new-gpio

Version 1.4.0 – 17 May 2016
Kit Bishop

Document History		
Version	**Date**	**Change Details**
Version 1.0	26 November 2015	Initial version
Version 1.1	5 December 2015	Added PWM control methods in GPIOAccess and GPIOPin classes in libnew-gpio.so library Added PWM control in new-gpiotest program Some reorganisation of contents of this document
Version 1.2	18 January 2016	Added IRQ functionality to GPIOAccess and GPIOPin classes in libnew-gpio.so library Added control of IQ in new-gpiotest program Changes in method of obtaining result of method calls Additional minor changes to parameters to new_gpiotest program Add section on pre-requisites for IRQ usage
Version 1.3	31 January 2016	Some re-organisation of packaged components and component renaming The new-gpiotest program is now just named new-gpio Provided Makefile files for all components Added both static and dynamic link versions of all components Added new class, RGBLED, for control of RGB leds (e.g. as in expansion led) Changed syntax of parameters to new-gpio program to be the same (where relevant) as is used for the existing fast-gpio program Added additional operations to new-gpio program for control of expansion led Added new program, new-expled, that does the same as the existing expled script but written in C++ using libnew-gpio
Version 1.3.1	1 February 2016	Minor correction to packaged Makefile for libnew-gpio
Version 1.3.2	5 February 2016	Some reorganisation of sources and improvements to Makefiles **NO** functional changes to code
Version 1.3.3	8 February 2016	Extended **new-gpio** and **new-expled** programs to accept decimal rgb values for expansion led setting
Version 1.3.4	14 February 2016	Some small improvements to Makefiles Added sources for a template gpio program **NO** functional changes to code
Version 1.3.5	19 February 2016	Fixed code bug in expansion led access

Version 1.4.0	17 May 2016	Fixed code bug that caused Ctrl-C to not be caught Code improvements to reduce memory usage to some extent. Major extensions to **libnew-gpio** library to provide functions inspired by Arduino pin control functionality. Specifically: • functions to generate and control tone output • 8 bit shift in and shift out functions • pulse out function • pulse in functions • function to return frequency on a pin Substantial re-write of **new-gpio** program to accommodate the new **libnew-gpio** functions and to provide a scripting facility with: • flow control • user defined variables and assignable expressions • rudimentary file access
	18 May 2016	Added example of use of script file for **new-gpio** program

Contents

1. Background

new-gpio is alternative C++ code for accessing the Omega GPIO pins.

The rationale for producing this code was two-fold:

- A desire for GPIO access with different features and capability than **fast-gpio**
- An exercise in developing C++ code for the Omega

new-gpio consists of three main components:

- **libnew-gpio** – a library containing the classes used to interact with GPIO pins
- **new-gpio** – a program for interacting with GPIO pins using **libnew-gpio** – this can be considered equivalent to the Omega supplied **fast-gpio** program but with extensions
- **new-expled** – a simple test program for controlling the expansion dock led using **libnew-gpio** – this can be considered equivalent to the Omega supplied **expled** script but with extensions

In all cases, each component is supplied using dynamic link and static linking and all sources, make files and build products are supplied.

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 OpenWrt toolchain for building the code:

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

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

NOTE: Some of the code in the class **GPIOAccess** as described below was derived from code in **fast-gpio**

2. Pre-requisites

To use theGPIO interrupt handling facilities in **new-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
```

3. Files Supplied

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

- **new-gpio.pdf** – this documentation as a PDF file
- **source** – directory containing all source files and make files:
  - **libnew-gpio** – directory containing all sources and Makefile for **libnew-gpio** library
    - **Makefile** – the Makefile for **libnew-gpio** library
    - **hdr** – directory containing header (***.h**) files for **libnew-gpio** library
    - **src** – directory containing source (***.cpp**) files for **libnew-gpio** library
  - **new-gpio** – directory containing all sources and Makefile for **new-gpio** program
    - **Makefile** – the Makefile for **new-gpio** program
    - **hdr** – directory to containing header (***.h**) files for **new-gpio** program
    - **src** – directory containing source (***.cpp**) files for **new-gpio** program

    - **new-expled** – directory containing all sources and Makefile for **new-expled** program
        - **Makefile** – the Makefile for **new-expled** program
        - **src** – directory containing source (***.cpp**) files for **new-expled** program
    - **gpio-template** – directory containing template sources code that can be used as a basis for a new program that uses **libnew-gpio**
        - **Makefile** – the Makefile for the template – will need modifying as per information in the first few lines of the file
        - **hdr** – directory to contain any header (***.h**) files for the template program
        - **src** – directory to contain all source (***.cpp**) files for the template program. Initially contains a very basic skeleton **main.cpp** source file
- **bin** – directory containing pre-built binary files:
    - **libnew-gpio** – directory containing the compiled **libnew-gpio** library files:
        - **libnew-gpio.a** – the static link library for **libnew-gpio**
        - **libnew-gpio.so** – the dynamic link library for **libnew-gpio**
    - **new-gpio** – directory containing the built **new-gpio** program files
        - **static-linked** – directory containing the **new-gpio** program file statically linked to **libnew-gpio.a**
        - **dynamic-linked** – directory containing the **new-gpio** program file dynamically linked to **libnew-gpio.so**
    - **new-expled** – directory containing the built **new-expled** program files
        - **static-linked** – directory containing the **new-expled** program file statically linked to **libnew-expled.a**
        - **dynamic-linked** – directory containing the **new-expled** program file dynamically linked to **libnew-expled.so**

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 libnew-gpio.a static library

To use **libnew-gpio.a** static library you simply need to statically link your program to that library file.

4.2. Using and Installing libnew-gpio.so

To use **libnew-gpio.so** dynamic library you need to dynamically link your program to that library file.

For any program that uses **libnew-gpio.so** the library file needs to be copied 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 new-gpio and new-expled Programs

If you want to use the statically linked version of these programs, simply copy the relevant program file from the **static-linked** directory as above to any suitable directory on your Omega from which you wish to run it.

If you want to use the dynamically linked version of these programs, simply copy the relevant program file from the **dynamic-linked** directory as above to any suitable directory on your Omega from which you wish to run it. You will also need to install the **libnew-gpio.so** library file as described above.

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** will need modifying in one or two ways:

- You **NEED** to and **MUST** change **TOOL_BIN_DIR** to the "bin" directory of your OpenWrt uClibc toolchain. E.G. make appropriate change to **<xxxx>** in:

  TOOL_BIN_DIR=<xxxx>/OpenWrt-Toolchain-ar71xx-generic_gcc-4.8-linaro_uClibc-0.9.33.2.Linux-x86_64/toolchain-mips_34kc_gcc-4.8-linaro_uClibc-0.9.33.2/bin

- You **MAY** need to change **LIBNEW-GPIO_DIR** to relative directory of libnew-gpio if you are not using the sources as originally supplied.
  This is relevant only to the **new-gpio** and **new-expled** programs.
  The default if using the standard **source** directory structure as supplied is:

  LIBNEW-GPIO_DIR=../libnew-gpio

5.2. Makefile targets

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

- **make**
  The default target. Performs a complete build of both static and dynamic link versions of component.
  This is directly equivalent to:
  make static dynamic
- **make static**
  Performs a complete build of just the static link version of component.

- **make dynamic**
  Performs a complete build of just the dynamic link version of component.

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

- **make clean-static**
  Removes all previous build files for static link versions only
  .
- **make clean-dynamic**
  Removes all previous build files for dynamic link versions only

For the **Makefile** for **new-gpio** and **gpio-expled**, if the following is added to the **make** command line:

builddep=1

then **libnew-gpio** that these programs depend on will also be built before building the programs.

5.3. Makefile build distribution products

In all cases, the build distribution products will be in the **.dist** directory or sub-directories thereof after **make** has been run.

6. Description of the libnew-gpio Library

The **libnew-gpio** library contains six main components for access and usage of **new-gpio** and some components that have internal usage only. These main components and their source files are:

- **GPIOTypes** – defines a few basic types used elsewhere
  File:
  GPIOTypes.h
- **GPIOAccess** – a class used for direct access to the Omega GPIO hardware.
  Contains only **static** methods for access.
  Files:
  GPIOAccess.h
  GPIOAccess.cpp
- **GPIOPin** – a class used to represent instances of a GPIO pin.
  Contains methods to interact with the specific pin.
  Files:
  GPIOPin.h
  GPIOPin.cpp
- **GPIOShiftIn** – a class used to represent instances of a GPIO shift input on 2 pins.
  Contains methods to perform shift input.
  Files:
  GPIOShiftIn.h
  GPIOShiftIn.cpp
- **GPIOShiftOut** – a class used to represent instances of a GPIO shift output on 2 pins.
  Contains methods to perform shift output.
  Files:
  GPIOShiftOut.h
  GPIOShiftOut.cpp
- **RGBLED** – a class used to represent instances of an RGB led (such as the led on the expansion dock).
  Contains methods to interact with the RGB led.
  Files:
  RGBLED.h
  RGBLED.cpp

The contents of these main components are described in following sections.

6.1. GPIOTypes

The file **GPIOTypes.h** contains definitions of some basic types used elsewhere.

6.1.1. enum GPIO_Result

enum GPIO_Result 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 GPIO_Direction

enum GPIO_Direction 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 GPIO_Bit_Order

enum GPIO_Bit_Order is used to represent the bit ordering to be used for shift in and shift out functions. It has values:

- **GPIO_MSB_FIRST = 0** – the most significant bit is first
- **GPIO_LSB_FIRST = 1** – the least significant bit is first

6.1.4. enum GPIO_Irq_Type

enum GPIO_Irq_Type 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

6.1.5. typedef void (*GPIO_Irq_Handler_Func) (int pinNum, GPIO_Irq_Type type);

GPIO_Irq_Handler_Func represents the type of the function to be specified for handling of an interrupt. Any such function passed for handling of an interrupt will be called when the interrupt occurs.

While the parameters passed are not strictly speaking required for interrupt handling in general, they are provided to allow the same handler 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.

Parameters:

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

Returns:

- <none>

6.1.6. GPIO_Irq_Handler_Object

GPIO_Irq_Handler_Object is a pure virtual abstract class that can be used as the base class of an object used to handle an interrupt as an alternative to using a **GPIO_Irq_Handler_Func**.

The form of the class is:

