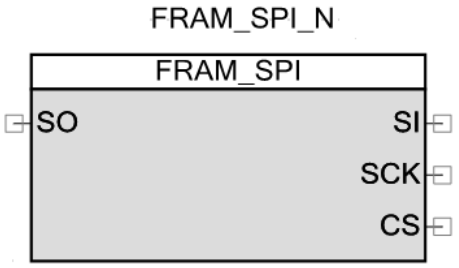
F-RAM SPI PSoC4 Creator Component User Guide



Features

- Mode 0, Mode 3 SPI operating modes
- Supports up to 4 MHz of SPI operation[1]
- F-RAM Read/Write

Notes:

1. This library component is tested up to 4MHz of SPI Clock generation. This can go up the maximum frequency supported by PSoC4 SPI component.

General Description

The FRAM_SPI component provides a PSoC4 SPI Master interface to SPI F-RAM devices. This component can be directly used in user's PSoC4 designs. High level API's are provided for different F-RAM operations. User can directly use these API's as it reduces the time to integrate F-RAM in the user application.

Input/output Connections

This section describes the various input and output connections for the FRAM_SPI component.

SO – Input

The SO input carries the master input (PSoC4) – slave output serial data from the F-RAM device.

SI – Output

The SI output carries the master output – slave (F-RAM) input serial data from the master device (PSoC4).

SCK– Output

The SCK output carries the master (PSoC4) synchronization clock output to the F-RAM device.

CS – Output

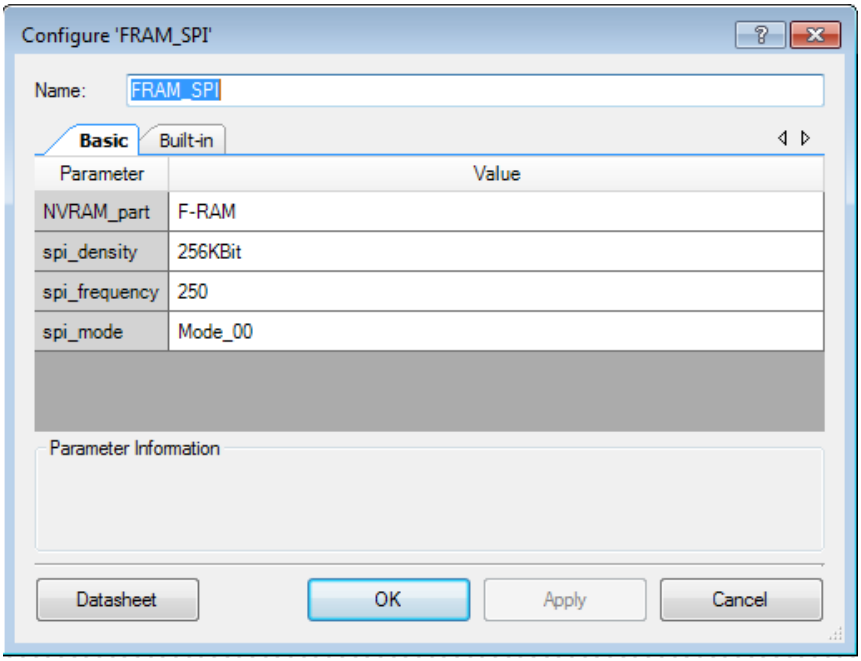
The CS output carries the chip select signal to the F-RAM device.

Parameters

Drag an FRAM_SPI component onto your design and double-click on it to open the Configure dialog.

Basic Tab

This tab is used to set the F-RAM part, density, SPI frequency and mode of operation. Eg: F-RAM part is selected by configuring FRAM_part to F-RAM, 256 KBit density, 250 for 250 KHz and Mode 0.



Hardware Configuration

FRAM_SPI component has following default hardware configuration.

Shift Direction: MSB first

Data Bits: 8

Data Lines: MOSI + MISO

Clock for SPI block: External

To change the above configuration user has to import the component instead of using ‘dependencies’.

Application Programming Interface

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 "FRAM_SPI" 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 "FRAM_SPI".

