CE218564 - Interfacing Quad-SPI Memory with PSoC® 5LP

Objective

This example demonstrates Quad SPI interface with a serial Cypress FRAM using PSoC® 5LP.

Overview

The objective of this code example is to interface Cypress' Quad-SPI F-RAM/nvSRAM/flash device with Cypress' PSoC 5LP controller. The code example has a Quad-SPIM User Component, designed specifically for Cypress Quad-SPI memories. The User Component is configurable for different frequencies. The User Component is imported into the code example; the usage of the supported APIs is shown in Application Programming Interface.

Requirements

Tool: PSoC Creator™ 4.1 See also Upgrade Information below.

Programming Language: C (GCC 5.4), ARM® Cortex®-M3 Assembler

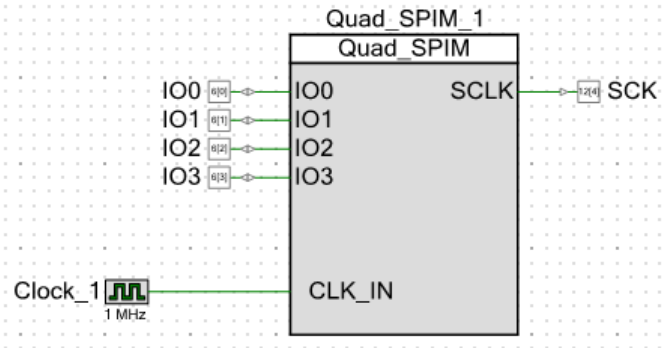
Associated Parts: All PSoC 5LP parts

Related Hardware: CY8CKIT-001

Design

The code example implements the Quad-SPI User Component with APIs to access Cypress Quad-SPI memories. These APIs include Quad-SPI memory read/writes and register read/write APIs.

Figure 1. Quad SPI Design Schematics



Design Considerations

The maximum possible serial clock frequency is 5 MHz.

Hardware Setup

A daughter board with memory must be mounted with the PSoC 5LP kit. Modify the pin out configuration for PSoC 5LP to match with the daughter board.

Components/User Modules

Table 1 lists the PSoC Creator Components/PSoC Designer user modules used in this example, as well as the placement/ hardware resources used by each.

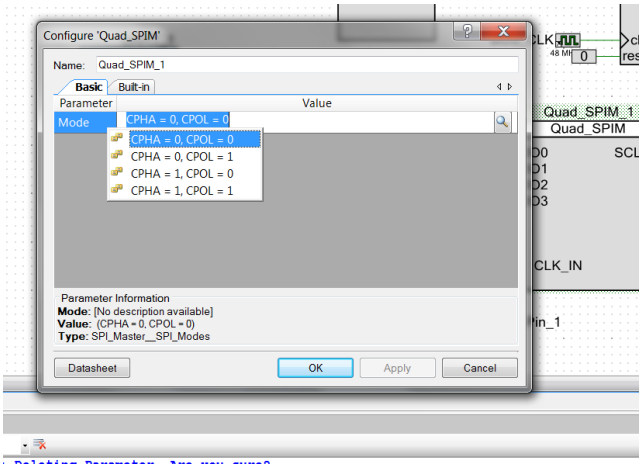
Table 1. List of PSoC Creator Components/PSoC Designer User Modules

Component or User Module	Version	Placement/HardwareResources
SPI Master	2.50	4 datapath cells, 52 macrocells, 1 control cell, and 2 interrupts
Control Register	1.80	1 control register
2:1 Multiplexer	1.10	2 multiplexers
Tri state buffer	1.10	4 tristate buffers
AND gate	1.00	2 AND gate

Parameter Settings

Double-click the Component to configure the quad serial peripheral interface (Quad SPI) Master Mode parameter.

Figure 2. QSPI User Module Configuration



Application Programming Interface

API routines allow you to configure the Component using software. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail. By default, PSoC Creator assigns the instance name "Quad_SPIM_1" to the first instance of a Component in a given design. You can rename the instance to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "Quad_SPIM".

API	Description
`Quad_SPIM_SPI_WRITE`	Memory write using SPI mode
`Quad_SPIM_SPI_READ`	Memory read using SPI mode
`Quad_SPIM_QPI_WRITE`	Memory write using Quad-SPI mode
`Quad_SPIM_QPI_READ`	Memory read using Quad-SPI mode
`Quad_SPIM_START`	Initializing routine for the Quad-SPIM Component
`Quad_SPIM_SPI_Reg_write`	Write memory register using SPI mode
`Quad_SPIM_SPI_Reg_Read`	Read memory register using SPI mode
`Quad_SPIM_QPI_Reg_write`	Write memory register using QPI mode
`Quad_SPIM_QPI_Reg_Read`	Read memory register using QPI mode
`Quad_SPIM_Erase`	Erase the memory (applicable only for flash memories)

void Quad_SPIM_SPI_WRITE(uint32 Address, uint8 *DATA, uint8 data_count)

Description: Write `data_count` number of data into memory in SPI mode.

