java-gpio

Version 1.0.0 – 16 February 2016
Kit Bishop

Document History

Version	Date	Change Details
Version 1.0.0	16 February 2016	Initial version

Contents

1. Background

java-gpio is Java code for accessing the Omega GPIO pins.

The rationale for producing this code was a desire to have GPIO access from Java code.

java-gpio consists of four main components:

- **java-gpio-lib** – a Java library package containing the Java classes used to interact with GPIO pins using code available in the separately provided **libnew-gpio** library
- **libnew-gpio-jni** – a C++ dynamic link library that provides a JNI bridge between **java-gpio-lib** and the C++ code in **new-gpio** using **libnew-gpio** library used to actually access the GPIO pins
- **java-gpio** – contains a Java program for interacting with GPIO pins using **java-gpio-lib** – this can be considered equivalent to the Omega supplied **fast-gpio** program but with extensions and to the separately provided **new-gpio** program
- **java-expled** – a contains a Java program for controlling the expansion dock led using **java-gpio-lib** – this can be considered equivalent to the Omega supplied **expled** script but with extensions and to the separately provided **new-expled** program

NOTES:

a. **libnew-gpio-jni** is provided in 2 forms:
    - **libnew-gpio-jni-all.so** – which contains the code from **libnew-gpio** which does not need separately installing
    - **libnew-gpio-jni.so** – which does not contain **libnew-gpio** which needs separately installing

b. **java-gpio-lib**, **java-gpio** and **java-expled** are each provided in 2 forms:

        - **withlib** – which contains all required components within the jar file so no additional components need installing
        - **nolib** – which does not contain additional required components which need separately installing

    c. Information about and files relating to the components libnew-gpio, new-gpio and new-expled referred to above can be found in GitHub at https://github.com/KitBishop/new-gpio

These components are described in more details in this document, as are the files contained in the package supplied with this document.

The software was developed on a KUbuntu-14.04 system running in a VirtualBox VM and uses the following tools for building the code:

- The C++ code in **libnew-gpio-jni** is built using the OpenWrt toolchain:

  The toolchain used can be found at:

  - https://s3-us-west-2.amazonaws.com/onion-cdn/community/openwrt/OpenWrt-Toolchain-ar71xx-generic_gcc-4.8-linaro_uClibc-0.9.33.2.Linux-x86_64.tar.bz2

  and details of its setup and usage can be found at:

  - https://community.onion.io/topic/9/how-to-install-gcc/22
- The Java code is built using standard Java JDK 8, though any version from Java JDK 5 onwards should be usable

java-gpio comes with **NO GUARANTEES** ☺ but you are free to use it and do what you want with it.

2. Pre-requisites

To use **java-gpio**, your Omega **must** fulfil the following pre-requisites:

- Must have been upgraded to version **0.0.6-b265** or later
- Must have the **kmod-gpio-irq** package installed by running:

  ```
  opkg update
  opkg install kmod-gpio-irq
  ```
- Must have sufficient disk space to install **jamvm** (a lightweight implementation of a full Java Virtual Machine) as in next point.
  Because **jamvm** needs more disk space to install than the Omega standardly has, you must extend the disk space using a USB drive as described in: https://wiki.onion.io/Tutorials/Using-USB-Storage-as-Rootfs
- Must have the **jamvm** package installed. This is a lightweight implementation of a full Java Virtual Machine
  This may be installed by running:

  ```
  opkg install jamvm
  ```

- Optionally, it is useful (but not absolutely necessary) to extend the Omega's working memory with swap space as described in: https://community.onion.io/topic/533/using-linux-swap-space-on-the-omega

3. Files Supplied

java-gpio is supplied in files in a GitHub repository at https://github.com/KitBishop/java-gpio. This repository contains the following directories and files :

