COMMUNICATION

DATA TRANSFER.

Knowledge of data transfer is very important for any embedded system developer. In any embed ed system data is moved between several units like between RAM and CPU. There are many methods and technique for data transfer each having its own pros and cons. So different data transfer technique is used in different situations. Some example of data transfer are

- ***Simple parallel transfer***-Used to transfer 8,16,32 ... bits of data in the same time.
- ***Asynchronous Serial Transfer (USART)*** –It is an old but still in use mode of serial communication uses only 2 lines (+1 additional line for GND).
- ***SPI*** - *Serial Peripheral Interface* - It is a standard mode of communication between different ICs.
- ***USB*** **-** A very advance, high speed and complicated serial Bus used in PCs to connect almost anything to it.

CLASSIFICATION

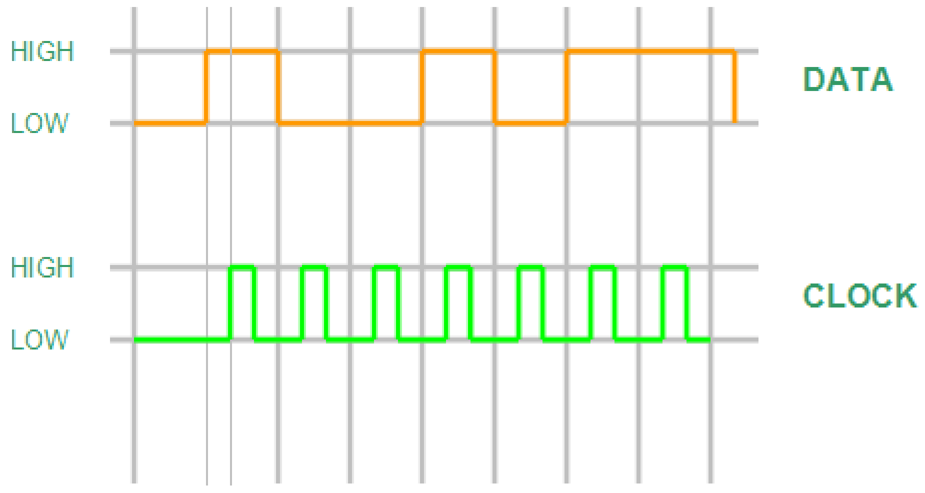
Modes of Data Transfer can be broadly divided into two types:

1. PARALLEL TRANSFER - In this mode a number of bits (say 8,16 or 32) are transferred at a time. Thus they require as many electrical line as the number of bits to be transferred at once. This method is fast but its disadvantage is that it uses more number of lines. So they are basically used when the units involved in data transfer are physically close and almost fixed with each other for a long time. For eg. CPU and RAM, the PCI Cards inside the PC. These are close to each other and packed inside the CPU box and are disconnected from each other less frequently.
2. SERIAL TRANSFER - In this mode only one bit is transfer at once. So to transfer 8 bits, 8 cycles are required. So these require less number of physical lines (like SPI use 3 lines). The advantage is that due to less number ICs using these technologies are small with low PIN count.

Modes of Data Transfer can also be divided into

1. SYNCHRONOUS TRANSMISSION-. In this type the actual data is transferred BIT by BIT on the DATA line. The clock line signals the end of 1 bit and the start of another bit. When the clock line changes its level, that is when it goes HIGH from a LOW level or vice versa the data is transferred. When the CLOCK line goes HIGH it signals that a new bit is available for transfer.
   The "other" device which is receiving the data reads the data line at the rising edge or the falling edge of the clock depending upon our settings .

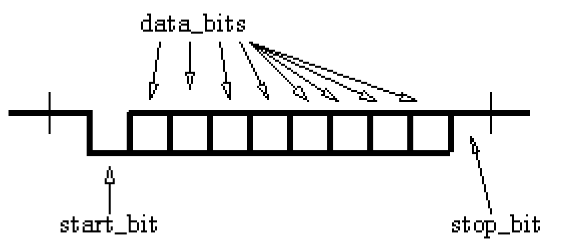
The diagram corresponds to the transfer of the data 10010111. It corresponds to the value of the data at every rising edge of the clock.

