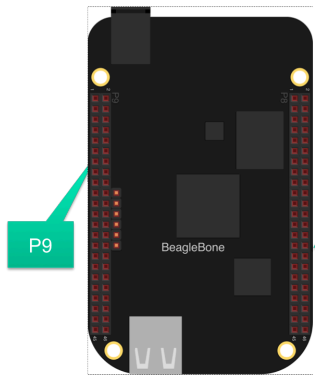


# BeagleBone Black Hands-on Workshop

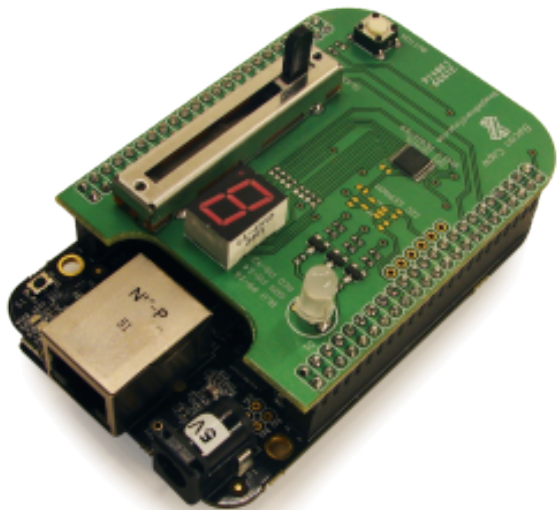
BeagleBone Black is a US\$45, 1GHz computer that ships with a Linux distribution on its on-board 2GB flash and has extensive I/O capabilities. We'll be focusing on how you can wire up sensors and actuators using an included JavaScript library and IDE.



P9				P8			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	MMC1_DAT6	3	4	MMC1_DAT7
VDD_5V	5	6	VDD_5V	MMC1_DAT2	5	6	MMC1_DAT3
SYS_5V	7	8	SYS_5V	GPIO_66	7	8	GPIO_67
PWR_BTN	9	10	SYS_RESETN	GPIO_69	9	10	GPIO_68
UART4_RXD	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
UART4_TXD	13	14	EHRPWM1A	EHRPWM2B	13	14	GPIO_26
GPIO_48	15	16	EHRPWM1B	GPIO_47	15	16	GPIO_46
GPIO_4	17	18	GPIO_5	GPIO_27	17	18	GPIO_65
I2C2_SCL	19	20	I2C2_SDA	EHRPWM2A	19	20	MMC1_CMD
EHRPWM0B	21	22	EHRPWM0A	MMC1_CLK	21	22	MMC1_DAT5
GPIO_49	23	24	UART1_TXD	MMC1_DAT4	23	24	MMC1_DAT1
GPIO_117	25	26	UART1_RXD	MMC1_DAT0	25	26	GPIO_61
GPIO_125	27	28	SPI1_CS0	LCD_VSYNC	27	28	LCD_PCLK
SPI1_DO	29	30	GPIO_122	LCD_HSYNC	29	30	LCD_AC_BIAS_E
SPI1_SCLK	31	32	VDD_ADC	LCD_DATA14	31	32	LCD_DATA15
AIN4	33	34	GNDA_ADC	LCD_DATA13	33	34	LCD_DATA11
AIN6	35	36	AIN5	LCD_DATA12	35	36	LCD_DATA10
AIN2	37	38	AIN3	LCD_DATA8	37	38	LCD_DATA9
AIN0	39	40	AIN1	LCD_DATA6	39	40	LCD_DATA7
GPIO_20	41	42	ECAPPWM0	LCD_DATA4	41	42	LCD_DATA5
DGND	43	44	DGND	LCD_DATA2	43	44	LCD_DATA3
DGND	45	46	DGND	LCD_DATA0	45	46	LCD_DATA1

## Bacon Cape

The Bacon Cape is an add-on board for BeagleBone Black available from boardzoo.com designed to teach and demonstrate simple electronics interfaces without the need to perform any wiring. The individual components can be used to complete the same exercises if you are willing to able them up properly.



