BITS Pilani presentation

BITS Pilani
Pilani Campus

Ramani Kalpathi
Automotive Electronics

innovate achieve lead

BITS Pilani

Pilani Campus

AEL ZG512 Embedded Systems
Lecture No. 5

Embedded Architecture 2 – ARM Based Controller

innovate achieve lead

- Communication Peripherals- Synchronous & Asynchronous
  - I2C
  - SPI
  - UART
  - CAN

- Introduction to ARM Cortex Architectures
  - ARM Cortex-M Architecture

Board Design - System Booting related Concepts

Communication Protocols

I2C Communication

I2C – Inter Integrated Circuit

Three I2C modules in the LPC23xx devices

Operation at 400Kbps speed

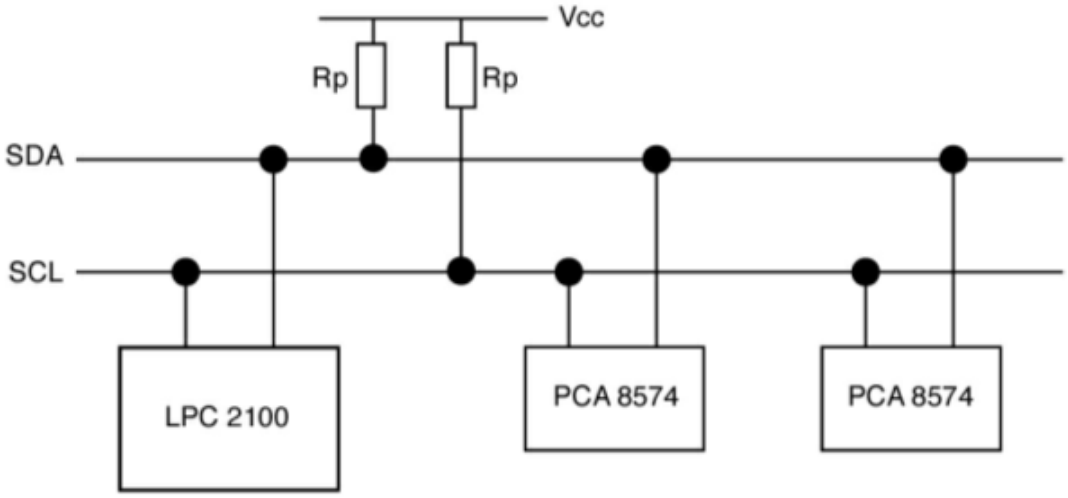
Master mode or slave mode

SDA – Serial data

SCL – Serial Clock

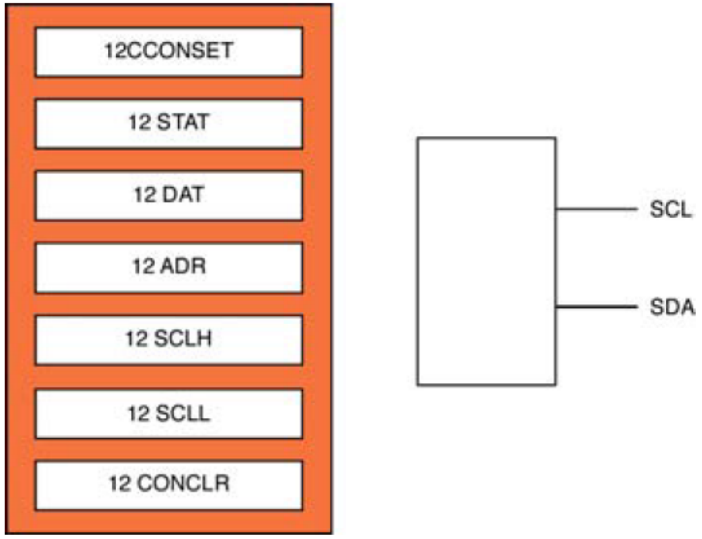
Two pins are selected to be changed from GPIO to I2C mode

I2C Bus Interface



Typical I2C bus configuration. The bus consists of separate clock and data lines with a pull up resistor on each line. The two external devices used in the example are port expander chips.

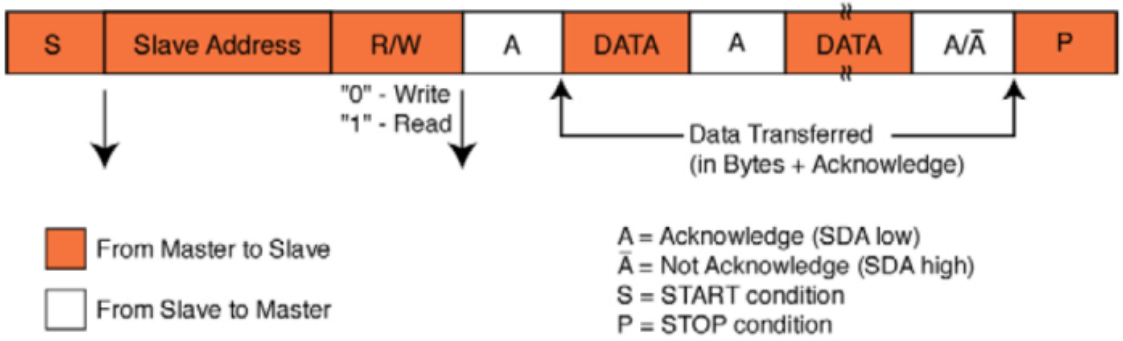
Special Function Registers



I2C peripheral registers.

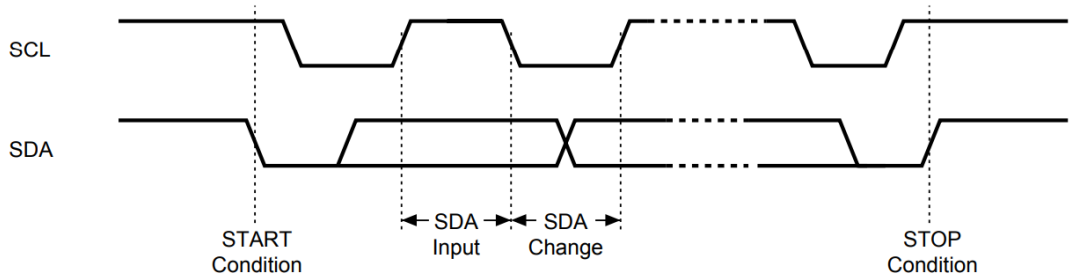
The programmers' interface includes two timing registers: set and clear registers for the control register, an address register to hold the node address when in slave mode and a data register to send and receive bytes of data .

I2C Transaction



Typical I2C transaction :A I2C bus transaction is characterised by a start condition, slave address data exchange and stop condition with acknowledge handshaking.

