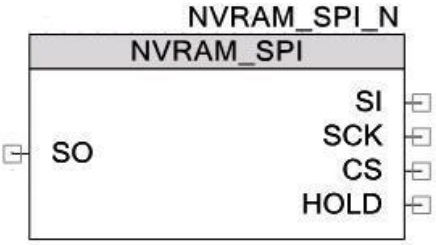
NVRAM SPI PSoC3 Creator Component User Guide

Features



- Mode 0, Mode 3 SPI operating modes
- Supports up to 4MHz of SPI operation[1]
- nvSRAM / F-RAM Read/Write
- nvSRAM RTC / F-RAM Processor Companion Read/Write[2]
- nvSRAM Commands (STORE, RECALL, ASENB, ASDISB)
- nvSRAM / F-RAM SLEEP Command[2]
- nvSRAM / F-RAM Product ID Read[2]
- nvSRAM / F-RAM Serial Number Read[2]
- nvSRAM / F-RAM Serial Number Write[2]

Notes:

1. This library component is tested up to 4MHz of SPI Clock generation. This can go up the maximum frequency supported by PSoC3 SPI component.
2. These features are not supported by all the parts. Please refer to the specific device datasheet for supportability.

General Description

The NVRAM_SPI component provides a PSoC3 SPI Master interface to SPI nvSRAM and F-RAM devices. This component can be directly used in user's PSoC3 designs. High level API's are provided for different nvSRAM / F-RAM operations. User can directly use these API's as it reduces the time to integrate nvSRAM / F-RAM in the user application.

Input/output Connections

This section describes the various input and output connections for the NVRAM (nvSRAM / F-RAM) component.

SO – Input

The SO input carries the master input (PSoC3) – slave output serial data from the nvSRAM / F-RAM device.

SI – Output

The SI output carries the master output – slave (nvSRAM / F-RAM) input serial data from the master device (PSoC3).

SCK– Output

The SCK output carries the master (PSoC3) synchronization clock output to the nvSRAM / F-RAM.

CS – Output

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

HOLD – Output

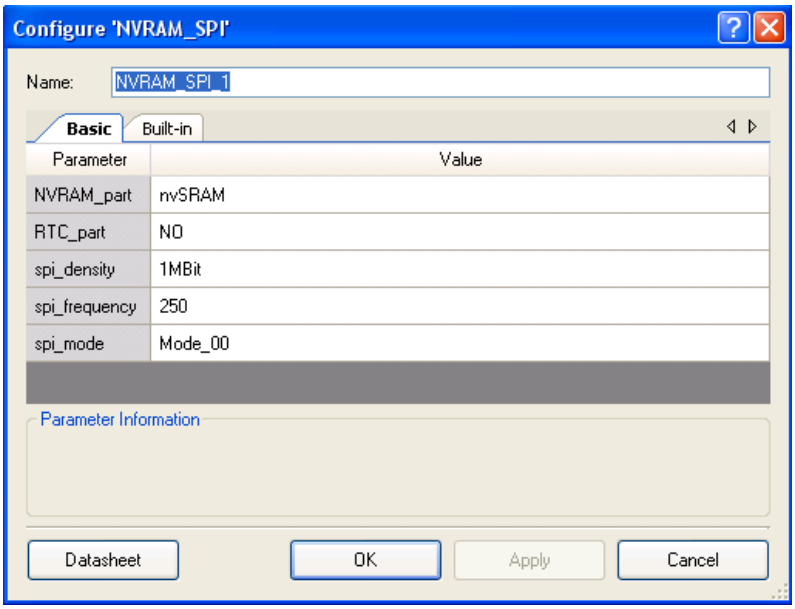
The HOLD output carries the SPI HOLD signal to the nvSRAM / F-RAM device.

Parameters

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

Basic Tab

This tab is used to set the NVRAM part, RTC part, density, SPI frequency and mode of operation. Eg: nvSRAM part is selected by configuring NVRAM_part to nvSRAM, non-RTC part is selected by configuring RTC_part to “NO”, 1MBit density, 250 for 250 KHz and Mode 0.



Hardware Configuration

NVRAM_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 "NVRAM_SPI_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 "NVRAM_SPI".

Function	Description	Applicability (nvSRAM / F-RAM)
NVRAM_SPI_Init	Initialization routine for NVRAM_SPI component	nvSRAM & F-RAM
NVRAM_SPI_Write	SPI NVRAM SRAM Write	nvSRAM & F-RAM
NVRAM_SPI_Read	SPI NVRAM SRAM Read	nvSRAM & F-RAM
NVRAM_SPI_RTC_Write	SPI NVRAM RTC Write	nvSRAM and F-RAM Processor companion
NVRAM_SPI_RTC_Read	SPI NVRAM RTC Read	nvSRAM and F-RAM Processor companion
NVRAM_SPI_nvCommand	Send NVRAM Command	nvSRAM only
NVRAM_SPI_Sleep	NVRAM Sleep mode entry	nvSRAM & F-RAM
NVRAM_SPI_Status_Reg_Read	NVRAM Status Register Read	nvSRAM & F-RAM
NVRAM_SPI_Status_Reg_Write	NVRAM Status Register Write	nvSRAM & F-RAM
NVRAM_SPI_Serial_No_Write	NVRAM Serial Number Write	All nvSRAM & F-RAM processor companions
NVRAM_SPI_Serial_No_Read	NVRAM Serial Number Read	All nvSRAM & F-RAM 1MBit & FRAM Processor companions

Function	Description	Applicability (nvSRAM / F-RAM)
NVRAM_SPI_Device_ID_Read	NVRAM Device ID Read	All nvSRAM & F-RAMs above 128Kbit

void NVRAM_SPI_Init(void)

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

Parameters: None

Return Value: None

Side Effects: None

void NVRAM_SPI_Write(uint32 addr, uint8 *data_write_ptr, uint32 total_data_count)

Description: Write total_data_count number of data into NVRAM.

