Intel® Edison GPIO Pin Multiplexing Guide

Print

by Dan O'Donovan and David Hunt

Introduction

The Intel® Edison platform contains external input/output pin connections which may be configured to be used in a variety of interfacing modes, such as GPIO, PWM, SPI, I2C, ADC, for compatibility with Arduino Uno shield hardware. This article describes the pin functions available, detailed GPIO pin mapping for pin control and I/O, and use of Linux command line tools to configure the external I/O pin functions correctly for the desired mode of operation.

1. To use the information in this guide, all you need is access to the Linux command line on an Intel® Edison Arduino baseboard. If you want to dig a bit deeper, then the schematics should be available at maker.intel.com in the Edison Community section.

GPIO allocation and shield pin control

The 20 Arduino-compatible shield I/O pins on the Edison board are numbered IO0-IO19 (see Figure 1 below). All pins support basic GPIO functionality. Some of the pins also support PWM, ADC, SPI or I2C functions. Selection of different pin functions on Edison is achieved through use of SoC pin control interfaces and GPIO output signals dedicated for multiplexing control. The following sections detail the mapping of each of the GPIO pins available on Edison platform to their respective functions, which can be broadly categorised as follows:

- External GPIO
  - Used for digital input/output signalling via the external shield pins
- Pin multiplexing control
  - Used for selecting different functions available on a given shield pin
- Pin buffer (level-shifter) direction control
  - Used to configure the buffer on a given shield pin for input or output
- Pin pull-up resistor control
  - Used to enable/disable a pull-up resistor on a given shield pin
- Miscellaneous

To use any of the supported functions on a shield pin, it is first necessary to configure the multiplexing, buffer direction, and pull-up resistor controls applicable to that pin.

