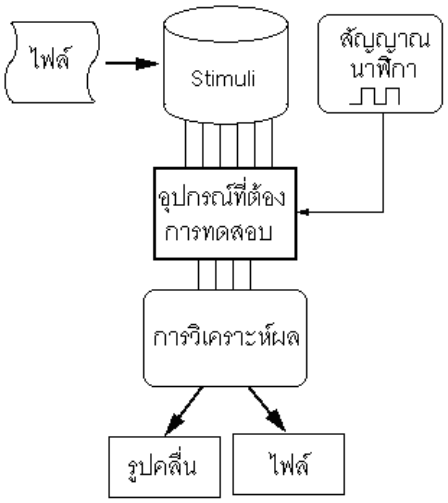
บทที่ 5

Testbench

5.1 Testbench คืออะไร

คือโปรแกรมภาษา VHDL ที่เขียนขึ้นมาเพื่อใช้ทดสอบระบบดิจิตอลที่ออกแบบด้วยภาษา VHDL (ถ้ากรณีระบบนั้นเขียนด้วยภาษา HDL อื่นก็สามารถทำได้เช่นเดียวกัน) ปกติแล้วระบบดิจิตอลที่ออกแบบขึ้นต้องมีการทดสอบการทำงานว่าเป็นไปตามที่ออกแบบไว้หรือไม่ การทดสอบนี้อาจทำได้ด้วยการจำลองการทำงาน หรือการสร้างเป็นของจริงแล้วใช้สัญญาณของจริงต่างๆป้อนเข้าระบบแล้วตรวจสอบผลการทำงาน แต่วิธีหลังนี้ บ้างครั้งอาจทำได้ยาก หรืออาจเกิดการเสียหายขึ้นได้ถ้าการทำงานไม่เป็นไปตามที่วางแผนไว้ ดังนั้นการทดสอบด้วยวิธีจำลองการทำงาน จะช่วยลดความยุ่งยาก และความเสียหายลงได้ เพราะว่าการจำลองการทำงาน จะกระทำในลักษณะของโปรแกรมบนเครื่องคอมพิวเตอร์เท่านั้น ดังนั้นสามารถทำได้อย่างรวดเร็วและไม่ต้องใช้อุปกรณ์อื่นๆเข้ามาช่วย

การจำลองการทำงานถ้าเป็นระบบเล็กๆและมีเงื่อนไขการทำงานที่ไม่ซับซ้อนนัก อาจทำได้โดยการสร้างสัญญาณและป้อนสัญญาณต่างๆด้วยมือ เช่นถ้าจำลองการทำงานด้วยโปรแกรม MODELSIM ก็ใช้คำสั่ง FORCE เพื่อกำหนดให้สัญญาณต่างๆมีค่าเป็นอะไร แล้วก็ใช้คำสัง RUN เพื่อดูผลการทำงาน แต่ถ้าเป็นระบบที่มีความซับซ้อนมากขึ้น ก ารทำเช่นนี้ย่อมไม่สะดวก ยิ่งถ้าต้องมีการแก้ไขโปรแกรมแล้วทดสอบหลายๆครั้ง โอกาสผิดพลาดย่อมเกิดขึ้นได้ ดังนั้นการแก้ไขปัญหานี้จึงควรใช้วิธีเขียนเป็นโปรแกรม Testbench เพื่อใช้สร้างสัญญาณที่ต้องการป้อนให้กับระบบ และวิเคราะห์ผลการทำงานที่ได้ออกจากระบบการทำด้วยวิ ธีนี้ ไม่ว่าจะทดสอบกี่ครั้งสัญญาณที่ใช้ทดสอบย่อมเหมือนเดิมหรือมีการแก้ไขให้ดีกว่าเดิมทุกครั้ง การทดสอบก็จะรอบคอบและทดสอบได้ครอบคลุมเงื่อนไขการทำงานทั้งหมด



รูปที่ 5-1 ระบบการทดสอบด้วย Testbench

Testbench ประกอบด้วยอะไรบ้าง ตามรูป 8-1 จะเห็นได้ว่า Testbench ประกอบด้วย ส่วนที่ใช้สร้างสัญญาณให้กับอุปกรณ์ที่ต้องการทดสอบ (Stimuli) ซึ่งสัญญาณนั้นอาจจะอ่านมาจากไฟล์ก็ได้ และถ้าในกรณีที่อุปกรณ์เป็นจำพวกรีจิสเตอร์ ก็ต้องมีส่วนสำหรับสร้ างสัญญาณนาฬิกา หลังจากป้อนสัญญาณเข้า

อุปกรณ์ที่ต้องการทดสอบแล้ว ก็ต้องนำสัญญาณ ที่ออกจากอุปกรณ์นั้นมาวิเคราะห์ ส่วนนี้คือส่วนการวิเคราะห์ผล โดยผลที่ได้อาจแสดงอยู่ในรูปของรูปคลื่นของ ไดอะแกรมเวลา หรือเก็บผลลัพธ์ลงไฟล์เพื่อนำมาตรวจสอบภายหลังได้ โมเดลภาษาเป็นการป้อนสัญญาณ (testvectors) เข้าสู่อุปกรณ์ที่ต้องการตรวจสอบ (DUT) แล้วทำการตรวจสอบ หรือเก็บผลลัพธ์ที่ได้ลงไฟล์

