Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

CEG 3155: Digital Systems II

Final Exam

Note: Closed book exam. No calculators are allowed.

Name: _____ **Student Number:** _____

Question	Max. Points	Points
1	15	
2	15	
3	10	
4	30	
5A or 5B	20	
6A or 6B	10	
Total	**100**	**100**

Question 1 (5 points each, total 15 points)

Short questions

1. A) Which of these operations is legal in VHDL?

```
SIGNAL a: BIT;
SIGNAL b: BIT_VECTOR(7 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL e: INTEGER RANGE 0 TO 255;


...
a <= b(5);
b(0) <= a;
c <= d(5);
d(0) <= c;
a <= c;
b <= d;

e <= b;
e <= d;
```

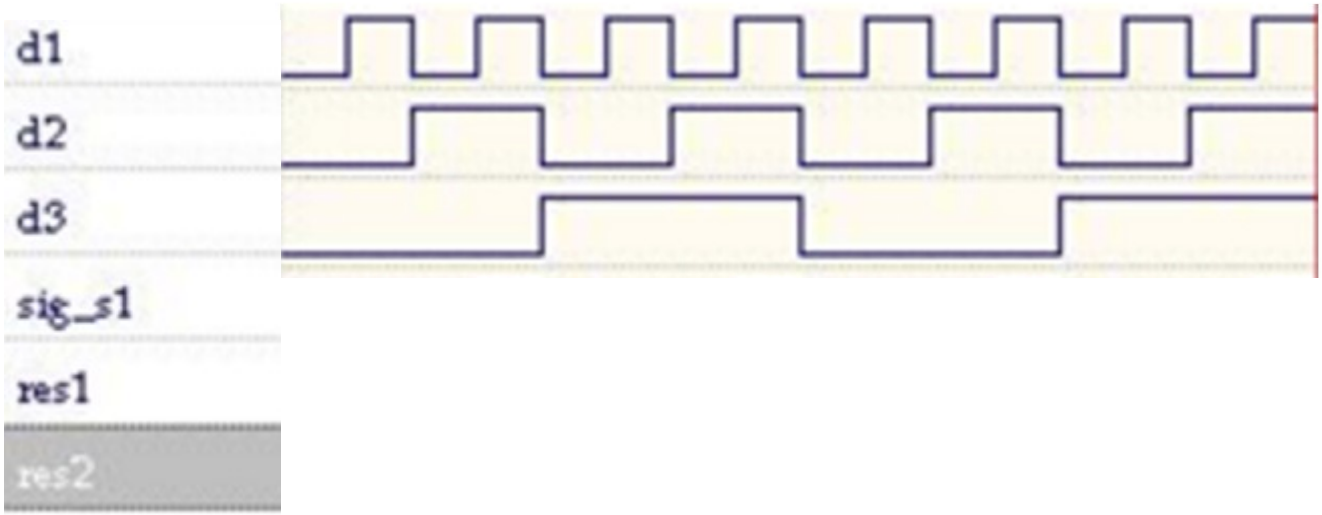
Solution:

```
SIGNAL a: BIT;
SIGNAL b: BIT_VECTOR(7 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL e: INTEGER RANGE 0 TO 255;
...
a <= b(5);     -- legal (same scalar type: BIT)
b(0) <= a;     -- legal (same scalar type: BIT)
c <= d(5);     -- legal (same scalar type: STD_LOGIC)
d(0) <= c;     -- legal (same scalar type: STD_LOGIC)
a <= c;        -- illegal (type mismatch: BIT x STD_LOGIC)
b <= d;        -- illegal (type mismatch: BIT_VECTOR x
               -- STD_LOGIC_VECTOR)
e <= b;        -- illegal (type mismatch: INTEGER x BIT_VECTOR)
e <= d;        -- illegal (type mismatch: INTEGER x
               -- STD_LOGIC_VECTOR)
```

