# Notes - Unit 10

## SERIAL COMMUNICATION

### SERIAL COMMUNICATION INTERFACE (SCI)

- SCI: Interface that transfers data in asynchronous mode using the TIA-232 standard (originally called RS-232).
- TIA-232 standard: Often used in data communication between a Data Terminal Equipment (DTE) and a Data Communication equipment (DCE). DTE can be either a computer or a terminal. A DCE is a modem.
- Four aspects of the TIA-232 standard:
  - ✓ Electrical specifications: Data rates (lower than 20,000 bps), voltages (logic '1': -3 to -25V; logic '0': 3 to 25 V), transfer distances.
  - ✓ Functional specifications: 22 signals divided into six categories (signal ground and shield, primary communications channel, secondary communications channel, modem status and control signals, transmitter and receiver timing signals, channel test signals).
  - ✓ Mechanical specifications: A 25-pin D-type connectors is specified. Since only a small subset is actually used, a 9-pin connector (DB9) is used in most PCs.
  - ✓ Procedural specifications: This is the sequence of events that occurs during data transmission using the TIA-232 standard. Common examples include connecting two DTEs via DCEs.

### DATA FORMAT

- Character encoding is set by the serial port hardware (it is not defined by the TIA-232 standard). SCI uses asynchronous data transfer, where data is transferred character by character, and start and stop bits are used to indicate the beginning bit. This method of transmission is used when data (characters) are sent intermittently as opposed to in a solid stream. Also, the transmitter and receiver are not synchronized by a common clock; instead they use their own clocks.
- **Format of a Frame:** Start bit ('0'), 7 to 9 data bits (LSB transmitted first), optional parity bit, and a stop bit ('1').
- Data communication: `RxD` (receive pin), `TxD` (transmit pin)
- **Receiver**: It uses a clock signal whose frequency is a multiple (usually 16) of the incoming data rate.
- Start bit: Detected when the `RxD` pin was '1' for at least 3 sampling times. After the first low sample (i.e., after the falling edge is detected), the third, fifth, and seventh samples are checked. If the majority of these samples are low, then the start bit is valid. Otherwise, the SCI restart the process.
- Data bit: If the majority of the 8th, 9th, and 10th samples are 1, the data bit is 1. If the majority are 0, the data bit is 0.
- **Baud rate**: Number of signal or symbol changes per second. For RS-232, baud rate is identical to bit rate, since each symbol uses one bit.
- Data Transmission Errors:
  - ✓ Framing error: The stop bit is absent.
  - ✓ Receiver overrun: One of more characters received but not read because subsequent characters were received.
  - ✓ Parity error: Detected by a parity error detecting circuit.
- Null modem connection: When 2 DTE devices are located side by side and use the TIA-232 interface to exchange data, the modems are redundant. Here, a null modem connection is more efficient: both DTEs are fooled into thinking they are connected by modems.

### HCS12 SERIAL COMMUNICATION INTERFACE:

- Two SCI modules: SCI0 and SCI1.
- Interrupts: Each module can generate an Interrupt: SCI0 Interrupt, SCI1 Interrupt. Each module has four interrupt request sources, which are OR'ed in order to get only one Interrupt signal.
- Dragon12-Light Board: SCI0 connected to the Serial Monitor. SCI1 can be used with the USB port.
- Each module has a `RxD`, TxD pin, an E-clock input, and an SCIn Interrupt.
- 1 start bit, 8 (or 9) data bits, optional parity bit, 1 stop bit.
- SCI operation: Two baud rate registers, two control registers, two status registers, two data registers.
- SCI can send a break (transmission or reception of logic 0 for a frame or longer) to attract the attention of the other party of communications.
- HCS12: parity bit supported. Odd and even parity. When enabled, a parity bit is generated by hardware for transmitted and received data. Received parity errors are flagged in hardware (in the `SCInSR1` register).
- Interfacing SCI with TIA-232: The SCI circuit uses 0 and 5v to represent '0' and '1' logic levels respectively, and thus it cannot be connected to the TIA-232 interface circuit directly. A TIA-232 transceiver (e.g. MAX232, ICL232) is required to translate the voltage levels of the SCI signals (`RxD` and `TxD`) to and from those of the corresponding SCI signals.

