Writing you First MQX-Lite Application

by: Luis Garabito
Applications Engineer
TICS, Mexico

1 Introduction

The time invested to develop applications for the first time in a new environment can be significant. It is necessary to understand how the environment works and then be able to generate applications for this environment.

The purpose of this application note is to provide the knowledge that enables developers to start quickly and easily with the development of their first application on Freescale MQX-Lite RTOS.

This document provides the bases that developers will need to understand to create basic Freescale MQX-Lite applications.

This Application Note is based on the KL2: Kinetis KL2 USB MCUs Family, specifically, the KL25Z128VLK4 micro controller. The Freescale Freedom development platform board (FREEDOM – KL25Z) is also used for this example.

2 CodeWarrior 10.3 Start up

Once the CodeWarrior 10.3 is installed and started it is necessary to define a workspace where all the projects are allocated and administrated by CodeWarrior. In Figure 1 the workspace “D:\workspace_MQXLite” is configured.

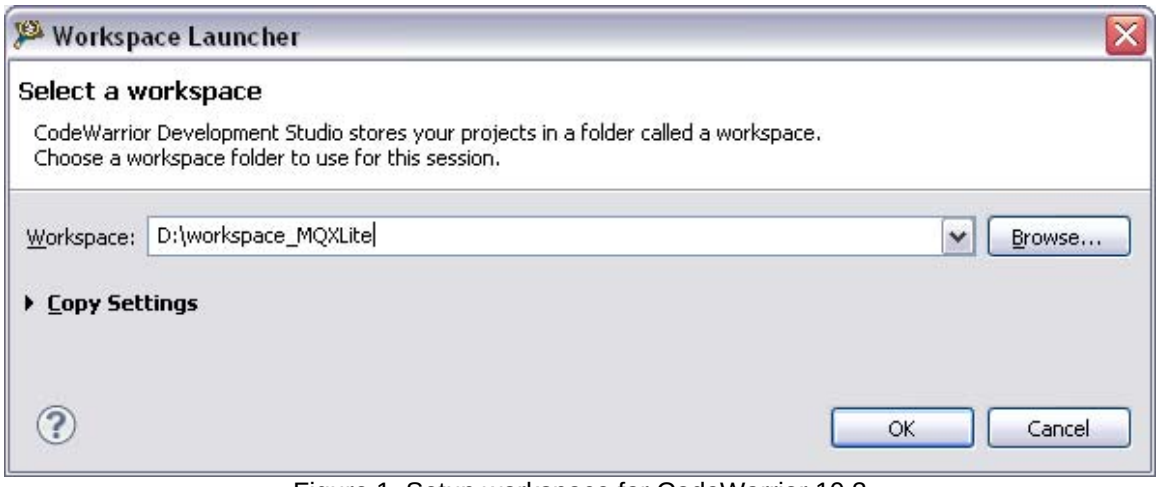


Figure 1. Setup workspace for CodeWarrior 10.3

Figure 2 shows CodeWarrior appearance when it finishes the start up. One of the new features in CodeWarrior 10.3 version is the Commander window located in the bottom right of the screen.

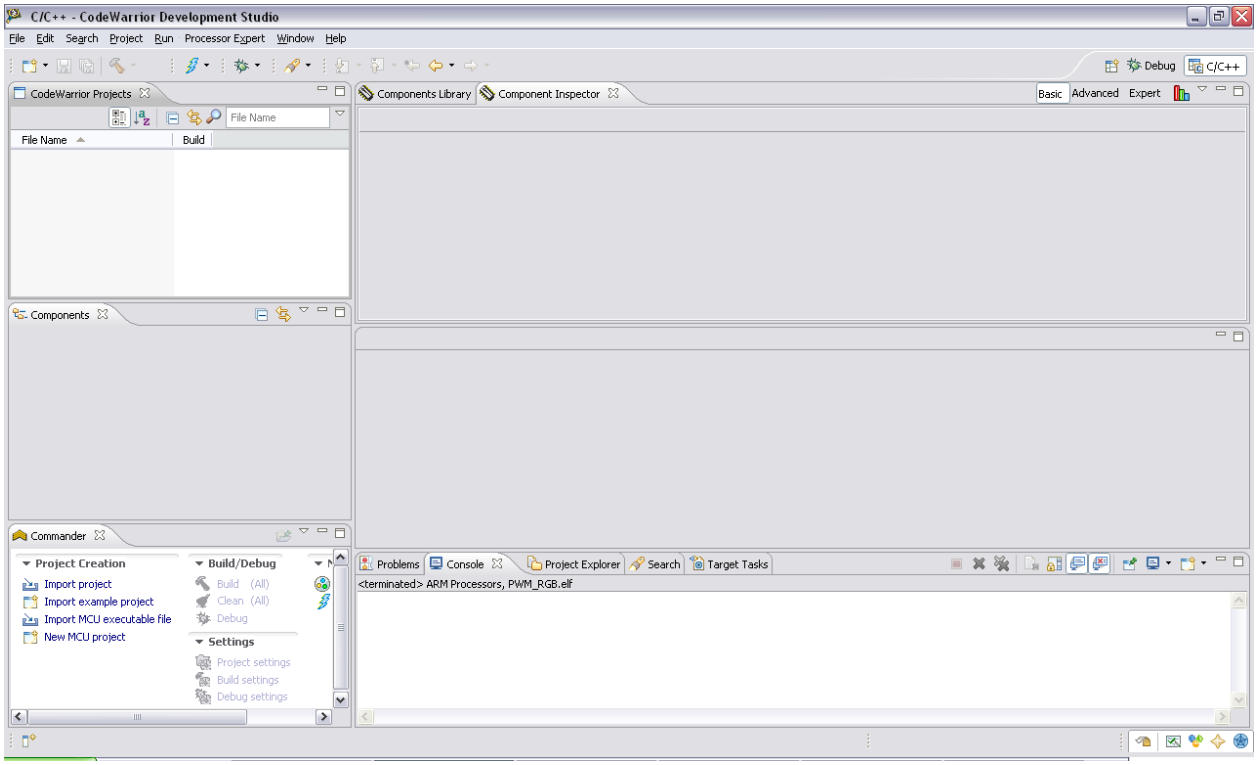


Figure 2. CodeWarrior 10.3

3 Creating projects

There are two possible paths to follow. The first path is to create an MQX-Lite project and the second is to create from zero. Both options are described in this document. The first option is using the wizard to create an MQX-Lite project.

3.1 Creating a MQXLite base project

The next steps are used to create and configure a MQX-Lite project for the MKL25Z128 (48 MHz) derivative:

1. The Commander window includes an option to create a new project for MQX-Lite. Figure 3 shows this option.

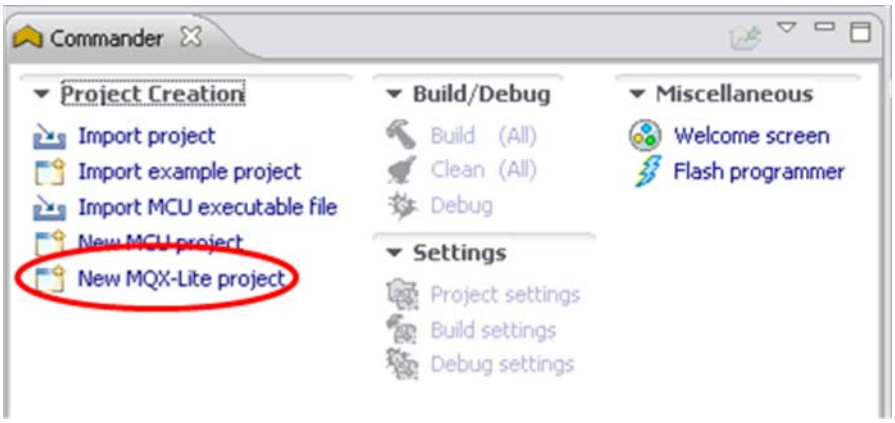


Figure 3. Commander window

2. The “New MQX-Lite Project” wizard starts and requests for a name for the project and a location. In this case the name “MQX-Lite_Freedom” and “D:\workspace_MQXLite” are used respectively. To finish this step click in the “Next >” button at the bottom of the window. Figure 4 demonstrates this.

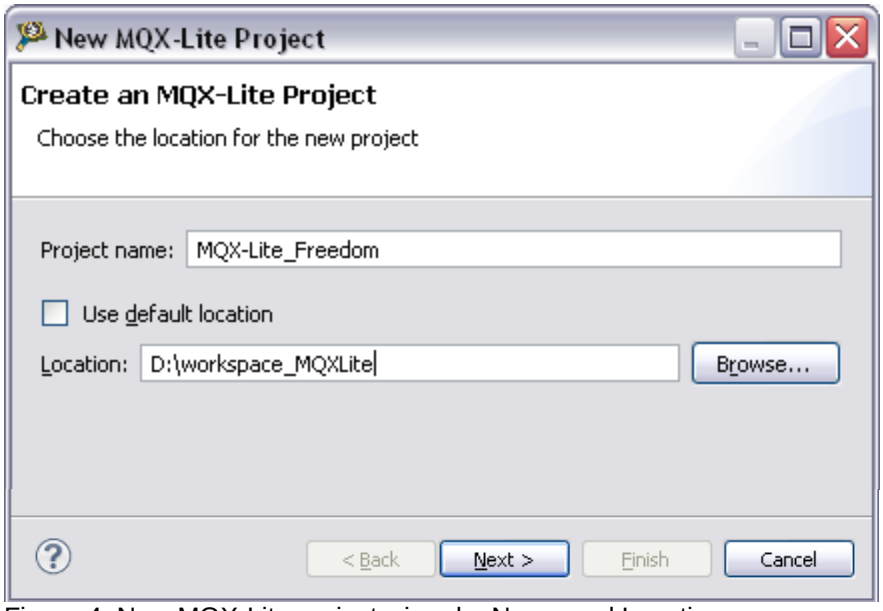


Figure 4. New MQX-Lite project wizard – Name and Location

3. The next window in the wizard allows selecting the desired MCU. For this case the MKL25Z128 (48 MHz) derivative is selected. To perform this selection go and navigate in the tree as Figure 5 shows. Once the MKL25Z128 option is selected click the “Next >” button.

Figure 5. Kinetis L family tree

4. The following step lets you choose the connection to be used. The Freedom – KL25Z board uses the Open source SDA connection and this must be selected as Figure 6 explains. Click “Next >” to go on.

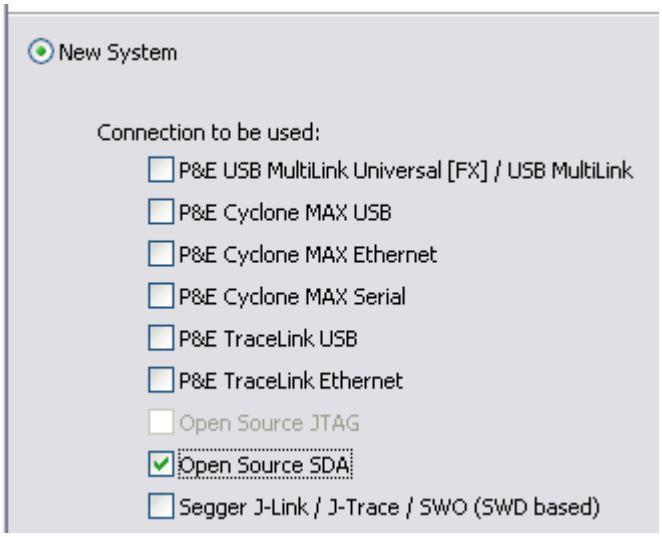


Figure 6. Selecting connection

5. The next window has the options set by default. For this window we just need to click in the “Finish” button as Figure 7 illustrates.

