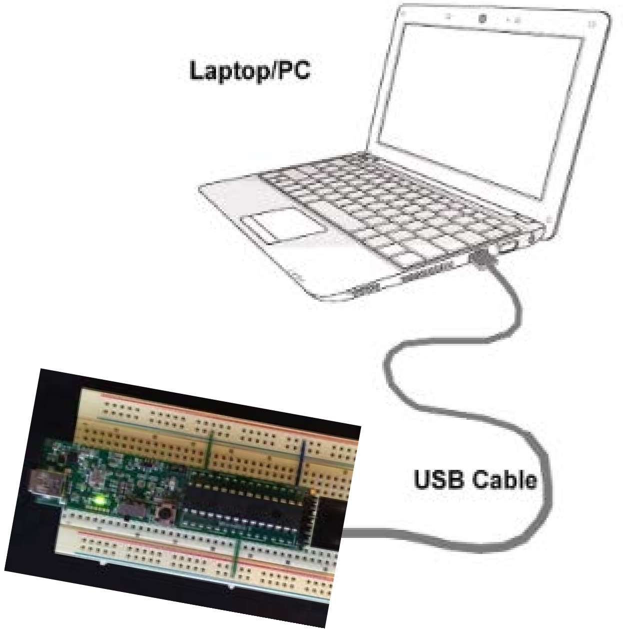
Exercise 2 Prep

Serial Communication with UART

Purpose

- Serial communication
  - Displays, storage, PC communication
  - Few signals required
  - More practical for long distances

- PC ←→ microcontroller
  - Debugging (print to monitor)
  - Keyboard input

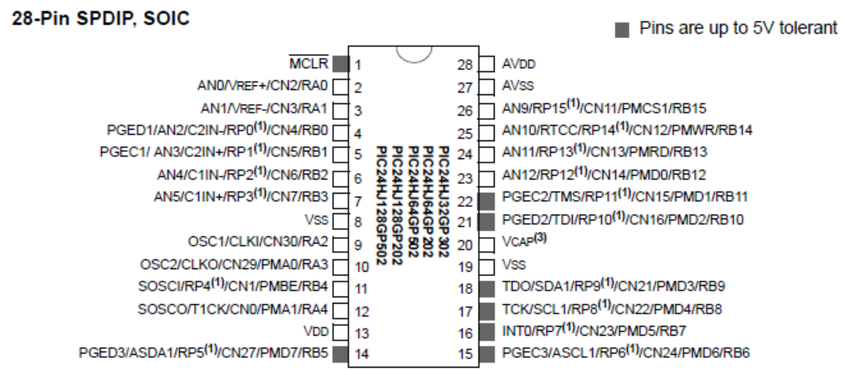


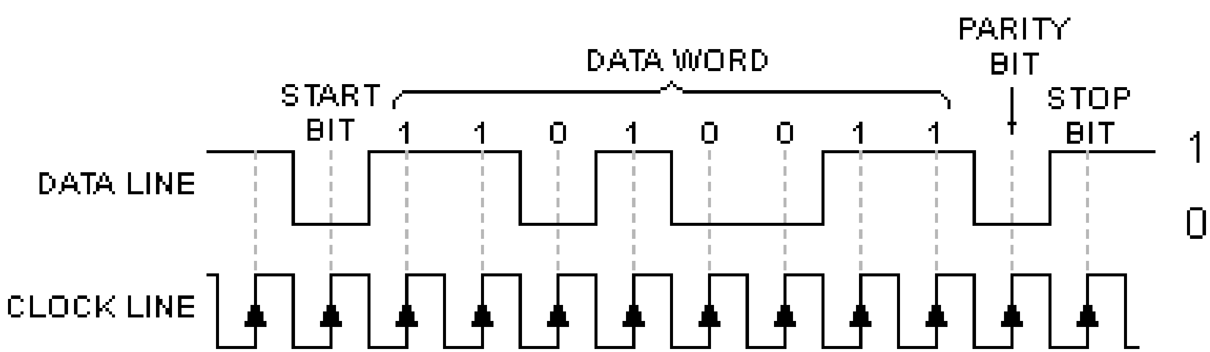
Exercise 2 Overview

- Hardware
  - FTDI serial cable from PIC24 to PC
- Software
  - Textbook UART and Serial library functions
  - Program to communicate using UART
    - Capture key press from keyboard, display on monitor
    - Password program
- Testing
  - Putty terminal emulator (PC) or “screen” (MAC)

