จุฬาลงกรณ์มหาวิทยาลัย	ชื่อ ภูมิพัฒน์ ชัยประเสริฐสุด
คณะวิศวกรรมศาสตร์	เลขประจำตัว 6432133721
ภาควิชาวิศวกรรมคอมพิวเตอร์	หมายเลขเครื่อง
2110-263 DIGITAL COMPUTER LOGIC LAB I	วันที่ 8 กันยายน 2565

4. การเปลี่ยนแปลงรหัสเลขและการออกแบบวงจรแบบ Hierachy

วัตถุประสงค์

1. เพื่อให้นิสิตเข้าใจรหัสเลขแบบต่าง ๆ
2. เพื่อให้นิสิตสร้าง Library ของอุปกรณ์ที่สร้างขึ้นเองได้
3. เพื่อให้นิสิตสามารถออกแบบและสร้างวงจรตรรกะขนาดใหญ่ แบ่งเป็น Hierachy หลายระดับ
4. เพื่อให้นิสิตรู้จักอุปกรณ์และการใช้งานโปรแกรมจำลองวงจรเพิ่มเติม

บทนำ

ในระบบ digital “ตัวเลข” ที่ใช้ในการออกแบบมีแค่ 0 และ 1 เท่านั้น แต่ในการใช้งานจำเป็นจะต้องรับและแสดงค่าที่เป็นเลขฐาน 10 จึงต้องมีการนำเลขฐาน 10 มาเข้ารหัส (encode) เพื่อให้ใช้ 0 และ 1 ทดแทนเลขเหล่านี้ได้ นอกเหนือจากการใช้การแปลงเป็นเลขฐาน 2 (binary) ตรงๆแล้ว ยังมีรหัสอื่นๆอีก ซึ่งรหัสเหล่านี้จะมีคุณสมบัติต่างๆเช่น

* self complement คือ 9’s complement ของเลขแต่ละจำนวนจะได้จากการ invert 0 และ 1 ในแต่ละหลักของรหัสนั้น (9’s complement ของเลขใดคือเลขที่บวกกับเลขนั้นแล้วได้ผลเป็น 9 เช่น 9’s complement ของ 2 คือ 7) รหัสที่มีคุณสมบัตินี้เช่น Excess-3 , 2 4 2 1 code, 6 4 2 -3 code
* cyclic คือ เลขแต่ละจำนวนที่อยู่เรียงกันจะต่างกันเพียง 1 บิต รหัสที่มี คุณสมบัตินี้เช่น cyclic code

รหัสเหล่านี้ใช้แทนเลขฐาน 10 โดยการแทนเลขแต่ละหลักของฐาน 10 เช่น ใน Excess-3 3 แทนด้วย 0110 และ 5 แทนด้วย 1000 ดังนั้น ถ้าจะแทนเลข 53 จะใช้ 8 บิต ใน Excess-3 คือ 1000 0110

รหัสลักษณะนี้แบ่งได้เป็น 2 ชนิดคือ Weighted Code และ Nonweighted Code

* Weighted code คือ รหัสที่แต่ละบิตมีตัวคูณสำหรับคูณค่าในบิตนั้น เช่น รหัสแบบ 6 4 2 -3 เลข 1010 แทน 8 ซึ่งได้มาจาก 6x1 + 4x0 + 2x1 + -3x0 = 8 เป็นต้น

* Nonweighted code คือ รหัสที่ไม่มี ตัวคูณในแต่ละ บิต เช่น Excess-3 ได้จากการเลื่อนรหัสไป 3 (บวก 3 ให้เลขแต่ละจำนวน) หรือ cyclic code เกิดจากการเรียงลำดับเลขใหม่

Decimal	Binary	Excess-3	Cyclic	2 4 2 1 code	6 4 2 -3 code
0	0 0 0 0	0 0 1 1	0 0 0 0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 1 0 0	0 0 0 1	0 0 0 1	0 1 0 1
2	0 0 1 0	0 1 0 1	0 0 1 1	0 0 1 0	0 0 1 0
3	0 0 1 1	0 1 1 0	0 0 1 0	0 0 1 1	1 0 0 1
4	0 1 0 0	0 1 1 1	0 1 1 0	0 1 0 0	0 1 0 0
5	0 1 0 1	1 0 0 0	0 1 1 1	1 0 1 1	1 0 1 1
6	0 1 1 0	1 0 0 1	0 1 0 1	1 1 0 0	0 1 1 0
7	0 1 1 1	1 0 1 0	0 1 0 0	1 1 0 1	1 1 0 1
8	1 0 0 0	1 0 1 1	1 1 0 0	1 1 1 0	1 0 1 0
9	1 0 0 1	1 1 0 0	1 1 0 1	1 1 1 1	1 1 1 1

