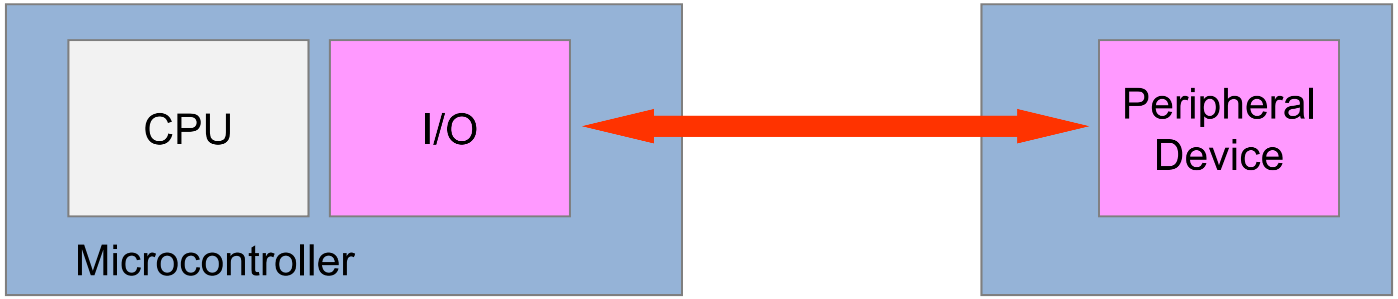
Serial Data Transfer

Computer Engineering 2

Motivation

- **Communication CPU – Peripheral Devices**



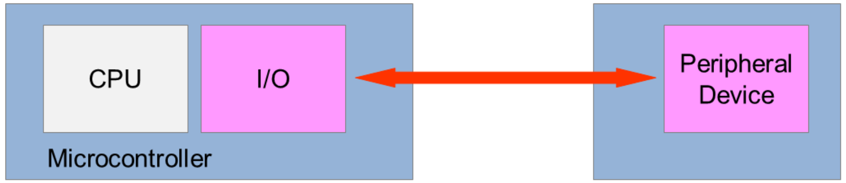
- **Parallel Bus**
  - n address lines
  - m data lines
  - Control lines (NWE, NOE)
  - Decode logic

- **Serial Bus**
  - 2 to 4 lines
    - (CLK)
    - Data: Din, Dout
    - (Select)

Motivation

Serial Connection → UART, SPI, I2C, etc.

- Provide simple, low level physical connection
  - Simpler → save PCB area
  - Reduce number of switching lines → reduce power, improve EMC [1)]



- Requires "higher level" protocol
  - Usually in software
  - Error detection, reliability, quality-of-service (QoS)
  - Interpretation of commands

1) Electromagnetic compatibility:
Simultaneous switching of lines creates unwanted emission

Agenda

- **Serial Communication**
- **Asynchronous Serial Interface**
  - Universal Asynchronous Receiver Transmitter – UART
  - Longer Distances: RS-232 / RS-485
- **SPI – Serial Peripheral Interface**
  - SPI Basics
  - SPI Modes
  - SPI – STM32F4xxx
  - SPI – Flash Devices
- **I2C Bus**
  - Operation
- **Comparison**

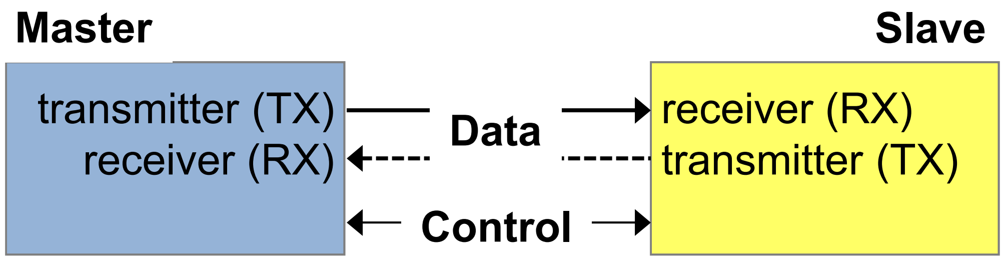
Learning Objectives

At the end of this lesson you will be able

- to explain how a UART works and which synchronization mechanism is used between transmitter and receiver
- to draw and interpret UART timing diagrams including data and overhead bits
- to explain what SPI is and how it works
- to outline the differences between the four SPI modes of operation
- to draw and interpret SPI timing diagrams
- to interpret the SPI block diagram of the STM32F4xxx
- to outline how SPI data transmission and reception has to be handled by software on the STM32F4xxx
- to explain what I2C is and how it works
- to interpret an I2C timing diagram

Serial Communication

- **Set-up**
  - Serial data line(s)
  - Optional control lines



- **Communication Modes**

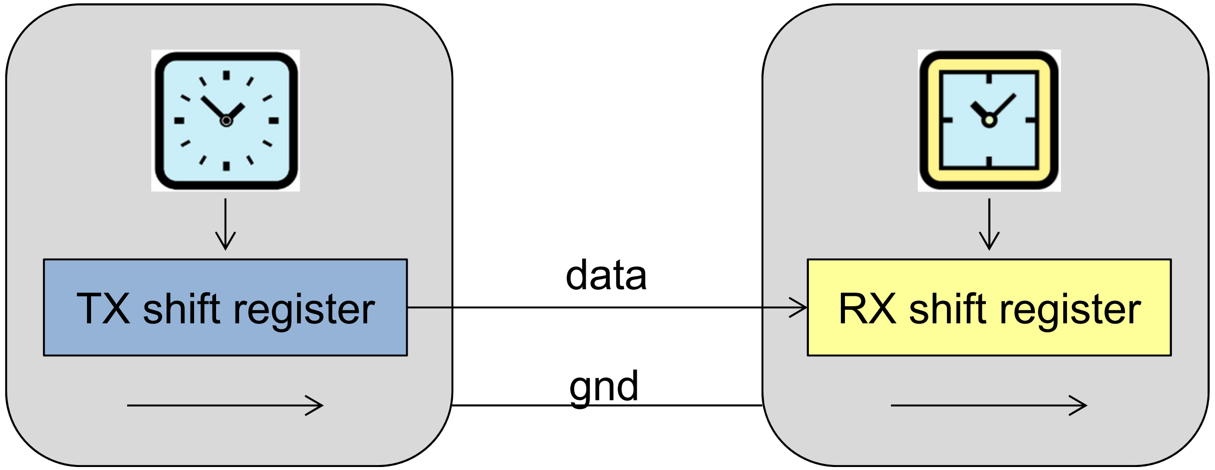
Simplex	Unidirectional, one way only
Half-duplex	Bidirectional, only one direction at a time
Full-duplex	Bidirectional, both directions simultaneously

- **Timing**

Asynchronous	Each node uses an individual clock
Synchronous	Both nodes use same clock Clock often provided by master

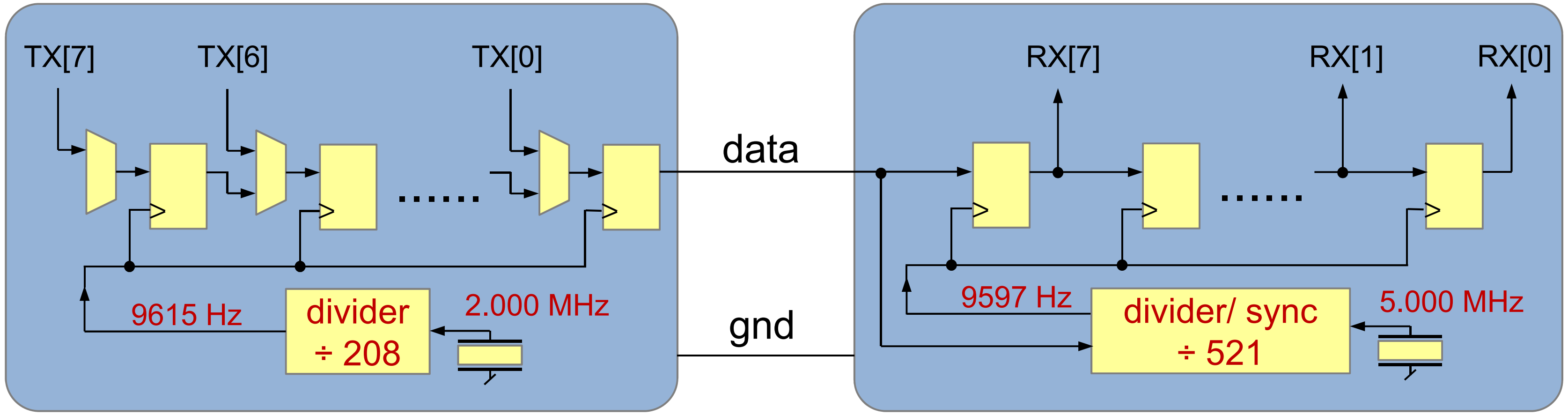
Asynchronous Serial Interface

- **Universal Asynchronous Receiver Transmitter – UART**
  - Connecting shift registers with diverging clock sources
    - Same target frequency
    - Different tolerances and divider ratios
    - Requires synchronization at start of each data item in receiver



Asynchronous Serial Interface

- **Implementation of UART → Shift register**
  - Example: 9'600 Baud with selected quartz frequencies



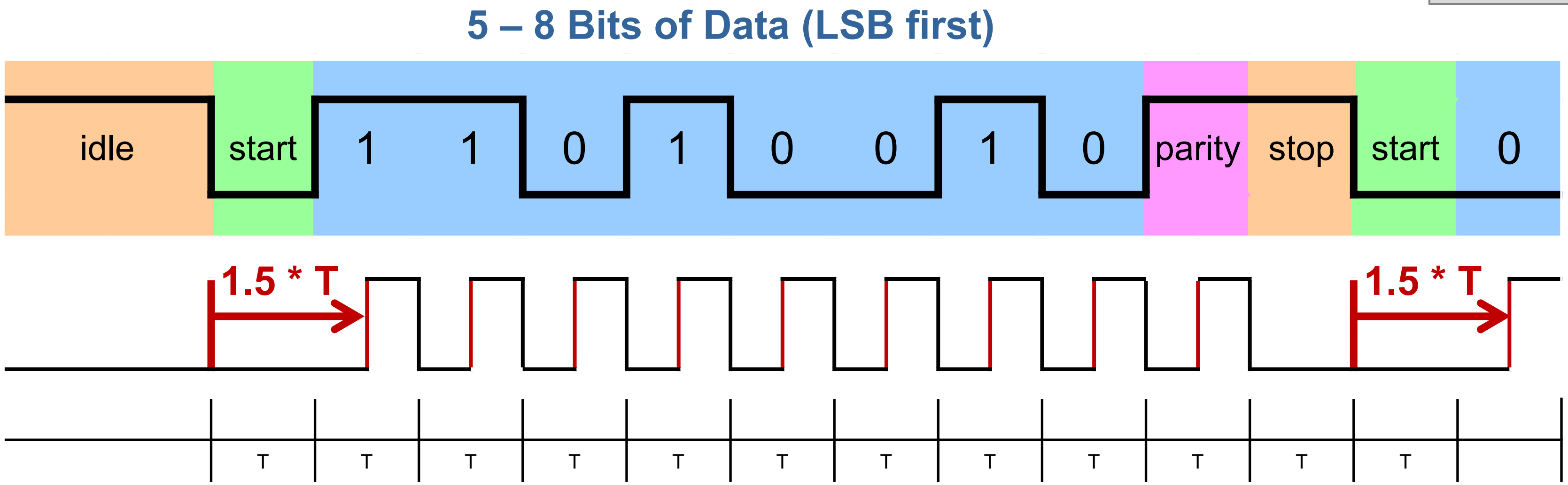
  - Transmitter and receiver use closest integer to divide clock
    - 2.000 ÷ 208 = 9615 Hz → slightly too fast
    - 5.000 ÷ 521 = 9597 Hz → slightly too slow

settings allow successful transmission

Asynchronous Serial Interface

UART Timing

e.g. 9'600 Baud = symbols / s
➔ T = (1 / 9600) s = 0.104 ms



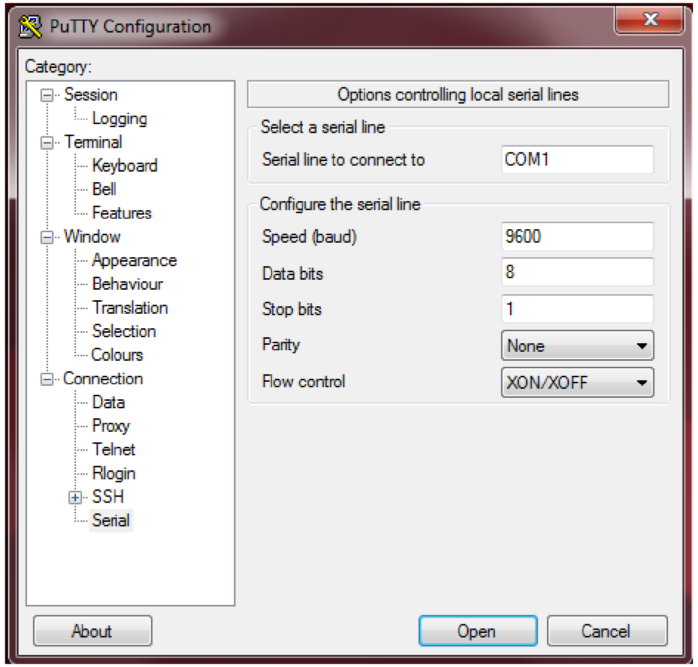
Idle → logic '1'
stop → logic '1'
start → logic '0'

- Transition stop ('1') → start ('0')
  - Receiver detects edge at the start of each data block (5 to 8 bits)
  - Allows receiver to sample data "in middle of bits" → red edges
  - Clocks have to be accurate enough to allow sampling up to parity bit, i.e. max. +/- 0.5 bit times aggregated deviation until last bit