New Interfacing Concepts

- PIC24 UART peripheral for serial communication
- RS232 Protocol





Serial Interfaces

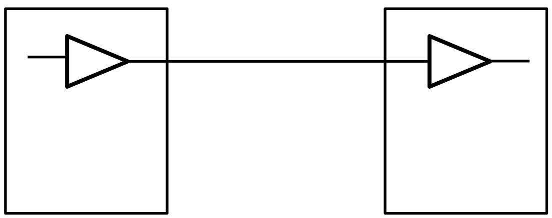
- Serial (vs Parallel Data) Transfer
  - One bit at a time to minimize wires (cost)
  - Less bandwidth (data per second) than parallel
- Types of embedded system serial interfaces:
  - SPI – Serial Peripheral Interface
  - I^2C – Inter-Integrated Circuit
  - RS-232 – Asynchronous, NRZ (Non-Return-to Zero)
  - CAN – Controller Area Network
  - Some others…

Serial Transfer Issues

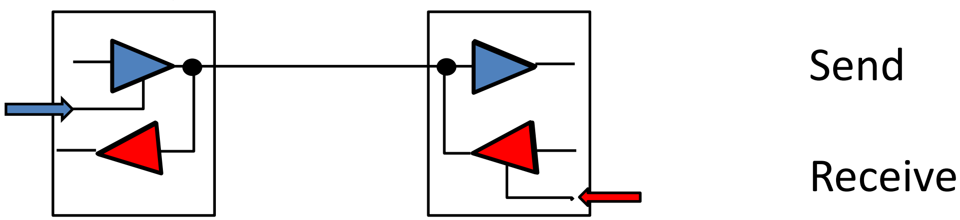
- Reliability in different environments
  - Parallel wires can experience “crosstalk”
  - Long wires introduce delays
- Different protocols for different environments
  - RS-232 – asynchronous, long distances
  - I2C, SPI – synchronous, short distances
  - CAN – synchronous, long distances
- Speed
  - Synchronous serial is faster, but more susceptible to errors from delays
  - Asynchronous is slower, but more reliable for long distances

Serial I/O Channels

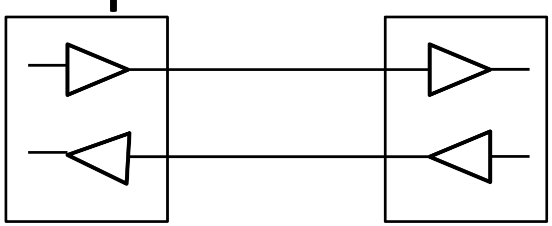
- Simplex Channel – one direction only
  - One unidirectional wire



- Half-duplex Channel – one direction at a time

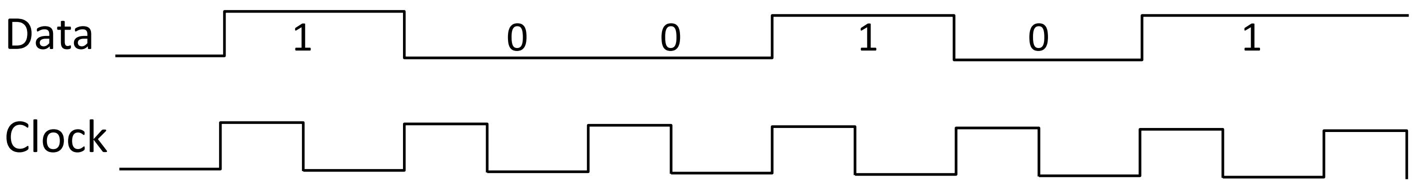


- Full-duplex Channel – both directions at once



Synchronous and Asynchronous Serial

- <u>Synchronous</u> – requires a clock AND data signal, and they are "synchronized".



  – Easy logic (shift registers) for sending and receiving
  – Used in SPI and I2C protocols

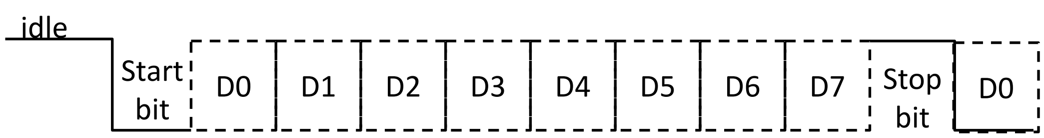
- <u>Asynchronous</u> – no clock signal, so both sides have to "agree" on a data rate

