An Example Application for the TMU Package

Dave Galloway

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

April 2013

Introduction

This document describes a simple application circuit for the TMU package on the DE-4 demonstration board, 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 ˜tm4/tmu/examples/de4/counter.

Sign on to one of the EECG Debian Linux workstations (ex: islanders.eecg). Make a new directory, and copy some of the files from the example directory:

```
mkdir counter
cd counter
mydir=‘pwd‘
pushd ˜tm4/tmu/examples/de4/counter
cp makefile top.qpf top.sdc top.qsf $mydir
cp counter.v top.v top.ports suncounter.c $mydir
cp EXT_PLL_CTRL.v $mydir
popd
```

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

```
PATH="/jayar/i/tm4/bin:$PATH"
```

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

The files that you copied include:

`counter.v`	the circuit that will run on the board, described in Verilog
`top.v`	a top level wrapper that connects counter.v to the
`EXT_PLL_CTRL.v`	a Verilog file that describes some pll information needed by the DE-4

`top.sdc`	a file containing timing constraints for Altera’s Quartus software
`top.qsf`	a file containing compiler settings for Altera’s Quartus software
`top.qpf`	another file that Altera’s Quartus software wants to have
`top.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 board, type:

```
make bits
```

The circuit source will be compiled by the `quartus_sh` and `qcmd` commands that are supplied with Altera’s Quartus software. The results, along with a number of intermediate files, will be placed into your current directory. The whole process will take about 5 minutes.

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

```
make linux_counter
```

Downloading the Example to the Board

First, you must decide which demonstration board you want to use. By default, the commands will use the DE-4 board attached to skynet.eecg. If you want to use a different board, you must set the TM_SERVER environment variable to the name of the workstation that is attached to the board you want, something like this:

```
export TM_SERVER=some_other_machine.eecg
```

At this point, you may want to run the status monitor display on your workstation. Make sure your DISPLAY environment variable is set correctly and run:

```
tmstatus -t &
```

This will produce a new window on your screen which displays the name of the person currently using the board, if any.

Now use the `tmget` command to reserve the board for 10 minutes:

```
tmget
```

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 board, run:

```
quartus
```

open the `top.qpf` project and use the Quartus programmer tool to communicate with the USBblaster cable attached to the board.

Interacting with the Circuit

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

```
% 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.

Releasing the Board

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

```
tmrelease
```

How the Sample Design Works

The circuit is written in Verilog. The complete counter.v file contains this:

```
module counter (
   input   clk,
   input   reset,
   output reg [31:0] result,
   output reg count_ready,
   input want_count);

   reg [31:0] count;

   always @(posedge clk or posedge reset) begin
      if(reset) begin
         count <= 0;
         count_ready <= 0;
         result <= 0;
      end
      else begin
         if(count_ready && want_count) begin
            count_ready <= 0;
            count <= count + 1;
         end
         else begin
```

```
            result <= count;
            count_ready <= 1;
         end
      end
   end

endmodule
```

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 TMU ports package to communicate with the outside world. The connections to the outside world are handled by the component named `usb_portmux`. This component is automatically supplied by the TMU ports package. Read *The TMU Ports Package* for a description of how this works. The `top.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 `top.v` that connects the counter module to the automatically generated `usb_portmux` component contains:

```
module top(


// The code above this line was auto-generated by the Terasic DE4 System Builder.

// Connect the user's counter.v circuit to the usb_portmux wrapper that was
// generated by the tmu command.

wire [31:0] result;
wire count_ready;
wire want_count;

usb_portmux usb_portmux_inst(
      .reset_n(rstn),
      .clk(OSC_50_BANK2),
      .OSC1_50(OSC_50_BANK2),

      .result(result),
      .count_ready(count_ready),
      .want_count(want_count),

      .OTG_D(OTG_D),
      .OTG_A(OTG_A),
      .OTG_CS_n(OTG_CS_n),
      .OTG_WE_n(OTG_WE_n),
      .OTG_OE_n(OTG_OE_n),
      .OTG_HC_IRQ(OTG_HC_IRQ),
      .OTG_DC_IRQ(OTG_DC_IRQ),
      .OTG_RESET_n(OTG_RESET_n),
      .OTG_HC_DREQ(OTG_HC_DREQ),
      .OTG_HC_DACK(OTG_HC_DACK),
      .OTG_DC_DREQ(OTG_DC_DREQ),
      .OTG_DC_DACK(OTG_DC_DACK)
);

counter counter_inst(
      .reset(˜rstn),
      .clk(OSC_50_BANK2),

      .result(result),
      .count_ready(count_ready),
      .want_count(want_count)
);

endmodule
```

It connects to a DE-4 clock signal called `OSC_50_BANK2`, and to a number of DE-4 signals with names that start with `OTG_` that connect to the USB interface chip on the DE-4 board. It connects those signals to the `usb_portmux` component, and also connects the user’s signals between the counter and `usb_portmux`.

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

```
/* A program to test a simple counter circuit, using the ports package */

#include <stdio.h>
#include <stdlib.h>

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

      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]);

      result = (int *) malloc(count * sizeof(int));
      if(result == NULL) {
           printf("Can’t allocate memory\n");
           exit(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]);
           if((i % 10) == 9) {
                  printf("\n");
                  }
           }
      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.