

I/O Interfacing Standards for External Devices

Common Digital I/O Interfacing Standards for External Devices

- Serial Port
- PS/2
- SPI, SCI, SI, and SDIO
- I²C
- Audio AC97
- USB
- Firewire
- GPIO
- PWM
- CAN bus

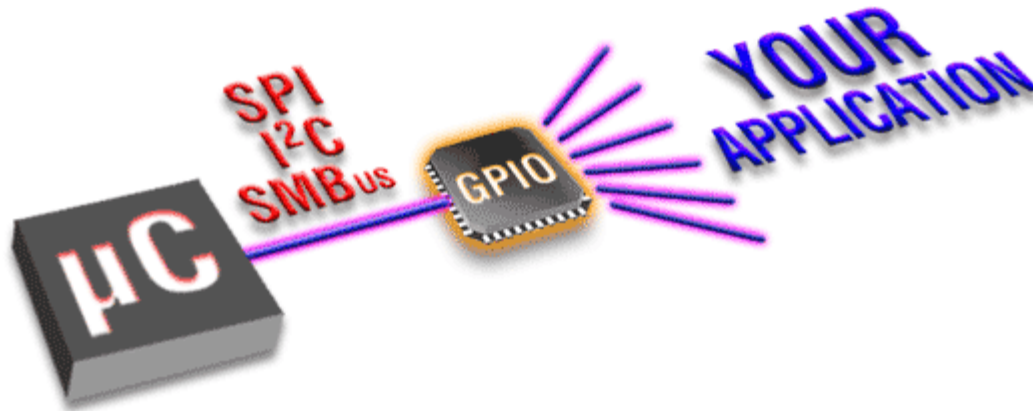
Serial communication

- Can support high speed communication
- Support Synchronous, Asynchronous, and Iso-synchronous

General Purpose Input/Output (GPIO)

- Sometimes it is called bus expander
- GPIO is an interface available on microcontrollers/microprocessors
- It can act as input or output
- GPIO often are arranged into group of 8 pins
- GPIO port is often individually configurable

GPIO interface



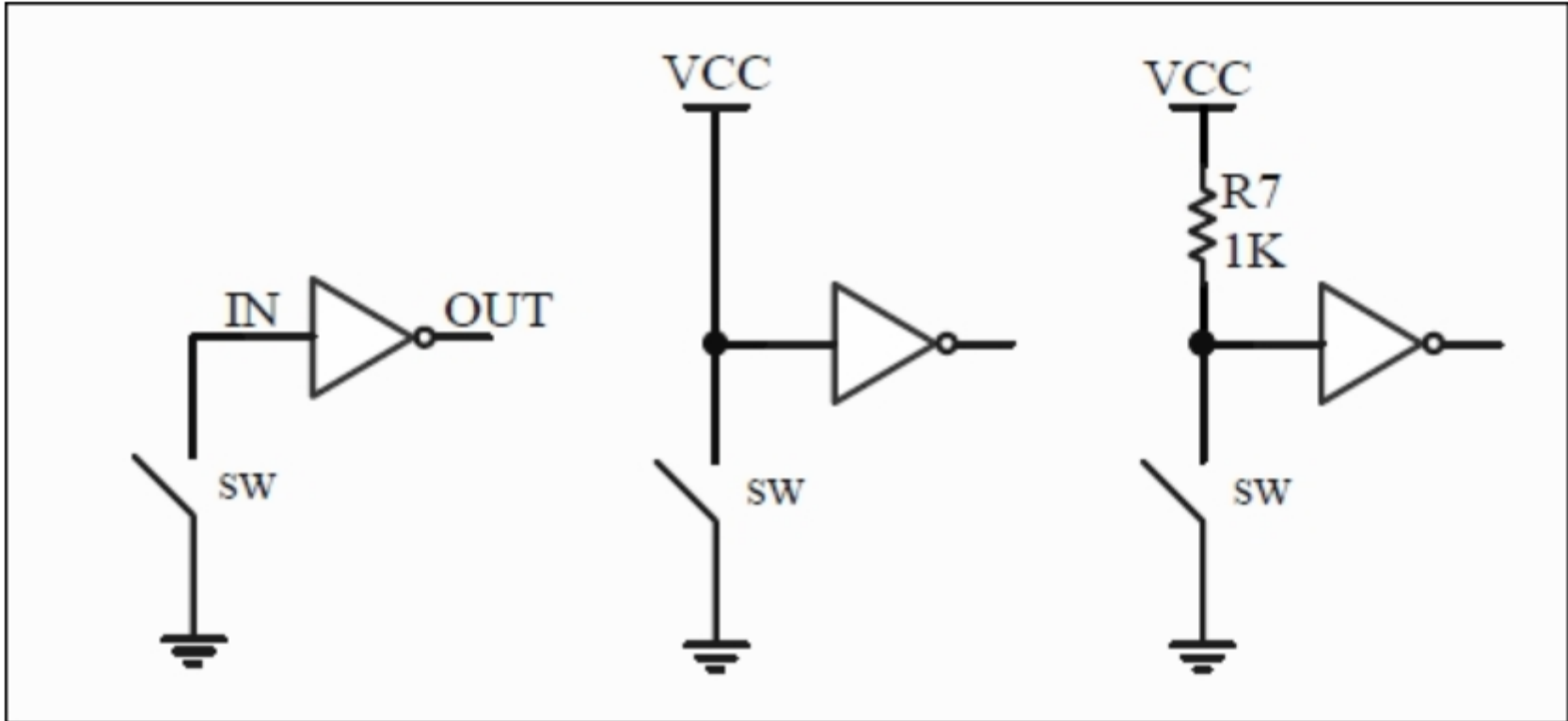
GPIO Advantage

- Lower power about 1 μA
- Can fit in small package
- Lower Cost
- Faster time to market

Pull-up Pull-down resistors

- Makes the circuitry susceptible to noise
- Without the input, the logic state is unknown
- Pull-up resistor pulls the terminal to the voltage supply
- Pull-down resistor pulls the terminal to the ground

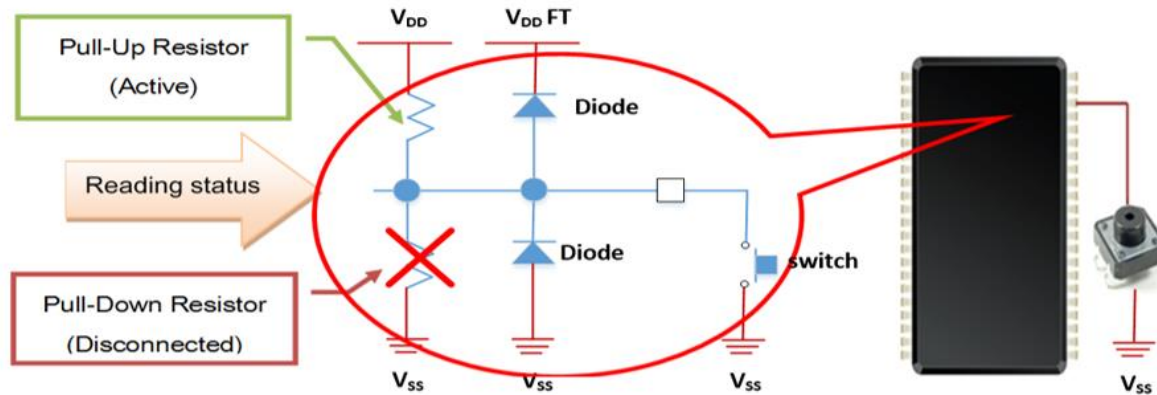
Pull-up resistors



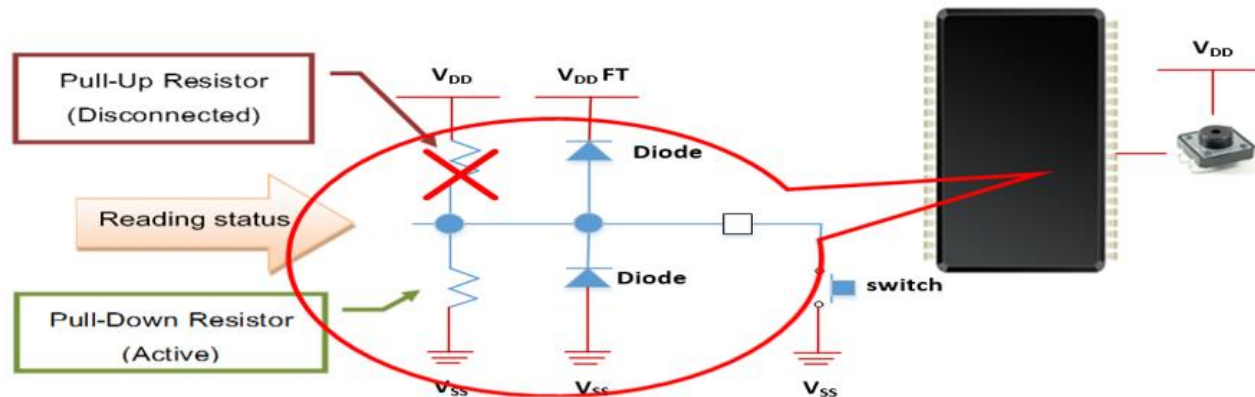
Connection with and without pull-up resistors

Input example

Pull-Up

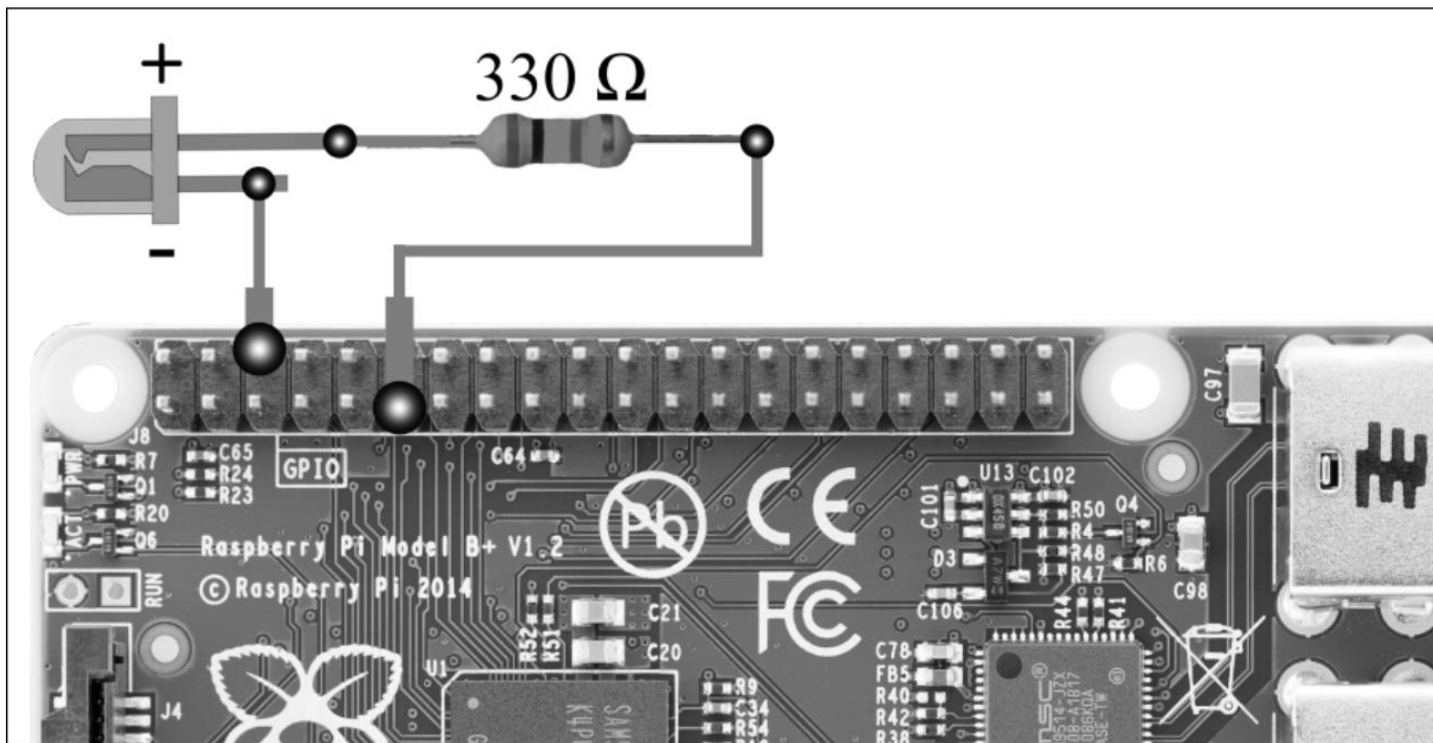


Pull-Down



Raspberry Pi

- Provide Pull-up resistors inside the board



Pin naming

- Board mode/BCM mode

Physical/RasPi names (Board)				Broadcom names (BCM)			
Left		Right		Left		Right	
Pin Function	Pin No.	Pin No.	Pin Function	Pin Function	Pin No.	Pin No.	Pin Function
3.3V	1	2	5V	3.3V	1	2	5V
I2C0 SDA	3	4	DNC	I2C0 SDA	3	4	DNC
I2C0 SCL	5	6	GND	I2C0 SCL	5	6	GND
GPIO 7	7	8	UART TX	GPIO 4	7	8	UART TX

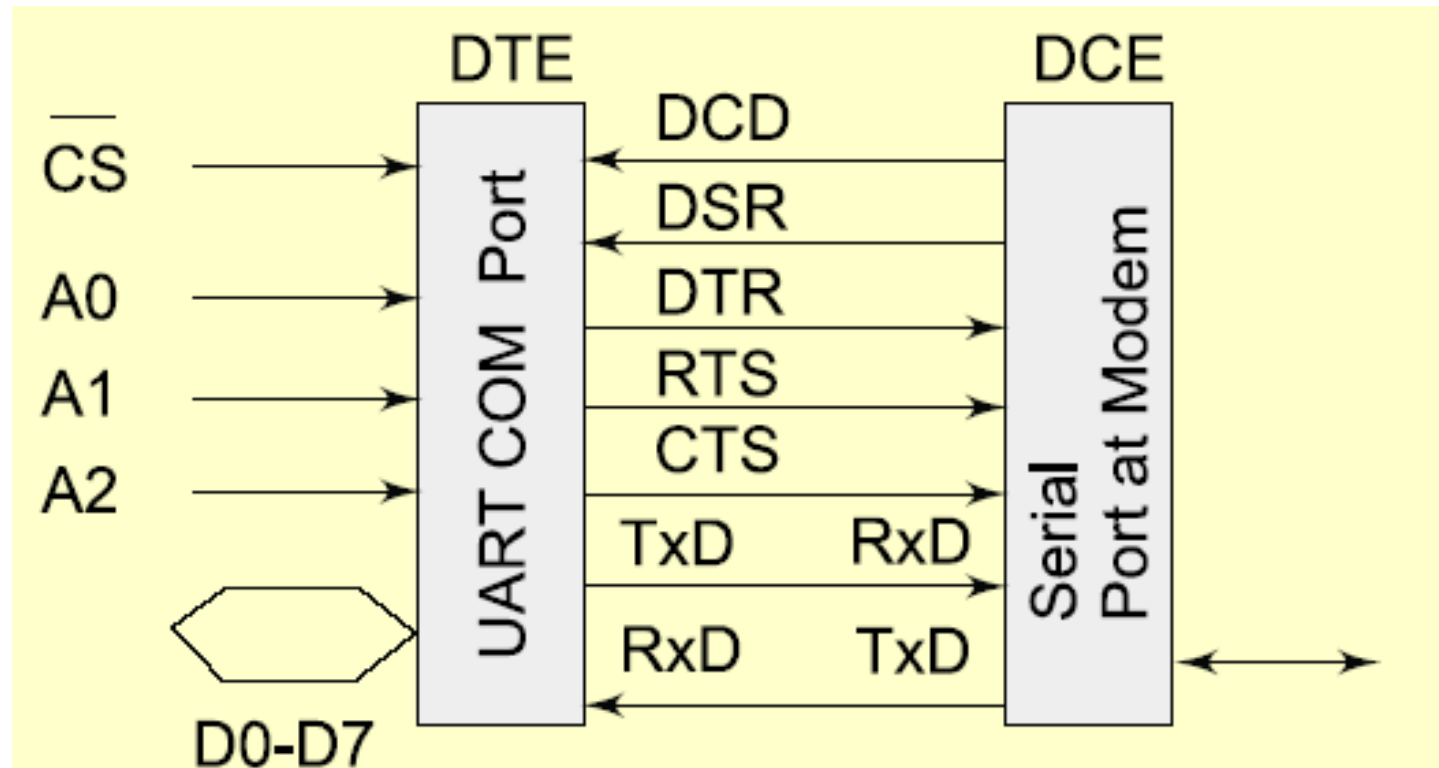
LED Light up

```
import RPi.GPIO as GPIO
from time import sleep
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT)
Print "All Set in Python! Let's Blink"
for i in range(1,10):
    GPIO.output(17,GPIO.HIGH)
    sleep(1)
    GPIO.output(17,GPIO.LOW)
    sleep(0.5)
GPIO.cleanup()
```


RS232C

- RS232C communication is between Data Terminal Equipment (DTE) e.g. computer and Data Communication Equipment (DCE) e.g. modem
- RS232C (Recommend Standard for Number 232C) specify communication standard such as voltage level, terminating resistances, cable length etc.

RS232C port connection



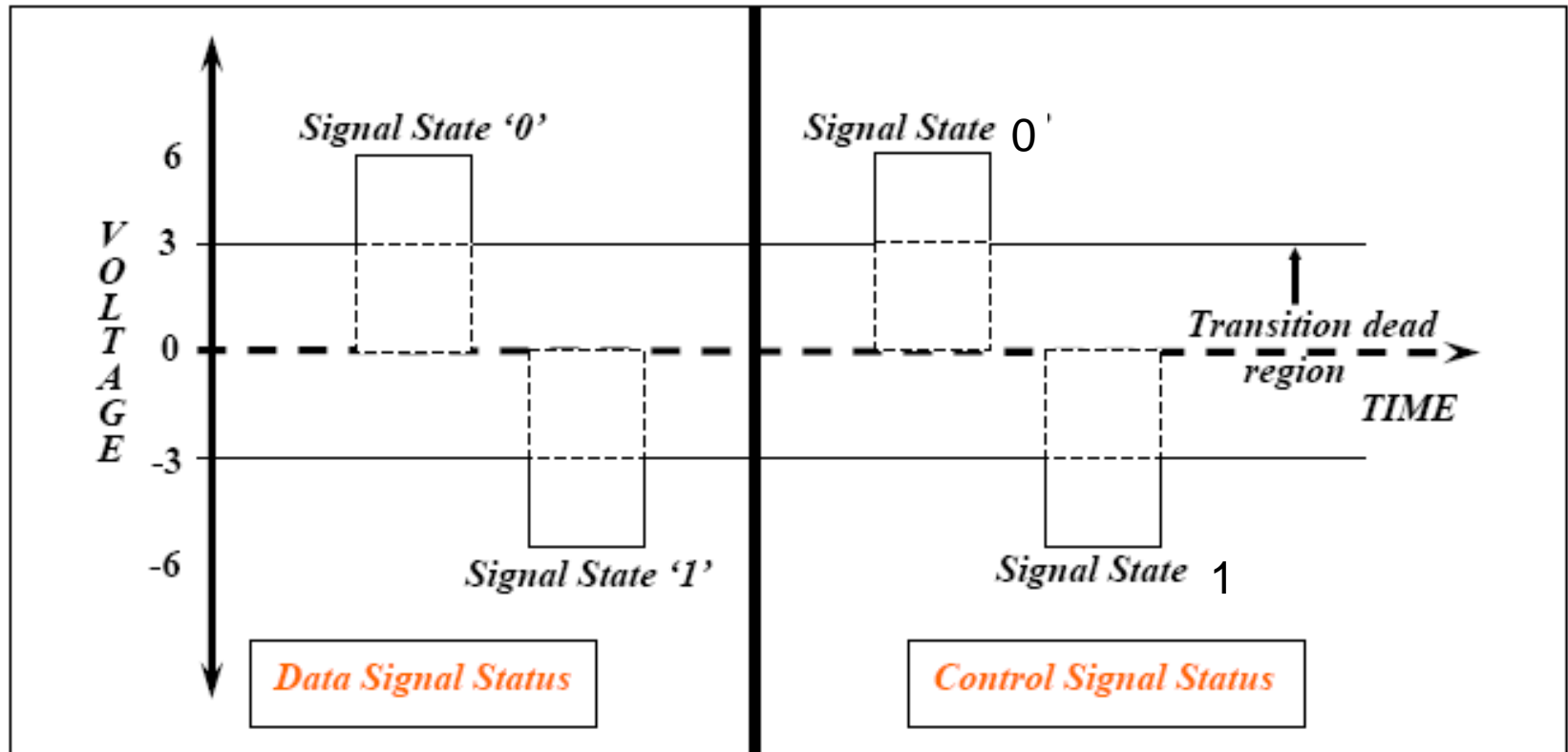
RS-232 Serial Interface

- Transmit and Receive data lines
- No clock signal exchanged – Sender and receiver need to have same baud rate
- Baud rate is the clock rate of the bits
- Normal Bits: Start, 8 Data, Stop
- Voltage levels: a 0 is **>3V** and a 1 is **<-3V**
- Special RS232 voltage level converter chips are typically used in interface

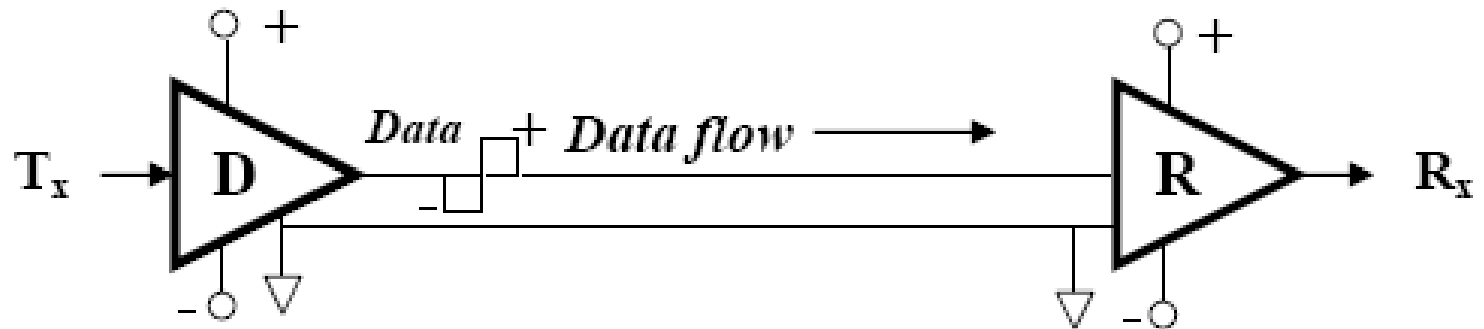
RS-232 standard

- Data rate from 20 kbps to over 1 Mbps
- Range up to 50 feet maximum
- It is robust interface up to 115,200 baud rate (pulse per second)
- Voltage as high/low as ± 15 Volt
- Single-ended means communication is over a single wire reference to ground
- There are 9 pins (DB-9) and 25 pins format (DB-25)

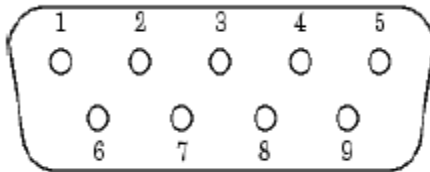
RS-232 signal



RS-232 single ended uni-direction



RS-232 (DB9) male connector



Pin 1: Carrier Detect (CD)

Pin 2: Receive Data (RD)

Pin 3: Transmit Data (TD)

Pin 4: Data Terminal Ready (DTR)

Pin 5: Ground (GND)

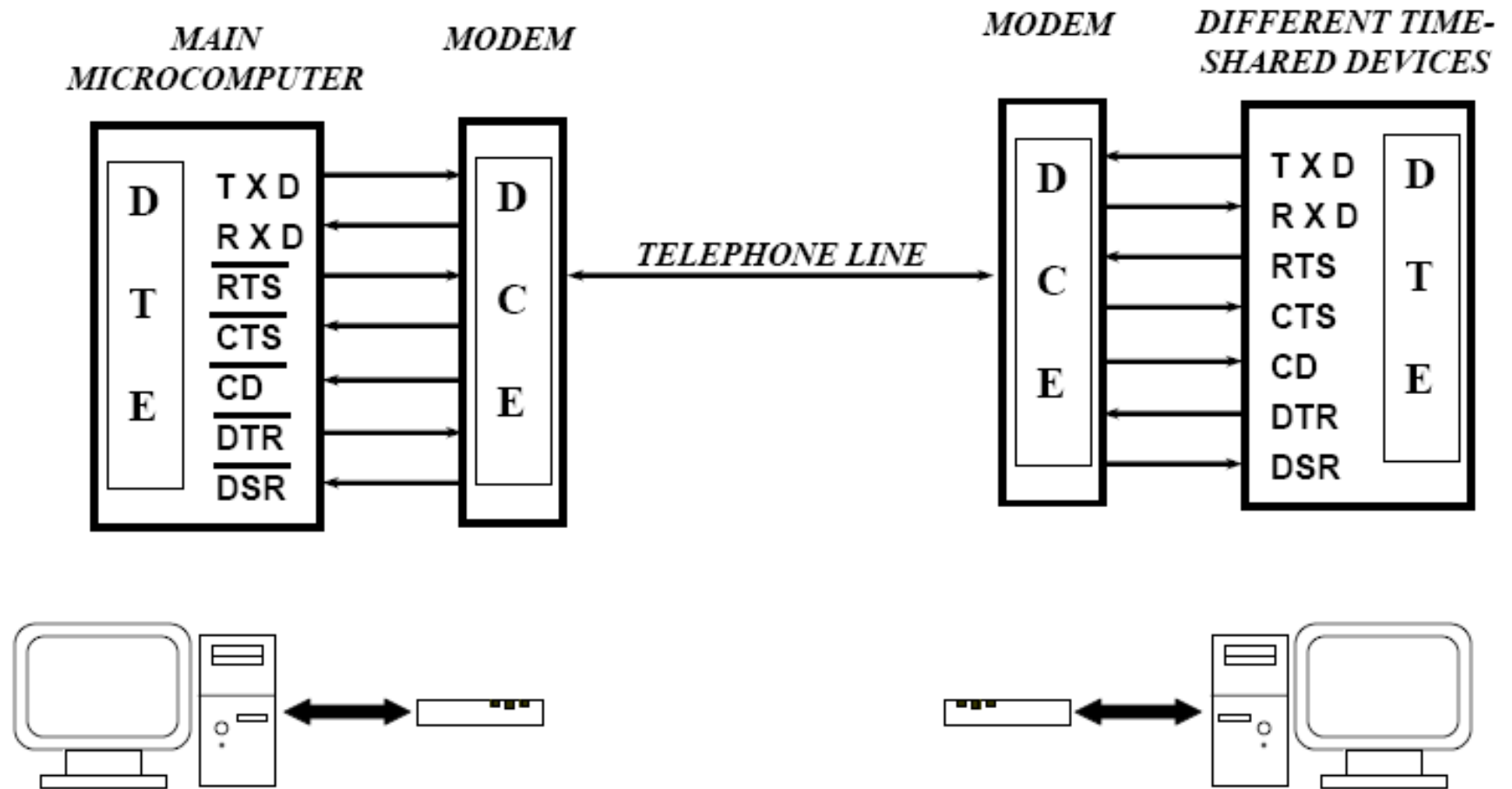
Pin 6: Data Set Ready (DSR)

