Serial Communication
Questions?
Input/Output Systems

Processor needs to communicate with other devices:

• Receive signals from sensors
• Send commands to actuators
• Or both (e.g., disks, audio, video devices, other processors)
An Example: SICK Laser Range Finder

- Laser is scanned horizontally
- Using phase information, can infer the distance to the nearest obstacle
- Resolution: ~.5 degrees, 1 cm
- Can handle full 180 degrees at 20 Hz
Serial Communication

• Communicate a set of bytes using a single signal line
• We do this by sending one bit at a time:
  – The value of the first bit determines the state of a signal line for a specified period of time
  – Then, the value of the 2\textsuperscript{nd} bit is used
  – Etc.
Serial Communication

The sender and receiver must have some way of agreeing on when a specific bit is being sent

- Some cases: the sender will also send a clock signal (on a separate line)
- Other cases: each side has a clock to tell it when to write/read a bit
  - The sender/receiver must first synchronize their clocks before transfer begins
Asynchronous Serial Communication

• The sender and receiver have their own clocks, which they do not share
• This reduces the number of signal lines

But: we still need some way to agree that data is valid. How?
Asynchronous Serial Communication

How can the two sides agree that the data is valid?

• Must both be operating at essentially the same transmit/receive frequency

• A data byte is prefaced with a bit of information that tells the receiver that bits are coming

• The receiver uses the arrival time of this start bit to synchronize its clock
A Typical Data Frame

The start bit indicates that a byte is coming
A Typical Data Frame

The stop bits allow the receiver to immediately check whether this is a valid frame

• If not, the byte is thrown away
Data Frame Handling

Most of the time, we do not deal with the data frame level. Instead, we rely on:

• Hardware solutions: Universal Asynchronous Receiver Transmitter (UART)
  – Very common in computing devices
• Software solutions in libraries
One (Old) Standard: RS232-C

Defines a logic encoding standard:

- “High” is encoded with a voltage of -5 to -15 (-12 to -13V is typical)
- “Low” is encoded with a voltage of 5 to 15 (12 to 13V is typical)
RS232 on the Teensy 3.5

Our Teensy has 7 Universal, Asynchronous serial Receiver/Transmitters (UARTs):
• #0: USB; #1 … 6: RX/TX pins
• Each handles all of the bit-level manipulation
  – Software only worries about the byte level
• 1 … 6 use 0V and 3.3V to encode “lows” and “highs”
  – Must convert if talking to a true RS232C device (+/- 13V)
Serial Initialization

Options include:

- `Serial.begin(9600);`
- `SerialX.begin(9600);`
  - Where X = 1 … 6
Generating Serial Output

```cpp
int val = 42;
float f = 6.282;

Serial.println("foo:");
Serial.println(val);
Serial.printf("foo: %d (%f)\n", val, f);
```
Reading Serial Input

• Serial.read() will return the next character in the buffer
• If the buffer is empty, then this function will block until a character is available to be read
• This can be very dangerous in a real-time domain
Checking for Characters

What we would like to do is to ask ahead of time as to whether a character is ready to be read …
Checking for Characters

What we would like to do is to ask ahead of time as to whether a character is ready to be read ...

```java
loop() {
    if(Serial.available()) {
        char c = Serial.read();
        <do something with the read char>
    }
    <do something else while waiting>
}
```
Character Representation

• A “char” is just an 8-bit number
• This allows us to perform meaningful mathematical operations on the characters
Character Representation: ASCII
Serial Challenge

• Suppose that we know that we will be receiving a sequence of 3 decimal digits from the serial port

• How do we translate these digits into an integer representation?
Serial Challenge II

• Suppose that we know that we will be receiving a sequence of k decimal digits from the serial port

• How do we translate these digits into an integer representation?

• Can assume that the digits will fit within a uint16_t
Character Representation: ASCII
Synchronous Serial Communication
Synchronous Serial Communication

- A clock signal is also provided
- This allows for very fast communication
- Client/server model of communication: one side (the client) is in control of when/what are communicated
  - Client initiates any data transfer and provides the clock signal
  - Also referred to as a “master/slave” model
Serial Peripheral Interface (SPI)

Signal lines:
- SCK: serial clock
- MOSI: master-out-slave-in: communication of data from client to server
- MISO: master-in-slave-out: server to client
- CS: chip select: client brings this line low before data are exchanged
Serial Peripheral Interface (SPI)

- Servers can only transmit/receive data when CS is low
- Data exchange happens simultaneously
- Only one client in the circuit
- Servers can be daisy-chained into a single circuit
- Teensy has hardware support for the
Inter Integrated Circuit (I2C)

Signals:
• SCL: clock signal
• SDA: data signal

Servers have unique addresses (ID numbers) that are used by the client to initiate the conversation
Inter Integrated Circuit (I2C)

- Both the client and the server write to the data bus:
  - First the client writes data
  - Followed by the server writing data
- Multiple clients can exist
- The client always provides the clock signal
- Support for I2C in hardware on the Teensy
Controller Area Network

- Communication across devices that are separated by some distance (10s of meters)
- Can function in electrically noisy environments
- Slow communication speeds (compared to I2C and SPI)
- Client/server model, but servers are not explicitly addressed. Instead, message types are addressed