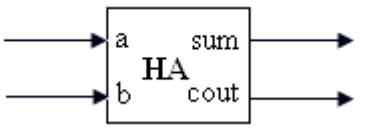
5.2 โครงสร้างของ VHDL Testbench

โครงสร้างของ Testbench ก็คล้ายๆกับการเขียนโมเดลของระบบหรือวงจรดิจิตอลทั่วๆไป คือประกอบด้วยส่วนประกอบหลัก 2 ส่วนคือ Entity และ Architecture แต่เนื่องจาก Testbench ใช้สำหรับทดสอบการทำงานของอุปกรณ์อื่นๆ ดังนั้น Testbench ไม่ต้องการการสังเคราะห์เป็นของจริง โครงสร้างของ Testbench ในส่วนของ Entity จึงไม่ต้องมีพอร์ทออกภายนอก จะมีเฉพาะส่วน Architecture ที่ต้องมีการกำหนด Component สำหรับเป็นอุปกรณ์ที่ต้องการทดสอบขึ้นมา นอกจากนั้นก็เป็นส่วนของการสร้างสัญญาณต่างๆและส่วนของการตรวจสอบผลการทำงาน โครงสร้างจึงเป็นดังนี้

```
entity TB_TEST is
end TB_TEST;

architecture BEH of TB_TEST is
  -- component declaration of the DUT
  -- internal signal definition
begin
  -- component instantiation of the DUT
  -- clock generation
  -- stimuli generation
end BEH;
```

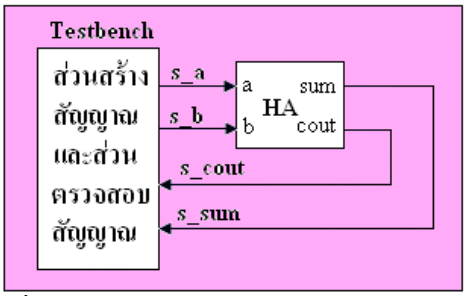
ตัวอย่างการเขียน Testbench สำหรับทดสอบการทำงานของ Half Adder (HA)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ha is
    port(a, b: in STD_LOGIC;
         sum, cout: out STD_LOGIC);
end ha;
architecture Behavioral of ha is
begin
   sum <= a xor b;
   cout <= a and b;
end Behavioral;
```



โมเดล Half Adder

Testbench สำหรับวงจร Half adder ประกอบด้วยส่วนสร้างสัญญาณเพื่อป้อนเข้าที่อินพุท a อินพุท b และส่วนตรวจสอบผลการทำงาน สำหรับตัวอย่างนี้จะยังไม่มีการเขียนโปรแกรมในส่วนการตรวจสอบผล



รูปที่ 5-2 Testbench ของวงจร Half adder

Testbench ของโปรแกรม Half adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity tb_ha1 is
end tb_ha1;
architecture beh of tb_ha1 is
component ha
        port(a, b: in STD_LOGIC;
                sum, cout: out STD_LOGIC);
end component;
signal s_a   : std_logic := '0';
signal s_b   : std_logic := '0';
signal s_sum : std_logic;
signal s_cout : std_logic;
begin
        DUT: ha  port map(s_a, s_b, s_sum, s_cout);
        test: process
        begin
                wait for 200 ns;
                s_a <= '1';
                s_b <= '0';
                wait for 200 ns;
                s_a <= '0';
                s_b <= '1';
                wait for 200 ns;
                s_a <= '1';
                s_b <= '1';
        end process;
end beh;
```

ข้อสังเกต std_logic ต้องกำหนดค่าให้ ถ้าไม่กำหนดจะมีค่าเป็น U

5.3 การสร้างสัญญาณทดสอบ

สัญญาณสำหรับการทดสอบมีอยู่ด้วยกัน 2 ประเภท ประเภทแรกมีรูปแบบเกิดขึ้นซ้ำๆกัน เช่น สัญญาณนาฬิกา ส่วนประเภทที่สอง เป็นสัญญาณที่เกิดขึ้นไม่ซ้ำกัน ขึ้นอยู่กับว่าต้องการให้มีรูปแบบใดๆ สัญญาณรีเซ็ต สัญญาณที่ใช้เป็น Test Vector ดังตัวอย่างต่อไปนี้

สัญญาณนาฬิกา (CLK)

การสร้างสัญญาณนาฬิกาแบบง่ายๆ

```
S_CLK <= not S_CLK after PERIOD/2;
```

การสร้างแบบนี้ไม่มีลูป แต่ควรกำหนดค่าเริ่มต้นให้กับ S_CLK เป็น ‘1’ หรือ ‘0’ แต่ไม่เป็น ‘น’ สัญญาณที่ได้จะเป็นสัญญาณสี่เหลี่ยม ส่วนแบบที่สองเป็นแบบที่ช่วงโลจิก 0 กับ โลจิก 1 ไม่เท่ากัน

```
process
        constant one_period: time :=  10 ns;
        constant zero_period: time :=  20 ns;
begin
        wait for one_period;
        clk <= ‘1’;
        wait for zero_period;
        clk <= ‘0’;
end process;
```

