

F-RAM I²C PSoC4 Creator Component User Guide

Features

- Supports 100 Kbps, 400 Kbps, 1 Mbps I²C operation^[1]
- F-RAM Read/Write

Notes:

1. These correspond to the speeds supported by PSoC4 I²C component.

FRAM_I2C
FRAM_I2C

General Description

The FRAM_I2C component provides a PSoC4 I²C Single Master interface to I²C 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 reduce the time to integrate the F-RAM in their application.

Input/output Connections

This section describes the various input and output connections for the F-RAM I²C component.

SDA – Input/output

The SDA carries data between Master (PSoC4) and the Slave (F-RAM). This is a bidirectional pin.

SCL – Input/output

The SCL is the clock output from the Master (PSoC4) to Slave (F-RAM). This is also a bidirectional pin.

The SDA and SCL lines appear under pins in the .cydwr file.

Application Programming Interface

Application Programming Interface (API) routines allow the user to configure the component using software. The table below lists and describes each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name "FRAM_I2C_1" to the first instance of FRAM_I2C component in a given design. User can rename the instance to any unique name that follows the syntactic rules of 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_I2C".

Function	Description
FRAM_I2C_Init	Initialization routine for FRAM_I2C component
FRAM_I2C_Write	I ² C F-RAM Write
FRAM_I2C_Current_Read	I ² C F-RAM Current Read
FRAM_I2C_Random_Read	I ² C F-RAM Random Read

void FRAM_I2C_Init (void)

Description: Initialization routine for FRAM_I2C component. Initializes the PSoC4 I²C block.

Parameters: None

Return Value: None

Side Effects: None

uint32 FRAM_I2C_Write (uint8 slave_id, uint32 addr, uint8 *data_write_ptr, uint32 total_data_count)

Description: Write total_data_count number of data bytes into F-RAM specified by slave_id.

Parameters: uint8 : 7 bit Slave ID
 uint32 addr: 32 bit F-RAM 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: uint32 : Error Status

FRAM_I2C_MSTR_NO_ERROR	-- (0x00u)
FRAM_I2C_MSTR_ERR_ARB_LOST	-- (0x01u)
FRAM_I2C_MSTR_ERR_LB_NAK	-- (0x02u)
FRAM_I2C_MSTR_NOT_READY	-- (0x04u)
FRAM_I2C_MSTR_BUS_BUSY	-- (0x08u)
FRAM_I2C_MSTR_ERR_ABORT_START	-- (0x10u)
FRAM_I2C_MSTR_ERR_BUS_ERR	-- (0x100u)
FRAM_I2C_MSTR_ERR	-- (0xFFu)

Side Effects: None

uint32 FRAM_I2C_Current_Read (uint8 slave_id, uint8 *data_read_ptr, uint32 total_data_count)

Description: Read total_data_count number of data bytes from the current address of the F-RAM specified by slave_id.

Parameters: uint8 : 7 bit Slave ID
 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: uint32 : Error Status

FRAM_I2C_MSTR_NO_ERROR	-- (0x00u)
FRAM_I2C_MSTR_ERR_ARB_LOST	-- (0x01u)
FRAM_I2C_MSTR_ERR_LB_NAK	-- (0x02u)
FRAM_I2C_MSTR_NOT_READY	-- (0x04u)
FRAM_I2C_MSTR_BUS_BUSY	-- (0x08u)
FRAM_I2C_MSTR_ERR_ABORT_START	-- (0x10u)
FRAM_I2C_MSTR_ERR_BUS_ERR	-- (0x100u)
FRAM_I2C_MSTR_ERR	-- (0xFFu)

Side Effects: None

uint32 FRAM_I2C_Random_Read (uint8 slave_id, uint32 addr, uint8 *data_read_ptr, uint32 total_data_count)

Description: Read total_data_count number of data bytes from the F-RAM specified by slave_id.