Function	Description
FRAM_SPI_Init	Initialization routine for FRAM_SPI component
FRAM_SPI_Write	SPI F-RAM SRAM Write
FRAM_SPI_Read	SPI F-RAM SRAM Read
FRAM_SPI_BurstWrite	SPI F-RAM SRAM Burst Write
FRAM_SPI_BurstRead	SPI F-RAM SRAM Burst Read
FRAM_SPI_Status_Reg_Write	SPI F-RAM Status Register Write
FRAM_SPI_Status_Reg_Read	SPI F-RAM Status Register Read

void FRAM_SPI_Init(void)

Description: Initialization routine for FRAM_SPI component. Initializes the SPI block, CS and Hold pin.

Parameters: None

Return Value: None

Side Effects: None

uint8 FRAM_SPI_Write(uint32 addr, uint8 data_write)

Description: Write a byte of data into F-RAM.

Parameters: uint32 addr: 32 bit FRAM address for write.
uint8 data_write: Data byte to be written.

Return Value: uint8: 0 – Communication Error
1 – Success

Side Effects: None

uint8 FRAM_SPI_Read(uint32 addr, uint8 *data_read)

Description: Read a data byte from F-RAM.

Parameters: uint32 addr : 32 bit F-RAM address for read.
uint8 *data_read. : 8 bit pointer variable to hold data

Return Value: uint8: 0 – Communication Error
1 – Success

Side Effects: None

uint8 FRAM_SPI_BurstWrite(uint32 addr, uint8 *data_write_ptr, uint32 total_data_count)

Description: Write total_data_count number of data into FRAM.

Parameters: uint32 addr: 32 bit FRAM address for write.
uint8 *data_write_ptr: Pointer to an array of data bytes to be written.
uint32 total_data_count: Number of data bytes to be written.

Return Value: uint8: 0 – Communication Error
1 – Success

Side Effects: None

uint8 FRAM_SPI_BurstRead(uint32 addr, uint8 *data_read_ptr, uint32 total_data_count)

Description: Read total_data_count number of data from FRAM.

Parameters: uint32 addr : 32 bit FRAM address for read.
uint8 *data_read_ptr: Pointer to an array for storing data bytes.
uint32 total_data_count: Number of data bytes to be read.

Return Value: uint8: 0 – Communication Error
1 – Success

Side Effects: None

uint8 FRAM_SPI_ Status_Reg_Write (uint8 data_byte)

Description: FRAM Status Register Write

Parameters: uint8 data_byte -> 1 byte Status register data to be written

Return Value: uint8: 0 – Communication Error
1 – Success

Side Effects: None

uint8 FRAM_SPI_ Status_Reg_Read (uint8 * status)

Description: FRAM Status Register Read

Parameters: uint8 * status : pointer to hold status register value

Return Value: uint8: 0 – Communication Error
1 – Success

Side Effects: None

Defines

All the FRAM definitions are defined in FRAM_SPI_definition.h

```
// FRAM Commands
#define FRAM_WREN             ((uint8)0x06)   // Write Enable
#define FRAM_WRDI             ((uint8)0x04)   // Write Disable
#define FRAM_SRAM_WRITE_CMD   ((uint8)0x02)   // SRAM Write
#define FRAM_SRAM_READ_CMD    ((uint8)0x03)   // SRAM Read
#define FRAM_RDSR_CMD         ((uint8)0x05)   // Read Status Register
#define FRAM_WRSR_CMD         ((uint8)0x01)   // Write Status Register
```

Setup

FRAM_SPI archive contains both the example project and FRAM_SPI component. Unzip the archive FRAM_SPI. Following snapshots show how to use FRAM_SPI component in your design.

1. **Open PSoC Creator and open your design (workspace). In the below figure, new project 'Design01' is created.**