นอกจากนี้แล้วยังมีการเข้ารหัสเพื่อใช้ในการส่งข้อมูล ซึ่งมีจุดประสงค์เพื่อการตรวจสอบว่าข้อมูลที่ได้รับมีความถูกต้องตรงกับที่ผู้ส่งได้ส่งมาหรือไม่ (error detection) เช่น การใช้ parity บิต คือเพิ่มบิต ในการส่งข้อมูลอีก 1 บิต เพื่อให้จำนวนของ เลข 1 ในทั้งชุดเป็นจำนวนคู่ (even) และถ้าถือว่าระบบที่ใช้ส่ง (เช่น สายส่ง) สามารถก่อให้เกิดความผิดพลาดได้ไม่เกิน 1 บิตต่อข้อมูลแต่ละชุด ถ้าผู้รับได้รับข้อมูลที่มีจำนวน 1 เป็นเลขคี่แสดงว่าข้อมูลที่ได้รับไม่ถูกต้อง และถ้าผู้รับได้รับข้อมูลที่มีจำนวน 1 เป็นเลขคู่แสดงว่าข้อมูลที่ได้รับถูกต้อง ตัวอย่างอื่นเช่น รหัส 2-out-of-5 คือ จะมี 1 แค่ 2 ตัวในแต่ละเลข ถ้าผู้รับได้รับรหัสที่มี 1 จำนวน 2 ตัวพอดีในแต่ละชุดแสดงว่าถูกต้อง มิฉะนั้นถือว่าผิดพลาด เนื่องจากรหัส 2 ชนิดนี้สามารถตรวจสอบการผิดพลาดได้แค่ 1 บิต จึงเรียกว่า Single error detection

Decimal	With Even Parity bit 8 4 2 1 p	2-out-of-5 Code
0	0 0 0 0 0	0 0 0 1 1
1	0 0 0 1 1	1 1 0 0 0
2	0 0 1 0 1	1 0 1 0 0
3	0 0 1 1 0	0 1 1 0 0
4	0 1 0 0 1	1 0 0 1 0
5	0 1 0 1 0	0 1 0 1 0
6	0 1 1 0 0	0 0 1 1 0
7	0 1 1 1 1	1 0 0 0 1
8	1 0 0 0 1	0 1 0 0 1
9	1 0 0 1 0	0 0 1 0 1

ยังมีรหัสที่นอกจากจะใช้ตรวจสอบได้ว่ามีความผิดพลาดหรือไม่ ยังบอกได้ว่า ความผิดพลาดนั้นอยู่ที่บิตใด ซึ่งเมื่อทราบว่าเป็นบิตใดย่อมทำให้ทราบว่า ข้อมูลที่ถูกต้องเป็นอย่างไรด้วย โดยการกลับ 0 เป็น 1 หรือกลับ 1 ให้เป็น 0 ในบิตนั้น รหัสประเภทนี้เรียกว่า Error Correction Code ตัวอย่างของรหัสประเภทนี้คือ Hamming code ซึ่งสามารถแก้ความผิดพลาดได้ไม่เกิน 1 บิตเท่านั้น (single error correction)

Hammimg Code ที่เป็น single error correction สำหรับเลข 0-9 ประกอบด้วย 7 บิต เรียงกันดังนี้

โดย m คือตัวข้อมูล และ p คือ parity บิต ที่แทรกเพิ่มเพื่อใช้ในการตรวจแก้ถ้ามีความผิดพลาดเกิดขึ้น

1	2	3	4	5	6	7
p1	p2	m1	p3	m2	m3	m4

ขอให้สังเกตุหมายเลขตำแหน่งที่ใช้ว่า เริ่มจาก 1 และเริ่มจากซ้ายไปขวา

การคำนวณหาตำแหน่งที่ผิดพลาดทำโดย หาค่าของ C1, C2 และ C3

C1= XOR(บิต 4 ,บิต 5 ,บิต 6,บิต 7)

C2= XOR(บิต 2 ,บิต 3 ,บิต 6,บิต 7)

C3= XOR(บิต 1 ,บิต 3 ,บิต 5,บิต 7)

ค่าของ C1C2C3 จะบอกตำแหน่งที่ผิดเช่น C1C2C3=000 ไม่มีที่ผิด C1C2C3=100 ตำแหน่ง 4 ผิด

