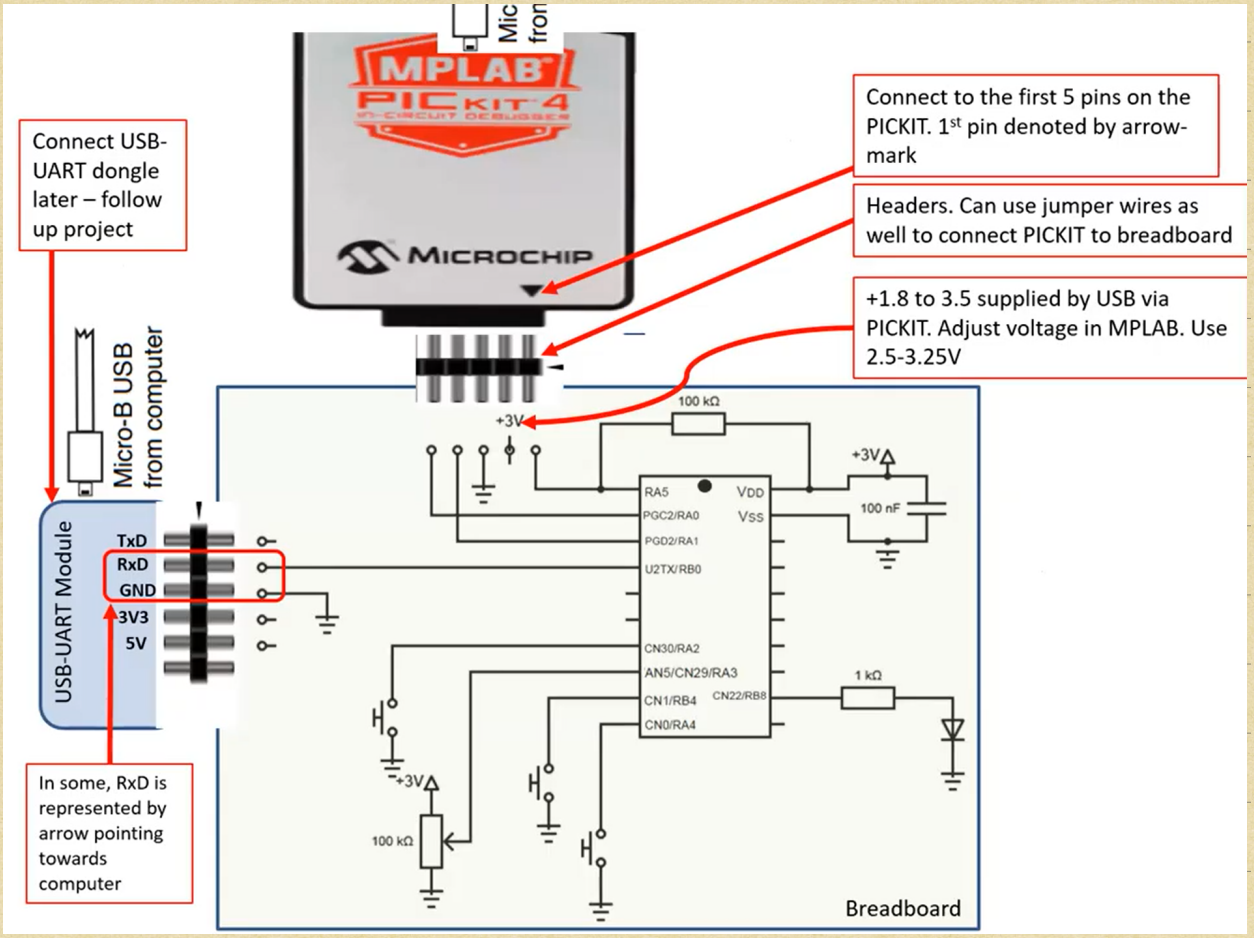
IO Change Notification (CN)

