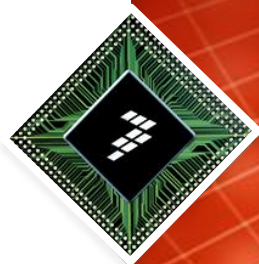




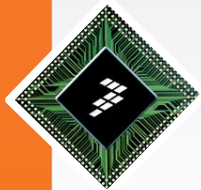
TRK-MPC5604P Fast Start kit Training



March 2014

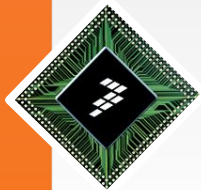
Freescale, the Freescale logo, AltVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, Qorivva, SafeAssure, the SafeAssure logo, StarCore, Symphony and VortiQa are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Airfast, BeeKit, BeeStack, CoreNet, Flexis, Layerscape, MagniV, MXG, Platform in a Package, QorIQ Converge, QUICC Engine, Ready Play, SMARTMOS, Tower, TurboLink, Vybrid and Xtrinsic are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. © 2013 Freescale Semiconductor, Inc.





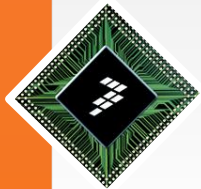
TRK-MPC5604P Fast Start Kit hardware contents

- 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.

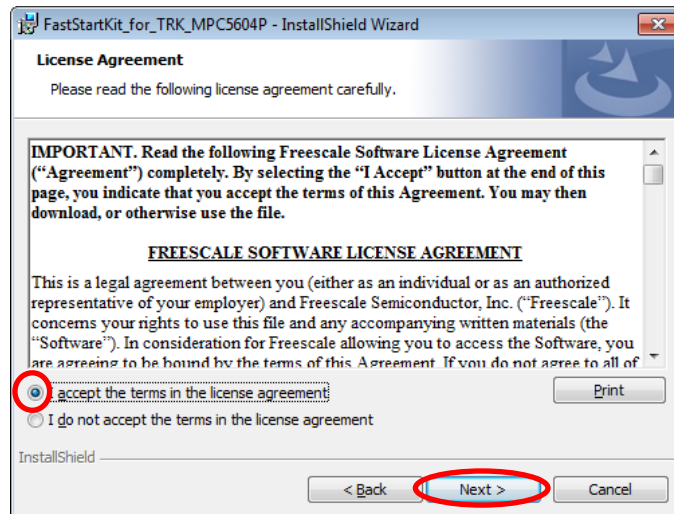
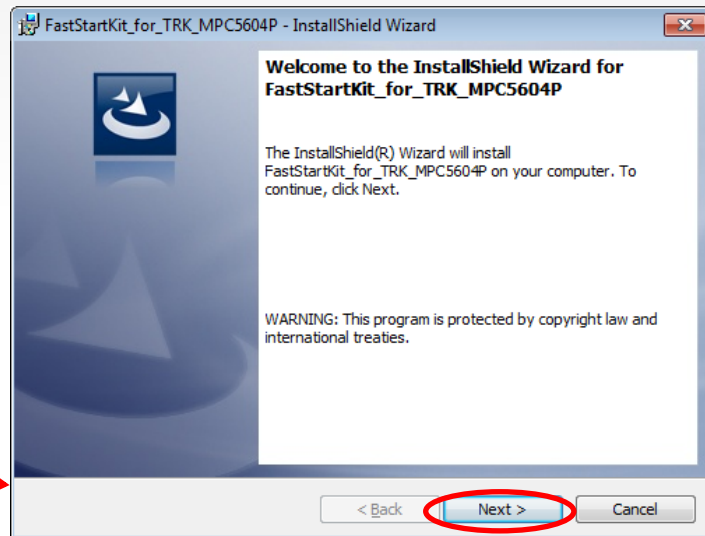


Installing Software tools

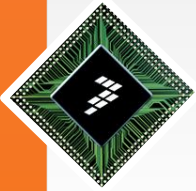
Start installation process by executing **setup.exe** located in the installation media.

Click on *Next*

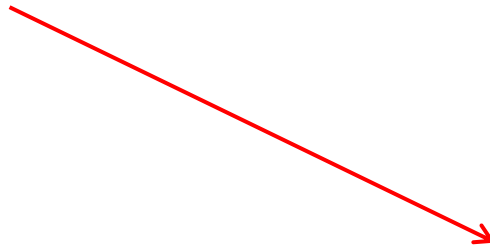
Accept the License agreement and click on *Next* to start installation of CodeWarrior 10.5



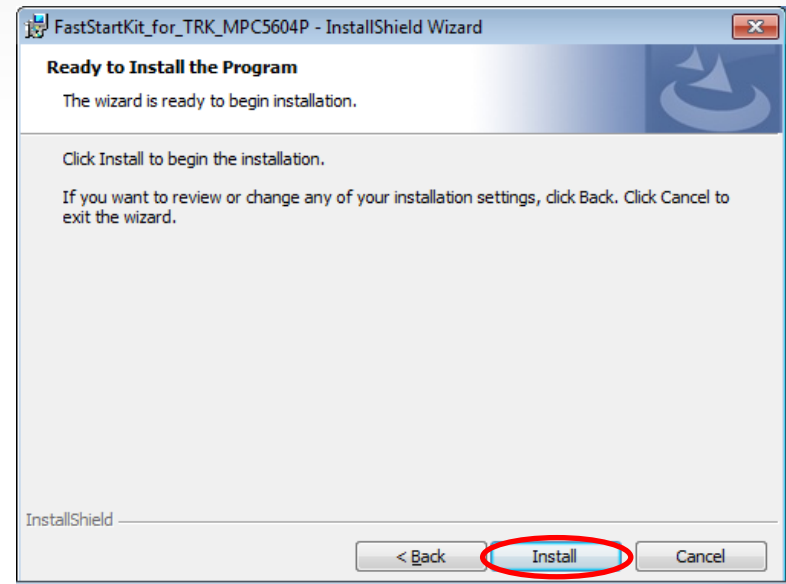
Installing Software tools

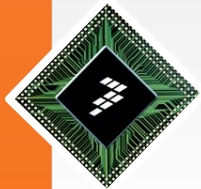


Select *Install* to begin installation of *Fast Start Kit* tools.



This will launch FreeMASTER installer.

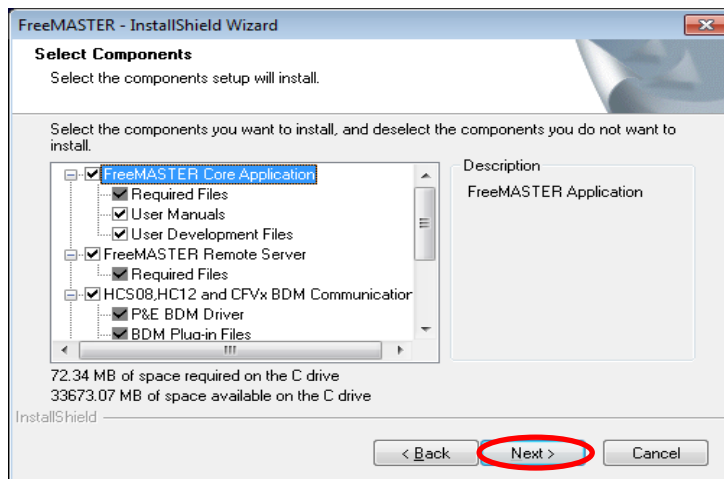
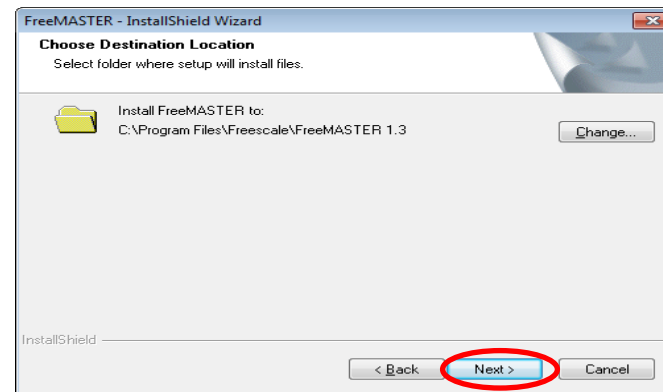
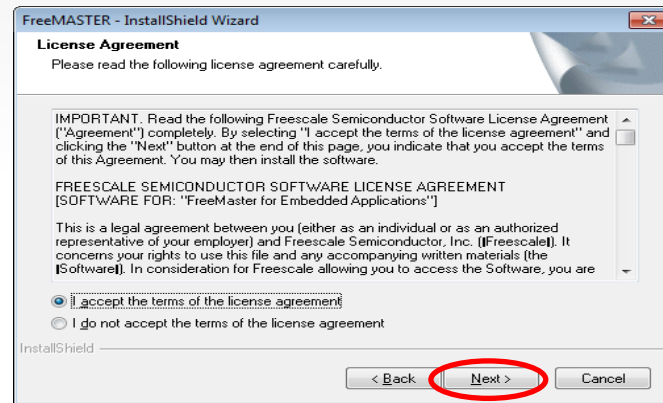


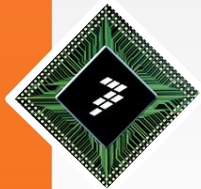


Installing Software tools – FreeMASTER

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

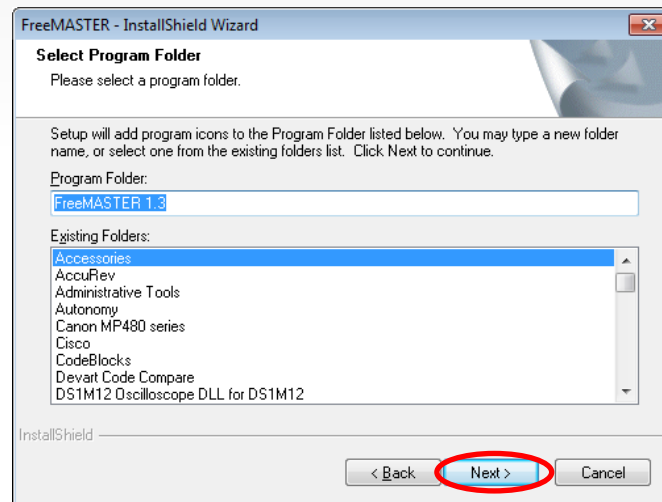
Accept default options in the “Select Components” window by selecting *Next*.



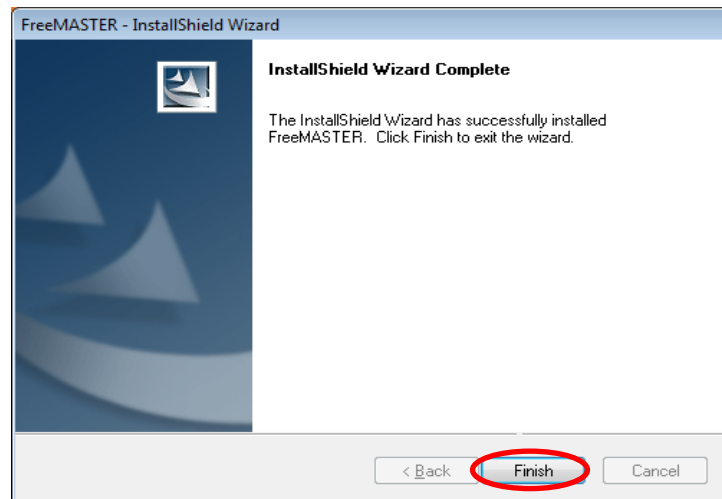


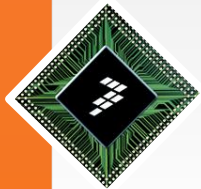
Installing Software tools – FreeMASTER

Accept default Program Folder by clicking on *Next*.



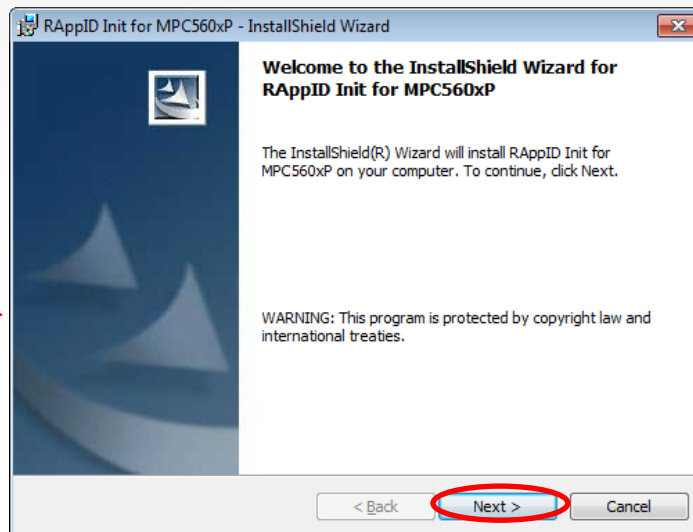
Select *Finish* to complete FreeMASTER installation. This will launch RAppID Init installer.



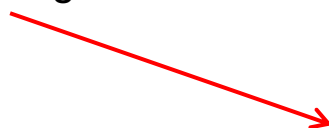


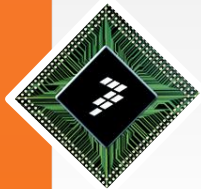
Installing Software tools – RAppID Init

Start RAppID Init installation by clicking on *Next* button



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



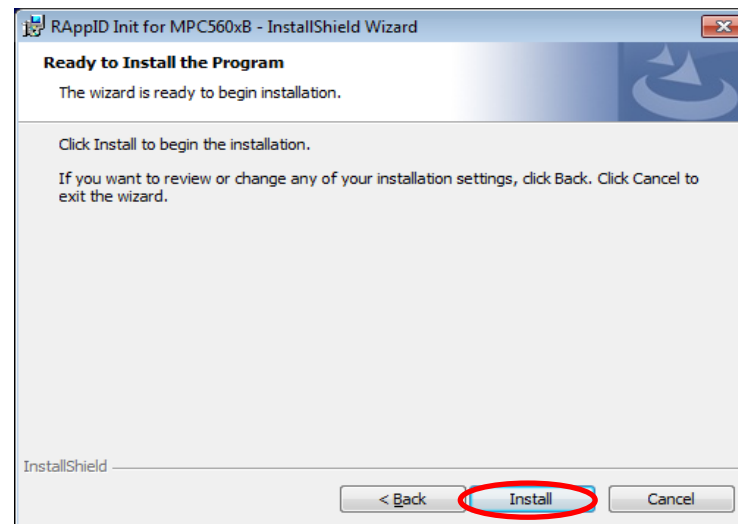
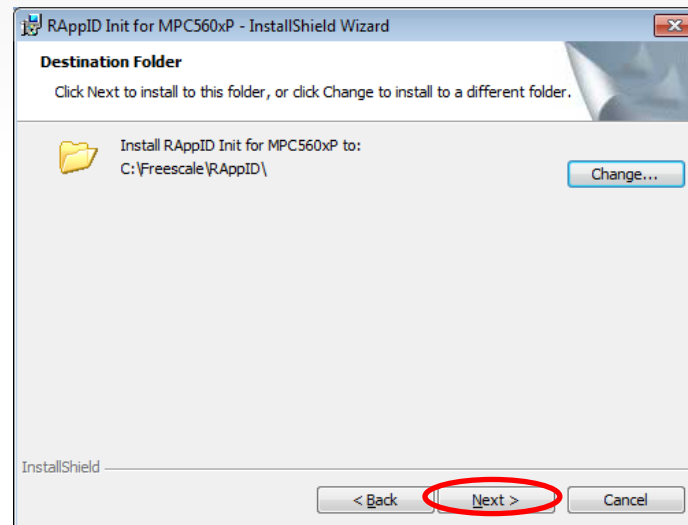
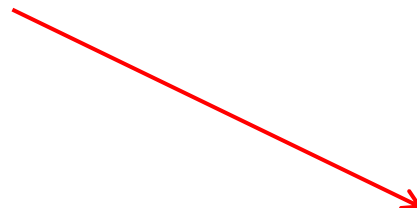


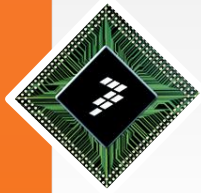
Installing Software tools – RAppID Init

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



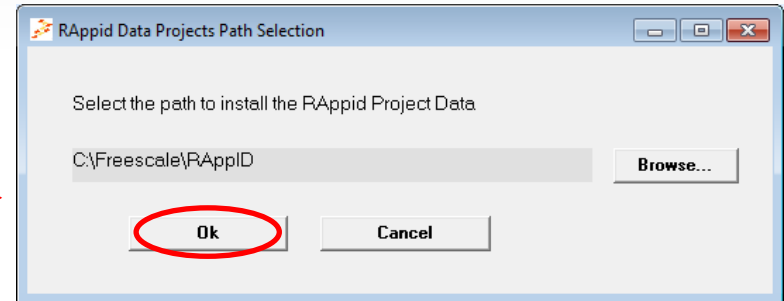
Click On *Install* button



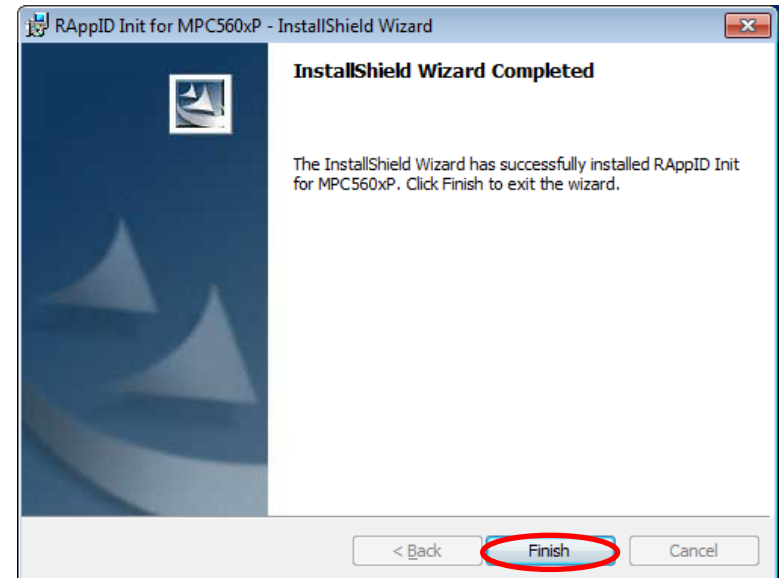


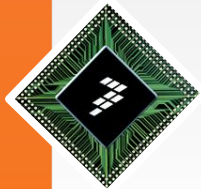
Installing Software tools – RAppID Init

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.



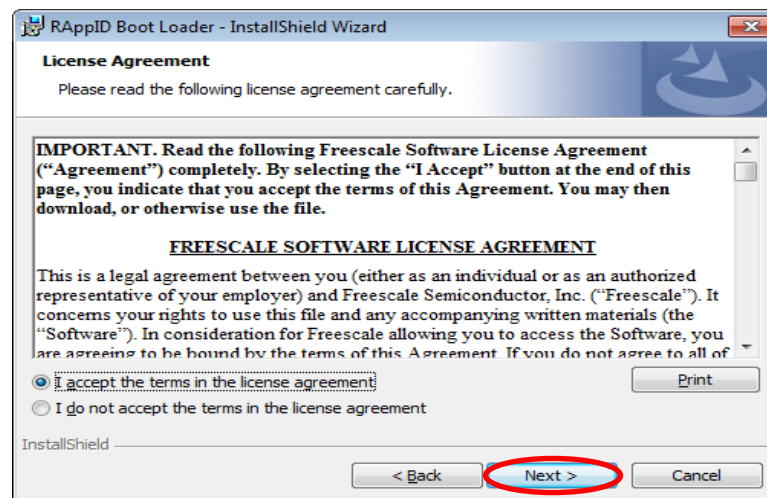
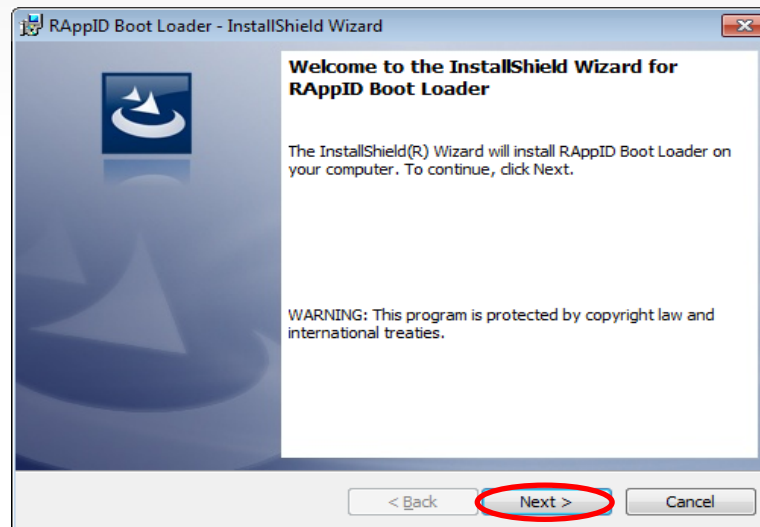
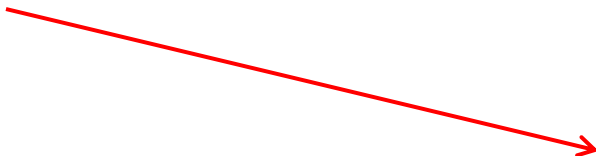


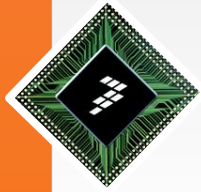
Installing Software tools – RAppID Boot loader

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



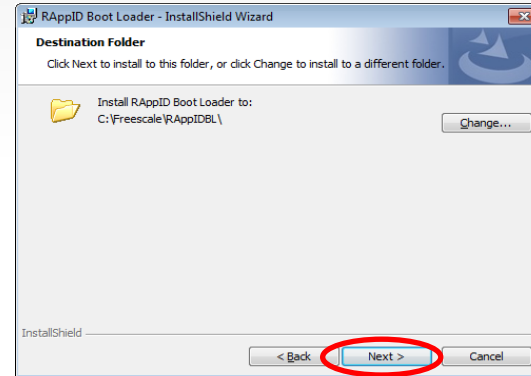
Accept license agreement and click on *Next*



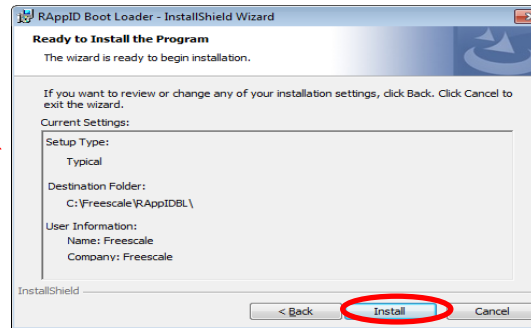


Installing Software tools – RAppID Boot loader

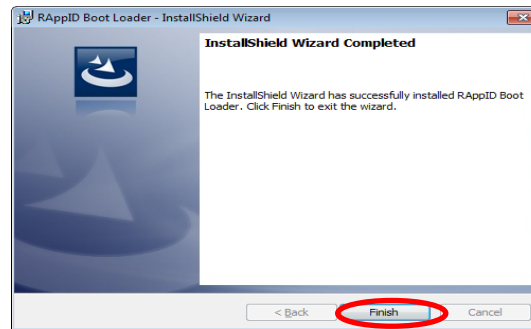
Accept default destination folder by accepting by clicking on *Next*.



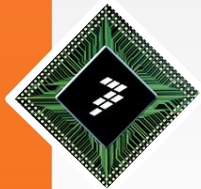
Start installation by selecting *Install*.



After installation is complete, select *Finish* to complete installation.



This will start CodeWarrior installation.