การสร้างสัญญาณโดยใช้คำสั่งแบบขนาน

เช่นการสร้างสัญญาณรีเซ็ท

```
S_RESET <=  '0',
            '1' after 20 ns,
            '0' after 40 ns;
```

ตัวอย่างการสร้าง Testbench สำหรับวงจรนับ

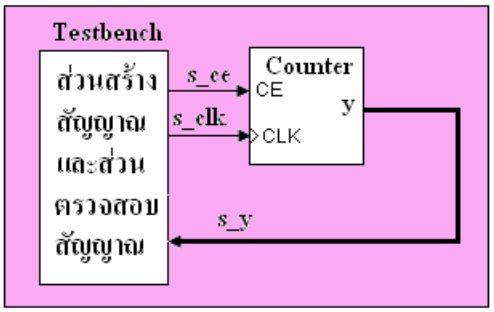
รูปที่ 5-3 วงจรนับ

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

entity coup is
    port(CLK: in STD_LOGIC;
            CE: in STD_LOGIC;
              y: inout INTEGER range 15 downto 0);
end coup;
architecture Behavioral of coup is
begin
process (CLK)
begin
  if CLK='1' and CLK'event then
    if CE = '1' then
      if y = 15 then
          y <= 0;
```

```
            else
               y <= y + 1;
            end if;
          else
            y <= y;
          end if;
        end if;
      end process;
      end Behavioral;
```

ลักษณะของ Testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity tb2c1 is
end tb2c1;
architecture beh of tb2c1 is
        component c1
        port(CLK: in STD_LOGIC;
                CE: in STD_LOGIC;
        y: inout INTEGER range 15 downto 0);
        end component;
        constant PERIOD : time := 10 ns;
        signal s_clk   : std_logic := '0';
        signal s_ce     : std_logic := '0';
        signal s_y      : integer range 15 downto 0;
begin
        DUT: c1  port map(s_clk, s_ce, s_y);
        s_clk <= not s_clk after period;
         S_ce <= '0',
        '1' after 100 ns,
        '0' after 1200 ns,
        '1' after 1400 ns;
end beh;
```



รูปที่ 5-4 Testbench ของวงจรนับ

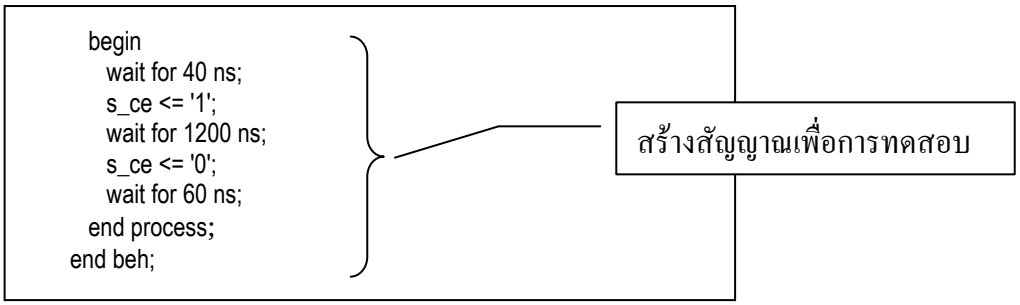
การสร้างสัญญาณโดยใช้คำสั่งแบบลำดับ

จากตัวอย่างวงจรนับถ้าใช้คำสั่งแบบลำดับสร้าง Testbench จะได้ดังนี้

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity counter_tb is
end counter_tb;
architecture beh of counter_tb is
  component coup
    port(CLK: in STD_LOGIC;
            CE: in STD_LOGIC;
              y: inout INTEGER range 15 downto 0);
  end component;
  constant PERIOD : time := 10 ns;
  signal s_clk   : std_logic := '0';
  signal s_ce     : std_logic := '0';
  signal s_y      : integer range 15 downto 0;
begin
  DUT: coup  port map(s_clk, s_ce, s_y);
  s_clk <= not s_clk after period;
  test: process
```

ส่วนกำหนดค่าของ architecture ได้แก่
- component
- internal signals
- subprograms
- constants

สร้างสัญญาณเพื่อการทดสอบ



ตัวอย่าง Testbench ของ Buffer Register

X เป็นสัญญาณอินพุท

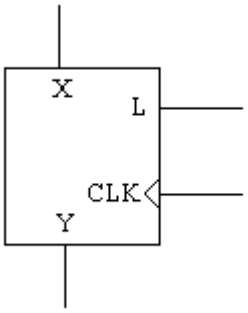
Y เป็นสัญญาณเอาท์พุท

L เป็นสัญญาณควบคุมการเก็บ กำหนดให้

ถ้า L = 0 ให้ Y = X เป็นการโหลดข้อมูลเก็บ

ถ้า L = 1 ให้ Y คงเดิมไม่เปลี่ยนแปลง

CLK เป็นสัญญาณนาฬิกา



รูปที่ 5-5 บล็อกไดอะแกรมของบัฟเฟอร์รีจิสเตอร์

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity reg3  is
   Port ( x : in std_logic_vector(3 downto 0);
          clk, ld : in std_logic;
          y : out std_logic_vector(3 downto 0));
end reg3;
architecture Behavioral of reg3 is
begin
  process (clk, ld)
  begin
     if clk = '1' and clk'event then
        if ld = '0' then
           y <= x;
        end if;
     end if;
  end process;
end Behavioral;
```