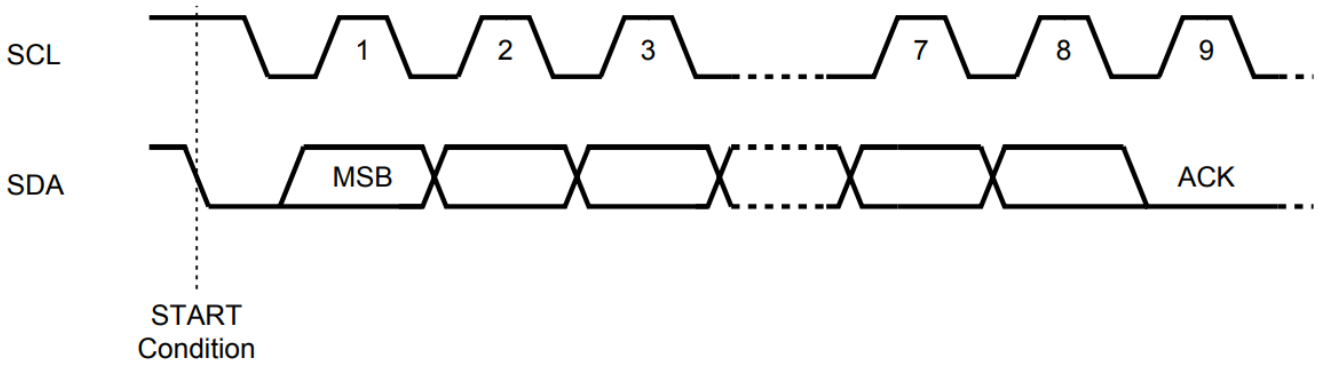
I2C Start – Stop Waveform



Start Condition: With SCL high, SDA should transition from High to Low

Stop Condition: With SCL high, SDA should transition from Low to High

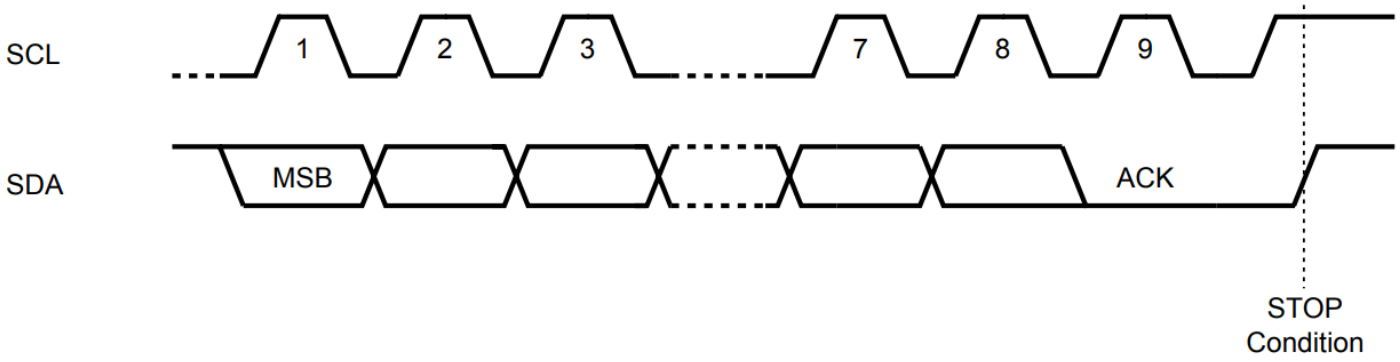
Data Transfer after Start



After Start condition, 7-bit address is transferred on SDA on every rising edge of SCL

The eighth bit is high if it is a Read operation
The eighth bit is low if it is a Write operation
For a write operation, the 9th bit is an acknowledge bit from the slave
Note that the SDA is therefore bi-directional

Completion of data Transfer



Once data transfer is completed, the Master issues a STOP condition after the last ACK bit is received

Typical Data Transfer Sequence
Write data to one address of EEPROM


```
I2C_Init(100000); // Configure the peripheral for I2C
I2C_Start(); // Issue I2C start signal
I2C_Wr(0xA2); // Send slave address + Write bit for 24cO2 device
I2C_Wr(2); // Send byte (address of EEPROM location to write)
I2C_Wr(0xF0); // Send data (data to be written)
I2C_Stop();
```

Configure the I2C peripheral

Issue Start sequence

Send 7 bit Device Slave address + low bit for data write

Send address of device memory where data needs to be written

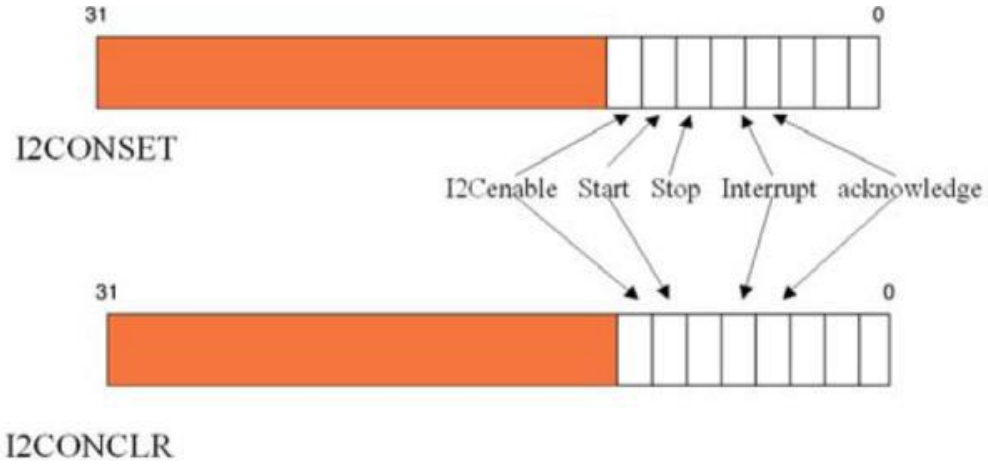
Send data that needs to be loaded into that particular address in EEPROM

Data Transfer - Read sequence

I2C_Init(100000); // Configure the peripheral for I2C
I2C_Start(); *// Issue I2C start signal*
I2C_Wr(0xA2); *// Send byte via I2C (device address + W)*
I2C_Wr(2); *// Send byte (data address)*
I2C_Repeated_Start();*//Issue I2C signal repeated start*
I2C_Wr(0xA3); *// Send byte (device address + R)*
value = I2C_Rd(0);*// Read the data (NO acknowledge)*
I2C_Stop();

Configure the I2C peripheral, Issue Start sequence
Send 7 bit Device Slave address + low bit for data write
Send address of device memory where data needs to be read
Send read sequence and store the data in a variable value

I2C Control on LPC23XX Devices



I2C control registers: The control registers are used to enable the I2C peripheral and interrupt as well as controlling the I2C bus start, stop and ack conditions.

Control Register for Start, Stop, enable, acknowledge and interrupt
For I2C Master mode, the I2C peripheral must be enabled and acknowledge bit must be set to 0.

Coding for Byte Transfer


```
void I2CTransferByte(unsigned Addr,unsigned Data)
{

   I2CAddress = Addr;       // Place address and data in Globals to be used by
                            // the interrupt
   I2CData = Data;
   I2CONCLR = 0x000000FF;   // Clear all I2C settings
   I2CONSET = 0x00000040;   // Enable the I2C interface
   I2CONSET = 0x00000020;   // Start condition
}
```

I2C Clock stretching

If the slave is not ready to accept data it can delay the clock by pulling the clock line low and stretching the clock thereby delaying data transfer.

If the master finds the clock line not high the data transfer is delayed. Master generates the clock for any transfer.