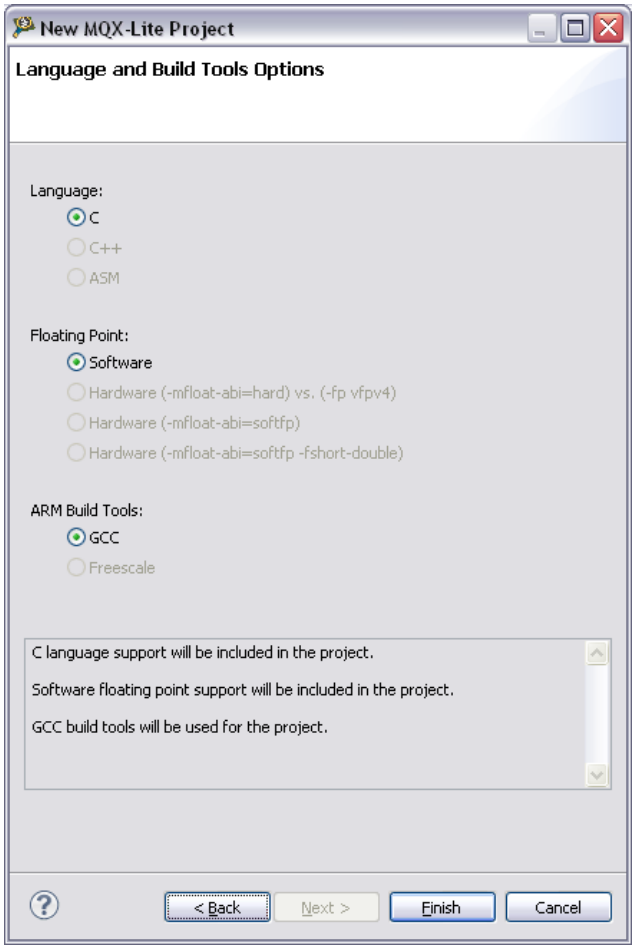


Figure 7. Programming language and build options selection

6. When the wizard finishes CodeWarrior has a new appearance that looks like figure 8. There are two main changes. The “CodeWarrior Projects” window has a new project included and the “Components” window shows the Processor Expert added components.

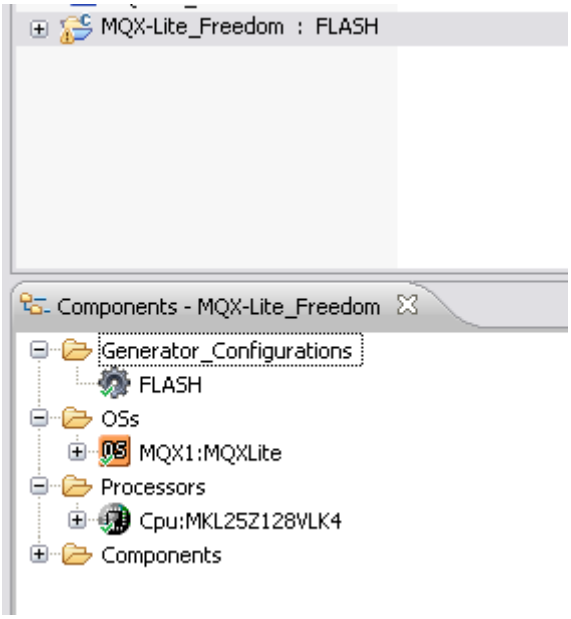


Figure 8. “CodeWarrior Projects” and “Components” windows

3.2 Creating a empty base project

The next steps are used to create and configure a project for the MKL25Z128 (48 MHz) derivative:

7. The Commander window includes an option to create a new project for MCU. Figure 9 shows this option.

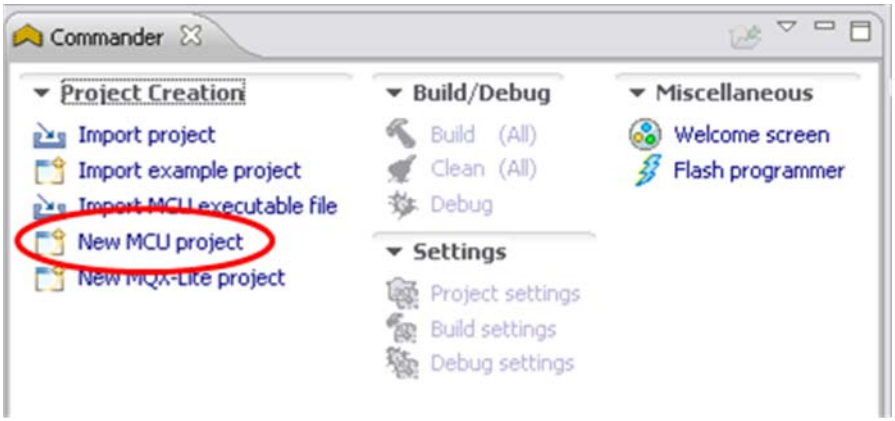


Figure 9. Commander window

8. The “New Bareboard Project” wizard starts and requests for a name for the project and a location. In this case the name “MQXLite_Freedom” and “D:\workspace_MQXLite” are used respectively. To finish this step click in the “Next >” button at the bottom of the window. Figure 10 demonstrates this.

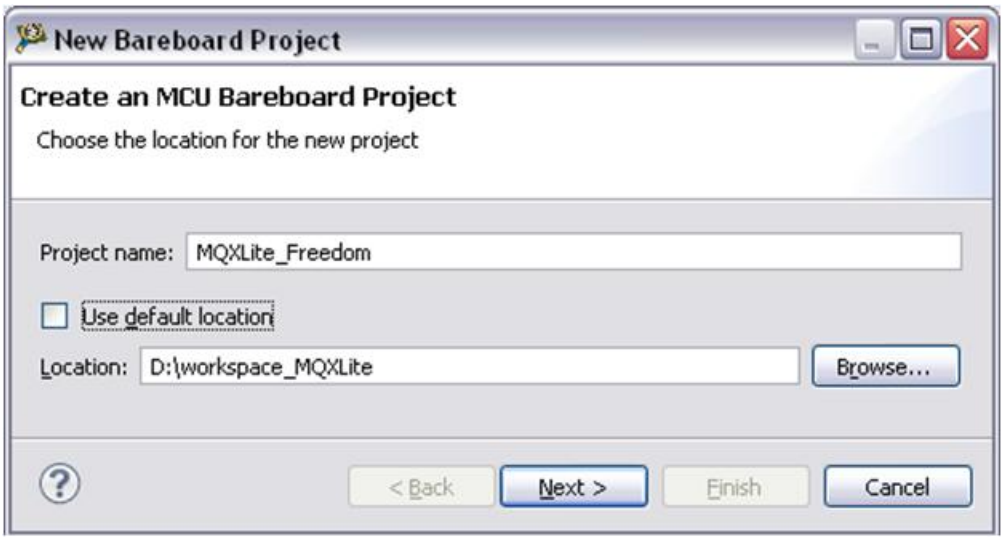


Figure 10. New project wizard – Name and Location

9. The next window in the wizard allows selecting the desired MCU. For this case the MKL25Z128 (48 Mhz) derivative is selected. To perform this selection go and navigate in the tree as Figure 11 shows. Once the MKL25Z128 option is selected click the “Next >” button.

Figure 11. Kinetis L family tree

10. The following step lets you choose the connection to be used. The Freedom – KL25Z board uses the Open source SDA connection and this must be selected as Figure 12 explains. Click “Next >” to go on.

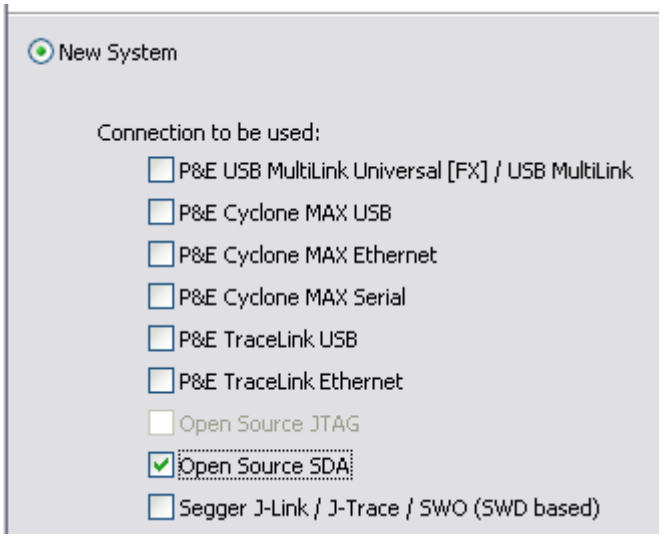


Figure 12. Selecting connection

11. The next window has the option to select the programming language to be used. The C language must be selected as Figure 13 illustrates.

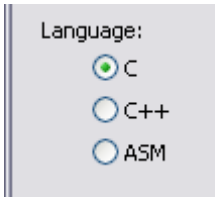


Figure 13. Programming language selection

12. The final step is to choose the rapid application development to be used. In this case the option “Processor Expert” should be selected. Processor Expert can generate for you all the device initialization code. It includes many low-level drivers. The rest of the options should remain with the default settings. To end the wizard go and click the “Finish” button.

Figure 14. Selecting Rapid Application Development

13. When the wizard finishes CodeWarrior has a new appearance that looks like figure 15. There are two main changes. The “CodeWarrior Projects” window has a new project included and the “Components” window shows the Processor Expert added components.

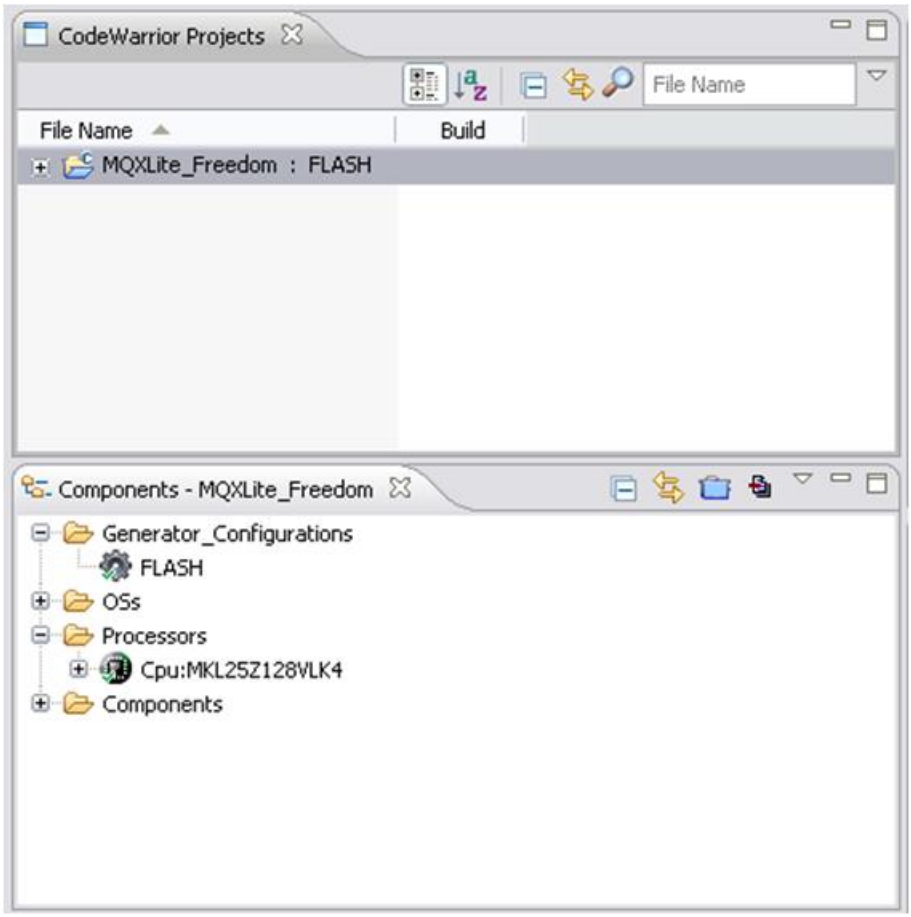


Figure 15. “CodeWarrior Projects” and “Components” windows

4 Configure the project

First, it is necessary to setup the right configurations for the hardware used in the Freedom – KL25Z board. Figure 16 has all the needed steps to configure the clock settings and configurations.