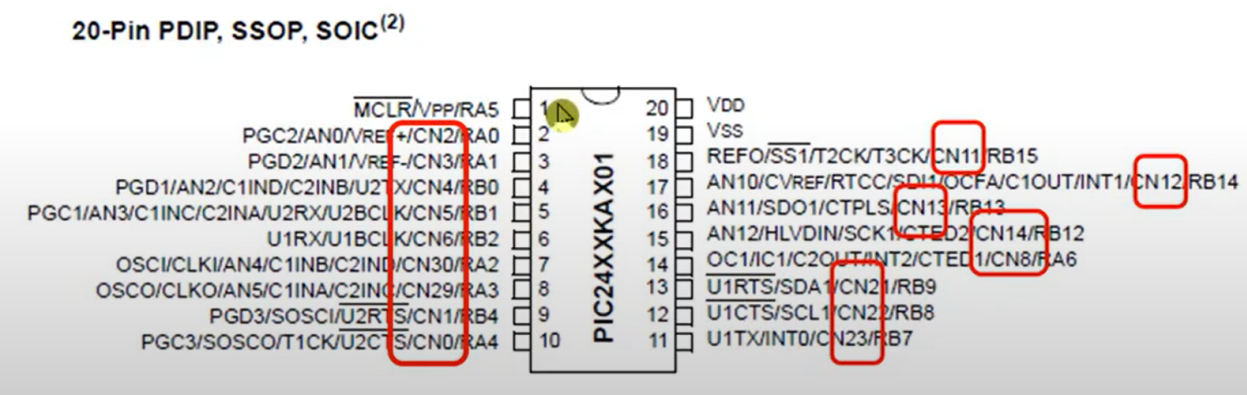
IO Change Interrupts

- Commonly used to trigger processors awake due to external stimuli.
- Human induced → push buttons and power buttons on iPhones and android Phones.
- Signal induced: Low to High or High to Low on a signal Bus e.g. Infra red receiver in TV, wireless, wired serial communication buses.



IO Change Notification

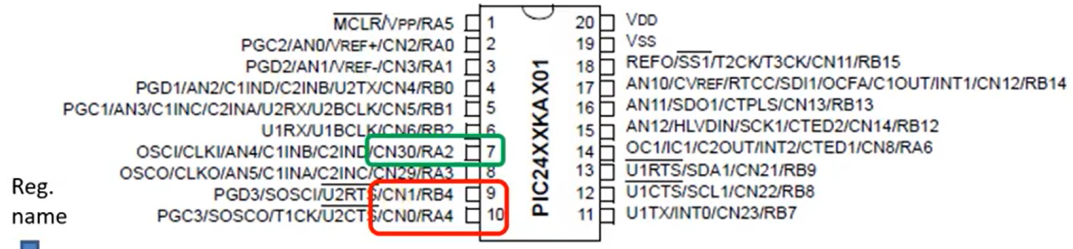
- Used to detect Change in IO State
- Can be used to trigger interrupt on input pins labelled CNx due to LO-Hi or Hi-Lo transition.



Steps:

- Configure IO pins as Input: TrisxBITS.TRISx# = 1;
- Configure -11- to be either pulled up or low using internal resistors in PIC.
- Configure Change on pin Interrupt
  - Set interrupt priority
  - Clear interrupt flag
  - Enable interrupt
  - Note: One interrupt common to all IO changes configured.
- In interrupt routine check the PORTs to determine which IO pin had a change of notification.

20-Pin PDIP, SSOP, SOIC(2)