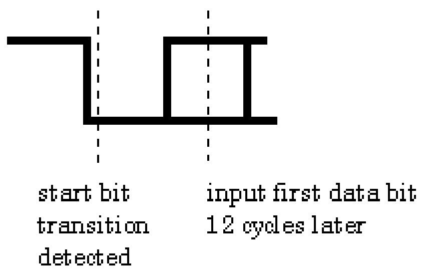


2. ASYNCHRONOUS TRANSMISSION - Asynchronous transmission allows data to be transmitted without the sender having to send a clock signal to the receiver. Instead, the sender and receiver must agree on timing parameters in advance and special bits are added to each word which are used to synchronize the sending and receiving units. When a word is given for Asynchronous transmissions, a bit called the "Start Bit" is added to the beginning of each word that is to be transmitted. The Start Bit is used to alert the receiver that a word of data is about to be sent, and to force the clock in the receiver into synchronization with the clock in the transmitter. One solution is to have both devices share the same clock source.

Baud Rate

Baud Rate is a measurement of transmission speed in asynchronous communication. The devices that allows communication must all agree on a single speed of information - 'bits per second'.




start bit transition detected
input first data bit 12 cycles later

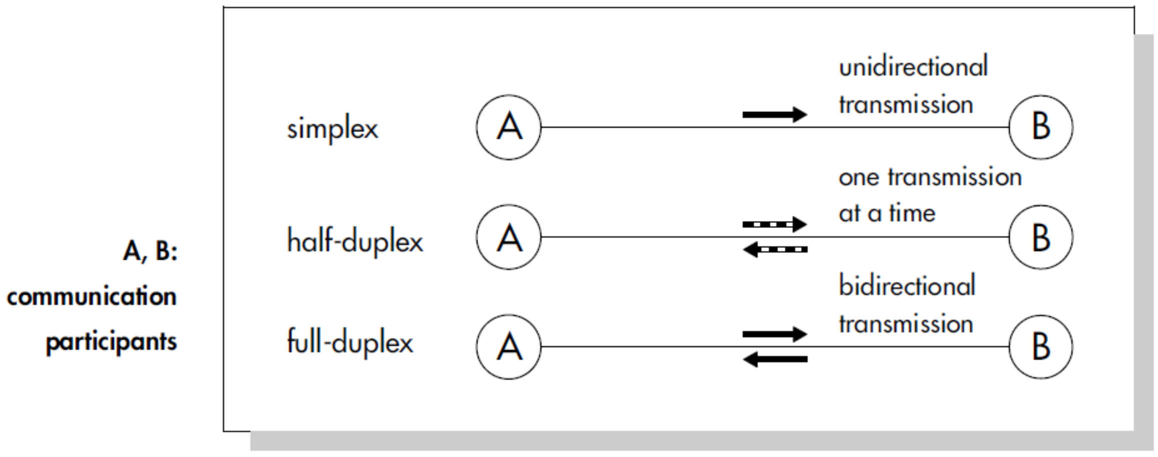
When the entire data word has been sent, the transmitter may add a Parity Bit that the transmitter generates. The Parity Bit may be used by the receiver to perform simple error checking. Then at least one Stop Bit is sent by the transmitter.

When the receiver has received all of the bits in the data word, it may check for the Parity Bits (both sender and receiver must agree on whether a Parity Bit is to be used), and then the receiver looks for a Stop Bit.

In short, asynchronous data is 'self synchronizing'.

Transmission	*Advantages*	*Disadvantages*
Asynchronous	Simple & Inexpensive	High Overhead
Synchronous	Efficient	Complex and Expensive