1. Double click in the “ProcessorExpert.pe” item under the “CodeWarrior” project window. This will update the content of the “Components” window where “Cpu: MKL25Z128VLK4” component must be selected to enable the content in the “Component Inspector,” right of the windows mentioned above. There are 5 things that need to be modified. Figure 16 shows, in detail, the options and the right values to be modified.

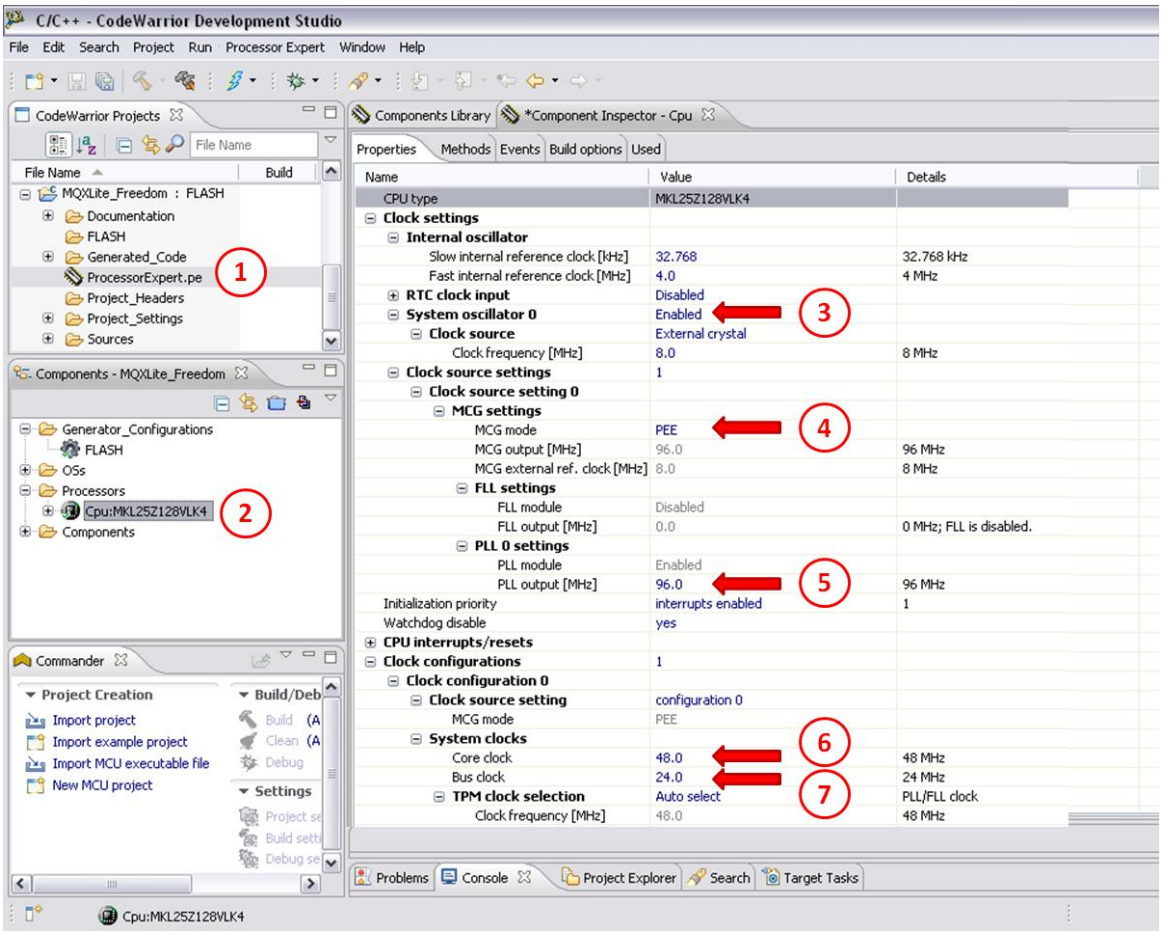


Figure 16. Clock settings and configurations

2. After the changes it is possible to generate code using Processor Expert. To do this click in the generate code button located in the up right corner of the “Components” window. Figure 17 illustrates this.

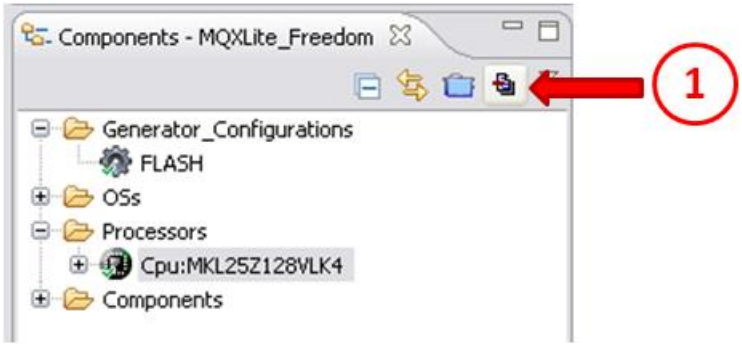


Figure 17. Generating Processor Expert code

3. It is necessary to compile the project to verify that there are no errors while using the generated code. To compile the project go to the “CodeWarrior Projects” window. Click in the Project name and then click in the compile button. Figure 18 shows the steps.

Figure 18. Compiling the project

4. Validate that the “Problems” tab has no errors listed to ensure a success compilation. Figure 19 shows a “problems” tab with 0 error or warnings.

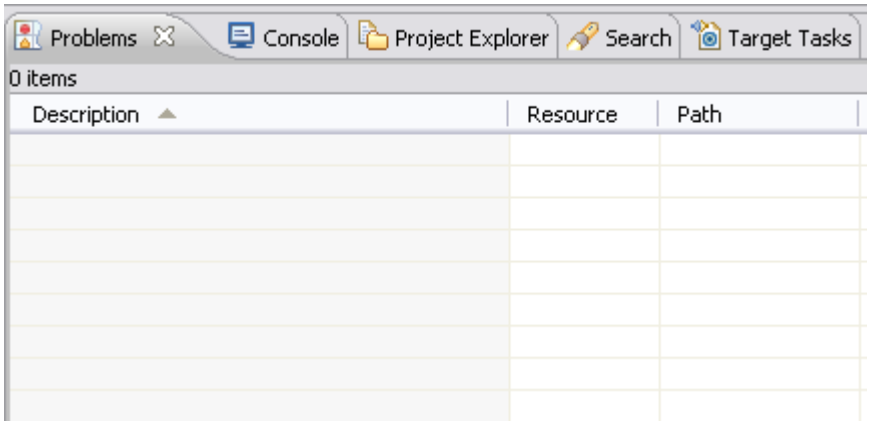


Figure 19. Problems tab with 0 errors

5 Your first Freescale MQX-Lite Application

The architecture of the application is to configure three PWM timers for the RGB LED and the communication using I2C to read data from the inertial sensor, MMA8451Q. Each one of the PWMs will control each one of the LEDs in the RGB LED. The I2C is used to read the data coming from the inertial sensor. The RGB LED color changes while the board is moving. If the board stops moving the RGB LED holds the current color.

The Stationery project is now ready to add the MQX-Lite application. The first step is to add a console that can help to debug and output data from the MCU using the UART port. To add and configure the Processor Expert component is necessary to follow these steps.

1. Click in the “Components Library” tab as figure 20 shows. Then click in Alphabetical tab and look for the “ConsoleIO” component. Double clicking in the “ConsoleIO” component

includes it in the project. This is illustrated in Figure 21.

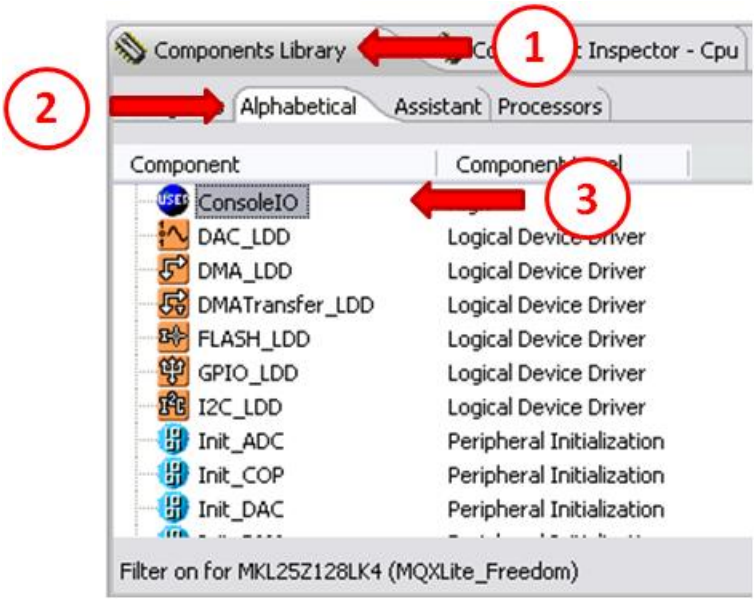


Figure 20. Components Library

Figure 21. New component added to Processor Expert project: ConsoleIO

2. It is necessary to configure the new added component. To do this expand the “CsIO1:ConsoleIO” component just added in the step before. By clicking in the “IO1:Serial_LDD[ConsoleIO\ConsoleIO_Serial_LDD]” option from the “CsIO1:ConsoleIO” component (look figure 22) the Component Inspector tap is updated and the values there will be updated as Figure 23 shows.

