Freescale FRDM-KL25Z

Sheng-Hsiu Lin

4/23/2013

1. Introduction

This paper is prepared for the 32-bit embedded systems course in 2013 spring semester. I will provide a hardware overview to describe the specifications and peripherals of KL25Z module. I will also describe methods to debug and to load programs onto the device. In addition, I will discuss the CodeWarrior integrated development environment (IDE) which I used for development. Then I will include three sample projects demonstrating how to develop project using bareboard, Process Expert, and MQXLite (Freescale RTOS) respectively. In the end, I will discuss what I think this board is good for, and my opinions on using it as a college level course learning material.

2. Hardware Overview

FRDM-KL25Z has the following main components:

- MKL25Z128VLK4 core chip
- OpenSDA debugging interface
- 8MHz crystal
- Capacitive touch slider
- MMA8451Q accelerometer
- Tri-color (RGB) LED
- I/O Connectors

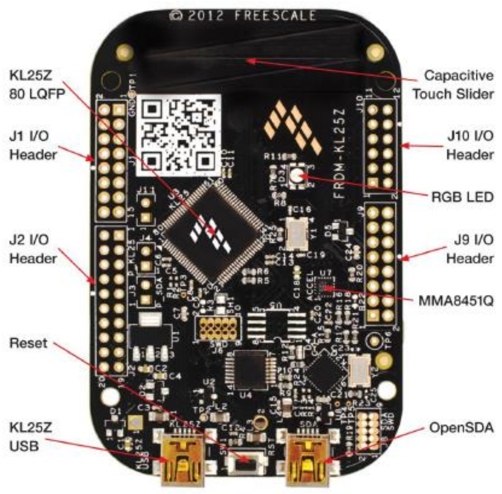


Figure 1-KL25Z design layout

2.1. MKL25Z128VLK4 core chip

MKL25Z128VLK4 is mounted in an 80-pin LQFP (Low Profile Quad Flat Package). It is a 32-bit ARM Cortex-M0+ core. It has 128KB program flash memory and 16KB RAM. The core clock runs on an external 8MHz crystal oscillator. It can scale clock up to 48MHz.

The followings are some peripherals of MKL25Z128VLK4 chip:

- Nine low-power modes
- 4-channel DMA (Dynamic Memory Access) controller
- COP software watchdog
- General purpose input/output (GPIO) ports
- 16-bit SAR ADC (successive approximation analog-to-digital convertor)
- 12-bit DAC (digital-to-analog convertor)
- One 6-channel TPM (Timer/PWM) and two 2-channel TPM. Each channel can be configured to perform one of the following three jobs:
  - Input capture
  - Output compare
  - PWM
- Two 8-bit SPI
- Two I2C
- One low power UART
- Two regular UART
- And more (for a complete list see [1]; for details of each peripheral see [2]).

2.2. OpenSDA debugging interface

OpenSDA is an open standard serial and debug adapter. It is used to bridge serial and debug communications between KL25Z and a USB host such as the computer used to develop KL25Z.

OpenSDA is developed by P&E Microcomputer Systems for Freescale Semiconductor, Inc. The software application is embedded in Freescale CodeWarrior IDE.

OpenSDA features a bootloader mechanism. Users can connect KL25Z via OpenSDA to computer USB port and access OpenSDA bootloader as a flash drive. User can load program onto KL25Z by simply moving their precompiled binary program onto the drive.

Otherwise, program can also be loaded onto KL25Z by using CodeWarrior IDE (explained later). See [3] and [4] for more information about OpenSDA.

