Application Report
SLAA382–December 2007

Using the USCI I²C Master

Uli Kretzschmar — *MSP430 Systems*
Christian Hernitscheck — *MSP430 Application Europe*

ABSTRACT

This document is an overview of the use of the I²C master function set for MSP430 devices with the USCI module. These functions can be used by MSP430 master devices to ensure proper initialization of the USCI module and provide I²C transmit and receive functionality. A similar version with DMA support has also been included. The USCI I²C master function set only supports single-master transmitter/receiver mode using 7-bit device addressing.

Note: The USCI I²C master package includes a demonstration application that can be used on any MSP430 2xx device with the USCI module.

Contents

1 Introduction 2
2 Usage From C 3
2.1 Example With DMA 3
2.2 Example Without DMA 4
3 Compiling the USCI I²C Master Code 5
4 Included Files 5
4.1 Function Description 5
5 Examples of USCI I²C Master Usage 8
5.1 Receiving n Bytes 8
5.2 Transmitting n Bytes 8
5.3 Checking Presence of a Slave 9
6 Code Size 9
7 References 9

All trademarks are the property of their respective owners.

1 Introduction

When using the MSP430 with peripherals, I^2C is often used for communication. There are several MSP430 devices that have an incorporated USCI module, which is capable of this communication protocol.

The USCI I^2C master function set offers sample code that make I^2C communication easy. Instead of having to configure the different registers of the UCSI module, the user can easily use the included functions with well-defined parameters to start a communication. These functions serve only for setting up the USCI module. The user is free to include low-power mode functionality to allow the CPU to be turned off at the application level or continue calculations during I^2C communication.

The USCI I^2C master package includes functions that support both transmit and receive operations:

- Master transmitter (the master addresses a slave and transmits data to it)

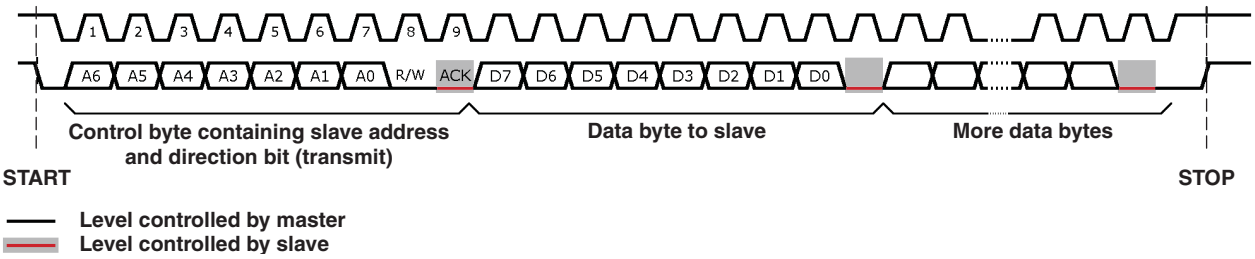

Figure 1. Master Transmitter

- Master receiver (the master addresses a slave and receives data from it)

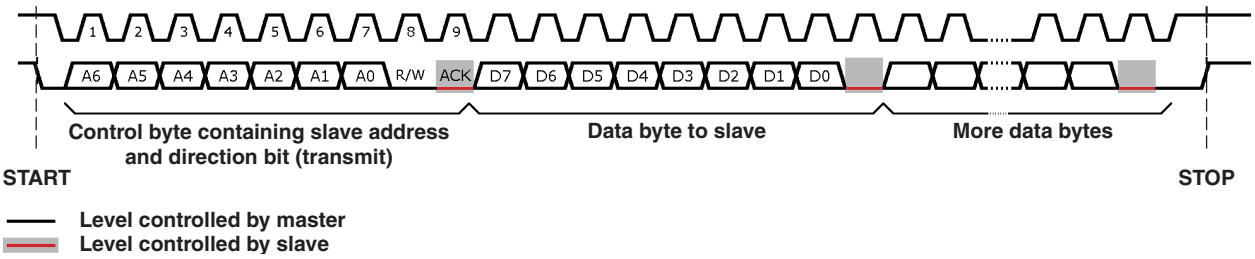

Figure 2. Master Receiver

Both of these functions support only 7-bit addressing.

2 Usage From C

The file TI_USCI_I2C_master.c or TI_USCI_I2C_master_dma.c must be added to the project. The first file supports I^2C communication using only the USCI module, while the second file supports I^2C communication using USCI and DMA module. The corresponding header file (TI_USCI_I2C_master.h or TI_USCI_I2C_master_dma.h) must be included to access to the master function set.

