

# READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

## ECEN 2370 Embedded Software Engineering Lab #2 LETIMER and PWM Fall 2020

**Objective:** To learn the embedded software modular encapsulated structure of coding as well as proper documentation of c-functions. You will take the project framework provided and develop several c-functions to pulse the LED using the LETIMER PWM functionality connecting the LETIMER0 peripheral output to LED0 and to LED1. After initializing the system, the PWM will continue to pulse **both LEDs, LED0 and LED1**, at the same time without intervention from the most energy consuming peripheral, the CPU.

Key Learning Objectives from this Assignment:

- **Debugging:** Using ASSERT statements to validate proper programming or operation of the microcontroller. Assertion tests reduce the time, the number of lines of programming code, between functional and non-functional code allowing the programmer to focus on a smaller code segment to debug thus reducing the time to resolve the error.
- **Commenting:** Proper commenting can help plan your code development and aid in code review thus reducing time for project development and obtaining assistance. You should first develop the comment block before you develop your function, and then update/make changes to the function comment block if necessary as you develop your routine.
- **Code Structure, Modularity, and Encapsulation:** Software is developed to be portable and reusable. A software driver is low level code that communicates directly with the hardware such as in this assignment, the LETIMER peripheral. The application developer should not need to understand or be required to program hardware specifics. For example, if you have used a printf statement sending a string to a terminal, you did not communicate directly to the hardware, you used an “interface” (input argument) to communicate to a library, “middleware” (printf), which then communicated to a lower level of code, “the driver” (a serial port) to send the string to the terminal. Moving forward in this course, the application code resides in app.c and the driver will be named after the microcontroller peripheral appended with .c such as letimer.c. Once we add an external device such as an I2C temperature and humidity sensor, you will then develop a module that interfaces between the application code and the driver, middleware.
- **STRUCT and strings:** To pass more than a few arguments to a function, it is easier to define a software STRUCT that contains all the pertinent information. In this manner, only a pointer to the STRUCT is passed via the function’s input argument. In this assignment, in opening a driver, the STRUCT can be analogous to a recipe in cooking.

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

The recipe is the STRUCT and all the ingredients specified in the recipe are the elements of the STRUCT.

**Note:** You will need to download Lab 2 framework from the Canvas Lab 2 folder.

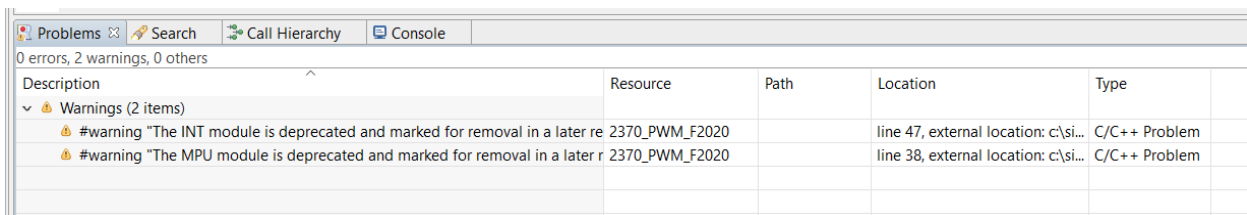
**Due:** Sunday, September 13<sup>th</sup>, at 11:59pm

**Making change to clarify documentation:**

- *Italicized text will signify informational details of the assignment*
- Standard text will signify activities required to complete the assignment

**Lack of best programming practices will result in point deductions starting on this assignment:**

- Magic numbers are not to be used
  - Magic numbers are use of discrete numbers instead of a constant
    - `count = 32768 * period;`
      - 32768 is a magic number
    - `count = COUNT_PER_SECOND * period;`
      - where COUNT\_PER\_SECOND is defined at 32768 in the proper .h header file is the correct implementation
- Warning statements in a build
  - When you build the project and look in the problem tab, all warnings from your code should be resolved. Some warnings can cause your program to fail.
  - The below warnings are acceptable. These warnings are stating that in future versions of the library, these routines will no longer be supported. Since we do not have control of these warnings, they will not be deducted from your grade.



The screenshot shows the 'Problems' window in Visual Studio. It displays two warnings related to deprecated modules. The table below represents the data shown in the screenshot.

Description	Resource	Path	Location	Type
Warning: The INT module is deprecated and marked for removal in a later release.	2370_PWM_F2020		line 47, external location: c:\si...	C/C++ Problem
Warning: The MPU module is deprecated and marked for removal in a later release.	2370_PWM_F2020		line 38, external location: c:\si...	C/C++ Problem

**Instructions:**

1. Download Student\_PWM\_F20.zip from the Canvas Lab 2 folder.
2. Import Student\_PWM\_F20.zip into your Simplicity workspace
3. Change the name of the project by adding your two initials in front of the project name PWM\_Lab.zip.
  - a. Ex. For Keith Graham, the name would change to KG\_PWM\_F20
4. Right click your project and perform a clean project

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

5. Before we go into the project work, you will add a resource to help debug your code.
6. **ASSERTIONS:** *Assertions are statements used to test the validity of the code. A programmer places asserts in sections of the code when they want to validate something or perform error checks. If an assertion fails, it can crash the program to alert the programmer, throw an error, jump to a function and wait in an infinite loop.*

*In our case, Silicon Labs provides a library to handle assertions. Here is a detailed description provided by Silicon Labs:*


*An assert is an error checking module.*

*By default, EMLIB library assert usage **is not included** in order to **reduce footprint and processing overhead**. Further, EMLIB assert usage is decoupled from ISO C assert handling (NDEBUG usage), to allow a user to use ISO C assert without including EMLIB assert statements.*

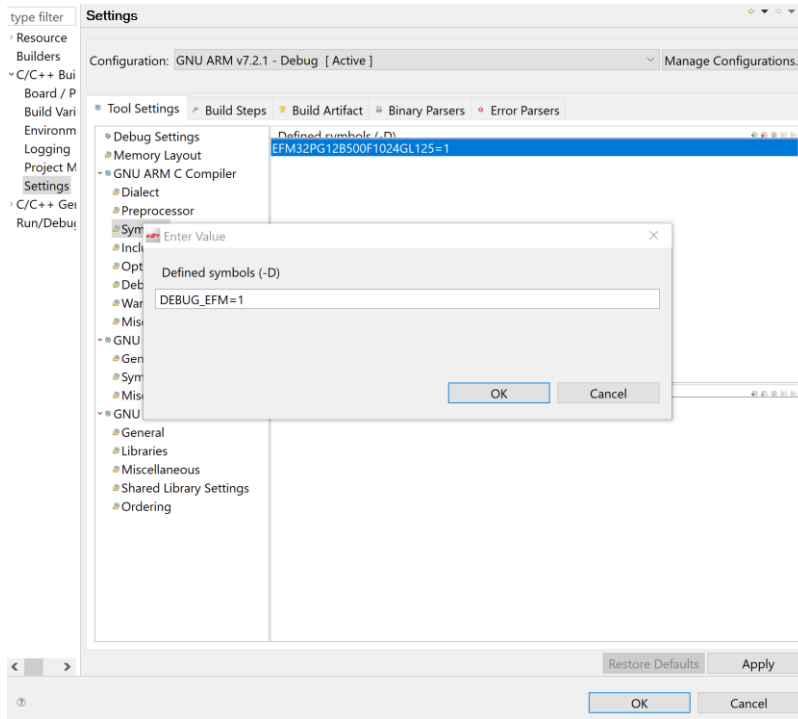
*Below are available defines for controlling EMLIB assert inclusion. The defines are typically defined for a project to be used by the preprocessor.*

- *If `DEBUG_EFM` is defined, the internal EMLIB library assert handling will be used. This is implemented as a simple `while(true)` loop. `DEBUG_EFM` is not defined by default.*
- *If `DEBUG_EFM_USER` is defined instead, the user must provide their own implementation of the `assertEFM()` function.*
- *If both `DEBUG_EFM` and `DEBUG_EFM_USER` are undefined then all `EFM_ASSERT()` statements are no operation.*

*From the description provided, we understand that there are two ways to use asserts, either by using a Silicon Labs function or our own function.*

7. To enable the `EFM_ASSERT` resource, a Symbol must be added to enable them while the code is built and ran. To add a symbol, perform the following:
  - a. Right click your project, select properties, expand C/C++ Building, click on settings, expand GNU ARM C Compiler, and then click on symbols. Click on  to add `DEBUG_EFM=1`. Then click OK, then Apply, and then OK.
    - i. Do not delete the existing symbols
    - ii. Do not add spaces

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN



Symbols are used by the compiler to enable or disable certain functionalities or sections of code. Silicon Labs disables asserts by default because they want to lower the processing overhead.

Your project now has `DEBUG_EFM` defined, which means we will be using the provided assert function.

### **8. Now we understand what asserts do, but how and why do we use them?**

*When writing firmware, you want to catch bugs early and quickly, and asserts will help you do just that.*

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

Here is an example, have a look at this simple code:

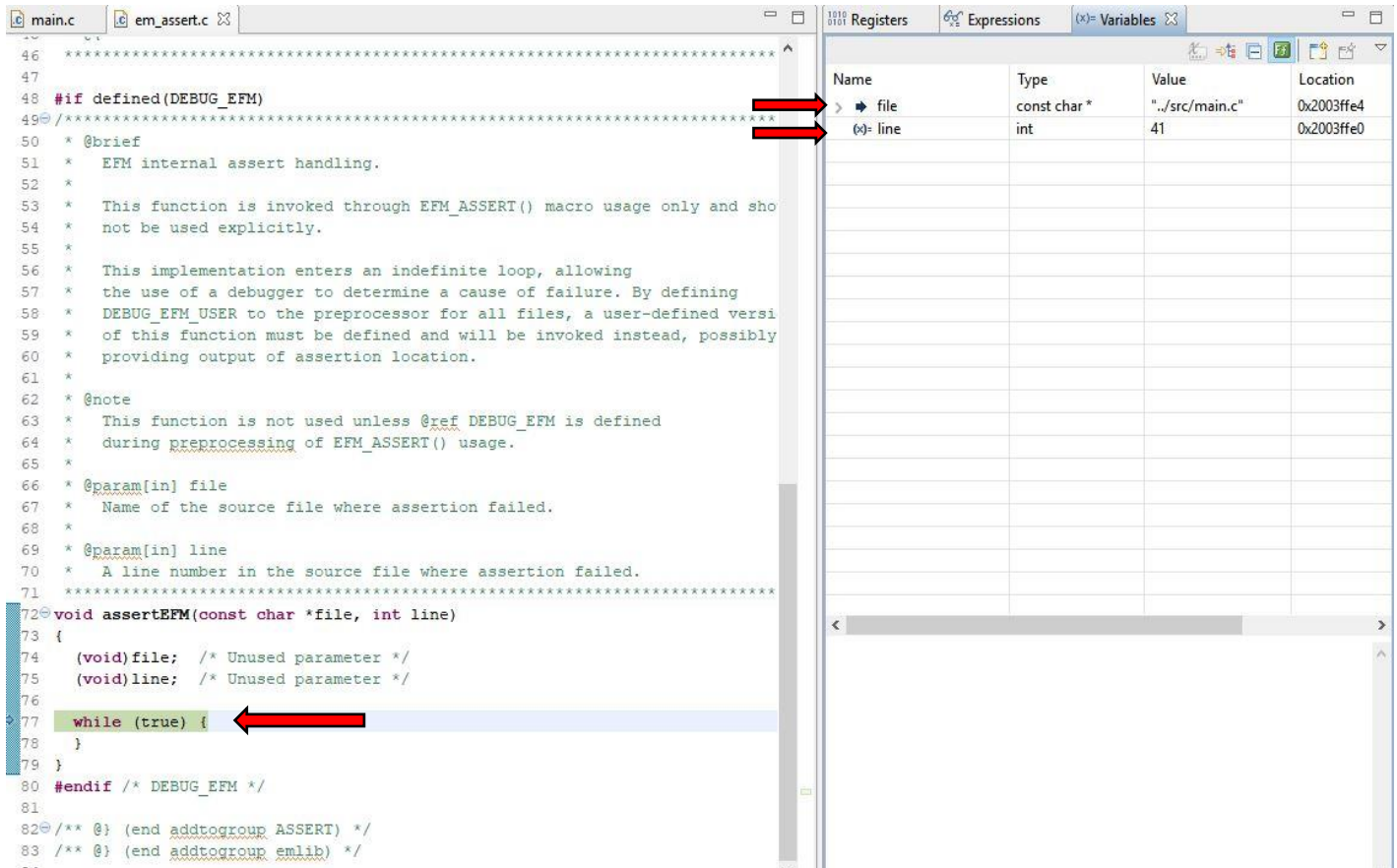
```
26 int main(void)
27 {
28     /* Chip errata */
29     CHIP_Init();
30
31     //Init functions etc...
32
33     /* For this simple example, lets just define some test variables: */
34     uint8_t SOMETHING_I_EXPECT = 0;
35     uint8_t SOMETHING_I_GOT = 0;
36
37     /* Maybe at this point you want to check that a critical part of the code
38      * has successfully executed and updated the value of something.
39      * This is what you would do: */
40
41     EFM_ASSERT(SOMETHING_I_EXPECT == SOMETHING_I_GOT);
42
43     /* If the assert passes, we will get to this line and execute the while(1) loop.
44      * However, if it fails, we will jump to a function in em_assert.c and be stuck in
45      * a while loop. Using the debugger we can view at exactly what file and line the
46      * assertion failed.
47      */
48
49     while (1)
50     {
51         //Do work
52     }
53 }
```

The example above will work with no issues, as `SOMETHING_I_EXPECT` is in fact equal to `SOMETHING_I_GOT`.

