ANALOG DEVICES

AN-1367
APPLICATION NOTE

One Technology Way • P.O. Box 9106 • Norwood, MA 02062-9106, U.S.A. • Tel: 781.329.4700 • Fax: 781.461.3113 • www.analog.com

I^{2}C Interface Between the ADE7953 and the ADuCM360

by Fermi Lim, Daniel Kim, and Hariharan Mani

INTRODUCTION

This application note demonstrates the implementation of an I^{2}C interface between the ADE7953 (the slave) and the ADuCM360 (the master), using C programming language. The ADE7953 is a single-phase energy metering IC, and the ADuCM360 is an ARM® Cortex®-M3-based microcontroller. The ADE7953 contains registers that are 8, 16, 24, or 32 bits long. While writing the source code, it is important to make sure that the data size for the read/write operation is identified based on the address of the register. The register list in the ADE7953 data sheet contains all the relevant information.

This application note introduces the ARM Cortex-M3 core and the ADuCM360 microcontroller unit (MCU), followed by the initialization steps that need to be completed within the ADuCM360, and finally the implementation of the I^{2}C interface between the ADE7953 and the ADuCM360. The sample codes described in this application note can also be used for other Analog Devices, Inc., processors that are based on the ARM Cortex-M core.

The complete source code for establishing the I^{2}C interface is available as a downloadable file, AN_1367_I2C_interface.zip, at www.analog.com/ADE7953 or www.analogcom/ADuCM360.

This application note describes how to implement the source code in the Visual C++ 2012 integrated development environment (IDE), and provides useful insight regarding the I^{2}C communication ports of the ADE7953 and the ADuCM360.

To validate the source code, the following equipment and software were used:

- ADE7953 and ADuCM360 evaluation module boards (EVMs)
- Software development environment: Keil MDK-ARM Version 4.72
- Terminal emulator: Tera Term Version 4.79
- IDE: Visual C++ 2012
- Laptop personal computer (PC)

The entire lab setup is shown in Figure 1. The two evaluation boards are connected with the help of wires, as shown in Figure 2. Because an isolation interface was not considered for this setup, both boards are powered by a common supply voltage and share a common ground. If isolating the I^{2}C interface, the propagation delay of the isolators must be taken into account.

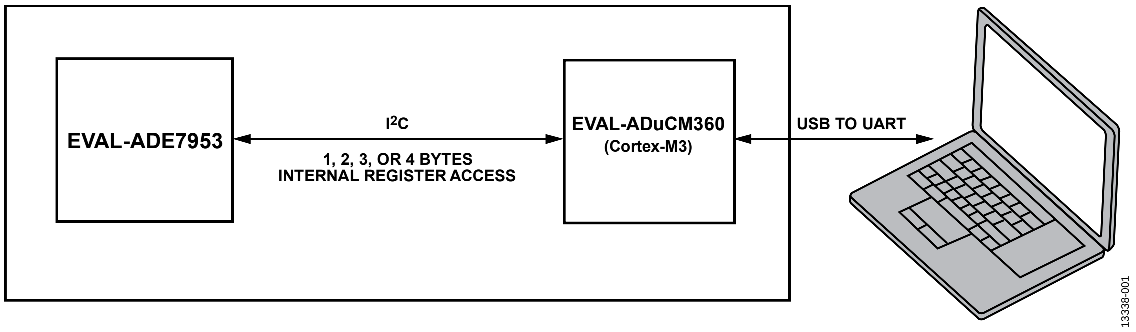


Figure 1. Lab Setup

TABLE OF CONTENTS

REVISION HISTORY

9/15—Revision 0: Initial Version

SETUP DESCRIPTION

The terminal emulator software, Tera Term Version 4.79, is used to read from or write to any of the ADE7953 registers through the ADuCM360. The universal asynchronous receiver/transmitter (UART) communication interface is used between the ADuCM360 and the PC to send and receive the read/write commands and data. The Tera Term Version 4.79 software, including examples of complete source code for the UART interface-based communication, is available for free from various sources, including the OSDN Corporation.

Figure 3 shows the terminal window where the get and set commands of the example code are used to access the ADE7953 registers. The syntax for the two commands are

- Set REG_ADDR value
  The set command writes the register located at Address REG_ADDR with a specific value.
- Get START_REG_ADDR REGISTER_NUMBER
  The get command reads the contents of the registers starting from Address START_REG_ADDR continuously starting from REGISTER_NUMBER. The number of register locations to be read is determined by REGISTER_NUMBER.

The uniqueness of the source code is that it allows the PC to communicate with the ADE7953 even without the knowledge of the data size of each of the registers, because the register lookup table is already implemented.

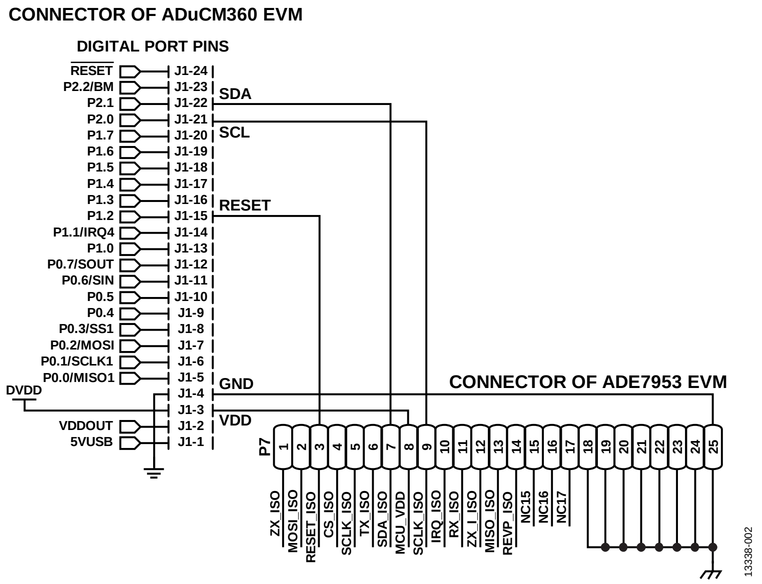
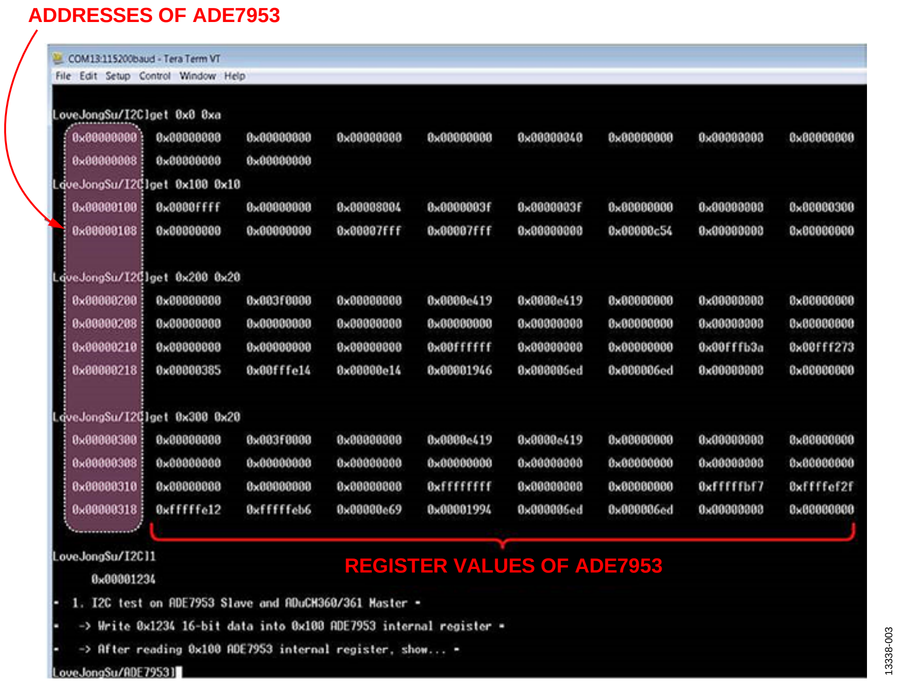
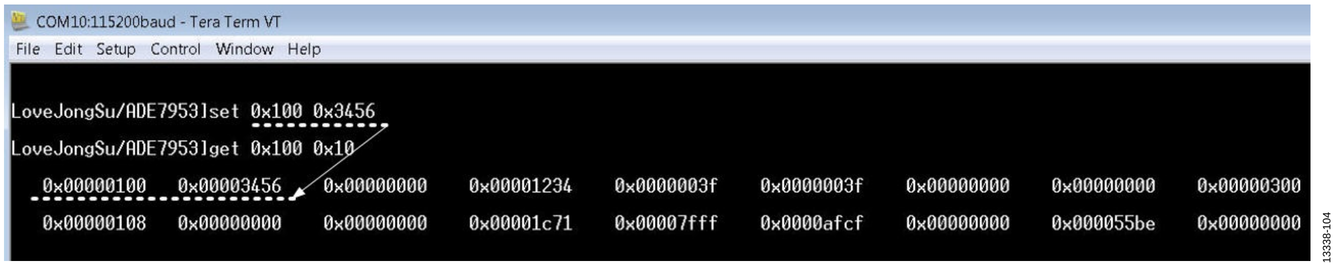


