Lab 3B: Modify Switch On/Off

This hands-on exercise will demonstrate how to make a modification on one of the sample applications that ships as part of the Z-Wave SDK.

This exercise is part of the series “Z-Wave 1-Day Course”.

1: Include using SmartStart

2: Decrypt Z-Wave RF Frames using the Zniffer

3A: Compile Switch On/Off and Enable Debug

3B: Modify Switch On/Off

4: Understand FLiRS devices

KEY FEATURES

- Change GPIO
- Implement PWM
- Use on-board RGB LED

1 Introduction

This exercise is building on top of the previous exercise “3A: Compile Switch On/Off and enable debug”, which demonstrated how to compile and use the Switch On/Off sample application.

In this exercise we will be making a modification to the sample application, by change the GPIO that controls the LED. In addition, we will be using a RGB LED and learn how to use PWM to change colors.

1.1 Hardware Requirements

- 1 WSTK Main Development Board
- 1 Z-Wave Radio Development Board: ZGM130S SiP Module
- 1 UZB Controller
- 1 USB Zniffer

1.2 Software Requirements

- Simplicity Studio v4
- Z-Wave 7 SDK
- Z-Wave PC Controller
- Z-Wave Zniffer

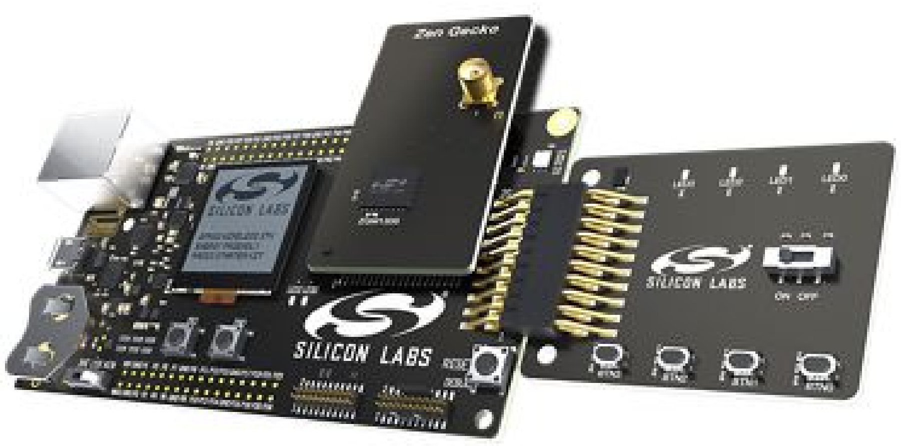


Figure 1: Main Development Board with Z-Wave SiP Module

1.3 Prerequisites

Previous Hands-On exercises has covered how to use the PC Controller and Zniffer application to build a Z-Wave network and capturing the RF communication for development purpose. This exercise assumes you are familiar with these tools.

Previous Hands-On exercises has also covered how to use the sample applications that ships with the Z-Wave SDK. This exercise assumes you are familiar with using and compiling one of the sample applications.

2 Navigate the Board Interface

The Z-Wave framework comes with a hardware abstraction layer (HAL) defined by board.h and board.c, providing the possibility of having implementations for each of your hardware platforms.

The Hardware Abstraction Layer (HAL) is program code between a system's hardware and its software that provides a consistent interface for applications that can run on several different hardware platforms. To take advantage of this capability, applications should access hardware through the API provided by the HAL, rather than directly. Then, when you move to new hardware, you only need to update the HAL.

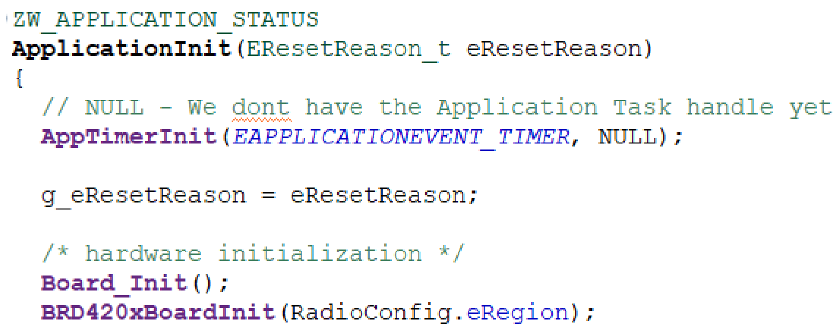
2.1 Open Sample Project

For this exercise you need to open the Switch On / Off sample application. If you completed exercise "3A Compile Switch OnOff and enable debug", it should already be opened in your Simplicity Studio IDE.