Pin 7: Request to Send (RTS)

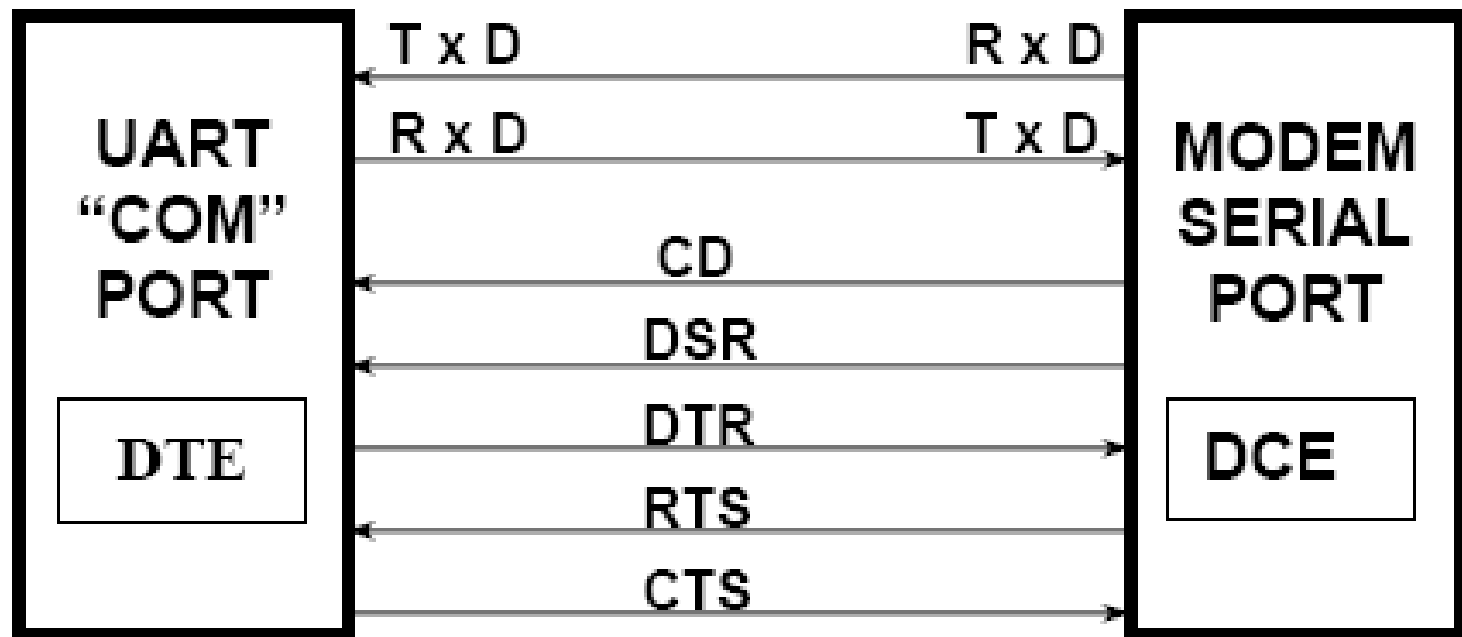
Pin 8: Clear to Send (CTS)

Pin 9: Ring Indicator (RI)

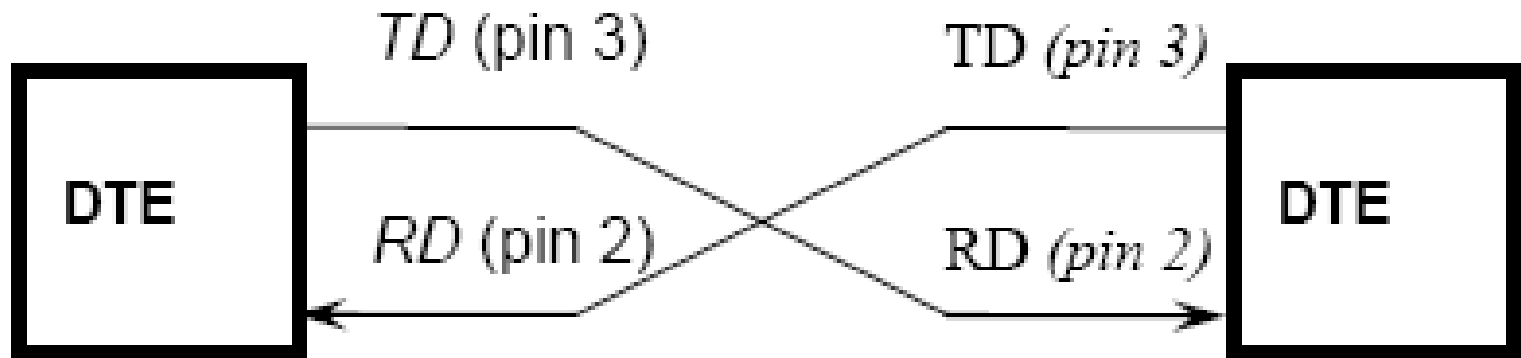
Connect computer-modem



From DTE-DCE



Connect two PC directly



RS232 Handshaking

Assume modem wants to send data to PC

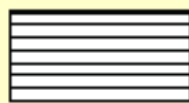
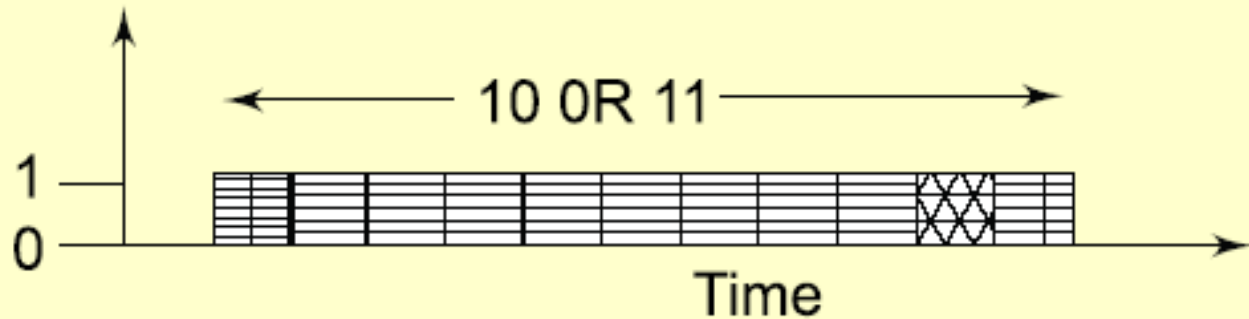
- RI indicate data available
- When modem connects, modem will send DCD signal at time t_0
- Modem will send DSR signal at time t_1 when it receive data to send
- PC will response with DTR at time t_2
- Modem will send RTS at time t_3
- PC response with CTS at time t_4

RTS and CTS can also be sent again during the transaction

UART

- UART is the name for the hardware used for a RS-232 Serial Interface
- UART – Universal Asynchronous Receiver Transmitter
- Early PCs had a UART chip, but this functionality is now found inside a larger chip that also contains other I/O features

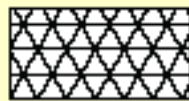
UART transmission



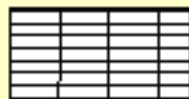
UART Serial
Bits for Data



Start Bit



P Bit
(Optional)



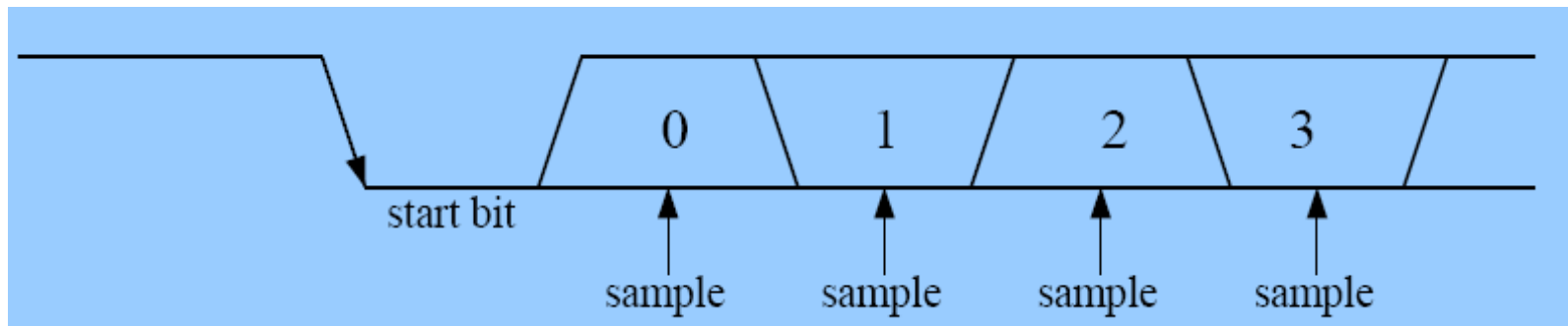
Stop Bit



In a different
Phase or
Frequency
for State 1
and
State 0

UART initial communication

- Need to know how fast the data bits are coming
- Need to know where the start bit begins
- Then, we know when to sample

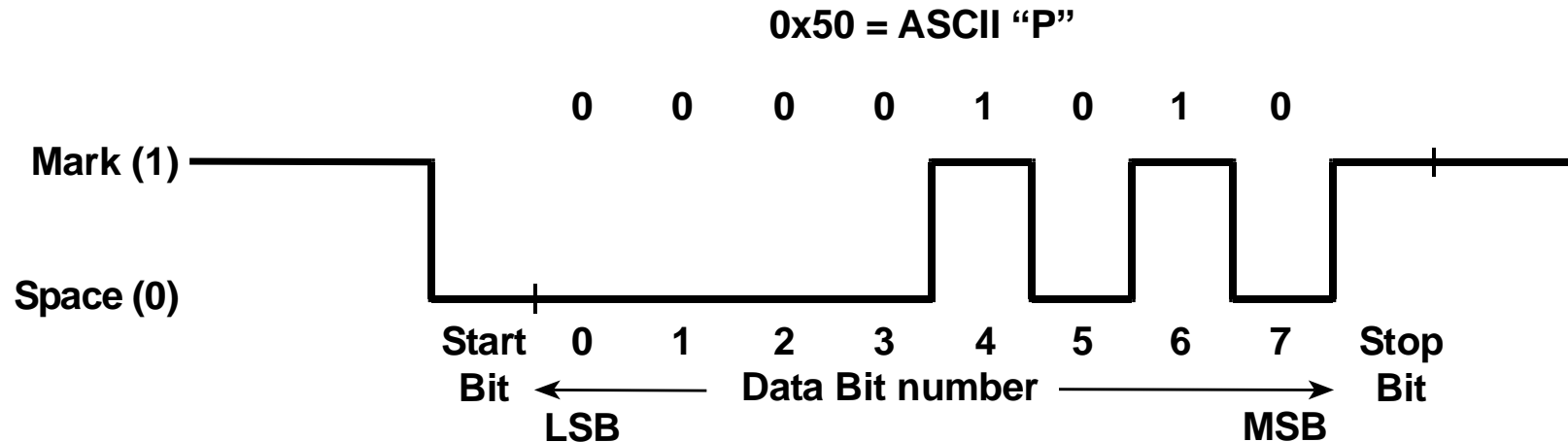


UART communication

- Non-return to zero. In the idle state, the logic state is 1.
- Start bit: transition to 0
- Data bit consists of start bit, 8 bit data, P-bit and stop bit
- Data bits can be changed to 5,6,7, and 8 bits
- The stop bit can be for a minimum of $1.5T$, $2T$ instead of T , when T is normal interval
- P bit can be priority or for other purpose
- Stop bit: transition to 1

RS-232C Serial interface

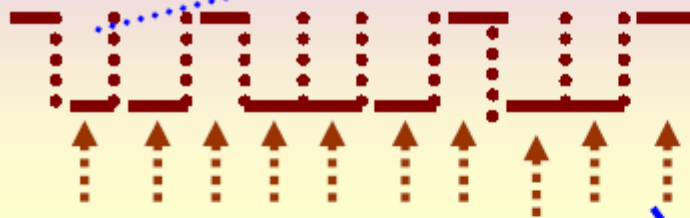
transmission of an 8-bit data value



UART output 8 bits in 10T

UART output 8 bits (01000100)
in 10T format

Start bit

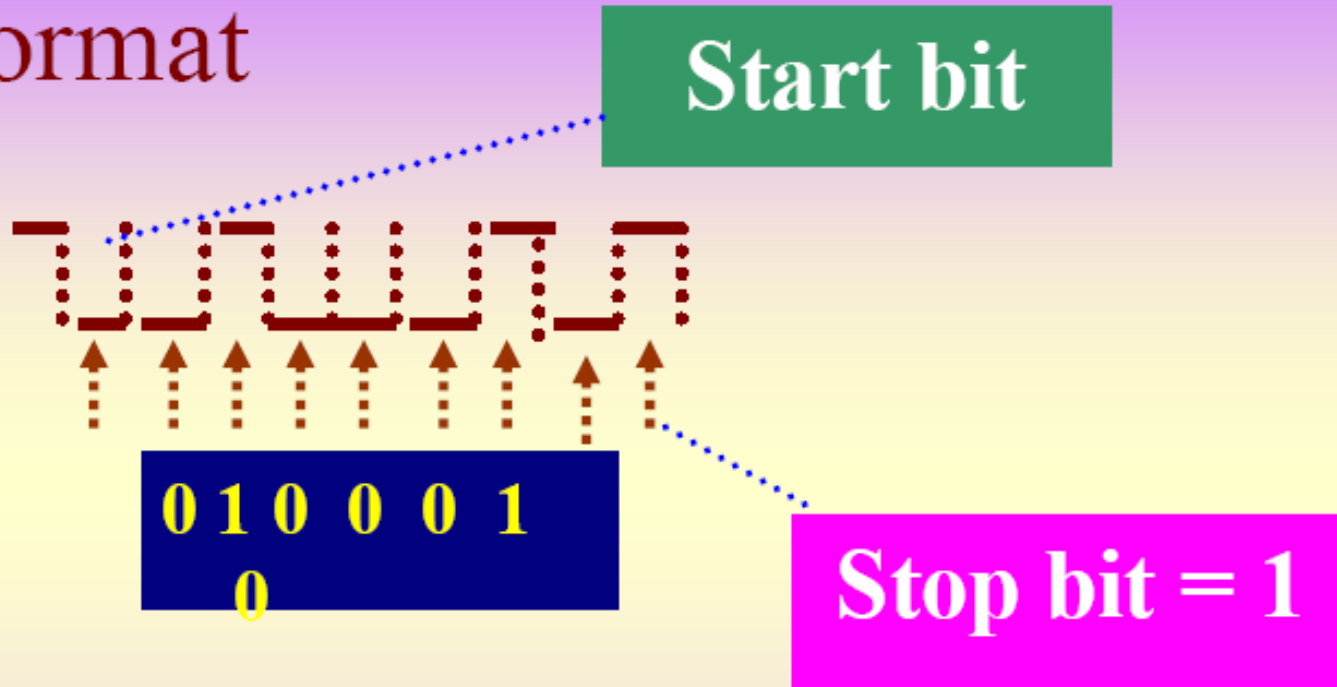


0 1 0 0 0 1 0
0

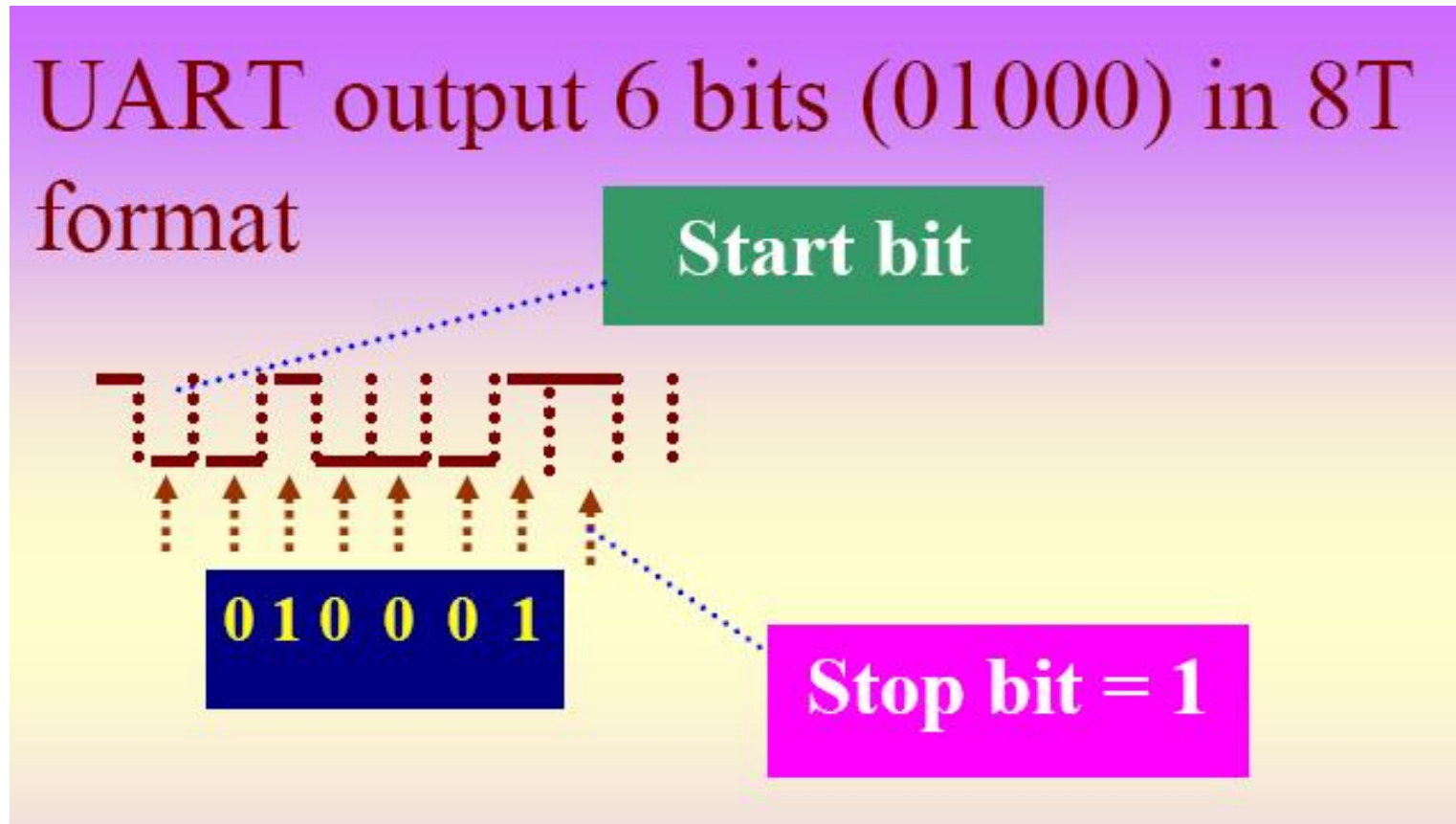
Stop bit = 1

UART output 7 bits in 9T

UART output 7 bits (0100010) in 9T format

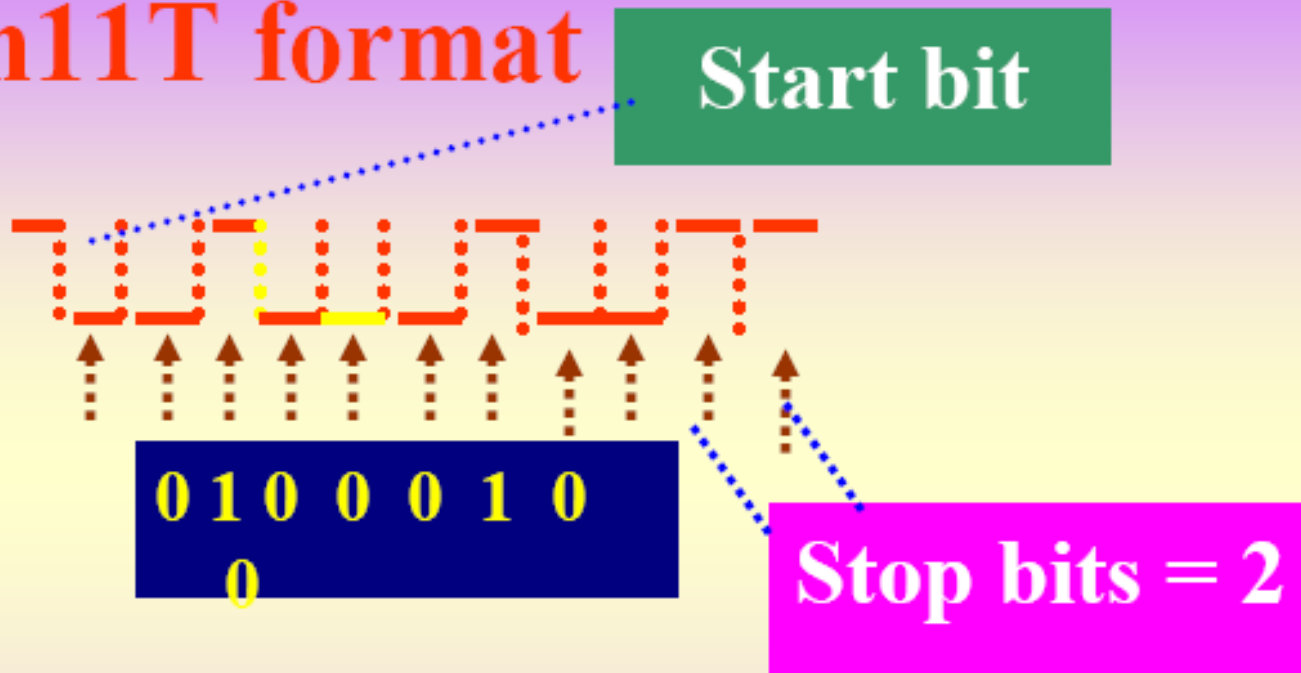


UART output 6 bits in 8T



UART output 8 bit in 11T

UART output 8 bits (01000100)
in 11T format



Raspberry Pi UART

- Since we use 0 and 3.3 V logic levels in Pi (not the +/-12 V used by RS-232), we need a 3.3 V to RS-232 level converter.
- Another option is to use USB to RS-232 converter

Raspberry Pi Setup

- Default configuration

Speed (baud rate): 115200

Bits: 8

Parity: None

Stop Bits: 1

Flow Control: None

- Checking the serial port

ls -l /dev/ttyS0, /dev/ttyS1

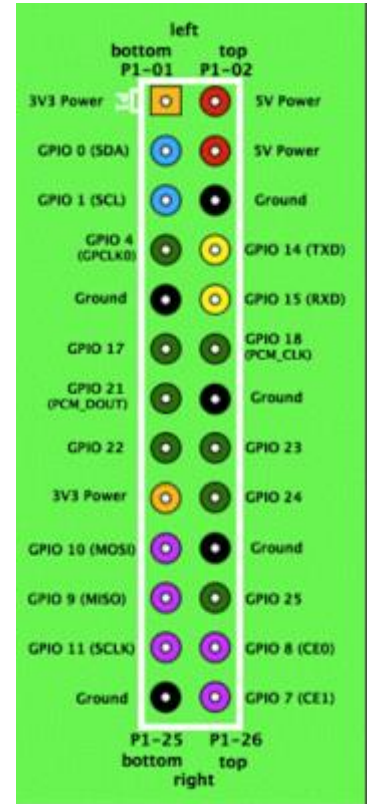
Or

ls -l /dev/ttyUSB0, /dev/ttyUSB1

Python Installation

- Install Python Serial Port

`sudo apt-get install python-serial`



Serial Port Programming

```
import serial

port = serial.Serial("/dev/ttyAMA0", baudrate=115200, timeout=3.0)

while True:
    port.write("\r\nSay something:")
    rcv = port.read(10)
    port.write("\r\nYou sent:" + repr(rcv))
```


Run the program

- To run the program
`python serialtest.py`
- Output

```
Say something:  
You sent: 'abcabc'
```


Read Data from Serial Port

- Example of the Program

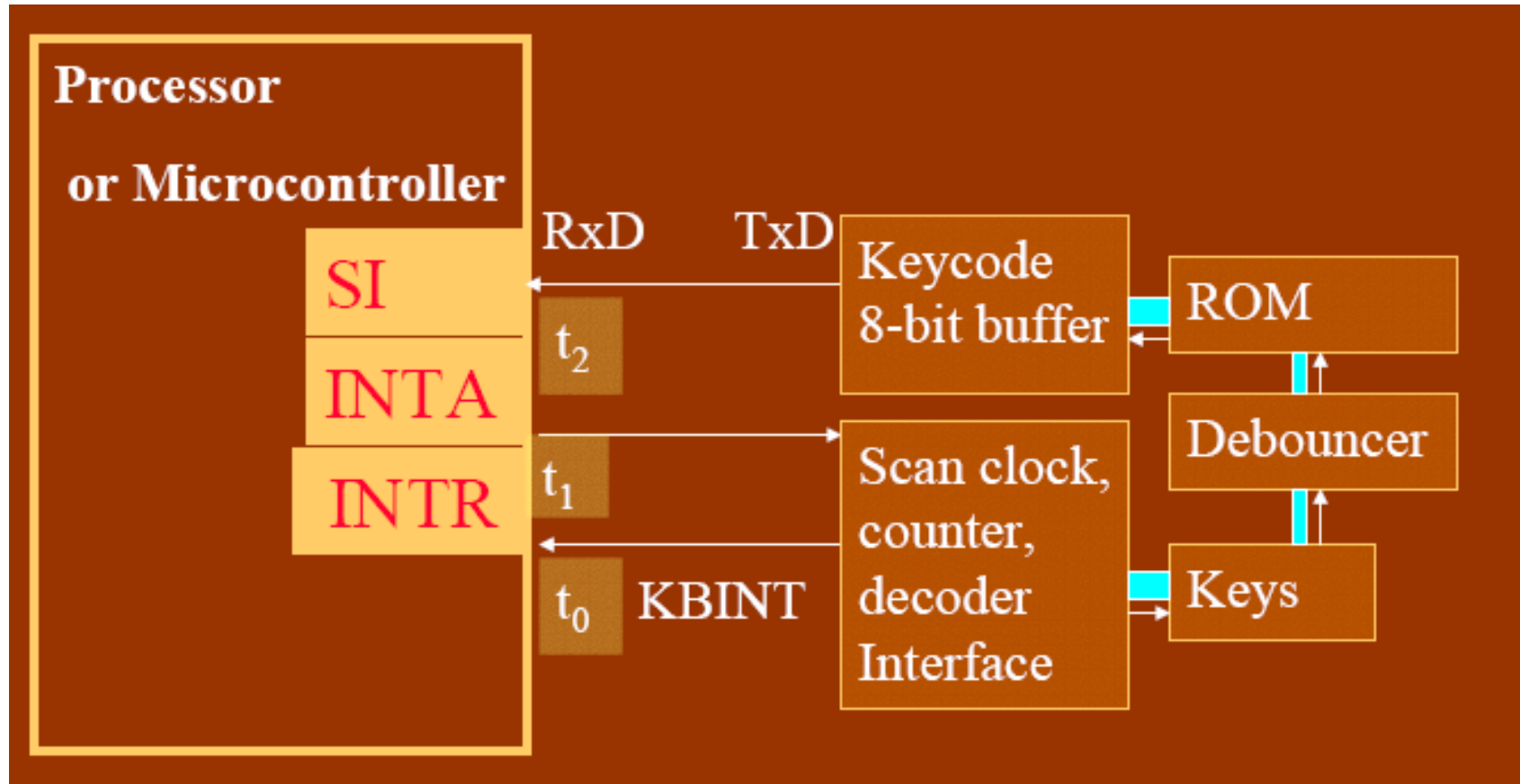
```
import serial
import time

def readlineCR(port):
    rv = ""
    while True:
        ch = port.read()
        rv += ch
        if ch=='\r' or ch=='':
            return rv

port = serial.Serial("/dev/ttyAMA0", baudrate=115200, timeout=3.0)

while True:
    port.write("\r\nSay something:")
    rcv = readlineCR(port)
    port.write("\r\nYou sent:" + repr(rcv))
```


