Application Report
SLAA382A–December 2007–Revised March 2015

Using the USCI I²C Master

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

ABSTRACT

This document describes how to use 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 or receiver mode using 7-bit device addressing.

Related code files and additional information can be downloaded from http://www.ti.com/lit/zip/slaa382

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

When using an MSP430 device with peripherals, I²C is often used for communication. There are several MSP430 devices that incorporate a USCI module that supports this communication protocol.

The USCI I²C master function set offers sample code that make I²C 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 I²C communication.

The USCI I²C master package includes functions that support both transmit and receive operations:

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

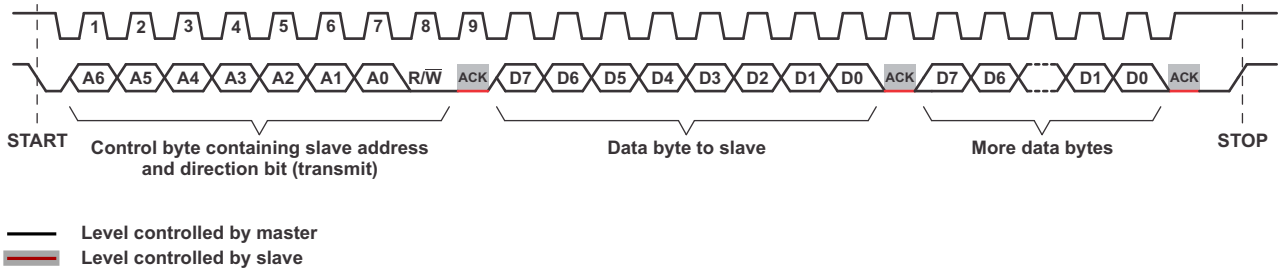

Figure 1. Master Transmitter

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

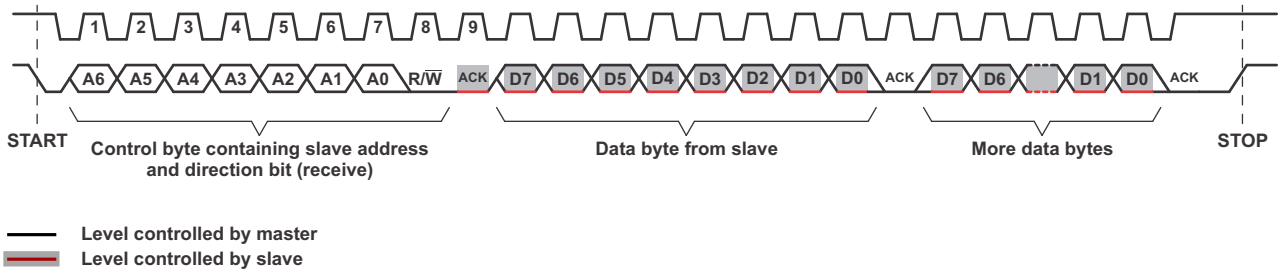

Figure 2. Master Receiver

Both of these functions support only 7-bit addressing.

2 Use 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 I²C communication using only the USCI module, while the second file supports I²C 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 demonstration 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 use 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 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 I²C 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 I²C 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 I²C 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 use of the USCI I²C function set without DMA support is the same as the use 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 use 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 use 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 use 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 use 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 use 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 use 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. *MSP430F241x, MSP430F261x Mixed-Signal Microcontrollers* (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)

Revision History

Changes from Original (December 2007) to A Revision	Page
• Added link to related code files	1
• Corrected Figure 2	2

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

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

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license 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 significant portions 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. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265