In this section we will be looking at the board files and understand how the LEDs are initialized.

1. From the main file "`SwitchOnOff.c`", locate "`ApplicationInit()`" and notice the call to `Board_Init()`.

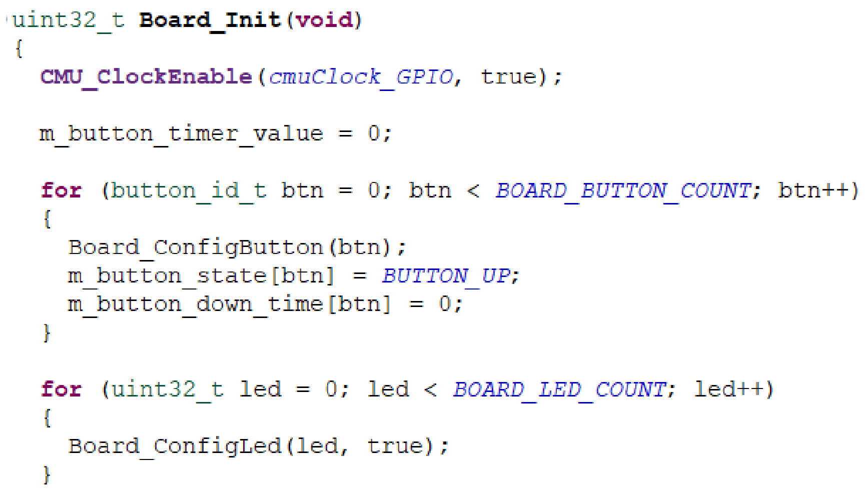
2. Place your courser on `Board_Init()` and press on F3 to open the declaration.



```
ZW_APPLICATION_STATUS
ApplicationInit(EResetReason_t eResetReason)
{
  // NULL - We dont have the Application Task handle yet
  AppTimerInit(EAPPLICATIONEVENT_TIMER, NULL);

  g_eResetReason = eResetReason;

  /* hardware initialization */
  Board_Init();
  BRD420xBoardInit(RadioConfig.eRegion);
```

Figure 2: SwitchOnOff.c - ApplicationInit()

3. In `Board_Init()`notice how LEDs contained in `BOARD_LED_COUNT` are being initialized by called `Board_ConfigLed()`

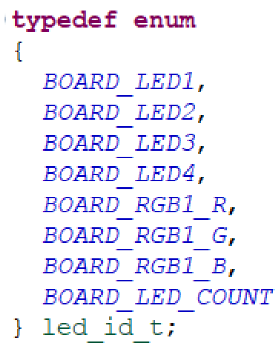


```
uint32_t Board_Init(void)
{
  CMU_ClockEnable(cmuClock_GPIO, true);

  m_button_timer_value = 0;

  for (button_id_t btn = 0; btn < BOARD_BUTTON_COUNT; btn++)
  {
    Board_ConfigButton(btn);
    m_button_state[btn] = BUTTON_UP;
    m_button_down_time[btn] = 0;
  }

  for (uint32_t led = 0; led < BOARD_LED_COUNT; led++)
  {
    Board_ConfigLed(led, true);
  }
```

Figure 3: board.c - Board_Init()