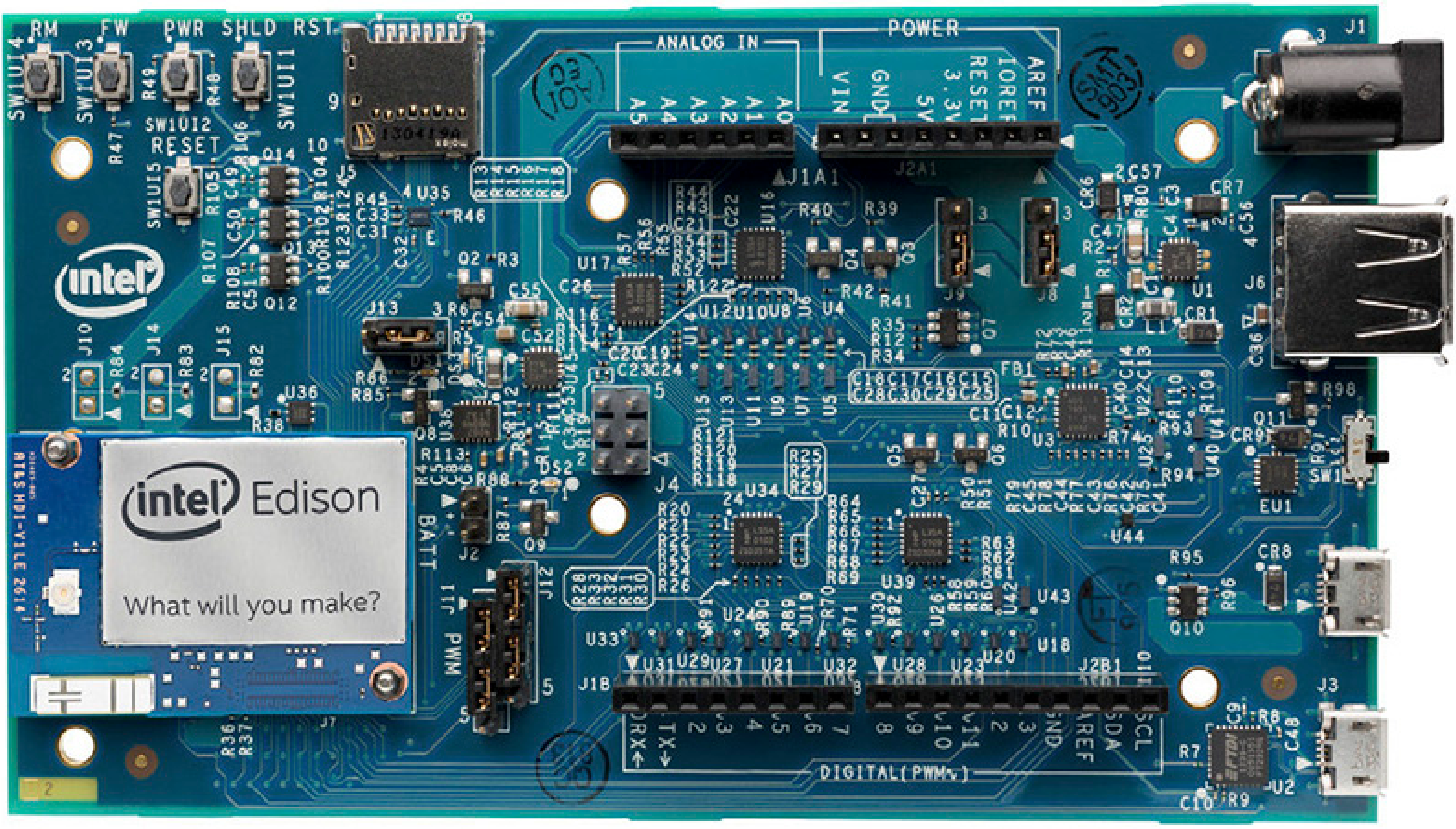

Figure 1. Intel® Edison Arduino board

Shield pin GPIO mapping

The following table describes the mapping of GPIO and PWM pin numbers (in Linux) to shield I/O pins. The following details are included:

- Shield Pin – Digital I/O pin number as per Arduino Uno pin numbering scheme
- Linux – The pin number assigned under Linux
- Muxed Functions – Other signals available on this shield pin

Shield Pin	GPIO Linux Pin	PWM Linux Pin	Muxed functions	Notes
IO0	130		UART1_RXD	
IO1	131		UART1_TXD	
IO2	128		UART1_CTS*	
IO3	12	0	PWM0	Depends on PWM Swizzler**
IO4	129		UART1_RTS*	
IO5	13	1	PWM1	Depends on PWM Swizzler**
IO6	182	2	PWM2	Depends on PWM Swizzler**
IO7	48		-	
IO8	49		-	
IO9	183	3	PWM3	Depends on PWM Swizzler**
IO10	41	Swiz	SPI_2_SS1 I2S_2_FS* PWM4_OUT	 Depends on PWM Swizzler**
IO11	43	Swiz	SPI_2_TXD I2S_2_TXD*	

		PWM5_OUT	Depends on PWM Swizzler**
IO12	42	SPI_2_RXD I2S_2_RXD*	
IO13	40	SPI_2_CLK I2S_2_CLK*	
IO14	44	AIN0	
IO15	45	AIN1	
IO16	46	AIN2	
IO17	47	AIN3	
IO18	14	AIN4 I2C_6_SDA	
IO19	165	AIN5	

- Table 1: Shield pin GPIO mapping

* Some additional functions are available on certain SoC pins, such as I2S and UART flow control, but are not currently planned to be supported by the Arduino library. However, it may be possible to utilise these from Linux.
** The SoC offers only 4 PWM pins. A jumper pin matrix labelled "PWM swizzler" on the base board allows these 4 pins to be connected to any subset of the 6 shield header pins normally used for PWM. From the factory, IO3, IO5, IO6 and IO9 will be connected to the 4 available SoC PWM pins as described above. This can be manually altered to connect IO10 or IO11.

Summary Pin Function Multiplexing Control

All GPIO pins on the Arduino header require some intenal GPIOs to be set up before the pin is usable. This is usually as simple as setting an output enable, pull-up enable and mode. However, some pins have extra functionality such as SPI, PWM, or I2C, so these pins need extra multiplexing (muxing) in order to be usable.

The table below attempts to simplify this so that a programmer can easily see all the muxing pins affected for a given Arduino header pin. The colour codes in the table show related boxes. For example the blue boxes are meant to show the relationship between the Pin Mux pins and the Pin Modes.

This table is a synopsis of the more detailed tables below, which contain extra information such as schematic pin numbers. For most needs, this synopsized table should suffice.

