Getting Started with Intel® Galileo Gen 2

Print

Intel® recently announced the release of their Galileo Gen 2 board - the successor to the original Galileo which was made available in October 2013.

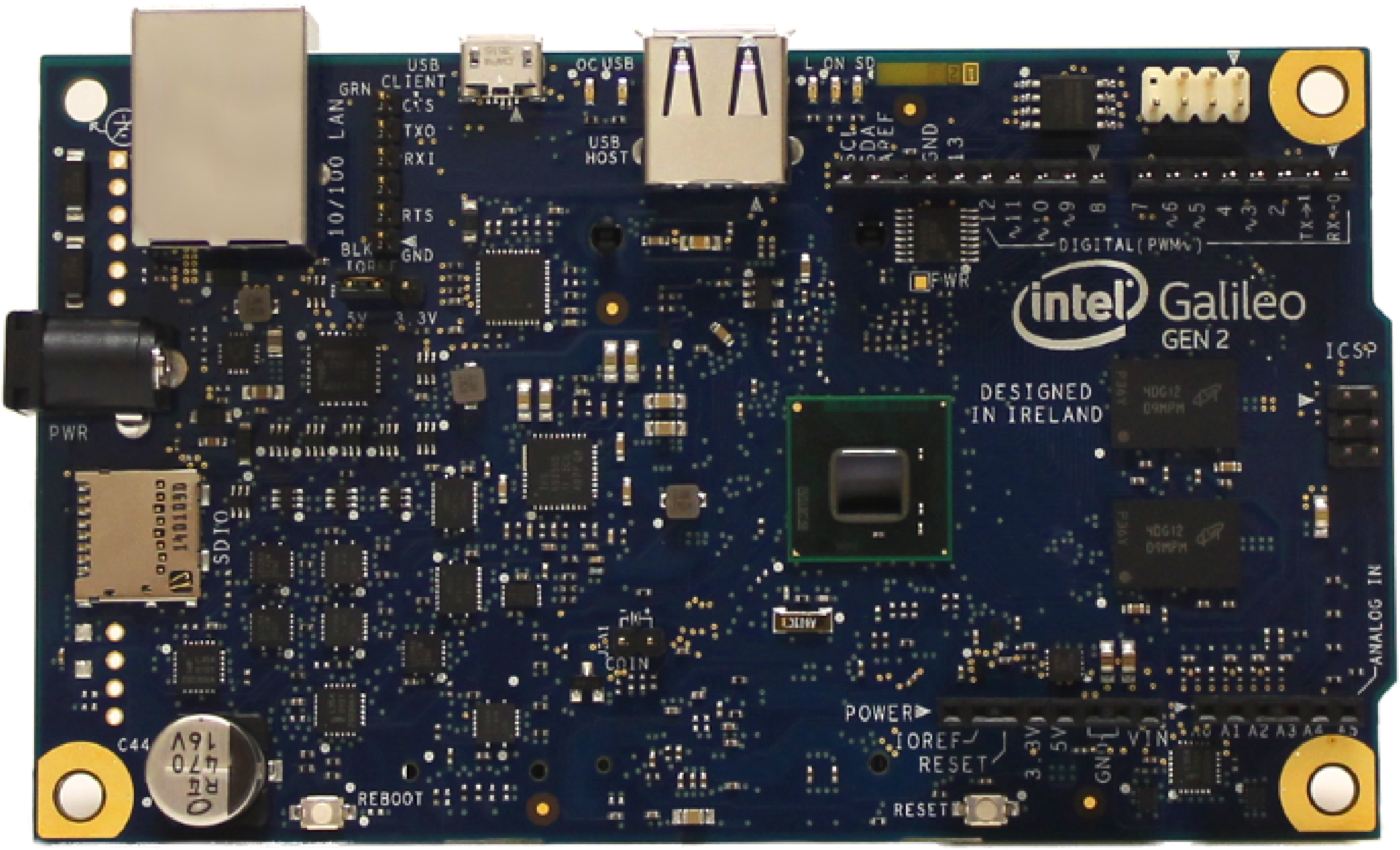

Figure 1
Intel® Galileo Gen 2 board, launched in July 2014

Information, software, schematics and other resources are available from Intel:
Release 1.0.2 of Galileo Software
Intel Galileo Gen 2 Development Board Documents

1. Introduction

Galileo Gen 2 retains the Arduino Uno shield interface and same basic feature set, but includes a significant number of additions and improvements over the original Galileo board.

