ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

Experiment Adv DSD-06:	Sequential circuit and Finite state machine coding style

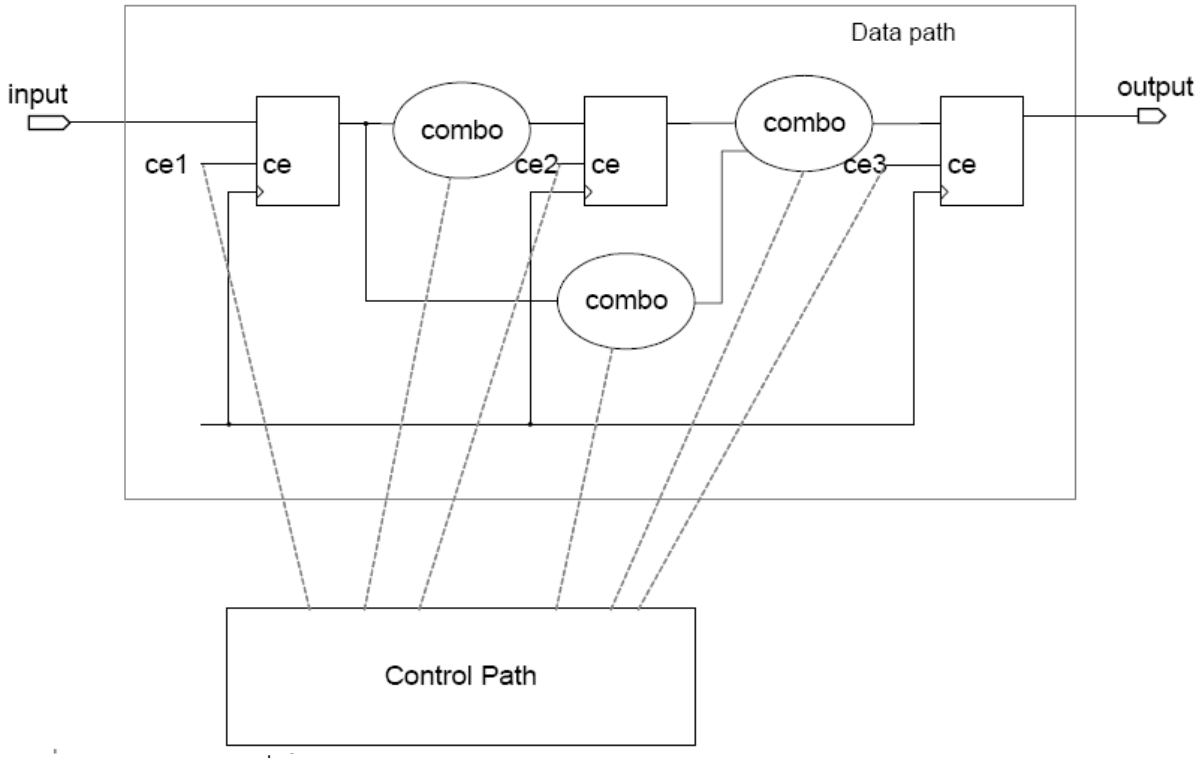
วัตถุประสงค์:

1. เข้าใจวิธีการ และสามารถออกแบบ Finite state machine diagram ได้
2. สามารถเขียน VHDL code จาก Finite state machine diagram ได้

บทนำ

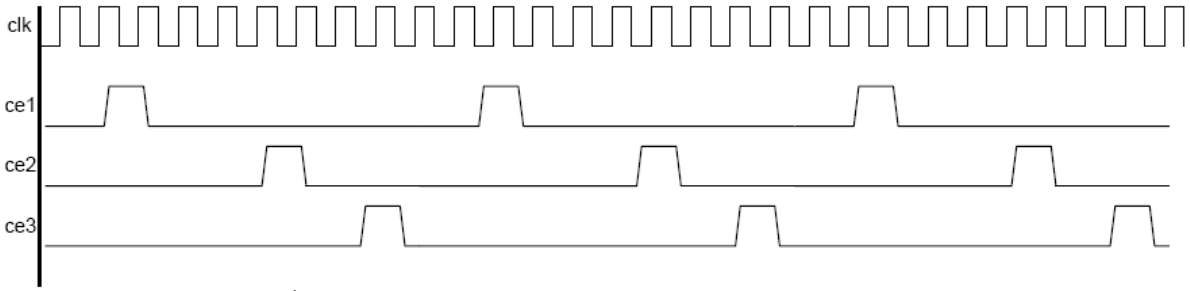
การทดลองนี้เป็นการทดลองออกแบบวงจร Sequential Circuit ซึ่งส่วนใหญ่จะใช้ในการออกแบบวงจรสาหรับสร้างสัญญาณควบคุม เนื่องจากวงจรลักษณะนี้สามารถสร้างสัญญาณที่มีลักษณะเป็น Sequential ได้ การทดลองจะให้ลองเขียนโค้ดเพื่อสร้างวงจร Sequential Circuit โดยใช้วิธีเขียนโค้ดแบบ Finite State Machine (FSM) ซึ่งจะมีรูปแบบต่างจากที่เคยทดลองในแลบที่ผ่านๆมา

