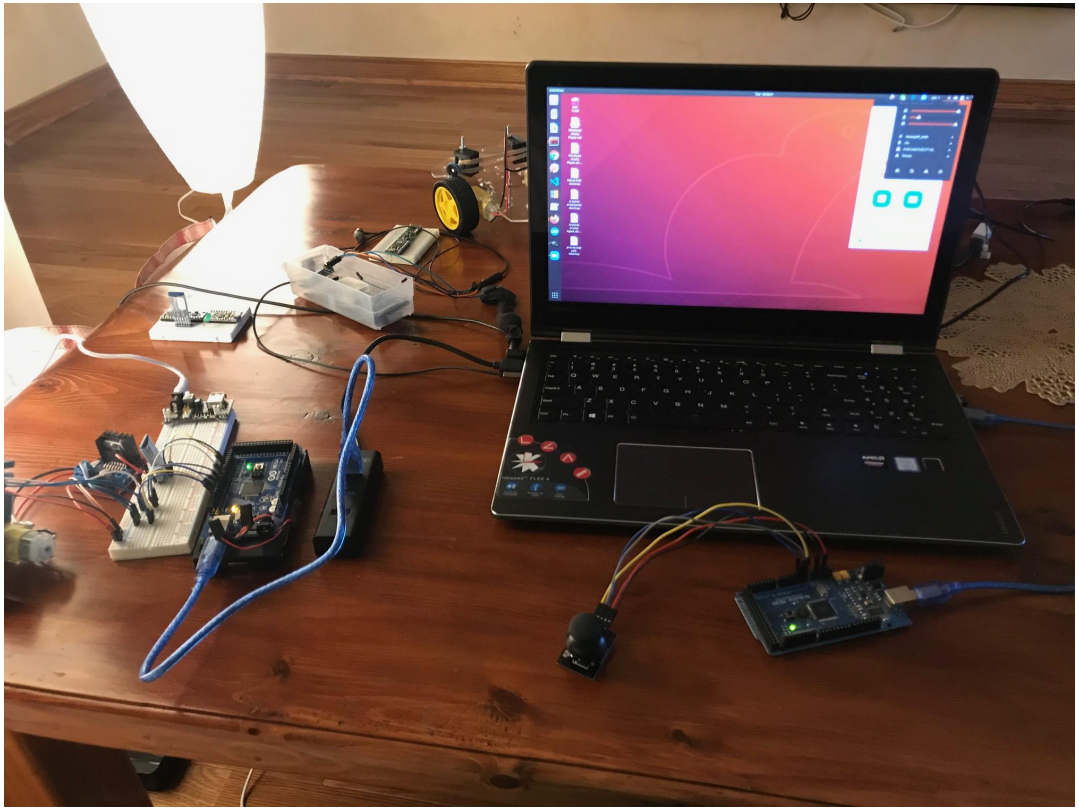
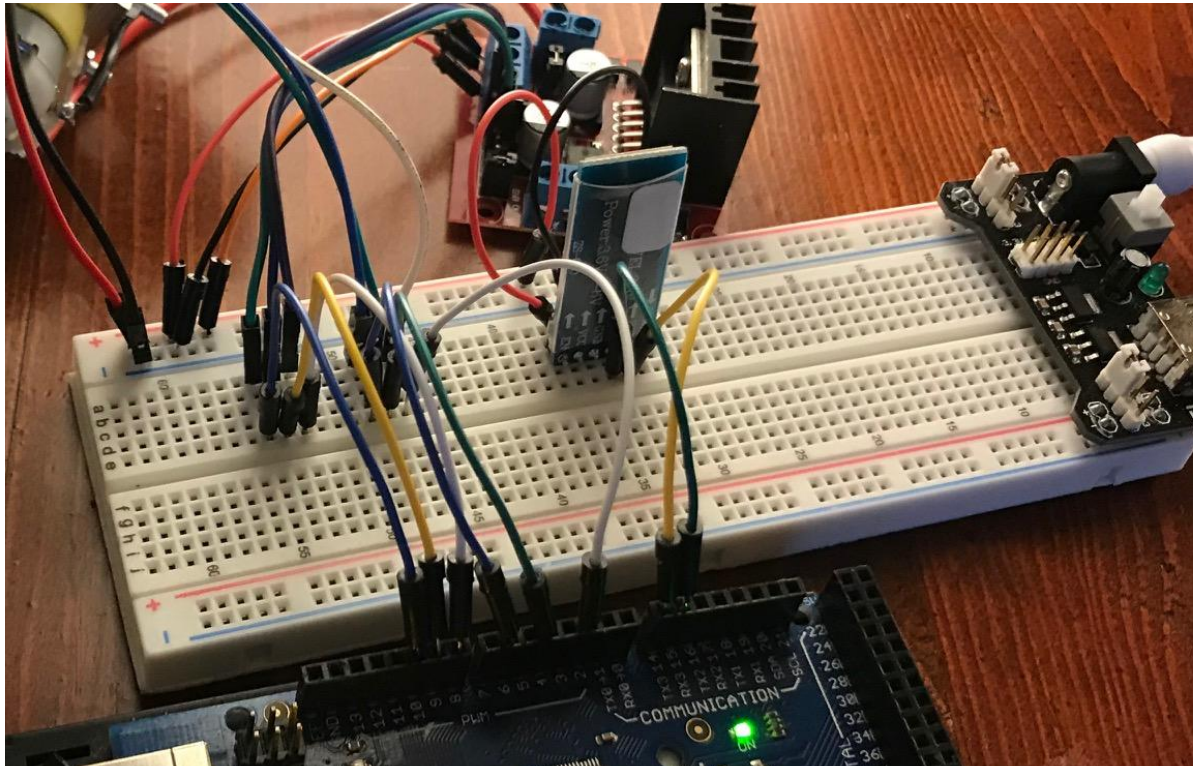
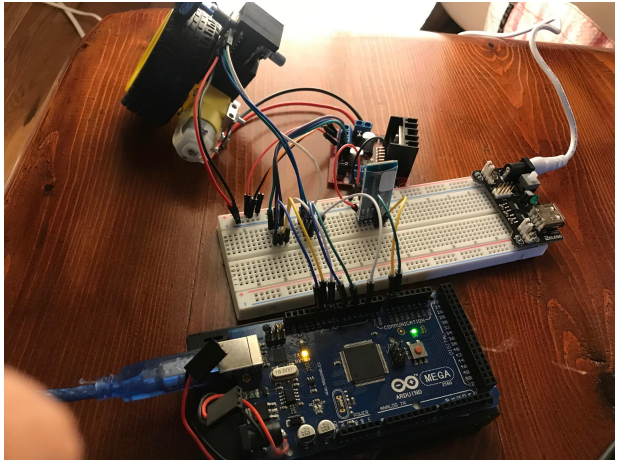
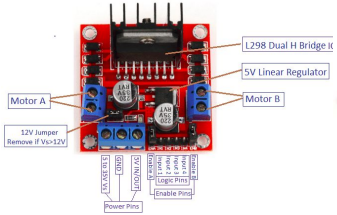
Arduino motor controlling through Bluetooth joystick

