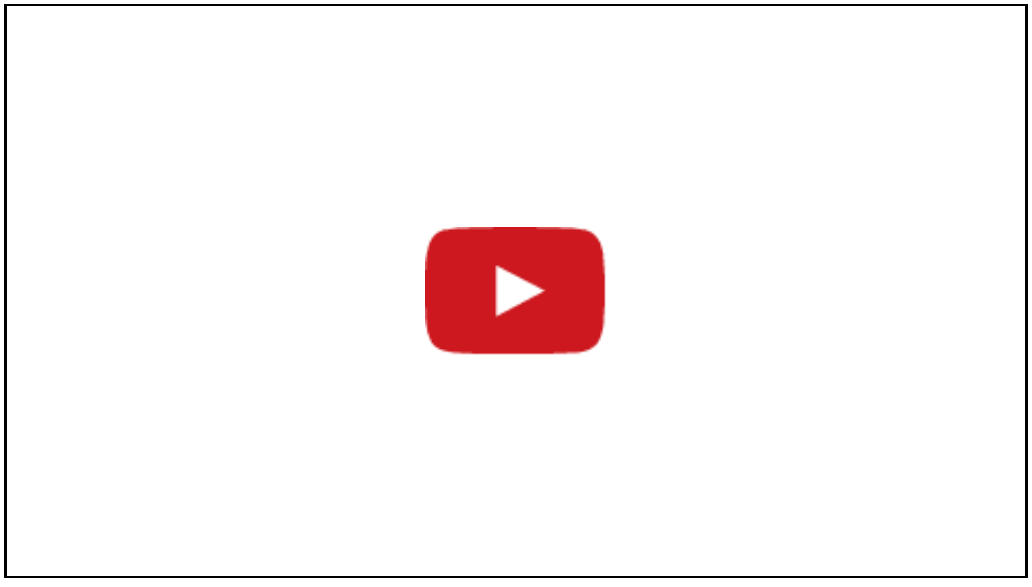
GPIOs on the Beaglebone Black using the Device Tree Overlays

‹ Previous Next ›



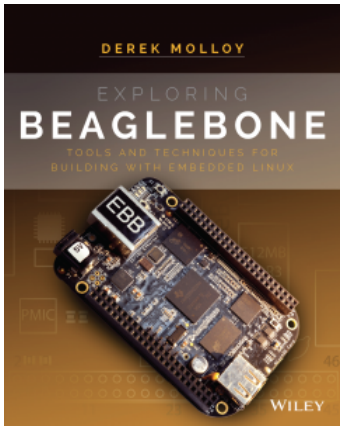
GPIOs on the Beaglebone Black using the Device Tree Overlays

> *This post provides supplementary information to the video that I have just posted on using GPIOs with the Beaglebone Black.*

Derek M

F

Search ...



My new book on the BeagleBone. See: www.exploringbeaglebone.cc for further information. Available from Dec'2014 and Jan'2015.

The Video

In this video I am going to continue my series on the Beaglebone by demonstrating how to use its GPIOs for both input and output applications. In this video I will wire simple input and output circuits that are attached to two GPIOs – one that lights an LED and the other that receives a button input. I covered this topic below before in a previous video. I am updating it here because there have been significant changes to the Linux kernel. This video will cover the Linux device tree for ARM embedded systems and explain how you can create custom device tree overlays to configure the GPIOs for your applications at run time from within the Linux userspace. I will explain the use of internal and external pullup and pulldown resistors and I will make available and describe a set of C++ code examples for reading and writing to the Beaglebone's GPIOs. I have also built a set of PDF tables that aggregate the information that you need and make it easier to configure GPIOs on your Beaglebone's P8 and P9 headers.