*Note: You must add the `#include "em_assert.h"` to all `.c` files that are using the `EFM_ASSERT()` function. Best practice is to add the `"em_assert.h"` to the appropriate module's `.h` file.*

Now, let's change the example so the assertion fails, by setting `SOMETHING_I_GOT` to 1. As the example above explains, we will be stuck in a while loop in a function in the `em_assert.c` file (which you can find under the "emlib" folder in your project):

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN



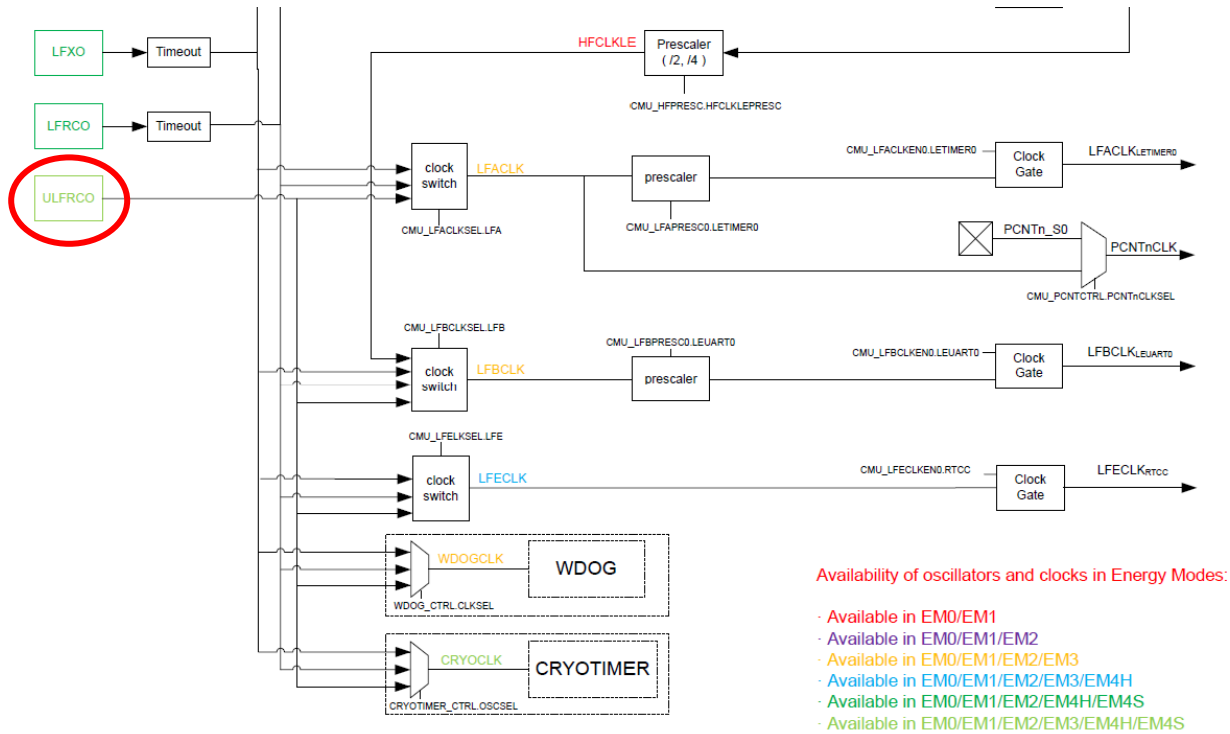
That function (`assertEFM`) helps us get important information, which are the file name and line number where the assertion failed.

You can see why using assertions is extremely helpful, especially as your project grows and you need rigorous error checks at different points.

9. **SETTING UP YOUR PERIPHERALS:** First, let's look at the `cmu.c` file. This `.c` folder is dedicated to set up the clock tree for the project. With adding a new peripheral to the project, `LETIMER0`, you will need to configure the clock tree. Establishing the clock tree should be in the `cmu_open()` function found in the `cmu.c` file located in your project under `/src/Source_Files`. Enabling the clock to the peripheral will be done in the `open_function` of that particular driver/peripheral.

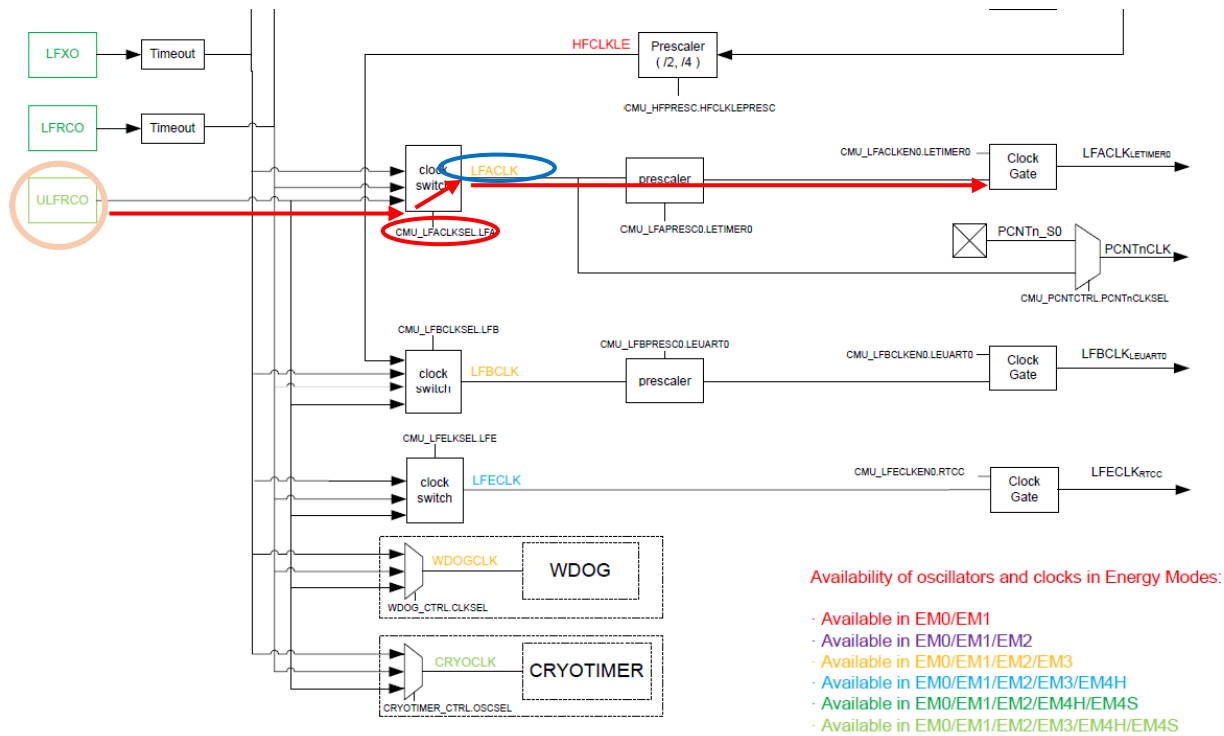
- a. For the `LETIMER0`, we will use the `ULFRCO` oscillator to enable operation down into the Pearl Gecko Energy Mode 3 level.
  - i. Normally, you would be required to enable the oscillator for a clock tree, but by specifying to use the `ULFRCO` for the `LETIMER0`, there is no requirement to enable the `ULFRCO` oscillator since the `ULFRCO` is always enabled in all Energy Modes.

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN



- ii. If one of the other oscillators were required, you would first need to turn on the oscillator using the following emlib routine
  1. [CMU\\_OscillatorEnable\(\)](#)
- iii. In building the clock tree, the next step is to route the oscillator to the desired clock branch. This routing is done through selecting the proper input of the clock switch mux.
- iv. The setting of the clock switch branch mux select lines, CMU\_LFACKSEL\_LFA, for the LETIMER0 is performed using the following emlib routine
  1. CMU\_ClockSelectSet()
  2. Using the Silicon Labs Hardware Abstraction Layer, HAL, document, what clock branch “definition” would you use to specify the LFACLK branch?

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN



- v. You must also enable the Low Energy (LE) clock tree to the Low Energy peripherals
  1. **CMU\_ClockEnable**(cmuClock\_CORELE, true);
  2. This function acts as a global enable to the Low Energy Clock branches
- vi. For this assignment, I have commented out the lines of code to initialize the oscillators and clock tree settings for the LETIMER0. You must do the following:
  1. Uncomment the lines of code in **cmu.c** to enable the oscillators and set the clock tree. You can uncomment the lines by selecting the lines of code and while the lines of code are highlighted, click **ctrl+/-**
  2. Replace the XXXXX with the correct symbol/enumeration that can be found in the Pearl Gecko HAL documentation for the function call and requirement of the project.
  3. HAL documentation can be found at: <https://docs.silabs.com/mcu/latest/efm32pg12/>

10. READ THIS ENTIRE ASSIGNMENT BEFORE YOU BEGIN THE LAB

11. DO NOT WRITE ANY CODE IN THESE FUNCTIONS UNTIL YOU GET TO SECTIONS OF THIS LAB SPECIFICALLY FOR THESE FUNCTIONS!

a. `app_letimer_pwm_open()` function



## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

- b. `letimer_pwm_open()` function
- c. `app_peripheral_setup()` function

### 12. You will now change the frequency of operation of the Pearl Gecko to 32 MHz

- a. HAL documentation can be found at:  
<https://docs.silabs.com/mcu/latest/efm32pg12/>
- b. The lab framework has set the Pearl Gecko CPU clock frequency by default to 1 MHz. You will find the following Silicon Lab enumeration in `brd_config.h` that is used to configure the High Frequency Resistor Capacitor Oscillator (HFRCO) to 1 MHz
  - i. `cmuHFRCOFreq_1M0Hz`
- c. For this assignment, you will need to change the Pearl Gecko CPU frequency to 32MHz.
  - i. Generally, Silicon Labs creates enumerations (enum) TypeDefs for key parameters that are used to configure the microcontroller. The type def is can be reversed engineered by the following:
    - 1. `cmuHFRCOFreq_1M0HZ`
      - a. `cmu = CMU_ (name of the peripheral)`
      - b. `HFRCOFreq_ (specific name of enum's TypeDef)`
      - c. TypeDef
    - 2. Concatenating the above, gives you the full enum TypeDef of:
      - a. `CMU_HFRCOFreq_TypeDef`
  - ii. Now, go to the HAL documentation per the link above and type this enumeration TypeDef in the HAL's documentation search window and hit enter (or return)

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

The top screenshot shows the Silicon Labs documentation website. The search bar contains the text "CMU\_HFRCCOFreq". The search results show "EFM32PG12 Gecko MCU Documentation". The page includes a sidebar with "EFM32 Pearl Gecko 12" and "More Documentation". The main content area shows "You are viewing documentation for version: 5.9 (latest)" and "Release Notes | Downloads". Below this, it states "This API reference guide covers MCU peripherals, middleware Silicon Labs Radio Abstraction Interface Layer (RAIL) and win". It also lists "MCU Peripherals, Middleware, and Board Support" with a bulleted list of links: "CMSIS-CORE Device headers for the EFM32PG12 Gecko MCL", "EMLIB Peripheral Library", "EnergyAware Driver Library", "Platform Middleware", "Board Support Package", and "Kit Driver Library".

