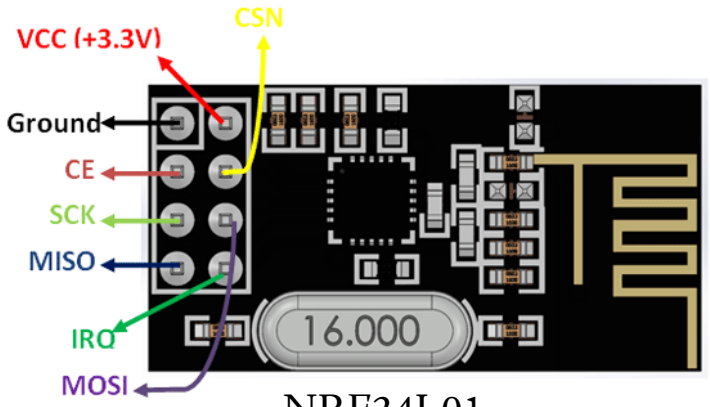
Serial Peripheral Interface (SPI)

Embedded Systems and Communication Protocols for IoT (252461)

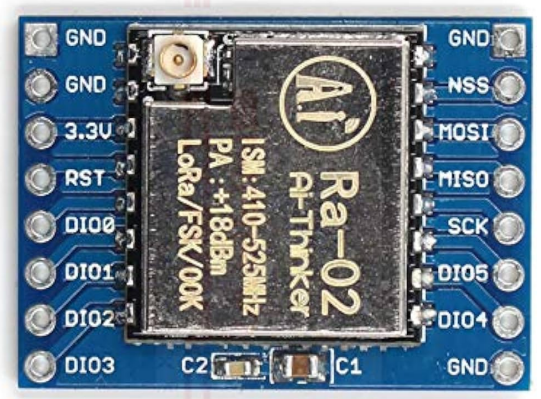
Dr. Witsarut Achariyaviriya
Department of Electrical Engineering, Faculty of Engineering
Chiangmai University

Introduction to SPI

- SPI is one of the most **widely used interfaces** in electronics, facilitating communication between microcontrollers and various peripheral ICs.
- SPI is a **synchronous**, **full duplex** master-slave interface.
- SPI communication involves **four wires**, providing a structured and efficient means of data exchange.
  - SCK - Serial Clock
  - MOSI - Master Out Slave In
  - MISO - Master In Slave Out
  - SS - Select Slave



