

การทดลองที่ 5: การใช้งาน Testbench

วัตถุประสงค์

1. เพื่อให้นักศึกษาฝึกเขียนภาษา VHDL เบื้องต้น
2. เพื่อศึกษาการใช้งาน Testbench เบื้องต้น
3. เพื่อศึกษาการทดสอบวงจรที่ออกแบบ

หมายเหตุ นักศึกษาที่มีหนังสือ ให้พิมพ์เฉพาะหน้าแรกของเอกสารการทดลองนี้ก็พอ

การทดลอง

1. ให้นักศึกษาทดลองตามเอกสารการทดลองที่นำมา (หนังสือ บทที่ 6 หน้า 411 - 439) ก่อน
2. ทำการเขียน Testbench เพื่อทดสอบการทำงานของวงจรบวกเลข 4 บิต (Lab ที่ 2) โดยกำหนดให้
 - 2.1 รับข้อมูลอินพุตการบวก 4 บิต สองชุดจาก ไฟล์ชื่อ “Input_4_Bit_Adder.txt”
 - 2.2 เขียนข้อมูลเอาต์พุตการบวก ในไฟล์ชื่อ “Output_4_Bit_Adder.txt”
 - 2.3 โดยข้อมูลที่ใช้อป้อนเป็นอินพุตต้องมีไม่น้อยกว่า 10 ชุด
 - 2.4 ข้อมูลเอาต์พุตให้แสดงเป็นตัวเลขของ 7 Segment ดังนี้ ที่ไฟล์เอาต์พุต

```
* * * * *      *   * * * *   * * * *   *   *   * * * *
*   *          *       *       *   *   *   *
*   *          *   * * * *   * * * *   * * * *   * * * *
*   *          *   *       *       *       *
* * * *        *   * * * *   * * * *       *   * * * *

* * * *   * * * *   * * * *   * * * *   * * * *   *
*           *   *   *   *   *   *   *   *   *
* * * *        *   * * * *   * * * *   * * * *   * * * *
*   *          *   *   *       *   *   *   *   *   *
* * * *        *   * * * *   * * * *   *   *   * * * *

* * * *          *   * * * *   * * * *
*           *   *       *
*           * * * *   * * * *   * * * *
*           *   *   *       *
* * * *   * * * *   * * * *   *
```

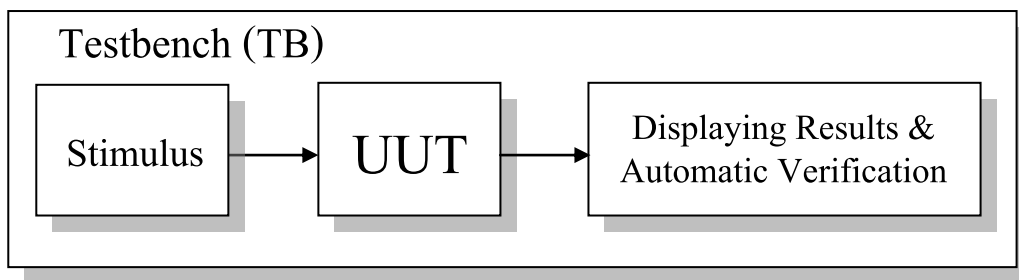
3. ทำการจำลองการทำงาน แล้วเรียกอาจารย์ผู้คุมการทดลองมาตรวจ

รายละเอียดการทดลอง

6.1 Testbench

Testbench เป็นเครื่องมือทางซอฟต์แวร์ที่ใช้ภาษา HDL (VHDL หรือ Verilog) เขียนขึ้นเพื่อใช้ในการตรวจสอบ (Verification) โค้ดของวงจรดิจิทัลที่ออกแบบด้วยการจำลองการทำงาน (Simulation) ต่อไปเราเรียกโค้ดของวงจรหรือระบบดิจิทัลที่ออกแบบว่า Unit under test (UUT) หรือ Design under test (DUT) Testbench จะทำหน้าที่สร้างสัญญาณทดสอบ (Stimulus) หรือ Test vector ได้โดยอิสระเพื่อป้อนให้กับ UUT หรือ DUT ที่อยู่ในรูปของ Component โดยเอาต์พุตที่ได้จะถูกนำไปแสดงผลในรูปแบบต่างๆ เช่น รูปคลื่นสัญญาณ (Waveform) หรือ รายงานข้อผิดพลาด หรือไฟล์ข้อมูล เป็นต้น โดยที่การทำงานทั้งหมดนี้จะกระทำภายใน Testbench เอง ดังนั้น Entity ของ Testbench จึงไม่มี I/O Port ที่ติดต่อกับวงจรรภายนอก

การเขียน Testbench ที่มีความซับซ้อนนั้นเราสามารถนำเอาต์พุตจริง (Actual results) ไปเปรียบเทียบกับเอาต์พุตตามทฤษฎีซึ่งเป็นเอาต์พุตที่คาดการณ์ว่าจะได้ (Expected results) ถ้าหากผลที่ได้ไม่ตรงกันก็แสดงว่ามีข้อผิดพลาดเกิดขึ้น และให้รายงานผลโดยอัตโนมัติ (ทางหน้าต่าง Transcript ที่อยู่ด้านล่างสุดของหน้าต่าง Xilinx-ISE) ซึ่งวิธีนี้จะดีกว่าการสังเกตข้อผิดพลาดจากรูปคลื่นสัญญาณเอาต์พุตที่ได้จากวิธีจำลองการทำงานโดยตรงโดยใช้ Waveform Editor ในการทดลองในบทที่ 3 และบทที่ 4 ซึ่งถ้าเป็นวงจรขนาดใหญ่ก็ยิ่งทำให้เสียเวลามาก บล็อกไดอะแกรมของ Testbench แสดงดังรูปที่ 6.1



รูปที่ 6.1 บล็อกไดอะแกรมอย่างง่ายของ Testbench

คำสั่งเบื้องต้นที่ควรทราบเพิ่มเติมในการเขียน Testbench ได้แก่คำสั่ง wait, wait for และ assert ซึ่งมีความหมายดังนี้

- คำสั่ง wait เป็นคำสั่งให้ Process หยุดทำงาน
- คำสั่ง wait for เป็นคำสั่งให้ Process หยุดรอตามเวลาที่กำหนด เช่น wait for 100 ns;
- คำสั่ง assert เป็นคำสั่งที่ทำงานเมื่อเงื่อนไขบูลีน (Condition) เป็นเท็จ ในการเขียน Testbench นั้นเราจะใช้ร่วมกับคำสั่ง report และ severity เพื่อรายงานระดับความรุนแรงของข้อผิดพลาด โดยมีรูปแบบการเขียนดังนี้

```

assert CONDITION_EXPRESSION
      report TEXT_STRING
      severity SEVERITY_LEVEL;

```

โดยที่ ชนิดข้อมูล SEVERITY_LEVEL จะบอกระดับความรุนแรงดังนี้ NOTE, WARNING, ERROR และ FAILURE

ตัวอย่าง 6.1 โค้ดแอนด์เกต 2 อินพุตแสดงดังรูปที่ 6.2a) มี Testbench แสดงดังรูปที่ 6.2b) โดยผลจำลองการทำงานแสดงดังรูปที่ 6.2c) และรูปที่ 6.2d) โดยรูปที่ 6.2d) นั้นจะไม่มีคำสั่งเริ่มต้นให้ Signal A และ Signal B (บรรทัดที่ 14-15) ที่ป้อนให้อินพุต

```

2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity andgate2 is
6     Port ( A,B : in  STD_LOGIC;
7           Y : out  STD_LOGIC);
8 end andgate2;
9
10 architecture Behavioral of andgate2 is
11 begin
12     Y <= A and B;
13 end Behavioral;

```

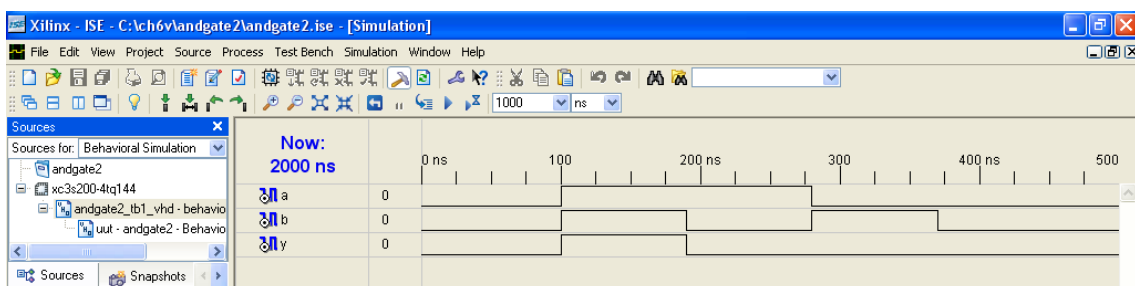
6.2a) โค้ดของวงจรแอนด์เกต 2 อินพุต

```

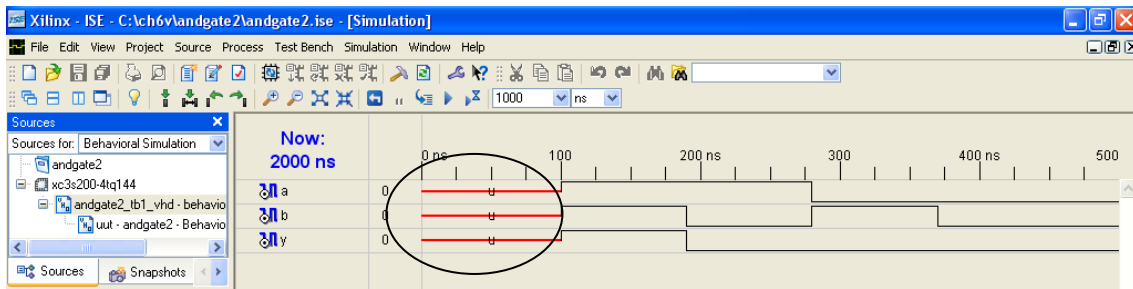
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.ALL;
4
5 ENTITY andgate2_tb1_vhd IS
6 END andgate2_tb1_vhd;
7
8 ARCHITECTURE behavior OF andgate2_tb1_vhd IS
9     COMPONENT andgate2 -- Component Declaration for the UUT
10     PORT(A : IN std_logic;
11          B : IN std_logic;
12          Y : OUT std_logic);
13     END COMPONENT;
14     SIGNAL A : std_logic := '0'; --Input
15     SIGNAL B : std_logic := '0'; --Input
16     SIGNAL Y : std_logic; --Output
17 BEGIN
18
19     uut: andgate2 PORT MAP(A => A,B => B,Y => Y); -- Instantiate the UUT
20
21     tb : PROCESS
22     BEGIN
23         wait for 100 ns;
24         A <= '1';
25         B <= '1';
26         wait for 90 ns;
27         A <= '1';
28         B <= '0';
29         wait for 90 ns;
30         A <= '0';
31         B <= '1';
32         wait for 90 ns;
33         A <= '0';
34         B <= '0';
35         wait; -- will wait forever
36     END PROCESS tb;
37 END;

```

6.2b) โค้ดของ Testbench ที่ใช้ตรวจสอบความถูกต้องของแอนด์เกตแบบ 2 อินพุต



6.2c) Waveform ของเอาต์พุตที่ได้จากผลจำลองการทำงานที่แสดงผลทางหน้าต่างหลัก



6.2d) Waveform ของเอาต์พุตที่ได้จากผลจำลองการทำงาน ซึ่งเราไม่ได้ใส่ค่าเริ่มต้นให้ Signal A และ Signal B

รูปที่ 6.2 โค้ด VHDL และ Testbench ของแอนด์เกต 2 อินพุต

จากบรรทัดที่ 5-6 รูปที่ 6.2b) จะเห็นได้ว่า Testbench จะไม่มี Port เนื่องจากไม่มีส่วนใดที่ติดต่อกับวงจรภายนอก โค้ดบรรทัดที่ 14-16 แสดงการใช้ Signal เชื่อมต่อสัญญาณระหว่างตัวสร้างสัญญาณทดสอบ (Stimulus) กับ UUT ที่เป็น Component โดยที่ Signal นี้จะใช้ชื่อเดียวกับ Port ของ UUT และเนื่องจาก Signal A และ Signal B เพราะเป็นชนิดข้อมูล std_logic (มีลอจิก 9 สถานะ) การไม่ใส่ค่าเริ่มต้น (Initial value) ให้กับ Signal จะทำให้ลอจิกเป็น 'U' (Uninitialized) แสดงในรูปที่ 6.2d)

การสร้างสัญญาณทดสอบหรือ Test vector ในรูปที่ 6.2b) บรรทัดที่ 21-36 จะใช้คำสั่ง wait for โดยมี Time_expression คือ 100 ns และ 90 ns ตามลำดับ ซึ่ง Time_expression นี้จะเป็นชนิดข้อมูล time

จากรูปที่ 6.2a) ในบรรทัดที่ 12 ถ้าเขียนโมเดลของเกตเป็น $Y \leq A \text{ and } B$ after 20 ns; (เกตมีเวลาล่าช้าที่เอาต์พุตหลังจากป้อนอินพุต หรือ Propagation delay time (tp) = 20 ns) และเขียน Testbench ใหม่ที่มีการตรวจสอบข้อผิดพลาดแสดงดังรูปที่ 6.3 ซึ่งหลังจากป้อนอินพุตแล้ว 10 ns (wait for gatedelay;) ถ้าตรวจพบว่าเงื่อนไขบูลีนในคำสั่ง assert เป็นเท็จแล้วให้ใช้คำสั่ง report เพื่อรายงานข้อผิดพลาดและบอกระดับความรุนแรง (Severity) ทางหน้าต่าง Transcript (ด้านล่างสุด) ซึ่งผลที่ได้แสดงดังรูปที่ 6.4 ซึ่งรายงานว่า “at 110.000 ns: Error: Test1 failed!” และ “at 210.000 ns: Error: Test2 failed!”

```

2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4
5  ENTITY andgate2_tb1_vhd IS
6  END andgate2_tb1_vhd;
7
8  ARCHITECTURE behavior OF andgate2_tb1_vhd IS
9      COMPONENT andgate2                -- Component Declaration for UUT
10         PORT (A : IN std_logic;
11              B : IN std_logic;
12              Y : OUT std_logic);
13     END COMPONENT;
14     SIGNAL A : std_logic := '0';--Inputs
15     SIGNAL B : std_logic := '0';--Inputs
16     SIGNAL C : std_logic := '0';--Inputs
17     SIGNAL Y : std_logic;      --Outputs
18 BEGIN
19
20     uut: andgate2 PORT MAP(A => A,B => B,Y => Y);-- Instantiate the UUT
21
22 tb : PROCESS
23     constant gatedelay : time := 10 ns;
24     BEGIN
25         wait for 100ns;
26         A <= '1';
27         B <= '1';
28         wait for gatedelay;
29         assert (Y = '1') report "Test1 failed!" severity ERROR;
30         wait for 90ns;

