MODUL 7

Serial Peripheral Interface (SPI)

Praktikum Sistem Tertanam PTA 2025/2026-1

Tujuan Praktikum

1. Mempelajari prinsip kerja SPI beserta register-register yang digunakan.

2. Mempelajari penyusunan program BMP280 berbasis SPI pada mikrokontroler ATMega328p menggunakan Bahasa C.

3. Mempelajari akuisisi data Labview menggunakan ATMega328p

Materi Pendamping

SPI merupakan komunikasi serial synchronous yang berarti harus menggunakan clock yang sama untuk mensinkronisasi deteksi bit pada receiver. Biasanya hanya digunakan untuk komunikasi jarak pendek dengan mikrokontroler lain yang terletak pada papan rangkaian yang sama. Bus SPI dikembangkan untuk menyediakan komunikasi dengan kecepatan tinggi dengan menggunakan pin mikrokontroler yang sedikit.

SPI melibatkan master dan slave. Keduanya mengirimkan dan menerima data secara terus menerus, namun master bertanggung jawab untuk menyediakan sinyal clock untuk transfer data. Gambar 3 menunjukkan komunikasi antara master dan slave pada komunikasi SPI. Master menyediakan clock dan data 8 bit pada pin master-out-slave-in (MOSI) dimana data tersebut ditransfer satu bit per pulsa clock menuju pin MOSI pada slave. Delapan bit data juga diberikan dari slave ke master melalui pin master-in-slave-out menuju pin MISO pada master. Biasanya pin $(SS)^{-}$(slave select) diberi ground (active low) untuk menjadikannya sebagai slave.

Terdapat tiga register yang digunakan untuk menginisialisasi SPI, yaitu SPSR (SPI Status Register), SPCR (SPI Control Register), dan SPDR (SPI Data Register)

SPSR (SPI Status Register)

SPSR merupakan register yang menunjukkan flag status dari komunikasi SPI.

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	

1. Bit 7 – SPIF: SPI Interrupt Flag. Pada mode master, bit ini bernilai 1 pada dua keadaan yaitu ketika serial transfer selesai, atau ketika pin SS sebagai input dan diarahkan berlogika low oleh perangkat luar. Apabila bit SPIF bernilai 1, maka akan menyebabkan interrupt apabila bit SPIE pada register SPCR bernilai 1 untuk mengaktifkan global interrupt.

2. Bit 6 – WCOL: Write Collision Flag. Bernilai 1 terjadi pada penulisan SPDR selama transfer data.
3. Bit 5:1 – RES: Reserved Bits. Bit ini belum digunakan, digunakan sebagai cadangan (reserved) oleh desainer (pabrik) pembuat chip.
4. Bit 0 – SPI2X: Double SPI Speed. Ketika SPI dalam mode master, apabila bernilai 1, maka kecepatan SPI akan dilipatgandakan.

Tabel 1. Frekuensi SCK

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	$f_{osc}/64$

SPCR (SPI Control Register)

SPCR merupakan register yang digunakan untuk mengontrol komunikasi SPI

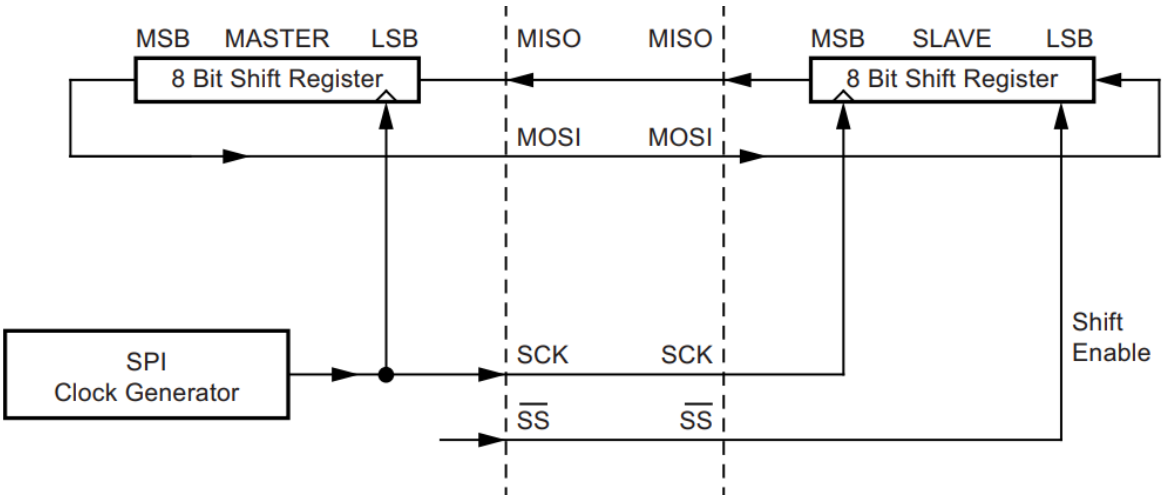
Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