This article shall discuss some of these changes and provide additional information for existing software users looking to migrate from the original Galileo board. Some familiarity with the Galileo board is assumed.

2. What's new in Galileo Gen 2

Some of the notable changes are:

- **New power supply options:**
  - **DC power supply jack now accepts a 7-15VDC input**
    NOTE: the 5V power supply for the original Galileo is not compatible with Galileo Gen 2
  - **Power-over-Ethernet (POE)**
    The board can be powered via the Ethernet cable, eliminating the need for a separate power supply when connected to a POE-enabled Ethernet switch or injector. A 12VDC supply voltage is required.
  - **VIN shield pin**
    The board can be powered from a shield, through the VIN pin on the Arduino shield header. Input voltage must be in the range 7-15VDC.
- **ADC performance**
  The Galileo Gen 2 uses a Texas Instruments ADS108S102 ADC, which allows for a 4x increase in ADC sampling performance in Linux compared with Galileo Gen1
- **GPIO performance**
  12 of the 20 Arduino shield pins are connected directly to the Quark X1000 SoC to allow significantly faster GPIO performance on those pins.
- **PWM resolution**
  Galileo Gen 2 employs an NXP PCA9685 PWM driver IC with 12-bit resolution, allowing for more fine-grained control on the PWM duty cycle.
- **FTDI-compatible Console UART port**
  The Console UART port (which provides access to the Linux serial console) now uses a standard 6-pin FTDI header and works at 3.3V TTL voltage levels. FTDI cable # TTL-232R-3V3 is recommended.
- **Full size USB Host port**
  The USB Host port connector now has a full-size Type-A receptacle.
- **Physical size**
  As a significant number of components were added on the board, the overall length has increased to 124mm

Many other minor improvements have been included in the Galileo Gen 2 design. For a complete list, please refer to the relevant documentation from Intel.

3. Arduino users – what you need to know

The Arduino IDE has already been updated to include support for Galileo Gen 2, since Release 1.0.2 of Galileo Software. Hardware differences are mostly hidden under the Arduino API implementations so, for most users, it should be a matter of simply recompiling existing sketches for the new board. To compile your sketches for Galileo Gen 2, simply download and install the latest IDE version and make sure to select the Intel® Galileo Gen 2 entry in the Tools->Board menu on the IDE before compiling and uploading your sketch:

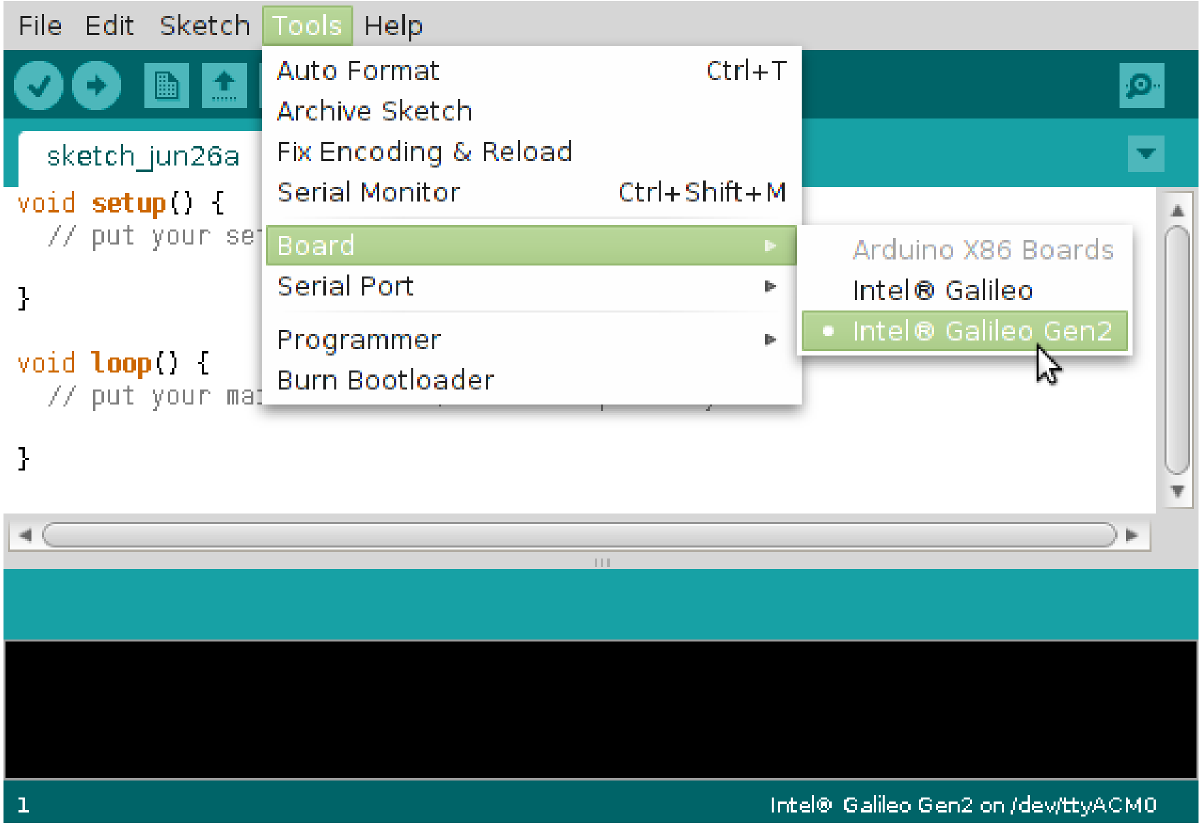

Figure 2
Arduino IDE now supports Galileo Gen 2

In addition to the existing feature set supported by Intel Galileo, the new Galileo Gen 2 platform adds the following notable features for Arduino users:

3.1 New UART serial port on shield pins IO2/IO3

The Quark X1000 SoC includes two UART interfaces. On the original Galileo, UART #0 was connected to shield pins IO0/IO1 (for use with Arduino shields), and UART #1 was reserved for the Linux system console. On Galileo Gen 2, UART #1 may also be connected to IO2/IO3 and detached from the Linux system console for use with Arduino shields.

To utilise this UART in an Arduino sketch, simply use the `Serial2` object. For example:

```
/* Sketch to illustrate the use of Serial2
 * Prints "Hello World" via the UART on pins 2/3
 */
void setup() {
  Serial2.begin(9600);
}
void loop() {
  Serial2.println(“Hello World”);
  delay(1000);
}
```

Note that, when UART #1 is in use by an Arduino sketch, it is no longer available for use as a Linux system console. To restore the Linux system console, it may be necessary to either:

1. Call Serial2.end() from your sketch when the UART is no longer needed, or
2. Load a sketch that does not call `Serial2.begin();` then reboot/power-cycle the Galileo Gen 2 board

In addition, the SoftwareSerial library has been ported for use on Galileo Gen 2. If configured to use shield pins 2-3, it will avail of UART #1 via the Serial2 object above. At this time, Galileo Gen 2 does not support “bit-banged” software implementations of the UART serial protocol on GPIO shield pins, and is thus restricted to work with this hardware-based UART only in its SoftwareSerial implementation.

3.2 Faster GPIO performance on shield pins IO0-IO6 and IO9-IO13

GPIO functionality on these shield pins is provided directly by the Quark X1000. Accessing these pins through the standard Arduino API functions digitalWrite() and digitalRead() allows a sketch to toggle these pins at a frequency of approximately 390 kHz. This performance capability was available on the original Galileo, but only on digital shield pins IO2 and IO3, and only when OUTPUT_FAST pin mode was selected for those pins.

For even better performance, the following Galileo-specific API functions may also be used:

```
fastGpioDigitalWrite(id)
fastGpioDigitalRead(id)
fastGpioDigitalRegSnapshot(id)
fastGpioDigitalRegWriteUnsafe(id, mask)
```
?