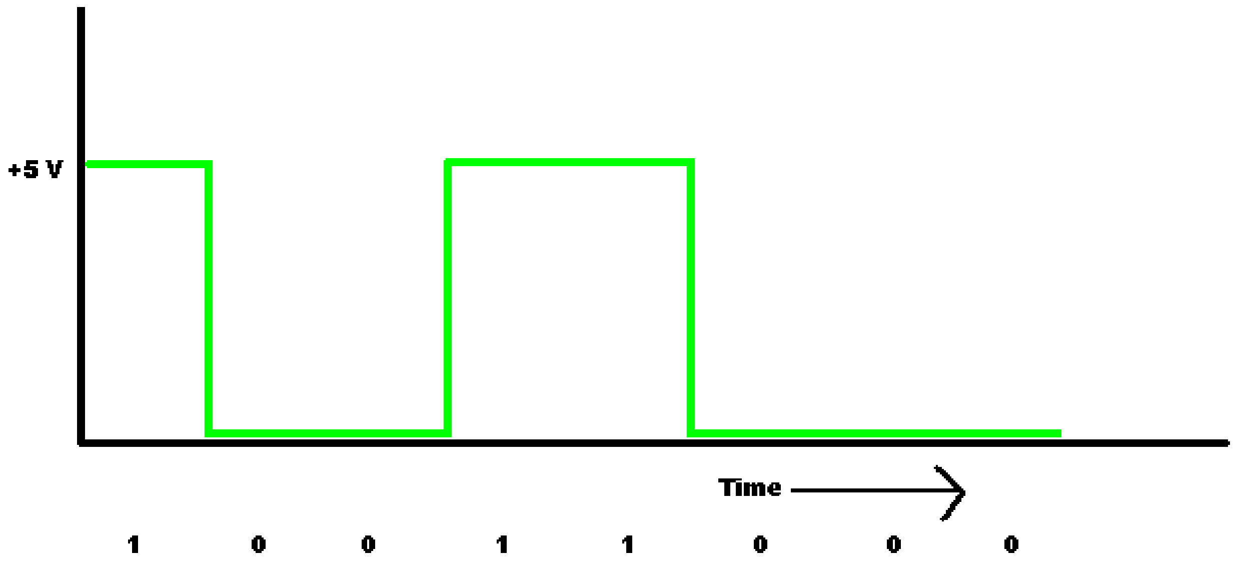
DIFFERENT COMMUNICATION TECHNIQUES



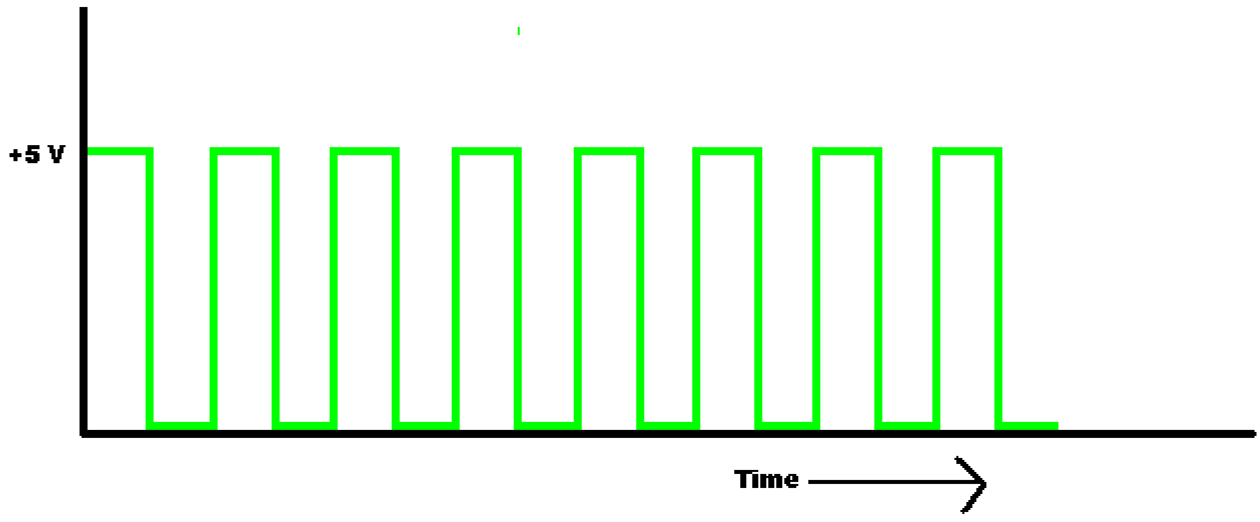
SERIAL PERIPHERAL INTERFACING (SPI)

In SPI, data is transmitted serially, i.e. bit by bit as opposed to parallel communication where all the data is sent multiple bits at a time. We will study synchronous SPI, where there is a clock generated and the data is transferred at the rate of the clock pulse. What is clock pulse?