Asynchronous Serial

- Requires one wire for data transfer
- Two wires for full-duplex asynchronous transfer
- Data encoded in “Non-Return-to-Zero” (NRZ) format: 1 = high, 0 = low. Example RS-232
- (Some asynchronous high speed serial use other encodings with two wires for synchronization (FireWire – IEEE 1394))

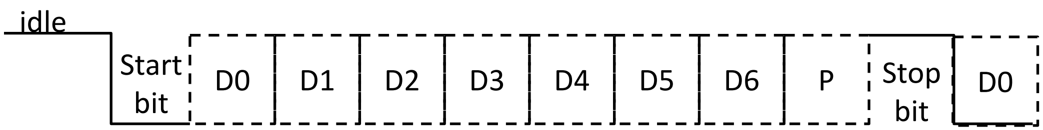
Asynchronous NRZ RS-232

- Both sides of communication agree on speed of transmission – "**baud rate**".
- Time to send one bit is "bit time"
- Transmission line is high (1) when idle.
- A "start bit" (0) or "space condition" initiates a data transfer.
- Least significant bit is sent first
- Transmission ends with a "stop bit" (1)

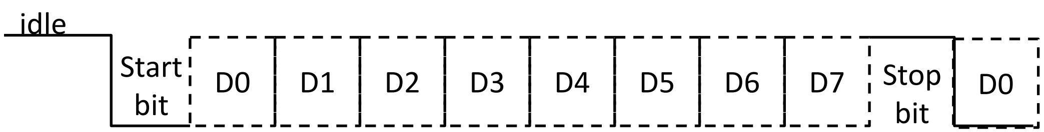


Asynchronous Data Frame Options

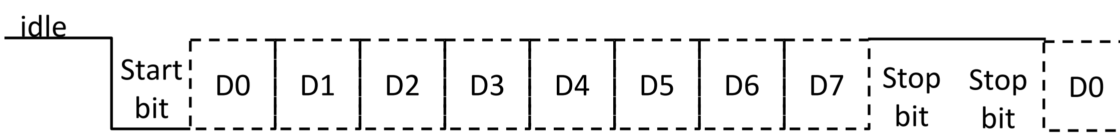
- 7-bits of data, parity bit, 1 stop bit



- 8-bits of data, 1 stop bit

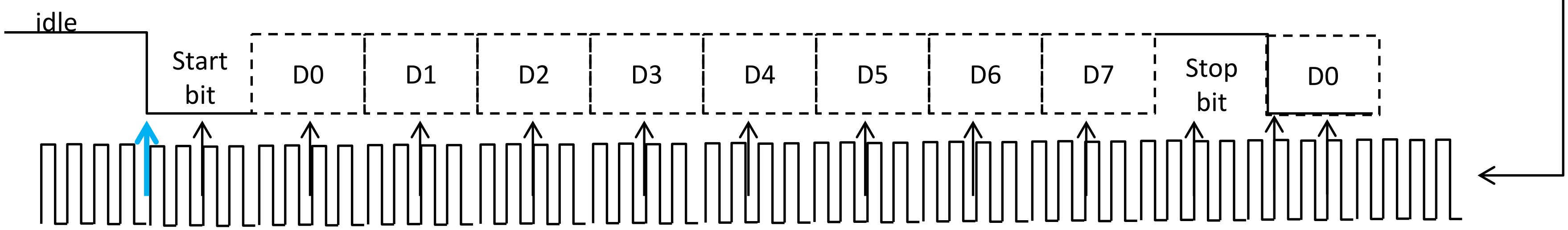


- 8-bits of data, 2 stop bits



How does it work?

- Sender sends frame of data at agreed upon baud rate.
- Receiver uses a clock that is much faster (4x, 16x or 64x) to sample incoming data.
- Receiver checks for “mark” (negative edge) that indicates start bit.
- Once received, it waits ½ bit time and samples again to ensure the start bit is still low.
- If so, it samples the next 8 bits at a frequency of one sample per bit time, and latches them into a shift register.

