ECE M16 Take-Home-Final Exam Summer 2022

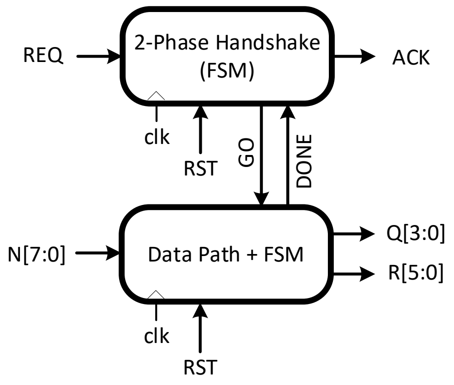
Notes:

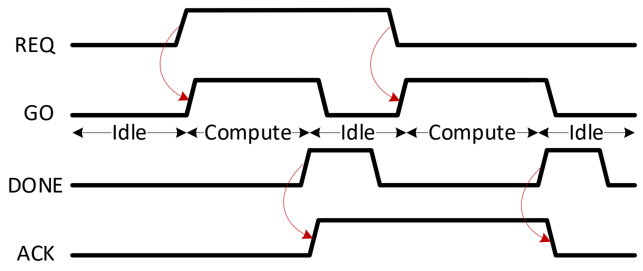
- **You have 24 hours to finish the exam. It is worth 100 points.**
- **The exam is to be done individually.**
- **Open everything.**
- **You need to submit your design details, the logisim schematic(s), answers to specific questions, and waveform samples showing the proper functionality (through chronogram) as specified.**
- **You have to submit your final PDF no later than 4PM Thursday 8/11.**

Make sure proper details and all the schematics are included. Partial credit will be given even if your system is not fully functional.

Square root two calculator based on digit by digit computation: You have to design a sequential system that receives an 8-bit unsigned input, $N[7:0]$, and outputs $Q[3:0]$ which is a 4-bit unsigned integer representing the square root of N rounded to the nearest value representable, as well as $R[5:0]$, the remainder, which is nonzero if N is not a perfect square. For simplicity, your outputs (Q and R) may be both 8-bits, same as the input, or you can use the *Bit Extender* function in logisim to adjust as appropriate. You need to fully verify your design in logisim, similar to the homework assignments.

I/O protocol: To communicate with the outside world, the system uses a REQ and ACK based handshake protocol that is slightly different from the one you saw in homework 4. Here we use a handshake protocol that is called a *two-phase handshake* . In the two-phase handshake the outside world presents data every time REQ changes value (i.e. 0 to 1, or 1 to 0) as opposed to only when it goes from 0 to 1 as was in homework 4. This eliminates the dead time when the system had to wait for REQ and ACK to be reset to 0. Furthermore, to make a more efficient use of the system, we think of doing the square root computation as calling a function where one prepares the argument, calls the function, and then waits for it to be done. So, we can design a subsystem that starts doing the square root computation when given a $GO = 1$ signal and which when done asserts a $DONE = 1$ signal. A typical timing diagram is shown below. The algorithm will be then as follows: Wait for $REQ = 1$. Then set $GO = 1$, and wait for $DONE = 1$. Then set $ACK = 1$. Wait for $REQ = 0$. Then set $GO = 1$, and wait for $DONE = 1$. Then set $ACK = 0$.





Reset: Sometime after the power up, your system will get for at least one clock edge $RST = 1$. You should design the system to move to the initialization state as a result when you see $RST = 1$. You can use the reset input for all your D flip flops and registers. For simplicity, assume you always start from the reset.

