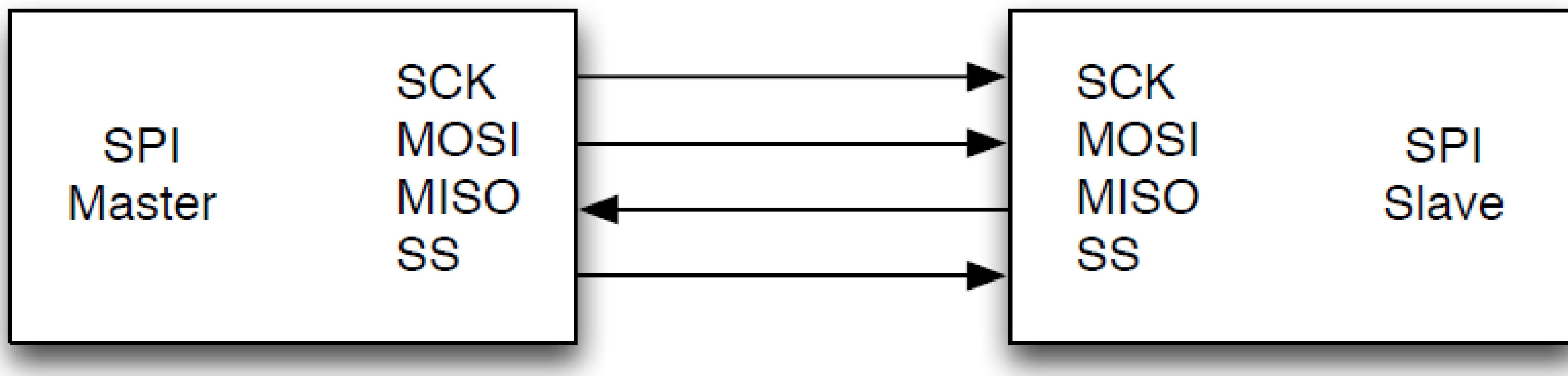
EIE3105: SPI Communication

Dr. Lawrence Cheung

Semester 2, 2021/22

SPI

- Serial Peripheral Interface Bus
  - Synchronous serial communication
  - Full duplex, master-slave architecture
  - The master device originates the frame for reading and writing.
  - Multiple slave devices are supported through selection with individual slave select (SS) lines.

