Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

CE220573 – Interfacing I²C NVSRAM and F-RAM with PSoC 3/5LP

Objective

This example demonstrates a data logging application using Cypress' I²C memories.

Overview

This code example shows a data logging application using Cypress' I²C memories (F-RAM™ and NVSRAM). The code example has a User Component I2C_RAM, designed specifically for Cypress' I²C memories. The User Component is configurable for different frequencies and density. The User Component is imported into the code example. Table 2 lists the usage of the supported APIs.

Requirements

Tool: PSoC Creator™ 4.2

Programming Language: C (GCC 5.4), Arm® Cortex®-M3 Assembler, Keil 9.51

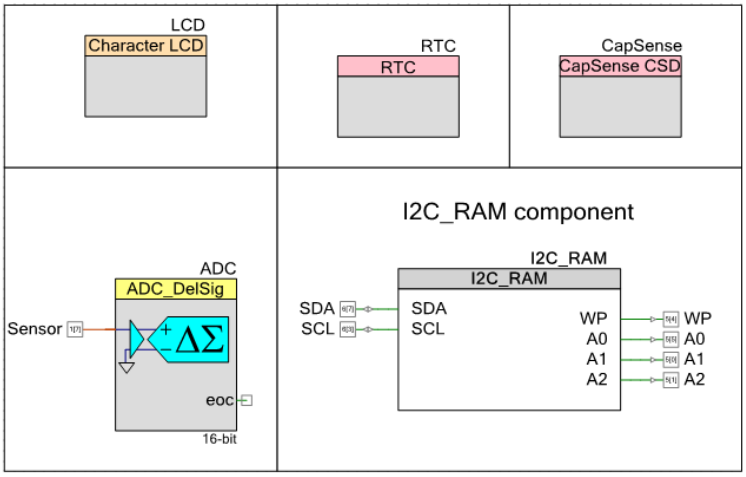
Associated Parts: All PSoC® 5LP and PSoC 3 parts

Related Hardware: CY8CKIT-001

Design

This code example implements a data logging application using Cypress' I²C memories. The User Component (I2C_RAM) which is a part of this code example has built-in APIs to perform memory, and register Write/Reads over the I²C bus. The code example simulates a sensor output by utilizing the on-board variable resister (VR) as shown in Figure 4. The code example samples VR output at an interval of 1 second and writes the sampled data into memory using the I2C_RAM Component. If you do not stop manually, the example will continue to sample for 59 seconds. You can interrupt the sampling within the 59-second window by using the CapSense button on the kit. Once the sampling is complete, the example will perform I2C read on memory to display the content on the LCD.

Figure 1. Data Logging Application Design Schematics



Design Considerations

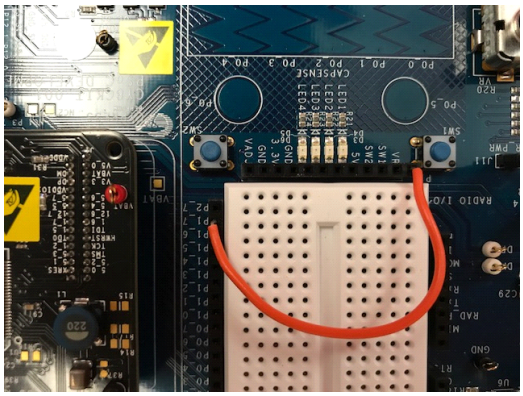
The maximum possible serial clock frequency is 1 MHz.

Hardware Setup

Mount a daughter board with memory on CY8CKIT-001. Modify the pin out configuration for PSoC 3/5LP to match the daughter board.

VR is used as a sensor. The analog data from sensor is converted to digital using PSoC 3/5LP 10-bit ADC and logged into I²C F-RAM. Use a wire to connect the VR (variable resistance terminal) to a general-purpose I/O (GPIO) [**P1.7**] of MCU as shown in Figure 2.

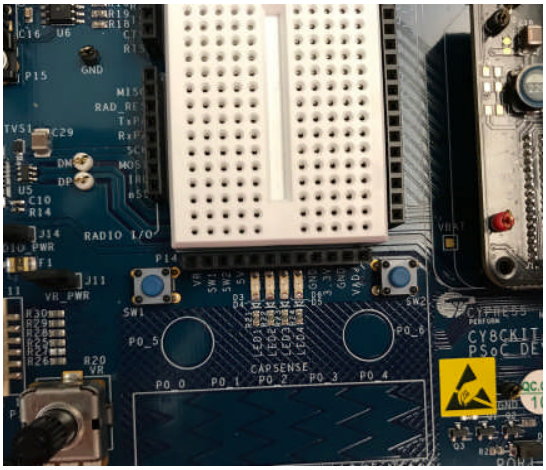
Figure 2. Connecting VR to the GPIO P1.7



Working

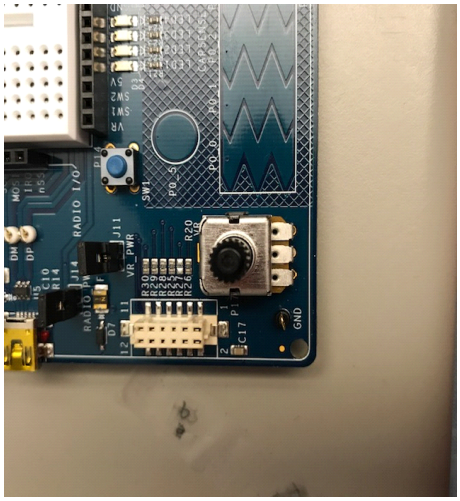
1. Build the project and program PSoC 5LP.
2. Press the CapSense switch [**P0_6]** to initiate sampling. Char LCD displays the sensor data and time in seconds. Figure 3 shows the snapshot of LCD while sampling the ADC output.

Figure 3. CapSense [P0_5] and [P0_6]



3. Rotate the variable resistance R20 (Figure 4) to change the sensor value and note down the value and corresponding time.

Figure 4. Variable Resistor as Sensor (VR)



4. Sampling stops automatically when time reaches 59 seconds. Press the CapSense switch [**P0_5**], in between, to stop the sampling.
5. Press the CapSense switch [**P0_5**] to start reading the sampled data from memory and display it on the LCD.
6. The code example will read all sampled data and show the summary of sampling by calculating the maximum, minimum, and average values.
7. Press **Reset** to restart the code example.