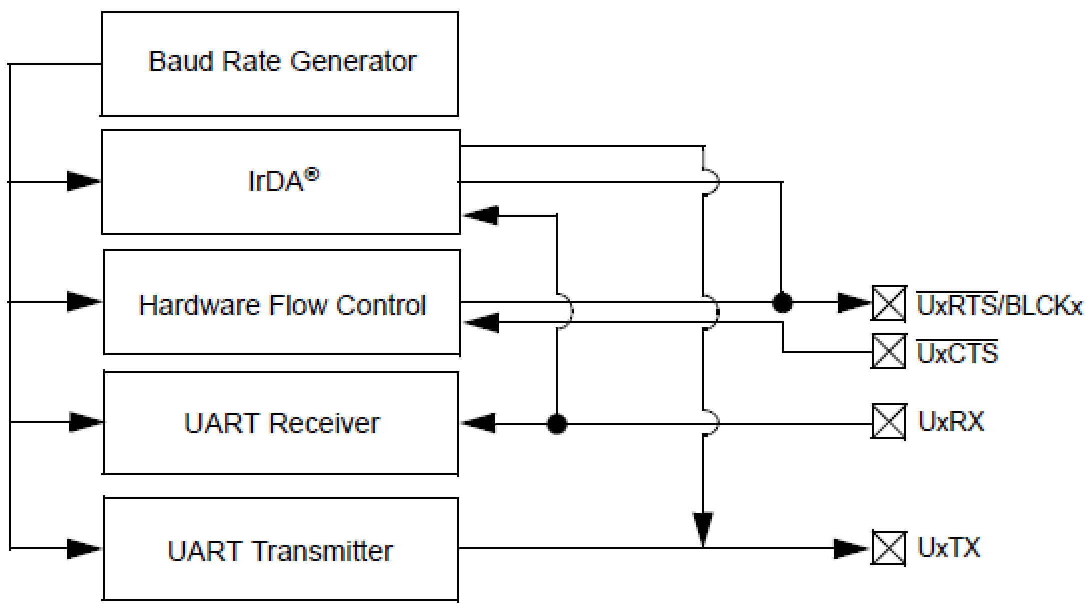


Asynchronous Serial with PIC24

- Software approach – implement asynchronous protocol with a program.
  - Somewhat painful to write.
  - Cannot implement full duplex – CPU is either sending or receiving.
- Hardware approach – use one of the two UART peripherals.

UARTx

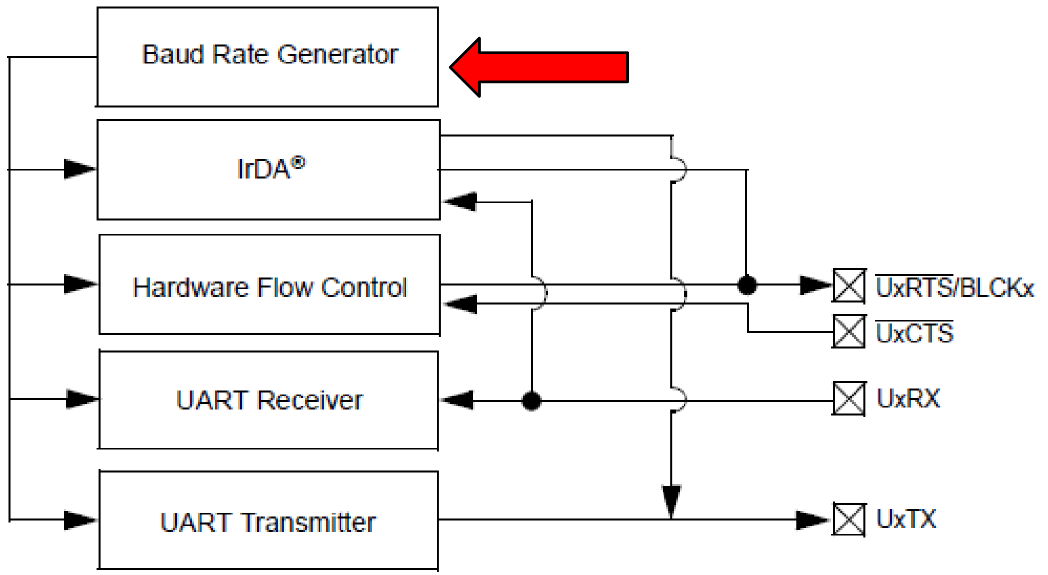
- Our particular PIC24 has two identical UART modules, UART1 and UART2.



