

EECS 4221: Operating System Design
Winter 2019

Alternative Project 1: BeagleBone
Black and Angstrom Linux

Niruyan Rakulan
214343438
Electrical Engineering Undergraduate

Table of Contents

Table of Contents	1
A. Review: Important Features of BeagleBone Black and Angstrom	2
Most Important Features of BeagleBone Black	2
Most Important Features of Angstrom	5
Linux Kernel Modules (LKM)	6
How To Use LKMs in Linux	7
B. Setting Up The Experiments	8
Steps to Install Angstrom On To The BeagleBone Black	8
Steps to Setup BeagleBone Black	8
Board Safety	8
Connecting To The Internet and Correcting Time	8
C. Creating Software in User Space and in Kernel Mode (LKMs)	11
User Space	11
Hello World in C	12
7-Segment Display That Shows The Current Time Using Python	12
Kernel Mode	13
helloworld.c LKM Driver	13
Building Linux Kernel Modules: Hello World	13
helloworld.c	13
Makefile	14
helloworld.c LKM (with Parameter) Driver	16
helloworld.c (with param)	17
Controlling LED in Kernel Mode using a "Button" Driver	18
controled.c	20
Controlling 7-Seg Driver	23
sevenseg.c	24
Counting the Number of Seconds LED is Lit	29
counter.c	29
D. Final Application (Duty Ratio Detector)	38
Purpose of the Application	38
User Mode (PWM Generator and Controller)	38
pwm_gen.py	40
Kernel Mode (PWM Duty Ratio Detector)	41
pwm_detect.c	41
Results	47
Conclusion	50
Troubles	50
References	50

A. Review: Important Features of BeagleBone Black and Angstrom

Most Important Features of BeagleBone Black

The BeagleBone Black Board is a relatively low-cost development board. It features a Texas Instruments Sitara AM335x Cortex A8 ARM microprocessor (running at 1GHz), 512MB of DDR3 ram, 4KB of EEPROM, and 4GB of eMMC storage(Rev C). The board features 92 Pins(46 on header P8 and 46 on header P9), many of which are GPIOs allowing to be interfaced with sensors (ie. photoresistor) and actuators (ie. motor); the modes of the pins can be changed to serve other purposes such as ADC, and PWM.

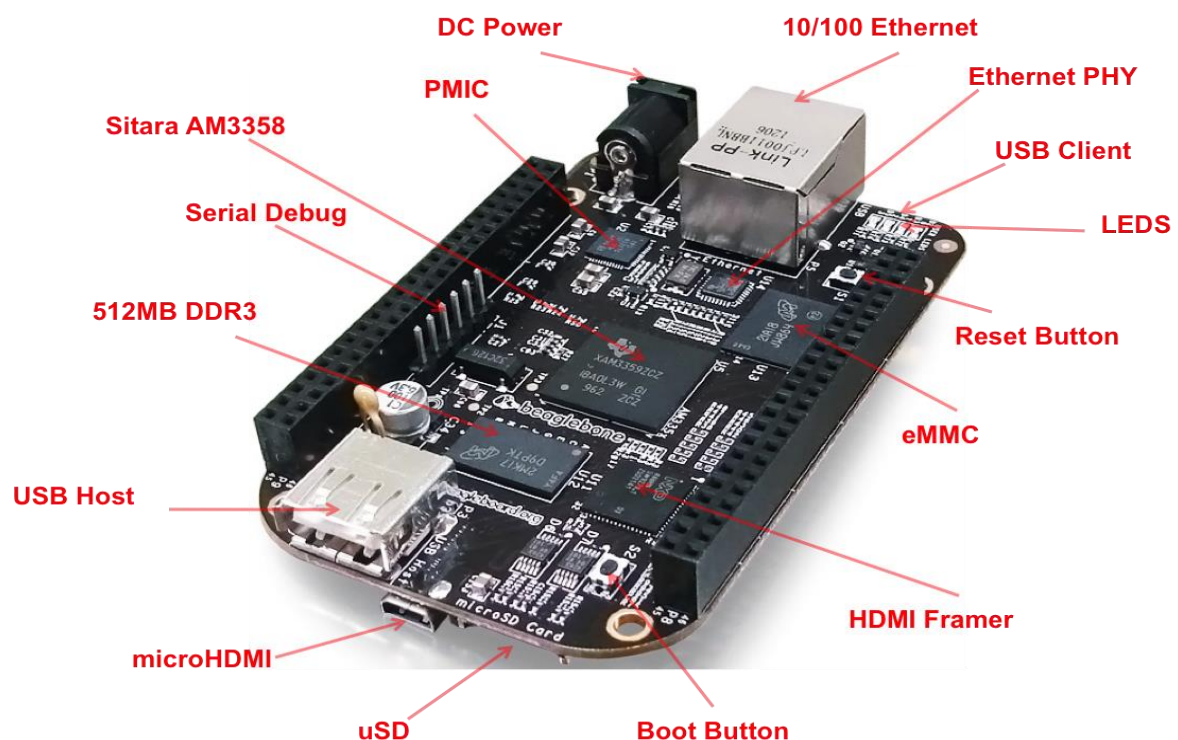


Figure 1: BeagleBone Black board layout. From <http://beagleboard.org/Support/bone101>

The BeagleBone can either be interfaced thru a tethered computer, or as a standalone desktop (with the board attached to a monitor thru a HDMI cable); this report will use the tethered approach to a Windows 10 computer. The tethered approach powers the board, as well as allows the user to use their PC to interface with the board.

	Feature	
Processor	Sitara AM3358BZCZ100	
Graphics Engine	1GHz, 2000 MIPS	
SDRAM Memory	SGX530 3D, 20M Polygons/S	
Onboard Flash	512MB DDR3L 800MHZ	
PMIC	2GB, 8bit Embedded MMC	
Debug Support	TPS65217C PMIC regulator and one additional LDO.	
Power Source	Optional Onboard 20-pin CTI JTAG, Serial Header	
PCB	miniUSB USB or DC Jack	5VDC External Via Expansion Header
Indicators	3.4" x 2.1"	6 layers
HS USB 2.0 Client Port	1-Power, 2-Ethernet, 4-User Controllable LEDs	
HS USB 2.0 Host Port	Access to USB0, Client mode via miniUSB	
Serial Port	Access to USB1, Type A Socket, 500mA LS/FS/HS	
Ethernet	UART0 access via 6 pin 3.3V TTL Header. Header is populated	
SD/MMC Connector	10/100, RJ45	
User Input	microSD , 3.3V	
Video Out	Reset Button Boot Button Power Button	
Audio	16b HDMI, 1280x1024 (MAX) 1024x768,1280x720,1440x900 ,1920x1080@24Hz w/EDID Support	
Expansion Connectors	Via HDMI Interface, Stereo	
Weight	Power 5V, 3.3V , VDD_ADC(1.8V) 3.3V I/O on all signals McASP0, SPI1, I2C, GPIO(69 max), LCD, GPMC, MMC1, MMC2, 7 AIN(1.8V MAX), 4 Timers, 4 Serial Ports, CAN0, EHRPWM(0,2),XDMA Interrupt, Power button, Expansion Board ID (Up to 4 can be stacked)	
Power	1.4 oz (39.68 grams)	
	Refer to Section 6.1.7	

Figure 2: BeagleBone Black specs. From <https://media.digikey.com/pdf/Data%20Sheets/Circuitco%20Elect/BB-BBLK-000%20Manual.pdf>

Cape Expansion Headers

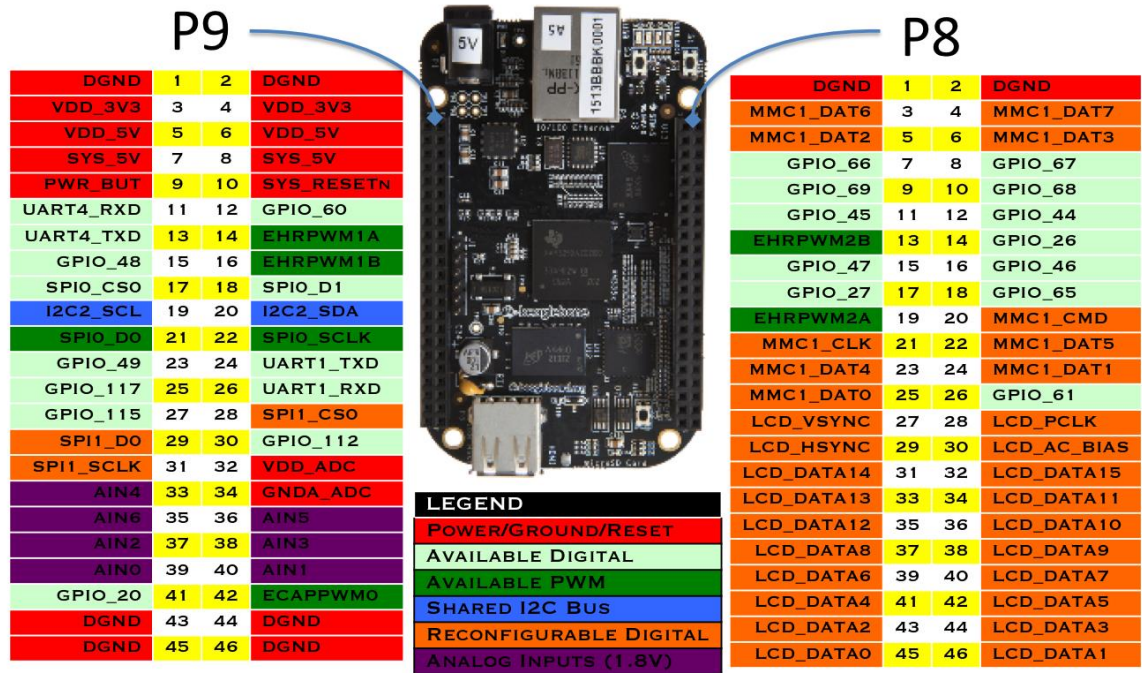


Figure 3: Pin layout of board. From <http://beagleboard.org/Support/bone101>

PIN	PROC	NAME	MODE0	MODE1	MODE2	MODE3	MODE4	MODE5	MODE6	MODE7
1										
2										
3	R9	GPIO1_6	gpmc_ad6	mmc1_dat6						gpio1[6]
4	T9	GPIO1_7	gpmc_ad7	mmc1_dat7						gpio1[7]
5	R8	GPIO1_2	gpmc_ad2	mmc1_dat2						gpio1[2]
6	T8	GPIO1_3	gpmc_ad3	mmc1_dat3						gpio1[3]
7	R7	TIMER4	gpmc_advn_ale		timer4					gpio2[2]
8	T7	TIMER7	gpmc_oen_ren		timer7					gpio2[3]
9	T6	TIMER5	gpmc_be0n_cle		timer5					gpio2[5]
10	U6	TIMER6	gpmc_wen		timer6					gpio2[4]
11	R12	GPIO1_13	gpmc_ad13	lcd_data18	mmc1_dat5	mmc2_dat1	eQEP2B_in		pr1_pru0_pru_r30_15	gpio1[13]
12	T12	GPIO1_12	gpmc_ad12	lcd_data19	mmc1_dat4	mmc2_dat0	Eqep2a_in		pr1_pru0_pru_r30_14	gpio1[12]
13	T10	EHRPWM2B	gpmc_ad9	lcd_data22	mmc1_dat1	mmc2_dat5	ehrpwm2B			gpio0[23]
14	T11	GPIO0_26	gpmc_ad10	lcd_data21	mmc1_dat2	mmc2_dat6	ehrpwm2_tripzone_in			gpio0[26]
15	U13	GPIO1_15	gpmc_ad15	lcd_data16	mmc1_dat7	mmc2_dat3	eQEP2_strobe		pr1_pru0_pru_r31_15	gpio1[15]
16	V13	GPIO1_14	gpmc_ad14	lcd_data17	mmc1_dat6	mmc2_dat2	eQEP2_index		pr1_pru0_pru_r31_14	gpio1[14]
17	U12	GPIO0_27	gpmc_ad11	lcd_data20	mmc1_dat3	mmc2_dat7	ehrpwm0_synco			gpio2[27]
18	V12	GPIO2_1	gpmc_clk_mux0	lcd_memory_clk	gpmc_wait1	mmc2_clk			mcsap0_fsr	gpio2[1]
19	U10	EHRPWM2A	gpmc_ad8	lcd_data23	mmc1_dat0	mmc2_dat4	ehrpwm2A			gpio2[22]
20	V9	GPIO1_31	gpmc_csn2	gpmc_be1n	mmc1_cmd			pr1_pru1_pru_r30_13	pr1_pru1_pru_r31_13	gpio1[31]
21	U9	GPIO1_30	gpmc_csn1	gpmc_clk	mmc1_clk			pr1_pru1_pru_r30_12	pr1_pru1_pru_r31_12	gpio1[30]
22	V8	GPIO1_5	gpmc_ad5	mmc1_dat5						gpio1[5]
23	U8	GPIO1_4	gpmc_ad4	mmc1_dat4						gpio1[4]
24	V7	GPIO1_1	gpmc_ad1	mmc1_dat1						gpio1[1]
25	U7	GPIO1_0	gpmc_ad0	mmc1_dat0						gpio1[0]
26	V6	GPIO1_29	gpmc_csn0							gpio1[29]
27	U5	GPIO2_22	lcd_vsync	gpmc_a8				pr1_pru1_pru_r30_8	pr1_pru1_pru_r31_8	gpio2[22]
28	V5	GPIO2_24	lcd_pclk	gpmc_a10				pr1_pru1_pru_r30_10	pr1_pru1_pru_r31_10	gpio2[24]
29	R5	GPIO2_23	lcd_hsync	gpmc_a9				pr1_pru1_pru_r30_9	pr1_pru1_pru_r31_9	gpio2[23]
30	R6	GPIO2_25	lcd_ac_bias_en	gpmc_a11						gpio2[25]
31	V4	UART5_CTSN	lcd_data14	gpmc_a18	eQEP1_index	mcsap0_axr1	uart5_rxd		uart5_ctsn	gpio0[10]
32	T5	UART5_RTSN	lcd_data15	gpmc_a19	eQEP1_strobe	mcsap0_ahclkx	mcsap0_axr3		uart5_rtsn	gpio0[11]
33	V3	UART4_RTSN	lcd_data13	gpmc_a17	eQEP1B_in	mcsap0_fsr	mcsap0_axr3		uart4_rtsn	gpio0[9]
34	U4	UART3_RTSN	lcd_data11	gpmc_a15	ehrpwm1B	mcsap0_ahclkx	mcsap0_axr2		uart3_rtsn	gpio2[17]
35	V2	UART4_CTSN	lcd_data12	gpmc_a16	eQEP1A_in	mcsap0_acklr	mcsap0_axr2		uart4_ctsn	gpio0[8]
36	U3	UART3_CTSN	lcd_data10	gpmc_a14	ehrpwm1A	mcsap0_axr0			uart3_ctsn	gpio2[16]
37	U1	UART5_TXD	lcd_data8	gpmc_a12	ehrpwm1_tripzone_in	mcsap0_acklx	uart5_bxd		uart2_ctsn	gpio2[14]
38	U2	UART5_RXD	lcd_data9	gpmc_a13	ehrpwm0_synco	mcsap0_fsx	uart5_rxd		uart2_rtsn	gpio2[15]
39	T3	GPIO2_12	lcd_data6	gpmc_a6		eQEP2_index		pr1_pru1_pru_r30_6	pr1_pru1_pru_r31_6	gpio2[12]
40	T4	GPIO2_13	lcd_data7	gpmc_a7		eQEP2_strobe	pr1_edio_data_out7	pr1_pru1_pru_r30_7	pr1_pru1_pru_r31_7	gpio2[13]
41	T1	GPIO2_10	lcd_data4	gpmc_a4		eQEP2A_in		pr1_pru1_pru_r30_4	pr1_pru1_pru_r31_4	gpio2[10]
42	T2	GPIO2_11	lcd_data5	gpmc_a5		eQEP2B_in		pr1_pru1_pru_r30_5	pr1_pru1_pru_r31_5	gpio2[11]
43	R3	GPIO2_8	lcd_data2	gpmc_a2		ehrpwm2_tripzone_in		pr1_pru1_pru_r30_2	pr1_pru1_pru_r31_2	gpio2[8]
44	R4	GPIO2_9	lcd_data3	gpmc_a3		ehrpwm0_synco		pr1_pru1_pru_r30_3	pr1_pru1_pru_r31_3	gpio2[9]
45	R1	GPIO2_6	lcd_data0	gpmc_a0		ehrpwm2A		pr1_pru1_pru_r30_0	pr1_pru1_pru_r31_0	gpio2[6]
46	R2	GPIO2_7	lcd_data1	gpmc_a1		ehrpwm2B		pr1_pru1_pru_r30_1	pr1_pru1_pru_r31_1	gpio2[7]