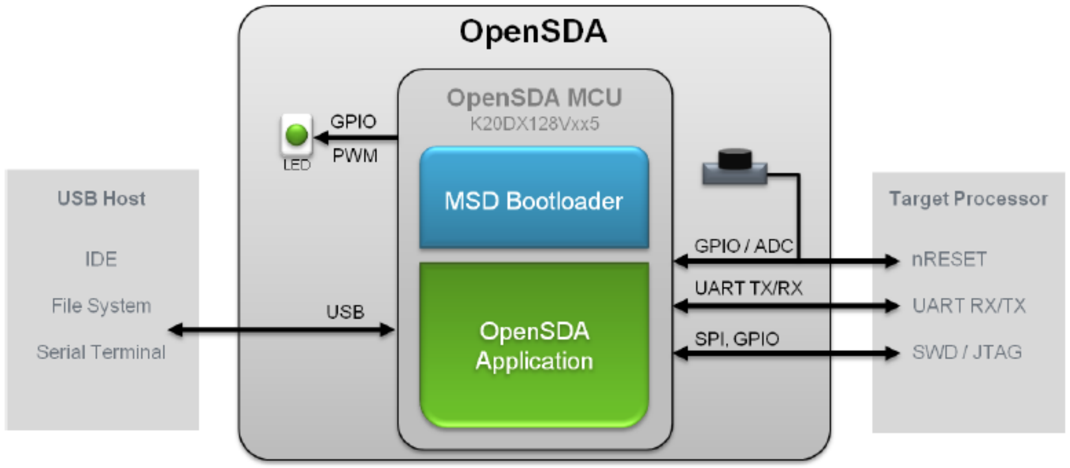


Figure 2- OpenSDA

2.3. Capacitive touch slider, accelerometer, and tri-color LED

KL25Z has several physical I/O devices built in on the board. It has a capacitive touch slider, a three-axis accelerometer (model name: Freescale MMA8451Q), and a tri-color LED. These devices are great for developers to learn to program the board.

- Touch slider inputs can be read from the two TSI (touch sense input) signals
  - TSI0_CH9 and
  - TSI0_CH10.
- The tri-color LED can be controlled by accessing
  - PTB_CH18 (red),
  - PTB_CH19 (green), and
  - PTD_CH1 (blue).
- Accelerometer inputs are located at
  - PTE_CH24, PTE_CH25 and
  - PTA_CH14, PTA_CH15.

3. Freescale CodeWarrior for MCUs

CodeWarrior for MCUs is a Freescale integrated development environment based on the Eclipse open development platform. It integrates the development tools for the ColdFire, DSC, Kinetics, Qorivva, PX, RS08, S08 and S12Z architectures into a single development environment. There are several licensing options available. Even the least basic edition costs more than $2,000 for one license. Fortunately, there is a special edition. This edition is free and does not have an evaluation time limit. The only limits are on code size and some special functionalities which we do not need (at least for the projects covered in this report). See [5] for detail licensing options.

CodeWarrior for MCUs supports codes in C, C++ and assembly languages. It has a Freescale component library and component inspector built in. They provide means to add, view and edit Freescale components such as a CPU.

CodeWarrior for MCUs provides a comprehensive debugging environment. The debugging connection can be made through

- P&E USB Multilink
- P&E Cyclone MAX
- Open Source SDA and
- Segger J-Link

We will only be using Open Source SDA as it is embedded in KL25Z.

Inside the debugging environment, we are able to monitor variables, registers, breakpoints, memory and modules. In addition, we can set break points in both C or assembly codes, and we can step through them. It has a built in terminal that can be configured to interact with the serial communication on KL25Z. There is a console that shows the current status.

See [6] and [7] for more information on getting started with CodeWarrior for MCUs.

4. Sample Projects

In this chapter I will present four projects. They hopefully will demonstrate various capabilities of KL25Z, and give a sense of how to program them.

- **Project 1 – BB_LED:**
    - This is a **bare-board project** that changes the color on the tri-color LED repeatedly. It shows how to (1) create a bare-board project, (2) include peripheral declarations (mapping port names to corresponding register memory location), and (3) read/write GPIO Ports.
- **Project 2 – PE_Serial_LED:**
    - This is a **Process Export project** that changes the color on the tri-color LED according to serial communication input. It shows how to (1) create a Process Expert project, (3) add components, (4) configure components, and (5) create, read from and write to a serial communication.
- **Project 3 – PE_PWM_LED:**
    - This is a **Process Export project** that generates PWM pulses, and uses the pulses to control LED. In addition to Project 2, it shows how to configure clock and generate PWM pulse.
- **Project 4 – MQX_Acce_LED:**
    - This is a **Freescale RTOS MQXLite project**. This project uses MQXLite to manage tasks that read inputs from the accelerometer and change the color on the tri-color LED. It shows how to (1) create an MQXLite project, (2) configure MQXLite to work with KL25Z, (3) read inputs from the accelerometer, and (4) program MQXLite task.

The complete source code of all above projects are zipped and attached. The file names are same as the project names.

4.1 Project 1 – BB_LED

To start a bare-board project, in CodeWarrior, click File -> New -> Bareboard Project. Type BB_LED in to project name and click Next. Now we need to select device to be used. In our case, choose Kinetics L Series -> KL2x family -> KL25Z family -> MKL25Z128. Select Application as project type. Next step we need to choose what type of connection to use. Select Open Source SDA only. On the next page, select the following options:

- Language: C

- Floating Point: Software
- I/O Support: UART (default)
- ARM Build Tools: GCC

On the next step, it asks whether we want to use any Rapid Application Development. There are three options:

- None: this option will generate startup code for the device. However, no device initialization code will be generated. We will need to write initialization code for each on-chip peripheral in order to use it.
- Device Initialization: this option will generate not only startup code for the device, but also initialization code for on-chip peripherals. In addition, selecting it will generate interrupt vector table and interrupt service routine template.
- Process Expert: PE project will be discussed in later examples.