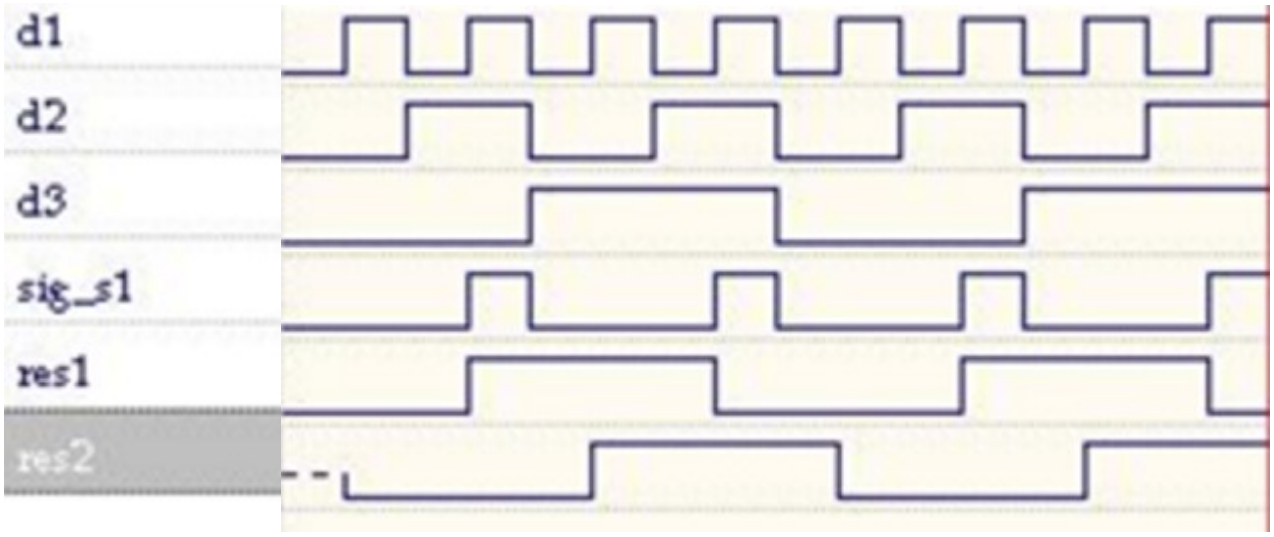
B) For the following code, draw the wafeform for sig_s1, res1 and res2

```
library ieee;
use ieee.std_logic_1164.all;
entity sig_var is
port(       d1, d2, d3:       in std_logic;
       res1, res2:       out std_logic);
end sig_var;
architecture behv of sig_var is
```

```
signal sig_s1: std_logic;
begin

  proc1: process(d1,d2,d3)

    variable var_s1: std_logic;

  begin
       var_s1 := d1 and d2;
       res1 <= var_s1 xor d3;
  end process;

proc2: process(d1,d2,d3)
  begin
       sig_s1 <= d1 and d2;
       res2 <= sig_s1 xor d3;
  end process;
end behv;
```

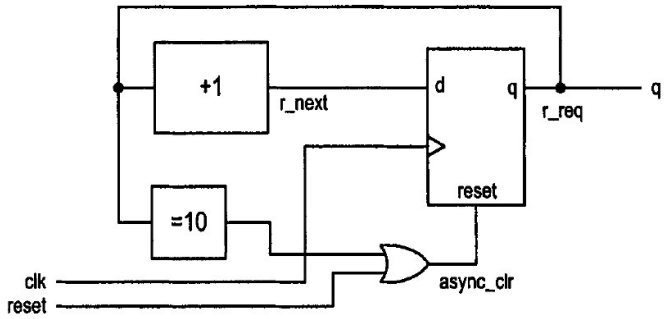


Solution:



C)

a. To design mod-10 counter, you need *n*-bit register. What is *n*?
b. For the implementation of the counter below, let setup time of the register to be 0.5 ns and the propagation time from clock to output to be 1 ns. The propagation delays of other components are: incrementor 5ns, comparator 3ns and multiplexer (and other combinational components if any) 0.75 ns. Determine the maximum clock rate.



Solution:

N=4

Tclk=0.5ns+1ns+5ns=6.5ns

f_clk=1/6.5ns=153.8MHz

Question 2 (15 marks)

Write behavioral VHDL code that represents a 24-bit up/down-counter with parallel load and asynchronous reset.

