TRK-MPC5604P Fast Start kit Training

March 2014

TRK-MPC5604P Fast Start Kit hardware contents

- StarterTRAK TRK-MPC5604P Evaluation Board
- USB cable
- Installation media
- Instruction pamphlet

TRK-MPC5604P Fast Start Kit software contents

- The TRK-MPC5604P Fast Start Kit contains Single installer that installs all software tools, documents and example projects:

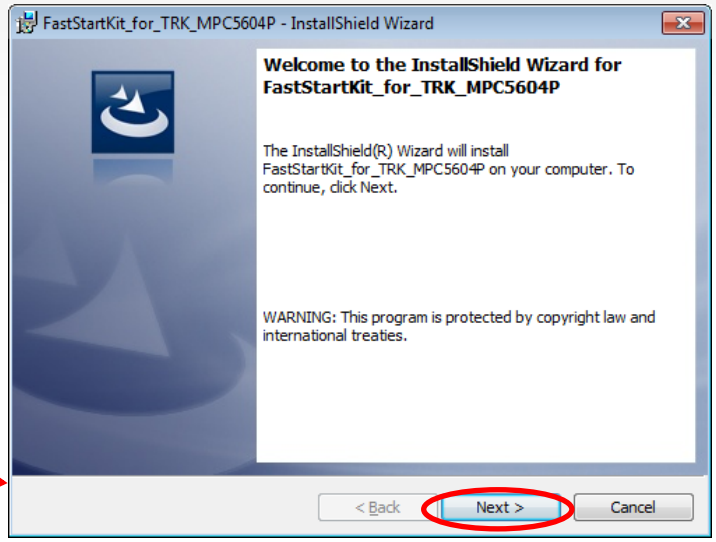
1. RAppID init and RAppID pin wizard for MPC560xP with permanent license
2. Utility to add RAppID generated code to CodeWarrior 10.5 Project
3. CodeWarrior for MPC56xx 10.5 SE
4. Low Level Drivers (LLDs) for MPC5604P
5. High Level Drivers (HLDs) for StarterTRAK TRK-MPC5604P
6. RAppID Bootloader utility
7. FreeMASTER utility
8. Simple LED application examples that demonstrates the usage of software tools, LLDs and HLDs.

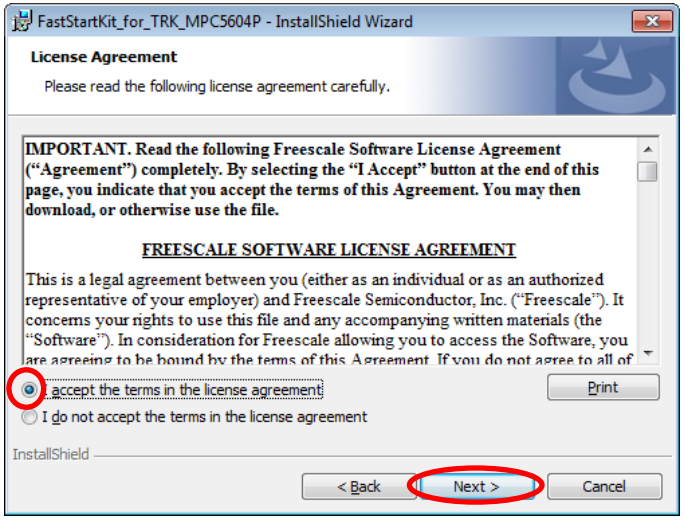
Installing Software tools

Start installation process by executing ***setup.exe*** located in the installation media.

Click on *Next*

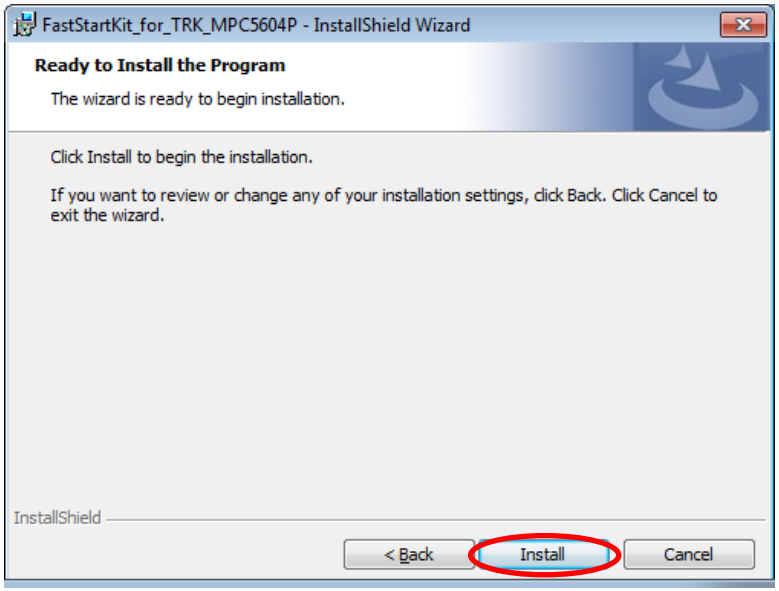
Accept the License agreement and click on *Next* to start installation of CodeWarrior 10.5





Installing Software tools

Select *Install to begin installation of Fast Start Kit tools.*

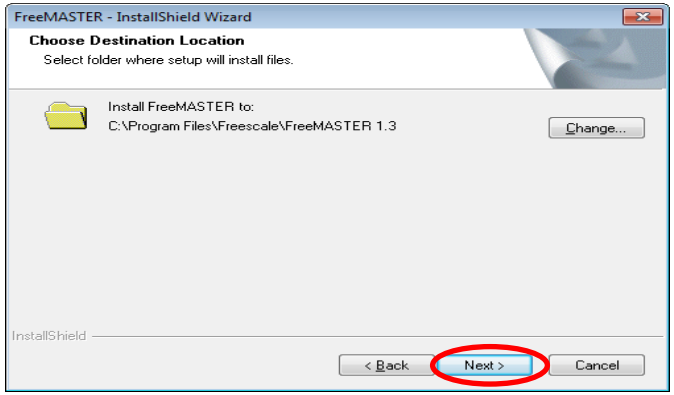


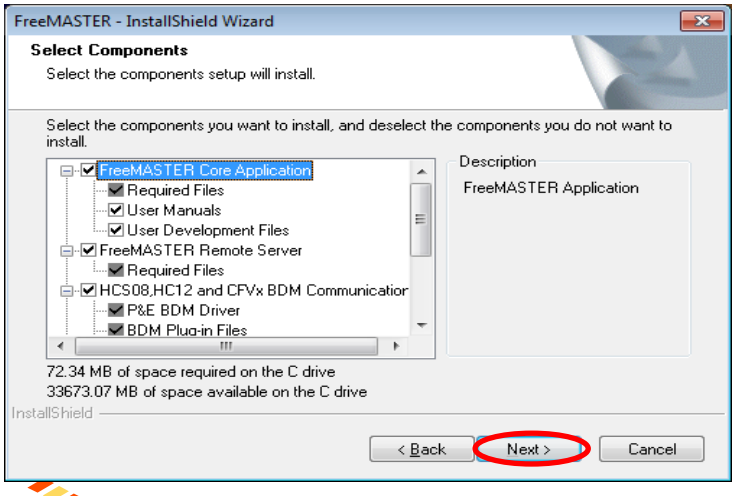
This will launch FreeMASTER installer.

Installing Software tools – FreeMASTER

Accept license agreement and default installation directory by clicking on *Next*.

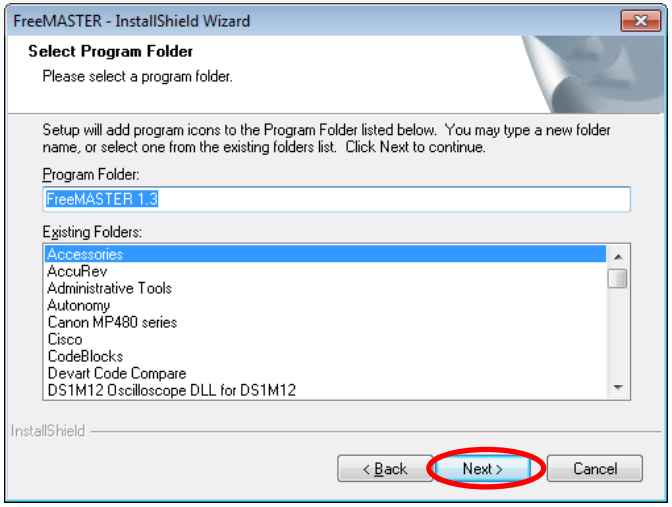
Accept default options in the “Select Components” window by selecting *Next*.



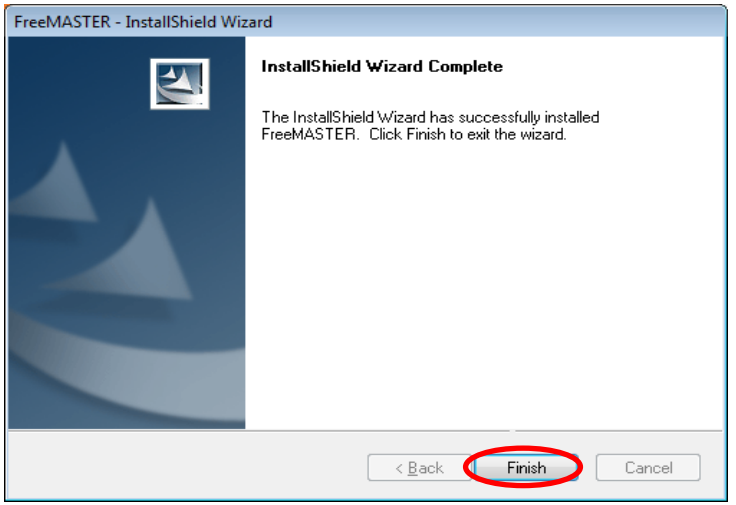


Installing Software tools – FreeMASTER

Accept default Program Folder by clicking on *Next*.

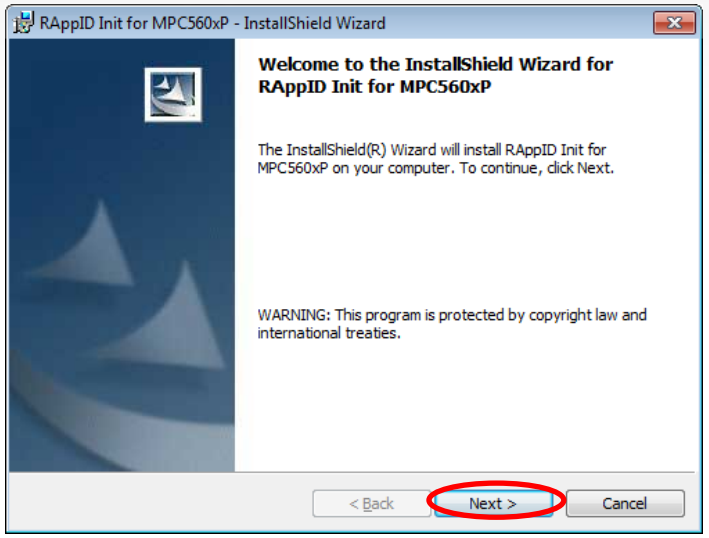


Select *Finish to complete FreeMASTER installation*. This will launch RAppID Init installer.

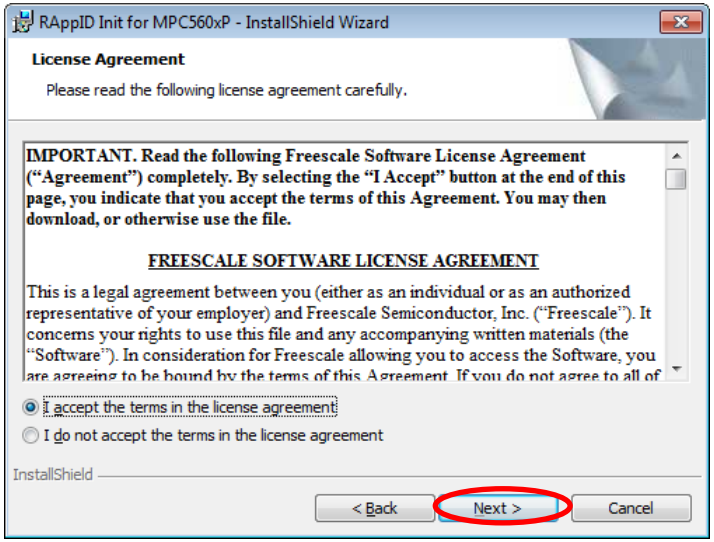


Installing Software tools – RAppID Init

Start RAppID Init installation by clicking on *Next* button

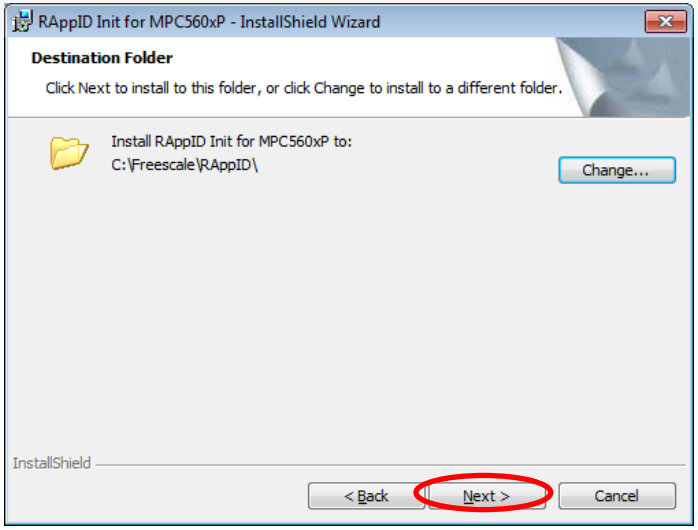


Accept license agreement and default directory location by clicking on *Next*

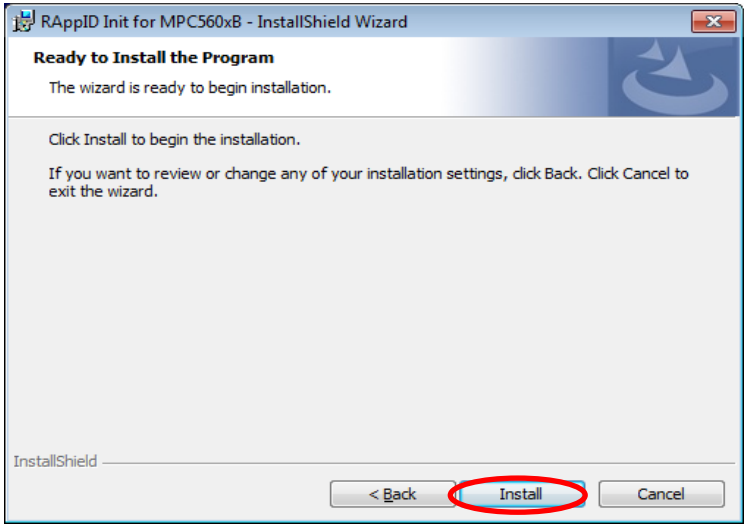


Installing Software tools – RAppID Init

Accept default location for destination folder by clicking on *Next*

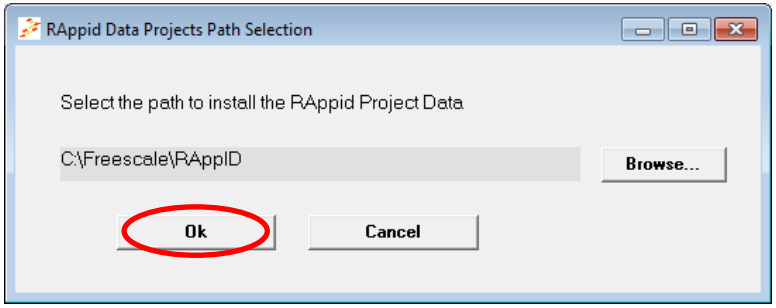


Click On *Install* button

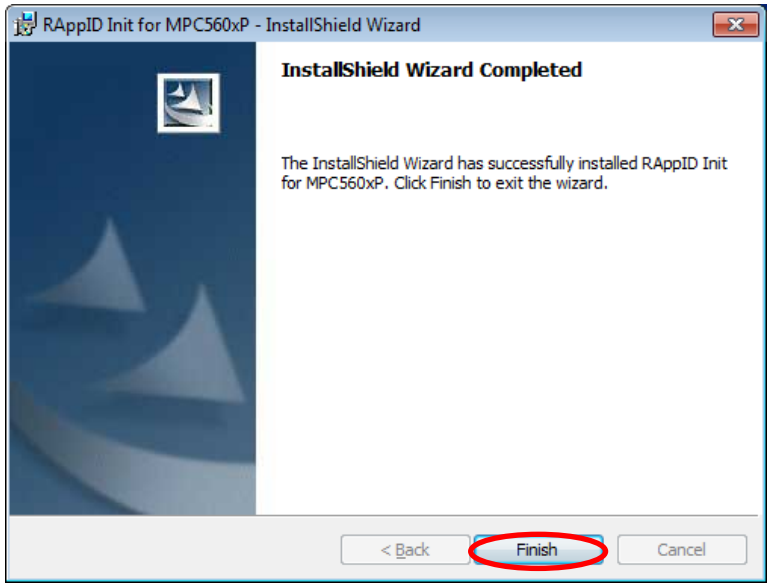


Installing Software tools – RAppID Init

Accept default location for RAppID project data folder by clicking on *Ok*



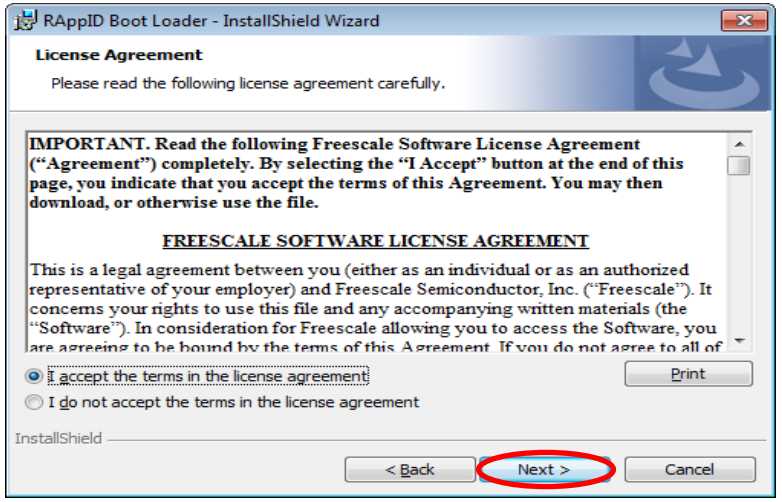
Click On *Finish* button to complete RAppID Init installation. This will launch RAppID Boot loader installer.



Installing Software tools – RAppID Boot loader

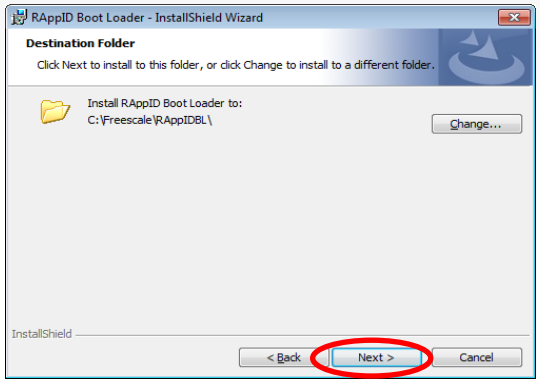
Click on *Next* to start RAppID boot loader installation.

Accept license agreement and click on Next

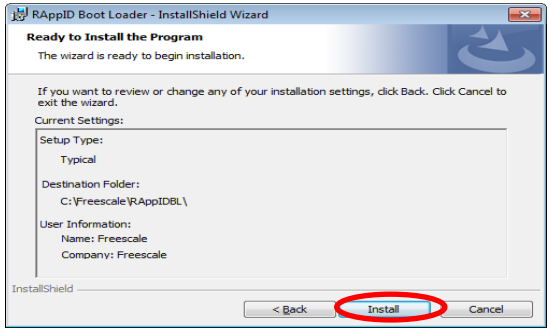


Installing Software tools – RAppID Boot loader

Accept default destination folder by accepting by clicking on *Next*.

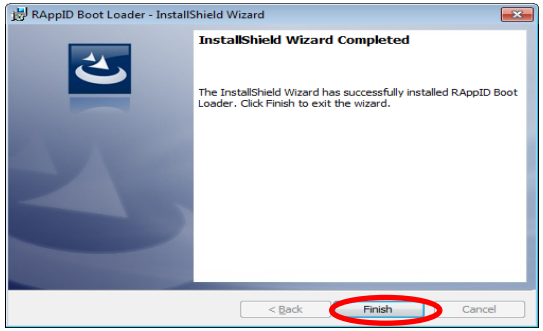


Start installation by selecting Install.



After installation is complete, select Finish to complete installation.

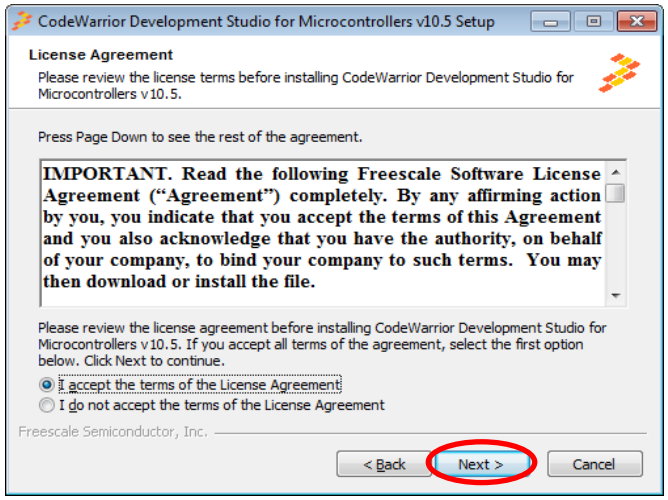
This will start CodeWarrior installation.

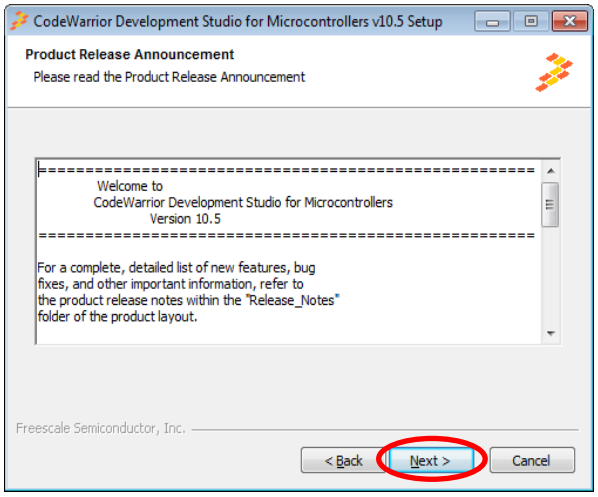


Installing Software tools - CodeWarrior

Start CodeWarrior installation by clicking on *Next*

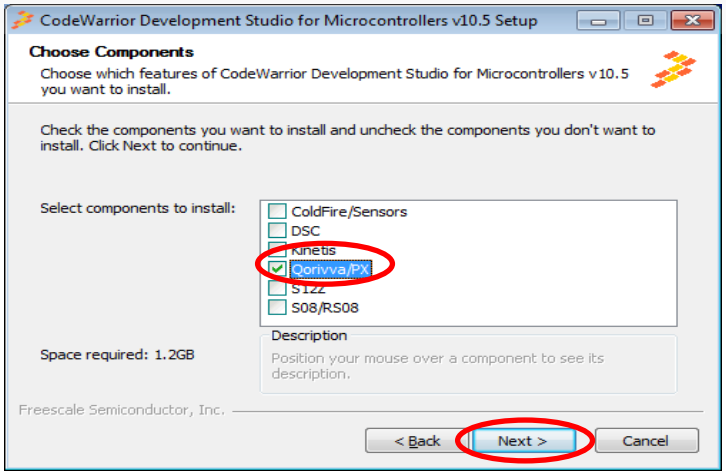
Accept License agreement and Product Release announcement by clicking on *Next*



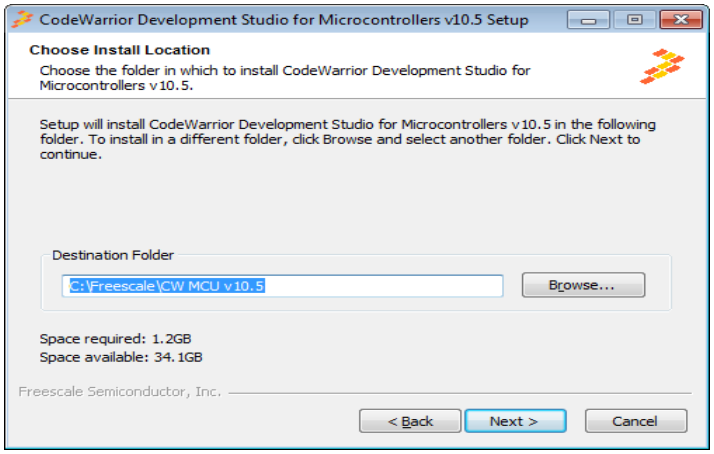


Installing Software tools - CodeWarrior

Choose Qorriva component and click on *Next*

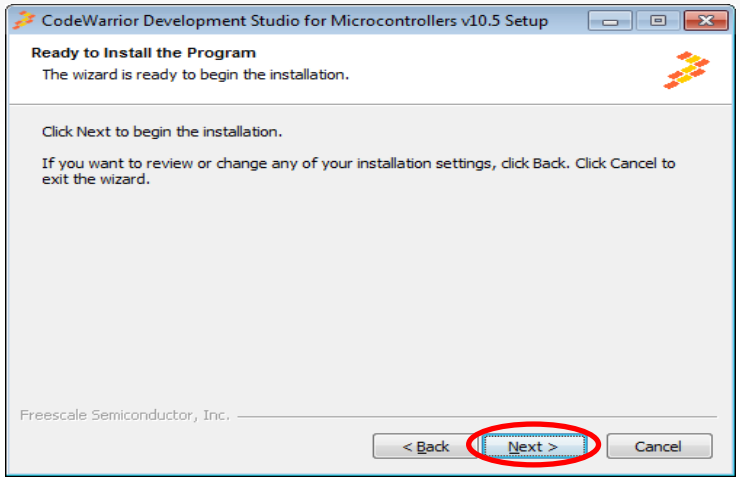


Accept the default install location by selecting *Next*

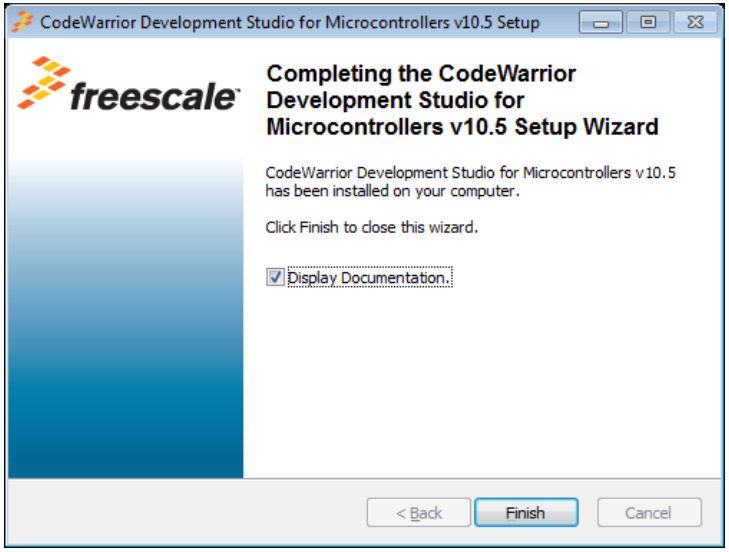


Installing Software tools - CodeWarrior

Select Next to begin CodeWarrior installation.



After installation is complete, select Finish to complete installation.



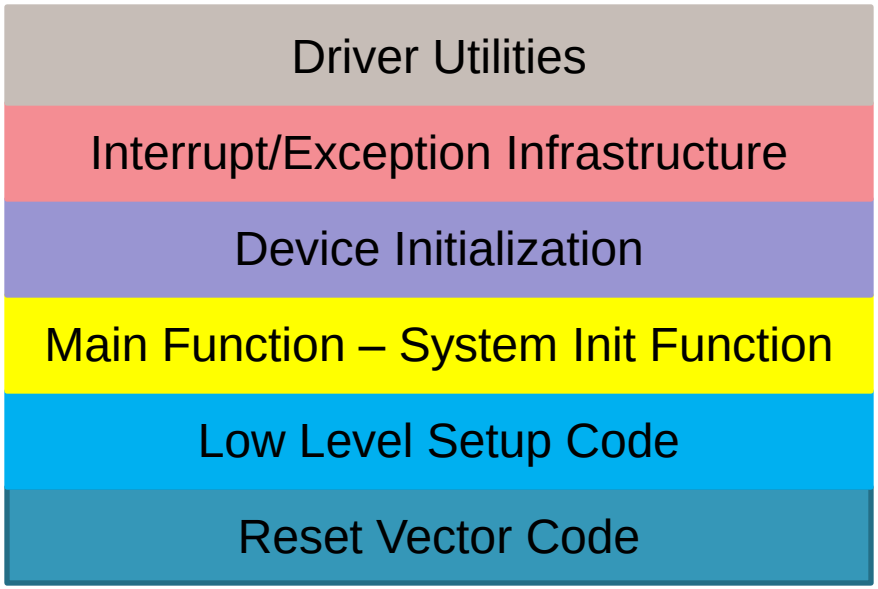
RAppID Init Overview

- 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 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

RAppID Init Overview

- 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

RAppID Generates What?



Low Functionality Drivers

Interrupt Vector Table, Handler, ISR functions

Device and Peripheral Initialization Code

Example Main, Init Sequence Function…

From Reset to Main, crt0, Stack,…

RCHW, Section Map…

CodeWarrior project maker utility

- The Fast Start Kit provides a utility to assist in adding RAppID generated code to a CodeWarrior project
- After creating a empty CodeWarrior project for the required microcontroller, the user can invoke the utility *rsp2cw10.exe* which will add RAppID generated code and sets up the CodeWarrior project by adding all the CodeWarrior setup variables to enable clean build.

