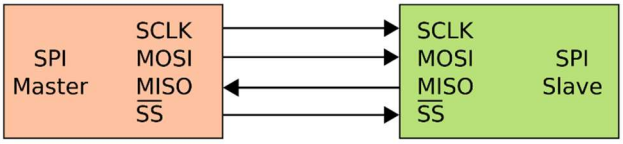
ปฏิบัติการบน RaspberryPi ครั้งที่ 8: การเชื่อมต่อกับอุปกรณ์ผ่าน SPI

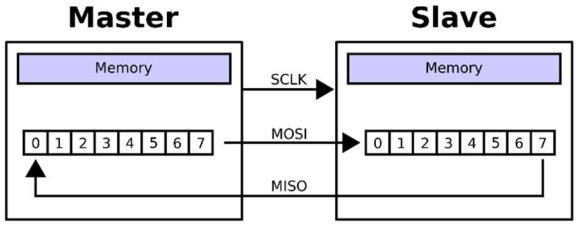
หลักการพื้นฐานของบัส SPI

บัส SPI (Serial Peripheral Interface) ถูกพัฒนาขึ้นมาโดยบริษัท Motorola ในทศวรรษที่ 1980 และในปัจจุบันนี้ได้กลายมาเป็นมาตรฐานโดยปริยายที่มีการใช้กันกว้างขวางมาก ตัวอย่างการใช้งานได้แก่เป็นมาตรฐานการเชื่อมต่อของ SDcard และจอแสดงผล LCD ขนาดเล็ก รวมถึงใช้เพื่อเชื่อมต่อกับวงจรสำหรับสื่อสารแบบอื่นเช่น CAN เป็นต้น

ลักษณะการเชื่อมต่อของ SPI เป็นในลักษณะของ Master-Slave โดยจะมีอุปกรณ์ที่ทำหน้าที่เป็น master เพื่อส่งข้อมูลและสัญญาณนาฬิกาไปยัง slave รวมทั้งอ่านข้อมูลที่ส่งกลับมาในเวลาเดียวกัน



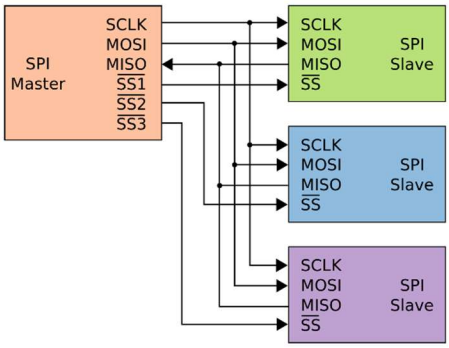
ลักษณะการเชื่อมต่อของ SPI



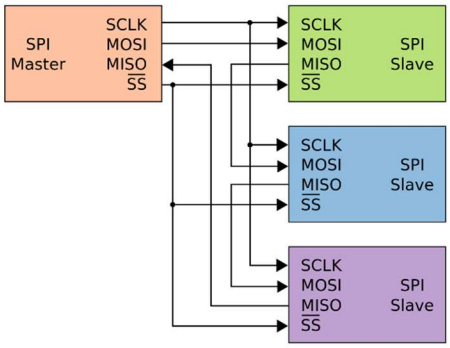
กลไกการทำงานภายในของวงจรภายใน

กลไกการทำงานพื้นฐานของ SPI นั้นถูกออกแบบมาเพื่อให้ง่ายและรวดเร็วในการทำงาน โดยวงจรด้าน Master และ Slave จะประกอบไปด้วย shift register ขนาด 8 บิต (หรือ 9 บิต) และควบคุมการ shift และส่งข้อมูลออกไป และรับข้อมูลกลับเข้ามาโดยใช้สัญญาณนาฬิกาที่สร้างโดยฝั่ง Master ผ่านขา SCLK ส่วนข้อมูลที่ส่งออกจาก Master ไปยัง Slave จะส่งผ่านขา MOSI (Master-Out-Slave-In) และข้อมูลที่รับกลับเข้ามาจะรับทางขา MISO (Master-In-Slave-Out)

ในกรณีที่มีอุปกรณ์ slave จำนวนหลายตัว สามารถต่อวงจรได้ในลักษณะแบบขนานกัน หรือแบบ daisy chain (หรือเรียกว่า ring) ดังรูปด้านล่าง

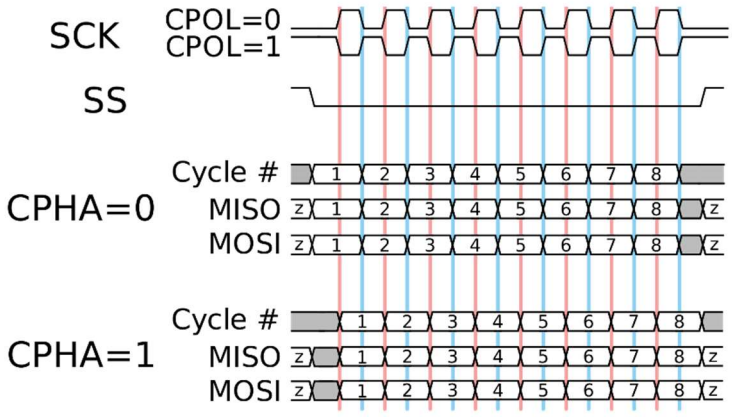


(ก) การเชื่อมต่อแบบขนาน



(ข) การเชื่อมต่อแบบ daisy-chain

สำหรับการเชื่อมต่อแบบขนานนั้น ขาส่งข้อมูลของ slave (MISO) จะต้องรองรับ tri-state ซึ่งจะต้องปล่อยขาลอยหากไม่ได้เลือกอุปกรณ์ด้วย SS (Signal Select) หากอุปกรณ์ slave ไม่รองรับ จะต้องต่อสัญญาณในลักษณะ daisy-chain ซึ่งจะต้องอาศัยการส่งข้อมูลจาก Master ผ่านไปยัง Slave แล้วให้ Slave ส่งข้อมูลต่อไปยัง Slave ถัดไปในวงไปเรื่อยๆ โดยสังเกตการณ์เชื่อมต่อขา MISO ของ slave ตัวแรกไปยัง MOSI ของ slave ตัวถัดไป (และอาศัยการเขียนซอฟต์แวร์ให้ส่งข้อมูลจาก slave ตัวแรกออกไปยัง slave ตัวถัดไปประกอบด้วย)



ลักษณะโหมดการทำงานของ SPI

การทำงานของอุปกรณ์ SPI จะแบ่งออกเป็นโหมดการทำงานต่างๆ ตามการตอบสนองของอุปกรณ์ต่อสัญญาณ CPOL (Clock polarity) และ CPHA (Clock phase) โดย CPOL ควบคุมเวลาที่ใช้ในการเขียนหรืออ่านข้อมูลบนบัส ถ้า CPOL เป็น 0 การเขียน/อ่านจะเกิดขึ้นในจังหวะขอบขาขึ้นของสัญญาณขา SCLK ถ้าเป็น 1 การเขียน/อ่าน จะเกิดขึ้นในจังหวะขอบขาลงของ SCLK ส่วน CPHA คุมเวลาที่จะให้ข้อมูลปรากฏขึ้นบนบัสข้อมูล โดยถ้า CPHA เป็น 0 ข้อมูลตัวแรก (บิตสูงสุดของไบต์ข้อมูล) จะปรากฏบนบัสตั้งแต่ช่วง SS (Signal/chip select) เริ่มสั่งการให้อุปกรณ์ slave รับส่งข้อมูล และการเลื่อนบิตถัดๆ ไปจะปรากฏในช่วงขอบขาลงของ SCLK ส่วนกรณี CPHA เป็น 1 จะทำงานกลับกัน โดยการส่งข้อมูลบิตแรกและบิตถัดๆ ไป จะเกิดขึ้นในช่วงขอบขาขึ้นของ SCLK ในกรณีของอุปกรณ์ที่ใช้ไมโครคอนโทรลเลอร์ตระกูล ARM จะมีโหมดต่างๆ ดังนี้

SPI Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

การจัดการบัส SPI บน Raspberry Pi ด้วย pigpio แบบ bitbang

Raspberry PI OS นั้นรองรับการส่งข้อมูลผ่านทาง SPI โดยทำงานเป็นอุปกรณ์ฝั่ง master และมีบัส SPI ให้เชื่อมต่อได้ 2 บัส คือ SPI0 และ SPI1 (ส่วนบัส SPI2 ไม่มีการต่อสัญญาณออกมาภายนอก) และ Raspberry Pi เวอร์ชัน 4 เพิ่ม SPI เข้ามาอีก 4 บัส (SPI3 ถึง SPI6)
โดยมีขาสัญญาณ SPI ดังนี้

SPI0		
SPI Function	Header Pin	GPIO
MOSI	19	GPIO 10
MISO	21	GPIO 9
SCLK	23	GPIO 11
CE0	24	GPIO 8
CE1	26	GPIO 7

SPI1		
SPI Function	Header Pin	GPIO
MOSI	38	GPIO 20
MISO	35	GPIO 19
SCLK	40	GPIO 21
CE0	12	GPIO 18
CE1	11	GPIO 17
CE2	36	GPIO 16

สำหรับไลบรารี pigpio นั้น มีฟังก์ชันสำหรับจัดการกับบัส SPI ทั้งแบบจัดการอาศัยกลไกทางฮาร์ดแวร์ (ซึ่งจะใช้ได้กับขา SPI ที่ Raspberry Pi มีให้เท่านั้น) กับกลไกแบบ bitbang ที่สามารถนำขา GPIO ใดๆ มาทำหน้าที่เป็นขา SPI ได้ โดยฟังก์ชันที่ใช้จัดการแบบ bitbang กับพอร์ต GPIO เพื่อให้เป็น SPI มีดังนี้

```
int bbSPIOpen(unsigned CS, unsigned MISO, unsigned MOSI, unsigned SCLK,
              unsigned baud, unsigned spiFlags);
```

กำหนดขา GPIO เพื่อนำมาใช้เป็น SPI และเริ่มต้นการทำงานของ SPI ในโหมด bitbang กับขาชุดดังกล่าว

CS, MISO, MOSI, SCLK	หมายเลข GPIO ที่ใช้เป็นสัญญาณ CS, MISO, MOSI และ SCLK ตามลำดับ
baud	ความเร็วของสัญญาณนาฬิกา มีค่าตั้งแต่ 50 ถึง 250000
spiFlags	แฟล็กของการทำงานของ SPI ของชาชุดดังกล่าว มีบิตต่างๆ กำหนดไว้ดังนี้

21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	R	T	0	0	0	0	0	0	0	0	0	0	0	p	m	m

บิตต่างๆ มีค่าดังนี้

p	0	CS เป็น low เมื่อส่งสัญญาณ
	1	CS เป็น high เมื่อส่งสัญญาณ
m	โหมดการทำงาน มีค่าตั้งแต่ 0-3 ตามโหมดมาตรฐานของ SPI	
T	1	ส่งบิตต่ำสุดก่อน
	0	ส่งบิตศุงสุดก่อน
R	1	รับบิตต่ำสุดก่อน
	0	รับบิตสูงสุดก่อน

ค่ากลับคืน	OK	ทำงานได้ตามปกติ
	PI_BAD_USER_GPIO	หมายเลขพอร์ตที่กำหนดอยู่นอกขอบเขตที่รองรับ
	PI_BAD_SPI_BAUD	ค่า baud rate ที่กำหนดอยูนอกขอบเขตที่รองรับ
	PI_GPIO_IN_USE	พอร์ตที่กำหนดนั้นถูกนำไปใช้งานอื่นอยู่ก่อนแล้ว

```
int bbSPIClose(unsigned CS);
```

ปิดการทำงานของ SPI ในชุดพอร์ตที่กำหนดไว้

CS	หมายเลข GPIO ที่ใช้เป็นสัญญาณ CS ตอนเปิดการทำงานด้วย bbSPIOpen()	
ค่ากลับคืน	OK	ทำงานได้ตามปกติ
	PI_BAD_USER_GPIO	หมายเลขพอร์ตที่กำหนดอยู่นอกขอบเขตที่รองรับ
	PI_NOT_SPI_GPIO	พอร์ตดังกล่าวไม่ใช่ขา CS ที่ใช้ตอนเปิดด้วย bbSPIOpen()

```
int bbSPIXfer(unsigned CS, char *inBuf, char *outBuf, unsigned count);
```

ส่งข้อมูลและรับข้อมูลผ่านพอร์ต SPI ที่กำหนดไว้

CS	หมายเลข GPIO ที่ใช้เป็นสัญญาณ CS ตอนเปิดการทำงานด้วย bbSPIOpen()
inBuf	อะเรย์แบบชนิด char ของข้อมูลที่ส่ง
outBuf	อะเรย์แบบชนิด char รับข้อมูลที่ส่งกลับคืนมา

count	จำนวนข้อมูลที่จะรับ/ส่ง	
ค่ากลับคืน	OK	ทำงานได้ตามปกติ
	PI_BAD_USER_GPIO	หมายเลขพอร์ตที่กำหนดอยู่นอกขอบเขตที่รองรับ
	PI_BAD_POINTER	ค่าอ้างอิงชี้ไปยังพื้นที่ที่ไม่สามารถเขียน/อ่านได้
	PI_NOT_SPI_GPIO	พอร์ตดังกล่าวไม่ใช่ขา CS ที่ใช้ตอนเปิดด้วย bbSPIOpen()

