

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: $\sim .5$ 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 2nd 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

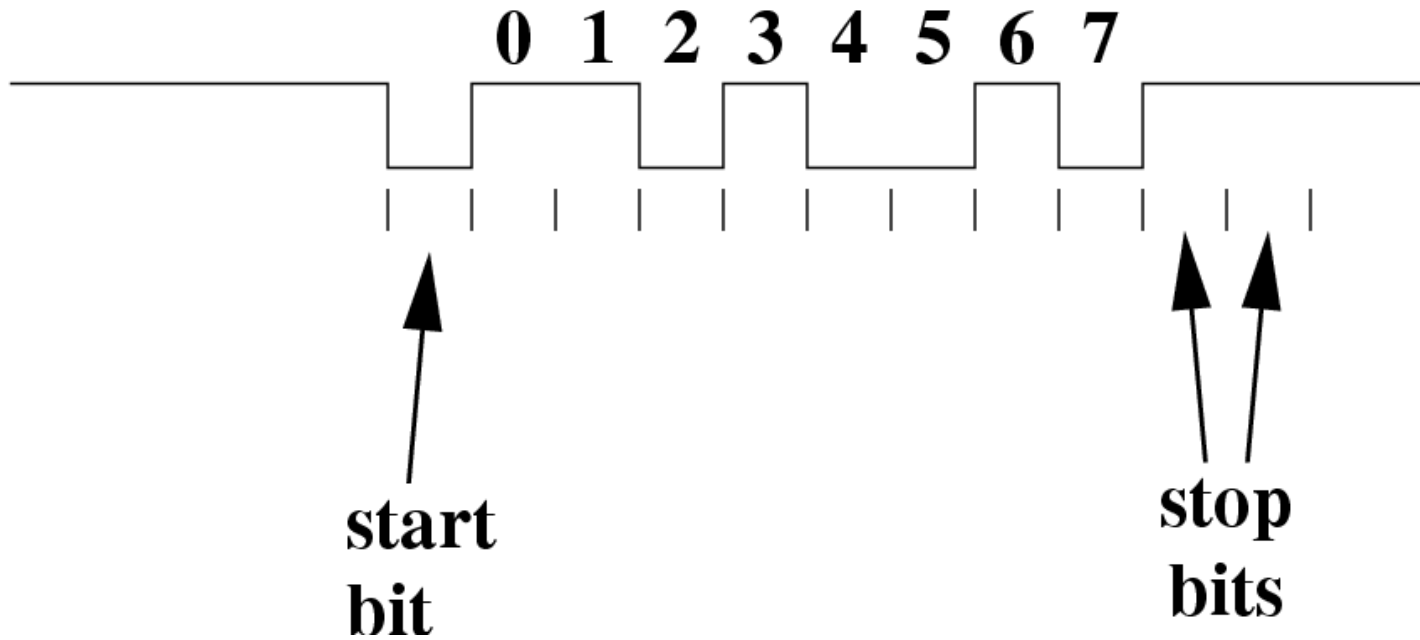
But: we still need some way to agree that data is valid. How?

Asynchronous Serial Communication

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

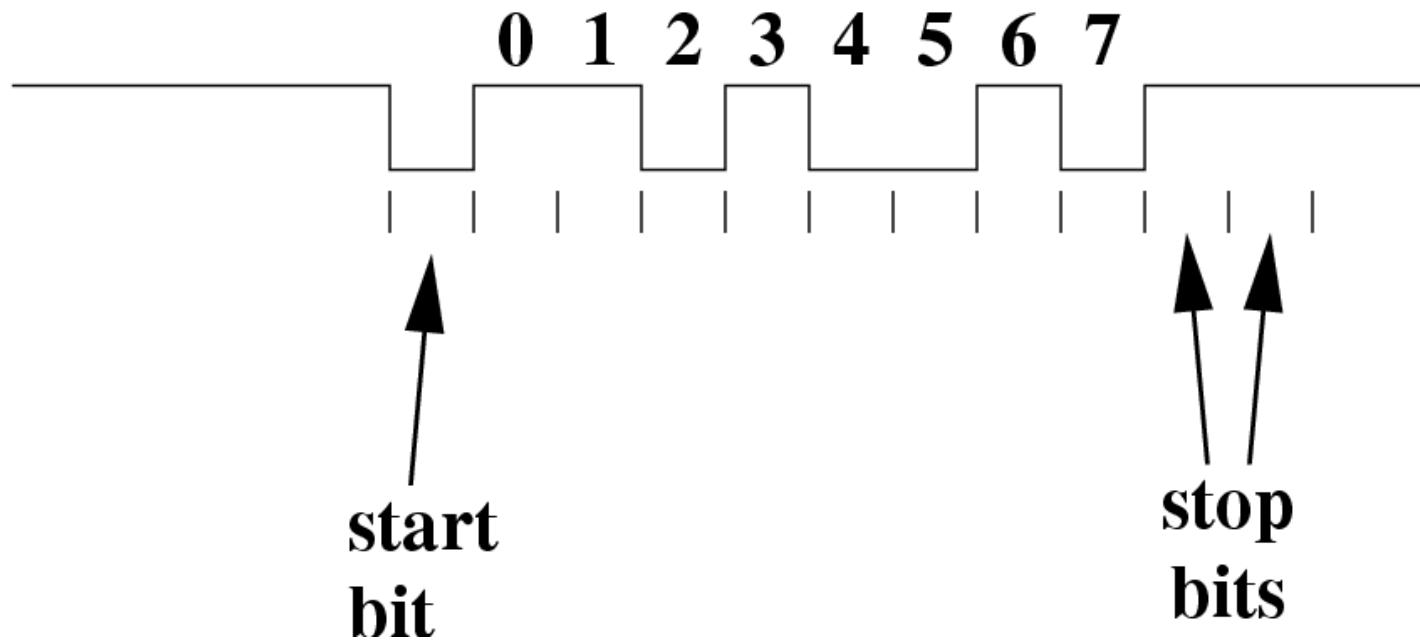
- Must both be operating at essentially the same transmit/receive frequency
- A data byte is prefaced with a bit of information that tells the receiver that data is coming
- The receiver uses the arrival time of this **start bit** to synchronize its clock

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 Mega2560

Our mega 8 has FOUR Universal, Asynchronous serial Receiver/Transmitters (UARTs):

- Each 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 a true RS232C device (+/- 13V)

Mega2560 UART C Interface

OULib support:

```
fp = serial_init_buffered(1, 9600, 40, 40)
```

Initialize port one for a transmission rate of 9600 bits per second (input and output buffers are both 40 characters long)

```
serial_buffered_input_waiting(fp)
```

Is there a character in the buffer?

See the Atmel HOWTO: [examples_2560/serial](#)

Mega2560 UART C Interface

Lib C support (standard C):

`char fgetc(fp) : receive a character`

`fputc('a', fp) : put a character out to the port`

`fputs("foobar", fp) : put a string out to the port`

`fprintf(fp, "foobar %d %s", 45, "baz") :
put a formatted string out to the port`

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(1, 9600, 40, 40)
sei();
```

- Getting a character:

```
char c;
C = fgetc(fp);
```

- Sending a character:

```
fputc('f', fp);
```

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

| Binary | Dec | Hex | Glyph |
|----------|-----|-----|-------|
| 010 0000 | 32 | 20 | SP |
| 010 0001 | 33 | 21 | ! |
| 010 0010 | 34 | 22 | " |
| 010 0011 | 35 | 23 | # |
| 010 0100 | 36 | 24 | \$ |
| 010 0101 | 37 | 25 | % |
| 010 0110 | 38 | 26 | & |
| 010 0111 | 39 | 27 | ' |
| 010 1000 | 40 | 28 | (|
| 010 1001 | 41 | 29 |) |
| 010 1010 | 42 | 2A | * |
| 010 1011 | 43 | 2B | + |
| 010 1100 | 44 | 2C | , |
| 010 1101 | 45 | 2D | - |
| 010 1110 | 46 | 2E | . |
| 010 1111 | 47 | 2F | / |
| 011 0000 | 48 | 30 | 0 |
| 011 0001 | 49 | 31 | 1 |
| 011 0010 | 50 | 32 | 2 |
| 011 0011 | 51 | 33 | 3 |
| 011 0100 | 52 | 34 | 4 |
| 011 0101 | 53 | 35 | 5 |
| 011 0110 | 54 | 36 | 6 |
| 011 0111 | 55 | 37 | 7 |
| 011 1000 | 56 | 38 | 8 |
| 011 1001 | 57 | 39 | 9 |
| 011 1010 | 58 | 3A | : |
| 011 1011 | 59 | 3B | ; |
| 011 1100 | 60 | 3C | < |
| 011 1101 | 61 | 3D | = |
| 011 1110 | 62 | 3E | > |
| 011 1111 | 63 | 3F | ? |

| Binary | Dec | Hex | Glyph |
|----------|-----|-----|-------|
| 100 0000 | 64 | 40 | @ |
| 100 0001 | 65 | 41 | A |
| 100 0010 | 66 | 42 | B |
| 100 0011 | 67 | 43 | C |
| 100 0100 | 68 | 44 | D |
| 100 0101 | 69 | 45 | E |
| 100 0110 | 70 | 46 | F |
| 100 0111 | 71 | 47 | G |
| 100 1000 | 72 | 48 | H |
| 100 1001 | 73 | 49 | I |
| 100 1010 | 74 | 4A | J |
| 100 1011 | 75 | 4B | K |
| 100 1100 | 76 | 4C | L |
| 100 1101 | 77 | 4D | M |
| 100 1110 | 78 | 4E | N |
| 100 1111 | 79 | 4F | O |
| 101 0000 | 80 | 50 | P |
| 101 0001 | 81 | 51 | Q |
| 101 0010 | 82 | 52 | R |
| 101 0011 | 83 | 53 | S |
| 101 0100 | 84 | 54 | T |
| 101 0101 | 85 | 55 | U |
| 101 0110 | 86 | 56 | V |
| 101 0111 | 87 | 57 | W |
| 101 1000 | 88 | 58 | X |
| 101 1001 | 89 | 59 | Y |
| 101 1010 | 90 | 5A | Z |
| 101 1011 | 91 | 5B | [|
| 101 1100 | 92 | 5C | \ |
| 101 1101 | 93 | 5D |] |
| 101 1110 | 94 | 5E | ^ |
| 101 1111 | 95 | 5F | _ |

| Binary | Dec | Hex | Glyph |
|----------|-----|-----|-------|
| 110 0000 | 96 | 60 | ` |
| 110 0001 | 97 | 61 | a |
| 110 0010 | 98 | 62 | b |
| 110 0011 | 99 | 63 | c |
| 110 0100 | 100 | 64 | d |
| 110 0101 | 101 | 65 | e |
| 110 0110 | 102 | 66 | f |
| 110 0111 | 103 | 67 | g |
| 110 1000 | 104 | 68 | h |
| 110 1001 | 105 | 69 | i |
| 110 1010 | 106 | 6A | j |
| 110 1011 | 107 | 6B | k |
| 110 1100 | 108 | 6C | l |
| 110 1101 | 109 | 6D | m |
| 110 1110 | 110 | 6E | n |
| 110 1111 | 111 | 6F | o |
| 111 0000 | 112 | 70 | p |
| 111 0001 | 113 | 71 | q |
| 111 0010 | 114 | 72 | r |
| 111 0011 | 115 | 73 | s |
| 111 0100 | 116 | 74 | t |
| 111 0101 | 117 | 75 | u |
| 111 0110 | 118 | 76 | v |
| 111 0111 | 119 | 77 | w |
| 111 1000 | 120 | 78 | x |
| 111 1001 | 121 | 79 | y |
| 111 1010 | 122 | 7A | z |
| 111 1011 | 123 | 7B | { |
| 111 1100 | 124 | 7C | |
| 111 1101 | 125 | 7D | } |
| 111 1110 | 126 | 7E | ~ |

Andrew H. Fag,
Time System

Reading a Byte from the Serial Port

```
int c;
```

```
c=fgetc(fp);
```

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

- Will only return with a value once a character 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 = fgetc(fp);
        <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

Mega2560 UART C Interface

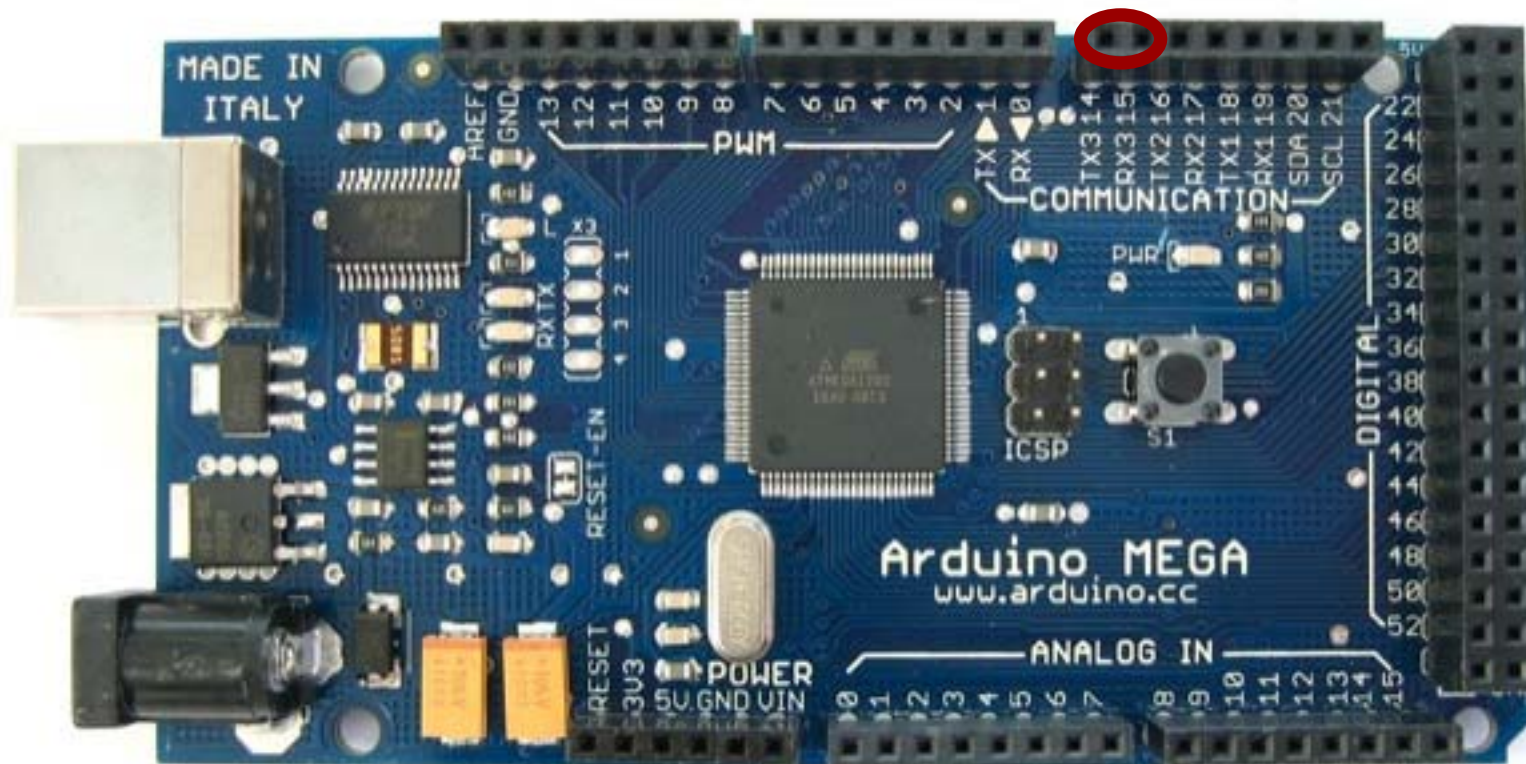
Also available:

- `fscanf ()` : formatted input

See the LibC documentation or the AVR C textbook

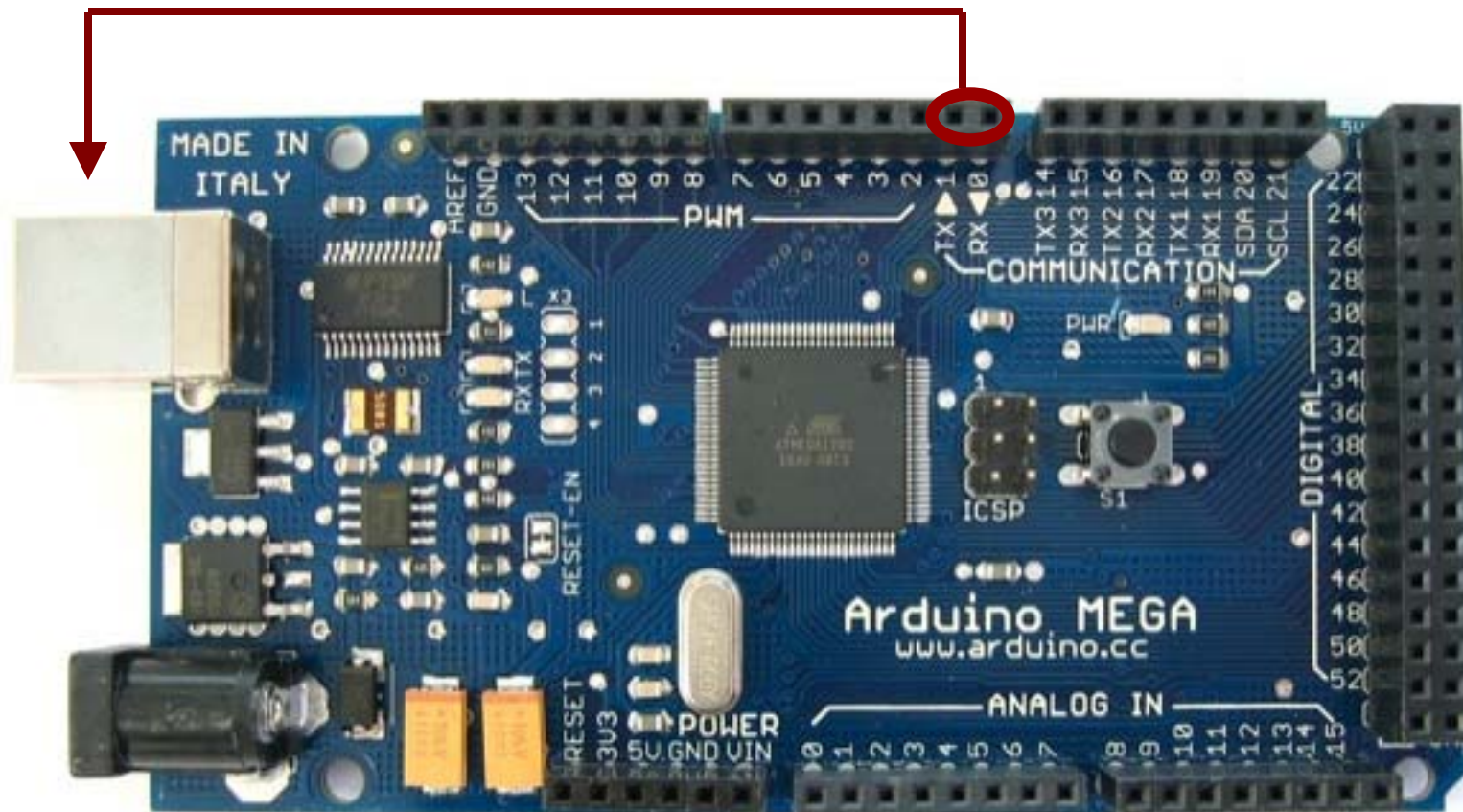
Physical Interface

Four matched pairs of transmit and receive pins (TX? and RX?)



Physical Interface

Port 0 is also connected to the USB port



See “hyperterm” on downloads page

Physical Interface

Our compass module also speaks 0/5 V
RS232

- See project 2 for details