Figure 22. Subcomponents expand

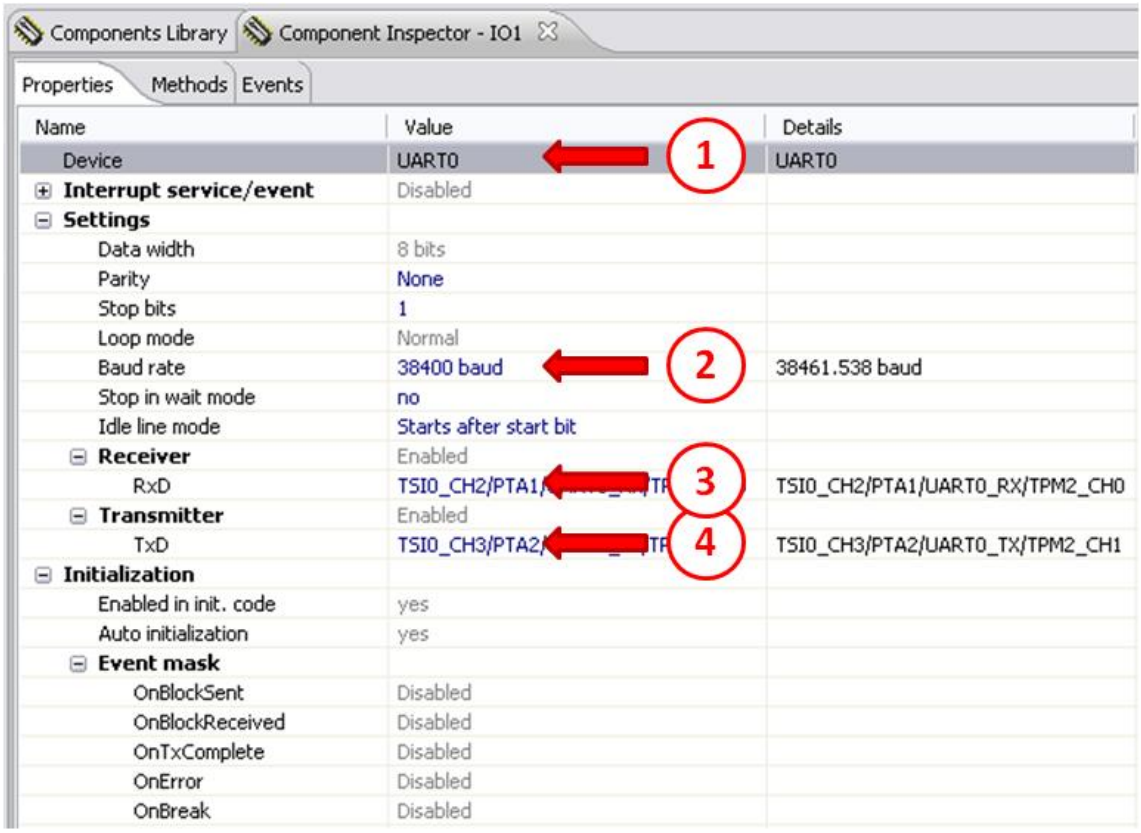


Figure 23. ConsoleIO component settings.

3. After the modifications it is ready to generate updated code using Processor Expert. To do this click in the generate code button located in the up right corner of the “Components” window.
4. It is necessary to compile the project to verify that there are no errors while using the generated code. To compile the project go to the “CodeWarrior Projects” window. Click in the Project name and then click in the compile button. In Figure 24 are the steps.

Figure 24. Compiling the project

5. Validate that the “Problems” tab has no errors listed to ensure a success compilation.
6. It is time now to add the MQXLite component. Click in the “Components Library” tab as figure 25 shows. Then click in the Alphabetical tab and look for the “MQXLite” component. Double clicking in the “MQXLite” component makes it to be included in the project. This is illustrated in Figure 26. NOTE: Skip this step if you used the wizard to create an MQX-Lite project.

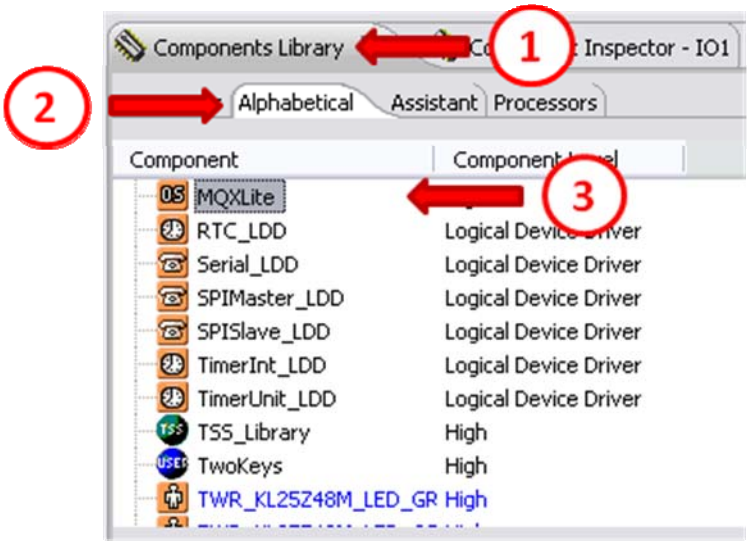


Figure 25. Components Library

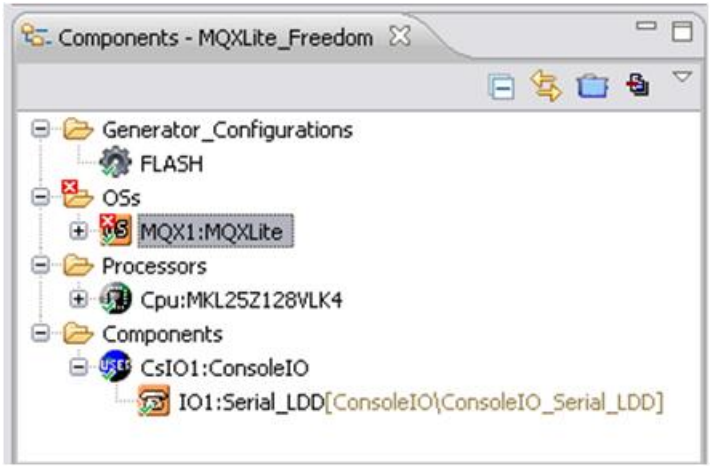


Figure 26. New component added to Processor Expert project: MQXLite

7. The MQXLite default configuration includes 2 basic subcomponents; a system tick and at least one task. By expanding the “MQX1:MQXLite” it is possible to see that the two first subcomponents are the ones mentioned before. Figure 27 testifies these facts.

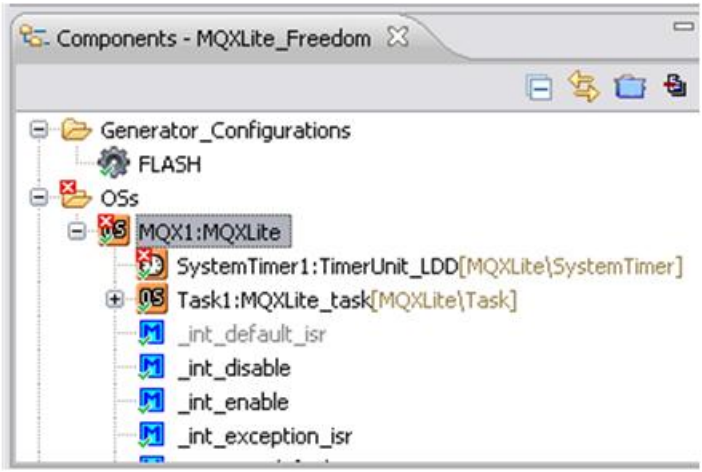


Figure 27. MQXLite subcomponents

8. The first MQXLIte subcomponent is the “SystemTimer1:TimerUnit_LDD”. This is the heart of the system. The system tick is being generated with the “SystemTimer1:TimerUnit_LDD”. Figure 27 shows that the “SystemTimer1:TimerUnit_LDD” subcomponent is marked with errors. This is because it is necessary to provide configure information for the clocks. This way the timer can configure the module correctly. Figure 28 shows the values to be updated in the “Component Inspector” tab when a click is given in the “SystemTimer1:TimerUnit_LDD” subcomponent.

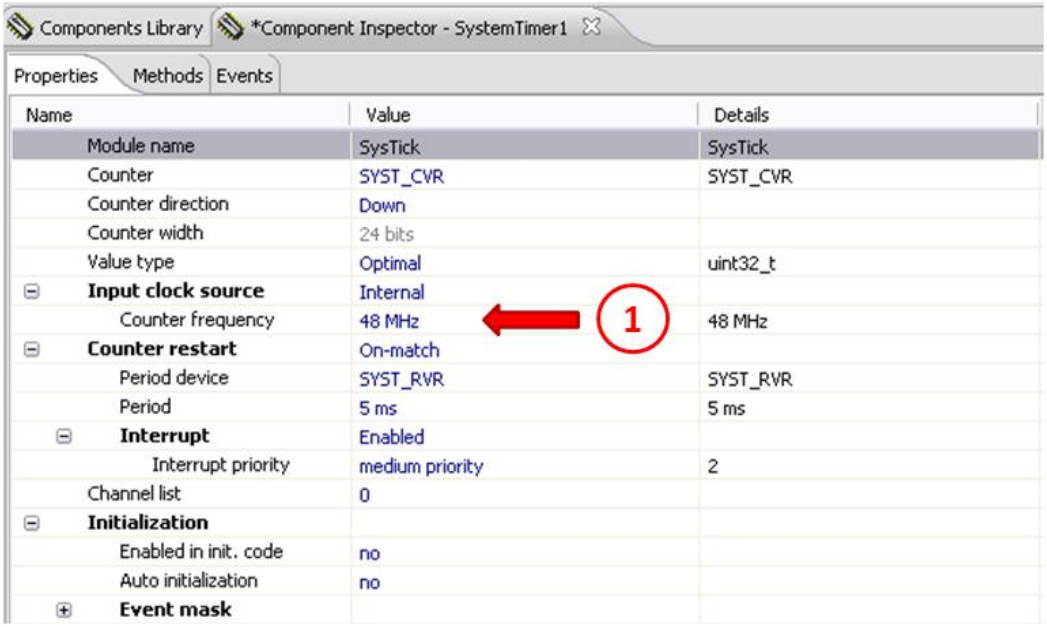


Figure 28. Configuring System Tick for MQXLite

9. At this point it is possible to re-generate code and re-compile to ensure that everything is configured correctly. Repeat the steps 3, 4 and 5 to perform these operations.
10. By default, when the MQXLite component is added a task is also included in this component. Take a look to figure 29 to identify where the default task is located.

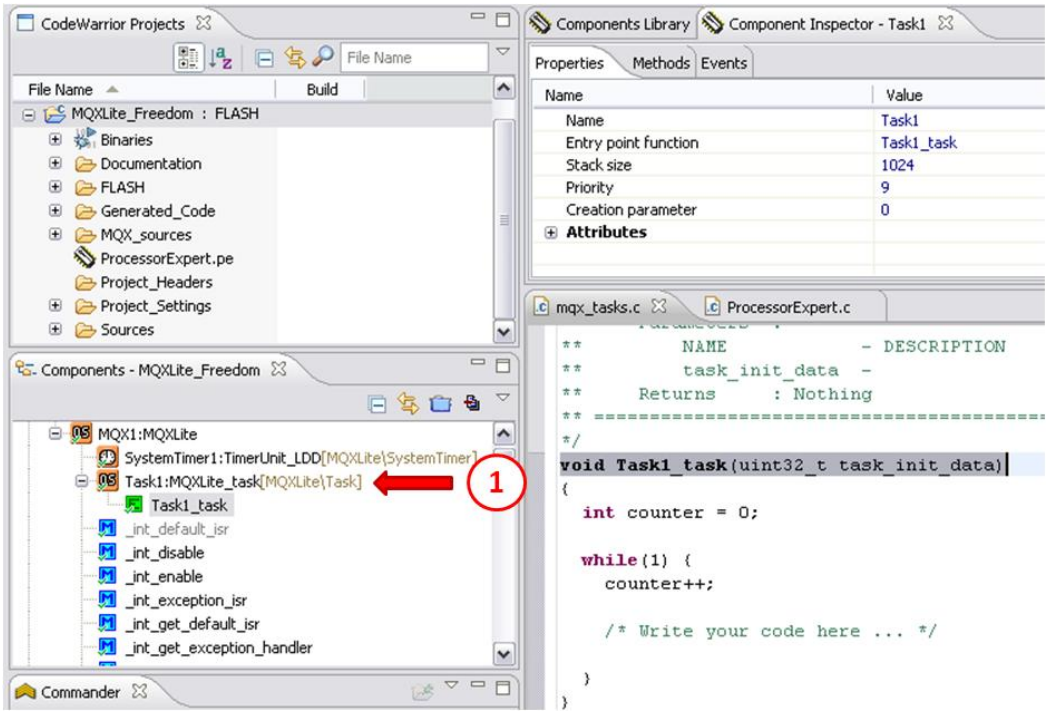


Figure 29. Default first MQXLite task

In order to access the source code of this default task it is necessary to double click in the name of the task component. In this case it is “Task1_task”. The code will be opened at the right side of the CodeWarrior screen showing the definition of the task source code.

