I2C / SPI / UART Protocols

I2C:

- I²C (Inter-Integrated Circuit) is a • Synchronous • multi-master • multi-slave • packet switched
- It is widely used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication.
- A particular strength of I2C is the capability of a microcontroller to control a network of device chips with just two general-purpose I/O pins and software. Many other bus technologies used in similar applications, such as Serial Peripheral Interface **(SPI)** Bus, require more pins and signals to connect multiple devices.

Design (H/w):

- I2C uses only two bidirectional open-drain lines, Serial Data Line (SDA) and Serial Clock Line (SCL), pulled up with resistors.
- Typical voltages used are +5 V or +3.3 V, although systems with other voltages are permitted.
- The I2C bus drivers are "open drain", meaning that they can pull the corresponding signal line low, but cannot drive it high. Thus, there can be no bus contention where one device is trying to drive the line high while another tries to pull it low, eliminating the potential for damage to the drivers or excessive power dissipation in the system. Each signal line has a pull-up resistor on it, to restore the signal to high when no device is asserting it low.
-The I2C reference design has a 7-bit address space.
- Common I2C bus speeds are the 100 kbit/s standard mode and the 400 kbit/s Fast mode.
- The maximal number of nodes is limited by the address space and also by the total bus capacitance of 400 pF, which restricts practical communication distances to a few meters.
- The relatively high impedance and low noise immunity requires a common ground potential, which again restricts practical use to communication within the same PC board or small system of boards.

Design (Architecture - Master~Slaves):

- The bus has two roles for nodes: master and slave:
    - Master node – node that generates the clock and initiates communication with slaves.
    - Slave node – node that receives the clock and responds when addressed by the master.
- The bus is a multi-master bus, which means that any number of master nodes can be present.
- Additionally, master and slave roles may be changed between messages (after a STOP is sent).
- There may be four potential modes of operation for a given bus device, although most devices only use a single role and its two modes:
    - master transmit – master node is sending data to a slave,
    - master receive – master node is receiving data from a slave
    - slave transmit – slave node is sending data to the master
    - slave receive – slave node is receiving data from the master.

I2C Protocol (Basics):

Frame Format:

- Messages are broken up into two types of frame:
    - Address frame: The master indicates the slave to which the message is being sent
    - Data frame: These are 8-bit data messages passed from master to slave or vice versa.
- Data is placed on the (Serial Data Line) SDA line after (Serial Clock Line) SCL goes low, and is sampled after the SCL line goes high.

Start Condition:

- To initiate the address frame, the master device leaves SCL high and pulls SDA low.
- This puts all slave devices on notice that a transmission is about to start.
- If two master devices wish to take ownership of the bus at one time, whichever device pulls SDA low first wins the race and gains control of the bus.
- It is possible to issue repeated starts, initiating a new communication sequence without relinquishing control of the bus to other masters.

Address Frame:

• The address frame is always first in any new communication sequence• For a 7-bit address, the address is clocked out most significant bit (MSB) first, followed by a R/W bit indicating whether this is a read (1) or write (0) operation.

NACK/ACK bit:

- It is the 9th bit of the frame
- This is the case for all frames (data or address).
- Once the first 8 bits of the frame are sent, the receiving device is given control over SDA.
- If the receiving device does not pull the SDA line low before the 9th clock pulse, it can be inferred that the receiving device either did not receive the data or did not know how to parse the message.
- In that case, the exchange halts, and it's up to the master of the system to decide how to proceed.

Data Frames:

• After the address frame has been sent, data can begin being transmitted.
• The master will simply continue generating clock pulses at a regular interval, and the data will be placed on SDA by either the master or the slave, depending on whether the R/W bit indicated a read or write operation.

Stop condition:

• Once all the data frames have been sent, the master will generate a stop condition.
• Stop conditions are defined by a 0->1 (low to high) transition on SDA after a 0->1 transition on SCL, with SCL remaining high.
• During normal data writing operation, the value on SDA should not change when SCL is high, to avoid false stop conditions.