Parameters: uint32 addr: 32 bit NVRAM 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: None

Side Effects: None

void NVRAM_SPI_Read(uint32 addr, uint8 *data_read_ptr, uint32 total_data_count)

Description: Read total_data_count number of data from NVRAM.

Parameters: uint32 addr : 32 bit NVRAM 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: None

Side Effects: None

void NVRAM_SPI_RTC_Write(uint8 addr, uint8 *data_write_ptr, uint8 total_data_count)

Description: Write total_data_count number of data into NVRAM RTC / F-RAM Processor Companion registers.

Parameters: uint8 addr : 8 bit NVRAM RTC / F-RAM Proc companion register address for write.
uint8 *data_write_ptr: Pointer to an array of RTC data bytes to be written.
uint8 total_data_count: Number of RTC data bytes to be written.

Return Value: None

Side Effects: None

void NVRAM_SPI_RTC_Read(uint8 addr, uint8 *data_read_ptr, uint8 total_data_count)

Description: Read total_data_count number of data from nvSRAM RTC / F-RAM Processor Companion registers.

Parameters: uint8 addr : 8 bit nvSRAM RTC / F-RAM Proc companion register address for read.
uint8 *data_read_ptr: Pointer to an array for storing RTC data bytes.
uint8 total_data_count: Number of RTC data bytes to be read.

Return Value: None

Side Effects: None

void NVRAM_SPI_ nvCommand (uint8 nvcmd)

Description: Send NVRAM Command. This is supported by nvSRAM only.

Parameters: uint8 nvcmd: 8 bit NVRAM Command

Return Value: None

Side Effects: None

void NVRAM_SPI_ Sleep (void)

Description: Send NVRAM Command for Sleep.

Parameters: None

Return Value: None

Side Effects: None

uint8 NVRAM_SPI_ Status_Reg_Read (void)

Description: NVRAM Status Register Read

Parameters: None

Return Value: uint8: 1 byte Status register data

Side Effects: None

void NVRAM_SPI_ Status_Reg_Write (uint8 data_byte)

Description: NVRAM Status Register Write

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

Return Value: None

Side Effects: None

void NVRAM_SPI_ Serial_No_Write (uint8 *data_ptr)

Description: NVRAM Serial Number Write. This is supported by nvSRAM and FRAM Processor companions only.

Parameters: uint8 *data_ptr -> pointer to an array of serial number data to be written.

Return Value: None

Side Effects: None

void NVRAM_SPI_ Serial_No_Read (uint8 *data_ptr)

Description: NVRAM Serial Number Read. This is supported by nvSRAM, 1MBit FRAM and FRAM processor companions only.

Parameters: uint8 *data_ptr -> pointer to an array for storing serial number data.

Return Value: None

Side Effects: None

void NVRAM_SPI_ Device_ID_Read (uint8 *data_ptr)

Description: NVRAM Device ID Read. This is supported by nvSRAM and FRAM above 128Kbit.

Parameters: uint8 *data_ptr -> pointer to an array for storing device id.

