

การทดลองที่ 3: การใช้งานคำสั่ง Process

วัตถุประสงค์

1. เพื่อให้นักศึกษาฝึกเขียนภาษา VHDL เบื้องต้น
2. เพื่อศึกษาการออกแบบวงจรด้วยคำสั่ง Process
3. เพื่อศึกษาการออกแบบวงจร FlipFlop

หมายเหตุ

1. นักศึกษาที่มีหนังสือ ให้พิมพ์เฉพาะหน้าแรกของเอกสารการทดลองนี้ก็พอ
2. ทำการทดลองตามหนังสือหัวข้อการทดลองที่ 4.1 – 4.3.2 หน้า ที่ 242 ถึง 272 ก่อน

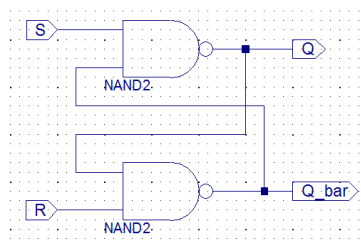
การทดลอง

1. ทำการทดลองตามรายละเอียดด้านล่างนี้ โดยตัวอย่างจะเป็นโปรแกรมภาษา VHDL ของการสร้าง Flip flop แบบต่างๆ และวงจรนับ
2. ทำการรวบรวม แก้ไข ปรับปรุง เปลี่ยนแปลง โปรแกรมตัวอย่างให้เป็นวงจรนับขึ้น 0 – 9 โดยแสดงผลที่ตัวแสดงผลเจ็ดส่วนจำนวน 1 หลัก โดยกำหนดให้ตัวเลขนับขึ้นทุกๆ 1,000 mS
3. ทำการโปรแกรมที่แก้ไขลงบอร์ดทดลอง แล้วเรียกอาจารย์ผู้คุมการทดลองมาตรวจ

รายละเอียดการทดลอง

4.1 แลตช์

แลตช์พื้นฐานได้แก่ แลตช์แบบ NAND Gate และ NOR Gate แสดงดังรูปที่ 4.1 และรูปที่ 4.2 ตามลำดับ โดยที่ Invalid จะเป็นสถานะที่ Q และ Q_bar (หรือ Q) มีลอจิกเหมือนกันซึ่งไม่เป็นความจริง ส่วนสถานะ Hold นั้นวงจรจะไม่มี การเปลี่ยนสถานะ (Latch) แลตช์เป็นหน่วยความจำพื้นฐานและไม่จัดว่าเป็นวงจรซีเควนเชียล เนื่องจากเอาต์พุตของแลตช์นั้นไม่ขึ้นกับค่าอินพุต และ/หรือค่าที่จำเอาไว้ (State)



4.1a) ฟังก์ชัน NAND Gate Latch

Input		Output		Remark
S	R	Q	Q_bar	
1	1	Q (0/1)	Q_bar (1/0)	Hold (No change)
0	1	1	0	Set
1	0	0	1	Clear
0	0	1	1	Invalid

4.1b) ตารางความจริงของวงจร NAND Gate Latch

```

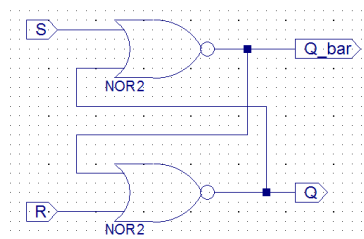
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity NAND_GATE_LATCH is
6     Port (R,S : in STD_LOGIC;
7           Q,Q_bar : buffer STD_LOGIC);
8 end NAND_GATE_LATCH;
9
10 architecture Behavioral of NAND_GATE_LATCH is
11 begin
12     Q <= S nand Q_bar;
13     Q_bar <= R nand Q;
14 end Behavioral;

```

Q และ Q_bar มี Mode เป็น buffer เนื่องจากในรูปที่ 4.1a) นั้นเอาต์พุตของ Q และ Q_bar ถูกต่อกลับเข้ามาที่อินพุตของแนนด์เกตอีกตัวที่เหลือ

รูปที่ 4.1c) ไลค์ของวงจร NAND Gate Latch

รูปที่ 4.1 NAND Gate Latch



4.2a) ฟังก์ชันวงจร NOR Gate Latch

Input		Output		Remark
S	R	Q	Q_bar	
0	0	Q (0/1)	Q_bar (1/0)	Hold (No change)
1	0	1	0	Set
0	1	0	1	Clear
1	1	0	0	Invalid

4.2b) ตารางความจริงของวงจร NOR Gate Latch

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity NOR_GATE_LATCH is
6     Port (R,S : in STD_LOGIC;
7           Q,Q_bar : buffer STD_LOGIC);
8 end NOR_GATE_LATCH;
9
10 architecture Behavioral of NOR_GATE_LATCH is
11 begin
12     Q <= R nor Q_bar;
13     Q_bar <= S nor Q;
14 end Behavioral;

```

Q และ Q_bar มี Mode เป็น buffer เนื่องจากในรูปที่ 4.1a) นั้นเอาต์พุตของ Q และ Q_bar ถูกต่อกลับเข้ามาที่อินพุตของนอร์เกตอีกตัวที่เหลือ

4.2c) ไลค์ของวงจร NOR Gate Latch

รูปที่ 4.2 NOR Gate Latch

แลตช์อีกชนิดที่ผู้อ่านควรทราบ คือ D Latch ซึ่ง D Latch จะมี D เป็นอินพุตและ Q เป็นเอาต์พุต โดยมีตารางความจริงแสดงดังรูปที่ 4.3a) และมีโค้ด VHDL แสดงดังรูปที่ 4.3b) โดยที่การแลตช์ค่าของอินพุตจะทำได้ก็ต่อเมื่อขา Gate G = '1'

Input		Output	Remark
G	D	Q	
0	0	Q (0/1)	Hold (No change)
0	1	Q (0/1)	Hold (No change)
1	0	0	
1	1	1	

4.3a) ตารางความจริงของ D Latch

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity D_LATCH is
6      Port ( G,D : in  STD_LOGIC;
7            Q : out STD_LOGIC);
8  end D_LATCH;
9
10 architecture Behavioral of D_LATCH is
11 begin
12     process (G,D)
13     begin
14         if G='1' then Q <= D;
15         end if;
16     end process;
17 end Behavioral;

```

4.3b) โค้ดของวงจร D Latch

รูปที่ 4.3 ตารางความจริงและโค้ดของวงจร D Latch

การทดลองที่ 4.1.1 แลตช์

วัตถุประสงค์

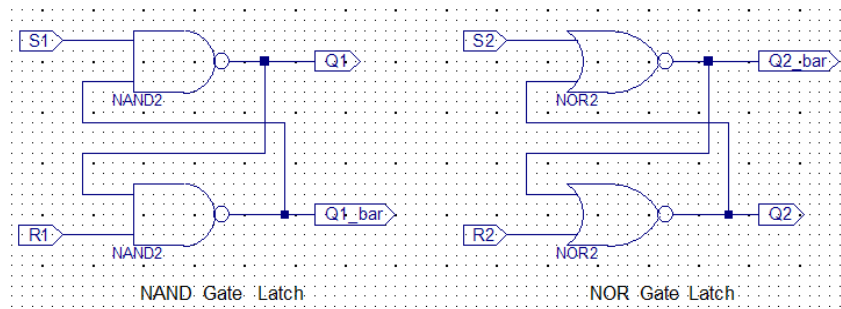
- 1) เพื่อทำความเข้าใจเกี่ยวกับหลักการทำงานของวงจรแลตช์ต่างๆ
- 2) เพื่อสร้างวงจรแลตช์โดยวิธีการเขียนด้วยโค้ด VHDL แล้วโปรแกรมลงชิพ CPLD และ FPGA

อุปกรณ์ทดลอง

บอร์ด CPLD Explorer XC9572XL หรือ CPLD Explorer XC9572 และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1 สร้างวงจร NAND Gate latch และ NOR Gate latch ด้วย CPLD

สร้าง Folder ชื่อ ch4v ไว้ในไดรฟ์ C แล้วเขียนโค้ด VHDL ของวงจรผังวงจรดังรูปที่ L1.1 ได้ดังรูปที่ L1.2 จากนั้นสร้างไฟล์โดยใช้ Project Location (หรือ Folder) ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ex4_1_1vcx1



รูปที่ L1.1 ฟังก์ชัน NAND Gate latch และ NOR Gate latch

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity ex4_1_1vcx1 is
6      Port ( S1,R1,S2,R2 : in  STD_LOGIC;
7            Q1,Q1_bar,Q2,Q2_bar : buffer STD_LOGIC);
8  end ex4_1_1vcx1;
9
10 architecture Behavioral of ex4_1_1vcx1 is
11 begin
12     -----NAND GATE LATCH-----
13     Q1 <= S1 nand Q1_bar;
14     Q1_bar <= R1 nand Q1;
15     -----NOR GATE LATCH-----
16     Q2_bar <= S2 nor Q2;
17     Q2 <= R2 nor Q2_bar;
18 end Behavioral;

```

รูปที่ L1.2 โค้ด VHDL ของวงจร NAND Gate latch และ NOR Gate latch

การกำหนดขาสัญญาณต่างๆ ให้กับวงจรโดยจะใช้ปุ่มกด PB1-PB4 เป็นอินพุตและ LED1-LED4 เป็นเอาต์พุต กล่าวคือ

S1 = PB1 = INPUT = p39 Q1 = LED1 = OUTPUT = p38

R1 = PB2 = INPUT = p40 Q1_bar = LED2 = OUTPUT = p37

S2 = PB3 = INPUT = p42 Q2 = LED3 = OUTPUT = p36

R2 = PB4 = INPUT = p43 Q2_bar = LED4 = OUTPUT = p35

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้

Xilinx PACE - [Design Object List - I/O Pins]						
I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
S1	Input	p39	2	9		
R1	Input	p40	2	11		
S2	Input	p42	2	14		
R2	Input	p43	2	15		
Q1	Output	p38	2	8	SLOW	
Q1_bar	Output	p37	2	6	SLOW	
Q2	Output	p36	2	5	SLOW	
Q2_bar	Output	p35	2	2	SLOW	

รูปที่ L1.3 Assign Package Pins

หลังจากโปรแกรมวงจรลงชิพ CPLD แล้วทดลองกดปุ่ม PB1-PB4 และให้สังเกตผลลัพธ์ที่ LED1-LED4 ว่าให้ลอจิกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นให้สังเกตผลสถานะที่เป็น “Invalid” ว่าเป็นอย่างไร แล้วทำการบันทึกผลการทดลอง

แม้ว่า Xilinx Synthesis Tool หรือ XST จะรองรับโหมด Buffer แต่ซอฟต์แวร์ XST แนะนำให้หลีกเลี่ยงการใช้โหมด Buffer ดังนั้นเราจึงเขียนโค้ดใหม่ได้ดังรูปที่ L1.4 จากนั้นให้ทำการทดลองซ้ำ

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity ex4_1_1vcx1 is
6      Port ( S1,R1,S2,R2 : in  STD_LOGIC;
7            Q1,Q1_bar,Q2,Q2_bar : out STD_LOGIC);
8  end ex4_1_1vcx1;
9
10 architecture Behavioral of ex4_1_1vcx1 is
11     signal Q1T,Q1T_bar,Q2T,Q2T_bar : STD_LOGIC;
12 begin
13     -----NAND GATE LATCH-----
14     Q1T <= S1 nand Q1T_bar;
15     Q1T_bar <= R1 nand Q1T;
16     Q1 <= Q1T;
17     Q1_bar <= Q1T_bar;
18     -----NOR GATE LATCH-----
19     Q2T_bar <= S2 nor Q2T;
20     Q2T <= R2 nor Q2T_bar;
21     Q2 <= Q2T;
22     Q2_bar <= Q2T_bar;
23 end Behavioral;

```

รูปที่ L1.4 โค้ด VHDL ที่เขียนโดยประกาศใช้ Signal

2 สร้างวงจร NAND Gate latch และ NOR Gate latch ด้วย FPGA

สร้าง Folder ชื่อ ch4v ไว้ในไดรฟ์ C (ในกรณีที่ยังไม่สร้าง Folder ชื่อ ch4v) จากนั้นออกแบบ NAND Gate latch และ NOR Gate latch ซึ่งมีผังวงจรแสดงดังรูปที่ L1.1 และโค้ด VHDL แสดงดังรูปที่ L2.1 โดยใช้ Project Location ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ex4_1_1vf

การกำหนดขาสัญญาณต่างๆ ให้กับวงจรโดยจะใช้ปุ่มกด PB1-PB4 เป็นอินพุตและ LED L0-L3 เป็นเอาต์พุต กล่าวคือ

S1 = PB1 = INPUT = p44	Q1 = L3 = OUTPUT = p76
R1 = PB2 = INPUT = p46	Q1_bar = L2 = OUTPUT = p69
S2 = PB3 = INPUT = p47	Q2 = L1 = OUTPUT = p77
R2 = PB4 = INPUT = p50	Q2_bar = L0 = OUTPUT = p70

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity ex4_1_1vf is
6      Port ( S1,R1,S2,R2 : in  STD_LOGIC;
7            Q1,Q1_bar,Q2,Q2_bar : buffer STD_LOGIC);
8  end ex4_1_1vf;
9
10 architecture Behavioral of ex4_1_1vf is
11 begin
12     -----NAND GATE LATCH-----
13     Q1 <= S1 nand Q1_bar;
14     Q1_bar <= R1 nand Q1;
15     -----NOR GATE LATCH-----
16     Q2_bar <= S2 nor Q2;
17     Q2 <= R2 nor Q2_bar;
18 end Behavioral;

```

รูปที่ L2.1 โค้ดของวงจร NAND Gate latch และ NOR Gate latch

พิมพ์ใน Assign Package Pins สรุปดังรูปที่ L2.2

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
Q1	Output	p76	BANK3	LVCN0533	N/A	3.30			SLOW		Unknown		<input type="checkbox"/>
Q1_bar	Output	p69	BANK4	LVCN0533	N/A	3.30			SLOW		Unknown		<input type="checkbox"/>
Q2	Output	p77	BANK3	LVCN0533	N/A	3.30			SLOW		Unknown		<input type="checkbox"/>
Q2_bar	Output	p70	BANK4	LVCN0533	N/A	3.30			SLOW		Unknown		<input type="checkbox"/>
R1	Input	p46	BANK5	LVCN0533	N/A	3.30					Unknown		<input type="checkbox"/>
R2	Input	p50	BANK5	LVCN0533	N/A	3.30					Unknown		<input type="checkbox"/>
S1	Input	p44	BANK5	LVCN0533	N/A	3.30					Unknown		<input type="checkbox"/>
S2	Input	p47	BANK5	LVCN0533	N/A	3.30					Unknown		<input type="checkbox"/>

รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมวงจรเชิง FPGA แล้วให้ทดลองกดปุ่ม PB1–PB4 และให้สังเกตผลที่ LED L0-L3 ว่าให้ลอจิกเอาต์พุตเป็นไปตามทฤษฎีหรือไม่ จากนั้นให้สังเกตผลสถานะที่เป็น “Invalid” ว่าเป็นอย่างไร แล้วทำการบันทึกผลการทดลอง

แม้ว่า Xilinx Synthesis Tool หรือ XST จะรองรับโหมด Buffer แต่ซอฟต์แวร์ XST แนะนำให้หลีกเลี่ยงการใช้โหมด Buffer ดังนั้นเราจึงเขียนโค้ดใหม่ได้ดังรูปที่ L2.3 จากนั้นให้ทำการทดลองซ้ำ

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity ex4_1_1vf is
6      Port ( S1,R1,S2,R2 : in  STD_LOGIC;
7            Q1,Q1_bar,Q2,Q2_bar : out STD_LOGIC);
8  end ex4_1_1vf;
9
10 architecture Behavioral of ex4_1_1vf is
11     signal Q1T,Q1T_bar,Q2T,Q2T_bar : STD_LOGIC;
12 begin
13     -----NAND GATE LATCH-----
14     Q1T <= S1 nand Q1T_bar;
15     Q1T_bar <= R1 nand Q1T;
16     Q1 <= Q1T;
17     Q1_bar <= Q1T_bar;
18     -----NOR GATE LATCH-----
19     Q2T_bar <= S2 nor Q2T;
20     Q2T <= R2 nor Q2T_bar;
21     Q2 <= Q2T;
22     Q2_bar <= Q2T_bar;
23 end Behavioral;