```

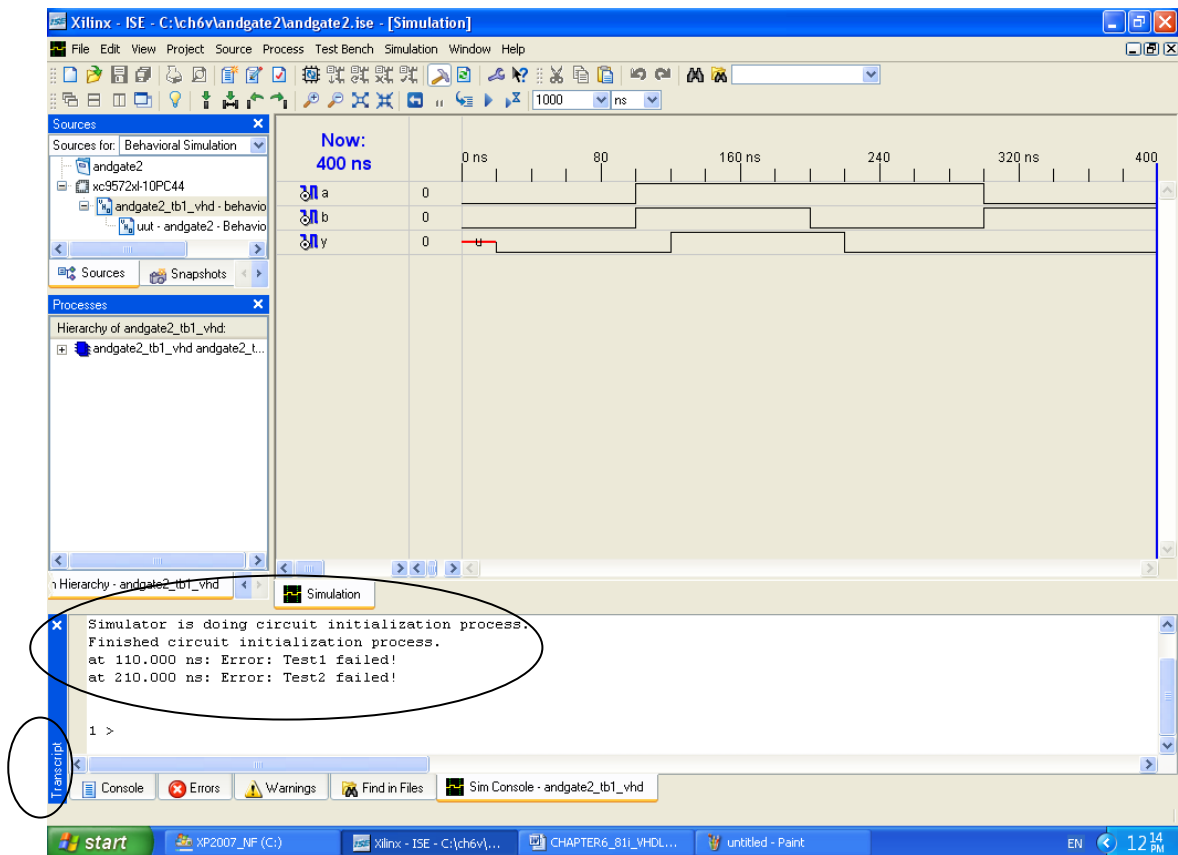
(ต่อ)

```

31     A <= '1';
32     B <= '0';
33     wait for gatedelay;
34     assert (Y = '0') report "Test2 failed!" severity ERROR;
35     wait for 90ns;
36     A <= '0';
37     B <= '1';
38     wait for gatedelay;
39     assert (Y = '0') report "Test3 failed!" severity ERROR;
40     wait for 90ns;
41     A <= '0';
42     B <= '0';
43     wait for gatedelay;
44     assert (Y = '0') report "Test4 failed!" severity ERROR;
45     wait; -- will wait forever
46 END PROCESS tb;
47 END behavior;

```

รูปที่ 6.3 โค้ด Testbench ที่ใช้ตรวจสอบความถูกต้องของแอนด์เกต 2 อินพุตที่มี Propagation delay time ≤ 10 ns



รูปที่ 6.4 รายงานข้อผิดพลาดทางหน้าต่าง Transcript เนื่องจากเกิดมี Propagation delay time มากเกินไป

6.1.1 การสร้าง Testbench

จากตัวอย่างที่ 6.1 เราได้เห็นภาพเกี่ยวกับ Testbench กันบ้างแล้ว ในการเขียน Testbench นั้นเราไม่จำเป็นต้องคำนึงถึง เรื่องการสังเคราะห์วงจร บางคำสั่งที่ใช้อาจจะนำไปสังเคราะห์วงจรไม่ได้ Testbench จะประกอบด้วยส่วนต่างๆ ดังนี้

- Entity declaration และ Architecture เป็นส่วนประกาศใช้ Entity (ที่ไม่ประกาศใช้ Port) และ Architecture body
- Signal declaration เป็นส่วนประกาศใช้ Signal ที่ใช้เชื่อมต่อส่วนที่ใช้สร้างสัญญาณทดสอบ (Stimulus) เข้ากับ Port ของ Component ของ UUT ที่เรานำไปทดสอบ ซึ่ง Signal นี้โดยทั่วไปแล้วจะใช้ชื่อเดียวกับ Port ของ UUT
- Instantiation of top-level design จะเป็นการประกาศใช้และใช้ Component ของ UUT ที่เรานำไปทดสอบ

- Provide stimulus เป็นส่วนที่ใช้สร้างสัญญาณทดสอบหรือ Test vector รวมทั้งสัญญาณ Clock และ Clear เป็นต้น

6.1.2 การสร้างสัญญาณนาฬิกา

การสร้างสัญญาณนาฬิกา (Clock) ซึ่ง Clock มีความจำเป็นสำหรับวงจรซีเควนเชียล ตัวอย่างเช่น

วิธีที่ 1 เช่นถ้า constant Period : TIME := 200 ns; -- Declare a clock period constant

แล้วจะได้ Clock <= not Clock after Period / 2; -- Clock generation

หรือถ้าไม่ประกาศใช้ Constant จะได้

Clock <= not Clock after 100 ns; -- Clock generation : Period = (100+100) ns, F = 5 MHz

วิธีที่ 2 Process

 constant Period : TIME := 200 ns; -- Declare a clock period constant

 begin

 Clock <= '1';

 wait for (Period / 2);

 Clock <= '0';

 wait for (Period / 2);

 end process;

ขอให้ผู้อ่านสังเกตว่าคำสั่ง process ในตัวอย่างของวิธีที่ 2 นี้จะไม่มี Sensitivity list เพราะใช้กับคำสั่ง wait

6.1.3 การสร้างสัญญาณทดสอบ

การสร้างสัญญาณทดสอบ (Stimulus) หรือ Test vector เพื่อป้อนให้กับ Component ของ UUT ตัวอย่างเช่น

Stimulus1 : process

begin

Reset <= '1';

wait for 100 ns;

CE <= '1';

Reset <= '0';

wait for 800 ns;

CE <= '0';

wait for 100 ns;

wait ;

end process Stimulus1;

การสร้างสัญญาณทดสอบหรือ Test vector อาจสร้างได้ด้วยคำสั่งต่างๆ ได้หลายวิธี เช่น

S <= "00",	(เริ่มต้น S = "00")
"01" after 30 ns,	(หลังจากเวลาผ่านไป 30 ns แล้ว S = "01")
"11" after 120 ns,	(หลังจากเวลาผ่านไป 120 ns แล้ว S = "11")
"01" after 150 ns;	(หลังจากเวลาผ่านไป 150 ns แล้ว S = "01")

ซึ่งการสร้างสัญญาณทดสอบในตัวอย่างที่กล่าวมานี้จะคล้ายกับการใช้คำสั่ง wait for แต่จะมีความแตกต่างตรงที่ใช้คำสั่ง wait for เป็นการสร้างสัญญาณต่อเนื่องไปเรื่อยๆ เท่ากับเวลาที่กำหนดในคำสั่ง wait for

นอกจากนี้แล้วเราอาจจะสร้างสัญญาณทดสอบจากข้อมูลอะเรย์ที่ประกาศใช้ Constant ไว้แล้ว หรืออาจจะอ่านจากไฟล์ที่อยู่ภายนอกก็ได้ การเขียน-อ่านไฟล์จากภายนอกสามารถทำได้โดยการเรียกใช้ Package ชื่อ TEXTIO และในกรณีของชนิดข้อมูล std_logic และชนิดข้อมูล std_logic_vector นั้นจะเรียกใช้ Package ชื่อ STD_LOGIC_TEXTIO ของบริษัท Synopsys

ตัวอย่างที่ 6.2 สร้าง Testbench อย่างง่ายของ D Flip-flop แบบ Asynchronous clear โดยที่ C เป็น Clock และ CE เป็นขา Clock enable input โดยมีโค้ดแสดงดังรูปที่ 6.5a) Testbench แสดงดังรูปที่ 6.5b) และเอาต์พุต Waveform แสดงดังรูปที่ 6.5c)

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity DFF_EN is
6      Port ( C,CLR,D,CE : in  STD_LOGIC;
7            Q : out  STD_LOGIC);
8  end DFF_EN;
9
10 architecture Behavioral of DFF_EN is
11     signal QT : STD_LOGIC:='0';
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 CE='1' then QT <= D;
18             end if;
19         end if;
20     end process;
21     Q <= QT;
22 end Behavioral;

```

6.5a) โค้ดของวงจร D Flip-flop แบบ Asynchronous clear

```

2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4
5  ENTITY DFF_EN_TB_vhd IS
6  END DFF_EN_TB_vhd;
7
8  ARCHITECTURE behavior OF DFF_EN_TB_vhd IS
9      -- Component Declaration for the Unit Under Test (UUT)
10     COMPONENT DFF_EN
11     PORT(
12         C : IN std_logic;
13         CLR : IN std_logic;
14         D : IN std_logic;
15         CE : IN std_logic;
16         Q : OUT std_logic
17     );
18     END COMPONENT;

```

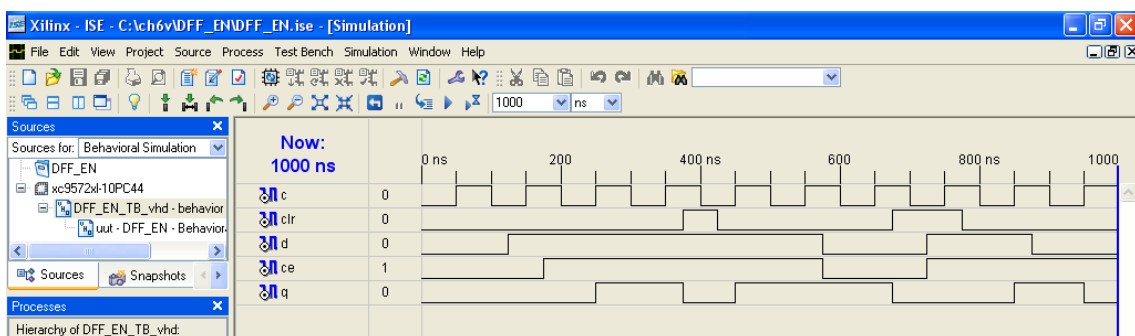
(ต่อ)

```

19  --Inputs
20  SIGNAL C :  std_logic := '0';
21  SIGNAL CLR :  std_logic := '0';
22  SIGNAL D :  std_logic := '0';
23  SIGNAL CE :  std_logic := '0';
24  --Outputs
25  SIGNAL Q :  std_logic;
26 BEGIN
27  -- Instantiate the Unit Under Test (UUT)
28  uut: DFF_EN PORT MAP(
29      C => C,
30      CLR => CLR,
31      D => D,
32      CE => CE,
33      Q => Q
34  );
35  --Generating clock
36  Clock_gen : PROCESS
37      BEGIN
38      C <= '0';
39      wait for 50 ns;
40      C <= '1';
41      wait for 50 ns;
42  END PROCESS Clock_gen;
43
44  --Providing stimulus
45  CLR <= '0', '1' after 375 ns, '0' after 425 ns, '1' after 675 ns, '0' after 775 ns;
46
47  tb : PROCESS
48      BEGIN
49      wait for 100 ns; -- Wait 100 ns for global reset to finish
50      wait for 25 ns;
51      D <= '1';
52      wait for 50 ns;
53      CE <= '1';
54      wait until C'event and C='1';
55      wait for 325 ns;
56      D <= '0';
57      CE <= '0';
58      wait for 150 ns;
59      D <= '1';
60      CE <= '1';
61      wait for 150 ns;
62      D <= '0';
63      wait;-- will wait forever
64  END PROCESS tb;
65 END behavior;

```

6.5b) Testbench ของวงจร D Flip-flop แบบ Asynchronous clear ซึ่งเราสร้างสัญญาณทดลองให้ดูหลายรูปแบบ



6.5c) เอาต์พุต Waveform ของวงจร D Flip-flop แบบ Asynchronous clear

รูปที่ 6.5 โค้ดและ Testbench ของวงจร D Flip-flop แบบ Asynchronous clear

6.1.4 การแสดงผลและตรวจสอบข้อผิดพลาดโดยอัตโนมัติ

การตรวจสอบเพื่อหาข้อผิดพลาดโดยอัตโนมัติเหมาะสำหรับวงจรหรือระบบที่มีขนาดใหญ่ ซึ่งจะช่วยลดการเสียเวลาลงไปได้มากและลดความผิดพลาดที่เกิดขึ้นเนื่องจากคน การตรวจสอบความถูกต้องโดยอัตโนมัติแบบ Self-checking testbench เป็นการตรวจสอบความถูกต้องวิธีหนึ่งที่นิยมใช้กันมาก ซึ่งเราอาจจะนำเอาต์พุตจริง (Actual results) ไปเปรียบเทียบกับเอาต์พุตตามทฤษฎีซึ่งเป็นเอาต์พุตที่คาดการณ์ว่าจะได้ (Expected results) ถ้าหากผลที่ได้ไม่ตรงกันก็แสดงว่ามีข้อผิดพลาดเกิดขึ้น และให้รายงานผลโดยอัตโนมัติ (ทางหน้าต่าง Transcript ที่อยู่ด้านล่างของหน้าต่าง Xilinx-ISE) การสร้าง Self-checking testbench นี้เหมาะอย่างมากสำหรับการออกแบบเชิงซิงโครนัส (Synchronous design) เพราะเอาต์พุตที่ได้จริงและเอาต์พุตที่คาดการณ์ว่าจะได้นั้นสามารถนำไปเปรียบเทียบกันตอนที่เป็นเวลาขอบขาขึ้นหรือขาลงหรืออาจกำหนดทุกๆ หลายคาบของ Clock ก็ได้

ตัวอย่างที่ 6.3 ฝึกการเขียน Testbench ของโค้ด VHDL วงจรวก (แบบไม่คิดเครื่องหมายหรือ Unsigned) 4 บิต โดยในรูปที่ 6.6a) จะเป็นการเขียนโค้ดโดยเรียกใช้ Package ชื่อ std_logic_unsigned ของ Synopsys โดยที่ Testbench ของวงจรวกนี้มีการสร้างสัญญาณทดสอบ (Stimulus) หรือ Test vector จากข้อมูลอะเรย์ที่ประกาศใช้ Constant แสดงดังรูปที่ 6.6b) ซึ่งในตัวอย่างนี้เราจะแสดงการเลือกทดสอบเฉพาะบางค่าเท่านั้น เพื่อประหยัดเวลาในการทดสอบและสร้างไฟล์ข้อมูลทดสอบ โดยรูปที่ 6.6c) นั้นจะแสดงเอาต์พุต Waveform และรายงานผลทางหน้าต่าง Transcript (ด้านล่างสุด) ของหน้าต่าง Xilinx-ISE

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity ADDER4BIT is
7  generic(N : integer :=4);
8  port( A,B : in std_logic_vector(N-1 downto 0);
9        CARRY : out std_logic;
10       SUM : out std_logic_vector(N-1 downto 0));
11 end ADDER4BIT;
12
13 architecture Behavioral of ADDER4BIT is
14     signal C : std_logic_vector(N downto 0);
15 begin
16     C <= ('0' & A) + ('0' & B);
17     SUM <= C(N-1 downto 0) after 5ns;
18     CARRY <= C(N) after 5ns;
19 end Behavioral;

