How to add Encoded Radio Communications to your Project

Table of Contents

Intro

Using radio communications in your project can make communicating over long distances easier as you don't need to worry about wiring up controllers.

There are 2 main types of radio communications that are available, cheaper without an encoder/decoder, or more expensive with an encoder/decoder. The encoder/decoder duo automatically encodes communications with various encoding standards. If you do not have an encoding radio, you will have to do encoding and decoding in software, which is slower and more difficult, but cheaper.

This is the significantly easier to use and highly recommended to invest in. This will communicate over serial pins as it's durable enough to work without many errors. With encoders, you can send characters of raw bytes of data quite easily. I will be referring to baud rate in this write up constantly, which is an outdated transmission measurement which is refers to "symbols/second". Symbols used to be defined as a single ASCII character, but has since been redefined to 2 standards, CS7 and CS8, as 7 and 8 characters. The standard baud rates that could be used are 300, 1200, 2400, 4800, 9600, 14400, 19200. There are other baud rates that can be used that aren't standardized (we used 600).

Hardware

The hardware steps for one way communication are as follows:

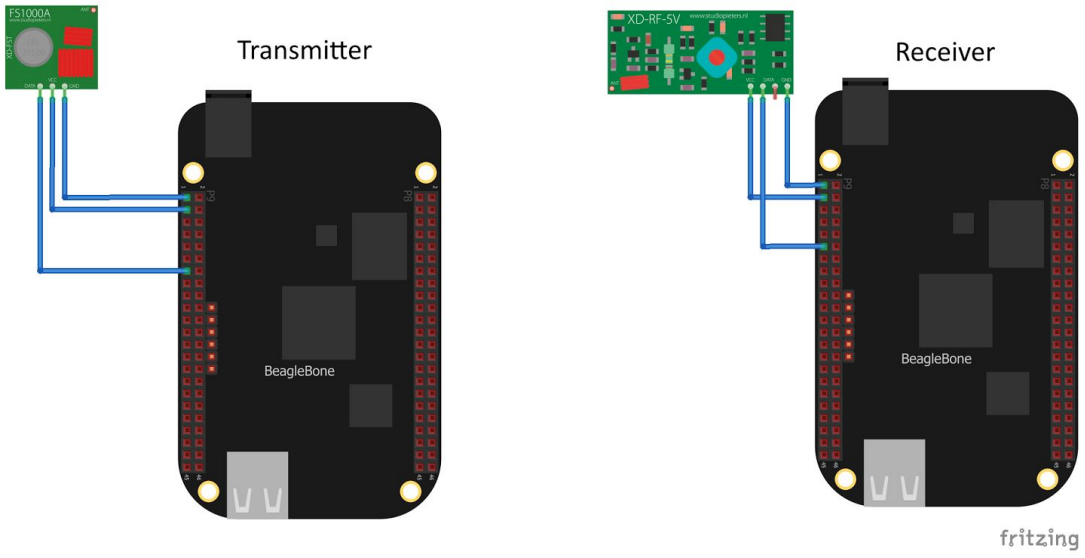
Transmitter

1. Connect the VDD to the 3.3v output of the transmitting beaglebone
2. Connect the Ground to the ground of the transmitting beaglebone
3. Connect the Data transmission pin to the transmitting UART TX pin of the beaglebone
    a. The Beaglebone green has 5 UART(Universal Asynchronous Receiver/Transmitter) pair pins, they are shown in table 1

Receiver

1. Connect the VDD to the 3.3v output of the receiving beaglebone
2. Connect the ground to the ground pin of the receiving beaglebone
3. Connect the Data transmission pin to the receiving UART RX pin of the beaglebone

If you have 2 radios, or transceivers (ones that can both **trans**mit and re**ceive**) you can hook up both beaglebones as both receiver and transmitter. CTS and RTS stand for clear to send and ready to send, they are primarily used for bidirectional communication over limited physical mediums and wouldn't be used for radio communications.



UART Pair	Tx Pin	Rx Pin	RTS	CTS
1	24	26	17	20
2	21	22	N/A	N/A
3	N/A	N/A	34	36
4	13	11	33	35

5	37	38	32	31

Testing

Testing communication for the radios is fairly simple. Like the hardware, most of the steps are the same on both the receiver and transmitter.

