

ThreePhaseMotorControl and TPP_MC33937 Processor Expert Components

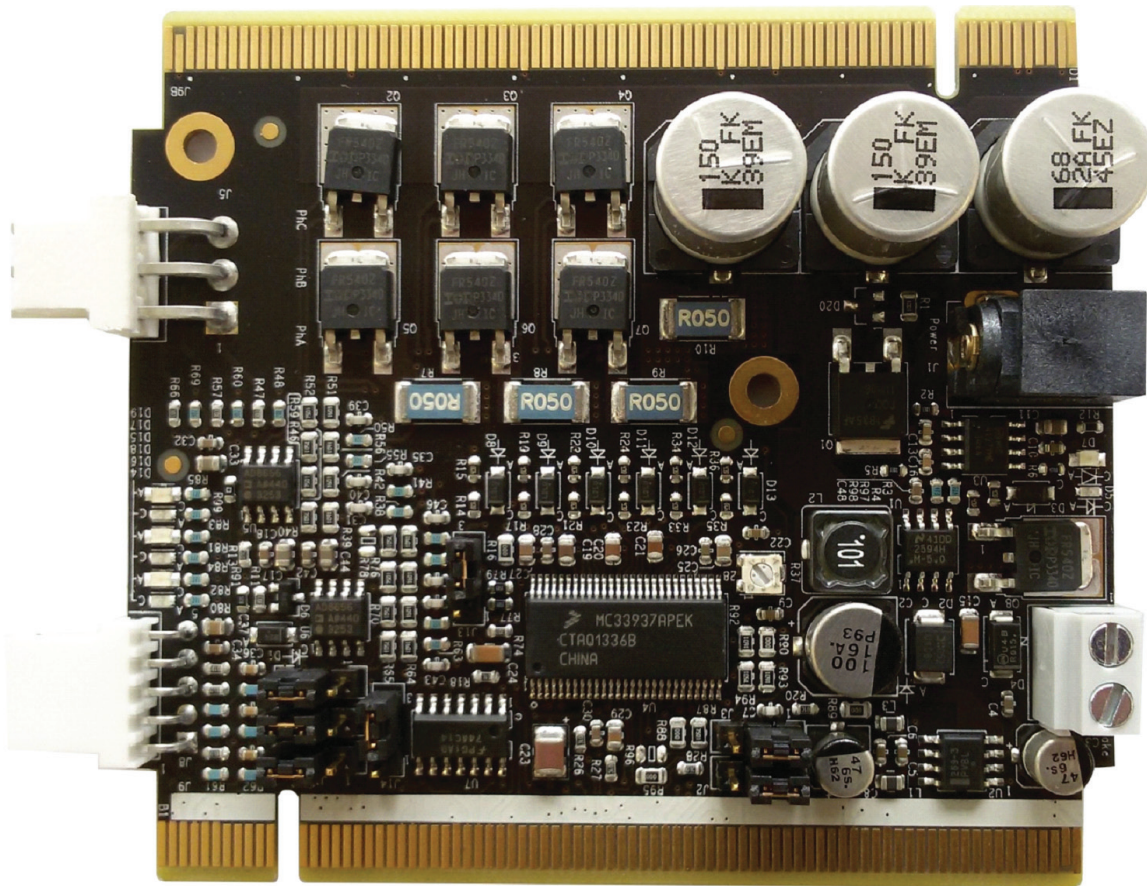


Figure 1. TWR-MC-LV3PH



Contents

1 Important Notice	3
2 Introduction	4
3 Overview	5
4 ThreePhaseMotorControl and TPP_MC33937 Components	6
5 Installing the Software and Setting up the Hardware	20
6 References	34
7 Revision History	35

1 Important Notice

Freescale provides the enclosed product(s) under the following conditions:

This evaluation kit is intended for use of ENGINEERING DEVELOPMENT OR EVALUATION PURPOSES ONLY. It is provided as a sample IC pre-soldered to a printed circuit board to make it easier to access inputs, outputs, and supply terminals. This evaluation board may be used with any development system or other source of I/O signals by simply connecting it to the host MCU or computer board via off-the-shelf cables. This evaluation board is not a Reference Design and is not intended to represent a final design recommendation for any particular application. Final device in an application will be heavily dependent on proper printed circuit board layout and heat sinking design as well as attention to supply filtering, transient suppression, and I/O signal quality.

The goods provided may not be complete in terms of required design, marketing, and or manufacturing related protective considerations, including product safety measures typically found in the end product incorporating the goods. Due to the open construction of the product, it is the user's responsibility to take any and all appropriate precautions with regard to electrostatic discharge. In order to minimize risks associated with the customers applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards. For any safety concerns, contact Freescale sales and technical support services.

Should this evaluation kit not meet the specifications indicated in the kit, it may be returned within 30 days from the date of delivery and will be replaced by a new kit.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typical", must be validated for each customer application by customer's technical experts.

Freescale does not convey any license under its patent rights nor the rights of others. Freescale products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale product could create a situation where personal injury or death may occur.

Should the Buyer purchase or use Freescale products for any such unintended or unauthorized application, the Buyer shall indemnify and hold Freescale and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges Freescale was negligent regarding the design or manufacture of the part. Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. © Freescale Semiconductor, Inc. 2015

2 Introduction

The ThreePhaseMotorControl and TPP_MC33937 Processor Expert Component User Guide describes how to use the ThreePhaseMotorControl and TPP_MC33937 components in Freescale's Processor Expert environment. The document is a stand-alone supplement supporting other MC33937 user guides. Refer to the specific product's documentation for additional information.

This document assumes the user is already a proficient user of CodeWarrior and Processor Expert. Therefore the user guide covers only information directly related to the MC33937 components. For information related to the general usage of Processor Expert, see the CodeWarrior Development Studio Processor Expert User Guide here:

www.freescale.com/files/soft_dev_tools/doc/user_guide/CWPEXUG.pdf.

3 Overview

The ThreePhaseMotorControl and the TPP_MC33937 Processor Expert components are designed to enable rapid application development for the Three Phase FET Pre-driver MC33937A and its Tower Module TWR-MC-LV3PH in Processor Expert software.

The Freescale MC33937A is a field effect transistor (FET) pre-driver designed for three-phase motor control and similar applications. The integrated circuit (IC) uses SMARTMOS technology. It contains three high-side FET pre-drivers and three low-side FET pre-drivers. The IC interfaces to an MCU via six direct input control signals, an SPI port for device setup and fault detection, an asynchronous reset, an interrupt, and two enable signals.

The 3-phase Low Voltage Motor Control board (TWR-MC-LV3PH) is a peripheral Tower System Module. With one of the available MCU tower modules accommodating a selected microcontroller, the board provides a ready-made, software-development platform for one-third horsepower off-line motors. Feedback signals allow a variety of algorithms to control 3-phase PMSM and BLDC motors.

The TWR-MC-LV3PH module features:

- Power supply voltage input 12-24 VDC, extended up to 50 V
- Output current up to 8.0 A
- Power supply reverse polarity protection circuitry
- 3-phase bridge inverter (6-MOSFET's)
- 3-phase MOSFET gate driver with overcurrent and undervoltage protection
- 3-phase and DC bus-current-sensing shunts
- DC bus-voltage sensing
- 3-phase back-EMF voltage sensing circuitry
- Low-voltage on-board power supplies
- Encoder/Hall sensor sensing circuitry
- Motor power and signal connectors
- User LED, power-on LED, and six PWM LED diodes

For more information about TWR-MC-LV3PH, refer to TWR-MC-LV3PH tool summary page.

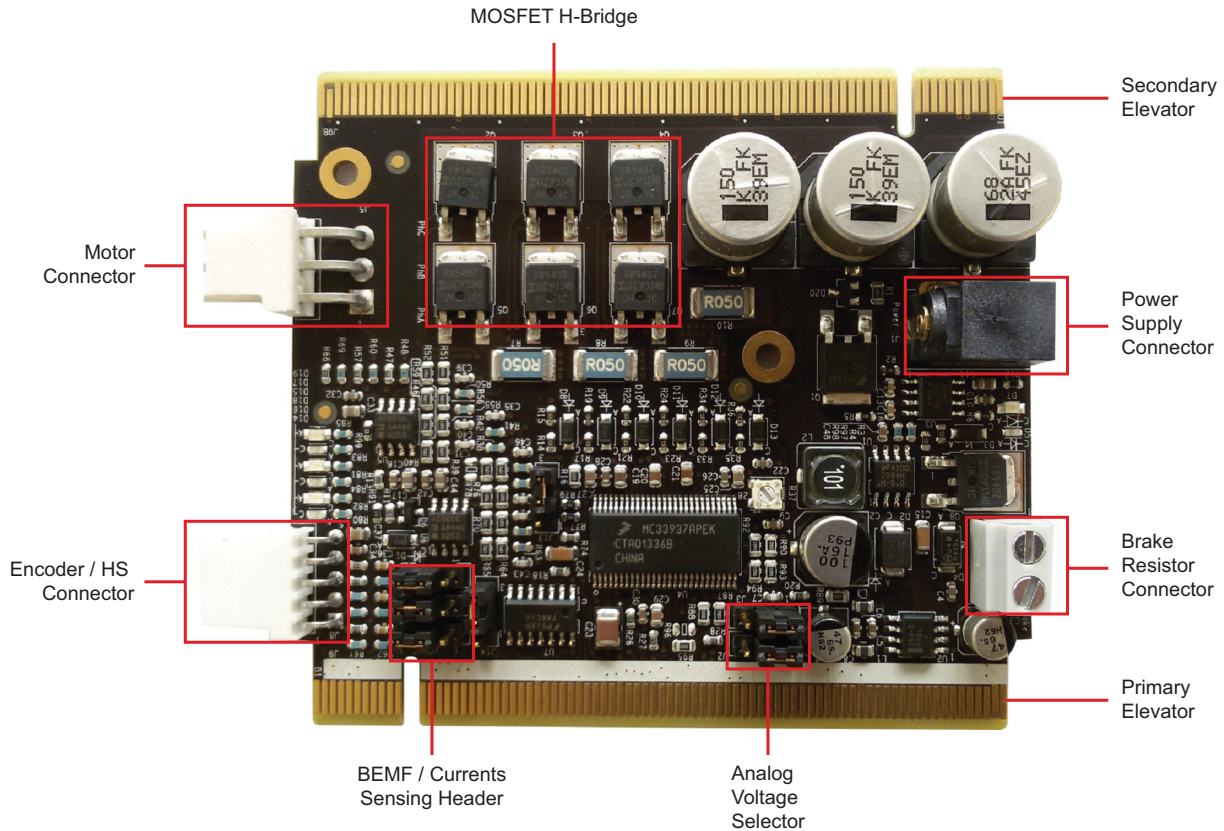


Figure 2. Board Description

4 ThreePhaseMotorControl and TPP_MC33937 Components

This chapter explains the functionality and use of the ThreePhaseMotorControl and TPP_MC33937 Processor Expert components. The TPP_MC33937 component supports Freescale MC33937 SPI configuration and initialization functions. The ThreePhaseMotorControl component extends this functionality to simple BLDC motor control logic.

Note

Import the SPI_Device component (included in this package) into Processor Expert prior to using the TPP_MC33937 component. Furthermore, have the TPP_MC33937 component imported to use the ThreePhaseMotorControl component.

4.1 TPP_MC33937 Processor Expert Component

The TPP_MC33937 Processor Expert component encapsulates the basic functionality of Freescale's MC33937 IC. This component does not, by itself, generate motor control signals. Its primary function is to enable the configuration and initialization of the MC33937 using the chip's SPI communication lines. To extend Processor Expert capabilities beyond the TPP_MC33937 component's basic functions, import and use the ThreePhaseMotorControl component. You can also develop a customized solution.

This user guide summarizes the main features of the TPP_MC33937 component. While in Processor Expert, access additional information about a specific item by mousing over the item. As the cursor hovers over the item, a pop-up displays a brief description of the item's function. In addition, each component has a Help feature accessible by right-clicking on the component in the component tree (see [Figure 3](#)).