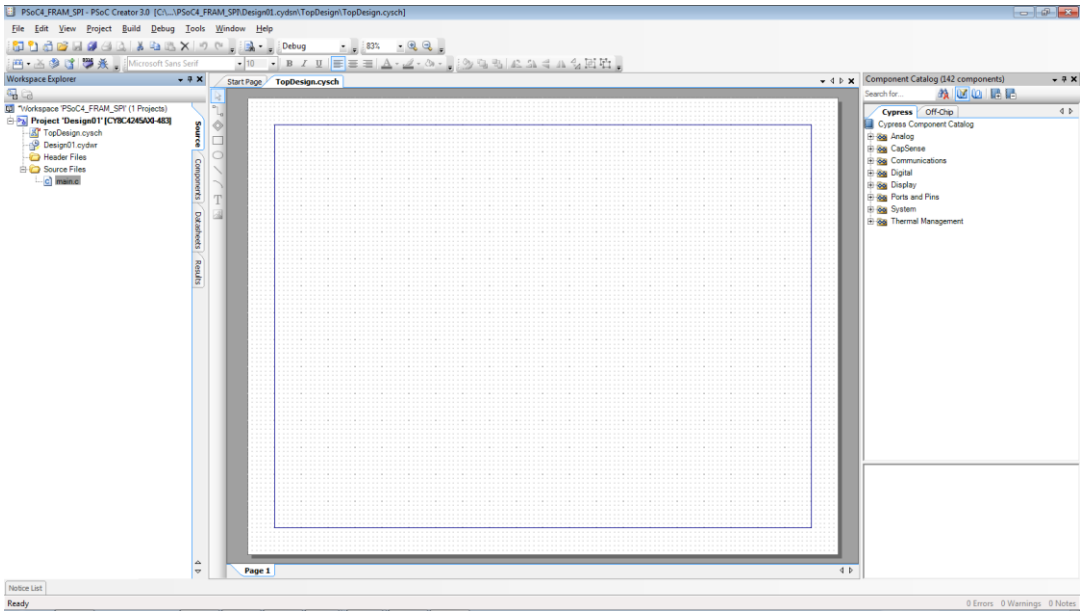


Fig 1: Create project 'Design01'

2. **Select 'Dependencies' tab on the workspace explorer and bring FRAM_SPI component into your design as shown below**.

