Assignment 2

วงจรเปรียบเทียบค่าจำนวน 2 บิต แสดงผลลัพธ์ทาง LED และ 7 SEGMENT

(2-bit comparator display on LED 3 state and display 7 SEGMENT)

จัดทำโดย

นางสาวนาดียา บุญญา รหัส 5910110167

เสนอ

อาจารย์ทวีศักดิ์ เรืองพีระกุล

รายวิชา 242-309 MICROCONTROLLER & INTERFAC

ภาคเรียนที่ 2 ปีการศึกษา 2564

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยสงขลานครินทร์

อุปกรณ์ที่ใช้ในการทำงาน

1. บอร์ด Arduino ATmega 2560	1 บอร์ด
2. สาย USB Port	1 เส้น
3. Dip Switch 2 bit	2 ตัว
4. Resistor 330 Ω	21 ตัว
5. Breadboard	3 บอร์ด
6. LED สีแดง	2 ดวง
7. LED สีฟ้า	2 ดวง
8. LED สีเขียว	2 ดวง
9. โปรแกรม AVR Studio	
10. โปรแกรม Xloader	

อุปกรณ์ที่ใช้ในการเขียนโปรแกรมและจำลองวงจร

- โปรแกรม AVR Studio7 ใช้ในการเขียนโปรแกรมและรันโปรแกรม
- โปรแกรม Xloader ใช้ในการอัปโหลดไฟล์โปรแกรมที่มีนามสกุล .hex ลงบอร์ด Arduino
- โปรแกรม Proteus ใช้ในการจำลองวงจร

การทำงานของโปรแกรม

1. วงจรทำการรับค่า Input ขนาด 2 บิต จำนวน 2 ชุด ผ่านทาง DIP Switch โดยรับค่า Input A เข้าผ่านพอร์ต C และ Input A พอร์ต D
2. โปรแกรมจะทำการเปรียบเทียบค่าจาก Input ทั้ง 2 ชุด โดยจะแสดงผลลัพธ์ออกมาแตกต่างกัน 3 กรณี ได้แก่ เท่ากับ มากกว่า และน้อยกว่า แสดงผลลัพธ์ออกทางพอร์ต B ออกสู่ LED 3 สี (สีเขียว สีน้ำเงิน สีแดง ตามลำดับ)

