Digital Systems II - DGD 4

TA Office hours
SITE 2021 - 11-1 PM

Incomplete assignments

- According to VHDL code only IF and THEN is required.

Example

According to the following code:

```
process(a,b)
Begin
    If (a=b) then
        Eq <= '1';
    End if
End process
```

Eq will only be one if a = b.
Any other cases will make EQ retain the original value of EQ.

Fixed:

```
process(a,b)
Begin
    If (a=b) then
        Eq <= '1';
    Else
        Eq <= '0';
    End if
End process
```

Another example

```
process(a,b)
begin
    if (a>b) then
        gt <= '1';
    elsif (a=b) then
        eq <= '1';
    else
        lt <= '1';
    end if
end process
```

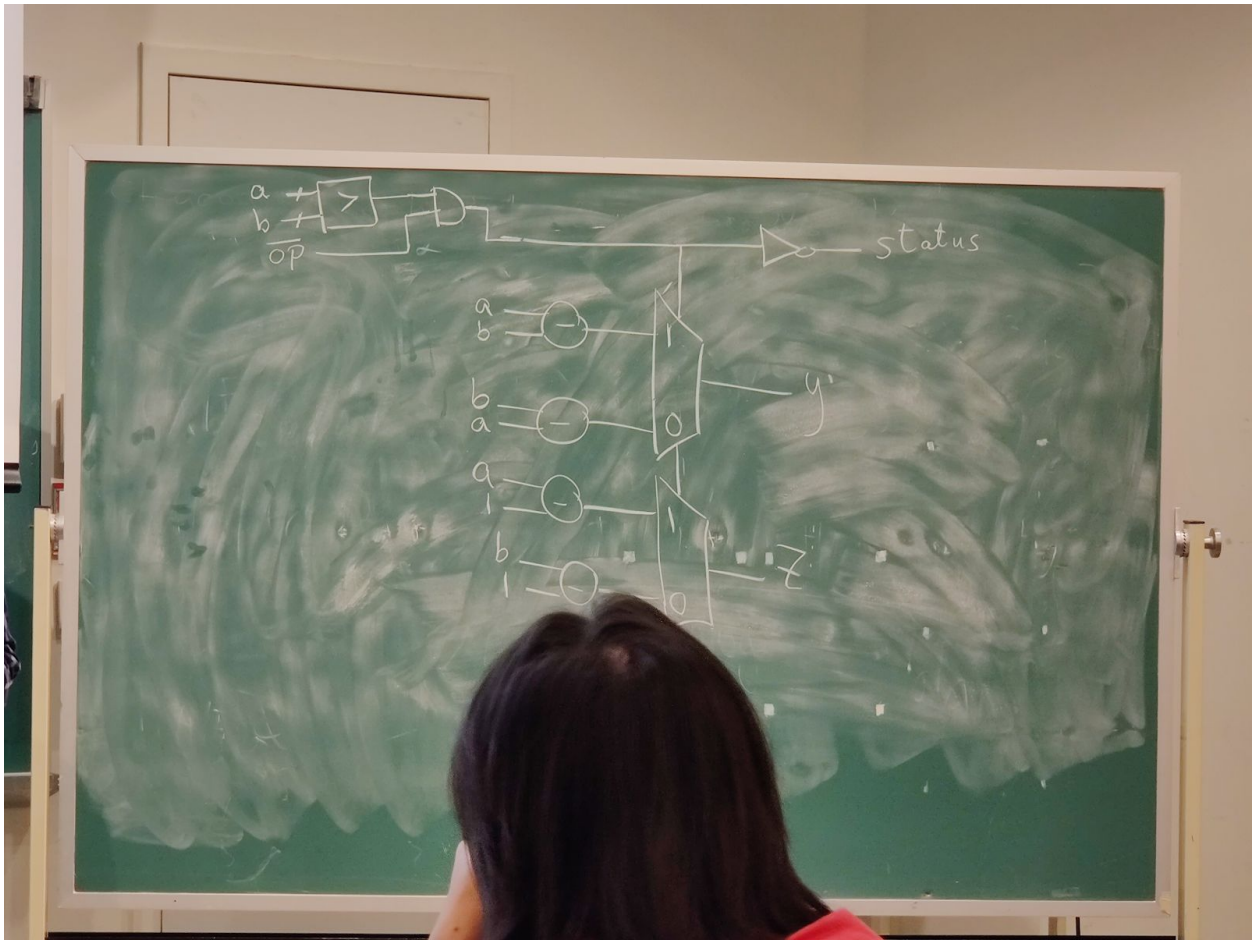
Fixed:

```
process(a,b)
begin
    gt <= '0';
    lt <= '0';
    eq <= '0';
    if (a>b) then
        gt <= '1';
    elsif (a=b) then
        eq <= '1';
    else
        lt <= '1';
    end if
end process
```