Figure 2. Connection Between ADuCM360 EVM and ADE7953 EVM


ADDRESSES OF ADE7953
COM13:115200baud - Tera Term VT
File Edit Setup Control Window Help
REGISTER VALUES OF ADE7953
13338-003

```
LoveJongSu/I2C]get 0x0 0xa
   0x00000000   0x00000000   0x00000000   0x00000000   0x00000000   0x00000040   0x00000000   0x00000000   0x00000000
   0x00000008   0x00000000   0x00000000
LoveJongSu/I2C]get 0x100 0x10
   0x00000100   0x0000ffff   0x00000000   0x00008004   0x0000003f   0x0000003f   0x00000000   0x00000000   0x00000300
   0x00000108   0x00000000   0x00000000   0x00007fff   0x00007fff   0x00000000   0x00000c54   0x00000000   0x00000000

LoveJongSu/I2C]get 0x200 0x20
   0x00000200   0x00000000   0x003f0000   0x00000000   0x0000e419   0x0000e419   0x00000000   0x00000000   0x00000000
   0x00000208   0x00000000   0x00000000   0x00000000   0x00000000   0x00000000   0x00000000   0x00000000   0x00000000
   0x00000210   0x00000000   0x00000000   0x00000000   0x00ffffff   0x00000000   0x00000000   0x00fffb3a   0x00fff273
   0x00000218   0x00000385   0x00fffe14   0x00000e14   0x00001946   0x000006ed   0x000006ed   0x00000000   0x00000000

LoveJongSu/I2C]get 0x300 0x20
   0x00000300   0x00000000   0x003f0000   0x00000000   0x0000e419   0x0000e419   0x00000000   0x00000000   0x00000000
   0x00000308   0x00000000   0x00000000   0x00000000   0x00000000   0x00000000   0x00000000   0x00000000   0x00000000
   0x00000310   0x00000000   0x00000000   0x00000000   0xffffffff   0x00000000   0x00000000   0xfffffbf7   0xffffef2f
   0x00000318   0xfffffe12   0xfffffeb6   0x00000e69   0x00001994   0x000006ed   0x000006ed   0x00000000   0x00000000

LoveJongSu/I2C]1
     0x00001234
- 1. I2C test on ADE7953 Slave and ADuCM360/361 Master -
-  -> Write 0x1234 16-bit data into 0x100 ADE7953 internal register -
-  -> After reading 0x100 ADE7953 internal register, show... -
LoveJongSu/ADE7953]
```

Figure 3. Terminal Window Showing the Use of Get Command


COM10:115200baud - Tera Term VT
File Edit Setup Control Window Help
13338-104

```
LoveJongSu/ADE7953]set 0x100 0x3456
LoveJongSu/ADE7953]get 0x100 0x10
   0x00000100   0x00003456   0x00000000   0x00001234   0x0000003f   0x0000003f   0x00000000   0x00000000   0x00000300
   0x00000108   0x00000000   0x00000000   0x00001c71   0x00007fff   0x0000afcf   0x00000000   0x000055be   0x00000000
```

Figure 4. Terminal Window Showing the Use of Get and Set Commands

ARM CORTEX-M3 CORE AND ADUCM360 MCU

The ARM Cortex-M3 core is based on the ARMv7-M architecture. The Cortex-M family is used primarily in the embedded market, in applications requiring a core clock frequency of less than 300 MHz. Most of the embedded applications such as ac motor control and other simple home appliances do not need any sophisticated features such as a memory management unit (MMU) and NEON™, thus making the Cortex-M family a good choice. The architecture of the Cortex-M core is simpler than other Cortex families.

Cortex-M family of cores include the following unique features:

- A fixed memory map
- A common nested vectored interrupt controller (NVIC)
- Thumb®-2 instruction set support only

The code written for one Cortex-M family MCU can be easily modified to use with another Cortex-M family MCU, even if it is from another vendor. Furthermore, ARM provides useful Cortex Microcontroller Software Interface Standard (CMSIS) software for various applications, for free.

Figure 5 compares a fixed memory map of the Cortex-M family core with that of the ADuCM360.

As shown in Figure 5, the ADuCM360 includes 8 kB SRAM and 128 kB Flash/EE memory based on a fixed memory map for the Cortex-M family. All of the ADuCM360 peripherals are mapped into the 0x40000000 to 0x4004FFFF address range. The address range of 0xE000E000 to 0xE000EE000 is for the Cortex-M core memory mapped registers (MMRs); the 0x40000000 to 0x4004FFFF address range is for the ADuCM360 MMRs.

Note the following regarding the NVIC:

- The Cortex-M family of cores has only one core interrupt, whereas older ARM cores have a fast interrupt as well as a normal interrupt.
- Two sets of registers are available: interrupt set enable registers (ISERs) and interrupt clear enable registers (ICERs).

Usually, peripherals such as UART and I²C have their own registers to enable/disable the interrupts. Only one of the interrupts transferred by each peripheral can reach the core according to the ISER/ICER values. The ISERs are located in the 0xE000E100 to 0xE000E13C address range, and ICERs are located in the 0xE000E180 to 0xE000E1BC address range. When a Cortex-M family MCU is used, modify ISERs and ICERs as needed. Of all available IDE tools, Keil MDK-ARM and IAR Embedded Workbench® are the most popular. Both these tools can be used when the executable image is within the specified size limitations. The Keil MDK-ARM IDE was used in the lab setup for this application note. To use the code with the IAR Embedded Workbench IDE, the **ADuCM360StartUp.s** file, from the downloadable file, AN_1367_I2C_interface.zip, must be changed. When changing the IDE from Keil MDK-ARM to IAR Embedded Workbench, or vice versa, only the assembler directives have to be modified and not the instructions, because the assembler directives depend on the assembler being used and the instruction set depends on the MCU core.