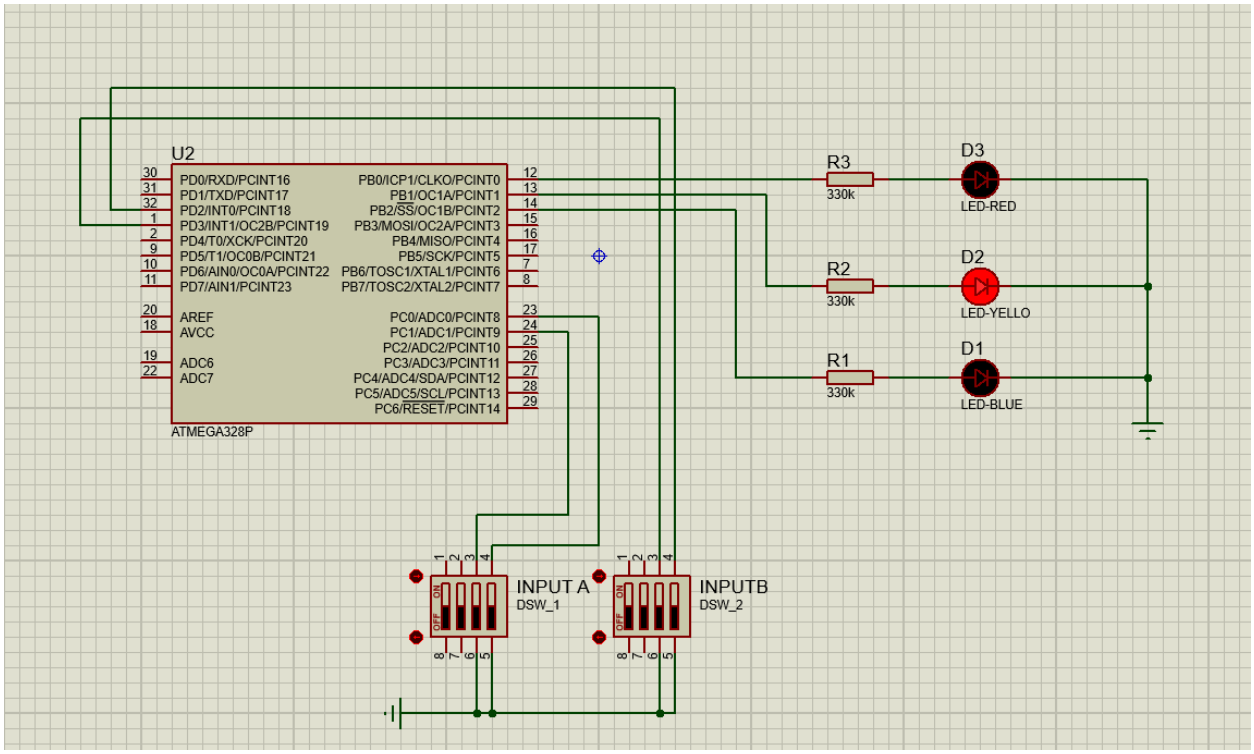
- สิ่งที่เพิ่มเข้ามา

  มีการแสดงผลค่า Output ผ่าน 7-segment เพื่อให้เห็นค่าเปรียบเทียบที่ชัดเจนยิ่งขึ้น

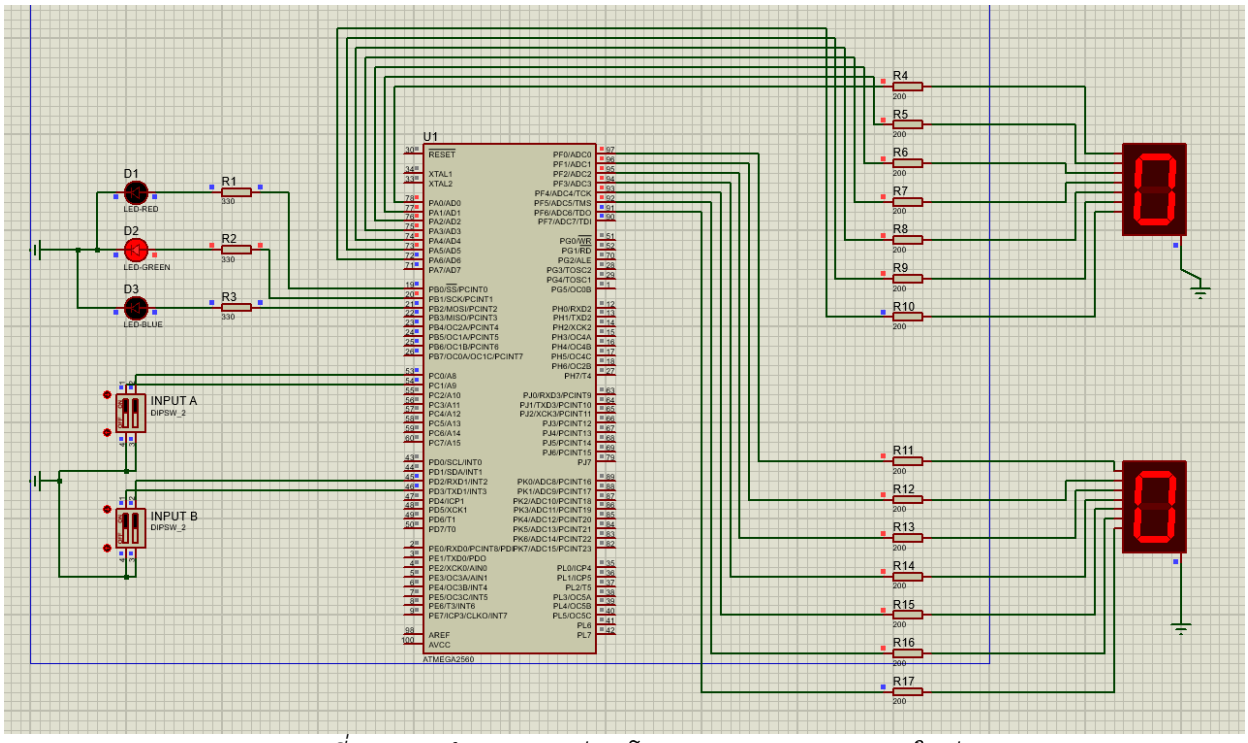
ตารางแสดงการเปรียบเทียบค่า และผลลัพธ์ทุกกรณี