แนวความคิดเบื้องต้นในการออกแบบ หลังจากกาหนดฟังก์ชั่นของวงจรที่จะออกแบบแล้ว ให้มองวงจรที่จะออกแบบเป็นสองส่วนคือ ส่วน Data path และ Control path Data path ก็คือส่วนที่สัญญาณต่างๆวิ่งเข้าไปทาการประมวลผลตั้งแต่อินพุทจนถึงเอาท์พุท ประกอบไปด้วย รีจีสเตอร์ และวงจรคอมไบเนชั่นต่างๆ ส่วน Control path จะเป็นวงจรที่สร้างรูปแบบสัญญาณไปควบคุม รีจีสเตอร์ และคอมไบเนชั่น เหล่านั้นอีกที ดูรูปที่ 6.1 ประกอบ วิธีการคิดและแบ่งวงจรเป็นสองส่วน



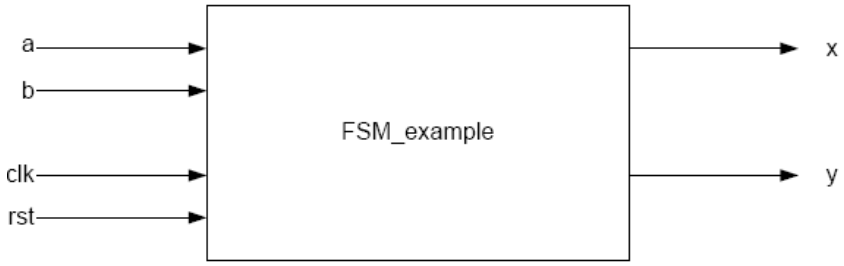
รูปที่ 6.1 แบ่งวงจรออกเป็นสองส่วน Data Path และ Control Path