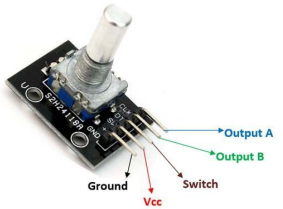


Arduino Wiring for DC Motor control



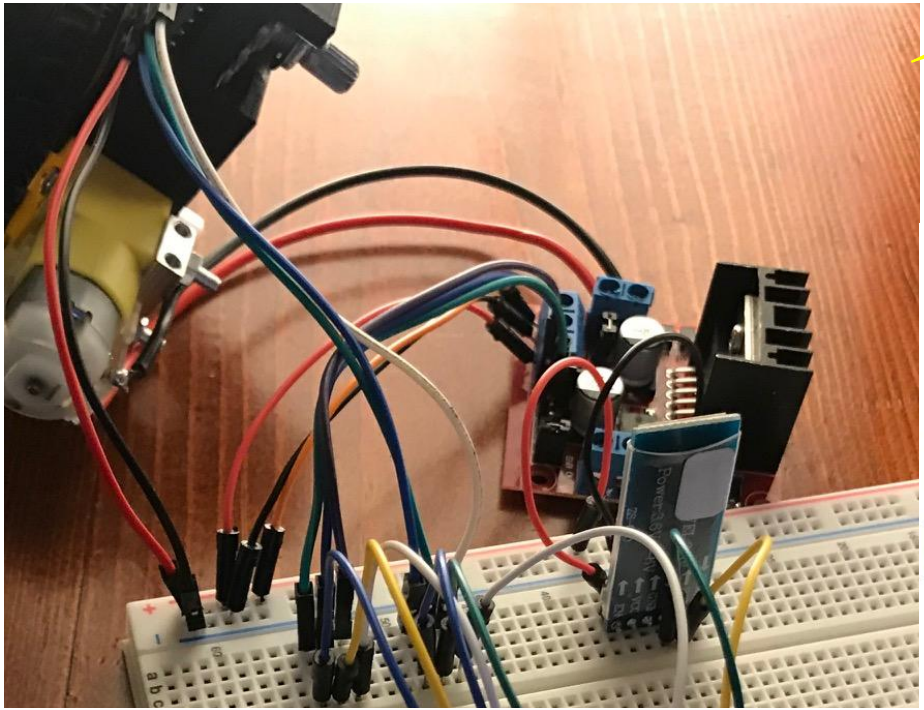


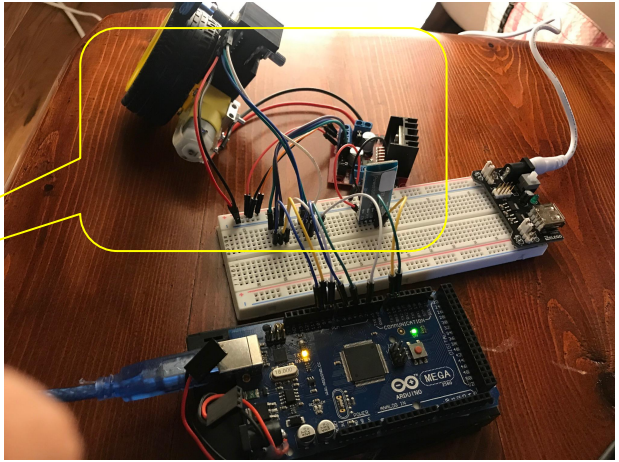


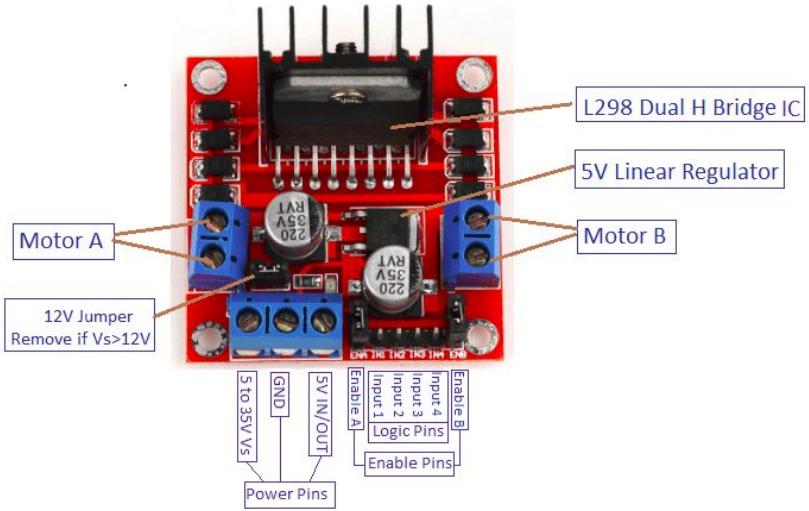


Arduino Wiring for DC Motor control

H-Bridge Motor Driver to Breadboard



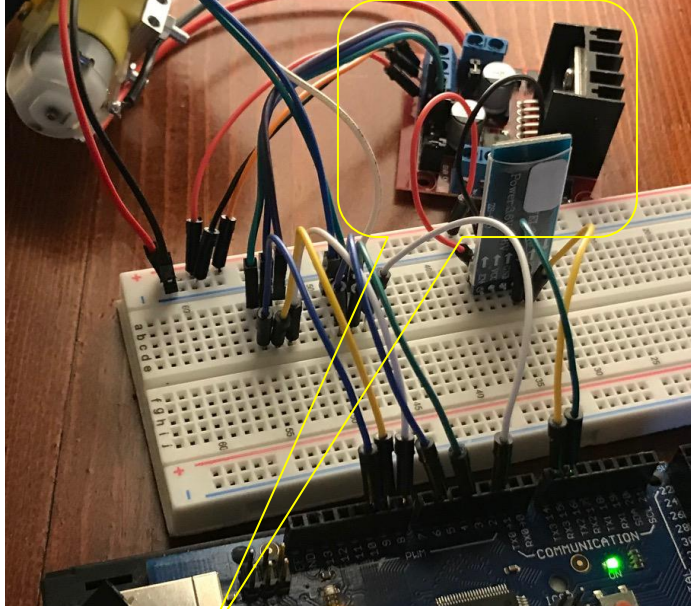


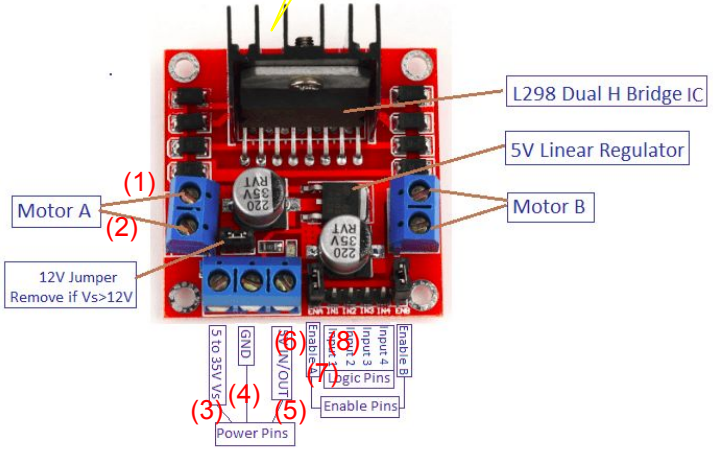


Arduino Wiring for DC Motor control

H-Bridge Motor Driver <--> Breadboard <--> Arduino

Motor A	Motor Driver	Breadboard	Arduino
Motor A (black)	Motor A (1)	--	--
Motor A (red)	Motor A (2)	--	--
	5 ~ 35 V -Motor Power (3)	Redline Bus	--
	GND (4)	Blueline Bus	GND
	5 V - Motor Driver Login Power (5)	Redline Bus	5V
	ENA (6)		D 8 (PWM)
	IN1 (7)		D 9 (PWM)
	IN2 (8)		D 10(PWM)

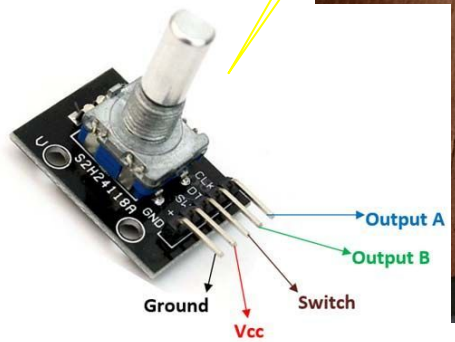


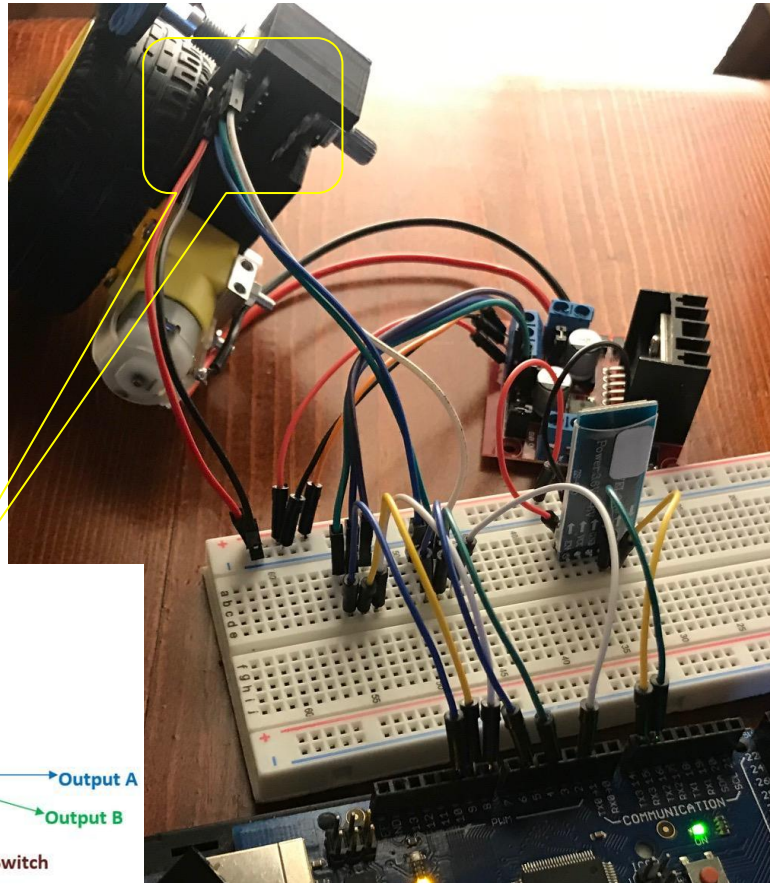


Arduino Wiring for DC Motor control

Rotary Encoder <--> Breadboard <--> Arduino

Rotary Encoder	Breadboard	Arduino
Output A (CLK)	White wire	D2 (PWM)
Output B (DT)	Green wire	D4 (PWM)
Switch (SW)	Blue wire	D6 (PWM)
VCC	Red wire	5V
GND	Black wire	GND

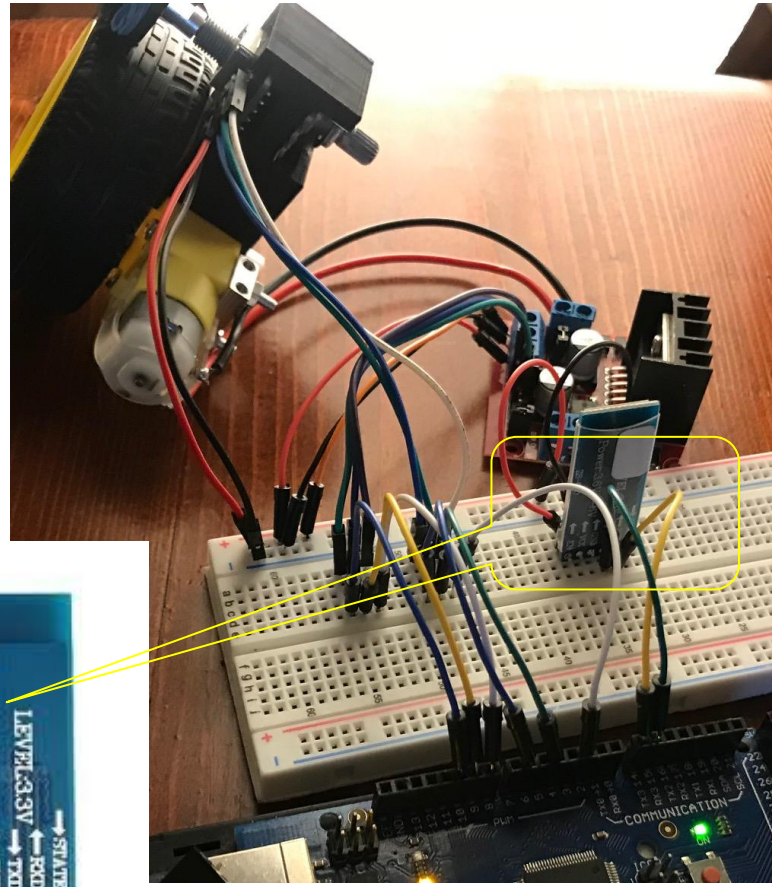




Arduino Wiring for DC Motor control

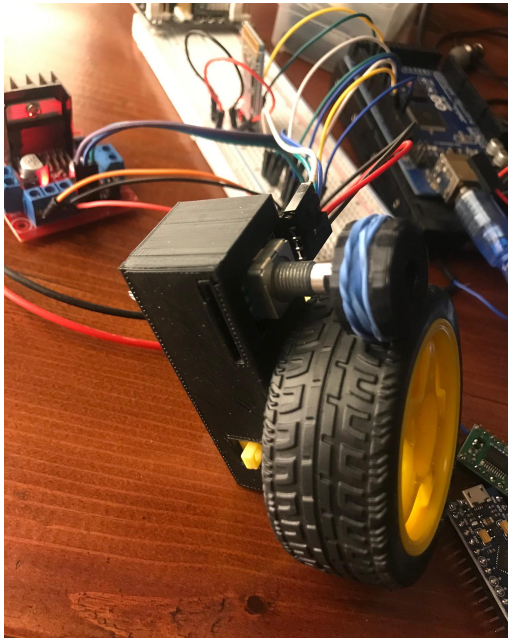
Bluetooth <--> Breadboard <--> Arduino

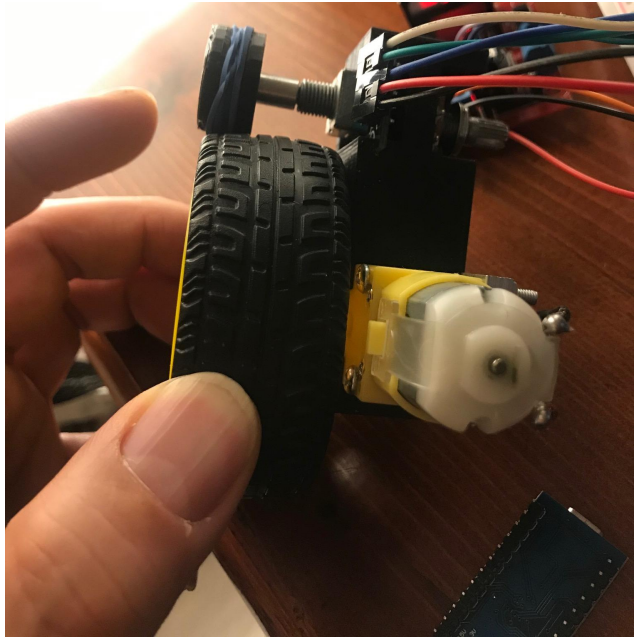
Bluetooth	Breadboard	Arduino
RX	White wire	TX3
TX	Green wire	RX3
VCC	Red wire	5V
GND	Black wire	GND



