Freescale Semiconductor
Application Note

Document Number:AN4894
Rev 0, Mar 2014

Getting Started with Qorivva Fast Start Kit for TRK-MPC5604P

by: Sudhakar Srinivasa

Contents

1 Introduction

This application note provides an overview of software tools provided with the Fast Start Kit for the TRK-MPC5604P. The application note also provides an example to use the software tools included in the Fast Start Kit. The example software application uses:

- RAppID init tool for configuring microcontroller and for auto generation of the initialization code
- CodeWarrior Development Studio for Microcontrollers v10.5 for building the project
- RAppID Boot loader for programming the code on to the target
- FreeMASTER utility for monitor and debug purposes

2 Overview of Qorivva Fast Start Kit for TRK-MPC5604P

The Fast Start Kit for TRK-MPC5604P contains TRK-MPC5604P evaluation board and many Freescale's software tools to help you get started with your application development. The software tools consist of:

- RAppID initialization tool,
- CodeWarrior Development Studio v10.5 (Special Edition),
- RAppID Boot loader utility,
- FreeMASTER utility,

- CodeWarrior Project Maker utility to add RAppID generated source code to an empty CodeWarrior project,
- Driver code for the MPC5604P,
- Example projects to help you to get started with the Fast Start Kit.

An overview of some of the software tools provided with Fast Start Kit is described below.

2.1 RAppID Init overview

RAppID Init is a family of graphical development tools for Freescale's MPC56xx Qorivva microcontrollers that enable the user to quickly and easily configure the microcontroller and generate complete initialization code and documentation. It is also a learning tool that can help gain understanding of the microcontroller and its peripherals. Some of the product highlights include:

- Intuitive, easy-to-use graphical user interface (GUI)
- Comprehensive initialization of the CPU, memory, and peripherals
- Automatic DMA register setting from peripherals for basic modes
- Built-in consistency checks to minimize incorrect settings
- Automatic report generation of settings
- Efficient C and assembly code generation for compilers from companies such as Wind River®, Green Hills®, and CodeWarrior
- Online documentation and built-in tool tips
- Installation comes with many example projects
- Generates complete infrastructure code for MCU startup
- Provisions for revision management
- Automatic date and time stamps on generated code and reports
- Modular code generation - generate code for any or all peripherals
- Option to generate code for RAM or Flash
- Flexible Initialization sequence
- Project import/export capability for distributed development teams

2.2 RAppID Boot Loader utility

The RAppID Boot Loader tool is developed by Freescale and helps you develop software for Freescale microcontrollers by providing a method to update software of these microcontrollers through a serial link. The RAppID Boot Loader works with the built in Boot Assist Module (BAM) included in the Freescale's Qorivva family of parts. The Boot Loader provides a streamlined method for programming code into FLASH or RAM on either target EVBs or custom boards. The Boot Loader has two modes of operation; for use as a stand-alone PC desktop GUI utility, or for integration with different user required tools chains through a command line interface (i.e. Eclipse Plug-in, MatLab/SimuLink etc.).

2.3 FreeMASTER utility

FreeMASTER is a user-friendly real-time debug monitor and data visualization tool for application development and information management. FreeMASTER supports completely non-intrusive monitoring of variables on a running system. You can display multiple variables changing over time on an oscilloscope-like display, or view the data in text form.

2.4 Low-level and high-level drivers

The Fast Start Kit for TRK-MPC5604P includes low-level and high-level drivers to help make the application development easier. The included drivers are for peripherals ADC, GPIO, UART, and CAN. The high-level drivers included are for the Potentiometer, Photo Sensor, and System Basis Chip (SBC) that are on the TRK-MPC5604P board.

2.5 Overview of example application

The example provided in this application note demonstrates the software tools provided with the Fast Start Kit and makes use of the input buttons, LEDs, and analog inputs provided in the TRK-MPC5604P evaluation board. This example uses multiple peripherals on the MPC5604P microcontroller like ADC, SIU, PIT, LINFlex (UART), and CAN. The example project turns on/off the LEDs based on different input commands:

- LED1 is turned on/off using PIT interrupt
- LED2 is turned on/off based on Potentiometer input value
- LED3 is turned on/off based on input button S3
- LED4 is turned on/off based on CAN command

The application communicates with FreeMASTER utility via UART and monitors variables used in the example project.

The next few sections in this application note describe the steps to configure microcontroller, generate code, and build and run a simple project on TRK-MPC5604P target. The figure below depicts the example software in a block diagram.

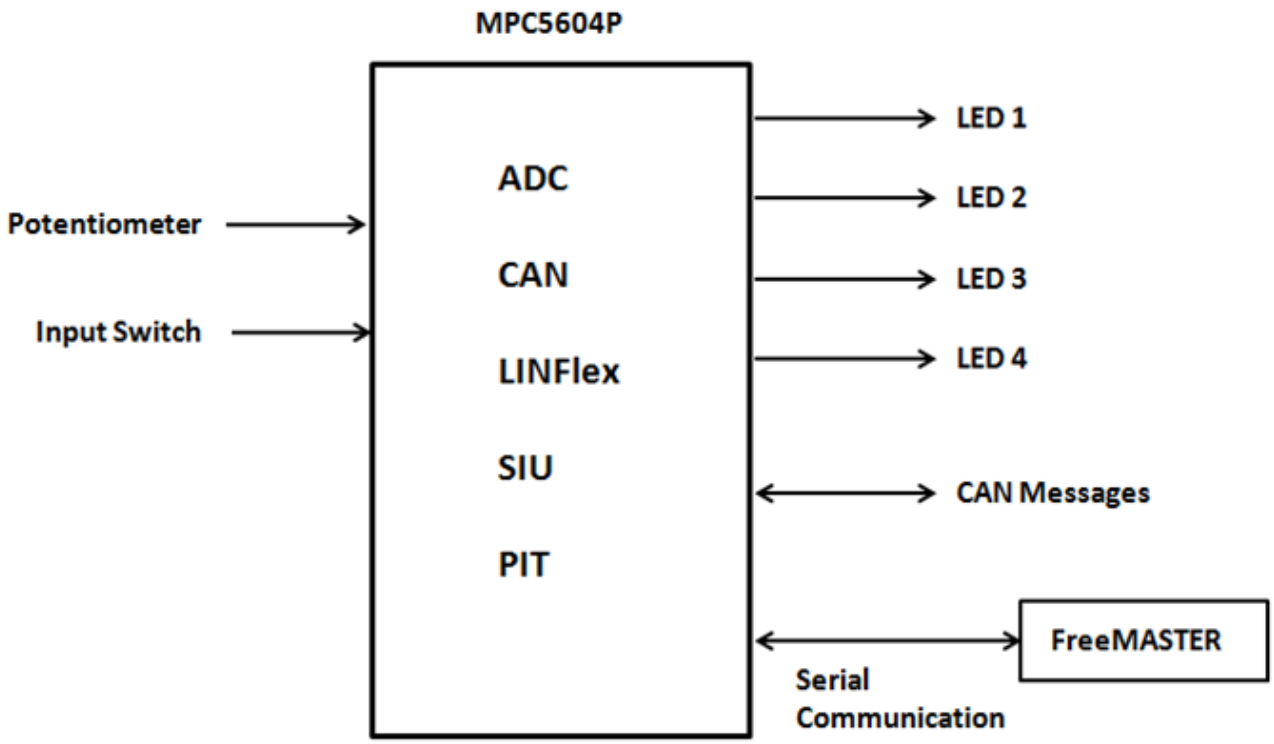

Figure 1. Block diagram of example application

3 Using RAppID tool to configure MPC5604P and generate code

This section describes the steps to configure MPC5604P microcontroller and generate initialization code for the example project using the RAppID tool.

3.1 Pin configuration

The table below shows TRK-MPC5604P board pin connections for the features used in the example project

Table 1. TRK-MPC5604P pin connections used in example project

Feature	Pin	Comments
Potentiometer	PE0	ADC input - ANP0
UART Tx	PB2	Connected to virtual serial port Tx
UART Rx	PB3	Connected to virtual serial port Rx
Switch S1	PD0	Not used in this example
Switch S2	PD1	Not used in this example
Switch S3	PD2	Used to turn on/off LED3 output
Switch S4	PD3	Not used in this example
LED 1	PD4	Turned on/off during PIT interrupt
LED 2	PD5	Turned on/off based on Potentiometer input value
LED 3	PD6	Turned on/off based on input switch S3
LED 4	PD7	Turned on/off based on CAN message
CAN Tx	PB0	Connected to CAN1 Tx via SBC
CAN Rx	PB1	Connected to CAN1 Rx via SBC

The Pin configuration table above shows the relevant pin connection for the functions you will use in this project. You will use the RAppID tool to configure the pins and peripherals for this project.

To create, configure, and generate code for the project using RAppID Init:

1. Double-click the RAppID desktop icon to launch the RAppID application.

Figure 2. RAppID desktop icon

The RAppID window appears.

Figure 3. RAppID main window

2. Click the new project wizard button to start a new project.

3.2 Select part and package

1. Select *MPC5604P* and click the **Start Wizard** button.

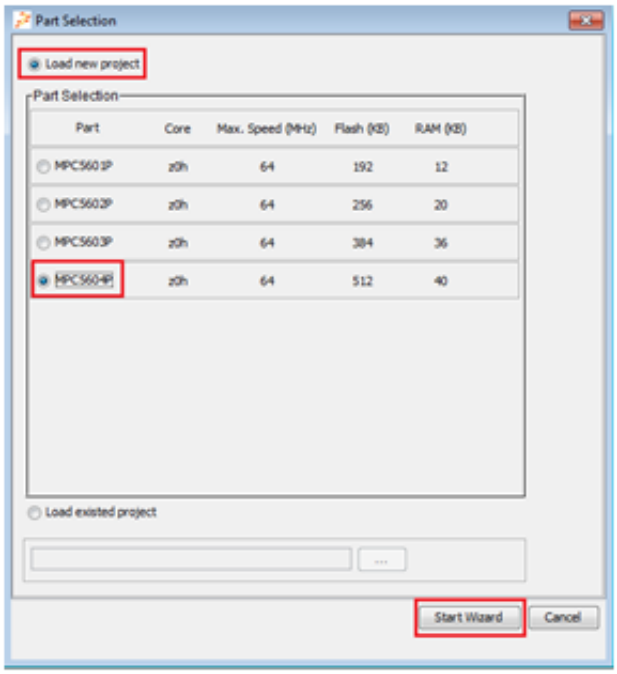

Figure 4. Part selection

2. Select the *144 QFP* package and click the **Next** button.

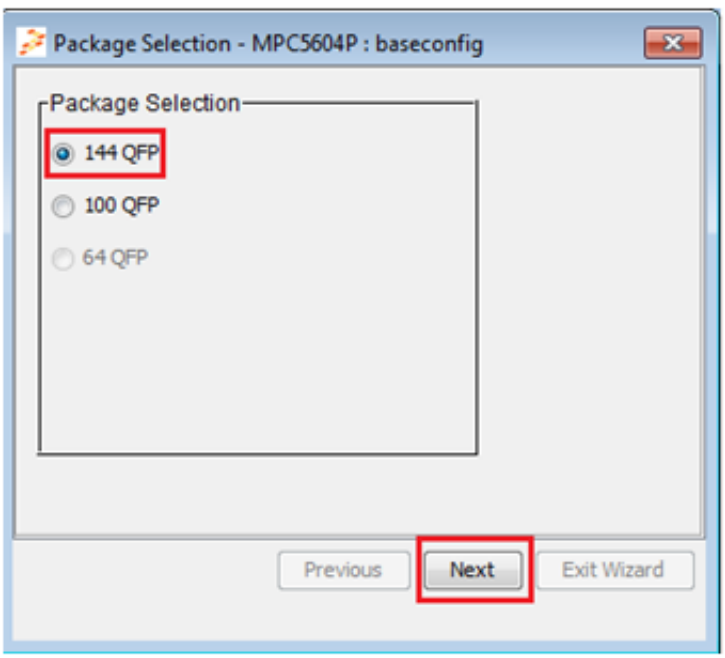

Figure 5. Package selection

3.3 Configure ADC pins

In TRK-MPC5604P, the potentiometer is connected to input ADC1 channel 5 (PE0).

To configure ADC inputs:

1. Select the ADC tab in the Pin Allocation window.

2. Select PE0 as ADC input and enter the user signal names as shown below.

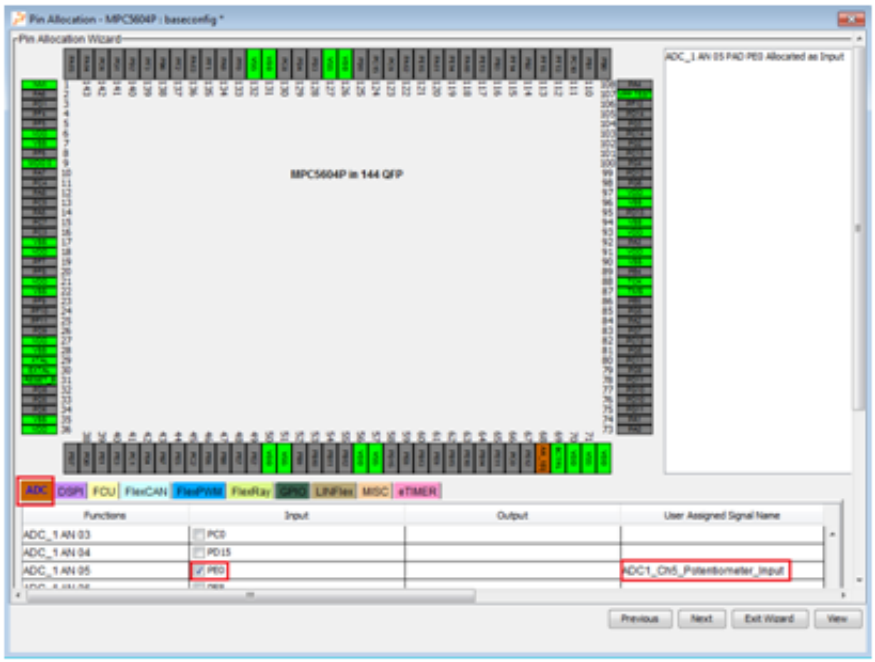

Figure 6. Configure ADC pins

3.4 Configure DSPI pins

TRK-MPC5604P contains MCZ3390S5EK system basis chip (SBC) with integrated CAN transceiver and LIN 2.0 interface. Since DSPI 0 of the MPC5604P is connected to SBC, you can use the DSPI 0 peripheral to configure the SBC and enable CAN communication by sending appropriate commands via DSPI 0.

The figure below shows the connections between the SBC and DSPI 0 peripheral.

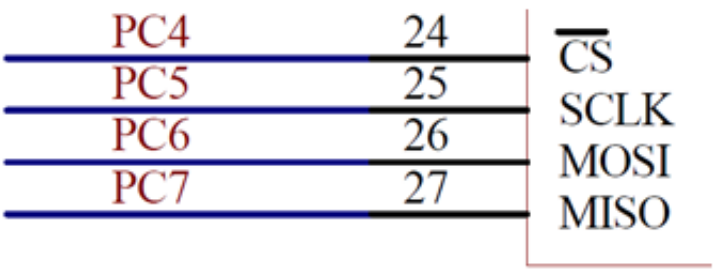

Figure 7. Connection between SBC and DSPI 0

Configure DSPI 0 pins PC4, PC5, PC6, and PC7 using RAppID as shown below.