ตัวอย่าง สมมุติว่า ได้รับข้อมูลเป็น 0001000

1	2	3	4	5	6	7
p1	p2	m1	p3	m2	m3	m4
0	0	0	1	0	0	0

C1= XOR(บิต 4 ,บิต 5 ,บิต 6,บิต 7) = XOR(1,0,0,0)=1

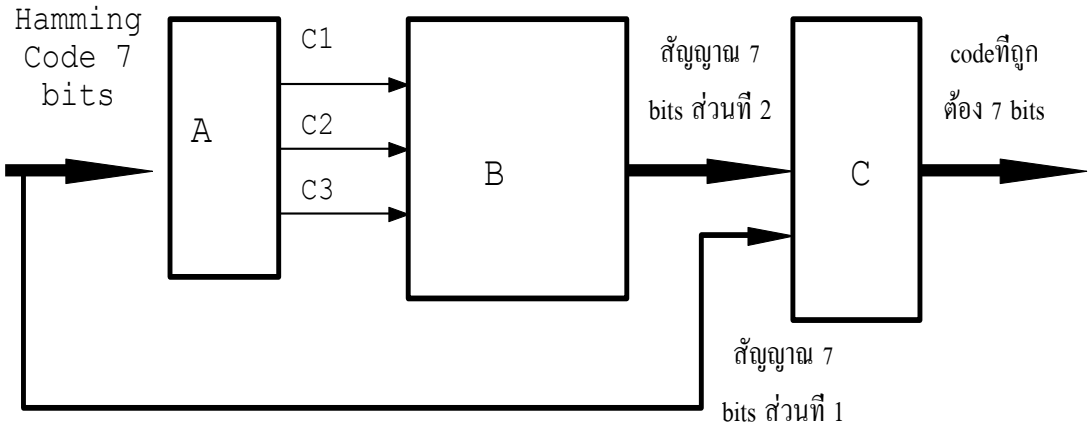
C2= XOR(บิต 2 ,บิต 3 ,บิต 6,บิต 7) = XOR (0,0,0,0)=0

C3= XOR(บิต 1 ,บิต 3 ,บิต 5,บิต 7) =XOR(0,0,0,0)=0

ดังนั้น บิต ที่ผิดคือ บิต 4 ซึ่งข้อมูลที่ถูกต้องคือ 0 0 0 0 0 0 0

การออกแบบวงจรตรรกะแบบ Hierachy

ในการออกแบบการเขียนตารางความจริงของปัญหาทั้งหมดในครั้งเดียวอาจทำได้ยาก เช่น ในกรณีของการออกแบบวงจรที่ตรวจสอบและแก้ไข Hamming code จำนวน input มี 7 บิต ซึ่ง ตารางความจริงจะมี 2^7 =128 row ซึ่งการเขียนตารางความจริงขนาดนั้นทำได้ยากและมีโอกาสผิดพลาดสูง การออกแบบควรทำโดยแบ่งวงจรเป็นส่วนย่อย (block) แต่ละ block จะทำงานย่อย และสร้าง input ให้ block ต่อไป ตัวอย่างเช่น Hamming code อาจแบ่งเป็น



โดย วงจร A ทำหน้าที่คำนวณหา ค่า C1C2C3

วงจร B เป็นวงจรส่งสัญญาณเพื่อ invert บิต ตามที่กำหนดโดย C1C2C3 และมีเอาท์พุท 7 บิต โดยถ้า C1C2C2=000 เอาท์พุททั้งหมดเป็น 0 ถ้า C1C2C2=001 เอาท์พุทที่ 1 เป็น 1 นอกนั้นเป็น 0 ถ้า C1C2C2=010 เอาท์พุทที่ 2 เป็น 1 นอกนั้นเป็น 0 ...