6 Controlling the RGB LED

The Freedom – KL25Z board includes a RGB LED that is connected to 3 PWMs from the MCU. Figure 30 shows these connections.

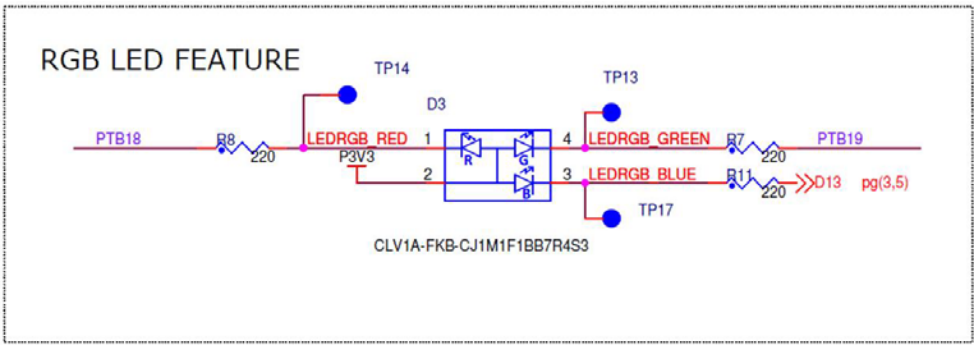


Figure 30. RGB LED schematics connections in the Freedom – KL25Z

The schematic above shows that the RGB LED is connected to PTB18, PTB19 and D13. This information is used to configure the 3 PWMs. The board also includes an I2C Inertial sensor. The data from this sensor is used to control the RGB LED.

1. The first step is to create 2 timer components. The first timer is controlling the R (red) G (green) and the second timer controls the B (blue)
2. Look for the “TimerUnit_LDD” in the component library and add it to the Processor Expert window with a simple double click.
3. It is necessary to change the name to standardize the source code. To modify the name of the component give a right click on it and then give the “PWMTimerRG” name. This name means that this PWM controls the Red and Green LED inside the RGB LED. Figure 31 shows the details.

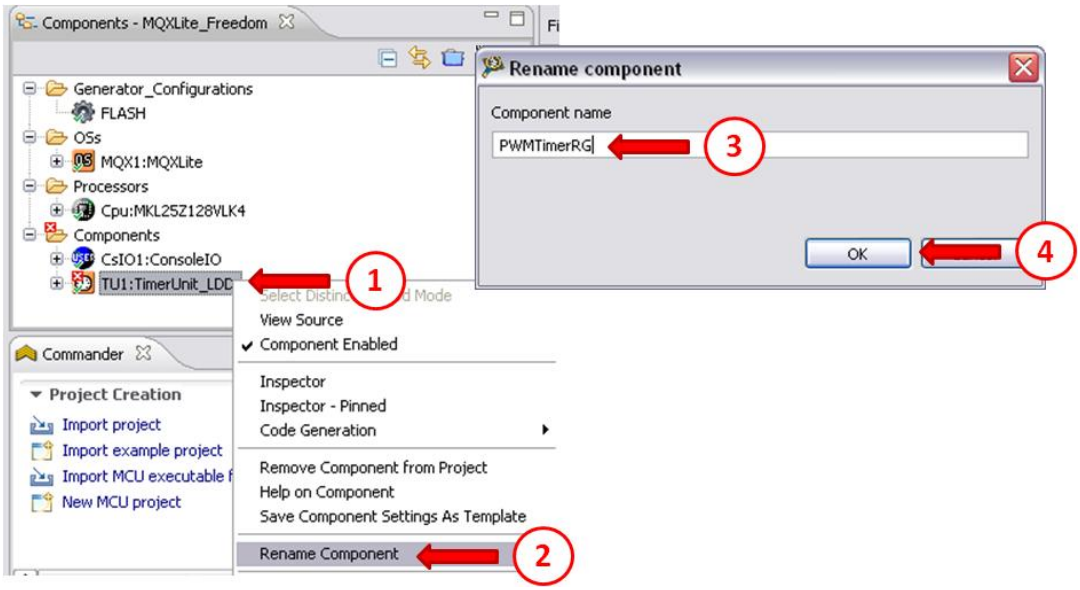
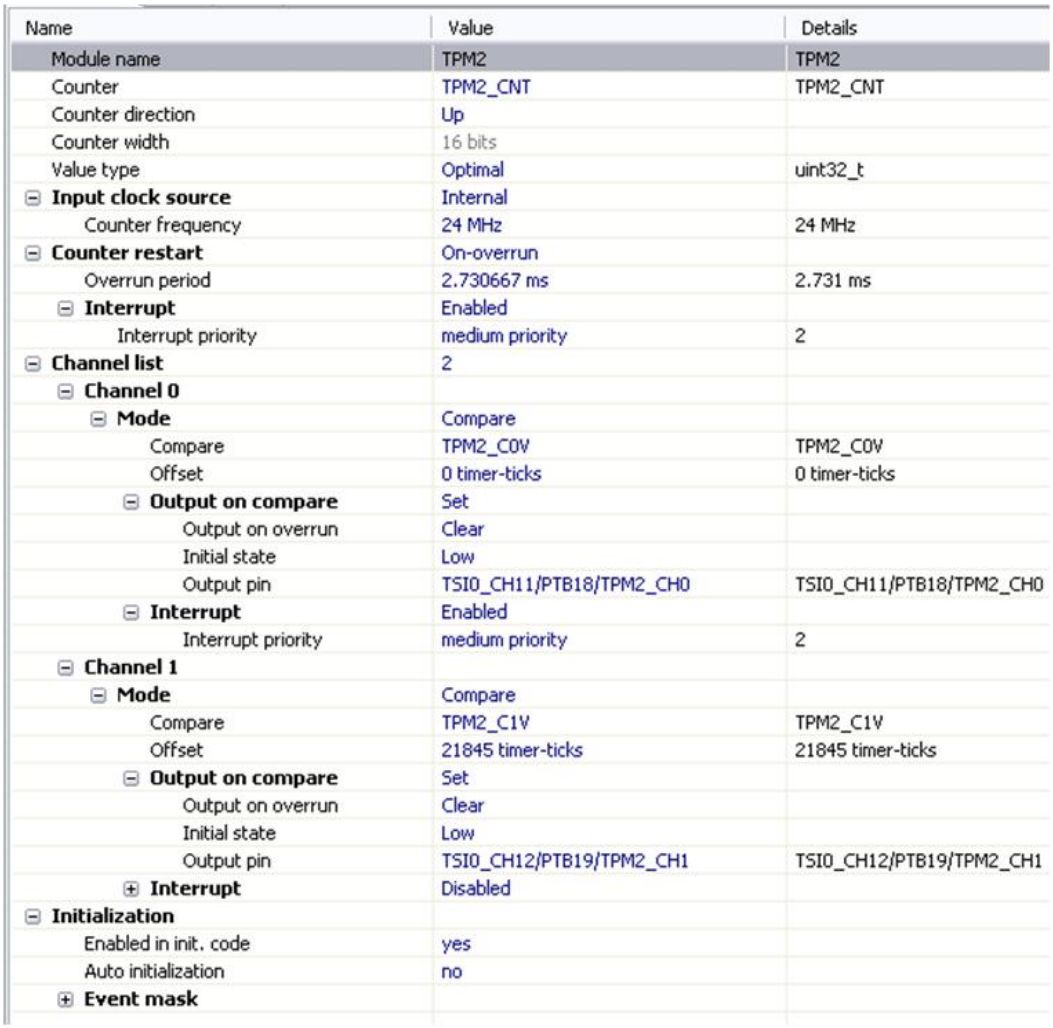


Figure 31. Modify component name.

4. By clicking in the new component the Component Inspector appears allowing the configuration for this. The configuration shall be the same as Figure 32.



Name	Value	Details
Module name	TPM2	TPM2
Counter	TPM2_CNT	TPM2_CNT
Counter direction	Up	
Counter width	16 bits	
Value type	Optimal	uint32_t
Input clock source	Internal	
Counter frequency	24 MHz	24 MHz
Counter restart	On-overrun	
Overrun period	2.730667 ms	2.731 ms
Interrupt	Enabled	
Interrupt priority	medium priority	2
Channel list	2	
Channel 0		
Mode	Compare	
Compare	TPM2_C0V	TPM2_C0V
Offset	0 timer-ticks	0 timer-ticks
Output on compare	Set	
Output on overrun	Clear	
Initial state	Low	
Output pin	TSI0_CH11/PTB18/TPM2_CH0	TSI0_CH11/PTB18/TPM2_CH0
Interrupt	Enabled	
Interrupt priority	medium priority	2
Channel 1		
Mode	Compare	
Compare	TPM2_C1V	TPM2_C1V
Offset	21845 timer-ticks	21845 timer-ticks
Output on compare	Set	
Output on overrun	Clear	
Initial state	Low	
Output pin	TSI0_CH12/PTB19/TPM2_CH1	TSI0_CH12/PTB19/TPM2_CH1
Interrupt	Disabled	
Initialization		
Enabled in init. code	yes	
Auto initialization	no	
Event mask		

Figure 32. Configuring the “PWMTimerRG” component

5. Look for the “TimerUnit_LDD” in the component library and add it to the Processor Expert window with a simple double click.
6. It is necessary to change the name to standardize the source code. To modify the name of the component give a right click on it and then type the “PWMTimerB” name. This name means that this PWM controls the Blue LED inside the RGB LED. Figure 33 shows the details.

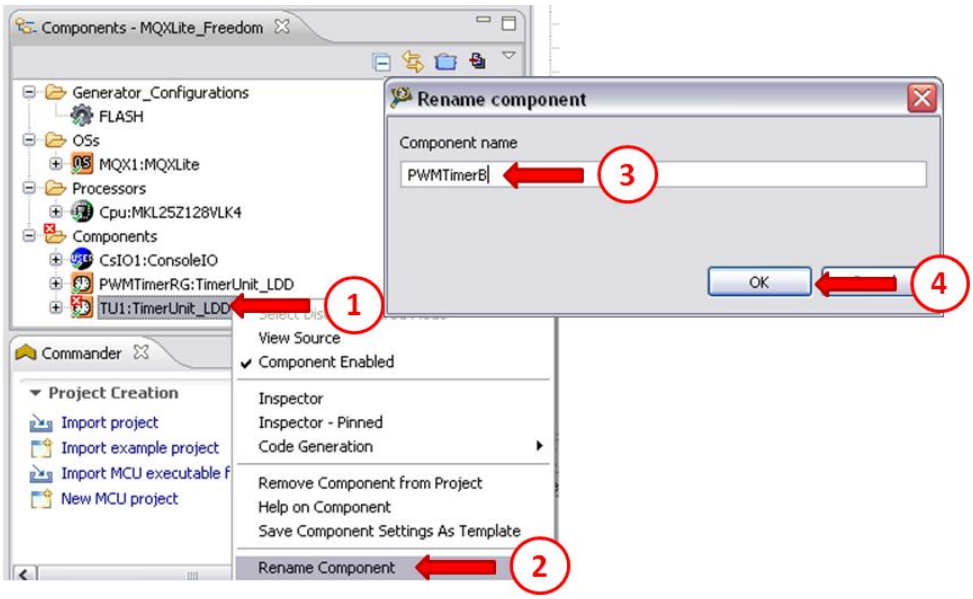


Figure 33. Modify component name.