Asynchronous Serial Interface

UART – TX and RX Work with Same Settings

- Transmission rate
  - 2400, 4800, 9600, 19200, 38'400, 57'600, 115'200 … bit/s
  - In case of UART: symbol rate = bit rate i.e. baud corresponds to bit/s
- Number of data bits
  - Between 5 and 8 bit
- Number of stop bits
  - 1, 1.5 or 2 bit
- Parity
  - none
  - mark (logic '1')
  - space (logic '0')
  - even
  - odd



Options controlling local serial lines

Select a serial line

Serial line to connect to: COM1

Configure the serial line

Speed (baud): 9600

Data bits: 8

Stop bits: 1

Parity: None

Flow control: XON/XOFF

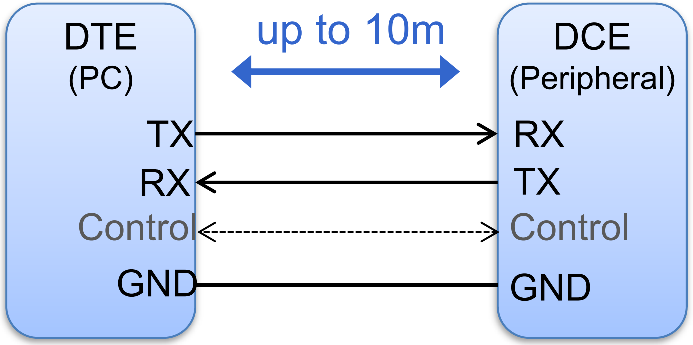
Asynchronous Serial Interface

UART Characteristics

- Synchronization
  - Each data item (5-8 bits) requires synchronization
- Asynchronous data transfer
  - Mismatch of clock frequencies in TX and RX
  - Requires overhead for synchronization → additional bits
  - Requires effort for synchronization → additional hardware
- Advantage
  - Clock does not have to be transmitted
  - Transmission delays are automatically compensated
- On-board connections
  - Signal levels are 3V or 5V with reference to ground
  - Off-board connections require stronger output drivers (circuits)

Asynchronous Serial Interface

RS–232[1] – Interconnecting Equipment by Cable



source: Olimex

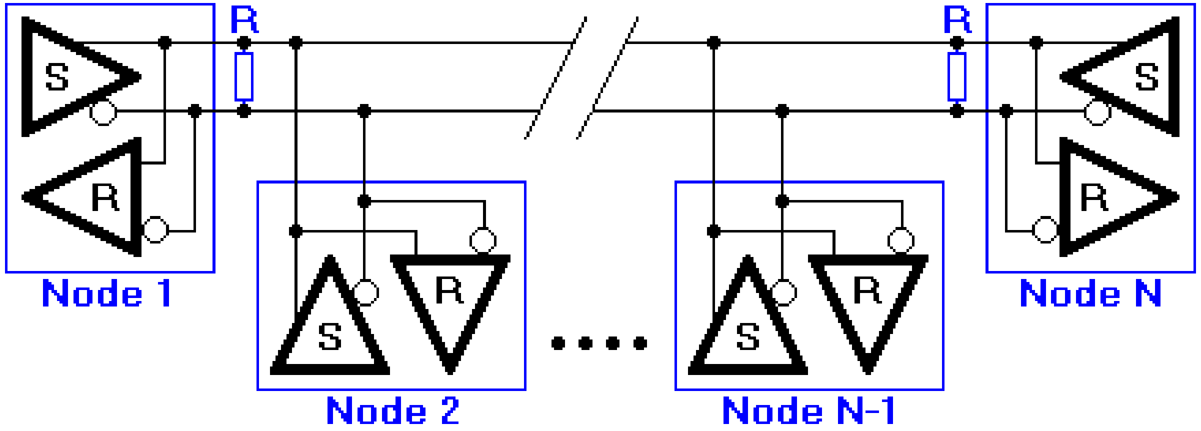
- Simple, bidirectional point-to-point interface based on UART
- Optional control signals, e.g. CTS – Clear To Send
- Ground → common reference level for all signals (single ended)
- Driver circuit allows transmissions up to ~ 10 m
  - Logic '1' -3V to -15V
  - Logic '0' 3V to 15V

[1] standardized by the Electronic Industries Alliance (EIA)

Asynchronous Serial Interface

RS–485 – Differential Transmission Based on UART

- Differential signals
  - Less susceptible to disturbances → longer distances, 100+ Meters
- Transmit and receive share the same lines
  - Multi-point communication
  - Half-duplex
- Industrial automation → E.g. lowest layer of Profibus

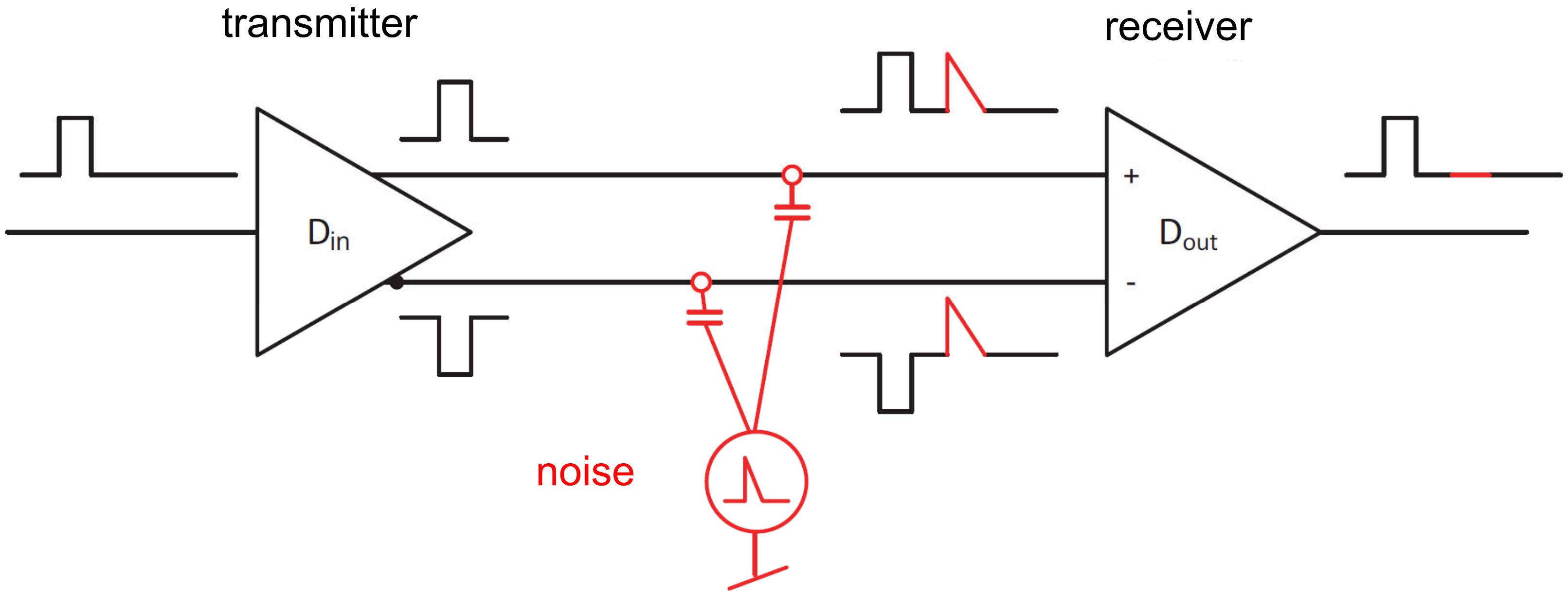


source: http://www.lammertbies.nl/comm/info/RS-485.html

Asynchronous Serial Interface

RS–485 – Differential Transmission

- Transmitter transforms single ended signal into differential signal
- Capacitive coupled noise affects both lines
- Receiver forms the difference of the two signals
  - Noise on the two lines cancels itself to a large extent



SPI Basics

- **SPI – Serial Peripheral Interface**
  - Serial bus for on-board connections
    - Used for short distance communication
  - Connects microcontroller and external devices
    - Sensors, A/D converters, displays, flash memories, codecs
    - IOs, Real Time Clocks (RTC), wireless transceivers...
  - Synchronous
    - Master distributes the clock to slaves
  - Compared to a parallel bus
    - Saves board area
    - Lowers pin count on both chips (TX and RX) → smaller, low-cost
    - Simplifies EMC (Electromagnetic compatibility)

SPI Basics

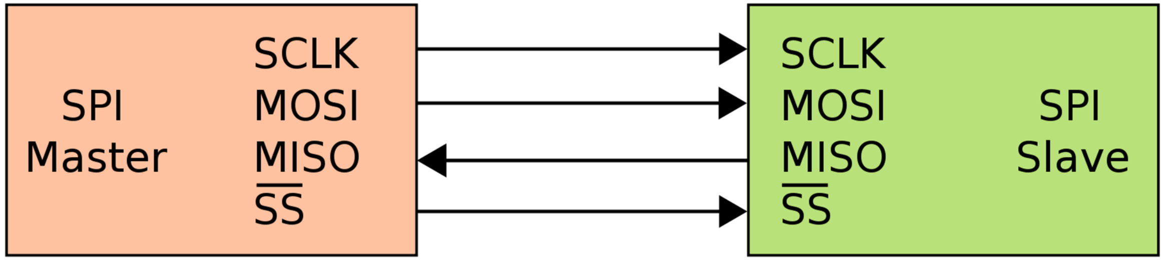
- **SPI → De facto Standard**
  - No official standards organization
  - No legally binding specification
    - Only chip datasheets and application notes
    - Many different variants exist
  - Introduced by Motorola (today NXP) around 1979
  - Also called 4-wire Bus

> A de facto standard is a custom, convention, product, or system that has achieved a dominant position by public acceptance or market forces.

source: Wikipedia

SPI Basics

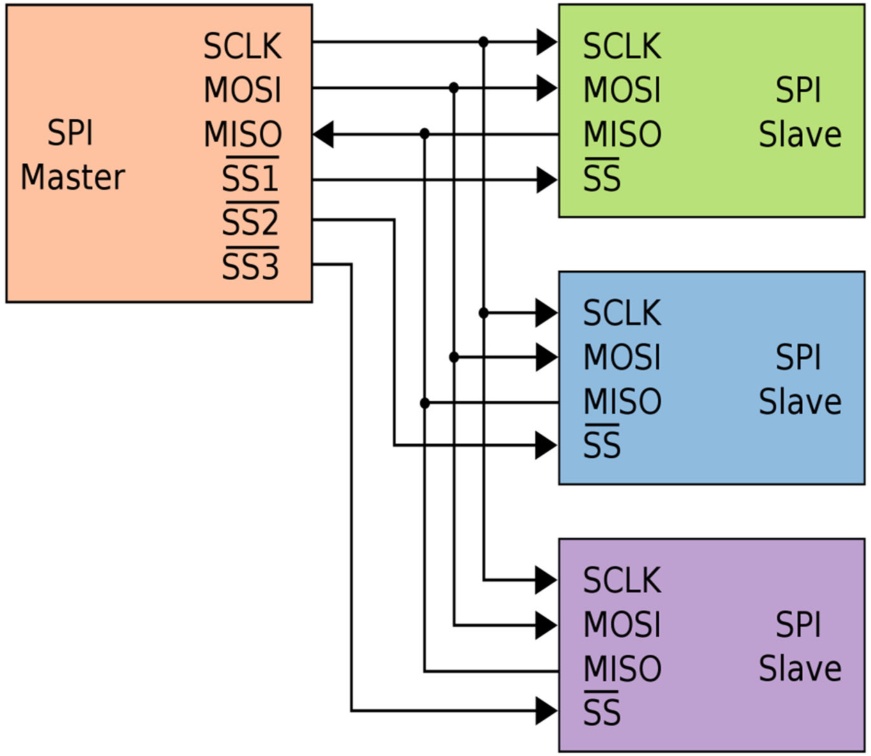
- **Synchronous Serial Data Connection**
  - Full duplex



  - Master (single master)
    - Generates clock (SCLK)
    - Initiates data transfer by setting $\overline{SS}$ = 0 (Slave Select)
  - MOSI – Master Out Slave In
    - Data from master to slave
  - MISO – Master In Slave Out
    - Data from slave to master

