



# การทดลองที่ 9 State Machines, it's Application

หน้า  
1 / 10

ภาควิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์	ภาคการศึกษาที่..... 1 .....	ปีการศึกษา..... 2564 .....
รหัสวิชา 010113026 ชื่อวิชา Digital Laboratory	ตอนเรียน ..... 7 .....	หมายเลขอีเมล..... ~ .....
รหัสนักศึกษา..... 6201011631188 .....	ชื่อ-นามสกุล..... นงนัทธา กาญจนานุรุห์ .....	
อาจารย์ผู้สอน..... CSP .....	เวลาที่ทำการทดลอง 13.00 - 16.00 .....	วันที่ 30/09/64 .....

## การทดลองที่ 9

### State Machines and its Applications

#### วัตถุประสงค์

- เพื่อให้สามารถใช้โปรแกรมคอมพิวเตอร์เพื่ोจำลองการทำงานของวงจรลอจิกเกทได้
- เพื่อให้สามารถประยุกต์ใช้วงจรและอุปกรณ์ดิจิทัลเพื่อออกแบบระบบงานที่ซับซ้อนได้
- เพื่อให้สามารถประยุกต์ใช้การออกแบบระบบงานดิจิตอลด้วยเทคนิคทาง state diagram ได้

#### อุปกรณ์

- ระบบคอมพิวเตอร์ 1 เครื่อง พร้อมติดตั้งโปรแกรม Quartus II เวอร์ชัน 8.0 (Student Edition) ขึ้นไป
- บอร์ดทดลอง Cyclone3-Lab01 1 บอร์ด
- สาย J-TAG 1 เส้น ใช้รุ่น USB-Blaster (สำหรับเครื่อง Notebook) หรือรุ่น Byte-Blaster (สำหรับเครื่อง PC)
- บอร์ดแสดงผล 7-segment (รุ่นแป้นพิมพ์ Keypad สีขาว)

## การทดลอง

### A1 บทนำ: แนวทางการออกแบบระบบงาน

ในการทดลองนี้จะสร้างระบบงานชื่อ “เครื่องจำหน่ายเครื่องดื่มแบบหยดเหรียญ Vending Machine” ซึ่งจะเป็นตัวอย่างการนำเทคนิคการออกแบบแบบ state diagram มาช่วยในการออกแบบ ดังนั้นจึงมีความจำเป็นที่จะต้องทำความเข้าใจการทำงานของเครื่อง Vending Machine ให้เข้าใจพฤติกรรมของมันอย่างถูกต้องก่อน แล้วค่อยเริ่มต้นลงมือทำการออกแบบ หากทำตามขั้นตอนนี้แล้วจะช่วยให้ได้เห็นประโยชน์ที่ชัดเจนของแนวทางการออกแบบด้วยเทคนิค state diagram ได้เป็นอย่างดี

### การทำงานของเครื่อง Vending Machine:

#### การทำงานของเครื่อง Vending Machine

- มีเฉพาะเครื่องดื่มแบบกระป๋อง ราคากระป่องละ 15 บาท เมื่อเหรียญครบตามราคาก็ปล่อยสินค้าให้ทันที
- เหรียญที่รับ เฉพาะเหรียญ 5 บาท (Nikel ใช้อักษรย่อ N)  
และเหรียญ 10 บาท (Dime ใช้อักษรย่อ D)
- ไม่ทอนเงิน หากยอดเหรียญเกินราคាដ้วย เช่น 10 บาท 2 เหรียญ จะปล่อยสินค้าให้ 1 กระป่อง และรอรับเงินเพิ่มอีก 10 บาทเพื่อให้ครบ 15 บาท สำหรับการขายเพิ่มอีก 1 กระป่อง
- มีตัวเลขแสดงจำนวนเงินที่ใส่เข้าไปหากเงินยังไม่ถึง 15 บาท แต่ถ้าเกิน 15 บาทจะแสดงเงินที่เหลืออยู่หลังจากหักราคาขายสินค้าไปแล้ว



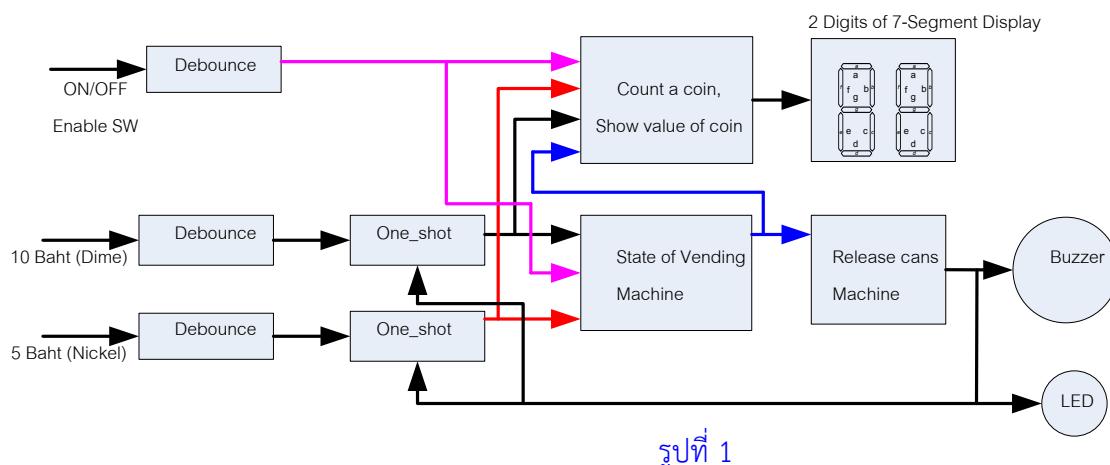
5 มีการป้องกันการกินเหรียญ ในช่วงที่ปล่อยกระปองน้ำเครื่องจะ **ไม่รับเหรียญเพิ่ม** จนกว่าจะสิ้นสุดขั้นตอนการปล่อยกระปอง (จะมีไฟสัญญาณกระพริบเตือน พร้อมส่งเสียงบีบราๆ 2 วินาที)

## A2 เตรียมการออกแบบ: โครงสร้างของเครื่อง Vending Machine

เราจะเป็นจะต้องจินตนาการกลไกของบอร์ดทดลองเพื่อให้ทำหน้าที่เป็นเครื่อง Vending Machine ดังนี้

- **ใช้ SW6 (สวิตซ์เลื่อน)** ทำหน้าที่เป็นสวิตซ์ปิดเปิดเครื่อง (ควบคุมที่ขาสัญญาณ Enable)
- **ใช้ PB2 (สวิตซ์กดติดปล่อยดับ)** ทำหน้าที่แทนการ**หยดเหรียญ 5 บาท 1 เหรียญ**ต่อการกดหนึ่งครั้ง
- **ใช้ PB3 (สวิตซ์กดติดปล่อยดับ)** ทำหน้าที่แทนการ**หยดเหรียญ 10 บาท 1 เหรียญ**ต่อการกดหนึ่งครั้ง
- **ใช้ 7 Segment 2 หลัก** แสดงจำนวนเงิน ซึ่ง**ควรจะมีตัวเลขเพียง 5, 10, 15, 20 บาทเท่านั้น**
- **ใช้บัสเซอร์ & LED0** ส่งเสียง & ส่งแสงไฟกระพริบ แทนการปล่อยกระปองน้ำดีม

## A3 แยกงานใหญ่ๆ ออกให้เป็นงานย่อยๆ: จากงานข้างต้นเมื่อนำมาแยกเป็นงานย่อยๆ จะได้โครงสร้างโดยรวมของระบบแสดงดังรูปที่ 1



แต่ละระบบงานย่อยมีหน้าที่ทำอะไรบ้าง :

1. **Debounce** เนื่องจากเราใช้สวิตซ์แทนการหยดเหรียญ จำเป็นจะต้องป้องกันสวิตซ์ทำงานผิดพลาด (เราได้ทำเตรียมไว้แล้วในการทดลองที่ 8)
2. **7-Segment** วงจรแสดงผลตัวเลข (เราได้ทำเตรียมไว้แล้วในการทดลองที่ 8 )
3. **Buzzer และ LED** เป็นอุปกรณ์ที่มีบนบอร์ดทดลอง ใช้ส่งเสียง และส่งแสง
4. **One\_Shot** เนื่องจากธรรมชาติของการหยดเหรียญ แต่ละคนอาจจะหยอดช้า เร็ว ต่างกัน และต้องหยอดทีละ 1 เหรียญ จำเป็นจะต้องสร้างวงจรตรวจจับเพติกรอมนี้โดยปราศจากข้อจำกัดด้านเวลา การทำงานของ One\_Shot จะทำงานโดยอยู่ระหว่างๆ สวิตซ์มีการถูกกดหรือไม่ **ถ้าสวิตซ์ถูกกด** มันจะปล่อยเอ้าท์พุทเปลี่ยนจาก '0' ไปเป็น '1' (เกิดขوبข้ามเพียง 1 ขอบ) แล้วรอจนถึงหนึ่งคาบของสัญญาณนาฬิกา ก็จะเปลี่ยนกลับไปเป็น '0' เมื่อันเดิม **โดยที่ไม่สนใจว่าสวิตซ์จะถูกกดนานเท่าใดก่อนที่จะปล่อย** (อย่างไรก็กด 1 ครั้งเท่ากัน)