Clock pulse is basically a sequence of alternating 0s and 1s that is used to indicate that one Bit of data has been sent. For example, if you were to send the data 10011000, the signal you sent would look like:



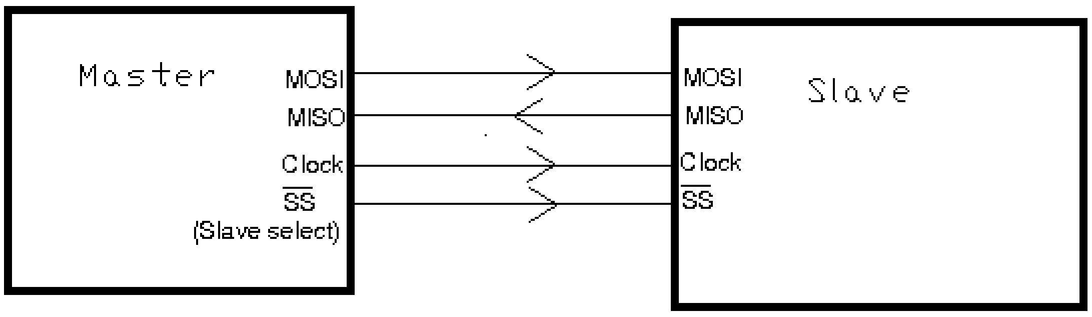
Now as it might be clear there should be some way to tell this signal from 1010 of 100010 of some other. For this, we can do either of two things, set a standard that at a particular rate, the data will be transferred and we keep checking for voltage at time moments T, 2T, 3T so on, or we also send a clock pulse at the same time. Like we decide that whenever we send a new signal, we will make the clock 1. Then the clock signal would look like:



Now if the receiving machine reads the incoming data at negative edges, it will always read what the sender actually meant. This the concept of synchronous data transfers. The sender and the receiver are synchronized by a clock pulse.

Now, in SPI set up when we have to set up communication between two systems, first we have make one system as master and the other as slave. The difference between master and the slave is that the clock pulse is generated by the master and both master and the slave agree to work on the clock frequency that is set up by the master.

The basic connections in any SPI set up are as follows:



MOSI is **M**aster **O**ut **S**lave **I**n, so it is the channel where the master sends the data to the slave and the slave receives it.

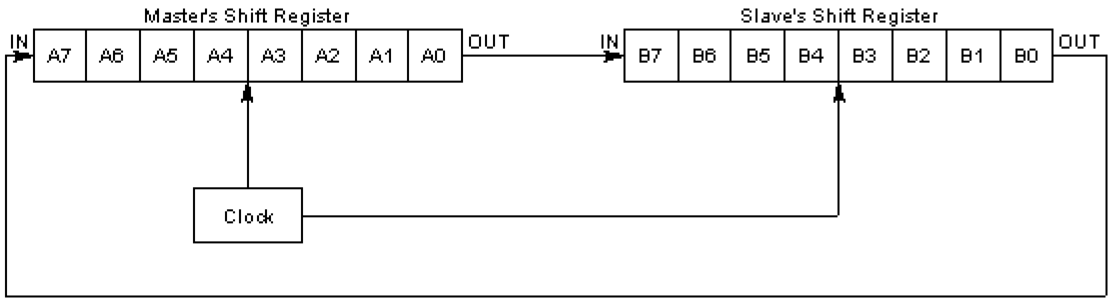
MISO is **M**aster **I**n **S**lave **O**ut**,** so it is the channel where the slave sends the data and the master receives it.

Clock is the clock that is send by the master.

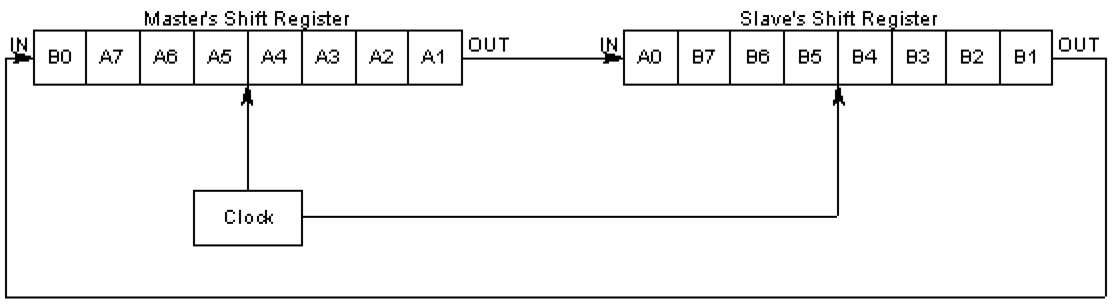
$\overline{SS}$ is slave select when the master wants to send data to a particular slave it makes $\overline{\text{its}}$ SS pin low, sends the data and then again makes it high. It is especially useful when multiple slaves are connected to the master, but the basic purpose is to select the slave and transmitting data to it.

