Today

- A bit more on bit masking
- Communicating between devices
  - Serial communication
- Project 1 is due on Thursday
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)
I/O Systems

Communication can happen in a variety of ways:

- Binary parallel signal
- Analog
- Serial signals
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

• Typically, each side has a clock to tell it when to write/read a bit

• In some cases, the sender will also send a clock signal (on a separate line)

• In other cases, the sender/receiver will 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
• Bidirectional transmission, but the two halves do not need to be synchronized in time

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 data is 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 personally 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 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 Mega8

Our mega 8 has a Universal, Asynchronous serial Receiver/Transmitter (UART)

• Handles all of the bit-level manipulation
• You only have to interact with it on the byte level
• Uses 0V and 5V to encode “lows” and “highs”
  – Must convert if talking to an RS232C device
Mega8 UART C Interface

Lib C support (standard C):
getchar(): receive a character

putchar(’a’): put a character out to the port

puts("foobar"): put a string out to the port

printf("foobar %d %s", 45, "baz"): put a formatted string out to the port (not recommended for the atmels)
Mega8 UART C Interface

OUlib support:

```c
fp = serial_init_buffered(0, 9600, 10, 10)
    Initialize the port @9600 bits per second (input and output buffers are both 10 characters long)

serial_buffered_input_waiting(fp)
    Is there a character in the buffer?
```

See the Atmel HOWTO: examples/serial
Summary: Using OULib + LibC

• At the top of your source file:
  
  ```
  #include "oulib_serial_buffered.h"
  ```

• Initialization (in your main() function):
  ```
  fp = serial_init_buffered(0, 9600, 10, 10)
  sei();
  ```

• Getting a character:
  ```
  char c;
  C = getchar();
  ```

• Sending a character:
  ```
  putchar('f');
  ```
Character Representation

• A “char” is just an 8-bit number
• In some cases, we just interpret it differently.
• But: we can still perform mathematical operations on it
## Character Representation: ASCII

<table>
<thead>
<tr>
<th>Binary</th>
<th>Dec</th>
<th>Hex</th>
<th>Glyph</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>1101</td>
<td>D</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>1110</td>
<td>E</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

- **Binary**: 8-bit representation of characters.
- **Dec**: Decimal value of the character.
- **Hex**: Hexadecimal value of the character.
- **Glyph**: ASCII character for each binary code.
Mega8 UART
Mega8 UART

- Transmit pin (PD1)
Mega8 UART

- Transmit pin (PD1)
- Transmit shift register
Writing a Byte to the Serial Port

`putchar('A');`
Transmit

`putchar('A');`
Transmit

When UART is ready, the buffer contents are copied to the shift register.

01000001

01000001
Transmit

The least significant bit (LSB) of the shift register determines the state of the pin.
Transmit

After a delay, the UART shifts the values to the right

\[ x = \text{value doesn't matter} \]
Transmit

Next shift

xx010000
Transmit

Several shifts later…
Receive

- Receive pin (PD0)
Receive

- Receive pin (PD0)
- Receive shift register
Receive

- “1” on the pin
- Shift register initially in an unknown state
Receive

“1” is presented to the shift register
Receive

“1” is shifted into the most significant bit (msb) of the shift register.

Andrew H. Fagg: Embedded Real-Time Systems: Serial Comm 50
Receive

Next bit is shifted in
Receive

And the next bit...
Receive

And the 8th bit

01101011

0
Receive

Completed byte is stored in the UART buffer.
Reading a Byte from the Serial Port

```c
int c;

c=getchar();
```
Receive

`getchar()` retrieves this byte from the buffer
Reading a Byte from the Serial Port

```c
int c;

c=getchar();
```

Note: getchar() “blocks” until a byte is available

• Will only return with a value once one is available to be returned
Processing Serial Input

int c;
while(1) {
    if(serial_buffered_input_waiting(fp)) {
        // A character is available for reading
        c = getchar();
        <do something with the character>
    }
    <do something else while waiting>
}

serial_buffered_input_waiting(fp) tells us whether a byte is ready to be read
Mega8 UART C Interface

printf() : formatted output
scanf() : formatted input

(available, but not recommended for the atmels)

See the LibC documentation or the AVR C textbook
Physical Interface

On our Atmels: +5V standard ("TTL")

• Pin 2: receive (PD0)
• Pin 3: transmit (PD1)

USB-2-RS232 board:

• Provides transmit/receive pins for the +5V standard
• Allows you to “talk” to your atmel chip through a terminal program

Compass: also speaks the +5V standard