Low Level Driver code

- In the installation disk, the following low level driver code is provided
  - GPIO
  - ADC
  - UART
  - CAN

GPIO Low Level Driver code

- **uint8_t GPIO_GetState (uint16_t ch)**
  - This function returns the state of requested GPIO pin

- **void GPIO_SetState (uint16_t ch, uint8_t value)**
  - This function sets the GPIO pin to the specified state

ADC Low Level Driver code

- **uint16_t A2D_GetSingleCh_ADC0 (uint32_t ch)**
  - This function sets up, starts, and returns a conversion for a single ADC0 channel
- **uint16_t A2D_GetSingleCh_ADC1 (uint32_t ch)**
  - This function Sets up, starts, and returns a conversion for a single ADC1 channel
- **uint16_t A2D_GetChResult_ADC0(uint32_t ch)**
  - This function returns the result for a single ADC0 channel
- **uint16_t A2D_GetChResult_ADC1(uint32_t ch)**
  - This function returns the result for a single ADC1 channel
- **void A2D_SetupCh_ADC0(uint32_t ch)**
  - This function sets up channel for the ADC0 conversion
- **void A2D_SetupCh_ADC1(uint32_t ch)**
  - This function sets up channel for the ADC1 conversion

UART Low Level Driver code

- **void UartTxMsg(uint8_t *u8TxData, uint32_t u32Size)**
  - This function transmits a message in buffer *u8TxData* of size *u32Size*
- **uint8_t UartRxDataByte(void)**
  - This function returns data from UART buffer
- **uint32_t UartRxNewDataSize(void)**
  - This function checks how much new data there is in UART buffer
- **uint8_t UartRxBufEmpty(void)**
  - This function checks if the UART buffer is empty
- **void UartBufInit(void)**
  - This function initializes UART Buffer
- **void UartRxFillBuf(void)**
  - This function fills the UART Buffer from the UART RX peripheral

CAN Low Level Driver code

- **void SetCanRxFilter(uint32_t id, uint8_t mb, uint8_t ext)**
  - This function sets up the mailboxes on the specified CAN channel and works for standard and extended IDs.
- **void CanTxMsg (uint32_t id, uint8_t mb, uint8_t dlc, uint8_t data[], uint8_t ext)**
  - This function transmits a CAN message.
- **can_msg_struct CanRxMsg (uint8_t mb)**
  - This function receives a CAN message.
- **uint8_t CanRxMbFull (uint8_t mb)**
  - This function checks if CAN Mail box is full.
- **uint8_t CanTxMbEmpty (uint8_t mb)**
  - This function checks if CAN Mail box is empty.

High Level Driver code

- In the installation disk, the following high level driver code is provided
  - Potentiometer
  - Photo Sensor
  - SBC

Potentiometer High Level Driver code

- **uint16_t Pot_Get_Value(void)**
  - This function sets up, starts, and returns conversion value for Potentiometer channel (PE0, ADC1 Channel 5 of TRK-MPC5604P board).

Photo Sensor High Level Driver code

- **uint16_t Photo_Sensor_Get_Value (void)**
  - This function sets up, starts, and returns conversion value for Photo sensor channel (PE1, ADC0 Channel 4 of TRK-MPC5604P board).

SBC High Level Driver code

- **void SBC_Init_DBG(void)**
  - Sets SBC in TRK-MPC5604P board to enable CAN. It is assumed that SBC is in Debug mode and watchdog refresh is required only on initialization.

RAppID Bootloader utility

- The RAppID Boot Loader is a tool developed by Freescale to help with the development of software for Freescale MCUs by allowing the customer a method to update software of a MCU through a serial link using CCP.
- The RAppID Boot Loader works with the built in Boot Assist Module (BAM) included in the Freescale Qorivva & PX series 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.).

FreeMASTER utility

- FreeMASTER allows users to debug applications in true real-time through its ability to watch and modify variables.
- Remote control capability allows it to be used as a diagnostic tool for debugging customer applications remotely across a network.
- It is an outstanding tool for demonstrating algorithm or application execution and variable outputs.
- It provides monitoring/visualization of application variables in the same manner as a classical oscilloscope with a CRT.
- Simple RS232 native connection and other options possible on selected platforms (BDM, JTAG, CAN,...)
- Built-in support for standard variable types (integer, floating point, bit fields)

LED Example Using RAppID Init

- The next few slides will demonstrate an example project that describes steps to configure MPC5604P, generate, build, flash and test the code using various tools provided with TRK-MPC5604P Fast Start Kit
- In this example we will use RAppID Init tool to configure and generate initialization code for MPC5604P
- We will use the GPIO, ADC and CAN driver code supplied with the installation
- We will use CodeWarrior 10.5 to build the code
- We will use RAppID Boot loader utility to flash the code to the target
- The example turns on/off LEDs based on Switch input, Potentiometer input and CAN commands.

LED Example overview

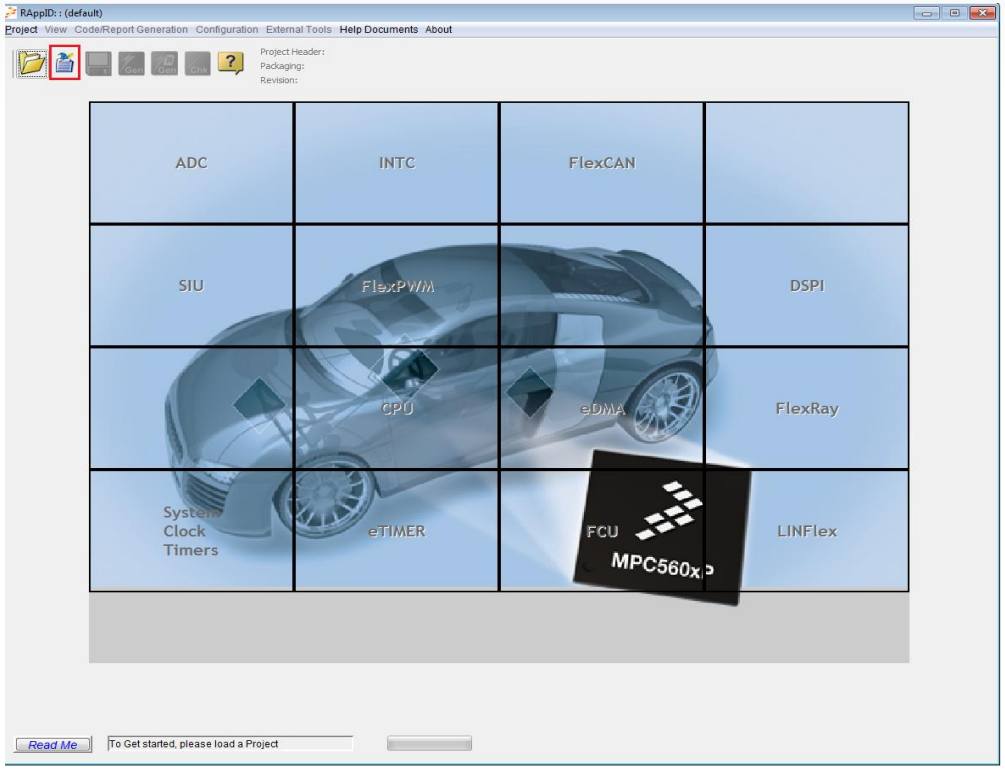
- LED1 turns on when switch S1 is in pressed state and turned off when S1 is in released state
- LED2 is turned On/Off based on potentiometer input
- LED3 is turned On/Off based on CAN command input
- PD10 is driven using FlexPWM signal and the duty signal can be increased by input switch S4 and decreased by input switch S3.
- PA0 is configured as eTimer input capture function which is used to calculate duty cycle of PWM output of PD4 by connecting output PD4 to input PA0.

Creating LED example using RAppID init tool

- The next few slides will demonstrate how to create RAppID project to configure all the pins and peripherals required for this example and generate code for CodeWarrior compiler.

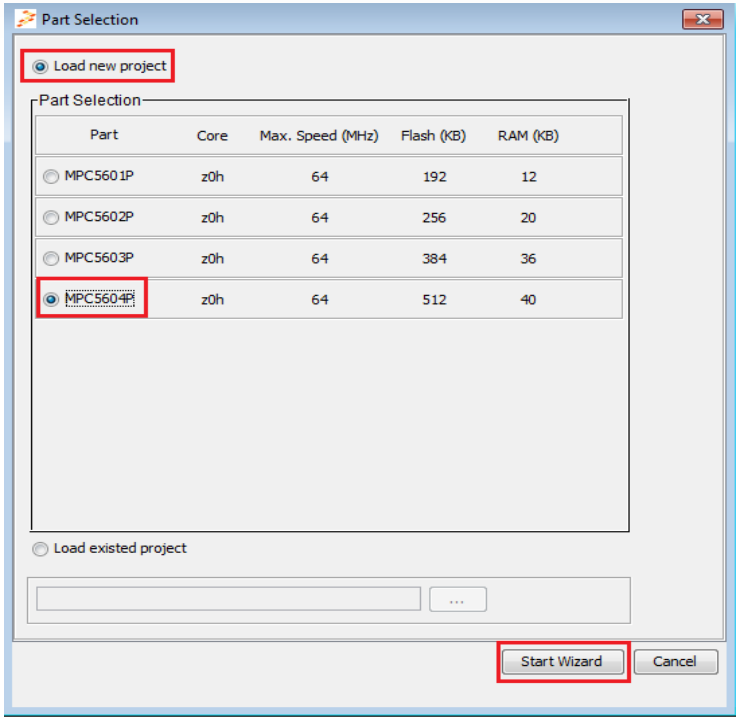
Create a New RAppID Project

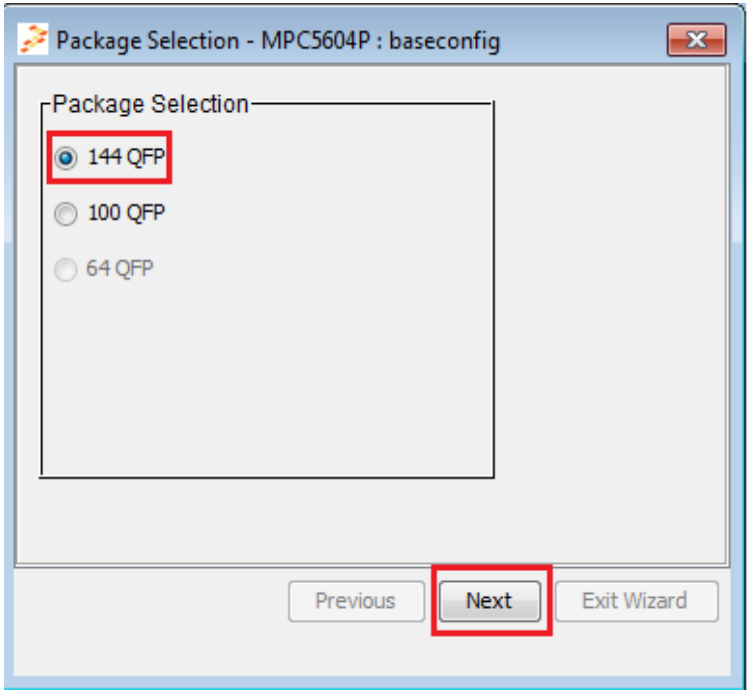
Launch RAppID init by double clicking on RAppID init desktop icon



Start a new project by clicking on “New Project Wizard” button

Select Part and Package

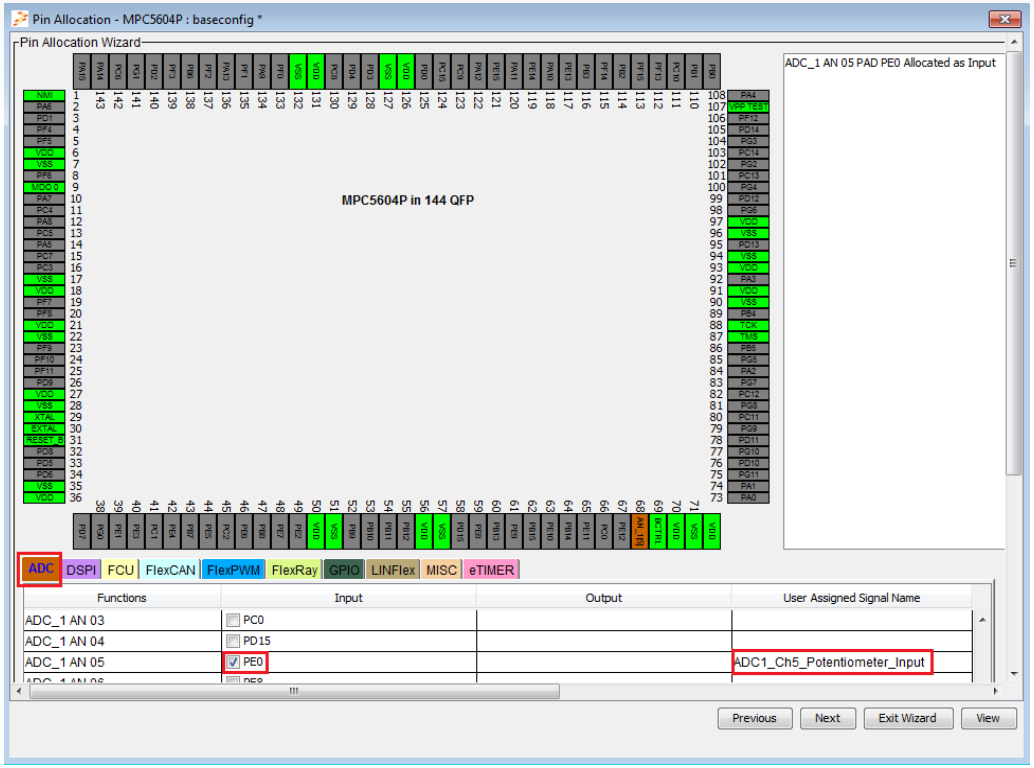




Select a Part MPC5604P and select Start Wizard

Select a Package 144 QFP and select Next

Configure ADC input



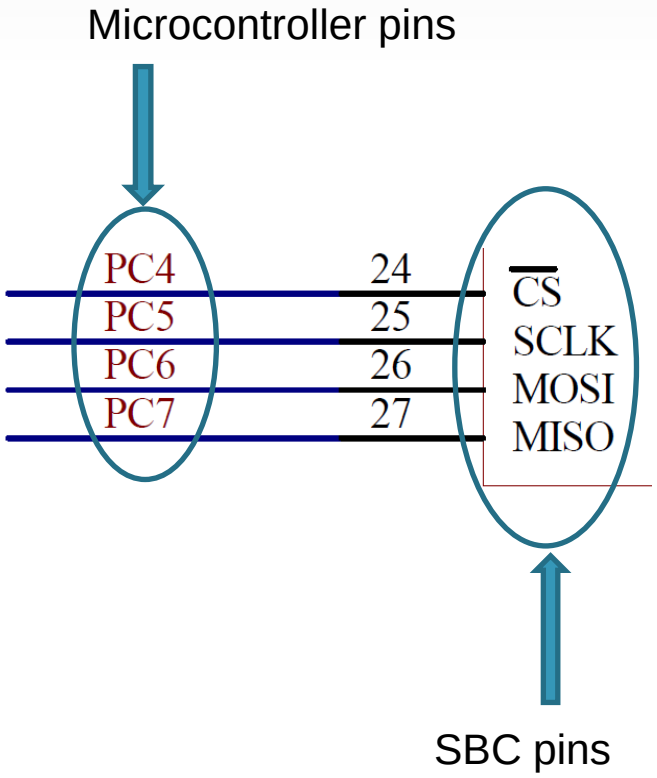
Potentiometer is connected to PE0.
In ADC tab, configure PE0 as ADC input and add appropriate name in User Assigned Signal Name edit box.

Configure DSPI pins

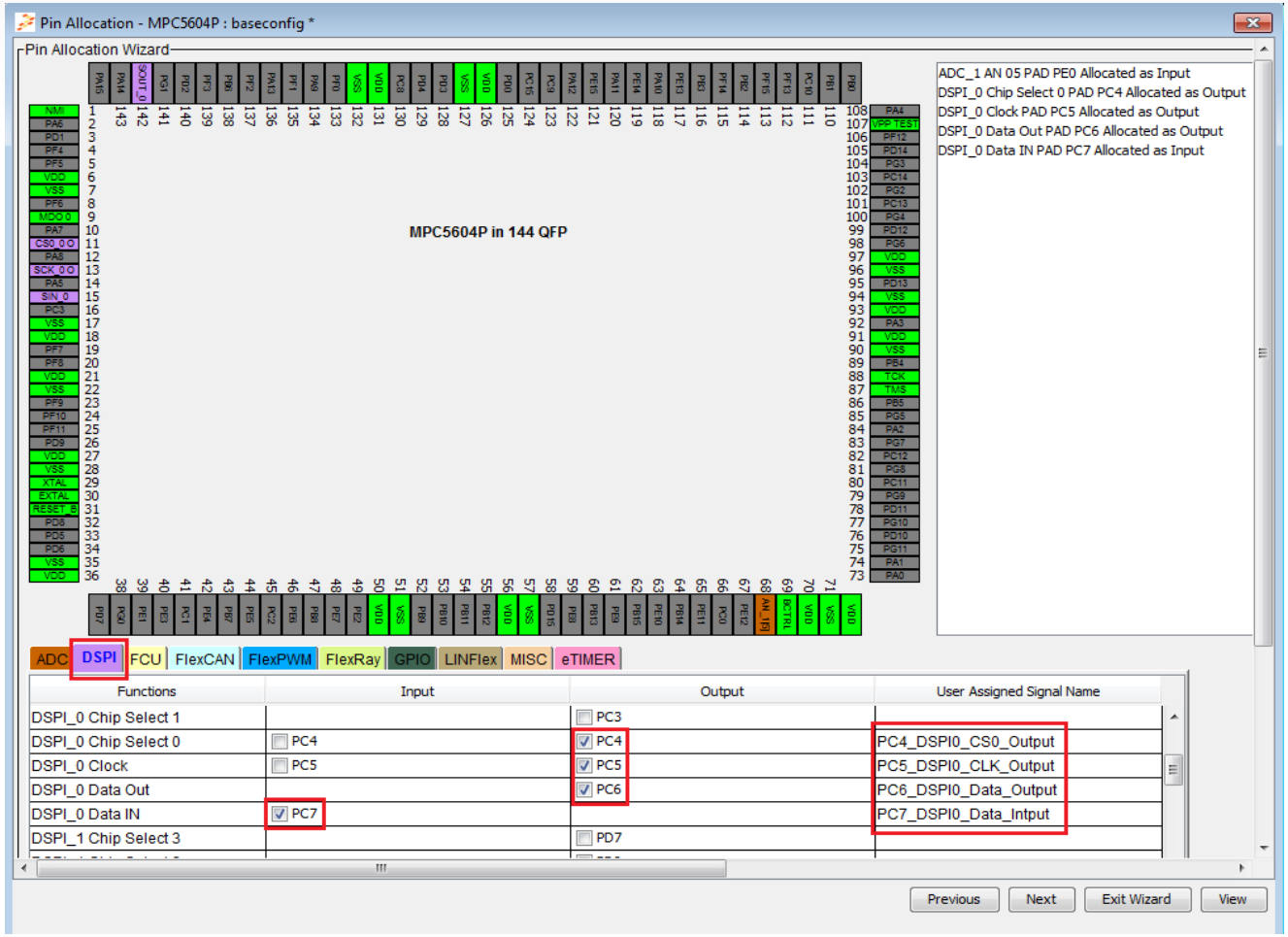
- TRK-MPC5604P contains MCZ3390S5EK system basis chip (SBC) with integrated CAN transceiver and LIN 2.0 interface.
- DSPI 0 is connected to SBC. We need to configure DSPI 0 as master and SBC as slave so that SBC can be configured to enable CAN by sending appropriate commands via DSPI 0.

Configure DSPI pins

- The connection between DSPI 0 of microcontroller and SBC is shown in this picture
- Configure DSPI 0 pins as follows to enable communication with SBC.
  - PC4 is connected to CS pin of SBC. Configure PC4 as DSPI_0 Chip select 0 output
  - PC5 is connected to Clock input pin of SBC. Configure PC5 as DSPI_0 Clock output
  - PC6 is connected to MOSI pin of SBC. Configure PC6 as DSPI_0 Data output
  - PC7 is connected to MISO pin of SBC. Configure PC7 as DSPI_0 Data input

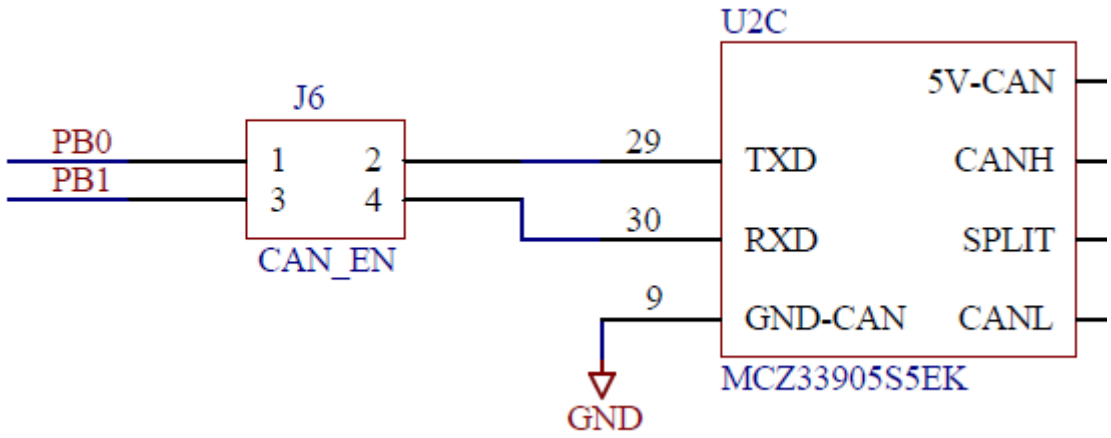


Configure DSPI pins

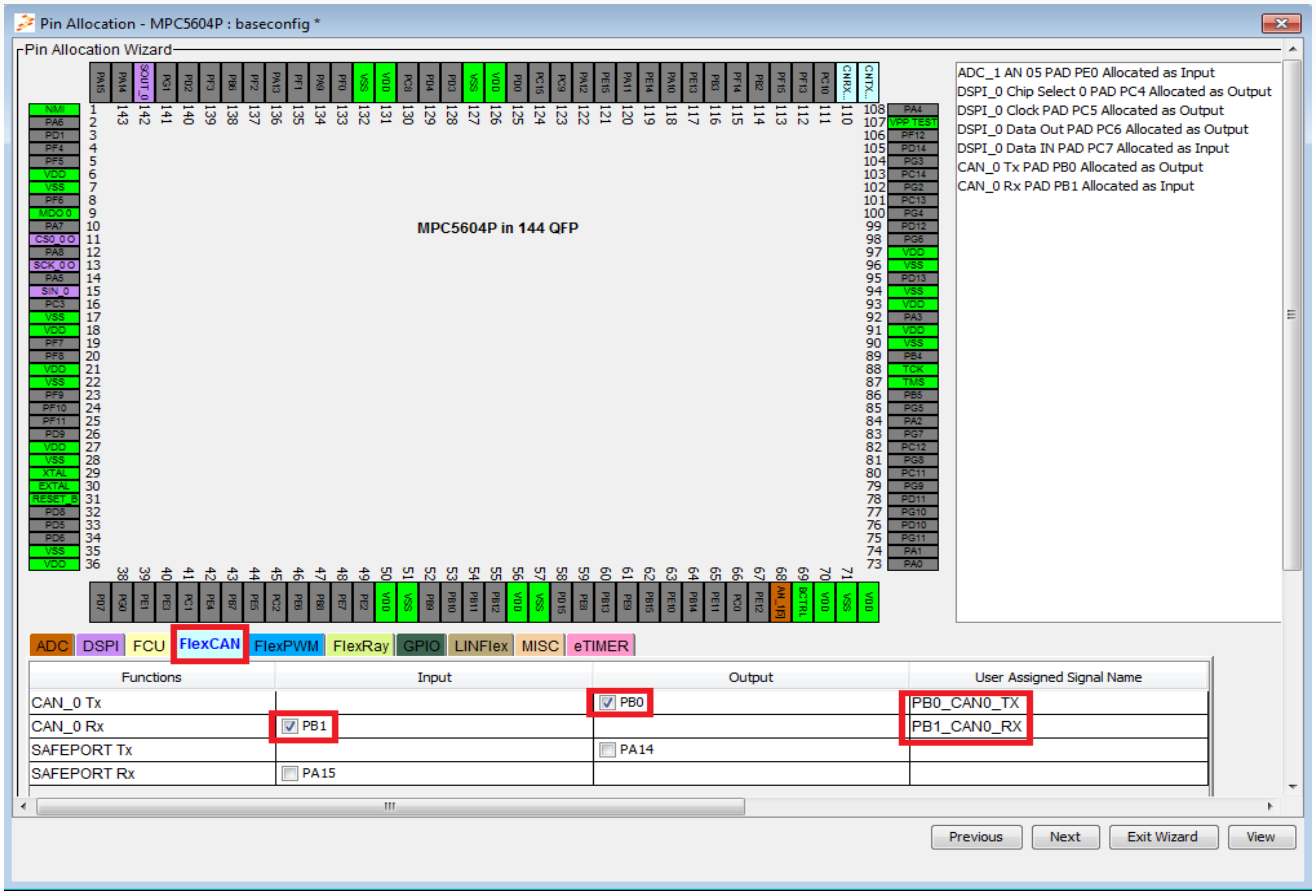


Configure FlexCAN pins

- The CAN TX and CAN RX pins of SBC are connected to the pins PB0 and PB1 of CAN 0 peripheral of the microcontroller.
- We need to configure PB0 as CAN 0 TX pin and PB1 as CAN 0 RX pin.

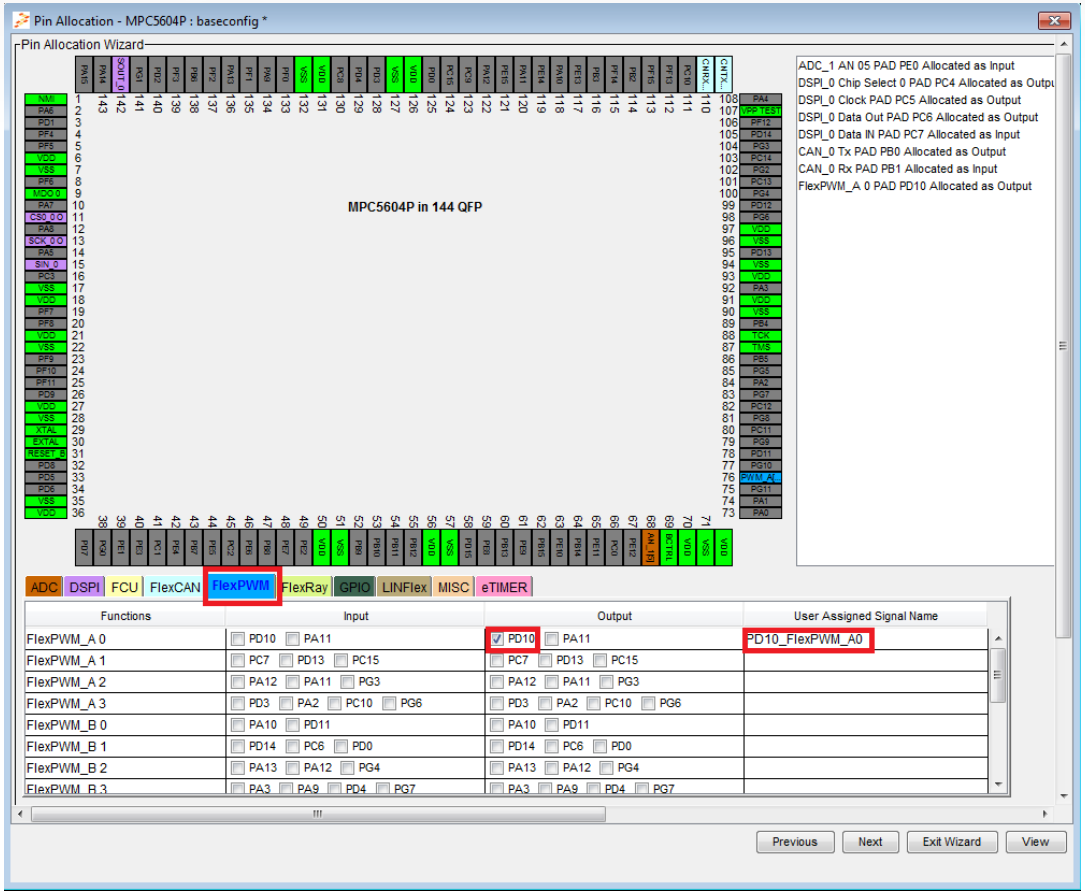


Configure FlexCAN pins



In FlexCAN tab, configure PB0 as CAN_0 Tx and PB01 as CAN_0 Rx pins and signal names.

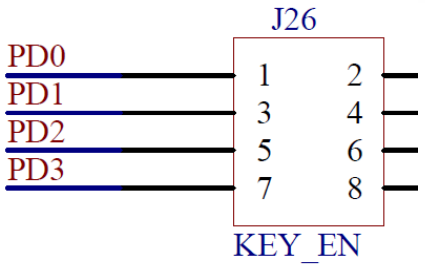
Configure FlexPWM pin

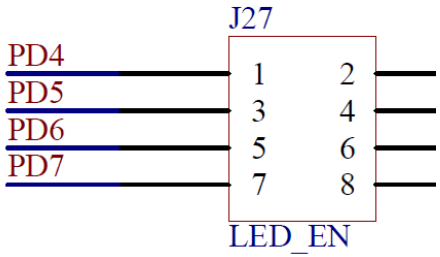


In this example, we are using PD10 as FlexPWM output. In FlexPWM tab, select PD10 as output and add user signal name.