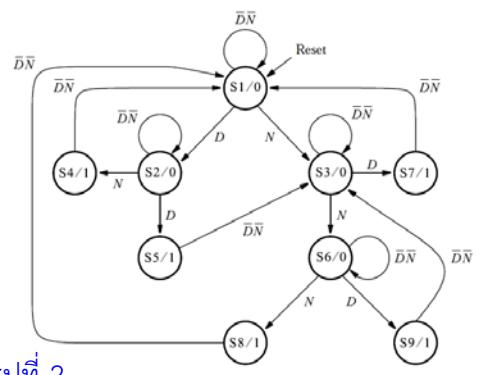


# การทดลองที่ 9 State Machines, it's Application

หน้า  
3 / 10

5. **State of Vending Machine** ระบบงาน (วงจร) ส่วนนี้ จะทำงานตามขั้นตอนที่ได้อธิบายไว้ในหน้าแรก (หัวข้อ A1) เมื่อเขียนเป็นแผนผังลำดับขั้นตอน หรือ state diagram จะได้ดังรูปที่ 2

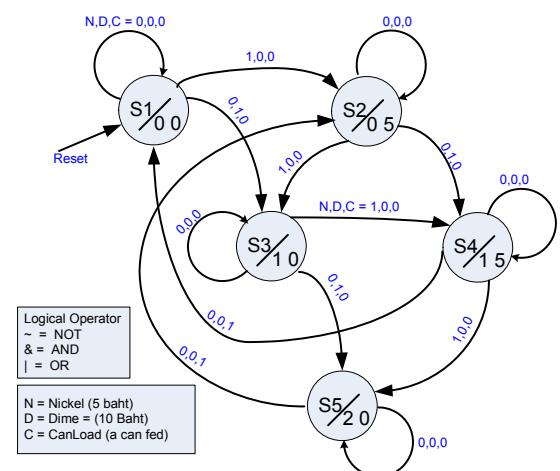
Example : ถ้าไม่มีการหยุดหรือยุบ ระบบจะรออยู่ที่สเตต S1 ที่ S1 ถ้าใส่เหรียญ 0.1 ระบบจะไปที่ S2 แต่ถ้าใส่เหรียญ 0.5 ระบบจะไปที่ S3 ปล่อยกระปอง และจะกดต่อไปที่ S1 เมื่อปล่อยกระปองเสร็จ ที่ S3 ถ้ามีการใส่เหรียญ 0.5 (เงินยังไม่ครบ) ระบบจะไปที่ S6 รอเงินเพิ่มอีก ...



รูปที่ 2

6. **Count, Show value of Coins** ระบบงานนี้จะทำการตรวจสอบเหรียญที่ถูกใส่เข้ามา ในทางปฏิบัติ จะได้จำนวนเป็น 0, 5, 10, 15, 20 บาท (ไม่เกิน 20 บาท เพราะเครื่องจะจ่ายกระปองให้ทันที) ดังนั้นการเพิ่มหรือลดของจำนวนเงินจึงมีรูปแบบที่ติดตัว ทำให้สามารถใช้ state diagram มาทำงานแทนวงจร Adder ได้ ซึ่งจะง่ายและสะดวกกว่ามาก ดังรูปที่ 3

รูปที่ 3



7. **Release cans Machine** เป็นระบบงานปล่อยกระปอง (ส่งเสียง & แสง) แต่ในชีวิตจริงเนื่องจากเป็นกลไกทางกล จะใช้เวลานานหลายวินาทีกว่าจะปล่อยกระปองเสร็จ ทำให้ระบบต้องหยุดรับเหรียญเพิ่ม ระบบนี้จึงต้องส่งสัญญาณไปสั่งครั้งเดียว (ให้ One\_shot หยุดทำงาน) ไว้ชั่วครู่เพื่อป้องกันข้อผิดพลาดในการทำงานไม่สัมพันธ์กันของระบบรับเหรียญกับระบบปล่อยกระปอง

## การทดลอง

### ขั้นที่ 1 เตรียมนำอุปกรณ์ที่เคยสร้างไว้จากการทดลอง 7 – 8 มาใช้งาน

#### คำสั่งการทดลอง

- ให้สร้างไฟล์เดอร์สำหรับเก็บงานขึ้นใหม่เพื่อเก็บงานในการทดลองนี้ชื่อ “Lab09\_State”
- นำงจต่างๆที่เคยสร้างไว้ในการทดลองที่ 5-8 (ทำการ copy ไฟล์ ดังรายชื่อด้านล่าง) มาไว้ในไฟล์เดอร์ที่สร้างขึ้นใหม่นี้
  - Debounce.qpf** และ **Debounce.bdf**
  - AsynchronousCounter.qpf** และ **AsynchronousCounter.bdf**
  - VHD\_7SEGM.qpf** และ **VHD\_7SEGM.vhd** (ให้ใช้วอร์ชันที่ปรับแก้ในการทดลองที่ 8)
  - ClockDivider.qpf** และ **ClockDivider.bdf**
- ให้ทำการเปิดโปรเจคไฟล์ตามข้อ 2 มาทำการ **คอมpile และสร้าง symbol file** ใหม่ทั้งหมด
- ให้ **ปิดโปรเจค** ที่ดำเนินการในขั้นตอนที่ 2-3 ก่อนที่จะทำงานต่อไป



## ขั้นที่ 2 สร้างระบบ State of Vending Machine เพิ่มขึ้นมาใช้งาน

5. สร้างอุปกรณ์ **State\_Deve** (ชื่อย่อของระบบงาน State of Vending Machines) โดยดำเนินการดังนี้

5.1 สร้างโปรเจคชื่อ “**State\_Deve**” ขึ้นมาและให้เก็บไว้ในไฟล์เดอร์เดิม

ใช้ชิป EP3C10E144C8

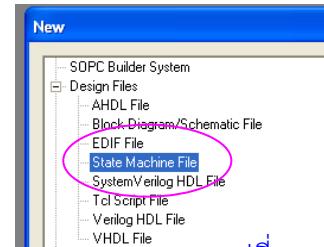
5.2 เปิดไฟล์สำหรับเก็บแผนภาพ **state diagram** โดยไปที่เมนู

**File > New**

เลือก **State Machine file** ดังรูปที่ 4 จะปรากฏหน้าต่าง

Editor ว่างๆ ของไฟล์ชื่อ **SM1.smf** ขึ้นมาพร้อมແບບเครื่อง

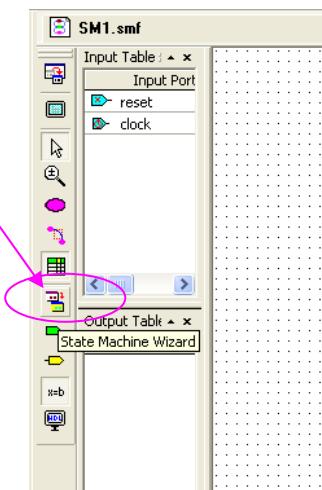
มีอ้างอิงของหน้าต่าง



รูปที่ 4

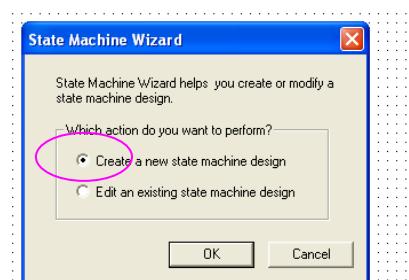
5.3 ให้เลือกใช้เครื่องมือเป็น **State Machine Wizard** ในรูปที่ 5

เราจะเริ่มด้วยการเขียน state table เป็นลำดับแรกแล้วจึงให้โปรแกรมทำการคอมไพล์เป็น state diagram ในภายหลัง

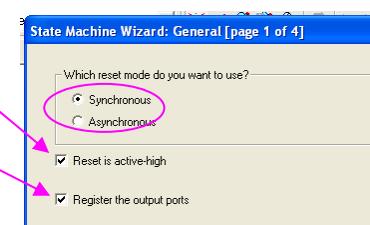


รูปที่ 5

เมื่อเลือก State Machine Wizard แล้วจะปรากฏหน้าต่างขึ้นมาเพื่อให้ยืนยันการออกแบบ ดังรูปที่ 6a