The code for this video is available by typing:

```
git clone git://github.com/derekmolloy/boneDeviceTree.git
```

The information below is covered in the video, but here it is just in case you need to get a text view:

Getting started

We can get some information about the pins in use:

```
root@beaglebone:/sys/kernel/debug/pinctrl/44e10800.pinmux# cat
registered pin groups:
group: pinmux_userled_pins
pin 21 (44e10854)
pin 22 (44e10858)
pin 23 (44e1085c)
pin 24 (44e10860)

group: pinmux_rstctl_pins
pin 20 (44e10850)

group: pinmux_i2c0_pins
pin 98 (44e10988)
pin 99 (44e1098c)

group: pinmux_i2c2_pins
pin 94 (44e10978)
pin 95 (44e1097c)

group: pinmux_emmc2_pins
pin 32 (44e10880)
pin 33 (44e10884)
pin 0 (44e10800)
pin 1 (44e10804)
```

Tags

Analog Discovery angstrom avconv beaglebone beaglebone black bitbake build building c++ C920 connman cpu curl disk distribution easydriver ffmpeg flash Flip-Flop git gparted https iso Java kernel LED LEDs linux live-cd Logic Analyzer nmap opencv portscan re-size resize RTP source stepper motor UDP vdi Video virtualbox VLC Wordpress x264

Categories

- Analog
- Beaglebone
- Blog
- Digital Electronics
- Embedded Systems
- General
- Linux
- Main Blog
- Raspberry PI
- Tools

```
pin 2 (44e10808)
pin 3 (44e1080c)
pin 4 (44e10810)
```

We can also get information about which pins are in use (allocated):

```
root@beaglebone:/sys/kernel/debug/pinctrl/44e10800.pinmux# ca
Pinmux settings per pin
Format: pin (name): mux_owner gpio_owner hog?
pin 0 (44e10800): mmc.5 (GPIO UNCLAIMED) function pinmux_emmc
pin 1 (44e10804): mmc.5 (GPIO UNCLAIMED) function pinmux_emmc
pin 2 (44e10808): mmc.5 (GPIO UNCLAIMED) function pinmux_emmc
pin 3 (44e1080c): mmc.5 (GPIO UNCLAIMED) function pinmux_emmc
pin 4 (44e10810): mmc.5 (GPIO UNCLAIMED) function pinmux_emmc
pin 5 (44e10814): mmc.5 (GPIO UNCLAIMED) function pinmux_emmc
pin 6 (44e10818): mmc.5 (GPIO UNCLAIMED) function pinmux_emmc
pin 7 (44e1081c): mmc.5 (GPIO UNCLAIMED) function pinmux_emmc
pin 8 (44e10820): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 9 (44e10824): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 10 (44e10828): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 11 (44e1082c): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 12 (44e10830): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 13 (44e10834): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 14 (44e10838): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 15 (44e1083c): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 16 (44e10840): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 17 (44e10844): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 18 (44e10848): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 19 (44e1084c): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 20 (44e10850): rstctl.3 (GPIO UNCLAIMED) function pinmux_
pin 21 (44e10854): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 22 (44e10858): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 23 (44e1085c): (MUX UNCLAIMED) (GPIO UNCLAIMED)
```

And a full list of the pins:

```
root@beaglebone:/sys/kernel/debug/pinctrl/44e10800.pinmux# ca
registered pins: 142
pin 0 (44e10800) 00000031 pinctrl-single
pin 1 (44e10804) 00000031 pinctrl-single
pin 2 (44e10808) 00000031 pinctrl-single
pin 3 (44e1080c) 00000031 pinctrl-single
pin 4 (44e10810) 00000031 pinctrl-single
pin 5 (44e10814) 00000031 pinctrl-single
pin 6 (44e10818) 00000031 pinctrl-single
pin 7 (44e1081c) 00000031 pinctrl-single
pin 8 (44e10820) 00000027 pinctrl-single
pin 9 (44e10824) 00000027 pinctrl-single
pin 10 (44e10828) 00000027 pinctrl-single
pin 11 (44e1082c) 00000027 pinctrl-single
pin 12 (44e10830) 00000027 pinctrl-single
pin 13 (44e10834) 00000027 pinctrl-single
pin 14 (44e10838) 00000027 pinctrl-single
pin 15 (44e1083c) 00000027 pinctrl-single
pin 16 (44e10840) 00000027 pinctrl-single
pin 17 (44e10844) 00000027 pinctrl-single
pin 18 (44e10848) 00000027 pinctrl-single
pin 19 (44e1084c) 00000027 pinctrl-single
pin 20 (44e10850) 00000017 pinctrl-single
pin 21 (44e10854) 00000007 pinctrl-single
```

› Uncategorized

Recent Posts

› Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux

› Analog Discovery: Getting Started with the Analog and Digital Tools

› Analog Discovery: Getting Started with the Logic Analyzer and Digital Pattern Generator