The bottom screenshot shows the same website with the search bar containing "search". The search results for "CMU\_HFRCCOFreq" in "/mcu/latest/efm32pg12" are displayed. The results include "CMU - v5.9 - MCU EFM32PG12 API Documentation" and "Deprecated List v5.9 - MCU EFM32PG12 API Documentation". A red arrow points to the "CMU - v5.9 - MCU EFM32PG12 API Documentation" link.

- iii. If you have a selection where one is “Deprecated,” do not select the deprecated option. Deprecated indicates that this usage is planned to become obsoleted sometime in the future
1. In the above example, click on CMU- v5.9

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

The screenshot shows the Silicon Labs EMLIB documentation for the CMU module. The top navigation bar includes the Silicon Labs logo and the URL docs.silabs.com. A search bar is located below the logo. The left sidebar lists various modules under the heading 'EFM32 Pearl Gecko 12', including CHIP, CMU, COMMON, CORE, CRYOTIMER, CRYPTO, CSEN, DBG, EMU, GPCRC, and GPIO. The CMU module is selected, and its detailed description is shown on the right. The description states that the CMU is the clock management unit (CMU) Peripheral API and contains functions for the CMU peripheral of Silicon Labs 32-bit MCUs and SoCs. Below the description, the 'Data Structures' section lists three structs: CMU\_DPLLInit\_TypeDef, CMU\_HFX0Init\_TypeDef, and CMU\_LFX0Init\_TypeDef.

SILICON LABS  
docs.silabs.com

search

EFM32 Pearl Gecko 12

CHIP

CMU

CMU\_DPLLInit\_TypeDef

CMU\_HFX0Init\_TypeDef

CMU\_LFX0Init\_TypeDef

COMMON

CORE

CRYOTIMER

CRYPTO

CSEN

DBG

EMU

GPCRC

GPIO

You are viewing documentation for version: 5.9 (latest) | 5.8 | Version History

### CMU

EMLIB

#### Detailed Description

Clock management unit (CMU) Peripheral API.

This module contains functions for the CMU peripheral of Silicon Labs 32-bit MCUs and SoCs. The CMU contains clock multiplexers, pre-scalers, calibration modules and wait-states.

#### Data Structures

struct CMU\_DPLLInit\_TypeDef

struct CMU\_HFX0Init\_TypeDef

struct CMU\_LFX0Init\_TypeDef

- iv. You will be taken to the CMU module of the HAL documentation. Perform a `ctrl+f` to open a search window in your browser and search this webpage for your enum TypeDef, **CMU\_HFRCOFreq\_TypeDef**
- v. You should now see the list of different frequencies that can be defined for the HFRCO oscillator
  1. Copy the enumeration for 32 MHz and replace the 1MHz enumeration to define the HFRCO frequency in `brd_config.h`
    - a. `#define MCU_HFXO_FREQ "32MHz enum"`
  2. Now, the next time you compile, your project will configure the HFRCO to 32 MHz instead of 1 MHz

13. Now, open up the **letimer.h** file. Complete the following define statement. In the `cmu.c` file, you set up the ULFRCO oscillator to the LETIMER0 clock tree. Replace **XX** with the number of HZ when using the ULFRCO oscillator.

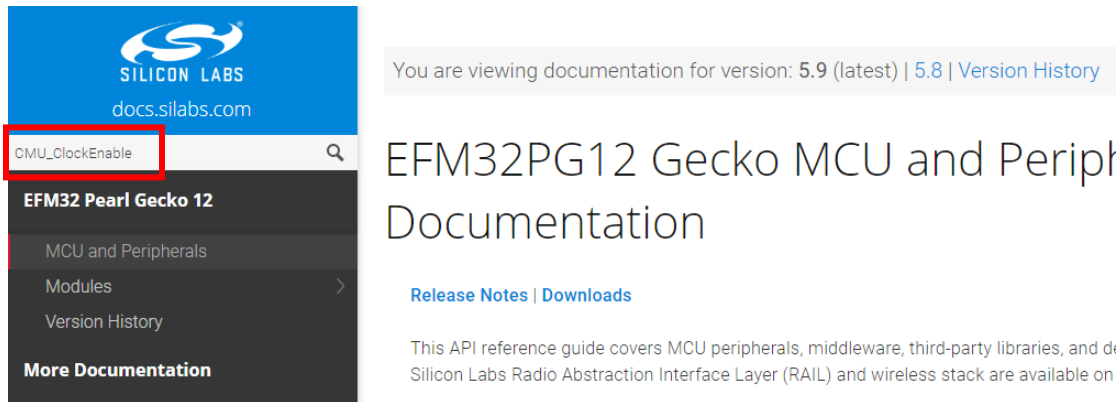
- i. `#define LETIMER_HZ XX`
- ii. You can locate this information in the Pearl Gecko Reference Manual

14. The **letimer.c/.h** files are designed to be modular and encapsulated. The function **xxx\_open.c** are meant to be drivers to open and initialize the letimer peripheral as a pwm peripheral and for routing its output signals to gpio pins.

- a. **SECTION 14 DESCRIBES THE GENERAL FLOW OF DEVELOPING A PERIPHERAL DRIVER. DO NOT ENTER ANY CODE UNTIL YOU GET TO THE NEXT SECTION. THIS SECTION IS FOR REFERENCE ONLY.**

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

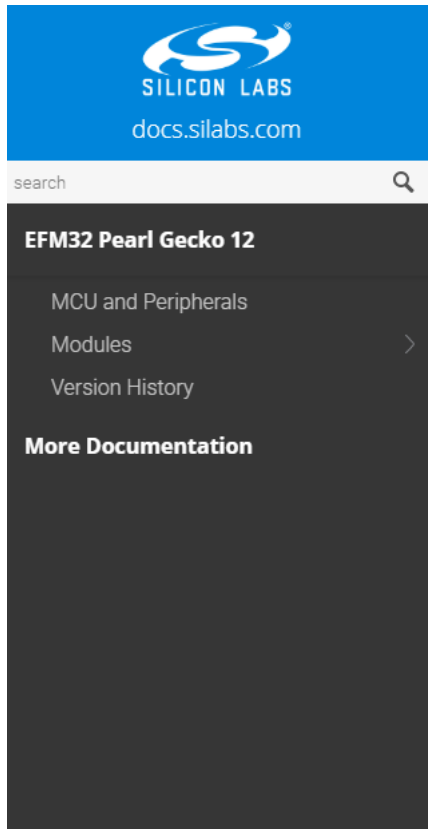
- b. With the Pearl Gecko being a low energy microcontroller, all clocks to a peripheral upon reset are disabled.
- c. The first operation in each **xxx\_open.c** must enable the clock to the peripheral. Without a clock to the peripheral, you cannot write to the peripheral. You can use the following instruction to enable the clock to the peripheral.
  - i. **CMU\_ClockEnable**(You can find the enumeration in the HAL doc, true);
  - ii. Let's take a look at how to find the enumeration to be used to enable the clock for the peripheral. Open up the HAL documentation per the earlier link. Now search for **CMU\_ClockEnable**.



The screenshot shows the Silicon Labs documentation website. The header includes the Silicon Labs logo and the URL docs.silabs.com. A search bar contains the text "CMU\_ClockEnable". Below the search bar, a sidebar menu lists "EFM32 Pearl Gecko 12", "MCU and Peripherals", "Modules", "Version History", and "More Documentation". The main content area displays "EFM32PG12 Gecko MCU and Periph Documentation". A version notice at the top right states "You are viewing documentation for version: 5.9 (latest) | 5.8 | Version History". Below the title, there are links for "Release Notes" and "Downloads". A paragraph at the bottom states: "This API reference guide covers MCU peripherals, middleware, third-party libraries, and the Silicon Labs Radio Abstraction Interface Layer (RAIL) and wireless stack are available on".

- iii. You will want to click on the **CMU-V5.9** selection below because you have been search for a **Clock Management Unit**, **CMU\_ClockEnable()**, function

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN



## EFM32 Pearl Gecko 12

Search results for "**CMU\_ClockEnable**" in **"/mcu/latest/efm32pg12"**

### BUS - v5.9 - MCU EFM32PG12 API Documentation

[/mcu/latest/efm32pg12/group-BUS](#)

[CMU\\_ClockEnable\(\)](#), [CMU\\_ClockSelectSet\(\)](#), [CMU\\_HFXOInit\(\)](#), [CMU\\_LFXOInit\(\)](#), [CMU\\_PCNTClockExternalSet\(\)](#), [CRYOTIMER\\_EM4WakeupEnable\(\)](#), [CRYOTIMER\\_ECRYPTO\\_KeyBufWrite\(\)](#), [CSEN\\_Disable\(\)](#), [CSEN\\_Enable\(\)](#), [EMU...](#)

### SLEEPTIMER - v5.9 - MCU EFM32PG12 API Documentation

[/mcu/latest/efm32pg12/group-SLEEPTIMER](#)

[cmuSelect\\_LFRCO](#)); [CMU\\_ClockEnable](#) ( [cmuClock\\_RTCC](#) , true ); status = [sl\\_sleep](#) SL\_STATUS\_OK ) { printf( "Sleeptimer init error.\r\n" ); } status...

### CMU - v5.9 - MCU EFM32PG12 API Documentation

[/mcu/latest/efm32pg12/group-CMU](#)

and [UDELAY\\_Calibrate\(\)](#) . void [CMU\\_ClockEnable](#) ( [CMU\\_Clock\\_TypeDef](#) clock, bool enable ) Enable/disable a clock. In general, module clocking is disabled after a reset....

- iv. Once in the clock management unit, do a search on the web page for [CMU\\_ClockEnable\(\)](#) which will take you to the function's Function Prototype and click on it



- v. You will now be taken to the Doxygen comment information for the [CMU\\_ClockEnable\(\)](#) function. The first input argument, [CMU\\_Clock\\_TypeDef](#) defines what clock element to enable. We will need to click on the hyperlink [CMU\\_Clock\\_TypeDef](#) to obtain the list of clock elements

# READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

**void CMU\_ClockEnable** ( CMU\_Clock\_TypeDef clock, bool enable )

Enable/disable a clock.

In general, module clocking is disabled after a reset. If a module clock is disabled, the registers of that module are not accessible and reading from such registers may return undefined values. Writing to registers of clock-disabled modules has no effect. Avoid accessing module registers of a module with a disabled clock.

**Note**

If enabling/disabling an LF clock, synchronization into the low-frequency domain is required. If the same register is modified before a previous update has completed, this function will stall until the previous synchronization has completed. See [CMU\\_FreezeEnable\(\)](#) for a suggestion on how to reduce the stalling time in some use cases.

HFCLKLE prescaler is automatically modified when peripherals with clock domain HFBUSCLK is chosen based on the maximum HFLE frequency allowed.

**Parameters**

[in]	clock	The clock to enable/disable. Notice that not all defined clock points have separate enable/disable control. See the CMU overview in the reference manual.
[in]	enable	<ul style="list-style-type: none"> <li>true - enable specified clock.</li> <li>false - disable specified clock.</li> </ul>

- vi. Scroll through the enumerations defined within [CMU\\_Clock\\_TypeDef](#) until you find one that would be appropriate to enable the LETIMER0. As an example, [cmuClock\\_ADC0](#) would enable the clock to the Pearl Gecko's ADC0 peripheral
1. To enable the clock to the ADC0 peripheral, you would use
    - a. `CMU_ClockEnable(cmuClock_ADC0, true);`
  2. For this assignment, you will need to locate the enumeration to enable the clock to the LETIMER0 peripheral
    - a. What enumeration enables the clock to LETIMER0?

cmuClock_ACMP1	Analog comparator 1 clock
cmuClock_VDAC0	Voltage digital-to-analog converter 0 clock
cmuClock_IDAC0	Current digital-to-analog converter 0 clock
cmuClock_ADC0	Analog-to-digital converter 0 clock
cmuClock_I2C0	I2C 0 clock
cmuClock_I2C1	I2C 1 clock
cmuClock_CSEN_HF	Capacitive Sense HF clock

- d. Next, you must initialize the peripheral using the data being passed to it via the **STRUCT** pointer argument to the **xxx\_open.c** functions.