รูปที่ 6a



รูปที่ 6b

เมื่อเลือก **Create a ...** จะปรากฏหน้าต่าง **[Page 1 of 4]** สำหรับกำหนดคุณสมบัติทั่วไปดังรูปที่ 6b

- Synchronous กำหนดให้วงจรเปลี่ยน state โดยอาศัยสัญญาณนาฬิกา
- Reset in Active High กำหนดให้มีขาไว้สำหรับรีเซ็ต และทำการรีเซ็ตด้วยค่าลอจิก ‘1’
- Register the output port กำหนดให้สัญญาณเอ้าท์พุทออกจากตัวรีจิสเตอร์

กดปุ่ม **Next** เพื่อทำขั้นต่อไป

5.4 ที่หน้าต่าง **[Page 2 of 4]** จะเป็นการกำหนด ส่วนประกอบของ state ดังรูปที่ 7 ซึ่งประกอบด้วย

- **ชื่อ State** และจำนวนของ State
- **สัญญาณอินพุท**ที่ใช้ ควบคุมการเปลี่ยน state
- **ตารางแสดง state table** และเงื่อนไขการเปลี่ยน state (**State Transition**)

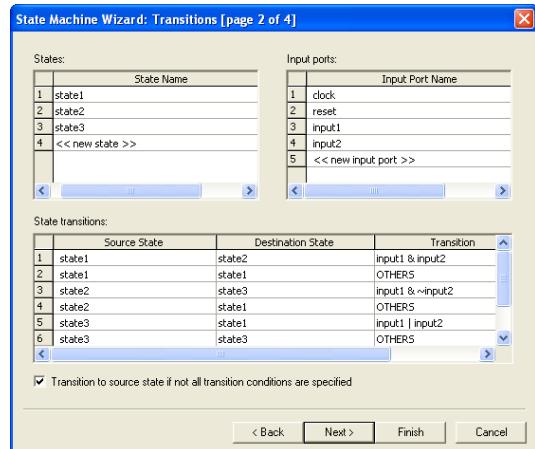


# การทดลองที่ 9 State Machines, it's Application

หน้า  
5 / 10

หมายเหตุ เครื่องหมายของ Operator  
หรือตัวกระทำทางลοจิกพื้นฐานสำหรับใช้  
ในการเขียน state มีอยู่ 4 ตัวเท่านั้นคือ

$\sim$	NOT
$\&$	AND
$ $	OR
OTHERS	-----



5.5 ให้เขียน(ปรับแก้) ชื่อ ของ state ในตาราง [Page 2 of 4]

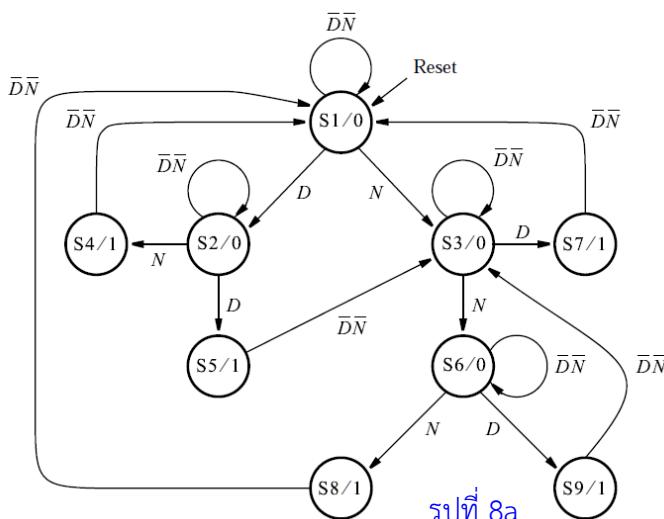
รูปที่ 7

ให้มีจำนวน state = 9 state ชื่อ S1... S9 ตามลำดับดังรูปที่ 8b ด้วยการปรับแก้ตามรูปที่ 8c  
โดยเปลี่ยนชื่อ(กดดับเบิลคลิก) จากเดิม **State 1** แก้เป็น **S1** (ตามตาราง state table ในรูปที่ 8b)

**State 2** แก้เป็น **S2**

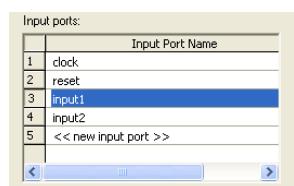
...

**State 9** แก้เป็น **S9**



Present state	Next state				Output z
	DN = 00	01	10	11	
S1	S1	S3	S2	-	0
S2	S2	S4	S5	-	0
S3	S3	S6	S7	-	0
S4	S1	-	-	-	1
S5	S3	-	-	-	1
S6	S6	S8	S9	-	0
S7	S1	-	-	-	1
S8	S1	-	-	-	1
S9	S3	-	-	-	1

รูปที่ 8b



รูปที่ 8d



รูปที่ 8c

จากนั้นกำหนดอินพุตพอร์ทในรูปที่ 8d ที่จะทำหน้าที่ควบคุมการเปลี่ยนของ state โดยใช้  
เปลี่ยนชื่อพอร์ทจาก **input1** (ของเดิม) เป็น **D** (ตามตาราง state table ของเรา)  
เปลี่ยนชื่อพอร์ทจาก **input2** (ของเดิม) เป็น **N** (ตามตาราง state table ของเรา)



# การทดลองที่ 9 State Machines, it's Application

หน้า  
6 / 10

5.6 เปลี่ยนค่าในตาราง state transition ของรูปที่ 9 โดยเปลี่ยนไปใช้ข้อมูลจากรูปที่ 8b หัวข้อของ state , เงื่อนไขการเปลี่ยน state (state transition) จนครบทุก state ก็กด Next เพื่อไปหน้าต่อ [Page 3 of 4]

State transitions:			
Source State	Destination State	Transition	
1 S1	S2	input1 & input2	
2 S1	S1	OTHERS	
3 S2	S2	input1 & ~input2	
4 S2	S3	OTHERS	
5 S3	S4	input1   input2	
6 S3	S5	OTHERS	
7 S3	S6	<< new transition >>	
	S7		
	S8		
	S9		

Transition to source state is not yet specified. All transition conditions are specified.

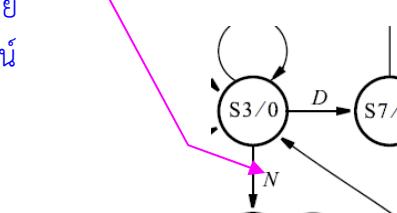
รูปที่ 9

## ข้อแนะนำในการเขียน State Transition ใน [Page 2 of 4]

- ก) จากรูปที่ 8a จะมีบาง transition ที่มี exciting input เพียงตัวแปรเดียวเช่น N ในรูปนี้ความหมายคือ  $N = '1'$  และไม่มีการล่าwiększ D เลย แต่เราจะต้องเขียนให้ครบทั้งสองอินพุตโดยแสดงเป็นนิพจน์ ที่มีครบทั้งสองตัวแปรคือ D และ N โดยเขียนเป็น

$N \& \sim D$

Transition
$N \& \sim D$
OTHERS



การเขียนเช่นนี้จะหมายความว่า  $N = '1'$

ตรงตามการกำหนดจาก state diagram ส่วน D ในที่นี่กำหนดเป็น  $\sim D$  ซึ่งมีค่าเป็น  $D = '0'$  นั่นเอง

- ข) ในบางกรณี ที่มี เงื่อนไขอื่นๆ ที่ไม่ใช่ดังในเงื่อนไขที่มีค่าเดียว เราสามารถใช้คำว่า OTHERS แทนการบอกว่าเป็นเงื่อนไขอื่นๆ ที่ไม่น้อยกว่าหนึ่งเงื่อนไข มาเป็นเงื่อนไขทางเลือกได้

6. กำหนดค่าของอินพุต [Page 3 of 4] ดังรูปที่ 10 โดยชื่อของพอร์ตอินพุตให้ใช้ชื่อเป็น Z และกำหนดค่าของ Z ที่จะได้ที่ state ต่างๆ (นำค่ามาจากรูปที่ 8b)