We select None for this project. Click on Finish. When CodeWarrior finished creating project, you should see Figure 3 on the left of your screen.

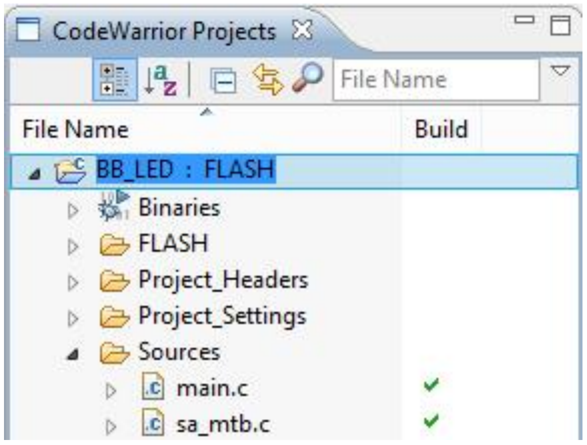


Figure 3 - BB_LED Project Structure

We will be editing and writing our code to main.c file. Double click on main.c to open it.

Next I will explain the code in main.c file provided in the sample project. It is a good idea to open that file while reading the following section.

On the top we have a delay() function. This function let the device wait for 1,000,000 ticks. It is used later as a time interval to change color on the tri-color LED.

Then we have three toggle function toggle_red(), toggle_green() and toggle_blue(). These functions switch on/off of the PTOR registers for each corresponding tri-color LED port channel. For example, the red LED is controlled by GPIO Port B Channel 18 as mentioned earlier. Hence toggle_red() sets GPIOB_PTOR = (1 << 18). When a toggle function is called, it switches on/off the corresponding color LED.

Inside of the main() function, we first initialize registers such as disabling COP, enabling clocks for PORTB and PORTD, set the corresponding Port Channel to output, and etc. For detail, see the comments in code.

In the end is our while forever loop. We toggle each color in turn:

Red On – wait – Red Off & Green On – wait – Green Off & Blue On – wait – Blue Off

To compile the project, single click on the project then click on Build icon on top.

Click on debug to debug the project.

4.2 Project 2 – PE_Serial_LED

Creating a PE project is very similar to creating a bareboard project. The only difference is that we select Process Expert in the last step of the project creation wizard.

Process Expert is a Freescale product. It is designed for rapid application development of embedded applications for Freescale MCUs. A PE project is created from embedded components. Embedded components are the basic elements of embedded systems. They include, for example, CPU core, CPU on-chip peripherals (PWM, Timer, SPI, I2C and UART), software algorithm and etc. There is an extensible embedded component library maintained by Freescale.

In addition to providing components as objects, Processor Expert "suggests, connects, and generates the drivers for embedded system hardware, peripherals, or algorithms".

For more information about Process Expert, see [8].

You should see Figure 4 after CodeWarrior finished creating the project. Double click ProcessorExpert.c file to open it. We will write our code in this file instead of creating a main.c file as in bareboard project.

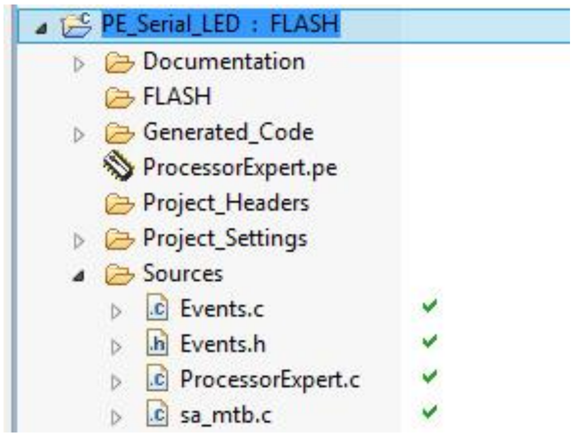


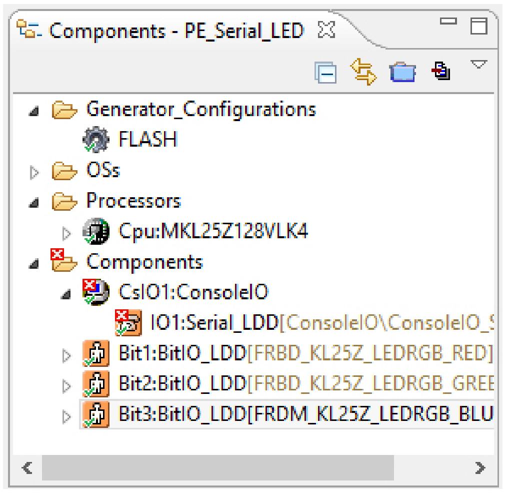
Figure 4 - Process Expert project structure