```



L2.3 โค้ด VHDL ที่เขียนโดยประกาศใช้ Signal

4.2 ฟลิปฟลอป

ฟลิปฟลอป (Flip-Flop) เป็นหน่วยความจำ (Memory element) ที่อาจประกอบด้วยแลตช์หลายตัว โดยที่เอาต์พุตของฟลิปฟลอปจะเปลี่ยนสถานะก็ต่อเมื่อมีการทริกด้วยขอบ (Edge triggered) ของสัญญาณนาฬิกา (Clock) เท่านั้น

1) D Flip-Flop

D Flip-Flop แต่ละชนิดแสดงดังตารางความจริงและโค้ด VHDL ในรูปที่ 4.4 ถึงรูปที่ 4.6 โดย C เป็นสัญญาณนาฬิกา (Clock) ที่ทริกด้วยขอบบวก (ขอบขาขึ้นหรือ Positive edge-triggered) โค้ดที่เขียนดังรูปที่ 4.4b) และรูปที่ 4.4c) จะให้ผลลัพธ์ตรงตามตารางความจริงในรูปที่ 4.4a) แต่ผังวงจรที่ได้จากการสังเคราะห์จะแตกต่างกันดังในรูปที่ 4.4d) และรูปที่ 4.4e) และผลการสังเคราะห์วงจรด้วย CPLD และ FPGA อาจได้ผลต่างกันแม้จะใช้โค้ด VHDL เดียวกันเพราะโครงสร้างภายใน CPLD และ FPGA แตกต่างกัน ถ้าต้องการ D Flip-Flop ที่ทริกด้วยขอบลงก็ให้แก้โค้ดบรรทัดที่ 14 ในรูปที่ 4.4b) เป็น C'event and C = '0'

Input		Output		Remark
D	C	Q	Q_bar	
0		0	1	Reset
1		1	0	Set

4.4a) ตารางความจริงของ D Flip-Flop

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity D_FF is
6      Port ( C,D : in  STD_LOGIC;
7            Q,Q_bar : out STD_LOGIC);
8  end D_FF;
9
10 architecture Behavioral of D_FF is
11 begin
12     process(c)
13     begin
14         if (C'event and C='1') then Q <= D; Q_bar <= not D;
15         end if;
16     end process;
17 end Behavioral;

```

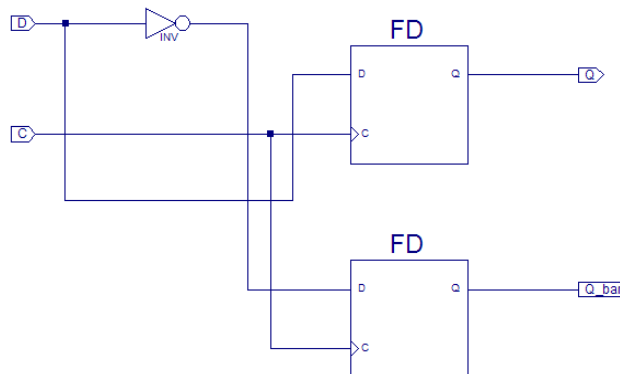
4.4b) โค้ด VHDL ของ D Flip-Flop ที่เขียนด้วยวิธีที่ 1

```

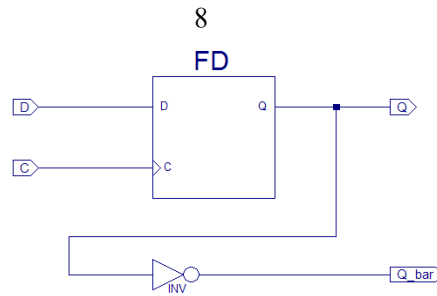
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity D_FF is
6      Port ( C,D : in  STD_LOGIC;
7            Q : buffer STD_LOGIC;
8            Q_bar : out STD_LOGIC);
9  end D_FF;
10
11 architecture Behavioral of D_FF is
12 begin
13     process(c)
14     begin
15         if (C'event and C='1') then Q <= D;
16         end if;
17     end process;
18     Q_bar <= not Q;
19 end Behavioral;

```

4.4c) โค้ด VHDL ของ D Flip-Flop ที่เขียนด้วยวิธีที่ 2



4.4d) ผลการสังเคราะห์จากโค้ดในรูปที่ 4.4b) เมื่อสังเคราะห์โดยใช้ CPLD (View RTL schematic)



4.4e) ผลการสังเคราะห์จากโค้ดในรูปที่ 4.4c) เมื่อสังเคราะห์โดยใช้ CPLD (View RTL schematic)

รูปที่ 4.4 โค้ด VHDL และผังวงจรของ D Flip-Flop แบบทริกด้วยขอบขาขึ้น

หลักการทํางาน D Flip-Flop แบบอะซิงโครนัสเคลียร์ (D Flip-Flop with asynchronous clear) แสดงดังตารางความจริงในรูปที่ 4.5a) โดยที่ “-” คือ Don’t care และโค้ด VHDL ของ D Flip-Flop แสดงดังรูปที่ 4.5b)

Input			Output	Remark
CLR	D	C	Q	
1	-	-	0	Clear
0	0		0	Reset
0	1		1	Set

4.5a) ตารางความจริงของ D Flip-Flop แบบอะซิงโครนัสเคลียร์

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity DFF_CLR is
6     port ( D,C,CLR : in  STD_LOGIC;
7           Q : out  STD_LOGIC);
8 end DFF_CLR;
9
10 architecture BEHAVIORAL of DFF_CLR is
11 begin
12     process (C,CLR)
13     begin
14         if CLR='1' then Q <= '0';
15         elsif C'event and C='1' then Q <= D;
16         end if;
17     end process;
18 end BEHAVIORAL;

```

4.5b) โค้ด VHDL ของ D Flip-Flop แบบอะซิงโครนัสเคลียร์

รูปที่ 4.5 โค้ด VHDL และผังวงจร D Flip-Flop แบบอะซิงโครนัสเคลียร์

หลักการทํางาน D Flip-Flop แบบซิงโครนัสรีเซต (D Flip-Flop with synchronous reset) แสดงดังตารางความจริงในรูปที่ 4.6a) และโค้ด VHDL แสดงดังรูปที่ 4.6b) เมื่อ R (Reset) = ‘1’ แล้วเอาต์พุต Q จะยังไม่รีเซตจนกว่าจะมีการทริกด้วยขอบขาขึ้นของสัญญาณนาฬิกา ซึ่งจะแตกต่างจาก D Flip-Flop แบบอะซิงโครนัสเคลียร์ที่เคลียร์ค่าเอาต์พุตทันทีเมื่อ CLR = ‘1’

Input			Output	Remark
R	D	C	Q	
1	-		0	Reset
0	0		0	Reset
0	1		1	Set

4.6a) ตารางความจริงของ D Flip-Flop แบบซิงโครนัสรีเซต


```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity DFF_RESET is
6      port ( D,C,RESET : in  STD_LOGIC;
7            Q : out  STD_LOGIC);
8  end DFF_RESET;
9
10 architecture BEHAVIORAL of DFF_RESET is
11 begin
12     process(C)
13     begin
14         if C'event and C='1' then
15             if RESET='1' then Q <= '0';
16             else Q <= D;
17             end if;
18         end if;
19     end process;
20 end BEHAVIORAL;





```

4.6b) โค้ด VHDL ของ D Flip-Flop แบบซิงโครนัสรีเซต

รูปที่ 4.6 โค้ด VHDL และผังวงจร D Flip-Flop แบบซิงโครนัสรีเซตที่ทริกด้วยขอบขาขึ้น

2) JK Flip-Flop

JK Flip-Flop แบบอะซิงโครนัสเคลียร์ (JK Flip-Flop with asynchronous clear) มีตารางความจริงแสดงดังรูปที่ 4.7a) ซึ่งมีโค้ดแสดงดังรูปที่ 4.7b) และรูปที่ 4.7c) โดยในการเขียนโค้ดนี้เราจะประกาศใช้ Signal เพื่อหลีกเลี่ยงการใช้โหมด Buffer ที่เกิดเนื่องจากเอาต์พุตอ่านค่ากลับ คือ $Q \leq Q$; และ $Q \leq \text{not } Q$;

Input				Ouput	Remark
CLR	J	K	C	Q	
1	-	-	-	0	Clear
0	0	0		Q (0/1)	Hold (No change)
0	0	1		0	Reset
0	1	0		1	Set
0	1	1		\overline{Q} (1/0)	Toggle

4.7a) ตารางความจริงของ JK Flip-Flop แบบอะซิงโครนัสเคลียร์

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity JKFF is
6      Port ( C,CLR,J,K: in  STD_LOGIC;
7            Q : out  STD_LOGIC);
8  end JKFF;
9
10 architecture Behavioral of JKFF is
11     signal JK : STD_LOGIC_VECTOR(1 downto 0);
12     signal QT : STD_LOGIC;
13 begin
14     JK <= (J&K); -- Concatenation
15     process(C,CLR)
16     begin
17         if CLR='1' then QT <= '0';
18         elsif (C'event and C='1') then
19             if JK="00" then QT <= QT;
20             elsif JK="01" then QT <= '0';
21             elsif JK="10" then QT <= '1';
22             elsif JK="11" then QT <= not QT;
23         end if;
24     end if;
25 end process;
26     Q <= QT;
27 end Behavioral;

```

4.7b) โค้ด VHDL ของ JK Flip-Flop ที่เขียนโดยใช้คำสั่ง if

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity JKFF is
6      Port ( C,CLR,J,K: in  STD_LOGIC;
7            Q : out  STD_LOGIC);
8  end JKFF;
9
10 architecture Behavioral of JKFF is
11     signal JK : STD_LOGIC_VECTOR(1 downto 0);
12     signal QT : STD_LOGIC;
13 begin
14     JK <= (J&K); -- Concatenation
15     process(C,CLR)
16     begin
17         if CLR='1' then QT <= '0';
18         elsif (C'event and C='1') then
19             case JK is
20                 when "00" => QT <= QT;
21                 when "01" => QT <= '0';
22                 when "10" => QT <= '1';
23                 when "11" => QT <= not QT;
24                 when others => null; --No action
25             end case;
26         end if;
27     end process;
28     Q <= QT;
29 end Behavioral;

```

4.7c) โค้ด VHDL ของ JK Flip-Flop ที่เขียนโดยใช้คำสั่ง case และใช้คำสั่ง null

รูปที่ 4.7 โค้ดของ JK Flip-Flop แบบอะซิงโครนัสเคลียร์ที่เขียนโดยการประกาศใช้ Signal

ในทำนองเดียวกัน จากรูปที่ 4.7b) และรูปที่ 4.7c) เราสามารถเขียนโค้ดของ JK Flip-Flop แบบอะซิงโครนัสเคลียร์ (JK Flip-Flop with asynchronous clear) โดยการประกาศใช้ Variable เพื่อหลีกเลี่ยงการใช้โหมด Buffer ที่เกิดเนื่องจากเอาต์พุตอ่านค่ากลับ คือ $Q \leq Q$; และ $Q \leq \text{not } Q$; ได้เช่นกัน โค้ดที่ได้แสดงดังรูปที่ 4.8a) และรูปที่ 4.8b)

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity JKFF is
6      Port ( C,CLR,J,K: in  STD_LOGIC;
7            Q : out  STD_LOGIC);
8  end JKFF;
9
10 architecture Behavioral of JKFF is
11 begin
12     process(C,CLR)
13         variable JK : STD_LOGIC_VECTOR(1 downto 0);
14         variable QT : STD_LOGIC;
15     begin
16         JK := (J&K); -- Concatenation
17         if CLR='1' then QT := '0';
18         elsif (C'event and C='1') then
19             if JK="00" then QT := QT;
20             elsif JK="01" then QT := '0';
21             elsif JK="10" then QT := '1';
22             elsif JK="11" then QT := not QT;
23             end if;
24         end if;
25         Q <= QT;
26     end process;
27 end Behavioral;

```

4.8a) โค้ด VHDL ของ JK Flip-Flop ที่เขียนโดยใช้คำสั่ง if แต่ประกาศใช้ Variable แทน Signal

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity JKFF is
6      Port ( C,CLR,J,K: in  STD_LOGIC;
7            Q : out  STD_LOGIC);
8  end JKFF;
9
10 architecture Behavioral of JKFF is
11 begin
12     process(C,CLR)
13         variable JK : STD_LOGIC_VECTOR(1 downto 0);
14         variable QT : STD_LOGIC;
15     begin
16         JK := (J&K); -- Concatenation
17         if CLR='1' then QT := '0';
18         elsif (C'event and C='1') then
19             case JK is
20                 when "00" => QT := QT;
21                 when "01" => QT := '0';
22                 when "10" => QT := '1';
23                 when "11" => QT := not QT;
24                 when others => null;
25             end case;
26         end if;
27         Q <= QT;
28     end process;
29 end Behavioral;

```



4.8b) โค้ด VHDL ของ JK Flip-Flop ที่เขียนโดยใช้คำสั่ง case แต่ประกาศใช้ Variable แทน Signal

รูปที่ 4.8 โค้ดของ JK Flip-Flop แบบอะซิงโครนัสเคลียร์ที่เขียนโดยการประกาศใช้ Variable แทน Signal

ในการทำงานเดียวเราก็สามารถออกแบบ JK Flip-Flop ที่ทริกด้วยขอบขาลงได้เช่นกัน โดยทำการแก้ไขโค้ดในรูปที่ 4.7 และรูปที่ 4.8 ทั้งหมดโดยแก้ไขโค้ดจาก C'event and C = '1' เป็น C'event and C = '0' แทน