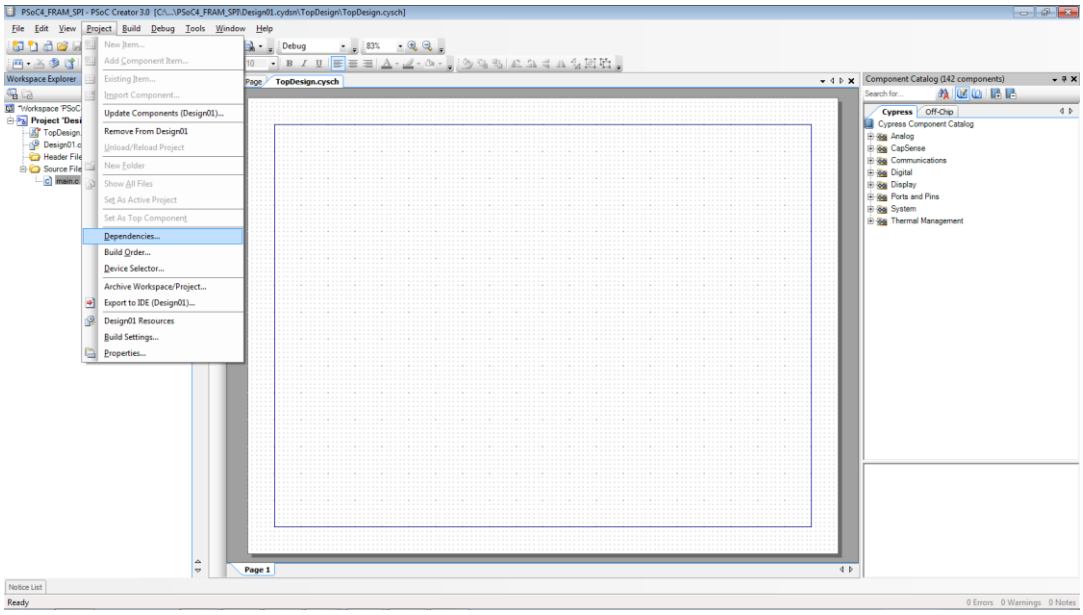


Fig 2: Right click on the project and Open Dependencies

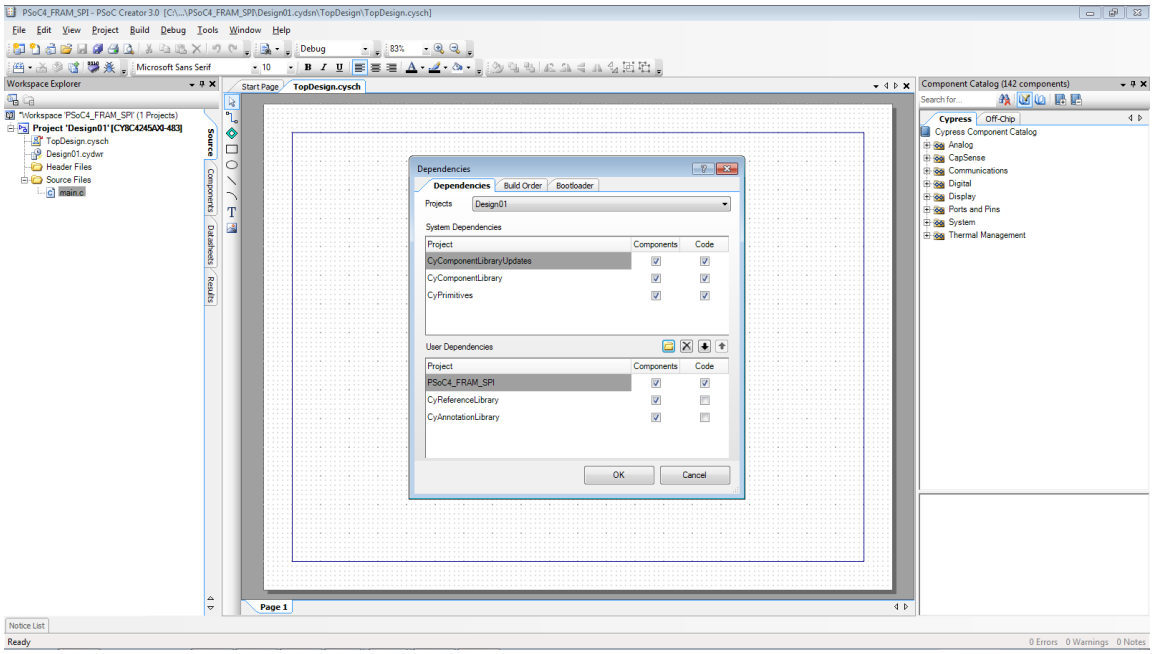


Fig 3: Click on New Entry (User Dependencies) and Select 'PSoC4_FRAM_SPI.cyprj' from PSoC4_FRAM_SPI.cydsn folder.