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

- i. These STRUCTs can be found in the **letimer.h** files
- e. Since these opens are dedicated to PWM functionality, the structures only includes variables that need to be changed to generically implement the PWM function. The variables that are constantly set for PWM operation are not included in the input structure argument and should be set within the **letimer\_pwm\_open.c** function to enable proper pwm operation.
  - i. Information on how to implement the initialization of each peripheral can be found in the Pearl Gecko Reference manual
    - 1. LETIMER chapter
  - ii. Let's take a look at the flow of how a peripheral is enabled in the screen print below
    - 1. Do not implement the **letimer\_pwm\_open()** function at this time. Wait until you reach the section labled as **letimer\_pwm\_open()** function to develop the routine.
  - iii. Most of the Silicon Labs' peripherals have a TypeDef structure that is used to initialize the peripheral. You want to define a local version of this TypeDef to store the values that will be used to initial the peripheral per the input arguments or application specifics such as in this case a free running PWM. Please refer to the **GREEN ARROW**.

```
void letimer_pwm_open(LETIMER_TypeDef *letimer, APP_LETIMER_PWM_TypeDef *app_letimer_struct){
    /* Declare local variables */
    LETIMER_Init_TypeDef letimer_pwm_values;

    /* Enable the clock to the LETIMER0 peripheral */

    /* Use EFM_ASSERT statements to verify whether the LETIMER clock tree properly configured
    * and enabled */

    // Initialize letimer values for PWM operation
    letimer_pwm_values.bufTop = xxx; // what value should xxx be?
    letimer_pwm_values.comp0Top = xxx; // what value should xxx be?
    letimer_pwm_values.debugRun = app_letimer_struct->yyy; // what yyy variable from the input arg should be used?
    letimer_pwm_values.enable = app_letimer_struct->yyy; // what yyy variable from the input arg should be used?
    letimer_pwm_values.out0Pol = xxx; // what value should xxx be?
    letimer_pwm_values.out1Pol = xxx; // what value should xxx be?
    letimer_pwm_values.repMode = zzz; // What zzz value, enumeration should be used from HAL documentation?
    letimer_pwm_values.ufoa0 = zzz; // What zzz value, enumeration should be used from HAL documentation?
    letimer_pwm_values.ufoa1 = zzz; // What zzz value, enumeration should be used from HAL documentation?

    LETIMER_Init(letimer, &letimer_pwm_values); // Initialize letimer

    // Initialize COMP0 and COMP1

    // Set the REP0 mode bits for PWM
    // use the values from app_letimer_struct input argument for ROUTELOC and ROUTEPEN enable

    // We are not enabling any interrupts at this time. If you were, you would do it now

    // We will not enable the LETIMER0 at this time
}
```

- iv. The **RED ARROWS** are comments that will help you understand the flow of initializing the peripheral. You should input the code to implement the statement after the comment.



## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

- v. The **BLUE ARROW** points to where the local `TypeDef` variable is initialized by the application specifics of the assignment and the input arguments from the input argument structure `app_letimer_struct`.
- vi. Other than the `LETIMER_INIT()` function, what else must be programmed or initialized in the `LETIMER` peripheral before it is enabled?
  - 1. For example, using the `HAL` documentation, what library functions would you use to store the values into the `COMP0` and `COMP1` registers?
- f. Let's review `STRUCTS` and how to access information on them and utilize them in `Simplicity`
  - i. `STRUCTS` are very useful in declaring and using variables that are associated with each other such as all the variables used to initialize a microcontroller peripheral. With the variables being grouped together in memory, a single pointer can be used to send the information contained in the `STRUCT` as a function argument.
    - 1. A **struct** in the [C programming language](#) (and many derivatives) is a [composite data type](#) (or [record](#)) declaration that defines a physically grouped list of variables under one name in a block of memory, allowing the different variables to be accessed via a single [pointer](#) or by the struct declared name which returns the same address. (Wikipedia)
  - ii. For any variable, you can go directly to where it is declared even if it is in a different file by selecting the variable and pressing Function F3. To learn more about the `LETIMER_Init_TypeDef` `STRUCT`, select `LETIMER_Init_TypeDef` indicated by the **GREEN ARROW** above and press Function F3. `Simplicity` will open where `LETIMER_Init_TypeDef` is declared in the file `em_letimer.h`. You may need to scroll up to see the complete declaration.

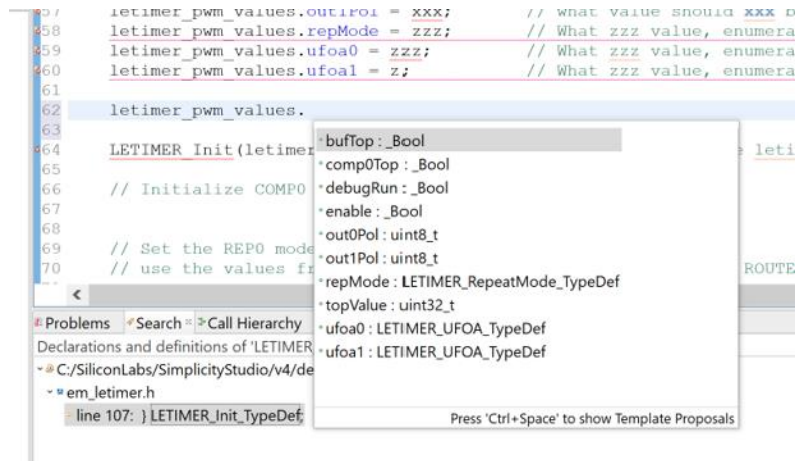
```
/** LETIMER initialization structure. */
typedef struct {
    bool enable;           /**< Start counting when initialization completes. */
    bool debugRun;         /**< Counter shall keep running during debug halt. */
#ifdef LETIMER_CTRL_RTCCOTEN
    bool rtcComp0Enable;   /**< Start counting on RTC COMP0 match. */
    bool rtcComp1Enable;   /**< Start counting on RTC COMP1 match. */
#endif
    bool comp0Top;         /**< Load COMP0 register into CNT when counter underflows. */
    bool bufTop;           /**< Load COMP1 into COMP0 when REP0 reaches 0. */
    uint8_t out0Pol;        /**< Idle value for output 0. */
    uint8_t out1Pol;        /**< Idle value for output 1. */
    LETIMER_UFOA_TypeDef ufoa0; /**< Underflow output 0 action. */
    LETIMER_UFOA_TypeDef ufoa1; /**< Underflow output 1 action. */
    LETIMER_RepeatMode_TypeDef repMode; /**< Repeat mode. */
    uint32_t topValue;      /**< Top value. Counter wraps when top value matches counter */
} LETIMER_Init_TypeDef;
```

- iii. You can see that in the example code below, a `STRUCT` is being accessed in two methods:
  - 1. `letimer_pwm_values.debugRun = app_letimer_struct->xxx`

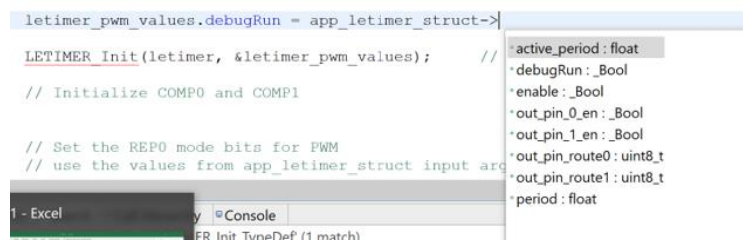


## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

- iv. To access STRUCTs within the space that they are defined such as a local STRUCT in this example, all the STRUCT variables can be accessed by STRUCT name, period, STRUCT element name.
  - 1. letimer\_pwm\_values.debugRun
- v. A nice feature in Simplicity is for all local STRUCTs, after you complete the STRUCT name and period, if you wait a second or two, it will list all the possible variables of the STRUCT to select to complete defining the STRUCT element that you would like to access



- vi. The second structure being accessed in our example is a STRUCT that is being passed to the function via a pointer. With this STRUCT, you will use the second method of accessing a STRUCT due to the use of being passed a pointer
  - 1. STRUCT name, `->`, STRUCT element name
    - a. STRUCT name is the base pointer to the STRUCT
    - b. `->` indicates add STRUCT address offset of the STRUCT element
- vii. Similar to accessing a local STRUCT, if you complete the STRUCT name plus `->` and wait a second or two, Simplicity will bring up all the elements of the STRUCT for you to select



- g. The **LETIMER xxx\_open.c** function should not enable the timer. You will want to enable, turn-on, the LETIMER after all modules and peripherals have been initialized in the `app.c` function `app_peripheral_setup(void)`.

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

- i. What function in the HAL documentation can be used to enable the LETIMER0 before your program enters the while(1) loop in main.c?
  1. What would be the c-code to enable LETIMER0 if writing directly to the LETIMER CMD register?
- ii. Check out the code in `app_peripheral_setup()` at the end of the function. Why is the call to start the LETIMER not what is defined in the HAL documentation?
  1. For the code to be modular and encapsulated, you want all accesses to the peripheral registers to be done by your driver .c code, `letimer.c`
  2. What have you been using that models this programming concept?
    - a. The `em_lib` routines in the HAL documentation
    - b. You will not find another module such as the ADC module manipulate a register in another peripheral like the LETIMER.
  3. What is the `em_lib` function to enable or start the letimer peripheral?
  4. With the `em_lib` function that you found in the HAL documentation to enable or start the LETIMER, complete the `letimer_start()` function found in `letimer.c`

### 15. `app_letimer_pwm_open()` function:

- a. The project is structured to have a driver, `letimer_pwm_open()`, which will configure the LETIMER0 peripheral for PWM operation.
- b. For this assignment, the PWM function is to have a period of 2.7 seconds with an active time, on-time of the LED, of 0.15 seconds. I have included the relevant required application information in the `app.h` file for this lab. **Both LED0 and LED1 will turn-on and turn-off in unison.**

```
21 //*****
22 // defined files
23 //*****
24 #define      PWM_PER      2.7      // PWM period in seconds
25 #define      PWM_ACT_PER  0.15     // PWM active period in seconds
26
27
```

- c. The application requirements for the `letimer_pwm_open()` function can be found by looking at the function prototype of the `letimer_pwm_open()` found in `letimer.h`
  - i. `void letimer_pwm_open(LETIMER_TypeDef *letimer, APP_LETIMER_PWM_TypeDef *app_letimer_struct);`
- d. With the main objective of this function to send the application specific requirements to the `letimer_pwm_open()` function, the first line of code must be the declaration of a local `APP_LETIMER_PWM_TypeDef STRUCT`