```

6.6a) โค้ด VHDL ของวงจรวก 4 บิต

```

2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4
5  ENTITY ADDER4BIT_TB_vhd IS
6  END ADDER4BIT_TB_vhd;
7
8  ARCHITECTURE behavior OF ADDER4BIT_TB_vhd IS
9  -- Component Declaration for the Unit Under Test (UUT)
10     COMPONENT ADDER4BIT
11     PORT(
12         A : IN std_logic_vector(3 downto 0);
13         B : IN std_logic_vector(3 downto 0);
14         CARRY : OUT std_logic;
15         SUM : OUT std_logic_vector(3 downto 0)
16     );
17     END COMPONENT;
18 --Inputs
19     SIGNAL A : std_logic_vector(3 downto 0) := (others=>'0');
20     SIGNAL B : std_logic_vector(3 downto 0) := (others=>'0');

```

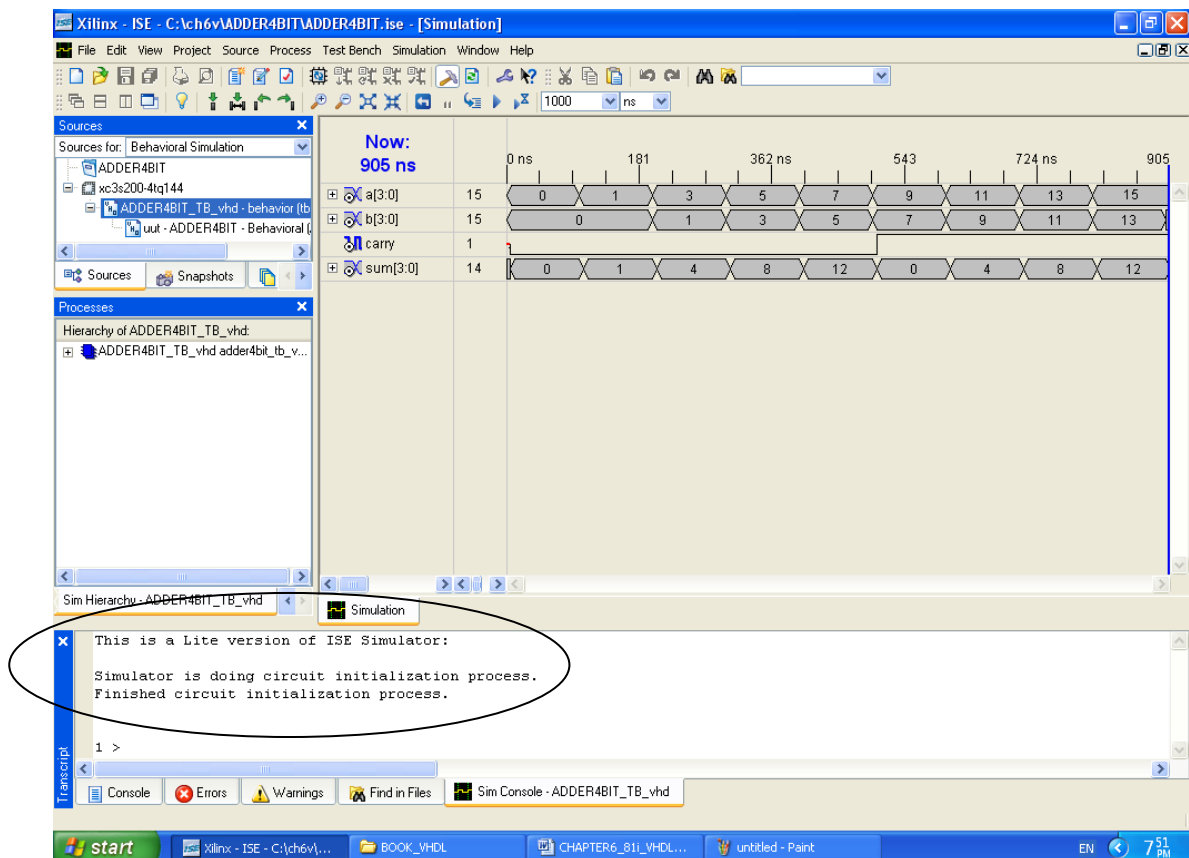
(ต่อ)

```

21 --Outputs
22 SIGNAL CARRY : std_logic;
23 SIGNAL SUM : std_logic_vector(3 downto 0);
24 TYPE adder_array IS ARRAY(0 to 9) of std_logic_vector(3 downto 0);
25 TYPE carry_array IS ARRAY(0 to 9) of std_logic;
26 CONSTANT Ain : adder_array := ("0000","0001","0011","0101","0111",
27 "1001","1011","1101","1111","1111");
28 CONSTANT Bin : adder_array := ("0000","0000","0001","0011","0101",
29 "0111","1001","1011","1101","1111");
30 CONSTANT SUMout : adder_array := ("0000","0001","0100","1000","1100",
31 "0000","0100","1000","1100","1110");
32 CONSTANT CARRYout : carry_array := ('0', '0', '0', '0', '0',
33 '1', '1', '1', '1', '1');
34 BEGIN
35 --Instantiate the Unit Under Test (UUT)
36 uut: ADDER4BIT PORT MAP(A => A,B => B,CARRY => CARRY,SUM => SUM);
37 --Stimulus
38 tb : PROCESS
39 BEGIN
40 FOR i IN adder_array'RANGE LOOP
41 A <= Ain(i);
42 B <= Bin(i);
43 WAIT FOR 10ns;
44 ASSERT SUM = SUMout(i) REPORT "SUM failed!" SEVERITY error;
45 ASSERT CARRY = CARRYout(i) REPORT "CARRY failed!" SEVERITY error;
46 WAIT FOR 90ns;
47 END LOOP;
48 wait; -- will wait forever
49 END PROCESS tb;
50 END behavior;

```

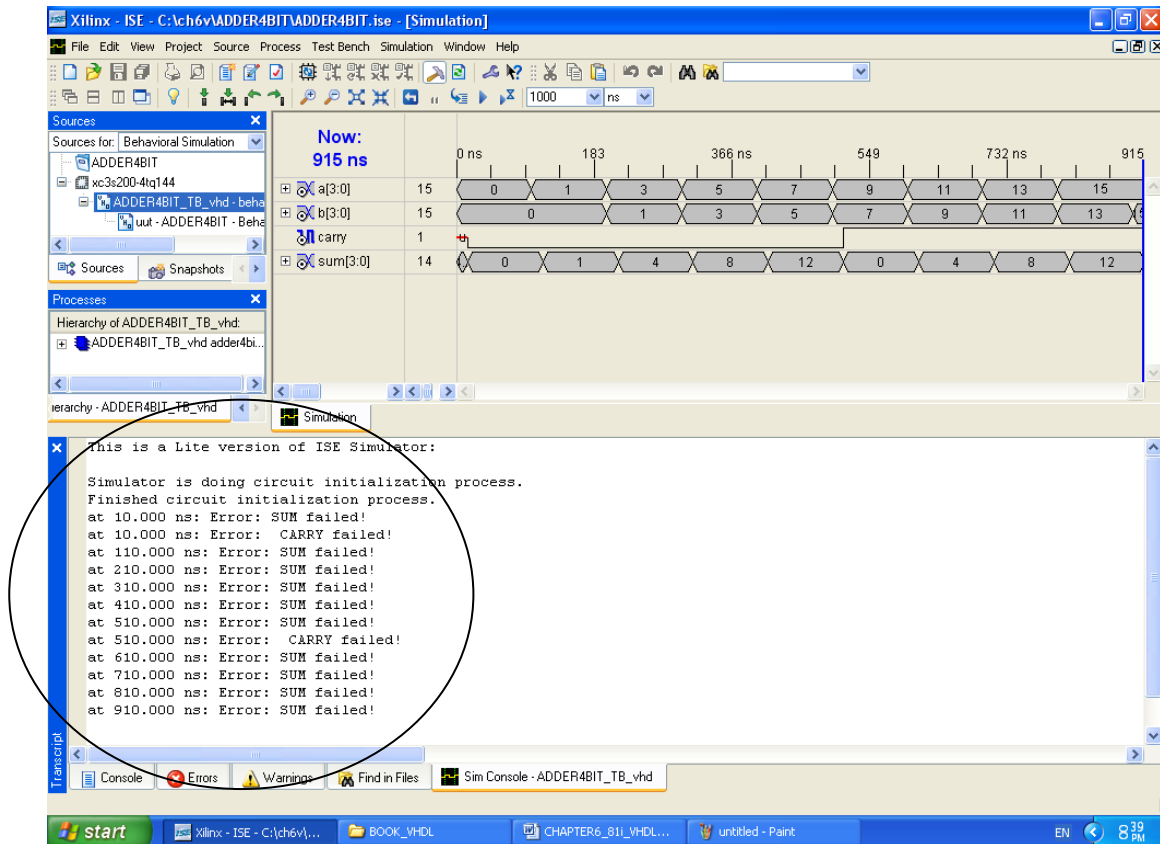
6.6b) Testbench ที่ใช้ตรวจสอบความถูกต้องของวงจรบวก 4 บิต



6.6c) Waveform และรายงานผลทางหน้าต่าง Transcript (ที่อยู่ด้านล่างสุด) ซึ่งไม่มีข้อผิดพลาด

รูปที่ 6.6 โค้ด VHDL ของวงจรบวก 4 บิต Testbench และผลจำลองการทำงาน

โค้ดบรรทัดที่ 17-18 ในรูปที่ 6.6a) จะมีคำว่า “after 5 ns” ซึ่งในการสังเคราะห์วงจรนั้นซอฟต์แวร์ XST (ที่ติดตั้งใน WebPACK) จะละเว้นโดยไม่สนใจ Delay นี้ แต่ ISE Simulator จะตีความว่า Propagation delay time = 5 ns และถ้าเราแก้ไขค่า Propagation delay time จาก “after 5 ns” เสียใหม่เป็น “after 15 ns” แล้วจะได้ผลจำลองการทำงานแสดงดังรูปที่ 6.7



รูปที่ 6.7 Waveform และข้อผิดพลาดที่หน้าต่าง Transcript เนื่องจากวงจรมี Propagation delay time มากเกินไป

การทดลองที่ 6.1.1 Testbench ของบัฟเฟอร์สองทิศทาง


วัตถุประสงค์

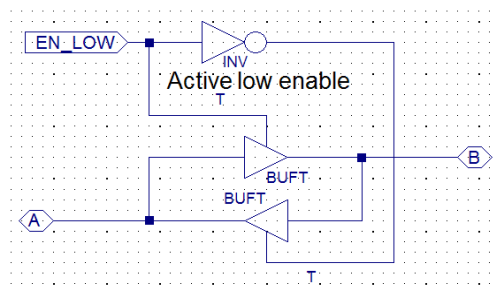
- 1) เพื่อทำความเข้าใจเกี่ยวกับการสร้าง Testbench เพื่อทำ Behavioral Simulation ของบัฟเฟอร์สองทิศทาง
- 2) เพื่อทดลองใช้ซอฟต์แวร์ ISE WebPACK 8.1i สร้างบัฟเฟอร์สองทิศทางโดยใช้ CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างบัฟเฟอร์สองทิศทางด้วย CPLD

1.1) ก่อนเข้าโปรแกรม ISE WebPACK ให้สร้าง Folder ชื่อ ch6v ไว้ในไดรฟ์ C (ถ้าไม่มี) เขียนโค้ดของบัฟเฟอร์สองที่มีฟังก์ชันดังรูปที่ L1.1 ไว้ใน Project Location ชื่อ ch6v แล้วกำหนดให้ Project Name และ Source File ชื่อ ex6_1_1vcx1 โดยโค้ดของบัฟเฟอร์สองแสดงดังรูปที่ L1.2 แล้วคลิก  บันทึกไฟล์และ Check Syntax







รูปที่ L1.1 ฟังก์ชันบัฟเฟอร์สองทิศทาง

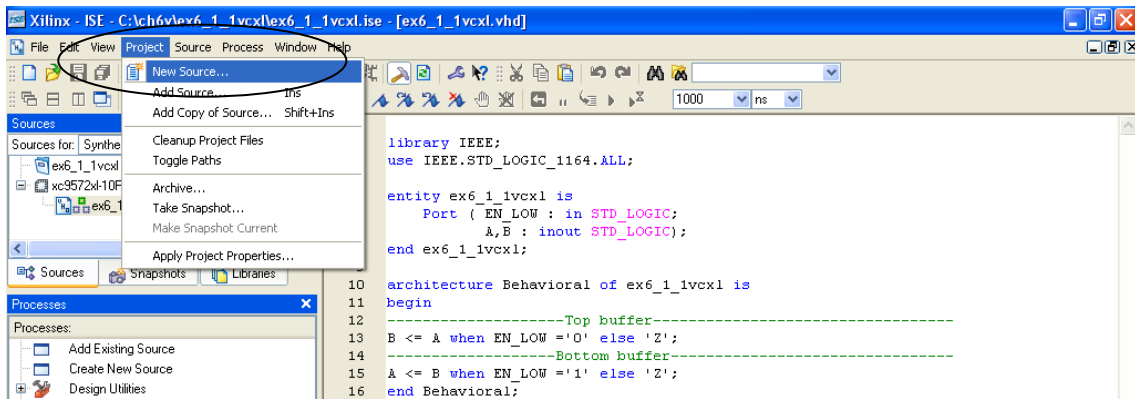
```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity ex6_1_2vcx1 is
6      Port ( EN_LOW : in STD_LOGIC;
7            A,B : inout STD_LOGIC);
8  end ex6_1_2vcx1;
9
10 architecture Behavioral of ex6_1_2vcx1 is
11 begin
12     -----Top buffer-----
13     B <= A when EN_LOW = '0' else 'Z';
14     -----Bottom buffer-----
15     A <= B when EN_LOW = '1' else 'Z';
16 end Behavioral;

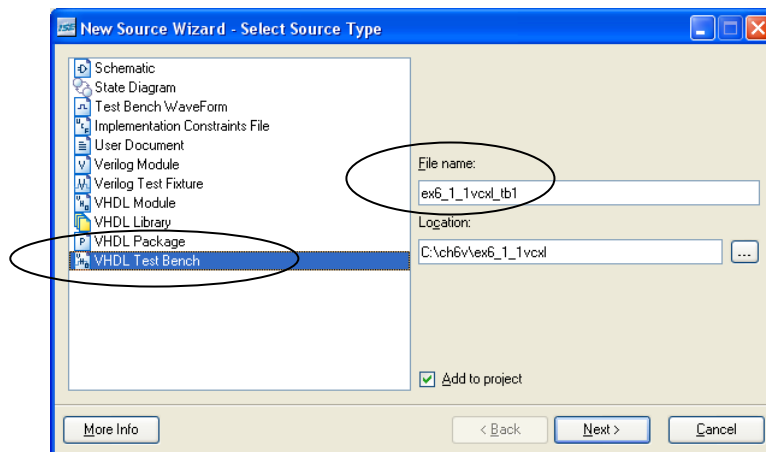
```

