ECE 491 Lab 3

Timers and Pulse Width Modulation (PWM)

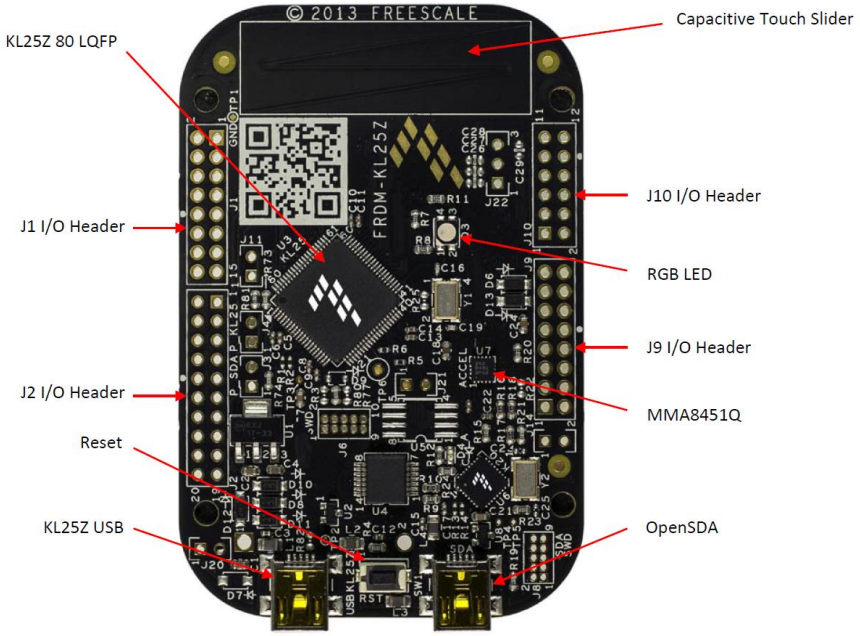
1. Introduction:

The purpose of this lab is to get familiar with Freedom Board KL25z Timer and Pulse Width Modulation (PWM) configuration. We shall be using the Kinetics Design Studio (KDS) Integrated Development Environment (IDE) and Processor Expert (PE) for rapid code development. The groups will demonstrate the following for checkoff

- Blink the external LED at 500ms.
- Use PWM to control the intensity of the same LED
- Use PWM to move the servo position to extreme left, extreme right and in the middle.

2. Background and Tools:

a) Freedom Board FRDM-KL25Z:

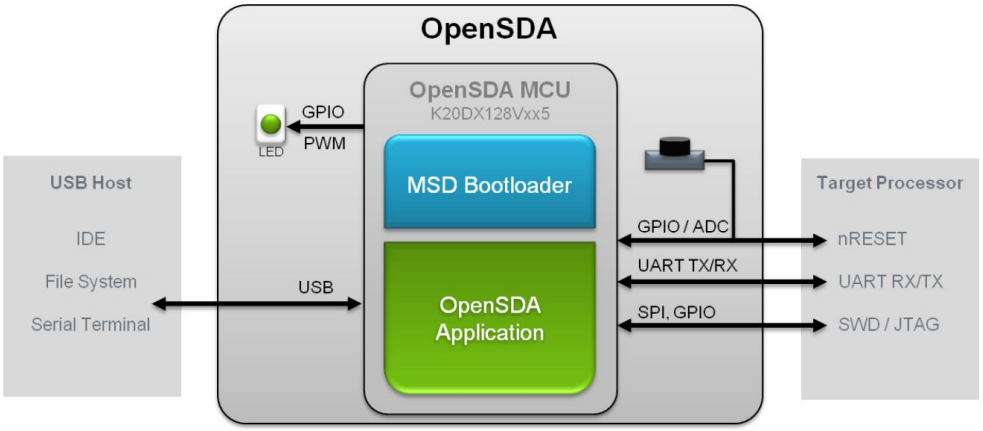


 Group name _____ Date _____

To learn more about the freedom board FRDM-KL25Z refer to user guide and pinout documents on blackboard. (Files FRDM-KL25Z User Manual (Rev 2).pdf and FRDM-KL25Z pinouts (Rev 1.0).pdf). You should always refer to these documents for relevant pinouts and schematics before you make your connections in order to protect your board.

