

EE 201 Midterm Exam

Tufts University

7 March 2019

Name: _____

Question	Points
Numbers	4
Boolean equations and circuits	12
Number circuits	8
What's in the box?	4
Timing	4
Building proteins	5
VHDL	8
Total	45
Bonus	1

Instructions:

1. This examination contains 11 pages, including this page and the VHDL reference sheet.
2. The VHDL reference sheet is on the last 2 pages of this booklet. You may not use any other notes.
3. You may not use a calculator, slide rule, abacus, or other calculating devices besides your brain, your fingers, and this paper.
4. You have **seventy-five (75) minutes** to complete the examination. As a courtesy to your classmates, we ask that you not leave during the last ten minutes.
5. Write your answers in this booklet. If you have scratch work on another piece of paper which you'd like counted for partial credit, please make a note on the appropriate place in the booklet, and make sure to hand in the scratch work.

Question 1: Numbers

(a) [1 pt] Write 1110 0011 in hex.

(b) [1 pt] Write 1110 0011 in decimal, assuming it is an unsigned number.

(c) [2 pts] Write 1110 0011 in decimal, assuming it is a 2's complement signed number.

Question 2: Boolean equations and circuits

(a) [4 pts] Write a minimal boolean equation for this truth table.

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>Y</i>
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	X
1	0	0	0	X
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	X
1	1	1	1	X

- (b) [4 pts] Implement the boolean equation $AB + \overline{A}C$ using only NAND gates. Assume only A , B , and C are available, not their complements (i.e., A can be an input to your circuit, but \overline{A} cannot).

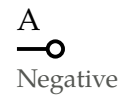
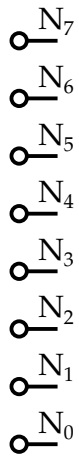
[4 pts] Re-implement the same boolean equation using a 4:1 multiplexer and an inverter.

Question 3: Number circuits

[8 pts] Draw a circuit below that analyzes the 8-bit signed 2's complement input N and produces three results:

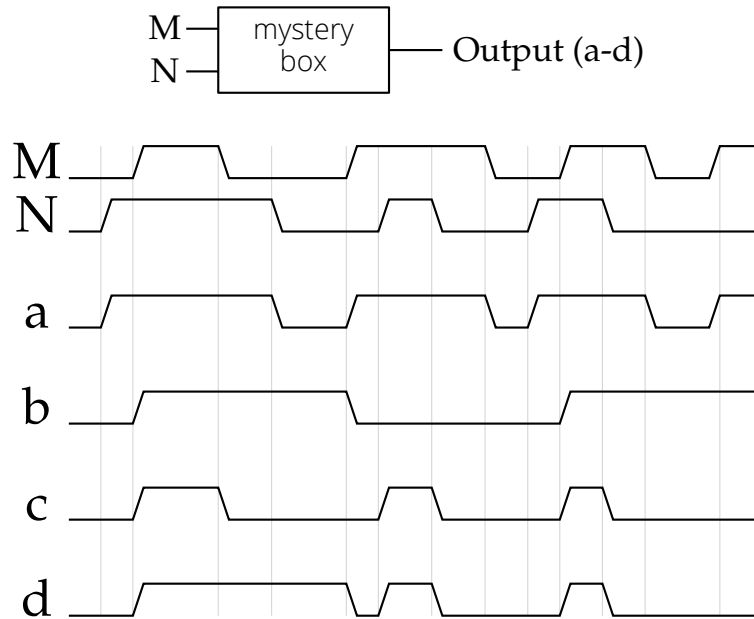
- (a) Output A should be high if the input number is negative, and low otherwise.
- (b) Output B should be high if the input number is zero, and low otherwise.
- (c) Output C should be high if the input number is greater than +31, and low otherwise.

N_7 is the most significant bit. You may use gates with as many inputs as you need (e.g., 4-input AND, 9-input NOR).



Question 4: What's in the box?

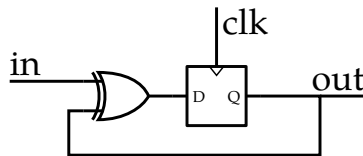
[5 pts] For each of the four waveforms below, identify what is in the box. It could be an SR latch, a D latch, a D flip-flop, or a single combinational logic gate (NAND, XOR, etc).



Question 5: Timing

[5 pts] Given the timing specifications below, what is the maximum clock speed (in Hz) that this circuit can run at? Assume the input can keep up with whatever rate you choose. *Leave your answer as a fraction if necessary.*

Gate	t_{pd} (ns)	t_{cd} (ps)
2-input XOR	23	12
D flip-flop Clock-Q	15	8
D flip-flop setup time	12 ns	
D flip-flop hold time	7 ns	



Question 6: Protein decoder

Once a “start codon” is found in a DNA strand, a ribosome begins matching codons to amino acids to build the protein. Two such amino acids are Asparagine (coded by either AAT or AAC) and Lysine (coded by either AAA or AAG).

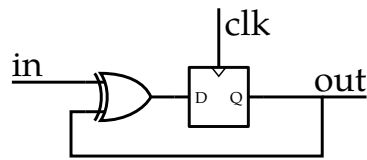
(a) [4 pts] Draw the state diagram for an FSM which detects Asparagine and Lysine.

- There should be two outputs, one for each amino acid.
- The output should go high for one cycle when the third nucleotide is received (i.e., not delayed) and a matching sequence is detected.
- Note that a repeated sequence (e.g., AAAAAAA) should only produce a Lysine once every third nucleotide.

(b) [1 pt] How many flip-flops do you need to implement this FSM?