INPUT A	INPUT B	สีของ LED
00	00	เขียว
01	01	เขียว
10	10	เขียว
11	11	เขียว
01	00	น้ำเงิน
10	00	น้ำเงิน
11	00	น้ำเงิน
10	01	น้ำเงิน
11	01	น้ำเงิน
11	10	น้ำเงิน
00	01	แดง
00	10	แดง
01	10	แดง
00	11	แดง
01	11	แดง
10	11	แดง

แบบจำลองวงจรผ่านโปรแกรม Proteus

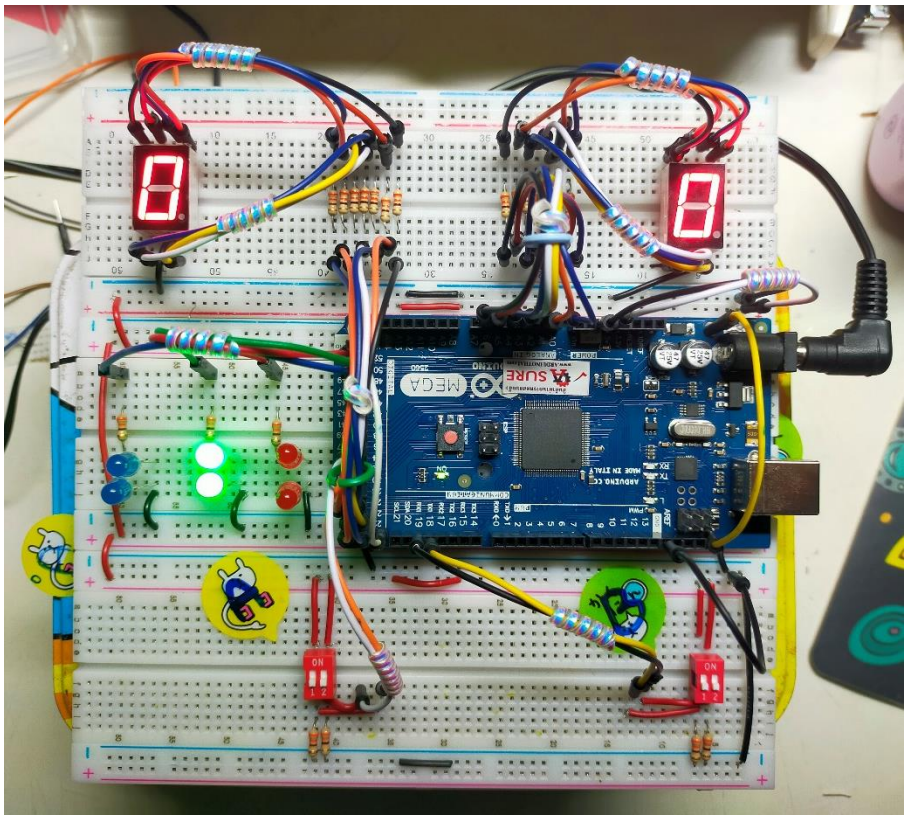


รูปภาพที่ 1 ภาพจำลองวงจรผ่าน โปรแกรม Proteus (วงจรเก่า)

