CSE3666

Homework 4

Firm deadline: By the end of Monday, 3/18/2024.
Total points: 100

Submit your work in a single PDF file in HuskyCT.

1. We build a state machine in this problem. The machine has four states, 1-bit input, and 1-bit output. The initial state is state S0. Depending on the current state and the input bit, the state machine transits from one state to another, as shown in the following state table. In the table, each row specifies what the state machine should do in a cycle, for each state and input combination. `b` is the input and `z` is the output.

State	b	NextState	z
S0	0	S0	1
S0	1	S1	0
S1	0	S2	0
S1	1	S0	1
S2	0	S1	0
S2	1	S2	0
S3	0	S0	1
S3	1	S1	0

Implement the state machine in MyHDL. The skeleton code is in `q1.py`. We complete the design in 4 steps. Steps 2, 3, and 4 are combinational circuit design.

Since we have four states, we will use a signal of two bits to keep track of the state. In the skeleton code, the signal is `state`. It has two bits. The underlying data type of the signal is a 2-bit vector. It is the output of a register. Its value indicates the current state. 0 means S0, 1 means S1 and so on. We can access each bit in `state` with `state[0]` or `state[1]`.

Step 1. Instantiate a register to keep the state. This step is already done. Signal `next_state` is the input of the register and `state` is the output. Study the code and learn how to instantiate a block and a register.

Step 2. We start by turning the state table (above) into a truth table like the following. Then we write a logic expression for each `next_state[0]`, `next_state[1]`, and `z`. The logic expressions will be used in later steps.

state[1]	state[0]	b	next_state[1]	next_state[0]	z
0	0	0	0	0	1
0	0	1	0	1	0
…					

Step 3. Complete the `next_state_logic()` function in `q1.py`. The function generates `next_state[1]` and `next_state[0]`, based on the logic expressions in Step 2. Note that `next_state` is the input to the register and determines the state in the next cycle. See the additional requirements in the comments.

Step 4. Complete the `output_logic()` function. The function uses the logic expression in Step 2 to generate the output signal `z` from the current state and the current input. See the additional requirements in the comments.

Testing

When testing the circuit, we can specify bits on the command line. Here is the output of the program where the bit string is `10011`.

```
python q1.py 10011
state b | ns z v
  0   1 | 1  0 1
  1   0 | 2  0 2
  2   0 | 1  0 4
  1   1 | 0  1 9
  0   1 | 1  0 19
  1   1 | 0  1 39
```

In the submitted PDF file, include the following for this problem.

a. Step 2. Truth table and logic expressions.
b. Step 3. Code for function `next_stage_logic`.
c. Step 4. Code for function `output_logic`.
d. The output of the program when the bit string is 111000101.

2. Consider the multiplier we have studied. Inside the control module, there is also a register that counts the steps and a combinational circuit. Note that the register in the control module is triggered by the same clock and the output of the control module depends on the register.

   Assume the following.

   - The propagation delay of the registers is 2 ns.
   - The delay of the adder is 10 ns,
   - The delay of the combinational circuit in the control module is 1 ns.
   - Do not consider the propagation delays on wire.

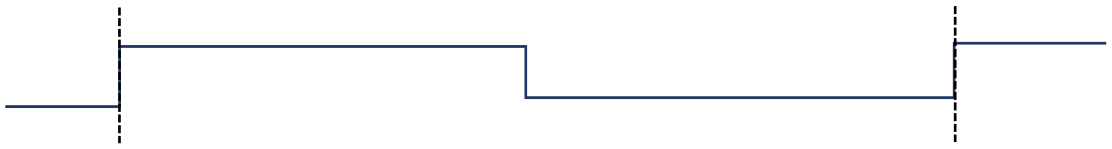
   Answer the following questions. Round answers to the nearest tenth if necessary.

   a. In a figure like below, show the timing of the following events, relative to the beginning of a cycle (i.e., when after the rising edge does each of the following events happen?)

      a1. Reg-Ready. The output of registers is available.

      a2. Control-Ready. The output of control module is available.

      a3. Adder-Ready. The output of adder is available.



   b. What is the minimum cycle time for this multiplier to work properly?
   c. What is the highest clock rate in MHz that this multiplier can run at?
   d. If we build sequential circuit with the same kind of registers, what is the highest clock rate in MHz we can achieve?

3. Assume we have built a 5-bit multiplier, based on the design we have discussed, and use it to calculate 27 * 17. Fill out the following table with bits stored in registers after each step. Verify with decimal arithmetic that a) the product register has the correct answer if bits are considered as unsigned, and b) the lower half the product register has the correct bits if bits in 27 and 17 are considered as signed.

Steps	Multiplicand	Multiplier	Product
init			
1			
2			
…			

4. Translate the following C function to RISC-V assembly code. We can use M extension in this question. The function converts an unsigned number into a string representing the number in decimal. For example, after the following function call, the string placed in buffer is "3666".

   ```
   uint2decstr(buffer, 3666);
   ```

   Assume the caller has allocated enough space for the string. Skeleton code is in `q4.s`, where the function is empty. The assembly code should follow RISC-V calling convention. Clearly mark in comments how each statement is translated into instructions. Only include the instructions (and comments) in the function in the PDF file.

```
// char * means the address of a character
char * uint2decstr(char s[], unsigned int v)
{
    unsigned int r;

    if (v >= 10) {
        s = uint2decstr(s, v / 10);
    }
    r = v % 10;       // remainder
    s[0] = '0' + r;
    s[1] = 0;
    return &s[1];     // return the address of s[1]
}
```