b) Serial and Debug Adapter (OpenSDA):

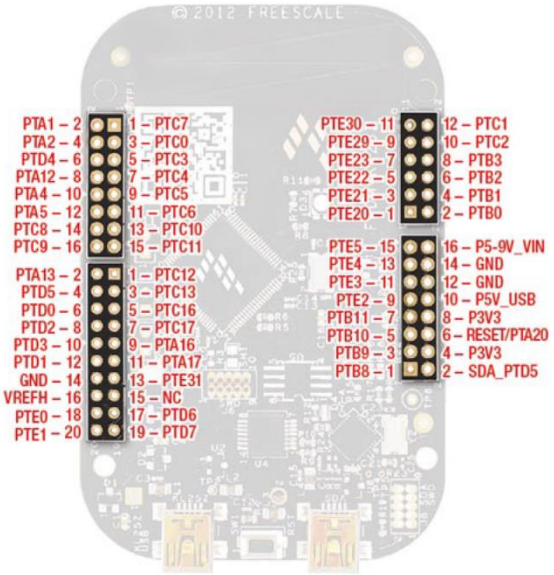
OpenSDA is an open-standard serial and debug adapter. It bridges serial and debug communications between a USB host and an embedded target processor as shown in figure. The hardware circuit is based on a Freescale Kinetis K20 family microcontroller (MCU) with 128 KB of embedded flash and an integrated USB controller. OpenSDA features a mass storage device (MSD) bootloader, which provides a quick and easy mechanism for loading different OpenSDA Applications such as flash programmers, run-control debug interfaces, serial-to-USB converters, and more. Refer to the OpenSDA User's Guide for more details.



c) Input/Output Connectors:

The KL25Z128VLK4 microcontroller is packaged in an 80-pin LQFP. Some pins are utilized in on-board circuitry, but many are directly connected to one of four I/O headers.

The pins on the KL25Z microcontroller are named for their general purpose input/output port pin function. For example, the 1st pin on Port A is referred to as PTA1. The I/O connector pin names are given the same name as the KL25Z pin connected to it, where applicable. Note that all pinout data is available in spreadsheet format in *FRDM-KL25Z Pinouts*. See the Reference Documents section for details.



d) Kinetis Design Studio (KDS) with Processor Expert:

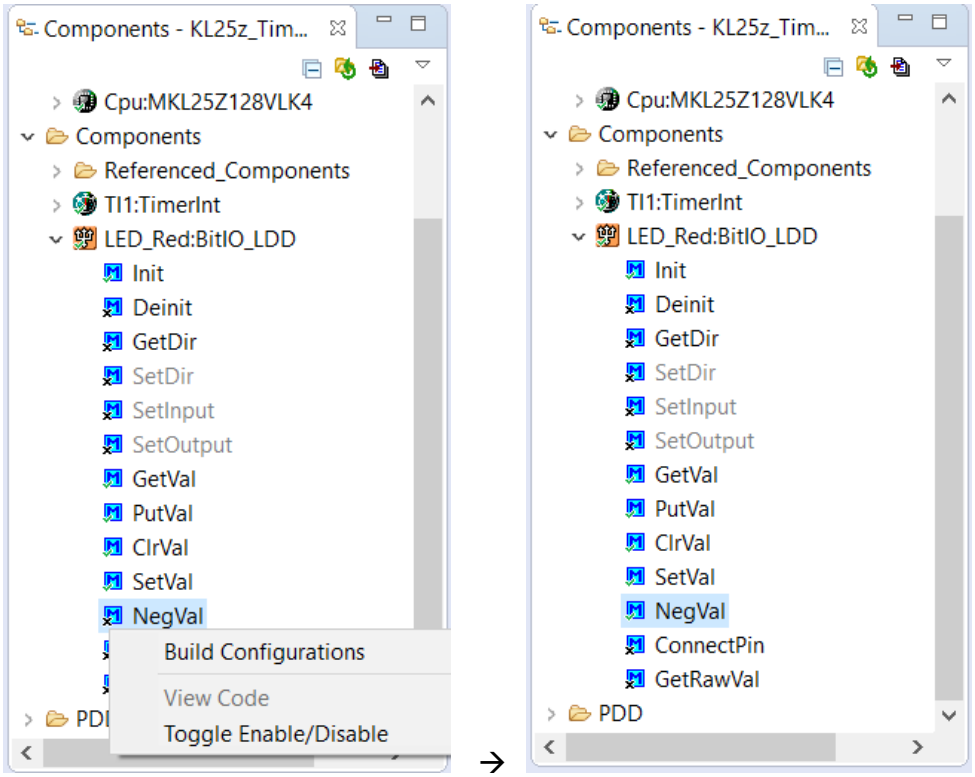
The Kinetis Design Studio (KDS) is used for development which is an integrated development environment for Kinetis MCUs. It enables robust editing, compiling and debugging of your designs. We shall use the Processor Expert integrated in KDS IDE to generate your codes with its knowledge base and helps create powerful applications with a few mouse clicks.