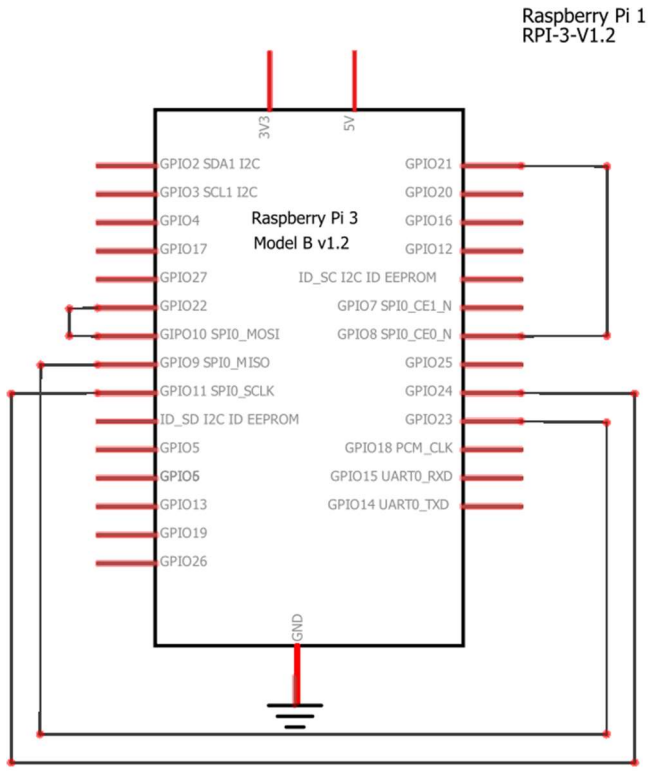
การเขียนโปรแกรมจำลองการทำงานของอุปกรณ์ Slave ด้วยการใช้กลไกอินเทอร์รัปต์และ Bit-banging บน Raspberry Pi เพื่อทดสอบการทำงานของบัส SPI บน Raspberry Pi

เนื่องจากตัว Raspberry Pi นั้นไม่รองรับการทำงานในรูปแบบ Slave เราจึงไม่สามารถนำเอา Raspberry Pi สองตัวมาต่อเพื่อสื่อสารกันผ่านบัส SPI ได้ แต่เพื่อการศึกษาถึงกลไกการรับส่งข้อมูลผ่านบัส SPI ในเบื้องต้น เราจะเขียนโปรแกรมเพื่อจำลองการรับและการสร้างสัญญาณของบัส MISO และ MOSI ทางฝั่ง slave บนตัว Raspberry Pi โดยใช้พอร์ต GPIO ตามปกติ และเพื่อให้กลไกทำงานได้แม่นยำมากขึ้น เราจะอาศัยการจัดการอินเทอร์รัปต์ของฝั่งขา CE และ SCLK ที่เราจะจำลองขึ้นมาโดยใช้ขา GPIO ด้วยเช่นกัน

สำหรับการทดลองในขั้นต้น เราจะใช้บอร์ด Raspberry Pi เพียงตัวเดียว โดยเราจะใช้ขา GPIO 21 ถึง GPIO 24 เพื่อจำลองขาสัญญาณ SPI ฝั่ง slave โดยกำหนดให้ GPIO21 เป็น CE (SS) ขา GPIO22 เป็น MOSI (ฝั่ง slave ซึ่งจะทำหน้าที่รับข้อมูล) ขา GPIO32 เป็น MISO (ทำหน้าที่ส่งข้อมูล) และขา GPIO24 เป็น SCLK ในขณะที่เราใช้ขาในชุดของ SPI0 เพื่อทำหน้าที่เป็นฝั่ง Master ดังรูป

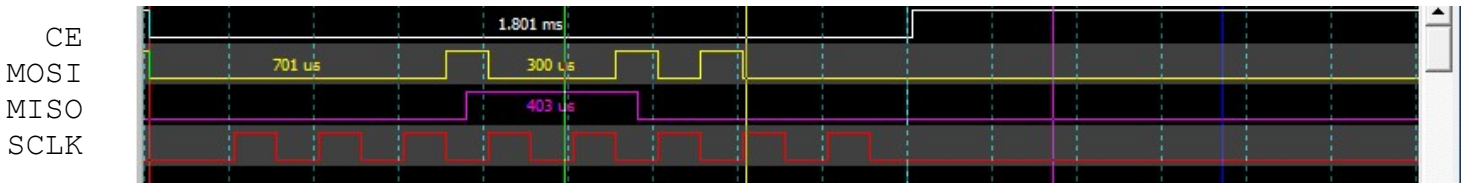
จากนั้นให้ทดลองใช้โปรแกรมตัวอย่างต่อไปนี้เพื่อศึกษาการทำงานของบัส SPI ในเบื้องต้น

```
#include <pigpio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdarg.h>
#include <stdint.h>
#include <stdio.h>
#include <time.h>
#include <sched.h>
#include <string.h>

#define SSPI_CE      21  //GPIO 21
#define SSPI_SCLK    24  //GPIO 24
#define SSPI_MOSI    22  //GPIO_22
#define SSPI_MISO    23  //GPIO_23
#define INFINITE     100000000

struct SSPI_type{
    unsigned char cpol;
    unsigned char cpha;
    unsigned char ready;
    unsigned char datain;
    unsigned char dataout;
}SSPI_data;

void gpio_stop(int sig);
int running = 1;

int SSPI_init();
void isr_SSPI_CE(int gpio,int level,uint32_t tick);
void isr_SSPI_SCLK(int gpio,int level,uint32_t tick);

int main(){
    unsigned char data[2],rdata[2];
    int count=0;
      int spi;
```



```
    // Slave side
    if(SSPI_init()<0) return -1;
       signal(SIGINT,gpio_stop);
    SSPI_data.cpha = 0;
    SSPI_data.cpol = 0;
    // Master side
       if(bbSPIOpen(8,9,10,11,5000,0)<0) return -1;

    while(running){
        data[0] = (unsigned char)count;
        printf("Master:Sending out data %.2X  ",data[0]);
             bbSPIXfer(8,(char *)data,(char *)rdata,1);
        printf("Master:Receiving in data %.2X\n",rdata[0]);
        count++;
        sleep(1);
        printf("Slave :Receiving in data %.2X\n",SSPI_data.datain);
        SSPI_data.dataout=SSPI_data.datain+3;
        sleep(1);
    }
       bbSPIClose(8);
       gpioTerminate();
    return 0;
}

int SSPI_init(){
    SSPI_data.cpol = SSPI_data.cpha = 0;
    SSPI_data.ready=0;

       if(gpioInitialise() < 0) return -1;

       gpioSetMode(SSPI_CE,PI_INPUT);
    gpioSetPullUpDown(SSPI_CE,PI_PUD_OFF);
    gpioSetMode(SSPI_SCLK,PI_INPUT);
    gpioSetPullUpDown(SSPI_SCLK,PI_PUD_OFF);
    gpioSetMode(SSPI_MOSI,PI_INPUT);
    gpioSetPullUpDown(SSPI_MOSI,PI_PUD_OFF);
    gpioSetMode(SSPI_MISO,PI_OUTPUT);

    if(gpioSetISRFunc(SSPI_CE,EITHER_EDGE,INFINITE,isr_SSPI_CE)<0) return -1;
    if(gpioSetISRFunc(SSPI_SCLK,EITHER_EDGE,INFINITE,isr_SSPI_SCLK)<0) return -1;

    return 0;
}

void isr_SSPI_CE(int gpio,int level,uint32_t tick){
       if(level==2) return;
    if(gpioRead(SSPI_CE))
//     if(level)
        SSPI_data.ready=1;
    else{
        SSPI_data.datain=0;
        if(!SSPI_data.cpha){
            if(SSPI_data.dataout&0x80)
                gpioWrite(SSPI_MISO,1);
            else
                gpioWrite(SSPI_MISO,0);
            SSPI_data.dataout <<= 1;
        }
    }

}

void isr_SSPI_SCLK(int gpio,int level,uint32_t tick){
       if(level==2) return;
    if(gpioRead(SSPI_CE))
        return;

    if(gpioRead(SSPI_SCLK)){
```

```
//      if(level){
         if(SSPI_data.cpha){
             if(SSPI_data.dataout&0x80)
                 gpioWrite(SSPI_MISO,1);
             else
                 gpioWrite(SSPI_MISO,0);
             SSPI_data.dataout <<= 1;
         }else{
             SSPI_data.datain <<=1;
             SSPI_data.datain |=gpioRead(SSPI_MOSI);
         }
     }else{
         if(SSPI_data.cpha){
             SSPI_data.datain <<=1;
             SSPI_data.datain |=gpioRead(SSPI_MOSI);
         }else{
             if(SSPI_data.dataout&0x80)
                 gpioWrite(SSPI_MISO,1);
             else
                 gpioWrite(SSPI_MISO,0);
             SSPI_data.dataout <<= 1;
         }
     }
}

void gpio_stop(int sig){
    printf("User pressing CTRL-C");
    running = 0;
}
```



ตัวอย่างลักษณะสัญญาณของขา SPI ฝั่ง master จากโปรแกรมที่ให้ไว้

รายละเอียดที่น่าสนใจเกี่ยวกับการจำลอง SPI ฝั่ง slave จากโปรแกรมตัวอย่าง

- ตัวอย่างโปรแกรมข้างบนนี้ ฝั่ง SPI master นั้นเราใช้กลไกการทำ bitbang ของไลบรารี pigpio ในที่นี้เราไม่สามารถใช้กลไกทางฮาร์ดแวร์ตามปกติได้ เนื่องจากค่า baud rate ต่ำสุดที่รองรับโดยไลบรารีนั้น เร็วเกินกว่าที่ฝั่ง slave ที่เราสร้างขึ้นจากรับข้อมูลได้ทัน
- เนื่องจากเราใช้กลไกการสร้างสัญญาณแบบ bit-banging ผสมผสานกับการควบคุมการเกิดสัญญาณอาศัยการอินเทอร์รัปต์ ส่งผลให้การทำงานของซอฟต์แวร์ไม่สามารถทำงานได้อย่างรวดเร็วเท่ากับการทำงานของฮาร์ดแวร์โดยตรง ตามปกติแล้วบัส SPI สามารถทำงานได้สูงถึง 10Mbps หรือมากกว่า แต่ด้วยข้อจำกัดของการจำลองนี้ทำให้เราสามารถส่งสัญญาณได้ไม่สูงมากนัก จากโปรแกรมตัวอย่างนี้เราใช้สัญญาณนาฬิกาเพียง 5kHz เท่านั้น (ได้แค่ 5kbps)
- เรากำหนดขา CE (ตัวเลือกการทำงาน slave) SCLK (ขาสัญญาณนาฬิกาฝั่งรับ) และ MOSI (ขารับข้อมูลจาก master) เป็นอินพุต ในขณะที่ขา MISO (ฝั่งส่งข้อมูลไป master) เป็นขาเอาต์พุต แต่เราจะเริ่มกำหนดเมื่อเริ่มส่งข้อมูลบิตแรก และเราจะเปลี่ยนสถานะกลับเป็นอินพุตเมื่อส่งข้อมูลเสร็จสิ้น ทั้งนี้เผื่อในกรณีที่มีการต่ออุปกรณ์ slave มากกว่าหนึ่งตัวแบบขนาน จะได้ไม่เกิดการชนกันของสัญญาณขาเอาต์พุตของ slave ทั้งสองตัว
- กำหนดขา CE และ SCLK ให้กำเนิดสัญญาณอินเทอร์รัปต์โดยกำหนดให้สร้างสัญญาณทั้งฝั่งขอบขาขึ้น และขอบขาลง โดยเราสร้างฟังก์ชัน isr_SSPI_CE ไว้ตอบสนองอินเทอร์รัปต์ของขา CE (ฝั่ง slave) และ isr_SSPI_SCLK ไว้ตอบสนองอินเทอร์รัปต์ของขา SCLK (ฝั่ง slave)
- ในส่วนของฟังก์ชันจัดการอินเทอร์รัปต์ของขา CE (ฝั่ง slave) เราตรวจสอบสถานะของขา ณ ขณะนั้น (ซึ่งจะเป็นสถานะหลังจากการเปลี่ยนแปลงแล้ว)

- ถ้าสถานะเป็น 1 หมายถึงจบสิ้นการส่งข้อมูลครบทุกบิตแล้ว เราก็จะนิยามตัวแปรสมาชิก ready ในโครงสร้าง SSPI_DATA ให้เป็น 1 เพื่อเอาไว้แจ้งว่าข้อมูลได้รับครบแล้ว (ในตัวอย่างนี้ไม่ได้นำมาใช้งาน)
- ถ้าสถานะเป็น 0 แสดงว่าในขณะนี้อยู่ในช่วงของการรับข้อมูลจาก master ซึ่งในกรณีของ cpha เป็น 0 นั้น ข้อมูลฝั่งขา master และฝั่ง slave จะต้องปรากฏบนบัสทันที ดังนั้นในฝั่ง slave ที่เราจำลองขึ้นนี้ เราจึงต้องเริ่มส่งข้อมูลบิตแรก ซึ่งเป็นบิตสูงสุด (b7) ออกไปโดยทันที จากนั้นเราเลื่อนบิตขึ้นหนึ่งบิต เพื่อรอการทำงานในขั้นตอนถัดไป