ปรับแก้ให้ถูกต้องตามตารางในรูปที่ 8b

Output ports:			
	Output Port Name		
1	Z		
2	<< new output port >>		

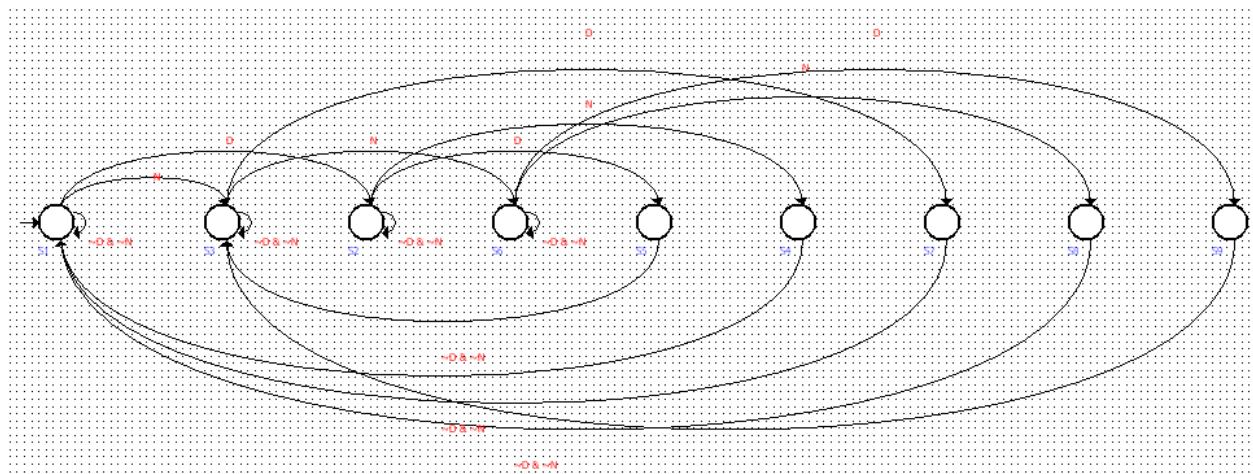
  

Action conditions:				
Output Port	Output Value	In State	Additional Conditions	
4	1	S4	<< condition >>	
5	1	S5	<< condition >>	
6	0	S6	<< condition >>	
7	1	S7	<< condition >>	
8	1	S8	<< condition >>	
9	1	S9	<< condition >>	
10	<< output value >>		<< condition >>	

รูปที่ 10



7. เมื่อเสร็จครบทุกขั้นตอนแล้ว หากไม่มีขั้นตอนใดผิดพลาด โปรแกรมก็จะทำการสร้างรูปแบบของ state Diagram ขึ้นมาให้ดังแสดงในรูปที่ 11 ทำการบันทึกไฟล์เพื่อเตรียมทำขั้นตอนต่อไป



รูปที่ 11

หมายเหตุ: รูปที่ปรากฏอยู่นี้อาจจะไม่ตรงกับรูปที่ นศ. ทำในห้องปฏิบัติการ

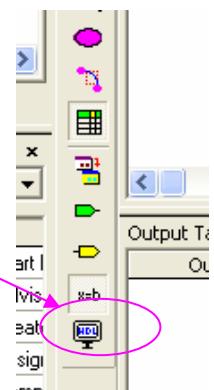
8. ทำการแปลงไฟล์ของ state diagram ที่ได้จากในขั้นตอนที่ 7 ให้เป็นไฟล์แบบ

ภาษา VHDL โดยไปที่ແຕບເຄື່ອນໄຫວ້າດ້ານຊ້າຍຂອງໜ້າຕ່າງ ແລະເລືອກປຸ່ມ

Generate VHDL File      ดังรูปที่ 12

เมื่อได้ไฟล์แบบ VHDL ແລ້ວໃໝ່**ทำการຄອມໄພລ໌**ຕາມປັກຕິ  
ແລະ**ทำการສ້າງ symbol file** ຂອງຈະຈົນມາ ເພື່ອເຕີມ  
ໃໝ່ໃນขັ້ນຕອນດັ່ງໄປ

รูปที่ 12



9. ทำการປຶດໂປຣເຈກທີ່ສ້າງມາໃນขັ້ນຕອນທີ 5- 8 ກ່ອນທີ່ຈະດຳເນີນການຕ່ອໄປ

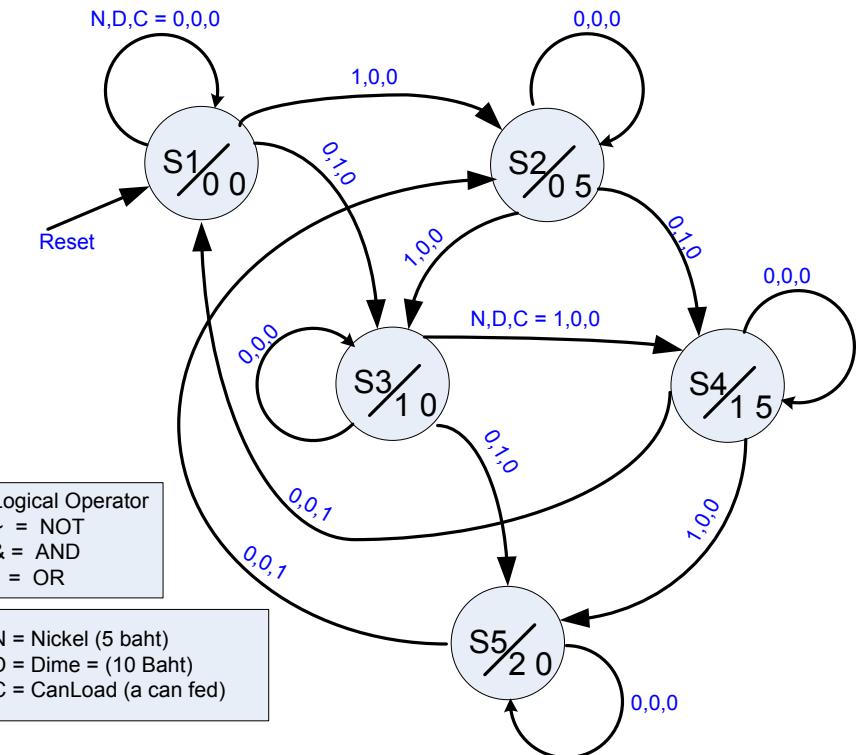
## ขັ້ນທີ 3 ສ້າງຮະບບ Count\_Unit ແລະຮະບບ Count8to1 ເພີ່ມຈະຈົນໃຊ້ຈານ

10. ໃ້າໃຈໃນขັ້ນຕອນທີ 5 – 9 ມາທຳການສ້າງອຸປະນົດຕ້ອໄປນີ້

- ຮະບບງານ **Count\_Unit** ດັ່ງ state diagram ໃນຮູບທີ 13

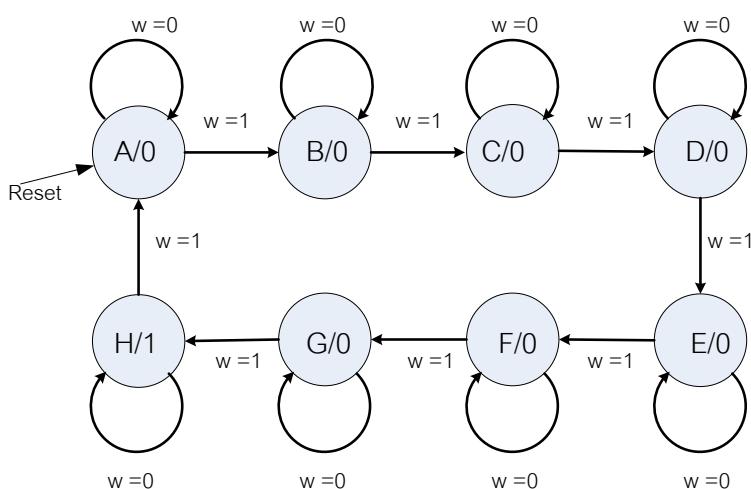
- ຮະບບງານ **Count8to1** (ອູ່ງຍາຍໃນຮະບບງານ Hold5Sec ອີກຈັ້ນໜຶ່ງ) ດັ່ງ state diagram ໃນຮູບທີ 14

(ໃຫ້ສຶກຂາເພີ່ມເຕີມໄດ້ຈາກໜັງສື່ວເຮີນ ວິຊາດິຈິທັກ : Fundamentals of Digital logic with VHDL Design 3<sup>rd</sup>, Stephen Brown, Z. Vranesic)



รูปที่ 13

ระบบงานชื่อ Count\_Unit



รูปที่ 14

ระบบงานชื่อ Count8to1



## ขั้นที่ 4 สร้างระบบ One\_Shot และ Hold5Sec เพิ่มขึ้นมาใช้งาน

11. ทำการปิดโปรเจคที่สร้างมาในขั้นตอนที่ 10 ก่อนที่จะดำเนินการต่อไป
12. สร้างอุปกรณ์ระบบ **One\_Shot** ด้วยภาษา VHDL ดังในรูปที่ 15 และสร้าง **symbol file** ของวงจรขึ้นมาเพื่อเตรียมไว้ใช้ในขั้นถัดไป