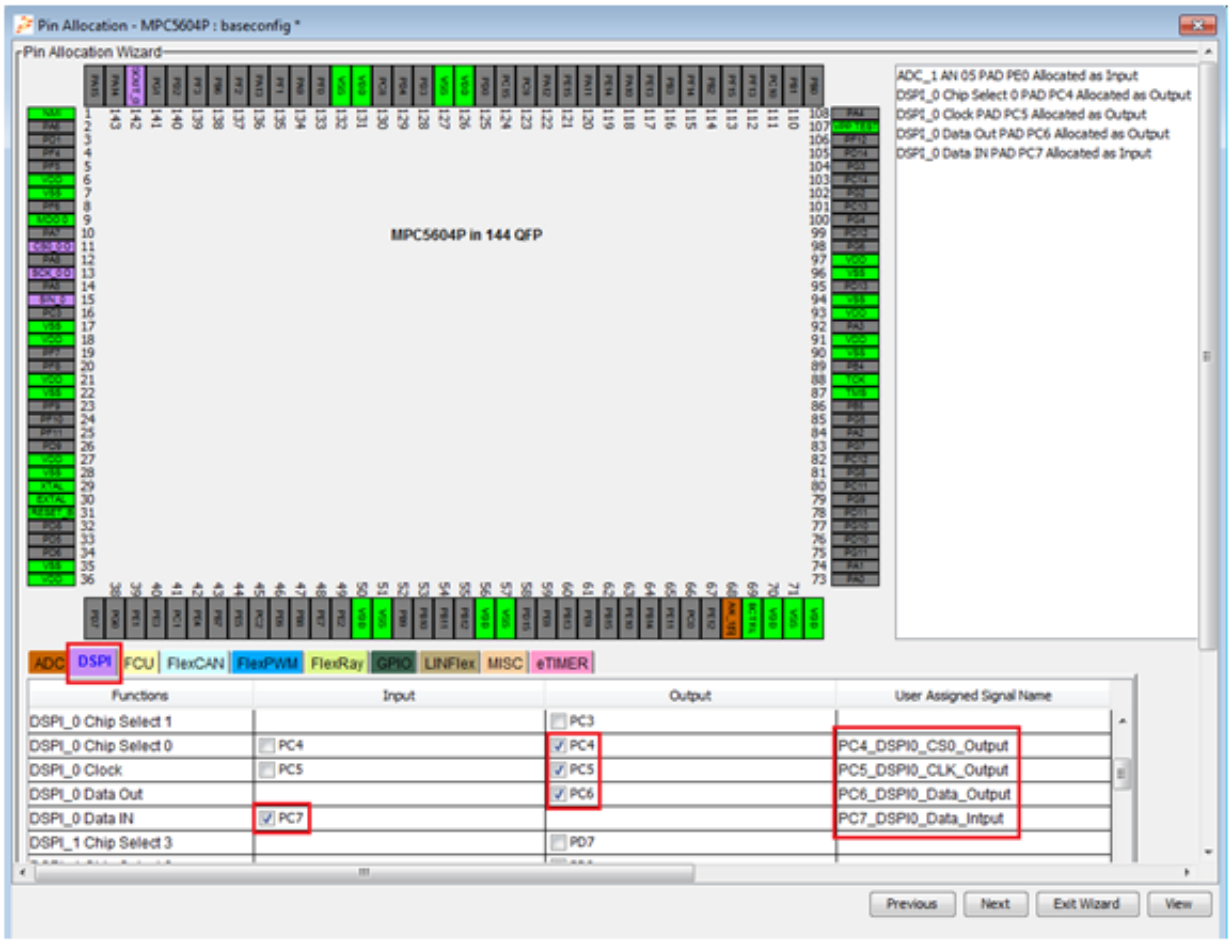

Figure 8. Configure DSPI 0 pins

3.5 Configure FlexCAN pins

The CAN TX and CAN RX pins of the SBC are connected to the pins PB0 and PB1 of CAN 0 peripheral of the microcontroller as shown in the figure below.

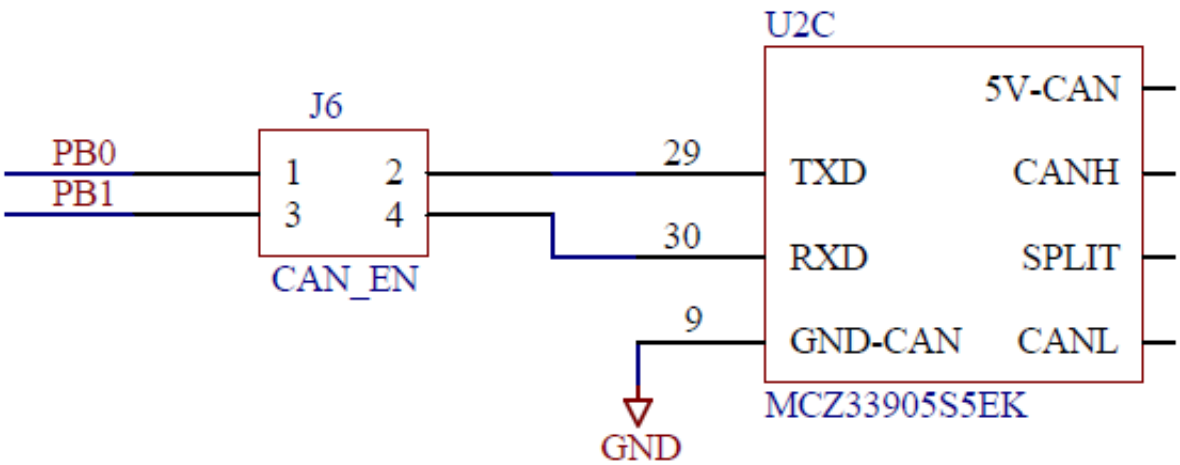

Figure 9. CAN TX and CAN RX pins of SBC connected to pins PB0 and PB1

Configure FlexCAN pins PB0 and PB1 using RAppID as shown below.

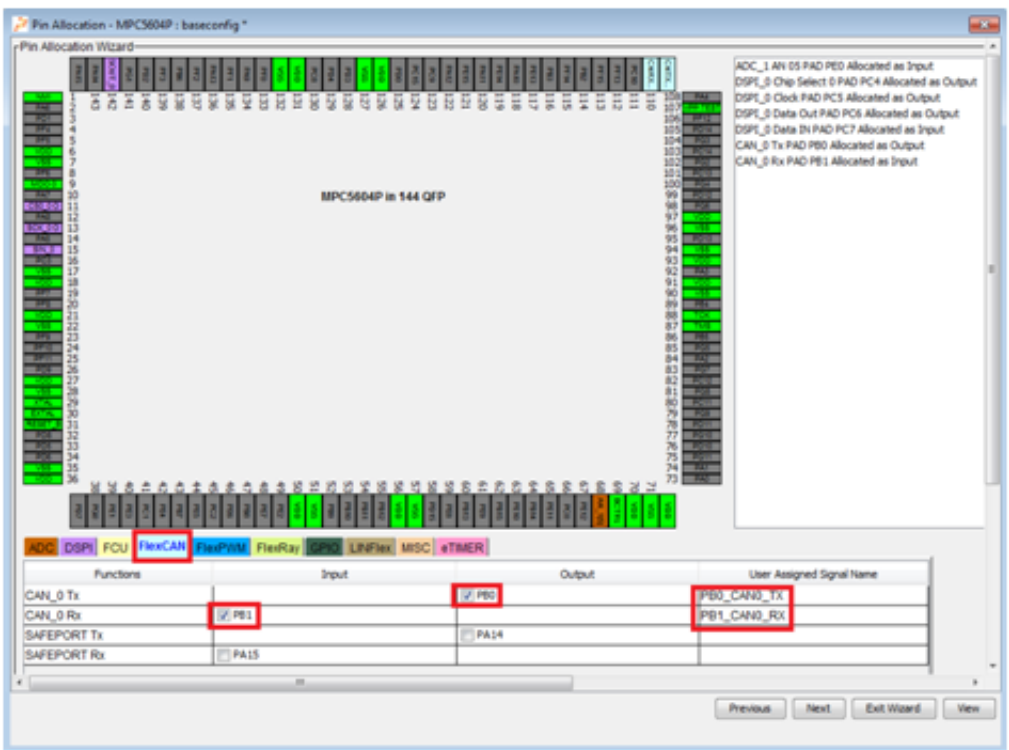

Figure 10. Configure FlexCAN pins

3.6 Configure GPIO pins

In this example, you use Switch S3 as input and LED1, LED2, LED3, and LED4 as general purpose outputs. The switch S3 is connected to PD2 pin of the microcontroller and LED1, LED2, LED3, and LED4 are connected to PD4, PD5, PD6, and PD7 pins of microcontroller. Configure GPIO pins using RAppID as shown below.

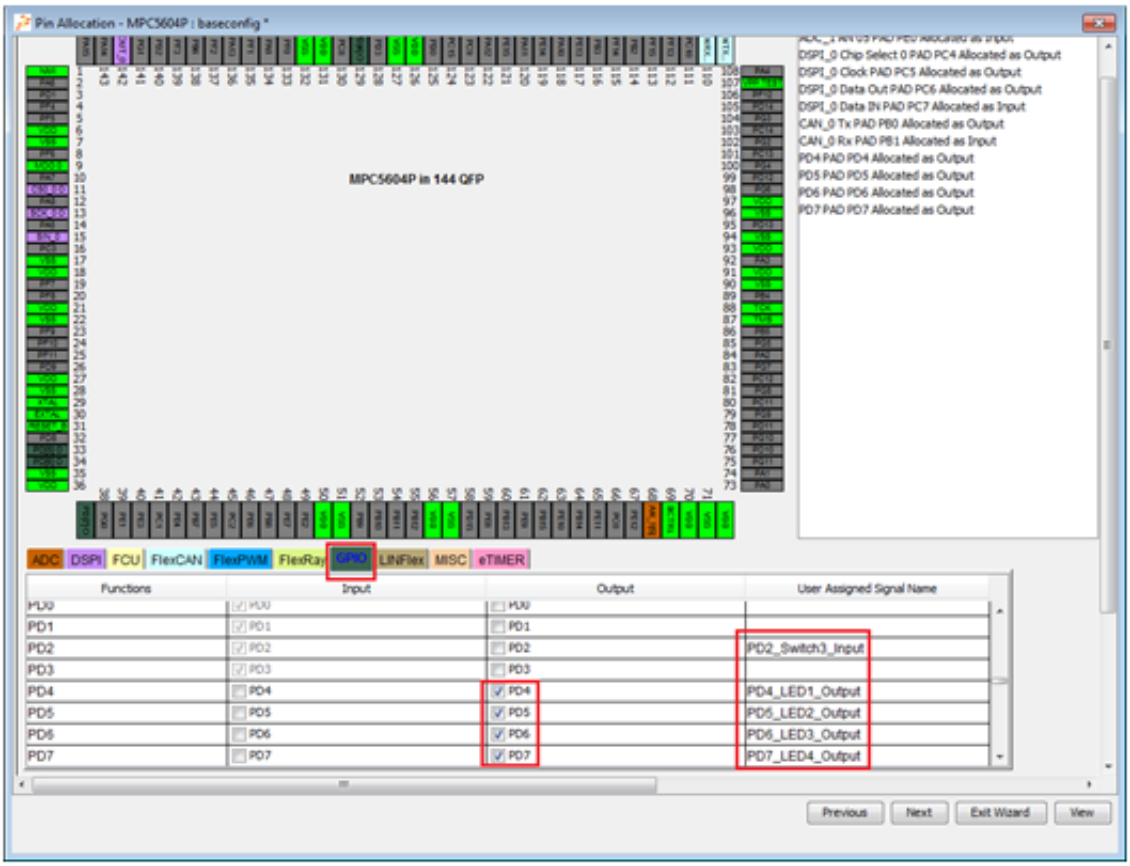

Figure 11. Configure GPIO pins

3.7 Configure LINFlex (UART) pins

In this example, use the virtual serial port of TRK-MPC5604P board for serial communication. The PB2 and PB3 pins of microcontroller in TRK-MPC5604P board are connected to TX and RX pins of virtual serial port. Configure LINFlex 0 pins using RAppID as shown below.

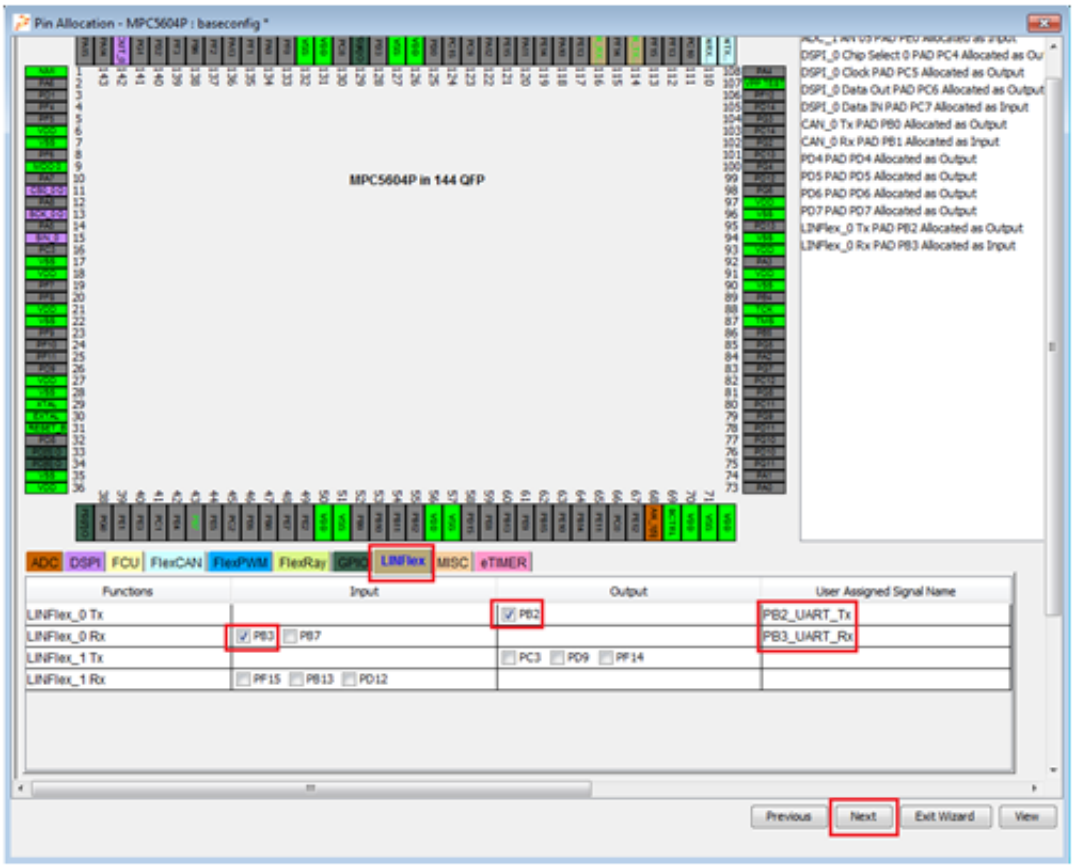

Figure 12. Configure LINFlex (UART) pins

Now we are finished with the entire pin configuration required for this project. Next, configure Mode Entry. Select *Next* button and skip Core Configuration window by selecting *Next* button.

3.8 Mode entry configuration

RAppID generates code to put the target in DRUN mode. This example uses PLL as clock source in the DRUN mode.

To configure mode entry:

1. Select the General Configuration tab.
2. Select System PLL (PLL0) from the drop-down menu as SYSCLK source for DRUN.

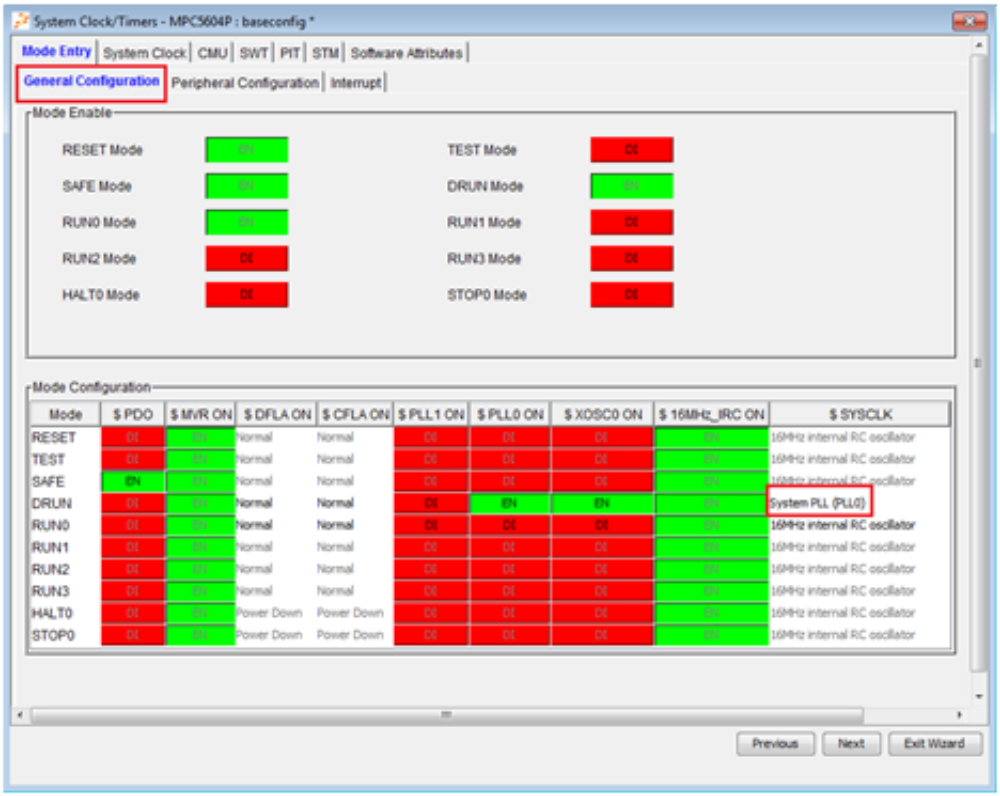

Figure 13. Mode Entry clock configuration

3. Configure peripheral behavior during run modes and non-run modes. RAppID provides an easy way to configure peripheral behavior during these two modes by providing one click buttons: Normal, Run, Low Power, and Stop modes. The Normal button sets Run Peripheral Configuration 0 and Low Power Configuration 0 to be enabled in all modes and selects Configuration 0 during run and non-run modes.
4. Click the Normal button to select Peripheral Configuration 0 for run and non-run modes.