Now we need to add the necessary components to our project. Click on Processor Export -> Show Views and you should see a Component Library tab. All provided components are listed here. In this project, we will need the following:

- LEDRGB_RED:BitIO_LDD
- LEDRGB_GREEN:BitIO_LDD
- LEDRGB_BLUE:BitIO_LDD
- CsIO1:ConsoleIO

Find each component in the Component Library. Double click to add to project. When you are done, you should see Figure 5 in your Components section.

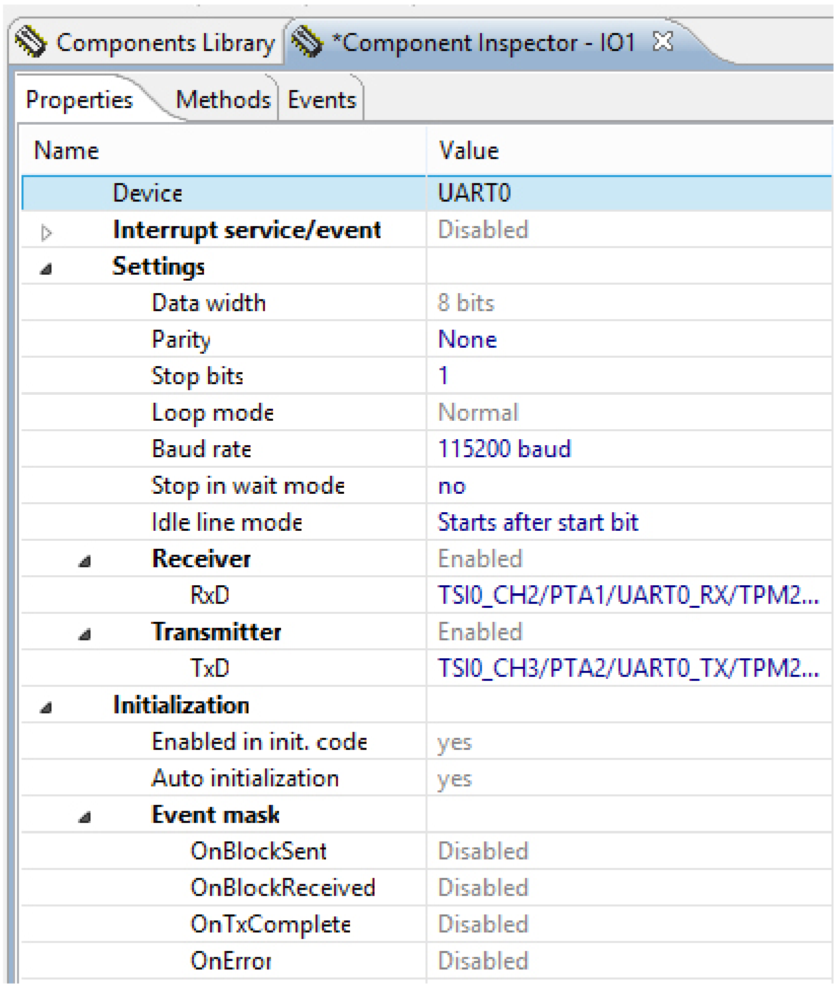
Next step is to configure each component so that they will work as we wish. Single click on IO1:Serial_LDD to select it, and click on Component Inspector on the right side of the window to see the detail configurations. Set the values according to Figure 6.