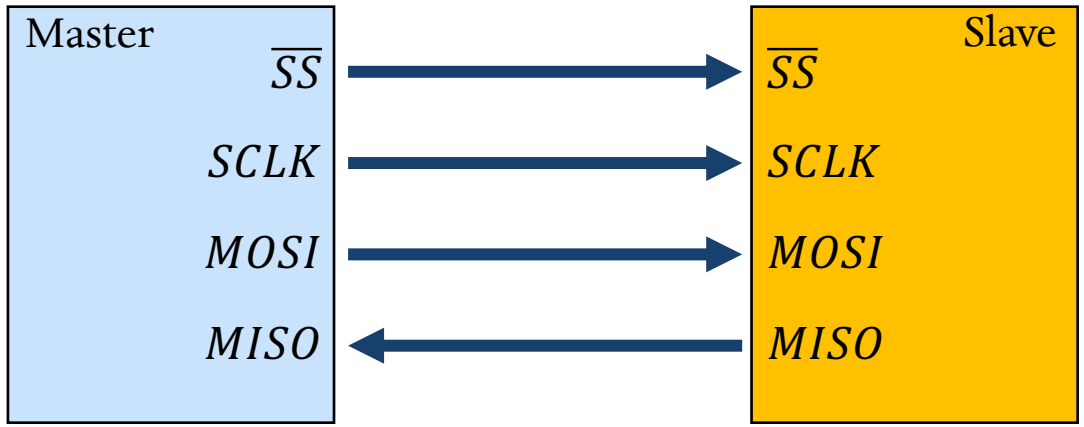
NRF24L01



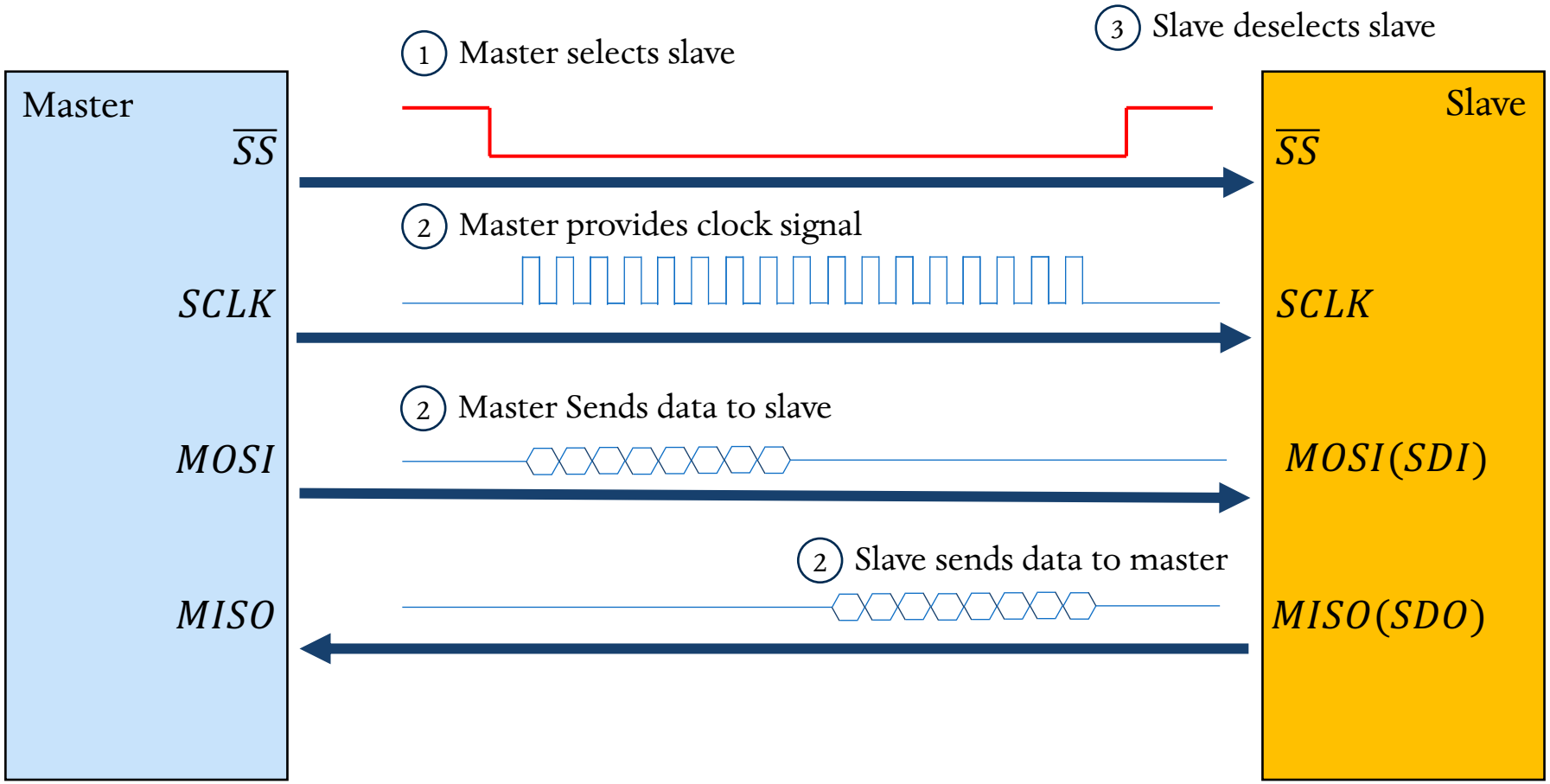
LoRa

Introduction to SPI

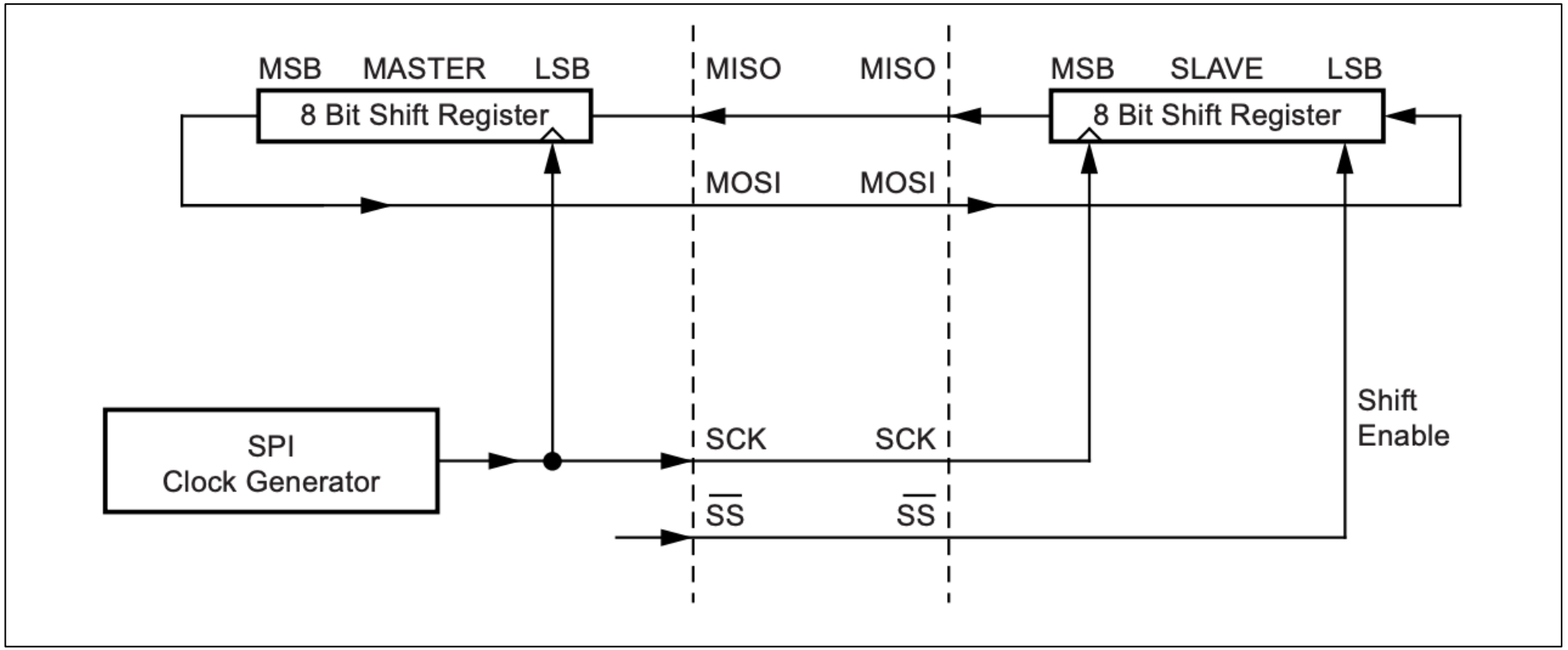
- SPI is one of the most **widely used interfaces** in electronics, facilitating communication between microcontrollers and various peripheral ICs.
- SPI is a **synchronous**, **full duplex** master-slave interface.
- SPI communication involves **four wires**, providing a structured and efficient means of data exchange.
  - SCK - Serial Clock
  - MOSI - Master Out Slave In
  - MISO - Master In Slave Out
  - SS - Select Slave