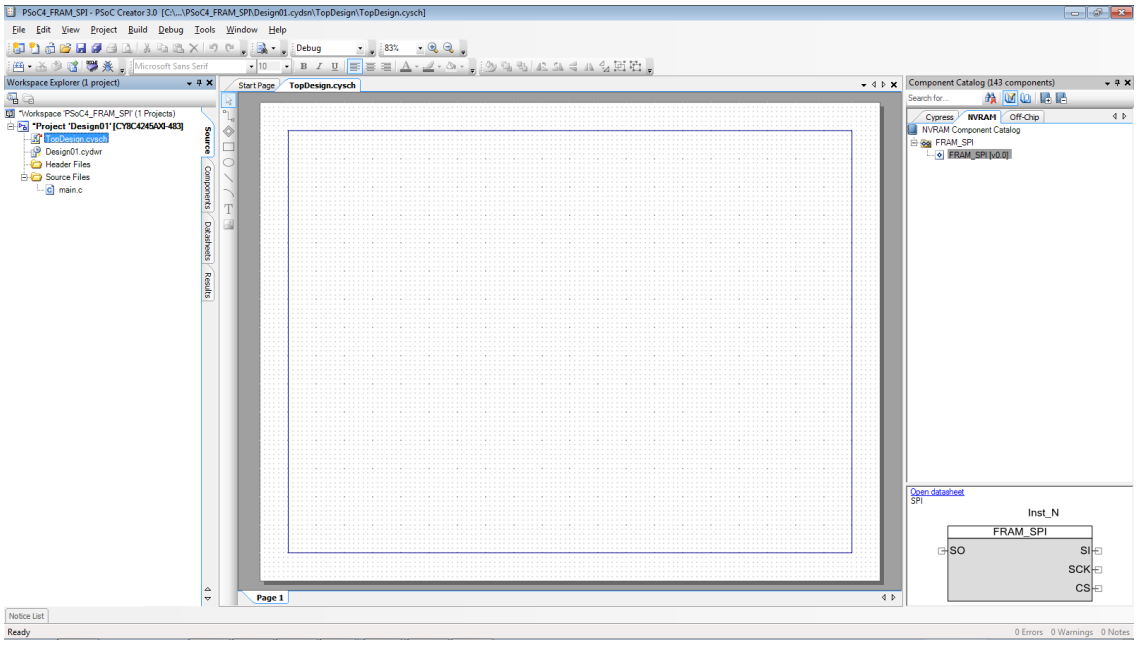


Fig 4: FRAM_SPI component appears under FRAM/FRAM_SPI in component Catalog

3. Add FRAM_SPI component to your design as follows

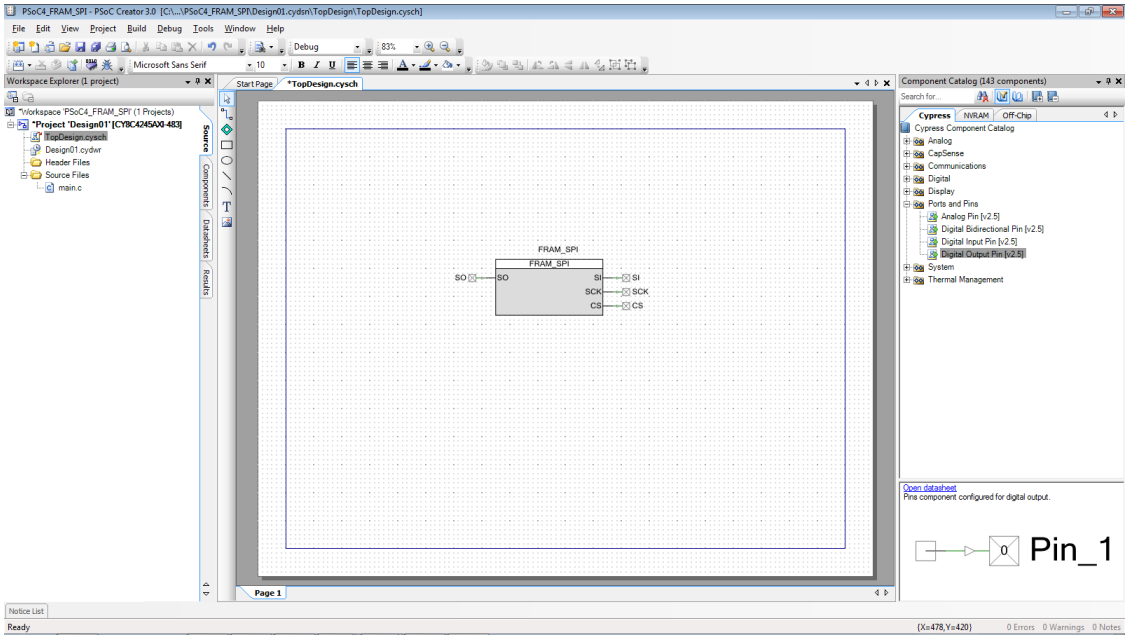


Fig 5: Drag and drop FRAM_SPI component onto TopDesign.cysch and assign Digital IOs.

