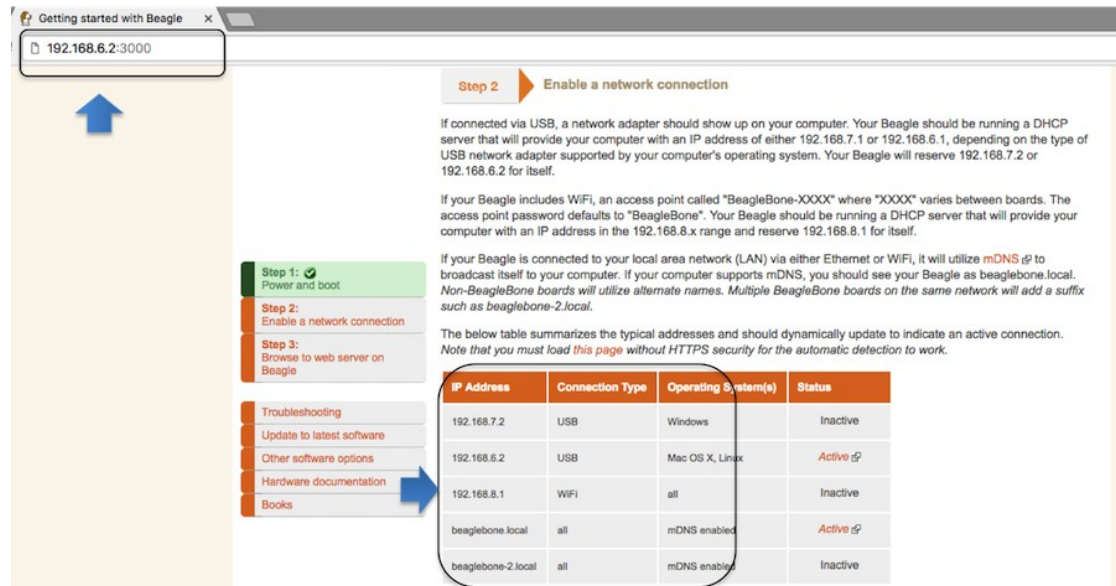


- MicroSD software image and other materials available from bbb.io/techlab

See bbb.io/start for instructions on using Etcher.io to write a microSD card



Getting started with Beagle

192.168.6.2:3000

Step 2: Enable a network connection

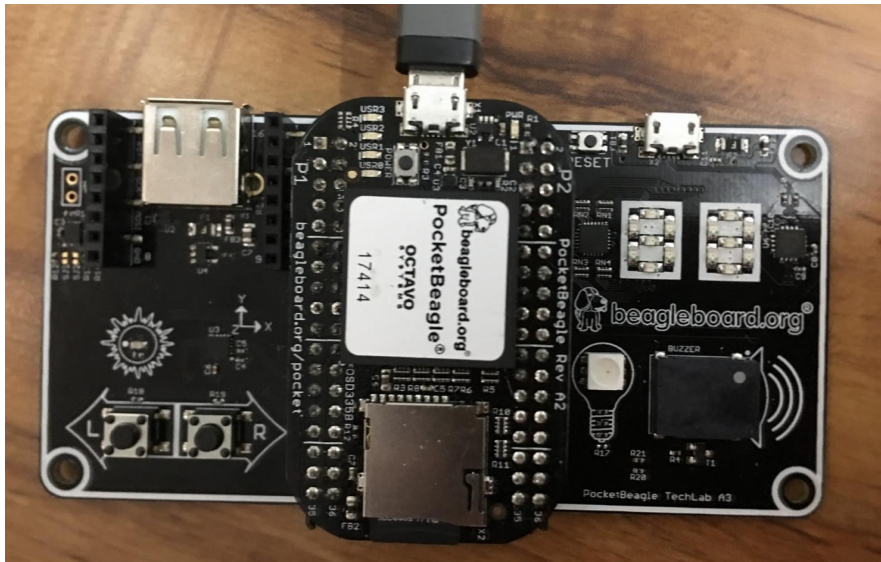
If connected via USB, a network adapter should show up on your computer. Your Beagle should be running a DHCP server that will provide your computer with an IP address of either 192.168.7.1 or 192.168.6.1, depending on the type of USB network adapter supported by your computer's operating system. Your Beagle will reserve 192.168.7.2 or 192.168.6.2 for itself.

If your Beagle includes WiFi, an access point called "BeagleBone-XXXX" where "XXXX" varies between boards. The access point password defaults to "BeagleBone". Your Beagle should be running a DHCP server that will provide your computer with an IP address in the 192.168.8.x range and reserve 192.168.8.1 for itself.

If your Beagle is connected to your local area network (LAN) via either Ethernet or WiFi, it will utilize mDNS to broadcast itself to your computer. If your computer supports mDNS, you should see your Beagle as beaglebone.local. Non-BeagleBone boards will utilize alternate names. Multiple BeagleBone boards on the same network will add a suffix such as beaglebone-2.local.

The below table summarizes the typical addresses and should dynamically update to indicate an active connection. Note that you must load this page without HTTPS security for the automatic detection to work.

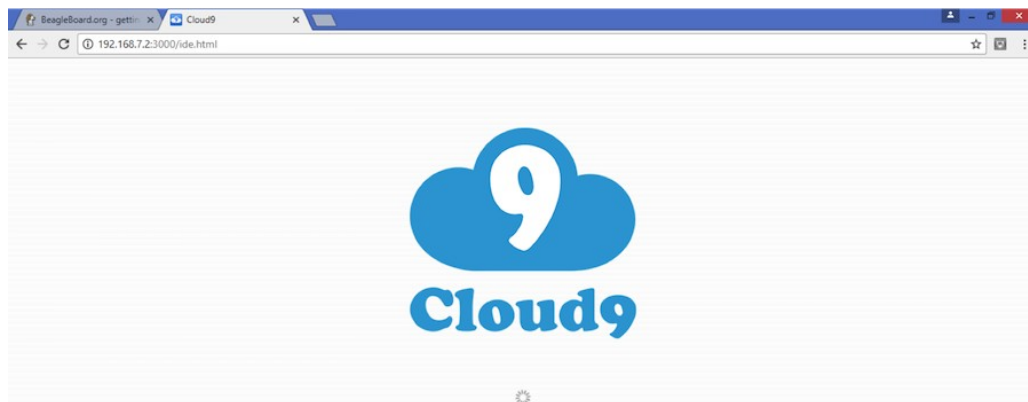
IP Address	Connection Type	Operating System(s)	Status
192.168.7.2	USB	Windows	Inactive
192.168.6.2	USB	Mac OS X, Linux	Active
192.168.8.1	WiFi	all	Inactive
beaglebone.local	all	mDNS enabled	Active
beaglebone-2.local	all	mDNS enabled	Inactive



Plug into the microUSB on PocketBeagle to provide power and a network connection. Look for the "heartbeat" pulse on the USR0 LED to know the board has Linux up-and-running.

