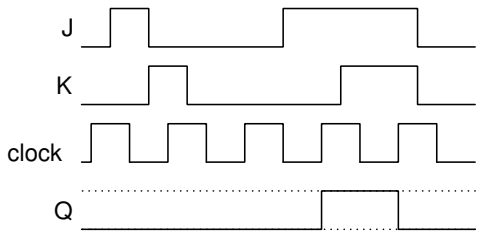
ELEX 2117 : Digital Techniques 2
2020 Winter Term

Solutions to Midterm Exam

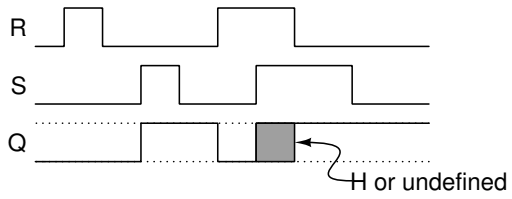
Midterm 1

Question 1

There were two timing diagrams, one for a J-K flip-flop:



and one for an SR flip-flop:



and two truth tables, one for a T flip-flop:

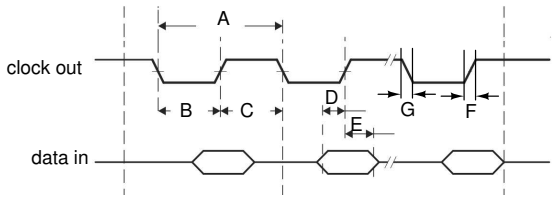
T	$\overline{\text{CLR}}$	clk	Q	$\overline{Q}$
x	0	**x**	**0**	**1**
0	1	↑	Q_0	$\overline{Q_0}$
1	1	↑	$\overline{Q_0}$	Q_0
0	**1**	↑	Q_0	$\overline{Q_0}$

and one for a J-K flip-flop:

J	K	$\overline{\text{CLR}}$	clk	Q	$\overline{Q}$
x	**x**	0	↑	**0**	**1**
0	**1**	1	↑	0	1
1	**0**	1	↑	1	0
1	**1**	1	↑	$\overline{Q_0}$	Q_0

Question 2

There were two timing diagrams with different labels. Only one is shown here:



On this device the clock signal is an output from the microcontroller. It is also an input to the flip-flops in the peripheral device and in the microcontroller. These flip-flops store the data transmitted over the interface[1].

The **`clock out`** signal is specified as an output so timing specifications between two points on the clock (A, B, C, G and F above) will be guaranteed specifications.

The **`data in`** signal is an input so timing specifications for this signal (D, E) relative to a clock edge will be requirements.

letter	**name**	**var.**	**R/G**
A	period	T	G
$\frac{\texttt{C}}{\texttt{B+C}}$	duty cycle	t_{DC}	G
D	setup time	t_{SU}	R
E	hold time	t_H	R
F	rise time	t_R	G
G	fall time	t_F	G

Question 3

There were two versions of the question with different values. In each case the minimum clock period is $T_{min} = t_{CO} + t_{PD} + t_{SU}$ and the maximum clock period is $f_{max} = 1/T_{min}$.

[1] Since we had not covered this situation before, specification D will also be marked correct if specified as a propagation delay. Specification E cannot be a propagation delay because the change in the output happens before the change in the input.

- for $t_{CO} = 3$ ns, $t_{SU} = 5$ ns and $t_{PD} = 8$ ns, $T_{min} = 3+8+5 = 16$ ns and $f_{max} = 1/16$ GHz $= 62.5$ MHz
- for $t_{CO} = 5$ ns, $t_{SU} = 2$ ns and $t_{PD} = 3$ ns, $T_{min} = 5+3+2 = 10$ ns and $f_{max} = 1/10$ GHz $= 100$ MHz

Question 4

The figure shows a circuit that adds 1 to `Q` if `Q` is less than 3, otherwise the value is unchanged. Thus if the value of `Q` is less than 3 it increases by 1 each clock edge until it reaches the value 3. There were two versions of the question, with initial values of 1 and 0.