Keyboard interface to serial interface to Microcontroller



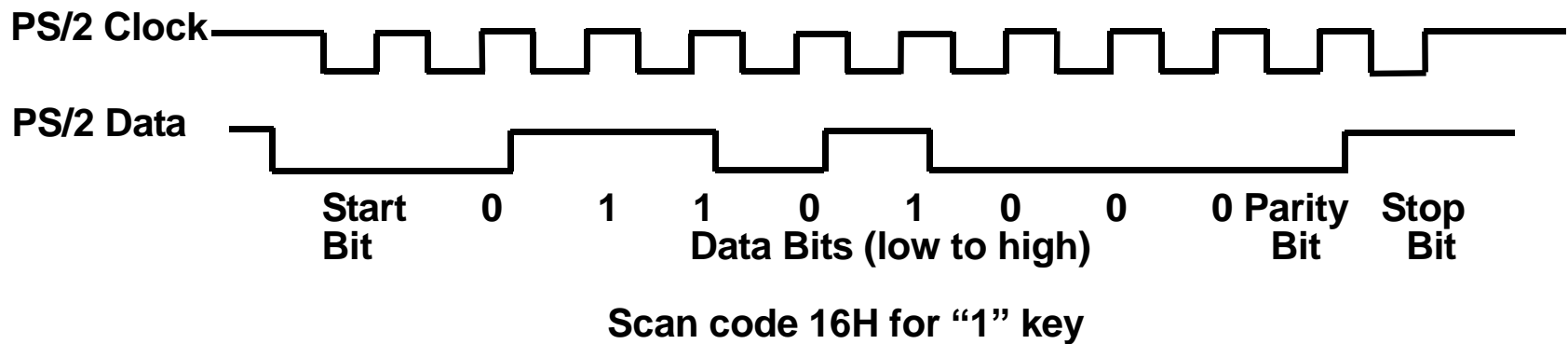
Keyboard interface

- KBINT is an interrupt from keyboard controller
- TxD is serial UART data output
- Bounce creates on pressing
 - Each bounce creates a false pulse
- Scan clock generates clock to keep checking key press for each interval
- Encoder is used to encode keyboard output

Personal System 2 (PS/2) Interface

- IBM standard from an early PC
- Used in low-cost keyboards and mice
- Two signal lines: clock & data
- Keyboard sends key scan codes
- Mouse sends 3 byte data packets
- 11bits sent: Start, 8 Data, Parity, Stop
- Can also send commands to Mouse and Keyboard
- Uses Open Collector TTL bus signals
 - 1 Pull-up resistor
 - Device can only force signal low
- Small microcontroller used in a PS/2 device

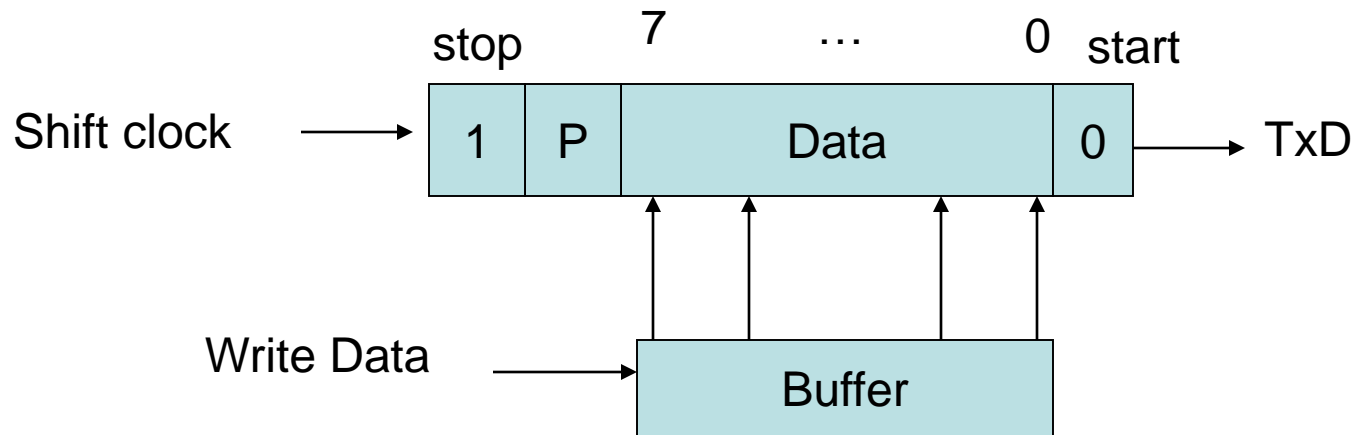
PS/2 Keyboard Transmission of a Keyboard Scan Code



Serial Peripheral Interface (SPI)

- Developed by Motorola in the 1980s
- Works like a shift register for input and output
- 4 signals: clock, shift-in, shift-out, and chip enable
- Chip enable turns on a tri-state driver on each slave devices shift-out pin
- Two setup modes: one large shift register with chip enables all connected or independent shift registers each with a chip enable
- Common in A/D and D/A chips

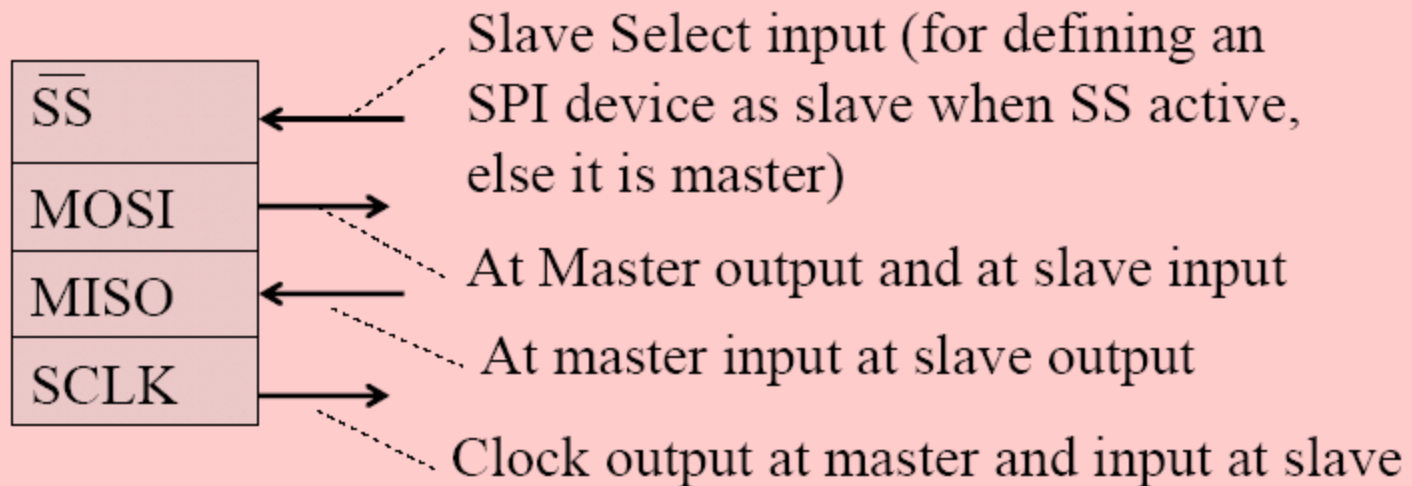
Shift register



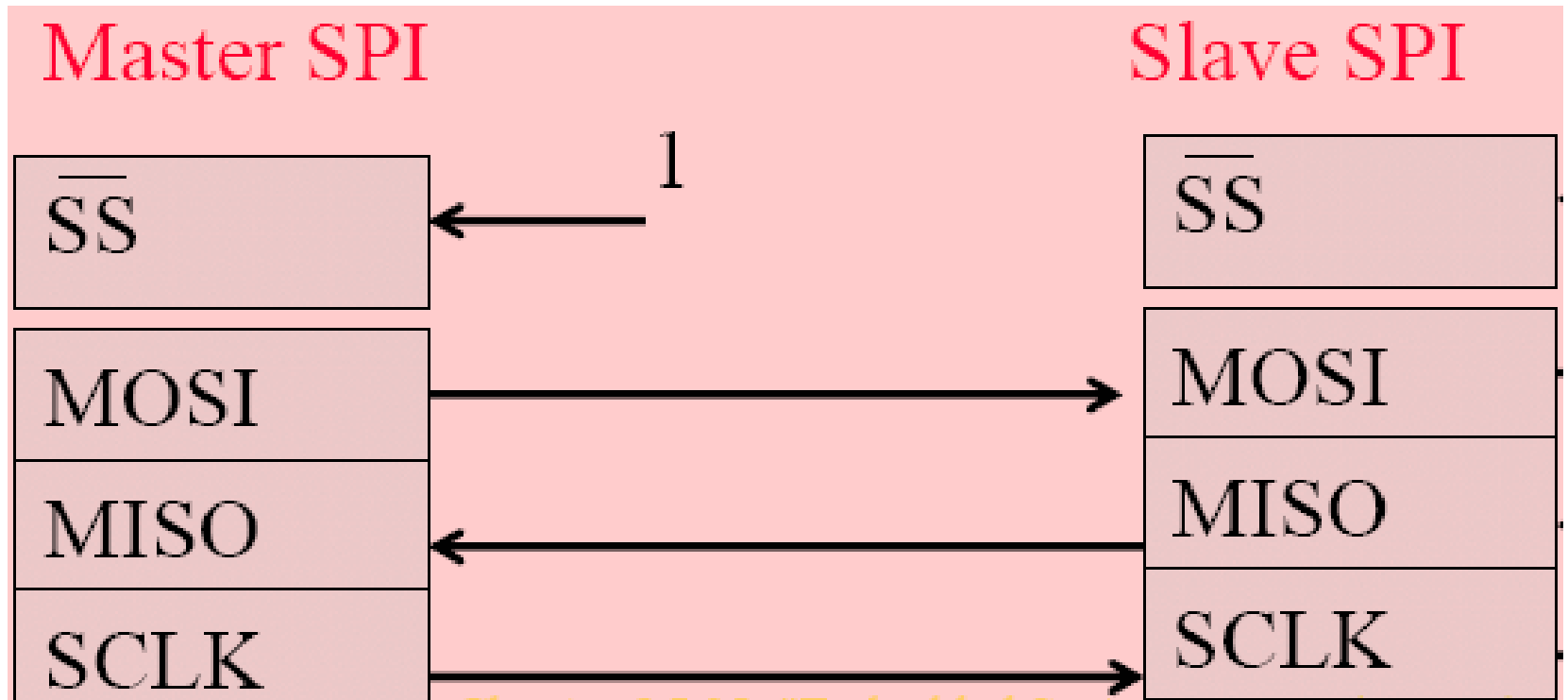
SPI

- Full duplex synchronous communication
- Four signals are used : SS', MISO, MOSI, SCLK
- SS' signal is used to decide whether this device is slave or not (0 is slave)
- SPI device clock rates can range from 30 kHz to 3 MHz

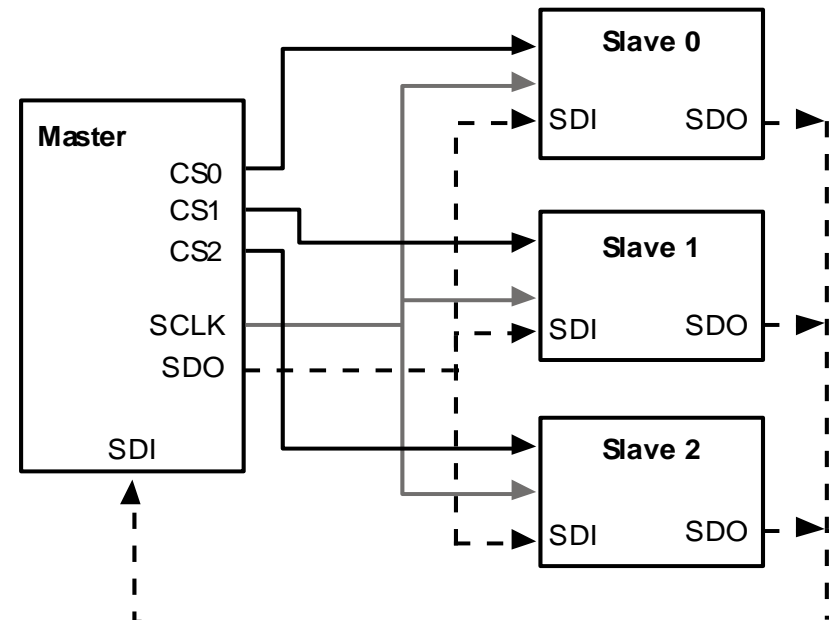
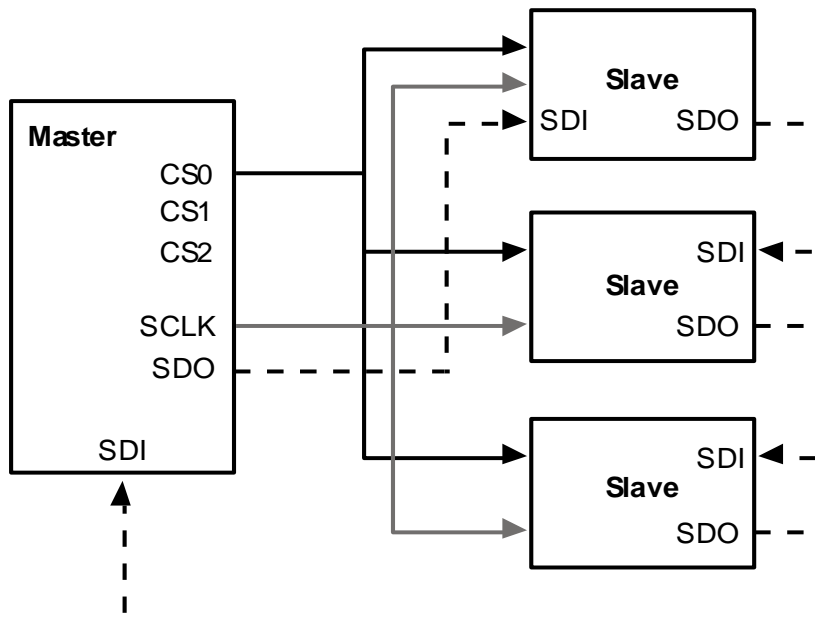
SPI signals



SPI connection



The two SPI slave device configuration options.



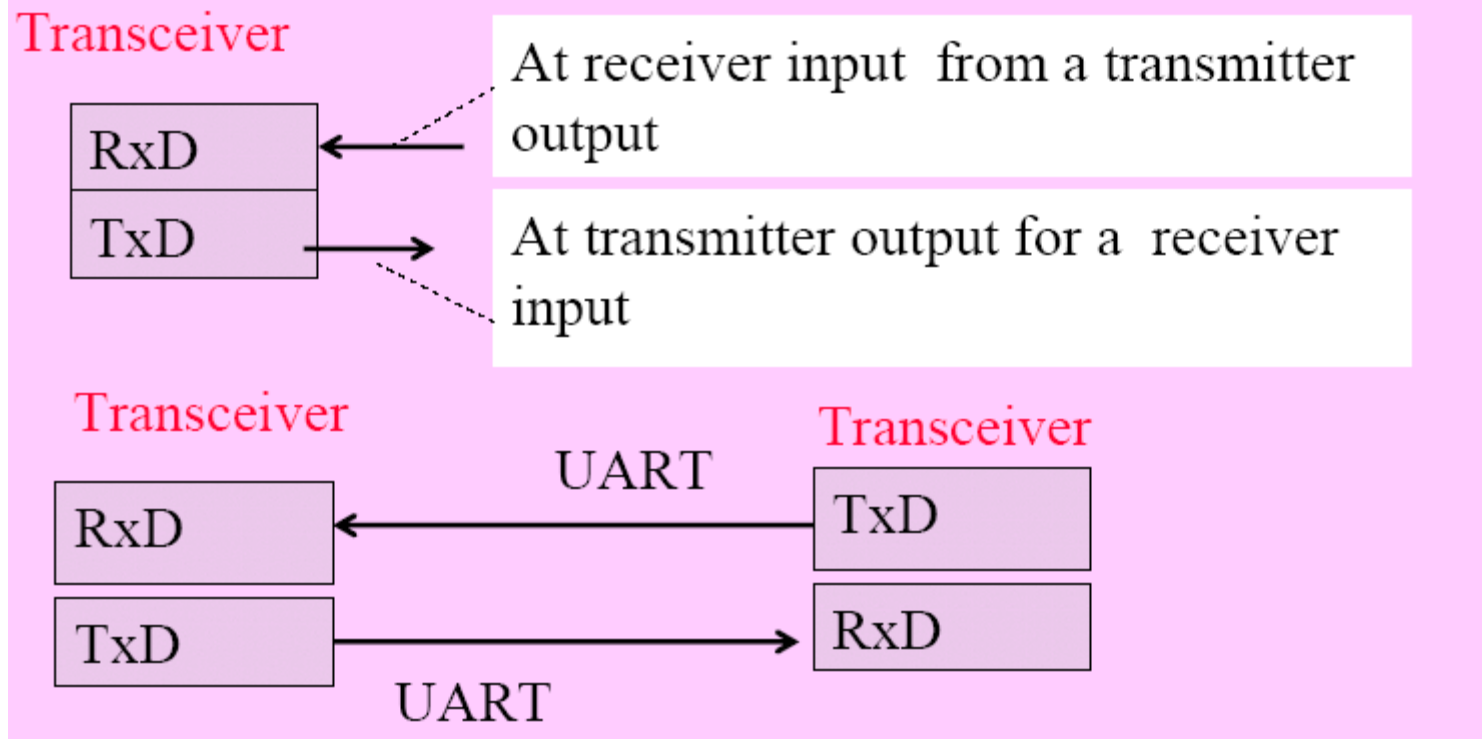
SPI programming

- Program clock rate
- Program master or slave devices
- Program negative/positive edge value

Serial Connect Interface (SCI)

- UART asynchronous port
- Full-duplex mode
- SCI programmable for reception and transmission
- SCI is available in 68HC11, and 68HC12

SCI full duplex signals



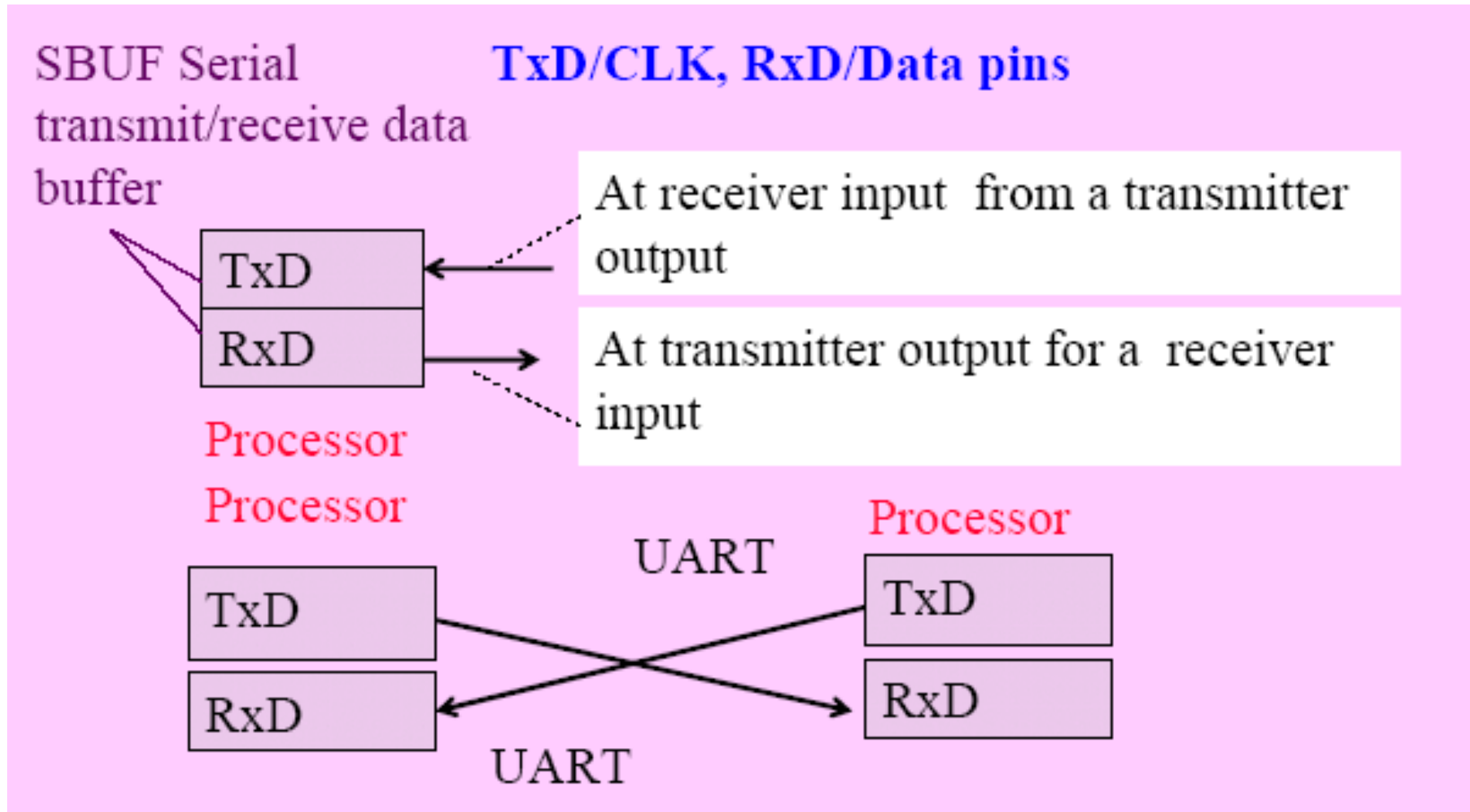
SCI

- SCI baud rates are fixed with prescaling bits
- Baud rate is selectable among 32 possible value
- Support wake-up feature to wake up the other side

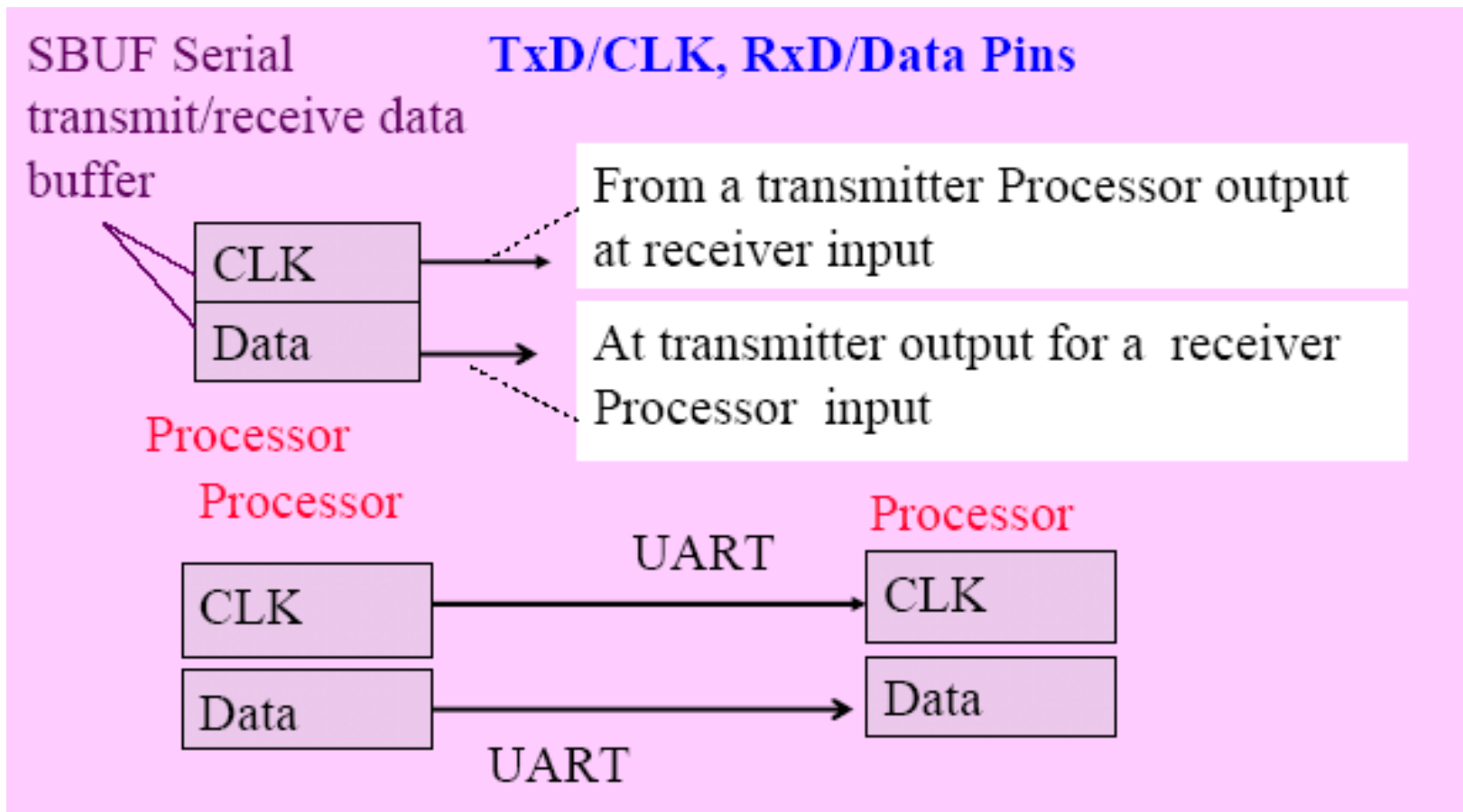
Serial Interface (SI)

- Support both Synchronous and Asynchronous transfer
- For Asynchronous, UART10T and UART11T is supported
- For Synchronous, Universal Synchronous Receiver and Transmitter
- SI is internal serial I/O for 8051

SI transfer in synchronous and asynchronous mode



SI mode 0: Synchronous mode

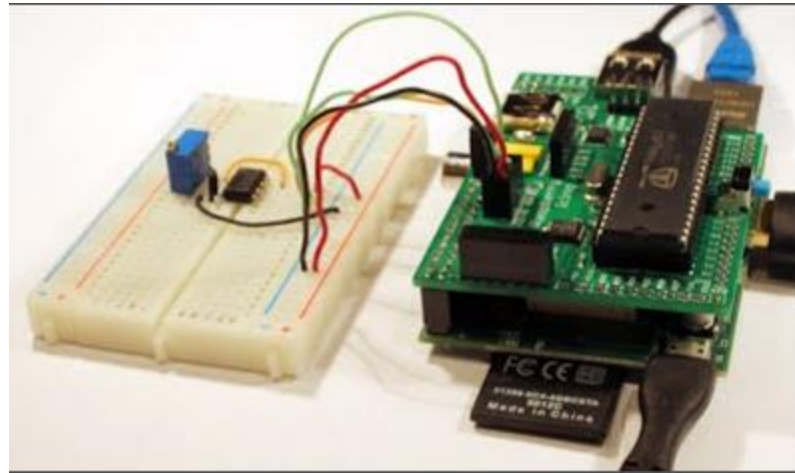


SI modes

- Mode 0: Half duplex synchronous mode
- Mode 1,2,3: Asynchronous mode with different setup for baudrate

SPI Programming

- Simple SPI Setup on Raspberry Pi



SPI Port

- Enable SPI

/etc/modprobe.d/raspi-blacklist.conf

Add '#' in front of the line *spi-bcm2708*. Save the file.