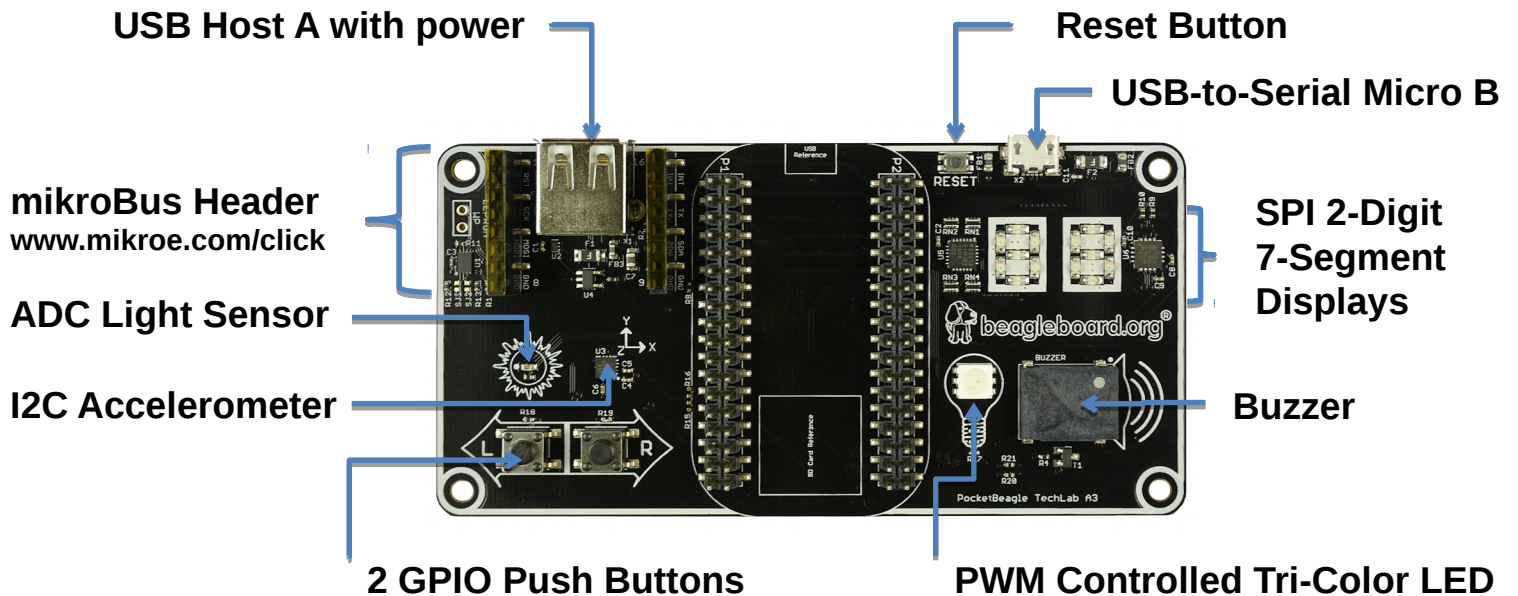
Get to the Cloud9 IDE

- Served on port 3000
- Windows: <http://192.168.7.2:3000>
- Linux/Mac: <http://192.168.6.2:3000>

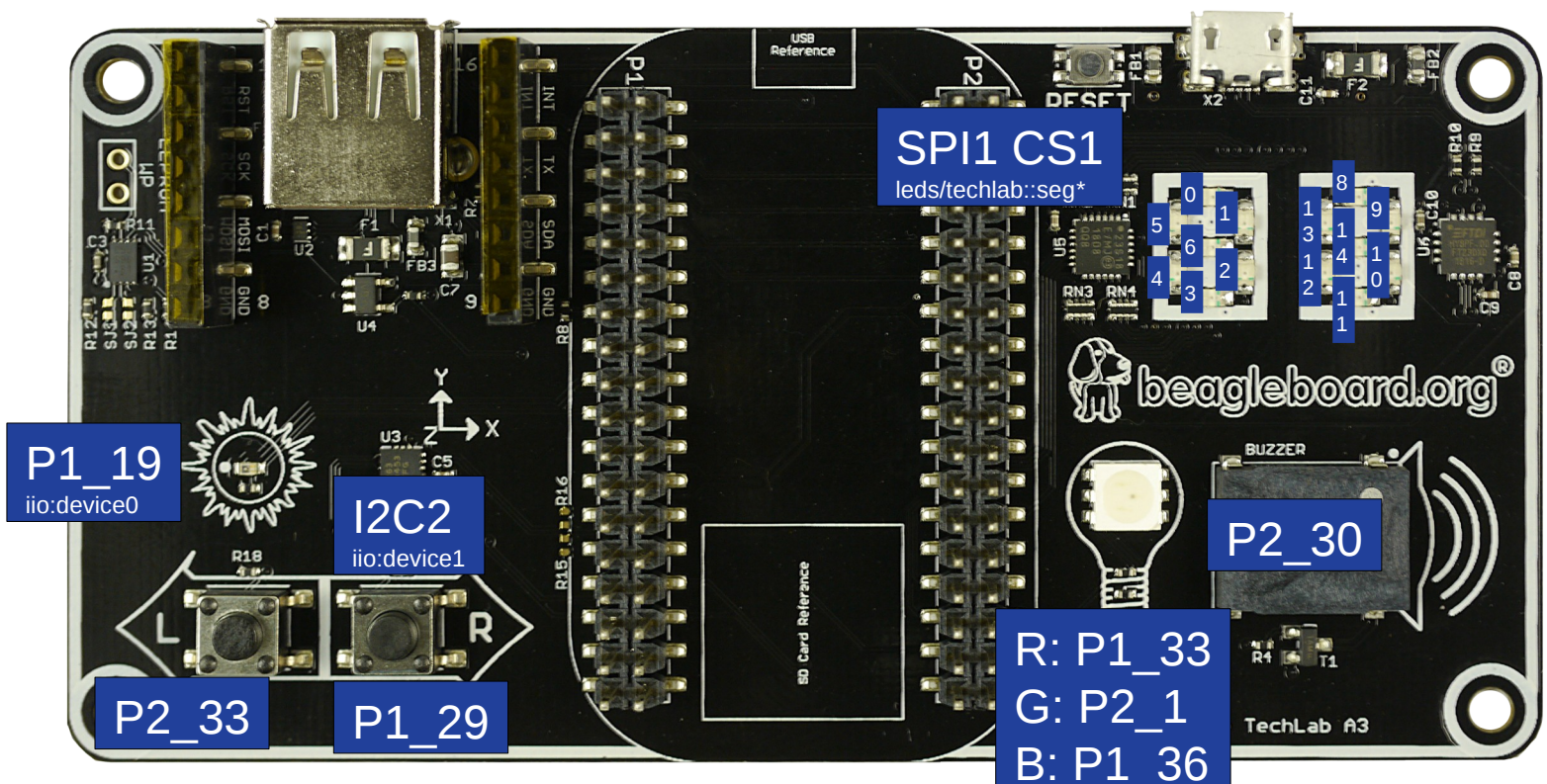


The **BeagleBoard.org Foundation** is a 501(c)(3) non-profit corporation existing to provide education in and collaboration around the design and use of open-source software and hardware in embedded computing.

