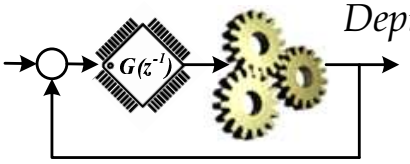
# Chapter 6
# ATmega328 Serial Communication Interfaces

**Assoc. Professor Dr. Rosbi bin Mamat**

rosbi@fke.utm.my



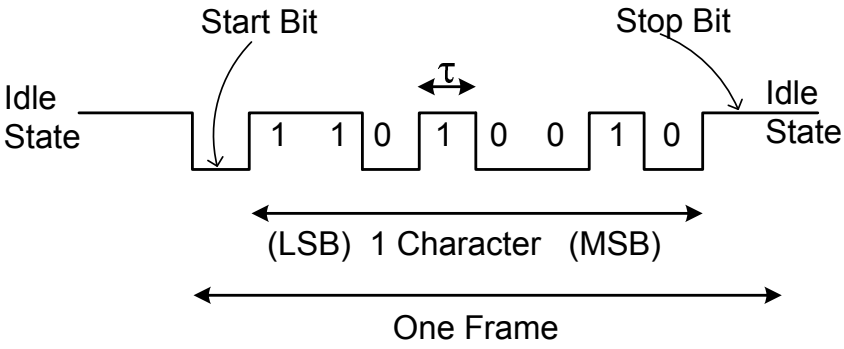

*Dept. of Control & Mechatronics Engineering*
*Faculty of Electrical Engineering*
*Universiti Teknologi Malaysia*

## 6.1 Serial Communication

- Serial data communication involves transmission / reception of data bit-by-bit.
- **Advantage of serial comm.:** Smaller number of comm. lines is required compared to parallel comm.
  - 2 lines (transmit & receive) are required in *asynchronous full duplex* serial comm.
  - 3 lines (transmit, receive & clock) are required in *synchronous* serial comm.
- **Disdvantage of serial comm.:** requires more time to transmit/receive compared to parallel comm.
- ATmega328 has 3 types of serial comm. interfaces:
  1. Universal Synchronous Asynchronous Receiver & Transmitter (USART).
  2. Serial Peripheral Interface (SPI).
  3. Two Wire Interface (TWI).

# 6.2 USART

- Is an *asynchronous* data comm.
- Uses 2 pins on Port D:
  - `TXD/PD1` – the serial data transmission line.
  - `RXD/PD0` – the serial data reception line.
- Data is transmit/receive in a serial frame as follow:






# 6.2 USART

- Each bit is sent with a specific time duration $\tau$, called **bit-time**. The smaller is $\tau$, the faster is data transmission. The rate of data transmission / reception is called the **Baud rate**. Standard Baud rate: 300, 600, 1200, 2400, 4800, 9600, 19200,...

$$Baud\ \ Rate = \frac{1}{\tau}$$

- In the ATmega328, the Baud rate is generated from internal clock. The Baud rates at the transmitter & receptor **must be the same** to avoid comm. error. The baud error should be < ± 2% to avoid comm. error.

# 6.2 USART

- Possible baud rate values for Arduino Uno with 16 MHz clock crystal:

| Baud Rate (bps) | $f_{osc}$ = 16.0000MHz | | | |
|---|---|---|---|---|
| | U2Xn = 0 | | U2Xn = 1 | |
| | UBRRn | Error | UBRRn | Error |
| 2400 | 416 | -0.1% | 832 | 0.0% |
| 4800 | 207 | 0.2% | 416 | -0.1% |
| 9600 | 103 | 0.2% | 207 | 0.2% |
| 14.4k | 68 | 0.6% | 138 | -0.1% |
| 19.2k | 51 | 0.2% | 103 | 0.2% |
| 28.8k | 34 | -0.8% | 68 | 0.6% |
| 38.4k | 25 | 0.2% | 51 | 0.2% |
| 57.6k | 16 | 2.1% | 34 | -0.8% |
| 76.8k | 12 | 0.2% | 25 | 0.2% |
| 115.2k | 8 | -3.5% | 16 | 2.1% |
| 230.4k | 3 | 8.5% | 8 | -3.5% |
| 250k | 3 | 0.0% | 7 | 0.0% |