	Linux GPIO Pin	GPIO Pin Mux			SoC Pin Modes		Output Enable * (high = output)	Pull-up Enable**
		Linux Pin	0 (low)	1 (high)	0	1	Linux	Linux
IO0	130				GPIO	UART	248	216
IO1	131				GPIO	UART	249	217
IO2	128				GPIO	UART	250	218
IO3	12				GPIO	PWM	251	219
IO4	129				GPIO	UART	252	220
IO5	13				GPIO	PWM	253	221
IO6	182				GPIO	PWM	254	222
IO7	48				GPIO		255	223
IO8	49				GPIO		256	224
IO9	183				GPIO	PWM	257	225
IO10	41	263	PWM	see 240	GPIO	I2S or SPI	258	226
		240	GPIO or I2S	GPIO or SPI_FS				
IO11	43	262	PWM	see 241	GPIO	I2S or SPI	259	227
		241	GPIO or I2S	GPIO or SPI TXD				
IO12	42	242	GPIO or I2S	GPIO or SPI RXD	GPIO	I2S or SPI	260	228
			GPIO or I2S	GPIO or SPI		I2S or		

IO13		40	243		CLK	GPIO	SPI	261	229
IO14	(A0)	44	200	GPIO	A0	GPIO		232	208
IO15	(A1)	45	201	GPIO	A1	GPIO		233	209
IO16	(A2)	46	202	GPIO	A2	GPIO		234	210
IO17	(A3)	47	203	GPIO	A3	GPIO		235	211
IO18	(A4)	14	204	GPIO or I2C SDA	A4	GPIO	I2C-6	236	212
IO19	(A5)	165	205	GPIO or I2C SCL	A5	GPIO	I2C-6	237	213

Note: Before setting up any muxing, it is recommended to set pin 214 (TRI_STATE_ALL) LOW, make all your changes, then set pin 214 HIGH again.

Example 1: Setting up IO0 for output. Pin 0 has no muxing requirements, so just needs output enable and pull-up enable set.
Exampel 2: Setting IO10 to SPI, 263 needs to be set HIGH, which enables non-PWM functionality, contolled by 240. 240 needs to be set HIGH, and then setting the SoC Pin Mode to mode1 selects the SPI function.

There are several more detailed examples at the end of this article.

GPIO interrupt support

All GPIO inputs on the Edison platform are interrupt-capable, and all interrupt types are supported on all inputs. The following table lists the specific edge- and level-triggered interrupt types which are supported on each pin.

ShieldPin	GPIO			Edge-Triggered			Level-Triggered*	
	Source	Pin	Linux	Rising	Falling	Both	Low	High
IO0	SoC	GP130_UART1_RXD	130	Y	Y	Y	Y	Y
IO1	SoC	GP131_UART1_TXD	131	Y	Y	Y	Y	Y
IO2IO3	SoC	GP128_UART1_CTS	128	Y	Y	Y	Y	Y
	SoC	GP12_PWM0	12	Y	Y	Y	Y	Y
IO4IO5	SoC	GP129	129	Y	Y	Y	Y	Y
	SoC	GP13_PWM1	13	Y	Y	Y	Y	Y
IO6	SoC	GP182_PWM2	182	Y	Y	Y	Y	Y
IO7	SoC	GP48	48	Y	Y	Y	Y	Y
IO8	SoC	GP49	49	Y	Y	Y	Y	Y
IO9	SoC	GP183_PWM3	183	Y	Y	Y	Y	Y
IO10	SoC	GP41	41	Y	Y	Y	Y	Y
IO11	SoC	GP43	43	Y	Y	Y	Y	Y
IO12	SoC	GP42	42	Y	Y	Y	Y	Y
IO13	SoC	GP40	40	Y	Y	Y	Y	Y
IO14	SoC	GP44	44	Y	Y	Y	Y	Y
IO15	SoC	GP45	45	Y	Y	Y	Y	Y
IO16	SoC	GP46	46	Y	Y	Y	Y	Y
IO17	SoC	GP47	47	Y	Y	Y	Y	Y
IO18	SoC	GP14	14	Y	Y	Y	Y	Y
IO19	SoC	GP165	165	Y	Y	Y	Y	Y

Table 2: GPIO Interrupt Support
* Level-triggered interrupts are not supported by the Arduino library, a limitation of the GPIO sysfs interface.

Detailed Pin Function Multiplexing Control

The following table lists the GPIO outputs dedicated to pin multiplexing control. Different functions may be selected for specific shield I/O pins by setting these GPIO outputs to 0/1 (low/high). Additionally, some of the SoC GPIO pins also feature internal mux options. These are listed as “SoC Pin Modes”. Currently, these are configured by setting the required pin mode for the corresponding SoC GPIO pin N, via /sys/kernel/debug/gpio_debug/gpioN/current_pinmux, to “mode[0/1/2/...]”