- **java-gpio.pdf** – this documentation as a PDF file
- **source** – directory containing all source files and make files:
  - **libnew-gpio-jni** – directory containing all C++ sources and Makefile for **libnew-gpio-jni** library
    - **Makefile** – the Makefile for **libnew-gpio-jni** library
    - **hdr** – directory containing header (***.h**) files for **libnew-gpio-jni** library
    - **src** – directory containing source (***.cpp**) files for **libnew-gpio-jni** library
  - **java-gpio-lib** – directory containing all Java sources and Makefile for **java-gpio-lib** jar library
    - **Makefile** – the Makefile for **java-gpio-lib** jar library
    - **src** – directory containing source (***.java**) files for **java-gpio-lib** jar library
      Note All Java files are within the package path: **nz.net.bishop.omega.gpio**
      This naming follows the convention of including the domain name within the package path. If required, you may move these files to a different path and make the associated changes in the source files.
  - **java-gpio** – directory containing all Java sources and Makefile for **java-gpio** program jar
    - **Makefile** – the Makefile for **java-gpio** program jar
    - **src** – directory containing source (***.java**) files for **java-gpio** program jar
  - **jar-expled** – directory containing all Java sources and Makefile for **java-expled** program jar
    - **Makefile** – the Makefile for **java-expled** program jar
    - **src** – directory containing source (***.java**) files for **java-expled** program jar
  - **java-gpio-template** – directory containing template source code that can be used as a basis for a new program jar that uses **java-gpio-lib**
    - **Makefile** – the Makefile for the template – will need modifying as per information in the first few lines of the file
    - **src** – directory to contain all source (***.java**) files for the template program jar
      Initially contains a very basic skeleton **my_main_class.java** source file
- **bin** – directory containing pre-built binary files:
  - **libnew-gpio-jni** – directory containing the compiled **libnew-gpio-jni** library files:
    - **libnew-gpio-jni-all.so** – the dynamic link library for **libnew-gpio-jni** which also includes the **new-gpio** library code
    - **libnew-gpio-jni.so** – the dynamic link library for **libnew-gpio-jni** which does not include any of the **new-gpio** library code.
  - **java-gpio-lib** – directory containing the built **java-gpio-lib** library jar files

        - **withlib** – directory containing the **java-gpio-lib** library jar file which also includes all required **libnew-gpio-jni** and **libnew-gpio** dynamic link library code
        - **nolib** – directory containing the **java-gpio-lib** library jar file which does not include any required **libnew-gpio-jni** and **libnew-gpio** dynamic link library code
    - **java-gpio** – directory containing the built **java-gpio** program jar files
        - **withlib** – directory containing the **java-gpio** program jar file which also includes all required **libnew-gpio-jni** and **libnew-gpio** dynamic link library code and the required **java-gpio-lib** java library code
        - **nolib** – directory containing the **java-gpio** program jar file which does not include any required **libnew-gpio-jni** and **libnew-gpio** dynamic link library code nor the required **java-gpio-lib** java library code
    - **java-expled** – directory containing the built **java-expled** program jar files
        - **withlib** – directory containing the **java-expled** program jar file which also includes all required **libnew-gpio-jni** and **libnew-gpio** dynamic link library code and the required **java-gpio-lib** java library code
        - **nolib** – directory containing the **java-expled** program jar file which does not include any required **libnew-gpio-jni** and **libnew-gpio** dynamic link library code nor the required **java-gpio-lib** java library code

4. Usage and Installation

Installing the software is simple. It primarily consists of copying the library and test program to suitable locations on your Omega.

4.1. Using java-gpio-lib Java Library

To build your own Java code that uses the **java-gpio-lib** Java library the file **java-gpio-lib.jar** must be on the **classpath** when running **javac**

See the Makefile in **java-gpio-template** for an example.

4.2. Using and Installing libnew-gpio-jni Dynamic Link Libraries

Depending upon how you wish to run your Java program (see next section) you may need to install **libnew-gpio-jni.so** or **libnew-gpio-jni-all.so** dynamic library.

If you need either of these libraries you need to copy the relevant file to the **/lib** directory on your Omega.

Alternatively, you can copy the library to any location that may be set up in any **LD_LIBRARY_PATH** directory on your Omega. For example, I use the following for testing:

- Created directory **/root/lib**
- Copied the library to **/root/lib**
- Added the following lines to my **/etc/profile** file:

```
LD_LIBRARY_PATH=/root/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

4.3. Installing the java-gpio and java-expled and your own Java Programs

Depending upon how you want to allocate resource usage on your Omega, there are 4 methods that can be used to install a Java program that uses **java-gpio** code.

4.3.1. Standalone Java Program Installation

For a standalone installation of a Java program (either supplied **java-gpio**, **java-expled** or your own Java program built using instructions supplied elsewhere herein) all you need to do is copy the relevant **<program>.jar** file from the build **withlib** directory to any suitable directory on your Omega.

4.3.2. Java Program and java-gpio-lib only

To install just the Java program and the **java-gpio-lib** Java library you only need to install as follows:

a. Copy the relevant **<program>.jar** file from the build **nolib** directory to any suitable directory on your Omega

b. Copy the **java-gpio-lib.jar** file from the build **withlib** directory to the same directory as the **<program>.jar** file

4.3.3. Java Program with java-gpio-lib and libnew-gpio-jni only

To install just Java program, the **java-gpio-lib** Java library and the **libnew-gpio-jni** dynamic link library you only need to install as follows:

a. Copy the relevant **<program>.jar** file from the build **nolib** directory to any suitable directory on your Omega
b. Copy the **java-gpio-lib.jar** file from the build **nolib** directory to the same as the **<program>.jar** file to the same directory as the **<program>.jar** file
c. Install **libnew-gpio-jni-all.so** file as described above.

4.3.4. Java Program with all components separately

To install the Java program with all components installed separately you need to install as follows:

a. Copy the relevant **<program>.jar** file from the build **nolib** directory to any suitable directory on your Omega
b. Copy the **java-gpio-lib.jar** file from the build **nolib** directory to the same as the **<program>.jar** file to the same directory as the **<program>.jar** file
c. Install **libnew-gpio-jni.so** file as described above.
d. Install **libnew-gpio.so** dynamic link library as is described in the documentation for **new-gpio**

4.4. Running installed programs

To run any Java program installed as above, simply use a command of the form:

```
jamvm –jar <program>.jar <any-parameters>
```

Where **<any-parameters>** are parameters to be passed to the Java program

5. Using Makefiles

Each component in the **source** directory contains a **Makefile** that can be used to build the relevant component.

5.1. Modify Makefile

Each **Makefile** may or will need modifying. For each of the supplied **Makefile** files, instructions for modifications that must or may need making are described in the first few lines of the relevant **Makefile**.

5.2. Makefile targets

Each **Makefile** implements the same set of targets:

- **make**

The default target. Performs a complete build of versions both with and without relevant library components included in the build.
This is directly equivalent to:

make nolib withlib

- **make nolib**
  Performs a complete build without including any relevant libraries etc. in the built code.

- **make dynamic**
  Performs a complete build including all relevant libraries etc. in the built code.

- **make clean**
  Removes all previous build files, both nolib and withlib versions.
  This is directly equivalent to:

  make clean-nolib clean-withlib

- **make clean-nolib**
  Removes all previous build files for nolib versions only
  .
- **make clean-withlib**
  Removes all previous build files for withlib versions only

If the following is added to the **make** command line:

builddep=1

thenany components that this particular component depends on will also be built before building the this particular component.

6. Description of the java-gpio-lib Java Library

The **java-gpio-lib** Java library contains several Java classes etc. for Java access to the GPIO pin functionality. The source files for these are as follows:

- Miscellaneous support files**:**
  - **GPIODirection.java** – a java enum representing a GPIO pin direction
  - **GPIOResult.java** – a Java enum representing the returned result of a GPIO operation
  - **GPIOIrqType.java** – a Java enum representing an interrupt type
  - **IGPIOIrqHanlder.java** – a Java interface used to represent the methods to be implemented by any class to be used for handling GPIO interrupt
- **GPIOAccess.java** – a Java class with only static methods used for direct access to the Omega GPIO hardware.
- **GPIOPin.java** – a Java class used to represent instances of a GPIO pin.
- **RGBLED.java** – a Java class used to represent instances of an RGB led (such as the led on the expansion dock).

The contents of these components are described in following sections.

6.1. Miscellaneous support files

These files contain code of some basic types used elsewhere.

6.1.1. enum GPIOResult

enum GPIOResult is used to represent the returned result of GPIO operations. It has values:

- **GPIO_OK(0)** – represents a successful result
- **GPIO_BAD_ACCESS(1)** – indicates a failure to access the GPIO hardware registers
- **GPIO_INVALID_PIN(2)** – indicates that a pin number has been used that is not accessible by GPIO
- **GPIO_INVALID_OP(3)** – indicates that an invalid operation has been attempted on a pin. E.G. attempting to set a pin that is in input mode, or reading a pin that is in output mode

6.1.2. enum GPIODirection

enum GPIODirection is used to represent the direction for a GPIO pin. It has values:

- **GPIO_INPUT(false)** – represents an input pin
- **GPIO_OUTPUT(true)** – represents an output pin

6.1.3. enum GPIOIrqType

enum GPIOIrqType is used to represent the type of interrupt used for a GPIO pin. It has values:

- **GPIO_IRQ_NONE(0)** – indicates no interrupt
- **GPIO_IRQ_RISING(1)** – indicates an interrupt on the rising edge (i.e. low to high change) on a pin

- **GPIO_IRQ_FALLING(2)** – indicates an interrupt on the falling edge (i.e. high to low change) on a pin
- **GPIO_IRQ_BOTH(3)** – indicates an interrupt on the either of a rising edge or on a falling edge on a pin

6.1.4. interface IGPIOIrqHandler

IGPIOIrqHandler is an interface that must be implemented by any class whose instances are to be used as objects to handle GPIO interrupts.

Any such interrupt handler class must implement the method:

void handleInterrupt(int pinNum, GPIOIrqType type);

When an instance of any such class is used to handle an interrupt, the **handleInterrupt** method of the object is called to handle the interrupt. The **handleInterrupt** method has the following characteristics:

Parameters:

- **int pinNum** – the number of the pin for which the handler is being called
- **GPIOIrqType type** – the type of interrupt for which the handler is being called

Returns:

- <none>

While the parameters passed are not strictly speaking required for interrupt handling in general, they are provided to allow the same handler objects to be used for multiple pins and interrupt types. Any actual handler can then (if required) control its action depending upon the pin and interrupt type. If no such distinction is required, these parameters can be ignored in the actual implementation of a passed handler.

6.2. Class GPIOAccess

The **GPIOAccess** class is the main class by which all access is made to the GPIO hardware.

The class contains only static methods and no instance of this class will ever actually be created hence there are no constructors or destructors.

6.2.1. GPIOAccess Public Methods

Note that in general, the success or failure of any method can be ascertained by calling the **getlastResult** method immediately after the call to the particular method.

6.2.1.1. *static public void setDirection(int pinNum, GPIODirection dir);*

Sets the direction for a pin.

Parameters:

- **int pinNum** – the number of the pin

- **GPIODirection dir** – the direction to set the pin to

Returns:

- <none>

6.2.1.2. *static public GPIODirection getDirection(int pinNum);*

Queries the direction of a pin.

Parameters:

- **int pinNum** – the number of the pin