Components / User Modules

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

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

Component or User Module	Version	Placement/Hardware Resources
Character LCD	2.20	1
RTC	2.0	1
CapSense	3.50	1
ADC	3.20	1
I2C	3.50	1
Control_reg	1.80	1

Parameter Settings

Double-click the I2C_RAM Component to configure the parameter as shown in Figure 5.

Figure 6 shows the I2C Component parameter. Figure 6 and Figure 7 show the CapSense parameters and Figure 8 shows the ADC parameters.

Figure 5. I2C User Module Configuration

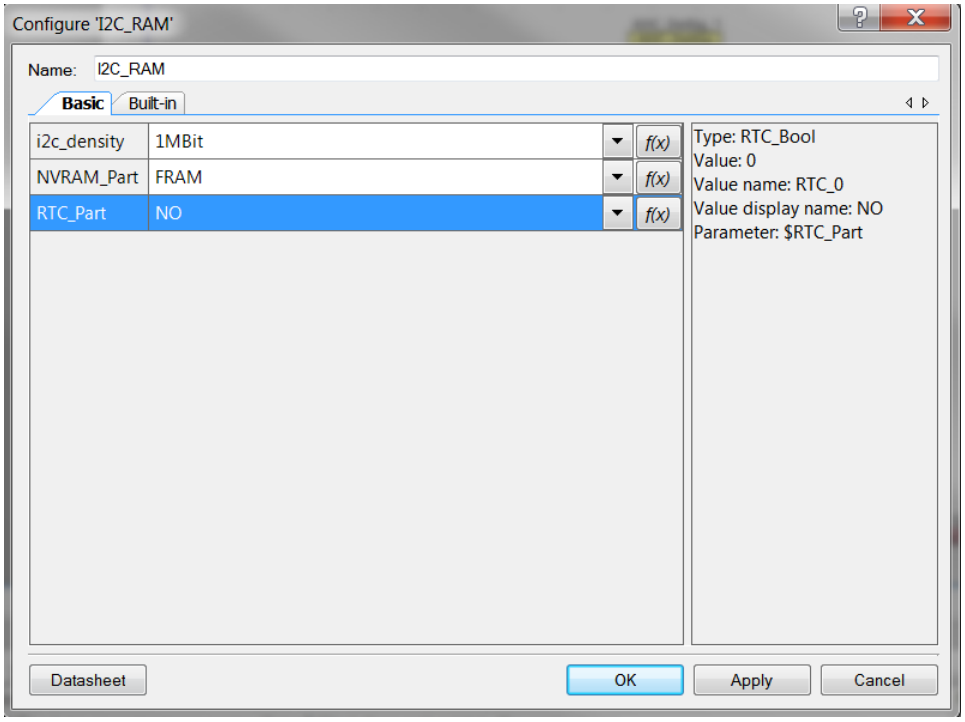


Figure 6. CapSense Component General Parameter

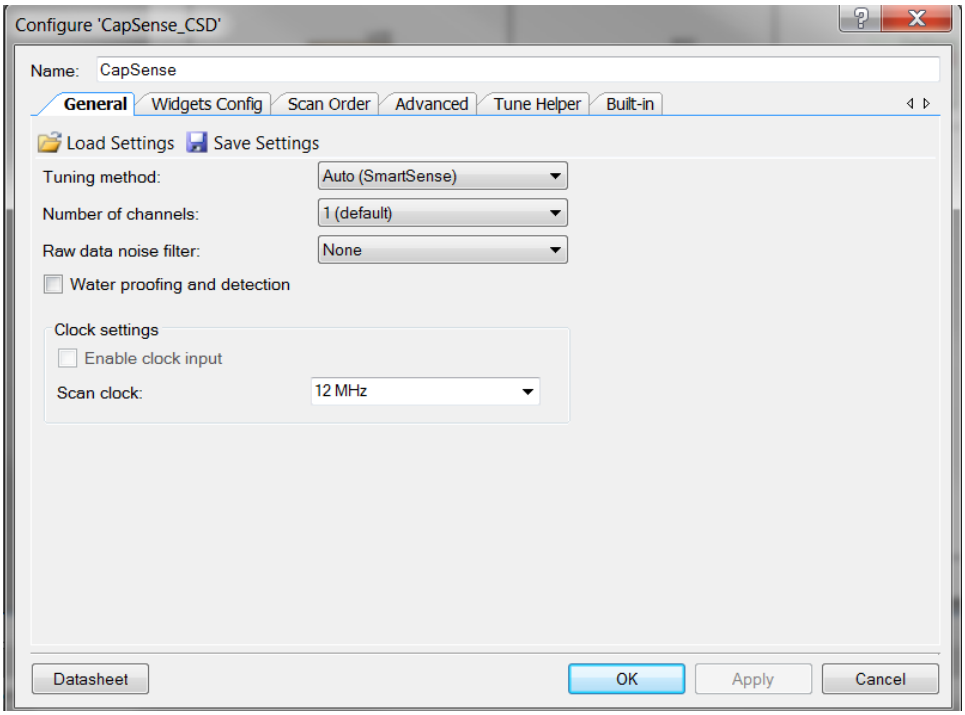


Figure 7. CapSense Widget Config Editor Window

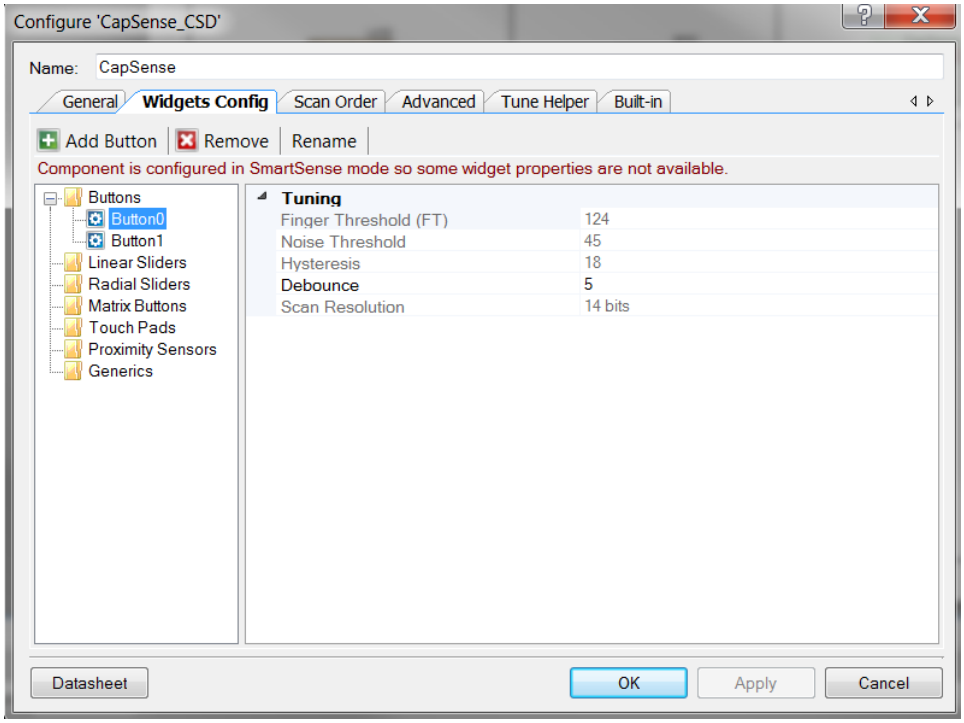
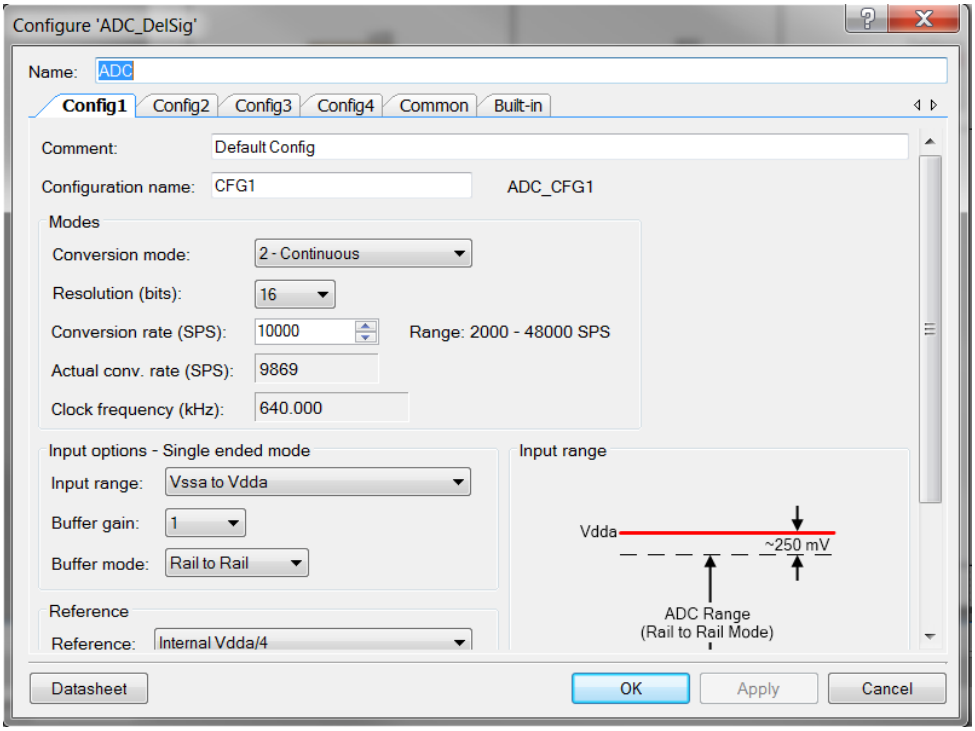


Figure 8. ADC Configuration Editing Window



Application Programming Interface

API routines allow you to configure the I2C_RAM Component using software. Table 2 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 "I2C_RAM_1" to the first instance of a I2C_RAM Component in each 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 "I2C_RAM".

Table 2. APIs

API	Description
I2C_RAM_Init()	Initialization routine for I2C_RAM_I2C component
I2C_RAM_Write(uint8, uint32 , uint8* , uint32)	I2C nvRAM Write Function
I2C_RAM_Current_Read(uint8, uint8* , uint32)	I2C nvRAM Current Read Function
I2C_RAM_Random_Read(uint8, uint32 , uint8* , uint32)	I2C nvRAM Radom Read Function
I2C_RAM_nvCommand(uint8, uint8)	function to Send nvRAM Command