UARTx Special Function Registers

- Data registers (shift registers!)
  - UxTXREG – transmit register – use to output data
  - UxRXREG – receive register – use to input data
- Control registers
  - UxMODE – UARTx Mode Register
  - UxSTA – Status and Control Register
  - UxBRG – Baud Rate Register

Baud Rate Generator



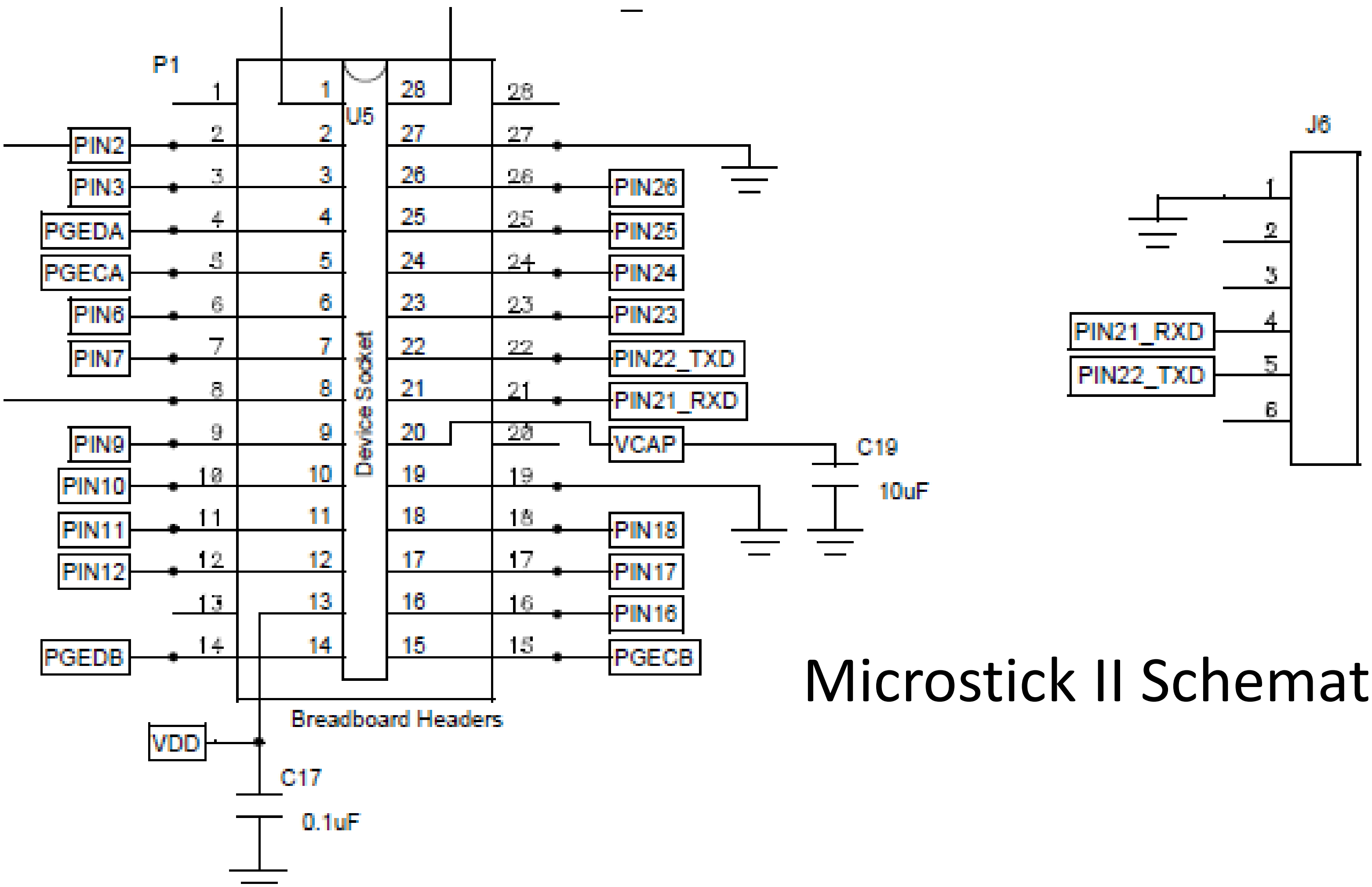
$$Baud\ Rate = \frac{F_{CY}}{16 \times (UxBRG + 1)}$$

$$UxBRG = \frac{F_{CY}}{16 \times Baud\ Rate} - 1$$

Initialization of UART should write to the UxBRG register to set the baud rate.