4. Place your courser on `BOARD_LED_COUNT` and press on F3 to open the declaration.

5. The LEDs defined in `led_id_t` are as follows:



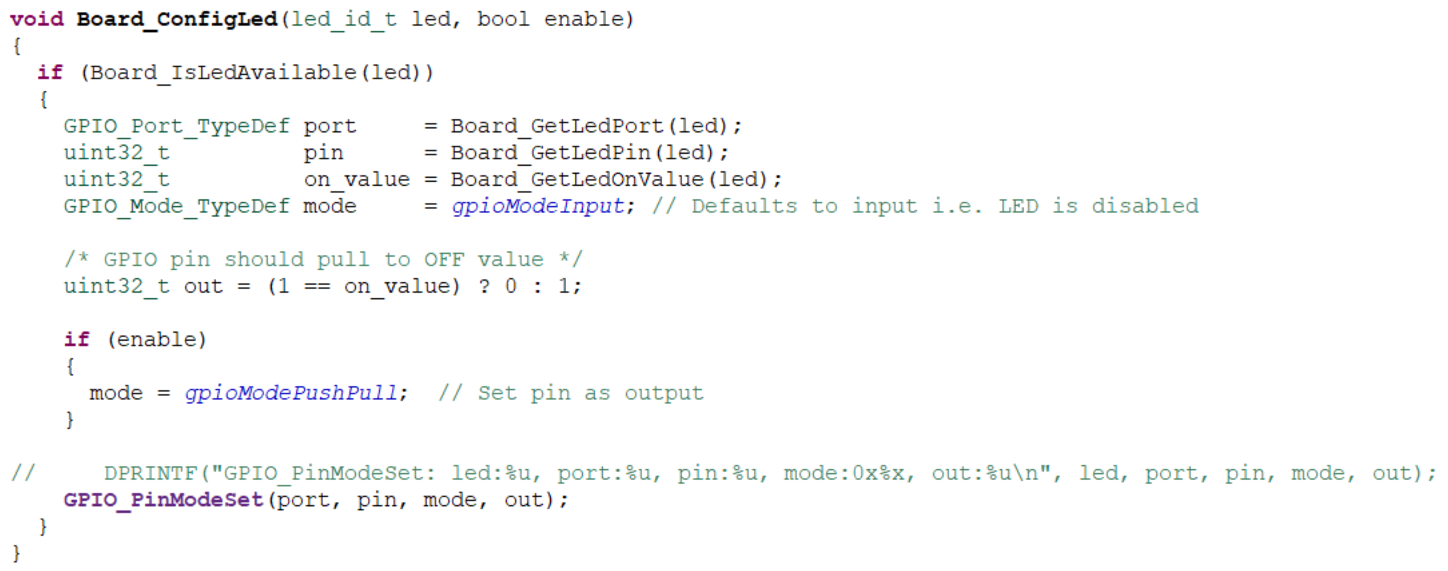
```
typedef enum
{
  BOARD_LED1,
  BOARD_LED2,
  BOARD_LED3,
  BOARD_LED4,
  BOARD_RGB1_R,
  BOARD_RGB1_G,
  BOARD_RGB1_B,
  BOARD_LED_COUNT
} led_id_t;
```

Figure 4: board.h - LED Identifier Type

6. Return to the `board.c` file.

7. Place your courser on `Board_ConfigLed()` and press on F3 to open the declaration.

8. Notice all the LEDs defined in `led_id_t` are then configured in `Board_ConfigLed()` as output.



```
void Board_ConfigLed(led_id_t led, bool enable)
{
  if (Board_IsLedAvailable(led))
  {
    GPIO_Port_TypeDef port     = Board_GetLedPort(led);
    uint32_t          pin      = Board_GetLedPin(led);
    uint32_t          on_value = Board_GetLedOnValue(led);
    GPIO_Mode_TypeDef mode     = gpioModeInput; // Defaults to input i.e. LED is disabled

    /* GPIO pin should pull to OFF value */
    uint32_t out = (1 == on_value) ? 0 : 1;

    if (enable)
    {
      mode = gpioModePushPull;  // Set pin as output
    }

//     DPRINTF("GPIO_PinModeSet: led:%u, port:%u, pin:%u, mode:0x%x, out:%u\n", led, port, pin, mode, out);
    GPIO_PinModeSet(port, pin, mode, out);
  }
}
```

Figure 5: board.c - Board_ConfigLed()

What this means is, that all LEDs on the development board are already defined as outputs and ready to use.

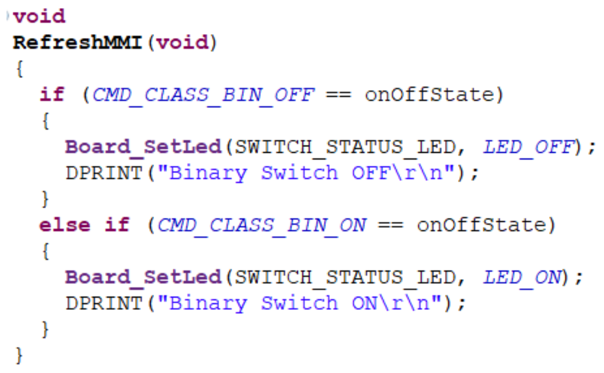
3 Make a Modification to a Z-Wave Sample Application

In this exercise we will be modifying the GPIOs used for the LED in the Switch On/Off sample application. In the previous section we learned how all LEDs on the development board are already initialized as output and ready to use.