Parameters: uint8 : 7 bit Slave ID
 uint32 addr: 32 bit F-RAM 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: uint32 : Error Status

FRAM_I2C_MSTR_NO_ERROR	-- (0x00u)
FRAM_I2C_MSTR_ERR_ARB_LOST	-- (0x01u)
FRAM_I2C_MSTR_ERR_LB_NAK	-- (0x02u)
FRAM_I2C_MSTR_NOT_READY	-- (0x04u)
FRAM_I2C_MSTR_BUS_BUSY	-- (0x08u)
FRAM_I2C_MSTR_ERR_ABORT_START	-- (0x10u)
FRAM_I2C_MSTR_ERR_BUS_ERR	-- (0x100u)
FRAM_I2C_MSTR_ERR	-- (0xFFu)

Side Effects: None

Slave ID

For F-RAM devices, the 4 most significant bits [7:4] of the slave ID are defined for accessing SRAM. Last 3 bits are decided by A2, A1, A0 pin states. Some devices (Example: I²C 1-MBit) do not define A0 pin in which case it becomes don't care.

Let's assume all 3 pins are connected to ground (A2 = 0, A1 = 0, A0 = 0).

To access F-RAM SRAM, 7 bit slave id = 1010 000 = 'h50

For 1-MBit F-RAM devices, A0 becomes MA16 (MSB address bit A16). Setting of MA16 bit in slave id is taken care inside the APIs. User setting of this bit is ignored and hence this bit can be set as 0 in slave ID.

Setup

The associated project archive contains both the example project and FRAM_I2C component. Unzip the archive. Following snapshots show how to use FRAM_I2C component in your design.

1. Open PSoC Creator and open your design (workspace). In the below figure, new PSoC4 project 'Design01' is created.

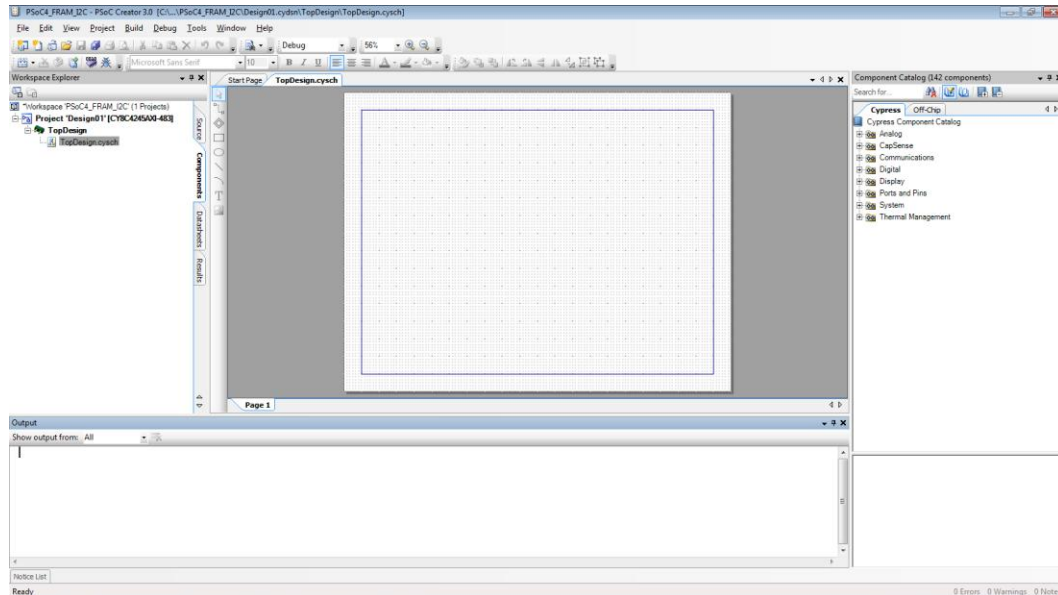


Figure 1: Create project 'Design01'

