Lecture 15: STM32 Serial Communication II

Seyed-Hosein Attarzadeh-Niaki

Review

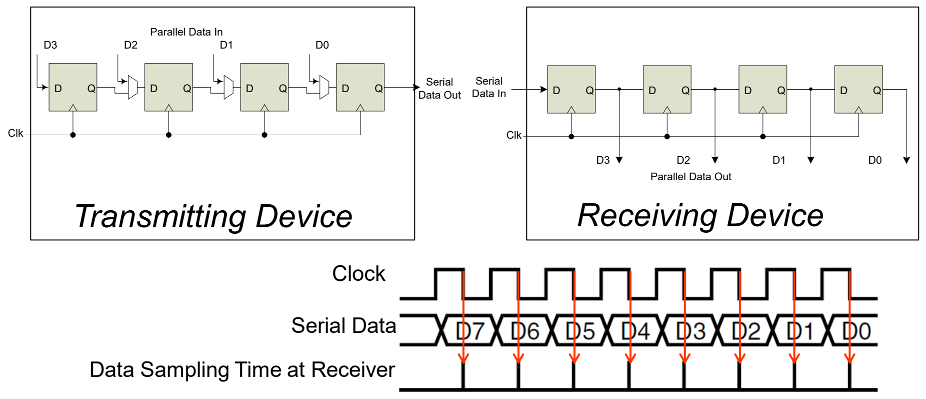
- Serial communication concepts
  - Synchronous and asynchronous serial buses
- STM32F401 universal synchronous asynchronous receiver transmitter (USART)
  - Baud-rate generation
  - Registers
  - Programming with polling and interrupt
- Basic UART problems and solutions

Outline

- Serial peripheral interface (SPI)
  - SPI bus protocol
  - SPI in STM32
  - SPI examples

SERIAL PERIPHERAL INTERFACE (SPI)

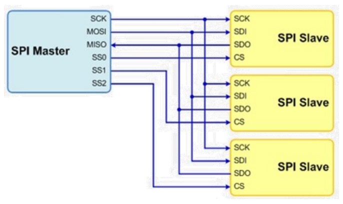
Recall: ***Synchronous*** Serial Data Transmission

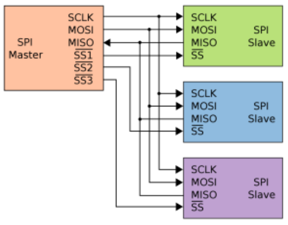


- Use shift registers and a clock signal to convert between serial and parallel formats
- Synchronous: an explicit clock signal is along with the data signal

The SPI Bus Protocol

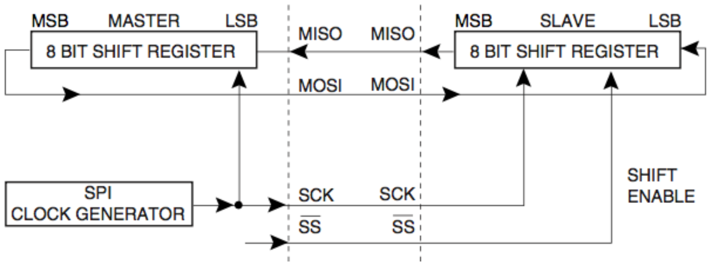
- SPI (serial peripheral interface)
  - 4-wire interface
- Originally by Motorola (later Freescale, now NXP)
- Used for short distance communication, primarily in embedded systems.
- Two pins used for data transfer
  - **SDI** (Din) and **SDO** (Dout)
- **SCLK** (shift clock) pin synchronizes data transfer between two chips.
- **CE** (chip enable) initiates and terminates the data transfer
- SDI, SDO, SCLK, CE A.K.A. **MOSI**, **MISO**, **SCK**, **SS**
- SPI interface of STM32F4 also supports I^{2}S (Inter-IC Sound) audio protocol.