The master program TI_USCI_I2C_master.c (or TI_USCI_I2C_master_dma.c) runs on an MSP430 master device and is connected to an MSP430 slave running the slave program (TI_USCI_I2C_slave.c). [4]

Note: The master demostration applications were developed for use with the 2xx family. However, they can be easily modified for use with any MSP430 device with the USCI module.

Note: One of two different source files for the USCI master can be used, depending on whether or not DMA operation is desired. TI_USCI_I2C_master.c and TI_USCI_I2C_master.h must be used for operation without DMA, and TI_USCI_I2C_master_dma.c and TI_USCI_I2C_master_dma.h must be used for operation with DMA.

The usage of DMA causes some overhead in the initialization and interrupt routines for cases when only a few bytes are sent within a protocol. Therefore, it is recommended to use the DMA supported version if a large number of bytes are to be moved.

2.1 Example With DMA

Note that these functions with DMA support work only if an MSP430 version with an integrated DMA module is used.

```
#include "msp430x26x.h"
#include "TI_USCI_I2C_master_dma.h"

unsigned char array[9] = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09 };

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                  // Disable Watchdog

  _EINT();                                   // enable interrupts

  TI_USCI_I2C_DMA_transmitinit(0x48,0x3f);   // initialize USCI and DMA module
  while ( TI_USCI_I2C_notready() );          // wait for bus to be free
  TI_USCI_I2C_DMA_transmit(8,array);         // transmit the first 8 bytes of array

  LPM0;                                      // put CPU to sleep during
                                             // communication
}
```

This short program transmits the slave address and eight bytes of data. During the transmission of the first seven data bytes, the CPU is in Low-Power Mode 0, which is defined in the main program. The DMA module manages loading the seven data bytes that need to be be sent. The master transmit function configures the interrupt to trigger the transmission of the last data byte (eighth data byte in the previous code example). This means that the CPU is running during the execution of the interrupt service routines.

2.1.1 Initialization

As shown in the previous example, configuring the device in master-transmit mode with DMA support requires that the function TI_USCI_I2C_DMA_transmitinit is called once before transmission begins.

Two parameters must be passed in this function. The first is the address of the slave in the I^2C communication, and the second is a prescale factor that is used to set the baud rate. The resulting baud rate is the DCO frequency divided by the prescale value.

Calling the initialization routine while an I^2C communication is still active can result in undefined behavior.

2.1.2 Sending a Protocol Frame

After initialization of the USCI module, a protocol frame can be sent. Sending a protocol frame is done with the following steps:

1. Check whether or note the bus is free. This can be done using the TI_USCI_I2C_notready function, which returns a number greater than zero if the bus is busy. The return value is zero when the bus is free.
2. Use TI_USCI_I2C_DMA_transmit function to send an I^2C frame. This function has two parameters: the first determines the number of bytes to be sent, and the second is a pointer to a data array that holds the data to be sent.

2.2 Example Without DMA

If the MSP430 device does not have an integrated DMA module, the following functions might be used.

```
#include "msp430x26x.h"
#include "TI_USCI_I2C_master.h"

unsigned char array[5] = { 0x1, 0x2, 0x3, 0x4, 0x5 };

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                       // Disable Watchdog

  _EINT();                                        // enable interrupts

  TI_USCI_I2C_transmitinit(0x48,0x3f);            // initialize USCI
  while ( TI_USCI_I2C_notready() );               // wait for bus to be free
  TI_USCI_I2C_transmit(3,array);                  // transmit the first 3 bytes
                                                  // of array

  LPM0;                                           // put CPU to sleep during
                                                  // communication
}
```

The usage of the USCI I^2C function set without DMA support is the same as the usage of the functions supporting DMA. The functions can be distinguished by their suffixes.

- Functions beginning with TI_USCI_I2C_DMA_ need a DMA for operation.
- Functions without DMA in their names (for example, TI_USCI_I2C_transmit) do not use DMA.

It is, of course, also possible to use the sample code without DMA support for devices with a DMA module.

3 Compiling the USCI I²C Master Code

This application package is distributed as source code and is intended to be compiled with a project. To accomplish this:

- Add TI_USCI_I2C_master.c (or TI_USCI_I2C_master_dma.c for DMA support) to the project.
- Include the necessary header definitions by adding `#include "TI_USCI_I2C_master.h"` (or `#include "TI_USCI_I2C_master_dma.h"` for DMA support) to the user file.
- Change the MSP430 device-specific include file (MSP430 standard header file) in the C file of the function set.
- Adjust the definitions of SDA_PIN and SCL_PIN in the header file (TI_USCI_I2C_master.h or TI_USCI_I2C_master_dma.h).