3) T Flip-Flop

T Flip-Flop แบบอะซิงโครนัสเคลียร์ มีตารางความจริงและตัวอย่างโค้ดดังรูปที่ 4.9a) ถึงรูปที่ 4.9c)

Input			Ouput	Remark
CLR	T	C	Q	
1	-	-	0	Clear
0	0		Q (0/1)	Hold (No change)
0	1		\overline{Q} (1/0)	Toggle

4.9a) ตารางความจริงของ T Flip-Flop

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity TFF is
6      Port ( C,CLR,T : in  STD_LOGIC;
7            Q : out  STD_LOGIC);
8  end TFF;
9
10 architecture Behavioral of TFF is
11     signal QT : STD_LOGIC;
12 begin
13     process(C,CLR)
14     begin
15         if CLR='1' then QT <= '0';
16         elsif (C'event and C='1') then
17             if T='0' then QT <= QT;
18             elsif T='1' then QT <= not QT;
19             end if;
20         end if;
21     end process;
22     Q <= QT;
23 end Behavioral;

```

4.9b) โค้ด VHDL ของ T Flip-Flop แบบอะซิงโครนัสเคลียร์ที่เขียนโดยใช้คำสั่ง if

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity TFF is
6      Port ( C,CLR,T : in  STD_LOGIC;
7            Q : out  STD_LOGIC);
8  end TFF;
9
10 architecture Behavioral of TFF is
11     signal QT : STD_LOGIC;
12 begin
13     process(C,CLR)
14     begin
15         if CLR='1' then QT <= '0';
16         elsif (C'event and C='1') then
17             case T is
18                 when '0' => QT <= QT;
19                 when '1' => QT <= not QT;
20                 when others => null;
21             end case;
22         end if;
23     end process;
24     Q <= QT;
25 end Behavioral;

```

4.9c) โค้ด VHDL ของ T Flip-Flop แบบอะซิงโครนัสเคลียร์ที่เขียนโดยใช้คำสั่ง case

รูปที่ 4.9 โค้ด VHDL และผังวงจรของ T Flip-Flop แบบอะซิงโครนัสเคลียร์ที่ทริกด้วยขอบขาขึ้น

การทดลองที่ 4.2.1 D Flip-Flop แบบอะซิงโครนัสเคลียร์

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับหลักการทำงานของวงจร D Flip-Flop แบบอะซิงโครนัสเคลียร์
- 2) เพื่อสร้างวงจร D Flip-Flop แบบอะซิงโครนัสเคลียร์แล้วโปรแกรมลงชิพ CPLD และ FPGA

อุปกรณ์ทดลอง

บอร์ด CPLD Explorer XC9572XL หรือ CPLD Explorer XC9572 และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1 สร้างวงจร D Flip-Flop แบบอะซิงโครนัสเคลียร์ด้วย CPLD

สร้างไฟล์โดยใช้ Project Location ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ex4_2_1vcx1 จากนั้นเขียนโค้ดของวงจร D Flip-Flop แบบอะซิงโครนัสเคลียร์ดังรูปที่ L1.1 โดยมี C เป็นขาสัญญาณนาฬิกา (Clock) ที่ทริกด้วยขอบขาลง และมี CLR เป็นขาเคลียร์ที่ทำงานแบบ Active low (จะเคลียร์เมื่อ CLR = '0')

```

2  -----D-flip-flop with asynchronous clear-----
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5
6  entity ex4_2_1vcx1 is
7      Port ( C,CLR,D : in  STD_LOGIC;
8            Q,Q_bar : out STD_LOGIC);
9  end ex4_2_1vcx1;
10
11 architecture Behavioral of ex4_2_1vcx1 is
12 begin
13 process (C,CLR)
14     begin
15         if CLR='0' then
16             Q    <= '0';
17             Q_bar <= '1';
18         elsif (C'event and C='0') then
19             Q    <= D;
20             Q_bar <= not D;
21         end if;
22     end process;
23 end Behavioral;

```

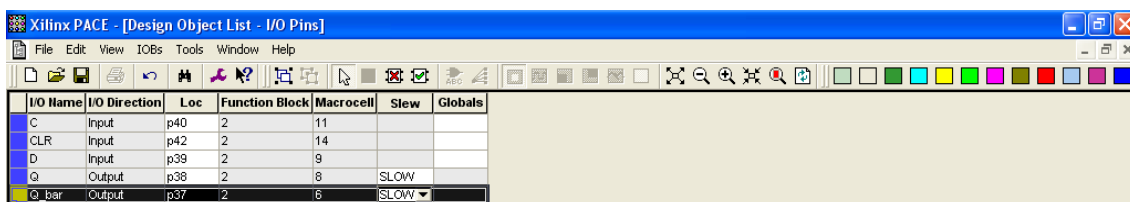
รูปที่ L1.1 D Flip-Flop แบบอะซิงโครนัสเคลียร์

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB3 เป็นอินพุตและ LED1-LED2 เป็นเอาต์พุต กล่าวคือ

D = PB1 = INPUT = p39 CLR = PB3 = INPUT = p42 Q = LED1 = OUTPUT = p38

C = PB2 = INPUT = p40 Q_bar = LED2 = OUTPUT = p37

โดยพิมพ์ใน Assign Package Pins สรุปดังรูปที่ L1.2



I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
C	Input	p40	2	11		
CLR	Input	p42	2	14		
D	Input	p39	2	9		
Q	Output	p38	2	8	SLOW	
Q_bar	Output	p37	2	6	SLOW	

รูปที่ L1.2 Assign Package Pins

หลังจากโปรแกรมวงจรที่ออกแบบลงชิพ CPLD แล้วให้กดปุ่ม PB2 และให้สังเกตผลลัพธ์ที่ LED1-LED2 จากนั้นให้กดปุ่ม PB1 ค้างไว้แล้วกดปุ่ม PB2 และให้สังเกตผลลัพธ์ที่ LED1-LED2 อีกครั้ง และให้ปล่อยปุ่ม PB1 แล้วกดปุ่ม PB2 แล้วให้สังเกตผลลัพธ์ที่ LED1-LED2 ว่า LED1 ติดสว่างหรือไม่ จากนั้นให้กดปุ่ม PB3 แล้วให้สังเกตผลลัพธ์ที่ LED1-LED2 แล้วให้สรุปผลการทดลองทั้งหมดว่าเป็นไปตามทฤษฎีหรือไม่

2 สร้างวงจร D Flip-Flop แบบอะซิงโครนัสเคลียร์ด้วย FPGA

สร้างไฟล์โดยใช้ Project Location ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ex4_2_1vf จากนั้นเขียนโค้ดของวงจร D Flip-Flop แบบอะซิงโครนัสเคลียร์ ดังรูปที่ L2.1 โดยมี C เป็นขาสัญญาณนาฬิกา (Clock) ที่ทริกด้วยขอบขาลง และมี CLR เป็นขาเคลียร์ที่ทำงานแบบ Active low (จะเคลียร์เมื่อ CLR= '0')

```

2  -----D-flip-flop with asynchronous clear-----
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5
6  entity ex4_2_1vf is
7      Port ( C,CLR,D : in  STD_LOGIC;
8            Q,Q_bar : out STD_LOGIC);
9  end ex4_2_1vf;
10
11 architecture Behavioral of ex4_2_1vf is
12 begin
13     process (C,CLR)
14     begin
15         if CLR='0' then
16             Q      <= '0';
17             Q_bar  <= '1';
18         elsif (C'event and C='0') then
19             Q      <= D;
20             Q_bar  <= not D;
21         end if;
22     end process;
23 end Behavioral;

```

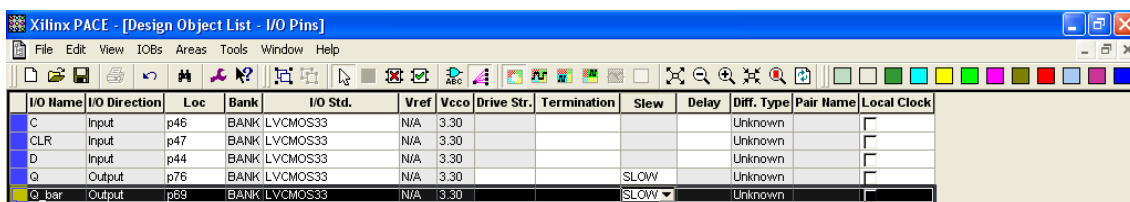
รูปที่ L2.1 โค้ด D Flip-Flop แบบอะซิงโครนัสเคลียร์

การกำหนดขาสัญญาณต่างๆจะใช้ปุ่มกด PB1-PB3 เป็นอินพุตและ LED L2-L3 เป็นเอาต์พุต กล่าวคือ

D = PB1 = INPUT = p44 CLR = PB3 = INPUT = p47 Q = L3 = OUTPUT = p76

C = PB2 = INPUT = p46 Q_bar = L2 = OUTPUT = p69

โดยพิมพ์ใน Assign Package Pins ดังรูปที่ L2.2



I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
C	Input	p46	BANK	LVC MOS33	N/A	3.30					Unknown		<input type="checkbox"/>
CLR	Input	p47	BANK	LVC MOS33	N/A	3.30					Unknown		<input type="checkbox"/>
D	Input	p44	BANK	LVC MOS33	N/A	3.30					Unknown		<input type="checkbox"/>
Q	Output	p76	BANK	LVC MOS33	N/A	3.30			SLOW		Unknown		<input type="checkbox"/>
Q_bar	Output	p69	BANK	LVC MOS33	N/A	3.30			SLOW		Unknown		<input type="checkbox"/>

รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมวงจรที่ออกแบบลงชิพ FPGA แล้วให้กดปุ่ม PB2 และให้สังเกตผลลัพธ์ที่ LED L2-L3 จากนั้นให้กดปุ่ม PB1 ค้างไว้แล้วกดปุ่ม PB2 และให้สังเกตผลลัพธ์ที่ LED L2-L3 อีกครั้ง และให้ปล่อยปุ่ม PB1 แล้วกดปุ่ม PB2 แล้วให้สังเกตผลลัพธ์ที่ LED L2-L3 ว่า LED L3 ติดสว่างหรือไม่ จากนั้นให้กดปุ่ม PB3 แล้วให้สังเกตผลลัพธ์ที่ LED L2-L3 แล้วให้สรุปผลการทดลองทั้งหมดว่าเป็นไปตามทฤษฎีหรือไม่

การทดลองที่ 4.2.2 D Flip-Flop แบบซิงโครนัสรีเซต

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับหลักการทำงานของวงจร D Flip-Flop แบบซิงโครนัสรีเซต
- 2) เพื่อสร้างวงจร D Flip-Flop แบบซิงโครนัสรีเซตแล้วโปรแกรมลงชิพ CPLD และ FPGA

อุปกรณ์ทดลอง

บอร์ด CPLD Explorer XC9572XL หรือ CPLD Explorer XC9572 และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1 สร้างวงจร D Flip-Flop แบบซิงโครนัสรีเซตด้วย CPLD

สร้างไฟล์โดยใช้ Project Location ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ex4_2_2vcx1 จากนั้นเขียนโค้ดของวงจร D Flip-Flop แบบซิงโครนัสรีเซตดังรูปที่ L1.1 โดยมี C เป็นขาสัญญาณนาฬิกา (Clock) ที่ตรึงด้วยขอบขาลงและมี RESET เป็นขาที่ทำงานแบบ Active low (จะรีเซตเมื่อ RESET = '0')

```

2  -----D-flip-flop with synchronous reset-----
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5
6  entity ex4_2_2vcx1 is
7      Port ( C,RESET,D : in  STD_LOGIC;
8            Q,Q_bar : out STD_LOGIC);
9  end ex4_2_2vcx1;
10
11  architecture Behavioral of ex4_2_2vcx1 is
12  begin
13  process(C)
14  begin
15      if (C'event and C='0') then
16          if RESET='0' then Q <= '0'; Q_bar <= '1';
17          else Q <= D; Q_bar <= not D;
18          end if;
19      end if;
20  end process;
21  end Behavioral;

```

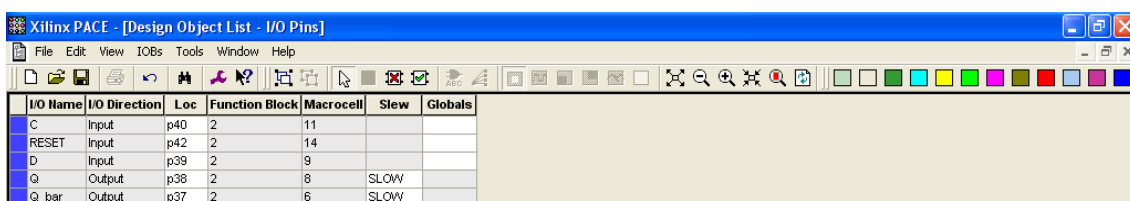
รูปที่ L1.1 D Flip-Flop แบบซิงโครนัสรีเซต

การกำหนดขาสัญญาณต่างๆ จะใช้ปุ่มกด PB1-PB3 เป็นอินพุตและ LED1-LED2 เป็นเอาต์พุต กล่าวคือ

D = PB1 = INPUT = p39 RESET = PB3 = INPUT = p42 Q = LED1 = OUTPUT = p38

C = PB2 = INPUT = p40 Q_bar = LED2 = OUTPUT = p37

โดยพิมพ์ใน Assign Package Pins ดังรูปที่ L1.2



I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
C	Input	p40	2	11		
RESET	Input	p42	2	14		
D	Input	p39	2	9		
Q	Output	p38	2	8	SLOW	
Q_bar	Output	p37	2	6	SLOW	

รูปที่ L1.2 Assign Package Pins

หลังจากโปรแกรมวงจรที่ออกแบบลงชิพ CPLD แล้วให้กดปุ่ม PB2 และให้สังเกตผลลัพธ์ที่ LED1-LED2 จากนั้นให้กดปุ่ม PB1 ค้างไว้แล้วกดปุ่ม PB2 และให้สังเกตผลลัพธ์ที่ LED1-LED2 อีกครั้ง และให้ปล่อยปุ่ม PB1 แล้วกดปุ่ม PB2 แล้วให้สังเกตผลลัพธ์ที่ LED1-LED2 ว่า LED1 ติดสว่างหรือไม่ ให้กดปุ่ม PB3 แล้วให้สังเกตผลลัพธ์ที่ LED1-LED2 จากนั้นให้กดปุ่ม PB3 ค้างไว้แล้วกดปุ่ม PB2 และให้สังเกตผลลัพธ์ที่ LED1-LED2 ให้สรุปผลการทดลองทั้งหมดความเป็นไปตามทฤษฎีหรือไม่