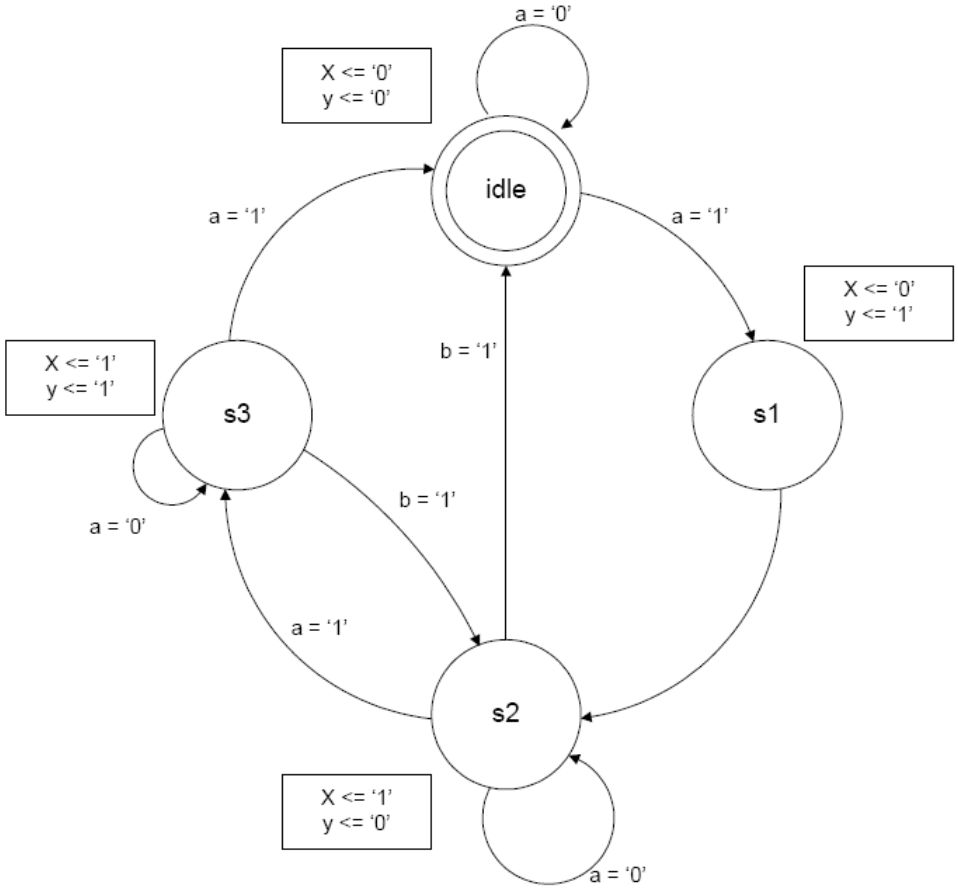
จากรูปที่ 6.1 จะเห็นว่าสัญญาณอินพุท และเอาท์พุทจะอยู่กับส่วน Data path ส่วน Control Path จะทำหน้าที่เพียงสร้างสัญญาณควบคุมทั้งหมด เพื่อกำหนดจังหวะและรูปแบบการทางานของส่วน Data Path ทั้งหมด ซึ่งสัญญาณควบคุมก็จะถูกออกแบบให้ทำงานสัมพันธ์สอดคล้องกันทั้งระบบ การออกแบบ Control Path จะต่างจากการออกแบบ data path เนื่องจากเอาท์พุทของ Control path จะเป็นสัญญาณที่เป็นลำดับขั้นสัมพันธ์กัน หรือที่เรียกว่าเป็น Sequence ตามจังหวะการทำงานที่นักออกแบบต้องการ เช่น ตัวอย่างรูปที่ 6.1 นักออกแบบอาจต้องการให้สัญญาณ ce1, ce2 และ ce3 มีรูปแบบการทำงานดังแสดงในรูปที่ 6.2



รูปที่ 6.2 ตัวอย่างสัญญาณควบคุม มีลักษณะเป็น Sequence

ยกตัวอย่างสมมุติว่าต้องการออกแบบวงจร Sequential ใด ๆ วงจรหนึ่งมีพอร์ตอินพุท เอาท์พุท และมีฟังก์ชั่นการทางานตาม FSM ตามตัวอย่างในรูปที่ 6.3





รูปที่ 6.3

ในความหมายของ FSM ในรูปที่ 6.3 คือ หลังจากวงจรมีการรีเซตแล้ว จะเริ่มการทำงานที่สถานะ idle คือให้ output ตามเงื่อนไขในตารางสี่เหลี่ยม คือให้ x= '0', y = '0' หลังจากนั้นการเปลี่ยนสถานะจะขึ้นกับอินพุท ถ้าอินพุท a = '0' ก็จะคงสถานะ idle เช่นเดิม และให้เอาท์พุทเช่นเดิม เมื่อใดก็ตามที่ a = '1' (พิจารณาตามขอบของสัญญาณนาฬิกา) สถานะจะเปลี่ยนไปที่ S1 ซึ่งให้เอาท์พุท x <= '0', y <= '1' และในขอบสัญญาณนาฬิกาลูกถัดไปจะเปลี่ยนสถานะไป S2 ทันที เนื่องจากการเปลี่ยนสถานะของ S1 ไม่ขึ้นกับสัญญาณอินพุทใด ๆ ส่วนที่สถานะอื่น ๆ ก็สามารถพิจารณาได้ด้วยหลักการเดียวกัน หลังจากออกแบบ FSM Diagram แล้ว ก็สามารถนาโครงสร้างมาเขียนเป็น VHDL Code ได้ ตามตัวอย่างรูปที่