3. Blink an external LED

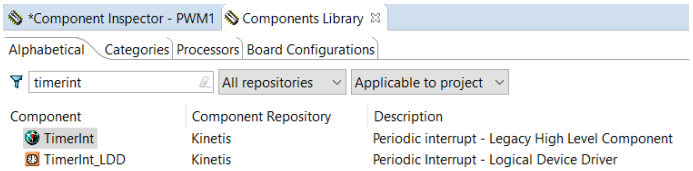
In first part of this exercise, an external LED connected to one of the GPIO will be switched on intermittently at 500ms using the timer interrupt.

a) Using timer interrupt library function.

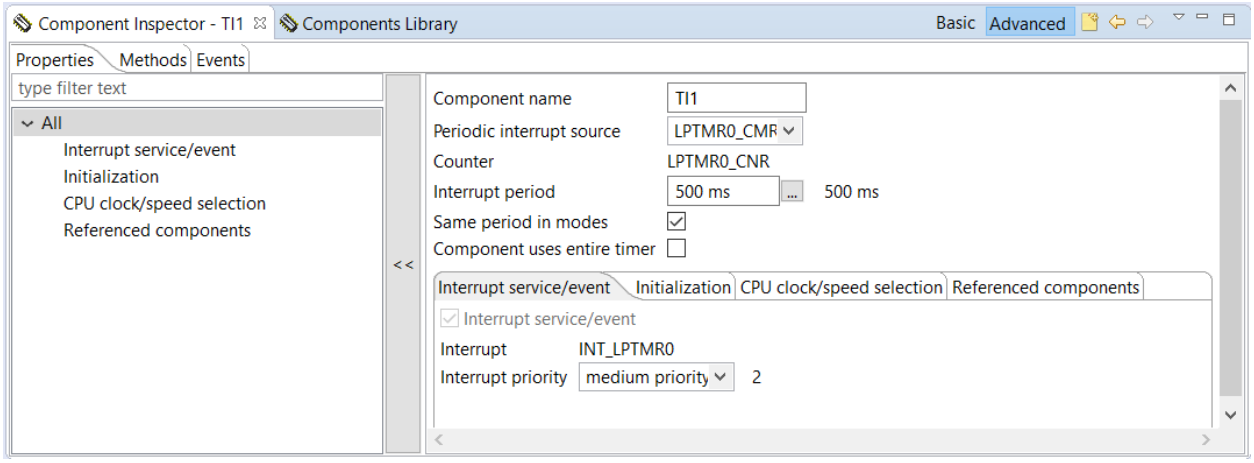
1. Start a new project (refer to section KDS IDE starting a new project from previous lab).
2. Configure any GPIO to connect to the external LED (example PTB0 as per previous lab).
3. The "BitIO_LDD_Negva()"l method will be used to switch the LED On and Off, so make sure that the method is enabled by right-clicking the method and click on "toggle Enable/disable".