Parameters: `uint32 Address`: 32-bit memory address for write

`uint8 *DATA`: Pointer to an array of data bytes to be written

`uint32 data_count`: Number of data bytes to be written

Return: None

Side Effects: None

void Quad_SPIM_SPI_READ(uint32 Address, uint8 *DATA, uint8 data_count)

Description: Read `total_data_count` number of data from memory in SPI mode

Parameters: `uint32 Address`: 32-bit memory address for read

`uint8 *DATA`: Pointer to an array for storing data bytes

`uint32 data_count`: Number of data bytes to be read

Return: None

Side Effects: None

void Quad_SPIM_QPI_WRITE(uint32 Address, uint8 *DATA, uint8 data_count)

Description: Write `data_count` number of data into memory in Quad-SPI mode

Parameters: `uint32 Address`: 32-bit memory address for write

`uint8 *DATA`: Pointer to an array of data bytes to be written

`uint32 data_count`: Number of data bytes to be written

Return: None

Side Effects: None

void Quad_SPIM_QPI_READ(uint32 Address, uint8 *DATA, uint8 data_count)

Description: Read `total_data_count` number of data from memory in Quad-SPI mode

Parameters: `uint32 Address`: 32-bit memory address for read

`uint8 *DATA`: Pointer to an array for storing data bytes

`uint32 data_count`: Number of data bytes to be read

Return:

Usage:

void Quad_SPIM_SPI_START (void)

Description: Initialization routine for the Quad_SPIM Component. Initializes the SPI blocks and CS pins.

Parameters: None

Return Value: None

Side Effects: None

`void Quad_SPIM_SPI_Reg_READ(uint8 opcode, uint8 *reg_value)`

Description: Read register in SPI mode

Parameters: `uint8 opcode`: 8-bit opcode for the register read

`uint8 *reg_value`: Pointer to the buffer for register read value

Return: None

Usage:

`void Quad_SPIM_QPI_Reg_READ(uint8 opcode, uint8 *reg_value)`

Description: Read status register in QPI mode

Parameters: `uint8 opcode`: 8-bit opcode for the register read

`uint8 *reg_value`: Pointer to the buffer for register read value

Return: None

Usage:

`void Quad_SPIM_SPI_Reg_WRITE(uint8 *reg_value, uint8 length)`

Description: Write status register in SPI mode

Parameters: `uint8 *reg_value`: 8-bit data buffer

`uint8 length`: number of register to write

Return: None

Usage:

`void Quad_SPIM_QPI_Reg_WRITE(uint8 *reg_value, uint8 length)`

Description: Write status register in QPI mode

Parameters: `uint8 *reg_value`: 8-bit data buffer

`uint8 length`: number of registers to write

Return: None

Usage:

Operation

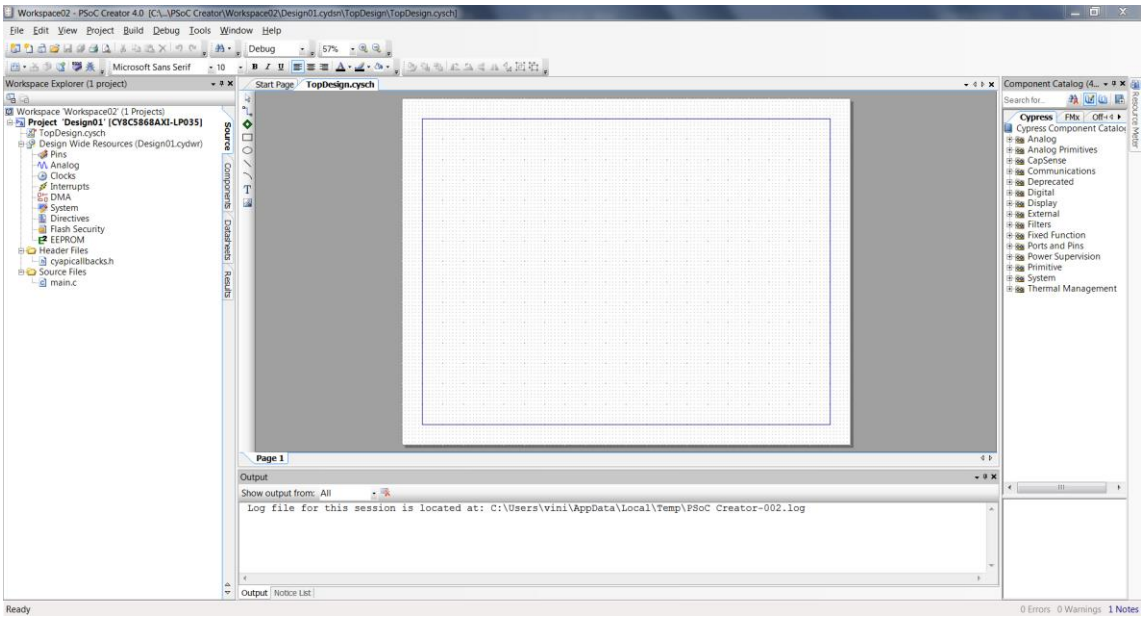
This section shows how to import the Quad SPI User Component into the code example and how to use its APIs.