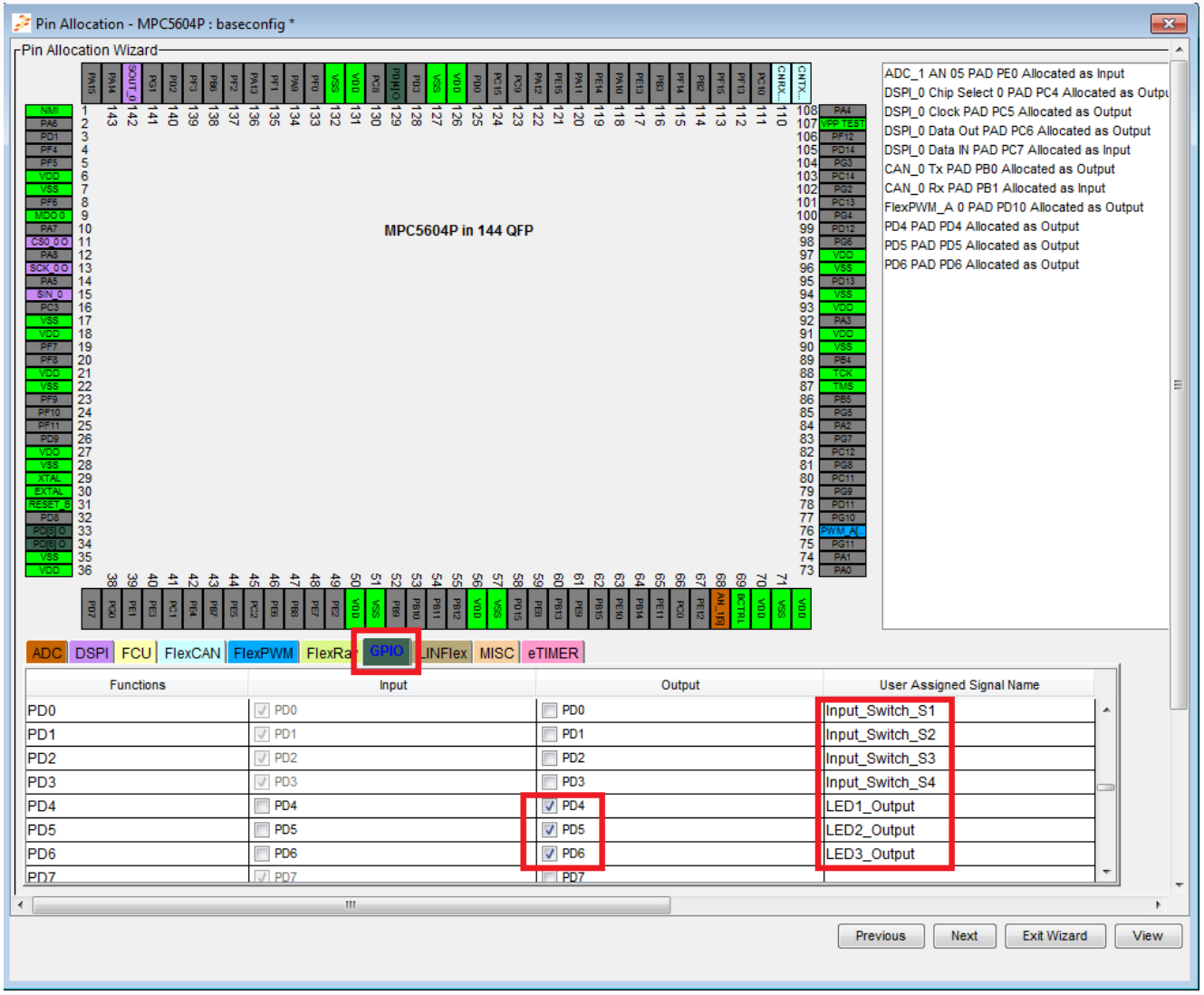
Configure GPIO pins

- In this example, we are using Switch S1, S3 and S4 as input and the LED1, LED2 and LED3 as outputs.
- Switch S1, S3 and S4 are connected to PD0, PD2 and PD3. Configure these 3 pins as inputs.
- LED1, LED2 and LED3 are connected to PD4 PD5 and PD6. Configure these 3 pins as outputs.



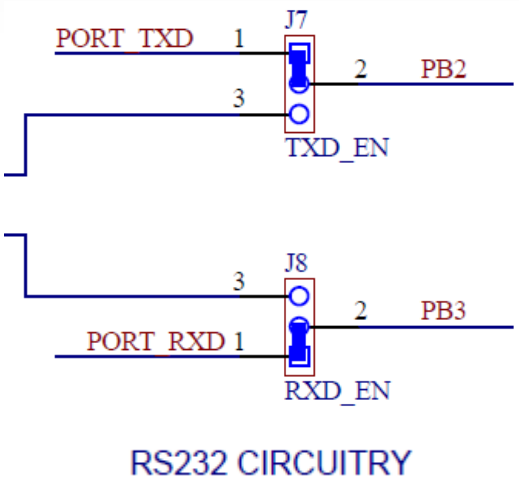


Configure GPIO pins



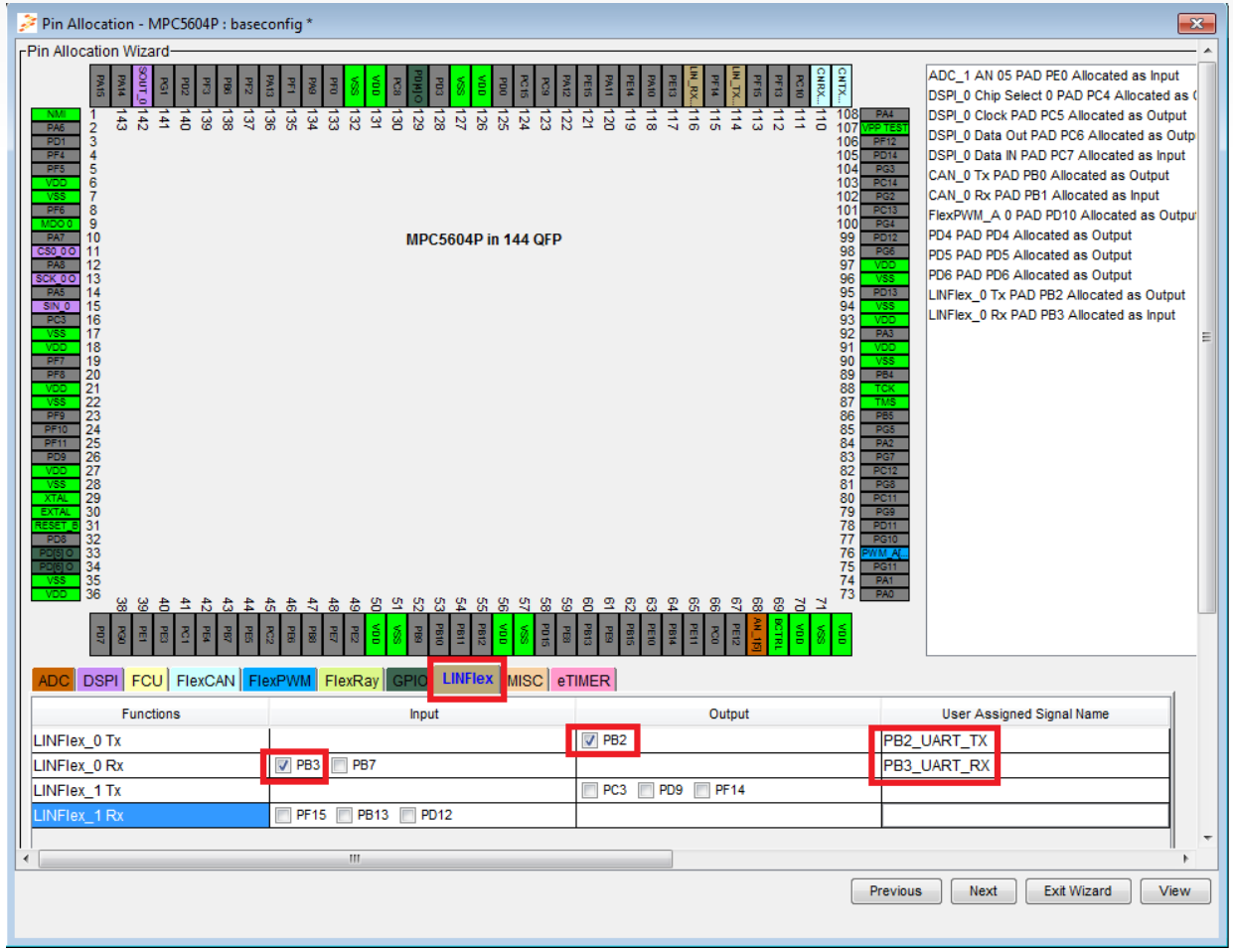
Configure LINFlex (UART) pins

- In this example, we will use Virtual serial port of TRK-MPC5604P board for serial communication.
- The PB2 and PB3 of microcontroller in TRK-MPC5604P board are connected to TX and RX pins of virtual serial port.
- We need to configure PB2 as LINFlex0 TX pin and PB3 as LINFlex0 RX pin.



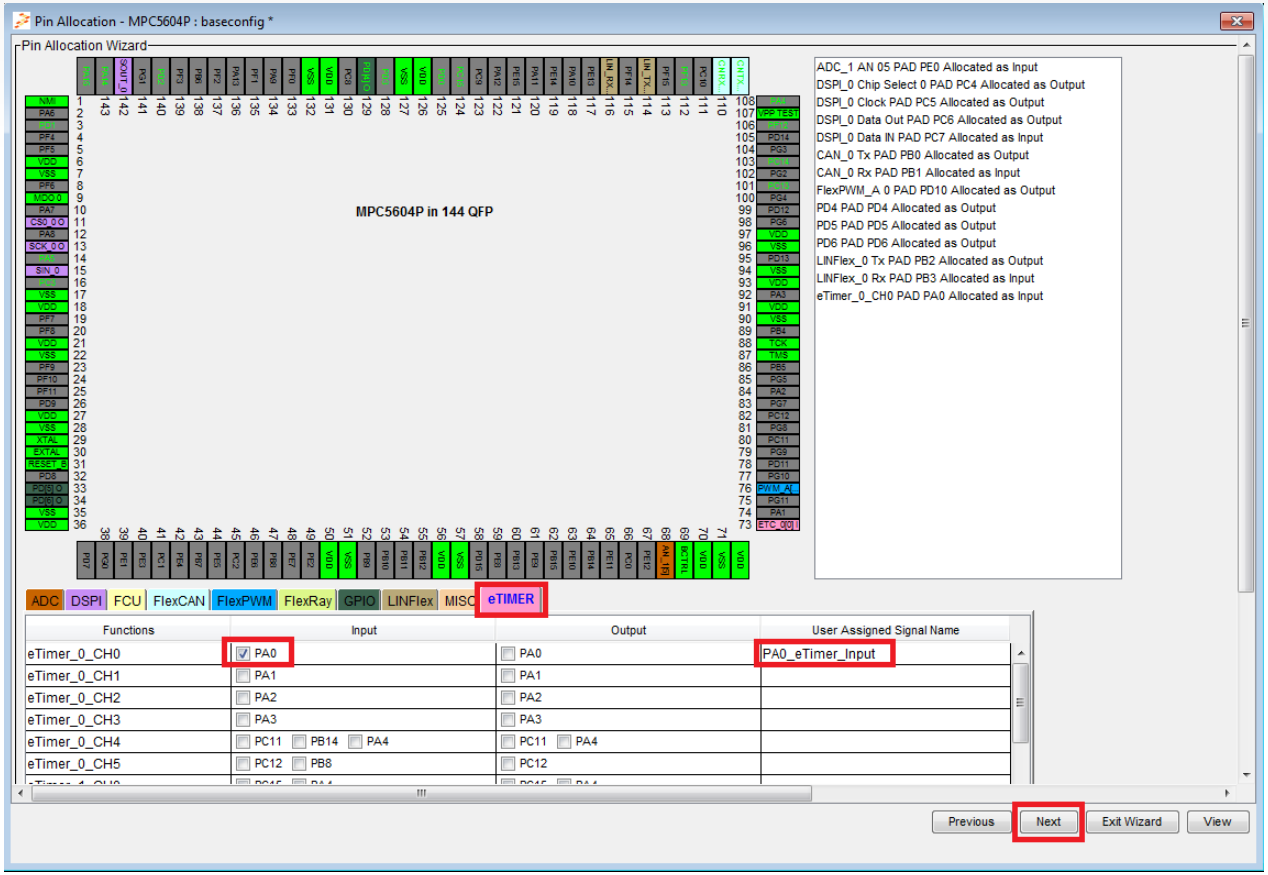
RS232 CIRCUITRY

Configure LINFlex (UART) pins



In LINFlex tab, configure PB2 as LINFlex_0 Tx and PB3 as LINFlex_0 Rx pins and add user signal names.

Configure eTimer pin

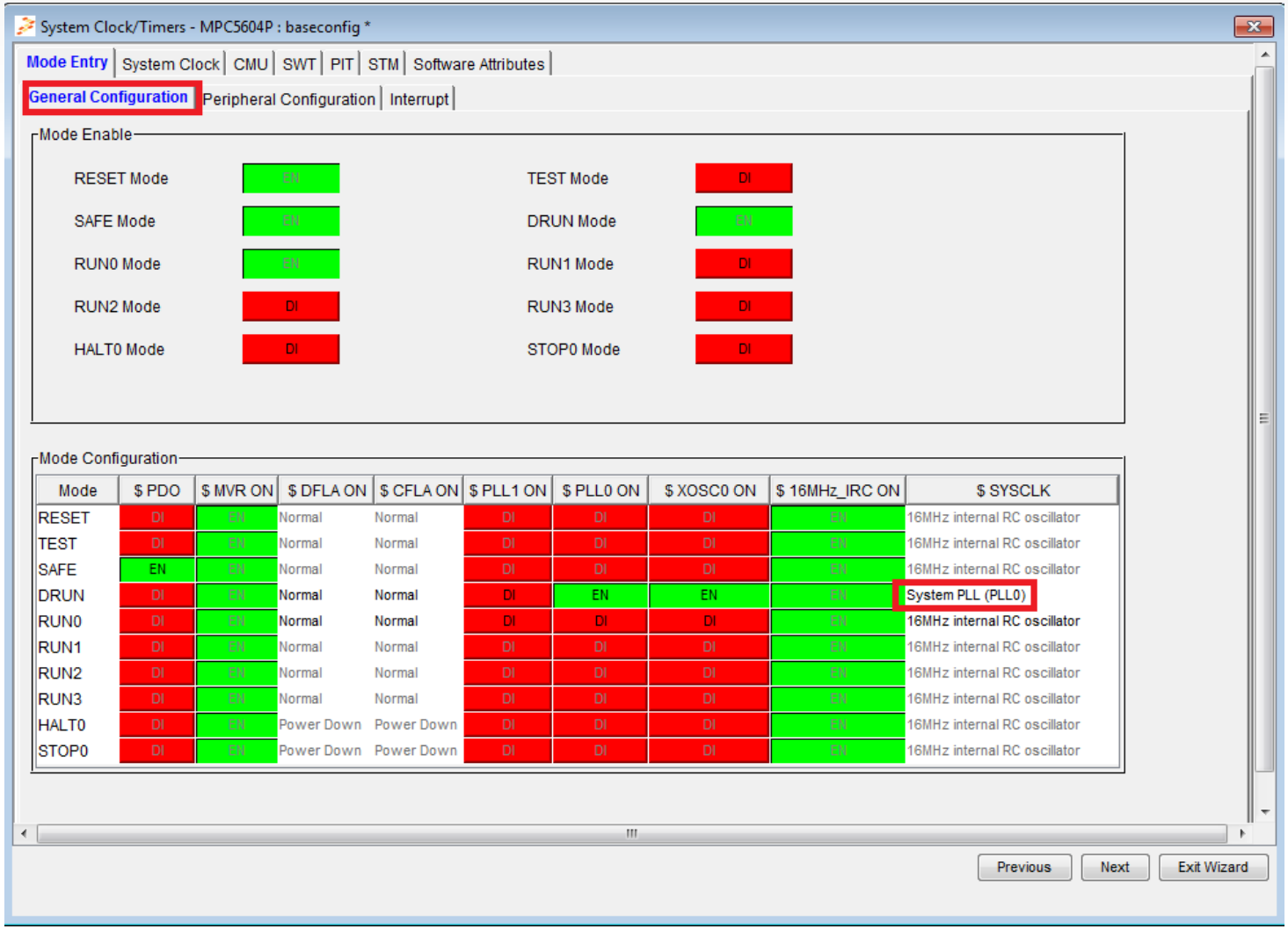


In this example, we are using eTimer Channel 1 as input capture function to calculate duty signal of PWM signal. In eTimer tab, configure PA0 as eTimer input pin and add user signal names.
Select Next and skip next window by selecting Next to configure Mode Entry

Configure Mode Entry

- RAppID tool generates code to set the microcontroller in DRUN mode at startup.
- In this example, we will use system PLL (PLL0) as system clock source. Select System PLL from the drop down under SYSTEM column in DRUN mode.

Configure Mode Entry



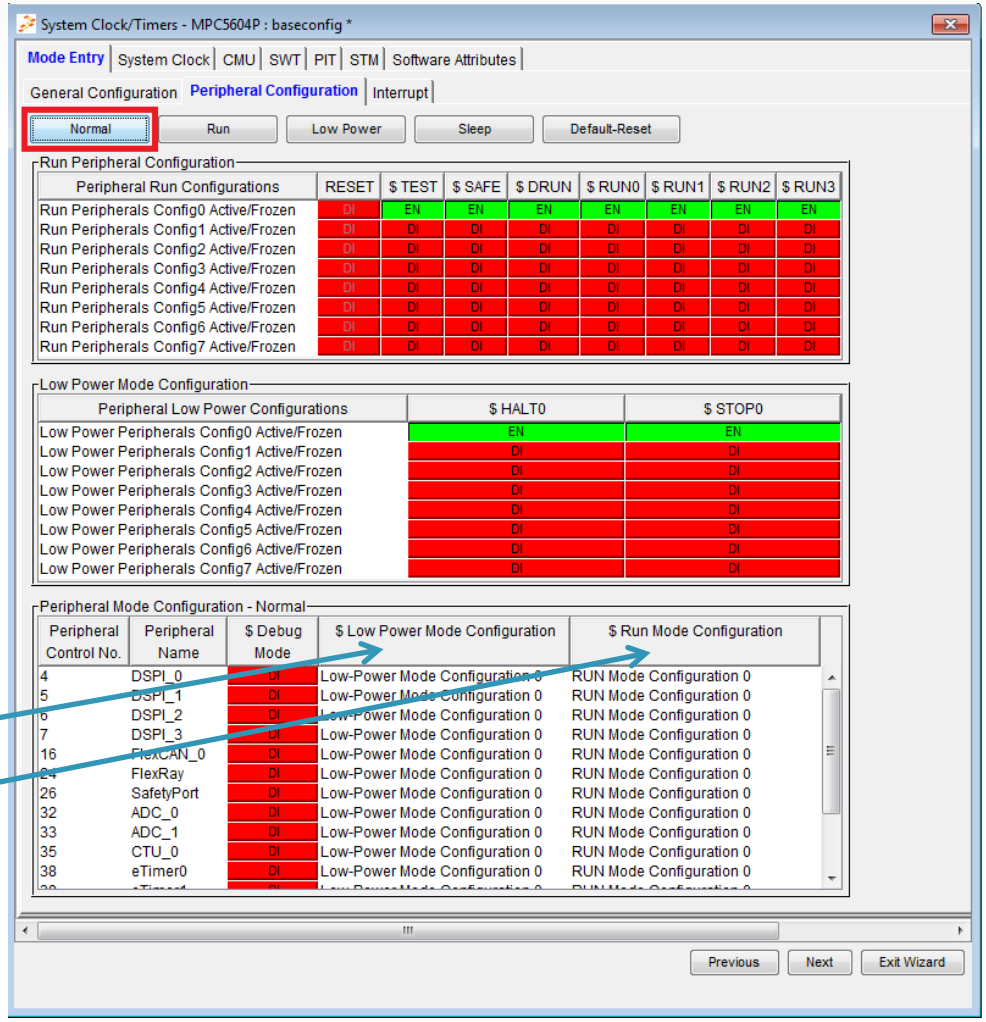
Configure Mode Entry

Configure the peripherals to be enabled during different operational modes.

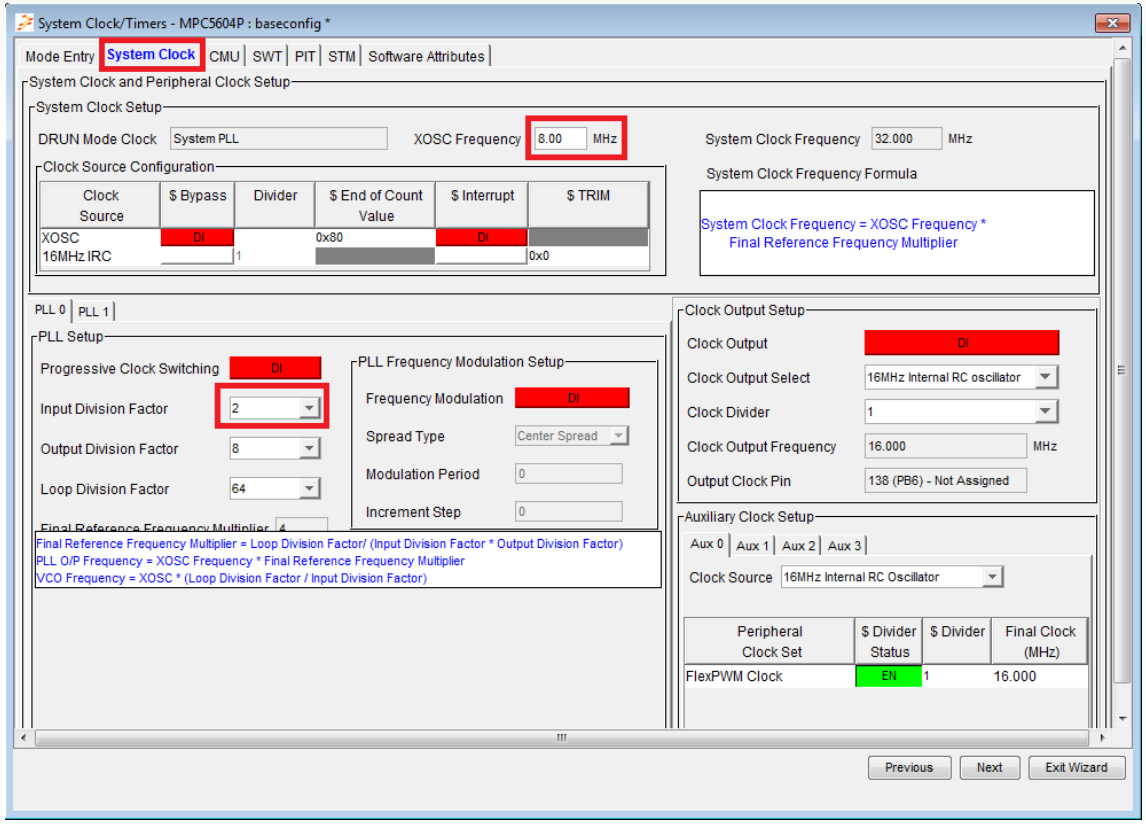
Select “Normal” configuration

This will enable the “peripheral run configuration” and “low power configuration” across different operational modes

Then it will assign these two configurations across all peripherals



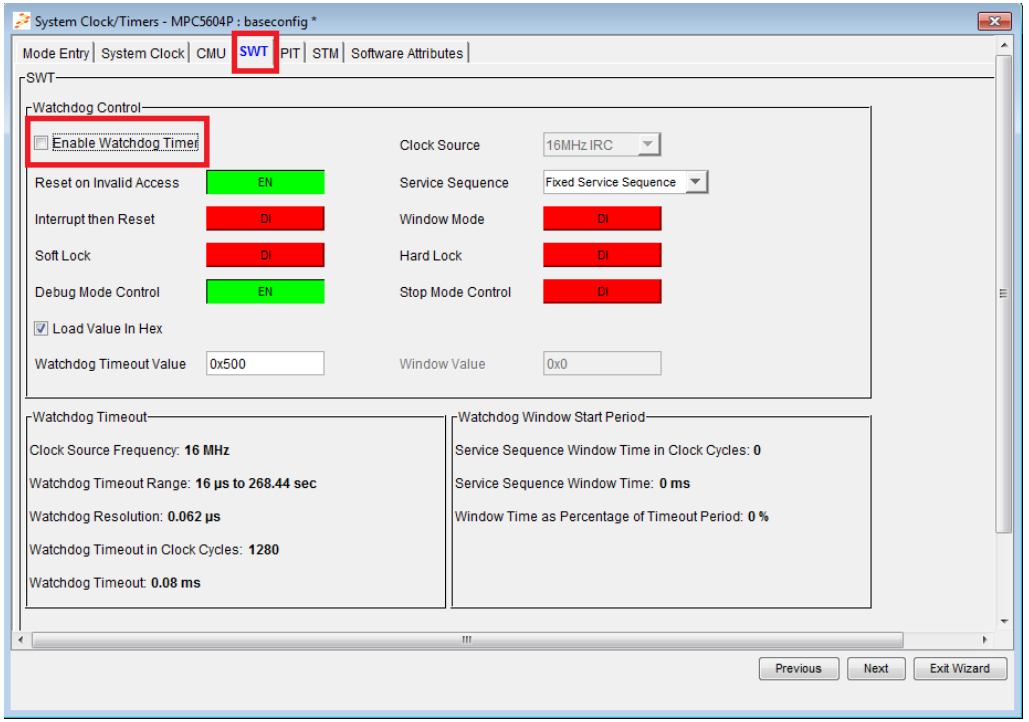
Configure Clock



We will configure system clock to 32MHz. The default XOSC frequency in RAppID is 40MHz but TRK-MPC5604P uses 8 MHz crystal. Change XOSC value to 8 MHz. Change Input Division factor to 2 to set the system clock at 32 MHz.

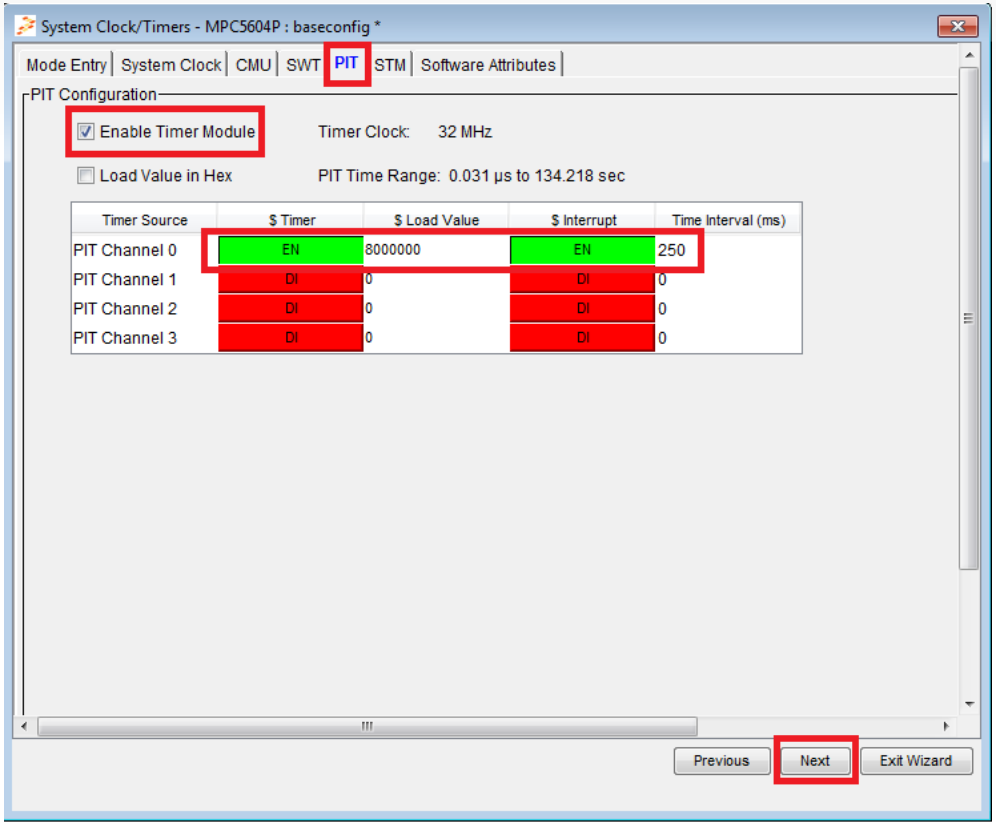
Disable Watchdog

By default, watchdog is enabled in MPC5604P. In this example, we will not use Watchdog feature. Disable Watchdog Timer in SWT tab .

