

1 Linux Device Tree Overlay

1.1 Introduction

Since the adoption of the Device Tree standard to describe embedded Linux systems, there has been one major limitation: the static nature of device trees (please find more information on Linux device trees [here](#)<link to be added to previous article>). Device trees could not cope with changes in non-discoverable hardware, such as modifications to pin muxing, during run time. However, most modern embedded systems support adding and removing non-discoverable hardware during run time. This made it difficult to define the full hardware configuration statically at boot time in a device tree. Pantelis Antoniu, an active Linux kernel developer, implemented a solution to this issue using Device Tree Overlays (DTOs). The idea was to be able to dynamically insert a fragment of a device tree into a live device tree to update the hardware configuration of the system. For example, a fragment could change the status property of a device node from “disabled” to “okay” and then the device corresponding to that node would be created.

A device tree overlay is a file that consists of one or more device tree fragments that describe changes to the system hardware. This article will help you become familiar with device tree overlays by explaining the structure through an example, building a device tree overlay for the peripheral header of the Lesson 2 board, and then adapting the generic overlay for an example Click board.

1.2 Understanding Device Tree Overlays

The structure of a device tree overlay is a direct extension of a device tree. First, let's understand the structure by looking at the **PB-I2C1-MPU-9DOF-CLICK.dts** device tree overlay file. This overlay was written for the MPU 9DOF Click board which can be attached to the headers of the BeagleBoard.org ® PocketBeagle®. Since the PocketBeagle® uses the OSD335x-SM device (which is very similar to the OSD335x), it will be easy to leverage this overlay for the Lesson 2 board (The **PB-I2C1-MPU-9DOF-CLICK.dts** file is available for download [here](#)).

```
/*
 * Copyright (C) 2017 Robert Nelson <robertcnelson@gmail.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */
/dts-v1/;
/plugin/;

#include <dt-bindings/board/am335x-bbw-bbb-base.h>
```

```
#include <dt-bindings/gpio/gpio.h>
#include <dt-bindings/pinctrl/am33xx.h>

/ {
    fragment@0 {
        target = <&am33xx_pinmux>;
        __overlay__ {

            mpu9150_pins: pinmux_mpu9150_pins {
                pinctrl-single,pins = <
                    AM33XX_IOPAD(0x0824, PIN_INPUT | MUX_MODE7 )
/* (T10) gpmc_ad9.gpio0[23] INT */
                >;
            };
        };

    fragment@1 {
        target = <&ocp>;
        __overlay__ {

            P2_03_pinmux {
                status = "disabled";
            };
        };

    fragment@2 {
        target = <&i2c1>;
        __overlay__ {
            status = "okay";

            #address-cells = <1>;
            #size-cells = <0>;

            mpu9150@69 {
                compatible = "invensense,mpu9150";
                reg = <0x69>;
                interrupt-parent = <&gpio0>;
                interrupts = <23 1>;
                i2c-gate {
                    #address-cells = <1>;
                    #size-cells = <0>;
                    ax8975@c {
                        compatible = "ak,ak8975";
                        reg = <0x0c>;
                    };
                };
            };
        };
    };
};
```

The above overlay code consists of three major parts:

First, the overlay begins with tokens (shown below). The first token indicates the file version and the second one indicates that this file is a plugin (i.e. an overlay).

```
/dts-v1/;
/plugin/;
```

Next, the tokens are followed by include statements. This is where required header files are included in the overlay to add symbol definitions and macros.

```
#include <dt-bindings/board/am335x-bbw-bbb-base.h>
#include <dt-bindings/gpio/gpio.h>
#include <dt-bindings/pinctrl/am33xx.h>
```

Finally, the fragments describe the functional changes to the device tree for the overlay, starting from the root node (/):

