ABSTRACT
This application note describes how C2000™ I2C can be used to communicate with EEPROM using both polling method (or) Interrupt method. This document implements different EEPROM write protocols such as Byte Write, Word Write, Paged Write operations and different EEPROM read operation such as Byte Read, Word Read, Paged Read operations.

EEPROM used for this example example is AT24C256 (which has write cycle time of 6ms and paged operation of 64 bytes)

I2C EEPROM example code is available in below path:
Polling method example: <C2000Ware>\driverlib\<device>\examples\c28x\i2c\i2c_ex4_eeprom_polling
Interrupt method example: <C2000Ware>\driverlib\<device>\examples\c28x\i2c\i2c_ex6_eeprom_interrupt

Table of Contents
1 Introduction.............................................................................................................................................................................2
2 Hardware Connection............................................................................................................................................................2
3 C2000 I2C Source Code..........................................................................................................................................................3
3.1 I2CHandle Description.......................................................................................................................................................3
3.2 I2CBusScan...........................................................................................................................................................................3
3.3 I2C_MasterTransmitter.......................................................................................................................................................4
3.4 I2C_MasterReceiver...........................................................................................................................................................4
4 EEPROM Byte Write ...................................................................................................................................................................5
5 EEPROM Byte Read ..............................................................................................................................................................6
6 EEPROM Word Write ............................................................................................................................................................7
7 EEPROM Word Read ............................................................................................................................................................8
8 EEPROM Paged Write ...........................................................................................................................................................9
9 EEPROM Paged Read ..........................................................................................................................................................10

List of Figures
Figure 4-1. EEPROM Byte Write command.................................................................................................................................5
Figure 5-1. EEPROM Byte Read command..................................................................................................................................6
Figure 6-1. EEPROM Word Write Command................................................................................................................................7
Figure 7-1. EEPROM Word Read Command................................................................................................................................8
Figure 8-1. EEPROM Paged Write Command...............................................................................................................................9
Figure 9-1. EEPROM Paged Read Command.............................................................................................................................10

List of Tables
Table 3-1. Project Filename and Descriptions....................................................................................................................................3
Table 3-2. I2C Interrupts Used in i2c_ex6_eeprom_interrupt Example..........................................................................................3
Table 3-3. Details of I2CHandle.....................................................................................................................................................3

Trademarks
C2000™ is a trademark of Texas Instruments.
All trademarks are the property of their respective owners.
1 Introduction

The C28x-I2C module used in the application note has the following features:

- Compliance with the NXP Semiconductors I2C bus specification (version 2.1):
  - Support for 8-bit format transfers
  - 7-bit and 10-bit addressing modes
  - General call
  - START byte mode
  - Support for multiple master-transmitters and slave-receivers
  - Support for multiple slave-transmitters and master-receivers
  - Combined master transmit/receive and receive/transmit mode
  - Data transfer rate from 10 kbps up to 400 kbps (Fast-mode)
- Receive FIFO and Transmitter FIFO (16-deep x 8-bit FIFO)
- Supports two ePIE interrupts:
  - I2Cx Interrupt – Any of the below events can be configured to generate an I2Cx interrupt:
    - Transmit-data ready
    - Receive-data ready
    - Register-access ready
    - No-acknowledgment received
    - Arbitration lost
    - Stop condition detected
    - Addressed as slave
  - I2Cx_FIFO interrupts:
    - Transmit FIFO interrupt
    - Receive FIFO interrupt
- Module enable/disable capability
- Free data format mode

2 Hardware Connection

The below schematics shows how an EEPROM device can be connected to C2000 I2C module. EEPROM used in this application report is AT24C256. In AT24C256, user configurable pins (A0, A1) called device address pins can be used to address as many as four AT24C256 devices on the same I2C bus. These A0, A1 pins are pulled down which makes the slave address of EEPROM = 0x50. The write protect input pin needs to be connected to ground to allow EEPROM write operation.

For information regarding selection of pull resistor, see the I2C Bus Pull-Up Resistor Calculation.
3 C2000 I2C Source Code

The C2000Ware software example provided in Table 3-1 shows how to use I2C module to communicate with EEPROM via I2C bus. This example has been developed for EEPROM AT24C256, which requires 2 bytes for addressing the EEPROM memory with slave address of 0x50. Table 3-2 shows the I2C interrupts used in EEPROM interrupt based example.

