A complete guide to the RC car cape development for the BeagleBone Black

Developed by Alexandr Sachkov
12/1/2014
Introduction

Originally developed for
Project RIDN

Team Members:

Alexandr Sachkov
Harveen Atwal
Ravkirt Takhar
Shaheer Shabbir

Permission is granted to Simon Fraser University and Dr. Brian Fraser to distribute this guide for student access.

If you ever wanted an RC car that could be driven programmatically or if you are just looking for a cool project for your BeagleBone Black, then this guide is for you.

In just a few easy steps, we will create a custom cape to interface BeagleBone Black with any commercially available RC car. This guide assumes some previous knowledge of the BeagleBone Black and embedded Linux as well as some ability to solder.
Tools and Materials:

- BeagleBone Black
- RC car of your choice (and its remote control)
- Breadboard or, for a more permanent solution, a Proto board
- Soldering Iron with a fine tip
- Solder
- Flux or Flux-Core solder (not required but it improves the quality of your solder joints)
- Spool of wire (20 - 22 gauge)
- Philips screw driver
- Wire strippers (or a knife)
- 2N3904 transistors x4
- 470 Ohm resistors x4
Build procedure:

Step 1: Soldering connections onto the car's remote control

Remove the batteries from your RC remote control, extract all the screws and open the back cover. Carefully extract all the screws holding the circuit in place, and remove the circuit (Be very careful not to damage any components). Remove the analog sticks if they are still attached to the board.

You will have to solder some wires to this board. Prepare 5 wire slices long enough to reach from the RC circuit to your Breadboard, and strip both ends by about 1/8 inch. Leave some slack as you can always trim them later.

Examine the RC circuit and find the 2 locations where the analog sticks used to be attached. Each analog stick controls the direction of rotation of one of the motors, so after examining their locations, you should notice that each one contains two analog switches where the front-most switch spins the motor forward, and the other, in reverse.

It's time for some circuit tracing. Pick a switch, then flip over the board and find the two contacts that connect to it. The trace for one of the contacts will lead to a wider trace running along the outer perimeter of the circuit which by convention is ground. You want the other contact. Dip one end of a wire into flux and solder it to the contact. Be careful not to overheat the board and do not let the solder flow over multiple traces. Now repeat the process for the other 3 switches, and make sure to label the wires with their corresponding switch locations.

Find the battery clip contacts on the board and solder the fifth wire to the negative contact. Reinsert the circuit into the bottom half of the remote control as we will still need its battery back to power the remote.
Step 2: Build the transistor switching circuit

It is time to build the switching circuit for your RC remote.

1. Insert the four 2N3904 transistors into the Breadboard so that their labels are facing you.
2. Insert the four 470 Ohm resistors with one end connecting to the center pin of each transistor. Leave the other end disconnected for now.
3. Connect the left-most pin of each transistor to the ground rail on the Breadboard.
4. Connect the motor control wires to the right-most transistor pins, one wire per transistor. These transistors will act as electronic switches for the motor controls.
5. Connect the battery clip wire to the ground rail on the Breadboard.

Prepare 5 wire slices long enough to reach from your Breadboard to the cape expansion headers on the BeagleBone. Connect one end of each wire to the other end of each resistor. Then connect the fifth wire to the ground rail.

Now check with the BeagleBone cape extension header pinout diagram and find 4 available GPIO pins (they are highlighted with a light green color). Connect your 4 transistor wires to GPIO pins. Note which GPIO pin each switch is connected to. Connect your ground wire to any pin labeled DGND.

Congratulations! You are all done with the cape hardware!
Step 3: Writing the driver for the RC cape

The cape hardware is built but we still need a driver to control it which we will write using BoneScript.

BoneScript is a Node.js library that comes preinstalled on the BeagleBone and provides a simple interface to control GPIO. It also makes it easy to interface with a Node.js server in case you want to control the car remotely.