รูปที่ L1.2 โค้ด VHDL ของวงจรบัฟเฟอร์สองทิศทาง

1.2) สร้าง Testbench คลิก Project -> New Source ดังรูปที่ L1.3 แล้วพิมพ์ ex6_1_1vcx1_tb1 ในช่อง File name และคลิก VHDL Test Bench ดังรูปที่ L1.4 คลิก Next 2 ครั้ง คลิก Finish แล้วจะได้หน้าต่าง Xilinx-ISE ที่ใช้เขียน Testbench จากนั้นแก้ไขโค้ดเป็นดังรูปที่ L1.5 คลิก  บันทึกไฟล์ คลิก  แล้วคลิกที่ Behavioral Simulation และคลิกชื่อไฟล์ ex6_1_1vcx1_tb1_vhdl ในหน้าต่าง Source คลิก "+" หน้า Xilinx ISE Simulator ในหน้าต่าง Processes จนเป็น "-" แล้วคลิกขวาที่ Check Syntax และคลิก Run ดังในรูปที่ L1.6 ถ้าปรากฏ  ที่หน้า Check Syntax ถือว่าผ่าน แต่ถ้าปรากฏ  ถือว่าไม่ผ่านและให้แก้ไขโค้ด Testbench ใหม่)



รูปที่ L1.3 ขั้นตอนเริ่มสร้าง Source file ของ Testbench



รูปที่ L1.4 ขั้นตอนพิมพ์ชื่อ Source file และคลิกสร้าง Testbench

```

2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4
5  ENTITY ex6_1_1vcx1_tb1_vhd IS
6  END ex6_1_1vcx1_tb1_vhd;
7
8  ARCHITECTURE behavior OF ex6_1_1vcx1_tb1_vhd IS
9      -- Component Declaration for the Unit Under Test (UUT)
10     COMPONENT ex6_1_1vcx1
11     PORT (EN_LOW : IN std_logic;
12           A : INOUT std_logic;
13           B : INOUT std_logic);
14     END COMPONENT;
15     --Inputs
16     SIGNAL EN_LOW : std_logic := '0';
17     --BiDirs
18     SIGNAL A : std_logic;
19     SIGNAL B : std_logic;
20 BEGIN
21     -- Instantiate the Unit Under Test (UUT)
22     uut: ex6_1_1vcx1 PORT MAP (EN_LOW => EN_LOW, A => A, B => B);
23
24     tb : PROCESS
25     BEGIN
26         EN_LOW <= '1'; -- B->A
27         A <= 'Z';      -- Multiple driver : ('Z','1') = '1'
28         B <= '1';      -- B=Input
29         wait for 100 ns;
30         EN_LOW <= '0'; -- A->B
31         A <= '1';      -- A=Input
32         B <= 'Z';      -- Multiple driver : ('Z','1') = '1'
33         wait for 100 ns;

```

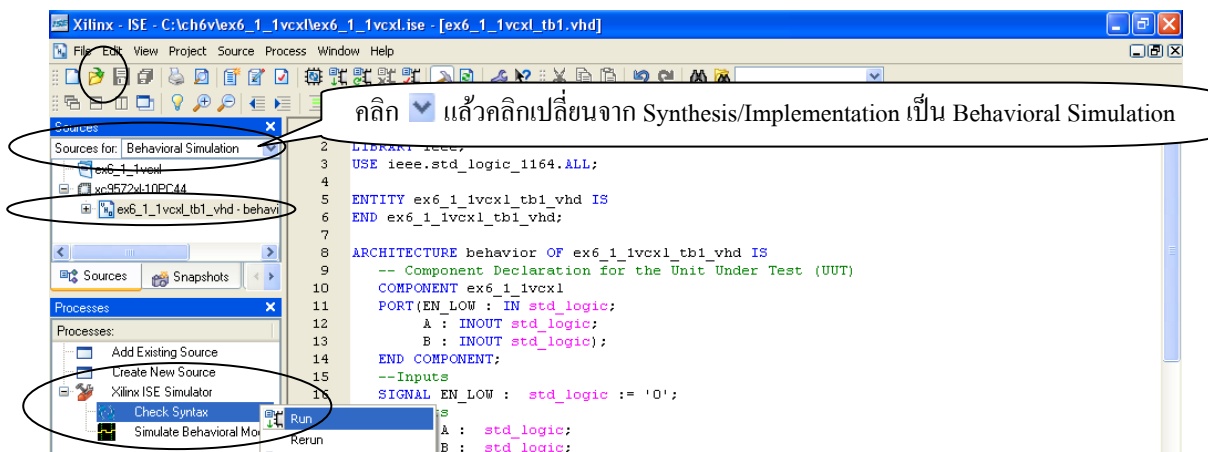
(ต่อ)

```

34      A <= '0';      -- A->B,A=Input
35      B <= 'Z';      -- Multiple driver : ('Z','0') = '0'
36      wait for 100 ns;
37      EN_LOW <= '1'; -- B->A
38      A <= 'Z';      -- Multiple driver : ('Z','0') = '0'
39      B <= '0';      -- B=Input
40      wait for 100 ns;
41      A <= 'Z';      -- Multiple driver : ('Z','Z') = 'Z'
42      B <= 'Z';      -- B->A,B=Input
43      wait for 100 ns;
44      A <= '0';      -- Multiple driver : ('0','1') = 'X'
45      B <= '1';      -- B->A,B=Input
46      wait for 100 ns;
47      A <= '1';      -- Multiple driver : ('1','1') = '1'
48      B <= '1';      -- B->A,B=Input
49      wait for 100 ns;
50      EN_LOW <= '0'; -- A->B
51      A <= '1';      -- A=Input
52      B <= '0';      -- Multiple driver : ('0','1') = 'X'
53      wait for 200 ns;
54      wait; -- will wait forever
55  END PROCESS tb;
56 END behavior;

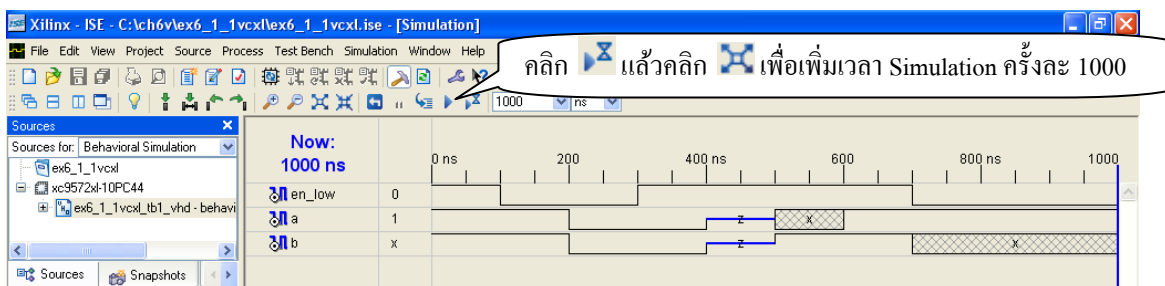
```

รูปที่ L1.5 โค้ดของ Testbench ที่เขียนเสร็จสมบูรณ์แล้ว




รูปที่ L1.6 ขั้นตอนคลิกเปลี่ยนจาก Synthesis/Implementation เป็น Behavioral Simulation

1.3) ทำ Behavioral simulation เลื่อนเมาส์ลงด้านล่างของ Check Syntax ในรูปที่ L1.6 คลิกขวาที่ Simulate Behavioral Model และคลิก Run แล้วจะได้ดังรูปที่ L1.7 ให้พิจารณาผลจำลองการทำงานที่ได้ว่าเป็นไปตามทฤษฎีหรือไม่ จากนั้นคลิก **X** (สีดำ) และคลิก Yes และคลิก **X** (สีดำ) อีกครั้งเพื่อปิด Testbench กลับไปที่หน้าต่าง Xilinx-ISE ที่หน้าต่าง Source ให้คลิก **V** แล้วคลิกเปลี่ยนจาก Behavioral Simulation กลับไปเป็น Synthesis/Implementation ตามเดิม ซึ่งการเขียนโค้ดวงจรย่อยหรือโค้ดสำหรับใช้ทำเป็น Component นั้นเราอาจจะเขียนและตรวจสอบจนถึงขั้นตอนนี้ก็ถือว่าเพียงพอแล้ว แต่อย่างไรก็ตามถ้าเราต้องการนำวงจรนี้ไปดาวน์โหลดลง CPLD ก็ให้ทำขั้นตอน Synthesis, Implementation และ Generate Programming File ต่อไป



รูปที่ L1.7 ผล Behavioral simulation ของวงจรบัพเฟอร์สองทิศทางที่ได้จากการเขียน Testbench

2 สร้างบัพเฟอร์สองทิศทางด้วย FPGA

2.1) ก่อนเข้าโปรแกรม ISE WebPACK ให้สร้าง Folder ชื่อ ch6v ไว้ในไดเรกทอรี C (ถ้าไม่มี) เขียนโค้ดของบัพเฟอร์สองทิศทางที่มีผังวงจรดังรูปที่ L1.1 ไว้ใน Project Location ชื่อ ch6v แล้วกำหนดให้ Project Name และ Source File ชื่อ ex6_1_1vf โดยโค้ดของบัพเฟอร์สองทิศทางดังรูปที่ L2.1 แล้วคลิก  บันทึกไฟล์และ Check Syntax

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity ex6_1_1vf is
6      Port ( EN_LOW : in STD_LOGIC;
7            A,B : inout STD_LOGIC);
8  end ex6_1_1vf;
9
10 architecture Behavioral of ex6_1_1vf is
11 begin
12 -----Top buffer-----
13 B <= A when EN_LOW = '0' else 'Z';
14 -----Bottom buffer-----
15 A <= B when EN_LOW = '1' else 'Z';
16 end Behavioral;

```

รูปที่ L2.1 โค้ด VHDL ของวงจรบัพเฟอร์สองทิศทาง

2.2) การสร้าง Testbench ในกรณีที่เป็น FPGA จะมีขั้นตอนเหมือน CPLD ทุกประการ ตามขั้นตอนที่ได้ทำการทดลองในข้อ 1 โดยที่เราใช้ Source file ชื่อ ex6_1_1vf_tb1 จากนั้นจึงเขียน Testbench เป็นดังรูปที่ L2.2 บันทึกไฟล์และ Check Syntax

```

2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4
5  ENTITY ex6_1_1vf_tb1_vhd IS
6  END ex6_1_1vf_tb1_vhd;
7
8  ARCHITECTURE behavior OF ex6_1_1vf_tb1_vhd IS
9      -- Component Declaration for the Unit Under Test (UUT)
10     COMPONENT ex6_1_1vf
11     PORT(EN_LOW : IN std_logic;
12          A : INOUT std_logic;
13          B : INOUT std_logic);
14     END COMPONENT;
15     --Inputs
16     SIGNAL EN_LOW : std_logic := '0';
17     --BiDirs
18     SIGNAL A : std_logic;
19     SIGNAL B : std_logic;
20 BEGIN
21     -- Instantiate the Unit Under Test (UUT)
22     uut: ex6_1_1vf PORT MAP (EN_LOW => EN_LOW, A => A, B => B);
23
24     tb : PROCESS
25     BEGIN
26         EN_LOW <= '1'; -- B->A
27         A <= 'Z';      -- Multiple driver : ('Z','1') = '1'
28         B <= '1';      -- B=Input
29         wait for 100 ns;
30         EN_LOW <= '0'; -- A->B
31         A <= '1';      -- A=Input
32         B <= 'Z';      -- Multiple driver : ('Z','1') = '1'
33         wait for 100 ns;
34         A <= '0';      -- A->B, A=Input
35         B <= 'Z';      -- Multiple driver : ('Z','0') = '0'
36         wait for 100 ns;

```

(ต่อ)

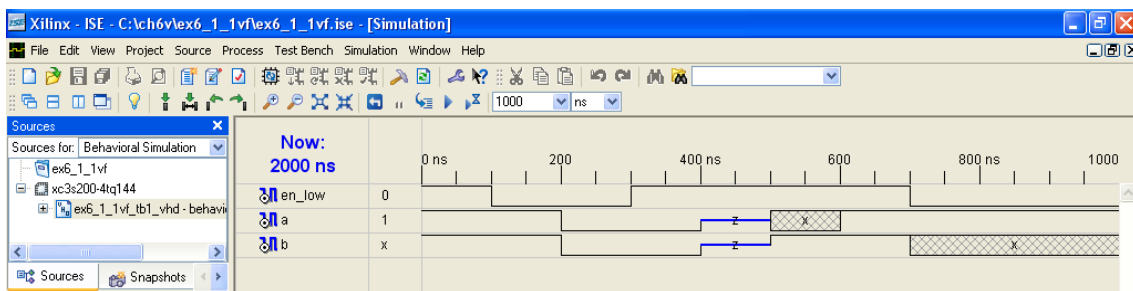
```