<table>
<thead>
<tr>
<th>Source Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_ex4_eeprom_polling.c</td>
<td>This program will show how to perform different EEPROM write and read commands using I2C polling method</td>
</tr>
<tr>
<td>l2cLib_FIFO_polling.c</td>
<td>C28x-I2C Library source file for FIFO using polling</td>
</tr>
<tr>
<td>l2cLib_FIFO_polling.h</td>
<td>C28x-I2C Library header file for FIFO using polling</td>
</tr>
<tr>
<td>i2c_ex6_eeprom_interrupt.c</td>
<td>This program will show how to perform different EEPROM write and read commands using I2C interrupt method</td>
</tr>
<tr>
<td>l2cLib_FIFO_master_interrupt.c</td>
<td>C28x-I2C Library source file for FIFO interrupts</td>
</tr>
<tr>
<td>l2cLib_FIFO_master_interrupt.h</td>
<td>C28x-I2C Library header file for FIFO interrupts</td>
</tr>
</tbody>
</table>

| STOP condition | Register Access Ready |
| Addressed as slave | TX FIFO interrupt |
| Arbitration lost | RX FIFO interrupt |
| NACK condition | |

3.1 I2CHandle Description

Table 3-3 provides the details of I2CHandle structure defined in the example code.

<table>
<thead>
<tr>
<th>Structure Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>Base address of I2C module used</td>
</tr>
<tr>
<td>SlaveAddr</td>
<td>Slave address of EEPROM</td>
</tr>
<tr>
<td>pControlAddr</td>
<td>Pointer to variable which stores EEPROM address</td>
</tr>
<tr>
<td>NumOfAddrBytes</td>
<td>Number of address byte required by EEPROM</td>
</tr>
<tr>
<td>pTX_MsgBuffer</td>
<td>Pointer to TX message buffer</td>
</tr>
<tr>
<td>pRX_MsgBuffer</td>
<td>Pointer to RX message buffer</td>
</tr>
<tr>
<td>NumOfDataBytes</td>
<td>Number of data bytes in a I2C transaction</td>
</tr>
<tr>
<td>currentHandlePtr</td>
<td>Pointer to I2CHandle object</td>
</tr>
<tr>
<td>numofSixteenByte</td>
<td>Number of 16 bytes in data packet</td>
</tr>
<tr>
<td>remainingbytes</td>
<td>Number of bytes which are less than 16 bytes</td>
</tr>
<tr>
<td>WriteCycleTime_in_us</td>
<td>Write cycle time (in us) defined based on EEPROM</td>
</tr>
</tbody>
</table>

3.2 I2CBusScan

Function name: I2CBusScan(I2CA_BASE, pAvailableI2C_slaves)

Description: This function can be used to scans for all I2C device connected to I2C bus by sending different slave addresses and checking for ACK / NACK condition

Arguments:

base : base is the base address of the I2C instance used
pAvailableI2C_slaves : Address of the array that stores the I2C addresses available on the I2C bus
### 3.3 I2C_MasterTransmitter

**Function name:** I2C_MasterTransmitter(I2CA_BASE, struct I2CHandle *pI2C_Params)

**Description:** This function can be used to transmit 'N' number of bytes defined in pI2C_Params. This function can be used to perform following EEPROM write operations.

1. EEPROM byte write
   - Set EEPROM.NumOfDataBytes = 1, because one byte needs to be transmitted
2. EEPROM word write
   - Set EEPROM.NumOfDataBytes = 2, because two bytes needs to be transmitted
3. EEPROM Paged write
   - Set EEPROM.NumOfDataBytes = 'N', because 'N' bytes needs to be transmitted - Page size depends upon EEPROM chosen

Below setting applies to all the above EEPROM write operations

Set EEPROM.NumOfAddrBytes = 2, because high address byte and low address byte needs to be transmitted

**Arguments:**
- base : base is the base address of the I2C instance used
- pI2C_Params : Pointer to I2C address handle of the slave

**Return:** Status of I2C transaction

### 3.4 I2C_MasterReceiver

**Function name:** I2C_MasterReceiver(I2CA_BASE, struct I2CHandle *pI2C_Params)

**Description:** This function can be used to receive 'N' number of bytes defined in pI2C_Params. This function can also be used to perform the following EEPROM read operations.