› Beaglebone: Controlling the on-board LEDs using Java

› Beaglebone: Controlling the on-board LEDs using C++

```
25 pin 22 (44e10858) 00000017 pinctrl-single
26 pin 23 (44e1085c) 00000007 pinctrl-single
```

Setting up the Circuit

Using the same circuit as in my old video. Since each GPIO module provides 32 dedicated GPIOs (general purpose input/output) and the GPIOs support 4 banks of 32 GPIOs (so, 128 GPIOs in total) the naming of GPIO0_5, would be GPIO 5 as (0 x32 + 5 = 5)

(Pin 12 on the P9 Header) GPIO1_28 – The LED = 1 x 32 + 28 = GPIO 60 (Offset 0x078, P9-12 GPIO1_28) #88

NOTE: GPIO 60 is not PIN 60!!!

If we check pins again and search for pin 60, by using the offset we can see:

```
1 root@beaglebone:/sys/kernel/debug/pinctrl/44e10800.pinmux# more
2 pin 30 (44e10878) 00000030 pinctrl-single
```

The pin mode on pin 30 is 30 HEX. What does that mean?

Well to understand this you need the document to beat all documents – the AM3359 Technical Reference Manual. http://www.ti.com/product/am3359 and you can see the link for this document. The version I am using is called the “AM335x ARM Cortex-A8 Microprocessors (MPUs) Technical Reference Manual (Rev.H). It is a 18.5MB document with 4,727 pages (no typo there – 4,700 pages!). The current direct link is: http://www.ti.com/lit/ug/spruh73j/spruh73j.pdf

The GPIOs section is Chapter 25 and begins on page 4,056. The page I am most interested in is Page 815, Section 9.3.51. You can search the PDF for “conf_<module>” if you are using a different version of the document.

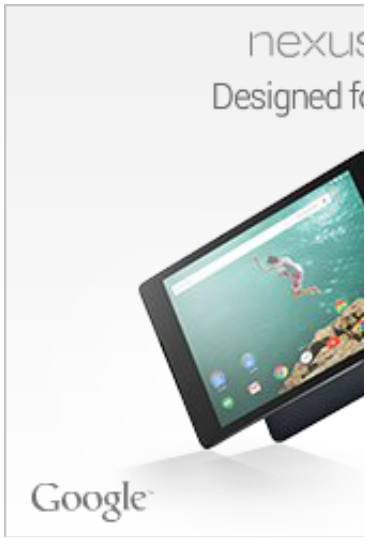
The value: 0x30 Hex is 110000 in Binary, so what does that mean?

Well, you have to see the table:

Bit	Field	Reset	Description
6	conf_<module>_<pin>_slewctrl	X	Slew Control. Slew Rate: Fast is 0, Slow is 1

Archives

- June 2014
- January 2014
- December 2013
- November 2013
- October 2013
- July 2013
- June 2013
- May 2013
- April 2013



5	conf_<module>_<pin>_rxactive	1h	Receiver Active. Input Enable: Receiver Disable 0, Receiver Enable 1
4	conf_<module>_<pin>_putypesel	X	Pad Pullup/Pulldown Type. Pulldown is 0, Pullup is 1
3	conf_<module>_<pin>_puden	X	Pad Pullup/Pulldown enable. Enabled is 0, Disabled is 1
2-0	conf_<module>_<pin>_mmode	X	Mode. Pad functional mux select. A number between 0 and 7 i.e. 000 and 111. This depends on which mode we require.

Well if you look at this table, you see that 0x30 means the slew rate is fast, the receiver is enabled, the pad is set for pullup and pullup is enabled. The Mode is 0. which means when you look a the table for this pin on the P9 header (Table 8. Expansion Header P9 Pinout), you see that pin is set as:

Beaglebone P9 Header, Pin 12, Mode 0 is gpmc_be1n, we would like to set it to Mode 7, which is gpio1[28] (Note the LED is currently on).

27 means 100111 = Fast, Enable Receiver, Pulldown type, enabled, mux mode 7.

37 means 110111 = Fast, Enable Receiver, Pullup type, enabled, mux mode 7.