2. Select 'Components' tab on the workspace explorer and select the project as shown below.

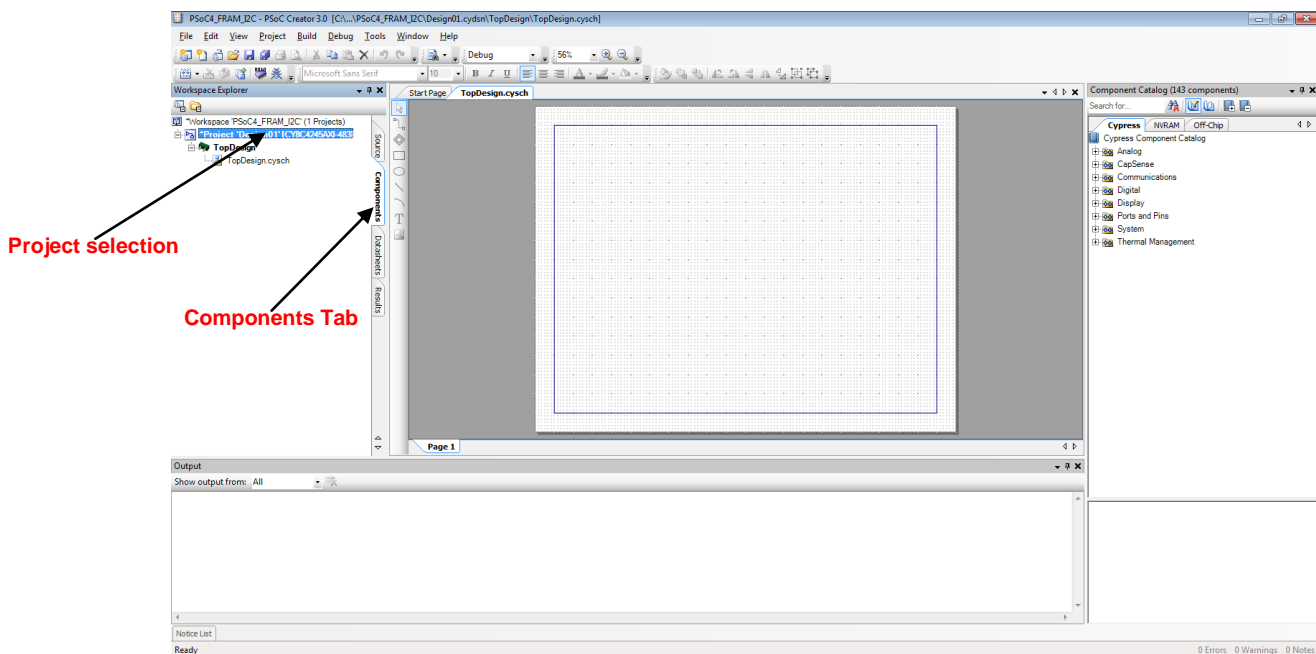


Figure 2: Select 'Components' tab

Right click on the project and select 'import component'. Select the "Import from project/library" as "PSoC4_FRAM_I2C" with the development kit and Source Component as "FRAM_I2C". Target project is the user project "Design01"

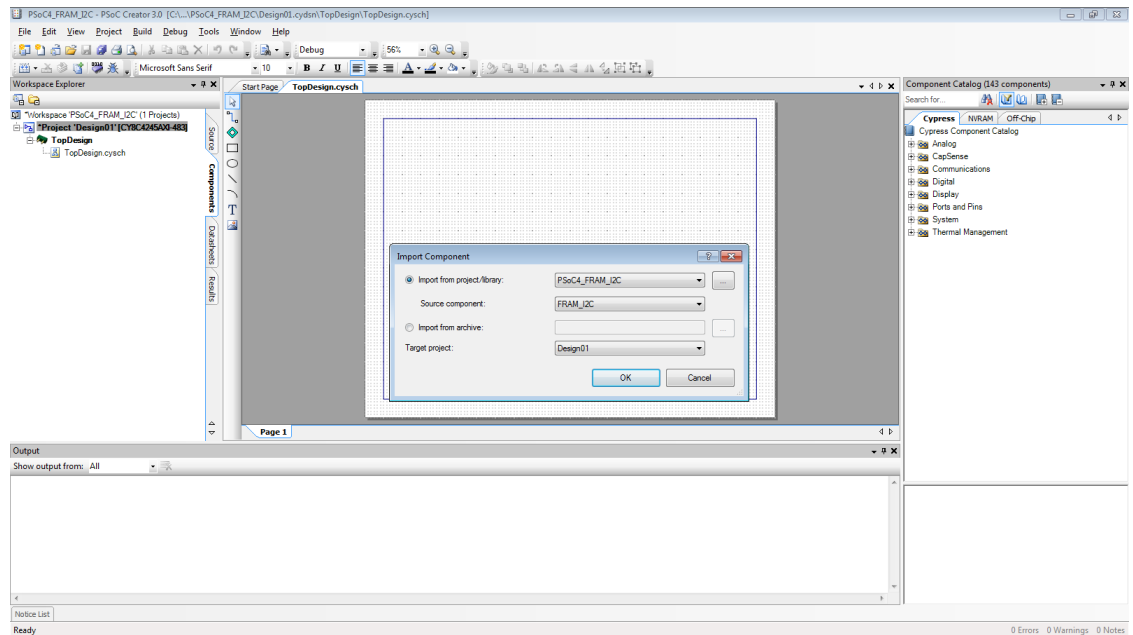


Figure 3: Import FRAM_I2C component.