ShieldPin	GPIO Pin Mux					SoC Pin Modes				
	Pin	Linux	0 (low)	1 (high)	Power-on default	Pin	Linux	0	1	2
IO0	-					GP130	130	GPIO	UART	
IO1	-					GP131	131	GPIO	UART	
IO2	-					GP128	128	GPIO	UART	
IO3	-					GP12	12	GPIO	PWM	
IO4	-					GP129	129	GPIO	UART	
IO5	-					GP13	13	GPIO	PWM	
IO6	-					GP182	182	GPIO	PWM	
IO7	-					GP48	48	GPIO		
IO8	-					GP49	49	GPIO		
IO9	-					GP183	183	GPIO	PWM	
IO10	U34_ IO1.7	263	PWM4_OUT	GP41 SSP5_FS_1	Pulled-down input	GP41 GP111	41 111	GPIO GPIO	I2S SPI	
	U16_ IO1.0	240	GP41	SSP5_FS_1	Pulled-up input*					
IO11	U34_ IO1.6	262	PWM5_OUT	GP43 SSP5_TXD	Pulled-down input	GP43 GP115	43 115	GPIO GPIO	I2S SPI	
	U16_ IO1.1	241	GP43	SSP5_TXD	Pulled-up input*					
IO12	U16_ IO1.2	242	GP42	SSP5_RXD	Pulled-up input*	GP42 GP114	42 114	GPIO GPIO	I2S SPI	
IO13	U16_ IO1.3	243	GP40	SSP5_CLK	Pulled-up input*	GP40 GP109	40 109	GPIO GPIO	I2S SPI	
IO14	U17_ IO0.0	200	GP44	A0	Pulled-up input*	GP44	44	GPIO		
IO15	U17_ IO0.1	201	GP45	A1	Pulled-up input*	GP45	45	GPIO		
IO16	U17_ IO0.2	202	GP46	A2	Pulled-up input*	GP46	46	GPIO		
IO17	U17_ IO0.3	203	GP47	A3	Pulled-up input*	GP47	47	GPIO		
IO18	U17_ IO0.4	204	GP14 I2C6_SDA	A4	Pulled-up input*	GP14 GP27	14 27	GPIO GPIO	I2C-6	I2C-8
IO19	U17_ IO0.5	205	GP165 I2C6_SCL	A5	Pulled-up input*	GP165 GP28	165 28	GPIO GPIO	I2C-6	I2C-8

Table 3: Pin Function Multiplexing Control
* These pins are pulled-up inputs at power-on. This effectively enables the Mux switches (i.e. Mux function 1 is selected).

Pin direction and pull-up control

For most shield pins on the Edison board, there is a buffer/level-shifter which needs to be configured for input or output direction, and an external 47k pull-up/down resistor which may be optionally enabled. Both are driven by dedicated GPIO outputs, listed below. When configuring a shield pin as an output, it is advisable to configure the buffer for output BEFORE setting the SoC GPIO pin direction to output. To disconnect the external pull-up/down resistors, it is necessary to configure as high-impedance inputs the GPIOs which drive them. Note also that the GPIO signals from the PCAL9555A GPIO expanders have internal 100k pull-up resistors which are connected to the GPIO pins by default. These need to be disabled in many cases, by configuring those pins as high-impedance inputs.

Shield pin	Output Enable GPIO (high = output)			Pull-up Enable GPIO		
	Pin	Linux	Power-on default	Pin	Linux	Power-on default
IO0	U34_ IO0.0	248	Pulled-down*	U39_IO0.0	216	Pulled-up input**
IO1	U34_ IO0.1	249	Pulled-down*	U39_IO0.0	217	Pulled-up input**
IO2	U34_ IO0.2	250	Pulled-down*	U39_IO0.0	218	Pulled-up input**