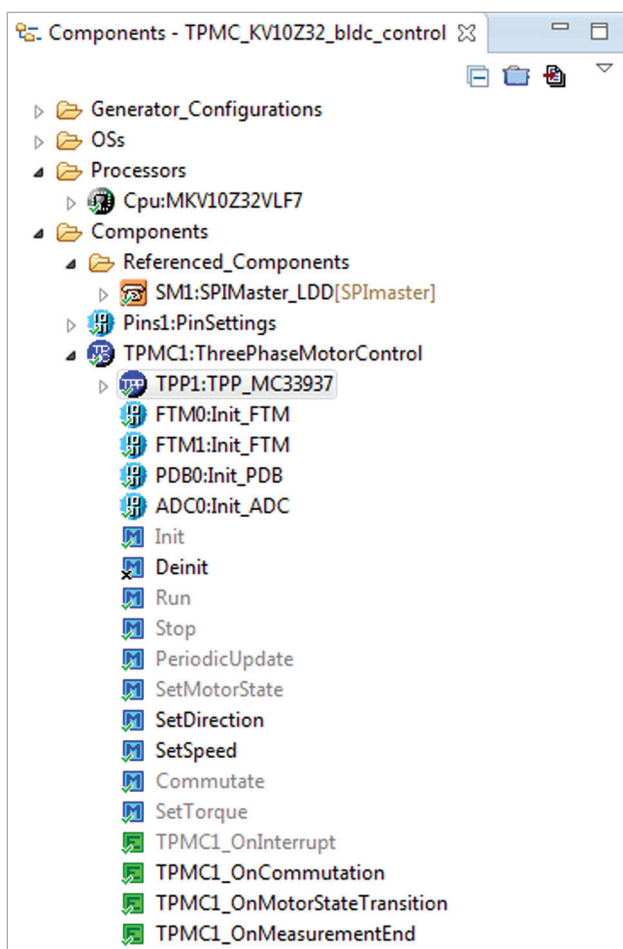


Figure 3. Component Tree

4.1.1 General Settings

When selecting the TPP component in the component tree, Processor Expert displays the following settings in the Component Inspector screen (see [Figure 4](#)).

Name	Value	Details
Component Name	TPP1	
Tower Board (TWR-MC-LV3P)	no	
SPI Component	SPI_Device	
Device Mode		
Lock	Disabled	
Full On	Disabled	
Desaturation	Disabled	P If enabled, there are not possible further changes to device settings. Description for the current value (Disabled) - Disables mode lock.
Deadtime	14000	D
Enable Pins		
Enable 1 Pin	Enabled	
Component link	BitIO_LDD	
Pin for EN1	PTD7/UART0_TX/FTM0_CH1/FTM1_CH1...	
Enable 2 Pin	Disabled	
Reset Pin	Enabled	
Component link	BitIO_LDD	
Pin for RST	ADC0_SE11/CMP1_IN0/PTC2/SPI0_PCS2/...	
Interrupt Pin	Enabled	
Component link	ExtInt_LDD	
Pin for INT	ADC1_SE2/ADC1_DP2/PTB3/I2C0_SDA/U...	
Device interrupts		
Over-Temperature	Disabled	
Desaturation	Disabled	
Low-VLS	Disabled	
Over-Current	Disabled	
Phase Error	Disabled	
Framing Error	Disabled	
Write Error	Enabled	
Reset Event	Enabled	
Feedback Pins	Disabled	
PhaseA		
Component link	ExtInt_LDD	
External Interrupt Pin	ADC0_SE0/ADC0_DP0/PTE20/FTM1_CH0...	Property is disabled
PhaseB		
Component link	ExtInt_LDD	
External Interrupt Pin	ADC0_SE0/ADC0_DP0/PTE20/FTM1_CH0...	Property is disabled
PhaseC		
Component link	ExtInt_LDD	
External Interrupt Pin	ADC0_SE1/ADC0_DP1/ADC1_SE0/PTE16/...	Property is disabled
Initialization		
Auto initialization	no	
Operational mode after init.	Standby	

Figure 4. TPP_MC33937 Properties

- The Tower Board option indicates the use of the component in a Tower System configuration or with a standalone MC33937. If this option is set to yes, both the Enable 2 Pin and the Feedback Pins properties are inaccessible, because in the TWR-MC-LV3PH, these pins are not routed to Tower elevator modules. If the user has a customized board and needs these pins to be accessible in the component, wire them manually to the Tower MCU module. Similar functionality is provided by the ThreePhaseMotorControl component, which provides zero-crossing determination based on Back EMF measurement.
- The SPI Component option shares a SPI bus with other components using SPI_Device.
- The Device Mode option sets the following features:
 - Lock prevents any additional changes after initialization.
 - Full On controls output stages directly. No deadtime is added into the output signal (see the MC33937 datasheet).
 - Desaturation forces the IC to enter fault-protection mode when desaturation occurs.

- The Deadtime option sets a mandatory delay when toggling from high-side to low-side outputs. This is helpful when shorting is an issue. This option is not available when Full On mode is activated.
- The Enable Pins manages the device's operational mode.
- The Reset Pin is also used for management of the device's operational mode.
- The Interrupt Pin provides a means of determining when an internal device event occurs. Such an event causes an interrupt only when the corresponding option is enabled for that event. For more information on interrupt handling, see [Section 4.1.5](#).
- The Feedback Pins indicate when a zero-crossing event occurs. This event occurs only once in a phase period. Simultaneous occurrences are not possible. A zero-crossing event can be used to determine the commutation moment in motor control. This transition is done internally in the device and is reflected as a transition from LOW to HIGH state (or HIGH to LOW) on the Feedback pins.
- The Initialization option has two parameters:
 - Auto initialization eliminates Init method implementations within the code. With this parameter set, the Init method is automatically called during all CPU peripheral initialization.
 - Operational Mode after Initialization selects whether the device is active or in standby mode after initialization.

4.1.2 SPI Configuration

TPP_MC33937 component uses the SPI_Device component to control SPI communication. SPI_Device serves as an arbiter when simultaneous requests occur on the SPI bus. The SPI_Device component uses the shared SPI_Master_LDD component to support these SPI functions. SPI_Master_LDD is located under Referenced Components in the component tree (see [Figure 5](#)).

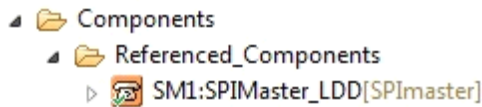


Figure 5. SPI Configuration (SPI_Master_LDD location)

You must select the SPI_Master_LDD component pins (Input, Output, Clock) and timing settings according to the recommendations in the MC33937 datasheet. Typical settings are shown in [Figure 6](#).

Name	Value
Component name	SPI1
Device	SPI0
▶ Interrupt service/event	Enabled
▲ Settings	
▲ Input pin	Enabled
Pin	PTD3/SPI0_SIN/UART0_TX/FTM0_CH3/I2...
▲ Output pin	Enabled
Pin	PTD2/LLWU_P13/SPI0_SOUT/UART0_RX/...
▲ Clock pin	
Pin	PTC5/LLWU_P9/SPI0_SCK/LPTMR0_ALT2...
Chip select list	0
▶ CS external demultiplexer	Disabled
▶ Attribute set list	1
Clock rate	2.097152 MHz
Delay between chars	0.190735 μs
CS to CLK delay	0.190735 μs
CLK to CS delay	0.190735 μs
HW input buffer size	1
HW input watermark	1
HW output buffer size	1
HW output watermark	1
▲ Initialization	
Initial chip select	0 D
Initial attribute set	0 D
Enabled in init. code	yes
Auto initialization	yes
▶ Event mask	
▶ CPU clock/configuration selection	

Figure 6. SPI Configuration (SPI_Master)

Finally, set the chip select pin, which is found directly under SPI_Device in the component tree (see [Figure 7](#)). The component's current implementation requires initializing the CS pin to 1, as specified in the datasheet.

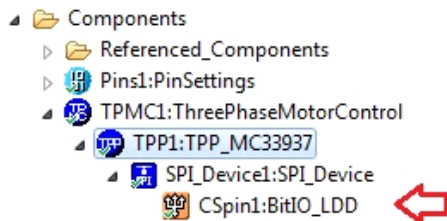


Figure 7. SPI Configuration (Chip Select Pin)

4.1.3 Component API

TPP_MC33937 component offers an Application Programming Interface (API) which can be embedded in the code to enable real-time configuration of the device. The methods and events applying to TPP_MC33937 are listed in the component tree (see [Figure 8](#)). The component's Help documentation contains a Typical Usage section that describes how to best implement this API.

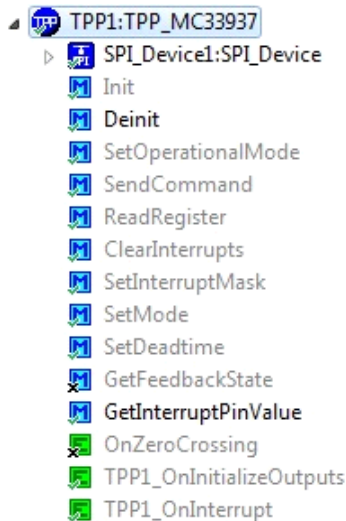


Figure 8. Pre-driver Methods and Events

Notice that some of the TPP_MC33937 methods and events are marked with a tick (for example, Init) and others with a cross (for example GetFeedbackState). The tick indicates items that are generated. The cross indicates items not generated. These settings can be changed in the Component Inspector tab (see Figure 9). Note that some of the items are always generated, because they are needed for proper functionality. This forced behavior depends on various combinations of settings of the component properties.

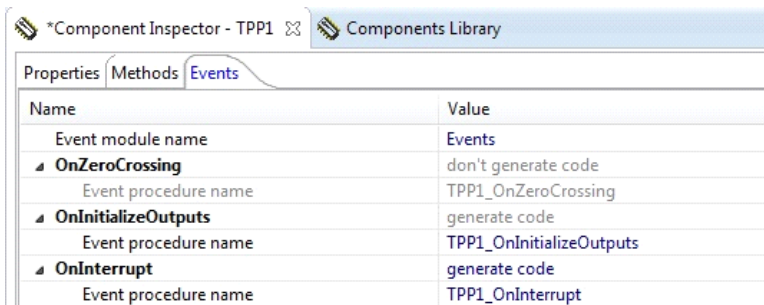


Figure 9. Generated Methods and Events

For summarization of available API methods and events and their descriptions, see Table 1.

Table 1. TPP Component API Methods and Events

Method / Event	Description
Init	Initializes the device. Configures device mode and interrupts. Prepares low-side and high-side output stages.
Deinit	Deinitializes the device by putting it into sleep mode which causes the internal configuration to be erased
SetOperationalMode	Sets device operational mode. Internal method logic is responsible for mode transition consistency
SendCommand	Sends (via SPI) commands to the device and receives the content of device status register 0
ReadRegister	Reads selected device status register [0 - 3]
ClearInterrupts	Clears device internal events flags. Clears only those flags actually set in device status register.
SetInterruptMask	Sets device interrupts mask. If the selected interrupt is masked, check the occurrence of the event by reading device status register manually.
SetMode	Sets device mode. Note that after a lock, the device mode settings cannot be changed.
SetDeadtime	Sets dead time value. If value is set to 0, you must handle the delay between low-side and high-side output stages by toggling the PWM phases signals. To prevent shorts, a minor real-time delay occurs in the output stages unless Full On mode is enabled. Note that this method disables maskable interrupts for the time interval needed to prevent deadtime calibration interference.
GetFeedbackState	Gets feedback pins state. This method is used for zero-crossing detection. Note that only one value from all three pins should be 0.
GetInterruptPinValue	Gets interrupt pin value. Note that this pin holds its level at high until all interrupt flags are cleared.

Table 1. TPP Component API Methods and Events (continued)

Method / Event	Description
OnZeroCrossing	This event handler is invoked when zero-crossing occurs on one of three phases
OnInitializeOutputs	This event handler is invoked during initialization of the device. Initialize outputs according to datasheet recommendations. This part of initialization is usually handled by motor control logic.
OnInterrupt	This event handler is invoked when one or more device internal events occur and corresponding interrupts are enabled. It is up to user to take corrective actions and clear corresponding flags.

A code can be written to directly access and modify the TPPx_TDeviceData structure (see [Figure 10](#)). However, unless there is a compelling reason for doing otherwise, it is recommended to use the Processor Expert interface to control these settings.

```

typedef struct {
    TPP1_TDeviceMode Mode;           /* the device operational mode */
    uint8_t StatusRegister[4];      /* the device status registers */
    uint8_t InterruptMask;          /* the device interrupts mask */
    uint8_t ModeMask;               /* the device general settings mask (lock, full-on, desaturation) */
    uint16_t Deadtime;              /* dead time value */
    TUserDataPtr UserDataPtr;       /* the pointer to user data */
} TPP1_TDeviceData, *TPP1_TDeviceDataPtr; /* Device mode and settings structure (reflects settings stored in pre-driver). */
    
```

Figure 10. TPPx_TDeviceData Structure

4.1.4 Operational Mode Management

There are five operational modes (see Device Operational Flow Diagram in datasheet):

- Init mode is used only at device initialization.
- Active mode is used when the device is properly configured and output stages are prepared.
- Fault protection mode is not used directly by internal component logic. However, this mode assists in handling interrupts.
- Sleep mode erases the device's internal configuration and turns off its output stages.
- Standby mode turns off the output stages, but the device's internal configuration remains intact.