After importing the component, a new component FRAM_I2C appears under a new tab named NVRAM in the component catalog library as shown below.

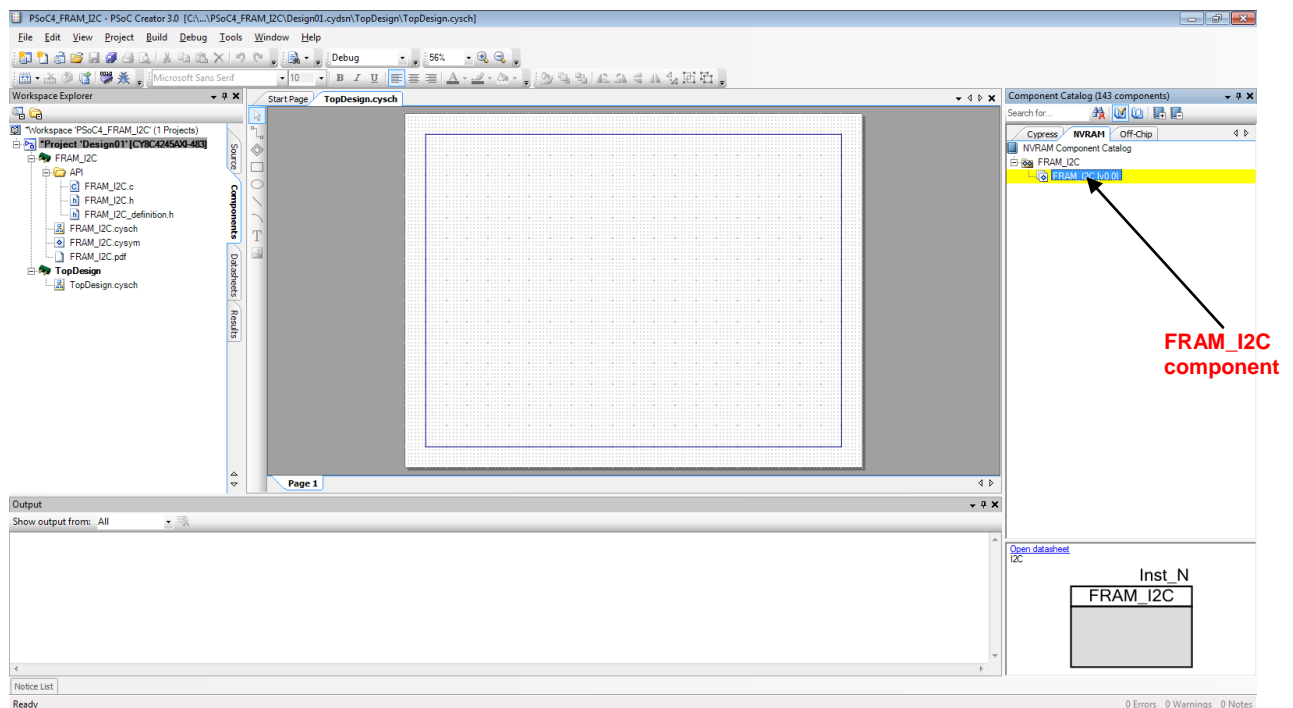


Figure 4: FRAM_I2C component appears under FRAM_I2C in the component Catalog

