



Intel® NUC

Programming for the Custom Solutions Header

Whitepaper

V1.0

September 2016

Intel NUC models DE3215TYx, NUC5i3MYx and NUC5i5MYx may contain design defects or errors known as errata that may cause the product to deviate from published specifications. Current characterized errata, if any, are documented in their respective Intel NUC Specification Updates.

Revision History

Revision	Revision History	Date
1.0	Initial release of the Intel NUC Custom Solutions Header Programming Whitepaper	September 2016

Disclaimer

This product specification applies to only the standard Intel NUC models DE3815TYx, NUC5i3MYx and NUC5i5MYx.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

All Intel NUC Boards are evaluated as Information Technology Equipment (I.T.E.) for use in personal computers (PC) for installation in homes, offices, schools, computer rooms, and similar locations. The suitability of this product for other PC or embedded non-PC applications or other environments, such as medical, industrial, alarm systems, test equipment, etc. may not be supported without further evaluation by Intel.

Intel Corporation may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families: Go to:

[Learn About Intel® Processor Numbers](#)

Intel NUC may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

Intel, the Intel logo, Intel NUC and Intel Core are trademarks of Intel Corporation in the U.S. and/or other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2016 Intel Corporation. All rights reserved.

Table of Contents

Contents

1.0	Introduction.....	5
2.0	Hardware	6
2.1	Intel NUC DE3815TY.....	6
2.2	Intel NUCs NUC5i3MY and NUC5i5MY	7
3.1	BIOS Setup	8
3.2	Operating System Driver Setup	8
3.2.1	Microsoft Windows I ² C/GPIO drivers	9
3.2.2	Linux I ² C/GPIO drivers	9
3.3	Operating System API Setup.....	9
3.3.1	Microsoft Windows I ² C/GPIO API collateral.....	9
3.3.2	Linux I ² C/GPIO API collateral.....	9
3.4	Code Samples	10
3.4.1	Linux GPIO.....	10
3.4.2	Linux I ² C	13
3.4.3	Linux PWM (DE3815TY only)	13

Preface

The *Intel NUC: Programming for the Custom Solutions Header* whitepaper provides information for the experienced programmer to access additional features on certain Intel NUC models. It is not intended to cover programming basics but will provide links to Intel and non-Intel sources for additional information, including code samples. Intel provides these as examples only and makes no claim as to the viability of these code samples. By doing so, the user assumes all risk, inherent or otherwise.

Common Notation

#	Used after a signal name to identify an active-low signal (such as USBP0#)
API	Application Programming Interface
BIOS	Basic Input / Output System
CSH	Custom Solutions header
GPIO	General Purpose Input/Output
I ² C	Inter-Integrated Circuit
PWM	Pulse-Width Modulation
SCI/SMI	System Control Interrupt/System Management Interrupt
*	Symbol used to indicate other names and brands may be claimed as the property of others.

1.0 Introduction

Intel® NUC kits and boards are fully-featured mini-computers that conserve space and can be placed anywhere at home or in office while displaying no loss in computing performance. Several Intel NUC models go further by providing programmable features to expand their capabilities. This whitepaper will serve as a guide to understanding the available interfaces for accessing these programmable features.

Signal	Purpose	TY	MY
1.8V, 3.3V, and 5V standby	Can be used to power custom solution (such as daughter card, etc.) with up to 2 A of current rating capability per each of these voltages. Pins can also be used to monitor the presence of 1.8V, 3.3V and 5V standby power. Standby power is always on, even when board power is off.	✓	3.3V and 5V only
DMIC_CLK and DMIC_Data	Clock output and data I/O for a digital microphone interface (DMIC)	✓	✓
HDMI Consumer Electronics Control (HDMI CEC)	Provides the standard communication signal from the HDMI connector (HDMI* home page). There is no HDMI CEC controller onboard; rather, the HDMI CEC signal is exposed through this header for third party solutions to monitor/control CEC activity between multiple HDMI devices. HDMI-CEC adapters are available from vendors such as Pulse-Eight* .	✓	
I2C0_CLK and I2C0_DATA I2C1_CLK and I2C1_DATA	Inter-Integrated Circuit (I ² C) bus interface signals that allow connection of low-speed peripherals. An I ² C bus specification and user manual may be found here .	✓	✓
PWM[0] PWM[1]	Pulse-width modulation (PWM) signals that can be used to control analog loads, such as motors or fans. The power rating capability per each PWM signal is 5V at 250mA.	✓	
SCI/SMI Interrupt	Provides the input for direct connection to a signal (such as a front panel push-button) capable of triggering an OS-level command in Windows* (formerly referred to by Microsoft as Direct Application Launch). The voltage level I/O for this pin is 3.3V. Refer to the Intel direct application launch utility to map triggered events to Windows files.	✓	✓
SMB_CLK, SMB_DATA, and SMB_ALERT#	System Management Bus (SMBus) interface signals. General SMBus information can be found on the platform EDS and at SMBus Specifications .	✓	✓

