

17th December 2017



PRIFYSGOL
BANGOR
UNIVERSITY

VLSI Design
Lab Report 4

Martin Le Mintier

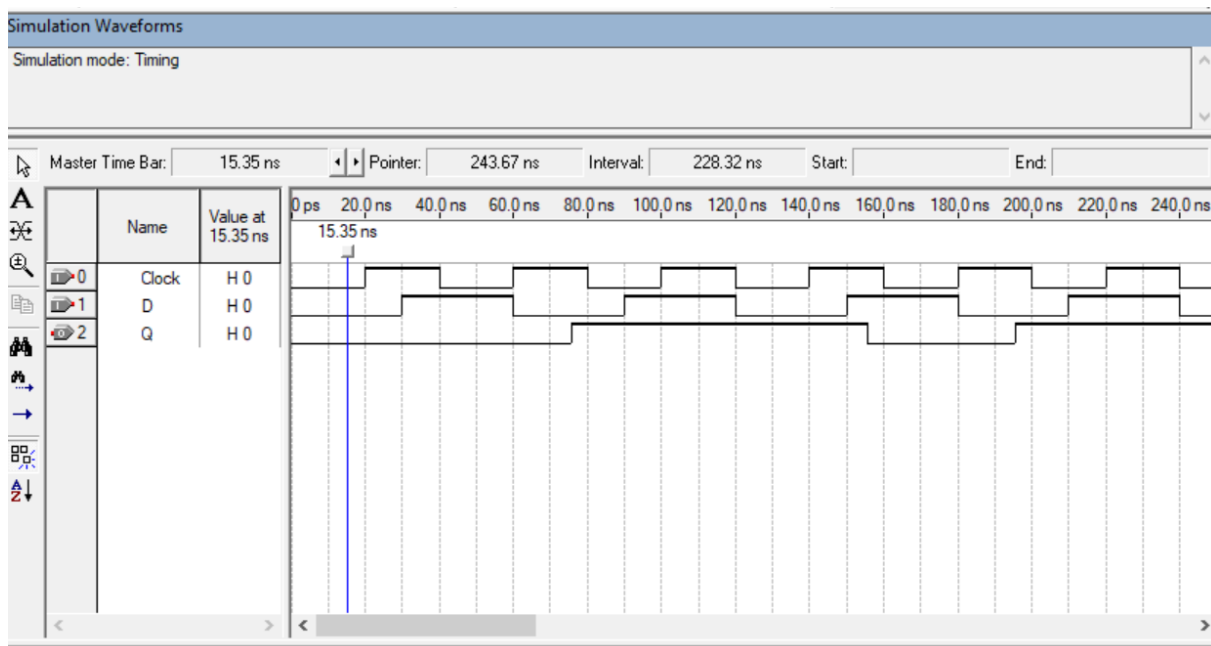
500494525

Flip Flop and Register

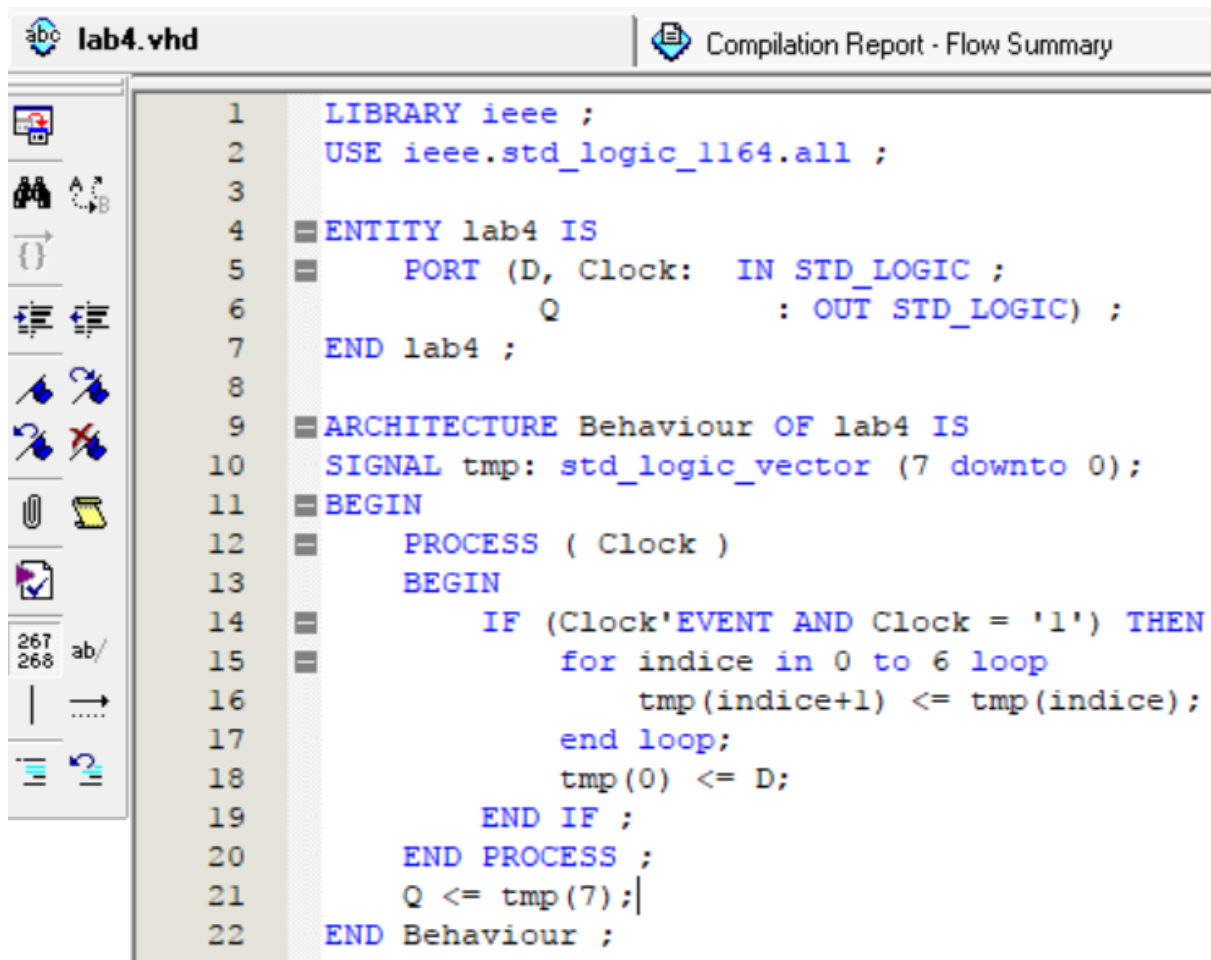
Here is the VHDL file with the cod from the subject.

```
1  LIBRARY ieee ;
2  USE ieee.std_logic_1164.all ;
3
4  ENTITY lab4 IS
5  PORT (D, Clock:  IN STD_LOGIC ;
6        Q          : OUT STD_LOGIC) ;
7  END lab4 ;
8
9  ARCHITECTURE Behaviour OF lab4 IS
10 BEGIN
11     PROCESS ( Clock )
12     BEGIN
13         IF (Clock'EVENT AND Clock = '1') THEN
14             Q <= D ;
15         END IF ;
16     END PROCESS ;
17 END Behaviour ;
18
```

And here is what we can see on the Simulation :