Setup

The Quad_SPIM archive contains both the example project and Quad_SPIM Component.

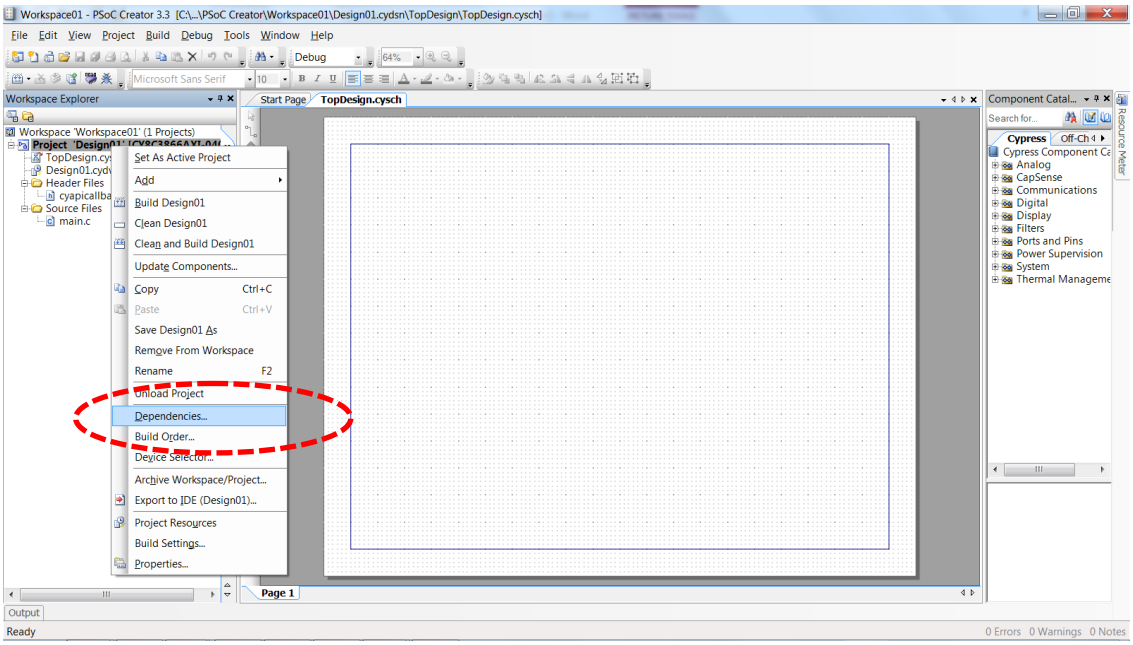
1. Open PSoC Creator and open your design (workspace) as shown in Figure 3. Create a new project named ‘Design01’.

Figure 3. Create Project 'Design01'



2. Right-click the project and open the **Dependencies** tab on the Workspace Explorer, and then bring QUAD_SPIM Component into your design, as shown in Figure 4.

Figure 4. Open Dependencies Tab



3. Click on **New Entry (User Dependencies)** and then select *CE218564_Quad_SPI.cyprj* from the *CE218564_Quad_SPI.cydsn* folder (see Figure 5). The QUAD_SPIM Component appears under default/QUAD_SPIM in Component Catalog (see Figure 6).