IO3	U34_ IO0.3	251	Pulled-down*	U39_IO0.0	219	Pulled-up input**
IO4	U34_ IO0.4	252	Pulled-down*	U39_IO0.0	220	Pulled-up input**
IO5	U34_ IO0.5	253	Pulled-down*	U39_IO0.0	221	Pulled-up input**
IO6	U34_ IO0.6	254	Pulled-down*	U39_IO0.0	222	Pulled-up input**
IO7	U34_ IO0.7	255	Pulled-down*	U39_IO0.7	223	Pulled-up input**
IO8	U34_ IO1.0	256	Pulled-down*	U39_IO0.7	224	Pulled-up input**
IO9	U34_ IO1.1	257	Pulled-down*	U39_IO0.7	225	Pulled-up input**
IO10	U34_ IO1.2	258	Pulled-down*	U39_IO0.7	226	Pulled-up input**
IO11	U34_ IO1.3	259	Pulled-down*	U39_IO0.7	227	Pulled-up input**
IO12	U34_ IO1.4	260	Pulled-down*	U39_IO0.7	228	Pulled-up input**
IO13	U34_ IO1.5	261	Pulled-down*	U39_IO0.7	229	Pulled-up input**
IO14	U16_ IO0.0	232	Pulled-down*	U17_ IO1.0	208	Pulled-up input**
IO15	U16_ IO0.1	233	Pulled-down*	U17_ IO1.1	209	Pulled-up input**
IO16	U16_ IO0.2	234	Pulled-down*	U17_ IO1.2	210	Pulled-up input**
IO17	U16_ IO0.3	235	Pulled-down*	U17_ IO1.3	211	Pulled-up input**
IO18	U16_ IO0.4	236	Pulled-down*	U17_ IO1.4	212	Pulled-up input**
IO19	U16_ IO0.5	237	Pulled-down*	U17_ IO1.5	213	Pulled-up input**

Table 4: Pin direction and pull-up control
* These pins are externally pulled-down inputs at power-on. This effectively selects input direction for level shifters.
** These pins are internally pulled-up inputs at power-on. This effectively enables pull-ups (as 100k + 47k in series).

Miscellaneous GPIOs

The following GPIOs are used other platform functions and for Arduino shield compatibility.

Function	GPIO		Direction	Power-on default	Initial setup
	Pin	Linux			
TRI_STATE_ALL	U17_IO1.6	214	Output	Pulled-down	
SHLD_RESET	U17_IO1.7	215	Output	Pulled-up input*	
SHLD_RESET	U17_IO0.7	207	Input	Pulled-up input*	

Table 5: Miscellaneous GPIOs * These pins are pulled-up inputs at power-on. In this state, they have the same effect as outputs set high.

Shield pin configuration guide

1. Identify the Arduino shield pin number of the pin you wish to use, in the range IO0-IO19.
2. Identify the functions available for the given pin, and select the function you wish to use
    1. Typical functions are GPIO, PWM, UART, I2C, SPI, ADC.
    2. Only some functions are available on each pin.
3. Determine which GPIO signals, if any, need to be configured to select the correct pin muxing option for the selected function. Some pins only have a single function, or do not require mux control.
4. Determine which GPIO signals, if any, need to be configured to select the pin buffer direction for input or output, and determine the direction that is required.
5. Determine which GPIO signals, if any, need to be configured to select the pull-up resistor control, and whether the pull-up resistor should be enabled or disabled. Generally, for most pin functions, the pull-up resistors should typically be disabled. For GPIO input functions, the pull-up resistor may optionally be enabled or disabled according to the needs of the user.
6. Export the above GPIO numbers for access in the Linux user-space environment (i.e. from the command shell)
7. Configure the above GPIO numbers for output
8. Assert the TRI_STATE_ALL signal to disconnect the shield pins
9. Set the above GPIO numbers to assert their output logic levels as high or low.

10. Set the SoC GPIO pin mode for the required functionality
11. De-assert the TRI_STATE_ALL signal to reconnect the shield pins

Example 1: Configure IO5 as a GPIO input, with pull-up resistor disabled

1. The shield number is IO5. According to Table 1, the GPIO number is 13.
2. The function required is GPIO. According to Table 1, other functions available on this shield pin are: PWM
3. According to Table 3, GPIO 43 pin-mux must be set to mode0 to select GPIO According to Table 4, GPIO 253 must be set to 0 to disable the output direction for IO5.
4. According to Table 4, GPIO 221 must be set as a high-impedance input to disable the external pull-up resistor for IO5.
5. According to Table 5, the TRI_STATE_ALL signal is controlled by GPIO 214