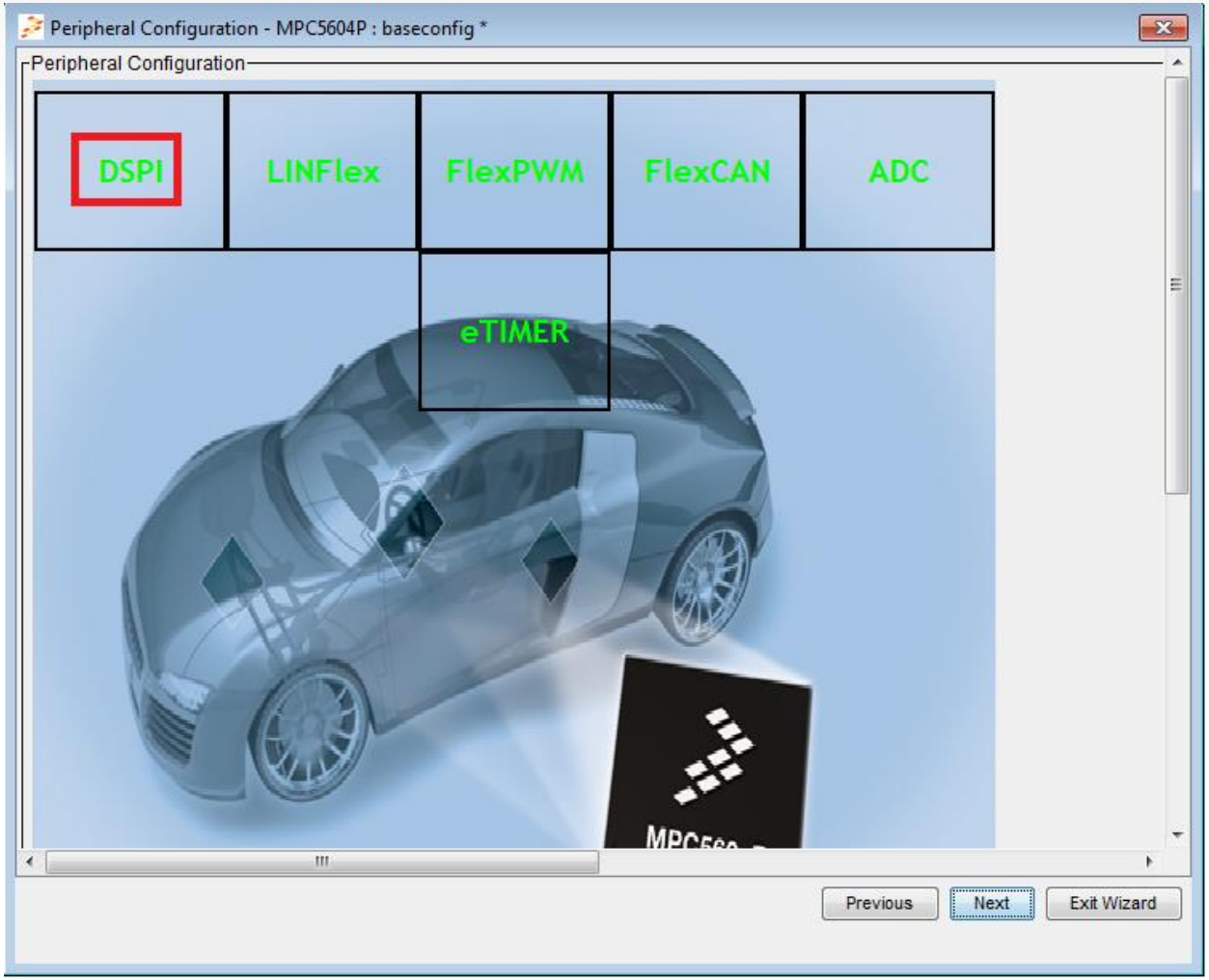


Configure PIT

We will monitor switch inputs S3 and S4 every 250ms in PIT Channel 0 interrupt. Select PIT tab and
-Enable Timer module
-Enable PIT Channel 0 timer
- Set load value to 8000000 to set the time out value to 250 ms
-Enable PIT Channel 0 interrupt
-Select Next to start peripheral configuration.



Configure Peripherals



In Peripheral Configuration window, select DSPI to configure DSPI peripheral

Configure DSPI 1

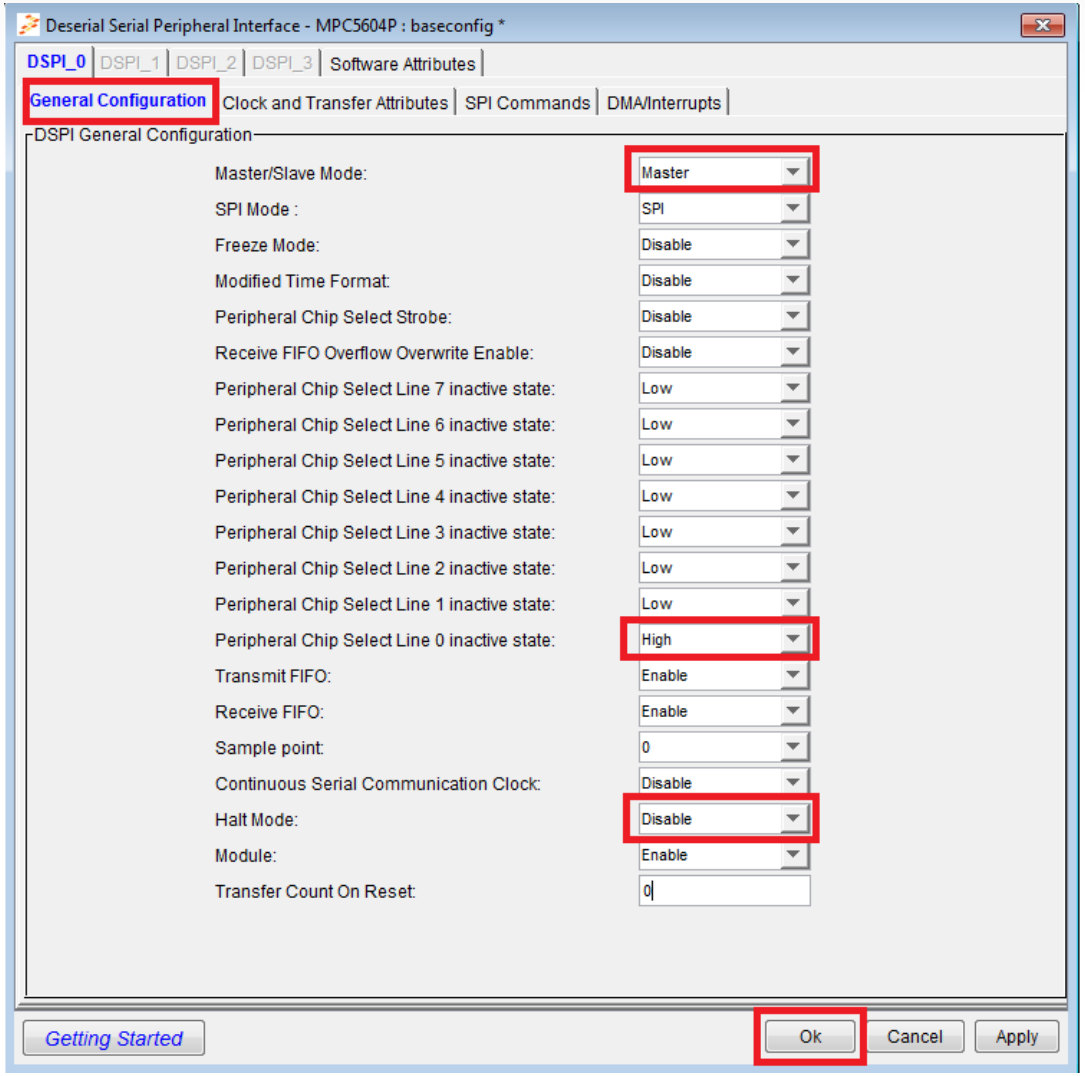
DSPI 0 should be set to master mode to send commands to SBC. Select Master mode

DSPI 0 Chip Select 0 is connected to SBC. Set Chip select 0 inactive state to High

Disable Halt mode

Select OK to finish DSPI 0 configuration.

In the Peripheral Configuration window, select LinFlex tile to start LinFlex configuration.



Configure LINFlex 0

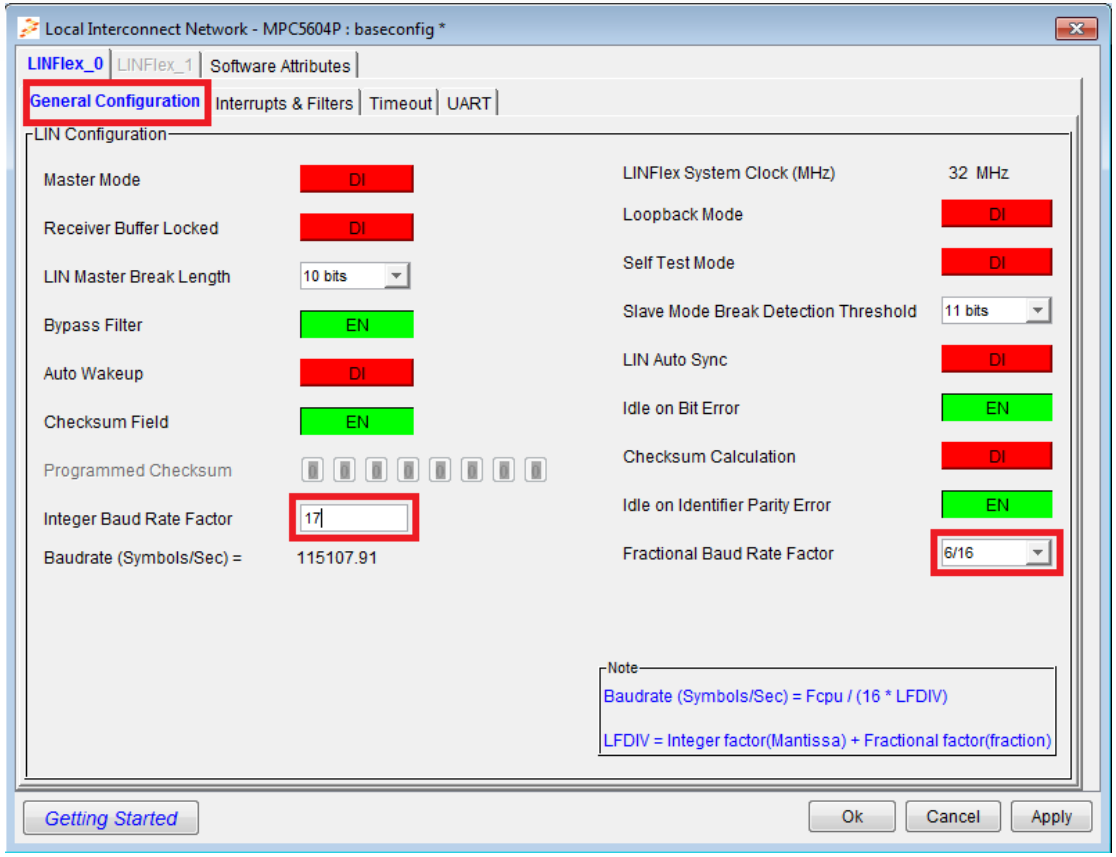
- We will use UART of LINFlex 0 to communicate serially via virtual serial port of TRK-MPC5604P board
- We will use baud rate of 115,200
- When Baud Rate Factor and Fractional Baud Rate Factor values are selected, RAppID automatically calculates and displays the resulting baud rate.

Configure LINFlex 0

Set Baud Rate Factor to 17

Set Fractional Baud Rate Factor to 6/16

This should set the Baud rate to approx. 115,200



Configure LINFlex 0

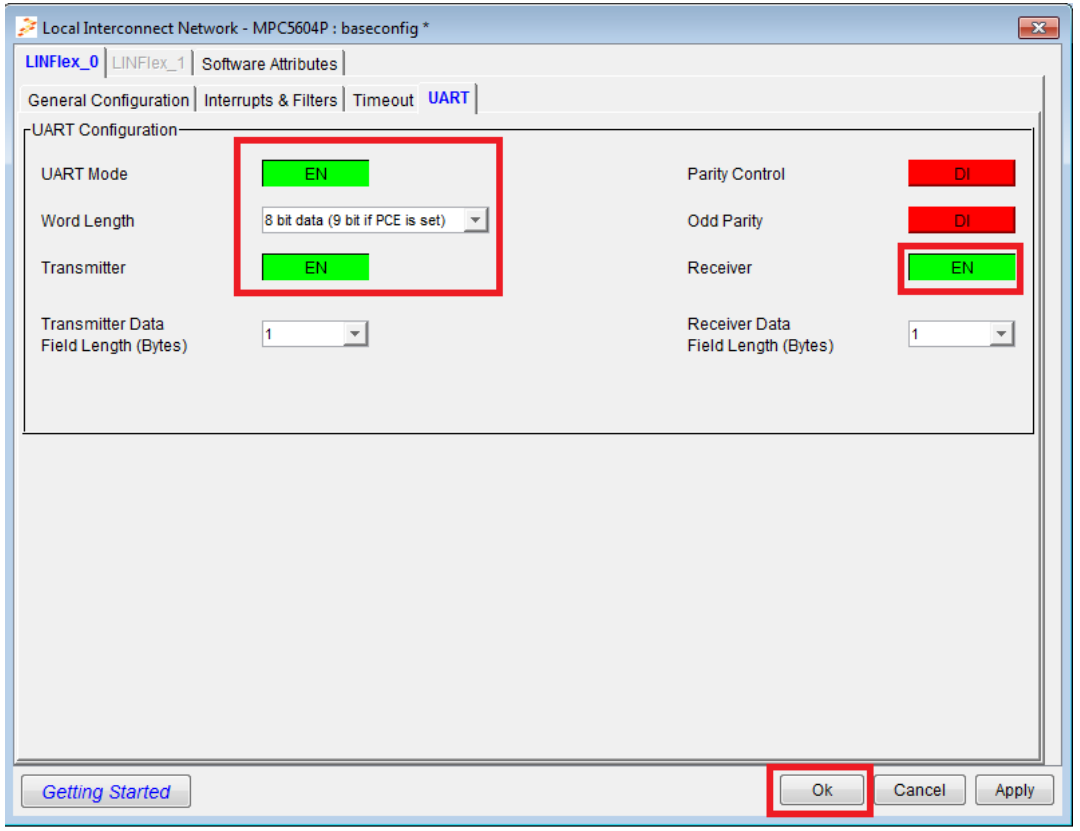
In UART tab, Enable UART

Set Word Length to 8 bit data

Enable Transmitter and Receiver

Select OK to finish LINFlex 0 configuration

In the Peripheral Configuration window, select FlexPWM tile to start FlexPWM configuration.



Configure FlexPWM

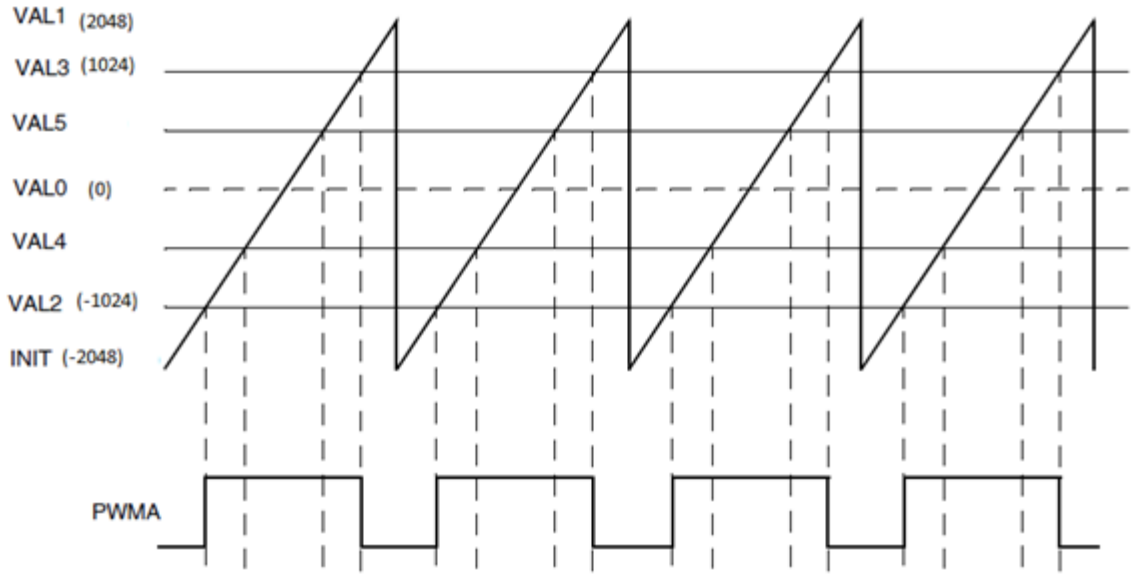
In this example, we will configure FlexPWM sub module 0 , PWM A output to generate center-aligned PWM signals. The values of FlexPWM parameters are set as shown.

The PWM clock source is 16 MHz IRC clock.

The Period of PWM signal = (2048 + 2048) * 1/16 = 256 ms

The PWM signal high time = (1024+1024) * 1/16 = 128 ms.

Duty Cycle = 128/256 = 50%



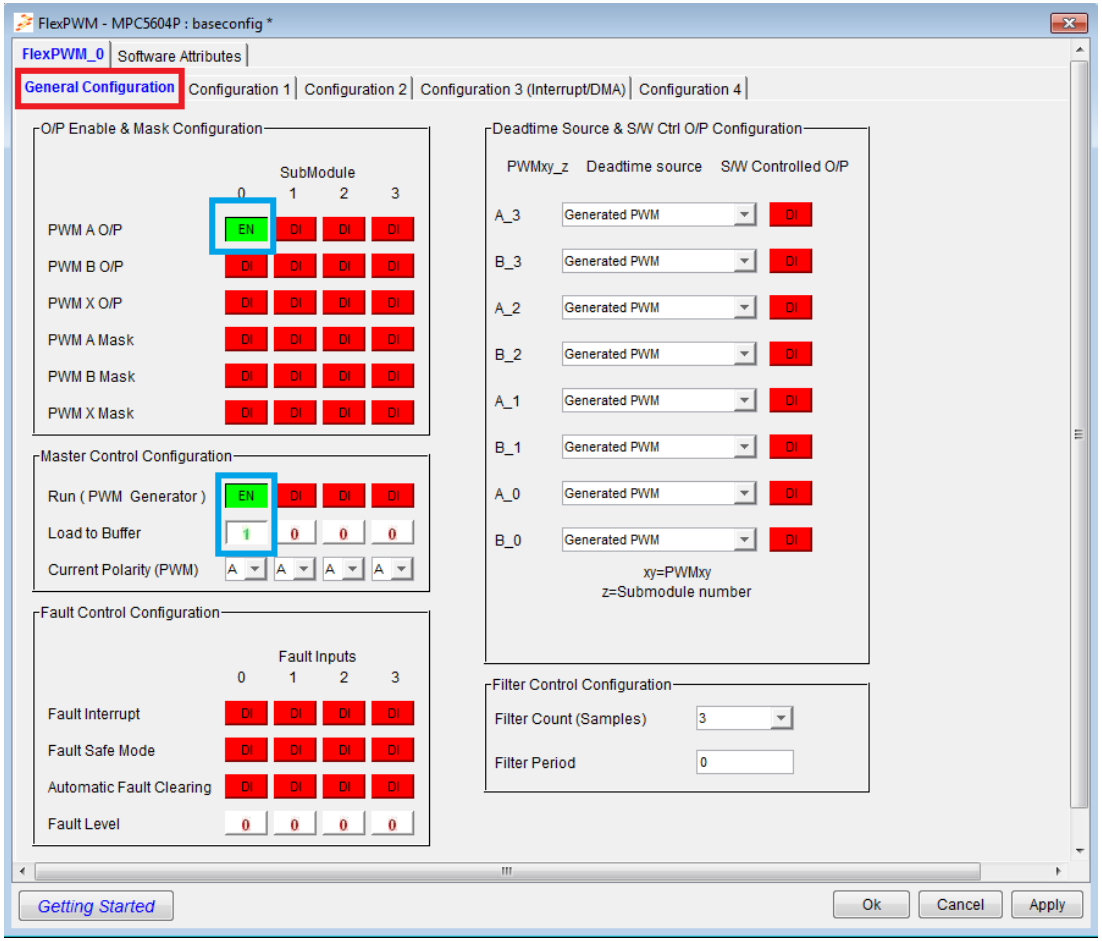
Configure FlexPWM

We need to configure PD10 output (PWM A, sub module 0) for PWM signal

-Enable PWA 0 Output

- Enable PWM Generator for sub module 0

-Set Load Okay bit

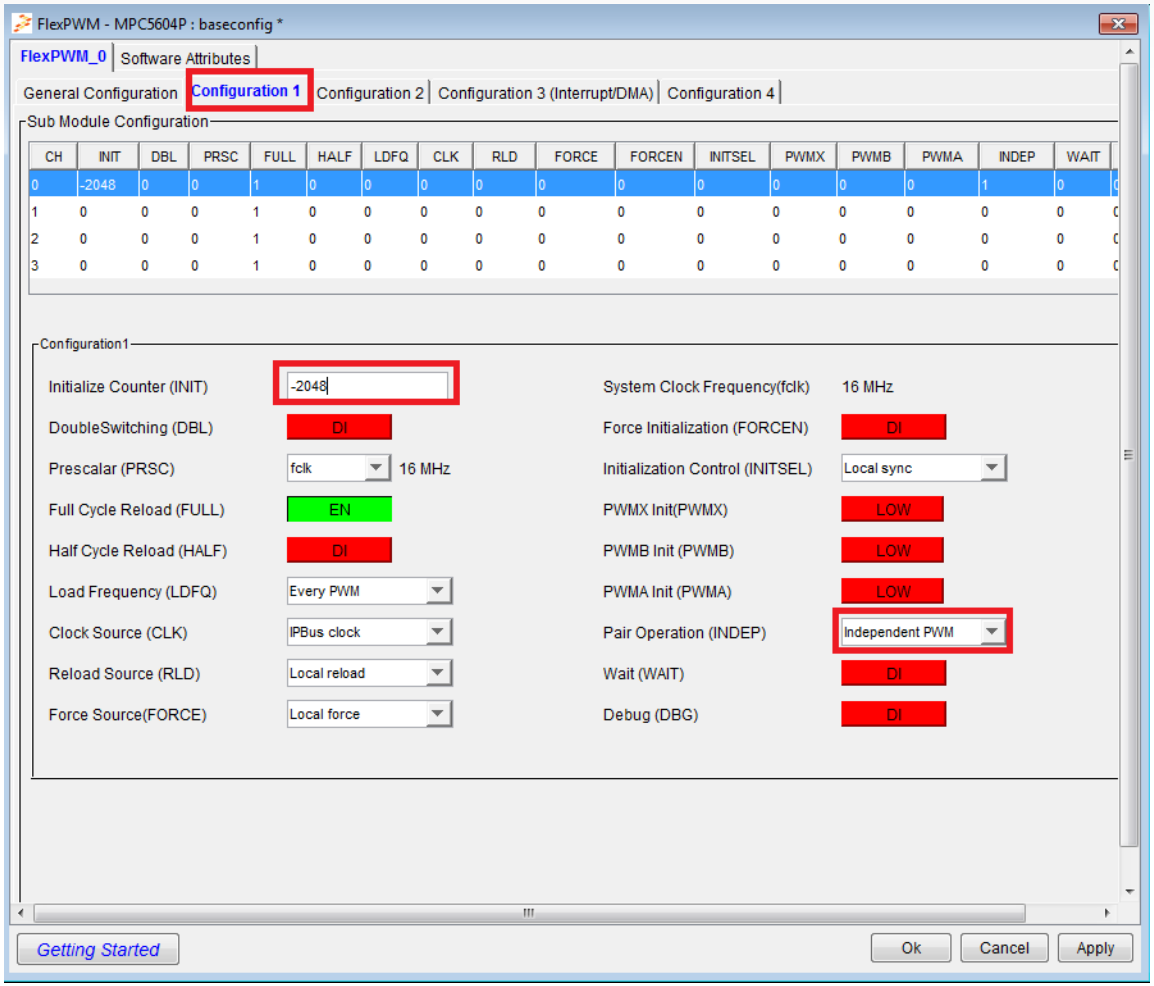


Configure FlexPWM

In Configuration 1 tab

-Set INIT counter value to -2048

-Set Pair Operation to Independent PWM



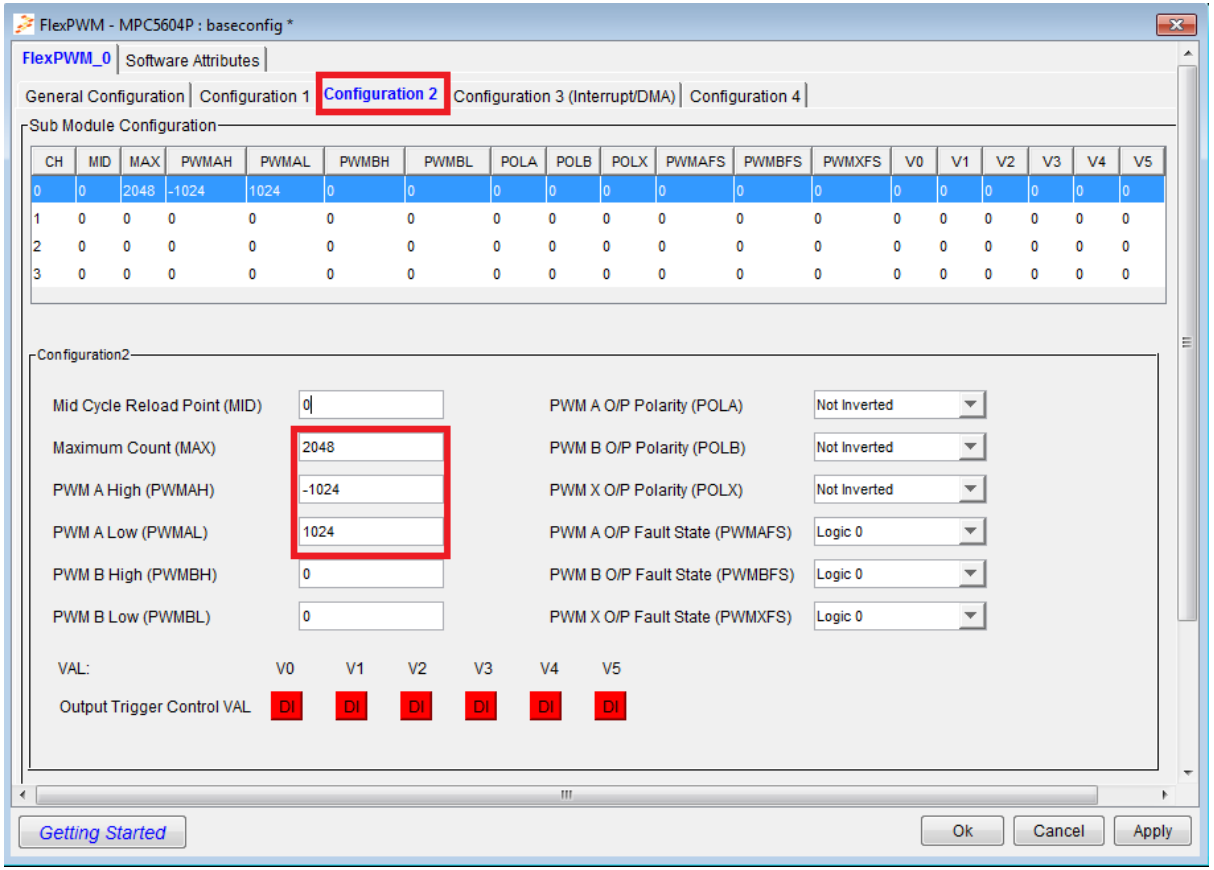
Configure FlexPWM

In Configuration 2 tab

-Set MAX counter value to 2048

-Set PWM A High to -1024

-Set PWM A Low to 1024



Configure FlexPWM

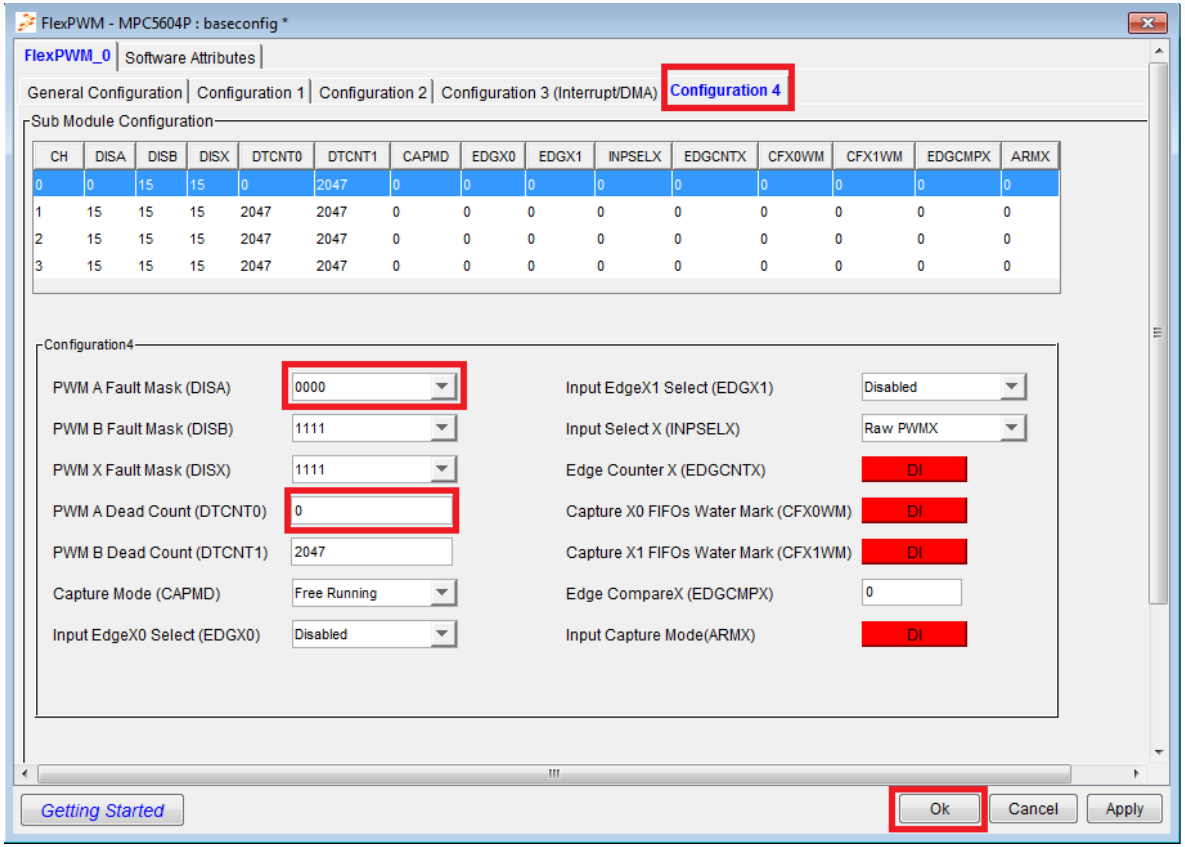
In Configuration 4 tab

-Set PWM A Fault Mask to 0

-Set PWM A Dead Count to 0

-Select Ok to finish FlexPWM configuration.

-In the Peripheral Configuration window, select FlexCAN tile to start FlexCAN configuration.