1. Bit 7 – SPIE: SPI Interrupt Enable. Untuk mengaktifkan SPI interrupt, maka bit ini harus diberikan nilai 1.
2. Bit 6 – SPE: SPI Enable. Untuk mengaktifkan SPI, maka bit ini harus diberikan nilai 1.
3. Bit 5 – DORD: Data Order. Untuk memilih transmisi MSB kemudian LSB atau sebaliknya. Apabila bernilai 1 maka LSB ditransmisikan pertama kali. Sebaliknya, apabila bernilai 0 maka MSB ditransmisikan pertama kali.
4. Bit 4 – MSTR: Master/Slave Select. Apabila bernilai 1 maka mode master aktif dan apabila bernilai 0 maka mode slave aktif.
5. Bit 3 – CPOL: Clock Polarity. Untuk mengatur nilai dasar clock dalam keadaan idle. Apabila bernilai 1 maka nilai dasar clock adalah 1 dan apabila bernilai 0 maka nilai dasar clock adalah nol.
6. Bit 2 – CPHA: Clock Phase. Apabila bernilai 0 maka sampel diambil dari leading (first) clock edge dan apabila bernilai 1 maka sampel diambil dari trailing (second) clock.
7. Bit 1:0 – SPR1:0: SPI Clock Rate Select 1 and 0. Kedua bit ini untuk mengontrol besar SCK (Serial Clock) dalam Mode Master.

SPDR (SPI Data Register)

Bit	7	6	5	4	3	2	1	0	
0x2E (0x4E)	MSB							LSB	SPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

SPI Data Register adalah sebuah read/write register. Untuk menuliskan ke dalam SPI Shift Register, data harus dituliskan ke SPDR. Menulis pada register SPDR akan menginisialisasi transmisi data. Perhatikan bahwa kita tidak bisa menuliskan ke SPDR sebelum bit terakhir selesai ditransmisikan, atau akan terjadi tabrakan data. Kita dapat membaca data yang diterima sebelum bit data lainnya selesai diterima sepenuhnya.