Components - PE_Serial_LED

- Generator_Configurations
  - FLASH
- OSs
- Processors
  - Cpu:MKL25Z128VLK4
- Components
  - CsIO1:ConsoleIO
    - IO1:Serial_LDD[ConsoleIO\ConsoleIO_S
  - Bit1:BitIO_LDD[FRBD_KL25Z_LEDRGB_RED]
  - Bit2:BitIO_LDD[FRBD_KL25Z_LEDRGB_GREE
  - Bit3:BitIO_LDD[FRDM_KL25Z_LEDRGB_BLU

Figure 5 - PE Components



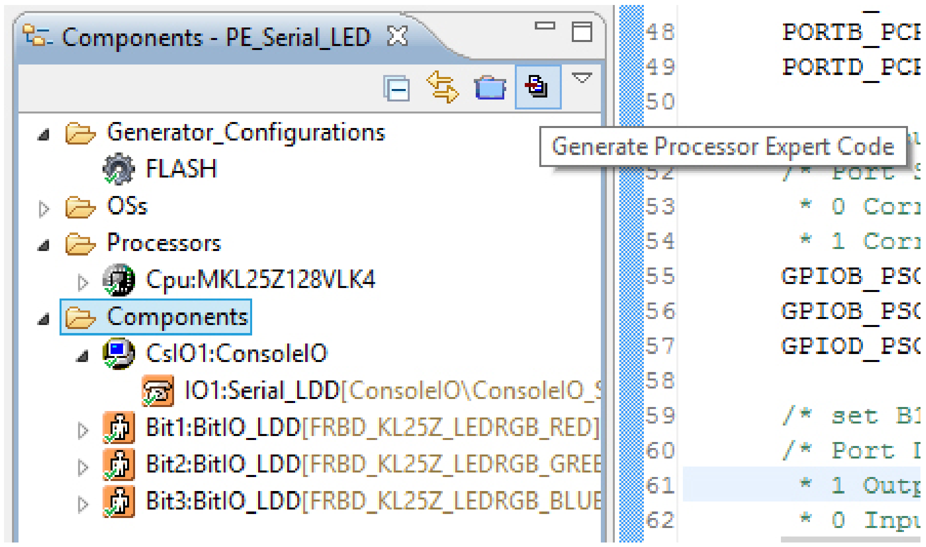
Name	Value
Device	UART0
Interrupt service/event	Disabled
Settings	
Data width	8 bits
Parity	None
Stop bits	1
Loop mode	Normal
Baud rate	115200 baud
Stop in wait mode	no
Idle line mode	Starts after start bit
Receiver	Enabled
RxD	TSI0_CH2/PTA1/UART0_RX/TPM2...
Transmitter	Enabled
TxD	TSI0_CH3/PTA2/UART0_TX/TPM2...
Initialization	
Enabled in init. code	yes
Auto initialization	yes
Event mask	
OnBlockSent	Disabled
OnBlockReceived	Disabled
OnTxComplete	Disabled
OnError	Disabled

Figure 6 - Serial Communication Setting

Similarly for the tri-color LED. Make sure the I/O pin are set correctly for each LED:

- Red LED:
  - Pin for I/O: TSI0_CH11/PTB18/TPM2_CH0
- Green LED:
  - Pin for I/O: TSI0_CH12/PTB19/TPM2_CH1
- Blue LED:
  - Pin for I/O: ADC0_SE5b/PTD1/SPI0_SCK/TPM0_CH1

Once we are all done, click on the `Generate Processor Expert Code` icon on the top right corner of `Components` view (see Figure 7).


Components - PE_Serial_LED
Generate Processor Expert Code
Generator_Configurations
FLASH
OSs
Processors
Cpu:MKL25Z128VLK4
Components
CsIO1:ConsoleIO
IO1:Serial_LDD[ConsoleIO\ConsoleIO_S
Bit1:BitIO_LDD[FRBD_KL25Z_LEDRGB_RED]
Bit2:BitIO_LDD[FRBD_KL25Z_LEDRGB_GREE
Bit3:BitIO_LDD[FRBD_KL25Z_LEDRGB_BLUE

```
    PORTB_PCR
    PORTD_PCR


    /* Port S
     * 0 Corr
     * 1 Corr
    GPIOB_PSO
    GPIOB_PSO
    GPIOD_PSO

    /* set Bi
    /* Port I
     * 1 Outp
     * 0 Inpu
```

Figure 7 - Generate Processor Expert Code

Now we are ready to add our code to `ProcessorExpert.c` file. Please see the full working code in the provided sample project source. Following is a list of explanation to each function called in `ProcessorExpert.c` file:

- Blue_LED_ClrVal(Blue_LED_DeviceData): turn on blue LED
- Green_LED_SetVal(Green_LED_DeviceData): turn off green LED
- InpData[0] = uart_getchar(TERM_PORT_NUM): read one char from UART
- uart_putchar(TERM_PORT_NUM, InpData[0]): write one char to UART

Copy and paste the code in `ProcessorExpert.c` to your `ProcessorExpert.c` file. Build and run, the device should perform as described.