- Reboot

sudo reboot

- Check the device

lsmod

- Add Python library for SPI

sudo apt-get update

sudo apt-get install python-dev



Add Python SPI Wrapper

```
mkdir python-spi
```

```
cd python-spi
```

```
wget https://raw.githubusercontent.com/doceme/py-spidev/master/setup.py
```

```
wget https://raw.githubusercontent.com/doceme/py-spidev/master/spidev\_module.c
```

```
sudo python setup.py install
```

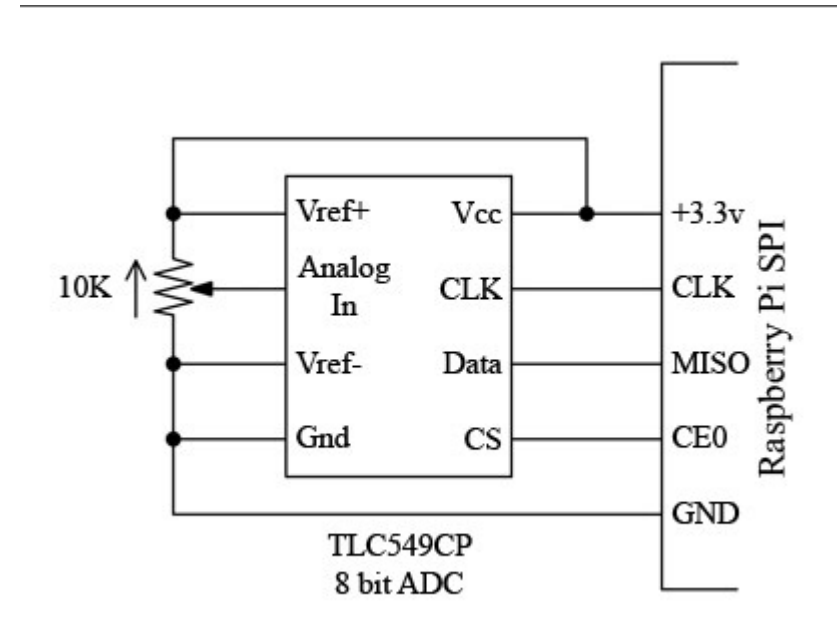

Python Code

```
import spidev  
import time  
  
spi = spidev.SpiDev()  
spi.open(0,0)  
while True:  
    resp = spi.xfer2([0x00])  
    print resp[0]  
    time.sleep(1)
```


Output

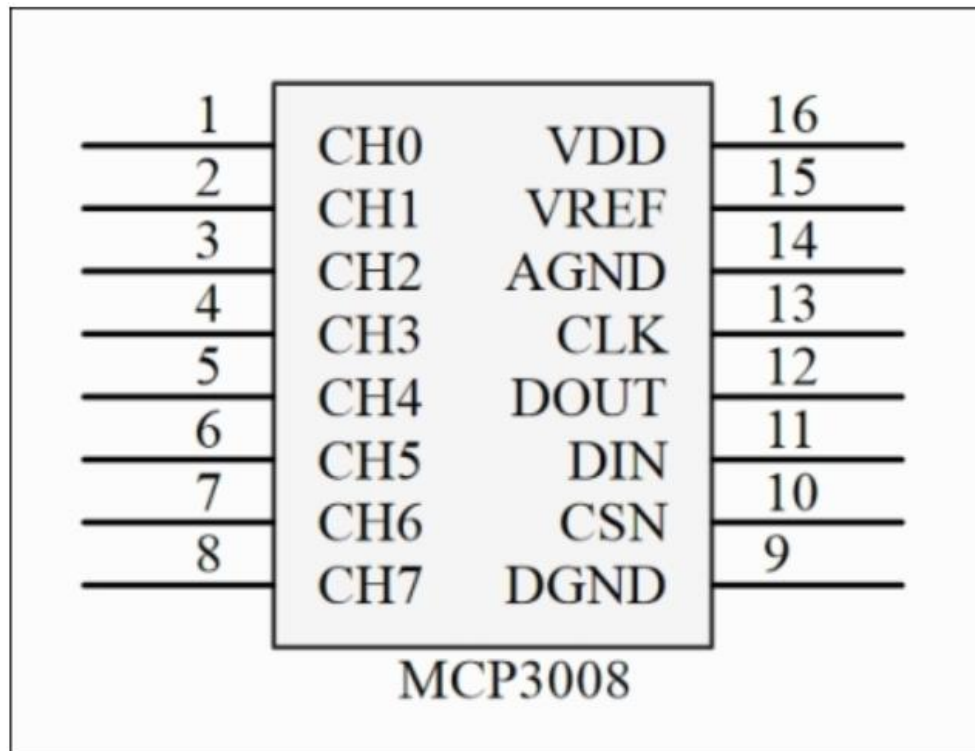
`spi.open(0,0)` will open bus 0, CE0

`spi.xfer2` will keep CE asserted and read data from slave

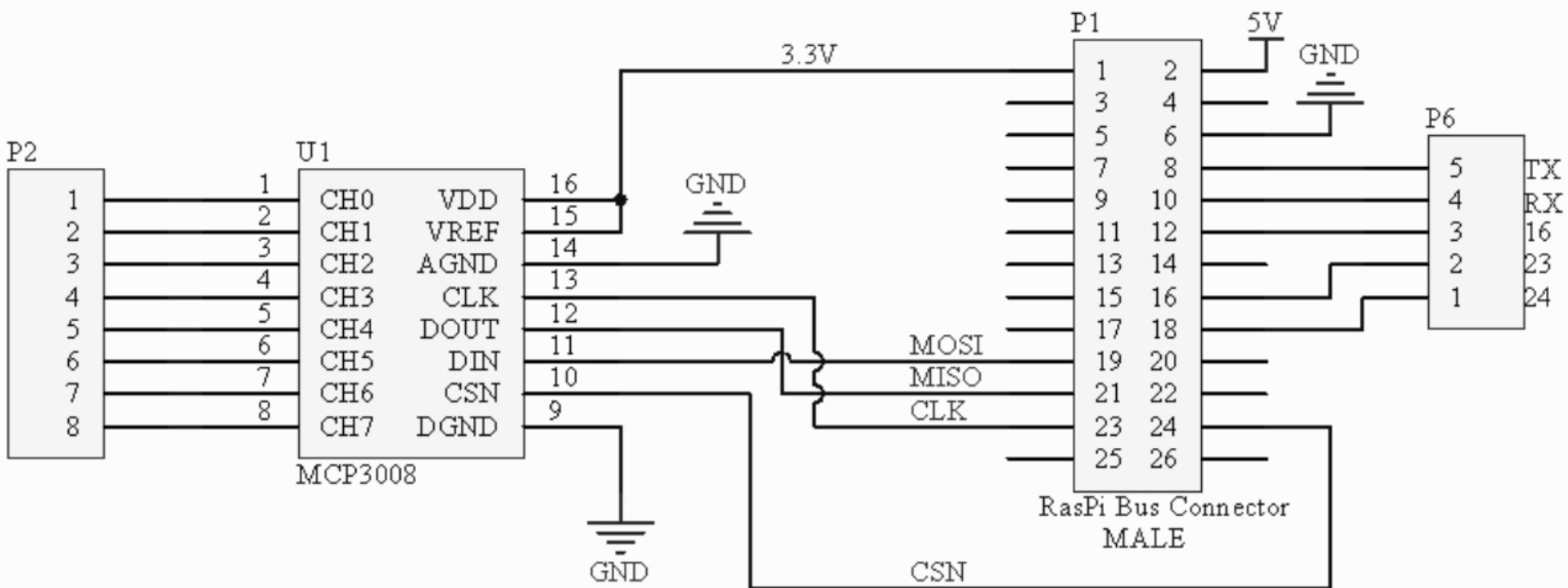


MCP3008

- ADC sensor: 10 bits data reading
- Provide SPI Interface



Circuit Connection



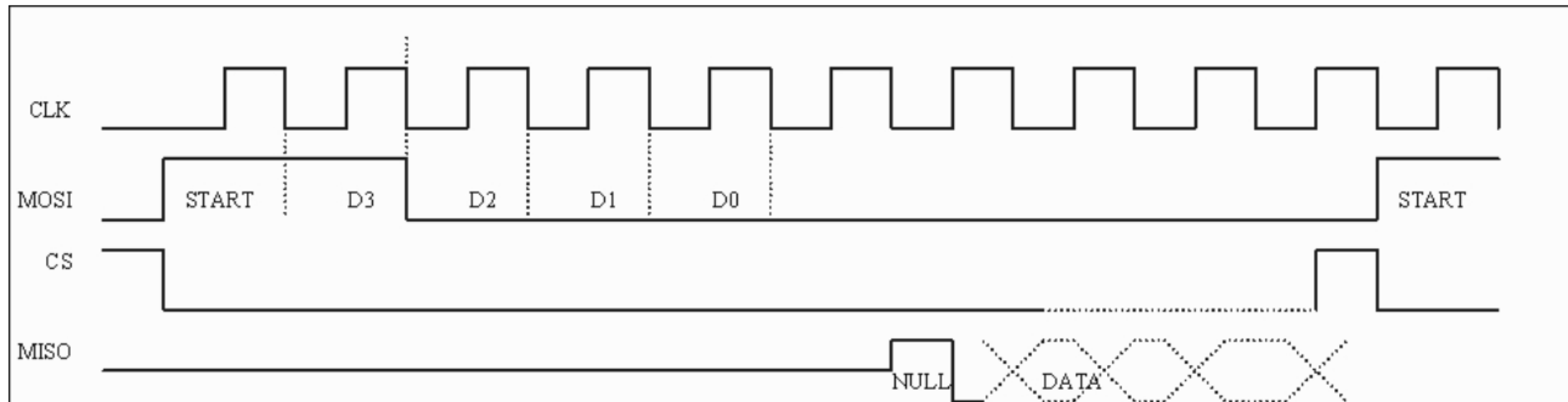
SPI Command

- Data flow:
 - Start bit
 - Control bit (4-bit length)
 - Wait
- `spi.xfer()` is for active high chip
- `spi.xfer2()` is for active low chip

Single-ended mode (D3)	D2	D1	D0	Channel to read
1	0	0	0	1
1	0	0	1	2
1	0	1	1	3
1	0	1	0	4
1	1	0	1	5
1	1	0	0	6
1	1	1	1	7
1	1	1	0	8

xfer2 command

- Send and receive the array of data
- For MCP 3008, the first command is 1.
- Next, we send the channel that we want to read the data
- Finally, we send the blank signal
- Read MCP3008 manual, for more details




```

import spidev
import time
import os

#start the SPI bus by opening the spi port
spi = spidev.SpiDev() spi.open(0,0)
#SPI port 0 opened and Device Chip Select set to 0
#function to read the channels of MCP3008
def readadc(channel):
    value = spi.xfer2([1,(8+channel)<<4,0])
    read = ((value[1]&3) << 8) + value[2]
    return read
while True:
#creating the list for the different values of each channels
    datalist = []
    for i in range(0,8):
        #read channel one by one using range of 0 to 8
        data = readadc(i)
        #append data into the datalist created
        datalist.append(data)
        #convert temperature value from data received
        temperature = ((data * 330)/float(1023))-50
        print temperature

```


Receive Data

ADC Data is 10 bits

- Received data = 00000000 01000010 01100010

value[0] = 00000000

value[1] = 01000010

value[2] = 01100010

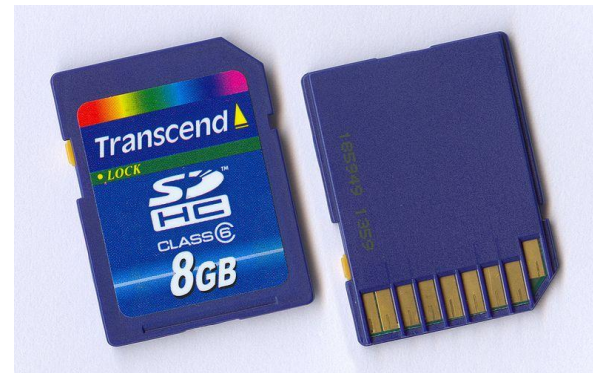
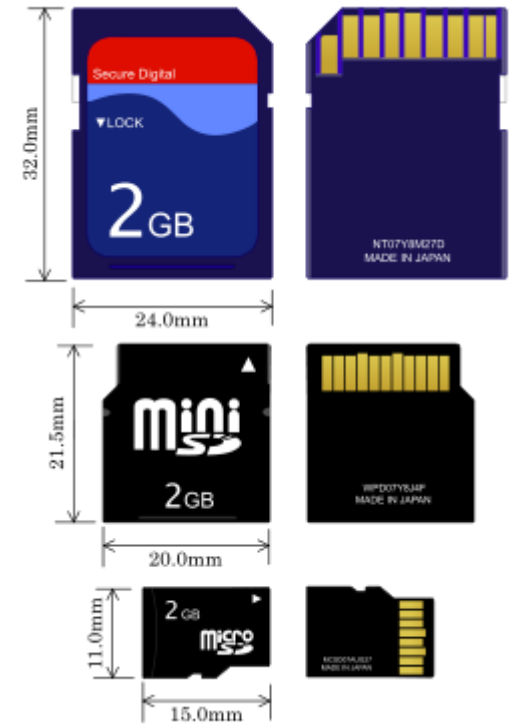
Final data = (value[1] & 3) << 8 + value[2] = 610

Secure Digital Input Output (SDIO)

- From Secure Digital Association (over 700 companies) started from 3 companies in 1999 (Panasonic, Sandisk, Toshiba)
- Create a new memory card called SD format for IO
- SDIO card has become popular in mobile devices, PDA, digital cameras and embedded systems

SD card

- Based on SD technology
- SDHC (Secure Digital High Capacity) is the extension of SD card



SDIO

- SD card size: 0.14cm x 2.4 cm x 3.2 cm
- SDIO host controller inside the card
- Controller may include SPI controller to support SPI mode
- Can have up to 8 logical function inside SDIO host controller
- Provide additional memory in SD format
- Functions include several protocol such as IrDA, Wi-fi, Ethernet, GPS, Bluetooth, RFID

SDIO Function and card

- Up to 8 logical functions
- CRC data check during transfer
- Retransmission on error
- Support data transfer in block of bytes
- Data rate from 20 Mbps to 100 Mbps



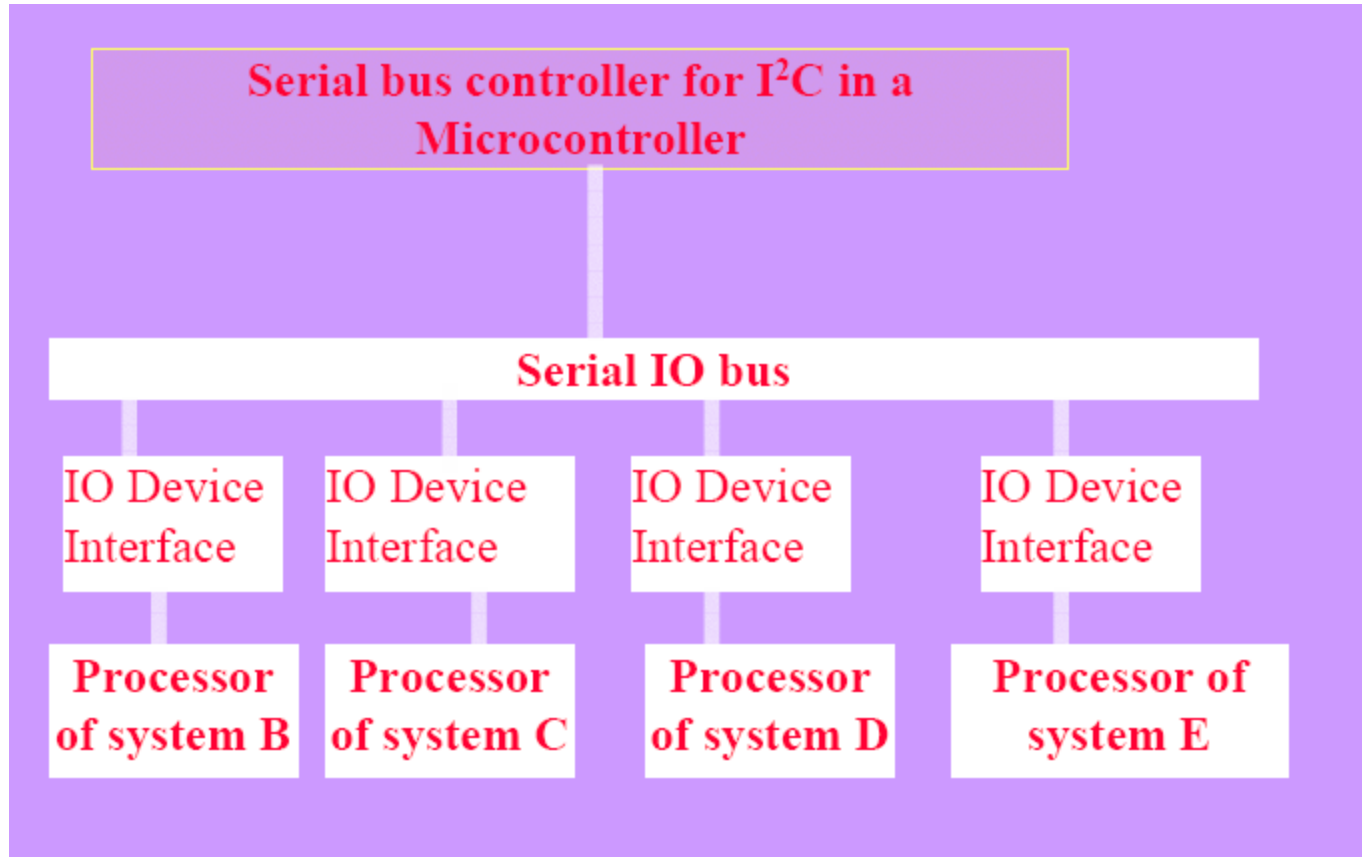
Inter IC (I²C) bus

- Developed by Philips in 1980s
- Low pin count serial interface used to connect chips on a circuit board
- Also used by Intel in the PC motherboard System Management Bus (SMB)
- 7 bit address
- 1 bit Read/Write
- 8 Data bits
- Probably have one in your TV

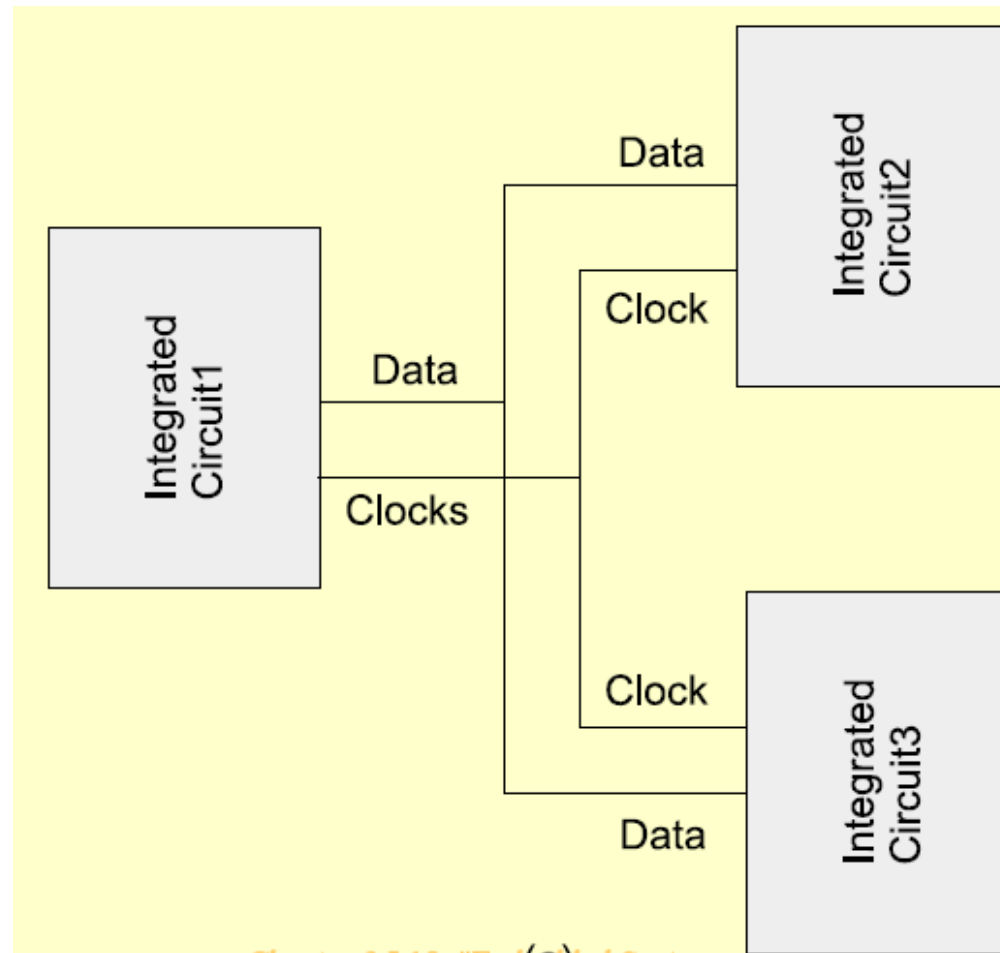
I²C

- ICs mutually network through a common synchronize serial bus
- I²C bus also support I/O interconnect
- 400 kbps up to 2 meters and 100 kbps for longer distances
- Version 2 support up to 2.4 MHz
- Version 4 support 5MHz Ultra Fast Mode