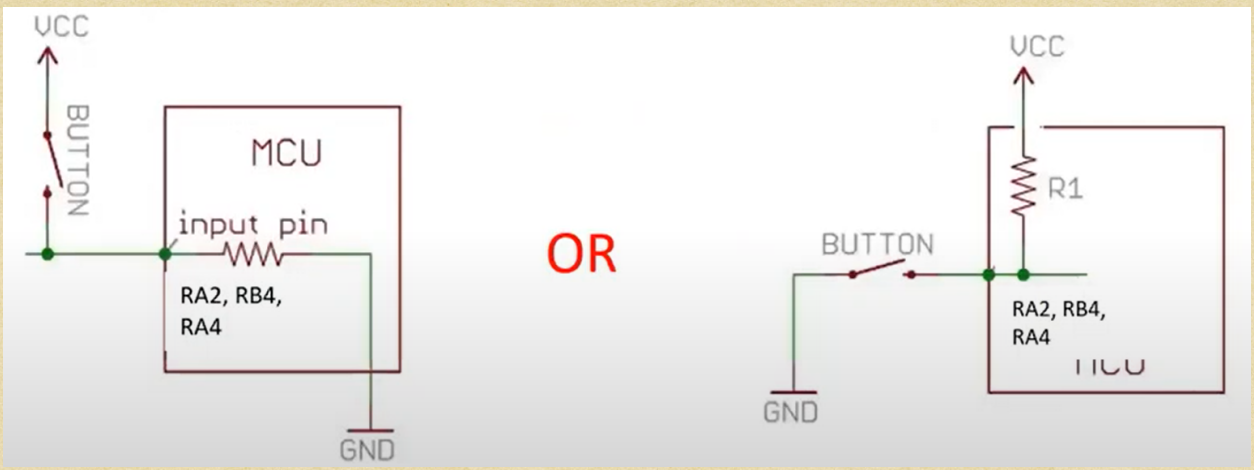


Reg. name

TABLE 4-4: ICN REGISTER MAP

File Name	Addr	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
CNEN1	0060	CN15IE(1)	CN14IE	CN13IE	CN12IE	CN11IE	—	CN9IE	CN8IE	CN7IE(1)	CN6IE	CN5IE	CN4IE	CN3IE	CN2IE	CN1IE	CN0IE	0000
CNEN2	0062	—	CN30IE	CN29IE	—	CN27IE(1)	—	—	CN24IE(1)	CN23IE	CN22IE	CN21IE	—	—	—	—	CN16IE(1)	0000
CNPU1	0068	CN15PUE(1)	CN14PUE	CN13PUE	CN12PUE	CN11PUE	—	CN9PUE	CN8PUE	CN7PUE(1)	CN6PUE	CN5PUE	CN4PUE	CN3PUE	CN2PUE	CN1PUE	CN0PUE	0000
CNPU2	006A	—	CN30PUE	CN29PUE	—	CN27PUE(1)	—	—	CN24PUE(1)	CN23PUE	CN22PUE	CN21PUE	—	—	—	—	CN16PUE(1)	0000
CNPD1	0070	CN15PDE(1)	CN14PDE	CN13PDE	CN12PDE	CN11PDE	—	CN9PDE	CN8PDE	CN7PDE(1)	CN6PDE	CN5PDE	CN4PDE	CN3PDE	CN2PDE	CN1PDE	CN0PDE	0000
CNPD2	0072	—	CN30PDE	CN29PDE	—	CN27PDE(1)	—	—	CN24PDE(1)	CN23PDE	CN22PDE	CN21PDE	—	—	—	—	CN16PDE(1)	0000

- Step 1: Set RB4 and RA4 as inputs
- Step2: Configure internal pull-up OR internal Pull down on RB4/CN1 pin & CN0/RA4



// Pull down enabled on CNx

CNPDxbits.CNxPDE = 1

// Pull up enabled on CNx

CNPUxbits.CNxPUE = 1;

Step3 : Configure CN Interrupt

REGISTER 8-17: IPC4: INTERRUPT PRIORITY CONTROL REGISTER 4

U-0	R/W-1	R/W-0	R/W-0	U-0	R/W-1	R/W-0	R/W-0
—	CNIP2	CNIP1	CNIP0	—	CMIP2	CMIP1	CMIP0
bit 15							bit 8

U-0	R/W-1	R/W-0	R/W-0	U-0	R/W-1	R/W-0	R/W-0
—	MI2C1P2	MI2C1P1	MI2C1P0	—	SI2C1P2	SI2C1P1	SI2C1P0
bit 7							bit 0

bit 14-12 **CNIP<2:0>:** Input Change Notification Interrupt Priority bits
111 = Interrupt is priority 7 (highest priority interrupt)
•
•
•
001 = Interrupt is priority 1
000 = Interrupt source is disabled

REGISTER 8-10: IEC1: INTERRUPT ENABLE CONTROL REGISTER 1

R/W-0	R/W-0	R/W-0	U-0	U-0	U-0	U-0	U-0
U2TXIE	U2RXIE	INT2IE	—	—	—	—	—
bit 15							bit 8

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	INT1IE	CNIE	CMIE	MI2C1IE	SI2C1IE
bit 7							bit 0

bit 3 **CNIE:** Input Change Notification Interrupt Enable bit
1 = Interrupt request enabled
0 = Interrupt request not enabled

