# Lecture 16: STM32 Serial Communication III

Seyed-Hosein Attarzadeh-Niaki



## Review

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

# Outline

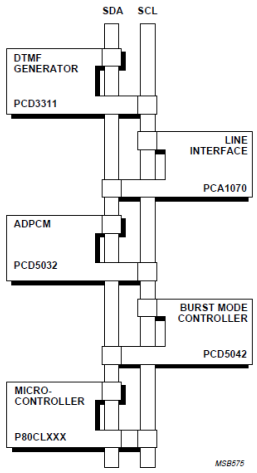
- Inter-integrated circuit ($I^2C$)
  - $I^2C$ bus protocol
  - $I^2C$ in STM32
  - $I^2C$ examples



# INTER-INTEGRATED CIRCUIT ($I^2C$)

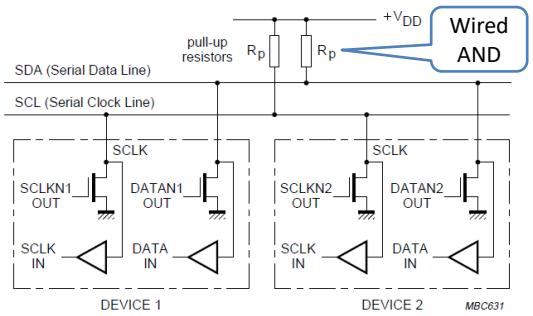
# I²C Bus Overview

- "Inter-Integrated Circuit" bus
- Originally by Philips
- Multiple devices connected by a shared serial bus
- Synchronous, multi-master/slave
- Bus is typically controlled by master device, slaves respond when addressed
- AKA 2-wire interface (TWI)
- I²C bus has two signal lines
  - **SCL**: Serial clock
  - **SDA**: Serial data
- Full details available in "The I²C-bus Specification"





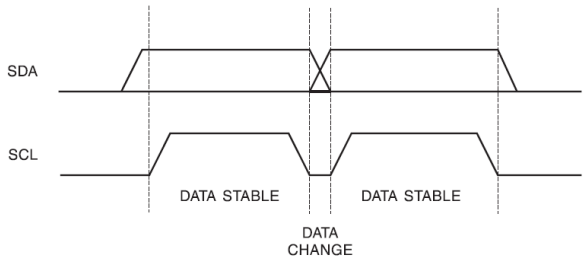

# I²C Bus Connections




- Resistors pull up lines to VDD
- Open-drain transistors pull lines down to ground
- Both master and slave can receive or transmit data
- Master generates SCL clock signal
  - Can range up to 400 kHz, 1 MHz, or more

# Bit Format

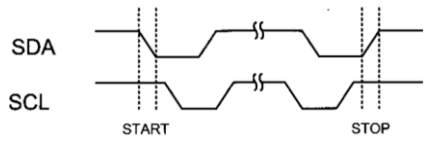
- $I^2C$ is a synchronous serial protocol
  - Each data bit transferred on the SDA line is synchronized by a high-to-low pulse of clock on the SCL line.
- According to $I^2C$ protocol
  - Data line cannot change when the clock line is high
    - It can change only when the clock line is low.
  - The STOP and START conditions are the only exceptions to this rule.





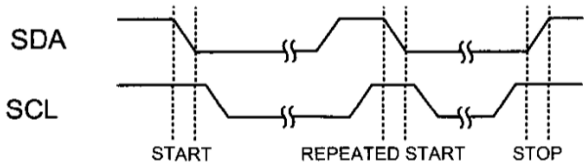

# START and STOP Conditions

- $I^2C$ is a connection-oriented communication protocol.
  - Transmission is initiated by a START condition and is terminated by a STOP condition.
  - START and STOP conditions are generated *by the master*.
- STOP and START conditions must be **distinguished** from bits of address or data.
  - That is why they do not obey the bit format rule mentioned before.
- START and STOP conditions are generated by:
  - Keeping the SCL line high and then changing the SDA line.
    - **START condition**: *high-to-low* change in the SDA line when SCL is *high*.
    - **STOP condition**: *low-to-high* change in the SDA line when SCL is *high*.