PIN	PROC	NAME	MODE0	MODE1	MODE2	MODE3	MODE4	MODE5	MODE6	MODE7
1,2						GND				
3,4						DC_3.3V				
5,6						VDD_5V				
7,8						SYS_5V				
9						PWR_BTN				
10	A10					SYS_RESETn				
11	T17	UART4_RXD	gpmc_wait0	mi2_crs	gpmc_csn4	rmii2_crs_dv	mmc1_sdcd		uart4_rxd_mux2	gpio0[30]
12	U18	GPIO1_28	gpmc_be1n	mi2_col	gpmc_csn6	mmc2_dat3	gpmc_dir		mcaspo_aclkr_mux3	gpio1[28]
13	U17	UART4_TXD	gpmc_wpn	mi2_nxerr	gpmc_csn5	rmii2_nxerr	mmc2_sdcd		uart4_txd_mux2	gpio0[31]
14	U14	EHRPWM1A	gpmc_a2	mi2_bxd3	rgmii2_tdt3	mmc2_dat1	gpmc_a18		ehrpwm1A_mux1	gpio1[18]
15	R13	GPIO1_16	gpmc_a0	gmii2_bxen	rmii2_tdt1	mi2_bxen	gpmc_a16		ehrpwm1_tripzone_input	gpio1[16]
16	T14	EHRPWM1B	gpmc_a3	mi2_bxd2	rgmii2_tdt2	mmc2_dat2	gpmc_a19		ehrpwm1B_mux1	gpio1[19]
17	A16	I2C1_SCL	spi0_cs0	mmc2_sdw	I2C1_SCL	ehrpwm0_synci	pr1_uart0_bxd			gpio0[5]
18	B16	I2C1_SDA	spi0_d1	mmc1_sdw	I2C1_SDA	ehrpwm0_tripzone	pr1_uart0_rxd			gpio0[4]
19	D17	I2C2_SCL	uart1_rtsn	timer5	dcant0_rx	I2C2_SCL	spi1_cs1	pr1_uart0_rts_n		gpio0[13]
20	D18	I2C2_SDA	uart1_ctsn	timer6	dcant0_tx	I2C2_SDA	spi1_cs0	pr1_uart0_cts_n		gpio0[12]
21	B17	UART2_TXD	spi0_d0	uart2_bxd	I2C2_SCL	ehrpwm0B	pr1_uart0_rts_n		EMU3_mux1	gpio0[3]
22	A17	UART2_RXD	spi0_sclk	uart2_rxd	I2C2_SDA	ehrpwm0A	pr1_uart0_cts_n		EMU2_mux1	gpio0[2]
23	V14	GPIO1_17	gpmc_a1	gmii2_rxdv	rgmii2_rxdv	mmc2_dat0	gpmc_a17		ehrpwm0_synco	gpio1[17]
24	D15	UART1_TXD	uart1_bxd	mmc2_sdw	dcant1_rx	I2C1_SCL		pr1_uart0_bxd	pr1_pru0_pru_r31_16	gpio0[15]
25	A14	GPIO3_21*	mcaspo_ahdcikx	eQEP0_strobe	mcaspo_axr3	mcaspo_axr1	EMU4_mux2	pr1_uart0_rxd	pr1_pru0_pru_r31_7	gpio3[21]
26	D16	UART1_RXD	uart1_rxd	mmc1_sdw	dcant1_tx	I2C1_SDA		pr1_uart0_rxd	pr1_pru1_pru_r31_16	gpio3[14]
27	C13	GPIO3_19	mcaspo_fsr	eQEP0B_in	mcaspo_axr3	mcaspo1_fsr	EMU2_mux2	pr1_pru0_pru_r30_5	pr1_pru0_pru_r31_5	gpio3[19]
28	C12	GPIO3_18	mcaspo_ahdckr	ehrpwm0_synci	mcaspo_axr2	spi1_cs0	eCAP2_in_PWM2_out	pr1_pru0_pru_r30_3	pr1_pru0_pru_r31_3	gpio3[17]
29	B13	SP1_D0	mcaspo_fsr	ehrpwm0B		spi1_d0	mmc1_sdcd_mux1	pr1_pru0_pru_r30_1	pr1_pru0_pru_r31_1	gpio3[15]
30	D12	SP1_D1	mcaspo_axr0	ehrpwm0_tripzone		spi1_d1	mmc2_sdcd_mux1	pr1_pru0_pru_r30_2	pr1_pru0_pru_r31_2	gpio3[16]
31	A13	SP1_SCLK	mcaspo_aclkr	ehrpwm0A		spi1_sclk	mmc0_sdcd_mux1	pr1_pru0_pru_r30_0	pr1_pru0_pru_r31_0	gpio3[14]
32						VADC				
33	C8					AIN4				
34						AGND				
35	A8					AIN6				
36	B8					AIN5				
37	B7					AIN2				
38	A7					AIN3				
39	B6					AIN0				
40	C7					AIN1				
41#	D14	CLKOUT2	xdma_event_intr1		tskin	clkout2	timer7_mux1	pr1_pru0_pru_r31_16	EMU3_mux0	gpio0[20]
	D13	GPIO3_20	mcaspo_axr1	eQEP0_index		Mcaspo1_axr0	emu3	pr1_pru0_pru_r30_6	pr1_pru0_pru_r31_6	gpio3[20]
	C18	GPIO0_7	eCAP0_in_PWM0_out	uart3_bxd	spi1_cs1	pr1_eap0_eap_capin_apwm_o	spi1_sclk	mmc0_sdw	xdma_event_intr2	gpio0[7]
42@	B12	GPIO3_18	Mcaspo_aclkr	eQEP0A_in	Mcaspo_axr2			pr1_pru0_pru_r30_4	pr1_pru0_pru_r31_4	gpio3[18]
43-46						GND				

Figure 4: Expansion Header P8 and P9 Pinout. From <https://media.digikey.com/pdf/Data%20Sheets/Circuitco%20Elect/BB-BBLK-000%20Manual.pdf>

The above diagrams and charts are important to know when working with the BeagleBone Black Board. For more info, refer to the BeagleBone Black System Reference Manual from <https://media.digikey.com/pdf/Data%20Sheets/Circuitco%20Elect/BB-BBLK-000%20Manual.pdf>. These charts are incredibly important for future use because of how the pins are configured, and where they are. If you want to use a PWM signal from a BeagleBone, you have to find the pins that are able to use PWM.

Most Important Features of Angstrom

Angstrom is a Linux distribution primarily focused for embedded devices. Angstrom is a monolithic kernel suited for embedded devices given that resources are scarce on an embedded device. The distribution is open source, allowing for many to support the project; their github can be found at <https://github.com/Angstrom-distribution>. Angstrom uses opkg as its package manager; opkg is a lightweight package manager mainly found on embedded devices. Since Angstrom Os is a Linux Distribution, the features and advantages found in Linux are transferred over such as it being open source, lightweight, ease of use(gui or command line), and compatibility. The Linux operating system gives the programmer more advantages over using no kernel such as the use of timers, and scheduling, that is not available if no OS had been used.

The OS was flashed to the onboard flash memory, and was provided from <https://beagleboard.org/latest-images> (done in Assignment 1). The latest version of Angstrom on the site is from 2013-09-04. BeagleBone Black has since transitioned from Angstrom to Debian; this is probably due to Debian being more popular and more supported.

To learn more about Angstrom Distribution, you can visit: <http://www.angstrom-distribution.org/>

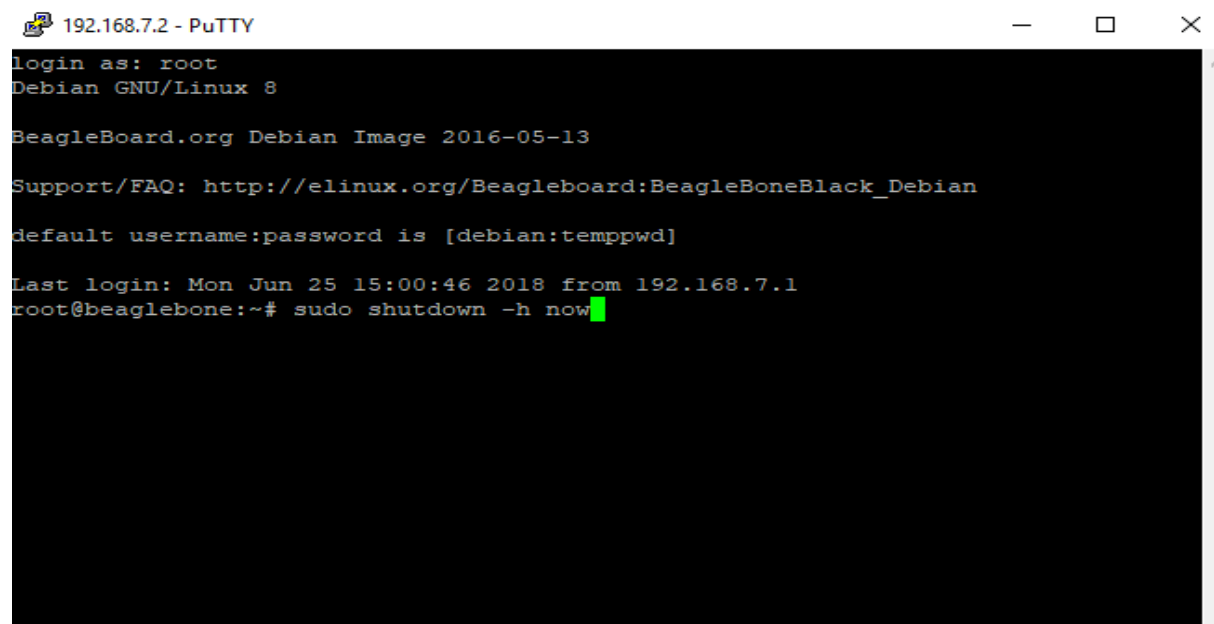
A screenshot of a PuTTY terminal window titled '192.168.7.2 - PuTTY'. The terminal shows a login sequence for a BeagleBone Black running Debian GNU/Linux 8. The text displayed is: 'login as: root', 'Debian GNU/Linux 8', 'BeagleBoard.org Debian Image 2016-05-13', 'Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian', 'default username:password is [debian:temppwd]', and 'Last login: Mon Jun 25 15:00:46 2018 from 192.168.7.1'. The prompt is 'root@beaglebone:~#', and the user has entered the command 'sudo shutdown -h now'.

Figure 5: Image of terminal access to BeagleBone Black thru PuTTY on a Windows PC.

Linux Kernel Modules (LKM)

In Assignment 1, the programs created for the BeagleBone Black were made for user space (C and Python) as supposed to being run in kernel space; the user programs were able to access the GPIO pins through system calls. Linux Kernel Modules (LKM) are pieces of code that can be loaded and unloaded from the kernel; therefore LKMs are always run in kernel mode. This relieves some of the overhead experienced with the earlier programs. An example of a LKM is a device driver; for example you might have a USB-to-Bluetooth adapter you want to use with the BeagleBone Black, the LKM (device driver) will be loaded when the device is in use, and unloaded when it is not. The LKM associated with the device can be used to utilize this device at boot, without the kernel knowing how the hardware works. This kind of modularity allows the kernel to be small, so the kernel does not have to include the drivers of every device, and the LKMs can be loaded and unloaded by the user without having to reset the BeagleBone Black. A downside of using LKMs is that it is more complicated to write for in comparison to user applications since it is lower level code. Another downside is that the kernel does not support floating-point operations.

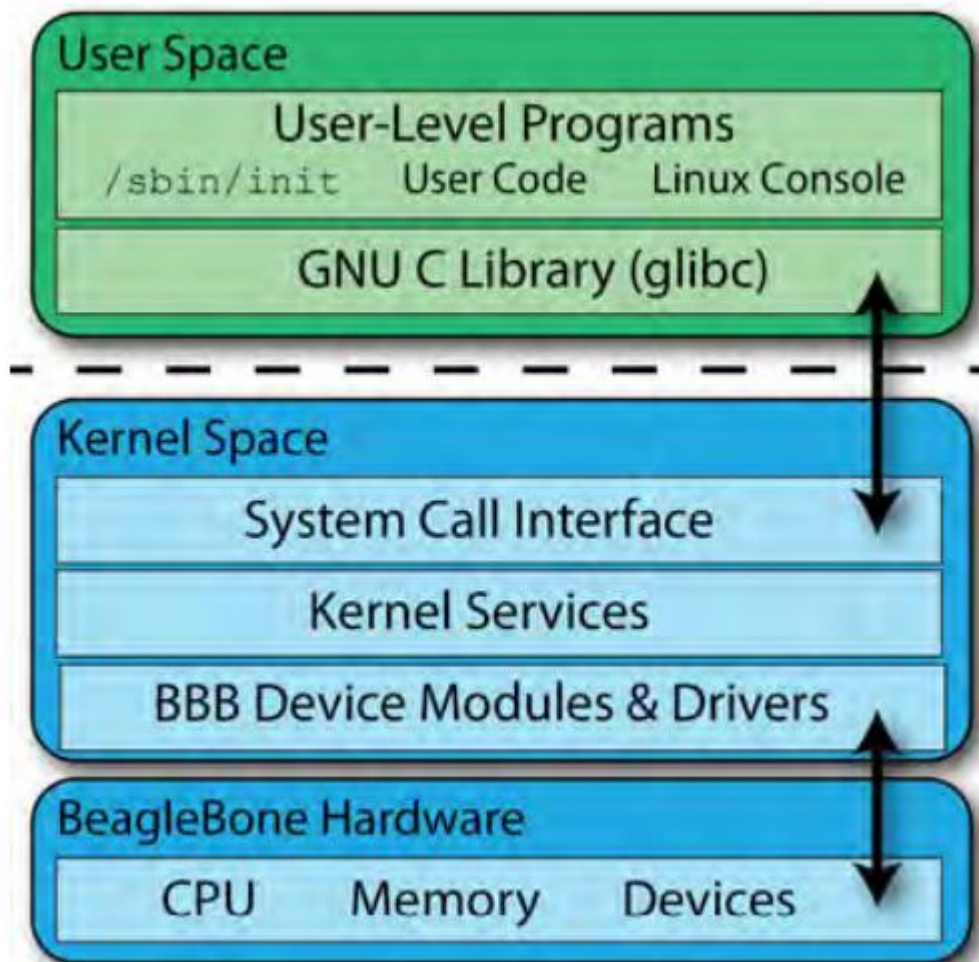


Figure 6: Image of the boundary between applications that run in User Mode, and the Linux Kernel Running in Kernel Mode.
From <http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/>

How To Use LKMs in Linux

insmod [filename] - inserts a module (*.ko) onto the kernel

The user loads module thru module daemon insmod; this executes modprobe to load the module in given a string.modprobe uses insmod to load any prerequisite modules, and then loads the module.

lsmod - shows the current modules that are loaded onto the kernel

rmmod [filename] - removes a module (*.ko) from the kernel

dmesg - returns the log outputs from the kernels (printed by printk function in module).

Useful for debugging purposes; can use printk statements to see if pieces of code worked.

B. Setting Up The Experiments

Steps to Install Angstrom On To The BeagleBone Black

The steps needed to install Angstrom on to the BeagleBone are outlined in Alternate Assignment 1. From

<https://docs.google.com/document/d/1bfqWsL0ZMUN1Vau8c5Mbq6AySsi5OSOyqleXtilrAQ/edit?usp=sharing>

Steps to Setup BeagleBone Black

Board Safety

Again, in an effort to minimize risk of damage to the board, the use of an anti-static wristband and an anti-static mat is highly encouraged to minimize the risk of Electrostatic Discharge.



Figure 7: Use of anti-static wrist band, and anti-static mat encouraged to minimize board damage. From amazon.ca.

Before attaching any peripherals to the board, it is best to consult the BeagleBone Black System Reference Manual in case the current drawn, or voltage provided to the pins damages the board. For this project, we will use more devices, and to ensure that we do not damage them, it is best to consult each devices respective data sheet before proceeding.

Connecting To The Internet and Correcting Time

Assuming the steps from Assignment 1 were followed (the steps that are vital for this project is the creation of the startup script that allows for access to the internet from Assignment 1), these next steps will allow your BeagleBone Black board to re-access the internet through USB. These step have to be conducted each time your PC resets.

Connecting the BeagleBone to the internet is an important step for the steps ahead. Internet access will allow you to download updates and gits for future projects. It also allows you to

correct the time, which will reset each time your BeagleBone resets. The current time will be utilized for some of the applications in this project.

1. On Windows, search for “Network and Sharing Center”, and then go to “Change Adapter Options”. A window with Ethernet and Wifi connections should open up.

OR

Right-click Lan, or Wifi symbol on bottom right of taskbar, and “Open Network and Internet Settings”, and then click “Change Adapter Options”.

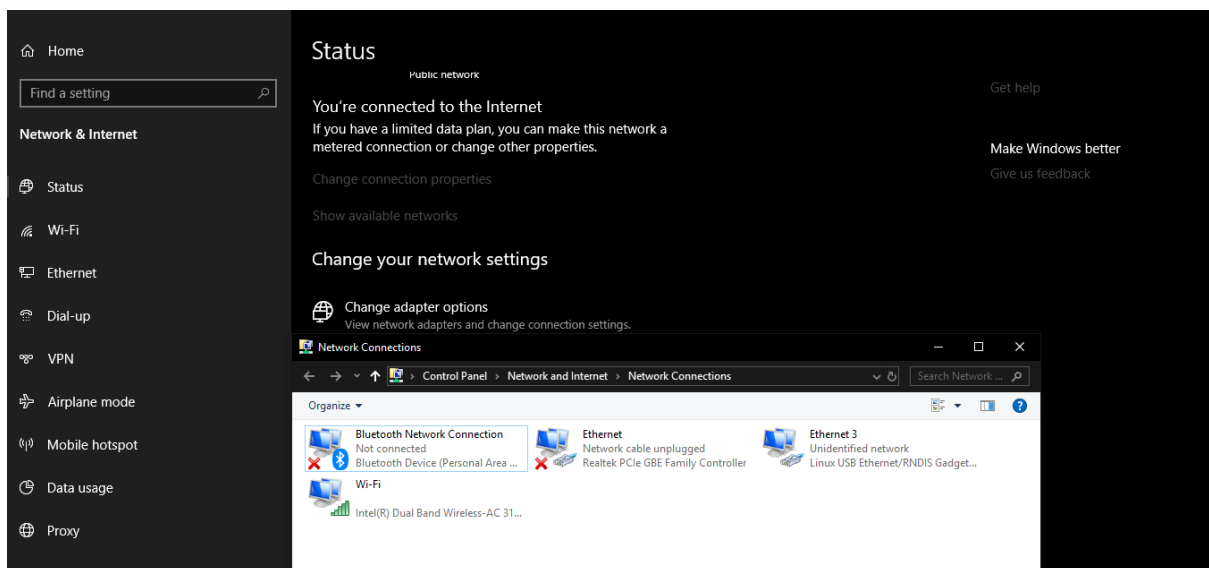


Figure 8: Window that opens up with Ethernet and Wifi Connections

2. Right click on the adapter that your PC is connected to the internet with (either Ethernet or Wifi), and go to “Properties”. Go to the “Sharing Tab” near the top of the window and check the “Allow other network users to connect through this computer’s Internet Connection”, and in the box below “Home Networking Connection”, enter the adapter that is referring to the BeagleBone Network (Ethernet 3 in my case), and close this window. Note: This step has to be done every time your PC reboots.

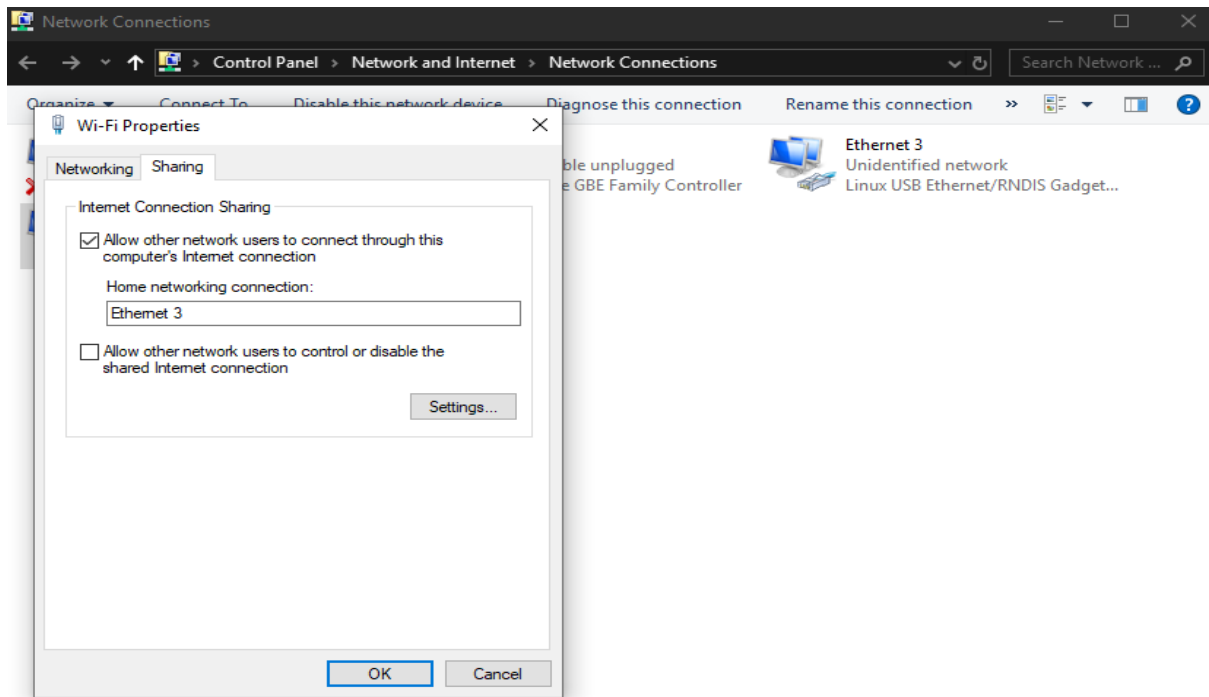


Figure 9: Allow the Beralge Board to connect to the internet through USB.

