จุฬาลงกรณ์มหาวิทยาลัย	ชื่อ _____
คณะวิศวกรรมศาสตร์	เลขประจำตัว _____
ภาควิชาวิศวกรรมคอมพิวเตอร์	หมายเลขเครื่อง _____
2110-263 DIGITAL COMPUTER LOGIC LAB I	วันที่ _____

3. การออกแบบวงจรตรรกะด้วยการลดขนาดนิพจน์บูลีน

<u>วัตถุประสงค์</u>

1. เพื่อให้นิสิตสามารถใช้โปรแกรม Espresso ช่วยในการลดนิพจน์บูลีน
2. เพื่อให้นิสิตออกแบบและจำลองการทดสอบวงจรตรรกะ
3. เพื่อให้นิสิตรู้จักอุปกรณ์ของโปรแกรมจำลองวงจรเพิ่ม
4. เพื่อให้นิสิตเข้าใจรหัสเลขแบบต่าง ๆ
5. เพื่อให้นิสิตสร้าง อุปกรณ์ เองได้
6. เพื่อให้นิสิตสามารถออกแบบและสร้างวงจรตรรกะขนาดใหญ่ แบ่งเป็น Hierarchy หลายระดับ
7. เพื่อให้นิสิตรู้จักอุปกรณ์และการใช้งานโปรแกรมจำลองวงจรเพิ่มเติม

<u>บทนำ</u>

การพยายามลดนิพจน์บูลีนลงก่อน จะทำให้การสร้างวงจรทำได้ง่ายขึ้นและมีขนาดเล็กลง วิธีการลดนิพจน์บูลีนมีหลายวิธี เช่น Karnaugh-Map (K-Map) Quine-McCluskey Method ซึ่งเป็นวิธีที่ใช้กันมานาน ในปัจจุบันมีการใช้คอมพิวเตอร์ช่วยในขั้นตอนนี้ และมีหลายโปรแกรมที่สามารถทำงานนี้ได้ โปรแกรมที่จะใช้ในการปฏิบัติการนี้คือ Espresso ของ University of California at Berkeley Source code ของ Espresso ได้ถูกนำมา Recompile เพื่อให้ใช้ได้ภายใต้ DOS

การใช้โปรแกรม Espresso เบื้องต้น

โปรแกรม Espresso เป็นโปรแกรมที่ทำงานภายใต้ DOS ที่ทำงานได้ทั้ง Interpreter mode และ Compile mode

- การใช้งานแบบ Interpreter ให้เรียกโปรแกรม Espresso โดยเรียก Command Prompt ขึ้นมาก่อน โดยไปที่ Start -> Programs -> Accessories -> Command Prompt แล้วพิมพ์คำว่า espresso ในโฟลเดอร์ที่มี ไฟล์ espresso.exe อยู่ จากนั้นให้ป้อนคำสั่งต่างๆ และ ข้อมูลลงไป ตาม Berkeley PLA Format เมื่อสิ้นสุดโปรแกรม (โดยคำสั่ง ".e") โปรแกรมก็จะสร้าง ผลของนิพจน์ที่ทำการลดขนาดให้เรียบร้อย การใช้งานแบบนี้จะไม่มีการ save ดังนั้นเมื่อทำผิดก็เริ่มใหม่

ตัวอย่าง Y = f(A,B,C,D) = Σ m(0,3,5,12,13) + Σ d(1,2,15)

สามารถสร้าง Berkeley PLA Format ได้ดังนี้

.i 4
.o 1
.ilb A B C D
.ob Y
.p 8
0000 1
0011 1
0101 1
1100 1
1101 1
0001 -
0010 -
1111 -
.e

- การทำงานแบบ Compiler วิธีนี้จะต้องใช้ Text Editor เช่น Notepad สร้างโปรแกรมและข้อมูล ลงไฟล์ก่อน ซึ่งfile type ควรเป็น .pla เช่น in.pla เสร็จแล้วใช้คำสั่ง

    espresso in.pla

ถ้าต้องการให้ผลลัพธ์ไปเก็บลงไฟล์ เช่น out.pla ก็ใช้คำสั่ง

espresso in.pla > out.pla

นอกจากนี้ยังมี Option อื่นๆอีกมากมาย เช่น

- ต้องการให้เอาต์พุตที่ออกมาเป็นสมการบูลีน

    espresso –o eqntott in.pla

- ต้องการได้ เอาต์พุตเป็น Inverse logic

    espresso –epos in.pla หรือ **espresso –epos –o eqntott in.pla**

    เมื่อเอาต์พุตที่ได้มาทำ Inverse อีกครั้ง ก็จะได้ฟังก์ชันออกมาในรูปของPOS(product-of-sums)