Note: To enable these features, please use BIOS 0049 for DE3815TY, BIOS 0036 for NUC5i3MY and BIOS 0029 for NUC5i5MY (or later).

The examples in this document have been tested using Ubuntu* 15.04.

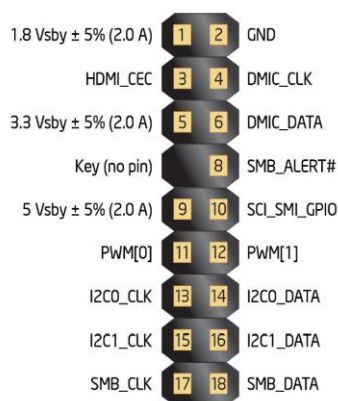
2.0 Hardware

Programmable headers are available on several Intel NUC models, with some variations. These variations are the result of customer feedback and the changing programming environment. Our intent is to improve the capabilities of the NUC.

2.1 Intel NUC DE3815TY

The Intel NUC DE3815TY was designed with embedded uses in mind, enabling new features previously unavailable via the Custom Solutions Header (CSH). It does this by allowing the user to assign GPIO, I²C and PWM signals to specific I/O pins on the headers in system BIOS. See **Table 1** below for a listing of the CSH programmable pin assignments.

Note: Please refer to the [Intel NUC Board DE3815TYBE Technical Product Specification](#) for a more complete listing of headers and their signals. Refer to section 3.4.1 to determine GPIO base address number and how Linux GPIO values are calculated.



Custom Solutions Header

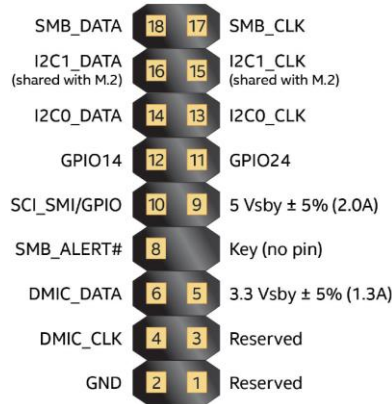
Pin	Header signal name	GPIO signal	Linux GPIO#	Alternate function signal	On-board circuit (GPIOs default to input)
10	SCI/SMI Interrupt	N/A	N/A	RESERVED	Output; 3.3V; 2.2k PU to 3.3VSB [†]
11	PWM[0]	GPIO_[base+94]	504	SIO_PWM[0]	Output; 5V driver; 680ohm PU to 5V and 680ohm PD to GND
12	PWM[1]	GPIO_[base+95]	505	SIO_PWM[1]	Output; 5V driver; 680ohm PU to 5V and 680ohm PD to GND
13	I2C0_CLK	GPIO_[base+79]	489	SIO_I2C0_CLK	Bi-dir; 3.3V I/O; 2.2k PU to 3.3VSB
14	I2C0_DATA	GPIO_[base+78]	488	SIO_I2C0_DATA	Bi-dir; 3.3V I/O; 2.2k PU to 3.3VSB
15	I2C1_CLK	GPIO_[base+81]	491	SIO_I2C1_CLK	Bi-dir; 3.3V I/O; 2.2k PU to 3.3VSB
16	I2C1_DATA	GPIO_[base+80]	490	SIO_I2C1_DATA	Bi-dir; 3.3V I/O; 2.2k PU to 3.3VSB

Table 1. DE3815TY Custom Solutions header programmable pins

[†]Note: FET gate not powered during standby

2.2 Intel NUCs NUC5i3MY and NUC5i5MY

The Intel NUC NUC5i3MY and NUC5i5MY are richly-featured commercial NUCs available in both board and kit form, and contain a Custom Solutions header as well. The programmable pins are listed in **Table 2** below.