```
class GPIO_Irq_Handler_Object {
public:
    virtual void handleIrq(int pinNum, GPIO_Irq_Type type) = 0;
};
```

Any object actually used as an instance of **GPIO_Irq_Handler_Object** must inherit from this class and provide a non-abstract method for **handleIrq**. When an instance of any such class is used to handle an interrupt, the **handleIrq** method of the object is called to handle the interrupt. The **handleIrq** method has the following characteristics:

Parameters:

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

Returns:

- <none>

6.1.7. typedef void (*GPIO_PulseIn_Handler_Func) (int pinNum, long int len);

GPIO_PulseIn_Handler_Func represents the type of the function to be specified for handling of completion of a pulse in operation. Any such function passed for handling of a pulse in will be called when the pulse in completes.

While the pinNum parameter passed is not strictly speaking required for pulse in handling in general, it is provided to allow the same handler to be used for multiple pins and pulse in operations. Any actual handler can then (if required) control its action depending upon the pin. If no such distinction is required, this parameter can be ignored in the actual implementation of a passed handler.

Parameters:

- **int pinNum** – the number of the pin for which the handler is being called
- **long int len** – the length of the pulse input in microseconds

Returns:

- <none>

6.1.8. GPIO_PulseIn_Handler_Object

GPIO_PulseIn_Handler_Object is a pure virtual abstract class that can be used as the base class of an object used to handle completion of a pulse in operation as an alternative to using a **GPIO_PulseIn_Handler_Func**.

The form of the class is:

```
class GPIO_PulseIn_Handler_Object {
public:
    virtual void handlePulseIn(int pinNum, long int len) = 0;
};
```

Any object actually used as an instance of **GPIO_PulseIn_Handler_Object** must inherit from this class and provide a non-abstract method for **handlePulseIn**. When an instance of any such class is used to handle pulse in completion, the **handlePulseIn** method of the object is called to handle the pulse input. The **handlePulseIn** method has the following characteristics:

Parameters:

- **int pinNum** – the number of the pin for which the handler is being called
- **long int len** – the length of the pulse input in microseconds

Returns:

<none>

6.2. Class GPIOAccess

The **GPIOAccess** class is the main method 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** immediately after the call to the particular method.

6.2.1.1. static void setDirection(int pinNum, GPIO_Direction dir);

Sets the direction for a pin.

Parameters:

- **int pinNum** – the number of the pin
- **GPIO_Direction dir** – the direction to set the pin to

Returns:

- <none>

6.2.1.2. static GPIO_Direction 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 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 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 void setPWM(int pinNum, int freq, int duty, int durationMs = 0);

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

NOTE: PWM output is performed by software toggling of the state of the pin. Consequently, the frequency and duty cycle of the output is only reasonably accurate up to a few hundred Hz.

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
- the duration expires

Parameters:

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

- **int freq** – sets the PWM frequency in Hz
- **int duty** – sets the PWM duty cycle percentage
- **int durationMs** – sets the duration in milliseconds for which the PWM output is performed. A value of zero give unterminated output. The default value is zero

Returns:

- <none>

6.2.1.6. static void startPWM(int pinNum, int durationMs = 0);

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
- **int durationMs** – sets the duration in milliseconds for which the PWM output is performed. A value of zero give unterminated output. The default value is zero

Returns:

- <none>

6.2.1.7. static 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 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 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 int getPWMDuration(int pinNum);

Returns the currently set PWM duration for a pin

Parameters:

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

Returns:

- The PWM duration

6.2.1.11. static bool 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

6.2.1.12. static void setIrq(int pinNum, GPIO_Irq_Type type, GPIO_Irq_Handler_Func handler, long int debounceMs = 0);

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

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 function 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
- **GPIO_Irq_Type type** – specifies the interrupt type to apply
- **GPIO_Irq_Handler_Func handler** – a pointer to the function to be called to handle the interrupt
- **Long int debounceMs = 0** – specifies an optional debounce period in milliseconds to be applied.
  Default value is **0**
  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>

*6.2.1.13. static void setIrq(int pinNum, GPIO_Irq_Type type, GPIO_Irq_Handler_Object * handlerObj, long int debounceMs = 0);*

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
- **GPIO_Irq_Type type** – specifies the interrupt type to apply
- **GPIO_Irq_Handler_Object handlerObj** – a pointer to the handler object to be used to handle the interrupt
- **long int debounceMs = 0** – specifies an optional debounce period in milliseconds to be applied.
  Default value is **0**
  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>

6.2.1.14. static void resetIrq(int pinNum);

Removes any interrupt handling for a pin.

Parameters:

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

Returns:

- <none>

6.2.1.15. static 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>

6.2.1.16. static 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>

6.2.1.17. static void enableIrq(int pinNum, bool enable);

Enables or Disables interrupt handling for a pin according to parameter.

Parameters:

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

Returns:

- <none>

6.2.1.18. static bool 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

6.2.1.19. static GPIO_Irq_Type getIrqType(int pinNum);

Returns the current interrupt type for a pin.

Parameters:

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

Returns:

- the interrupt type

6.2.1.20. static GPIO_Irq_Handler_Func getIrqHandler(int pinNum);

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

Parameters:

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

Returns:

- the interrupt handler function

*6.2.1.21. static GPIO_Irq_Handler_Object * getIrqHandlerObj(int pinNum);*

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

Parameters:

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

Returns:

- pointer to the interrupt handler object

6.2.1.22. static void enableIrq();

Enables interrupt handling for all pins with interrupt handling set up.

Parameters:

- <none>

Returns:

- <none>

6.2.1.23. static void disableIrq();

Disables interrupt handling for all pins with interrupt handling set up.

Parameters:

- <none>

Returns:

- <none>

6.2.1.24. static void enableIrq(bool enable);

Enables or disables interrupt handling for all pins with interrupt handling set up according to parameter.

Parameters:

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

Returns:

- <none>

6.2.1.25. static bool irqEnabled();

Returns an indication as to whether interrupt handling is currently enabled or disabled for all pins.

Parameters:

- <none>

Returns:

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

6.2.1.26. static void setTone(int pinNum, int freq, int durationMs = 0);

Starts the tone output on a pin with the given frequency. Tone output is equivalent to PWM output with a 50% duty cycle.

NOTE: Tone output is performed by software toggling of the state of the pin. Consequently, the frequency of the output is only reasonably accurate up to a few hundred Hz.

NOTE: Tone 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 **stopTone** method is called for the pin
- the process that started the thread (i.e. made the call to this method) terminates
- the duration expires

Parameters:

- **int pinNum** – the number of the pin
- **int freq** – sets the Tone frequency in Hz
- **int durationMs** – sets the duration in milliseconds for which the Tone output is performed. A value of zero give unterminated output. The default value is zero

Returns:

- <none>

6.2.1.27. static void startTone(int pinNum, int durationMs = 0);

Starts the Tone output on a pin using the last used frequency value for the pin.

Parameters:

- **int pinNum** – the number of the pin
- **int durationMs** – sets the duration in milliseconds for which the Tone output is performed. A value of zero give unterminated output. The default value is zero

Returns:

- <none>

6.2.1.28. static void stopTone(int pinNum);

Stops any current Tone output on a pin.

Parameters:

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

Returns:

- <none>

6.2.1.29. static int getToneFreq(int pinNum);

Returns the currently set Tone frequency for a pin.

Parameters:

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

Returns:

- The Tone frequency in Hz

6.2.1.30. static int getToneDuration(int pinNum);

Returns the currently set Tone duration for a pin

Parameters:

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

Returns:

- The Tone duration

6.2.1.31. static bool isToneRunning(int pinNum);

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

Parameters:

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

Returns:

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

6.2.1.32. static void shiftOut(int dataPinNum, int clockPinNum, int val, long int clockPeriodNS, GPIO_Bit_Order bitOrder = GPIO_MSB_FIRST);

Performs an 8 bit shift out operation using indicated data and clock pins. Intended to send an 8 bit value serially to (e.g.) a shift register.

Parameters:

- **int dataPinNum** – the number of the pin for the data
- **int clockPinNum** – the number of the pin for the clock
- **int val** – the value to be sent
- **long int clockPeriodNS** – the duration of each clock cycle in nano-seconds
- **GPIO_BIT_ORDER bitOrder** – the order that the bits are transferred. One of:
    - **GPIO_MSB_FIRST** for most significant bit first
    - **GPIO_LSB_FIRST** for least significant bit first

  Defaults to **GPIO_MSB_FIRST**

Returns:

- <none>

6.2.1.33. static int shiftIn(int dataPinNum, int clockPinNum, long int clockPeriodNS, GPIO_Bit_Order bitOrder = GPIO_MSB_FIRST);

Performs an 8 bit shift in operation using indicated data and clock pins. Intended to receive an 8 bit value serially from (e.g.) a shift register.

Parameters:

- **int dataPinNum** – the number of the pin for the data
- **int clockPinNum** – the number of the pin for the clock
- **long int clockPeriodNS** – the duration of each clock cycle in nano-seconds
- **GPIO_BIT_ORDER bitOrder** – the order that the bits are transferred. One of:
    - **GPIO_MSB_FIRST** for most significant bit first
    - **GPIO_LSB_FIRST** for least significant bit first

  Defaults to **GPIO_MSB_FIRST**

Returns:

- The value received

6.2.1.34. static void pulseOut(int pinNum, long int pulseLenUS, int pulseLevel = 1);

Sends a single pulse on the given pin.

Parameters:

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

- **long int pulseLenUS** – the length of the pulse in micro-seconds
- **int pulseLevel** – the polarity of the pulse: 1 or 0. Defaults to 1

Returns:

- <none>

6.2.1.35. static long int pulseIn(int pinNum, int pulseLevel = 1, long int timeoutUS = 0);

Awaits and returns the length of an input pulse on the given pin. Returns when the pulse is received or any time out has expired – whichever comes first.

Parameters:

- **int pinNum** – the number of the pin