ขั้นตอนการทดลอง

ออกแบบวงจร Light Pattern

1. ให้ออกแบบวงจรไฟวิ่ง โดยมี 3 pattern ตามเงื่อนไขในรูปที่ 6.5 โดยใช้วิธีออกแบบวงจรแบบ Finite State Machine การทำงานหลังจากรีเซตแล้ว ไฟจะทำการแสดงผลเป็นไฟวิ่งตาม Pattern#1 วนไปเรื่อย ๆ ถ้าหาก push switch BTN0 โดนกด(สัญญาณ sw1 เป็น '1') ก็จะเปลี่ยนรูปแบบเป็น Pattern#2 ถ้ากดอีกก็จะเปลี่ยนเป็น Pattern#3 และถ้ามีการกด sw1 อีกครั้งก็จะกลับไป Pattern#1 อีกครั้ง

```
architecture FSM of FSM_example is
        type stateTYPE is (idle,S1,S2,S3);
        signal cur_state :stateTYPE;

begin

process(rst,clk)
begin
        if rst = '1' then
              cur_state <= idle;
        elsif clk'event and clk= '1' then
           case cur_state is
          ---- idle state
                      when idle =>
                      --- output condition
                          x <= '0';
                          y <= '0';
                      --- state transition
                      if a = '1' then
                          cur_state <= S1;
                      else
                           cur_state <= idle;
                      end if;
        ----- S1 state
                      when S1 =>
                        --- output condition
                        x <= '0';
                        y <= '1';
                      --- state transition
                          cur_state <= S2;

        ----- S2 state
                      when S2 =>
                        --- output condition
                          x <= '1';
                          y <= '0';
                      --- state transition
                          if a = '1' then
                                cur_state <= S3;
                          elsif b = '1' then
                                cur_state <= idle ;
                          end if;

        ----- S3 state
                      when S3 =>
                        --- output condition
                          x <= '1';
                          y <= '1';
                      --- state transition
                          if a = '1' then
                              cur_state <= idle;
                          elsif b = '1' then
                                cur_state <= S2 ;
                          end if;
                      when others  => cur_state <= idle;
            end case;

        end if;
end process;

END;
```

State TYPE declaration

State signal declaration

Assign 'idle' as Start state

'Idle' state

Output condition of 'idle' state

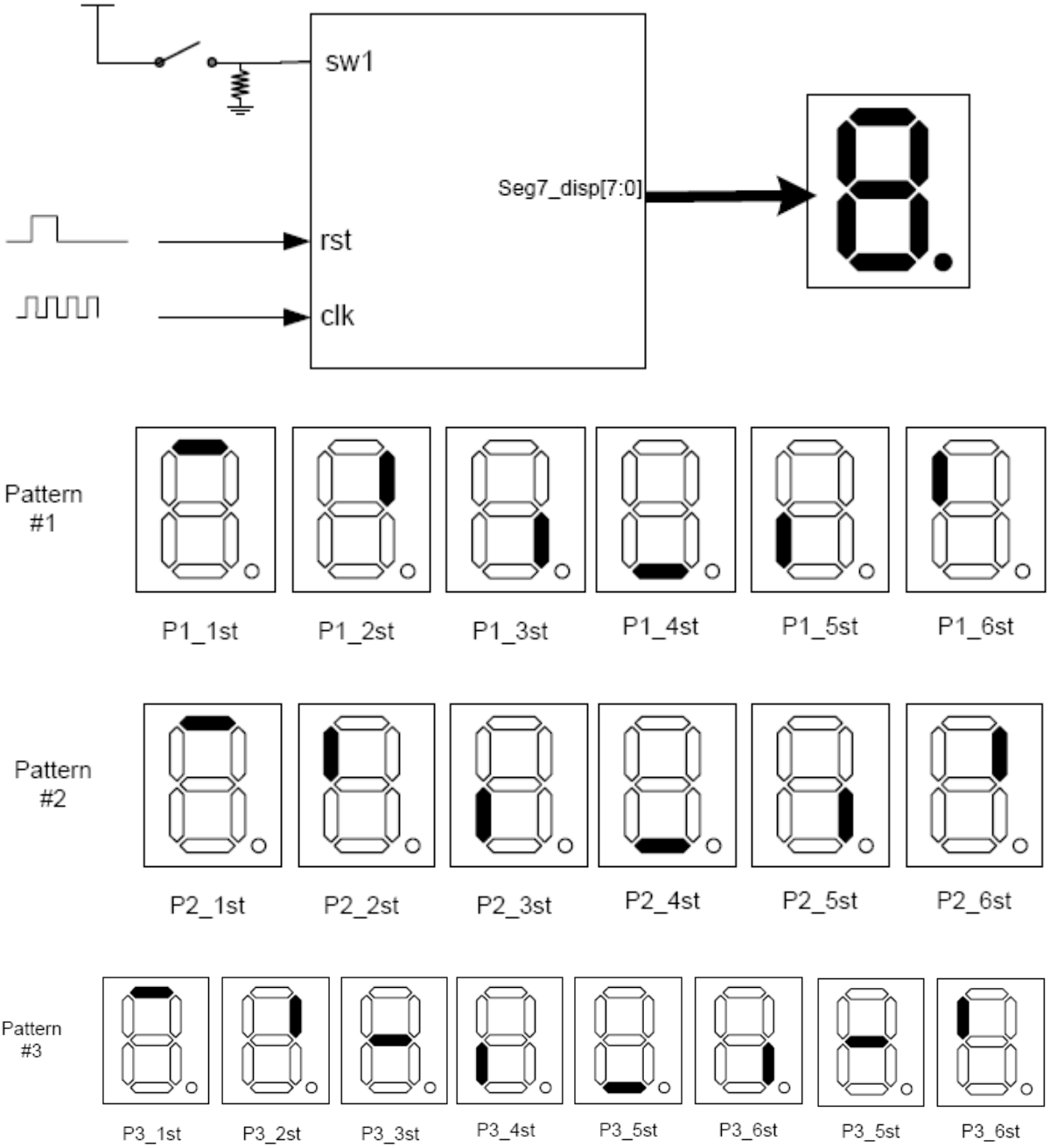
State transition condition
Depend on input