Arduino Wiring for DC Motor control

Encoder <--> Wheel





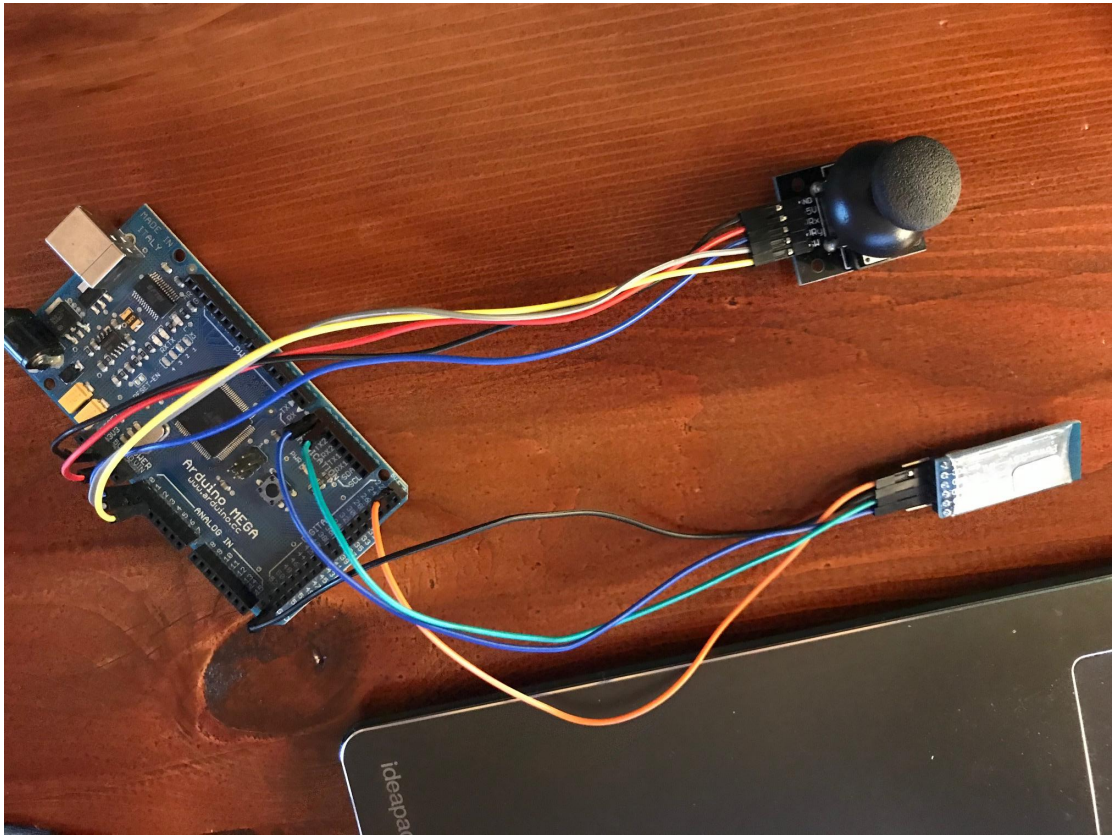
DC Motor Control Pin Map

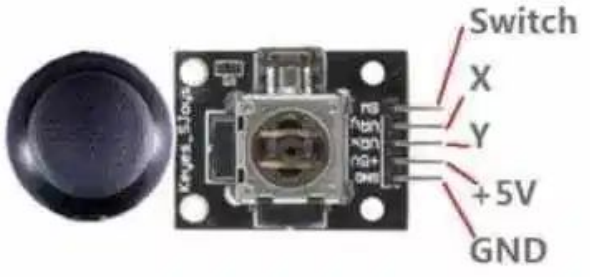
```
#define PIN_WHEEL_ENCODER_CLK_LEFT      2
#define PIN_WHEEL_ENCODER_DT_LEFT       4
#define PIN_WHEEL_ENCODER_SW_LEFT       6

#define PIN_MOTOR_PWM_LEFT  8
#define PIN_MOTOR_IN0_LEFT     9
#define PIN_MOTOR_IN1_LEFT     10

// Bluetooth is connected to Serial3 (RX3,TX3)
// Serial3.Begin (115200);
```

Arduino Wiring for Bluetooth Joystick Control





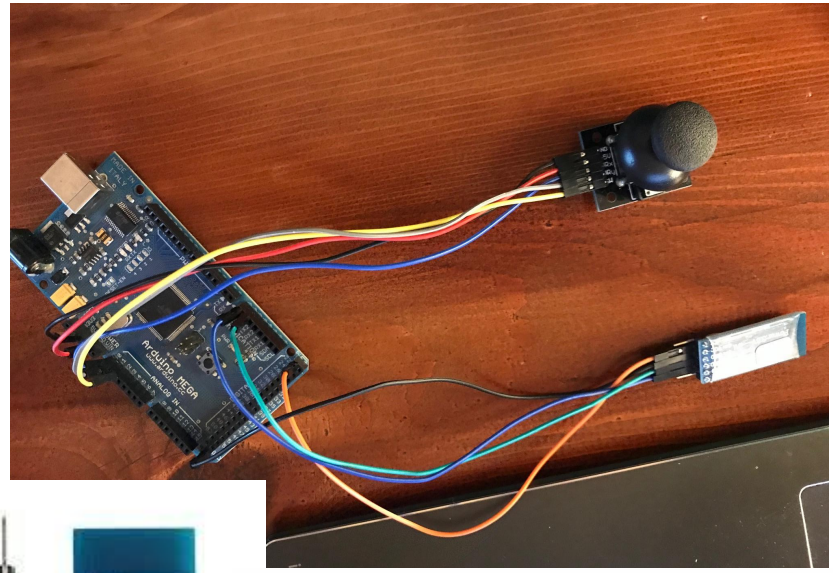
Arduino Wiring for Bluetooth Remote Control

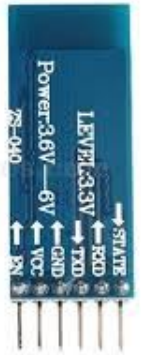
Bluetooth <--> Arduino

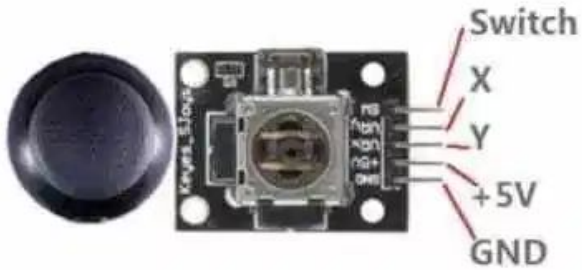
Bluetooth	Breadboard	Arduino
RX	White wire	TX3
TX	Green wire	RX3
VCC	Red wire	5V
GND	Black wire	GND

Joystick <--> Arduino

Joystick	Breadboard	Arduino
SW	Yellow wire	A2
X	Grey wire	A1
Y	Blue wire	A0
+5	Red wire	5V
GND	Black wire	GND







Bluetooth Joystick Controller Pin Map

```
#define PIN_JOY_X   A0
#define PIN_JOY_Y   A1
#define PIN_JOY_SW   A2
```

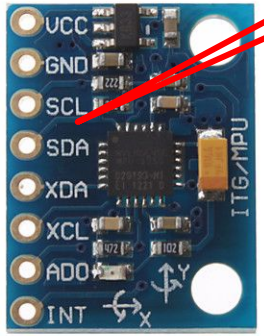
```
// Bluetooth is connected to Serial3 (RX3,TX3)
// Serial3.Begin (115200);
```

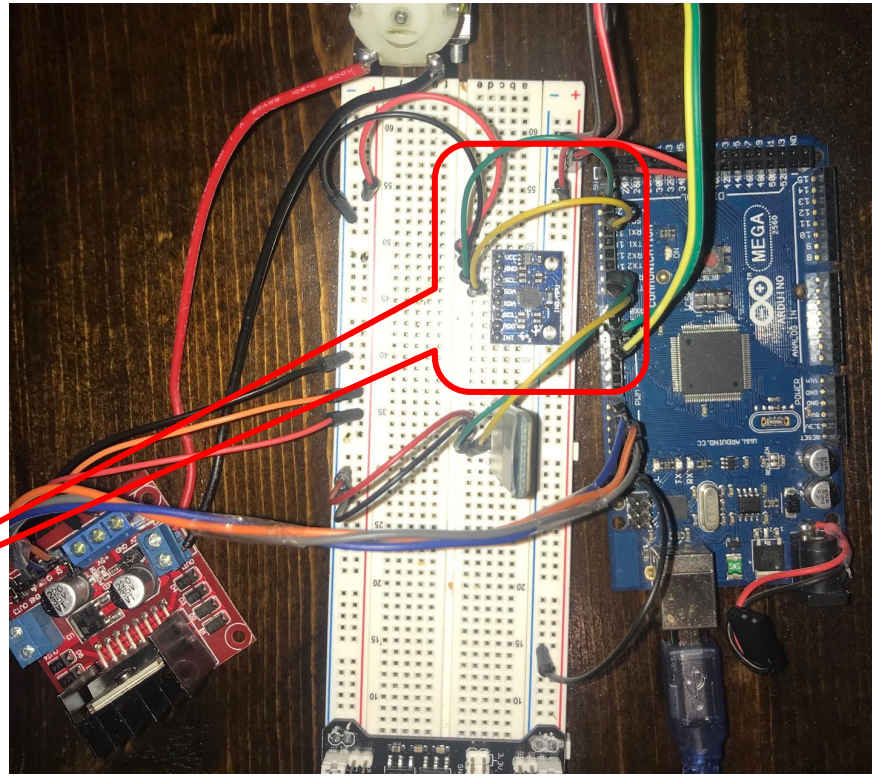
IMU Sensor

Arduino Wiring for IMU Sensor

IMU Sensor <--> Breadboard <--> Arduino

Bluetooth	Breadboard	Arduino
VCC	Red wire	5V
GND	Black wire	GND
SCL	Green wire	SCL
SDA	Yellow wire	SDA

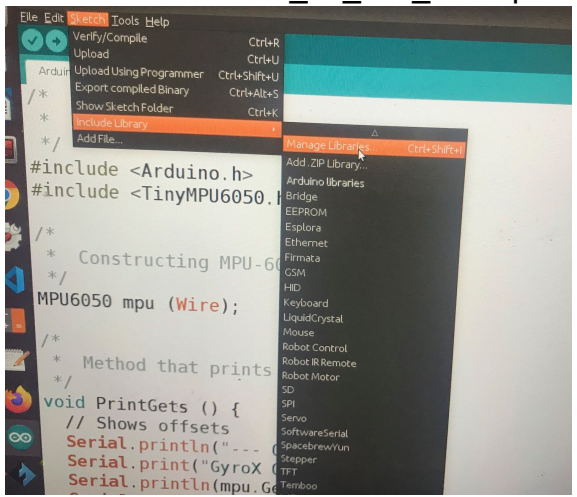


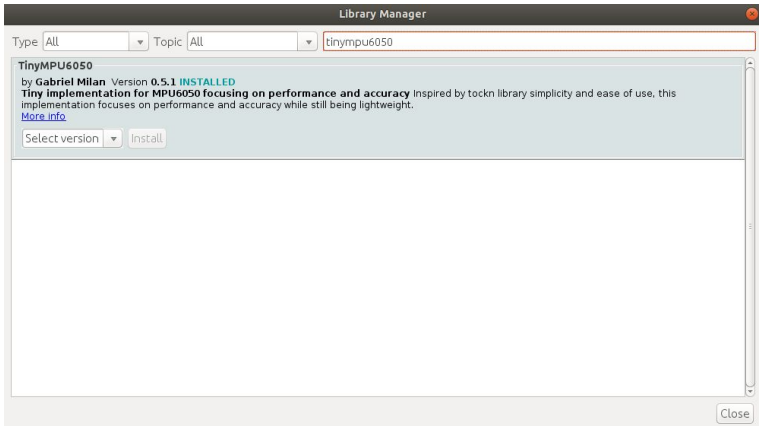


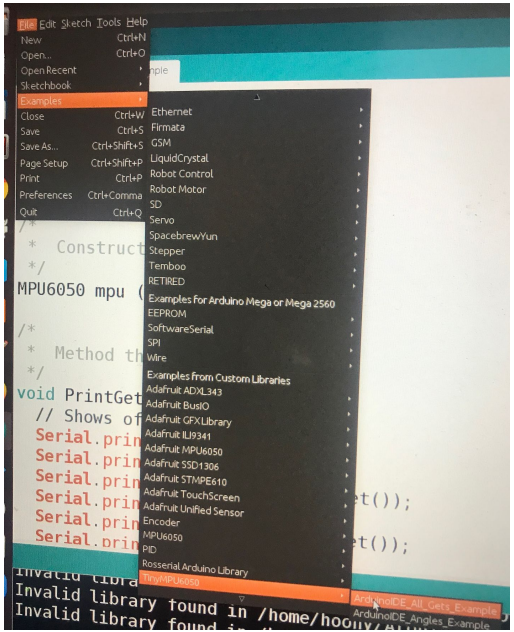
Arduino Wiring for IMU Sensor

TinyMPU6050 Library

1. Install Tiny MPU 6050 Library
   a. Manage Library
   b. Search “TinyMPU6050” and then install
2. Open example
   a. /File/Example/TinyMPU6050/ArduinoIDE_All_Get_Example







