ECE4530 Fall 2017
Homework 5: FSMD Intergration, Performance Analysis

Assignment posted on 12 October 5PM
Solutions due on 24 October 5PM

Homework 5 is a continuation of Homework 4.

In the previous Homework, you designed a divder coprocessor that computes an 8-bit quotient and a 16-bit remainder out of two 16-bit numbers.

```
module divider(input wire         clk,
               input wire         reset,
               input wire  [15:0] N,
               input wire  [15:0] D,
               input wire         start,
               output wire [ 7:0] Q,
               output wire [15:0] R,
               output wire        done);
```

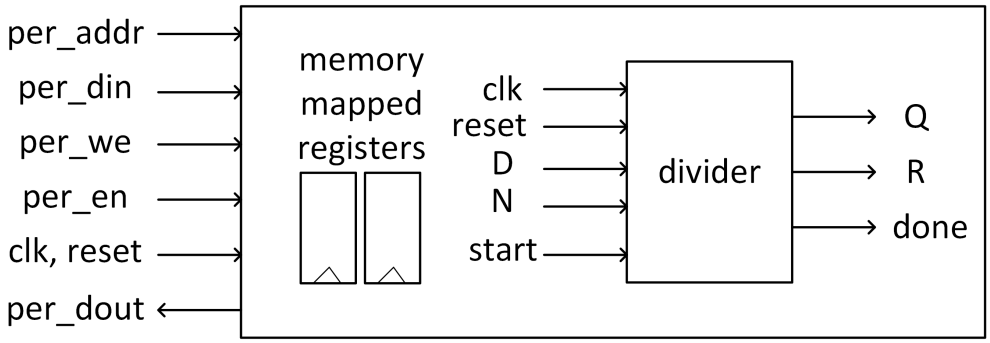
You also developed an on-paper design that describes how you would integrate this coprocessor on the peripheral bus of the MSP 430 microcontroller.

In this Homework, you will carry out this plan and realize the design in Verilog, develop a software test program for it, and test the codesign on your DE1-SoC FPGA board. You will also estimate the performance speedup you have obtained by accelerating the software procedure `intdiv` in hardware.

```
void intdiv(int n,   // numerator,   16 bit
            int d,   // denominator, 16 bit
            int *q,  // quotient,    8 bit
            int *r   // remainder,   16 bit
                     // this function computes the relation 256.n = d.q + r
            ) {
  ... // See HW4
}
```

Part 1: FSMD Integration

The first step is to take the FSMD you have developed in the previous homework, and design a coprocessor interface around it. The recommended approach to achieve this is to encapsulate your earlier solution, as shown in the following figure.



The outer coprocessor interface handles the synchronization with software, and drives the inner divider module. The outer coprocessor interface can be developed as an FSMD, similar to your divider module.

The software driver will use this interface to send the input arguments (N and D) to the hardware coprocessor, and to retrieve the results (Q and R) from it.

To integrate the coprocessor into the MSP430 microprocessor, you can start from the sample design discussed in Lecture 10 (`msp430de`). You will not receive 'starting code' for this design; everything needs to be set up by yourself, or it needs to be assembled from the components and materials you have received earlier.

For this part, please turn in the following.

1. The Verilog design for your coprocessor (including the peripheral bus interface, ie. everything as shown in the figure above).

2. A bitstream (sof file) for the complete MSP430 design, compiled with your integrated coprocessor.

Organize everything in your github repository as follows:

1. `msp430de` is a subdirectory with the integrated design.

2. `msp430de/hardware/msp430de1soc/msp430/mydivider.v` is the top-level of your coprocessor design. If you need multiple Verilog files, add all of them in the directory `msp430de/hardware/msp430de1soc/msp430`. If you add Verilog files, you may need to adjust the `msp430de1soc.qsf` file to reflect the complete collection of files.

3. `msp430de/hardware/msp430de1soc/msp430de1soc.sof` is the bitstream.

4. `verilogfiles.txt` is a text file in the main directory that lists the names of the your verilog files (the ones that you wrote), at one file name per line.

Part 2: Functional Test

You will demonstrate the proper operation of your coprocessor as follows.

- Write a testbench software driver program, that operates the coprocessor for every argument N from 1,000 to 2,999 and for every D from N+1 to 3,000. This will give around 2 million testcases. Thus, the cases to be tested could be expressed through the following C code snippet:

```
int main() {
  int q, r, d, n;

  printf("     N     D     Q     R\n");
  for (n=1000; n<3000; n++) {
    for (d=n+1; d<=3000; d++) {
      // make a division
      intdiv(n, d, &q, &r);

      // show result
      printf("# %4x %4x %2x %4x\n", n, d, q, r);

      // verify result
      assert((n * 256) == (d * q + r));
    }
  }
  return 0;
}
```

  For every combination of `n` and `d`, you have to execute the software reference implementation and the hardware coprocessor, and compare the two outputs.

- The testbench will display hex '00' in the upper 2 HEX displays when this test completes correctly for every value from 0 to 65535.

- The testbench will display '11' in the upper 2 HEX displays, and the failing argument value in the lower 4 HEX displays.

For this part, please turn in the following.

1. `msp430de/software/functionaltest` is a subdirectory containing a program that makes a functional test of your coprocessor following the above guidelines.

Part 3: Performance Test

You will demonstrate the acceleration of the coprocessor, as compared to the software-only implementation, as follows.

- Write a testbench software driver program, that operates the coprocessor for every argument N from 1,000 to 2,999 and for every D from N+1 to 3,000. This is around 2 million testcases. Each time, measure the time it takes to complete the function call (using TimerLap(), as applied in Homework 2). For all of the test cases, remember the highest value of the execution time, and the smallest value of the execution time.

- In the same testbench, collect the minimum and maximum execution time for the software version of the `intdiv()` routine. Use TimerLap(), as applied in Homework 2.

- When the test completes, display a sequence of time measurements on the hex displays. The upper 2 HEX displays will count as 0, 1, 2, 3, with a period of approximately 1000ms per number (e.g. using `longdelay(1000);`.

  In the lower 4 HEX displays will show the results, which change in lockstep with the numbers appearing on the upper 2 HEX display. The lower HEX displays will show (1) the minimal execution time of the software divider; (2) the maximal execution time of the software divider; (3) the minimal execution time of the hardware divider; and (4) the maximal execution time of the hardware divider respectively. You do not have to adjust your results for the overhead of using TimerLap(). It is sufficient to provide the estimate as measured by TimerLap().

For this part, please turn in the following.

1. `msp430de/software/performancetest` is a subdirectory containing a program that makes a performance test of your coprocessor following the above guidelines.

What to turn in

You do not have to submit anything on Canvas. However, you will have to update your github repository as specified under Part 1, Part 2, and Part 3 above.

The root of your repository will include the homework assignment, the file `verilogfiles.txt` and the `msp430de` design.

Within the `msp430de` design, that hardware will have been customized with your hardware coprocessor. The software will contain (at least) two applications, one for the functional test and one for the performance test.

Keep in mind that you have to include the compiled bitstream of your implementation. We will verify if your bitstream is genuine for the files you submit.