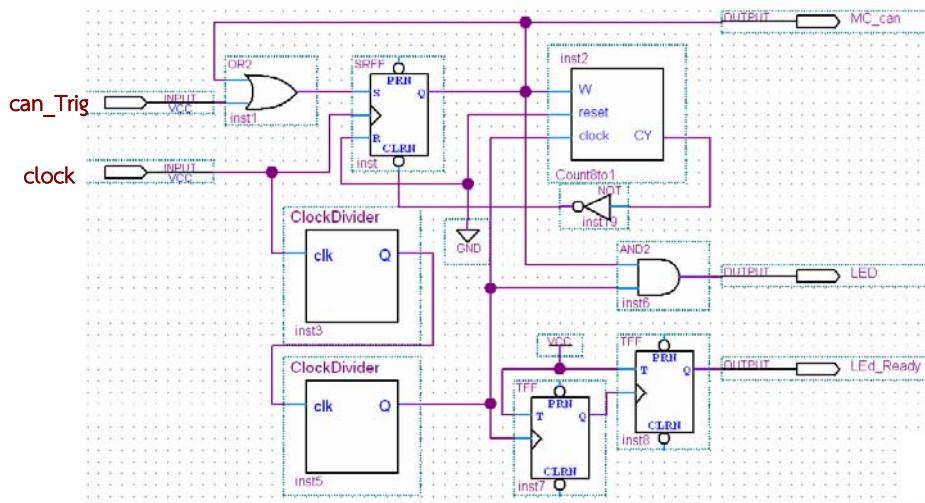
```

1 LIBRARY ieee ;
2 USE ieee.std_logic_1164.all ;
3
4
5 ENTITY One_Shot IS
6   PORT ( Clock, SW, Enable : IN STD_LOGIC ;
7           Q_Shot : OUT STD_LOGIC ) ;
8 END One_Shot ;
9
10 ARCHITECTURE Behavior OF One_Shot IS
11   SIGNAL Temp : STD_LOGIC ;
12   SIGNAL Status : STD_LOGIC ;
13
14 BEGIN
15   PROCESS ( Clock, Temp, Status )
16   BEGIN
17     IF Enable = '0' THEN
18       Temp <= '0';
19       Status <= '0';
20     ELSIF (Clock'EVENT AND Clock = '1') THEN
21       IF SW = '1' THEN
22         IF Temp = '1' THEN
23           Status <= '0';
24         ELSE
25           Status <= '1';
26           Temp <= '1';
27         END IF;
28       ELSE
29         Status <= '0';
30         Temp <= '0';
31       END IF;
32     END IF;
33   END PROCESS ;
34   Q_Shot <= Status ;
35 END Behavior ;

```

13. ทำการปิดโปรเจคที่สร้างมาในขั้นตอนที่ 12 ก่อนที่จะดำเนินการต่อไป
14. สร้างอุปกรณ์ระบบ **Hold5Sec** ดังในรูปที่ 16 และสร้าง **symbol file** ขึ้นมาเพื่อเตรียมไว้ใช้ในขั้นถัดไป

รูปที่ 15



รูปที่ 16

## ขั้นที่ 5 ประกอบงานทุกส่วนเข้าเป็นระบบของ Vending Machine ที่สมบูรณ์

15. ทำการปิดโปรเจคที่สร้างมาในขั้นตอนที่ 14 ก่อนที่จะดำเนินการต่อไป
16. สร้างโปรเจคชื่อ “VendingMC” ขึ้นมาและให้เก็บไว้ในโฟเดอร์เดียวกันกับโปรเจคที่สร้างตอนก่อนหน้า
  - a) จากนั้นสร้างวงจรทดลองดังรูปที่ 17 ให้ใช้ชิปเบอร์ EP3C10E144C8 และทำการคอมไพล์

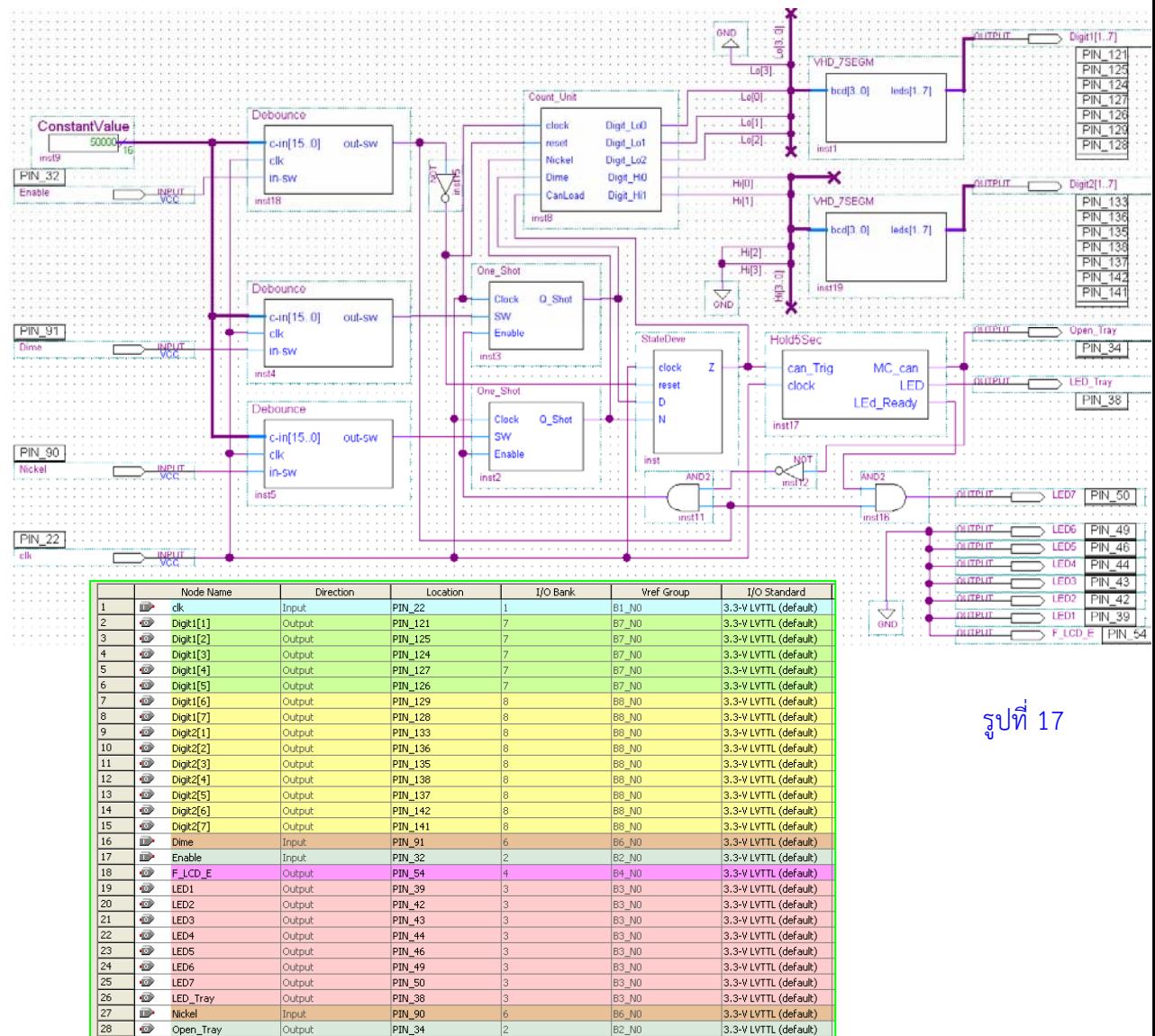


# การทดลองที่ 9 State Machines, it's Application

หน้า  
10 / 10

b) กำหนดขา PIN ตามรูป คอมแพล์ซึ่อกรอบ และทำการโปรแกรม configuration ลงบอร์ดทดลอง

## 17. ทำการทดสอบการทำงานของระบบ



รูปที่ 17

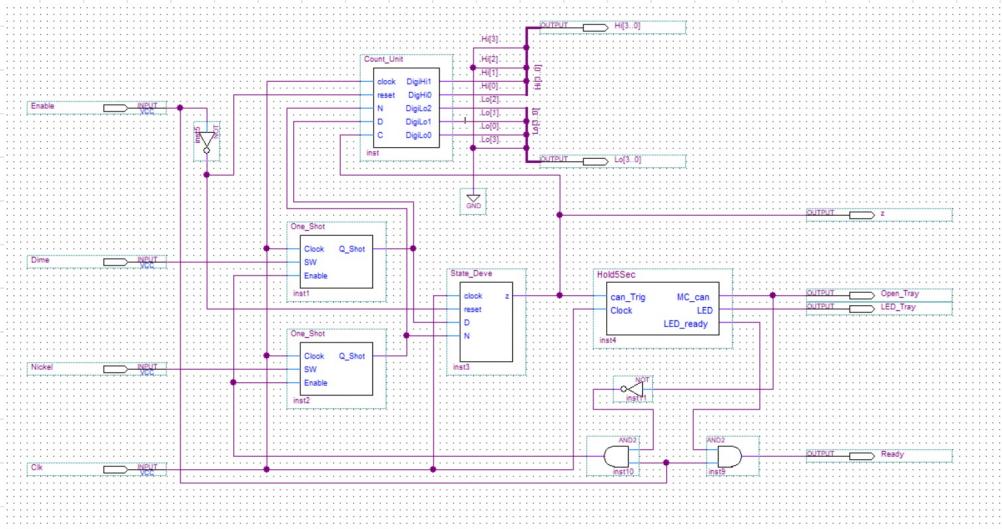
### งานมอบหมายท้ายการทดลอง

(ให้เขียนลงบนกระดาษ A4 ส่งในคราวถัดไป)