Some sample sketches code demonstrating the use of these:

```
/* Sketch to illustrate the use of fastGpioDigitalWrite
 * The following fast GPIO pin IDs may be used on Galileo Gen 2:
 *   GPIO_FAST_IO0 (digital pin 0)
 *   GPIO_FAST_IO1 (digital pin 1)
 *   GPIO_FAST_IO2 (digital pin 2)
 *   GPIO_FAST_IO3 (digital pin 3)
 *   GPIO_FAST_IO4 (digital pin 4)
 *   GPIO_FAST_IO5 (digital pin 5)
 *   GPIO_FAST_IO6 (digital pin 6)
 *   GPIO_FAST_IO9 (digital pin 9)
 *   GPIO_FAST_IO10 (digital pin 10)
 *   GPIO_FAST_IO11 (digital pin 11)
 *   GPIO_FAST_IO12 (digital pin 12)
 *   GPIO_FAST_IO13 (digital pin 13)
 */
void setup() {
  pinMode(2, OUTPUT);
}
void loop() {
  bool value = HIGH;
  while (1) {
    fastGpioDigitalWrite(GPIO_FAST_IO2, value);
    value = !value;
  }
}
```
?

```
/* Sketch to illustrate the use of fastGpioDigitalRegWriteUnsafe
* The following fast GPIO pin IDs may be used on Galileo Gen 2:
 *   GPIO_FAST_IO0 (digital pin 0)
 *   GPIO_FAST_IO1 (digital pin 1)
 *   GPIO_FAST_IO2 (digital pin 2)
 *   GPIO_FAST_IO3 (digital pin 3)
 *   GPIO_FAST_IO4 (digital pin 4)
 *   GPIO_FAST_IO5 (digital pin 5)
 *   GPIO_FAST_IO6 (digital pin 6)
 *   GPIO_FAST_IO9 (digital pin 9)
 *   GPIO_FAST_IO10 (digital pin 10)
 *   GPIO_FAST_IO11 (digital pin 11)
 *   GPIO_FAST_IO12 (digital pin 12)
```
?

```
 *   GPIO_FAST_IO13 (digital pin 13)
 */
void setup() {
  pinMode(3, OUTPUT);
}
void loop() {
  unsigned long snapshot = fastGpioDigitalRegSnapshot(GPIO_FAST_IO3);
  while (1) {
    snapshot ^= GPIO_FAST_IO3;
    fastGpioDigitalRegWriteUnsafe(GPIO_FAST_IO3, snapshot);
  }
}
```

Informal performance testing using the sketch examples above suggests the following performance capability:

`fastGpioDigitalWrite` **660-680 kHz** (varies depending on which digital pin is used)

`fastGpioDigitalRegWriteUnsafe` **1.2-2.9 MHz** (varies depending on which digital pin is used)

The latter function, `fastGpioDigitalRegWriteUnsafe`, is significantly faster because it involves only a single write to the underlying hardware register for the GPIO port. The application code is responsible for maintaining the state of that register (obtained initially using `fastGpioDigitalRegSnapshot`) and toggling only the bit for the respective digital pin.

`fastGpioDigitalWrite` is a slightly “safer”, but slower, alternative as it performs a read of the GPIO port register prior to modifying the bit for the selected digital pin. Nonetheless, it assumes that there are no other threads or processes running in the system is also accessing the same GPIO port registers concurrently. Caution must be exercised in particular when used in conjunction with interrupt handlers in the Arduino sketches (e.g. see attachInterrupt and attachTimerInterrupt) as these handlers run from within a separate thread in the Arduino sketch application.

Pin	Fast-Mode Frequency
0	2.9 MHz
1	2.9 MHz
2	2.9 MHz
3	2.9 MHz
4	1.2 MHz
5	1.2 MHz
6	1.2 MHz
7	N/A
8	N/A
9	1.2 MHz
10	2.9 MHz
11	1.2 MHz
12	2.9 MHz
13	1.2 MHz

Note that the performance figures above are not guaranteed. The Arduino sketch is not the only software process running in the system, and the underlying Linux OS is not designed for real-time applications in its current configuration. Therefore, some variation in the stability and consistency of the GPIO output signals should be expected.