3. Right click the adapter referring to the board (Ethernet 3 in my case), and highlight Internet Protocol Version 4 with the mouse, and hit the Properties button. Check the “Obtain the IP address automatically” and the “Obtain DNS server address automatically” bubble. Close all the windows. Note: This step has to be done every time your PC reboots.

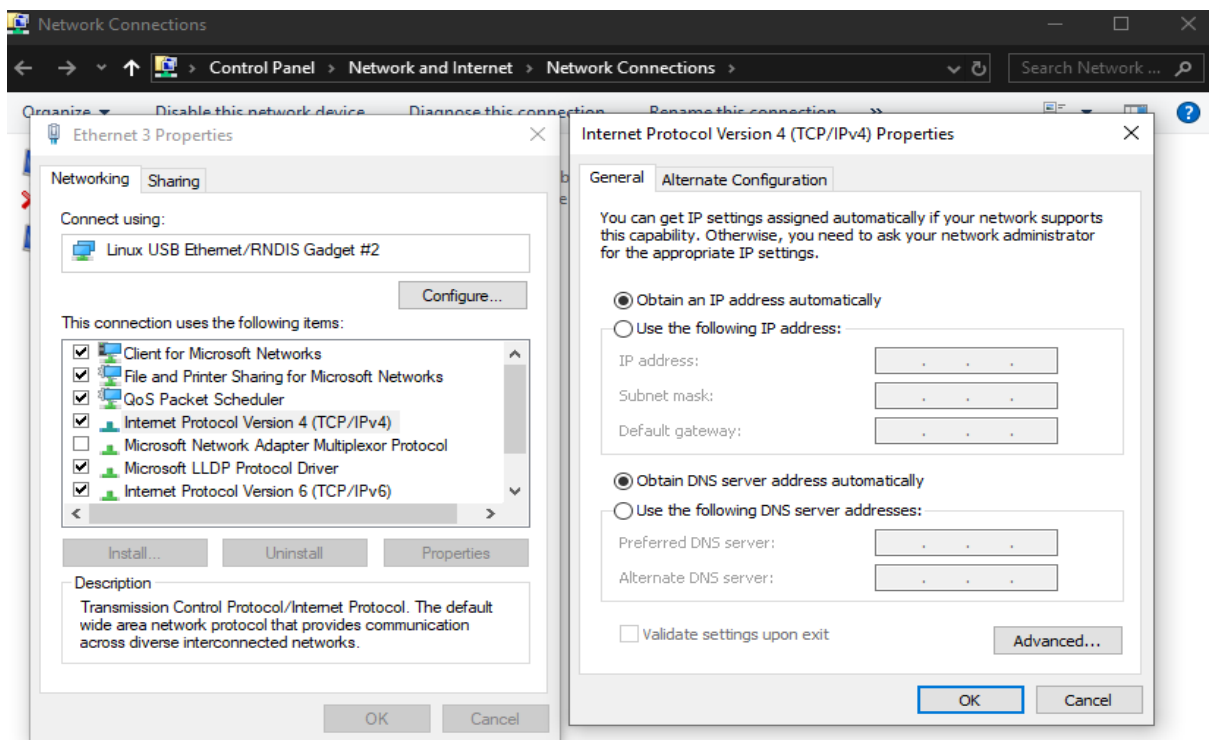
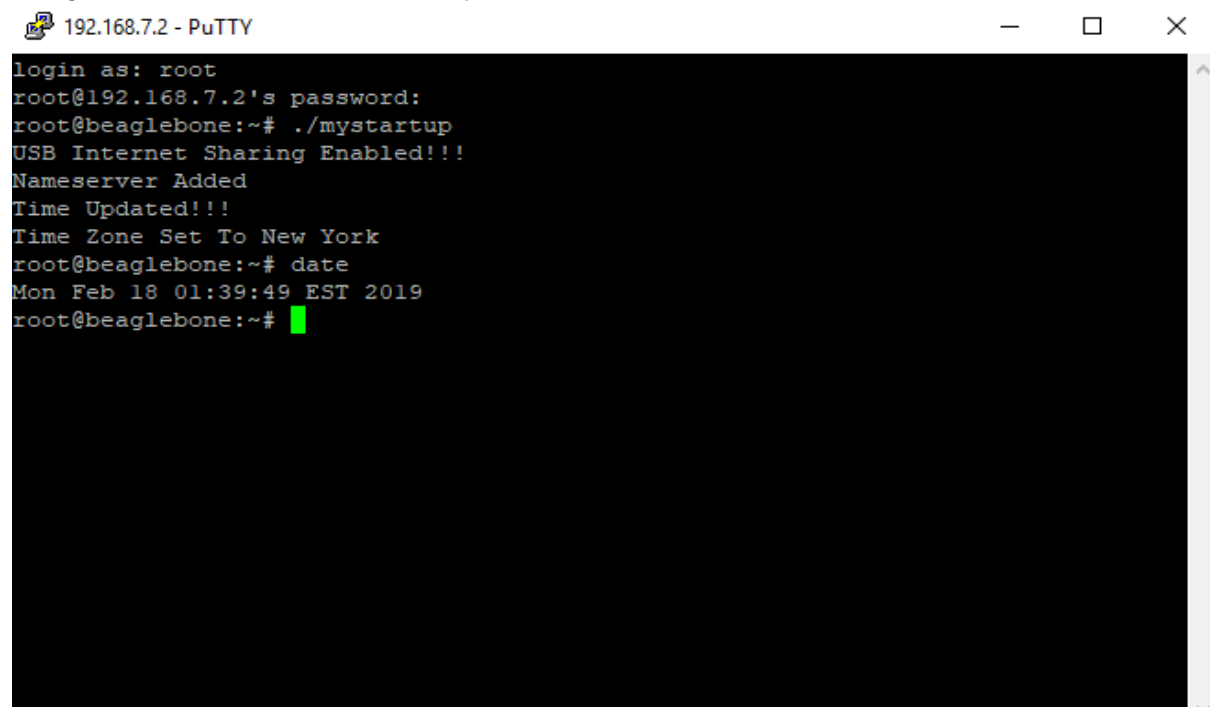


Figure 10: Obtain IP Address Automatically as supposed to using the IP Address assigned by Windows.

Once you access the BeagleBone Black thru Putty, the message in the Figure below will appear; "Time Zone Set To New York" with no other error messages above that ensures that the BeagleBone was able to correct time thru the NTP server and ultimately, the BeagleBone Black was successfully able to connect to the internet.



```
login as: root
root@192.168.7.2's password:
root@beaglebone:~# ./mystartup
USB Internet Sharing Enabled!!!
Nameserver Added
Time Updated!!!
Time Zone Set To New York
root@beaglebone:~# date
Mon Feb 18 01:39:49 EST 2019
root@beaglebone:~#
```


Figure 11: The mystartup script running ensuring internet connection.

C. Creating Software in User Space and in Kernel Mode (LKMs)

User Space

It was demonstrated in Assignment 1, that the user can create user-space programs that can utilize the software and hardware provided by the BeagleBone Black. The `hello.c` program prints "Hello World". The `ledseg.py` program displays the current time on the 7 segment display.

Hello World in C



```
GNU nano 2.2.5 File: hello.c

#include<stdio.h>
int main()
{
    printf("Hello World!\n");
}

[ Read 5 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text   ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Figure 12: Code for the hello.c program that prints "Hello World"

7-Segment Display That Shows The Current Time Using Python

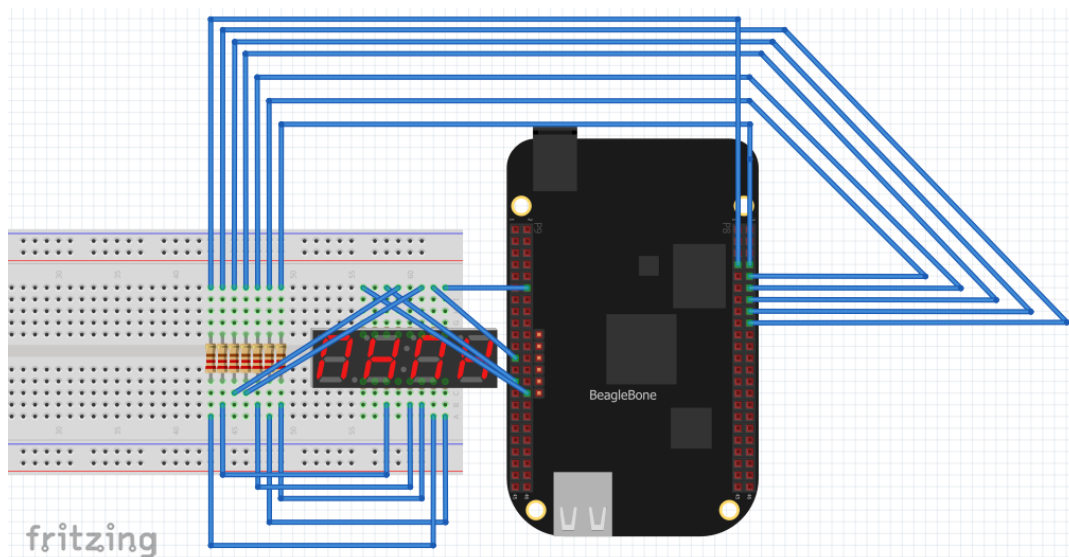


Figure 13: Schematic of how the 7-Segment display is connected to the BeagleBone



Figure 14: Output of the Python, 7-Seg program that displays the time

The codes for both are available in Alternate Assignment 1. Instead of focusing on user mode programs, LKMs will be used to interface with the hardware.

Kernel Mode

To create a LKM, we will have to create a C file. We have to include library header `linux/module.h` if we want to make a module. We will use a generic Makefile to compile all the LKMs throughout this project.

helloworld.c LKM Driver

Building Linux Kernel Modules: Hello World

To build kernel code, you must first have Linux headers installed on your device.

```
opkg update
opkg install kernel-headers
```

Then make the helloworld LKM and its Makefile as shown below (influenced by <http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/>):

helloworld.c

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

MODULE_LICENSE("GPL");
//modified from http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/
```



```
static int __init helloworld_initial(void)
{
    printk(KERN_INFO "Hello World.\n");
    return 0;
}

static void __exit helloworld_exit(void)
{
    printk(KERN_INFO "Goodbye World.\n");
}

module_init(helloworld_initial);
module_exit(helloworld_exit);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Kernel modules have at least two functions: the “initial” function which is called when the module is added to the kernel thru insmod, and the “exit” function, which is called when the module is removed from the kernel thru rmmod. These functions are fed to the macros module_init() and module_exit(). The initial function prints “Hello World” to the log (can be seen thru dmesg which is good for debugging purposes); if a module is not loaded in successfully a non-zero number is returned. The “KERN_INFO” refers to a priority level. MODULE_LICENSE (“GPL”) specifies the type of licensing this code will take on; this specific module use a General Public License (GPU). More statements could have been made inside the module file such as: MODULE_AUTHOR, MODULE_DESCRIPTION. For now these statements are excluded, but for future reference, it is good practice to specify it.

Makefile

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
obj-m += helloworld.o
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

This Makefile will be used for all the other LKMs ahead, with just the first line modified. Before making the Makefile, make a symbolic link as followed:

```
ln -s /usr/src/kernel /lib/modules/$(uname -r)/build
```

The symbolic link will link the build directory in /lib/modules to the kernel in order to run the Makefile.

You can use the command insmod helloworld.ko to load the LKM onto the kernel.

```
insmod helloworld.ko
```

To view the modules that are currently loaded (and to check if helloworld is loaded), use the Linux command:

```
lsmod
```

To remove the module, use rmmod:

```
rmmod helloworld.ko
```

```
root@beaglebone:/workspace/project# ls
Makefile      helloworld.c  helloworld.mod.c  helloworld.o
Module.symvers helloworld.ko  helloworld.mod.o  modules.order
root@beaglebone:/workspace/project# insmod helloworld.ko
root@beaglebone:/workspace/project# lsmod
Module                Size  Used by
helloworld            727  0
fuse                 51875  3
ip_tables            8294  0
x_tables             15072  1 ip_tables
g_multi              55905  2
libcomposite         15228  1 g_multi
rfcomm               25106  0
mtnet760lUsta        25701  0
mt760lUsta           753921  1 mtnet760lUsta
mtutil760lUsta       65890  2 mt760lUsta,mtnet760lUsta
ircomm_tty           14503  0
ircomm               8846  1 ircomm_tty
irda                 89974  2 ircomm_tty,ircomm
ipv6                229989  12
hidp                 10112  0
bluetooth            146100  4 hidp,rfcomm
rfkill               16510  3 bluetooth
autofs4              17432  2
root@beaglebone:/workspace/project#
```

Figure 15: Using lsmod to view the modules loaded onto the kernel.

```

root@beaglebone:/workspace/project# rmmod helloworld.ko
root@beaglebone:/workspace/project# lsmod
Module                  Size  Used by
fuse                    51875  3
ip_tables               8294   0
x_tables               15072  1 ip_tables
g_multi                 55905  2
libcomposite           15228  1 g_multi
rfcomm                  25106  0
mtnet7601Usta          25701  0
mt7601Usta              753921 1 mtnet7601Usta
mtutil7601Usta          65890  2 mt7601Usta, mtnet7601Usta
ircomm_tty             14503  0
ircomm                  8846   1 ircomm_tty
irda                    89974  2 ircomm_tty, ircomm
ipv6                   229989 12
hidp                    10112  0
bluetooth              146100  4 hidp, rfcomm
rfkill                  16510  3 bluetooth
autofs4                 17432  2
root@beaglebone:/workspace/project# █

```

Figure 16: Removing the helloworld LKM from the Kernel.

The shell command `dmesg` should return the printk messages “Hello World” and “Goodbye World”.

```

[ 395.377765] Hello World.
[ 395.713892] gadget: rndis req2l.00 v0000 i0000 l12
[ 395.715882] gadget: rndis reqal.01 v0000 i0000 l4096
[ 398.222507] gadget: rndis req2l.00 v0000 i0000 l12
[ 398.223996] gadget: rndis reqal.01 v0000 i0000 l4096
[ 400.740853] gadget: rndis req2l.00 v0000 i0000 l12
[ 400.744102] gadget: rndis reqal.01 v0000 i0000 l4096
[ 401.969391] [drm:output_poll_execute], [CONNECTOR:5:HDMI-A-1] status updated
from 2 to 2
[ 403.243807] gadget: rndis req2l.00 v0000 i0000 l12
[ 403.244040] gadget: rndis reqal.01 v0000 i0000 l4096
[ 405.747650] gadget: rndis req2l.00 v0000 i0000 l12
[ 405.748269] gadget: rndis reqal.01 v0000 i0000 l4096
[ 408.249173] gadget: rndis req2l.00 v0000 i0000 l12
[ 408.252423] gadget: rndis reqal.01 v0000 i0000 l4096
[ 410.531175] Goodbye World.

```

Figure 17: Contents of `dmesg` command; shows the loading and unloading of the module

helloworld.c LKM (with Parameter) Driver

Sometimes it is desirable to pass a parameter to the module when it is loaded; for operational purposes or maybe debugging. Instead of having the module print “Hello World”, we can have a module that prints “Hello <param>”. We slightly modify the above code to include parameters. The code was influenced by <http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/>

helloworld.c (with param)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

MODULE_LICENSE("GPL");
//modified from http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/

//default value of parameter if not used
static char *name="World";
//declares the parameter as character pointer
module_param(name, charp, S_IRUGO);

//initial function
static int __init helloworld_initial(void)
{
    printk(KERN_INFO "Hello %s.\n", name);
    return 0;
}

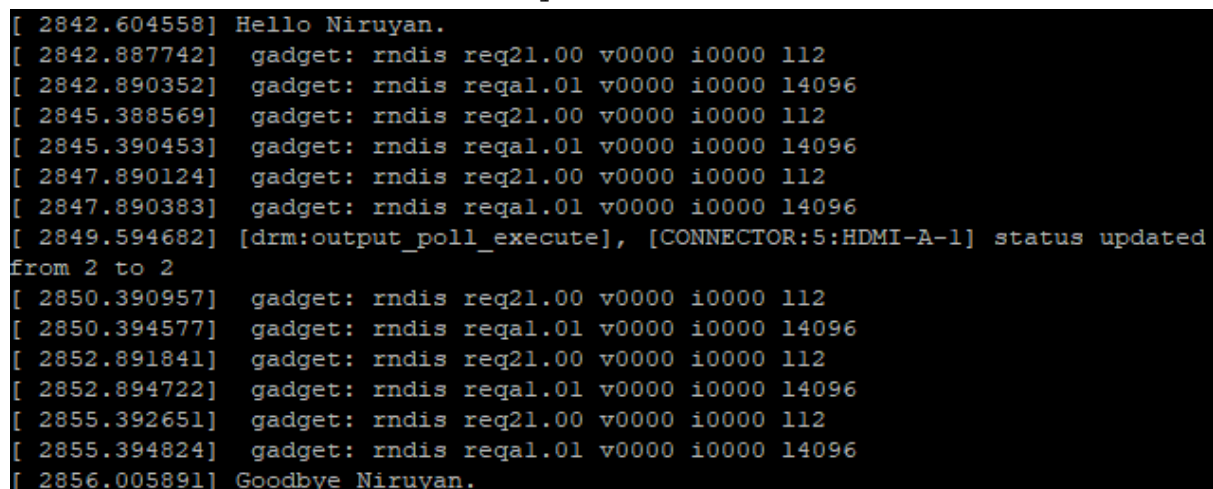
//exit function
static void __exit helloworld_exit(void)
{
    printk(KERN_INFO "Goodbye %s.\n", name);
}

module_init(helloworld_initial);
module_exit(helloworld_exit);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The same Makefile, with the first line edited, can be used. When loading the module, include the parameter when loading the module onto the Kernel:

```
insmod helloworld.ko name=Niruyan
```



The image shows a terminal window with a black background and white text. It displays the output of the 'insmod' command used to load the 'helloworld.ko' module with the parameter 'name=Niruyan'. The output consists of several lines of kernel log messages. The first line is '[2842.604558] Hello Niruyan.' followed by a series of 'gadget: rndis req21.00 v0000 i0000 112' and 'gadget: rndis reqal.01 v0000 i0000 14096' messages. A line '[2849.594682] [drm:output_poll_execute], [CONNECTOR:5:HDMI-A-1] status updated from 2 to 2' is also present. The sequence ends with '[2856.005891] Goodbye Niruyan.'

Figure 18: Figure of the printk statements issued by the helloworld module (with parameters initialized)

To see the parameters that are available to each module once it is loaded in, you can traverse to:

```
cd /sys/module/helloworld/parameters
```

The current value of the parameter can be found using:

```
cat name
```

```
root@beaglebone:/workspace/project2# cd /sys/module/helloworld/parameters/
root@beaglebone:/sys/module/helloworld/parameters# ls
name
root@beaglebone:/sys/module/helloworld/parameters# cat name
Niruyan
root@beaglebone:/sys/module/helloworld/parameters#
```

Figure 19: You can find the parameters of a LKM by traversing to the sys/module/ directory as explained above.

Controlling LED in Kernel Mode using a “Button” Driver

For this experiment, we are going to have a “button” that sends an interrupt to a GPIO port on the BeagleBone Black Board. Instead of a button (since none were readily available), we will use a resistor as a “button” (one of the ends of the resistor is connected to 3.3V, and the other end will periodically be connected to the GPIO pin by the user if they want to turn on or off the LED, which will mimic the action of a button); this will momentarily create a rising edge (mimicking a button) that the BeagleBone will detect. The interrupt only detects rising edges so if they want to turn it on and then off, the user has to momentarily connect the GPIO port to change state once, and then connect it again to change back to its original state. When the BeagleBone receives the interrupt, it toggles the gate signal of the MOSFET (thru a different GPIO port) that will allow the current to flow thru the LED. Functions concerned with controlling GPIO pins in LKMs can be found at: <https://elixir.bootlin.com/linux/v3.8/source/include/linux/gpio.h>.

Since we are using a “button”, we have to de-bounce it. Due to the mechanical nature of buttons, the button can change multiple states within a short period of time (as shown in the Figure below). The BeagleBone is able to detect this and treats it as multiple interrupts even though the user intended for one. To avoid this issue we will call the debounce function included that will neglect any other interrupts from the same source for a short period of time (200 ms for this case).

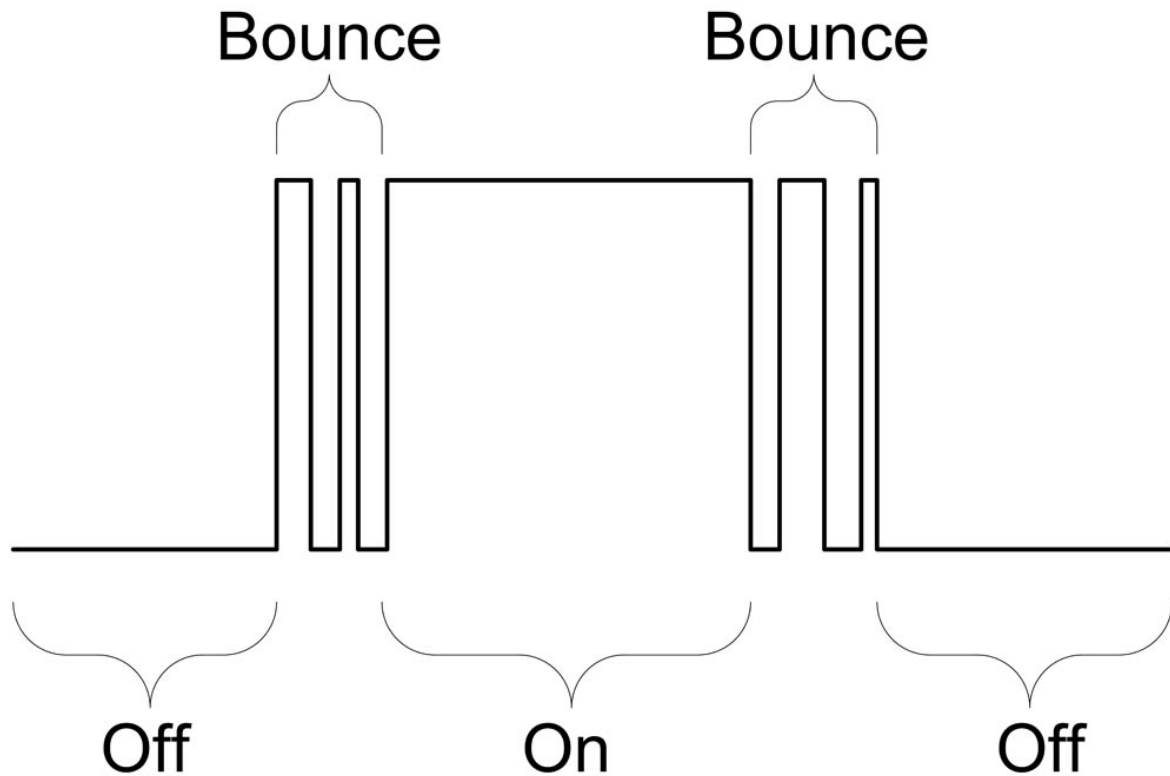


Figure 20: Due to the mechanical nature of buttons, the button will “bounce” when going from off to on or vice-versa. To combat this, a delay is implemented by code so that the buttons state cannot change in a relatively short time (every 200 ms).

The LKM to drive the LED is shown below. It initializes the direction of the GPIO port (pin that controls LED is output, and the pin that receives interrupt is input). Whenever the “button” is clicked, a interrupt handler is evoked. In the ISR, it toggles the LED, increments the number of clicks, and a printk statement is issued, containing the state of the LED and the number of clicks. When the module is removed, the GPIO ports are reset. The same Makefile, with the first line modified, was used. Note, the button does no directly control the LED; buttons controls the LED thru the BeagleBone.

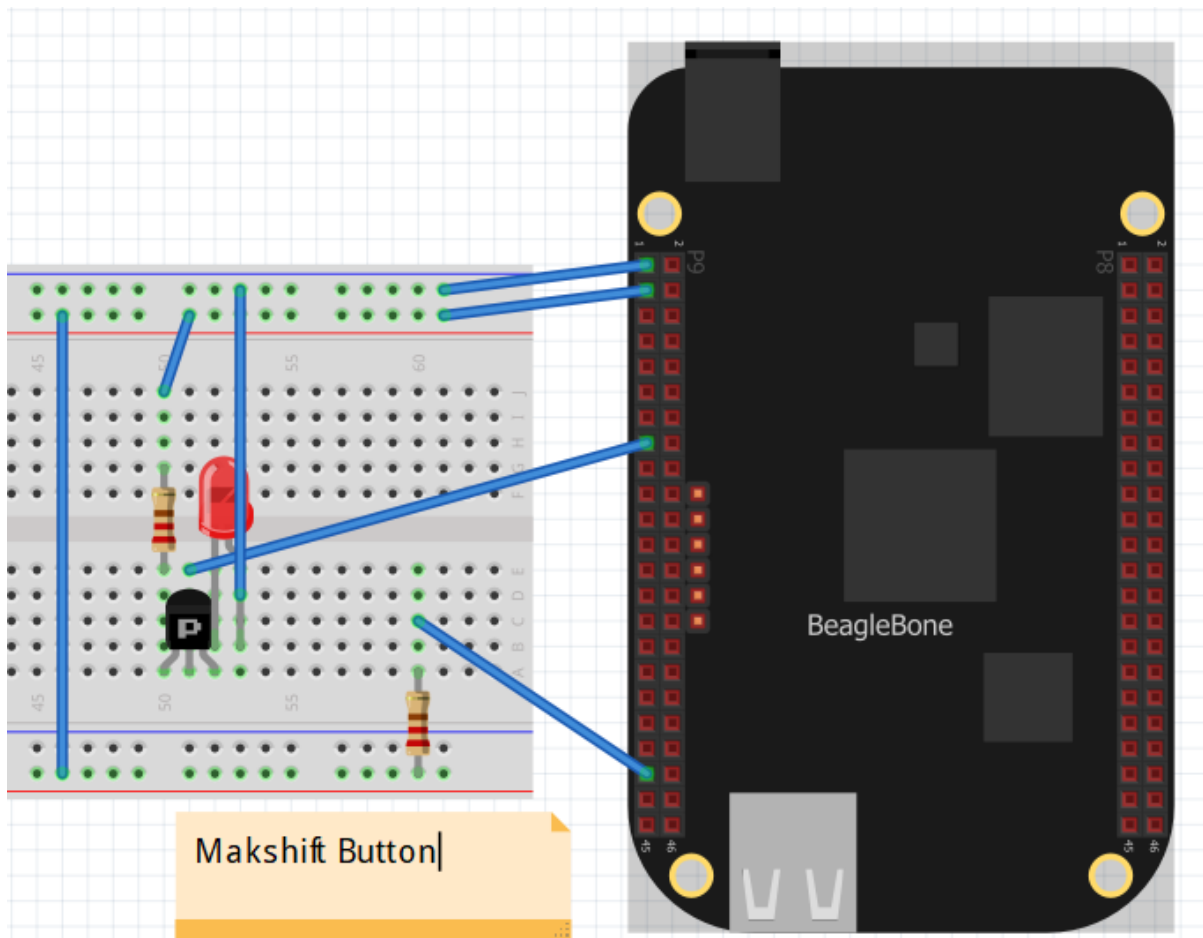


Figure 21: Schematic for Button Controlled LED.

controled.c

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/gpio.h>
#include <linux/interrupt.h>

MODULE_LICENSE("GPL");
//modified from http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/

static unsigned int gpioLED = 48; //GPIO Pin connected to Gate of MOSFET
static unsigned int gpioButton = 20; //GPIO Pin connected to the resistor
static unsigned int irqNumber; //The interrupt number; will get later on
static bool ledState=0; //State of the LED
static unsigned int numberPresses = 0; //number of presses of button

//prototype the IRQ Handler of the button interrupt
static irq_handler_t button_irq_handler(unsigned int irq, void *dev_id, struct pt_regs *regs);

//Initial Function
static int __init initial(void){

//holds value of how interrupt was handled
int result = 0;

//checks to see if valid GPIO pin
if (!gpio_is_valid(gpioLED)){
    printk(KERN_INFO "Invalid GPIO\n");
}

```

```

        return -ENODEV;
    }

//Initial State of the LED
    ledState = true;
//enable GPIO 48 (LED) from sysfs
    gpio_request(gpioLED, "sysfs");
//Set LED_GPIO to output and gives initial value of on
    gpio_direction_output(gpioLED, ledState);
//the gpioLED cannot change from output to input
    gpio_export(gpioLED, false);

// Enable GPIO 20
    gpio_request(gpioButton, "sysfs");
//Set GPIO as Input
    gpio_direction_input(gpioButton);
//Set debounce
    gpio_set_debounce(gpioButton, 200);
//the GPIO cannot change from input to output
    gpio_export(gpioButton, false);

//Map GPIO Pin to Interrupt Number
    irqNumber = gpio_to_irq(gpioButton);

//Requests Interrupt
    result = request_irq(irqNumber, (irq_handler_t) button_irq_handler, IRQF_TRIGGER_RISING, "button_irq_handler", NULL);

    return result;
}

//exit function
static void __exit exit(void){
    // Turn the LED off, makes it clear the device was unloaded
    gpio_set_value(gpioLED, 0);
    // Unexport the LED GPIO
    gpio_unexport(gpioLED);
    // Free the IRQ number, no *dev_id required in this case
    free_irq(irqNumber, NULL);
    // Unexport the Button GPIO
    gpio_unexport(gpioButton);
    // Free the LED GPIO
    gpio_free(gpioLED);
    // Free the Button GPIO
    gpio_free(gpioButton);
    printk(KERN_INFO "Goodbye from LED\n");
}

//ISR of button
static irq_handler_t button_irq_handler(unsigned int irq, void *dev_id, struct pt_regs *regs){
//toggle LED value
    ledState = !ledState;
//Toggle LED
    gpio_set_value(gpioLED, ledState);
//Increment number of button presses
    numberPresses++;
    printk("The LED now has a state %d. The button has been pressed %d times so far\n", ledState, numberPresses);
    return (irq_handler_t) IRQ_HANDLED;
}
module_init(initial);
module_exit(exit);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

The same Make file, with the first line edited, can be used.