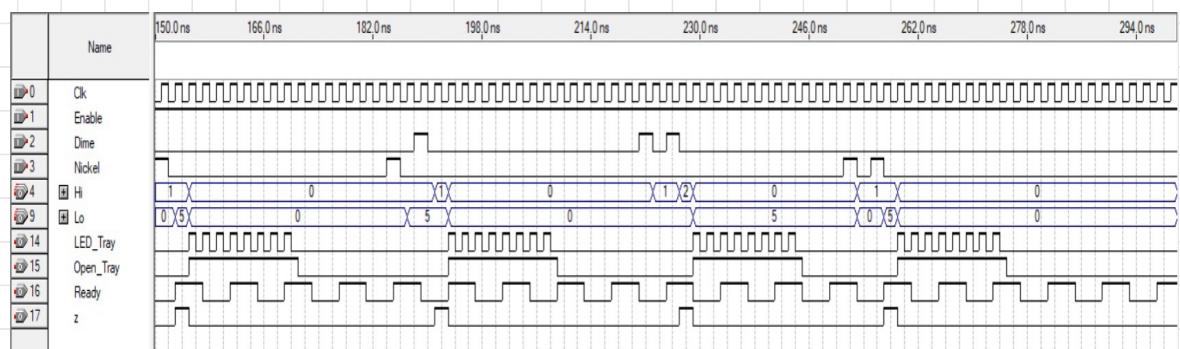
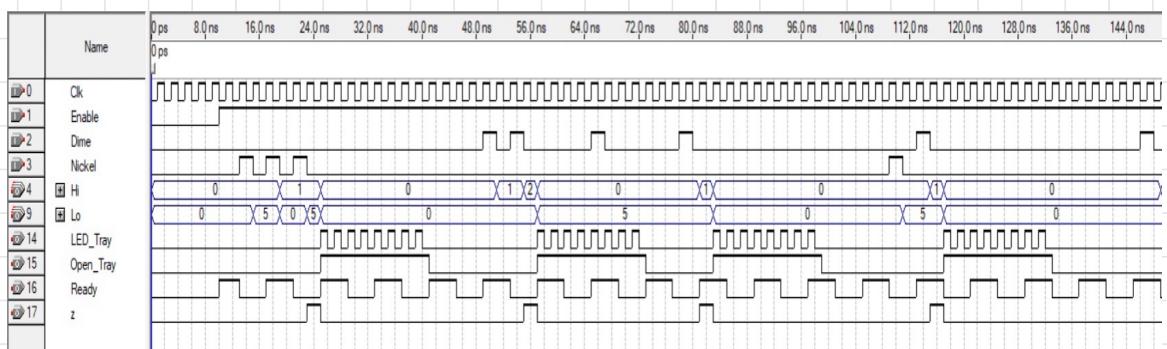
- ทำการทดสอบการทำงานของระบบให้เข้าใจอย่างลึกซึ้ง (โดยอาศัยแนวคิดจากการออกแบบ, หัวข้อ A1 – A3) เมื่อมีความเข้าใจระบบอย่างดีแล้วให้ น.ศ. อธิบายวิธีการทดสอบอย่างเป็นขั้นตอน จนทำให้อาจารย์ผู้ตัวจริงสามารถสัมผัสได้ว่า น.ศ. มีความรู้จริงในงานที่ทำมาทั้งหมดนี้ โดยเขียนเป็นรายงานส่งหลังทำการทดลองเสร็จสิ้น
- ระบบในรูปที่ 8a , รูปที่ 13 , และรูปที่ 14 สามารถที่จะออกแบบโดยวิธีอื่นๆ โดยที่ไม่ใช้วิธีการทำ Finite State Machine ได้หรือไม่ ? ถ้าได้ จะต้องทำอย่างไร ทำรายงานส่งด้วย? ถ้าไม่ได้ ให้อธิบายสาเหตุ ทั้งนี้ น.ศ. จะต้องอธิบายจนทำให้อาจารย์ผู้ตัวจริงสามารถสัมผัสได้ว่า น.ศ. มีความรู้จริงในงานที่ทำมาทั้งหมดนี้

ประโยชน์ ... ทำให้อาจารย์ผู้ตัวจริงสามารถสัมผัสได้ว่า ... Cr. NCH, : established in 2014

## Block Diagram



# warm Simulation

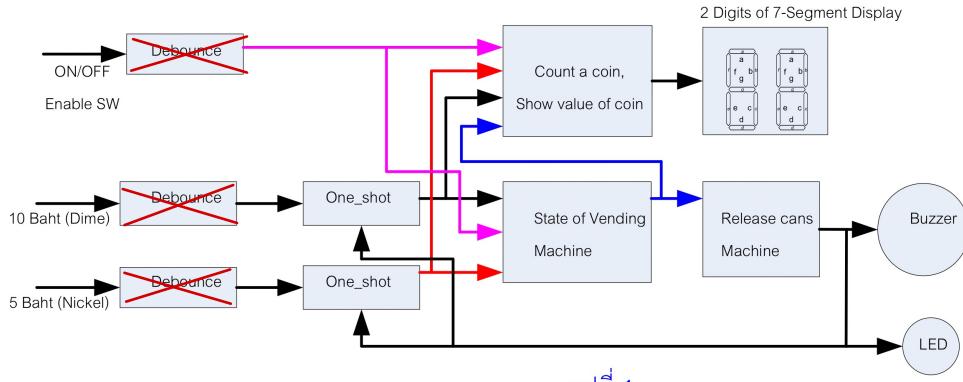


# ค่ารากวิถีของบ

(ให้เขียนลงบนกระดาษ A4 ส่งในคราวถัดไป)

- ทำการทดสอบการทำงานของระบบให้เข้าใจอย่างลึกซึ้ง (โดยอาศัยแนวคิดจากการออกแบบ, หัวข้อ A1 –A3)

เมื่อมีความเข้าใจระบบอย่างดีแล้วให้ น.ศ. อธิบายวิธีการทดสอบอย่างเป็นขั้นตอน จนทำให้อาจารย์ผู้ตรวจสอบสามารถสัมผัสด้วยว่า น.ศ. มีความรู้จริงในการที่ทำมาทั้งหมดนี้ โดยเขียนเป็นรายงานส่งหลังทำการทดลองเสร็จสิ้น



รายที่ 1

① วิเคราะห์การทำงานเมื่อ  $En = 0$  ค่าลูกปัดเป็น `0` ผู้ใช้งาน En ควบคุมการทำงานทั่วไป (หันเข้าสู่ปุ่มเปิดปิด ON-OFF)

② กรณีที่  $En = 1$  มีการนำเข้า Input Dime หรือ Nickel (ชนิดเงินๆ) เครื่องจะทำงาน

③ กรณีที่  $En = 1$  เมื่อทำการป้อน Input Dime หรือ Nickel จะต้อง One\_Shot

④ One\_Shot เป็นส่วนที่ไม่ได้เครื่องรับเงิน แต่เป็นส่วนที่เครื่องต้องรับเงิน สำหรับการเปลี่ยนเงิน เครื่องจะรับเงิน

⑤ State of Vending Machine เป็นส่วนสำคัญ ที่ต้องรับเงิน ส่วนของห้องน้ำ เว้นที่ช่องด้วย สำคัญของห้องน้ำนั้นเป็นไปตาม Design ที่ออกแบบไว้ ให้กับห้องน้ำ

⑥ Count a coin หมายความว่าเป็นจำนวนที่นั่ง เว้นที่ช่องด้วย ให้แสดงผลไปยัง 7-seg เพื่อแจ้งว่ามีจำนวนเงิน เก็บรวมกันเท่าไร

⑦ Release cam's Machine เก็บส่วนหัวที่ต้องบันค้างไว้ซึ่งจะเป็น State of Vending Machine ทั้งหมด Output ของเครื่องจะอยู่ในไฟ LED