So, the commands in Linux to achieve this are as follows:

```
# echo 13 > /sys/class/gpio/export
# echo 253 > /sys/class/gpio/export
# echo 221 > /sys/class/gpio/export
# echo 214 > /sys/class/gpio/export
# echo low > /sys/class/gpio/gpio214/direction
# echo low > /sys/class/gpio/gpio253/direction
# echo in > /sys/class/gpio/gpio221/direction
# echo mode0 > /sys/kernel/debug/gpio_debug/gpio13/current_pinmux
# echo in > /sys/class/gpio/gpio13/direction
# echo high > /sys/class/gpio/gpio214/direction
```

Now, it should be possible to use IO5 as a GPIO input. For example:

```
# cat /sys/class/gpio/gpio13/value
```

Example 2: Configure IO11 as a GPIO input, with pull-up resistor disabled

1. The shield number is IO11. According to Table 1, the GPIO number is 43.
2. The function required is GPIO. According to Table 1, other functions available on this shield pin are: PWM, SPI, I2S
3. According to Table 3, GPIO 262 must be set to 1 to select GPIO/SPI, GPIO 241 must be set to 0 to select GPIO, and GPIO 43 pin-mux must be set to ‘mode0’ to select GPIO
4. According to Table 4, GPIO 259 must be set to 0 to disable the output direction for IO11.
5. According to Table 4, GPIO 227 must be set as a high-impedance input to disable the external pull-up resistor for IO5.
6. According to Table 5, the TRI_STATE_ALL signal is controlled by GPIO 214

So, the commands in Linux to achieve this are as follows:

```
# echo 43 > /sys/class/gpio/export
# echo 262 > /sys/class/gpio/export
# echo 241 > /sys/class/gpio/export
# echo 259 > /sys/class/gpio/export
# echo 227 > /sys/class/gpio/export
# echo 214 > /sys/class/gpio/export
# echo low > /sys/class/gpio/gpio214/direction
# echo high > /sys/class/gpio/gpio262/direction
# echo low > /sys/class/gpio/gpio241/direction
# echo mode0 > /sys/kernel/debug/gpio_debug/gpio43/current_pinmux
# echo low > /sys/class/gpio/gpio259/direction
# echo in > /sys/class/gpio/gpio227/direction
# echo in > /sys/class/gpio/gpio43/direction
# echo high > /sys/class/gpio/gpio214/direction
```

Now, it should be possible to use IO11 as a GPIO input. For example:

```
# cat /sys/class/gpio/gpio43/value
```

Example 2: Configure IO7 as a GPIO input, with pull-up resistor enabled

1. The shield number is IO7. According to Table 1, the GPIO number is 48.
2. The function required is GPIO. According to Table 1, there are no other functions available on this pin
3. For IO7, there are no applicable mux options listed in Table 3.
4. According to Table 4, GPIO 255 must be set to 0 to disable the output direction for IO7.
5. According to Table 4, GPIO 223 must be set to output high to enable the external pull-up resistor for IO7.
6. According to Table 5, the TRI_STATE_ALL signal is controlled by GPIO 214

```
# echo 48 > /sys/class/gpio/export
# echo 255 > /sys/class/gpio/export
# echo 223 > /sys/class/gpio/export
# echo 214 > /sys/class/gpio/export
# echo low > /sys/class/gpio/gpio214/direction
# echo low > /sys/class/gpio/gpio255/direction
# echo high > /sys/class/gpio/gpio223/direction
# echo in > /sys/class/gpio/gpio48/direction
# echo high > /sys/class/gpio/gpio214/direction
```

Now, it should be possible to use IO7 as a GPIO input. For example:

```
# cat /sys/class/gpio/gpio48/value
```

Example 3: Configure IO6 as a PWM output

1. The shield number is IO6. According to Table 1, the GPIO number is 182.
2. The function required is PWM. According to Table 1, other functions available on this pin are: GPIO
3. According to Table 3, GPIO 182 pin-mux must be set to ‘mode1’ to select PWM
4. According to Table 4, GPIO 254 must be set to 1 to enable the output direction for IO6.
5. According to Table 4, GPIO 222 must be set as a high-impedance input to disable the pull-up resistor for IO6.
6. According to Table 5, the TRI_STATE_ALL signal is controlled by GPIO 214

```
# echo 254 > /sys/class/gpio/export
# echo 222 > /sys/class/gpio/export
# echo 214 > /sys/class/gpio/export
# echo low > /sys/class/gpio/gpio214/direction
# echo high > /sys/class/gpio/gpio254/direction
# echo in > /sys/class/gpio/gpio222/direction
# echo mode1 > /sys/kernel/debug/gpio_debug/gpio182/current_pinmux
# echo high > /sys/class/gpio/gpio214/direction
```