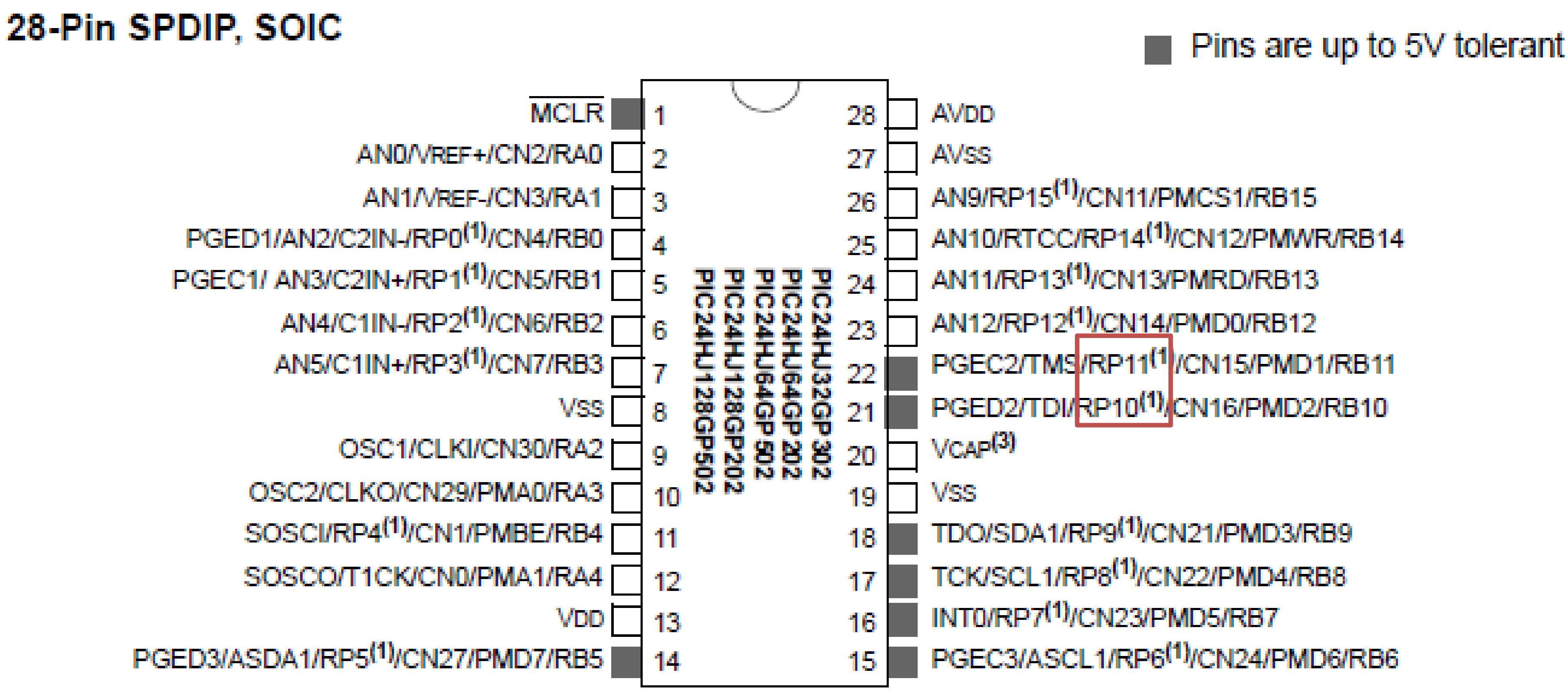
We will use baud rate of **230400 bits/second**

Where are Rx and Tx Pins?