Return Value: None

Side Effects: None

Defines

All the NVRAM definitions are defined in NVRAM_SPI_definition.h

```
// nvSRAM Commands
#define nvSRAM_STORE_CMD        ((uint8)0x3C)    // Soft Store
#define nvSRAM_RECALL_CMD       ((uint8)0x60)    // Soft Recall
#define nvSRAM_ASENB_CMD        ((uint8)0x59)    // AutoStore Enable
#define nvSRAM_ASDISB_CMD       ((uint8)0x19)    // AutoStore Disable

// RTC Definitions
#define nvSRAM_RTC_REG_FLAGS            ((uint8)0x00)    // RTC Falgs Register
#define nvSRAM_RTC_REG_CENTURY          ((uint8)0x01)    // RTC Century Register
#define nvSRAM_RTC_REG_ALARM_SEC        ((uint8)0x02)    // RTC Alarm Seconds
#define nvSRAM_RTC_REG_ALARM_MIN        ((uint8)0x03)    // RTC Alarm Minutes
#define nvSRAM_RTC_REG_ALARM_HOUR       ((uint8)0x04)    // RTC Alarm Hours
#define nvSRAM_RTC_REG_ALARM_DAY        ((uint8)0x05)    // RTC Alarm Day (Date)
#define nvSRAM_RTC_REG_INT_FLAGS        ((uint8)0x06)    // RTC Interrupt Register
#define nvSRAM_RTC_REG_WATCHDOG         ((uint8)0x07)    // RTC WatchDog Register
#define nvSRAM_RTC_REG_CALIBRATION      ((uint8)0x08)    // RTC Calibration Register
#define nvSRAM_RTC_REG_SECOND           ((uint8)0x09)    // RTC Seconds Register
#define nvSRAM_RTC_REG_MINUTE           ((uint8)0x0A)    // RTC Minutes Register
#define nvSRAM_RTC_REG_HOUR             ((uint8)0x0B)    // RTC Hour Register
#define nvSRAM_RTC_REG_DAY              ((uint8)0x0C)    // RTC Day of Week Register
#define nvSRAM_RTC_REG_DATE             ((uint8)0x0D)    // RTC Day (Date) Register
```

```
#define nvSRAM_RTC_REG_MONTH           ((uint8)0x0E)    // RTC Month Register
#define nvSRAM_RTC_REG_YEAR            ((uint8)0x0F)    // RTC Year Register

// RTC Flags Register
#define nvSRAM_RTC_FLAGS_READ_ENABLE_BIT    ((uint8)0x01)  // RTC Flags Read Enable Bit
#define nvSRAM_RTC_FLAGS_WRITE_ENABLE_BIT   ((uint8)0x02)  // RTC Flags Write Enable Bit
#define nvSRAM_RTC_FLAGS_CAL_BIT            ((uint8)0x04)  // RTC Flags CAL (Calibration
Mode) Bit
#define nvSRAM_RTC_FLAGS_BPF_BIT            ((uint8)0x08)  // RTC Flags BPF (Backup
Power Fail) Bit
#define nvSRAM_RTC_FLAGS_OSCF_BIT           ((uint8)0x10)  // RTC Flags OSCF (Oscillator
Fail) Bit
#define nvSRAM_RTC_FLAGS_PF_BIT             ((uint8)0x20)  // RTC Flags PF (Power Fail
Flag) Bit
#define nvSRAM_RTC_FLAGS_AF_BIT             ((uint8)0x40)  // RTC Flags AF (Alarm Flag)
Bit
#define nvSRAM_RTC_FLAGS_WDF_BIT            ((uint8)0x80)  // RTC Flags WDF (WatchDog
Flag) Bit

// RTC Interrupt Control Register
#define nvSRAM_RTC_INT_SQ0_BIT              ((uint8)0x01)  // Square Wave Freq Bit 0.
#define nvSRAM_RTC_INT_SQ1_BIT              ((uint8)0x02)  // Square Wave Freq Bit 1.
#define nvSRAM_RTC_INT_P_L_BIT              ((uint8)0x04)  // Pulse or Level Bit for INT
pin.
#define nvSRAM_RTC_INT_H_L_BIT              ((uint8)0x08)  // High/Low active mode for
INT pin.
#define nvSRAM_RTC_INT_SQWE_BIT             ((uint8)0x10)  // Square Wave Enable Bit.
#define nvSRAM_RTC_INT_PFE_BIT              ((uint8)0x20)  // Power Fail Interrupt
Enable Bit.
#define nvSRAM_RTC_INT_AIE_BIT              ((uint8)0x40)  // Alarm Interrupt Enable
Bit.
#define nvSRAM_RTC_INT_WIE_BIT              ((uint8)0x80)  // Watchdog Interrupt Enable
Bit.

// RTC Watchdog Timer Register
#define nvSRAM_RTC_WDT_WRITE_ENABLE_BIT     ((uint8)0x40)  // Watchdog write enable bit

// Oscillator Calibration Register
#define nvSRAM_RTC_CAL_SIGN_BIT             ((uint8)0x20)  // RTC Calibration sign bit
#define nvSRAM_RTC_CAL_OSCEN_BIT            ((uint8)0x80)  // RTC Oscillator Enable Bit

// NVRAM Commands
#define NVRAM_WREN             ((uint8)0x06)   // Write Enable
#define NVRAM_WRDI             ((uint8)0x04)   // Write Disable
#define NVRAM_SRAM_WRITE_CMD   ((uint8)0x02)   // SRAM Write
#define NVRAM_SRAM_READ_CMD    ((uint8)0x03)   // SRAM Read
#define NVRAM_RTC_WRITE_CMD    ((uint8)0x12)   // RTC Register Write
#define NVRAM_RTC_READ_CMD     ((uint8)0x13)   // RTC Register Read
#define NVRAM_RDSR_CMD         ((uint8)0x05)   // Read Status Register
#define NVRAM_WRSR_CMD         ((uint8)0x01)   // Write Status Register
#define NVRAM_RDSN_CMD         ((uint8)0xC3)   // Read Serial Number
#define NVRAM_WRSN_CMD         ((uint8)0xC2)   // Write Serial Number
#define NVRAM_RDID_CMD         ((uint8)0x9F)   // Read Device ID
#define NVRAM_SLEEP_CMD        ((uint8)0xB9)   // Sleep Command

#define FRAM_SPI_SN_REGISTER   ((uint8)0x10)
```