Returns:

- The current direction of the pin

6.2.1.3. *static public void set(int pinNum, int value);*

Sets the output state of a pin. Only valid for output pins.

Parameters:

- **int pinNum** – the number of the pin
- **int value** – the value to set the pin to

Returns:

- <none>

6.2.1.4. *static public int get(int pinNum);*

Queries the input state of a pin. Only valid for input pins.

Parameters:

- **int pinNum** – the number of the pin

Returns:

- The current state of the pin

6.2.1.5. *static public void setPWM(int pinNum, int freq, int duty);*

Starts the PWM output on a pin with the given frequency and duty values.

NOTE: PWM output on a pin is run on a separate thread for that pin. When this method is called the thread will be started (or its data updated if it is already running) and the call to the method then returns. The thread continues to run until one of the following occurs:

- the **stopPWM** method is called for the pin
- the process that started the thread (i.e. made the call to this method) terminates

Parameters:

- **int pinNum** – the number of the pin
- **int freq** – sets the PWM frequency in Hz
- **int duty** – sets the PWM duty cycle percentage

Returns:

- <none>

6.2.1.6. *static public void startPWM(int pinNum);*

Starts the PWM output on a pin using the last used frequency and duty values for the pin.

Parameters:

- **int pinNum** – the number of the pin

Returns:

- <none>

6.2.1.7. *static public void stopPWM(int pinNum);*

Stops any current PWM output on a pin.

Parameters:

- **int pinNum** – the number of the pin

Returns:

- <none>

6.2.1.8. *static public int getPWMFreq(int pinNum);*

Returns the currently set PWM frequency for a pin.

Parameters:

- **int pinNum** – the number of the pin

Returns:

- The PWM frequency in Hz

6.2.1.9. *static public int getPWMDuty(int pinNum);*

Returns the currently set PWM duty cycle percentage for a pin

Parameters:

- **int pinNum** – the number of the pin

Returns:

- The PWM duty cycle percentage

6.2.1.10. static public void setIrq(int pinNum, GPIO_Irq_Type type, IGPIOIrqHandler handler);

Setups up interrupt handling for a pin with a given handler object.

NOTE: IRQ handling on a pin is run on a separate thread for that pin. When this method is called the thread will be started and the call to the method then returns. The thread continues to run and to call the handler method of the object whenever the relevant interrupt occurs until one of the following occurs:

- the **resetIrq** method is called for the pin
- the process that started the thread (i.e. made the call to this method) terminates

Parameters:

- **int pinNum** – the number of the pin
- **GPIOIrqType type** – specifies the interrupt type to apply
- **IGPIOIrqHandler handler** – an object that implements the **IGPIOIrqHandler** interface to be used to handle the interrupt

Returns:

- <none>

6.2.1.11. static public void setIrq(int pinNum, GPIO_Irq_Type type, IGPIOIrqHandler handler, long int debounceMs);

Setups up interrupt handling for a pin with a given handler object.

NOTE: IRQ handling on a pin is run on a separate thread for that pin. When this method is called the thread will be started and the call to the method then returns. The thread continues to run and to call the handler method of the object whenever the relevant interrupt occurs until one of the following occurs:

- the **resetIrq** method is called for the pin
- the process that started the thread (i.e. made the call to this method) terminates

Parameters:

- **int pinNum** – the number of the pin
- **GPIOIrqType type** – specifies the interrupt type to apply
- **IGPIOIrqHandler handler** – an object that implements the **IGPIOIrqHandler** interface to be used to handle the interrupt
- **long int debounceMs = 0** – specifies an optional debounce period in milliseconds to be applied. If value is **0** no debounce handling is applied
  Debounce handling is used to deal with interrupts that come from a potentially noisy mechanical source such as buttons or switches.

When a non-zero value is used for **debounceMs**, any input signal changes that would normally cause an interrupt but which occur within a time less than the **debounceMs** time since the previous signal change will be ignored so as not to trigger handling of a false signal.

Returns:

- <none>

6.2.1.12. static public void resetIrq(int pinNum);
Removes any interrupt handling for a pin.

Parameters:

- **int pinNum** – the number of the pin

Returns:

- <none>

6.2.1.13. static public void enableIrq(int pinNum);
Enables interrupt handling for a pin that has previously been disabled by **disableIrq**.

Parameters:

- **int pinNum** – the number of the pin

Returns:

- <none>

6.2.1.14. static public void disableIrq(int pinNum);
Disables interrupt handling for a pin that has previously been enabled by **enableIrq**.

Parameters:

- **int pinNum** – the number of the pin

Returns:

- <none>

6.2.1.15. static public void enableIrq(int pinNum, boolean enable);
Enables or Disables interrupt handling for a pin according to parameter.

Parameters:

- **int pinNum** – the number of the pin
- **boolean enable** – indicates whether interrupt handling is to be enabled (**true**) or disabled (**false**)

Returns:

- <none>

6.2.1.16. *static public boolean irqEnabled(int pinNum);*

Returns an indication as to whether interrupt handling is currently enabled or disabled for a pin.

Parameters:

- **int pinNum** – the number of the pin

Returns:

- **true** – interrupt handling is enabled, **false** – interrupt handling is disabled

6.2.1.17. *static public GPIOIrqType getIrqType(int pinNum);*

Returns the current interrupt type for a pin.

Parameters:

- **int pinNum** – the number of the pin

Returns:

- the interrupt type

6.2.1.18. *static public IGPIOIrqHandler getIrqHandler(int pinNum);*