# Repeated Start

- The bus is considered busy between each pair of START and STOP conditions
  - No other master tries to take control of the bus when busy
- If a master has the control of the bus and wishes to initiate a new transfer (without releasing the bus):
  - It must issues a new START condition between a pair of START and STOP conditions

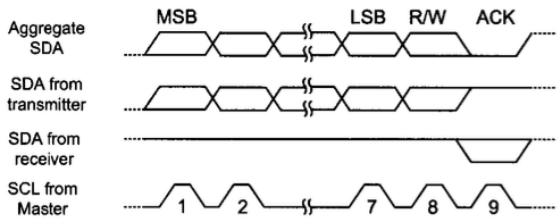




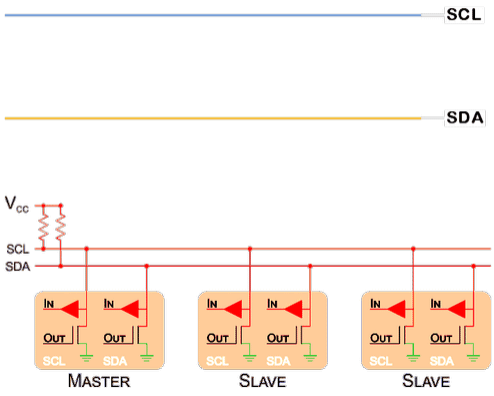

# Packet Format in $I^2C$

9 bits:

- 8 bit data (by transmitter), MSB first
  - Address packets: 7 address bits (slave address, SLA) + 1 R/W bit
  - Data packets
- 1 bit ACK (by receiver) or NACK
  - The receiver pulls the SDA line low to indicate an ACK
  - Otherwise NACK -> master issues STOP or repeated START



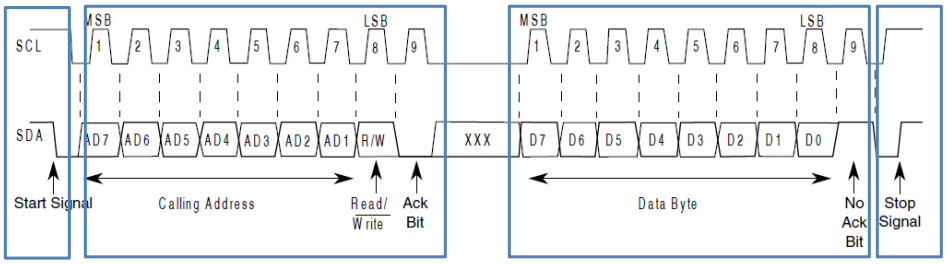
# I²C In Action!




https://www.allaboutcircuits.com



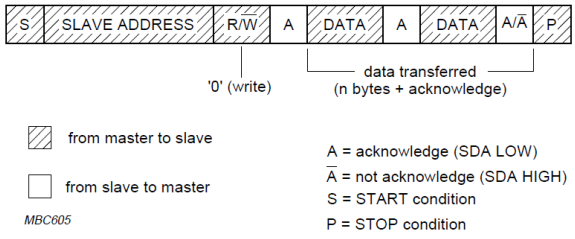
# The I²C Bus Protocol



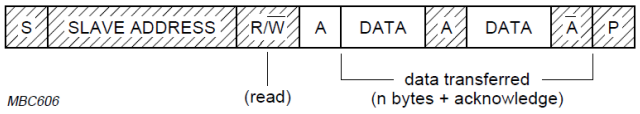

- Message-oriented data transfer with four parts
  - Start condition
  - Slave Address transmission
    - Address
    - Command (read or write)
    - Acknowledgement by receiver
  - Data fields
    - Data byte
    - Acknowledgement by receiver
  - Stop condition