2 สร้างวงจร D Flip-Flop แบบซิงโครนัสรีเซ็ตด้วย FPGA

สร้างไฟล์โดยใช้ Project Location ชื่อ ch4v แล้วกำหนด Project Name และ Source File ชื่อ ex4_2_2vf จากนั้นเขียนโค้ดของวงจร D Flip-Flop แบบซิงโครนัสรีเซ็ตดังรูปที่ L2.1 โดยมี C เป็นขาสัญญาณนาฬิกา (Clock) ที่ทริกด้วยขอบขาลงและมี RESET เป็นขาที่ทำงานแบบ Active low (จะรีเซ็ตเมื่อ RESET= '0')

```

2  -----D-flip-flop with synchronous reset-----
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5
6  entity ex4_2_2vf is
7      Port ( C,RESET,D : in  STD_LOGIC;
8            Q,Q_bar : out  STD_LOGIC);
9  end ex4_2_2vf;
10
11  architecture Behavioral of ex4_2_2vf is
12  begin
13      process(C)
14      begin
15          if (C'event and C='0') then
16              if RESET='0' then Q <= '0'; Q_bar <= '1';
17              else Q <= D; Q_bar <= not D;
18              end if;
19          end if;
20      end process;
21  end Behavioral;

```

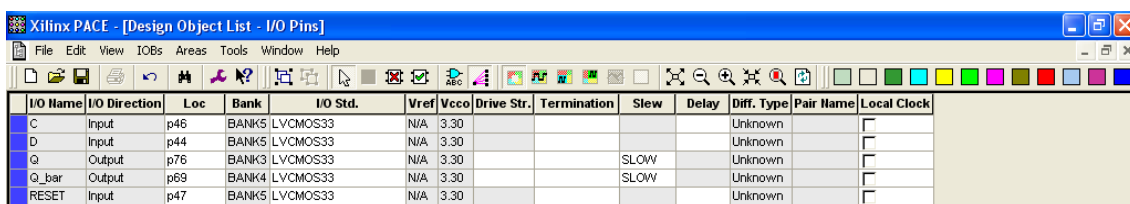
รูปที่ L2.1 โค้ด D Flip-Flop แบบซิงโครนัสรีเซ็ต

การกำหนดขาสัญญาณต่างๆจะใช้ปุ่มกด PB1-PB3 เป็นอินพุตและ LED L2-L3 เป็นเอาต์พุต กล่าวคือ

D = PB1 = INPUT = p44 RESET = PB3 = INPUT = p47 Q = L3 = OUTPUT = p76

C = PB2 = INPUT = p46 Q_bar = L2 = OUTPUT = p69

โดยพิมพ์ใน Assign Package Pins ดังรูปที่ L2.2



I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
C	Input	p46	BANK5	LVC MOS33	N/A	3.30					Unknown		<input type="checkbox"/>
D	Input	p44	BANK5	LVC MOS33	N/A	3.30					Unknown		<input type="checkbox"/>
Q	Output	p76	BANK3	LVC MOS33	N/A	3.30			SLOW		Unknown		<input type="checkbox"/>
Q_bar	Output	p69	BANK4	LVC MOS33	N/A	3.30			SLOW		Unknown		<input type="checkbox"/>
RESET	Input	p47	BANK5	LVC MOS33	N/A	3.30					Unknown		<input type="checkbox"/>

รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมวงจรที่ออกแบบลงชิพ FPGA แล้วให้กดปุ่ม PB2 และให้สังเกตผลลัพธ์ที่ LED L2-L3 จากนั้นให้กดปุ่ม PB1 ค้างไว้แล้วกดปุ่ม PB2 และให้สังเกตผลลัพธ์ที่ LED L2-L3 อีกครั้ง และให้ปล่อยปุ่ม PB1 แล้วกดปุ่ม PB2 แล้วให้สังเกตผลลัพธ์ที่ LED L2-L3 ว่า LED L3 ติดสว่างหรือไม่ ให้กดปุ่ม PB3 แล้วให้สังเกตผลลัพธ์ที่ LED L2-L3 จากนั้นให้กดปุ่ม PB3 ค้างไว้แล้วกดปุ่ม PB2 และให้สังเกตผลลัพธ์ที่ LED L2-L3 ให้สรุปผลการทดลองทั้งหมดความเป็นไปตามทฤษฎีหรือไม่

การทดลองที่ 4.2.3 วงจรดีเบเอย์อย่างง่าย

วัตถุประสงค์

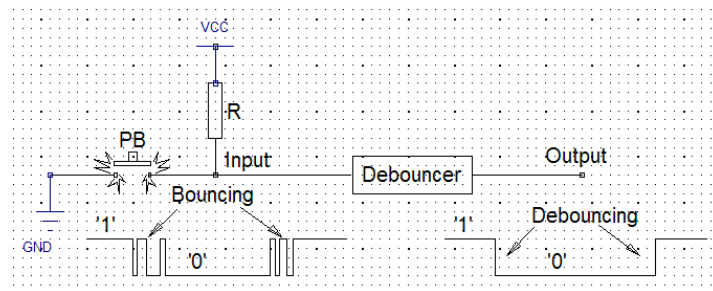
- 1) เพื่อทำความเข้าใจเกี่ยวกับหลักการทำงานของวงจอดีเบเอย์ (Debouncer) อย่างง่าย
- 2) เพื่อสร้างวงจอดีเบเอย์อย่างง่ายแล้วโปรแกรมลงชิพ CPLD และ FPGA

อุปกรณ์ทดลอง

บอร์ด CPLD Explorer XC9572XL หรือ CPLD Explorer XC9572 และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

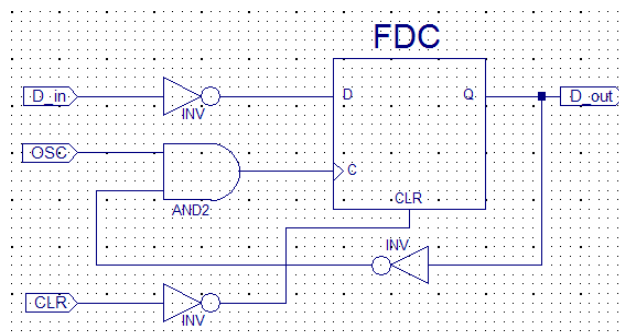
1 สร้างวงจอดีเบเอย์อย่างง่ายด้วย CPLD

การกดคีย์บอร์ดหรือปุ่มกดเพื่อสร้างสัญญาณนาฬิกาครั้งละ 1 พัลส์ (Pulse) อาจจะทำไม่ได้เนื่องจากการกดคีย์บอร์ดในช่วงเริ่มต้นหน้าสัมผัสอาจยังแตะกันไม่สนิท (Bouncing) ทำให้สัญญาณเอาต์พุตที่ได้ไม่แน่นอนจนกว่าหน้าสัมผัสจะแตะกันสนิท และในขณะที่ปล่อยคีย์บอร์ดก็จะเกิดลักษณะเดียวกัน คือ หน้าสัมผัสจะยังไม่แยกจากกันสนิท ซึ่งสัญญาณที่ได้แสดงดังตัวอย่างในรูปที่ L1.1 โดยจะให้สัญญาณนาฬิกาหลายพัลส์ต่างๆ ที่มีการกดคีย์บอร์ดเพียงครั้งเดียว โดยปกติแล้วเวลาเกิดเบเอย์ (Bouncing) ขณะกดหรือปล่อยคีย์บอร์ดจะไม่เกิน 20 มิลลิวินาที ปัญหาการเกิดเบเอย์นี้สามารถแก้ไขได้โดยใช้วงจอดีเบเอย์ (Debouncer) หรือวงจร โมโนสเตเบิล (Monostable) เพื่อทำให้ได้พัลส์เอาต์พุตออกมาครั้งละ 1 พัลส์แสดงดังในรูปที่ L1.1



รูปที่ L1.1 ตัวอย่างสัญญาณอินพุตและเอาต์พุตของวงจอดีเบเอย์ขณะกดคีย์บอร์ดหรือปุ่มกด

วงจอดีเบเอย์อย่างง่ายที่ใช้ D Flip-Flop แบบอะซิงโครนัสเคลียร์ (FDC) มีผังวงจรแสดงดังรูปที่ L1.2



รูปที่ L1.2 วงจอดีเบเอย์อย่างง่ายที่อินพุตเป็นแบบ Active low และเอาต์พุตเป็นแบบ Active High

จากวงจอดีเบเอย์อย่างง่ายในรูปที่ L1.2 เมื่อไม่มีการกดปุ่ม $D_{in} = '1'$ ($D = '0'$) นั้น D Flip-Flop จะถูกทริกด้วย Clock ความถี่สูงจาก OSC ผ่านทางแอนด์เกตได้ตลอดเวลาเพราะว่า $D_{out} = '0'$ จึงทำให้แอนด์เกตอีกขาที่ต่อจากอินเวอร์เตอร์เป็นลอจิก '1' แต่เมื่อมีการกดปุ่ม $D_{in} = '0'$ ($D = '1'$) จะทำให้ D Flip-Flop ถูกทริกและเอาต์พุต $D_{out} = '1'$ ดังนั้นแอนด์เกตอีกขาที่ต่อจาก

อินเวอร์เตอร์จึงเป็นลอจิก '0' ทำให้ Clock จาก OSC ไม่สามารถผ่านแอนด์เกตไปทริก D Flip-Flop ได้อีก ดังนั้นเอาต์พุตของ D Flip-Flop จึงถูก Latch ค่าค้างไว้คือ $D_{out} = '1'$ โดยที่พัลส์ลูกต่อๆ ไปจาก D_{in} ไม่สามารถผ่านไปที่เอาต์พุตของ D Flip-Flop ได้ จนกว่าจะกดปุ่ม CLR ดังนั้นการสร้างพัลส์เอาต์พุต D_{out} 1 พัลส์จะต้องกดปุ่ม D_{in} 1 ครั้งและกดปุ่มเคลียร์ CLR อีก 1 ครั้ง วงจรดีเบบเซอร์นี้จะทำงานแบบ Manual ได้ดีมาก แต่อย่างไรก็ตามวงจรดีเบบเซอร์นี้จะมีประสิทธิภาพต่ำมากเมื่อทำงานโดยอัตโนมัติ ด้วยการป้อนความถี่ประมาณ 2-4 Hz ที่ขา CLR ซึ่งทำให้วงจรนี้เคลียร์เอาต์พุตทุกๆ 0.25-0.5 วินาที ดังนั้นถ้ากดเร็วไปก็อาจจะไม่ให้พัลส์ออกมา แต่ถ้ากดช้าไปก็อาจให้พัลส์เกิน 1 พัลส์ หรือถ้ากดปุ่มค้างไว้จะให้พัลส์ต่อเนื่องหรือ Clock ความถี่ประมาณ 2-4 Hz เช่นเดียวกับที่ขา CLR ซึ่งการออกแบบวงจรดีเบบเซอร์ประสิทธิภาพสูงจะอธิบายในข้อ 4.5 และข้อ 4.6 (วิธี Finite state machine)

จากหลักการที่กล่าวมาแล้ว เราสามารถเขียนโค้ดของวงจรดีเบบเซอร์ได้ดังรูปที่ L1.3 โดยเขียนไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_2_3vcxl

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity ex4_2_3vcxl is
6      Port ( OSC,D_in,CLR : in  STD_LOGIC;
7            D_out : out  STD_LOGIC);
8  end ex4_2_3vcxl;
9
10 architecture Behavioral of ex4_2_3vcxl is
11     signal QT,C_DB : STD_LOGIC;
12 begin
13     C_DB <= not QT and OSC;
14     process(OSC,CLR)
15     begin
16         if CLR='0' then QT <= '0';
17         elsif C_DB'event and C_DB='1' then
18             if D_in = '0' then QT <= '1';
19             end if;
20         end if;
21     end process;
22     D_out <= QT;
23 end Behavioral;

```


รูปที่ L1.3 โค้ดของวงจรดีเบบเซอร์อย่างง่ายสำหรับปุ่มกดและเคลียร์แบบ Active low

การกำหนดคาสัญญาณต่างๆ จะใช้ฮอสซิลเลเตอร์ $OSC = 32.768\text{kHz}$ และปุ่มกด PB1-PB2 เป็นอินพุต และมี LED4 เป็นเอาต์พุต กล่าวคือ

$D_{in} = PB1 = INPUT = p39$ $Q = LED1 = OUTPUT = p38$

$CLR = PB2 = INPUT = p40$ $OSC = OSC = INPUT = p5$

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้



I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
OSC	Input	p5	1	9		
D_in	Input	p39	2	9		
CLR	Input	p40	2	11		
D_out	Output	p35	2	2	SLOW	

รูปที่ L1.4 Assign Package Pins

หลังจากโปรแกรมวงจรที่ออกแบบลงชิพ CPLD แล้วให้กดปุ่ม PB1 และ PB2 สลับกันไปเรื่อยแล้วให้สังเกตดูผลที่ LED4 จากนั้นให้สรุปผลการทดลองว่าเป็นไปตามทฤษฎีหรือไม่

2 สร้างวงจรดีเบเอนเซอร์อย่างง่ายด้วย FPGA

ให้ทำความเข้าใจหลักการทำงานของวงจรดีเบเอนเซอร์ (Debouncer) หรือโมโนสเตเบิล (Monostable) ในข้อ 1 จากนั้นจึงเขียนโค้ดของวงจรดีเบเอนเซอร์ได้ดังรูปที่ L2.1 โดยเขียนไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_2_3vf

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity ex4_2_3vf is
6      Port ( OSC,D_in,CLR : in  STD_LOGIC;
7            D_out : out  STD_LOGIC);
8  end ex4_2_3vf;
9
10 architecture Behavioral of ex4_2_3vf is
11     signal QT,C_DB : STD_LOGIC;
12 begin
13     C_DB <= not QT and OSC;
14     process(OSC,CLR)
15     begin
16         if CLR='0' then QT <= '0';
17         elsif C_DB'event and C_DB='1' then
18             if D_in='0' then QT <= '1';
19             end if;
20         end if;
21     end process;
22     D_out <= QT;
23 end Behavioral;

```

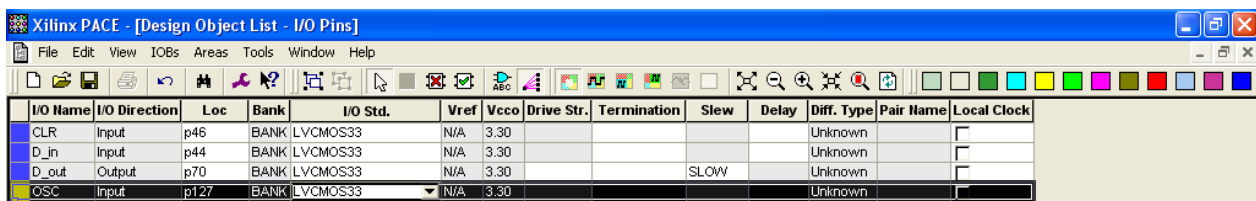
รูปที่ L2.1 โค้ดของวงจรดีเบเอนเซอร์อย่างง่ายสำหรับปุ่มกดและเคลียร์แบบ Active low

การกำหนดขาสัญญาณต่างๆ จะใช้ออสซิลเลเตอร์ OSC = 25MHz ปุ่มกด PB1-PB2 เป็นอินพุต และ LED L0 เป็นเอาต์พุต กล่าวคือ