For more information on dynamic operational mode management, refer to Typical Usage section in Component Help.

4.1.5 Interrupt Handling

There are two ways to handle device interrupts:

- Use the OnInterrupt event. If you choose this method, do not use SPI communication inside the event handler. Doing so could cause a deadlock condition or an unsuccessful data transfer. A better method is to have the event handler set a flag and allow the application code to process the interrupt.
- Read the value of the Interrupt pin with the GetInterruptPinValue method.

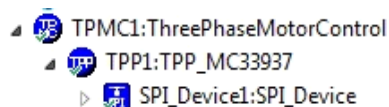
It is possible to omit interrupt handling completely and periodically read device status register instead. However, this approach reduces the effectiveness of the interrupt handling process and in some cases may result in a lag between the time the event occurs and the time that the status register is read.

In all cases the application code is responsible for taking corrective actions and clearing interrupt flags. When multiple pending interrupts occur at the same time, the OnInterrupt event is invoked only once. The application code must correctly recognize and handle the multiple pending interrupts. Also, the Interrupt pin value toggles only after all of the faults are cleared.

Interrupt handlers should be as short as possible. Time-consuming handlers could disrupt the time-sensitive motor control logic. Refer to the Typical Usage section in Component Help for an example of how to effectively implement device interrupt processing.

4.2 ThreePhaseMotorControl Processor Expert Component

The ThreePhaseMotorControl component extends the functionality of TPP_MC33937 component with simple BLDC motor control logic. To use the ThreePhaseMotorControl component in the project ([Figure 11](#)), all three components from this package must be imported into the development studio (CodeWarrior, Kinetis Design Studio, etc.).


Figure 11. Components in the Component Tree

This component encapsulates only basic BLDC motor control logic. It does not provide a complete solution for all BLDC motor applications. For additional information, refer to the Component Help documentation.

4.2.1 General Settings

When adding and selecting ThreePhaseMotorControl components in the project, view the following properties in Processor Expert Inspector (see [Figure 12](#)).

Name	Value	Details
Component Name	TPMC1	
▲ Pre-Driver Settings		
Pre-Driver Link	TPP_MC33937	Pre-Driver settings are directly in inherited component TPP1
▲ PWM Settings		
PWM Timer Link	Init_FTM_VAR0	
PWM Timer Device	FTM0	FTM0
PWM Frequency	20 kHz	
Measurement Trigger	Enabled	
▲ Pins		
AT	ADC1_SE3/PTC1...	
AB	PTE25/FTM0_C...	
BT	CMP1_IN1/PTC...	
BB	PTC4/LLWU_P8...	
CT	PTD4/LLWU_P1...	
CB	ADC0_SE3/PTD5...	
▲ Deadtime Insertion	Timer	
Deadtime	875 <input type="checkbox"/> D	
Motor Control	Disabled	
Auto Initialization	yes	

Figure 12. Properties in the Basic Configuration

In this basic configuration, the timer module generating pre-driver PWM signals ([Figure 13](#)) is preset to support a complementary bipolar switching PWM. In complementary bipolar switching mode, two phases are powered with complementary PWM signals. In one phase, the duty cycle is greater than 50%. In the other phase, the complementary duty cycle value is less than 50%. With a bipolar PWM switching pattern, the PWM signals for the top and bottom switches must be swapped when a commutation event occurs (see [Figure 13](#)).

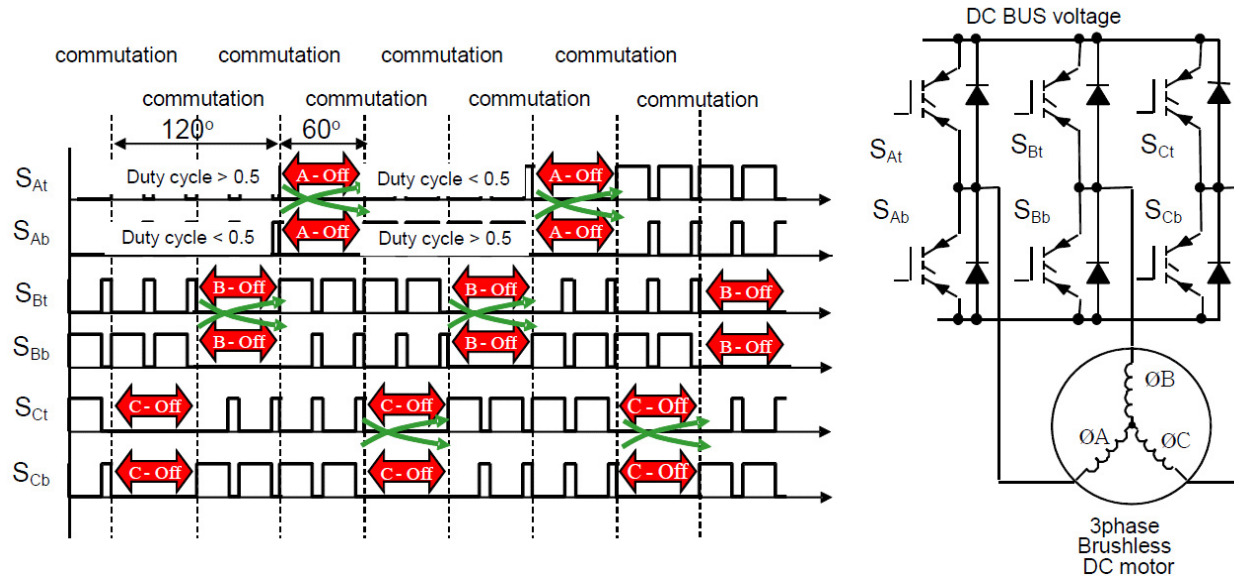


Figure 13. Complementary Bipolar PWM Switching

The following configuration drives a BLDC motor implementing the commutation timing. The only available properties in the component properties tab are the Commutate method, the SetTorque method, and the OnCommutation event.

The options for these properties are:

- The **Pre-Driver Settings** group contains a link to the inherited TPP_MC33937 component. All pre-driver settings must be set within the component itself. The only exception is **Deadtime**, which is handled by the ThreePhaseMotorControl component.
- The **PWM Settings** group contains FlexTimer module inherited settings. Only three of the five options — PWM Frequency, Measurement Trigger, and Pins — can be modified.
 - PWM Frequency selects values between 10 kHz and 20 kHz as the PWM frequency
 - Measurement Trigger sets an A/D converter trigger
 - Pins associates each of the output pins with a corresponding input pin name
- The **Deadtime Insertion** option specifies which component inserts deadtime into the PWM signal. There are two options:
 - Pre-driver specifies the Pre-driver component handling deadtime insertions. The Deadtime value entered under this option is in nanoseconds. When entered, the value is automatically set in the Pre-driver component.
 - Timer specifies the FTM timer component handling deadtime insertions. The Deadtime value entered under this option is in nanoseconds. When entered, the value is automatically set in the Pre-driver component. If the pre-driver value is set to 0, the pre-driver inserts a default deadtime. (To prevent the minimal deadtime insertion, set the Full On mode in the pre-driver component.)
- The Auto Initialization option, when set, causes the **Init** method to be called automatically during CPU peripheral initialization. If this option is not set, the code must call the **Init** method.

When enabling the Motor Control group, a timer component (forced commutation) is added to the project ([Figure 14](#)) and a simple state machine is added into the code. This starts up any BLDC motor at a desired speed (assume that the motor load does not change while the rotor is spinning). Changing the load while the rotor is spinning may cause the motor to stop because no speed regulator is implemented.

Properties		
Name	Value	Details
Component Name	TPMC1	
Pre-Driver Settings		
Pre-Driver Link	TPP_MC33937	Pre-Driver settings are directly in inherited component TPP1
PWM Settings		
PWM Timer Link	Init_FTM_VAR0	
PWM Timer Device	FTM0	FTM0
PWM Frequency	20 kHz	
Measurement Trigger	Enabled	
Pins		
PA_HS	ADC1_SE3/PTC1...	
PA_LS	PTE25/FTM0_C...	
PB_HS	CMP1_IN1/PTC...	
PB_LS	PTC4/LLWU_P8...	
PC_HS	PTD4/LLWU_P1...	
PC_LS	ADC0_SE3/PTD5...	
Deadtime Insertion	Timer	
Deadtime	875	<input type="checkbox"/>
Motor Control	Enabled	
Commutation Timer Link	Init_FTM_VAR0	
Commutation Timer Device	FTM1	FTM1
Direction	Clockwise	
Minimum RPM	60	<input type="checkbox"/>
Initial RPM	5000	<input type="checkbox"/>
Alignment Period	1000	<input type="checkbox"/>
Freewheeling Period	2000	<input type="checkbox"/>
Number of Pole Pairs	4	<input type="checkbox"/>
Acceleration	500	<input type="checkbox"/>
Torque Settings		
Alignment Torque	10	<input type="checkbox"/>
Startup Torque	30	<input type="checkbox"/>
Spin Torque	50	<input type="checkbox"/>
Current and Voltage Measurement	Disabled	
BEMF Integration	Disabled	
Auto Initialization	yes	

Figure 14. Properties with Motor Control Enabled

When the Motor Control group is enabled, the following additional properties show up (see [Figure 14.](#))

- The Commutation Timer Link option specifies the name of the linked commutation timer component.
- The Commutation Timer Device option specifies the name of the linked commutation timer device.
- The Direction option determines the initial direction (clockwise or counterclockwise) in which the motor rotates. If set in a clockwise/counterclockwise direction and the motor spins the other way, then change the order of phases, so phases A, B, and C of the motor correspond to phases A, B, and C of the pre-driver. The direction can be changed later in code. This option assumes the phases A, B, and C of the predriver are connected to phases A, B, and C (U, V, W) of the motor.
- The Minimum RPM option determines minimum speed (in revolutions per minute) which can be set while the motor is running. It is also used during startup as the first set speed before the motor accelerates to its initial speed.
- The Initial RPM option determines the speed (in revolutions per minute) the motor reaches after the Run method for starting the motor is called. The motor speed can be altered later by application code.
- The Alignment Period option specifies the duration of the rotor alignment (in milliseconds) before the motor starts up.
- The Freewheeling Period option specifies the delay between entering the Freewheeling state and the automatic transition to the Ready state. This value is used in the Stop method (and in all other methods —for example, SetDirection — uses the Stop method.)
- The Number of Pole Pairs option determines the number of pole pairs in the rotor's permanent magnet. This value affects the rotor's rotation speed. If the wrong number of poles is specified in this option, the motor rotates at a different speed than the one specified.

- The Acceleration option determines how fast the motor is capable of accelerating (in RPM per second). This value usually depends on the motor load. Changing the load while the motor is spinning may result in a loss of synchronization between the commutation timer and the actual motor movement, in which case the motor stops. To keep the motor spinning while the load changes, modify the application code to include a speed regulator using a measured speed to control motor acceleration. This speed regulator with speed measurement is not part of the component.
- The Torque Settings group specifies the magnitude of torque in percent to be applied in three states: alignment, startup, and spin. Torque influences current consumption which decreases with increasing speed. Avoid setting torque to 100% in any of the three option even though the setting accepts 100% as a value.
 - Alignment Torque specifies the magnitude of torque in percent applied when the motor is aligned.
 - Startup Torque specified the magnitude of torque in percent applied when the motor is starting.
 - Spin Torque specifies the magnitude of torque in percent applied when the motor is spinning at its specified speed. (Either the initial RPM set in the component properties or the most current speed set in code.)

Current and Voltage Measurement / BEMF Integration

- These options add PDB (programmable delay block) and A/D converter into the project (Figure 17). Current and Voltage Measurement enables supply current, supply voltage, and Back-EMF voltage measurement. BEMF Integration enables Back-EMF integration. Current and voltage measurement can be enabled while back EMF integration is disabled. However, enabling back EMF integration automatically enables current and voltage measurement.
- The Back-EMF integration algorithm provides a useful technique for controlling sensorless BLDC motors. The Back-EMF sensing technique is based on the fact that only two phases of a Brushless DC motor are energized at a time. The third phase is a disconnected phase which can be used to sense the Back-EMF voltage generated by the motor. Figure 15 shows a typical Back-EMF signal in one electrical revolution. The Deadtime Insertion option specifies which component inserts deadtime into the PWM signal.