I2C_RAM_Status_Reg_Read(uint8)	nvRAM Status Register Read Function
I2C_RAM_Status_Reg_Write(uint8 ,uint8)	nvRAM Status Register Write Function
I2C_RAM_Serial_No_Write(uint8 ,uint8*)	Function to write a serial number into the serial number register
I2C_RAM_Sleep(uint8)	This function puts the device into sleep mode
I2C_RAM_Serial_No_Read(uint8 ,uint8*)	Function to read the serial number of the device
I2C_RAM_Device_ID_Read(uint8 ,uint8*)	Function to read the device ID the slave

`void I2C_RAM_Init()`

Description: Initialization routine for I2C_RAM_I2C component

Parameters: None

Return: None

Side Effects: None

`uint8 I2C_RAM_Write(uint8 I2C_Address, uint32 Address, uint8* DATA, uint32 data_count)`

Description: I2C nvRAM Write Function

Parameters: uint8 I2C_Address: 7 bit slave ID

uint32 Address: 32-bit memory address for write

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

uint8 I2C_RAM_Current_Read(uint8 I2C_Address, uint8* DATA, uint32 data_count)

Description: I2C nvRAM Current Read Function

Parameters: `uint8 I2C_Address: 7bit slave ID`

`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

uint8 I2C_RAM_Random_Read(uint8 I2C_Address, uint8* DATA, uint32 data_count)

Description: I2C nvRAM Random Read Function

Parameters: `uint8 7 bit slave ID`

`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

uint8 I2C_RAM_nvCommand(uint8 I2C_Address, uint8 nvcmd)

Description: function Send nvRAM Command

Parameters: `uint8 I2C_Address: 7 bit Slave ID`

`uint8 nvcmd: 8 bit nvRAM Command`

Return: uint8 error status

Side Effects: None

uint8 I2C_RAM_Status_Reg_Read(uint8 I2C_Address)