Serial I/O I²C bus



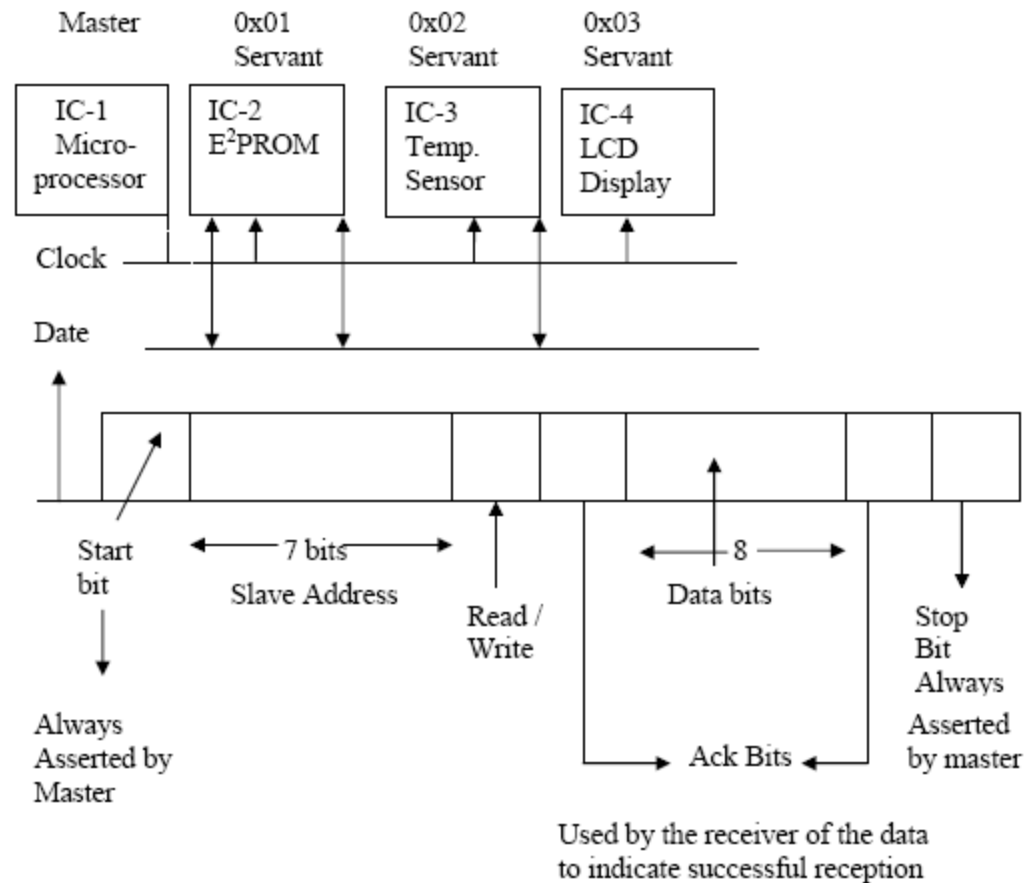
Distributed I²C bus connection



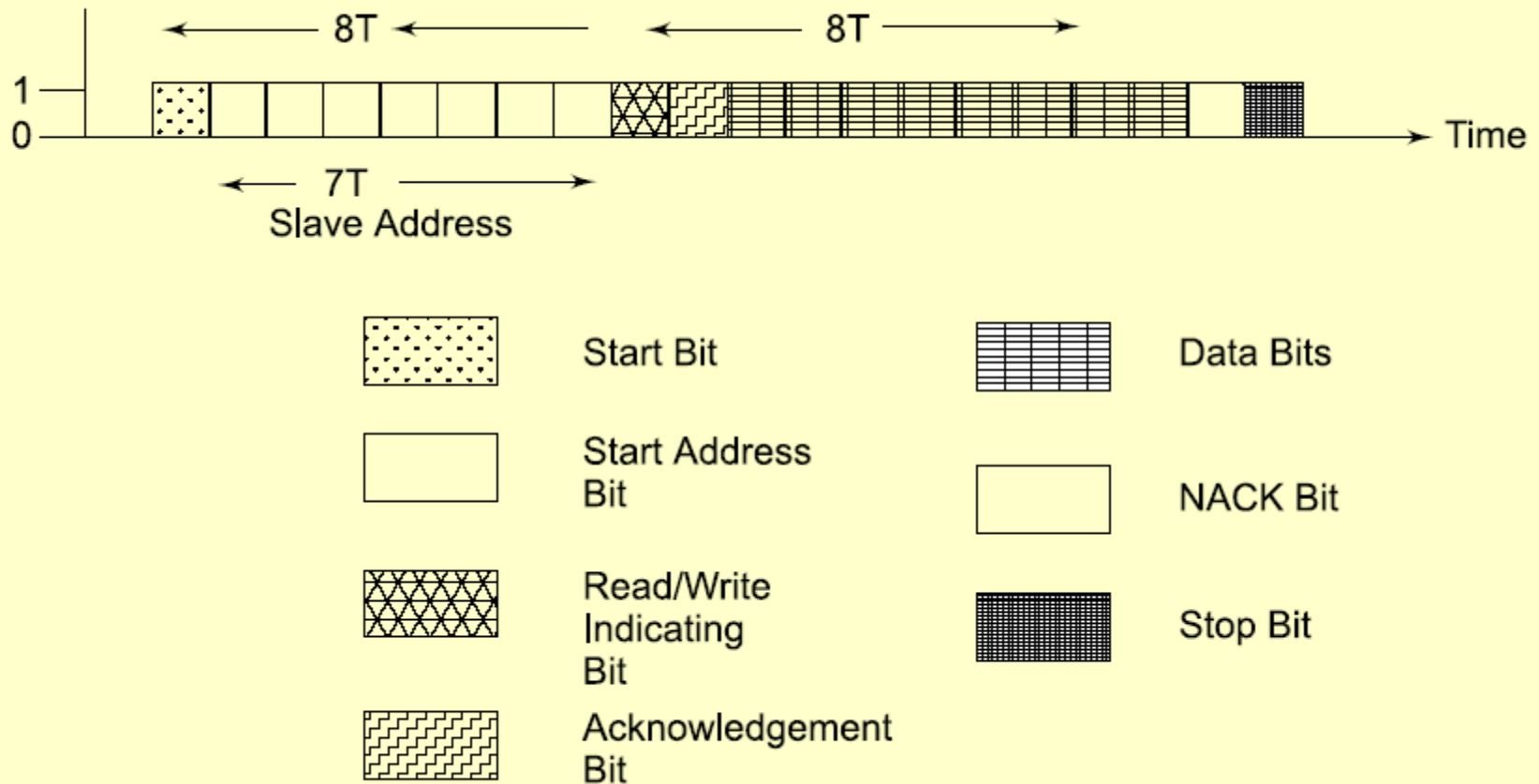
I²C bus protocol

- The bus has two lines – one is for clock and another one is bidirectional data
- It is a Master and Slave system
- Master can address 127 slaves at an instances
- Number of Masters can be connected on the bus
- However, only one Master can be active at a time (use bus arbitration)
- Each devices has 7-bit address
- Master controls the clock, but a slave can slow clock down by drive the clock low (clock stretching)

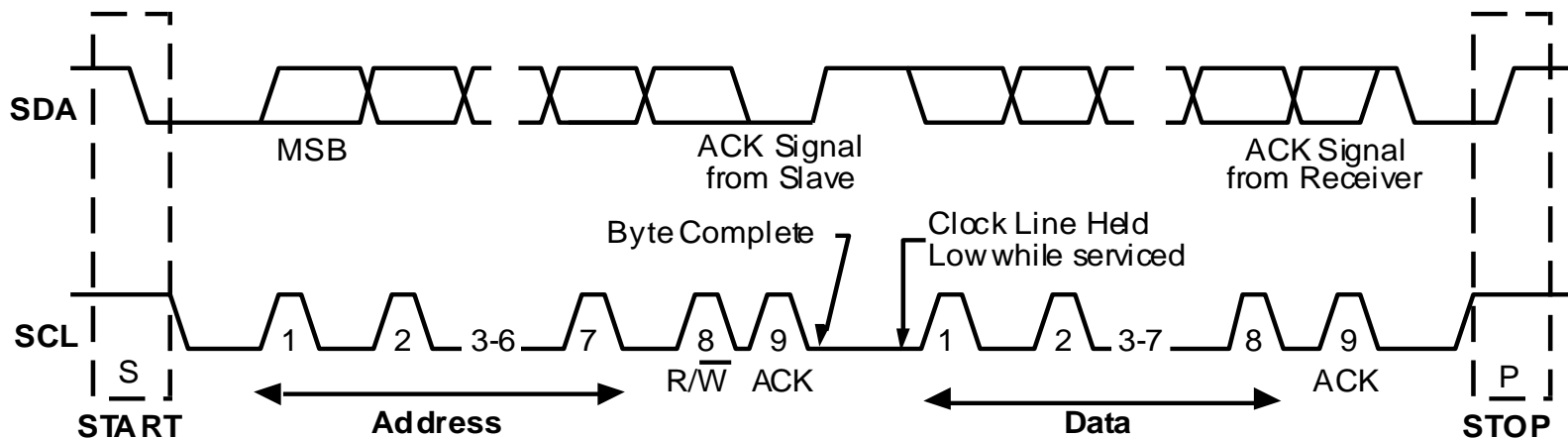
I²C communication



I²C bus protocol



I²C interface serial transmission of an 8-bit data value



I²C bus field

- First field – similar to start bit in UART
- Second field – 7 bit address field
- Third field – 1 bit control field define read/write'
- Fourth field – 1 bit control field whether the present data is an acknowledgement from slave
- Fifth field – 8 bit data field (MSB first)
- Sixth field – 1 bit control field NACK (negative acknowledgement) to tell slave that the acknowledgement is not required
- Seventh field -1 bit stop bit similar to UART

Microprocessor sends data

1. The microprocessor sends an I2C start sequence.
2. The microprocessor serially clocks out the 7-bit I2C address of the slave with the R/W' bit set Low (i.e., write operation)
3. The slave set ACK signal
4. The microprocessor serially clocks out the 8-bit data value(s).
5. If the ACK is used to tell that the receiver get the data
6. The microprocessor sends a stop sequence.
(The microprocessor can keep sending the data as it wishes, bus owner)

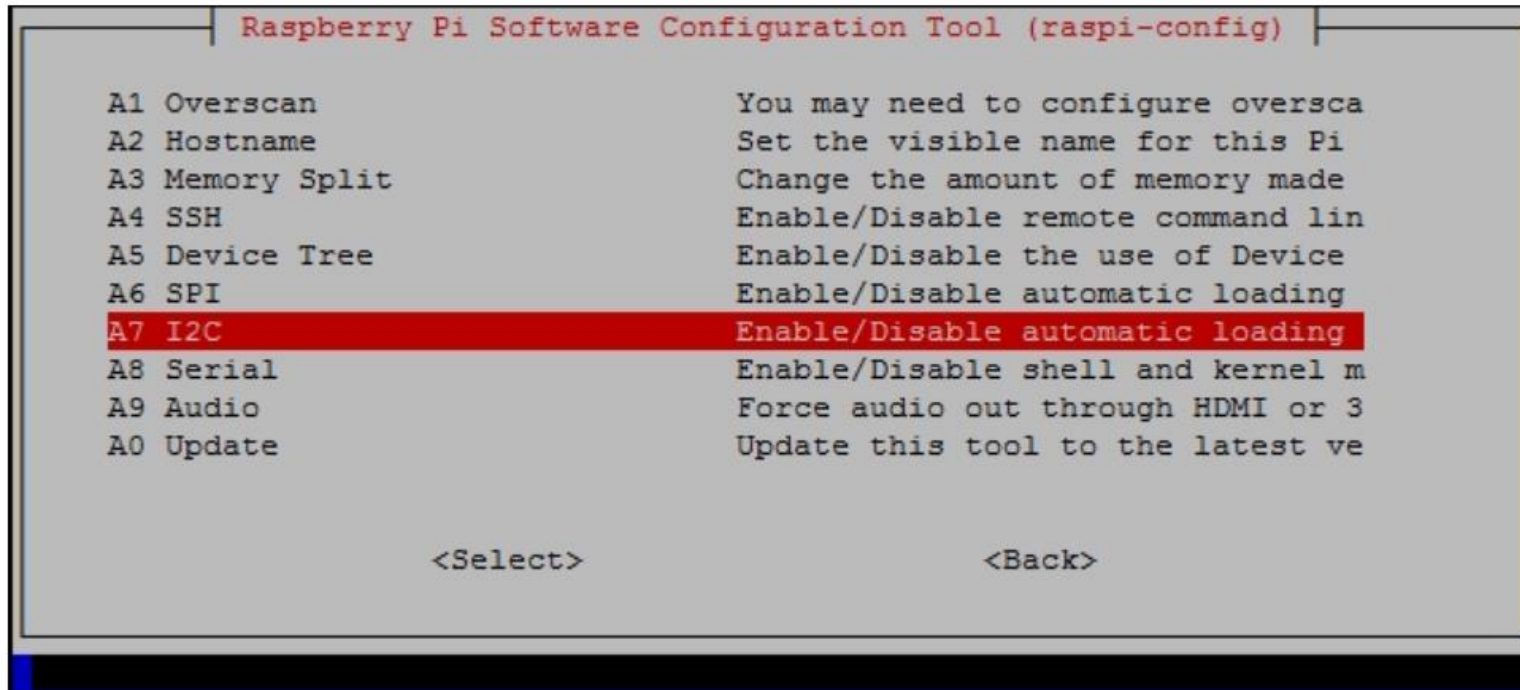
Microprocessor reads data

1. The microprocessor sends a start sequence.
2. The microprocessor serially clocks out the 7-bit I2C address of the slave with the R/W' bit set High.
3. The slave set ACK signal
4. The microprocessor serially clocks in the 8-bit data value(s) off the bus from the slave.
5. If the ACK is used to tell that the Master get the data, and the slave should keep sending next data
6. The microprocessor sends a stop sequence.
(The microprocessor can keep reading the data as it wishes. If the ACK is set to 0, the slave will stop sending the data)

Enable I2C on Pi

- Enabling the I2C Port

sudo raspi-config



Enable I2C on Pi

- Enabling the I2C on old Raspbian

in /etc/modules Add lines

i2c-bcm2708

i2c-dev

Edit module blacklist in /etc/modprobe.d/raspi-blacklist.conf

Add # to i2c-bcm2708

- Install I2C utilities

sudo apt-get install python-smbus i2c-tools

sudo reboot

Checking for connected devices

- Check the status for I2C0 or I2C1 port

`sudo i2cdetect -y 0`

//or

`sudo i2cdetect -y 1`

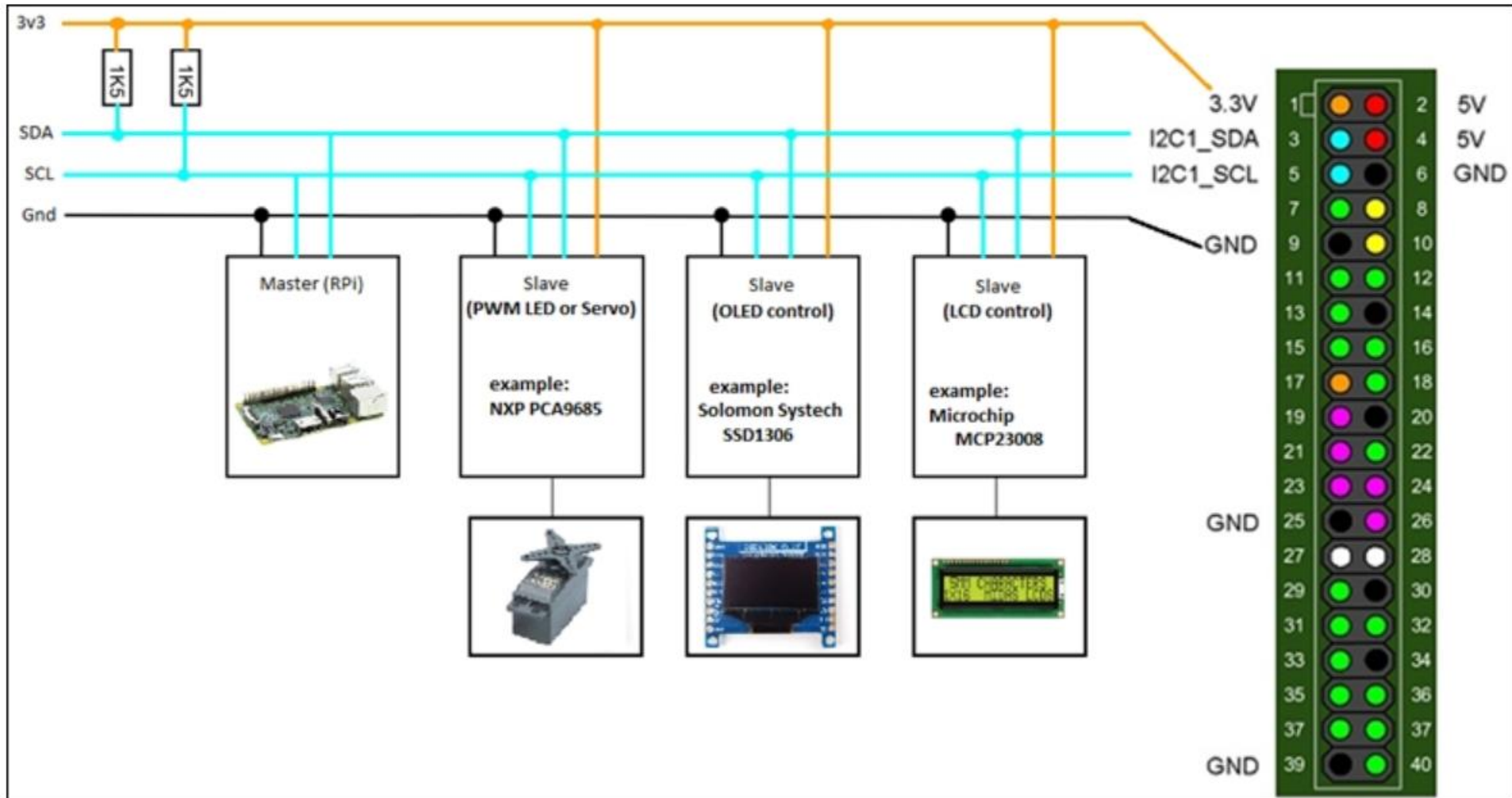
7 bits I2C address of all
found devices will be shown

```
pi@raspberrypi:~$ sudo i2cdetect -y 0
   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- --
pi@raspberrypi:~$ sudo i2cdetect -y 1
   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- 48 -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- --
pi@raspberrypi:~$
```

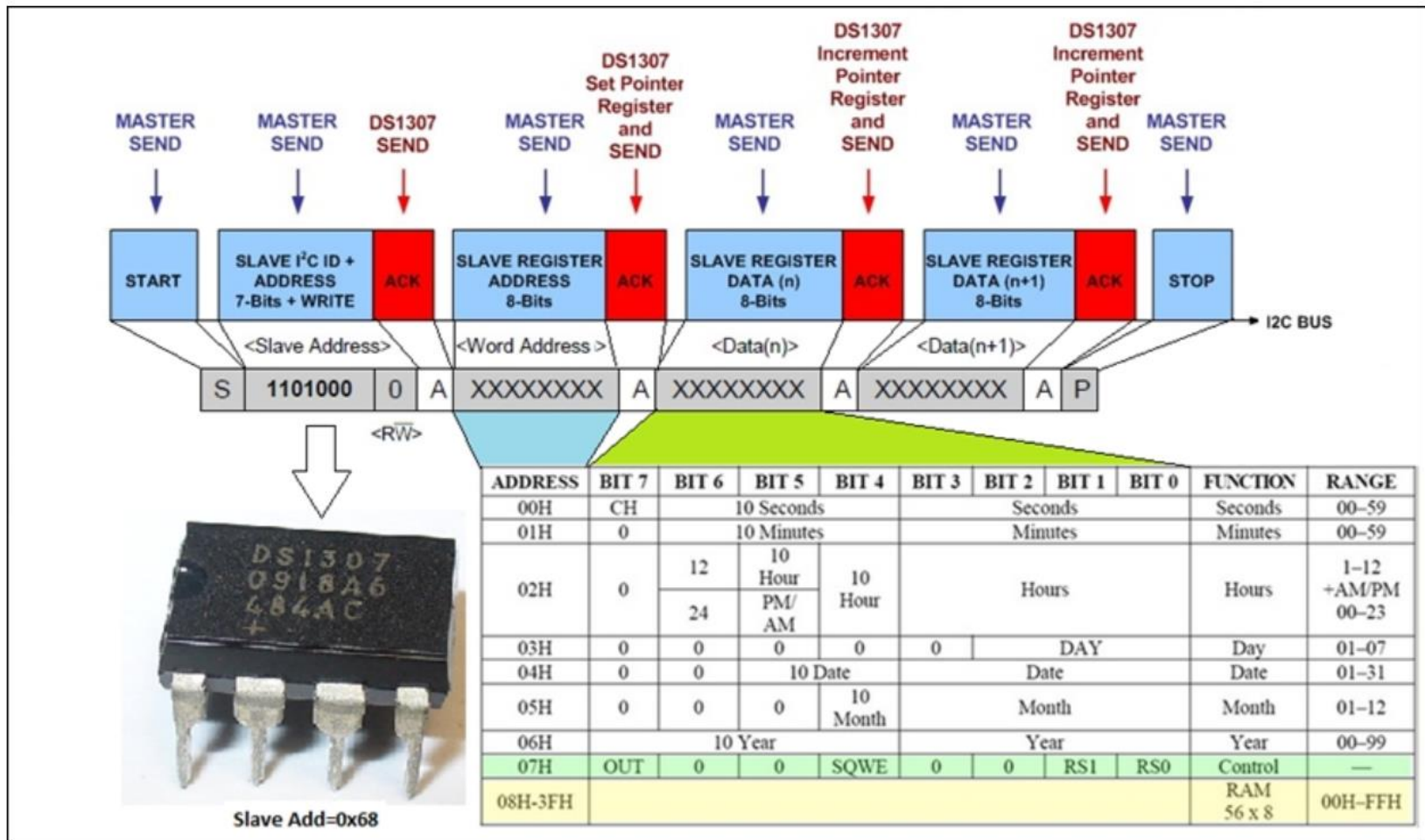

Example of the code

```
#!/usr/bin/python
import smbus
bus = smbus.SMBus(1)
# 0 = /dev/i2c-0 (port I2C0), 1 = /dev/i2c-1 (port I2C1)
# DEVICE_ADDRESS = 0x15
#7 bit address (will be left shifted to add the read write bit)
DEVICE_REG_MODE1 = 0x00
DEVICE_REG_LEDOUT0 = 0x1d
#Write a single register
bus.write_byte_data(DEVICE_ADDRESS, DEVICE_REG_MODE1,
0x80)
#Write an array of registers
ledout_values = [0xff, 0xff, 0xff, 0xff, 0xff, 0xff]
bus.write_i2c_block_data(DEVICE_ADDRESS,
DEVICE_REG_LEDOUT0, ledout_values)
```


Raspberry Pi Connection

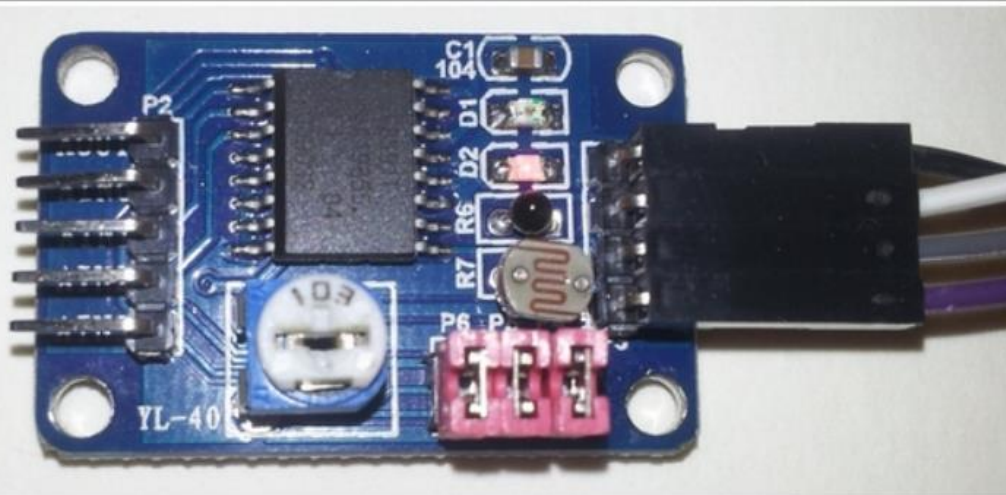


Example of I²C message



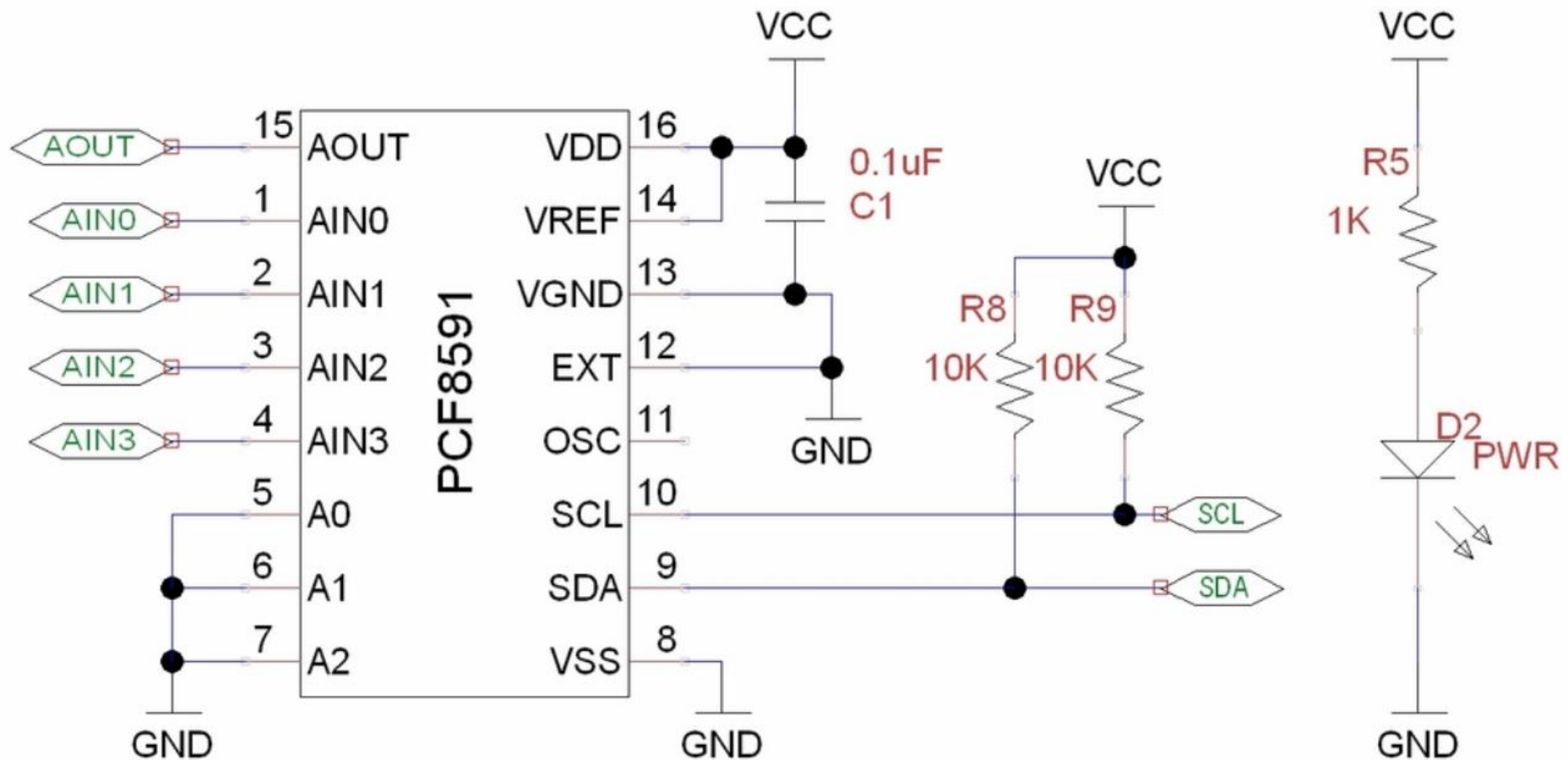
Example: ADC using I²C

- ADC device (4 channels)



I ² C Device	Raspberry Pi GPIO		I ² C Device
VCC	1	2	
SDA	3	4	
SCL	5	6	GND

Circuit Connection




```

#data_adc.py
import wiringpi2
import time
DEBUG=False
LIGHT=0;TEMP=1;EXT=2;POT=3
ADC_CH=[LIGHT,TEMP,EXT,POT]
ADC_ADR=0x48
ADC_CYCLE=0x04
BUS_GAP=0.25
DATANAME=["0:Light", "1:Temperature", "2:External", "3:Potentiometer" ]

class device:
    # Constructor:
    def __init__(self,addr=ADC_ADR):
        self.NAME = DATANAME
        self.i2c = wiringpi2.I2C()
        self.devADC=self.i2c.setup(addr)
        pwrup = self.i2c.read(self.devADC) #flush powerup value
        if DEBUG==True and pwrup!=-1:
            print("ADC Ready")
        self.i2c.read(self.devADC) #flush first value
        time.sleep(BUS_GAP)
        self.i2c.write(self.devADC,ADC_CYCLE)
        time.sleep(BUS_GAP)
        self.i2c.read(self.devADC) #flush first value

    def getName(self):
        return self.NAME

    def getNew(self):
        data=[]
        for ch in ADC_CH:
            time.sleep(BUS_GAP)
            data.append(self.i2c.read(self.devADC))
        return data

```



```
def main():  
    ADC = device(ADC_ADR)  
    print (str(ADC.getName()))  
    for i in range(10):  
        dataValues = ADC.getNew()  
        print (str(dataValues))  
        time.sleep(1)  
  
if __name__ == '__main__':  
    main()
```

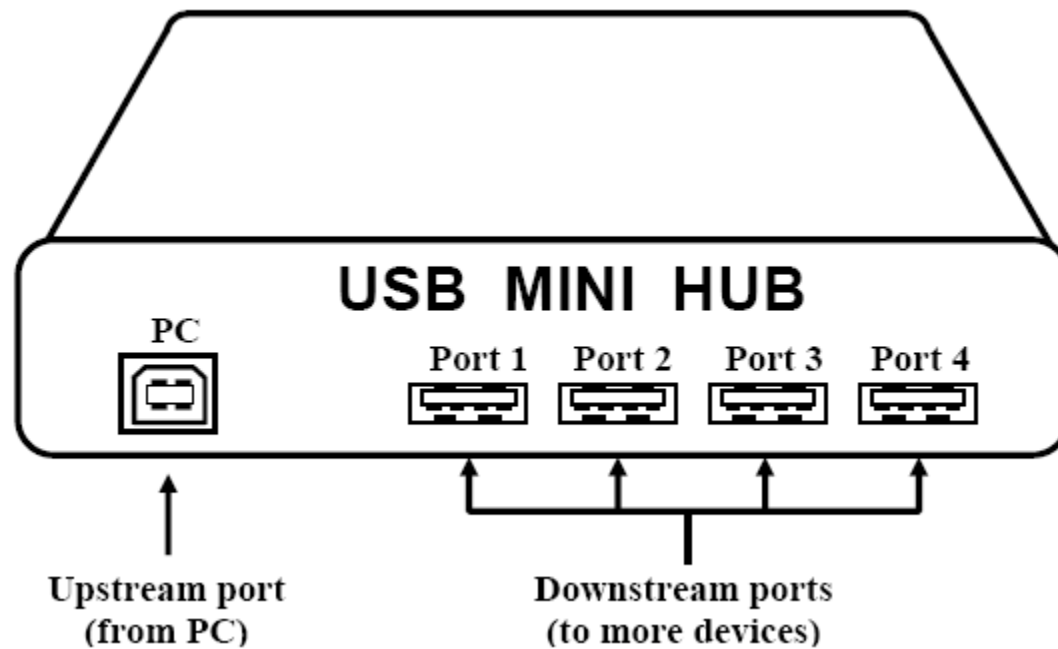

Questions?