Setup

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

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

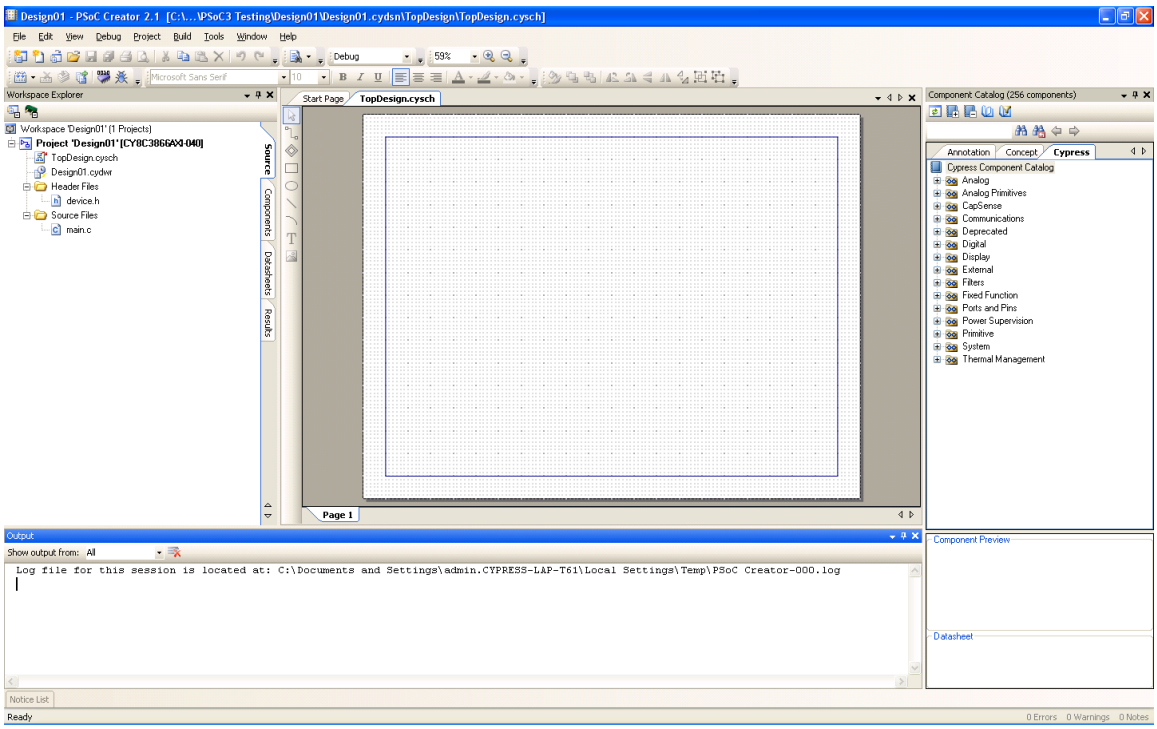


Fig 1: Create project 'Design01'