'S1' state

Output condition of 'idle' state

State transition
Not depend on input

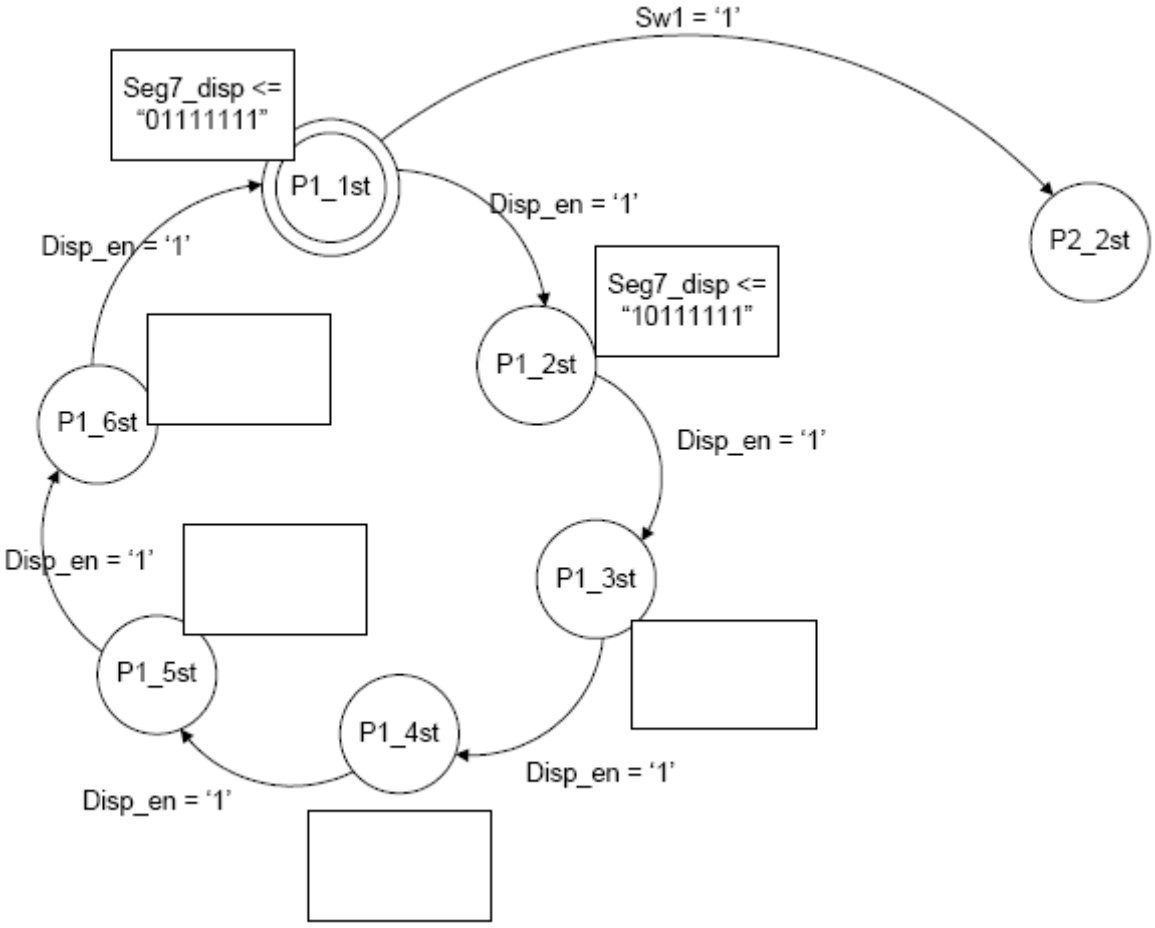
รูปที่ 6.4



รูปที่ 6.5

2. ทำการเขียน FSM state diagram เขียนเพิ่มเติมจากรูปที่ให้ในรูปที่ 6.6 ให้สมบูรณ์ โดยดูรูปที่ 6.3 เป็นตัวอย่าง

Note : สัญญาณ disp_en ทำหน้าที่หน่วงเวลาในการแสดงผล เนื่องจากบอร์ดที่ใช้ทดลอง ทำงานที่ 50Mhz ซึ่งถ้าไม่หน่วงเวลาจะไม่สามารถมองเห็นรูปแบบของไฟวิ่งได้ โดยออกแบบด้วย counter นับ 0 – 10,000,000 และสัญญาณ disp_en จะเป็น ‘1’ ทุกๆรอบการนับเท่ากับ 10,000,000 และเป็น ‘0’ ที่ค่านับค่าอื่น



รูปที่ 6.6

3. เปิด Project Navigator สร้าง project ใหม่ใช้ชื่อว่า Lab06 เขียน VHDL code เพื่อสร้างวงจรให้ทำงานตามรูปที่ 6.6 ตั้งชื่อไฟล์ว่า light_pattern.vhd และ Entity ชื่อ light_pattern ดูตัวอย่างการเขียนแบบ FSM ได้ในตัวอย่างในรูปที่ 6.4

Note: ให้สร้าง process สาหรับสร้างสัญญาณ disp_en และ process สาหรับ FSM แยกกัน

4. สร้าง Test bench สาหรับทดสอบการทางาน และจาลองการทางาน, ดูรูปสัญญาณด้วย Modelsim, ทำการแก้ไขให้ทำงานให้ถูกต้อง

5. ทำการ implement โดยกำหนด Pin สำหรับสัญญาณต่างๆ ดังรูปที่ 6.7

```