Solution

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY prob7_21 IS
    PORT ( R                  : IN     STD_LOGIC_VECTOR(23 DOWNTO 0) ;
           Clock, Resetn, L, U : IN     STD_LOGIC ;
           Q                  : BUFFER STD_LOGIC_VECTOR(23 DOWNTO 0) ) ;
END prob7_21 ;

ARCHITECTURE Behavior OF prob7_21 IS
BEGIN
    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            Q <= (OTHERS => '0') ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            IF L = '1' THEN
                Q <= R ;
            ELSIF U = '1' THEN
                Q <= Q+1 ;
            ELSE
                Q <= Q-1 ;
            END IF ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Question 3 (10 marks)

Represent the FSM in Figure 1 in form of an ASM chart.

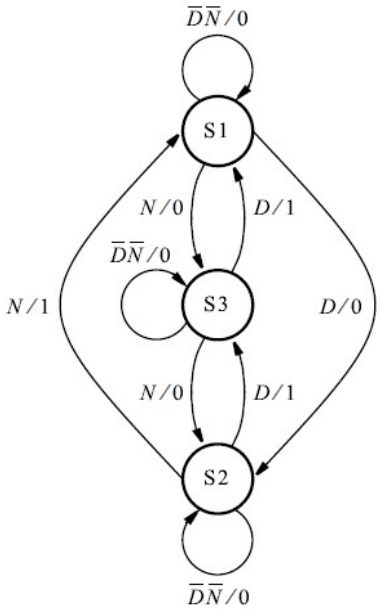
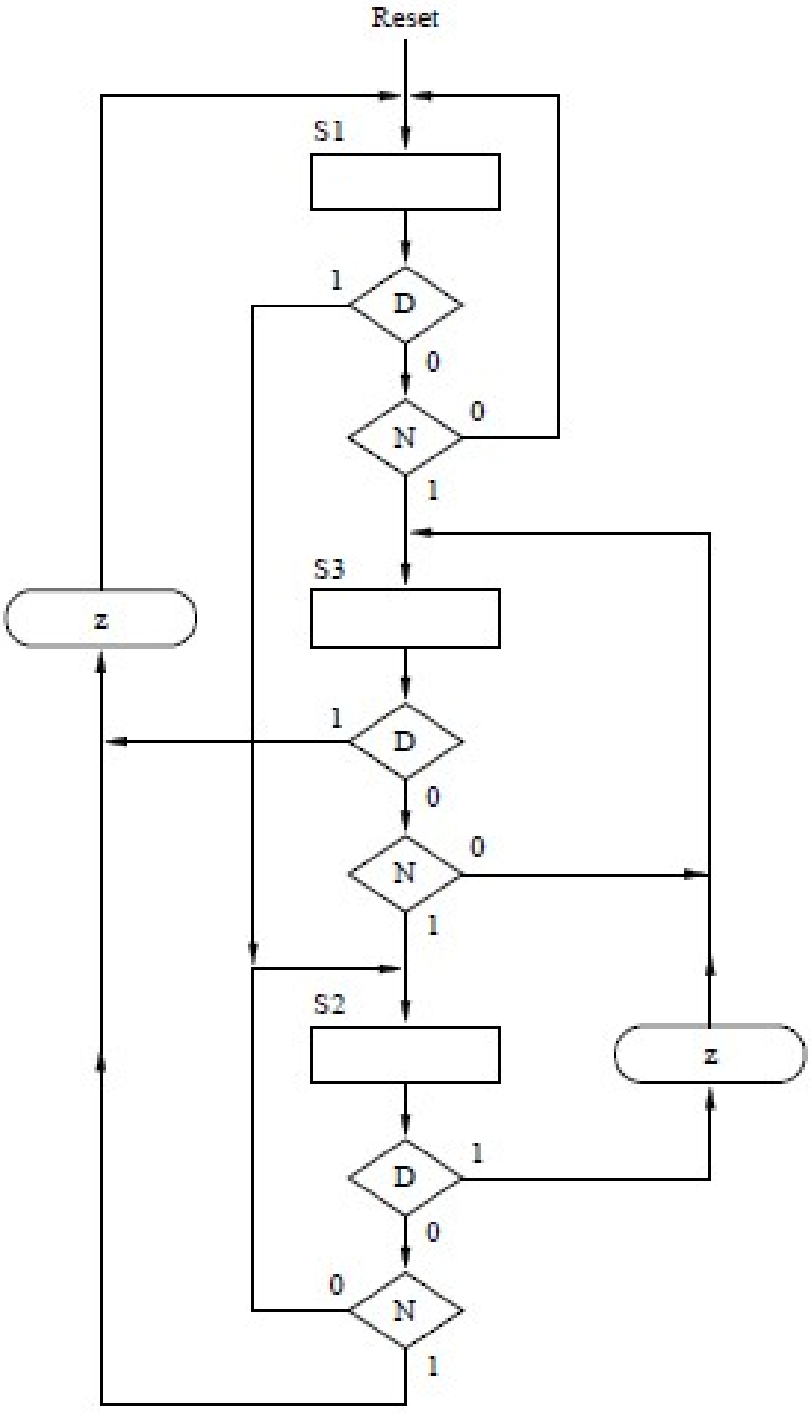


