**FRDM-KL46Z Baremetal Blinky LED using Periodic Interrupt Timer (PIT)**

Being fairly inexperienced with freescale processors and  having recently acquired a FRDM-KL64Z board i thought it would be a good idea to get some of the basic peripherals working on the board using baremetal.This helped me to get my head round some of the basic ways to initialize peripherals.

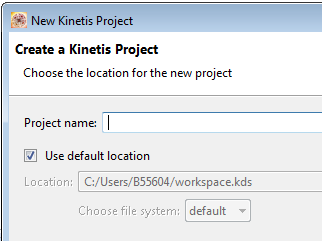
There are 3 things to do:

1.Set Up Kinetis Design Studio (KDS)

2.  LED GPIO

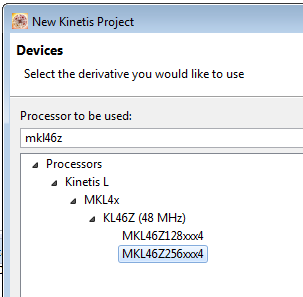
3. PIT

**1.Set Up Kinetis Design Studio (KDS)**

Create a new workspace in Kinetis and then open new Kinetis project 'File>New>Kinetis Project':  


project name : Blinky\_LED and click Next

Select Device by filtering processor and choosing MKL46Z256xxx4

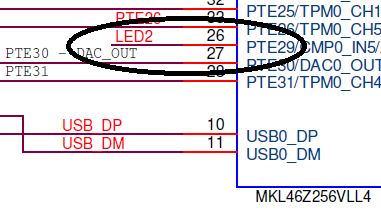


The click finish to create the project for the board.This will build and debug no problem. If Any problems occur it could be a problem with the boot loader, the SDA file can be updated by looking in the following:

[***Quick Start Guide for Freedom FRDM-KL46Z***](http://www.google.fr/url?sa=t&rct=j&q=&esrc=s&source=web&cd=4&ved=0CDQQFjAD&url=http%3A%2F%2Fwww.mikrokontroler.pl%2Fsystem%2Ffiles%2FQuick%2520Start%2520Guide%2520for%2520Freedom%2520KL46Z%2520Rev1.pdf&ei=SfqLVdCFOIavsQHn9YCwCg&usg=AFQjCNFJaN-5iLfHjQ1YxkhTE059CXDkSw&bvm=bv.96782255,d.bGg)

**2. LED GPIO**

For this particular board you can see from the following figures that the LED is tied to the positive rail and the RED LED on the board is connected to PTE29 on the processor which is Port E pin 29.



Now you have know the Port as each peripheral has it's own clock, you can initialise the clock for Port E, this is done in **main** as follows:

**/\* Enable the clock gate \*/**

**SIM\_SCGC5 |= SIM\_SCGC5\_PORTE\_MASK;**

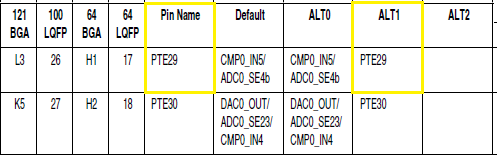
With the Clock enabled the following code is used to enable PTE29 as a GPIO setting it as an output so the LED can be used. The original value is then set as high so the LED is OFF.

**PORTE\_PCR29 |= PORT\_PCR\_MUX(0x01);**

**GPIOE\_PDDR  |= GPIO\_PDDR\_PDD(RED\_LED);**

**GPIOE\_PDOR  |= GPIO\_PDOR\_PDO(RED\_LED);**

The MUX value chooses the **ALT1** which is GPIO:



Using:

**for (;;) {**

**GPIOE\_PTOR |= GPIO\_PTOR\_PTTO(RED\_LED); //Toggle LED output**

**}**

It is now possible to get the LED to toggle, although this will not be needed in the main loop when the PIT works.