REGISTER 8-6: IFS1: INTERRUPT FLAG STATUS REGISTER 1

R/W-0, HS	R/W-0, HS	R/W-0, HS	U-0	U-0	U-0	U-0	U-0
U2TXIF	U2RXIF	INT2IF	—	—	—	—	—
bit 15							bit 8

U-0	U-0	U-0	R/W-0, HS	R/W-0, HS	R/W-0, HS	R/W-0	R/W-0
—	—	—	INT1IF	CNIF	CMIF	MI2C1IF	SI2C1IF
bit 7							bit 0

bit 3 **CNIF:** Input Change Notification Interrupt Flag Status bit
1 = Interrupt request has occurred
0 = Interrupt request has not occurred

```
///// Change of pin Interrupt subroutine
void __attribute__((interrupt, no_auto_psv)) _CNInterrupt(void)
{
                    if(IFS1bits.CNIF == 1)
  {
    if(PORTBbits.RB4==0)
    {
      CN1flag = 1;  // user defined global variable used as flag
    }
    else if (PORTAbits.RA4 == 0)
    {
      CN0flag = 1; // user defined global variable used as flag
     }
  }
                    IFS1bits.CNIF = 0;                          // clear IF flag
                    Nop(); //No operation function
}
```

Aside:

use LATBbits.LATB4=1; to write to a pin.

if (PORTBbits.RB4=1) to read a pin

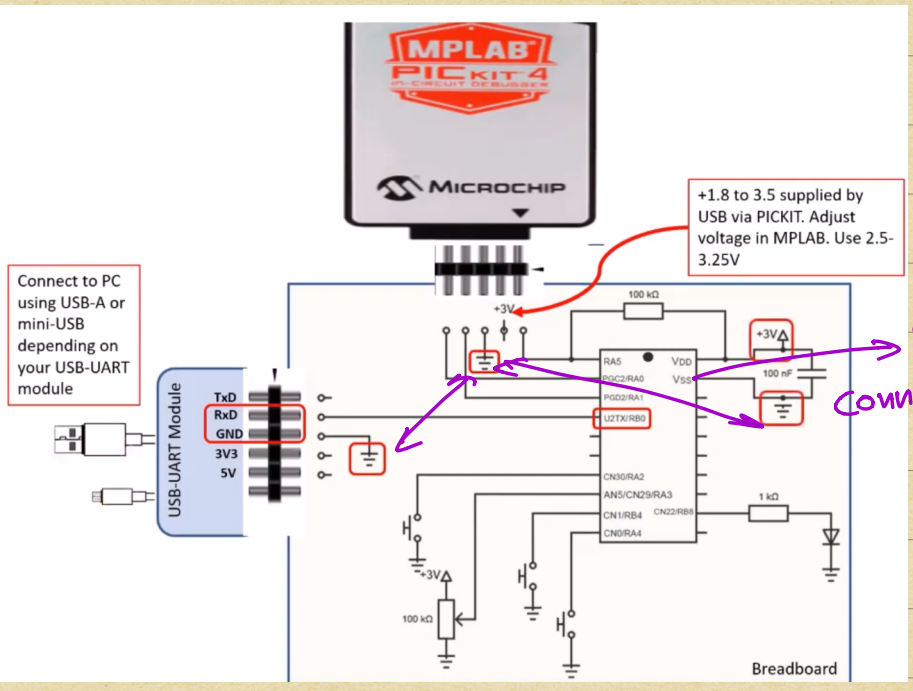
There is a common interrupt routine for all CN inputs.

Interrupts get triggered for any change in state hi-lo or lo-hi and also for debounces on push buttons.

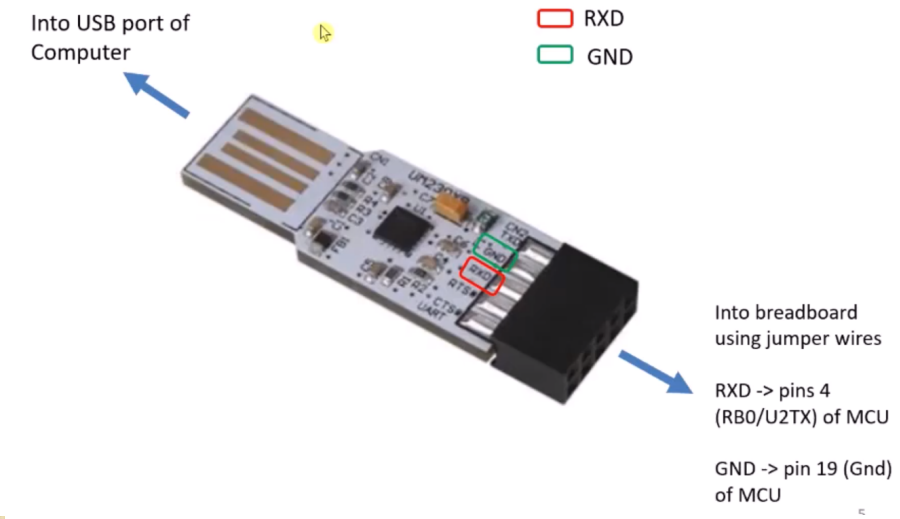
The code must filter debounces.