3. Add FRAM_I2C component to your design as follows and rename it to FRAM_I2C

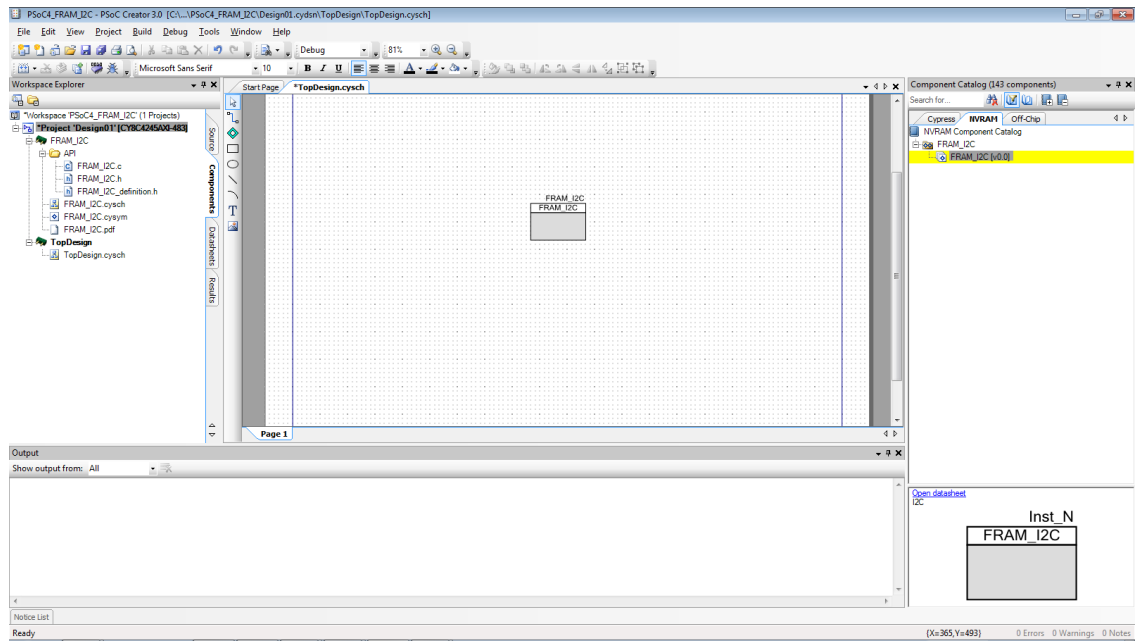


Figure 5: Drag and drop FRAM_I2C component onto TopDesign.cysch.

4. FRAM_I2C Configuration

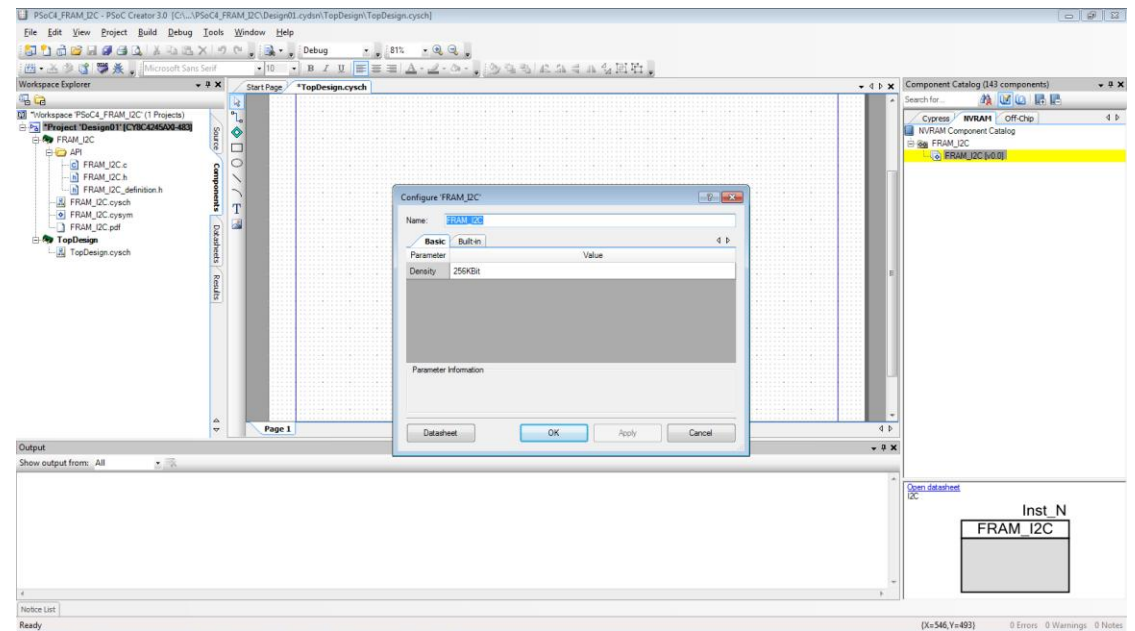


Figure 6: Right Click on the FRAM_I2C component and select configure.