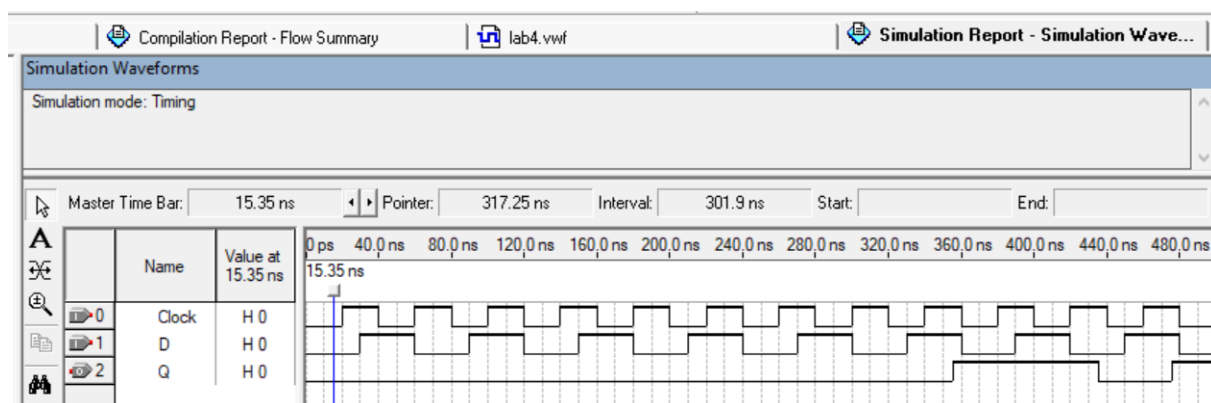


Now if we try to convert our code to an 8bit register



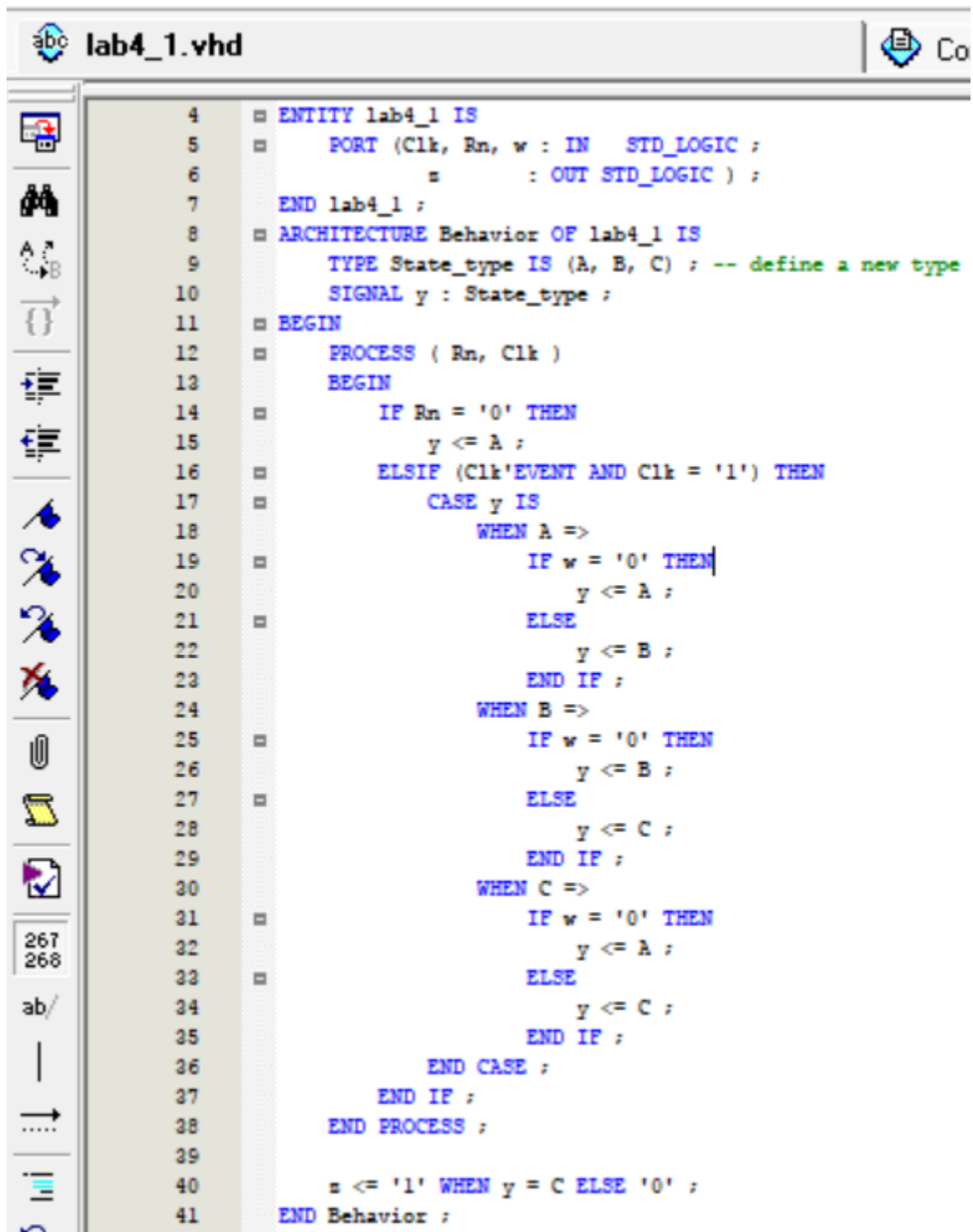
```
1  LIBRARY ieee ;
2  USE ieee.std_logic_1164.all ;
3
4  ENTITY lab4 IS
5  PORT (D, Clock: IN STD_LOGIC ;
6        Q          : OUT STD_LOGIC) ;
7  END lab4 ;
8
9  ARCHITECTURE Behaviour OF lab4 IS
10 SIGNAL tmp: std_logic_vector (7 downto 0);
11 BEGIN
12     PROCESS ( Clock )
13     BEGIN
14         IF (Clock'EVENT AND Clock = '1') THEN
15             for indice in 0 to 6 loop
16                 tmp(indice+1) <= tmp(indice);
17             end loop;
18             tmp(0) <= D;
19         END IF ;
20     END PROCESS ;
21     Q <= tmp(7);
22 END Behaviour ;
```