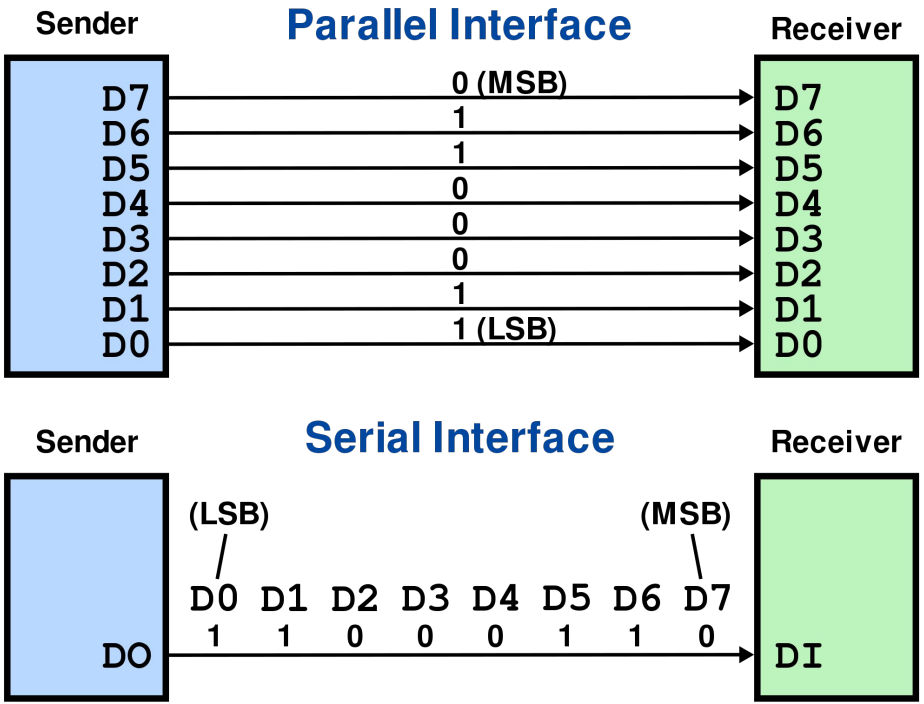
Communication

Internal & External Events for Embedded System Communication

- **Internal Event**
  - Program Condition, System Clock Event, ADC Event, etc.
    - Outgoing message
- **External Event**
  - Analog Input
    - Analog Sensor, Potentiometer
  - Digital Input
    - Button, Digital Sensor (In main program or Interrupt routine)
    - Message (Serial/ Parallel Communication)
      - Incoming message
    - Outgoing message

Serial/Parallel

Example transfers of `01100011`

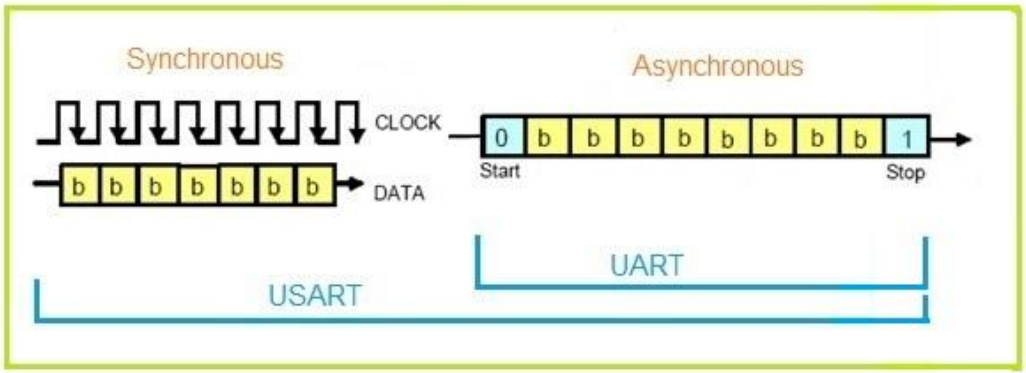


Communication Interfaces

- Universal Synchronous/Asynchronous Receiver Transmitter (USART - UART/USRT)
  - https://www.youtube.com/watch?v=MebhACqcdno
  -
- Inter-Integrated Circuit (I^2C)
  - https://www.youtube.com/watch?v=qeJN_80CiMU
  - https://www.youtube.com/watch?v=6IAkYpmA1DQ
- Serial Peripheral Interface (SPI)
  - https://www.youtube.com/watch?v=qyHaiDMf7p4

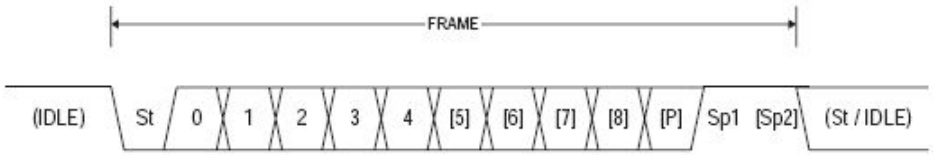
USART

- USART = Universal Synchronous/Asynchronous Receiver Transmitter
- The USART is used for synchronous and asynchronous serial communication.
- UART is commonly used at Arduino
- Asynchronous communication does not use a clock to validate data.
- Serial data is transferred one bit at a time.
- Asynchronous serial interfaces are cheap, easy to use, and until recently, very common.
- USB is well on its way to replace the serial comm ports on PCs.
- The USART communicates in a full-duplex mode (simultaneous tx, rx)



UART

- Serial frame format:

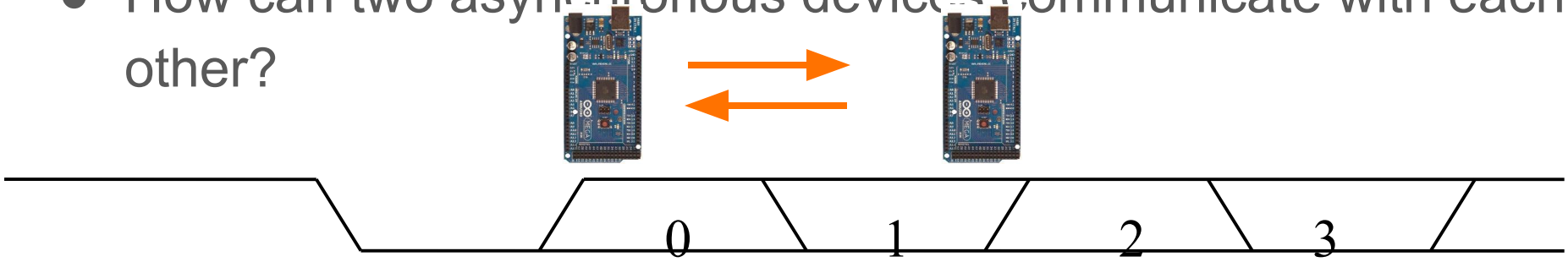


St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxD or TxD). An IDLE line must be high.

- Every frame will have at least
- -one start bit
- -some data bits (5,6,7,8 or 9)
- -one stop bit
- Parity is optional

UART

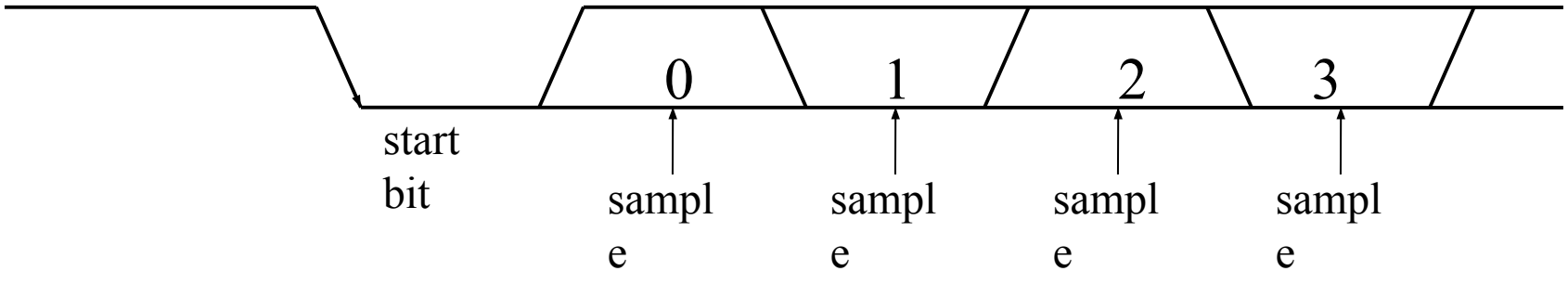
- How can two asynchronous devices communicate with each other?



- There is no known phase relationship between the two devices
- How can a receiving board “know” where the center of a bit frame is?
- How can it know where the start bit is?

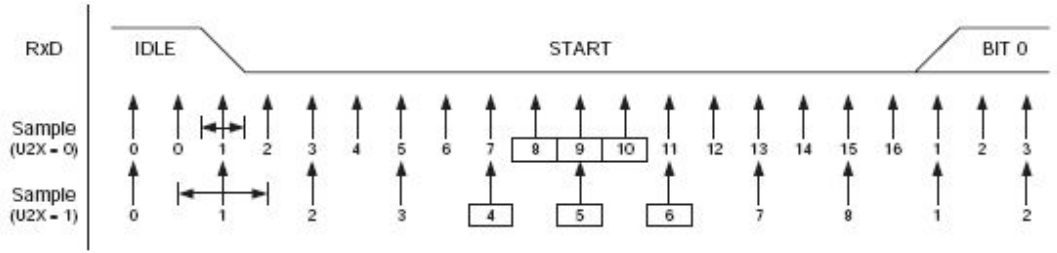
UART

- Need to know how fast the bits are coming (baud rate)
- Need to know where the start bit begins (its falling edge)
- Then we know when to sample



UART

- The USART uses a 16x internal clock to sample the start bit.



- The incoming data is continuously sampled until a falling edge is detected.
- Once detected, the receiver waits 6 clocks to begin sampling.
- One clock before the expected center of the start bit, 3 samples are taken.
- If 2 or 3 are detected as high, the search for another falling edge is started.
- If at least one sample is low, the start bit is validated. If so, begin sampling 16 clocks from center of start bit.
- Revalidate again with each data byte. Synchronization happens on each byte.

UART

- The USART internal clock is set by the UBRR register.

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

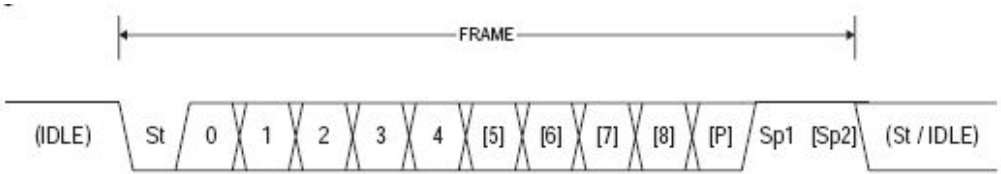
- The internal clock is derived from the internal CPU clock.
- Both transmitter and receiver use the same clock to operate from. The transmitter is synchronous to the clock. In coming data to the receiver is not.
- The baud rates are not exact power of 2 multiples of the CPU clock, thus baud rate inaccuracies occur at some settings. This is why you see some "funny" crystal oscillator frequencies.

Baud Rate (bps)	f_{osc} = 16.0000 MHz			
	U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	-0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	-0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	-0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	-3.5%	16	2.1%
230.4k	3	8.5%	8	-3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max (1)	1 Mbps		2 Mbps	

1. UBRR = 0, Error = 0.0%

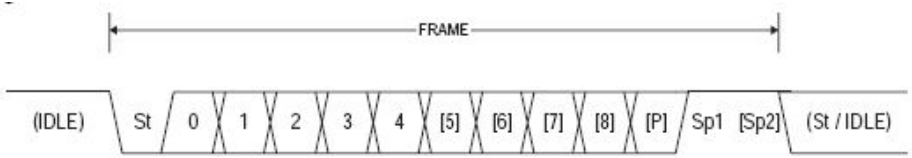
UART – Error Detection

- Error detection
  - Parity
    - created by an XOR of the data bits
    - parity can be even or odd
    - P_{even}= dn XOR dn-1 XOR dn-2 ..... XOR d0
    - P_{odd}= dn XOR dn-1 XOR dn-2 ..... XOR d0 XOR 1
- If we have a data byte (0b 0010 1101) and we want odd parity, the parity bit is set to a “one” to make a total of 9 bits which is an odd number of 1's.... i.e., odd parity is:



UART - Error Detection

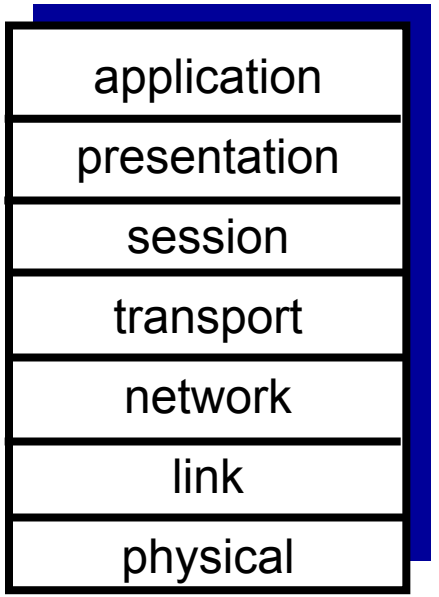
- **Frame error**
  - a frame error occurs when the receiver is expecting the stop bit and does not find it.
  - also known as a synchronization failure
  - the frame (byte) must be resent



- **Data overrun error**
  - data was not removed in the receive buffer before another byte came and overwrote it.