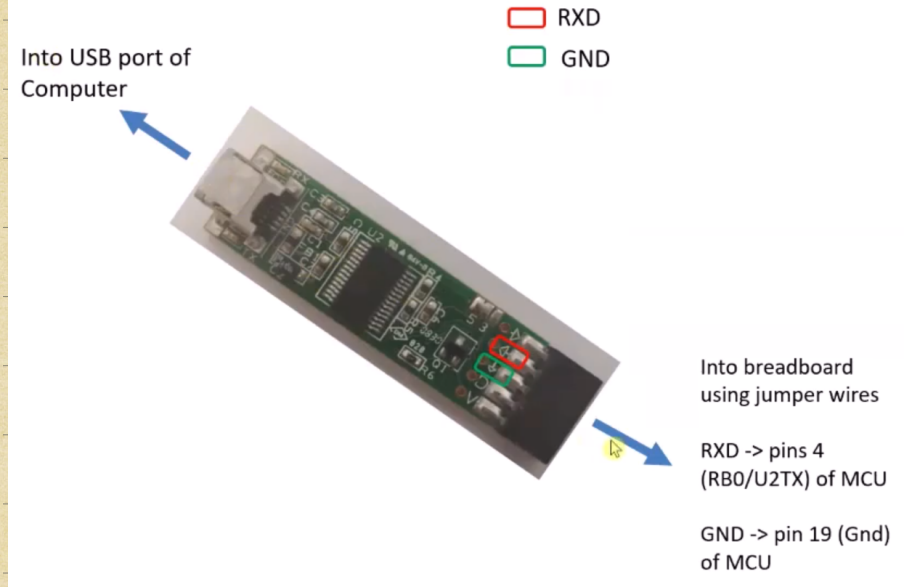
UART→USB Display Functions

- UART
  - Universal Asynchronous receiver & transmitter. RS-232 Communication Protocol
    - Common industrial controllers, machinery & electrical test equipment
    - On the Microcontroller Side
- USB
  - Universal Serial Bus
  - Common in consumer electronics
  - On the PC side
- USB-UART Dongle
  - Provided in attached HW kit
  - Voltage & Serial Data translation between microcontroller (UART) and PC (USB).



USB-UART Modules





UART Display Functions

```
// Takes character and number. E.g Displays S on terminal 2 times
XmitUART2('S', 2);  // same as XmitUART2(83, 2);

// Takes in array of char i.e. a string within quotes and displays it at the beginning of new line
Disp2String("\n\r Hello U of C"); //
// Takes in 16 bit unsigned number and displays it in Hex format
Disp2Hex(Number_16_bit);

// Takes in 32 bit unsigned number and displays it in Hex format
Disp2Hex32(Number_32_bit);

//Displays decimal or floating point number
char str[10];
f= -15.5678;
sprintf (str, "%1.3f", f);  // Converts -15.567 stored in f into an array of characters
Disp2String(str);      // Displays -15.567 on terminal

d= -56;
sprintf (str, "%1.0d", d); // Converts -56 stored in d into an array of characters
Disp2String(str);      // Displays -56 on terminal
```

carriage
\n new line

sprintf → converts the float or double given to an array of characters.

workaround to display floats or integers

ASCII characters — that's what is sent to the terminal.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	⌂

data packets from the microcontroller are sent in 9 bit registers.
1st bit → start bit & 8 are the information. (i.e. number 1 → fill the 8 bits with value 49).