Gambar 1. Skema transmisi pada SPI

Contoh Pemrograman Dasar: BMP280

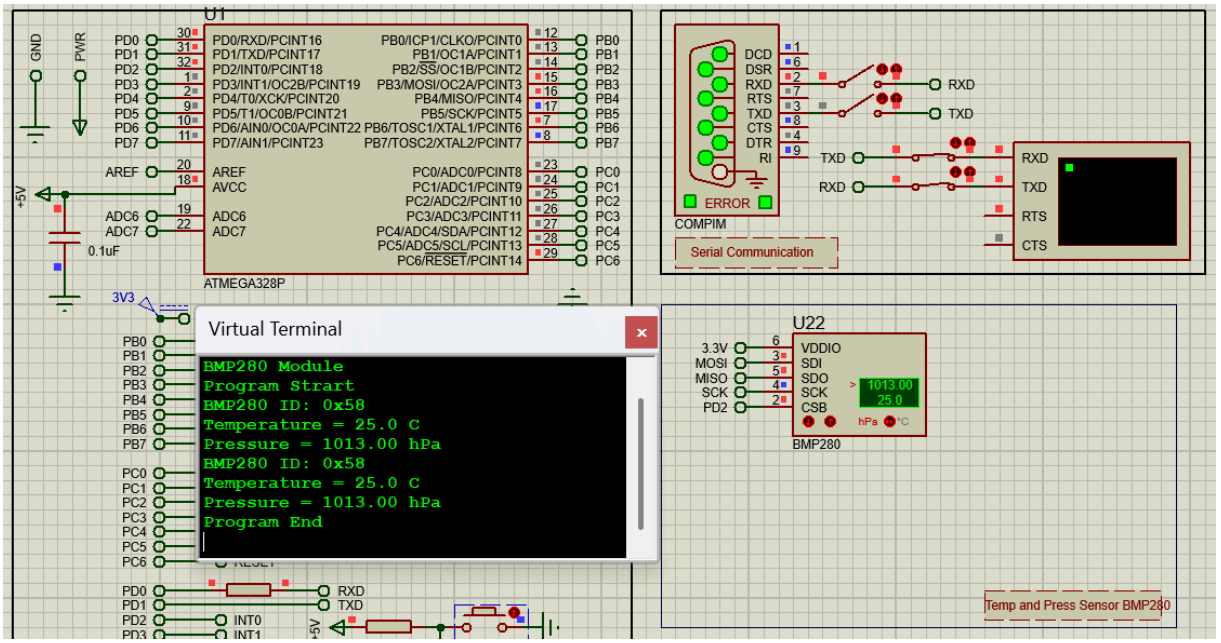
```
#include <mega328p.h>
#include <stdio.h>
#include <stdlib.h>
#include <delay.h>
#include <stdint.h>

// Declare your global variables here
#define CSB_low()(PORTD &= ~(1 << 2))
#define CSB_high()(PORTD |=  (1 << 2))

unsigned char temp_xlsb,temp_lsb,temp_msb,press_xlsb,press_lsb,press_msb;
int32_t raw_T,raw_P,Temp,Press;     //awalnya Temp_whole dan float disini
unsigned int Temp_whole,Temp_float,Press_high,Press_low;
unsigned short dig_T1;
short  dig_T2, dig_T3;
unsigned short dig_P1;
short  dig_P2, dig_P3, dig_P4, dig_P5, dig_P6, dig_P7, dig_P8, dig_P9;
long t_fine;
```

```
char DS;
uint8_t id;
unsigned char program = 0;

//Fungsi SPI
unsigned char spi_coms(unsigned char data,unsigned char RW) //RW 0 = write, 1 = read, 2 plain data  yang penting DODR = 0
{
if (RW == 0)
  {
  data &= ~(1 << 7); // set bit7 ke 0 kalo write
  }
else if (RW == 1)
  {
  data |= (1 << 7); // set bit7 ke 1 kalo read
  }
SPDR = data;
while (!(SPSR & (1 << SPIF)));   //tunggu kelar / SPIF set
return SPDR;
}

void compensation_init(void)
{
unsigned char buffer[24];
unsigned char i;
short dig[12];

CSB_low(); //SPI start
spi_coms(0x88,1);  //set pointer dan ready read
for (i=0 ; i <= 23 ; i++)
  {
  buffer[i] = spi_coms(0xFF,2);
  };
CSB_high(); //SPI stop

for (i=0 ; i<23 ; i=i+2)
  {
  dig[i/2] = buffer[i] | (buffer[i+1] << 8);
  };
dig_T1 = dig[0];
dig_T2 = dig[1];
dig_T3 = dig[2];
dig_P1 = dig[3];
dig_P2 = dig[4];
dig_P3 = dig[5];
dig_P4 = dig[6];
dig_P5 = dig[7];
dig_P6 = dig[8];
dig_P7 = dig[9];
dig_P8 = dig[10];
dig_P9 = dig[11];
```

```
}

int compensate_T(int32_t adc_T)
{
long var1, var2 , T;
var1 = ((((adc_T >> 3) - ((int32_t)dig_T1 << 1))) * ((int32_t)dig_T2)) >> 11;
var2 = (((((adc_T >> 4) - ((int32_t)dig_T1)) * ((adc_T >> 4) - ((int32_t)dig_T1))) >> 12) *((int32_t)dig_T3)) >> 14;
t_fine = var1 + var2;
T = (t_fine * 5 + 128) >> 8; //masih harus dibagi 100
   return T;
}

uint32_t compensate_P(int32_t adc_P)
{
float var1, var2, p;

var1 = ((float)t_fine / 2.0f) - 64000.0f;
var2 = var1 * var1 * ((float)dig_P6) / 32768.0f;
var2 = var2 + var1 * ((float)dig_P5) * 2.0f;
var2 = (var2 / 4.0f) + ((float)dig_P4 * 65536.0f);
var1 = (((float)dig_P3 * var1 * var1 / 524288.0f) + ((float)dig_P2 * var1)) / 524288.0f;
var1 = (1.0f + var1 / 32768.0f) * ((float)dig_P1);
if (var1 == 0.0f)
return 0;  //meninggal kalo dibagi 0

p = 1048576.0f - (float)adc_P;
p = (p - (var2 / 4096.0f)) * 6250.0f / var1;
var1 = ((float)dig_P9) * p * p / 2147483648.0f;
var2 = p * ((float)dig_P8) / 32768.0f;
p = p + (var1 + var2 + (float)dig_P7) / 16.0f;
if (p < 0.0f) p = 0.0f;
if (p > 4294967295.0f) p = 4294967295.0f;
return (uint32_t)(p + 0.5f);
}

interrupt [USART_RXC] void recieve_isr(void)
{
DS = UDR0;