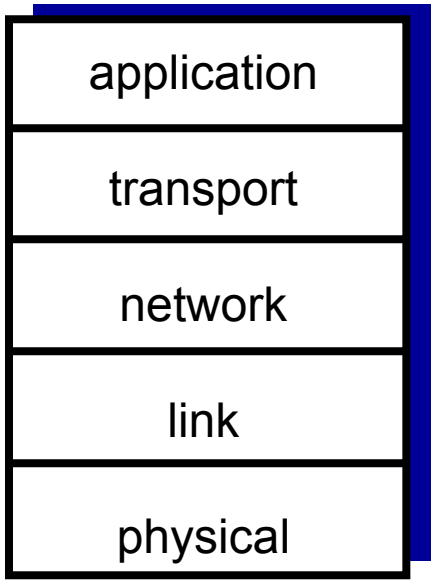
ISO/OSI reference model

- *presentation:* allow applications to interpret meaning of data, e.g., encryption, compression, machine-specific conventions
- *session:* synchronization, checkpointing, recovery of data exchange
- Internet stack "missing" these layers!
  - these services, *if needed,* must be implemented in application
  - needed?

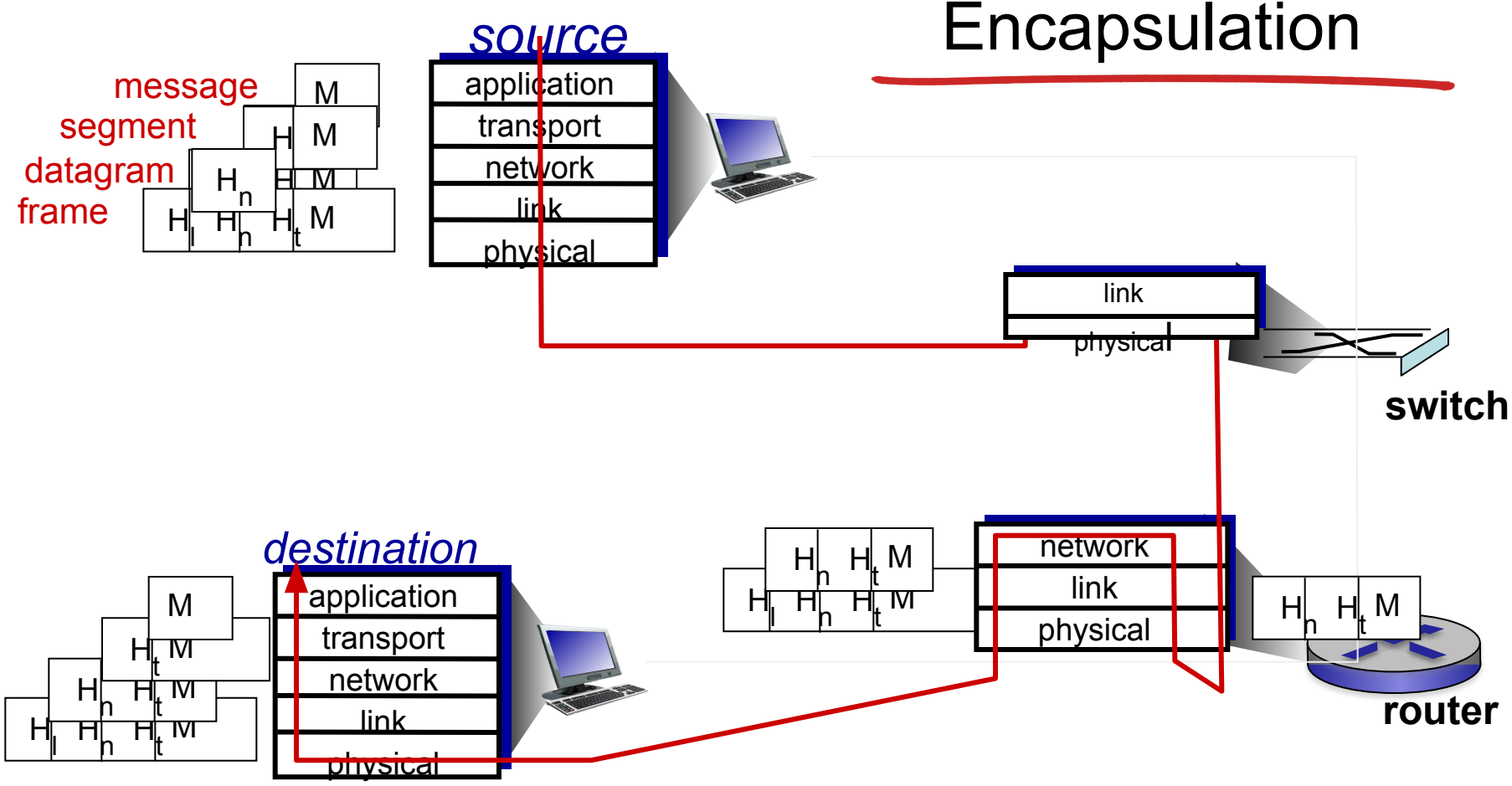


Internet protocol stack

- *application:* supporting network applications
  - FTP, SMTP, HTTP
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.111 (WiFi), PPP
- *physical:* bits “on the wire”



Encapsulation



Encoding & Decoding

- Encoding
  - One Bit, One Byte(one character), Bytes (Words)
  - Header/Payload/Footer
  - Tags
- Decoding
  - Parsing message
  - Interpreting

Protocol

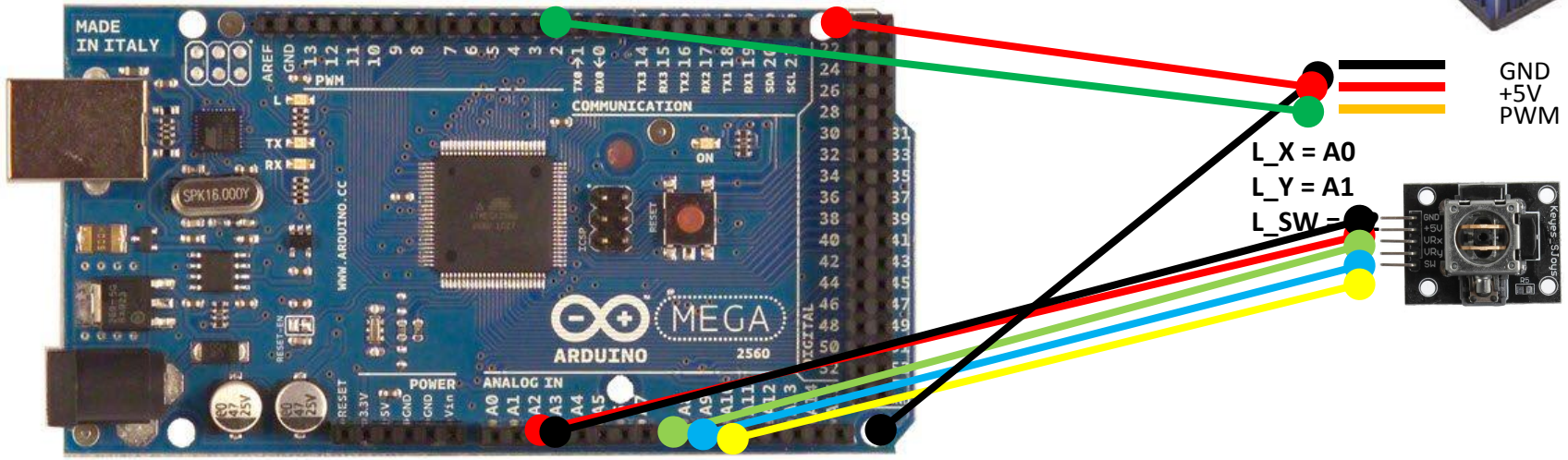
- **Set of rules to send and receive messages**
  - Serial/Parallel & Synchronous/Asynchronous
  - Signal Control
    - Message Unit [8 bit, 16 bit, 32 bit, 64 bit…]
    - Speed: baud[9600,19200,38400,57600,115200, …. 10GB…]
    - Flow control
  - Message Format
    - Start Message [Header]
    - Message Body [Payload]
    - End Message [Footer]
  - Byte Code / Plain Text
    - One bit, One byte, Bytes
    - Text Based Encoding

Protocol

- **Message Format**
  - Message structure (packet structure)
    - Start Message [Header]
    - Message Body [Payload]
    - End Message [Footer]
  - Message Start & End(Packet start & end)
  - Error Detect/Correction
    - CRC, Checksum, etc.
  - Payload Format
- **Let’s see an example protocol (MultiWii protocol)**

Servo Motor Control

- #include <Servo.h>
  - Servo myservo; // create servo object to control a servo
  - myservo.attach(9); // attaches the servo on pin 9 to the servo object
  - myservo.write(val) ; // val is the angle of the servo head



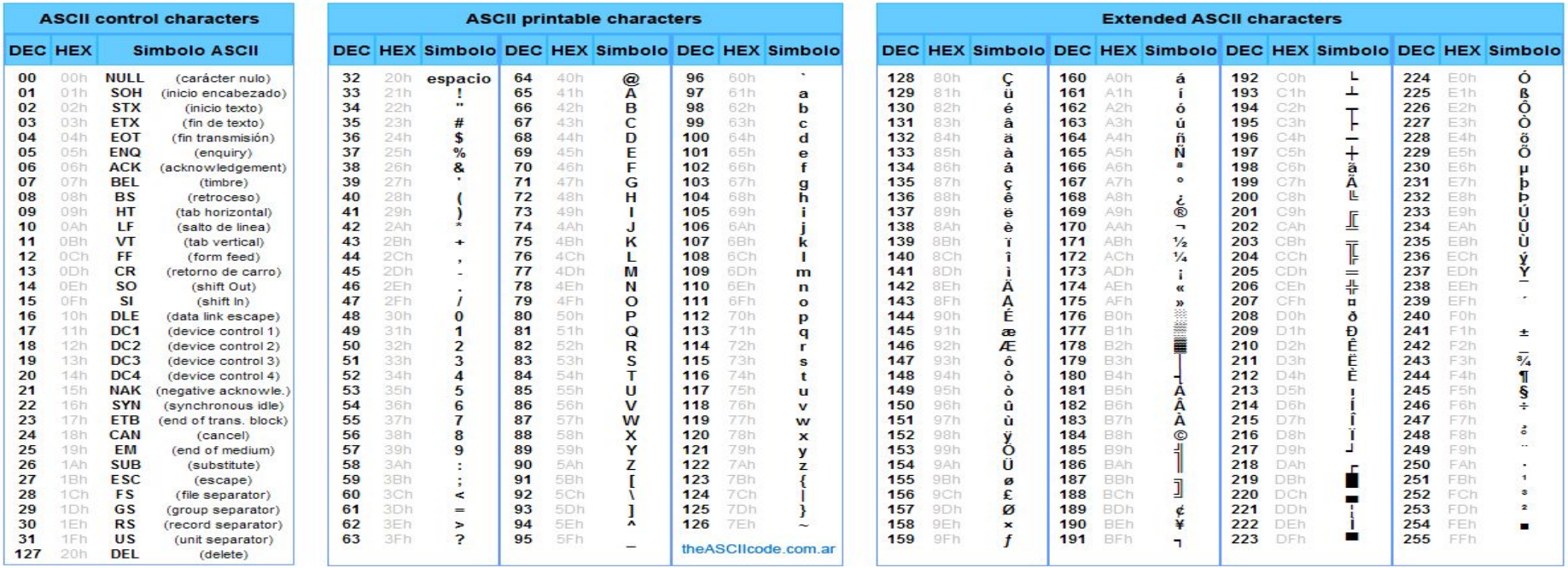
Serial Communication (UART)

- Serial Port Setting
  - RX, TX pin are connected to FTDI chip for usb serial
  - Serial.begin(baud[9600,19200,38400,57600,115200])
  - char Serial.read(), void Serial.print(“string”), void Serial.write(byte);
- Let’s read x Joystick value and send it over Serial
  - X value > 800 □ send ‘i’, X value < 200 □ send ‘p’, otherwise send ‘o’,
- Let’s make
  - Simple message interpreter
  - ~~a => left turn , s => right turn, w => forward, z => backward, b => stop~~
  - i □ servo(50), o □ servo(100), p □ servo(150),
- Let’s make a simple protocol to sending values over Serial
  - Message parser design
  - Message delimiter (0x0D), parameter delimiter (0x20) space
  - Joystick message (x -> servo position, y -> LED brightness)