Description: nvRAM Status Register Read Function

Parameters: `uint8` **`I2C_Address: 7 bit Slave ID`**

Return: uint8 1 byte status register value

Side Effects: None

uint8 I2C_RAM_Status_Reg_Write(uint8 I2C_Address,uint8 DATA)

Description: nvRAM Status Register Write Function

Parameters: `uint8` **`I2C_Address: 7 bit Slave ID`**

`uint8 DATA: 8 bit status register value`

Return: uint8 error status

Side Effects: None

uint8 I2C_RAM_Serial_No_Write(uint8 I2C_Address,uint8* DATA)

Description: nvSRAM Serial Number Write Function

Parameters: `uint8` **I2C_Address: 7 bit Slave ID**

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

Return: uint8 error status

Side Effects: None

uint8 I2C_RAM_Serial_No_Read(uint8 I2C_Address,uint8* DATA)

Description: nvSRAM Serial Number Read Function

Parameters: `uint8` **I2C_Address: 7 bit Slave ID**

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

Return: uint8 error status

Side Effects: None

uint8 I2C_RAM_Sleep(uint8 I2C_Address)

Description: Enables FRAM sleep mode

Parameters: `uint8` **I2C_Address: 7 bit Slave ID**

Return: uint8 error status

Side Effects: None

uint8 I2C_RAM_Device_ID_Read(uint8 I2C_Address,uint8* DATA)

Description: nvSRAM Device ID Read Function

Parameters: `uint8` **I2C_Address: 7 bit Slave ID**

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

Return: uint8 error status

Side Effects: None

Operation

This section shows how to import the I2C_RAM User Component into the code example and the how to use the APIs of the User Component.

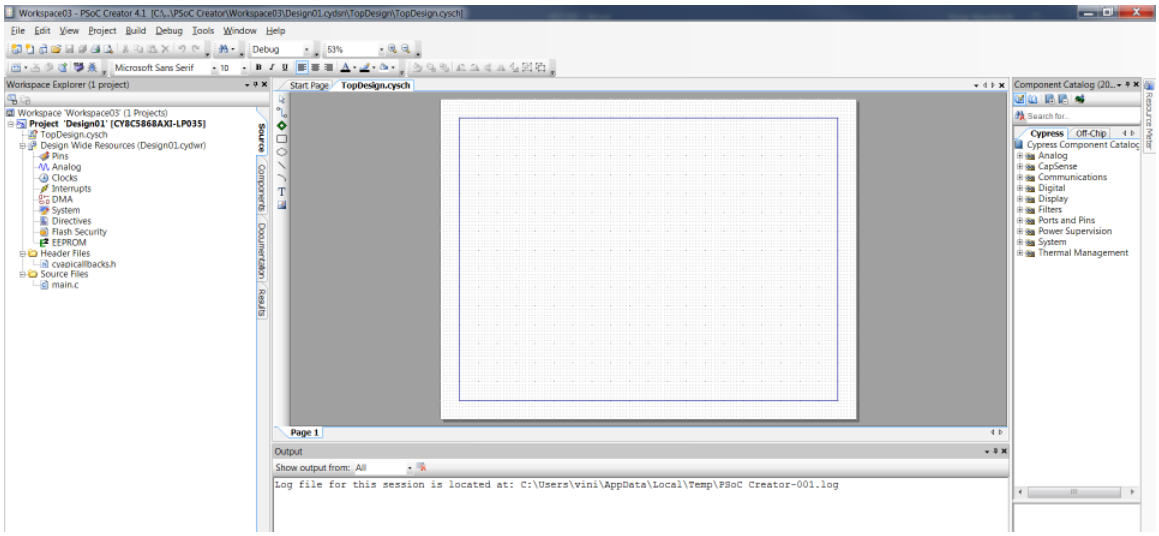
Setup

I2C_RAM archive contains both example project and I2C_RAM Component.

Follow these steps to use the I2C_RAM Component in your design:

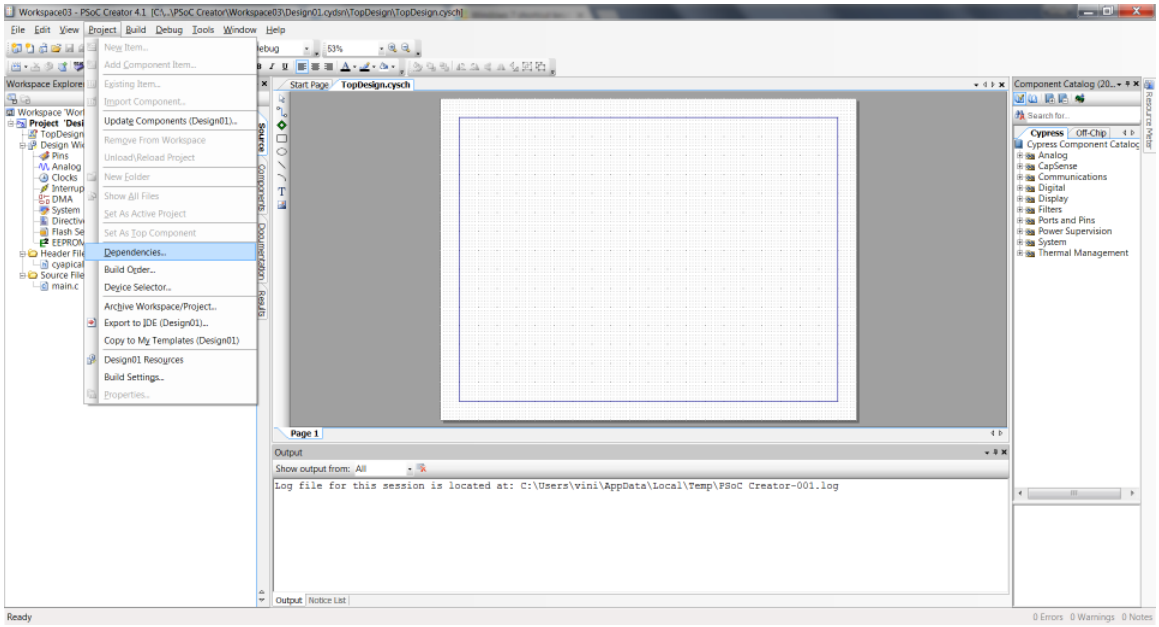
1. Open PSoC Creator and open your design (workspace) as shown in Figure 9.