TechLab Pocket Cape



TechLab Wiring Summary



PocketBeagle®



- PocketBeagle® is an ultra-tiny-yet-complete, low-cost, open-source USB-key-fob computer.
- Processor System:
Octavo Systems OSD3358-SM 21mm x 21mm system-in-package that includes 512MB DDR3 RAM, 1-GHz ARM Cortex-A8 CPU, 2x 200-MHz PRUs, ARM Cortex-M3, 3D accelerator, power/battery management and EEPROM
- 72 expansion pin headers with power and battery I/Os, high-speed USB, 8 analog inputs, 44 digital I/Os and numerous digital interface peripherals
- microUSB host/client and microSD connectors

PocketBeagle Expansion Header Pin-out

P1													
SYS		VIN	1	2	87			6	AIN 3.3V	9	PRU1		
USB1 V_EN		GPIO	109	3	4	89					11		
USB1			VBUS	5	6	5	GPIO	CS	SPI0	TX	PRU		
			VIN	7	8	2		CLK		RX	UART2		
			DN	9	10	3		MISO		TX			
			DP	11	12	4		MOSI		RX	PRU		
			ID	13	14	3.3V							
GND		15	16	GND	SYS								
AIN 1.8V			REF-	17	18	REF+	AIN 1.8V						
			0	19	20	20	GPIO			16(in)	PRU0		
			1	21	22	GND	SYS						
			2	23	24	VOUT	SYS						
			3	25	26	12	GPIO	SDA	I2C2	TX	CAN0		
4	27	28	13	SCL		RX							
PRU0	7	QEP0	STRB	A	GPIO	TX	UART0	15	PRU1				
	4					RX				14			
	1	PWM0	B			111	33	34	26				
PRU1	10				88	35	36	110	A	PWM0	0	PRU0	

P2													
PWM1		A	GPIO	50	1	2	59	GPIO					
PWM2		B		23	3	4	58						
UART4		RX		30	5	6	57						
		TX		31	7	8	60						
CAN1	RX	I2C1		SCL	15	9	10		52				
	TX		SDA	14	11	12	PWR BTN	SYS					
			SYS	VOUT	13	14	VIN	BAT					
				GND	15	16	TEMP						
			GPIO	65	17	18	47	GPIO	STRB	QEP2	15i	PRU0	
				27	19	20	64						
			SYS	GND	21	22	46	GPIO	IDX	QEP2	14(in)	PRU0	
				3.3V	23	24	44		A	QEP2	14(out)	PRU0	
CAN1	RX	SPI1	GPIO	41	25	26	NRST	SYS					
	TX			MISO	40	27	28	116	GPIO	IDX	QEP0	16	PRU0
PRU1	eCAP			CLK	7	29	30	113				3	
PRU1	16(in)			CS	19	31	32	112			2		
PRU0	15(out)	QEP2	B	45	33	34	115	GPIO	B	QEP0	15	PRU0	
PRU1	8	AIN 3.3V	5	86	35	36	7		AIN 1.8V				

Great getting started information is at beagleboard.org/pocket

Blink PocketBeagle on-board USRx LED

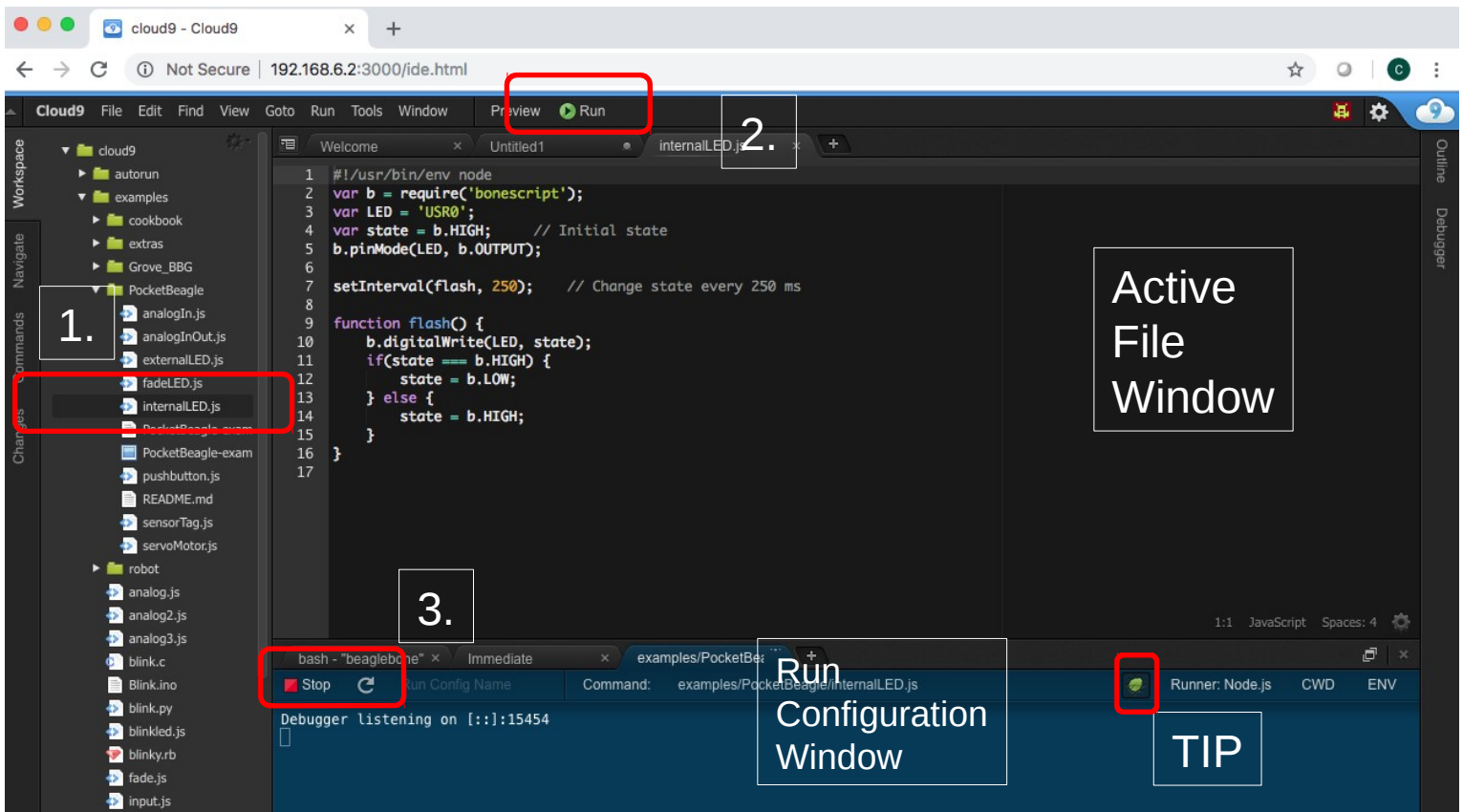
Goal: Blink USR3 LED on PocketBeagle.