Question 7: VHDL

- (a) [6 pts] Write VHDL code to implement the circuit from question 5 (shown below).



You'll need to fill in the appropriate port declarations as well as the architecture in the code skeleton below.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity parity is
  port (
    -- Add your declarations for the inputs and outputs here

  );
end parity;

architecture synth of parity is

begin
  -- Implement the circuit functionality


end architecture;
```


(b) [2 pts] Circle the VHDL constructs below which can be used in synthesizable logic.

- and
- case
- report
- unsigned
- wait

Question 8: Bonus

[1 pt] Many 64-bit systems actually only use 48 bits in certain parts of the processor. Approximately what is the largest unsigned number that can be represented with 48 bits? Express your answer in decimal, using scientific notation if it's helpful.

ES 4 VHDL reference sheet

r.2019.02.19

GRAY_ITALICS represent user-defined names or operations

keywords

www.ece.tufts.edu/es/4

Purple constructs are only available in VHDL 2008.

literals (constants)

```
-- This is a comment
/* Multi-line comment
   (VHDL 2008 only) */
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
```

} You almost always need these libraries;
just put this at the top of every file.

```
entity ENTITY_NAME is
  port(
    PORT_NAME : in std_logic; -- Single bit input
    ANOTHER    : out std_logic_vector(3 downto 0) -- 4-bit output
  );
end;
```

No semicolon on the last one!
Don't forget these semicolons!

```
architecture ARCH_NAME of ENTITY_NAME is
  -- Component declarations, if using submodules
  component SUB_ENTITY is
    port(
      -- Port list for the entity you're including
    );
  end component;

  -- Signal declarations, if using intermediate signals
  signal NAME : TYPE;
begin
  -- Architecture definition goes here
end;
```

Instantiate a submodule

```
INSTANCE_NAME : MODULE_NAME
  generic map (
    GENERIC => CONSTANT,
  );
  port map(
    PORT => VALUE,
    ANOTHER => LOCAL_SIGNAL
  );
```

Continuous assignments

```
RESULT_SIGNAL <= SIGNAL1 and SIGNAL2;  Also works for or, not, nand, nor, xor
RESULT_SIGNAL <= '1' when (SIGNAL1 = x"5") else '0';  Note '=' for comparison (not '==')
HIGHEST_BIT <= EIGHT_BIT_VEC(7);  Extract a single bit (7 is MSB, 0 is LSB)
TWO_BIT_VEC <= EIGHT_BIT_VEC(3 downto 2);  Extract multiple bits
SIX_BIT_VEC <= "000" & EIGHT_BIT_VEC(3 downto 2) & SINGLE_BYTE;  Concatenate
```

Types

std_logic Basic logic type, can take values 0, 1, X, Z (and others)

std_logic_vector (n downto m) Ordered group of std_logic

unsigned (n downto m) Like std_logic_vector, but preferred for numerically meaningful signals

signed (n downto m)

integer Can't be synthesized, but constants are integers by default

Literals

'0', '1', 'X', 'Z'

"00001010", x"0c" 8-bit binary, hex

9x"101" 3b"101" 7d"101"

9-bit hex 3-bit binary 7-bit decimal

5, 38, 2e10

Type conversion

to_unsigned(INTEGER, WIDTH) Use to_unsigned for unsigned constants before VHDL 2008.

unsigned(LOGIC_VECTOR) (Same things for signed)

std_logic_vector(UNSIGNED)


Process blocks

```
process (SENSITIVITY) is
begin
    -- if/case/print go here
end process;
```


If sensitivity includes:

all↓  Combinational logic

clk↑  Flip-flop / register

clk↑ + data↓  Latch

Nothing  Testbench (repeated evaluation)

Something else  Bad things you probably didn't want

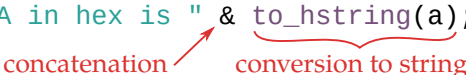
Specify all signals by name prior to VHDL 2008

Reporting stuff

assert *CONDITION* report "MESSAGE" severity error; Print message if condition is false

report "MESSAGE" severity error; Severity can be NOTE, WARNING, ERROR, FATAL
"FATAL" ends the simulation

report "A is " & to_string(a); Use image function prior to VHDL 2008

report "A in hex is " & to_hstring(a);

concatenation conversion to string

Writing to files (or stdout)

variable *BUF* : line; Declare buffer in process block

write(*BUF*, string'("MESSAGE")); Append message to buffer

writeline(output, *BUF*); Write buffer to stdout (like report, but just the text)

file *RESULTS* : text; Declare file handle in process block

file_open(*RESULTS*, "FILENAME", WRITE_MODE);

writeline(*RESULTS*, *BUF*);

If/else

```
if CONDITION then
    SIGNAL <= VALUE1;
elsif OTHER_CONDITION then
    SIGNAL <= VALUE2; Note spelling of "elsif"!
else
    SIGNAL <= VALUE3;
end if;
```

Case

```
case INPUT_SIGNAL is
    when VALUE1 => OPERATION1;
    when VALUE2 => OPERATION2;
    when others => DEFAULT;
end case;
```

Sequential logic

```
process (CLOCK) is
begin
    if rising_edge(CLOCK) then
        -- Clocked assignments go here
    end if;
end process;
```

For loop

```
for INDEXVAR in MIN to MAX loop
    -- loop body here
end loop;
```

To count down:

```
for INDEXVAR in MAX downto MIN loop
```

Enumerated types

```
type TYPENAME is (VAL1, VAL2, VAL3);
```

signal *NAME* : *TYPENAME*; Just like any other type