NET "clk"  LOC = "T9"  ;
NET "rst"  LOC = "L14"  ;

NET "seg7_disp<0>"  LOC = "P16"  ;
NET "seg7_disp<1>"  LOC = "N16"  ;
NET "seg7_disp<2>"  LOC = "F13"  ;
NET "seg7_disp<3>"  LOC = "R16"  ;
NET "seg7_disp<4>"  LOC = "P15"  ;
NET "seg7_disp<5>"  LOC = "N15"  ;
NET "seg7_disp<6>"  LOC = "G13"  ;
NET "seg7_disp<7>"  LOC = "E14"  ;

NET "sw1"  LOC = "M13"  ;
```

รูปที่ 6.7

6. สร้างไฟล์สาหรับดาวน์โหลด และทดสอบการทำงานบนบอร์ด โดยดูรูปแบบของไฟวิ่งว่าเป็นตามข้อกำหนดหรือไม่ และกด BTN0 เพื่อดูการเปลี่ยนรูปแบบไฟวิ่งว่าทำงานถูกต้องหรือไม่

สรุป

a. ในการออกแบบวงจรที่ซับซ้อน และที่มีขนาดใหญ่ที่ดีให้มองวงจรเป็นสองส่วนคือ Data path และ Control path ส่วน Control path จะมีลักษณะเป็น Sequential Circuit สามารถออกแบบได้ด้วย Finite State machine

b. การเขียนโค้ดเพื่อสร้าง Finite State machine จะต้องประกาศ TYPE เพื่อกาหนด state ต่างๆของวงจร และใช้ case – statement ในการระบุเงื่อนไขของแต่ละ state

c. ภายใต้ case แต่ละ case จะประกอบด้วยเงื่อนไขของเอาท์พุท และเงื่อนไขการเปลี่ยน state โดยเงื่อนไขการเปลี่ยน state อาจจะขึ้นกับอินพุท หรือไม่ขึ้นกับ อินพุทก็ได้