SPI Communication

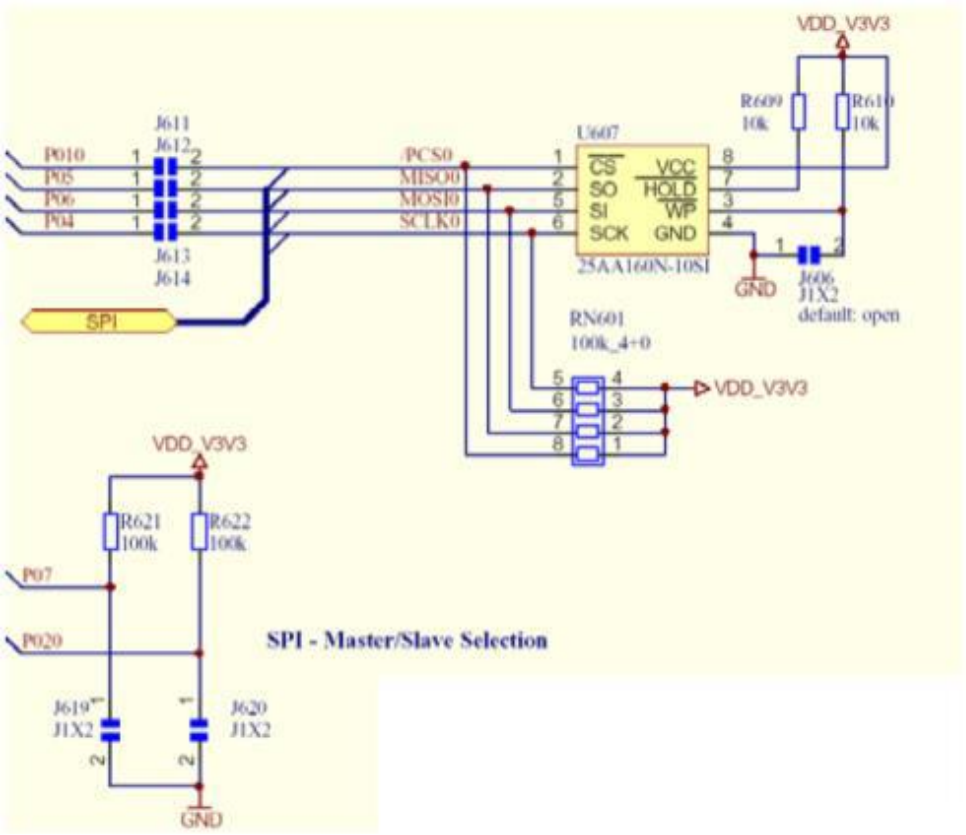
Serial Peripheral Interface is characterized by the following Interface lines to a device from the controller.

- Serial clock
- Serial Data Output
- Serial Data Input
- Slave Select

Master generates the clock, Data can flow bi-directional and simultaneous between master and slave

Slave select signal is used to enable data transfer using a hardware signal

SPI Communication

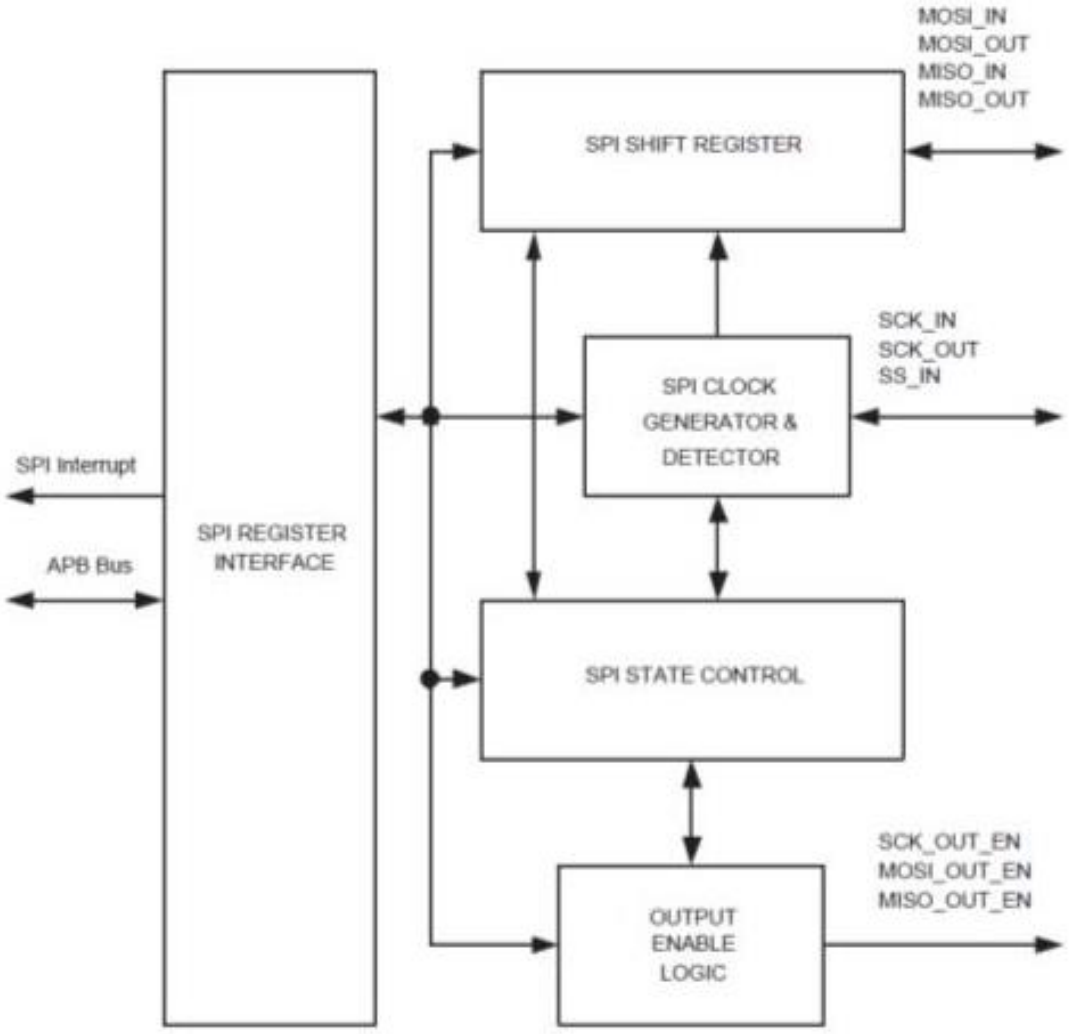


SPI EEROM peripheral: This diagram shows how to interface an external EEROM onto the SPI bus of the LPC2000. It should be noted that pins P0.7 and P0.20 must be pulled high to enable the SPI peripheral as a master.

SPI Registers

Name	Description	Access	Reset Value[1]	Address
S0SPCR	SPI Control Register. This register controls the operation of the SPI.	R/W	0x00	0xE002 0000
S0SPSR	SPI Status Register. This register shows the status of the SPI.	RO	0x00	0xE002 0004
S0SPDR	SPI Data Register. This bi-directional register provides the transmit and receive data for the SPI. Transmit data is provided to the SPI0 by writing to this register. Data received by the SPI0 can be read from this register.	R/W	0x00	0xE002 0008
S0SPCCR	SPI Clock Counter Register. This register controls the frequency of a master's SCK0.	R/W	0x00	0xE002 000C
S0SPINT	SPI Interrupt Flag. This register contains the interrupt flag for the SPI interface.	R/W	0x00	0xE002 001C

SPI Block Diagram



SPI Data Transfer mechanism: Master Mode

1. Set SPI clock counter register to desired clock rate
2. Program the SPI control register to the desired settings
3. Write the data to be transmitted to the SPI data register. The SPI transfer will be started.
4. Wait for the SPIF bit in the SPI status register to be set to 1 after completion of data transfer.
5. Read the SPI status register.
6. Read the received data from the SPI data register if required.
7. Go to step 3 if more data needs to be sent out.