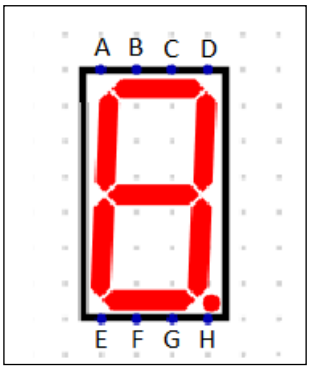
    เช่นตามตัวอย่างจะได้ $\overline{f}$ (A,B,C,D) = (A & !B) | (B & C) | (!A & B & !D)

เพราะฉะนั้น f (A,B,C,D) = (!A + B) (!B + !C) (A + !B + D)

- ต้องการกำหนดให้มีเอาต์พุตมากกว่าหนึ่งค่า ทำได้โดยกำหนดจำนวนเอาต์พุตที่ต้องการที่คำสั่ง “.o “ และตัวแปรที่ต้องการที่คำสั่ง “.ob “และใส่ค่าของเอาต์พุตเพิ่มตามจำนวนที่กำหนด เช่น จากตัวอย่างข้างต้น ถ้าต้องการ 2 เอาต์พุตก็ใส่เป็น 0000 **11** เป็นต้น

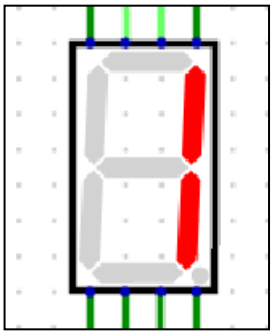
อุปกรณ์ใหม่ ที่จะใช้ในการปฏิบัติการคือ 7-segment display และ Terminal

- 7-segment display เป็นอุปกรณ์ที่ใช้แสดงผลที่เห็นได้ทั่วไป ประกอบด้วย ขีด (segment) 7 ขีด และ จุด (dot) 1 จุด ขีดและจุดเหล่านี้ คือ LED (Light Emitting Diode) ซึ่งเมื่อรับไฟฟ้าหรือสัญญาณ 1 จะเปล่งแสงออกมา 7-segment display จะมี ขา input 8 ขาสำหรับกำกับขีดแต่ละขีดและจุด โดยจะมีทั้งหมด 8 ขา A – H ดังรูป

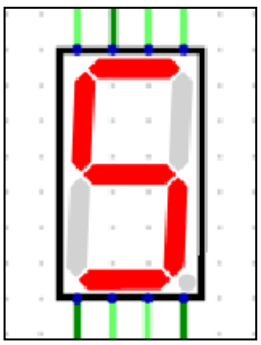


<u>ตัวอย่างการใช้งาน</u>

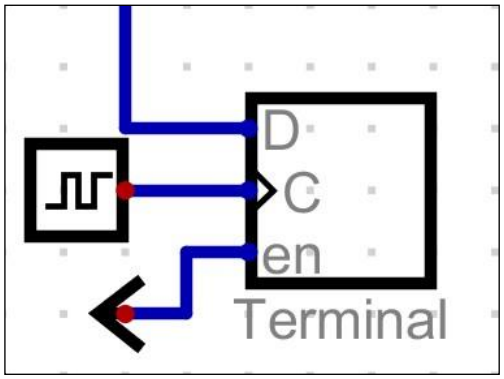
- แสดงเลข 1 โดยใส่กระแสไฟฟ้าในขา B,C



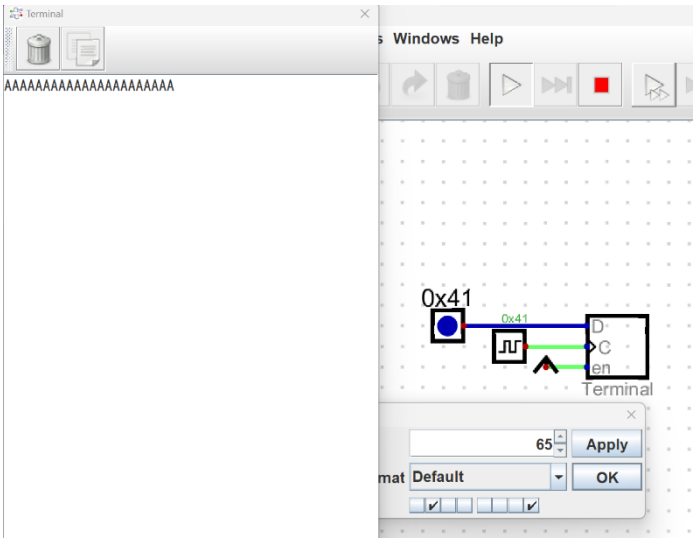
- แสดงเลข 5 โดยใส่กระแสไฟฟ้าในขา A, C, D, F, G