37     EN_LOW <= '1'; -- B->A
38     A <= 'Z';      -- Multiple driver : ('Z','0') = '0'
39     B <= '0';      -- B=Input
40     wait for 100 ns;
41     A <= 'Z';      -- Multiple driver : ('Z','Z') = 'Z'
42     B <= 'Z';      -- B->A,B=Input
43     wait for 100 ns;
44     A <= '0';      -- Multiple driver : ('0','1') = 'X'
45     B <= '1';      -- B->A,B=Input
46     wait for 100 ns;
47     A <= '1';      -- Multiple driver : ('1','1') = '1'
48     B <= '1';      -- B->A,B=Input
49     wait for 100 ns;
50     EN_LOW <= '0'; -- A->B
51     A <= '1';      -- A=Input
52     B <= '0';      -- Multiple driver : ('0','1') = 'X'
53     wait for 200 ns;
54     wait; -- will wait forever
55 END PROCESS tb;
56 END behavior;

```

รูปที่ L2.2 โค้ดของ Testbench ที่เขียนเสร็จสมบูรณ์แล้ว

2.3) ทำ Behavioral simulation เสร็จแล้วจะได้ดังรูปที่ L2.3 ให้พิจารณาผลจำลองการทำงานที่ได้ว่าเป็นไปตามทฤษฎีหรือไม่ จากนั้นปิด Testbench กลับไปที่หน้าต่าง Xilinx-ISE ซึ่งการเขียนโค้ดวงจรย่อยหรือโค้ดสำหรับใช้ทำเป็น Component นั้นเราอาจจะเขียนและตรวจสอบจนถึงขั้นตอนนี้ก็ถือว่าเพียงพอแล้ว แต่อย่างไรก็ตามถ้าเราต้องการนำวงจรนี้ไปดาวน์โหลดลง FPGA ก็ให้ทำขั้นตอน Synthesis, Implementation และ Generate Programming File ต่อไป



รูปที่ L2.3 ผล Behavioral simulation ของวงจรบัพเฟอร์สองทิศทางที่ได้จากการเขียน Testbench

หมายเหตุ หลังจากทำ Behavioral simulation แล้วเราสามารถกลับไปทำการสังเคราะห์วงจร ทำการกำหนดค่า CPLD หรือ FPGA ทำการ Implementation และทำการ Generate programming file และดาวน์โหลดตามขั้นตอนปกติ

การทดลองที่ 6.1.2 Testbench ของวงจรตรวจจับลำดับที่เป็น Mealy-type FSM

วัตถุประสงค์

- 1) เพื่อสร้าง Testbench และทำ Behavioral Simulation ของวงจรตรวจจับลำดับ (Sequence detector)
- 2) เพื่อตรวจสอบการเกิด Glitch ที่เอาต์พุต ซึ่งวิธีการ Simulation โดยใช้ Waveform Editor ตรวจไม่พบ
- 3) เพื่อทดลองใช้ซอฟต์แวร์ ISE WebPACK 8.1i สร้างวงจรตรวจจับลำดับโดยใช้ CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรตรวจจับลำดับด้วย CPLD

1.1) ออกแบบวงจรตรวจจับลำดับ (Sequence detector) เป็น Mealy-type FSM ตามตัวอย่างที่ 4.5 ของบทที่ 4 ซึ่งมี X เป็นอินพุต และ Z เป็นเอาต์พุต โดยที่วงจรนี้จะให้ Z = '1' เมื่อตรวจจับลอจิก X = '1' จำนวน 3 ตัวติดต่อกัน จากนั้นให้นำโค้ดวงจรนี้ในรูปที่ 4.72 ของบทที่ 4 มาเขียนใหม่ไว้ใน Project Location ชื่อ ch6v แล้วกำหนดให้ Project Name และ Source File ชื่อ ex6_1_2vcx1 โดยจะแก้ไขชื่อ Entity เป็น ex6_1_2vcx1 โค้ดที่ได้แสดงดังรูปที่ L1.1 ทำการบันทึกไฟล์และ Check Syntax

```

2  ----- Mealy-type FSM.-----
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5
6  entity ex6_1_2vcx1 is
7      Port ( X,Clock,Reset : in  STD_LOGIC;
8            Z : out  STD_LOGIC);
9  end ex6_1_2vcx1;
10
11  architecture Behavioral of ex6_1_2vcx1 is
12      type State_type is (A,B,C);
13      signal State,Next_state : State_type ;
14  begin
15      P1_combination : process (X,State)
16          begin
17              case State is
18                  when A =>
19                      if X = '1' then
20                          Next_state <= B;
21                          Z <= '0';
22                      else
23                          Next_state <= A;
24                          Z <= '0';
25                      end if;
26                  when B =>
27                      if X = '1' then
28                          Next_state <= C;
29                          Z <= '0';
30                      else
31                          Next_state <= A;
32                          Z <= '0';
33                      end if;
34                  when C =>
35                      if X = '1' then
36                          Next_state <= C;
37                          Z <= '1';
38                      else
39                          Next_state <= A;
40                          Z <= '0';

```

(ต่อ)

```

41         end if;
42     end case;
43 end process P1_combination;
44
45 P2_state_resister : process (Clock,Reset)
46 begin
47     if (Reset ='1') then
48         State <= A;
49     elsif (Clock'event and Clock ='1') then
50         State <= Next_state;
51     end if;
52 end process P2_state_resister;
53 end Behavioral;

```

รูปที่ L1.1 ได้ดงจรตรวจจับลำดับที่เป็น Mealy-type FSM

1.2) สร้าง Testbench โดยมีขั้นตอนเหมือนกับการทดลองที่ 6.1.1 ทุกประการ โดยสร้างไว้ใน Source file ชื่อ ex6_1_2vcxl_tb1
เสร็จแล้วจะได้ดังรูปที่ L1.2 บันทึกไฟล์และ Check Syntax โค้ดของ Testbench

```

2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4
5  ENTITY ex6_1_2vcxl_tb1_vhd IS
6  END ex6_1_2vcxl_tb1_vhd;
7
8  ARCHITECTURE behavior OF ex6_1_2vcxl_tb1_vhd IS
9      -- Component Declaration for the Unit Under Test (UUT)
10     COMPONENT ex6_1_2vcxl
11     PORT(X : IN std_logic;
12          Clock : IN std_logic;
13          Reset : IN std_logic;
14          Z : OUT std_logic);
15     END COMPONENT;
16     --Inputs
17     SIGNAL X : std_logic := '0';
18     SIGNAL Clock : std_logic := '0';
19     SIGNAL Reset : std_logic := '0';
20     --Outputs
21     SIGNAL Z : std_logic;
22 BEGIN
23     -- Instantiate the Unit Under Test (UUT)
24     uut: ex6_1_2vcxl PORT MAP(X => X,Clock => Clock,Reset => Reset,Z => Z);
25
26     Clock_gen : PROCESS --Generating clock
27     BEGIN
28         Clock <= '1';
29         wait for 50 ns;
30         Clock <= '0';
31         wait for 50 ns;
32
33     END PROCESS Clock_gen;
34
35     tb : PROCESS          -- Stimulus.
36     BEGIN
37         wait for 100 ns; -- Wait 100 ns for global reset to finish
38         wait for 20 ns;
39         Reset <= '1';
40         wait for 100 ns;
41         Reset <= '0';
42         wait for 100 ns;
43         X <= '1';
44         wait for 100 ns;
45         X <= '0';
46         wait for 200 ns;
47         X <= '1';
48         wait for 200 ns;
49         X <= '0';
50         wait for 200 ns;

```

(ต่อ)

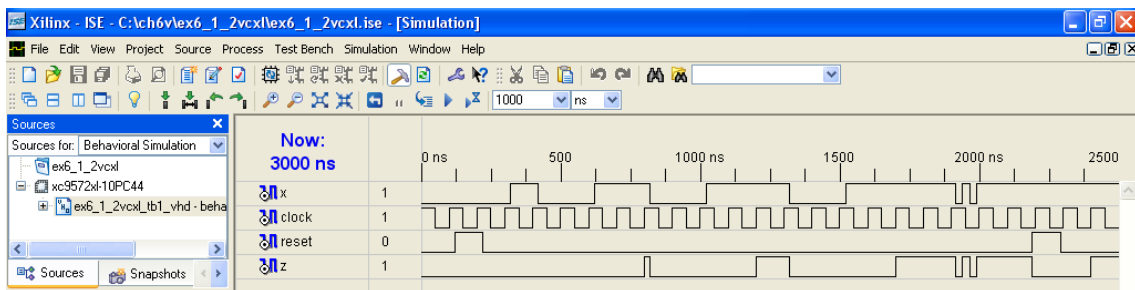
```

51     X <= '1';
52     wait for 300 ns;
53     X <= '0';
54     wait for 200 ns;
55     X <= '1';
56     wait for 395 ns;
57     X <= '0';
58     wait for 25 ns;
59     X <= '1';
60     wait for 25 ns;
61     X <= '0';
62     wait for 25 ns;
63     X <= '1';
64     wait for 200 ns;
65     Reset <= '1';
66     wait for 100 ns;
67     Reset <= '0';
68     wait; -- will wait forever
69 END PROCESS tb;
70 END behavior;

```

รูปที่ L1.2 โค้ด Testbench ของวงจรตรวจจับลำดับที่เป็น Mealy-type FSM

1.3) ทำ Behavioral simulation จะได้ดังรูปที่ L1.3 ให้พิจารณาว่าผลที่ได้ว่าเป็นไปตามทฤษฎีหรือไม่



รูปที่ L1.3 ผล Behavioral simulation ของวงจรตรวจจับลำดับที่เป็น Mealy-type FSM ที่ได้รับการเขียน Testbench

2 สร้างวงจรตรวจจับลำดับด้วย FPGA

2.1) ออกแบบวงจรตรวจจับลำดับ (Sequence detector) เป็น Mealy-type FSM ตามตัวอย่างที่ 4.5 ในบทที่ 4 ซึ่งมี X เป็นอินพุต และ Z เป็นเอาต์พุต โดยที่วงจรนี้จะให้ Z = '1' เมื่อตรวจจับลอจิก X = '1' จำนวน 3 ตัวติดต่อกัน การสร้างวงจรตรวจจับลำดับด้วย FPGA จะมีขั้นตอนเหมือน CPLD ทุกประการ จากนั้นให้นำโค้ดวงจรนี้ในรูปที่ 4.72 ของบทที่ 4 มาเขียนใหม่ไว้ใน Project Location ชื่อ ch6v แล้วกำหนดให้ Project Name และ Source File ชื่อ ex6_1_2vf โดยจะแก้ไขชื่อ Entity เป็น ex6_1_2vf ทำการบันทึกไฟล์และ Check Syntax

2.2) ขั้นตอนสร้าง Testbench จะเหมือน CPLD ทุกประการ โดยใช้ไฟล์ชื่อ ex6_1_1vf_tb1 และเราจะแก้ไขโค้ดในรูปที่ L1.2 โดยจะแก้คำว่า ex6_1_1vcxl เป็น ex6_1_1vf และ ex6_1_1vcxl_tb1 เป็น ex6_1_1vf_tb1

2.3) ขั้นตอนทำ Behavioral simulation จะเหมือน CPLD ทุกประการ แล้วให้พิจารณาผลที่ได้ว่าเป็นไปตามทฤษฎีหรือไม่

6.2 การอ่านและเขียนไฟล์ข้อมูลเบื้องต้น

ภาษา VHDL จะไม่มีคำสั่งแสดงผลโดยตรงแต่จะมีเฉพาะคำสั่งอ่านและเขียนไฟล์จากภายนอกโดยการเรียกใช้ Package ชื่อ TEXTIO (เรียกตาม IEEE Std 1076 ว่า Package TEXTIO) ที่อยู่ในไลบรารีชื่อ Standard ของ IEEE Std 1076 คำสั่งเหล่านี้มีประโยชน์ในการใช้ค่าเริ่มต้น (Initial value) ให้ RAM และใช้สร้างสัญญาณทดสอบหรือ Test vector โดยการอ่านค่าจากไฟล์ที่อยู่ภายนอก โดยที่ VHDL93 จะต่างกับ VHDL87 ตรงที่ไม่มีคำสั่งเปิด-ปิดไฟล์โดยตรง ซึ่งในทางปฏิบัติอาจไม่จำเป็นต้องใช้

การนิยามของชนิดข้อมูลด้วยการประกาศใช้ชนิดข้อมูล (Type declaration) โดยกำหนดล่วงหน้า (Predefined) ไว้แล้วใน Package TEXTIO ของ IEEE Std 1076 การนิยามของชนิดข้อมูลที่สำคัญในกรณีนี้ได้แก่

```
type LINE is access STRING ; -- A LINE is a pointer to a STRING value.
```

```
type TEXT is file of STRING ; -- A file of variable-length ASCII records.
```

โดยที่ access และ file คือ ชนิดข้อมูล access และชนิดข้อมูล file ตามลำดับ

การประกาศใช้ชนิดข้อมูล (Type declaration) ของชนิดข้อมูล File โดยผู้ใช้เอง (User defined) ซึ่งมาตรฐาน IEEE Std 1076 ไม่ได้นิยามไว้ล่วงหน้า (Predefined) นั้นจะมีรูปแบบดังนี้

```
type FILE_TYPE_NAME is file of TYPE_NAME ;
```

โดยที่ TYPE_NAME คือ ชนิดข้อมูล ได้แก่ Scalar type, Record type หรือ Constrained array subtype

การประกาศใช้ขอบเขต (Object declaration) ประเภท File (File declaration) ตาม IEEE Std 1076-1993 และ IEEE Std 1076-1987 นั้นจะมีความแตกต่างกัน แต่ XST รองรับการใช้งานทั้ง 2 รูปแบบ โดยมีรูปแบบเป็นดังนี้

```
file INPUT_OBJECT_FILE : text open READ_MODE is "INPUT_FILE.text" ; -- VHDL93
```

```
file OUTPUT_OBJECT_FILE : text open WRITE_MODE is "OUTPUT_FILE.text" ; -- VHDL93
```

```
file INPUT_OBJECT_FILE : text in is "INPUT_FILE.text" ; -- VHDL87
```

```
file OUTPUT_OBJECT_FILE : text out is "OUTPUT_FILE.text" ; -- VHDL87
```

โดยที่ INPUT_FILE.text หรือ OUTPUT_FILE.text คือ ไฟล์ข้อมูลที่จะเขียนด้วยโปรแกรม Notepad ซึ่งนามสกุลอาจใช้ชื่ออื่นก็ได้ ส่วน INPUT_OBJECT_FILE และ OUTPUT_OBJECT_FILE จะเป็นที่เก็บไฟล์ชั่วคราว

คำสั่งจัดการเกี่ยวไฟล์ข้อมูลจะเขียนใน Procedure เพื่อให้อ่านหรือเขียนเรียงตามลำดับ โดยที่ STD_LOGIC_TEXTIO package (ของบริษัท Synopsys) ที่อยู่ในไลบรารี ieee จะเป็น Overloaded procedure ของ standard TEXTIO คำสั่งที่ใช้อยู่ได้แก่

```
readline (INPUT_OBJECT_FILE, INPUT_LINE) เป็นการสั่งอ่านบรรทัดใหม่จากที่เก็บไฟล์อินพุตชั่วคราว
```

```
read (INPUT_LINE, VALUE) ; -- Reads a new object from the line.
```

```
write (OUTPUT_LINE, VALUE) ; -- Writes a new object into the line.
```

```
writeline (OUTPUT_OBJECT_FILE, OUTPUT_LINE) เป็นการสั่งเขียนบรรทัดใหม่จากที่เก็บไฟล์เอาต์พุตชั่วคราว
```

```
endfile (FILE_NAME) ; -- Returns boolean true if the end of file is reached.
```

การเรียกใช้ Package TEXTIO ที่อยู่ในไลบรารีชื่อ Standard ของ IEEE Std 1076 หรือ STD_LOGIC_TEXTIO package ของบริษัท Synopsys ที่อยู่ในไลบรารี ieee (ซึ่งเป็น Overloaded procedure ของ standard TEXTIO) เป็นตามลำดับดังนี้

