ECE550

PRACTICE Midterm

This is a full length practice midterm exam. If you want to take it at exam pace, give yourself 75 minutes to take the entire test. Just like the real exam, each question has a point value. There are 75 points from 6 question, so pace yourself accordingly.

Questions:

1. Combinatorial Logic: 15 pts
2. Sequential Logic: 10 pts
3. FSMs: 10 pts
4. Asm Programming: 20 pts
5. Datapaths: 10 pts
6. Memory Hierarchy: 10 pts

This is the solution set to the practice exam. The solutions appear in blue.

Question 1 Combinatorial Logic [15 pts]

Given the following truth-table:

a	b	c	x
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

1. Write the sum-of-product formula

   Answer:
   (NOT a AND b AND NOT c) or (a AND NOT b AND NOT c) or (a AND b AND NOT c)

2. Simplify the formula

   Answer:
   (a OR b) AND (NOT c)

3. Write VHDL that implements the formula

```
entity q1 is
  port (
    a : in  std_logic;
    b : in  std_logic;
    c : in  std_logic;
    x : out std_logic);
end q1;
architecture basic of q1 is
begin
```

x <= (a or b) and (not c);

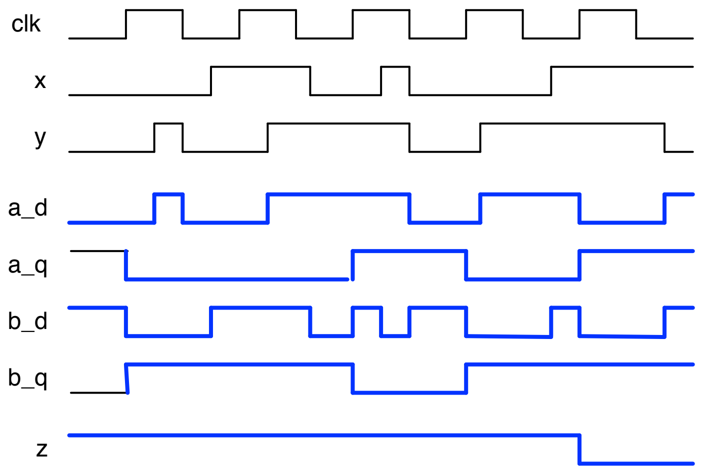
```
end basic;
```

Question 2 Sequential Logic [10 pts]

Consider the following VHDL fragment, in which `x`, and `y` are inputs, `z` is an output, and there are two DFFs (`a` and `b`) whose `d` inputs are `a_d` and `b_d` respectively, and whose `q` outputs are `a_q` and `b_q` respectively:

```
b_d <= a_q xor x;
a_d <= (b_q and a_q) xor y;
z   <= a_q xor b_q;
```

Complete the waveform below (assume that the DFFs are triggered by the rising edge of `clk`): Note that `a_q` is initially 1, and `b_q` is initially 0:

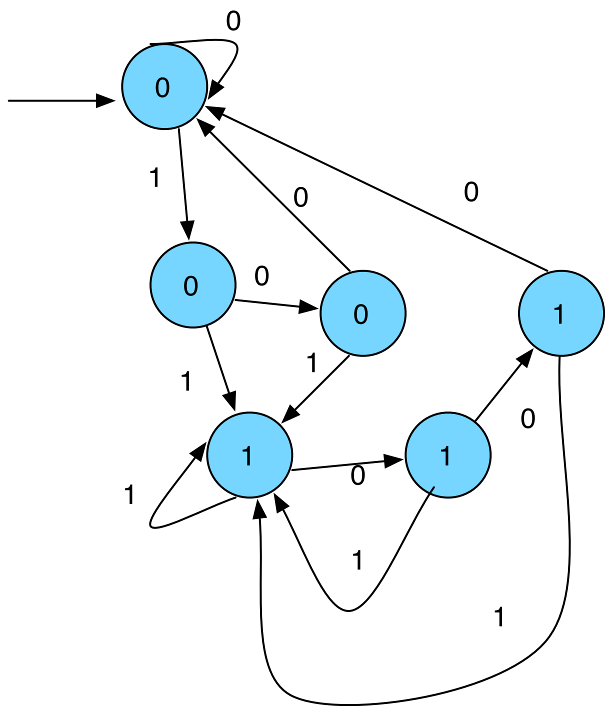


Question 3 FSMs [10 pts]

Draw a state machine diagram for a finite state machine which accepts a single bit input (either 0 or 1—you can just label each edge with 0 or 1). This state machine also has a single bit of output, which is initally 0.