If the initial value is 1 then the next values in the sequence will be 2, 3, 3 and 3. If the initial value is 0, the subsequent values will be 1, 2, 3, 3.

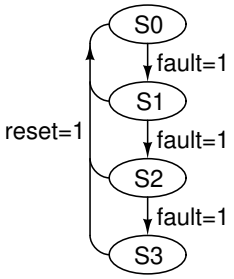
Midterm 2

Question 1

There were two versions of this question, one turning on the warning after two faults, the other three. The three-fault version simply has one more state and is shown below.

(a) The hint says to use the number of faults since the most recent reset as the state. We'll name them `S0`, `S1`, `S2` and (optionally) `S3`.

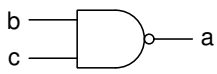
(b) The state transition diagram simply moves from one state to the next higher count when `fault` is asserted and to `S0` when `reset` is asserted:



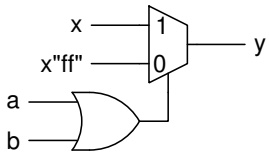
(c) In this case the output is only high in state `S3` so the expression, in VHDL format, for the output sould be: `warn <= '1' when state = S3 else '0' ;`.

Question 2

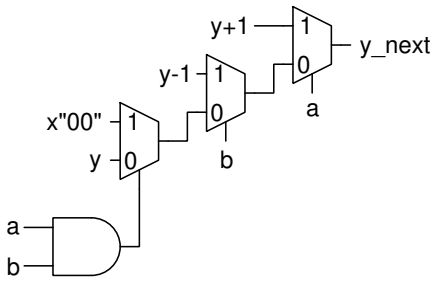
There were two versions of this question, differing only in the order.



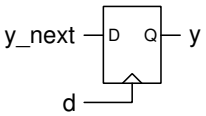

```
a <= b nand c ;
```




```
y <= x when a = '1' or b = '1' else
     x"ff" ;
```



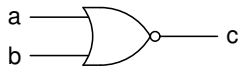

```
y_next <= y+1 when a = '1' else
     y-1 when b = '1' else
     x"00" when a = '1' and b = '1' else
     y ;
```



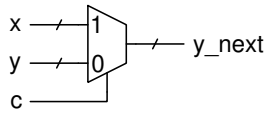

```
y <= y_next when rising_edge(d) ;
```

Question 3

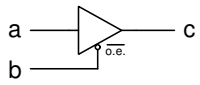
There were two version of this question differing only in the order of the block diagrams. The VHDL for each diagram is shown below:



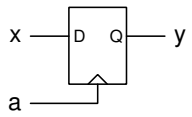

```
c <= a nor b ;
-- or
c <= not ( a or b ) ;
```



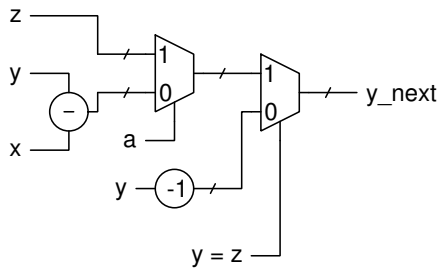

```
y_next <= x when c = '1' else y ;
```




```
c <= a when not ( b = '1' ) else 'Z' ;
```




```
y <= x when rising_edge(a) ;
```




```
-- the simplest solution is to negate the control
-- input of the highest-priority multiplexer
-- the condition could be:
--    y /= z             -- best
--    not ( y = z )      -- OK
--    ( y = z ) = false  -- should work

y_next <=
       y - 1 when y /= z else
       z when a = '1' else
       y - x ;

-- an alternate solution is to use logic that
-- ensures the y=z condition has priority

y_next <=
       z   when y = z and a = '1' else
       y-x when y = z and a = '0' else
       y-1 ;  -- if y /= z

-- another solution that was accepted because it
-- was not explicitly disallowed is to use an
-- intermediate variable:

tmp <= z when a = '1' else y-x ;
y_next <= tmp when y = z else y-1 ;
```