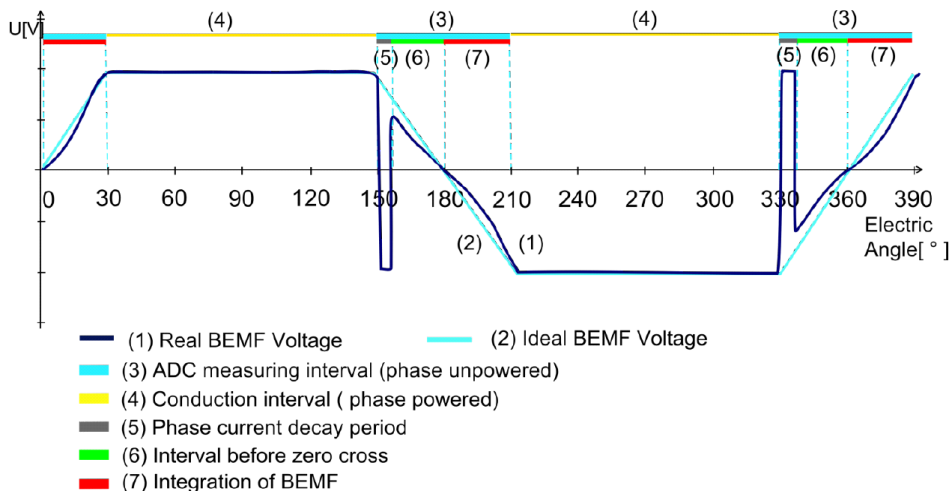


Figure 15. Single-phase Back-EMF Voltage

In Figure 15, interval (3) shows the ADC measuring Back-EMF voltage when the phase is unpowered. Because of the transient pulse after commutation in interval (5), the number of measured samples after commutation can be ignored. When a zero-cross occurs, the subsequent pulses are integrated—as shown in interval (7) — until the value reaches the integration threshold and commutation is performed. Back-EMF integration is fully implemented in the ThreePhaseMotorControl component, but the timer still controls the commutation. To commute using the BEMF integration value, disable the commutation timer and use an alternative speed controller controlling the torque of the motor to reach the desired speed. The integration threshold remains constant for all speeds (see Figure 16).

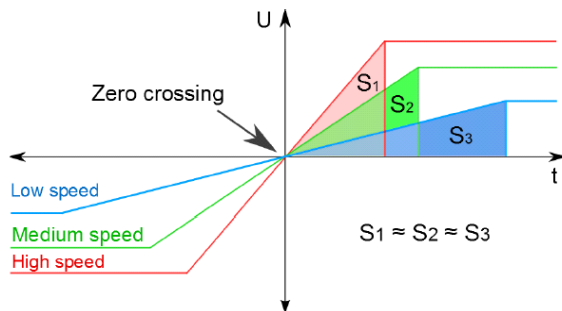


Figure 16. Back-EMF Integration Threshold

Figure 17 shows the properties displayed when measurement and integration are enabled.

Motor Control	Enabled	
Commutation Timer Link	Init_FTM_VAR0	
Commutation Timer Device	FTM1	FTM1
Direction	Clockwise	
Minimum RPM	60	<input type="button" value="D"/>
Initial RPM	500	<input type="button" value="D"/>
Alignment Period	1	<input type="button" value="D"/>
Freewheeling Period	3000	<input type="button" value="D"/>
Number of Pole Pairs	8	<input type="button" value="D"/>
Acceleration	100	<input type="button" value="D"/>
Torque Settings		
Alignment Torque	10	<input type="button" value="D"/>
Startup Torque	30	<input type="button" value="D"/>
Spin Torque	40	<input type="button" value="D"/>
Current and Voltage Measur	Enabled	
PDB Link	Init_PDB_VAR0	
ADC Link	Init_ADC_VAR0	
ADC Device	ADC0	ADC0
Supply Current Measure	Enabled	
Average Filter	Enabled	
Current Limit	0	<input type="button" value="D"/>
IDCB Pin	ADC0_SE4/ADC0_DM0/PTE21/FT...	ADC0_SE4/ADC0_DM0/PTE21/FTM1_CH1/UART0_RX
Supply Voltage Measure	Enabled	
VDCB Pin	ADC0_SE5/ADC0_DM1/ADC1_SE5/...	ADC0_SE5/ADC0_DM1/ADC1_SE5/PTE17/SPI0_SCK/UART1_RX/FT
BEMF Measurement	Enabled	
BEMF A Pin	ADC0_SE6/ADC1_SE1/ADC1_DP1/...	ADC0_SE6/ADC1_SE1/ADC1_DP1/PTE18/SPI0_SOUT/UART1_CTS_I
BEMF B Pin	ADC0_SE9/ADC1_SE9/PTB1/I2C0_S...	ADC0_SE9/ADC1_SE9/PTB1/I2C0_SDA/FTM1_CH1/FTM0_FLT2/EV
BEMF C Pin	ADC0_SE7/ADC1_SE7/ADC1_DM1/...	ADC0_SE7/ADC1_SE7/ADC1_DM1/PTE19/SPI0_SIN/UART1_RTS_b/
BEMF Integration	Enabled	
Zero-Cross Detection	SW	
Ignore Samples	15	<input type="button" value="D"/>
Auto Initialization	yes	

Figure 17. Measurement and Integration Properties

- The PDB Link option identifies the linked Programmable Delay Block (PDB) component.
- The ADC Link option identifies the linked Analog-to-Digital Converter (ADC) component.
- The ADC Device option identifies the Analog-to-Digital Converter (ADC) device.

- The Supply Current Measurement options control DC bus current measurement settings.
 - Average Filter enables and disables the averaging filter. When set, the averaging filter calculates the average value from 100 samples.
 - Current Limit sets the desired current limit for monitoring purposes. The value is expressed in milliamps ranging from 0–8000 mA. When the limit is exceeded a corresponding flag is set (See [Figure 18](#)). Setting the value to 0 disables current limit measurement.
 - IDCB Pin specifies the ADC pin for DC bus current measurement.
- The Supply Voltage Measurement options control DC bus voltage measurement settings.
 - VDCB Pin specifies the ADC pin for DC bus voltage measurement.
- The BEMF Measurement options control Back-EMF measurement settings.
 - BEMF A Pin specifies the ADC pin for Back-EMF phase A.
 - BEMF B Pin specifies the ADC pin for Back-EMF phase B.
 - BEMF C Pin specifies the ADC pin for Back-EMF phase C.
- The BEMF Integration options enable and disable Back-EMF integration motor control.
 - Zero-Cross Detection specifies whether to use a hardware or software zero-cross detection method. When the option is set to HW (hardware), zero-cross detection is handled by feedback pins in TPP_MC33937. When set to SW (software), zero-cross detection must be handled by software monitoring of the BEMF voltage (notice that in [Figure 17](#), Zero-Cross Detection option is set to SW and is Read-only). This is because the feedback pins aren't connected on the TWR-MC-LV3PH.
 - Ignore Samples specifies what percentage of samples should be ignored in the commutation period following a commutation.
- The Auto Initialization option, when set, causes the `Init` method to be called automatically during all CPU peripheral initialization. If this option is not set, the code must call the `Init` method.

4.2.2 Component API

Component API consists of the methods and events listed in [Table 2](#). Most of the parameters from component properties and those used in methods and events are stored in a special structure accessible to the user. The structure differs depending on which features of the component are enabled. Some parameters are crucial for the proper functioning of the application. These parameters are marked with a red dot in [Figure 18](#) and should not be changed while the application is running. This structure also contains all measured values (yellow dot) and flags for signalization (green dot). All the other parameters can be changed at will. Whenever possible, use prepared methods to change the parameters rather than changing them directly.

```

/* Device state and motor control settings. */
typedef struct {
    TPMC1_TMotorControlState MotorState;           /* current motor state */
    TPMC1_TMotorRotationDirection Direction;       /* direction of motor rotation */
    uint16_t Acceleration;                         /* acceleration in RPM/s */
    ● uint8_t PolePairs;                           /* number of pole pairs of the rotor - do not change while running */
    uint32_t AlignmentPeriod;                     /* duration of alignment of the rotor */
    uint32_t FreewheelingPeriod;                 /* duration of staying in FREEWHEEL state before transition to READY state */
    uint8_t AlignmentTorque;                     /* percent of torque/current during alignment */
    uint8_t StartupTorque;                       /* percent of torque/current during startup */
    uint8_t SpinTorque;                          /* percent of torque/current during spinning */
    uint16_t MinimumSpeed;                       /* minimum speed */
    uint16_t Speed;                              /* desired speed */
    ● uint16_t OldSpeed;                          /* current speed - do not change! */
    ● uint16_t Modulo;                            /* modulo of commutation timer - do not change! */
    ● uint8_t CommutationSector;                 /* current commutation sector - do not change! */
    uint16_t ReferenceVoltage;                   /* reference voltage of ADC */
    ● uint32_t IDCBmA;                           /* measured DC bus current in mA */
    ● uint32_t VDCBmV;                           /* measured DC bus voltage in mV */
    ● uint32_t PhaseBEMF;                        /* measured Back-EMF voltage in mV */
    ● uint32_t IntegrationValue;                 /* integrated Back-EMF */
    uint8_t IgnoreSamples;                       /* percent of ignored samples after commutation */
    ● uint32_t SampleCnt;                        /* measurement samples counter - do not change! */
    ● bool CommutationFlag;                      /* commutation occurred */
    ● bool SpeedControlFlag;                    /* set speed changed - checked in PeriodicUpdate() */
    ● bool FullSpeedFlag;                       /* set speed reached */
    ● bool CurrentLimitFlag;                    /* current higher than limit */
    ● bool BEMFlowFlag;                         /* polarity of Back-EMF - true = under VDCB/2, false = over VDCB/2 */
    ● bool IntegrationFlag;                     /* Back-EMF measured and prepared for integration - checked in PeriodicUpdate() */
    TUserDataPtr UserDataPtr;                   /* void pointer to user specified data */
} TPMC1_TDeviceData, *TPMC1_TDeviceDataPtr; /* Device data pointer */
    
```

Figure 18. Device Data Structure

Figure 19 illustrates the methods available depending on which component features are enabled. A tick in the lower left corner of an icon indicates the associated item will be generated. A cross indicates the item is not generated. Methods and events whose names are rendered in black can be enabled or disabled by discretion. Items whose names are rendered in gray are Read-only and are either required by the application or not supported.

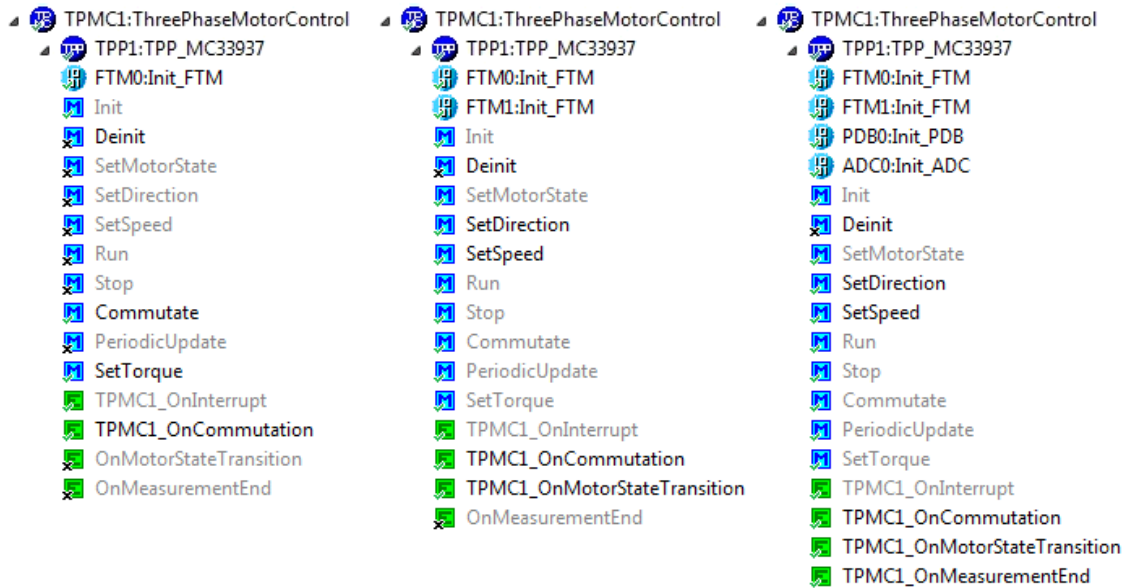


Figure 19. Components and Methods/Events with Motor Control Disabled (left), Motor Control Enabled (middle), and Measurement with Integration Enabled (right)

Table 2 summarizes all TPMC component API methods and events.