if (DS == '=')
   {
   if (program == 0)
      {
      program = 2;
      printf("Program Strart\n\r");
      }
   else
      {
      program = 0;
      printf("Program End\n\r");
      }
   }
```

```
else if(DS == '-')
   {
   if (program != 0)
      {
      program = 1;
      }
   };

}

void main(void)
{
// Declare your local variables here

// Crystal Oscillator division factor: 1
#pragma optsize-
CLKPR=(1<<CLKPCE);
CLKPR=(0<<CLKPCE) | (0<<CLKPS3) | (0<<CLKPS2) | (0<<CLKPS1) | (0<<CLKPS0);
#ifdef _OPTIMIZE_SIZE_
#pragma optsize+
#endif

DDRD = 0x06; //PD2 buat chip select
PORTD.2 = 1; //Select chip selectnya harus high buat off
DDRB |= (1 << 3) | (1 << 5);  // MOSI (PB3), SCK (PB5) jadi output janlup... scknya searah
DDRB &= ~(1 << 4);            // MISO PB4 input
PORTB &= ~(1 << 4);           // no pull-up on MISO


UBRR0H = 0x00;
UBRR0L = 0x67;
UCSR0B = (1 << RXEN0) | (1 << TXEN0) | (1 << 7);
UCSR0C = (1 << 1) | (1 << 2);

SPSR &= ~(1 << SPI2X); //Matiin SPI 2x mode
SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR1); // sck_freq = 250khz, Master mode
SPCR &= ~(1 << DORD);  // MSB first

printf("main\n\r");
compensation_init();
printf("BMP280 Module\n\r");

#asm("SEI");

while (1)
    {
    // Place your code here
    if (program == 1)
     {
     CSB_low();
     spi_coms(0xD0, 1);  //pointe rke register ID
```

```
      id = spi_coms(0xFF, 2);
      CSB_high();
      printf("BMP280 ID: 0x%X\n\r", id);

      CSB_low();
      spi_coms(0xF4,0);  //pointer to ctrl meas register
      spi_coms(0b00100101,0);  //oversampling x1 temp and press, mode forced
      CSB_high();

      delay_ms(40);

      CSB_low();
      spi_coms(0xF7,1); //pointer to press_msb
      press_msb = spi_coms(0xFF,2); //baca semua data
      press_lsb = spi_coms(0xFF,2);
      press_xlsb = spi_coms(0xFF,2);
      temp_msb = spi_coms(0xFF,2);
      temp_lsb = spi_coms(0xFF,2);
      temp_xlsb = spi_coms(0xFF,2);
      CSB_high();

      raw_P = ((int32_t)press_msb << 12) | ((int32_t)press_lsb << 4) | ((int32_t)press_xlsb >> 4); //Convert to the 20bit data
      raw_T = ((int32_t)temp_msb << 12) | ((int32_t)temp_lsb << 4) | ((int32_t)temp_xlsb >> 4);
      Temp = compensate_T(raw_T);
      Press = compensate_P(raw_P);
      Temp_whole = Temp/100;
      Temp_float = Temp % 100;
      Press_high = (unsigned int)(Press / 100);
      Press_low  = (unsigned int)(Press % 100);
      if (Temp_float < 0) Temp_float = -Temp_float;
      Temp_float = Temp_float - 2; //Nge0in diproteus
      Press_low = Press_low - 3;   //Nge0in diproteus
      printf("Temperature = %d.%d C\n\r",Temp_whole,Temp_float);
      printf("Pressure = %u.%02u hPa\n\r",Press_high,Press_low);
      program = 2;
      };

    }
}
```



Gambar 2. Hasil Pemrograman Dasar

Semangat dalam Menjalakan Praktikum ❓❓❓❓❓❓❓❓❓❓