Figure 5. Importing User Component

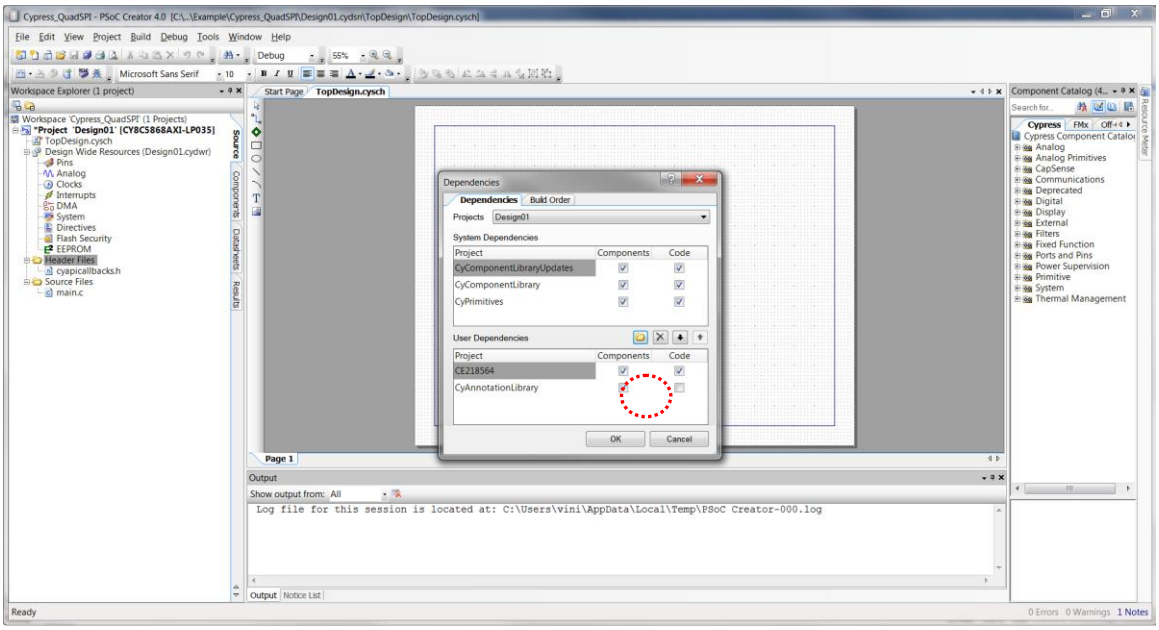
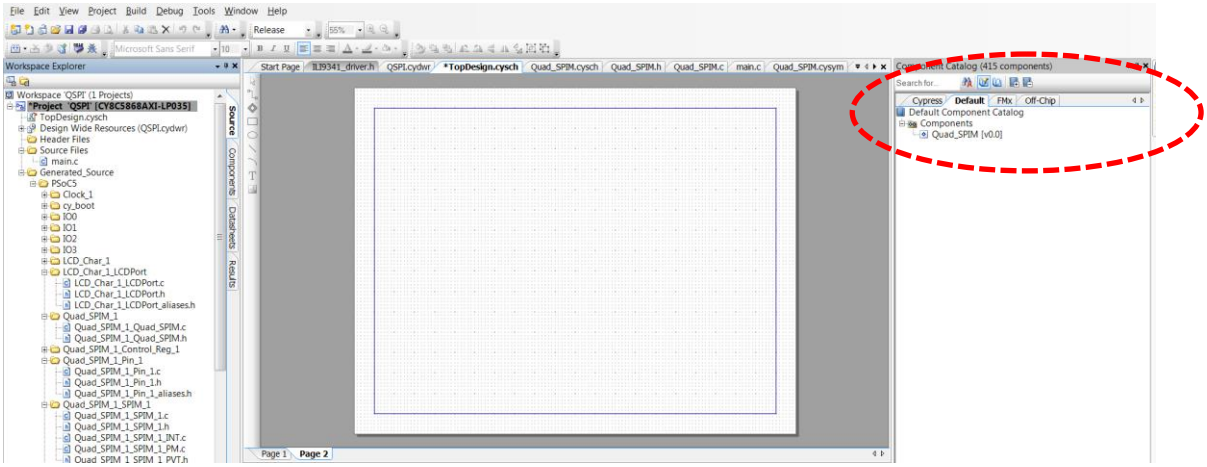
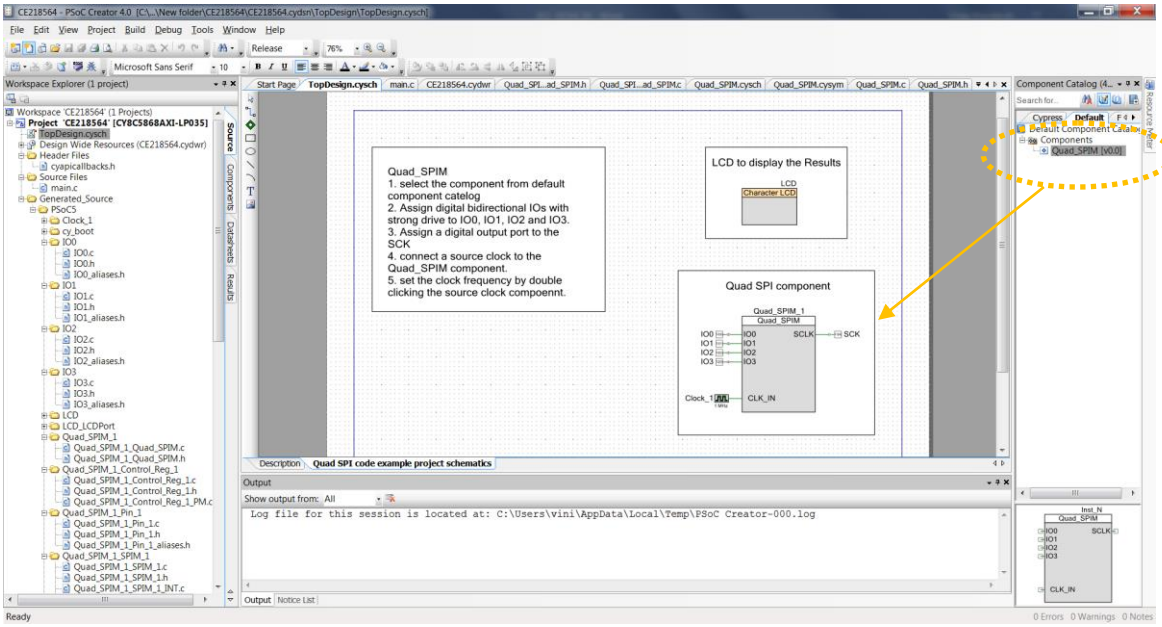