Configure FlexCAN

In TRK-MPC5604P board, the CAN 0 peripheral is connected to CAN transceiver in SBC.
In this example, we will use CAN speed of 500 k bits/sec.

Enable CAN module

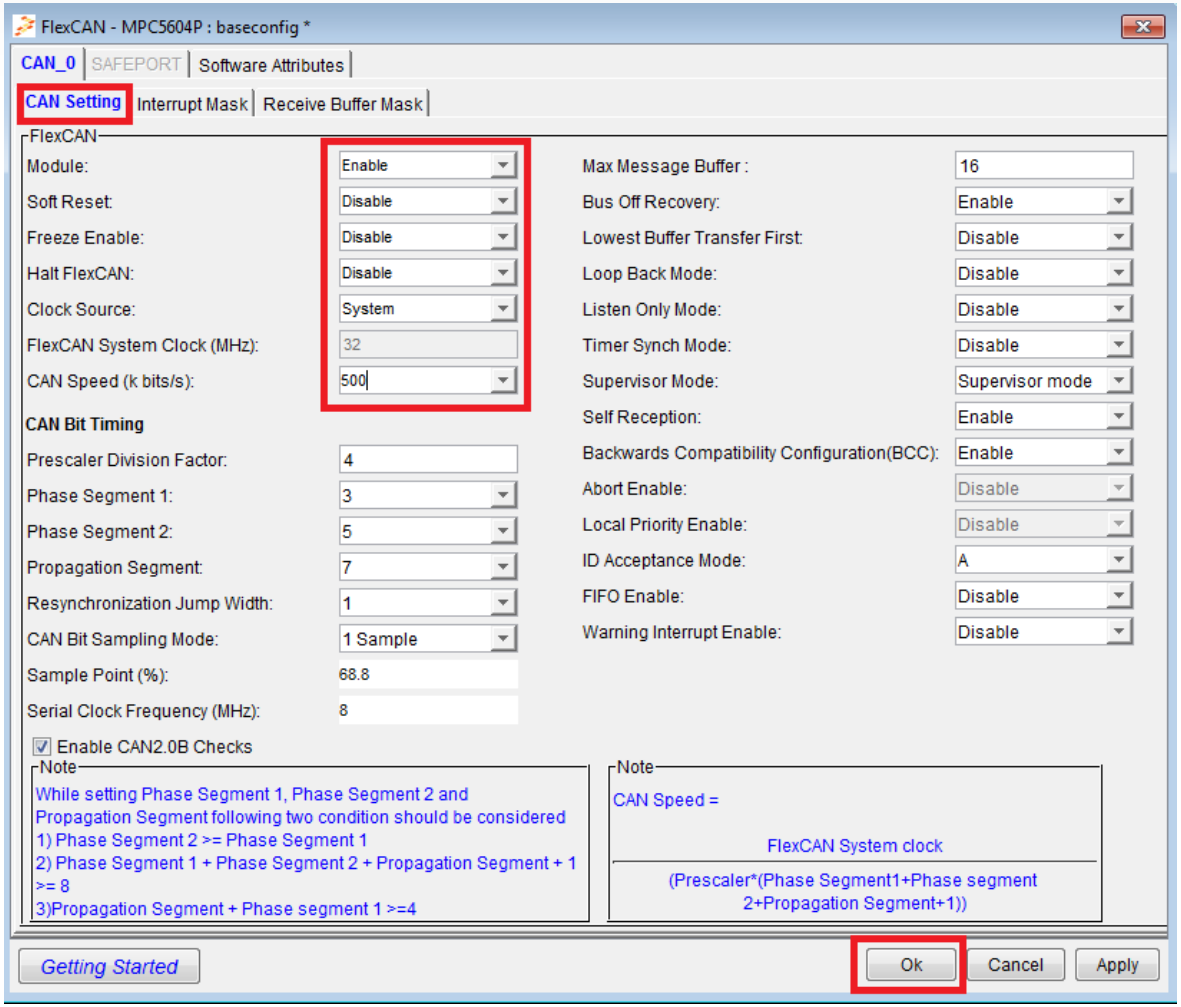
Disable Freeze and Halt modes

Set Clock Source to System

Set CAN speed to 500. RAppID will configure Phase segments and Propagation segment values automatically.

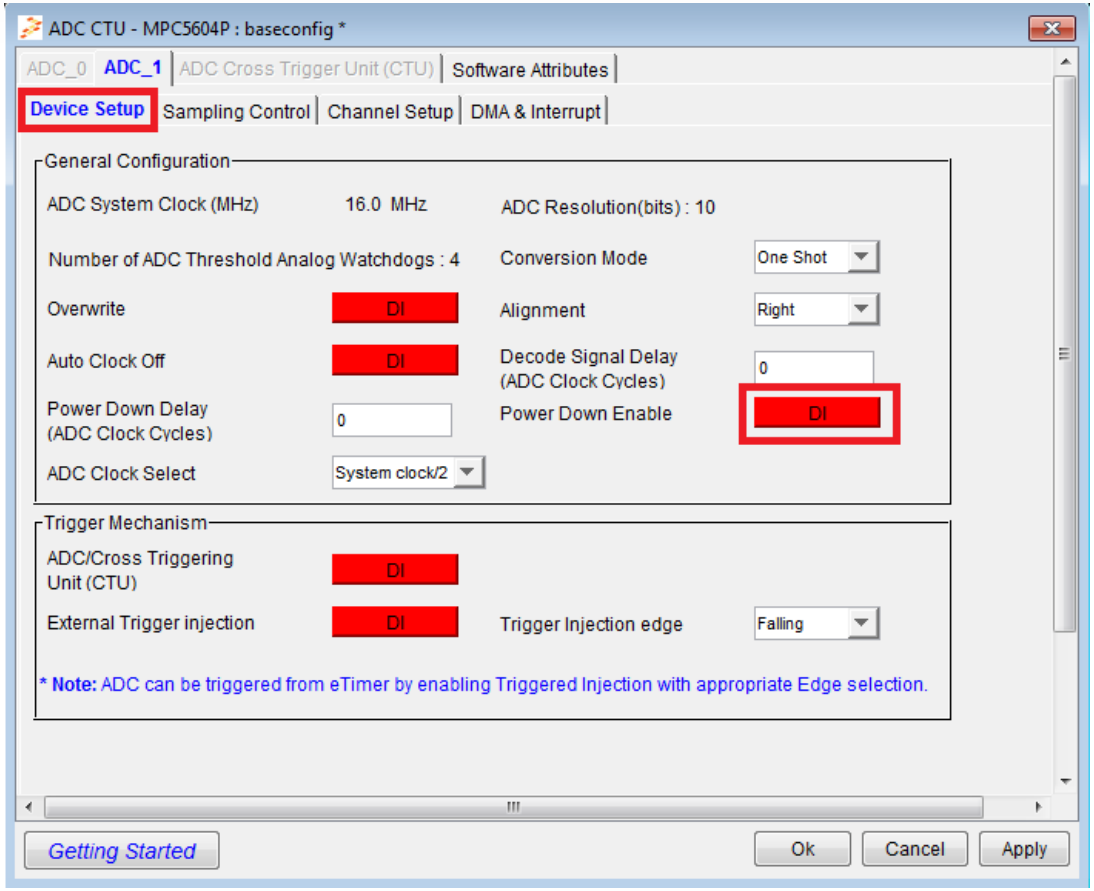
Select OK to finish CAN configuration.

In the Peripheral Configuration window, select ADC tile to start ADC configuration.



Configure ADC 1

In the Device Setup tab, Disable Power Down option to put ADC in normal mode

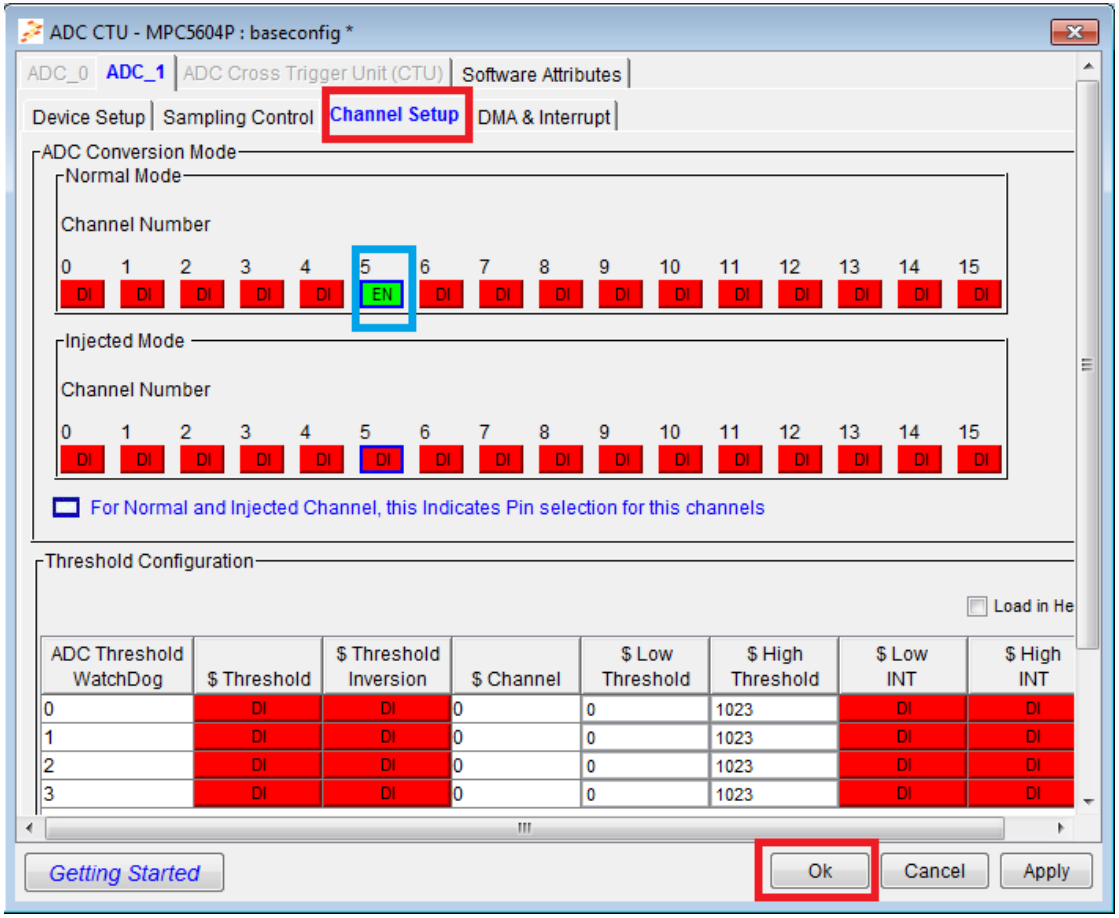


Configure ADC 1

The Potentiometer in TRK-MPC5604P board is connected to PE0 which is ADC 1 Channel 5.
In this example we will use Normal conversion mode.
Enable Channel 5 in Normal mode

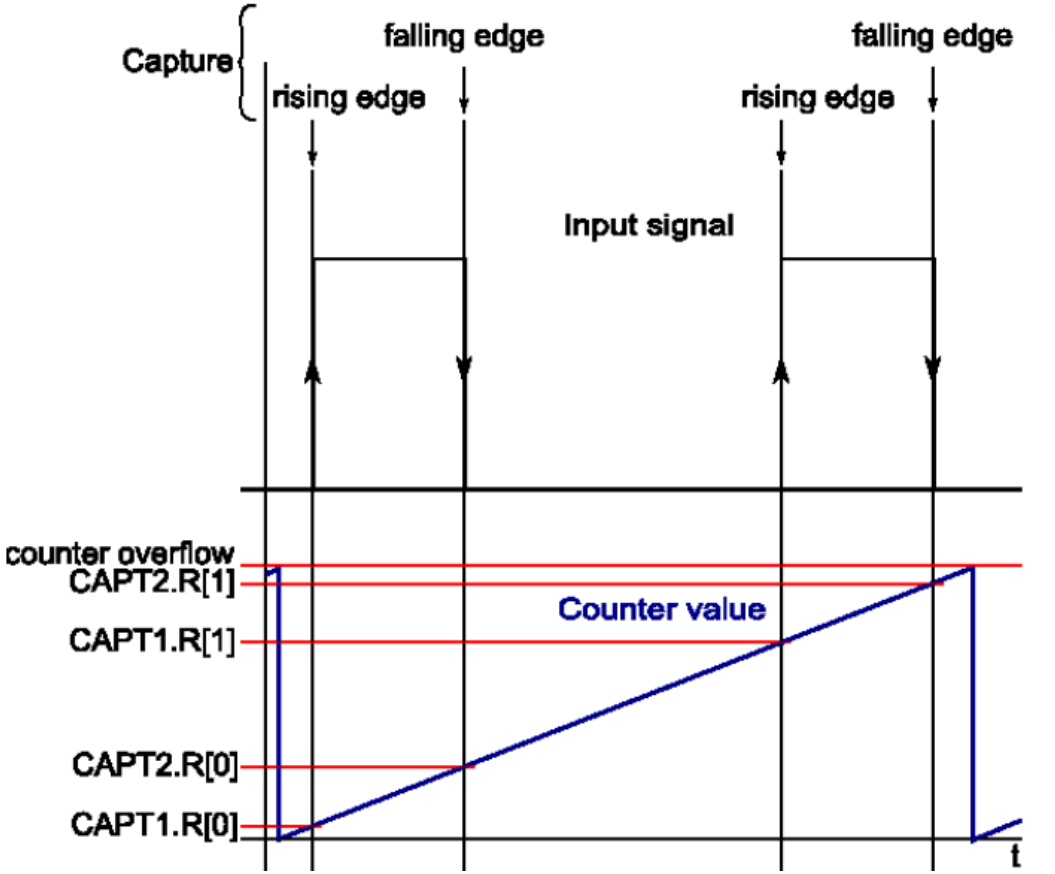
Select OK to finish ADC peripheral configuration.

In the Peripheral Configuration window, select eTimer tile to start eTimer configuration.



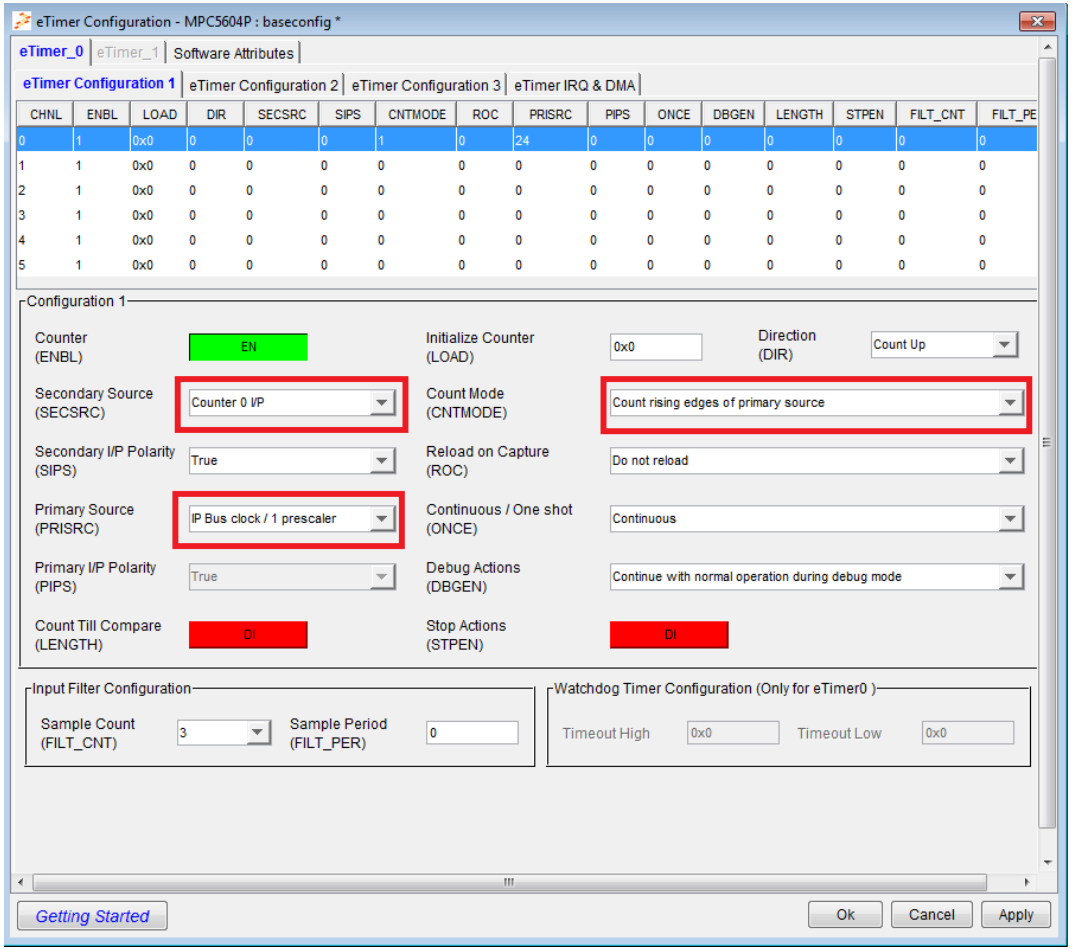
Configure eTimer

In this example, we will use channel 0 of the eTimer for measuring the frequency and duty cycle. The function uses the capture functionality of the eTimer.
The capture 1 register is set for capture the counter value on rising edge of signal and the capture 2 register is set for capture the counter value on falling edge of input signal.
The capture registers have two-deep FIFO so they are able to capture two values.



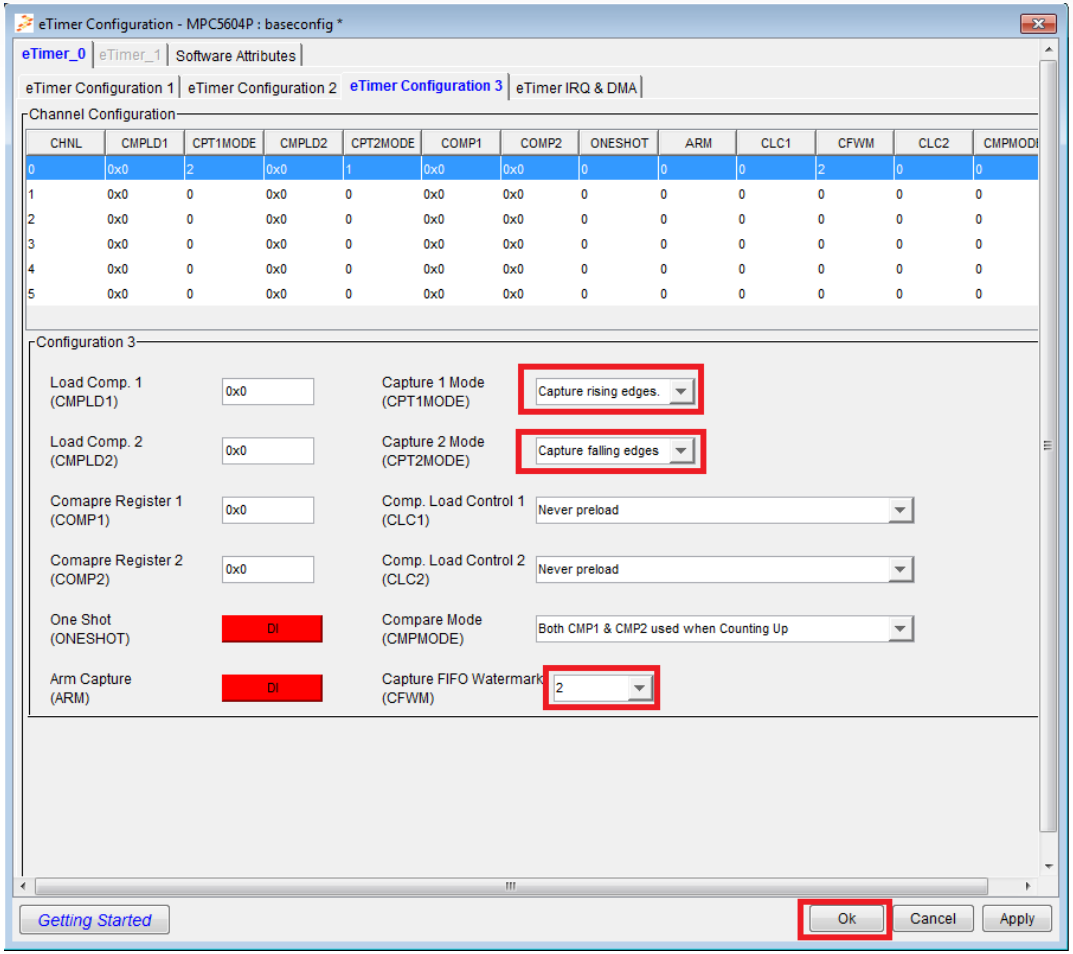
Configure eTimer

The Motor Control clock is used as the primary source of clock and the input signal as secondary source. The counter is counting repeatedly the primary source and captures its values on edges produced by the secondary source or input.
To accomplish this, select values in eTimer Configuration 1 tab as shown in the picture.



Configure eTimer

The capture 1 register is set for capture the counter value on rising edge of signal and the capture 2 register is set for capture the counter value on falling edge of input signal. The capture registers have two-deep FIFO so they are able to capture two values.
To accomplish this, select values in eTimer Configuration 3 tab as shown in the picture.



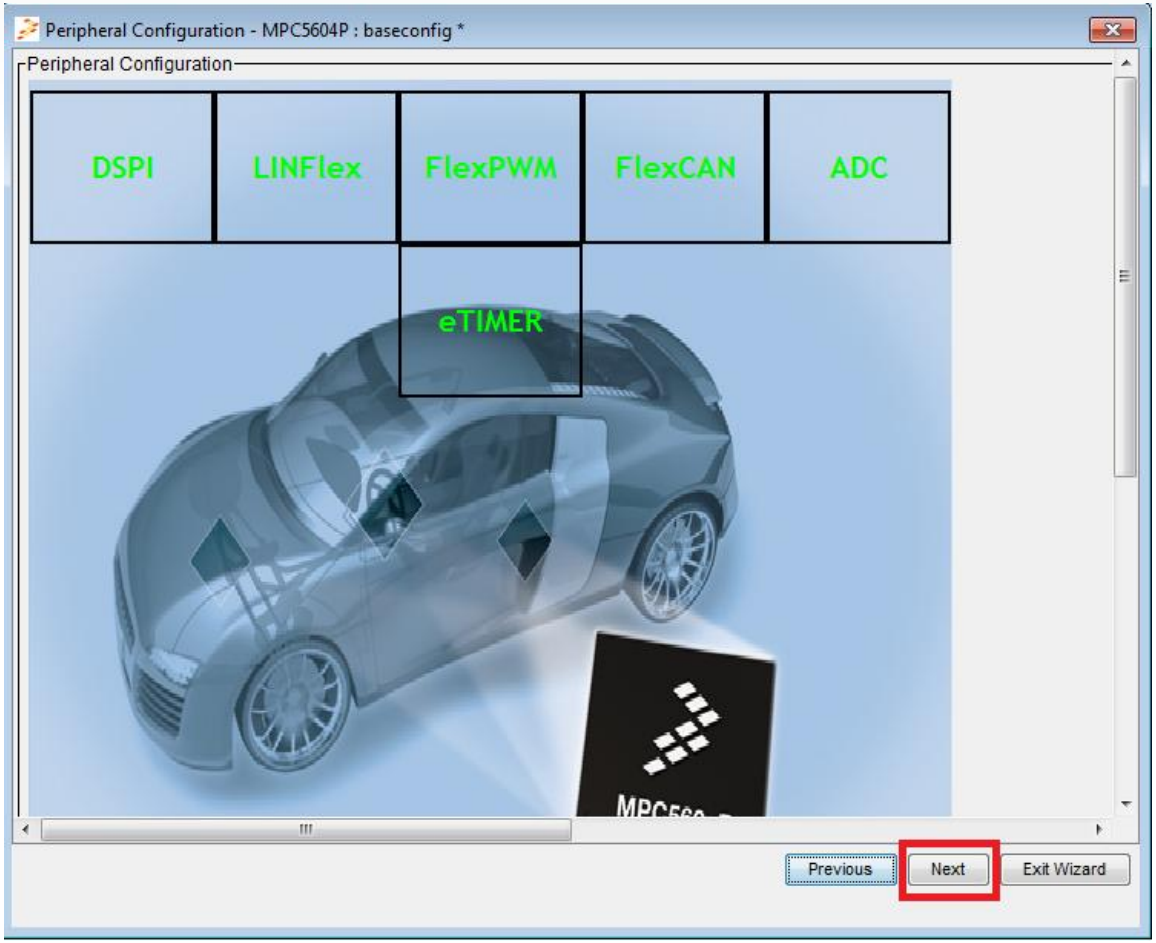
Configure eTimer

The frequency of the input PWM signal can be calculated as follows:
f [kHz] = motc_clk [kHz]/(CAPT1.R[1] - CAPT1.R[0])

The duty cycle of the input PWM signal can be calculated as follows:
*duty cycle= ((CAPT2.R[0] - CAPT1.R[0])*100)/(CAPT1.R[1] - CAPT1.R[0])*

Configure PIT Interrupt

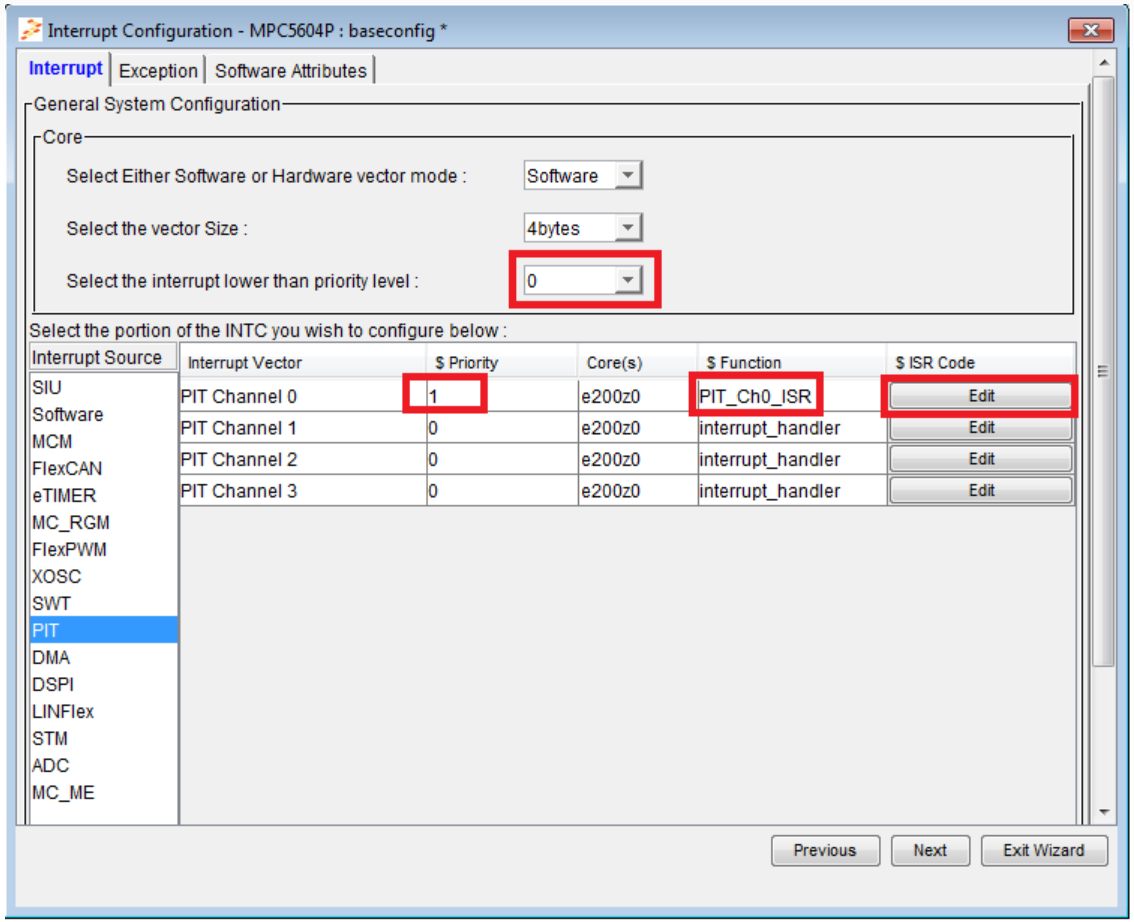
This completes Peripheral configuration.
Select *Next* to configure PIT interrupt.



Configure PIT Interrupt

In this example, PIT Channel 0 interrupt is used to monitor the state of input switches S3 and S4.

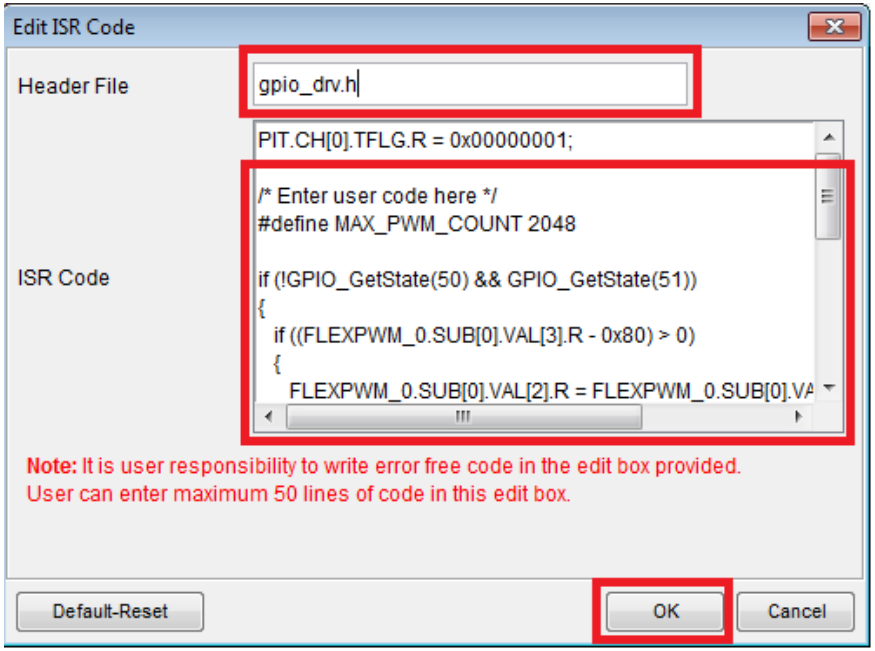
Configure PIT Ch0 interrupt as shown.
Click on PIT Channel 0 Edit button to add code to ISR function.



