Technical Article

MSP MCU I2C Getting Started Guide

Ryan Brown

Here is an introductory guide to starting your project involving I2C communication using an ultra-low-power MSP microcontroller (MCU):

Introduction

I^2C (or I2C, for Inter-Integrated Circuit) is a form of two-line communication used to communicate most commonly between microcontrollers and peripheral ICs at a low speed in short-distance, intra-board applications. Because of its widespread adoption, learning to use I2C communication with MSP MCUs has become essential for helping engineers develop their applications. This guide provides the tools and resources necessary to understand I2C protocol and implement it using an ultra-low-power MSP MCU to access and control IC devices.

Device Terminology

The device driving the SCL clock line is called the master and the devices which respond to it are known as slaves. In most applications the MSP MCU is the master device and the peripheral ICs are the slaves, however there are times when the MSP device is a slave to other MCUs or processors.

Physical Bus

The I2C bus consists of two wires, SCL and SDA. SCL is the clock line used to synchronize all data transfers and SDA is actual the data line. A third line, common GND, is also required but often not mentioned. Because both lines are “open drain” drivers they each require pull-up resistance to a power supply line so that the outputs remain high during no operation. For MSP MCU applications the supply voltage should match that of the MSP MCU’s Vcc. The value of the pull-up resistors is traditionally 4.7 kΩ but this value ranges from less than a kΩ to over 10 kΩ, depending on the slave devices used. Please refer to the device’s datasheet in order to use the correct pull-up resistance. Multiple slave devices can share an I2C bus using single pull-up resistors for the SCL and SDA lines.

I2C Software Protocol

Regardless of application each I2C-capable device is required to follow the software protocol commonly defined for all I2C devices, the general structure of which always remains the same. Communication begins with a start sequence and concludes with a stop sequence, with 8-bit data transfer sequences in between. A start bit is followed with the slave address, the length of which is typically seven bits (although in rare cases, 10-bit addressing is used). These seven bits are placed in the upper 7 bits of a byte and the LSB (Least Significant Bit) is used to store the Read/Write (R/W) bit. This bit lets the slave device know whether it will be written to (bit value zero) or read from (bit value one). For a write, the transaction sequence is as follows:

1. Send the start sequence
2. Send the slave address with R/W bit low
3. Send the register number
4. Send the data byte(s)
5. Send the stop sequence

The read transaction sequence is very similar to that of a write, with the exception that instead of sending data bytes it will re-send the start sequence (known as a repeated start) and the slave address (although this time, with the R/W bit high for a read) so that it may receive data instead of sending it. The transaction is concluded after the master sends the typical stop sequence. Below is the read transaction sequence:

1. Send the start sequence
2. Send the slave address with R/W bit low
3. Send the register number
4. Send the start sequence again (repeated start)
5. Send the slave address with R/W bit high
6. Read data byte(s)
7. Send the stop sequence

MSP MCU Communication Peripherals

There are four different possible peripherals available on MSP devices to realize serial communication. Only one of these will exist per device. In order of ease by which they can be used to realize I2C communication on a MSP MCU, from hardest to easiest, they are listed as such:

- USART: Universal synchronous/asynchronous receiver/transmitter. This is the oldest form of communication and exists on most MSP430F1xx MCUs. It does not support I2C, therefore a software-based bit-bang solution must be used to communicate with I2C devices.
- USI: Universal serial interface. Another more simple form of communication which is used on cost-effective or space-limited devices such as some components in the MSP430G2xx family. The I2C state machine does not exist on the device and must be implemented in software, commonly through the use of separate functions.
- USCI: Universal serial communication interface. A standard communication peripheral optimized for ISR and flag usage. Common in the MSP430F5xx/F6xx family, this peripheral includes a hardware-based I2C state machine and is therefore requires less code to operate.
- eUSCI: Enhanced universal serial communication Interface. The most advanced communication peripheral available on MSP devices which improves on existent USCI functionality and is included in all MSP430FRxx (FRAM) MCUs.

