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

COURS: CEG3555/ CEG3155

PROFESSEURS: Mohamed Ibrahim
Miodrag Bolic

TRIMESTRE: Automne 2017

DATE: 16 décembre 2017

TEMPS: 14h00 - 17h00

Systèmes numériques II

Solution d'examen final

Note: Examen à livre fermé. Aucune calculatrice n'est autorisée.

Nom: _____ **Numéro d'étudiant: _____**

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

Nombre de pages totales: 15

Question 1 (5 points chacun, total 15 points)

Courtes questions

1. A) Laquelle de ces opérations est légale en VHDL?

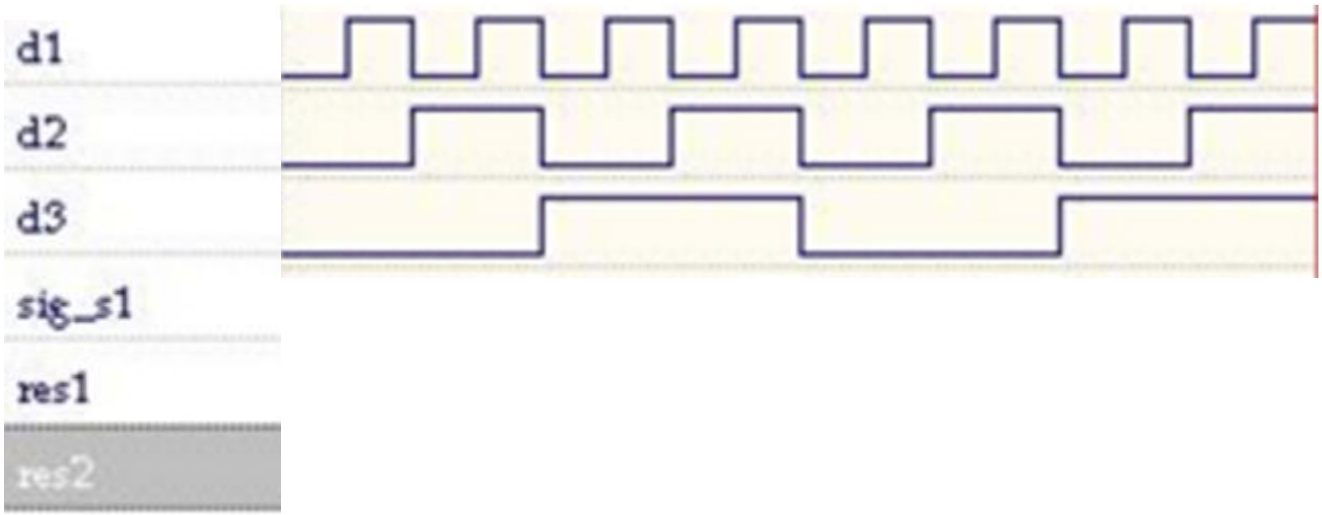
```
SIGNAL a: BIT;
SIGNAL b: BIT_VECTOR(7 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL e: INTEGER RANGE 0 TO 255;


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

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

Solution

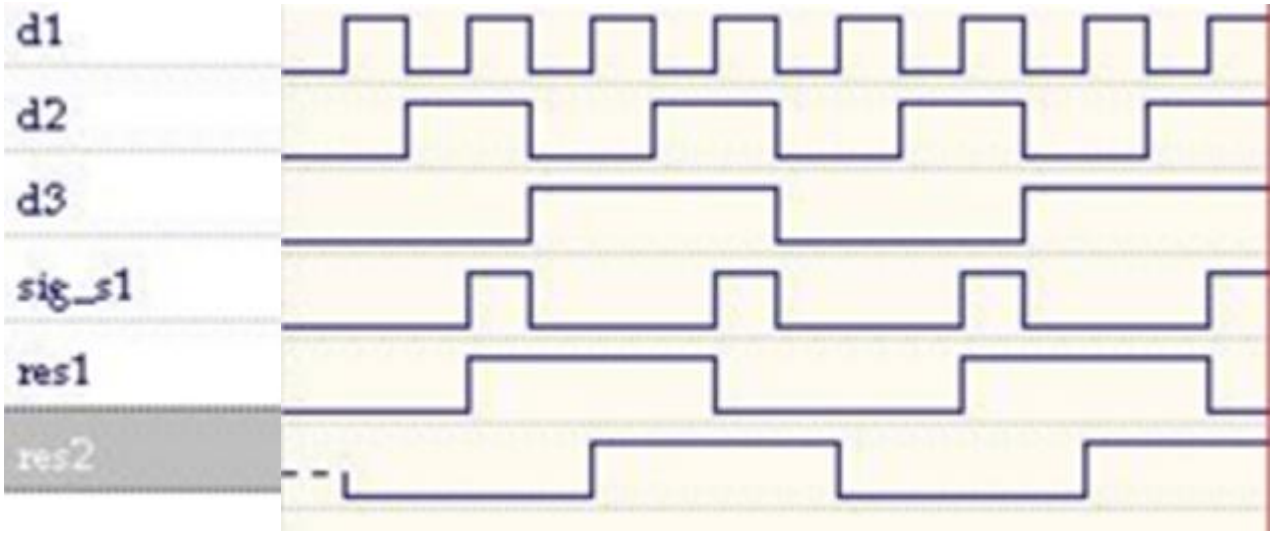
```
SIGNAL a: BIT;
SIGNAL b: BIT_VECTOR(7 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL e: INTEGER RANGE 0 TO 255;
...
a <= b(5);     -- legal (same scalar type: BIT)
b(0) <= a;     -- legal (same scalar type: BIT)
c <= d(5);     -- legal (same scalar type: STD_LOGIC)
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) Pour le code suivant, dessinez la forme d'onde pour sig_s1, res1 et 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;
```

