

จุฬาลงกรณ์มหาวิทยาลัย

ชื่อ _____

คณะวิศวกรรมศาสตร์

เลขประจำตัว _____

ภาควิชาวิศวกรรมคอมพิวเตอร์

หมายเลขเครื่อง _____

2110-263 DIGITAL COMPUTER LOGIC LAB I

วันที่ _____

4. การเปลี่ยนแปลงรหัสเลขและการออกแบบวงจรแบบ Hierarchy

วัตถุประสงค์

1. เพื่อให้นิสิตเข้าใจรหัสเลขแบบต่าง ๆ
2. เพื่อให้นิสิตสร้าง อุปกรณ์ เองได้
3. เพื่อให้นิสิตสามารถออกแบบและสร้างวงจรตรรกะขนาดใหญ่ แบ่งเป็น Hierarchy หลายระดับ
4. เพื่อให้นิสิตรู้จักอุปกรณ์และการใช้งานโปรแกรมจำลองวงจรเพิ่มเติม

บทนำ

ในระบบ digital “ตัวเลข” ที่ใช้ในการออกแบบมีแค่ 0 และ 1 เท่านั้น แต่ในการใช้งานจริงจะต้องรับและแสดงค่าที่เป็นเลขฐาน 10 จึงต้องมีการนำเลขฐาน 10 มาเข้ารหัส (encode) เพื่อให้ใช้ 0 และ 1 ทดแทนเลขเหล่านั้นได้ นอกเหนือจากการใช้การแปลงเป็นเลขฐาน 2 (binary) ตรงๆ แล้ว ยังมีรหัสอื่นๆ อีก ซึ่งรหัสเหล่านี้จะมีคุณสมบัติต่างๆ เช่น

- * 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 Non-weighted Code

- * Weighted code คือ รหัสที่แต่ละบิตมีตัวคูณสำหรับคูณค่าในบิตนั้น เช่น รหัสแบบ 6 4 2 -3 เลข 1010 แทน 8 ซึ่งได้มาจาก $6 \times 1 + 4 \times 0 + 2 \times 1 + -3 \times 0 = 8$ เป็นต้น

- * Non-weighted 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)

Hamming 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

ขอให้สังเกตหมายเลขตำแหน่งที่ใช้ว่า เริ่มจาก 1 และเริ่มจากซ้ายไปขวา

การคำนวณหาตำแหน่งที่ผิดพลาดทำโดย หาค่าของ C1, C2 และ C3

$C1 = \text{XOR (บิต 4, บิต 5, บิต 6, บิต 7)}$

$C2 = \text{XOR (บิต 2, บิต 3, บิต 6, บิต 7)}$

$C3 = \text{XOR (บิต 1, บิต 3, บิต 5, บิต 7)}$

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

$C1 = \text{XOR (บิต 4, บิต 5, บิต 6, บิต 7)} = \text{XOR (1,0,0,0)} = 1$

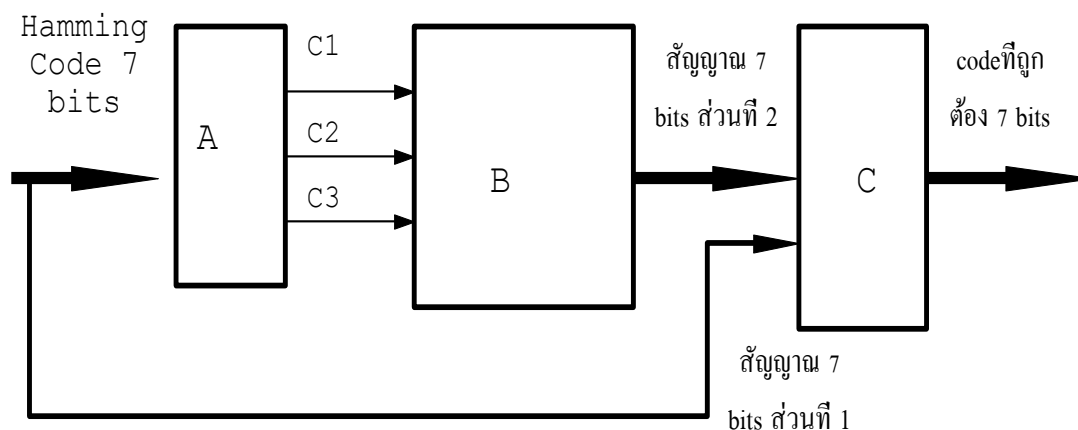
$C2 = \text{XOR (บิต 2, บิต 3, บิต 6, บิต 7)} = \text{XOR (0,0,0,0)} = 0$