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

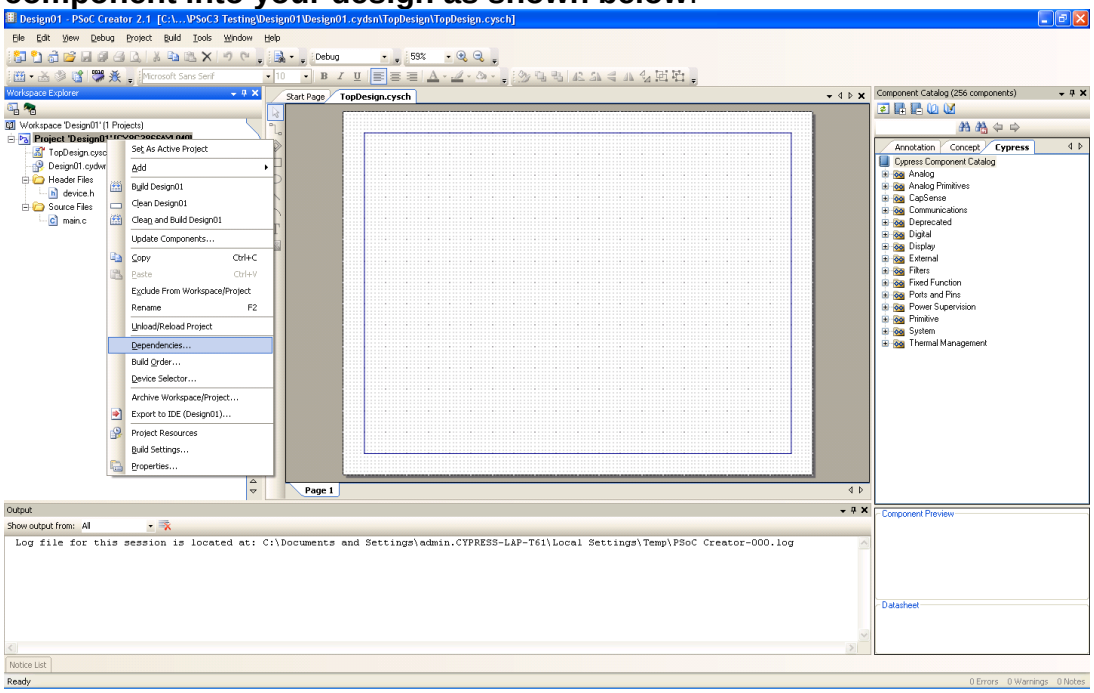


Fig 2: Right click on the project and Open Dependencies

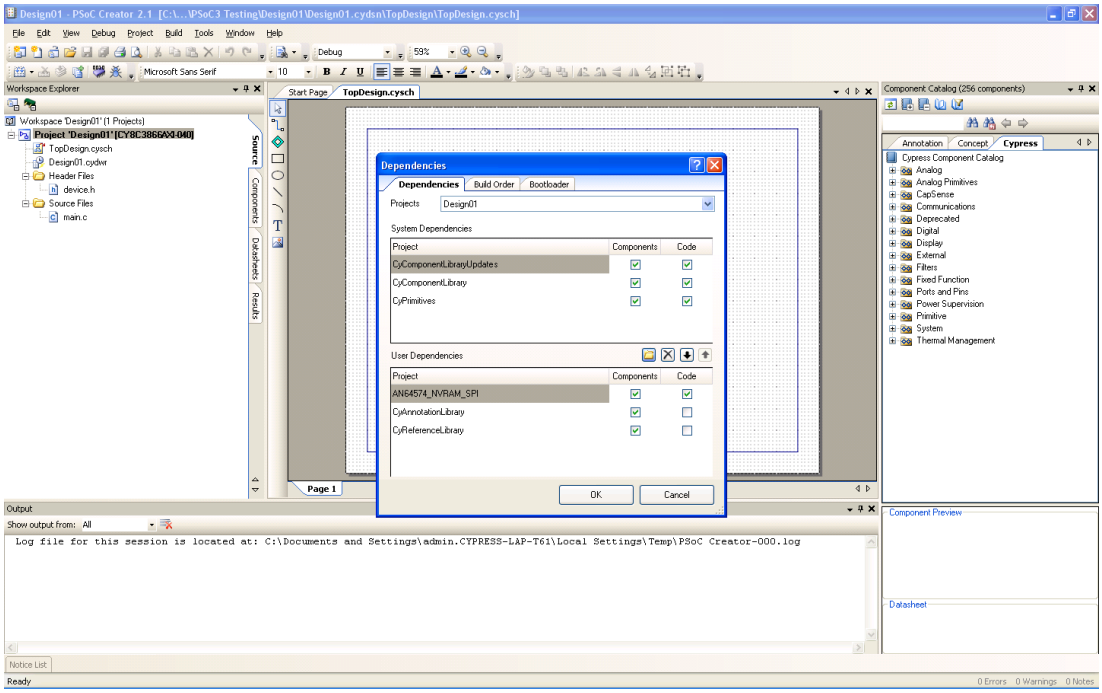


Fig 3: Click on New Entry (User Dependencies) and Select 'AN64574_NVRAM_SPI.cyprj' from AN64574_NVRAM_SPI.cydsn folder.