- **int pulseLevel** – the polarity of the pulse: 1 or 0. Defaults to 1
- **long int timeoutUS** – the time in micro-seconds to wait for the pulse. If zero, waits indefinitely. Defaults to 0

Returns:

- The length of the pulse in micro-seconds

6.2.1.36. static void pulseIn(int pinNum, GPIO_PulseIn_Handler_Func handler , int pulseLevel = 1, long int timeoutUS = 0);

Starts a pulse in operation and returns immediately. The **GPIO_PulseIn_Handler_Func** will be called when the input pulse is completed.

NOTE: pulse input is run on a separate thread for the pin. When this method is called the thread will be started and the call to the method then returns. The thread continues to run until one of the following occurs:

- the pulse in is completed
- the **stopPulseIn** method is called for the pin
- the process that started the thread (i.e. made the call to this method) terminates
- the timeout expires

Parameters:

- **int pinNum** – the number of the pin
- **GPIO_PulseIn_Handler_Func handler** – the handler function to be called on completion of the pulse input
- **int pulseLevel** – the polarity of the pulse: 1 or 0. Defaults to 1
- **long int timeoutUS** – the time in micro-seconds to wait for the pulse. If zero, waits indefinitely. Defaults to 0

Returns:

- <none>

6.2.1.37. static void pulseIn(int pinNum, GPIO_PulseIn_Handler_Object * handlerObj , int pulseLevel = 1, long int timeoutUS = 0);

Starts a pulse in operation and returns immediately. The **GPIO_PulseIn_Handler_Object** will be called when the input pulse is completed.

NOTE: pulse input is run on a separate thread for the pin. When this method is called the thread will be started and the call to the method then returns. The thread continues to run until one of the following occurs:

- the pulse in is completed
- the **stopPulseIn** method is called for the pin
- the process that started the thread (i.e. made the call to this method) terminates
- the timeout expires

Parameters:

- **int pinNum** – the number of the pin
- **GPIO_PulseIn_Handler_Object * handlerObj** – the handler object to be called on completion of the pulse input
- **int pulseLevel** – the polarity of the pulse: 1 or 0. Defaults to 1
- **long int timeoutUS** – the time in micro-seconds to wait for the pulse. If zero, waits indefinitely. Defaults to 0

Returns:

- <none>

6.2.1.38. static void stopPulseIn(int pinNum);

Stops any current pulse input on a pin.

Parameters:

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

Returns:

- <none>

6.2.1.39. static bool isPulseInRunning(int pinNum);

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

Parameters:

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

Returns:

- **true** if pulse in is running; **false** if pulse is not running

6.2.1.40. static long int getFrequency(int pinNum, long int sampleTimeMS);

Attempts to return the input frequency of a signal on the given pin.

NOTE: This method works by measuring the average period of signal changes on the input pin during the given sample time. It does so by using interrupts on the pin and measuring the time between successive interrupts. Consequently:

- The actual value may be influenced by any other processes running on the system
- The accuracy is likely to be improved by a greater sample time at the expense of greater run time
- The method is only reliable up to a few KHz input frequency

Parameters:

- **int pinNum** – the number of the pin
- **long int sampleTimeMS** – the time in milli-seconds over which the input frequency is sampled

Returns:

- the detected input frequency in Hz

6.2.1.41. static bool isPinUsable(int pinNum);

Returns an indication as to whether or not a specific pin number can be used for a GPIO pin.

Parameters:

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

Returns:

- **true** or **false –** indicating whether or not **pinNum** is a valid GPIO pin

6.2.1.42. static bool 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

6.2.1.43. static GPIO_Result 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 and Destructor

6.3.1.1. Constructor - GPIOPin(int pinNum);

Creates a new GPIOPin instance for a given pin.

Parameters:

- **int pinNum** – the pin number

6.3.1.2. Destructor - ~GPIOPin(void);

Destroys an instance of a GPIOPin.

NOTE: This also ensures that any threads for the pin are terminated.

Parameters:

- <none>

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. <void> setDirection(GPIO_Direction dir);

Sets the direction of the GPIOPin.

Parameters:

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

Returns:

- <none>

6.3.2.2. GPIO_Result getDirection();

Obtains the current direction of the GPIOPin

Parameters:

- <none>

Returns:

- The current direction of the pin

6.3.2.3. 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. int get();

Directly returns the value of the GPIOPin.

Parameters:

- <none>

Returns:

- the current value of the pin

6.3.2.5. void setPWM(int freq, int duty, int durationMs = 0);

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

NOTE: PWM output is performed by software toggling of the state of the pin. Consequently, the frequency and duty cycle of the output is only reasonably accurate up to a few hundred Hz.

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
- the duration expires

Parameters:

- **int freq** – sets the PWM frequency in Hz
- **int duty** – sets the PWM duty cycle percentage
- **int durationMS** – the duration in milliseconds to keep the PWM running. A value of zero gives unterminated output. The default is zero

Returns:

- <none>

6.3.2.6. void startPWM(int durationMs = 0);

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

Parameters:

- **int durationMS** – the duration in milliseconds to keep the PWM running. A value of zero gives unterminated output. The default is zero

Returns:

- <none>

6.3.2.7. void stopPWM();

Stops any current PWM output on the GPIOPin

Parameters:

- <none>

Returns:

- <none>

6.3.2.8. int getPWMFreq();

Returns the currently set PWM frequency for the GPIOPin

Parameters:

- <none>

Returns:

- The PWM frequency in Hz

6.3.2.9. 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. int getPWMDuration();

Returns the currently set PWM duration for the GPIOPin

Parameters:

- <none>

Returns:

- The PWM duration

6.3.2.11. bool 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

6.3.2.12. void setIrq(GPIO_Irq_Type type,GPIO_Irq_Handler_Func handler,long int debounceMs=0);

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

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 function 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:

- **GPIO_Irq_Type type** – specifies the interrupt type to apply
- **GPIO_Irq_Handler_Func handler** – a pointer to the function to be called to handle the interrupt
- **Long int debounceMs = 0** – specifies an optional debounce period in milliseconds to be applied.
  Default value is **0**
  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. void setIrq(GPIO_Irq_Type type, GPIO_Irq_Handler_Object * handlerObj, long int debounceMs = 0);

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:

- **GPIO_Irq_Type type** – specifies the interrupt type to apply
- **GPIO_Irq_Handler_Object handlerObj** – a pointer to the handler object to be used to handle the interrupt
- **long int debounceMs = 0** – specifies an optional debounce period in milliseconds to be applied.
  Default value is **0**
  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>

6.3.2.14. void resetIrq();

Removes any interrupt handling for the GPIOPin.

Parameters:

- <none>

Returns:

- <none>

6.3.2.15. void enableIrq();

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

Parameters:

- <none>

Returns:

- <none>

6.3.2.16. void disableIrq();

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

Parameters:

- <none>

Returns:

- <none>

6.3.2.17. void enableIrq(bool enable);

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

Parameters:

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

Returns:

- <none>

6.3.2.18. bool 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

6.3.2.19. GPIO_Irq_Type getIrqType();

Returns the current interrupt type for the GPIOPin.

Parameters:

- <none>

Returns:

- the interrupt type

6.3.2.20. GPIO_Irq_Handler_Func getIrqHandler();

Returns the any currently established interrupt handler function for the GPIOPin.

Parameters:

- <none>

Returns:

- the interrupt handler function

*6.3.2.21. GPIO_Irq_Handler_Object * getIrqHandlerObj();*

Returns the any currently established interrupt handler object for the GPIOPin.

Parameters:

- <none>

Returns:

- pointer to the interrupt handler object

6.3.2.22. void setTone(int freq, int durationMs = 0);

Starts the tone output on the pin with the given frequency. Tone output is equivalent to PWM output with a 50% duty cycle.

NOTE: Tone output is performed by software toggling of the state of the pin. Consequently, the frequency of the output is only reasonably accurate up to a few hundred Hz.

NOTE: Tone 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 **stopTone** method is called for the pin
- the process that started the thread (i.e. made the call to this method) terminates
- the duration expires

Parameters:

- **int freq** – sets the Tone frequency in Hz
- **int durationMs** – sets the duration in milliseconds for which the Tone output is performed. A value of zero give unterminated output. The default value is zero

Returns:

- <none>

6.3.2.23. void startTone(int durationMs = 0);

Starts the Tone output on the pin using the last used frequency value for the pin.

Parameters:

- **int durationMs** – sets the duration in milliseconds for which the Tone output is performed. A value of zero give unterminated output. The default value is zero

Returns:

- <none>

6.3.2.24. void stopTone();

Stops any current Tone output on the pin.

Parameters:

- <none>

Returns:

- <none>

6.3.2.25. int getToneFreq();

Returns the currently set Tone frequency for the pin.

Parameters:

- <none>

Returns:

- The Tone frequency in Hz

6.3.2.26. int getToneDuration();

Returns the currently set Tone duration for the pin

Parameters:

- <none>

Returns:

- The Tone duration