3.3 Higher resolution capability for PWM duty-cycle

The standard Arduino API for PWM output, analogWrite() assumes by default an 8-bit resolution capability for the PWM outputs and so expects a value in the range 0-255. However, the Galileo Gen 2 PWM driver supports 12-bit resolution, which allows for a more fine-grained set of 4096 steps for the duty_cycle. To avail of this, there is a function called analogWriteResolution() that may be used to set the desired resolution to 12.

`analogWriteResolution()` is not enabled in the Release 1.0.2 by default, but can be enabled by adding to the file named
`‘hardware/arduino/x86/cores/arduino/AnalogIO.h’` in the Arduino IDE folder:

```
void adcInit(void);
uint32_t analogRead(uint32_t);<br>
void analogWriteResolution(int res);<br>
void analogReadResolution(uint32_t res);
void analogReference(uint8_t mode);
```

The following sample sketch, adapted from the “Fading” example sketch included with the Arduino IDE, illustrates the use of this function on Galileo Gen 2:

```
/* Sketch to illustrate use of analogWriteResolution to increase
 * PWM resolution on Galileo Gen 2
 * Based on the ‘Fading’ example included with the Arduino IDE
 */
void setup() {
  analogWriteResolution(12);
}
void loop() {
    // fade in from min to max in increments of 5 points:
  for(int fadeValue = 0 ; fadeValue <= 4095; fadeValue +=5) {
    // sets the value (range from 0 to 4095):
    analogWrite(ledPin, fadeValue);
    // wait for 5 milliseconds to see the dimming effect
    delay(5);
  }
  // fade out from max to min in increments of 5 points:
  for(int fadeValue = 4095 ; fadeValue >= 0; fadeValue -=5) {
    // sets the value (range from 0 to 4095):
    analogWrite(ledPin, fadeValue);
    // wait for 5 milliseconds to see the dimming effect
    delay(5);
  }
}
```

4. Linux users – what you need to know

4.1 Board-level device drivers

4.1.1 NXP PCAL9535A GPIO Device Driver

Galileo Gen 2 includes 3 PCAL9535A GPIO expanders, connected to the Quark X1000 via I2C at addresses 0x25, 0x26 and 0x27. They are supported by the **pca953x** kernel GPIO driver, extended for Galileo Gen 2 to add support for PCAL9535A interrupt handling.

Some additional points to note on the GPIO lines provided by the PCAL9535As on Galileo Gen 2:

- The majority of these GPIO lines are used to control internal components on the Galileo Gen 2 board, such as voltage-level-shifters, pull-up resistors and pin multiplexing control switches.
- 6 of these GPIO lines are used to provide direct GPIO functionality to shield pins IO14-1O19. These 6 shield pins are shared with the ADC and, for that reason, have 22nF capacitors connected to them which affect the performance of these pins in GPIO mode.
- 2 of these GPIO lines are connected to IO2/IO3 in parallel with GPIO lines from the Quark SoC. The reason is to provide an option to receive change-mode interrupts (triggered on both falling and rising edges) as this is not supported by those Quark GPIOs but is desirable for Arduino Uno compatibility.

File I/O access to the GPIO ports from user space is provided via entries in the following sysfs directory:

`/sys/class/gpio/`

4.1.2 NXP PCA9685 PWM/GPIO Device Driver

Galileo Gen 2 includes a 12-bit 16-channel PWM driver IC, connected to the Quark X1000 via I2C at address 0x47. It is supported by the **pca9685** kernel MFD (GPIO/PWM) driver, newly developed for Galileo Gen 2.

Some additional points to note on this PWM driver on Galileo Gen 2:

- Of the 16 PWM output channels on the PCA9685, 6 are used as normal PWM outputs on Galileo Gen 2, while the remaining 12 are used as GPIO outputs
- PWM period is shared across all channels, not configurable per channel.
- Valid period range is from 666666 to 41666666 nanoseconds

File I/O access to the PWM channels from user space is provided via entries in the following sysfs directory:

`/sys/class/pwm/pwmchip0/`

Further information on the use of PWM user-space interfaces can be found in relevant kernel documentation and other sources.