$C3 = \text{XOR (บิต 1, บิต 3, บิต 5, บิต 7)} = \text{XOR (0,0,0,0)} = 0$

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

การออกแบบวงจรตรรกะแบบ Hierarchy

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 และมีเอาต์พุต 7 บิต โดยถ้า C1C2C3=000 เอาต์พุตเป็น 0000000 ถ้า C1C2C3=001 เอาต์พุตเป็น 1000000 ถ้า C1C2C3=010 เอาต์พุตเป็น 0100000 ถ้า C1C2C3=011 เอาต์พุตเป็น 0010000 ถ้า . . .

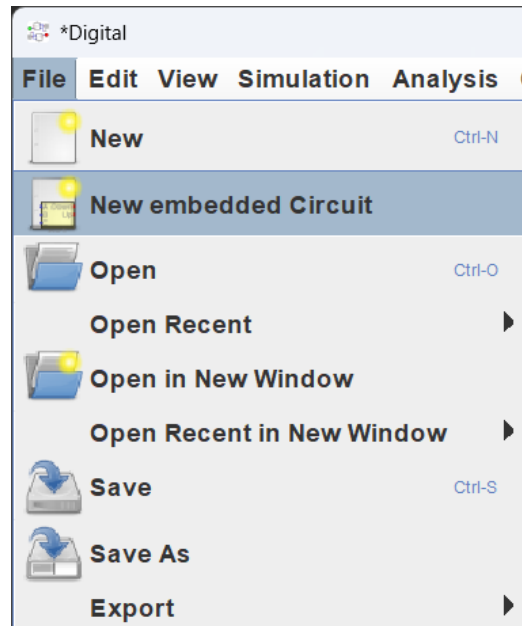
วงจร C เป็นวงจรที่ invert แต่ละบิต ของ input ส่วนที่ 1 ตามสัญญาณของ input ส่วนที่ 2 โดยถ้า input บิตใดของสัญญาณส่วนที่ 2 เป็น 1 จะ invert บิตนั้นของสัญญาณส่วนที่ 1 เมื่อทำงานเสร็จ output 7 บิตของวงจร C จะเป็นข้อมูลที่ถูกต้องแล้ว

การออกแบบวงจรขนาดใหญ่ในแผ่นเดียวกัน จะทำให้ดูยาก แก้ไขปรับปรุงวงจรไม่สะดวก จึงแนะนำให้ ออกแบบเป็น Hierarchy หรือ Block ขึ้น ซึ่งเมื่อออกแบบเป็นก้อนเล็กๆแล้ว สามารถทดสอบไปที่ละก้อน เมื่อ ต้องการปรับปรุงแก้ไขก็ไปแก้ไขที่ก้อนเล็กนั้นและทดสอบซึ่งจะทำให้ง่ายกว่าต้องดูทั้งหมดของวงจร โดยเฉพาะถ้า ฟังก์ชันของก้อนนั้นมีใช้หลายแห่งในวงจร การแก้ไขก็ทำได้ง่าย