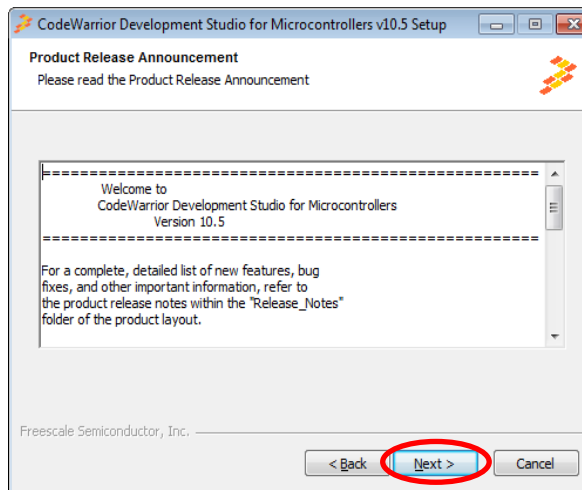
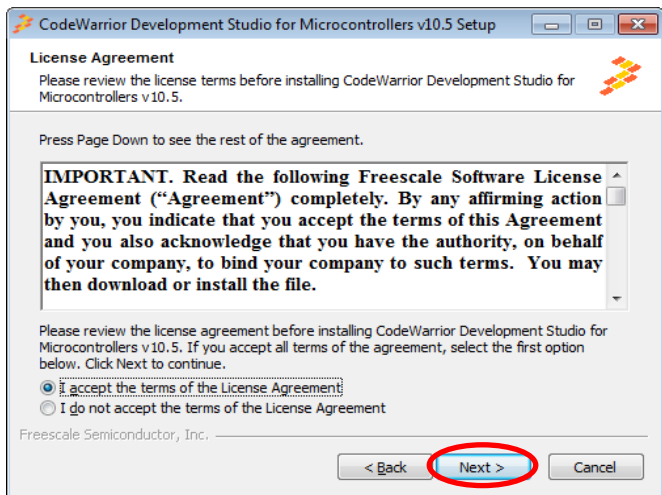
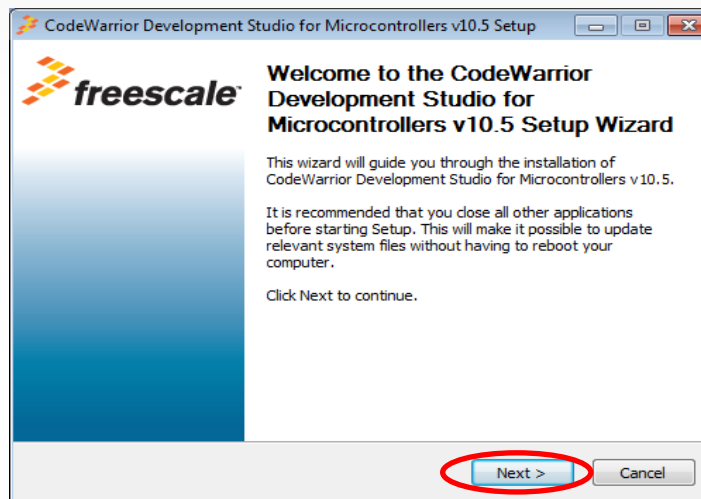
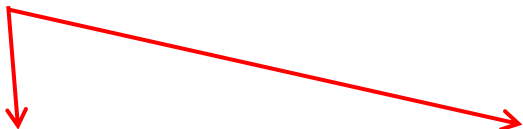


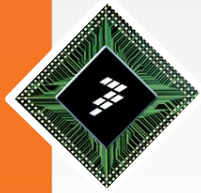
Installing Software tools - CodeWarrior

Start CodeWarrior installation by clicking on *Next*



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



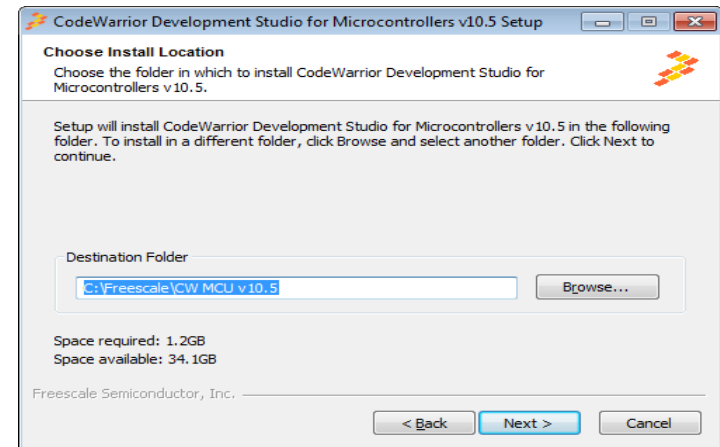
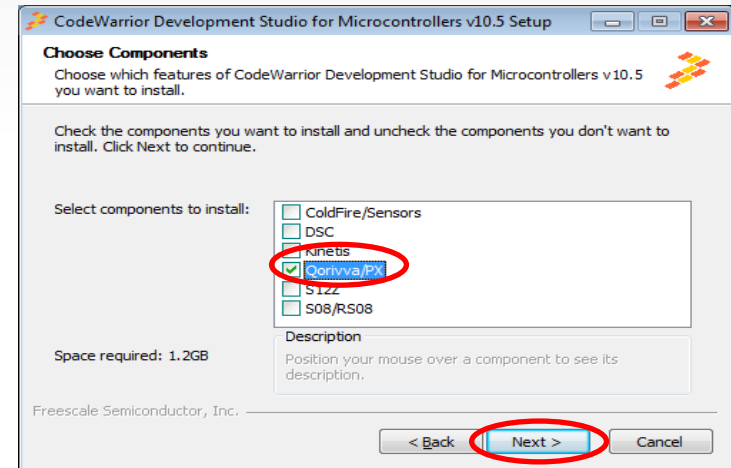


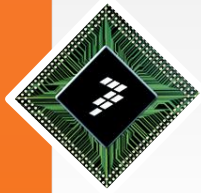
Installing Software tools - CodeWarrior

Choose Qorriva component and click on *Next*



Accept the default install location by selecting *Next*



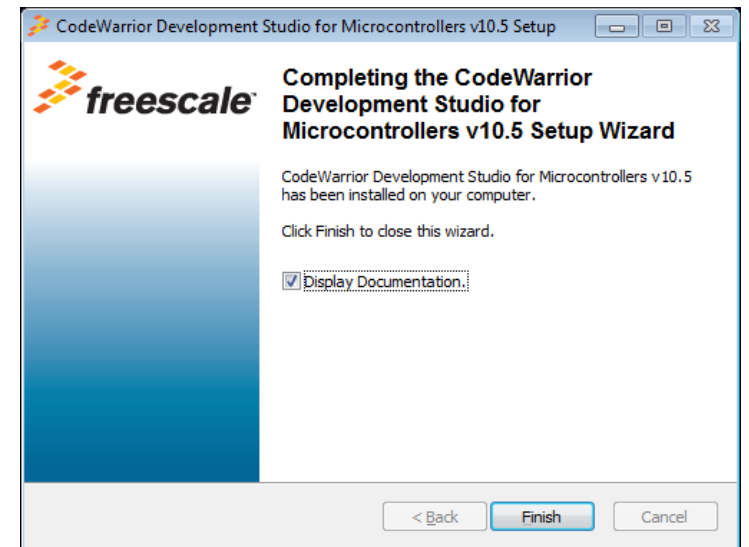
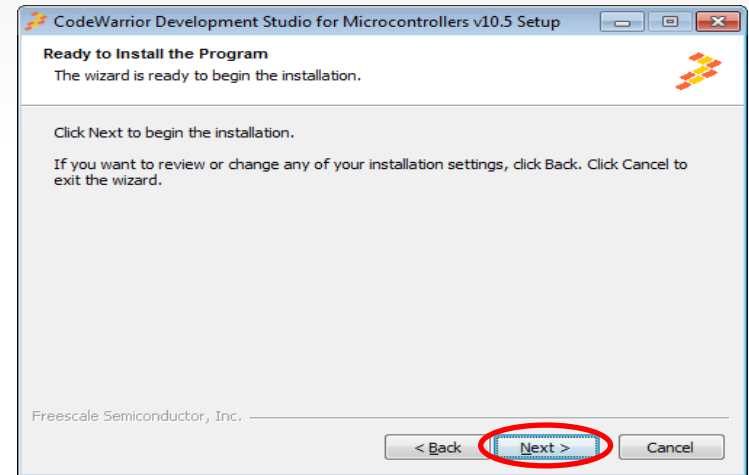


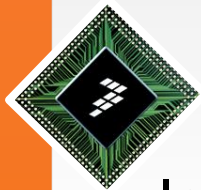
Installing Software tools - CodeWarrior

Select Next to begin CodeWarrior installation.



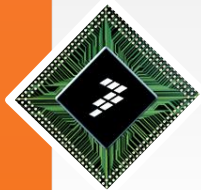
After installation is complete, select Finish to complete installation.





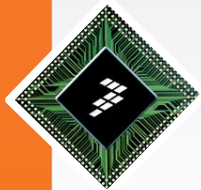
RAppID Init Overview

- 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

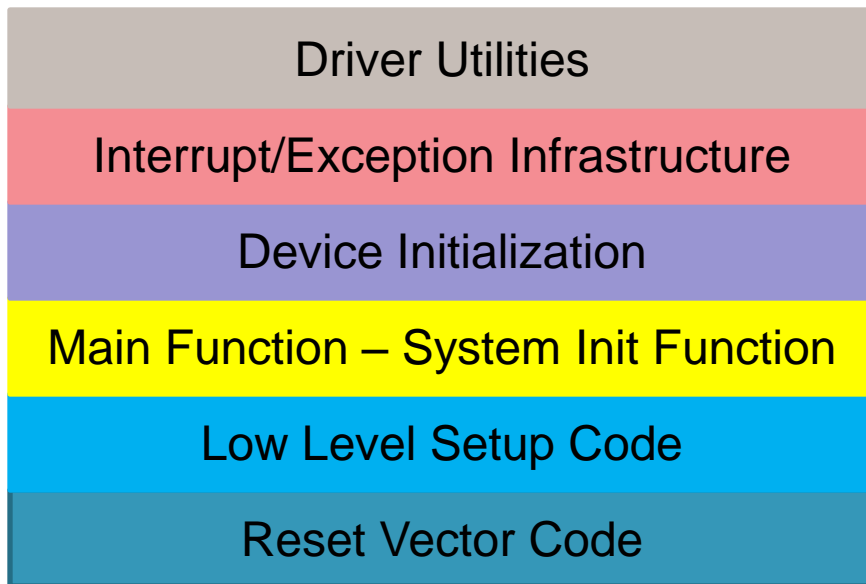


RAppID Init Overview

- 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

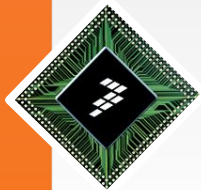
Interrupt Vector Table, Handler, ISR functions

Device and Peripheral Initialization Code

Example Main, Init Sequence Function...

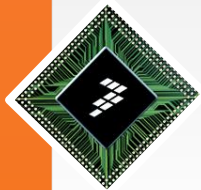
From Reset to Main, crt0, Stack,...

RCHW, Section Map...



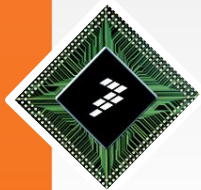
CodeWarrior project maker utility

- The Fast Start Kit provides a utility to assist in adding RAppID generated code to a CodeWarrior project
- After creating an 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.



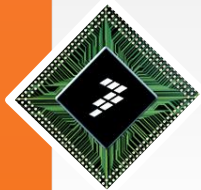
Low Level Driver code

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



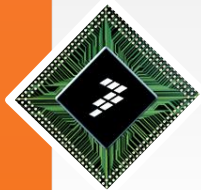
GPIO Low Level Driver code

- **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



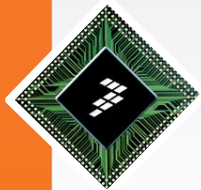
ADC Low Level Driver code

- **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



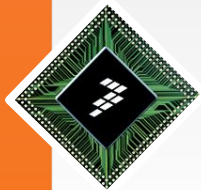
UART Low Level Driver code

- **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



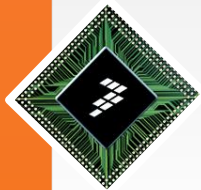
CAN Low Level Driver code

- **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).

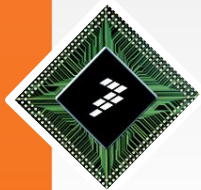
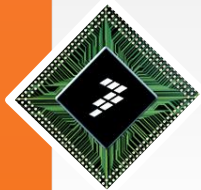


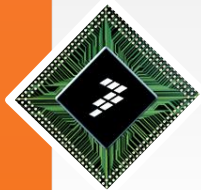
Photo Sensor High Level Driver code

- **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).



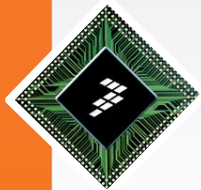
SBC High Level Driver code

- **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.



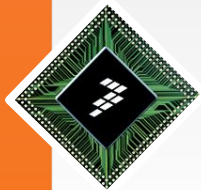
RAppID Bootloader utility

- 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.).



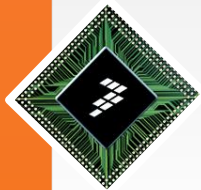
FreeMASTER utility

- 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)



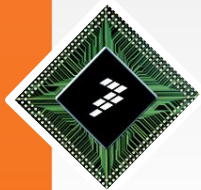
LED Example Using RAppID Init

- 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.



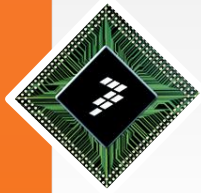
LED Example overview

- 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.

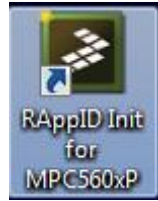


Creating LED example using RAppID init tool

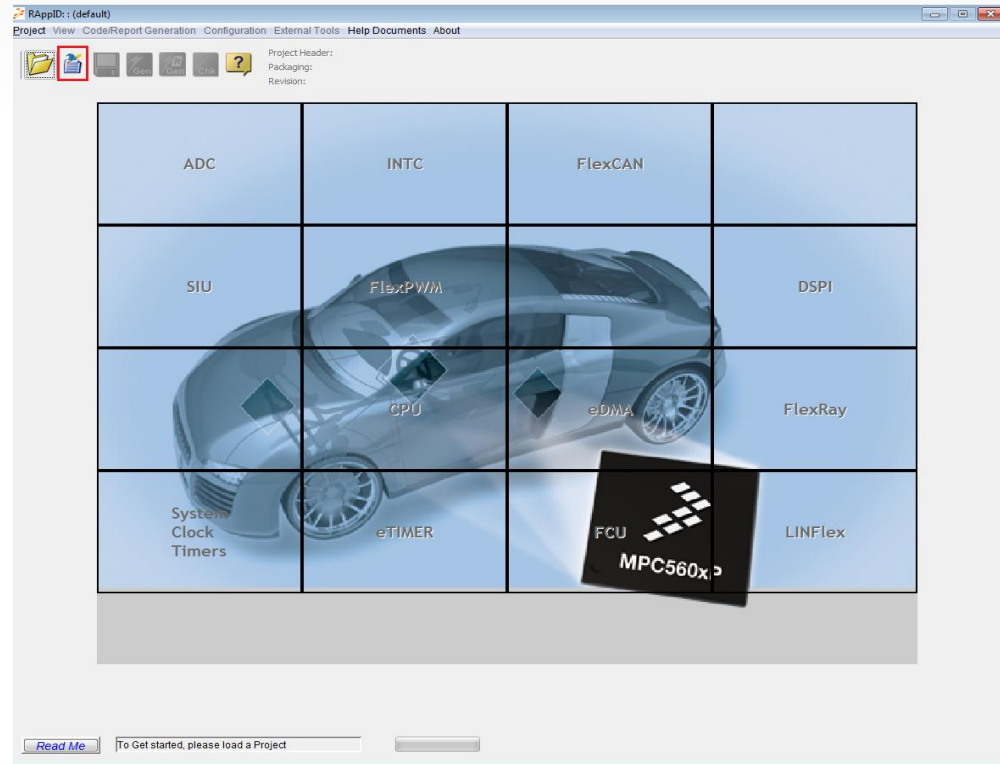
- The next few slides will demonstrate how to create RAppID project to configure all the pins and peripherals required for this example and generate code for CodeWarrior compiler.



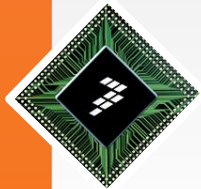
Create a New RAppID Project



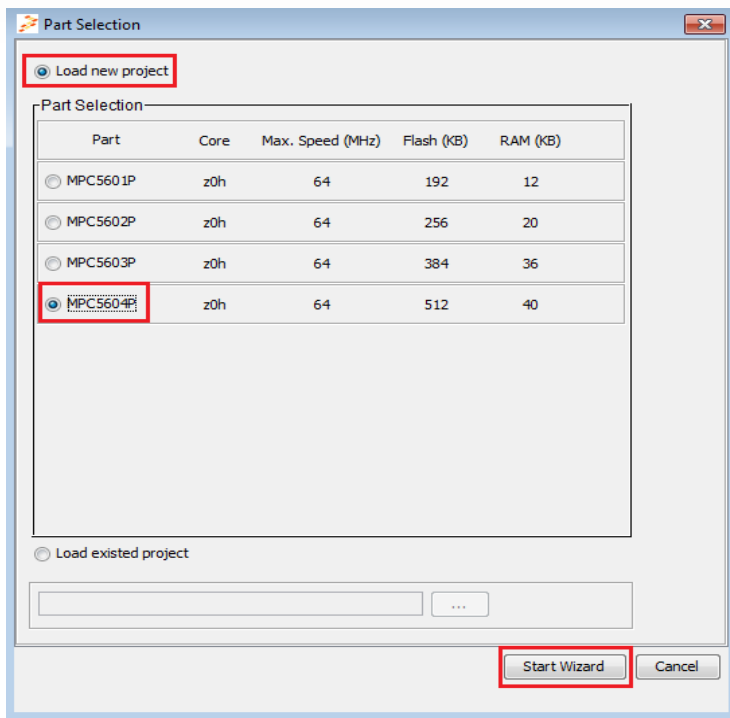
Launch RAppID init by double clicking on RAppID init desktop icon



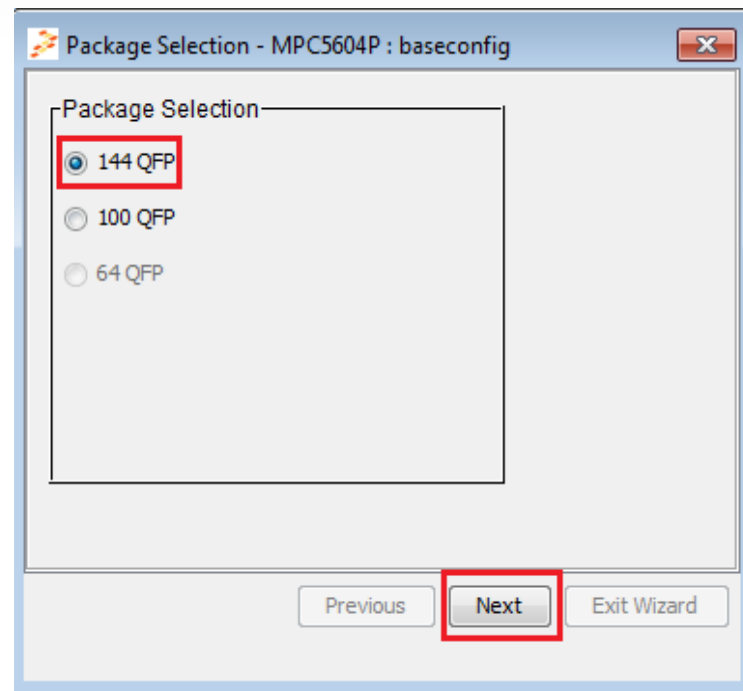
Start a new project by clicking on "New Project Wizard" button



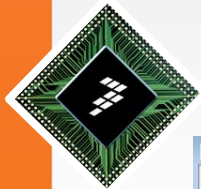
Select Part and Package



Select a Part MPC5604P and select Start Wizard



Select a Package 144 QFP and select Next



Configure ADC input

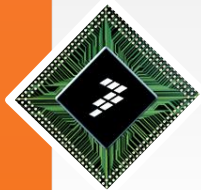
Pin Allocation Wizard - MPC5604P : baseconfig *

MPC5604P in 144 QFP

ADC_1 AN 05 PAD PE0 Allocated as Input

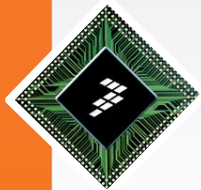
Functions	Input	Output	User Assigned Signal Name
ADC_1 AN 03	<input type="checkbox"/> PC0		
ADC_1 AN 04	<input type="checkbox"/> PD15		
ADC_1 AN 05	<input checked="" type="checkbox"/> PE0		ADC1_Ch5_Potentiometer_Input
ADC_1 AN 06	<input type="checkbox"/> PE6		

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



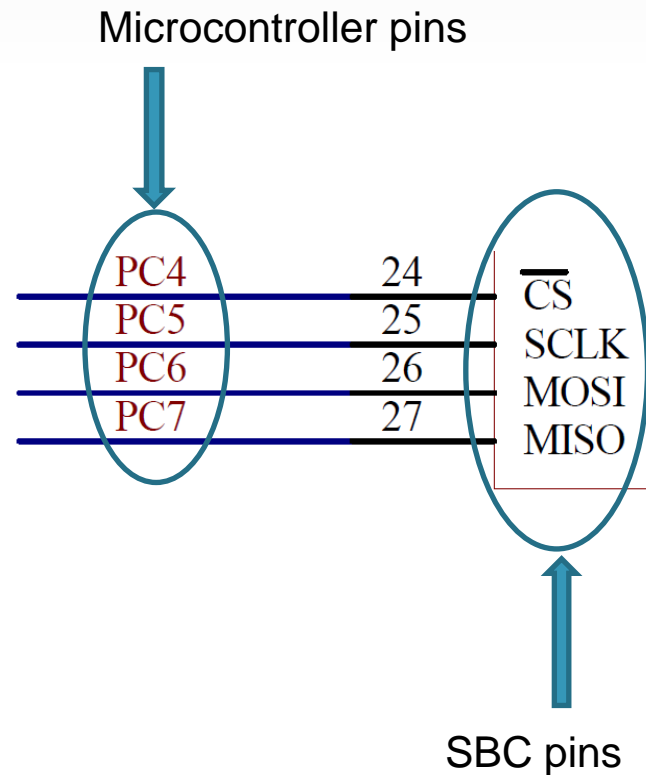
Configure DSPI pins

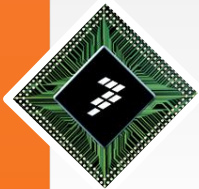
- TRK-MPC5604P contains MCZ3390S5EK system basis chip (SBC) with integrated CAN transceiver and LIN 2.0 interface.
- DSPI 0 is connected to SBC. We need to configure DSPI 0 as master and SBC as slave so that SBC can be configured to enable CAN by sending appropriate commands via DSPI 0.