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

- i. `APP_LETIMER_PWM_TypeDef` struct name of your choice;
- e. With a local STRUCT now declared, the next step is to configure the value of each “struct of your choice” elements that will be sent to `letimer_pwm_open()` to satisfy the application requirements
  - i. “struct name of your choice”.element1 = xxx;
  - ii. “struct name of your choice”.element2 = xxx;
  - iii. “struct name of your choice”.elementn = xxx;
- f. Please note that in the process of configuring / initializing the LETIMERO peripheral, you will be using two different STRUCTs. The first STRUCT is the `APP_LETIMER_PWM_TypeDef` that you just initialized. This STRUCT contains additional information to complete the configuration of the LETIMERO peripheral than what is included in the `LETIMER_Init_TypeDef`. For example, the `period` and `active_period` elements in `APP_LETIMER_PWM_TypeDef` are not required for the `LETIMER_Init_TypeDef`, but are required to initialize the values of the LETIMERO COMP0 and COMP1 registers.
- g. With the STRUCT to initialize your driver now configured, you have one of the input arguments to `letimer_pwm_open()` ready
- h. *The next, first, argument in `letimer_pwm_open()` is `LETIMER_TypeDef *letimer`. If you select `LETIMER_TypeDef` and press Function F3, you will see its declaration*

```
* @brief EFM32PG12B_LETIMER Register Declaration
*****
/** LETIMER Register Declaration */
typedef struct {
    __IOM uint32_t CTRL;          /**< Control Register */
    __IOM uint32_t CMD;           /**< Command Register */
    __IM uint32_t STATUS;         /**< Status Register */
    __IOM uint32_t CNT;           /**< Counter Value Register */
    __IOM uint32_t COMP0;         /**< Compare Value Register 0 */
    __IOM uint32_t COMP1;         /**< Compare Value Register 1 */
    __IOM uint32_t REP0;          /**< Repeat Counter Register 0 */
    __IOM uint32_t REP1;          /**< Repeat Counter Register 1 */
    __IM uint32_t IF;             /**< Interrupt Flag Register */
    __IOM uint32_t IFS;           /**< Interrupt Flag Set Register */
    __IOM uint32_t IFC;           /**< Interrupt Flag Clear Register */
    __IOM uint32_t IEN;           /**< Interrupt Enable Register */

    uint32_t RESERVED0[1U];       /**< Reserved for future use */
    __IM uint32_t SYNCBUSY;        /**< Synchronization Busy Register */

    uint32_t RESERVED1[2U];       /**< Reserved for future use */
    __IOM uint32_t ROUTEPEN;       /**< I/O Routing Pin Enable Register */
    __IOM uint32_t ROUTELOC0;      /**< I/O Routing Location Register */

    uint32_t RESERVED2[2U];       /**< Reserved for future use */
    __IOM uint32_t PRSSEL;         /**< PRS Input Select Register */
} LETIMER_TypeDef;               /** @*/
```

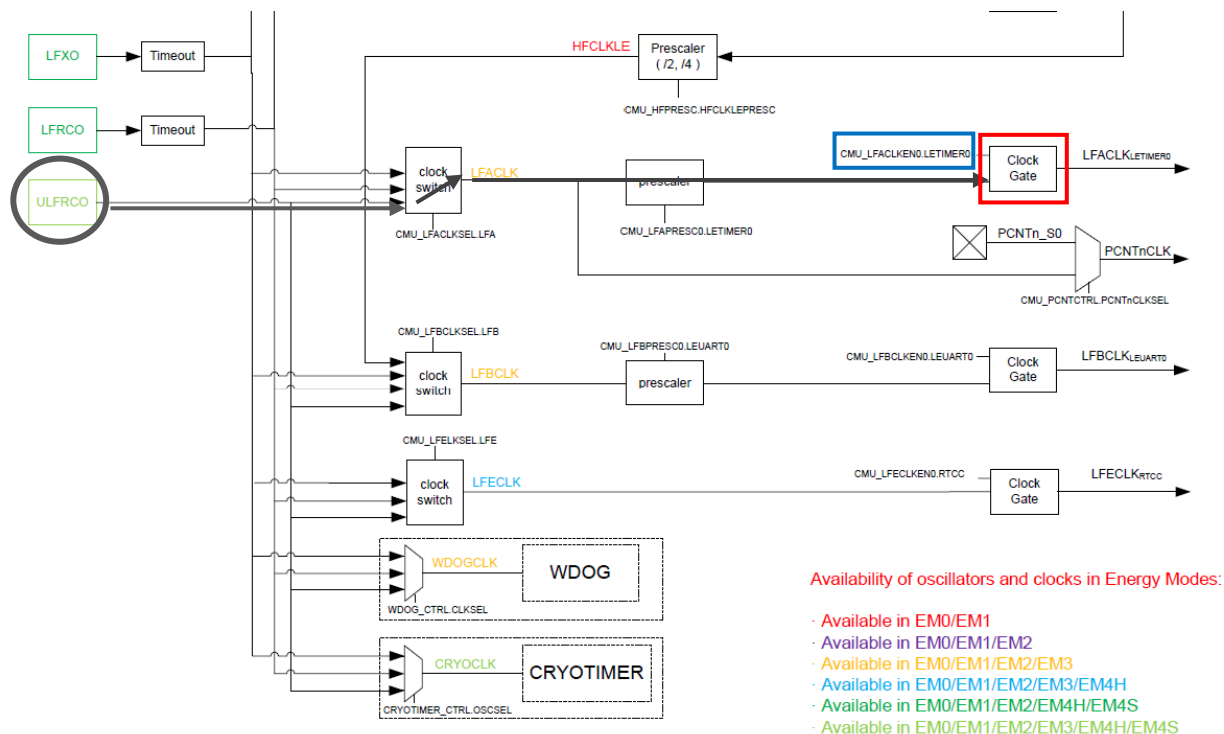
- i. *The `LETIMER_TypeDef` STRUCT defines the registers/elements for each of the registers in the LETIMER peripheral which also provides the offset address to access these registers from the base address of the STRUCT*
- j. *If you search for the LETIMERO declaration, you will see the following:*
  - i. `#define LETIMERO ((LETIMER_TypeDef *) LETIMERO_BASE) /**< LETIMERO base pointer */`

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

- ii. The comment of this define statement clarifies that `LETIMER0` is the base address that will be used to access this `STRUCT` which implies to inform the driver of the `LETIMER` peripheral, all you need to do is use the `LETIMER0` defined constant for the argument `LETIMER_TypeDef *letimer`
- k. Now, you have both arguments to open the letimer pwm driver. The last line of code for this routine should be:
  - i. `letimer_pwm_open(LETIMER0, "struct name of your choice");`

### 16. `letimer_pwm_open()` function:

- a. The imported project, `PWM_F2020.zip`, provides the complete framework required for this project where all the functions that are required to implement the project have their function prototypes defined in `app.h` and `letimer.h`.
- b. As mentioned above, what is the first thing that should be done in a driver?
  - i. Enable the clock to the `LETIMER` peripheral
  - ii. Previously in the `cmu_open()` function, you routed the `ULFRCO` oscillator onto the `LFA` clock branch through the `LFACLK` clock switch, highlighted in dark gray / black below. The `ULFRCO` clock still is not yet at the `LETIMER0` peripheral due to the clock gate in the below diagram, highlighted in red. The clock gate acts as an AND gate.



- iii. The clock gate, "AND logic," has two inputs. One input is the schematic signal `CMU_LFACLKEN0_LETIMER0`, highlighted in blue, and the other signal is now the `ULFRCO` clock which has been routed to the "AND gate." Per the AND Gate Truth Table, when the schematic signal

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

*CMU\_LFACKEN0.LETIMER0 is deasserted, 0, the output of the clock gate is always 0. When the schematic signal CMU\_LFACKEN0.LETIMER0 is asserted, 1, the output of the clock gate is equal to the ULFRCO clock signal. To pass the ULFRCO clock to the LETIMER0 peripheral, you will program the schematic signal CMU\_LFACKEN0.LETIMER0 to be asserted, set to 1.*

### AND Gate Truth Table

TABLE 3-2

Truth table for a 2-input AND gate.

INPUTS		OUTPUT
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

1 = HIGH, 0 = LOW

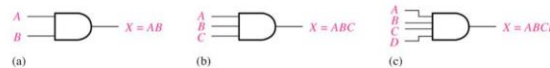


Figure 3-14 Boolean expressions for AND gates with two, three, and four inputs.

- iv. With the proper function input arguments, the `CMU_ClockEnable()` will either assert, set to 1, or deassert, clear to 0, the `CMU_LFACKEN0.LETIMER0` signal. Setting the function argument `Enable` to true will assert, set to 1, this control signal while `Enable` set to false will deassert, clear to 0 `CMU_LFACKEN0.LETIMER0`.
- v. With the concept that the `letimer_pwm_open()` function is a generic driver that could support any of the LETIMERs found in the Pearl Gecko or similar micro-controller, you will need to use an if statement to properly select the correct input for the `CMU_ClockEnable()` function. You will use the `LETIMER_TypeDef *letimer` argument found in the `letimer_pwm_open()` function call.
  1. For example if (`letimer == LETIMER0`), you will use the `CMU_ClockEnable()` that enables the LETIMER0 clock
  2. Since there is no LETIMER1 in the Pearl Gecko, do not add an IF statement for LETIMER1.
- vi. **CMU\_ClockEnable(You can find the enumeration in the HAL doc, true);**
- c. As a low energy micro-controller, all clocks to the peripheral and the peripheral's registers are disabled by default. Without proper clock tree configuration and enabling, data written to a peripheral register will not occur. The instruction will not "crash/terminate" the program or notify you that the write did not complete correctly. To test the clock tree configuration and that the clock is enabled, add the following sequence of code with an ASSERTION test. The code tries to start the LETIMER by writing a command bit to the CMD register and verifies whether the LETIMER has started by reading the RUNNING status bit from the STATUS register. If successful, the code then stops the LETIMER by writing a bit to the CMD register to STOP. After stopping the LETIMER, the code then must re-initialize the CNT register to the value 0 so that upon enabling after initialization,

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

it will quickly load the top value from COMP0 instead of the default value of 0xffff.

```
// Verify whether the LETIMER clock tree properly configured and enabled
// You must select a register that utilizes the clock enabled to be tested
letimer->CMD = LETIMER_CMD_START;
while (letimer->SYNCBUSY);
EFM_ASSERT(letimer->STATUS & LETIMER_STATUS_RUNNING);
letimer->CMD = LETIMER_CMD_STOP;
while(letimer->SYNCBUSY);

// Must reset the LETIMER counter register since enabling the LETIMER to verify that
// the clock tree has been correctly configured to the LETIMER may have resulted in
// the counter counting down from 0 and underflowing which by default will load
// the value of 0xffff. To load the desired COMP0 value quickly into this
// register after complete initialization, it must start at 0 so that the underflow
// will happen quickly upon enabling the LETIMER loading the desired top count from
// the COMP0 register.
letimer->CNT = 0;
```

- d. *If the clock is not enabled, the program will remain stuck at the while (letimer->SYNCBUSY) instruction or the ASSERTION test will fail and the code will jump to the assertEFM()'s while(1) loop.*
  - i. *Indicating one of the following:*
    - 1. *The peripheral clock is not enabled*
    - 2. *The peripheral clock tree is not properly configured*
    - 3. *Or, the input argument pointer for letimer is incorrect*
  - ii. *You will learn more about SYNCBUSY in a future lab assignment*
- e. *By passing pointers to LETIMER to open via its data structure, [LETIMER\\_TypeDef](#) \*letimer, your driver/open function can be universal, supporting all peripherals of the same kind.*
  - i. *With the base address of the data structure, letimer, all of its registers can be accessed via using the -> assignment such as letimer->IF will access the LETIMER Interrupt Flag register*
- f. *After the LETIMER clock has been enabled, the next step is to use the configuration information sent to the letimer\_pwm\_open() function through the letimer\_pwm\_open argument [APP\\_LETIMER\\_PWM\\_TypeDef](#) \*app\_letimer\_struct to initialize the LETIMER through the LETIMER\_Init() function.*
- g. *The [LETIMER\\_Init\\_TypeDef](#) STRUCT used by the LETIMER\_Init() function is not the same as the [APP\\_LETIMER\\_PWM\\_TypeDef](#). To initialize / assign values to the [LETIMER\\_Init\\_TypeDef](#) STRUCT elements, you will be using values from the input argument STRUCT, [app\\_letimer\\_struct](#), as well as direct assignments that will make this LETIMERO peripheral implement a PWM function.*



## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

```
void letimer_pwm_open(LETIMER_TypeDef *letimer, APP_LETIMER_PWM_TypeDef *app_letimer_struct){
    /* Declare local variables */
    LETIMER_Init_TypeDef letimer_pwm_values;

    /* Enable the clock to the LETIMER0 peripheral */

    /* Use EFM_ASSERT statements to verify whether the LETIMER clock tree properly configured
    * and enabled */

    // Initialize letimer values for PWM operation
    letimer_pwm_values.bufTop = xxx; // what value should xxx be?
    letimer_pwm_values.comp0Top = xxx; // what value should xxx be?
    letimer_pwm_values.debugRun = app_letimer_struct->yyy; // what yyy variable from the input arg should be used?
    letimer_pwm_values.enable = app_letimer_struct->yyy; // what yyy variable from the input arg should be used?
    letimer_pwm_values.out0Pol = xxx; // what value should xxx be?
    letimer_pwm_values.out1Pol = xxx; // what value should xxx be?
    letimer_pwm_values.repMode = zzz; // What zzz value, enumeration should be used from HAL documentation?
    letimer_pwm_values.ufoa0 = zzz; // What zzz value, enumeration should be used from HAL documentation?
    letimer_pwm_values.ufoa1 = zzz; // What zzz value, enumeration should be used from HAL documentation?

    LETIMER_Init(letimer, &letimer_pwm_values); // Initialize letimer

    // Initialize COMP0 and COMP1

    // Set the REP0 mode bits for PWM
    // use the values from app_letimer_struct input argument for ROUTELOC and ROUTEPEN enable

    // We are not enabling any interrupts at this time. If you were, you would do it now

    // We will not enable the LETIMER0 at this time
}
```

- h. The LETIMER is on a different oscillator / frequency compared to the main processor clock, so the LETIMER's peripheral registers will be asynchronous to the processor's clock. To synchronize these clocks, these registers go through a two-stage synchronization circuit.