When selecting an MSP device with I2C applications in mind one should understand that the code structure varies depending upon the peripheral existent on the specific MSP derivative. Each variant includes different registers, ISRs, and functions that must be taken into account. It must also be made clear that not all device families use the same peripheral (USCI and eUSCI exists in the MSP430F5xx/6xx family, USI and USCI exists in the MSP430G2xx family, etc) which can be quite confusing when referring to the Family User's Guide. Therefore caution must be taken into account to view the correct material and select the correct example code upon which to start developing one's application. Texas Instruments provides basic I2C code examples for USI, USCI, and eUSCI communication which can be found on the MSP derivative's product page under Tools & software -> Software -> Examples (available as a ZIP file, note that these packages only contain the code examples relevant to the peripherals that exist on the particular device). For devices which use USART or do not include a communication peripheral, I2C bit-bang solutions are provided online through community effort. Regardless of peripheral used, pull-up resistors will always be required to accomplish I2C communication. Some MSP devices have internal pull-up resistors but use of these is not recommended as several slave devices require a particular resistance not accommodated internally.

Tips for Implementing I2C with the MSP

When attempting to communicate between a peripheral IC and MSP using I2C, here are some suggestions that should be reviewed to help avoid common implementation errors:

- Start with example I2C code provided specifically for your MSP derivative (product page -> Tools & software -> Software -> Examples). Review the changes made to the I2C registers with that of the Family User's Guide (take caution that you are looking at the correct peripheral section) so that you get a firm grasp on what alterations are necessary for successful communication.
- Use the pull-up resistance and address specified by the slave's datasheet (can sometimes vary based on input variables). Remember that the 7 bits of the slave address are stored in the upper 7 bits of the byte followed by a R/W bit set by the communication peripheral, therefore when setting the slave address register the value may need to be shifted to the left by one bit.
- Begin by writing to a register and monitoring the MSP device for an ACK. Leverage fault flags and bench equipment to alert of communication failure. Use the debugging tools offered by CCS or IAR to

understand how the code operates, what registers are being accessed, and when/how functions/ISRs are accessed. After doing this it will become easier to add functionality for reading from a register.
- The USCI/eUSCI state diagram indicates that the UCTXSTP bit needs to be set before the last byte is received. In applications where only one byte is being received, the UCTXSTP bit is set along with the UCTXSTT bit. If multiple bytes are received then UCTXSTP should be set after reception of the N-1th byte. This ensures that the stop sequence is sent immediately after receiving the last byte.

Debugging Advice

Before giving up on code that doesn't seem to operate correctly, here are some checkpoints to consider for debugging the system:

- Confirm the pull-up resistance & slave address values, verifying them with the slave datasheet.
- Re-check communication peripheral initialization, including: register settings, proper pinouts, enabled interrupts, peripheral on/released for operation, etc.
- Use any tools available (IDE debugger, logic analyzer, oscilloscope, etc.) to confirm that the MSP430 and slave device are following the I2C software protocol verbatim.
- Check the device erratasheet for known I2C issues and see if the errata description matches the application's symptoms.
- Research the issue on the E2E forum to see if any similar cases have been solved. Try various combinations of I2C-related key words and take advantage of the search filters.

E2E Support

If proper debugging and research methods have failed, the TI E2E community forum can be a great resource for directly communicating with device experts. Make sure to include detailed information regarding all aspects of the issue at hand to help community members and TI engineers better service the request, including:

- MSP430 derivative
- LaunchPad or TI target board being used, or schematic if a custom board
- Slave device
- Precise description of issue or problem being seen
- Behavior observed while using the debugger (CCS or IAR)
- I2C initialization and function/ISR code snippets (not full code)
- Logic analyzer and oscilloscope images with appropriate labels

Additional Resources

Migrating from the USCI Module to the eUSCI Module (SLAA522): http://www.ti.com/lit/an/slaa522a/slaa522a.pdf

Using the USCI I2C Master (SLAA382): http://www.ti.com/lit/an/slaa382a/slaa382a.pdf

Using the USCI I2C Slave (SLAA383): http://www.ti.com/lit/an/slaa383/slaa383.pdf

Using the I2C Bus (blog): http://www.robot-electronics.co.uk/i2c-tutorial

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