# Master Read/Write

- Master Writing Data to Slave




- Master Reading Data from Slave

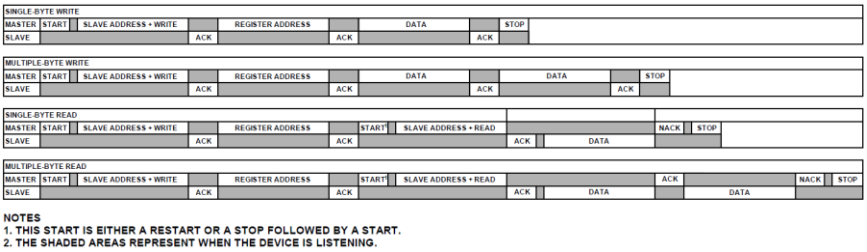





# I²C Addressing

- Each device (IC) has seven-bit address
  - Different types of devices have different default addresses
  - Sometimes can select a secondary default address by tying a device pin to a different logic level
- What if we treat the first byte of data as a register address?
  - Can specify registers within a given device: eight bits
  - Example: An accelerometer has up to 58 registers

# I²C with Register Addressing



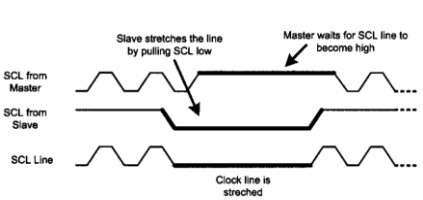

- Master drives communication
  - Sends start condition, address of slave, read/write command
  - Listens for acknowledgement from slave
  - Sends register address (byte)
  - Listens for acknowledgement from slave



# Advanced Features of I²C

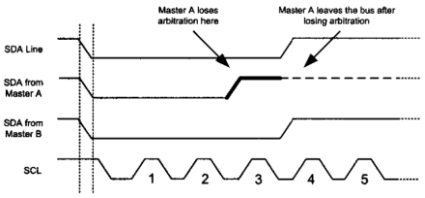
**Clock stretching**

- A kind of flow control
- Slave can freeze the transmission until it becomes ready again
  - by pulling the SCL low so that the master cannot rise it



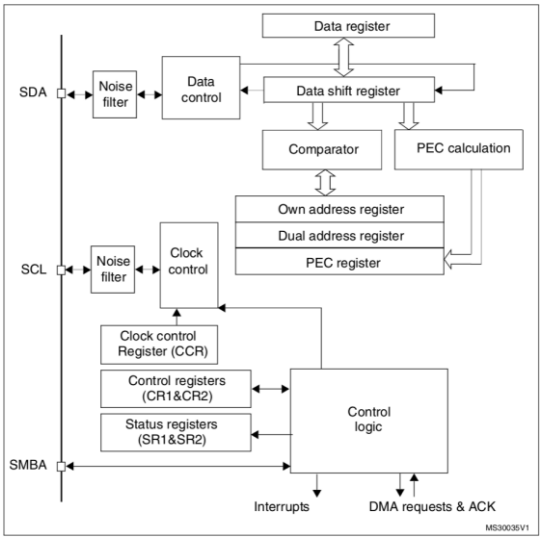

**Arbitration**

- Multiple masters supported
- Each master must wait until it has the bus control
  - Each master initiates a transfer
  - Checks the level of the bus and compare it with the level it expects



## STM32F401 I²C Controller

- Parallel-bus/I²C protocol converter
- Multimaster capability: the same interface can act as Master or Slave
- Master Clock generation and Start and Stop generation
- Slave features:
  - Programmable I²C Address detection
  - Dual Addressing Capability to acknowledge 2 slave addresses
  - Stop bit detection
- Generation and detection of 7-bit/10-bit addressing and General Call
- Supports different communication speeds:
  - 100 kHz, 400 kHz, up to 1 MHz
- Analog noise filter
- Programmable digital noise filter
- Status flags:
  - Transmitter/Receiver mode flag
  - End-of-Byte transmission flag
  - I²C busy flag
- Error flags:
  - Arbitration lost condition for master mode
  - Acknowledgment failure after address/ data transmission
  - Detection of misplaced start or stop condition
  - Overrun/Underrun if clock stretching is disabled
- 2 Interrupt vectors for successful address/ data communication and for error condition
- Optional clock stretching
- 1-byte buffer with DMA capability
- Configurable PEC (packet error checking) generation or verification