### Baud rate generation:

- SCI: The baud rate is identical to the data (or bit) rate. The data rate is the rate at which the incoming data arrives as well as the rate at which we transmit data.

- Transmitter: The clock frequency (also Baud Rate) is given by:

$$Baud\ Rate = Tx\ clock = \frac{E - clock}{16 \times SBR}$$

- *Receiver*: The clock frequency here is 16 times the baud rate (or data rate), and it is given by:

$$Rx\ clock = \frac{E - clock}{SBR}$$

- The value $SBR$ (SCI Baud Rate) can be written on the registers `SCInBDH:SCInBDL`, n=0,1. We can only program 13 bits, i.e. the three MSBs of `SCInBDH` are zero. SBR is obtained by: $SBR = \left\lfloor \frac{E-clock}{16\times(Baud\ Rate)} \right\rfloor$
- By default, after reset, SBR=4. Thus, for E-clock=24 MHz, Baud Rate = 375000.

**SCI operation**:
- Control register `SCInCR1`, n=0,1
  - ✓ `M = SCInCR1(4)`: Data format mode bit. '0' for 8 data bits, '1' for 9 data bits.
  - ✓ `PE = SCInCR1(1)`: Parity enable bit. '0' to disable, '1' to enable
  - ✓ `PT = SCInCR1(0)`: Parity type bit (both TX and RX). '0' for even parity, '1' for odd parity.
- Control Register `SCInCR2`, n=0,1
  - ✓ `TIE = SCInCR2(7)`: Transmit interrupt enable bit. Enables(1)/disables(0) the TDRE interrupt requests.
  - ✓ `TCIE = SCInCR2(6)`: Transmit complete interrupt enable bit. Enables(1)/disables(0) TC interrupt requests.
  - ✓ `RIE = SCInCR2(5)`: Receiver full interrupt enable bit. Enables(1)/disables(0) RDRF and OR interrupt requests.
  - ✓ `ILIE = SCInCR2(4)`: Idle Line Interrupt enable bit. Enables(1)/disables(0) IDLE interrupt requests.
  - ✓ `TE = SCInCR2(3)`: Transmitter enable bit. '1' to enable the transmitter, '0' to disable.
  - ✓ `RE = SCInCR2(2)`: Receiver enable bit. '1' to enable receiver, '0' to disable.
- Status Register `SCInSR1`, n=0,1
  - ✓ `TDRE = SCInSR1(7)`: Transmit Data Register Empty flag. This flag is set to '1' every time a byte is transferred from the TX buffer to the Transmit shift register. If it is set to '1', we can write data onto the TX buffer.
  - ✓ `RDRF = SCInSR1(6)`: Receiver Data Register Full Flag
  - ✓ `IDLE = SCInSR1(5)`: Idle line detected Flag.
  - ✓ `OR = SCInSR1(4)`: overrun error Flag.
  - ✓ `PF = SCInSR1(0)`: parity error Flag.
- Status Register `SCInSR2`, n=0,1
- SCI module allows full duplex, asynchronous, non-return-to-zero (NRZ) serial communication between the CPU and remote devices (including other CPUs). NRZ: 1s have a value, 0s another value, no rest condition. The transmitter and receiver operate independently. The CPU monitors the SCI status, writes data to be transmitted, and process received data.

**Character transmission:**
- To transmit data, we write onto the SCI data registers (`SCInDRH/SCInDRL`, n=0,1), also called TX buffer. These bits are then transferred to the Transmit shift register. A start bit and stop bit are appended. `SCInDRH`: Only 9th is written here.
- Procedure:
  1. Configure SCI Transmitter:
     - Set Baud Rate (via SBR).
     - Configure word length, parity, etc. via the `SCInCR1` register
     - Enable Transmitter and Interrupt requests by writing to `SCInCR2`.
  2. Set a Transmit procedure for each character. Poll the `TDRE` flag, if it is '1', we can write on `SCInDRH/SCInDRL`. We write the 9th bit on the MSB of `SCInDRH` if the SCI is in 9-bit format. A new Transfer cannot happen until the TDRE flag has been cleared.
- Setting `TE` from 0 to 1: It automatically loads the Transmit shift register with a preamble: 10 logic ones (M=0) or 11 logic 1s (M=1). After the preamble shifts out, the contents of the SCI Data Register is transferred into the Transmit shift register. When parity is enabled, the MSB (last bit to the sent) is the parity bit.
- `TDRE` flag: it becomes '1' when the SCI data register transfers a byte to the transmit shift register. It means that the SCI data register is ready to accept new data. If TIE is also '1', the TDRE flag generates a transmit interrupt request. This `TDRE` bit is cleared by reading the `SCInSR1` register and writing a byte into the `SCInDRL` register. Also, all status flags related to reception are cleared by reading the SCI status register following by reading the `SCInDRL` register.
- When the transmitter is not transmitting a frame, the `TxD` output goes to the idle state, logic 1. Same happens if TE is '0'.