Parameters

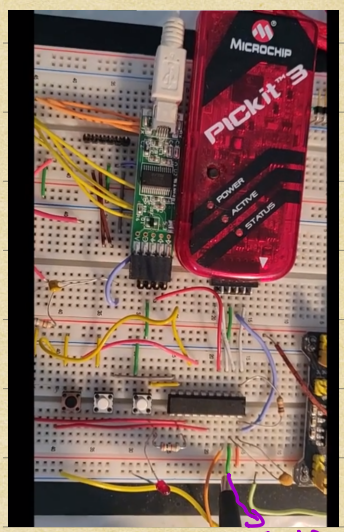
- Number of interrupts & their priority levels?
- Terminal window:
  - DispString("\r"); // return carriage
  - DipString("\n"); // new line

Possible Bugs

- MPLAB sometimes confuses USB-UART module as PICKIT
  - Solution: Connect / Program PICKIT into PC's USB port first.
- Incorrect serial data sync between USB-UART module & PC
  - Solution: Re-program the MCU again; remove & re-connect USB-UART module; reset or re-start terminal program. Make sure that when you set up the terminal window the baud rate on the terminal window, matches up with the

clock speed of the microcontroller.

Display Driver Hardware Video Connections at 9th min.
Pin 20 → connected to the 3 Volt pin of PIC kit.
Big Resistor connects pin 1 & 20
Pin 19 is connected to ground.
For the UART module:
Rxd pin is connected to pin #4
Gnd pin is connected to the ground bus of the breadboard which is connected to the ground of the pic-kit & also connected to the ground of the MCU.
The other end is connected to one USB port & another port ties to the pic-kit. Pin #2 → one that has the arrow towards the computer is the RxD pin. If we have the newer dongle, i.e., not the micro-usb one, it is pin #3. & pin #2 is the GND. (reverse on the micro-usb one)