6.3.2.27. bool isToneRunning();

Returns an indication of whether or not Tone is currently running on the pin

Parameters:

- <none>

Returns:

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

6.3.2.28. void pulseOut(long int pulseLenUS, int pulseLevel = 1);

Sends a single pulse on the pin.

Parameters:

- **long int pulseLenUS** – the length of the pulse in micro-seconds
- **int pulseLevel** – the polarity of the pulse: 1 or 0. Defaults to 1

Returns:

- <none>

6.3.2.29. long int pulseIn(int pulseLevel = 1, long int timeoutUS = 0);

Awaits and returns the length of an input pulse on the pin. Returns when the pulse is received or any time out has expired – whichever comes first.

Parameters:

- **int pulseLevel** – the polarity of the pulse: 1 or 0. Defaults to 1
- **long int timeoutUS** – the time in micro-seconds to wait for the pulse. If zero, waits indefinitely. Defaults to 0

Returns:

- The length of the pulse in micro-seconds

6.3.2.30. void pulseIn(GPIO_PulseIn_Handler_Func handler , int pulseLevel = 1, long int timeoutUS = 0);

Starts a pulse in operation and returns immediately. The **GPIO_PulseIn_Handler_Func** will be called when the input pulse is completed.

NOTE: pulse input is run on a separate thread for the pin. When this method is called the thread will be started and the call to the method then returns. The thread continues to run until one of the following occurs:

- the pulse in is completed
- the **stopPulseIn** method is called for the pin
- the process that started the thread (i.e. made the call to this method) terminates
- the timeout expires

Parameters:

- **GPIO_PulseIn_Handler_Func handler** – the handler function to be called on completion of the pulse input
- **int pulseLevel** – the polarity of the pulse: 1 or 0. Defaults to 1
- **long int timeoutUS** – the time in micro-seconds to wait for the pulse. If zero, waits indefinitely. Defaults to 0

Returns:

- <none>

6.3.2.31. void pulseIn(GPIO_PulseIn_Handler_Object * handlerObj , int pulseLevel = 1, long int timeoutUS = 0);

Starts a pulse in operation and returns immediately. The **GPIO_PulseIn_Handler_Object** will be called when the input pulse is completed.

NOTE: pulse input is run on a separate thread for the pin. When this method is called the thread will be started and the call to the method then returns. The thread continues to run until one of the following occurs:

- the pulse in is completed
- the **stopPulseIn** method is called for the pin
- the process that started the thread (i.e. made the call to this method) terminates
- the timeout expires

Parameters:

- **GPIO_PulseIn_Handler_Object * handlerObj** – the handler object to be called on completion of the pulse input
- **int pulseLevel** – the polarity of the pulse: 1 or 0. Defaults to 1
- **long int timeoutUS** – the time in micro-seconds to wait for the pulse. If zero, waits indefinitely. Defaults to 0

Returns:

- <none>

6.3.2.32. void stopPulseIn();

Stops any current pulse input on the pin.

Parameters:

- <none>

Returns:

- <none>

6.3.2.33. bool isPulseInRunning();

Returns an indication of whether or not pulse input is currently running on the pin

Parameters:

- <none>

Returns:

- **true** if pulse in is running; **false** if pulse is not running

6.3.2.34. long int getFrequency(long int sampleTimeMS);

Attempts to return the input frequency of a signal on the pin.

NOTE: This method works by measuring the average period of signal changes on the input pin during the given sample time. It does so by using interrupts on the pin and measuring the time between successive interrupts. Consequently:

- The actual value may be influenced by any other processes running on the system
- The accuracy is likely to be improved by a greater sample time at the expense of greater run time
- The method is only reliable up to a few KHz input frequency

Parameters:

- **long int sampleTimeMS** – the time in milli-seconds over which the input frequency is sampled

Returns:

- the detected input frequency in Hz

6.3.2.35. int getPinNumber();

Returns the pin number for the GPIOPin

Parameters:

- <none>

Returns:

- The pin number

6.3.2.36. GPIO_Result getLastResult();

Returns the result of the latest call to other methods.

Parameters:

- <none>

Returns:

- The result of the last method call

6.4. Class GPIOShiftIn

The **GPIOShiftIn** class represents instances of a shift input using 2 GPIO pins – one for the shift data and one for the shift clock.

6.4.1. GPIOShiftIn Constructor and Destructor

6.4.1.1. Constructor - GPIOShiftIn(int dataPin, int clockPin);

Creates a new GPIOShiftIn instance specific that uses the specified pins.

Parameters:

- **int dataPin** – the pin number to be used for the data
- **int clockPin** – the pin number to be used for the clock

6.4.1.2. Destructor - ~GPIOShiftIn(void);

Destroys an instance of a GPIOShiftIn.

Parameters:

- <none>

6.4.2. GPIOShiftIn Public Methods

6.4.2.1. int read();

Reads the data on the data pin using the clock on the clock pin.

Parameters:

- <none>

Returns:

- The value read

6.4.2.2. void setClockPeriodNS(long int clockPerNS);

Sets the period of one clock cycle on the clock pin.

Parameters:

- **long int clockPerNS** – the clock period in nano-seconds

Returns:

- <none>

6.4.2.3. long int getClockPeriodNS();

Returns the period of one clock cycle on the clock pin.

Parameters:

- <none>

Returns:

- the clock period in nano-seconds

6.4.2.4. void setBitOrder(GPIO_Bit_Order bitOrd);

Sets the bit order to be used for the shift in.

Parameters:

- **GPIO_Bit_Order bitOrd** – the bit order to be used

Returns:

- <none>

6.4.2.5. GPIO_Bit_Order getBitOrder();

Returns the bit order used for the shift in.

Parameters:

- <none>

Returns:

- the bit order used

*6.4.2.6. GPIOPin * getDataPin();*

Returns a reference to the GPIOPin instance used for the data.

Parameters:

- <none>

Returns:

- A reference to the GPIOPin

*6.4.2.7. GPIOPin * getClockPin();*

Returns a reference to the GPIOPin instance used for the clock.

Parameters:

- <none>

Returns:

- A reference to the GPIOPin

6.5. Class GPIOShiftOut

The **GPIOShiftOut** class represents instances of a shift output using 2 GPIO pins – one for the shift data and one for the shift clock.

6.5.1. GPIOShiftOut Constructor and Destructor

6.5.1.1. Constructor - GPIOShiftOut(int dataPin, int clockPin);

Creates a new GPIOShiftOut instance specific that uses the specified pins.

Parameters:

- **int dataPin** – the pin number to be used for the data
- **int clockPin** – the pin number to be used for the clock

6.5.1.2. Destructor - ~GPIOShiftOut(void);

Destroys an instance of a GPIOShiftOut.

Parameters:

- <none>

6.5.2. GPIOShiftOut Public Methods

6.5.2.1. GPIO_Result write(int val);

Writes a value on the data pin using the clock on the clock pin.

Parameters:

- **int val** – the value to be written

Returns:

- The result of performing the operation

6.5.2.2. void setClockPeriodNS(long int clockPerNS);

Sets the period of one clock cycle on the clock pin.

Parameters:

- **long int clockPerNS** – the clock period in nano-seconds

Returns:

- <none>

6.5.2.3. long int getClockPeriodNS();

Returns the period of one clock cycle on the clock pin.

Parameters:

- <none>

Returns:

- the clock period in nano-seconds

6.5.2.4. void setBitOrder(GPIO_Bit_Order bitOrd);

Sets the bit order to be used for the shift out.

Parameters:

- **GPIO_Bit_Order bitOrd** – the bit order to be used

Returns:

- <none>

6.5.2.5. GPIO_Bit_Order getBitOrder();

Returns the bit order used for the shift out.

Parameters:

- <none>

Returns:

- the bit order used

*6.5.2.6. GPIOPin * getDataPin();*

Returns a reference to the GPIOPin instance used for the data.

Parameters:

- <none>

Returns:

- A reference to the GPIOPin

*6.5.2.7. GPIOPin * getClockPin();*

Returns a reference to the GPIOPin instance used for the clock.

Parameters:

- <none>

Returns:

- A reference to the GPIOPin

6.6. 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.

6.6.1. RGBLED Constructors and Destructor

6.6.1.1. 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>

6.6.1.2. 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

6.6.1.3. Destructor - ~RGBLED(void);

Destroys an instance of an RGBLED.

Parameters:

- <none>

6.6.2. RGBLED Public Methods

6.6.2.1. <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>

6.6.2.2. <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>

6.6.2.3. <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>

6.6.2.4. <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>

6.6.2.5. int getRed();

Returns the setting of the red component of the led.

Parameters:

- <none>

Returns:

- The current value for the red component

6.6.2.6. int getGreen();

Returns the setting of the green component of the led.

Parameters:

- <none>

Returns:

- The current value for the green component

6.6.2.7. int getBlue();

Returns the setting of the blue component of the led.

Parameters:

- <none>

Returns:

- The current value for the blue component

6.6.2.8. GPIOPin * getRedPin();

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

Parameters:

- <none>

Returns:

- Reference to the GPIOPin for the red component

6.6.2.9. GPIOPin * getGreenPin();

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

Parameters:

- <none>

Returns:

- Reference to the GPIOPin for the green component

6.6.2.10. GPIOPin * getBluePin();

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

Parameters:

- <none>

Returns:

- Reference to the GPIOPin for the blue component

6.6.2.11. void setActiveLow(bool 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:

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

Returns:

- <none>

6.6.2.12. bool isActiveLow();

Returns whether or not activeLow is set for the RGBLED.

Parameters:

- <none>

Returns:

- Whether or not activeLow is set

7. Usage of the new-gpio Program

7.1. Description of new-gpio Program

The **new-gpio** program is used to perform a variety of operations on the GPIO pins with scripting capabilities.

The **new-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.

Using the command **./new-gpio** without any parameters displays basic information on obtaining help, so:

```
**ERROR** No operations supplied

A C++ program to control and interact with GPIO pins via scripted operations
        Program version:1.4.0
        GPIO Library version:1.4.0
Help is available by using one of:
        ./new-gpio -?  - for basic usage
        ./new-gpio -h  - for full help
```

Using the command **./new-gpio -?** displays brief help on the use of the program and its parameters, so:

```
A C++ program to control and interact with GPIO pins via scripted operations
        Program version:1.4.0
        GPIO Library version:1.4.0
Basic Usage:
./new-gpio [any length sequence of spaces separated <input-element>s]
An <input-element> is one of: <option> <file-input> <operation>
An <option> is one of:
        -v - verbose output
```

```
        -q - quiet output
        -o - result output
        -r - report output
        -i - ignore errors
        -e - error output
        -s - automatic setting of pin direction
        -h - various levels of help output
        -? - this basic usage output
A <file-input> is a:
        @<file-name> - input commands from file
An <operation> is of the form:
        <operation-name> <operation parameters>
                An <operation-name> is one of:
                        info
                        set
                        read
                        set-input
                        set-output
                        get-direction
                        pwm
                        pwmstop
                        irq
                        irq2
                        irqstop
                        expled
                        expledstop
                        tone
                        tonestop
                        shiftout
                        shiftin
                        pulseout
                        pulsein
                        frequency
                        delay
                        exec
                        assign
                        filein
                        fileout
                        filedelete
                        while
                        endwhile
                        if
                        else
                        endif
                        break
                        exit
                The <operation-parameters> depend on the specific operation

More information can be displayed by using one of:
        -h or -h:all - for all help
        -h:-<option-letter> - for help on the option
        -h:@ - for help on file input
        -h:<operation-name> - for help on the operation and its parameters

Sources available at: https://github.com/KitBishop/new-gpio
```

Using the command **./new-gpio -h** displays full help on the use of the program and all its parameters, so:

```
A C++ program to control and interact with GPIO pins via scripted operations
        Program version:1.4.0
        GPIO Library version:1.4.0
Usage:
./new-gpio [any length sequence of spaces separated <input-element>s]
An <input-element> is one of: <option> <file-input> <operation>

An <option> is one of:
```

```
        -v and -v+ - sets verbose mode; equivalent to -o -r -e
        -v- - resets verbose mode; equivalent to -q
                Defaults are: -o+ -r- -e+
        -q and -q+ - sets quiet mode; equivalent to -o- -r- -e-
        -q- - resets quiet mode; equivalent to -v
        -o and -o+ - enables output to stdout of any results of operation
        -o- - disables output to stdout of any results of operation
                Default is: -o+
        -r and -r+ - enables output to stderr of report on actions taken
        -r- - disables output to stderr of report on actions taken
                Default is: -r-
        -e and -e+ - enables output to stderr of any errors during processing
        -e- - disables output to stderr of any errors during processing
                Default is: -e+
        -i and -i+ - enables ignoring of any errors during processing
        -i- - disables ignoring of any errors during processing
                Default is: -i-
        -s and -s+ - causes the program to ensure that the pin direction
                is set appropriately for each operation
        -s- - does not set the direction for each operation
                Default is: -s-
        -d - enables debugging output on entered data
                When used anywhere in the input causes debugging output
                to be displayed on scanned input and processed operation data
                By default, debugging output is disabled
        -h or -h:all - displays all available help
        -h:-<option-letter> - displays help for the given option letter
        -h:<operation> - displays help for the given operation
        -h:@ - displays help for input from file
        -? - displays basic usage help

A <file-input> is:
        @<file-name> - causes input to be taken from the given file
                Input is free form sequence of space separated standard
                        input elements: <option> <file-input> <operation>
                Any line with first non-blank character of # is ignored

An <operation> is one of:
        info <pin-number>
                Displays information on given <pin-number> or all pins
                <pin-number> must be one of:
                        0,1,6,7,8,12,13,14,15,16,17,18,19,23,26 or all
        set <pin-number> <value>
                Sets given <pin-number> or all pins to the given value
                <pin-number> must be one of:
                        0,1,6,7,8,12,13,14,15,16,17,18,19,23,26 or all
                <value> must be one of: 0 or 1
        read <pin-number>
                Reads, displays and returns value of given <pin-number>
                <pin-number> must be one of:
                        0,1,6,7,8,12,13,14,15,16,17,18,19,23,26
        set-input <pin-number>
                Sets given <pin-number> or all pins to be input pins
                <pin-number> must be one of:
                        0,1,6,7,8,12,13,14,15,16,17,18,19,23,26 or all
        set-output <pin-number>
                Sets given <pin-number> or all pins to be output pins
                <pin-number> must be one of:
                        0,1,6,7,8,12,13,14,15,16,17,18,19,23,26 or all
        get-direction <pin-number>
                Reads, displays and returns the current direction
                of given <pin-number>
                <pin-number> must be one of:
                        0,1,6,7,8,12,13,14,15,16,17,18,19,23,26
        pwm <pin-number> <frequency> <duty> <optional-duration>
                Starts a separate process to perform PWM output
                on given <pin-number> with given information
                <pin-number> must be one of:
                        0,1,6,7,8,12,13,14,15,16,17,18,19,23,26
```

```
                <frequency> is the frequency of the PWM pulses
                        Must be greater than 0
                <duty> is the duty cycle % of the PWM pulses
                        Must be in the range 0 to 100
                <optional-duration> is an optional parameter that gives the
                duration of the output in milliseconds
                        If present, must be greater than or equal to 0
                        When absent or 0, duration is indefinite
                The separate process runs until the <optional-duration> expires
                or the 'pwmstop' operation is performed on the same <pin-number>
        pwmstop <pin-number>
                Stops any separate process that is currently performing PWM
                output on given <pin-number>
                <pin-number> must be one of:
                        0,1,6,7,8,12,13,14,15,16,17,18,19,23,26
        irq <pin-number> <irq-type> <command> <optional-debounce>
                Starts a separate process to respond to interrupts
                on given <pin-number> and perform given <command> on interrupt
                <pin-number> must be one of:
                        0,1,6,7,8,12,13,14,15,16,17,18,19,23,26
                <irq-type> is the type of interrupt to catch
                Must be one of:
                        rising - interrupt occurs on rising edge
                        falling - interrupt occurs on falling edge
                        both - interrupt occurs on both rising and falling edges
                <command> is the command to be performed on interrupt
                Must be enclosed in " characters if it contains spaces
                or other special character
                The <command> may contain any number of <varsub>s which will be
                replaced by actual values at the time the irq operation is
                actually invoked.
                A <varsub> is any sequence like {<varref>}
                where '<varref>' is one of:
                  - <variable-name> = current value of variable
                    as assigned in an 'assign' operation
                    or not substituted if variable has never been assigned
                  - $n or $nn = current value of pin n or nn
                  - $! = value of latest result set by earlier commands
                  - $? = status of last executed command: 0=error, 1=ok
                  - $[<file-name>] = file <file-name> exists; 0=no, 1=yes
                <optional-debounce> is an optional parameter that gives a
                debounce time in milliseconds.  Any interrupts that occur within
                this time of a previous interrupt will be ignored.
                Used to cater for noisy mechanical signals
                If present, must be greater than or equal to 0
                When absent or 0, no debounce testing is applied
                The separate process runs until the 'irqstop' operation
                is performed on the same <pin-number>
        irq2 <pin-number> <rising-command> <falling-command> <optional-debounce>
                Starts a separate process to respond to interrupts
                on given <pin-number> and perform given commands on interrupt
                both for rising and falling edges
                <pin-number> must be one of:
                        0,1,6,7,8,12,13,14,15,16,17,18,19,23,26
                <rising-command> is the command to be performed on interrupt
                        rising edge.
                <falling-command> is the command to be performed on interrupt
                        falling edge.
                Each must be enclosed in " characters if it contains spaces
                or other special character
                Each command may contain any number of <varsub>s which will be
                replaced by actual values at the time the irq2 operation is
                actually invoked.
                A <varsub> is any sequence like {<varref>}
                where '<varref>' is one of:
                  - <variable-name> = current value of variable
                    as assigned in an 'assign' operation
                    or not substituted if variable has never been assigned
                  - $n or $nn = current value of pin n or nn
```