Basic of SPI Protocol

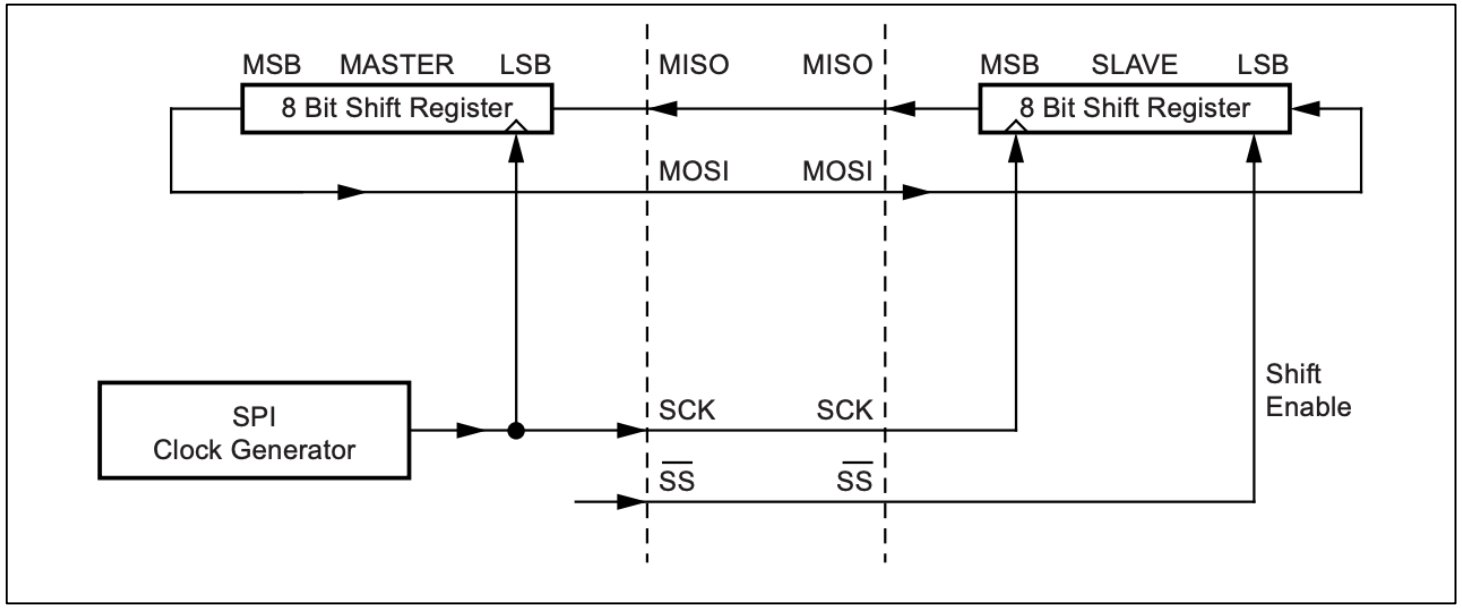


SPI Master-slave Interconnection



MOSI - Master Out Slave In

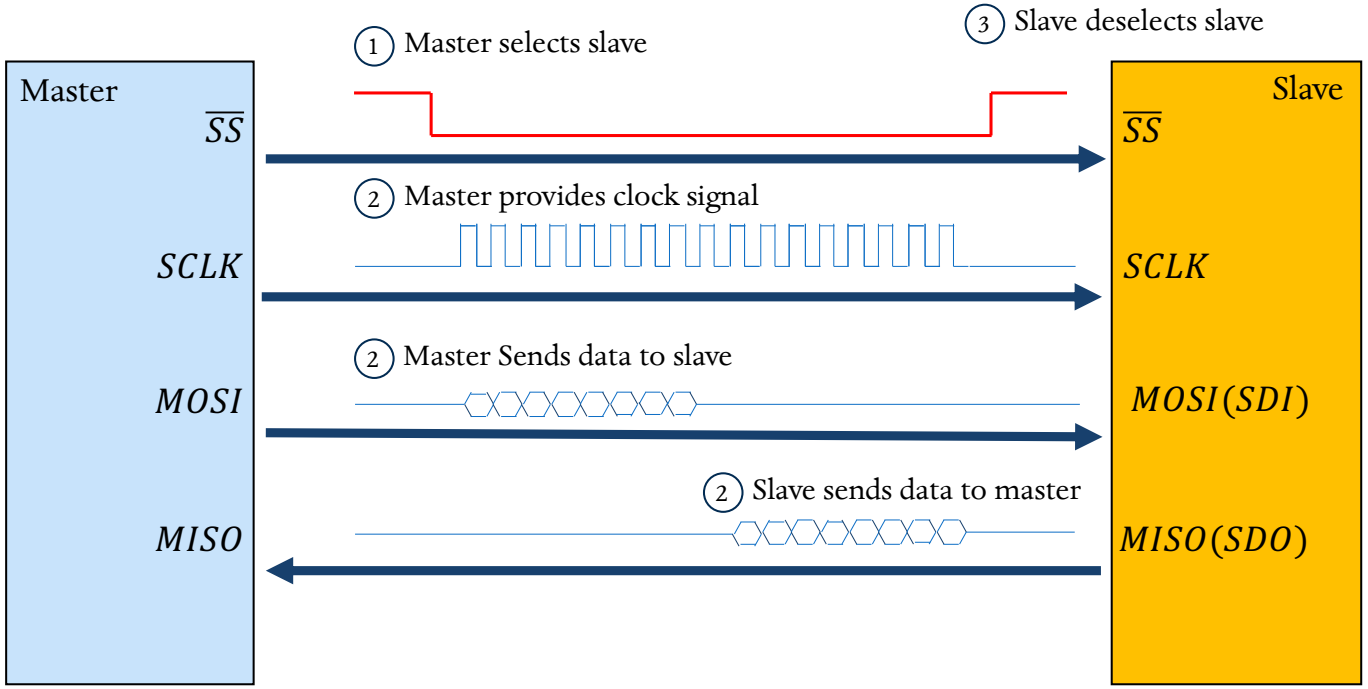
- Used to send data from master to slave(s)
  - Data is usually sent as bytes.
  - Need to cofigurate wether LSB or MSB transmit first (**DORD**: Data Order in case of ATmega328P).
- Number of bytes depends on implementation
  - Multiple bytes can be sent sequentially
  - Slave Select ($\overline{SS}$) may be held low between bytes