4. FRAM_SPI Configuration

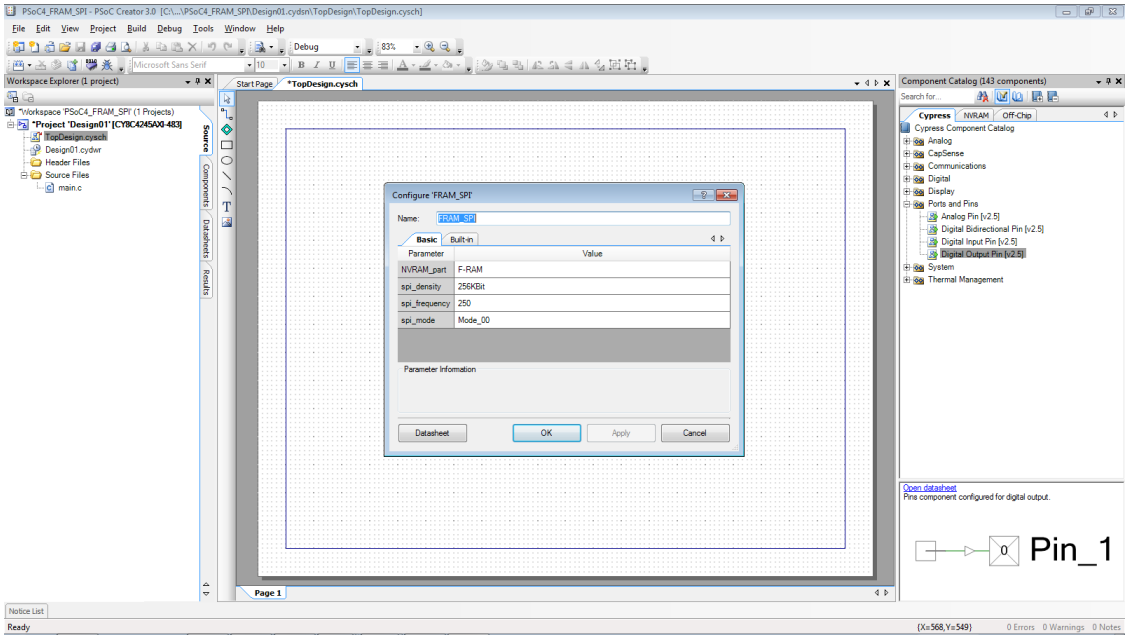


Fig 6: Right click on the FRAM_SPI component in TopDesign.cysch and select configure. Set parameters FRAM_part, spi_density, spi_frequency and spi_mode as per your design. In this project we have selected 256Kbit F-RAM part, 250 KHz SPI frequency and mode 0.

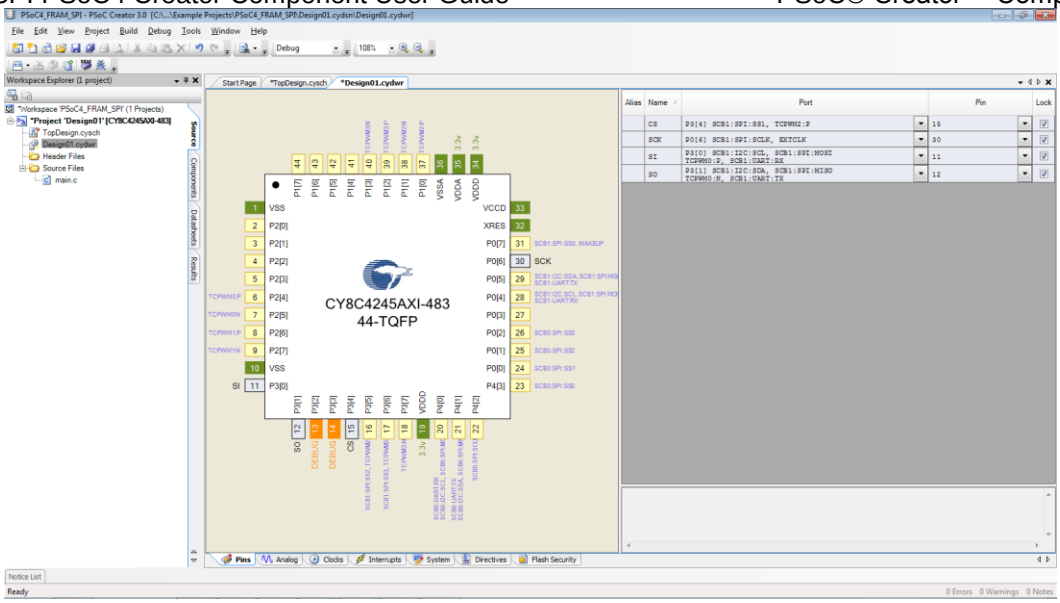


Fig 7: Assign appropriate input/output pins as per your design and build the project.

