
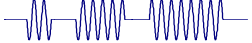
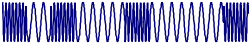
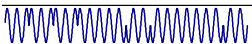


Communication methods

- Communication methods
 - Media and signalling conventions used to transmit data between digital devices
 - Different physical layers methods including:
 - wires, radio frequency (RF), optical (IR)
 - Different encoding schemes including:
 - amplitude, frequency, and pulse-width modulation

Modulation Technique	Waveform
No encoding (Baseband)	
On-Off Keying (OOK)	
Frequency Shift Keying (FSK)	
Binary Phase Shift Keying (BPSK)	

Communication methods

- Dimensions to consider
 - bandwidth – number of wires – serial/parallel
 - speed – bits/bytes/words per second
 - timing methodology – synchronous or asynchronous
 - number of destinations/sources
 - arbitration scheme – daisy-chain, centralized, distributed
 - protocols – provide some guarantees as to correct communication

Bandwidth

- Serial
 - Single wire or channel to transmit information one bit at a time
 - Requires synchronization between sender and receiver
 - Sometimes includes extra wires for clock and/or handshaking
 - Good for inexpensive connections (e.g., terminals)
 - Good for long-distance connections (e.g., LANs)
 - Examples: RS-232, Ethernet, I2C, IrDA, USB, Firewire, Bluetooth
- Parallel
 - Multiple wires to transmit information one byte or word at a time
 - Good for high-bandwidth requirements (CPU to disk)
 - More expensive wiring/connectors/current requirements
 - Examples: SCSI-2, PCI bus (PC), PCMCIA (Compact Flash)
- Issues
 - Encoding, data transfer rates, cost of connectors and wires, modularity, error detection and/or correction

Speed

- Serial
 - low-speed, cheap connections
 - RS-232 1K-20K bits/sec, copper wire
 - medium-speed efficient connections
 - I2C 10K-400K bits/sec, board traces
 - IrDA 9.6K-4M bits/sec, line-of-sight, 0.5-6.0m
 - high-speed, expensive connections
 - USB 1.5M bytes/sec, USB2 60M bytes/sec
 - Ethernet 1.5M-1G bits/sec, twisted-pair or co-axial
 - Firewire 12.5-50M bytes/sec
- Parallel
 - low-speed, not too wide
 - SCSI-2 10M bytes/sec, 8 bits wide
 - PCI bus, 250M bytes/sec, 32 bits wide
 - PCMCIA (CF+), 9-10M bytes/sec, 16 bits wide
 - high-speed, very wide – memory systems in large multi-processors
 - 200M-2G bytes/sec, 128-256 bits wide

Speed

- Issues
 - length of the wires (attenuation, noise, capacitance)
 - connectors (conductors and/or transducers)
 - environment (RF/IR interference, noise)
 - current switching (spikes on supply voltages)
 - number and types of wires (cost of connectors, cross-talk)
 - flow-control (if communicating device can't keep up)

Timing methodology

- Asynchronous
 - less wires (no clock)
 - no skew concerns
 - synchronization overhead
 - appropriate for loosely-coupled systems (CPU and peripherals)
 - common in serial schemes
- Synchronous
 - clock wires and skew concerns
 - no synchronization overhead
 - can be high-speed if delays are small and can be controlled
 - appropriate for tightly-couple systems (CPU and memory/disk)
 - common in parallel schemes

Timing methodology

- Issues
 - clock period and wire delay
 - synchronization and skew
 - encoding of timing and data information
 - handshaking
 - flow-control
 - power consumption

Number of devices communicating

- Single source – single destination
 - point-to-point
 - cheap connections, no tri-stating necessary
- Single source – multiple destination
 - fanout limitations
 - addressing scheme to direct data to one destination
- Multiple source – multiple destination
 - arbitration between senders
 - tri-stating capability is necessary
 - collision detection
 - addressing scheme
 - priority scheme
 - fairness considerations

Arbitration schemes

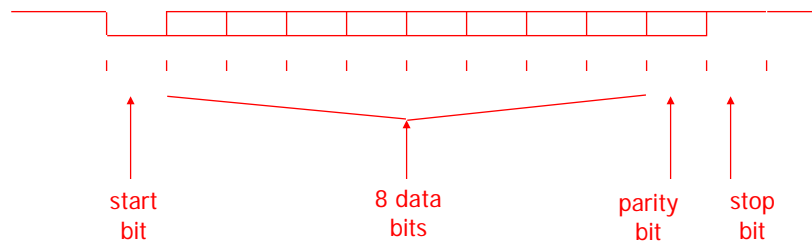
- Daisy-chain or token passing
 - devices either act or pass to next
 - fixed priority order
 - as many wires as devices
 - fairness issues
- Centralized
 - request to central arbiter
 - central arbiter implements priority scheme
 - wires from/to each device can be costly
 - can be dynamically changing priority/fairness
- Distributed
 - no central arbiter
 - common set of wires (or ether) observed by all devices
 - fixed priority/fairness scheme

