name of the directory containing your design will be displayed at the top of the status display.

Interacting with the Circuit

To communicate with the circuit, run the my_linux_counter program that you compiled earlier:

```
% my_linux_counter
0 1 2 3 4 5 6 7 8 9
```

Every time you run it, you will get another 10 numbers from the counter.

Stopping the Circuit and Releasing the Transmogrifier

You can stop the design, disabling the chips in the machine, by typing:

```
tm3 stop
```

You should now release the Transmogrifier so that someone else can use it:

```
tm3release
```

How the Sample Design Works

The circuit is written in VHDL. The complete counter.vhd file contains this:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity counter is port(
      want_count : in std_logic;
      count_ready : out std_logic;
      result : out std_logic_vector(31 downto 0);
      tm3_clk_v0 : in std_logic
      );
end;

architecture arch_counter of counter is

      type count_states is (count_null, count_idle, count_count);
      signal counter : std_logic_vector(31 downto 0);
      signal count : std_logic;
      signal count_state, next_count_state : count_states;
      signal count_reset : std_logic;
begin

process(count_state, want_count) begin
      case(count_state) is
            when count_null =>
```

```
                count_reset <= ’1’;
                next_count_state <= count_idle;
                count_ready <= ’0’;
                count <= ’0’;
            when count_idle =>
                count_reset <= ’0’;
                count_ready <= ’1’;
                count <= ’0’;
                if want_count = ’1’ then
                    next_count_state <= count_count;
                else
                    next_count_state <= count_idle;
                end if;
            when count_count =>
                count_ready <= ’0’;
                count_reset <= ’0’;
                if want_count = ’1’ then
                    next_count_state <= count_count;
                    count <= ’0’;
                else
                    next_count_state <= count_idle;
                    count <= ’1’;
                end if;
    end case;
end process;

process(count_reset,tm3_clk_v0) begin
    if count_reset = ’1’ then
        counter <= "00000000000000000000000000000000";
        count_state <= count_idle;
    elsif tm3_clk_v0’event and tm3_clk_v0 = ’1’ then
        if count = ’1’ then
            counter <= counter + 1;
        end if;
        count_state <= next_count_state;
    end if;
    result <= counter;
end process;
end arch_counter;
```

The circuit has one 32-bit output port called `result`, and two 1-bit handshaking variables called `want_count` and `count_ready`. The circuit sits in an infinite loop, waiting for someone to ask for a count. It performs a handshake with the outside world using the `want_count` and `count_ready` variables, increments the counter, and waits for the next request.

The circuit uses the TM-3 ports package to communicate with the outside world. Read *The TM-3 Ports Package* for a description of how this works. The `counter.ports` file contains:

```
# Name      direction  bits  Handshake_from_circuit     Handshake_from_workstation
result      o     32    count_ready      want_count
```

It describes the single 32-bit output called `result` along with the two handshaking variables.

The suncounter.c program that runs on the workstation looks like this:

```
/* A program to test a simple counter on the TM-3, using the ports package */

#include <stdio.h>

main(argc, argv)
      int argc;
      char *argv[];
      {
      int portresult;
      int result[512 * 1024];
      int i, count;

      tm_init("");

      if((portresult = tm_open("result", "r")) < 0) {
            printf("Can’t open port result\n");
            exit(1);
            }

      count = 10;
      if(argc>1)
            count = atoi(argv[1]);

      if(tm_read(portresult, result, count * sizeof(int))
                  != (count * sizeof(int))) {
            fprintf(stderr, "suncounter: error in reading\n");
            exit(1);
            }
      for(i=0; i<count; i++)
            printf("%d ", result[i]);
      printf("\n\n");
      exit(0);
      }
```

It initializes the ports package by calling `tm_init()`, opens the `result` port with `tm_open`, reads 10 values of the counter with a single call to `tm_read()` and then prints them out.

Other Example Circuits

The ˜tm3/examples directory contains several other simple circuits:

`complement` Complements an input, producing an output. No handshaking.

`summer` Takes a series of numbers from an input, adds them and produces the sum. Uses input and output handshaking.

`counter` Similar to the countervhd circuit, but written in Transmogrifier C.

`sramtest` runs a standalone test of the SRAMs. Has a status port so that the workstation can watch the test proceed.