- Terminal เป็นอุปกรณ์ที่ใช้แสดงผลรหัสตัวอักษรตามค่า ASCII ที่รับเข้ามา โดยมีขา input คือ D รับสัญญาณรหัส ASCII 8 Databits ส่วนขา C คือ Clock ให้ต่อกับ Clock และขา en คือ enable ใช้เพื่อเปิด/ปิดอุปกรณ์ โดยอุปกรณ์จะเปิดเมื่อได้ขา en ได้รับสัญญาณ 1



<u>ตัวอย่างการใช้งาน</u>



ก่อนเริ่มการ simulate อย่าลืมเปิด real time clock เมื่อเริ่มการ simulate จะยังไม่มีอะไรเกิดขึ้น จนกว่าเราจะตั้งค่าข้อมูลที่ส่งมาที่ขา D จากตัวอย่างเราให้ input เป็น 65 ซึ่งในรหัส ASCII คือ ตัวอักษร A แล้วจะมีหน้าต่าง Terminal เปิดขึ้นแสดงตัวอักษร A ซึ่งเป็นรหัส ASCII ที่เราส่งเข้าไป

ในระบบ digital “ตัวเลข” ที่ใช้ในการออกแบบมีแค่ 0 และ 1 เท่านั้น แต่ในการใช้งานจำเป็นจะต้องรับและแสดงค่าที่เป็นเลขฐาน 10 จึงต้องมีการนำเลขฐาน 10 มาเข้ารหัส (encode) เพื่อให้ใช้ 0 และ 1 ทดแทนเลขเหล่านี้ได้ นอกเหนือจากการใช้การแปลงเป็นเลขฐาน 2 (binary) ตรงๆแล้ว ยังมีรหัสอื่นๆอีก 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 6x1 + 4x0 + 2x1 + -3x0 = 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) สามารถก่อให้เกิด

ความผิดพลาดได้ไม่เกิน 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

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

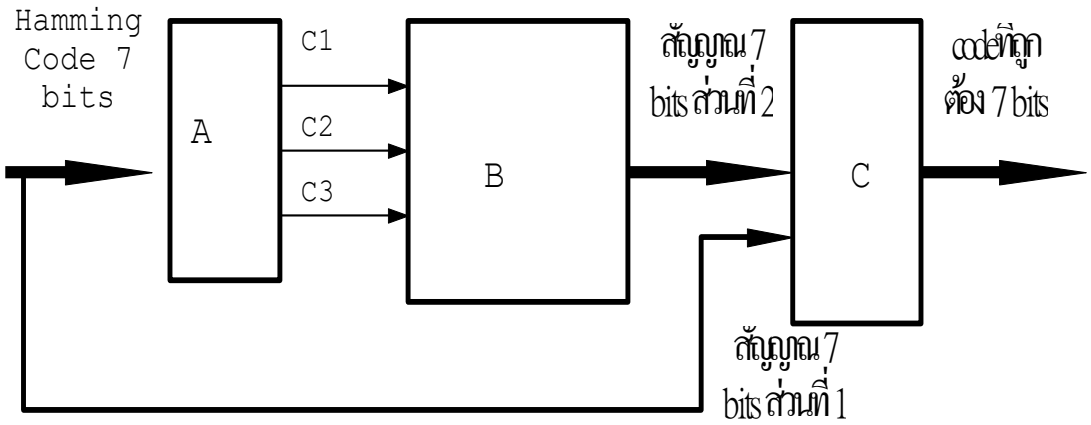
C2= XOR (บิต 2, บิต 3, บิต 6, บิต 7) = XOR (0,0,0,0) =0

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 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 . . .