Configure PIT Interrupt

Add the following code to ISR to decrease Flex PWM duty cycle when S3 is pressed and increase the duty cycle when S4 is pressed.

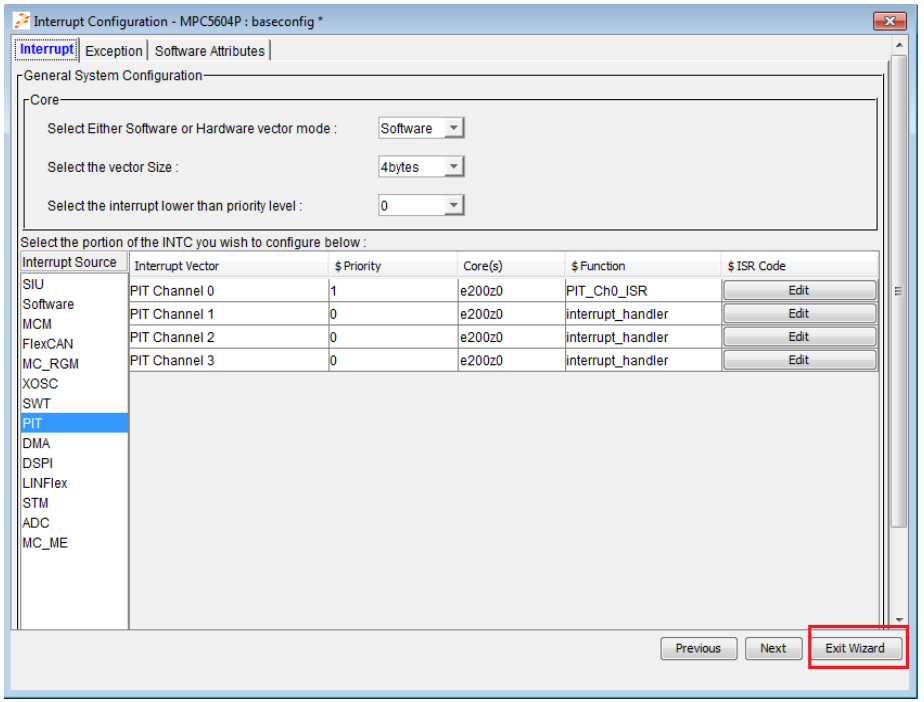
```
/* Enter user code here */
#define MAX_PWM_COUNT 2048

if (!GPIO_GetState(50) && GPIO_GetState(51))
{
  if ((FLEXPWM_0.SUB[0].VAL[3].R - 0x80) > 0)
  {
    FLEXPWM_0.SUB[0].VAL[2].R = FLEXPWM_0.SUB[0].VAL[2].R + 0x80;
   FLEXPWM_0.SUB[0].VAL[3].R = FLEXPWM_0.SUB[0].VAL[3].R - 0x80;
   FLEXPWM_0.MCTRL.B.LDOK = 1;
  }
}
if (GPIO_GetState(50) && !GPIO_GetState(51))
{
  if ((FLEXPWM_0.SUB[0].VAL[3].R + 0x80) < MAX_PWM_COUNT)
  {
     FLEXPWM_0.SUB[0].VAL[2].R = FLEXPWM_0.SUB[0].VAL[2].R - 0x80;
     FLEXPWM_0.SUB[0].VAL[3].R = FLEXPWM_0.SUB[0].VAL[3].R + 0x80;
     FLEXPWM_0.MCTRL.B.LDOK = 1;
  }
}
```



Select OK to exit from Edit Code window.

Configure PIT Interrupt

Select Exit Wizard to exit to main RAppID window



Code Generation

- We have completed all pin and peripheral configurations required for this example project.
- Now we are ready to generate code. By default, RAppID generates code for RAM. In this example we will generate code for Flash.
- By default, RAppID generates code for all peripherals. In this example, we will select only the peripherals that are used in the example for code generation.

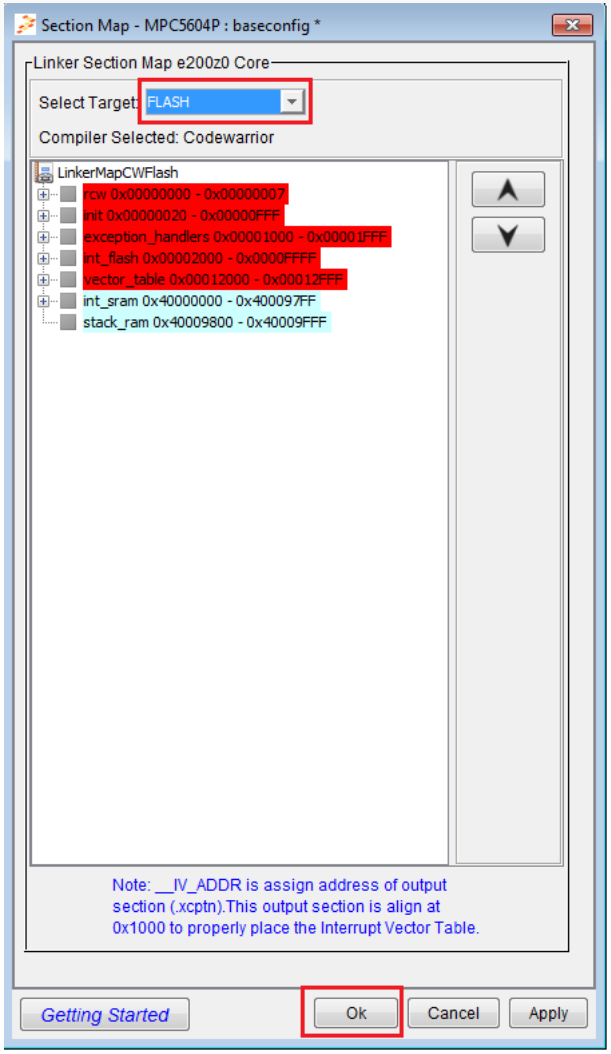
Configure Section Map

By Default, RAppID is configured to generate code for RAM. For this example, we will change the configuration to generate code for Flash.

From main RAppID window, select menu *View->View Section Map*.

Change Target to FLASH

Select Ok



Configure Code Generation

From main RAppID window, select menu *Configuration->Code Generation*.

Unselect code generation option for unused peripherals as shown.

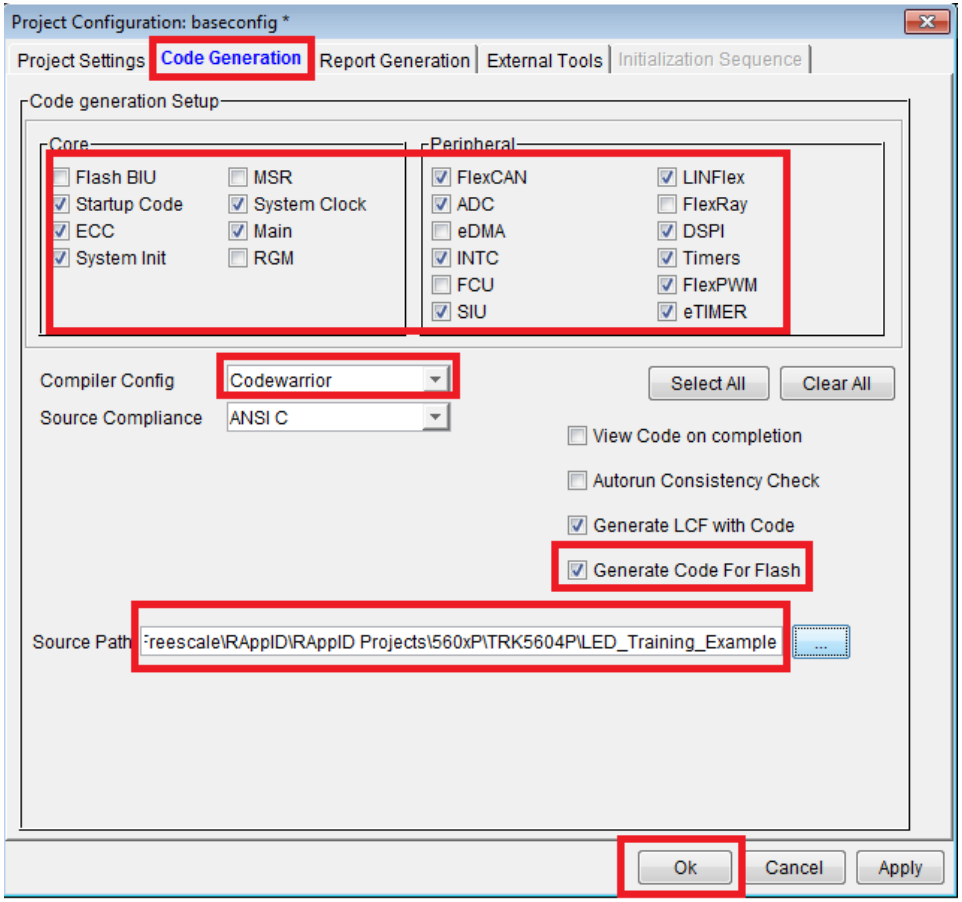
Check ECC for code generation. This is required to generate initialization code for SRAM and ECC register.

Select CodeWarrior compiler

Select Code for Flash option

Select the directory where code should be generated

Select Ok

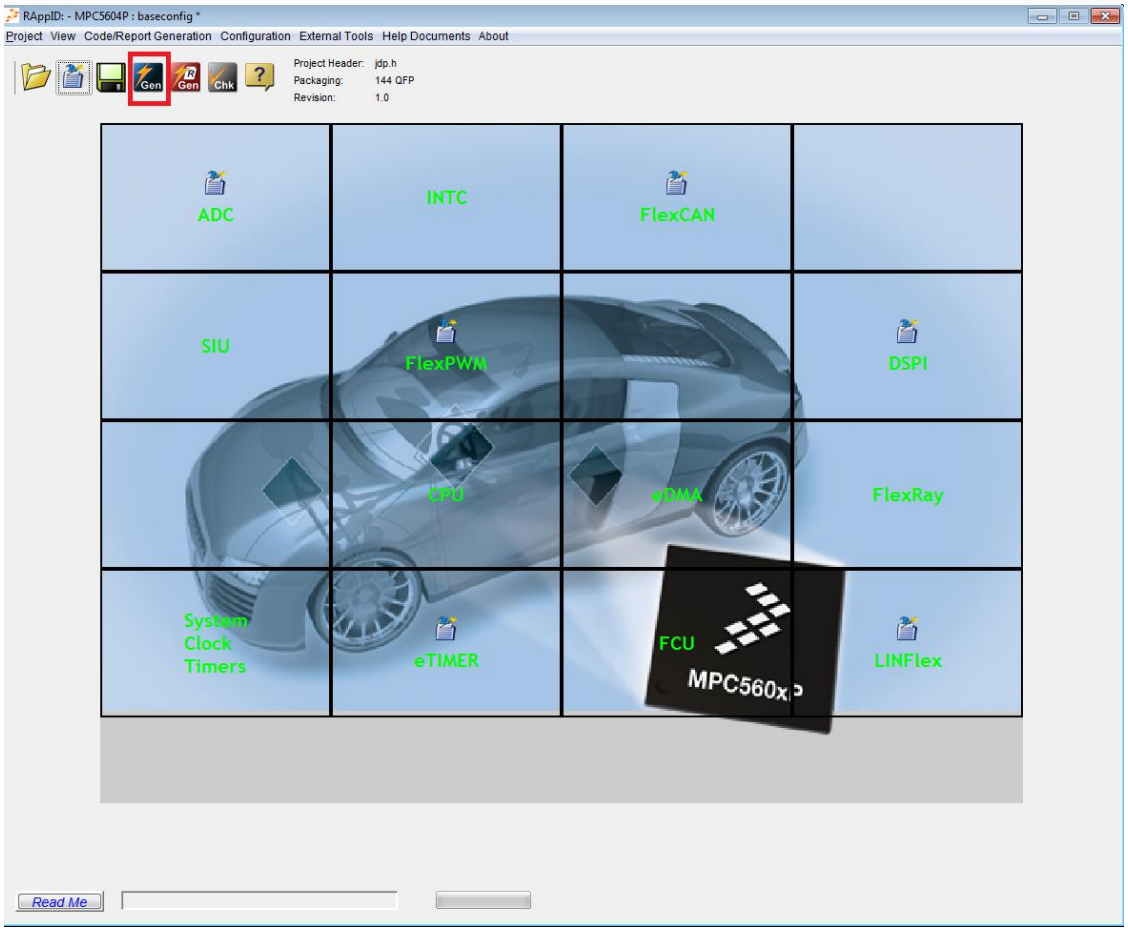


Generate code

Select Code Generation icon to generate code

Save the project when prompted

Now we are ready to build and run the project.



Report Generation

- RAppID init tool can generate a comprehensive report on the project
- Since we have finished with configuration for this example, we are ready to generate a report for this project.

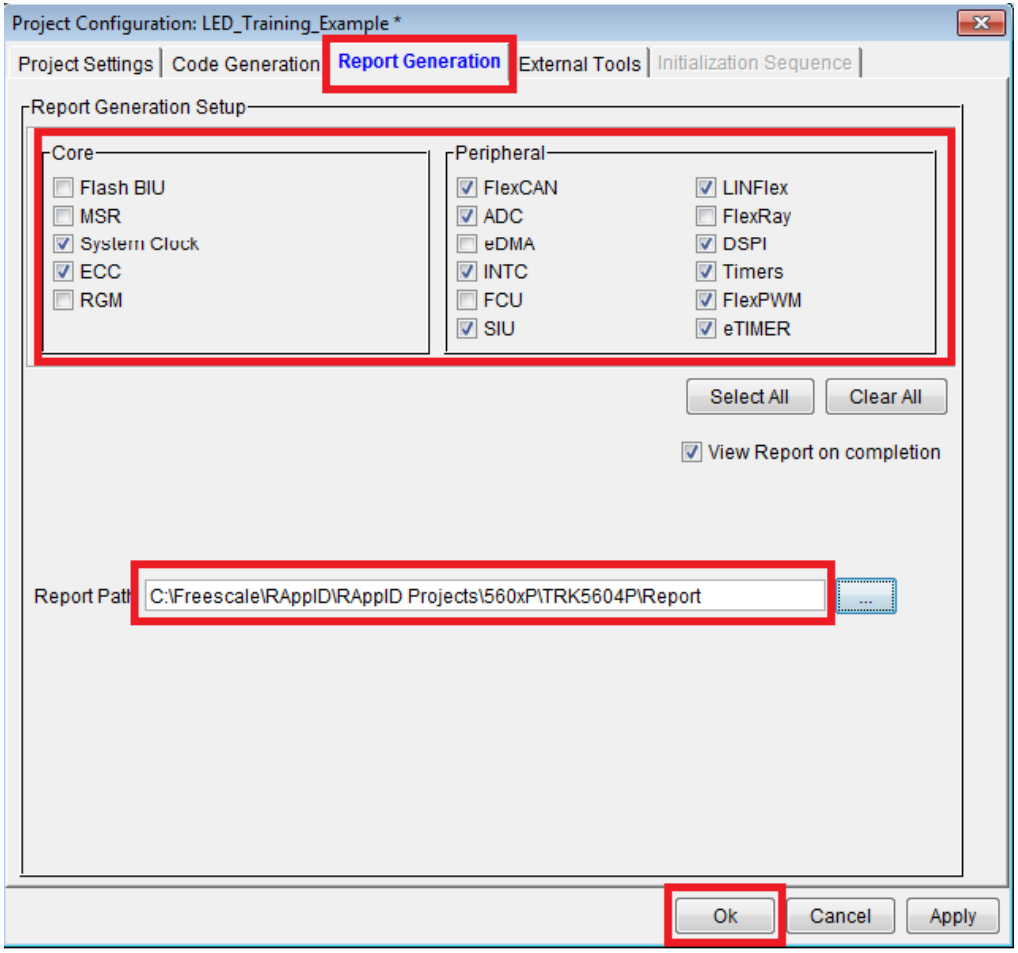
Configure Report Generation

From main RAppID window, select menu Configuration->Report Generation.

Unselect report generation option for unused peripherals

Select the directory where report should be generated

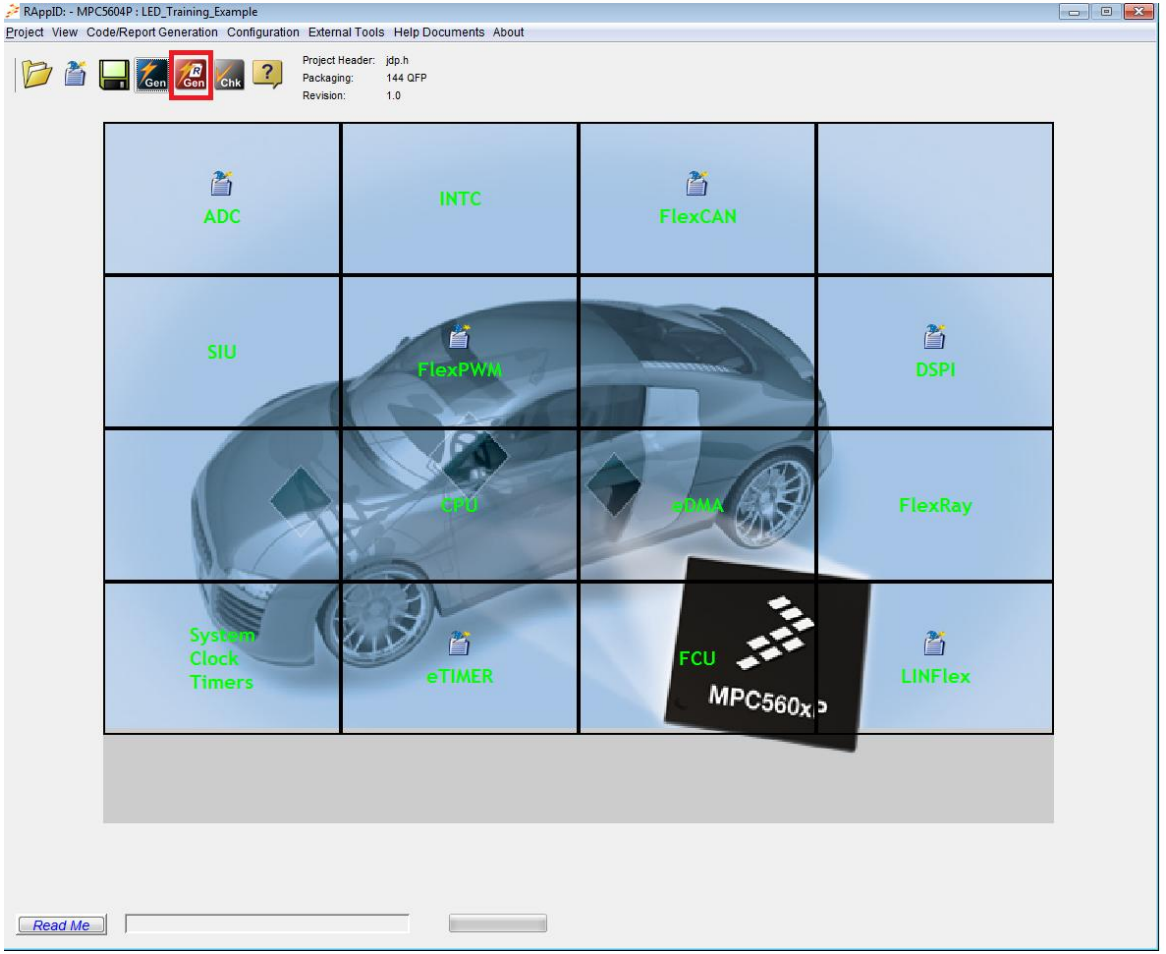
Select Ok



Generate Report

Select Report Generation icon to generate report

Save the project when prompted



RAppID Init Project Report

- RAppID tool generates a detailed report of the project that covers
  - Initialization Technique, both system and peripheral
  - Detailed Pin Allocation report
  - Section Map report
  - A detailed report of configuration of each peripheral

Project Report – Initialization Technique

Configuration Report for MPC5604P
Package: 144 QFP

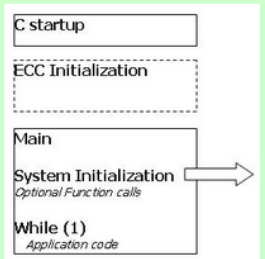
freescale semiconductor

Project Name: baseconfig (Revision :1.0)
Compiler Compliancy: Codewarrior, ANSI C
Note: *RApplD bit description is in Motorola (Big Endian) format.*

Last Save Date: Fri, 04 May 2012 02:50:28
Tool Version: 1.4.0.4

Device Settings for: Initialization Technique

System Main Initialization



System Initialization (e200z0 Core)
- INTC Initialization
- RGM Initialization
- SWT Initialization
- MSR Initialization
- Mode Entry Initialization
- System Clock Initialization
- CMU Initialization
- STM Initialization
- PIT Initialization
- FLASH BIU Initialization
- EDMA Initialization
- FlexCAN Initialization
- FlexPWM Initialization
- ADC Initialization
- eTIMER Initialization
- DSPI Initialization
- FlexRay Initialization
- LINFlex Initialization
- FCU Initialization
- Mode Entry Post Configuration Init
- SIU Initialization

Peripheral Initialization

Project Report – Pin Allocation

Configuration Report for MPC5604P
Package: 144 QFP

freescale semiconductor

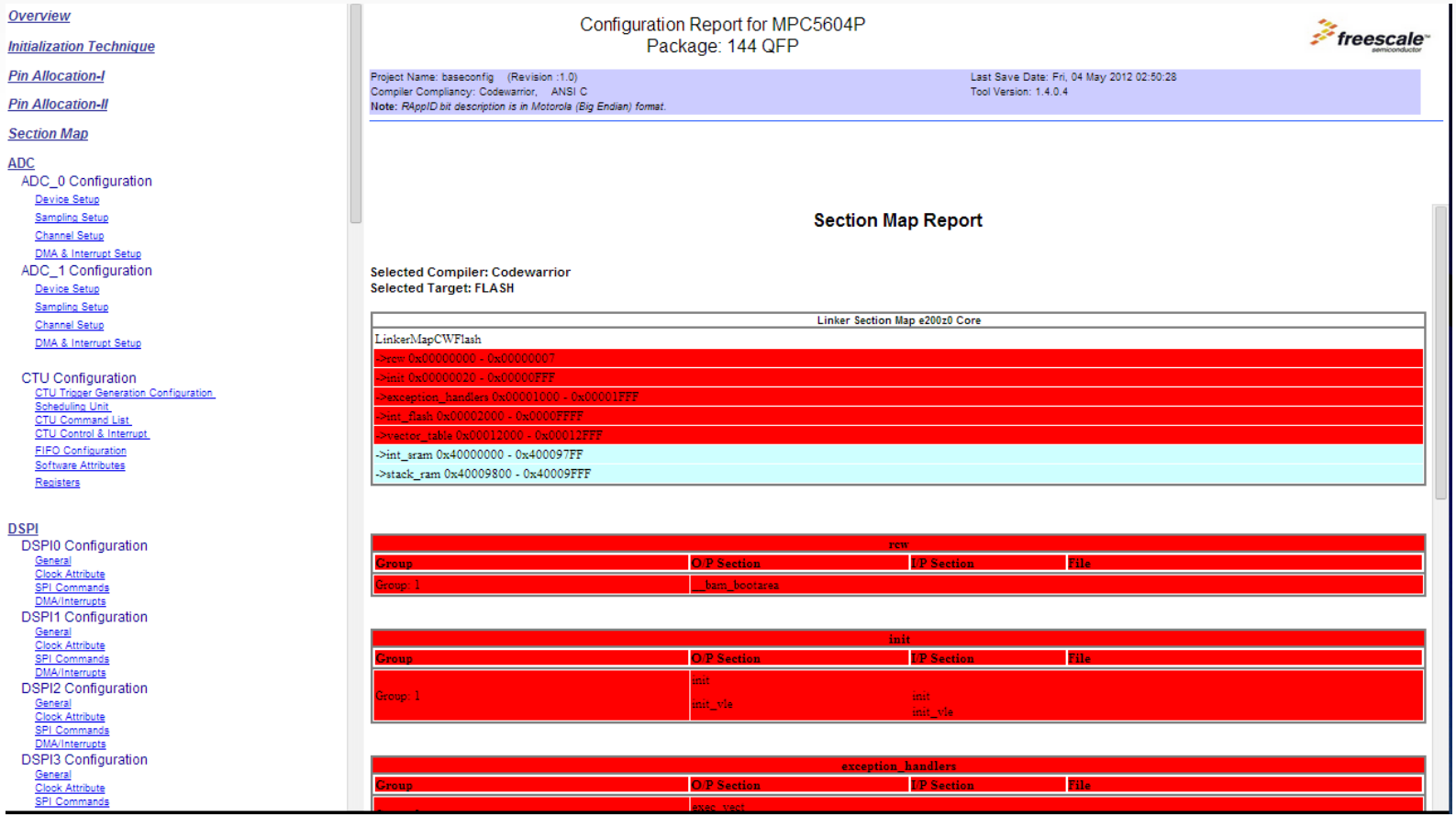
Project Name: baseconfig (Revision :1.0)
Compiler Compliancy: Codewarrior, ANSI C
Note: *RApplD bit description is in Motorola (Big Endian) format.*

Last Save Date: Fri, 04 May 2012 02:50:28
Tool Version: 1.4.0.4

Function Assignment for MPC5604P 144 QFP package

			Primary Function		Alternate Function 1		Alternate Function 2		Alternate Function 3		Input Function 0		Input Function 1		Analog Pad Conrol		Assigned Function		
SIU PCR#	Package Pin #	Port Name	GPIO (PA=3'b000)	dir	ALT 1 (PA=3'b001)	dir	ALT 2 (PA=3'b010)	dir	ALT 3 (PA=3'b011)	dir	IN 0	dir	IN 1	dir	APC	dir	Assigned Function	dir	User Assigned Signal Name
0	73	PA[0]	GPIO	IO	ETC_0[0]	IO	SCK_2	IO	FCU[0]	O							GPIO	I	
1	74	PA[1]	GPIO	IO	ETC_0[1]	IO	SOUT_2	O	FCU[1]	O							ETC_0[1] I	I	PA1_eTimer_Input
2	84	PA[2]	GPIO	IO	ETC_0[2]	IO			PWM_A[3]	IO	SIN_2	I					GPIO	I	
3	92	PA[3]	GPIO	IO	ETC_0[3]	IO	CS0_2	IO	PWM_B[3]	IO							GPIO	I	
4	108	PA[4]	GPIO	IO	ETC_1[0]	IO	CS1_2	O	ETC_0[4]	IO							GPIO	I	
5	14	PA[5]	GPIO	IO	CS0_1	IO	ETC_1[5]	IO	CS7_0	O							GPIO	I	
6	2	PA[6]	GPIO	IO	SCK_1	IO											GPIO	I	
7	10	PA[7]	GPIO	IO	SOUT_1	O											GPIO	I	
8	12	PA[8]	GPIO	IO							SIN_1	I					GPIO	I	
9	134	PA[9]	GPIO	IO	CS1_2	O			PWM_B[3]	IO	FAULT[0]	I					GPIO	I	
10	118	PA[10]	GPIO	IO	CS0_2	IO	PWM_B[0]	IO	PWM_X[2]	IO							GPIO	I	
11	120	PA[11]	GPIO	IO	SCK_2	IO	PWM_A[0]	IO	PWM_A[2]	IO							GPIO	I	
12	122	PA[12]	GPIO	IO	SOUT_2	O	PWM_A[2]	IO	PWM_B[2]	IO							GPIO	I	
13	136	PA[13]	GPIO	IO			PWM_B[2]	IO			SIN_2	I	FAULT[0]	I			GPIO	I	
14	143	PA[14]	GPIO	IO	SP_TXD	O	ETC_1[4]	IO									GPIO	I	
15	144	PA[15]	GPIO	IO			ETC_1[4]	IO			SP_RXD	I					GPIO	I	
16	109	PB[0]	GPIO	IO	CNTXD_0	O	ETC_1[2]	IO	DEBUG[0]	O							CNTXD_0	O	PB0_CAN0_TX
17	110	PB[1]	GPIO	IO			ETC_1[3]	IO	DEBUG[1]	O	CNRXD_0	I					CNRXD_0	I	PB1_CAN0_RX

Project Report – Section Map



Overview
Initialization Technique
Pin Allocation-I
Pin Allocation-II
Section Map
ADC
ADC_0 Configuration
Device Setup
Sampling Setup
Channel Setup
DMA & Interrupt Setup
ADC_1 Configuration
Device Setup
Sampling Setup
Channel Setup
DMA & Interrupt Setup
CTU Configuration
CTU Trigger Generation Configuration
Scheduling Unit
CTU Command List
CTU Control & Interrupt
FIFO Configuration
Software Attributes
Registers
DSPI
DSPI0 Configuration
General
Clock Attribute
SPI Commands
DMA/Interrupts
DSPI1 Configuration
General
Clock Attribute
SPI Commands
DMA/Interrupts
DSPI2 Configuration
General
Clock Attribute
SPI Commands
DMA/Interrupts
DSPI3 Configuration
General
Clock Attribute
SPI Commands

Configuration Report for MPC5604P
Package: 144 QFP

freescale semiconductor

Project Name: baseconfig (Revision :1.0)
Compiler Compliancy: Codewarrior, ANSI C
Note: RAppID bit description is in Motorola (Big Endian) format.
Last Save Date: Fri, 04 May 2012 02:50:28
Tool Version: 1.4.0.4

Section Map Report

Selected Compiler: Codewarrior
Selected Target: FLASH

Linker Section Map e200z0 Core
LinkerMapCWFlash
->rcw 0x00000000 - 0x00000007
->init 0x00000020 - 0x00000FFF
->exception_handlers 0x00001000 - 0x00001FFF
->int_flash 0x00002000 - 0x0000FFFF
->vector_table 0x00012000 - 0x00012FFF
->int_sram 0x40000000 - 0x400097FF
->stack_ram 0x40009800 - 0x40009FFF

rcw			
Group	O/P Section	I/P Section	File
Group: 1	__ram_bootarea		

init			
Group	O/P Section	I/P Section	File
Group: 1	init init_vle	init init_vle	

exception_handlers			
Group	O/P Section	I/P Section	File
	exec_vect		

Project Report – Peripheral Configuration

Configuration Report for MPC5604P
Package: 144 QFP

freescale semiconductor

Project Name: baseconfig (Revision :1.0)
Compiler Compliancy: Codewarrior, ANSI C
Note: *RAppID bit description is in Motorola (Big Endian) format.*

Last Save Date: Fri, 04 May 2012 02:50:28
Tool Version: 1.4.0.4

Device Settings for: ADC_0

ADC General Setup Information

General Device Settings

Normal Conversion will not start when external Start Signal is detected. Clock will not automatically switch off when ADC is idle.