Overview: BoneScript is a Node.js library customized for the Beagle family and featuring familiar Arduino function calls. Here we will use it to blink an LED built into your PocketBeagle.

Do this in the Cloud9 IDE:

1. Navigate to [TechLab/internalLED.js](#) and double-click on it.
2. Click the Run button in the toolbar to execute the script in the active file window
3. You will see the run configuration window open with a Stop button. Click the Stop button to halt the program.
4. Try changing the LED or blink time, save the program and run again.

TIP: Click the green bug to disable the debugger and begin execution quicker.



internalLED.js

```
#!/usr/bin/env node
var b = require('bonescript');
var LED = 'USR3';
var state = b.HIGH;    // Initial state
b.pinMode(LED, b.OUTPUT);

setInterval(flash, 250); // Change state every 250 ms

function flash() {
  b.digitalWrite(LED, state);
  if(state === b.HIGH) {
    state = b.LOW;
  } else {
    state = b.HIGH;
  }
}
```



Read a button

Goal: Sense the external world by reading a digital input.

Overview: Reading a switch attached to a GPIO (general purpose input/output) port is as easy as configuring the port as an input and attaching an interrupt handler to it. Note the buttons are “active low”.

Do this in the Cloud9 IDE:

1. Navigate to [TechLab/pushbutton.js](#) and double-click on it.
2. Click the Run button in the toolbar to execute the script in the active file window
3. Press the “L” button on TechLab and check the output (Value=1 or Value=0) in the configuration window.
Click the Stop button on the IDE to halt the program.

Challenge #1: Can you modify the program to read from the “R” button?

Challenge #2: Can you modify the program to toggle the USR3 LED?

Challenge #3: Can you modify the program to turn the USR3 LED on with the “L” button and off with the “R” button?

The screenshot shows the Cloud9 IDE interface. The top panel displays the file explorer with a tree view of the project structure. The main editor window shows the `pushbutton.js` file, which contains JavaScript code for reading a button state using the `bonescript` library. The code includes a `doAttach` function to configure the button and a `printStatus` function to handle the interrupt. The bottom panel shows the terminal output, which includes the command to run the script and the resulting output: `Hit ^C to stop`, `Interrupt handler attached`, and `Value = 1, err = undefined`.

```
#!/usr/bin/env node
var b = require('bonescript');
var button = "P2_3";

console.log('Hit ^C to stop');
b.pinMode(button, b.INPUT, 7, 'pulldown', 'fast', doAttach);

function doAttach(err, x) {
  if(err) {
    console.log('pinMode err = ' + err);
    return;
  }
  b.attachInterrupt(button, true, b.CHANGE, printStatus);
}

function printStatus(err, x) {
  if(err) {
    console.log('attachInterrupt err = ' + err);
    return;
  }
  if(x.attached) {
    console.log("Interrupt handler attached");
    return;
  }
  process.stdout.write('value = ' + x.value + ', err = ' + err + '\n');
}
```

bash - "beaglebone" x Immediate x [New] - Idle x examples/PocketBeagle...
Stop Run Config Name Command: examples/PocketBeagle/pushbutton.js Runner: Node.js CWD ENV
Debugger listening on (::):15454
Hit ^C to stop
Interrupt handler attached
Value = 1, err = undefined

Read a button

pushbutton.js

```
#!/usr/bin/env node
var b = require('bonescript');
var button = "P2_33";

console.log('Hit ^C to stop');
b.pinMode(button, b.INPUT, 7, null, null, doAttach);

function doAttach(err, x) {
  if(err) {
    console.log('pinMode err = ' + err);
    return;
  }
  b.attachInterrupt(button, true, b.CHANGE, printStatus);
}

function printStatus(err, x) {
  if(err) {
    console.log('attachInterrupt err = ' + err);
    return;
  }
  if(x.attached) {
    console.log("Interrupt handler attached");
    return;
  }
  process.stdout.write('value = ' + x.value + '      \r');
}
```


Read an analog sensor

Goal: Sense the external world by reading a variable analog input

Overview: Reading a light sensor attached to an analog input pin.

Do this in the Cloud9 IDE:

1. Navigate to [TechLab/analogIn.js](#) and double-click on it.
2. Click the Run button in the toolbar to execute the script in the active file window
3. Cover the light sensor and check the output in the configuration window. Click the Stop button to halt the program.