https://www.youtube.com/watch?v=0nVNwozXsIc

MISO - Master In Slave Out

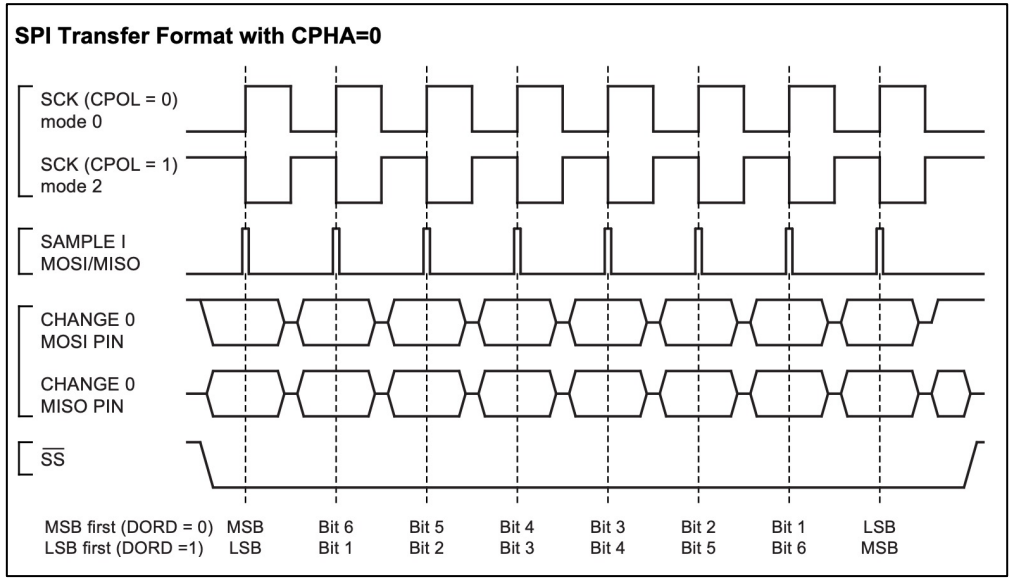
- Used to send data from slave(s) to master
- Not all SPI implementations use MISO
  - Some devices only receive data from master
- MISO sent as a response to data on MOSI
- Master must know how many clock cycles to generate so slave can send all of its data

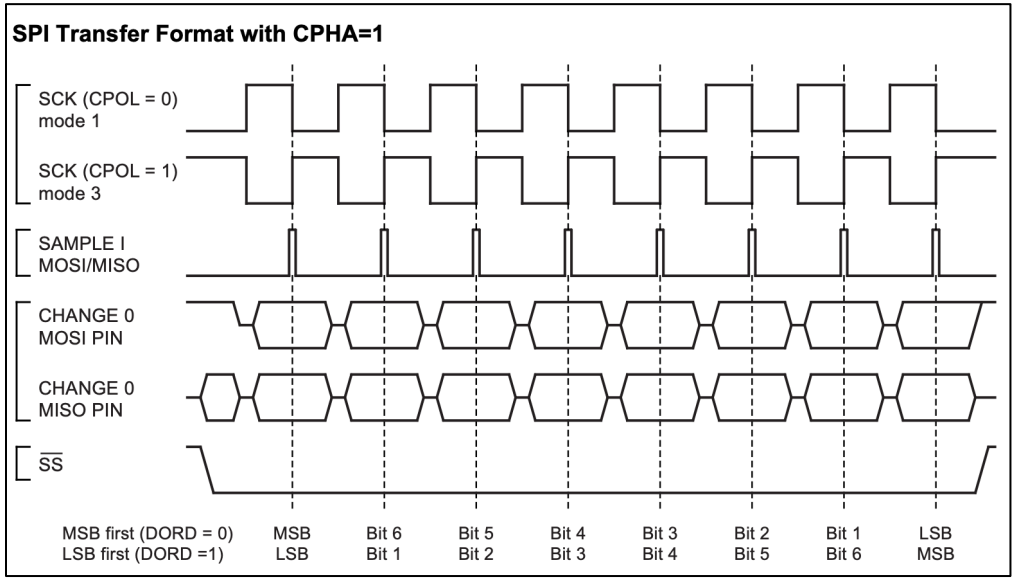


https://www.youtube.com/watch?v=0nVNwozXsIc

Clock Polarity (CPOL) and Clock Phase (CPHA)

SPI Mode	Conditions	Leading Edge	Trailing eDge
0	CPOL=0, CPHA=0	Sample (rising)	Setup (falling)
1	CPOL=0, CPHA=1	Setup (rising)	Sample (falling)
2	CPOL=1, CPHA=0	Sample (falling)	Setup (rising)
3	CPOL=1, CPHA=1	Setup (falling)	Sample (rising)