# I²C Slave Mode

- By default I²C operates in slave mode
- Detect start condition
- Compare the address received from the SDA line with interface address (OAR1) and with OAR2 (if ENDUAL=1) or the General Call address (if ENGC=1)
  - Header or address not matched: ignores it and wait for another start condition
  - Header matched (10-bit mode only)
  - Address matched: ACK or Interrupt or read the DUALF bit
- Following the address reception and after clearing ADDR Slave will either
  - Send bytes from DR register to SDA line via shift register if in transmitter mode
  - Receive bytes from SDA line to DR register via shift register if in receiver mode
- ACK or interrupt upon after each byte is completed.

# I²C Master Mode

- Generation of a Start condition will switch the Salve mode to Master mode
- Master initiates a data transfer and generates the clock signal.
- After the Start condition, read the Status register 1 and write the Data register with the Slave address
- Then wait for a read of the SR1 register followed by a read of the SR2 register
- Then send the slave address with LSB cleared to enter Transmitter mode or with LSB set to enter Receiver mode
- Following the address reception and after clearing ADDR Master will either
  - Send bytes from DR register to SDA line via shift register if in transmitter mode
  - Receive bytes from SDA line to DR register via shift register if in receiver mode
- Wait for ACK or ACK and then continue for the next byte
- Send Stop condition



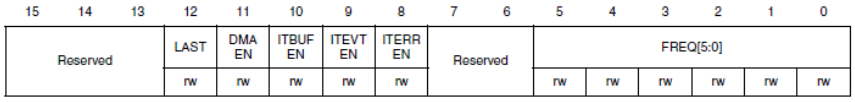
# I²C Control Register 1 I2C_CR1

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SWRST | Res. | ALERT | PEC | POS | ACK | STOP | START | NO STRETCH | ENGC | ENPEC | ENARP | SMB TYPE | Res. | SMBUS | PE |
| rw | | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | | rw | rw |

- ACK: Acknowledge enable
- STOP: Stop generation
  - In master mode, if set, will generate a Stop condition after the current byte transfer
  - In slave mode, if set, will release the SCL and SDA line after the current byte transfer
- START: Start generation
- PE: Peripheral enable

- When STOP, START or PEC bit is set, the software must not perform any write access to CR1 before this bit is cleared by hardware in case it does not set a second STOP, START or PEC request.
- Must not reset PE bit before the end of the communication in master mode.

# I²C Control Register 2 I2C_CR2

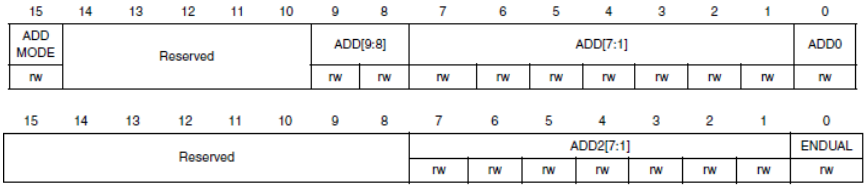


| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | LAST | DMA EN | ITBUF EN | ITEVT EN | ITERR EN | Reserved | | FREQ[5:0] | | | | | |
| | | | rw | rw | rw | rw | rw | | | rw | rw | rw | rw | rw | rw |

- Interrupt enable bits
- FREQ[5:0]: Peripheral clock frequency
  - I2C peripheral connected to APB
  - Configure the clock frequency between 2MHz(0b000010) to 42MHz(0b101010)
  - Baud rate = Freq (if not in fast mode)



# I²C Own address registers



| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADD MODE | Reserved | | | | | ADD[9:8] | | ADD[7:1] | | | | | | | ADD0 |
| rw | | | | | | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | ADD2[7:1] | | | | | | | ENDUAL |
| | | | | | | | | rw | rw | rw | rw | rw | rw | rw | rw |

