# Overview

BasysMX3 incorporates a wide range of modules. Digilent provides a set of libraries, allowing user easy access to each module’s functionality.

Basically, these libraries hide the hardware implementation details; they are wrappers over the lower level functions that access the registers, allowing the user to call the needed functionality in an easy and intuitive manner.

# Libraries usage

Normally each library has a .c and .h file. In order to use this library:

* include in your project the .c and .h files corresponding to the module you want to use (for example led.c and led.h). This can be easily done by copying the files in the project folder and then Add Existing Item (right mmouse click on the project source folders and project header files folders.
* In your code, include the header of the module

#include "led.h"

* In your code, call the needed functions, for example:

LED\_Init();

LED\_SetValue(4, 1); //turn on LED4

Using these libraries is not mandatory, of course you can write your own functionality (eventually using the libraries as inspiration).

# LED

## Overview

The LED library groups the functions that access onboard LEDs (labeled “LD0” – “LD7”).

Library files

The library is implemented in these files: led.c, led.h. Include them in the project.

Include the library header file whenever you want to use the library functions:

#include "led.h"

LED numbering

The numbering is consistent with the LEDs labeling:

|  |  |
| --- | --- |
| Number | LED |
| 0 | LD0 |
| 1 | LD1 |
| ... |  |
| 7 | LD7 |

LED values

The library deals with logic value assigned to a LED, having this meaning:

|  |  |
| --- | --- |
| Value | Meaning |
| 0 | LED is off (not lighting) |
| 1 | LED is on (lighting) |

## Library functions

### LED\_Init

Synopsis

void LED\_Init();

Description

This function initializes the hardware involved in the LED module: the pins corresponding to LEDs are initialized as digital outputs.

Example

#include "led.h"

LED\_Init();

### LED\_SetValue

Synopsis

void LED\_SetValue(unsigned char bNo, unsigned char bVal);

Parameters

* bNo – the number of LED whose value will be set. The value must be between 0 and 7.
* bVal – the value to be assigned to the specific LED:

|  |  |
| --- | --- |
| bVal parameter value | Action |
| 0 | LD<bNo> is turned off |
| Other than 0 | LD<bNo> is turned on |

Description

This function sets the value to the LED specified by bNo to the value specified by bVal.

If the value provided for bNo is not between 0 and 7, only the LSB 3 bits will be considered.

Example

#include "led.h"

LED\_Init();

LED\_SetValue(3, 1); // turns on the LD3

### LED\_ToggleValue

Synopsis

void LED\_ToggleValue(unsigned char bNo);

Parameters

* bNo – the number of LED whose value will be toggled. The value must be between 0 and 7.

Description

This function toggles the value of the LED specified by bNo. If the LED was off, it is turned on. If the LED was on, it is tuned off.

If the value provided for bNo is not between 0 and 7, only the LSB 3 bits will be considered.

Example

#include "led.h"

LED\_Init();

LED\_ToggleValue(3); // toggles the value of LD3

### LED\_SetGroupValue

Synopsis

void LED\_SetGroupValue(unsigned char bVal);

Parameters

* bVal – the 8 bit value B7 B6 B5 B4 B3 B2 B1 B0 that specifies the values for the 8 LEDs.

|  |  |
| --- | --- |
| Bi value | Action |
| 0 | LD<i> is turned off |
| 1 | LD<i> is turned on |

Description

This function sets the value for all 8 LEDs, according to the value provided in bVal. Each bit from bVal corresponds to a LED: Bit 0 (LSB) corresponds to LD0, bit 7 (MSB) corresponds to LD7.

Example

#include "led.h"

LED\_Init();

LED\_SetGroupValue(0xAA); // assigns alternate values to the LEDs

# SWT

## Overview

The SWT library groups the functions that access onboard Switches (labeled “SW0” – “SW7”).

Library files

The library is implemented in these files: swt.c, swt.h. Include them in the project.

Include the library header file whenever you want to use the library functions:

#include "swt.h"

Switches numbering

The numbering is consistent with the switches labeling:

|  |  |
| --- | --- |
| Number | LED |
| 0 | SW0 |
| 1 | SW1 |
| ... |  |
| 7 | SW7 |

Switch values

The library deals with logic value assigned to a switch, having this meaning:

|  |  |
| --- | --- |
| Value | Meaning |
| 0 | switch is off (position backwards, towards the edge of the board) |
| 1 | switch is off (position forward, towards LEDs) |

## Library functions

### SWT\_Init

Synopsis

void SWT\_Init();

Description

This function initializes the hardware involved in the SWT module: the pins corresponding to switches are initialized as digital inputs.