D_in = PB1 = INPUT = p44 Q = L0 = OUTPUT = p70

CLR = PB2 = INPUT = p46 OSC = OSC = INPUT = p127

โดยพิมพ์ใน Assign Package Pins ดังรูปที่ L2.2



I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
CLR	Input	p46	BANK	LVCMOS33	N/A	3.30					Unknown		<input type="checkbox"/>
D_in	Input	p44	BANK	LVCMOS33	N/A	3.30					Unknown		<input type="checkbox"/>
D_out	Output	p70	BANK	LVCMOS33	N/A	3.30			SLOW		Unknown		<input type="checkbox"/>
OSC	Input	p127	BANK	LVCMOS33	N/A	3.30					Unknown		<input type="checkbox"/>

รูปที่ L2.2 Assign Package Pins

หลังจากโปรแกรมวงจรที่ออกแบบลงชิพ FPGA แล้วให้กดปุ่ม PB1 และ PB2 สลับกันไปเรื่อยแล้วให้สังเกตผลลัพธ์ LED L0 จากนั้นให้สรุปผลการทดลองว่าเป็นไปตามทฤษฎีหรือไม่

การทดลองที่ 4.2.4 JK Flip-Flop และ T Flip-Flop แบบอะซิงโครนัสเคลียร์

วัตถุประสงค์

- 1) เพื่อทำความเข้าใจเกี่ยวกับวงจร JK Flip-Flop และ T Flip-Flop แบบอะซิงโครนัสเคลียร์และวงจรดีเบเอนเซอร์
- 2) เพื่อสร้าง JK Flip-Flop และวงจรดีเบเอนเซอร์ด้วยโค้ด VHDL แล้วโปรแกรมลง CPLD และ FPGA

อุปกรณ์ทดลอง

บอร์ด CPLD Explorer XC9572XL หรือ CPLD Explorer XC9572 และ FPGA Discovery-III XC3S200F หรือ FPGA Discovery-III XC3S200F4

1 สร้างวงจร JK Flip-Flop แบบอะซิงโครนัสเคลียร์ด้วย CPLD

จากการทดลองที่ 4.2.3 ให้สร้างไฟล์โค้ดวงจรดีเบเอนเซอร์ดังรูปที่ L1.1 เพื่อทำเป็น Component ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ch4vcx1_DEBOUNCER (ซึ่งโค้ดนี้ต้องถูกตรวจสอบความถูกต้องแล้ว)

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity ch4vcx1_DEBOUNCER is
6      Port ( OSC,D_in,CLR : in  STD_LOGIC;
7            D_out : out  STD_LOGIC);
8  end ch4vcx1_DEBOUNCER;
9
10 architecture Behavioral of ch4vcx1_DEBOUNCER is
11     signal QT,C_DB : STD_LOGIC;
12 begin
13     C_DB <= not QT and OSC;
14 process(OSC,CLR)
15     begin
16         if CLR='0' then QT <= '0';
17         elsif C_DB'event and C_DB='1' then
18             if D_in='0' then QT <= '1';
19             end if;
20         end if;
21 end process;
22     D_out <= QT;
23 end Behavioral;

```

รูปที่ L1.1 โค้ดของวงจรดีเบเอนเซอร์อย่างง่ายสำหรับปุ่มกดและเคลียร์แบบ Active low

จากนั้นเขียนไฟล์โค้ด JK Flip-Flop แบบอะซิงโครนัสเคลียร์ที่รวม Component ของวงจรดีเบเอนเซอร์ไว้แล้วแสดงดังในรูปที่ L1.2 ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_2_4vcx1 ซึ่งขั้นตอน Add Source ของไฟล์ Component เข้าไปในไฟล์ Project สามารถอ่านได้จากข้อ 2.18.2.1 ของบทที่ 2 (ดูรูปที่ E3.13 ถึงรูปที่ E3.20)

```

2  -----Main code:JK Flip-Flop with DEBOUNCER-----
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5
6  entity ex4_2_4vcx1 is
7      Port ( OSC,Din,CLR_DB,J,K,CLR : in  STD_LOGIC;
8            Q,Q_bar : out  STD_LOGIC);
9  end ex4_2_4vcx1;
10
11 architecture Behavioral of ex4_2_4vcx1 is

```

(ต่อ)

```

12 component ch4vcx1_DEBOUNCER
13   Port ( OSC,D_in,CLR : in  STD_LOGIC;
14         D_out : out  STD_LOGIC);
15 end component;
16 signal C_DB,C,QT : STD_LOGIC;
17 signal JK : STD_LOGIC_VECTOR(1 downto 0);
18 begin
19   -----Debounce-----
20   DEBOUNCER_JKFF : ch4vcx1_DEBOUNCER port map ( OSC => OSC,
21                                                 D_in => Din,
22                                                 CLR => CLR_DB,
23                                                 D_out=> C );
24   -----JK FLIP-FLOP-----
25   JK <= (J&K); -- Concatenation
26   process(C,CLR)
27   begin
28     if CLR='1' then QT <= '0';
29     elsif (C'event and C='1') then
30       if JK="00" then QT <= QT;
31       elsif JK="01" then QT <= '0';
32       elsif JK="10" then QT <= '1';
33       elsif JK="11" then QT <= not QT;
34     end if;
35   end if;
36 end process;
37   Q <= QT; Q_bar <= not QT;
38 end Behavioral;

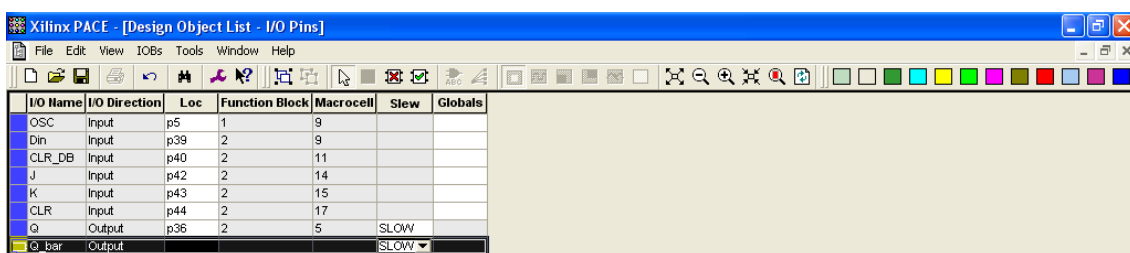
```

รูปที่ L1.2 โค้ด JK Flip-Flop ที่รวมวงจรดีเบานเซอร์ในรูปของ Component ไว้แล้ว

การกำหนดขาสัญญาณต่างๆ ใช้ฮอสซิลเลเตอร์ OSC = 32.768 kHz และปุ่มกด PB1-PB4 เป็นอินพุต มี LED3-LED4 เป็นเอาต์พุตกล่าวคือ

Din = PB1 = INPUT = p39	Q = LED3 = OUTPUT = p36
CLR_DB = PB2 = INPUT = p40	Q_bar = LED4 = OUTPUT = p35
J = PB3 (Slide SW1) = INPUT = p42	OSC = OSC = INPUT = p5
K = PB4 (Slide SW2) = INPUT = p43	CLR = PB5 (Slide SW3) = INPUT = p44

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้



I/O Name	I/O Direction	Loc	Function Block	Macrocell	Stew	Globals
OSC	Input	p5	1	9		
Din	Input	p39	2	9		
CLR_DB	Input	p40	2	11		
J	Input	p42	2	14		
K	Input	p43	2	15		
CLR	Input	p44	2	17		
Q	Output	p36	2	5	SLOW	
Q_bar	Output				SLOW	

รูปที่ L1.3 Assign Package Pins

หลังจากโปรแกรมวงจรลง CPLD แล้วให้เลื่อน Slide SW3 ไปที่ ON (CLR = '0') จากนั้นเซตค่า J และ K เป็นค่าต่างๆ เริ่มจาก J = '0' และ K = '0' ไปจนกระทั่ง J = '1' และ K = '1' โดยใช้ Slide SW1 และ Slide SW2 (Slide SW1 ON แล้ว J = '0', OFF แล้ว J = '1' และ Slide SW2 ON แล้ว K = '0', OFF แล้ว K = '1') แล้วกดปุ่ม PB1 และปุ่ม PB2 สลับกันเพื่อสร้าง Clock พร้อมกับให้สังเกตดูที่ LED3 และ LED4 ในแต่ละเงื่อนไขว่าเป็นตามทฤษฎีหรือไม่ กดปุ่ม PB1 และปุ่ม PB2 สลับกันจนกระทั่ง LED3 ติดสว่างเสร็จแล้วเลื่อน Slide SW3 ไปที่ตำแหน่ง OFF (CLR = '1') พร้อมกับให้สังเกตดูที่ LED3 ว่าดับเป็นตามทฤษฎีหรือไม่ (กรณีที่เป็น T Flip-Flop ให้เซต J = K = '1') เสร็จแล้วจึงบันทึกผลการทดลอง

2 สร้างวงจร JK Flip-Flop แบบอะซิงโครนัสเคลียร์ด้วย FPGA

จากการทดลองที่ 4.2.3 ให้สร้างไฟล์โค้ดวงจรดีเบานเซอร์ดังรูปที่ L2.1 เพื่อทำเป็น Component ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ch4vf_DEBOUNCER (ซึ่งโค้ดนี้ต้องถูกตรวจสอบความถูกต้องแล้ว)

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity ch4vf_DEBOUNCER is
6      Port ( OSC,D_in,CLR : in  STD_LOGIC;
7            D_out : out  STD_LOGIC);
8  end ch4vf_DEBOUNCER;
9
10 architecture Behavioral of ch4vf_DEBOUNCER is
11     signal QT,C_DB : STD_LOGIC;
12 begin
13     C_DB <= not QT and OSC;
14 process(OSC,CLR)
15 begin
16     if CLR='0' then QT <= '0';
17     elsif C_DB'event and C_DB='1' then
18         if D_in = '0' then QT <= '1';
19         end if;
20     end if;
21 end process;
22 D_out <= QT;
23 end Behavioral;

```

รูปที่ L2.1 โค้ดของวงจรดีเบานเซอร์อย่างง่ายสำหรับปุ่มกดและเคลียร์แบบ Active low

จากนั้นเขียนไฟล์โค้ด JK Flip-Flop แบบบอชิงโครนัสเคลียร์ที่รวม Component ของวงจรดีเบานเซอร์ไว้แล้วแสดงดังในรูปที่ L2.2 ไว้ใน Project Location ชื่อ ch4v กำหนด Project Name และ Source File ชื่อ ex4_2_4vf ซึ่งขั้นตอน Add Source ของไฟล์ Component เข้าไปในไฟล์ Project ของ FPGA นั้นจะเหมือนกับ CPLD ซึ่งอธิบายในข้อ 2.18.2.1 ของบทที่ 2 (ดูรูปที่ E3.13 ถึงรูปที่ E3.20) และในข้อ 2.19 ของบทที่ 2

```

2  -----Main code:JK Flip-Flop with DEBOUNCER-----
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5
6  entity ex4_2_4vf is
7      Port ( OSC,Din,CLR_DB,J,K,CLR : in  STD_LOGIC;
8            Q,Q_bar : out  STD_LOGIC);
9  end ex4_2_4vf;
10
11 architecture Behavioral of ex4_2_4vf is
12     component ch4vf_DEBOUNCER
13     Port ( OSC,D_in,CLR : in  STD_LOGIC;
14           D_out : out  STD_LOGIC);
15     end component;
16     signal C_DB,C,QT : STD_LOGIC;
17     signal JK : STD_LOGIC_VECTOR(1 downto 0);
18 begin
19     -----Debounce-----
20     DEBOUNCER_JKFF : ch4vf_DEBOUNCER port map ( OSC => OSC,
21                                                  D_in => Din,
22                                                  CLR => CLR_DB,
23                                                  D_out=> C );
24     -----JK FLIP-FLOP-----
25     JK <= (J&K); -- Concatenation
26 process(C,CLR)
27 begin
28     if CLR='1' then QT <= '0';
29     elsif (C'event and C='1') then
30         if JK="00" then QT <= QT;
31         elsif JK="01" then QT <= '0';
32         elsif JK="10" then QT <= '1';
33         elsif JK="11" then QT <= not QT;

```

(ต่อ)

```

34         end if;
35     end if;
36 end process;
37 Q <= QT; Q_bar <= not QT;
38 end Behavioral;

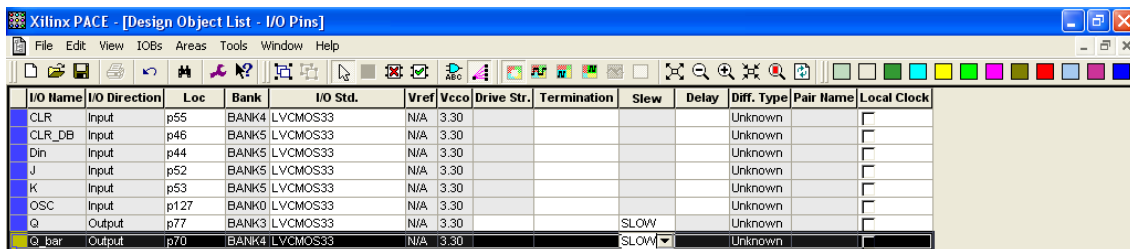
```

รูปที่ L2.2 โค้ด JK Flip-Flop ที่รวมวงจรดีโมเชอร์ในรูปของ Component ไว้แล้ว

การกำหนดขาสัญญาณต่างๆ โดยใช้ข้อสวิตช์เตอร์ OSC = 25MHz และปุ่มกด PB1-PB2 และ Dip SW1-Dip SW3 เป็นอินพุต มี LED L0-L1 เป็นเอาต์พุตกล่าวคือ

Din = PB1	= INPUT = p44	Q = L1	= OUTPUT = p77
CLR_DB = PB2	= INPUT = p46	Q_bar = L0	= OUTPUT = p70
J = Dip SW1	= INPUT = p52	OSC = OSC	= INPUT = p127
K = Dip SW2	= INPUT = p53	CLR = Dip SW3	= INPUT = p55

โดยพิมพ์ใน Assign Package Pins สรุปดังนี้



I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
CLR	Input	p55	BANK4	LVC MOS33	N/A	3.30					Unknown		<input type="checkbox"/>
CLR_DB	Input	p46	BANK5	LVC MOS33	N/A	3.30					Unknown		<input type="checkbox"/>
Din	Input	p44	BANK5	LVC MOS33	N/A	3.30					Unknown		<input type="checkbox"/>
J	Input	p52	BANK5	LVC MOS33	N/A	3.30					Unknown		<input type="checkbox"/>
K	Input	p53	BANK5	LVC MOS33	N/A	3.30					Unknown		<input type="checkbox"/>
OSC	Input	p127	BANK0	LVC MOS33	N/A	3.30					Unknown		<input type="checkbox"/>
Q	Output	p77	BANK3	LVC MOS33	N/A	3.30			SLOW		Unknown		<input type="checkbox"/>
Q_bar	Output	p70	BANK4	LVC MOS33	N/A	3.30			SLOW		Unknown		<input type="checkbox"/>