Sample Firmware Source Code

Below is a sample code taken from the example projects. Component instance is FRAM_SPI. The example project can be opened by selecting PSoC4_FRAM_SPI workspace in PSoC4_FRAM_SPI folder.

```
#include <device.h>
#include "FRAM_SPI.h"

// Size of data buffer
#define BUFFER_SIZE             (16u)

// Example F-RAM Address
#define EXAMPLE_ADDR_1          (0x2000)

// Example F-RAM Address
#define EXAMPLE_ADDR_2          (0x3456)

// Example F-RAM Data
#define EXAMPLE_DATA_BYTE       (0xa5)

// Example F-RAM Status Register value
#define EXAMPLE_STS_REG_VALUE   (0x08)

// Data Copmarison Result macro
#define FAIL ((uint8)0u)
#define PASS ((uint8)1u)

/*******************************************************************************
*  The main function calls F-RAM Read/Write APIs
*  1. Initialize data array
*  2. Init F-RAM SPI Block
*  3. Start-up Delay
*  4. Write EXAMPLE_DATA_BYTE to EXAMPLE_ADDR_1 address
```

```
*  5. Read a byte from address EXAMPLE_ADDR_1 and store in fram_rd_byte
*  6. Write 16 bytes of data from data_bytes array to EXAMPLE_ADDR_2 address
*  7. Read 16 bytes of data from EXAMPLE_ADDR_2 address
*  8. Write EXAMPLE_STS_REG_VALUE to status register
*  9. Read status register and store in 'statug_reg' variable
* 10. Compare status register value and fram_rd_byte. Glow BLUE LED if success
* 11. Clear status register
*******************************************************************************/
int main()
{
    uint8 data_bytes[BUFFER_SIZE];
      uint8 fram_rd_data[BUFFER_SIZE];
    uint8 fram_rd_byte;
    uint8 status_reg = 0;
      uint8 i;
    uint8 status;
    uint8 result;

    // Enable global interrupts.
    CyGlobalIntEnable;

    // Turn OFF Blue LED
    BLUE_LED_Write(1);

    // Initialization of variables
      for(i = 0; i < BUFFER_SIZE; i++)
      {
         fram_rd_data[i] = 0;
         data_bytes[i] = i+1;
      }

    // FRAM component initialization
    FRAM_SPI_Init();

    // Start-up Delay
    CyDelay(100);

    // FRAM Write : Write 1 byte to FRAM address 0x2000
    status = FRAM_SPI_Write(EXAMPLE_ADDR_1, EXAMPLE_DATA_BYTE);

    // FRAM Read : Write 4 bytes from FRAM address 0x3456 to array rd_data
      status = FRAM_SPI_Read(EXAMPLE_ADDR_1, &fram_rd_byte);

    // FRAM Write : Write 4 bytes from array data_bytes to FRAM address 0x3456
    status = FRAM_SPI_BurstWrite(EXAMPLE_ADDR_2, data_bytes, BUFFER_SIZE);

    // FRAM Read : Write 4 bytes from FRAM address 0x3456 to array rd_data
      status = FRAM_SPI_BurstRead(EXAMPLE_ADDR_2,fram_rd_data,BUFFER_SIZE);

    // FRAM Status Register Write : Write 0x08 to status register
      status = FRAM_SPI_Status_Reg_Write(EXAMPLE_STS_REG_VALUE);

    // FRAM Status Register Read : Read FRAM Status Register
    status = FRAM_SPI_Status_Reg_Read(&status_reg);

    // FRAM Status Register Write : Write 0x00 to status register
      status = FRAM_SPI_Status_Reg_Write(0x00);
```

```
    result = PASS;  // Initialize result to PASS

    // Compare the status resiter and single byte read data
    if((status_reg != EXAMPLE_STS_REG_VALUE) || (fram_rd_byte != EXAMPLE_DATA_BYTE) ||
    (status == SPI_COM_ERROR))
    {
        result = FAIL;  // F-RAM Status or memory Write/Read failed
    }

    // Compare the burst read data with written data
    for(i=0;i<BUFFER_SIZE;i++)
    {
        if(fram_rd_data[i] != data_bytes[i])
        {
            result = FAIL;  // F-RAM Burst Write/Read failed
            break;
        }
    }

    // If PASS, turn ON PSoC4 BLUE LED
    if(result == PASS)
        BLUE_LED_Write(0);

    // loop forever
      for(;;){}

}
```

Component Changes

This section lists the major changes in the component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
1.0	F-RAM User Module for interfacing F-RAM with PSoC4	Initial Version

© Cypress Semiconductor Corporation, 2015. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

This Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.