```
use STD.TEXTIO.all ; หรือ use IEEE.STD_LOGIC_TEXTIO.all ; ตามลำดับ
```

ตัวอย่างที่ 6.4 ใค้วงจรบวก 4 บิตมี Propagation delay time = 5 ns แสดงดังรูปที่ 6.8a) มี Testbench ที่สร้างสัญญาณทดสอบจากไฟล์ข้อมูลที่อยู่ภายนอกแสดงดังรูปที่ 6.8b) ในตัวอย่างนี้จะแสดงการเลือกทดสอบเฉพาะบางค่าเท่านั้น โดยไฟล์ของอินพุตและ

ไฟล์ของเอาต์พุตที่อยู่ภายนอกนั้นจะเป็นไฟล์ของค่า Integer (เป็นเลขฐาน 10) แสดงดังรูปที่ 6.10c) และรูปที่ 6.10d) ตาม ลำดับ ไฟล์นี้จะเขียนและอ่านด้วยโปรแกรม Notepad และใช้ไฟล์นามสกุล .txt ซึ่งไฟล์ของอินพุตในรูปที่ 6.10c) นั้น คอลัมน์ที่ 1 เป็น A คอลัมน์ที่ 2 เป็น B และคอลัมน์ที่ 3 เป็นผลลัพธ์ที่คาดว่าจะได้ (Expected result) หรือผลลัพธ์ตามทฤษฎี ส่วนในรูปที่ 6.10e) นั้น จะแสดงเอาต์พุต Waveform และรายงานผลทางหน้าต่าง Transcript (ที่อยู่ด้านล่างสุด) ของหน้าต่าง Xilinx-ISE

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity ADDER4BIT_TEXTIO is
7      Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
8            B : in  STD_LOGIC_VECTOR (3 downto 0);
9            C : out STD_LOGIC_VECTOR (4 downto 0));
10 end ADDER4BIT_TEXTIO;
11
12 architecture Behavioral of ADDER4BIT_TEXTIO is
13 begin
14
15     C <= (('0' & A) + ('0' & B)) after 5ns; --Propagation delay time = 5 ns
16
17 end Behavioral;

```

6.8a) โค้ด VHDL ของวงจรบวก 4 บิต

```

2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4  USE ieee.std_logic_arith.all;
5  USE ieee.std_logic_unsigned.all;
6
7  USE std.textio.all;
8
9  ENTITY ADDRE4BIT_TEXTIO_tbi_vhd IS
10 END ADDRE4BIT_TEXTIO_tbi_vhd;
11
12 ARCHITECTURE behavior OF ADDRE4BIT_TEXTIO_tbi_vhd IS
13     -- Component Declaration for the Unit Under Test (UUT)
14     COMPONENT ADDER4BIT_TEXTIO
15     PORT(A : IN std_logic_vector(3 downto 0);
16          B : IN std_logic_vector(3 downto 0);
17          C : OUT std_logic_vector(4 downto 0));
18     END COMPONENT;
19     --Inputs
20     SIGNAL A : std_logic_vector(3 downto 0) := (others=>'0');
21     SIGNAL B : std_logic_vector(3 downto 0) := (others=>'0');
22     --Outputs
23     SIGNAL C : std_logic_vector(4 downto 0);
24 BEGIN
25     -- Instantiate the Unit Under Test (UUT)
26     uut: ADDER4BIT_TEXTIO PORT MAP(A => A,B => B,C => C);
27     -- Stimulus
28     process
29     -- File declaration and variable declaration
30     -- FILE file_ABCin : TEXT IS IN "data_ABC.txt";           --VHDL87
31     FILE file_ABCin : TEXT OPEN READ_MODE IS "data_ABC.txt"; --VHDL93
32     -- FILE file_SUM : TEXT IS OUT "data_SUM.txt";           --VHDL87
33     FILE file_SUM : TEXT OPEN WRITE_MODE IS "data_SUM.txt";  --VHDL93
34     variable line_ABCin : LINE;
35     variable line_SUM : LINE;
36     variable Ain,Bin,Cin,Cout : integer;
37     variable SUM : std_logic_vector(4 downto 0);
38     begin
39         while NOT ENDFILE(file_ABCin) loop
40             READLINE(file_ABCin,line_ABCin); --Get line of input file.

```

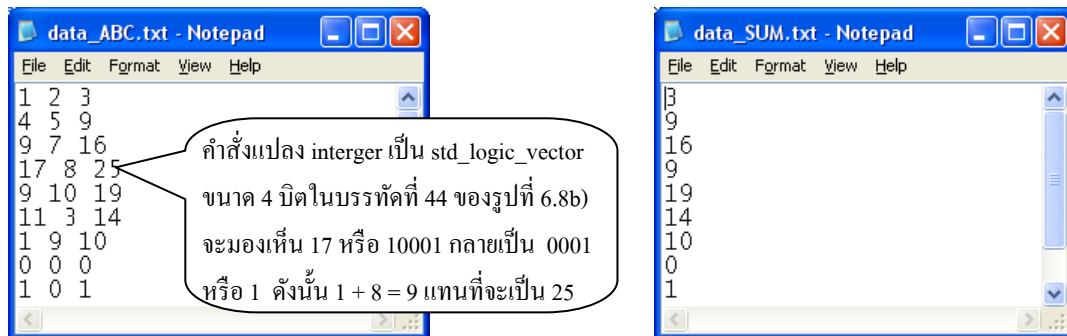
(ต่อ)

```

41      READ(line_ABCin,Ain);           --Get 1st operand.
42      READ(line_ABCin,Bin);           --Get 2nd operand.
43      READ(line_ABCin,Cin);           --Get expected result.
44      A <= conv_std_logic_vector(Ain,4);
45      B <= conv_std_logic_vector(Bin,4);
46      SUM := conv_std_logic_vector(Cin,5);
47      wait for 10 ns;
48      Cout := conv_integer(C);
49      WRITE(line_SUM,Cout);             --Save result to line.
50      WRITELINE(file_SUM,line_SUM);    --Write line to output file.
51      ASSERT C = SUM REPORT "Test failed!" SEVERITY error;
52      wait for 30 ns;
53  end loop;
54  wait; -- will wait forever
55 end process;
56 END behavior;

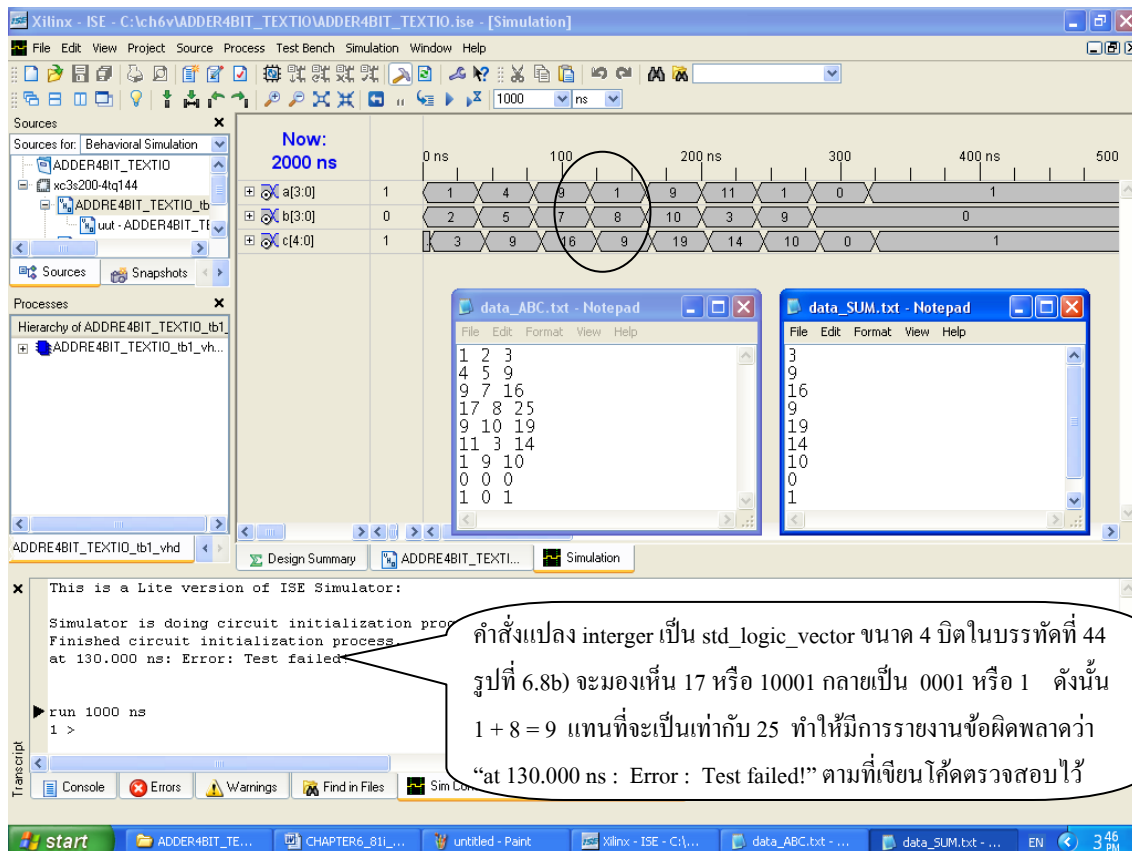
```

6.8b) Testbench ของวงจรบวก 4 บิตที่มีการสร้างสัญญาณทดสอบจากไฟล์ข้อมูลที่อยู่นอก



รูปที่ 6.8c) ไฟล์ integer ของอินพุตชื่อ data_ABC.txt

รูปที่ 6.8d) ไฟล์ integer ของเอาต์พุตชื่อ data_SUM.txt



6.8e) Waveform เอาต์พุตและไฟล์อินพุตและเอาต์พุต (หลังจากป้อนอินพุต 10 ns หรือ wait for 10 ns;) จะให้ผลตรงกัน

รูปที่ 6.8 โค้ดของวงจรบวก 4 บิตและ Testbench รวมทั้งไฟล์ข้อมูลและผลจำลองการทำงาน

จากรูปที่ 6.8b) ในบรรทัดที่ 4 ต้องเรียกใช้ std_logic_arith package เพราะใช้คำสั่ง conv_std_logic_vector ส่วนในบรรทัดที่ 5 ต้องเรียกใช้ std_logic_unsigned package เพราะใช้คำสั่ง conv_integer และในบรรทัดที่ 7 ต้องเรียกใช้ Package TEXTIO เพราะเราใช้คำสั่งเกี่ยวกับการอ่าน-เขียนไฟล์จากภายนอก (ที่เขียนด้วยโปรแกรม Notepad)

จากรูปที่ 6.8b) ในบรรทัดที่ 31 และบรรทัดที่ 33 เป็นตัวอย่างการประกาศใช้ออบเจกต์ประเภทไฟล์ (File declaration) ที่เขียนตาม VHDL93 และส่วนที่เราใส่ Comment ใวนั้นจะเป็นวิธีเขียนตาม VHDL87 ซึ่ง XST จะรองรับการเขียนทั้ง 2 วิธี เนื่องจากการอ่าน-เขียนข้อมูลจะเป็นซีเคิวนะยิล เราจึงไม่สามารถเลือกอ่านค่าข้อมูลตัวใดตัวหนึ่งหรือบรรทัดใดบรรทัดหนึ่งในลักษณะสุ่มได้ ดังนั้นในบรรทัดที่ 40 จึงเป็นการสั่งให้อ่านข้อมูลที่ละบรรทัด ในขณะที่บรรทัดที่ 41-43 เป็นการสั่งให้อ่านค่าของข้อมูลที่ละค่าในบรรทัดนั้น และในทำนองเดียวกันบรรทัดที่ 49 จะเป็นการสั่งให้เขียนข้อมูลลงในบรรทัดและบรรทัดที่ 50 จะเป็นการเขียนข้อมูลบรรทัดนั้นลงไฟล์ และจากบรรทัดที่ 47 เราใช้คำสั่ง wait for 10 ns; เพราะว่าวงจรบวก 4 บิตมีค่า Propagation delay time = 5 ns ดังนั้นหลังจากป้อนอินพุตเราจะต้องรอเป็นเวลา ≥ 5 ns แล้ววงจรบวกจึงจะให้เอาต์พุตค่าใหม่

ในทำนองเดียวกันเราสามารถเขียน Testbench ที่สร้างสัญญาณทดสอบจากไฟล์ชนิดข้อมูล std_logic_vector ได้เช่นกัน ซึ่งสามารถเขียน Testbench ของโค้ดวงจรบวก 4 บิตในรูปที่ 6.8a) ได้ดังรูปที่ 6.9a) โดยที่ไฟล์ของอินพุตที่เป็นไฟล์ของค่าชนิดข้อมูล std_logic_vector แสดงดังรูปที่ 6.9b) และไฟล์ของเอาต์พุตของค่าที่เป็น std_logic_vector แสดงดังรูปที่ 6.9c) ตามลำดับ ส่วนในรูปที่ 6.9d) นั้นจะแสดงเอาต์พุต Waveform และรายงานผลทางหน้าต่าง Transcript ของหน้าต่าง Xilinx-ISE

จากรูปที่ 6.9a) ในบรรทัดที่ 5-6 นั้นต้องเรียกใช้ Package TEXTIO และ Package ชื่อ STD_LOGIC_TEXTIO เพราะเราใช้คำสั่งเกี่ยวกับการอ่าน-เขียนไฟล์จากภายนอกที่เป็นชนิดข้อมูล std_logic_vector แม้ว่า Package ชื่อ STD_LOGIC_TEXTIO จะคอมไพล์ไว้ในไลบรารี ieee ก็ตามแต่ก็เป็น Non-standard package

```

2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4
5  USE std.textio.all;
6  USE ieee.std_logic_textio.all;
7
8  ENTITY ADDRE4BIT_TEXTIO_tb2_vhd IS
9  END ADDRE4BIT_TEXTIO_tb2_vhd;
10
11 ARCHITECTURE behavior OF ADDRE4BIT_TEXTIO_tb2_vhd IS
12     -- Component Declaration for the Unit Under Test (UUT)
13     COMPONENT ADDER4BIT_TEXTIO
14     PORT(A : IN std_logic_vector(3 downto 0);
15          B : IN std_logic_vector(3 downto 0);
16          C : OUT std_logic_vector(4 downto 0));
17     END COMPONENT;
18     --Inputs
19     SIGNAL A : std_logic_vector(3 downto 0) := (others=>'0');
20     SIGNAL B : std_logic_vector(3 downto 0) := (others=>'0');
21     --Outputs
22     SIGNAL C : std_logic_vector(4 downto 0);
23 BEGIN
24     -- Instantiate the Unit Under Test (UUT)
25     uut: ADDER4BIT_TEXTIO PORT MAP(A => A,B => B,C => C);
26     -- Stimulus
27     process
28     -- File declaration and variable declaration
29     -- FILE file_ABCin : TEXT IS IN "data_ABCin.txt";           --VHDL87
30     FILE file_ABCin : TEXT OPEN READ_MODE IS "data_ABCin.txt"; --VHDL93
31     -- FILE file_SUM : TEXT IS OUT "data_SUMout.txt";          --VHDL87
32     FILE file_SUM : TEXT OPEN WRITE_MODE IS "data_SUMout.txt"; --VHDL93
33     variable line_ABCin : LINE;
34     variable line_SUM : LINE;
35     variable Ain,Bin : std_logic_vector(3 downto 0);