In SPI, the data the data is exchanged between the transmitter and the receiver. It happens as follows:

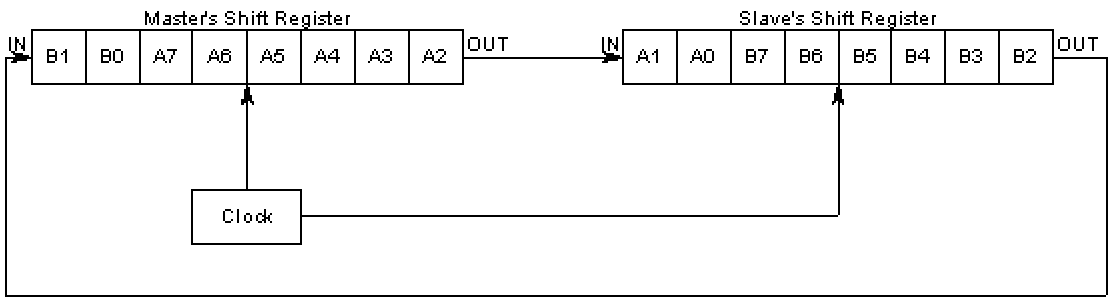
First the master and the slave keep the data bits to transfer in their respective registers. Suppose master wants to send b_{m1},b_{m2},b_{m3}.....b_{m8} and the slave wants to send bs1, b_{s2}, b_{s3} .....$b_{s8.}$ What happens is as follows:



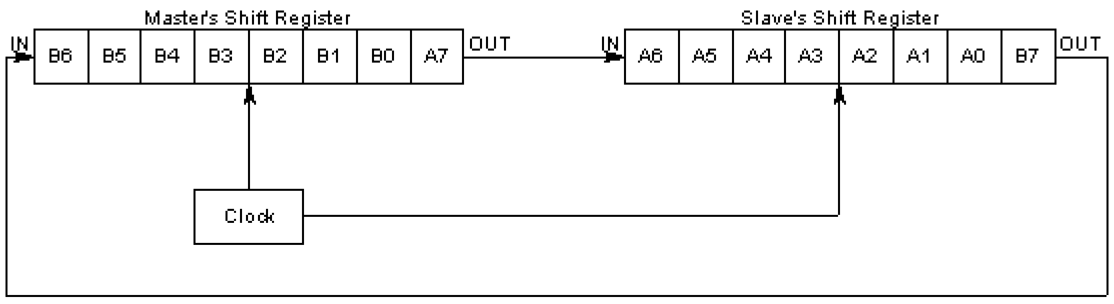
Master generates the first clock pulse:



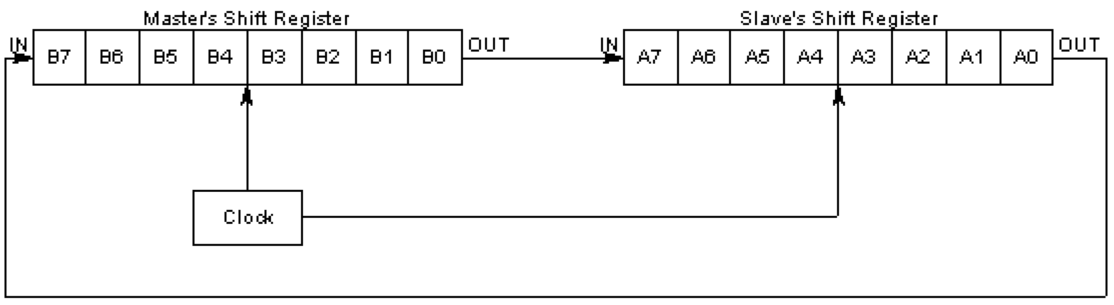
Master generates the second clock pulse:



Master generates the seventh clock pulse:



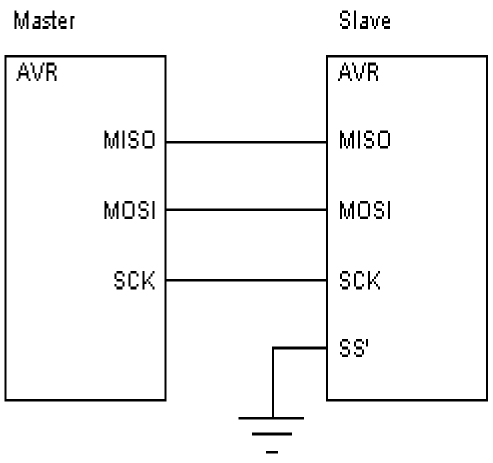
Master generates the last clock pulse:



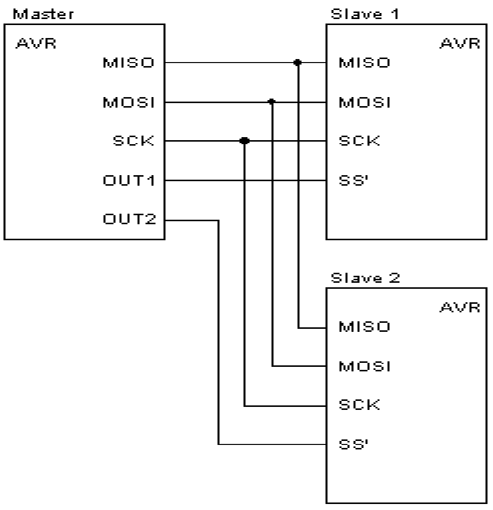
The data is transferred one bit at a time between the master and the slave, and at the end of 8 cycles the data is completely exchanged.

The following figures show a typical setup used with SPI:

One master and one slave

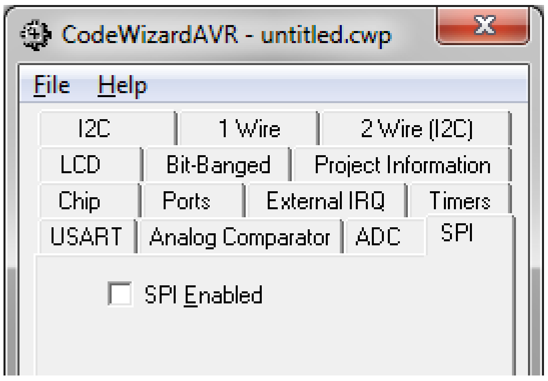


One master and two slaves



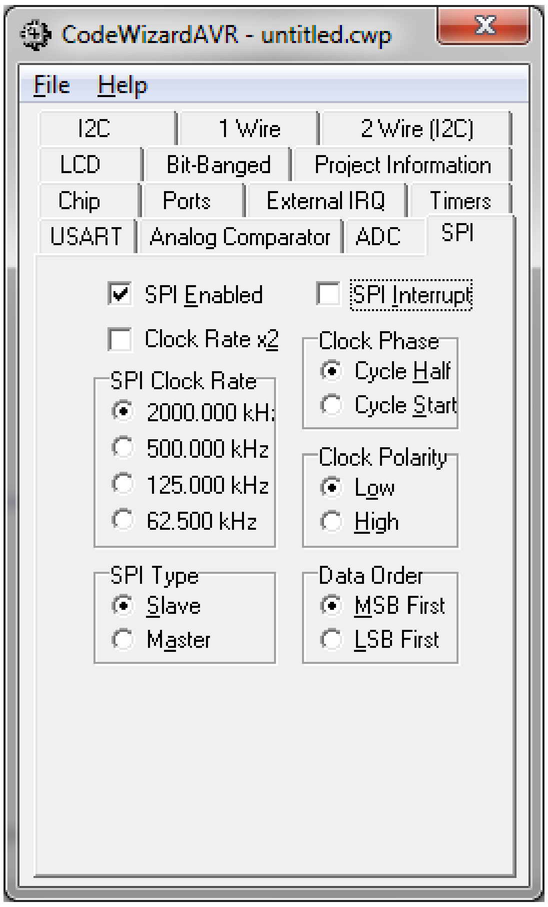
HOW TO USE CODE WIZARD TO SET UP SPI CONNECTION

Select SPI Tab in CV AVR.



☐ SPI Enabled

Enable SPI.



[✓] SPI Enabled
[ ] SPI Interrupt
[ ] Clock Rate x2

SPI Clock Rate: [✓ 2000.000 kH:] 500.000 kHz 125.000 kHz 62.500 kHz

Clock Phase: [✓ Cycle Half] Cycle Start

Clock Polarity: [✓ Low] High

SPI Type: [✓ Slave] Master

Data Order: [✓ MSB First] LSB First

Select the clock rate, and decide whether your mcu is to be master or slave. Leave other settings to default.

SPI Interrupt: It is generated when one byte is transferred between the master and the slave.

IMPORTANT: These settings should be the same for both the devices which are communicating with each other.