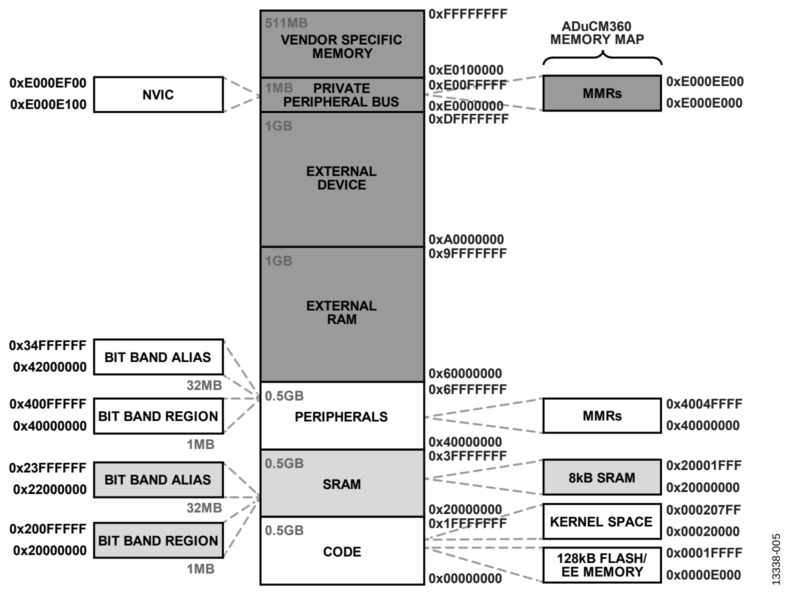


Figure 5. Memory Map Comparison Between Cortex-M and the ADuCM360

Figure 6 and Figure 7 shows the tool chains of the Keil MDK-ARM and IAR Embedded Workbench IDE, respectively. In Figure 6, armcc, armasm, and armlink are displayed in shaded gray to indicate that they form the C/C++ compiler, assembler, and linker, respectively. Similarly, in Figure 7, iccarm, iasmarm, and ilinkarm are indicated in shaded gray to denote that they are the compiler, assembler, and linker, respectively.

Figure 8 shows the architecture of the ADuCM360. Among the peripherals of the ADuCM360, only the I^{2}C, UART, and GPIOs are used to communicate with the ADE7953 and the PC.

First, configure UART to communicate with PC, and then configure the I^{2}C to interface with the ADE7953.

Though the focus of this application note is the I^{2}C interface, the sample codes provided in the downloadable file, AN_1367_I2C_interface.zip, have modules for UART interface as well as GPIO usage.