Testbench ของ Register

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity reg3_tb  is
end reg3_tb;
architecture Behavioral of reg3_tb is
  component reg3
    Port ( x : in std_logic_vector(3 downto 0);
           clk, ld : in std_logic;
           y : out std_logic_vector(3 downto 0));
```

```
  end component;
  signal xsig, ysig : std_logic_vector(3 downto 0);
  signal clksig, ldsig : std_logic;
begin
  ic1: reg3 port map(xsig, clksig, ldsig, ysig);
   clksig <= not clksig after 100 ns;
  test:  process
  begin
     ldsig <= '1';
     xsig <= "1100";
     wait for 450 ns;
     ldsig <= '0';
     wait for 300 ns;
     ldsig <= '1';
     xsig <= "1001";
     wait for 300 ns;
     ldsig <= '0';
     wait for 300 ns;
  end process;
end Behavioral;
```

5.4 การใช้คำสั่ง Assertion

การวิเคราะห์ผลการทำงานก็เป็นส่วนหนึ่งใน Testbench คำสั่ง Assertion เป็นเครื่องที่ใช้ดักจับการผิดพลาดของการทำงานได้ โดยคำสั่งนี้จะทำการตรวจสอบเงื่อนไข ถ้าเงื่อนไข เป็นเท็จ จะมีการแสดงข้อความที่อยู่ในออปชั่น Report ออกทาง simulator's transcript window และถ้ามีการระบุออปชั่น Severity level ค่านี้ก็จะถูกส่งให้กับ Simulator โดยค่านี้มีได้ 4 ระดับได้แก่

- Note เป็นระดับหมายเหตุ ไม่มีผลต่อการทำงานของวงจร
- Warning เป็นระดับการแจ้งเตือน โปรแกรมจำลองการทำงานยังคงทำงานต่อได้
- Error เป็นระดับที่โมเดลทำงานผิดพลาดต้องแก้ไข โปรแกรมจำลองการทำงานจะหยุดทำงาน
- Failure เป็นระดับความผิดพลาดที่รุนแรงก่อให้เกิดความเสียหาย คอมพิวเตอร์อาจหยุดทำงาน ได้

รูปแบบการใช้งาน

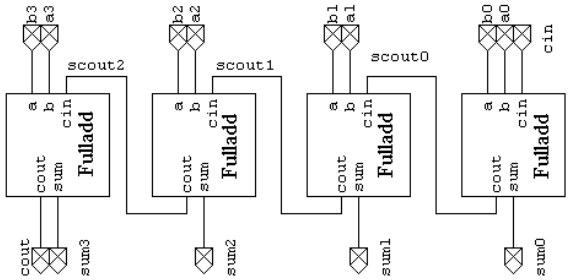
```
assert condition_expression
        report text_string
        severity severity_level ;
```

ตัวอย่างการใช้งาน assert

```
x1 <= '1';
x2 <= '1';
assert y = (x1 and x2)
        report "circuit failed"
        severity Error;
```

ถ้า x1 and กับx2 เป็นเท็จจะปรากฏข้อความ "circuit failed" ออกที่ transcript window และ ค่าError จะถูกส่งให้ Simulator

สำหรับตัวอย่างนี้แสดงให้เห็นการใช้ record เก็บสัญญาณทดสอบและสัญญาณเอาท์พุท เพื่อใช้ในการตรวจสอบว่า วงจรทำ งานถูกต้องหรือไม่ ถ้าทำงานไม่ถูกต้องคำสั่ง Assert จะรายงานความผิดพลาดออกมา วงจรที่ใช้ทดสอบเป็นวงจร Ripple Adder ขนาด 4 บิต ซึ่งเขียนเป็นโปรแกรมไว้ดังนี้



รูปที่ 5-6 บล็อกไดอะแกรมของวงจร Ripple adder ขนาด 4 บิต

วงจร Full adder

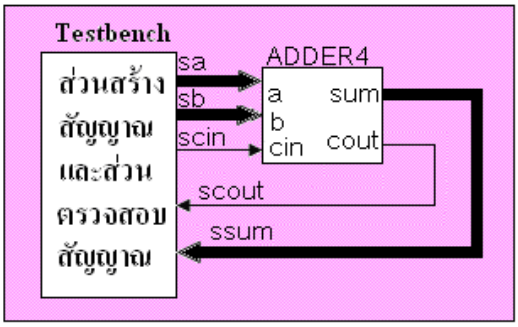
```
entity fulladd is
  port (a : in bit;
        b : in bit;
        cin : in bit;
        sum : out bit;
        cout : out bit);
end fulladd;
architecture addflow of fulladd is
begin
  sum <= a xor b xor cin;
  cout <= (a and b) or (a and cin) or (b and cin);
end addflow;
```

วงจร Ripple adder ขนาด 4 บิตที่ใช้คอมโปเนนต์ Full adder