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

Pin Allocation - MPC5604P : baseconfig *

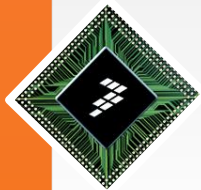
Pin Allocation Wizard

MPC5604P in 144 QFP

ADC_1 AN 05 PAD PE0 Allocated as Input
 DSPI_0 Chip Select 0 PAD PC4 Allocated as Output
 DSPI_0 Clock PAD PC5 Allocated as Output
 DSPI_0 Data Out PAD PC6 Allocated as Output
 DSPI_0 Data IN PAD PC7 Allocated as Input

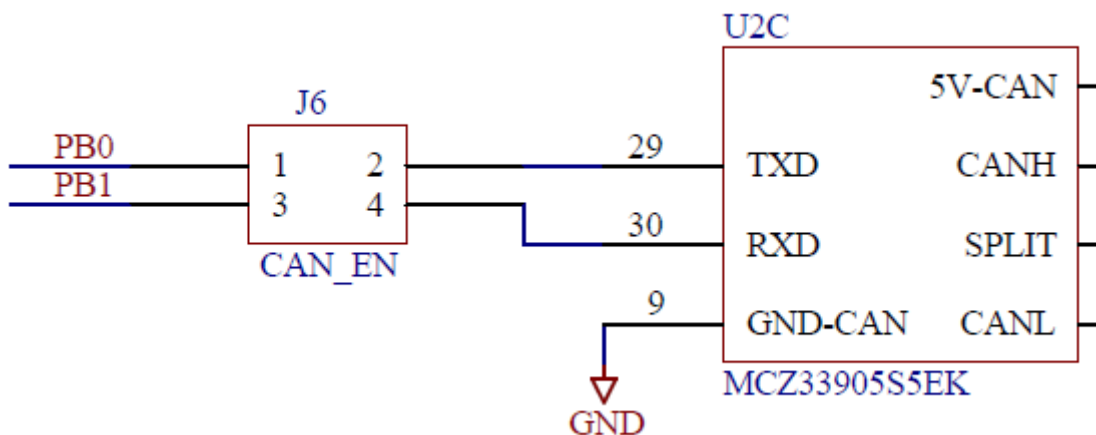
Functions	Input	Output	User Assigned Signal Name
DSPI_0 Chip Select 1		<input type="checkbox"/> PC3	
DSPI_0 Chip Select 0	<input type="checkbox"/> PC4	<input checked="" type="checkbox"/> PC4	PC4_DSPI0_CS0_Output
DSPI_0 Clock	<input type="checkbox"/> PC5	<input checked="" type="checkbox"/> PC5	PC5_DSPI0_CLK_Output
DSPI_0 Data Out		<input checked="" type="checkbox"/> PC6	PC6_DSPI0_Data_Output
DSPI_0 Data IN	<input checked="" type="checkbox"/> PC7		PC7_DSPI0_Data_Intput
DSPI_1 Chip Select 3		<input type="checkbox"/> PD7	

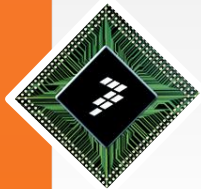
Previous Next Exit Wizard View



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

Pin Allocation - MPC5604P : baseconfig *

Pin Allocation Wizard

MPC5604P in 144 QFP

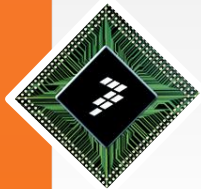
ADC DSPIC FCU **FlexCAN** FlexPWM FlexRay GPIO LINFlex MISC eTIMER

Functions	Input	Output	User Assigned Signal Name
CAN_0 Tx		<input checked="" type="checkbox"/> PB0	PB0_CAN0_TX
CAN_0 Rx	<input checked="" type="checkbox"/> PB1		PB1_CAN0_RX
SAFEPORT Tx		<input type="checkbox"/> PA14	
SAFEPORT Rx	<input type="checkbox"/> PA15		

ADC_1 AN 05 PAD PE0 Allocated as Input
 DSPIC_0 Chip Select 0 PAD PC4 Allocated as Output
 DSPIC_0 Clock PAD PC5 Allocated as Output
 DSPIC_0 Data Out PAD PC6 Allocated as Output
 DSPIC_0 Data IN PAD PC7 Allocated as Input
 CAN_0 Tx PAD PB0 Allocated as Output
 CAN_0 Rx PAD PB1 Allocated as Input

Previous Next Exit Wizard View

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

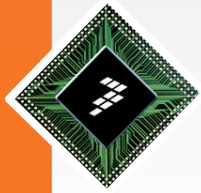


Configure FlexPWM pin

The screenshot shows the 'Pin Allocation Wizard' for the MPC5604P. The 'FlexPWM' tab is selected. The table below shows the configuration for FlexPWM outputs:

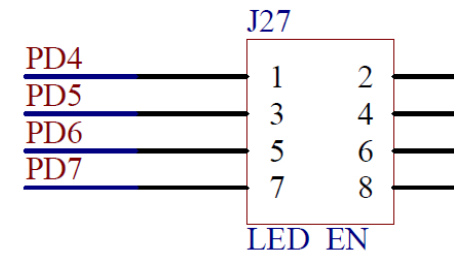
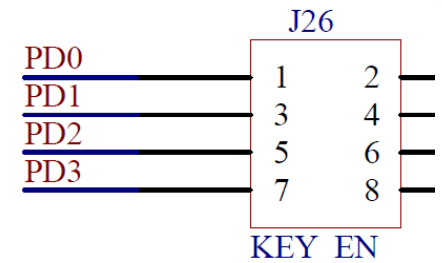
Functions	Input	Output	User Assigned Signal Name
FlexPWM_A_0	<input type="checkbox"/> PD10 <input type="checkbox"/> PA11	<input checked="" type="checkbox"/> PD10 <input type="checkbox"/> PA11	PD10_FlexPWM_A0
FlexPWM_A_1	<input type="checkbox"/> PC7 <input type="checkbox"/> PD13 <input type="checkbox"/> PC15	<input type="checkbox"/> PC7 <input type="checkbox"/> PD13 <input type="checkbox"/> PC15	
FlexPWM_A_2	<input type="checkbox"/> PA12 <input type="checkbox"/> PA11 <input type="checkbox"/> PG3	<input type="checkbox"/> PA12 <input type="checkbox"/> PA11 <input type="checkbox"/> PG3	
FlexPWM_A_3	<input type="checkbox"/> PD3 <input type="checkbox"/> PA2 <input type="checkbox"/> PC10 <input type="checkbox"/> PG6	<input type="checkbox"/> PD3 <input type="checkbox"/> PA2 <input type="checkbox"/> PC10 <input type="checkbox"/> PG6	
FlexPWM_B_0	<input type="checkbox"/> PA10 <input type="checkbox"/> PD11	<input type="checkbox"/> PA10 <input type="checkbox"/> PD11	
FlexPWM_B_1	<input type="checkbox"/> PD14 <input type="checkbox"/> PC6 <input type="checkbox"/> PD0	<input type="checkbox"/> PD14 <input type="checkbox"/> PC6 <input type="checkbox"/> PD0	
FlexPWM_B_2	<input type="checkbox"/> PA13 <input type="checkbox"/> PA12 <input type="checkbox"/> PG4	<input type="checkbox"/> PA13 <input type="checkbox"/> PA12 <input type="checkbox"/> PG4	
FlexPWM_B_3	<input type="checkbox"/> PA3 <input type="checkbox"/> PA9 <input type="checkbox"/> PD4 <input type="checkbox"/> PG7	<input type="checkbox"/> PA3 <input type="checkbox"/> PA9 <input type="checkbox"/> PD4 <input type="checkbox"/> PG7	

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

Pin Allocation - MPC5604P : baseconfig *

Pin Allocation Wizard

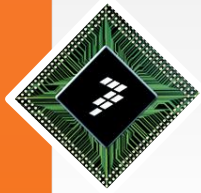
MPC5604P in 144 QFP

ADC DSPI FCU FlexCAN FlexPWM FlexRa **GPIO** LINFlex MISC eTIMER

Functions	Input	Output	User Assigned Signal Name
PD0	<input checked="" type="checkbox"/> PD0	<input type="checkbox"/> PD0	Input_Switch_S1
PD1	<input checked="" type="checkbox"/> PD1	<input type="checkbox"/> PD1	Input_Switch_S2
PD2	<input checked="" type="checkbox"/> PD2	<input type="checkbox"/> PD2	Input_Switch_S3
PD3	<input checked="" type="checkbox"/> PD3	<input type="checkbox"/> PD3	Input_Switch_S4
PD4	<input type="checkbox"/> PD4	<input checked="" type="checkbox"/> PD4	LED1_Output
PD5	<input type="checkbox"/> PD5	<input checked="" type="checkbox"/> PD5	LED2_Output
PD6	<input type="checkbox"/> PD6	<input checked="" type="checkbox"/> PD6	LED3_Output
PD7	<input checked="" type="checkbox"/> PD7	<input type="checkbox"/> PD7	

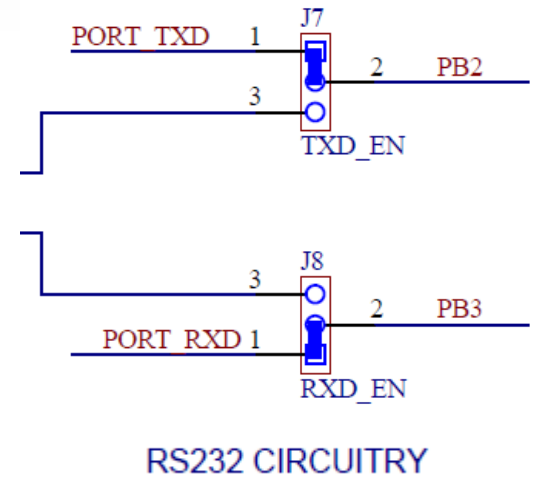
ADC_1 AN 05 PAD PE0 Allocated as Input
 DSPI_0 Chip Select 0 PAD PC4 Allocated as Output
 DSPI_0 Clock PAD PC5 Allocated as Output
 DSPI_0 Data Out PAD PC6 Allocated as Output
 DSPI_0 Data IN PAD PC7 Allocated as Input
 CAN_0 Tx PAD PB0 Allocated as Output
 CAN_0 Rx PAD PB1 Allocated as Input
 FlexPWM_A 0 PAD PD10 Allocated as Output
 PD4 PAD PD4 Allocated as Output
 PD5 PAD PD5 Allocated as Output
 PD6 PAD PD6 Allocated as Output

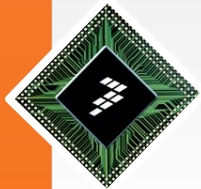
Previous Next Exit Wizard View



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.

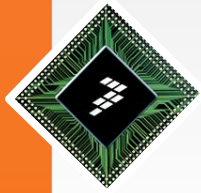




Configure LINFlex (UART) pins

Functions	Input	Output	User Assigned Signal Name
LINFlex_0 Tx		<input checked="" type="checkbox"/> PB2	PB2_UART_TX
LINFlex_0 Rx	<input checked="" type="checkbox"/> PB3		PB3_UART_RX
LINFlex_1 Tx		<input type="checkbox"/> PC3 <input type="checkbox"/> PD9 <input type="checkbox"/> PF14	
LINFlex_1 Rx	<input type="checkbox"/> PF15 <input type="checkbox"/> PB13 <input type="checkbox"/> PD12		

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

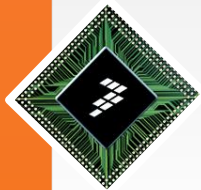


Configure eTimer pin

Functions	Input	Output	User Assigned Signal Name
eTimer_0_CH0	<input checked="" type="checkbox"/> PA0	<input type="checkbox"/> PA0	PA0_eTimer_Input
eTimer_0_CH1	<input type="checkbox"/> PA1	<input type="checkbox"/> PA1	
eTimer_0_CH2	<input type="checkbox"/> PA2	<input type="checkbox"/> PA2	
eTimer_0_CH3	<input type="checkbox"/> PA3	<input type="checkbox"/> PA3	
eTimer_0_CH4	<input type="checkbox"/> PC11 <input type="checkbox"/> PB14 <input type="checkbox"/> PA4	<input type="checkbox"/> PC11 <input type="checkbox"/> PA4	
eTimer_0_CH5	<input type="checkbox"/> PC12 <input type="checkbox"/> PB8	<input type="checkbox"/> PC12	

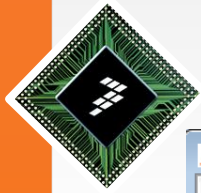
In this example, we are using eTimer Channel 1 as input capture function to calculate duty signal of PWM signal. In eTimer tab, configure PA0 as eTimer input pin and add user signal names.

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

System Clock/Timers - MPC5604P : baseconfig *

Mode Entry | System Clock | CMU | SWT | PIT | STM | Software Attributes

General Configuration | Peripheral Configuration | Interrupt

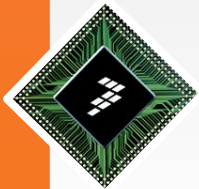
Mode Enable

RESET Mode	EN	TEST Mode	DI
SAFE Mode	EN	DRUN Mode	EN
RUN0 Mode	EN	RUN1 Mode	DI
RUN2 Mode	DI	RUN3 Mode	DI
HALT0 Mode	DI	STOP0 Mode	DI

Mode Configuration

Mode	\$ PDO	\$ MVR ON	\$ DFLA ON	\$ CFLA ON	\$ PLL1 ON	\$ PLL0 ON	\$ XOSC0 ON	\$ 16MHz_IRC ON	\$ SYSCLK
RESET	DI	EN	Normal	Normal	DI	DI	DI	EN	16MHz internal RC oscillator
TEST	DI	EN	Normal	Normal	DI	DI	DI	EN	16MHz internal RC oscillator
SAFE	EN	EN	Normal	Normal	DI	DI	DI	EN	16MHz internal RC oscillator
DRUN	DI	EN	Normal	Normal	DI	EN	EN	EN	System PLL (PLL0)
RUN0	DI	EN	Normal	Normal	DI	DI	DI	EN	16MHz internal RC oscillator
RUN1	DI	EN	Normal	Normal	DI	DI	DI	EN	16MHz internal RC oscillator
RUN2	DI	EN	Normal	Normal	DI	DI	DI	EN	16MHz internal RC oscillator
RUN3	DI	EN	Normal	Normal	DI	DI	DI	EN	16MHz internal RC oscillator
HALT0	DI	EN	Power Down	Power Down	DI	DI	DI	EN	16MHz internal RC oscillator
STOP0	DI	EN	Power Down	Power Down	DI	DI	DI	EN	16MHz internal RC oscillator

Previous Next Exit Wizard



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

The screenshot shows the 'System Clock/Timers - MPC5604P : baseconfig *' window. The 'Mode Entry' tab is selected, and the 'Normal' configuration is highlighted. Below this, three tables show configuration details for various peripherals.

Run Peripheral Configuration

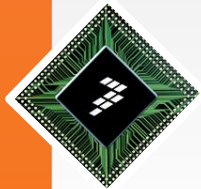
Peripheral Run Configurations	RESET	\$ TEST	\$ SAFE	\$ DRUN	\$ RUN0	\$ RUN1	\$ RUN2	\$ RUN3
Run Peripherals Config0 Active/Frozen	DI	EN	EN	EN	EN	EN	EN	EN
Run Peripherals Config1 Active/Frozen	DI	DI	DI	DI	DI	DI	DI	DI
Run Peripherals Config2 Active/Frozen	DI	DI	DI	DI	DI	DI	DI	DI
Run Peripherals Config3 Active/Frozen	DI	DI	DI	DI	DI	DI	DI	DI
Run Peripherals Config4 Active/Frozen	DI	DI	DI	DI	DI	DI	DI	DI
Run Peripherals Config5 Active/Frozen	DI	DI	DI	DI	DI	DI	DI	DI
Run Peripherals Config6 Active/Frozen	DI	DI	DI	DI	DI	DI	DI	DI
Run Peripherals Config7 Active/Frozen	DI	DI	DI	DI	DI	DI	DI	DI

Low Power Mode Configuration

Peripheral Low Power Configurations	\$ HALT0	\$ STOPO
Low Power Peripherals Config0 Active/Frozen	EN	EN
Low Power Peripherals Config1 Active/Frozen	DI	DI
Low Power Peripherals Config2 Active/Frozen	DI	DI
Low Power Peripherals Config3 Active/Frozen	DI	DI
Low Power Peripherals Config4 Active/Frozen	DI	DI
Low Power Peripherals Config5 Active/Frozen	DI	DI
Low Power Peripherals Config6 Active/Frozen	DI	DI
Low Power Peripherals Config7 Active/Frozen	DI	DI

Peripheral Mode Configuration - Normal

Peripheral Control No.	Peripheral Name	\$ Debug Mode	\$ Low Power Mode Configuration	\$ Run Mode Configuration
4	DSPL_0	DI	Low-Power Mode Configuration 0	RUN Mode Configuration 0
5	DSPL_1	DI	Low-Power Mode Configuration 0	RUN Mode Configuration 0
6	DSPL_2	DI	Low-Power Mode Configuration 0	RUN Mode Configuration 0
7	DSPL_3	DI	Low-Power Mode Configuration 0	RUN Mode Configuration 0
16	FlexCAN_0	DI	Low-Power Mode Configuration 0	RUN Mode Configuration 0
24	FlexRay	DI	Low-Power Mode Configuration 0	RUN Mode Configuration 0
26	SafetyPort	DI	Low-Power Mode Configuration 0	RUN Mode Configuration 0
32	ADC_0	DI	Low-Power Mode Configuration 0	RUN Mode Configuration 0
33	ADC_1	DI	Low-Power Mode Configuration 0	RUN Mode Configuration 0
35	CTU_0	DI	Low-Power Mode Configuration 0	RUN Mode Configuration 0
38	eTimer0	DI	Low-Power Mode Configuration 0	RUN Mode Configuration 0



Configure Clock

System Clock/Timers - MPC5604P : baseconfig *

Mode Entry: **System Clock** | CMU | SWT | PIT | STM | Software Attributes |

System Clock and Peripheral Clock Setup

System Clock Setup

DRUN Mode Clock: System PLL | XOSC Frequency: **8.00** MHz | System Clock Frequency: 32.000 MHz

Clock Source Configuration

Clock Source	\$ Bypass	Divider	\$ End of Count Value	\$ Interrupt	\$ TRIM
XOSC	DI		0x80	DI	
16MHz IRC		1			0x0

System Clock Frequency Formula

System Clock Frequency = XOSC Frequency * Final Reference Frequency Multiplier

PLL 0 | PLL 1 |

PLL Setup

Progressive Clock Switching: **DI**

Input Division Factor: **2**

Output Division Factor: 8

Loop Division Factor: 64

Final Reference Frequency Multiplier: 4

PLL Frequency Modulation Setup

Frequency Modulation: **DI**

Spread Type: Center Spread

Modulation Period: 0

Increment Step: 0

Clock Output Setup

Clock Output: **DI**

Clock Output Select: 16MHz Internal RC oscillator

Clock Divider: 1

Clock Output Frequency: 16.000 MHz

Output Clock Pin: 138 (PB6) - Not Assigned

Auxiliary Clock Setup

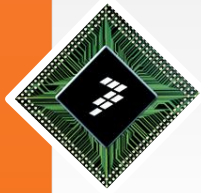
Aux 0 | Aux 1 | Aux 2 | Aux 3 |

Clock Source: 16MHz Internal RC Oscillator

Peripheral Clock Set	\$ Divider Status	\$ Divider	Final Clock (MHz)
FlexPWM Clock	EN	1	16.000

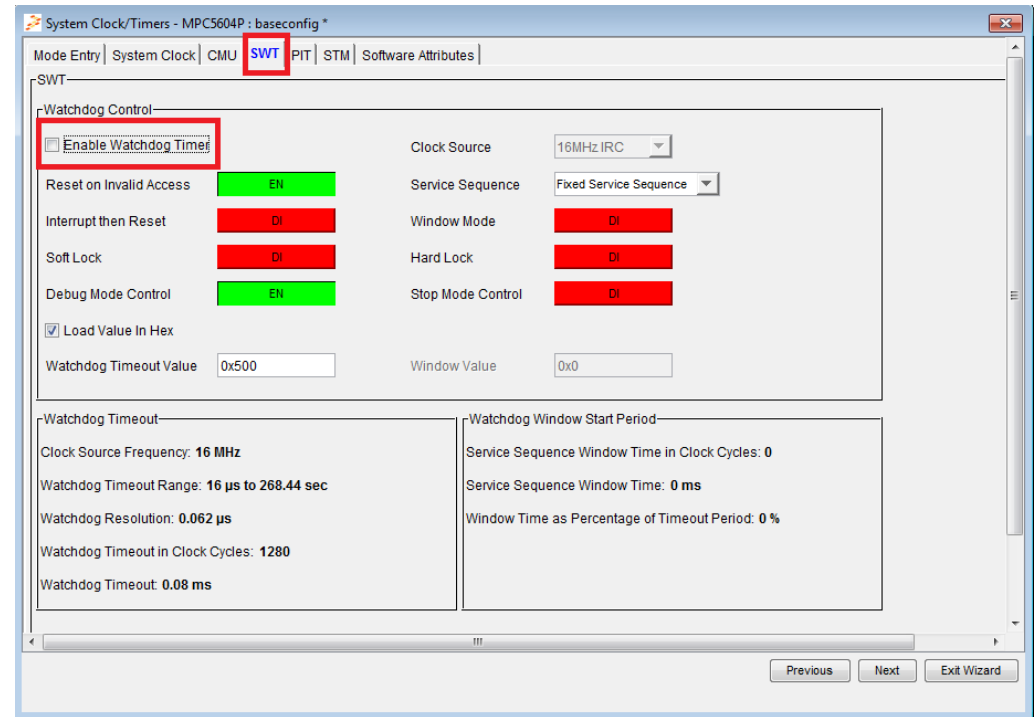
Previous | Next | Exit Wizard

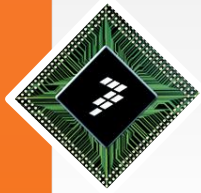
We will configure system clock to 32MHz. The default XOSC frequency in RAppID is 40MHz but TRK-MPC5604P uses 8 MHz crystal. Change XOSC value to 8 MHz. Change Input Division factor to 2 to set the system clock at 32 MHz.



Disable Watchdog

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

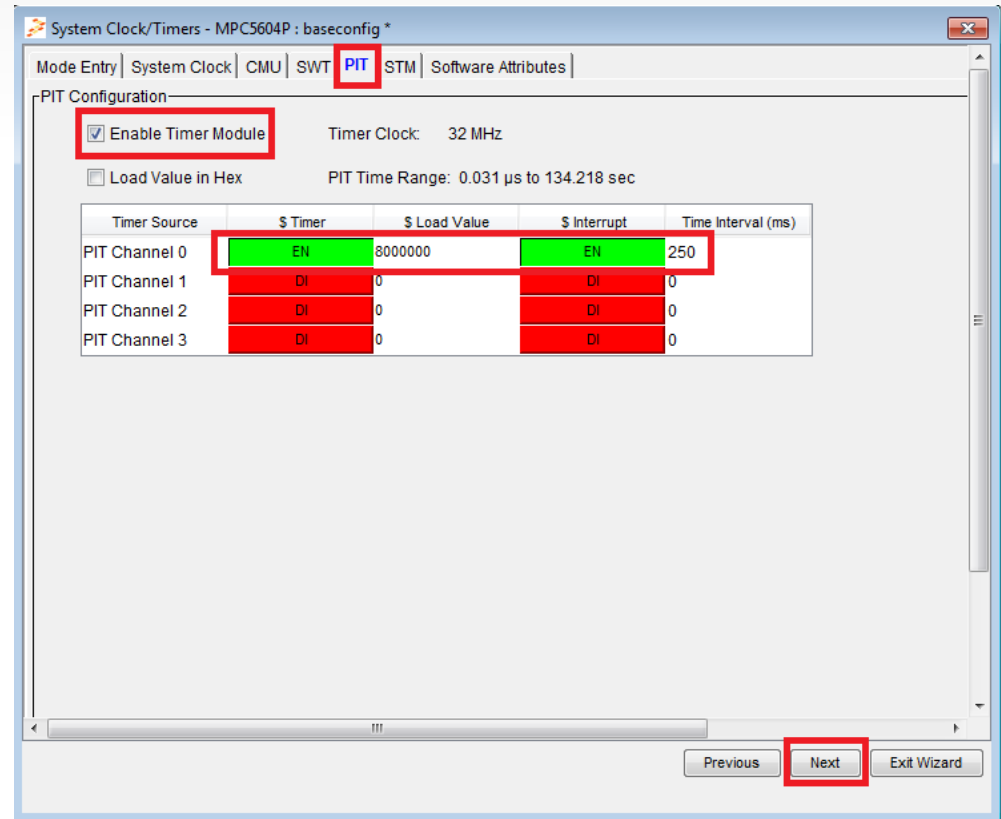




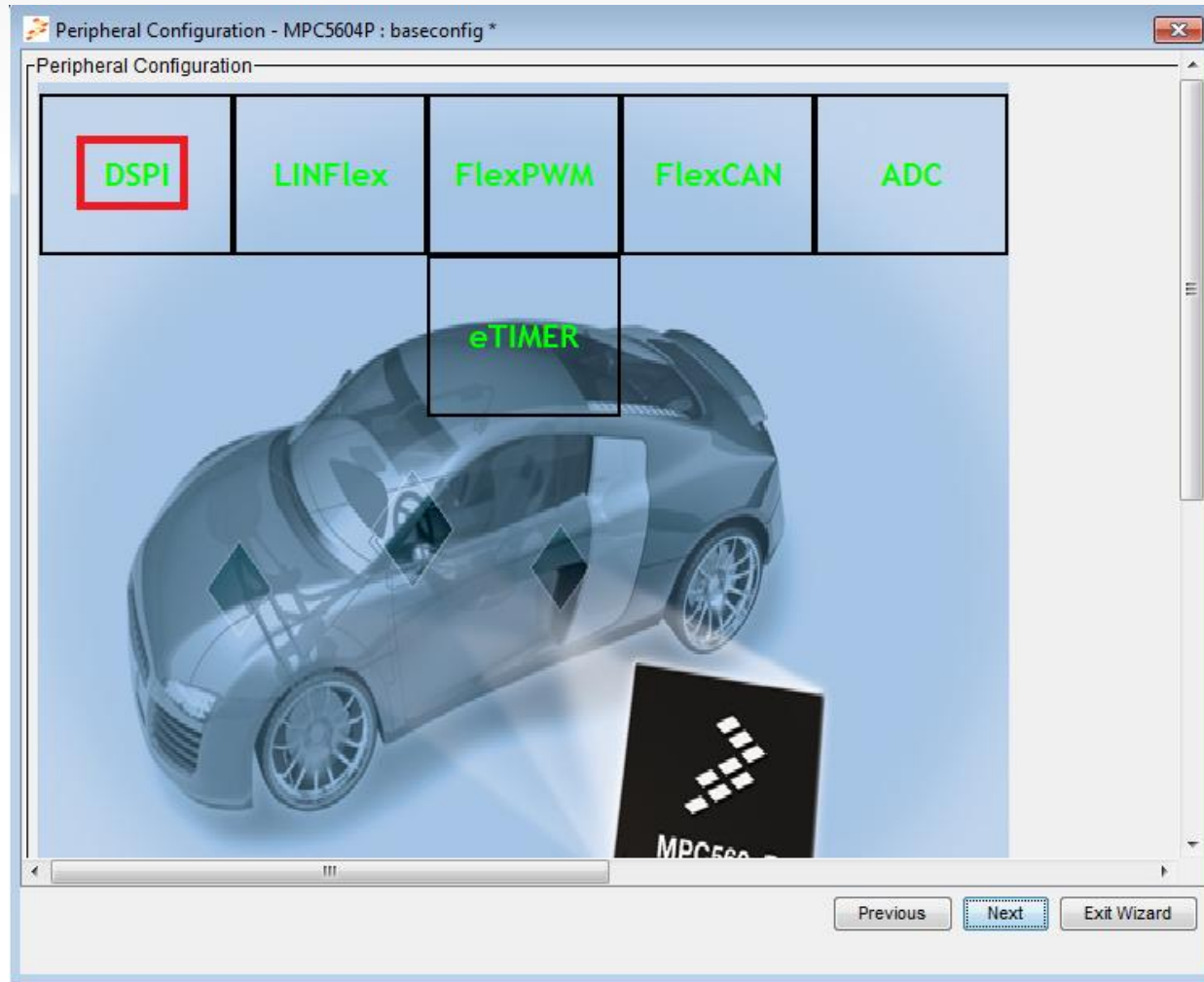
Configure PIT

We will monitor switch inputs S3 and S4 every 250ms in PIT Channel 0 interrupt. Select PIT tab and

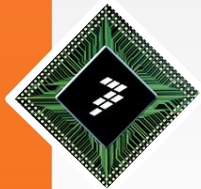
- Enable Timer module
- Enable PIT Channel 0 timer
- Set load value to 8000000 to set the time out value to 250 ms
- Enable PIT Channel 0 interrupt
- Select Next to start peripheral configuration.