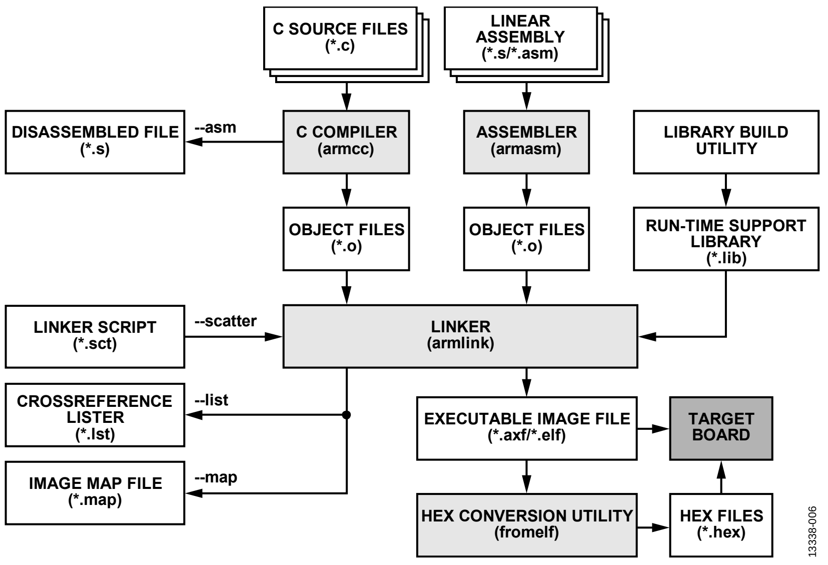


Figure 6. Keil MDK-ARM Tool Chain

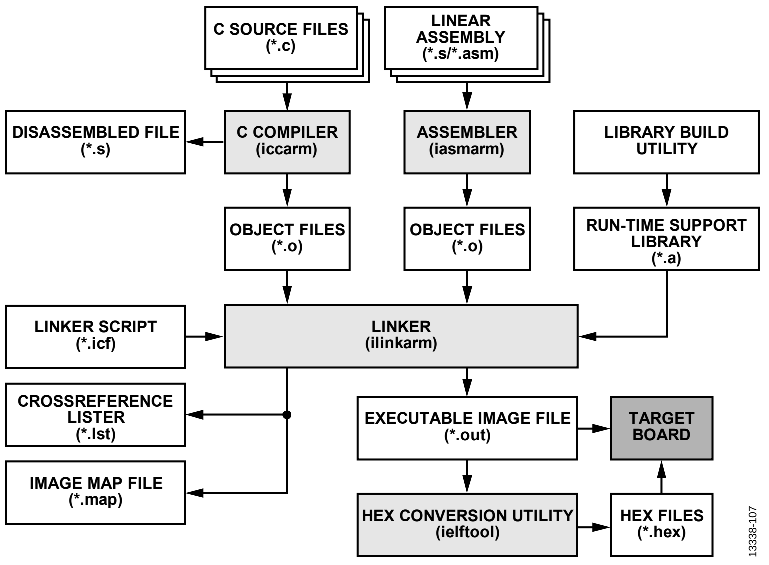


Figure 7. IAR Embedded Workbench Tool Chain

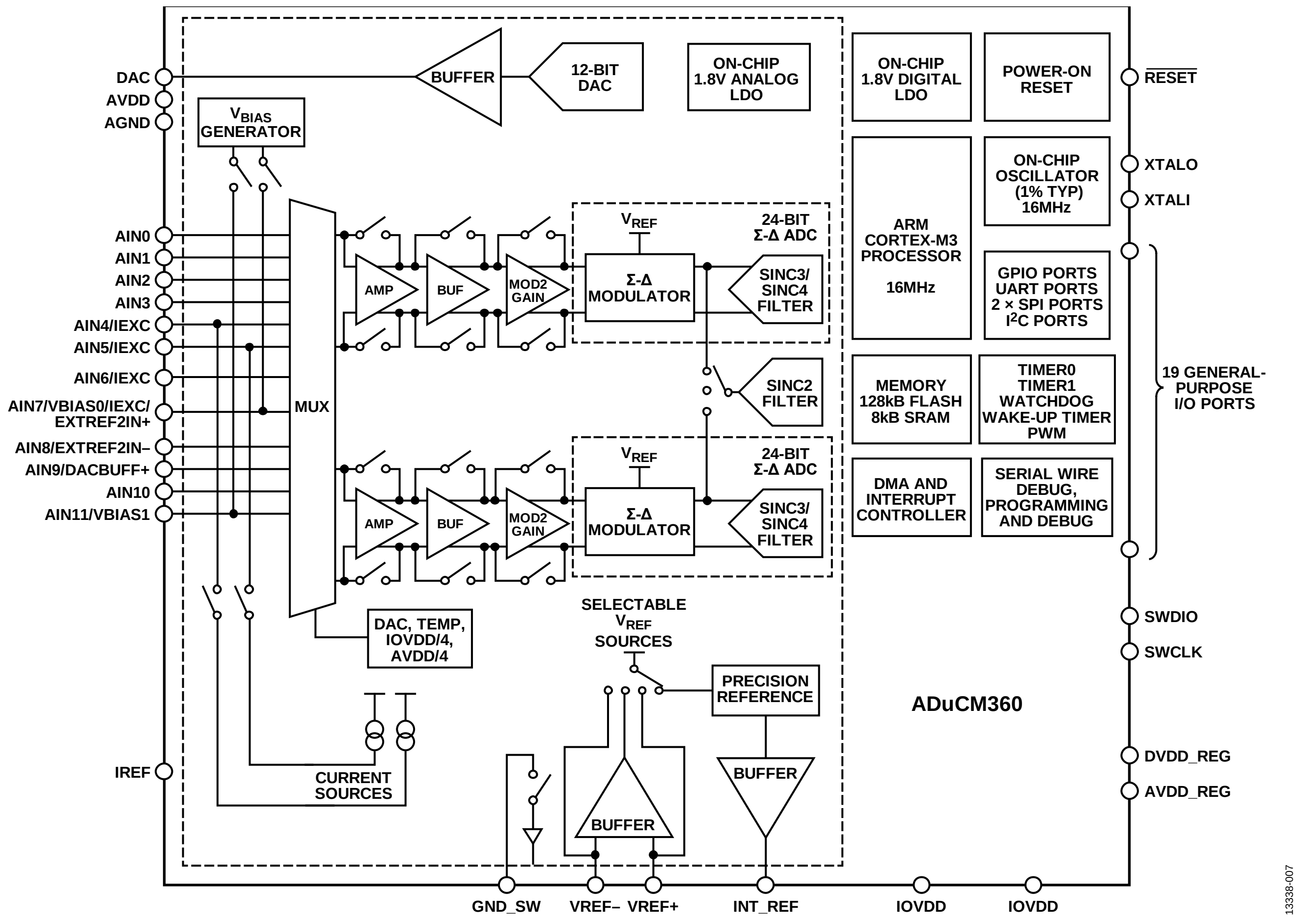


Figure 8. ADuCM360 Block Diagram

INITIALIZING THE ADUCM360 FOR UART AND I²C

When developing any embedded C code, first disable the watchdog timer, and then configure the clock for the core and peripherals as needed.

After any reset operation, the watchdog timer is enabled automatically. To disable the watchdog timer of the ADuCM360, write 0’b0 to T3CON[5], the watchdog timer enable bit.

```
// Step 1. Disable the watchdog timer.
   *pT3CON=0x0;
```

Next, configure the clocks for the UART, I²C, and the Cortex-M3 core. Figure 9 and Figure 10 show how to configure the clocks for the UART and I²C. Configure the core clock as 16 MHz, and then set the control registers of the UART and the I²C interface accordingly. When selecting the I²C system clock divide ratio via CLKCON1, ensure that the core clock frequency is less than or equal to that of the I²C system clock, that is, HCLK ≤ I2CCLK. This is true for all peripherals whose clock can be divided by CLKCON1. If the peripheral clock is slower, the clock to the peripheral is gated and the peripheral does not work. For more information, refer to the ADuCM360/ADuCM361 Hardware User Guide.

As shown in Figure 9, the UART interface with the PC is set up with a baud rate of 115,200 bps. Therefore, configure the Tera Term tool accordingly as well. Refer to the project file of Tera Term named ‘MyTera_Setup2014.INI’, which is available from the downloadable file, AN_1367_I2C_interface.zip.

To use the Tera Term project file, select the **Restore setup** option from the **Setup** menu, as shown in Figure 11. Next, select the **MyTera_Setup2014.INI** project file from the location saved on the PC.

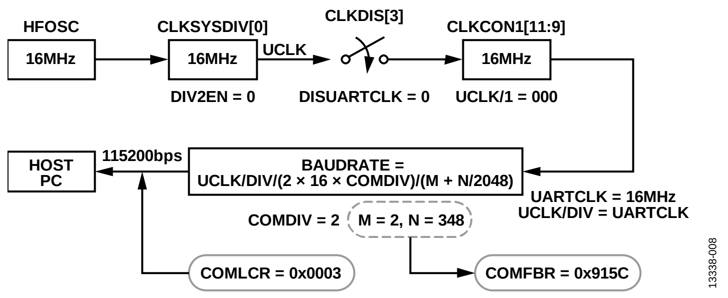


Figure 9. Clock Configuration for UART

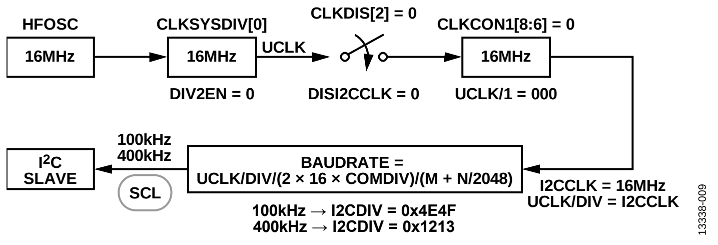


Figure 10. Clock Configuration for I²C

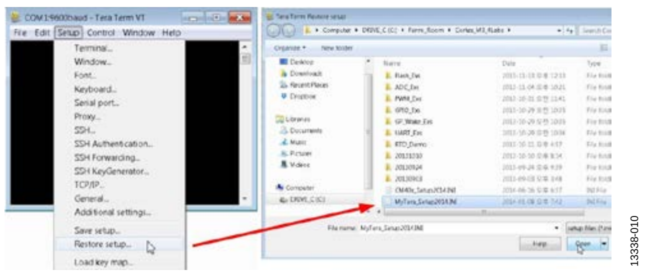


Figure 11. Tera Term Configuration

The following C code instructions, which are extracted from the **ADuCM360_I2C_AppNote.c** file (from the downloadable file, AN_1367_I2C_interface.zip), show how to configure the UART interface. All the variables in the code denote MMRs related to the clock, GPIOs, and UART.

```
// Step 2. Configure clocks.
// UART speed rate is 115,200 bps:
UCLK/DIV = 16 MHz.
*pCLKDIS=0x03F7;
*pCLKSYSDIV=0x00;
*pCLKCON0=0x0000;
*pCLKCON1=0x0000;
// Step 3. Set up the behavior of the
chosen UART.
*pCOMCON=0x00;     // UART peripheral is
enabling.
// P0.2 is UART Tx; P0.1 is UART Rx.
*pGP0CON=0x003C;
*pCOMDIV=0x0002;   // COMDIV = 2;
*pCOMFBR=0x915C;   // Set to 115,200 bps
// WordLength = 8 bits, stop bit = 1 bit,
no parity check.
*pCOMLCR=0x0003;
// COMTX and COMRX are enabled; COMIEN[1]
= COMIEN[0] = 1
*pCOMIEN=0x0003;
```

The interrupt enable register for the UART, COMIEN, is for UART peripheral interrupts only (see the ADuCM360/ADuCM361 Hardware User Guide for more information). For the UART transmitter (Tx)/receiver (Rx) interrupts to be transferred to Cortex-M3 core, set the corresponding ISER, as follows:

```
// ISER = 0xE000E100; enabling UART
interrupt.
write_reg(0xE000E100,0x00020000);
```

It is important to differentiate between an interrupt and an exception. An exception can be classified as synchronous or asynchronous. Any system fault can generally be called a synchronous exception, whereas an interrupt is an asynchronous exception. The UART is located in the 17th position in the interrupt vector table of the ADuCM360. Therefore, to enable the UART interrupt, set the 0xE000E100 address location of the ISER as 0x00020000. Similarly, to enable the I^{2}C interrupts, set the ISER as follows:

```
OldRegVal=read_reg(0xE000E100);
RegVal = OldRegVal | 0x00200000; // for
I2C
write_reg(0xE000E100,RegVal);  //ISER =
0xE000E100;
```

Next, initialize the I^{2}C peripheral of the ADuCM360. The ADE7953 is always the slave and, therefore, the ADuCM360 must be configured as the master.