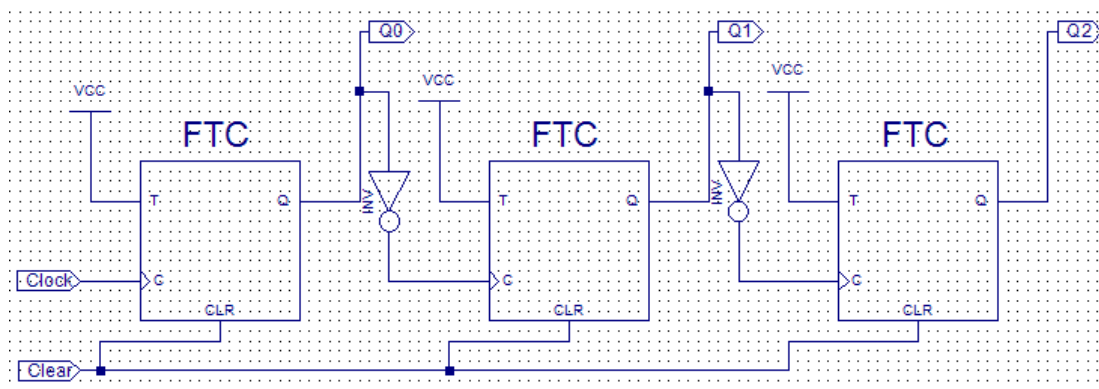
รูปที่ L2.3 Assign Package Pins

หลังจากโปรแกรมวงจรลง FPGA แล้วให้เซต Dip SW3 ไปที่ ON (CLR = '0') จากนั้นเซตค่า J และ K เป็นค่าต่างๆ เริ่มจาก J = '0' และ K = '0' ไปจนกระทั่ง J = '1' และ K = '1' โดยใช้ Dip SW1 และ Dip SW2 (Dip SW1 ON แล้ว J = '0', OFF แล้ว J = '1' และ Dip SW2 ON แล้ว K = '0', OFF แล้ว K = '1') แล้วกดปุ่ม PB1 และปุ่ม PB2 สลับกันเพื่อสร้าง Clock พร้อมกับให้สังเกตดูที่ LED L0 และ L1 ในแต่ละเงื่อนไขว่าเป็นตามทฤษฎีหรือไม่ กดปุ่ม PB1 และปุ่ม PB2 สลับกันจนกระทั่ง LED L1 ติดสว่างเสร็จแล้วให้เซต Dip SW3 ไปที่ OFF (CLR = '1') พร้อมกับให้สังเกตดูที่ LED L1 ว่าดับเป็นตามทฤษฎีหรือไม่ (กรณีที่เป็น T Flip-Flop ให้เซต J = K = '1') เสร็จแล้วจึงบันทึกผลการทดลอง

4.3 วงจรนับเลขไบนารี (Binary counter)

4.3.1 วงจรนับแบบอะซิงโครนัส (Asynchronous counter)

วงจรนับแบบอะซิงโครนัสหรือแบบรีปเปิล (Ripple counter) จะสร้างจาก T Flip-Flop โดยเซต $T = '1'$ หรือถ้าใช้ JK Flip-Flop จะเซต $J = K = '1'$ เพื่อทำงานในโหมด Toggle ทุกๆ ครั้งที่มีการทริกด้วยขอบขาขึ้นของ Clock (หรือออกแบบให้ทริกขอบขาลงก็ได้) วงจรนับขึ้นแบบเลขไบนารี (Binary up-counter) ขนาด 3 บิตแสดงดังรูปที่ 4.10 ซึ่งจะใช้ T Flip-Flop แบบ อะซิงโครนัสเคลียร์ (FTC) ไปทำเป็นวงจรนับ 8 กล่าวคือนับได้สูงสุด $= 2^N - 1$ โดยที่ N คือ จำนวนบิตหรือจำนวน Flip-Flop วงจรนับจะเคลียร์เมื่อ $Clear = '1'$ ในกรณีของวงจรนับขึ้นเราจะใส่อินเวอร์เตอร์ INV เพื่อให้เอาต์พุตจาก Q_0 และ Q_1 กลายเป็นขอบขาขึ้นเพื่อใช้ทริก T Flip-Flop ตัวถัดไป และขอให้สังเกตว่าการ Toggle ของ T Flip-Flop แต่ละตัวจะทำให้ความถี่สัญญาณนาฬิกา (Clock) ที่เอาต์พุตลดลงครึ่งหนึ่งจากความถี่เดิมหรือทำหน้าที่เป็นวงจรหารสองของความถี่เดิมนั่นเอง



รูปที่ 4.10 ฟังวงจรนับขึ้นแบบอะซิงโครนัสหรือแบบรีปเปิล 3 บิต

จากผังวงจรนับขึ้นแบบอะซิงโครนัสหรือแบบรีปเปิล 3 บิตในรูปที่ 4.10 นั้นเราสามารถนำโค้ดของ T Flip-Flop แบบ อะซิงโครนัสเคลียร์แสดงดังรูปที่ 4.11 มาทำเป็น Component เพื่อนำไปเขียนโค้ดของวงจรนับ 8 ได้ดังรูปที่ 4.12 ซึ่งผลการสังเคราะห์วงจร (View technology schematic) จะได้ดังรูปที่ 4.13 และผลจำลองการทำงานของวงจรนับจะได้ดังรูปที่ 4.14

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity TFF is
6      Port ( C,CLR,T : in  STD_LOGIC;
7            Q : out  STD_LOGIC);
8  end TFF;
9
10 architecture Behavioral of TFF is
11     signal QT : STD_LOGIC;
12 begin
13     process(C,CLR)
14     begin
15         if CLR='1' then QT <= '0';
16         elsif (C'event and C='1') then
17             if T='0' then QT <= QT;
18             elsif T='1' then QT <= not QT;
19             end if;
20         end if;
21     end process;
22     Q <= QT;
23 end Behavioral;

```

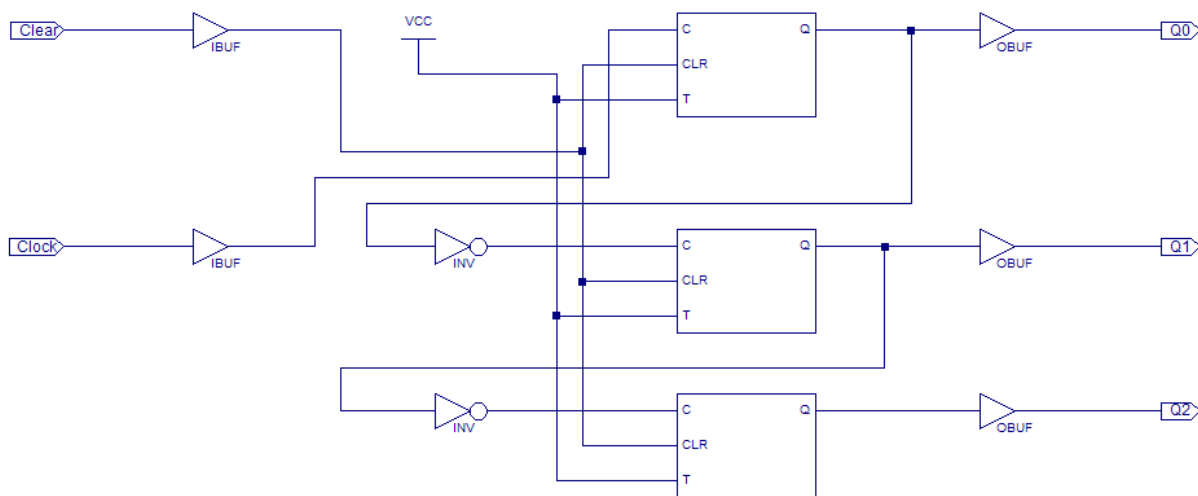
รูปที่ 4.11 โค้ดของวงจร T Flip-Flop แบบอะซิงโครนัสเคลียร์ที่ทริกด้วยขอบขาขึ้นที่จะนำไปสร้างเป็น Component


```

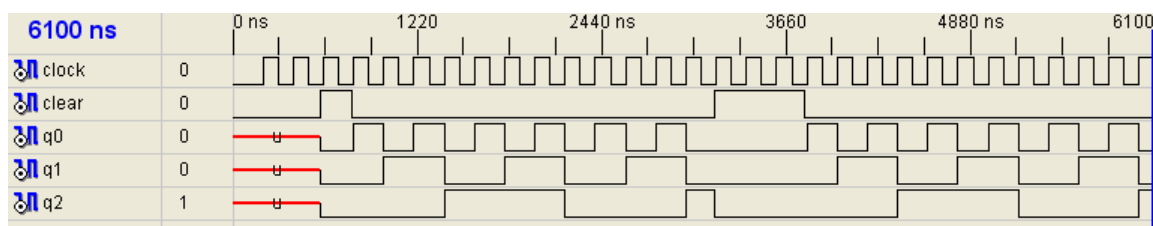
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity COUNTER_3BIT_ASYNC_UP is
6      Port ( Clock,Clear : in  STD_LOGIC;
7            Q0,Q1,Q2 : out  STD_LOGIC);
8  end COUNTER_3BIT_ASYNC_UP;
9
10 architecture Behavioral of COUNTER_3BIT_ASYNC_UP is
11     signal Tin0,Tin1,Tin2 : STD_LOGIC;
12     signal C1,C2 : STD_LOGIC;
13     signal Q0T,Q1T,Q2T : STD_LOGIC;
14     component TFF
15         Port ( C,CLR,T : in  STD_LOGIC;
16               Q : out  STD_LOGIC);
17     end component;
18 begin
19     Tin0 <= '1'; -- T=Vcc
20     Tin1 <= '1'; -- T=Vcc
21     Tin2 <= '1'; -- T=Vcc
22     BIT_0 : TFF port map( C=>Clock,CLR=>Clear,T=>Tin0,Q=>Q0T );
23     C1 <= not Q0T;
24     BIT_1 : TFF port map( C=>C1,CLR=>Clear,T=>Tin1,Q=>Q1T );
25     C2 <= not Q1T;
26     BIT_2 : TFF port map( C=>C2,CLR=>Clear,T=>Tin2,Q=>Q2T );
27     Q0 <= Q0T;
28     Q1 <= Q1T;
29     Q2 <= Q2T;
30 end Behavioral;

```

รูปที่ 4.12 โค้ดของวงจรนับขึ้นแบบอะซิงโครนัส 3 บิตที่สร้างจาก Component ในรูปที่ 4.11

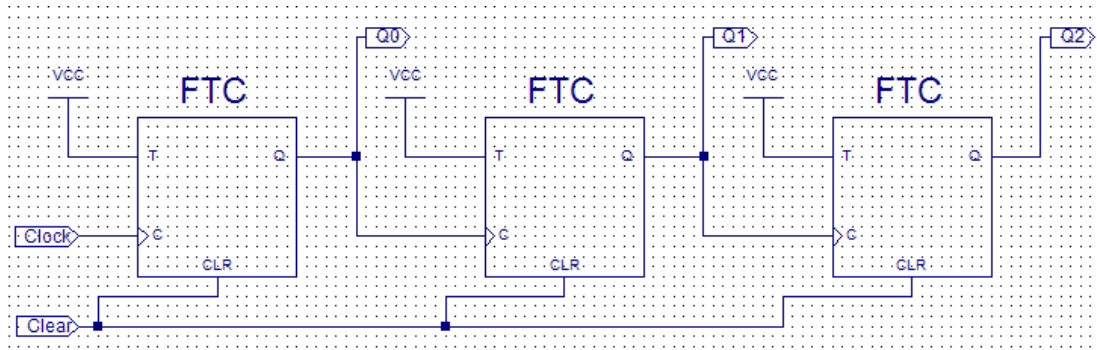


รูปที่ 4.13 ฟังก์ชันการทำงานของวงจรนับขึ้นแบบอะซิงโครนัส 3 บิตที่ได้จากการสังเคราะห์วงจร (View technology schematic)



รูปที่ 4.14 ผลจำลองการทำงานของวงจรนับขึ้นแบบอะซิงโครนัส 3 บิต ซึ่งค่า U (Uninitial) เกิดจากการไม่ได้ค่าเริ่มต้น

ในการทำนองเดียวกันการออกแบบวงจรนับลงแบบอะซิงโครนัสหรือแบบรีปีต 3 บิตนั้นจะได้ดังรูปที่ 4.15 ซึ่งวงจรนี้จะใช้ T Flip-Flop แบบอะซิงโครนัสเคลียร์ (FTC) มาทำเป็นวงจรนับ 8 และโค้ดของวงจรนับลงแบบอะซิงโครนัส 3 บิตที่สร้างจาก Component ในรูปที่ 4.11 นั้นจะได้ดังรูปที่ 4.16 และผลจำลองการทำงานวงจรนับจะดังรูปที่ 4.17



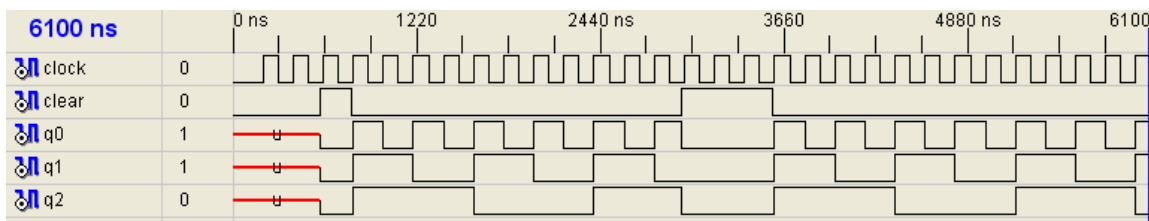
รูปที่ 4.15 ฟังวงจรนับลงแบบอะซิงโครนัสหรือแบบรีปีต 3 บิต

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity COUNTER_3BIT_ASYNC_DN is
6      Port ( Clock,Clear : in  STD_LOGIC;
7            Q0,Q1,Q2 : out  STD_LOGIC);
8  end COUNTER_3BIT_ASYNC_DN;
9
10 architecture Behavioral of COUNTER_3BIT_ASYNC_DN is
11     signal Tin0,Tin1,Tin2 : STD_LOGIC;
12     signal Q0T,Q1T,Q2T : STD_LOGIC;
13     component TFF
14         Port ( C,CLR,T : in  STD_LOGIC;
15               Q : out  STD_LOGIC);
16     end component;
17 begin
18     Tin0 <= '1'; -- T=Vcc
19     Tin1 <= '1'; -- T=Vcc
20     Tin2 <= '1'; -- T=Vcc
21     BIT_0 : TFF port map( C=>Clock,CLR=>Clear,T=>Tin0,Q=>Q0T );
22     BIT_1 : TFF port map( C=>Q0T,CLR=>Clear,T=>Tin1,Q=>Q1T );
23     BIT_2 : TFF port map( C=>Q1T,CLR=>Clear,T=>Tin2,Q=>Q2T );
24     Q0 <= Q0T;
25     Q1 <= Q1T;
26     Q2 <= Q2T;
27 end Behavioral;