Microstick II Schematic

Remappable Pins: RP10 and RP11



FTDI Cable

Receive and Transmit pins routed to header on Microstick II

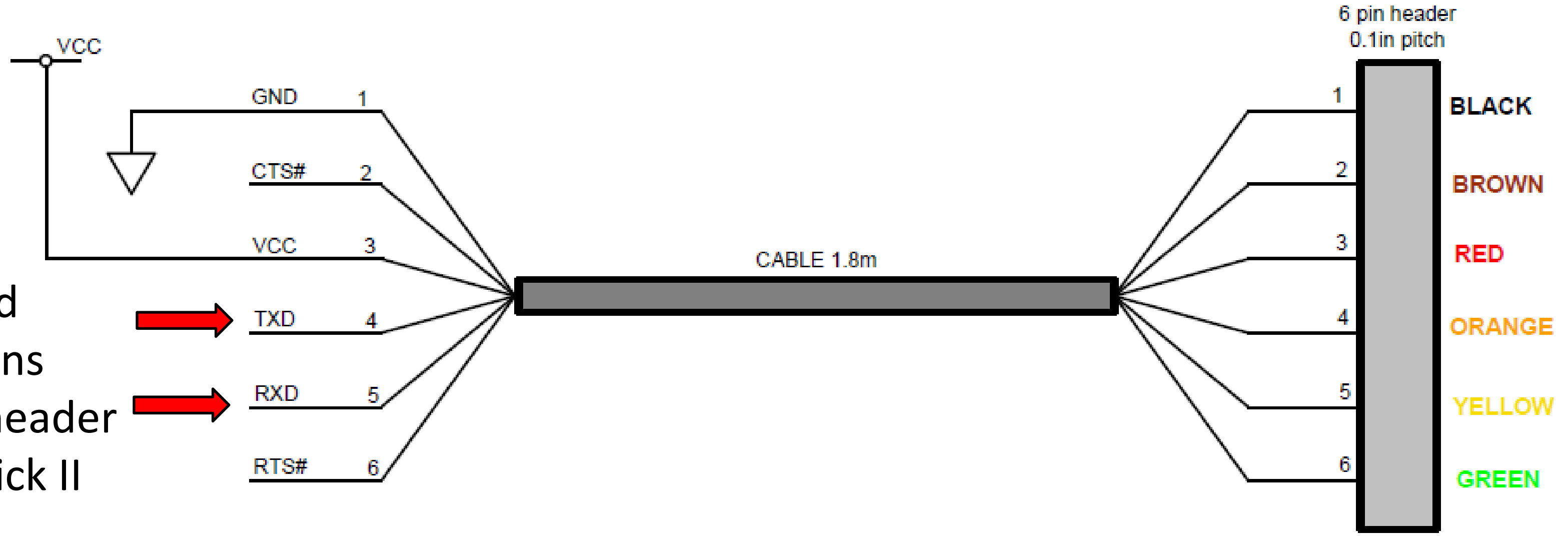
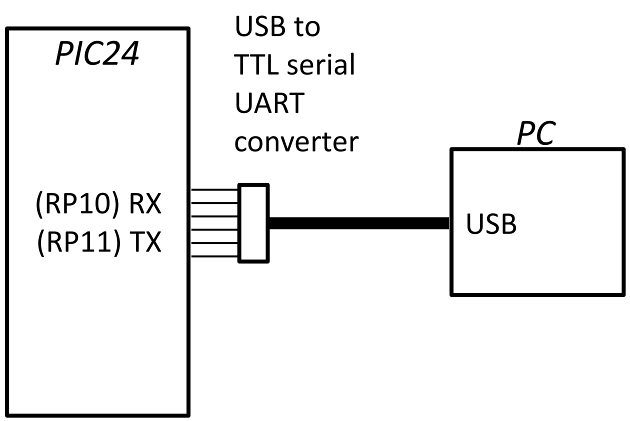
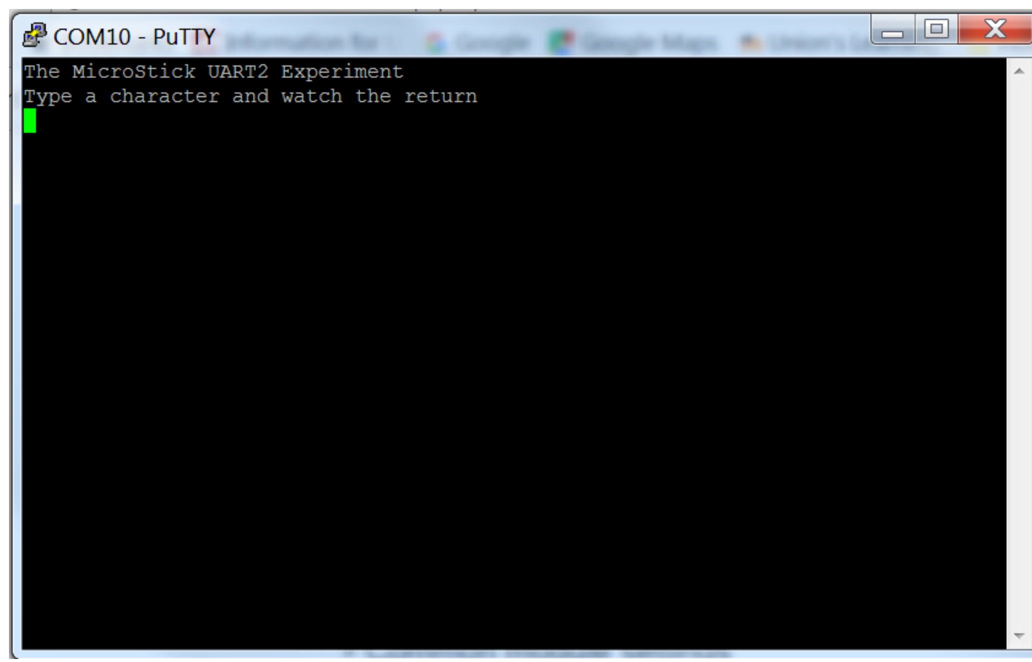