As a master, the I^{2}C modes available in the ADuCM360 include

- Standard mode: 100 kHz
  First set CLKSYSDIV[0] = 0, CLKCON1[8:6] = 000 and then then set I2CDIV[15:8] = 0x4E, I2CDIV[7:0] = 0x4F
- Fast mode: 400 kHz
  First set CLKSYSDIV[0] = 0, CLKCON1[8:6] = 000 and then set I2CDIV[15:8] = 0x12, I2CDIV[7:0] = 0x13

The generation method of SCL in standard mode is shown in Figure 12.

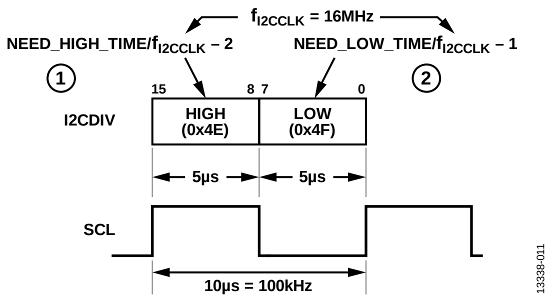


Figure 12. Generation Method of Standard 100 kHz SCL

As shown in Figure 10 and Figure 12, f_{I2CCLK} is 16 MHz, and if CLKSYSDIV[0] = 0 and CLKCON1[8:6] = 0 are initialized, the SCL bit rate, f_{SCL}, is calculated using the following formula:

$$f_{SCL} = \frac{f_{I2CCLK}}{(Low + High + 3)} [\text{Hz}]$$

where:
Low is the low period of SCL, which is set by I2CDIV[7:0].
High is the high period of SCL, which is set by I2CDIV[15:8].

In standard mode, I2CDIV[7:0] is set as 0x4F (0d'78) and I2CDIV[15:8] is set as 0x4E (0d'79). Using these values, 100 kHz is obtained as the SCL clock rate.

To use this code with Visual C++, the following piece of instructions was added:

```
#ifdef FermiEmulation_Mode
//to initialize all of the ADuCM360
register set.
    ADuCM360_RegsInit();
#endif
```

For more information about this sample code, see the I^{2}C Interface: ADuCM360 and ADE7953 section. This sample code is based on the super loop architecture. Therefore, this code style cannot handle multiple tasks. The use of menus via the Tera Term tool allows any graphic user interface (GUI) component to be implemented easily with Visual C++ or Visual Basic. Most of factory automation systems use this approach.

I^{2}C INTERFACE: ADUCM360 AND ADE7953

An I^{2}C interface is implemented using two pins, SCL and SDA. (Note that, throughout this section, multifunction pins, such as P2.0/SCL/UARTCLK, are referred to by a single function, for example, SCL, when only that function is relevant.)

- SCL: serial clock pin. Only the master can generate the I^{2}C clock; SCL can be used in either standard mode or fast mode.
- SDA: serial data pin.

Both SCL and SDA are bidirectional and must be connected to a positive supply voltage using a pull-up resistor. Although, SCL is generated by the ADuCM360 in this case, in theory, it can be bidirectional because the I^{2}C interface of the ADuCM360 can be set as a slave. The I^{2}C interface of the ADE7953 can only be used as a slave.

Note that the ADE7953 requires a minimum delay of 0.1 µs between the SCL and SDA edges; see the $t_{HD;DAT}$ specification in the data sheet. The ADE7953 data sheet also lists other timing specifications that must be maintained.

Figure 14 shows a typical I^{2}C transfer sequence.

The ADuCM360 has two I^{2}C interfaces: one via the P0.1/SCLK1/SCL/SIN and P0.2/MOSI1/SDA/SOUT pins and the other via the P2.0/SCL/UARTCLK and P2.1/SDA/UARTDCD pins. For this example, the P2.0/SCL/UARTCLK and P2.1/SDA/UARTDCD pins are used.

```
// P2.0/SCL/UARTCLK is used for SCL,
P2.1/SDA/UARTDCD is used for SDA.
*pGP2CON |= 0x05;
// Master Enable, Tx/Rx Request Interrupt
Enable.
*pI2CMCON = 0x131;
```

Every slave has an address for I^{2}C operation and a master can identify a slave using this address, which is set by the vendor. For example, the ADE7953 address is 0x38.

The I^{2}C addressing type can be either 7-bit or 10-bit. Due to the addressing type of the ADE7953, 7-bit addressing is used in this case.

The I^{2}C interface of the ADuCM360 has a 2-byte Tx and Rx FIFO scheme, as shown in Figure 13.

The core clock and peripheral clock are usually different. If data in one clock domain is transferred into the other clock domain, metastability can occur leading to error in communication. To avoid metastability, it is common to use a FIFO or dual-port RAM. Metastability cannot be resolved by the use of combinational logic.

