Archie Ross
3/25/2023

CSE 3666 Homework 4

1. **State Machine**

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].

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

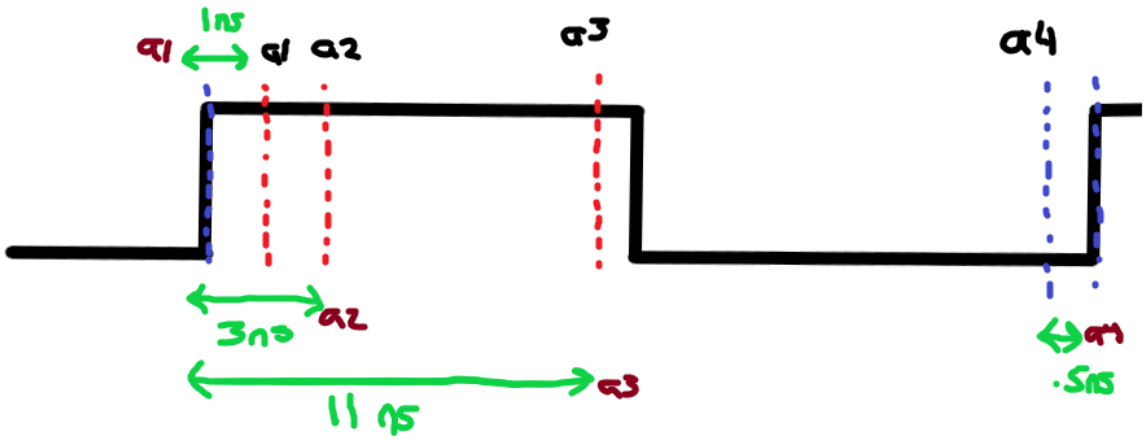
Steps 3 and 4 along with part d: (q1.py is also attached to the submission)

```
def next_state_logic():
    # TODO
    # Two statements to set two bits in next_state
    next_state.next[1] = (b & ~state[0] & ~state[1]) | (~b & ~state[0] & ~state[1])
    next_state.next[0] = (~b & state[0] & ~state[1]) | (b & ~state[0] & state[1])
# generate output
@always_comb
def output_logic():
    if state[1] == 1 and state[0] == 0:
        z.next = 1
    else:
        z.next = 0
```

```
(venv) PS C:\Users\volac\Downloads> python q1.py 111000101
state b | ns z v
  0   1 | 2  0 1
  2   1 | 1  1 3
  1   1 | 0  0 7
  0   0 | 0  0 14
  0   0 | 0  0 28
  0   0 | 0  0 56
  0   1 | 2  0 113
  2   0 | 2  1 226
  2   1 | 1  1 453
  1   1 | 0  0 907
```

2. **Clock Cycles**

a. **In a figure like below, show the timing of the following events.**
a1. Reg-Ready. The output of registers is available.
a2. Control-Ready. The output of the control module is available.
a3. Adder-Ready. The output of adder is available.
a4. Deadline. The input to registers must be ready. This is the example in the figure.



b. **What is the minimum cycle time for this multiplier to work properly?**
The minimum cycle is just the total time it takes for all modules to run as well as the deadline buffer which has to be .5ns. This leaves us with 11.5ns

c. **What is the highest clock rate in MHz that this multiplier can run at?**
The calculation is as follows: 1000 / 11.5ns = 86.96 MHz

d. **If we build a sequential circuit with the same kind of registers, what is the highest clock rate in MHz we can achieve?**
Using the same kind of registers would mean no calculation is taking place. We can calculate the highest clock rate by taking the 1ns and taking the deadline buffer to get 1.5ns. We then do the conversion like above to get 1000/1.5 = 666.67MHz

3. **5 Bit Multiplier Table**

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	0000011011	10001	0000000000
1	0000110110	01000	0000011011
2	0001101100	00100	0000011011
3	0011011000	00010	0000011011
4	0110110000	00001	0000011011
5	1101100000	00000	0111001011

4. **uint2decstr() Translation**

Translate the following C function to RISC-V assembly code. 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]
}
```

```
uint2decstr:
        addi    sp, sp, -8 #allocate save space
        sw      ra, 4(sp) #save ra
        sw      a1, 0(sp) #save a1
        addi    t1, x0, 10 #set t1 equal to 10
        bltu    a1, t1, else #if a1 < 10 go to else
        divu    a1, a1, t0 #uint2decstr(buffer, a1/10)
        jal     ra, uint2decstr #recursive call ^

else:
        lw      ra, 4(sp) #restore ra
        lw      a1, 0(sp) #restore a1
        addi    t1, x0, 10 #t1 is 10 again
        remu    t2, a1, t1 #convert remainder to decimal and add to mem
        addi    t1, t2, '0'
        sb      t1, 0(a0) #least sd in mem
        sb      t1, 1(a0) #2nd least sd in mem
        addi    a0, a0, 1 #increment pointer
        addi    sp, sp, 8 #put back the space on the stack
        jr      ra
```