7. By clicking in the new component the Component Inspector window appears to do the configuration. The configuration shall be the same as Figure 34.

Name	Value	Details
Module name	TPM0	TPM0
Counter	TPM0_CNT	TPM0_CNT
Counter direction	Up	
Counter width	16 bits	
Value type	Optimal	uint32_t
Input clock source	Internal	
Counter frequency	24 MHz	24 MHz
Counter restart	On-overrun	
Overrun period	2.730667 ms	2.731 ms
Interrupt	Disabled	
Channel list	1	
Channel 0		
Mode	Compare	
Compare	TPM0_C1V	TPM0_C1V
Offset	43690 timer-ticks	43690 timer-ticks
Output on compare	Set	
Output on overrun	Clear	
Initial state	Low	
Output pin	ADC0_SE5b/PTD1/SPI0_SCK/TPM0_CH1	ADC0_SE5b/PTD1/SPI0_SCK/TPM0_C...
Interrupt	Disabled	
Initialization		
Enabled in init. code	yes	
Auto initialization	no	
Event mask		

Figure 34. Configuring the “PWMTimerB” component

8. For the “TimerUnit_LDD” components it is necessary to define the functions to be added by the auto-generated code. To do this, click in the “Methods” tab under the Component Inspector. These settings shall apply for the “PWMTimerRG” and the “PWMTimerB”. Figure 35 illustrates the right configuration for the methods creation.

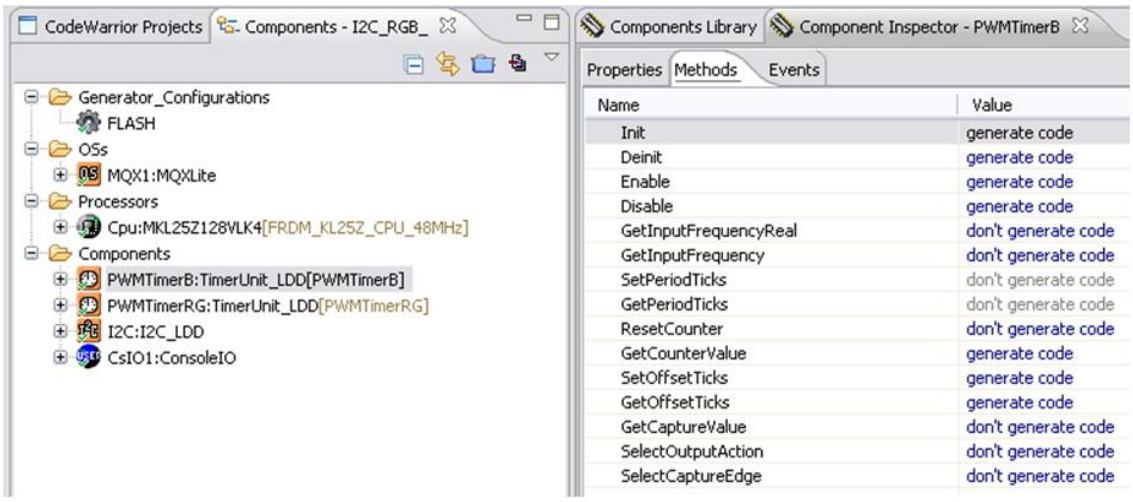


Figure 35. “PWMTimerRG” and the “PWMTimerB” methods creation definition

9. Look for the “I2C_LDD” in the component library and add it to the Processor Expert window with a simple double click.
10. It is necessary to change the name to standardize the source code. To modify the name of the component give a right click on it and give the “I2C” name.
11. By clicking in the new component the Component Inspector appears allowing the configuration for this. The configuration shall be the same as Figure 36.

Name	Value	Details
I2C channel	I2C0	I2C0
Interrupt service	Enabled	
Interrupt priority	medium priority	2
Settings		
Mode selection	MASTER	
MASTER mode	Enabled	
Initialization		
Address mode	7-bit addressing	
Target slave address init	1D H	
SLAVE mode	Disabled	
Pins		
SDA pin		
SDA pin	PTE25/TPM0_CH1/I2C0_SDA	PTE25/TPM0_CH1/I2C0_SDA
SCL pin		
SCL pin	PTE24/TPM0_CH0/I2C0_SCL	PTE24/TPM0_CH0/I2C0_SCL
Internal frequency (multiplier factor)	24 MHz	24 MHz
Bits 0-2 of Frequency divider register	101	
Bits 3-5 of Frequency divider register	100	
SCL frequency	75 kHz	Clock conf. 0: 75 kHz
SDA Hold	2.042 us	Clock conf. 0: 2.042 us
SCL start Hold	6.583 us	Clock conf. 0: 6.583 us
SCL stop Hold	6.708 us	Clock conf. 0: 6.708 us
Initialization		
Enabled in init code	yes	
Auto initialization	no	

Figure 36. Configuring the “I2C” component

12. For the “I2C_LDD” components it is necessary to define the functions to be added by the auto-generated code. To do this click in the “Methods” tab under the Component Inspector. Figure 37 illustrates the right configuration for the methods creation.

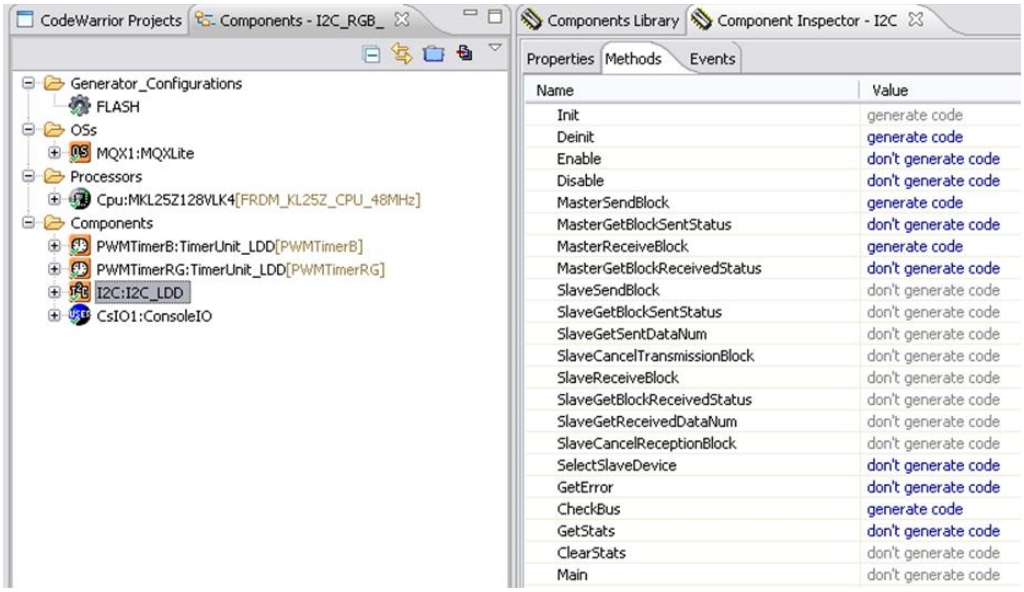


Figure 37. The “I2C” methods creation definition

13. All the needed components were added at this point. To validate that everything is correct it re-generate and compile the code. Check that the “Problems” tab for no errors listed to ensure a successful compilation.

6.1 Adding the source code

The source code to develop this application is divided in 2 sections; the code for the I2C and the code for the PWMs. The first code enables the communication with the inertial sensor through I2C. Then the PWM code is added. The full source code can be found in the appendix. The following steps make emphasis in the key sections of the source code.

1. All this should happen in the MQXLite task. Expand the MQXLite component until you reach the “Task1_task” subcomponent. Double click on it to make the source code appear at the right hand of the screen. Figure 38 is an example of how it looks.

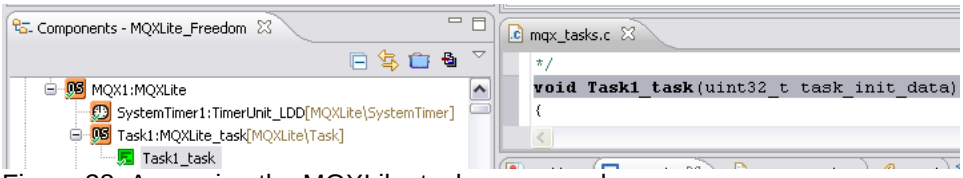


Figure 38. Accessing the MQXLike task source code

2. The Task1_task holds the source code for the only MQXLite task that we have configured. The following code initializes the I2C and is added at the beginning of the Task1_task() function.

```
I2C_DeviceData = I2C_Init(&DataState);
Error = !ReadAccRegs(I2C_DeviceData, &DataState, CTRL_REG_1, ACC_REG_SIZE, &Data);
if (!Error) {
  Data = (ACTIVE_BIT_MASK | F_READ_BIT_MASK); /* Set active mode bit and fast read mode bit *
  Error = !WriteAccRegs(I2C_DeviceData, &DataState, CTRL_REG_1, ACC_REG_SIZE, &Data);
}
if (!Error) {
  Data = 0;
  Error = !ReadAccRegs(I2C_DeviceData, &DataState, CTRL_REG_1, ACC_REG_SIZE, &Data);
  if (!Error) {
    if (Data != (ACTIVE_BIT_MASK | F_READ_BIT_MASK)) {
      Error = TRUE;
    }
  }
}
```

Figure 39. I2C initialization source code

The function I2C_Init() starts the low level driver. The function ReadAccRegs() is ensuring that we have communication with the device (in this case, the inertial sensor). We need to activate the device and set the fast read mode and validate that these values were written correctly in the device register.

The I2C driver is configured as an interrupt. It is necessary to open the Event.c file to add the source code of the I2C interrupt service routine. Figure 40 shows the source code of the interrupt routines that need to be modified. The function declarations are already in the Event.c file. It is necessary to add only the body of the function.

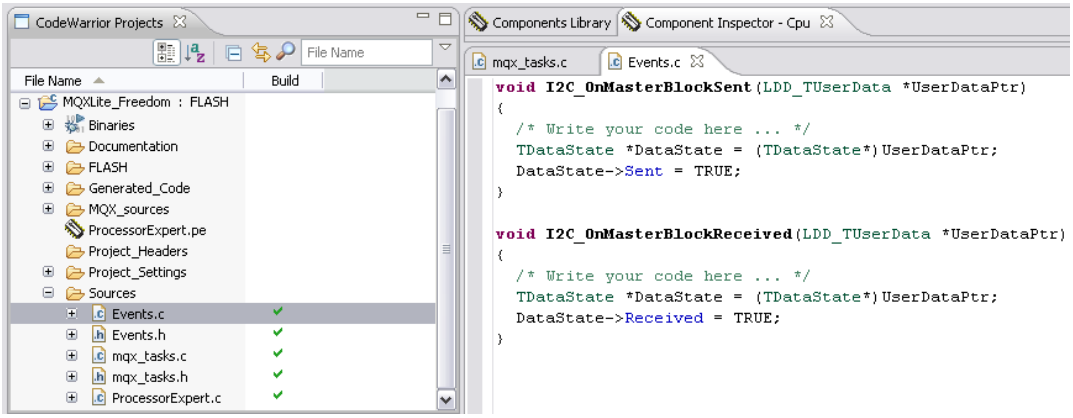


Figure 40. ISR Source code

The function I2C_OnMasterBlockSent() is called when I2C is in master mode and finishes the transmission of the data successfully. The function I2C_OnMasterBlockReceived() is called when I2C is in master mode and finishes the reception of the data successfully. Then both functions update the state and data to be sent to the application.