วิธีการออกแบบเป็น Hierarchy หรือ Block มีวิธีทำดังนี้

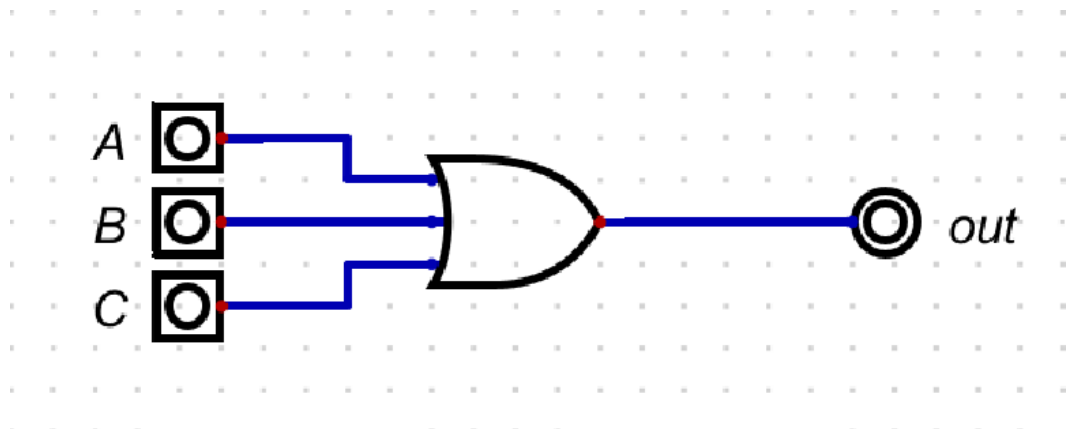
- สร้างวงจรใหม่

ไปที่ file -> new embedded Circuit เพื่อทำการเปิดหน้าต่างใหม่ขึ้นมา

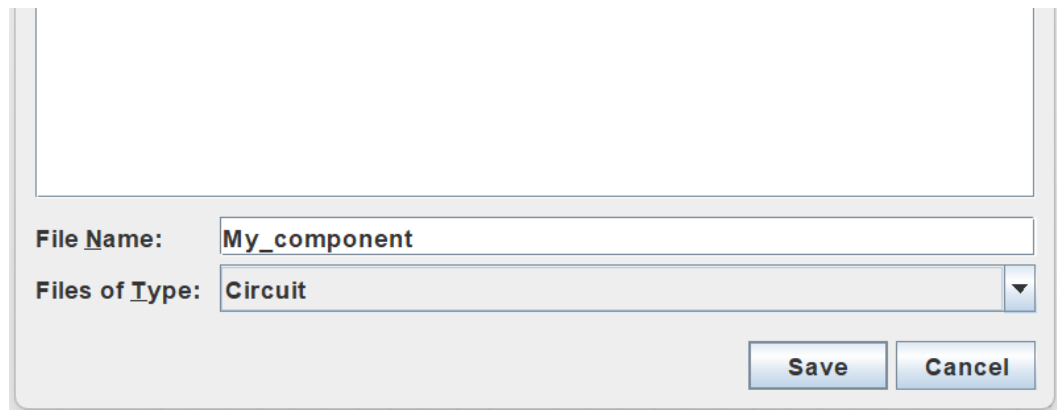


- ออกแบบวงจร

ในหน้าต่างใหม่ให้ทำการออกแบบวงจรได้ตามต้องการโดยจะ ต้องมี Input port และ output port และต้องตั้งชื่อให้ทุก port