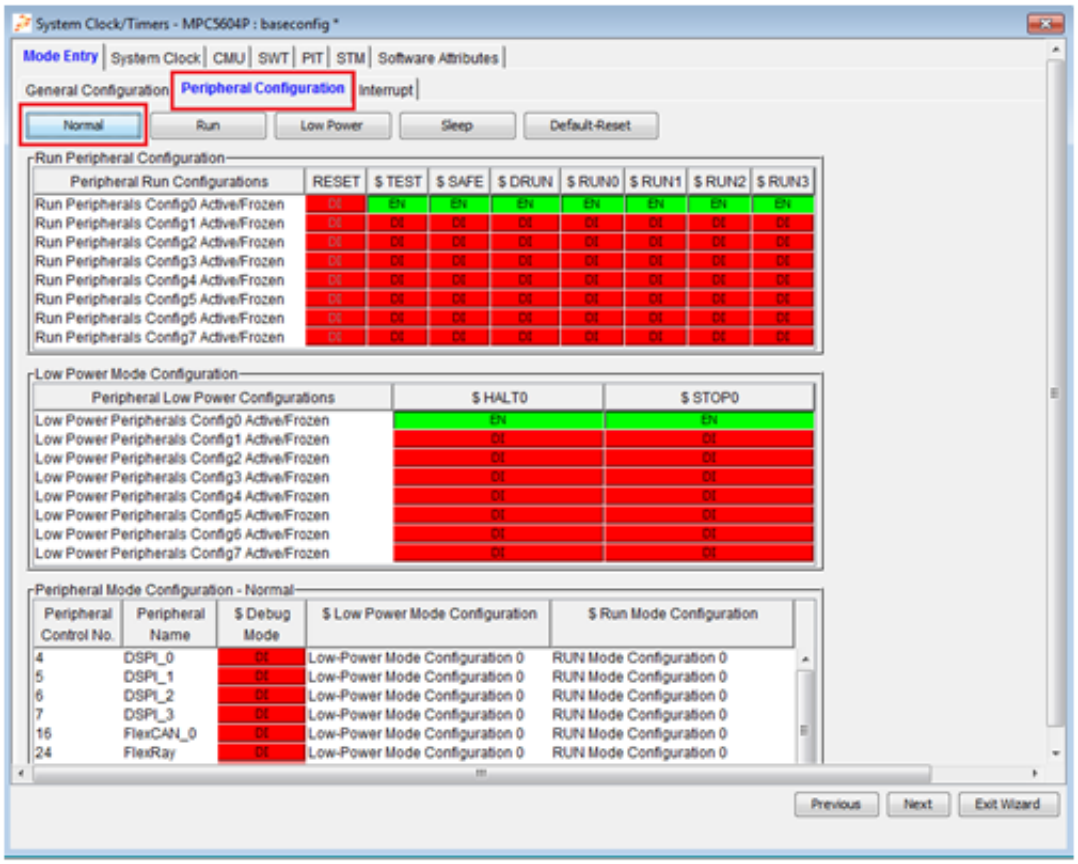

Figure 14. Mode Entry peripheral configuration

3.9 System clock

Select System Clock tab to configure System Clock parameters. The default XOSC frequency in RAppID is 40 MHz but TRK-MPC5604P uses 8 MHz crystal. Change XOSC value to 8 MHz as shown in the figure below. Select Input Division Factor value of 2 and Output Division Factor value of 4 from the drop down list as shown in figure below. This will configure the system clock to 64 MHz.

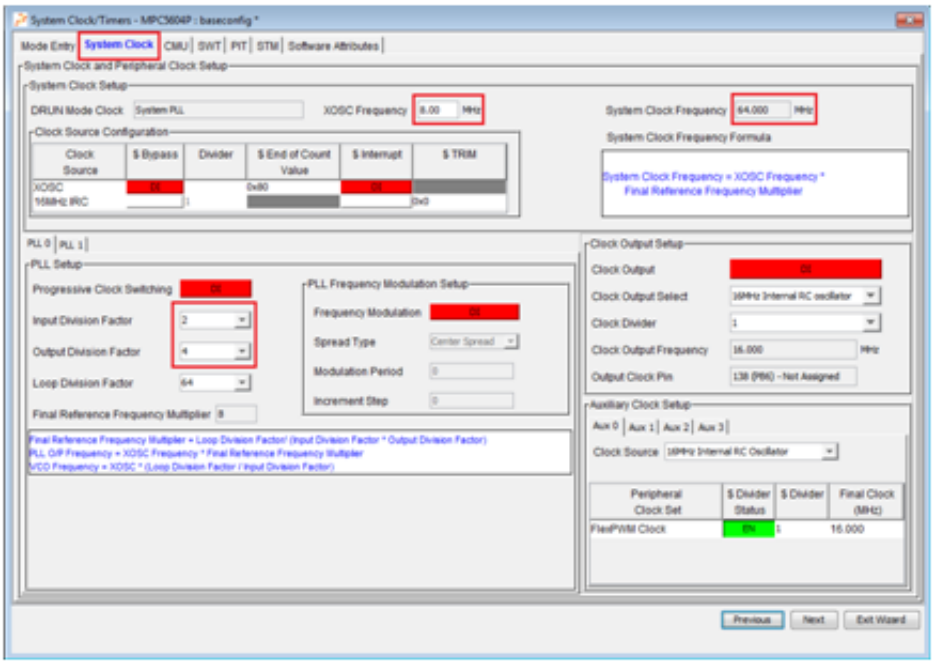

Figure 15. System clock configuration

3.10 Watchdog configuration

By default, the watchdog is enabled in MPC5604P. This example does not use the Watchdog feature Thus, you need to disable the Watchdog Timer in SWT tab as shown below.

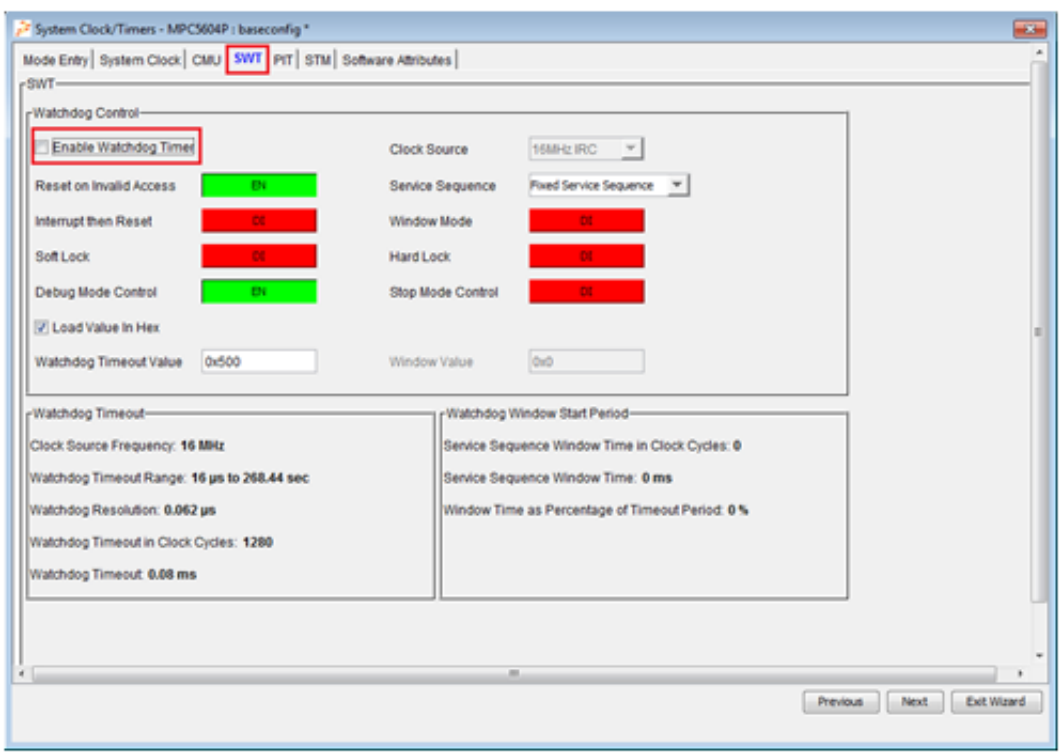

Figure 16. SWT configuration

3.11 PIT configuration

You will use the PIT channel 0 interrupt to toggle LED1 output every 250 ms. Configure PIT interrupt at 250 ms using RAppID as follows:

1. Enable timer module
2. Enable channel 0 Timer and enter the Load Value as 16,000,000. This will result in PIT timeout value at 250 ms with system time at 64 MHz.
3. Enable Channel 0 interrupt.
4. This completes the basic system configuration. Click Next to begin Peripheral configuration.

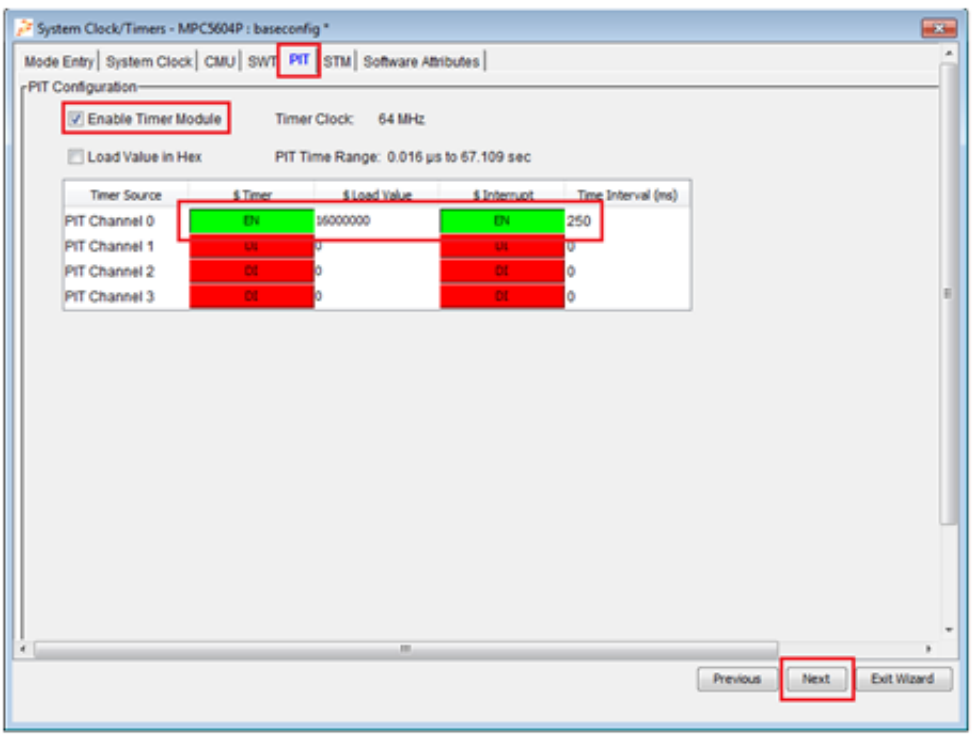

Figure 17. PIT configuration

3.12 Peripheral configuration

In the next window, RAppID displays all the peripherals you need to configure, based on the pins configured in the previous steps. To start configuring DSPI peripheral, select the DSPI tile shown below.

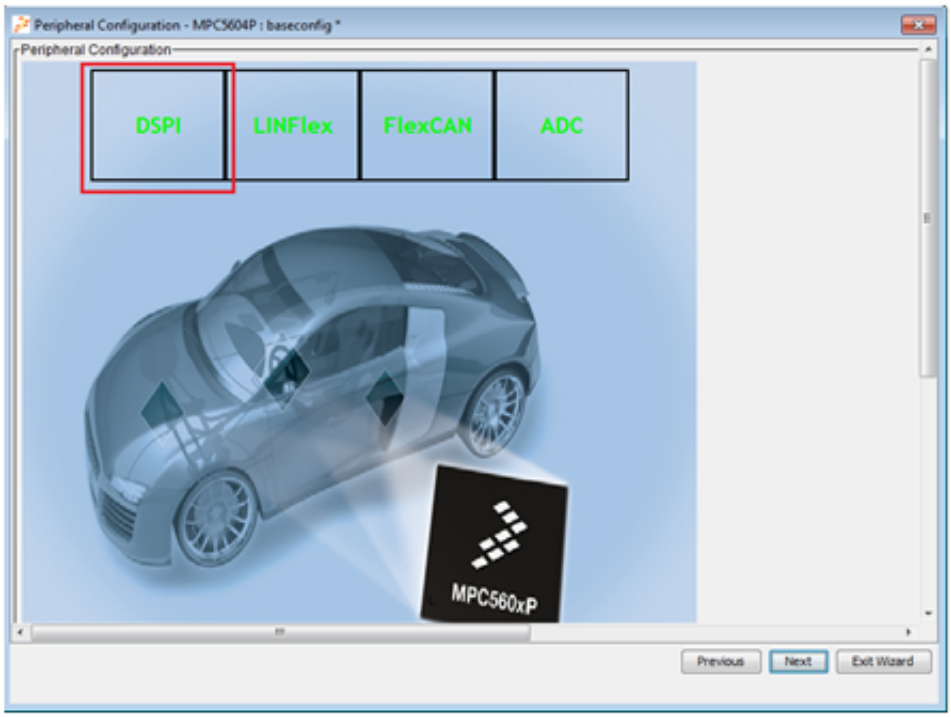

Figure 18. Select DSPI

3.13 DSPI 0 configuration

1. To send commands to SBC, set DSPI 0 to master mode.
2. Configure DSPI 0 peripheral as shown below.
3. Select *Master* mode, set Chip select 0 inactive state to *High* and *Disable* Halt mode.
4. Click *OK* to finish the DSPI 0 configuration

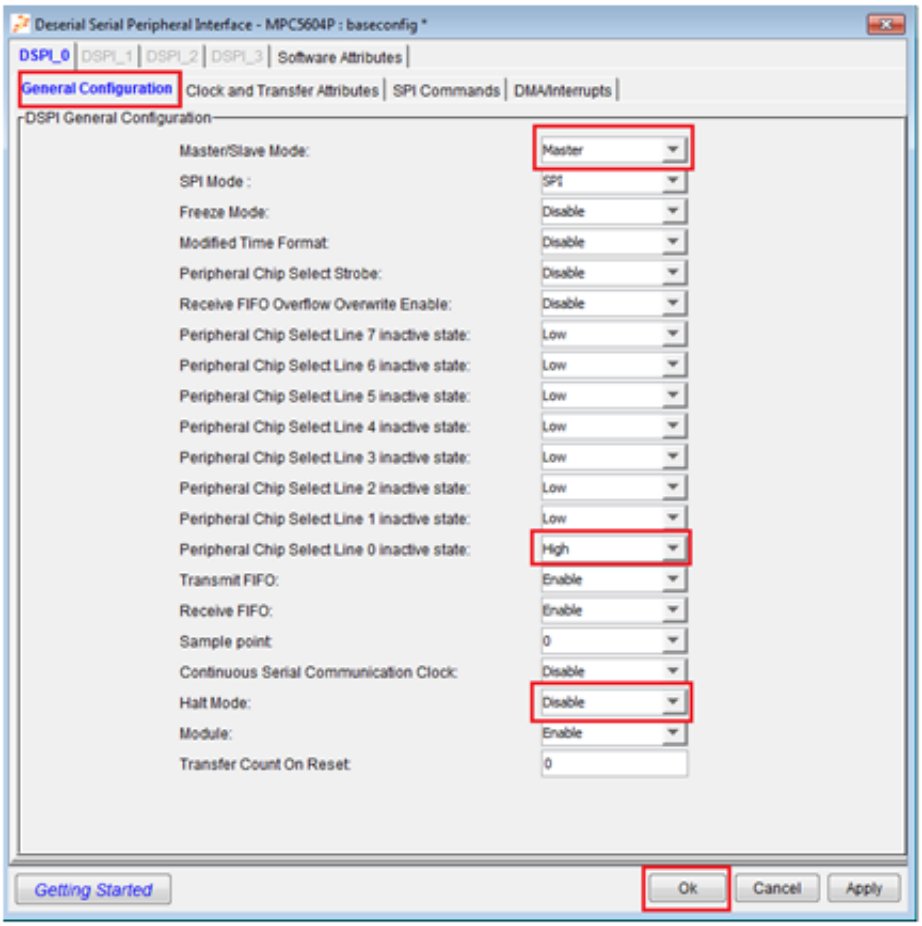

Figure 19. DSPI configuration

3.14 LINFlex (UART) configuration

To configure the LINFlex peripheral:

1. Select LINFlex tile from the Peripheral Configuration window.

   You will use the virtual serial port of TRK-MPC5604P board to communicate with FreeMASTER utility at a baud rate of 115,200.