3.1 Use the RGB LED

We will be using the onboard RGB LED on the Z-Wave development module, instead of the LED on the button board.

1. Locate the `RefreshMMI` function, as seen in Figure 6, in the `SwitchOnOff.c` main application file.

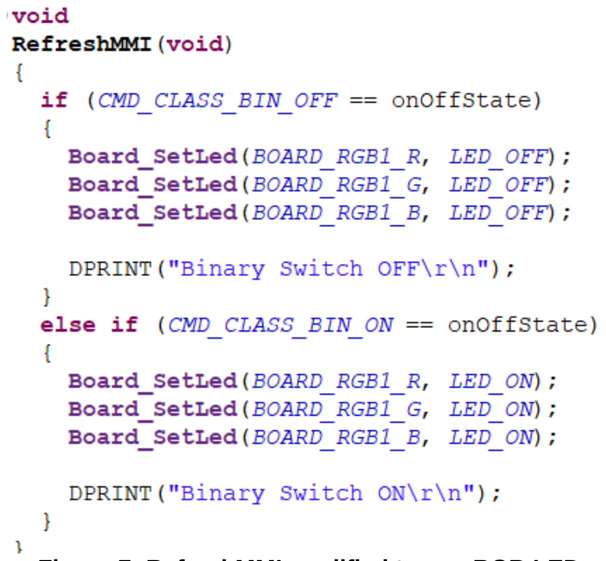


```
void
RefreshMMI(void)
{
  if (CMD_CLASS_BIN_OFF == onOffState)
  {
    Board_SetLed(SWITCH_STATUS_LED, LED_OFF);
    DPRINT("Binary Switch OFF\r\n");
  }
  else if (CMD_CLASS_BIN_ON == onOffState)
  {
    Board_SetLed(SWITCH_STATUS_LED, LED_ON);
    DPRINT("Binary Switch ON\r\n");
  }
}
```

Figure 6: RefreshMMI without any modifications

2. We will be using the function "`Board_SetLed`" but change the GPIO to
   - o `BOARD_RGB1_R`
   - o `BOARD_RGB1_G`
   - o `BOARD_RGB1_B`

3. Call "`Board_SetLed`" 3 times in both the OFF state and in the ON state, as shown in Figure 7.



```
void
RefreshMMI(void)
{
  if (CMD_CLASS_BIN_OFF == onOffState)
  {
    Board_SetLed(BOARD_RGB1_R, LED_OFF);
    Board_SetLed(BOARD_RGB1_G, LED_OFF);
    Board_SetLed(BOARD_RGB1_B, LED_OFF);

    DPRINT("Binary Switch OFF\r\n");
  }
  else if (CMD_CLASS_BIN_ON == onOffState)
  {
    Board_SetLed(BOARD_RGB1_R, LED_ON);
    Board_SetLed(BOARD_RGB1_G, LED_ON);
    Board_SetLed(BOARD_RGB1_B, LED_ON);

    DPRINT("Binary Switch ON\r\n");
  }
}
```

Figure 7: RefreshMMI modified to use RGB LED

Our new modification is now implemented, and you are ready to compile.

The steps to program a device are covered in exercise “3A Compile Switch OnOff and enable debug”, and briefly repeated here:

1. Click on the “*Build*” button to start building the project.
2. When the build finishes, expand the “Binaries” folder and right click on the *.**hex** file to select “*Flash to Device..*”.
3. Select the connected hardware in the pop-up window. The “Flash Programmer” is now prefilled with all needed data, and you are ready to click on “Program”.
4. Click “Program”.

After a short while the programming finishes, and your end device is now flashed with your modified version of Switch On/Off.

3.1.1 Test the functionality

In previous exercises we have already included the device into a secure Z-Wave network using SmartStart. Refer to exercise “Include using SmartStart” for instructions.

> ***Hint*** The internal file system is not erased between reprogramming. This allows a node to stay in a network and keep the same network keys when you reprogram it.
>
> If you need to change e.g. the frequency at which the module operates or the DSK, you need to “Erase” the chip before the new frequency will be written to the internal NVM.

As such, your device is already included in the network.

Test the functionality by verifying you can turn ON and OFF the RGB LED.

- Test the functionality using the “Basic Set ON” and “Basic Set OFF” in the PC Controller. The RGB LED should be turning ON and OFF.
- The RGB LED can also be turned ON and OFF using BTN0 on the hardware.