Figure 1 **Mealy-type FSM for Question 2.**

Solution

An ASM chart for the FSM in Figure 2 is

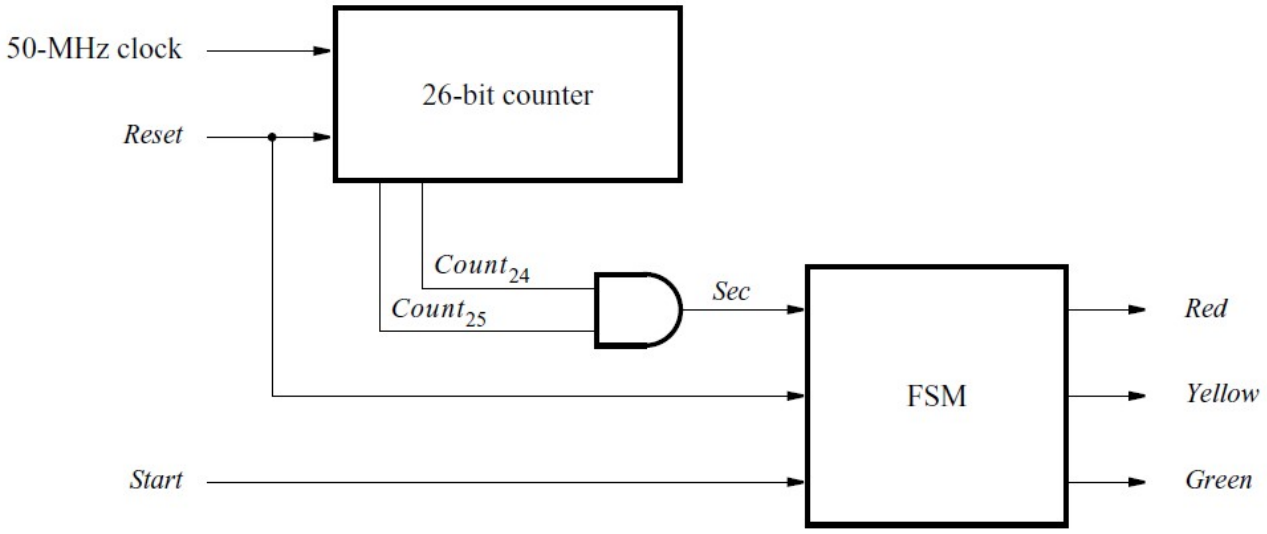


Question 4 (30 marks)

a) Design a circuit for controlling the lights used to start a race, which works as follows. There are three inputs: *Reset*, *Start* and *Clock*. There are three outputs: *Red*, *Yellow* and *Green*, which turn on the lights. Only one light can be on at any time. The *Reset* signal forces the circuit into a state in which the red light is turned on. When the *Start* signal is activated, the red light stays on for at least one second longer, then the yellow light is turned on. The yellow light stays turned on about one second and then the green light is turned on. The green light stays on for at least three seconds and then the red light is turned on and the circuit returns to its reset state. Assume that you can use 1 MHz clock.

Solution

A suitable circuit is



The two most-significant bits of the 26-bit counter become equal to 1 after the elapsed time of just over a second. They are used to generate a "clock" signal, called *Sec*, for the finite state machine. At each positive edge of the *Sec* signal the FSM changes state as follows

Present state	Next state		Output light
	$Start = 0$	$Start = 1$	
A	A	B	Red
B	C	C	Red
C	D	D	Yellow
D	E	E	Green
E	F	F	Green
F	A	A	Green

b) Write VHDL code that can be used to synthesize the circuit specified in Question 8.46.