Be careful, not all pins work in this way and there are external resistors on the board that affect the behaviour. For example, pins GPIO2_6 to GPIO2_14 all have external 42.2k resistors to GND and 100k resistors to high.

We can export the pins by echoing the GPIO number to **/sys/class/gpio/export**

```
root@beaglebone:/sys/class/gpio# ls -al
total 0
drwxr-xr-x  2 root root    0 Jan  1  2000 .
drwxr-xr-x 48 root root    0 Jan  1  2000 ..
--w-------  1 root root 4096 Jun  1 22:21 export
lrwxrwxrwx  1 root root    0 Jan  1  2000 gpiochip0 -> ../../d
lrwxrwxrwx  1 root root    0 Jan  1  2000 gpiochip32 -> ../../
lrwxrwxrwx  1 root root    0 Jan  1  2000 gpiochip64 -> ../../
lrwxrwxrwx  1 root root    0 Jan  1  2000 gpiochip96 -> ../../
--w-------  1 root root 4096 Jan  1  2000 unexport
root@beaglebone:/sys/class/gpio# echo 117 > export
root@beaglebone:/sys/class/gpio# echo 60 > export
root@beaglebone:/sys/class/gpio# ls -al
total 0
drwxr-xr-x  2 root root    0 Jan  1  2000 .
drwxr-xr-x 48 root root    0 Jan  1  2000 ..
--w-------  1 root root 4096 Jun  1 22:22 export
lrwxrwxrwx  1 root root    0 Jun  1 22:22 gpio117 -> ../../dev
lrwxrwxrwx  1 root root    0 Jun  1 22:22 gpio60 -> ../../devi
lrwxrwxrwx  1 root root    0 Jan  1  2000 gpiochip0 -> ../../d
lrwxrwxrwx  1 root root    0 Jan  1  2000 gpiochip32 -> ../../
lrwxrwxrwx  1 root root    0 Jan  1  2000 gpiochip64 -> ../../
lrwxrwxrwx  1 root root    0 Jan  1  2000 gpiochip96 -> ../../
--w-------  1 root root 4096 Jan  1  2000 unexport
root@beaglebone:/sys/class/gpio# cd gpio60
root@beaglebone:/sys/class/gpio/gpio60# ls -al
total 0
drwxr-xr-x 3 root root    0 Jun  1 22:22 .
drwxr-xr-x 8 root root    0 Jan  1  2000 ..
-rw-r--r-- 1 root root 4096 Jun  1 22:24 active_low
-rw-r--r-- 1 root root 4096 Jun  1 22:24 direction
-rw-r--r-- 1 root root 4096 Jun  1 22:24 edge
drwxr-xr-x 2 root root    0 Jun  1 22:24 power
lrwxrwxrwx 1 root root    0 Jun  1 22:24 subsystem -> ../../..
-rw-r--r-- 1 root root 4096 Jun  1 22:22 uevent
-rw-r--r-- 1 root root 4096 Jun  1 22:24 value
root@beaglebone:/sys/class/gpio/gpio60# echo "out" > direction
root@beaglebone:/sys/class/gpio/gpio60# cat direction
out
root@beaglebone:/sys/class/gpio/gpio60# echo 0 > value
root@beaglebone:/sys/class/gpio/gpio60# echo 1 > value
root@beaglebone:/sys/class/gpio/gpio60# echo 0 > value
root@beaglebone:/sys/class/gpio/gpio60# echo 1 > value
```

Light goes on and off.

The Beaglebone Black System Reference Manual (SRM) is available at: http://circuitco.com/support/index.php?title=BeagleBoneBlack

Page 65 of the SRM has the table that you need to map the GPIO to the Offset!