2. To select baud rate of 115,200, set Integer Baud Rate Factor to *34* and Fractional Baud Rate Factor to *12/16*.

   When Baud Rate Factor and Fractional Baud Rate Factor values are selected, RAppID automatically calculates and displays the resulting baud rate. As indicated in the figure below, the resulting baud rate is 115,107 which is close enough to the required baud rate of 115,200.

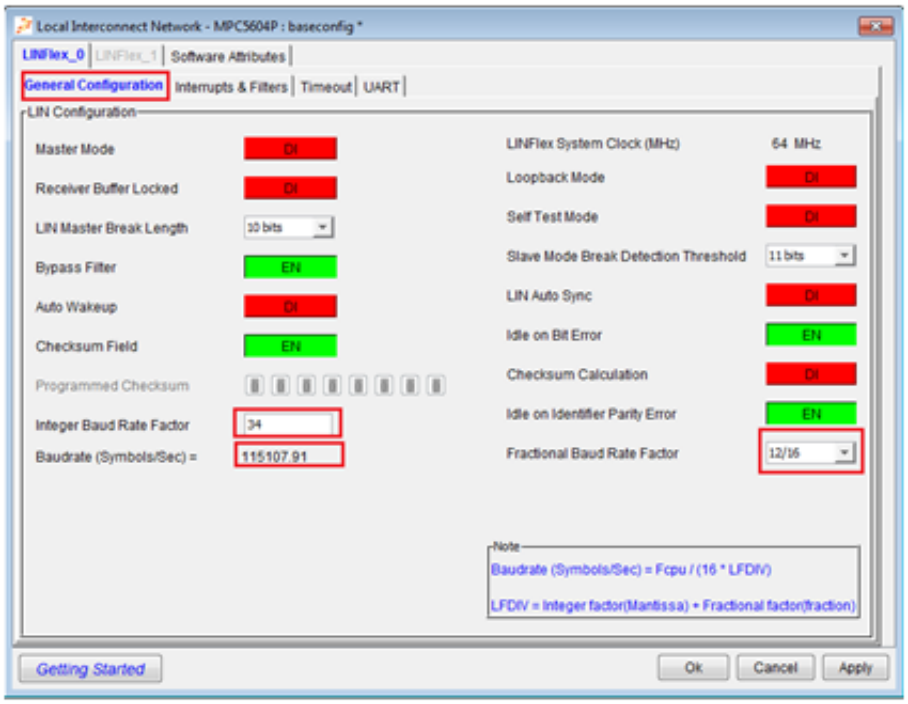

Figure 20. LINFlex configuration

3. In the UART tab, *enable* UART, set Word Length to *8-bit* data and *enable* Transmitter and Receiver.
4. Select *OK* to finish LINFlex configuration.

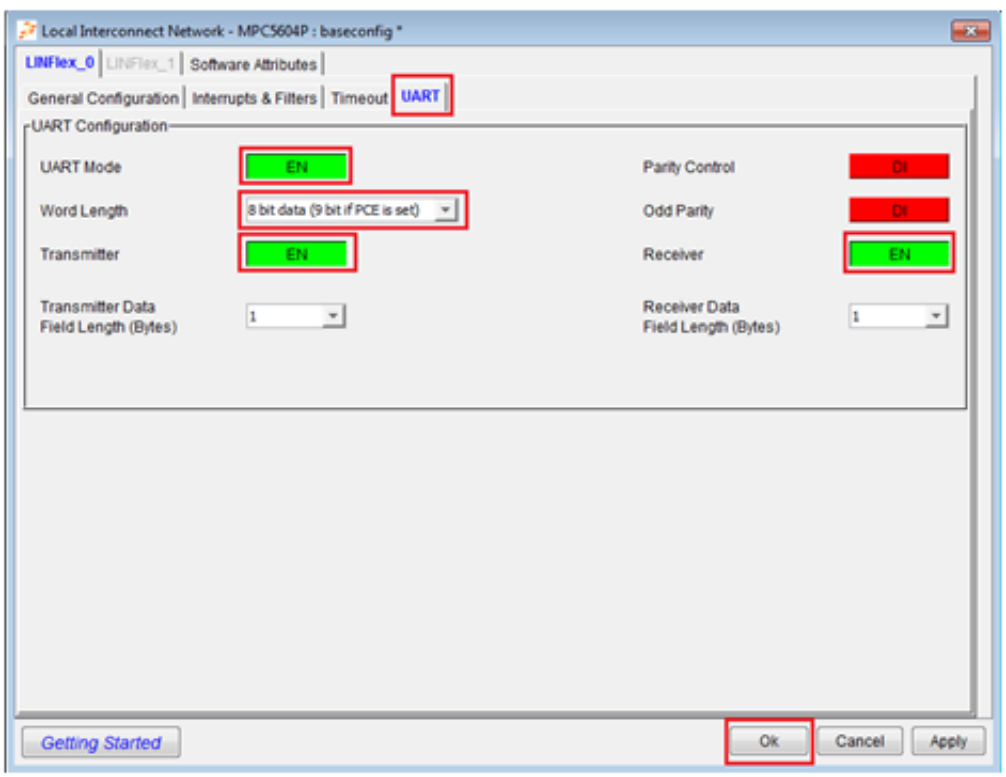

Figure 21. UART configuration

3.15 FlexCAN configuration

To configure FlexCAN peripheral:

1. Select FlexCAN tile from the Peripheral Configuration window. In TRK-MPC5604P board, the CAN 0 peripheral is connected to CAN transceiver in SBC. In this example, you will use CAN speed of 500 kbit/s.
2. To configure CAN 0:
   - *Enable* CAN0 module
   - *Disable* Freeze and Halt modes
   - Set Clock Source to System
   - Set CAN speed to *500* kbit/s. RAppID init configures Phase segments and Propagation segment values automatically.

   `
3. Select *OK* to finish CAN 0 configuration.

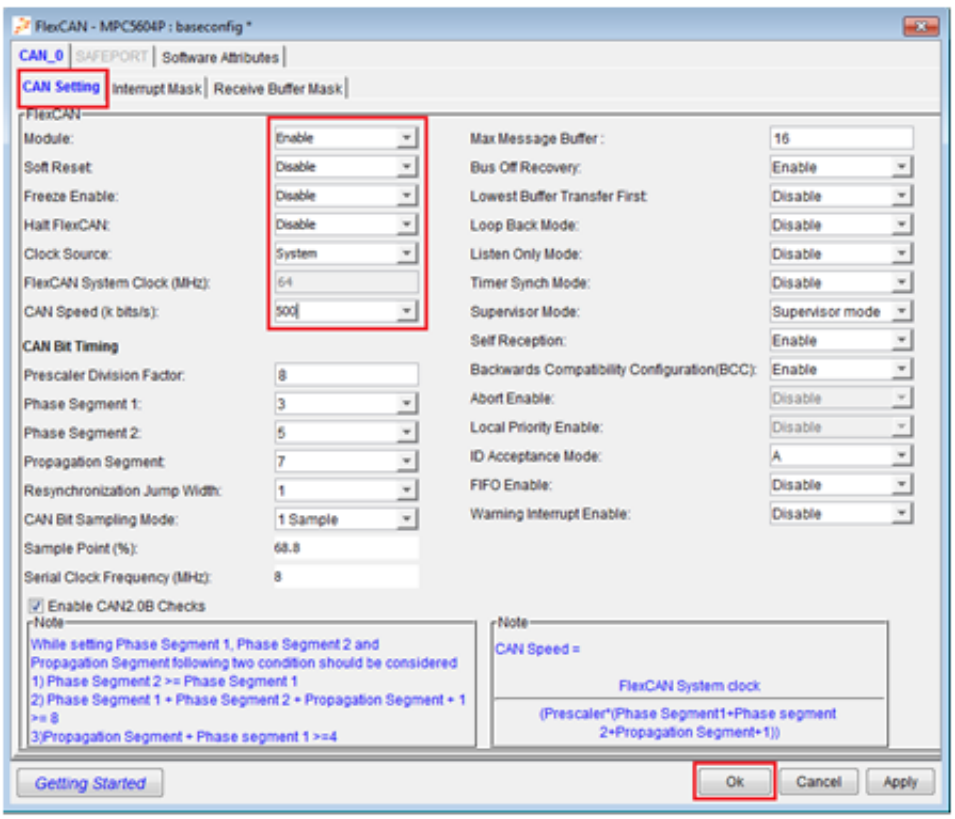

Figure 22. CAN configuration

3.16 ADC configuration

To configure ADC peripheral:

1. Select ADC tile from the Peripheral Configuration window.
2. In the Device Setup tab, *disable* Power Down Enable option.

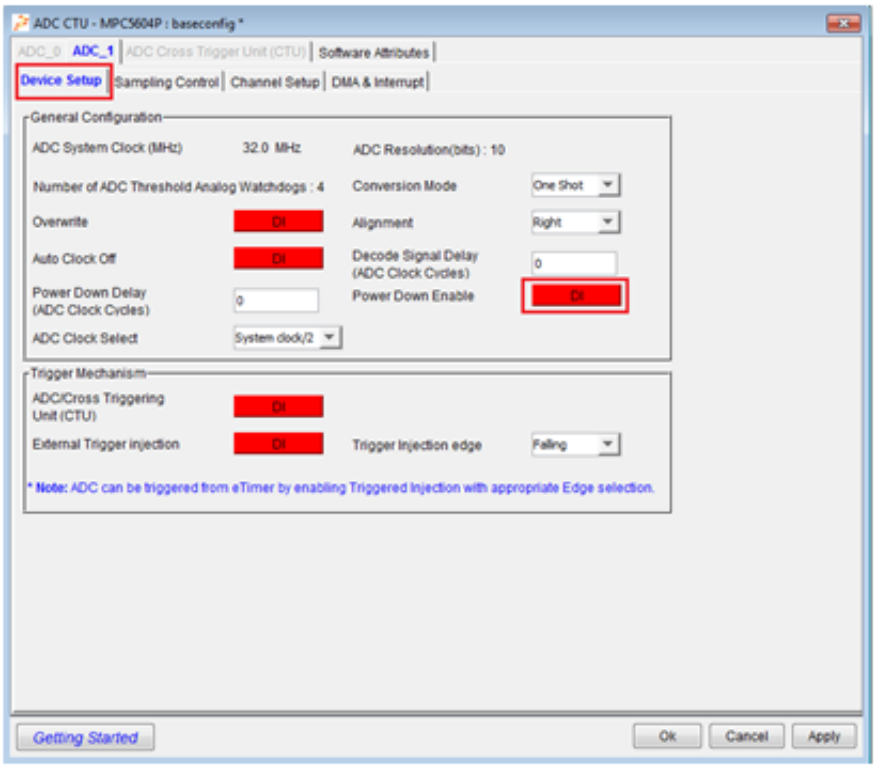

Figure 23. ADC configuration

3. In the Channel Setup tab, enable Channel 5 in Normal Mode. This channel represents Potentiometer input.
4. Select OK to finish the ADC configuration.

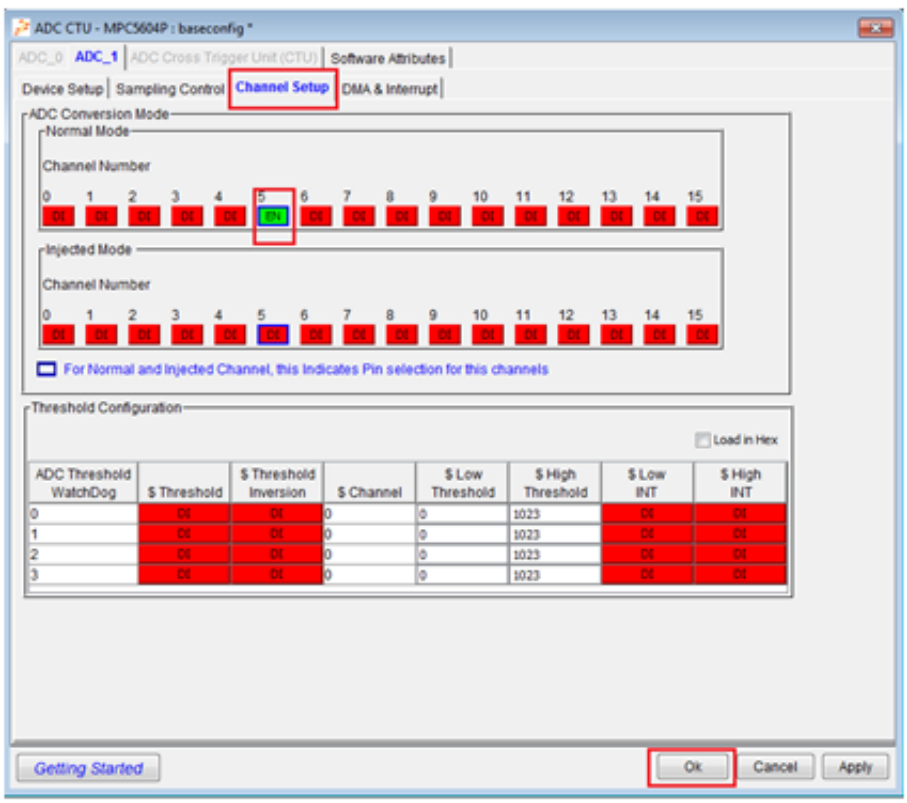

Figure 24. ADC channels configuration

This completes the configuration of peripherals.

1. Select *Next* to configure PIT interrupt.

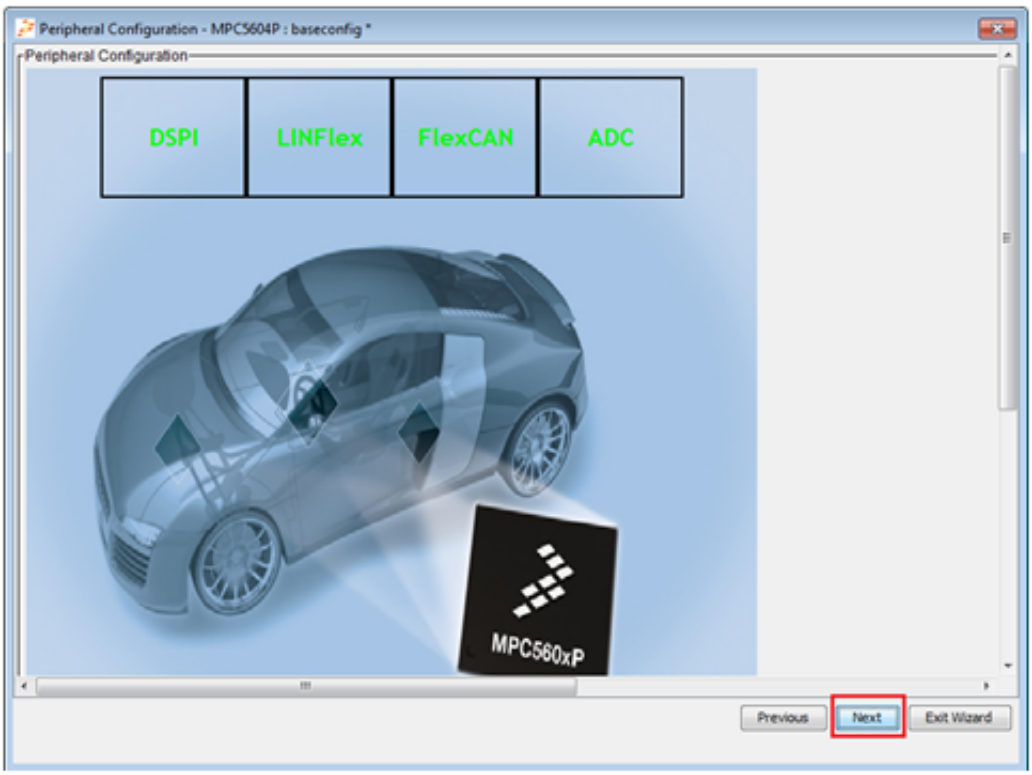

Figure 25. Completing peripheral configuration

3.17 Interrupt configuration

In this example, PIT Channel 0 interrupt is used to toggle LED1 output. Configure PIT Ch0 interrupt as shown. Click on PIT Channel 0 Edit button to add code to ISR function.