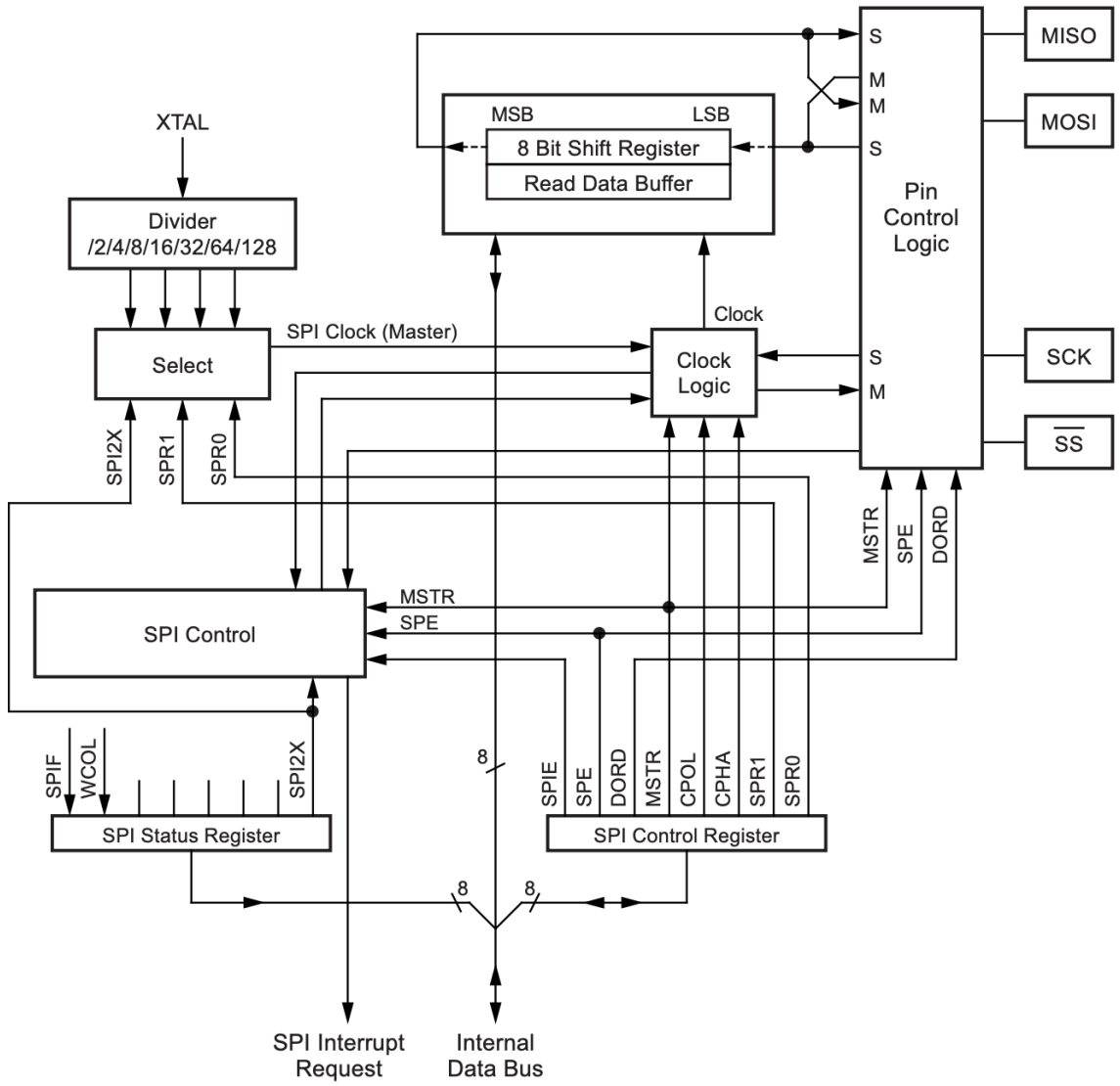




SPI Peripheral Block Diagram (ATmega328P)

Related Register For SPI Configueration

- **SPCR** - SPI Control Register
  - This register controls the configuration and operation of the SPI module.
- **SPSR** - SPI Status Register
  - This register contains flags indicating the status of the SPI module.
- **SPDR** - SPI Data Register
  - This register is used for data transfer between the register file and the SPI shift register.
  - Writing to this register initiates data transmission.
  - Reading from this register retrieves the most recently received data.



SPCR – SPI Control Register

Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

SPI Control Register (SPCR)

SPCR - SPI Control Register: This register controls the configuration and operation of the SPI module.

- Bit 7 (SPIE): SPI Interrupt Enable
- Bit 6 (SPE): SPI Enable (1=enable)
- Bit 5 (DORD): Data Order (MSB/LSB first)
- Bit 4 (MSTR): Master/Slave Select
- Bit 3 (CPOL): Clock Polarity (0 = leading edge Rising)
- Bit 2 (CPHA): Clock Phase (0 = leading edge sample)
- Bit 1 (SPR1): SPI Clock Rate Select 1
- Bit 0 (SPR0): SPI Clock Rate Select 0

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

SPSR – SPI Status Register

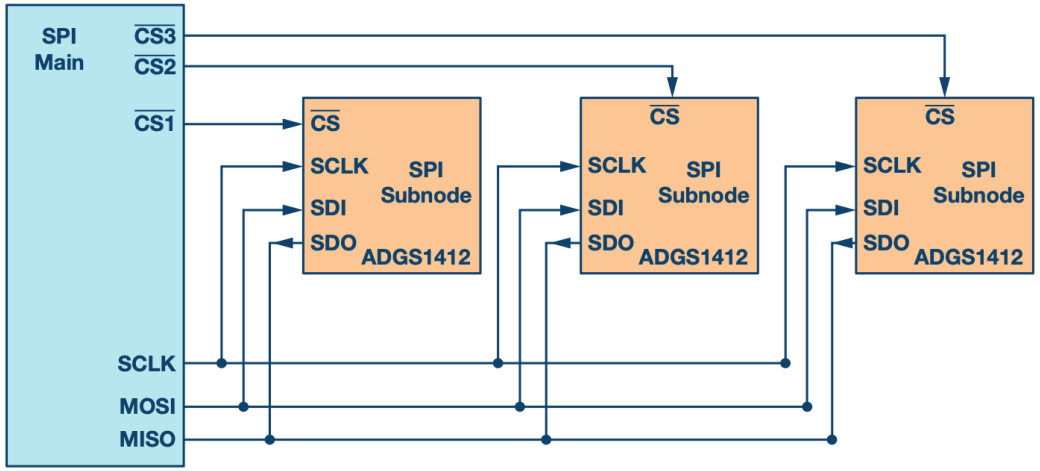
Bit	7	6	5	4	3	2	1	0	
0x2D (0x4D)	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

SPI Status Register (SPSR)

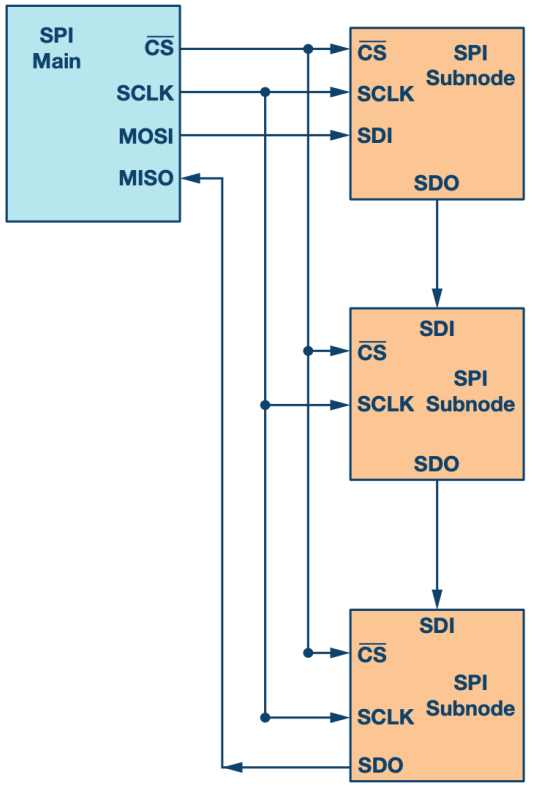
SPSR - SPI Status Register: This register contains flags indicating the status of the SPI module.