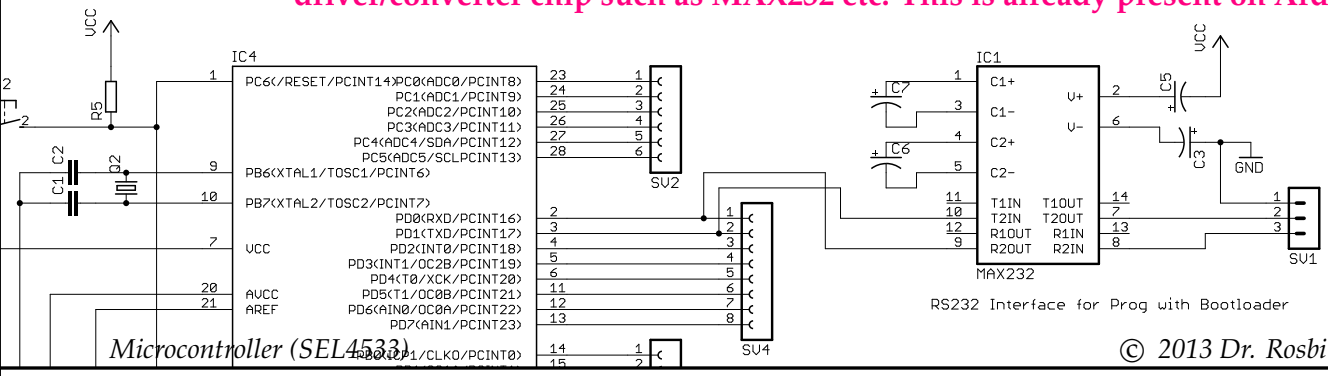
# 6.3 USART – Arduino Libraries

- USART functions can be used with **Serial Monitor** – see Chap 3:
  - `Serial.begin(baud)` – to enable input/output to serial monitor with `baud` speed. Must be written in `setup()`
  - `Serial.available()` – Get the number of bytes (characters) available for reading from the serial port.
  - `Serial.println(val)` – to display `val` value to serial monitor with *newline* added.
  - `Serial.print(val)` – as above but without *newline*.
  - `Serial.print("Error")` – display message "Error" without *newline*.
  - `Serial.read()` – Reads incoming serial data.
  - others functions – refer to `arduino.cc`.

# 6.3 USART – Arduino Libraries

- **Example:** A ATmega328 is to be connected to a PC through the serial port COM with 9600 baud rate. Draw the interfacing circuit & write a program on ATmega328 to do: *When a character '1' is received from USART, send the string "satu" to PC through USART. When a character '2' is received from USART, send the string "rosbi" to PC. Ignore other characters received.*

**Signals on RXD & TXD pins on ATmega328 are at TTL (0 – 5V) level. While the signals at COM1 are at RS232 (+12V or -12V) level. We must use the RS232 driver/converter chip such as MAX232 etc. This is already present on Arduino.**






# 6.3 USART – Arduino Libraries

- **Example:**

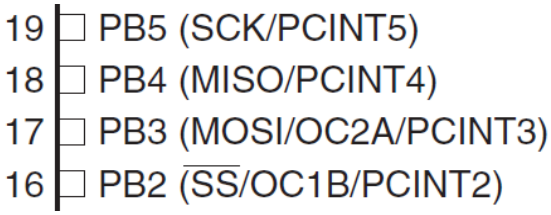
```
void setup() {
  Serial.begin(9600);
}

void loop() {
  char ch;

  if (Serial.available() > 0) { // any char to read?
    ch = Serial.Read();         // read a char
    if (ch == '1')              // check & send
      Serial.print("satu");
    else if (ch == '2')
      Serial.print("rosbi");
  }
}
```

# 6.4 Serial Peripheral Interface (SPI)

- SPI is a **synchronous** data communication.
- SPI uses 4 pins on Port B:

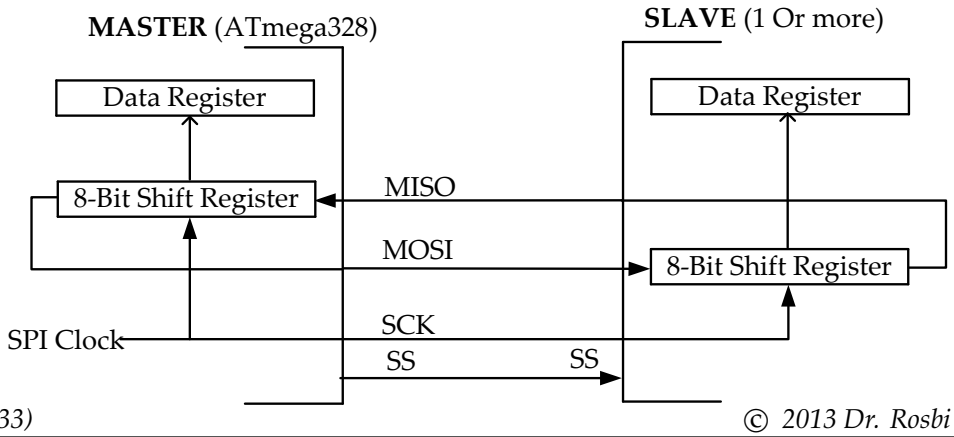



  - `MISO/PB4` – Master In Slave Out, the Slave line for sending data to the master.
  - `MOSI/PB3` – Master Out Slave In, the Master line for sending data to the peripherals (Slave).
  - `SCK /PB5` – The clock pulses which synchronize data transmission generated by the master.
  - `SS /PB2` – The pin on each peripheral that the master can use to enable & disable it.



# 6.4 Serial Peripheral Interface (SPI)

- Connection using SPI is in the Master-Slave configuration.
  - Master – Normally, is the ATmega328. Master **initiate** the data transfer. SPI clock is also generated by master.
  - Slave – Consist of 1 or more SPI I/O peripherals. The slave transfer data as a **reaction** to master.



# 6.5 SPI – Arduino Libraries

- `Spi.begin()` – Initializes the SPI bus by setting `SCK`, `MOSI`, and `SS` to outputs, set `SCK` & `MOSI` low, & SS high. Must be written in `setup()`
- `Spi.end()` – Disables the SPI bus.
- `SPI.setBitOrder(order)` – Sets the order of the bits shifted out of & into the SPI bus, either `LSBFIRST` or `MSBFIRST`.
- `SPI.setClockDivider(divider)` – Sets the SPI clock divider. The default setting is SPI_CLOCK_DIV4, which sets the SPI clock to 4 MHz for Uno.
- `SPI.setDataMode(mode)` – Sets the SPI data mode: clock polarity & phase. Available modes: SPI_MODE0 – SPI_MODE3. refer to `arduino.cc`

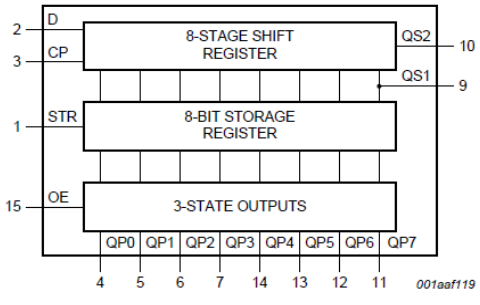


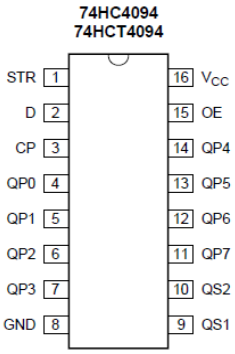
# 6.5 SPI – Arduino Libraries

- `SPI.transfer(val)` – Transfers one byte over the SPI bus, both sending & receiving. `val`: the byte to send out. Returns: the byte read from the bus.
- A few things to consider when connecting SPI peripherals:
  - Is data shifted in MSB or LSB first? This is controlled by the `SPI.setBitOrder()` function.
  - Is the data clock idle when high or low? Are samples on the rising or falling edge of clock? These modes are controlled by the `SPI.setDataMode()` function.
  - What speed is the SPI running at? This is controlled by the `SPI.setClockDivider()` function.

# 6.5 SPI – Arduino Libraries

- **Example:** – *How to interface a 74HC4094 shift register to SPI bus to produce 8-bit port for connecting a 7-segment LED.*