- What's the speed and distance of I²C (version 1)?
- I²C is?
 - Master-slave -Peer-to-Peer -Broadcast
- How can the device initiate the talk to the master?

Universal Serial Bus (USB)

- Developed by IBM, Intel, Microsoft, DEC, NEC, and Northern telecom
- Serial Synchronous communication
- Seven bit address code
- Connecting from the device to the host computer is called upstream port
- Connecting from host computer to the device is called downstream port
- Tree (star) topology is used when the host is the root of the tree, a hub is used to make a sub-branch of the tree
- www.usb.org

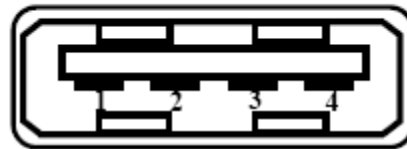
USB Mini Hub



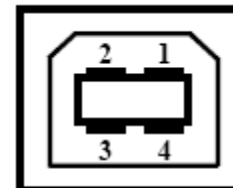
USB

- There are two interface type: type A and type B
- Type A: plug on the upstream end (PC)
- Type B: plug on the downstream end (device)
- Both types are polarized (cannot be inserted in the wrong way)
- Cannot connects two PCs with USB

USB interface



Type A socket
(from front)



Pin connections	
Pin No.	Signal
1	+ 5V Power
2	- Data
3	+ Data
4	Ground

Universal Serial Bus (USB)

- 1-bit serial sent over twisted pair wire
- Data transfer is in term of packet
- USB Packets contain address and data
- Up to 127 devices on bus
- Special purpose low-cost USB single chip microcontrollers are used.
- Length is limited to a few feet – a bus and not a network
- Data D+ or D- line can operate up to 3.3 volt

USB applications

- Flash memory cards
- Pen-like memory devices
- Digital cameras
- Printers
- Mouse-device
- Pocket-PC
- Video games
- Scanners

USB

- Serial transmission between host and serial devices
- The data transfer is of four types
 - Controlled data transfer – Bulk data transfer
 - Interrupt driven data - Iso-Synchronous transfer
- Maximum 127 devices can connect a host

USB

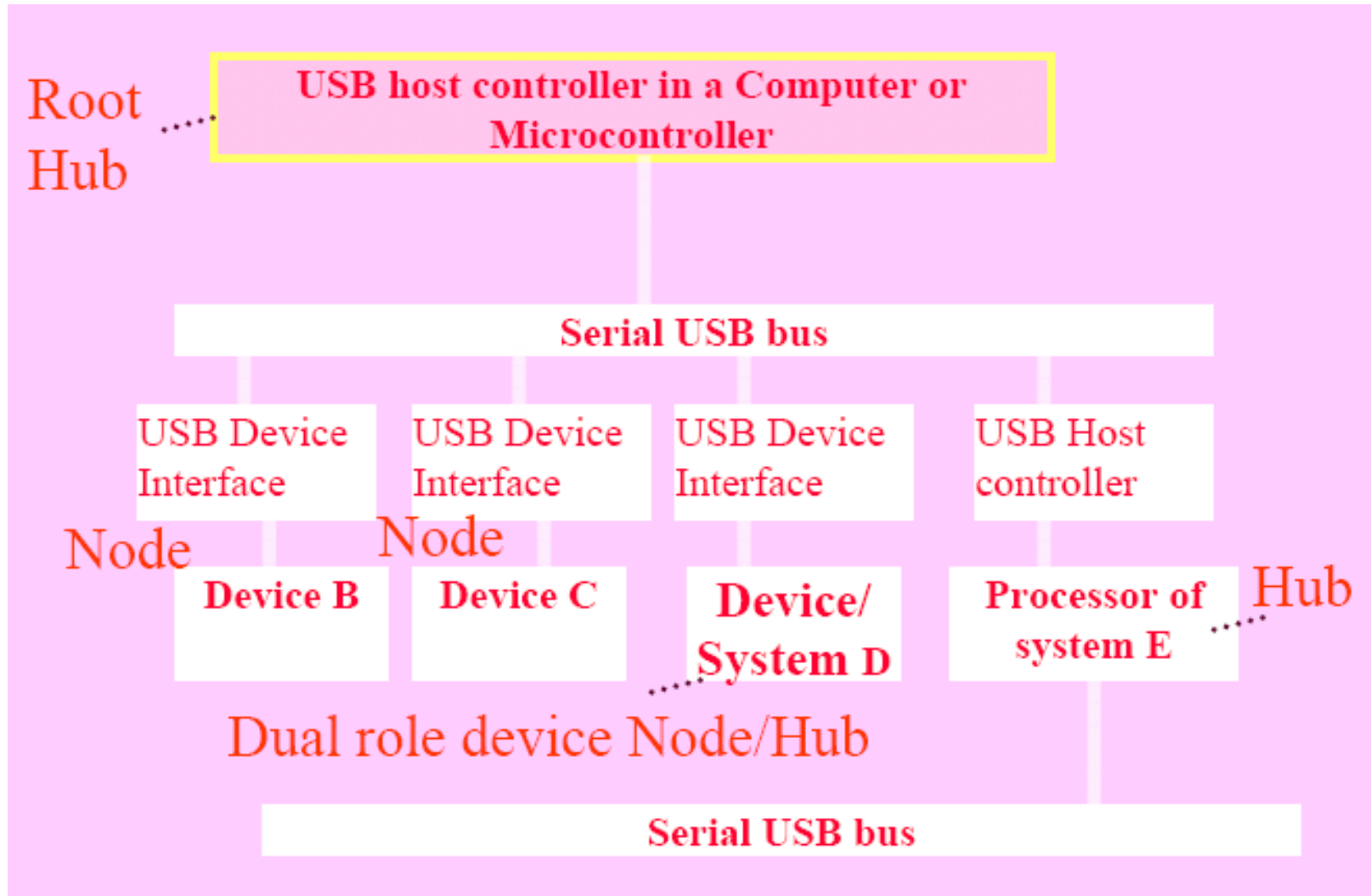
Standards:

- USB1.1 (a low speed 1.5 Mbps 3 meter channel and high speed 12 Mbps 25 meters)
- USB 2.0 (high speed 480 Mbps 25 meter channel)
- USB 3.1 (can transfer up to 10 Gbit/s)
- Wireless USB (high speed 480 Mbps 3m)

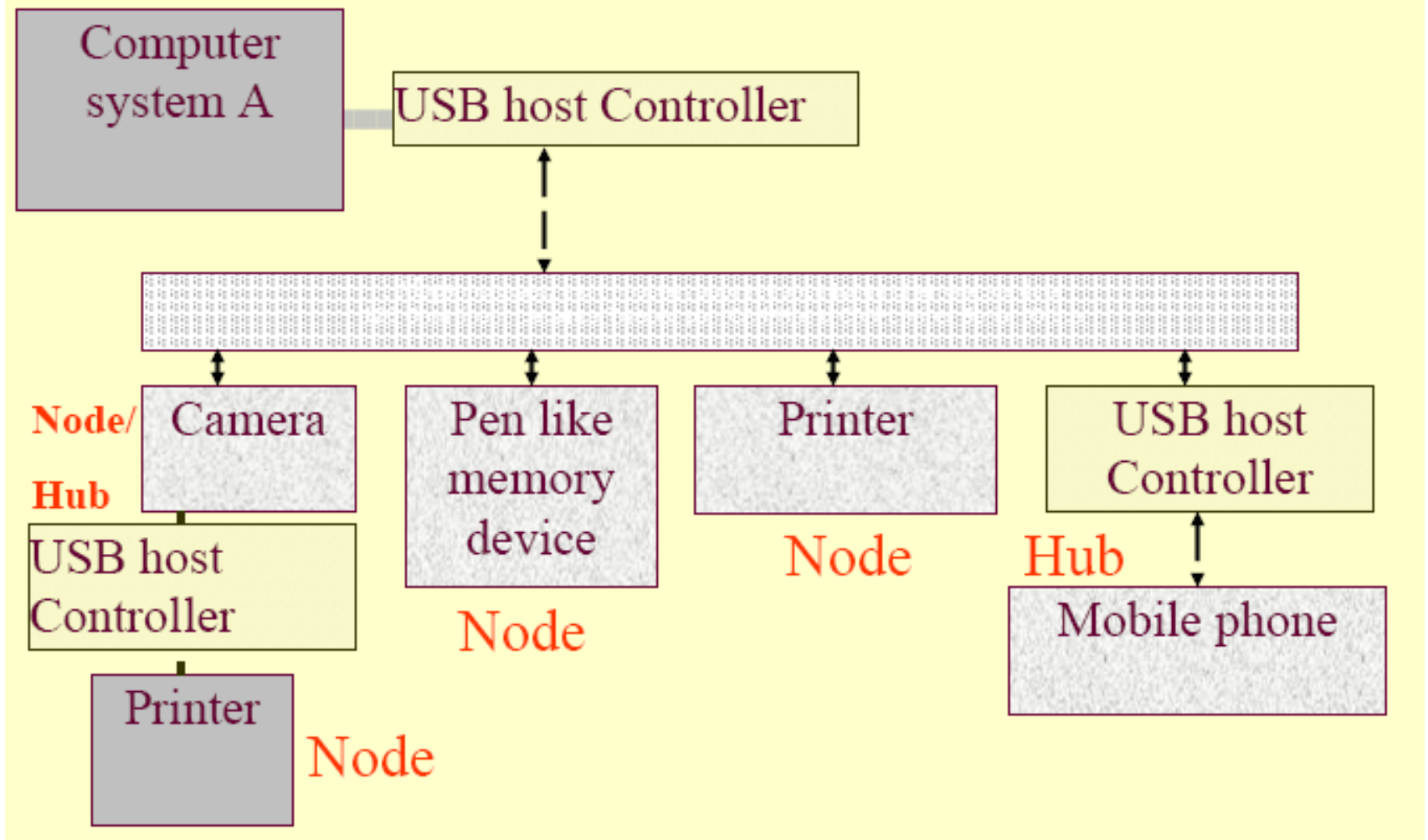
Host connection to the devices or node

- Using USB port driving software and host controller
- Host computer or system has a host controller, which connect to a root hub
- A hub is one that connects to other nodes or hubs
- A tree-like topology

USB bus



USB bus



Dual Role Devices (DRDs)

- A device can be used as either USB host or USB device.
- For example, a USB camera is USB host when connects with a printer and is a USB device when connects with a computer

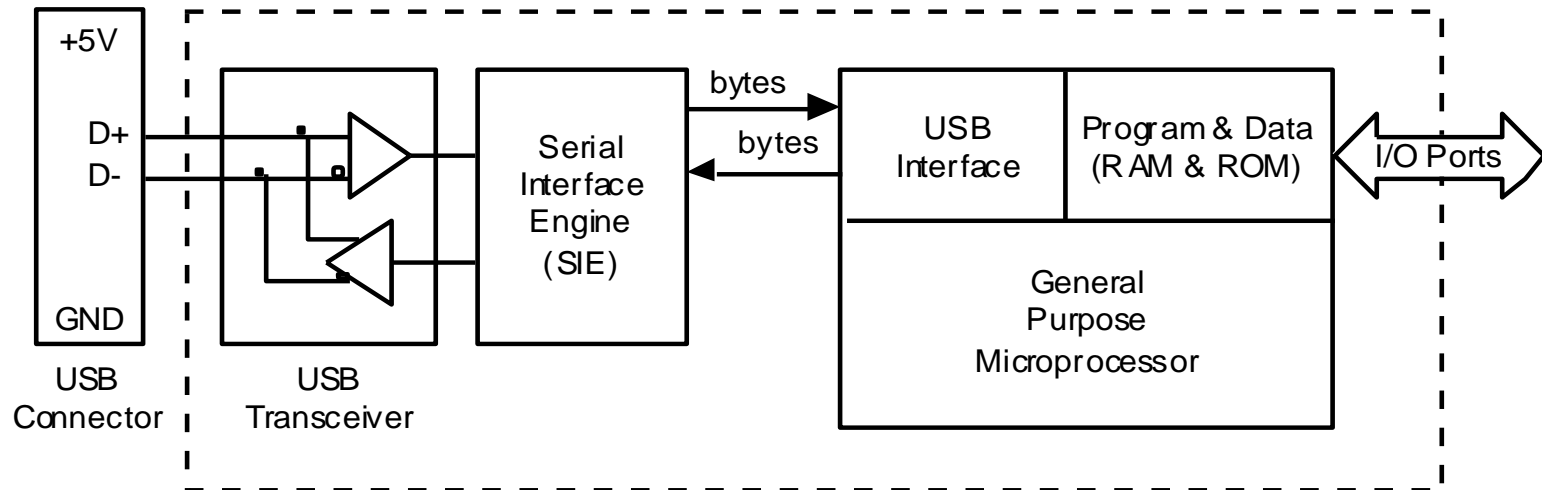
Hub connection

- The root hub connects to the hub and node at level 1
- A hub at level 1 connects to the hub and node at level 2
- Only the nodes are presented at the last level
- USB hubs can have up to 5 levels

USB device features

- Can be hot plugged (attached), configured and used, reset, reconfigured and used
- Bandwidth sharing with other devices: Host schedules the sharing of bandwidth among the attached devices at an instance
- Can be attached and reattached
- Attaching and detaching USB device or host without rebooting
- A device can be bus-powered or self-powered

A low-cost USB Microcontroller is typically used in USB Devices



USB Data Rates

- USB 2.0 – Up to 480 Mbps
- USB 1.0 – 12 Mbps and 1.5 Mbps
- USB 2.0 supports USB 1.0 devices
- Needed higher USB 2.0 data rates for external drives and video
- USB 3.0 is 10x USB 2.0

USB Enumeration

- Hot pluggable – longer power and ground (outer) pins on USB connector
- At power up, software (OS) can read product and vendor ID codes (PID/VID)
- Codes are used to locate device driver
- The required Device Driver is loaded
- The new device is assigned its USB address (0..127) with a command

USB protocol

- USB bus cable has 4 wires: 1 for 5 V, 2 for twisted pairs, 1 for ground
- Non Return To zero (NRZ) signal is used
- Synchronization clock encoded by inserting synchronous code (SYNC) field between each USB packet
- A polled bus

USB protocol

- Host controller regularly polls the presence of the device
- The device does a handshaking through a handshaking packet
- A CRC field in a data packet permits error detection
- Least significant bit first
- Changing voltage on Data signal can change the speed

USB Data format

Four different format support:

- Bulk asynchronous mode: transfer non-critical data
- Isochronous mode: critical data
- Interrupt: from device to host
- Control: from host to control bus and send request to devices

USB Packets

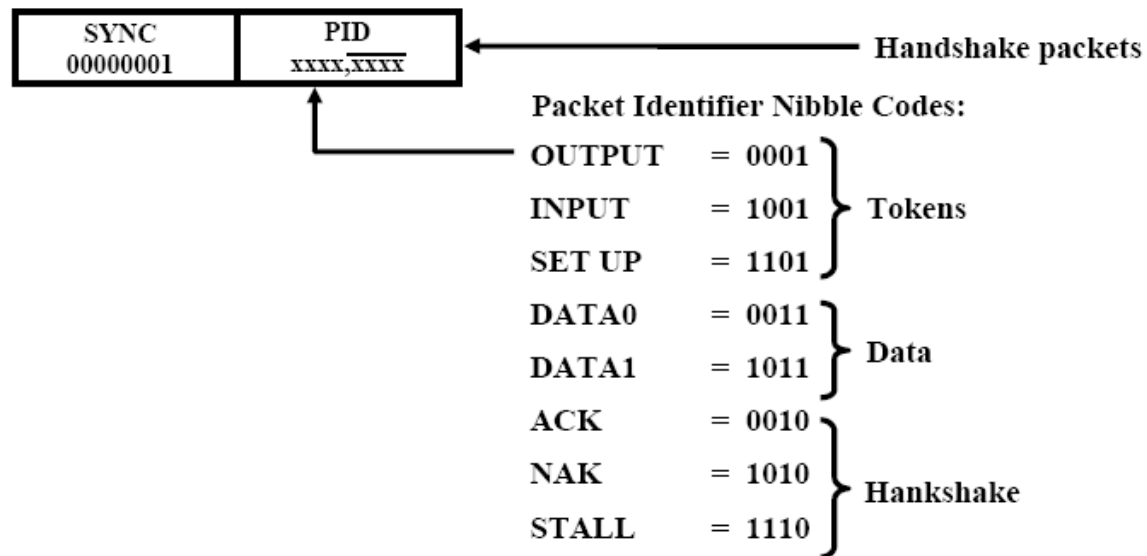
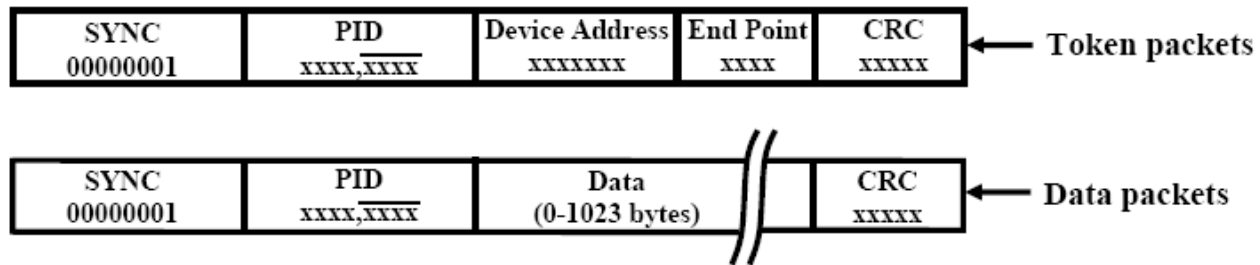
- Contain a packet type
- USB Address and Endpoint fields
- Each device can have several endpoints
- Endpoint is a hardware buffer between the USB Serial Interface Engine (SIE) and the firmware running on the microcontroller (typically 8-64 bytes)
- CRC error code checked and Ack sent back

USB Data Packet

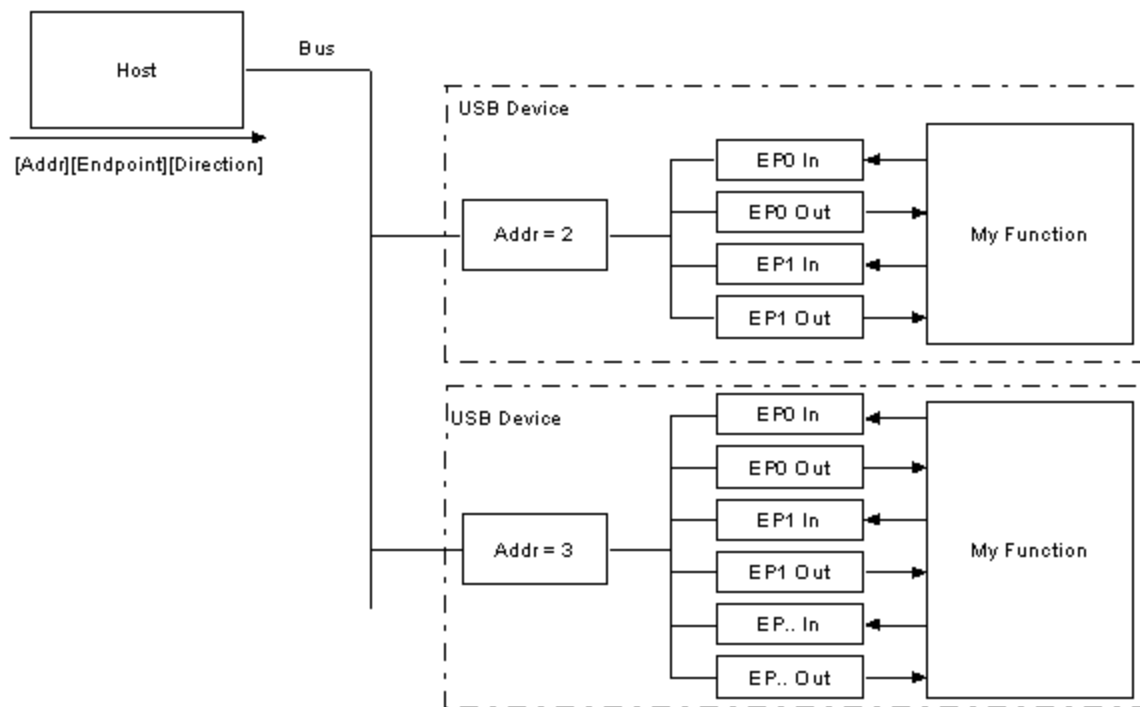
- Sync bit
- Packet Identifier (PID)
- Device address
- Data
- Cyclic Redundancy Check (CRC)

Data is transmitted LSB bit first

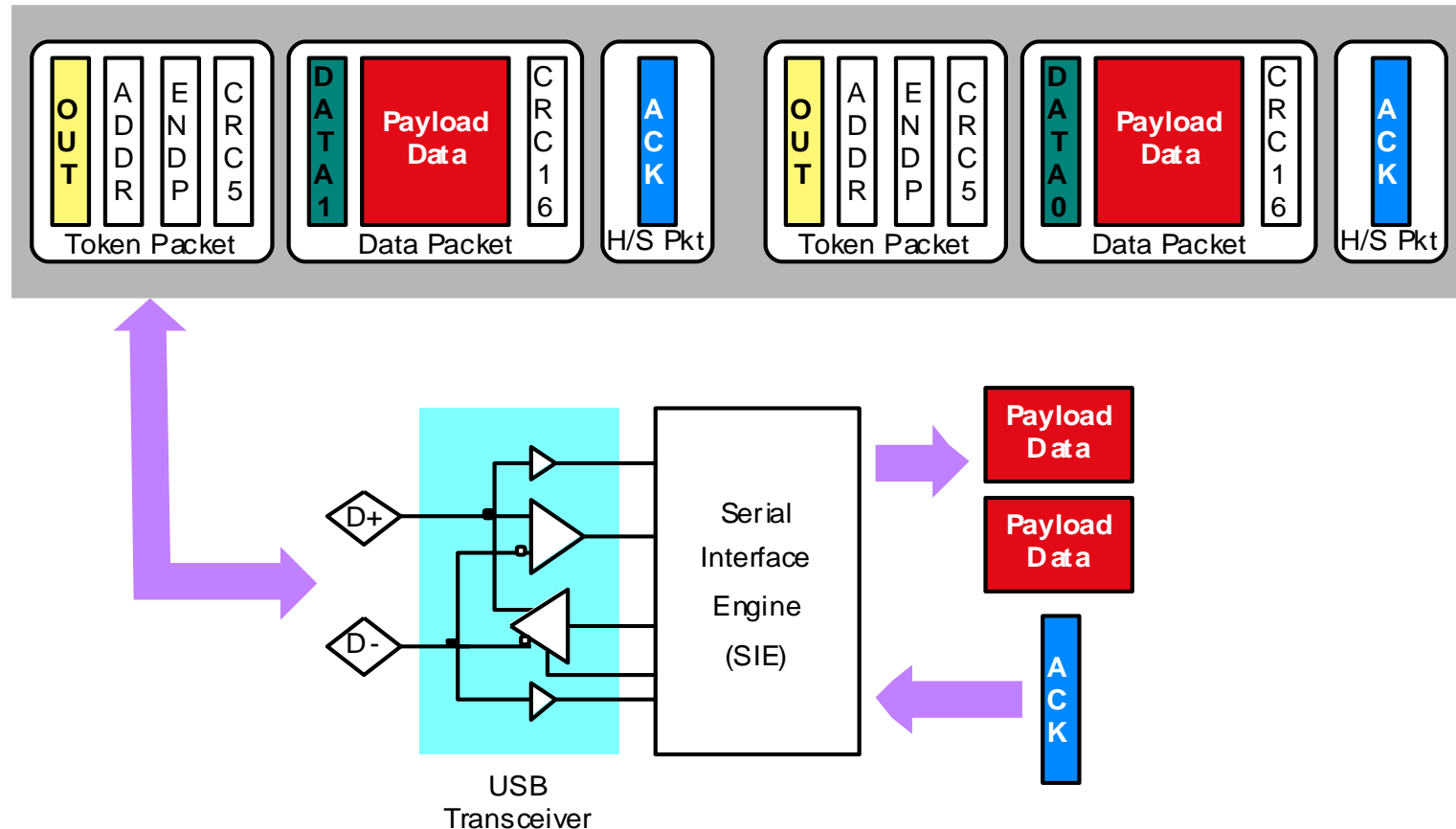
USB Data Packet



Endpoint



The USB Serial Interface Engine (SIE) handles USB packets



Communication support

- Stream with no USB defined protocol
- Default control for provide access
- Message for control function for the device

Serial Interface Engine (SIE)

- Special hardware unit that handles sending and receiving high speed serial USB packets
- Serial data rates needed are too high to handle using only software running on the microcontroller
- Microcontroller only transfers bytes to/from endpoints in the SIE

USB Software

- OS typically has a USB driver that handles enumeration and loads driver for device
- Some common USB device drivers may be provided with OS (storage, serial, keyboard, mouse)
- Most other devices need a device driver from the manufacturer

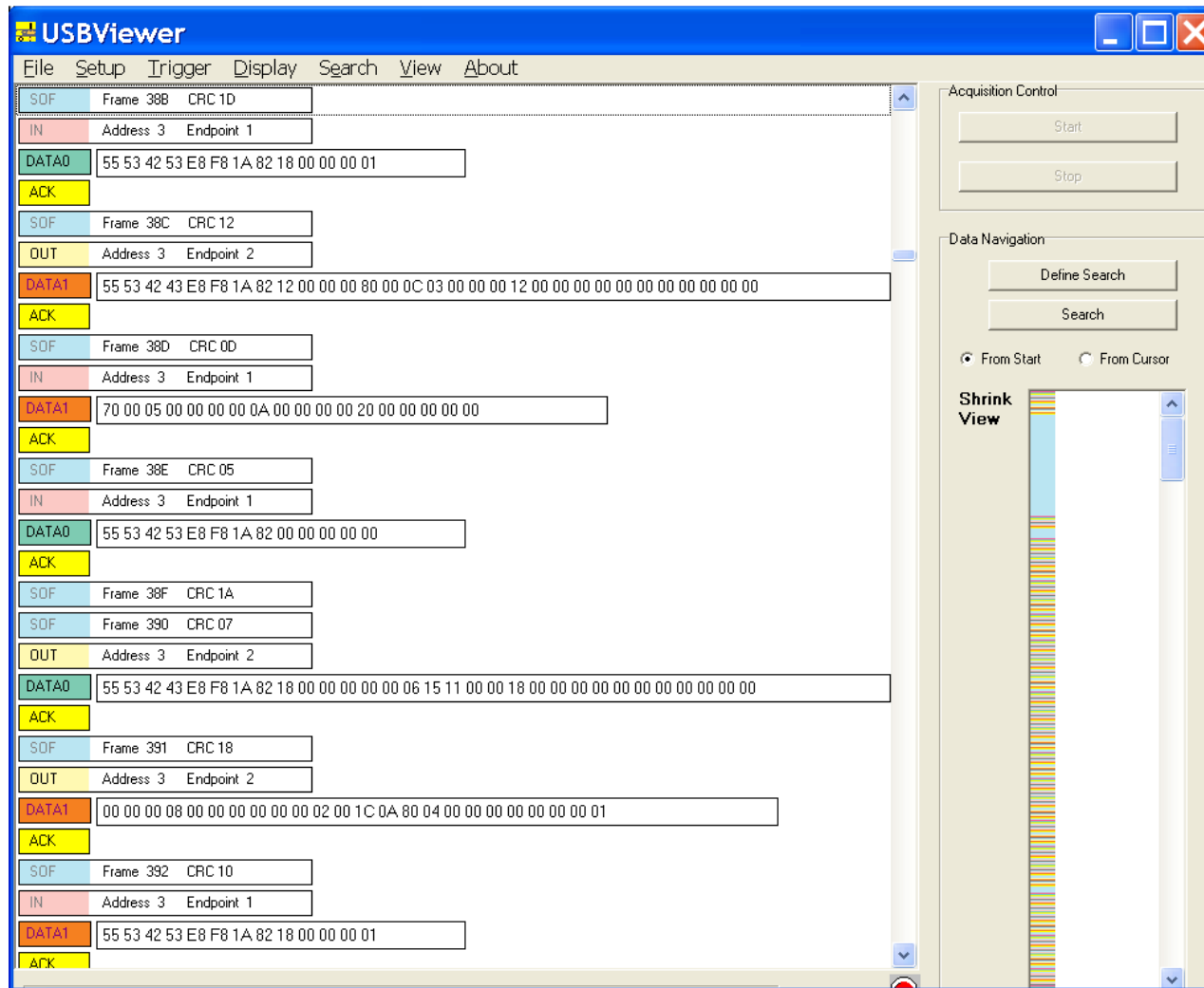


Figure 3.9 A USB protocol analyzer captures and displays the USB packets exchanged with a USB Flash Drive (www.usbdeveloper.com) .