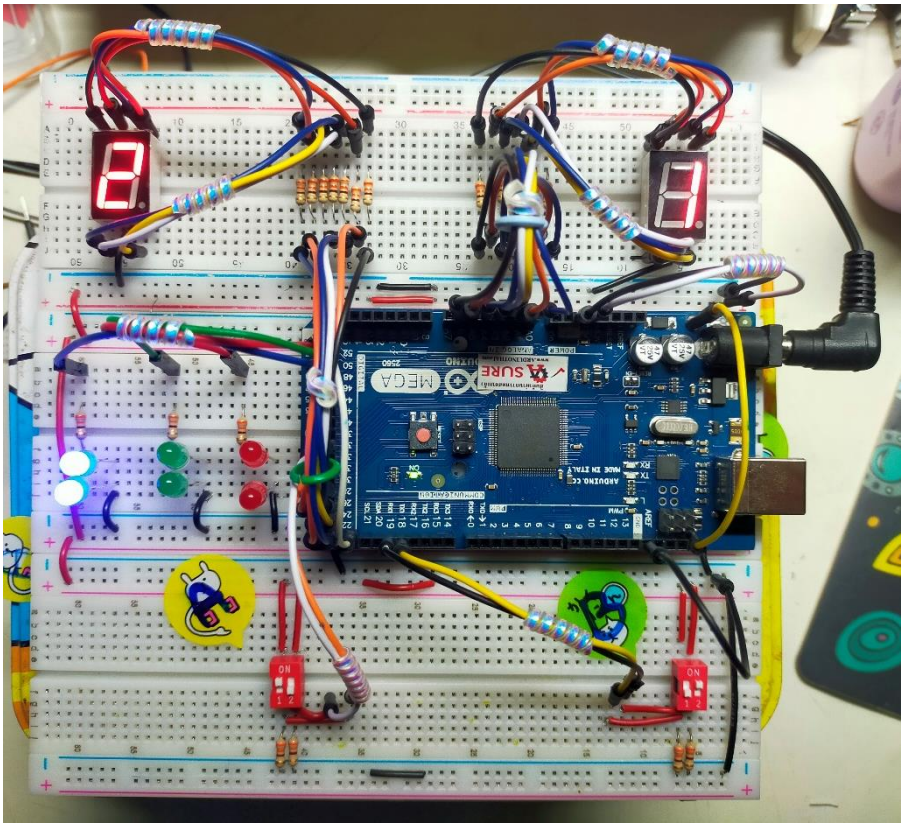


รูปภาพที่ 2 ภาพจำลองวงจรผ่าน โปรแกรม Proteus (วงจรใหม่)

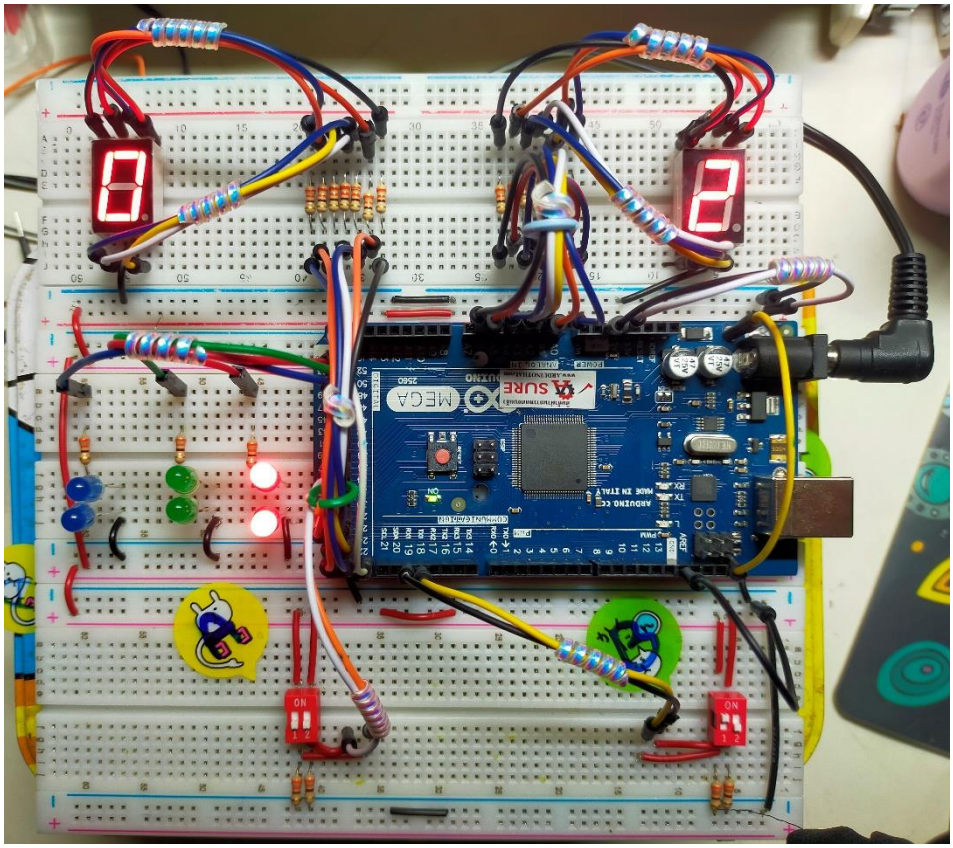
ตัวอย่างวงจรเปรียบเทียบค่า ทั้ง 3 กรณี จากวงจรจริง