Receiver

1. Enable the UART that the radio communication pin is plugged into. "#" denotes the UART that is being used.
    - `echo BB-UART# > /sys/devices/platform/bone_capemgr/slots`
2. Set the baud rate of the UART to one that the radio can work with. I highly recommend testing this thoroughly with different baud rates and different baud rates as wrong baud rates can send malformed packets. Where # refers to the UART pin, * refers to either cs7 (for ascii characters) or cs8(for UTF8 or raw bytes), and ^ is the baud rate
    - `stty -F /dev/tty0# ^^^,cs*`
3. Read the output of the pin, which will be persistent until you exit it with CTRL+C
    - `cat /dev/tty0#`

Transmitter

(The first 2 steps are the same)
The baud rate

1. Enable the UART that the radio communication pin is plugged into. "#" denotes the UART that is being used.
    - `echo BB-UART# > /sys/devices/platform/bone_capemgr/slots`
2. Set the baud rate of the UART to one that the radio can work with. I highly recommend testing this thoroughly with different baud rates and different baud rates as wrong baud rates can send malformed packets. Where # refers to the UART pin, and * refers to either cs7 (for ascii characters) or cs8(for UTF8 or raw bytes)
    - `stty -F /dev/tty0# ^^^,cs*`
3. Echo to the UART pin
    - `echo "Hello World!" > /dev/tty0#`

The data that you echoed to the serial UART pin, should magically appear in the cat of the receiving beaglebone. If you want to transmit raw data instead of characters, you will want to use CS8, and to echo the hex-value of the data in the form "`echo //FF//00//## > /dev/tty0#`". If you do with while using cat on the receiving end, more than likely it will receive a value that doesn't refer to a character, so it won't know what to render. It will output garbage data. If you want to see the hex values on the receiving end, you will have to write a C program to receive it and display it properly.

C programming

Programming C to work with writing to and reading from serial ports is fairly simple. It's not even as difficult as writing to and reading from a file. We are using a library called terminos

that makes communicating over terminal communications very easily. This will setup bidirectional communication on the radio, but you can use it for unidirectional communication as well if you only need to communicate one way.

For the library information, see
http://pubs.opengroup.org/onlinepubs/007908799/xsh/termios.h.html

Like the Hardware and testing, the first couple steps are identical.

Transmitter and Receiver

1. Add the dependency
    - `#include <termios.h>`
    - `#include <fcntl.h>`
2. Enable the UART - you can do this manually beforehand, but is recommended to do in the code to make sure it doesn't segfault. If it is already enabled then it will have a runtime warning. I'm going to assume you already having a function that can write to a file called "writeToFile(file location, value to write)".
    - `writeToFile("/sys/devices/platform/bone_capemgr/slots", "BB-UART#");`
3. Open the file with open(), in reading mode,, with TTY options and in nonblocking mode - which returns an error immediately if it can't read/write to the specified location
    - `int fileLocation = open("/dev/tty#", O_RDWR | O_NOCTTY | O_NONBLOCK);`
4. Test the fileLocation variable to make sure it's a non-negative number, if it is, then the UART is being used by another process.
5. Setup and run functions on the terminos struct, that will configure the TTY port to operate properly
    - `struct terminos ttyControl;`
    - `tcgetattr(fileLocation, &ttyControl);`
        - Initializes values in the struct to talk to terminals
    - Set radio options in the struct

Struct Value	Set to
c_cflag	B^^^ | CS8 | CLOCAL | CREAD
c_iflag	IGNPAR
c_oflag	0
c_lflag	0

    - `tcflush(fileLocation, TCIOFLUSH);`
        - This clears any data in the buffer before sending or receiving.
    - `tcsetattr(fileLocation, TSCANOW, &options);`
        - This associates the terminos struct with the terminal location

Receiver

6. The terminal is all set up to read from, now all you need to do is read from the file using read(). This will read to a previously created unsigned char buffer. To continuously read from the file put the read in a while loop. $ refers to the size of the buffer-1.
    - `read(fileLocation, (void*)unsignedCharBuff, $);`

Sender

6. The terminal can now be written to using the write() command.
    - `write(fileLocation, &string or &char, size of string or char);`
        - If you are transmitting raw data in unsigned from (unsigned char), you set the size of char to 1.