3. Now the PWMs are initialized.

```
PWMTimerRG_DeviceData = PWMTimerRG_Init(NULL);
PWMTimerB_DeviceData = PWMTimerB_Init(NULL);
```

Figure 41. PWM initialization source code

4. Then the code enters into an infinite loop where it reads for any update from the I2C device. This data is then used to modify the ticks in the PWM and give a different color in the RGB LED. Figure 42 is the view of this part of the code.

```
while(1)
{
  Error = !ReadAccRegs(I2C_DeviceData, &DataState, OUT_X_MSB, 3 * ACC_REG_SIZE, (uint8_t*) Color);
  if (!Error) {

      PWMTimerRG_Enable(PWMTimerRG_DeviceData);
      PWMTimerB_Enable(PWMTimerB_DeviceData);
      PWMTimerRG_SetOffsetTicks(PWMTimerRG_DeviceData, 0,1000*(1<<(abs(Color[0]/10))));
      PWMTimerRG_SetOffsetTicks(PWMTimerRG_DeviceData, 1, 1000*(1<<(abs(Color[1]/10))));
      PWMTimerB_SetOffsetTicks(PWMTimerB_DeviceData, 0, 1000*(1<<(abs(Color[2]/10))));

  }
_time_delay_ticks(1);
}
```

Figure 42. Main infinite loop

5. Once all the code changes were done it is necessary to recompile the project and validate that there are no errors. A strong suggestion is to replace the content of the files mqx_tasks.c, mqx_tasks.h and Events.c from the appendixes into the CW project. This ensures that no code is missing.

6.2 Debugging the source code

The source code is ready for debugging. The following steps explain how to perform this operation.

1. Go to the debug button (bug icon) in the tools bar and expand the debugging menu just like Figure 43 demonstrates it. Then click in “Debug Configurations…” option to enter into the available debug settings for the project.

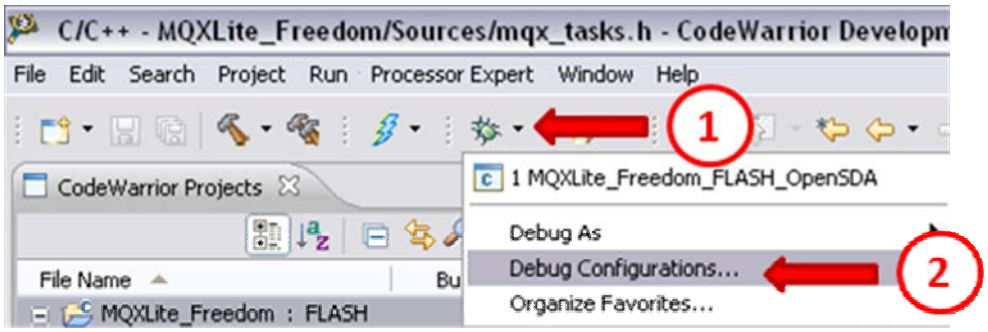


Figure 43. Entering into Debug Configurations.

2. Select the options “MQXLite_Freedom_Flash_OpenSDA” and then click “Debug”.

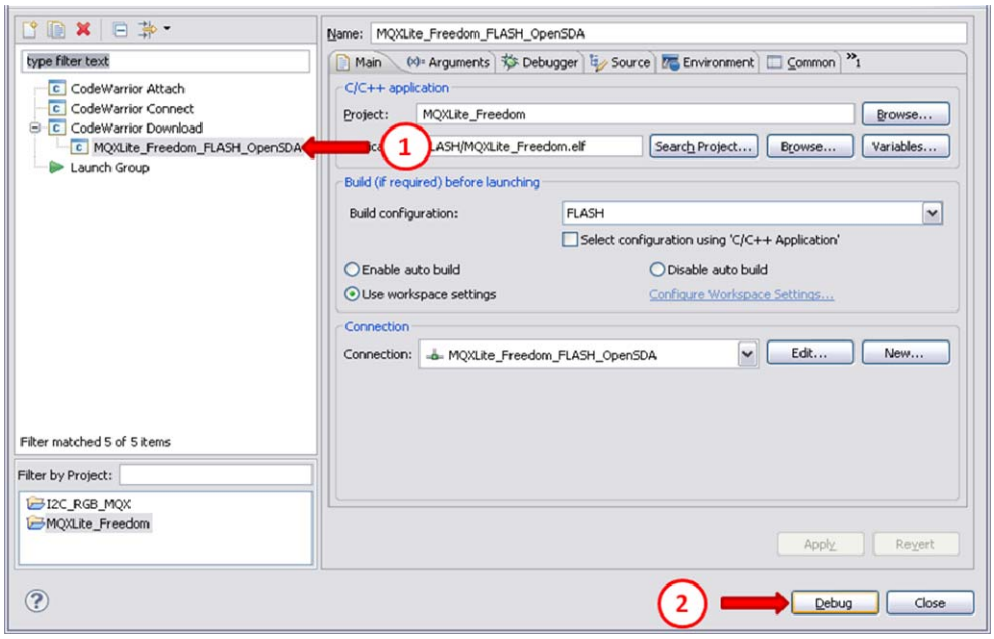


Figure 44. Starting debugging session

3. At this point the CodeWarrior perspective view changes and after the source code is downloaded to the board we are able to start the debugging. Figure 45 illustrates a debugging session.

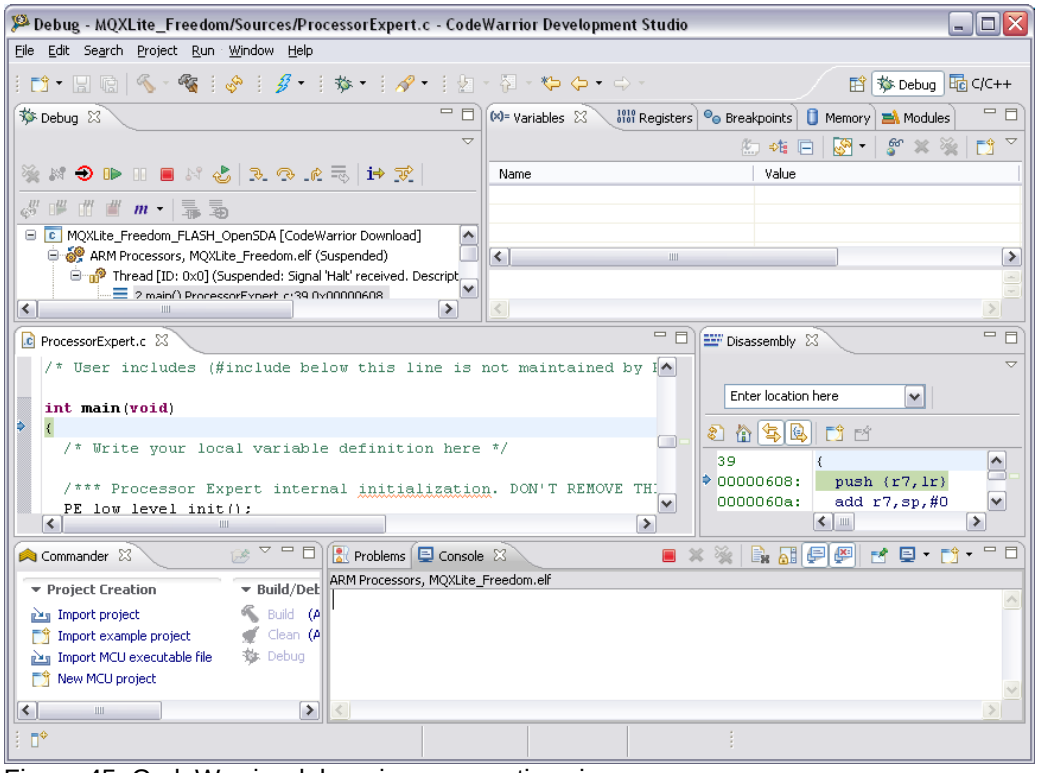


Figure 45. CodeWarrior debugging perspective view

4. At this point it is possible to let the code free run by pressing the “F8” key or debug it step-by-step by pressing the “F6” key.

7 Conclusion

This document describes the steps needed to start a simple MQXLite project and make usage of the I2C and PWM module from the KL25Z128VLK4 micro controller (found in the FREEDOM – KL25Z board).

The addition of new components and source code using Processor Expert and MQXLite is simple and fast. This helps to spend less time in the environment tools which gives more time to develop the application architecture.

The introduction of MQXLite + Processor Expert + KL25Z board is a key combination for prototyping new products.