Example

#include "swt.h"

SWT\_Init();

### SWT\_GetValue

Synopsis

unsigned char SWT\_GetValue(unsigned char bNo)

Parameters

* bNo – the number of switch whose value will be get. The value must be between 0 and 7.

Return

* unsigned char – the value corresponding to the specified switch:

|  |  |
| --- | --- |
| return value | when |
| 0 | SW<bNo> is turned off |
| 1 | SW<bNo> is turned on |

Description

This function gets the value of the switch specified by bNo. If the value provided for bNo is not between 0 and 7, only the LSB 3 bits will be considered.

Example

#include “swt.h”

SWT\_Init();

unsigned char val = SWT\_GetValue(3); // gets the value of SW3

### SWT\_GetGroupValue

Synopsis

unsigned char SWT\_GetGroupValue()

Return

* unsigned char – the 8 bit value B7 B6 B5 B4 B3 B2 B1 B0 where each bit Bi corresponds to SW<i>:

|  |  |
| --- | --- |
| Bi has the value | when |
| 0 | SW<i> is turned off |
| 1 | SW<i> is turned on |

Description

This function gets the value of the all 8 switches as a single value on 8 bits.

Each bit from returned value corresponds to a switch: Bit 0 (LSB) corresponds to SW0, bit 7 (MSB) corresponds to SW7.

Example

#include “swt.h”

SWT\_Init();

unsigned char val = SWT\_GetGroupValue(); // gets the value of all SWTs

# BTN

## Overview

The BTN library groups the functions that access onboard Buttons (labeled “BTNU”, “BTNL”, “BTNC”, “BTNR”, “BTND”).

The library just performs basic digital input. No debouncing functionality is provided.

Library files

The library is implemented in these files: swt.c, btn.h. Include them in the project.

Include the library header file whenever you want to use the library functions:

#include "btn.h"

Buttons designation

Each button can be identified using a number or a letter (uppercase or lowercase), see table below:

|  |  |  |
| --- | --- | --- |
| Number | Letter | Button |
| 0 | ‘U’, ‘u’ | BTNU |
| 1 | ‘L’, ‘l’ | BTNL |
| 2 | ‘C’, ‘c’ | BTNC |
| 3 | ‘R’, ‘r’ | BTNR |
| 4 | ‘D’, ‘d’ | BTND |

Button values

The library deals with logic value assigned to a button, having this meaning:

|  |  |
| --- | --- |
| Value | Meaning |
| 0 | button is not pressed |
| 1 | button is pressed |

## Library functions

### BTN\_Init

Synopsis

void BTN\_Init();

Description

This function initializes the hardware involved in the BTN module: the pins corresponding to buttons are initialized as digital inputs.

Example

#include "btn.h"

BTN\_Init();

### BTN\_GetValue

Synopsis

unsigned char BTN\_GetValue(unsigned char btn)

Parameters

* btn – the identification of button whose value will be read. The value can be the number or the letter associated to a button. It must be between 0 and 4, or one of ‘U’, ‘u’, ‘L’, ‘l’, ‘C’, ‘c’, ‘R’, ‘r’, ‘D’, ‘d’.

|  |  |  |
| --- | --- | --- |
| Number | Letter | Button |
| 0 | ‘U’, ‘u’ | BTNU |
| 1 | ‘L’, ‘l’ | BTNL |
| 2 | ‘C’, ‘c’ | BTNC |
| 3 | ‘R’, ‘r’ | BTNR |
| 4 | ‘D’, ‘d’ | BTND |

Return

* unsigned char – the value corresponding to the specified button:

|  |  |
| --- | --- |
| return value | when |
| 0 | Corresponding button is not pressed |
| 1 | Corresponding button is pressed |

Description

This function gets the value of the BTN specified by btn. If the value provided for btn is not between 0 and 45, or one of ‘U’, ‘u’, ‘L’, ‘l’, ‘C’, ‘c’, ‘R’, ‘r’, ‘D’, ‘d’, then the function returns 0xFF.