Figure 9. Creating Project 'Design01'



2. Right-click the project and open the **Dependencies** tab on the workspace explorer, and then bring I2C_RAM Component into your design, as shown in Figure 10.

Figure 10. Opening Dependencies Tab



3. Click **New Entry (User Dependencies)** and then select *CE220573_I2C_RAM.cyprj* from the *CE220573_I2C_RAM.cydsn* folder (see Figure 11). The I2C_RAM Component appears under default/I2C_RAM in the Component Catalog (see Figure 12).

Figure 11. Importing User Component

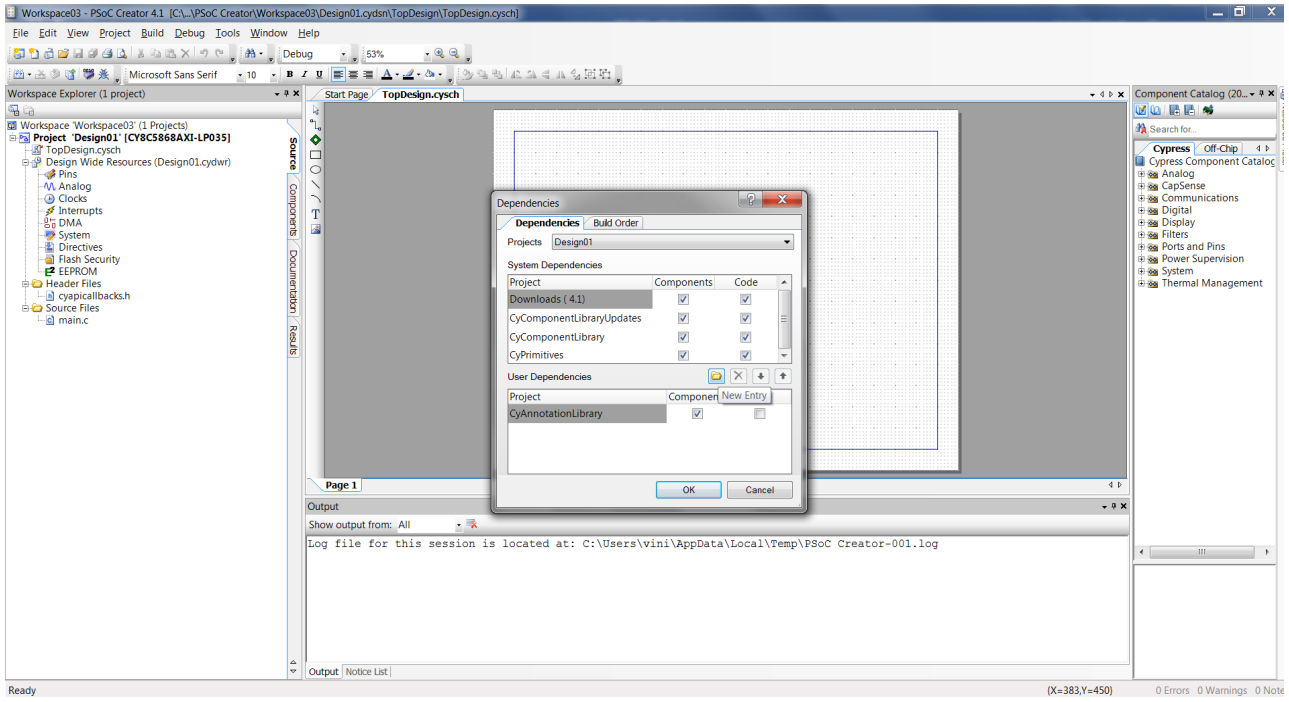
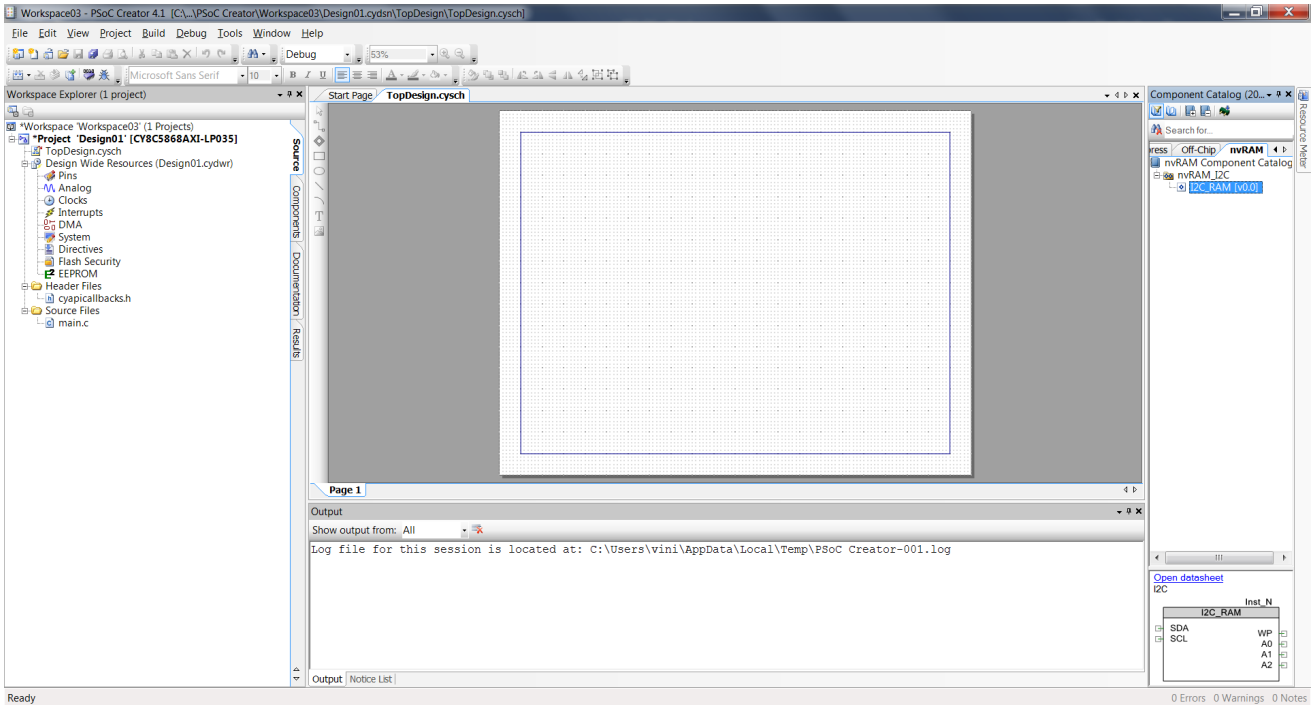
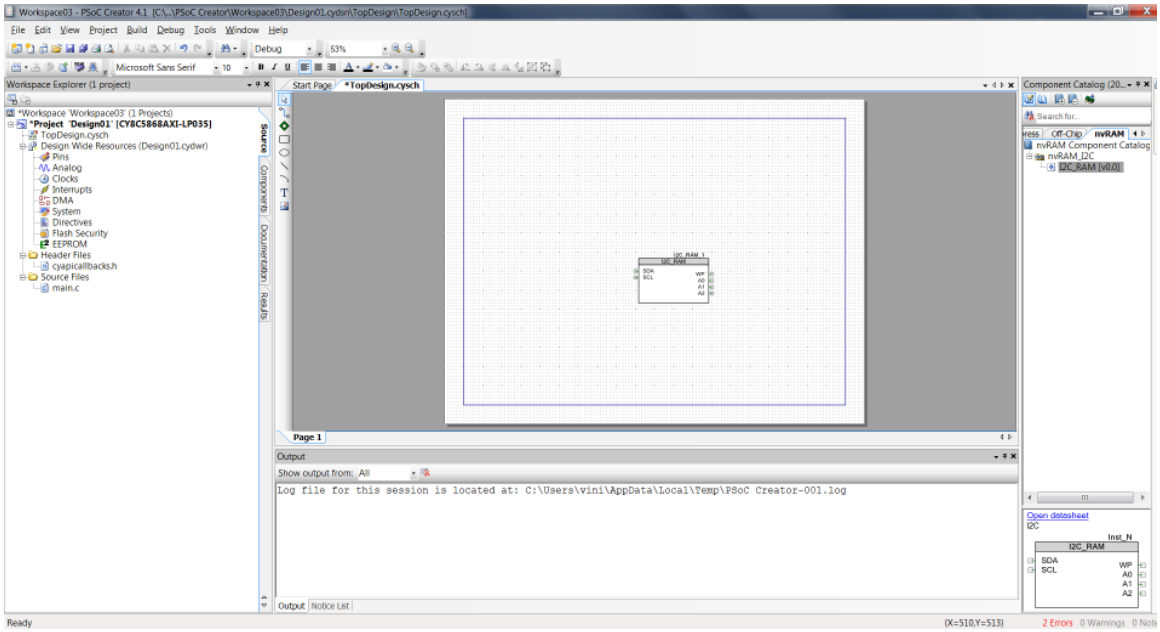