First of all, we will write some code to control a single car motor. Create a new file called "rcmotor.js" and add the following code to it:

```javascript
var b = require('bonescript');

function RCMotor(fwdPort, rvsPort){
    var fwdPortNumber = fwdPort;
    var rvsPortNumber = rvsPort;
    
    b.pinMode(fwdPortNumber, 'out');
    b.pinMode(rvsPortNumber, 'out');
    
    var off = 0;
    var on = 1;
    
    this.fwd = function(){
        b.digitalWrite(rvsPortNumber, off);
        b.digitalWrite(fwdPortNumber, on);
    };
    
    this.rvs = function(){
        b.digitalWrite(fwdPortNumber, off);
        b.digitalWrite(rvsPortNumber, on);
    };
    
    this.off = function()
    {
        b.digitalWrite(fwdPortNumber, off);
        b.digitalWrite(rvsPortNumber, off);
    };
}

module.exports = RCMotor;
```

The above code creates an object called RCMotor which takes two parameters fwdPort and rvsPort. These parameters are the GPIO pin numbers that you connected your transistor wires to. The forward port refers
to the wire that drives the corresponding motor forward, and the other in reverse. Make sure that the two wires drive the same motor, or otherwise, your build will misbehave.

RCMotor has 3 public functions to drive it forward, in reverse, and stop.

Next, we will write some code that builds on the motor code and represents the RC car as a whole. Create a file named "rccontroller.js" in the same directory as the "rcmotor.js" and add the following code to it:

```javascript
var RCMotor = require('./rcmotor.js');

function RCController(){
    var leftMotor = new RCMotor("P8_8", "P8_10");
    var rightMotor = new RCMotor("P8_12", "P8_14");

    var no_direction = 0;
    var fwd_direction = 1;
    var rvs_direction = 2;

    var current_direction = no_direction;

    this.stop = function(){
        console.log("stopping.");
        leftMotor.off();
        rightMotor.off();
    }

    this.stop();

    this.fwd = function(){
        console.log("driving forward.");
        leftMotor.fwd();
        rightMotor.fwd();
    }

    this.rvs = function(){
        console.log("driving in reverse.");
        leftMotor.rvs();
        rightMotor.rvs();
    }

    this.left = function(){
        console.log("turning left.");
        leftMotor.off();
    }
```
```
rightMotor.fwd();

this.rotateLeft = function(){
  console.log("rotating left.");
  leftMotor.rvs();
  rightMotor.fwd();
}

this.right = function(){
  console.log("turning right.");
  leftMotor.fwd();
  rightMotor.off();
}

this.rotateRight = function(){
  console.log("rotating right.");
  leftMotor.fwd();
  rightMotor.rvs();
}
```

`module.exports = RCController;`

The above code creates an RCController containing two motors that are passed their corresponding GPIO pin numbers. In this example, I am using pins 8, 10, 12, and 14 on the P8 expansion header where:

- pin 8 drives the left motor forward
- pin 10 \(\rightarrow\) left motor in reverse
- pin 12 \(\rightarrow\) right motor forward
- pin 14 \(\rightarrow\) right motor in reverse

The public interface contains 7 intuitive functions to control the car.
Step 4: Testing!

Congratulations, you made it this far!

It is time to test the build. Double check your wiring configurations and stick the batteries back into the RC remote.

Create a file called "rctest.js" in the same directory as the "rccontroller.js" and add the following code to it:

```javascript
var RCController = require('./rccontroller.js');
var controller = new RCController();

controller.fwd();
setTimeout(function(){
    controller.rvs();
    setTimeout(function(){
        controller.stop();
    },3000);
}, 3000);
```

The above code will make your car drive forward for 3 seconds followed by reversing for 3 seconds. If your RC remote contains an LED indicator, then you can leave your car off.
Run the above code using the following command:

```
# nodejs rctest.js
```

On some boards it is:
```
# node rctest.js
```

You should see the controller LED light up and stay lit throughout the test. Once this test is passed, turn on your car and make sure that the car behaves as expected.
**Troubleshooting:**

- **Node.js or BoneScript is not installed:**
  Install them. There are many online guides for this.

- **GPIO pin **** is taken:**
  This is caused by a bug in the BoneScript. Reconfigure to use different GPIO pins and try again. You could also upgrade to the later firmware image which fixes the bug.

- **The controller LED does not turn on:**
  Double check your wiring, make sure you reinstalled the battery into the remote control. If the problem persists, unplug the wires from the BeagleBone GPIO header and touch them one at a time to the positive battery terminal on the remote control. The LED should light up each time. If the LED lights up, your problem is caused by the software configuration. Otherwise, check your Breadboard connections and solder joins on the remote control circuit.

- **The car drives in mysterious ways:**
  Check for proper configurations one motor and direction at a time. Determine which motor/direction is malfunctioning, then check its corresponding wiring and GPIO configurations.

For further support, please contact me at **alexsachkov@hotmail.com**