SPI Basics

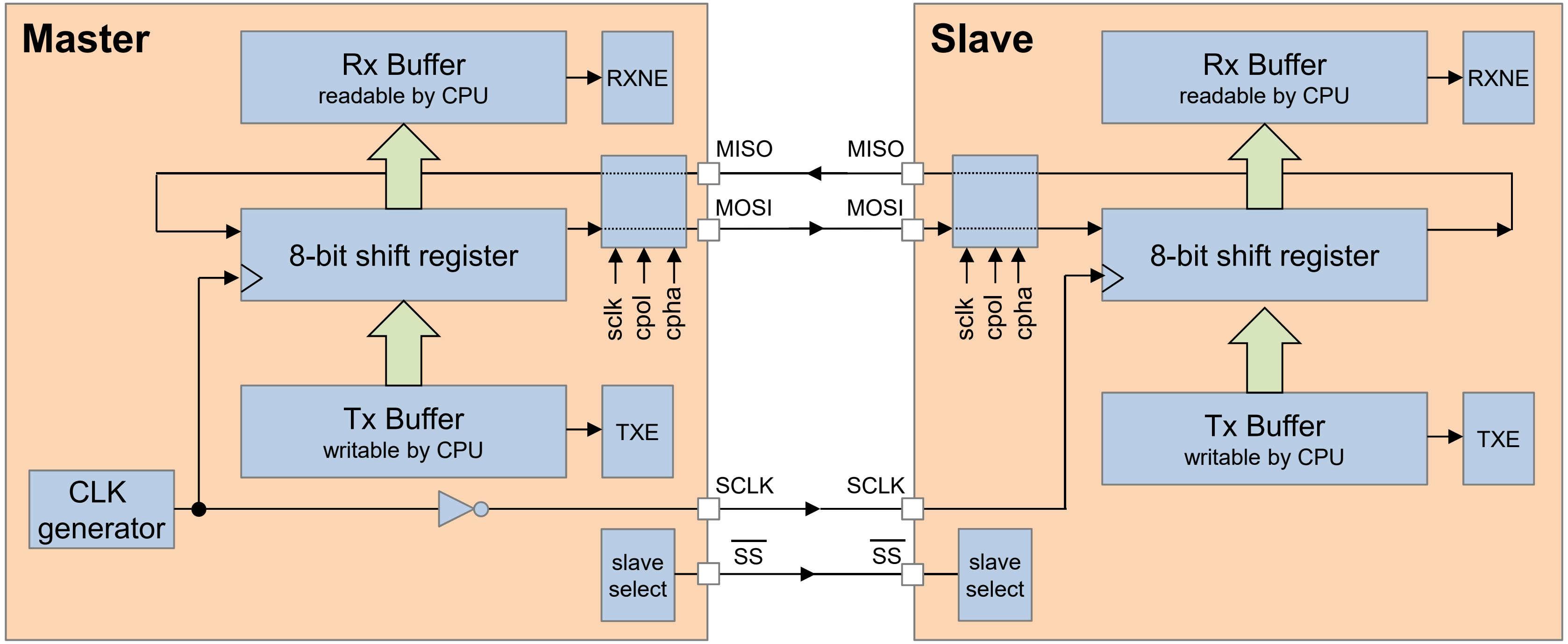
- **Single Master – Multiple Slaves**
  - Master generates a common clock signal for all slaves
  - MOSI
    - From master output to all slave inputs
  - MISO
    - All slave outputs connected to single master input
  - Slaves
    - Individual select $\overline{SS1}$, $\overline{SS2}$, $\overline{SS3}$
    - $\overline{SSx}$ = '1' → Slave output MISOx is tri-state



Source: Wikipedia

SPI Basics

Implementation Using Shift Registers



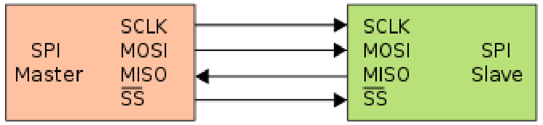
'LSB first' vs. *'MSB first'* is configurable in most microcontrollers.
Slaves are often hard-wired.

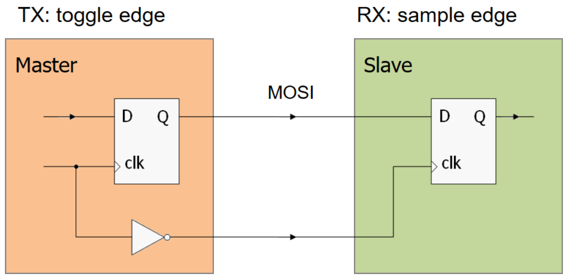
Status bits with Interrupt

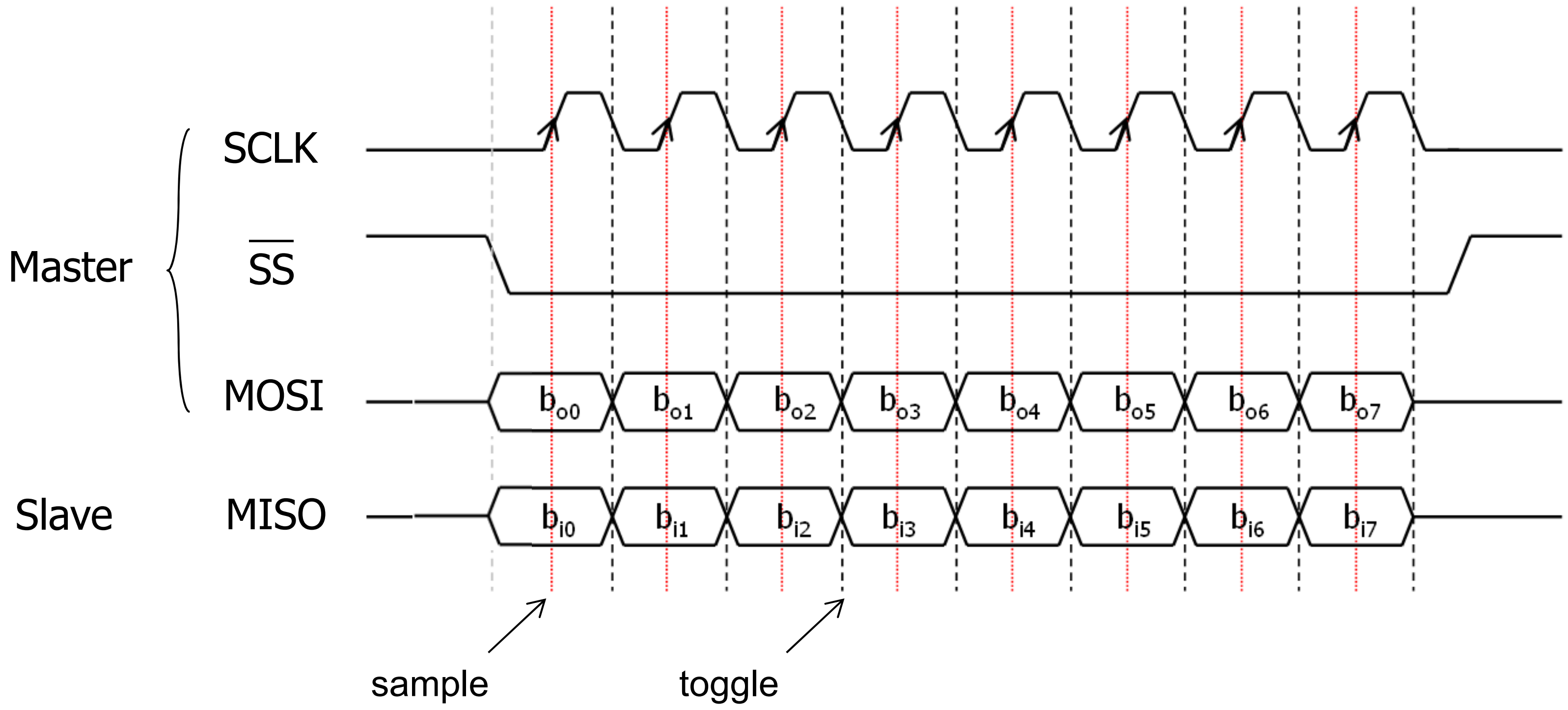
TXE	Tx Buffer Empty
RXNE	Rx Buffer Not Empty

SPI Basics

- **Timing**
  - Toggle output on one clock edge
  - Sample on other clock edge



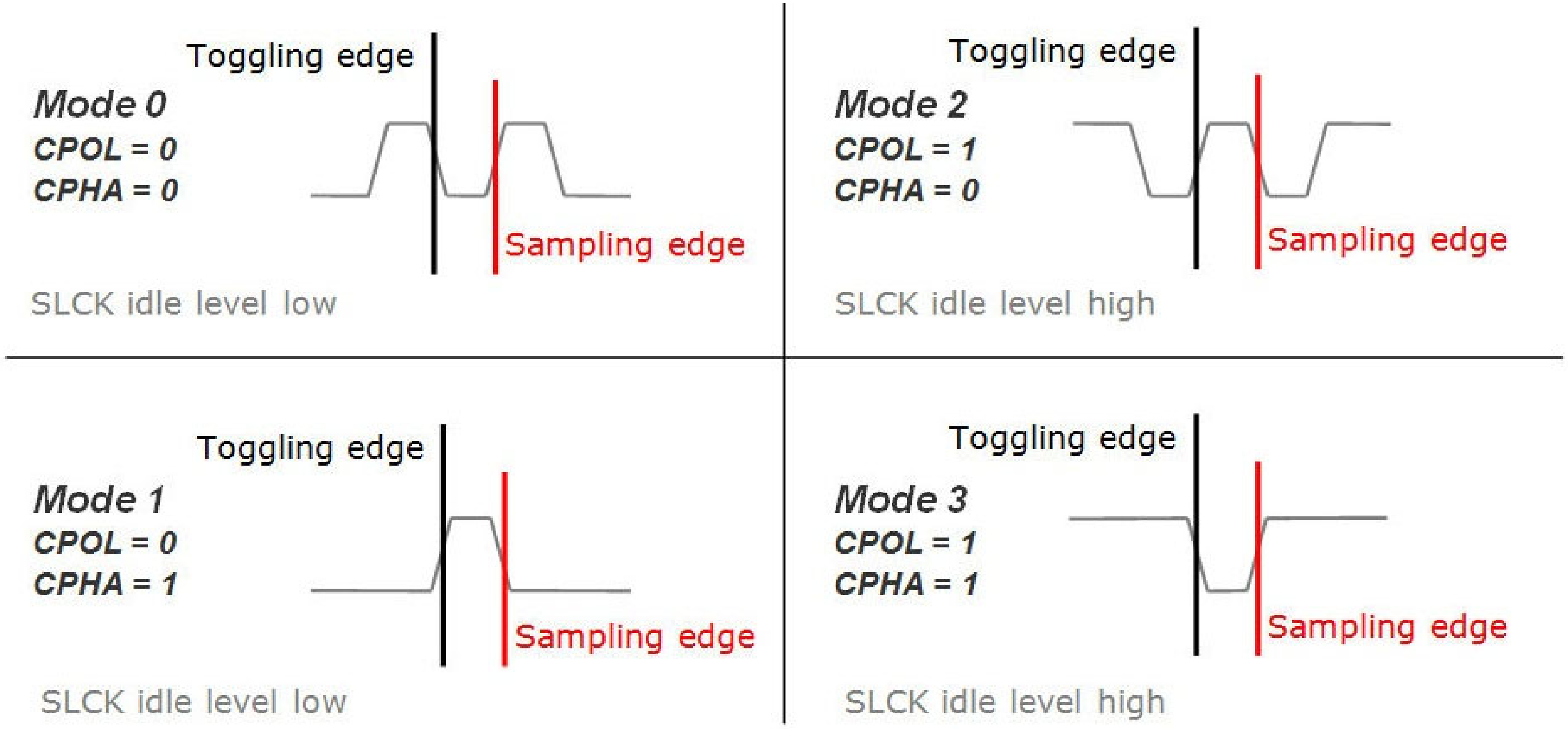



Master
SCLK
$\overline{SS}$
MOSI
Slave
MISO
b_{o0} b_{o1} b_{o2} b_{o3} b_{o4} b_{o5} b_{o6} b_{o7}
b_{i0} b_{i1} b_{i2} b_{i3} b_{i4} b_{i5} b_{i6} b_{i7}
sample
toggle

SPI Modes

- **Clock Polarity and Clock Phase**
  - TX provides data on 'Toggling Edge'
  - RX takes over data with 'Sampling Edge'

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

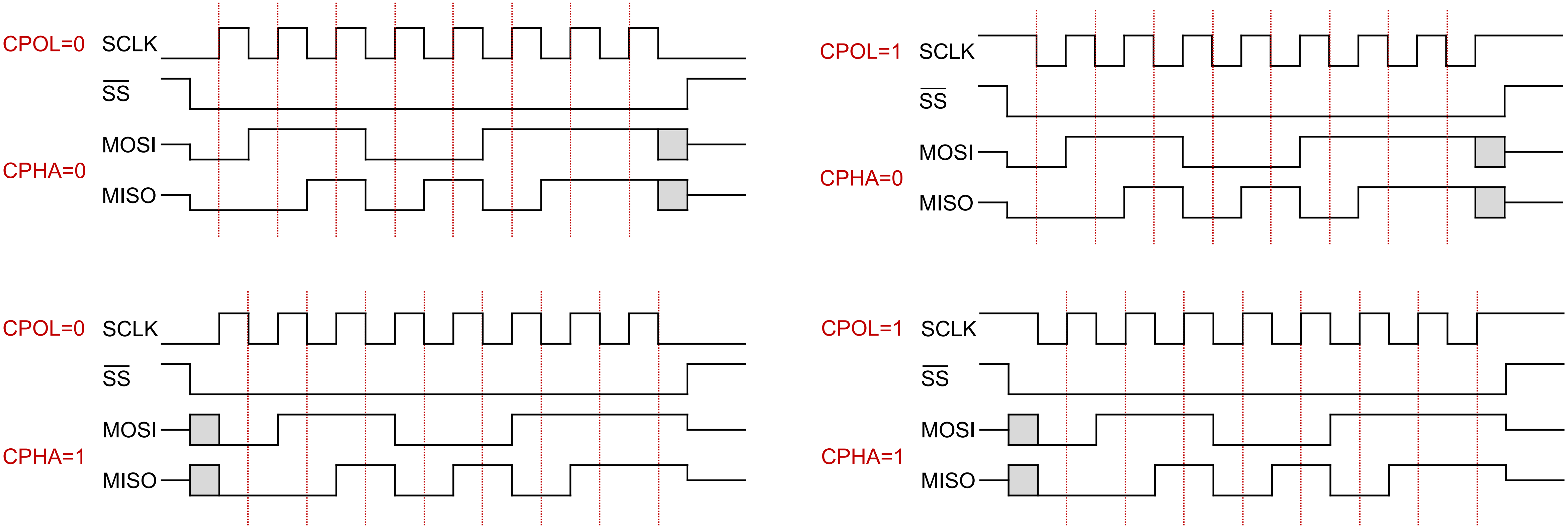


Source: http://www.byteparadigm.com

SPI Modes

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

- **Clock Polarity and Clock Phase**
  - TX provides data on 'Toggling Edge‘ RX takes over data with 'Sampling Edge'