GPIO1_28 maps to P9-12 with an offset of 160 (pin 88 44e10960)

Setup for Device Tree Overlays

https://github.com/jadonk/validation-scripts/blob/master/test-capemgr/README.md

As is described in this guide:

```
root@beaglebone:/lib/firmware# export SLOTS=/sys/devices/bone_c
root@beaglebone:/lib/firmware# export PINS=/sys/kernel/debug/pi
```

We now have $SLOTS and $PINS that we can echo

```
root@beaglebone:/lib/firmware# echo $SLOTS
/sys/devices/bone_capemgr.9/slots
root@beaglebone:/lib/firmware# echo $PINS
/sys/kernel/debug/pinctrl/44e10800.pinmux/pins
```

But we can also cat these values, for example cat $SLOTS:

```
root@beaglebone:~/temp# cat $SLOTS
 0: 54:PF---
 1: 55:PF---
 2: 56:PF---
 3: 57:PF---
 4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC
 5: ff:P-O-L Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HD
```

Just so that I always have these environment variables I am adding them to my .profile. So my ~/.profile looks like this:

```
export SLOTS=/sys/devices/bone_capemgr.9/slots
export PINS=/sys/kernel/debug/pinctrl/44e10800.pinmux/pins
export CURL_CA_BUNDLE=/etc/ssl/certs/ca-certificates.crt
```

Which includes the environment variable to set up the certs to fix the configuration issue for curl as discussed here: Git and Curl SSL Certificates Configuration on Beaglebone Black

If we wish to set these values now, without typing them twice we can use the '.' so:

```
root@beaglebone:~# . ~/.profile
root@beaglebone:~# echo $CURL_CA_BUNDLE
/etc/ssl/certs/ca-certificates.crt
```

And you can see that the variables have been set.

Using an Overlay

Overlays allow the initial device tree that was described at boot to be modified in userspace at run time. This is useful as we are able to enable any device without having to recompile the kernel and/or reboot. When you enable output using the pinmux settings, you're only enabling the output driver circuitry at the pin. When you change the mux, you're selecting which internal signal gets connected to this pins output driver. So, the pin mux (physical pin) is completely separate from the

gpio block (internal signal). You have to enable both.

In this overlay example I am using the bone-pinmux-helper to enable the pins. The GPIO is treated as a separate peripheral, just like all other peripherals.

```
root@beaglebone:~/boneDeviceTree/overlay# ls -al
total 16
drwxr-xr-x 2 root root 4096 Jun  6 23:59 .
drwxr-xr-x 6 root root 4096 Jan  1  2000 ..
-rw-r--r-- 1 root root 1129 Jun  6 23:56 DM-GPIO-Test.dts
-rwxr-xr-x 1 root root  124 Jun  6 23:29 build
root@beaglebone:~/boneDeviceTree/overlay# more ./build
#!/bin/bash

echo "Compiling the overlay from .dts to .dtbo"

dtc -O dtb -o DM-GPIO-Test-00A0.dtbo -b 0 -@ DM-GPIO-Test.dts
root@beaglebone:~/boneDeviceTree/overlay# ./build
Compiling the overlay from .dts to .dtbo
root@beaglebone:~/boneDeviceTree/overlay# ls -al
total 20
drwxr-xr-x 2 root root 4096 Jun  6 23:59 .
drwxr-xr-x 6 root root 4096 Jan  1  2000 ..
-rw-r--r-- 1 root root  952 Jun  6 23:59 DM-GPIO-Test-00A0.dtb
-rw-r--r-- 1 root root 1129 Jun  6 23:56 DM-GPIO-Test.dts
-rwxr-xr-x 1 root root  124 Jun  6 23:29 build
root@beaglebone:~/boneDeviceTree/overlay# cp DM-GPIO-Test-00A0
```

Now, note when you echo DM-GPIO-Test > $SLOTS, make sure that you don't pass DM-GPIO-Test-00A0.dtbo

```
root@beaglebone:~# cd /lib/firmware/
root@beaglebone:/lib/firmware# cat $SLOTS
 0: 54:PF---
 1: 55:PF---
 2: 56:PF---
 3: 57:PF---
 4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMM
 5: ff:P-O-L Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-H
root@beaglebone:/lib/firmware# echo DM-GPIO-Test > $SLOTS
root@beaglebone:/lib/firmware# cat $SLOTS
 0: 54:PF---
 1: 55:PF---
 2: 56:PF---
 3: 57:PF---
 4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMM
 5: ff:P-O-L Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-H
 6: ff:P-O-L Override Board Name,00A0,Override Manuf,DM-GPIO-T
root@beaglebone:/lib/firmware#
```

Checking the pins (for example, pins 88 and 85):

```
root@beaglebone:/lib/firmware# cat $PINS |grep 960
pin 88 (44e10960) 00000007 pinctrl-single
root@beaglebone:/lib/firmware# cat $PINS |grep 954
pin 85 (44e10954) 00000027 pinctrl-single
root@beaglebone:/lib/firmware#
```

1020 is a20 – remember that it is in hexadecimal.