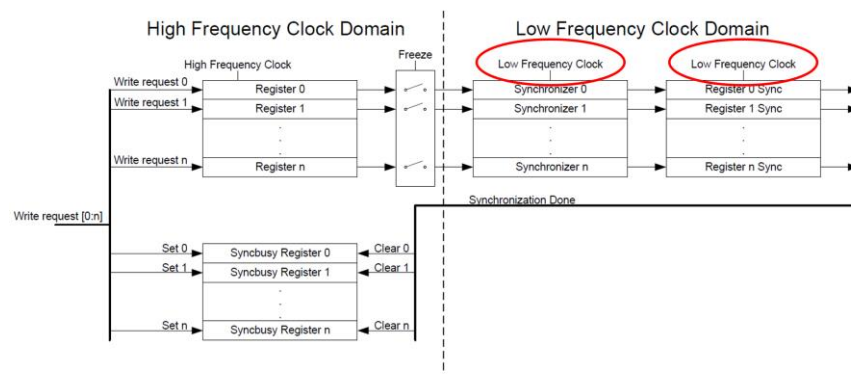


Figure 4.8. Write operation to Low Energy Peripherals

- i. If you go into the LETIMER\_Init() library function, if LETIMER\_Init\_TypeDef input argument enables the LETIMER, the CMD register of the LETIMER peripheral will be written to.

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

```
/* Start the timer if specified to be enabled and not already running. */
if (init->enable && !(letimer->STATUS & LETIMER_STATUS_RUNNING)) {
#ifdef _EFM32_GECKO_FAMILY
    /* LF register about to be modified requires sync; busy check. */
    regSync(letimer, LETIMER_SYNCBUSY_CMD);
#elif defined(LETIMER_SYNCBUSY_START)
    while (letimer->SYNCBUSY & LETIMER_SYNCBUSY_START) {
    }
#endif
    letimer->CMD = LETIMER_CMD_START;
}
```

- j. Per the LETIMER peripheral SYNCBUSY register, the CMD register can be tested for synchronization to be completed

20.5.13 LETIMERn\_SYNCBUSY - Synchronization Busy Register

Offset	Bit Position																															
0x034	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0	
Access																														R		
Name																														CMD		

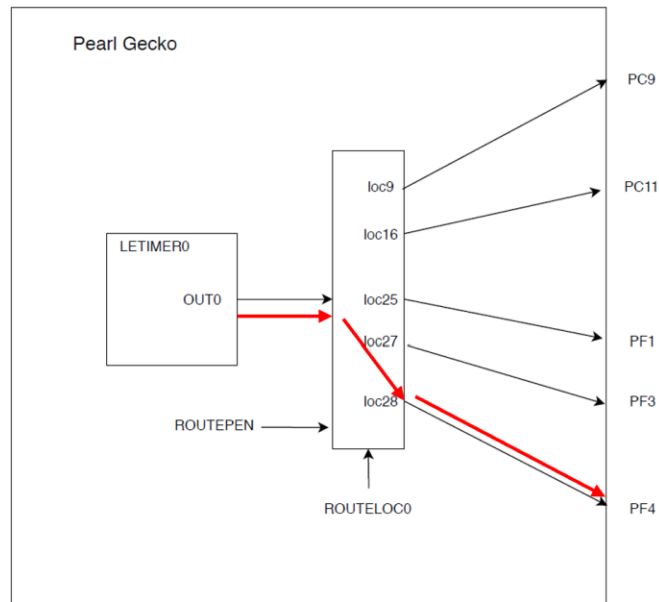
Bit	Name	Reset	Access	Description
31:2	Reserved	To ensure compatibility with future devices, always write bits to 0. More information in <a href="#">1.2 Conventions</a>		
1	CMD	0	R	<b>CMD Register Busy</b> Set when the value written to CMD is being synchronized.
0	Reserved	To ensure compatibility with future devices, always write bits to 0. More information in <a href="#">1.2 Conventions</a>		

- k. To validate whether the CMD register is in process of being synchronized and stall until the synchronization has completed, use a while() statement that continues to check the SYNCBUSY register until it returns false, 0, indicating no synchronization is in process.
- while(letimer->SYNCBUSY);
  - Add this check and stall immediate following the [LETIMER\\_Init\(\)](#) function call
- l. After you have initialized the LETIMER, you will then need to calculate the values to store in the LETIMER's COMP0 and COMP1 registers.
- m. Similar to accessing the peripheral registers using a base pointer of a data structure, the data configuration information is being passed to letimer\_pwm\_open() via a structure, [APP\\_LETIMER\\_PWM\\_TypeDef](#) \*app\_letimer\_struct. Any variable defined in this structure can be accessed by using the -> assignment. For example, the [period](#) variable in the structure [app\\_letimer\\_struct](#) defined by APP\_LETIMER\_PWM\_TypeDef can be accessed like the example below:
- period\_cnt = [app\\_letimer\\_struct->period](#) \* LETIMER\_HZ;



## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

- ii. The above c-line of code will calculate the LETIMER counts required for the LETIMER period based on the defined number, LETIMER\_HZ, of HZ for the LETIMER clock source
- n. With the COMP0 and COMP1 registers now defined, you must complete the LETIMER driver open by routing the LETIMER0 outputs to the output pins of the Pearl Gecko to blink the LED.



The LETIMER ROUTEPEN register enables the output from the internal routing de-multiplexer

The LETIMER ROUTELOC0 sets the select lines to the routing de-multiplexer

- o. LED0 is connected to gpioPortF, pin 4 per Lab 1. To route the output of LETIMER0 peripheral to the demultiplexer output which allows the signal to the proper Pearl Gecko's Output pin, the select lines, [ROUTELOC0](#), of the demultiplexer must be configured. Once the select lines have been configured, then you must enable the output of the demultiplexer, [ROUTEPEN](#), so the LETIMER0 output will now drive directly the Pearl Gecko output.
  - i. You should assign the bit in the register with the value sent to the [letimer\\_pwm\\_open\(\)](#) function via the [app\\_letimer\\_struct](#)
  - ii. Ensure that you are writing the correct bit location for OUT0 and OUT1 in the ROUTEPEN register
- p. To route the output of OUT0 of the LETIMER0 to the Pearl Gecko GPIO pin, the ROUTELOC0 register must be set with the correct routing selection value. This value can be found in the Pearl Gecko datasheet.

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

GPIO Name	Pin Alternate Functionality / Desc			Previous	Next
	Analog	Timers	Comm		
PI3	BUSADC0Y BUSADC0X	PCNT1_S0IN #5 PCNT1_S1IN #4 PCNT2_S0IN #5 PCNT2_S1IN #4	US2_TX #8 US2_RX #7 US2_CLK #6 US2_CS #5 US2_CTS #4 US2_RTS #3 US3_TX #9 US3_RX #8 US3_CLK #7 US3_CS #6 US3_CTS #5 US3_RTS #4 I2C1_SDA #5 I2C1_SCL #4		LES_ALTEX7 ETM_TDO #2
PF5	BUSAY BUSBX	TIM0_CC0 #29 TIM0_CC1 #28 TIM0_CC2 #27 TIM0_CDTI0 #26 TIM0_CDTI1 #25 TIM0_CDTI2 #24 TIM1_CC0 #29 TIM1_CC1 #28 TIM1_CC2 #27 TIM1_CC3 #26 WTIM1_CC0 #29 WTIM1_CC1 #27 WTIM1_CC2 #25 WTIM1_CC3 #23 LE- TIM0_OUT0 #29 LE- TIM0_OUT1 #28 PCNT0_S0IN #29 PCNT0_S1IN #28	US0_TX #29 US0_RX #28 US0_CLK #27 US0_CS #26 US0_CTS #25 US0_RTS #24 US1_TX #29 US1_RX #28 US1_CLK #27 US1_CS #26 US1_CTS #25 US1_RTS #24 US2_TX #18 US2_RX #17 US2_CLK #16 US2_CS #15 US2_CTS #14 US2_RTS #13 LEU0_TX #29 LEU0_RX #28 I2C0_SDA #29 I2C0_SCL #28		PRS_CH0 #5 PRS_CH1 #4 PRS_CH2 #3 PRS_CH3 #2 ACMP0_O #29 ACMP1_O #29
PF4	BUSBY BUSAX	TIM0_CC0 #28 TIM0_CC1 #27 TIM0_CC2 #26 TIM0_CDTI0 #25 TIM0_CDTI1 #24 TIM0_CDTI2 #23 TIM1_CC0 #28 TIM1_CC1 #27 TIM1_CC2 #26 TIM1_CC3 #25 WTIM1_CC0 #28 WTIM1_CC1 #26 WTIM1_CC2 #24 WTIM1_CC3 #22 LE- TIM0_OUT0 #28 LE- TIM0_OUT1 #27 PCNT0_S0IN #28 PCNT0_S1IN #27	US0_TX #28 US0_RX #27 US0_CLK #26 US0_CS #25 US0_CTS #24 US0_RTS #23 US1_TX #28 US1_RX #27 US1_CLK #26 US1_CS #25 US1_CTS #24 US1_RTS #23 US2_TX #17 US2_RX #16 US2_CLK #15 US2_CS #14 US2_CTS #13 US2_RTS #12 LEU0_TX #28 LEU0_RX #27 I2C0_SDA #28 I2C0_SCL #27		PRS_CH0 #4 PRS_CH1 #3 PRS_CH2 #2 PRS_CH3 #1 ACMP0_O #28 ACMP1_O #28

- q. The number after the peripheral output, LETIM0\_OUT0, is the routing location, #28, to write to the correct bit field in the LETIMER0->ROUTELOC0 register

r. Particular registers must be configured in a particular way.

### 17. app\_peripheral\_setup() function

- a. In the app.c file, you must add the arguments to the function calls within the **app\_peripheral\_setup.c** function. You will need to uncomment these lines of code. You can use **ctrl+/** as a short cut to uncomment the lines of code at one time. These functions will be used to set up the structures that will be used by the peripheral's open driver functions.
  - i. `app_letimer_pwm_open(, );`
  - ii. The arguments for these functions can be found in their function prototype definitions located in their associated .h files.