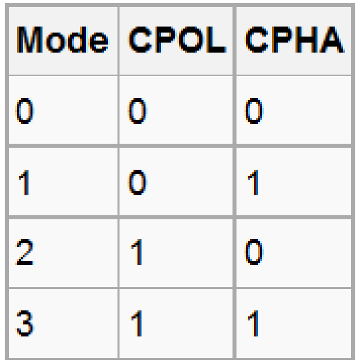


sampling

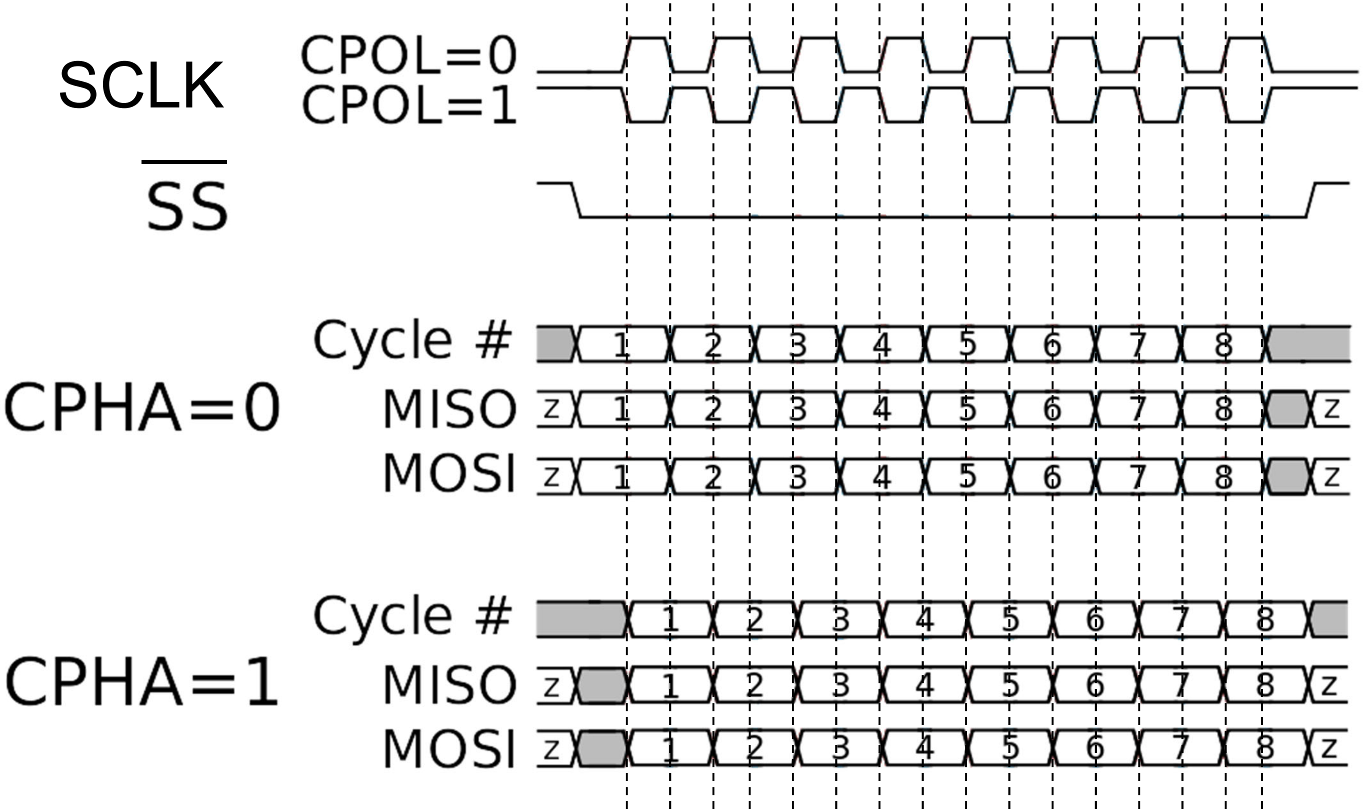
Example with: MOSI = 0x67 MISO = 0x2B

SPI Modes

- **Data and Clock**
  - Summary of all possible combinations



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

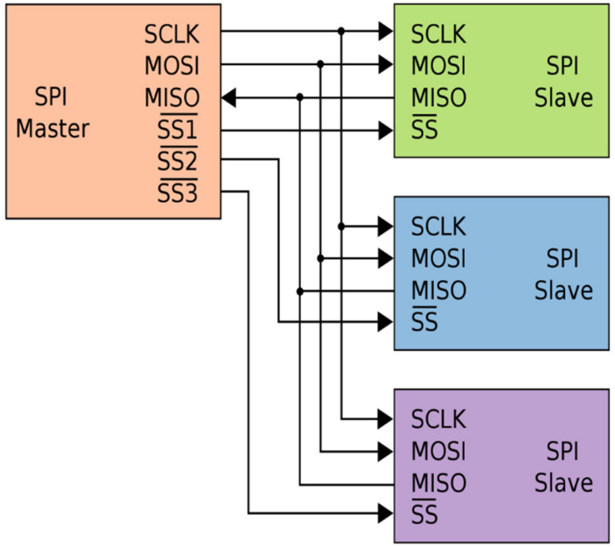


Source: Wikipedia

SPI

Properties

- No defined addressing scheme
  - Use of $\overline{SS}$ instead → KISS
- Transmission without receive acknowledge and error detection
  - Has to be implemented in higher level protocols
- Originally used only for transmission of single bytes
  - $\overline{SS}$ deactivated after each byte
  - Today also used for streams (endless transfers)
- Data rate
  - Highly flexible as clock signal is transmitted
- No flow-control available
  - Master can delay the next clock edge
  - Slave can not influence the data rate
- Susceptible to spikes on clock line

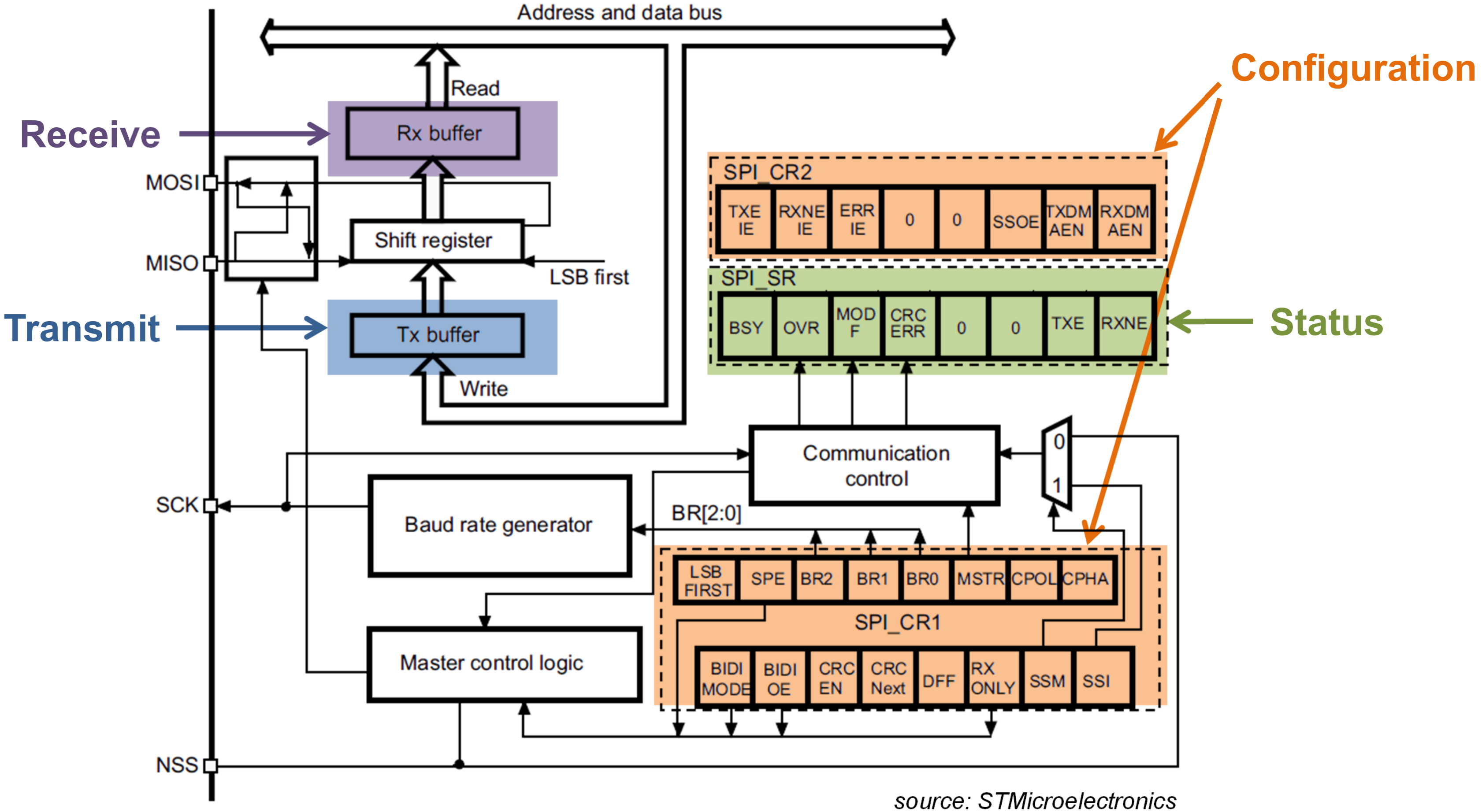


Source: Wikipedia

SPI – STM32F4xxx

Implementation

Receive data	read from SPI_DR register
Transmit data	write to SPI_DR register



SPI_CR2

TXE IE	RXNE IE	ERR IE	0	0	SSOE	TXDM AEN	RXDM AEN

SPI_SR

BSY	OVR	MOD F	CRC ERR	0	0	TXE	RXNE

SPI_CR1

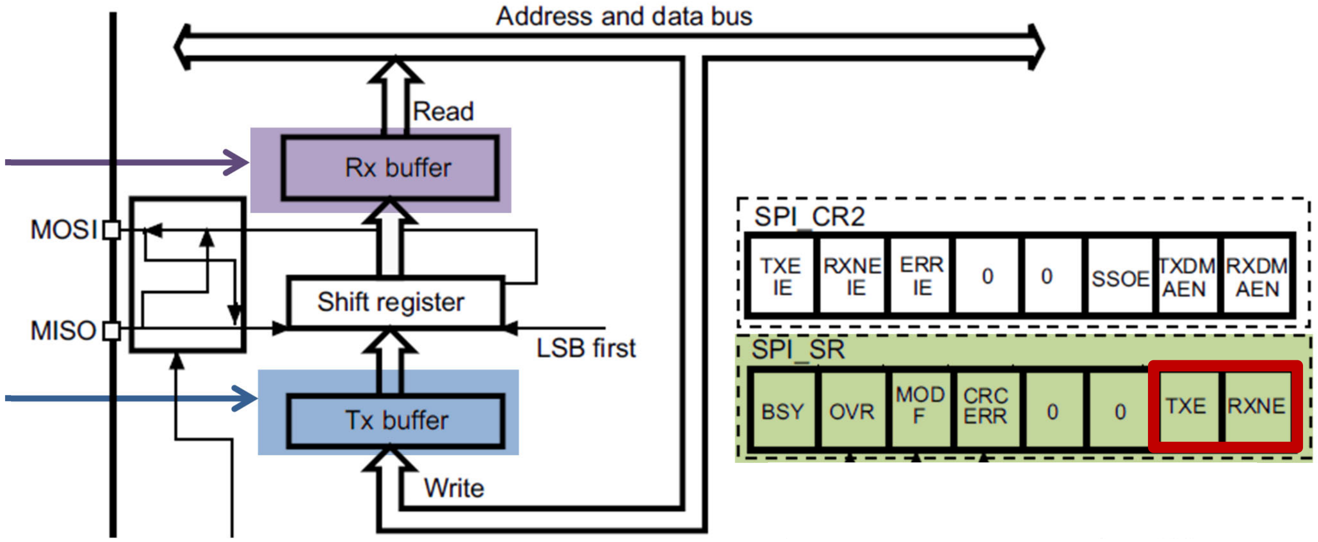
LSB FIRST	SPE	BR2	BR1	BR0	MSTR	CPOL	CPHA

BIDI MODE	BIDI OE	CRC EN	CRC Next	DFF	RX ONLY	SSM	SSI

source: STMicroelectronics

SPI – STM32F4xxx

- **Synchronizing Hardware and Software**
  - When shall software access the shift register?
    - **TXE** **TX Buffer Empty** **Software can write next TX Byte to register SPI_DR**
    - **RXNE** **RX Buffer Not Empty** **A byte has been received. Software can read it from SPI_DR**