4.3. Project 3 – PE_PWM_LED

The technique of making this project is exactly the same as the technique of making Project 2. For information on PWM configuration see [1], [2] and [9]. See attached sample project for codes and comments.

4.4. Project 4 – MQX_Acce_LED

MQX is a RTOS (real-time operating system) developed by Freescale. There are two editions: MQX and MQX Lite.

MQX is more powerful and is designed for higher end microprocessors such as ColdFire. MQX provides communication protocols such as USB and TCP/IP. In addition, it has a file system. It is provided free to customers who bought MQX enabled microprocessors. See Freescale website [10] for more information and MQX download link. See [12] for information on creating a simple MQX application.

MQX Lite has less capabilities but consumes less resources. It is designed for lower level microprocessor such as our KL25Z. MQX Lite is included in CodeWarrior MCU IDE as a Processor Expert component (i.e. you can add MQX Lite as a component to your project). A minimum MQX Lite can be as small as 4KB. Similar to MQX, it can be configured to perform real-time, priority-based preemptive multitasking.

See [15] and [16] for more information on MQX and MQX Lite.

This project is replicate of [11]. In this project, we will create a MQX Lite application that reads inputs from the accelerometer on our KL25Z board, and adjust the color on tri-color LED according to the physical orientation of the board.

There are two ways to create a MQX Lite project. You can either click on `File -> New -> MQX-Lite Project` or create a `Processor Expert Project` and then add MQX-Lite as a component.

Then we need to add these components to our project:

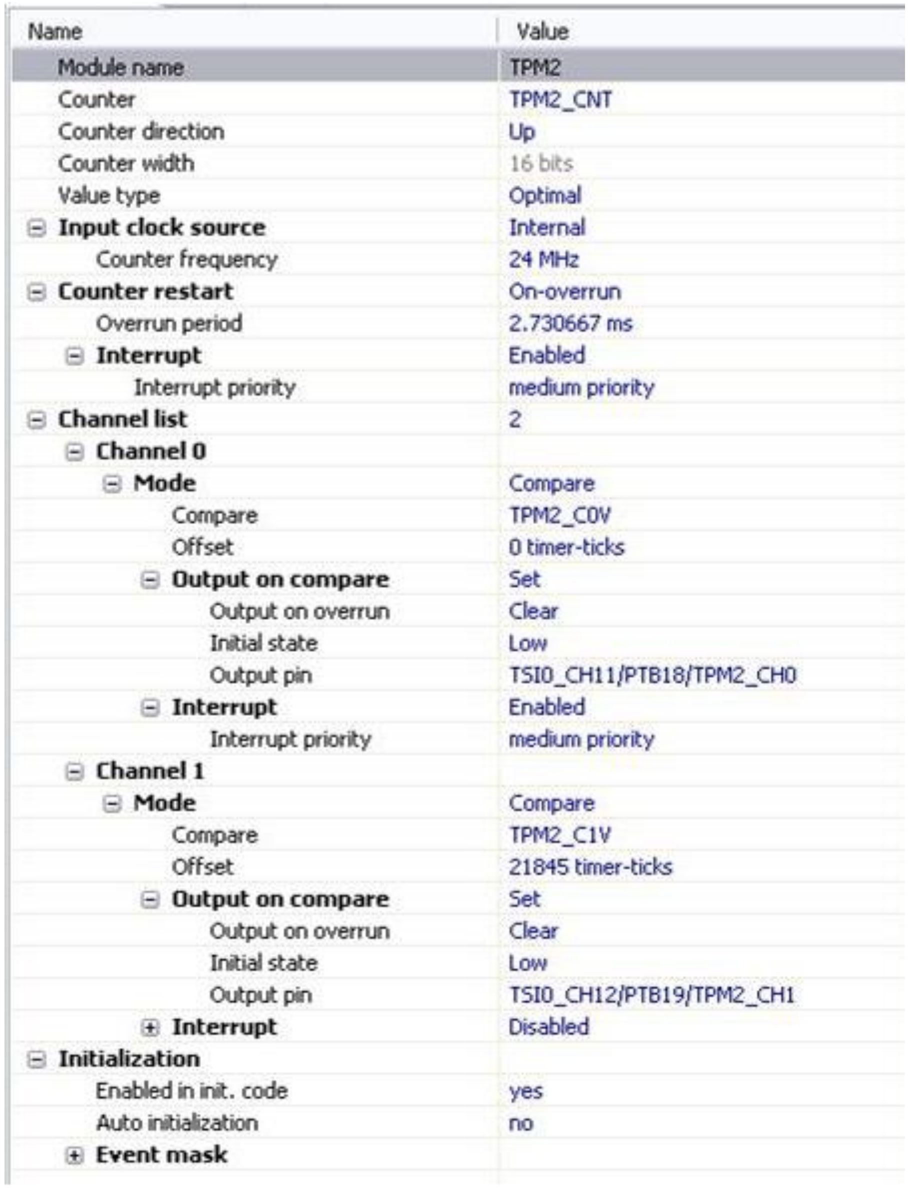
- ConsoleIO
- Two TimerUnit_LDD
- I2C_LDD