Solution

The control circuit can be specified by the following VHDL code:

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY prob8_47 IS
    PORT ( Clock, Resetn, Start : IN STD_LOGIC ;
           Red, Yellow, Green : OUT STD_LOGIC ) ;
END prob8_47 ;

ARCHITECTURE Behavior OF prob8_47 IS
    TYPE State_type IS (A, B, C, D, E, F) ;
    SIGNAL y : State_type ;
    SIGNAL Count : STD_LOGIC_VECTOR(25 DOWNTO 0) ;
    SIGNAL Sec : STD_LOGIC ;
BEGIN
    PROCESS ( Resetn, Sec )
    BEGIN
        IF Resetn = '0' THEN
            y <= A ;
        ELSIF (Sec'EVENT AND Sec = '1') THEN
            CASE y IS
                WHEN A =>
                    IF Start = '0' THEN
                        y <= A ;
                    ELSE
                        y <= B ;
                    END IF ;
                WHEN B =>  y <= C ;
                WHEN C =>  y <= D ;
                WHEN D =>  y <= E ;
                WHEN E =>  y <= F ;
                WHEN F =>  y <= A ;
            END CASE ;
        END IF ;
    END PROCESS ;
    PROCESS (Resetn, Clock)
    BEGIN
        IF Resetn = '0' THEN
            Count <= (OTHERS => '0') ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            Count <= Count + 1 ;
        END IF ;
    END PROCESS ;
    Sec <= Count(25) AND Count(24) ;
    Red <= '1' WHEN ((y = A) OR (y = B)) ELSE '0' ;
    Yellow <= '1' WHEN y = C ELSE '0' ;
    Green <= '1' WHEN ((y = D) OR (y = E) OR (y = F)) ELSE '0' ;
END Behavior ;
```

Question 5A (15 marks)

Implement in VHDL the equation (a*b*c*data_in) using pipelining techniques we studied in the class. Note that a, b and c are constants here and the variable 'data_in' changes every clock cycle. The result of the calculation will be available at the port names 'data_out' also at every clock cycle. One multiplication operation takes less time than the clock period. For VHDL of 2-input multiplier please use: z<= y * x;

Solution

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity pipelined is
port (Clk : in std_logic;
        data_in : in integer;
        data_out : out integer;
        a,b,c : in integer
     );
end pipelined;

architecture Behavioral of pipelined is

signal i,data,result : integer := 0;
signal temp1,temp2 : integer := 0;

begin

data <= data_in;
data_out <= result;

--process for calcultation of the equation.
PROCESS(Clk)
BEGIN
    if(rising_edge(Clk)) then
    --Implement the pipeline stages using a for loop and case statement.
    --'i' is the stage number here.
    --The multiplication is done in 3 stages here.
    --See the output waveform of both the modules and compare them.
        for i in 0 to 2 loop
```

```
            case i is
                when 0 => temp1 <= a*data;
                when 1 => temp2 <= temp1*b;
                when 2 => result <= temp2*c;
                when others => null;
            end case;
        end loop;
    end if;
END PROCESS;

end Behavioral;
```

Question 5B (20 marks)

Derive the minimal flow table that specifies the same functional behavior as the flow table in Figure 3.

Present state	Next state $w_2w_1 = 00$	01	10	11	Output z
A	(A)	B	C	–	0
B	D	(B)	–	–	0
C	P	–	(C)	–	0
D	(D)	E	F	–	0
E	G	(E)	–	–	0
F	M	–	(F)	–	0
G	(G)	H	I	–	0
H	J	(H)	–	–	0
I	A	–	(I)	–	1
J	(J)	K	L	–	0
K	A	(K)	–	–	1
L	A	–	(L)	–	1
M	(M)	N	O	–	0
N	A	(N)	–	–	1
O	A	–	(O)	–	1
P	(P)	R	S	–	0
R	T	(R)	–	–	0
S	A	–	(S)	–	1
T	(T)	U	V	–	0
U	A	(U)	–	–	1
V	A	–	(V)	–	1