### Components

- Tricolored RGB LED configured as active-high
  - Red: P9 pin 42
  - Green: P9 pin 16
  - Blue: P9 pin 14
- 74HC595 serial shift reg with active-low 7 segment display
  - Data: P9 pin 18
  - Clock: P9 pin 22
  - Latch: P9 pin 17
  - Clear: P9 pin 15
- 10k slider potentiometer
  - Wiper: P9 pin 36
- Normally-open push button configured as active-low
  - Contact: P8 pin 19
- MMA8452Q triple axis accelerometer on I2C2
  - SDA: P9 pin 20
  - SCL: P9 pin 19

[http://elinux.org/Bacon\\_Cape](http://elinux.org/Bacon_Cape)

### Task #1 – Blink red LED

Browse to <http://192.168.7.2:3000> from Chrome, then type and ‘run’ the following program. You can leave out any comments, namely text between “/\*” and “\*/” as well as on a line following “//”. ‘setInterval()’ is a built-in JavaScript function that runs the function in the first argument at an event rate of the number of milliseconds in the second argument. You can use <http://192.168.7.2> to see documentation on the BoneScript library functions. The board is acting as the web server over the virtual network connection over the USB cable.

Red LED is controlled using pin 42 on connector P9.

```
/*
 * Setup
 */
var b = require("bonescript"); // Read BoneScript library

// Map used pins
var LED_RED = "P9_42";

// Define global variables
var state = false;

// Configure pins
b.pinMode(LED_RED, b.OUTPUT);

/*
 * Add handlers
 */
setInterval(blink, 1000); // call blink() every 1s

/*
 * Define functions
 */
function blink() {
    state = !state;
    b.digitalWrite(LED_RED, state ? b.HIGH : b.LOW);
}
```

### Task #2 – Use button to set blue LED

Read from the button and light the blue LED when the button is pressed. Note that when the button is *not* pressed, the state of the digital input pin will be a 1 or **high** due to the resistor pulling the pin “up”. When the button is pressed, the digital input pin will be grounded **low** and have the value 0. Reading every 100ms is a reasonable rate. You’ll need to use pinMode() to set the button to be an input.

```
function readBUTTON() {
    var status = b.digitalRead(BUTTON);
    b.digitalWrite(LED_BLUE, status ? b.LOW : b.HIGH);
}
```

### Task #3 – Use potentiometer to control green LED brightness

Configure the LED brightness to track the analog voltage applied to P9\_36. Reading every 100ms is a reasonable rate. Use the following code snippet.

```
function readPOT() {
    var value = b.analogRead(POT); // Read voltage from POT
    b.analogWrite(LED_GREEN, value); // Set LED brightness
}
```

### Task #4 – Use shiftOut() to set 7 segment display (requires BoneScript 0.2.3)

Use the following array to store the values to be shifted out to the 7 segment displays.

```
var s=[0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90];
```

You'll need to configure the DATA, CLOCK, LATCH and CLEAR signals as output and set the CLEAR signal to be HIGH all the time. 100ms is a reasonable update rate. Use the following code snippet to perform the update.

```
function update7Seg() {
    digit=(digit+1)%10; // Increment and wrap digit (0-9)

    // Shift out the character LED pattern
    b.shiftOut(S_DATA, S_CLOCK, b.MSBFIRST, s[digit]);
    b.digitalWrite(S_LATCH, b.HIGH);
    b.digitalWrite(S_LATCH, b.LOW);
}
```

### Task #5 – Edit task #4 to use analogRead() value to update 7 segment LED

Use the following code snippet to switch from incrementing digit during update7Seg() to reading the value from the potentiometer.

```
// Read the voltage from potentiometer
var value = b.analogRead(POT);

// Convert floating point value 0-1 to digit 0-9
digit = parseInt(value*10, 10) % 10;
```