```
entity adder4 is
  port (a : in bit_vector(3 downto 0);
        b : in bit_vector(3 downto 0);
        cin : in bit;
        sum : out bit_vector(3 downto 0);
        cout : out bit);
end adder4;
architecture adder4beh of adder4 is
component fulladd
  port (a : in bit;
        b : in bit;
        cin : in bit;
        sum : out bit;
        cout : out bit);
end component;
signal scout : bit_vector( 2 downto 0);
begin
  c1 : fulladd port map (a(0), b(0), cin, sum(0), scout(0));
  c2 : fulladd port map (a(1), b(1), scout(0), sum(1), scout(1));
```

```
  c3 : fulladd port map (a(2), b(2), scout(1), sum(2), scout(2));
  c4 : fulladd port map (a(3), b(3), scout(2), sum(3), cout);
end adder4beh;
```

Testbench ของวงจร Ripple adder ขนาด 4 บิต



รูปที่ 5-7 Testbench สำหรับวงจร Ripple adder ขนาด 4 บิต

```
entity test3adder4 is
end test3adder4;

architecture beh of test3adder4 is
component adder4
        port(a, b : in  bit_vector (3 downto 0);
                cin  :  in bit;
                sum : out bit_vector (3 downto 0);
                cout : out bit);
end component;

signal sa, sb, ssum  : bit_vector (3 downto 0);
signal scin, scout         : bit;

type test_record is record
   a :  bit_vector(3 downto 0);
   b :  bit_vector(3 downto 0);
   sum :  bit_vector(3 downto 0);
   cout :  bit;
end record;

type test_array is array(positive range <>) of test_record;
   constant test_patterns : test_array := (
        (a => "1111", b => "0001", sum => "0000", cout => '1'),
        (a => "0001", b => "0001", sum => "0010", cout => '1'),
        (a => "0001", b => "0001", sum => "0011", cout => '0'),
        (a => "1010", b => "0011", sum => "1101", cout => '0'),
        (a => "0011", b => "1010", sum => "1101", cout => '0'),
        (a => "0101", b => "0001", sum => "1010", cout => '0'),
        (a => "0011", b => "1100", sum => "1111", cout => '0'),
        (a => "0011", b => "1100", sum => "0000", cout => '0'),
        (a => "0101", b => "0101", sum => "1010", cout => '0'),
        (a => "0000", b => "0000", sum => "0000", cout => '0'));
begin
adder: adder4  port map (sa, sb, scin, ssum, scout);
scin <= '0';
```

```
test: process
variable vector : test_record;
begin
        for i in test_patterns'range loop
                vector := test_patterns(i);

                sa <= vector.a;                 -- apply the stimuls
                sb <= vector.b;                 -- apply the stimuls

                wait for 100 ns;                -- wait for the outputs to settle

                assert ssum  = vector.sum       -- check the results
                        report "Summation is wrong";
                assert scout = vector.cout
                        report "Cout is wrong";
        end loop;
end process;
end beh;
```

5.5 Text IO

Textual input และ Output หรือ Text I/O เป็น process สำหรับการอ่านและเขียนไฟล์อักขระ (Text files) ไฟล์อักขระนี้เป็นไฟล์ที่บรรจุด้วยรหัส ASCII ซึ่งมีคุณสมบัติดังนี้

- ประกอบด้วยอักษรของรหัส ASCII เป็นบรรทัด
- แต่ละบรรทัดปิดท้ายด้วยรหัส Carriage return
- แต่ละบรรทัดอาจมีหลายฟิลด์ได้ โดยแต่ละฟิลด์แยกกันด้วยเว้นวรรค

TEXTIO ถูกกำหนดไว้ใน Package TEXIO ของ IEEE std 1076 –1987 ดังนั้นเวลาจะใช้ TEXTIO ต้องเรียกใช้ use STD.TEXTO.ALL; โดยในแพกเกจจะประกอบด้วยโปรแกรมย่อยสำหรับจัดการกับไฟล์อักขระอย่างง่ายๆ

Package TEXTIO ทำอะไรได้บ้าง

Package TEXTIO of STD library

Important functions and procedures:

readline(...), read(...),

writeline(...), write(...),

endfile(...)

Additional data types (text, line)

READ / WRITE overloaded for all

predefined data types:

bit, bit_vector

boolean

character, string

integer, real

time

ขันตอนการจัดการไฟล์ด้วย TEXTIO

(ก) กำหนด Type (Type Declarations)

- Logical name of file
- type of file
- mode of file (in or out)
- physical file by path name

(ข) อ่านหรือเขียนข้อมูลกับไฟล์ทีละบรรทัดด้วยคำสั่ง READLN และ WRITELN ซึ่งเป็นการกระทำระหว่าง ไฟล์กับตัวแปรเก็บข้อมูลที่ละบรรทัด

(ค) อ่านหรือเขียนข้อมูลจากตัวแปรที่เก็บข้อมูลของแต่บรรทัดที่ละฟิลด์ เป็นการกระทำระหว่างตัวแปรเก็บข้อมูลที่ละบรรทัดกับ ตัวแปรหรือสัญญาณ

(ง) ทำฟังก์ชั้นการจบไฟล์ ENDFILE

รูปแบบการกำหนดไฟล์