- ในส่วนฟังก์ชันจัดการอินเทอร์รัปต์ของขา SCLK (ฝั่ง slave) เราตรวจสอบสถานะของขา CE เพื่อในกรณีที่มีการเชื่อมต่ออุปกรณ์นี้มีการเชื่อมต่ออุปกรณ์ slave แบบขนาน เราจะได้ไม่ต้องสนใจข้อมูลที่ส่งมาจาก slave อีกตัวหนึ่ง
- ในกรณีที่สัญญาณขา CE เป็น low (ซึ่งหมายถึงมีการเลือกทำงาน slave ตัวที่เราจำลองขึ้นนี้) เราจะตรวจสอบสัญญาณของขา SCLK ว่าเป็นสัญญาณ 1 (ช่วงหลังขอบขาขึ้น) หรือสัญญาณ 0 (ช่วงหลังขอบขาลง) เพื่อจัดการตามกลไกการรับส่งข้อมูลของบัส SPI
  - หากSCLK เป็น 1 (ช่วงขอบขาขึ้น) เราจะตรวจสอบค่า CPHA ต่อไป
    - ถ้า CPHA เป็นศูนย์ แสดงว่าในขณะนี้ข้อมูลฝั่ง master ต้องปรากฏบนบัส MOSI ให้พร้อมอ่านได้แล้ว เราจะเลื่อนบิตเดิมขึ้นแล้วอ่านบิตดังกล่าวต่อเข้ามาที่บิตล่างสุด (เลื่อนค่าไปทางซ้ายและนำค่าใหม่ต่อเข้ามาทางด้านขวา)
    - ถ้า CPHA เป็นหนึ่ง แสดงว่าในขณะนี้ slave จะต้องส่งข้อมูลบิตถัดไปในทันที เราจะนำค่าบิตสูงสุดส่งออกไป แล้วเลื่อนทุกบิตขึ้นหนึ่งตำแหน่ง (เพื่อพร้อมกันเลื่อนบิตและส่งบิตตัวถัดไปในรอบถัดไป)
  - หาก SCLK เป็น 0 (ช่วงขอบขาลง) เราจะตรวจสอบค่า CPHA ต่อไป
    - ถ้า CPHA เป็นหนึ่ง แสดงว่าในขณะนี้ข้อมูลฝั่ง master ต้องปรากฏบนบัส MOSI ให้พร้อมอ่านได้แล้ว เราจะเลื่อนบิตเดิมขึ้นแล้วอ่านบิตดังกล่าวต่อเข้ามาที่บิตล่างสุด (เลื่อนค่าไปทางซ้ายและนำค่าใหม่ต่อเข้ามาทางด้านขวา)
    - ถ้า CPHA เป็นศูนย์ แสดงว่าในขณะนี้ slave จะต้องส่งข้อมูลบิตถัดไปในทันที เราจะนำค่าบิตสูงสุดส่งออกไป แล้วเลื่อนทุกบิตขึ้นหนึ่งตำแหน่ง (เพื่อพร้อมกันเลื่อนบิตและส่งบิตตัวถัดไปในรอบถัดไป)

ปฏิบัติการ: Raspberry Pi สองตัวคุยกันผ่านบัส SPI และซอฟต์แวร์ SPI ที่จำลองเป็น slave

อุปกรณ์ที่ต้องการ

- บอร์ด Raspberry Pi พร้อมอแดปเตอร์
- สายจัมพ์จากขา GPIO ของบอร์ด
- โปรโตบอร์ด และสายจัมพ์อีกตามต้องการ

ให้นักศึกษาสองกลุ่มมาพัฒนาโปรแกรมร่วมกัน โดยกลุ่มหนึ่งพัฒนาโปรแกรมตัวฝั่ง master เพื่อทำการส่งข้อมูลทีละอักขระไปยังฝั่ง slave และฝั่งslave ให้นำเอาข้อมูลจาก master มาแสดงบนจอภาพ และในทางกลับกัน ให้เตรียมข้อมูลไว้ล่วงหน้าเพื่อส่งคืนกลับไปยังผู้ส่งในช่วงเวลาที่ master ส่งข้อมูลมาด้วยเช่นกัน โดยให้เตรียมข้อมูลไว้ก่อนในอะเรย์ของ char ส่วนฝั่ง master ก็ให้เตรียมข้อมูลไว้ในอะเรย์ของ char เช่นกันเพื่อส่งข้อมูลไปยัง slave และรับข้อมูลกลับมาเก็บไว้ในอะเรย์อีกตัวหนึ่งเพื่อแสดงข้อมูลจากฝั่ง slave มาบนจอภาพ

กำหนดขนาดอะเรย์ของฝั่งผู้รับและฝั่งผู้ส่งเป็น 80 อักขระ และจบข้อความด้วย NULL character ส่วนการส่งข้อมูลฝั่ง master นั้น เพื่อให้การรับข้อมูลทำได้ครบทุกอักขระ ให้ส่งข้อมูลทั้ง 80 อักขระ (ทั้งอะเรย์) ไปยังฝั่ง slave ทั้งนี้เนื่องจากข้อมูลอักขระแรกของฝั่ง slave จะได้กลับมาหลังจากที่ master ได้ส่งข้อมูลไปแล้ว ดังนั้นข้อมูลอักขระแรกของ slave จะเป็นไบต์ที่สอง ไม่ใช่ไบต์แรก และความยาวสูงสุดของข้อมูงฝั่งรับจะได้แค่ 79 อักขระ (รวม NULL character)

การจัดการบัส SPI บน Raspberry Pi ด้วย pigpio กับบัส SPI บน Raspberry Pi

สำหรับการจัดการบัส SPI ผ่านไลบรารี pigpio กับฮาร์ดแวร์ SPI ของ Raspberry Pi นั้น มีฟังก์ชันให้ใช้งานดังนี้

```
int spiOpen(unsigned spiChan, unsigned baud, unsigned spiFlags);
```

เริ่มต้นการทำงานของ SPI ในโหมด bitbang กับ SPI ที่กำหนด

spiChan หมายเลขบัส SPI ที่กำหนด

baud ความเร็วของสัญญาณนาฬิกา มีค่าตั้งแต่ 32k ถึง 125M (ค่าที่สูงกว่า 30M อาจจะทำงานได้ไม่ถูกต้อง)

spiFlags แฟล็กของการทำงานของ SPI ของชาชุดดังกล่าว มีบิตต่างๆ กำหนดไว้ดังนี้

21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
b	b	b	b	b	b	R	T	n	n	n	n	W	A	u2	u1	u0	p2	p1	p0	m	m

บิตต่างๆ มีค่าดังนี้

mm โหมดการทำงาน มีค่าตั้งแต่ 0-3 ตามโหมดมาตรฐานของ SPI

px 0 กำหนดให้ขา CEx ทำงานแบบ active low

1 กำหนดให้ขา CEx ทำงานแบบ active high

ux เปิดการใช้งานขา CEx ถ้ากำหนดค่าบิตดังกล่าวเป็น 0 เช่น p0 ถ้ากำหนดเป็น 0 หมายความว่าเซ็ต GPIO08 ให้ทำงานเป็น CE0 (แทนที่จะเป็น GPIO08 ตามปกติ เป็นต้น)

A 0 ใช้ค่านี้สำหรับ SPI ตัวหลัก (SPI0)

1 ใช้ค่านี้สำหรับ SPI ตัวอื่นๆ (SPI1 หรือตัวอื่นที่บอร์ดรองรับ)

W 0 กำหนดใช้งานบัส SPI แบบ 4-wire (มีครบทุกขาทั้ง CS,MOSI,MISO,SCLK)

1 กำหนดใช้งานบัส SPI แบบ 3-wire (ไม่มีขา MISO) (ใช้งานเฉพาะ SPI0 เท่านั้น)

nnnn จำนวนไบต์ (0-15) ที่จะเขียนข้อมูลไปก่อนที่จะเริ่มต้นรับข้อมูลกลับ

T 1 ส่งบิตต่ำสุดก่อน

0 ส่งบิตศุงสุดก่อน

R 1 รับบิตต่ำสุดก่อน

0 รับบิตสูงสุดก่อน

bbbbbb จำนวนบิตข้อมูลที่จะรับส่ง (0-32บิต) ค่าปกติเป็น 0 สำหรับรับส่งแบบ 8 บิต ใช้งานเฉพาะ SPI1 เป็นต้นไป

ค่ากลับคืน

ค่า 0 หรือมากกว่า	หมายเลขแฮนเดิลของ SPI บัสที่จะนำไปใช้งานต่อไป
PI_BAD_SPI_CHANNEL	หมายเลขบัส SPI ไม่รองรับ
PI_BAD_SPI_SPEED	ค่า baud rate ที่กำหนดอยู่นอกขอบเขตที่รองรับ
PI_BAD_FLAGS	แฟลกที่กำหนดไม่รองรับ
PI_NO_AUX_SPI	บอร์ด Raspberry Pi ที่ใช้ไม่รองรับ SPI1 (Aux SPI)
PI_SPI_OPEN_FAILED	ไม่สามารถเปิดใช้พอร์ต SPI เพื่อทำงานได้

```
int spiClose(unsigned handle);
```

ปิดการทำงานของ SPI ในชุดพอร์ตที่กำหนดไว้

handle	หมายเลขแฮนเดิลของ SPI ที่ได้จาก spiOpen()	
ค่ากลับคืน	ศูนย์	ทำงานได้ตามปกติ
	PI_BAD_HANDLE	หมายเลขแฮนเดิลไม่ถูกต้อง ไม่ได้มาจาก bbSPIOpen()

```
int spiRead(unsigned handle, char *buf, unsigned count);
```

รับข้อมูลผ่านพอร์ต SPI ที่กำหนดไว้

handle	หมายเลขแฮนเดิลที่ได้จาก spiOpen()	
buf	อะเรย์แบบชนิด char รับข้อมูลที่ส่งกลับคืนมา	
count	จำนวนข้อมูลที่จะรับ	
ค่ากลับคืน	ค่ามากกว่าศูนย์	จำนวนข้อมูลที่อ่านได้จริง
	PI_BAD_HANDLE	หมายเลขแฮนเดิลไม่ถูกต้อง
	PI_BAD_SPI_COUNT	ขนาดข้อมูลที่จะรับไม่ถูกต้อง
	PI_SPI_XFER_FAILED	การรับส่งข้อมูลเกิดความผิดพลาด

```
int spiWrite(unsigned handle, char *buf, unsigned count);
```

ส่งข้อมูลผ่านพอร์ต SPI ที่กำหนดไว้

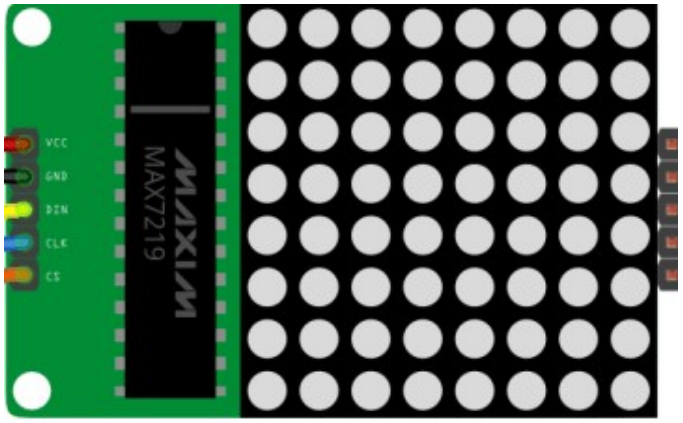
handle	หมายเลขแฮนเดิลที่ได้จาก spiOpen()	
buf	อะเรย์แบบชนิด char เก็บข้อมูลที่จะใช้ส่ง	
count	จำนวนข้อมูลที่จะส่ง	
ค่ากลับคืน	ค่ามากกว่าศูนย์	จำนวนข้อมูลที่ส่งไป
	PI_BAD_HANDLE	หมายเลขแฮนเดิลไม่ถูกต้อง
	PI_BAD_SPI_COUNT	ขนาดข้อมูลที่จะรับไม่ถูกต้อง
	PI_SPI_XFER_FAILED	การรับส่งข้อมูลเกิดความผิดพลาด

```
int spiXfer(unsigned handle, char *txBuf, char *rxBuf, unsigned count);
```

รับส่งข้อมูลผ่านพอร์ต SPI ที่กำหนดไว้

handle	หมายเลขแฮนเดิลที่ได้จาก spiOpen()	
txBuf	อะเรย์แบบชนิด char เก็บข้อมูลที่จะใช้ส่ง	
rxBuf	อะเรย์แบบชนิด char เก็บข้อมูลที่จะรับ	
count	จำนวนข้อมูลที่จะรับ/ส่ง	
ค่ากลับคืน	ค่ามากกว่าศูนย์	จำนวนข้อมูลที่ส่งไป
	PI_BAD_HANDLE	หมายเลขแฮนเดิลไม่ถูกต้อง
	PI_BAD_SPI_COUNT	ขนาดข้อมูลที่จะรับไม่ถูกต้อง
	PI_SPI_XFER_FAILED	การรับส่งข้อมูลเกิดความผิดพลาด