Figure 6. Quad_SPIM Component Under Component Catalog



4. Drag and drop the QUAD_SPIM Component onto *TopDesign.cysch* and assign Digital I/Os from the **Ports and Pins** Component, Clock source etc. as shown in Figure 7.

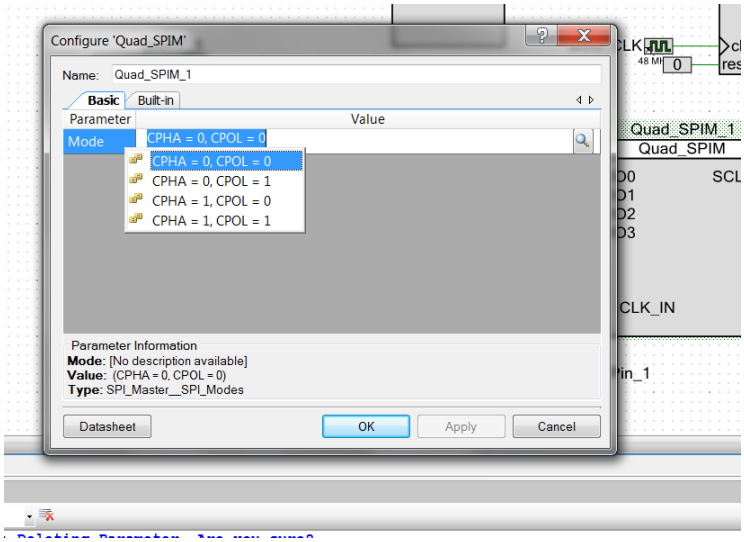
Figure 7. Quad_SPIM Component Usage



5. Do the following to configure the Quad_SPIM Component:

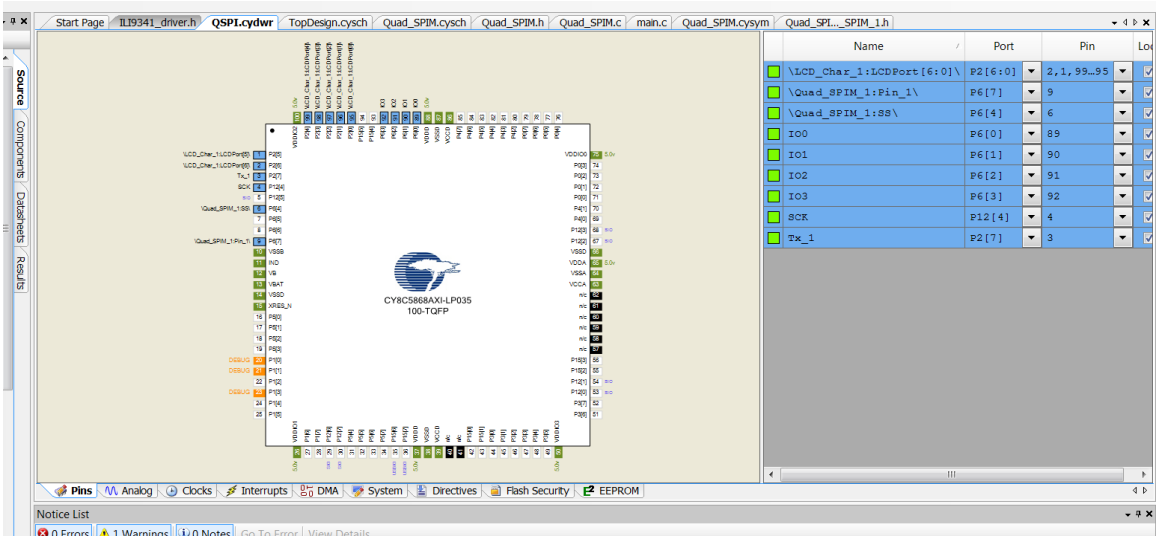
   a. Right-click the Quad_SPIM_1 Component in *TopDesign.cysch* and select **Configure.** Select Quad SPI Mode. This project uses mode 0.

Figure 8. User Component Configuration



   a. Assign appropriate input/output pins as per your design and build the project.

Figure 9. Pin Assignment



Exporting to Eclipse IDE

This section explains the steps necessary to build the exported project successfully on Eclipse IDE.

After exporting the project to Eclipse IDE, open it on Eclipse and exclude custom Component sources from build.

In Eclipse, right-click on the custom Components in the Eclipse > **Resource Configurations** > **Exclude from build** > **Select All,** and then click OK. Refer to PSoC Creator Help for exporting PSoC projects to Eclipse IDE.