The first fragment of the overlay (fragment@0) is used to set the **P2_03** pin of the PocketBeagle (i.e. the GPMC_AD9 pin, bit 23 of the GPIO0 peripheral of the AM335x) to **GPIO input mode**. When the MPU 9DOF Click board is connected to the PocketBeagle, the **INT** pin of the Click connects to the **P2_03** pin. Therefore, the pin must be configured so that the PocketBeagle can receive interrupt signals from the MPU 9DOF Click. To configure the pin, the **target** property is used to specify the device tree node that needs to be overlaid. In this case, the **am33xx_pinmux** node is specified to alter the pin muxing of the AM335x IO. The properties listed in the **__overlay__** node will be overlaid on the original properties of the target device tree node.

```
fragment@0 {
    target = <&am33xx_pinmux>;
    __overlay__ {
        mpu9150_pins: pinmux_mpu9150_pins {
            pinctrl-single,pins = <
                AM33XX_IOPAD(0x0824, PIN_INPUT | MUX_MODE7 )
/* (T10) gpmc_ad9.gpio0[23] INT */
            >;
        };
    };
};
```

The second fragment of the overlay (fragment@1) is then used to disable the current functionality associated with the **P2_03** pin. Currently in the PocketBeagle device tree, the **P2_03** pin is configured to use the **bone-pinmux-helper** driver in the **OCF** (On Chip Peripheral) node (see **am335x-pocketbeagle.dts**). Therefore, the driver must be disabled to ensure that the **P2_03** pin is available to receive interrupts from the Click board.

```
fragment@1 {
    target = <&ocp>;
    __overlay__ {

        P2_03_pinmux {
            status = "disabled";
        };

    };
};
```

The third fragment (fragment@2) is used to enable the I2C communication with the Click board. The Click board communicates over the I2C1 bus. Therefore, the I2C1 interface needs to be enabled by setting the **status** property of the **i2c1** node to **okay**. Also, all the required information of the IC on the MPU 9DOF Click board, such as the manufacturer name, IC name, device address on the I2C bus, etc. must be supplied in this fragment so the kernel can load the appropriate drivers. The device tree bindings for MPU-9150 IC on the MPU 9DOF Click can be found [here](#).

```
fragment@2 {
    target = <&i2c1>;
    __overlay__ {
        status = "okay";

        #address-cells = <1>;
        #size-cells = <0>;

        mpu9150@69 {
            compatible = "invensense,mpu9150";
            reg = <0x69>;
            interrupt-parent = <&gpio0>;
            interrupts = <23 1>;
            i2c-gate {
                #address-cells = <1>;
                #size-cells = <0>;
                ax8975@c {
                    compatible = "ak,ak8975";
                    reg = <0x0c>;
                };
            };
        };
    };
};
```

1.3 Generic Device Tree Overlay for the Peripheral Header

The Lesson 2 board has a peripheral header that supports removable Click boards. Similar to the device tree overlay in the previous section, a generic device tree overlay (*osd335x_L2_generic.dts*) can be built for the Lesson 2 board's peripheral header as shown below (The overlay can also be downloaded [here](#)).

The overlay has three parts like the previous example, however, there are more fragments to enable the different peripherals used by the header. The first fragment is used to configure the pin muxing for all the interfaces on the peripheral header such as SPI00, UART0, I2C1 and GPIO pins. The second, third and fourth fragments enable the I2C1, SPI0 and UART0 interfaces, respectively, and provide an outline to add device tree binding information corresponding to the type of click board that you will use.

Perk:

The device tree overlay we're building in this section is intended to be overlaid on the *osd335x-lesson2.dts* device tree which we built as part of the [Linux Device Tree](#) article. *osd335x-lesson2.dts* does not use *bone-pinmux-helper* driver. Hence, there is no need to disable the pinmux driver as in the example from the previous section.

```
/*
 * Copyright (C) 2017 2017 Octavo Systems - http://www.octavosystems.com/
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */
/dts-v1/;
/plugin/;

#include <dt-bindings/board/am335x-bbw-bbb-base.h>
#include <dt-bindings/gpio/gpio.h>
#include <dt-bindings/pinctrl/am33xx.h>

/ {

fragment@0 {
    target = <&am33xx_pinmux>;
    __overlay__ {

        pwm_pins: pinmux_pwm {
            pinctrl-single,pins = <
                AM33XX_IOPAD(0x0964, PIN_OUTPUT | MUX_MODE0) /*
(C18) ECAPO_IN_PWM0_OUT */
            >;
        };
    };
};
```

```
};

int_n_rst_pins: pinmux_int_rst {
    pinctrl-single,pins = <
        AM33XX_IOPAD(0x0824, PIN_INPUT | MUX_MODE7 ) /*
(T10) gpmc_ad9 MBUS_INT */
        AM33XX_IOPAD(0x0820, PIN_OUTPUT | MUX_MODE7 ) /*
(U10) gpmc_ad8 MBUS_RST */
    >;
};

i2c1_pins: pinmux_i2c1 {
    pinctrl-single,pins = <
        AM33XX_IOPAD(0x0968, PIN_INPUT_PULLUP | MUX_MODE3 )
/* (E18) I2C1_SDA/UART0_CTSN */
        AM33XX_IOPAD(0x096C, PIN_INPUT_PULLUP | MUX_MODE3 )
/* (E17) I2C1_SCL/UART0_RTSN */
    >;
};

spi0_pins: pinmux_spi0 {
    pinctrl-single,pins = <
        AM33XX_IOPAD(0x0950, PIN_INPUT | MUX_MODE0 ) /*
(A17) spi0_sclk.spi0_sclk */
        AM33XX_IOPAD(0x0954, PIN_INPUT | MUX_MODE0 ) /*
(B17) spi0_d0.spi0_d0 */
        AM33XX_IOPAD(0x0958, PIN_INPUT | MUX_MODE0 ) /*
(B16) spi0_d1.spi0_d1 */
        AM33XX_IOPAD(0x095c, PIN_INPUT | MUX_MODE0 ) /*
(A16) spi0_cs0.spi0_cs0 */
    >;
};

};

};

fragment@1 {

    target = <&i2c1>;
    __overlay__ {

        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&i2c1_pins>;

        /* Add device tree bindings
        * for your I2C1 device here.
        *
        */

    };
};
```

```
fragment@2 {

    target = <&spi0>;
    __overlay__ {

        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&spi0_pins>;

        /* Add device tree bindings
         * for your SPI device here.
         *
         */

    };
};
```

```
fragment@3 {

    target = <&uart0>;
    __overlay__ {

        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&uart0_pins>;

        /* Add device tree bindings
         * for your UART device here.
         *
         */

    };
};
```

1.4 Adapting the Generic Device Tree Overlay for a Specific Click Board

When you decide to use a specific Click board with the Lesson 2 board, it is straightforward to modify the generic device tree overlay created in the previous section, versus writing one from scratch, in order to support the Click board. Taking the MPU 9DOF Click example from Section 1.2, you would only need to add the MPU9150 device tree binding information under fragment@1 since MPU 9DOF Click uses the I2C bus for communication. The new device tree overlay (*osd335x_L2_generic_i2c1.dts*) would look like this (content that is different from generic DTO is highlighted):

```
/*
 * Copyright (C) 2017 Octavo Systems - http://www.octavosystems.com/
 *
 * This program is free software; you can redistribute it and/or modify
```

```
* it under the terms of the GNU General Public License version 2 as
* published by the Free Software Foundation.
*/
/dts-v1/;
/plugin/;

#include <dt-bindings/board/am335x-bbw-bbb-base.h>
#include <dt-bindings/gpio/gpio.h>
#include <dt-bindings/pinctrl/am33xx.h>

/ {

fragment@0 {
    target = <&am33xx_pinmux>;
    __overlay__ {

        pwm_pins: pinmux_pwm {
            pinctrl-single,pins = <
                AM33XX_IOPAD(0x0964, PIN_OUTPUT | MUX_MODE0 ) /*
(C18) ECAP0_IN_PWM0_OUT */
            >;
        };

        int_n_rst_pins: pinmux_int_rst {
            pinctrl-single,pins = <
                AM33XX_IOPAD(0x0824, PIN_INPUT | MUX_MODE7 ) /*
(T10) gpmc_ad9 MBUS_INT */
                AM33XX_IOPAD(0x0820, PIN_OUTPUT | MUX_MODE7 ) /*
(U10) gpmc_ad8 MBUS_RST */
            >;
        };

        i2c1_pins: pinmux_i2c1 {
            pinctrl-single,pins = <
                AM33XX_IOPAD(0x0968, PIN_INPUT_PULLUP | MUX_MODE3 )
/* (E18) I2C1_SDA/UART0_CTSN */
                AM33XX_IOPAD(0x096C, PIN_INPUT_PULLUP | MUX_MODE3 )
/* (E17) I2C1_SCL/UART0_RTSN */
            >;
        };

        spi0_pins: pinmux_spi0 {
            pinctrl-single,pins = <
                AM33XX_IOPAD(0x0950, PIN_INPUT | MUX_MODE0 ) /*
(A17) spi0_sclk.spi0_sclk */
                AM33XX_IOPAD(0x0954, PIN_INPUT | MUX_MODE0 ) /*
(B17) spi0_d0.spi0_d0 */
                AM33XX_IOPAD(0x0958, PIN_INPUT | MUX_MODE0 ) /*
(B16) spi0_d1.spi0_d1 */
                AM33XX_IOPAD(0x095c, PIN_INPUT | MUX_MODE0 ) /*
(A16) spi0_cs0.spi0_cs0 */
            >;
        };
    };
};
```