Custom Solutions Header

Pin	Header signal name	GPIO Signal	Linux GPIO #	Alternate function signal	On-board circuit (GPIOs default to input)
10	PCH_GPIO44	GPIO_[base+44]	462	N/A	Bi-dir; unbuffered; 10k PU to 3.3VSB
11	PCH_GPIO24	GPIO_[base+24]	442	N/A	Bi-dir; unbuffered; 10k PU to 3.3VSB
12	PCH_GPIO14	GPIO_[base+14]	432	SCI/SMI Interrupt	Bi-dir; unbuffered; 10k PU to 3.3VSB
13	I2C0_CLK	GPIO_[base+5]	423	I2C0_SCL	Bi-dir; 3.3V I/O; 8.2k PU to 3.3V
14	I2C0_DATA	GPIO_[base+4]	422	I2C0_SDA	Bi-dir; 3.3V I/O; 8.2k PU to 3.3V
15	I2C1_CLK	GPIO_[base+7]	425	I2C1_SCL	Bi-dir; 3.3V I/O; 8.2k PU to 3.3V
16	I2C1_DATA	GPIO_[base+6]	424	I2C1_SDA	Bi-dir; 3.3V I/O; 8.2k PU to 3.3V

Table 2. NUC5i3MY and NUC5i5MY Custom Solutions header programmable pins

3.0 Software

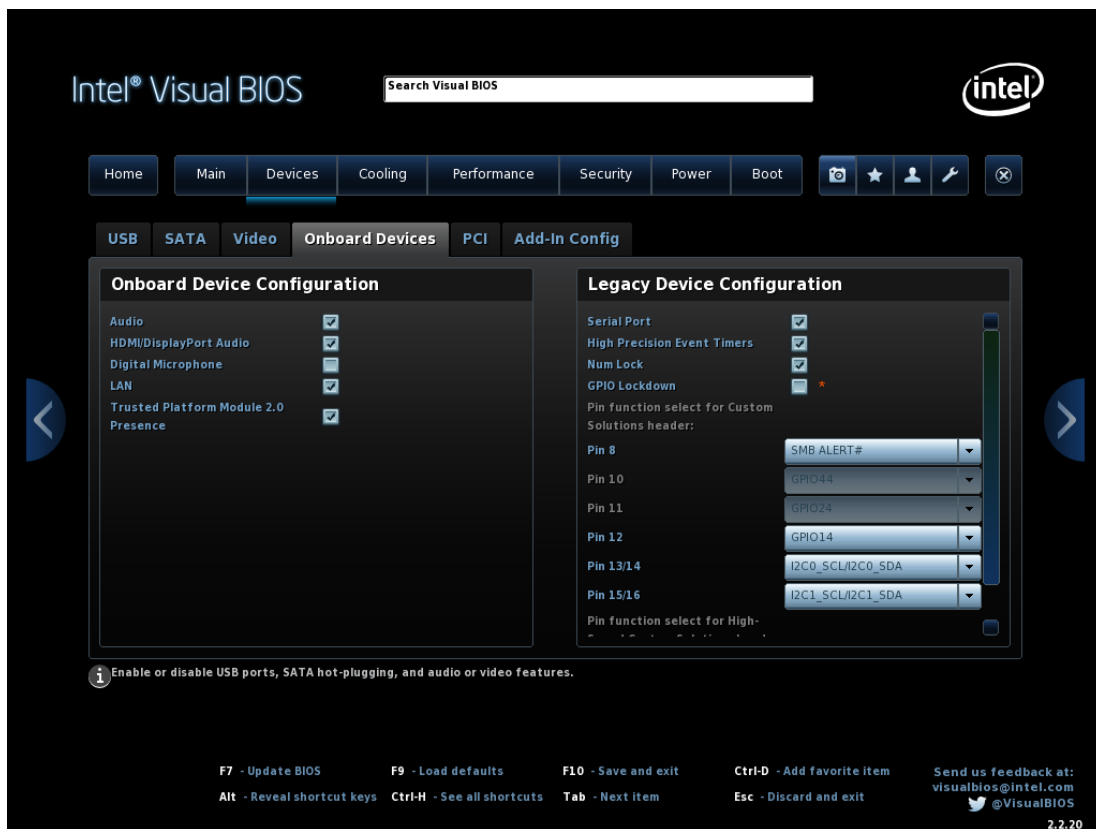
The programmable features of Intel NUCs are accessed through a combination of system BIOS, operating system APIs and OS-level drivers. These may vary depending on the Intel NUC model, the OS used and the particular subsystem being accessed.