```
192.168.7.2 - PuTTY
[ 537.762701] gadget: rndis reqal.01 v0000 i0000 l4096
[ 540.259712] gadget: rndis req21.00 v0000 i0000 l12
[ 540.262852] gadget: rndis reqal.01 v0000 i0000 l4096
[ 542.406886] [drm:output_poll_execute], [CONNECTOR:5:HDMI-A-1] status updated from 2 to 2
[ 542.434450] The LED now has a state 0. The button has been pressed 1 times so far
[ 542.437194] The LED now has a state 1. The button has been pressed 2 times so far
[ 542.437633] The LED now has a state 0. The button has been pressed 3 times so far gadget: rndis req21.00 v0000 i0000 l12
[ 542.763042] gadget: rndis reqal.01 v0000 i0000 l4096
[ 544.419359] The LED now has a state 1. The button has been pressed 4 times so far gadget: rndis req21.00 v0000 i0000 l12
[ 545.263036] gadget: rndis reqal.01 v0000 i0000 l4096
[ 545.369181] The LED now has a state 0. The button has been pressed 5 times so far
[ 545.738556] The LED now has a state 1. The button has been pressed 6 times so far
[ 546.049868] The LED now has a state 0. The button has been pressed 7 times so far
[ 546.319728] The LED now has a state 1. The button has been pressed 8 times so far
[ 546.591592] The LED now has a state 0. The button has been pressed 9 times so far
[ 546.816916] The LED now has a state 1. The button has been pressed 10 times so far
[ 546.822511] The LED now has a state 0. The button has been pressed 11 times so far
[ 547.042735] The LED now has a state 1. The button has been pressed 12 times so far
[ 547.310059] The LED now has a state 0. The button has been pressed 13 times so far
[ 547.758176] The LED now has a state 1. The button has been pressed 14 times so far
[ 547.764350] gadget: rndis req21.00 v0000 i0000 l12
[ 547.767201] gadget: rndis reqal.01 v0000 i0000 l4096
[ 548.282820] The LED now has a state 0. The button has been pressed 15 times so far
[ 548.579161] The LED now has a state 1. The button has been pressed 16 times so far
```

Figure 22: dmesg showing when the button is pressed and how many times

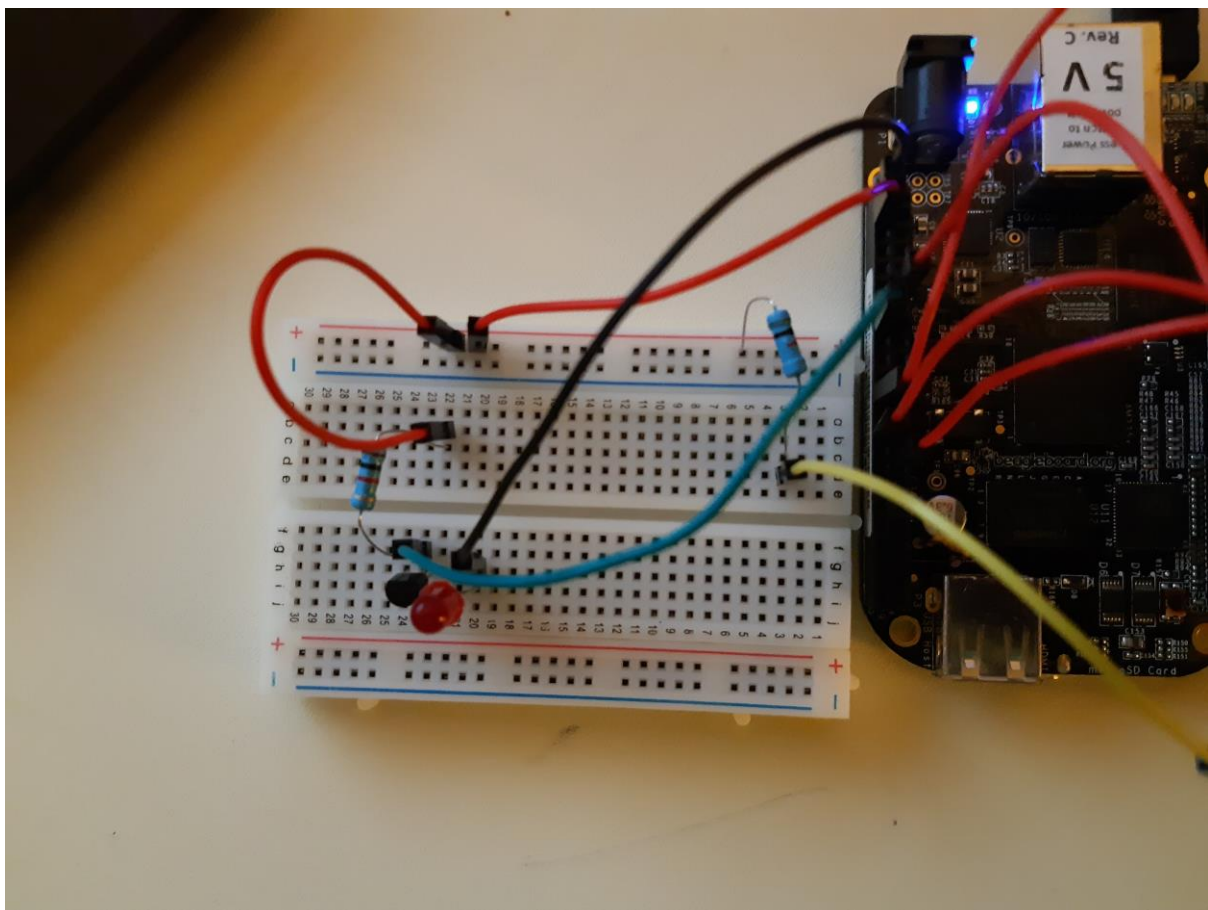


Figure 23: Using the LKM driver to turn off the LED.

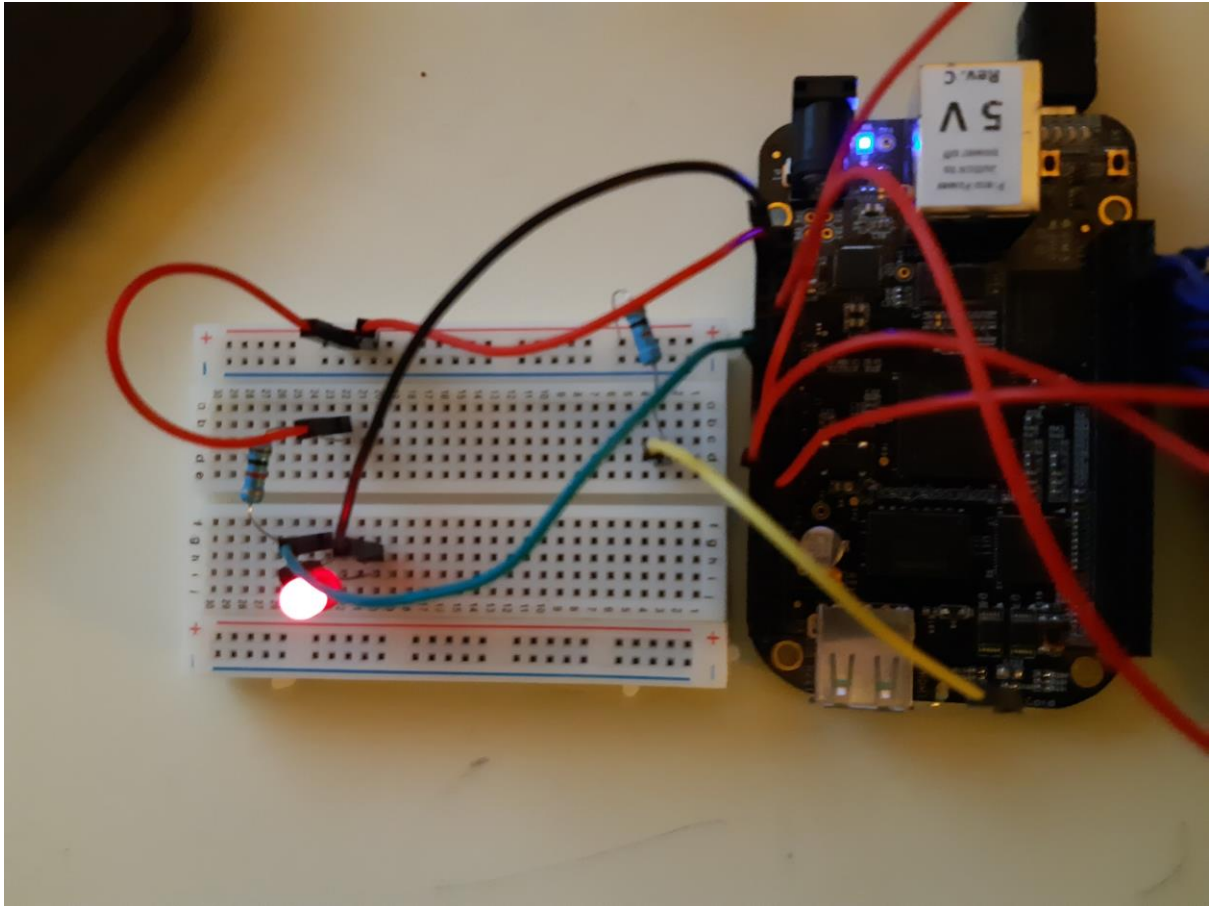


Figure 24: Using the LKM driver to turn on the LED.

Controlling 7-Seg Driver

We will now move the 7-Seg display code from Userspace to Kernel Space; the functionality of this code slightly differs from the one made in python. The GPIO pins are first initialized; this is similar to how it was done in the python program. Instead of using a sleep function to switch between the digits, we will use a periodic timer that will invoke a timer handler (`_TimerHandler`). The timer is invoked every 5ms; this results in low artifacting. Everytime the timer handler is invoked, it grabs the system time, issues a `printk` system to the log for debugging purposes (prints the current time, and which digit is currently being lit), and calls the `lightswitch` function (similar to the `lightswitch` function made in the python program where a digit and number is given) once every 5ms (timer interrupt time), to light the LED. When the module is removed, the GPIO pins are reset.

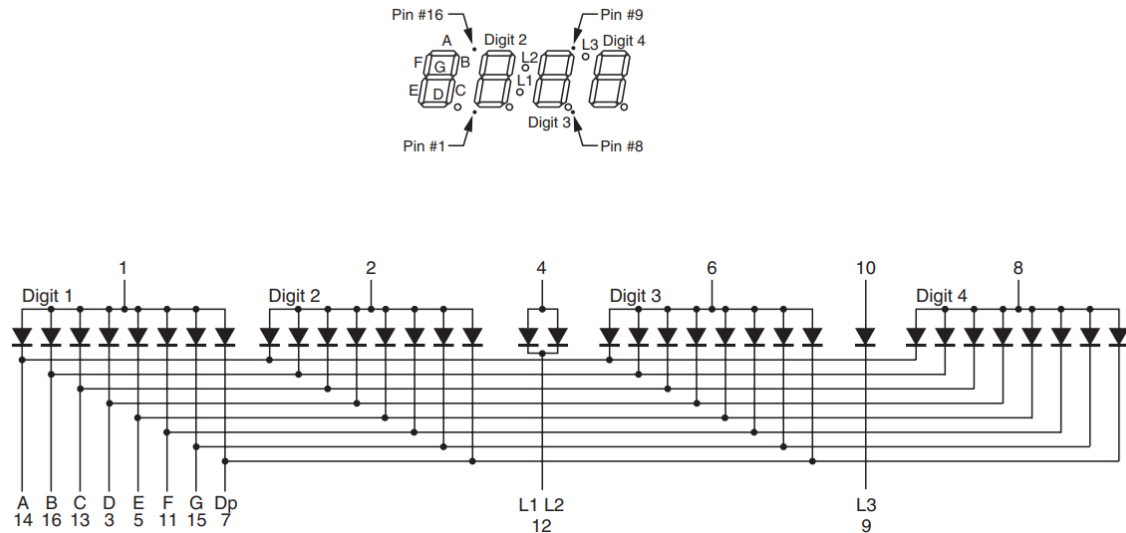


Figure 25: Schematic of 7-Seg display.

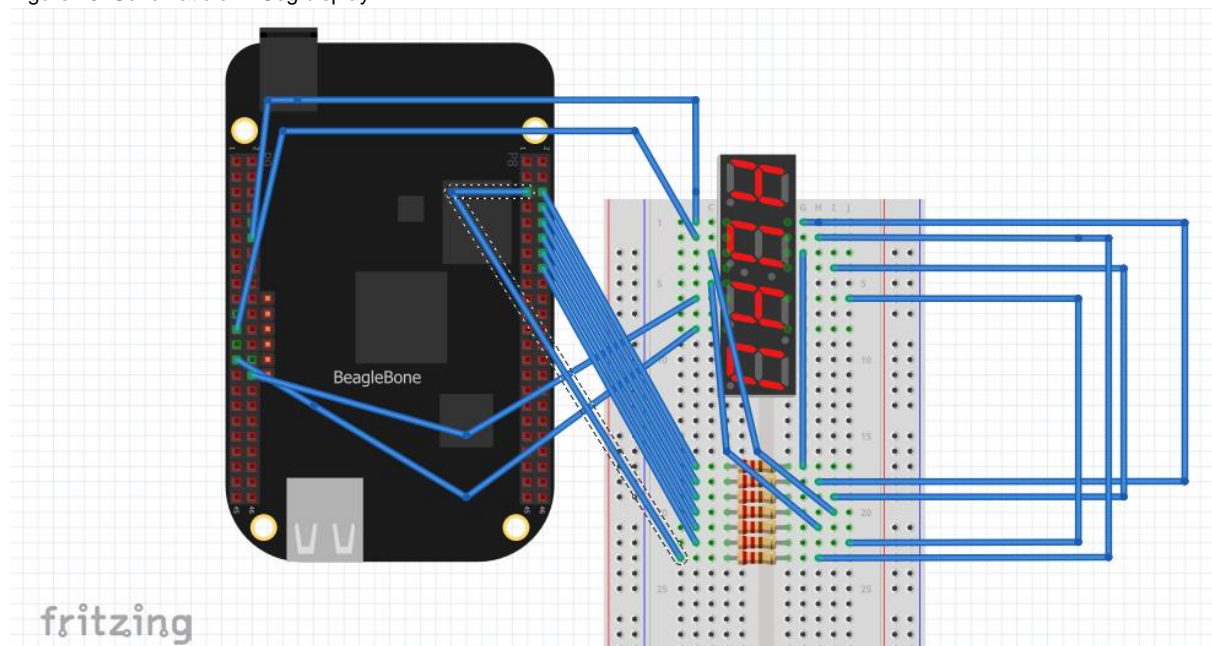


Figure 26: Wiring of 7-Seg Display to show time.

sevensseg.c

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/gpio.h>
#include <linux/interrupt.h>
#include <linux/time.h>
#include <linux/timer.h>

MODULE_LICENSE("GPL");

//Map GPIO pins to role
static unsigned int digit1 = 60;
static unsigned int digit2 = 49;
static unsigned int digit3 = 112;
static unsigned int digit4 = 115;

```

```

static unsigned int sega = 67;
static unsigned int segb = 68;
static unsigned int segc = 44;
static unsigned int segd = 26;
static unsigned int sege = 46;
static unsigned int segf = 65;
static unsigned int segg = 66;

//Rate at which each digits switch
static int seg_clock_refresh = 5;

//Initialize timer
struct timer_list seg_timer;

//prototype function that will
static void lightswitch(int digit, int num);
static void _TimerHandler(unsigned long data);

//initialize structure for time keeping
static struct timespec ts;

//which digit is being lit
static int whichdigit = 0;

//Initial Function
static int __init initialz(void){

    printk(KERN_INFO "Hello from 7Seg\n");

//INITIALIZE GPIO PINS
//digit1 GPIO initialization
    gpio_request(digit1, "sysfs");
    gpio_direction_output(digit1, 0);
    gpio_export(digit1, false);

//digit2 GPIO initialization
    gpio_request(digit2, "sysfs");
    gpio_direction_output(digit2, 0);
    gpio_export(digit2, false);

//digit3 GPIO initialization
    gpio_request(digit3, "sysfs");
    gpio_direction_output(digit3, 0);
    gpio_export(digit3, false);

//digit4 GPIO initialization
    gpio_request(digit4, "sysfs");
    gpio_direction_output(digit4, 0);
    gpio_export(digit4, false);

//sega GPIO initialization
    gpio_request(sega, "sysfs");
    gpio_direction_output(sega, 1);
    gpio_export(sega, false);

//segb GPIO initialization
    gpio_request(segb, "sysfs");
    gpio_direction_output(segb, 1);
    gpio_export(segb, false);

//segc GPIO initialization
    gpio_request(segc, "sysfs");
    gpio_direction_output(segc, 1);
    gpio_export(segc, false);

```



```

//segd GPIO initialization
gpio_request(segd, "sysfs");
gpio_direction_output(segd, 1);
gpio_export(segd, false);

//sege GPIO initialization
gpio_request(sege, "sysfs");
gpio_direction_output(sege, 1);
gpio_export(sege, false);

//segf GPIO initialization
gpio_request(segf, "sysfs");
gpio_direction_output(segf, 1);
gpio_export(segf, false);

//segg GPIO initialization
gpio_request(segg, "sysfs");
gpio_direction_output(segg, 1);
gpio_export(segg, false);

//initialize timer
setup_timer(&seg_timer, _TimerHandler, 0);
mod_timer(&seg_timer, jiffies + msecs_to_jiffies(seg_clock_refresh));

return 0;
}

static void __exit exitz(void){
//release all the GPIO Pins
gpio_set_value(digit1, 0);
gpio_set_value(digit2, 0);
gpio_set_value(digit3, 0);
gpio_set_value(digit4, 0);

gpio_set_value(sega, 1);
gpio_set_value(segb, 1);
gpio_set_value(segc, 1);
gpio_set_value(segd, 1);
gpio_set_value(sege, 1);
gpio_set_value(segf, 1);
gpio_set_value(segg, 1);

gpio_unexport(digit1);
gpio_unexport(digit2);
gpio_unexport(digit3);
gpio_unexport(digit4);

gpio_unexport(sega);
gpio_unexport(segb);
gpio_unexport(segc);
gpio_unexport(segd);
gpio_unexport(sege);
gpio_unexport(segf);
gpio_unexport(segg);

gpio_free(digit1);
gpio_free(digit2);
gpio_free(digit3);
gpio_free(digit4);

gpio_free(sega);
gpio_free(segb);
gpio_free(segc);
gpio_free(segd);
gpio_free(sege);
gpio_free(segf);

```

```

gpio_free(seg);

del_timer(&seg_timer);

printk(KERN_INFO "Goodbye from 7Seg\n");
}

static void lightswitch(int digit, int num){
//similar to the python program made
gpio_set_value(digit1, 0);
gpio_set_value(digit2, 0);
gpio_set_value(digit3, 0);
gpio_set_value(digit4, 0);

gpio_set_value(sega, 1);
gpio_set_value(segb, 1);
gpio_set_value(segc, 1);
gpio_set_value(segd, 1);
gpio_set_value(sege, 1);
gpio_set_value(segf, 1);
gpio_set_value(seg, 1);

if (num == 1){
    gpio_set_value(segb, 0);
    gpio_set_value(segc, 0);
}
else if(num == 2){
    gpio_set_value(sega, 0);
    gpio_set_value(segb, 0);
    gpio_set_value(segd, 0);
    gpio_set_value(sege, 0);
    gpio_set_value(seg, 0);
}
else if(num == 3){
    gpio_set_value(sega, 0);
    gpio_set_value(segb, 0);
    gpio_set_value(segc, 0);
    gpio_set_value(segd, 0);
    gpio_set_value(seg, 0);
}
else if(num == 4){
    gpio_set_value(segb, 0);
    gpio_set_value(segc, 0);
    gpio_set_value(segf, 0);
    gpio_set_value(seg, 0);
}
else if(num == 5){
    gpio_set_value(sega, 0);
    gpio_set_value(segc, 0);
    gpio_set_value(segd, 0);
    gpio_set_value(segf, 0);
    gpio_set_value(seg, 0);
}
else if(num == 6){
    gpio_set_value(sega, 0);
    gpio_set_value(segc, 0);
    gpio_set_value(segd, 0);
    gpio_set_value(sege, 0);
    gpio_set_value(segf, 0);
    gpio_set_value(seg, 0);
}
else if(num == 7){
    gpio_set_value(sega, 0);
    gpio_set_value(segb, 0);
    gpio_set_value(segc, 0);

```

```

}
else if(num == 8){
    gpio_set_value(sega, 0);
    gpio_set_value(segb, 0);
    gpio_set_value(segc, 0);
    gpio_set_value(segd, 0);
    gpio_set_value(sege, 0);
    gpio_set_value(segf, 0);
    gpio_set_value(segg, 0);
}
else if(num == 9){
    gpio_set_value(sega, 0);
    gpio_set_value(segb, 0);
    gpio_set_value(segc, 0);
    gpio_set_value(segf, 0);
    gpio_set_value(segg, 0);
}
else if(num == 0){
    gpio_set_value(sega, 0);
    gpio_set_value(segb, 0);
    gpio_set_value(segc, 0);
    gpio_set_value(segd, 0);
    gpio_set_value(sege, 0);
    gpio_set_value(segf, 0);
}
else{
    gpio_set_value(sega, 1);
    gpio_set_value(segb, 1);
    gpio_set_value(segc, 1);
    gpio_set_value(segd, 1);
    gpio_set_value(sege, 1);
    gpio_set_value(segf, 1);
    gpio_set_value(segg, 1);
}

if (digit == 1){
    gpio_set_value(digit1, 1);
}
else if (digit == 2){
    gpio_set_value(digit2, 1);
}
else if (digit == 3){
    gpio_set_value(digit3, 1);
}
else if (digit == 4){
    gpio_set_value(digit4, 1);
}
else{
    gpio_set_value(digit1, 0);
    gpio_set_value(digit2, 0);
    gpio_set_value(digit3, 0);
    gpio_set_value(digit4, 0);
}
}

//Timer handler function
static void _TimerHandler(unsigned long data){
    gettimeofday(&ts);
    printk(KERN_INFO "%.2lu:%.2lu:%.2lu %d\n", ((ts.tv_sec-3600*4)/3600)%(24),(ts.tv_sec/60)%(60), ts.tv_sec%60, whichdigit);
    if (whichdigit == 0){
        lightswitch(1, ((ts.tv_sec-3600*4)/3600)%(24)/10);
    }
    else if (whichdigit == 1){
        lightswitch(2, ((ts.tv_sec-3600*4)/3600)%(24)%10);
    }
    else if (whichdigit == 2){