We have now verified that the modification is working as expected and have successfully changed the GPIO used in a Sample Application.

3.2 Change the RGB color component

In this section, we will be modifying the RGB LED and try to mix the color components.

> *"A color in the RGB color model is described by indicating how much of each of the red, green, and blue is included. The color is expressed as an RGB triplet (r,g,b), each component of which can vary from zero to a defined maximum value. If all the components are at zero the result is black; if all are at maximum, the result is the brightest representable white."*
>
> - *From Wikipedia on RGB Color Model.*

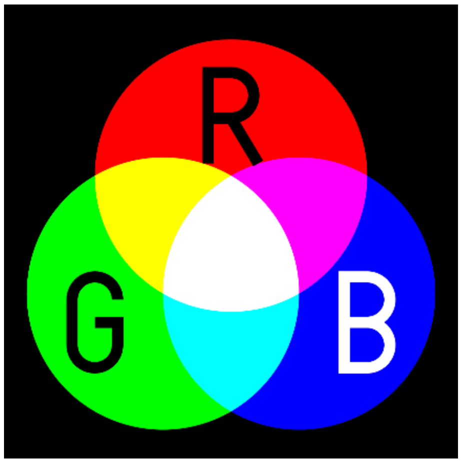
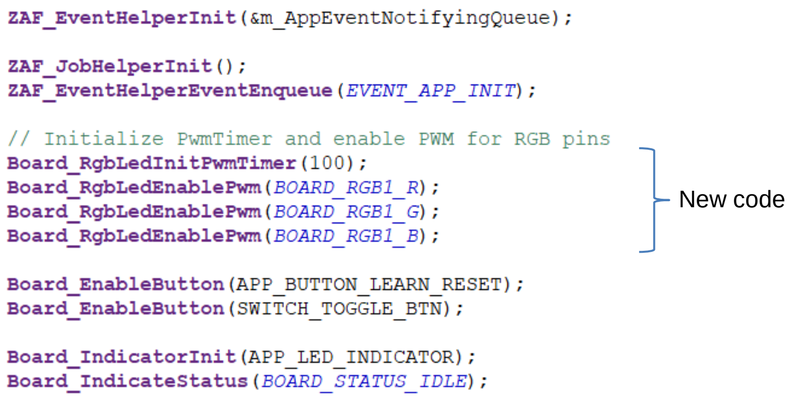


Figure 8: RGB Color Components Mixed Together [Source: Wikipedia]

Since we enabled all color components in previous section the RGB LED is white when ON. By turning on and off of the individual components, we can change the LED. In addition, by adjusting the intensity of each color components, we can make all the colors in between. For that, we will be using PWM to control the GPIOs.

1. In `ApplicationTask()` initialize the PwmTimer and setup the RGB pins to PWM, as shown in Figure 9.



```
ZAF_EventHelperInit(&m_AppEventNotifyingQueue);

ZAF_JobHelperInit();
ZAF_EventHelperEventEnqueue(EVENT_APP_INIT);

// Initialize PwmTimer and enable PWM for RGB pins
Board_RgbLedInitPwmTimer(100);
Board_RgbLedEnablePwm(BOARD_RGB1_R);
Board_RgbLedEnablePwm(BOARD_RGB1_G);
Board_RgbLedEnablePwm(BOARD_RGB1_B);

Board_EnableButton(APP_BUTTON_LEARN_RESET);
Board_EnableButton(SWITCH_TOGGLE_BTN);

Board_IndicatorInit(APP_LED_INDICATOR);
Board_IndicateStatus(BOARD_STATUS_IDLE);
```