Figure 12. Displaying I2C_RAM Component Under Component Catalog



4. To add the I2C_RAM Component to your design, drag and drop the I2C_RAM Component onto *TopDesign.cysch* and assign Digital I/Os from the **Ports and Pins** Component, Clock source, and so on. as shown in Figure 13.

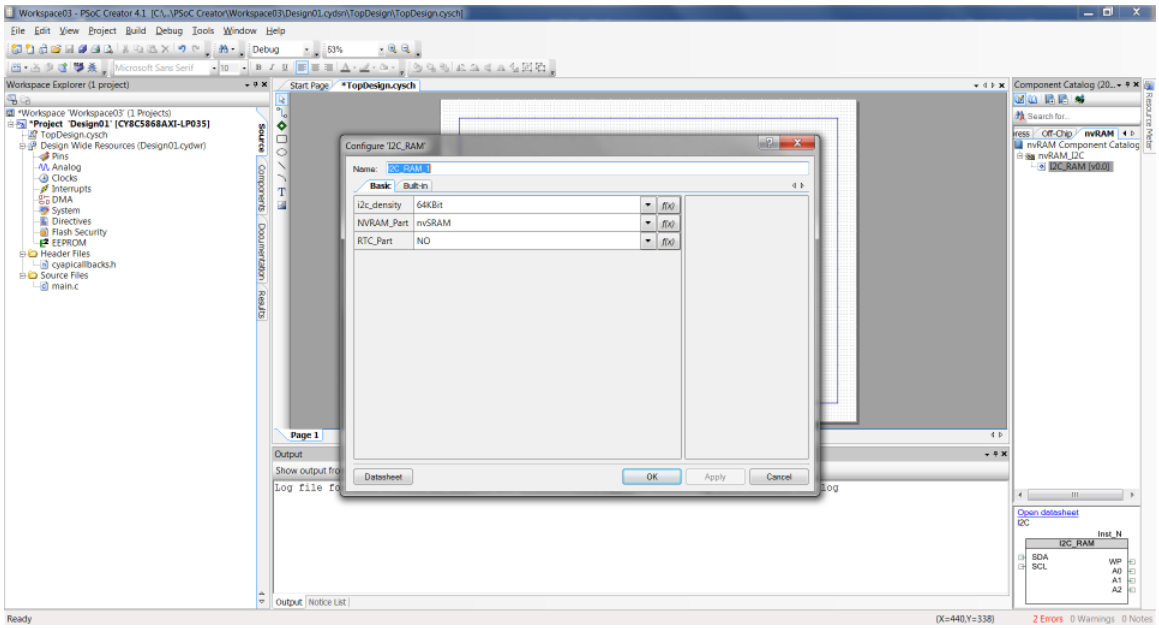
Figure 13. I2C_RAM Component Usage



5. Configure I2C_RAM:

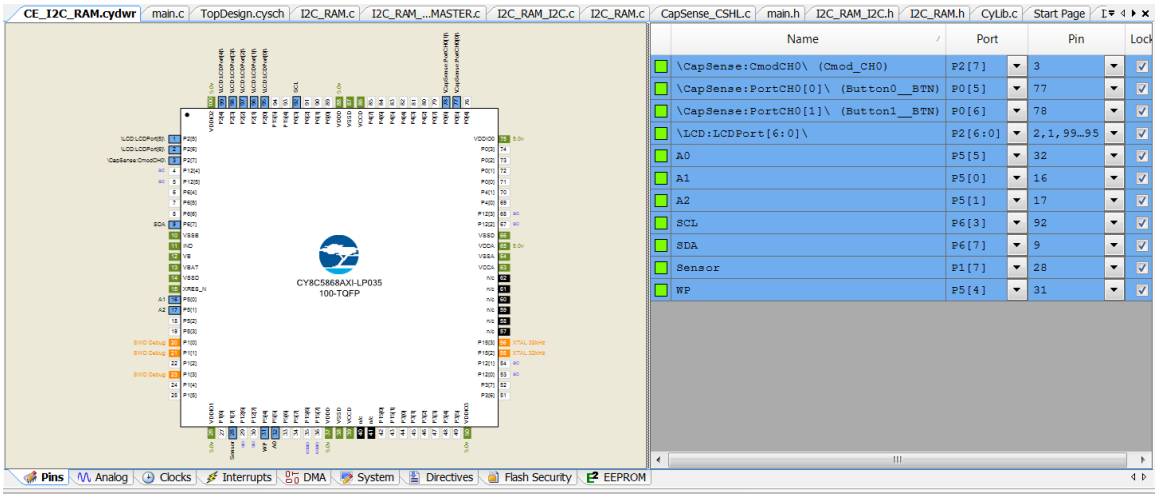
   a. Right-click the **I2C_RAM_1 Component** in *TopDesign.cysch* and select **Configure** and set the required parameters. Select the density and nvRAM part accordingly. Keep the **RTC_part** parameter as **No**.

Figure 14. User Component Configuration



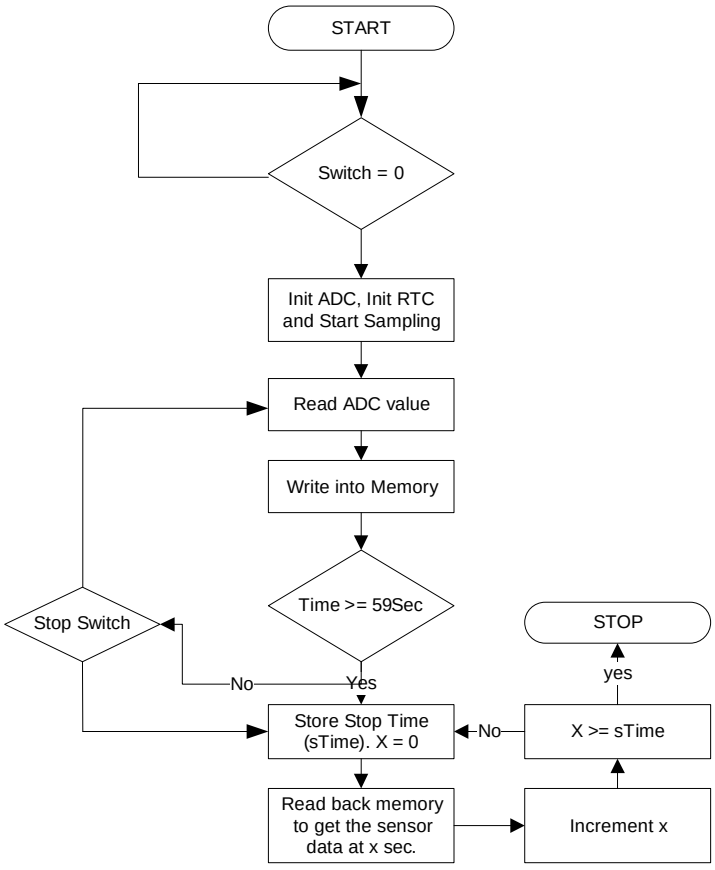
b. Assign appropriate input/output pins as per your design and build the project.

Figure 15. Pin Assignment



Code Example (Flowchart)

Figure 16. Code Example Flow Chart



Exporting to Eclipse IDE

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

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

In Eclipse, right-click the custom components and select **Resource Configurations** > **Exclude from Build…**. In the **Exclude from build** dialog, click **Select All**, and then click **OK**. For more information on exporting PSoC projects to Eclipse IDE, see PSoC Creator Help.