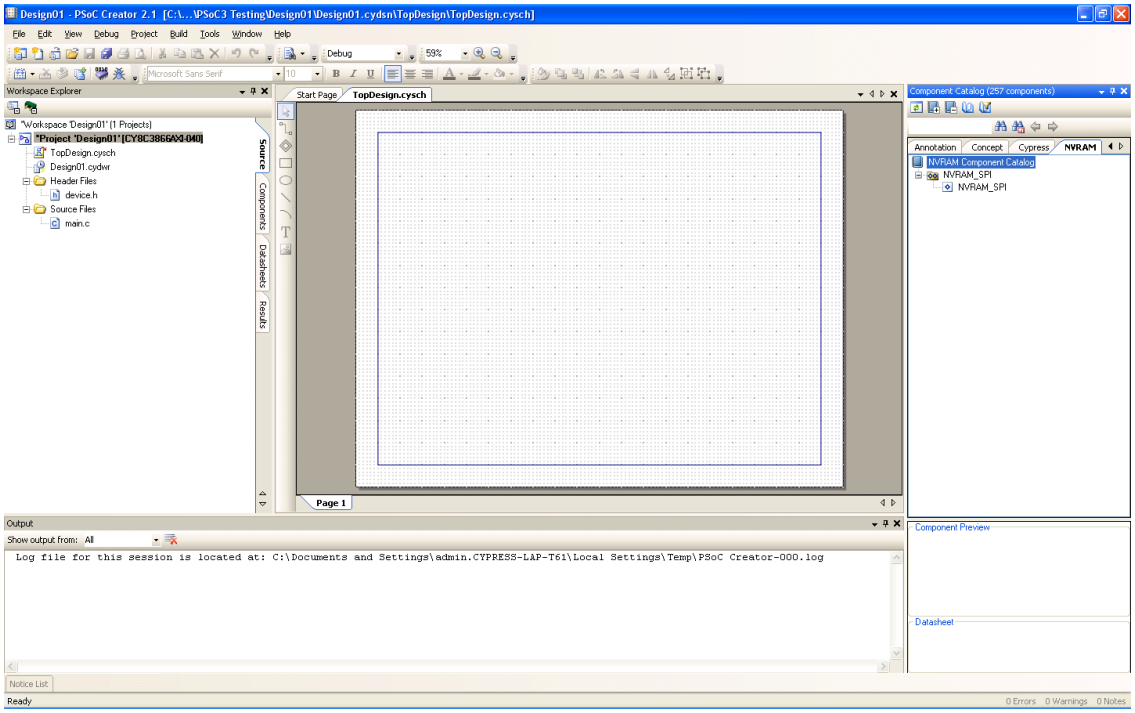


Fig 4: NVRAM_SPI component appears under NVRAM/NVRAM_SPI in component Catalog

3. Add NVRAM_SPI component to your design as follows

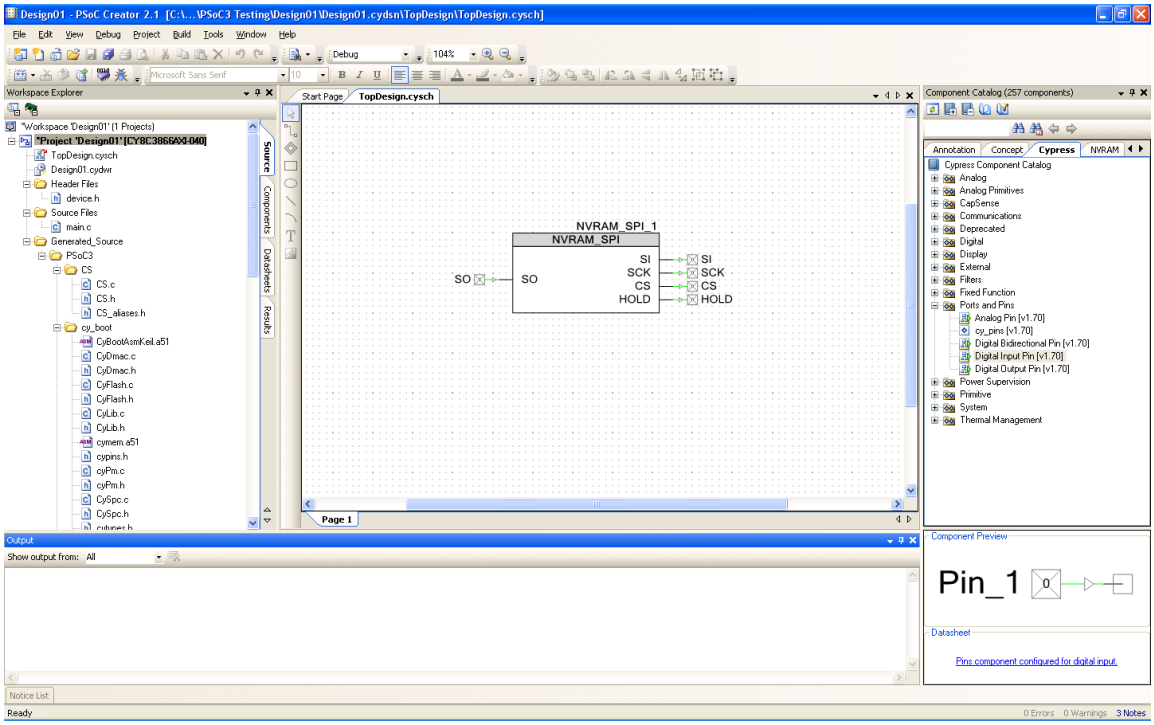


Fig 5: Drag and drop NVRAM_SPI component onto TopDesign.cysch and assign Digital IOs.