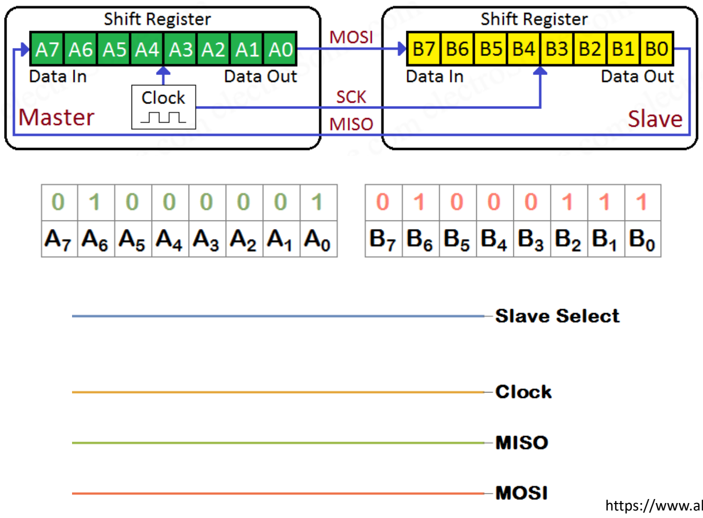


The SPI Bus Protocol

- The *master clock* generator provides clock to the shift registers in both the master and slave
- The *clock input* of the shift registers can be *falling- or rising-edge triggered*
- Shift registers are 8 bits long
- Master send: master places the byte in shift register and generates 8 clock pulses
- Master receive: slave places the byte in its shift register and after 8 clock pulses the data will be received by the master shift register
- SPI is *full duplex*



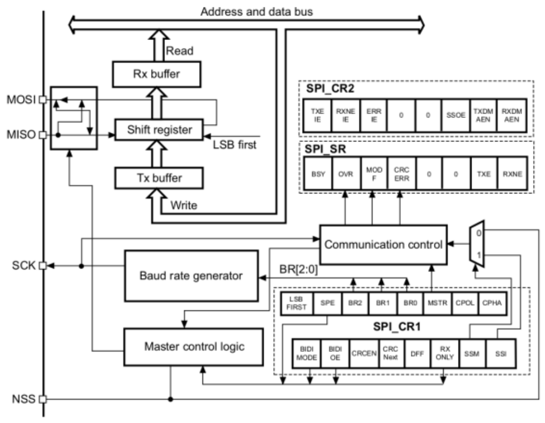
SPI In Action!



https://www.allaboutcircuits.com

SPI Block Diagram and Features

- Full-duplex synchronous transfers on three lines
- Simplex synchronous transfers on two lines with or without a bidirectional data line
- 8- or 16-bit transfer frame format selection
- Master or slave operation
- Multimaster mode capability
- 8 master mode baud rate prescalers ($f_{PCLK}/2$ max.)
- Slave mode frequency ($f_{PCLK}/2$ max)
- Programmable clock polarity and phase
- Programmable data order with MSB-first or LSB-first shifting
- Dedicated transmission and reception flags with interrupt capability
- SPI bus busy status flag
- Hardware CRC feature for reliable communication:
  - CRC value can be transmitted as last byte in Tx mode
  - Automatic CRC error checking for last received byte
- Master mode fault, overrun and CRC error flags with interrupt capability
- 1-byte transmission and reception buffer with DMA capability: Tx and Rx requests
- Supports I²S



Using the SPI

- Slave mode
  - Decide data frame format (DFF)
  - Select the relationship (CPOL/CPHA)
  - MSB or LSB first? (LSBFIRST in CR1)
  - Using DMA? (DMAT in CR3)
  - Handle the NSS or SSM and SSI bit depending on the mode
  - TI mode protocol? (FRF in CR2)
  - Clear the MSTR and set SPE in CR1
- MOSI is input and MISO is output
- Transmit
  - Parallel-load data byte into Tx buffer during a write cycle
  - Transfer data from buffer to shift register
- Receive
  - Transfer data from shift register to Rx buffer and set the RXNE flag

- Master mode
  - Baud rate (BR in CR1)
  - Select the relationship (CPOL/CPHA)
  - Decide data frame format (DFF)
  - MSB or LSB first? (LSBFIRST in CR1)
  - Handle the NSS or SSM and SSI bit depending on the mode
  - TI mode protocol? (FRF in CR2)
  - Set MSTR and SPE in CR1
- MOSI is output and MISO is input
- Transmit
  - Write a byte into Tx buffer
  - Transfer data from buffer to shift register
- Receive
  - Transfer data from shift register to RX buffer and set the RXNE flag