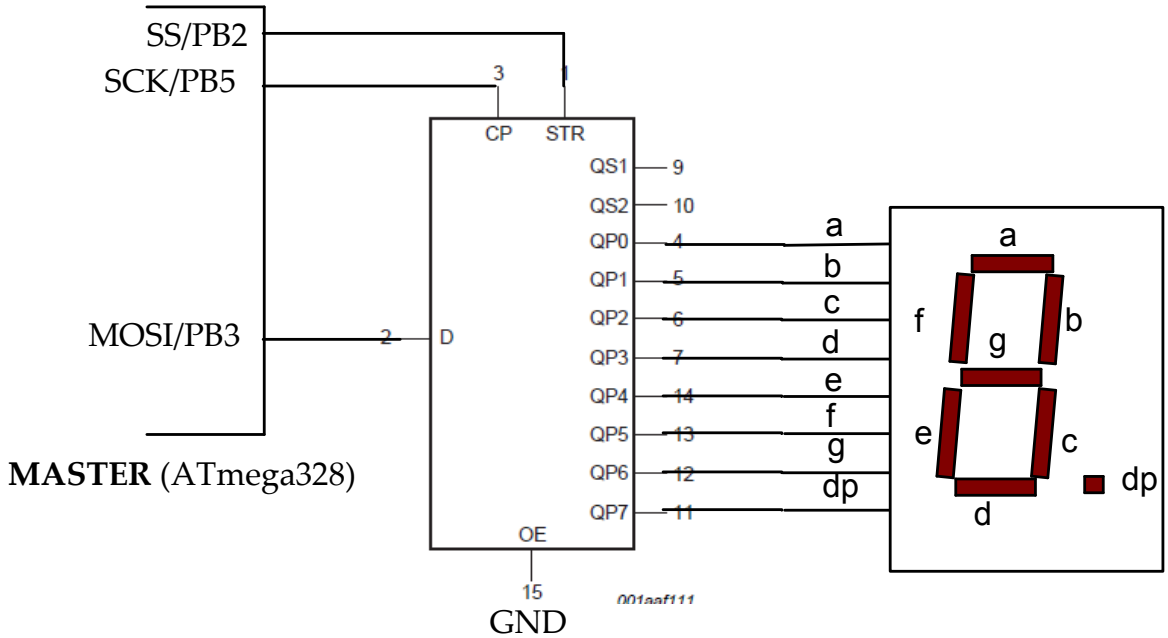




| Symbol | Pin | Description |
|---|---|---|
| STR | 1 | strobe input |
| D | 2 | data input |
| CP | 3 | clock input |
| QP0 to QP7 | 4, 5, 6, 7, 14, 13, 12, 11 | parallel output |
| $V_{SS}$ | 8 | ground supply voltage |
| QS1, QS2 | 9, 10 | serial output |
| OE | 15 | output enable input |
| $V_{DD}$ | 16 | supply voltage |



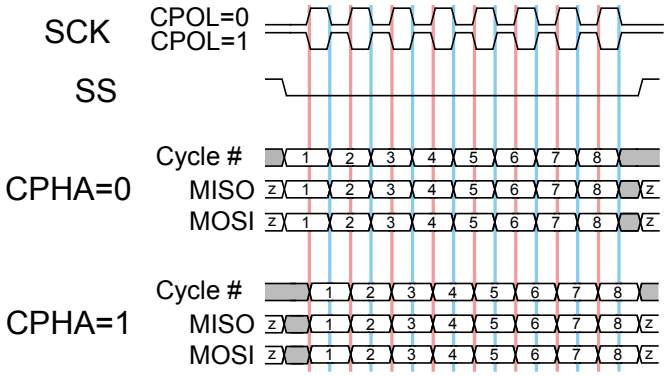
# 6.5 SPI – Arduino Libraries

- **Example:** – *Interfacing 74HC4094 shift register*



# 6.5 SPI – Arduino Libraries

- **Example:** – Some considerations on clock phase (whether data is shifted on the rising or falling edge of clock signal), clock polarity (whether the clock is idle when high or low):




| Mode | Clock Polarity (CPOL) | Clock Phase (CPHA) |
|---|---|---|
| SPI_MODE0 | 0 | 0 |
| SPI_MODE1 | 0 | 1 |
| SPI_MODE2 | 1 | 0 |
| SPI_MODE3 | 1 | 1 |



# 6.5 SPI – Arduino Libraries

- **Example:** – Some considerations:
  - We want the `CPHA=0`, `CPOL=0` (SPI_MODE0) – `SPI.setDataMode(SPI_MODE0)`.
  - We want the clock about 1MHz – `SPI.setClockDivider(SPI_CLOCK_DIV16)`.
  - We want to send MSB first – `SPI.setBitOrder(MSBFIRST)`.

# 6.5 SPI – Arduino Libraries

- **Example:** – The program:

```
/* Display 'rOSbi' on 7-seg LED through SPI */

const unsigned char myname[]= {0xAF, 0xA3, 0x92, 0x83, 0xCF};

void setup() {
  SPI.begin();
  SPI.setBitOrder(MSBFIRST);
  SPI.setClockDivider(SPI_CLOCK_DIV16);
  SPI.setDataMode(SPI_MODE0);
}

void loop() {
  unsigned char ch;
  int j;

  for (j = 0; j < 5; j++)  {     // get 5 char 1-by-1
    ch = myname[j];              // get 1 char
    SPI.transfer(ch);           // display the char
    delay(1000);                // hold the display 1 sec
  }
}
```