3.1 BIOS Setup

In order for the Intel NUC to support the GPIO, I²C and PWM interfaces, the BIOS includes specific settings in the “Legacy Device Configuration” pane found in the “Advanced” / “Devices” / “Onboard Devices” page. Please ensure the Intel NUC is running the latest BIOS for these interfaces to be accessible by the operating system.

In order to use GPIO signals:

- **(Intel NUCs NUC5i3MY and NUC5i5MY only)** “GPIO Lockdown” checkbox must be cleared (i.e. disabled)
- Menu items under “Pin function select for Custom Solutions header” must be set to the desired pin operation (GPIO or alternate function).



3.2 Operating System Driver Setup

Drivers for programming the GPIO and I²C interfaces under Windows and Linux operating systems can be found via the [Intel Download Center](#).

3.2.1 Microsoft Windows I²C/GPIO drivers

Note: Make sure to enable the GPIO and I²C host controllers in the BIOS before the OS drivers are installed; otherwise, driver installation may fail.

- A. Windows 7 drivers:
 - a. **DE3815TY:** https://downloadcenter.intel.com/Detail_Desc.aspx?DwnldID=23888
 - b. **NUC5i3MY / NUC5i5MY:** I²C/GPIO interfaces are not supported for Windows 7
- B. Windows 8.1 drivers:
 - a. **DE3815TY:** https://downloadcenter.intel.com/Detail_Desc.aspx?DwnldID=24096
 - b. **NUC5i3MY / NUC5i5MY:**
https://downloadcenter.intel.com/Detail_Desc.aspx?DwnldID=24694&lang=eng&ProdId=3858

3.2.2 Linux I²C/GPIO drivers

- A. **DE3815TY:** Drivers are provided in standard distributions
- B. **NUC5i3MY / NUC5i5MY:** kernel **3.18rc1** includes the *gpio-lynxpoint.c* I²C/GPIO driver source code.

3.3 Operating System API Setup

I²C, GPIO API documentation may be downloaded from Microsoft and Intel websites.

3.3.1 Microsoft Windows I²C/GPIO API collateral

- A. Windows 7:
 - a. **DE3815TY:** The “Software Developers Manual for Windows 7 IO Driver” is contained inside the package “Intel Embedded Drivers for Windows* 7 (32 and 64-bit)” found at https://downloadcenter.intel.com/Detail_Desc.aspx?DwnldID=24548
 - b. **NUC5i3MY / NUC5i5MY:** I²C/GPIO interfaces are not supported for Windows 7
- B. Windows 8.1:
DE3815TY, NUC5i3MY and NUC5i5MY:
 - a. I²C: [http://msdn.microsoft.com/en-us/library/windows/hardware/hh450906\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/hardware/hh450906(v=vs.85).aspx)
 - b. GPIO (IOCTL): [http://msdn.microsoft.com/en-us/library/windows/hardware/hh439515\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/hardware/hh439515(v=vs.85).aspx)

3.3.2 Linux I²C/GPIO API collateral

- A. I²C: <https://www.kernel.org/doc/Documentation/i2c/>
- B. GPIO: <https://www.kernel.org/doc/Documentation/gpio/>
(Very useful reference: <https://www.kernel.org/doc/Documentation/gpio/sysfs.txt>)

3.4 Code Samples

The following examples illustrate functionality of the GPIO and I²C interfaces (and PWM for DE3815TY). Linux examples were tested using Ubuntu 15.04.

3.4.1 Linux GPIO

GPIO signals under Linux are identified by adding the signal's offset to the base address of its controller. The GPIO controller base addresses documented below were observed when testing under Ubuntu 15.04.

Note for DE3815TY:

There are three GPIO controllers on DE3815TY, which can be listed with the following command.