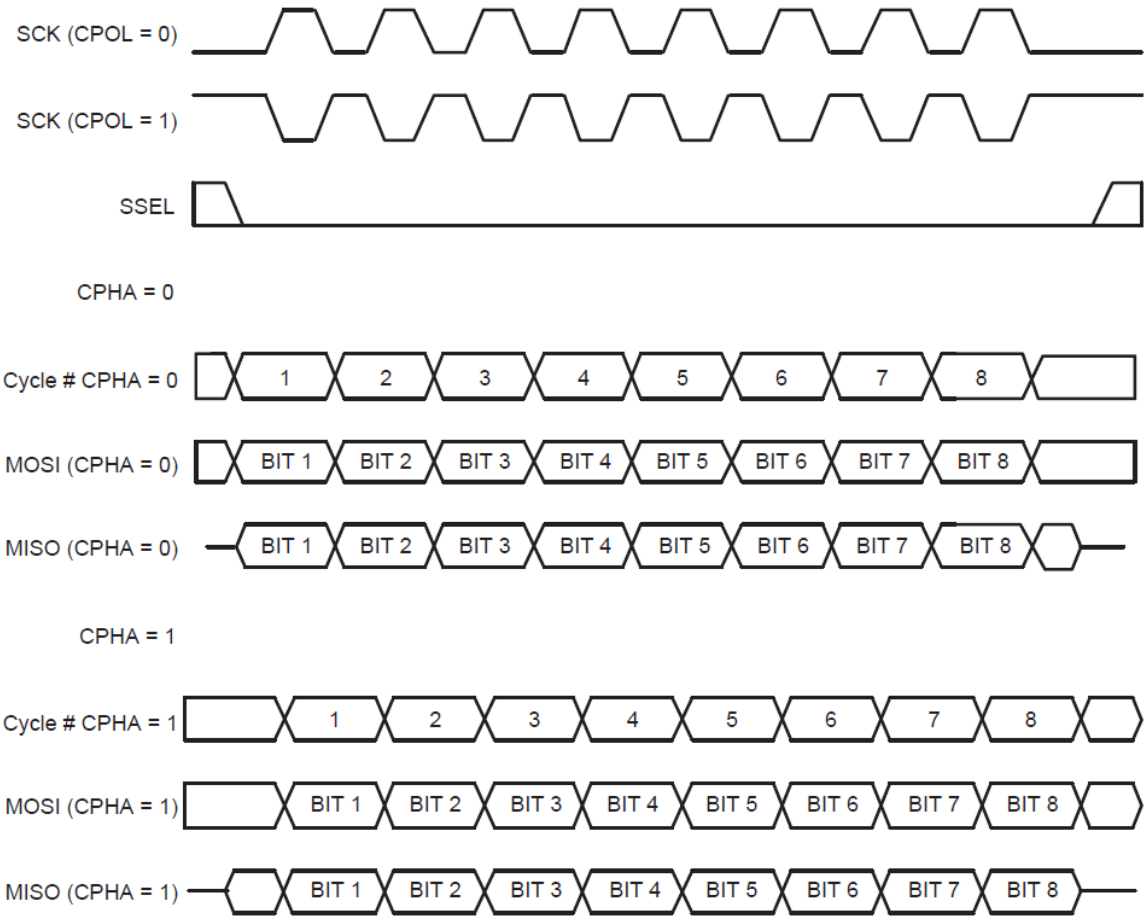
SPI Data Transfer mechanism: Slave Mode

1. Program the SPI control register to the desired settings.
2. Write the data to be transmitted to the SPI data register.
3. Wait for the SPIF bit in the SPI status register to be set to
   1. The SPIF bit will be set after the last

sampling clock edge of the SPI data transfer.

4. Read the SPI status register.
5. Read the received data from the SPI data register which is optional.
6. Go to Step2 if more data needs to be sent.

SPI-Waveforms



SPI Transfer Sequence

- Slave needs to have a low signal for Slave Select
- Clock is generated by the Master
- For every clock cycle one bit of data is transferred from Master Output to Slave Input
- For every clock cycle one bit of data is transferred from Slave Output to Master input

The data transfer on the rising edge or falling edge or the phase of the clock are all selectable

SPI code using Library routines

```
void main()
{ Spi_Init(); // initialize spi
  Spi_Lcd_Init(); // initialize lcd over spi interface
  Spi_Lcd_Cmd(LCD_CLEAR); // Clear display
  Spi_Lcd_Cmd(LCD_CURSOR_OFF); // Turn cursor off
  Spi_Lcd_Out(1,6, “Welcome"); // Print text to LCD, 1st
    row, 7th column
  Spi_Lcd_Chr_CP('!'); // append !
  }
```

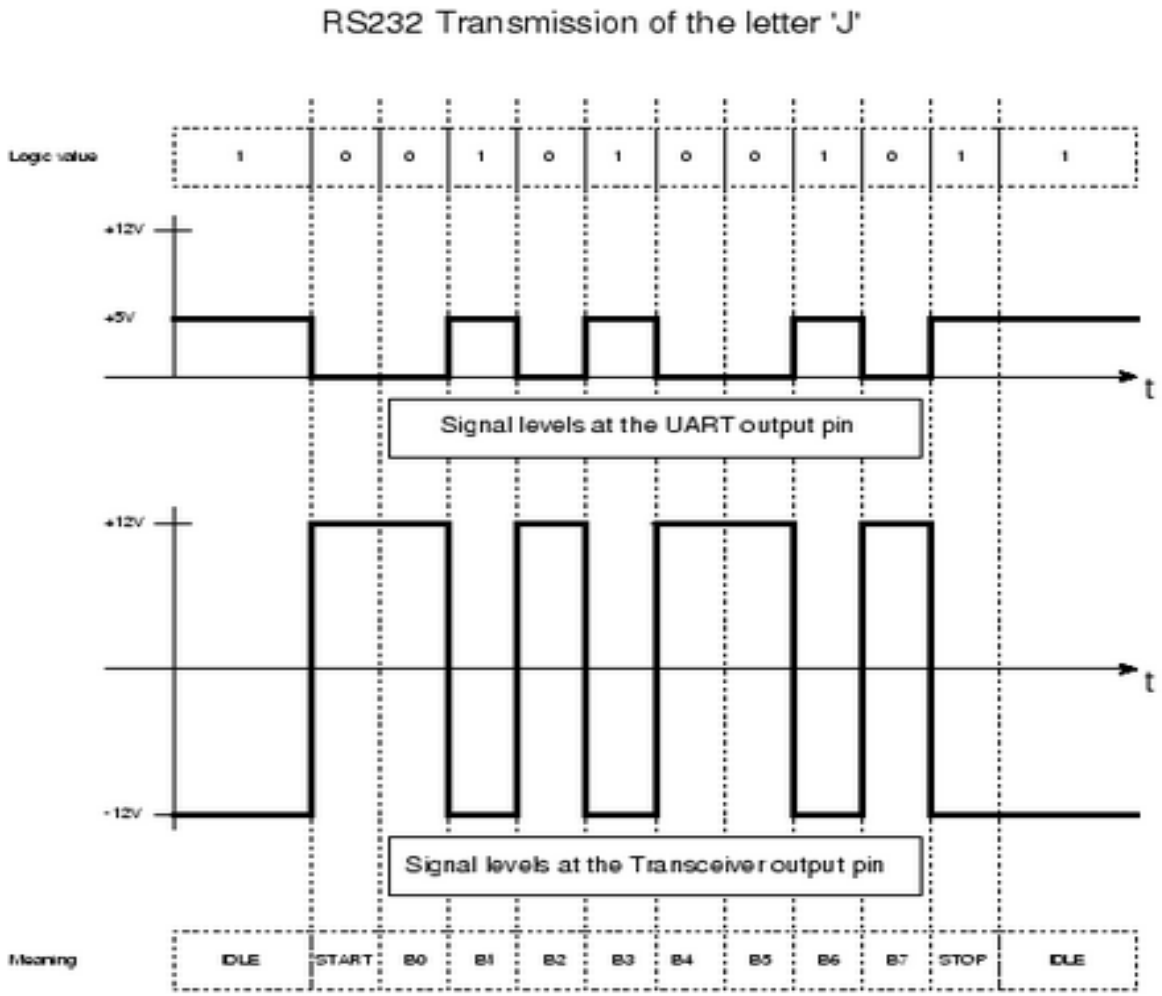
UART-RS232 Communication

- Serial data transmission - TX for Transmit and RX for Receive
- TX,RX and GND are the three wires RS232 interface.
- TX from one system will need to be connected to RX of the second
- RX from first system will need to be connected to TX of the second
- GND will need to be connected between the two systems.
- Transmission happens one byte after another in an asynchronous protocol which means there is no common clock signal.
- Both systems agree on a suitable data rate and a sequence for starting and ending data transfer.
- Transmission speed - bits per second is called the baud rate
- At 9600 baud, each bit is transferred in a period of 1/9600 seconds.
- 5V to 25V for a logic 0 and -5V to -25V for logic 1
- Level translator chip MAX3232 -Logic 1 is measured as -12V, logic 0 is measured as 12V.