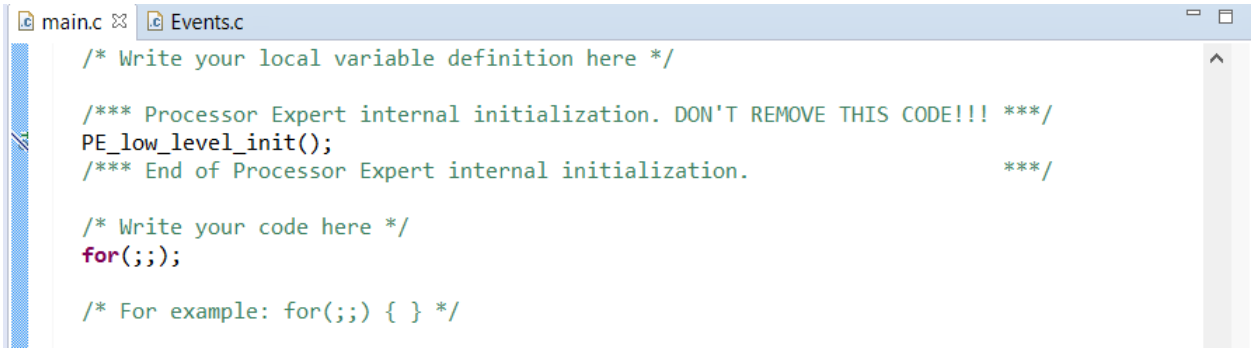
4. Add a "TimerInt_LDD" from the component library.



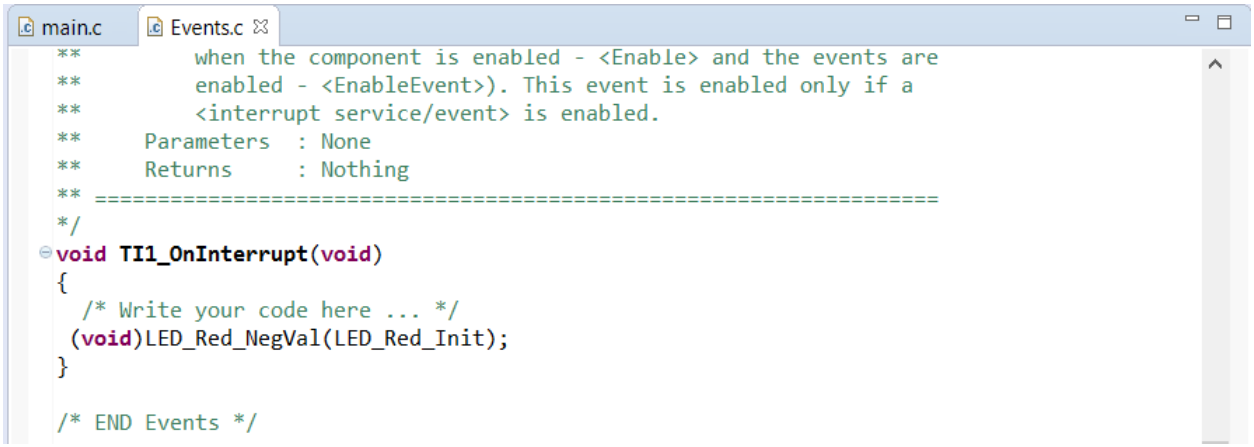
5. Double click to add it to the project and configure as follows: Name the component, Set the time interrupt period to be 500ms



6. Now generate the processor expert code by clicking the icon on the top right corner of the component sub-window.
7. Add a loop code in the main.c file.