Figure 17. Exclude Custom Component (I2C_RAM) in Eclipse

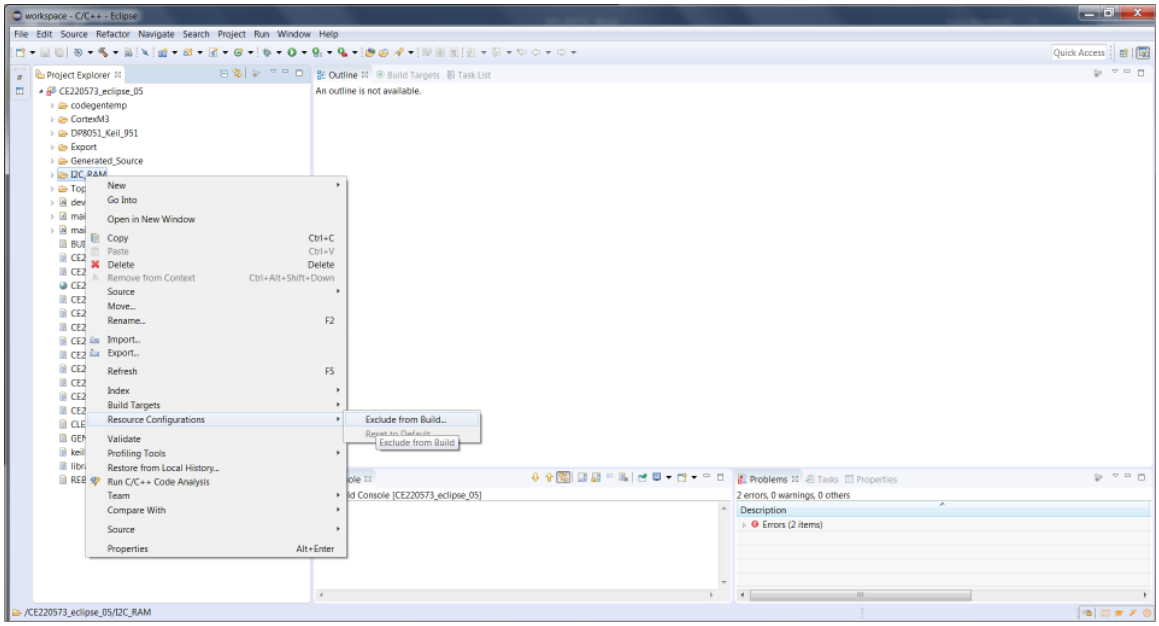


Figure 18. Exclude Custom Components for All Builds

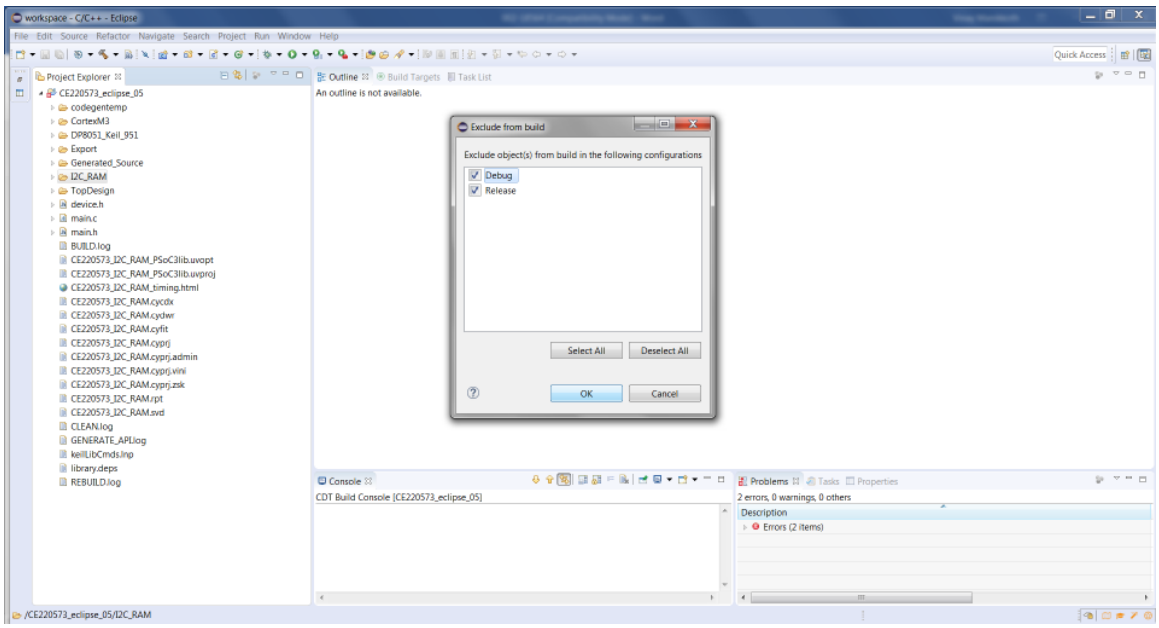
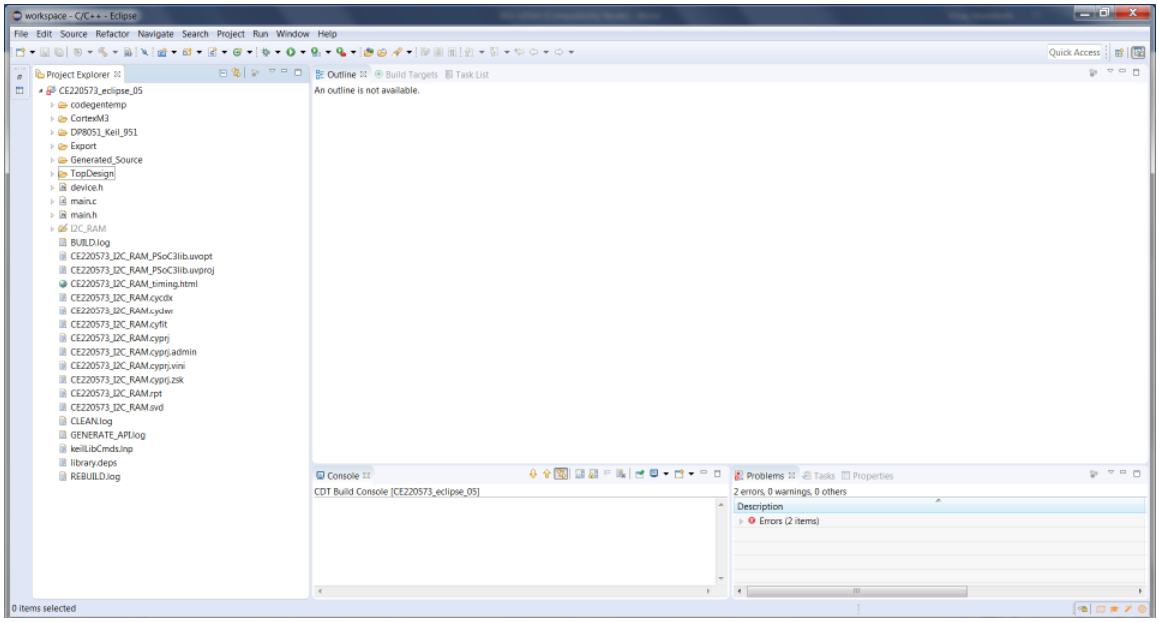


Figure 19. Custom Components Excluded in Eclipse and Build the Project



Code Example

This section provides the sample codes to access the I2C_RAM User Component APIs. The complete code can be found in *main.c* file of the code example. Component instance is I2C_RAM_1.

```
// API I2C_RAM _1_Start initializes the I2C_RAM user component

      I2C_RAM_I2C_Start();
```

```
// API I2C_RAM_1_Write writes the DATA into the memory at the “address”.
      I2C_RAM_Write(I2C_Address,sTime,sValue,1);
```

```
// API I2C_RAM_1_Current_Read Reads the DATA from current address of the memory.

      I2C_RAM_Current_Read(I2C_Address,0x0001,buff);
```

```
// API I2C_RAM_1_Random_Read Reads the DATA from address “ADDR” of the memory.

      I2C_RAM_Random_Read(I2C_Address,0x0001,buff,ADDR);
```

```
// API I2C_RAM_1_Read_DeviceID reads the Device ID of the device.
      I2C_RAM_Device_ID_Read(I2C_Address,device_ID);
```

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	
FM24W256	256-Kbit (32 K x 8) Serial (I2C) F-RAM	

Document History

Document Title: CE220573 - Interfacing I²C NVSRAM and F-RAM with PSoC 3/5LP

Document Number: 002-20573

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	4406551	VINI	01/02/2018	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 MCU

Cypress Developer Community

Community Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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