รูปภาพที่ 2 เมื่อ Input B = Input A จะแสดงไฟ LED สีเขียว



รูปภาพที่ 3 เมื่อ Input B < Input A จะแสดงไฟ LED สีน้ำเงิน



รูปภาพที่ 4 เมื่อ Input B > Input A จะแสดงไฟ LED สีแดง

โปรแกรมภาษาแอสเซมบลี

```
.INCLUDE"m2560def.inc"          // เรียกใช้คลังโปรแกรมชื่อ m2560def.inc
.EQU ALL_PIN_OUT = 0xff         // กำหนดให้ ALL_PIN_OUT มีค่าเท่ากับ 0xff
.EQU ALL_PIN_IN = 0x00          // กำหนดให้ ALL_PIN_OUT มีค่าเท่ากับ 0x00
.DEF STATUS = R16               // กำหนดชื่อสัญลักษณ์ตัวแปร STATUS ให้กับ R16
.DEF VAR_A = R17                // กำหนดชื่อสัญลักษณ์ตัวแปร VAR_A ให้กับ R17
.DEF VAR_B = R18                // กำหนดชื่อสัญลักษณ์ตัวแปร VAR_B ให้กับ R18
.DEF TMP_A   = R19              // กำหนดชื่อสัญลักษณ์ตัวแปร TMP_A ให้กับ R19
.DEF TMP_B   = R20              // กำหนดชื่อสัญลักษณ์ตัวแปร TMP_B ให้กับ R20
.CSEG                           // เริ่มต้นการทำงานในส่วนของ code segment
.ORG 0x0000                     // เริ่มต้นโปรแกรมที่ตำแหน่ง 0x0000
        rjmp RESET              // กระโดดไปยัง RESET
RESET:
    ldi     STATUS, ALL_PIN_OUT
    out     DDRB, STATUS        // กำหนดให้ PORT B ทำหน้าที่เป็น Output สถานะ ผ่าน
                                LED
    ldi      VAR_A, ALL_PIN_OUT
    out      DDRA, VAR_A        // กำหนดให้ PORT A ทำหน้าที่เป็น output A
    ldi     VAR_A, ALL_PIN_IN
    Out     DDRC, VAR_A         // กำหนดให้ PORT C ทำหน้าที่เป็น Input A

    ldi     VAR_B, ALL_PIN_OUT
    out     DDRF, VAR_B         // กำหนดให้ PORT F ทำหน้าที่เป็น output B
    ldi     VAR_B, ALL_PIN_IN
    out     DDRD, VAR_B         // กำหนดให้ PORT D ทำหน้าที่เป็น Input B

    ldi       TMP_A, 0x00       // TMP_A <= 0
    ldi       TMP_B, 0x00       // TMP_B <= 0

MAIN:
    in        VAR_A, PINC       // อ่านค่าจากพอร์ต C เก็บในตัวแปร VAR_A
    ;ldi         VAR_A,0x03     // test
```

```
        andi    VAR_A,0x03          // กรองค่าที่ได้เพื่อให้เหลือเพียง 2 บิตล่างสุด (0b000011)
        in      VAR_B, PIND         // อ่านค่าจากพอร์ต D เก็บในตัวแปร VAR_B
        ;ldi    VAR_B,0x0c          // test
        andi    VAR_B,0x0c          // กรองค่าที่ได้เพื่อให้เหลือเพียง 2 บิต (0b00001100)
        lsr     VAR_B               // shift bit ไปยังตำแหน่งบิตล่างสุด
        lsr     VAR_B

        jmp     BODY_A              // กระโดดไปที่ BODY_A
BODY3:  jmp     BODY_B              // กระโดดไปที่ BODY_B

SHOW:
        cp      VAR_B, VAR_A        // เปรียบเทียบค่าใน VAR_B กับ VAR_A