```

```

    lightswitch(3, (ts.tv_sec/60)%(60)/10);
}
else if(whichdigit == 3){
    lightswitch(4, (ts.tv_sec/60)%(60)%10);
}
else{
    lightswitch(1, 2);
}
whichdigit=(whichdigit+1)%4;

mod_timer(&seg_timer, jiffies + msecs_to_jiffies(seg_clock_refresh));
}

module_init(initialz);
module_exit(exitz);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

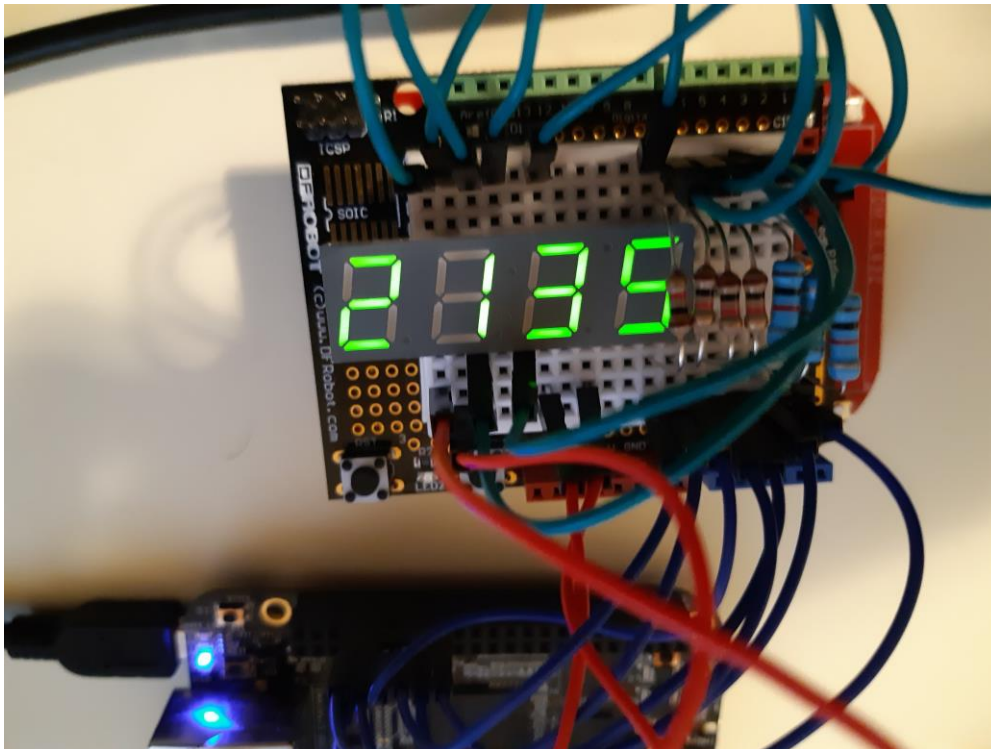


Figure 27: Showing the current time using 7-Seg LKM

Counting the Number of Seconds LED is Lit

This program will count the number of seconds a LED is lit. This will use two interrupts, one switch to toggle the state of the LED (acts like a switch so able to detect falling and rising edges), and another to reset the time (acts like a reset button for a stopwatch). The program will count the number of seconds it is lit, and display it on the 7-Segment Display. The switch will always toggle the state of the LED, so if the switch was open when the module was first loaded in, closing the switch will turn it off (since the LED is initially on at the start), or if the switch was closed when the module was loaded, opening it will turn it off (since the LED is initially on at the start). The reset “button” (again implemented with a momentary short) will reset the counter on the seven segment display; can be reset if LED is on or off. If the LED is lit, the time is incremented every LED refresh time (every 5ms).

counter.c

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/gpio.h>
#include <linux/interrupt.h>
#include <linux/time.h>
#include <linux/timer.h>

MODULE_LICENSE("GPL");

static unsigned int digit1 = 60;
static unsigned int digit2 = 49;
static unsigned int digit3 = 112;
static unsigned int digit4 = 115;

static unsigned int sega = 67;
static unsigned int segb = 68;
static unsigned int segc = 44;
static unsigned int segd = 26;
static unsigned int sege = 46;
static unsigned int segf = 65;
static unsigned int segg = 66;

static unsigned int switch_led = 27;
static unsigned int switch_irqnumber;
static unsigned int reset_counter = 47;
static unsigned int reset_irqnumber;

static unsigned int led = 20;

static unsigned int led_state = 1;

static int seg_clock_refresh = 5;
struct timer_list seg_timer;

static unsigned int msec_elap = 0; //number of presses of button

static void lightswitch(int digit, int num);
static void _TimerHandler(unsigned long data);

static struct timespec ts;
static int whichdigit = 0;

//prototype the IRQ Handler of the button interrupt
static irq_handler_t switch_irq_handler(unsigned int irq, void *dev_id, struct pt_regs *regs);
static irq_handler_t reset_irq_handler(unsigned int irq, void *dev_id, struct pt_regs *regs);

//Initial Function
static int __init initialz(void){
    int result = 0;

    printk(KERN_INFO "Hello from Counter\n");

    //////////////////////////////////////
    //INITIALIZE GPIO PINS
    //digit1 GPIO initialization
    gpio_request(digit1, "sysfs");
    gpio_direction_output(digit1, 0);
    gpio_export(digit1, false);

    //digit2 GPIO initialization
    gpio_request(digit2, "sysfs");
    gpio_direction_output(digit2, 0);
    gpio_export(digit2, false);

    //digit3 GPIO initialization

```

```

gpio_request(digit3, "sysfs");
gpio_direction_output(digit3, 0);
gpio_export(digit3, false);

//digit4 GPIO initialization
gpio_request(digit4, "sysfs");
gpio_direction_output(digit4, 0);
gpio_export(digit4, false);

//sega GPIO initialization
gpio_request(sega, "sysfs");
gpio_direction_output(sega, 1);
gpio_export(sega, false);

//segb GPIO initialization
gpio_request(segb, "sysfs");
gpio_direction_output(segb, 1);
gpio_export(segb, false);

//segc GPIO initialization
gpio_request(segc, "sysfs");
gpio_direction_output(segc, 1);
gpio_export(segc, false);

//segd GPIO initialization
gpio_request(seg, "sysfs");
gpio_direction_output(seg, 1);
gpio_export(seg, false);

//sege GPIO initialization
gpio_request(sege, "sysfs");
gpio_direction_output(sege, 1);
gpio_export(sege, false);

//segf GPIO initialization
gpio_request(segf, "sysfs");
gpio_direction_output(segf, 1);
gpio_export(segf, false);

//segg GPIO initialization
gpio_request(segg, "sysfs");
gpio_direction_output(segg, 1);
gpio_export(segg, false);

//LED GPIO initialization
gpio_request(led, "sysfs");
gpio_direction_output(led, led_state);
gpio_export(led, false);

//Switch GPIO initialization
gpio_request(switch_led, "sysfs");
gpio_direction_input(switch_led);
gpio_set_debounce(switch_led, 5000);
gpio_export(switch_led, false);
switch_irqnumber = gpio_to_irq(switch_led);

//Reset GPIO initialization
gpio_request(reset_counter, "sysfs");
gpio_direction_input(reset_counter);
gpio_set_debounce(reset_counter, 5000);
gpio_export(reset_counter, false);
reset_irqnumber = gpio_to_irq(reset_counter);

result = request_irq(switch_irqnumber, (irq_handler_t) switch_irq_handler,
(IRQF_TRIGGER_RISING|IRQF_TRIGGER_FALLING), "switch_irq_handler", NULL);

```



```

    result = request_irq(reset_irqnumber, (irq_handler_t) reset_irq_handler, IRQF_TRIGGER_RISING, "reset_irq_handler",
NULL);
    setup_timer(&seg_timer, _TimerHandler, 0);
    mod_timer(&seg_timer, jiffies + msecs_to_jiffies(seg_clock_refresh));
    //////////////////////////////////////
return result;
}
//Exit Function
static void __exit exitz(void){
    gpio_set_value(digit1, 0);
    gpio_set_value(digit2, 0);
    gpio_set_value(digit3, 0);
    gpio_set_value(digit4, 0);

    gpio_set_value(sega, 1);
    gpio_set_value(segb, 1);
    gpio_set_value(segc, 1);
    gpio_set_value(segd, 1);
    gpio_set_value(sege, 1);
    gpio_set_value(segf, 1);
    gpio_set_value(segg, 1);

    gpio_set_value(led, 0);

    free_irq(switch_irqnumber, NULL);
    free_irq(reset_irqnumber, NULL);

    gpio_unexport(digit1);
    gpio_unexport(digit2);
    gpio_unexport(digit3);
    gpio_unexport(digit4);

    gpio_unexport(sega);
    gpio_unexport(segb);
    gpio_unexport(segc);
    gpio_unexport(segd);
    gpio_unexport(sege);
    gpio_unexport(segf);
    gpio_unexport(segg);

    gpio_unexport(switch_led);
    gpio_unexport(reset_counter);
    gpio_unexport(led);

    gpio_free(digit1);
    gpio_free(digit2);
    gpio_free(digit3);
    gpio_free(digit4);

    gpio_free(sega);
    gpio_free(segb);
    gpio_free(segc);
    gpio_free(segd);
    gpio_free(sege);
    gpio_free(segf);
    gpio_free(segg);

    gpio_free(switch_led);
    gpio_free(reset_counter);
    gpio_free(led);

    del_timer(&seg_timer);

    printk(KERN_INFO "Goodbye from Counter\n");
}

```

```
//LED light function
static void lightswitch(int digit, int num){
```

```
    gpio_set_value(digit1, 0);
    gpio_set_value(digit2, 0);
    gpio_set_value(digit3, 0);
    gpio_set_value(digit4, 0);
```

```
    gpio_set_value(sega, 1);
    gpio_set_value(segb, 1);
    gpio_set_value(segc, 1);
    gpio_set_value(segd, 1);
    gpio_set_value(sege, 1);
    gpio_set_value(segf, 1);
    gpio_set_value(segg, 1);
```

```
    if (num == 1){
        gpio_set_value(segb, 0);
        gpio_set_value(segc, 0);
    }
```

```
    else if(num == 2){
        gpio_set_value(sega, 0);
        gpio_set_value(segb, 0);
        gpio_set_value(segd, 0);
        gpio_set_value(sege, 0);
        gpio_set_value(segg, 0);
    }
```

```
    else if(num == 3){
        gpio_set_value(sega, 0);
        gpio_set_value(segb, 0);
        gpio_set_value(segc, 0);
        gpio_set_value(segd, 0);
        gpio_set_value(segg, 0);
    }
```

```
    else if(num == 4){
        gpio_set_value(segb, 0);
        gpio_set_value(segc, 0);
        gpio_set_value(segf, 0);
        gpio_set_value(segg, 0);
    }
```

```
    else if(num == 5){
        gpio_set_value(sega, 0);
        gpio_set_value(segc, 0);
        gpio_set_value(segd, 0);
        gpio_set_value(segf, 0);
        gpio_set_value(segg, 0);
    }
```

```
    else if(num == 6){
        gpio_set_value(sega, 0);
        gpio_set_value(segc, 0);
        gpio_set_value(segd, 0);
        gpio_set_value(sege, 0);
        gpio_set_value(segf, 0);
        gpio_set_value(segg, 0);
    }
```

```
    else if(num == 7){
        gpio_set_value(sega, 0);
        gpio_set_value(segb, 0);
        gpio_set_value(segc, 0);
    }
```

```
    else if(num == 8){
        gpio_set_value(sega, 0);
        gpio_set_value(segb, 0);
        gpio_set_value(segc, 0);
        gpio_set_value(segd, 0);
        gpio_set_value(sege, 0);
    }
```

```

    gpio_set_value(segf, 0);
    gpio_set_value(segq, 0);
}
else if(num == 9){
    gpio_set_value(sega, 0);
    gpio_set_value(segb, 0);
    gpio_set_value(segc, 0);
    gpio_set_value(segf, 0);
    gpio_set_value(segq, 0);
}
else if(num == 0){
    gpio_set_value(sega, 0);
    gpio_set_value(segb, 0);
    gpio_set_value(segc, 0);
    gpio_set_value(segq, 0);
    gpio_set_value(sege, 0);
    gpio_set_value(segf, 0);
}
else{
    gpio_set_value(sega, 1);
    gpio_set_value(segb, 1);
    gpio_set_value(segc, 1);
    gpio_set_value(segq, 1);
    gpio_set_value(sege, 1);
    gpio_set_value(segf, 1);
    gpio_set_value(segq, 1);
}

if (digit == 1){
    gpio_set_value(digit1, 1);
}
else if (digit == 2){
    gpio_set_value(digit2, 1);
}
else if (digit == 3){
    gpio_set_value(digit3, 1);
}
else if (digit == 4){
    gpio_set_value(digit4, 1);
}
else{
    gpio_set_value(digit1, 0);
    gpio_set_value(digit2, 0);
    gpio_set_value(digit3, 0);
    gpio_set_value(digit4, 0);
}
}

//Timer handler that refreshes display
static void _TimerHandler(unsigned long data){
    getnstimeofday(&ts);
    if (led_state==1){
        msec_elap+=5;
    }
    if (whichdigit == 3){
        lightswitch(1, msec_elap/1000%10000/1000);
    }
    else if (whichdigit == 2){
        lightswitch(2, msec_elap/1000%1000/100);
    }
    else if (whichdigit == 1){
        lightswitch(3, msec_elap/1000%100/10);
    }
    else if(whichdigit == 0 ){
        lightswitch(4, msec_elap/1000%10);
    }
    else{