USE ASCII code

ASCII code

- American Standard Code for Information Interchange (ASCII)



ASCII control characters

DEC	HEX	Símbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de linea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

ASCII printable characters

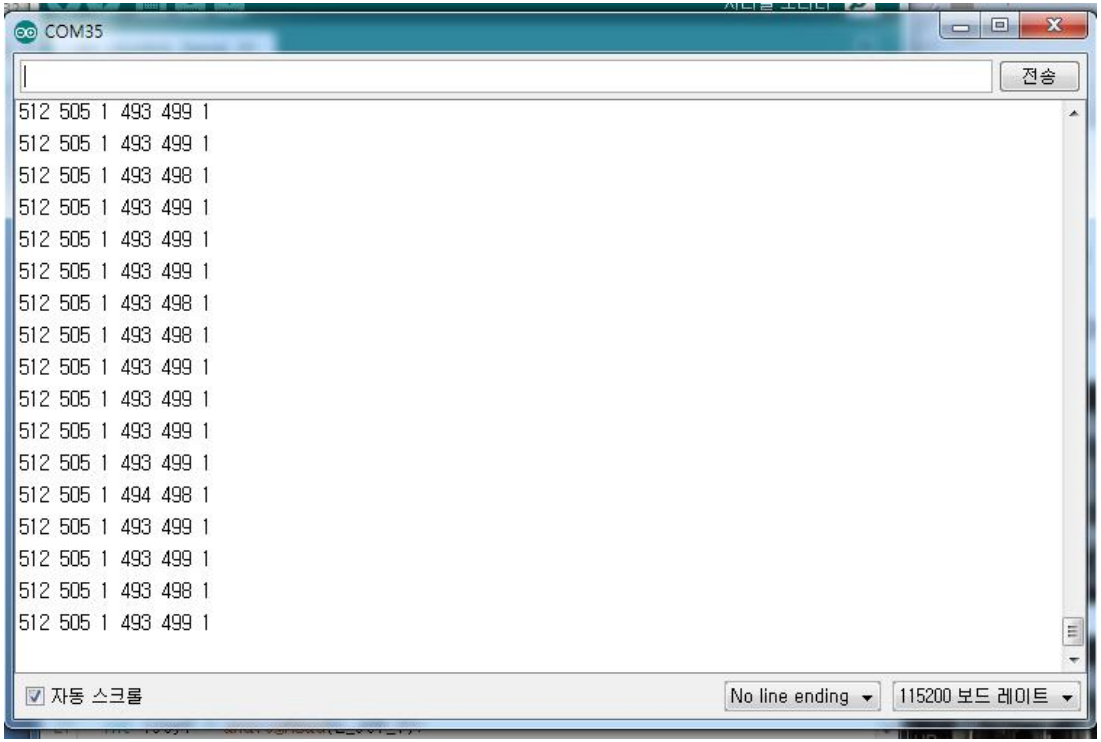
DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h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

theASCIIcode.com.ar

Extended ASCII characters

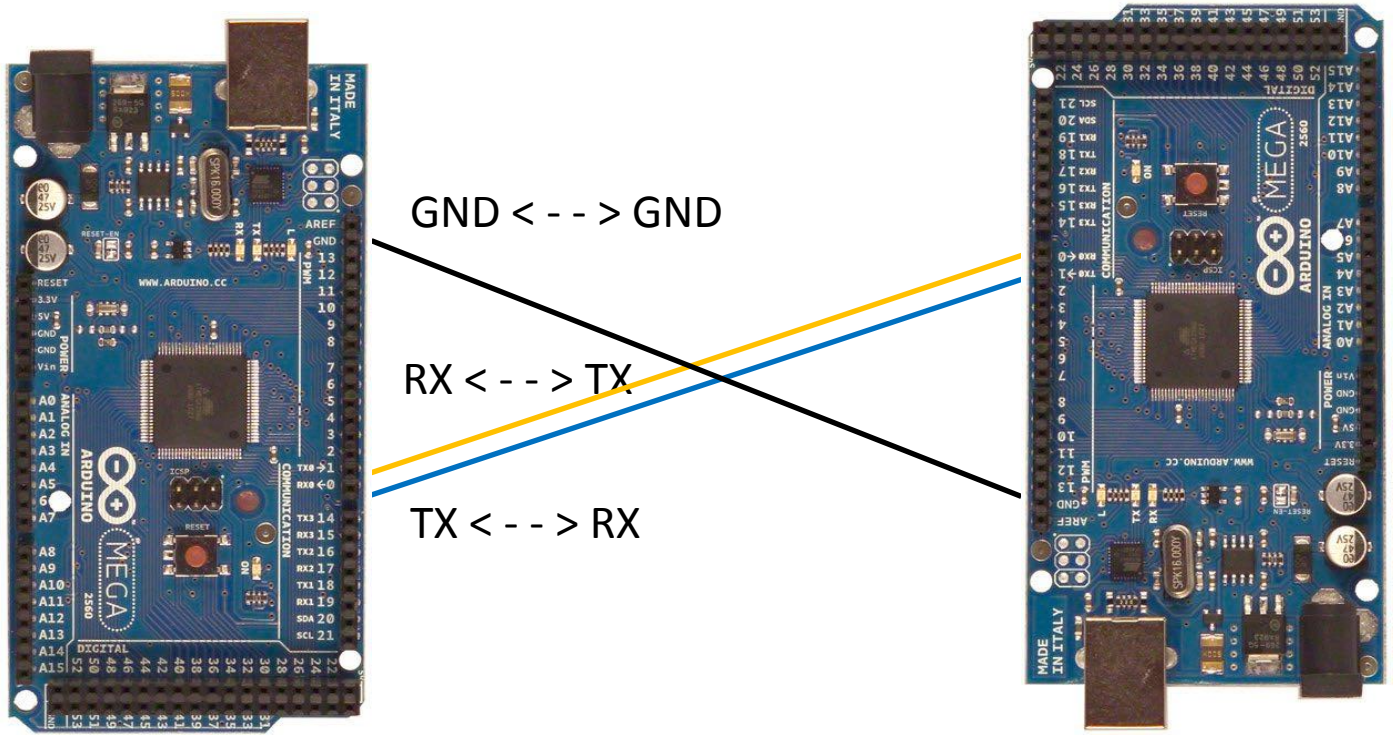
DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	å	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	░	208	D0h	ð	240	F0h	≡
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ò	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99h	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	█	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	¦	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	Ì	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	

(UART)


COM35
전송
512 505 1 493 499 1
512 505 1 493 499 1
512 505 1 493 498 1
512 505 1 493 499 1
512 505 1 493 499 1
512 505 1 493 499 1
512 505 1 493 498 1
512 505 1 493 498 1
512 505 1 493 499 1
512 505 1 493 499 1
512 505 1 493 499 1
512 505 1 493 499 1
512 505 1 494 498 1
512 505 1 493 499 1
512 505 1 493 499 1
512 505 1 493 498 1
512 505 1 493 499 1
자동 스크롤
No line ending
115200 보드 레이트

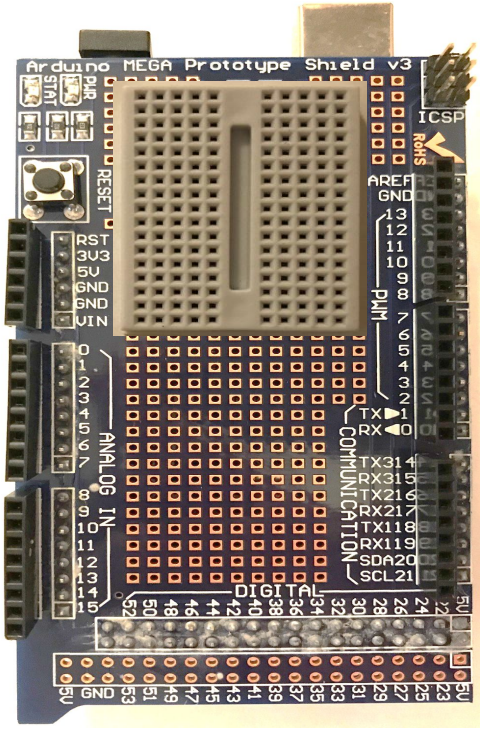
Serial to Serial

» Let’s make a wired remote controller using cross cabling between two arduino boards



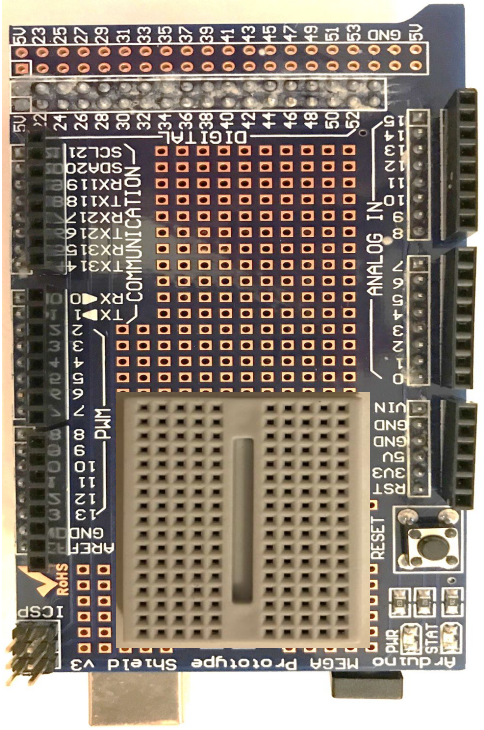
Serial to Serial over Wireless

» Let’s make a wired remote controller using cross cabling between two arduino boards



Slave

Master



Serial to Serial over Internet

» Let's make a remote control servo motor which can be controlled through internet

Server
IP: XXX.XXX.XXX.XXX:8080

Client
IP: XXX.XXX.XXX.XXX

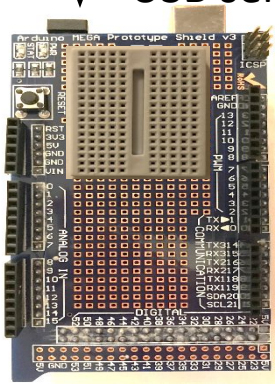
Serial Forwarder over TCP

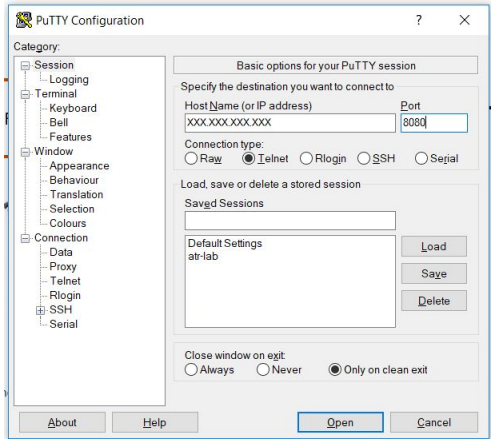
Internet

Putty- Telnet

USB serial (COM x)