SPI Control Register 1 (SPI_CR1)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIDI MODE	BIDI OE	CRC EN	CRC NEXT	DFF	RX ONLY	SSM	SSI	LSB FIRST	SPE	BR [2:0]			MSTR	CPOL	CPHA
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- DFF: data frame format
  - 0: 8-bit ; 1: 16-bit
- LSBFIRST: frame format
  - 0:MSB first; 1:LSB first
- SPI: SPI enable
- BR[2:0]:Baud rate control
  - 000: $f_{PCLK}/2$; 100: $f_{PCLK}/32$
  - 001: $f_{PCLK}/4$; 101: $f_{PCLK}/64$
  - 010: $f_{PCLK}/8$; 110: $f_{PCLK}/128$
  - 011: $f_{PCLK}/16$; 111: $f_{PCLK}/256$
- MSTR: Master selection
  - 0: Slave; 1:Master

SPI Control Register 2 (SPI_CR2)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TXEIE	RXNEIE	ERRIE	FRF	Res.	SSOE	TXDMAEN	RXDMAEN
								rw	rw	rw	rw		rw	rw	rw

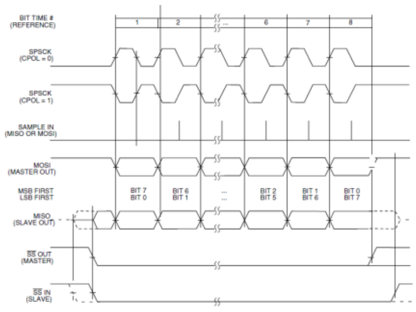
- TXEIE: Tx buffer empty interrupt enable
- RXNEIE: Rx buffer not empty interrupt enable
- ERRIE: Error interrupt enable
- FRF: Frame format
  - 0: SPI Motorola mode
  - 1: SPI TI mode
- SSOE: SS output enable
- TXDMAEN: Tx buffer DMA enable
- RXDMAEN: Rx buffer DMA enable

Clock Polarity and Phase in SPI Device

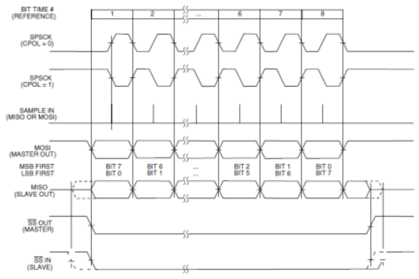
- The master and slave(s) must agree on the *CPOL (clock polarity)* and *CPHA (clock phase)* with respect to the data.
- CPOL = 0: the base value of the clock is zero
  CPOL = 1: the base value of the clock is one
- CPHA = 0: change on the trailing (2nd) clock edge
  sample on the leading (1st) clock edge
  CPHA = 1: change on the leading (1st) clock edge
  sample on the trailing (2nd) clock edge
- If the base value of the clock is zero, the leading (first) clock edge, is the rising edge
  if the base value of the clock is one, the leading (first) clock edge is falling edge.

Clock and Phase Settings

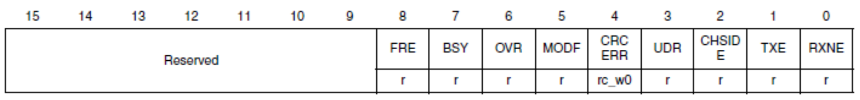
CPHA = 1



CPHA = 0



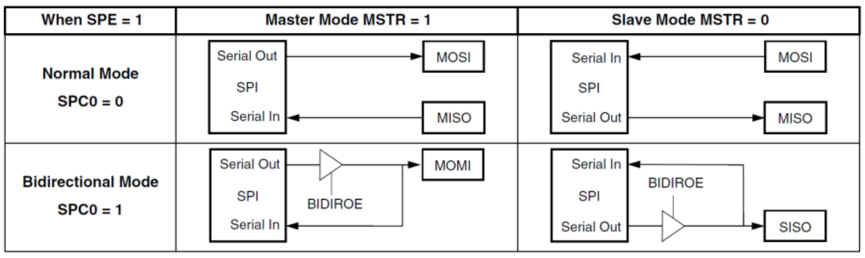
SPI Status Register (SPI_SR)