And here is what we can see on the Simulation :



One Process FSM

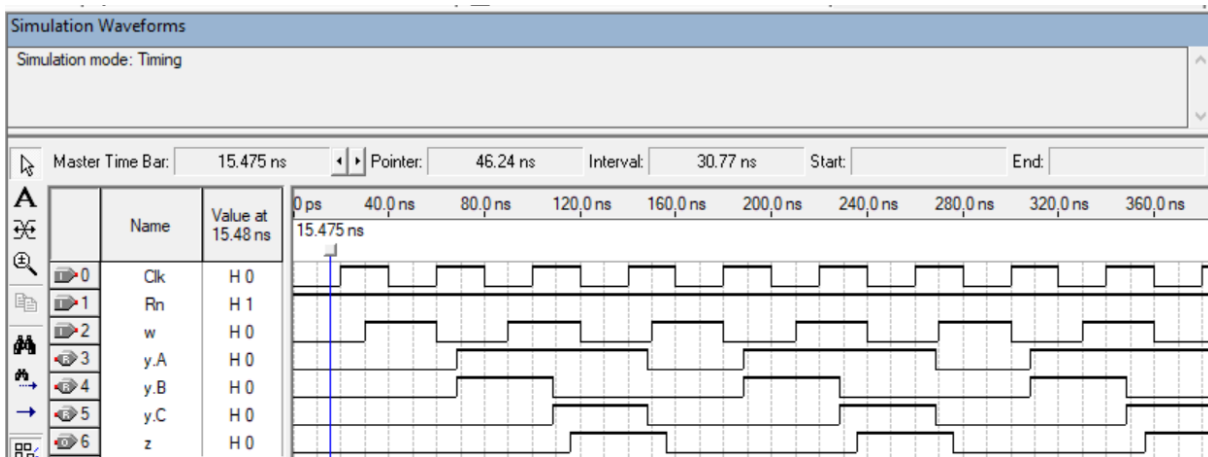
Here is the VHDL file with the cod from the subject.