CONNECTING MCU TO ANOTHER MCU

Just connect the MOSI, MISO, SCK, and $\overline{\text{SS}}$ to $\overline{\text{SS}}$. Use the following function to send data: spi(character)

When you use this function in the master, it writes the character to the register and sends the data to slave. In case of the slave, the data is written to the register and the cpu waits for the master to send data when it also transmits the data written into the register.

SAMPLE PROGRAMME:

MASTER:	SLAVE:
char a=spi(0xFF);	char b=spi(‘1’);
lcd_putchar(a);//displays 1 on the lcd	

USART

Like many microcontrollers, AVR also has a dedicated hardware for serial communication. This part is called the USART - Universal Synchronous Asynchronous Receiver Transmitter. This special hardware makes your life as a programmer easier. You just have to supply the data you need to transmit and it will do the rest. The advantage of hardware USART is that you just need to write the data to one of the registers of USART and your done, you are free to do other things while USART is transmitting the byte.

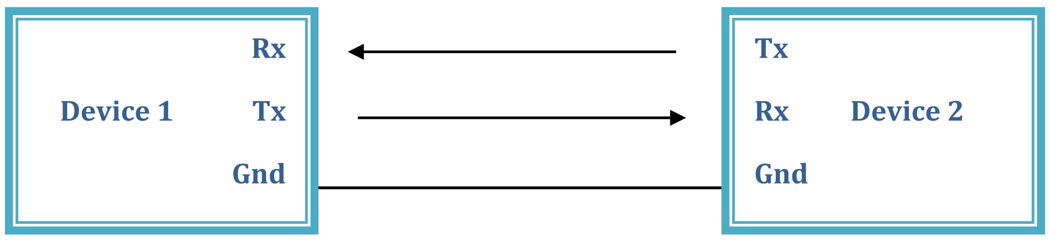
Also the USART automatically senses the start of transmission of RX line and then inputs the whole byte and when it has the byte it informs you (CPU) to read that data from one of its registers.

The USART of AVR is very versatile and can be setup for various different modes as required by your application. In this tutorial we will show you how to configure the USART in a most common configuration and simply send and receive data.

HARDWARE ASPECT OF USART

USART consists of only three connections – Rx, Tx and GND. Rx means Receive and Tx means Transmit. The GND connection is for a common reference level.

Here's a simple diagram explaining the connections:



Notice how Tx is connected to Rx, and Rx is connected to Tx.

BAUD RATE

Baud is a measurement of transmission speed in asynchronous communication. The computer, any adaptors, and the UART must all agree on a single speed of information - 'bits per second'.

DATA TRANSMISSION

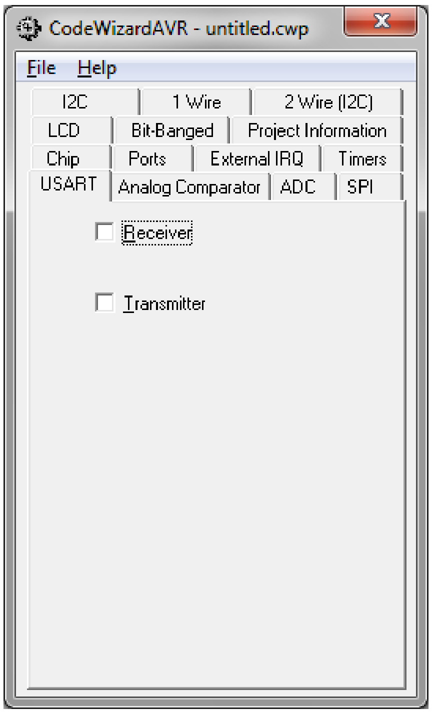
In Asynchronous mode, data is transmitted in frames. Each frame has a start bit, data bits, optional parity bit, and stop bits.

A start bit signals the beginning of data transmission. Data bits are the actual data to be transmitted. Stop bits signal the end of transmission.