4. NVRAM_SPI Configuration

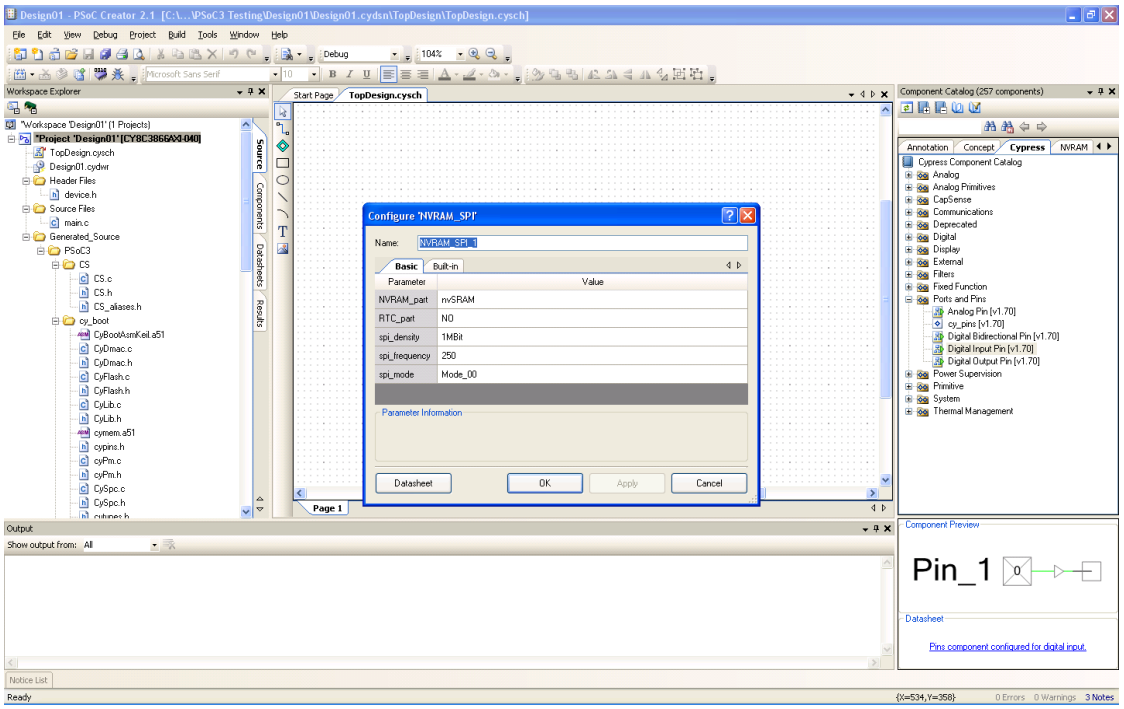


Fig 6: Right click on the NVRAM_SPI_1 component in TopDesign.cysch and select configure. Set parameters NVRAM_part, RTC_part, spi_density, spi_frequency and spi_mode as per your design. In this project we have selected 1MBit non-RTC nvSRAM part, 250 KHz SPI frequency and mode 0.

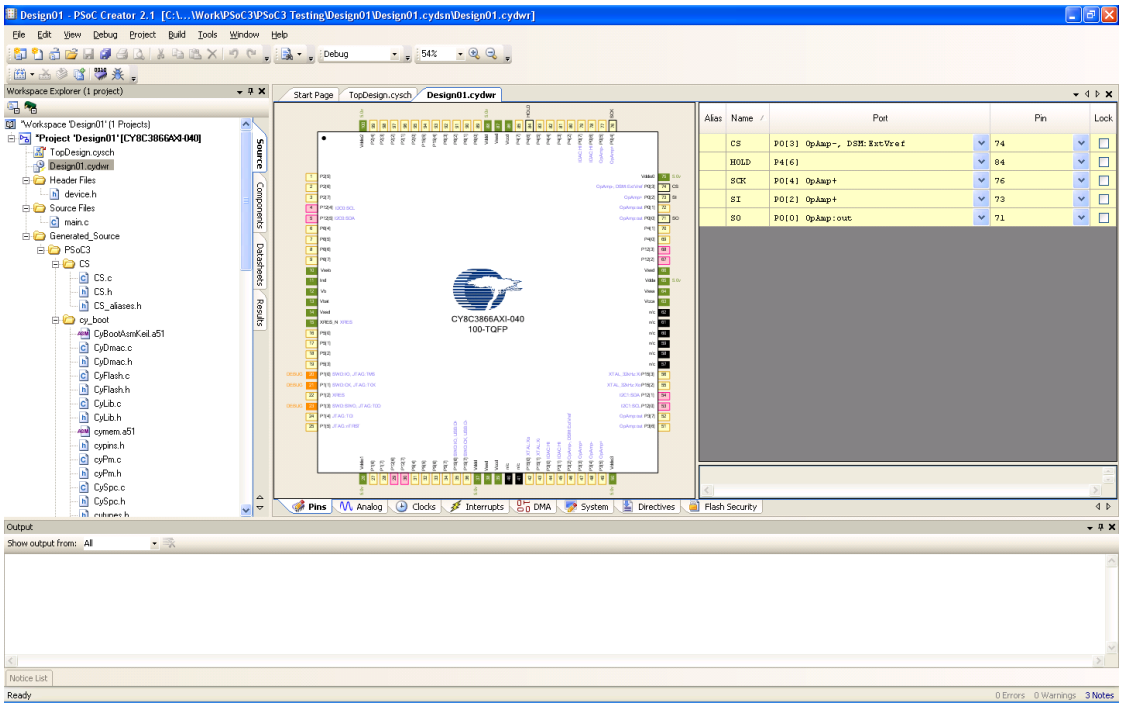


Fig 7: Assign appropriate input/output pins as per your design and build the project.