Table 2. TPMC Component API Methods/Events

Method / Event	Description
Init	Initializes the component
Deinit	Deinitializes the component
SetMotorState	Sets motor state
SetDirection	Sets direction of motor spinning
SetSpeed	Sets speed of motor spinning
Run	Starts motor spinning
Stop	Stops motor spinning
Commutate	Commutes the rotor to selected sector
PeriodicUpdate	Method for checking flags set by other functions (must be called periodically)
SetTorque	Sets torque in given motor state
OnInterrupt	This event handler is invoked when one or more device internal events occurred while corresponding interrupts are enabled
OnCommutation	This event is invoked after commutation
OnMotorStateTransition	This event is invoked after any transition to another motor state
OnMeasurementEnd	This event is invoked after an ADC measurement completes

4.2.3 Motor State Management

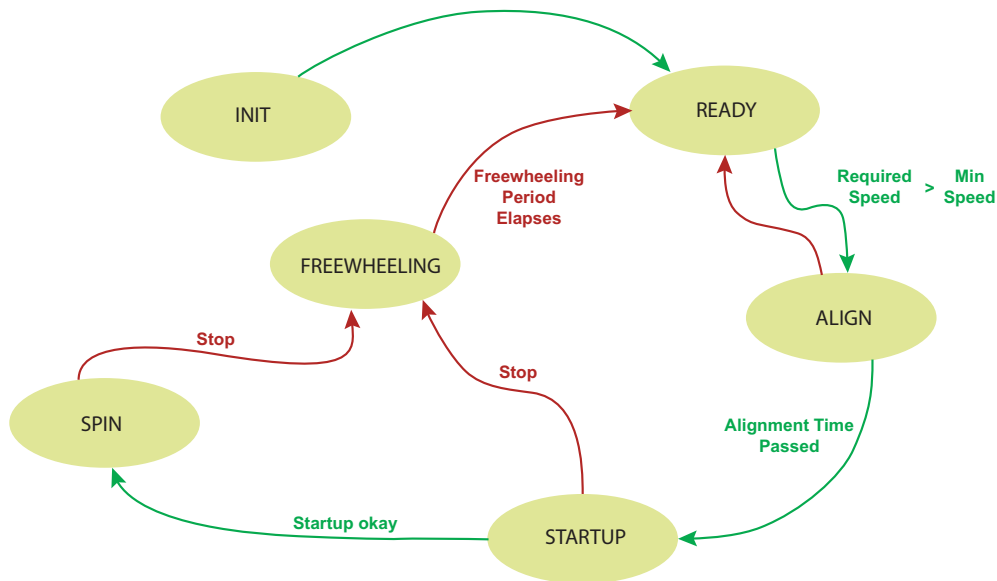


Figure 20. Motor State Machine

As indicated in the state diagram in Figure 20, there are six states that the motor can be in.

- **Init** - The state the application is in immediately after a call to INIT.
- **Ready** - An application wait state. The pre-driver is in standby mode and the commutation timer is stopped.
- **Align** - An internal state between Ready and Startup. The motor's rotor is aligned to its starting position. Transition to the Startup state is automatic once the alignment period elapses.
- **Startup** - this state enables the commutation timer and accelerates the motor to its initial speed. Measurement (if enabled) also starts in this state.
- **Spin** - In this state, the motor has reached its initial speed. The motor speed can subsequently be changed without leaving the Spin state.
- **Freewheel** - This state involves the controlled stopping of the motor. The commutation timer and measurement are disabled. When the Freewheel period elapses, the application automatically transitions to the Ready state (assuming the Stop method is used).

The state machine in Figure 20 is not definitive. The SetMotorState method implements other solutions. Every motor state transition (OnMotorStateTransition) actually consists of two states (the parameters of the original state and those of the current state) and application code can be implemented to execute between these two states.

4.3 Known Issues with Both Components

Be aware of the following issues with the ThreePhaseMotorControl and the TPP_MC33937 components:

- The current implementation allows for only a single instance of TPP component. This issue is caused by a bug in the SPI_Device component and will be fixed in a future release.
- Using TPP methods which work internally with the SPI in interrupt service routines might result in a deadlock condition because of priority issues.
- The TPMC component is currently limited to a single instance in a project. This is caused by a TPP limitation and by limited hardware resources.
- Two false warnings are issued by the Init_ADC component:

Warning: Channel does not allow single mode

Warning: The frequency of the ADC clock is above the 16 MHz limit in high-speed conversion mode

Ignore these warnings. Both features work, in spite of the warning message.

5 Installing the Software and Setting up the Hardware

This section describes how to install CodeWarrior 10.6 and how to use Processor Expert for application development. Processor Expert software is available as part of the CodeWarrior Development Studio for Microcontrollers, Kinetis Design Studio, or as an Eclipse-based plug-in for installation into an independent Eclipse environment (Microcontroller Driver Suite). For more information about Processor Expert refer to this link:

www.freescale.com/tools/embedded-software-and-tools/software-development-tools/processor-expert-and-embedded-components:BEAN_STORE_MAIN.

For installation and usage of other development environments (for example IAR Embedded Workbench) please refer to "Jump Start Your Design" section on TWR-MC-LV3PH product summary page: www.freescale.com/TWR-MC-LV3PH.

5.1 Installing CodeWarrior and Creating a Project

5.1.1 Installing CodeWarrior on the Computer

This procedure explains how to obtain and install the latest version of CodeWarrior on the system. If you have already installed CodeWarrior 10.6 or later, skip this section.

1. Obtain the latest CodeWarrior installer file from the Freescale CodeWarrior website: www.freescale.com/webapp/sps/site/homepage.jsp?code=CW_HOME.
2. Run the executable file and follow the instructions.
3. In the GUI window select the components to install (see Figure 21). Select the Kinetis component and click on **Next** to complete the installation.

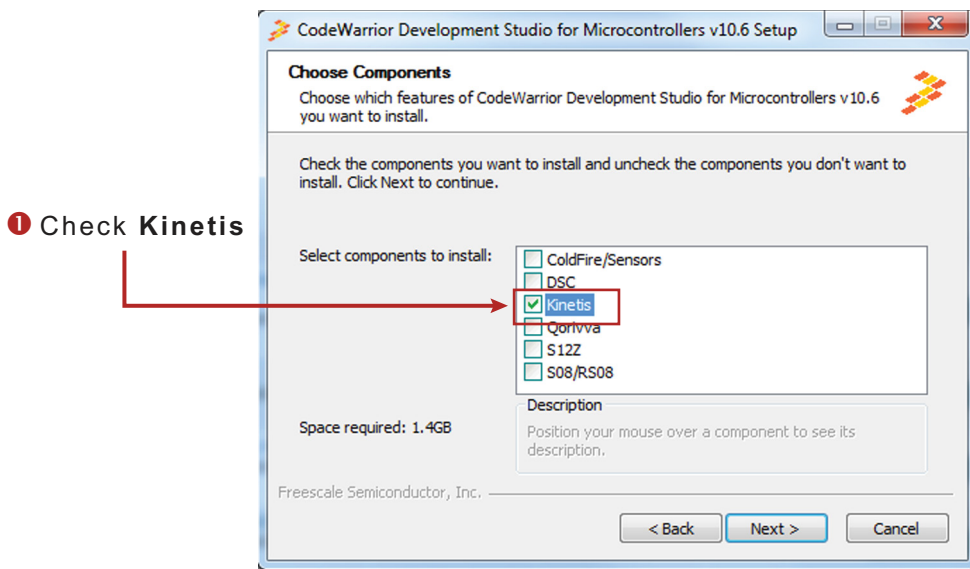


Figure 21. Choose Component GUI

5.1.2 Get the Components and Example Project

Two components have been developed based on customer requirements and the need for flexible solutions. The first component (TPP_MC33937) controls device configuration (using SPI) and the handling of operational modes. The second component (ThreePhaseMotorControl) is built upon the TPP_MC33937 and extends its functionality with BLDC motor sensorless control logic.

This procedure explains how to obtain the latest version of the components and example projects.

1. Download zip file with components and example projects from TWR-MC-LV3PH product summary page: www.freescale.com/TWR-MC-LV3PH.
2. Unzip the downloaded file and check the folder to confirm that it matches the items in Table 3.

Table 3. Zip Package Content

Folder Name	Folder Contents
Components	Processor Expert Components folder
TPP_MC33937_Bxxxx.PEUpd	Three-phase Pre-driver component
ThreePhaseMotorControl_Bxxxx.PEUpd	BLDC Motor control component
SPI_Device_Bxxxx.PEUpd	SPI communication component
CodeWarrior_Examples	Example projects folder for CodeWarrior.
TPP_KV10Z32_tpp_setting	This example project shows how to set Three Phase Pre-driver using TPP component methods and handle errors which might occur.
TPP_KV10Z32_bldc_Freemaster	This example project demonstrates how to drive a brushless DC motor using Three Phase Pre-driver component (TPP) and FreeMASTER application.
TPMC_KV10Z32_bldc_Freemaster	This example project shows how to use FreeMASTER and the Three Phase Motor Control (TPMC) component to drive a brushless DC motor.
TPMC_KV10Z32_bldc_control	This example shows how to use TPMC (Three Phase Motor Control) component to change speed and direction of motor rotation. The example also shows how to handle MC33937 (Three Phase Pre-driver) interrupts.
KDS_Examples	Example projects folder for Kinetis Design Studio.
TPP_KV10Z32_tpp_setting	See CW example description.
TPMC_KV10Z32_bldc_control	See CW example description.
TPMC_KV31F120M_basic_config	This example shows how to use the TPMC (Three Phase Motor Control) component in basic a configuration and user implemented commutation timing.
TPMC_KV31F120M_measurement	This example shows how to use the TPMC (Three Phase Motor Control) component for BLDC motor running and feedback measurement.

These components have been validated for use in the following IDE's:

- CodeWarrior 10.6 and above.
- Kinetis Design Studio 2.0.0 and above.
- Driver Suite 10.4 and above.
- Processor Expert version RT6 and above.

5.1.3 Import Components to CodeWarrior

To import the components into CodeWarrior, do the following:

1. Launch CodeWarrior and click **Processor Expert -> Import Component(s)** in the menu bar.
2. Find and select all package files (.PEUpd) in the sub-folder ThreePhaseMotorControl_PEx_SWComponents.
3. In the dialog box **Proceed Update?** click **Yes** (or **Yes to all** for more packages).
4. The update now proceeds.
5. After successful installation of the update package(s), the newly installed component(s) become available in Components Library view (See [Figure 22](#)).

The components depend on each other in the following order: TPMC uses TPP which uses SPI_Device.

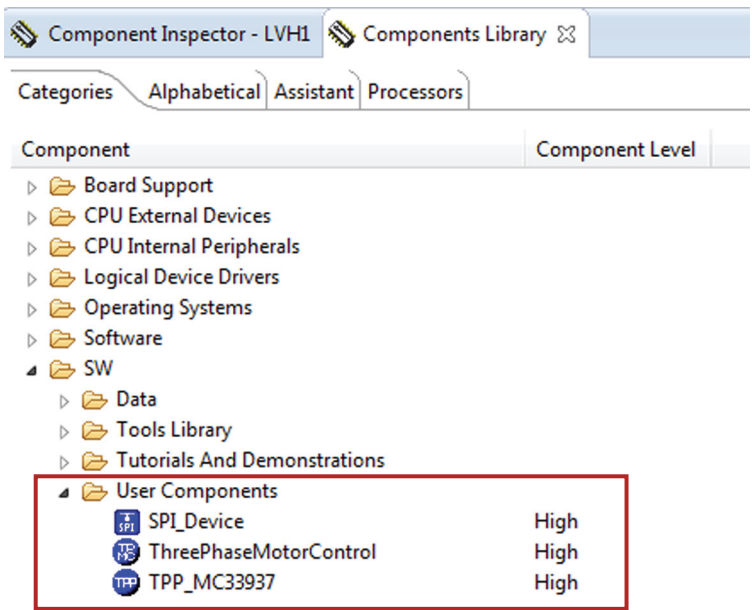


Figure 22. Component Library View

5.1.4 Create a New Project with the TPMC Component

To create a new project, do the following:

1. In the CodeWarrior menu bar, click **File -> New -> Bareboard Project**. Enter a project name in the **Project name** box. Then click **Next**.