        breq    GREEN               // ถ้า VAR_B เท่ากับ VAR_A กระโดดไปทำงานส่วน
                                    GREEN
        brge    RED                 // ถ้า VAR_B มากกว่า VAR_A กระโดดไปทำงานส่วน RED
        jmp     BLUE                // ถ้า VAR_B น้อยกว่า VAR_A กระโดดไปทำงานส่วน
                                    BLUE
RED:
        ldi     STATUS,0x01         // กำหนดค่าให้ตัวแปร STATUS เพื่อส่งตรรกะสูงให้ PB0
        out     PORTB, STATUS       // นำค่าออกทางพอร์ต B
        jmp     END                 // กระโดดไปที่ END
GREEN:
        ldi     STATUS,0x02         // กำหนดค่าให้ตัวแปร STATUS เพื่อส่งตรรกะสูงให้ PB1
        out     PORTB, STATUS       // นำค่าออกทางพอร์ต B
        jmp     END                 // กระโดดไปที่ END
BLUE:
        ldi     STATUS,0x04         // กำหนดค่าให้ตัวแปร STATUS เพื่อส่งตรรกะสูงให้ PB2
        out     PORTB, STATUS       // นำค่าออกทางพอร์ต B
        jmp     END                 // กระโดดไปที่ END

BODY_A:
        ldi     ZL, low (TB_7SEG*2)     // load  Z register low
        ldi     ZH, high (TB_7SEG*2)    // load  Z register high
```

```
        add     ZL, VAR_A               // Z <= Z + VAR_A
        adc     ZH, TMP_A               // R0 <- [Z]
        lpm
        mov     TMP_A, r0               // คัดลอกจาก R0 ไปยัง TMP_A

        out     PORTA, TMP_A
        jmp BODY3                       // กระโดดไปที่ BODY3

BODY_B:
        ldi     ZL, low (TB_7SEG*2)     // load  Z register low
        ldi     ZH, high (TB_7SEG*2)    // load  Z register high

        add     ZL, VAR_B               // Z <= Z + VAR_B
        adc     ZH, TMP_B               // R0 <- [Z]
        lpm
        mov     TMP_B, r0               // คัดลอกจาก R0 ไปยัง TMP_B

        out     PORTF, TMP_B
        jmp SHOW                        // กระโดดไปที่ SHOW

TB_7SEG:
        .DB     0b00111111, 0b00000110  // 0 and 1          - - a - -
        .DB     0b01011011, 0b01001111  // 2 and 3          f       b
        .DB     0b01100110, 0b01101101  // 4 and 5          - - g - -
        .DB     0b01111101, 0b00000111  // 6 and 7          e       c
        .DB     0b01111111, 0b01101111  // 8 and 9          - - d - -

END:
        jmp     RESET                   // กระโดดไปยัง RESET เพื่อตรวจสอบการทำงานอีกครั้ง
.DSEG
.ESEG
```

รายงานผลการทดลอง

จากการที่ได้ทดลองจำลองวงจรและต่อวงจรจริง ผลการทดลองที่ได้ทำงานถูกต้องตามที่ระบุไว้

วิดีโอการทดลอง : https://youtu.be/A1jh04XZFqE