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							FRE	BSY	OVR	MODF	CRC ERR	UDR	CHSIDE	TXE	RXNE
							r	r	r	r	rc_w0	r	r	r	r

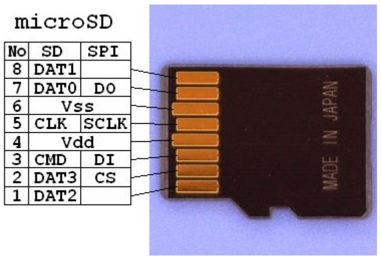
- FRE: Frame format error
- BSY: Busy flag
- OVR: Overrun flag
- MODF: Mode fault
- UDR: Underrun flag
- TXE: Transmit buffer empty
- RXNE: Receive buffer not empty

Normal and Bidirectional Modes

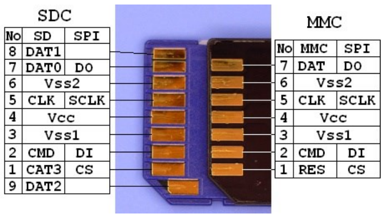


SPI Example: Secure Digital Card Access

- SD cards have two communication modes
  - Native 4-bit
  - Legacy SPI 1-bit
- SPI mode 0
  - CPHA=0
  - CPOL=0
- VDD from 2.7 to 3.6 V
- CS: Chip Select (active low)
- Source – FatFS FAT File System Module:
  - http://elm-chan.org/docs/mmc/mmc_e.html
  - http://elm-chan.org/fsw/ff/00index_e.html



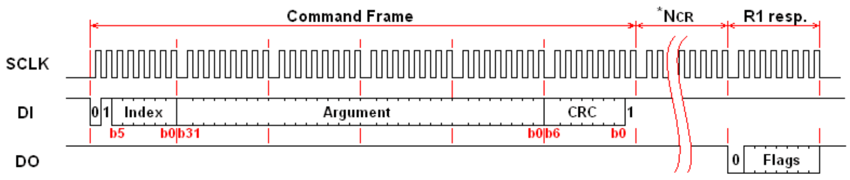
No	SD	SPI
8	DAT1	
7	DAT0	DO
6	Vss	Vss
5	CLK	SCLK
4	Vdd	Vdd
3	CMD	DI
2	DAT3	CS
1	DAT2	



No	SD	SPI
8	DAT1	
7	DAT0	DO
6	Vss2	Vss2
5	CLK	SCLK
4	Vcc	Vcc
3	Vss1	Vss1
2	CMD	DI
1	CAT3	CS
9	DAT2	

No	MMC	SPI
7	DAT	DO
6	Vss2	Vss2
5	CLK	SCLK
4	Vcc	Vcc
3	Vss1	Vss1
2	CMD	DI
1	RES	CS

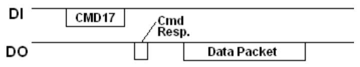
SPI Commands



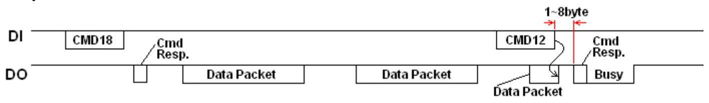
- Host sends a six-byte command packet to card
  - Index, argument, CRC
- Host reads bytes from card until card signals it is ready
  - Card returns
    - 0xff while busy
    - 0x00 when ready without errors
    - 0x01-0x7f when error has occurred