Figure 10: Exclude Custom Component (Quad_SPIM) in Eclipse

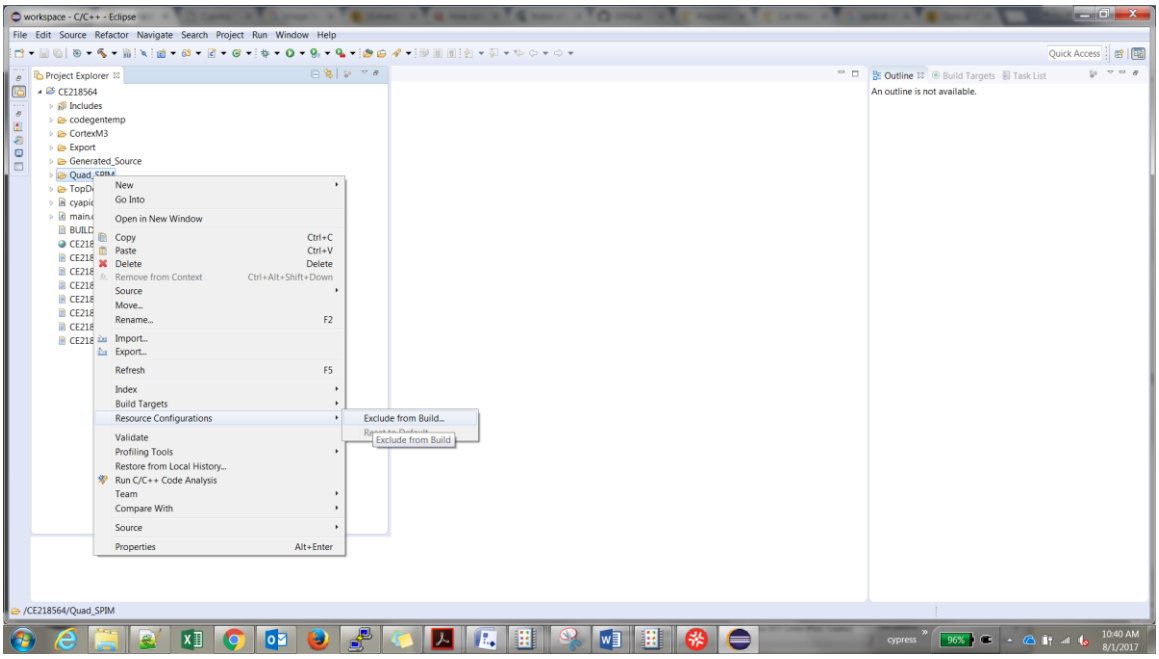