Configure Peripherals



In Peripheral Configuration window, select DSPI to configure DSPI peripheral



Configure DSPI 1

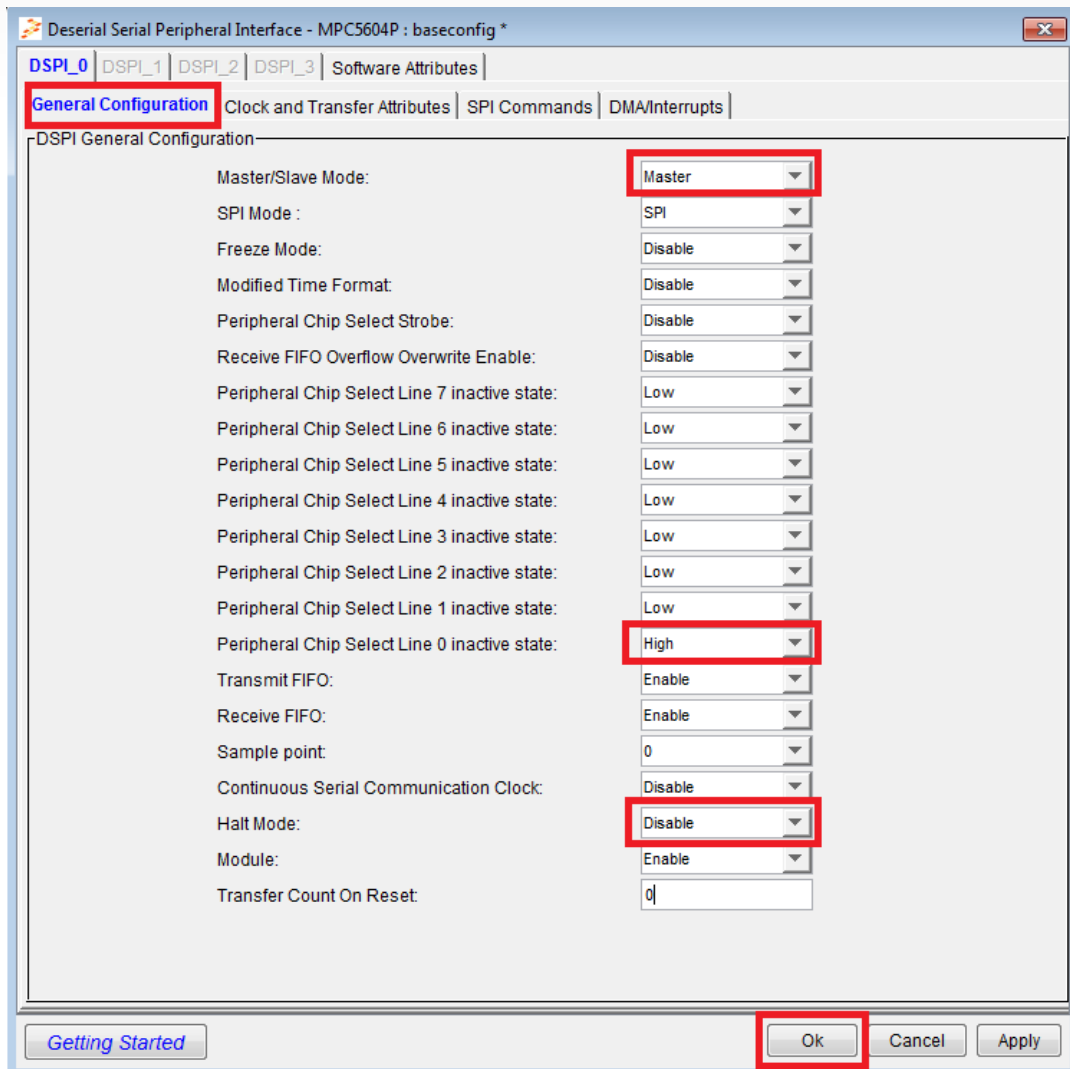
DSPI 0 should be set to master mode to send commands to SBC. Select Master mode

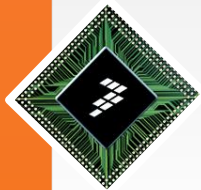
DSPI 0 Chip Select 0 is connected to SBC. Set Chip select 0 inactive state to High

Disable Halt mode

Select OK to finish DSPI 0 configuration.

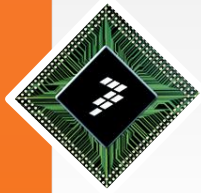
In the Peripheral Configuration window, select LinFlex tile to start LinFlex configuration.





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.

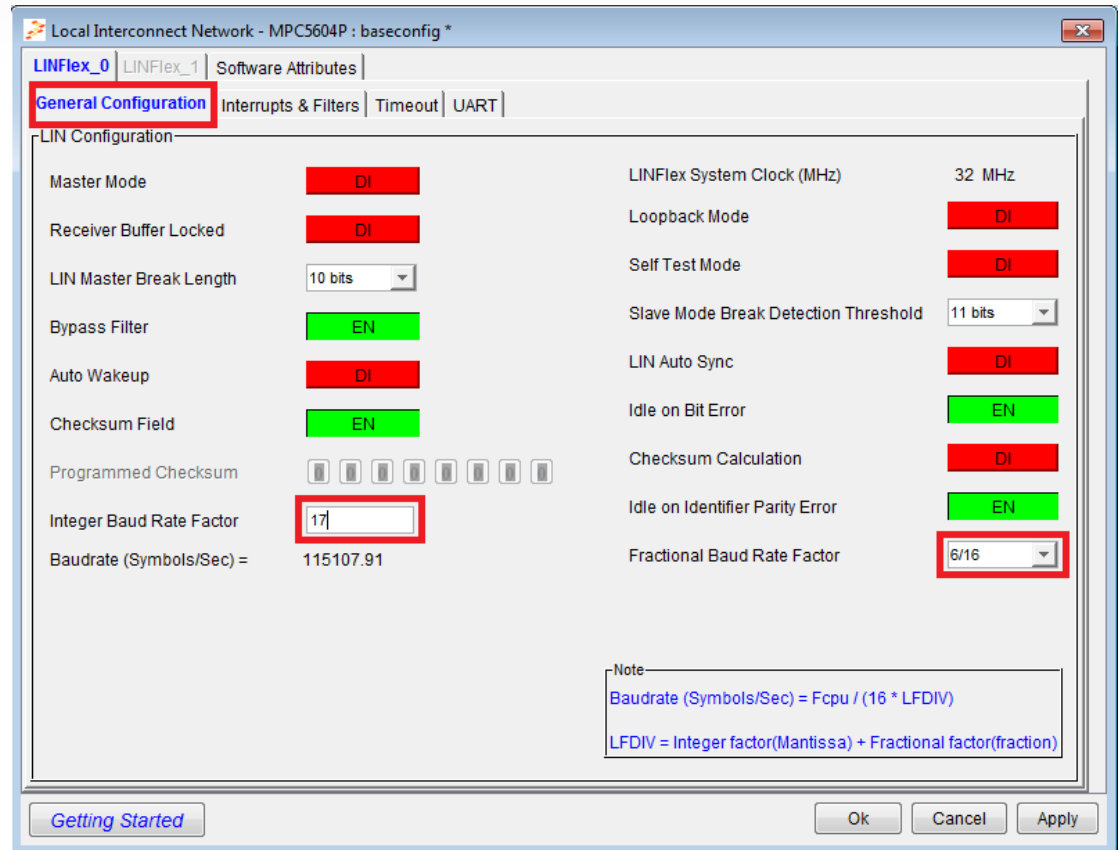


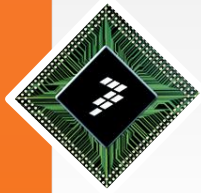
Configure LINFlex 0

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

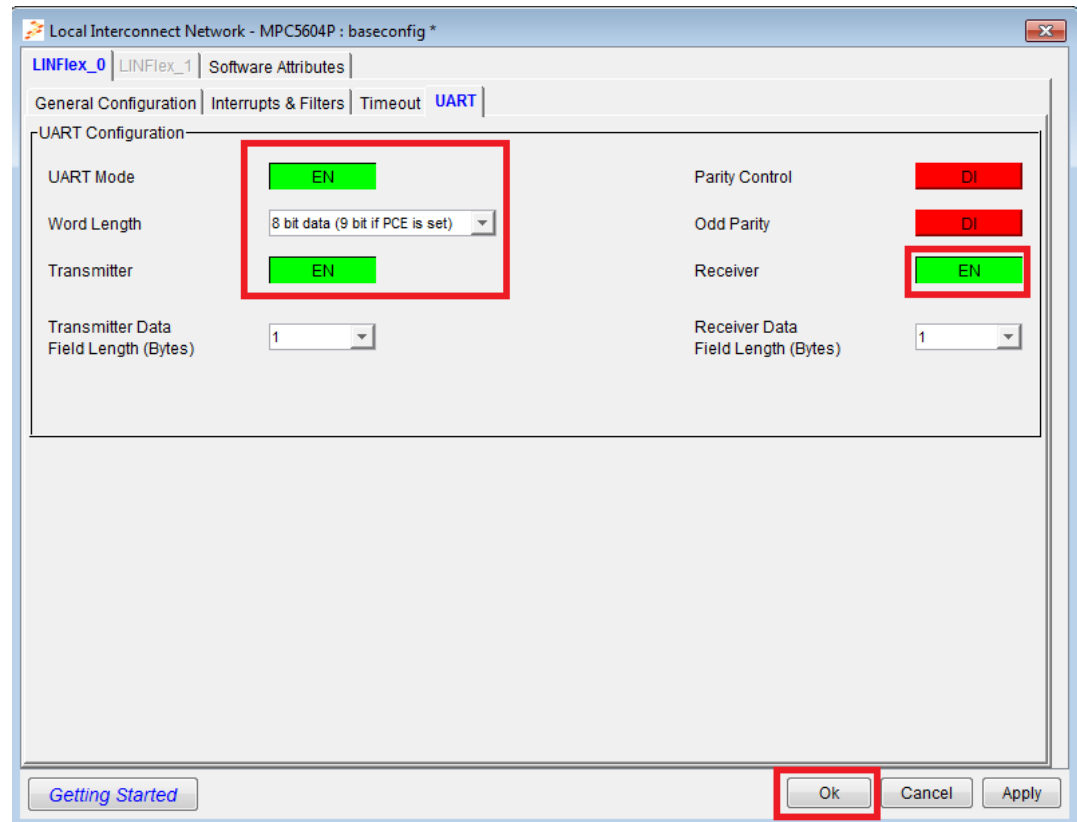
In UART tab, Enable UART

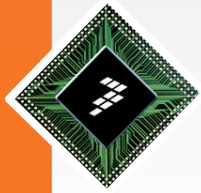
Set Word Length to 8 bit data

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

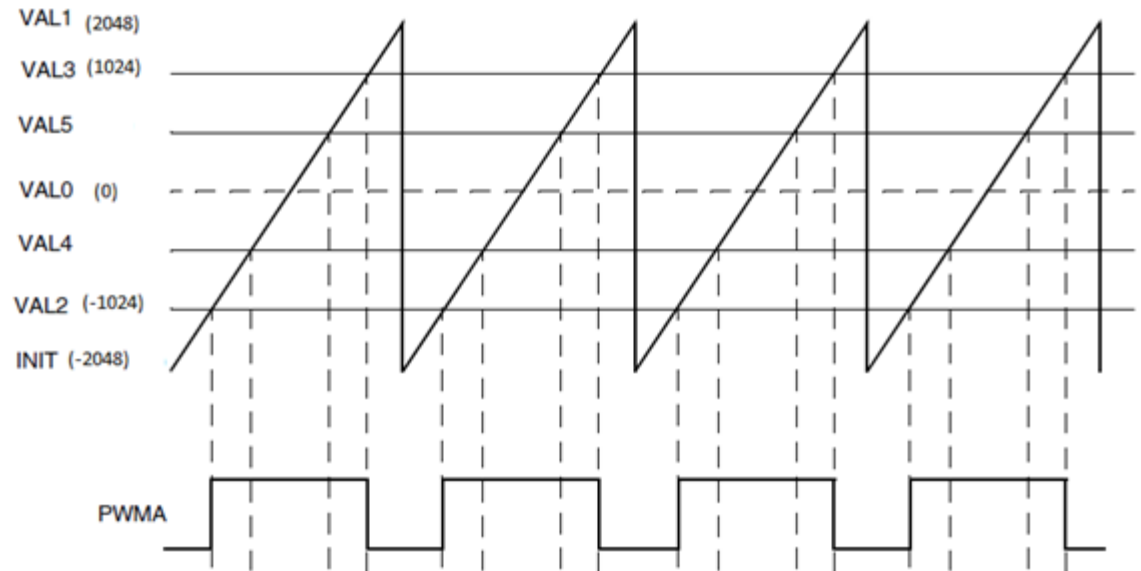
In this example, we will configure FlexPWM sub module 0, PWM A output to generate center-aligned PWM signals. The values of FlexPWM parameters are set as shown.

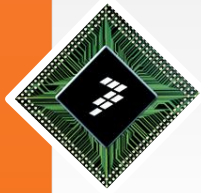
The PWM clock source is 16 MHz IRC clock.

The Period of PWM signal = $(2048 + 2048) * 1/16 = 256 \text{ ms}$

The PWM signal high time = $(1024+1024) * 1/16 = 128 \text{ ms.}$

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

The screenshot shows the FlexPWM configuration interface for MPC5604P. The 'General Configuration' tab is active. The 'O/P Enable & Mask Configuration' section shows a grid where 'EN' is selected for PWM A O/P at SubModule 0. The 'Master Control Configuration' section shows 'Run (PWM Generator)' set to 'EN' and 'Load to Buffer' set to '1'. The 'Fault Control Configuration' section shows 'Fault Interrupt' set to 'DI' for all submodules. The 'Deadtime Source & S/W Ctrl O/P Configuration' section shows 'Generated PWM' selected for all PWMxy_z outputs. The 'Filter Control Configuration' section shows 'Filter Count (Samples)' set to 3 and 'Filter Period' set to 0.

	0	1	2	3
PWM A O/P	EN	DI	DI	DI
PWM B O/P	DI	DI	DI	DI
PWM X O/P	DI	DI	DI	DI
PWM A Mask	DI	DI	DI	DI
PWM B Mask	DI	DI	DI	DI
PWM X Mask	DI	DI	DI	DI

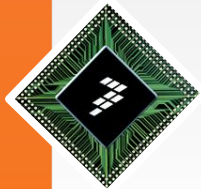
	0	1	2	3
Run (PWM Generator)	EN	DI	DI	DI
Load to Buffer	1	0	0	0
Current Polarity (PWM)	A	A	A	A

	0	1	2	3
Fault Interrupt	DI	DI	DI	DI
Fault Safe Mode	DI	DI	DI	DI
Automatic Fault Clearing	DI	DI	DI	DI
Fault Level	0	0	0	0

PWMxy_z	Deadtime source	S/W Controlled O/P
A_3	Generated PWM	DI
B_3	Generated PWM	DI
A_2	Generated PWM	DI
B_2	Generated PWM	DI
A_1	Generated PWM	DI
B_1	Generated PWM	DI
A_0	Generated PWM	DI
B_0	Generated PWM	DI

xy=PWMxy
z=Submodule number

	0	1	2	3
Filter Count (Samples)	3			
Filter Period	0			



Configure FlexPWM

In Configuration 1 tab

-Set INIT counter value to -2048

-Set Pair Operation to Independent PWM

FlexPWM - MPC5604P : baseconfig *

FlexPWM_0 | Software Attributes

General Configuration **Configuration 1** | Configuration 2 | Configuration 3 (Interrupt/DMA) | Configuration 4

Sub Module Configuration

CH	INIT	DBL	PRSC	FULL	HALF	LDFQ	CLK	RLD	FORCE	FORCEN	INITSEL	PWMX	PWMB	PWMA	INDEP	WAIT
0	-2048	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

Configuration 1

Initialize Counter (INIT)

DoubleSwitching (DBL)

Prescaler (PRSC) 16 MHz

Full Cycle Reload (FULL)

Half Cycle Reload (HALF)

Load Frequency (LDFQ)

Clock Source (CLK)

Reload Source (RLD)

Force Source(FORCE)

System Clock Frequency(fclk) 16 MHz

Force Initialization (FORCEN)

Initialization Control (INITSEL)

PWMX Init(PWMX)

PWMB Init (PWMB)

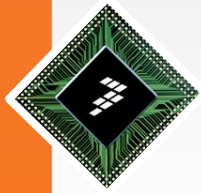
PWMA Init (PWMA)

Pair Operation (INDEP)

Wait (WAIT)

Debug (DBG)

Getting Started



Configure FlexPWM

In Configuration 2 tab

-Set MAX counter value to 2048

-Set PWM A High to -1024

-Set PWM A Low to 1024

FlexPWM - MPC5604P : baseconfig *

FlexPWM_0 | Software Attributes | **Configuration 2** | Configuration 3 (Interrupt/DMA) | Configuration 4

Sub Module Configuration

CH	MID	MAX	PWMAH	PWMAL	PWMBH	PWMBL	POLA	POLB	POLX	PWMAFS	PWMBFS	PWMXFS	V0	V1	V2	V3	V4	V5
0	0	2048	-1024	1024	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Configuration2

Mid Cycle Reload Point (MID) PWM A O/P Polarity (POLA)

Maximum Count (MAX) PWM B O/P Polarity (POLB)

PWM A High (PWMAH) PWM X O/P Polarity (POLX)

PWM A Low (PWMAL) PWM A O/P Fault State (PWMAFS)

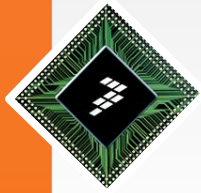
PWM B High (PWMBH) PWM B O/P Fault State (PWMBFS)

PWM B Low (PWMBL) PWM X O/P Fault State (PWMXFS)

VAL: V0 V1 V2 V3 V4 V5

Output Trigger Control VAL

Getting Started



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.

FlexPWM - MPC5604P : baseconfig *

FlexPWM_0 | Software Attributes

General Configuration | Configuration 1 | Configuration 2 | Configuration 3 (Interrupt/DMA) | **Configuration 4**

Sub Module Configuration

CH	DISA	DISB	DISX	DTCNT0	DTCNT1	CAPMD	EDGX0	EDGX1	INPSELX	EDGCNTX	CFX0WM	CFX1WM	EDGCMPIX	ARMX
0	0	15	15	0	2047	0	0	0	0	0	0	0	0	0
1	15	15	15	2047	2047	0	0	0	0	0	0	0	0	0
2	15	15	15	2047	2047	0	0	0	0	0	0	0	0	0
3	15	15	15	2047	2047	0	0	0	0	0	0	0	0	0

Configuration4

PWM A Fault Mask (DISA) **0000**

PWM B Fault Mask (DISB) 1111

PWM X Fault Mask (DISX) 1111

PWM A Dead Count (DTCNT0) **0**

PWM B Dead Count (DTCNT1) 2047

Capture Mode (CAPMD) Free Running

Input EdgeX0 Select (EDGX0) Disabled

Input EdgeX1 Select (EDGX1) Disabled

Input Select X (INPSELX) Raw PWMX

Edge Counter X (EDGCNTX) DI

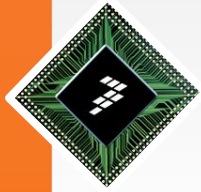
Capture X0 FIFOs Water Mark (CFX0WM) DI

Capture X1 FIFOs Water Mark (CFX1WM) DI

Edge CompareX (EDGCMPIX) 0

Input Capture Mode (ARMX) DI

Getting Started **Ok** Cancel Apply



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

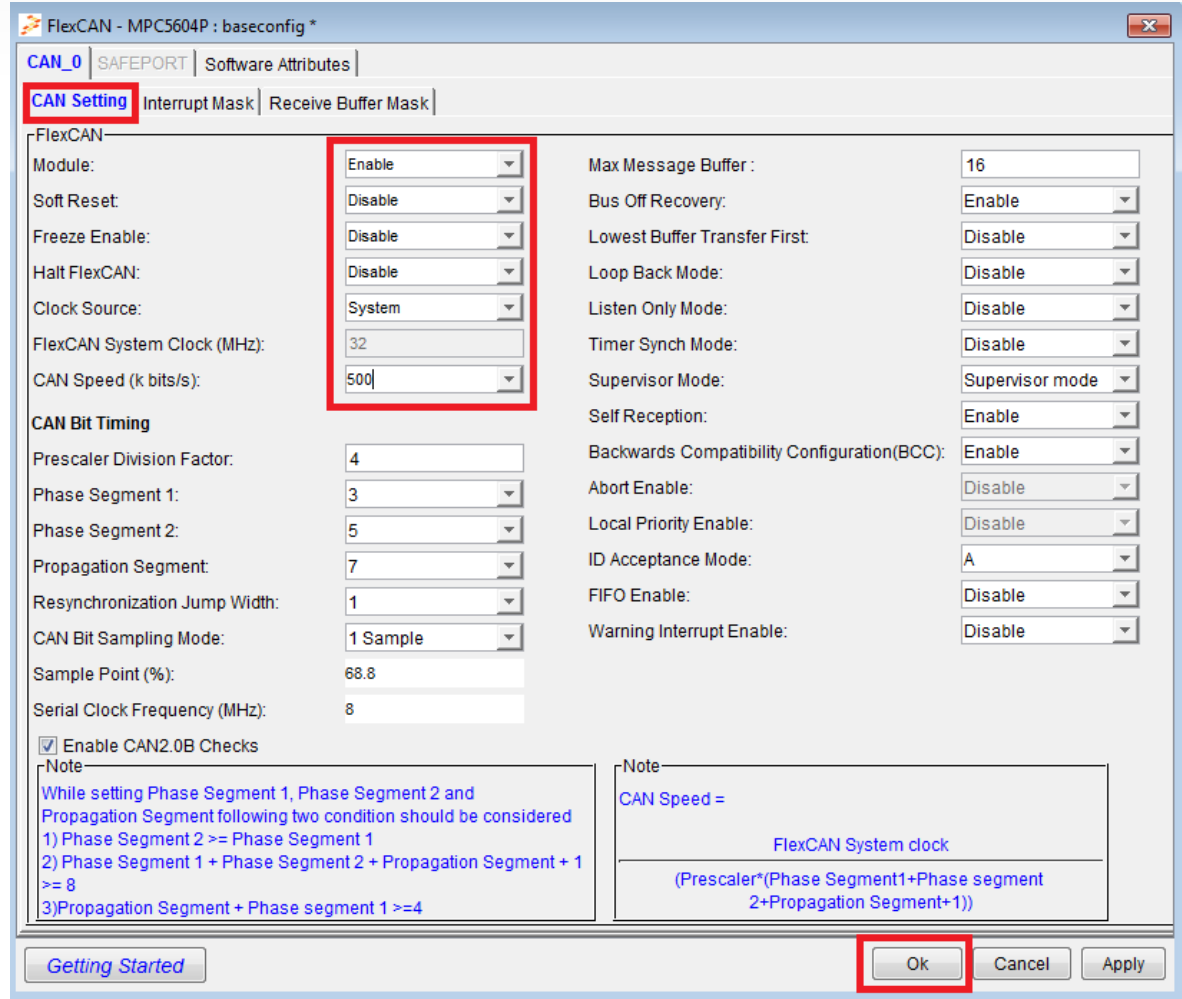
Disable Freeze and Halt modes

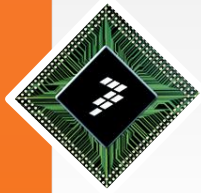
Set Clock Source to System

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

Select OK to finish CAN configuration.