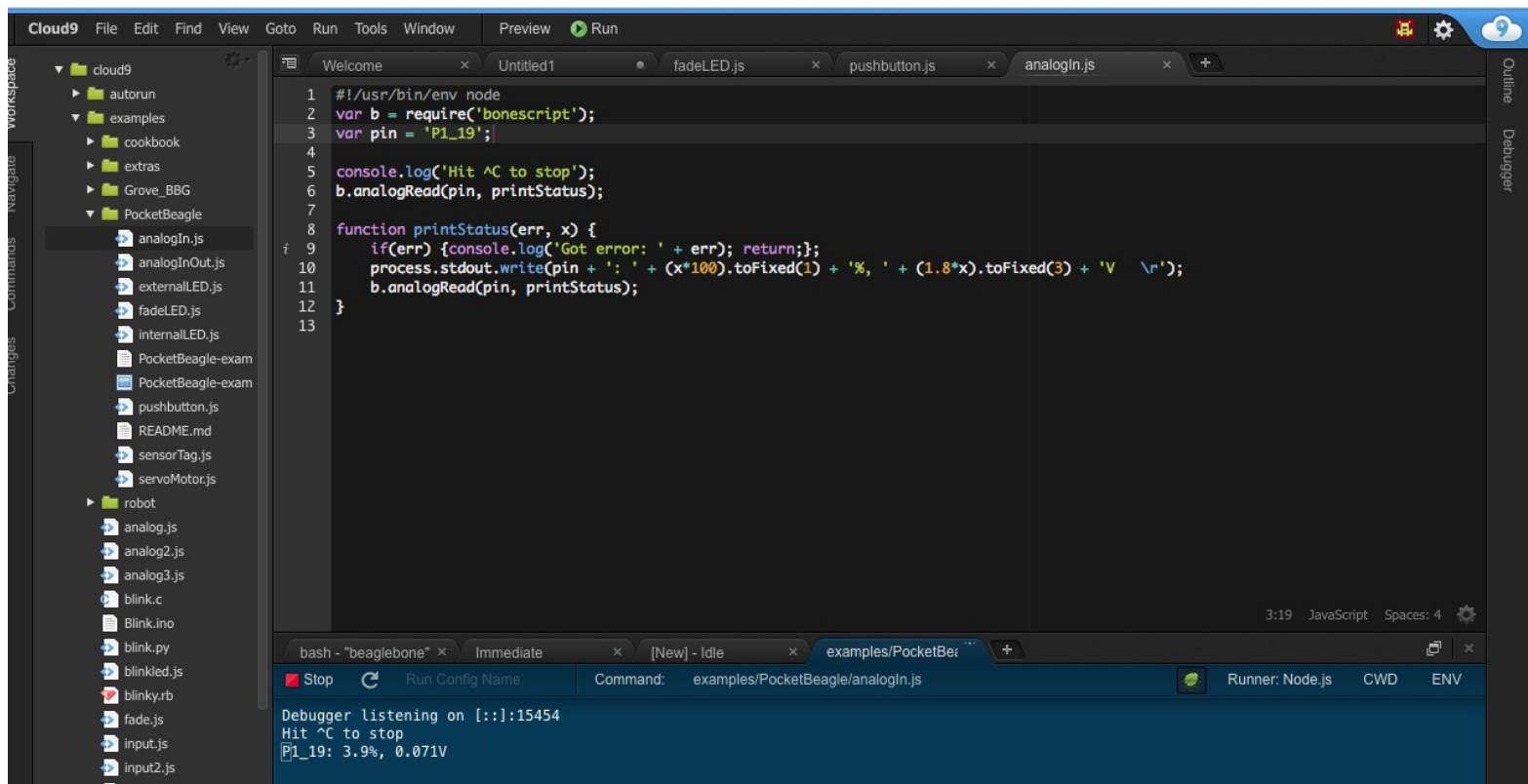
Challenge #1: Can you change how often the light sensor is read? What happens and why?

Challenge #2: Can you activate the USR3 LED based upon a voltage threshold from the light sensor?

Challenge #3: (Advanced Coding Lab)

Try using the I2C accelerometer input from `/sys/bus/iio/devices/iio:device1/in_accel_x_raw`.

Hint: use `b.readTextFile()`



The screenshot shows the Cloud9 IDE interface. On the left, a file explorer displays a project structure with folders like 'cloud9', 'examples', 'cookbook', 'extras', 'Grove_BBG', and 'PocketBeagle'. The 'analogIn.js' file is selected. The main editor window shows the code for 'analogIn.js':

```
1 #!/usr/bin/env node
2 var b = require('bonescript');
3 var pin = 'P1_19';
4
5 console.log('Hit ^C to stop');
6 b.analogRead(pin, printStatus);
7
8 function printStatus(err, x) {
9   if(err) {console.log('Got error: ' + err); return;};
10  process.stdout.write(pin + ': ' + (x*100).toFixed(1) + '%, ' + (1.8*x).toFixed(3) + 'V \r');
11  b.analogRead(pin, printStatus);
12 }
13
```

At the bottom, the 'Immediate' window shows the command 'examples/PocketBeagle/analogIn.js' and the output of the program:

```
Debugger listening on [::]:15454
Hit ^C to stop
P1_19: 3.9%, 0.071V
```

Read an analog sensor

analogIn.js

```
#!/usr/bin/env node
var b = require('bonescript');
var pin = 'P1_19';

console.log('Hit ^C to stop');
doAnalogRead();

function printStatus(err, x) {
  if(err) {console.log('Got error: ' + err); return;};
  process.stdout.write(pin + ': ' + (x*100).toFixed(1) +
    '%, ' + (1.8*x).toFixed(3) + 'V  \r');
  setTimeout(doAnalogRead, 100);
}

function doAnalogRead() {
  b.analogRead(pin, printStatus);
}
```


Fade an LED

Goal: Utilize a hardware pulse-width-modulator (PWM) to light an LED with variable brightness

Overview: Linux provides LED drivers that understand how to utilize PWM drivers, making use of PocketBeagle's built-in PWM hardware. They are controlled with simple text files where you can set the brightness.

Do this in the Cloud9 IDE:

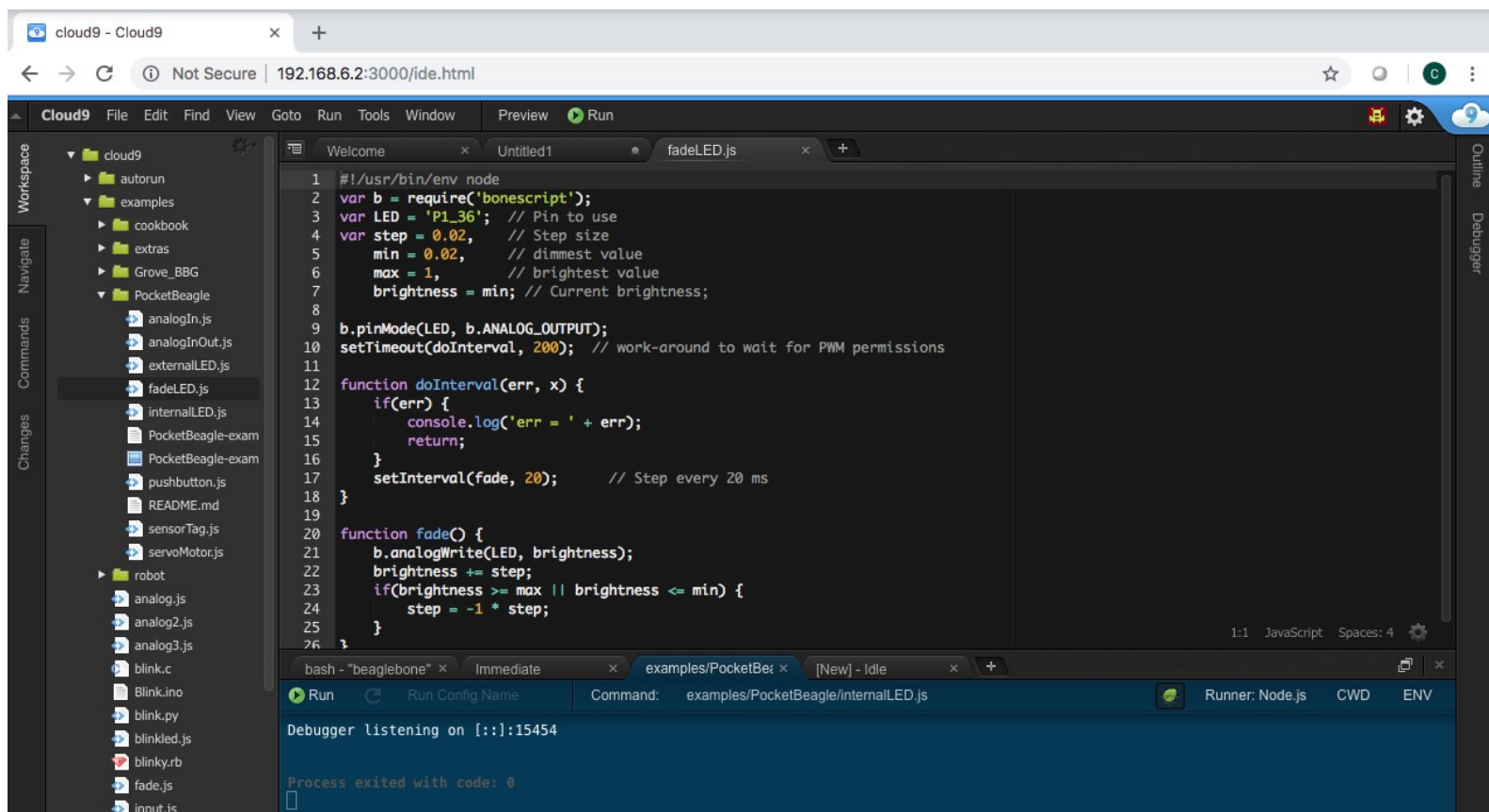
1. Navigate to [TechLab/fadeLED.js](#) and double-click on it.
2. Click the Run button in the toolbar to execute the script in the active file window
3. You will see the run configuration window open with a Stop button. Click the Stop button to halt the program.