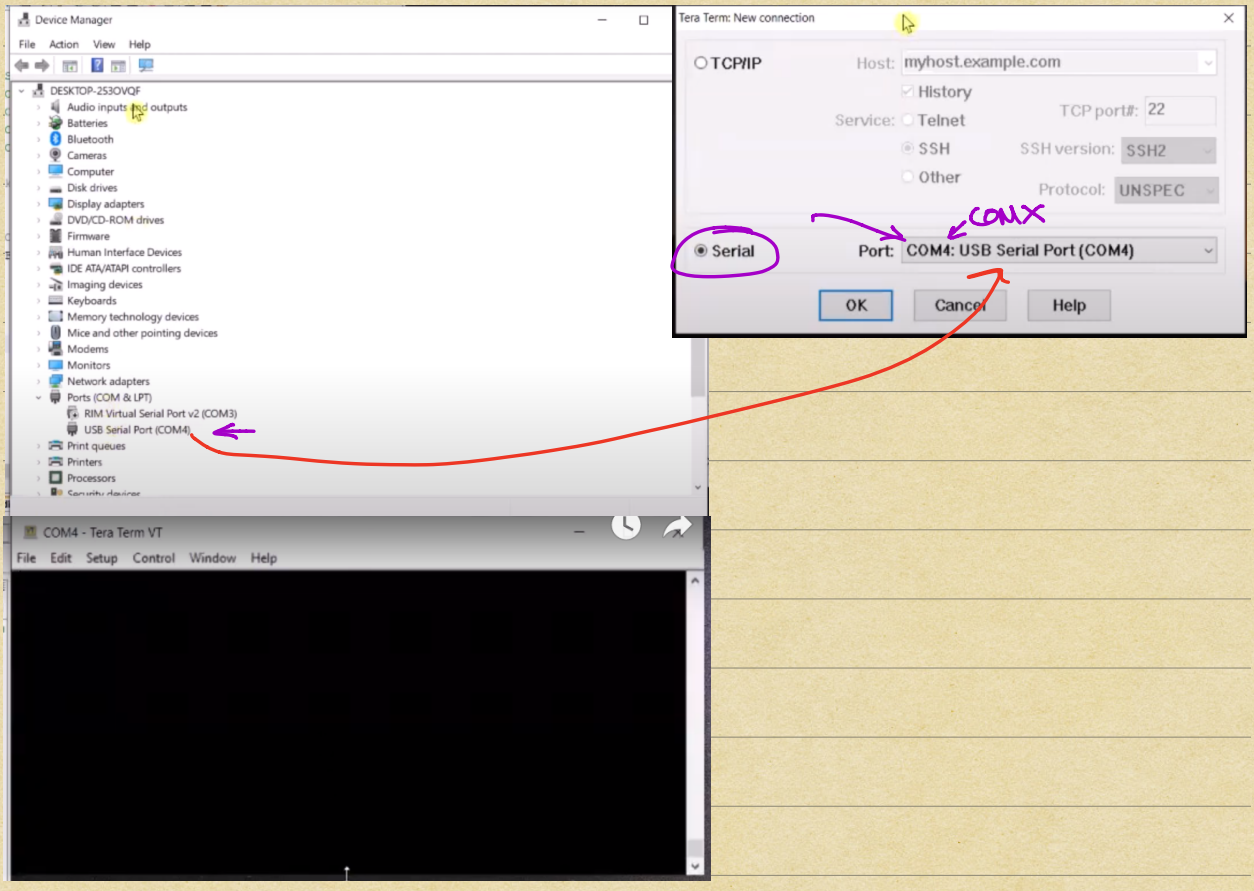


GND of breadboard

To test we use Tera-Term → New Connection → Serial → Port → COM 4/3/2:USB Serial Port (COM2/3/4 etc) → OK

Go to Device Manager → Ports → USB Serial Port (COM 2/3/4) → should appear under Device Management

Note: For good measure open Tera-Term after you connect the hardware. Terminal Window will open



If we assume that we selected an 8 MHz clock

we need to go to terminal → Setup → Serial Port → Check see if the port matches → Speed is 9600 → Data is 8-bit → No Parity → → 1 Stop bit → No Flow Control → 0 transmit Delay

Make sure UART2 settings correspond to our bread board
↳ Set appropriate registers / trisbits for UART functionality (UART.c file, in function InitUART2. There is a Microcontroller UART sheet.

Adjust Speed depending on your clock speed.
Make sure you initialize the hardware

void XmitUART2 → sends a character number to the UART buffer (function receives a char) and displays the character code the specified amount of times

All the display functions use xmitUART2

Disp2Hex → takes a 16 bit number and displays in Hex.
Disp2Hex32 → same but for a 32 bit number
Disp2Dec → Display the 8-bit number in decimal form.
Disp2String → Displays the array of characters.

call XmitUART('G', 1); should sends a G once to the terminal
XmitUART('\n', 1); should send a \n new line character
XmitUART('\r', 1); should send a \r return character
Disp2String("\n\r Hello U of C"); sends a message in one line
To clear the screen go Edit → Clear Buffer → clear Screen
Disp2String("\r Hello U of C"); Writes on the same line over and over if in a while loop (while(1))

Snippet:

uint_8 i; → Max value 255

Disp2String("\r Hello U of C - Today is day ";

Disp2Dec(i);

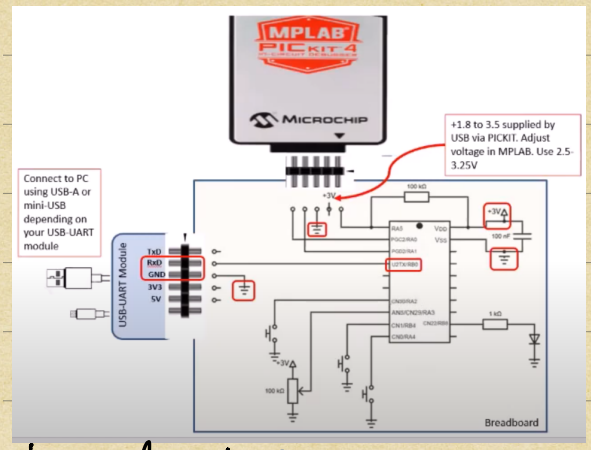
i = (i+1);

Disp2Hex(0x00F8); ← prints hex number over n' over

Let's say we want to display the PORTA value

Disp2String("\rPortA is : ");

Disp2Hex(PORTA);



Since RA4 and RA2 are connected to ground we observe that when buttons are pressed the terminal value changes to 0x0004 or 0x0010 depending on what is pressed. When both are pressed the value is 0x0000

Displaying floating point numbers:

double f; // what we want to do is convert our number to a char

char str[10]; // array and then print that

f = -15.5678;

sprintf(str,"%1.3f ",f); // str is -15.568

Disp2String(str);

XmitUART2('\n',1);

XmitUART2('\r',1);

Note: sprintf uses quite a bit of memory so use it wisely.
Use it sparingly.

Printing signed integers:

```
signed int d;
char str[10];
d= -58;
sprintf(str,"%1.0d",d);
DispStr2String(str);
XmitUART2('\n',1);
XmitUART2('\r',1);
```