การติดต่อกับไอซี MAX7219 ผ่านบัส SPI

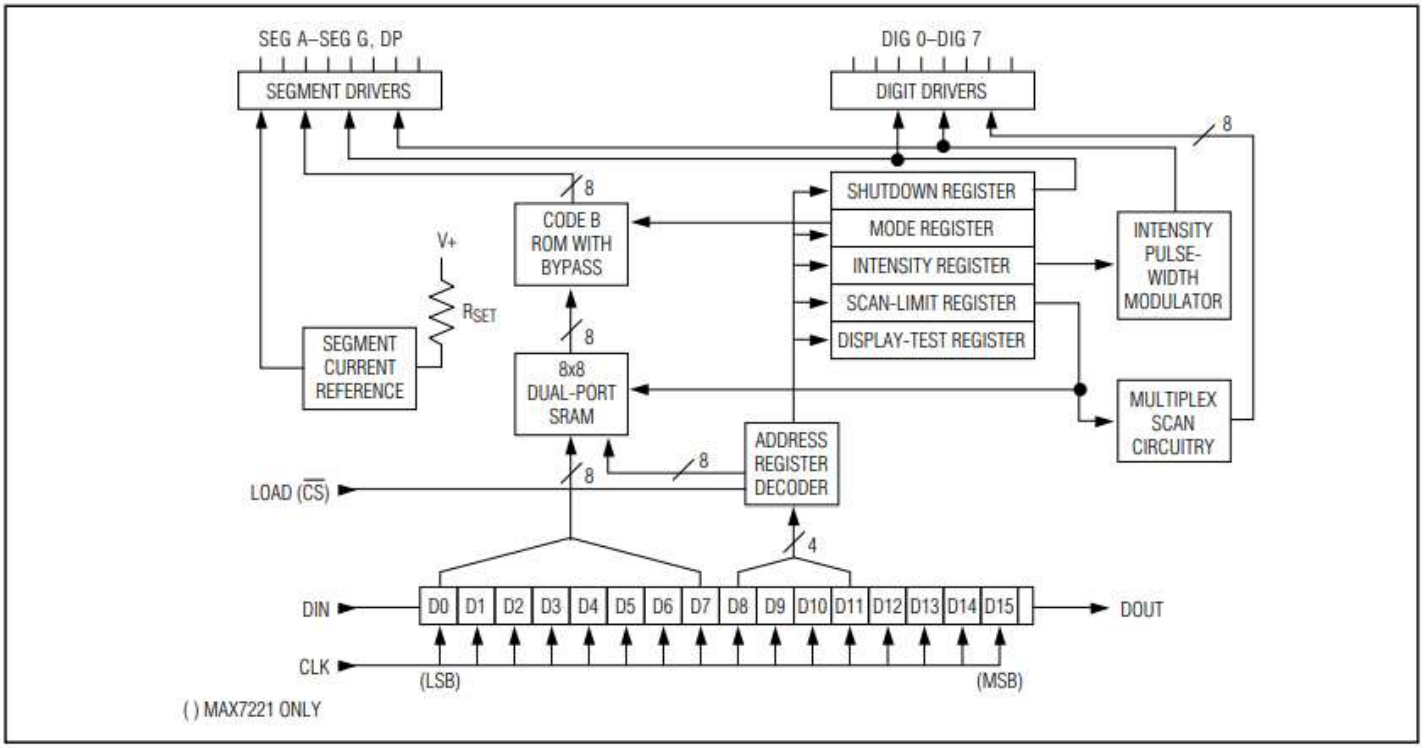


สำหรับอุปกรณ์ที่ไม่จำเป็นต้องมีการส่งข้อมูลกลับมายัง master นั้น อุปกรณ์เหล่านี้มักจะถูกต่อสัญญาณไว้เพียง 3 เส้นด้วยกันคือ MOSI CS และ CLK ส่วน MISO จะไม่มีการต่อสาย ซึ่งบางทีเราก็เรียกว่า 3-wire SPI

สำหรับไอซี MAX7219 นี้ ถูกนำมาใช้งานกันอย่างแพร่หลายสำหรับการแสดงผล 7-segment LED และ 8x8 matrix LED และยังสามารถนำมาต่อพ่วงกัน โดยต่อขา DOUT กับ DIN ของไอซีตัวถัดไป เพื่อเพิ่มขยาย LED ที่จะนำมาใช้แสดงผลได้ด้วย

กลไกการทำงานภายในของ MAX7219 เป็นดังไดอะแกรมต่อไปนี้

Functional Diagram



การรับคำสั่งจาก master นั้น จะอาศัยการ latch ข้อมูลไว้ในเรจิสเตอร์ขนาด 16 บิต โดย 8 บิตล่างจะใช้สำหรับการแสดงผลในแต่ละ segment ของ LED และ8 บิตบน (ใช้งานเฉพาะ 4 บิต) จะถูกนำไปใช้กำหนดหลักของ LED ที่จะใช้แสดง นั่นหมายความว่าไอซี MAX7219 สามารถแสดง 7-segment LED ได้ทั้งหมด 8 หลัก หรือนำมาใช้แสดง LED แบบ matrix ได้ในขนาด 8x8

โหมดการทำงานของไอซี ซึ่งกำหนดโดยการระบุค่าในตำแหน่ง D8-D11 มีดังต่อไปนี้

REGISTER	ADDRESS					HEX CODE
	D15–D12	D11	D10	D9	D8	
No-Op	X	0	0	0	0	0xX0
Digit 0	X	0	0	0	1	0xX1
Digit 1	X	0	0	1	0	0xX2
Digit 2	X	0	0	1	1	0xX3
Digit 3	X	0	1	0	0	0xX4
Digit 4	X	0	1	0	1	0xX5
Digit 5	X	0	1	1	0	0xX6
Digit 6	X	0	1	1	1	0xX7
Digit 7	X	1	0	0	0	0xX8
Decode Mode	X	1	0	0	1	0xX9
Intensity	X	1	0	1	0	0xXA
Scan Limit	X	1	0	1	1	0xXB
Shutdown	X	1	1	0	0	0xXC
Display Test	X	1	1	1	1	0xXF

โหมดต่างๆ มีรายละเอียดดังนี้ (X หมายถึงเป็นแต่ละบิตเป็นค่าใดก็ได้)

โหมด No-Op (0xXXX0)

โหมด No-op ใช้สำหรับกรณีที่ต่อไอซี MAX7219 หลายๆ ตัวแบบ daisy-chain กัน (นำขาสัญญาณ DOUT ของไอซีตัวแรกต่อเข้ากับ DIN ของไอซีตัวถัดไป ทำเช่นนี้เรื่อยไปตามที่ต้องการ) ตามหลักการพื้นฐานที่เราได้ทำความเข้าใจในเบื้องต้นแล้วว่า กลไกการทำงานภายในของวงจรบัส SPI คือการเลื่อนค่าในเรจิสเตอร์ของอุปกรณ์ตัว master ส่งไปยัง slave และในเวลาเดียวกันนั้น ข้อมูลจาก slave ก็จะถูกเลื่อนบิตกลับมายังให้เรจิสเตอร์ของอุปกรณ์ master ส่วนในกรณีที่เราเชื่อมต่อหลายตัวแบบ daisy-chain นั้น ข้อมูลจาก slave ตัวแรกจะถูกส่งต่อไปยังให้ slave ตัวที่สอง และส่งต่อกันไปเรื่อยๆ จนถึงตัวสุดท้าย

ดังนั้น ในกรณีที่เราต้องการส่งข้อมูลไปยัง slave ตัวที่สอง ในกรณี MAX7219 นี้ เราจะส่งข้อมูลขนาด 16 บิตสองตัวออกไปจาก master โดยข้อมูล 16 บิตตัวแรกคือค่าที่จะส่งไปยัง slave ตัวที่สอง และข้อมูล 16 บิตตัวถัดไปก็คือคำสั่ง No-op นี้ โดยหลักการที่ว่าข้อมูลจะถูก latch ก็ต่อเมื่อขา CS เปลี่ยนลอจิกกลับจาก 0 เป็น 1 เมื่อสิ้นสุดการส่งข้อมูล แม้ว่าไอซีทุกตัวจะได้รับข้อมูลไป แต่ไอซี MAX7219 ตัวใดที่ได้รับ No-Op จะไม่มีการ latch ข้อมูลหรือเปลี่ยนแปลงค่าภายในแต่อย่างได จึงทำให้ข้อมูลถูกส่งไปยังไอซีตัวที่ต้องการเท่านั้น

โหมดแสดงข้อมูล Digit(0-7) (0xX1dd - 0xX8dd)

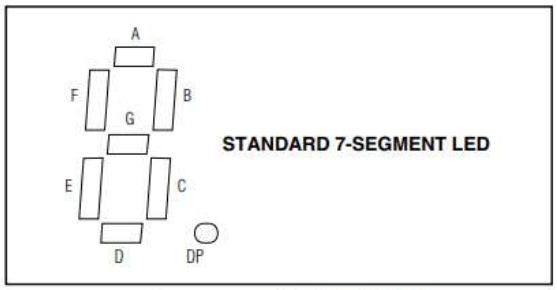
เป็นโหมดคำสั่งที่ใช้ส่งข้อมูลการแสดงให้กับ LED แต่ละหลัก โดยหลักแรกใช้ค่า 1 ไปจนถึงหลักสุดท้ายเป็นค่า 8 และ dd ก็คือข้อมูลแต่ละบิตที่จะใช้แสดงของหลักดังกล่าว

โหมดเลือกประเภทการแสดง Decode Mode (0xX9dd)

เราสามารถใช้ MAX7219 เพื่อขับ LED ได้ทั้งแบบ 7-segment และแบบ 8แถวเรียง ผสมผสานกันไปได้ โดยหากเราต้องการให้ LED หลักใดแสดงผลแบบ 7-segment ก็กำหนดบิตประจำหลักนั้นให้เป็น 1 หากกำหนดเป็น 0 ค่าในแต่ละบิตจะถูกนำไปใช้แสดงโดยไม่มีการถอดรหัสแต่อย่างใด (ซึ่งในกรณี 8x8 matrix ที่นำมาใช้งานนี้จะใช้งานในโหมดไม่ถอดรหัส)

7-SEGMENT CHARACTER	REGISTER DATA						ON SEGMENTS = 1							
	D7*	D6–D4	D3	D2	D1	D0	DP*	A	B	C	D	E	F	G
0		X	0	0	0	0		1	1	1	1	1	1	0
1		X	0	0	0	1		0	1	1	0	0	0	0
2		X	0	0	1	0		1	1	0	1	1	0	1
3		X	0	0	1	1		1	1	1	1	0	0	1
4		X	0	1	0	0		0	1	1	0	0	1	1
5		X	0	1	0	1		1	0	1	1	0	1	1
6		X	0	1	1	0		1	0	1	1	1	1	1
7		X	0	1	1	1		1	1	1	0	0	0	0
8		X	1	0	0	0		1	1	1	1	1	1	1
9		X	1	0	0	1		1	1	1	1	0	1	1
—		X	1	0	1	0		0	0	0	0	0	0	1
E		X	1	0	1	1		1	0	0	1	1	1	1
H		X	1	1	0	0		0	1	1	0	1	1	1
L		X	1	1	0	1		0	0	0	1	1	1	0
P		X	1	1	1	0		1	1	0	0	1	1	1
blank		X	1	1	1	1		0	0	0	0	0	0	0

*The decimal point is set by bit D7 = 1



	REGISTER DATA							
	D7	D6	D5	D4	D3	D2	D1	D0
Corresponding Segment Line	DP	A	B	C	D	E	F	G

โหมดความสว่าง Intensity (0xXAXd)

เราสามารถกำหนดค่าความสว่างของ LED ได้ทั้งหมด 16 ระดับ โดยการกำหนดที่ 4 บิตสุดท้ายของคำสั่ง

โหมดกำหนดจำนวนหลักที่ใช้งาน Scan-Limit (0xXBX0-0xXBX7)

เราสามารถกำหนดจำนวนหลักของ LED ที่จะใช้งานได้ตั้งแต่ 1 ตัว ไปจนถึง 8 ตัวโดยการกำหนดที่ 3 บิตสุดท้ายจากค่า 0 (1 หลัก) ไปจนถึง 7 (8หลัก)

โหมดสั่งเริ่มทำงาน Shutdown (0xXCX0-0xXCX1)

เราสามารถสั่งให้ LED เริ่มนำข้อมูลมาใช้ในการแสดงผล (สั่งเริ่มแสกนสัญญาณเพื่อให้ LED ติด) ได้โดยการสั่ง 0xC01 หรือสั่งให้หยุดการทำงาน (ทำให้หน้าจอดับ) โดยการสั่ง 0xC00

โหมดทดสอบ Display Test (0xXFXX)

ใช้เพื่อทดสอบ LED ว่าทำงานทุกดวงหรือไม่ (LED จะสว่างทุกดวง)