8. Open the events.c file, go to the TI1_OnInterrupt() and add the following code.

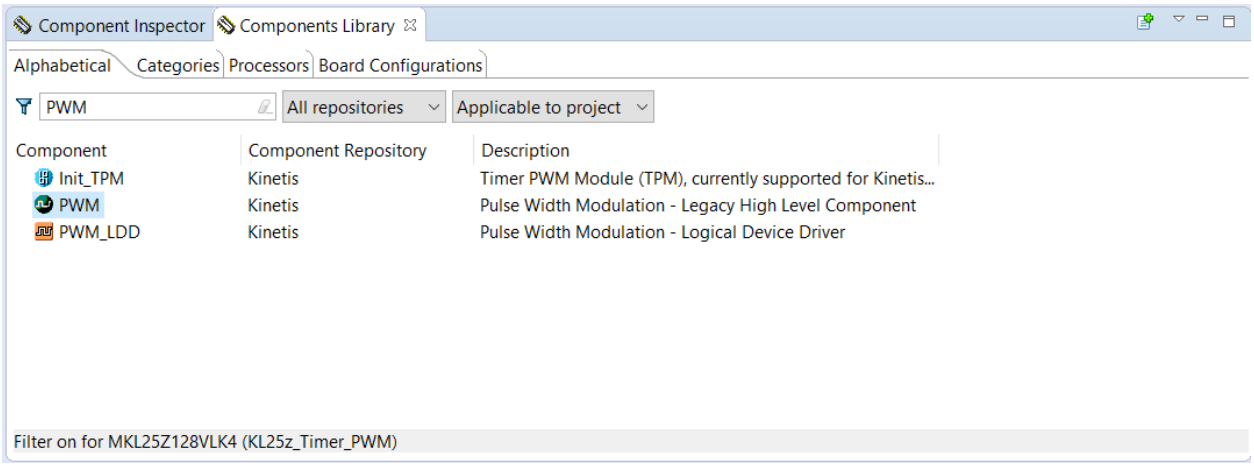


9. This code will generate the interrupt in the main loop at every 500ms and switch the current value of the output at that GPIO to its opposite value which gives the blinkly effect.

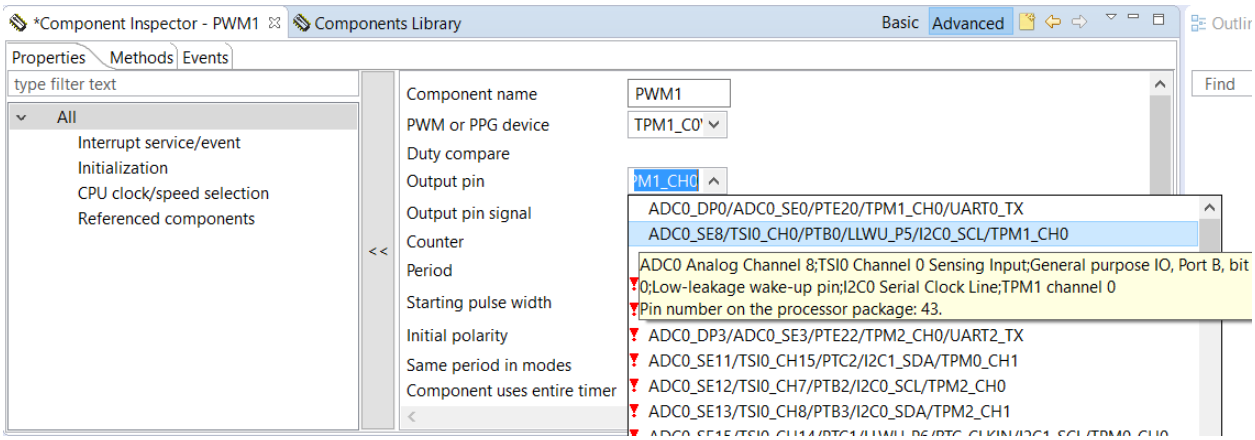
4. Control the intensity of the LED using PWM.