Figure 11. Exclude Custom Components for All Builds

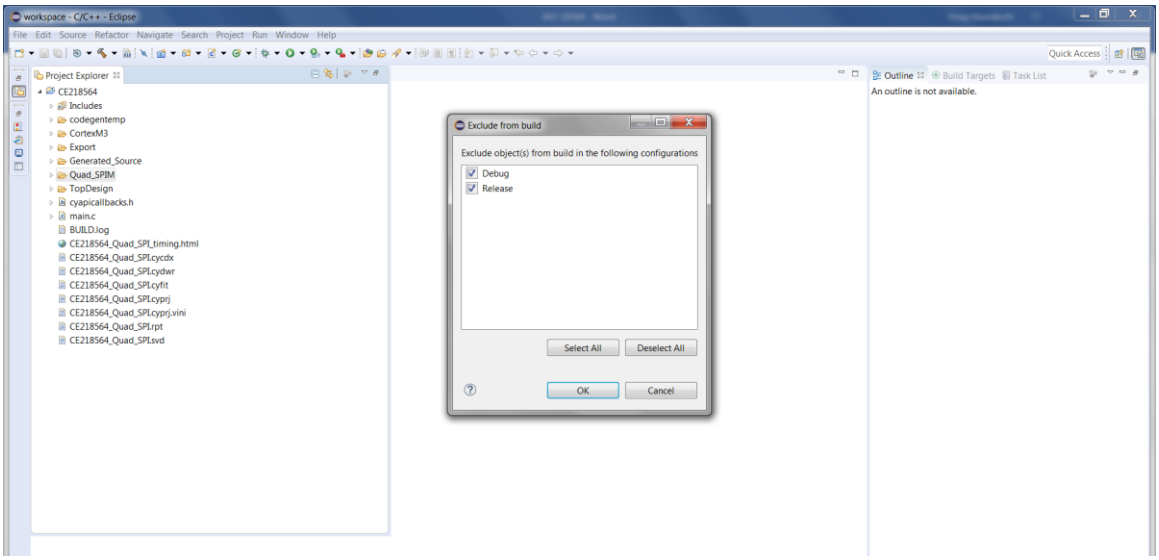
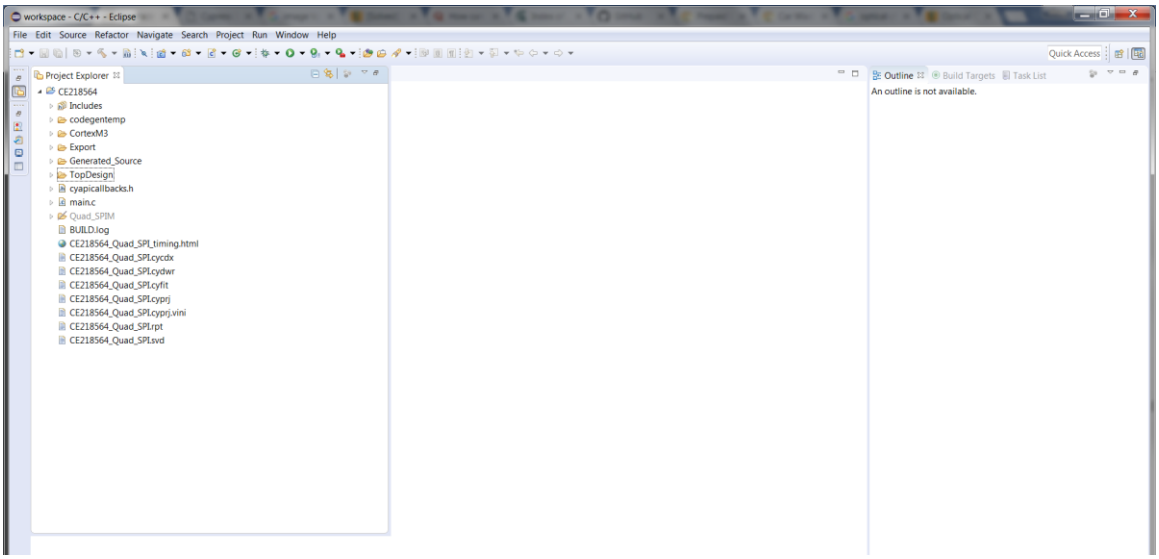