The screenshot shows a VHDL editor window titled 'lab4_1.vhd'. The code defines an entity 'lab4_1' with three ports: 'Clk' (input), 'Rn' (input), and 'w' (output), all of type 'STD_LOGIC'. The architecture 'Behavior' of 'lab4_1' is defined, featuring a 'State_type' with values 'A', 'B', and 'C'. A signal 'y' is declared of type 'State_type'. The process 'PROCESS (Rn, Clk)' contains the logic for the FSM. It starts with an 'IF Rn = '0'' condition, where 'y' is assigned 'A'. If 'Rn' is '1', it checks for a clock edge ('Clk'EVENT AND Clk = '1'). Inside this, a 'CASE y IS' statement handles the state transitions. For state 'A', if 'w' is '0', 'y' remains 'A'; if 'w' is '1', 'y' transitions to 'B'. For state 'B', if 'w' is '0', 'y' remains 'B'; if 'w' is '1', 'y' transitions to 'C'. For state 'C', if 'w' is '0', 'y' remains 'C'; if 'w' is '1', 'y' transitions back to 'A'. The process ends with 'END CASE', 'END IF', and 'END PROCESS'. A final assignment 'w <= '1' WHEN y = C ELSE '0'' is shown at the bottom.

```
4  ENTITY lab4_1 IS
5  PORT (Clk, Rn, w : IN  STD_LOGIC ;
6         w         : OUT STD_LOGIC ) ;
7  END lab4_1 ;
8  ARCHITECTURE Behavior OF lab4_1 IS
9      TYPE State_type IS (A, B, C) ; -- define a new type
10     SIGNAL y : State_type ;
11 BEGIN
12     PROCESS ( Rn, Clk )
13     BEGIN
14         IF Rn = '0' THEN
15             y <= A ;
16         ELSIF (Clk'EVENT AND Clk = '1') THEN
17             CASE y IS
18                 WHEN A =>
19                     IF w = '0' THEN
20                         y <= A ;
21                     ELSE
22                         y <= B ;
23                     END IF ;
24                 WHEN B =>
25                     IF w = '0' THEN
26                         y <= B ;
27                     ELSE
28                         y <= C ;
29                     END IF ;
30                 WHEN C =>
31                     IF w = '0' THEN
32                         y <= A ;
33                     ELSE
34                         y <= C ;
35                     END IF ;
36             END CASE ;
37         END IF ;
38     END PROCESS ;
39
40     w <= '1' WHEN y = C ELSE '0' ;
41 END Behavior ;
```

And here is what we can see on the Simulation :