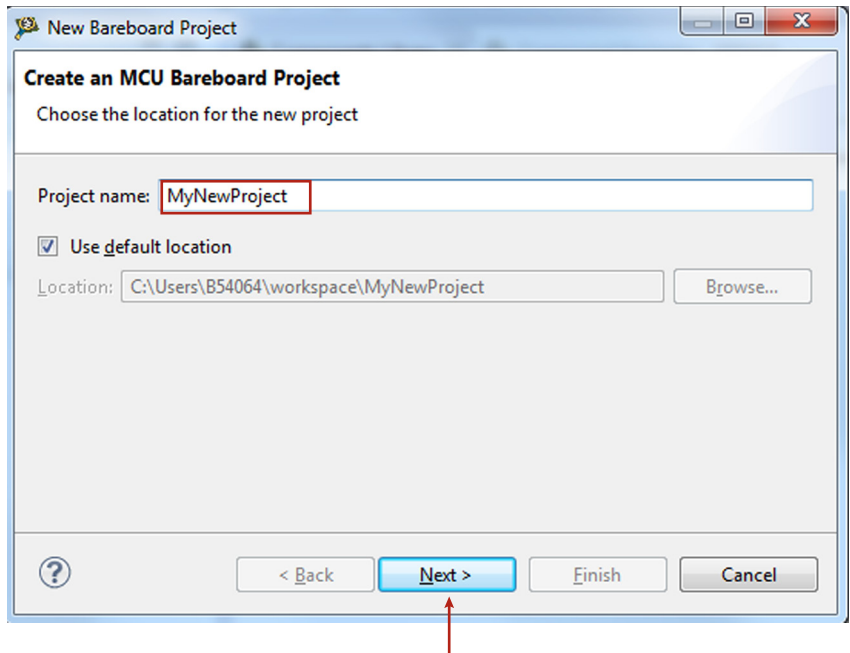


Figure 23. New Bareboard Project

2. In the **Devices** dialog box, select which MCU the project uses. The example in [Figure 24](#) uses the MKV10Z32.

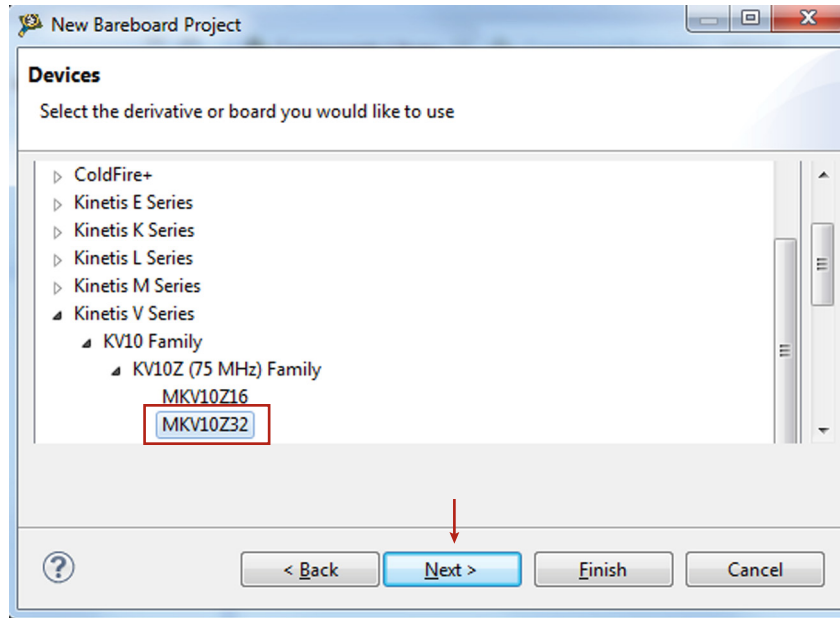


Figure 24. Select MCU

3. Select the connections to be used. The example uses OpenSDA.

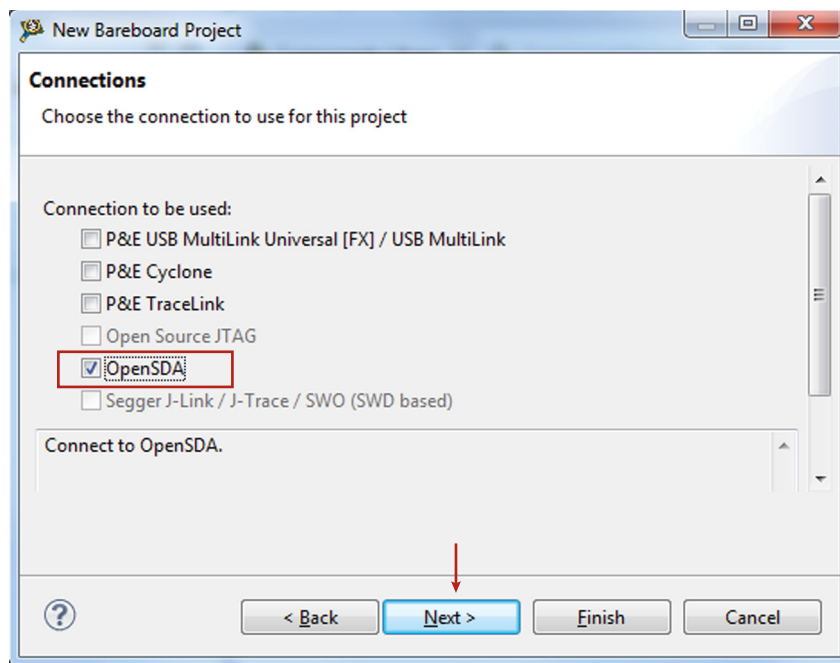


Figure 25. Select Connections

4. Select the language and build tools the project uses. In the example, C language and GCC compiler are selected.

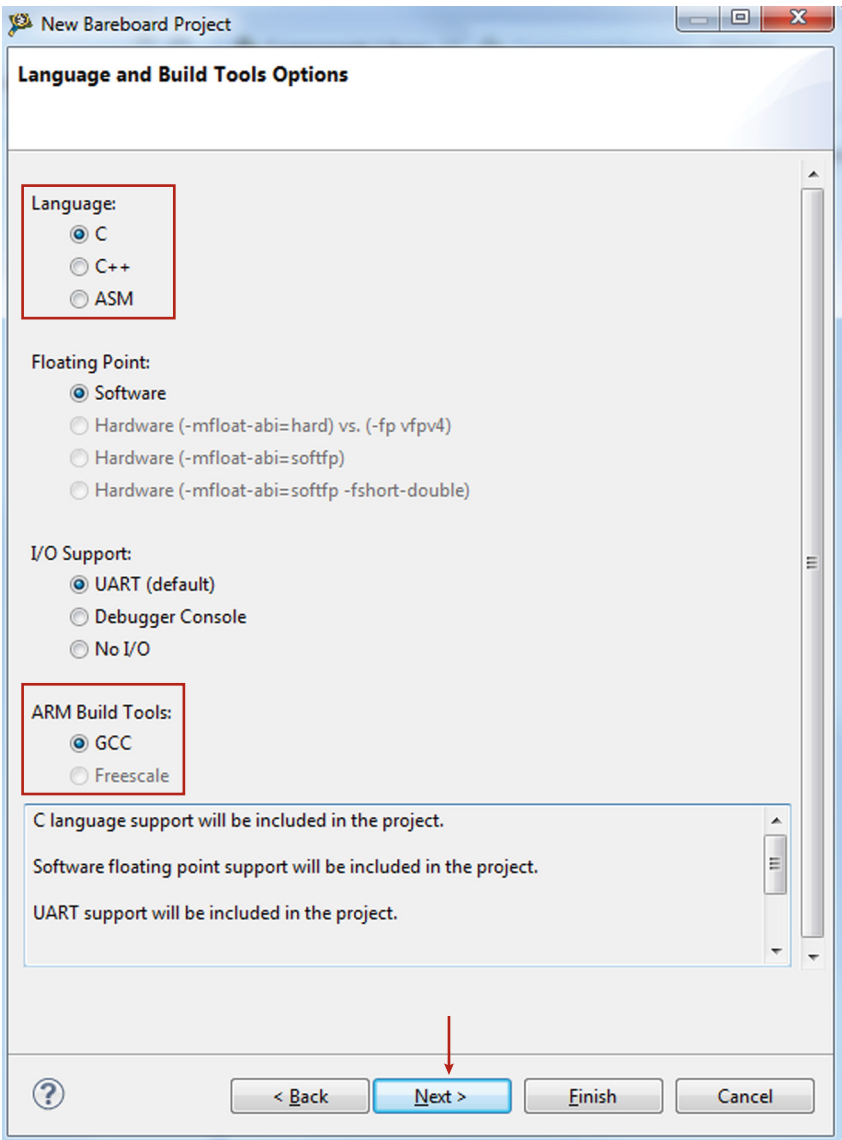


Figure 26. Select Language and Build Tools

- 5. Select the Processor Expert option in the Rapid Application Development screen. Then click **Finish**.