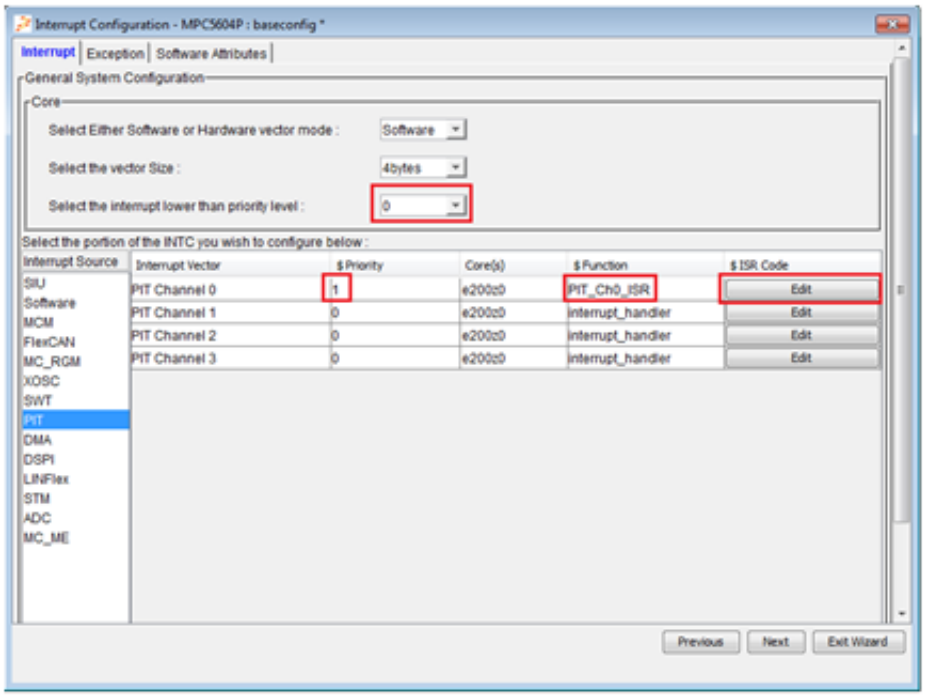

Figure 26. PIT Interrupt configuration

Add code to ISR function to toggle LED 1 output as shown below and select OK button to exit ISR code window.

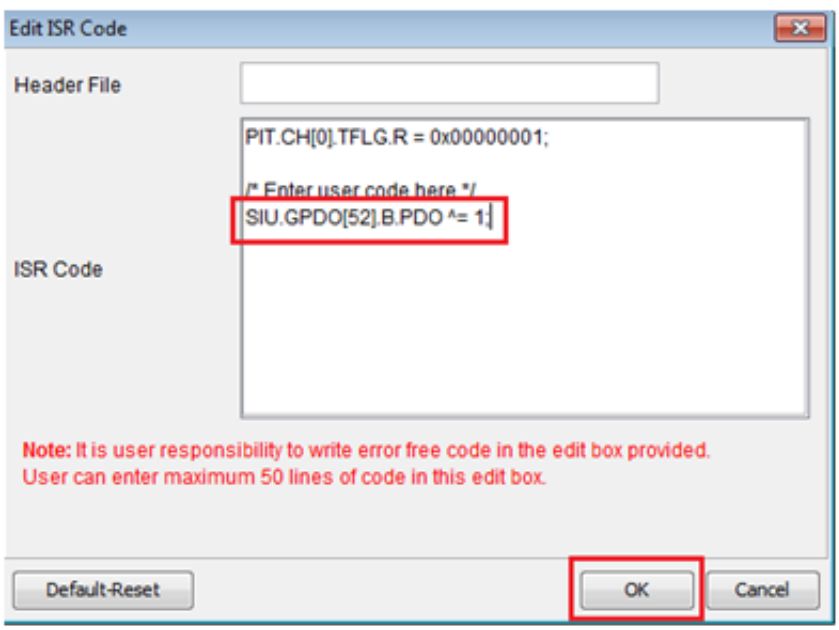

Figure 27. PIT Interrupt code

This completes the microcontroller configuration. Select *Exit* Wizard to exit to main RAppID window.

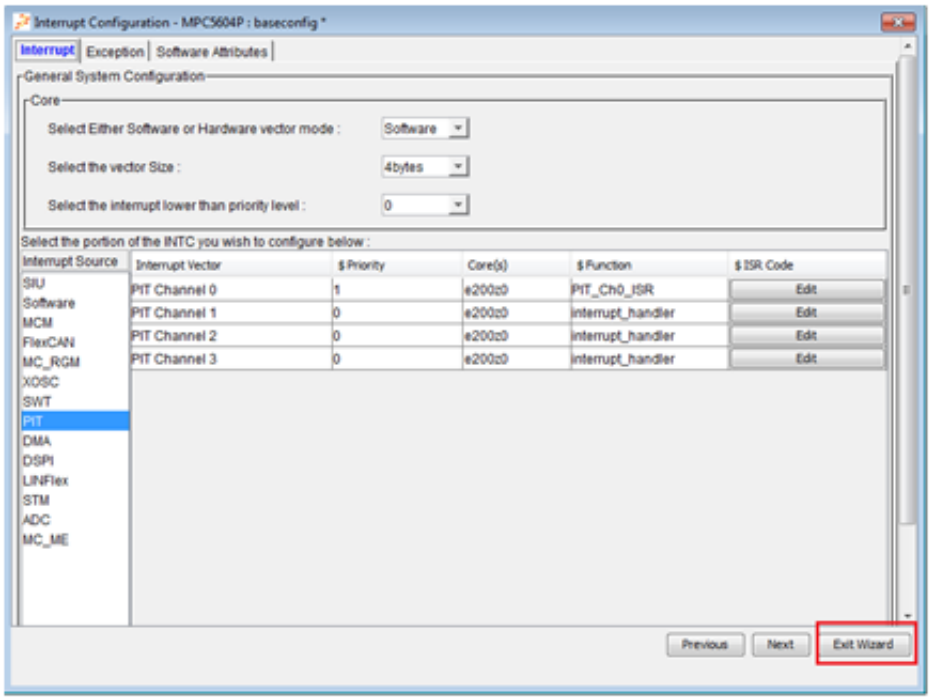

Figure 28. Exit Wizard

3.18 Code configuration

Once you have completed all pin and peripheral configurations required for this example project, you are ready to generate the code.

1. From main RAppID window, select menu *Configuration* > *Code Generation*. This will pop up Code Generation window shown below. By default, RAppID generates code for all peripherals. In this example, you need to select only the peripherals that are used in the example for code generation.
2. Deselect the peripherals that are not required (Flash BIU, MSR, RGM, eDMA, FCU, FlexRay, and eTIMER).
3. Select code generation for ECC to initialize SRAM and ECC registers.
4. Select CodeWarrior compiler option.
5. By default, RAppID generates code for RAM. In this example you generate code for Flash. Select *Generate Code For Flash* option.
6. Enter the path where code should be generated and select *OK* to complete Code Configuration options.

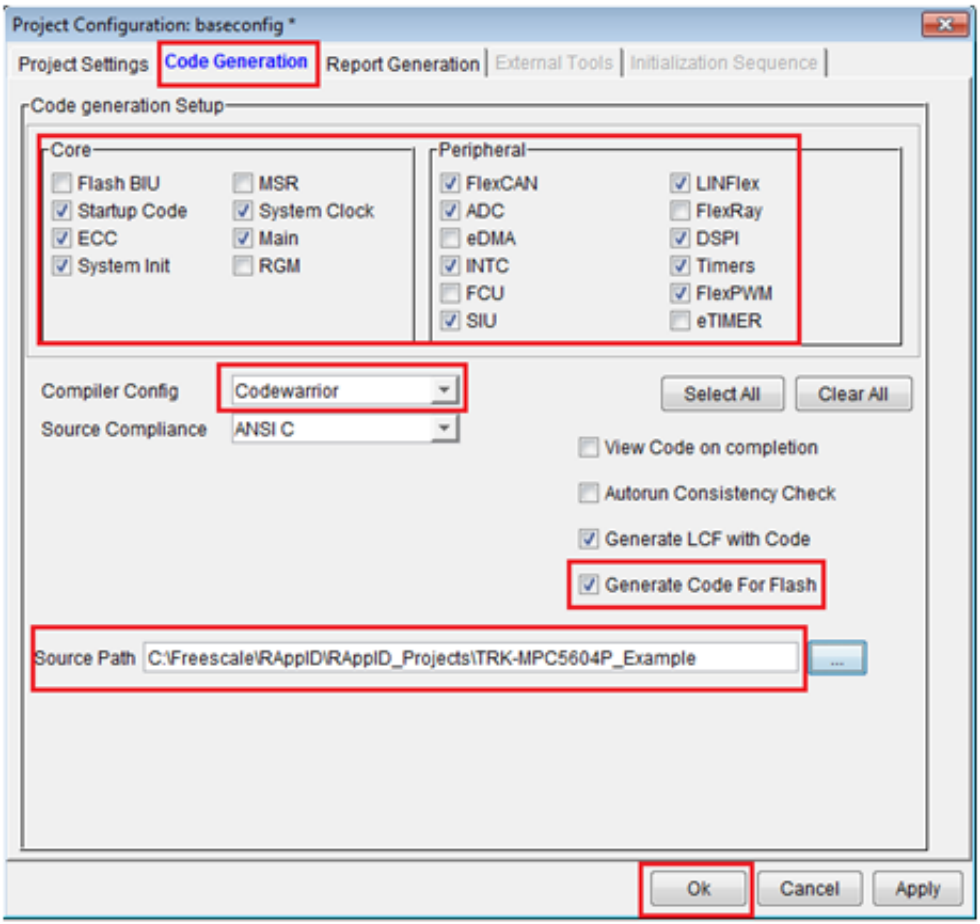

Figure 29. Code configuration

Since you are generating code for Flash, you need to generate linker file for Flash as well.

7. From main RAppID window, select menu *View*> *View Section Map*.
8. Change Target to *Flash*.
9. Select *Ok*.

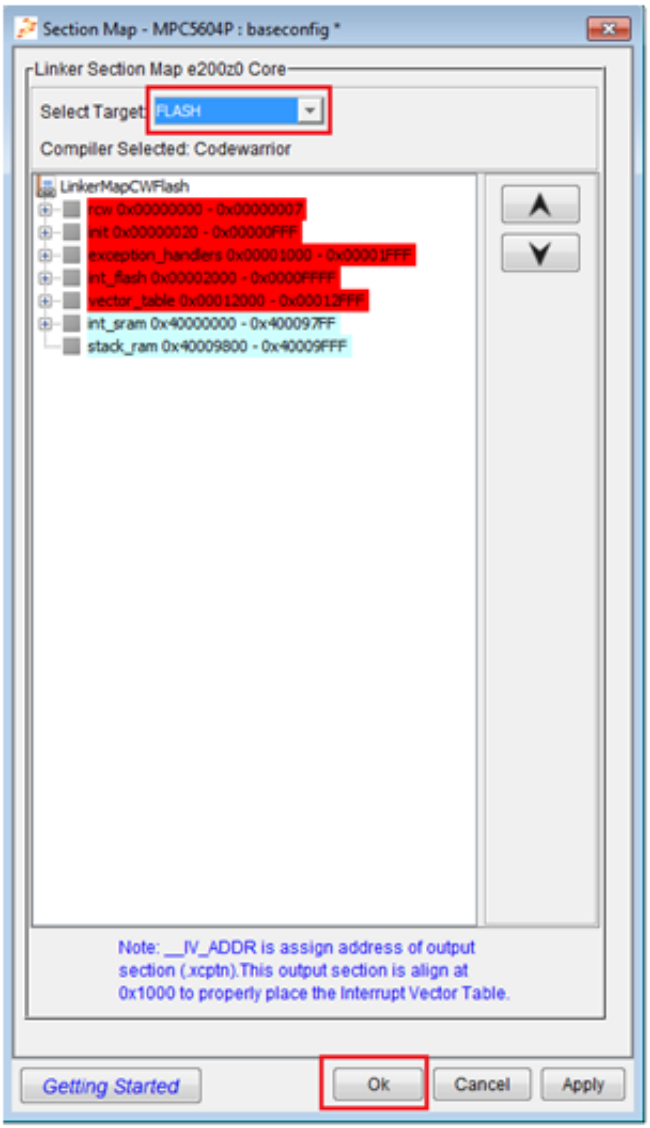

Figure 30. Section map

3.19 Generate code

Select Generate Code icon to generate code. Save the project when prompted. You are now ready to build the project using CodeWarrior.

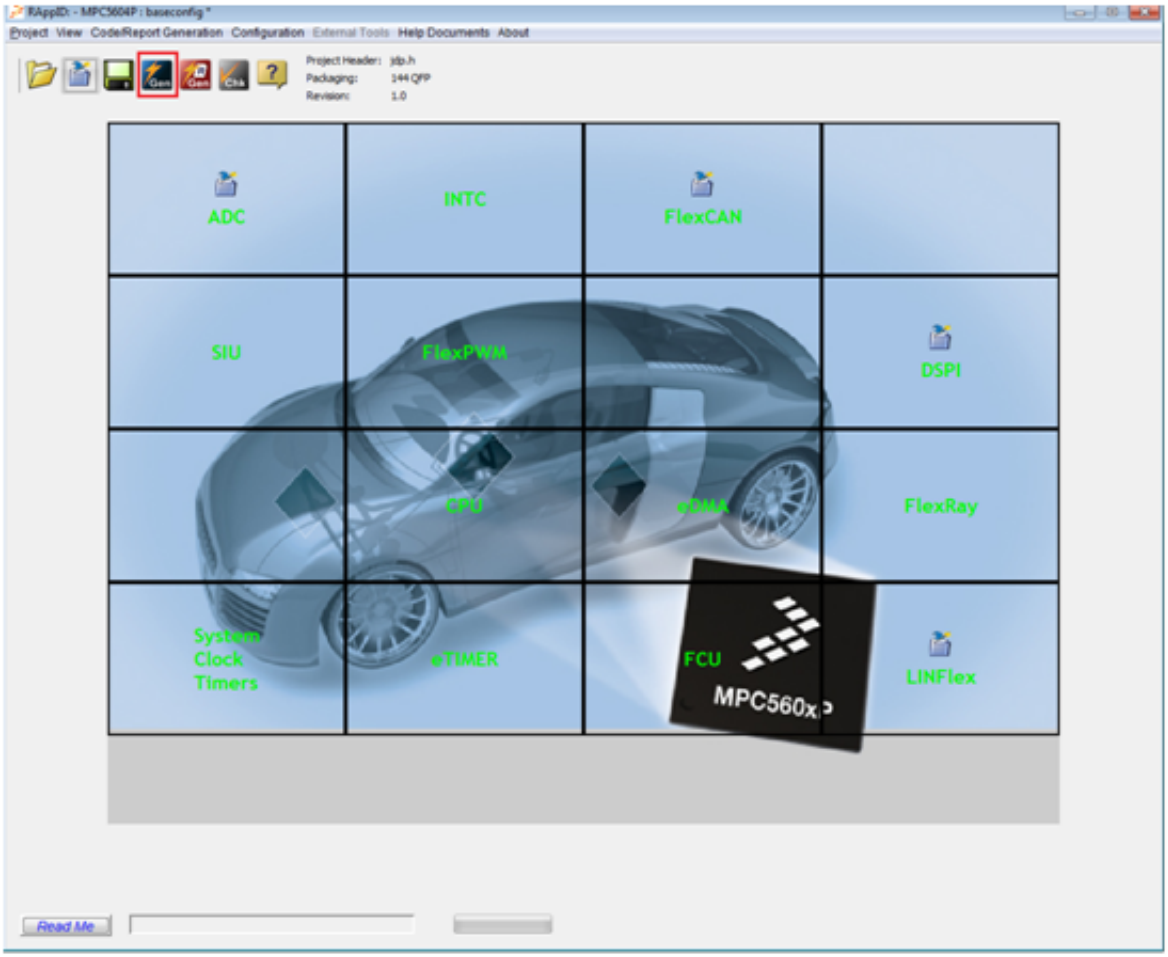

Figure 31. Generate code

In addition to RAppID generated code, you will use the driver code supplied with Fast Start Kit and FreeMASTER code in this example project. During the installation, driver code should be installed at the folder *C:\Freescale\FastStartKit \DriverCode\TRK-MPC5604P*. Copy all the driver code to the folder where RAppID code is generated.

The FreeMASTER code should be installed in the folder *C:\Freescale\FastStartKit \FreeMASTER Serial Communication V1.6*. Copy all the code from sub-folders *src_common* and *src_platforms\MPC56xx* to the location where RAppID code is generated.

FreeMASTER can be used in polling or interrupt mode via CAN or SCI. In this example, you will use FreeMASTER in poll mode via SCI.

To use FreeMASTER in polling or interrupt mode you will have to make changes to FreeMASTER configuration header file.

1. Rename *freemaster_cfg.h.example* file to *freemaster_cfg.h*. This file contains all the macro definitions available for the FreeMASTER configuration.
2. Select poll driven SCI communication and disable TSA by making following changes to *freemaster_cfg.h*:

```
#define FMSTR_SHORT_INTR        0
#define FMSTR_POLL_DRIVEN       1
#define FMSTR_USE_TSA           0
```

4 Build code using CodeWarrior

To compile and build RAppID generated source code, create an empty CodeWarrior project and use *rsp2cw10* utility to add the RAppID generated source code to the project.