- Whenever the FSM receives an input bit of 1 followed by anything other than two 0s (so 11, or 101) the output goes to 1.
- The output remains at 1 until 3 consecutive 0s are received by the FSM.
- Once three *consecutive* 0s are received by the FSM, the output returns to 0.
- The output now remains 0 until it again sees a 1 which is not followed (immediately) by two 0s.

Note that when the FSM sees a 1 not followed by two 0s, the output goes to 1 as soon as the situation is detected (e.g., on the second 1 of 11 or 101). Label each state with the bit it outputs. Be sure to indicate your start state (with an arrow to it from nowhere).



Question 4 Asm Programming [20 pts]

Translate the following C function to MIPS assembly. **Answer on the next 2 pages** where you have each C-code line written out for you with space to write the MIPS assembly for that line directly under it.

```
int countMatching(int * ptr, int n) {
  int count = 0;
  while (n > 0) {
      int x = testFn(*ptr);
      if (x != 0) {
         count++;
      }
      ptr++;
      n--;
  }
  return count;
}
```

Answer on next 2 pages

```
#int countMatching(int * ptr, int n) {

    addiu $sp, $sp, -32  # make space for stack frame

    sw $fp, 0($sp)       # save regs to stack

    sw $ra, 4($sp)

    sw $s0, 8($sp)

    sw $s1, 12($sp)

    sw $s2, 16($sp)

    addiu $fp, $sp, 28   # setup new fp

    move $s0, $a0        # move ptr into s0

    move $s1, $a1        # move n into s1


#  int count = 0;

        li $s2, 0        # count in s2

#  while (n > 0) {

.L_cm_while:

        blez $s1, .L_cm_end_while

#      int x = testFn(*ptr);

        lw $a0, 0($s0)  # put *ptr in $a0

        jal testFn      # call testFn

                        # x returned in v0

#      if (x != 0) {

        beqz $v0, .L_cm_endif

#         count++;
```

```
        addi $s2, $s2, 1
#      }
.L_cm_endif:
#      ptr++;
        addiu $s0, $s0, 4  # 4 because its an int *
#      n--;
        addi  $s1, $s1, -1
#  }
        b .L_cm_while
.L_cm_endwhile:


#  return count;
        move $v0, $s2      # put count in v0
        lw $s2, 16($sp)    # restore regs from stack
        lw $s1, 12($sp)
        lw $s0, 8($sp)
        lw $ra, 4($sp)
        lw $fp, 0($sp)
        addiu $sp, $sp, 32 # put sp back (release space)
        jr $ra             # return to caller
# }
```

Question 5 Datapaths [10 pts]

1. If a single cycle datapath has a clock period of 100ns, and is split into a multi-cycle data path with 5 stages, the clock will be slower than 20ns. Give *two* reasons why:

   -

     Answer:
     The stages will not be divided perfectly evenly

   -

     Answer:
     There is extra logic—the DFFs between stages—which is added

2. In the 1990s, processor design was about "performance at all cost," however now, performance must be balanced against other design considerations. Name two such design considerations, and for each one explain (briefly) why it is important.

   -

     Answer:
     Power. Power translates into heat. Heat must be cooled. Mobile devices and large data centers both have significant cooling constraints.

   -

     Answer:
     Energy. Energy directly money

Question 6 Memory Hierarchy [10 pts]

A processor uses 12-bit addresses, and has a 64-byte direct-mapped cache. The cache is organized in 8 sets, with 8-bytes per block. The contents of the cache are shown below:

	Tag	Data
Set 0	0F	`12 34 56 78 99 43 59 B3`
Set 1	15	`11 22 33 44 B4 C6 A9 D2`
Set 2	31	`2F 3E 4D 5C A0 C9 DA 9C`
Set 3	15	`37 83 A9 C0 3F F3 2C AE`
Set 4	3E	`FF EE DD CC F9 F8 7C 8D`
Set 5	1C	`1F 1E 1D 1C 1B 1A 10 19`
Set 6	0A	`F6 D7 C8 E9 00 01 02 03`
Set 7	0A	`77 66 55 AA 05 06 07 08`

Note that the least-significant byte in each block is on the right, and the most-significant is on the left (so in the first block, byte 0 is B3 and byte 7 is 12).

For each of the following addresses (a) split the address into tag/index/offset and write them in hex (b) determine if the access is a hit or miss (c) if the access is a hit, write the data that a 1-byte load to that address returns (if it is a miss, leave that field blank):

1. 0x55E = 0101 0101 1110
   - Tag: 01 0101 = 0x15
   - Index: 011 = 0x3
   - Offset: 110 = 0x6
   - Hit or miss? Hit
   - Value loaded: 0x83
2. 0x297 = 0010 1001 0111
   - Tag: 00 1010 = 0x0A
   - Index: 010 = 0x2
   - Offset: 111 = 0x7
   - Hit or miss? Miss
   - Value loaded: