

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: $\sim .5$ 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

- 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

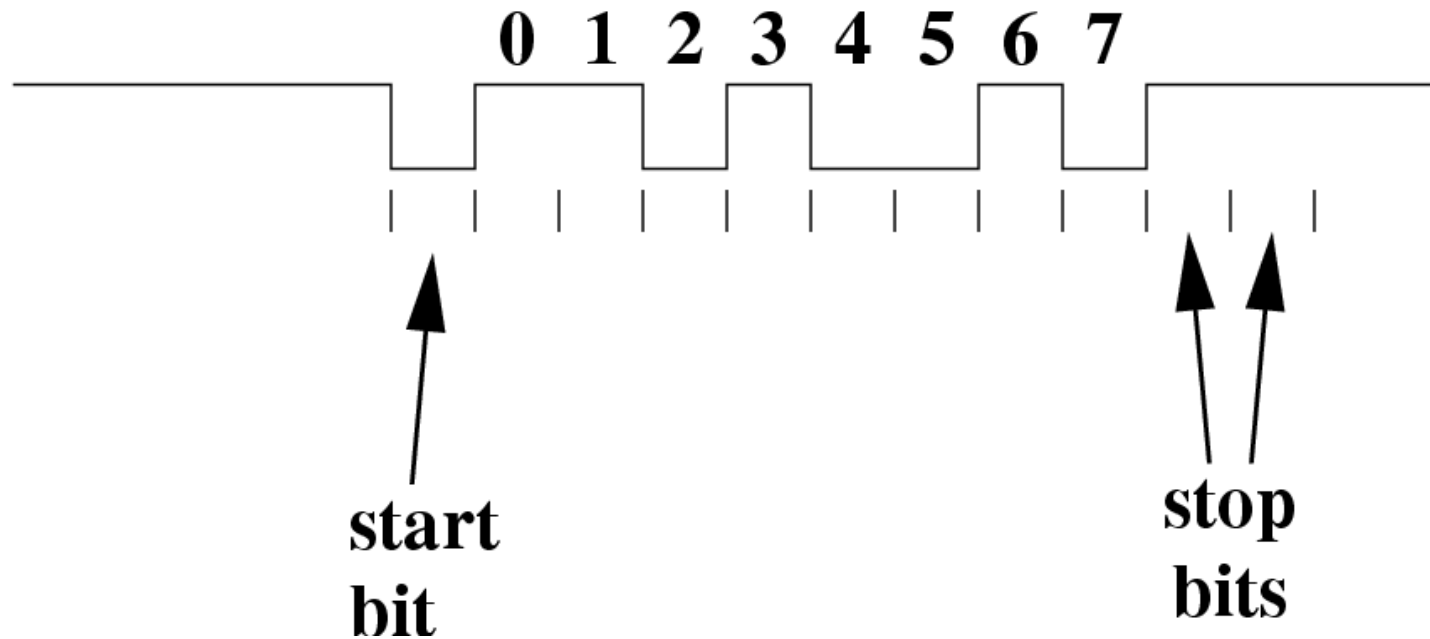
But: we still need some way to agree that data is valid. How?

Asynchronous Serial Communication

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

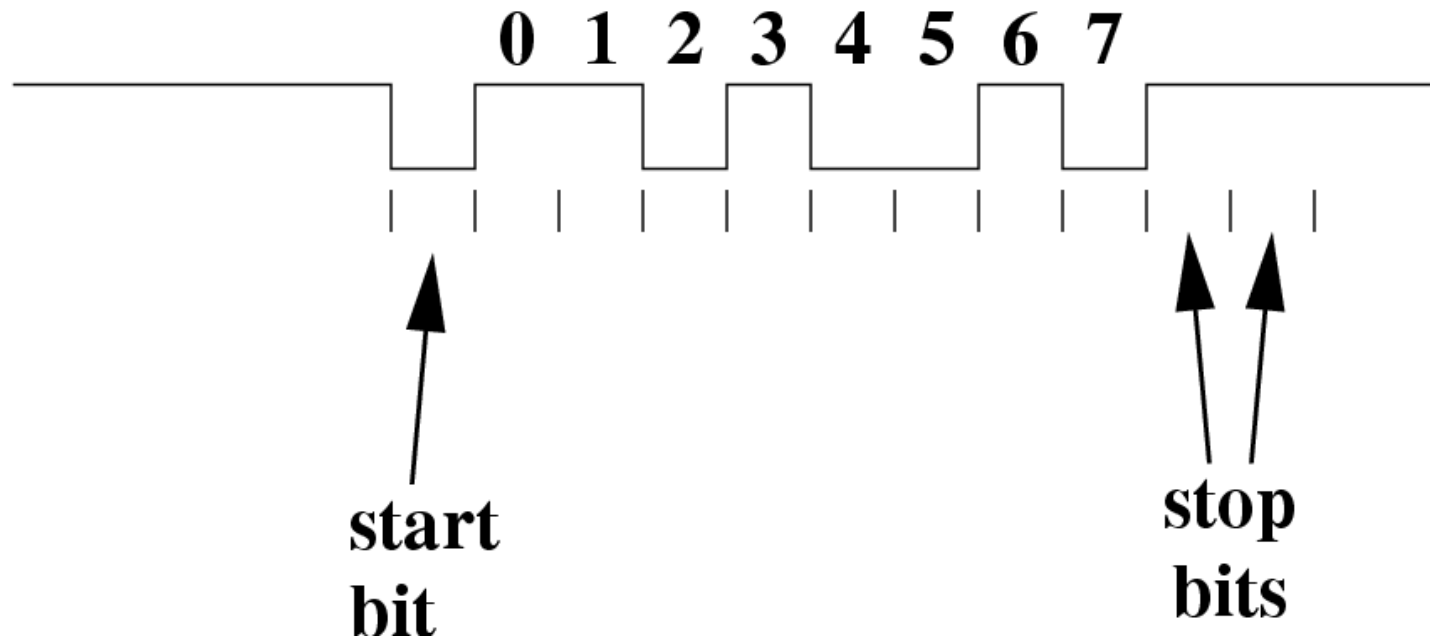
- Must both be operating at essentially the same transmit/receive frequency
- A data byte is prefaced with a bit of information that tells the receiver that bits are coming
- The receiver uses the arrival time of this **start bit** to synchronize its clock

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 Mega2560

Our mega 2560 has FOUR Universal, Asynchronous serial Receiver/Transmitters (UARTs):

- Each handles all of the bit-level manipulation
 - Software only worries about 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

Lib C support (standard C):

`char fgetc(fp) :` receive a character from a port

`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

Mega2560 UART C Interface

OUlib support:

```
fp = serial_init_buffered(1, 38400, 40, 40)
```

Initialize port one for a transmission rate of 38400 bits per second (input and output buffers are both 40 characters long)

Note: declare fp as a global variable:

```
FILE *fp;
```

```
serial_buffered_input_waiting(fp)
```

Is there a character in the buffer?

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

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

Character Representation

- A “char” is just an 8-bit number
- This allows us to perform meaningful mathematical operations on the characters

Character Representation: ASCII

Andrew H. Fag
Time System

Binary	Dec	Hex	Glyph	Binary	Dec	Hex	Glyph	Binary	Dec	Hex	Glyph
010 0000	32	20	SP	100 0000	64	40	@	110 0000	96	60	`
010 0001	33	21	!	100 0001	65	41	A	110 0001	97	61	a
010 0010	34	22	"	100 0010	66	42	B	110 0010	98	62	b
010 0011	35	23	#	100 0011	67	43	C	110 0011	99	63	c
010 0100	36	24	\$	100 0100	68	44	D	110 0100	100	64	d
010 0101	37	25	%	100 0101	69	45	E	110 0101	101	65	e
010 0110	38	26	&	100 0110	70	46	F	110 0110	102	66	f
010 0111	39	27	'	100 0111	71	47	G	110 0111	103	67	g
010 1000	40	28	(	100 1000	72	48	H	110 1000	104	68	h
010 1001	41	29	)	100 1001	73	49	I	110 1001	105	69	i
010 1010	42	2A	*	100 1010	74	4A	J	110 1010	106	6A	j
010 1011	43	2B	+	100 1011	75	4B	K	110 1011	107	6B	k
010 1100	44	2C	,	100 1100	76	4C	L	110 1100	108	6C	l
010 1101	45	2D	-	100 1101	77	4D	M	110 1101	109	6D	m
010 1110	46	2E	.	100 1110	78	4E	N	110 1110	110	6E	n
010 1111	47	2F	/	100 1111	79	4F	O	110 1111	111	6F	o
011 0000	48	30	0	101 0000	80	50	P	111 0000	112	70	p
011 0001	49	31	1	101 0001	81	51	Q	111 0001	113	71	q
011 0010	50	32	2	101 0010	82	52	R	111 0010	114	72	r
011 0011	51	33	3	101 0011	83	53	S	111 0011	115	73	s
011 0100	52	34	4	101 0100	84	54	T	111 0100	116	74	t
011 0101	53	35	5	101 0101	85	55	U	111 0101	117	75	u
011 0110	54	36	6	101 0110	86	56	V	111 0110	118	76	v
011 0111	55	37	7	101 0111	87	57	W	111 0111	119	77	w
011 1000	56	38	8	101 1000	88	58	X	111 1000	120	78	x
011 1001	57	39	9	101 1001	89	59	Y	111 1001	121	79	y
011 1010	58	3A	:	101 1010	90	5A	Z	111 1010	122	7A	z
011 1011	59	3B	;	101 1011	91	5B	[	111 1011	123	7B	{
011 1100	60	3C	<	101 1100	92	5C	\	111 1100	124	7C	
011 1101	61	3D	=	101 1101	93	5D	]	111 1101	125	7D	}
011 1110	62	3E	>	101 1110	94	5E	^	111 1110	126	7E	~
011 1111	63	3F	?	101 1111	95	5F	_				

Buffers

A buffer is an array that temporarily stores data in sequential order

```
fp = serial_init_buffered(1, 38400, 40, 40)
```

- Declares both the input and output buffer sizes to be 40 bytes

Output Buffer

- Any characters that are produced (e.g., with `fputc()` or `fprintf()`) are first placed in the output buffer
- Then, the serial hardware removes one byte at a time to send it

Output Buffer

- Advantage: `fputc()` and `fprintf()` don't have to wait for the bytes to be transmitted
 - Your program can keep doing the rest of its job
- But: if the buffer fills up, these functions will block until there is space
 - You must choose your buffer size somewhat carefully

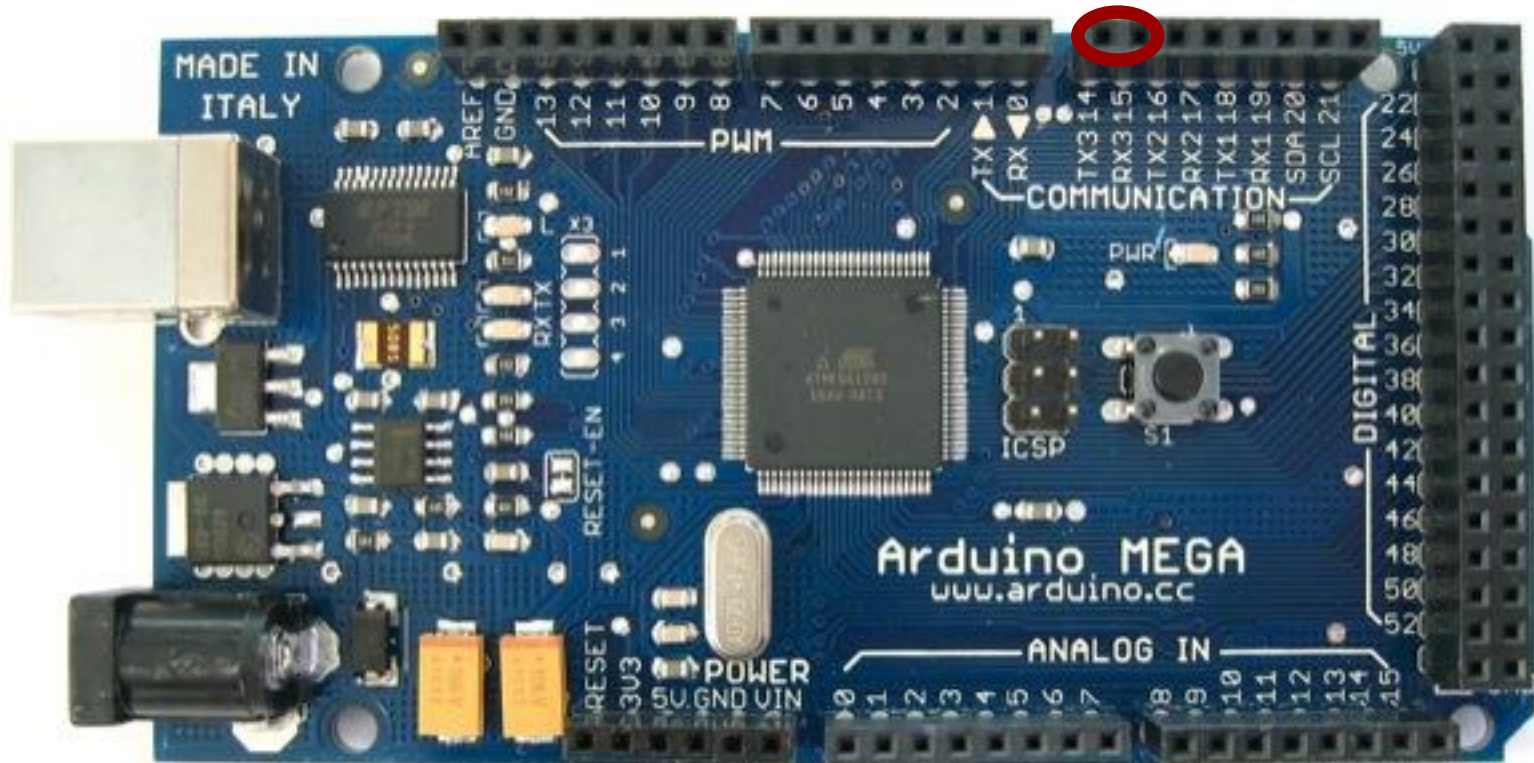
Input Buffer

Temporary storage of bytes as they are received

- Your program can read these bytes at its leisure
- With OULIB: if the buffer fills up, then additional bytes will be lost

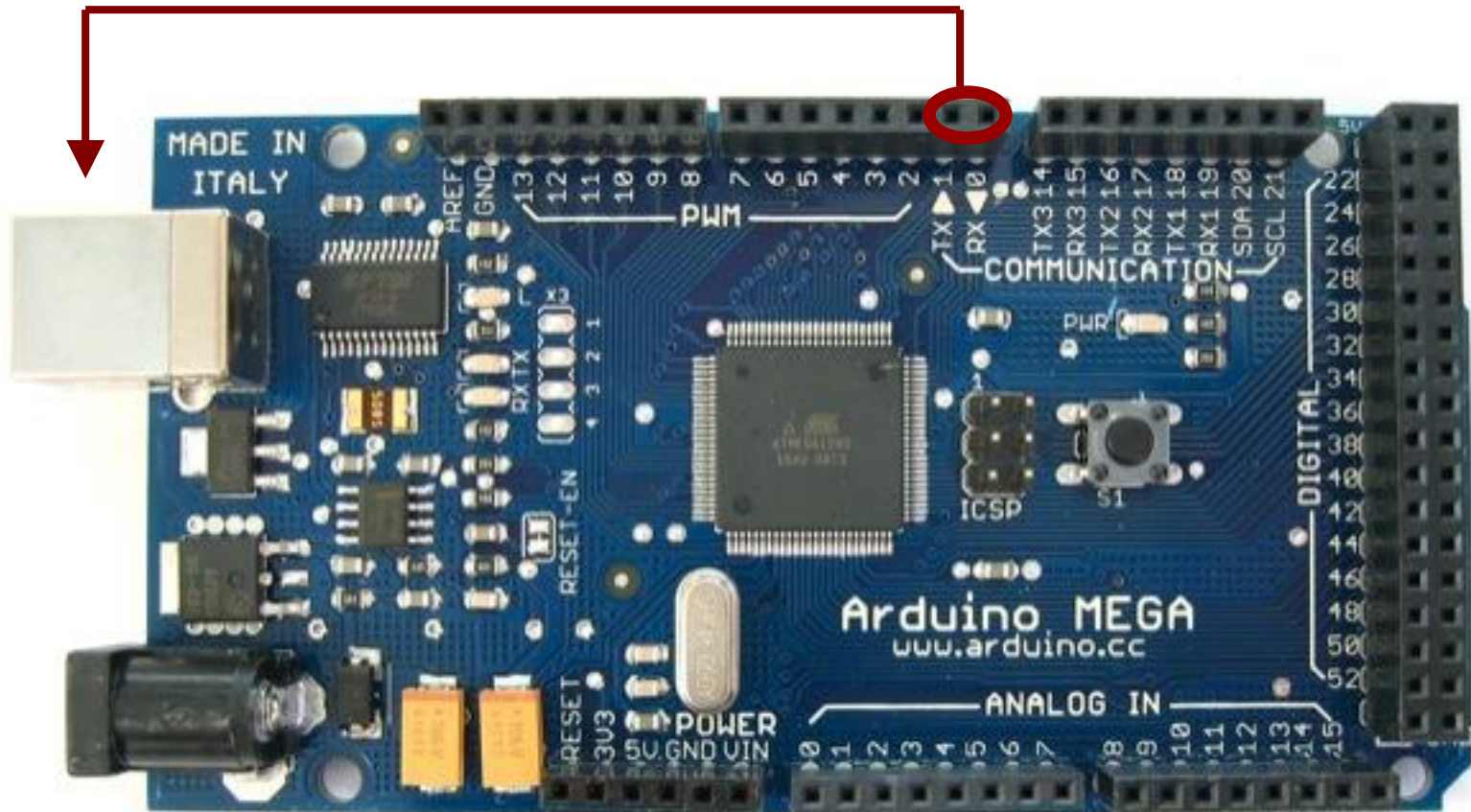
Physical Interface

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



Physical Interface

Port 0 is also connected to the USB port



See “realterm” on downloads page ^{eal-}₁

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?
- Bonus: what if we don't know how many digits are coming? (we read digits until a non-digit is read)

Character Representation: ASCII

Andrew H. Fag
Time System

Binary	Dec	Hex	Glyph	Binary	Dec	Hex	Glyph	Binary	Dec	Hex	Glyph
010 0000	32	20	SP	100 0000	64	40	@	110 0000	96	60	`
010 0001	33	21	!	100 0001	65	41	A	110 0001	97	61	a
010 0010	34	22	"	100 0010	66	42	B	110 0010	98	62	b
010 0011	35	23	#	100 0011	67	43	C	110 0011	99	63	c
010 0100	36	24	\$	100 0100	68	44	D	110 0100	100	64	d
010 0101	37	25	%	100 0101	69	45	E	110 0101	101	65	e
010 0110	38	26	&	100 0110	70	46	F	110 0110	102	66	f
010 0111	39	27	'	100 0111	71	47	G	110 0111	103	67	g
010 1000	40	28	(	100 1000	72	48	H	110 1000	104	68	h
010 1001	41	29	)	100 1001	73	49	I	110 1001	105	69	i
010 1010	42	2A	*	100 1010	74	4A	J	110 1010	106	6A	j
010 1011	43	2B	+	100 1011	75	4B	K	110 1011	107	6B	k
010 1100	44	2C	,	100 1100	76	4C	L	110 1100	108	6C	l
010 1101	45	2D	-	100 1101	77	4D	M	110 1101	109	6D	m
010 1110	46	2E	.	100 1110	78	4E	N	110 1110	110	6E	n
010 1111	47	2F	/	100 1111	79	4F	O	110 1111	111	6F	o
011 0000	48	30	0	101 0000	80	50	P	111 0000	112	70	p
011 0001	49	31	1	101 0001	81	51	Q	111 0001	113	71	q
011 0010	50	32	2	101 0010	82	52	R	111 0010	114	72	r
011 0011	51	33	3	101 0011	83	53	S	111 0011	115	73	s
011 0100	52	34	4	101 0100	84	54	T	111 0100	116	74	t
011 0101	53	35	5	101 0101	85	55	U	111 0101	117	75	u
011 0110	54	36	6	101 0110	86	56	V	111 0110	118	76	v
011 0111	55	37	7	101 0111	87	57	W	111 0111	119	77	w
011 1000	56	38	8	101 1000	88	58	X	111 1000	120	78	x
011 1001	57	39	9	101 1001	89	59	Y	111 1001	121	79	y
011 1010	58	3A	:	101 1010	90	5A	Z	111 1010	122	7A	z
011 1011	59	3B	;	101 1011	91	5B	[	111 1011	123	7B	{
011 1100	60	3C	<	101 1100	92	5C	\	111 1100	124	7C	
011 1101	61	3D	=	101 1101	93	5D	]	111 1101	125	7D	}
011 1110	62	3E	>	101 1110	94	5E	^	111 1110	126	7E	~
011 1111	63	3F	?	101 1111	95	5F	_				