### Task #6 – Use i2cOpen()/i2cWriteBytes()/i2cReadBytes() to read accelerometer

The accelerometer is wired up to I2C-2 at address 0x1C and must be placed in STANDBY mode by writing a 0 to register 0x2A, have the scale set to 2G by writing 0 to register 0x0E and then set to ACTIVE mode by writing 1 to register 0x2A. This is a fairly complicated procedure, so the entire program code to solve this task is provided below. Input the program to get a bit of a feel for using the I2C functions and work on improving the simplicity of an I2C task in future exercises.

```
var b = require("bonescript"); // Read BoneScript library
var port = '/dev/i2c-2';
var address = 0x1c;
b.i2copen(port, address, {}, onI2C); // Open I2C port
```

```

b.i2cwriteBytes(port, 0x2a, [0x00]); // Set STANDBY mode
b.i2cwriteBytes(port, 0x0e, [0x00]); // Set scale to 2G
b.i2cwriteBytes(port, 0x2a, [0x01]); // Set ACTIVE mode
setInterval(readA, 200); // Call readA() every 200ms

function onI2C() {
}

function readA() {
  b.i2cReadBytes(port, 1, 6, onReadBytes);
}

function onReadBytes(x) {
  if(x.event == 'callback') {
    var X = convertToG(x.res[0]); // First byte is X
    var Y = convertToG(x.res[2]); // Third byte is Y
    var Z = convertToG(x.res[4]); // Fifth byte is Z
    console.log('X: ' + X + ' Y: ' + Y + ' Z: ' + Z);
  }
}

function convertToG(x) {
  if(x >= 128) x = -((x^0xFF)+1); // Get two's complement
  x = x / 64; // Scale to G
  x = x.toFixed(2); // Limit decimal places
  return(x);
}

```

### Task #7 – Combine tasks 1, 2, 3, 5 and 6

Using the code you created for the previous tasks, combine them such that the red LED blinks, the blue LED reflects the status of the button, the potentiometer adjusts both the green LED brightness as well as the 7 segment display and the accelerometer values are being sampled. Then, adjust the duty cycle of red LED using the blink() function below such that the red LED only pulses for 10ms.

```

function blink() {
  state = !state;
  if(state) setTimeout(blink, 10); // On for 10ms
  if(!state) setTimeout(blink, 990); // off for 990ms
  b.digitalWrite(LED_RED, state ? b.HIGH : b.LOW);
}

```

Finally, eliminate the console.log() output and use the absolute value of the accelerometer X axis to set the brightness of the green LED, rather than the potentiometer.

```

var G = Math.abs(convertToG(x.res[0]));
if(G > 1) G = 1;
b.analogWrite(LED_GREEN, G);

```

## Task #8 – Use BoneScript within a web page to display potentiometer voltage

The BoneScript library includes a web server running by default on the BeagleBone Black and a remote procedure call map for every BoneScript function by using callbacks. Use the “Preview” button in Cloud9 IDE to serve up the below HTML page.

```
<html>
  <head>
    <title>Bacon Cape HTML Demo</title>
    <script src="http://192.168.7.2/static/jquery.js"></script>
    <script src="http://192.168.7.2/static/bonescript.js"></script>
  </head>
  <body>
    <h1>Bacon Cape HTML Demo</h1>
    <h2>sliderStatus = <span id="sliderStatus"></span></h2>
    <script>
      // Connect with board connected at fixed USB address and execute run() upon initialization
      setTargetAddress("192.168.7.2", { initialized: run });

      // The demo BoneScript application within the 'run()' function
      function run() {
        var b = require("bonescript"); // Read BoneScript library
        var POT = 'P9_36';
        setInterval(readPOT, 100);

        function readPOT() {
          // Fetch slider location using asynchronous callback function onAnalogRead()
          b.analogRead(POT, onAnalogRead);
        }

        // Handle data back from potentiometer
        function onAnalogRead(x) {
          if(!x.err) {
            sliderStatus = x.value.toFixed(2); // Limit number of decimal places
            $('#sliderStatus').html(sliderStatus); // jQuery command to update value
          }
        }
      }
    </script>
  </body>
</html>
```

## Task #9 – Edit task #8 code to display accelerometer values and button status

This is a tough one if you aren't previously familiar with callbacks. Every BoneScript function takes an optional callback as the last argument. When running remotely, a time-dependent call must use a callback to know when it completes. Also, extend the code to set the LEDs using the code from task #7.