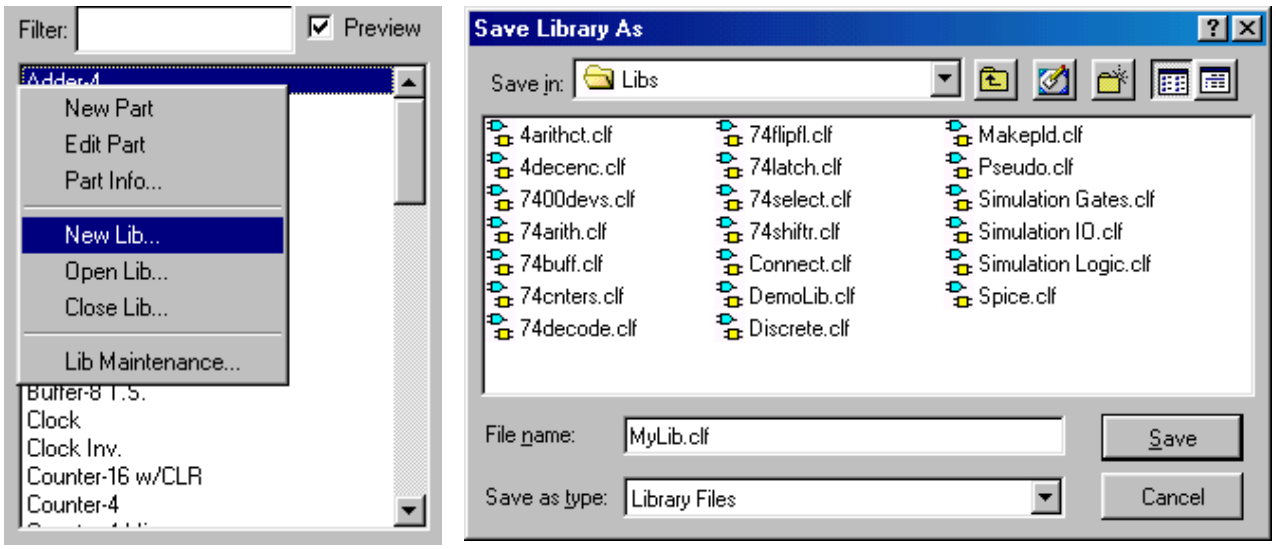
วงจร C เป็นวงจรที่ invert แต่ละบิต ของ input ส่วนที่ 1 ตามสัญญาณของ input ส่วนที่ 2 โดยถ้า input บิตใดของสัญญาณส่วนที่ 2 เป็น 1 จะ invert บิตนั้นของสัญญาณส่วนที่ 1 เมื่อทำงานเสร็จ output 7 บิตของวงจร C จะเป็นข้อมูลที่ถูกต้องแล้ว

ในกรณีที่วงจรมีขนาดใหญ่ทำให้วาดลงไปใน 1 หน้าไม่พอ โปรแกรม Logic works™ จะเพิ่มพื้นที่ให้เองโดยอัตโนมัติ สังเกตุได้โดยคลิ๊ก Schematic -> Design Preferences.. แล้วติ๊ก Show Printed Page Breaks แล้วในหน้าต่างออกแบบจะเห็นกรอบของหน้าปรากฏขึ้น การออกแบบวงจรขนาดใหญ่ในแผ่นเดียวกัน จะทำให้ดูยาก แก้ไขปรับปรุงวงจรไม่สะดวก จึงแนะนำให้ออกแบบเป็น Hierachy หรือ Block ขึ้น ซึ่งเมื่อออกแบบเป็นก้อนเล็กๆ แล้ว สามารถทดสอบไปทีละก้อน เมื่อต้องการปรับปรุงแก้ไขก็ไปแก้ไขที่ก้อนเล็กนั้นและทดสอบซึ่งจะทำให้ง่ายกว่าต้องดูทั้งหมดของวงจร โดยเฉพาะถ้าฟังก์ชันของก้อนนั้นมีใช้หลายแห่งในวงจร การแก้ไขก็ทำที่เดียว

วิธีการออกแบบเป็น Hierachy หรือ Block นั้นทำได้โดยสร้าง Library สำหรับเก็บอุปกรณ์ใหม่และสร้างสัญลักษณ์ของอุปกรณ์ใหม่ โดยมีวิธีทำดังนี้

- สร้าง Library ใหม่

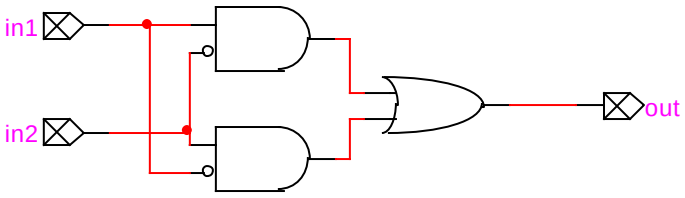
  เลื่อนเมาส์ไปที่หน้าต่าง Parts คลิกขวาแล้วเลือก New Lib…