Receive read SPI_DR

Transmit write SPI_DR

SPI_CR2

TXE IE	RXNE IE	ERR IE	0	0	SSOE	TXDM AEN	RXDM AEN

SPI_SR

BSY	OVR	MOD F	CRC ERR	0	0	TXE	RXNE

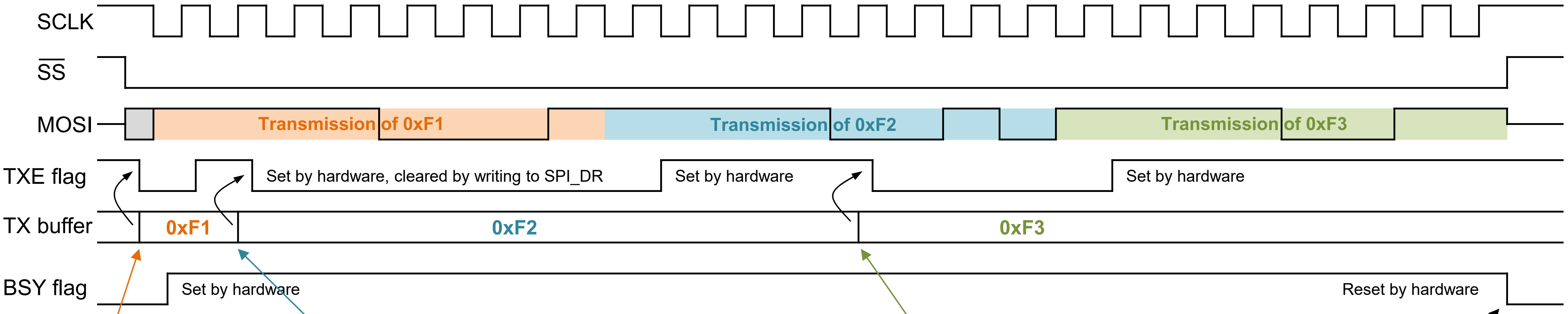
source: STMicroelectronics

SPI – STM32F4xxx

Transmitting in Software

- Example: SW wants to transmit bytes 0xF1, 0xF2, 0xF3

Example with
CPOL=1, CPHA=1, MSB first



Software writes 0xF1 into SPI_DR

Software waits until TXE=1 and writes 0xF2 into SPI_DR

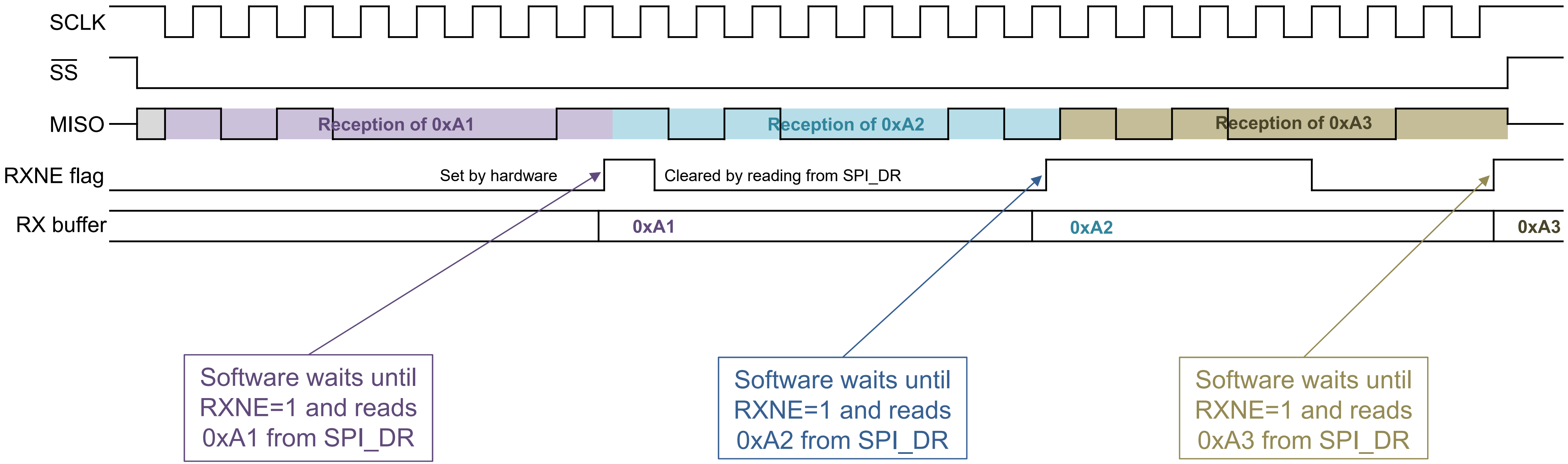
Software waits until TXE=1 and writes 0xF3 into SPI_DR

BSY flag going low indicates to software that transmission has completed

SPI – STM32F4xxx

Example with
CPOL=1, CPHA=1, MSB first

- **Receiving in Software**
  - Example: SW receives bytes 0xA1, 0xA2, 0xA3



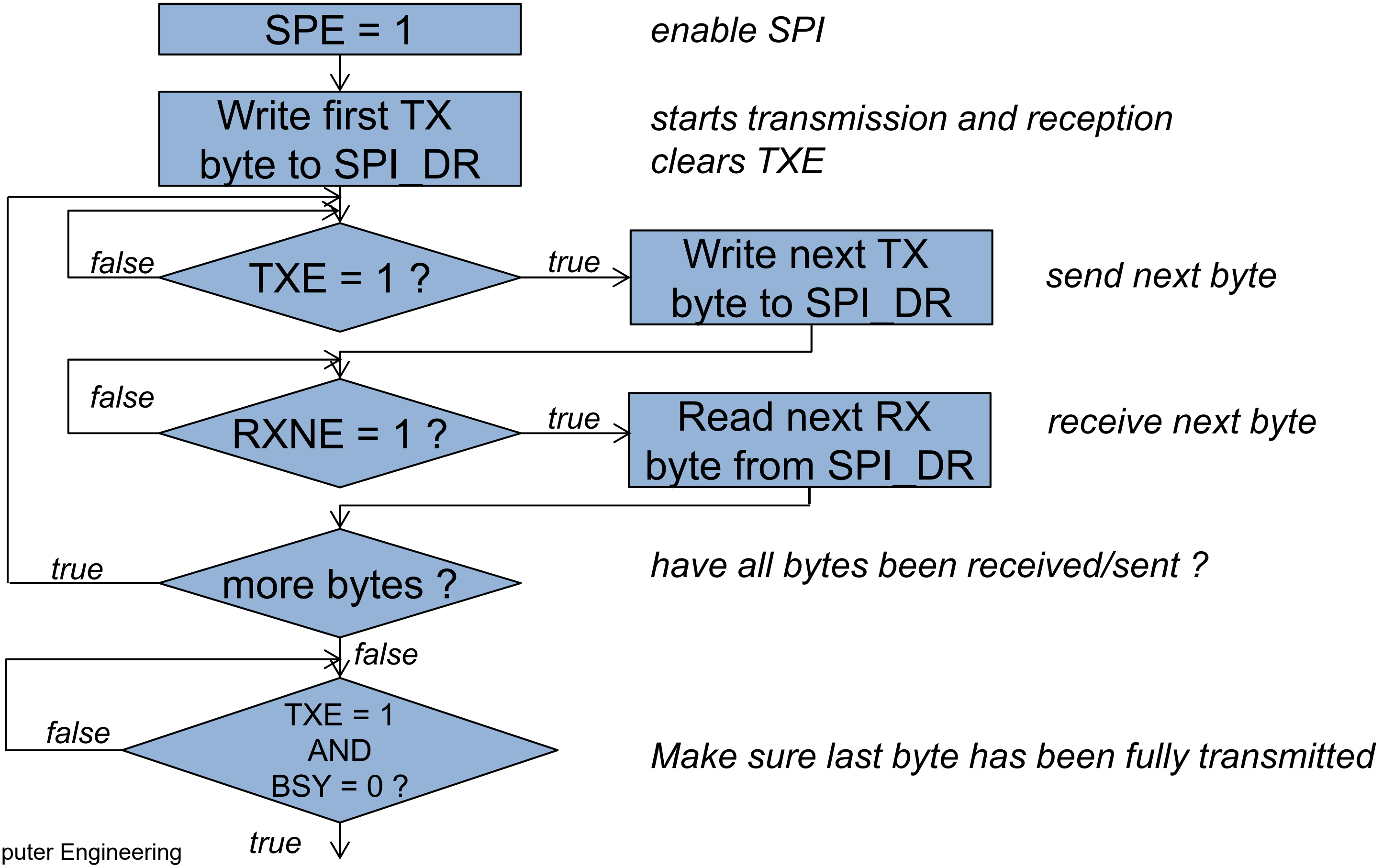
Software waits until RXNE=1 and reads 0xA1 from SPI_DR

Software waits until RXNE=1 and reads 0xA2 from SPI_DR

Software waits until RXNE=1 and reads 0xA3 from SPI_DR

SPI – STM32F4xxx

- **Software: Simultaneously Handling Data Transmission and Reception**
  - Full duplex: Check TXE and RXNE bits



enable SPI

starts transmission and reception
clears TXE

send next byte

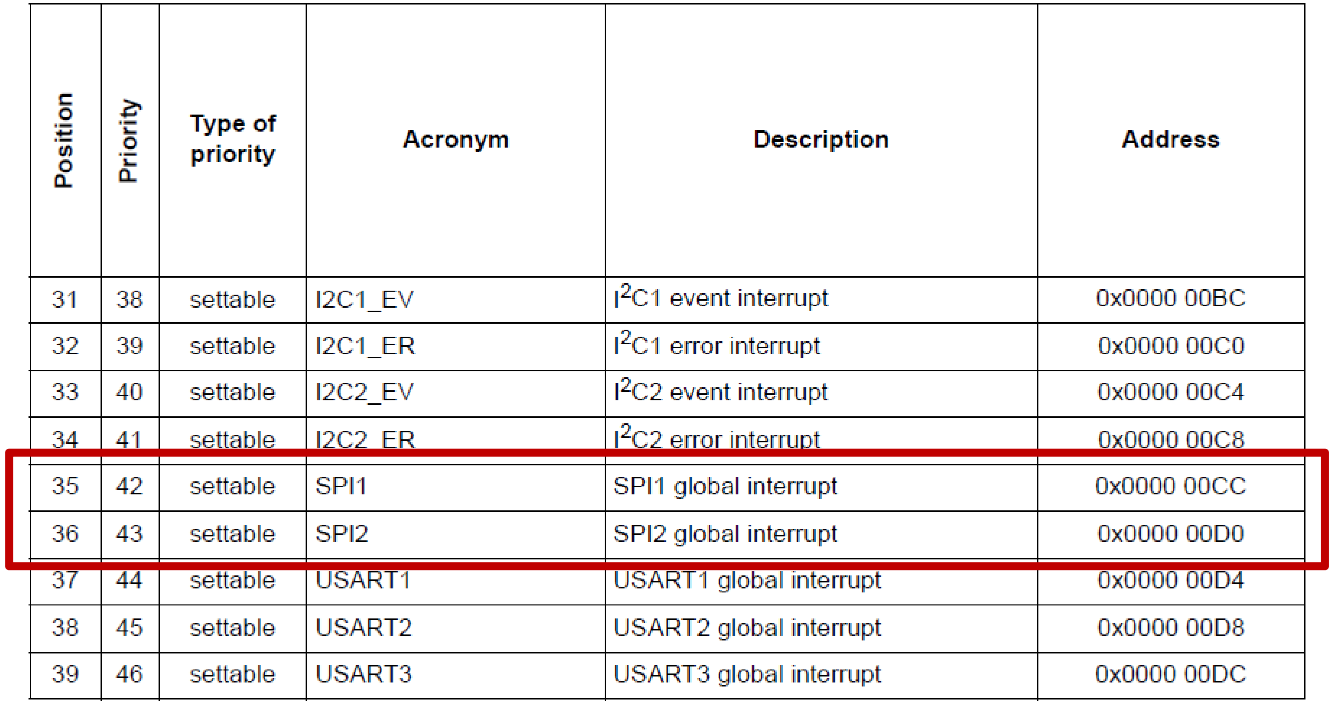
receive next byte

have all bytes been received/sent ?

Make sure last byte has been fully transmitted

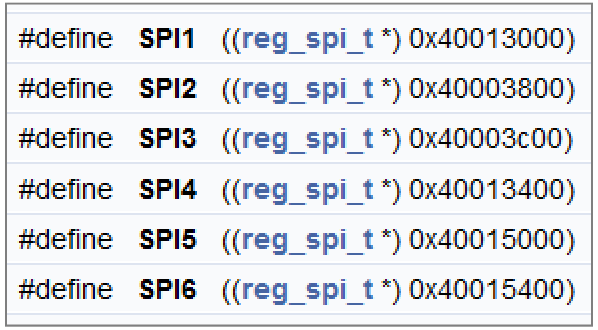
SPI – STM32F4xxx

- **Interrupts**
  - Interrupts can be used instead of polling for TXE and RXNE bits