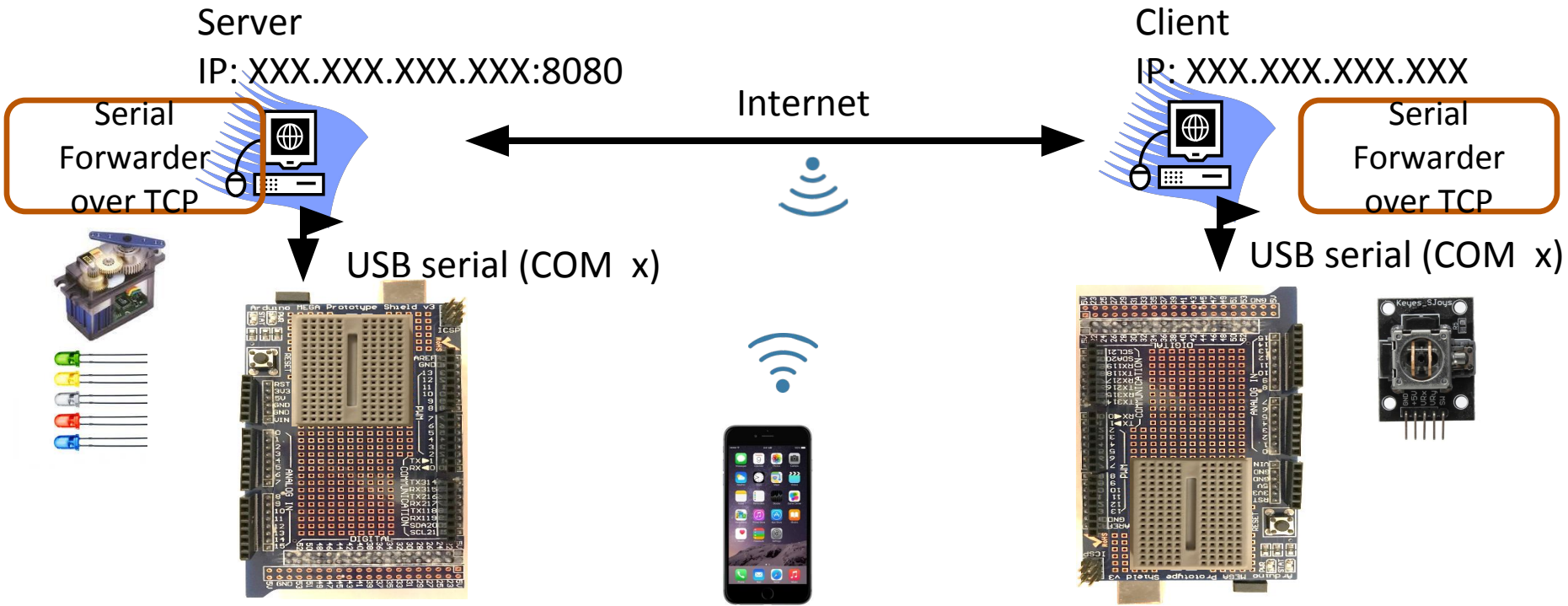
Photo-resistor





Serial to Serial over Internet

» Let's make a remote control servo motor which can be controlled through internet



Server
IP: XXX.XXX.XXX.XXX:8080

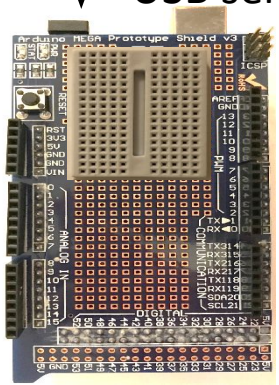
Client
IP: XXX.XXX.XXX.XXX

Internet

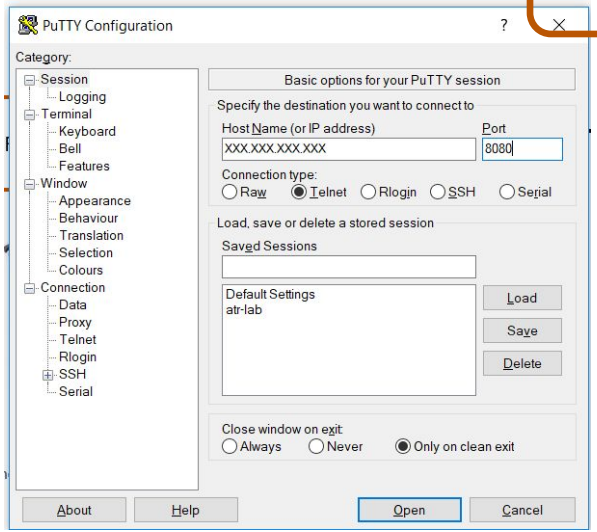
Serial Forwarder over TCP

USB serial (COM x)

Photo-resistor



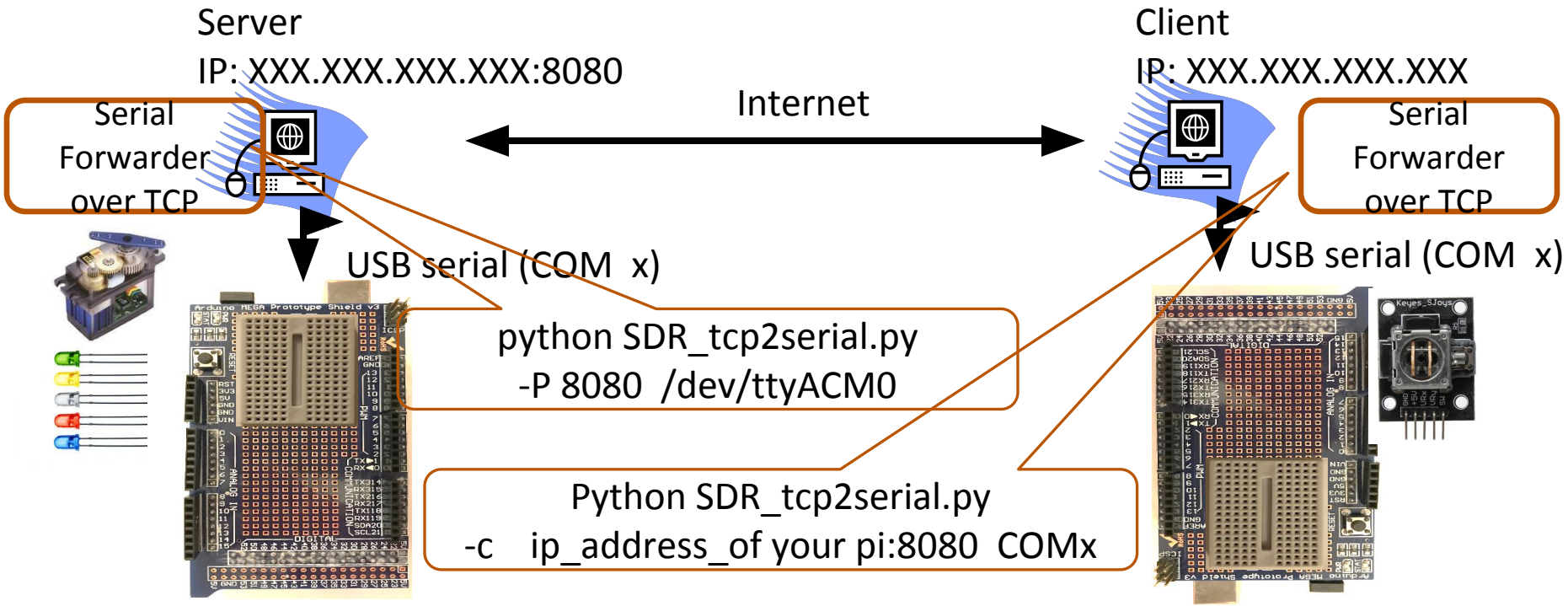
Putty- Telnet



TCP over Serial

Serial to Serial over Internet

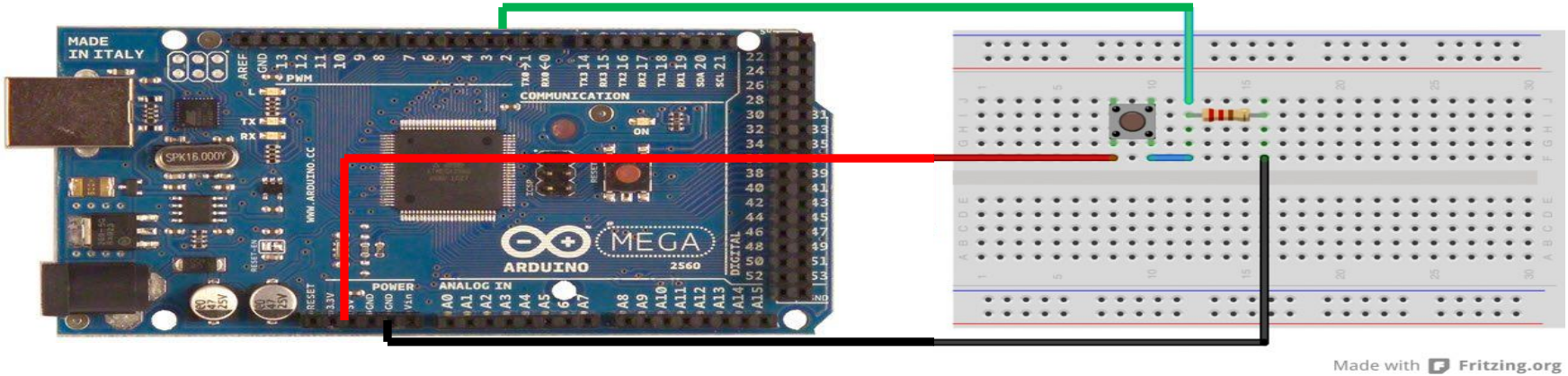
» Let's make a remote control servo motor which can be controlled through internet



TCP over Serial

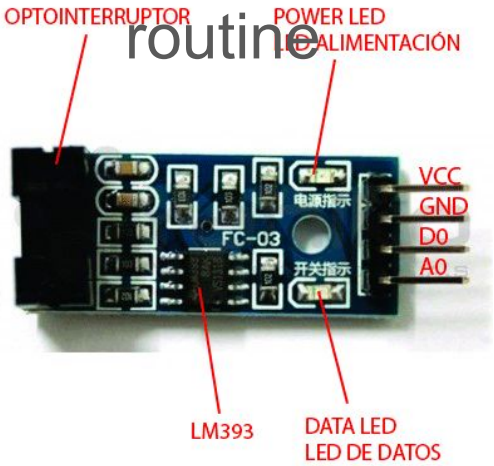
Push Button Counter using interrupt

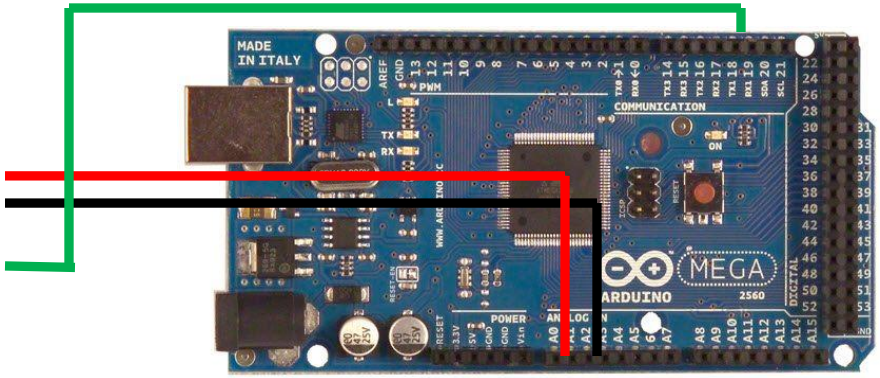
- Mega 2560 : 2, 3, 18, 19, 20, 21
- attachInterrupt()
  - no delay() and no millis() works.
- Let’s make a push counter with an interrupt routine



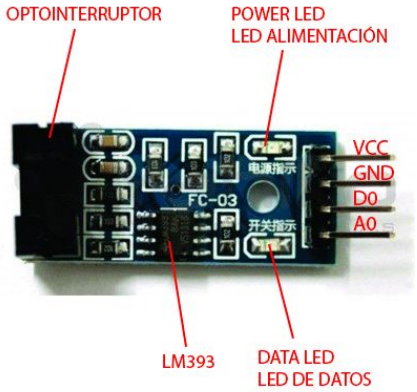
Optocoupler Counter using interrupt

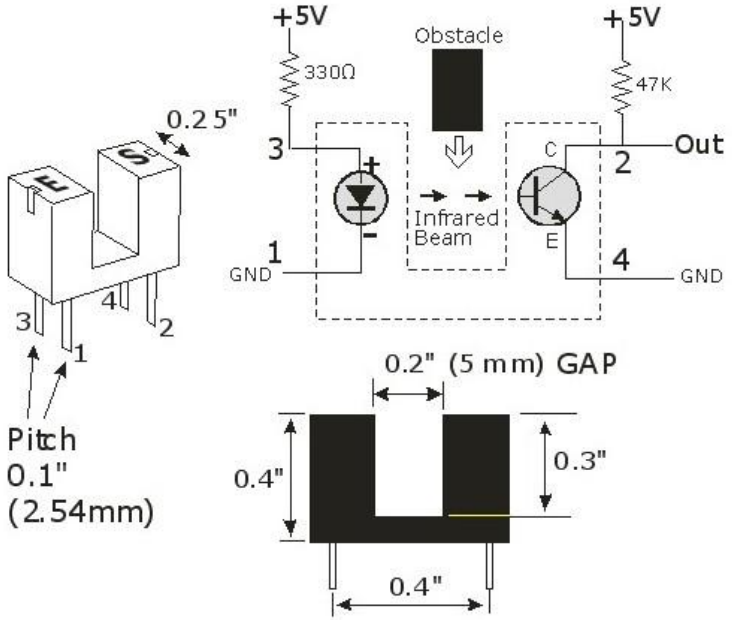
- Mega 2560 : 2, 3, 18, 19, 20, 21
- attachInterrupt()
- Let’s make a optercoupler counter with an interrupt routine