Next we use component inspector to edit their properties to the following:

- ConsoleIO

Device:	UART0
Baud rate:	38400 baud
RxD:	TSI0_CH2/PTA1/UART0_RX/TPM2_CH0
TxD:	TSI0_CH3/PTA2/UART0_TX/TPM2_CH1

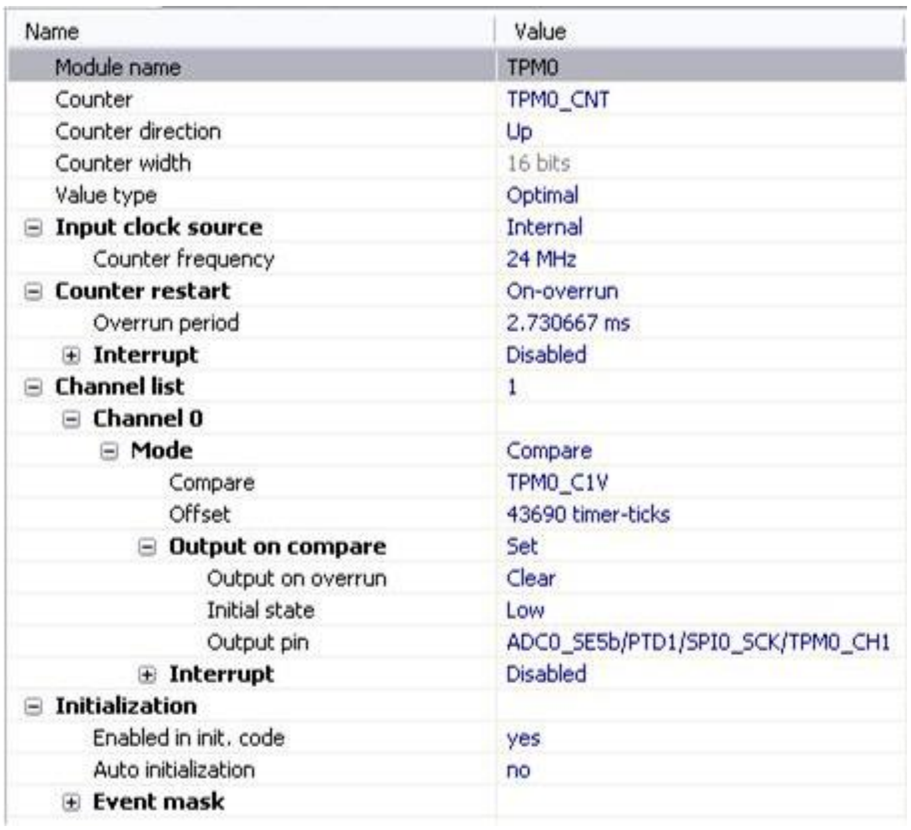
- TimerUnit1



Name	Value
Module name	TPM2
Counter	TPM2_CNT
Counter direction	Up
Counter width	16 bits
Value type	Optimal
⊟ **Input clock source**	Internal
Counter frequency	24 MHz
⊟ **Counter restart**	On-overrun
Overrun period	2.730667 ms
⊟ **Interrupt**	Enabled
Interrupt priority	medium priority
⊟ **Channel list**	2
⊟ **Channel 0**	
⊟ **Mode**	Compare
Compare	TPM2_C0V
Offset	0 timer-ticks
⊟ **Output on compare**	Set
Output on overrun	Clear
Initial state	Low
Output pin	TSI0_CH11/PTB18/TPM2_CH0
⊟ **Interrupt**	Enabled
Interrupt priority	medium priority
⊟ **Channel 1**	
⊟ **Mode**	Compare
Compare	TPM2_C1V
Offset	21845 timer-ticks
⊟ **Output on compare**	Set
Output on overrun	Clear
Initial state	Low
Output pin	TSI0_CH12/PTB19/TPM2_CH1
⊞ **Interrupt**	Disabled
⊟ **Initialization**	
Enabled in init. code	yes
Auto initialization	no
⊞ **Event mask**	

Figure 8 - TimerUnit1 setting. Channel 0 controls Red LED and Channel 1 controls Green LED.