Now, it should be possible to use IO6 as a PWM ouput. For example:

```
# echo 2 > /sys/class/pwm/pwmchip0/export
# echo 2000000 > /sys/class/pwm/pwmchip0/pwm2/duty_cycle
# echo 1 > /sys/class/pwm/pwmchip0/pwm2/enable
```

Example 3: Configure IO14 as an ADC input

Note on ADC usage: The ADC sits on the SPI bus, and as such the SPI pins need to be set to a valid configuration (after cold boot) before the ADC can be used. Just Pins 10-13 as input (even just setting pin 13 will do), and the ADC will then function fine on 14-19.

1. The shield number is IO14. According to Table 1, the GPIO number is 44.
2. The function required is ADC. According to Table 1, other functions available on this pin are: GPIO
3. According to Table 3, GPIO 200 must be set to 1 to select ADC.
4. According to Table 4, GPIO 232 must be set to 0 to disable the output direction for IO14.
5. Any GPIO lines which are directly connected to IO14 should be configured as high-impedance inputs to prevent possible current leakage. According to Table 4, GPIO 208 is used to enable a pull-up resistor for IO14.
6. According to Table 5, the TRI_STATE_ALL signal is controlled by GPIO 214

So, the commands in Linux to achieve this are as follows:

```
# echo 200 > /sys/class/gpio/export
# echo 232 > /sys/class/gpio/export
# echo 208 > /sys/class/gpio/export
# echo 214 > /sys/class/gpio/export
# echo low > /sys/class/gpio/gpio214/direction
# echo high > /sys/class/gpio/gpio200/direction
# echo low > /sys/class/gpio/gpio232/direction
# echo in > /sys/class/gpio/gpio208/direction
# echo high > /sys/class/gpio/gpio214/direction
```

Now, it should be possible to use IO14 as an ADC input. For example:

```
# cat /sys/bus/iio/devices/iio:device1/in_voltage0_raw
```

Example 4: Configure IO18/IO19 for I2C connectivity

1. The shield number is IO18 and IO19. Corresponding GPIO numbers are 28 and 27, respectively.
2. The function required is I2C. According to Table 1, other functions available on these pins are: GPIO, ADC
3. According to Table 3, GPIO 204 must be set to 0 to select GPIO/I2C, and GPIO 28 pin-mux must be set to ‘mode1’ to select I2C for IO18
4. According to Table 3, GPIO 205 must be set to 0 to select GPIO/I2C, and GPIO 27 pin-mux must be set to ‘mode1’ to select I2C for IO19
5. GPIO 14 and GPIO 165 are also connected to the I2C signals, and should be configured as high-impedance inputs when I2C is in use on these pins, to prevent them driving a signal on the I2C bus.
6. According to Table 4, GPIO 236 must be set to 0 to disable the output direction for GPIO 14, and GPIO 237 must be set to 0 to disable the output direction for GPIO 165.
7. According to Table 4, GPIO 212 and 213 must be set as high-impedance inputs to disable the pull-up resistors for IO18 and IO19, respectively.
8. According to Table 5, the TRI_STATE_ALL signal is controlled by GPIO 214

So, the commands in Linux to achieve this are as follows:

```
# echo 28 > /sys/class/gpio/export
# echo 27 > /sys/class/gpio/export
# echo 204 > /sys/class/gpio/export
# echo 205 > /sys/class/gpio/export
# echo 236 > /sys/class/gpio/export
# echo 237 > /sys/class/gpio/export
# echo 14 > /sys/class/gpio/export
# echo 165 > /sys/class/gpio/export
# echo 212 > /sys/class/gpio/export
# echo 213 > /sys/class/gpio/export
# echo 214 > /sys/class/gpio/export
# echo low > /sys/class/gpio/gpio214/direction
# echo low > /sys/class/gpio/gpio204/direction
# echo low > /sys/class/gpio/gpio205/direction
# echo in > /sys/class/gpio/gpio14/direction
# echo in > /sys/class/gpio/gpio165/direction
# echo low > /sys/class/gpio/gpio236/direction
# echo low > /sys/class/gpio/gpio237/direction
# echo in > /sys/class/gpio/gpio212/direction
# echo in > /sys/class/gpio/gpio213/direction
# echo mode1 > /sys/kernel/debug/gpio_debug/gpio28/current_pinmux
# echo mode1 > /sys/kernel/debug/gpio_debug/gpio27/current_pinmux
# echo high > /sys/class/gpio/gpio214/direction
```

Now, it should be possible to use IO18 and IO19 for I2C communication.