```
file logical_file_name : TEXT is in "..physical_file_name";
```

Logical file name เป็นชื่อที่ใช้สำหรับการอ้างถึงเท่านั้นทำหน้าที่เป็น file handle นั้นเอง

TEXE ชนิดของไฟล์สำหรับ TEXTIO เป็นได้เพียง TEXT เท่านั้น

Mode เป็น IN สำหรับอ่านจากไฟล์
OUT สำหรับเขียนไฟล์
โดยไฟล์หนึ่งๆเปิดได้เพียงโหมดเดียว

physical_file_name ชื่อไฟล์ที่จะใช้งานจริง

การอ่านแและขียนไฟล์

รูปแบบการอ่านไฟล์

```
READLN (logical_file_name, line_name);
```

รูปแบบการเขียนไฟล์

```
WRITELN (logical_file_name, line_name);
```

การอ่านหรือเขียนต้องทำที่ละบรรทัด และต้องเป็นการกระทำระหว่างไฟล์ที่ถูกระบุด้วย logical_file_name กับ ตัวแปร line_name ที่เป็นตัวแปรที่กำหนดด้วยคำสั่ง Variable ให้เป็นชนิด Line เท่านั้น

การอ่านเขียนค่ากับตัวแปรชนิด Line

รูปแบบการอ่านที่ละฟิลด์

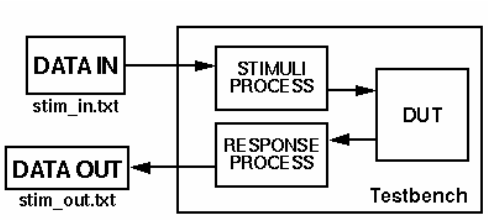
```
READ (line_name, object_name);
HREAD (line_name, object_name);      -- อ่านแบบ HEXแล้วแปลงเป็น  binary vector
```

รูปแบบการเขียนที่ละฟิลด์

```
WRITE (line_name, object_name);
HWRITE (line_name, object_name);     -- แปลง binary vector เป็น HEX
```

- การอ่านหรือเขียนแต่ละครั้งจะได้ครั้งละฟิลด์ ฟิลด์ซ้ายมือสุดจะถูกกระทำเป็นฟิลด์แรก
- HREAD และ HWRITE เป็นฟังก์ชั้นใน IEEE.std_logic_textio Package
- Object_name เป็นอะไรก็ได้เช่น signal, variable

การใช้ TEXTIO ใน TESTBENCH



รูปที่ 5-8 ลักษณะการใช้ TextIO ใน Testbench

ตัวอย่างโปรแกรมคูณขนาด 4 บิต คูณ 4 บิต

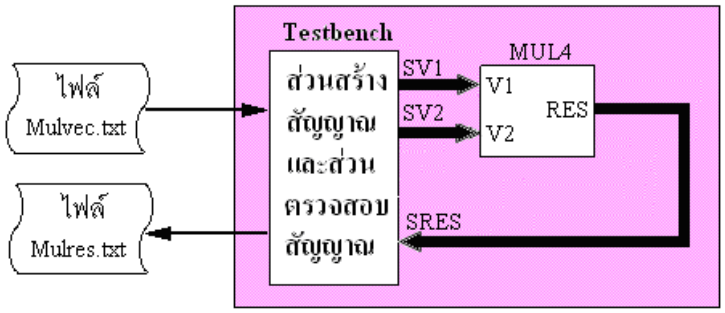
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity MUL4 is
port (V1, V2 : in std_logic_vector(3 downto 0);
      RES : out std_logic_vector(7 downto 0));
end MUL4;
```

```
architecture RTL of MUL4 is
begin
  RES <= V1 * V2;
end RTL;
```

ตัวอย่างไฟล์ข้อมูลสำหรับป้อนให้กับ V1 และ V2 ชื่อ mulvec.txt

```
0 0
0 8
0 F
1 0
1 8
1 F
4 0
4 8
4 F
8 8
8 F
F 0
F 8
F 9
```

โปรแกรม Testbench แบบที่ 1 อ่านค่าจากไฟล์อินพุท Mulvec.txt แล้วส่งเข้าสู่วงจรคูณ ตรวจสอบผลการทำงานจากหน้าจอ



รูปที่ 5-9 ลักษณะการใช้ TextIO ใน Testbench

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;
entity MUL4_TEB is
end MUL4_TEB;
architecture BEH of MUL4_TEB is
component MUL4
port (V1, V2 : in std_logic_vector(3 downto 0);
      RES : out std_logic_vector(7 downto 0));
end component;
signal SV1, SV2 : std_logic_vector(3 downto 0);
signal SRES : std_logic_vector(7 downto 0);
begin
DUT: MUL4 port map(V1 => SV1, V2 => SV2, RES => SRES);
```