Challenge #1: Try changing the fade interval, save the program and run again.

Challenge #2: Try using the light sensor input to set the LED brightness.

Challenge #3: (Advanced Coding Lab)

Try using the I2C accelerometer input for all 3 color LEDs..



Fade an LED

fadeLED.js

```
#!/usr/bin/env node
var b = require('bonescript');
var LED = '/sys/class/leds/techlab::blue/brightness';
var step = 10,          // Step size
    min = 0,            // dimmest value
    max = 255,          // brightest value
    brightness = min;   // Current brightness;

doInterval();

function doInterval(err, x) {
    if(err) {
        console.log('err = ' + err);
        return;
    }
    setInterval(fade, 20);    // Step every 20 ms
}

function fade() {
    b.writeTextFile(LED, brightness);
    brightness += step;
    if(brightness >= max || brightness <= min) {
        step = -1 * step;
    }
}
```

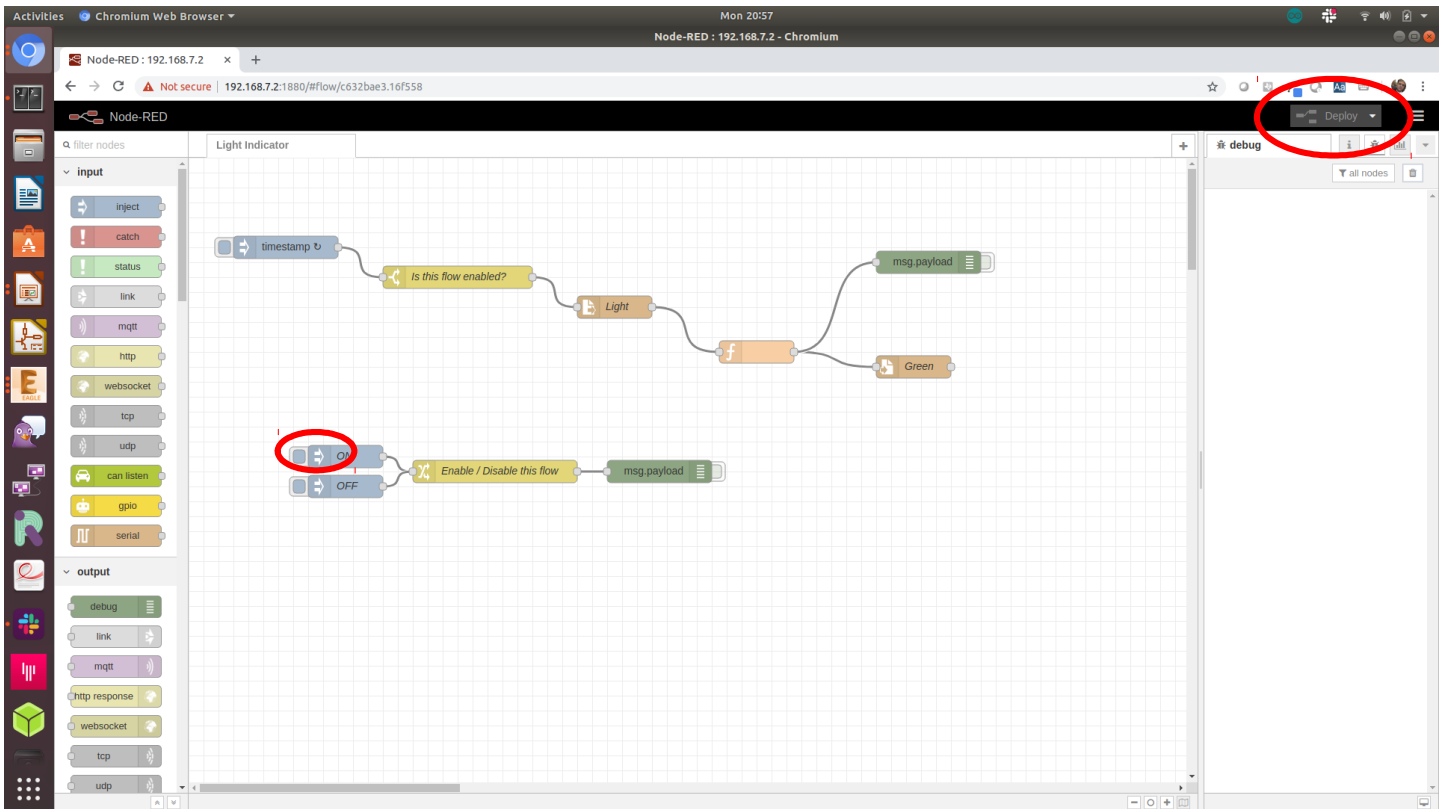
Using Node-RED to read and write files

Goal: Read light sensor data and output to green LED brightness

Overview: Node-RED is a flow-based development tool developed originally by IBM for wiring together hardware devices, APIs and online services as part of the Internet of Things. Node-RED provides a browser-based flow editor, which can be used to create JavaScript functions. Linux turns devices into virtual files, making Node-RED well suited to interacting with the physical world.