```

(ต่อ)

```

36     variable Cin,Cout : std_logic_vector(4 downto 0);
37 begin
38     while NOT ENDFILE(file_ABCin) loop
39         READLINE(file_ABCin,line_ABCin);    --Get line of input file.
40         READ(line_ABCin,Ain);                --Get 1st operand.
41         READ(line_ABCin,Bin);                --Get 2nd operand.
42         READ(line_ABCin,Cin);                --Get expected result.
43         A <= Ain;
44         B <= Bin;
45         wait for 10 ns;
46         WRITE(line_SUM,C);                  --Save result to line.
47         WRITELINE(file_SUM,line_SUM);       --Write line to output file.
48         ASSERT C = Cin REPORT "Test failed!" SEVERITY error;
49         wait for 30 ns;
50     end loop;
51     wait; -- will wait forever
52 end process;
53 END behavior;

```

6.9a) Testbench ของวงจรบวก 4 บิตที่มีการสร้างสัญญาณทดสอบจากไฟล์ชนิดข้อมูล std_logic_vector

```

File Edit Format View Help
0001 0010 00011
0100 0101 01001
1001 0111 10000
1111 1000 10111
1001 1010 10011
1011 0011 01110
0001 1001 01010
0000 0000 00000
0001 0000 00001

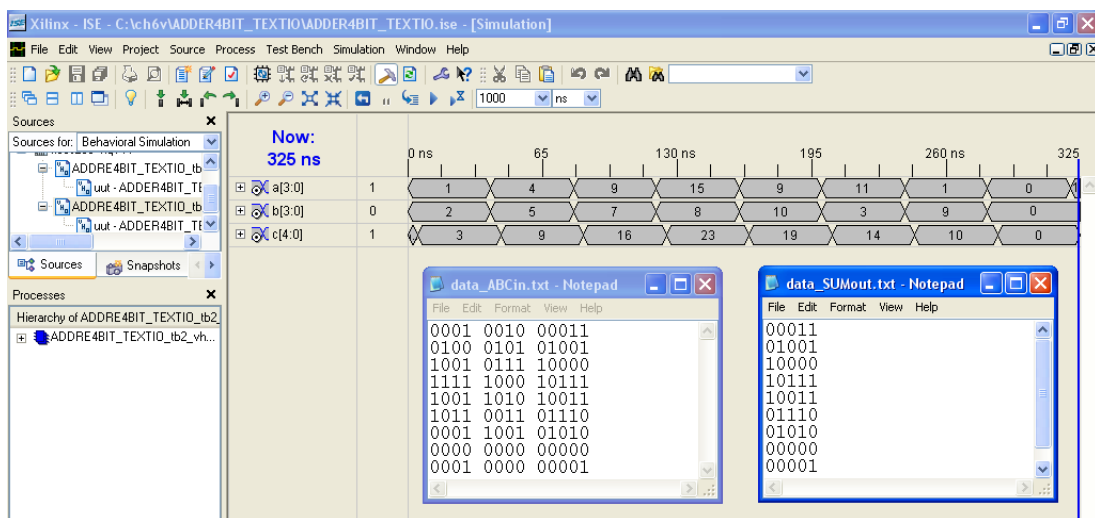
```

```

File Edit Format View Help
00011
01001
10000
10111
10011
01110
01010
00000
00000
00001

```

6.9b) ไฟล์ชนิดข้อมูล std_logic_vector ชื่อ data_ABCin.txt 6.9c) ไฟล์ชนิดข้อมูล std_logic_vector ชื่อ data_SUMout.txt



6.9d) Waveform เอาต์พุตและไฟล์อินพุตและเอาต์พุต (หลังจากป้อนอินพุต 10 ns หรือ wait for 10 ns;) จะให้ผลตรงกัน

รูปที่ 6.9 Testbench รวมทั้งไฟล์ข้อมูลและผลจำลองการทำงานของวงจรบวก 4 บิต

ตัวอย่างที่ 6.5 ฝึกเขียนโค้ดในการใส่ค่าเริ่มต้น (Initial value) ให้กับ RAM โดยเรียกใช้ STD_LOGIC_TEXTIO Package ซึ่งในตัวอย่างนี้จะใส่ค่าเริ่มต้นให้กับ RAM 4 บิตขนาด 8 Word แสดงดังรูปที่ 6.10a) โดยเราให้อ่านค่าเริ่มต้นที่เป็นชนิดข้อมูล std_logic_vector ทุกค่าจากไฟล์ชื่อ RAM_INITIAL_DATA.txt ที่เราเขียนโดยใช้โปรแกรม Notepad แสดงดังรูปที่ 6.10b)

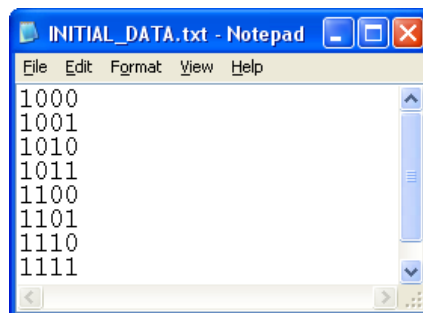
หลังจากใส่ค่าเริ่มต้นให้กับ RAM แล้ว จากนั้นเราจึงจำลองการทำงานของ RAM โดยให้อ่านค่าเอาต์พุตจาก RAM ทุกค่าจะได้ดังรูปที่ 6.10c) ซึ่งจะพบว่าค่าเริ่มต้นที่เขียนไว้ในไฟล์จะตรงกับค่าเอาต์พุตทุกๆ Address


```

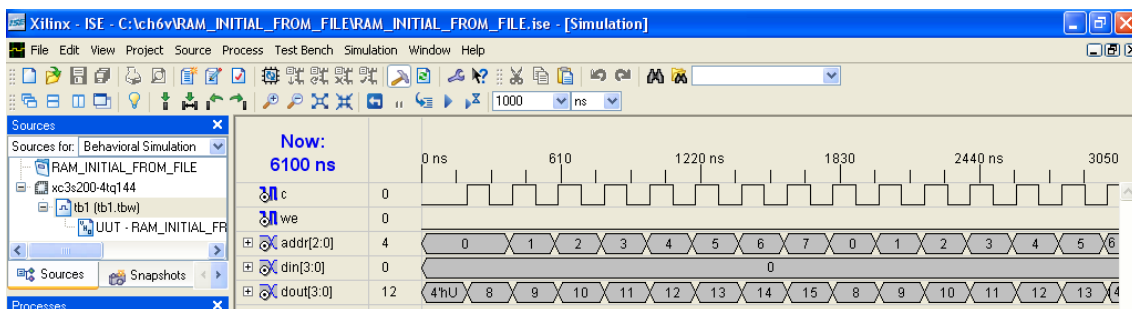
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5  use std.textio.all;
6  use ieee.std_logic_textio.all;
7
8  entity RAM_INITIAL_FROM_FILE is
9      port(C,WE : in std_logic;
10         ADDR : in std_logic_vector(2 downto 0);
11         Din : in std_logic_vector(3 downto 0);
12         Dout : out std_logic_vector(3 downto 0));
13 end RAM_INITIAL_FROM_FILE;
14
15 architecture Behavioral of RAM_INITIAL_FROM_FILE is
16     type RamType is array(0 to 7) of std_logic_vector(3 downto 0);
17     function InitRamFromFile (RamFileName : in string) return RamType is
18         FILE RamFile : text is in RamFileName;
19         variable RamFileLine : line;
20         variable INITIAL_DATA : RamType;
21     begin
22         for I in RamType'range loop
23             readline (RamFile, RamFileLine);
24             read (RamFileLine, INITIAL_DATA(I));
25         end loop;
26         return INITIAL_DATA;
27     end function;
28     signal RAM : RamType := InitRamFromFile("INITIAL_DATA.txt");--Function call
29 begin
30     process (C)
31     begin
32         if C'event and C = '1' then
33             if WE = '1' then
34                 RAM(conv_integer(ADDR)) <= Din;
35             end if;
36             Dout <= RAM(conv_integer(ADDR));
37         end if;
38     end process;
39 end Behavioral;

```

6.10a) โค้ด VHDL ของ RAM ขนาด 4 บิต 8 Word



6.10b) ค่าเริ่มต้นที่เขียนไว้ในไฟล์ชื่อ INITIAL_DATA.txt



6.10c) Waveform เอาต์พุตที่ได้จากการจำลองการทำงานและไฟลีนินพุตจะให้ผลตรงกัน

รูปที่ 6.10 โค้ด VHDL ของ RAM ขนาด 4 บิต 8 Word พร้อมกับใส่ค่าเริ่มต้น

การทดลองที่ 6.2.1 Testbench ของวงจรบวก 4 บิต

วัตถุประสงค์

- 1) เพื่อสร้าง Testbench และทำ Behavioral Simulation ของวงจรบวก 4 บิต
- 2) เพื่อทดลองใช้ซอฟต์แวร์ ISE WebPACK 8.1i สร้างวงจรบวก 4 บิตโดยใช้ CPLD และ FPGA

อุปกรณ์ทดลอง

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

1 สร้างวงจรบวก 4 บิตด้วย CPLD

1.1) ออกแบบวงจรบวก 4 บิต ในรูปที่ 6.8a) ของตัวอย่างที่ 6.3 แล้วนำโค้ดวงจรนี้ในรูปที่ 6.8a) ไปเขียนใหม่ไว้ใน Project Location ชื่อ ch6v แล้วกำหนดให้ Project Name และ Source File ชื่อ ex6_2_1vcx1 โดยแก้ไขชื่อ Entity เป็น ex6_2_1vcx1 แสดงดังรูปที่ L1.1 ทำการบันทึกไฟล์และ Check Syntax จากนั้นนำโค้ด Testbench ในรูปที่ 6.8b) ไปเขียนใหม่ไว้ใน Source File ชื่อ ex6_2_1vcx1_tb1 และแก้ไขชื่อ Entity เป็น ex6_2_1vcx1_tb1 แสดงดังรูปที่ L1.2 ทำการบันทึกไฟล์และ Check Syntax

```

2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity ex6_2_1vcx1 is
7      Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
8            B : in  STD_LOGIC_VECTOR (3 downto 0);
9            C : out STD_LOGIC_VECTOR (4 downto 0));
10 end ex6_2_1vcx1;
11
12 architecture Behavioral of ex6_2_1vcx1 is
13 begin
14
15     C <= (('0' & A) + ('0' & B));
16
17 end Behavioral;

```

รูปที่ L1.1 โค้ด VHDL ของวงจรบวก 4 บิต (คำว่า after 5 ns ไม่ต้องเขียนเพราะ XST จะละเว้น (Ignored))

```

2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4  USE ieee.std_logic_arith.all;
5  USE ieee.std_logic_unsigned.all;
6
7  USE std.textio.all;
8
9  ENTITY ex6_2_1vcx1_tb1_vhd IS
10 END ex6_2_1vcx1_tb1_vhd;
11
12 ARCHITECTURE behavior OF ex6_2_1vcx1_tb1_vhd IS
13     -- Component Declaration for the Unit Under Test (UUT)
14     COMPONENT ex6_2_1vcx1
15     PORT(A : IN std_logic_vector(3 downto 0);
16          B : IN std_logic_vector(3 downto 0);
17          C : OUT std_logic_vector(4 downto 0));
18     END COMPONENT;

```

(ต่อ)

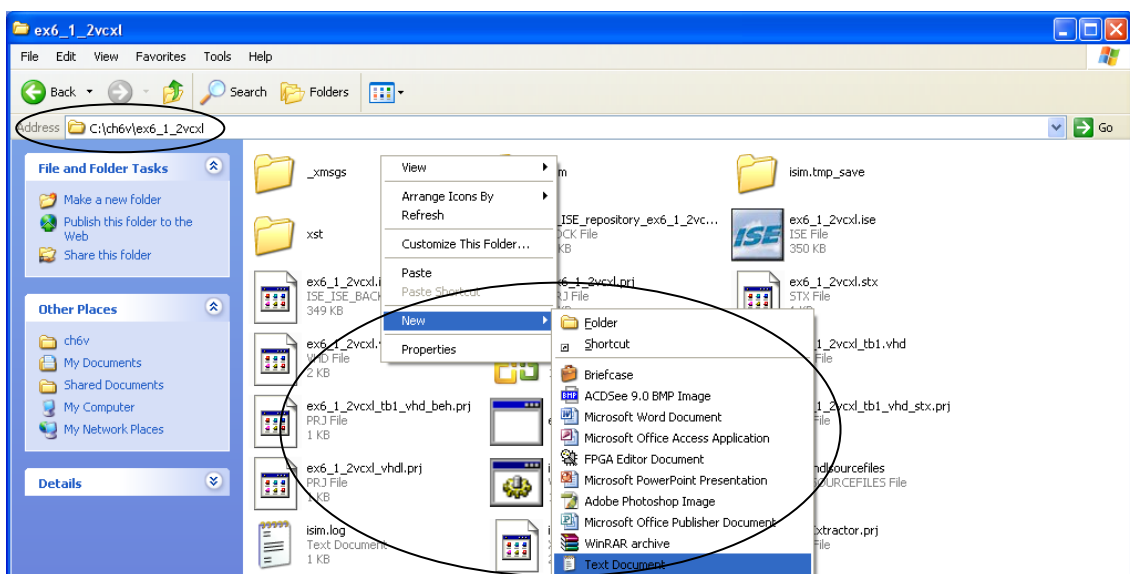
```

19  --Inputs
20  SIGNAL A :  std_logic_vector(3 downto 0) := (others=>'0');
21  SIGNAL B :  std_logic_vector(3 downto 0) := (others=>'0');
22  --Outputs
23  SIGNAL C :  std_logic_vector(4 downto 0);
24  BEGIN
25  -- Instantiate the Unit Under Test (UUT)
26  uut: ex6_2_1vcxl PORT MAP(A => A,B => B,C => C);
27  -- Stimulus
28  process
29  -- File declaration and variable declaration
30  -- FILE file_ABCin : TEXT IS IN "data_ABC.txt";           --VHDL87
31  FILE file_ABCin : TEXT OPEN READ_MODE IS "data_ABC.txt"; --VHDL93
32  -- FILE file_SUM : TEXT IS OUT "data_SUM.txt";           --VHDL87
33  FILE file_SUM : TEXT OPEN WRITE_MODE IS "data_SUM.txt";  --VHDL93
34  variable line_ABCin : LINE;
35  variable line_SUM : LINE;
36  variable Ain,Bin,Cin,Cout : integer;
37  variable SUM : std_logic_vector(4 downto 0);
38  begin
39      while NOT ENDFILE(file_ABCin) loop
40          READLINE(file_ABCin,line_ABCin); --Get line of input file.
41          READ(line_ABCin,Ain);            --Get 1st operand.
42          READ(line_ABCin,Bin);            --Get 2nd operand.
43          READ(line_ABCin,Cin);            --Get expected result.
44          A <= conv_std_logic_vector(Ain,4);
45          B <= conv_std_logic_vector(Bin,4);
46          SUM := conv_std_logic_vector(Cin,5);
47          wait for 10 ns;                  --Wait the propagation delay time.
48          Cout := conv_integer(C);
49          WRITE(line_SUM,Cout);            --Save result to line.
50          WRITELINE(file_SUM,line_SUM);    --Write line to output file.
51          ASSERT C = SUM REPORT "Test failed!" SEVERITY error;
52          wait for 30 ns;
53      end loop;
54      wait; -- will wait forever
55  end process;
56  END behavior;

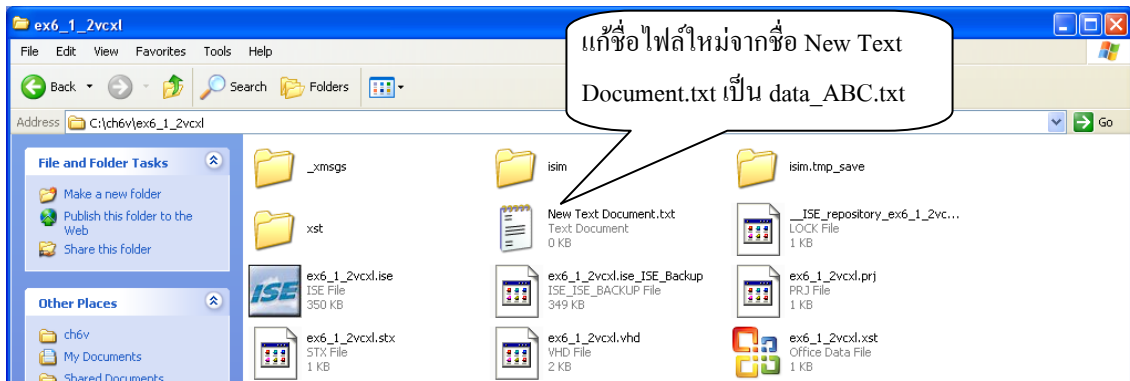
```

