

Configuring the TMS320F280x DSP as an I2C Master or Slave Processor

Todd Anderson

ABSTRACT

This application report provides details on how to configure a TMS320F280x device so that it can be used as an I2C slave processor and the protocol used to support that configuration.

This application report contains project code that can be downloaded from <http://www.ti.com/lit/zip/SPRAAN8>.

Contents

1	Introduction	1
2	Setup	3
3	Application Descriptions	6
4	References	7

List of Figures

1	Master-Slave Writes	2
2	Master-Slave Reads	2
3	Hardware Setup for I2C Master - Slave.....	3
4	Code Composer Studio Setup for Heterogeneous Mode	4
5	Heterogeneous Parallel Debug Manager.....	5
6	Screen Shot Showing Two eZdsp Windows.....	5

List of Tables

1	eZdsp I2C Connections	3
---	-----------------------------	---

1 Introduction

The I2C processor can be useful in a variety of configurations. A common configuration involves a processor used as a master and one or more processors used as slaves. This report covers the example code that communicates via the I2C to update six read/write locations in the RAM of a slave 2808 processor. The code does this by using the following I2C commands: I2C write (to write data from master to slave) and I2C read (to read data from the slave).

I2C write (from host): The write command is straight-forward. The I2C address (slave) is followed by three bytes of data – the first byte is used to define the register that is being written. In the example to follow, it is a value between 0 and 5.

The next two bytes of data are the value that is placed in the *register* location. The high byte is sent first, followed by the low byte of data.

A particular protocol that is sometimes used for master-slave writes is shown in [Figure 1](#). The full command is enveloped by a start bit (S) and a stop bit (P). After the address is written, there is a one-bit field where the write condition is specified.

Code Composer Studio is a trademark of Texas Instruments.

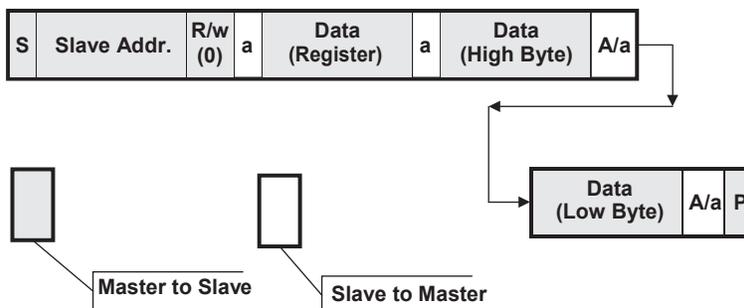


Figure 1. Master-Slave Writes

I2C read (from host): The I2C read contains a second start bit, known as a repeated start bit. The first part of the command is actually a write, and the second part of the command is the read.

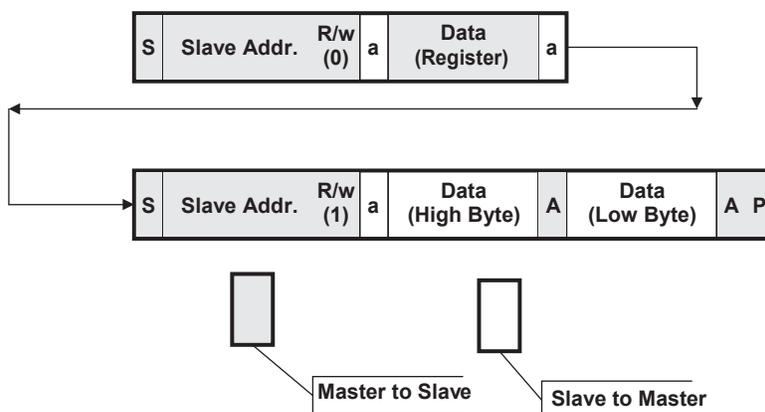


Figure 2. Master-Slave Reads

2 Setup

In the absence of an I2C protocol board, this application report makes use of two eZdsp2808 boards + a JTAG controller. If there is an I2C controller available, then the USB-JTAG master I2C board is not required.

2.1 Hardware (eZdsp2808)

The setup consists of two eZdsp boards; one is connected directly to the host computer, and the other is connected via a JTAG controller (XDS510USB). By using this setup, a mode of the Code Composer Studio™ software called heterogeneous mode can be used, which allows access to windows of both master and slave at the same time as shown in Figure 3. External connections must be made to the I2C clock and data pins. Table 1 shows the eZdsp I2C connections.