Sample Firmware Source Code

Below is a sample code for different nvSRAM / F-RAM operations taken from the example project. Component instance is NVRAM_SPI_1. The example project can be opened by selecting AN64574_NVRAM_SPI workspace in AN64574_NVRAM_SPI folder. The application is mainly written for nvSRAM but can be used for F-RAM also.

```
void main()
{

      uint8 data_bytes[10] = {1,2,3,4,5,6,7,8,9,10};
      uint8 led_state = 0, status_reg = 0;
      uint8 rtc_data[10] = {1,2,3,4,5,6,7,8,9,64};
      uint8 nvram_rd_data[10] = {1,2,3,4,5,6,7,8,9,10};
      uint8 nvram_rd_data_before_recall[10];
      uint8 nvram_rd_data_after_recall[10];
      uint8 device_id[4];
      uint8 serial_number[8];
      uint8 rtc_send_data;
      uint8 i;

      rtc_send_data = 0;
      for(i = 0; i < 16; i++)
      {
         rtc_data[i] = 0;
      }

      for(i = 0; i < 10; i++)
      {
         nvram_rd_data[i] = 0;
         nvram_rd_data_before_recall[i] = 0;
         nvram_rd_data_after_recall[i] = 0;
      }

      for(i = 0; i < 8; i++)
         serial_number[i] = 0;

      for(i = 0; i < 4; i++)
         device_id[i] = 0;

      NVRAM_SPI_1_Init();
      CYGlobalIntEnable; // Enable global interrupts.

      CyDelay(1000);

    // NVRAM Write : Write 4 bytes from array data_bytes to NVRAM location 0x123456
      CyDelay(1);
      NVRAM_SPI_1_Write(0x123456, data_bytes, 4);

    // NVRAM Status Register Write : Write 0x08 to status register
      CyDelay(1);
      NVRAM_SPI_1_Status_Reg_Write(0x08);

    // NVRAM Status Register Read : Read NVRAM Status Register
      CyDelay(1);
      status_reg = NVRAM_SPI_1_Status_Reg_Read();
```

```
    // NVRAM Status Register Write : Write 0x00 to status register
      CyDelay(1);
      NVRAM_SPI_1_Status_Reg_Write(0x00);

#if (NVRAM_SPI_1_NVRAM_part == 1)
    // NVRAM Serial Number Write : Write 8 bytes from array data_bytes to serial number
register
      CyDelay(1);
      NVRAM_SPI_1_Serial_No_Write(data_bytes);
#else
      #if (NVRAM_SPI_1_RTC_part == 1)
         NVRAM_SPI_1_Serial_No_Write(data_bytes);
      #endif
#endif

#if (NVRAM_SPI_1_RTC_part == 1)
    NVRAM_SPI_1_RTC_Read(0,rtc_data,1);

    // NVRAM RTC Write : Set Write Bit
      rtc_send_data = 0x02;
      CyDelay(1);
    NVRAM_SPI_1_RTC_Write(nvSRAM_RTC_REG_FLAGS, &rtc_send_data, 1);

    // NVRAM RTC Write : Write 1 byte data to nvSRAM_RTC_REG_SECOND register (0x09)
      CyDelay(1);
    NVRAM_SPI_1_RTC_Write(nvSRAM_RTC_REG_CENTURY, data_bytes, 8);

    // NVRAM RTC Write : Clear Write Bit
      rtc_send_data = 0x00;
      CyDelay(1);
    NVRAM_SPI_1_RTC_Write(nvSRAM_RTC_REG_FLAGS, &rtc_send_data, 1);
#endif

#if (NVRAM_SPI_1_NVRAM_part == 1)
    // nvSRAM Command :  Send SoftStore Command
      CyDelay(1);
      NVRAM_SPI_1_nvCommand(nvSRAM_STORE_CMD);
      CyDelay(10);
#endif
    // NVRAM SRAM Read : Read 4 bytes of data from SRAM starting from address 0x123456.
    //                   Data will be stored into nvram_rd_data array.
      CyDelay(1);
      NVRAM_SPI_1_Read(0x123456,nvram_rd_data,4);

#if (NVRAM_SPI_1_RTC_part == 1)
    // NVRAM RTC Read : Set Read Bit
      rtc_send_data = 0x01;
      CyDelay(1);
    NVRAM_SPI_1_RTC_Write(nvSRAM_RTC_REG_FLAGS, &rtc_send_data, 1);

    // NVRAM RTC Read : Read 8 bytes of data from RTC registers starting from
nvSRAM_RTC_REG_FLAGS (0x00) register
    CyDelay(1);
    NVRAM_SPI_1_RTC_Read(nvSRAM_RTC_REG_CENTURY,rtc_data,8);
```

```
// NVRAM RTC Read : Reset Read Bit
      rtc_send_data = 0x00;
      CyDelay(1);
    NVRAM_SPI_1_RTC_Write(nvSRAM_RTC_REG_FLAGS, &rtc_send_data, 1);
#endif

    // NVRAM Serial Number Read : Read 8 bytes of serial number to serial_number array
      CyDelay(1);
    NVRAM_SPI_1_Serial_No_Read(serial_number);

    // NVRAM Product ID Read : Read 4 bytes of device id to device_id array
      CyDelay(1);
      NVRAM_SPI_1_Device_ID_Read(device_id);

    // NVRAM Write : Write 4 bytes from array data_bytes to NVRAM location 0x123456
      CyDelay(1);
    NVRAM_SPI_1_Write(0x123456, &data_bytes[5], 4);

    // NVRAM SRAM Read : Read 4 bytes of data from SRAM starting from address 0x123456.
    //                   Data will be stored into nvram_rd_data_before_recall array.
      CyDelay(1);
      NVRAM_SPI_1_Read(0x123456,nvram_rd_data_before_recall,4);

#if (NVRAM_SPI_1_NVRAM_part == 1)
    // nvSRAM Command :  Send SoftRecall Command
      CyDelay(1);
      NVRAM_SPI_1_nvCommand(nvSRAM_RECALL_CMD);
#endif

    // NVRAM SRAM Read : Read 4 bytes of data from SRAM starting from address 0x123456.
    //                   Data will be stored into nvram_rd_data_after_recall array.
      CyDelay(1);
      NVRAM_SPI_1_Read(0x123456,nvram_rd_data_after_recall,4);

    // 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 nvSRAM with PSoC3	Initial Version
2.0	Added FRAM support Renamed component name from nvSRAM to nvRAM. Added configuration options to select nvSRAM / FRAM, Density and RTC/non-RTC	Changes made for FRAM support. No user impact for the existing nvSRAM APIs. For FRAM, existing APIs are updated and new APIs are added.

© Cypress Semiconductor Corporation, 2010. 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.