Conversion of ADC command shall be one channel chain at a time. After conversion data is right allign also old conversion data will be not overwritten by newer result, hence newer result will be discarded.

Delay between extrenal decode signal and start of sampling pulse is set to 0 clock cycles. Delay between extrenal power signal and start of sampling pulse is set to 0 clock cycles.

ADC Clock selected as System Clock/2.

Cross Triggering unit Disabled.

Triggered Injection is Disabled. with Trigger Injection edge selected as Falling.

Register

ADC_0_MCR

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OWR EN	WL SIDE	MODE	EDG LEV	TRGEN	EDGE	XSTRT EN	N START	Res.	JTRG EN	JEDGE	J START	Reserved		CTUEN	Res.
rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw			rw	

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Reserved							ADCLK SEL	ABORT CHAIN	ABORT	ACKO	OFF RE FRESH	OFF CANC	Reserved		PWDN

Adding Low and high level Driver code

- We will use driver code supplied with the installation media to perform low level functions. Copy the driver code to the directory where RAppID code was generated

Adding FreeMASTER code

- We will use FreeMASTER utility to monitor global variables. To add this ability, we have to add FreeMASTER code which is provided with the installation media.
- 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.

  - Rename *freemaster_cfg.h.example* file to *freemaster_cfg.h*. This file contains all the macro definitions available for the FreeMASTER configuration.

  - 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
```

Edit main.c

- The main.c generated by RAppID initializes peripherals and includes an empty while loop. We have to add code to main.c to:
  - Turn on LED1 when Switch S1 is pressed and turn off when S1 is not pressed
  - Turn on LED2 when potentiometer input value is <=512 otherwise turn off LED2
  - Turn on LED3 when command ‘1’ is sent via CAN and turn off LED3 when ‘0’ is sent
  - Add FreeMASTER supporting code
  - Add code to calculate frequency and duty cycle of eTimer channel 0 PWM input signal

Edit main.c – Global variables

Add following global variables to main.c:

```
/* 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};
vuint16_t potValue;/* Potentiometer input value */
vuint16_t switchState;  /* State of input switch S1 */
vuint32_t PWMDutyCycle; /* Duty cycle of input PWM signal */
vuint32_t PWMFreq;      /* Frequency of input PWM signal */
```

Note: This example CodeWarrior project along with source code is provided with the installation disk. You can use Import project option to create this example project within CodeWarrior.

Edit main function

Add following include files:

```
#include "freemaster.h"
#include "pot_hld.h"
#include "sbc_hld.h"
#include "CANapi.h"
#include "gpio_drv.h"
```

Add following code to *main* function:

Add a call to driver to initialize SBC to enable CAN communication:

```
SBC_Init_DBG();
```

The 4 LEDs are active low and they turn on at start up. Turn them off:

```
GPIO_SetState(52, 1);
GPIO_SetState(53, 1);
GPIO_SetState(54, 1);
GPIO_SetState(55, 1);
```

Edit main function (Continued)

We will receive CAN messages with CAN ID = 1. Add a call to CAN driver to setup mailbox 0 with Id = 1:

SetCanRxFilter(1, 0, 0);

Call FreeMASTER internal variables initialization function:

FMSTR_Init();

Edit main function (Continued)

Within while loop, add function calls to process FreeMASTER, GPIO, ADC, CAN and duty cycle calculation.

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

Add ProcessCAN function

We will add function *ProcessCAN* to process messages received via CAN. This function will call the CAN driver functions to check if any CAN message is received. If a CAN message is received with first data byte value of 1, it will turn On LED3 and if the first data byte value is 0, it will turn Off LED3.

If the first data byte is either 0 or 1, it will transmit CAN message ***msgOKCAN***, other wise it will transmit CAN message ***msgErrorCAN***. The code is shown below:

Add ProcessCAN function (Continued)

```
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 received data byte is 0, turn off LED3 and send positive response */
       {
                GPIO_SetState(54, 1);
                CanTxMsg (2, 1, 8, (uint8_t *)msgOKCAN, 0);
       }
       else if (msgCanRX.data[0] == 1)    /* If received data byte is 1, turn on LED3 and send positive response */
       {
                GPIO_SetState(54, 0);
                CanTxMsg (2, 1, 8, (uint8_t *)msgOKCAN, 0);
       }
```

Add ProcessCAN function (Continued)

```
        else                    /* If received data byte is not 0 or 1, send a negative response */
        {
                CanTxMsg (2, 1, 8, (uint8_t *)msgErrorCAN, 0);
        }
    }
}
```

Add ProcessADC function

We will add function *ProcessADC* to process potentiometer input. If potentiometer value is <=512, it will turn on LED2, otherwise turn off LED2. The code is shown below:

```
void ProcessADC(void)
{
    potValue = Pot_Get_Value();
    if(potValue <= 512)       /* If Potentiometer input is <= 512 turn on LED2, other wise turn off LED2 */
    {
            GPIO_SetState(53, 0);
    }
    else
    {
            GPIO_SetState(53, 1);
    }
}
```

Add ProcessGPIO function

We will add function *ProcessGPIO* to process input switch S1. If switch S1 is pressed, it will turn On LED1, otherwise turn Off LED1. The code is shown below:

```
void ProcessGPIO(void)
{
    switchState = GPIO_GetState(48); /* Check switch S1 state */
    if (!switchState)          /* If Switch S1 is pressed, turn on LED1, other wise turn off LED1*/
    {
            GPIO_SetState(52, 0);
    }
    else
    {
            GPIO_SetState(52, 1);
    }
}
```

Add CalcDC function

We will add function *CalcDC* to calculate duty cycle and frequency of the PWM signal captured by eTimer channel 0. The code is shown below:

```
void CalcDC(void)
{
        vuint32_t  PWMPeriod;
        vuint16_t measure[4];

        ETIMER_0.CHANNEL[0].CCCTRL.B.ARM = 1;
        if (ETIMER_0.CHANNEL[0].STS.B.ICF1 == 0 || ETIMER_0.CHANNEL[0].STS.B.ICF2 == 0)
        {
                return;
        }
        ETIMER_0.CHANNEL[0].CCCTRL.B.ARM = 0;
        ETIMER_0.CHANNEL[0].STS.B.ICF1 = 0x1; //clear capture 1 flag
        ETIMER_0.CHANNEL[0].STS.B.ICF2 = 0x1; //clear capture 2 flag
        measure[0] = ETIMER_0.CHANNEL[0].CAPT1.R; //read first capture1 value
        measure[1] = ETIMER_0.CHANNEL[0].CAPT1.R; //read second capture 1 value
        measure[2] = ETIMER_0.CHANNEL[0].CAPT2.R; //read first capture2 value
        measure[3] = ETIMER_0.CHANNEL[0].CAPT2.R; //read second capture2 value
```

Add CalcDC function (Continued)