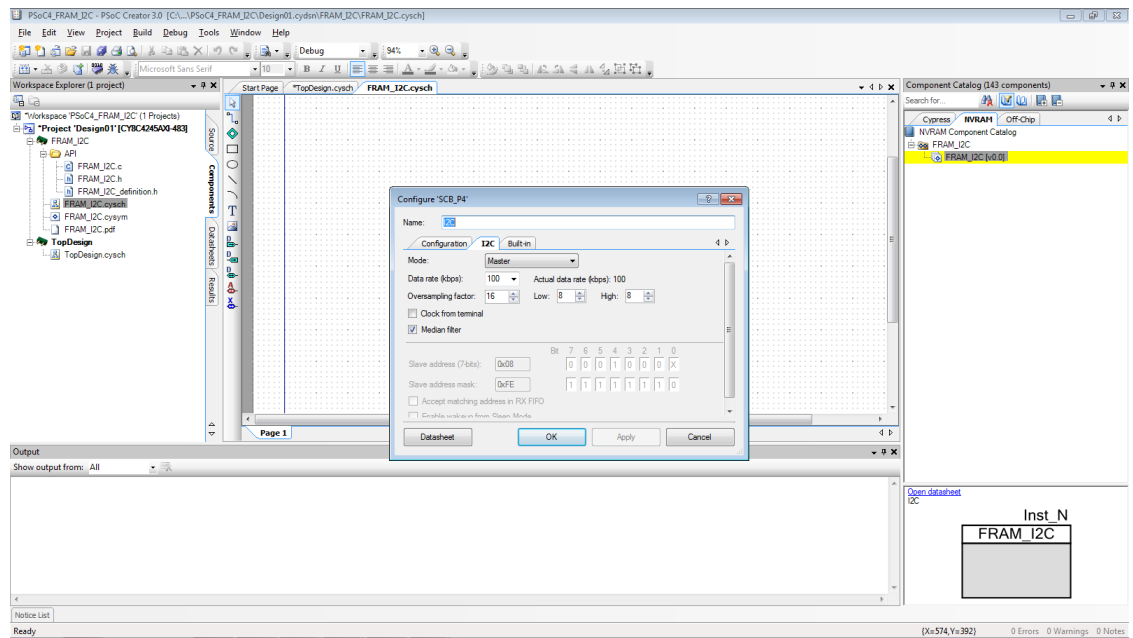


Figure 7: Click on ‘Components’, select “FRAM_I2C.cysch”. Right click on I²C block, select ‘configure’ and change the data rate accordingly. This data rate will set the I²C access speed.

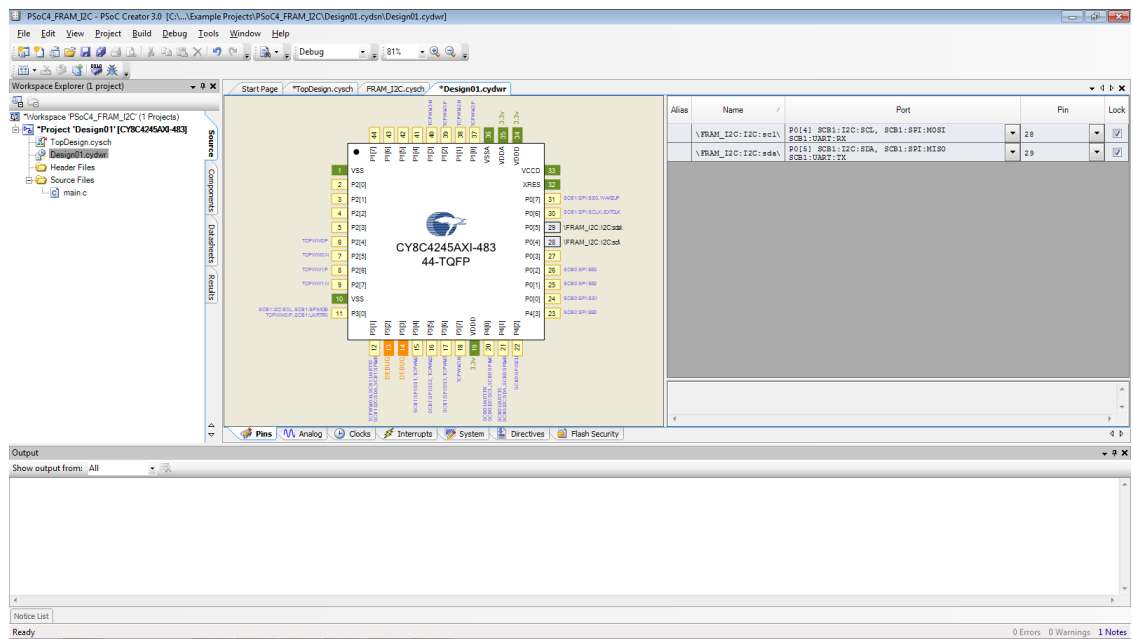


Figure 8: Assign appropriate input/output pins as per your design and build the project.