Now if we type **dmesg**, we can see the impact of this operation:

```
[   62.334146] bone-capemgr bone_capemgr.9: part_number 'DM-GPI
[   62.334223] bone-capemgr bone_capemgr.9: slot #6: generic ov
[   62.334242] bone-capemgr bone_capemgr.9: bone: Using overrid
[   62.334260] bone-capemgr bone_capemgr.9: slot #6: 'Override
[   62.334363] bone-capemgr bone_capemgr.9: slot #6: Requesting
[   62.334381] bone-capemgr bone_capemgr.9: slot #6: Requesting
[   62.338787] bone-capemgr bone_capemgr.9: slot #6: dtbo 'DM-G
[   62.338970] bone-capemgr bone_capemgr.9: slot #6: #2 overlay
[   62.342899] bone-capemgr bone_capemgr.9: slot #6: Applied #2
```

Now we can work with the GPIOs directly:

```
root@beaglebone:/sys/class/gpio# echo 60 > export
root@beaglebone:/sys/class/gpio# echo 49 > export
root@beaglebone:/sys/class/gpio# ls -al
total 0
drwxr-xr-x  2 root root    0 Jan  1  2000 .
drwxr-xr-x 48 root root    0 Jan  1  2000 ..
--w-------  1 root root 4096 Jun  7 14:00 export
lrwxrwxrwx  1 root root    0 Jun  7 14:00 gpio49 -> ../../devi
lrwxrwxrwx  1 root root    0 Jun  7 14:00 gpio60 -> ../../devi
lrwxrwxrwx  1 root root    0 Jan  1  2000 gpiochip0 -> ../../d
lrwxrwxrwx  1 root root    0 Jan  1  2000 gpiochip32 -> ../../
lrwxrwxrwx  1 root root    0 Jan  1  2000 gpiochip64 -> ../../
lrwxrwxrwx  1 root root    0 Jan  1  2000 gpiochip96 -> ../../
--w-------  1 root root 4096 Jan  1  2000 unexport
root@beaglebone:/sys/class/gpio# cd gpio49
root@beaglebone:/sys/class/gpio/gpio49# ls
active_low  direction  edge  power  subsystem  uevent  value
root@beaglebone:/sys/class/gpio/gpio49# echo "in" > direction
root@beaglebone:/sys/class/gpio/gpio49# cat direction
in
root@beaglebone:/sys/class/gpio/gpio49# cat value
0
root@beaglebone:/sys/class/gpio/gpio49# cat value
0
root@beaglebone:/sys/class/gpio/gpio49# cat value
0
root@beaglebone:/sys/class/gpio/gpio49# cat value
0
root@beaglebone:/sys/class/gpio/gpio49# cat value
1
root@beaglebone:/sys/class/gpio/gpio49# cat value
1
root@beaglebone:/sys/class/gpio/gpio49# cat value
1
root@beaglebone:/sys/class/gpio/gpio49# cat value
0
root@beaglebone:/sys/class/gpio/gpio49# cat value
0
root@beaglebone:/sys/class/gpio/gpio49#
```

All is in order.

The C++ Code

All of the C++ code is available in the gpio directory of the github repository. The description of this code and its use can be found in the video.

Citation

If you use this video in your research, please cite:

> *Molloy, D. [DerekMolloyDCU]. (2012, May, 3). Beaglebone: GPIO Programming on ARM Embedded Linux [Video file]. Retrieved from http://www.youtube.com/watch?v=SaIpz0…*

Further Reading:

Understanding the GPIOs: https://www.kernel.org/doc/Documentation/gpio.txt

Understanding Overlays: https://github.com/jadonk/validation-scripts/blob/master/test-capemgr/README.md

Understanding the Device Tree: http://devicetree.org/Device_Tree_Usage

By Derek | June 12th, 2013 | Beaglebone, Blog | 81 Comments

About the Author: Derek

Dr. Derek Molloy is a Senior Lecturer in the School of Electronic Engineering, Faculty of Engineering & Computing at Dublin City University, Ireland. He lectures in Object-oriented Programming,