Returns the any currently established interrupt handler object for a pin.

Parameters:

- **int pinNum** – the number of the pin

Returns:

- the interrupt handler object

6.2.1.19. *static public boolean isPWMRunning(int pinNum);*

Returns an indication of whether or not PWM is currently running on a pin

Parameters:

- **int pinNum** – the number of the pin

Returns:

- **true** if PWM is running; **false** if PWM is not running

6.2.1.20. *static public boolean isAccessOk();*

Returns an indication as to whether or not the GPIO hardware is accessible.

Parameters:

- <none>

Returns:

- **true** or **false –** indicating whether or not the hardware is accessible

6.2.1.21. static public GPIOResult getLastResult();

Returns the result of the latest call to other methods.

Parameters:

- <none>

Returns:

- The result of the last method call

6.3. Class GPIOPin

The **GPIOPin** class represents instances of a GPIO pin.

6.3.1. GPIOPin Constructor

6.3.1.1. Constructor - GPIOPin(int pinNum);

Creates a new GPIOPin instance for a given pin.

Parameters:

- **int pinNum** – the pin number

6.3.2. GPIOPin Public Methods

Note that in general, the success or failure of any method can be ascertained by calling the **getlastResult** immediately after the call to the particular method.

6.3.2.1. public void setDirection(GPIODirection dir);

Sets the direction of the GPIOPin.

Parameters:

- **GPIO_Direction dir** – the direction to set the pin to

Returns:

- <none>

6.3.2.2. public GPIOResult getDirection();

Obtains the current direction of the GPIOPin

Parameters:

- <none>

Returns:

- The current direction of the pin

6.3.2.3. public void set(int value);

Sets the value of the GPIOPin.

Parameters:

- **int value** – the value to set the pin to

Returns:

- <none>

6.3.2.4. public int get();

Directly returns the value of the GPIOPin.

Parameters:

- <none>

Returns:

- the current value of the pin

6.3.2.5. public void setPWM(int freq, int duty);

Starts the PWM output on the GPIOPin with the given frequency and duty values.

NOTE: PWM output on a pin is run on a separate thread for that pin. When this method is called the thread will be started (or its data updated if it is already running) and the call to the method then returns. The thread continues to run until one of the following occurs:

- the **stopPWM** method is called for the pin
- the GPIOPin destructor for the pin is called
- the process that started the thread (i.e. made the call to this method) terminates

Parameters:

- **int freq** – sets the PWM frequency in Hz
- **int duty** – sets the PWM duty cycle percentage

Returns:

- <none>

6.3.2.6. public void startPWM();

Starts the PWM output on the GPIOPin using the last used frequency and duty values

Parameters:

- <none>

Returns:

- <none>

6.3.2.7. public void stopPWM();

Stops any current PWM output on the GPIOPin

Parameters:

- <none>

Returns:

- <none>

6.3.2.8. public int getPWMFreq();

Returns the currently set PWM frequency for the GPIOPin

Parameters:

- <none>

Returns:

- The PWM frequency in Hz

6.3.2.9. public int getPWMDuty();

Returns the currently set PWM duty cycle percentage for the GPIOPin

Parameters:

- <none>

Returns:

- The PWM duty cycle percentage

6.3.2.10. public boolean isPWMRunning();

Returns an indication of whether or not PWM is currently running on the GPIOPin

Parameters:

- <none>

Returns:

- **true** if PWM is running; **false** if PWM is not running

6.3.2.11. public void setIrq(GPIOIrqType type, IGPIOIrqHandler handler);

void setIrq(GPIOIrqType type, IGPIOIrqHandler handler);
void setIrq(GPIOIrqType type, IGPIOIrqHandler handler, int debounceMs);
Setups up interrupt handling for the GPIOPin with a given handler object.

NOTE: IRQ handling on a pin is run on a separate thread for that pin. When this method is called the thread will be started and the call to the method then returns. The thread continues to run and to call the handler method of the object whenever the relevant interrupt occurs until one of the following occurs:

- the **resetIrq** method is called for the pin
- the process that started the thread (i.e. made the call to this method) terminates

Parameters:

- **GPIOIrqType type** – specifies the interrupt type to apply
- **IGPIOIrqHandler handler** – an object of a class that implements the **IGPIOIrqHandler** interface to be used to handle the interrupt

Returns:

- <none>

6.3.2.12. public void setIrq(GPIOIrqType type, GPIOIrqHandler handler, int debounceMs);

Setups up interrupt handling for the GPIOPin with a given handler object.

NOTE: IRQ handling on a pin is run on a separate thread for that pin. When this method is called the thread will be started and the call to the method then returns. The thread continues to run and to call the handler method of the object whenever the relevant interrupt occurs until one of the following occurs:

- the **resetIrq** method is called for the pin
- the process that started the thread (i.e. made the call to this method) terminates

Parameters:

- **GPIOIrqType type** – specifies the interrupt type to apply
- **IGPIOIrqHandler handler** – an object of a class that implements the **IGPIOIrqHandler** interface to be used to handle the interrupt
- **int debounceMs** – specifies an optional debounce period in milliseconds to be applied.
  If value is **0** no debounce handling is applied
  Debounce handling is used to deal with interrupts that come from a potentially noisy mechanical source such as buttons or switches.

When a non-zero value is used for **debounceMs**, any input signal changes that would normally cause an interrupt but which occur within a time less than the **debounceMs** time since the previous signal change will be ignored so as not to trigger handling of a false signal.

Returns:

- <none>