Example 5: Configure IO10-13 for SPI connectivity

1. The shield pins are IO10, IO11, IO12 and IO13. Corresponding GPIO numbers are GPIO 111, 115, 114, and 109, respectively.
2. The function required is SPI. According to Table 1, other functions available on these pins are: GPIO, PWM
3. According to Table 3, GPIO 263 must be set to 1 to select GPIO/SPI, GPIO 240 must be set to 1 to select SPI, and GPIO 111 pin-mux must be set to 'mode1' to select SPI for IO10
4. According to Table 3, GPIO 262 must be set to 1 to select GPIO/SPI, GPIO 241 must be set to 1 to select SPI, and GPIO 115 pin-mux must be set to 'mode1' to select SPI for IO11
5. According to Table 3, GPIO 242 must be set to 1 to select SPI, and GPIO 114 pin-mux must be set to 'mode1' to select SPI for IO12
6. According to Table 3, GPIO 243 must be set to 1 to select SPI, and GPIO 109 pin-mux must be set to 'mode1' to select SPI for IO13
7. According to Table 4, GPIO 258 must be set to 1 to enable the output direction for IO10, GPIO 259 must be set to 1 to enable the output direction for IO11, GPIO 260 must be set to 0 to disable the output direction for IO12, and GPIO 261 must be set to 1 to enable the output direction for IO13.
8. According to Table 4, GPIOs 226-229 must be set as high-impedance inputs to disable the pull-up resistors for IO10-13.
9. According to Table 5, the TRI_STATE_ALL signal is controlled by GPIO 214

So, the commands in Linux to achieve this are as follows:

```
# echo 111 > /sys/class/gpio/expor
# echo 115 > /sys/class/gpio/export
# echo 114 > /sys/class/gpio/export
# echo 109 > /sys/class/gpio/export
# echo 263 > /sys/class/gpio/export
# echo 240 > /sys/class/gpio/export
# echo 262 > /sys/class/gpio/export
# echo 241 > /sys/class/gpio/export
# echo 242 > /sys/class/gpio/export
# echo 243 > /sys/class/gpio/export
# echo 258 > /sys/class/gpio/export
# echo 259 > /sys/class/gpio/export
# echo 260 > /sys/class/gpio/export
# echo 261 > /sys/class/gpio/export
# echo 226 > /sys/class/gpio/export
# echo 227 > /sys/class/gpio/export
# echo 228 > /sys/class/gpio/export
# echo 229 > /sys/class/gpio/export
# echo 214 > /sys/class/gpio/export
# echo low > /sys/class/gpio/gpio214/direction
# echo high > /sys/class/gpio/gpio263/direction
# echo high > /sys/class/gpio/gpio240/direction
# echo high > /sys/class/gpio/gpio262/direction
# echo high > /sys/class/gpio/gpio241/direction
# echo high > /sys/class/gpio/gpio242/direction
# echo high > /sys/class/gpio/gpio243/direction
# echo high > /sys/class/gpio/gpio258/direction
# echo high > /sys/class/gpio/gpio259/direction
# echo low > /sys/class/gpio/gpio260/direction
# echo high > /sys/class/gpio/gpio261/direction
# echo in > /sys/class/gpio/gpio226/direction
# echo in > /sys/class/gpio/gpio227/direction
# echo in > /sys/class/gpio/gpio228/direction
# echo in > /sys/class/gpio/gpio229/direction
# echo mode1 > /sys/kernel/debug/gpio_debug/gpio111/current_pinmux
# echo mode1 > /sys/kernel/debug/gpio_debug/gpio115/current_pinmux
# echo mode1 > /sys/kernel/debug/gpio_debug/gpio114/current_pinmux
# echo mode1 > /sys/kernel/debug/gpio_debug/gpio109/current_pinmux
# echo high > /sys/class/gpio/gpio214/direction
```

About Us

Emutex is a software engineering company based in Ireland. We are passionate about developing innovative software solutions for embedded systems. To learn more about us, please visit our company website: http://www.emutex.com

Details

Published: 10 October 2014

Hits: 101979

GPIO Intel Edison