To adjust the PWM period for all channels at run-time, an extra sysfs attribute is provided by this driver (note that all channels will be disabled for a brief moment while the PWM period configuration is updated):

`/sys/class/pwm/pwmchip0/device/pwm_period`

4.1.3 TI ADS108S102 IIO-ADC Device Driver

Galileo Gen 2 includes a 6-channel 10-bit ADC, connected to the Quark X1000 via SPI bus 0, chip-select 0. It is supported by the **ads1x8s102** kernel IIO-ADC driver, newly developed for Galileo Gen 2.

File I/O access to the ADC channels from user space is provided via entries in the following sysfs directory:

`/sys/bus/spi/devices/iio:device0/`

Further information on the use of IIO user-space interfaces can be found in relevant kernel documentation and other sources.

Some additional points to note on this ADC on Galileo Gen 2:

- The ADC inputs are sourced from Arduino shield I/O pins IO14-IO19 (a.k.a. A0-A5)
- Correct pin configuration is required before attempting to read ADC measurements from these pins. Further information is provided elsewhere in this document.
- The ADC resolution is 10-bit, but the ADC data is up-scaled to 12-bits when presented via the IIO user-space interface. This is to allow driver compatibility with the ADC128S102, which is a 12-bit variant of the same ADC chip.

4.1.4. ON-Semi CAT24C08 EEPROM Device Driver

Galileo Gen 2 includes a 1kByte EEPROM, connected to the Quark X1000 via I2C at address 0x54. It is supported by the standard **at24** kernel eeprom driver.

File I/O access to the EEPROM from user space is provided via the following sysfs entry:

```
/sys/bus/i2c/devices/0-0054/eeprom
```

4.2 I/O pin configuration overview

Similar to the original Galileo board, some board-level configuration is needed before a given shield pin can be used for a specific purpose. For most of the pins, it is necessary to configure some or all of the following:

- Pin multiplexing switch(es), a.k.a "pin muxing", to connect a shield pin to the correct interface on the Quark SoC or a board-level IC, depending on what functionality is required for the pin
- Voltage level-shifter (a.k.a "buffer") direction as input or output, depending on whether the pin signal will be driven "out" by the Quark SoC or board-level IC, or driven "in" by an external circuit.
- Pull-up resistor, depending on whether the user wishes to have a 22k pull-up resistor connected to the pin or not.

All of the above are configurable using dedicated GPIO signals from the GPIO expanders, which in turn can be controlled according to the needs of the application software. The Arduino library provided for Galileo Gen 2 implements the pin configurations for the various Arduino application use cases. Control of the GPIO signals is achieved via the sysfs interface under /sys/class/gpio.

Sergey Kiselev wrote an excellent guide for GPIO pin configuration for the original Galileo board. Here, we aim to provide a similar guide for Galileo Gen 2:

4.3 Pin configuration options for Galileo Gen 2

Shield pin	Function	Linux	Level Shifter GPIO L: dir_out H: dir_in I: *	22k Pull-Up GPIO L: pulldown H: pullup I: off	Pin Mux 1 GPIO	Pin Mux 2 GPIO	Interrupt modes L: low-level H:high-level R:rising-edge F:falling-edge B:both edges
IO0	UART0 RX	ttyS0	gpio32	gpio33	-	-	-
	GPIO	gpio11			-	-	L/H/R/F
IO1	UART0 TX	ttyS0	gpio28	gpio29	gpio45 (H)	-	-
	GPIO	gpio12			gpio45 (L)	-	L/H/R/F
IO2	UART1 RX	ttyS1	gpio34	gpio35	gpio77 (H)	-	
	GPIO	gpio13			gpio77 (L)	-	L/H/R/F
	GPIO	gpio61	-		gpio77 (L)	-	R/F/B
IO3	UART1 TX	ttyS1	gpio16	gpio17	gpio76(H)	-	-
	GPIO	gpio14			gpio76(L)	gpio64(L)	L/H/R/F
	PWM	pwm1			gpio76(L)	gpio64(H)	-
	GPIO	gpio62	-		gpio76(L)	gpio64(L)	R/F/B
IO4	GPIO	gpio6	gpio36	gpio37	-	-	R/F/B
IO5	GPIO	gpio0	gpio18	gpio19	gpio66(L)	-	R/F/B
	PWM	pwm3			gpio66(H)	-	-
IO6	GPIO	gpio1	gpio20	gpio21	gpio68(L)	-	R/F/B
	PWM	pwm5			gpio68(H)	-	-
IO7	GPIO	gpio38	-	gpio39	-	-	-
IO8	GPIO	gpio40	-	gpio41	-	-	-
IO9	GPIO	gpio4	gpio22	gpio23	gpio70(L)	-	R/F/B
	PWM	pwm7			gpio70(L)	-	-
	GPIO	gpio10			gpio74(L)	-	L/H/R/F