4.1 Create CodeWarrior project

To create a CodeWarrior project:

1. Launch CodeWarrior 10.5 from Windows Start menu and provide a workspace name.
2. Select *OK*.

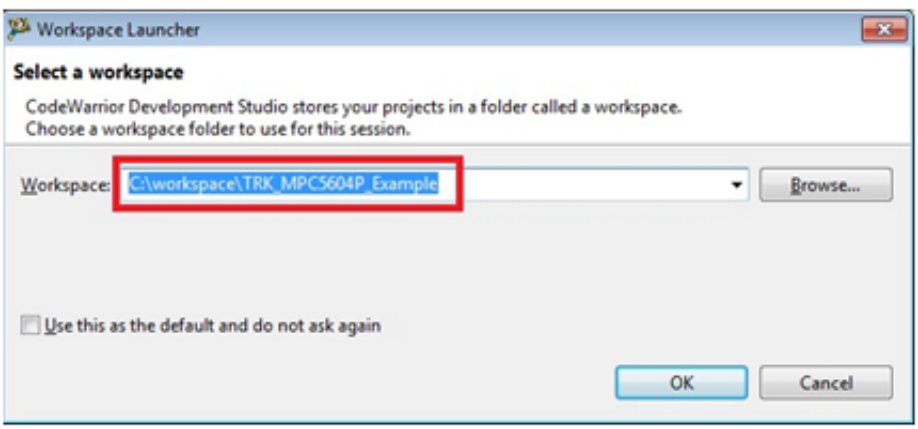

Figure 32. CodeWarrior Workspace Launcher

To create an empty bare board project:

1. Select *New MCU project* from Commander window.

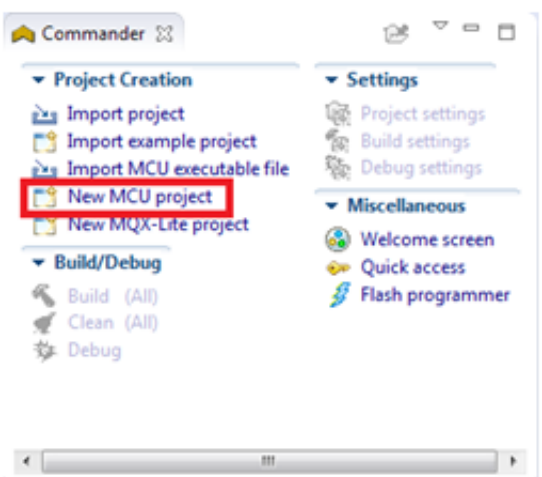

Figure 33. Start New CodeWarrior project

2. Provide a name for the project and select *Next*.

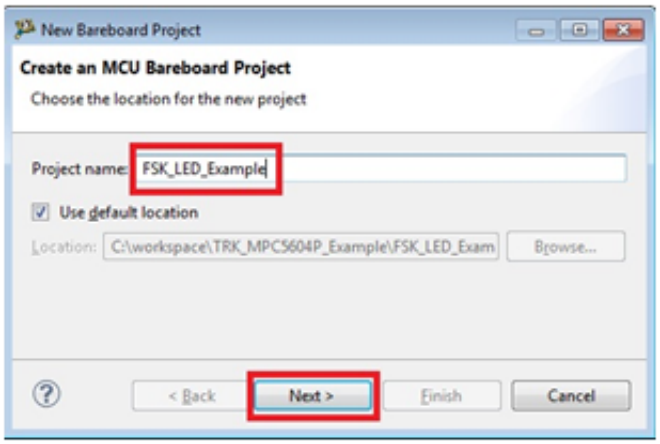

Figure 34. Provide a project name

3. Select *5604P device* and select *Next*.

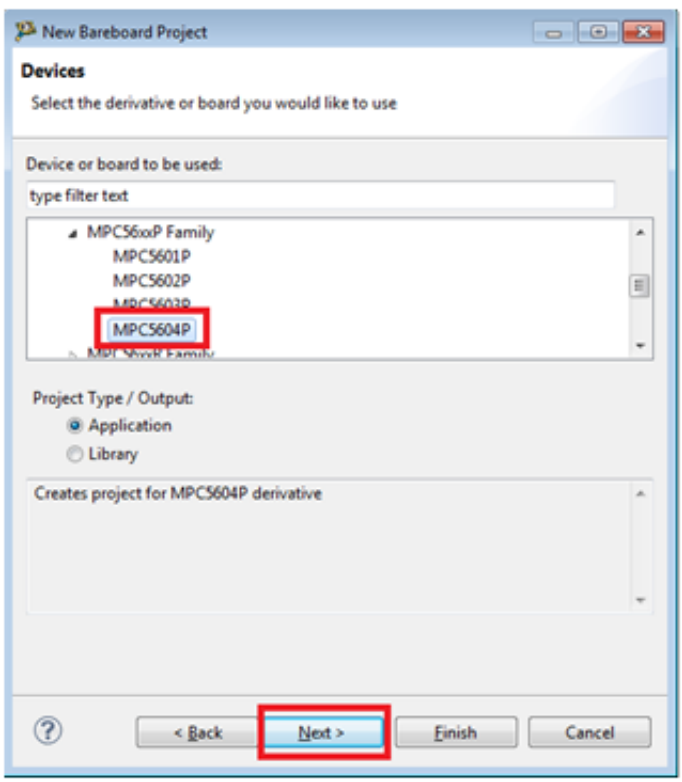

Figure 35. Select devices

4. Click *Next* and *Finish* to accept default options for Connections and Language options.

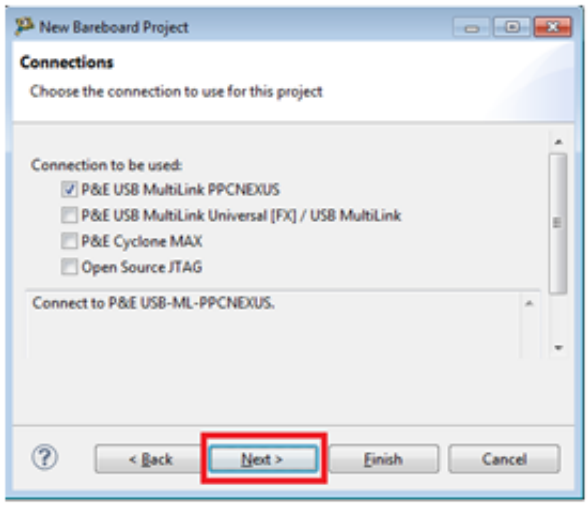

Figure 36. Select connections

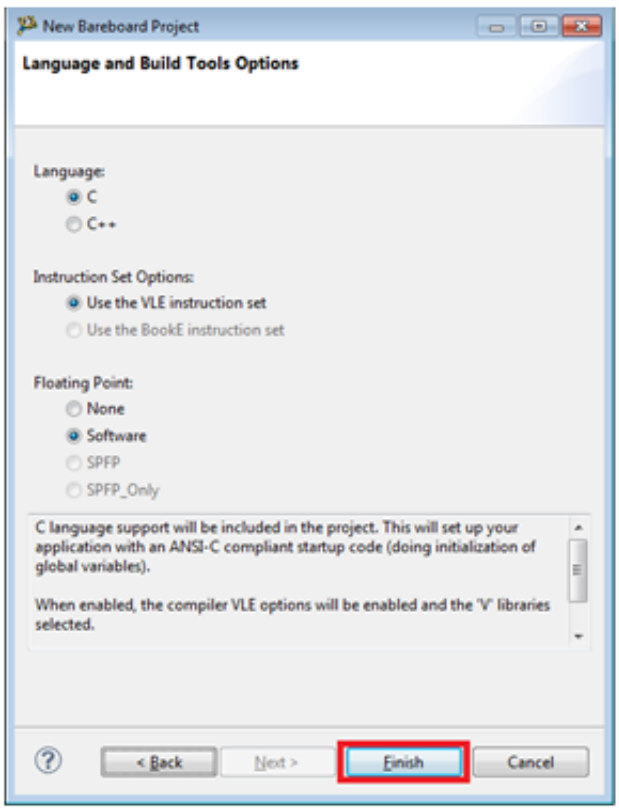

Figure 37. Select language and build tools options

By default, CodeWarrior sets the option to create build for RAM. Since your example project is for Flash, change the CodeWarrior build option to FLASH as shown below.

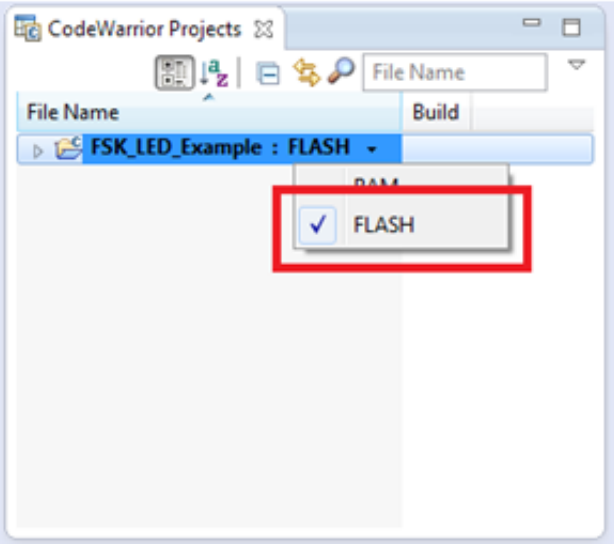

Figure 38. Select FLASH build option

4.2 Add RAppID code to CodeWarrior project

CodeWarrior creates a new bare board project with CodeWarrior generated code and linker file included in the project. Since the code in the example project is generated using RAppID, you need to remove all the CodeWarrior generated code. Next, you need to add the RAppID generated code, the linker file along with the FreeMASTER, and the driver code to the CodeWarrior project.

You can accomplish this using the CodeWarrior project maker utility – *rsp2cw10*.

1. Select *Project* > *Close Project* to close the FSK_LED_Example project.
2. Run the *rsp2cw10* utility. This can be done by selecting the executable from the directory where this utility is installed or by using RAppID menu *External Tools->CWInterface*.

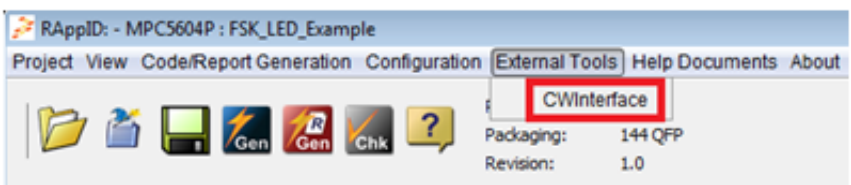

Figure 39. Select External tools option

The Code Warrior Project Maker utility expects three inputs:

- The path where RAppID generated source code resides. All the code that exists in this folder will be copied to the CodeWarrior project.
- The path where CodeWarrior project resides
- The path of RAppID generated linker file to be included in the CodeWarrior project.

3. Enter the required values as shown below and select *OK*.

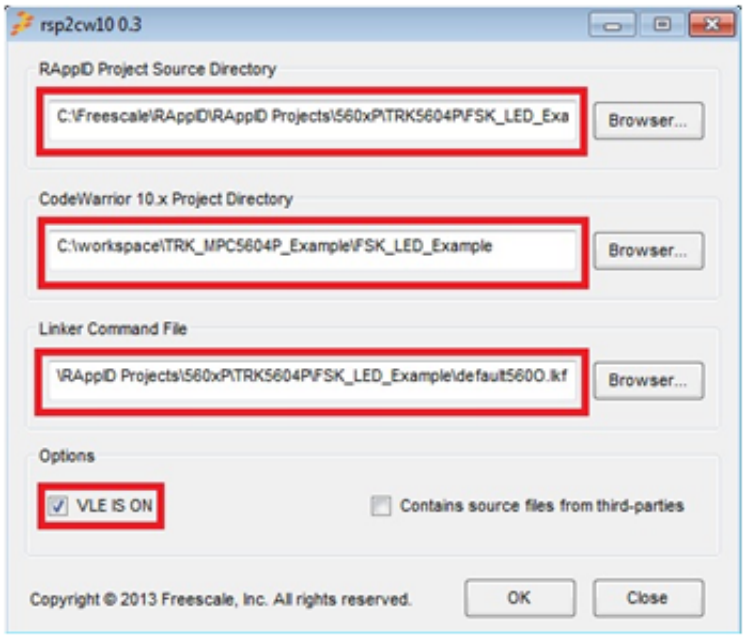

Figure 40. CodeWarrior Project Maker

4. Once the tool is executed, all the RAppID generated source files, driver code and FreeMASTER code is added to the CodeWarrior project.
5. Open the CodeWarrior project that was previously closed and verify that the project now includes all the required source files as shown below.

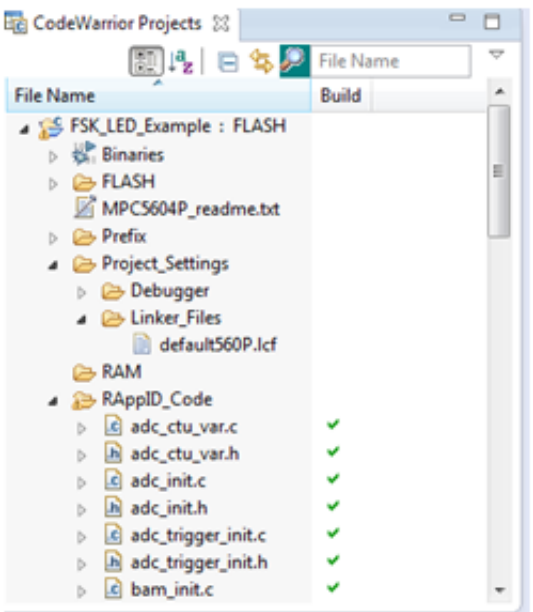

Figure 41. CodeWarrior Projects view

4.3 Add application code to auto generated code

4.3.1 Adding code to main.c

The main.c generated by RAppID initializes peripherals and includes an empty while loop. To add code to main.c:

1. Initialize SBC to enable CAN
2. Add code to support FreeMASTER utility
3. Process ADC – Turn on/off LED2 based on Potentiometer ADC inputs
4. Process CAN messages: Turn on/off LED 4 based on CAN message received and transmit appropriate response.
5. Process GPIO: Turn on/off LED 3 based on status of input switch S3

The complete c-code listing for main.c is shown below.

```
Example 1. Listing of main.c
/*******************  Dependent Include files here **********************/

##include "rappid_ref.h"
#include "rappid_utils.h"
#include "sys_init.h"
#include "freemaster.h"
#include "adc_drv.h"
#include "sbc_hld.h"
#include "gpio_drv.h"
#include "CANapi.h"
#include "pot_hld.h"

/**********************  Function Prototype here ************************/

void main(void);
void ProcessCAN(void);
void ProcessADC(void);
void ProcessGPIO(void);

/**********************  Global variables here ************************/
/* CAN messages to transmit */
unsigned char msgOKCAN[8] = {1,1,0,0,0,0,0,0};
unsigned char msgErrorCAN[8] = {1,0xFF,0,0,0,0,0,0};

uint16_t potValue;/* Potentiometer ADC input value*/
uint8_t switchState;/* State of switch S3 input */


/*********************  Initialization Function(s) ************************/

void main(void)
{

/* ------------------------------------------------------------ */
/*             System Initialization Function                   */
/* ------------------------------------------------------------ */
   sys_init_fnc();

   /* Initialize SBC */
   SBC_Init_DBG();

   /* FreeMASTER internal variables initialization */
   FMSTR_Init();

   /* Initialize CAN filter */
   SetCanRxFilter(1, 0, 0);

   /* Turn off LEDs */
   GPIO_SetState(52, 1);
   GPIO_SetState(53, 1);
   GPIO_SetState(54, 1);
   GPIO_SetState(55, 1);

/********* Enable External Interrupt *********/
   EnableExternalInterrupts();
```

```
   while(1)
    {
FMSTR_Poll();
  ProcessCAN();
  ProcessADC();
ProcessGPIO();
    }

}


/*****************************************************************************
*   Function: ProcessCAN
*
*   Description: Process CAN messages
*
*****************************************************************************/
void ProcessCAN(void)
{
can_msg_struct msgCanRX;

if (CanRxMbFull(0) == 1)/* Check if CAN message received */
{
    msgCanRX = CanRxMsg(0);
    if (msgCanRX.data[0] == 0)/* If first data byte is 0, turn off LED4 and send positive
response */
    {
    GPIO_SetState(55, 1);
    CanTxMsg (2, 1, 8, (uint8_t *)msgOKCAN, 0);
    }
    else if (msgCanRX.data[0] == 1)/* If first data byte is 1, turn on LED4 and send
positive response */
    {
    GPIO_SetState(55, 0);
    CanTxMsg (2, 1, 8, (uint8_t *)msgOKCAN, 0);
    }
    else/* If first data byte is not 0 or 1, send a negative response */
    {
    CanTxMsg (2, 1, 8, (uint8_t *)msgErrorCAN, 0);
    }
}
}



/*****************************************************************************
*   Function: ProcessADC
*
*   Description: Processes ADC input
*
*****************************************************************************/

void ProcessADC(void)
{
potValue = Pot_Get_Value();
if(potValue <= 512) /* If Potentiometer input is <= 512 turn on LED2, otherwise turn off
LED2 */
{
GPIO_SetState(53, 0);
}
else
{
GPIO_SetState(53, 1);
}
}
/*****************************************************************************
*   Function: ProcessGPIO
*
*   Description: Processes switch input
*
```

```
*****************************************************************************/
void ProcessGPIO(void)
{
switchState = GPIO_GetState(50); /* Check switch S3 state */
if (switchState)/* If Switch S3 is pressed, turn on LED3, otherwise turn off LED3*/
{
GPIO_SetState(54, 1);
}
else
{
GPIO_SetState(54, 0);
}
}
```

4.4 Build code

After completing the code modifications described in the previous section, you are ready to build the example project.