Figure 3 Flow table for Question 5

Present state	Next state				z
	$w_2w_1 = 00$	01	10	11	
A	(A)	B	C	–	0
B	D	(B)	–	–	0
C	G	(C)	(C)	–	0
D	(D)	C	F	–	0
F	J	(F)	(F)	–	0
G	(G)	F	I	–	0
I	A	(I)	(I)	–	1
J	(J)	I	I	–	0

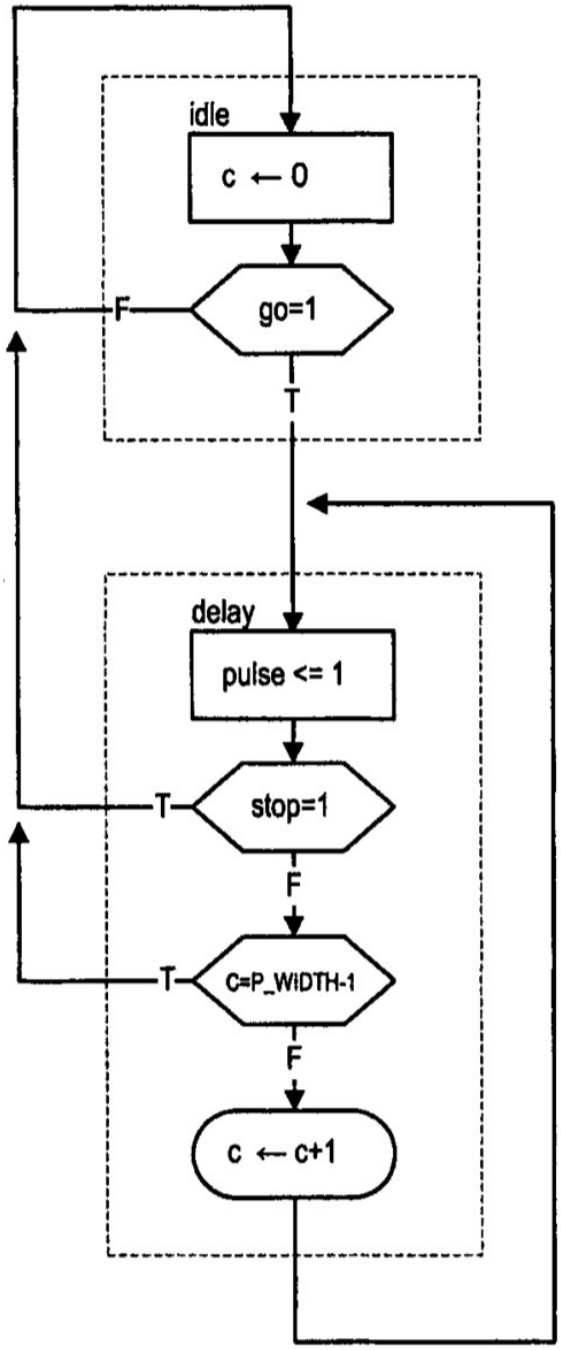
Question 6A (10 marks)

A one-shot pulse generator is a circuit that generates a single fixed-width pulse upon activation of a trigger signal. We assume that the width of the pulse is N clock cycles. The detailed specifications are listed below.

- There are three input signals, N, go and stop, and one output signal, pulse.
- The go signal is the trigger signal that is usually asserted for only one clock cycle. During normal operation, assertion of the go signal activates the pulse signal for five clock cycles.
- If the go signal is asserted again during this interval, it will be ignored.
- If the stop signal is asserted during this interval, the pulse signal will be cut short and return to '0'.

Show ASMD chart for the one-shot pulse generator.