รูปที่ L1.2 Testbench ของวงจรบวก 4 บิตที่มีการสร้างสัญญาณทดสอบจากไฟล์ข้อมูลที่อยู่ภายนอก

1.2) การสร้างไฟล์อินพุตและไฟล์เอาต์พุตที่อยู่ภายนอกจะใช้โปรแกรม Notepad โดยเข้าไปที่ Folder ชื่อ C:\ch6v\ex6_2_1vcxl จากนั้นเลื่อนเมาส์ไปที่พื้นที่ว่าง คลิกขวาแล้วเลื่อนเมาส์ไปที่ New -> Text Document ดังรูปที่ L1.3 แล้วคลิกที่ Text Document ดังรูปที่ L1.4 แล้วคลิกแก้ไขชื่อไฟล์จาก New Text Document.txt เป็นชื่อ data_ABC.txt

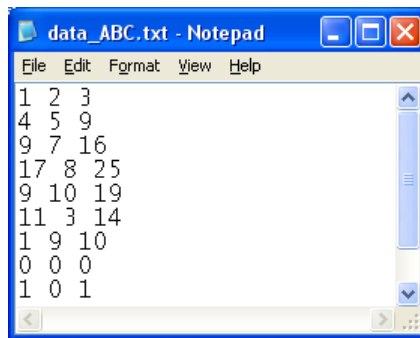


รูปที่ L1.3 ขั้นตอนการสร้างไฟล์อินพุตและไฟล์เอาต์พุตด้วยโปรแกรม Notepad

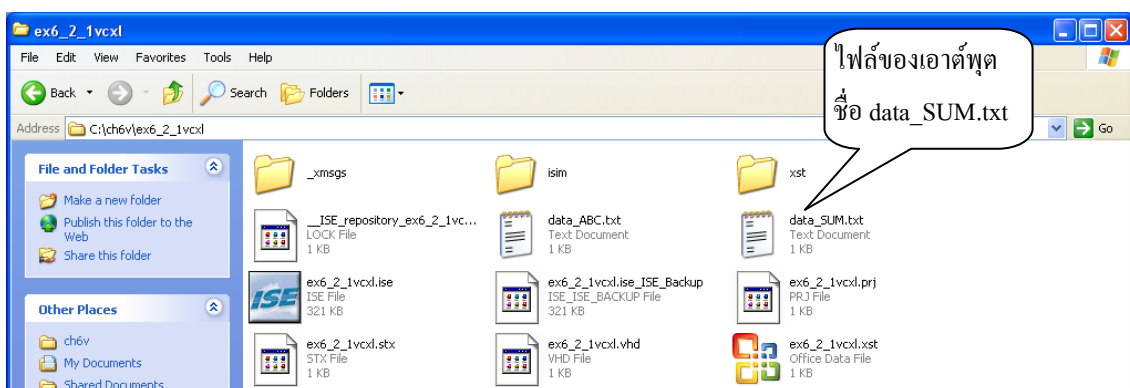


รูปที่ L1.4 ไฟล์ที่สร้างใหม่ชื่อ New Text Document.txt

หลังจากที่แก้ไขไฟล์จากชื่อ New Text Document.txt เป็น data_ABC.txt เสร็จแล้วให้ดับเบิลคลิกที่ชื่อ data_ABC.txt เพื่อเปิดไฟล์และพิมพ์ค่าของอินพุตดังรูปที่ L1.5 แล้วบันทึกไฟล์โดยคลิก File -> Save แล้วปิดโปรแกรม Notepad จากนั้นสร้างไฟล์เอาต์พุตชื่อ data_SUM.txt แสดงดังในรูปที่ L1.6 ซึ่งใช้เป็นไฟล์สำหรับรับค่าผลลัพธ์จากการบวก

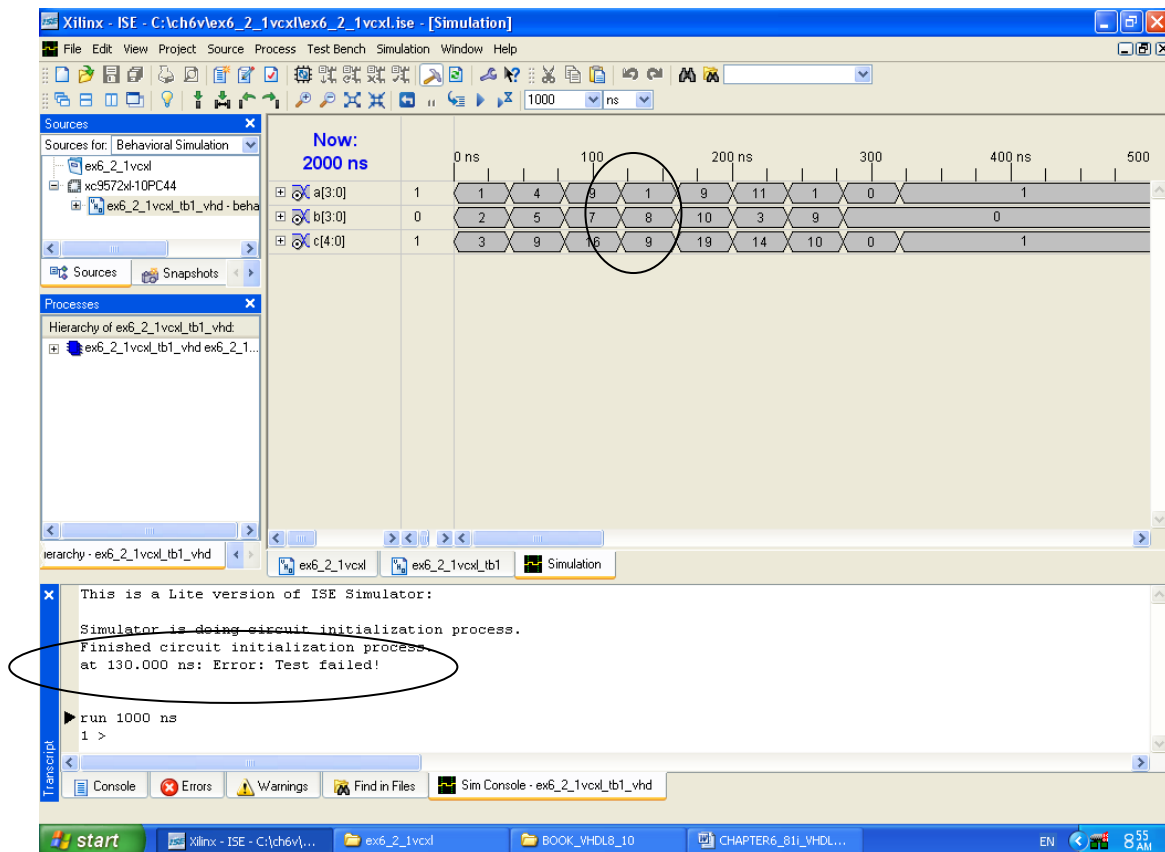


รูปที่ L1.5 ไฟล์ค่า integer ของอินพุตชื่อ data_ABC.txt

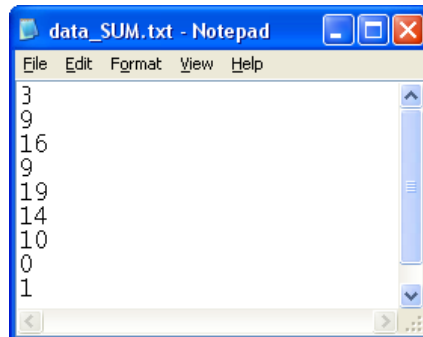


รูปที่ L1.6 ไฟล์ที่สร้างใหม่ชื่อ data_SUM.txt ซึ่งเป็นไฟล์สำหรับรับค่าผลลัพธ์จากการบวก

1.3) ทำ Behavioral Simulation เสร็จแล้วจะได้ผลจำลองการทำงานดังรูปที่ L1.7 ซึ่งในหน้าต่าง Transcript นั้นจะรายงานความผิดพลาดว่า “at 130.000 ns : Error : Test failed!” และให้พิจารณาว่าผลจำลองการทำงานที่ทางหน้าต่างหลักและหน้าต่าง Transcript เป็นตามทฤษฎีหรือไม่ จากนั้นให้เข้าไปดูผลลัพธ์ที่ไฟล์เอาต์พุตโดยเข้าไปที่ Folder ชื่อ C:\ch6v\ex6_2_1vcxl และดับเบิลคลิกที่ไฟล์ชื่อ data_SUM.txt แล้วจะได้ดังรูปที่ L1.8 และให้พิจารณาว่าผลจำลองการทำงานที่ปรากฏในไฟล์เป็นตามทฤษฎีหรือไม่



รูปที่ L1.7 Waveform ของเอาต์พุต (หลังจากป้อนอินพุต 10 ns หรือ wait for 10 ns;)



รูปที่ L1.8 ไฟล์ integer ของเอาต์พุตชื่อ data_SUM.txt

1.4) ทำขั้นตอน Synthesis เสร็จแล้วทำ Design Implementation และ Generate Programming File ตามลำดับ โดยที่ในขั้นตอน Design Implementation นั้นให้กำหนดขาสัญญาณต่างๆ ดังนี้

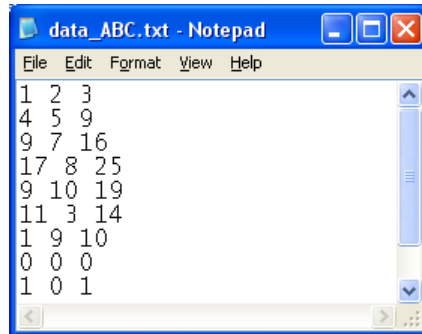
A(3) = PB1 = INPUT = p39	B(3) = PB5 = INPUT = p44	C(4) = MN4 = OUTPUT = p3
A(2) = PB2 = INPUT = p40	B(2) = PB6 = INPUT = p1	C(3) = LED1 = OUTPUT = p38
A(1) = PB3 = INPUT = p42	B(1) = I/O6 of K1 = INPUT = p7	C(2) = LED2 = OUTPUT = p37
A(0) = PB4 = INPUT = p43	B(0) = I/O7 of K1 = INPUT = p6	C(1) = LED3 = OUTPUT = p36
		C(0) = LED4 = OUTPUT = p35

หลังจากโปรแกรมวงจรลง CPLD แล้วให้ทำการทดลองว่าได้ผลบวกเป็นไปตามทฤษฎีหรือไม่

2 สร้างวงจรบวก 4 บิตด้วย FPGA

ออกแบบวงจรบวก 4 บิต ในรูปที่ 6.8a) ของตัวอย่างที่ 6.4 จะมีขั้นตอนต่างๆ เหมือนกับการออกแบบด้วย CPLD ทุกประการที่ได้ทำการทดลองไปแล้วในข้อ 1 โดยนำโค้ดวงจรในรูปที่ L1.1 ไปเขียนใหม่ไว้ใน Project Location ชื่อ ch6v แล้วกำหนดให้ Project Name และ Source File ชื่อ ex6_2_1vf โดยแก้ไขชื่อ Entity เป็น ex6_2_1vf ส่วนโค้ด Testbench ในรูปที่ L1.2 ไปเขียนใหม่ไว้ใน Source File ชื่อ ex6_2_1vf_tb1 และแก้ไขชื่อ Entity เป็น ex6_2_1vf_tb1

การสร้างไฟล์อินพุตที่อยู่ภายนอกจะใช้โปรแกรม Notepad โดยเข้าไปที่ Folder ชื่อ C:\ch6v\ex6_2_1vf จากนั้นสร้างไฟล์ใหม่ชื่อ data_ABC.txt เพื่อใช้สำหรับเก็บข้อมูลอินพุตและพิมพ์ค่าของอินพุตดังรูปที่ L2.1 แล้วบันทึกไฟล์และปิดโปรแกรม Notepad จากนั้นสร้างไฟล์เอาต์พุตชื่อ data_SUM.txt เพื่อใช้เป็นไฟล์สำหรับรับค่าผลลัพธ์จากการบวก



รูปที่ L2.1 ไฟล์ค่า integer ของอินพุตชื่อ data_ABC.txt

ทำ Behavioral Simulation เสร็จแล้วให้พิจารณาว่าผลจำลองการทำงานที่ทางหน้าต่างหลักและหน้าต่าง Transcript เป็นตามทฤษฎีหรือไม่ จากนั้นเข้าไปดูผลลัพธ์ที่ไฟล์เอาต์พุตชื่อ data_SUM.txt และให้พิจารณาว่าผลจำลองการทำงานที่ปรากฏในไฟล์เป็นตามทฤษฎีหรือไม่

จากนั้นทำขั้นตอน Synthesis เสร็จแล้วทำ Design Implementation และ Generate Programming File ตามลำดับ โดยที่ในขั้นตอน Design Implementation นั้นให้กำหนดขาสัญญาณต่างๆ ดังนี้

A(3) = Dip SW1 = INPUT = p52	B(3) = Dip SW5 = INPUT = p59	C(4) = L4 = OUTPUT = p74
A(2) = Dip SW2 = INPUT = p53	B(2) = Dip SW6 = INPUT = p60	C(3) = L3 = OUTPUT = p76
A(1) = Dip SW3 = INPUT = p55	B(1) = Dip SW7 = INPUT = p63	C(2) = L2 = OUTPUT = p69
A(0) = Dip SW4 = INPUT = p56	B(0) = Dip SW8 = INPUT = p68	C(1) = L1 = OUTPUT = p77
		C(0) = L0 = OUTPUT = p70

หลังจากโปรแกรมวงจรลง FPGA แล้วให้ทำการทดลองว่าได้ผลบวกเป็นไปตามทฤษฎีหรือไม่