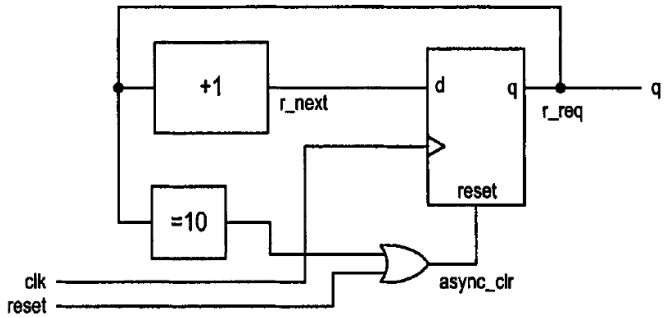


Solution



C)

a. Pour concevoir le compteur mod-10, vous avez besoin d'un registre à n bits. Quelle est la valeur de n?
b. Pour la mise en œuvre du compteur ci-dessous, le temps d'établissement du registre est 0.5 ns et le temps de propagation de l'horloge à la sortie est 1 ns. Les retards de propagation des autres composants sont: l'incrémenteur 5 ns, le comparateur 3 ns et le multiplexeur (et les autres composants combinatoires le cas échéant) 0.75 ns. Déterminez le taux d'horloge maximum.



Solution

N=4

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

f_clk=1/6.5ns=153.8MHz

Question 2 (15 points)

Écrire du code VHDL comportemental qui représente un compteur/décompteur de 24 bits avec une charge parallèle et une réinitialisation asynchrone.

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 points)

Convertir la machine à états finis de la Figure 1 à une machines à états algorithmiques (*Algorithm State Machines, ASM*).

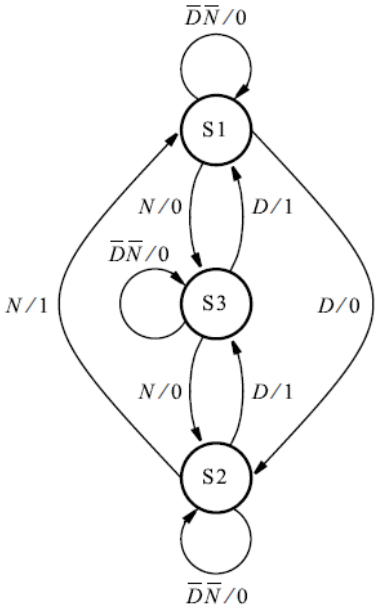
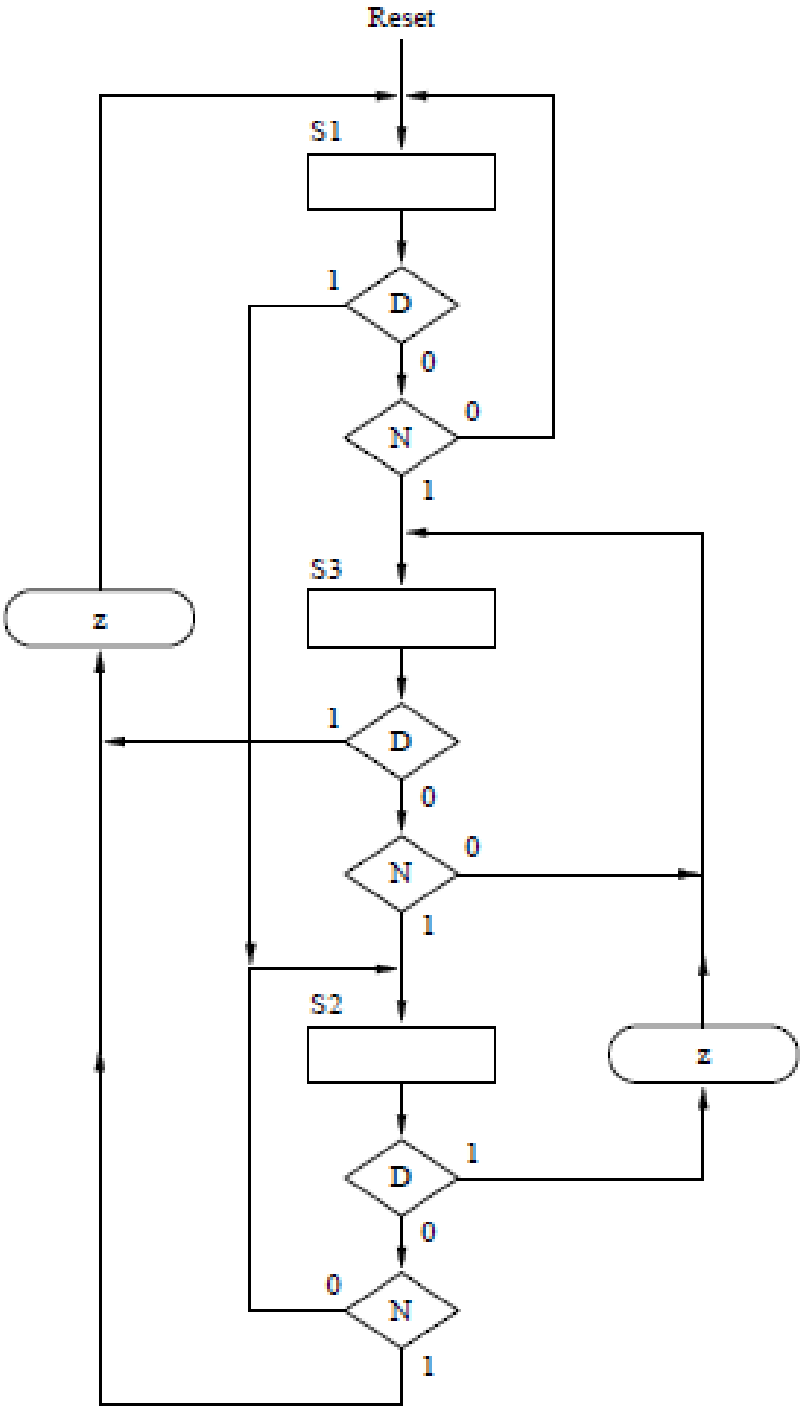


Figure 1 machine à états finis de type Mealy de la Question 2.

Solution

Une machines à états algorithmiques (*Algorithm State Machines, ASM*) est:

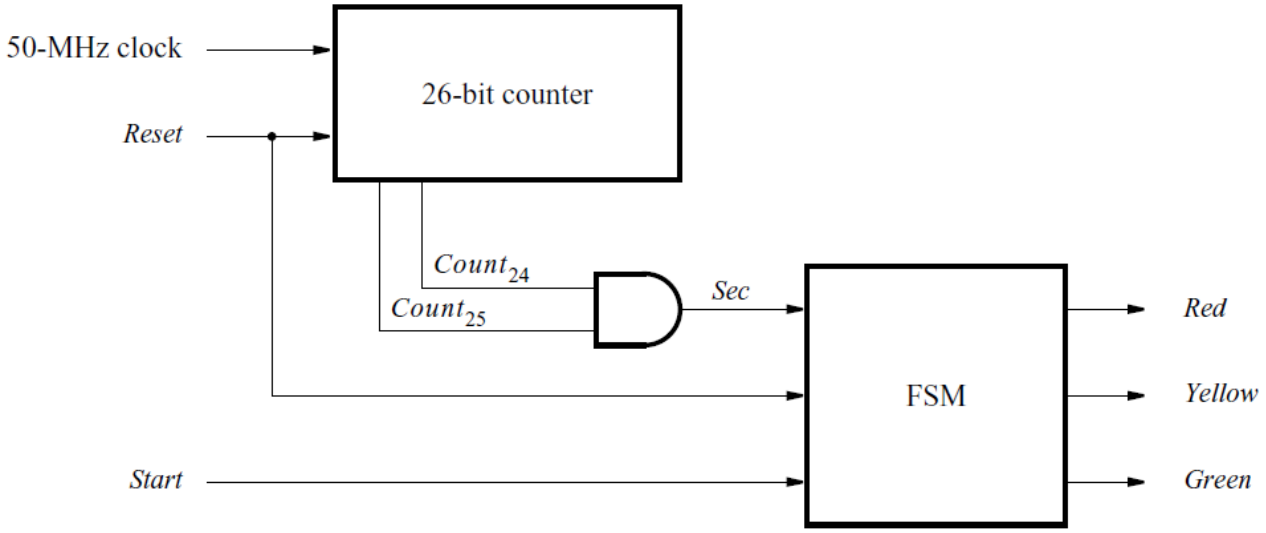


Question 4 (30 points)