SD Card Transactions

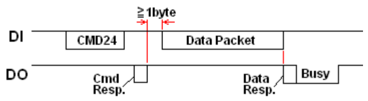
- Single Block Read



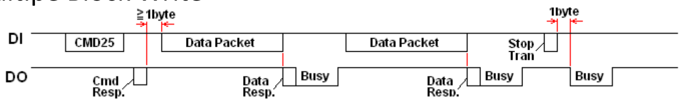
- Multiple Block Read



- Single Block Write

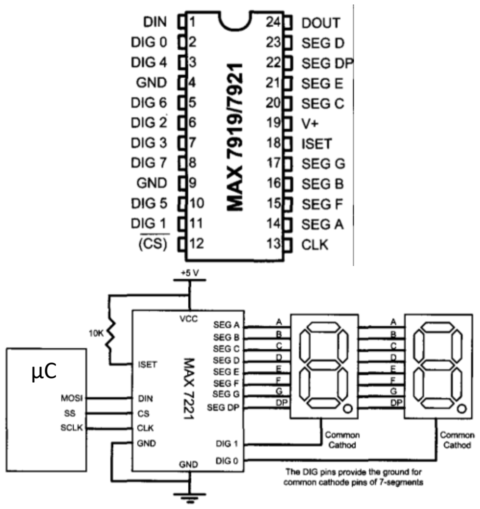


- Multipe Block Write

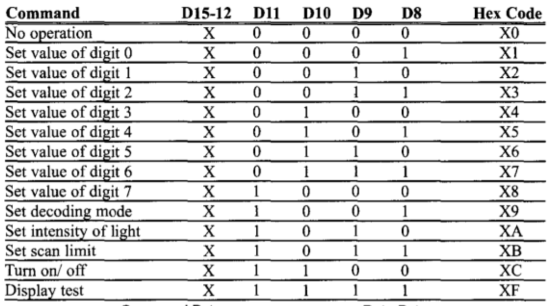


MAX7221

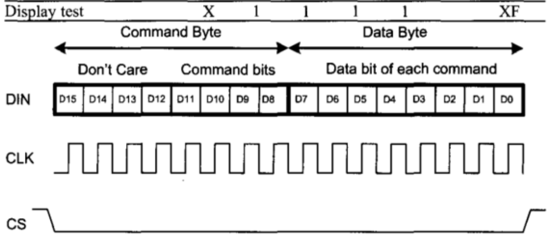
- Contains an internal decoder that converts binary numbers to 7-seg codes
- Send a binary number to MAX7221, and the chip decodes the binary data and displays the number
- Can be controlled using SPI



MAX7221 Packet Format

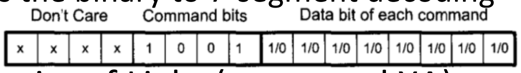


Command	D15-12	D11	D10	D9	D8	Hex Code
No operation	X	0	0	0	0	X0
Set value of digit 0	X	0	0	0	1	X1
Set value of digit 1	X	0	0	1	0	X2
Set value of digit 2	X	0	0	1	1	X3
Set value of digit 3	X	0	1	0	0	X4
Set value of digit 4	X	0	1	0	1	X5
Set value of digit 5	X	0	1	1	0	X6
Set value of digit 6	X	0	1	1	1	X7
Set value of digit 7	X	1	0	0	0	X8
Set decoding mode	X	1	0	0	1	X9
Set intensity of light	X	1	0	1	0	XA
Set scan limit	X	1	0	1	1	XB
Turn on/ off	X	1	1	0	0	XC
Display test	X	1	1	1	1	XF



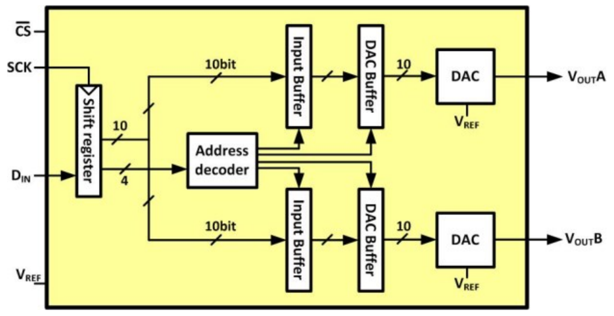
MAX7221 Commands

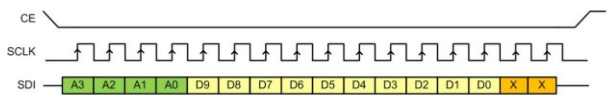
- Set value of digit 0 - digit 7 (commands X1-X8)
  - Either BCD or decoded values
- Set decoding mode (command X9)
  - Bypass the binary to 7-segment decoding



- Set Intensity of Light (command XA)
  - A value between 0-16
- Set Scan Limit (command XB)
  - Sets the number of connected 7-segments
- Turn On/Off (command XC)