1. EEPROM byte read
   - Set Number of bytes to be read (EEPROM.NumOfDataBytes = 1)
2. EEPROM word read
   - Set Number of bytes to be read (EEPROM.NumOfDataBytes = 2) - Two bytes make a word
3. EEPROM Paged read
   - Set Number of bytes to be read (EEPROM.NumOfDataBytes = 'N') - There is no page size restriction for paged read.

Below setting applies to all the above EEPROM read operations

Set EEPROM.NumOfAddrBytes = 2, because high address byte and low address byte needs to be transmitted

**Arguments:**
- base : base is the base address of the I2C instance used
- pI2C_Params : Pointer to I2C address handle of the slave

**Return:** Status of I2C transaction
4 EEPROM Byte Write

Figure 4-1 shows how EEPROM Byte Write protocol is defined in AT24C256. C2000 I2C is configured in Master Transmitter mode (I2CMDR.MST = 1, I2CMDR.TRX = 1) to transmit EEPROM address (both High address byte, Low address byte) followed by data byte.

![Figure 4-1. EEPROM Byte Write command](image)

Code flow:
1. START condition + Transmit Slave address (0x50) + Write bit + ACK bit (from slave)
2. Transmit EEPROM high address byte + ACK bit (from slave)
3. Transmit EEPROM low address byte + ACK bit (from slave)
4. Transmit data byte + ACK bit (from slave)
5. Generate STOP condition

```c
EEPROM_SlaveAddr = 0x50;
EEPROM_base = I2CA_BASE;
EEPROM.pControlAddr = &ControlAddr;
EEPROM.NumOfAddrBytes = 2;
EEPROM.pTX_MsgBuffer = TX_MsgBuffer;
EEPROM.pRX_MsgBuffer = RX_MsgBuffer;
EEPROM.NumOfAttempts = 5;
EEPROM.Delay_us = 10;
EEPROM.WriteCycleTime_in_us = 6000;

// Example 1: EEPROM Byte Write
// Write 11 to EEPROM address 0x0
ControlAddr = 0;
EEPROM.NumOfDataBytes = 1;
TX_MsgBuffer[8] = 11;
status = I2C_MasterTransmitter(&EEPROM);

// Wait for EEPROM write cycle time
// This delay is not mandatory. User can run their application code
// It is however important to wait for EEPROM write cycle time before another read / write transaction
DEVICE_DELAY_US(EEPROM.WriteCycleTime_in_us);
```

![EEPROM Data Transfer](image)

---

*Interfacing EEPROM Using C2000 I2C Module*

Copyright © 2022 Texas Instruments Incorporated
5 EEPROM Byte Read

Figure 5-1 shows how EEPROM Byte Read protocol is defined in AT24C256. C2000 I2C is configured in Master Transmitter mode (I2CMDR.MST = 1, I2CMDR.TRX = 1) to transmit EEPROM address (both High address byte, Low address byte) and then C2000 I2C generates Repeated START condition in Master Receiver (I2CMDR.MST = 1, I2CMDR.TRX = 0) mode to receive a data byte from EEPROM.

Code flow:
1. START condition + Transmit Slave address (0x50) + Write bit + ACK bit (from slave)
2. Transmit EEPROM high address byte + ACK bit (from slave)
3. Transmit EEPROM low address byte + ACK bit (from slave)
4. Repeated START condition + Transmit Slave address (0x50) + Read bit + ACK bit (from slave)
5. Receive data byte + ACK bit (from master)
6. Generate STOP condition

```c
//Example 2: EEPROM Byte Read
//Make sure 11 is written to EEPROM address 0x0
ControlAddr = 0;
EEPROM.pControlAddr = &ControlAddr;
EEPROM.NumOfDataBytes = 1;
status = IIC_MasterReceiver(&EEPROM);
while(IIC_getStatus(EEPROM.base) & IIC_STS_BUS_BUSY);
```

![Figure 5-1. EEPROM Byte Read command](image-url)
6 EEPROM Word Write

Figure 6-1 shows how EEPROM Word Write protocol is defined in AT24C256. C2000 I2C is configured in Master Transmitter mode (I2CMDR.MST = 1, I2CMDR.TRX = 1) to transmit EEPROM address (both High address byte, Low address byte) followed by two data bytes.