Do this:

1. Open Node-RED by pointing your browser to <http://192.168.7.2:1880>



2. Make sure the big “DEPLOY” button in the top right corner is greyed-out by clicking it. This makes sure any changes you’ve made have been started on your PocketBeagle. The program will run continuously.

3. Click the highlightable button to the left of the “ON”. Cover the light sensor to notice the brightness of the green LED adjust.

4. Try double-clicking on each node to see the parameters used for the demo.

5. Click the highlightable button to the left of the “OFF” to stop adjusting the brightness of the green LED.

6. Explore

Challenge #1: Try updating the blue LED rather than the green LED. Remember to click the “DEPLOY” button to save and run your changes.

Challenge #2: Try reading from the I2C accelerometer rather than the light sensor.

Challenge #3: Use a “gpio in” node to use the status of the “L” or “R” buttons to update the LED.

Explore the Linux command line

Goal: Learn to send several basic commands to the shell

Overview: The true power of Linux to automate many aspects of your life cannot be achieved without some utilization of the command line shell. The Cloud9 IDE makes it easy to access this directly from your web browser. Another great resource for learning is linuxcommand.org.

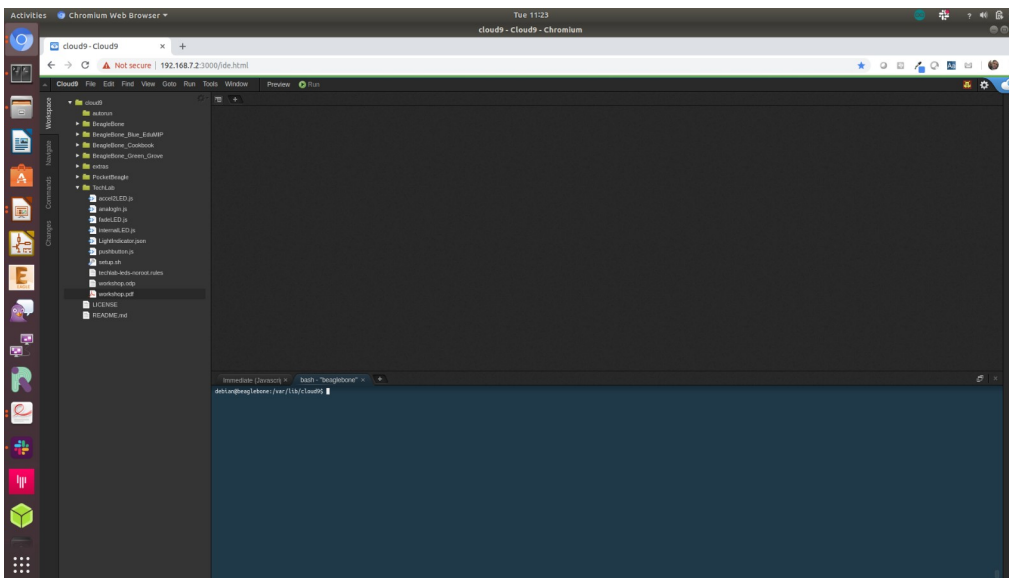
Do this in the Cloud9 IDE:

1. Click in the terminal window at the bottom half of the IDE. You can also open a new terminal by clicking the “+” in the window tabs and selecting “New Terminal”.
2. Try typing in the “commands to try”. Press <ENTER> after each command. Take note of how the prompt changes to show you the current active directory. Use the up and down arrows to cycle through commands you’ve typed before, in case you want to repeat any.

Challenge #1: Can you repeat the last command below that prints GPIO values and change the values?

Challenge #2: Using the “watch” command, can you monitor the I2C accelerometer status?

Challenge #3: Can you use the “config-pin” command to switch the red LED to a GPIO, set it high and low and then switch it back to PWM mode? What happens if the red LED is in GPIO mode and you change the brightness with the “/sys/class/leds” entry?



commands to try

```
cd /sys/class/leds
ls
echo 1 > techlab\:\:seg0/brightness
config-pin p1.33 pwm
echo 10 > techlab\:\:red/brightness

cd /sys/class/gpio
config-pin p1.29 gpio
cat gpio45/value gpio117/value
```

Some useful commands

- pwd - show current directory
- cd - change current directory
- ls - list directory contents
- chmod - change file permissions
- chown - change file ownership
- cp - copy files
- mv - move files
- rm - remove files
- mkdir - make directory
- rmdir - remove directory
- cat - dump file contents
- less - progressively dump file
- vi - edit file (complex)
- nano - edit file (simple)
- head - trim dump to top
- tail - trim dump to bottom
- echo - print/dump value
- env - dump environment variables
- export - set environment variable
- history - dump command history
- grep - search dump for strings
- man - get help on command
- apropos - show list of man pages
- find - search for files
- tar - create/extract file archives
- gzip - compress a file
- gunzip - decompress a file
- du - show disk usage
- df - show disk free space
- mount - mount disks
- tee - write dump to file in parallel
- hexdump - readable binary dumps

Toggle LED based on a button press using a PRU

Goal: Utilize a programmable real-time unit (PRU) to read and set GPIOs.

Overview: A PRU is a microcontroller built into the PocketBeagle that can be programmed in C or assembly and controlled by the main ARM processor that runs Linux. Some pins can be directly mapped to PRU registered for the lowest possible latency, but other GPIOs can be utilized via the on-chip-peripheral (OCP) bus inside the processor. Here we'll learn to execute an example written in C to access the direct-mapped PRU GPIOs.

Do this in the Cloud9 IDE:

1. Navigate to [TechLab/led-button.pru0c](#) and double-click on it.
2. Click the Run button in the toolbar to execute the script in the active file window.
3. Click the run configuration window and type 'temppwd' when prompted for the password.

Note: The program will keep running until you execute a different program or reboot.