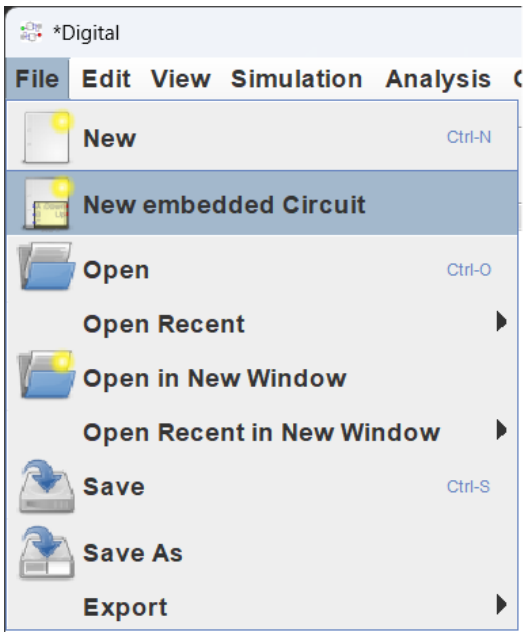
วงจร C เป็นวงจรที่ invert แต่ละบิต ของ input ส่วนที่ 1 ตามสัญญาณของ input ส่วนที่ 2 โดยถ้า input บิตใดของสัญญาณส่วนที่ 2 เป็น 1 จะ invert บิตนั้นของสัญญาณส่วนที่ 1 เมื่อทำงานเสร็จ output 7 บิตของวงจร C จะเป็นข้อมูลที่ถูกต้องแล้ว

การออกแบบวงจรขนาดใหญ่ในแผ่นเดียวกัน จะทำให้ดูยาก แก้ไขปรับปรุงวงจรไม่สะดวก จึงแนะนำให้ออกแบบเป็น Hierarchy หรือ Block ขึ้น ซึ่งเมื่อออกแบบเป็นก้อนเล็กๆแล้ว สามารถทดสอบไปทีละก้อน เมื่อต้องการปรับปรุงแก้ไขก็ไปแก้ไขที่ก้อนเล็กนั้นและทดสอบซึ่งจะทำให้ง่ายกว่าต้องดูทั้งหมดของวงจร โดยเฉพาะถ้าฟังก์ชันของก้อนนั้นมีใช้หลายแห่งในวงจร การแก้ไขก็ทำที่เดียว

