Homework 4

Due Date: By the end of Friday, 3/25/2022.

Total points: 200

Check your answers to Problems 3 and 4 in HW4-Test in HuskyCT.

In addition, submit your work in Problems 1, 2 and 5 in a PDF file in HuskyCT.

1. Design a circuit that takes 4 bits as input and outputs F, which is 1 only when the 4-bit input, interpreted as an unsigned number, is positive and divisible by 3. The input signals are A, B, C, and D. D is the least significant bit.

   We can start a truth table like the one below and then write a logic equation for F.

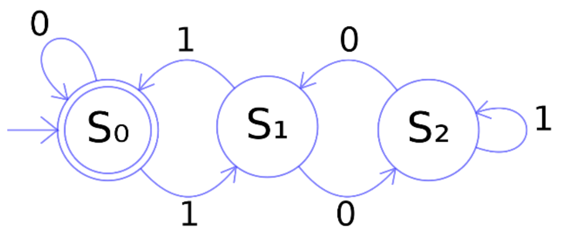
   We do not simplify the logic expression in this problem.

Row no.	**A**	**B**	**C**	**D**	**F**
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
…					

   Implement the circuit in MyHDL. The skeleton code is in `q1.py`. Compare the truth table generated by the script with the one constructed manually.

   The PDF file needs to include `comb()` function only.

2. We build a state machine that detects if a binary number of an arbitrary length is divisible by 3. The state machine has three states S0, S1, and S2. The bits in the number are fed into the machine from left to right, i.e., from the most to the least significant bit, one bit per clock cycle. The state machine starts from S0. Depending on the current state and the input bit, the state machine transits from one state to another, as shown in the following diagram. The numbers in the state names (S0, S1, and S2) are the remainder when we divide by 3 the bits that the machine has seen. If the state is S0, the bits are divisible by 3.



Implement the state machine in MyHDL. The skeleton code is in `hw4q2.py`. We can complete the design in 3 steps. Steps 2 and 3 are combinational circuit.

Step 1. Instantiate a register to keep the state. We leverage the Register block that is provided in the skeleton code. The input and output signals of the state register have already been created. You can find an example of instantiating a Register in shift_register.py on the following page.

cse3666/shift-regsiter.py at master · zhijieshi/cse3666 (github.com)

Step 2. Complete the `next_state_logic()` function, which generates the state to be saved in the state register in the next cycle.

Step 3. Complete the `z_logic()` function, which generates the output signal z, which indicates whether the number is divisible by 3. The z signal only depends on the current state.

In the PDF file, list your code for each step. In addition, show the output of the program when eight random bits you choose are sent to the machine. We can specify bits on the command line. Here is an example where the bit string is `110111`.

```
python hw4q4.py 110111
b | z v
1 | 0 1         # the first bit is the MSB, which is 1
1 | 1 3         # two left most bits 11. It is divisible by 3
0 | 1 6         # now the machine sees three bits 110
1 | 0 13        # and so on
1 | 1 27
1 | 0 55        # all 6 bits are fed to the state machine
```

Although it is not required, you are encouraged to generate the trace file and observe the waveforms of signals.

3. Consider the multiplier we have studied. If the delay of the adder is 10 ns, the setup time, the hold time, and the propagation delay of the registers is 0.2 ns, 0.25 ns, and 0.3 ns, respectively.

   When entering answers in HuskyCT, round answers to the nearest tenth if necessary. For example, enter 3 for 3, 0.3 for 1/3, and 0.7 for 2/3. Keep the 0 before the decimal point.

   a. What is the minimum cycle time for this multiplier to work properly?
   b. What is the highest clock rate that this multiplier can run at?
   c. If we build a sequential circuit with the same kind of registers and want the circuit run at 1 GHz, what is the maximum delay of the combinational module?
   d. If we build sequential circuit with the same kind of registers, what is the highest clock rate we can achieve?

4. 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.

Steps	Multiplicand	Multiplier	Product
init			
1			
2			
3			
4			
5			

5. Translate the following C function to RISC-V assembly code. The function converts an unsigned number into a string that represents 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 `q5.s`, where the function is empty. Clearly mark in comments how each statement is translated into instructions. The PDF file needs to include the instructions in the function only.

```
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]
}
```