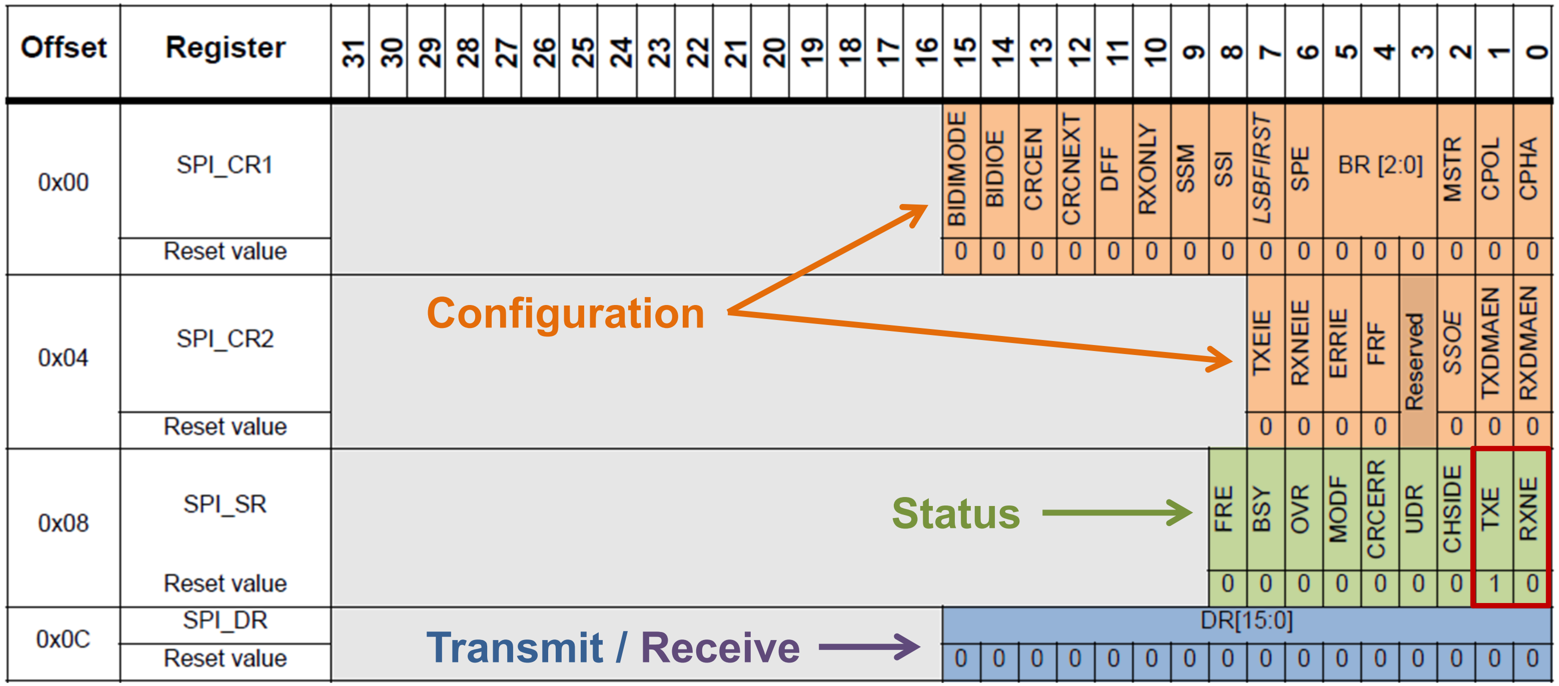
Position	Priority	Type of priority	Acronym	Description	Address
31	38	settable	I2C1_EV	I^2C1 event interrupt	0x0000 00BC
32	39	settable	I2C1_ER	I^2C1 error interrupt	0x0000 00C0
33	40	settable	I2C2_EV	I^2C2 event interrupt	0x0000 00C4
34	41	settable	I2C2_ER	I^2C2 error interrupt	0x0000 00C8
35	42	settable	SPI1	SPI1 global interrupt	0x0000 00CC
36	43	settable	SPI2	SPI2 global interrupt	0x0000 00D0
37	44	settable	USART1	USART1 global interrupt	0x0000 00D4
38	45	settable	USART2	USART2 global interrupt	0x0000 00D8
39	46	settable	USART3	USART3 global interrupt	0x0000 00DC

SPI – STM32F4xxx



```
#define  SPI1  ((reg_spi_t *) 0x40013000)
#define  SPI2  ((reg_spi_t *) 0x40003800)
#define  SPI3  ((reg_spi_t *) 0x40003c00)
#define  SPI4  ((reg_spi_t *) 0x40013400)
#define  SPI5  ((reg_spi_t *) 0x40015000)
#define  SPI6  ((reg_spi_t *) 0x40015400)
```

- **SPI Registers**
  - Total of 6 SPI blocks
    - Set of registers for each of them



Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	SPI_CR1																	BIDIMODE	BIDIOE	CRCEN	CRCNEXT	DFF	RXONLY	SSM	SSI	*LSBFIRST*	SPE	BR [2:0]			MSTR	CPOL	CPHA
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	SPI_CR2																									TXEIE	RXNEIE	ERRIE	FRF	Reserved	SSOE	TXDMAEN	RXDMAEN
	Reset value																									0	0	0	0		0	0	0
0x08	SPI_SR																								FRE	BSY	OVR	MODF	CRCERR	UDR	CHSIDE	TXE	RXNE
	Reset value																								0	0	0	0	0	0	0	1	0
0x0C	SPI_DR																	DR[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

source: STM32F42xxx Reference Manual

SPI – STM32F4xxx

Register Bits

SPI_CR1	SPI control register 1
BIDIMODE	Bidirectional data mode enable
BIDIOE	Output enable in bidir mode
CRCEN	Hardware CRC calculation enable
CRCNEXT	CRC transfer next
DFF	Data frame format (8-bit vs. 16-bit)
RXONLY	Receive only
SSM	Software slave management
SSI	Internal slave select
LSBFIRST	Frame format (bit order)
SPE	SPI enable
BR[2:0]	Baud rate control
MSTR	Master selection (master vs. slave)
CPOL	Clock polarity
CPHA	Clock phase

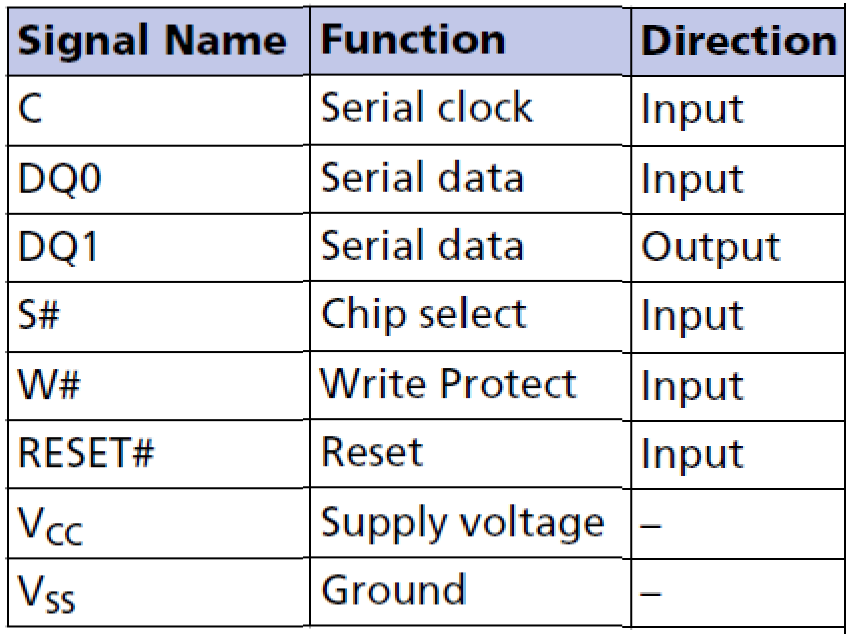
SPI_DR	SPI data register
DR[15:0]	Data register

SPI_CR2	SPI control register 2
TXEIE	Tx buffer empty interrupt enable
RXNEIE	RX buffer not empty interrupt enable
ERRIE	Error interrupt enable
FRF	Frame format (Motorola vs. TI mode)
SSOE	SS output enable
TXDMAEN	Txbuffer DMA enable

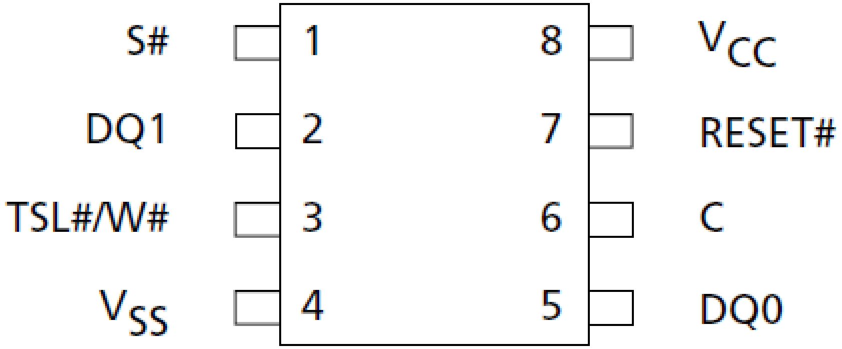
SPI_SR	SPI status register
FRE	Frame format error
BSY	Busy flag (Txbuffer not empty)
OVR	Overrun flag
MODF	Mode fault
CRCERR	CRC error flag
UDR	Underrun flag
CHSIDE	Channel side (not used for SPI)
TXE	Transmit buffer empty
RXNE	Receive buffer not empty

SPI – Flash Devices

- **Save Board Area**
  - E.g. Micron M25PE40 Serial Flash Memory
    - 4 Mbit NOR flash
    - 6 x 5 mm package size
    - SCLK: up to 75 MHz



Signal Name	Function	Direction
C	Serial clock	Input
DQ0	Serial data	Input
DQ1	Serial data	Output
S#	Chip select	Input
W#	Write Protect	Input
RESET#	Reset	Input
V_{CC}	Supply voltage	–
V_{SS}	Ground	–

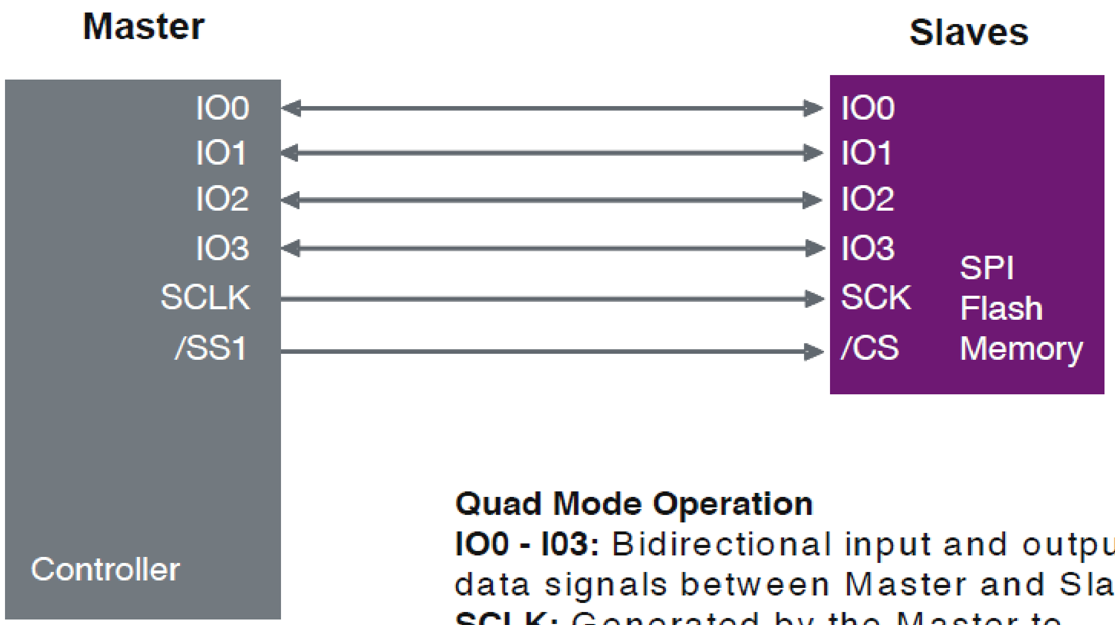


Source: Micron

SPI – Flash Devices

- **Some Vendors Use More Than One Data Line**
  - Example: Flash using Quad I/O for Higher Bandwidth

FIGURE 1: QUAD I/O SERIAL INTERCONNECTION



Quad Mode Operation
IO0 - I03: Bidirectional input and output data signals between Master and Slaves
SCLK: Generated by the Master to synchronize data transfers

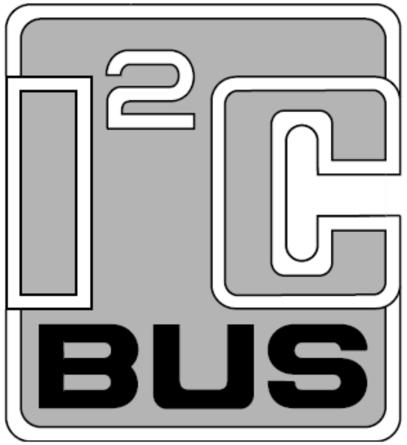
Source: Micron

I2C Bus

- **I²C – Inter-Integrated Circuit, pronounced I-squared-C**
  - Bidirectional 2-wire
  - Defined by Philips Semiconductors, now NXP
  - First release in 1982

Reference Documentation

UM10204 – I2C-bus specification and user manual
Rev. 6 — 4 April 2014

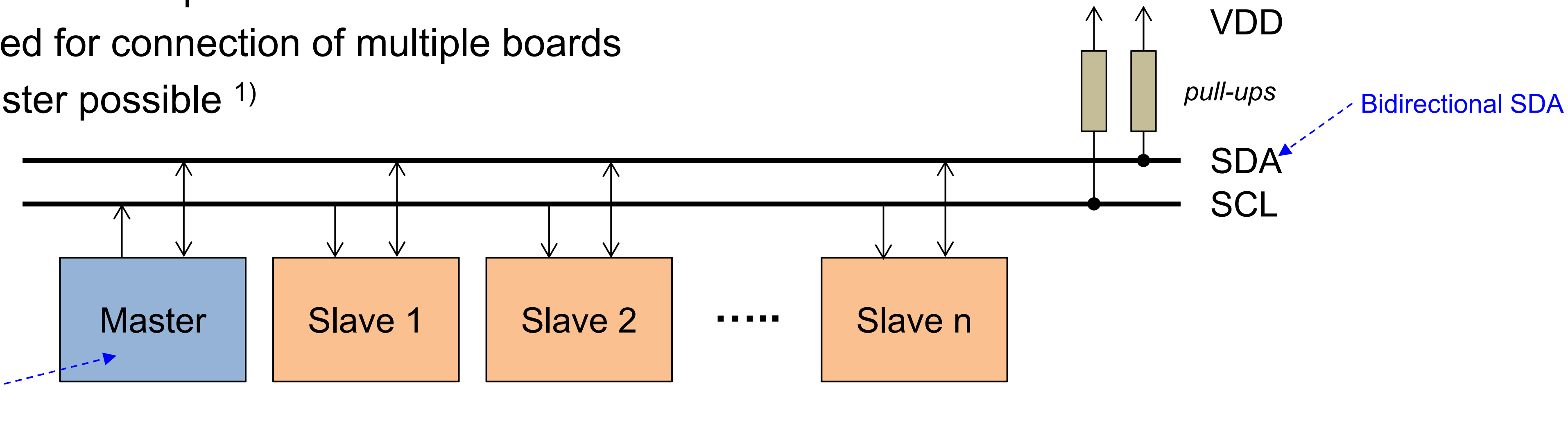


source: NXP

I2C Bus

Overview

- 2-wire bus Clock → SCL Data → SDA
- Synchronous, half-duplex
- Each device on bus addressable through unique address
  - NXP assigns manufacturer IDs
- 8-bit oriented data transfers
- Different bit rates up to 5 Mbit/s
- Well suited for connection of multiple boards
- Multi-master possible [1)]