- OAR1
  - ADDMODE: 7-bit slave address or 10-bit slave address
  - Bit 14 should be kept at 1 by software
  - 10-bit mode: ADD[9:0]: interface address
  - 7-bit mode: ADD[7:1]: interface address
- OAR2
  - Dual addressing mode
  - ENDUAL: Enable dual addressing mode with 1
  - ADD2[7:1]: secondary address in dual addressing mode

# I²C Status Register 1 I2C_SR1

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SMB ALERT | TIME OUT | Res. | PEC ERR | OVR | AF | ARLO | BERR | TxE | RxNE | Res. | STOPF | ADD10 | BTF | ADDR | SB |
| rc_w0 | rc_w0 | | rc_w0 | rc_w0 | rc_w0 | rc_w0 | rc_w0 | r | r | | r | r | r | r | r |

- Error related bits
- TxE: Data register empty (transmitters)
  - 0: DR not empty
  - 1: DR empty
- RxNE: Date register empty (receivers)
  - 0: DR empty
  - 1: DR not empty
- STOPF: Stop detection (Slave mode)
- BTF: Byte transfer finished
- ADDR: Address sent (master mode)/matched(slave mode)
- SB: Start bit(Master mode)



# I²C Status Register 2 I2C_SR2

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PEC[7:0] | | | | | | | | DUALF | SMB HOST | SMBDE FAULT | GEN CALL | Res. | TRA | BUSY | MSL |
| r | r | r | r | r | r | r | r | r | r | r | r | | r | r | r |

- GENCALL: General call address (slave mode)
- TRA: Transmitter / Receiver
- BUSY: Bus busy
- MSL: Master / Slave

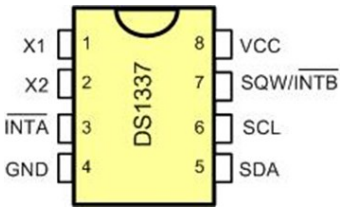
# Required sequence

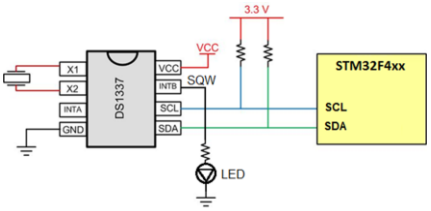
- Program the peripheral input clock in I2C_CR2 Register to generate right timings
- Configure the clock control registers
- Configure the rise time registers
- Enable the peripheral
- Set the START bit to generate a Start condition



# DS1337 Real-Time Clock

- RTC provides accurate time and date information
- DS1337 is a serial RTC with an $I^2C$ bus
- provides seconds, minutes, hours, day, date, month, and year information
- X1-X2: Connect to standard 32.768 kHz quartz crystal
- SCL, SDA: $I^2C$ bus
  - address of DS1337: (1001101)



# Address Map of the DS1337

- CH bit in address 00: disables the oscillator
- Byte addresses 0-6 are set aside for the time and date (BCD format)
  - select 12-hour or 24-hour mode with bit 6 of hour location 02