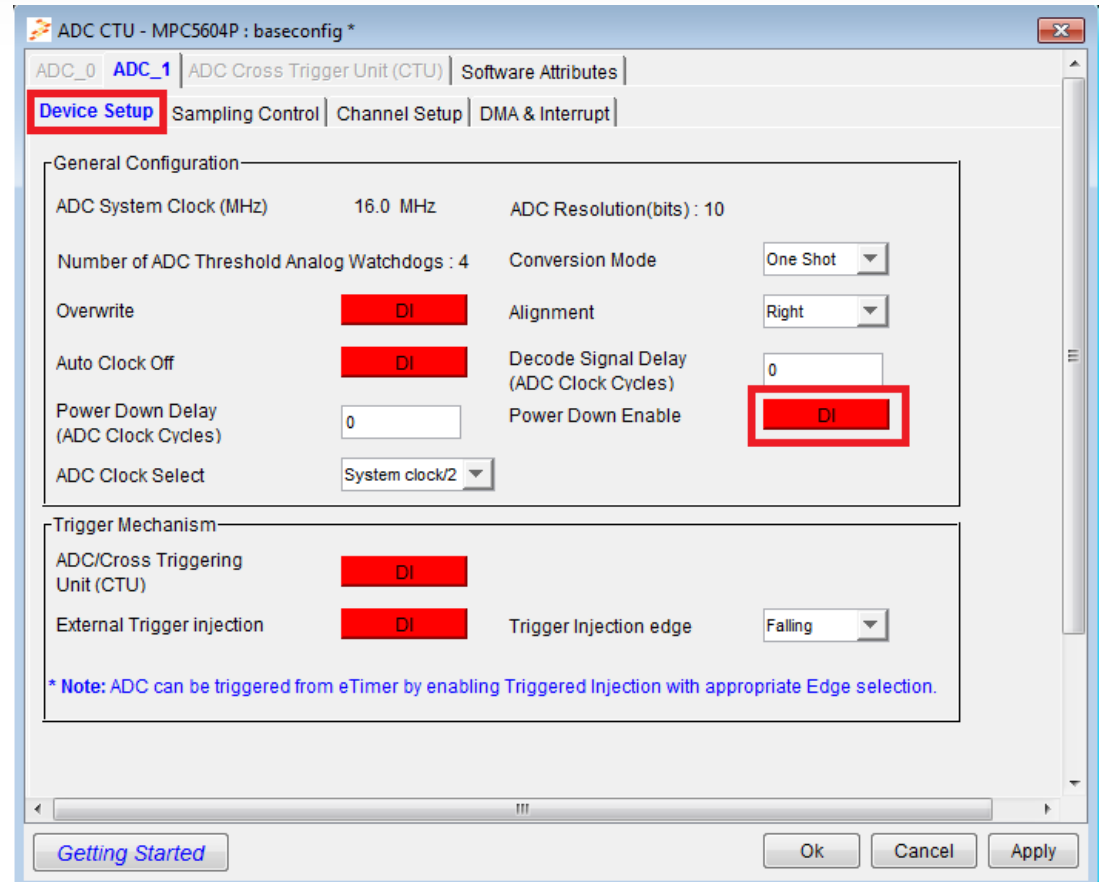
In the Peripheral Configuration window, select ADC tile to start ADC configuration.

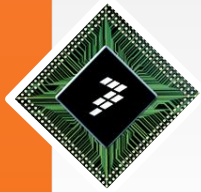




Configure ADC 1

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.

ADC CTU - MPC5604P : baseconfig *

ADC_0 | **ADC_1** | ADC Cross Trigger Unit (CTU) | Software Attributes

Device Setup | Sampling Control | **Channel Setup** | DMA & Interrupt

ADC Conversion Mode—
Normal Mode

Channel Number

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
DI DI DI DI DI **EN** DI DI DI DI DI DI DI DI DI

Injected Mode

Channel Number

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
DI DI DI DI DI DI DI DI DI DI DI DI DI DI DI

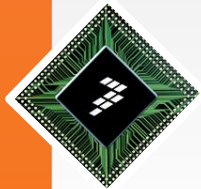
For Normal and Injected Channel, this Indicates Pin selection for this channels

Threshold Configuration

Load in He

ADC Threshold WatchDog	\$ Threshold	\$ Threshold Inversion	\$ Channel	\$ Low Threshold	\$ High Threshold	\$ Low INT	\$ High INT
0	DI	DI	0	0	1023	DI	DI
1	DI	DI	0	0	1023	DI	DI
2	DI	DI	0	0	1023	DI	DI
3	DI	DI	0	0	1023	DI	DI

Getting Started | **Ok** | Cancel | Apply

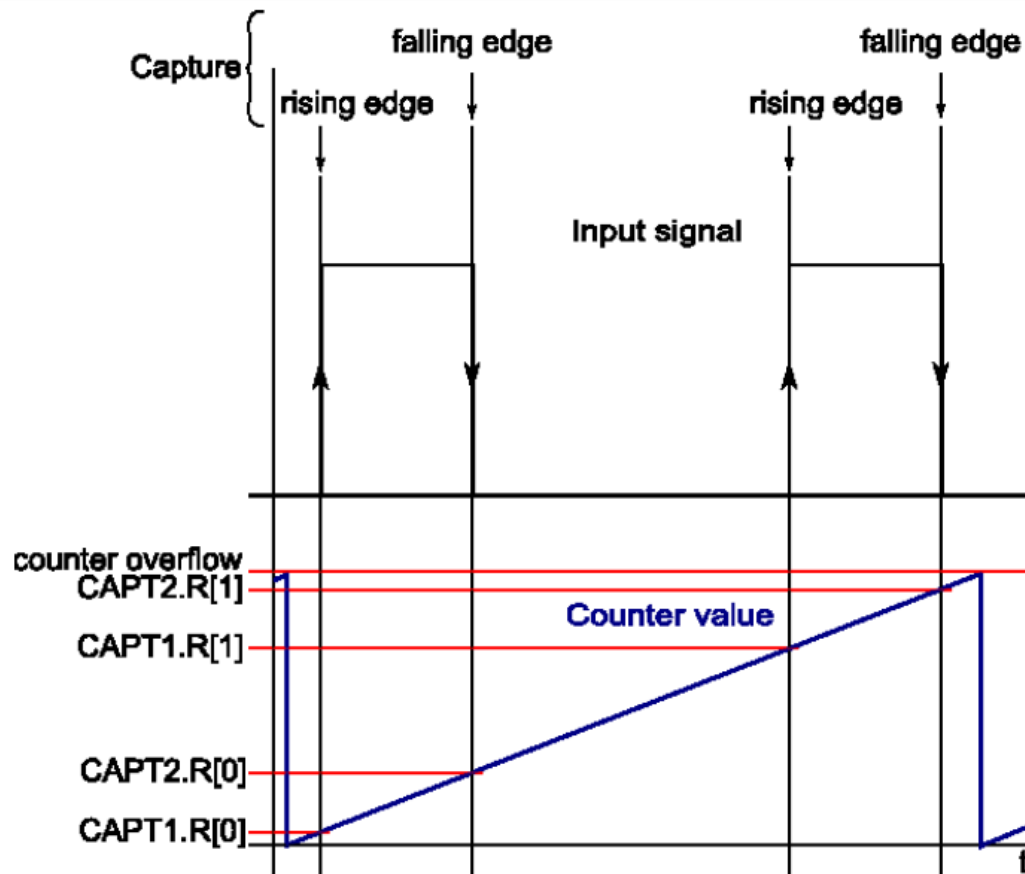


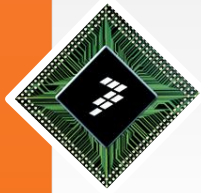
Configure eTimer

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.

The screenshot shows the 'eTimer Configuration - MPC5604P : baseconfig' window. The 'eTimer Configuration 1' tab is active. A table at the top lists channel configurations. Below the table, the 'Configuration 1' section contains several settings:

- Counter (ENBL): **EN** (highlighted in green)
- Initialize Counter (LOAD): 0x0
- Direction (DIR): Count Up
- Secondary Source (SECSRC): **Counter 0 IP** (highlighted in red)
- Count Mode (CNTMODE): **Count rising edges of primary source** (highlighted in red)
- Secondary I/P Polarity (SIPS): True
- Reload on Capture (ROC): Do not reload
- Primary Source (PRISRC): **IP Bus clock / 1 prescaler** (highlighted in red)
- Continuous / One shot (ONCE): Continuous
- Primary I/P Polarity (PIPS): True
- Debug Actions (DBGEN): Continue with normal operation during debug mode
- Count Till Compare (LENGTH): **DI** (highlighted in red)
- Stop Actions (STPEN): **DI** (highlighted in red)

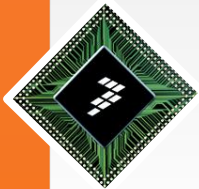
The 'Input Filter Configuration' section shows:

- Sample Count (FILT_CNT): 3
- Sample Period (FILT_PER): 0

The 'Watchdog Timer Configuration (Only for eTimer0)' section shows:

- Timeout High: 0x0
- Timeout Low: 0x0

Buttons at the bottom include 'Getting Started', 'Ok', 'Cancel', and 'Apply'.



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.

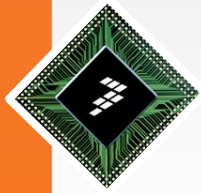
The screenshot shows the 'eTimer Configuration - MPC5604P : baseconfig' window. The 'eTimer Configuration 3' tab is selected. The 'Channel Configuration' table is visible, and the 'Configuration 3' section contains the following settings:

CHNL	CMPLD1	CPT1MODE	CMPLD2	CPT2MODE	COMP1	COMP2	ONESHOT	ARM	CLC1	CFWM	CLC2	CMPMODE
0	0x0	2	0x0	1	0x0	0x0	0	0	0	2	0	0
1	0x0	0	0x0	0	0x0	0x0	0	0	0	0	0	0
2	0x0	0	0x0	0	0x0	0x0	0	0	0	0	0	0
3	0x0	0	0x0	0	0x0	0x0	0	0	0	0	0	0
4	0x0	0	0x0	0	0x0	0x0	0	0	0	0	0	0
5	0x0	0	0x0	0	0x0	0x0	0	0	0	0	0	0

Configuration 3 settings:

- Load Comp. 1 (CMPLD1): 0x0
- Load Comp. 2 (CMPLD2): 0x0
- Compare Register 1 (COMP1): 0x0
- Compare Register 2 (COMP2): 0x0
- One Shot (ONESHOT):
- Arm Capture (ARM):
- Capture 1 Mode (CPT1MODE): Capture rising edges.
- Capture 2 Mode (CPT2MODE): Capture falling edges.
- Comp. Load Control 1 (CLC1): Never preload
- Comp. Load Control 2 (CLC2): Never preload
- Compare Mode (CMPMODE): Both CMP1 & CMP2 used when Counting Up
- Capture FIFO Watermark (CFWM): 2

Buttons: Getting Started, Ok, Cancel, Apply



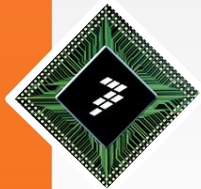
Configure eTimer

The frequency of the input PWM signal can be calculated as follows:

$$f \text{ [kHz]} = \text{motc_clk [kHz]} / (\text{CAPT1.R[1]} - \text{CAPT1.R[0]})$$

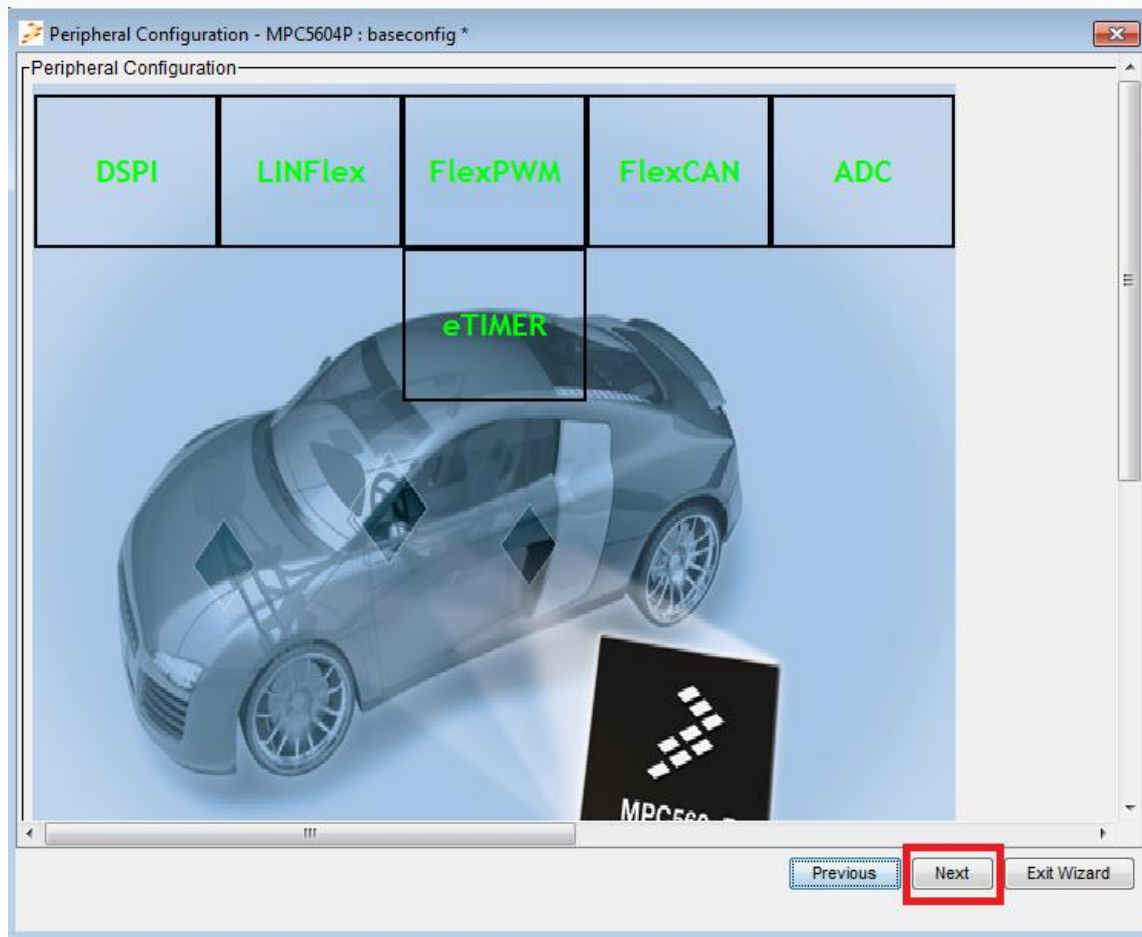
The duty cycle of the input PWM signal can be calculated as follows:

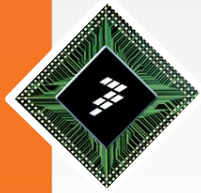
$$\text{duty cycle} = ((\text{CAPT2.R[0]} - \text{CAPT1.R[0]}) * 100) / (\text{CAPT1.R[1]} - \text{CAPT1.R[0]})$$



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.

Interrupt Configuration - MPC5604P : baseconfig *

Interrupt | Exception | Software Attributes

General System Configuration

Core

Select Either Software or Hardware vector mode : Software

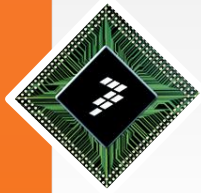
Select the vector Size : 4bytes

Select the interrupt lower than priority level : 0

Select the portion of the INTC you wish to configure below :

Interrupt Source	Interrupt Vector	\$ Priority	Core(s)	\$ Function	\$ ISR Code
SIU	PIT Channel 0	1	e200z0	PIT_Ch0_ISR	Edit
Software	PIT Channel 1	0	e200z0	interrupt_handler	Edit
MCM	PIT Channel 2	0	e200z0	interrupt_handler	Edit
FlexCAN	PIT Channel 3	0	e200z0	interrupt_handler	Edit
eTIMER					
MC_RGM					
FlexPWM					
XOSC					
SWT					
PIT					
DMA					
DSPi					
LINFlex					
STM					
ADC					
MC_ME					

Previous Next Exit Wizard



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.

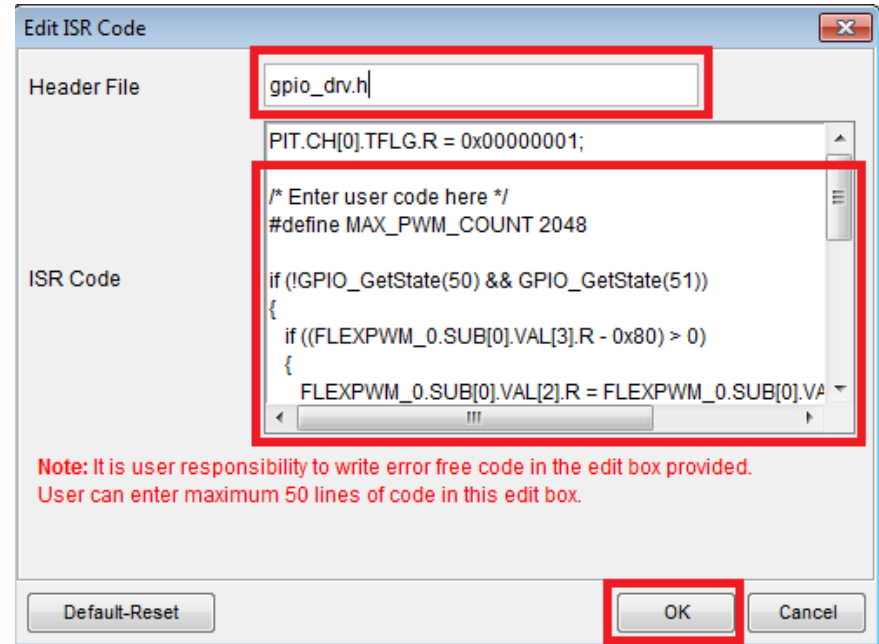
```

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

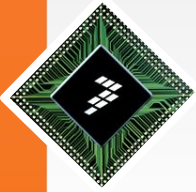
if (!GPIO_GetState(50) && GPIO_GetState(51))
{
    if ((FLEXPWM_0.SUB[0].VAL[3].R - 0x80) > 0)
    {
        FLEXPWM_0.SUB[0].VAL[2].R = FLEXPWM_0.SUB[0].VAL[2].R + 0x80;
        FLEXPWM_0.SUB[0].VAL[3].R = FLEXPWM_0.SUB[0].VAL[3].R - 0x80;
        FLEXPWM_0.MCTRL.B.LDOK = 1;
    }
}
if (GPIO_GetState(50)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

```

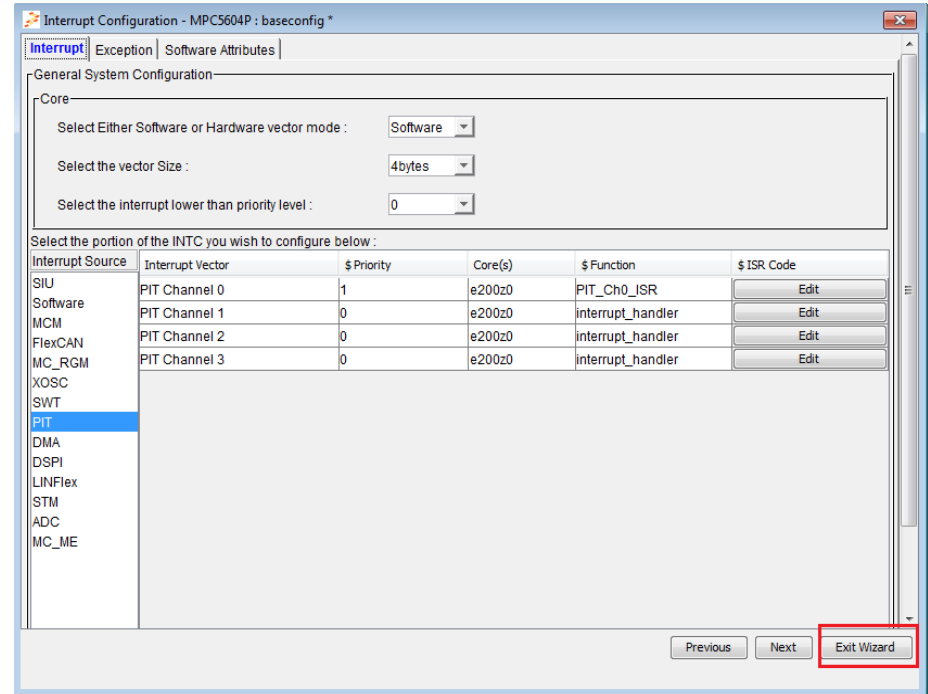
Select OK to exit from Edit Code window.

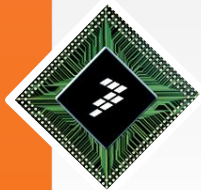


Configure PIT Interrupt



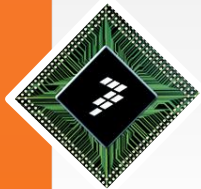
Select Exit Wizard to exit to main RAppID window





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.



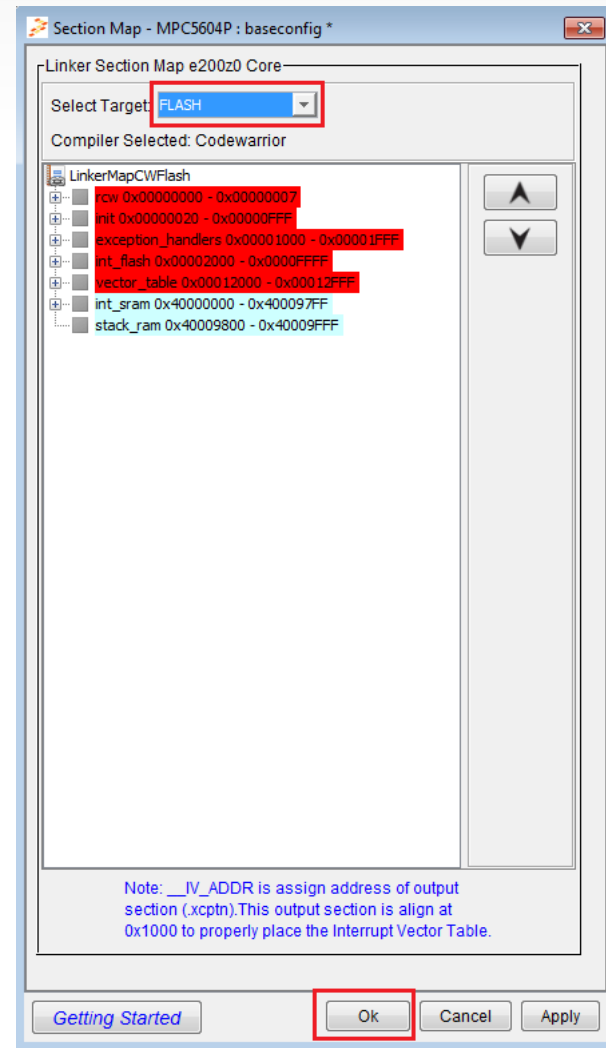
Configure Section Map

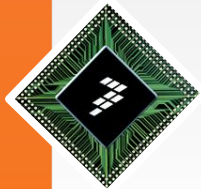
By Default, RAppID is configured to generate code for RAM. For this example, we will change the configuration to generate code for Flash.