a) Concevoir un circuit de contrôle des feux utilisés pour démarrer une course, qui fonctionne comme suit. Il y a trois entrées: Reset, Start et Clock. Il y a trois sorties: Rouge, Jaune et Vert, qui allument les lumières. Une seule lumière peut être allumée à tout moment. Le signal Reset force le circuit dans un état dans lequel la lumière rouge est allumée. Lorsque le signal Start est activé, la lumière rouge reste allumé pendant au moins une seconde de plus, puis la lumière jaune s'allume. La lumière jaune reste allumé environ une seconde, puis la lumière vert s'allume. La lumière vert reste allumé pendant au moins trois secondes, puis la lumière rouge s'allume et le circuit revient à son état initial. Supposons que vous pouvez utiliser une horloge de 1 MHz.

Solution

Un circuit approprié est



Les deux bits les plus significatifs du compteur de 26 bits deviennent égaux à 1 après le temps écoulé d'un peu plus d'une seconde. Ils sont utilisés pour générer un signal "horloge", appelé Sec, pour la machine à états finis. A chaque front positif du signal Sec, la machine à états finis (finite state machine, FSM) change d'état comme suit

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) Écrire un code VHDL qui peut être utilisé pour synthétiser le circuit spécifié à la question 4a.

Solution

Le circuit de contrôle peut être spécifié par le code VHDL suivant:

```
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 (20 points)

Implémenter en VHDL l'équation (a * b * c * data_in) en utilisant les techniques de pipelining que nous avons étudiées dans la classe. Notez que a, b et c sont des constantes ici et la variable 'data_in' change à chaque cycle d'horloge. Le résultat du calcul sera disponible dans le port 'data_out' également à chaque cycle d'horloge. Une opération de multiplication prend moins de temps que la période d'horloge. Pour le VHDL du multiplicateur à 2 entrées, utilisez: 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 points)

Dérivez la table de flux minimal qui spécifie le même comportement fonctionnel que la table de flux de la 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 **Table de flux pour la question 5**

Solution

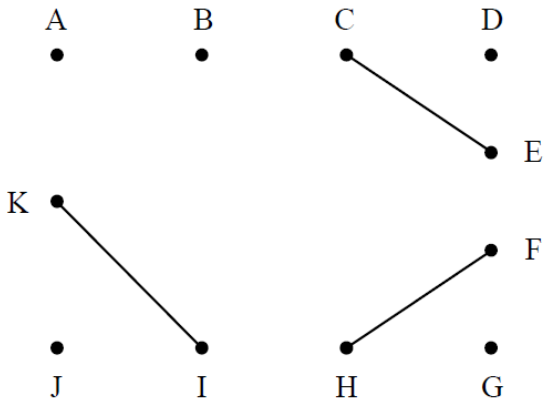
La procédure de partitionnement donne:

P_1 = (ADGJMPT)(BEHR)(CF)(ILOSV)(KNU)
P_2 = (AD)(GP)(JMT)(B)(E)(HR)(C)(F)(ILOSV)(KNU)
P_3 = (A)(D)(GP)(JMT)(B)(E)(HR)(C)(F)(ILOSV)(KNU)
$P_4 = P_3$

Cela donne la table de flux:

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

Le diagramme de fusion correspondant est :



Cela donne la table de flux réduit:

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

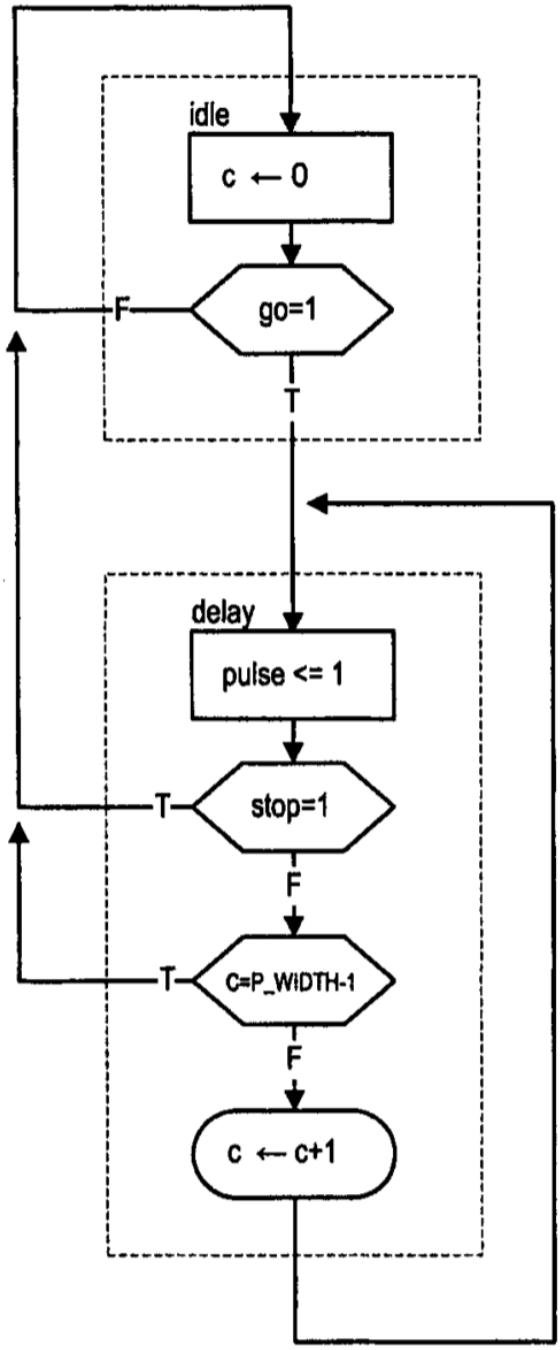
Question 6A (10 points)

Un générateur d'impulsions *one-shot* est un circuit qui génère une seule impulsion de largeur fixe lors de l'activation d'un signal de déclenchement. Nous supposons que la largeur de l'impulsion est de N cycles d'horloge. Les spécifications détaillées sont énumérées ci-dessous.

- Il y a trois signaux d'entrée, N, go et stop, et un signal de sortie, pulse.
- Le signal go est le signal de déclenchement qui est généralement affirmé pour un seul cycle d'horloge. En fonctionnement normal, l'assertion du signal go active le signal d'impulsion pendant cinq cycles d'horloge.
- Si le signal go est réaffirmé pendant cet intervalle, il sera ignoré.
- Si le signal stop est activé pendant cet intervalle, le signal d'impulsion sera coupé et retournera à '0'.

Présenter le diagramme ASMD pour le générateur d'impulsion one-shot.

Solution



Question 6B (10 marks)

Trouver une implémentation de la fonction *f* à un coût minimum sans «hazard».

f (x1, . . . , x4) =∑m(0, 4, 11, 13, 15) + D(2, 3, 5, 10)

Solution

L'implémentation de la fonction *f* à un coût minimum sans «hazard»:

$$f = \overline{x}_1\overline{x}_3\overline{x}_4 + x_1x_2x_4 + x_1x_3x_4$$

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 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 );
```

VHDL if Statement:

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