OSD335x Lesson 2: Linux Device Tree Overlay

Rev.1 5/30/2018



```

};

};

fragment@1 {
    target = <&i2c1>;
    __overlay__ {

        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&i2c1_pins>;

        #address-cells = <1>;
        #size-cells = <0>;

        mpu9150@69 {
            compatible = "invensense,mpu9150";
            reg = <0x69>;
            interrupt-parent = <&gpio0>;
            interrupts = <23 1>;
            i2c-gate {
                #address-cells = <1>;
                #size-cells = <0>;
                ax8975@c {
                    compatible = "ak,ak8975";
                    reg = <0x0c>;
                };
            };
        };
    };
};

fragment@2 {
    target = <&spi0>;
    __overlay__ {

        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&spi0_pins>;

        /* Add device tree bindings
         * for your SPI device here.
         *
         */

    };
};

```

```
fragment@3 {  
  
    target = <uart0>;  
    __overlay__ {  
  
        status = "okay";  
        pinctrl-names = "default";  
        pinctrl-0 = <uart0_pins>;  
  
        /* Add device tree bindings  
        * for your UART device here.  
        *  
        */  
  
    };  
};  
};
```

You can also directly download the *osd335x_L2_generic_i2c1.dts* device tree overlay file [here](#).

1.5 Building and using a Device Tree Overlay

The process to compile a device tree overlay is similar to the process to compile a device tree as discussed in Section 1.3 of [Linux Device Tree](#) article. For the OSD335x Family of devices, Robert Nelson's *Device Tree Rebuilder* can be used to build the DTO. You can download it [here](#). The steps to build the DTO are as follows:

- Copy the *osd335x_L2_generic_i2c1.dts* file to the *dtb-rebuilder/src/arm/* directory.
- Build the file using *make* command.
- Once the build process completes, the *osd335x_L2_generic_i2c1.dtb* file will be available in the *dtb-rebuilder/src/arm/* directory. Change the extension of the file to *.dtbo* (i.e., *osd335x_L2_generic_i2c1.dtbo*) to make sure the linux kernel recognizes it as an overlay.
- Copy the *osd335x_L2_generic_i2c1.dtbo* file to the */lib/firmware* directory (You will have to run this command as root)
- Open the *uEnv.txt* file in the */boot* directory using your favorite text editor and set *custom_cape = osd335x_L2_generic_i2c1.dtbo* as shown in Figure . Save and close the file.