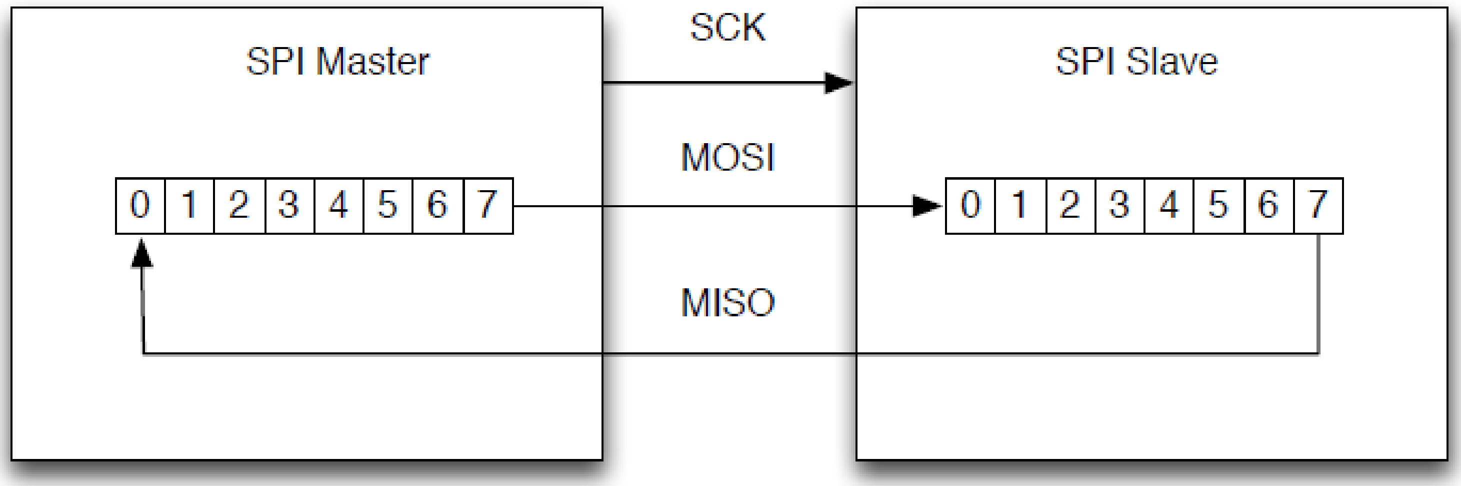


SPI

- Operations
  - SS must be replicated for every slave connected to the bus.
  - All communication is controlled by the master.
  - The master selects the slave it wishes to communicate with by lowering the appropriate SS line.
  - Then it transfer a single word (commonly one byte) serially to the slave over the MOSI (Master Out Slave In).
  - At the same time, it accepts a single byte from the slave over the MISO (Master In Slave Out).

SPI

– This transfer is realized by generating 8 clock pulses on the signal SCK (serial clock).



SPI

- The operations of the shift register (74HC299) and infra-red LEDs
  - When PB15 = 1, all LEDs are ON and the shift register is in parallel mode.
    - The shift register gets all readings from the photo-resistors.
  - When PB15 = 0, all LEDs are OFF and the shift register is in shift mode.
    - The shift register is ready to set the readings (data) back to the microcontroller.

SPI Communication

- GPIO Initialization

```
// Setup PB13 and PB15
// PB13 = SPI2_SCK, PB15 = IR LED / MODE
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_Init(GPIOB, &GPIO_InitStructure);
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_Init(GPIOB, &GPIO_InitStructure);
RCC_APB1PeriphClockCmd(RCC_APB1ENR_SPI2EN, ENABLE);
```

SPI Communication

- SPI Initialization

```
typedef struct
{
   uint16_t SPI_Direction;
   uint16_t SPI_Mode;
   uint16_t SPI_DataSize;
   uint16_t SPI_CPOL;
   uint16_t SPI_CPHA;
   uint16_t SPI_NSS;
   uint16_t SPI_BaudRatePrescaler;
   uint16_t SPI_FirstBit;
   uint16_t SPI_CRCPolynomial;
} SPI_InitTypeDef;
```

SPI Communication

- SPI_Direction = Specify the SPI unidirectional or bidirectional data mode.
  - SPI_Direction_2Lines_FullDuplex: Full duplex in 2 lines
  - SPI_Direction_2Lines_RxOnly: Receive data only in 2 lines
  - SPI_Direction_1Line_Rx: Receive data only in 1 line
  - SPI_Direction_1Line_Tx: Transmit data only in 1 line
  - Use SPI_Direction_2Lines_FullDuplex in this project.

SPI Communication

- SPI_Mode = Specify the SPI operation mode.
  - SPI_Direction_Master: Master mode
  - SPI_Direction_Slave: Slave mode
  - Use SPI_Direction_Master in this project because the microcontroller is a master to get data from the slave (74HC299).

SPI Communication

- SPI_DataSize = Specify the SPI data size.
  - SPI_DataSize_16b: 16 bits
  - SPI_DataSize_8b: 8 bits
  - Use SPI_DataSize_8b in this project because we have eight photo-resistors and one byte (eight bits) is enough to get all data (each photo-resistor gives 1-bit data, i.e., 0 or 1).

SPI Communication

- SPI_CPOL = Specify the serial clock steady state.
  - SPI_CPOL_Low: Low when the communication is ready (active)
  - SPI_CPOL_High: High when the communication is ready (active)
  - Use SPI_CPOL_High in this project. Actually both modes are fine. It depends on how your program sets the clock (PB13).

SPI Communication

- SPI_CPHA = Specify the clock active edge for the bit capture.
  - Rising is the first edge and falling is the second edge.
  - SPI_CPHA_1Edge: Data is captured on the first edge.
  - SPI_CPHA_2Edge: Data is captured on the second edge.
  - Use SPI_CPHA_2Edge in this project. The selection of this mode depends on the application.

SPI Communication

- SPI_NSS = Specify whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit.
  - SPI_NSS_Soft: By software
  - SPI_NSS_Hard: By hardware
  - Use SPI_NSS_Soft in this project because later we use program code to send and receive data.