LTC1661 DAC



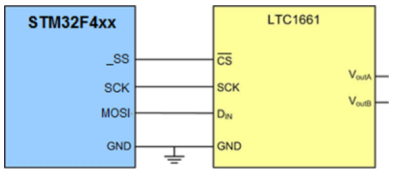


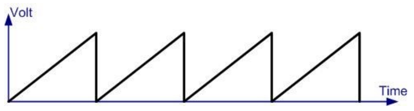
LTC1661 DAC Control Functions

A3 A2 A1 A0	Interrupt Register	DAC Register	Power Down Status	Comments
0 0 0 0	No Change	No Update	No Change	No operation. power-down status unchanged
0 0 0 1	Load DAC A	No Update	No Change	Load input register A with data. DAC outputs unchanged. power-down Status unchanged
0 0 1 0	Load DAC B	No Update	No Change	Load input register B with data. DAC outputs unchanged. power-down status unchanged
1 0 0 0	No Change	Update Outputs	Wake	Load both DAC Regs with existing contents of input Regs. Outputs update. Part wakes up
1 0 0 1	Load DAC A	Update Outputs	Wake	Load input Reg A. Load DAC Regs with new contents of input Reg A and existing contents of Reg B. Outputs update.
1 0 1 0	Load DAC B	Update Outputs	Wake	Load input Reg B. Load DAC Regs with existing contentsof input Reg A and new contents of Reg B. Outputs update
1 1 0 1	No Change	No Update	Wake	Part wakes up. Input and DAC Regs unchanged. DAC outputs reflect existing contents of DAC Regs
1 1 1 0	No Change	No Update	Sleep	Part goes to sleep. Input and DAC Regs unchanged. DAC outputs set to high impedance state
1 1 1 1	Load ADCs A, B with same 10-bit code	Update Outputs	Wake	Load both input Regs. Load both DAC Regs with new contents of input Regs. Outputs update. Part wakes up

Example: Using SPI1 to generate a sawtooth waveform on LTC1661

- SPI1 is configured as master with software slave select.
- Clock rate is set to 1 MHz.
- Polarity/Phase are 0, 0
- PA7 MOSI
- PA5 SCK
- PA4 SS





```
#include "stm32f4xx.h"

void SPI1_init(void);
void DAC_write(short data);

int main(void) {
    short i;
    SPI1_init();
    while(1) {
        for (i = 0; i < 1024; i++) {
            DAC_write(i);      /* write the letter through SPI1 */
        }
    }
}
/* enable SPI1 and associated GPIO pins */
void SPI1_init(void) {
    RCC->AHB1ENR |= 1;           /* enable GPIOA clock */
    RCC->APB2ENR |= 0x1000;      /* enable SPI1 clock */

    /* PORTA 5, 7 for SPI1 MOSI and SCLK */
    GPIOA->MODER &= ~0x0000CC00;    /* clear pin mode */
    GPIOA->MODER |=  0x00008800;    /* set pin alternate mode */
    GPIOA->AFR[0] &= ~0xF0F00000;   /* clear alt mode */
    GPIOA->AFR[0] |=  0x50500000;   /* set alt mode SPI1 */

    /* PORTA4 as GPIO output for SPI slave select */
    GPIOA->MODER &= ~0x00000300;    /* clear pin mode */
    GPIOA->MODER |=  0x00000100;    /* set pin output mode */

    SPI1->CR1 = 0x31C;             /* set the Baud rate, 8-bit data frame */
    SPI1->CR2 = 0;
    SPI1->CR1 |= 0x40;                /* enable SPI1 module */
}
/* This function enables slave select, writes one byte to SPI1,
   wait for transmit complete and deassert slave select. */
void DAC_write(short data) {
    data &= 0x03FF;                   /* make sure data is 10-bit */
    while (!(SPI1->SR & 2)) {}        /* wait until Transfer buffer Empty */
    GPIOA->BSRR = 0x00100000;         /* deassert slave select */
    SPI1->DR = 0x90 | (data >> 6);    /* write command and upper 4 bits of data */
    while (!(SPI1->SR & 2)) {}        /* wait until Transfer buffer Empty */
    SPI1->DR = (data << 2) & 0xFF;
    while (SPI1->SR & 0x80) {}        /* wait for transmission done */
    GPIOA->BSRR = 0x00000010;         /* assert slave select */
}
```

Next Lecture

- I^2C communication