- Bit 7 (SPIF): SPI Interrupt Flag.
  - When a serial transfer is complete, the SPIF Flag is set.
  - the SPIF bit is cleared by first reading the SPI status register with SPIF set.
- Bit 6 (WCOL): Write Collision Flag.
  - The WCOL bit is set if the SPI data register (SPDR) is written during a data transfer.
  - the SPIF bit is cleared by first reading the SPI status register with SPIF set.
- Bit 0 (SPI2X): Double SPI Speed Bit.
  - When this bit is written logic one the SPI speed (SCK frequency) will be doubled.

Multi-slave SPI configuration

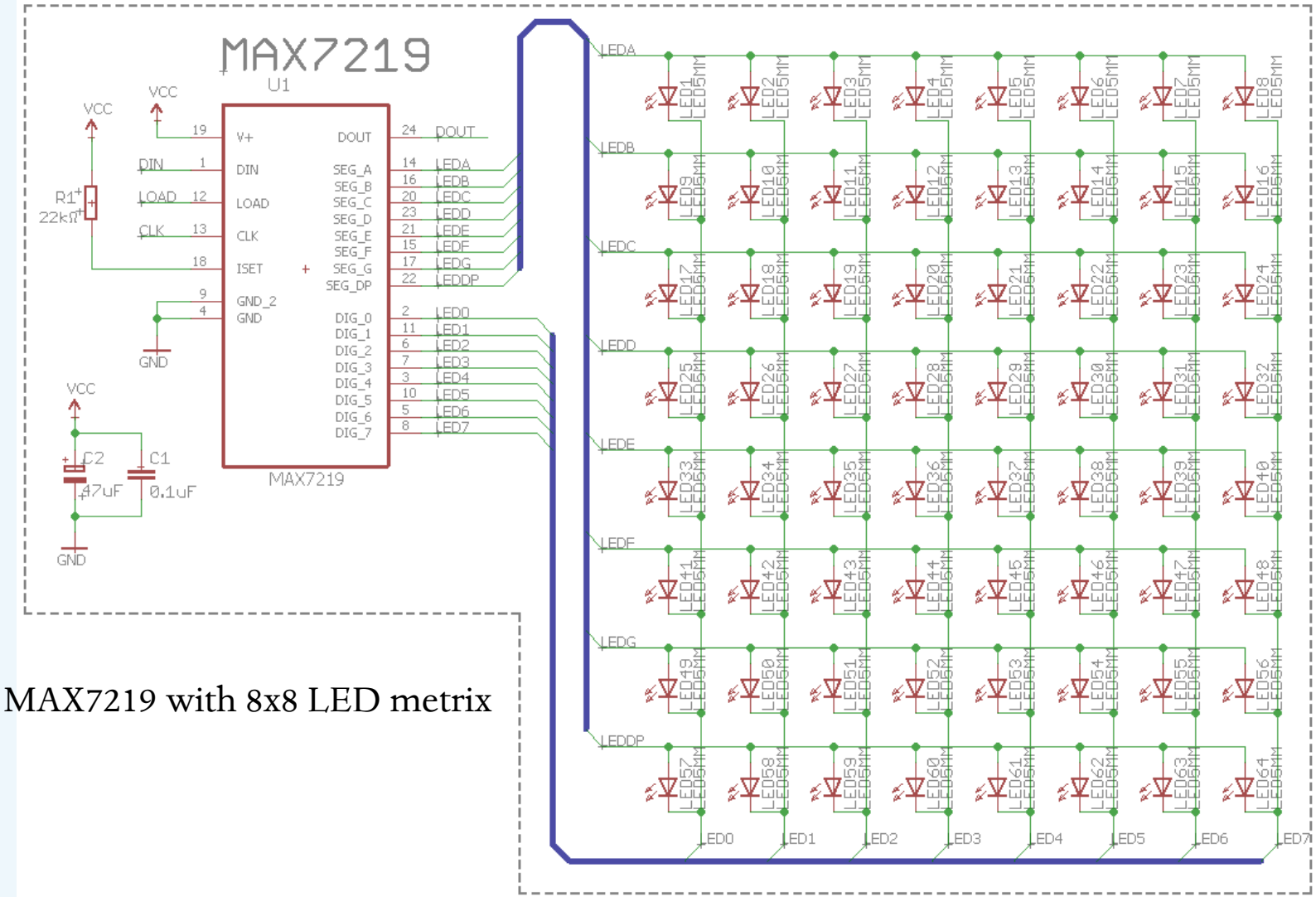


Regular configuration

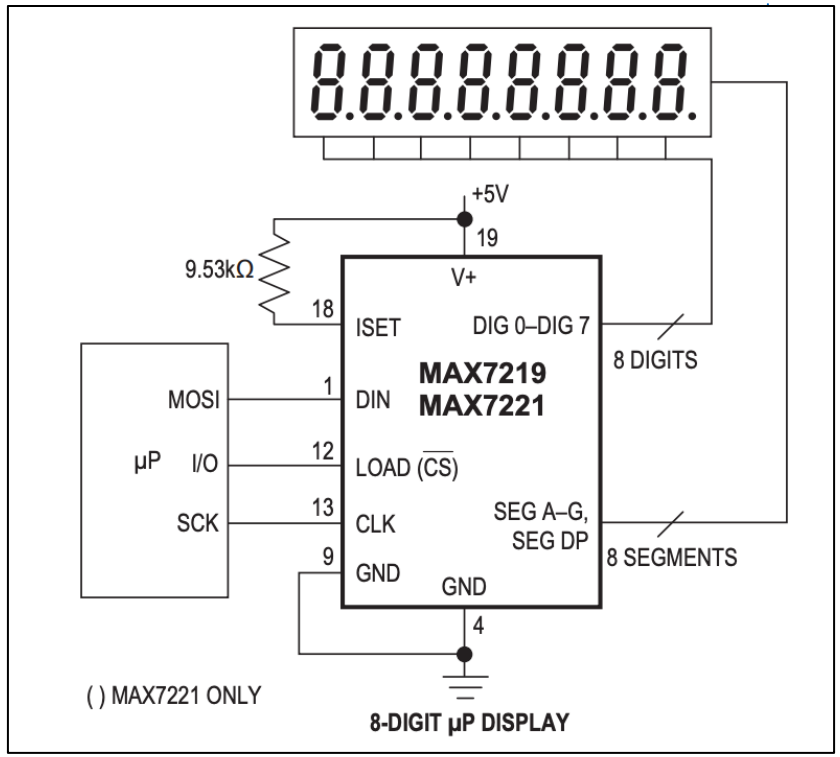


daisy-chain configuration

SPI Communication Example with MAX7219

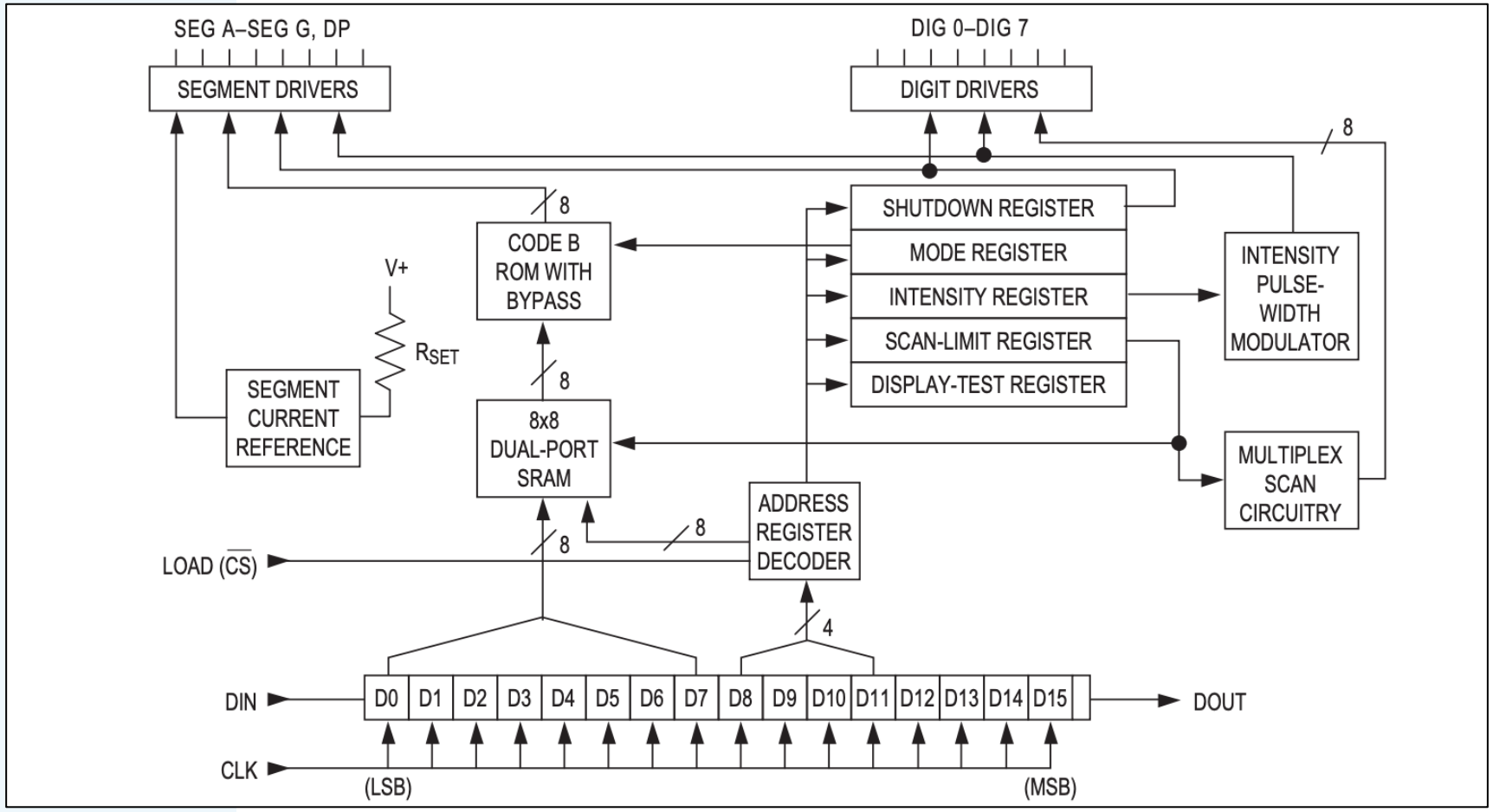


MAX7219 with 8x8 LED metrix



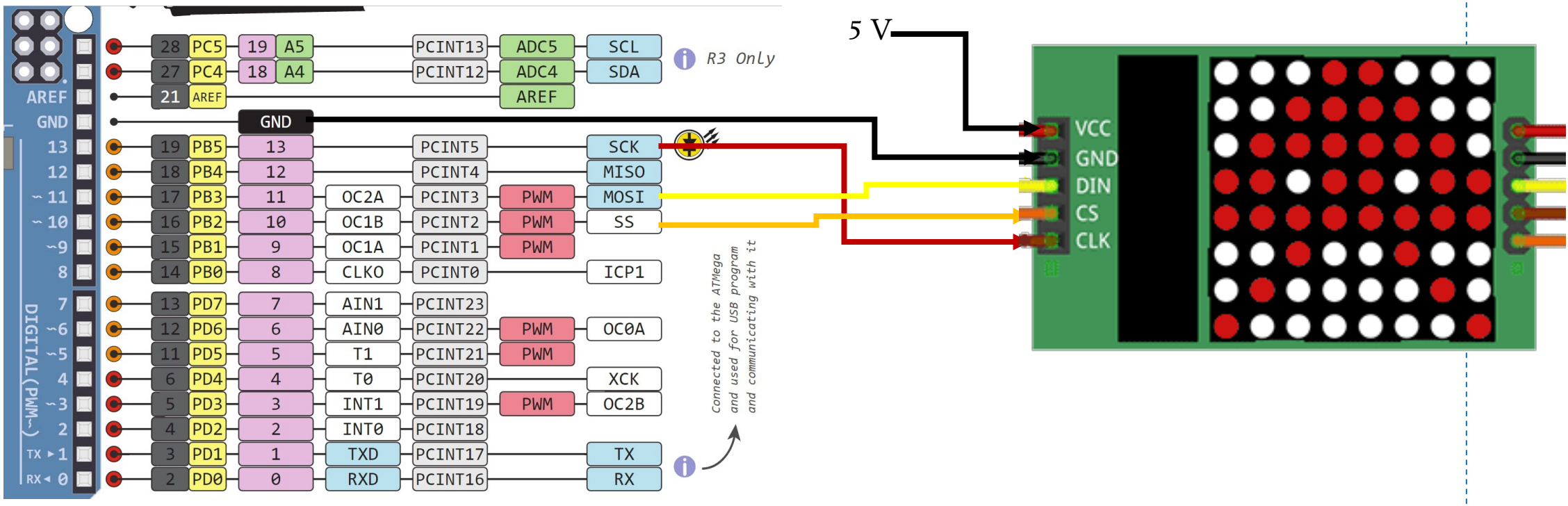
MAX7219 with 8 of 7 segments

Functional Diagram of MAX7219



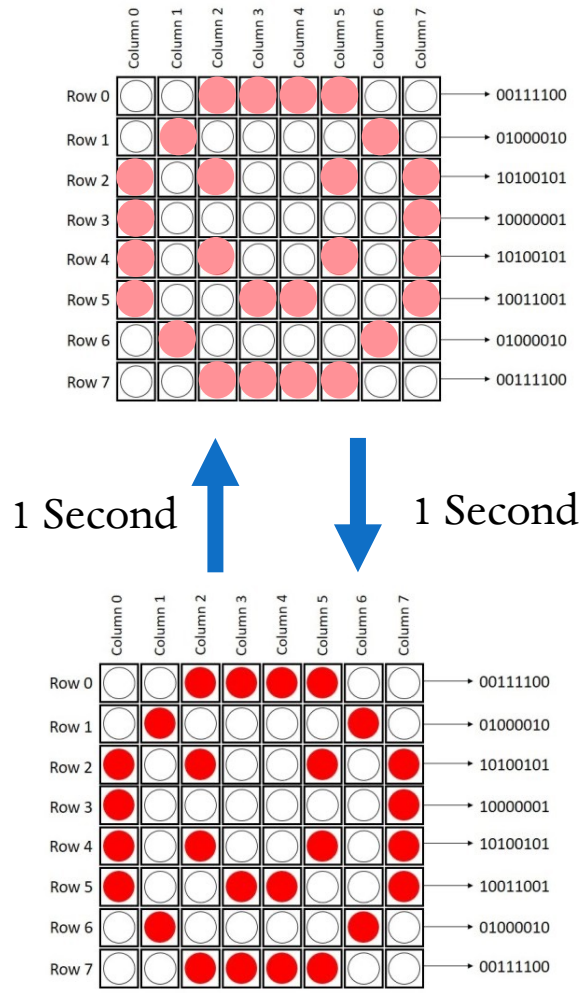
REGISTER	ADDRESS					HEX CODE
	D15–D12	D11	D10	D9	D8	
No-Op	X	0	0	0	0	0xX0
Digit 0	X	0	0	0	1	0xX1
Digit 1	X	0	0	1	0	0xX2
Digit 2	X	0	0	1	1	0xX3
Digit 3	X	0	1	0	0	0xX4
Digit 4	X	0	1	0	1	0xX5
Digit 5	X	0	1	1	0	0xX6
Digit 6	X	0	1	1	1	0xX7
Digit 7	X	1	0	0	0	0xX8
Decode Mode	X	1	0	0	1	0xX9