Two defines are used when it comes to using the LED:

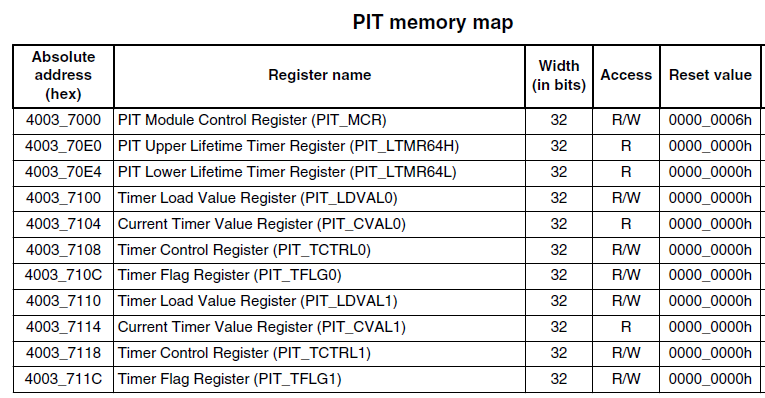
**#define RED\_LEFT\_SHIFT 29UL**

**#define RED\_LED (1<<RED\_LEFT\_SHIFT)**

These are used to create a 32 bit number that can be used to ensure that the Pin on Port E that is being made into GPIO outputs is Pin 29.

**3. PIT Setup**

When it comes to setup of the PIT there are a few things that need setting up for use which can be seen in the following table:



The things that need focusing on as follows;

* **PIT\_MCR –** need to ensure the Module Disable (MDIS) is not enabled
* **PIT\_LDVALn –** choose one of two registers (0 or 1) to load a value that is the timeout period before the interrupt occurs
* **PIT\_TCTRLn –** used to ensure control bits are enabled for specified timer
* **PIT\_TFLGn –** has the interrupt flag

The following code is used inside main to initialise the PIT:

**SIM\_SCGC6 |= SIM\_SCGC6\_PIT\_MASK; //PIT Peripheral Clock enable**

**NVIC\_EnableIRQ(*PIT\_IRQn*);**

**PIT\_TFLG0 |= PIT\_TFLG\_TIF(1);**

**PIT\_MCR &=! (PIT\_MCR\_MDIS\_MASK)**

**PIT\_TCTRL0 |= PIT\_TCTRL\_TEN\_MASK | PIT\_TCTRL\_TIE\_MASK;**

**PIT\_LDVAL0 |= PIT\_LDVAL\_TSV(PERIOD);**

Where PERIOD is defined as follows:

**#define PERIOD 10900000**

This defined value is used to produce an interrupt that occurs roughly every *1 second*.

You can also see from the code that the NVIC needs to be notified that there will be a PIT interrupt. The handler for the interrupt can then be used which clears the flag and toggles the LED. The clock for the PIT peripheral has also been enabled.

**void PIT\_IRQHandler (void)**

**{**

**PIT\_TFLG0 |= PIT\_TFLG\_TIF(1); //clear the flag**

**GPIOE\_PTOR |= GPIO\_PTOR\_PTTO(RED\_LED); //Toggle Red LED**

**}**

Now that every part of the code has been written the full working code is as follows:

**#define RED\_LEFT\_SHIFT 29UL**

**#define RED\_LED (1<<RED\_LEFT\_SHIFT)**

**#define PERIOD 10900000**

**void PIT\_IRQHandler (void)**

**{**

**PIT\_TFLG0 |= PIT\_TFLG\_TIF(1); //clear the flag**

**GPIOE\_PTOR |= GPIO\_PTOR\_PTTO(RED\_LED); //Toggle Red LED**

**}**

**int main(void)**

**{**

**SIM\_SCGC5 |= SIM\_SCGC5\_PORTE\_MASK**

**SIM\_SCGC6 |= SIM\_SCGC6\_PIT\_MASK;**

**PORTE\_PCR29 |= PORT\_PCR\_MUX(0x01);**

**GPIOE\_PDDR |= GPIO\_PDDR\_PDD(RED\_LED);**

**GPIOE\_PDOR |= GPIO\_PDOR\_PDO(RED\_LED);**

**NVIC\_EnableIRQ(*PIT\_IRQn*);**

**PIT\_TFLG0 |= PIT\_TFLG\_TIF(1);**

**PIT\_MCR &=! (PIT\_MCR\_MDIS\_MASK);**

**PIT\_TCTRL0 |= PIT\_TCTRL\_TEN\_MASK | PIT\_TCTRL\_TIE\_MASK;**

**PIT\_LDVAL0 |= PIT\_LDVAL\_TSV(PERIOD);**

**for (;;){};**

**return 0;**

**}**

Using this code it is then possible to produce an LED that changes state in 1 second intervals. This interval can be changed by simply changing the delay value.