UART-RS232 interface

- On the PC side the levels are reverse compared to what the microcontroller sends.
- On the scope a start bit will be a +12V pulse and the stop bit a -12V level.
- 9600 bits per second (104μµs per bit)
- Zoom in, and set a vertical cursor at the first edge. That's the start of your start bit. Move the second cursor to each of the next edges. The difference between the cursors should be multiples of 104μµs. Each 104μµs is one bit, first the start bit (1), then 8 data bits, total time 832μµs, and a stop bit (0).

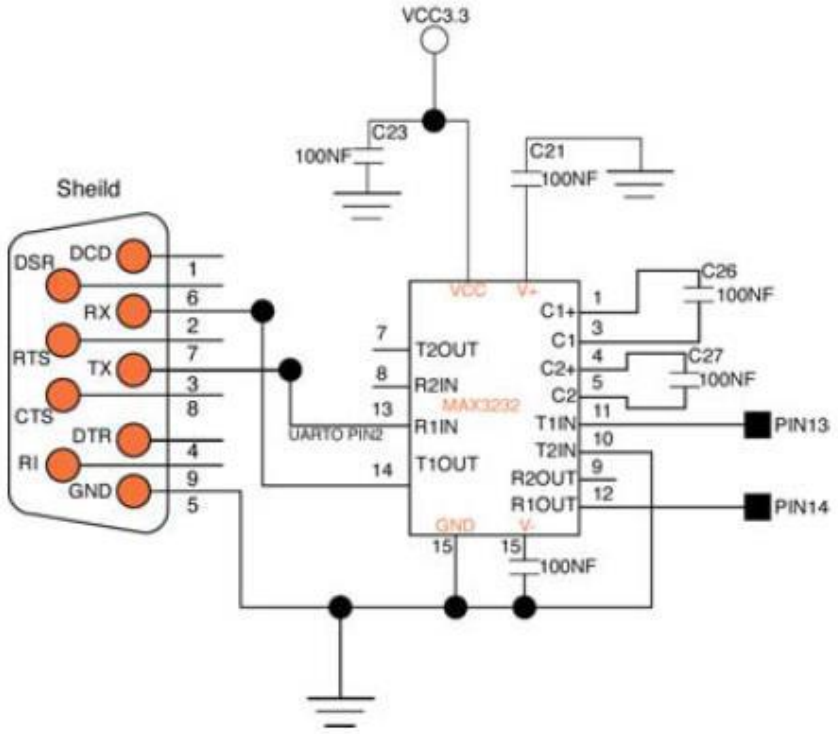
UART – Transmit sequence



To transmit character
J (0x4A)
= 0100 1010

Note that the 0 bit reads +12V and the 1 bit reads -12V on the scope.

UART Communication



UART-Asynchronous communication

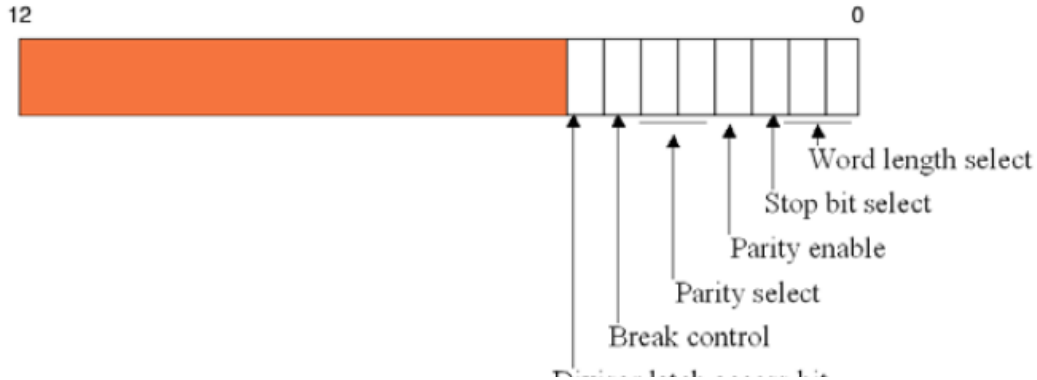
Peer to Peer communication

Full Duplex – RX from first device connected to TX of second device; TX from first device connected to RX of second device

Data is received in RX one bit at a time, Data is transmitted out of TX one bit at a time based on programmed bit timing or baud rate – 9600 baud, 19200, 115200 are common baud rates.

For transferring data to PC, line drivers such as MAX3232 used for level shifting.

UART Configuration



UART Line control register: The LCR configures the format of transmitted data. Setting the DLAB bit allows programming of the BAUD rate generators.

Baud rate configuration

The baud rate can be configured as follows.

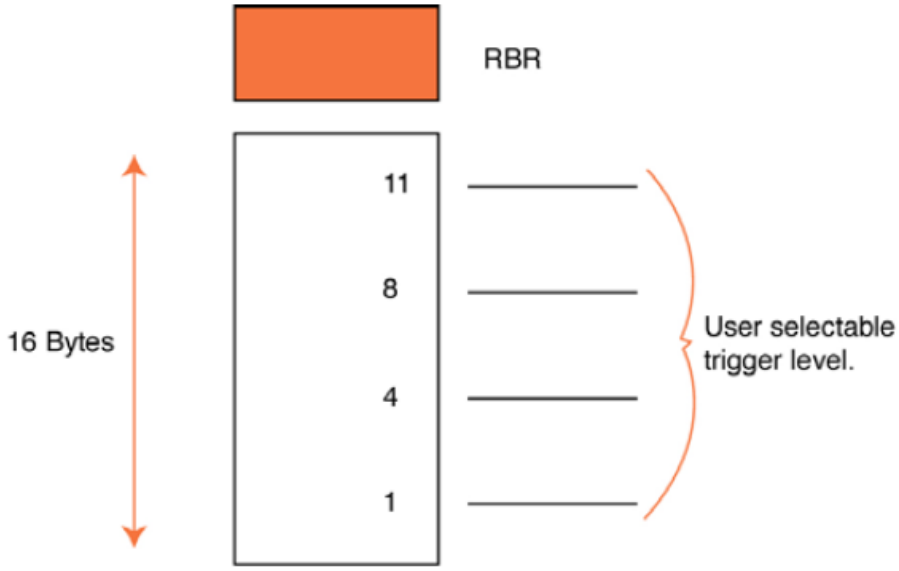
```
void init_serial (void)          /* Initialize Serial Interface       */
{
  PINSEL0     = 0x00050000;      /* Enable RXD1 and TXD1              */
  U1LCR       = 0x00000083;      /* 8 bits, no Parity, 1 Stop bit     */
  U1DLL       = 0x000000C2;      /* 9600 Baud Rate @ 30MHz VPB Clock  */
  U1LCR       = 0x00000003;      /* DLAB = 0                          */
}
```

The pinselect block must be programmed to switch the processor pins from GPIO to UART functions. The UART line control register is used to configure the format of the transmitter data register. The character format is set to 8 bits, no parity and one stop bit. The DLAB bit has to be set for programming the Baud rate generator/