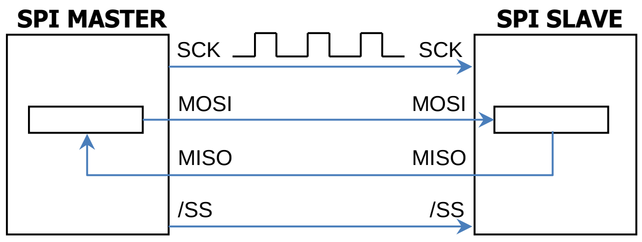
**Character reception:**
- Receive: via the `RxD` pin. When receiving 9-bit data, the MSB in the `SCInDRH` register holds the 9th bit.
- The frame is stored into the receive shift register, and then the data portion of the frame is received into the SCI data register.
- RDRF (Receive data register full) flag: It is set to '1' when the receive byte can be read. IF `RIE` was enabled, an interrupt request is generated.

**Example**: Use SCI1 to transmit an 8-bit character:
Baud rate = 9600, 8 bits, no parity
*Procedure:*
1. Set baud rate to 9600. `SCI1BDH=0x00; SCIBDL=0x9C`
2. Select 8 data bits, no parity. `SCI1CR1 = 0x00`
3. Enable transmitter, interrupt requests disabled: `SCI1CR2 = 0x08`
4. Wait until TDRE=1. If so, write character on SCI1DRL: `while (!(SCI1SR1&0x80)); SCI0DRL = ‘p’`

**Example**: Use SCI1 to read a character using the polling method:
Baud rate = 9600, 8 bits, no parity
*Procedure:*
1. Set baud rate to 9600. `SCI1BDH=0x00; SCIBDL=0x9C`
2. Select 8 data bits, no parity. `SCI1CR1 = 0x00`
3. Enable receiver, interrupt requests disabled: `SCI1CR2 = 0x04`
4. Wait until RDRF = 1, If so, read character from SCI1DRL: `while (!(SCI1SR1&0x20)); cx = SCI1DRL`

## SERIAL PERIPHERAL INTERFACE (SPI)

- It is 4-wire serial bus, sometimes called SSI (synchronous serial interface). It allows synchronous communication (both sides use a common clock to synchronize data) between a processor and peripherals. Each SPI module can operate as a Master or as a Slave. The figure below depicts two SPI modules, one acting as a Master and the other as a Slave. There are four associated pins for each SPI module:
  - ✓ `/SS`: Slave Select
  - ✓ `SCK`: Serial Clock
  - ✓ `MOSI`: Master out/Slave in
  - ✓ `MISO`: Master in/Slave out
- The direction of signals depends on whether the SPI Module is a Master or a Slave.
- Each SPI Module has an 8-bit Shift register. In a Master-Slave configuration, the two shift registers can be viewed as one 16-bit shift register connected by the MISO and MOSI signals.



- HCS12DG256: Three SPI modules (SPI0, SPI1, SPI2).
- The SPI0 module uses the PS4-PS7 pins. PS4: MISO, PS5: MOSI, PS6: SCK, PS7: /SS. The SPI1 uses the PP0-PP3 pins, and the SPI2 module uses the PP4-PP7 pins.