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.

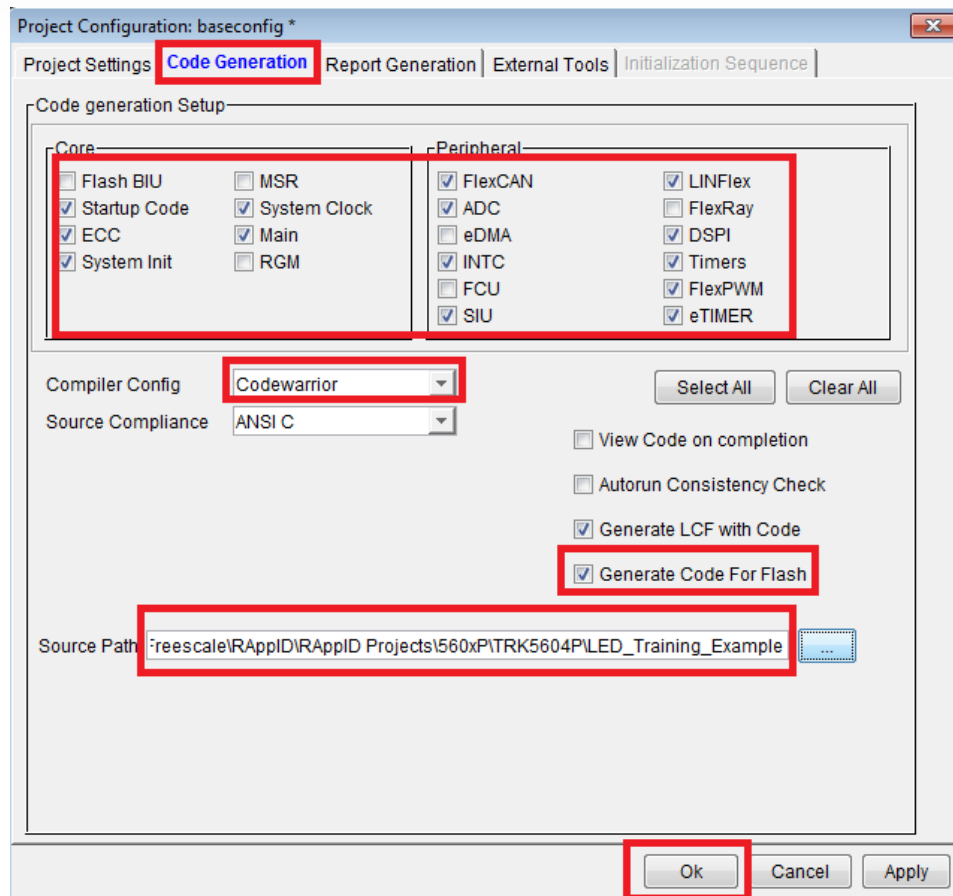
Check ECC for code generation. This is required to generate initialization code for SRAM and ECC register.

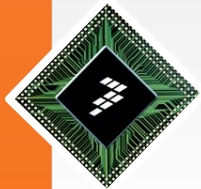
Select CodeWarrior compiler

Select Code for Flash option

Select the directory where code should be generated

Select Ok



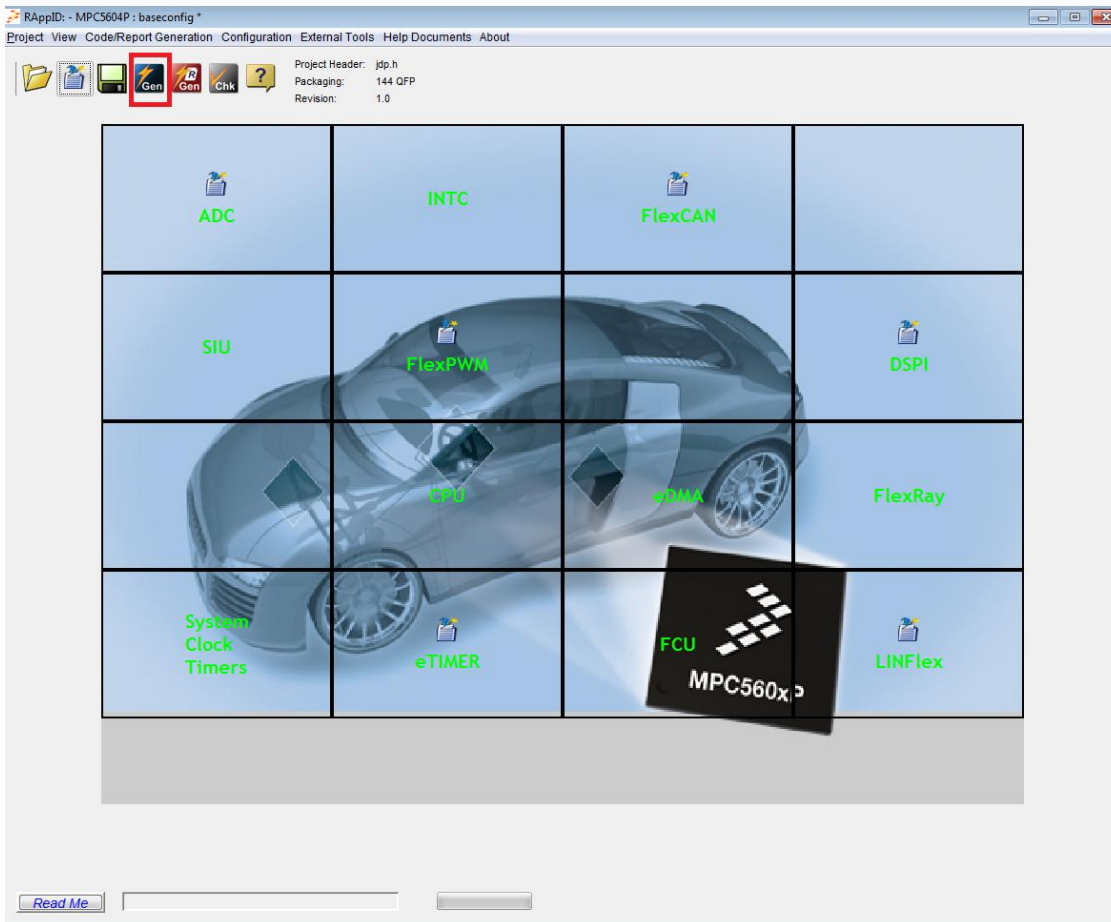


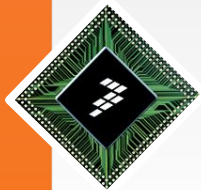
Generate code

Select Code Generation icon to generate code

Save the project when prompted

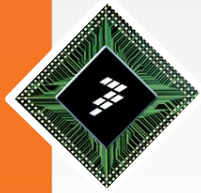
Now we are ready to build and run the project.





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.



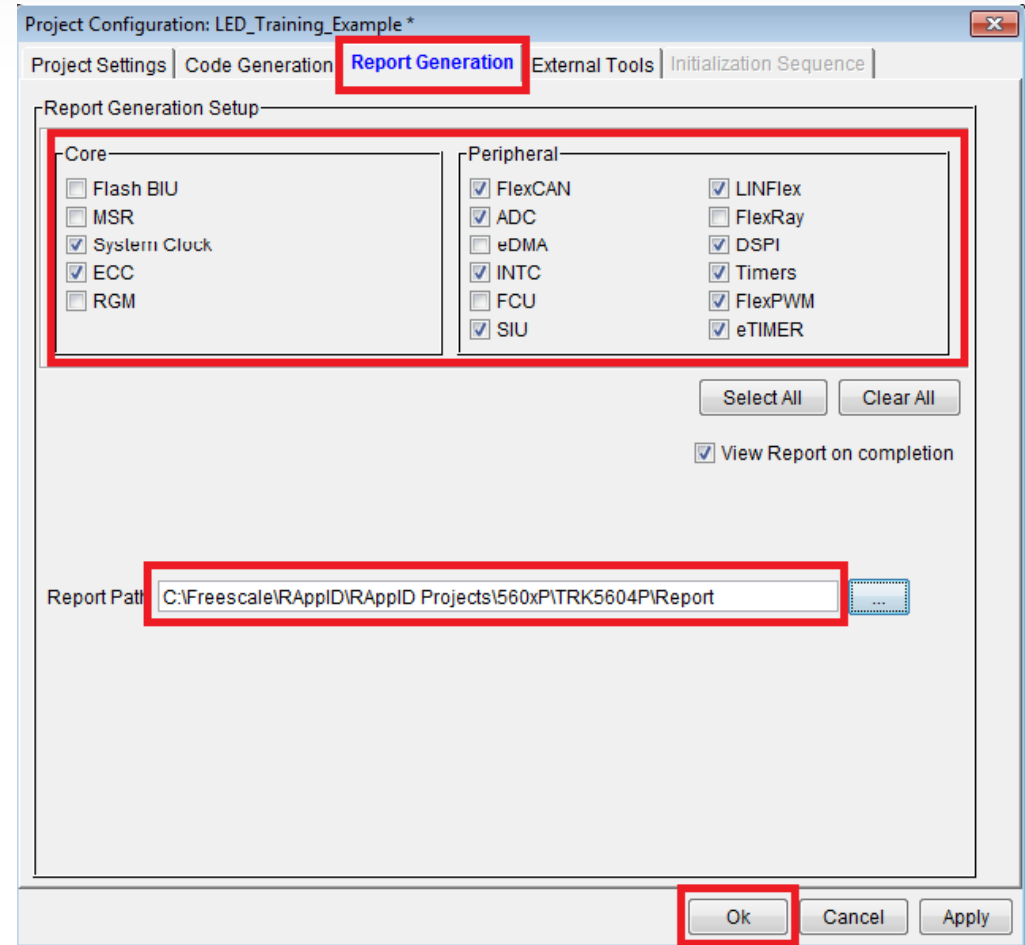
Configure Report Generation

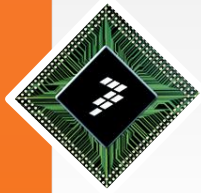
From main RAppID window, select menu Configuration->Report Generation.

Unselect report generation option for unused peripherals

Select the directory where report should be generated

Select Ok

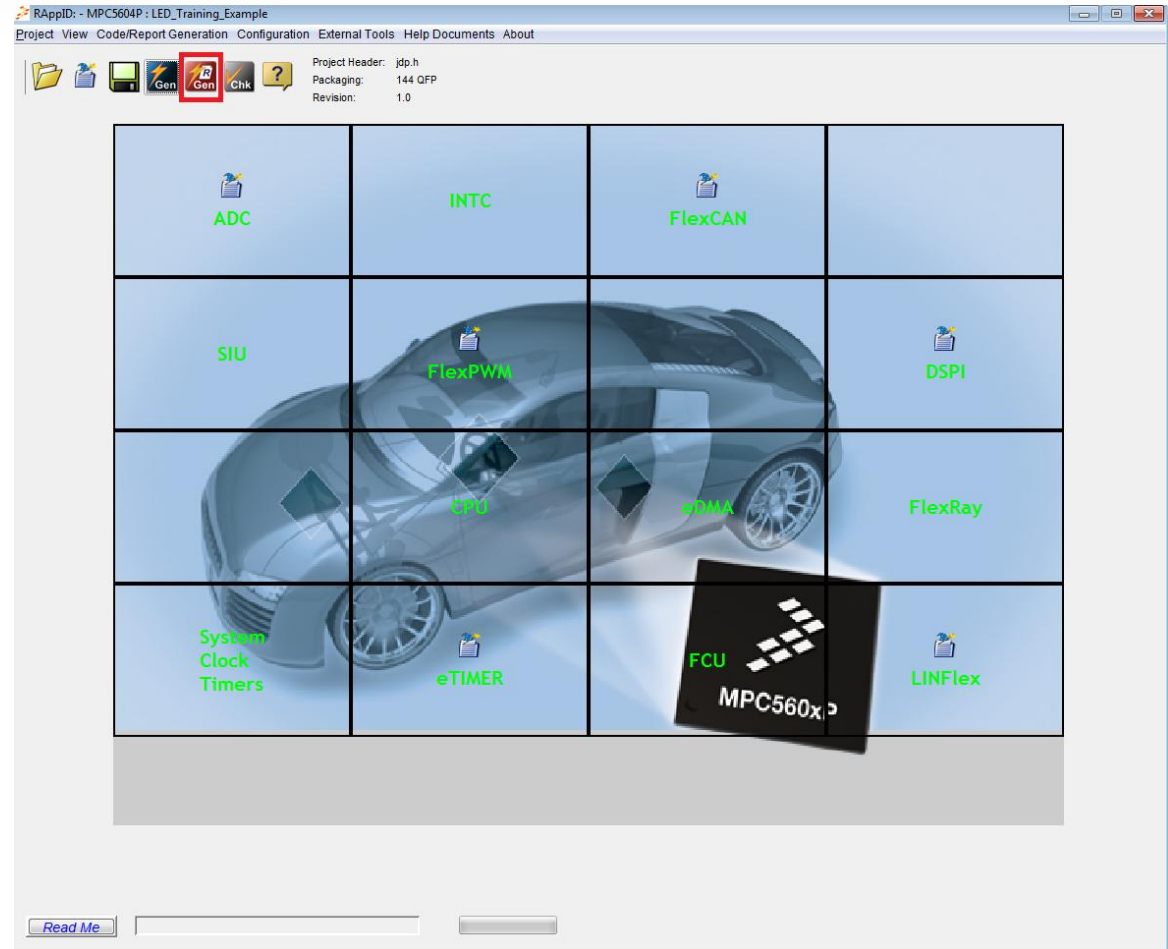


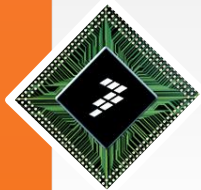


Generate Report

Select Report Generation icon to generate report

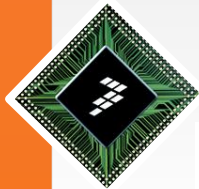
Save the project when prompted





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

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](#)

DSP1

DSP10 Configuration

[General](#)

[Clock Attribute](#)

[SPI Commands](#)

[DMA/Interrupts](#)

DSP11 Configuration

[General](#)

[Clock Attribute](#)

[SPI Commands](#)

[DMA/Interrupts](#)

DSP12 Configuration

[General](#)

[Clock Attribute](#)

[SPI Commands](#)

[DMA/Interrupts](#)

Configuration Report for MPC5604P Package: 144 QFP



Project Name: baseconfig (Revision:1.0)

Compiler Compliance: Codewarrior, ANSI C

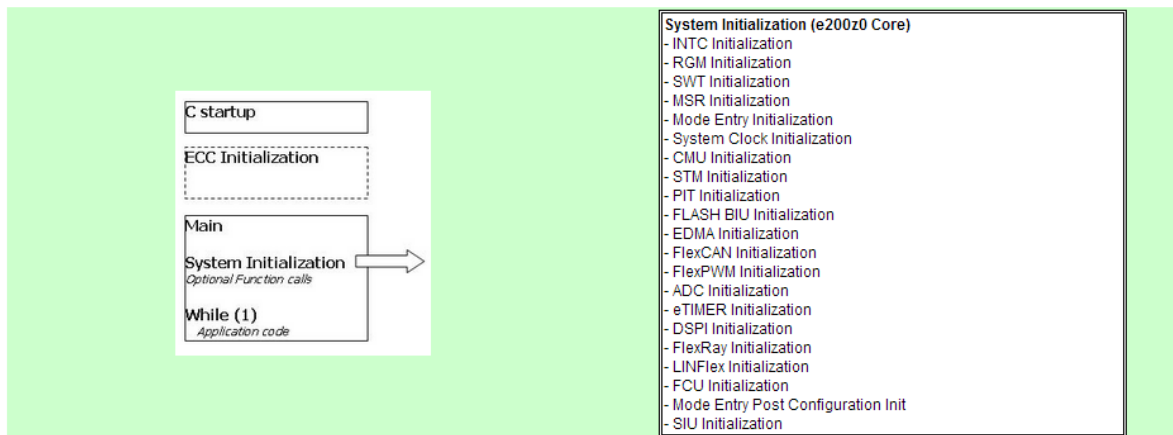
Note: RAppID bit description is in Motorola (Big Endian) format.

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

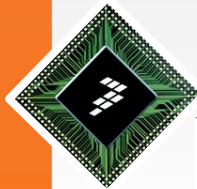
Tool Version: 1.4.0.4

Device Settings for: Initialization Technique

System Main Initialization



Peripheral Initialization



Project Report – Pin Allocation

[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

DSPIO 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](#)

Configuration Report for MPC5604P Package: 144 QFP



Project Name: baseconfig (Revision :1.0)

Compiler Compliance: Codewarrior, ANSI C

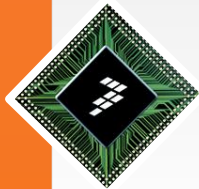
Note: RAppID bit description is in Motorola (Big Endian) format.

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

Tool Version: 1.4.0.4

Function Assignment for MPC5604P 144 QFP package

STU PCR#	Package Pin #	Port Name	Primary Function		Alternate Function 1		Alternate Function 2		Alternate Function 3		Input Function 0		Input Function 1		Analog Pad Control		Assigned Function		
			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



Project Name: baseconfig (Revision :1.0)
Compiler Compliancy: Codewarrior, ANSI C

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

Note: RAppID bit description is in Motorola (Big Endian) format.

Section Map Report

Selected Compiler: Codewarrior
Selected Target: FLASH

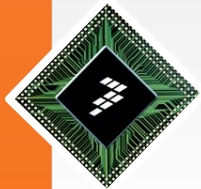
Linker Section Map e200z0 Core

LinkerMapCWFflash			
->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	__bam_bootarea		

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

exception_handlers			
Group	O/P Section	I/P Section	File
Group: 1	exception_handlers		



Project Report – Peripheral Configuration



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](#)

DSP1

DSP10 Configuration

- [General](#)
- [Clock Attribute](#)
- [SPI Commands](#)
- [DMA/Interrupts](#)

DSP11 Configuration

- [General](#)
- [Clock Attribute](#)
- [SPI Commands](#)
- [DMA/Interrupts](#)

DSP12 Configuration

- [General](#)
- [Clock Attribute](#)
- [SPI Commands](#)
- [DMA/Interrupts](#)

DSP13 Configuration

- [General](#)
- [Clock Attribute](#)
- [SPI Commands](#)

Configuration Report for MPC5604P Package: 144 QFP

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

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](#) align also old conversion data will be [not overwritten](#) by newer result, hence newer result [will](#) be discarded.

Delay between external decode signal and start of sampling pulse is set to [0](#) clock cycles. Delay between external power signal and start of sampling pulse is set to [0](#) clock cycles.

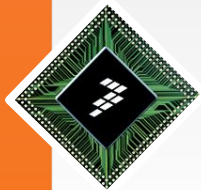
ADC Clock selected as [System Clock/2](#).

Cross Triggering unit [Disabled](#).

Triggered Injection is [Disabled](#), with Trigger Injection edge selected as [Falling](#).

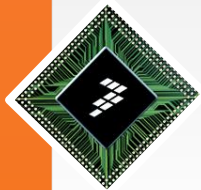
Register

ADC_0_MCR															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OWR EN	VL SIZE	MODE	EDG LEV	TRIGEN	EDGE	XSTRT EN	N START	Res.	JTRG EN	JEDGE	J START	Reserved	CTUVEN	Res.	
rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw		rw		
Reserved								ADCLK SEL	ABORT CHAIN	ABORT	ACKO	OFF DE #RESH	OFF CANC	Reserved	PWDN
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31



Adding Low and high level Driver code

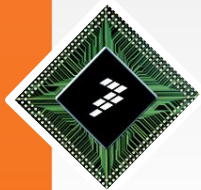
- We will use driver code supplied with the installation media to perform low level functions. Copy the driver code to the directory where RAppID code was generated



Adding FreeMASTER code

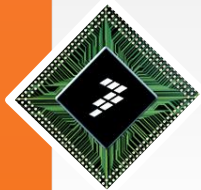
- We will use FreeMASTER utility to monitor global variables. To add this ability, we have to add FreeMASTER code which is provided with the installation media.
- Copy all the code from sub-folders *src_common* and *src_platformsMPC56xx* to the location where RAppID code is generated.
- FreeMASTER can 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

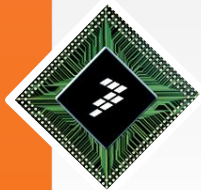


Edit main.c – Global variables

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"
```

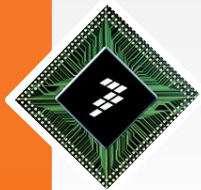
Add following code to *main* function:

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);
```



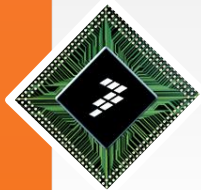
Edit main function (Continued)

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:

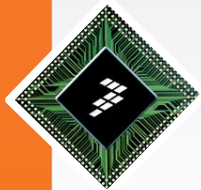
```
FMSTR_Init();
```

Edit main function (Continued)

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

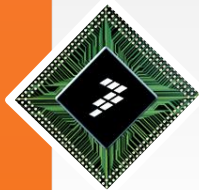
```
while(1)
{
    FMSTR_Poll();
    ProcessGPIO();
    ProcessADC();
    ProcessCAN();
    CalcDC();
}
```



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.