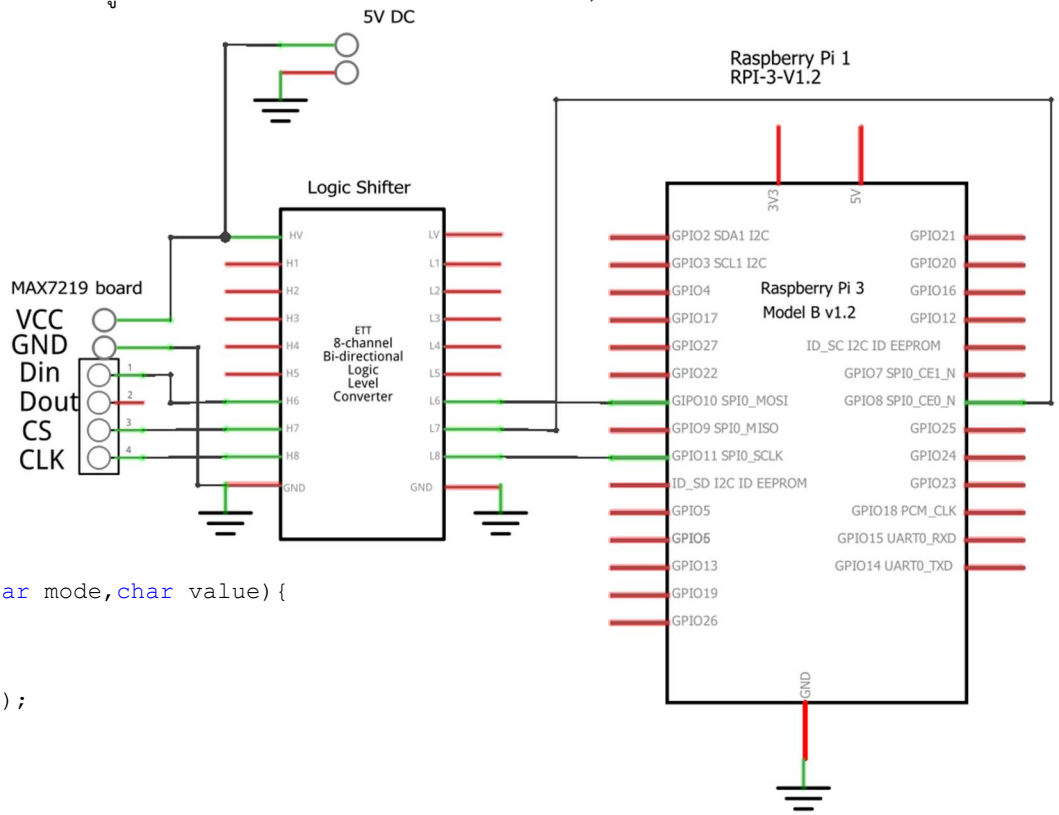
เราสามารถสั่งให้ LED เริ่มนำข้อมูลมาใช้ในการแสดงผล (สั่งเริ่มแสกนสัญญาณเพื่อให้ LED ติด) ได้โดยการสั่ง 0xC01 หรือสั่งให้หยุดการทำงาน (ทำให้หน้าจอดับ) โดยการสั่ง 0xC00

ปฏิบัติการ: การติดต่อกับไอซี MAX7219 ผ่านบัส SPI

อุปกรณ์ที่ต้องการ

- บอร์ด Raspberry Pi พร้อมอแดปเตอร์
- บอร์ด logic shifter 1 บอร์ด
- บอร์ดทดลองไอซี MAX7219 พร้อมกับ 8x8 matrix LED

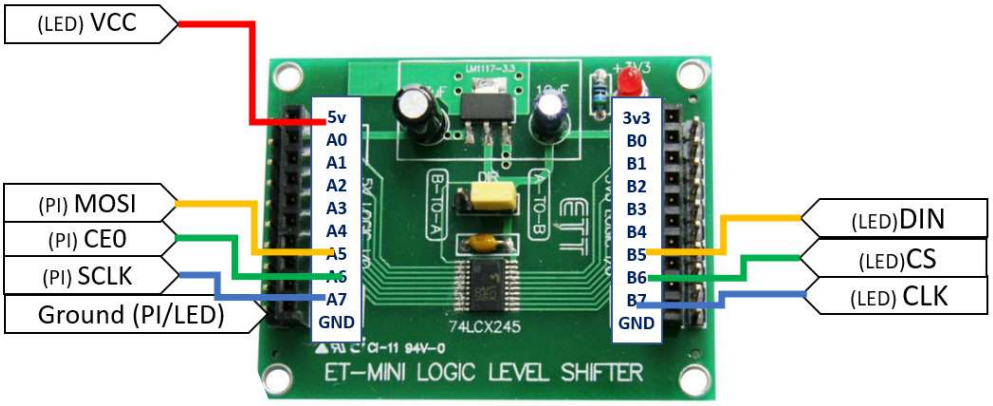
ตัวอย่างโปรแกรมต่อไปนี้ เป็นการนำเอาไอซี MAX7219 ที่นำมาต่อกับ LED แบบ 8x8 matrix มาต่อกับ raspberry pi ผ่านบัส SPI แบบ 3-wire ให้สังเกตการณ์ใช้ฟังก์ชัน spiXfer() ของ pigpio ซึ่งส่งข้อมูลขนาด 16 บิต ในลักษณะของการส่งข้อมูล 8 บิตสองตัวต่อกัน โดยขา CS จะคงสถานะลอจิก 0 ตลอดในช่วงของการส่งข้อมูลทั้งสองไบต์ จึงทำให้เกิดการผสมรวมกันเป็น 16 บิตหนึ่งชุดข้อมูลตามข้อกำหนดของไอซี MAX7219 (ในโปรแกรมตัวอย่าง เราใช้อะเรย์ data มาเพื่อรับข้อมูลกลับด้วย ซึ่งไม่ได้มีการนำมาใช้ต่อแต่ประการใด)



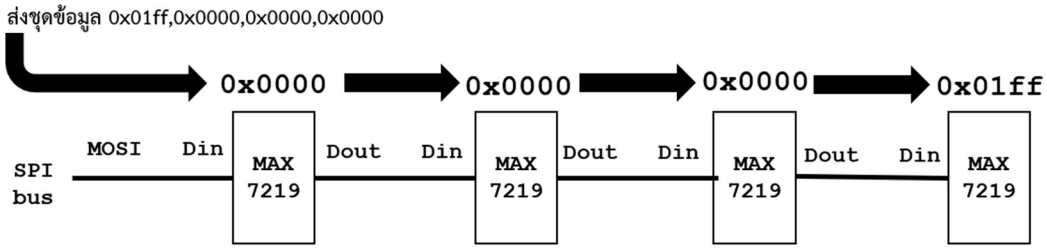

```
#include <pigpio.h>

#include <stdlib.h>
#include <unistd.h>
#include <stdint.h>
#include <stdio.h>

#define DECODE_MODE 0x09
#define INTENSITY   0x0a
#define SCAN_LIMIT  0x0b
#define SHUTDOWN    0x0c
#define DISPLAY_TEST 0x0f

void mySpiWrite(int port,char mode,char value){
   char data[2];
   data[0]=mode;
   data[1]=value;
   spiXfer(port,data,data,2);
}

int main(){
       int i,j;
       int spi;

    if(gpioInitialise()<0) return -1;
    if((spi=spiOpen(0,100000,0))<0) return -1;
    mySpiWrite(spi,INTENSITY,10);
    mySpiWrite(spi,DECODE_MODE,0);
    mySpiWrite(spi,SCAN_LIMIT,7);
    for(j=0;j<64;j++){
        mySpiWrite(spi,SHUTDOWN,0);
        for(i=0;i<8;i++)
            mySpiWrite(spi,i+1,1<<((i+j)%8));
        mySpiWrite(spi,SHUTDOWN,1);
        usleep(100000);
    }
```

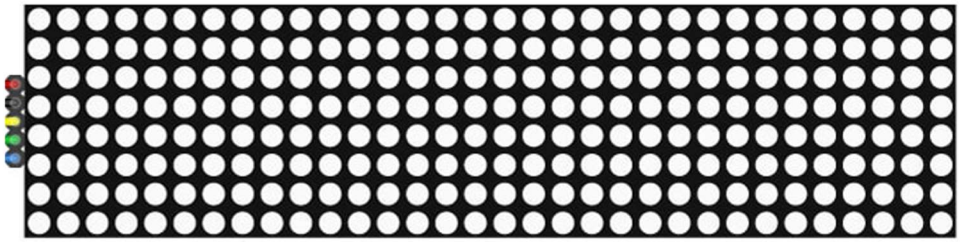
```
    spiClose(spi);
    gpioTerminate();
       return 0;
}
```

การใช้งานบอร์ดรุ่นใหม่ของ ETT ที่ใช้ไอซี 74LCX245 กับ MAX7219/8x8LED matrix



ตัวอย่างต่อไปเราใช้บอร์ด MAX7219 จำนวนสี่ตัวมาต่อ daisy-chain กัน หรือใช้บอร์ดแบบที่ต่อสำเร็จมาแล้ว ทำให้ได้ LED ขนาด 32x8 matrix ให้สังเกตการณ์ใช้คำสั่ง No-Op ต่อท้ายข้อมูลที่จะแสดงผล เพื่อดันให้ข้อมูลส่งไปยังไอซีตัวถัดๆ ไปใน daisy-chain โดยในฟังก์ชัน mySpiWriteX() ที่สร้างขึ้น มีการกำหนดอะเรย์ data ขนาด 8 หน่วย และกำหนดค่าเริ่มต้นให้ทุกสมาชิกเป็นศูนย์ จากนั้นเราใส่ข้อมูลเฉพาะ 2 ไบต์แรก ซึ่งหมายถึงข้อมูลขนาด 16 บิตที่จะส่งให้กับ MAX7219 จากนั้นเราใช้ฟังก์ชัน spiXfer เพื่อส่งข้อมูลขนาด 16 บิตนี้ และอาจตามด้วยข้อมูล 0 ขนาด 16 บิตจำนวน 1 ถึง 3 ตัว (ซึ่งก็คือ No-Op จำนวน 1 ถึง 3 ตัว) เพื่อส่งผลให้ข้อมูลไปถึงไอซีในลำดับที่เราต้องการได้





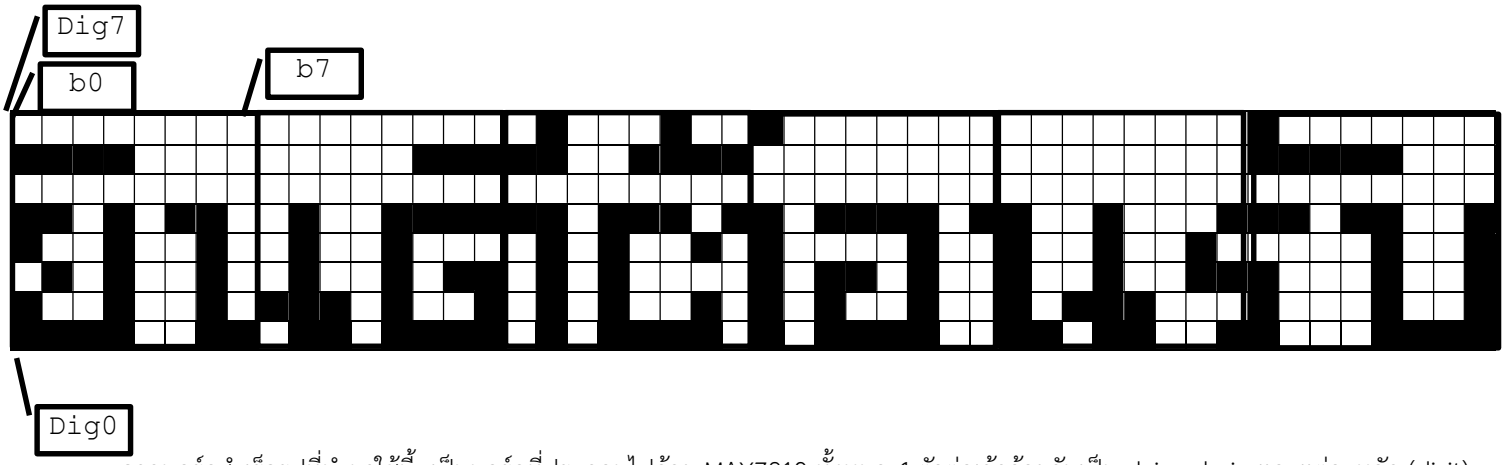
ตัวอย่างเช่น สมมติว่ามีการต่อ MAX7129 ทั้งหมดสี่ตัวแบบ daisy-chain และส่งชุดข้อมูล 16 บิตจำนวนสี่ตัว (เท่ากับ 8 บิต 8 ตัว) ไปยัง SPI ที่เชื่อมต่อกับ MAX7219 ดังกล่าว จะเห็นว่าข้อมูลแต่ละชุด จะเลื่อนผ่านไปยังไอซีแต่ละตัวตามลำดับ ทำให้ไอซีตัวสุดท้ายได้ข้อมูล 16 บิตตัวแรก และตัวรองสุดท้ายได้ข้อมูล 16 บิตตัวถัดมา เป็นเช่นนี้เรื่อยไปจนถึงไอซีตัวแรกที่รับข้อมูล 16 บิตตัวสุดท้ายไว้ เมื่อขาสัญญาณ CS เปลี่ยนสถานะลอจิกจาก 0 เป็น 1 จะทำให้ไอซีแต่ละตัวรับค่าของตนไว้ ในที่นี้ MAX7219 พิจารณาข้อมูล 0xX0XX เป็น No-Op หมายความว่าจะไม่นำข้อมูลที่ได้นี้ไปทำอะไรต่อไป