```
                - $! = value of latest result set by earlier commands
                - $? = status of last executed command: 0=error, 1=ok
                - $[<file-name>] = file <file-name> exists; 0=no, 1=yes
              <optional-debounce> is an optional parameter that gives a
              debounce time in milliseconds.  Any interrupts that occur within
              this time of a previous interrupt will be ignored.
              Used to cater for noisy mechanical signals
              If present, must be greater than or equal to 0
              When absent or 0, no debounce testing is applied
              The separate process runs until the 'irqstop' operation
              is performed on the same <pin-number>
      irqstop <pin-number>
              Stops any separate process that is currently performing IRQ
              handling on given <pin-number>
              <pin-number> must be one of:
                      0,1,6,7,8,12,13,14,15,16,17,18,19,23,26
      expled <led-hex-value>
      Or:
      expled rgb <red> <green> <blue>
              Starts a separate process to output given colour to
              expansion dock LED.
              <led-hex_value> specifies the LED colours in hex in the form:
                      '0xrrggbb' where each colour component is in the hex
                      range '00' (off) to 'ff' (fully on)
              <red> <green> <blue> specify the LED colours in decimal
                      Each must be in the range 0 (off) to 100 (fully on)
              The separate process runs until the 'expledstop' operation
              is performed
              NOTE: expled uses pins:15 (blue), 16 (green), 17 (red)
      expledstop
              Stops any separate process that is currently performing
              expled output
      tone <pin-number> <frequency> <optional-duration>
              Starts a separate process to output a continous
              tone to given <pin-number> with given information
              <pin-number> must be one of:
                      0,1,6,7,8,12,13,14,15,16,17,18,19,23,26
              <frequency> is the frequency of the tone
                      Must be greater than 0
              <optional-duration> is an optional parameter that gives the
              duration of the output in milliseconds
                      If present, must be greater than or equal to 0
                      When absent or 0, duration is indefinite
              The separate process runs until the <optional-duration> expires
              or the 'tonestop' operation is performed on the same pin
              NOTE: equivalent to doing a 'pwm' operation with a <duty> of 50%
      tonestop <pin-number>
              Stops any separate process that is currently performing tone
              output on given <pin-number>
              <pin-number> must be one of:
                      0,1,6,7,8,12,13,14,15,16,17,18,19,23,26
      shiftout <datapin> <clockpin> <clockperiodNS> <bitorder> <value>
              Performs a serial output of a value on a data pin
              clocked by a clock pin
              <datapin> is the pin number for the data. Must be one of:
                      0,1,6,7,8,12,13,14,15,16,17,18,19,23,26
              <clockpin> is the pin number for the clock. Must be one of:
                      0,1,6,7,8,12,13,14,15,16,17,18,19,23,26
              <datapin> and <clockpin> must be different
              <clockperiodNS> is the period of the clock in nano-seconds
                      Must not be less than 100
              <bitorder> is the order the bits are transferred
                      Must be one of
                              msb - most significant bit first
                              lsb - least significant bit first
              <value> is the value sent
                      Must be >= 0 and <= 255
      shiftin <datapin> <clockpin> <clockperiodNS> <bitorder>
              Performs a serial input of a value from a data pin
```

```
                clocked by a clock pin
                <datapin> is the pin number for the data. Must be one of:
                        0,1,6,7,8,12,13,14,15,16,17,18,19,23,26
                <clockpin> is the pin number for the clock. Must be one of:
                        0,1,6,7,8,12,13,14,15,16,17,18,19,23,26
                <datapin> and <clockpin> must be different
                <clockperiodNS> is the period of the clock in nano-seconds
                        Must not be less than 100
                <bitorder> is the order the bits are transferred
                        Must be one of
                                msb - most significant bit first
                                lsb - least significant bit first
        pulseout <pin-number> <duration> <level>
                Outputs a single pulse to given <pin-number> with supplied data
                <pin-number> must be one of:
                        0,1,6,7,8,12,13,14,15,16,17,18,19,23,26
                <duration> is the length of pulse in micro-seconds. Must be >0
                <level> is the pulse level. Must be one of: 0 or 1
        pulsein <pin-number> <level> <timeout>
                Waits for and displays and returns duration of pulse input on
                given <pin-number>
                <pin-number> must be one of:
                        0,1,6,7,8,12,13,14,15,16,17,18,19,23,26
                <level> is the pulse level. Must be one of: 0 or 1
                <timeout> is the time to wait for the pulse and its completion
                in micro-seconds.
                        Must be >=0  If 0, timeout is indefinite
        frequency <pin-number> <sampletime>
                Obtains, displays and returns frequency of input on
                given <pin-number>
                Operates by counting the number of pulses during the sample time
                <pin-number> must be one of:
                        0,1,6,7,8,12,13,14,15,16,17,18,19,23,26
                <sampletime> is the time in milliseconds over which the
                frequency is taken. Must be >0
        delay <delayMS>
                Pauses execution for the given period
                <delayMS> is the delay period in milliseconds. Must be >=0
        exec <command>
                Executes the given command
                <command> is the command to execute.
                Must be enclosed in " characters if it contains spaces
                or other special character
                The <command> may contain any number of <varsub>s which will be
                replaced by actual values at run time.
                A <varsub> is any sequence like {<varref>}
                where '<varref>' is one of:
                  - <variable-name> = current value of variable
                    as assigned in an 'assign' operation
                    or not substituted if variable has never been assigned
                  - $n or $nn = current value of pin n or nn
                  - $! = value of latest result set by earlier commands
                  - $? = status of last executed command: 0=error, 1=ok
                  - $[<file-name>] = file <file-name> exists; 0=no, 1=yes
        assign <variable-name> <expression>
                Assigns the given expression to the given variable
                <variable-name> is the user variable to assign to.
                Any sequence of letters and digits starting with a letter.
                Names are case sensitive
                <expression> is the expression value to assign.
                Must be enclosed in " characters if it contains spaces
                or other special character.
                Formulated as a standard integer expression using:
                - Parentheses: ( and )
                - Unary Operators: + - ! ~
                - Binary Operators: * / % + - >> << > >= < <= == != & | ^ && ||
                - Integer constant values
                - <varref> which will be replaced by actual value at run time.
                  A <varref> is one of:
```

```
              - <variable-name> as defined above = current value of variable
                or 0 if variable has never been assigned
              - $n or $nn = current value of pin n or nn
              - $! = value of latest result set by earlier commands
              - $? = status of last executed command: 0=error, 1=ok
              - $[<file-name>] = file <file-name> exists; 0=no, 1=yes
      filein <file-name> <variable-name>
              Reads integer value from first line of file to variable
              Along with 'fileout', and 'filedelete' operations and
              $[<filename>] expression item, provides a rudimentary
              mechanism for communicating with other programs.
              <file-name> is the name of the file to read from.
              Must be enclosed in " characters if it contains spaces
              or other special character.
              <variable-name> is the user variable to which to assign
              the value read.
              Any sequence of letters and digits starting with a letter.
              Names are case sensitive
      fileout <file-name> <expression>
              Writes the value of the expression as the first line
              of given file.
              Along with 'filein', and 'filedelete' operations and
              $[<filename>] expression item, provides a rudimentary
              mechanism for communicating with other programs.
              <file-name> is the name of the file to write to.
              Must be enclosed in " characters if it contains spaces
              or other special character.
              <expression> is the expression value to write.
              Must be enclosed in " characters if it contains spaces
              or other special character.
              Formulated as a standard integer expression using:
              - Parentheses: ( and )
              - Unary Operators: + - ! ~
              - Binary Operators: * / % + - >> << > >= < <= == != & | ^ && ||
              - Integer constant values
              - <varref> which will be replaced by actual value at run time.
                A <varref> is one of:
                - <variable-name> as defined above = current value of variable
                  or 0 if variable has never been assigned
                - $n or $nn = current value of pin n or nn
                - $! = value of latest result set by earlier commands
                - $? = status of last executed command: 0=error, 1=ok
                - $[<file-name>] = file <file-name> exists; 0=no, 1=yes
      filedelete <file-name>
              Deletes the file with given name.
              Along with 'filein', and 'fileout' operations and
              $[<filename>] expression item, provides a rudimentary
              mechanism for communicating with other programs.
              <file-name> is the name of the file to delete.
              Must be enclosed in " characters if it contains spaces
              or other special character.
      while <conditional-expression>
      endwhile
              Repeatedly execute all operations between 'while' and 'endwhile'
              so long as <conditional-Expression> is true
              <conditional-expression> is true if the expression evaluates
              to non-zero.
              <conditional-expression> must be enclosed in " characters
              if it contains spaces or other special character.
              Formulated as a standard integer expression using:
              - Parentheses: ( and )
              - Unary Operators: + - ! ~
              - Binary Operators: * / % + - >> << > >= < <= == != & | ^ && ||
              - Integer constant values
              - <varref> which will be replaced by actual value at run time.
                A <varref> is one of:
                - <variable-name> = current value of variable
                  as assigned in an 'assign' operation
                  or 0 if variable has never been assigned
```