Wireless USB

- Wireless extension of USB 2.0 and it operates at ultra wide band 3.1 GHz to 10.6 GHz frequencies
- For short range personal area network (high speed 480 Mbps 3 meter or 110 Mbps 10 meter channel)
- FCC has recommend a host wire adapter and a device wire adapter
- Wireless USB also supports dual-role devices

Python Interface

- It treats USB like UART communication:

```
#!/usr/bin/python
import serial
ser = serial.Serial('/dev/ttyUSB0', 4800, timeout = 1)
x = ser.read(1200)
print x
```


Questions?

- What's the maximum speed and distance of USB v2?
- Why USB can send the data in higher speed and better distance than I²C?
- What're other advantages/disadvantages comparing with I²C?

Firewire

- It was initiated by Apple Inc. and developed by IEEE 1394 working group
- IEEE 1394a and IEEE 1394b standard
- IEEE 1394a is up to 400 Mbps
- IEEE 1394b is up to 800 Mbps
- Serial isochronous transfer or asynchronous transfer
- Transfer data at a guarantee rate
- Also used in real-time device and video conference

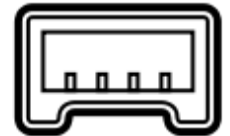
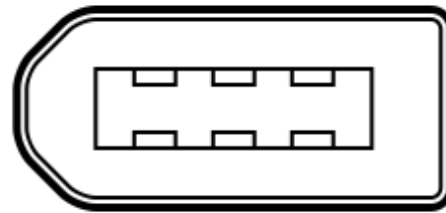
Firewire application

- Multimedia streaming devices
- Digital Video cameras
- Digital camcorders
- Digital Video Disks (DVDs)
- Music system multimedia peripherals
- Some harddisk drives
- Some high-speed printers

isochronous or asynchronous

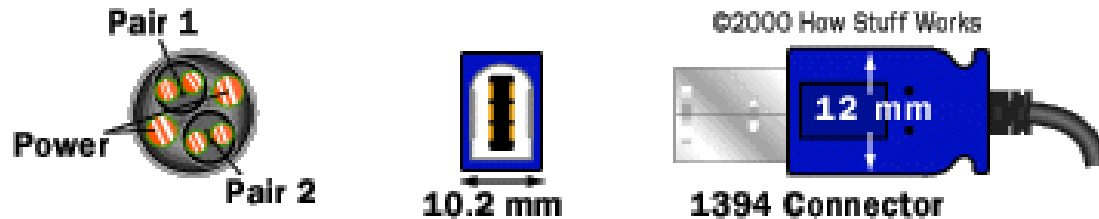
- Isochronous
 - time-critical
 - error-tolerant data
- asynchronous
 - data isn't error-tolerant
 - not time-critical

Firewire



Firewire interface

- Six pin configuration (Firewire400):
power, ground, two twisted pair sets
- Nine pin configuration (Firewire800):
two additional ground shield

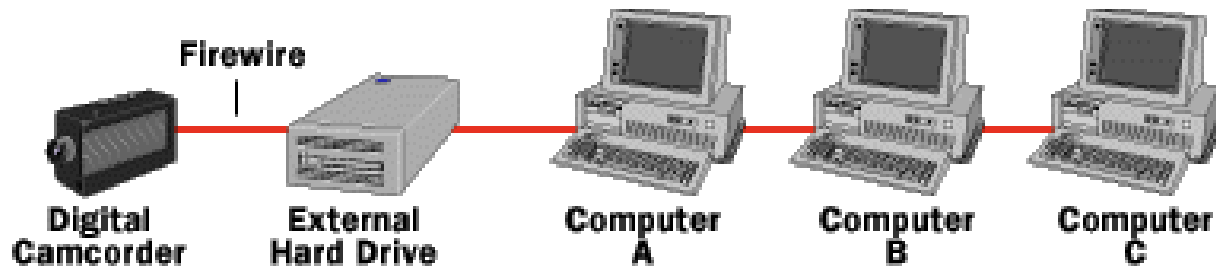


Firewire features

- Can interface up to 63 external devices
- Support plug and play and hot plugging
- Provides self-powered and bus powered support
- Twisted pairs
- Support multiple-host per bus

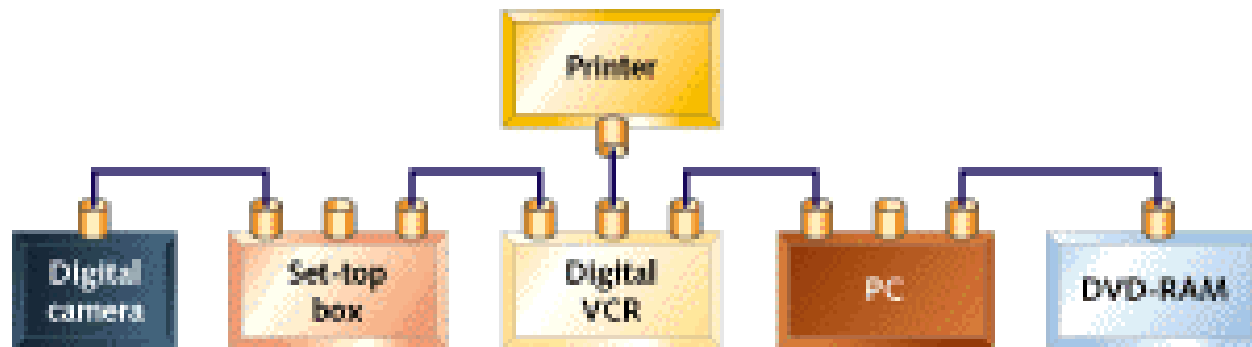
Daisy Chain configuration

- Can be up to 16 hop
- For Firewire800, up to 400 meter long

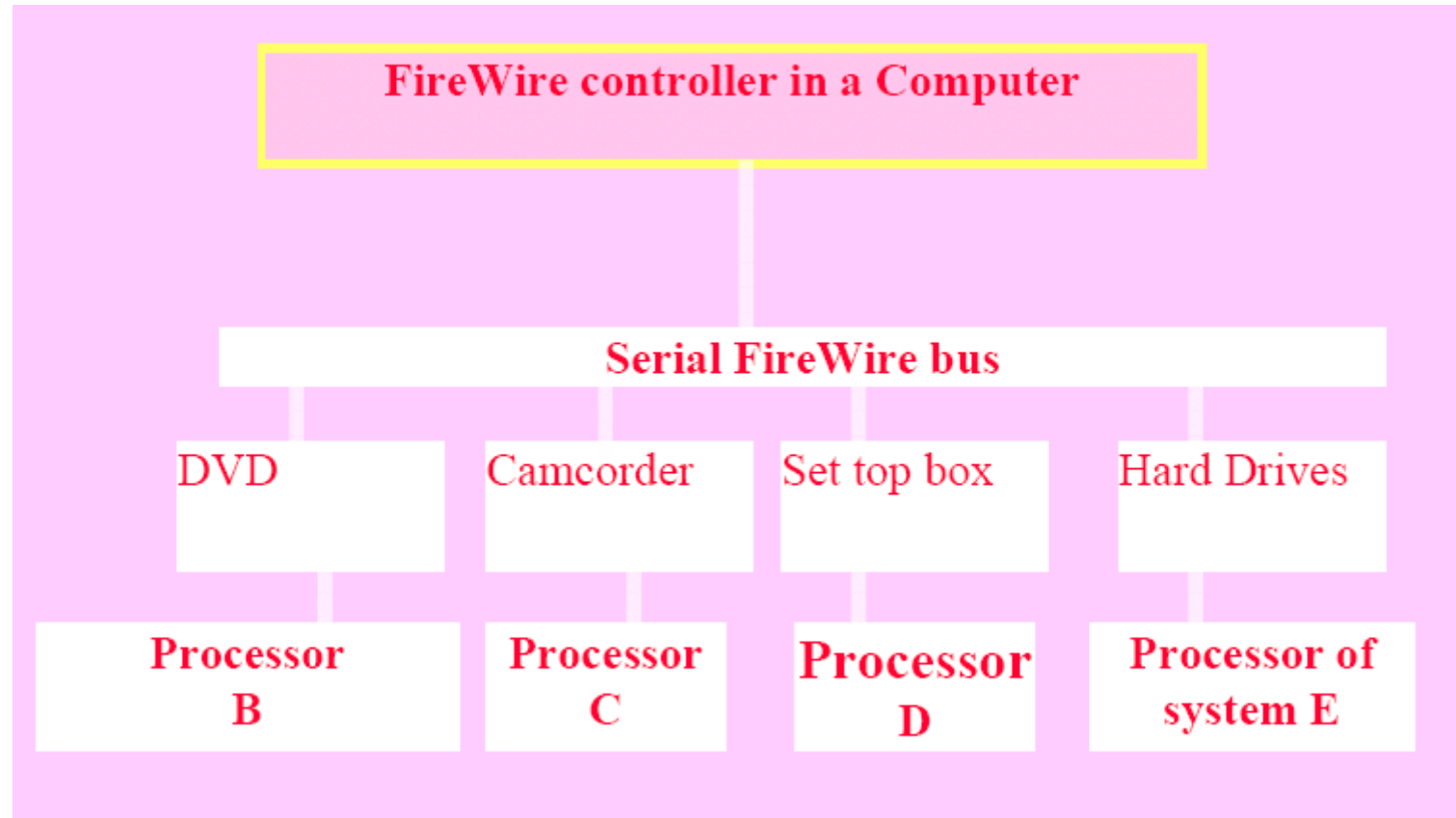


©2000 How Stuff Works

Another Firewire connection



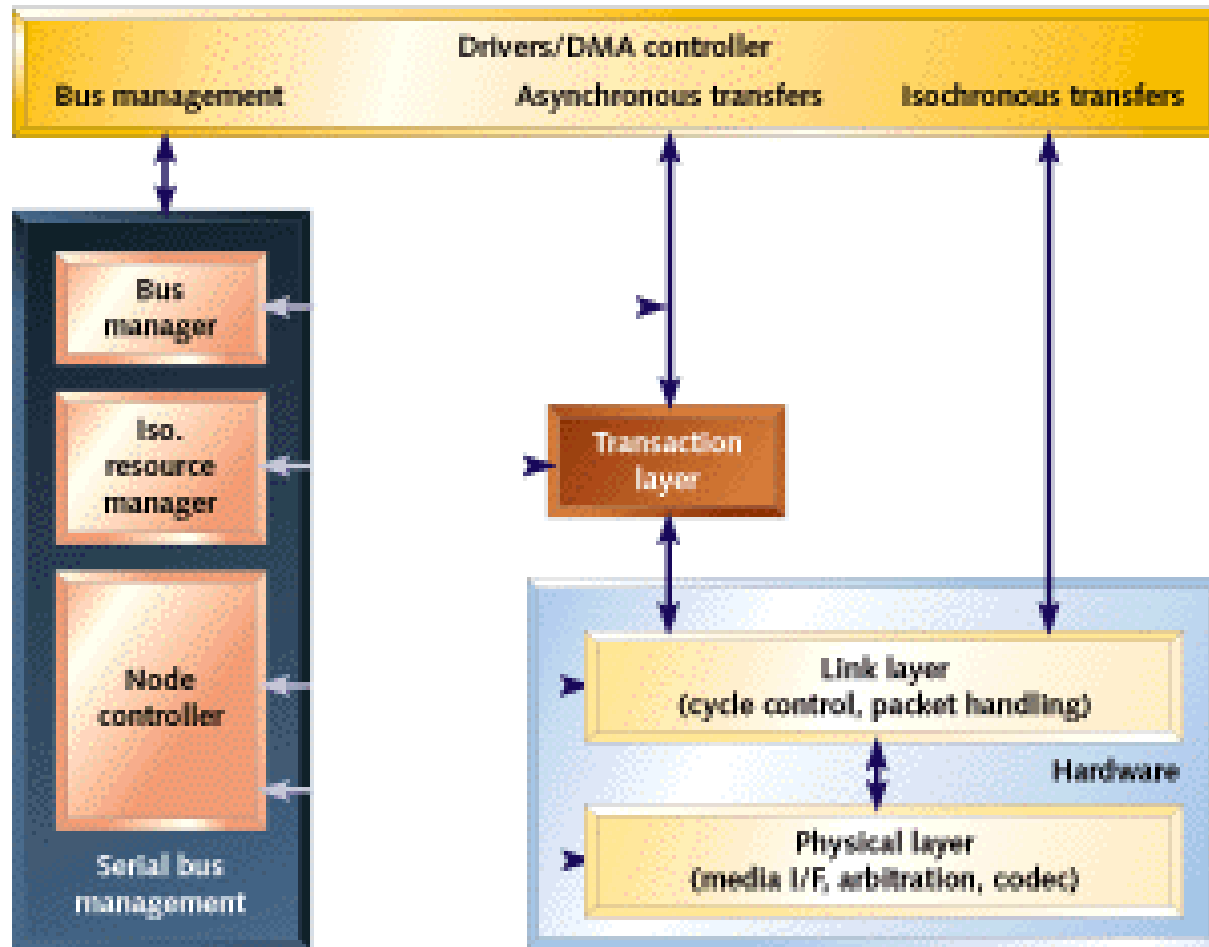
Serial Firewire Bus



Data link layer

Signal Name	Source	Description
LReq	Link request	Link request—used to initiate a request to send a packet, as well as a request to read directly from a PHY register.
SCLk	Physical layer	49.152MHz clock used to synchronize data readout. (The frequency may change depending on data rates with 1394b.)
Data[0:7]	Either	Data—higher transfer speeds use an increasing number of bits: 100Mbps —D[0:1] 200Mbps —D[0:3] 400Mbps —D[0:7] Note that the width of this data bus may expand to 16 bits with 1394b.
Ctl[0:1]	Either	Control interface—defines what state the interface is in.
LPS	Link Layer	Link power status—indicates that the link layer controller is powered.
Link On	Physical layer	Indicates that the physical layer has been powered on.
Direct	Neither	Indicates that no isolation barrier exists.
Backplane	Physical layer	High if physical layer is a backplane implementation.
Clk25	Neither	Indicates that SCLk is only 24.576MHz; valid in a backplane implementation only.

Firewire protocol layers



Comparison with USB

- Although USB 2.0 can run at higher rate than Firewire400, in practice, USB 2.0 rarely exceeds 280 Mbps
- This is because for USB, host controller has to manage low-level USB protocol whereas Firewire delegates the task to the interface hardware
- Simpler bus network

USB vs. Firewire

Feature	USB		FireWire	
	1.1	2.0	400	800
Data transfer rate	12 Mbps	480 Mbps	400 Mbps	800 Mbps
Number of devices	127	127	63	63
Plug and play	Yes	Yes	Yes	Yes
Hot-pluggable	Yes	Yes	Yes	Yes
Isochronous devices	Yes	Yes	Yes	Yes
Bus power	Yes	Yes	Yes	Yes
Bus termination required	No	No	No	No
Bus type	Serial	Serial	Serial	Serial
Cable type	Twisted pair (4 wires: 2 power, 1 twisted-pair set)	Twisted pair (4 wires: 2 power, 1 twisted-pair set)	Twisted pair (6 wires: 2 power, 2 twisted-pair sets)	Twisted pair (8 wires: 2 power, 2 twisted-pair sets, 2 ground)
Networkable	Yes - host-based	Yes - host-based	Yes - peer-to-peer	Yes - peer-to-peer
Network topology	Hub	Hub	Daisy chain	Daisy chain

Questions?

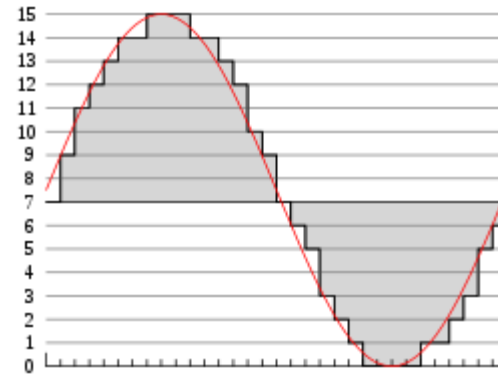
- What's the maximum speed and distance of Firewire?
- Why Firewire is faster?
- What is the similarity and difference comparing with USB?
- What do you think about USBv3 and Firewirev2?

Audio standard

- PCM
- AC97
- MP3
- WMA

Pulse Code Modulation (PCM)

- PCM is an audio format
- PCM is a digital representation of analog signal where the magnitude of the signal is sampled regularly by uniform interval



Audio

- For PCs, AC 97 is a common audio standard
- Needs analog signals so A/D and D/A hardware is used inside the Audio IC
- Analog mixers can add analog signals from different sources
- Audio driver typically provided with OS

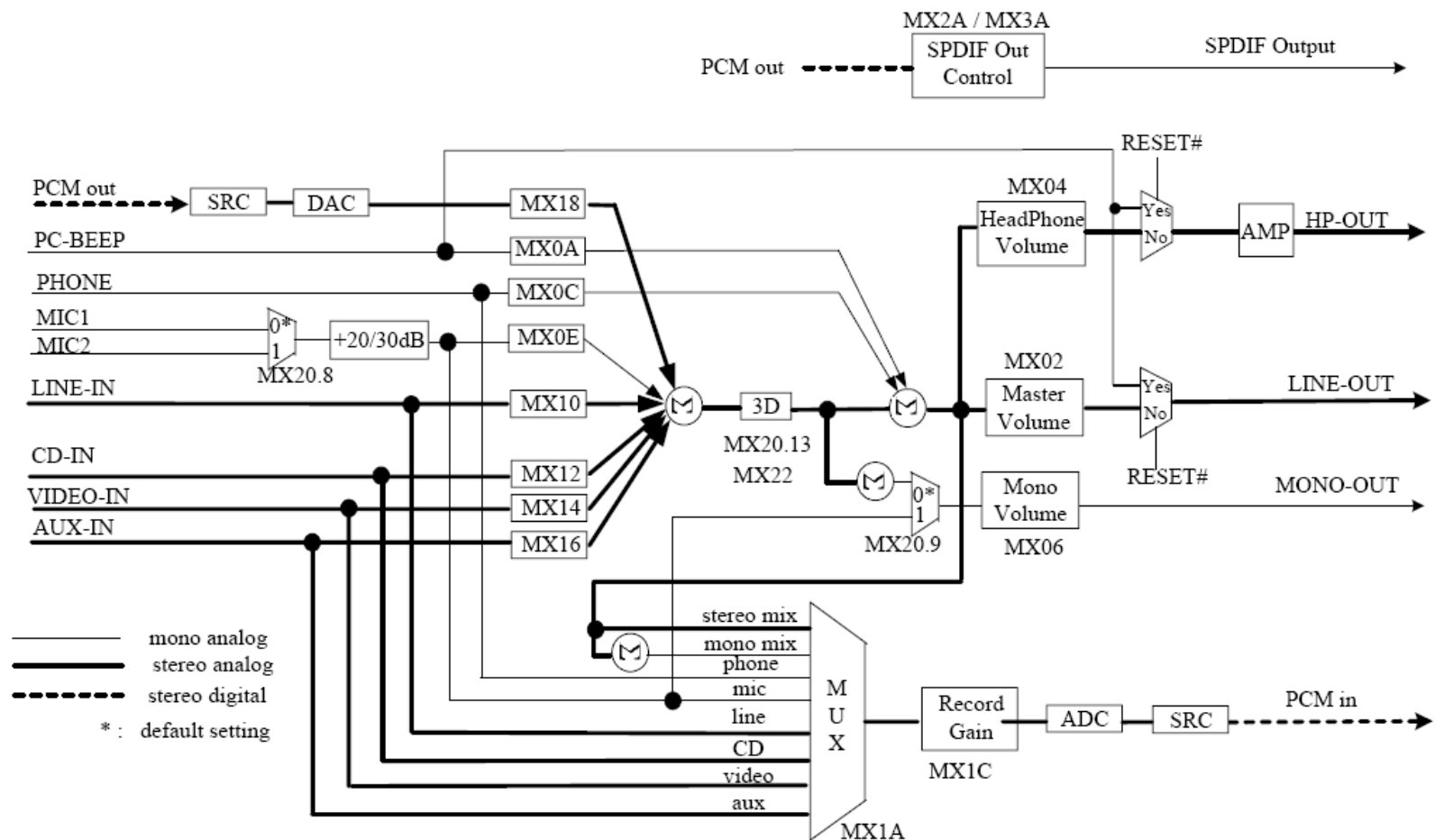
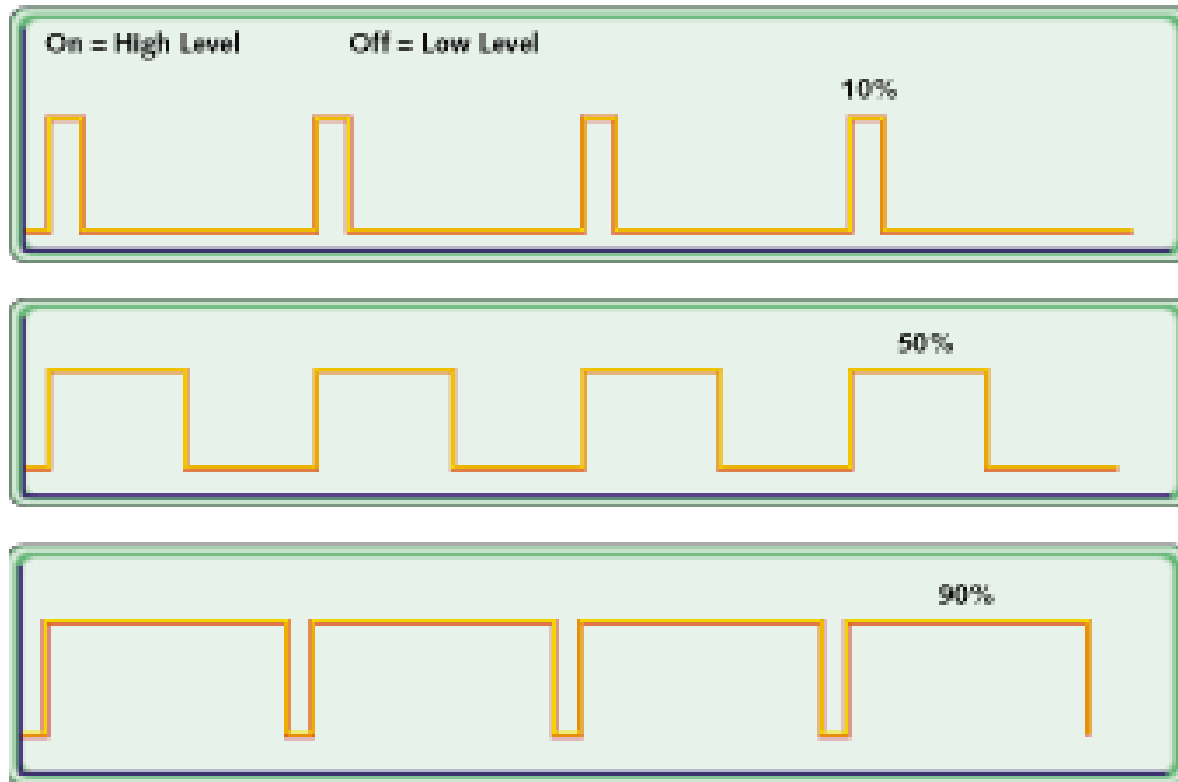