<u>หมายเหตุ</u> *การส่งข้อมูลแบบชุดที่ประกอบไปด้วยข้อมูลปลายทางและตามด้วนย No-Op นั้น หากส่งข้อมูลในขณะที่โหมด shutdown เป็น 1 จะพบว่า LED บางดวงอาจจะมีการกระพริบเกิดขึ้น ดังนั้นจึงมีการเซ็ตโหมด shutdown เป็น 0 เพื่อปิด LED ทั้งหมดเป็นการชั่วคราวก่อนส่งข้อมูล นักศึกษาอาจทดลองส่งข้อมูลโดยไม่ต้องเซ็ตโหมด shutdown เป็น 0 ก่อนส่งข้อมูล เพื่อดูว่า ระหว่างการที่หน้าจอ LED ปิดทั้งหน้าจอ (จะเกิดการกระพริบทั้งหน้าจอ) กับการที่มี LED บางดวงกระพริบนั้น อย่างใดจะเหมาะสมกว่า*

อีกทางเลือกหนึ่งเพื่อแก้ไขปัญหาดังกล่าว คือการสร้างอะเรย์เพื่อเก็บชุดข้อมูลทั้งหมดไว้ล่วงหน้า จากนั้นส่งข้อมูลไปเพียงชุดเดียวเพื่อส่งไปยังไอซีทุกตัวในคราวเดียว ดูตัวอย่างการใช้งานได้ตามตัวอย่างท้ายสุดที่ปรากฏในบทนี้

```
#include <pigpio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h>
#include <stdio.h>

#define NOOP        0
#define DECODE_MODE 0x09
#define INTENSITY   0x0a
#define SCAN_LIMIT  0x0b
#define SHUTDOWN    0x0c
#define DISPLAY_TEST 0x0f

void mySpiWrite(int port,char mode,char value){
   char data[2];
   data[0]=mode;
   data[1]=value;
   spiXfer(port,data,data,2);
}

void mySpiWriteX(int port,char mode,char value,int location){
   char data[8]={0};
   data[0]=mode;
   data[1]=value;
   spiXfer(port,data,data,2+location*2);
}

int main(){
       int i,j,k;
       int spi;

    if(gpioInitialise()<0) return -1;
    if((spi=spiOpen(0,100000,0))<0) return -1;
    for(k=0;k<4;k++){
        mySpiWriteX(spi,INTENSITY,10,k);
        mySpiWriteX(spi,DECODE_MODE,0,k);
        mySpiWriteX(spi,SCAN_LIMIT,7,k);
    }
    for(j=0;j<64;j++){
        for(k=0;k<4;k++)
            mySpiWriteX(spi,SHUTDOWN,0,k);
        for(i=0;i<8;i++)
            for(k=0;k<4;k++)
                mySpiWriteX(spi,i+1,1<<((i+j)%8),k);
        for(k=0;k<4;k++)
            mySpiWriteX(spi,SHUTDOWN,1,k);
        usleep(100000);
    }
    spiClose(spi);
    gpioTerminate();
       return 0;
}
```

ตัวอย่างสุดท้ายนี้ เป็นการทดลองการแสดงภาพไปยังชุด LED ขนาด 32x8 matrix ในลักษณะเลื่อนภาพจากซ้ายไปขวา โดยข้อมูลภาพที่จะแสดงเป็นดังนี้



จากบอร์ดสำเร็จรูปที่นำมาใช้นี้ เป็นบอร์ดที่ประกอบไปด้วย MAX7219 ทั้งหมด 4 ตัวต่อเข้าด้วยกันเป็น daisy-chain และแต่ละหลัก (digit) ของ LED ที่แสดงวางตัวในแนวนอนจากล่างขึ้นบน และบิตทางซ้ายเป็นบิตต่ำสุด ทำให้การโปรแกรมภาพขึ้นไปแสดงจะต้องมีการกำหนดบิตแต่ละบิตของแต่ละไบต์ข้อมูลที่จะส่งไปยัง digit ต่างๆ ตามที่กำหนดไว้ข้างต้น ส่งผลทำให้สามารถพัฒนาโปรแกรมเพื่อแสดงภาพและเลื่อนจากขวาไปทางซ้ายได้ดังนี้

```
#include <pigpio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h>
#include <stdio.h>
#include <signal.h>

#define NOOP        0
#define DECODE_MODE 0x09
#define INTENSITY   0x0a
#define SCAN_LIMIT  0x0b
#define SHUTDOWN    0x0c
#define DISPLAY_TEST 0x0f
int running = true;

void gpio_stop(int sig);
void mySpiWrite(int port,char mode,char value);
void mySpiWriteX(int port,char mode,char value,int location);
void mySpiWrite4(int port,char mode,char value);
void putImage(int port,char* data,int dlen,int offset);

char data[48]= {0xcf,0x49,0x4a,0x49,0x6b,0x00,0x0f,0x00,
                0xf6,0x97,0xd2,0x12,0xf2,0x00,0xe0,0x00,
                0x7a,0x4a,0x0a,0x4a,0xbb,0x00,0xf3,0x22,
                0x3d,0x25,0x2d,0x21,0xbb,0x00,0x00,0x01,
                0x9b,0x1d,0xc9,0x49,0x89,0x00,0x00,0x00,
                0xf1,0x91,0x91,0x90,0x9b,0x00,0x1f,0x01};
int main(){
       int i,k;
       int spi;

    if(gpioInitialise()<0) return -1;
    if((spi=spiOpen(0,100000,0))<0) return -1;
    signal(SIGINT,gpio_stop);

    mySpiWrite4(spi,INTENSITY,10);
    mySpiWrite4(spi,DECODE_MODE,0);
    mySpiWrite4(spi,SCAN_LIMIT,7);
    mySpiWrite4(spi,SHUTDOWN,0);

    for(k=0;k<8;k++){
        mySpiWriteX(spi,k+1,data[k],0);
```

```
        mySpiWriteX(spi,k+1,data[k+8],1);
        mySpiWriteX(spi,k+1,data[k+16],2);
        mySpiWriteX(spi,k+1,data[k+24],3);
    }
    mySpiWrite4(spi,SHUTDOWN,1);
    sleep(1);

       while(running){
             for(i=-31;i<48;i++){
                 if(!running)break;
            mySpiWrite4(spi,SHUTDOWN,0);

                    putImage(spi,data,48,i);
                     mySpiWrite4(spi,SHUTDOWN,1);
                    usleep(100000);
             }
       }
    spiClose(spi);
    gpioTerminate();
       return 0;
}

void mySpiWrite(int port,char mode,char value){
   char data[2];
   data[0]=mode;
   data[1]=value;
   spiXfer(port,data,data,2);
}

void mySpiWriteX(int port,char mode,char value,int location){
   char data[8]={0};
   data[0]=mode;
   data[1]=value;
   spiXfer(port,data,data,2+location*2);
}

void mySpiWrite4(int port,char mode,char value){
   char data[8];
   data[0]=data[2]=data[4]=data[6]=mode;
   data[1]=data[3]=data[5]=data[7]=value;
   spiXfer(port,data,data,8);
}

void putImage(int port,char* data,int dlen,int offset){
// LED size = 32
    uint16_t value=0;
       int j,k,joffset;

       for(j=0;j<4;j++){
        for(k=0;k<8;k++){
            if((j+(offset/8)<0)||(j+(offset/8)>=(dlen/8)))
                mySpiWriteX(port,k+1,0,j);
            else{
                value=0;
                joffset=offset+(j*8);
                if(joffset%8){ //loading previous dataset
                    if(joffset>=-7)
                       value|=data[(joffset/8)*8+(k%8)];
                    if(((joffset/8+1)<6)&&(joffset>=0))
                        value|=(data[(joffset/8+1)*8+(k%8)]<<8);
                    if(joffset<0)
                    (value <<=(8-joffset)),value>>=8;
                    else
                    value >>=joffset%8;
                    printf("dataS=%.4X \n",value);

                    mySpiWriteX(port,k+1,value,j);
                }else{
                    value=data[(joffset/8)*8+(k%8)];
```

```
                    mySpiWriteX(port,k+1,value,j);
                }
            }
        }
    }
}

void gpio_stop(int sig){
    printf("Exiting..., please wait\n");
    running = false;
}
```

จากโปรแกรมข้างบน สังเกตถึงการส่งข้อมูลการเปิด/ปิดการทำงาน และอื่นๆ ไปยังไอซีทั้งสี่ตัวพร้อมกันภายในชุดข้อมูลชุดเดียวด้วยฟังก์ชัน mySpiWrite4() ซึ่งแตกต่างจากตัวอย่างโปรแกรมก่อนหน้าซึ่งอาศัยการใช้ No-Op ผสมกับข้อมูลที่ต้องการส่งไปยังไอซีปลายทาง (จึงต้องส่งข้อมูลสี่ชุดไปยังไอซีทั้งสี่ตัว)

โปรแกรมตัวอย่างสุดท้ายนี้เป็นโปรแกรมแสดงข้อความบน 32x8 matrix LED โดยนิยามอะเรย์จำนวน 8 หน่วยของข้อมูลขนาด 32 บิต โดยทำหน้าที่แทนพื้นที่แนวนอน 32 บิตของ LED โดยมี 8 หลักแนวตั้ง ส่วนอักขระนั้น เราสร้างไฟล์ฟอนต์ซึ่งภายในมีการนิยามภาพของฟอนต์ทั้ง 256 อักขระตามมาตรฐาน ASCII อักขระแต่ละตัวมีขนาด 5x8 จุด การแสดงข้อความจะใช้การเลื่อนบิตข้อมูลฟอนต์ลงในตำแหน่งที่จะวาง แล้วทำ Bitwise OR ภาพฟอนต์กับพื้นที่อะเรย์ สำหรับไฟล์ฟอนต์นี้มีการดัดแปลงจากไฟล์ฟอนต์ทั่วไป เนื่องจากการวางตำแหน่ง LED บนบอร์ดทดลองนั้นมีลักษณะที่แตกต่างไปจากกลไกการจัดการฟอนต์บนอุปกรณ์แสดงผลทั่วไป