======>

***1**. The master is initially in master transmit mode by sending a START followed by the 7-bit address of the slave it wishes to communicate with*
***2**. This is finally followed by a single bit representing whether it wishes to write (0) to or read (1) from the slave.*
3. If the slave exists on the bus then it will respond with an ACK bit (active low for acknowledged) for that address.
***4**. The master then continues in either transmit or receive mode (according to the read/write bit it sent), and the slave continues in the complementary mode (receive or transmit,respectively).*
***5**. The address and the data bytes are sent most significant bit first.*
***6**. The master terminates a message with a STOP condition if this is the end of the transaction or it may send another START condition to retain control of the bus for another message (a "combined format" transaction).*

- *If the master wishes to write to the slave, then it repeatedly sends a byte with the slave sending an ACK bit. (In this situation, the master is in master transmit mode, and the slaves in slave receive mode.)*
- *If the master wishes to read from the slave, then it repeatedly receives a byte from the slave, the master sending an ACK bit after every byte except the last one. (In this situation, the master is in master receive mode, and the slave is in slave transmit mode.)*
- *An I2C transaction may consist of multiple messages.*

• I2C defines basic three types of transactions, each of which begins with a START and ends with a STOP:
    • Single message where a master writes data to a slave.
    • Single message where a master reads data from a slave.
    • Combined format, where a master issues at least two reads or writes to one or more slaves.
• In a combined transaction, each read or write begins with a START and the slave address. The START conditions after the first are also called repeated START bits. Repeated STARTs are not preceded by STOP conditions, which is how slaves know that the next message is part of the same transaction.

Timing Diagram:

***1**. Data transfer is initiated with a start bit (S) signaled by SDA being pulled low while SCL stays high.*
***2**. SCL is pulled low, and SDA sets the first data bit level while keeping SCL low*
***3**. The data are sampled (received) when SCL rises for the first bit (B1). For a bit to be valid, SDA must not change between a rising edge of SCL and the subsequent falling edge*
***4**. This process repeats, SDA transitioning while SCL is low, and the data being read while SCL is high (B2, ...Bn).*
***5**. The final bit is followed by a clock pulse, during which SDA is pulled low in preparation for the stop bit.*
***6**. A stop bit (P) is signaled when SCL rises, followed by SDA rising.*

Applications of I2C:

Common applications of the I2C bus are:
• Describing connectable devices via small ROM configuration tables to enable "plug and play" operation, such as
• Accessing real-time clocks and NVRAM chips that keep user settings.
• Accessing low-speed DACs and ADCs, • Controlling LCD displays.

SPI:

- The Serial Peripheral Interface bus (SPI) is synchronous.

- Serial communication interface specification used for short distance communication.

- *The interface was developed by Motorola in 1980s*
- *SPI devices communicate in **full duplex** mode using a **master-slave architecture with a single master.***

- The master device originates the frame for reading and writing. Multiple slave devices are supported through selection with individual slave select (SS) lines.

The trouble with asynchronous protocols:

• The common serial port, such as UART, is called "asynchronous" because there is no control over when data is sent or any guarantee that both sides are running at precisely the same rate.

• There can be an issue when two systems with slightly different clocks try to communicate with each other.

• To mitigate this issue, asynchronous serial connections add extra start and stop bits to each byte, so as to help the receiver sync up to data as it arrives.

• Both sides must also agree on the same baud rate in advance.

• Asynchronous serial works just fine, but has a lot of overhead in both the extra start and stop bits sent with every byte the complex hardware required to send and receive data. If both sides aren't set to the same speed, the received data will be garbage. This is because the receiver is sampling the bits at very specific times If the receiver is looking at the wrong times, it will see the wrong bits.

How is 'synchronous' beneficial? (Like SPI)

• A "synchronous" data bus uses separate lines for data and clock that keeps both sides in perfect sync.

• The clock is an oscillating signal that tells the receiver exactly when to sample the bits on the data line.

- *This could be the rising (low to high) or falling (high to low) edge of the clock signal; the datasheet will specify which one to use.*

• When the receiver detects that edge, it will immediately look at the data line to read the next bit

• Because the clock is sent along with the data, specifying the speed isn't important, although devices will have a top speed at which they can operate

The SPI Protocol - basics:

• The SPI bus can operate with a single master device and with one or more slave devices.

• One unique feature of SPI is that data can be transferred without interruption.

• Any number of bits can be sent or received in a continuous stream.

- *With I2C and UART, data is sent in packets, limited to a specific number of bits.*
- *Start and stop conditions define the beginning and end of each packet, so the data is interrupted during transmission.*

• Devices communicating via SPI are in a master-slave relationship.

• In SPI, only one side generates the clock signal The side that generates the clock is called the "master", and the other side is called the "slave".

• The master is the controlling device (usually a microcontroller), while the slave (usually a sensor, display, or memory chip) takes instruction from the master.

• There is always only one master but there can be multiple slaves.

The SPI Interface:

The SPI bus specifies four logic signals:

- *SCLK: Serial Clock (output from master).*
- *MOSI: Master Output Slave Input (data output from master).*
- *MISO: Master Input Slave Output, or Master In Slave Out (data output from slave).*
- *SS: Slave Select (often active low, output from master).*

Serial Clock (SCLK):

• SPI is a synchronous communication protocol.

• The clock signal synchronizes the output of data bits from the master with the sampling of bits by the slave.