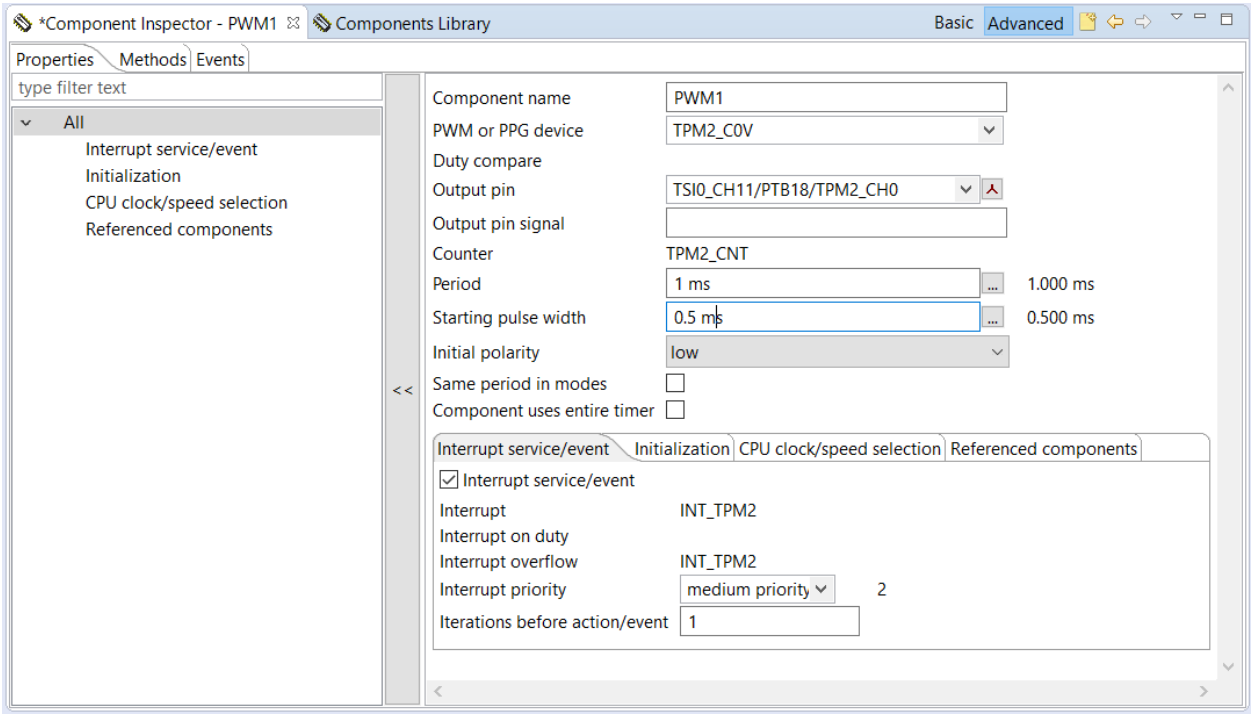
In this exercise, use any GPIO to control the intensity of the LED using the PWM method. Connect the external led as per the previous lab and write the following code.

1. Create a new project.
2. Add “PWM” from component library to the project, it will generate the referred timer component automatically.