### HCS12D: RELEVANT REGISTERS

- `SPInBR`, n=0,1,2. SPI Baud Rate Register. We can select the baud rate of the SPI module according to this formula:

**SPInBR:**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | SPPR2 | SPPR1 | SPPR0 | 0 | SPR2 | SPR1 | SPR0 |
| | | SPPR | | | | SPR | |

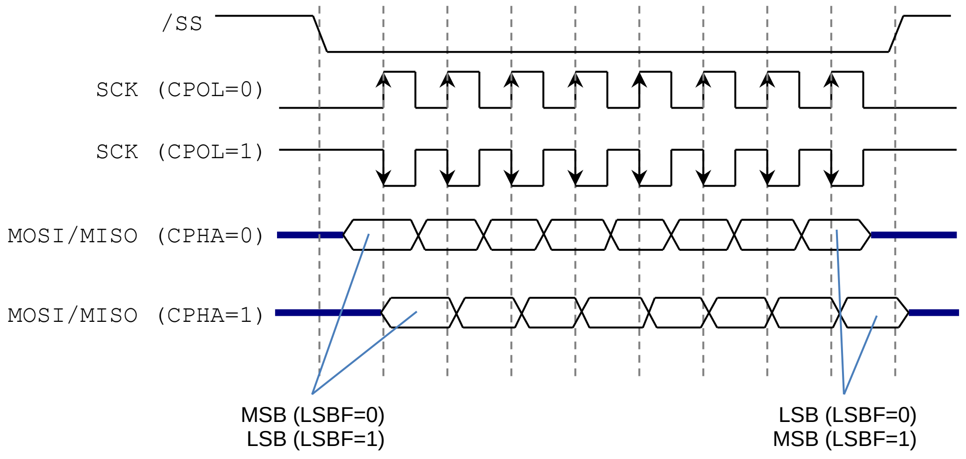
$$Baud\ Rate = \frac{E - clock}{Baud\ Rate\ Divisor}, Baud\ Rate\ Divisor = (SPPR + 1) \times 2^{(SPR+1)}$$

Example: For $6x10^6$ baud rate, we have Baud Rate Divisor = 4. Then SPPR=1, SPR=0, i.e. `SPInBR=0x10`

- `SPInCR1, SPInCR2` (SPI Control Register 1, SPI Control Register 2): They configure interrupt enable, interrupt sources, master/slave mode, SCK polarity/phase, bidirectional/normal mode, which bit (MSB/LSB) goes first, etc.
  - ✓ SPE (Bit 6): SPI Enable bit. It enables (1)/disables(0) the SPI function.
  - ✓ MSTR (Bit 4): It selects whether the SPI Module is configures as a Master (1) or as a Slave (0).
- `SPInDR` (SPI Data Register). It is both the input and output register for SPI data (8 bits). We write data to transmit here. We also read data received here.
- `SPInSR` (SPI Status Register): It records the progress of data transfers and errors.
  - ✓ SPIF (Bit 7) is asserted (and it can generate an Interrupt if SPIE bit in `SPInCR1` is ‘1’) when an SPI Transfer has completed. It is cleared by reading the `SPnSR` (with SPIF=’1’) followed by reading the SPInDR register.
  - ✓ SPTEF (Bit 5) is asserted when there is room in the Transmit Data Buffer; an interrupt is requested if the SPTIE bit in `SPInCR1` is ‘1’. It is cleared by reading the `SPInSR` register (when SPTEF=’1’) followed by writing a data value into the SPI Data Register (`SPInDR`).

## OPERATION

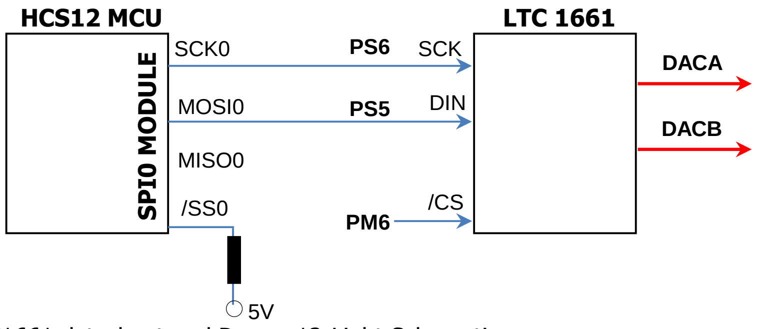
- SCK only works when data is being transmitted or received.
- SPI can work in bidirectional mode and normal mode. We will work with the Normal Mode.
- The CPOL (bit 3) and CPHA (bit 2) bits of `SPInCR1` determine the SCK format. These two bits must be identical in the Master and Slave SPI Modules.