Serial case studies

- RS-232 (IEEE standard)
 - serial protocol for point-to-point, low-cost, low-speed applications for PCs
- I2C (Philips)
 - up to 400Kbits/sec, serial bus for connecting multiple components
- Ethernet (popularized by Xerox)
 - most popular local area network protocol with distributed arbitration
- IrDA (Infrared Data Association)
 - up to 115kbps wireless serial (Fast IrDA up to 4Mbs)
- Firewire (Apple – now IEEE1394)
 - 12.5-50Mbytes/sec, consumer electronics (video cameras, TVs, audio, etc.)
- SPI (Motorola)
 - 10Mbits/sec, commonly used for microcontroller to peripheral connections
- USB (Intel – followed by USB-2)
 - 12-480Mbits/sec, isochronous transfer, desktop devices
- Bluetooth (Ericsson – cable replacement)
 - 700Kbits/sec, multiple portable devices, special support for audio

RS-232 (standard serial line)

- Point-to-point, full-duplex
- Synchronous or asynchronous
- Flow control
- Variable baud (bit) rates
- Cheap connections (low-quality and few wires)
- Variations: parity bit; 1, 1.5, or 2 stop bits

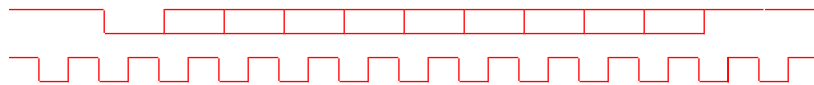


RS-232 wires

- TxD – transmit data
 - TxC – transmit clock
 - RTS – request to send
 - CTS – clear to send
 - RxD – receive data
 - RxC – receive clock
 - DSR – data set ready
 - DTR – data terminal ready
 - Ground
- all wires active low
"0" = -12v, "1" = 12v
- special driver chips that generate $\pm 12v$ from 5v

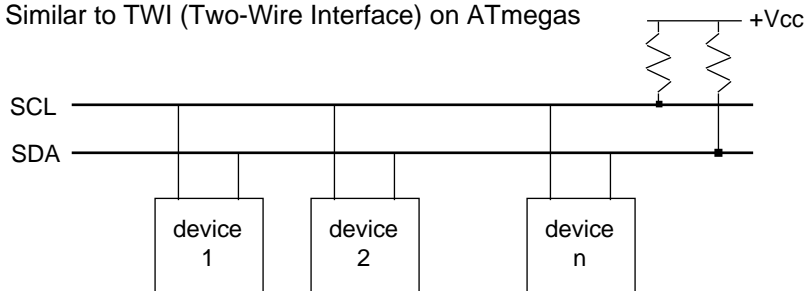
Transfer modes

- Synchronous
 - clock signal wire is used by both receiver and sender to sample data
- Asynchronous
 - no clock signal in common
 - data must be oversampled (16x is typical) to find bit boundaries
- Flow control
 - handshaking signals to control rate of transfer



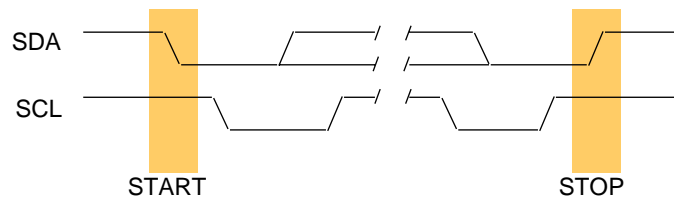
Inter-Integrated Circuit Bus (I2C)

- Modular connections on a printed circuit board
- Multi-point connections (needs addressing)
- Synchronous transfer (but adapts to slowest device)
- Similar to Controller Area Network (CAN) protocol used in automotive applications
- Similar to TWI (Two-Wire Interface) on ATmegs



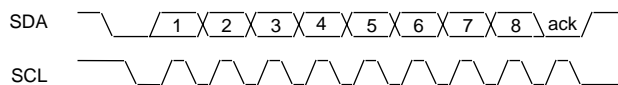
Serial data format

- SDA going low while SCL high signals start of data
- SDA going high while SCL high signals end of data
- SDA can change when SCL low
- SCL high (after start and before end) signals that a data bit can be read



Byte transfer

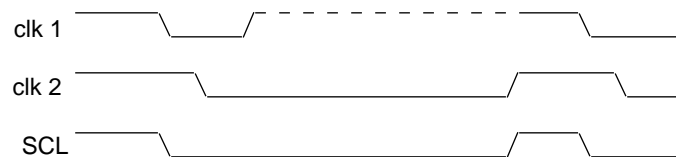
- Byte followed by a 1 bit acknowledge from receiver
- Open-collector wires
 - sender allows SDA to rise
 - receiver pulls low to acknowledge after 8 bits



- Multi-byte transfers
 - first byte contains address of receiver
 - all devices check address to determine if following data is for them
 - second byte usually contains address of sender

Clock synchronization

- Synchronous data transfer with variable speed devices
 - go as fast as the slowest device involved in transfer
- Each device looks at the SCL line as an input as well as driving it
 - if clock stays low even when being driven high then another device needs more time, so wait for it to finish before continuing
 - rising clock edges are synchronized