- TimerUnit2



Name	Value
Module name	TPM0
Counter	TPM0_CNT
Counter direction	Up
Counter width	16 bits
Value type	Optimal
⊟ **Input clock source**	Internal
Counter frequency	24 MHz
⊟ **Counter restart**	On-overrun
Overrun period	2.730667 ms
⊞ **Interrupt**	Disabled
⊟ **Channel list**	1
⊟ **Channel 0**	
⊟ **Mode**	Compare
Compare	TPM0_C1V
Offset	43690 timer-ticks
⊟ **Output on compare**	Set
Output on overrun	Clear
Initial state	Low
Output pin	ADC0_SE5b/PTD1/SPI0_SCK/TPM0_CH1
⊞ **Interrupt**	Disabled
⊟ **Initialization**	
Enabled in init. code	yes
Auto initialization	no
⊞ **Event mask**	

Figure 9 - TimerUnit2 setting. Channel 0 controls Blue LED.

- I2C
  - Target slave address: 1D
  - Internal frequency: 24MHz
  - Bits 0-2: 101
  - Bits 3-5: 100

When we finish configuring all components click on generate processor expert code icon and build the project.

When we created a MQX Lite project, a default Task1 is generated automatically. We can find it under MQXLite component (Figure 10). Double click Task1 to open the source code. We will add our code to this file. In a bigger project, we can add as many tasks as we need to and write codes in each corresponding files.

See provided sample projects for Task1 codes and comments.

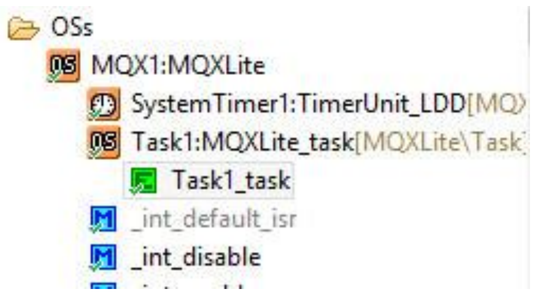

OSs
MQX1:MQXLite
SystemTimer1:TimerUnit_LDD[MQ
Task1:MQXLite_task[MQXLite\Task
Task1_task
_int_default_isr
_int_disable

Figure 10 - Default Task1

To discover more KL25Z sample projects, see [13] and [14].

5. Conclusion: Teaching with FRDM-KL25Z

- Pros:
    - KL25Z is cheap: the board costs $12 and a USB cable costs about $5.
    - Good for embedded programming beginner:
        - Easy hardware setup
        - Compact
    - Powerful and reliable:
        - Has enough computational power for projects
        - Has many peripherals embedded (see Chapter 2 for list)
        - The board has been very reliable for as long as I had it
        - Ready-to-use RTOS
- Cons:
    - Hardware not easily extensible:
        - No hardware developing area
        - Need to manually solder headers
    - Using Processor Expert rapid application development may not allow student to learn much about the hardware.

As shown in my sample projects, KL25Z has many peripherals for students to play with. However, using them in a more practical application will very likely need some hardware modifications. For example, if we were to use PWM to control motor, we would need to solder a development area on to the board. Nonetheless, I think this board is a good candidate for teaching.

References

All of the following reference documents are provided in reference.zip except those listed with hyperlink.

[1] KL25 Sub-Family Data Sheet.

[2] Kinetis L Peripheral Module Quick Reference.

[3] OpenSDA User’s Guide.

[4] FRDM-KL25Z User's Manual.

[5] Quick Reference Guide for CodeWarrior Suites.

[6] CodeWarrior Development Studio for Microcontrollers V10.x Getting Started Guide.

[7] CodeWarrior Development Studio for Microcontrollers V10.3 Quick Start Guide.

[8] Processor Expert User Manual.

[9] KL25 Sub-Family Reference Manual.

[10] Freescale MQX™ Software Solutions. http://www.freescale.com/webapp/sps/site/homepage.jsp?code=MQX_HOME

[11] Writing your First MQX-Lite Application.

[12] Writing Your First MQX Application.

[13] FREEDOM-KL25Z Sample Code Guide for CodeWarrior.

[14] FRDM-KL25Z and TWRKL25Z48M Sample Code Package. http://cache.freescale.com/files/32bit/software/KL25_SC.exe?fr=gtl

[15] Freescale MQX Real-Time Operating System User’s Guide.

[16] Freescale MQX Software Solutions.