Solution:



Question 6B (10 marks)

Find a hazard-free minimum-cost implementation of the function

$f(x1, \ldots, x4) = \sum m(0, 4, 11, 13, 15) + D(2, 3, 5, 10)$

Reference Card

REFERENCE CARD

Revision 2.1

()	Grouping	[ ]	Optional
{}	Repeated	|	Alternative
bold	As is	CAPS	User Identifier
italic	VHDL-93	c	commutative

b	::=	BIT
bv	::=	BIT_VECTOR
u/l	::=	STD_ULOGIC/STD_LOGIC
uv	::=	STD_ULOGIC_VECTOR
lv	::=	STD_LOGIC_VECTOR
un	::=	UNSIGNED
sg	::=	SIGNED
in	::=	INTEGER
na	::=	NATURAL
sm	::=	SMALL_INT (subtype INTEGER range 0 to 1)

1. IEEE's STD_LOGIC_1164

1.1. Logic Values

'U'	Uninitialized
'X'/'W'	Strong/Weak unknown
'0'/'L'	Strong/Weak 0
'1'/'H'	Strong/Weak 1
'Z'	High Impedance
'-'	Don't care

1.2. Predefined Types

STD_ULOGIC	Base type
Subtypes:	
STD_LOGIC	Resolved STD_ULOGIC
X01	Resolved X, 0 & 1
X01Z	Resolved X, 0, 1 & Z
UX01	Resolved U, X, 0 & 1
UX01Z	Resolved U, X, 0, 1 & Z

STD_ULOGIC_VECTOR(na **to** | **downto** na)
Array of STD_ULOGIC
STD_LOGIC_VECTOR(na **to** | **downto** na)
Array of STD_LOGIC

1.3. Overloaded Operators

Description	Left	Operator	Right
bitwise-and	u/l,uv,lv	**and**, **nand**	u/l,uv,lv
bitwise-or	u/l,uv,lv	**or**, **nor**	u/l,uv,lv
bitwise-xor	u/l,uv,lv	**xor**, ***xnor***	u/l,uv,lv
bitwise-not		**not**	u/l,uv,lv

1.4. Conversion Functions

From	To	Function
u/l	b	**TO_BIT**(from[, xmap])
uv,lv	bv	**TO_BITVECTOR**(from[, xmap])
b	u/l	**TO_STDULOGIC**(from)
bv,uv	lv	**TO_STDLOGICVECTOR**(from)
bv,lv	uv	**TO_STDULOGICVECTOR**(from)

2. IEEE's NUMERIC_STD

2.1. Predefined Types

UNSIGNED(na **to** | **downto** na) Array of STD_LOGIC
SIGNED(na **to** | **downto** na) Array of STD_LOGIC

2.2. Overloaded Operators

Left	Op	Right	Return
	abs	sg	sg
	-	sg	sg
un	**+,-,*,/,rem,mod**	un	un
sg	**+,-,*,/,rem,mod**	sg	sg
un	**+,-,*,/,rem,mod** c	na	un
sg	**+,-,*,/,rem,mod** c	in	sg
un	**<,>,<=,>=,=,/=**	un	bool
sg	**<,>,<=,>=,=,/=**	sg	bool
un	**<,>,<=,>=,=,/=** c	na	bool
sg	**<,>,<=,>=,=,/=** c	In	bool

2.3. Predefined Functions

SHIFT_LEFT(un, na)	un
SHIFT_RIGHT(un, na)	un
SHIFT_LEFT(sg, na)	sg
SHIFT_RIGHT(sg, na)	sg
ROTATE_LEFT(un, na)	un
ROTATE_RIGHT(un, na)	un
ROTATE_LEFT(sg, na)	sg
ROTATE_RIGHT(sg, na)	sg
RESIZE(sg, na)	sg
RESIZE(un, na)	un
STD_MATCH(u/l, u/l)	bool
STD_MATCH(ul, ul)	bool
STD_MATCH(lv, lv)	bool
STD_MATCH(un, un)	bool
STD_MATCH(sg, sg)	bool

2.4. Conversion Functions

From	To	Function
un,lv	sg	**SIGNED**(from)
sg,lv	un	**UNSIGNED**(from)
un,sg	lv	**STD_LOGIC_VECTOR**(from)
un,sg	in	**TO_INTEGER**(from)
na	un	**TO_UNSIGNED**(from, size)
in	sg	**TO_SIGNED**(from, size)

3. IEEE's NUMERIC_BIT

3.1. Predefined Types

UNSIGNED(na **to** | **downto** na) Array of BIT
SIGNED(na **to** | **downto** na) Array of BIT