Sample Firmware Source Code

Below is the sample code for different F-RAM operations taken from the example project. Component instance is FRAM_I2C. The example project can be opened by selecting FRAM_I2C workspace. In this example project, A2 = 0, A1 = 0, A0 = 0.

```
#include <device.h>
#include "FRAM_I2C.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)

// Macro for Communication Error
#define COM_ERR()    for(;;){}

// Data Comparison 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 I2C 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 15 bytes of data from EXAMPLE_ADDR_2 address through random read
function
*   8. Read 16th byte through current read function
*   9. Compare the read data and glow BLUE LED if read is PASS
*****/

int main()
{
    uint8 data_bytes[BUFFER_SIZE];
    uint8 fram_rd_data[BUFFER_SIZE];
    uint8 fram_wr_byte;
    uint8 fram_rd_byte, fram_rd_current_byte;
    uint8 i;
    uint8 result;

    // Enable global interrupts.
    CyGlobalIntEnable;
```

```

// Turn OFF Blue LED
BLUE_LED_Write(1);

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

// Initialize I2C F-RAM Component
FRAM_I2C_Init();

// Start-up delay
CyDelay(100);

// F-RAM Write : Write 1 byte of data to F-RAM location EXAMPLE_ADDR_1
fram_wr_byte = EXAMPLE_DATA_BYTE;
if(FRAM_I2C_Write(FRAM_SLAVE_SRAM_ADDR, EXAMPLE_ADDR_1, &fram_wr_byte, 1)
!= FRAM_I2C_MSTR_NO_ERROR)
{
    // I2C Error. loop forever
    COM_ERR();
}

// F-RAM Read : Read 1 byte of data from F-RAM location EXAMPLE_ADDR_1
if(FRAM_I2C_Random_Read(FRAM_SLAVE_SRAM_ADDR,
EXAMPLE_ADDR_1,&fram_rd_byte,1) != FRAM_I2C_MSTR_NO_ERROR)
{
    // I2C Error. loop forever
    COM_ERR();
}

// F-RAM Write : Write 16 bytes from array data_bytes to F-RAM location
EXAMPLE_ADDR_2
if(FRAM_I2C_Write(FRAM_SLAVE_SRAM_ADDR, EXAMPLE_ADDR_2, data_bytes,
BUFFER_SIZE) != FRAM_I2C_MSTR_NO_ERROR)
{
    // I2C Error. loop forever
    COM_ERR();
}

// F-RAM Random Read : Read 15 bytes of data from SRAM starting from
address EXAMPLE_ADDR_2. //
// Data will be stored into fram_rd_data array.
//
if(FRAM_I2C_Random_Read(FRAM_SLAVE_SRAM_ADDR,
EXAMPLE_ADDR_2,fram_rd_data,BUFFER_SIZE-1) != FRAM_I2C_MSTR_NO_ERROR)
{
    // I2C Error. loop forever
    COM_ERR();
}

// F-RAM Read : Read the 16th byte through current location read
if(FRAM_I2C_Current_Read(FRAM_SLAVE_SRAM_ADDR, &fram_rd_current_byte,1)
!= FRAM_I2C_MSTR_NO_ERROR)

```

```
{  
    // I2C Error. loop forever  
    COM_ERR();  
}  
  
result = PASS; // Initialize result to PASS  
  
// Compare the single byte write and read data, current read data  
if((fram_rd_byte != fram_wr_byte) || (fram_rd_current_byte !=  
data_bytes[BUFFER_SIZE-1]))  
{  
    result = FAIL; // F-RAM Status or memory Write/Read failed  
}  
  
// Compare the burst read data with written data  
for(i=0;i<(BUFFER_SIZE-1);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	Created APIs for interfacing I ² C 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.