Data Transmission

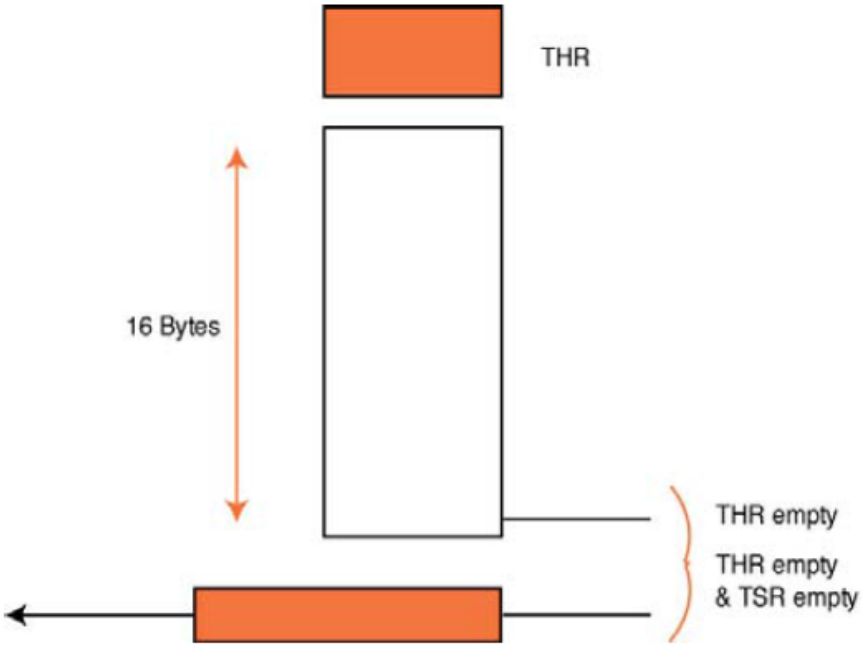
Once the UART is initialized, characters can be transmitted by writing to the Transmit Holding Register.
Characters can be received by reading the Receive Buffer Register. Both these registers hold the same memory location but writing a character places the value in the Transmit FIFO and reading the location loads a character from the Receive FIFO.

Receive Buffer



UART RX FIFO: Each UART has a sixteen byte receive FIFO which can be programmed to generate a UART interrupt at various trigger levels. The character timeout interrupt can be used to read bytes which do not reach a trigger level.

Transmit Buffer

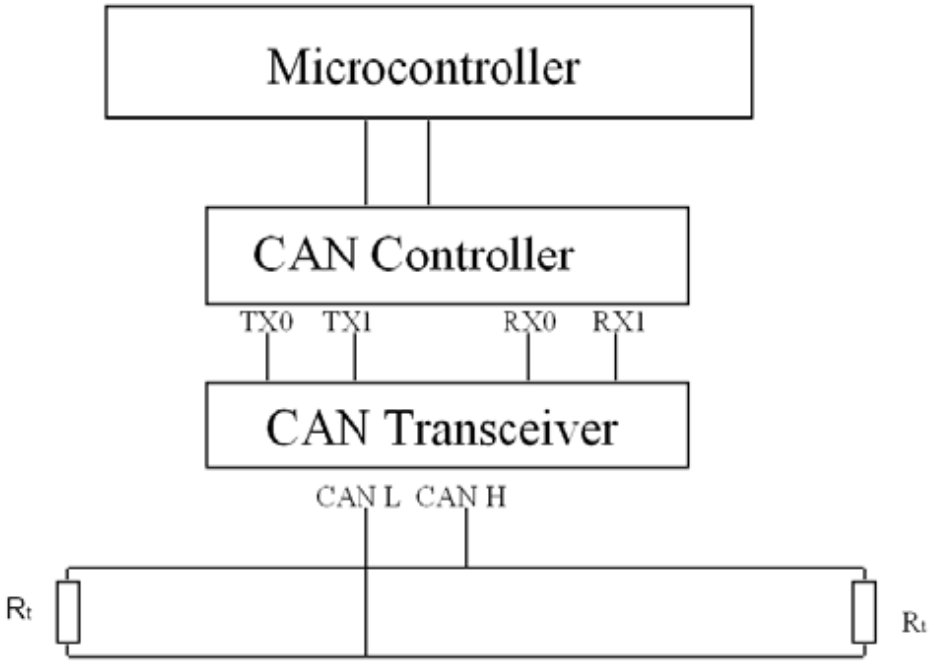


UART Transmit FIFO: Like the RX FIFO, the TX FIFO is 16 bytes deep and can generate an interrupt when empty and when it has finished transmitting.

CAN Communication

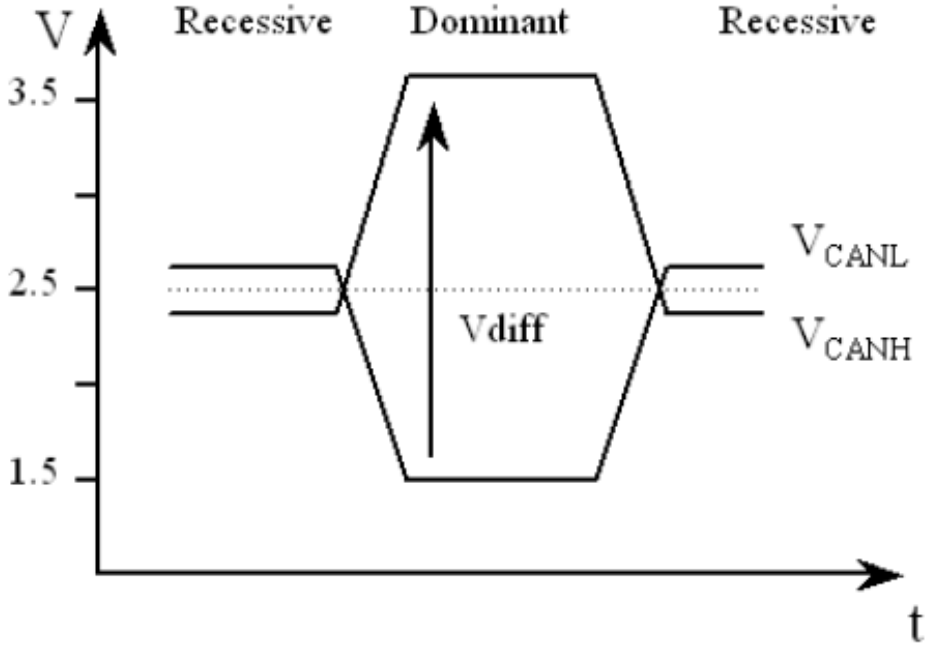
- CAN bus interface is used to communicate between modules in an automobile
- The LPC 23XX devices have two CAN controllers on the chip.
- The CAN protocol involves forming the message packet, error containment, acknowledgement and arbitration.
- The CAN node consists of a microcontroller interfaced to a CAN controller.
- CAN peripheral is interfaced to a CAN transceiver that provides a differential output to the CAN bus wires identified as CAN HIGH (CANH) and CAN LOW (CANL). The CAN bus wires need 120 ohm termination resistor at each CAN node.