Example

#include “btn.h”

BTN\_Init();

unsigned char val = BTN\_GetValue(‘C’); // gets the value of BTNC

### BTN\_GetGroupValue

Synopsis

unsigned char BTN\_GetGroupValue()

Return

* unsigned char – the 8 bit value 0 0 0 B4 B3 B2 B1 B0 where bit Bi corresponds to BTN<i>:

|  |  |  |  |
| --- | --- | --- | --- |
| Bit | has the value | When | |
| B0 | 0 | BTNU | is not pressed |
| 1 | is pressed |
| B1 | 0 | BTNL | is not pressed |
| 1 | is pressed |
| B2 | 0 | BTNC | is not pressed |
| 1 | is pressed |
| B3 | 0 | BTNR | is not pressed |
| 1 | is pressed |
| B4 | 0 | BTND | is not pressed |
| 1 | is pressed |

Description

This function gets the value of the all 5 buttons as a single value on 8 bits.

The 5 LSB bits from returned value corresponds to a button: bit 0 corresponds to BTNU, bit 1 corresponds to BTNL, bit 2 corresponds to BTNC, bit 3 corresponds to BTNR, bit 4 corresponds to BTND (see return value explanation).

Example

#include “btn.h”

BTN\_Init();

unsigned char val = BTN\_GetGroupValue(); // gets the value of all BTNs

# RGBLED

## Overview

The RGBLED library groups the functions that access the RGBLED module.

Each color can be identified using the 3 basic components: red, green and blue.

The BasysMX3 provides 3 digital signals to control the RGBLED module, one for each basic color R, G, B: LED8\_R, LED8\_G and LED8\_B.

Using either 0 or 1 values for these signals will only give the user a limited number of colors (2 colors for each component), so most of the time this is not enough in applications using the RGB feature. The solution is to send a sequence of 1 and 0 values on these digital lines, switched rapidly with a frequency higher than the human perception. The “duty factor” will finally determine the color, as the human eye will “integrate” the discrete illumination values into the final color sensation. The most common ways to implement this are: PWM (pulse width modulation) and PDM (pulse density modulation).

The RGBLED library uses PDM, still provides the PWM implementation using commented code. The advantages of PDM approach are: it saves hardware resources to be used elsewhere (3 output compare modules), the implementation is rather simple and it provides a good example of PDM. The advantages of using PWM are: it uses hardware modules (3 output compare modules), thus decreasing the processor load, and it is well known to many users.

Implementation features (PDM method)

The PDM method implemented in the RGBLED library has the following features:

* The digital pins corresponding to each color (LED8\_R, LED8\_G and LED8\_B) are configured as simple digital outputs.
* Timer5 is configured to generate an interrupt every approx. 300 µs.
* Each color component has a 8 bit resolution: a color is set by providing its components (R, G and B) as 8 bits values.
* In the Timer5 interrupt service routine, for each component color (R, G and B):
  + One 16 bits accumulators are used, one for each color.
  + the 8 bits component color value is added to the corresponding 16 bits accumulator.
  + the 9th bit of the accumulator is considered the carry bit. The resulted carry bits are assigned to the digital pins corresponding to each color (LED8\_R, LED8\_G and LED8\_B).
  + carry bit is cleared in the accumulator.

RGBLED implemented using PWM

As mentioned above, the library source file (rgbled.c) contains as commented code the PWM implementation of the functionality needed for RGBLED module.

It has the following features:

* The digital pins corresponding to each color (LED8\_R, LED8\_G and LED8\_B) are mapped to the output of OC3, OC4, OC5 modules of PIC32.
* The OC3, OC4, OC5 and Timer2 modules are configured to work together to generate PWM.
* Every time a color is set, the OCxRS register sets the appropriate PWM.

Library files

The library is implemented in these files: rgbled.c, rgbled.h. Include them in the project.

Include the library header file whenever you want to use the library functions:

#include " rgbled.h"

## Library functions

### RGBLED\_Init

Synopsis

void RGBLED\_Init();

Description

This function initializes the hardware involved in the RGBLED module: the pins corresponding to R, G and B colors are initialized as digital outputs and Timer 5 in configured.

Example

#include "rgbled.h"

RGBLED\_Init();

### RGBLED\_SetValue

Synopsis

void RGBLED\_SetValue(unsigned char bValR, unsigned char bValG, unsigned char bValB);

Parameters

* bValR – the value that corresponds to R component of the color.
* bValG – the value that corresponds to G component of the color.
* bValB – the value that corresponds to B component of the color.

Description

This function sets the color value by providing the values for the 3 components: R, G and B, as 3 separate 8 bits values.