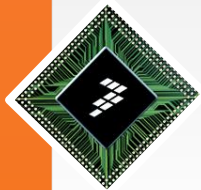
If the first data byte is either 0 or 1, it will transmit CAN message ***msgOKCAN***, other wise it will transmit CAN message ***msgErrorCAN***. The code is shown below:



Add ProcessCAN function (Continued)

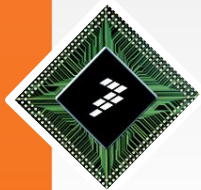
```
void ProcessCAN(void)
{
    can_msg_struct msgCanRX;

    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)

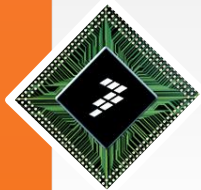
```
else          /* If received data byte is not 0 or 1, send a negative response */
{
    CanTxMsg(2, 1, 8, (uint8_t *)msgErrorCAN, 0);
}
}
}
```



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:

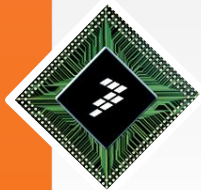
```
void ProcessADC(void)
{
    potValue = Pot_Get_Value();
    if(potValue <= 512)    /* If Potentiometer input is <= 512 turn on LED2, other wise turn off LED2 */
    {
        GPIO_SetState(53, 0);
    }
    else
    {
        GPIO_SetState(53, 1);
    }
}
```



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);
    }
}
```

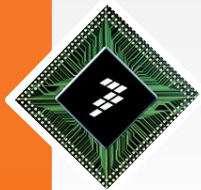


Add CalcDC function

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:

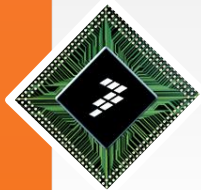
```
void CalcDC(void)
{
    vuint32_t PWMPeriod;
    vuint16_t measure[4];

    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
```