CAN Node



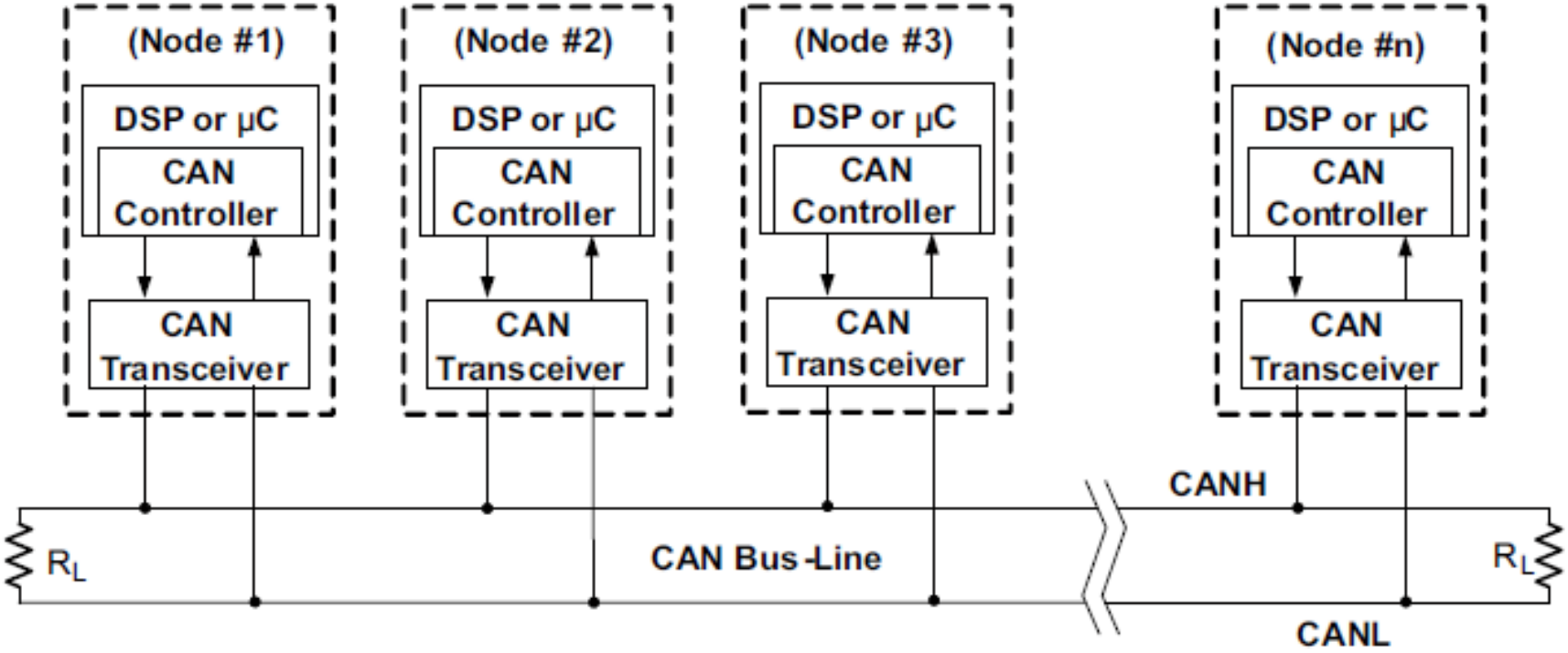
CAN node hardware: A typical CAN node has a microcontroller, CAN controller, physical layer and is connected to a twisted pair terminated by 120 Ohm resistors.

CAN Bus Voltages

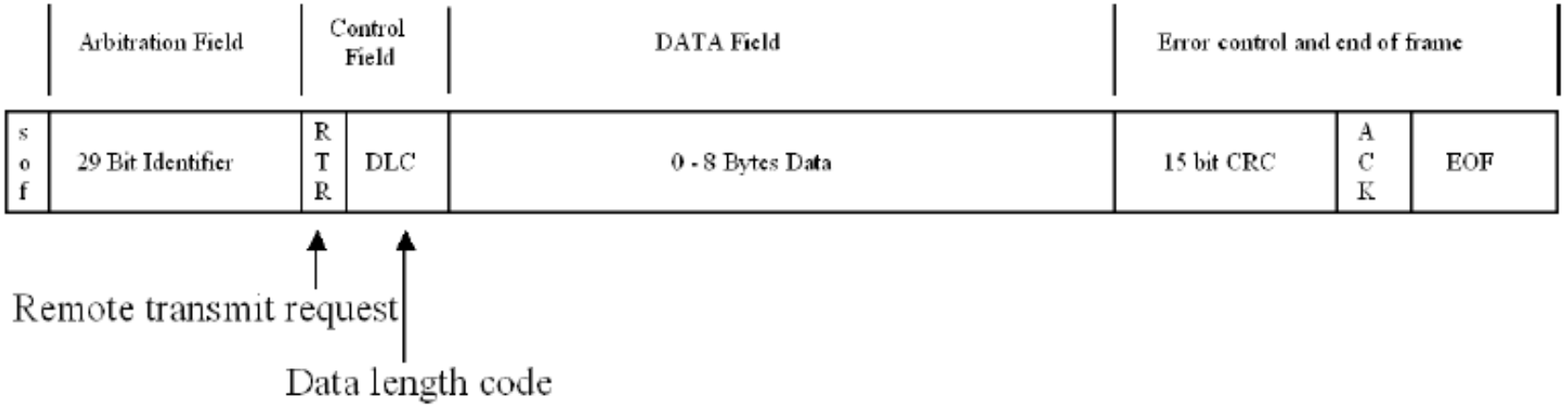


CAN Physical layer signals: On the CAN bus, logic zero is represented by a maximum voltage difference called "Dominant" and logic one by a bus idle state called "recessive". A dominant bit will overwrite a recessive bit.

CAN Nodes on a Bus

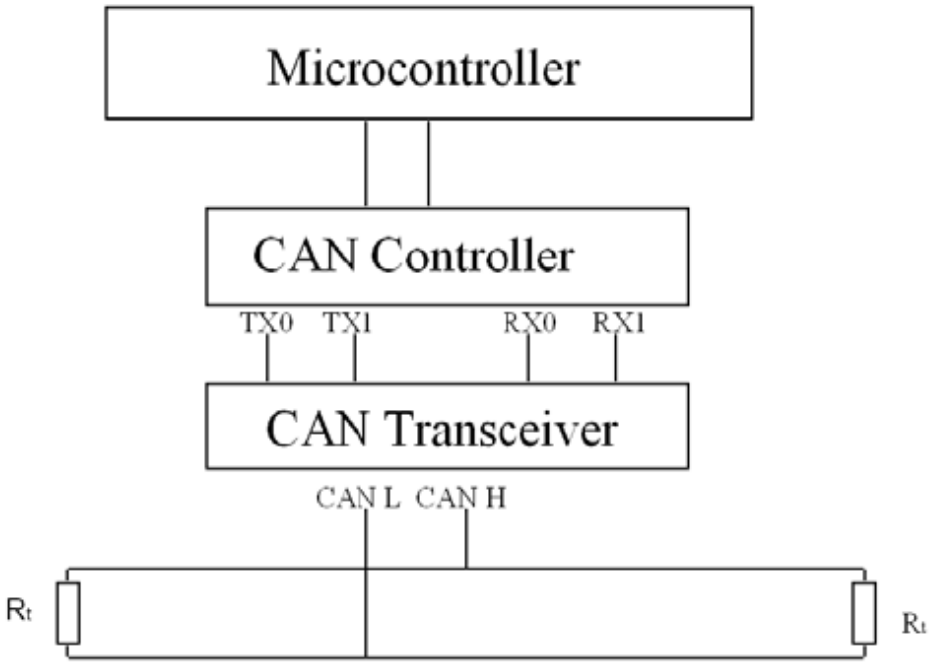


CAN Message packet



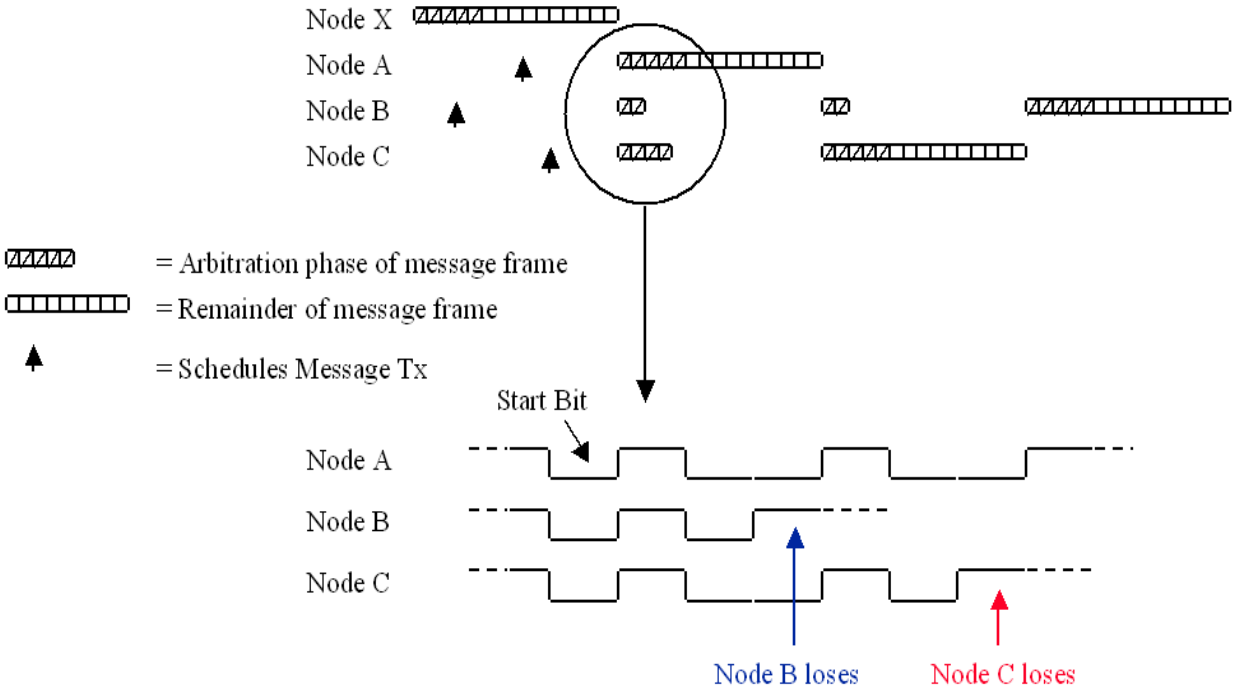
CAN message packet : The message packet is formed by the CAN controller, the application software provides the data bytes, the message identifier and the RTR bit