Appendix A

```
/** ###################################################################
**     Filename    : mqx_tasks.c
**     Project     : ProcessorExpert
**     Processor   : MKL25Z128VLK4
**     Component   : Events
**     Version     : Driver 01.00
**     Compiler    : GNU C Compiler
**     Date/Time   : 2012-09-12, 23:41, # CodeGen: 3
**     Abstract    :
**         This is user's event module.
**         Put your event handler code here.
**     Settings    :
**     Contents    :
**         Task1_task - void Task1_task(uint32_t task_init_data);
**
** ###################################################################*/
/* MODULE mqx_tasks */

#include "Cpu.h"
#include "Events.h"
#include "mqx_tasks.h"

/* User includes (#include below this line is not maintained by Processor Expert) */
/* MMA8451Q IOMap */
/* External 3-axis accelerometer data register addresses */
#define OUT_X_MSB 0x01
#define OUT_X_LSB 0x02
#define OUT_Y_MSB 0x03
#define OUT_Y_LSB 0x04
#define OUT_Z_MSB 0x05
#define OUT_Z_LSB 0x06
/* External 3-axis accelerometer control register addresses */
#define CTRL_REG_1 0x2A
/* External 3-axis accelerometer control register bit masks */
#define ACTIVE_BIT_MASK 0x01
#define F_READ_BIT_MASK 0x02

#define ACC_REG_SIZE 1U
#define READ_COUNT 5U

LDD_TDeviceData *I2C_DeviceData = NULL;
TDataState DataState;

LDD_TDeviceData *PWMTimerRG_DeviceData = NULL;
LDD_TDeviceData *PWMTimerB_DeviceData = NULL;

bool ReadAccRegs(LDD_TDeviceData *I2CPtr,
    TDataState *DataState, uint8_t Address, uint8_t RegCount, uint8_t *Buffer)
{
  LDD_I2C_TBusState BusState;
  DataState->Sent = FALSE;
  I2C_MasterSendBlock(I2CPtr, &Address, sizeof(Address), LDD_I2C_NO_SEND_STOP);
  while (!DataState->Sent) {}
  if (!DataState->Sent) {
    return FALSE;
  }
  DataState->Received = FALSE;
  I2C_MasterReceiveBlock(I2CPtr, Buffer, RegCount, LDD_I2C_SEND_STOP);
  while (!DataState->Received) {}
  do {I2C_CheckBus(I2CPtr, &BusState);}
  while (BusState != LDD_I2C_IDLE);
  if (!DataState->Received) {
    return FALSE;
  }
  return TRUE;
}

bool WriteAccRegs(LDD_TDeviceData *I2CPtr,
    TDataState *DataState, uint8_t Address, uint8_t RegCount, uint8_t *Data)
{
  LDD_I2C_TBusState BusState;
  const uint8_t MAX_REG_COUNT = 16;
  uint8_t SendBuffer[MAX_REG_COUNT];

  SendBuffer[0] = Address;
  memcpy(&SendBuffer[1], Data, RegCount);
  DataState->Sent = FALSE;
  I2C_MasterSendBlock(I2CPtr, &SendBuffer, RegCount + 1, LDD_I2C_SEND_STOP);
  while (!DataState->Sent) {}
```

```
  do {I2C_CheckBus(I2CPtr, &BusState);}
  while(BusState != LDD_I2C_IDLE);
  if (!DataState->Sent) {
    return FALSE;
  }
  return TRUE;
}

/*
** ===================================================================
**     Event       :  Task1_task (module mqx_tasks)
**
**     Component   :  Task1 [MQXLite_task]
**     Description :
**         MQX task routine. The routine is generated into mqx_tasks.c
**         file.
**     Parameters  :
**         NAME            - DESCRIPTION
**         task_init_data  -
**     Returns     : Nothing
** ===================================================================
*/
void Task1_task(uint32_t task_init_data)
{
  byte Data;
  LDD_TError Error = 0;
  signed char Color[3] = {0,127,127};  // initialize to turqoise

  printf("Project description:\n");
  printf("I2C example of communication with external accelerometer.\n");
  printf("PWM is used for dimming the RGB LED in dependence on tilt of the board.\n");
  printf("\n");

  // Initialize Accelerometer

  I2C_DeviceData = I2C_Init(&DataState);
  Error = !ReadAccRegs(I2C_DeviceData, &DataState, CTRL_REG_1, ACC_REG_SIZE, &Data);
  if (!Error) {
    Data = (ACTIVE_BIT_MASK | F_READ_BIT_MASK); /* Set active mode bit and fast read mode bit */
    Error = !WriteAccRegs(I2C_DeviceData, &DataState, CTRL_REG_1, ACC_REG_SIZE, &Data);
  }
  if (!Error) {
    Data = 0;
    Error = !ReadAccRegs(I2C_DeviceData, &DataState, CTRL_REG_1, ACC_REG_SIZE, &Data);
    if (!Error) {
      if (Data != (ACTIVE_BIT_MASK | F_READ_BIT_MASK)) {
        Error = TRUE;
      }
    }
  }
    /* Initialization passed? */
  if (!Error) {
    printf("PASSED.\n");
  } else {
    printf("FAILED.\n");
  }

  if (!Error) {
        printf("Tilt your Freedom Board to change the RGB LED colors.\n");
  }

  PWMTimerRG_DeviceData = PWMTimerRG_Init(NULL);
  PWMTimerB_DeviceData = PWMTimerB_Init(NULL);

  while(1)
  {
    Error = !ReadAccRegs(I2C_DeviceData, &DataState, OUT_X_MSB, 3 * ACC_REG_SIZE, (uint8_t*) Color);
// Read x,y,z acceleration data.
    if (!Error) {

        PWMTimerRG_Enable(PWMTimerRG_DeviceData);
        PWMTimerB_Enable(PWMTimerB_DeviceData);
        PWMTimerRG_SetOffsetTicks(PWMTimerRG_DeviceData, 0,1000*(1<<(abs(Color[0]/10)))); // x axis -
red LED
        PWMTimerRG_SetOffsetTicks(PWMTimerRG_DeviceData, 1, 1000*(1<<(abs(Color[1]/10)))); // y axis -
green LED
        PWMTimerB_SetOffsetTicks(PWMTimerB_DeviceData, 0, 1000*(1<<(abs(Color[2]/10)))); // z axis -
blue LED

    }
  _time_delay_ticks(1);
```

```
  }
}

/* END mqx_tasks */

/*
** ###################################################################
**
**     This file was created by Processor Expert 10.0 [05.02]
**     for the Freescale Kinetis series of microcontrollers.
**
** ###################################################################
*/
```

Appendix B

```
/** ###################################################################
**     Filename    : Events.c
**     Project     : ProcessorExpert
**     Processor   : MKL25Z128VLK4
**     Component   : Events
**     Version     : Driver 01.00
**     Compiler    : GNU C Compiler
**     Date/Time   : 2012-09-11, 22:14, # CodeGen: 0
**     Abstract    :
**         This is user's event module.
**         Put your event handler code here.
**     Settings    :
**     Contents    :
**         Cpu_OnNMIINT - void Cpu_OnNMIINT(void);
**
** ###################################################################*/
/* MODULE Events */

#include "Cpu.h"
#include "Events.h"
#include "mqx_tasks.h"

/* User includes (#include below this line is not maintained by Processor Expert) */

/*
** ===================================================================
**     Event       :  Cpu_OnNMIINT (module Events)
**
**     Component   :  Cpu [MKL25Z128LK4]
**     Description :
**         This event is called when the Non maskable interrupt had
**         occurred. This event is automatically enabled when the <NMI
**         interrrupt> property is set to 'Enabled'.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void Cpu_OnNMIINT(void)
{
  /* Write your code here ... */
}

/*
** ===================================================================
**     Event       :  IO1_OnBlockReceived (module Events)
**
**     Component   :  IO1 [Serial_LDD]
**     Description :
**         This event is called when the requested number of data is
**         moved to the input buffer.
**     Parameters  :
**         NAME            - DESCRIPTION
**       * UserDataPtr     - Pointer to the user or
**                           RTOS specific data. This pointer is passed
**                           as the parameter of Init method.
**     Returns     : Nothing
** ===================================================================
*/
void IO1_OnBlockReceived(LDD_TUserData *UserDataPtr)
{
  /* Write your code here ... */
}

/*
```

```
** ===================================================================
**     Event       :  IO1_OnBlockSent (module Events)
**
**     Component   :  IO1 [Serial_LDD]
**     Description :
**         This event is called after the last character from the
**         output buffer is moved to the transmitter.
**     Parameters  :
**         NAME            - DESCRIPTION
**       * UserDataPtr     - Pointer to the user or
**                           RTOS specific data. This pointer is passed
**                           as the parameter of Init method.
**     Returns     : Nothing
** ===================================================================
*/
void IO1_OnBlockSent(LDD_TUserData *UserDataPtr)
{
  /* Write your code here ... */
}

/*
** ===================================================================
**     Event       :  PWMTimerRG_OnCounterRestart (module Events)
**
**     Component   :  PWMTimerRG [TimerUnit_LDD]
**     Description :
**         Called if counter overflow/underflow or counter is
**         reinitialized by modulo or compare register matching.
**         OnCounterRestart event and Timer unit must be enabled. See
**         <SetEventMask> and <GetEventMask> methods. This event is
**         available only if a <Interrupt> is enabled.
**     Parameters  :
**         NAME            - DESCRIPTION
**       * UserDataPtr     - Pointer to the user or
**                           RTOS specific data. The pointer passed as
**                           the parameter of Init method.
**     Returns     : Nothing
** ===================================================================
*/
void PWMTimerRG_OnCounterRestart(LDD_TUserData *UserDataPtr)
{
  /* Write your code here ... */
}

/*
** ===================================================================
**     Event       :  PWMTimerRG_OnChannel0 (module Events)
**
**     Component   :  PWMTimerRG [TimerUnit_LDD]
**     Description :
**         Called if compare register match the counter registers or
**         capture register has a new content. OnChannel0 event and
**         Timer unit must be enabled. See <SetEventMask> and
**         <GetEventMask> methods. This event is available only if a
**         <Interrupt> is enabled.
**     Parameters  :
**         NAME            - DESCRIPTION
**       * UserDataPtr     - Pointer to the user or
**                           RTOS specific data. The pointer passed as
**                           the parameter of Init method.
**     Returns     : Nothing
** ===================================================================
*/
void PWMTimerRG_OnChannel0(LDD_TUserData *UserDataPtr)
{
  /* Write your code here ... */
}

/*
** ===================================================================
**     Event       :  I2C_OnMasterBlockSent (module Events)
**
**     Component   :  I2C [I2C_LDD]
**     Description :
**         This event is called when I2C in master mode finishes the
**         transmission of the data successfully. This event is not
**         available for the SLAVE mode and if MasterSendBlock is
**         disabled.
**     Parameters  :
**         NAME            - DESCRIPTION
**       * UserDataPtr     - Pointer to the user or
```

```
**                         RTOS specific data. This pointer is passed
**                         as the parameter of Init method.
**     Returns     : Nothing
** ===================================================================
*/
void I2C_OnMasterBlockSent(LDD_TUserData *UserDataPtr)
{
  /* Write your code here ... */
  TDataState *DataState = (TDataState*)UserDataPtr;
  DataState->Sent = TRUE;
}

void I2C_OnMasterBlockReceived(LDD_TUserData *UserDataPtr)
{
  /* Write your code here ... */
  TDataState *DataState = (TDataState*)UserDataPtr;
  DataState->Received = TRUE;
}

/*
** ===================================================================
**     Event       :  I2C_OnError (module Events)
**
**     Component   :  I2C [I2C_LDD]
**     Description :
**         This event is called when an error (e.g. Arbitration lost)
**         occurs. The errors can be read with GetError method.
**     Parameters  :
**         NAME            - DESCRIPTION
**       * UserDataPtr     - Pointer to the user or
**                           RTOS specific data. This pointer is passed
**                           as the parameter of Init method.
**     Returns     : Nothing
** ===================================================================
*/
void I2C_OnError(LDD_TUserData *UserDataPtr)
{
  /* Write your code here ... */
}

/* END Events */

/*
** ###################################################################
**
**     This file was created by Processor Expert 10.0 [05.02]
**     for the Freescale Kinetis series of microcontrollers.
**
** ###################################################################
*/
```

Appendix C

```
/** ###################################################################
**     Filename    : mqx_tasks.h
**     Project     : ProcessorExpert
**     Processor   : MKL25Z128VLK4
**     Component   : Events
**     Version     : Driver 01.00
**     Compiler    : GNU C Compiler
**     Date/Time   : 2012-09-12, 23:41, # CodeGen: 3
**     Abstract    :
**         This is user's event module.
**         Put your event handler code here.
**     Settings    :
**     Contents    :
**         Task1_task - void Task1_task(uint32_t task_init_data);
**
** ###################################################################*/

#ifndef __mqx_tasks_H
#define __mqx_tasks_H
/* MODULE mqx_tasks */

#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
#include "CsIO1.h"
#include "IO1.h"
#include "MQX1.h"
```

```c
#include "SystemTimer1.h"
#include "PWMTimerRG.h"
#include "PWMTimerB.h"
#include "I2C.h"
#include "PE_LDD.h"

typedef struct {
  volatile bool Sent;
  volatile bool Received;
} TDataState;

void Task1_task(uint32_t task_init_data);
/*
** ===================================================================
**     Event       :  Task1_task (module mqx_tasks)
**
**     Component   :  Task1 [MQXLite_task]
**     Description :
**         MQX task routine. The routine is generated into mqx_tasks.c
**         file.
**     Parameters  :
**         NAME            - DESCRIPTION
**         task_init_data  -
**     Returns     : Nothing
** ===================================================================
*/


/* END mqx_tasks */
#endif /* __mqx_tasks_H*/

/*
** ###################################################################
**
**     This file was created by Processor Expert 10.0 [05.02]
**     for the Freescale Kinetis series of microcontrollers.
**
** ###################################################################
*/
```