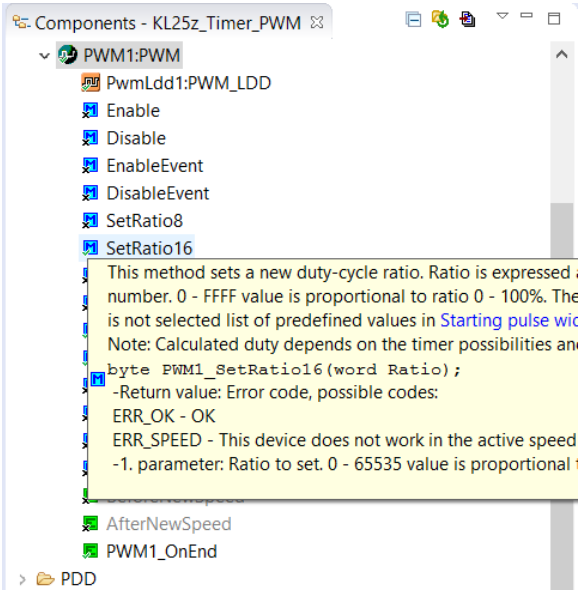


3. Double click the PWM component to open the component inspector and set the following
   - Output Pin: set it to corresponding GPIO (refer to pinouts document like PTB0). Processor expert will set the corresponding counter and clocks
   - Period: Period of the output signal. It is necessary to specify both a value and a unit. The setting can be made with the help of the Timing dialog box that opens when clicking on the button (...). Open it an select any desired value (say 1ms)
   - Starting pulse: Starting pulse width. It specifies the length of time that the output signal spends in the active level during the output cycle. The active level is defined in the “Initial polarity”. It is necessary to specify both a value and a unit. The setting can be made with the help of the Timing dialog box that opens when clicking on the button (...). Make sure that the starting pulse is less than the period otherwise it will generate error (say .5ms for 50% duty cycle)
   - Interrupt service/event: make sure it is enabled.





4. Generate the processor expert code.
5. We shall use the SetRatio16() method to set the duty cycle of the pulse width on the PWM1_OnEnd() interrupt
   - **SetRatio16** (): This method sets a new duty-cycle ratio. Ratio is expressed as a 16-bit unsigned integer number. 0 - FFFF value is proportional to ratio 0 - 100%.
   - **PWM1_OnEnd** (): This event is called when the specified number of cycles has been generated.



6. Double click the PWM1_OnEnd() method to open the events.c file pointing at the method and add the following code

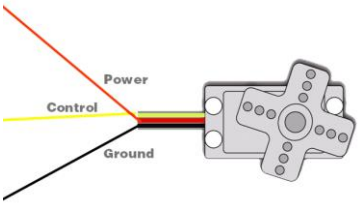
```
main.c  Events.c
*/
unsigned ratio=0;
void PWM1_OnEnd(void)
{
  /* Write your code here ... */
    PWM1_SetRatio16(++ratio);
}
```

7. This will increase the ratio from 0 to FFFF to set the duty cycle from 0 to 100% at the end of every period. You can add any other code here.
8. Build the project and debug

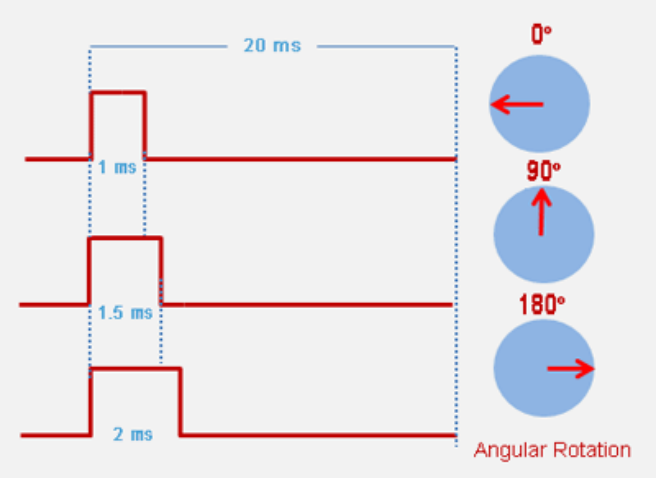
b) Extra Points: Read the in-build touch sensor input and set the intensity based on the read value for extra points!!!

5. PWM to control the servo.

In this exercise, connect the servo from your Freescale Kit as shown below and set the duty cycle (SetRatio() method in PWM library) to move the servo on extreme ends (Left and Right) and to move the link to its center position.



1. Add another PWM to the same project.
2. Configure it another available GPIO and repeat the steps from the previous section.
    a. For the servo control, in the component inspector PWM setting, you should set the period parameter as 20ms. And the starting pulse width of 1.5ms. A typical servo position is shown below:



3. In the PWM2_OnEnd() method, write a code to set the ratio manually to control the servo movement from left to right. Find the following parameters
    a. PWM period required by the servo: _____
    b. PWM duty cycle (SetRatio value) for extreme left: _____
    c. PWM duty cycle (SetRatio value) for extreme right: _____
    d. PWM duty cycle (SetRatio value) for center position: _____
4. You can also check your signal using the oscilloscope and capture images.

6. Checkout

The group has to successfully demonstrate programming all three functionality required by this lab.