To build the project, select *Build* menu in the Commander window. This will compile and build the project resulting in *.elf* file that contains debug symbols and *.mot* file which is s-record file.

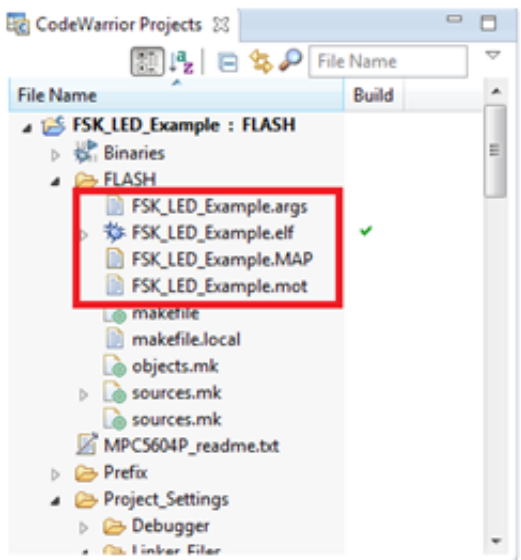

Figure 42. CodeWarrior build files

5 Flash code

Next you need to Flash the code using the RAppID Boot loader utility on to the target board. However, before flashing the code, ensure that the board is powered correctly and the jumpers are in the correct position.

5.1 Jumper settings and power connections

The TRK-MPC5604P board can be powered by external 12V supply or using USB connector. When the board is powered by USB connector, SBC is not powered. Whereas, the SBC is powered when the board is powered using the external 12V supply. Since, you are using CAN in this example, you need to power up SBC. Connect the J1 jumper across SBC_5V as

shown below and connect external power to JP1. Using the USB cable provided with Fast Start Kit, connect the board to your computer. This provides a virtual serial port connection. You can confirm this by checking the available COM port using the Windows Device manager.

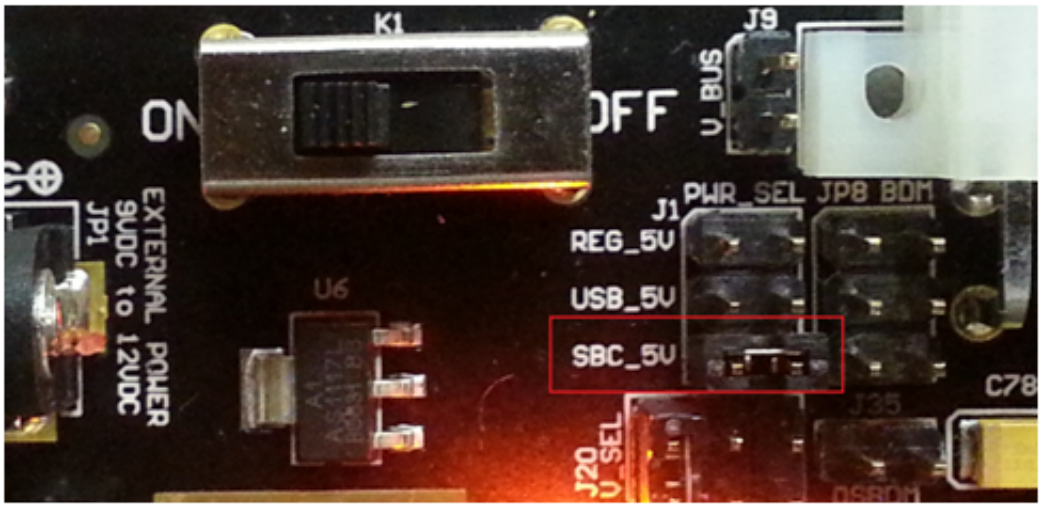

Figure 43. Connect the J1 jumper across SBC_5V

To enable the UART communication, connect jumper J7 (TXD_EN) and J8 (RXD_EN) to position 1-2 as shown below.

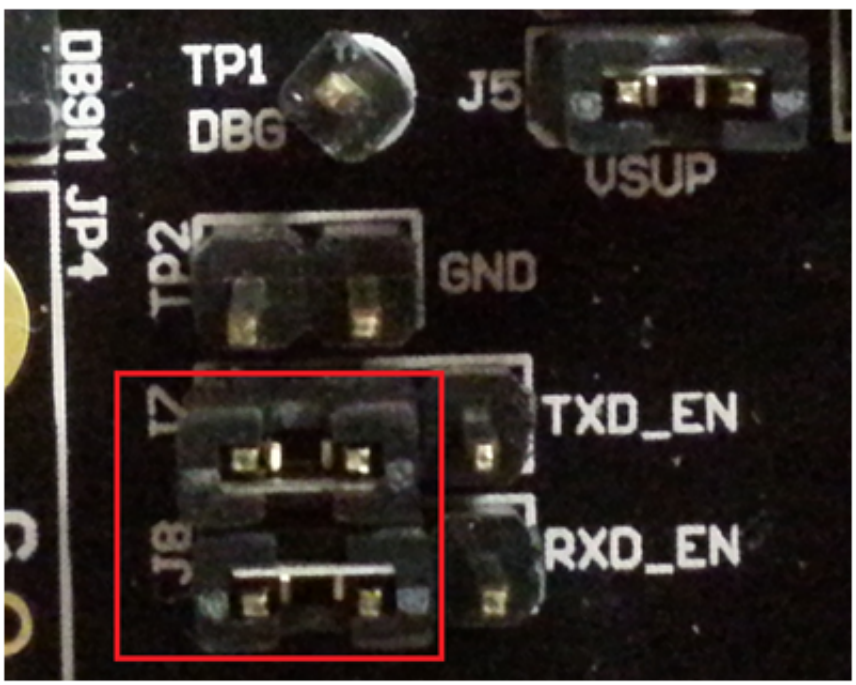

Figure 44. Connect UART jumpers

To enable CAN communication, connect J6 jumpers as shown below.

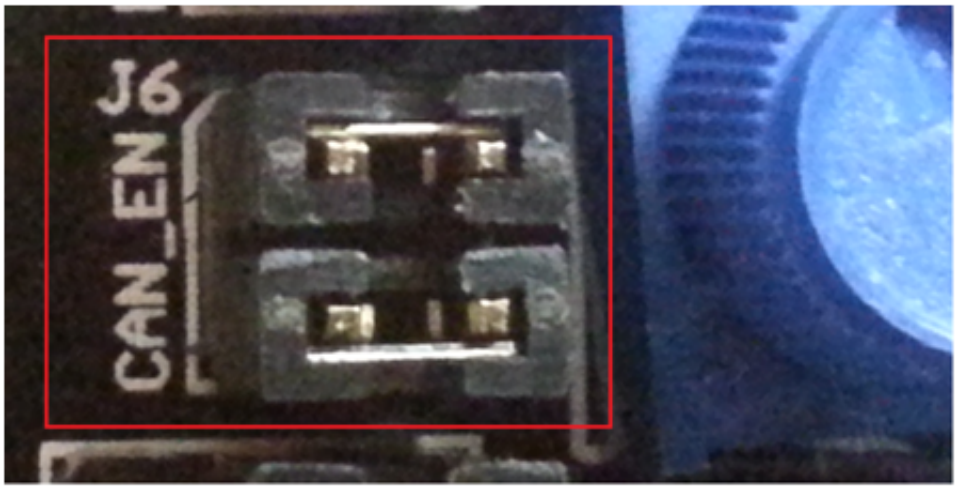

Figure 45. Connect CAN jumpers

To enable LEDs, connect all four jumpers in J27 as shown below. To enable input switches, connect all four jumpers in J26 as shown below.

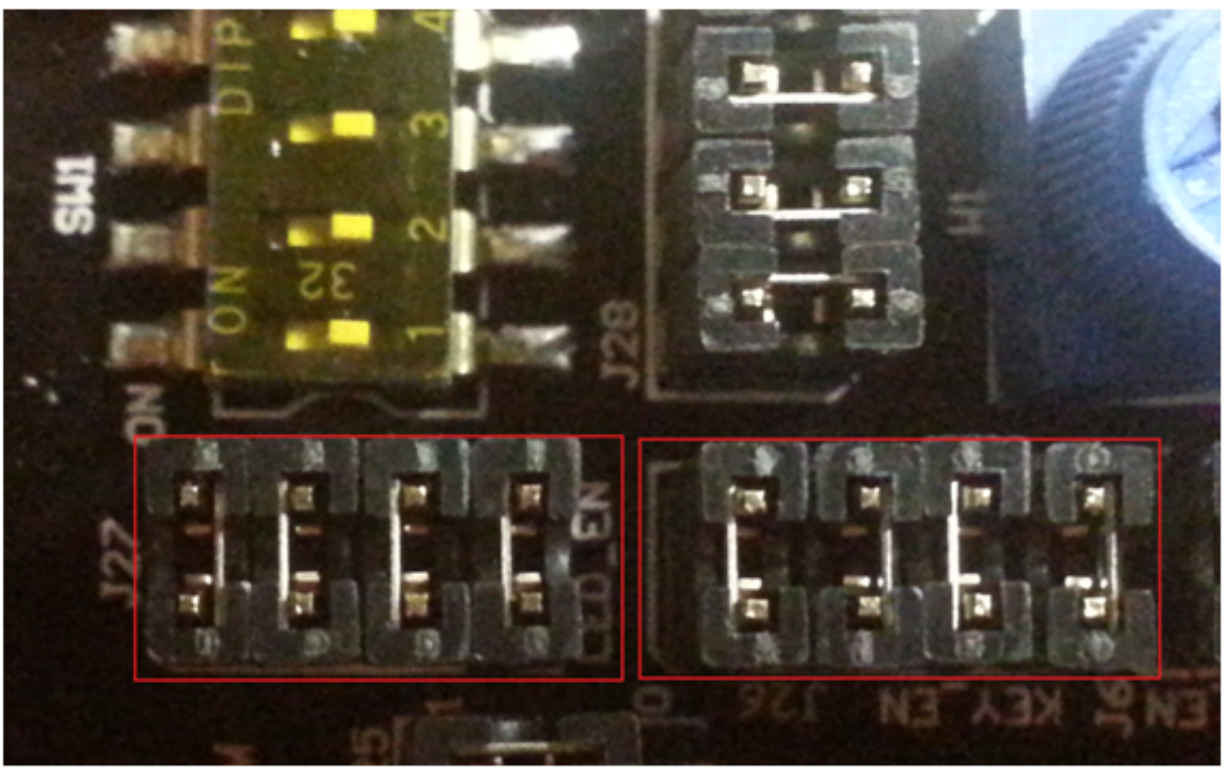

Figure 46. Connect LED and input switch jumpers

5.2 Flash code using Boot Loader utility

To flash code:

1. Set the jumper of J17 to position 1-2 which pulls FAB high and jumper J18 and J19 to position 2-3 to set ABS0 and ABS2 low as shown below.

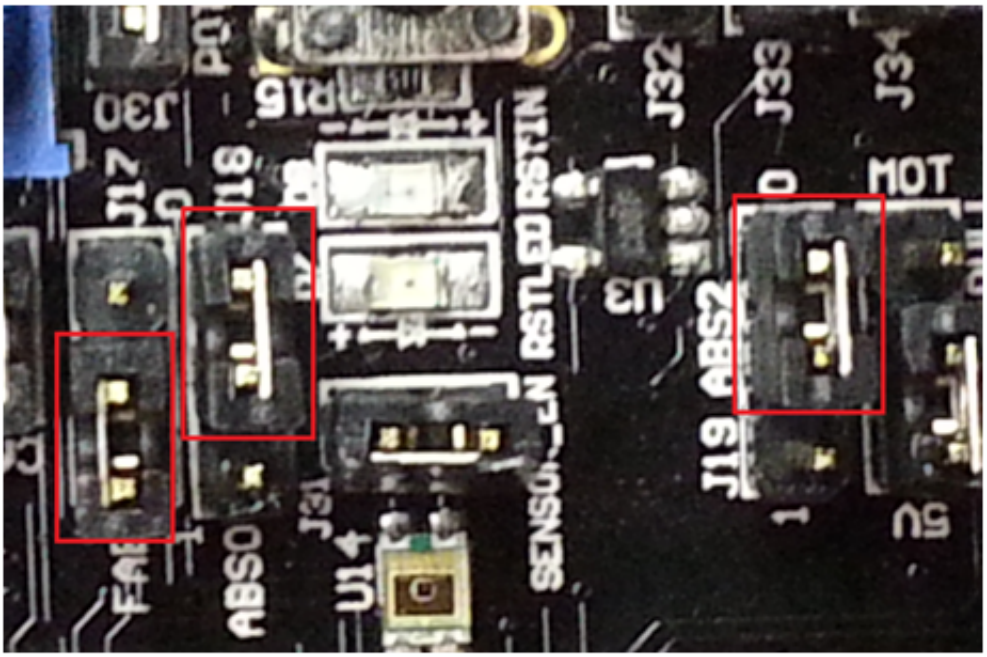

Figure 47. Jumper setting for Flash

2. Use the RAppID Boot loader utility to flash S-record file FSK_LED_Example.mot using serial port.
3. Launch RAppID Boot loader utility from Windows Start menu.
4. Select the values as shown in the RAppID Boot loader utility application window below. The COM channel may be different in your setup.

NOTE

If boot loader displays error about wrong password, try 0xFFFFFFFFFFFFFFFF instead of default password option. If your board has microcontroller with previous mask set, the default password will not work.

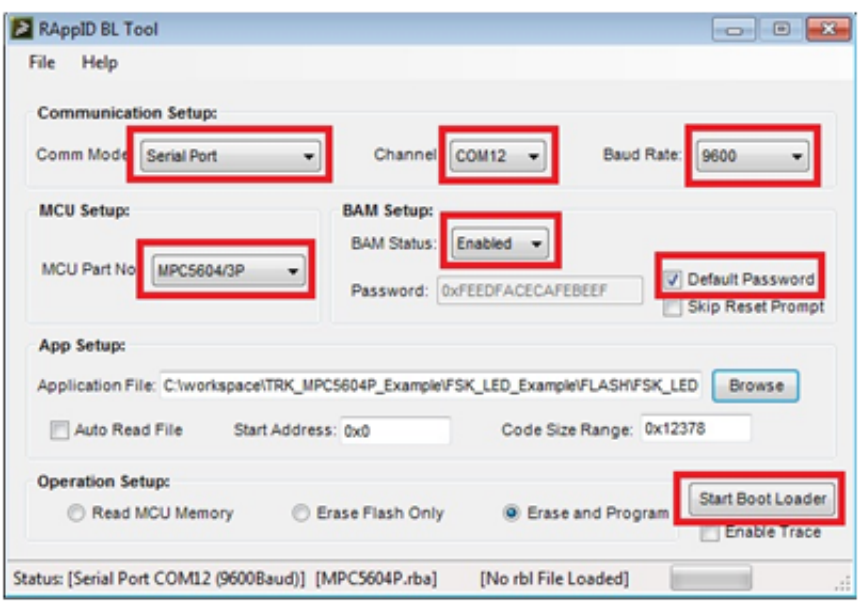

Figure 48. RAppID Boot loader application window

5. Click the Start Boot Loader button to start the flash process. When asked to cycle power to MCU, press the reset button on the board. Flashing process should start.
6. After Flash is complete, the code should begin executing.

7. To have the code execute after a power cycle, move the jumper J17 to position 2-3 to pull FAB low as shown in the figure below.
8. Turn the power off to the module and re-apply the power. The code should be running now.