Figure 4.1 TTL-232R-5V and TTL-232R-3V3, 6 Way Header Pin Out

Testing

- Use Putty terminal emulator (PC) or “screen” (MAC)
- Receive characters from keyboard
- Send characters to window

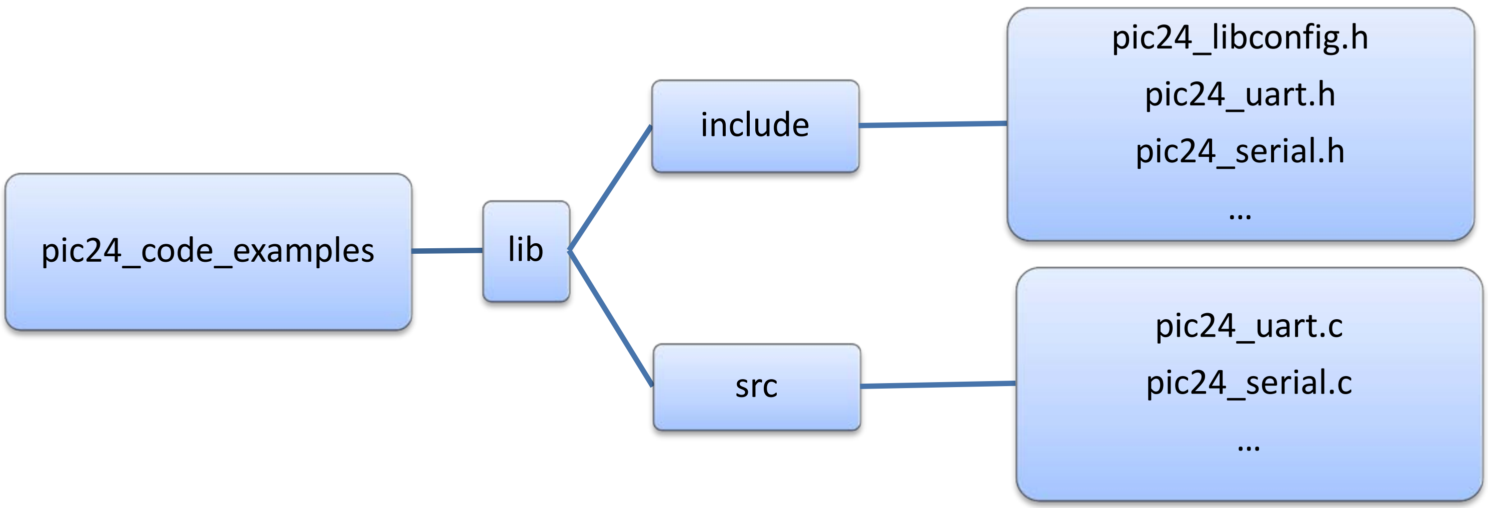




New Software Concepts

- Write program from scratch using template on Nexus.
- Functions in pic24_uart and pic24_serial library files.
- Strings and characters in C

Library Folders and Files



UART Functions in Library

- pic24_uart
  - configUART1(uint32_t u32_baudRate)
- pic24_serial
  - outString (const char *psz_s)
  - outChar(uint8_t u8_c)
  - inChar(void)

Strings and Characters in C

- When referring to an 8-bit character, use single quotes:

  'A'

  Compiler converts to ASCII code

  '8'

- When referring to a string, use double quotes:

  "a string"

  "77"

- String is an array of characters <u>terminated with the null character, '\0',</u>

Arrays in C

- Examples of declaring arrays:

  //An uninitialized array of 10 8-bit characters:

  ```
  uint8_t   u8_cvar[10];
  ```

  //An initialized array of 31 8-bit characters:

  ```
  uint8_t u8_str[] = "A string declared as an array.\n";
  ```

- Array index starts with 0 on leftmost element

  u8_str[0] is ‘A’, u8_str[1] is ‘ ‘

- Example of using array values

  ```
  u8_cvar[3] = u8_cvar[3] + 1;  //Add 1 to array element
  ```

References for UART

P24HJ128GP502 Datasheet

- Available on Nexus
- Section 18 describes UARTS

dsPIC33F/PIC24H Family Reference Manual

- Chapter 18 – UART Reference available on Nexus