Table 1. eZdsp I2C Connections

Signal	F2808 Pin	eZdsp Connector Pin
I2C Data	GPIO32 (pin 100)	P8-36
I2C Clock	GPIO33 (pin 5)	P8-38
Ground	V _{SS} (pin 2)	P8-40

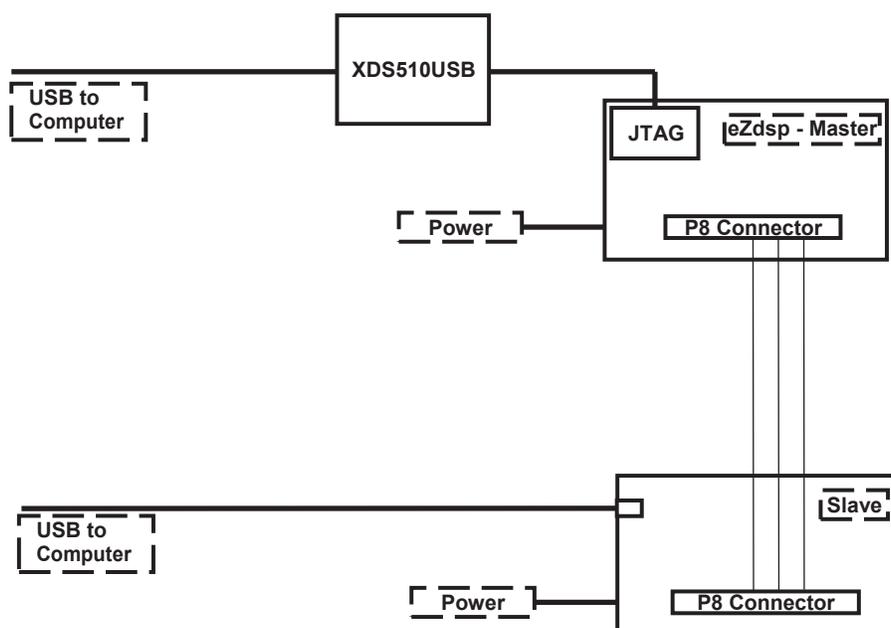


Figure 3. Hardware Setup for I2C Master - Slave

2.2 Software (Code Composer Studio)

The hardware setup allows the Code Composer Studio IDE software to be set up in heterogeneous mode, with two windows (one for each eZdsp). The code for the I2C master is loaded on one board, and the code for the I2C slave was loaded on the other board.

2.2.1 Code Composer Studio Setup

Figure 4 shows the setup by having two *dissimilar* emulators, the XDS510USB (set up for the 2808) and the F2808 eZdsp, it allows for much easier debug of the master and slave code because they can be viewed and worked with on the same screen.