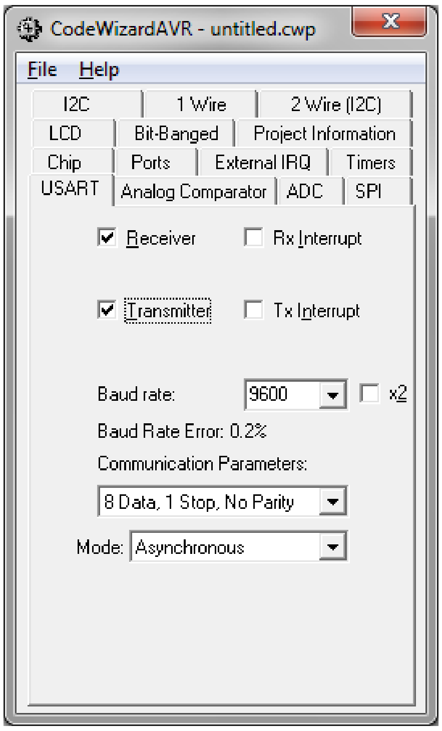
An optional parity bit can be transmitted before the stop bits. This bit represents the number of "logical highs" in the transmission. If there are odd numbers of logical highs in the transmission, then the parity bit has a logical high value. If there are even numbers, then the parity bit takes the value logical low. Parity bits are generally used in error detection.

SETTING UP USART IN CODEVISIONAVR

In CodeWizardAVR, select the USART tab to setup the USART.



As you can see, you can set the USART to either receive, transmit, or both. Checking the appropriate checkboxes will enable more options for configuration.



[✓] Receiver [ ] Rx Interrupt

[✓] Transmitter [ ] Tx Interrupt

Baud rate: 9600 [ ] x2

Baud Rate Error: 0.2%

Communication Parameters:

8 Data, 1 Stop, No Parity

Mode: Asynchronous

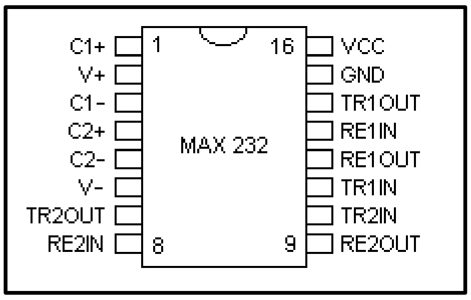
- Rx Interrupt and Tx Interrupt are special interrupts that are called every time data is received / transmitted. Selecting either option will enable more options - leave them to their default values.
- Baud rate will let you select the transmission rate in bps. The line below the option will tell you the error rate – do not set the baud rate so high that the error is large and the line becomes red.
- Communication Parameters will let you set other parameters – leave them to their defaults for most cases.

IMPORTANT: These settings should be the same for both the devices which are communicating with each other.

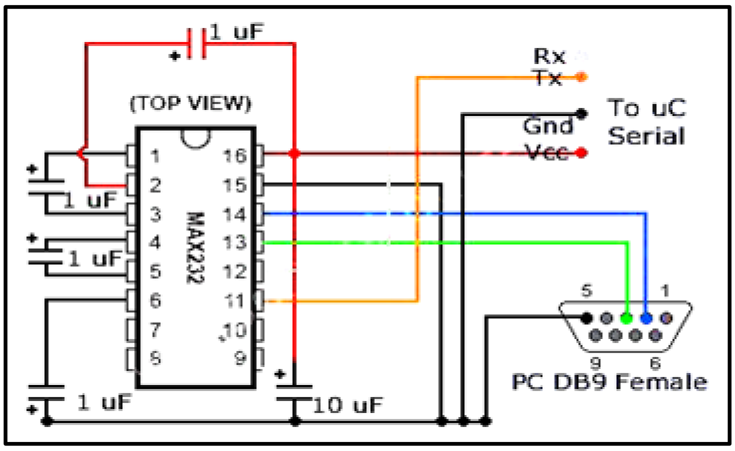
CONNECTING WITH A COMPUTER:

USART connection with a computer is accomplished through a protocol called RS232. RS232 is an asynchronous serial communication protocol widely used in computers and digital systems. A simple example is the serial port used in old computers.

One thing to note about this protocol is that, while in an MCU circuit, HIGH = 5V and LOW = 0, for RS232 the values are +12V and -12V respectively. For this conversion, an IC called MAX232 is used:



The schematics required are:



IMPLEMENTING USART IN YOUR CODE

Implementing USART is easy – all you need are two functions:

1. `putchar(char);`

   This function will allow you to transmit data through the USART interface. The argument is a character; you can transmit the ASCII code of the character in hexadecimal form. The transmitted data is stored in a special register in the device which is receiving this data.

2. `getchar();`

   This function reads data from the special register reserved for USART communication. This function will stall the program while waiting for the data to be transmitted and stored in the register if it does not already exist.

SAMPLE PROGRAM

Input MCU

```
// a is a char variable

a = inputFromUser();

putchar(a);
```

LCD MCU

```
a = getchar();

// Program will wait for data

// Data transmitted, now print

printChar(a);
```