```
                  - $n or $nn = current value of pin n or nn
                  - $! = value of latest result set by earlier commands
                  - $? = status of last executed command: 0=error, 1=ok
                  - $[<file-name>] = file <file-name> exists; 0=no, 1=yes
        if <conditional-expression>
        endif
        Or:
        if <conditional-expression>
        else
        endif
                Executes all operations between 'if' and 'else' or
                'endif' if the <conditional-expression> is true
                If 'else' is used, executes all operations between 'else' and
                'endif' if the <conditional-expression> is false
                <conditional-expression> is true if the expression evaluates
                to non-zero.
                <conditional-expression> must be enclosed in " characters
                if it contains spaces or other special character.
                Formulated as a standard integer expression using:
                - Parentheses: ( and )
                - Unary Operators: + - ! ~
                - Binary Operators: * / % + - >> << > >= < <= == != & | ^ && ||
                - Integer constant values
                - <varref> which will be replaced by actual value at run time.
                  A <varref> is one of:
                  - <variable-name> = current value of variable
                    as assigned in an 'assign' operation
                    or 0 if variable has never been assigned
                  - $n or $nn = current value of pin n or nn
                  - $! = value of latest result set by earlier commands
                  - $? = status of last executed command: 0=error, 1=ok
                  - $[<file-name>] = file <file-name> exists; 0=no, 1=yes
        break
                If within a 'while' loop, breaks execution of the loop
                If not within a 'while' loop, ends all execution
        exit <result>
                Exits execution with given result
                <result> is the value to return on exit

Sources available at: https://github.com/KitBishop/new-gpio
```

Notes:

1. Any output from **-h** and any output is written to **stderr**. Any output as a result of the **-o** option is written to **stdout**
2. The return value from the command will be one of the following:
    - **255 (-1)** – indicates an error has occurred in the last executed operation – either in the parameters or in executing the command
    - **0** – indicates normal successfully completion for the last executed operation 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

3. When any of the **pwm**, **tone**, **irq**, **irq2** or **expled** operations is used, the program forks a separate process to perform the associated operation.

This separate process continues after the program returns until such time as the associate **xxxstop** operation is performed on the same pin.
The ID of any such separate processes can be discovered by running:

new-gpio info <pin-number>

7.2. Scripting Example for new-gpio Program

An example script file for the **new-gpio** program can be found in the GIT sources at **source/new-gpio/new-gpio-example.txt**

You can run this script with the **new-gpio** program so:

- Copy **new-gpio-example.txt** to the same directory on the Omega as the **new-gpio** program
- Ensure pin 0 on your Omega is connected to a **low** signal
- Run the following command on your Omega:

  ./new-gpio @new-gpio-example.txt &

  Note that the **&** at the end of the command ensures the command runs in the background
- On the prompt, connect pin 0 to a **high** signal
- The main processing of the script will continue to run doing the following:
  - Once a second changes the expansion LED to red, then green, then blue cyclically
  - Once every ten displays a diagnostic of the current loop count value
- The main processing continues until one of the following occurs:
  - Pin 0 is connected to a **low** signal
  - A file in the directory with name **stop.txt** exists that has the value 99 as the first line:
    - execute the command:
      - **echo 99 > stop.txt**
- When the main processing is terminated as above:
  - The reason the processing was stopped will be displayed
  - The expansion LED will be turned off
  - The **new-gpio** program will be terminated

For reference, the contents of the **new-gpio-example.txt** file (with comments in lines starting with **#** which are ignored by **new-gpio**) is:

```
# Ensure that pin directions set appropriately by using the -s option
-s

# Use the -q option to suppress extraneous output
-q

# Set 'conut' variable to 0
assign count 0

# Ensure expled is turned off
expled rgb 0 0 0

# Initial instruction to start main processing loop
exec "echo"
exec "echo"
exec "echo Ensure pin 0 is connected to a high to start main processing"

# Wait until pin 0 is high
while !$0
endwhile
```

```
# Clear file stop.txt to start with 0
fileout stop.txt 0

# Provide feedback on how to stop main processing loop
exec "echo"
exec "echo Main processing started by pin 0 being connected to a high"
exec "echo It will continue until pin 0 is low or the file \'stop.txt\'"
exec "echo ' starts with a line containing 99'"
exec "echo"

# Main processing loop runs so long as pin 0 is high
while $0

# Set expled to red when count is a multiple of 3
    if "(count % 3) == 0"
       expled rgb 100 0 0
    endif

# Set expled to green when count is a (multiple of 3) + 1
    if "(count % 3) == 1"
       expled rgb 0 100 0
    endif

# Set expled to blue when count is a (multiple of 3) + 2
    if "(count % 3) == 2"
       expled rgb 0 0 100
    endif

# Delay for 1 second
    delay 1000

# Display current count when count is a multiple of 10
    if "((count % 10) == 0) && (count != 0)"
       exec "echo count is now {count}"
    endif

# Increment count
    assign count "count+1"

# Check that file stop.txt exists
# Actually redundant since if it doesn't reading from it will return 0 anyway
    if $[stop.txt]

# Read first line from file stop.txt to variable stopval
       filein "stop.txt" stopval

# Test if stopval read is 99 and break main loop
       if "stopval == 99"
           break
       endif
    endif

# End of while loop
endwhile

# Report on reason that main loop ended
if $[stop.txt]
    exec "echo"
    exec "echo Processing ended by existence of \'stop.txt\' starting with line
with value 99"
else
    exec "echo"
    exec "echo Processing ended by pin 0 going low"
endif

# Turn expled off
expled rgb 0 0 0
```

```
# Stop process responsible for running expled
expledstop
```

8. Usage of the new-expled Program

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

The **new-expled** program accepts a parameter to set the colour of the led. The **new_expled** program provides exactly the same functionality as the existing **expled** script except that it is wriiten in C++ and uses **libnew-gpio**.

The program will document its usage when the command **new-expled 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 **new-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 **new-expled stop** command is run.
   The ID of the separate process can be discovered by running the **new-gpio** command:
   new-gpio info <pin-number>
   Where <pin-number> is one of the expansion led pins: **15, 16, 17** or **all**

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

   As are these two:

   - **new-expled rgb <r> <g> <b>**
   - **new-gpio expled rgb <r> <g> <b>**

And these two:

- **new-expled stop**
- **new-gpio expledstop**

9. Further Development

Development of **new-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 **new-gpio**. In particular:

- Similar code for **i2c** access
- Code to provide **Omega<-->Arduino** communication and control