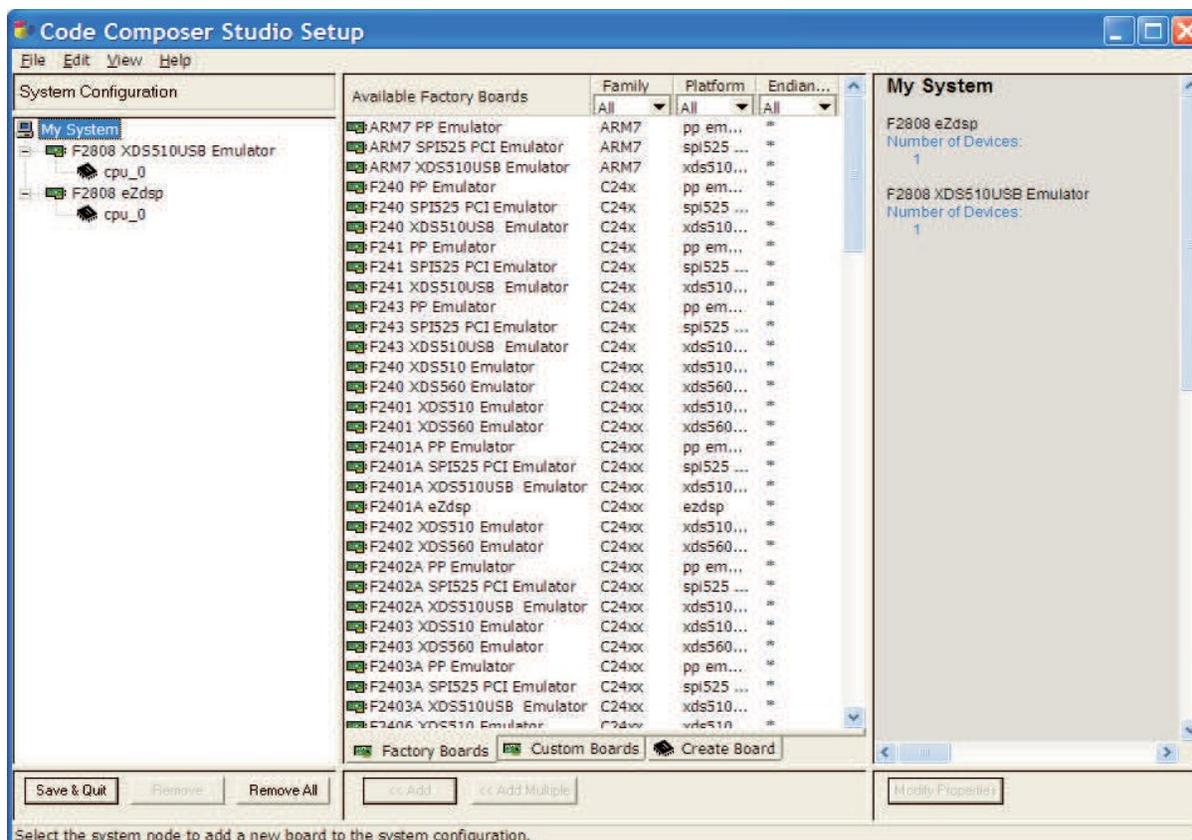


Figure 4. Code Composer Studio Setup for Heterogeneous Mode

2.2.2 Code Composer Studio Operation

This section provides the procedure used to debug the eZdsp.

1. Launch the Code Composer Studio software. As shown in [Figure 5](#), the parallel debug manager screen appears when there are multiple DSP's on a system.

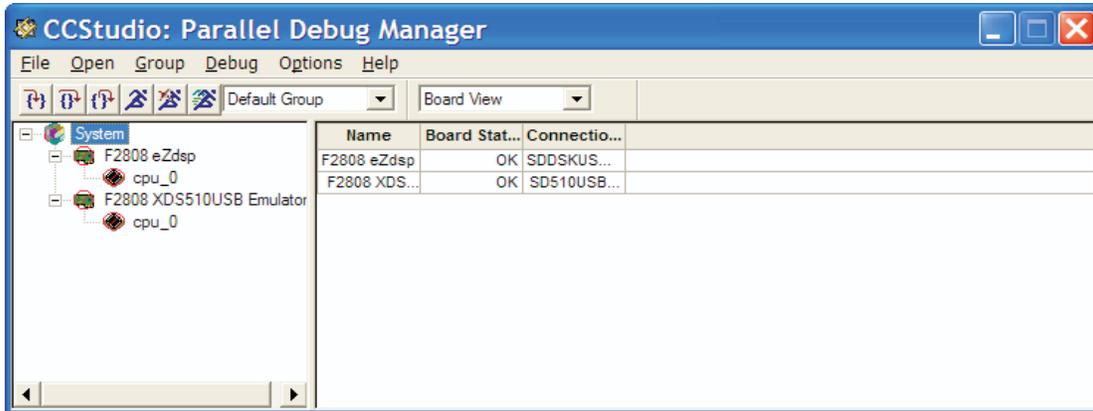


Figure 5. Heterogeneous Parallel Debug Manager

2. Connect to the CPU by: Debug → Connect. Connect can also be done with each of the eZdsp windows. You save time by performing it here.
3. Display each of the eZdsp's by: Open → F2808 eZdsp, followed by: Open → F2808 XDS510USB Emulator.

Completing the steps above results in a display illustrated in [Figure 6](#):