3.2. Overloaded Operators

Left	Op	Right	Return
	abs	sg	sg
	-	sg	sg
un	**+,-,*,/,rem,mod**	un	un
sg	**+,-,*,/,rem,mod**	sg	sg
un	**+,-,*,/,rem,mod** c	na	un
sg	**+,-,*,/,rem,mod** c	in	sg
un	**<,>,<=,>=,=,/=**	un	bool
sg	**<,>,<=,>=,=,/=**	sg	bool
un	**<,>,<=,>=,=,/=** c	na	bool
sg	**<,>,<=,>=,=,/=** c	in	bool

3.3. Predefined Functions

SHIFT_LEFT(un, na)	un
SHIFT_RIGHT(un, na)	un
SHIFT_LEFT(sg, na)	sg
SHIFT_RIGHT(sg, na)	sg
ROTATE_LEFT(un, na)	un
ROTATE_RIGHT(un, na)	un
ROTATE_LEFT(sg, na)	sg
ROTATE_RIGHT(sg, na)	sg
RESIZE(sg, na)	sg
RESIZE(un, na)	un

3.4. Conversion Functions

From	To	Function
un,bv	sg	**SIGNED**(from)
sg,bv	un	**UNSIGNED**(from)
un,sg	bv	**BIT_VECTOR**(from)
un,sg	in	**TO_INTEGER**(from)
na	un	**TO_UNSIGNED**(from)
in	sg	**TO_SIGNED**(from)

VHDL Entity/Architecture

Entity:

```
entity entity_name is
        port    ( signal_names : mode signal_type;
                  signal_names : mode signal_type;
                  . . .
                  signal_names : mode signal_type);
end entity_name;
```

Architecture:

```
architecture architecture_name of entity_name is
        type declarations
        signal declarations
        constant declarations
        function definitions
        procedure definitions
        component declarations
begin
        concurrent_statement;
        . . .
        concurrent_statement;
end architecture_name;
```

VHDL Predefined Types:

bit	character	severity_level
bit_vector	integer	string
boolean	real	time

VHDL Operators:

arithmetic:		+, -, *, /
logical:		**and, or, xor, nand, nor, xnor, not**
relational:		=, /=, <, <=, >, >=
shift left/right logical:		**sll, srl**
shift left/right arithmetic:		**sla, sra**
rotate left/right logical:		**rol, ror**
other:	concatenation:	**&**
	exponentiation:	**
	remainder:	**rem**
	division modulo:	**mod**

VHDL Concurrent Signal Assignments:

Simple Signal Assignment:

```
signal_name <= expression;
```

Conditional Signal Assignment:

```
signal_name <= expression when boolean_expression else
               expression when boolean_expression else
               . . .
               expression when boolean_expression else
               expression;
```

Selected Signal Assignment:

```
with expression select
        signal_name <= signal_value when choices,
                       signal_value when choices,
                       . . .
                       signal_value when others;
```

Process:

```
process ( signal_name, . . ., signal_name )
begin
...
end process;
```

VHDL Component Declaration:

```
component component_name
        port    ( signal_names : mode signal_type;
                  signal_names : mode signal_type);
end component;
```

VHDL if Statement:

```
if boolean_expression then sequential_statement
end if;
```

VHDL Process:

```
process ( signal_name, . . ., signal_name )
        type declarations
        variable declarations
        constant declarations
        function definitions
        procedure definitions
begin
        sequential_statement
        . . .
        sequential_statement
end process;
```

VHDL Array Declarations:

```
type type_name is array (start to end) of element_type;
type type_name is array (start downto end)of element_type;
type type_name is array (range_type) of element_type;
type type_name is array (range_type range start to end) of
        element_type;
type type_name is array (range_type range start downto end)
        of element_type;
```

VHDL CONSTANT Declaration:

```
CONSTANT const_name : signal_type := expression;
```

VHDL Component Instantiation:

```
label: component_name port map (port_signal_name_1 =>
        signal_1, port_signal_name_2 => signal_2, . . . ,
        port_signal_name_n => signal_n );
```

```
if boolean_expression then sequential_statement
elsif boolean_expression then sequential_statement
        . . .
elsif boolean_expression then sequential_statement
else sequential_statement
end if;
```

VHDL Case Statement:

```
case expression is
        when choices => sequential_statements
        . . .
        when choices => sequential_statements
end case;
```

for loop:

```
for identifier in range loop
        sequential_statement
        . . .
        sequential_statement
end loop;
```