```

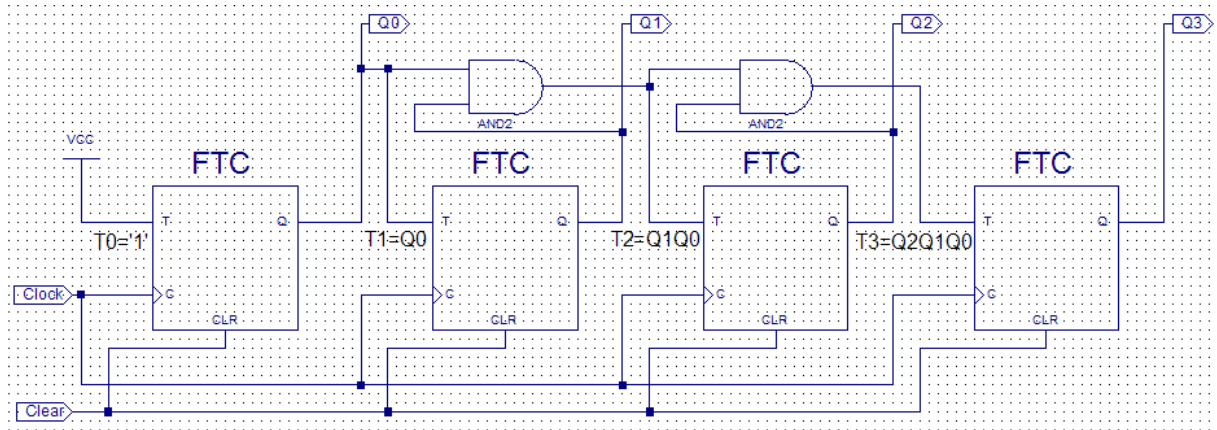
รูปที่ 4.16 วงจรนับลงแบบอะซิงโครนัส 3 บิตที่สร้างจาก Component ในรูปที่ 4.11



รูปที่ 4.17 ผลจำลองการทำงานวงจรนับลงแบบอะซิงโครนัส 3 บิต (U = Uninitial)

4.3.2 วงจรนับแบบซิงโครนัส (Synchronous counter)

วงจรรนับแบบซิงโครนัสเป็นวงจรที่ถูกออกแบบให้ Clock ทรัก Flip-Flop ทุกตัวพร้อมกัน ทำให้วงจรรนับมีเวลาหน่วงภายในน้อยจึงสามารถทำงานที่ความถี่สูงได้ วงจรรนับขึ้นแบบซิงโครนัส 4 บิตแสดงดังรูปที่ 4.18 ซึ่งวงจรรนับนี้จะใช้ T Flip-Flop แบบอะซิงโครนัสเคลียร์ (FTC) มาทำเป็นวงจรรนับ ซึ่งปกติจะใช้ T Flip-Flop หรือใช้ JK Flip-Flop (โดยเซตให้ $J = K = '1'$) มาทำเป็นวงจรรนับนี้ได้ ซึ่งค่าสูงสุดที่นับได้ $= 2^N - 1$ โดยที่ N คือ จำนวนบิต



รูปที่ 4.18 วงจรรนับขึ้นแบบซิงโครนัส 4 บิต

จากรูปที่ 4.18 เราสามารถเขียนโค้ดวงจรรนับนี้ได้ดังรูปที่ 4.19 โดยสรุปเป็นเงื่อนไขต่างๆ ไปในการออกแบบได้ดังนี้

$$T0 = 1$$

$$T1 = Q0$$

$$T2 = Q1 \cdot T1 = Q1 \cdot Q0$$

$$T3 = Q2 \cdot T2 = Q2 \cdot Q1 \cdot Q0$$

หรือ

$$T(N) = Q(N-1) \cdot T(N-1) = Q(N-1) \cdot Q(N-2) \dots Q1 \cdot Q0$$

โดยที่ $T(N)$ คือ เอาต์พุตของสัญญาณที่ใช้เป็นอินพุตของ T Flip-Flop บิตที่ N

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity COUNTER_4BIT_SYNC_UP is
6     Port ( Clock, Clear : in STD_LOGIC;
7           Q0, Q1, Q2, Q3 : out STD_LOGIC);
8 end COUNTER_4BIT_SYNC_UP;
9
10 architecture Behavioral of COUNTER_4BIT_SYNC_UP is
11     signal T0, T1, T2, T3 : STD_LOGIC;
12     signal Q0T, Q1T, Q2T, Q3T : STD_LOGIC;
13     component TFF
14         Port ( C, CLR, T : in STD_LOGIC;
15               Q : out STD_LOGIC);
16     end component;

```

(ต่อ)

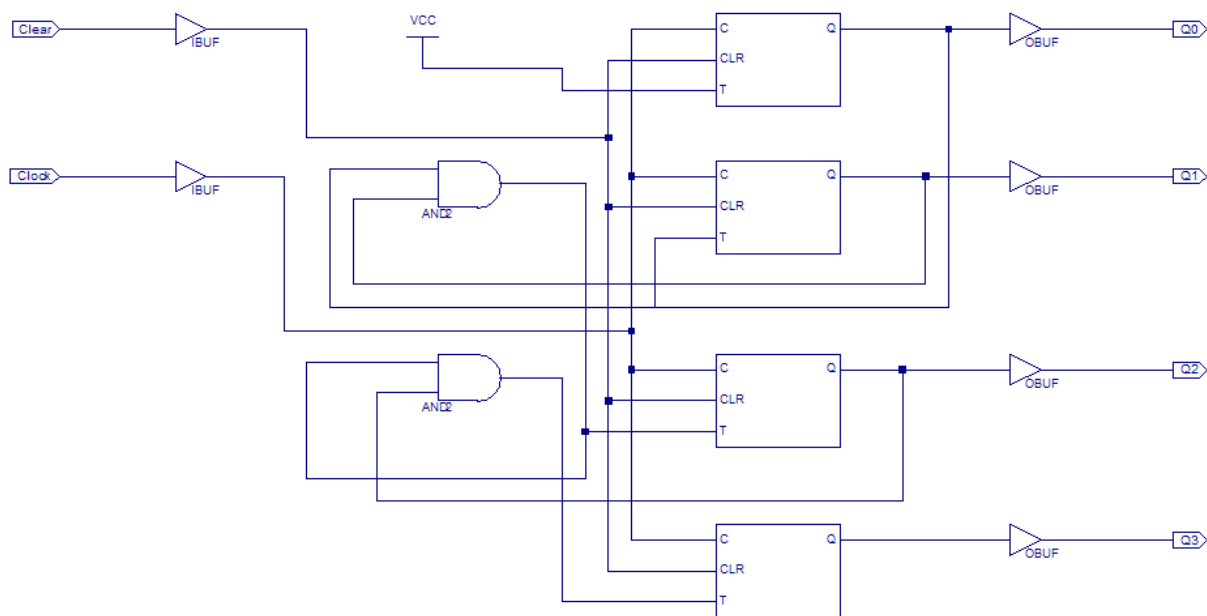
```

17 begin
18   T0 <= '1'; -- T=Vcc
19   BIT_0 : TFF port map( C=>Clock,CLR=>Clear,T=>T0,Q=>Q0T );
20   T1 <= Q0T;
21   BIT_1 : TFF port map( C=>Clock,CLR=>Clear,T=>T1,Q=>Q1T );
22   T2 <= Q1T and T1;
23   BIT_2 : TFF port map( C=>Clock,CLR=>Clear,T=>T2,Q=>Q2T );
24   T3 <= Q2T and T2;
25   BIT_3 : TFF port map( C=>Clock,CLR=>Clear,T=>T3,Q=>Q3T );
26   Q0 <= Q0T;
27   Q1 <= Q1T;
28   Q2 <= Q2T;
29   Q3 <= Q3T;
30 end Behavioral;

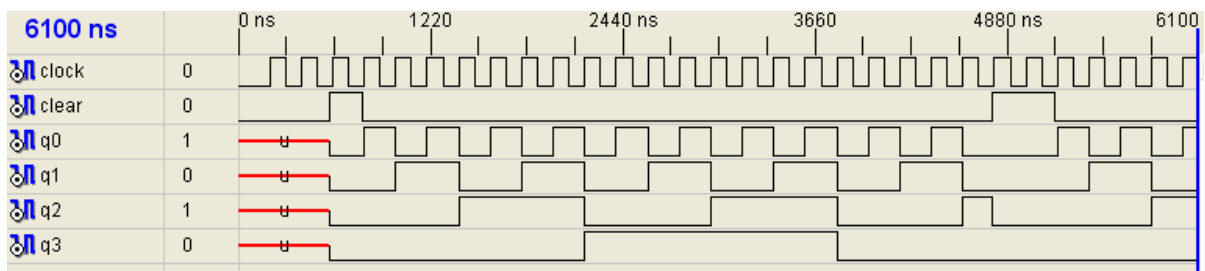
```

รูปที่ 4.19 โค้ดของวงจรนับขึ้นแบบซิงโครนัส 4 บิตที่สร้างจาก Component ในรูปที่ 4.11

จากโค้ดของวงจรนับขึ้นแบบซิงโครนัส 4 บิตในรูปที่ 4.19 เมื่อทำการสังเคราะห์วงจร (View technology schematic) จะได้ดังรูปที่ 4.20 และผลจำลองการทำงานของวงจรจะได้ดังรูปที่ 4.21

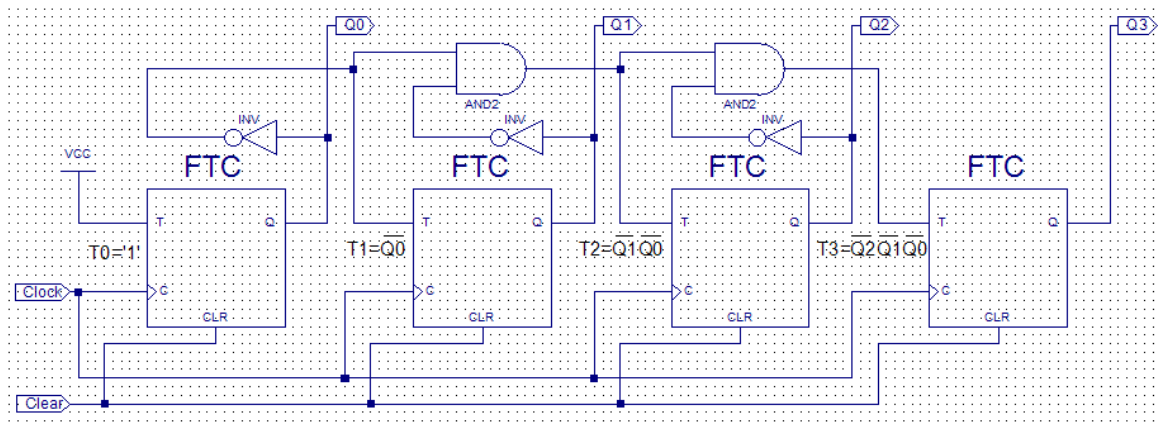


รูปที่ 4.20 ผังวงจรนับขึ้นแบบซิงโครนัส 4 บิตที่ได้จากการสังเคราะห์วงจร (View technology schematic)



รูปที่ 4.21 ผลจำลองการทำงานของวงจรนับขึ้นแบบซิงโครนัส 4 บิตที่ใช้โค้ดในรูปที่ 4.19 (U = Uninitial)

ในทำนองเดียวกันเราสามารถเขียนโค้ดของวงจรนับลงแบบซิงโครนัส 4 บิตจากผังวงจรในรูปที่ 4.22 ได้ดังรูปที่ 4.23 และเมื่อทำการจำลองการทำงานของวงจรจะได้ดังรูปที่ 4.24 ซึ่งค่าสูงสุดที่นับได้ $= 2^N - 1$ โดยที่ N คือ จำนวนบิต



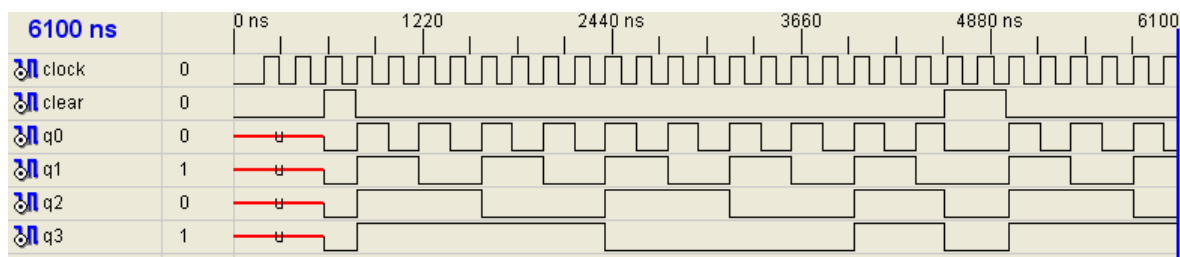
รูปที่ 4.22 ผังวงจรนับลงแบบซิงโครนัส 4 บิต

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity COUNTER_4BIT_SYNC_DN is
6      Port ( Clock, Clear : in  STD_LOGIC;
7            Q0,Q1,Q2,Q3 : out  STD_LOGIC);
8  end COUNTER_4BIT_SYNC_DN;
9
10 architecture Behavioral of COUNTER_4BIT_SYNC_DN is
11     signal T0,T1,T2,T3 : STD_LOGIC;
12     signal Q0T,Q1T,Q2T,Q3T : STD_LOGIC;
13     component TFF
14         Port ( C,CLR,T : in  STD_LOGIC;
15               Q : out  STD_LOGIC);
16     end component;
17 begin
18     T0 <= '1'; -- T=Vcc
19     BIT_0 : TFF port map( C=>Clock, CLR=>Clear, T=>T0, Q=>Q0T );
20     T1 <= not Q0T;
21     BIT_1 : TFF port map( C=>Clock, CLR=>Clear, T=>T1, Q=>Q1T );
22     T2 <= not Q1T and T1;
23     BIT_2 : TFF port map( C=>Clock, CLR=>Clear, T=>T2, Q=>Q2T );
24     T3 <= not Q2T and T2;
25     BIT_3 : TFF port map( C=>Clock, CLR=>Clear, T=>T3, Q=>Q3T );
26     Q0 <= Q0T;
27     Q1 <= Q1T;
28     Q2 <= Q2T;
29     Q3 <= Q3T;
30 end Behavioral;

```

รูปที่ 4.23 ไล้ดวงจรนับลงแบบซิงโครนัส 4 บิตที่สร้างจาก Component ในรูปที่ 4.11



รูปที่ 4.24 ผลจำลองการทำงานของวงจรนับลงแบบซิงโครนัส 4 บิตที่ใช้ไล้ดในรูปที่ 4.22 (U = Uninitial)