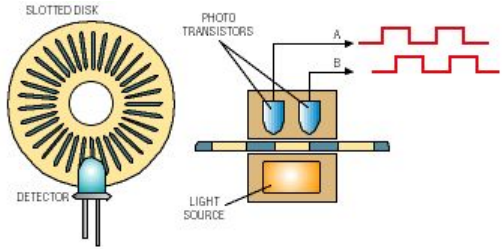




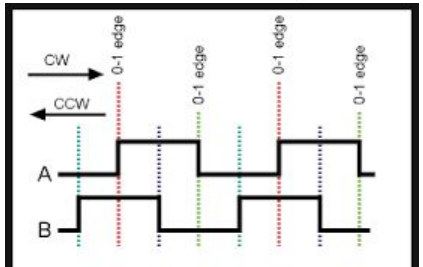
Optocoupler Counter



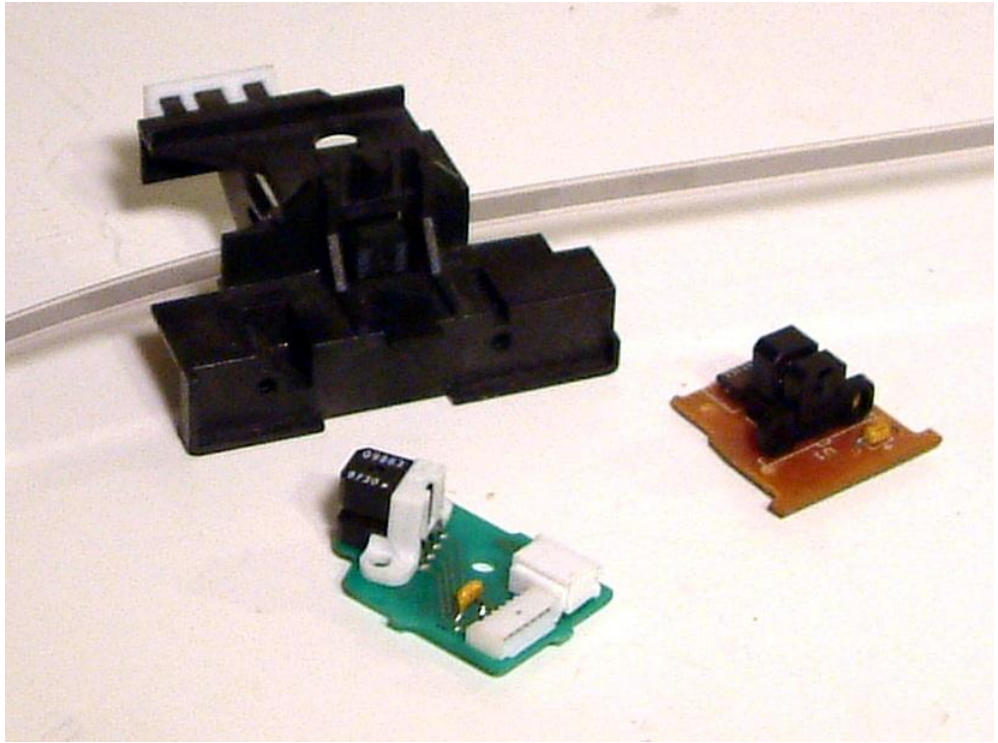




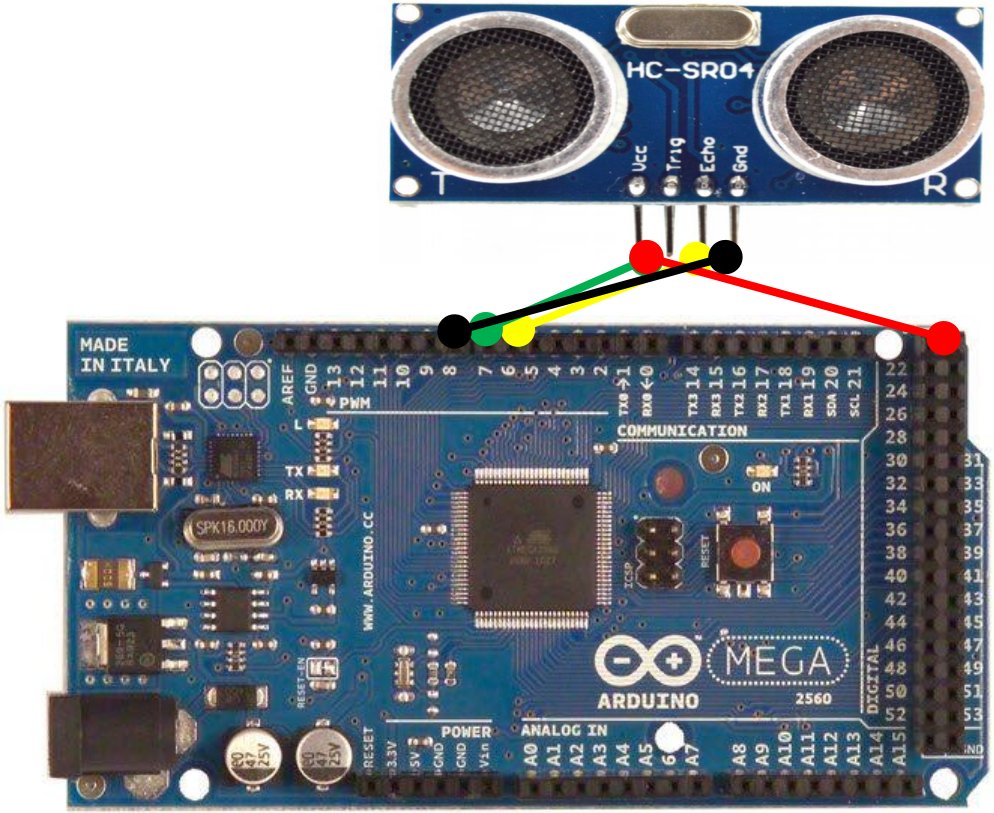
Two bit counter



Optocoupler Counter



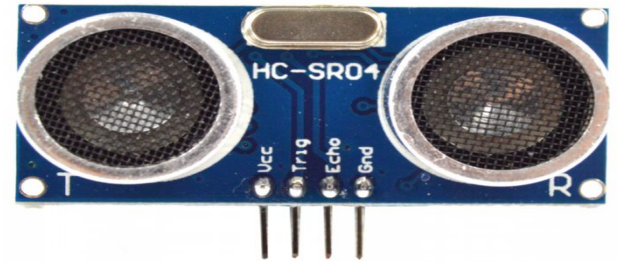
Ultrasonic Sensor



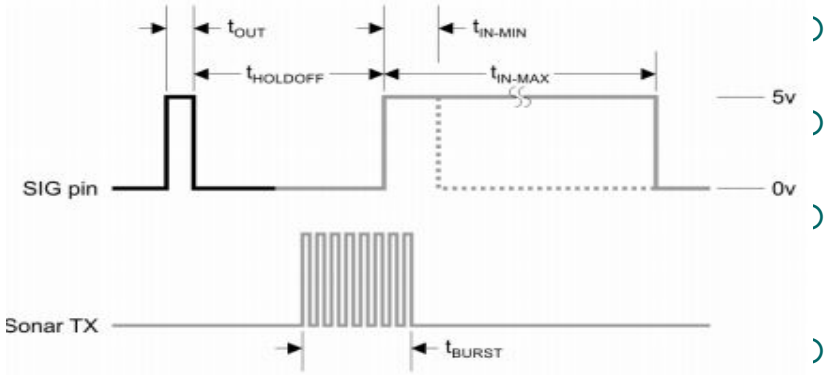
VCC = +5
Trig = 12
Echo = 11
GND = GND

Ultrasonic Distance Sensor

- PING ultrasonic distance sensor provides precise distance measurements from about 2 cm (0.8 inches) to 3 meters (3.3 yards).
- It works by transmitting an ultrasonic burst and providing an output pulse that corresponds to the time required for the burst echo to return to the sensor.
- By measuring the echo pulse width the distance to target can easily be calculated

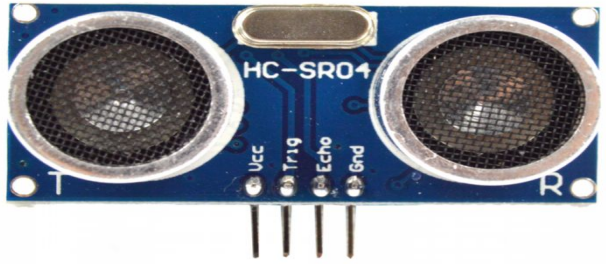


Theory of Operation

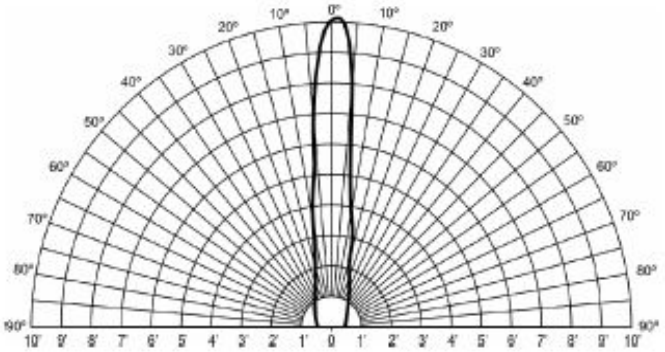


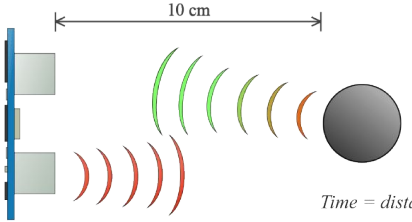
t_{OUT}	2 uS (min), 5 uS typical
$t_{HOLDOFF}$	750 uS
t_{BURST}	200 uS @ 40 kHz
t_{IN-MIN}	115 uS
t_{IN-MAX}	18.5 mS

- The PING sensor emits a short ultrasonic burst and then "listens" for the echo.
- Under control of a host microcontroller (trigger pulse), the sensor emits a short 40 kHz (ultrasonic) burst.
- This burst travels through the air at about 1130 feet per second, hits an object and then bounces back to the sensor.
- The PING sensor provides an output pulse to the host that will terminate when the echo is detected, hence the width of this pulse corresponds to the distance to the target.



Limited Detection Range





speed of sound:

v = 340 m/s

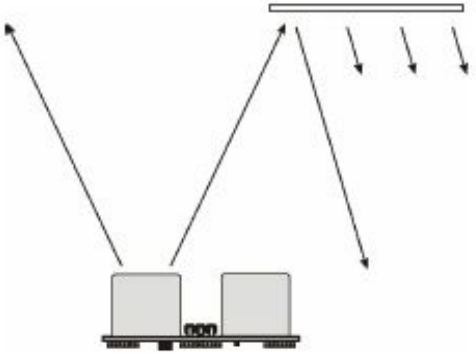
v = 0,034 cm/µs

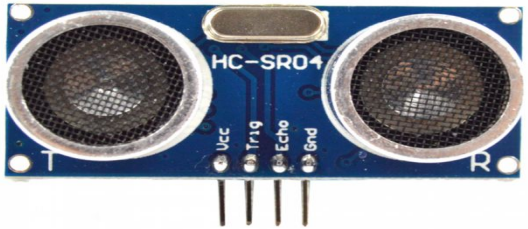
Time = distance / speed:

t = s / v = 10 / 0,034 = 294 µs

Distance:

s = t · 0,034 / 2





Let’s make a Ping Sonar #0

```
const int trigPin = 12;
const int echoPin = 11;
// defines variables
long duration;
int distance;
void setup() {
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
  Serial.begin(9600); // Starts the serial communication
}

void loop() {
  digitalWrite(trigPin, LOW); // Clears the trigPin
  delayMicroseconds(2);   // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);   // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);   // Calculating the distance
  distance= duration*0.034/2;   // Prints the distance on the Serial Monitor
  Serial.print("Distance: ");
  Serial.println(distance);
}
```