## Circuit connection

- 1 master, 1 slave configuration. Two possible connections
  - ✓ Master generates /SS, which is an input to the Slave. Master controls /SS (it must be zero for any transmission/reception to be valid)
  - ✓ Master has /SS pulled to high. And Slave has /SS=0. This way, Slave is always enabled.
- 1 master, many slaves: Here, we need to generate many /SS signals. This is done by using another port (e.g. PORT P) to generate these signals. The Master /SS is pulled to high.

## USING THE DIGITAL-TO-ANALOG CONVERTER (LTC1661):



- Connections are per the LTC1661 datasheet and Dragon12-Light Schematics.
- 10-bit ADC acts as a SPI0 Slave:
  - ✓ MISO: Not generated.
  - ✓ /SS signal: non-existent
  - ✓ DIN pin is connected to MOSI output from Master.
  - ✓ /CS pin: If '1', SCK is disabled. If '0', SCK enabled and data transfer must start. When it is pulled to '0', SCK must be low initially. Thus, we use CPOL=0 (SCK idle low).
- Data: 16-bit, consisting of 4 control bits, 10 data bits (D9-D0), and 2 don't care bits. As per the timing diagram of the LTC1661, the MSB must be sent first (LSBF=0).
- Analog Voltage: $V_{out} = \left(\frac{D}{1024}\right) \times V_{REF}, V_{REF} = 5v, D = D9 - D0$

**Example:** Generate 1.5v, 3.0v, and 4.5v repeatedly on DACA. Each voltage lasts for 30 ms.

- Configure Baud Rate: 6x10$^6$. Baud Rate divisor = 24/6 = 4. Then SPPR=1, SPR=0. → `SPI0BR = 0x10`
- `SPI0CR1`: `SPI0CR1=0x50`. SPIE=1 (interrupts disabled), SPE=1 (SPI0 enabled), SPTIE=0 (SPTEF interrupt disabled), MSTR = 1 (Master mode), CPOL=0 (SCK low when idle), CPHA=0 (SCK edge appear one-half cycle into the 8-cycle transfer operation), SSOE=0 (with MODFEN=0, the /SS pin is not used by SPI0), LSBF=0(MSB first).
- `SPI0CR2 = 0x02`. MODFEN=0 (with SSOE=0, the /SS pin is not used by SPI0), BIDIROE=0 (output buffer for bidirectional mode disabled), SPSWAI=1 (SPI clock stops in wait mode), SPC0=0 (normal (non-bidirectional) mode)
- `WOMS=0x00`. Since the /SS pin is not used, we need to pull it to high. WOMS=0x00 enables the Port S pull up.

- Inside an infinite loop write the following numbers:
  - ✓ `0x94CF` for 1.5v on DACA. Wait 30 ms
  - ✓ `0x999B` for 3.0v on DACA. Wait 30 ms
  - ✓ `0x9E6B` for 4.5v on DACA. Wait 30 ms

- Sending 16-bit data: We require 2 transfers here. We need to look at the timing diagram of the LTC1661. Everything starts when /CS=0 and finishes when /CS=1 (at this moment, the DAC responds with an analog voltage).
  - ✓ We first enable SPI transfer by setting /CS=0
  - ✓ We transfer one character
  - ✓ We transfer second character
  - ✓ We disable SPI transfer by setting /CS=1.

  - ❖ Operation to send one character
    - Wait until SPTEF (SPI data register empty interrupt flag) is '1'. This is bit 5 of `SPI0SR` (SPI0 status register)
    - Clear SPTEF flag. This is done by reading SPI0SR (**do it explicitly**) and by writing on `SPI0DR`.
    - Write an 8-bit data on data register: SPI0DR
    - Wait until SPIF = 1. SPI interrupt request bit. This bit (bit 7) is 1 when data transfer (8 SCK cycles) is completed.
    - Clear SPIF Flag. This is done by reading SPI0SR (done in while loop) and by reading `SPI0DR`. `temp = SPI0DR`.

**C Code**: `unit10a.c`