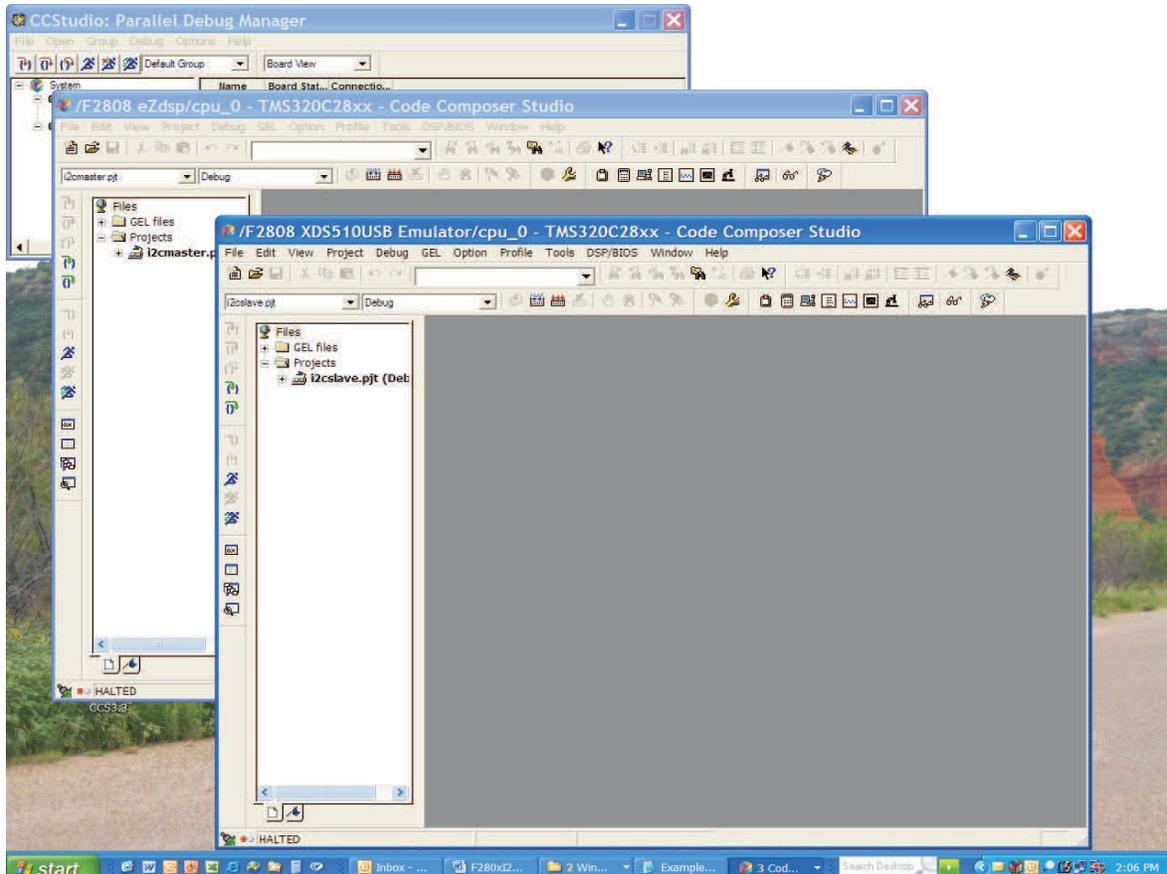


Figure 6. Screen Shot Showing Two eZdsp Windows

Simultaneous display of two application screens allows many advantages. If you have a breakpoint on the slave and send a command from the master, it is possible to determine right away if the slave received the command.

Notice that the eZdsp has the I2C master project loaded, and the XDS510USB has the I2C slave project loaded. From here, each individual project can be built and loaded into the respective 2808's. During debug, you can make use of breakpoints, single stepping, *run to cursor* and other Code Composer Studio software items.

Note: One of the first things that was done after loading each 2808 with the master/slave programs was a reset of the DSP. This allows a known environment for each of the processors.

3 Application Descriptions

The following are descriptions of the code in the master and the slave processors. No attempt was made to enhance the code beyond the basic I2C implementation. An example implementation that would involve additional enhancements could be a master wanting to set up an ADC measurement on the slave and then get the data from that measurement. There could also be a situation where the master may want to set up a PWM output on the slave. Such applications are left to the developers and are not covered in this applications report, only the necessary amount of code is introduced to show how master-slave I2C communication is performed.

3.1 Master Code

This section includes information on the initialization, application loop and interrupt service routine needed to implement master-slave I2C communication.

3.1.1 Initialization

1. Initialize system control.
2. Initialize GPIO.
3. Clear all interrupts and initialize PIE vector table.
4. Initialize device peripherals.
5. Initialize master I2C register.
6. Enable interrupts.

3.2 Slave Code – Application Loop

The application examines a flag bit [DataReady]. If there is data ready, then the application updates the particular register with a switch statement that has been sent from the master.

3.3 Interrupt Service Routine

Three bits determine the source of an I2C-related interrupt. These bits are copied into a register called IntSource. A switch routine is used to evaluate the IntSource register. In the example code only two cases are evaluated:

- I2C_RX_ISRC (I2C read)
- I2C_TX_ISRC (I2C write)

Finally, there is routine service cleanup necessary to enable subsequent I2C interrupts.

3.4 Suggested Start-up

The recommended procedure to run the setup is:

1. Download code to each of the eZdsp's.
2. Load the code, then reset each of the eZdsp boards by Debug → Reset CPU.
3. Run the slave eZdsp first.
4. Run the master eZdsp board next.
5. Set up a breakpoint just after the check of DataReady in the slave code and follow the interactions of the master-slave processors.

4 References

- *TMS320F2809, TMS320F2808, TMS320F2806, TMS320F2802, TMS320F2801, TMS320C2802, TMS320C2801, and TMS320F2801x DSPs Data Manual* ([SPRS230](#))
- *eZdsp 2808 Technical Reference Manual*, Spectrum Digital
- *TMS320x280x Inter-Integrated Circuit (I2C) Reference Guide* ([SPRU721](#))
- *Code Composer Studio Development Tools v3.3 Getting Started Guide* ([SPRU509](#))

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), 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, 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 (www.ti.com/legal/termsofsale.html) 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.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2019, Texas Instruments Incorporated