• One bit of data is transferred in each clock cycle, so the speed of data transfer is determined by the frequency of the clock signal.

• SPI communication is always initiated by the master since the master configures and generates the clock signal.

Slave Select (SS):

• The master can choose which slave it wants to communicate to by setting the slave's CS/SS line to a low voltage level.

• In the idle, non-transmitting state, the slave select line is kept at a high voltage level.

- *Multiple CS/SS pins may be available on the master, which allows for multiple slaves to be wired in parallel.*
- *If only one CS/SS pin is present, multiple slaves can be wired to the master by daisy-chaining*

Multiple Slaves:

- *Multiple CS/SS pins are available on the master*
- *Only one CS/SS pin is present, multiple slaves are wired to the master by daisy-chaining*

SPI The Protocol:

***1**. To begin communication, the bus master configures the clock, using a frequency supported by the slave device, typically up to a few MHz.*

***2**. The master then selects the slave device with a logic level 0 on the select line.*

- *If a waiting period is required, such as for an analog-to-digital conversion, the master must wait for at least that period of time before issuing clock cycles.*

***3**. During each SPI clock cycle, a full duplex data transmission occurs.*

- *The master sends a bit on the MOSI line and the slave reads it, while the slave sends a bit on the MISO line and the master reads it.*

The Protocol (summary)

***1**. The master outputs the clock signal*

***2**. The master switches the SS/CS pin to a low voltage state, which activates the slave*

***3**. The master sends the data one bit at a time to the slave along the MOSI line. The slave reads the bits as they are received*

***4**. If a response is needed, the slave returns data one bit at a time to the master along the MISO line. The master reads the bits as they are received*

Data Transmission in SPI:

• Transmissions normally involve two shift registers of some given word size (may be 8eight bits), one in the master and one in the slave

• They are connected in a virtual ring topology.

• Data is usually shifted out with the most-significant bit first. On the clock edge, both master and slave shift out a bit and output it on the transmission line to the counterpart.

• On the next clock edge, at each receiver the bit is sampled from the transmission line and set as a new least-significant bit of the shift register.

• After the register bits have been shifted out and in, the master and slave have exchanged register values.

• If more data needs to be exchanged, the shift registers are reloaded and the process repeats.

• Transmission may continue for any number of clock cycles.

• When complete, the master stops toggling the clock signal, and typically deselects the slave.

Advantages of SPI (+):

• Full duplex

• 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

• Not limited to 8-bit words

• 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

Limitations of SPI (--):

• Requires more pins on IC packages than I2C

• 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

Applications of SPI:

SPI is used to talk to a variety of peripherals, such as

• Sensors: temperature, pressure, ADC, touchscreens, video game controllers

• Control devices: audio, codecs, digital potentiometers, DAC

• Memory: flash and EEPROM

• Real-time clocks

• LCD, sometimes even for managing image data

• Any MMC or SD card

Review Questions !!!!

1. Compare and contrast UART, I2C and SPI, citing the features, advantages and disadvantages of each.

2. What are the demerits of asynchronous communication protocols? How are they overcome in SPI?

3. List out the Interface signals used in SPI protocol and brief out their significance.

4. List the set of operations involved in completing a serial communication using SPI protocol. Draw the hardware interface, in support.

5. List out the advantages and limitations of SPI protocol. Also, list some of the applications.

UART:

- *UART – Stands for Universal Asynchronous Receiver Transmitter*
- *USART – Stands for Universal Synchronous Asynchronous Receiver Transmitter*
- *In RS-232 we implement serial port with UART*
- *Actually UART receives/sends data to microprocessor/microcontroller through data bus. The remaining part of signal handling of RS-232 is done by UART i.e. start bit, stop bit, parity etc.*

WHY USE A UART?:

• A UART may be used when:

- *High speed is not required*
- *An inexpensive communication link between two devices is required*

• UART communication is very cheap

- *Single wire for each direction (plus ground wire), Asynchronous because no clock signal is transmitted*
- *Relatively simple hardware*

BASICS OF SERIAL COMMUNICATION

• Bit rate:

-Number of bits sent every second (BPS)

• Baud rate:

-Number of symbols sent every second, where every symbol can represent more thanone bit.

Ex. high-speed modems which use phase shifts to make every data transition period represent more than one bit.

-For the PIC 16f877A's USART, with every clock tick one bit is sent, each symbol represents one bit.

-So, we can consider bit rate and baud rate the same thing.