```
        PWMPeriod = (vuint32_t) (measure[1] - measure[0]);
        if (PWMPeriod != 0)
        {
                PWMDutyCycle = (vuint32_t)((measure[2] - measure[0]))*100/ (vuint32_t) (measure[1] -
measure[0]); //PWMOnTime/PWMPeriod;
                PWMFreq = 16000000/PWMPeriod;
        }
}
```

Create CodeWarrior project

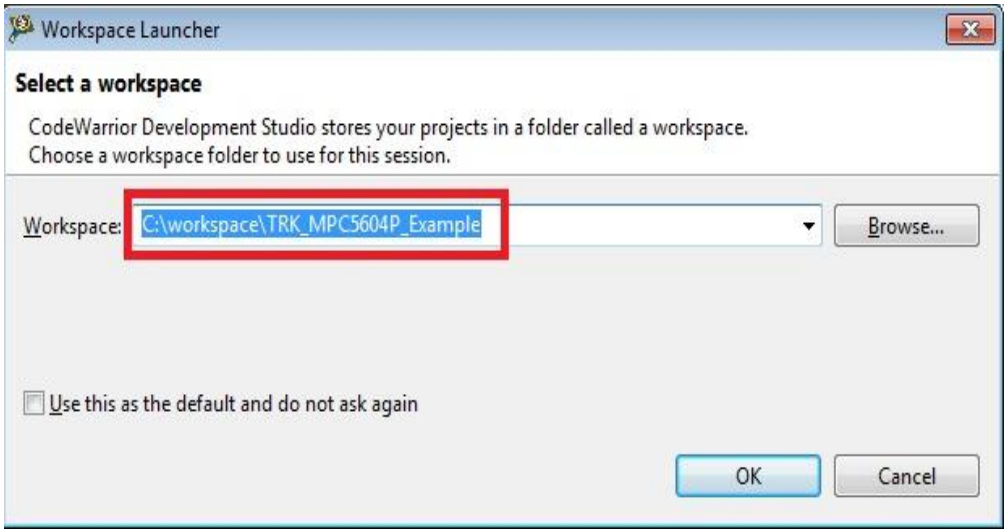
- Now we are ready to compile and build the code. To do this,
  - create an empty CodeWarrior project
  - add RAppID generated code and driver code to the CodeWarrior project using CodeWarrior script utility – rsp2cw10*.exe*
  - Build the code using Codewarrior
- The next few slides explains these steps

Create CodeWarrior project

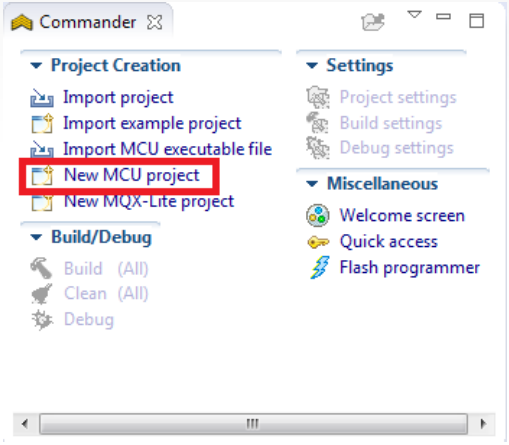
Launch CodeWarrior 10.5 from Windows Start menu

Provide a workspace name

Select OK



Create CodeWarrior project

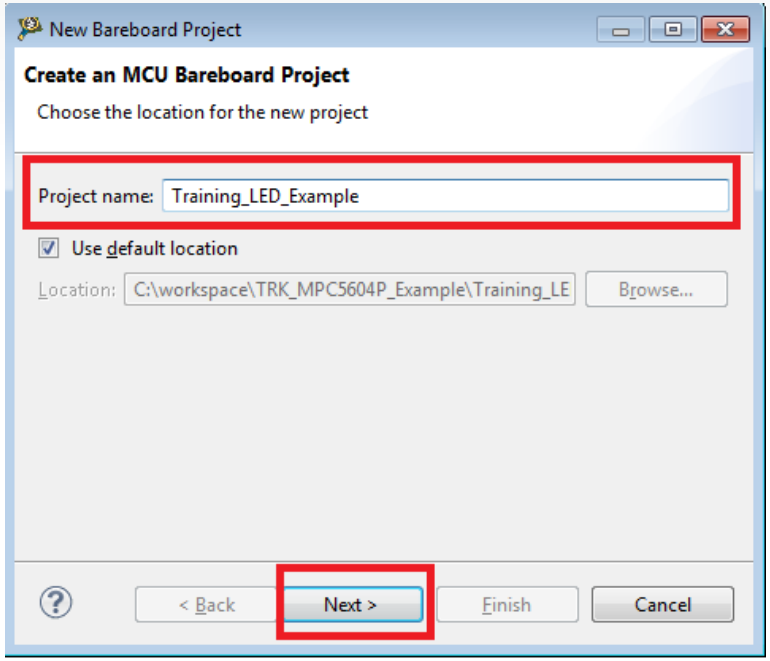


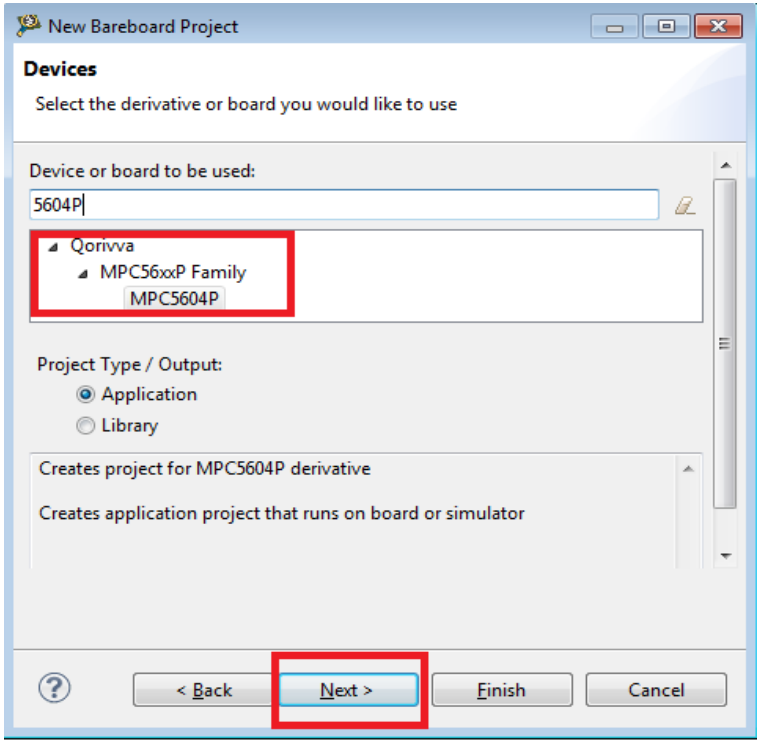
We will first create a Bareboard project for MPC5604P

To create a new Bareboard project, Select “New MCU project” from Commander window.

Create CodeWarrior project

Provide a name for the project – Training_LED_Example
select Next



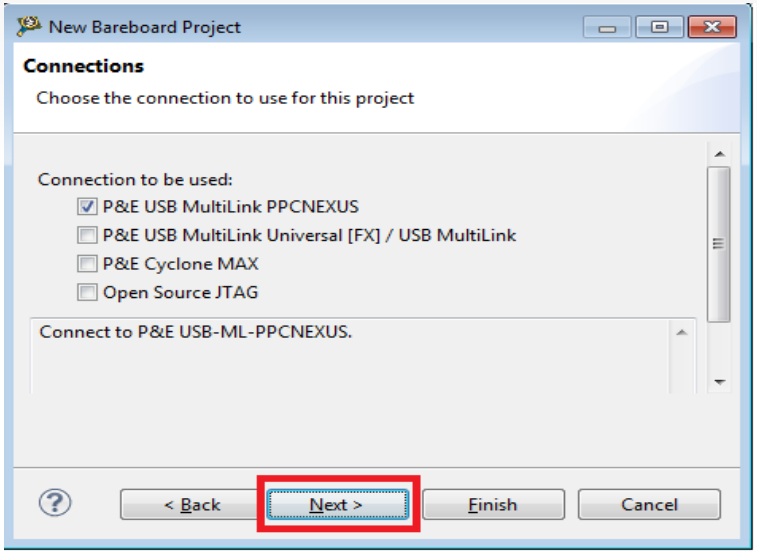


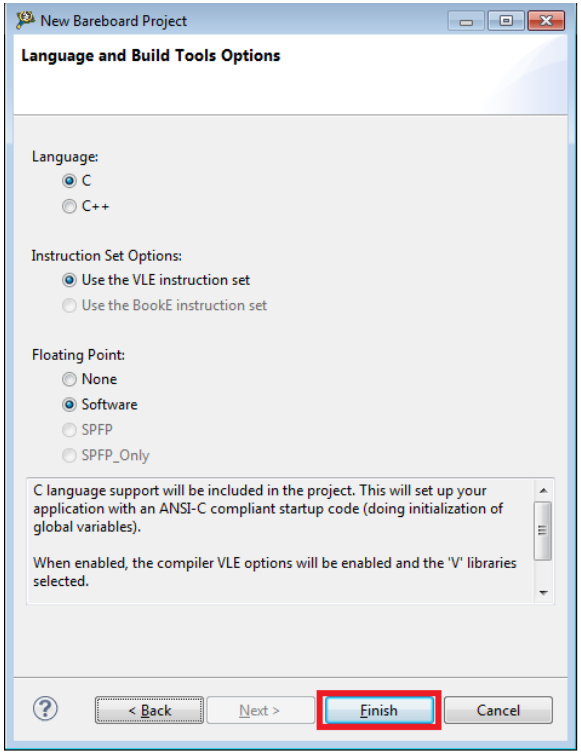
Select Qorivva->MPC560xP->MPC5604P device from the list

Select Next

Create CodeWarrior project

Accept the default connection option and select Next



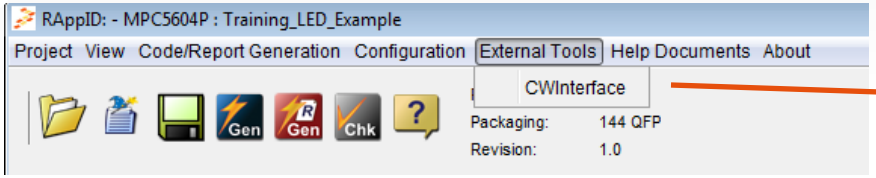


We will be using C code and VLE instructions for this example project. Accept the default options and select Finish

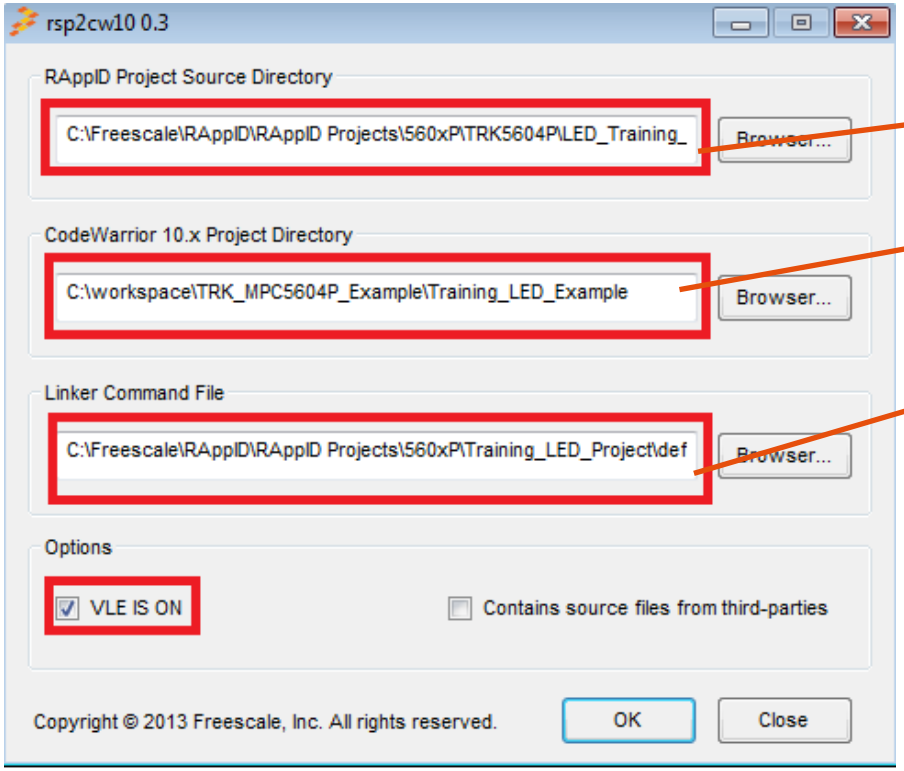
Create CodeWarrior project

- When Finish is selected, CodeWarrior creates a new bareboard project with CodeWarrior generated code and linker file included in the project.
- Since we have generated code using RAppID for Training_LED_Example project, we want to remove all the CodeWarrior generated code and add RAppID generated code and linker file along with the driver code to be included in the CodeWarrior project.
- This can be done using CodeWarrior project maker utility – *rsp2cw10.exe*
- First close the Training_LED_Example project by selecting menu Project->Close Project
- Run *rsp2cw10* utility which is provided with the installation disk: This can be done by selecting the file from the installed directory or by using the menu in RAppID - *External Tools->CWInterface*

Run cwpjmaker utility



Execute rsp2cw10 utility from RAppID menu



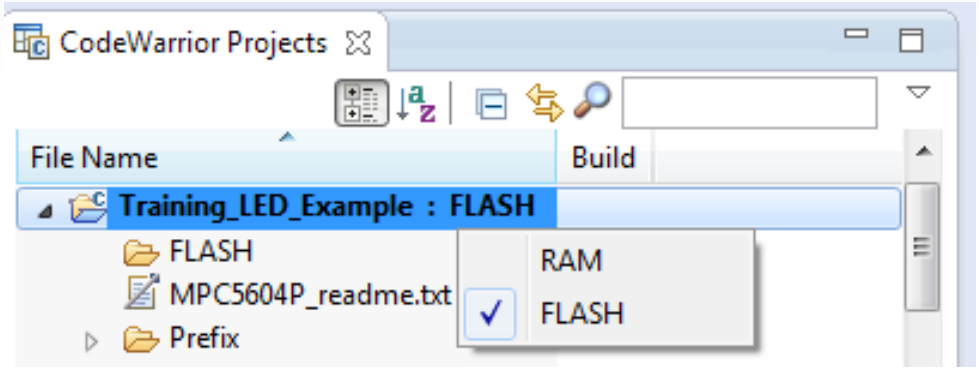
Provide directory path where RAppID code is generated

Provide directory path where CodeWarrior project is created

Provide the path of linker file created by RAppID tool

The script will remove CodeWarrior generated code from the project and will add all the code in the directory specified in the “RAppID Source Configuration” edit box.
The script will also add/modify all the CodeWarrior project settings required to build Training_LED_Example project

Create CodeWarrior project



Now re-open the Training_LED_Example project by selecting menu Project->Open Project

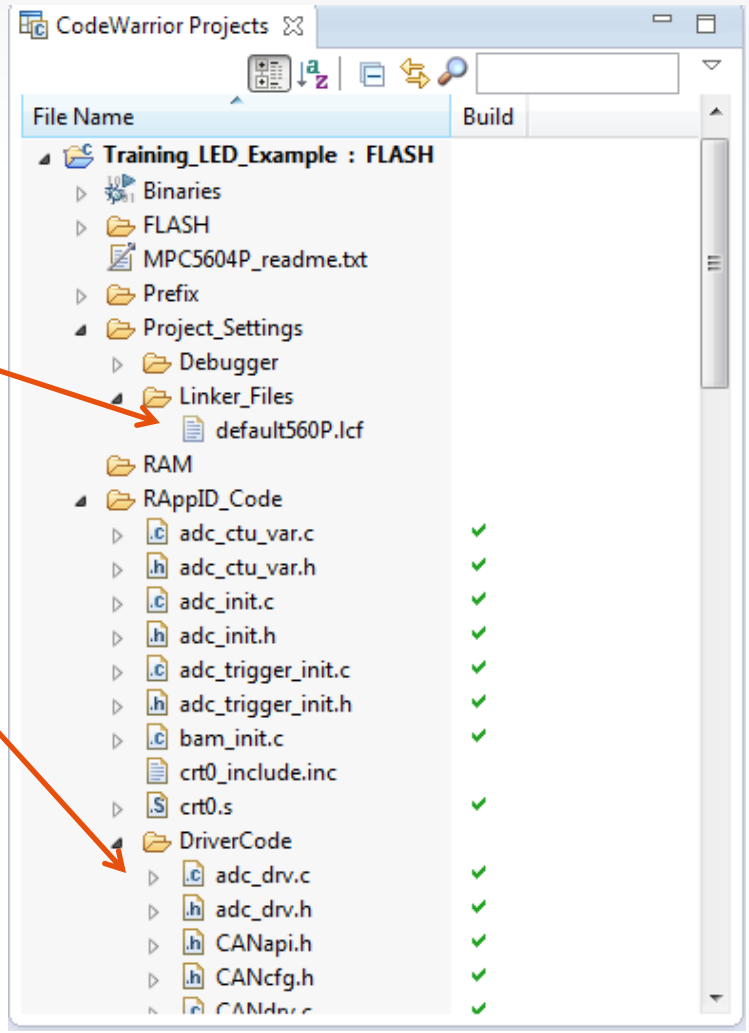
Since we generated code for Flash, we need to change the build configuration to Flash. Change the build option to Flash as shown.

Create CodeWarrior project

Now the project should contain:

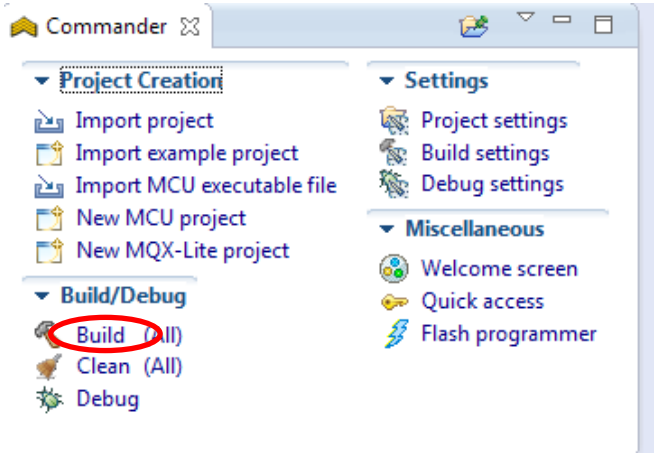
RAppID generated linker file

RAppID generated source files and driver code copied from the installation disk



Build project

To build Training_LED_Example project , select Build from Commander window

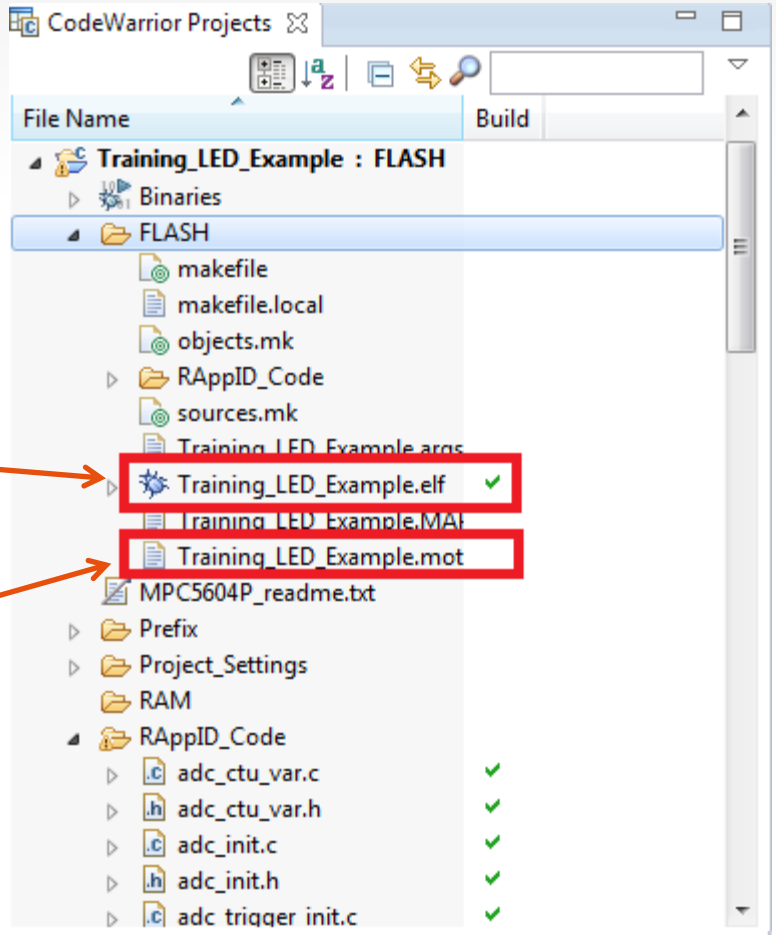


Build project

The build command will produce 2 executable files:

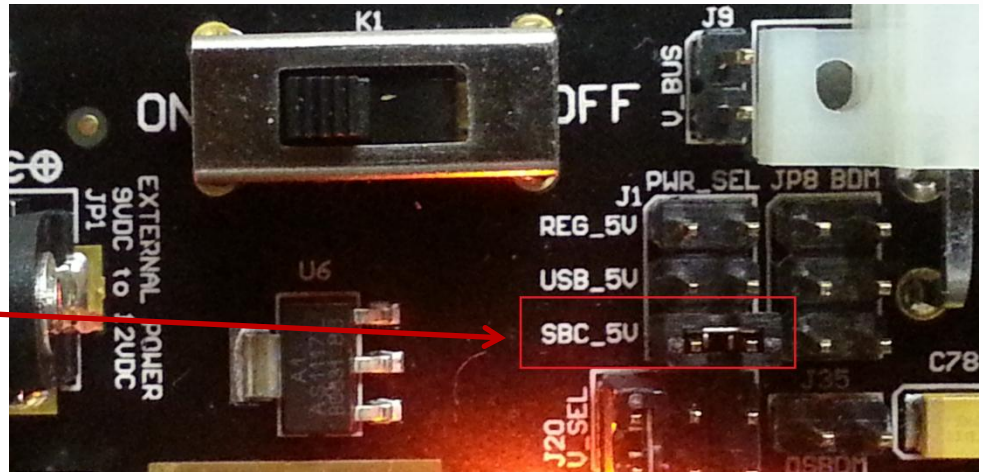
Training_LED_Example.elf (executable with debug symbols)

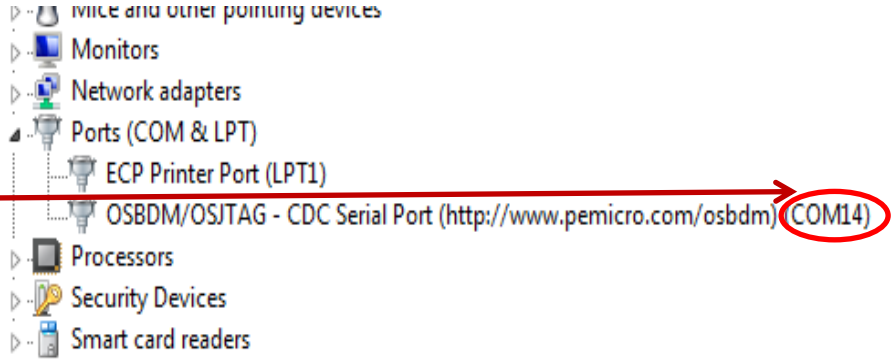
Training_LED_Example.mot (S-record)



Jumper settings and Power connections

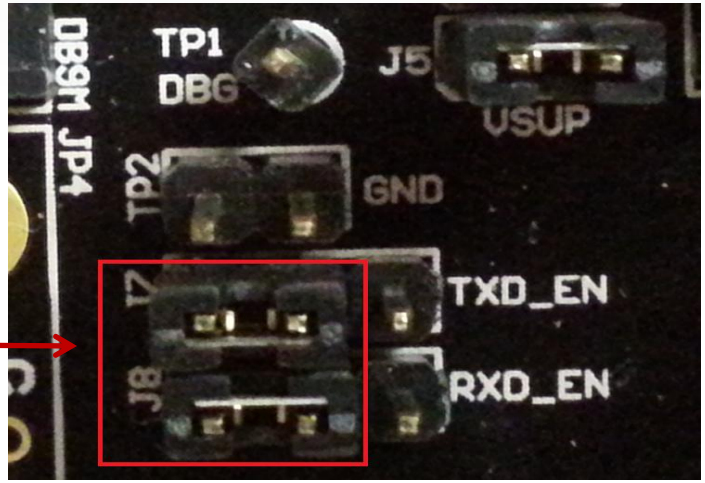
- Since we are using CAN in this example, we need to enable CAN.
- To enable CAN, the board needs to power SBC using external 12V supply.
- Connect jumpers across SBC_5V of J1
- Connect External power to JP1
- Connect your computer to JP2 via USB cable. This connection provides virtual serial port. Confirm this by checking available COM ports in Windows Device manager. In the example shown, the COM port assigned is COM14

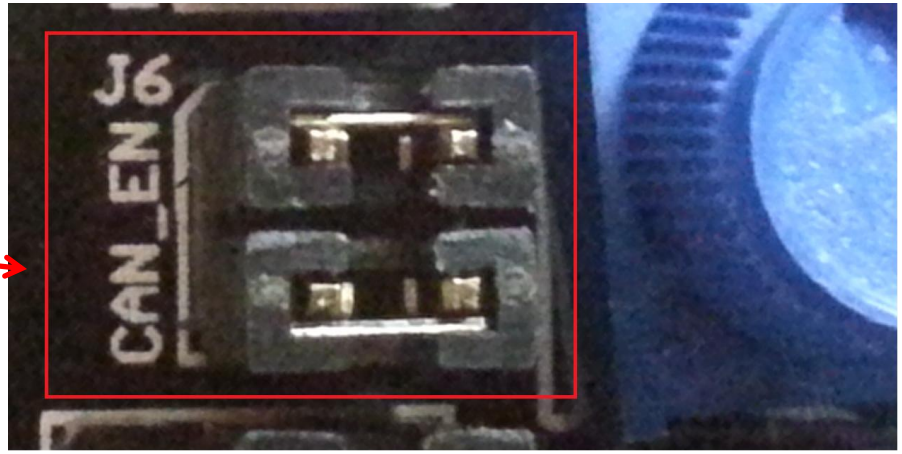




Jumper settings and Power connections

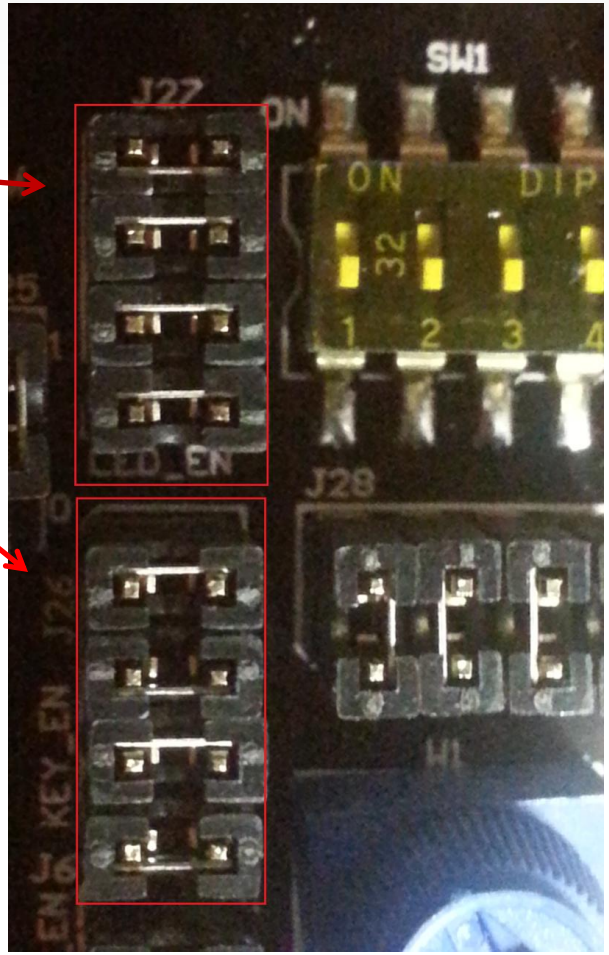
- In this example, we will use Virtual serial port for UART communication. To enable this, connect jumper J7 (TXD_EN) and J8 (RXD_EN) to position 1-2
- To enable CAN communication, connect J6 as shown.





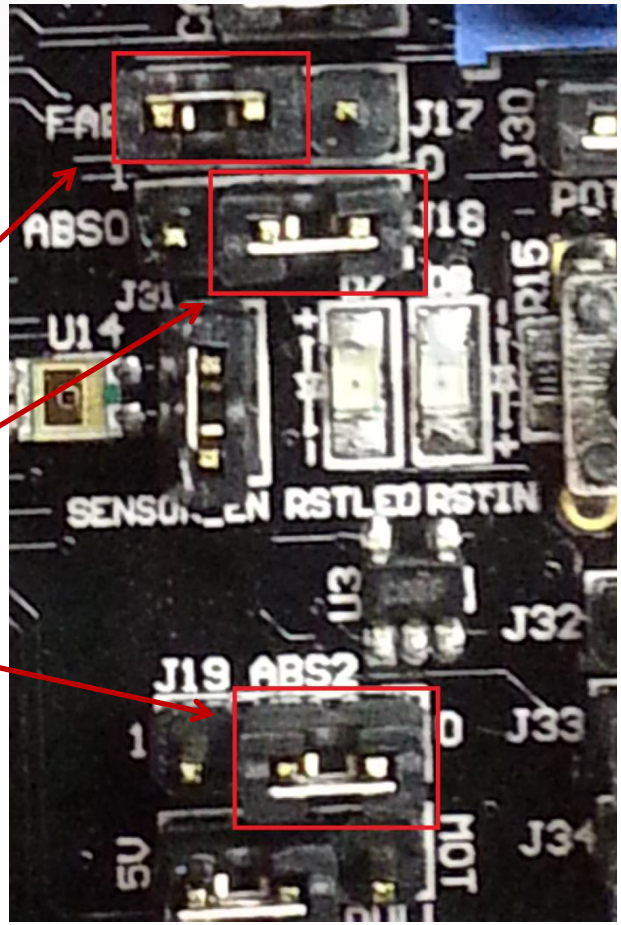
Jumper settings and Power connections

- To enable LEDs, connect all 4 jumpers in J27
- To enable input switches, connect all 4 jumpers in J26



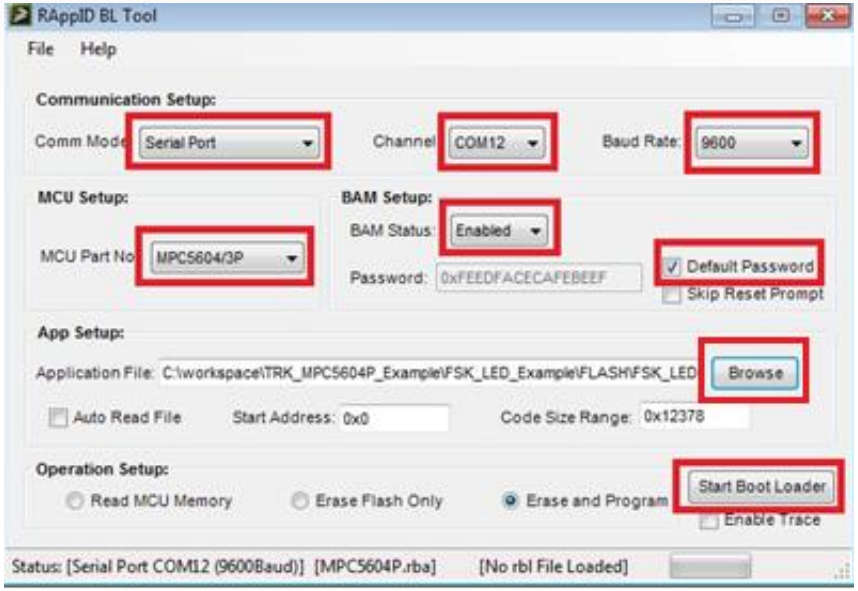
Flash code to target

- Before Flashing the code, make sure the TRK-MPC5604P board is connected to external power and to your computer via USB cable
- 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
- We will use RAppID Bootloader utility to flash S-record file Training_LED_Example.mot using serial port



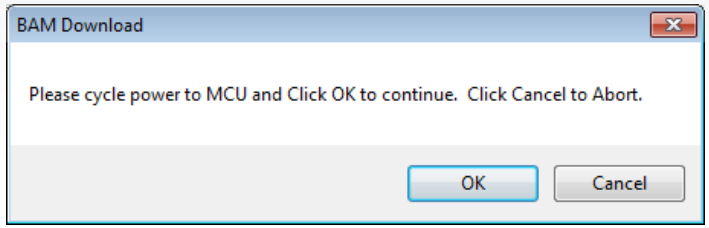
Flash code to target

- Launch RAppID bootloader utility from Windows Start menu
- Select Serial Port as Comm Mode
- Select the correct COM channel
- Select 9600 baud
- Select MPC5604/3P as MCU part number
- Set BAM status as Enabled
- Select default password
- Enter the path for the file to be flashed: Training_LED_Example.mot
- Select Start Boot Loader button to start the flash process



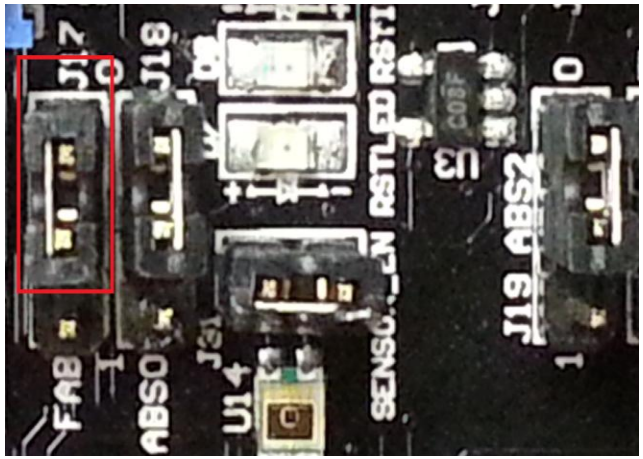
NOTE: If boot loader displays error about wrong password, try 0xFFFFFFFFFFFFFFFF instead of default password option. If your board has microcontroller with previous mask set, the default password will not work.

Flash code to target



- When asked to cycle power to MCU, press the reset button on the board. Flashing process should start.

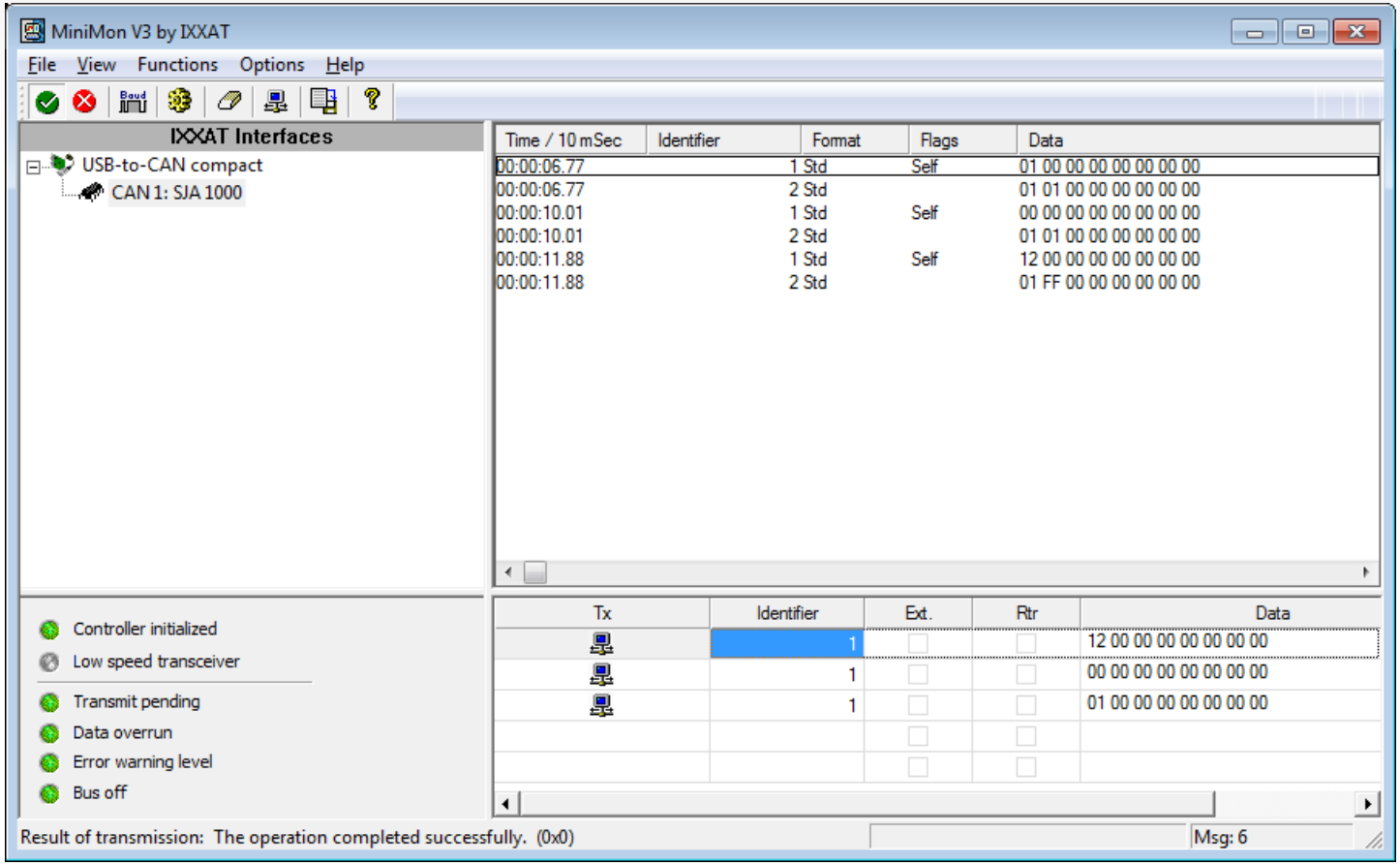
- After Flash is complete, Move the jumper J17 to position 2-3 to pull FAB low.
- Turn the power off to the module and re-apply the power
- The code should be running now.



Testing code

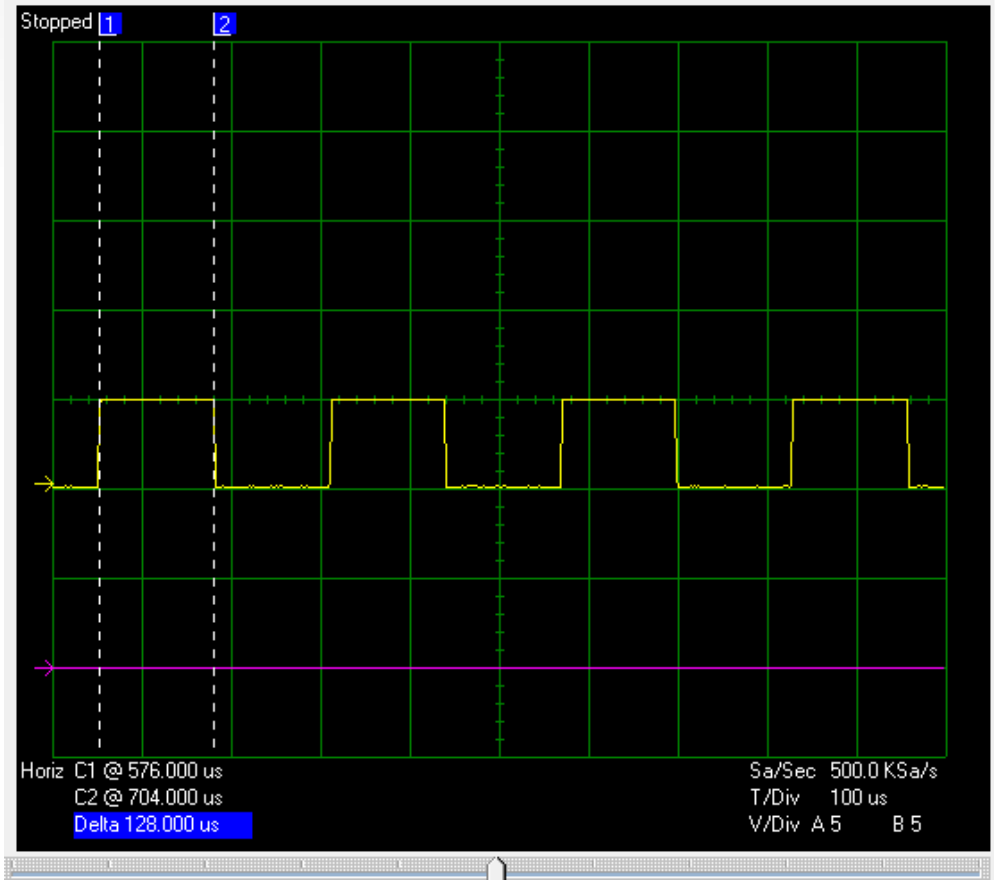
- Press switch S1 to turn on LED1 and release S1 to turn off LED1
- Turn the potentiometer halfway to observe LED2 turning On/Off.
- Send CAN command with first byte = 1 to turn on LED3 and first byte = 0 to turn off LED3
- Connect pin PD10 to a scope and check the duty cycle of PWM signal is 50%. Press switch S3 to decrease the duty cycle and S4 to increase the duty cycle.
- Connect PD10 to PA0 and using FreeMASTER, check the value of duty cycle calculated by eTimer input channel 0 (PA0).

CAN command to turn On/Off LED3



Using a CAN communication tool like CANalyzer or IXXAT MiniMon,
Send CAN command ID = 1 and first data byte = 1 to turn on LED3
Send CAN command ID = 1 and first data byte = 0 to turn off LED3

PWM output signal from PD10

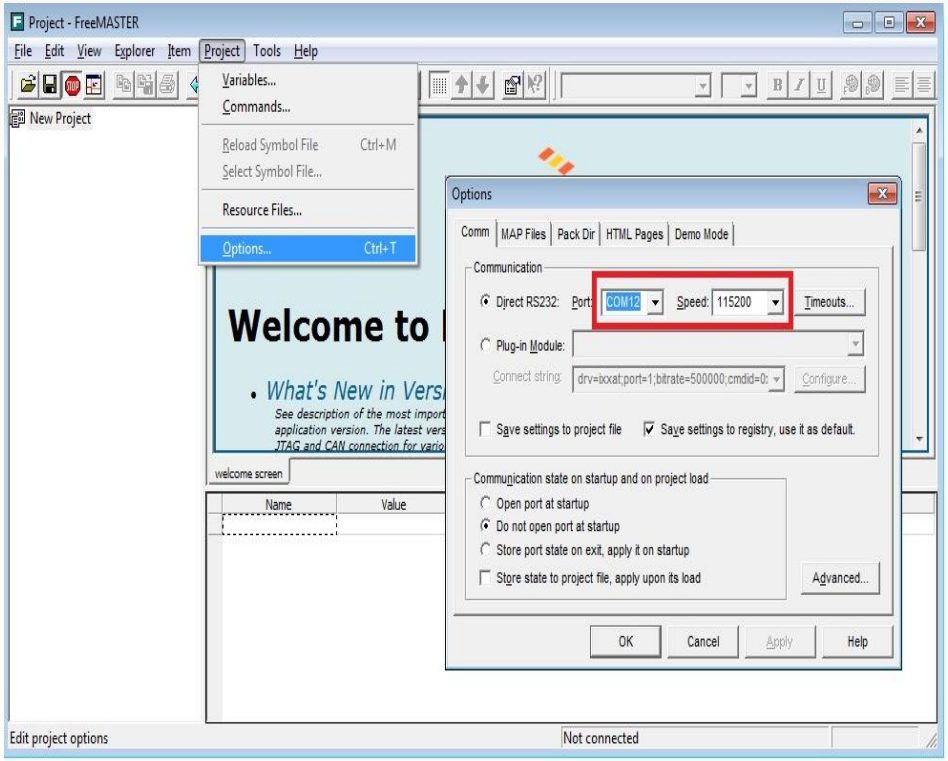


Connect PD10 to a scope and check the PWM output signal

Monitor variables using FreeMASTER

Launch FreeMASTER utility from Windows Start menu.

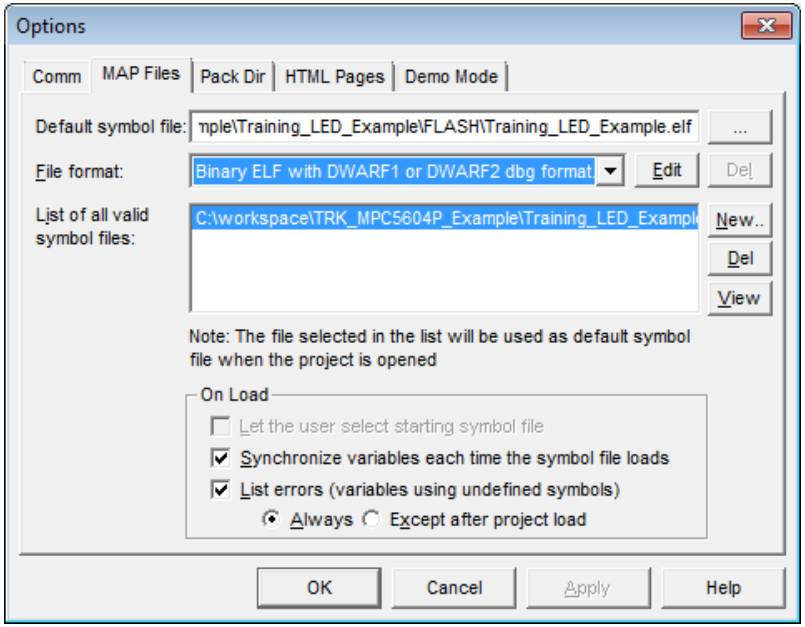
Select *Project > Options >Communication* from menu and set the communication port number and baud rate of 115200.



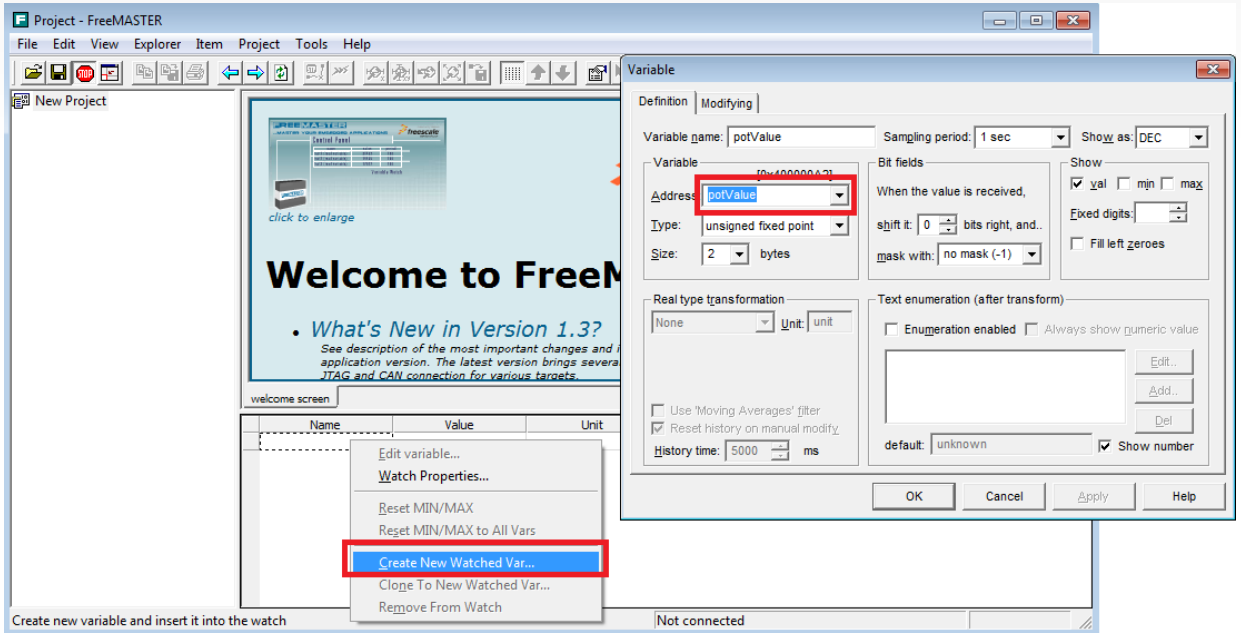
Monitor variables using FreeMASTER

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.

From the MAP tab, select the MAP file (.elf file) as shown.



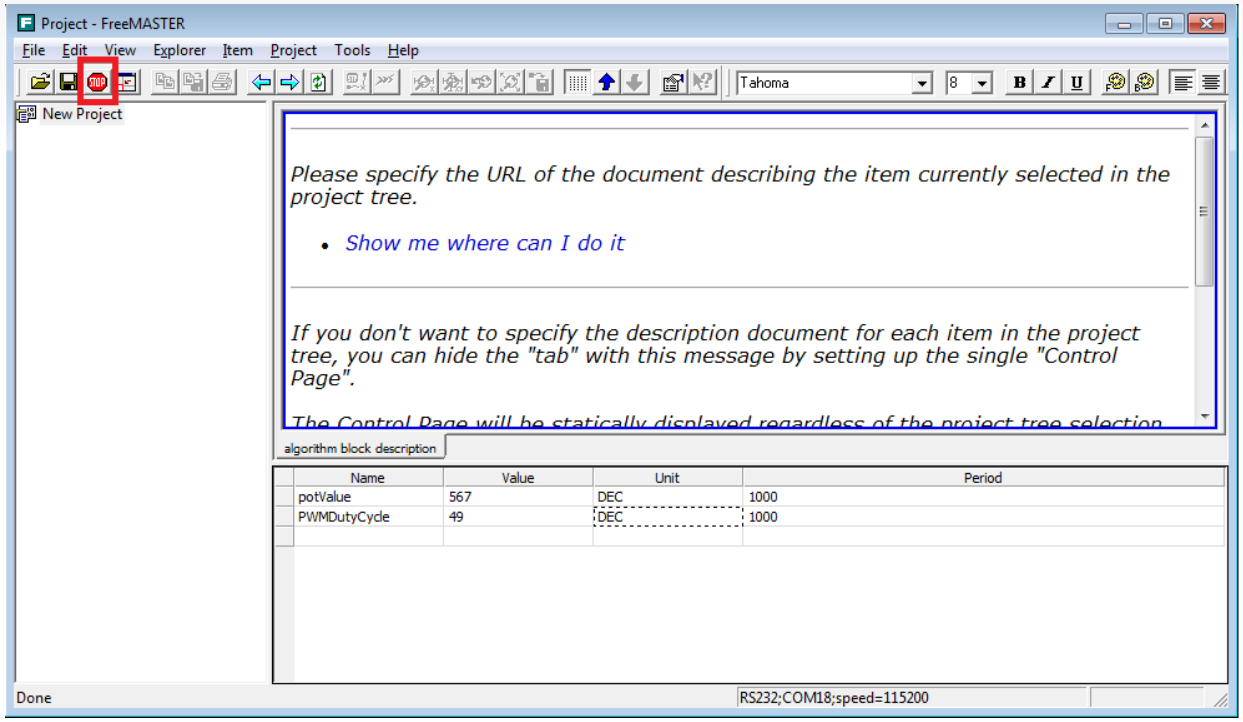
Monitor variables using FreeMASTER



Add the 2 global variables in this example project to monitor in the watch window – potValue and PWMDutyCycle as follows:

- 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 and and select OK.
- Using similar steps, add PWMDutyCycle to watch window.

Monitor variables using FreeMASTER



Select the icon *Start/Stop communication* to start communication and observe the 2 watch variables getting updated.

- When you rotate the potentiometer, the watch window should get updated.
- When switch S3 is pressed, the duty cycle value should decrease and when switch S4 is pressed, the duty cycle value should increase.

Training Summary

- Overview of tools provided with Fast Start Kit for TRK-MPC5604P
- Utilized RAppID Init Tools for fast easy infrastructure configuration and code generation
- Generated comprehensive report on the project using RAppID Init tool
- Utilized supporting tools provided with the kit to help build and flash code on to the target
- Described setting up and using the TRK-MPC5604P evaluation board


freescale™