Figure 3.9 Realtek ALC202 AC97 Device Block Diagram

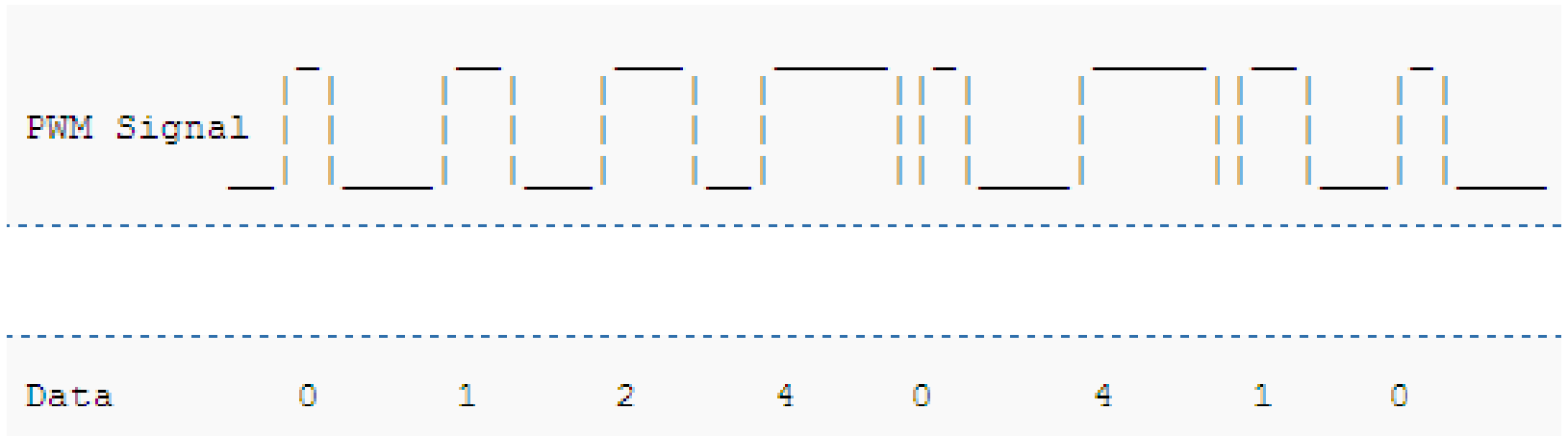
Pulse Width Modulation (PWM)

- PWM is a way for controlling analog circuits with digital output
- By controlling analog circuit digitally, system cost and power consumption can be reduced
- Used in audio, motor control, telecommunication
- The duty cycle is the proportional of the on time to the regular interval or period

PWM with 10%, 50%, and 90% duty cycle



PWM Signal



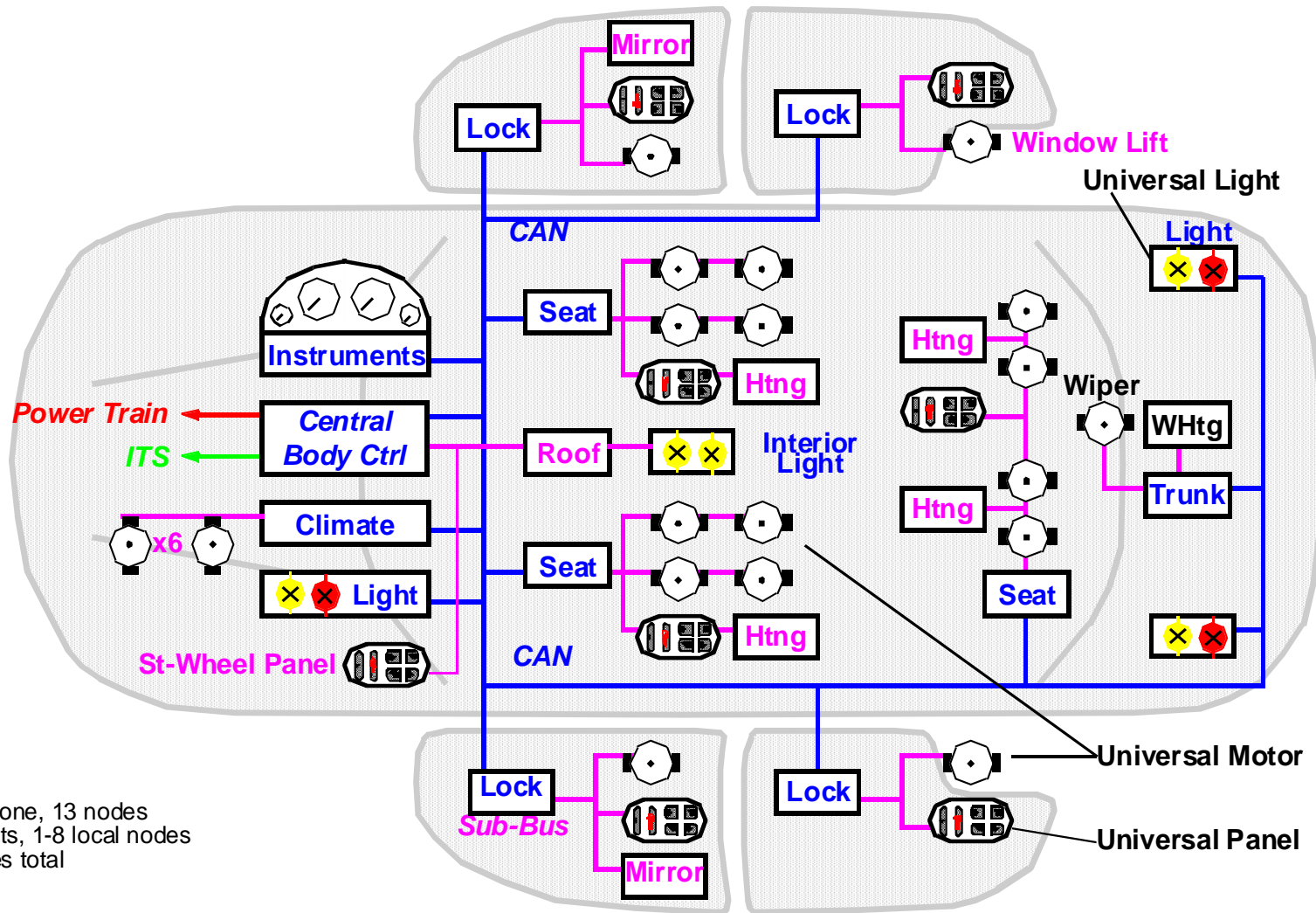
CAN and LIN Bus

- Serial Buses Developed for Automotive Industry in mid 1980s – More advanced than RS232 Serial
- CAN is used to connect car subsystem processors together, has multiple bus masters, & uses 2 wires
- LIN is used to connect a processor with it's smart sensors, a single master, & uses only 1 wire
- LIN used at lower levels along with CAN at higher levels in the system
- Both are now found in other embedded devices

CAN bus

- Two wires, half duplex, high-speed network system
- Can link up to 2032 devices
- Up to 1 Mbps thus facilitates real-time control
- Sophisticated error detection features
- CAN 2.0 A uses 11 bit identifiers
- CAN 2.0 B uses 29 bit identifiers
- ECU = Engine Control Unit

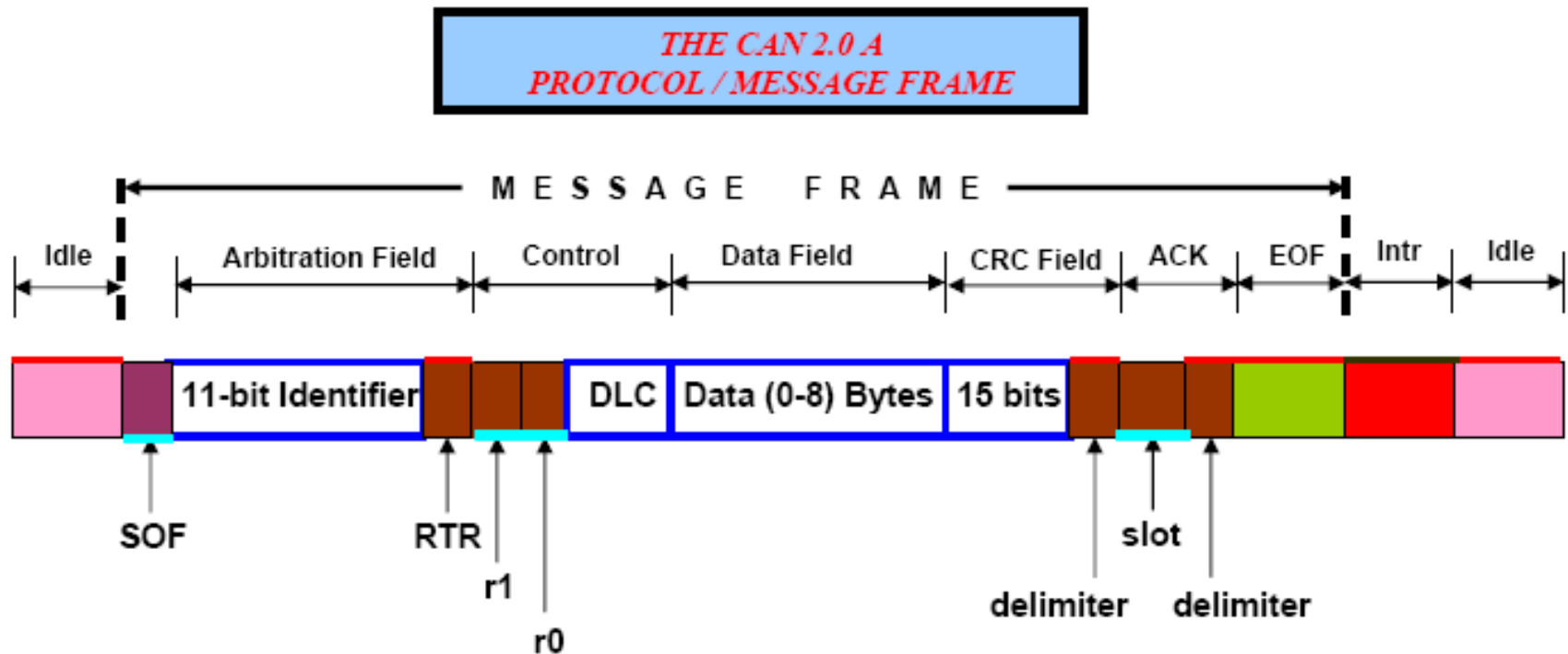
Car Body Network with CAN & LIN



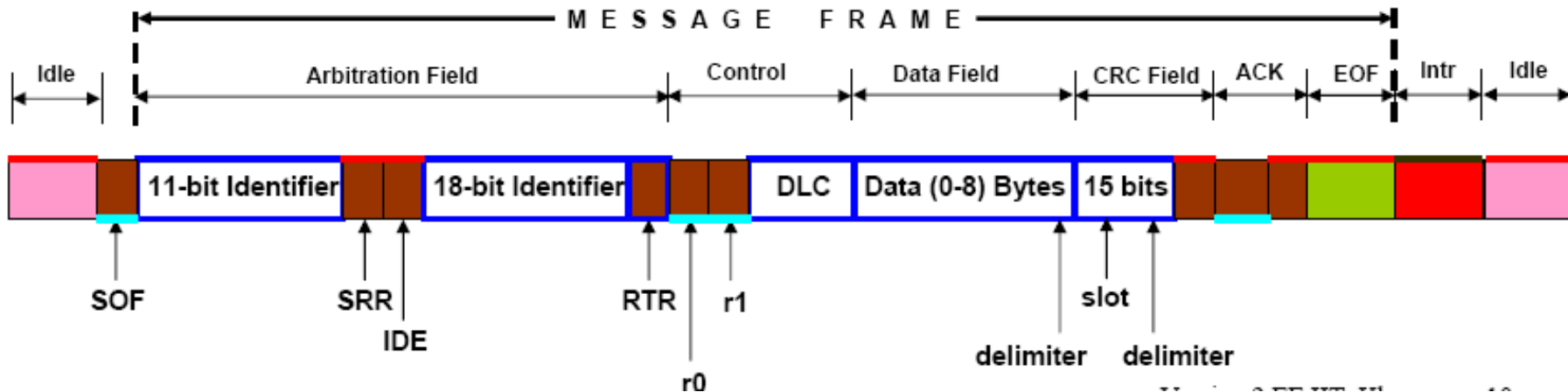
Controller Area Network (CAN)

- Messages contain a message number and not an address
- Lower number messages have priority
- Each devices checks the message number to see if it needs to handle the message
- Data rates up to 1M Baud
- Special CAN chips are available

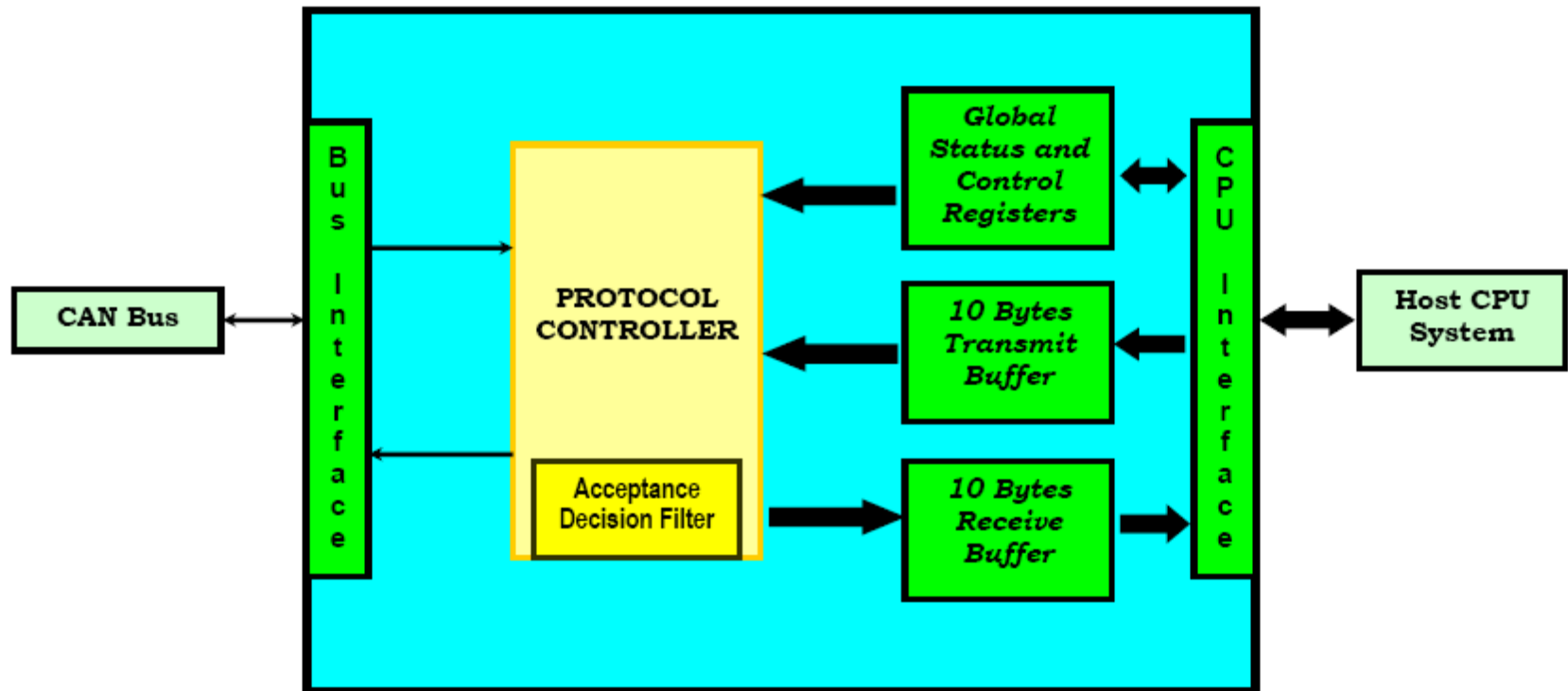
CAN 2.0 A message format



CAN 2.0 B message format



CAN controller topology



Advantages of CAN bus

- Protocol is highly reliable and error resistant
- World-wide accept standard
- Multi-master topology
- Sophisticated error detection
- Short latency time for high priority messages (real-time systems)

Local Interconnect Network (LIN)

- Can be implemented with a UART and microcontroller firmware
- Data rates up to 20K Baud
- Only one bus master
- Lower cost, lower data rates, and 1 less wire than CAN

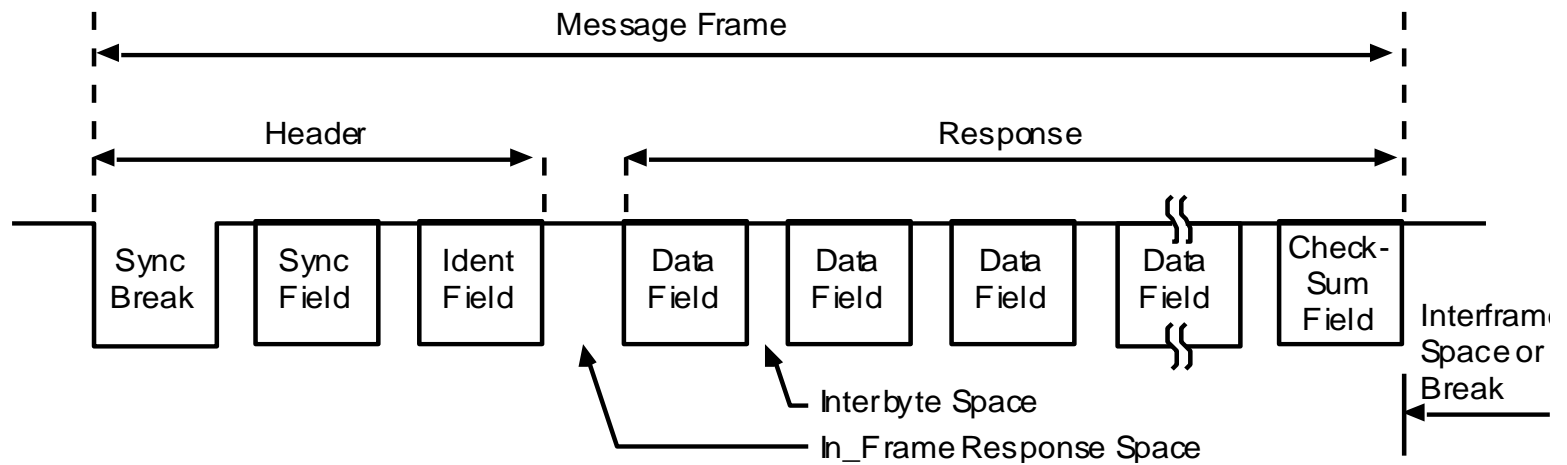


Figure 3.16 A Typical LIN Frame Consisting of Sync Break, Sync Field, Identifier, Data Field and Checksum.

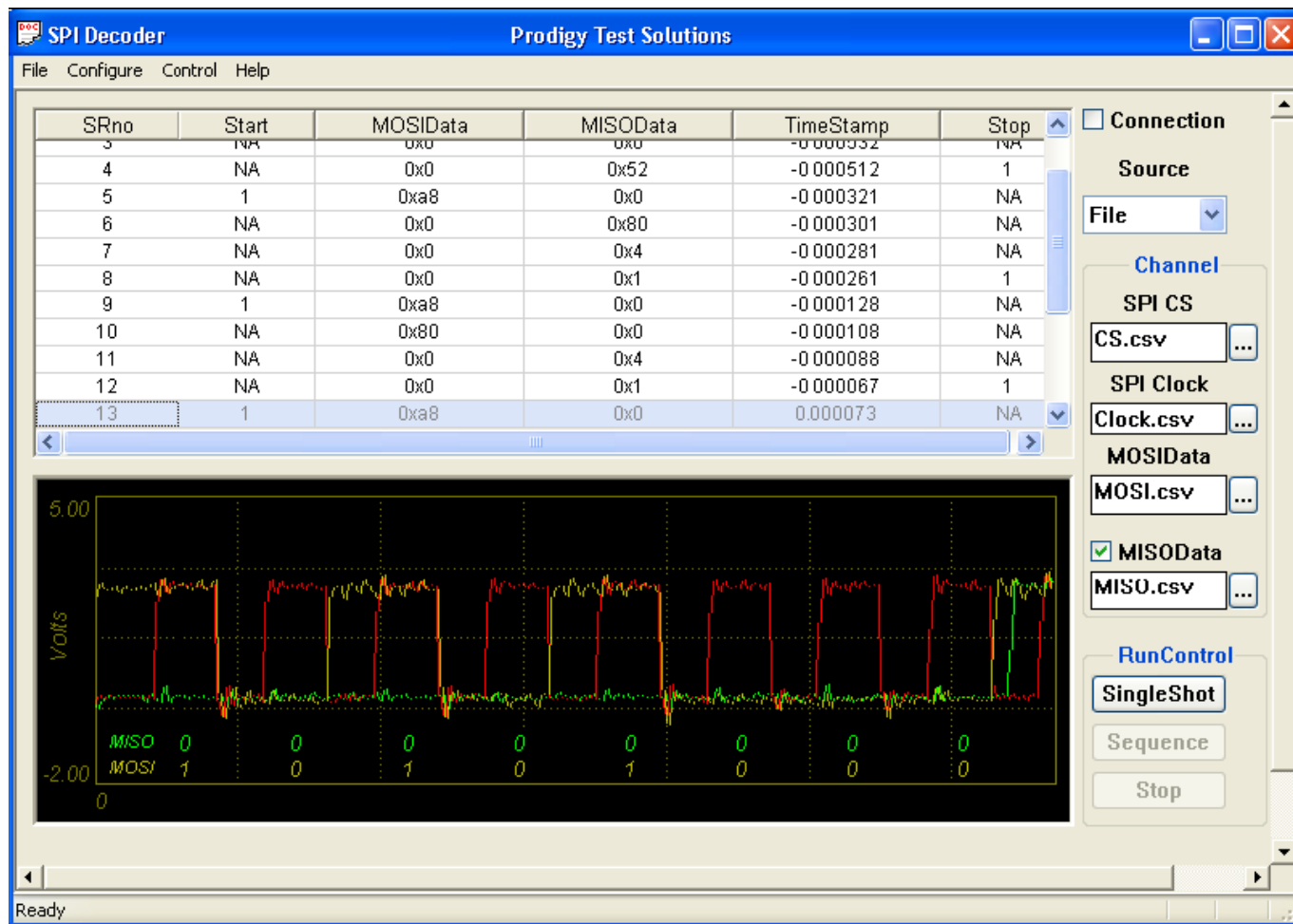


Figure 3.13 The Tektronix DPO7000 1-4 Channel Oscilloscope is an embedded device that runs Windows XP. The display above is from an optional software package for the oscilloscope that decodes and displays SPI bus signals. Images courtesy of Prodigy Test Solutions.

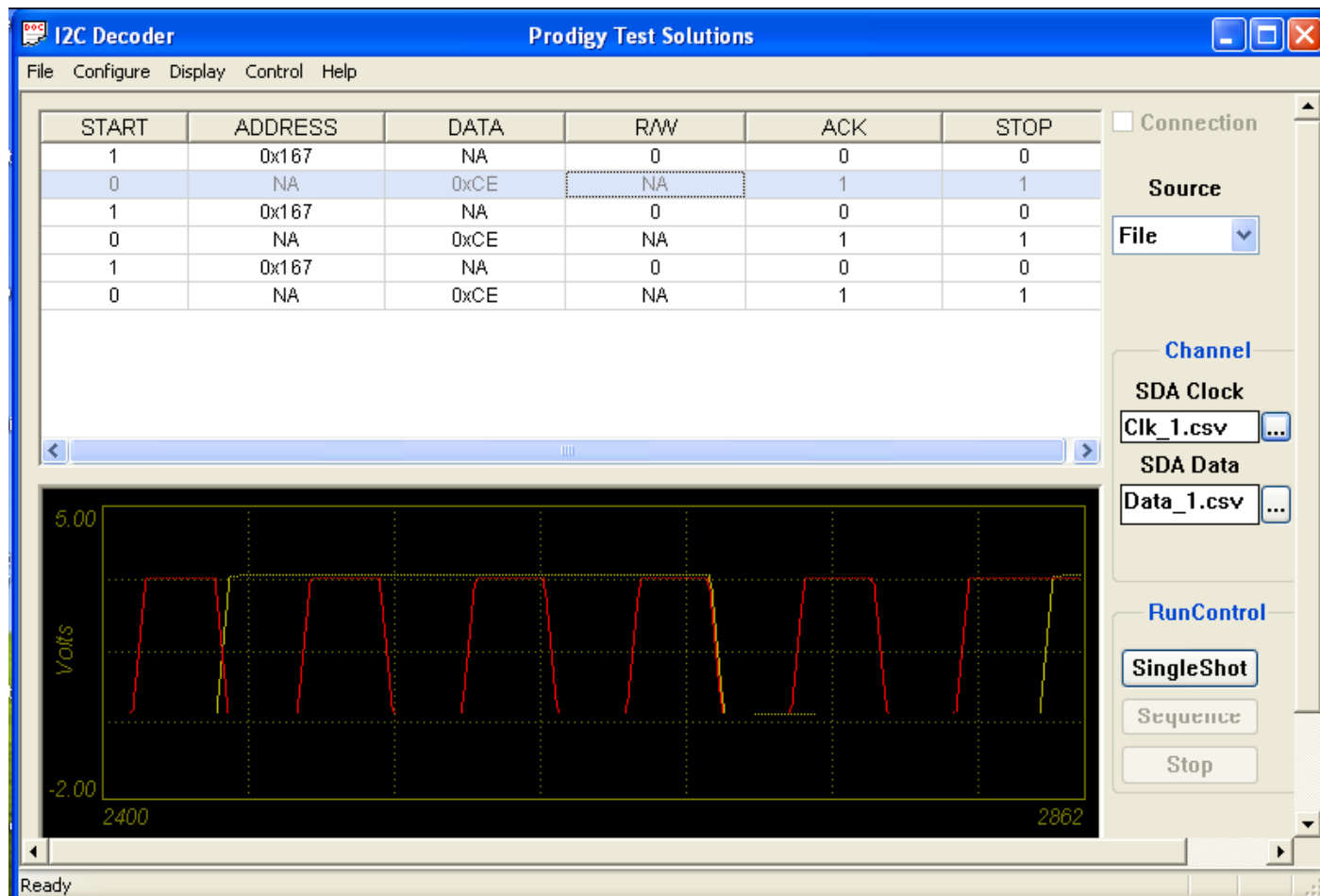


Figure 3.14 The Tektronix DPO7000 1-4 Channel Oscilloscope is an embedded device that runs Windows XP. The display above is from an optional software package for the oscilloscope that decodes and displays I²C bus signals. Images courtesy of Prodigy Test Solutions.

Questions?