Serial case studies

- **RS-232** (IEEE standard)
  - serial protocol for point-to-point, low-cost, low-speed applications for PCs
- **I2C** (Philips) TWI (Atmel)
  - 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 ±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 ATmegas

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**

```
+-----------------+            +-----------------+            +-----------------+
|                |            |                |            |                |
|    Host-specific Interface    |            |    Data-link Controller    |            |    Physical Channel    |
|                |            |                |            |                |
|    To Host    |            |    Data-link Layer    |            |    Ethernet Cable    |
|                |            |    Physical Layer    |            |                |
|                |            |                    |            |                |
|    Ethernet Controller Board    |            |    Serial Encode and Decode    |            |    Transceiver    |
|                |            |                    |            |                |
|    serial data    |            |    Transmit and Receive Electrical Interface    |            |                |
|                |            |                    |            |                |
|    Parallel data    |            |        +-----------------+            |    Physical Channel    |
|                |            |    Ethernet Controller Board    |            |
|                |            |        +-----------------+            |                |
|                |            |    Link Management    |            |                |
|                |            |                    |            |                |
|    +-----------------+            |            |    Serial Encode and Decode    |            |    Transceiver    |
|    Data Encapsulation    |            |                    |            |                |
|                |            |    Transmit and Receive Electrical Interface    |            |                |
```

CSE 466 Communication 11
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

- preamble (6 bytes)
- destination address (6 bytes)
- source address (6 bytes)
- type (2 bytes)
- data (46-1500 bytes)
- checksum (4 bytes) compute from data
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
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

<table>
<thead>
<tr>
<th>SPI Control Register – SPCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>SPIE</td>
</tr>
<tr>
<td>Read/Write</td>
</tr>
<tr>
<td>Initial Value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SPI Status Register – SP SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>SPIF</td>
</tr>
<tr>
<td>Read/Write</td>
</tr>
<tr>
<td>Initial Value</td>
</tr>
</tbody>
</table>
Using SPI as a Master

```c
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

```c
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

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

<table>
<thead>
<tr>
<th>PERFORMANCE</th>
<th>APPLICATIONS</th>
<th>ATTRIBUTES</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LOW-SPEED</strong></td>
<td>Keyboard, Mouse, Stylus, Game Peripherals, Virtual Reality Peripherals</td>
<td>Lowest Cost, Ease of Use, Dynamic Attach-Detach, Multiple Peripherals</td>
</tr>
<tr>
<td>10 – 100 kbps</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FULL-SPEED</strong></td>
<td>P078, Broadband, Audio, Microphone</td>
<td>Lowest Cost, Ease of Use, Dynamic Attach-Detach, Multiple Peripherals, Guaranteed Bandwidth, Guaranteed Latency</td>
</tr>
<tr>
<td>500 kbps – 10 Mbps</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>HIGH-SPEED</strong></td>
<td>Video, Storage, Imaging, Broadband</td>
<td>Low Cost, Ease of Use, Dynamic Attach-Detach, Multiple Peripherals, Guaranteed Bandwidth, Guaranteed Latency, High Bandwidth</td>
</tr>
<tr>
<td>25 – 400 Mbps</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

USB

- Tree of devices
  - one root controller

![Typical USB Architectural Configuration](imageURI)
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
Our USB Solution

3.0 DLP 2232M Module Simplified Block Diagram

- USB Type 'B' connector to Host PC/Mac
- MOSFET Power Switch
- 40 Pin, .6 Inch Header
- 83C56 EEPROM
- 5V Volt LDO
- FT232C
- Channel A Multi-purpose UART / FIFO Controller
- Channel B Multi-purpose UART / FIFO Controller
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