ECE4530 Fall 2017
Homework 4: Designing a Hardware Divider and its Coprocessor Interface

Assignment posted on 5 October 8AM
Solutions due on 12 October 5PM

Homework 4 is the first of a sequence of two Homework, going through the design process of a coprocessor for the MSP430 environment. In Homework 4, you will develop a hardware implementation of a hardware division as an FSMD (Finite State Machine with Datapath). In addition, you will prepare a concept design to integrate this FSMD with an MSP430 using synchronization primitives (SYNC0, SYNC1). In Homework 5, you will continue the actual implementation and integration of the divider with the MSP430, map it on the DE1SoC FPGA board, and evaluate the performance improvement of the hardware divider over an all-software implementation.

For each Homework, you have one week of design time.

As with other Homework, you have to make use of `github` to download the initial assignment. Navigate to `https://classroom.github.com/a/0o7BjLnr` to accept the starter files. You will find the following code.

- `hw4.pdf`: this assigment.
- `divider/div.v`: a reference implementation in C (see part 1).
- `dividerhw1/dividertb.v` and `dividerhw1/divisior.tv`: a hardware testbench that must be used by your design (see part 1).

Part 1: FSMD Design for a Hardware Divider

Implement an FSMD in Verilog for the following C function, which computes the 8-bit quotient and 16-bit remainder of a 16-bit by 16-bit divider.

```
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
            ) {
  unsigned i;
  int qt, rt;
```

```
  qt = 0;
  rt = 0;
  if (d == 0)
    return;

  rt = 2 * n - d;
  for (i=0; i<7; i++) {
    if (rt < 0) {
      qt = 2 * qt;
      rt = 2 * rt + d;
    } else {
      qt = 2 * qt + 1;
      rt = 2 * rt - d;
    }
  }
  if (rt < 0) {
    qt = 2 * qt;
    rt = rt + d;
  } else {
    qt = 2 * qt + 1;
  }

  *q = qt;
  *r = rt;
}
```

This design is a so-called non-restoring divider. From two input numbers `n` and `d`, it will compute a quotient `q` and a remainder `r` such that the following relation holds:

```
N * 256 = (D * Q + R)
```

where `N, D` and `R` are 16-bit numbers, and `Q` is an 8-bit number.

Your hardware design must be able to compute the same result as the above C function, and it must meet the following design contraints.

- It must use the following interface.

```
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);
```

`start` is a start command for the divider. After a single-cycle `start` pulse, the divider reads the inputs `N` and `D` and computes the quotient `Q` and remainder `R`. As soon as the output is available, the `done` status pin goes high and stays high (along with the data output) until the next `start` pulse is provided. A sample timing diagram is shown further in this assignment.

- You are free to choose the number of cycles needed to complete the division, as long as the `start` control pin and `done` status pin operate correctly according to the protocol.
- The design needs to be written in synthesizable Verilog. The easiest manner to achieve this is to apply an FSMD coding style (such as taught in ECE 3544). I will briefly cover FSMD coding styles next week; you can refer to the example code provided in Lecture 10.
- Your design needs to be driven by the testbench provided with the homework assignment.

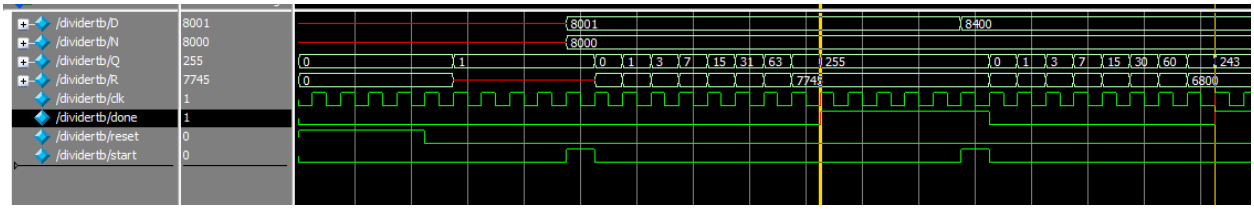

Figure 1: Timing Diagram of the Hardware Divider Interface

Figure 1 illustrates the interface timing of a correct solution. Note that the cycle budget of your solution does not have to be identical to the solution shown, as long as it follows the interface specifications given above.

Write your solution in a Verilog module/file `divider.v` in the subdirectory `dividerhw1`. You can verify the correctness of your solution as follows.

- The C reference implementation, available in the `divider` subdirectory, can be executed with the following commands.

```
gcc -o div div.c
./div
```

  You will see a sequence of 8000 divisions. The value of `N` is always the same, but the value of `D` is slowly increasing. The C program generates a testvector file, `divisor.tv` which can be used by the Verilog testbench.

- The Verilog implementation can be simulated in Modelsim or at the command line (in Cygwin) as follows.

```
vlib work
vlog divider.v
vlog dividertb.v
vsim dividertb -do "run -all"
```

Note that the command line method is not well suited to debug your design as it does not offer a waveform viewer. Use the command line method as the final verification, when your design appears to be working correctly.

You will need to provide `divider.v` as the answer to part 1. The file needs to be located in the `dividerhw1` directory.

Part 2: Coprocessor Interface Design for Square Root Processor

You now have to build a hardware/software interface for the MSP430 for this divider processor. The top-level software interface of the driver looks as follows:

```
void driver(int N, int D, int *Q, int *R);
```

You have to present a design for the following components:

- The implementation of `driver`, using operations on memory-mapped registers and synchronization primitives.
- The design of a Verilog module with an MSP430 peripheral interface, which contains memory-mapped registers, and which integrates the FSMD design of Part 1.

For this homework, you do not have to deliver a prototype (ie. you do not have to submit finished and verified C/Verilog code), but you have to create a design document. The design document will explain the following elements:

- Block diagram of the overall design, showing all modules and interconnections starting from the MSP430 peripheral bus up to the interface of the FSMD you created in Part 1.
- Design of the hardware interface between the MSP430 peripheral bus and the FSMD you designed for Part 1. You can, for example, draw a finite state machine. You can also explain how many registers you will allocate and what their purpose is. You can also explain how your coprocessor interface will synchronize with software, ie. what sort of SYNC primitives you will use.
- Design for the software driver, namely, the implementation of `hardwaredivider`. Again, you do not have to write detailed C code, but you need to explain how you will use `SYNC0`, `SYNC1` primitives.
- You need to provide an argument for the overall correctness of the design, ie. clarify why the proposed design is correct, according to your insight.

This question is, in essence, a documentation question which requires you to come up with a solid design before actually implementing it. Your answer will be graded for completeness, as well as for correctness. You can use the example discussed in Lecture10 as a basis.

The answer to this question has a minimum required length of two pages, including the graphics, tables, etc.

What to turn in

You will turn in your answer as a homework repository in `github`.

1. For part 1, make sure to add `divider.v` to the repository. The module needs to be present in the `dividerhw1` subdirectory.

2. A PDF document, `part2_username.pdf`, placed in the root directory of your repository, that provides the answer to part 2 of the Homework assignment.