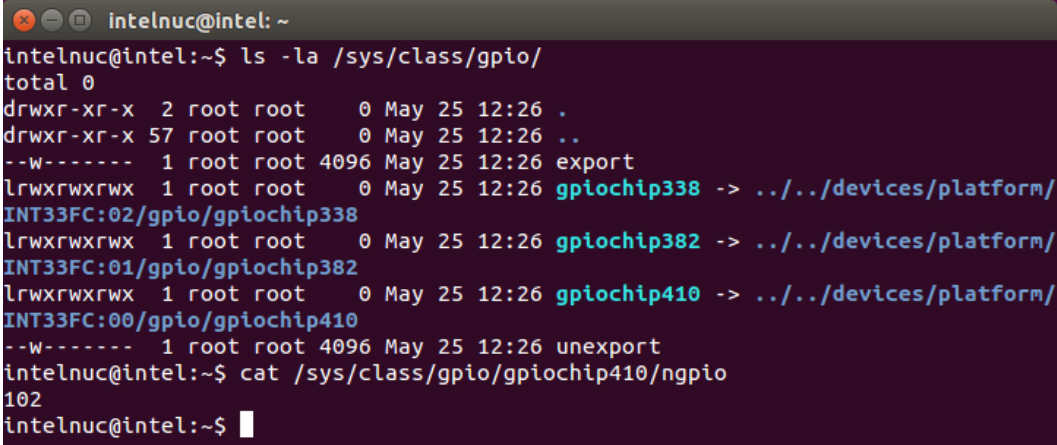
```
➤ ls -la /sys/class/gpio
```

Only one of these controllers supports the GPIO signals exposed on the Custom Solutions header. In order to determine the base address of such controller, run the following command for each of the “gpiochipXYZ” devices listed under /sys/class/gpio/ (where XYZ is a number).

```
➤ cat /sys/class/gpio/gpiochipXYZ/ngpio
```

The device where the above command reveals an *ngpio* value of 102 is the controller we are interested in. The “XYZ” value of this filename is the base address of the controller. In order to determine the absolute GPIO signal addresses, add this “XYZ” value (controller base address) to the signal offset address listed for each pin on **Table 1**.

For example, here is a screenshot after running the above commands on Ubuntu 15.04



```
intelnuc@intel:~  
intelnuc@intel:~$ ls -la /sys/class/gpio/  
total 0  
drwxr-xr-x  2 root root    0 May 25 12:26 .  
drwxr-xr-x 57 root root    0 May 25 12:26 ..  
-w-----  1 root root 4096 May 25 12:26 export  
lrwxrwxrwx  1 root root    0 May 25 12:26 gpiochip338 -> ../../devices/platform/  
INT33FC:02/gpio/gpiochip338  
lrwxrwxrwx  1 root root    0 May 25 12:26 gpiochip382 -> ../../devices/platform/  
INT33FC:01/gpio/gpiochip382  
lrwxrwxrwx  1 root root    0 May 25 12:26 gpiochip410 -> ../../devices/platform/  
INT33FC:00/gpio/gpiochip410  
-w-----  1 root root 4096 May 25 12:26 unexport  
intelnuc@intel:~$ cat /sys/class/gpio/gpiochip410/ngpio  
102  
intelnuc@intel:~$
```

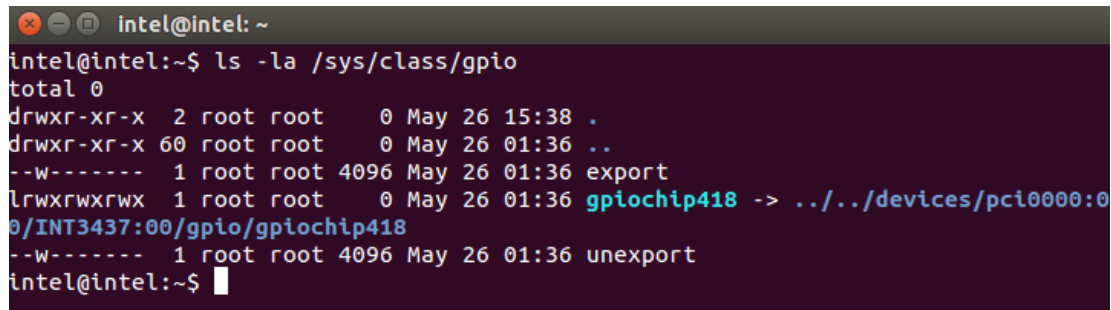
Therefore, the GPIO controller base address under Ubuntu 15.04 is 410. Using DE3815TY's pin11 as an example, its absolute address (adding the controller base address (410) to pin11's offset address (94)) would be 504.

