In circ_buf.c, if the circular buffer is full, data is lost. While circ_buf.c is written to send a fixed number of samples, it can be easily modified to stream samples indefinitely. If the UART baud is sufficiently higher than the rate at which data bits are generated in the ISR, lossless data streaming can be performed indefinitely. The circular buffer simply provides some cushion in cases where communication is temporarily delayed or disrupted.

11.3.5 Communication with MATLAB

So far, when we have used the UART to communicate with a computer, we have opened the serial port in a terminal emulator. MATLAB can also open serial ports, allowing communication with and plotting of data from the PIC32. As a first example, we will communicate with talkingPIC.c from MATLAB.

First, load talkingPIC.c onto the PIC32 (see Chapter 1 for the code). Next, open MATLAB and edit talkingPIC.m. You will need to edit the first line and set the port to be the PORT value from your Makefile.

Code Sample 11.6 talkingPIC.m. Simple MATLAB Code to Talk to talkingPIC on the PIC32.

```
port='COM3'; % Edit this with the correct name of your PORT.

% Makes sure port is closed
if ~isempty(instrfind)
    fclose(instrfind);
    delete(instrfind);
end
fprintf('Opening port %s....\n',port);

% Defining serial variable
mySerial = serial(port, 'BaudRate', 230400, 'FlowControl', 'hardware');

% Opening serial connection
fopen(mySerial);

% Writing some data to the serial port
fprintf(mySerial,'%f %d %d\n',[1.0,1,2])

% Reading the echo from the PIC32 to verify correct communication
data_read = fscanf(mySerial,'%f %d %d')

% Closing serial connection
fclose(mySerial)
```

The code talkingPIC.m opens a serial port, sends three numerical values to the PIC32, receives the values, and closes the port. Run talkingPIC.c on your PIC32, then execute talkingPIC.m in MATLAB.

We can also combine MATLAB with batch.c or circ_buf.c, allowing us to plot data received from an ISR. The example below reads the data produced by batch.c or circ_buf.c and plots it in MATLAB. Once again, change the port variable to match the serial port that your PIC32 uses.

Code Sample 11.7 uart_plot.m. MATLAB Code to Plot Data Received from the UART.

```
% plot streaming data in matlab
port ='/dev/ttyUSB0'

if ~isempty(instrfind)  % closes the port if it was open
  fclose(instrfind);
  delete(instrfind);
end

mySerial = serial(port, 'BaudRate', 230400, 'FlowControl','hardware');
fopen(mySerial);

fprintf(mySerial,'%s','\n'); %send a newline to tell the PIC32 to send data

len = fscanf(mySerial,'%d'); % get the length of the matrix

data = zeros(len,1);

for i = 1:len
  data(i) = fscanf(mySerial,'%d'); % read each item
end

plot(1:len,data);                % plot the data
```

11.3.6 Communication with Python

You can also communicate with the PIC32 from the Python programming language. This freely available scripting language has many libraries available that help it be used as an alternative to MATLAB. To communicate over the serial port you need the pyserial library. For plotting, we use the libraries matplotlib and numpy. The following code reads data from the PIC32 and plots it. As with the MATLAB code, you need to specify your own port where the port variable is defined. This code will plot data generated by either batch.c or circ_buf.c.

Code Sample 11.8 uart_plot.py. Python Code to Plot Data Received from the UART.

```
#!/usr/bin/python
# Plot data from the PIC32 in python
# requries pyserial, matplotlib, and numpy
import serial
import matplotlib.pyplot as plt
```

```
import numpy as np

port = '/dev/ttyUSB0' # the name of the serial port

with serial.Serial(port,230400,rtscts=1) as ser:
  ser.write("\n".encode())  #tell the pic to send data. encode converts to a byte array
  line = ser.readline()
  nsamples = int(line)
  x = np.arange(0,nsamples) # x is [1,2,3,... nsamples]
  y = np.zeros(nsamples)# x is 1 x nsamples an array of zeros and will store the data

  for i in range(nsamples): # read each sample
    line = ser.readline()   # read a line from the serial port
    y[i] = int(line)        # parse the line (in this case it is just one integer)

plt.plot(x,y)
plt.show()
```

11.4 Wireless Communication with an XBee Radio

XBee radios are small, low-power radio transmitters that allow wireless communication over tens of meters, according to the IEEE 802.15.4 standard (Figure 11.3). Each of the two communicating devices connect to an XBee through a UART, and then they can communicate wirelessly as if their UARTs were wired together. For example, two PIC32s could talk to each other using their UART3s using the NU32 library.

XBee radios have numerous firmware settings that must be configured before you use them. The main setting is the wireless channel; two XBees cannot communicate unless they use the same channel. Another setting is the baud, which can be set as high as 115,200. The easiest way to configure an XBee is to purchase a development board, connect it to your computer, and use the X-CTU program provided by the manufacturer. Alternatively, you can program XBees directly using the API mode over a serial port (either from your computer or the PIC32).

After setting up the XBees to use the desired baud and communication channel, you can use them as drop-in replacements, one at each UART, for the wires that would usually connect them.[3]

[3] One caveat occurs at higher baud rates. The XBee generates its baud by dividing an internal clock signal. This clock does not actually achieve a baud of 115,200, however; when set to 115,200 the baud actually is 111,000. Such baud rate mismatches are a common issue when using UARTs. Due to the tolerances of UART timing, the XBee may work for a time but occasionally experience failures. The solution is to set your baud to 111,000 to match the actual baud of the XBee.