```
uint2decstr:
        addi    sp, sp, -8 #allocate save space
        sw      ra, 4(sp) #save ra
        sw      a1, 0(sp) #save a1
        addi    t1, x0, 10 #set t1 equal to 10
        bltu    a1, t1, else #if a1 < 10 go to else
        divu    a1, a1, t0 #uint2decstr(buffer, a1/10)
        jal     ra, uint2decstr #recursive call ^

else:
        lw      ra, 4(sp) #restore ra
        lw      a1, 0(sp) #restore a1
        addi    t1, x0, 10 #t1 is 10 again
        remu    t2, a1, t1 #convert remainder to decimal and add to mem
        addi    t1, t2, '0'
        sb      t1, 0(a0) #least sd in mem
        sb      t1, 1(a0) #2nd least sd in mem
        addi    a0, a0, 1 #increment pointer
        addi    sp, sp, 8 #put back the space on the stack
        jr      ra
```

5. **Processing Speeds**

Consider two processors P1 and P2 that have the same ISA but different implementations. The ISA has four classes of instructions: class A, B, C, and D. The clock rate of two processors and the number of clock cycles required for each class on the processors are listed in the following table.

Processor	Clock Rate	Class A	Class B	Class C	Class D
P1	2 GHz	1	2	3	3
P2	3 GHz	1	1	4	5

Suppose the breakdown of instructions executed in a program is as follows:

10% class A, 20% class B, 50% class C, and 20% class D.

Round answers to the nearest hundredth if necessary. For example, 1/2 to 0.5, 2/3 to 0.67.

a. What is the overall CPI of the program on P1?

To do this we need to use the percentages and do as follows: (0.1 * 1) + (0.2 * 2) + (0.5 * 3) + (0.2 * 3) = 2.6

Leaving us with a CPI of 2.6 for P1

b. What is the overall CPI of the program on P2?

Following the same as above, (0.1 * 1) + (0.2 *1) + (0.5 * 4) + (0.2 * 5) = 3.3

Leaving us with a CPI of 3.3 for P2

c. How many times faster is the program on P2 than on P1? Note that the clock rate is different on P1 and P2.

Execution time of P1 = 2.6/2 = 1.3ns

Execution time of P2 = 3.3/3 = 1.1ns

1.3ns/1.1ns = 1.18 times.

P2 is 1.18 times faster than P1.

d. Suppose a compiler can optimize the program, replacing all class D instructions with class A instructions. Each class D instruction requires two class A instructions. What is the average CPI of the program on P2 after the optimization?

To do this we need to use the percentages again but recalculate them due to one D instruction now becoming 2 A instructions: (120 *.1), (120 *.2), (120 *.5), (120 * .2) = (A = 12), (B = 24), (C = 60), (D = 24)

The average CPI of the program on P2 will be 2.25

e. What is the speedup the compiler in d) can achieve on processor P2?

The speedup would be 2.25/3 = 0.75ns 1.22 times

6. **Speedup methods**

Suppose you have two different methods to accelerate a program. Method 1 can accelerate 20% of the program 100 times. Method 2 can accelerate 20% of the program 10 times and 15% of the program 6 times. The part of code enhanced by each method does not overlap.

Round answers to the nearest hundredth if necessary. For example, 1/2 to 0.5, 2/3 to 0.67.

a. What is the speedup Method 1 can achieve on the entire application?
1/((1 - 0.2) + (0.2 / 100 + 0.2)) = 1.25
Method 1 can speedup the application by 1.25
b. What is the speedup Method 2 can achieve on the entire application?
1/((1 - 0.2) + (0.2 / 10 * 2) + (0.15 / (6 + 0.9))) = 1.44
Method 2 can speedup the application by 1.44
c. What is the speedup if both methods are applied?
1/((1 - 0.2 - 0.03) + (0.2 / 100 + 0.2) + (0.2 / 10 * 2) + (0.15 / (6 + 0.9))) = 2.012
Combined, the speedup of method 1 and 2 on the application is 2.012
d. After both methods are applied, what is the best speedup (or the upper bound) one can achieve if they continue to optimize the code that is already enhanced by the two methods? Note that the reference is the code after both methods are applied.
1/((1 - 0.77) + (0.77 / 2.012)) = 1.1044
After both methods are applied the best optimization that can be achieved is 1.1044