⑧ Buzzer & LED เก็บส่วนแสดงผล ผู้ซื้อเงินค่าพิเศษ ให้เมื่อเข้ามา

2. ระบบในรูปที่ 8a , รูปที่ 13 , และรูปที่ 14 สามารถที่จะออกแบบโดยวิธีอื่นๆ โดยที่ไม่ใช้วิธีการทำ Finite State Machine ได้หรือไม่ ? ถ้าได้ จะต้องทำอย่างไร ทำรายงานส่งด้วยว่า ถ้ามีได้ ให้อธิบายเหตุที่ ห้ามนี้ น.ศ. จะต้องอธิบายจนทำให้อาจารย์ผู้ตรวจสอบสามารถสัมผัสด้วยว่า น.ศ. มีความรู้จริงในงานที่ทำมาทั้งหมดนี้

กรุณาอ่าน State\_Deve เก็บส่วนหัวที่ต้องบันค้างไว้ซึ่งจะเป็น Finite State Machine แบบ VHDL ดังนี้

```

18 LIBRARY ieee;
19 USE ieee.std_logic_1164.all;
20
21 ENTITY State_Deve IS
22 PORT (
23  reset : IN STD_LOGIC;
24  D : IN STD_LOGIC := '0';
25  N : IN STD_LOGIC := '0';
26  z : OUT STD_LOGIC
27 );
28 END State_Deve;
29
30
31 ARCHITECTURE BEHAVIOR OF State_Deve IS
32 TYPE type_fstate IS (S1,S2,S3,S4,S5,S6,S7,S8,S9);
33 SIGNAL fstate : type_fstate;
34 SIGNAL req_fstate : type_fstate;
35 SIGNAL req_z : STD_LOGIC := '0';
36
37 BEGIN
38 PROCESS (clock,req_fstate)
39 BEGIN
40  IF (clock='1') AND clock'event THEN
41   fstate <= req_fstate;
42  END IF;
43 END PROCESS;
44
45 PROCESS (fstate,reset,D,N,req_z)
46 BEGIN
47  IF (reset='1') THEN
48   req_z <= S1;
49   z <= '0';
50  ELSE
51   req_z <= '0';
52   z <= '0';
53  CASE fstate IS
54  WHEN S1 >
55   IF ((NOT(D = '1')) AND NOT((N = '1'))) THEN
56    req_fstate := S1;
57   ELSIF ((D = '1') AND NOT((N = '1'))) THEN
58    req_fstate := S2;
59   ELSIF ((NOT(D = '1')) AND (N = '1')) THEN
60    req_fstate := S3;
61
62   -- Inserting 'else' block to prevent latch inference
63   ELSE
64    req_fstate := S1;
65   END IF;
66
67  WHEN S2 >
68   IF ((NOT(D = '1')) AND NOT((N = '1'))) THEN
69    req_fstate := S2;
70   ELSIF ((NOT(D = '1')) AND (N = '1')) THEN
71    req_fstate := S4;
72   ELSIF ((D = '1') AND NOT((N = '1'))) THEN
73    req_fstate := S5;
74
75   -- Inserting 'else' block to prevent latch inference
76   ELSE
77    req_fstate := S2;
78   END IF;
79
80  WHEN S3 >
81   IF ((NOT(D = '1')) AND NOT((N = '1'))) THEN
82    req_fstate := S3;
83   ELSIF ((NOT(D = '1')) AND (N = '1')) THEN
84    req_fstate := S6;
85   ELSIF ((D = '1') AND NOT((N = '1'))) THEN
86    req_fstate := S7;
87
88   -- Inserting 'else' block to prevent latch inference
89   ELSE
90    req_fstate := S3;
91   END IF;
92
93  WHEN S4 >
94   IF ((NOT(D = '1')) AND NOT((N = '1'))) THEN
95    req_fstate := S4;
96
97   -- Inserting 'else' block to prevent latch inference
98   ELSE
99    req_fstate := S4;
100  END IF;
101
102  WHEN S5 >
103   IF ((NOT(D = '1')) AND NOT((N = '1'))) THEN
104    req_fstate := S5;
105
106   -- Inserting 'else' block to prevent latch inference
107   ELSE
108    req_fstate <= S5;
109  END IF;
110
111
112  req_z <= '1';
113
114  WHEN S6 =>
115   IF ((NOT(D = '1')) AND NOT((N = '1'))) THEN
116    req_fstate <= S6;
117   ELSIF ((NOT(D = '1')) AND (N = '1')) THEN
118    req_fstate <= S8;
119   ELSIF ((D = '1') AND NOT((N = '1'))) THEN
120    req_fstate <= S9;
121
122   -- Inserting 'else' block to prevent latch inference
123   ELSE
124    req_fstate <= S6;
125  END IF;
126
127  req_z <= '0';
128
129  WHEN S7 =>
130   IF ((NOT(D = '1')) AND NOT((N = '1'))) THEN
131    req_fstate <= S7;
132
133   -- Inserting 'else' block to prevent latch inference
134   ELSE
135    req_fstate <= S7;
136  END IF;
137
138  req_z <= '1';
139
140  WHEN S8 =>
141   IF ((NOT(D = '1')) AND NOT((N = '1'))) THEN
142    req_fstate <= S8;
143
144   -- Inserting 'else' block to prevent latch inference
145   ELSE
146    req_fstate <= S8;
147  END IF;
148
149  req_z <= '1';
150
151  WHEN S9 =>
152   IF ((NOT(D = '1')) AND NOT((N = '1'))) THEN
153    req_fstate <= S9;
154
155   -- Inserting 'else' block to prevent latch inference
156   ELSE
157    req_fstate <= S9;
158  END IF;
159
160
161  req_z <= '1';
162  WHEN OTHERS =>
163   req_z <= 'X';
164
165  -- Inserting 'Reach undefined state';
166
167  END CASE;
168  z <= req_z;
169
170  END IF;
171 END PROCESS;
172 END BEHAVIOR;

```

(3)

(4)