Arbitration

- Devices can start transmitting at any time
 - wait until lines are both high for some minimum time
 - multiple devices may start together - clocks will be synchronized
- All senders will think they are sending data
 - possibly slowed down by receiver (or another sender)
 - each sender keeps watching SDA - if ever different (driving high, but its really low) then there is another driver
 - sender that detects difference gets off the bus and aborts message
- Device priority given to devices with early 0s in their address
 - 00...111 has higher priority than 01...111

Inter-Integrated Circuit Bus (I2C)

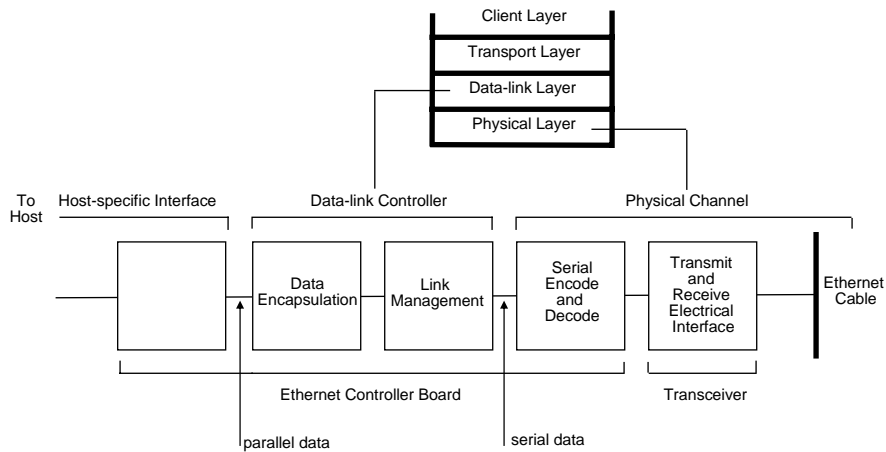
- Supports data transfers from 0 to 400KHz
- Philips (and others) provide many devices
 - microcontrollers with built-in interface
 - A/D and D/A converters
 - parallel I/O ports
 - memory modules
 - LCD drivers
 - real-time clock/calendars
 - DTMF decoders
 - frequency synthesizers
 - video/audio processors

Ethernet (Xerox local area network)

- Local area network
 - up to 1024 stations
 - up to 2.8 km distance
 - 10Mbits/sec serially on shielded co-axial cable
 - 1.5Mbits/sec on twisted pair of copper pair
- Developed by Xerox in late 70s
 - still most common LAN right now
 - being displaced by fiber-optics (can't handle video/audio rates or make required service guarantees)
- High-level protocols to ensure reliable data transmission
- CSMA-CD: carrier sense multiple access with collision detection

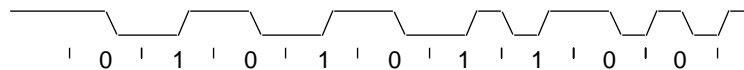
Ethernet layered organization

- Physical and data-link layers are our focus



Serial data format

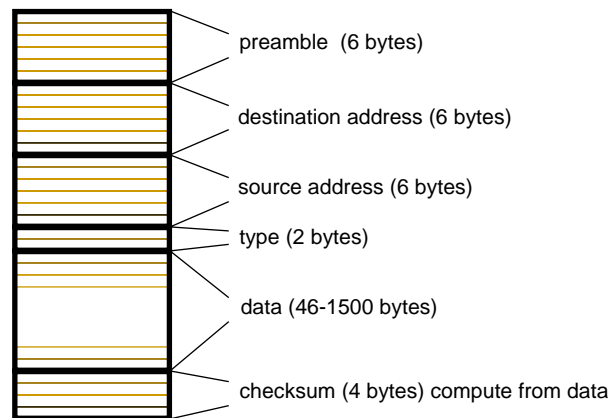
- Manchester encoding
 - signal and clock on one wire (XORed together)
 - "0" = low-going transition
 - "1" = high-going transition



- Extra transitions between 00 and 11 need to be filtered
 - preamble at beginning of data packet contains alternating 1s and 0s
 - allows receivers to get used to where important transitions should be and ignore extra ones (this is how synchronization is achieved)
 - preamble is 48 bits long: 10101...01011

Ethernet packet

- Packets size: 64 to 1518 bytes + 6 bytes of preamble

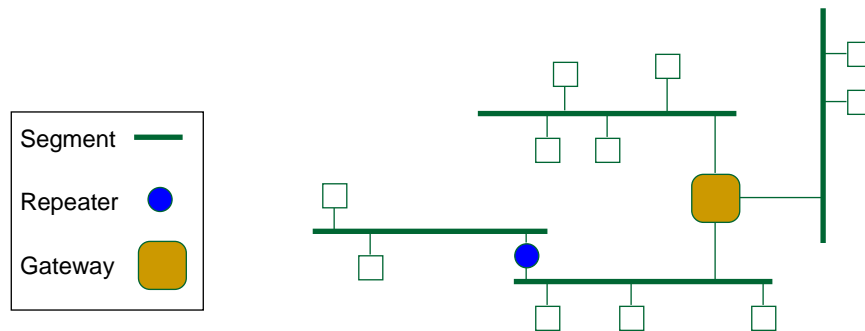


Arbitration

- Wait for line to be quiet for a while then transmit
 - detect collision
 - average value on wire should be exactly between 1 and 0
 - if not, then two transmitters are trying to transmit data
- If collision, stop transmitting
 - wait a random amount of time and try again
 - if collide again, pick a random number from a larger range (2x) and try again
- Exponential backoff on collision detection
- Try up to 16 times before reporting failure

Extending Ethernet

- Segments, repeaters, and gateways
 - segment: a single cable
 - repeater: transfers all messages on one segment to another and vice-versa
 - gateway: selectively forwards messages to other segments and helps isolate traffic

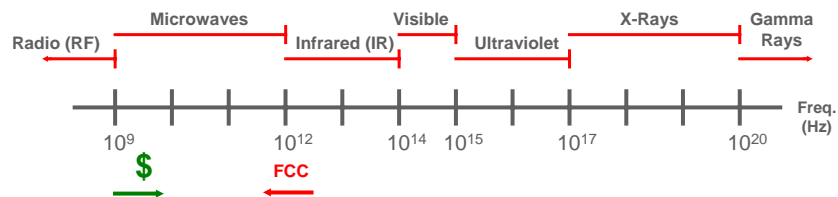


Infrared Data Association

- Consortium of over 160 companies
- Meet needs of the “mobile professional”
 - Short interactions with other devices (file transfer, printing)
 - Possibly using others' peripherals (visiting a customer's office)
- Goals:
 - Suitable replacement for cables
 - Interoperability
 - Minimal cost
 - “Point-and-shoot” model (intended use and to reduce interference)
- History:
 - First standard developed in 1994
 - Revisions as recently as late 1998 (i.e., still active)

IrDA: Infrared Data Association

- Characteristics of IR:
 - Implementation costs rise significantly around 1-10 GHz
 - one important exception is IR at around 500 THz – very inexpensive
 - Signals above 100 GHz cannot penetrate walls
 - Most signals below 300 GHz are regulated by the FCC



Speed

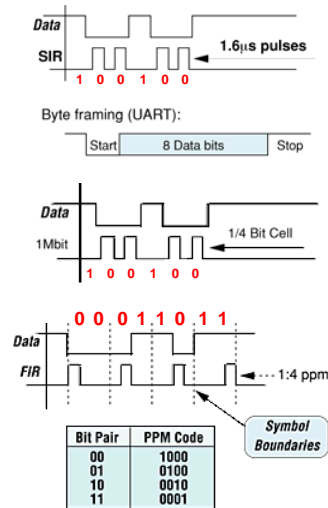
- Components include:
 - Transmitter (LED) and paired receiver (photodiode)
- IrDA supports wide range of speeds
 - 2400 bps to 4 Mbps
 - Exact physical-layer protocol used depends on speed of IrDA connection
 - Uses highest speed available on both devices
 - determined when connection is established
- Future promises even higher speeds:
 - 16-50 Mbps is not too far off
- Comparison to other wireless technologies:
 - Low-power RF (e.g., Bluetooth) slightly slower (.5 - 2 Mbps max)
 - Bound by walls, easy to control, intentional aspect
 - Much lower-power than high-speed RF (e.g., 802.11a at 50Mbps)

Low-speed Modulation

- Speed: 2400 bps - 115 kbps ("Serial Infrared", or SIR)
 - Only 0's require pulse (and thus power) ; pulse < full bit time
 - Standard UART byte framing
 - Pulse is constant 1.6 μ s long (so duty cycle varies with speed)
 - Average duty cycle: $\leq 9\%$

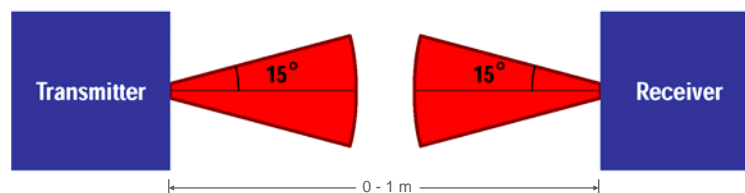
- Speed: 576 kbps - 1 Mbps
 - similar to SIR (pulse only for 0's ; pulse < full bit time)
 - pulse lasts 1/4 of bit time (so pulse varies with speed)
 - Average duty cycle: 12.5%

- Speed: 4 Mbps ("Fast Infrared", or FIR)
 - uses four-pulse-position-modulation scheme (4PPM)
 - pulse during exactly 1/4 of each symbol boundary
 - 4PPM makes synchronization easier to maintain
 - Duty cycle: 25% (independent of data)
 - Lowest power/bit



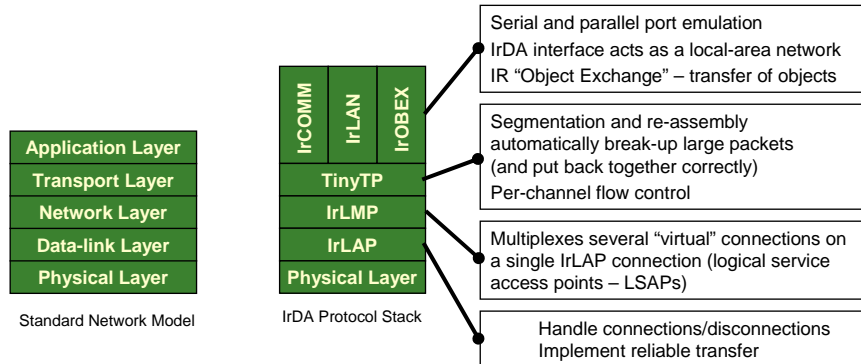
Range

- Linear:
 - IrDA standard requires 0-1 m
 - Realistically, some transceivers work at up to 10 m
- Angular:
 - Limited to a narrow cone (15° half-angle)
 - Done to help reduce interference between devices



IrDA Protocol Stack

- Analogous to the standard layered network model
- Consists of both required and optional components



Protocol Overhead

- Very simple model (point-to-point), so can expect reduced protocol overhead
- For layers in IrDA protocol stack, overhead per packet/frame is:
 - IrLAP = 2 bytes
 - IrLMP = 2 bytes
 - TinyTP = 1 byte

} **Total: 5 bytes**
- For perspective, compare to TCP/IP over Ethernet:
 - Ethernet = 18 bytes minimum
 - IP = 20 bytes
 - TCP = 20 bytes

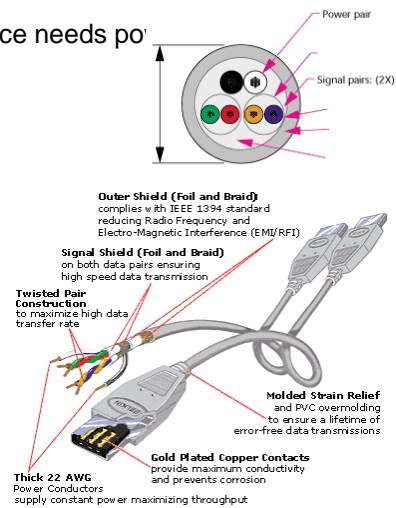
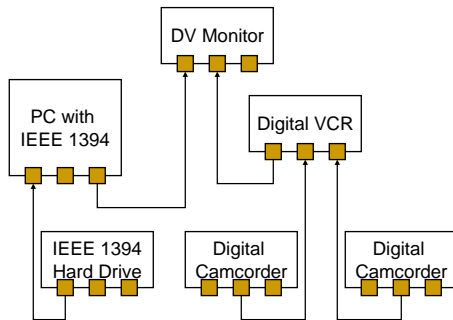
} **Total: 58 bytes (minimum)**
- IrDA takes advantage of its simpler model, and keeps protocol overhead very low.

Firewire

- Interconnection for high-bandwidth consumer electronic devices
 - e.g., still and video cameras, MP3 players, digital video recorders
 - IEEE 1394a standard
 - 12.5-400 Mbits/sec (soon to be 800 Mbits/sec with 1394b)
 - Most consumer devices use 100 Mbits/sec
 - Up to 63 devices connected at once on 4.5m cables
 - Up to 16 cables can be daisy-chained to 72m
 - Devices connect for power as well as communication
 - Hot-swappable devices
 - Asynchronous and isochronous data transfers

Firewire Electrical/Mechanical Spec

- 4-6 wires depending on whether device needs power
- Tree arrangement
 - each branch is bandwidth limited



Firewire data format

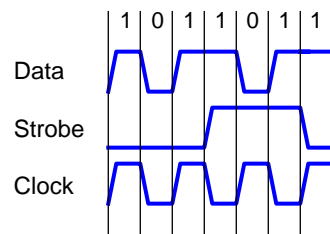
- Data is transferred in addressed packets, and is transaction-based
- Transfers can be asynchronous or isochronous
 - Asynchronous transfers are used mainly for bus configuration, setting up transfers and handshaking, but are also used for bulk data transfer to and from hard disk drives, etc.
 - Isochronous transfers are used for transporting timesensitive data like digital video and audio
- Data packets have a 64-bit address header
 - 10-bit network address
 - 6-bit node address
 - 48 bits for data memory addresses at the receiving node
- Ability to address 1023 networks of 63 nodes, each with up to 281TB (terabytes) of data addresses

Firewire data format (cont'd)

- Bus manager
 - One device on the bus (usually a PC)
- Isochronous resource manager
 - Allocates bus bandwidth for isochronous data transfers based on time-domain multiplexing (TDM) that guarantees a proportion of the total time slots to each device
 - Bandwidth allocation unit is 20.3ns, 6144 of them in a basic cycle of 125us
 - 25us of every cycle is always reserved for asynchronous control data transfers, so a maximum of 4195 units is available for isochronous transfers
 - Typically a stream from a DV camcorder to a PC or digital VCR might need to be allocated a channel of ~1800 bandwidth units, for about 30Mb/s
 - Asynchronous transfers can have multiple data packets per basic cycle, within the 25us reserved for this type of signalling

Firewire signalling

- Data-strobe signalling
 - Avoids two signals where both change at the same time
 - Keeps noise levels low
- Strobe easily derived at transmitter
 - Strobe = Clock xor Data
- Clock is easily recovered at receiver
 - Clock = Data xor Strobe

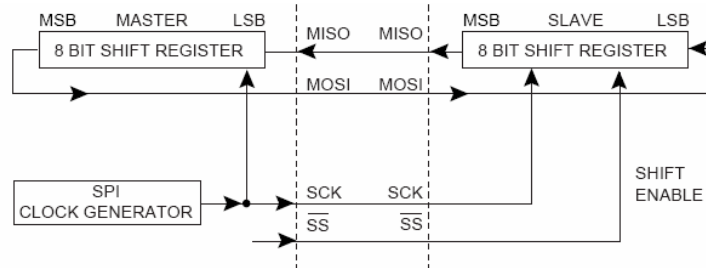


Serial Peripheral Interface

- Common serial interface on many microcontrollers
- Simple 8-bit exchange between two devices
 - Master initiates transfer and generates clock signal
 - Slave device selected by master
- One-byte at a time transfer
 - Data protocols are defined by application
 - Must be in agreement across devices

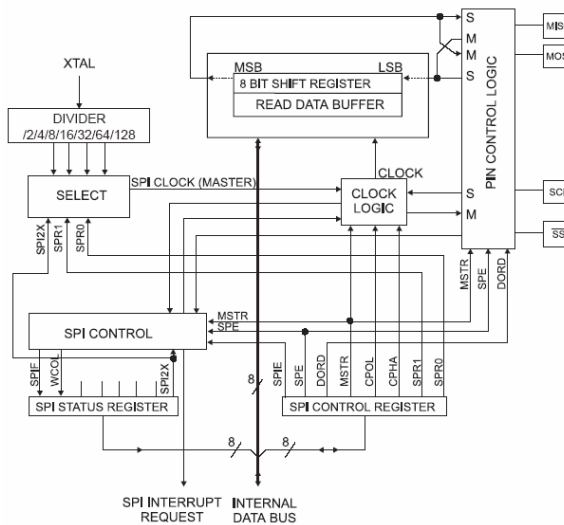
SPI Block Diagram

- 8-bits transferred in each direction every time
- Master generates clock
- Shift enable used to select one of many slaves



SPI on the ATmega16

- Prescaler for clock rate
- Interrupt on receive and on send complete
- Automatically generates SS



SPI Registers

SPI Control Register – SPCR

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

SPI Status Register – SPSR

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Using SPI as a Master

```
void SPI_MasterInit(void)
{
    /* Set MOSI and SCK output, all others input */
    DDRB = _BV(DD_MOSI) | _BV(DD_SCK);
    /* Enable SPI, Master, set clock rate fck/16 */
    SPCR = _BV(SPE) | _BV(MSTR) | _BV(SPR0);
}

void SPI_MasterTransmit(char cData)
{
    /* Start transmission */
    SPDR = cData;
    /* Wait for transmission complete */
    while(!(SPSR & _BV(SPIF)))
    ;
}
```

Using SPI as a Slave

```
void SPI_SlaveInit(void)
{
    /* Set MISO output, all others input */
    DDRB = _BV(DD_MISO);
    /* Enable SPI */
    SPCR = _BV(SPE);
}

char SPI_SlaveReceive(void)
{
    /* Wait for reception complete */
    while(!(SPSR & _BV(SPIF)))
    ;
    /* Return data register */
    return SPDR;
}
```

Data Payload on SPI

- Data is exchanged between master and slave
 - Master always initiates
 - May need to poll slave (or interrupt-driven)
- Decide on how many bytes of data have to move in each direction
 - Transfer the maximum for both directions
 - One side may get more than it needs
- Decide on format of bytes in packet
 - Starting byte and/or ending byte?
 - Can they be distinguished from data in payload?
 - Length information or fixed size?
- SPI buffer
 - Write into buffer, specify length, master sends it out, gets data
 - New data arrives at slave, slave interrupted, provides data to go to master, reads data from master in buffer

Sample code for FTDI SPI

```
int main(void)
{
    FTDI466API usbDevice;
    char buffer[256];
    unsigned char rxBuffer[256];
    unsigned char txBuffer[256];
    DWORD numBytesToSend;
    DWORD bytesSent;
    DWORD numBytesToRead;
    DWORD bytesReceived;
    // setup USB device for MPSSE mode
    bool setup = usbDevice.open();
    if(!setup)
        return 0;
    cout << "INITIALIZING SPI" << endl;
    // setup for SPI communication
    txBuffer[0] = 0x80; // setup PORT
    txBuffer[1] = 0x08; // make CS high
    txBuffer[2] = 0x0B; // outputs: SK, DO, CS, inputs: DI, GPIOL1-L4
    txBuffer[3] = 0x86; // set clk divisor to Tx at 200kHz
    txBuffer[4] = 0x1D; // speed low byte
    txBuffer[5] = 0x00; // speed high byte
    txBuffer[6] = 0x85; // disconnect TDI/DO output from TDO/DI input for loopback testing
    numBytesToSend = 7;
```

Sample code for FTDI SPI (cont'd)

```
// send the instructions of the USB device
bytesSent = usbDevice.write(txBuffer, numBytesToSend);

if(bytesSent != numBytesToSend)
    cerr << "Not all the bytes were sent when initializing MPSSE" << endl;

// see if there were any error codes when setting up SPI
numBytesToRead = usbDevice.getReceiveQueueSize();

if(numBytesToRead > 0)
{
    bytesReceived = usbDevice.read(rxBuffer, numBytesToRead);

    if(bytesReceived != numBytesToRead)
        cerr << "Problem when trying to retrieve the error bytes" << endl;

    for(unsigned int i = 0; i < bytesReceived; i++)
        cout << "Error Byte: " << rxBuffer[i] << endl;
}
```

Sample code for FTDI SPI (cont'd)

```
// loop to demonstrate the SPI protocol
for(int loop = 0; loop < 10; loop++)
{
    Sleep(1000);

    txBuffer[0] = 0x80; // setup PORT
    txBuffer[1] = 0x00; // make CS low
    txBuffer[2] = 0x0B; // outputs: SK, DO, CS, inputs: DI, GPIOL1-L4
    txBuffer[3] = 0x35; // clock out on negative edge, in on negative edge, MSB
    txBuffer[4] = 0x04; // low byte of length : note a length of zero is 1 byte, 1 is 2 bytes
    txBuffer[5] = 0x00; // high byte of length
    txBuffer[6] = 0x71; // payload
    txBuffer[7] = 0x72;
    txBuffer[8] = 0x73;
    txBuffer[9] = 0x74;
    txBuffer[10] = 0x75;
    txBuffer[11] = 0x80; // setup PORT
    txBuffer[12] = 0x08; // make CS high
    txBuffer[13] = 0x0B; // outputs: SK, DO, CS, inputs: DI, GPIOL1-L4

    numBytesToSend = 14;

    // send bytes
    bytesSent = usbDevice.write(txBuffer, numBytesToSend);
    if(bytesSent != numBytesToSend)
        cerr << "Not all the bytes were sent when initializing MPSSE" << endl;
}
```

Sample code for FTDI SPI (cont'd)

```
Sleep(5); // make sure the usb device has enough time to execute command - 5 ms latency timeout is set

// get number of bytes in the received queue
numBytesToRead = usbDevice.getReceiveQueueSize();
cout << "Received " << numBytesToRead << " Bytes" << endl;
if(numBytesToRead > 0)
{
    // get the received bytes
    bytesReceived = usbDevice.read(rxBuffer, numBytesToRead);

    if(bytesReceived != numBytesToRead)
        cerr << "Problem when trying to retrieve the bytes from the receive queue" <<
endl;
    else
    {
        // print out the bytes received over SPI in hex
        for(unsigned int i=0; i < bytesReceived; i++)
            cout << itoa(rxBuffer[i],buffer,16) << " ";
        cout << endl;
    }
}
}
```


Universal Serial Bus

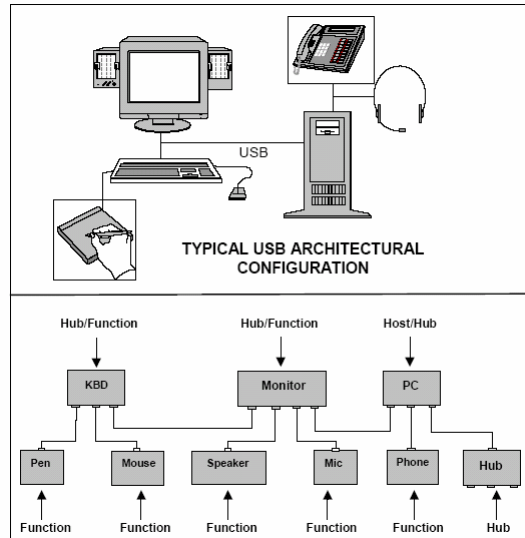
- Connecting peripherals to PCs
 - Ease-of-use
 - Low-cost
 - Up to 127 devices (optionally powered through bus)
 - Transfer rates up to 480 Mb/s
 - Variable speeds and packet sizes
 - Full support for real-time data for voice, audio, and video
 - Protocol flexibility for mixed-mode isochronous data transfers and asynchronous messaging
 - PC manages bus and allocates slots (host controller)
 - Can have multiple host controllers on one PC
 - Support more devices than 127

USB Peripherals

<u>PERFORMANCE</u>	<u>APPLICATIONS</u>	<u>ATTRIBUTES</u>
LOW-SPEED <ul style="list-style-type: none"> • Interactive Devices • 10 – 100 kb/s 	Keyboard, Mouse Stylus Game Peripherals Virtual Reality Peripherals	Lowest Cost Ease-of-Use Dynamic Attach-Detach Multiple Peripherals
FULL-SPEED <ul style="list-style-type: none"> • Phone, Audio, Compressed Video • 500 kb/s – 10 Mb/s 	POTS Broadband Audio Microphone	Lower Cost Ease-of-Use Dynamic Attach-Detach Multiple Peripherals Guaranteed Bandwidth Guaranteed Latency
HIGH-SPEED <ul style="list-style-type: none"> • Video, Storage • 25 – 400 Mb/s 	Video Storage Imaging Broadband	Low Cost Ease-of-Use Dynamic Attach-Detach Multiple Peripherals Guaranteed Bandwidth Guaranteed Latency High Bandwidth