ไฟล์ font5x8.cpp

```
#define font(ch,x) console_font_5x8[(ch<<3)+x]

unsigned char console_font_5x8[] = {
/* code=00 */
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
/* code=01 */
0x00,0x00,0x0E,0x15,0x1F,0x1B,0x0E,0x00,
/* code=02 */
0x00,0x00,0x0E,0x15,0x1F,0x1F,0x0E,0x00,
/* code=03 */
0x00,0x00,0x0A,0x1F,0x1F,0x0E,0x04,0x00,
/* code=04 */
0x00,0x00,0x04,0x0E,0x1F,0x0E,0x04,0x00,
/* code=05 */
0x00,0x00,0x0E,0x15,0x1F,0x04,0x04,0x00,
/* code=06 */
0x00,0x00,0x04,0x0E,0x1F,0x15,0x04,0x00,
/* code=07 */
0x00,0x00,0x00,0x04,0x0E,0x04,0x00,0x00,
/* code=08 */
0x00,0x1F,0x1F,0x1B,0x11,0x1B,0x1F,0x1F,
/* code=09 */
0x00,0x00,0x00,0x04,0x0A,0x04,0x00,0x00,
/* code=10 */
0x00,0x1F,0x1F,0x1B,0x11,0x1B,0x1F,0x1F,
/* code=11 */
0x00,0x00,0x1C,0x18,0x16,0x05,0x02,0x00,
/* code=12 */
0x00,0x00,0x04,0x0A,0x04,0x0E,0x04,0x00,
/* code=13 */
0x00,0x00,0x04,0x0A,0x02,0x03,0x01,0x00,
/* code=14 */
0x00,0x00,0x1C,0x12,0x1A,0x0B,0x01,0x00,
/* code=15 */
0x00,0x00,0x00,0x04,0x0A,0x04,0x00,0x00,
/* code=16 */
0x00,0x00,0x02,0x06,0x0E,0x06,0x02,0x00,
/* code=17 */
0x00,0x00,0x08,0x0C,0x0E,0x0C,0x08,0x00,
/* code=18 */
0x00,0x00,0x04,0x0E,0x04,0x0E,0x04,0x00,
/* code=19 */
0x00,0x00,0x0A,0x0A,0x0A,0x00,0x0A,0x00,
/* code=20 */
0x00,0x00,0x1E,0x0B,0x0B,0x0A,0x0A,0x0A,
/* code=21 */
0x00,0x00,0x18,0x06,0x09,0x12,0x0C,0x03,
/* code=22 */
0x00,0x00,0x00,0x00,0x00,0x1F,0x1F,0x00,
/* code=23 */
0x00,0x00,0x04,0x0E,0x04,0x0E,0x04,0x0E,
/* code=24 */
0x00,0x00,0x04,0x0E,0x04,0x04,0x04,0x00,
/* code=25 */
0x00,0x00,0x04,0x04,0x04,0x0E,0x04,0x00,
/* code=26 */
0x00,0x00,0x00,0x08,0x1F,0x08,0x00,0x00,
/* code=27 */
0x00,0x00,0x00,0x02,0x1F,0x02,0x00,0x00,
/* code=28 */
0x00,0x00,0x00,0x00,0x01,0x1F,0x00,0x00,
/* code=29 */
0x00,0x00,0x00,0x00,0x0A,0x1F,0x0A,0x00,0x00,
/* code=30 */
0x00,0x00,0x00,0x00,0x04,0x0E,0x1F,0x00,
/* code=31 */
0x00,0x00,0x00,0x00,0x1F,0x0E,0x04,0x00,
/* code=32 */
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
/* code=33 */
0x00,0x00,0x04,0x04,0x04,0x00,0x04,0x00,
/* code=34 */
0x00,0x00,0x0A,0x0A,0x00,0x00,0x00,0x00,
/* code=35 */
0x00,0x00,0x0A,0x1F,0x0A,0x1F,0x0A,0x00,
/* code=36 */
0x00,0x00,0x04,0x0C,0x02,0x0C,0x06,0x04,
/* code=37 */
0x00,0x02,0x15,0x0A,0x0C,0x16,0x09,0x00,
/* code=38 */
0x00,0x00,0x0C,0x02,0x16,0x09,0x16,0x00,
/* code=39 */
0x00,0x04,0x04,0x00,0x00,0x00,0x00,0x00,
/* code=40 */
0x00,0x00,0x04,0x02,0x02,0x02,0x04,0x00,
/* code=41 */
```

```
0x00,0x00,0x02,0x04,0x04,0x04,0x02,0x00,
/* code=42 */
0x00,0x00,0x0A,0x04,0x0E,0x04,0x0A,0x00,
/* code=43 */
0x00,0x00,0x00,0x04,0x0E,0x04,0x00,0x00,
/* code=44 */
0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x02,
/* code=45 */
0x00,0x00,0x00,0x0F,0x00,0x00,0x00,0x00,
/* code=46 */
0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x00,
/* code=47 */
0x00,0x00,0x08,0x04,0x04,0x02,0x02,0x00,
/* code=48 */
0x00,0x00,0x06,0x09,0x09,0x09,0x06,0x00,
/* code=49 */
0x00,0x00,0x04,0x06,0x04,0x04,0x04,0x00,
/* code=50 */
0x00,0x00,0x06,0x09,0x04,0x02,0x0F,0x00,
/* code=51 */
0x00,0x00,0x07,0x08,0x06,0x08,0x07,0x00,
/* code=52 */
0x00,0x00,0x08,0x0C,0x0A,0x0F,0x08,0x00,
/* code=53 */
0x00,0x00,0x0F,0x01,0x07,0x08,0x07,0x00,
/* code=54 */
0x00,0x00,0x06,0x01,0x07,0x09,0x06,0x00,
/* code=55 */
0x00,0x00,0x0F,0x08,0x04,0x02,0x02,0x00,
/* code=56 */
0x00,0x00,0x06,0x09,0x06,0x09,0x06,0x00,
/* code=57 */
0x00,0x00,0x06,0x09,0x0E,0x08,0x06,0x00,
/* code=58 */
0x00,0x00,0x00,0x00,0x04,0x00,0x04,0x00,
/* code=59 */
0x00,0x00,0x00,0x00,0x04,0x00,0x04,0x02,
/* code=60 */
0x00,0x00,0x08,0x04,0x02,0x04,0x08,0x00,
/* code=61 */
0x00,0x00,0x00,0x0E,0x00,0x0E,0x00,0x00,
/* code=62 */
0x00,0x00,0x02,0x04,0x08,0x04,0x02,0x00,
/* code=63 */
0x00,0x00,0x06,0x08,0x06,0x00,0x02,0x00,
/* code=64 */
0x00,0x00,0x0E,0x11,0x0D,0x01,0x0E,0x00,
/* code=65 */
0x00,0x00,0x06,0x09,0x0F,0x09,0x09,0x00,
/* code=66 */
0x00,0x00,0x07,0x09,0x07,0x09,0x07,0x00,
/* code=67 */
0x00,0x00,0x0E,0x01,0x01,0x01,0x0E,0x00,
/* code=68 */
0x00,0x00,0x07,0x09,0x09,0x09,0x07,0x00,
/* code=69 */
0x00,0x00,0x0F,0x01,0x07,0x01,0x0F,0x00,
/* code=70 */
0x00,0x00,0x0F,0x01,0x07,0x01,0x01,0x00,
/* code=71 */
0x00,0x00,0x06,0x09,0x01,0x09,0x0E,0x00,
/* code=72 */
0x00,0x00,0x09,0x09,0x0F,0x09,0x09,0x00,
/* code=73 */
0x00,0x00,0x0E,0x04,0x04,0x04,0x0E,0x00,
/* code=74 */
0x00,0x00,0x08,0x08,0x09,0x09,0x06,0x00,
/* code=75 */
0x00,0x00,0x09,0x05,0x03,0x05,0x09,0x00,
/* code=76 */
0x00,0x00,0x01,0x01,0x01,0x01,0x0F,0x00,
/* code=77 */
0x00,0x00,0x09,0x09,0x0F,0x09,0x09,0x00,
/* code=78 */
0x00,0x00,0x09,0x0B,0x0D,0x09,0x09,0x00,
/* code=79 */
0x00,0x00,0x06,0x09,0x09,0x09,0x06,0x00,
/* code=80 */
0x00,0x00,0x07,0x09,0x07,0x01,0x01,0x00,
/* code=81 */
0x00,0x00,0x06,0x09,0x09,0x09,0x06,0x08,
/* code=82 */
0x00,0x00,0x07,0x09,0x07,0x09,0x09,0x00,
/* code=83 */
0x00,0x00,0x0E,0x01,0x06,0x08,0x07,0x00,
/* code=84 */
0x00,0x00,0x1F,0x04,0x04,0x04,0x04,0x00,
/* code=85 */
0x00,0x00,0x09,0x09,0x09,0x09,0x06,0x00,
/* code=86 */
0x00,0x00,0x09,0x09,0x09,0x06,0x06,0x00,
/* code=87 */
0x00,0x00,0x11,0x15,0x15,0x0A,0x0A,0x00,
/* code=88 */
0x00,0x00,0x09,0x09,0x06,0x0A,0x09,0x00,
/* code=89 */
0x00,0x00,0x0A,0x0A,0x0A,0x04,0x04,0x00,
/* code=90 */
0x00,0x00,0x0F,0x04,0x02,0x01,0x0F,0x00,
/* code=91 */
0x00,0x00,0x06,0x02,0x02,0x02,0x06,0x00,
/* code=92 */
0x00,0x00,0x02,0x02,0x04,0x04,0x08,0x00,
/* code=93 */
0x00,0x00,0x06,0x04,0x04,0x04,0x06,0x00,
/* code=94 */
0x00,0x00,0x04,0x0A,0x00,0x00,0x00,0x00,
/* code=95 */
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x1F,
/* code=96 */
0x00,0x00,0x02,0x04,0x00,0x00,0x00,0x00,
/* code=97 */
0x00,0x00,0x00,0x06,0x08,0x0E,0x0A,0x00,
/* code=98 */
0x00,0x01,0x01,0x07,0x09,0x09,0x07,0x00,
/* code=99 */
0x00,0x00,0x00,0x0C,0x02,0x02,0x0C,0x00,
/* code=100 */
0x00,0x00,0x08,0x0E,0x09,0x09,0x0E,0x00,
/* code=101 */
0x00,0x00,0x00,0x06,0x0F,0x01,0x0E,0x00,
/* code=102 */
0x00,0x00,0x0C,0x02,0x07,0x02,0x02,0x00,
/* code=103 */
0x00,0x00,0x00,0x0E,0x09,0x0E,0x08,0x06,
/* code=104 */
0x00,0x00,0x01,0x07,0x09,0x09,0x09,0x00,
/* code=105 */
0x00,0x04,0x00,0x06,0x04,0x04,0x0E,0x00,
/* code=106 */
0x00,0x08,0x00,0x08,0x08,0x08,0x08,0x06,
/* code=107 */
0x00,0x00,0x01,0x05,0x03,0x05,0x09,0x00,
/* code=108 */
0x00,0x06,0x04,0x04,0x04,0x04,0x0E,0x00,
/* code=109 */
0x00,0x00,0x00,0x09,0x0F,0x09,0x09,0x00,
/* code=110 */
0x00,0x00,0x00,0x07,0x09,0x09,0x09,0x00,
/* code=111 */
0x00,0x00,0x00,0x06,0x09,0x09,0x06,0x00,
/* code=112 */
0x00,0x00,0x00,0x07,0x09,0x09,0x07,0x01,
/* code=113 */
0x00,0x00,0x00,0x0E,0x09,0x09,0x0E,0x08,
/* code=114 */
0x00,0x00,0x00,0x0A,0x06,0x02,0x02,0x00,
/* code=115 */
0x00,0x00,0x00,0x0E,0x03,0x0C,0x07,0x00,
/* code=116 */
0x00,0x00,0x02,0x0F,0x02,0x02,0x0C,0x00,
/* code=117 */
0x00,0x00,0x00,0x09,0x09,0x09,0x0E,0x00,
/* code=118 */
0x00,0x00,0x00,0x09,0x09,0x06,0x06,0x00,
/* code=119 */
```

```
0x00,0x00,0x00,0x09,0x09,0x0F,0x09,0x00,
/* code=120 */
0x00,0x00,0x00,0x09,0x06,0x06,0x09,0x00,
/* code=121 */
0x00,0x00,0x00,0x09,0x09,0x0E,0x08,0x06,
/* code=122 */
0x00,0x00,0x00,0x0F,0x04,0x02,0x0F,0x00,
/* code=123 */
0x00,0x08,0x04,0x02,0x02,0x04,0x08,0x00,
/* code=124 */
0x00,0x04,0x04,0x04,0x04,0x04,0x04,0x00,
/* code=125 */
0x00,0x02,0x04,0x08,0x08,0x04,0x02,0x00,
/* code=126 */
0x00,0x00,0x00,0x0A,0x05,0x00,0x00,0x00,
/* code=127 */
0x00,0x00,0x00,0x04,0x0A,0x11,0x1F,0x00,
/* code=128 */
0x00,0x00,0x0E,0x01,0x01,0x01,0x0E,0x04,
/* code=129 */
0x00,0x0A,0x00,0x09,0x09,0x09,0x0E,0x00,
/* code=130 */
0x08,0x04,0x00,0x06,0x0F,0x01,0x0E,0x00,
/* code=131 */
0x04,0x0A,0x00,0x03,0x04,0x05,0x0A,0x00,
/* code=132 */
0x00,0x0A,0x00,0x03,0x04,0x06,0x0D,0x00,
/* code=133 */
0x02,0x04,0x00,0x03,0x04,0x06,0x0D,0x00,
/* code=134 */
0x00,0x04,0x00,0x03,0x04,0x06,0x0D,0x00,
/* code=135 */
0x00,0x00,0x00,0x0C,0x02,0x02,0x0C,0x04,
/* code=136 */
0x04,0x0A,0x00,0x06,0x0F,0x01,0x0E,0x00,
/* code=137 */
0x00,0x0A,0x00,0x06,0x0F,0x01,0x0E,0x00,
/* code=138 */
0x02,0x04,0x00,0x06,0x0F,0x01,0x0E,0x00,
/* code=139 */
0x00,0x0A,0x00,0x06,0x04,0x04,0x0E,0x00,
/* code=140 */
0x04,0x0A,0x00,0x06,0x04,0x04,0x0E,0x00,
/* code=141 */
0x02,0x04,0x00,0x06,0x04,0x04,0x0E,0x00,
/* code=142 */
0x05,0x00,0x06,0x09,0x0F,0x09,0x09,0x00,
/* code=143 */
0x04,0x00,0x06,0x09,0x0F,0x09,0x09,0x00,
/* code=144 */
0x08,0x04,0x0F,0x01,0x07,0x01,0x0F,0x00,
/* code=145 */
0x00,0x00,0x00,0x1B,0x1E,0x07,0x1D,0x00,
/* code=146 */
0x00,0x00,0x0E,0x05,0x0F,0x05,0x0D,0x00,
/* code=147 */
0x04,0x0A,0x00,0x06,0x09,0x09,0x06,0x00,
/* code=148 */
0x00,0x0A,0x00,0x06,0x09,0x09,0x06,0x00,
/* code=149 */
0x02,0x04,0x00,0x06,0x09,0x09,0x06,0x00,
/* code=150 */
0x04,0x0A,0x00,0x09,0x09,0x09,0x0E,0x00,
/* code=151 */
0x02,0x04,0x00,0x09,0x09,0x09,0x0E,0x00,
/* code=152 */
0x00,0x0A,0x00,0x09,0x09,0x0E,0x08,0x06,
/* code=153 */
0x00,0x0A,0x00,0x06,0x09,0x09,0x06,0x00,
/* code=154 */
0x0A,0x00,0x09,0x09,0x09,0x09,0x06,0x00,
/* code=155 */
0x00,0x00,0x04,0x0E,0x01,0x01,0x0E,0x04,
/* code=156 */
0x00,0x0C,0x0A,0x02,0x07,0x02,0x0F,0x00,
/* code=157 */
0x00,0x1B,0x0A,0x0A,0x04,0x0E,0x04,0x00,
/* code=158 */
0x00,0x03,0x05,0x0D,0x1F,0x09,0x11,0x00,
/* code=159 */
0x00,0x0C,0x02,0x02,0x0F,0x02,0x02,0x01,
/* code=160 */
0x04,0x02,0x00,0x03,0x04,0x06,0x0D,0x00,
/* code=161 */
0x08,0x04,0x00,0x06,0x04,0x04,0x0E,0x00,
/* code=162 */
0x08,0x04,0x00,0x06,0x09,0x09,0x06,0x00,
/* code=163 */
0x08,0x04,0x00,0x09,0x09,0x09,0x0E,0x00,
/* code=164 */
0x0A,0x05,0x00,0x07,0x09,0x09,0x09,0x00,
/* code=165 */
0x0A,0x05,0x09,0x0B,0x0B,0x0D,0x09,0x00,
/* code=166 */
0x00,0x04,0x0A,0x0C,0x00,0x0E,0x00,0x00,
/* code=167 */
0x00,0x04,0x0A,0x04,0x00,0x0E,0x00,0x00,
/* code=168 */
0x00,0x04,0x00,0x04,0x02,0x09,0x06,0x00,
/* code=169 */
0x00,0x00,0x00,0x00,0x00,0x1F,0x01,0x00,
/* code=170 */
0x00,0x00,0x00,0x00,0x00,0x1F,0x10,0x00,
/* code=171 */
0x00,0x01,0x09,0x05,0x1A,0x11,0x1C,0x00,
/* code=172 */
0x00,0x11,0x09,0x05,0x12,0x19,0x1C,0x10,
/* code=173 */
0x00,0x04,0x00,0x04,0x04,0x0E,0x04,0x00,
/* code=174 */
0x00,0x00,0x00,0x00,0x00,0x0A,0x05,0x0A,0x00,
/* code=175 */
0x00,0x00,0x00,0x00,0x05,0x0A,0x05,0x00,
/* code=176 */
0x15,0x0A,0x15,0x0A,0x15,0x0A,0x15,0x0A,
/* code=177 */
0x17,0x0A,0x1D,0x0A,0x17,0x0A,0x1D,0x0A,
/* code=178 */
0x1B,0x0E,0x1B,0x0E,0x1B,0x0E,0x1B,0x0E,
/* code=179 */
0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,
/* code=180 */
0x04,0x04,0x04,0x07,0x04,0x04,0x04,0x04,
/* code=181 */
0x04,0x04,0x07,0x04,0x07,0x04,0x04,0x04,
/* code=182 */
0x0A,0x0A,0x0A,0x0B,0x0A,0x0A,0x0A,0x0A,
/* code=183 */
0x00,0x00,0x00,0x0F,0x0A,0x0A,0x0A,0x0A,
/* code=184 */
0x00,0x00,0x07,0x04,0x07,0x04,0x04,0x04,
/* code=185 */
0x0A,0x0A,0x0B,0x08,0x0B,0x0A,0x0A,0x0A,
/* code=186 */
0x0A,0x0A,0x0A,0x0A,0x0A,0x0A,0x0A,0x0A,
/* code=187 */
0x00,0x00,0x0F,0x08,0x0B,0x0A,0x0A,0x0A,
/* code=188 */
0x0A,0x0A,0x0B,0x08,0x0F,0x00,0x00,0x00,
/* code=189 */
0x0A,0x0A,0x0A,0x0F,0x00,0x00,0x00,0x00,
/* code=190 */
0x04,0x04,0x07,0x04,0x07,0x00,0x00,0x00,
/* code=191 */
0x00,0x00,0x00,0x07,0x04,0x04,0x04,0x04,
/* code=192 */
0x04,0x04,0x04,0x1C,0x00,0x00,0x00,0x00,
/* code=193 */
0x04,0x04,0x04,0x1F,0x00,0x00,0x00,0x00,
/* code=194 */
0x00,0x00,0x00,0x1F,0x04,0x04,0x04,0x04,
/* code=195 */
0x04,0x04,0x04,0x1C,0x04,0x04,0x04,0x04,
/* code=196 */
0x00,0x00,0x00,0x1F,0x00,0x00,0x00,0x00,
/* code=197 */
```

```
0x04,0x04,0x04,0x1F,0x04,0x04,0x04,0x04,
/* code=198 */
0x04,0x04,0x1C,0x04,0x1C,0x04,0x04,0x04,
/* code=199 */
0x0A,0x0A,0x0A,0x1A,0x0A,0x0A,0x0A,0x0A,
/* code=200 */
0x0A,0x0A,0x1A,0x02,0x1E,0x00,0x00,0x00,
/* code=201 */
0x00,0x00,0x1E,0x02,0x1A,0x0A,0x0A,0x0A,
/* code=202 */
0x0A,0x0A,0x1B,0x00,0x1F,0x00,0x00,0x00,
/* code=203 */
0x00,0x00,0x1F,0x00,0x1B,0x0A,0x0A,0x0A,
/* code=204 */
0x0A,0x0A,0x1A,0x02,0x1A,0x0A,0x0A,0x0A,
/* code=205 */
0x00,0x00,0x1F,0x00,0x1F,0x00,0x00,0x00,
/* code=206 */
0x0A,0x0A,0x1B,0x00,0x1B,0x0A,0x0A,0x0A,
/* code=207 */
0x04,0x04,0x1F,0x00,0x1F,0x00,0x00,0x00,
/* code=208 */
0x0A,0x0A,0x0A,0x1F,0x00,0x00,0x00,0x00,
/* code=209 */
0x00,0x00,0x1F,0x00,0x1F,0x04,0x04,0x04,
/* code=210 */
0x00,0x00,0x00,0x1F,0x0A,0x0A,0x0A,0x0A,
/* code=211 */
0x0A,0x0A,0x0A,0x1E,0x00,0x00,0x00,0x00,
/* code=212 */
0x04,0x04,0x1C,0x04,0x1C,0x00,0x00,0x00,
/* code=213 */
0x00,0x00,0x1C,0x04,0x1C,0x04,0x04,0x04,
/* code=214 */
0x00,0x00,0x00,0x1E,0x0A,0x0A,0x0A,0x0A,
/* code=215 */
0x0A,0x0A,0x0A,0x1F,0x0A,0x0A,0x0A,0x0A,
/* code=216 */
0x04,0x04,0x1F,0x04,0x1F,0x04,0x04,0x04,
/* code=217 */
0x04,0x04,0x04,0x07,0x00,0x00,0x00,0x00,
/* code=218 */
0x00,0x00,0x00,0x1C,0x04,0x04,0x04,0x04,
/* code=219 */
0x1F,0x1F,0x1F,0x1F,0x1F,0x1F,0x1F,0x1F,
/* code=220 */
0x00,0x00,0x00,0x00,0x1F,0x1F,0x1F,0x1F,
/* code=221 */
0x07,0x07,0x07,0x07,0x07,0x07,0x07,0x07,
/* code=222 */
0x18,0x18,0x18,0x18,0x18,0x18,0x18,0x18,
/* code=223 */
0x1F,0x1F,0x1F,0x1F,0x00,0x00,0x00,0x00,
/* code=224 */
0x00,0x00,0x00,0x16,0x09,0x09,0x16,0x00,
/* code=225 */
0x00,0x06,0x09,0x0F,0x09,0x09,0x07,0x01,
/* code=226 */
0x00,0x0E,0x02,0x02,0x02,0x02,0x02,0x00,
/* code=227 */
0x00,0x00,0x0E,0x0A,0x0A,0x0A,0x0A,0x00,
/* code=228 */
0x00,0x1F,0x12,0x04,0x02,0x11,0x1F,0x00,
/* code=229 */
0x00,0x00,0x1E,0x09,0x09,0x09,0x06,0x00,
/* code=230 */
0x00,0x00,0x00,0x09,0x09,0x09,0x17,0x01,
/* code=231 */
0x00,0x19,0x0A,0x04,0x04,0x04,0x04,0x00,
/* code=232 */
0x00,0x04,0x04,0x0E,0x11,0x0E,0x04,0x04,
/* code=233 */
0x00,0x00,0x0E,0x11,0x1F,0x11,0x0E,0x00,
/* code=234 */
0x00,0x00,0x0E,0x11,0x11,0x0A,0x1B,0x00,
/* code=235 */
0x06,0x01,0x02,0x06,0x09,0x09,0x06,0x00,
/* code=236 */
0x00,0x00,0x00,0x0E,0x15,0x15,0x0E,0x00,
/* code=237 */
0x00,0x00,0x10,0x0E,0x15,0x12,0x0D,0x00,
/* code=238 */
0x00,0x0C,0x02,0x0E,0x02,0x02,0x0C,0x00,
/* code=239 */
0x00,0x06,0x09,0x09,0x09,0x09,0x09,0x00,
/* code=240 */
0x00,0x00,0x0F,0x00,0x0F,0x00,0x0F,0x00,
/* code=241 */
0x00,0x00,0x04,0x1F,0x04,0x00,0x1F,0x00,
/* code=242 */
0x00,0x02,0x04,0x08,0x04,0x02,0x0F,0x00,
/* code=243 */
0x00,0x08,0x04,0x02,0x04,0x08,0x0E,0x00,
/* code=244 */
0x00,0x00,0x18,0x14,0x04,0x04,0x04,0x04,
/* code=245 */
0x04,0x04,0x04,0x04,0x05,0x03,0x00,0x00,
/* code=246 */
0x00,0x00,0x06,0x00,0x0F,0x00,0x06,0x00,
/* code=247 */
0x00,0x00,0x0A,0x05,0x00,0x0A,0x05,0x00,
/* code=248 */
0x00,0x00,0x04,0x0A,0x04,0x00,0x00,0x00,
/* code=249 */
0x00,0x00,0x00,0x06,0x06,0x00,0x00,0x00,
/* code=250 */
0x00,0x00,0x00,0x04,0x00,0x00,0x00,0x00,
/* code=251 */
0x00,0x00,0x18,0x08,0x04,0x05,0x02,0x00,
/* code=252 */
0x00,0x06,0x0A,0x0A,0x00,0x00,0x00,0x00,
/* code=253 */
0x00,0x06,0x08,0x04,0x0E,0x00,0x00,0x00,
/* code=254 */
0x00,0x00,0x00,0x0E,0x0E,0x0E,0x00,0x00,
/* code=255 */
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
};
```