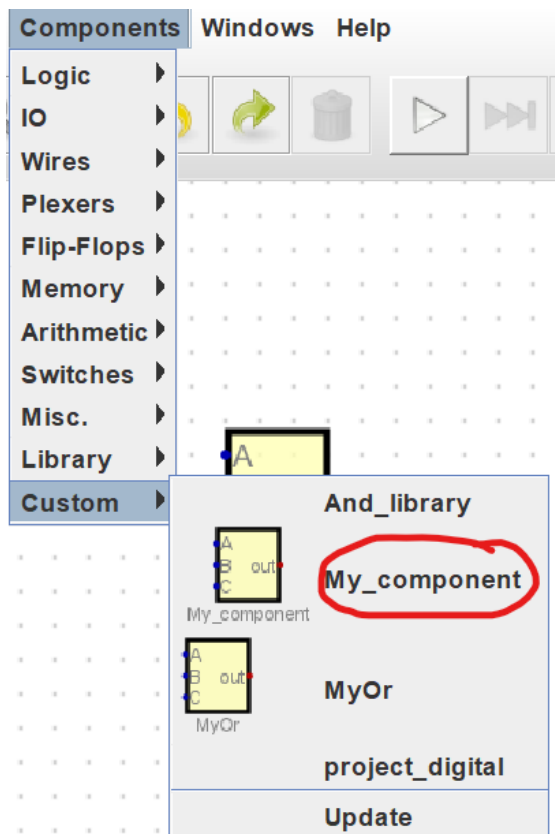


เมื่อสร้างเสร็จแล้วให้ทำการ save วงจรนี้ไว้ที่เดียวกับวงจรที่เราจะนำวงจรนี้ไปใช้งาน



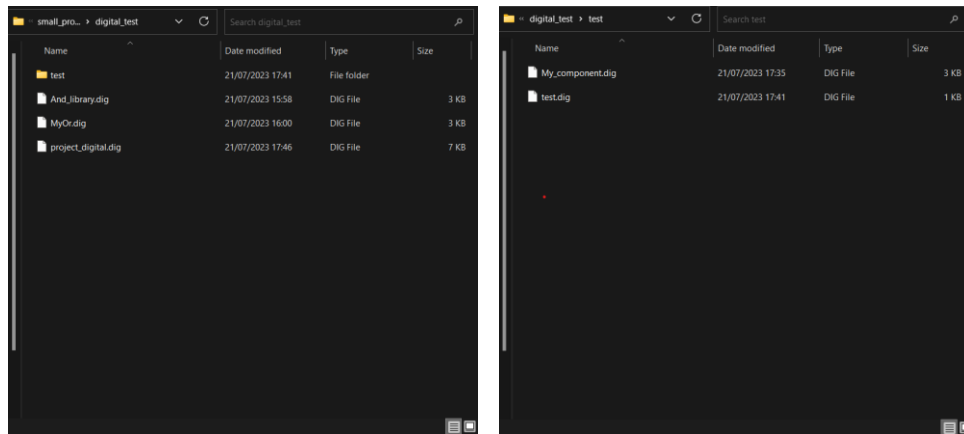
การใช้งาน component ที่สร้าง

ไปที่ components -> custom และทำการเลือก component ที่ได้ทำการสร้างไว้

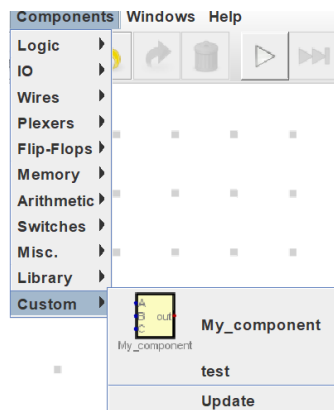


การจัดเก็บ component ที่สร้าง

component ที่สามารถใช้ได้ใน custom นั้นจะต้องเป็นไฟล์ .dig อยู่ใน directory เดียวกันหรือเก็บอยู่ใน folder ที่อยู่ใน directory เดียวกับไฟล์ที่เราใช้งานอยู่ ยกตัวอย่างเช่น



ในตัวอย่าง ทางรูปด้านขวา มี ไฟล์ And_library.dig, MyOr.dig, project_digital.dig และ folder ชื่อ test และรูปทางด้านซ้ายคือ ข้างใน folder test ซึ่งมีไฟล์ My_component.dig และ test.dig เมื่อเราเปิดไฟล์ test.dig จะสามารถใช้งาน my_component ได้ แต่จะไม่สามารถใช้งาน MyOr ที่ไม่ได้อยู่ใน folder ได้



ในทางกลับกัน ในไฟล์ project_digital.dig สามารถใช้งาน MyOr.dig, And_library.dig รวมไปถึง component ที่อยู่ใน folder test ได้ด้วยดังรูป