6.3.2.13. *public void resetIrq();*

Removes any interrupt handling for the GPIOPin.

Parameters:

- <none>

Returns:

- <none>

6.3.2.14. *public void enableIrq();*

Enables interrupt handling for the GPIOPin that has previously been disabled by **disableIrq**.

Parameters:

- <none>

Returns:

- <none>

6.3.2.15. *public void disableIrq();*

Disables interrupt handling for the GPIOPin that has previously been enabled by **enableIrq**.

Parameters:

- <none>

Returns:

- <none>

6.3.2.16. *public void enableIrq(boolean enable);*

Enables or Disables interrupt handling for the GPIOPin according to parameter.

Parameters:

- **boolean enable** – indicates whether interrupt handling is to be enabled (**true**) or disabled (**false**)

Returns:

- <none>

6.3.2.17. *public boolean irqEnabled();*
Returns an indication as to whether interrupt handling is currently enabled or disabled for the GPIOPin.

Parameters:

- <none>

Returns:

- **true** – interrupt handling is enabled, **false** – interrupt handling is disabled

6.3.2.18. *public GPIOIrqType getIrqType();*
Returns the current interrupt type for the GPIOPin.

Parameters:

- <none>

Returns:

- the interrupt type

6.3.2.19. *public GPIOIrqHandler getIrqHandler();*
Returns the any currently established interrupt handler object for the GPIOPin.

Parameters:

- <none>

Returns:

- pointer to the interrupt handler object

6.3.2.20. *public int getPinNumber();*
Returns the pin number for the GPIOPin

Parameters:

- <none>

Returns:

- The pin number

6.3.2.21. *public GPIOResult getLastResult();*
Returns the result of the latest call to other methods.

Parameters:

- <none>

<u>Returns:</u>

- The result of the last method call

6.4. Class RGBLED

The **RGBLED** class represents instances of an RGB led that uses 3 GPIO pins to control the led. A specific constructor is provided that directly represents and controls access to the Omega Expansion Dock led.

6.4.1. RGBLED Constructors

6.4.1.1. Constructor - RGBLED();

Creates a new RGBLED instance specific for access to the expansion dock led.

Use of this constructor is equivalent to:

- **RGBLED(17, 16, 15);**

Parameters:

- <none>

6.4.1.2. Constructor - RGBLED(int redPin, int greenPin, int bluePin);

Creates a new RGBLED instance that uses the given pins.

Parameters:

- **int redPin** – the pin number for the red component of the led
- **int greenPin** – the pin number for the green component of the led
- **int bluePin** – the pin number for the blue component of the led

6.4.2. RGBLED Public Methods

6.4.2.1. public void setColor(int redVal, int greenVal, int blueVal);

Sets the colour of the led according to the parameters.

Parameters:

- **int redVal** – the value for the red component – in the range 0 (off) to 100 (fully on).
- **int greenVal** – the value for the green component – in the range 0 (off) to 100 (fully on).
- **int blueVal** – the value for the blue component – in the range 0 (off) to 100 (fully on).

Returns:

- <none>

6.4.2.2. public void setRed(int redVal);

Sets the red component of the led according to the parameter.

Parameters:

- **int redVal** – the value for the red component – in the range 0 (off) to 100 (fully on).

Returns:

- <none>

6.4.2.3. *public void setGreen(int greenVal);*

Sets the green component of the led according to the parameter.

Parameters:

- **int greenVal** – the value for the green component – in the range 0 (off) to 100 (fully on).

Returns:

- <none>

6.4.2.4. *public void setBlue(int blueVal);*

Sets the blue component of the led according to the parameter.

Parameters:

- **int blueVal** – the value for the blue component – in the range 0 (off) to 100 (fully on).

Returns:

- <none>

6.4.2.5. *public int getRed();*

Returns the setting of the red component of the led.

Parameters:

- <none>

Returns:

- The current value for the red component

6.4.2.6. *public int getGreen();*

Returns the setting of the green component of the led.

Parameters:

- <none>

Returns:

- The current value for the green component

6.4.2.7. *public int getBlue();*

Returns the setting of the blue component of the led.

Parameters:

- <none>

Returns:

- The current value for the blue component

6.4.2.8. public GPIOPin getRedPin();

Returns the GPIOPin used to control the red component of the led.

Parameters:

- <none>

Returns:

- the GPIOPin for the red component

6.4.2.9. public GPIOPin getGreenPin();

Returns the GPIOPin used to control the green component of the led.

Parameters:

- <none>

Returns:

- the GPIOPin for the green component

6.4.2.10. public GPIOPin getBluePin();

Returns the GPIOPin used to control the blue component of the led.

Parameters:

- <none>

Returns:

- the GPIOPin for the blue component

6.4.2.11. public void setActiveLow(boolean actLow);

Sets whether the led uses active low control (i.e. a low value is output to turn a component on) or active high control (i.e. a high value is output to turn a component on).

By default, activeLow is set to true as is used by the expansion dock led.

Parameters:

- **boolean actLow** – sets whether activeLow is enabled or not

Returns:

- <none>

6.4.2.12. public boolean isActiveLow();

Returns whether or not activeLow is set for the RGBLED.

Parameters:

- <none>

Returns:

- Whether or not activeLow is set

7. Usage of the java-gpio Program

The **java-gpio** program is used to perform a variety of operations on the GPIO pins.

The **java-gpio** program accepts a set of parameters to control its operation. As far as they are in common, the syntax of these parameters is the same as is used for the existing **fast-gpio** program.

The program will document its usage when the command **jamvm –jar java-gpio.jar help** is used.

In addition, the usage is shown whenever any errors are detected in the parameters.

The usage information displayed is:

```
Usage
Commands - one of:
        ./new-gpio set-input <pin>
                Sets pin to be an input pin
        ./new-gpio set-output <pin>
                Sets pin to be an output pin
        ./new-gpio get-direction <pin>
                Gets and returns pin direction
        ./new-gpio read <pin>
                Gets and returns input pin value
        ./new-gpio set <pin> <val>
                Sets output pin value
        ./new-gpio pwm <pin> <freq> <duty>
                Starts PWM output on pin
        ./new-gpio pwmstop <pin>
                Stops PWM output on pin
        ./new-gpio irq <pin> <irqtype> <irqcmd> <debounce>
                Enables IRQ handling on pin
        ./new-gpio irqstop <pin>
                Terminates IRQ handling on pin
        ./new-gpio expled <ledhex>
                Starts output to expansion led
        ./new-gpio expled rgb <r> <g> <b>
                Starts output to expansion led using decimal rgb values
        ./new-gpio expledstop
                Terminates output to expansion led
        ./new-gpio info <pin>
                Displays information on pin(s)
        ./new-gpio help
                Displays this help information
```

```
Where:
        <pin> is one of
                0, 1, 6, 7, 8, 12, 13, 14, 15, 16, 17, 18, 19, 23, 26, all
                A <pin> of all can only be used for:
                        info, set-input, set-output, set
        <val> is only required for set:
                <val> is 0 or 1
        <freq> is PWM frequency in Hz > 0
        <duty> is PWM duty cycle % in range 0 to 100
        <irqtype> is the type for IRQ and is one of:
                falling, rising, both
        <irqcmd> is the shell command to be executed when the IRQ occurs
                Must be enclosed in " characters if it contains
                spaces or other special characters
                If it starts with the string [debug],
                debug output is displayed first
        <debounce> is optional debounce time for IRQ in milliseconds
                Defaults to 0 if not supplied
        <ledhex> specifies the hex value to be output to expansion led
                Must be a six digit hex value with or without leading 0x
                The order of the hex digits is: rrggbb
        <r> <g> <b> specify the decimal values for output to expansion led
                Each value is in the range 0..100
                0 = off, 100 = fully on
```

Notes:

1. The return value from the command will be one of the following:
   - **255 (-1)** – indicates an error has occurred – either in the parameters or in executing the command
   - **0** – indicates normal successfully completion for an operation (**<op>**) other than **get** or **getd**
   - For a successful **get** operation:
     - **0** – indicates the pin is **off**
     - **1** – indicates the pin is **on**
   - For a successful **getd** operation:
     - **0** – indicates the pin is an **input** pin
     - **1** – indicates the pin is an **output** pin

2. When the **pwm** operation is used, the program forks a separate process to perform the PWM output.
   This separate process continues after the program returns until such time as the **pwmstop** operation is performed on the same pin.
   The ID of the separate process can be discovered by running:
   jamvm –jar java-gpio.jar info <pin-number>

3. When the **irq** operation is used, the program forks a separate process to monitor and respond to pin state changes.
   Each time the relevant pin undergoes the relevant change in state, the **<irqcmd>** command specified is run.
   This separate process continues after the program returns until such time as the **irqstop** operation is performed on the same pin.
   The ID of the separate process can be discovered by running:

jamvm –jar java-gpio.jar info <pin-number>

4. When the **expled** operation is used, the program forks a separate process to perform the expansion led output.
   This separate process continues after the program returns until such time as the **expledstop** operation is performed.
   The ID of the separate process can be discovered by running:
   jamvm –jar java-gpio.jar info <pin-number>
   Where <pin-number> is one of the expansion led pins: **15, 16, 17** or **all**

8. Usage of the java-expled Program

The **java-expled** program is used to control the led on the expansion dock.

The **java-expled** program accepts a parameter to set the colour of the led. The **java_expled** program provides exactly the same functionality as the existing **expled** script except that it is written in Java and uses **java-gpio-lib**.

The program will document its usage when the command **jamvm –jar java-expled.jar help** is used.

In addition, the usage is shown whenever any errors are detected in the parameters.

The usage information displayed is:

```
Usage
Commands - one of:
        ./new-expled <ledhex>
                Starts output to expansion led
        ./new-expled rgb <r> <g> <b>
                Starts output to expansion led using decimal rgb values
        ./new-expled stop
                Terminates output to expansion led
        ./new-expled help
                Displays this usage information
Where:
        <ledhex> specifies the hex value to be output to expansion led
                Must be a six digit hex value with or without leading 0x
                The order of the hex digits is: rrggbb
        <r> <g> <b> specify the decimal values for output to expansion led
                Each value is in the range 0..100
                0 = off, 100 = fully on
```

Note:

1. When the **java-expled** is run to set the led, the program forks a separate process to perform the expansion led output.
   This separate process continues after the program returns until such time as the **jamvm –jar java-expled.jar stop** command is run.
   The ID of the separate process can be discovered by running the **java-gpio** command:
   jamvm –jar java-gpio.jar info <pin-number>
   Where <pin-number> is one of the expansion led pins: **15, 16, 17** or **all**

2. In relation to **java-gpio**, the following two commands are directly equivalent:
    - **jamvm –jar java-expled.jar <ledhex>**
    - **jamvm –jar java-gpio expled <ledhex>**

    As are these two:

    - **jamvm –jar java-expled.jar rgb <r> <g> <b>**
    - **jamvm –jar java-gpio expled rgb <r> <g> <b>**

    And these two:

    - **jamvm –jar java-expled.jar stop**
    - **jamvm –jar java-gpio expledstop**

9. Further Development

Development of **java-gpio** is on-going. There will be changes and additions to the code in the future.

9.1. For the Future

In addition, it is intended that further work be done in the future based on **java-gpio**. In particular:

- Similar code for **i2c** access