การเขียนไล้ด VHDL จากผังวงจรนับขึ้นแบบซิงโครนัส 4 บิตที่เราได้อธิบายไปแล้วนั้น ก่อนการเขียนไล้ดเราจะต้องออกแบบผังวงจรนับขึ้นมาก่อน ทำให้เสียเวลาและต้องใช้ทักษะในการออกแบบวงจรอย่างมาก ซึ่งแตกต่างจากการเขียนไล้ดวงจรนับแบบซิงโครนัส 4 บิตในตัวอย่างที่ 2.26 ของบทที่ 2 ที่เป็นการเขียนไล้ดหลากหลายสไตล์และเป็นการออกแบบโดยใช้ความสามารถของภาษา VHDL และซอฟต์แวร์ทูลช่วยในการออกแบบ ซึ่งทางเลือกที่ดีกว่าเพราะสามารถออกแบบได้รวดเร็วและมี

ประสิทธิภาพ ตัวอย่างการเขียนโค้ดของวงจรนับขึ้นแบบซิงโครนัส 4 บิตแสดงดังรูปที่ 4.25 ซึ่งเมื่อทำการสังเคราะห์วงจรโดยใช้ CPLD จะได้ผังวงจรดังรูปที่ 4.26 และผลจำลองการทำงานแสดงดังรูปที่ 4.27

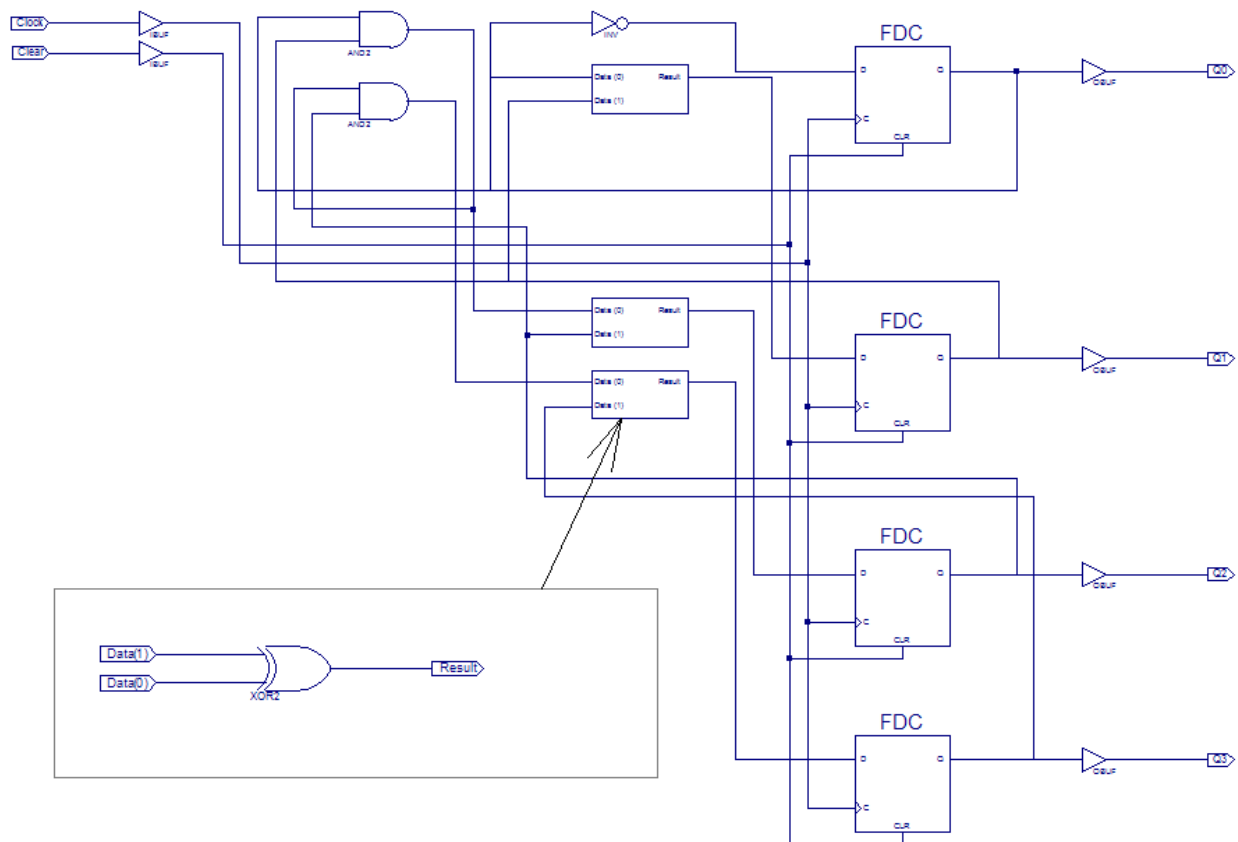
จากรูปที่ 4.26 คนที่มีความรู้เรื่องดิจิทัลมาเป็นอย่างดีก็จะมองออกว่าวงจร D Flip-Flop ที่มีอินเวอร์เตอร์ต่อกลับเข้ามาที่อินพุต D ก็คือ T Flip-Flop ที่ถูกกำหนดให้ $T = '1'$ และ D Flip-Flop ที่มี XOR ต่อกลับเข้ามาที่อินพุต D ก็คือ T Flip-Flop นั่นเอง นั่นก็หมายความว่า Xilinx Synthesis Tool หรือ XST สังเคราะห์วงจรนับขึ้น 4 บิตนี้เป็นวงจรนับแบบซิงโครนัส

```

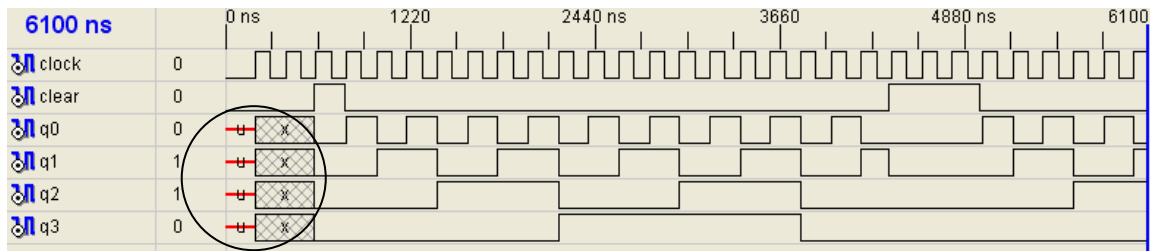
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity COUNTER_4BIT_SYNC_UP is
7      Port ( Clock,Clear : in  STD_LOGIC;
8            Q0,Q1,Q2,Q3 : out STD_LOGIC);
9  end COUNTER_4BIT_SYNC_UP;
10
11  architecture Behavioral of COUNTER_4BIT_SYNC_UP is
12      signal QT : STD_LOGIC_VECTOR (3 downto 0);
13  begin
14      process(Clock,Clear)
15      begin
16          if Clear='1' then QT <= "0000";
17          elsif Clock'event and Clock='1' then QT <= QT + 1;
18          end if;
19      end process;
20      Q0 <= QT(0); Q1 <= QT(1); Q2 <= QT(2); Q3 <= QT(3);
21  end Behavioral;

```

รูปที่ 4.25 โค้ดของวงจรนับขึ้นแบบซิงโครนัส 4 บิต



รูปที่ 4.26 โค้ดวงจรนับขึ้นแบบซิงโครนัส 4 บิตที่ XST สังเคราะห์วงจร (View technology schematic) โดยใช้ CPLD



รูปที่ 4.27 ผลจำลองการทำงานของโค้ดวงจรนับขึ้นแบบซิงโครนัส 4 บิต (U = Uninitial และ 'X' = Unknown)

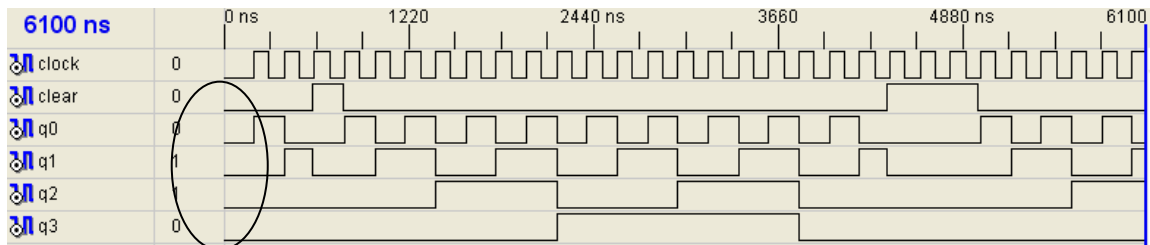
การกำหนดค่าเริ่มต้น (Initial value) ให้รีจิสเตอร์หรือ Flip-Flop ถ้าไม่มีการกำหนดค่าเริ่มต้นนั้น XST จะกำหนด (Default) ค่าเริ่มต้นเป็น '0' แต่ ISE Simulator จะกำหนดค่าเป็น 'U' (U = Uninitial) ดังรูปที่ 4.27 ซึ่งโค้ดวงจรนับในรูปที่ 4.25 นั้นเมื่อกำหนดค่าเริ่มต้นให้กับ Signal QT เป็น "0000" ดังในรูปที่ 4.28 แล้วจะให้ผลจำลองการทำงานแสดงดังรูปที่ 4.29

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity COUNTER_4BIT_SYNC_UP is
7      Port ( Clock,Clear : in  STD_LOGIC;
8            Q0,Q1,Q2,Q3 : out STD_LOGIC);
9  end COUNTER_4BIT_SYNC_UP;
10
11  architecture Behavioral of COUNTER_4BIT_SYNC_UP is
12      signal QT : STD_LOGIC_VECTOR (3 downto 0) := "0000";--Initial value
13  begin
14      process(Clock,Clear)
15      begin
16          if Clear='1' then QT <= "0000";
17          elsif Clock'event and Clock='1' then QT <= QT + 1;
18          end if;
19      end process;
20      Q0 <= QT(0); Q1 <= QT(1); Q2 <= QT(2); Q3 <= QT(3);
21  end Behavioral;

```

รูปที่ 4.28 โค้ดของวงจรนับขึ้นแบบซิงโครนัส 4 บิตที่ได้ค่าเริ่มต้นให้กับ Signal QT เป็น "0000"



รูปที่ 4.29 ผลจำลองการทำงานของวงจรที่กำหนดค่าเริ่มต้นให้กับ Signal QT เป็น "0000"

ตัวอย่างโค้ดวงจรนับขึ้น-นับลง 4 บิตแบบคาสเคดแสดงดังรูปที่ 4.30 เมื่อขา Direction หรือ DIR = '0' และเมื่อวงจรมับขึ้นไปถึงค่าสูงสุดแล้วขา Clock enable output หรือ CEO = '1' ในทำนองเดียวกันเมื่อขา DIR = '1' และวงจรมับลงจนถึงค่าต่ำสุดแล้วขา Clock enable output หรือ CEO = '1' โดยผลจำลองการทำงานแสดงดังรูปที่ 4.31

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity COUNTER_4BIT_UPDN is
7      Port ( C,CE,CLR,DIR : in  STD_LOGIC;
8            CEO : out  STD_LOGIC;
9            Q : out  STD_LOGIC_VECTOR (3 downto 0));
10  end COUNTER_4BIT_UPDN;

```

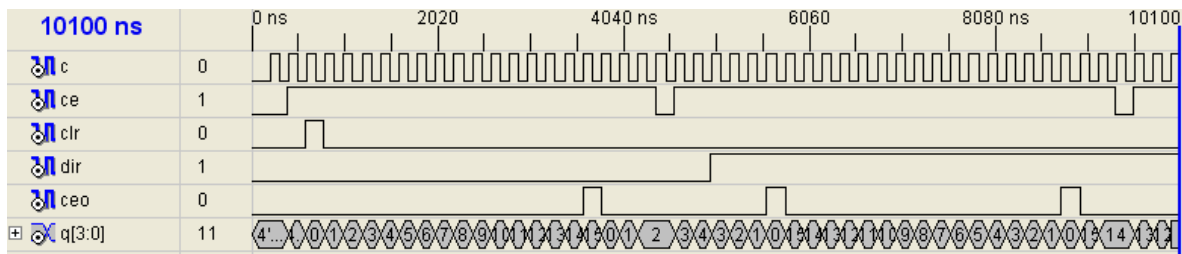
(ต่อ)

```

11
12 architecture Behavioral of COUNTER_4BIT_UPDN is
13     signal QT : STD_LOGIC_VECTOR (3 downto 0);
14 begin
15     -----4Bits Counter-----
16     process (C,CLR)
17     begin
18         if CLR='1' then QT <= "0000";
19     elsif C'event and C='1' then
20         if CE='1' then
21             if DIR='0' then QT <= QT + 1; --Count up
22             else QT <= QT - 1; --Count down
23             end if;
24         end if;
25     end if;
26 end process;
27 Q <= QT;
28     -----CEO-----
29 CEO <= CE when (DIR='0' and QT=15) else
30     CE when (DIR='1' and QT=0) else
31     '0';
32 end Behavioral;

```

รูปที่ 4.30 โค้ดของวงจรนับขึ้น-นับลง 4 บิตแบบคาสเคดเมื่อเรียกใช้ Std_logic_unsigned package



รูปที่ 4.31 ผลจำลองการทำงานของวงจรนับที่ใช้โค้ดในรูปที่ 4.29 หรือรูปที่ 4.30

วงจรนับขึ้น-นับลง 16 บิตที่สร้างจากวงจรนับแบบคาสเคด 4 บิตในรูปที่ 4.30 ที่เป็น Component แสดงดังรูปที่ 4.32 ซึ่งขอให้ผู้อ่านควรสังเกตว่าขาของ Component ที่ไม่ใช้งานให้ใช้คำสั่ง Open (ดูบรรทัดที่ 21 ในรูปที่ 4.32)

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity COUNTER_16BIT_UPDN is
6     Port ( C,CLR,DIR : in STD_LOGIC;
7           Q0,Q1,Q2,Q3 : out STD_LOGIC_VECTOR (3 downto 0) );
8 end COUNTER_16BIT_UPDN;
9
10 architecture Behavioral of COUNTER_16BIT_UPDN is
11     component COUNTER_4BIT_UPDN
12         Port ( C,CE,CLR,DIR : in STD_LOGIC;
13               CEO : out STD_LOGIC;
14               Q : out STD_LOGIC_VECTOR (3 downto 0) );
15     end component;
16     signal CEO : STD_LOGIC_VECTOR (2 downto 0);
17 begin
18     D_0 : COUNTER_4BIT_UPDN port map (C=>C,CE=>'1',CLR=>CLR,DIR=>DIR,CEO=>CEO(0),Q=>Q0);
19     D_1 : COUNTER_4BIT_UPDN port map (C=>C,CE=>CEO(0),CLR=>CLR,DIR=>DIR,CEO=>CEO(1),Q=>Q1);
20     D_2 : COUNTER_4BIT_UPDN port map (C=>C,CE=>CEO(1),CLR=>CLR,DIR=>DIR,CEO=>CEO(2),Q=>Q2);
21     D_3 : COUNTER_4BIT_UPDN port map (C=>C,CE=>CEO(2),CLR=>CLR,DIR=>DIR,CEO=>open,Q=>Q3);
22 end Behavioral;

```

รูปที่ 4.32 โค้ดและวงจรนับขึ้น-นับลงขนาด 16 บิตที่สร้างจากวงจรนับ 4 บิตในรูปที่ 4.30 มาทำเป็น Component