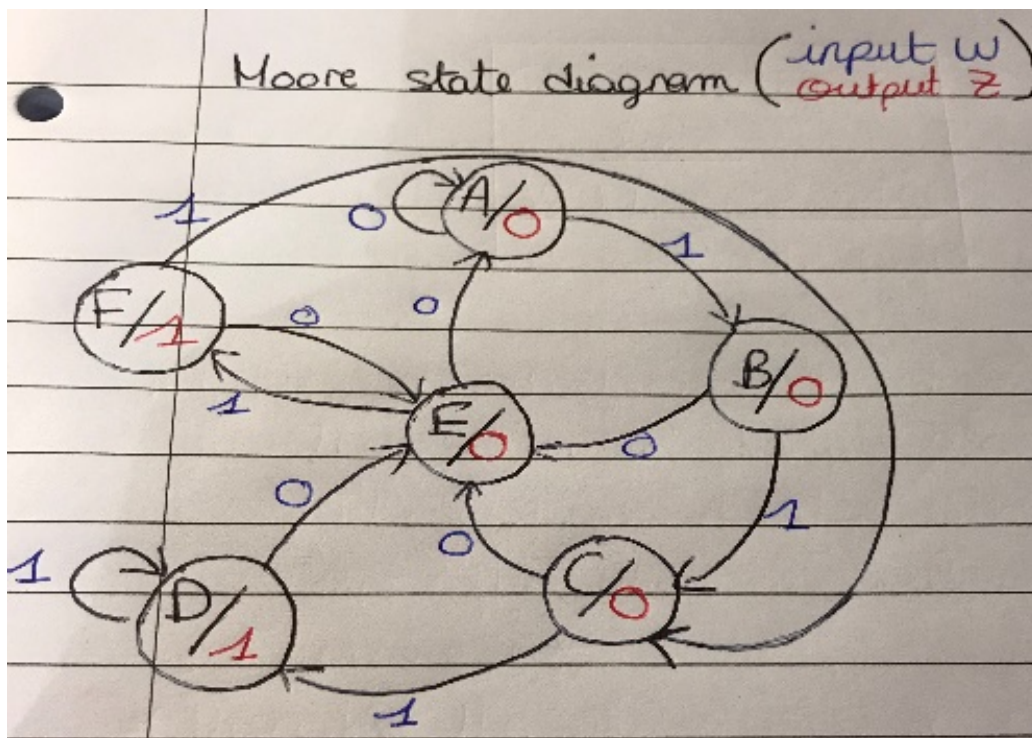


Sequence detector

Design a VHDL entity, implemented in a third new project, which detects the input sequences "101" or "111" on its input.

We tried Moore and Mealy Machine and then chose Moore because it was simpler to build.

Here is our state diagram:



And here is the flow table :

Associated Flow table :

Present State	Next State		Output Z	Partition minimisation:
	w=0	w=1		
A	<u>A</u>	B	0	• $P_1 = (ABCDEF)$
B	E	C	0	• $P_2 = (ABCE)(DF)$ ↳ $(ABCE) \xrightarrow{(AEEA)} (BCE) \rightarrow (BCDE)$ CE not equiv. to AB
C	E	D	0	↳ $(DF) \xrightarrow{(EE)} (DE)$ F not equiv. to D
D	E	<u>D</u>	1	• $P_3 = (AB)(CE)(D)(F)$
E	A	F	0	↳ $(AB) \xrightarrow{(AE)} (BE) \rightarrow (BF)$ A not equiv. to B
F	E	C	1	↳ $(CE) \xrightarrow{(EA)} (DE)$ C not equiv. to E

(not any reduction) \nLeftarrow • $P_4 = (A)(B)(C)(D)(E)(F)$

Here is the VHDL code that we wrote as we did in class :

```
lab4_2.vhd | Compilation Report - Flow Summary
1  LIBRARY ieee ;
2  USE ieee.std_logic_1164.all ;
3
4  ENTITY lab4_2 IS
5  PORT (Clk, Rn, w : IN  STD_LOGIC ;
6        z      : OUT STD_LOGIC ) ;
7  END lab4_2 ;
8
9  ARCHITECTURE Behavior OF lab4_2 IS
10     TYPE State_type IS (A, B, C, D, E, F) ; -- define a new type
11     SIGNAL State : State_type ;
12 BEGIN
13     PROCESS ( Rn, Clk )
14     BEGIN
15         IF Rn = '0' THEN
16             State <= A ;
17         ELSIF (Clk'EVENT AND Clk = '1') THEN
18             CASE State IS
19                 WHEN A =>
20                     IF w = '0' THEN
21                         State <= A ;
22                     ELSE
23                         State <= B ;
24                     END IF ;
25                 WHEN B =>
26                     IF w = '0' THEN
27                         State <= E ;
28                     ELSE
29                         State <= C ;
30                     END IF ;
```

```
lab4_2.vhd | Compilation Report - Flow Summary
31         WHEN C =>
32             IF w = '0' THEN
33                 State <= E ;
34             ELSE
35                 State <= D ;
36             END IF ;
37         WHEN D =>
38             IF w = '0' THEN
39                 State <= E ;
40             ELSE
41                 State <= D ;
42             END IF ;
43         WHEN E =>
44             IF w = '0' THEN
45                 State <= A ;
46             ELSE
47                 State <= F ;
48             END IF ;
49         WHEN F =>
50             IF w = '0' THEN
51                 State <= E ;
52             ELSE
53                 State <= C ;
54             END IF ;
55         END CASE ;
56     END IF ;
57 END PROCESS ;
58
59 z <= '1' WHEN (State = D OR State = F) ELSE '0' ;
60 END Behavior ;
```

And here is what we can see on the Simulation :