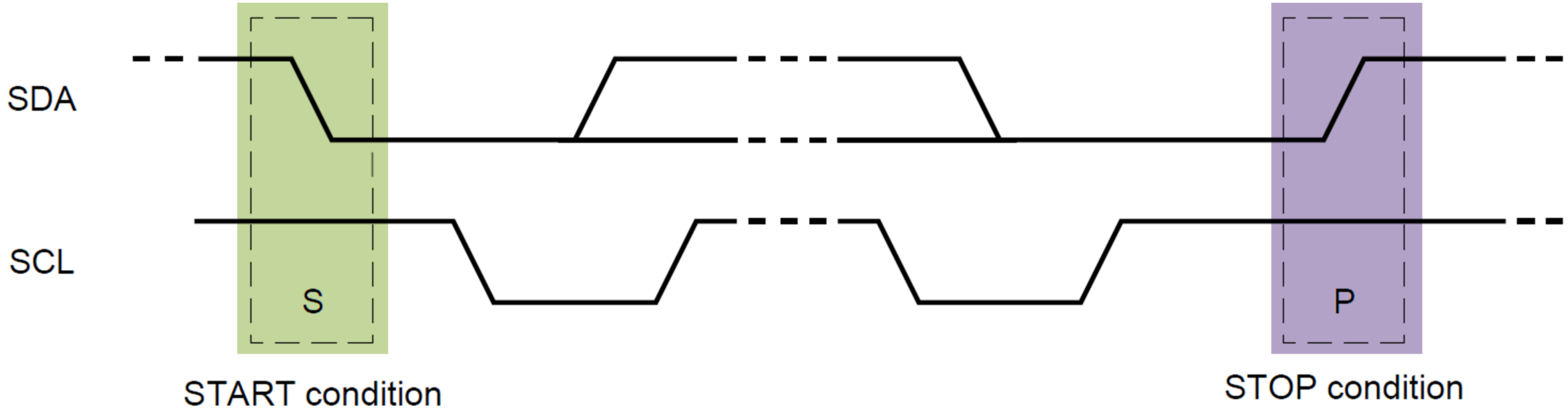


1) multi-master not covered in this course

I2C Bus

Operation

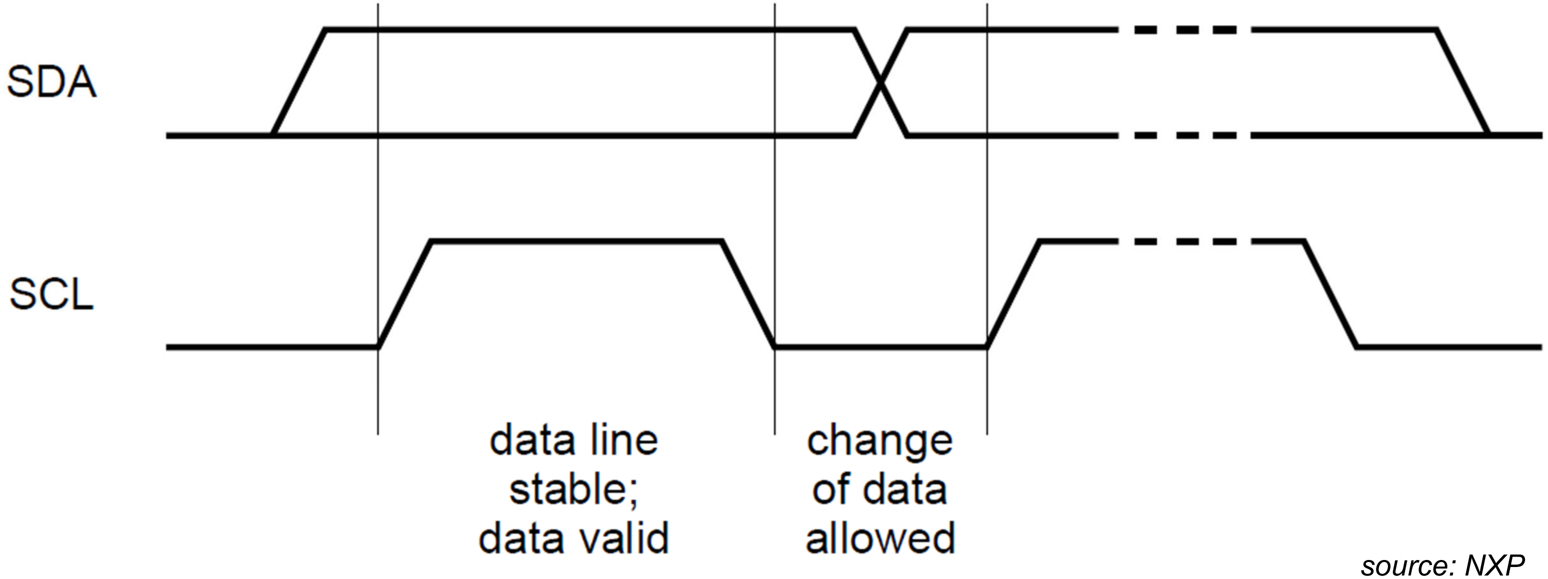
- Master drives clock line (SCL)
- Master initiates transaction through START condition
  - Falling edge on SDA when SCL high
- Master terminates transaction through STOP condition
  - Rising edge on SDA when SCL high



source: NXP

I2C Bus

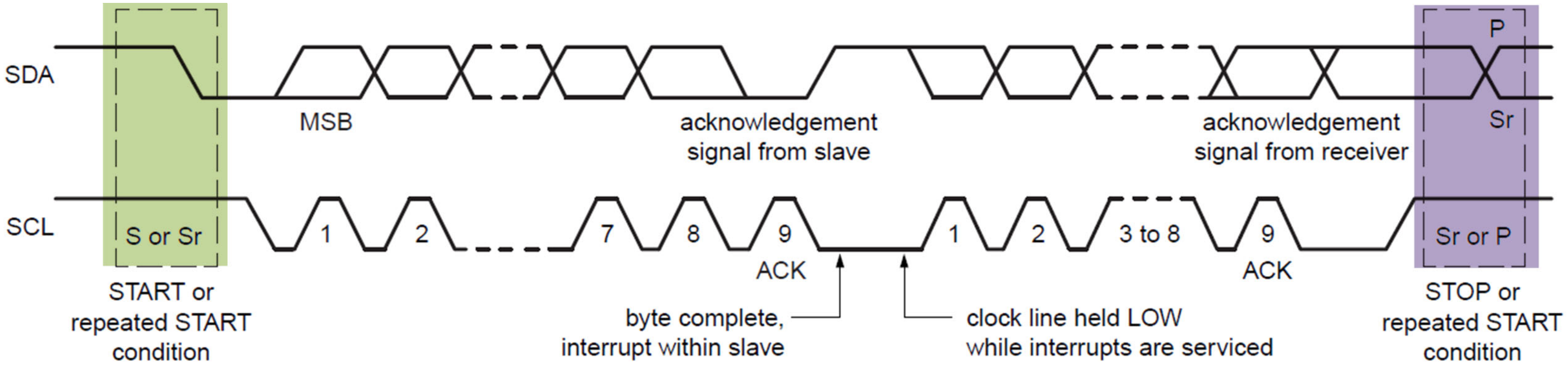
- **Driving Data on SDA**
  - Data driven onto SDA by master or by addressed slave
    - Depending on transaction (read/write) and point in time
    - Change of data only allowed when SCL is low
    - Allows detection of START and STOP condition



source: NXP

I2C Bus

- **Data Transfer on I2C**
  - 8-bit oriented transfers
  - Bit 9: Receiver acknowledges by driving SDA low
  - Master defines number of 8-bit transfers (STOP)