Figure 12. Custom Components Excluded in Eclipse and Build the Project



Code Example

This section provides the sample code to access the Quad_SPI User Component APIs. The complete code can be found in the *main.c* file of the code example. These examples use `Quad_SPIM_1` as the Component instance..

```
// API Quad_SPIM_1_START initializes the Quad SPI user component

      Quad_SPIM_1_START()
```

```
// API Quad_SPIM_1_SPI_WRITE writes "Burst_Length" bytes from array "W_data" to
//memory location "Address 1" in SPI mode

Quad_SPIM_1_SPI_WRITE(Address1, W_data, Burst_Length);
```

```
// API Quad_SPIM_1_QPI_WRITE writes "Burst_Length" bytes from array "W_data" to
//memory location "Address 1" in QPI mode

      Quad_SPIM_1_QPI_WRITE(Address1, W_data, Burst_Length);
```

```
// API Quad_SPIM_1_SPI_READ Reads "Burst_Length" bytes starting from location
//"Address1" in SPI mode

    Quad_SPIM_1_SPI_READ(Address1,R_data,Burst_Length);
```

```
// API Quad_SPIM_1_QPI_READ Reads "Burst_Length" bytes starting from location
//"Address1" in QPI mode

    Quad_SPIM_1_QPI_READ(Address1,R_data,Burst_Length,latency);
```

```
// API Quad_SPIM_1_SPI_Reg_Write Write the Registers in SPI mode. Example 1 gives
the code example for the updating the configuration register 2 (CR2) of S25FL128L
cypress Flash device to enable the QPI mode.

Example 1:

      //Reading Configuration register 2

      Quad_SPIM_1_SPI_Reg_Read(CR2_Read, reg_value+2);


      //Reading Configuration register 1

      Quad_SPIM_1_SPI_Reg_Read(CR1_Read, reg_value+1);


      //Reading status register 1

      Quad_SPIM_1_SPI_Reg_Read(SR1_Read, reg_value);


      //modifying the register content

      *(reg_value + 2) = *(reg_value + 2) | Enable_QPI;


      //Writting registers using the opcode WRR (0x01).

      Quad_SPIM_1_SPI_Reg_WRITE(reg_value,3);
```

```
// API Quad_SPIM_1_QPI_Reg_Write Write the Registers in QPI mode. Example 2 gives
the code example for the updating the configuration register 2 (CR2) of S25FL128L
cypress Flash device to disable the QPI mode.

Example 2:

      //Reading Configuration register 2

      Quad_SPIM_1_QPI_Reg_Read(CR2_Read, reg_value+2);


      //Reading Configuration register 1

      Quad_SPIM_1_QPI_Reg_Read(CR1_Read, reg_value+1);


      //Reading status register 1

      Quad_SPIM_1_QPI_Reg_Read(SR1_Read, reg_value);


      //modifying the register content

      *(reg_value + 2) = *(reg_value + 2) | Enable_QPI;


      //Writting registers using the opcode WRR (0x01).

      Quad_SPIM_1_QPI_Reg_WRITE(reg_value,3);
```

```
// API Quad_SPIM_1_SPI_Reg_Read Read the Registers in SPI mode. Example 3 gives the
code example for the reading the status register 1 (SR1) of S25FL128L cypress Flash
device to check the status of BUSY bit.

Example 3:

//Reading Configuration register 2

Quad_SPIM_1_SPI_Reg_Read(CR2_Read, reg_value+2);
```

```
// API Quad_SPIM_1_QPI_Reg_Read Read the Registers in QPI mode. Example 4 gives the
code example for the reading the status register 1 (SR1) of S25FL128L cypress Flash
device to check the status of BUSY bit.

Example 4:

      //Reading Configuration register 2

      Quad_SPIM_1_QPI_Reg_Read(CR2_Read, reg_value+2);
```

Related Documents

Application Notes		
AN64574	Designing with Serial Peripheral Interface (SPI) nvSRAM	This application note provides a few key design considerations and firmware tips to guide the users designing with SPI nvSRAM.
AN218375	Designing with Cypress Quad SPI (QSPI) F-RAM	This application note provides a few key design considerations and firmware tips to guide the users designing with QSPI F-RAM
Device Documentation		
PSoC 5LP Datasheets	PSoC 5LP Technical Reference Manuals	

Document History

Document Title: CE218564 - Interfacing Quad-SPI Memory with PSoC® 5LP

Document Number: 002-18564

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	5788082	VINI	08/09/2017	Initial release

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709