OSD335x Lesson 2: Linux Device Tree Overlay

Rev.1 5/30/2018



```
mymac — debian@beaglebone: ~ — ssh 192.168.7.2 -l debian — 112x35
GNU nano 2.7.4 File: /boot/uEnv.txt

#Docs: http://elinux.org/Beagleboard:U-boot_partitioning_layout_2.0
#
uname_r=4.9.45-ti-r57
#uuid=
#dtb=osd3358-bsm-refdesign.dtb
dtb=osd335x-lesson2.dtb

###U-Boot Overlays###
###Documentation: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian#U-Boot_Overlays
###Master Enable
enable_uboot_overlays=1
###
###Override capes with eeprom
#uboot_overlay_addr0=/lib/firmware/<file0>.dtbo
#uboot_overlay_addr1=/lib/firmware/<file1>.dtbo
#uboot_overlay_addr2=/lib/firmware/<file2>.dtbo
#uboot_overlay_addr3=/lib/firmware/<file3>.dtbo
###
###Additional custom capes
#uboot_overlay_addr4=/lib/firmware/<file4>.dtbo
#uboot_overlay_addr5=/lib/firmware/<file5>.dtbo
#uboot_overlay_addr6=/lib/firmware/<file6>.dtbo
#uboot_overlay_addr7=/lib/firmware/<file7>.dtbo
###
###Custom Cape
dtb_overlay=/lib/firmware/osd335x_L2_generic_i2c1.dtbo
#dtb_overlay=/lib/firmware/osd335x_L2_generic.dtbo
###
###Disable auto loading of virtual capes (emmc/video/wireless/adc)
#disable_uboot_overlay_emmc=1

[ Read 71 lines ]
^G Get Help      ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos      ^Y Prev Page
^X Exit          ^R Read File    ^\ Replace      ^U Uncut Text   ^T To Spell     ^_ Go To Line    ^V Next Page
```

Figure 1 Setting Device Tree Overlay

- f. Reboot the board. If you monitor the boot messages on the UART0 serial console, it will display the name of the device tree and device tree overlays that were loaded during boot-up as shown in Figure 1.

U-Boot SPL 2017.09-rc2-00002-g7c9353e752 (Aug 31 2017 - 09:25:14)
Trying to boot from MMC2

U-Boot 2017.09-rc2-00002-g7c9353e752 (Aug 31 2017 - 09:25:14 -0500), Build: jenkins-github_Bootloader-Builder-596

```
CPU : AM335X-GP rev 2.1
I2C:  ready
DRAM:  512 MiB
No match for driver 'omap_hsmmc'
No match for driver 'omap_hsmmc'
Some drivers were not found
Reset Source: Global external warm reset has occurred.
Reset Source: Power-on reset has occurred.
MMC:  OMAP SD/MMC: 0, OMAP SD/MMC: 1
Using default environment

Board: BeagleBone Black
<ethaddr> not set. Validating first E-fuse MAC
BeagleBone Black:
BeagleBone: cape eeprom: i2c_probe: 0x54:
BeagleBone: cape eeprom: i2c_probe: 0x55:
BeagleBone: cape eeprom: i2c_probe: 0x56:
BeagleBone: cape eeprom: i2c_probe: 0x57:
Net:   eth0: MII MODE
Could not get PHY for cpsw: addr 0
cpsw
Press SPACE to abort autoboot in 2 seconds
board_name=[A335BNLT] ...
board_rev=[] ...
switch to partitions #0, OK
mmc0 is current device
SD/MMC found on device 0
** Bad device 0:2 0x82000000 **
** Bad device 0:2 0x82000000 **
switch to partitions #0, OK
mmc0 is current device
Scanning mmc 0:1...
gpio: pin 56 (gpio 56) value is 0
gpio: pin 55 (gpio 55) value is 0
gpio: pin 54 (gpio 54) value is 0
gpio: pin 53 (gpio 53) value is 1
switch to partitions #0, OK
mmc0 is current device
gpio: pin 54 (gpio 54) value is 1
Checking for: /uEnv.txt ...
Checking for: /boot.scr ...
Checking for: /boot/boot.scr ...
Checking for: /boot/uEnv.txt ...
gpio: pin 55 (gpio 55) value is 1
2158 bytes read in 43 ms (48.8 KiB/s)
Loaded environment from /boot/uEnv.txt
debug: [dtb=osd335x-lesson2.dtb] ...
Using: dtb=osd335x-lesson2.dtb ...
Checking if uname_r is set in /boot/uEnv.txt...
gpio: pin 56 (gpio 56) value is 1
Running uname_boot ...
loading /boot/vmlinuz-4.9.45-ti-r57 ...
9464664 bytes read in 631 ms (14.3 MiB/s)
uboot_overlays: dtb=osd335x-lesson2.dtb in /boot/uEnv.txt, unable to use [uboot_base_dtb=am335x-boneblack-uboot.dtb] ...
loading /boot/dtbs/4.9.45-ti-r57/osd335x-lesson2.dtb ...
49963 bytes read in 167 ms (292 KiB/s)
uboot_overlays: [fdt_buffer=0x60000] ...
uboot_overlays: loading /lib/firmware/AM335X-20-00A0.dtb ...
378 bytes read in 81 ms (3.9 KiB/s)
uboot_overlays: loading /lib/firmware/BB-BONE-eMMC1-01-00A0.dtb ...
1105 bytes read in 193 ms (4.9 KiB/s)
uboot_overlays: loading /lib/firmware/BB-HDMI-TDA998x-00A0.dtb ...
4169 bytes read in 295 ms (13.7 KiB/s)
uboot_overlays: loading /lib/firmware/BB-ADC-00A0.dtb ...
695 bytes read in 203 ms (2.9 KiB/s)
uboot_overlays: loading /lib/firmware/AM335X-PRU-UIO-00A0.dtb ...
853 bytes read in 114 ms (6.8 KiB/s)
uboot_overlays: [dtb_overlay=lib/firmware/osd335x_L2_generic_i2c1.dtb] ...
```