<table>
<thead>
<tr>
<th>S</th>
<th>Slave address</th>
<th>W</th>
<th>ACK</th>
<th>A</th>
<th>High address byte</th>
<th>A</th>
<th>ACK</th>
<th>A</th>
<th>Low address byte</th>
<th>A</th>
<th>ACK</th>
<th>A</th>
<th>Data byte #1</th>
<th>A</th>
<th>ACK</th>
<th>A</th>
<th>Data byte #2</th>
<th>A</th>
<th>ACK</th>
<th>P</th>
</tr>
</thead>
</table>

**Figure 6-1. EEPROM Word Write Command**

Code flow:

1. START condition + Transmit Slave address (0x50) + Write bit + ACK bit (from slave)
2. Transmit EEPROM high address byte + ACK bit (from slave)
3. Transmit EEPROM low address byte + ACK bit (from slave)
4. Transmit data byte # 1 + ACK bit (from slave)
5. Transmit data byte # 2 + ACK bit (from slave)
6. Generate STOP condition

```c
//Example 3: EEPROM word (16-bit) write
//EEPROM address 0x1 = 22 & 0x2 = 33
ControlAddr = 1; //EEPROM address to write
EEPROM.NumOfDataBytes = 2;
TX_MsgBuffer[0] = 0x11;
TX_MsgBuffer[1] = 0x22;
EEPROM.pTX_MsgBuffer = TX_MsgBuffer;
status = I2C_MasterTransmitter(&EEPROM);

//Wait for EEPROM write cycle time
//This delay is not mandatory. User can run their application code instead.
//It is however important to wait for EEPROM write cycle time before you initiate
//another read / write transaction
DEVICE_DELAY_US(EEPROM.WriteCycleTime_in_us);
```

[Diagram showing the EEPROM Word Write command flow]
7 EEPROM Word Read

Figure 7-1 shows how EEPROM Word Read protocol is defined in AT24C256. C2000 I2C is configured in Master Transmitter mode (I2CMDR.MST = 1, I2CMDR.TRX = 1) to transmit EEPROM address (both High address byte, Low address byte) and then C2000 I2C generates Repeated START condition in Master Receiver (I2CMDR.MST = 1, I2CMDR.TRX = 0) mode to receive two data bytes from EEPROM.

<table>
<thead>
<tr>
<th>S</th>
<th>Slave address</th>
<th>W</th>
<th>ACK</th>
<th>High address byte</th>
<th>A</th>
<th>ACK</th>
<th>Low address byte</th>
<th>A</th>
<th>ACK</th>
<th>R</th>
<th>S</th>
<th>Slave address</th>
<th>R</th>
<th>A</th>
<th>ACK</th>
<th>Data byte #1</th>
<th>A</th>
<th>ACK</th>
<th>Data byte #2</th>
<th>N</th>
<th>A</th>
<th>P</th>
</tr>
</thead>
</table>

Figure 7-1. EEPROM Word Read Command

Code flow:

1. START condition + Transmit Slave address (0x50) + Write bit + ACK bit (from slave)
2. Transmit EEPROM high address byte + ACK bit (from slave)
3. Transmit EEPROM low address byte + ACK bit (from slave)
4. Generate Repeated START condition + Transmit Slave address (0x50) + Read bit + ACK bit (from slave)
5. Receive data byte # 1 + ACK bit (from master)
6. Receive data byte # 2 + NACK bit (from master)
7. Generate STOP condition

```c
//Example 4: EEPROM word (16-bit) read
//Make sure EEPROM address 1 has 0x11 and 2 has 0x22
ControlAddr = 1;
EEPROM.pControlAddr = &ControlAddr;
EEPROM.pRX_MsgBuffer = RX_MsgBuffer;
EEPROM.NumOfDataBytes = 2;

status = I2C_MasterReceiver(EEPROM);
```

---

8 Interfacing EEPROM Using C2000 I2C Module

Submit Document Feedback

Copyright © 2022 Texas Instruments Incorporated
8 EEPROM Paged Write

Figure 8-1 shows how EEPROM Paged Write protocol is defined in AT24C256. C2000 I2C is configured in Master Transmitter mode (I2CMDR.MST = 1, I2CMDR.TRX = 1) to transmit EEPROM address (both High address byte, Low address byte) followed by 'N' data bytes.