ไฟล์ fontimage32x8.cpp

```
#include "font5x8.cpp"
#include <stdint.h>

uint32_t fimage[8];
int f_loc;

void f_clear(){
	int i;

	for(i=0;i<8;i++)
		fimage[i]=0;
	f_loc=0;
}
```

```
void f_char(uint8_t ch){
       int i,j;
       if(f_loc>5) f_loc=0;

       for(i=0;i<8;i++){
              if(f_loc)
                  fimage[i] |= font(ch,i)<<(f_loc*5-2);
              else
                     fimage[i] != font(ch,i)>>3;
       }
       f_loc++;
}
```

ไฟล์หลักโปรแกรมตัวอย่าง

```
#include <pigpio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h>
#include <stdio.h>
#include <signal.h>
#include "fontimage32x8.cpp"
#include <time.h>

#define NOOP        0
#define DECODE_MODE 0x09
#define INTENSITY   0x0a
#define SCAN_LIMIT  0x0b
#define SHUTDOWN    0x0c
#define DISPLAY_TEST 0x0f
int running = true;

void gpio_stop(int sig);
void mySpiWrite(int port,char mode,char value);
void mySpiWriteX(int port,char mode,char value,int location);
void mySpiWrite4(int port,char mode,char value);

int main(){
       int i;
       int spi;
       time_t rawtime;
       struct tm *timeinfo;
       char text[12];
       char data[8];

    if(gpioInitialise()<0) return -1;
    if((spi=spiOpen(0,100000,0))<0) return -1;
    signal(SIGINT,gpio_stop);

    mySpiWrite4(spi,INTENSITY,10);
    mySpiWrite4(spi,DECODE_MODE,0);
    mySpiWrite4(spi,SCAN_LIMIT,7);
    mySpiWrite4(spi,SHUTDOWN,0);

       while(running){
         time (&rawtime);
         timeinfo = localtime(&rawtime);
         sprintf(text,"%.2d%.2d%.2d",timeinfo->tm_hour,timeinfo->tm_min,timeinfo->tm_sec);

              f_clear();
              for(i=0;i<6;i++){
             f_char(text[i]);
             if(i==1||i==3){
                f_loc-=2;
                f_char('\'');
                f_loc-=2;
             }
         }
              for(i=0;i<8;i++){
                  data[0]=data[2]=data[4]=data[6]=i+1;
```

```
                    data[7]=fimage[i]&0xff;
                    data[5]=(fimage[i]>>8)&0xff;
                    data[3]=(fimage[i]>>16)&0xff;
                    data[1]=(fimage[i]>>24)&0xff;
                    spiXfer(spi,data,data,8);
                }
                mySpiWrite4(spi,SHUTDOWN,1);
                usleep(250000);

        }
     spiClose(spi);
     gpioTerminate();
        return 0;
}

void mySpiWrite(int port,char mode,char value){
   char data[2];
   data[0]=mode;
   data[1]=value;
   spiXfer(port,data,data,2);
}

void mySpiWriteX(int port,char mode,char value,int location){
   char data[8]={0};
   data[0]=mode;
   data[1]=value;
   spiXfer(port,data,data,2+location*2);
}

void mySpiWrite4(int port,char mode,char value){
   char data[8];
   data[0]=data[2]=data[4]=data[6]=mode;
   data[1]=data[3]=data[5]=data[7]=value;
   spiXfer(port,data,data,8);
}

void gpio_stop(int sig){
     printf("Exiting..., please wait\n");
     running = false;
}
```

ปฏิบัติการ: การแสดงข้อมูลบน 8x8 matrix LED

ให้นักศึกษาปรับปรุงตัวอย่างโปรแกรมข้างบน เพื่อแสดงข้อมูลตามที่นักศึกษาต้องการบนบอร์ดทดลอง MAX7219 / 8x8 matrix LED