TRANSMISSION REQUIREMENT:

- *Before transmission begins, transmitter and receiver must agree on :*
  - Baud rate (75, 150, 300, 600, etc)
  - 1, 1.5 or 2 stop bits
  - 5, 6, 7 or 8 data bits
  - even, odd or no parity

FUNCTION OF VARIOUS PINS ON SERIAL PORT:

Pin No.	Pin Symbol	Function
1	*CD*	*Carrier Detect: It is used by Modem to inform PC that it has detected Carrier on Phone Line.*
2	*RD*	*Serial data is received on this line by PC.*
3	*TD*	*Serial Data is transmitted on this pin by PC.*
4	*DTR*	*Data Terminal Ready. When terminal (computer) powers up it asserts DTR high.*
5	*SG*	*It is signal ground with reference to which voltages are interpreted as high or low.*
6	*DSR*	*Data Set Ready. When modem powers up it asserts DSR high.*
7	*RTS*	*Request to Send. Request to send is sent from (DTE) terminal (PC) to modem (DCE) to inform it that PC wants to send some data to modem.*
8	*CTS*	*Clear To Send. Upon received RTS from DTE (PC), the modem (DCE) asserts CTS high whenever it is ready to receive data.*
9	*RI*	*Ring Indicator. It is set by modem to indicate the PC that a ringing signal has been detected on line.*

TRANSMITTER CONTD:

TXIF bit : in the PIR1 register

- *Indicates when data can be written to TXREG(when data is moved from TXREG into the Transmit Shift Register,*
- *It cannot be cleared in software. It will reset only when new data is loaded into the TXREG register.*
- *It doesn't indicate that the transmission has completed.*

TRMT bit:

- *Once the data in the TSR register has been clocked out on the TX pin(at the beginning of the STOP bit), the TRMT bit in the TXSTA register will be set,*
- *Indicating that the transmission has been completed.*

RECEIVER CONTD:

1. Every received bit is sampled at the middle of the bit's time period.
2. The USART can be configured to receive eight or nine bits by the RX9 bit in the RCSTA register.
3. After the detection of a START bit, eight or nine bits of serial data are shifted from the RX pin into the Receive Shift Register, one bit at a time.
4. After the last bit has been shifted in, the STOP bit is checked and the data is moved into the FIFO buffer.
5. RCREG is the output of the two element FIFO buffer. A next start bit can be sent immediately after the stop bit.
6. RCIF: indicates when data is available in the RCREG.

UART REGISTERS:

- To use and control the UART, special internal registers are assigned to them. Usually there will be at least four registers: control, status, receive and transmit registers. All these vary in size depending on the MCU.

1) Control Register - Contains settings for the UART. Some common settings/features include: Number of data bits, number of stop bits, parity control, UART TX/RX enable/disable, baud rate setting, RX/TX interrupt enable, etc.

2) Status Register - From its name, this contains information about the UART's condition or state. During run-time, this register may be helpful in guiding the processor on the next instruction to execute like when to retrieve data. Information that can be retrieved include: data send/receive ready, etc.

3) Receive Register - This is the where received data is temporarily stored.

4) Transmit Register - A buffer register/s for temporarily storing data to be sent

APPLICATIONS:

- Communication between distant computers

- *Serializes data to be sent to modem*
- *De-serializes data received from modem*

- PC serial port is a UART!

- *Serializes data to be sent over serial cable*
- *De-serializes received data*

• The UART is also responsible for baud rate generation.

• This determines the speed at which data is transmitted and received. One baud is one bit per second (bps).

• With modern UARTs, 230,400 baud can be achieved with a short cable length of a few feet.

SUMMARY:

To make asynchronous connection using the PIC's USART:

• At the Transmitter end:

1. Determine the value of the SPBRG register and the BRGH-bit according to the required baud rate.
2. The SYNC-bit [TXSTA<4>] is cleared, SPEN-bit [RCSTA<7>] is set to enable the serial port.
3. On 9-bit data transmission, the TX9-bit [TXSTA<6>] is set.
4. TXEN-bit [TXSTA<5>] is set to enable data transmission.
5. On 9-bit data transmission, value of the ninth bit should be written to the TX9D-bit [TXSTA<0>].
6. Transmission can be started again by writing 8-bit data to the TXREG register, usually wait for at least (1ms) between every two writes.

• At the Receiver end:

1.Determine the value of the SPBRG register and the BRGH-bit according to the required baud rate.
2.The SYNC-bit [TXSTA<4>] is cleared, SPEN-bit [RCSTA<7>] is set to enable the serial port.
3.On 9-bit data receive, the RX9-bit [RCSTA<6>] is set.
4.Data receive should be enabled by setting the CREN-bit [RCSTA<4>].
5.The RCSTA register should be read to get information on possible errors which have occurred during transmission.
6.On 9-bit data receive, the ninth bit will be stored in RX9D-bit [RCSTA<0>].
7.Received 8-bit data stored in the RCREG register should be read.