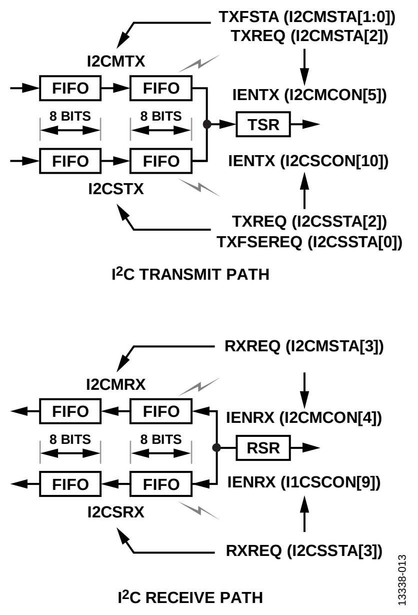


Figure 13. Datapath of I^{2}C Tx/Rx

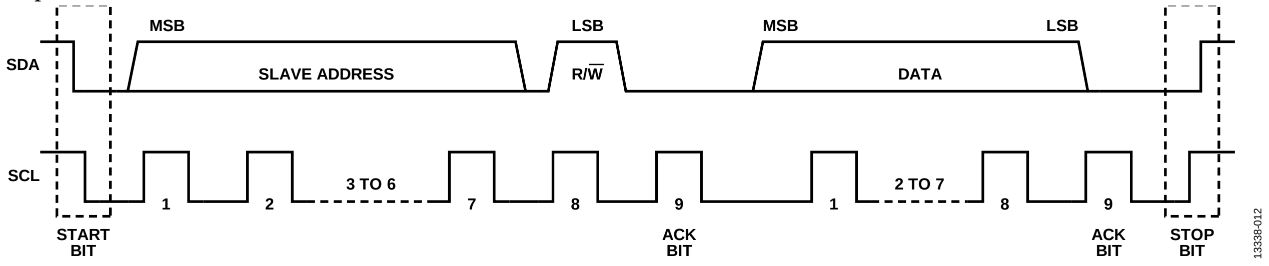


Figure 14. Typical I^{2}C Transfer Sequence

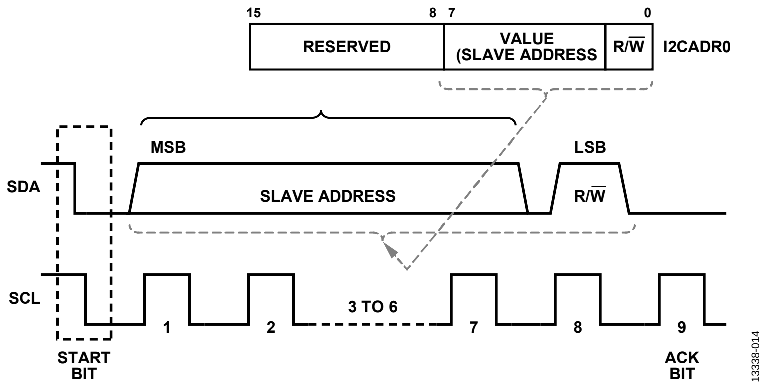


Figure 15. 7-Bit Addressing Mode

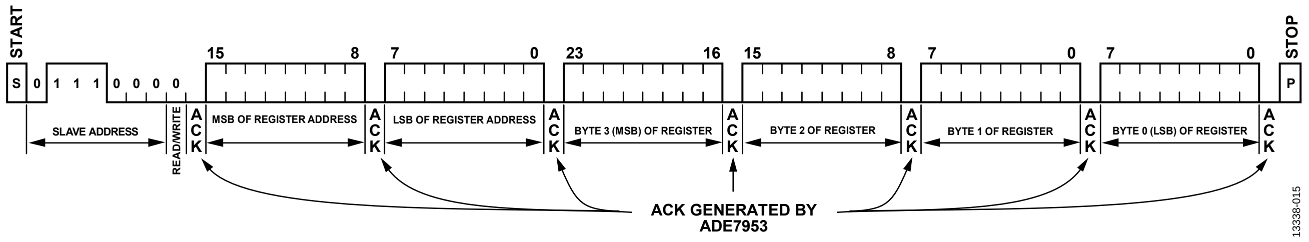


Figure 16. I²C Write Sequence of the ADE7953

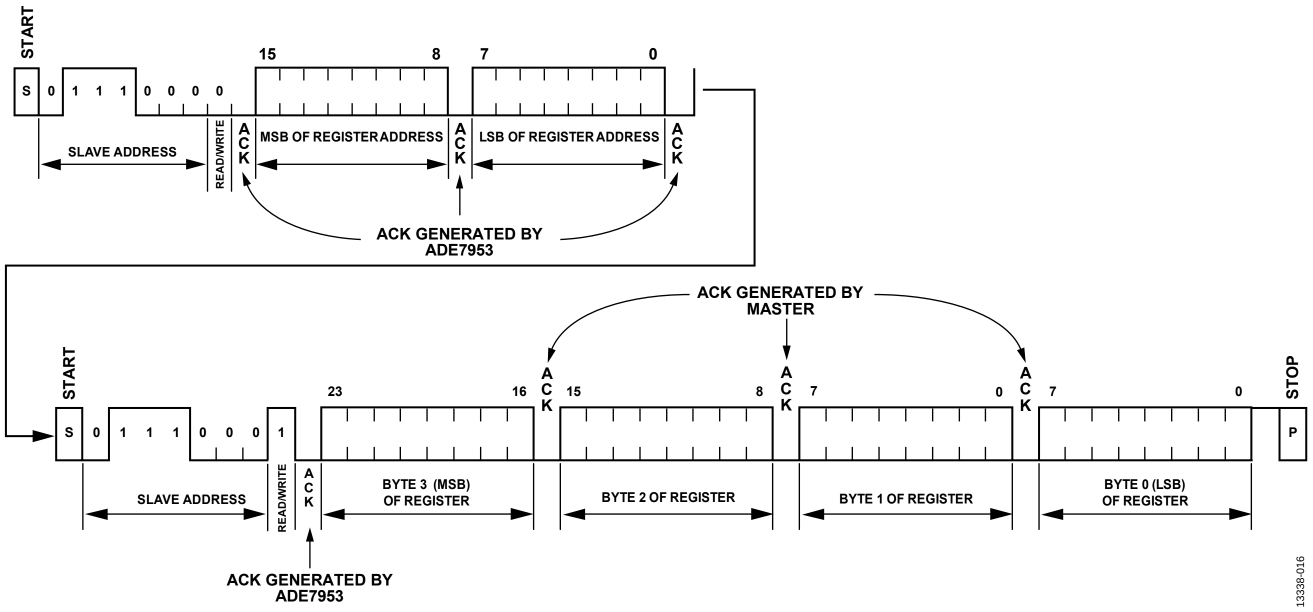


Figure 17. I²C Read Sequence of the ADE7953

Figure 15 shows how the master sends the 7-bit address (Bits[7:1]) and the directional (read or write) bit (Bit 0) to the slave.

The I2CADR0 register stores the slave address and the directional bit.

The I²C interface of the ADE7953 operates at a maximum serial clock frequency of 400 kHz.

The I²C write and read operations of the ADE7953 are shown in Figure 16 and Figure 17, respectively.

The ADE7953 I²C address is 0b'0111000x, where x is a directional bit. A 0 indicates a write, and a 1 indicates a read.

A write operation on the ADE7953 is initiated when the master issues a 7-bit device address and the directional bit is set to 0 (that is, 0x70). It is then followed by the 16-bit address of an internal register. Whenever each byte is received, the ADE7953 issues an acknowledge to the master. Following that, the master sends the register data, MSB first. The length of this data can be 8, 16, 24, or 32 bits long, depending on the register. When transmission of the final byte is complete, the master issues a stop condition, and the bus returns to the idle condition.

When writing source code, it is essential to understand how the I^{2}C is implemented in the target device.

The following piece of code represents the ADE7953 I^{2}C write operation:

```
void I2C_Write_ADE7953(void)
{
   int i=0;
   if (((I2CMasterTxDat[0] >4) &&
(I2CMasterTxDat[0] <=
6))||((I2CMasterTxDat[0] >9))) {
      printf("The specified ADE7953
register address is out of range\n");
   } else {
      ReadFlag=0;
      uiMasterTxIndex = 0;
      // Master case : send 1st data.
      *pI2CMTX =
I2CMasterTxDat[uiMasterTxIndex++];
      I2cMWrCfg(0x70);
      while (!ucComplete){}
      ucComplete = 0;
   }
}
```

The if condition in the preceding code is based on the ADE7953 address range. The MSB of the register address determines whether the address is out of range. Refer the ADE7953 data sheet and the ADE7953 Evaluation Board User Guide for more information.

The I2cMWrCfg(0x70) routine sends 0x01110000 (0x70) to the ADE7953, as shown in Figure 15 and Figure 16.

Whenever each byte is received, the ADE7953 issues an acknowledgement to the master and the ADuCM360 generates an I^{2}C interrupt to invoke the I2C0_Master_Int_Handler function found in the **ADuCM360_Test_Lib.h** file, which changes the value of ucComplete in the preceding code. If the ucComplete value is changed, I2C_Write_ADE7953() exits from the while loop and continues to send a byte to the ADE7953.

When writing to any of the ADE7953 registers, it is essential to make sure that the length of the data matches the description of the register. The same concept applies while reading back data from the ADE7953 IC. The I2C0_Master_Int_Handler function considers the register data length during both write and read operations.

Use Visual C++ debugger to help with debugging when there are bugs in the code, particularly while working on initial projects or with new ICs. Using the Visual C++ debugger may also help the user understand how a piece of code works.

Generally, the embedded code involves physical addresses but Visual C++ can only interpret virtual addresses, which are assigned by the MMU of the operating system (OS). The virtual addresses may change when the program is run. The Cortex-M family microcontrollers do not include a MMU; therefore, a MCU based on the Cortex-M core cannot be ported to, for example, Windows® CE or Linux® OS, easily. To learn how to use Visual C++ with embedded C code, refer to the Simulating an Embedded C Code Using Visual C++ section.