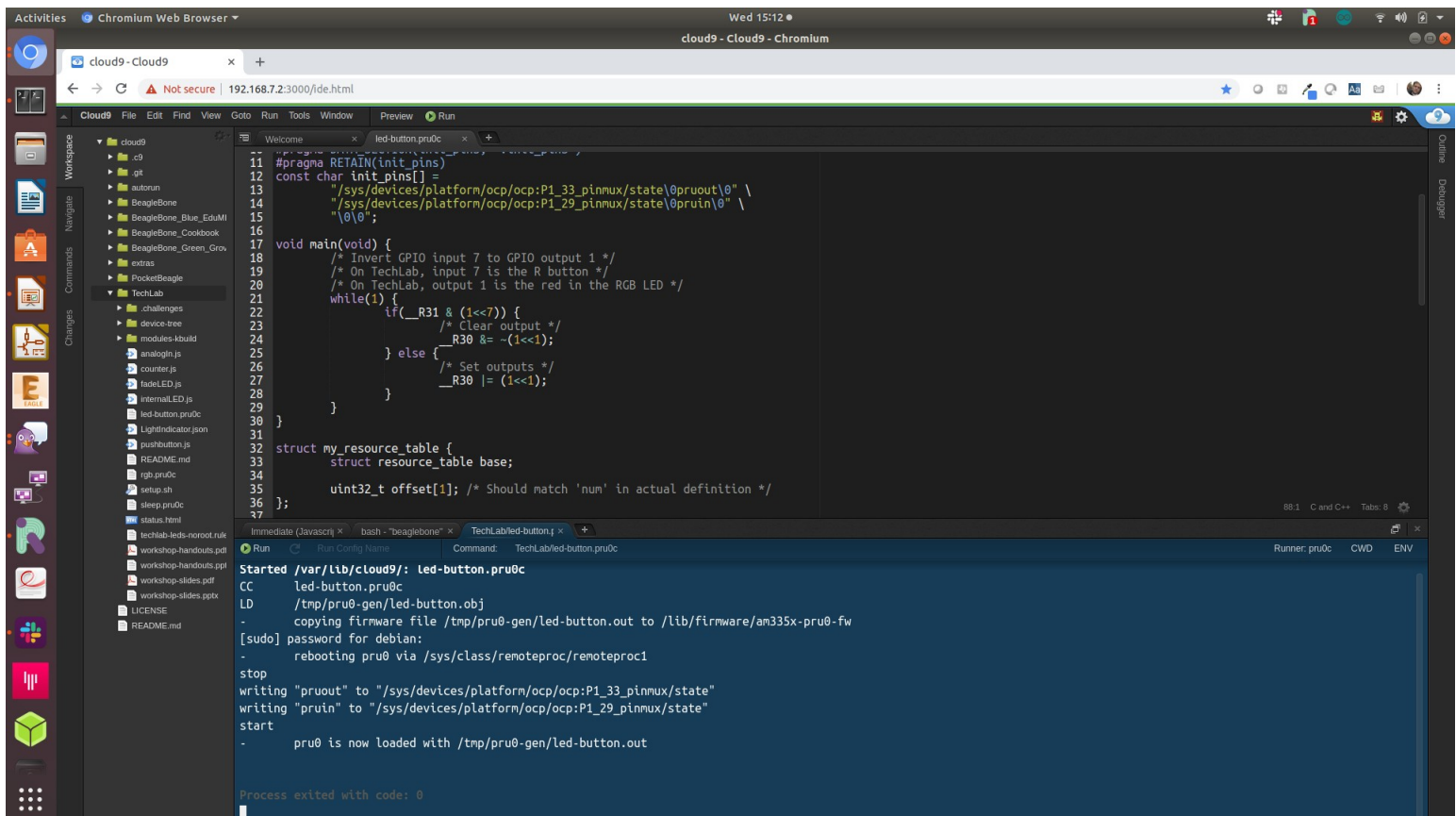
4. Press the 'R' button to note the red RGB LED light up.

Challenge #1: The buzzer is on PRU0 output 3 (P2_30). Try setting its output at the same time as the LED.

Challenge #2: Try toggling the buzzer with a for loop when the button is pressed. You can use the '`__delay_cycles`' function to add a delay. Each delay cycle should be 5ns, so 800Hz is 250000 cycles, a reasonable number to try.

Challenge #3: (Advanced Coding Lab)

Run [TechLab/rgb.pru0c](#) and note how the on-chip-peripherals are accessed as well as how a fade effect can be done easily creating a software pulse-width-modulator.



```
11 #pragma RETAIN(init_pins)
12 const char init_pins[] =
13     "/sys/devices/platform/ocp/ocp:P1_33_pinmux/state@pruout0" \
14     "/sys/devices/platform/ocp/ocp:P1_29_pinmux/state@pruin0" \
15     "\0";
16
17 void main(void) {
18     /* Invert GPIO input 7 to GPIO output 1 */
19     /* On TechLab, input 7 is the R button */
20     /* On TechLab, output 1 is the red in the RGB LED */
21     while(1) {
22         if(_R31 & (1<<7)) {
23             /* Clear output */
24             _R30 &= ~(1<<1);
25         } else {
26             /* Set outputs */
27             _R30 |= (1<<1);
28         }
29     }
30 }
31
32 struct my_resource_table {
33     struct resource_table base;
34     uint32_t offset[1]; /* Should match 'num' in actual definition */
35 };
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
Started /var/lib/cloud9/: led-button.pru0c
CC      /tmp/pru0-gen/led-button.o
LD      /tmp/pru0-gen/led-button.out
- copying firmware file /tmp/pru0-gen/led-button.out to /lib/firmware/am335x-pru0-fw
[sudo] password for debian:
- rebooting pru0 via /sys/class/remoteproc/remoteproc1
stop
writing "pruout" to "/sys/devices/platform/ocp/ocp:P1_33_pinmux/state"
writing "pruin" to "/sys/devices/platform/ocp/ocp:P1_29_pinmux/state"
start
- pru0 is now loaded with /tmp/pru0-gen/led-button.out

Process exited with code: 0
```

Fade LEDs using a PRU

led-button.pru0c

```
#include <stdint.h>
#include <pru_cfg.h>
#include <pru_ctrl.h>
#include <stddef.h>
#include <rsc_types.h>

volatile register unsigned int __R30;
volatile register unsigned int __R31;

#pragma DATA_SECTION(init_pins, ".init_pins")
#pragma RETAIN(init_pins)
const char init_pins[] =
    "/sys/devices/platform/ocp/ocp:P1_33_pinmux/state\0pruout\0" \
    "/sys/devices/platform/ocp/ocp:P1_29_pinmux/state\0pruin\0" \
    "\0\0";

void main(void) {
    /* Invert GPIO input 7 to GPIO output 1 */
    /* On TechLab, input 7 is the R button */
    /* On TechLab, output 1 is the red in the RGB LED */
    while(1) {
        if(__R31 & (1<<7)) {
            /* Clear output */
            __R30 &= ~(1<<1);
        } else {
            /* Set outputs */
            __R30 |= (1<<1);
        }
    }
}

struct my_resource_table {
    struct resource_table base;

    uint32_t offset[1];
};

#pragma DATA_SECTION(pru_remoteproc_ResourceTable, ".resource_table")
#pragma RETAIN(pru_remoteproc_ResourceTable)
struct my_resource_table pru_remoteproc_ResourceTable = {
    1, /* we're the first version that implements this */
    0, /* number of entries in the table */
    0, 0, /* reserved, must be zero */
    0, /* offset[0] */
};
```