SPI Communication

- SPI_BaudRatePrescaler = Specify the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock.
  - The communication clock is derived from the master clock.
  - The slave clock does not need to be set.
  - SPI_BaudRatePrescaler_2, SPI_BaudRatePrescaler_4, SPI_BaudRatePrescaler_8, …, SPI_BaudRatePrescaler_128 and SPI_BaudRatePrescaler_256.
  - E.g., SPI_BaudRatePrescaler_4 => 72 MHz / 4 = 18 MHz
  - We use SPI_BaudRatePrescaler_256 in this project because it is fast enough to get data.

SPI Communication

- SPI_FirstBit = Specify whether data transfers start from MSB or LSB bit.
  - SPI_FirstBit_MSB: MSB
  - SPI_FirstBit_LSB: LSB
  - We use SPI_FirstBit_MSB in this project but it is up to you.

SPI Communication

- SPI_SPI_CRCPolynomial = Specify the polynomial used for the CRC calculation.
  - 7: The polynomials present as a binary representation, with the high order bit inferred as it is always one and just beyond the scope of the register.
    - 0x07 => 0x107 => $x^8 + x^2 + x + 1$
  - 0: Do not use it. Sometimes it is even skipped (not to set any values into it).
  - We skip it in this project.

SPI Communication

- SPI Initialization (program code)

```
// SPI initialization
SPI_InitTypeDef   SPI_InitStructure;
SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
SPI_InitStructure.SPI_CPOL = SPI_CPOL_High;
SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;
SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
// 36 MHz / 256 = 140.625 kHz
SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_256;
SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
SPI_Init(SPI2, &SPI_InitStructure);
// Enable the receive interrupt
SPI_I2S_ITConfig(SPI2, SPI_I2S_IT_RXNE, ENABLE);
// Enable SPI2
SPI_Cmd(SPI2, ENABLE);
```

SPI Communication

```
void SPI_Init(SPI_TypeDef *SPIx, SPI_InitTypeDef
*SPI_InitStruct);
```

– Initialize the SPIx peripheral according to the specified parameters in the SPI_InitStruct.
  - SPIx = SPI1 or SP2

```
void SPI_Cmd (SPI_TypeDef *SPIx, FunctionalState
NewState);
```

– Enable or disable the specified SPI peripheral.

SPI Communication

- SPI Communication to get data from photo-resistors (program code)

```
// when this function is called, we will get the readings from
// the SPI2 IRQ handler
void readFloor()
{
     // Set PB15 to 1
     GPIO_SetBits(GPIOB, GPIO_Pin_15);
     // Initialize the data transmission from the master to the slave
     SPI_I2S_SendData(SPI2, 0);
     // Enable the interrupt to receive data by using the ISR handler
     NVIC_EnableIRQ(SPI2_IRQn);
}
```

SPI Communication

- SPI Communication to get data from photo-resistors (program code)

```
// put the readings to the variable c
void SPI2_IRQHandler() {
      // the received character has all the readings, valid in 2nd time
      char c = (char) SPI_I2S_ReceiveData(SPI2) & 0xff;
      // Check PB15. If it is 1, it means the data is ready
      if (GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_15) == 1) {
        // Set PB15 to 0 to trigger the shift register
        GPIO_ResetBits(GPIOB, GPIO_Pin_15);
        // Go to get the next reading
        SPI_I2S_SendData(SPI2, 0);
      }
      else {
        // disable the interrupt because it is not ready
        NVIC_DisableIRQ(SPI2_IRQn);
      }
}
```

SPI Communication

```
uint16_t SPI_I2S_ReceiveData(SPI_TypeDef *SPx)I
```

– Return the most recent received data by the SPIx/I2Sx peripheral.

```
void SPI_I2S_SendData (SPI_TypeDef *SPIx,
                        uint16_t Data)
```

– Transmit a Data through the SPIx/I2Sx peripheral.

Reference Readings

- Chapter 5, 6, 9 and 11 – *Discovering the STM32 Microcontroller*, Geoffrey Brown, 2012
- RM0008 Reference Manual (STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced ARM-based 320bit MCUs)
- Datasheet – STM32F103x8, STM32F103xB

End