SIMULATING AN EMBEDDED C CODE USING VISUAL C++

After completing the entire C code in Visual C++, to make an executable image for the ADuCM360, perform a rebuild in the Keil MDK-ARM or IAR Embedded Workbench IDE. This rebuild is possible because of the following C/C++ precompiler directive in the sample code:

```
#ifdef FermiEmulation_Mode
// to initialize all of the ADuCM360
register set.
   ADuCM360_RegsInit();
#endif
```

The following step by step procedure is useful to understand the debugging process when using the Visual C++ 2012 debugger tool. Note that the following method can be used with any Visual Studio version.

1. Create a new project as shown in Figure 18.

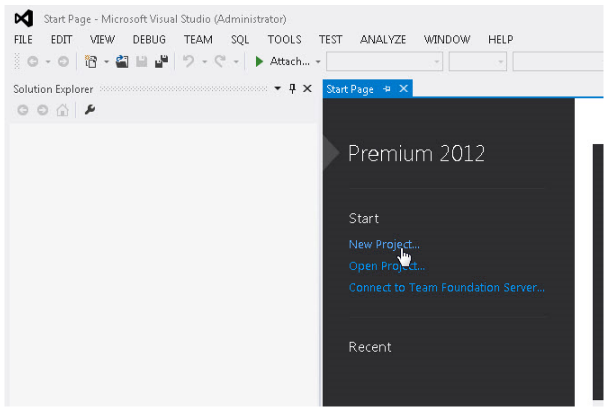
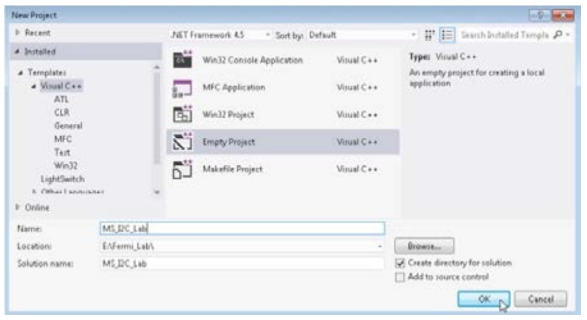

Start Page - Microsoft Visual Studio (Administrator)
FILE EDIT VIEW DEBUG TEAM SQL TOOLS TEST ANALYZE WINDOW HELP
Attach...
Solution Explorer
Start Page
Premium 2012
Start
New Project...
Open Project...
Connect to Team Foundation Server...
Recent
13338-017

Figure 18. Create New Project

2. Select **Empty Project** as shown in Figure 19.



Type: Visual C++
An empty project for creating a local application

Name: MS_I2C_Lab
Location: E:\Fermi_Lab\
Solution name: MS_I2C_Lab
[✓] Create directory for solution
[ ] Add to source control

Figure 19. Select Empty Project

3. Add all the sample files and existing items into the project, as shown in Figure 20 and Figure 21.

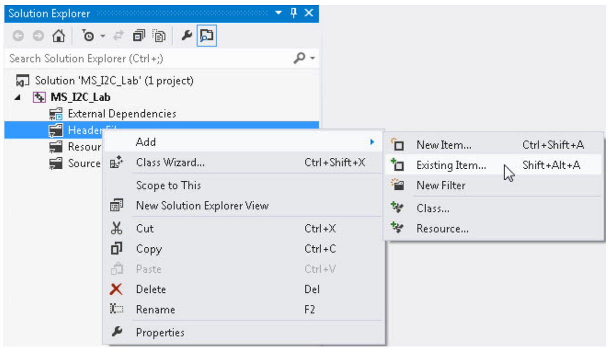
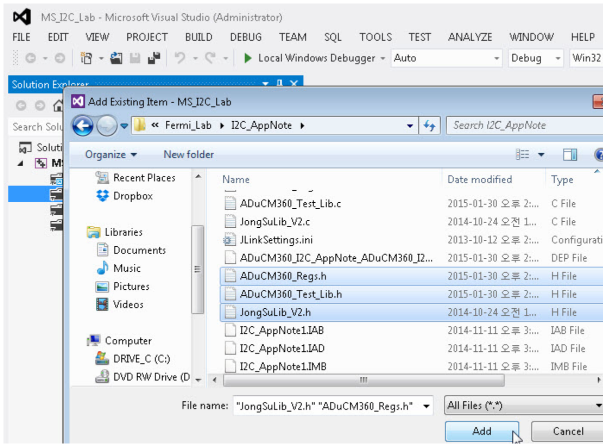


Figure 20. Add Sample Files


MS_I2C_Lab - Microsoft Visual Studio (Administrator)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ANALYZE WINDOW HELP
Local Windows Debugger
Auto
Debug
Win32
Add Existing Item - MS_I2C_Lab
« Fermi_Lab › I2C_AppNote ›
Search I2C_AppNote
Organize
New folder
Recent Places
Dropbox
Libraries
Documents
Music
Pictures
Videos
Computer
DRIVE_C (C:)
DVD RW Drive (D:
Add
Cancel
13338-020

Name	Date modified	Type
ADuCM360_Test_Lib.c	2015-01-30 오후 2:...	C File
JongSuLib_V2.c	2014-10-24 오전 1...	C File
JLinkSettings.ini	2013-10-12 오후 2:...	Configurati
ADuCM360_I2C_AppNote_ADuCM360_I2...	2015-01-30 오후 2:...	DEP File
ADuCM360_Regs.h	2015-01-30 오후 2:...	H File
ADuCM360_Test_Lib.h	2015-01-30 오후 2:...	H File
JongSuLib_V2.h	2014-10-24 오전 1...	H File
I2C_AppNote1.IAB	2014-11-11 오후 3:...	IAB File
I2C_AppNote1.IAD	2014-11-11 오후 3:...	IAD File
I2C_AppNote1.IMB	2014-11-11 오후 3:...	IMB File

File name: "JongSuLib_V2.h" "ADuCM360_Regs.h" All Files (*.*)

Figure 21. Add Existing Items

4. Modify certain properties as shown in Figure 22 to Figure 25.

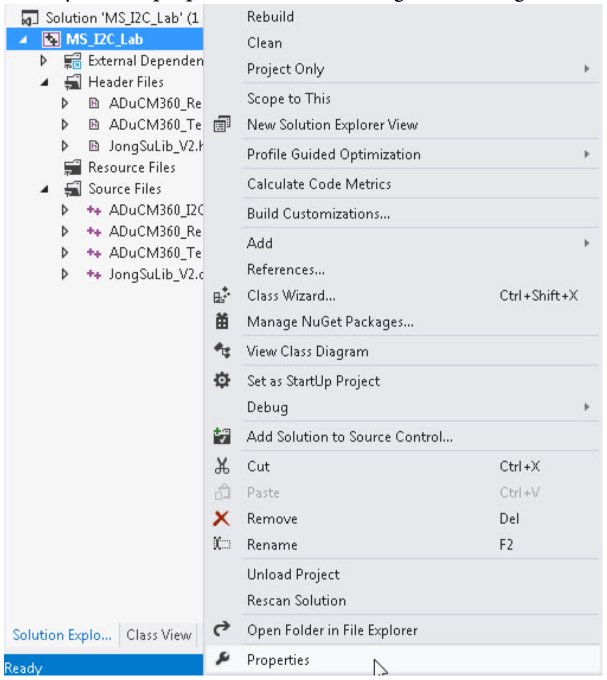
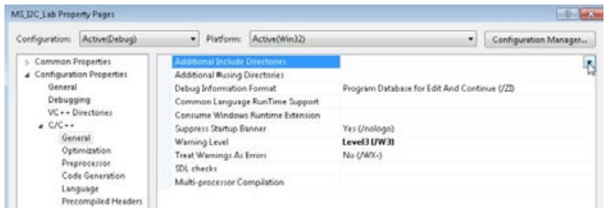


*Figure 22. Select **Properties***

5. Select the folder that contains the sample codes under **C/C++**. Click **Additional Include Directories**. Refer to Figure 23 and Figure 24.