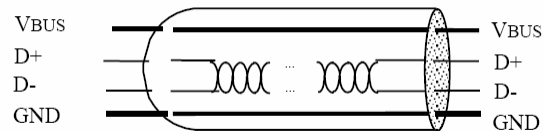
USB

- Tree of devices
– one root controller



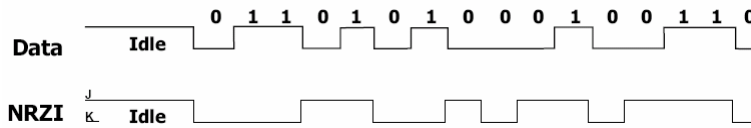
USB Data Transfer

- Data transfer speeds
 - Low is <0.8v, high is >2.0v differential
 - 480Mb/sec, 12Mb/sec, 1.5Mb/sec
 - Data is NRZI encoded (data and clock on one wire)
 - SYNC at beginning of every packet

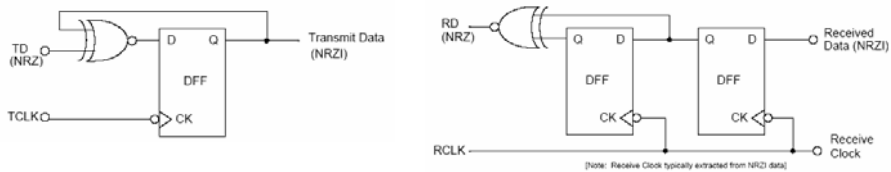
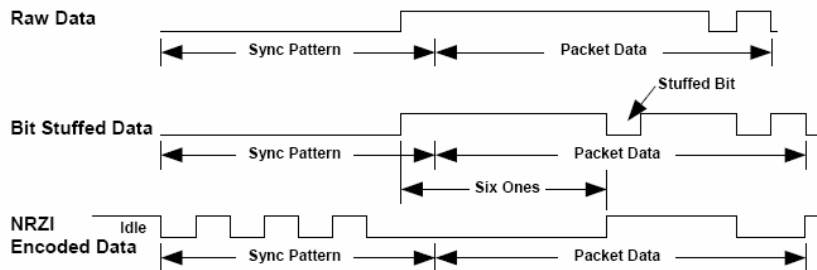


NRZI Encoding

- NRZI – Non-return to zero inverted
 - Toggles a signal to transmit a “0” and leaves the signal unchanged for a “1”
 - Also called transition encoding
 - Long string of 0s generates a regular waveform with a frequency half the bit rate
 - Long string of 1s generates a flat waveform – bit stuff a 0 every 6 consecutive 1s to guarantee activity on waveform



NRZI Encoding (cont'd)



USB Data Transfer Types

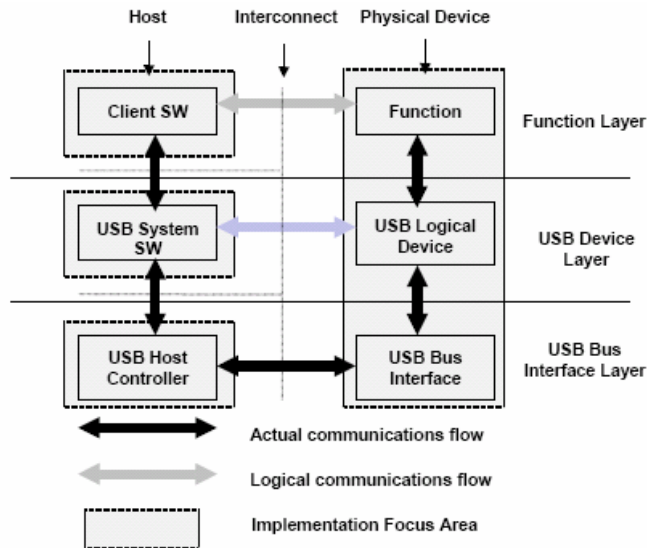
- Control Transfers:
 - Used to configure a device at attach time and can be used for other device-specific purposes, including control of other pipes on the device.
- Bulk Data Transfers:
 - Generated or consumed in relatively large and bursty quantities and have wide dynamic latitude in transmission constraints.
- Interrupt Data Transfers:
 - Used for timely but reliable delivery of data, for example, characters or coordinates with human-perceptible echo or feedback response characteristics.
- Isochronous Data Transfers:
 - Occupy a prenegotiated amount of USB bandwidth with a prenegotiated delivery latency. (Also called streaming real time transfers)

USB Packet Format

- Sync + PID + data + CRC
- Basic data packet
 - Sync: 8 bits (00000001)
 - PID: 8 bits (packet id – type)
 - Data: 8-8192 bits (1K bytes)
 - CRC: 16 bits (cyclic redundancy check sum)
- Other data packets vary in size
 - May be as short as only 8 bits of PID

USB Protocol Stack

- FTDI USB chip implements right side
- Communicates to physical device through SPI



More Communication Later

- Bluetooth
 - Popular radio frequency protocol
 - We'll discuss after looking at wireless sensors
- PCMCIA/CompactFlash
 - Popular parallel bus protocol
 - We'll discuss (time permitting) at end of quarter