Intensity	X	1	0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

Wiring


5 V
VCC
GND
DIN
CS
CLK
AREF
GND
DIGITAL(PWM~)
28 PC5 19 A5 PCINT13 ADC5 SCL
27 PC4 18 A4 PCINT12 ADC4 SDA
21 AREF AREF
GND
19 PB5 13 PCINT5 SCK
18 PB4 12 PCINT4 MISO
17 PB3 11 OC2A PCINT3 PWM MOSI
16 PB2 10 OC1B PCINT2 PWM SS
15 PB1 9 OC1A PCINT1 PWM
14 PB0 8 CLKO PCINT0 ICP1
13 PD7 7 AIN1 PCINT23
12 PD6 6 AIN0 PCINT22 PWM OC0A
11 PD5 5 T1 PCINT21 PWM
6 PD4 4 T0 PCINT20 XCK
5 PD3 3 INT1 PCINT19 PWM OC2B
4 PD2 2 INT0 PCINT18
3 PD1 1 TXD PCINT17 TX
2 PD0 0 RXD PCINT16 RX
R3 Only
Connected to the ATMega and used for USB program and communicating with it

Main Function

```
int main(void) {
        spiInit();
        max7219Init();
        displayPattern();
        while (1) {
                displayChangeIntensity();
                _delay_ms(1000);
        }
        return 0;
}
```



SPI initialization

```
void spiInit(){
        // Set CS, CLK, and DATA pins as outputs
        DDRB |= (1 << CS_PIN) | (1 << CLK_PIN) | (1 << DATA_PIN);

        // Set up SPI
        // Enable SPI, Master mode, Clock = F_CPU/16
        SPCR |= (1 << SPE) | (1 << MSTR) | (1 << SPR0);
}
```

SPI Send Function

```
void spiSend(unsigned char data) {