Question 1

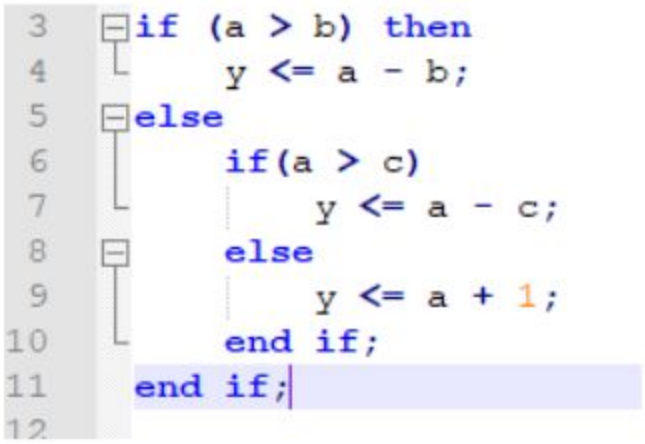
Draw the conceptual diagram of:

```
3   if (a > b and op="00") then
4       y <= a - b;
5       z <= a - 1;
6       status <= '0';
7   else
8       y <= b - a;
9       z <= b - 1;
10      status <= '1';
11  end if;
12
```



Question 2

Consider the code:



```
3   if (a > b) then
4       y <= a - b;
5   else
6       if(a > c)
7           y <= a - c;
8       else
9           y <= a + 1;
10      end if;
11  end if;
12
```

Rewrite the code using two concurrent conditional signal assignments:

```
signal temp1, temp2 : std_logic
tmp1 <= (a-c) when (a>c) else (a+1);
y <= (a-b) when (a>b) else temp1
```

Rewrite it with one concurrent conditional statement:

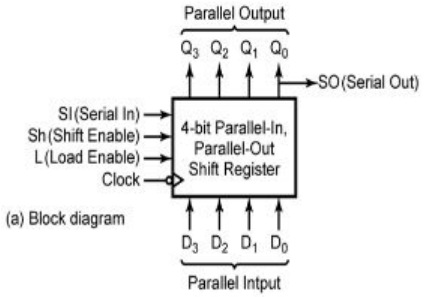
```
signal temp1, temp2 : std_logic
y <= (a-b) when (a>b) else (a-c) when (a>c) else (a+1);
```

Rewrite using a case statement:

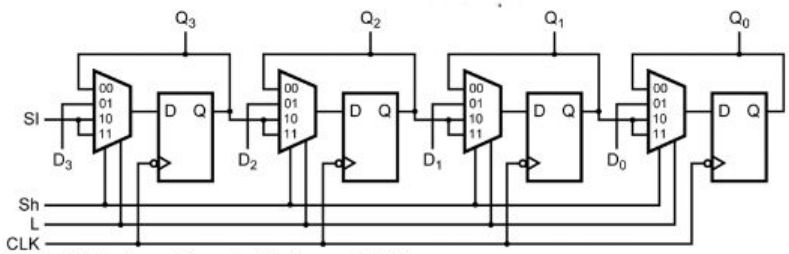
```
temp1 <= '1' when a>b else '0';
temp2 <= '1' when a>c else '0';

case(temp1&temp2) is
    when "00" =>
        y <= a+1;
    when "01" =>
        y <= a-c;
    when "10" =>
        y <= a-b;
    when others =>
        y <= a-b;
end case
```

Question 7

The following figure shows a parallel-in parallel out right shift register, Part a is the block diagram and part b is the implementation using flip flops and multiplexers



(a) Block diagram



(b) Implementation using flip-flops and MUXes

Inputs		Next State				Action
Sh (Shift)	Ld (Load)	Q_3^+	Q_2^+	Q_1^+	Q_0^+	
0	0	Q_3	Q_2	Q_1	Q_0	no change
0	1	D_3	D_2	D_1	D_0	load
1	X	SI	Q_3	Q_2	Q_1	right shift