<table>
<thead>
<tr>
<th>S</th>
<th>Slave address</th>
<th>W</th>
<th>High address byte</th>
<th>A</th>
<th>C</th>
<th>K</th>
<th>Low address byte</th>
<th>A</th>
<th>C</th>
<th>K</th>
<th>Data byte #1</th>
<th>A</th>
<th>C</th>
<th>K</th>
<th>Data byte #2</th>
<th>A</th>
<th>C</th>
<th>K</th>
<th>Data byte #N</th>
<th>A</th>
<th>C</th>
<th>K</th>
<th>P</th>
</tr>
</thead>
</table>

**Figure 8-1. EEPROM Paged Write Command**

Code flow:

1. START condition + Transmit Slave address (0x50) + Write bit + ACK bit (from slave)
2. Transmit EEPROM high address byte + ACK bit (from slave)
3. Transmit EEPROM low address byte + ACK bit (from slave)
4. Transmit data byte # 1 + ACK bit (from slave)
5. Transmit data byte # 2 + ACK bit (from slave)
6. Transmit more bytes
7. Transmit data byte # N + ACK bit (from slave)
8. Generate STOP condition

```c
//Example 5: EEPROM Page write
//Program address = data pattern from address 64

for(i=0;i<MAX_BUFFER_SIZE;i++)
{
    TX_MsgBuffer[i] = i+64;
}

ControlAddr = 64;  //EEPROM address to write
EEPROM.pNumOfDataBytes = MAX_BUFFER_SIZE;
EEPROM.pTX_MsgBuffer = TX_MsgBuffer;
status = I2C_MasterTransmitter(EEPROM);

//Wait for EEPROM write cycle time
//This delay is not mandatory. User can run their application code instead.
//It is however important to wait for EEPROM write cycle time before you initiate
//another read / write transaction

DEVICE_DELAY_us(EEPROM.WriteCycleTime_in_us);
```
9 EEPROM Paged Read

Figure 9-1 shows how EEPROM Paged Read protocol is defined in AT24C256. C2000 I2C is configured in Master Transmitter mode (I2CMDR.MST = 1, I2CMDR.TRX = 1) to transmit EEPROM address (both High address byte, Low address byte) and then C2000 I2C generates Repeated START condition in Master Receiver (I2CMDR.MST = 1, I2CMDR.TRX = 0) mode to receive 'N' data bytes from EEPROM.

<table>
<thead>
<tr>
<th>S</th>
<th>Slave address</th>
<th>W</th>
<th>A</th>
<th>C</th>
<th>K</th>
<th>R</th>
<th>S</th>
<th>Slave address</th>
<th>R</th>
<th>Data byte #1</th>
<th>A</th>
<th>C</th>
<th>K</th>
<th>Data byte #2</th>
<th>......</th>
<th>A</th>
<th>C</th>
<th>K</th>
<th>Data byte #N</th>
</tr>
</thead>
</table>

**Figure 9-1. EEPROM Paged Read Command**

Code flow:

1. START condition + Transmit Slave address (0x50) + Write bit + ACK bit (from slave)
2. Transmit EEPROM high address byte + ACK bit (from slave)
3. Transmit EEPROM low address byte + ACK bit (from slave)
4. Generate Repeated START condition + Transmit Slave address (0x50) + Read bit + ACK bit (from slave)
5. Receive data byte # 1 + ACK bit (from master)
6. Receive data byte # 2 + NACK bit (from master)
   
   o o o (Receive more bytes)
7. Receive data byte # N + ACK bit (from master)
8. Generate STOP condition

```c
//Example 6: EEPROM word Paged read
ControlAddr = 64;
EEPROM.pControlAddr = &controlAddr;
EEPROM.pRX_HagBuffer = RX_HagBuffer;
EEPROM.NumOfDataBytes = MAX_BUFFER_SIZE;

status = I2C_MasterReceiver(&EEPROM);
```

---

Interfacing EEPROM Using C2000 I2C Module

SPRCS8 – MAY 2022
Submit Document Feedback

Copyright © 2022 Texas Instruments Incorporated
IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2022, Texas Instruments Incorporated