วิธีการออกแบบเป็น Hierarchy หรือ Block มีวิธีทำดังนี้

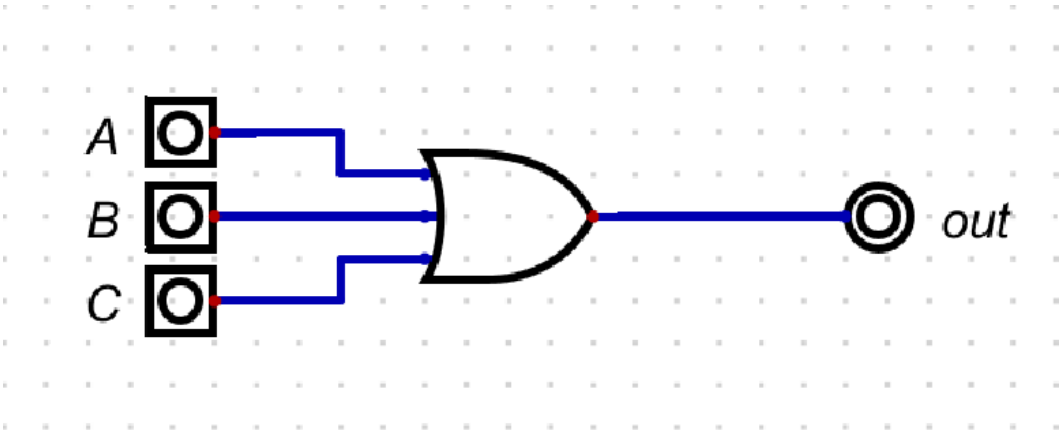
- สร้างวงจรใหม่

ไปที่ file -> new embedded Circuit เพื่อทำการเปิดหน้าต่างใหม่ขึ้นมา



- ออกแบบวงจร

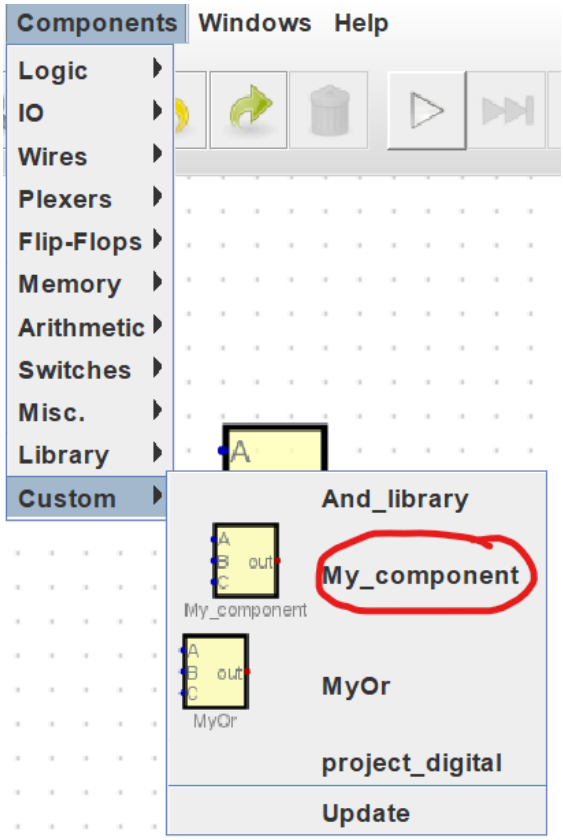
ในหน้าต่างใหม่ให้ทำการออกแบบวงจรได้ตามต้องการโดยจะ ต้องมี Input port และ output port **และต้องตั้งชื่อให้ทุก port**



เมื่อสร้างเสร็จแล้วให้ทำการ save วงจรนี้ไว้ที่เดียวกับวงจรที่เราจะนำวงจรนี้ไปใช้งาน

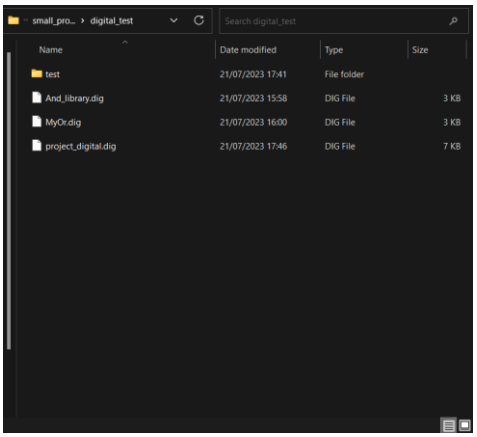
การใช้งาน component ที่สร้าง

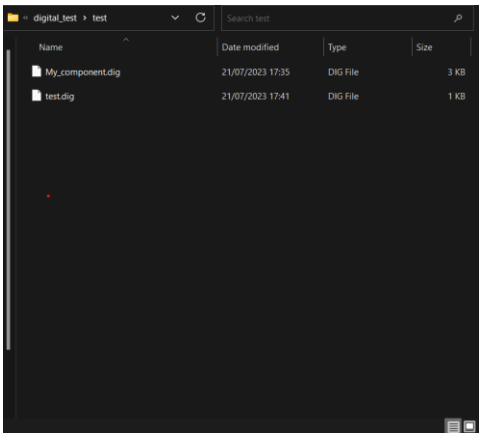
ไปที่ components -> custom และทำการเลือก component ที่ได้ทำการสร้างไว้



การจัดเก็บ component ที่สร้าง

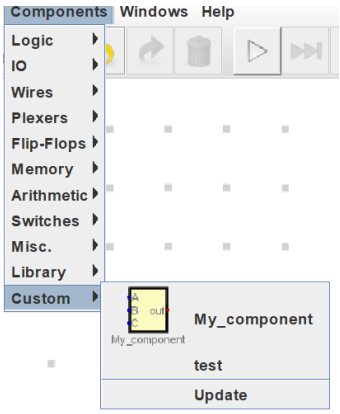
component ที่สามารถใช้ได้ใน custom นั้นจะต้องเป็นไฟล์. dig อยู่ใน directory เดียวกันหรือเก็บอยู่ folder ที่อยู่ directory เดียวกับไฟล์ที่เราใช้งานอยู่ ยกตัวอย่างเช่น





ในตัวอย่าง ทางรูปด้านขวา มี ไฟล์ And_library.dig, MyOr.dig, project_digital.dig และ floder ชื่อ test และรูปทางด้านซ้ายคือ ข้างใน floder test ซึ่งมีไฟล์ My_component.dig และ test.dig

เมื่อเราเปิดไฟล์ test.dig จะสามารถใช้งาน my_component ได้ แต่จะไม่สามารถใช้งาน MyOr ที่ไม่ได้อยู่ใน floder ได้



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