4. เขียนโค้ด ฝึกความแม่นยำ หาข้อบกพร่อง  
1 กด 4 ทิศที่มีผลลัพธ์แตกต่างกัน

# 13 սեղ Count\_Unit

## 6th Semester State Dev VLSI Design Lab

(3)

```

18 LIBRARY ieee;
19 USE ieee.std_logic_1164.all;
20
21 ENTITY Count_Unit IS
22 PORT
23   clock : IN STD_LOGIC;
24   reset : IN STD_LOGIC := '0';
25   N : IN STD_LOGIC := '0';
26   D : IN STD_LOGIC := '0';
27   C : IN STD_LOGIC := '0';
28   DigiHil : OUT STD_LOGIC;
29   DigiLo0 : OUT STD_LOGIC;
30   DigiLo1 : OUT STD_LOGIC;
31   DigiLo2 : OUT STD_LOGIC;
32   DigiLo3 : OUT STD_LOGIC;
33 );
34 END Count_Unit;
35
36 ARCHITECTURE BEHAVIOR OF Count_Unit IS
37 TYPE type_fstate IS (S1,S2,S3,S4,S5);
38 SIGNAL fstate : type_fstate;
39 SIGNAL reg_fstate : type_fstate;
40 SIGNAL reg_DigiHil : STD_LOGIC := '0';
41 SIGNAL reg_DigiLo0 : STD_LOGIC := '0';
42 SIGNAL reg_DigiLo1 : STD_LOGIC := '0';
43 SIGNAL reg_DigiLo2 : STD_LOGIC := '0';
44 SIGNAL reg_DigiLo3 : STD_LOGIC := '0';
45 BEGIN
46   PROCESS (clock,req_fstate)
47   BEGIN
48     IF (clock='1' AND clock'event) THEN
49       fstate <= req_fstate;
50     END IF;
51   END PROCESS;
52
53   PROCESS (fstate,reset,N,D,C,reg_DigiHil,reg_DigiLo0,reg_DigiLo1,reg_DigiLo2,reg_DigiLo3)
54 BEGIN
55   IF (reset='1') THEN
56     req_fstate <= S1;
57     reg_DigiHil <= '0';
58     reg_DigiLo0 <= '0';
59     reg_DigiLo1 <= '0';
60     reg_DigiLo2 <= '0';
61     reg_DigiLo3 <= '0';
62     DigiHil <= '0';
63     DigiLo0 <= '0';
64     DigiLo1 <= '0';
65     DigiLo2 <= '0';
66     DigiLo3 <= '0';
67   ELSE
68     reg_DigiHil <= '0';
69     reg_DigiLo0 <= '0';
70     reg_DigiLo1 <= '0';
71     reg_DigiLo2 <= '0';
72     reg_DigiLo3 <= '0';
73     DigiHil <= '0';
74     DigiLo0 <= '0';
75     DigiLo1 <= '0';
76     DigiLo2 <= '0';
77     DigiLo3 <= '0';
78   CASE fstate IS
79     WHEN S1 =>
80       IF (((NOT(N = '1')) AND NOT((D = '1'))) AND NOT((C = '1'))) THEN
81         reg_fstate <= S1;
82       ELSIF (((N = '1') AND NOT(D = '1')) AND NOT(C = '1')) THEN
83         reg_fstate <= S2;
84       ELSIF (((N = '1') AND (D = '1')) AND NOT(C = '1')) THEN
85         reg_fstate <= S3;
86       -- Inserting 'else' block to prevent latch inference
87     ELSE
88       reg_fstate <= S1;
89     END IF;
90
91     reg_DigiHil <= '0';
92     reg_DigiLo2 <= '0';
93     reg_DigiLo3 <= '0';
94
95     reg_DigiLo0 <= '0';
96     reg_DigiLo1 <= '0';
97
98     reg_DigiHil <= '0';
99     reg_DigiLo0 <= '0';
100    WHEN S2 =>
101      IF (((NOT(N = '1')) AND NOT((D = '1'))) AND NOT((C = '1'))) THEN
102        reg_fstate <= S2;
103      ELSIF (((N = '1') AND NOT(D = '1')) AND NOT(C = '1')) THEN
104        reg_fstate <= S3;
105      ELSIF (((NOT(N = '1')) AND (D = '1')) AND NOT(C = '1')) THEN
106        reg_fstate <= S4;
107
108      reg_DigiHil <= '0';
109      reg_DigiLo0 <= '0';
110      reg_DigiLo1 <= '0';
111      reg_DigiLo2 <= '0';
112      reg_DigiLo3 <= '0';
113
114      reg_DigiHil <= '0';
115      reg_DigiLo0 <= '0';
116      reg_DigiLo1 <= '0';
117      reg_DigiLo2 <= '0';
118
119      reg_DigiHil <= '0';
120      reg_DigiLo0 <= '0';
121      reg_DigiLo1 <= '0';
122      reg_DigiLo2 <= '0';
123
124      reg_DigiHil <= '0';
125      reg_DigiLo0 <= '0';
126      reg_DigiLo1 <= '0';
127      reg_DigiLo2 <= '0';
128
129      reg_DigiHil <= '0';
130      reg_DigiLo0 <= '0';
131      reg_DigiLo1 <= '0';
132      reg_DigiLo2 <= '0';
133
134      reg_DigiHil <= '0';
135      reg_DigiLo0 <= '0';
136      reg_DigiLo1 <= '0';
137      reg_DigiLo2 <= '0';
138
139      reg_DigiHil <= '0';
140      reg_DigiLo0 <= '0';
141      reg_DigiLo1 <= '0';
142      reg_DigiLo2 <= '0';
143
144      reg_DigiHil <= '0';
145      reg_DigiLo0 <= '0';
146      reg_DigiLo1 <= '0';
147      reg_DigiLo2 <= '0';
148
149      reg_DigiHil <= '0';
150      reg_DigiLo0 <= '0';
151      reg_DigiLo1 <= '0';
152      reg_DigiLo2 <= '0';
153
154      reg_DigiHil <= '0';
155      reg_DigiLo0 <= '0';
156      reg_DigiLo1 <= '0';
157      reg_DigiLo2 <= '0';
158
159      reg_DigiHil <= '0';
160      reg_DigiLo0 <= '0';
161      reg_DigiLo1 <= '0';
162      reg_DigiLo2 <= '0';
163
164      reg_DigiHil <= '0';
165      reg_DigiLo0 <= '0';
166
167      reg_DigiHil <= '0';
168      reg_DigiLo0 <= '0';
169
170      reg_DigiHil <= '0';
171
172      reg_DigiHil <= '0';
173      reg_DigiLo0 <= '0';
174
175      reg_DigiHil <= '0';
176
177      reg_DigiHil <= '0';
178
179      reg_DigiHil <= '0';
180
181      reg_DigiHil <= '0';
182
183      reg_DigiHil <= 'X';
184      reg_DigiLo0 <= 'X';
185      reg_DigiLo1 <= 'X';
186      reg_DigiLo2 <= 'X';
187      reg_DigiLo3 <= 'X';
188
189      reg_DigiHil <= '0';
190
191      DigiHil <= reg_DigiHil;
192      DigiLo0 <= reg_DigiLo0;
193      DigiLo1 <= reg_DigiLo1;
194      DigiLo2 <= reg_DigiLo2;
195
196   END PROCESS;
197
198 END BEHAVIOR;

```

(1)

(2)

(4)

# វគ្គទី 14 សម្រាប់ Count8to1

ការរកដើម្បីអនុវត្តន៍ Vhdl ដើម្បីផ្តល់ទម្រង់

```

18 LIBRARY ieee;
19 USE ieee.std_logic_1164.all;
20
21 ENTITY Count8to1 IS
22 PORT (
23     clock : IN STD_LOGIC;
24     reset : IN STD_LOGIC := '0';
25     w : IN STD_LOGIC := '0';
26     CY : OUT STD_LOGIC
27 );
28 END Count8to1;
29
30 ARCHITECTURE BEHAVIOR OF Count8to1 IS
31 TYPE type_fstate IS (A,B,C,D,E,F,G,H);
32 SIGNAL fstate : type_fstate;
33 SIGNAL reg_fstate : type_fstate;
34 SIGNAL reg_CY : STD_LOGIC := '0';
35 BEGIN
36     PROCESS (clock,reg_fstate,reg_CY)
37 BEGIN
38         IF (clock='1' AND clock'event) THEN
39             fstate <= reg_fstate;
40             CY <= reg_CY;
41         END IF;
42     END PROCESS;
43
44     PROCESS (fstate,reset,w)
45 BEGIN
46         IF (reset='1') THEN
47             reg_fstate <= A;
48             reg_CY <= '0';
49         ELSE
50             reg_CY <= '0';
51             CASE fstate IS
52                 WHEN A =>
53                     IF (NOT((w = '1'))) THEN
54                         reg_fstate <= A;
55                     ELSIF ((w = '1')) THEN
56                         reg_fstate <= B;
57                         -- Inserting 'else' block to prevent latch inference
58                     ELSE
59                         reg_fstate <= A;
60                     END IF;
61
62                     reg_CY <= '0';
63                     WHEN B =>
64                     IF (NOT((w = '1'))) THEN
65                         reg_fstate <= B;
66                     ELSIF ((w = '1')) THEN
67                         reg_fstate <= C;
68                         -- Inserting 'else' block to prevent latch inference
69                     ELSE
70                         reg_fstate <= B;
71                     END IF;
72
73                     reg_CY <= '0';
74                     WHEN C =>
75                     IF (NOT((w = '1'))) THEN
76                         reg_fstate <= C;
77                     ELSIF ((w = '1')) THEN
78                         reg_fstate <= D;
79                         -- Inserting 'else' block to prevent latch inference
80                     ELSE
81                         reg_fstate <= C;
82                     END IF;
83
84                     reg_CY <= '0';
85                     WHEN D =>
86                     IF (NOT((w = '1'))) THEN
87                         reg_fstate <= D;
88                     ELSIF ((w = '1')) THEN
89                         reg_fstate <= E;
90                         -- Inserting 'else' block to prevent latch inference
91                     ELSE
92                         reg_fstate <= D;
93                     END IF;
94
95                     reg_CY <= '0';
96                     WHEN E =>
97                     IF (NOT((w = '1'))) THEN
98                         reg_fstate <= E;
99                     ELSIF ((w = '1')) THEN
100                        reg_fstate <= F;
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
    reg_fstate <= r;
    -- Inserting 'else' block to prevent latch inference
    ELSE
        reg_fstate <= E;
    END IF;
    reg_CY <= '0';
    WHEN F =>
        IF (NOT((w = '1'))) THEN
            reg_fstate <= F;
        ELSIF ((w = '1')) THEN
            reg_fstate <= G;
            -- Inserting 'else' block to prevent latch inference
        ELSE
            reg_fstate <= F;
        END IF;
        reg_CY <= '0';
    WHEN G =>
        IF (NOT((w = '1'))) THEN
            reg_fstate <= G;
        ELSIF ((w = '1')) THEN
            reg_fstate <= H;
            -- Inserting 'else' block to prevent latch inference
        ELSE
            reg_fstate <= G;
        END IF;
        reg_CY <= '0';
    WHEN H =>
        IF (NOT((w = '1'))) THEN
            reg_fstate <= H;
        ELSIF ((w = '1')) THEN
            reg_fstate <= A;
            -- Inserting 'else' block to prevent latch inference
        ELSE
            reg_fstate <= H;
        END IF;
        reg_CY <= '0';
        WHEN OTHERS =>
            reg_CY <= 'X';
            report "Reach undefined state";
    END CASE;
    END IF;
    END PROCESS;
END BEHAVIOR;

```

(1)

(2)

(3)

ការស្នើសុំ Vhdl ទាំង 3 ឧបករណ៍ និងការស្នើសុំ នូវការបញ្ចប់របស់ការ  
ផ្តល់ទម្រង់បានសំខាន់សំខាន់ និងការបញ្ចប់របស់ការ

Finite State Machine

#