4 Included Files

TI_USCI_I2C_master.c	This file contains all necessary functions to perform I²C communication using the USCI module of the MSP430 without using the DMA.
TI_USCI_I2C_master.h	This file includes the definitions of the functions and variables that are used in TI_USCI_I2C_master.c. It also contains the precompiler variables SDA_PIN and SCL_PIN that define which pins of the MSP430 are used for I²C. This file must be included in any C program that calls the master function set. This file supports only USCI usage without DMA.
TI_USCI_I2C_master_dma.c	This file contains all necessary functions to perform I²C communication using the USCI module of the MSP430 when using the DMA.
TI_USCI_I2C_master_dma.h	This file includes the definitions of the functions and variables that are used in TI_USCI_I2C_master_dma.c. It also contains the precompiler variables SDA_PIN and SCL_PIN that define which pins of the MSP430 are used for I²C. This file must be included in any C program that calls the master function set with DMA support.

4.1 Function Description

4.1.1 General Functions (TI_USCI_I2C_master_dma.h and TI_USCI_I2C_master.h)

- **unsigned char TI_USCI_I2C_notready()**
  This function takes no parameters and returns zero if the I²C bus is not busy. If the I²C bus is busy, it returns a value different from zero.
- **unsigned char TI_USCI_I2C_slave_present(unsigned char slave_address)**
  This function checks whether or not a slave is connected to the I²C bus. It returns a number different from zero if the slave replies to its address with acknowledge. Otherwise, it returns zero.
  Unlike the other functions in this demonstration, this function blocks the CPU for as long as the communication on the bus lasts. It has the following parameter:
  - **unsigned char slave_address**
    This is the slave address that is to be checked. This address may differ from the address provided in the initialization procedure of the USCI module. Note that the 7-bit slave address is right justified.

4.1.2 Functions With DMA Support (TI_USCI_I2C_master_dma.h)

- **void TI_USCI_I2C_DMA_receiveinit(unsigned char slave_address, unsigned char prescale)**
  This function initializes the USCI module for master-receive operation with usage of the DMA module. It has the following parameters:
  - **unsigned char slave_address**
    This parameter sets the address of the slave in the communication. The 7-bit slave address is right justified.
  - **unsigned char prescale**
    This parameter sets the desired baud rate. This works in an indirect manner, the resulting baud rate is the quotient of DCO frequency and the prescale parameter.
- **void TI_USCI_I2C_DMA_transmitinit(unsigned char slave_address, unsigned char prescale)**
  This function initializes the USCI module for master-transmit operation with usage of the DMA module. It has the following parameters:
  - **unsigned char slave_address**
    This parameter sets the address of the slave in the communication. The 7-bit slave address is right justified.
  - **unsigned char prescale**
    This parameter sets the desired baud rate. This works in an indirect manner, the resulting baud rate is the quotient of DCO frequency and the prescale parameter.
- **void TI_USCI_I2C_DMA_receive(unsigned char byteCount, unsigned char *field)**
  This function starts an I^2C communication in master-receiver mode with usage of the DMA module. It has the following parameters:
  - **unsigned char byteCount**
    This is the number of bytes that are to be received.
  - **unsigned char *field**
    This is a pointer into an array variable that is used to store the received bytes. Since I^2C communication works bytewise, it makes sense to use a field of bytes, for example, unsigned char values.
- **void TI_USCI_I2C_DMA_transmit(unsigned char byteCount, unsigned char *field)**
  This function starts an I^2C communication in master-receiver mode with usage of the DMA module. It has the following parameters:
  - **unsigned char byteCount**
    This is the number of bytes that are to be transmitted.
  - **unsigned char *field**
    This is a pointer into an array of values that are to be sent. Since I^2C communication works bytewise, it makes sense to use a field of bytes, for example, unsigned char values.