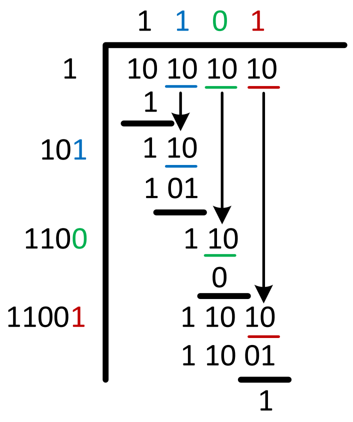
Algorithm: The algorithm to calculate the square root two on digit by digit basis (https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#Digit-by-digit_calculation) is based on the simple observation that for a 2-bit binary number $N = n_1 n_0$ we can write:

$$(n_1 n_0)^2 = (2n_1 + n_0)^2 = (2n_1)^2 + 4n_1 n_0 + {n_0}^2$$

And after regrouping:

$$n_1 {n_0}^2 - (2n_1)^2 = (n_1 \gg 2 + n_0).n_0$$

where $n_1 \gg 2 = 4n_1$ denotes n_1 *shifted to the left twice*. The algorithm can be best described by an example, where for instance we would like to find out the square root two of $10101010_2 = 170_{10}$:



In summary, we have to take the following steps: Pairing: we pair the numbers in groups of two from right to left. Initial guess: The leftmost group is 10 (i.e., 2_{10}). So, we find the largest bit, b, such that $b^2 \leq 10$. We find $b = 1$. This is the first bit of our square root. Subtraction: we subtract 1 from 10 and bring down the next group to the right of the remainder. That would be 110 shown in the next row. We repeat the guess and subtraction steps until the number is exhausted. If the number were a perfect square, we would get zero remainder at the end, with the number at the top representing the square root. In our example, since we started with $170 = 13^2 + 1$, we have $1101_2 = 13_{10}$ as the final answer with a remainder of 1.

Implementation: With the background information above, perform the following tasks:

1. (4) Using the digit by digit algorithm, take the square root two of $11000010_2 = 194_{10}$. Show the steps. You can use this to test drive and/or debug your design.
2. (8) Design a combinational logic sub-circuit that takes an 8-bit input $i[7:0]$, and a 2-bit control signal $c[1:0]$, and creates a 2-bit output $o[1:0]$ as follows: If $c[1:0] = 00$, then $o[1:0] = i[7:6]$, if $[1:0] = 01$, then $o[1:0] = i[5:4]$, and so on. You will use this sub-circuit to partition the input number into the pairs of 2, consistent with the algorithm described.
3. (8) Verify your design using logisim. Any gate is allowed.
4. (5) Design a 2-bit counter using D flip flops and gates only. The counter will keep track of which pair of the input number is being processed at a time. You can think of it as the controller part of the system.
5. (5) Verify your counter using logisim. Make sure your counter has a reset function (can be through the flip flop).
6. (10) Design the handshake circuit. The handshake is a Moore FSM. Show the transition table, state diagram, and other details.
7. (10) Verify your design using logisim. D flip flops and gates are allowed. Use the chronogram function of logisim to depict a sample waveform proving the correct functionality of your design, similar to the one showed earlier.
8. (50) We are finally set to design and verify the controller FSM and the data path to calculate the square root function. The square root two circuit can be fully designed with combinational logic. However, as the number of the input bits increases, it becomes too complicated. On the other hand, given the repetitive nature of the algorithm, a modular design using sequential logic to process each pair, one at a time, is more economical, and that is how it will be done here. Assume that you always start from reset (which you will include in your design). Write an algorithm (or flowchart) performing the square root function, and build your data path accordingly in logisim. Your design may include D flip flops and registers, adders/subtractors, shifters, multiplexers, comparators, and any kind or number of gates. Note that starting at reset, with everything zero, the first step is not really any different from the following steps. You should be able to complete the square root function in 3-4 clock cycles for an 8-bit input. Note that your system will also include a GO input, and a $DONE$ output, as well as of course the RST. Setting $GO = 1$ enables (and starts) the system, and when the square root is computed, $DONE = 1$ is asserted, and the system will go idle with the outputs held at the computed values. $GO = 0$ or $RST = 1$ will reset the system. Test your system with 11000010_2 input and show a snapshot of the logisim when the computation is done.