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

18. The above **app\_letimer\_pwm\_open()** function calls the LETIMER open functions. This function call should set the structure defined in the **letimer.h** file with the desired values to enable proper PWM operation.
19. Each function in **app.c**, **cmu.c**, and **gpio.c** should have a doxygen comment block before the function that fully describes what the function performs, the input arguments, and outputs. Below is an example from a Silicon Labs embedded library function.

```
/*
 * @brief
 *   Calibrate the clock.
 *
 * @details
 *   Run a calibration for HFCLK against a selectable reference clock.
 *   See the reference manual, CMU chapter, for more details.
 *
 * @note
 *   This function will not return until the calibration measurement is completed.
 *
 * @param[in] HFCycles
 *   The number of HFCLK cycles to run the calibration. Increasing this number
 *   increases precision but the calibration will take more time.
 *
 * @param[in] ref
 *   The reference clock used to compare HFCLK.
 *
 * @return
 *   The number of ticks the reference clock after HFCycles ticks on the HF
 *   clock.
 */
uint32_t CMU_Calibrate(uint32_t HFCycles, CMU_Osc_TypeDef reference)
{
```

20. I have provided the Doxygen block framework in the **app.c** template that you had downloaded to begin this project.

```
/*
 * @brief
 *
 *
 * @details
 *
 *
 * @note
 *
 *
 * @param[in]
 *
 *
 * @param[in]
 *
 */
```

21. In addition to each comment block, the file must have a doxygen module block that provides information on the module file name, author, date, and module brief. The brief provides a summary of what the code in its **.c** module performs. For example, in

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

cmu.c, the brief could be that the cmu.c module is responsible in enabling all oscillators and routing the clock tree for the application.

```
/**
 * @file
 * @author
 * @date
 * @brief
 *
 */
```

- a. Please note that the @file must refer to the exact name of the .c file that this Doxygen file document header is located.
22. You will need to add and complete the doxygen file and function comment blocks in cmu.c and gpio.c modules as well.
  23. For the letimer.c module, you will need to update the doxygen file comment block with you as the author.
  24. With the doxygen comment fields completed, its time to prepare and generate the doxygen report.
    - **Doxygen** (*[/ˈdɒksɪdʒən/ DOK-see-jən](#)*<sup>[2]</sup>) is a [documentation generator](#), a tool for writing software reference documentation. The documentation is written within code, and is thus relatively easy to keep up to date. Doxygen can cross reference documentation and code, so that the reader of a document can easily refer to the actual code. (Wikipedia)
    - Doxygen is [free software](#), released under the terms of the [GNU General Public License version 2](#) (GPLv2).
  25. You can download Doxygen by going to git hub at the link below and selecting the proper version based on your operating system
    - a. <http://www.doxygen.nl/>
  26. Below is a link to a short doxygen tutorial:
    - a. [http://fnch.users.sourceforge.net/doxygen\\_c.html](http://fnch.users.sourceforge.net/doxygen_c.html)
  27. Doxygen looks for commands within comment statements identified by the @ character such as:
    - @brief
    - @author
    - @date
    - @detail
    - @note
    - @param[in]

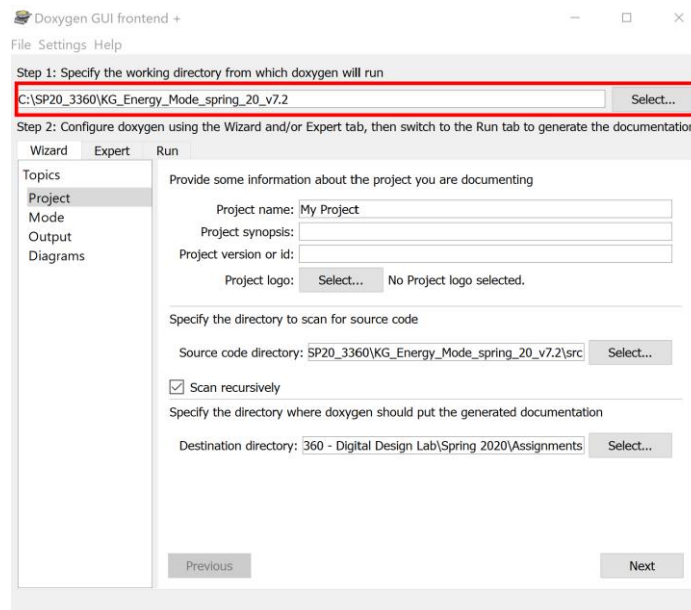
## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

*@return*

*@par (new paragraph)*

28. Once you have installed doxygen, open doxygen to initiate your doxygen comment report.

29. In the wizard step 1, browse to your workspace and then project



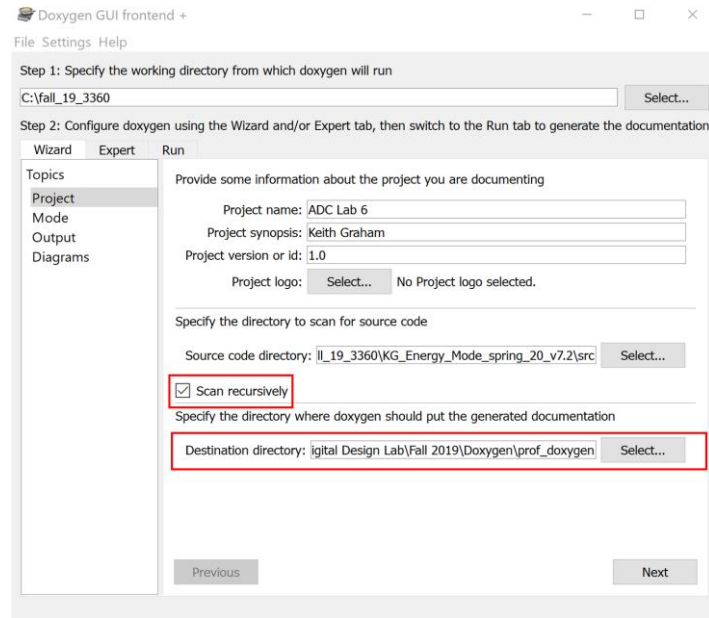
30. In wizard step 2, add the following:

- Type in your Project name, Project synopsis, and Version
- For source code directory, browse into your Simplicity Workspace, Project, and select your src folder

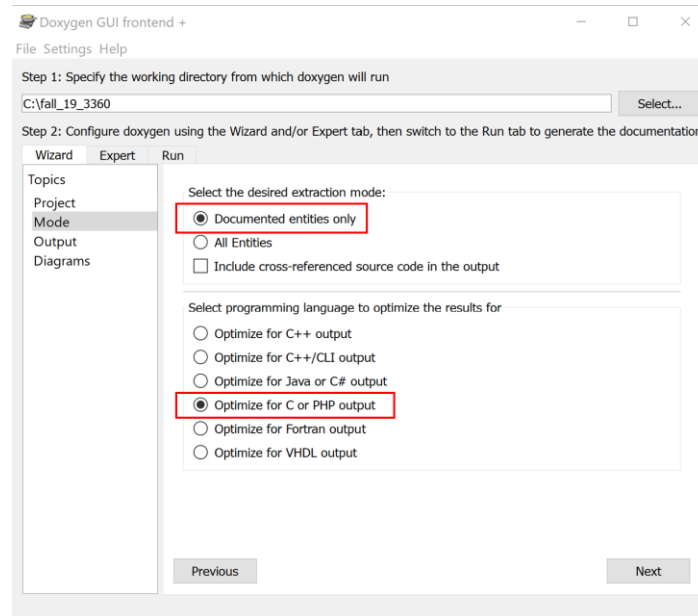
31. Continuing wizard step 2, add the following:

- Make sure that Scan recursively is checked to enable the scan of both your header and source files
- Also, browse to a folder outside of your workspace to store the output of this Doxygen document
- Then, click on Next to proceed to Mode menu

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

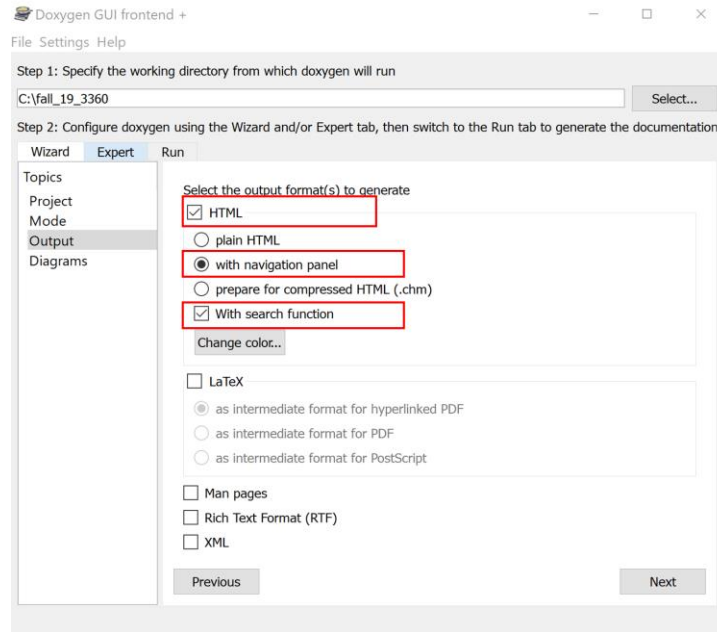


32. In the mode menu, perform the following:
- Select Documented entities only
  - And, Optimize for C or PHP output
  - Then click on Next to go to the Output menu

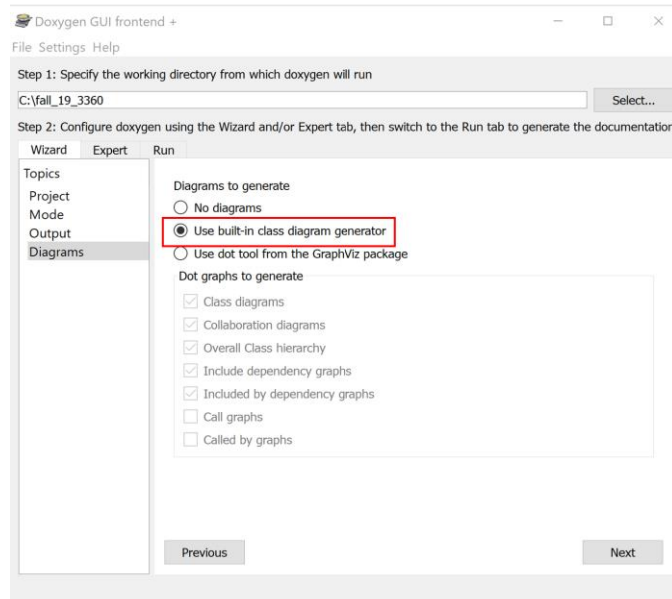


33. In the output menu, perform/verify the following options:
- Check HTML
  - With navigation panel
  - Check with search function
  - Then click on next to proceed to the Diagrams menu

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

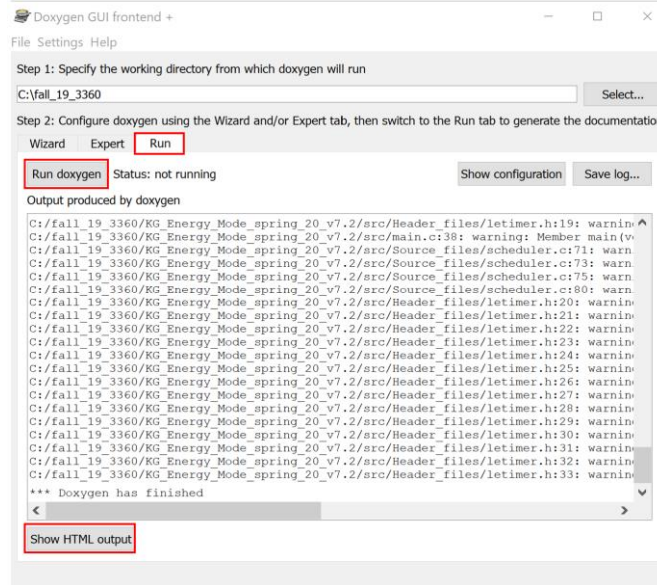


34. In the Output menu, select the following:
- Use built-in class diagram generator
  - Then, click on Next to move to the Run menu

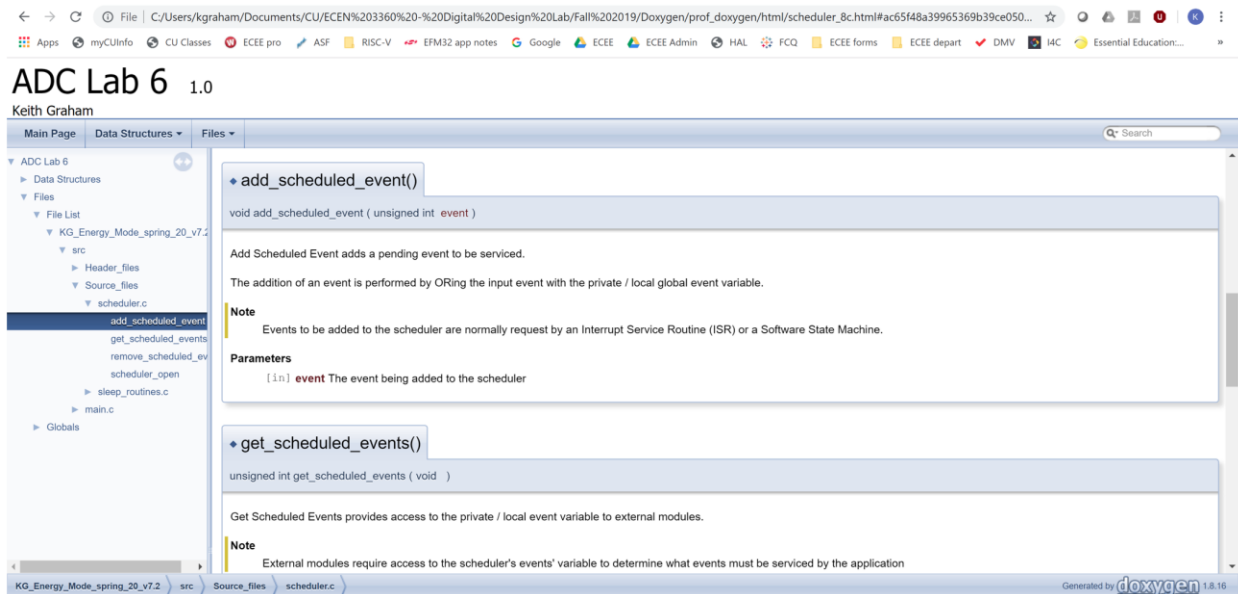


35. In the Run menu,
- Click on Run doxygen, and you will see the progress as your documentation is being auto generated
  - Once it is finished, to view your documentation in your browser, click on Show HTML output

# READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN



36. After clicking Show HTML output, it should open a web browser. Click on files to find your .c files (letimer.c and app.c)



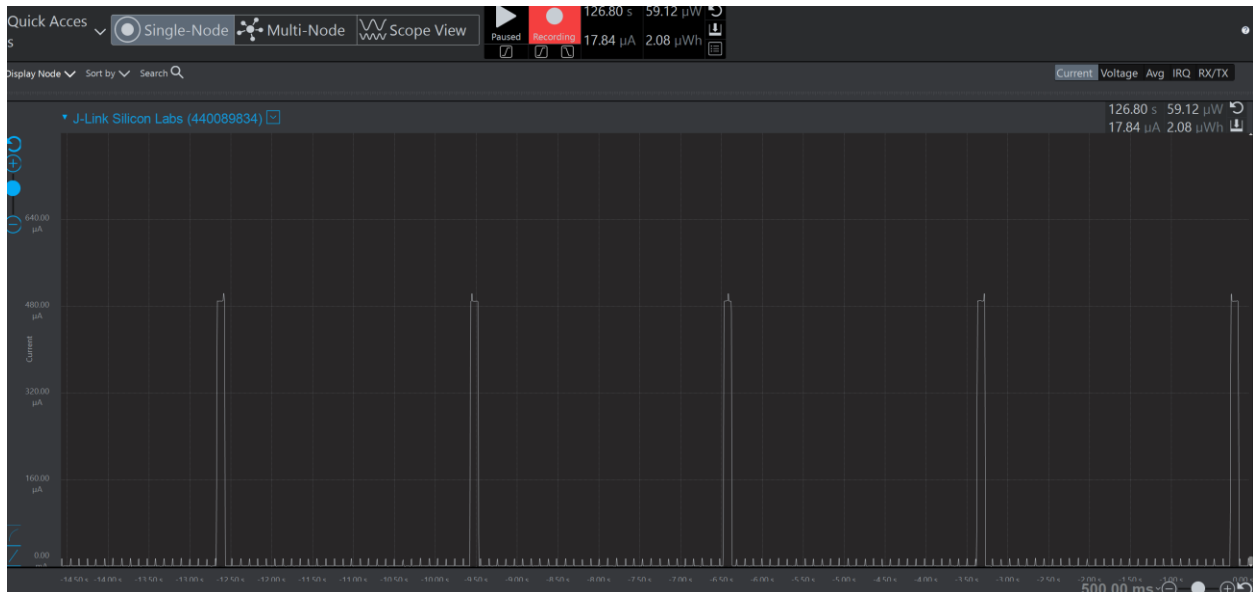
37. Doxygen debug hints:

- a. If your module such as app.c does not show up in your doxygen report
  - i. The file name in your doxygen module comment block must be identical to the name of the file. For instance for the module letimer.c
    1. @file letimer.c
- b. If you comments are not showing up in your doxygen report



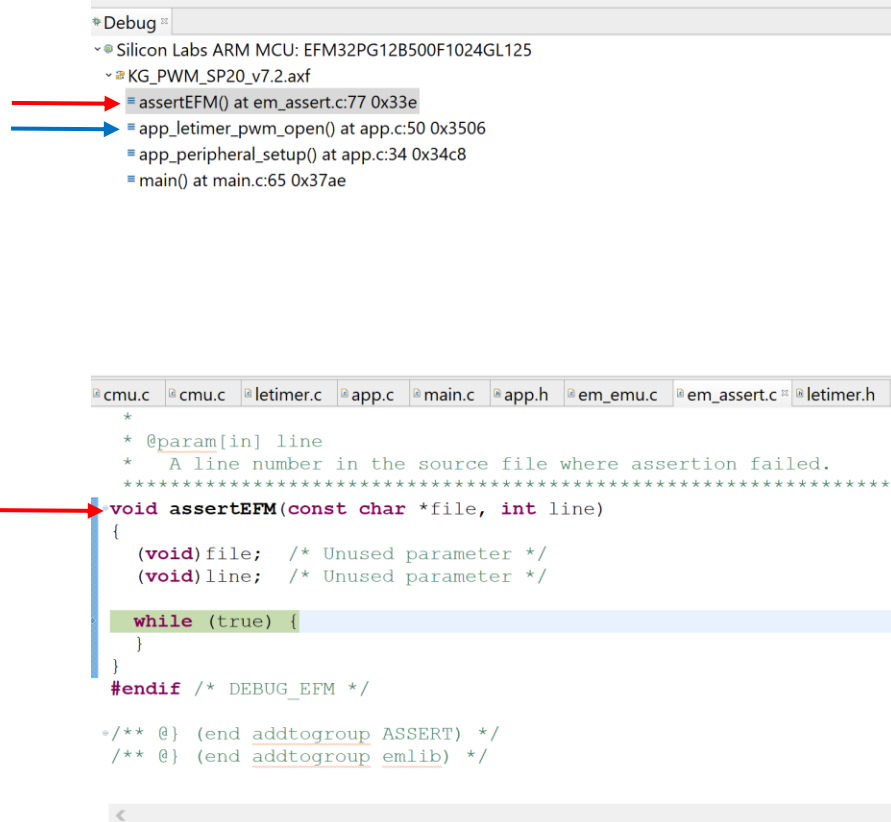
## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

- i. Doxygen uses the `/**` to indicate the start of a doxygen comment statement. If you use `/*`, the doxygen report generator will skip that comment assuming it is a standard and not a doxygen comment block
38. With your doxygen report generated, to open it up after you have closed the doxygen wizard, go into the generated html folder and click on [index.html](#). Opening this file will open your browser and doxygen report.
39. Now, let's go back to your project and run your project using the Energy Profiler. Make sure that you toggle the switch on the Pearl Gecko to reset the current monitor to get the best measured results. A correctly developed program will have an Energy Profiler like the screen shot below:



40. If your code never failed an `EFM_ASSERT()` test, go back to your code and try the following:
- a. Add the following line of code immediately before you call `letimer_pwm_open()` in your `app_letimer_pwm_open()` function
    - i. `EFM_ASSERT(false);`
  - b. Run your code in the debugger
    - i. Does your LED0 blink? I suspect no
    - ii. Pause your program in the debugger. It should have failed the `EFM_ASSERT()` test that you added to this function. You can determine which `EFM_ASSERT()` test failed by using the information in the upper right hand screen as previously discussed in this lab or you can use the upper left window.

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN



The image shows a debugger interface. The top window, titled 'Debug', displays a call stack with the following entries from bottom to top: `main() at main.c:65 0x37ae`, `app_peripheral_setup() at app.c:34 0x34c8`, `app_letimer_pwm_open() at app.c:50 0x3506`, and `assertEFM() at em_assert.c:77 0x33e`. A red arrow points to the `assertEFM()` entry, and a blue arrow points to the `app_letimer_pwm_open()` entry. The bottom window shows the source code for `em_assert.c`. A red arrow points to the function definition of `void assertEFM(const char *file, int line)`. The code includes comments about unused parameters and a `while (true)` loop.

```
cmu.c cmu.c letimer.c app.c main.c app.h em_emu.c em_assert.c letimer.h
*
* @param[in] line
*   A line number in the source file where assertion failed.
*****
void assertEFM(const char *file, int line)
{
    (void) file; /* Unused parameter */
    (void) line; /* Unused parameter */

    while (true) {
    }
}
#endif /* DEBUG_EFM */

/** @} (end addtogroup ASSERT) */
/** @} (end addtogroup emlib) */
```

- c. In the upper left hand window of your screen, you will see a list of all the functions that were called to get to the `assertEFM()` function
  - i. `main.c` called `app_peripheral_setup()`
  - ii. `app_peripheral_setup()` called `apps_letimer_pwm_open()`
  - iii. `apps_letimer_pwm_open()` called `assertEFM()`
- d. Click on `app_letimer_pwm_open()` indicated by the blue arrow above
- e. If you click on the routine that called `assertEFM()`, the function immediate after `assertEFM()`, it will open the file that contains the `EFM_ASSERT()` function call. The highlighted c-line of code will be the actual `EFM_ASSERT()` statement or as in this case, the c-line of code immediately after the `EFM_ASSERT()` call
- f. Before you forget, delete the `EFM_ASSERT(false)` line of code that you added in `app_letimer_pwm_open()` so that your program does not fail this `EFM_ASSERT()` and you can answer the questions for the Lab 2 worksheet.

### Deliverables:

1. Each person exports their project as an archived file. Upload the .zip file into the Lab 2 exported Project canvas assignment
2. Each person exports their doxygen report as an archived, .zip, file into their lab 2 exported Project canvas assignment

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

- Each person to provide their answers via Canvas/quiz/Lab2 Worksheet

Questions:

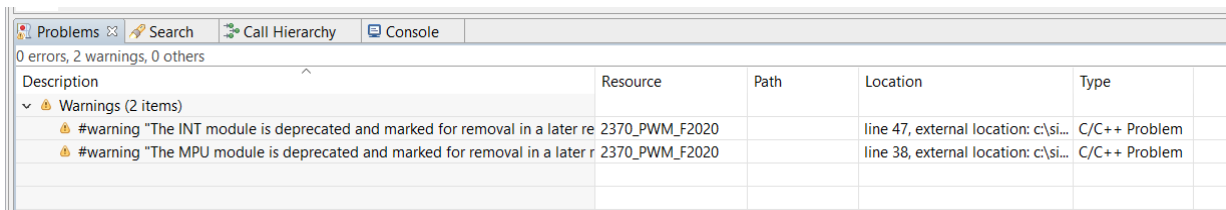
Please complete the Lab 2 worksheet to complete this assignment.

Point breakdown:

- Lab 2 has a total of 35 points
- Exported project will be graded on:
  - Functionality
    - Proper period of operation
      - Can be found in app.h
    - Proper active period of operation
      - Can be found in app.h
  - Meeting Energy / Current expectations
    - Energy Profiler results
  - Proper commenting of functions
  - No use of magic numbers
  - Best coding practices
- Partial credit will be given on code that is not completely functional

Possible point deductions

- Use of Magic numbers
  - Take this deduction only once for this assignment (-1 pt)
- Perform a Clean Build and then build the project
  - Point deduction for each warning statement which is not related to a Silicon Labs emlib routine (-1 pt / warning)
  - These warnings do not warrant any point deductions



Description	Resource	Path	Location	Type
0 errors, 2 warnings, 0 others				
Warnings (2 items)				
#warning "The INT module is deprecated and marked for removal in a later re	2370_PWM_F2020		line 47, external location: c:\si...	C/C++ Problem
#warning "The MPU module is deprecated and marked for removal in a later r	2370_PWM_F2020		line 38, external location: c:\si...	C/C++ Problem

Late Penalty deduction:

- Exported program
    - Due day + 1 day
    - 1 day late to 3 days late
    - 3 days late to 5 days late
- max score is 30 pts  
max score is 25 pts  
max score is 15 pts

## READ ENTIRE LAB ASSIGNMENT BEFORE YOU BEGIN

- After 5 days late

max score is 0 pts