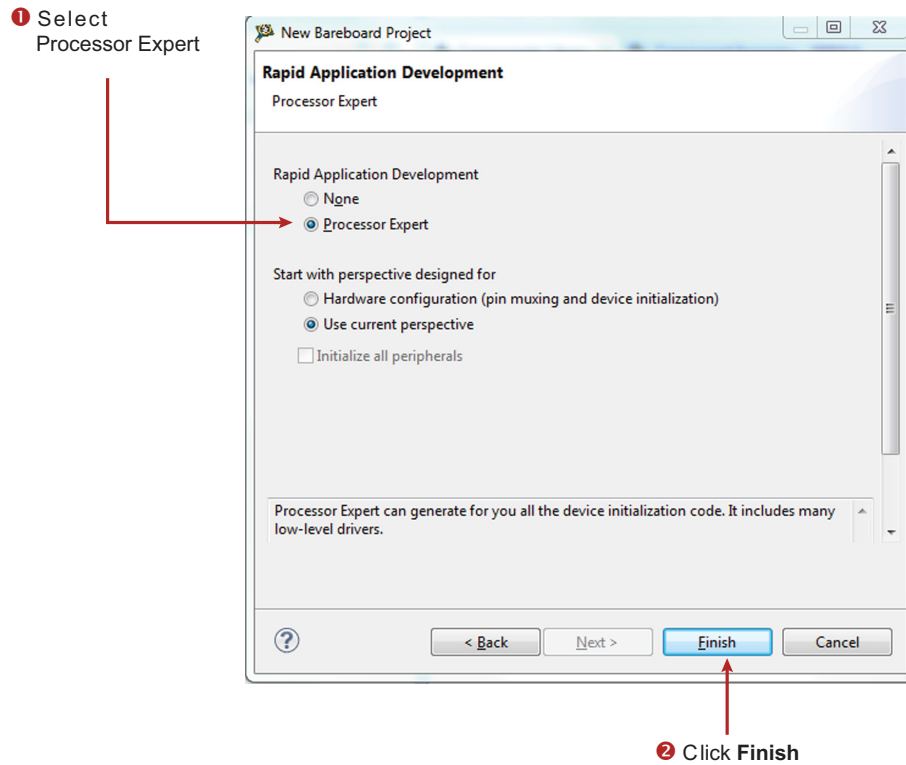


Figure 27. Rapid Application Development Screen

6. Find ThreePhaseMotorControl component in the PEx component library and add it to the project created.

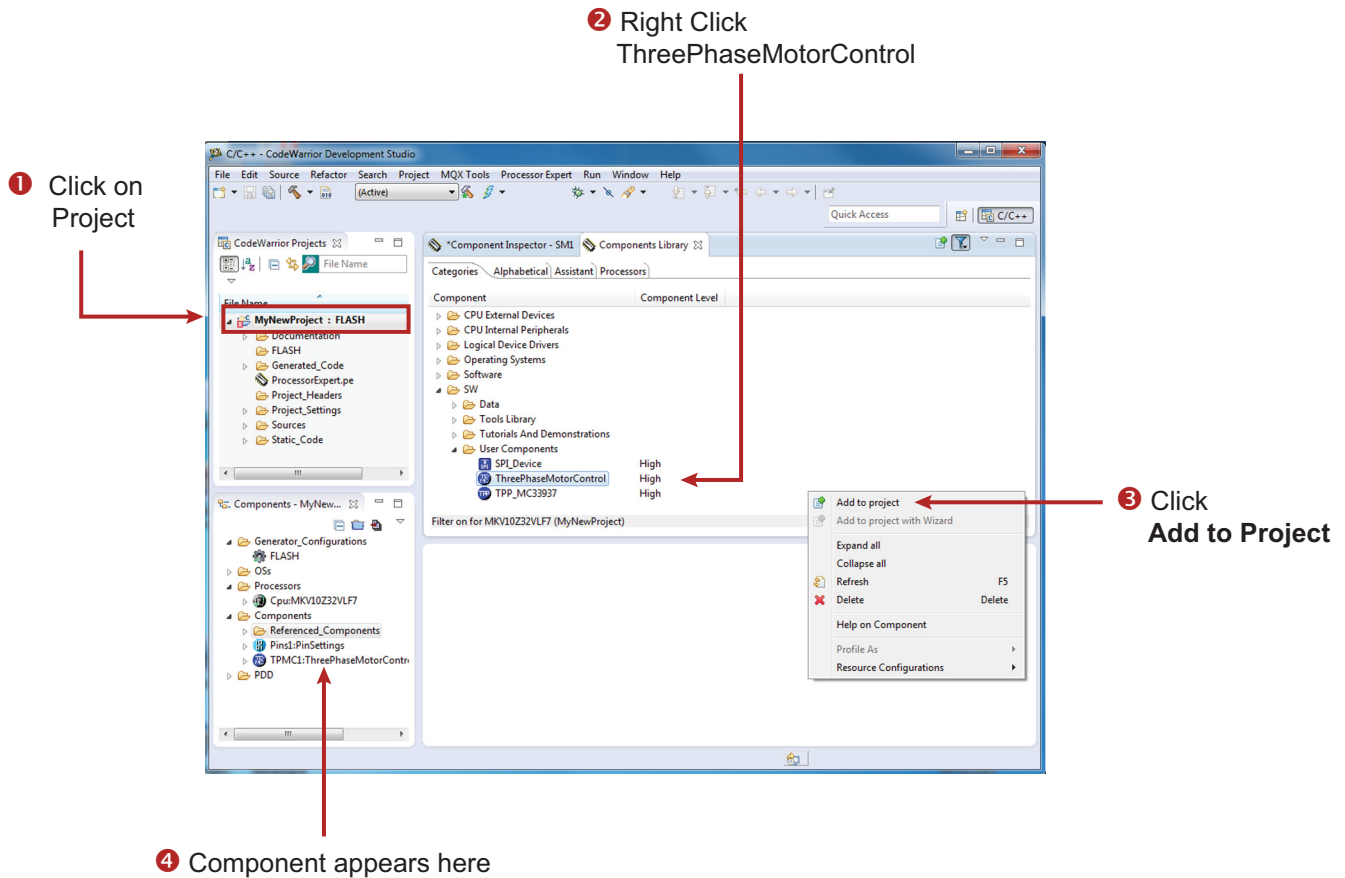


Figure 28. Add Component to Created Project

5.1.5 Configure the Project for the TWR-MC-LV3PH

Once a new project is created and added the ThreePhaseMotorControl component, the next step is to set the component options to match the requirements of the design being worked on. Section 4 provides information on all component options. Refer to this section to determine what settings apply to the environment.

Because there are interdependencies among the components, it is important that you configure the options in this order:

1. Set up the TPP component.
2. Set up the referenced SPI_Master_LDD component.
3. Set up the CS pin under inherited SPI_Device component.
4. Set up the TPMC component.

5.1.6 Generate the Source Code for the Application

With all the component options configured, generate the source code for the application:

1. In the Components panel of the project, highlight the **TPMC1:ThreePhaseMotorControl** item.
2. Click on the **Generate Processor Expert Code** icon in the Components panel menu bar.

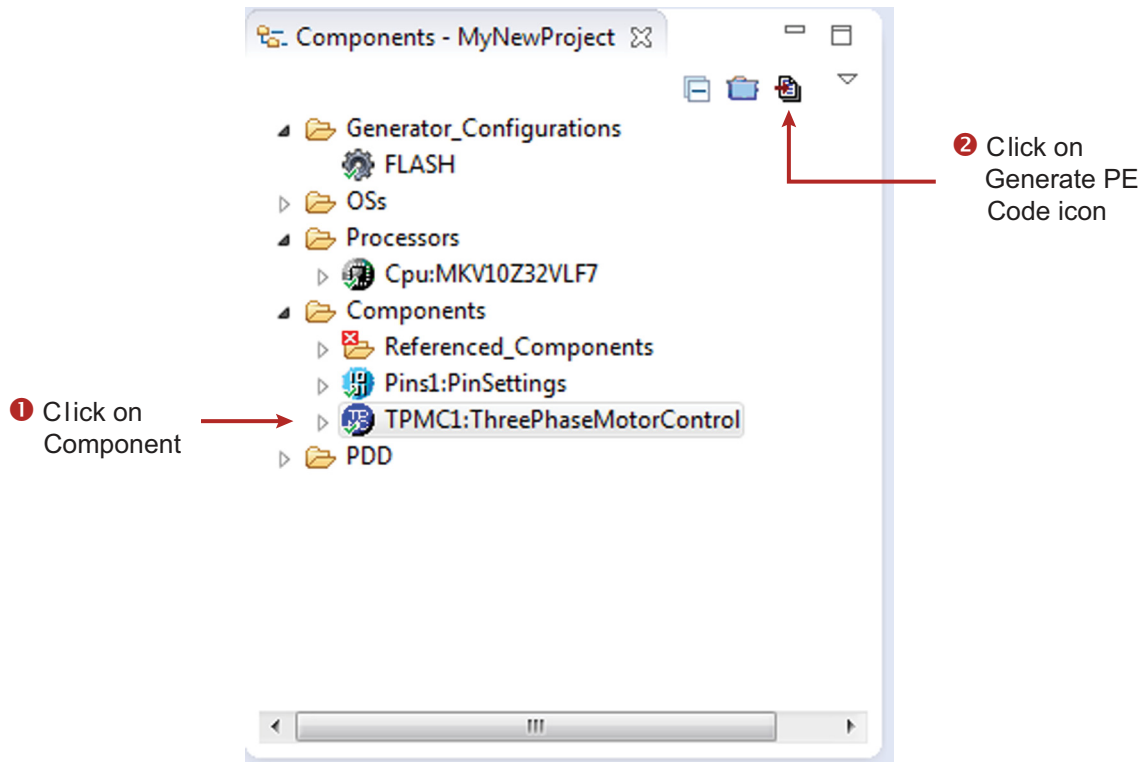


Figure 29. Generate Source Code

Processor Expert places the generated source code into a separate folder named **Generated_Code** in the project directory.

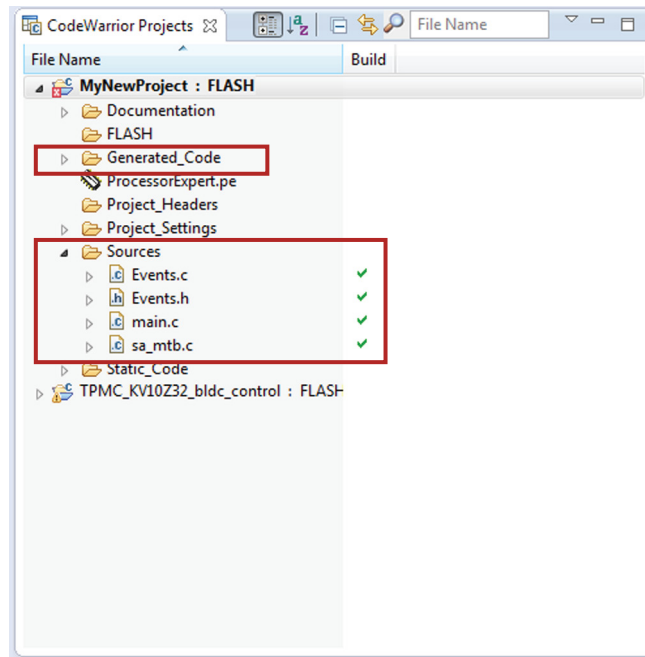


Figure 30. Generated Code and User Application Code

5.1.7 Implement Application Code

While Processor Expert generates the driver code based on the configuration options, develop and implement the application code. All of the application code must reside in the Sources folder in the project directory. Modify the code in main.c and Events.c, but retain the original comments related to usage directions.

5.1.8 Build the Project

When all of the code is created, the final step is to build the project.

1. Highlight the name of the project file.
2. In the Processor Expert tool bar, click on the **Build** icon.

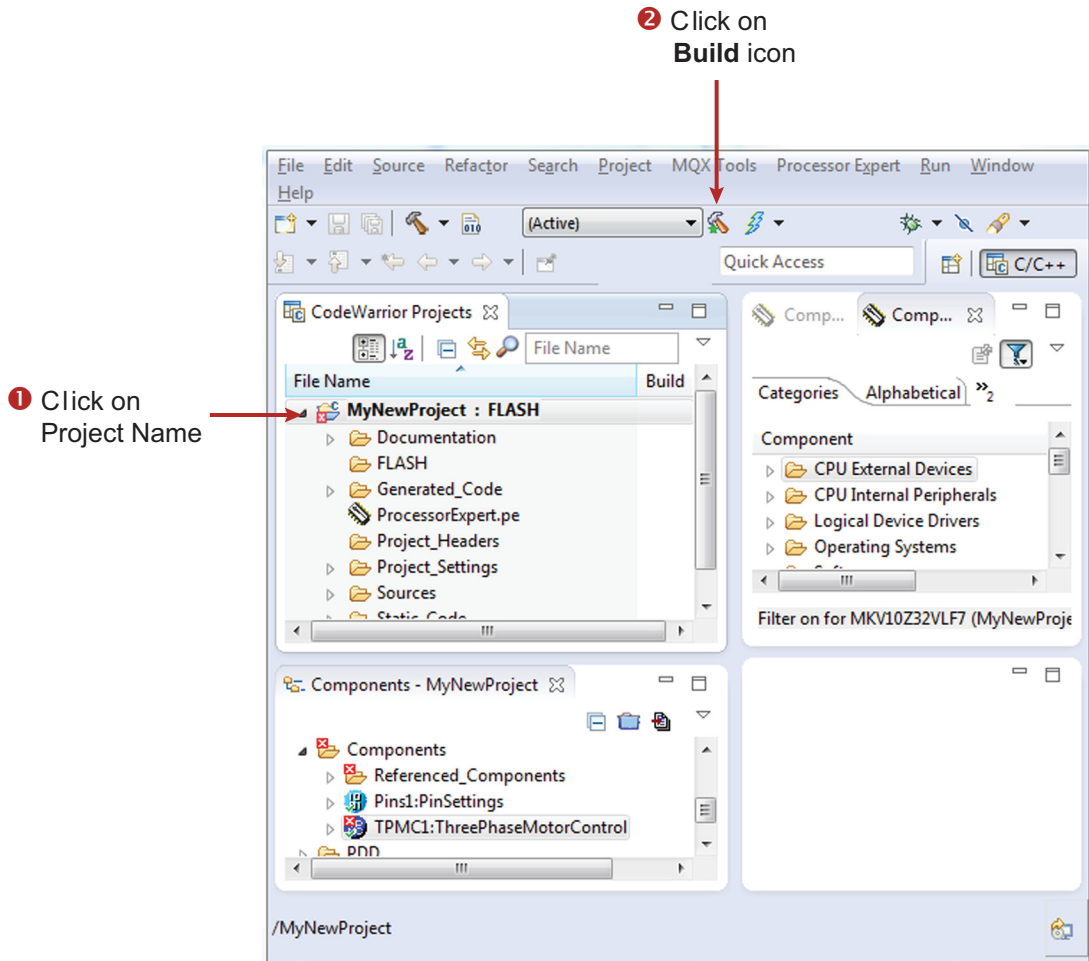


Figure 31. Build the Project

If project fails to compile, check whether at least the ISO C90 standard with GNU extensions in the compiler settings is selected.

In CodeWarrior

1. Right click on the project in CodeWarrior Project's window and select **Properties**.
2. Go to **C/C++ Build -> Settings -> Tool Settings -> ARM GCC C Compiler -> Miscellaneous**.
3. In Language Standard list select **ISO C90 with GNU extensions** (which should be the compiler default value).

In Kinetis Design Studio

1. Right click on the project in KDS Project Explorer window and select **Properties**.
2. Go to **C/C++ Build -> Settings -> Tool Settings -> Cross ARM C Compiler -> Optimization**.
3. In Language Standard list select **Toolchain default (GNU ISO C90)**.

5.1.9 Importing an Example Project

The zip file downloaded in [Section 5.1.2](#) already contains configured example projects. To import these examples into CodeWarrior, do the following:

1. In CodeWarrior, click **File** -> **Import...**
2. In the pop-up window, select **General** -> **Existing Projects into Workspace** and then click **Next**.
3. Browse for the downloaded example directory **ThreePhaseMotorControl_PEx_SW**. Open the CodeWarrior_Examples folder.
4. Select an example project(s) that you want to import and then click **Finish**.

The project is now in the CodeWarrior workspace to build and run. [Figure 32](#) illustrates this process.