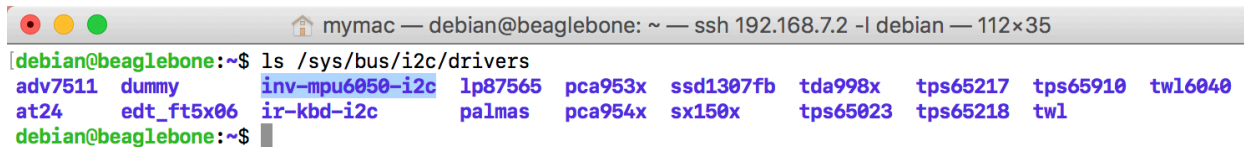
Figure 1 Serial Console Boot Messages

We can now login to Debian (default username = debian, password = temppwd).

1.6 Checking if the Device Tree Overlay works as intended

If the new DTO works as intended, the Linux kernel should recognize the MPU 9DOF Click and load appropriate drivers. From the command line, we can check if appropriate driver has been loaded for the IC on the MPU 9DOF by listing the i2c bus drivers (as shown in Figure 2) using the command:

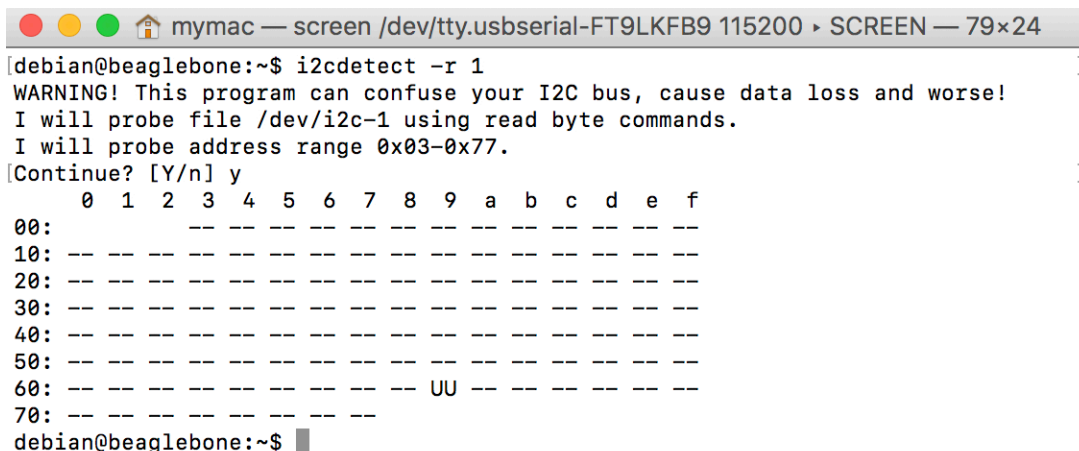
```
ls /sys/bus/i2c/drivers
```



```
mymac — debian@beaglebone: ~ — ssh 192.168.7.2 -l debian — 112x35
[debian@beaglebone:~$ ls /sys/bus/i2c/drivers
adv7511 dummy inv-mpu6050-i2c lp87565 pca953x ssd1307fb tda998x tps65217 tps65910 twl6040
at24 edt_ft5x06 ir-kbd-i2c palmas pca954x sx150x tps65023 tps65218 twl
debian@beaglebone:~$
```