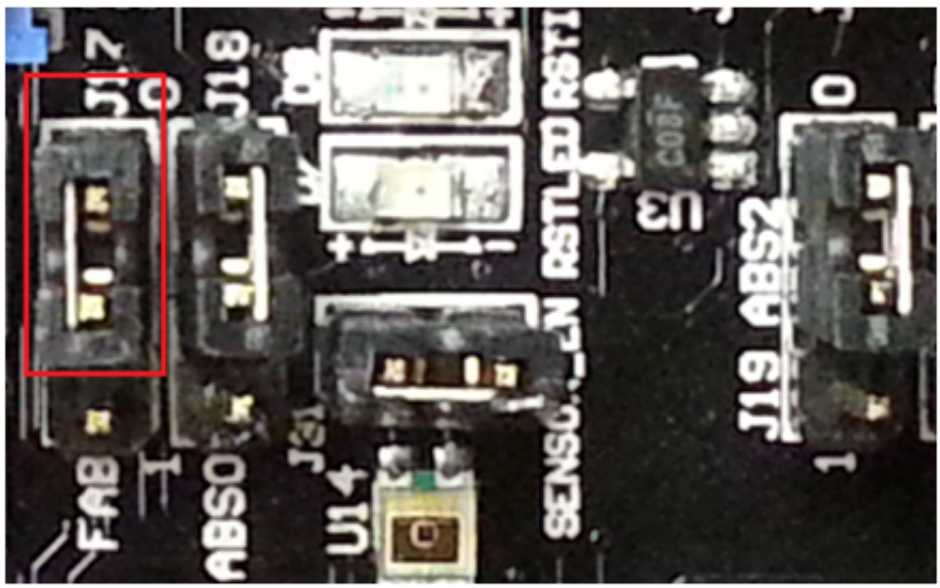

Figure 49. Jumper setting after Flash

6 Test code

6.1 Monitoring variables using FreeMASTER

Use FreeMASTER to monitor global variables used in this example project.

1. Launch FreeMASTER utility from Windows Start menu.
2. After the application is started, select *Project* > *Options* > *Communication* from menu and set the communication port number and baud rate of 115200 as shown below.

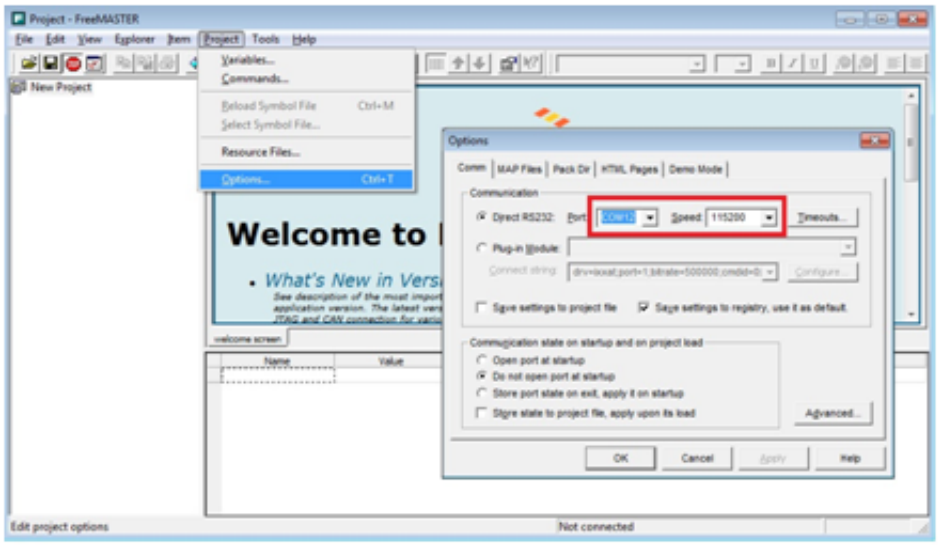

Figure 50. FreeMASTER communication options

3. Select the example application MAP file. In this example, the MAP file is the *.elf* file generated during the build process. FreeMASTER uses the information about the variables, their names, types, and addresses contained in the *.elf* file.
4. From the MAP tab, select the MAP file as shown below.

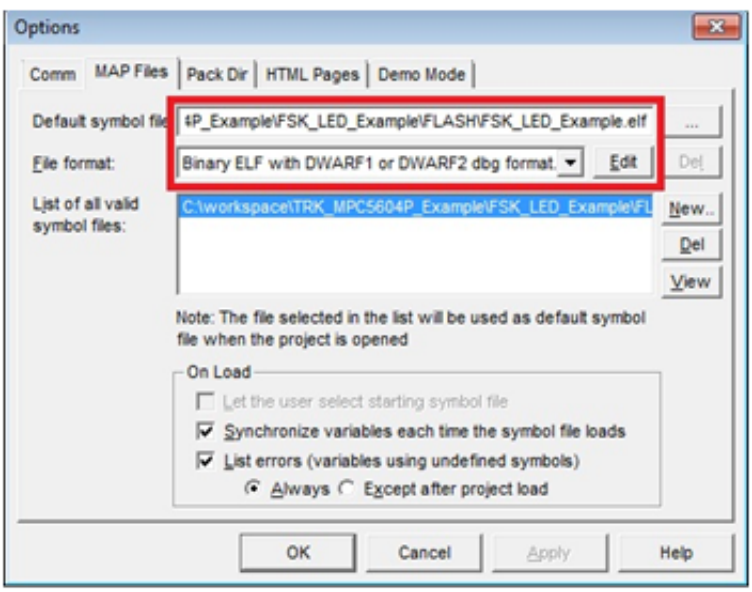

Figure 51. Select MAP file

5. Add the two global variables in this example project to monitor in the watch window – *switchState* and *potValue*.
6. To select variable to watch:
   - Right-click on the variable grid.
   - Select *Create New Watched Var...*from the menu. This will pop up a variable selection window.
   - Select *potValue* from the drop down as shown below and select OK. This will add the variable *potValue* to watch window.
   - Using similar steps, add *switchState* to watch window.

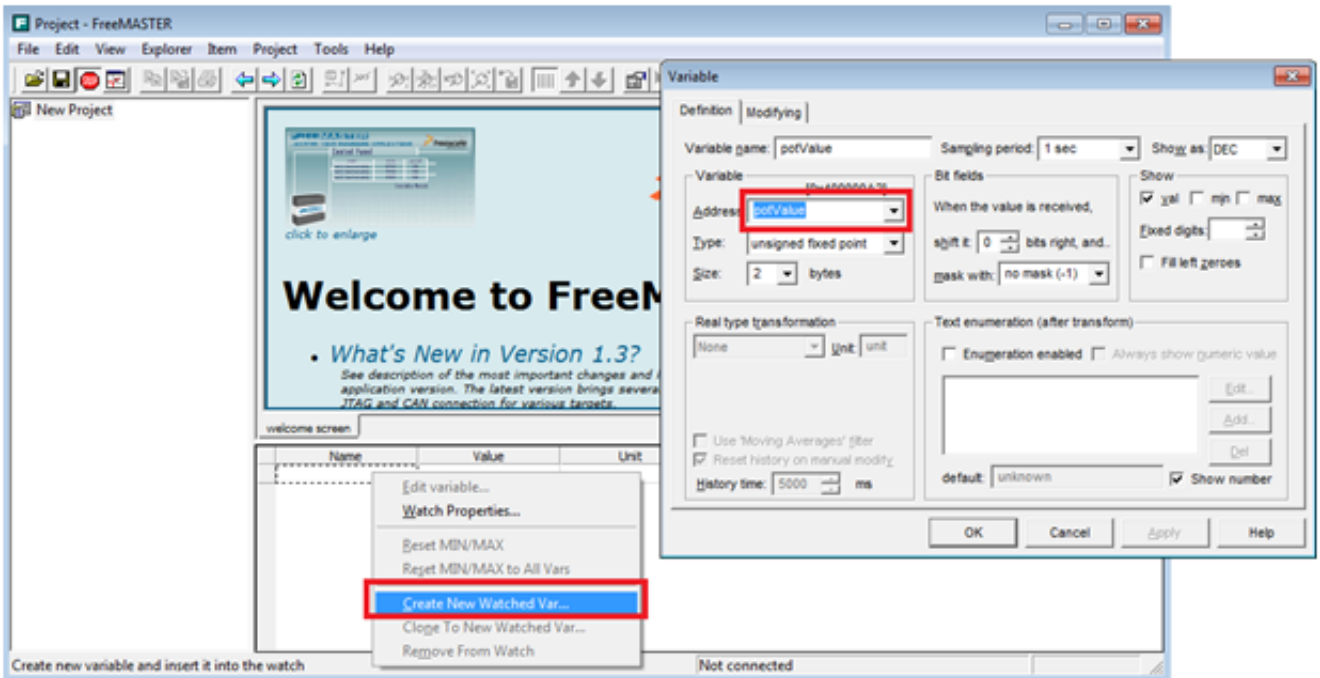

Figure 52. Add variable to monitor

7. Select the icon *Start/Stop communication* to start communication and observe the two watch variables getting updated.

6.2 Testing ADC and switch input

When you rotate the potentiometer, the watch window should update. LED2 should turn on when the *potValue* is below 512 counts and it should turn off when the count exceeds 512.

The *switchState* should change when the input button switch S3 on the board is pressed or released. LED3 should turn on when input switch S3is pressed and it should turn off when switch S3 is released..

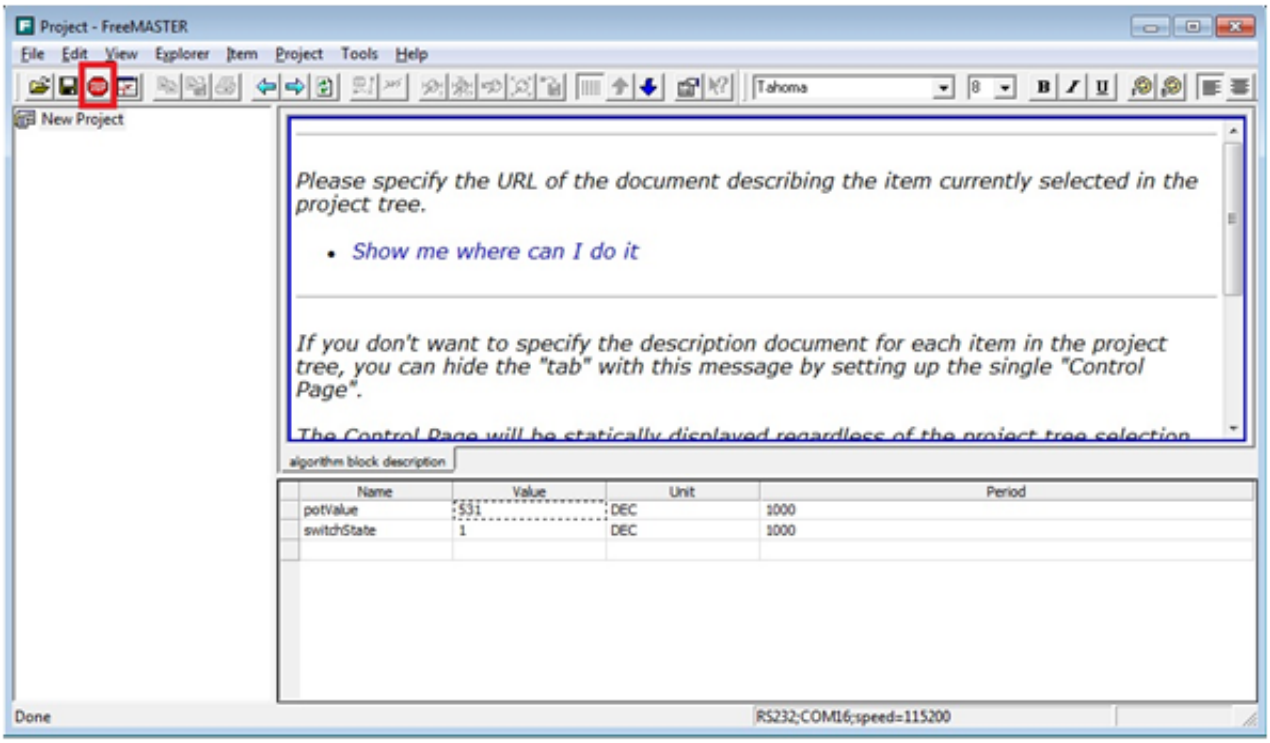

Figure 53. FreeMASTER main window

6.3 Testing CAN communication

You can use a CAN communication tool like CANalyzer or IXXAT MiniMon for sending and receiving CAN messages. Send a CAN message using a value of 1 in first byte to turn on LED4 and a value of 0 to turn off LED4. In both cases, you should see a positive CAN message transmitted by the microcontroller. When you send a CAN message with first byte other than 0 or 1, you should see a negative message being transmitted as shown in the figure below.

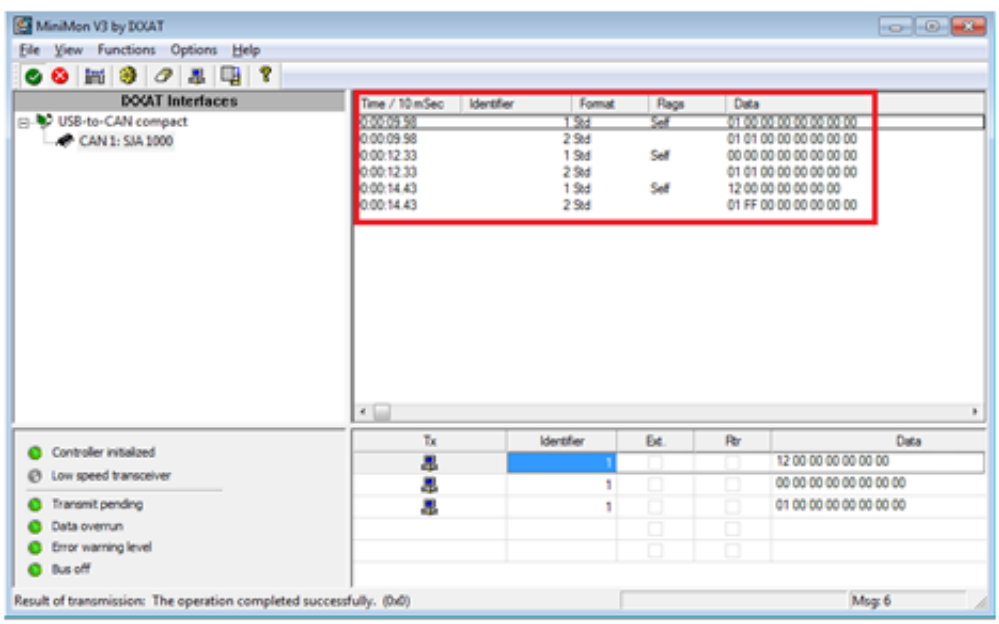

Figure 54. Testing CAN communication

6.4 Testing PIT interrupt

PIT interrupt should occur every 250 ms and LED1 should toggle at this rate. Connect PD4 pin on the TRK-5606B board to a scope to verify the LED1 output toggle rate. The figure below shows the scope trace of the LED1 output toggling at 250 ms rate.

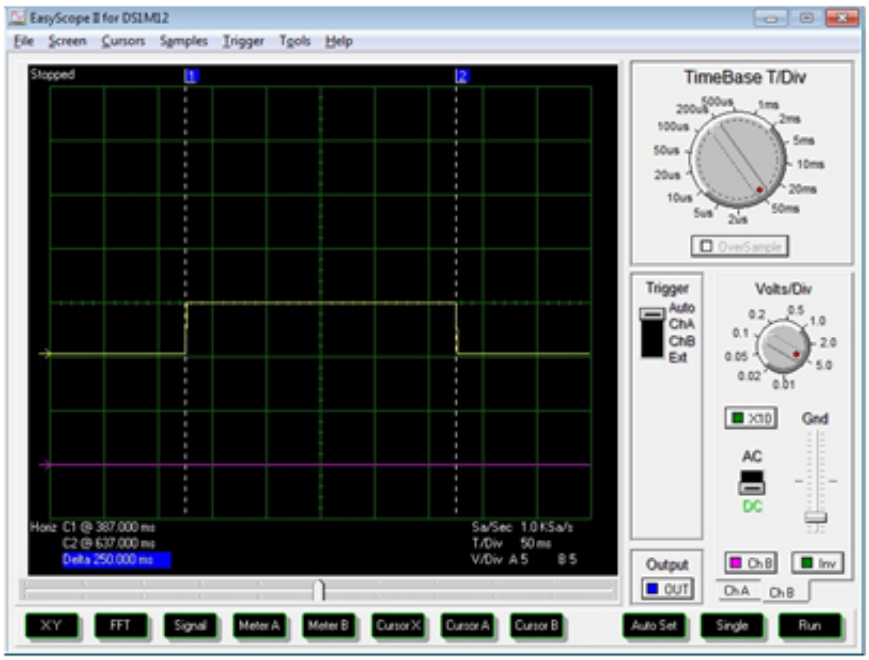

Figure 55. LED1 output

7 Conclusion

The Qorivva Fast Start Kit for TRK-MPC5604P contains all the hardware and software tools with comprehensive documentation and examples to help developers get quickly started on application development for the Freescale MPC5604P microcontroller. This application note provides an overview of all the tools provided with the Fast Start Kit and an example providing step by step instructions which will help users better understand the process of application development using Freescale microcontrollers and development tools.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN4894
Revision 0, Mar 2014