IO10	PWM	pwm11	gpio26	gpio27	gpio74(H)	-	-
IO11	GPIO	gpio5	gpio24	gpio25	gpio44(L)	gpio72(L)	R/F/B
	SPI MOSI	spidev1.0			gpio44(H)	gpio72(L)	-
	PWM	pwm9			-	gpio72(H)	-
IO12	GPIO	gpio15	gpio42	gpio43	-	-	L/H/R/F
	SPI MISO	spidev1.0					-
IO13	GPIO	gpio7	gpio30	gpio31	gpio46(L)	-	R/F/B
	SPI SCK	spidev1.0			gpio46(H)	-	-
IO14	GPIO	gpio48	-	gpio49	-	-	R/F/B
	ADC A0	in_voltage0_raw					-
IO15	GPIO	gpio50	-	gpio51	-	-	R/F/B
	ADC A1	in_voltage1_raw					-
IO16	GPIO	gpio52	-	gpio53	-	-	R/F/B
	ADC A2	in_voltage2_raw					-
IO17	GPIO	gpio54	-	gpio55	-	-	R/F/B
	ADC A3	in_voltage3_raw					-
IO18	GPIO	gpio56	-	gpio57	gpio60(H)	gpio78(H)	R/F/B
	ADC A4	in_voltage4_raw			gpio60(H)	gpio78(L)	-
	I2C SDA	i2c-0			gpio60(L)	-	-
IO19	GPIO	gpio58	-	gpio59	gpio60(H)	gpio79(H)	R/F/B
	ADC A4	in_voltage5_raw			gpio60(H)	gpio79(L)	-
	I2C SCL	i2c-0			gpio60(L)	-	-

Note: with the exception of the "Interrupt Modes" column, the following single-letter convention is used in the table above to indicate the state of a GPIO pin:

"L" The GPIO is configured as an output, with output level as LOW

"H" The GPIO is configured as an output, with output level as HIGH

"I" The GPIO is configured as a high-impedance input

4.4 Enabling chip-select line for spidev1.0

Typically, the SPI controller driver in Linux is responsible for implicitly controlling the chip-select line for a given device. For Galileo Gen 2, GPIO #10 is intended to be used as the chip-select signal when connecting to an external SPI device via the shield I/O pins. As such, the handling of the chip-select signalling is done underneath the SPI API, transparently to client code.

However, the Arduino library employs a different paradigm for this. Rather than assuming a dedicated chip-select line managed by the SPI controller, Arduino assumes that any GPIO output can be used for chip-select signalling and leaves the choice to the sketch application.

This creates a conflict between the Linux and Arduino use-cases. For Linux, the SPI controller driver wants to "own" GPIO #10, but for the Arduino use-case this must be left free as a general purpose I/O line. So, the SPI controller driver is instructed on start-up by default to NOT use GPIO #10, thereby satisfying the Arduino case.

To override this and use SPI at the Linux level (e.g. via `/dev/spidev1.0`) with implicit chip-select control, the user must first allow the SPI controller driver to use GPIO #10. To do this, edit the grub.conf file (i.e. `/media/mmcblk0p1/boot/grub/grub.conf`), append the following parameter to the kernel command line (the line beginning with ' kernel' for the relevant grub entry), and reboot:

```
intel_qrk_plat_galileo_Gen 2.gpio_cs=1
```

To verify this, the following command should return an error since the GPIO is now reserved by the SPI driver:

```
echo 10 > /sys/class/gpio/export
```

5. 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

Written by Dan O Donovan

Published: 30 July 2014

Intel Galileo