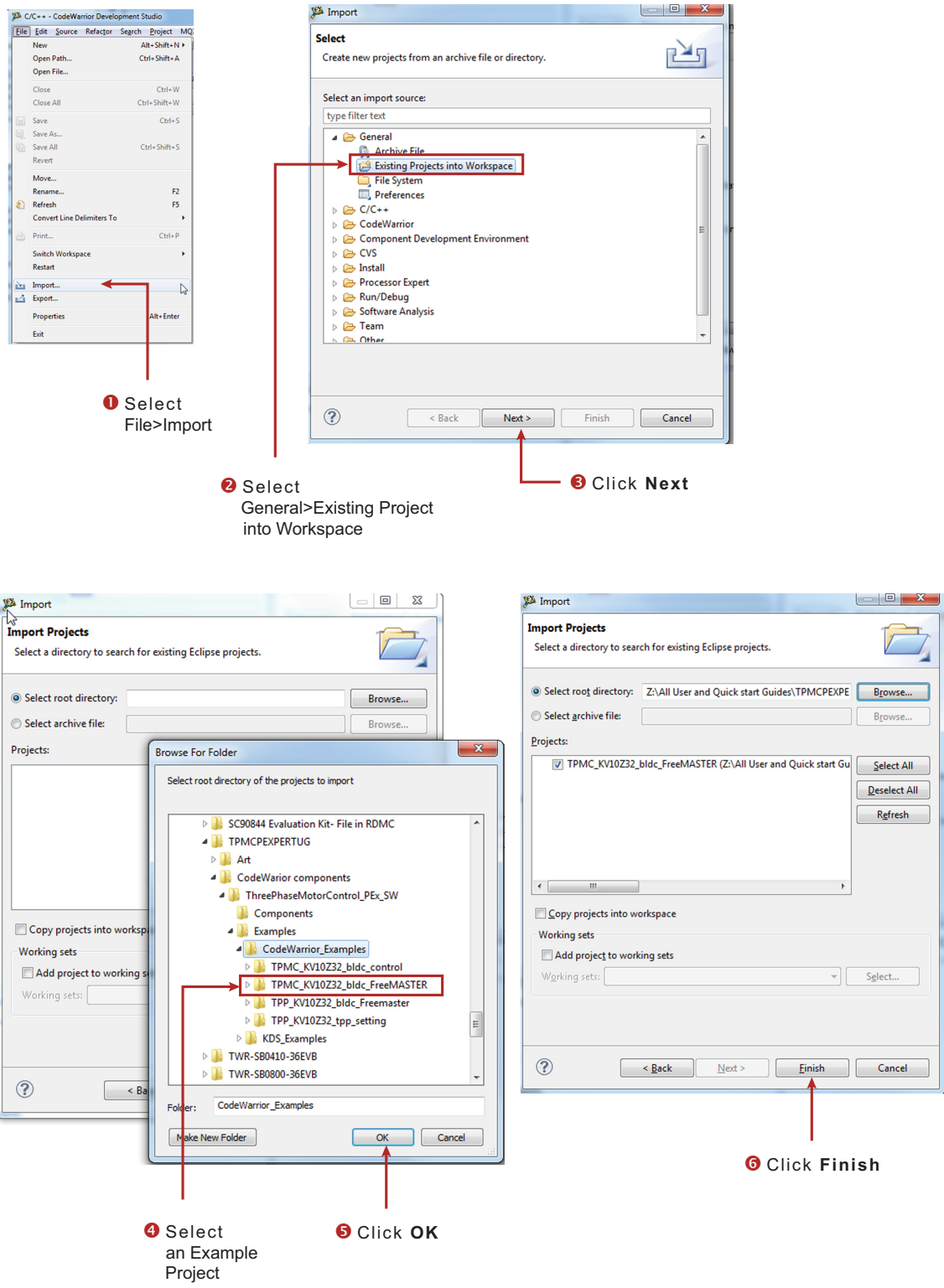


Figure 32. Importing an Example Project

5.2 Hardware Settings for Example Projects

The component package contains CodeWarrior example projects for the TWR-KV10Z32 board and Kinetis Design Studio projects for the TWR-KV10Z32 and the TWR-KV31F120M boards. To ensure that the example projects work, all board jumpers must be set properly. Each example project has a Readme file describing the required board settings.

5.2.1 Example Project Software Requirements

The following software is required when using the example projects:

- CodeWarrior 10.6 or above (for CodeWarrior projects)
- Kinetis Design Studio 2.0 or above (for KDS projects)
- FreeMASTER 1.4.4 plus FreeMASTER Communication Driver 1.8 or above (for projects using FreeMASTER)

5.2.2 Example Project Hardware Requirements

5.2.2.1 TWR-KV10Z32 Jumper Settings

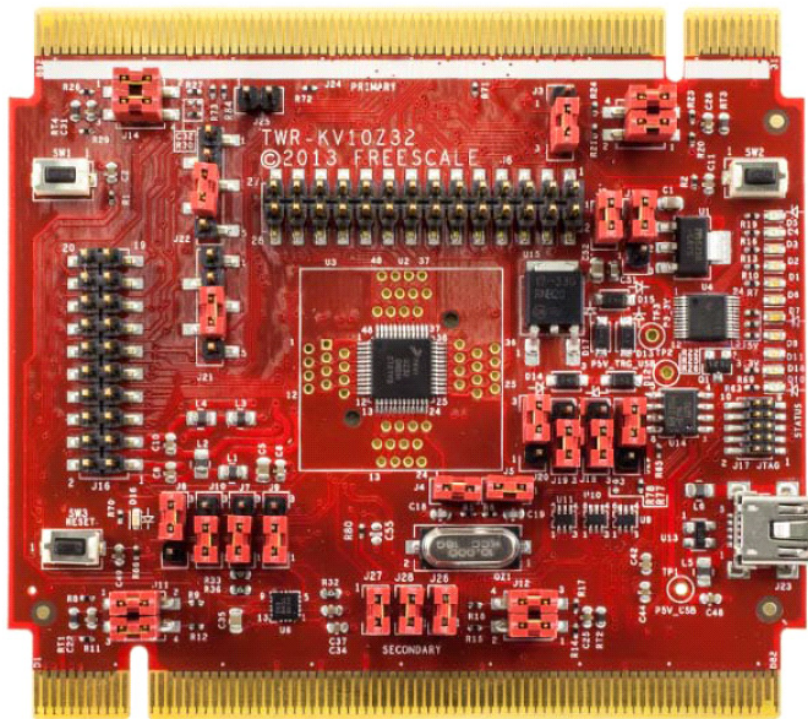


Figure 33. TWR-KV10Z32 jumper settings

5.2.2.2 TWR-KV31F120M Jumper Settings

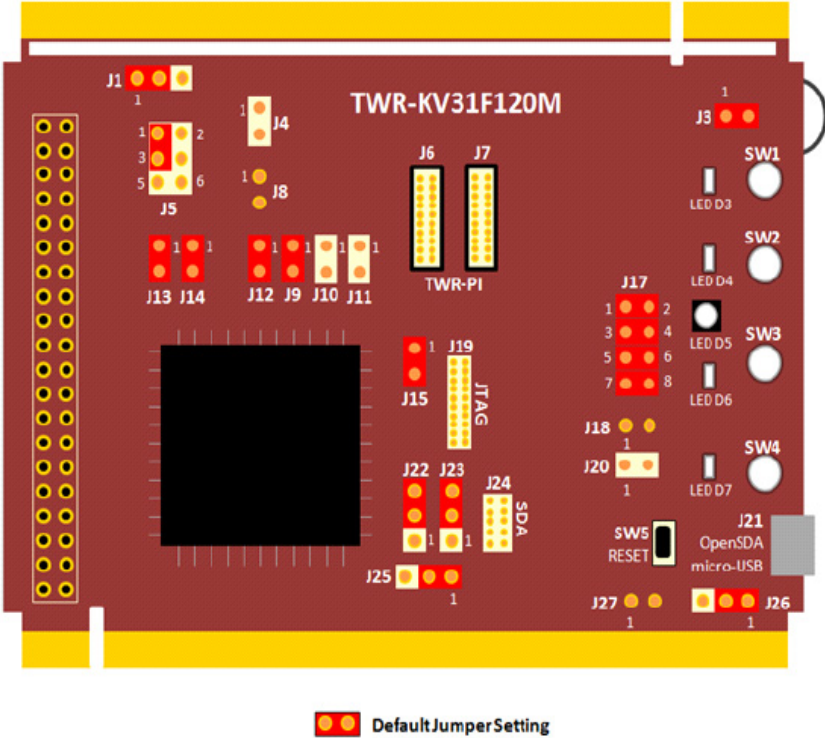


Figure 34. TWR-KV31F120M Jumper Settings

5.2.2.3 Additional Tower Module Settings

For both the TWR-KV10Z32 and the TWR-KV31F120M example projects, the following settings are typical:

- TWR-ELEV (Elevator Module) — The power switch is in the OFF position
- TWR-MC-LV3PH — Jumpers are set as shown in [Figure 35](#) (jumpers are in yellow)

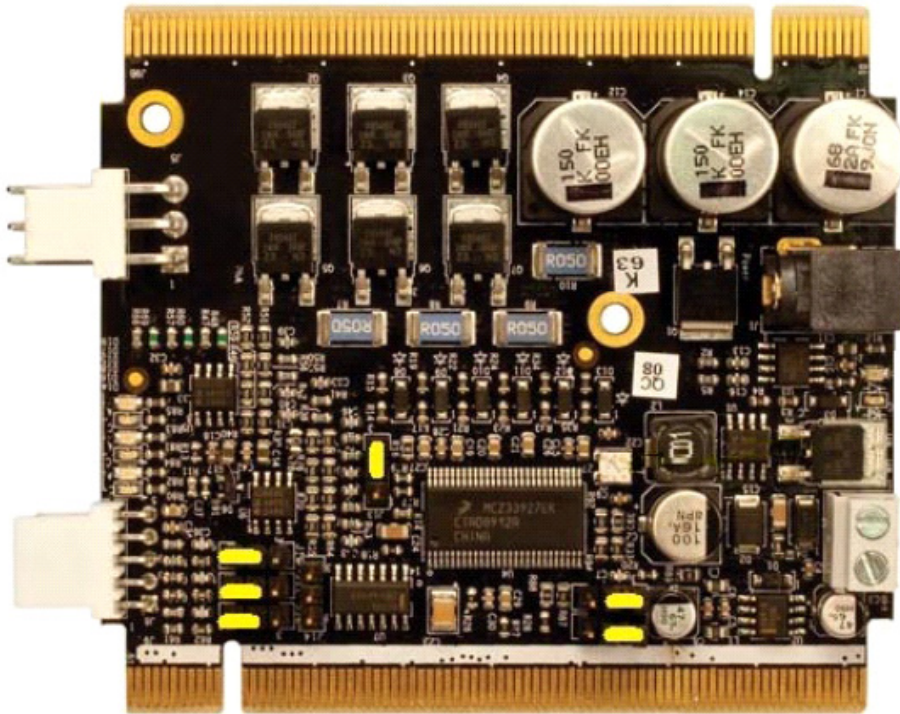


Figure 35. TWR-MC-LV3PH Jumper Settings

6 References

Freescale.com Support Pages	Description	URL
Processor Expert	User Guide	www.freescale.com/files/soft_dev_tools/doc/user_guide/CWPEXUG.pdf
CodeWarrior	Tool Summary Page	www.freescale.com/webapp/sps/site/homepage.jsp?code=CW_HOME
TWR-MC-LV3PH	Tool Summary Page	www.freescale.com/webapp/sps/site/prod_summary.jsp?code=TWR-MC-LV3PH
TWR-KV10Z32	Tool Summary Page	www.freescale.com/webapp/sps/site/prod_summary.jsp?code=TWR-KV10Z32
TWR-KV31F120M	Tool Summary Page	www.freescale.com/webapp/sps/site/prod_summary.jsp?code=TWR-KV31F120M
TWR-ELEV	Tool Summary Page	www.freescale.com/webapp/sps/site/prod_summary.jsp?code=TWR-ELEV

6.1 Support

Visit www.freescale.com/support for a list of phone numbers within your region.

6.2 Warranty

Visit www.freescale.com/warranty to submit a request for tool warranty.

7 Revision History

Revision	Date	Description of Changes
1.0	10/2015	<ul style="list-style-type: none"> • Initial release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions.

Freescale, the Freescale logo, CodeWarrior, Kinetis, and Processor Expert are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. SMARTMOS is a trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© 2015 Freescale Semiconductor, Inc.

Document Number: KTTPMCPEXPRTUG
Rev. 1.0
10/2015