CAN Node



CAN node hardware: A typical CAN node has a microcontroller, CAN controller, physical layer and is connected to a twisted pair terminated by 120 Ohm resistors.

Arbitration in CAN Bus



CAN arbitration:

Message arbitration guarantees that the most important message will win the bus and be sent without any delay. Stalled messages will then be sent in order of priority, lowest value identifier first.

Arbitration…

- Arbitration is the process by which each node puts out a stream of dominant (0) and recessive (1) bits and checks if the transmitted bit is present on the CAN bus.
- Nodes that find that the transmitted bit is different from the bit present on the bus lose the arbitration and stop transmitting data and starts to listen on the bus.
- In the waveform above Node B first finds that it transmits a 1 but some other node has transmitted a 0 which happens to be the bus state being dominant. Hence Node B loses arbitration and becomes a listener.

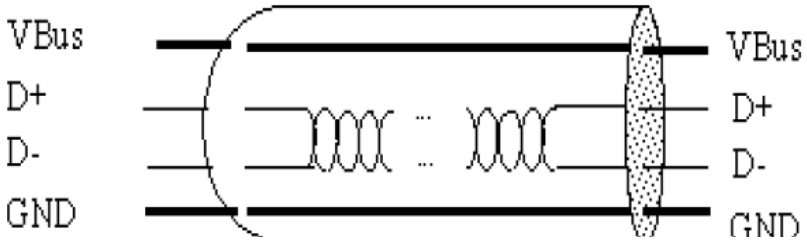
USB Interface

PERFORMANCE	APPLICATION	ATTRIBUTES
LOW SPEED Interactive Devices 10 - 100kb/s	Keyboard, Mouse Game peripherals Monitor Configuration	Lower cost Hot plugging Ease of use Multiple peripherals
FULL SPEED Phone, Audio Compressed Video 500Kb/s - 10Mb/s	Printers Scanners Telephony Audio	Low cost Hot plugging Ease of use Guaranteed latency Guaranteed Bandwidth Multiple devices
HIGH SPEED Video, Disk 25 - 500 Mb/s	Video Mass Storage	High Bandwidth Guaranteed latency Ease of use

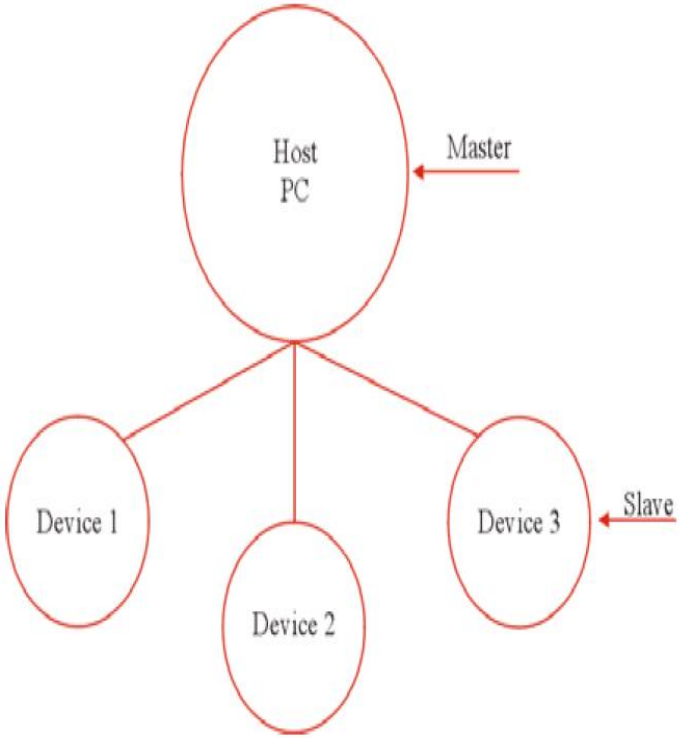
USB host and peripheral have standardised plugs and sockets to ensure easy installation.

- Once a device is connected to the PC, and the signal speed determined, the PC can begin to transmit or receive data with the device connected through a logical connection called the pipe.
- The pipe is terminated inside the device at what is called the Endpoint which could refer to a buffer where the data is stored and an interrupt that signals when the data has arrived.
- There are four types of pipes – control, interrupt, bulk and isochronous. Each device has a control pipe and it will appear at Endpoint zero.
- When a new device is plugged, it will appear as device zero and the PC can communicate to it by sending information to Endpoint zero. The remaining types of pipe are used for the user application.

USB Interface

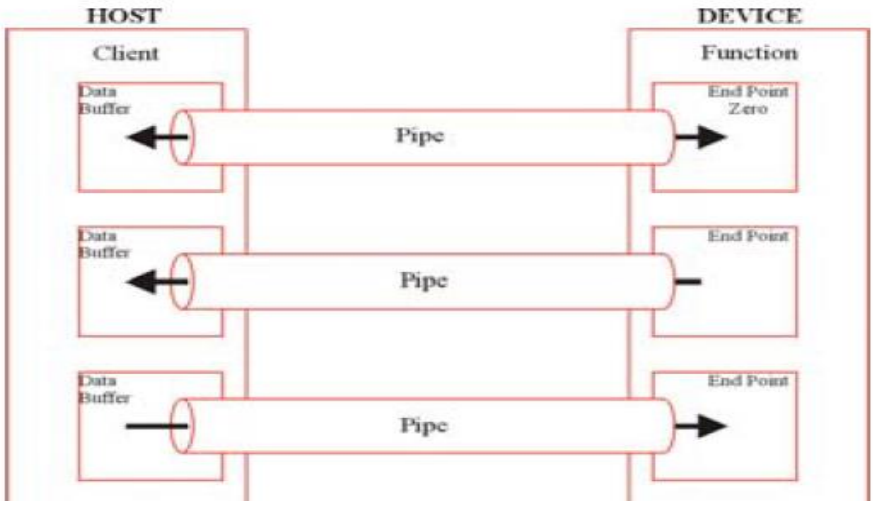


Standard USB cables carry four wires. Power as 5V and Gnd and two data wires called D+ and D-



To the programmer the USB network appears as a master slave star network.

USB Pipes



Each USB slave is characterised by a local address and a set of logical endpoint buffers. The Host creates logical connections called "pipes" to each endpoint which are used to transfer packets of information.

Every 1msec the PC sends a start of frame token to delineate the 12Mbit/sec bus into a series of frames. Each pipe is allocated a slot in each frame so it can transfer data as required.