        SPDR = data;
        while (!(SPSR & (1 << SPIF)));
}
```

MAX7219 Initialization

```
void max7219Send(unsigned char reg, unsigned char data) {
        PORTB &= ~(1 << CS_PIN); // CS low
        spiSend(reg);
        spiSend(data);
        PORTB |= (1 << CS_PIN); // CS high
}
void max7219Init() {
        max7219Send(0x09, 0x00); // Decode mode: No decode for digits 0–7
        max7219Send(0x0A, 0x0F); // Intensity: Set to maximum
        max7219Send(0x0B, 0x07); // Scan limit: Display digits 0–7
        max7219Send(0x0C, 0x01); // Shutdown mode: Normal operation
}
```

Display Smiling Face Function

```
void displayPattern() {
// Define a simple smiley face pattern
            unsigned char smiley[8] = {
                        0b00111100,0b01000010,0b10100101,0b10000001,
                        0b10100101,0b10011001,0b01000010,0b00111100
            };
// Display the pattern on the matrix
            for (int i = 0; i < 8; i++) {
                        max7219Send(i + 1, smiley[i]); // Start from register 1
            }
}
```

Change Intensity function

```
void displayChangeIntensity() {
        if(intensity == 1){intensity=15;}
        else{intensity=1;}
        max7219Send(0x0A, intensity);
}
```