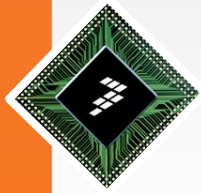
Add CalcDC function (Continued)

```
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

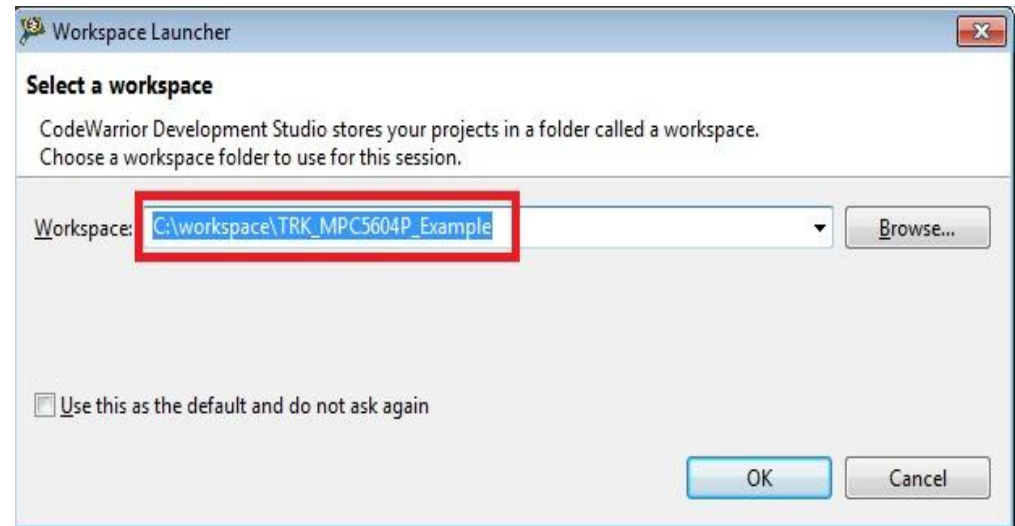


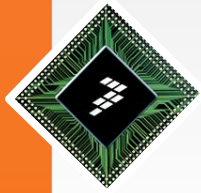
Create CodeWarrior project

Launch CodeWarrior 10.5 from Windows Start menu

Provide a workspace name

Select OK

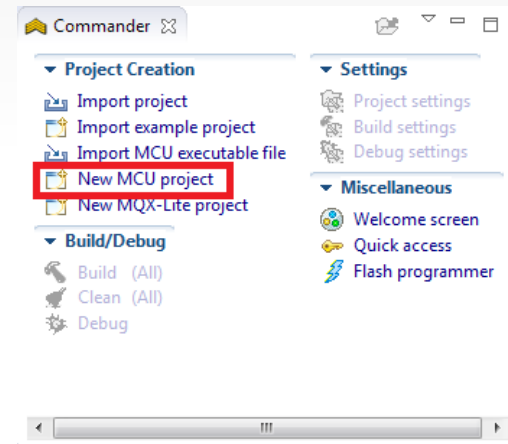


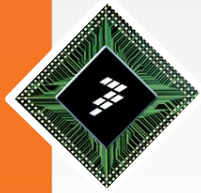


Create CodeWarrior project

We will first create a Bareboard project for MPC5604P

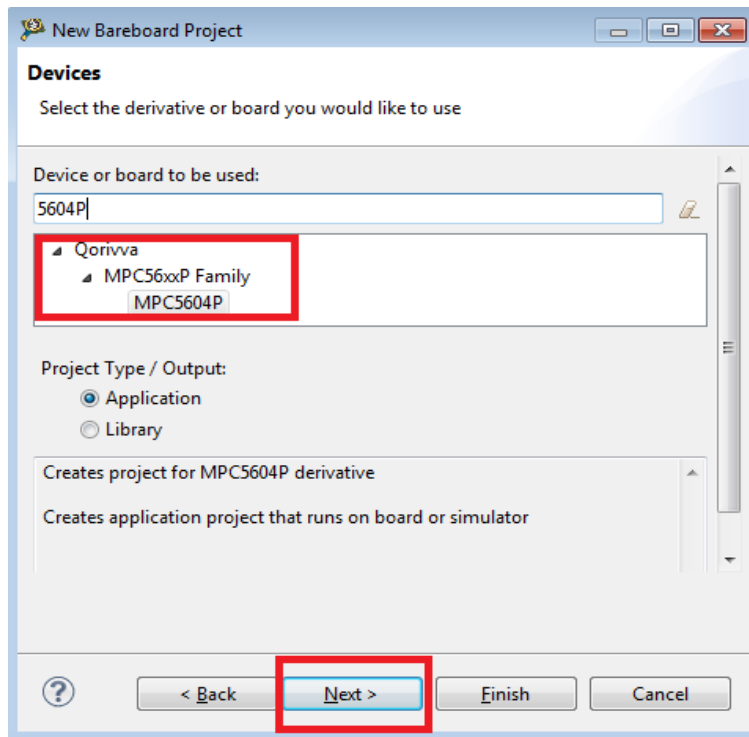
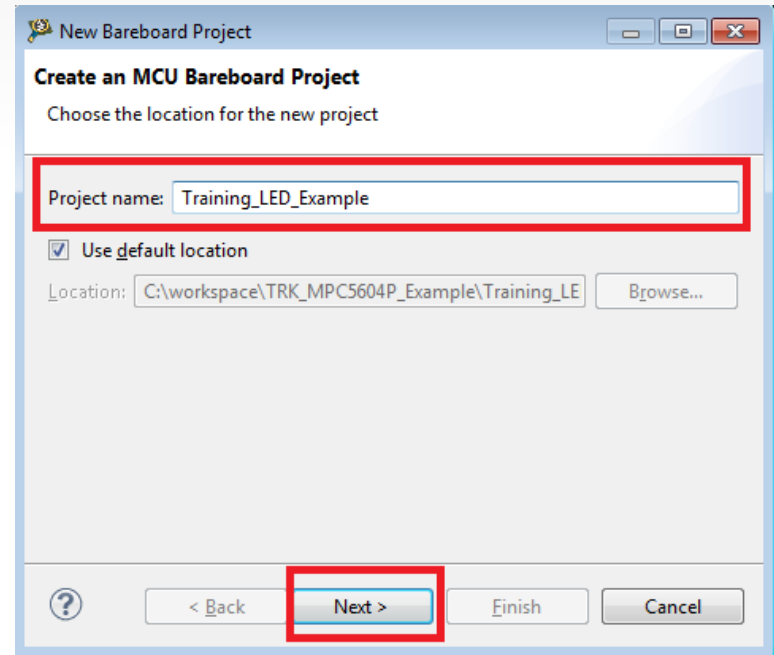
To create a new Bareboard project, Select “New MCU project” from Commander window.





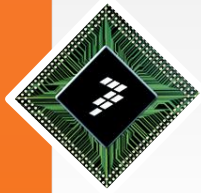
Create CodeWarrior project

Provide a name for the project –
Training_LED_Example
select Next



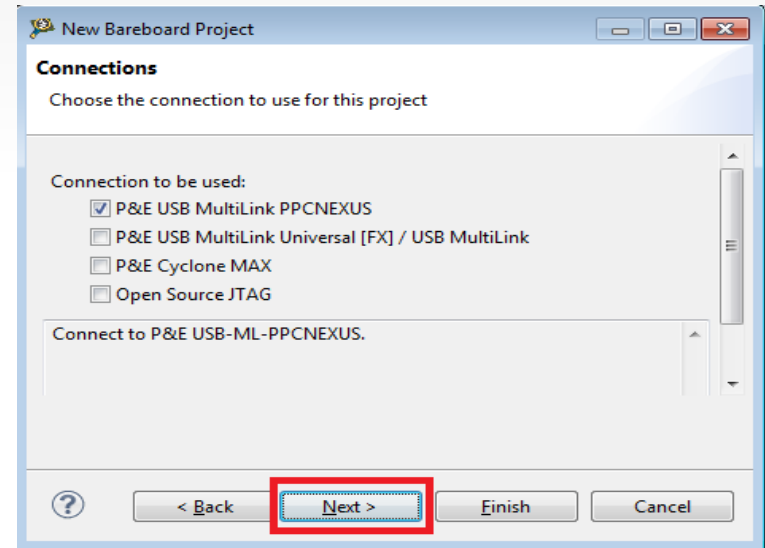
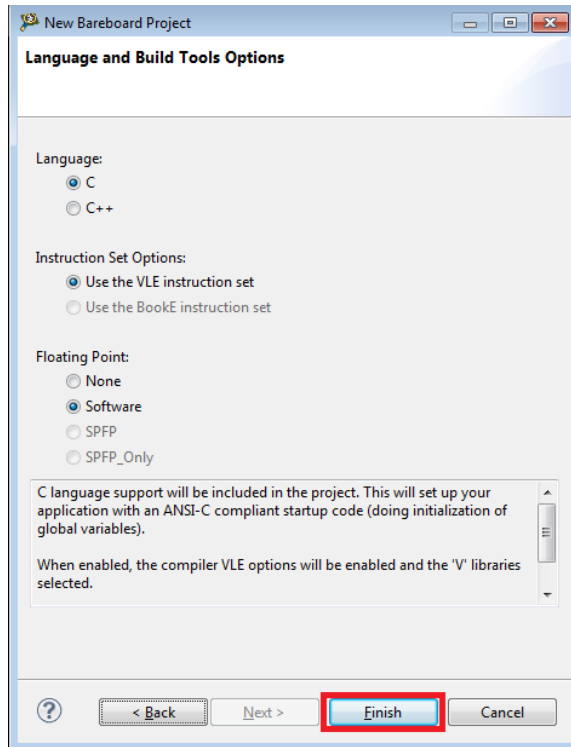
Select Qorivva->MPC560xP->MPC5604P device from the list

Select Next

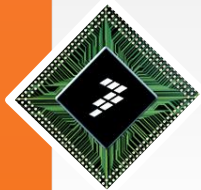


Create CodeWarrior project

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

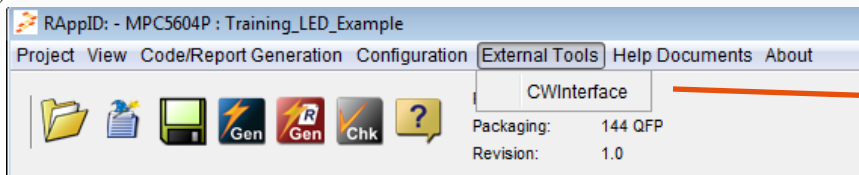


Create CodeWarrior project

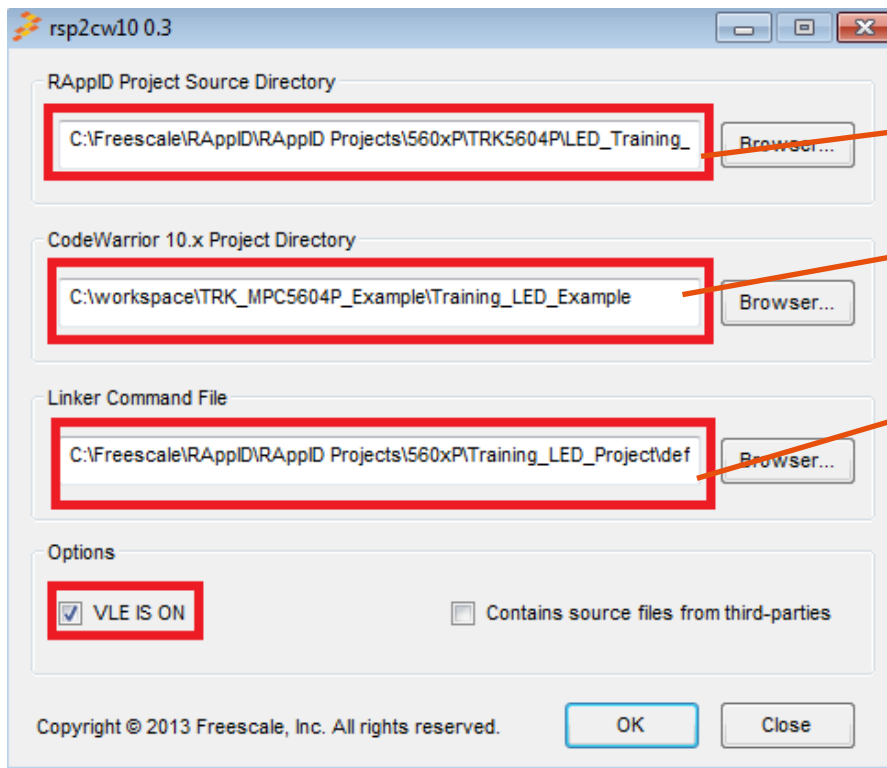
- When Finish is selected, CodeWarrior creates a new bareboard project with CodeWarrior generated code and linker file included in the project.
- Since we have generated code using RAppID for Training_LED_Example project, we want to remove all the CodeWarrior generated code and add RAppID generated code 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



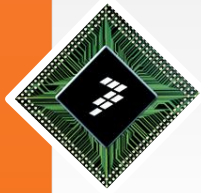
Provide directory path where RAppID code is generated

Provide directory path where CodeWarrior project is created

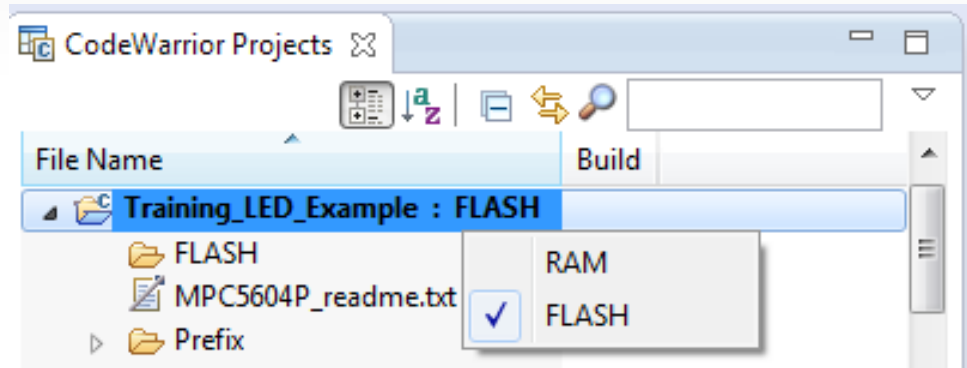
Provide the path of linker file created by RAppID tool

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

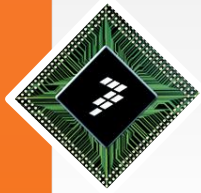


Create CodeWarrior project



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.

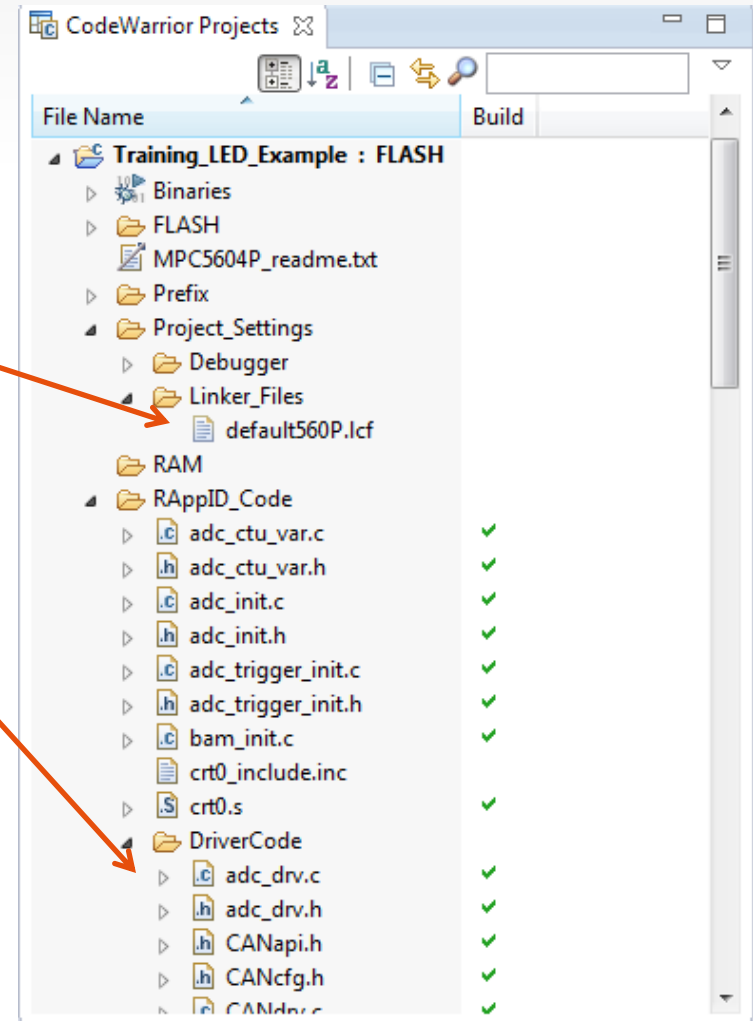


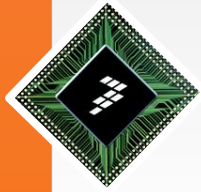
Create CodeWarrior project

Now the project should contain:

RAppID generated linker file

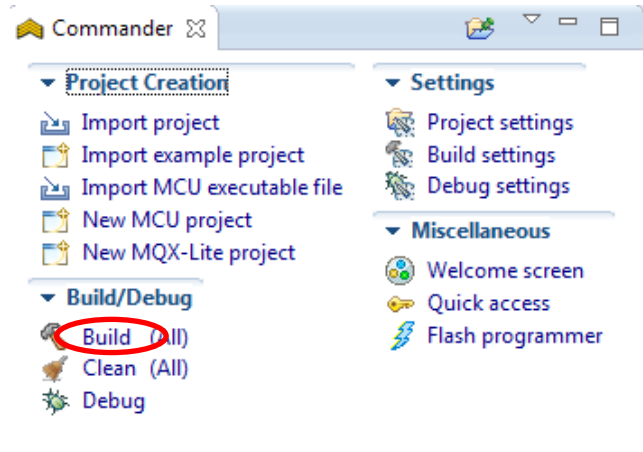
RAppID generated source files and driver code copied from the installation disk

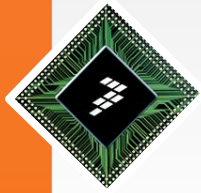




Build project

To build Training_LED_Example project , select Build from Commander window



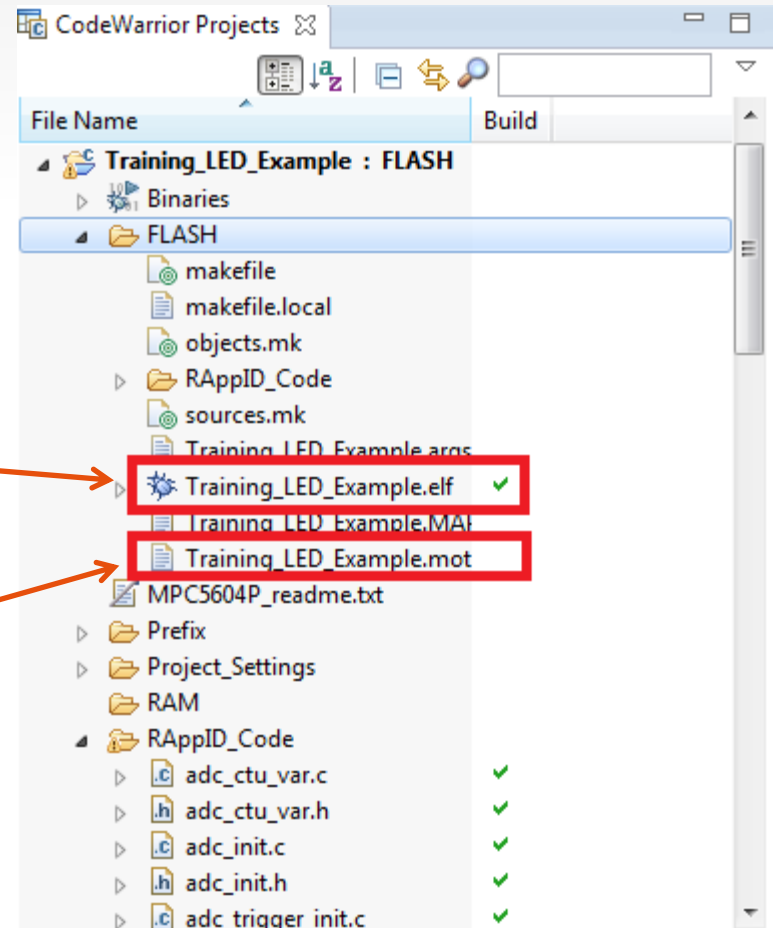


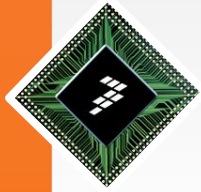
Build project

The build command will produce 2 executable files:

Training_LED_Example.elf
(executable with debug symbols)

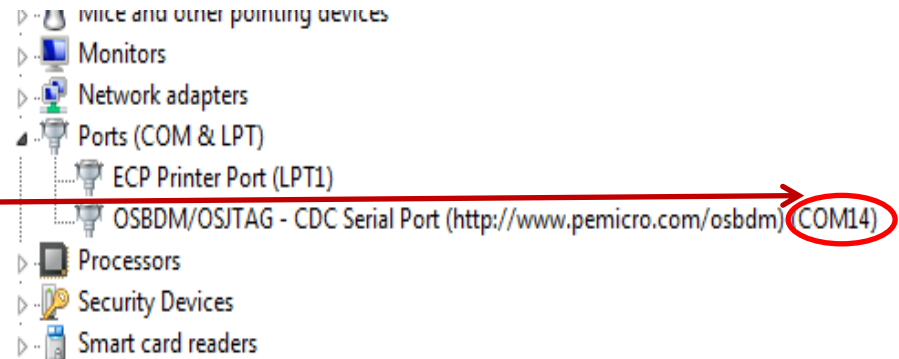
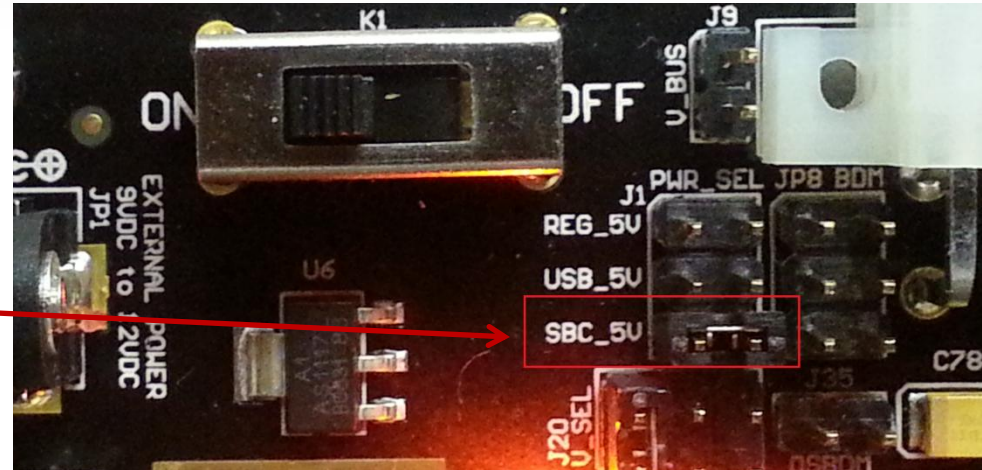
Training_LED_Example.mot
(S-record)

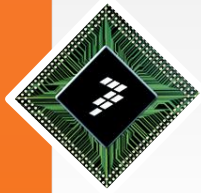




Jumper settings and Power connections

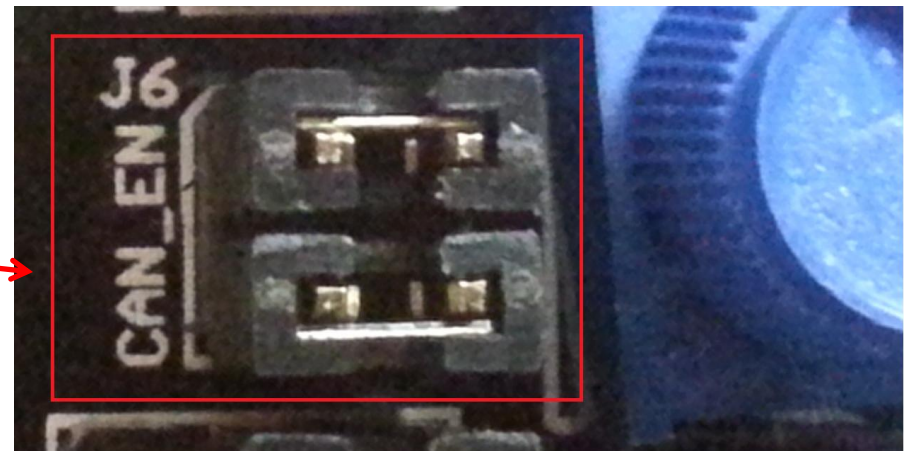
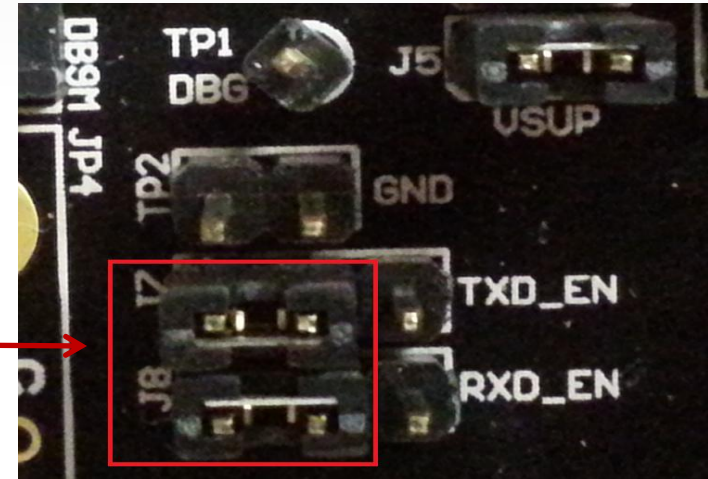
- Since we are using CAN in this example, we need to enable CAN.
- To enable CAN, the board needs to power SBC using external 12V supply.
- Connect jumpers across SBC_5V of J1
- Connect External power to JP1
- Connect your computer to JP2 via USB cable. This connection provides virtual serial port. Confirm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

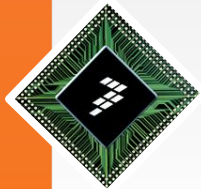




Jumper settings and Power connections

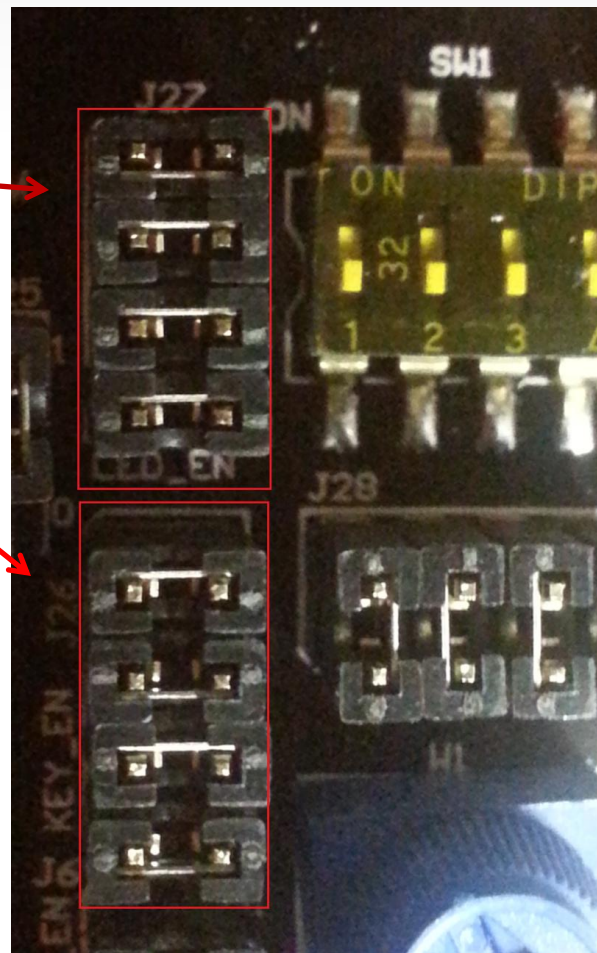
- In this example, we will use Virtual serial port for UART communication. To enable this, connect jumper J7 (TXD_EN) and J8 (RXD_EN) to position 1-2
- To enable CAN communication, connect J6 as shown.

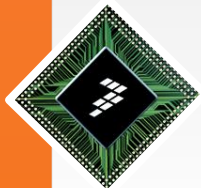




Jumper settings and Power connections

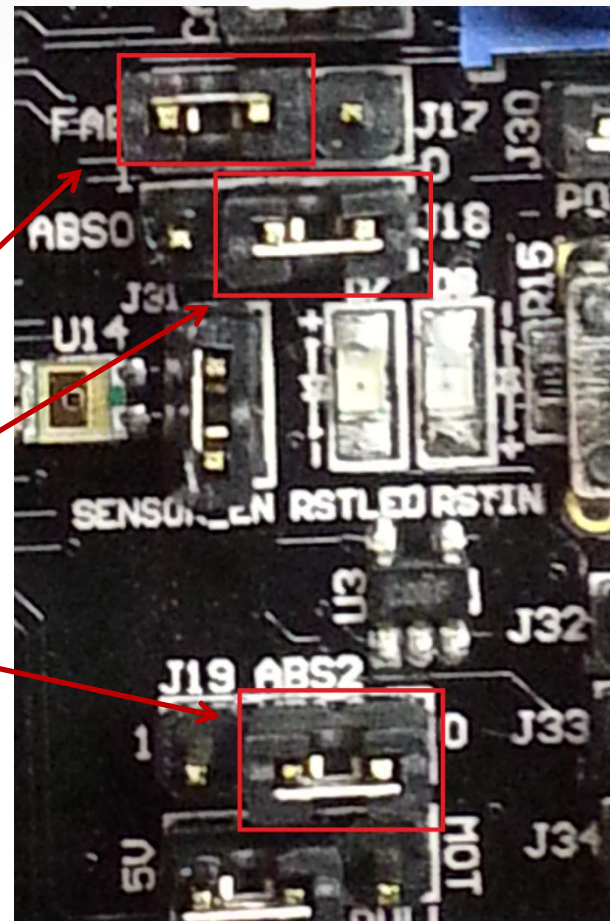
- To enable LEDs, connect all 4 jumpers in J27
- To enable input switches, connect all 4 jumpers in J26

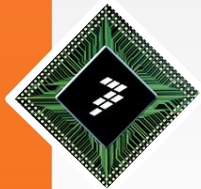




Flash code to target

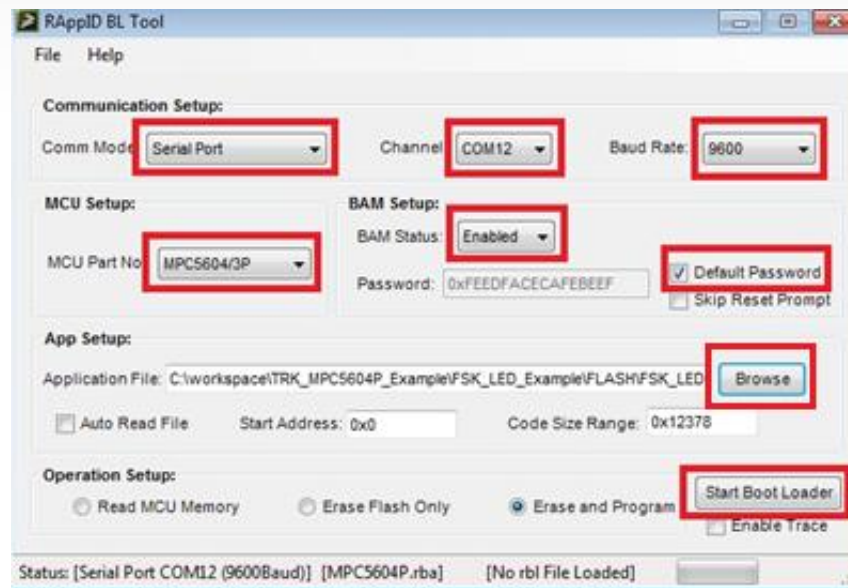
- Before Flashing the code, make sure the TRK-MPC5604P board is connected to external power and to your computer via USB cable
- Set the jumper of J17 to position 1-2 which pulls FAB high and jumper J18 and J19 to position 2-3 to set ABS0 and ABS2 low
- We will use RAppID Bootloader utility to flash S-record file Training_LED_Example.mot using serial port



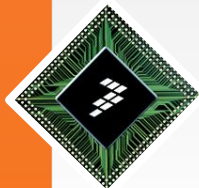


Flash code to target

- 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

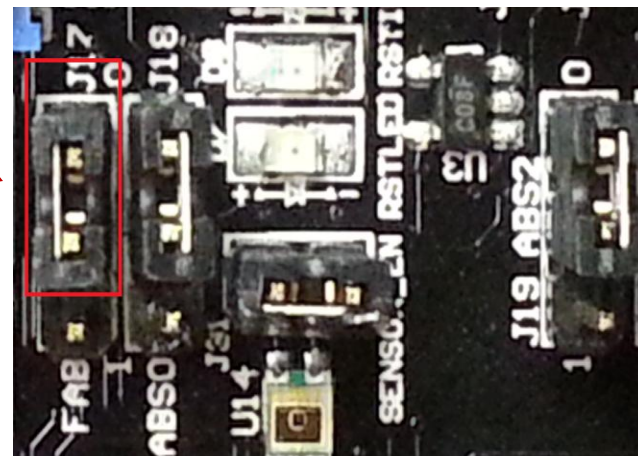
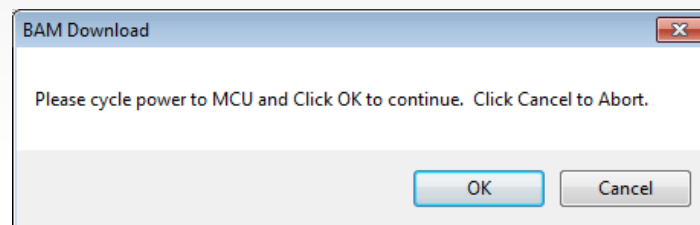


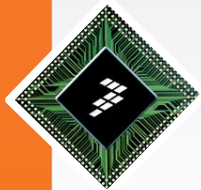
NOTE: If boot loader displays error about wrong password, try 0xFFFFFFFFFFFFFFFF instead of default password option. If your board has microcontroller with previous mask set, the default password will not work.



Flash code to target

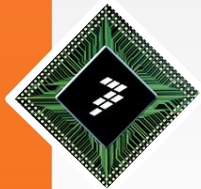
- When asked to cycle power to MCU, press the reset button on the board. Flashing process should start.
- After Flash is complete, Move the jumper J17 to position 2-3 to pull FAB low.
- Turn the power off to the module and re-apply the power
- The code should be running now.





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).



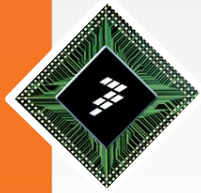
CAN command to turn On/Off LED3

The screenshot shows the IXXAT MiniMon V3 interface. The main window displays a list of CAN messages with columns for Time / 10 mSec, Identifier, Format, Flags, and Data. The data column shows hexadecimal values for each message. Below the main window, there is a status panel with several indicators: Controller initialized, Low speed transceiver, Transmit pending, Data overrun, Error warning level, and Bus off. At the bottom, a message box indicates 'Result of transmission: The operation completed successfully. (0x0)' and 'Msg: 6'.

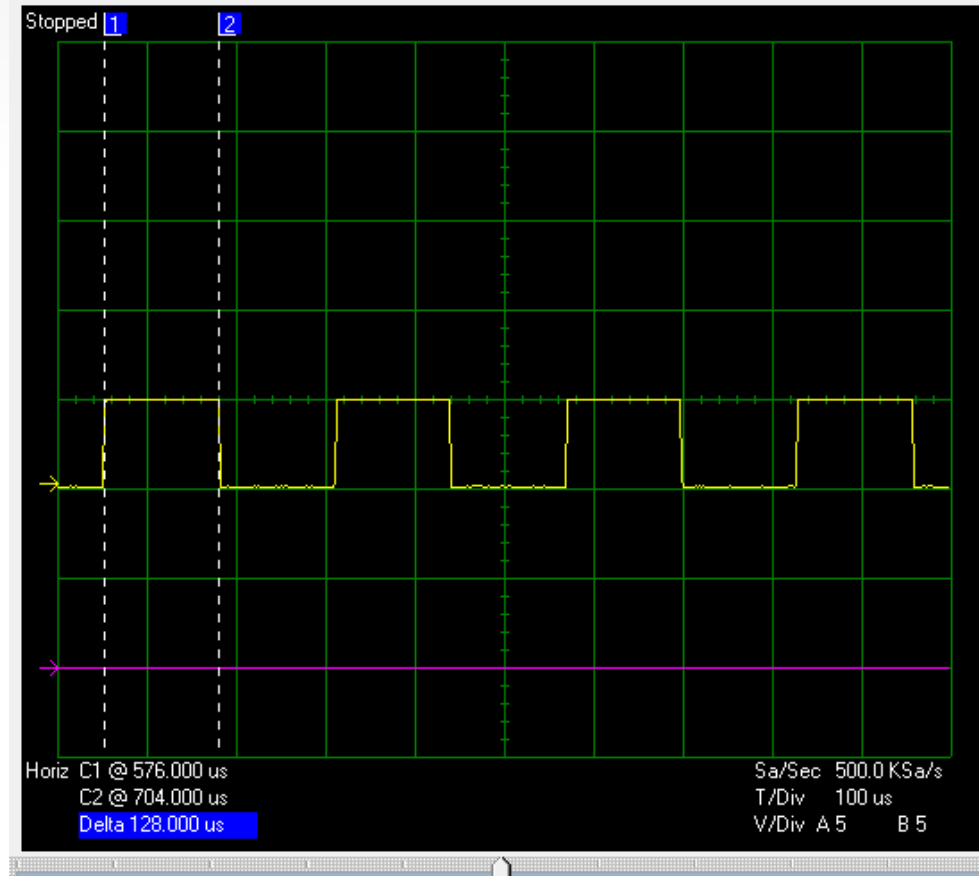
Time / 10 mSec	Identifier	Format	Flags	Data
00:00:06.77		1 Std	Self	01 00 00 00 00 00 00 00
00:00:06.77		2 Std		01 01 00 00 00 00 00 00
00:00:10.01		1 Std	Self	00 00 00 00 00 00 00 00
00:00:10.01		2 Std		01 01 00 00 00 00 00 00
00:00:11.88		1 Std	Self	12 00 00 00 00 00 00 00
00:00:11.88		2 Std		01 FF 00 00 00 00 00 00

Tx	Identifier	Ext.	Rtr	Data
	1	<input type="checkbox"/>	<input type="checkbox"/>	12 00 00 00 00 00 00 00
	1	<input type="checkbox"/>	<input type="checkbox"/>	00 00 00 00 00 00 00 00
	1	<input type="checkbox"/>	<input type="checkbox"/>	01 00 00 00 00 00 00 00

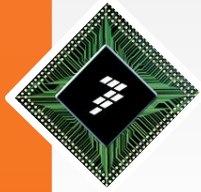
Using a CAN communication tool like CANalyzer or IXXAT MiniMon,
Send CAN command ID = 1 and first data byte = 1 to turn on LED3
Send CAN command ID = 1 and first data byte = 0 to turn off LED3



PWM output signal from PD10



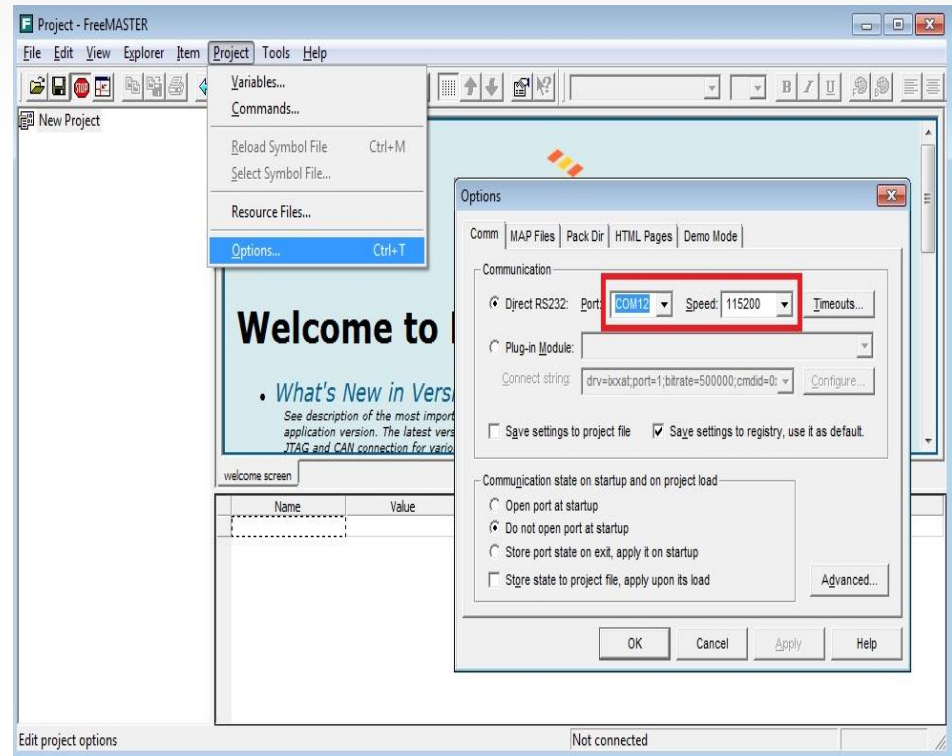
Connect PD10 to a scope and check the PWM output signal

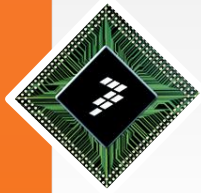


Monitor variables using FreeMASTER

Launch FreeMASTER utility from Windows Start menu.

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

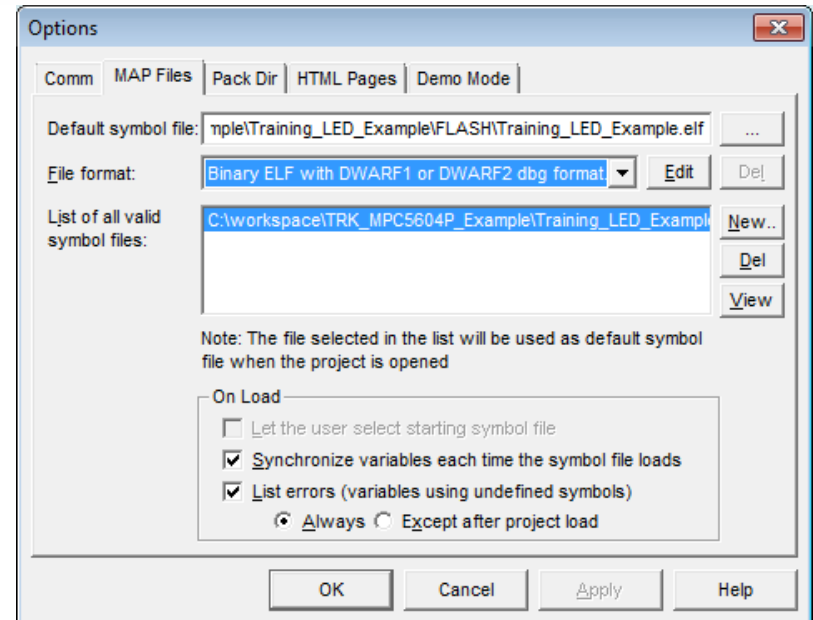


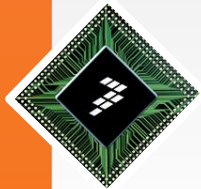


Monitor variables using FreeMASTER

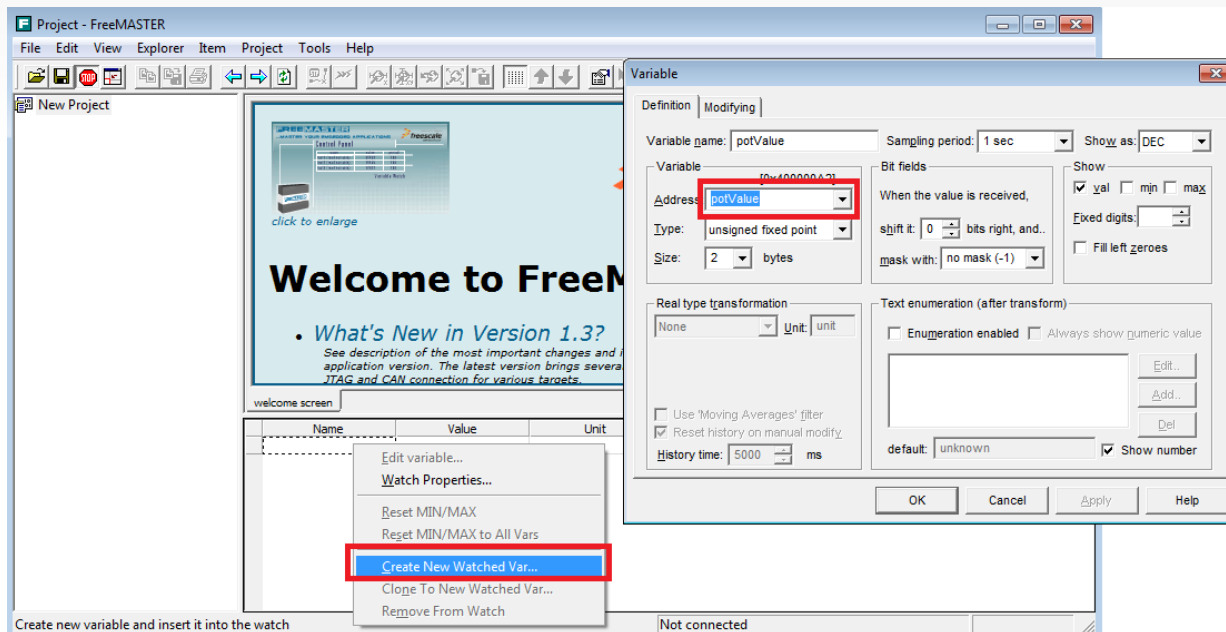
Select the example application MAP file. In this example, the MAP file is the `.elf` file generated during the build process. FreeMASTER uses the information about the variables, their names, types, and addresses contained in the `.elf` file.

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



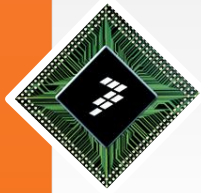


Monitor variables using FreeMASTER

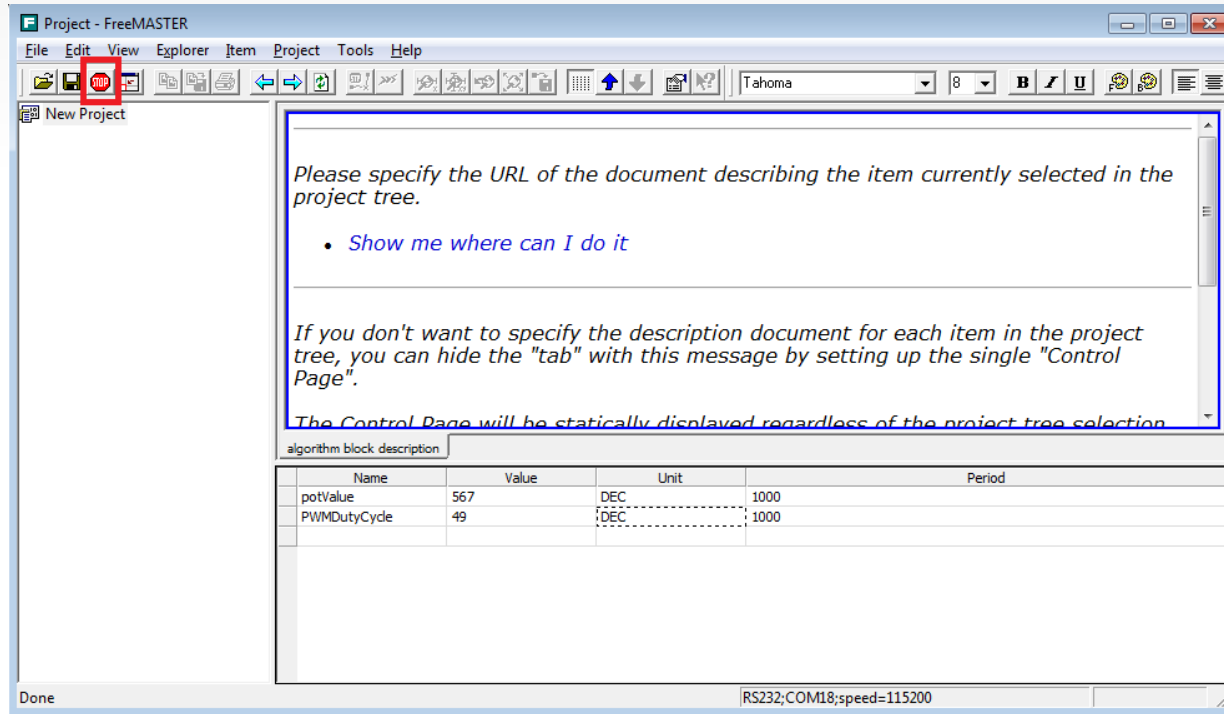


Add the 2 global variables in this example project to monitor in the watch window – potValue and PWMDutyCycle as follows:

- 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.

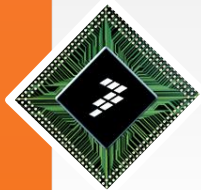


Monitor variables using FreeMASTER



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