| Address | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 | Function | Range |
|---|---|---|---|---|---|---|---|---|---|---|
| 00H | 0 | 10 Seconds | | | Seconds | | | | Seconds | **00-59** |
| 01H | 0 | 10 Minutes | | | Minutes | | | | Minutes | **00-59** |
| 02H | 0 | 12/24 | 10 hour PM/AM | 10hour | Hours | | | | Hours | 1-12<br>0-23 |
| 03H | 0 | 0 | 0 | 0 | 0 | Day | | | Day | 0-7 |
| 04H | 0 | 0 | 10 Date | | Date | | | | Date | 01-31 |
| 05H | Century | 0 | 0 | 10Mnt | Month | | | | Month<br>Century | 1-12+<br>Century |
| 06H | 10 Year | | | | Year | | | | Year | 00-99 |
| 07H | A1M1 | 10 Seconds | | | Seconds | | | | AlArm 1 Seconds | **00-59** |
| 08H | A1M2 | 10 Minutes | | | Minutes | | | | AlArm 1 Minutes | **00-59** |
| 09H | A1M3 | 12/24 | AM/PM<br>10 Hour | 10 Hour | Hour | | | | AlArm 1 Hours | 1-12<br>00-23 |
| 0AH | A1M4 | DY/DT | 10 Date | | Day | | | | AlArm 1 Day | 1-7 |
| | | | | | Date | | | | AlArm 1 Date | 01-31 |
| 0BH | A2M2 | 10 Minutes | | | Minutes | | | | AlArm 2 Minutes | **00-59** |
| 0CH | A2M3 | 12/24 | AM/PM<br>10 Hour | 10 Hour | Hour | | | | AlArm 2 Hours | 1-12<br>00-23 |
| 0DH | A2M4 | DY/DT | 10 Date | | Day | | | | AlArm 2 Day | 1-7 |
| | | | | | Date | | | | AlArm 2 Date | 01-31 |
| 0EH | EOSC | 0 | 0 | RS2 | RS1 | INTCN | A2IE | A1IE | Control | - |
| 0FH | OSF | 0 | 0 | 0 | 0 | 0 | A2F | A1F | Status | - |



# PROTOCOL COMPARISON

# Factors to Consider

- How fast can the data get through?
  - Depends on raw bit rate, protocol overhead in packet
- How many hardware signals do we need?
  - May need clock line, chip select lines, etc.
- How do we connect multiple devices (topology)?
  - Dedicated link and hardware per device or point-to-point
  - One bus for master transmit/slave receive, one bus for slave transmit/master receive
  - All transmitters and receivers connected to same bus – multi-point
- How do we address a target device?
  - Discrete hardware signal (chip select line)
  - Address embedded in packet, decoded internally by receiver
- How do these factors change as we add more devices?



# SPI Advantages Compared to I²C

- Full duplex communication in the default version of this protocol
- Push-pull drivers (as opposed to open drain) provide good signal integrity and high speed
- Higher throughput than I²C or SMBus. Not limited to any maximum clock speed, enabling potentially high speed
- Complete protocol flexibility for the bits transferred
  - Not limited to 8-bit words
  - Arbitrary choice of message size, content, and purpose
- Extremely simple hardware interfacing
  - Typically lower power requirements than I²C or SMBus due to less circuitry (including pull up resistors)
  - No arbitration or associated failure modes
  - Slaves use the master's clock and do not need precision oscillators
  - Slaves do not need a unique address – unlike I²C or GPIB or SCSI
  - Transceivers are not needed
- At most one unique bus signal per device (chip select); all others are shared
- Signals are unidirectional allowing for easy galvanic isolation
- Simple software implementation

# SPI Disadvantages Compared to I²C

- Requires more pins on IC packages than I²C, even in the three-wire variant
- No hardware flow control by the slave (but the master can delay the next clock edge to slow the transfer rate)
- No hardware slave acknowledgment (the master could be transmitting to nowhere and not know it)
- Typically supports only one master device (depends on device's hardware implementation)
- No error-checking protocol is defined
- SPI does not support hot swapping (dynamically adding nodes).
- Not a formal standard -> conformance checking not possible

- Both only handle short distances compared to RS-232, RS-485, or CAN-bus.



# Protocol Trade-Offs for Communication to N Devices

| Protocol | Speed | Signals Req. for Bidirectional Communication with N devices | Device Addressing | Topology |
|---|---|---|---|---|
| UART (Point to Point) | Fast – Tens of Mbit/s | 2*N: TxD, RxD | None | Point-to-point full duplex |
| UART (Multi-drop) | Fast – Tens of Mbit/s | 2: TxD, RxD | Added by user in software | Multi-drop |
| SPI | Fast – Tens of Mbit/s | 3+N: SCLK, MOSI, MISO, and one SS per device | Hardware chip select signal per device | Multi-point full-duplex, multi-drop half-duplex buses |
| I²C | Moderate – 100 kbit/s, 400 kbit/s, 1 Mbit/s, 3.4 Mbit/s. Packet overhead. | 2: SCL, SDA | In packet | Multi-point half-duplex bus |