Example

#include "rgbled.h"

RGBLED\_Init();

RGBLED\_SetValue(0xFF, 0x7F, 0); // set color R = 0xFF, G = 0x7FF, B = 0

RGBLED\_SetValueGrouped

Synopsis

void RGBLED\_SetValueGrouped(unsigned int uiValRGB);

Parameters

* uiValRGB – the value that groups in the 3 LSB bytes the color values in this pattern: xxxxxxxxRRRRRRRRGGGGGGGGBBBBBBBB.

|  |  |  |  |
| --- | --- | --- | --- |
| Bits | | when | |
| 23-16 | | Byte corresponding to R color | |
| 15-8 | | Byte corresponding to G color | |
| 7-0 (LSB byte) | | Byte corresponding to B color | |

Description

This function sets the color value by providing the values for the 3 components R, G and B as a 24 bits value, placed in the LSB 3 bytes of the uiValRGB like this: bits 23-16 correspond to color R, bits 15-8 correspond to color G, bits 7-0 (LSB byte) correspond to color B.

Example

#include "rgbled.h"

RGBLED\_Init();

RGBLED\_SetValueGrouped(0xFF7F00); // set color R = 0xFF, G = 0x7FF, B = 0

### RGBLED\_Close

Synopsis

void RGBLED\_Close();

Description

This function can be called when RGBLED library is no longer needed: it stops the Timer5 and turns off the RGBLED.

Example

#include "rgbled.h"

RGBLED\_Close()**;**

# ACL

## Overview

The ACL library groups the functions that access onboard accelerometer. It is a 3-Axis, 12-bit, Digital accelerometer, exposing an I2C digital interface.

The library implements I2C access to the onboard accelerator device and provides basic functions to configure the accelerometer and provide the accelerometer values (raw values and g values).

The Basys MX3 board also provides a digital pin (ACL\_INT2) that can be used as an interrupt raised by the accelerometer. The library implements no functionality related to this pin, it just configures the pin as a digital input, so user should properly configure the accelerometer related to this interrupt.

Note that the I2C access functions (labeled ACL\_I2C) can be used if another I2C implementation is needed.

Raw values and g Values

For each of the 3 axis, the value retrieved from the accelerator (over I2C) is an 2’s complement 12 bit value. It is considered as raw value, and is represented on 2 bytes.

The library provides conversion of the raw values to the values expressed in terms of g.

The conversion is done in the following manner:

* Compute the conversion constant: value in g corresponding for each count of the raw value. This value only changes when accelerator full scale range is changed.
* the selected accelerator full scale range corresponds to a range of g values.
* for example, for ±2g full scale range, the g values are between -2g and +1.999g, making a 4 g range.
* the range of g values is implemented using 12 bits raw value.
* for each count from the 12 bits raw value, it corresponds a value in g:
  + (<full scale range> / 2^12)
* for example, for ±2g full scale range, the value corresponding to each count is (4/2^12)g.
* To compute the value in g, the raw value is multiplied with the conversion constant (value corresponding to each count), considering the sign from the 12 bit raw value.

Float values computing

The method of converting the raw values in g values is implemented using float values computing. This costs performance, so user should be aware of it. The library is implemented so that the conversion constant (value corresponding to each count) is pre-computed when the full scale range is set.

If user chooses to use the raw values instead of g values (use ACL\_ReadRawValues instead of ACL\_ReadGValues), no float values computing is involved.

Library files

The library is implemented in these files: acl.c, acl.h. Include them in the project.

Include the library header file whenever you want to use the library functions:

#include "acl.h"

## Library functions

### ACL\_Init

Synopsis

void ACL\_Init();

Description

This function initializes the hardware involved in the ACL module: the I2C1 module of PIC32 is configured to work at 400 khz, the ACL is initialized at ±2g full scale range and the ACL\_INT2 pin is configured as digital input.

Example

#include "ACL.h"

ACL\_Init();

### ACL\_GetValue

Synopsis

unsigned char ACL\_GetValue(unsigned char ACL)

Parameters

* ACL – the identification of button whose value will be read. The value can be the number or the letter associated to a button. It must be between 0 and 4, or one of ‘U’, ‘u’, ‘L’, ‘l’, ‘C’, ‘c’, ‘R’, ‘r’, ‘D’, ‘d’.

|  |  |  |
| --- | --- | --- |
| Number | Letter | Button |
| 0 | ‘U’, ‘u’ | ACLU |
| 1 | ‘L’, ‘l’ | ACLL |
| 2 | ‘C’, ‘c’ | ACLC |
| 3 | ‘R’, ‘r’ | ACLR |
| 4 | ‘D’, ‘d’ | ACLD |

Return

* unsigned char – the value corresponding to the specified button:

|  |  |
| --- | --- |
| return value | when |
| 0 | Corresponding button is not pressed |
| 1 | Corresponding button is pressed |

Description

This function gets the value of the ACL specified by ACL. If the value provided for ACL is not between 0 and 45, or one of ‘U’, ‘u’, ‘L’, ‘l’, ‘C’, ‘c’, ‘R’, ‘r’, ‘D’, ‘d’, then the function returns 0xFF.