```

```

    lightswitch(1, 2);
}
whichdigit=(whichdigit+1)%4;

mod_timer(&seg_timer, jiffies + msecs_to_jiffies(seg_clock_refresh));
}
//IRQ for switch
static irq_handler_t switch_irq_handler(unsigned int irq, void *dev_id, struct pt_regs *regs){
    led_state=!led_state;
    gpio_set_value(led,led_state);
    printk("Light is now %d.Time is %d\n",led_state, msec_elap/1000);
    return (irq_handler_t) IRQ_HANDLED;
}
//IRQ for reset
static irq_handler_t reset_irq_handler(unsigned int irq, void *dev_id, struct pt_regs *regs){
    msec_elap = 0;
    printk("Reset now %d\n",msec_elap);
    return (irq_handler_t) IRQ_HANDLED;
}

module_init(initialz);
module_exit(exitz);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

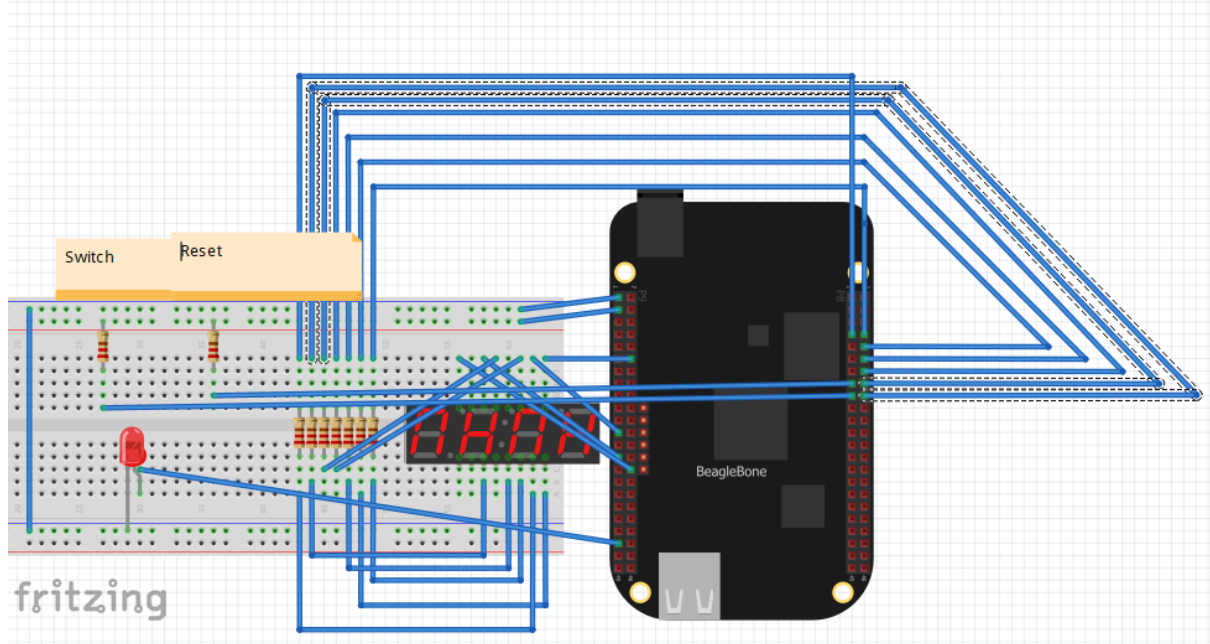


Figure 28: Schematic for counter.c

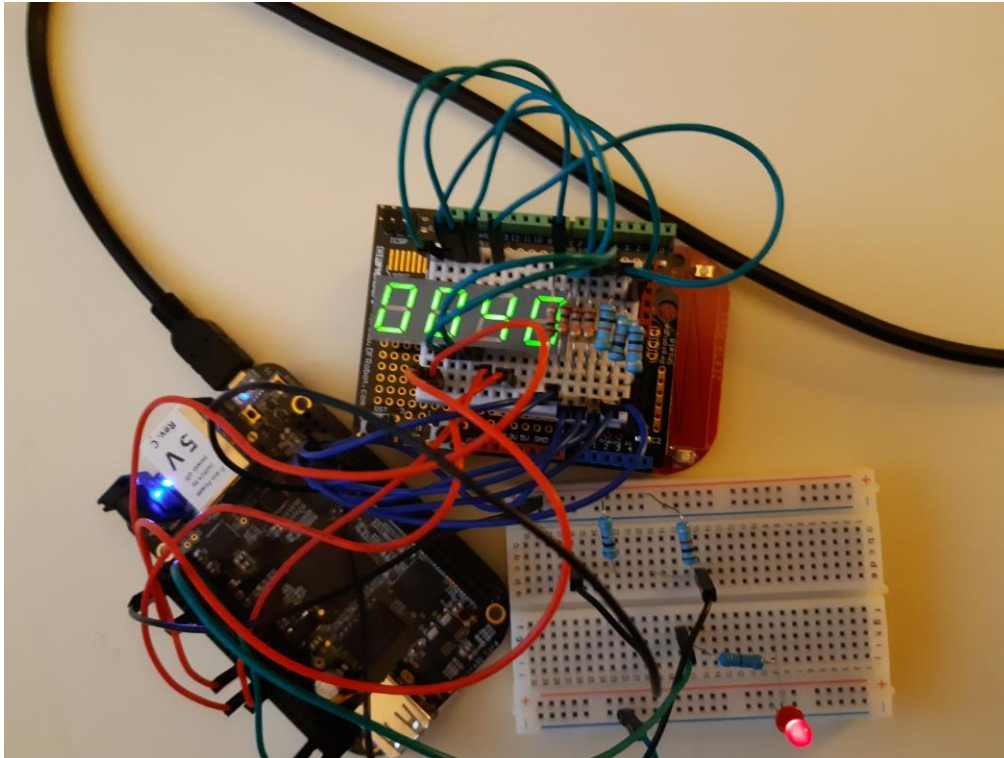


Figure 29: Counting the number of seconds the LED has been on.

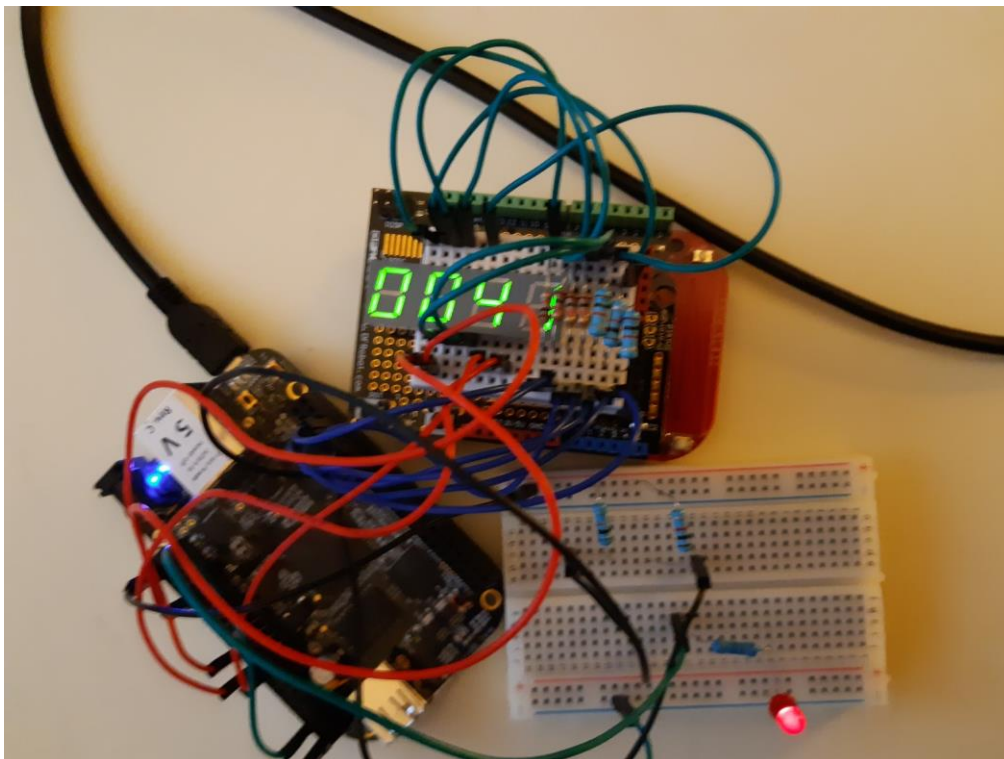


Figure 30: 1 Second after the previous picture

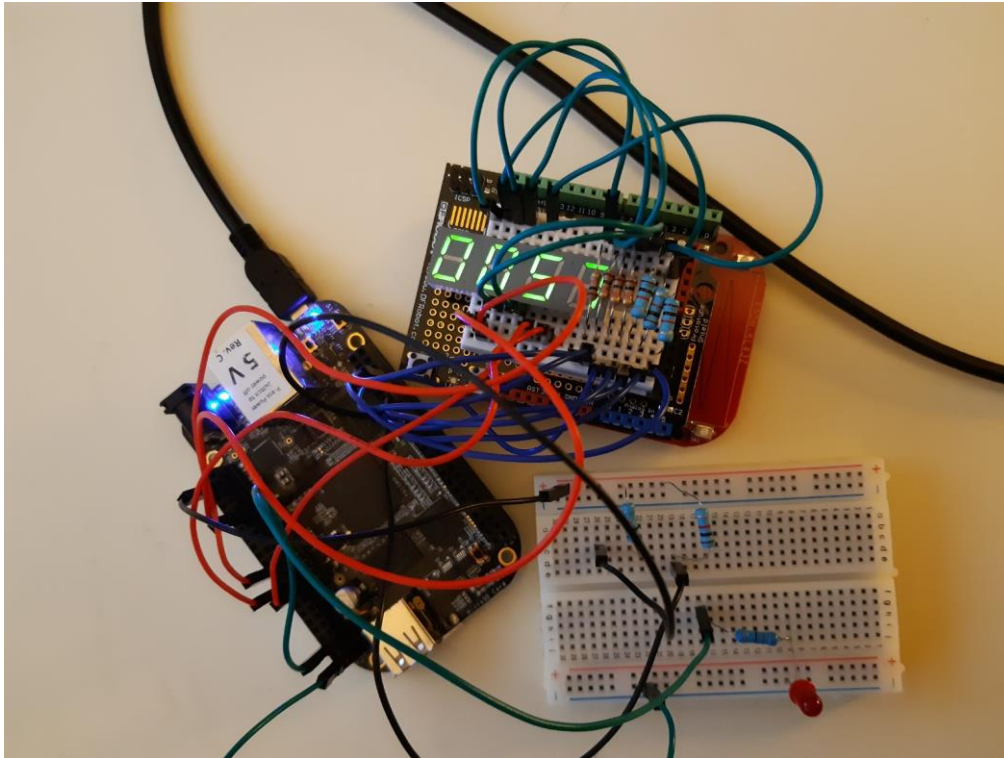


Figure 31: Timer stops at 57 seconds after switch is moved

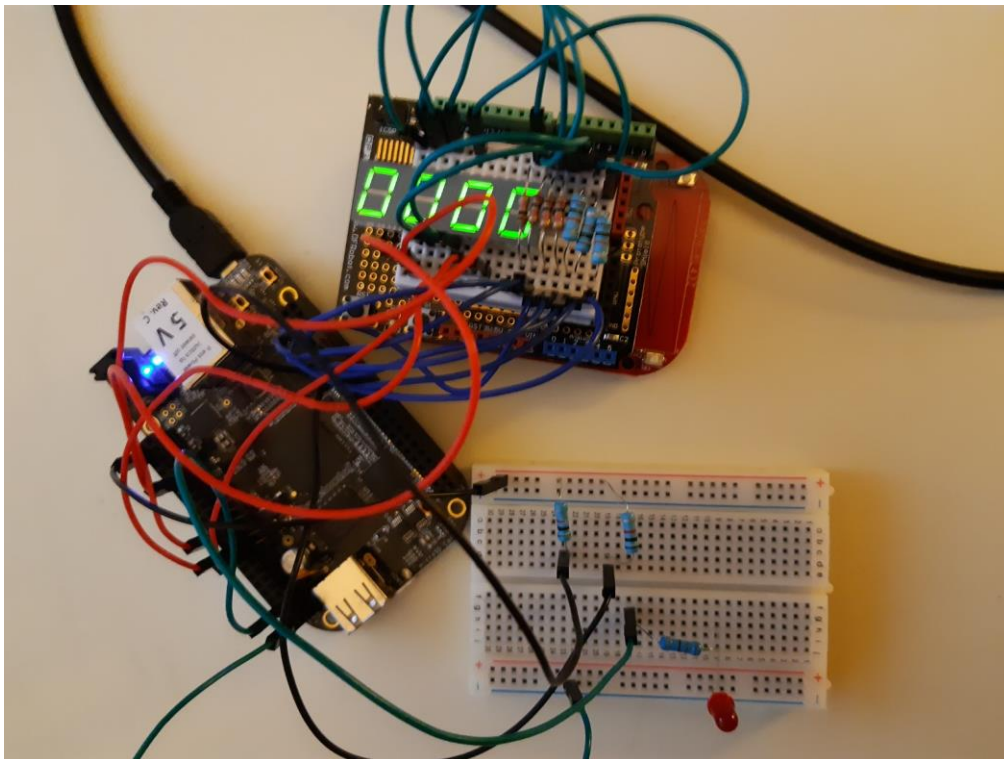


Figure 32: After reset button has been pressed.

D. Final Application (Duty Ratio Detector)

Purpose of the Application

In some applications, a Pulse Width Modulated Signal (PWM) might be needed to power a load (LED or DC Motor). To detect the duty-ratio (the percentage of the signal at which it is on), we can look at the LED and see how bright it is; the brighter the LED, the higher the duty ratio is. This observation however does not place a numerical value as to what the duty ratio is. We can also have the program producing the PWM signal to printf what the duty ratio is. However, this produces significant overhead because print statements require a lot of resources. To find the duty ratio, we will make a LKM to detect the duty-ratio of the signal. The steps to achieve this is explained below.

User Mode (PWM Generator and Controller)

By varying the duty ratio of the PWM signal, the brightness of a LED or the speed of DC motor can be varied. The BeagleBone Black Board has a couple PWM pins; as indicated by the green Pins on Figure 3: “Cape Expansion Headers”.

50% duty cycle



75% duty cycle



25% duty cycle



Figure 33: PWM Signal with varying duty ratios

We will use one of the PWM pins on the board to vary the brightness of the LED. A python program `pwm_gen.py` will be used to create the PWM signal.

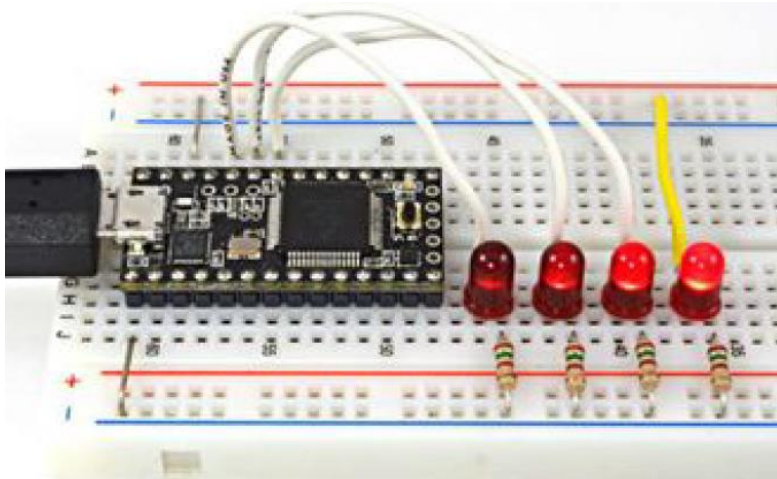


Figure 34: LEDs with different PWM signals.

To vary the duty ratio of the PWM signals, instead of the user entering a value thru the terminal, we will utilize a PS3 Controller to increase and decrease the duty ratio. We will use the triangle button to increase the duty ratio up to 100%, and the x button to decrease the duty ratio to 0%. We will use the `pwm_gen.py` program to read the values of the controller.



Figure 35: Image of PS3 controller. The triangle button to increase duty-ratio(up), and x button to decrease the duty-ratio(down).

When the controller is plugged in, the BeagleBone Black is able to recognize the PS3 controller (as shown by the message below after calling `dmesg | grep sony`). The PS3 controller drivers are already included in the Linux Kernel.

```
192.168.7.2 - PuTTY
root@beaglebone:/workspace# dmesg | grep sony
[ 7958.253118] sony 0003:054C:0268.0002: Fixing up Sony Sixaxis report descripto
r
[ 7958.297449] sony 0003:054C:0268.0002: input,hidraw0: USB HID v1.11 Joystick [
Sony PLAYSTATION(R)3 Controller] on usb-musb-hdrc.1.auto-1/input0
root@beaglebone:/workspace#
```

Figure 36: Message shown after the controller is plugged into the BeagleBone.

To read the values of the PS3 controller, a python program was made (from <https://stackoverflow.com/questions/16032982/getting-live-info-from-dev-input>). This reads the value from /dev/input/js0. Since the PS3 utilizes a gyro, and an accelerometer, an influx of inputs can be seen by the program. To only show the triangle and x inputs, the values were restricted to values that started with 268 (triangle) and 302 (x). Since they are mechanical buttons, they will bounce, causing the program to register multiple presses, even if 1 press was requested.

```
oot@beaglebone:/workspace/application# python pwm_gen.py
722787, 234946561, 1891, 11, 302173996)
722787, 419622326, 1900, 11, 302176699)
722796, 419622241, 1912, 11, 302182442)
722818, 302185144, 1932, 11, 302120960)
724267, 201392129, 3371, 11, 268612808)
724267, 419622156, 3381, 11, 268615511)
724277, 419622241, 3391, 11, 268619564)
724296, 268621929, 3412, 11, 268622605)
724328, 268615511, 3441, 11, 268604363)
724347, 201392128, 3451, 11, 268599297)
725238, 234946561, 4352, 11, 302158795)
725257, 302159808, 4371, 11, 302159133)
725287, 302158119, 4400, 11, 302153729)
726027, 268612808, 5141, 11, 268605714)
727187, 302170618, 6301, 11, 302171969)
727197, 419622241, 6311, 11, 302174672)
727228, 302171294, 6342, 11, 302165213)
727248, 234946560, 6352, 11, 302153729)
728037, 201392129, 7141, 11, 268610106)
731318, 234946561, 10422, 11, 302169267)
731328, 302169943, 10441, 11, 302170618)
731347, 234946560, 10451, 11, 302153729)
```

Figure 37: Values printed by the program. The values (far right) that start with 302 is the x button and values that start with 268 is a triangle. The multiple button presses in a short period of time is due to the button bounce.

A 10Hz signal with varying duty ratio was used; a 10 Hz signal was used, since 1/100th of 0.1 seconds is 1ms, so the duty ratio can easily be calculated by the detector by subtracting the fall time by the rise time.

The pwm_gen.c program can read the file where the PS3 controller actions are logged. It also creates a 10Hz signal with default duty ratio of 50%. If x is pressed on the controller, the duty ratio goes down by 1 each time (up to 0), and if a triangle is pressed the duty ratio goes up by 1 (up to 100).

pwm_gen.py

```
#####
import Adafruit_BBIO.GPIO as GPIO
import Adafruit_BBIO.PWM as PWM
import datetime
import sys
from time import sleep
import struct
```

#inspired by <https://stackoverflow.com/questions/16032982/getting-live-info-from-dev-input>

```

infile_path = "/dev/input/js0"
EVENT_SIZE = struct.calcsize("IIHHI")
file = open(infile_path, "rb")
event = file.read(EVENT_SIZE)

#initialize PWM pin
pwmpin="P9_42";
#initial duty cycle of 50
dutycycle=50;
PWM.start(pwmpin,dutycycle,10);
while event:
    (tv_sec, tv_usec, type, code, value) = struct.unpack("IIHHI", event)
    #if x is pressed
    if value/1000000 == 302:
        #print(struct.unpack("IIHHI",event))
        if dutycycle>0:
            dutycycle=dutycycle-1;
    #if y is pressed
    elif value/1000000 == 268:
        #print (struct.unpack("IIHHI",event))
        if dutycycle<100:
            dutycycle=dutycycle+1

    PWM.set_duty_cycle(pwmpin,dutycycle);
    event = file.read(EVENT_SIZE)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

Kernel Mode (PWM Duty Ratio Detector)

We will use a LKM to detect the duty ratio given off by the generator and display it on the seven segment display. For this, we will use 1 interrupt to detect both the rise and fall of the pulse. In its IRQ, we will find the time it takes for a pulse to rise, and the time it takes for it to fall, both in milliseconds. To find the duty ratio we simply subtract the two numbers; since we know for sure that it is a 10Hz signal, every millisecond it is on will correspond to 1% duty ratio.

The downside of using a LKM to detect the duty ratio is that the Kernel does not support floating-point operations, so the frequencies used by the PWM is limited. A 10Hz signal is used for convenience.

pwm_detect.c

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/gpio.h>
#include <linux/interrupt.h>
#include <linux/time.h>
#include <linux/timer.h>

MODULE_LICENSE("GPL");
//initialize GPIO Pins for 7-Seg
static unsigned int digit1 = 60;
static unsigned int digit2 = 49;
static unsigned int digit3 = 112;
static unsigned int digit4 = 115;

static unsigned int sega = 67;

```

```

static unsigned int segb = 68;
static unsigned int segc = 44;
static unsigned int segd = 26;
static unsigned int sege = 46;
static unsigned int segf = 65;
static unsigned int segg = 66;

//initialize Pin for interrupt
static unsigned int pwm_counter = 27;
static unsigned int pwm_counter_irqnumber;

//initialize variables needed for timer that refreshes LED display
static int seg_clock_refresh = 5;
struct timer_list seg_timer;
static int whichdigit = 0;

//variables that will hold the rise and fall time of pulse
static unsigned int pulse_rise_time = 0;
static unsigned int pulse_fall_time = 0;
static unsigned int dutycycle=0;

//prototype function for LED display
static void lightswitch(int digit, int num);
//Prototype variable for t5ms timer
static void _TimerHandler(unsigned long data);
//prototype the IRQ Handler of the button interrupt
static irq_handler_t pwm_counter_irq_handler(unsigned int irq, void *dev_id, struct pt_regs *regs);

//Initial Function
static int __init initialz(void){
    int result = 0;

    printk(KERN_INFO "Hello from Counter\n");

//INITIALIZE GPIO PINS
//digit1 GPIO initialization
    gpio_request(digit1, "sysfs");
    gpio_direction_output(digit1, 0);
    gpio_export(digit1, false);

//digit2 GPIO initialization
    gpio_request(digit2, "sysfs");
    gpio_direction_output(digit2, 0);
    gpio_export(digit2, false);

//digit3 GPIO initialization
    gpio_request(digit3, "sysfs");
    gpio_direction_output(digit3, 0);
    gpio_export(digit3, false);

//digit4 GPIO initialization
    gpio_request(digit4, "sysfs");
    gpio_direction_output(digit4, 0);
    gpio_export(digit4, false);

//sega GPIO initialization
    gpio_request(sega, "sysfs");
    gpio_direction_output(sega, 1);
    gpio_export(sega, false);

//segb GPIO initialization
    gpio_request(segb, "sysfs");
    gpio_direction_output(segb, 1);
    gpio_export(segb, false);

//segc GPIO initialization

```

```

    gpio_request(seg, "sysfs");
    gpio_direction_output(seg, 1);
    gpio_export(seg, false);

//segd GPIO initialization
    gpio_request(seg, "sysfs");
    gpio_direction_output(seg, 1);
    gpio_export(seg, false);

//sege GPIO initialization
    gpio_request(sege, "sysfs");
    gpio_direction_output(sege, 1);
    gpio_export(sege, false);

//segf GPIO initialization
    gpio_request(segf, "sysfs");
    gpio_direction_output(segf, 1);
    gpio_export(segf, false);

//segg GPIO initialization
    gpio_request(segg, "sysfs");
    gpio_direction_output(segg, 1);
    gpio_export(segg, false);

//Switch GPIO initialization
    gpio_request(pwm_counter, "sysfs");
    gpio_direction_input(pwm_counter);
    gpio_export(pwm_counter, false);
    pwm_counter_irqnumber = gpio_to_irq(pwm_counter);

    result = request_irq(pwm_counter_irqnumber, (irq_handler_t) pwm_counter_irq_handler,
(IRQF_TRIGGER_RISING|IRQF_TRIGGER_FALLING), "pwm_counter_irq_handler", NULL);

    setup_timer(&seg_timer, _TimerHandler, 0);
    mod_timer(&seg_timer, jiffies + msecs_to_jiffies(seg_clock_refresh));

return result;
}

//Exit function
static void __exit exitz(void){
//free Pins
    gpio_set_value(digit1, 0);
    gpio_set_value(digit2, 0);
    gpio_set_value(digit3, 0);
    gpio_set_value(digit4, 0);

    gpio_set_value(sega, 1);
    gpio_set_value(segb, 1);
    gpio_set_value(seg, 1);
    gpio_set_value(seg, 1);
    gpio_set_value(sege, 1);
    gpio_set_value(segf, 1);
    gpio_set_value(segg, 1);

    free_irq(pwm_counter_irqnumber, NULL);

    gpio_unexport(digit1);
    gpio_unexport(digit2);
    gpio_unexport(digit3);
    gpio_unexport(digit4);

    gpio_unexport(sega);
    gpio_unexport(segb);

```

```

gpio_unexport(seg);
gpio_unexport(segd);
gpio_unexport(sege);
gpio_unexport(segf);
gpio_unexport(segg);

gpio_unexport(pwm_counter);

gpio_free(digit1);
gpio_free(digit2);
gpio_free(digit3);
gpio_free(digit4);

gpio_free(sega);
gpio_free(segb);
gpio_free(segc);
gpio_free(seg);
gpio_free(sege);
gpio_free(segf);
gpio_free(segg);

gpio_free(pwm_counter);

del_timer(&seg_timer);

printf(KERN_INFO "Goodbye from Counter\n");
}

```

//7-Seg Function

```
static void lightswitch(int digit, int num){
```

```

    gpio_set_value(digit1, 0);
    gpio_set_value(digit2, 0);
    gpio_set_value(digit3, 0);
    gpio_set_value(digit4, 0);

```

