Design of Digital Circuits and Systems, HW2

Memories

Deliverables (Gradescope)

1) Homework problem solutions (*i.e.*, text, diagrams, screenshots, work) in a single PDF file ending in `.pdf` (all lowercase).
   a) At the end of this document, estimate how long you spent working on the homework and rate the difficulty on the following scale: Very Hard — Hard — Moderate — Easy — Very Easy
2) SystemVerilog files `sync_rom.sv`, `sign_mag_add.sv`, and `sign_mag_add_tb.sv`.

Problems

Problem 1:

Specify the *minimum* size (*i.e.*, number of words and number of bits per word) of separate ROMs that will accommodate the truth tables for the following combinational circuit components. Assume the output will be as wide as is needed to represent the largest possible result (no truncation):

a) A binary multiplier that multiplies two 4-bit binary words

b) A 3-bit adder-subtractor

Problem 2:

The following questions may depend on each other and are intended to be done in order:

a) How many 4 Mi × 16 RAM chips are needed to provide a total memory capacity of 64 Mi-bytes?

b) How many address bits are needed for our larger, 64 Mi-byte-memory assuming the same word size as the individual chips?

c) How many of the address bits are connected to *each* RAM chip?

d) How many of the address bits must be decoded for the chip select (*i.e.*, how many are needed for us to know which RAM chip we need to access)? Specify the size of the decoder.

Problem 3:

You are provided with two implementations of a 4-bit sign-and-magnitude adder: (1) a SystemVerilog module found in *sign_mag_add.sv* and (2) a corresponding ROM table found in *truthtable4.txt*.

> These operations can be tricky to interpret at first because you're so used to Two's Complement arithmetic. First interpret both inputs' values in sign-and-magnitude, add the two values together, and then attempt to encode the result back into sign-and-magnitude. If the result can't be properly represented, we call this *arithmetic overflow*.

a) Complete the testbench in *sign_mag_add_tb.sv* for an instance of `sign_mag_add` named `dut1`. Make sure that you include at least 8 different test cases that cover these situations:

- Some number + 0
- pos + neg > 0
- pos + pos (valid)
- neg + neg (valid)
- pos + neg = 0
- pos + neg < 0
- pos + pos (overflow)
- neg + neg (overflow)

b) Modify the following code (provided for you in *sync_rom.sv*) to instead work with the data found in *truthtable4.txt* (which you should open to view). If you open *truthtable4.txt* in Quartus, **don't** add the file to your project.

```
module sync_rom (input  logic clk,
                 input  logic [3:0] addr,
                 output logic [6:0] data);

   // signal declaration
   logic [6:0] rom [0:15];

   // load binary values from a dummy text file into ROM
   initial
      $readmemb("data.txt", rom);

   // synchronously reads out data from requested addr
   always_ff @(posedge clk)
      data <= rom[addr];

endmodule  // sync_rom
```

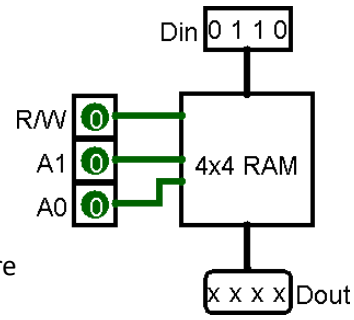
> The `$readmemb` argument shown (`"data.txt"`) uses a *relative* path. If you encounter the following warning in ModelSim (memory will also show up as all x's and a red line):
>
> *"# ** Warning: (vsim-7) Failed to open readmem file "data.txt" in read mode."*
>
> Solution: Replace the argument with the *absolute* path (*e.g.*, `"C:/371/hw2/data.txt"`).

c) Add an instance of `sync_rom` named `dut2` in your testbench from Part A alongside `dut1` and verify that they produce the same behavior in ModelSim. If you see timing differences between the two, you should make sure to understand where they come from but DON'T make other adjustments. Submit your modified code files, but you do not need to include a screenshot of your simulation in your submission PDF.

d) Compare the resource usage of these two implementations. Synthesize in Quartus with *sync_rom.sv* and then *sign_mag_add.sv* as your top-level module. Find the "Resource Usage Summary" page in the Compilation Report and compare the number of ALMs and memory bits used (omission means 0). Include this comparison in your submission PDF.

Problem 4:

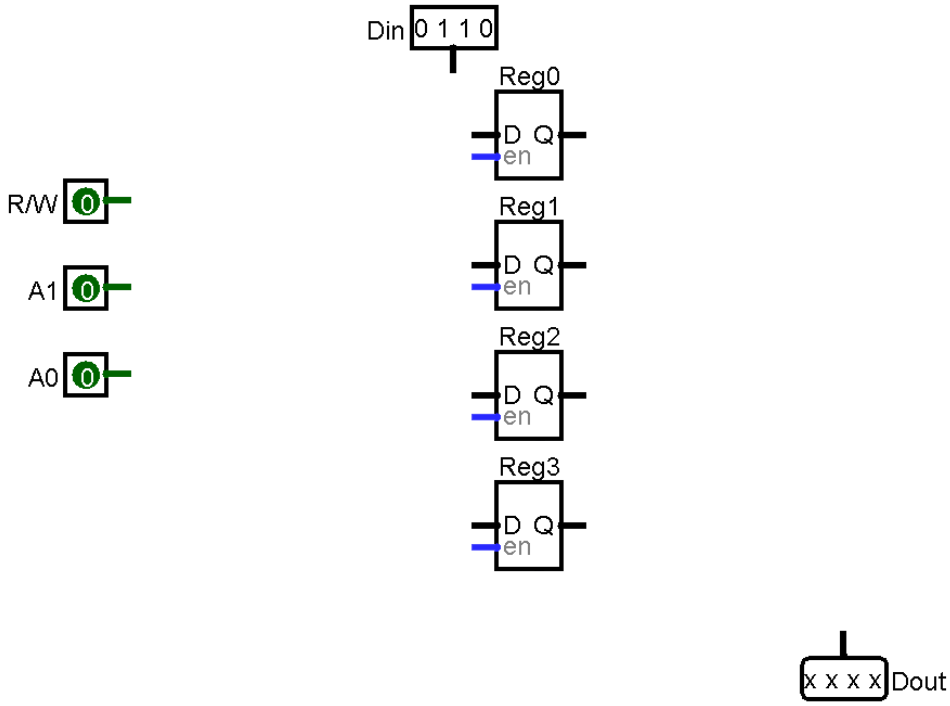
Let's examine a 4×4 RAM module as shown on the right:



- The `R/W` input is `1` for writing and `0` for reading.
- A_1A_0 forms the two-bit address that selects which word is read out to `Dout`.

In the following parts, draw out a memory diagram by adding/using wires, gates, and plexors (*i.e.*, multiplexors or demultiplexors). The clock inputs are not shown, but should be assumed to be present and connected correctly.

a) Complete the diagram below for an implementation of the 4×4 RAM as a *register file*. The `en` inputs to the registers are enable signals (*i.e.*, the register will update when `en=1` and remain the same when `en=0`).



b) Construct an 8×8 memory diagram using the given 4×4 RAM as a building block. Hint: how will the inputs and outputs differ for a 8×8 RAM compared to the 4×4 RAM block diagram shown above?