Figure 2 Checking if driver for 9DOF Click was automatically loaded

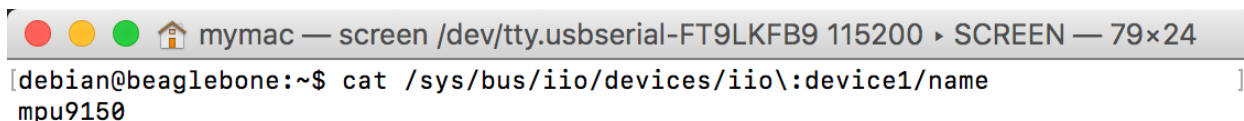
If the InvenSense MPU6050 I2C driver appears, you know that the device was properly configured and the correct driver was loaded. Also, given a kernel driver was loaded for MPU 9DOF click, we should see **UU** (Unit Unavailable) on the I2C1 bus at address 0x69 when the **i2cdetect** command is used to scan the I2C1 bus for devices as shown in Figure 3. When a kernel driver manages a device, it will not allow **i2cdetect** to probe the device. Therefore, you will see **UU** when scanned versus **69**, if the correct driver was not loaded.



```
mymac — screen /dev/tty.usbserial-FT9LKF9 115200 ▸ SCREEN — 79x24
[debian@beaglebone:~$ i2cdetect -r 1
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-1 using read byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n] y
  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- UU -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
debian@beaglebone:~$
```

Figure 3 Detecting Click board on I2C bus

Finally, we can check if the MPU 9DOF Click appears as an IIO (Industrial IO) device by searching for IIO devices in IIO devices directory as shown in Figure 4.



```
mymac — screen /dev/tty.usbserial-FT9LKF9 115200 ▸ SCREEN — 79x24
[debian@beaglebone:~$ cat /sys/bus/iio/devices/iio\:device1/name
mpu9150
-
```

Figure 4 Checking if MPU 9DOF IC is recognized

If all the above checks succeed, the device is working properly and you can directly access the accelerometer, compass and other values from the MPU 9DOF Click using Sysfs (more on Sysfs [here](#)). Most of the readable registers of the MPU 9DOF Click are available as files under the IIO device directory for the Click (**iio:device1**, in the picture below. However, the device number maybe different in your system) as shown in Figure 5.

```
mymac — screen /dev/tty.usbserial-FT9LKFB9 115200 ▸ SCREEN — 125x17
[debian@beaglebone:~$ ls /sys/bus/iio/devices/iio\:device1
buffer          in_accel_x_calibbias  in_anglvel_scale      in_anglvel_z_raw      power
current_timestamp_clock  in_accel_x_raw        in_anglvel_scale_available  in_gyro_matrix         sampling_frequency
dev             in_accel_y_calibbias  in_anglvel_x_calibbias  in_temp_offset        sampling_frequency_available
in_accel_matrix  in_accel_y_raw        in_anglvel_x_raw       in_temp_raw           scan_elements
in_accel_mount_matrix  in_accel_z_calibbias  in_anglvel_y_calibbias  in_temp_scale         subsystem
in_accel_scale    in_accel_z_raw        in_anglvel_y_raw       name                  trigger
in_accel_scale_available  in_anglvel_mount_matrix  in_anglvel_z_calibbias  of_node              uevent
]
[debian@beaglebone:~$ cat /sys/bus/iio/devices/iio\:device1/in_accel_x_raw
366
]
[debian@beaglebone:~$ cat /sys/bus/iio/devices/iio\:device1/in_anglvel_y_raw
5
]
[debian@beaglebone:~$ ]
```

Figure 5 Reading MPU 9DOF files and output values using Sysfs

For any questions or concerns, you can reach us at:

<https://octavosystems.com/forums/>