Configuration: Active(Debug) Platform: Active(Win32)

Additional Include Directories	
Additional #using Directories	
Debug Information Format	Program Database for Edit And Continue (/ZI)
Common Language RunTime Support	
Consume Windows Runtime Extension	
Suppress Startup Banner	Yes (/nologo)
Warning Level	**Level3 (/W3)**
Treat Warnings As Errors	No (/WX-)
SDL checks	
Multi-processor Compilation	

Figure 23. Search for Sample Codes

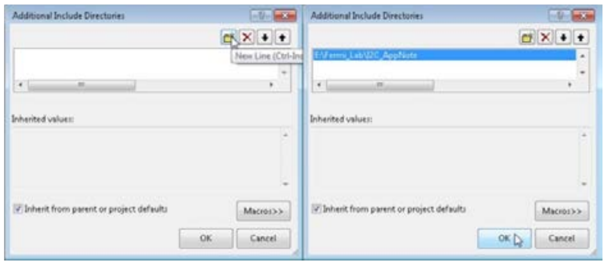
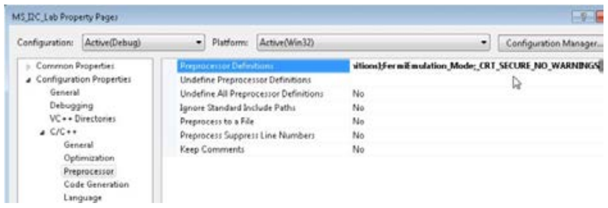


Figure 24. Select Folder with Visual Studio Sample Code

6. For Visual C++ compiler only, define **FermiEmulation_Mode**, as shown in Figure 25.
   This step predefines **FermiEmulation_Mode** based on which Visual C++ code can be programmed to work with the embedded C code.
   When using other Visual Studio versions, the corresponding options shown in Figure 22 to Figure 25 need to be searched in the menu options, but the procedure is the same. Add **_CRT_SECURE_NO_WARNINGS** only for .NET (see Figure 25).

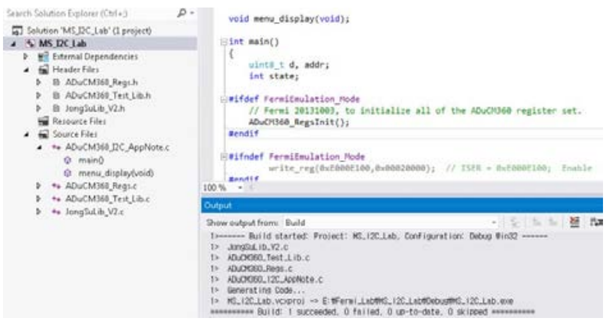


Configuration: Active(Debug) Platform: Active(Win32)

Preprocessor Definitions	itions);FermiEmulation_Mode;_CRT_SECURE_NO_WARNINGS
Undefine Preprocessor Definitions	
Undefine All Preprocessor Definitions	No
Ignore Standard Include Paths	No
Preprocess to a File	No
Preprocess Suppress Line Numbers	No
Keep Comments	No

Figure 25. Define the Code

7. Compile and link all the sample codes under Visual C++, as shown in Figure 26.



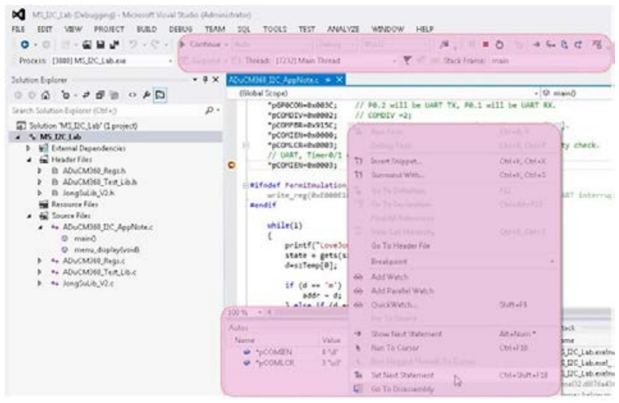
```
void menu_display(void);

int main()
{
    uint8_t d, addr;
    int state;

#ifdef FermiEmulation_Mode
    // Fermi 20131003, to initialize all of the ADuCM360 register set.
    ADuCM360_RegsInit();
#endif

#ifndef FermiEmulation_Mode
        write_reg(0xE000E100,0x00020000);  // ISER = 0xE000E100;  Enable
#endif
```

```
1>------ Build started: Project: MS_I2C_Lab, Configuration: Debug Win32 ------
1>  JongSuLib_V2.c
1>  ADuCM360_Test_Lib.c
1>  ADuCM360_Regs.c
1>  ADuCM360_I2C_AppNote.c
1>  Generating Code...
1>  MS_I2C_Lab.vcxproj -> E:\Fermi_Lab\MS_I2C_Lab\Debug\MS_I2C_Lab.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Figure 26. Compile and Link all Sample Codes

As shown in Figure 26, there are no errors and warnings in the code.

Figure 27 shows how to simulate the embedded code using Visual C++.



```
*pGP0CON=0x003C;    // P0.2 will be UART TX, P0.1 will be UART RX.
*pCOMDIV=0x0002;    // COMDIV =2;
*pCOMFBR=0x915C;
*pCOMIEN=0x0000;
*pCOMLCR=0x0003;
// UART, Timer0/1
*pCOMIEN=0x0003;

#ifndef FermiEmulation
    write_reg(0xE000E1
#endif

    while(1)
    {
        printf("LoveJo
        state = gets(s
        d=szTemp[0];

        if (d == 'm')
            addr = d;
        } else if (d =
```

Figure 27. Simulate Embedded Code

To understand how Visual C++ works, analyze both **ADuCM360_Regs.c** and **ADuCM360_Regs.h** carefully. These files define real addresses related to the ADuCM360 and enable the MMU to assign virtual addresses corresponding to the real addresses so that Visual C++ can debug the files. Note that the files can also be debugged by Keil MDK-ARM or IAR Embedded Workbench. If debugging is done using Keil MDK-ARM or IAR Embedded Workbench, FermiEmulation_Mode does not need to be used.

This debugging method is very simple and can be used on all types of processors. The advantage of using Visual C++ debugger is that the I2C0_Master_Int_Handler function and the set/get commands can be analyzed easily.

The following code of the I2C0_Master_Int_Handler function is for synchronization:

```
#ifndef FermiEmulation_Mode
        __asm{ nop}
    __asm{ nop}
    __asm{ nop}
    __asm{ DSB}
    __asm{ nop}
    __asm{ nop}
#endif
```

The printf function in the downloaded sample code is specially developed to reduce code size when using Visual C++; therefore, if the Keil MDK-ARM or IAR Embedded Workbench runtime library is used, the code size may be larger. The compact printf function can be found in the **JongSuLib_V2.c** library file, which includes many other useful functions, such as register read/write functions. One useful library file, named **ADuCM360_Test_Lib.c**, includes functions to support ADuCM360 peripherals such as I^{2}C, SPI, UART, and ADCs/DACs. These two general-purpose libraries can be ported to other projects as well, as needed.

CONCLUSION

This application note demonstrates how to implement I^{2}C C code to monitor and modify the registers of the ADE7953 using the ADuCM360. To develop any embedded code, it is recommended that Visual C++ be used because coding and debugging are very simple using this tool. Then use Keil MDK-ARM or IAR Embedded Workbench to compile and link the C code to make an executable image without additional modification.

I^{2}C refers to a communications protocol originally developed by Philips Semiconductors (now NXP Semiconductors).