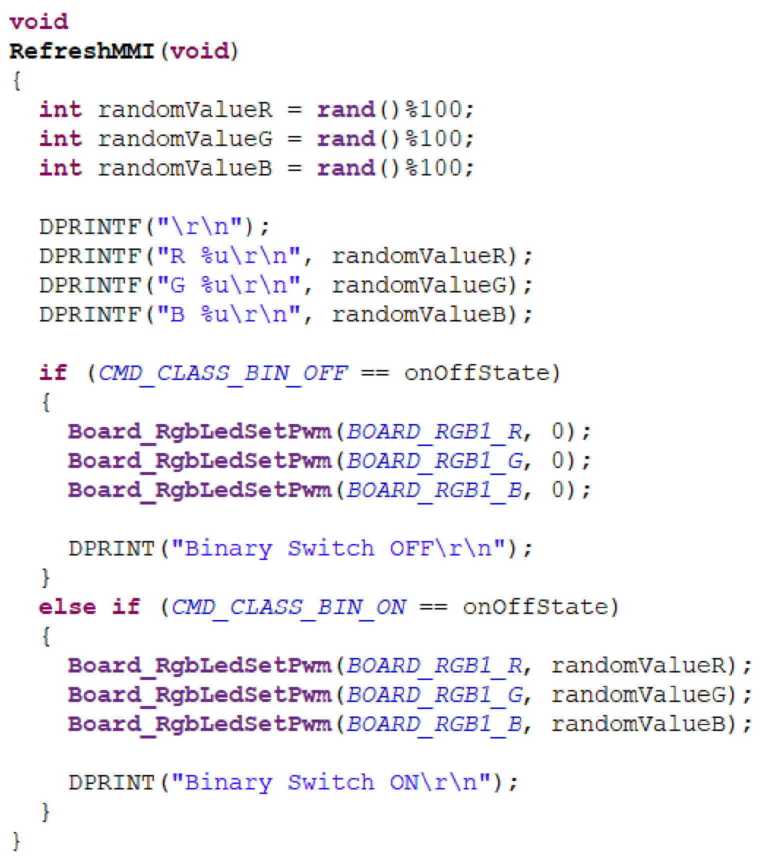
Figure 9: PWM initialized in ApplicationTask()

2. In `RefreshMMI()`, we will be using a random number for every color component. Use `rand()` to get a new value every time the LED is turned ON.

3. Use `DPRINTF()` to write the newly generated value to the serial debug port.

4. Replace `Board_SetLed()` with `Board_RgbLedSetPwm()`, in order to use the random value.

5. Refer to Figure 10 for the updated `RefreshMMI()`.



```
void
RefreshMMI(void)
{
  int randomValueR = rand()%100;
  int randomValueG = rand()%100;
  int randomValueB = rand()%100;

  DPRINTF("\r\n");
  DPRINTF("R %u\r\n", randomValueR);
  DPRINTF("G %u\r\n", randomValueG);
  DPRINTF("B %u\r\n", randomValueB);

  if (CMD_CLASS_BIN_OFF == onOffState)
  {
    Board_RgbLedSetPwm(BOARD_RGB1_R, 0);
    Board_RgbLedSetPwm(BOARD_RGB1_G, 0);
    Board_RgbLedSetPwm(BOARD_RGB1_B, 0);

    DPRINT("Binary Switch OFF\r\n");
  }
  else if (CMD_CLASS_BIN_ON == onOffState)
  {
    Board_RgbLedSetPwm(BOARD_RGB1_R, randomValueR);
    Board_RgbLedSetPwm(BOARD_RGB1_G, randomValueG);
    Board_RgbLedSetPwm(BOARD_RGB1_B, randomValueB);

    DPRINT("Binary Switch ON\r\n");
  }
}
```

Figure 10: RefreshMMI updated with PWM

Our new modification is now implemented, and you are ready to compile.

1. Click on the "*Build*" button to start building the project.
2. When the build finishes, expand the "Binaries" folder and right click on the *.**hex** file to select "*Flash to Device..*".
3. Select the connected hardware in the pop-up window. The "Flash Programmer" is now prefilled with all needed data, and you are ready to click on "Program".
4. Click "Program".

After a short while the programming finishes, and your end device is now flashed with your modified version of Switch On/Off.

3.2.1 Test the Functionality

Test the functionality by verifying you can change the color of the RGB LED.

1. Test the functionality using the "Basic Set ON" in the PC Controller.
2. Click on "Basic Set ON" to see a change in color.

We have now verified that the modification is working as expected and have successfully changed the GPIO to use PWM.

4 Discussion

In this exercise we have modified Switch On/Off from controlling a simple LED to controlling a multi-color LED. Depending on the PWM values, we can now change to any color and intensity.

- Should a “Binary Switch” be used as Device Type for this application?
- Which command classes are better suited for a multi-color LED?

In order to answer the question, you should refer to the Z-Wave specification:

- Z-Wave Plus v2 Device Type Specification
- Z-Wave Application Command Class Specification

This concludes the tutorial in how to modify and change the GPIOs of a Z-Wave Sample Application.