4.1.3 Functions Without DMA Support (TI_USCI_I2C_master.h)

- **void TI_USCI_I2C_receiveinit(unsigned char slave_address, unsigned char prescale)**
  This function initializes the USCI module for master-receive operation without DMA support. It has the following parameters:
  - **unsigned char slave_address**
    This parameter sets the address of the slave in the communication. The 7-bit slave address is right justified.
  - **unsigned char prescale**
    This parameter sets the desired baud rate. This works in an indirect manner, the resulting baud rate is the quotient of DCO frequency and the prescale parameter.
- **void TI_USCI_I2C_transmitinit(unsigned char slave_address, unsigned char prescale)**
  This function initializes the USCI module for master-transmit operation without DMA support. It has the following parameters:
  - **unsigned char slave_address**
    This parameter sets the address of the slave in the communication. The 7-bit slave address is right justified.
  - **unsigned char prescale**
    This parameter sets the desired baud rate. This works in an indirect manner, the resulting baud rate is the quotient of DCO frequency and the prescale parameter.
- **void TI_USCI_I2C_receive(unsigned char byteCount, unsigned char *field)**
  This function starts an I^2C communication in master-receiver mode without DMA support. It has the following parameters:
  - **unsigned char byteCount**
    This is the number of bytes that are to be received.
  - **unsigned char *field**
    This is a pointer into an array variable that is used to store the received bytes. Since I^2C communication works bytewise, it makes sense to use a field of bytes, for example, unsigned char values.
- **void TI_USCI_I2C_transmit(unsigned char byteCount, unsigned char *field)**
  This function is used to start an I^2C communication in master-transmit mode without DMA support. It has the following parameters:
  - **unsigned char byteCount**
    This is the number of bytes that are to be transmitted.
  - **unsigned char *field**
    This is a pointer into an array of values that are to be sent. Since I^2C communication works bytewise, it makes sense to use a field of bytes, for example unsigned char values.

5 Examples of USCI I²C Master Usage

The following examples use the DMA for I²C communication. If the use of the DMA is not wanted or not possible, the corresponding functions need to be chosen.

The usage of functions with and without DMA is the same. Only the function name differs by the suffix DMA_.

5.1 Receiving n Bytes

```
#include "msp430x26x.h"
#include "TI_USCI_I2C_master.h"

unsigned char array[5] = { 0, 0, 0, 0, 0 };

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                       // Stop WDT

  _EINT();                                        // enable interrupts

  TI_USCI_I2C_DMA_receiveinit(0x48,0x3f);         // initialize USCI and DMA module
  while ( TI_USCI_I2C_notready() );               // wait for bus to be free
  TI_USCI_I2C_DMA_receive(3,array);               // receive the first 3 bytes of
                                                  // array
  LPM0;                                           // put CPU to sleep during
                                                  // communication
}
```

5.2 Transmitting n Bytes

```
#include "msp430x26x.h"
#include "TI_USCI_I2C_master.h"

unsigned char array[5] = { 0x1, 0x2, 0x3, 0x4, 0x5 };

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                       // Disable Watchdog

  _EINT();                                        // enable interrupts

  TI_USCI_I2C_DMA_transmitinit(0x48,0x3f);        // initialize USCI and DMA module
  while ( TI_USCI_I2C_notready() );               // wait for bus to be free
  TI_USCI_I2C_DMA_transmit(3,array);              // transmit the first 3 bytes

  LPM0;                                           // put CPU to sleep during
                                                  // communication
}
```

5.3 Checking Presence of a Slave

This example shows how to check whether or not a slave with a certain address is connected to the I²C bus. This function differs from the functions described in Section 5.1 and Section 5.2, in that it blocks the CPU during its execution and returns whether or not a slave has acknowledged the master.

```
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                          // Stop WDT

  TI_USCI_I2C_transmitinit(transmit_cb,0x48,0x2f);
  _EINT();

  if (!TI_USCI_I2C_slave_present(0x11))              // check for slave
    while (1);                                       // trap cpu if slave with
                                                     // address 0x11 doesn't answer
  LPM0;                                              // Enter LPM0 w/ interrupt

}
```

6 Code Size

Table 1. Code Size (IAR)

Functions	Size Without DMA (Bytes)	Size With DMA (Bytes)
Transmit_Initialize and Transmit	172	254
Receive_Initialize and Receive	210	312

7 References

1. *MSP430x2xx Family User's Guide* (SLAU144)
2. MSP430x261x data sheet (SLAS541)
3. *I²C-Bus Specification and User Manual*, NXP Semiconductors, 2007 (http://www.nxp.com/acrobat/usermanuals/UM10204_3.pdf)
4. *Using the USCI I²C Slave* (SLAA383)

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
Low Power Wireless	www.ti.com/lpw	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2007, Texas Instruments Incorporated