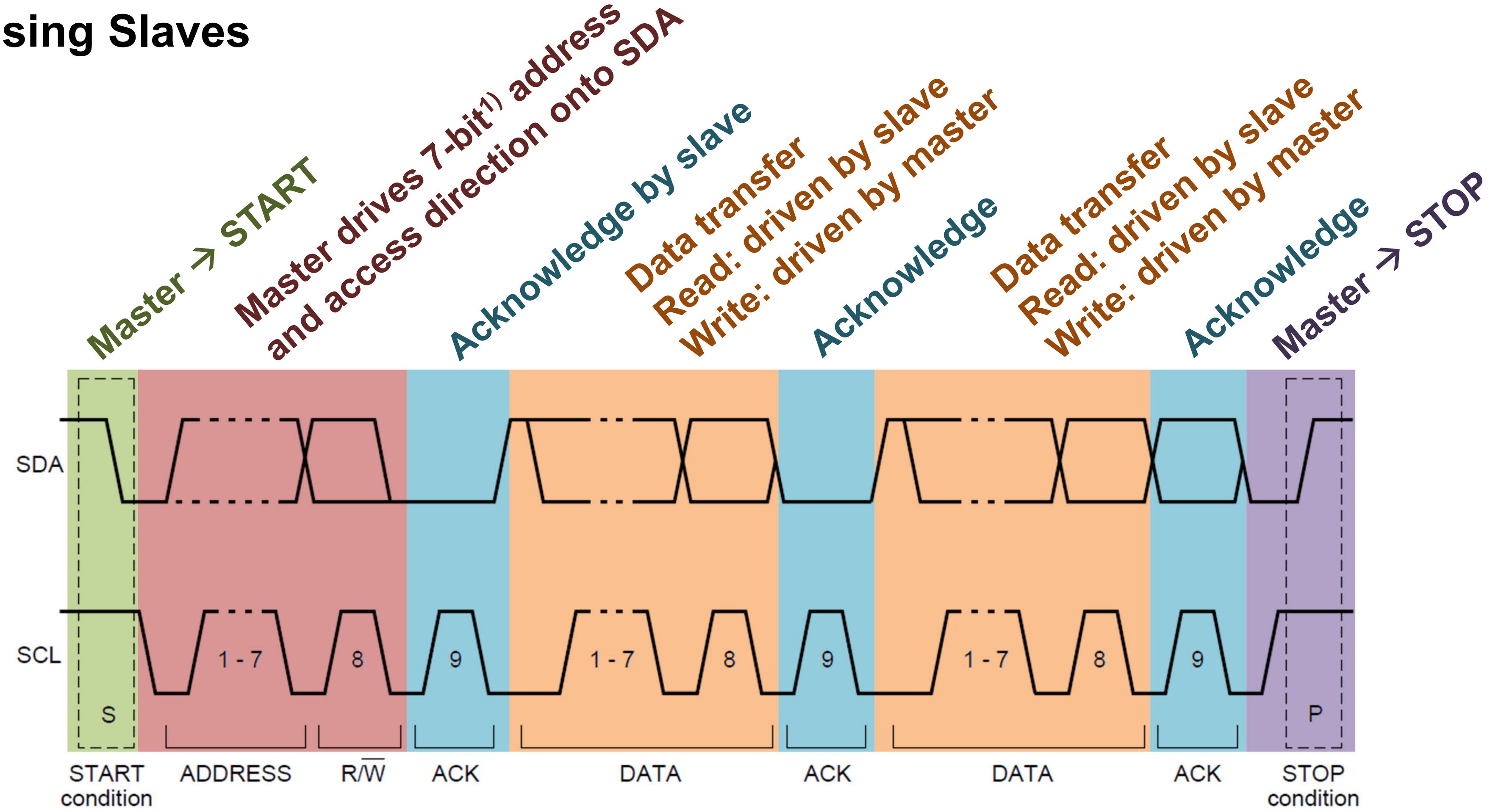
source: NXP

Bit numbering starts with 1; not with 0

'ACK' is active low

I2C Bus

- **Addressing Slaves**

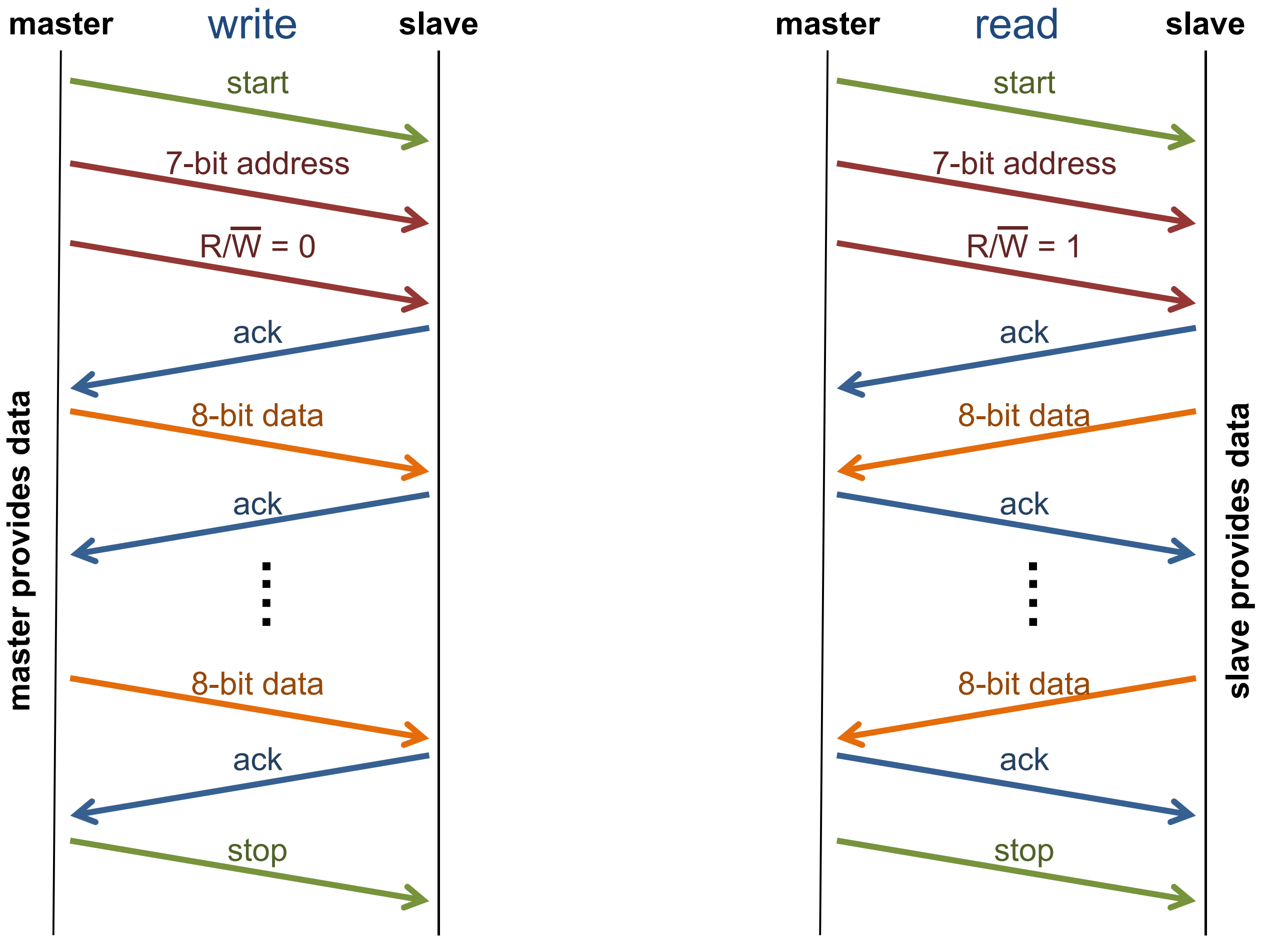


source: NXP

1) 10-bit address scheme available as option

I2C Bus

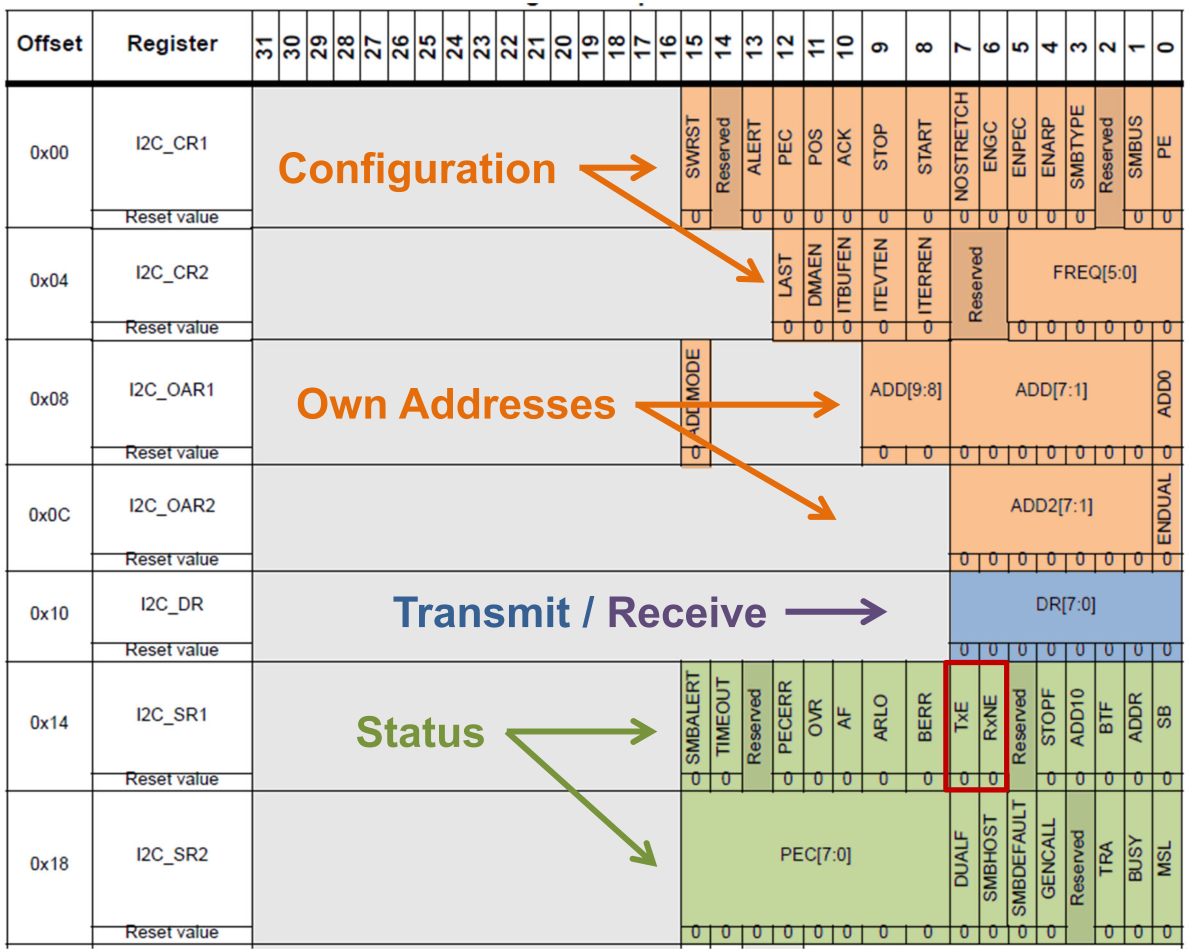
- **Accesses**



I2C – STM32F4xxx

I²C Registers

```
#define  I2C1  ((reg_i2c_t *) 0x40005400)
#define  I2C2  ((reg_i2c_t *) 0x40005800)
#define  I2C3  ((reg_i2c_t *) 0x40005c00)
```



Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	I2C_CR1																	SWRST	Reserved	ALERT	PEC	POS	ACK	STOP	START	NOSTRETCH	ENGC	ENPEC	ENARP	SMBTYPE	Reserved	SMBUS	PE
	Reset value																	0		0	0	0	0	0	0	0	0	0	0	0		0	0
0x04	I2C_CR2																				LAST	DMAEN	ITBUFEN	ITEVTEN	ITERREN	Reserved		FREQ[5:0]					
	Reset value																				0	0	0	0	0			0	0	0	0	0	0
0x08	I2C_OAR1																	ADDMODE						ADD[9:8]		ADD[7:1]							ADD0
	Reset value																	0						0	0	0	0	0	0	0	0	0	0
0x0C	I2C_OAR2																									ADD2[7:1]							ENDUAL
	Reset value																									0	0	0	0	0	0	0	0
0x10	I2C_DR																									DR[7:0]							
	Reset value																									0	0	0	0	0	0	0	0
0x14	I2C_SR1																	SMBALERT	TIMEOUT	Reserved	PECERR	OVR	AF	ARLO	BERR	TxE	RxNE	Reserved	STOPF	ADD10	BTF	ADDR	SB
	Reset value																	0	0		0	0	0	0	0	0	0		0	0	0	0	0
0x18	I2C_SR2																	PEC[7:0]								DUALF	SMBHOST	SMBDEFAULT	GENCALL	Reserved	TRA	BUSY	MSL
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0

Configuration

Own Addresses

Transmit / Receive

Status

I2C – STM32F4xxx

Register Bits

I2C_CR1	I^{2}C control register 1
SWRST	Software reset
ALERT	SMBus alert
PEC	Packet error checking
POS	Acknowledge / PEC Position
ACK	Acknowledge enable
STOP	Stop generation
START	Start generation
NOSTRETCH	Clock stretching disable (slave)
ENGC	General call enable
ENPEC	PEC enable
ENARP	ARP enable
SMBTYPE	SMBus type (device vs. host)
SMBUS	SMBus mode (I2C vs. SMBus)
PE	Peripheral enable

I2C_CR2	I^{2}C control register 2
LAST	DMA last transfer
DMAEN	DMA requests enable
ITBUFEN	Buffer interrupt enable
ITEVEN	Event interrupt enable
ITERREN	Error interrupt enable
FREQ[5:0]	Peripheral clock frequency

I2C_DR	I^{2}C data register 2
DR[7:0]	8-bit data register

I2C_OAR1	I^{2}C own address register 1
ADDMODE	Addressing mode (7-bit vs. 10-bit)
ADD[9:8]	Interface address
ADD[7:1]	Interface address
ADD0	Interface address

I2C_OAR2	I^{2}C own address register 2
ADD2[7:1]	Interface address in dual adr. mode
ENDUAL	Dual addressing mode enable

I2C_SR1	I^{2}C status register 1
SMBALERT	SMBus alert
TIMEOUT	Timout or Tlow error
PECERR	PEC error in reception
OVR	Overrun/Underrun
AF	Acknowledge failure
ARL0	Arbitration lost (master)
BERR	Bus error
TxE	Data register empty
RxNE	Data register not empty
STOPF	Stop detected (slave)
ADD10	10-bit header sent (master)
BTF	Byte transfer finished
ADDR	ADDR sent (master) / matched (slave)
SB	Start bit (master)

I2C_SR2	I^{2}C status register 2
PEC[7:0]	Packet error checking register
DUALF	Dual flag (slave)
SMBHOST	SMBus host header (slave)
SMBDEFAULT	SMBus device default address (slave)
GENCALL	General call address (slave)
TRA	Transmitter/receiver (R/W bit)
BUSY	Bus busy (communication ongoing on bus)
MSL	Master/slave

Comparison

UART	SPI	I2C
serial ports (RS-232)	4-wire bus	2-wire bus
TX, RX opt. control signals	*MOSI, MISO, SCLK, SS*	*SCL, SDA*
point-to-point	point-to-multipoint	(multi-) point-to-multi-point
full-duplex	full-duplex	half-duplex
asynchronous	synchronous	synchronous
only higher layer addressing	slave selection through $\overline{SS}$ signal	7/10-bit slave address
parity bit possible	no error detection	no error detection
chip-to-chip, PC terminal program	chip-to-chip, on-board connections	chip-to-chip, board-to-board connections

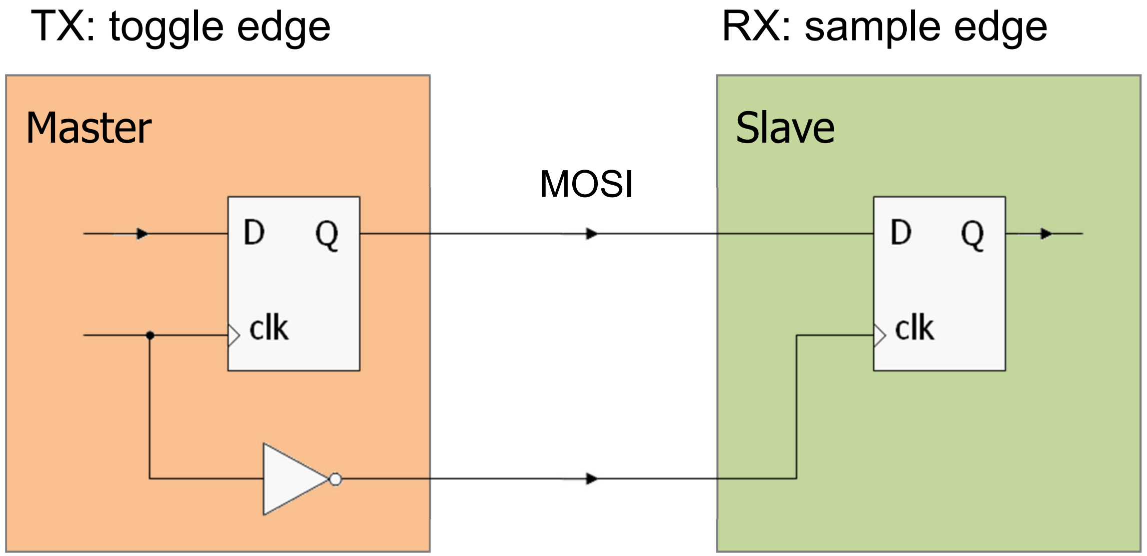
The three interfaces provide the lowest layer of communication and require higher level protocols to provide and interpret the transferred data.

Conclusions

- **Asynchronous Serial Interface**
  - Transmitter and receiver use diverging clocks
  - Synchronization using start/stop bits → overhead
  - Longer connections require line drivers → RS-232/RS-485
- **SPI**
  - Master/Slave
  - Synchronous full-duplex transmission (MOSI, MISO)
  - Selection of device through Slave Select ($\overline{SS}$)
  - No acknowledge, no error detection
  - Four modes → clock polarity and clock phase
- **I2C**
  - Synchronous half-duplex transmission (SCL, SDA)
  - 7-bit slave addresses

SPI Basics

- **Clocking Circuit**


TX: toggle edge
RX: sample edge
Master
Slave
MOSI
D
Q
clk
D
Q
clk