Note for NUC5i3MY / NUC5i5MY:

In order to determine the base addresses of the GPIO controller, run the following command:

➤ `ls -la /sys/class/gpio`

This should reveal a file named `gpiochipXYZ`, where “XYZ” is the controller base address which should be added to the GPIO signal offset address shown in **Table 2** to calculate GPIO absolute addresses under Linux.



```
intel@intel:~$ ls -la /sys/class/gpio
total 0
drwxr-xr-x  2 root root    0 May 26 15:38 .
drwxr-xr-x 60 root root    0 May 26 01:36 ..
--w-----  1 root root 4096 May 26 01:36 export
lrwxrwxrwx  1 root root    0 May 26 01:36 gpiochip418 -> ../../devices/pci0000:00/INT3437:00/gpio/gpiochip418
--w-----  1 root root 4096 May 26 01:36 unexport
intel@intel:~$
```

The above screenshot shows a controller base address of 418 under Ubuntu 15.04.

Notes:

If you see a file permissions error please use the following commands to set the correct read/write permissions:

- `sudo chmod 202 export unexport`
- `sudo chmod 646 active_low direction uevent value`

To confirm if your pin is active and to retrieve a list of available GPIO pins on the device and their status, do the following and save the file to the desktop or another easily accessible location:

- `cd Desktop`
- `sudo cat /sys/kernel/debug/gpio >> gpiopins.txt`

Look through the file and find the controller with base address that we are interested in (410). Pins that are listed as Sysfs and in/out or just out, are user editable pins.

```

gpiopins.txt (~/Desktop) - gedit
Open Save Undo
gpiopins.txt x
GPIOs 410-511, platform/INT33FC:00, INT33FC:00:
gpio-0 (Unrequested) ) in lo pad-85 offset:0x550 mux:1 up 20k
gpio-1 (Unrequested) ) in lo pad-89 offset:0x590 mux:1 up 20k
gpio-2 (Unrequested) ) in hi pad-93 offset:0x5d0 mux:1 up 20k
gpio-3 (sysfs) ) in lo pad-96 offset:0x600 mux:0 fall up 20k
gpio-4 (Unrequested) ) in hi pad-99 offset:0x630 mux:1 up 20k
gpio-5 (sysfs) ) in lo pad-102 offset:0x660 mux:0 up 20k
gpio-6 (Unrequested) ) in hi pad-98 offset:0x620 mux:1 up 20k
gpio-7 (sysfs) ) in out lo pad-101 offset:0x650 mux:0 fall rise up 20k
gpio-8 (Unrequested) ) in hi pad-34 offset:0x220 mux:2 down 20k
gpio-9 (Unrequested) ) in lo pad-37 offset:0x250 mux:2 down 20k
gpio-10 (Unrequested) ) in lo pad-36 offset:0x240 mux:2 down 20k
gpio-11 (Unrequested) ) in lo pad-38 offset:0x260 mux:2 down 20k
gpio-12 (Unrequested) ) in lo pad-39 offset:0x270 mux:2 down 20k
gpio-13 (Unrequested) ) in lo pad-35 offset:0x230 mux:2 down 20k
gpio-14 (sysfs) ) in lo pad-40 offset:0x280 mux:0 down 20k
gpio-15 (sysfs) ) in lo pad-84 offset:0x540 mux:0 down 20k
Plain Text Tab Width: 8 Ln 7, Col 64 INS

```

With the information given above you can create the GPIO port by echoing into the editable pins.

Examples:

Using pin13 of the Custom Solutions header on DE3815TY:

Note: Ensure pin 13 is set as a GPIO signal in the BIOS.

Create GPIO port:

```
➤ echo 489 > /sys/class/gpio/export
```

To use GPIO as input (read), set the *direction* variable to “in”, and then read the *value* variable:

```
➤ echo in > /sys/class/gpio/gpio489/direction
➤ cat /sys/class/gpio/gpio489/value
```

To use GPIO as output (write), set the *direction* variable to “out”, then set the *value* variable to “0” or “1”:

```
➤ echo out > /sys/class/gpio/gpio489/direction
➤ echo 1 > /sys/class/gpio/gpio489/value
```

Note: Output voltage level is driven by the *value* variable correlated with the *active_low* variable (i.e. when *active_low* is set to “0”, output voltage is 0V if value is set to “0”).