```
STIMULI: process
  variable NUM_IN : line;
  variable CHAR : character;
  variable NUM1 : std_logic_vector(3 downto 0);
  variable NUM2 : std_logic_vector(3 downto 0);
  file STIMULI_IN: text is in "mulvec.txt";
begin
  wait for 100 ns;
  while not endfile(STIMULI_IN) loop
    readline(STIMULI_IN, NUM_IN);
    hread(NUM_IN,NUM1);
    SV1 <= NUM1;
    read(NUM_IN, CHAR);
    hread(NUM_IN, NUM2);
    SV2 <= NUM2;
    wait for 500 ns;
  end loop;
end process STIMULI;
end BEH;
```

โปรแกรม Testbench แบบที่ 2 อ่านค่าจากไฟล์อินพุท Mulvec.txt แล้วส่งเข้าสู่วงจรคูณ ผลการทำงานเก็บลงไฟล์ Mulres.txt

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;

entity MUL4_TEB2 is
end MUL4_TEB2;

architecture BEH of MUL4_TEB2 is

component MUL4
port (V1, V2 : in std_logic_vector(3 downto 0);
      RES : out std_logic_vector(7 downto 0));
end component;

signal SV1, SV2 : std_logic_vector(3 downto 0);
signal SRES : std_logic_vector(7 downto 0);

begin
DUT: MUL4 port map(V1 => SV1, V2 => SV2, RES => SRES);

STIMULI: process
  variable NUM_IN : line;
  variable CHAR : character;
  variable NUM1 : std_logic_vector(3 downto 0);
  variable NUM2 : std_logic_vector(3 downto 0);
  file STIMULI_IN: text is in "mulvec.txt";
begin
  wait for 100 ns;
  while not endfile(STIMULI_IN) loop
    readline(STIMULI_IN, NUM_IN);
    hread(NUM_IN,NUM1);
```

```
    SV1 <= NUM1;
    read(NUM_IN, CHAR);
    hread(NUM_IN, NUM2);
    SV2 <= NUM2;
    wait for 500 ns;
  end loop;
end process STIMULI;

RESPONSE: process(SRES)
  variable NUM_OUT : line;
  variable CSPACE : character := ' ';
  file STIMULI_OUT: text is out "mulres.txt";
  begin
    write(NUM_OUT,now);
    write(NUM_OUT,CSPACE);
    write(NUM_OUT,SV1);
    write(NUM_OUT,CSPACE);
    write(NUM_OUT,SV2);
    write(NUM_OUT,CSPACE);
    write(NUM_OUT,SRES);
    write(NUM_OUT,CSPACE);
    hwrite(NUM_OUT,SRES);
    writeline(STIMULI_OUT,NUM_OUT);
end process RESPONSE;

end BEH;
```

ไฟล์ผลการทำงาน mulres.txt

```
0 ns UUUU UUUU UUUUUUUU 00
0 ns UUUU UUUU XXXXXXXX 00
100 ns 0000 0000 00000000 00
2100 ns 0001 1000 00001000 08
2600 ns 0001 1111 00001111 0F
3100 ns 0100 0000 00000000 00
3600 ns 0100 1000 00100000 20
4100 ns 0100 1111 00111100 3C
4600 ns 1000 1000 01000000 40
5100 ns 1000 1111 01111000 78
5600 ns 1111 0000 00000000 00
6100 ns 1111 1000 01111000 78
6600 ns 1111 1001 10000111 87
```

แบบฝึกหัด

5.1 จาก VHDL Code จงเขียนรูปคลื่นของสัญญาณ clk และ y

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ex_tb is
end ex_tb;
architecture beh of ex_tb is
constant PERIOD : time := 20 ns;
  signal clk   : std_logic := '0';
  signal y     : std_logic := '0';
begin
  clk <= not clk after PERIOD;
  test: process
  begin
    wait for 30 ns;
    y <= '1';
    wait for 40 ns;
    y <= '0';
    wait for 40 ns;
    y <= 'Z';
    wait for 30 ns;
    y <= '1';
  end process;
end beh
```

5.2 จากบล็อกไดอะแกรมของวงจรบวกในรูปที่ 5-10 ประกอบด้วยสัญญาณ ตัวตั้ง A และ ตัวบวก B มีขนาดอย่างละ บิต 8 ผลบวก S มีขนาด บิต 8 และตัวทดออกC มีขนาด บิต 1

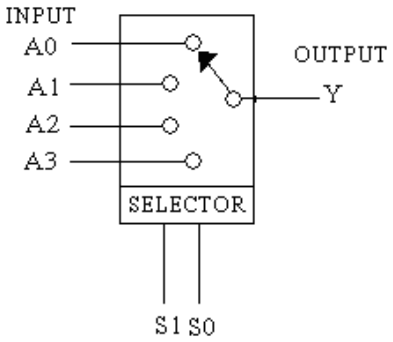
รูปที่ 5-10 บล็อกไดอะแกรมของวงจรบวก

กำหนดให้มี component ของวงจรเป็นดังนี้