```

    gpio_set_value(sega, 1);
    gpio_set_value(segb, 1);
    gpio_set_value(segc, 1);
    gpio_set_value(seg, 1);
    gpio_set_value(sege, 1);
    gpio_set_value(segf, 1);
    gpio_set_value(segg, 1);

```

```

    if (num == 1){
        gpio_set_value(segb, 0);
        gpio_set_value(segc, 0);
    }

```

```

    else if(num == 2){
        gpio_set_value(sega, 0);
        gpio_set_value(segb, 0);
        gpio_set_value(seg, 0);
        gpio_set_value(sege, 0);
        gpio_set_value(segg, 0);
    }

```

```

    else if(num == 3){
        gpio_set_value(sega, 0);
        gpio_set_value(segb, 0);
        gpio_set_value(segc, 0);
        gpio_set_value(seg, 0);
        gpio_set_value(segg, 0);
    }

```

```

    else if(num == 4){
        gpio_set_value(segb, 0);
    }

```

```

    gpio_set_value(seg_c, 0);
    gpio_set_value(seg_f, 0);
    gpio_set_value(seg_g, 0);
}
else if(num == 5){
    gpio_set_value(seg_a, 0);
    gpio_set_value(seg_c, 0);
    gpio_set_value(seg_d, 0);
    gpio_set_value(seg_f, 0);
    gpio_set_value(seg_g, 0);
}
else if(num == 6){
    gpio_set_value(seg_a, 0);
    gpio_set_value(seg_c, 0);
    gpio_set_value(seg_d, 0);
    gpio_set_value(seg_e, 0);
    gpio_set_value(seg_f, 0);
    gpio_set_value(seg_g, 0);
}
else if(num == 7){
    gpio_set_value(seg_a, 0);
    gpio_set_value(seg_b, 0);
    gpio_set_value(seg_c, 0);
}
else if(num == 8){
    gpio_set_value(seg_a, 0);
    gpio_set_value(seg_b, 0);
    gpio_set_value(seg_c, 0);
    gpio_set_value(seg_d, 0);
    gpio_set_value(seg_e, 0);
    gpio_set_value(seg_f, 0);
    gpio_set_value(seg_g, 0);
}
else if(num == 9){
    gpio_set_value(seg_a, 0);
    gpio_set_value(seg_b, 0);
    gpio_set_value(seg_c, 0);
    gpio_set_value(seg_f, 0);
    gpio_set_value(seg_g, 0);
}
else if(num == 0){
    gpio_set_value(seg_a, 0);
    gpio_set_value(seg_b, 0);
    gpio_set_value(seg_c, 0);
    gpio_set_value(seg_d, 0);
    gpio_set_value(seg_e, 0);
    gpio_set_value(seg_f, 0);
}
else{
    gpio_set_value(seg_a, 1);
    gpio_set_value(seg_b, 1);
    gpio_set_value(seg_c, 1);
    gpio_set_value(seg_d, 1);
    gpio_set_value(seg_e, 1);
    gpio_set_value(seg_f, 1);
    gpio_set_value(seg_g, 1);
}

if (digit == 1){
    gpio_set_value(digit1, 1);
}
else if (digit == 2){
    gpio_set_value(digit2, 1);
}
else if (digit == 3){
    gpio_set_value(digit3, 1);
}

```

```

}
else if (digit == 4){
    gpio_set_value(digit4, 1);
}
else{
    gpio_set_value(digit1, 0);
    gpio_set_value(digit2, 0);
    gpio_set_value(digit3, 0);
    gpio_set_value(digit4, 0);
}
}
//Timer interrupt handler to update display
static void _TimerHandler(unsigned long data){

    if (whichdigit == 3){
        lightswitch(1, dutycycle%10000/1000);
    }
    else if (whichdigit == 2){
        lightswitch(2, dutycycle%1000/100);
    }
    else if (whichdigit == 1){
        lightswitch(3, dutycycle%100/10);
    }
    else if(whichdigit == 0 ){
        lightswitch(4, dutycycle%10);
    }
    else{
        lightswitch(1, 2);
    }
    whichdigit=(whichdigit+1)%4;

    mod_timer(&seg_timer, jiffies + msecs_to_jiffies(seg_clock_refresh));
}

//Interrupt Handler for PWM signals
static irq_handler_t pwm_counter_irq_handler(unsigned int irq, void *dev_id, struct pt_regs *regs){
    if (gpio_get_value(pwm_counter)){
        pulse_rise_time=jiffies_to_msecs(jiffies);
        printk("UP UP UP\n");
    }
    else{
        pulse_fall_time=jiffies_to_msecs(jiffies);
        printk("DOWN DOWN\n");
    }
    if(pulse_fall_time>pulse_rise_time){
        dutycycle=pulse_fall_time-pulse_rise_time;
    }
    printk("Dutycycle is %d\n", dutycycle);
    return (irq_handler_t) IRQ_HANDLED;
}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

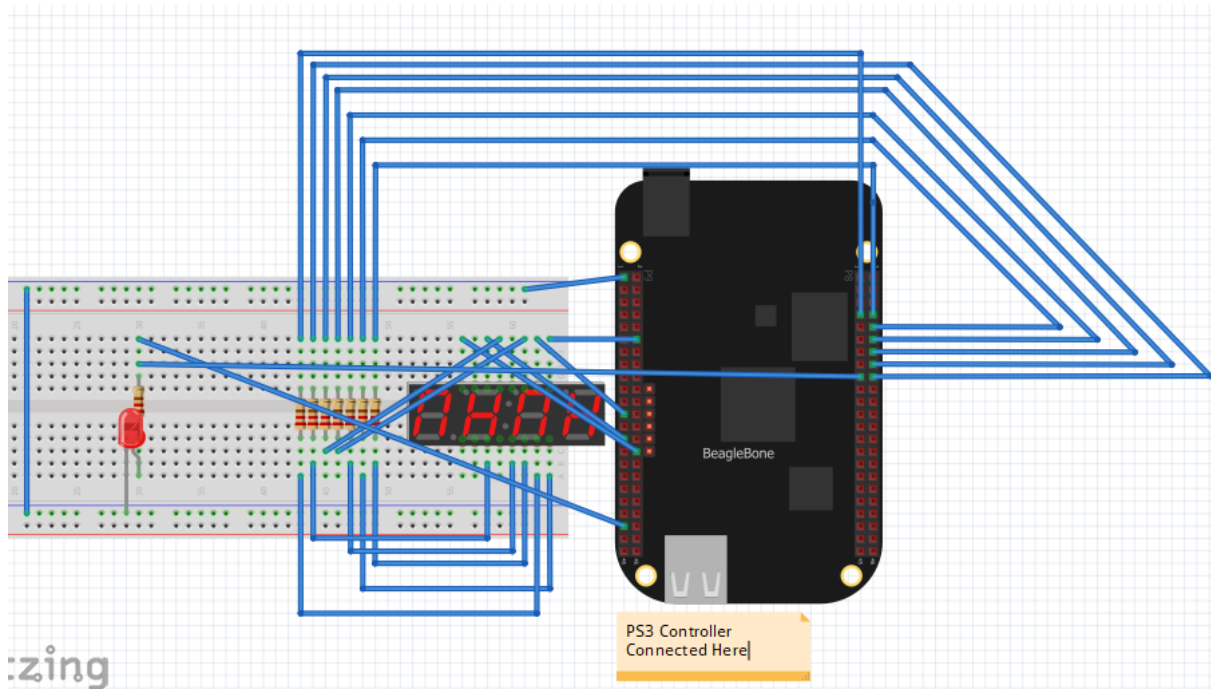



Figure 38: The schematic for the application.

Results

To run the two programs in tandem, the LKM `pwm_detect.c` will be loaded in first.

```
insmod pwm_detect.ko
```

Then the python program, `pwm_gen.py`, will be run.

```
python pwm_gen.py
```

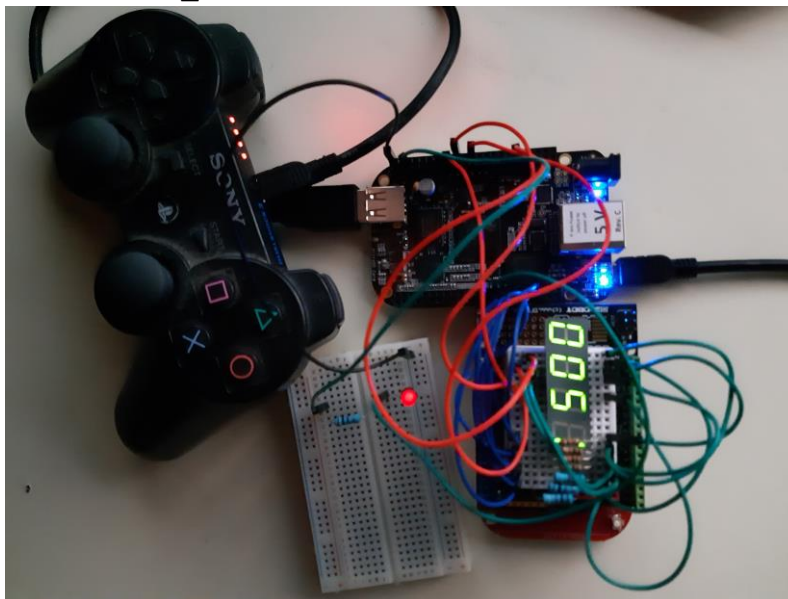


Figure 39: Image of circuit right after the python program is run. It should have a default value of 50%. LED shows around 49-51%.

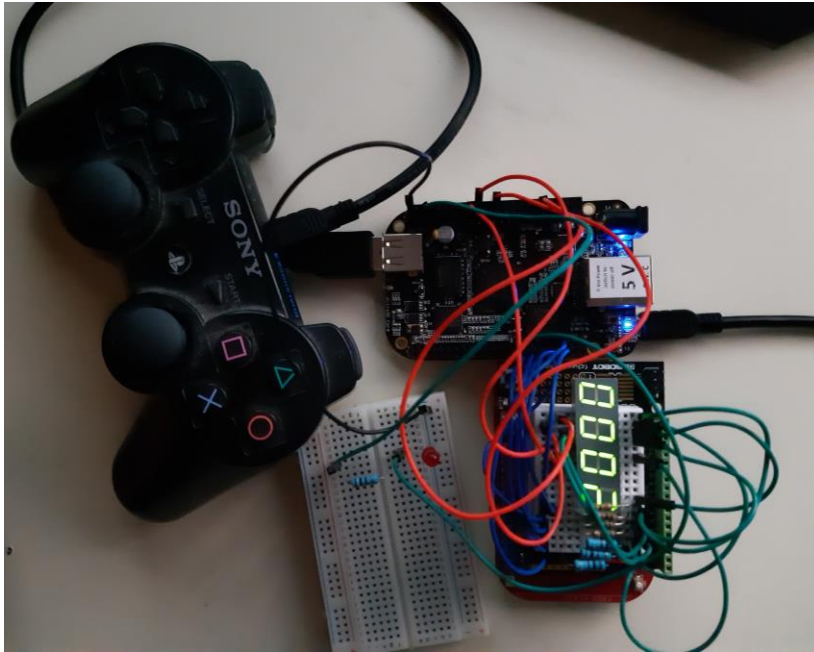


Figure 40: Image of circuit after x button has been pressed multiple times to lower the duty ratio to around 0%. LED displays 2%. Note that the LED is dimmer than the other 2 scenarios, and seems to be constantly off.

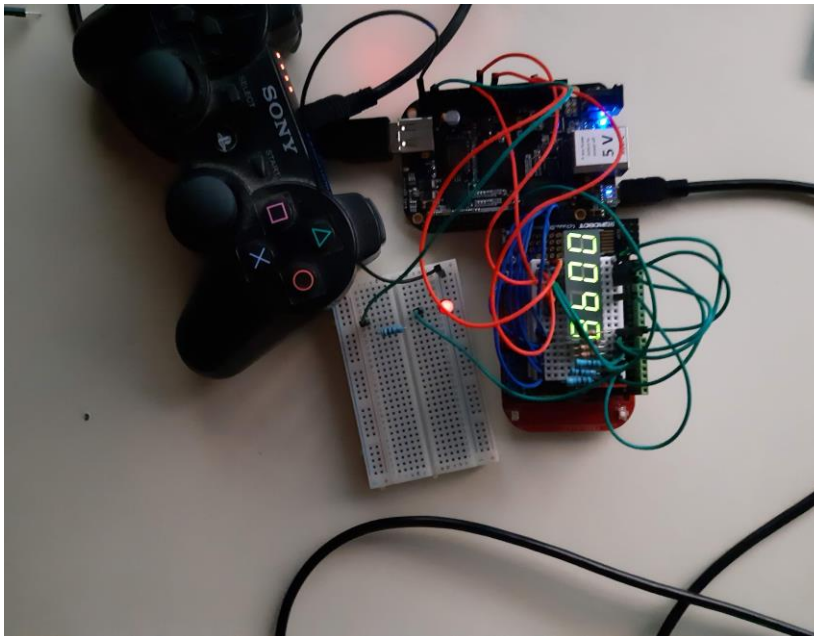


Figure 41: Image of circuit after triangle button has been pressed multiple times to increase the duty ratio to around 100%. LED displays 98%. Note that the LED is brighter than the other 2 scenarios, and seems to be constantly on.

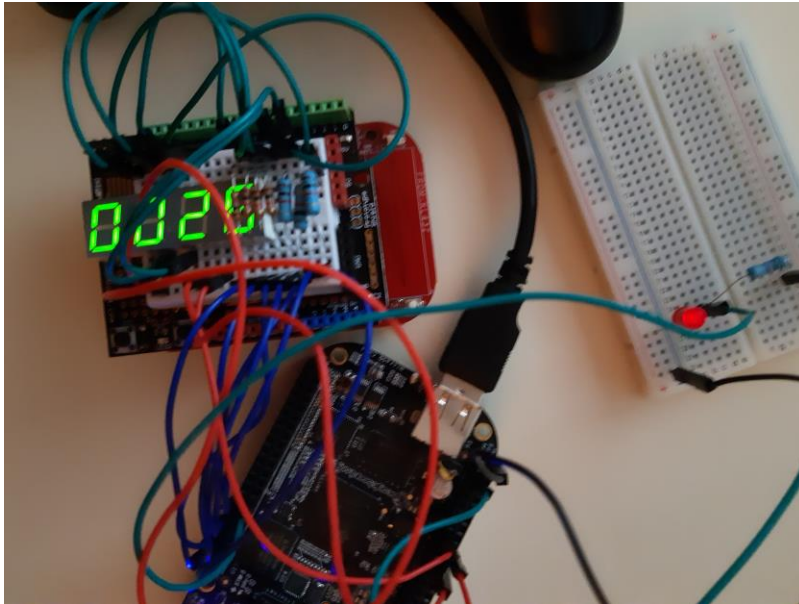


Figure 42: Image of circuit to have duty ratio of around 25%. Brightness is between 0 and 50% duty ratio

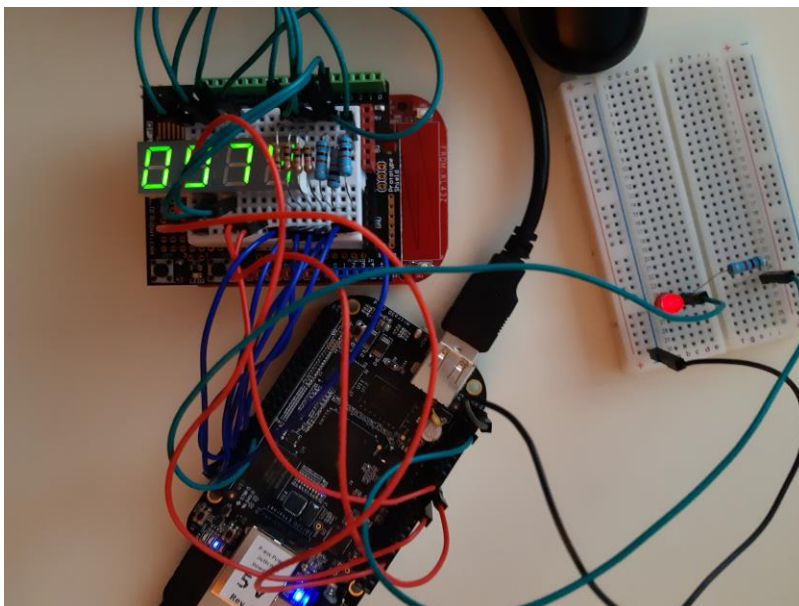


Figure 43: Image of circuit to have duty ratio of around 75%. Brightness is between 50 and 100% duty ratio

At default the LED should have a duty ratio of 50%, and the 7-Seg LED display shows around 49-51%. The deviation can be attributed to either one of these reasons or a combination of these reasons: the PWM not exactly having a duty ratio of 50% as intended, the detector not being able to detect the pulse rise and fall as exactly as it starts or ends, or conflict between the detector and generator from getting access to the CPU. Since it is a uniprocessor system, the two programs are competing for CPU time, so when the BeagleBone is transmitting, the BeagleBone can not read at the same time, so the duty ratio is not as precise as it can be. To circumvent this issue, we can get another BeagleBone to generate the PWM signal, and this BeagleBone to read it. The reported duty ratios on the 7 Segment display falls in line with the LED brightness.

Conclusion

Hopefully these experiments give the user a better understanding of how to interface the BeagleBone Black: design, compile, load, execute, and debug software, on the BeagleBone as well as use some of the hardware features like GPIO pins and PWM.

Troubles

1. When trying to make the Makefile associated with the hello_world LKM, the following error resulted:

```
make: *** /lib/modules/3.8.13/build: No such file or directory. Stop.  
make: *** [default] Error 2
```

To fix it, a symbolic link was made:

```
ln -s /usr/src/kernel /lib/modules/$(uname -r)/build
```

References

Writing a Linux Kernel Module - Part 1: Introduction. (n.d.). Retrieved from <http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/>

The Linux Kernel Module Programming Guide Retrieved from <http://www.tldp.org/LDP/lkmpg/2.6/html/>

Compiling a Kernel Module for BeagleBone (n.d.). Retrieved from <http://elene.dahners.com/blog/2012/10/22/compiling-a-kernel-module-for-beaglebone/>

Srikanta, A. (2016, July 23). How to Implement Periodic Timer in Linux Kernel. Retrieved from <https://qnaplus.com/how-to-implement-periodic-timer-in-linux-kernel/>

GitHub Link with codes: <https://github.com/niruyan1/beaglebonexercises>