แล้วตั้งชื่อ Library ที่ต้องการ เช่น MyLib.clf

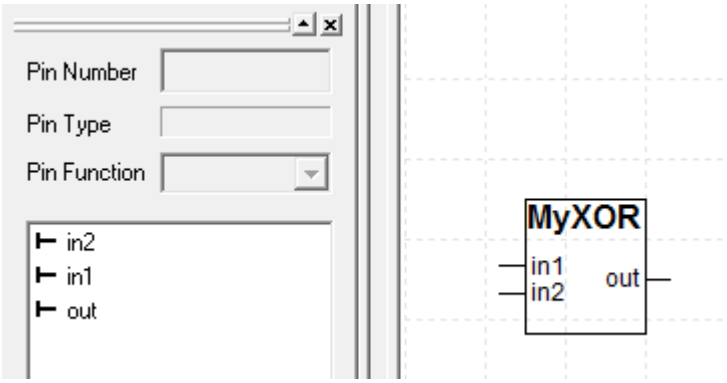
- การสร้างสัญลักษณ์อุปกรณ์

ในหน้าต่างออกแบบวงจรให้เชื่อมสัญญาณอินพุททั้งหมดกับอุปกรณ์ Port In และ สัญญาณเอาท์พุทกับอุปกรณ์ Port Out จาก Library Connectors.CLF แล้วตั้งชื่อแต่ละ Port (ใช้ปุ่ม A ใน toolbar หรือ คลิกขวาที่อุปกรณ์นั้นแล้วเลือก Name..) ให้ครบ ตัวอย่างดังรูป



เสร็จแล้วในหน้าต่าง Parts ให้คลิกขวาแล้วเลือก New Part โปรแกรมจะเปิดหน้าต่างวาดสัญลักษณ์อุปกรณ์ให้พร้อมทั้งมี toolbar ชุดใหม่เพื่อใช้ในการวาดสัญลักษณ์และติดขาสัญญาณ (Pin) ในทิศทางต่าง ๆ ก่อนวาดให้เลือก Subcircuit and Part Type จากเมนู Option ที่หน้าต่าง pop up ให้เลือก Create a subcircuit symbol and select an open circuit to attach to it (ตัวเลือกที่สอง) จากนั้นเลือกวงจรที่ต้องการจะสร้างสัญลักษณ์ จะปรากฏหน้าต่างใหม่โดยมี List ของ Pin (มาจากชื่อ Port In, Port Out จากวงจรที่เลือก) อยู่ทางซ้ายมือ ให้วาดสัญลักษณ์โดยเลือกใช้เครื่องมือวาดภาพจาก toolbar การวาง pin บนสัญลักษณ์อุปกรณ์ต้องวางตามลำดับก่อนหลังใน List ของ Pin จนหมด เมื่อใส่ label และจัดวางทุกอย่างเรียบร้อยแล้วให้เลือก File -> Save… จะปรากฏหน้าต่างให้เลือกใส่ชื่อ Part และเลือก Library ที่ต้องการนำอุปกรณ์ใหม่นี้ไปเก็บ (ให้เลือก MyLib.clf) เมื่อทำเสร็จเรียบร้อยแล้วจะปรากฏชื่ออุปกรณ์ใหม่นี้ในหน้าต่าง Parts ใน Library ที่เรากำหนด

*หากไม่ต้องการวาดสัญลักษณ์อุปกรณ์เองให้กด CTRL+J เพื่อ Auto Create Symbol ได้



การทดลอง

1. ออกแบบและสร้างวงจรที่มีอินพุทเป็น hex keyboard wo/STB 1อัน binary switch 2 อันและเอาท์พุทเป็น binary probe 4 อัน โดย hex keyboard จะเป็นตัวกำหนดค่า binary input และ switch เป็นตัวเลือกว่าต้องการให้แปลงเป็นรหัสใดโดย

Switch เป็น	output เป็นรหัส
00	Excess-3
01	Cyclic
10	2 4 2 1 code
11	6 4 2 -3 code

ข้อแนะนำออกแบบวงจรสำหรับรหัสแต่ละอันแล้วใช้ Multiplexer ในการเลือก output มาออก

2. ออกแบบและสร้างวงจรที่มีอินพุท เป็น binary switch 7 อัน และ เอาท์พุทเป็น binary probe 7อัน โดยวงจรจะทำหน้าที่ตรวจสอบและแก้ไข Hamming code ที่เข้ามาทางอินพุท (ซึ่งให้ถือว่าผิดพลาดได้ไม่เกิน 1 บิต) และให้หา Hamming Code ที่ถูกต้องออกมาทางเอาท์พุท

Decimal digit	Position	1 p_1	2 p_2	3 m_1	4 p_3	5 m_2	6 m_3	7 m_4
0		0	0	0	0	0	0	0
1		1	1	0	1	0	0	1
2		0	1	0	1	0	1	0
3		1	0	0	0	0	1	1
4		1	0	0	1	1	0	0
5		0	1	0	0	1	0	1
6		1	1	0	0	1	1	0
7		0	0	0	1	1	1	1
8		1	1	1	0	0	0	0
9		0	0	1	1	0	0	1

ตาราง Hamming code for BCD