```
component ADDER
   port(A : in   std_logic_vector(7 downto 0);
        B  : in   std_logic_vector(7 downto 0);
        C  : out std_logic;
        S  : out std_logic_vector(7 downto 0));
 end component;
```

จงเขียน Testbench เพื่อเพื่อทำการทดสอบ วงจรบวกนี้ กำหนดให้ ข้ อมูลที่จะป้อนเข้าสัญญาณ A และ B อ่านจากไฟล์อักขระ (Text File) ผลลัพธ์ที่ได้เก็บลงไฟล์อักขระเช่นเดียวกัน

5.3 จงเขียน Testbench สำหรับตรวจสอบการทำงานของ 4 -to-1 multiplexer ในรูป 5-11 (ก) โดยให้ข้อมูลอินพุทสำหรับป้อนเข้าสัญญาณ S1 S0 A3 A1 A2 และ A0 อ่านจากไฟล์และผลลัพธ์ที่ได้ให้เก็บไว้ในไฟล์ ตามรูปแบบไฟล์ที่กำหนดให้ไว้ในรูปที่ 5-11 (ข)



Input		Output
S1	S0	Y
0	0	Y = A0
0	1	Y = A1
1	0	Y = A2
1	1	Y = A3

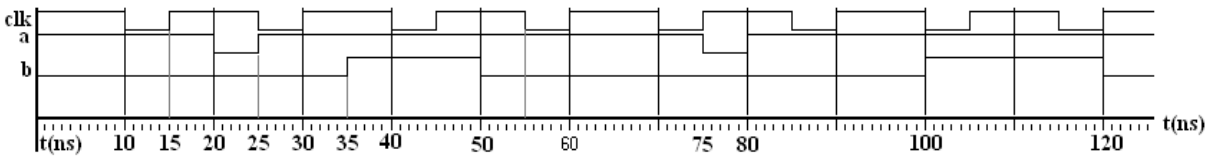
ตารางการทำงาน

รูปที่ 5-11 (ก) บล็อกไดอะแกรมและตารางการทำงานของ 4-to-1 multiplexer

Input File							Output File						
S1	S0	A3	A2	A1	A0		S1	S0	A3	A2	A1	A0	Y
0	0	0	0	0	0		0	0	0	0	0	0	
0	0	0	0	0	1		0	0	0	0	0	1	
0	1	0	0	0	0		0	1	0	0	0	0	
0	1	0	0	1	0		0	1	0	0	1	0	
1	0	0	0	0	0		1	0	0	0	0	0	
1	0	0	1	0	0		1	0	0	1	0	0	
1	1	0	0	0	0		1	1	0	0	0	0	
1	1	1	0	0	0		1	1	1	0	0	0	

รูปที่ 5-11 (ข) รูปแบบไฟล์

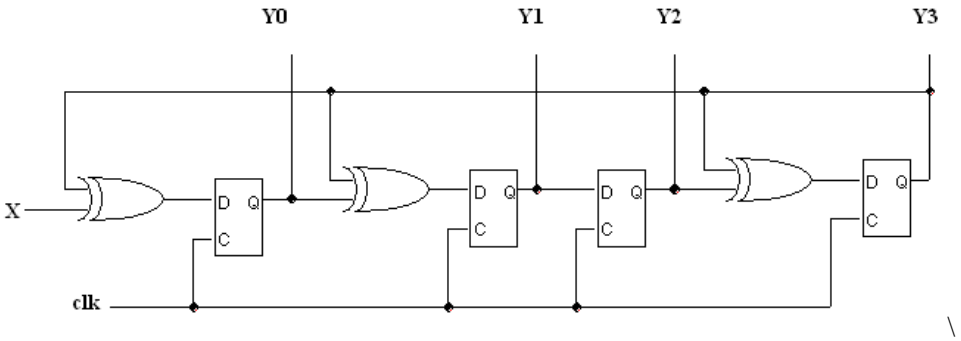
5.4 จากไดอะแกรมเวลาในรูปที่ 5-12 จงเขียน VHDL Testbench เพื่อสร้างสัญญาณทั้งสาม ให้ได้ตามรูป



รูปที่ 5-12 ไดอะแกรมเวลา

5.5 กำหนดให้ Entity coder ในรูป 5-13 เป็น Entity ของ VHDL code ที่ถูกเขียนขึ้นเพื่อให้มีการทำงานได้ตามโลจิกไดอะแกรมในรูปที่ 5-13 จงเขียน Testbench เพื่อทำการทดสอบว่า VHDL ที่เขียนขึ้นทำงานได้ถูกต้องหรือไม่ โดย Testbench นี้ต้องสามารถให้รายงานข้อผิดพลาดด้วยคำสั่ง Assertion ได้

```
entity coder is
   port (clk : in std_logic;
        x : in std_logic;
            y : out std_logic_vector(3 downto 0));
end coder;
```



กำหนดให้ ค่าเริ่มต้น Y3 ถึง Y0 = “0000”

รูปที่ 5-13 โลจิก ไดอะแกรม