To close the port:

```
➤ echo 489 > /sys/class/gpio/unexport
```

3.4.2 Linux I²C

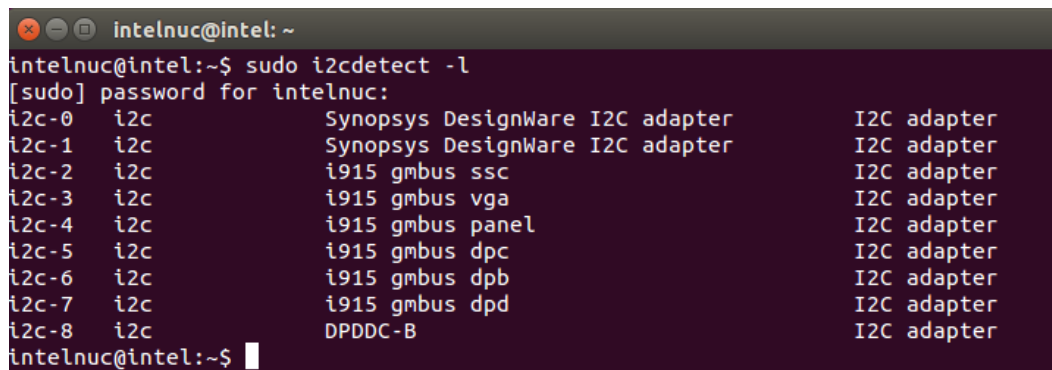
Note: Ensure that header pins are properly configured to the desired I²C bus. For this example, pins 13 & 14 must be set to I2C0_CLK/DATA in the BIOS.

I²C under Linux was tested using the opensource *Designware* I2C driver and i2c-tools** package available on many distributions.

I²C bus 0 example using pins 13 & 14 of the Custom Solutions header on DE3815TY:

To show which I²C buses are active on the system:

➤ `sudo i2cdetect -l`



```
intelnuc@intel: ~
intelnuc@intel:~$ sudo i2cdetect -l
[sudo] password for intelnuc:
i2c-0  i2c      Synopsys DesignWare I2C adapter      I2C adapter
i2c-1  i2c      Synopsys DesignWare I2C adapter      I2C adapter
i2c-2  i2c      i915 gmbus ssc                       I2C adapter
i2c-3  i2c      i915 gmbus vga                       I2C adapter
i2c-4  i2c      i915 gmbus panel                     I2C adapter
i2c-5  i2c      i915 gmbus dpc                       I2C adapter
i2c-6  i2c      i915 gmbus dpb                       I2C adapter
i2c-7  i2c      i915 gmbus dpd                       I2C adapter
i2c-8  i2c      DPDDC-B                              I2C adapter
intelnuc@intel:~$
```

Note: If the above command produces no results try running the following command, then repeat the previous command:

➤ `sudo modprobe i2c-dev`

To show which addresses on the selected I²C bus (I²C bus 0 in this example) have an I²C device:

➤ `sudo i2cdetect -r 0`

Note: The above command refers to I²C bus 0 (I2C0_CLK/DATA pins 13/14). To refer to I²C bus 1 (I2C1_CLK/DATA pins 15/16) replace “0” with “1” in the above command.

3.4.3 Linux PWM (DE3815TY only)

Note: Ensure that header pins are properly configured to the desired PWM signal. For this example, pin 11 must be set to PWM[0] in the BIOS.

PWM configuration parameters:

- `period`: sets the PWM signal period in nanoseconds (valid range is 80ns to 100000ns, with 40ns increments)
- `duty_cycle`: sets the duty cycle in nanoseconds (valid range is 80ns to 100000ns, with 40ns increments)
- `enable`: enables or disables the PWM signal (“1” is enabled, “0” is disabled)

An example of PWM[0] using pin11 of the Custom Solutions header of DE3815TY, with a period of *100000ns* and a duty cycle of *75000ns*:

```
➤ cd /sys/class/pwm/pwmchip1
➤ echo 0 > export
➤ cd pwm0
➤ echo 100000 > period
➤ echo 75000 > duty_cycle
➤ echo 1 > enable
```

4.0 Additional References

From WinHEC:

<http://video.ch9.ms/sessions/winhec/2015/files/DDF300%20-%20Accessing%20GPIO,%20I2C,%20and%20UART%20Devices.pptx>

MSDN reference on SPB:

[https://msdn.microsoft.com/en-us/library/windows/hardware/dn915108\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/hardware/dn915108(v=vs.85).aspx)

Microsoft driver samples on SPB interface:

<https://github.com/Microsoft/Windows-driver-samples/tree/master/spb/SpbTestTool>