Example

#include “ACL.h”

ACL\_Init();

unsigned char val = ACL\_GetValue(‘C’); // gets the value of ACLC

# Analog Input Control – AIC

## Overview

This library groups the functions that implement the AIC module. The library is used to handle analog input control on thumbwheel potentiometer or analog input connectors labeled AIC. It calls functions defined in the ADC library. Include the file as well as adc.c and adc.h in the project when this library is needed.

# ADC

## Overview

This library groups the functions that implement the ADC module. The ADC module implements basic ADC (analog to Digital converter) functionality. This library is used by AIC and MIC libraries, and can be used stand alone for reading any analog input pin. Include this file in the project when the AIC and MIC libraries are needed.

# Audio

## Overview

This library groups the functions that implement the AUDIO module. The library implements the generating audio function using a sine of a certain frequency, recording and playback functionality. In order to generate audio using a sine of a certain frequency, the sine values are produced using PWM on OC1.

Include the file in the project together with config.h when this library is needed.

# IRDA

## Overview

This library groups the functions that implement the IRDA module. The library implements the control for it and it also provides an UART interface over the IRDA transmit and receive pins.

Include the file together with utils.c and utils.h in the project when this library is needed.

# LCD

## Overview

This library groups the functions that implement the LCD device, using specific LCD control functions. It is accessed in a "parallel like" approach. The library provides functions for simple commands, displaying characters, handling user characters. Include the file together with config.h, utils.c and utils.h in the project when this library is needed.

# MIC

## Overview

This library groups the functions that implement the MIC device control. The library is used to handle analog input from the microphone. It relies on the ADC module. Include the file as well as adc.c and adc.h in the project when this library is needed.

# MOT

## Overview

This library groups the functions that implement the MOT module. There are two modes: PH/EN and IN/IN. After common initialization functions, separate functions are provided for each mode. The PH/EN mode uses OC2 and OC3 in order to generate PWM. Include the file in the project, together with config.h when this library is needed.

# PMODS

## Overview

This library groups the functions that implement the PMODS functionality. Pins from PMODA and PMODB can be initialized as digital input / output pins, their value can be access using set / get functions. Include the file in the project, together with config.h when this library is needed.

# SPIFLASH

## Overview

This library groups the functions that implement the SPIFLASH device. The library implements SPI access to the onboard SPI Flash memory and provides basic functions to configure the SPI Flash memory, write and read functions to access SPI Flash memory bytes. Include the file in the project, together with config.h, when this library is needed.

# SPIJA

## Overview

The library implements SPI2 access over pins on PMODA connector (JA). Include the file in the project, together with config.h, when this library is needed.

# SRV

## Overview

This library handles the 2 servo modules SERVO 1 and SERVO 2.

Servo pins are mapped over OC5 and OC4 that are properly configured (period 20 ms) in order to generate PWM for each servo. For each servo, the control function allows to set the pulse duration in microseconds. Include the file in the project, together with config.h when this library is needed.

# SSD

## Overview

This library deals with seven segment display modules.

As digits share the cathodes, in order to be able to display different information on each digit, periodically each digit is refreshed, while the other are disabled (Timer1 is used to generate an interrupt).

The library provides functions for setting the information to be displayed.

Include the file in the project, together with config.h, when this library is needed.

# UART

## Overview

This library implements the UART4 functionality connected to the USB – UART interface labeled UART. It provides basic functions to configure UART and transmit / receive functions. The receive can be done both with interrupts and with polling method.

Include the file in the project, together with config.h, when this library is needed.

# UARTJB

## Overview

This library implements UART1 interface over the PMODB pins: JB2 (RD11) is mapped as TX1, JB3 (RD10) is mapped as RX1. It provides basic functions to configure UART and transmit / receive functions. The receive can be done both with interrupts and with polling method.

Include the file in the project, together with config.h, when this library is needed.

# Hardware resources map

|  |  |
| --- | --- |
| Library | Action |
| RGBLED | Timer5 |
|  |  |

Commented functionality

|  |  |
| --- | --- |
| Library | Action |
| RGBLED | Timer2, OC3, OC4 and OC5 |
|  |  |