Application Report
SLAA734–February 2017

Solutions to Common eUSCI and USCI Serial Communication Issues on MSP430™ MCUs

Caleb Overbay, Ryan Brown *MSP430 Applications*

ABSTRACT

This document is a starting place when facing communication issues on an MSP430™ microcontroller (MCU) and guides the reader through the process of solving their communication challenges. It provides a general description of the most common issues seen when using either the eUSCI or USCI module to perform UART, SPI, or I^2C communication. A brief description of each issue is given, accompanied with solutions and possible debug steps where applicable. For more information on each protocol, see their respective protocol standard documents.

Contents

List of Figures

Trademarks

1 Introduction

Most MSP430 ultra-low-power MCUs enable UART, SPI, and I^2C communication through the use of either the USCI or eUSCI modules. Other legacy devices feature a USI or USART module that does not provide the same functionality. This document focuses exclusively on devices with a USCI or eUSCI module.

Throughout development, users often encounter issues that are common to both modules. However, it is more typical that an issue is specific to the communication protocol being implemented. This application report identifies and provides solutions to problems common to both modules and the specific protocols they implement. Each section starts with a checklist of the most common issues and their solutions, and then provides detailed explanations of more complex topics.

Finally, the USCI and eUSCI modules are similar in nature but have several key differences. *Migrating from the USCI Module to the eUSCI Module* is a separate application report that describes these differences and guides the user when transferring from USCI to eUSCI development.

2 General USCI and eUSCI Issues

The USCI and eUSCI modules are similar in functionality and several common issues affect both. This section describes where to start development as well as how to properly initialize GPIO, debugging clocks, and choose the right clock for the low-power mode your application is using. These are all general issues that are not specific to a communication protocol but the module itself.

Use the following checklist as a starting point when debugging a USCI or eUSCI module:

- ❑ Verify hardware connections between the master and slave devices.
- ❑ Check logic operators (|=, &=, and so on) when setting peripheral registers to ensure proper initialization.
- ❑ Check the correct bit order is being used (MSB or LSB first).

2.1 Starting Development

When beginning development on any communication peripheral, it is highly recommended to start with a TI-provided code example. A wide range of examples, including communication peripherals, can be found in TI's MSP430Ware for MSP Microcontrollers. These are tailored to a large variety of target devices and peripherals, increasing your chances of producing successful code and reducing development time.

2.2 Initializing GPIO Properly

To properly configure GPIO on an MSP430 MCU, several registers may need to be set or reset. The most important pin configuration registers for a communication peripheral are the Function Select register and possibly the Direction register. These are typically denoted as PxSEL and PxDIR in device data sheets and family user's guides.

Before attempting to configure these registers, first refer to the signal description section of your target device's data sheet to find the appropriate pins to be used with your desired communication peripheral. After locating the appropriate pins, locate the *Input/Output Diagrams* section of the data sheet to find the proper settings for the registers described above.

PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS AND SIGNALS[1]		
			P1DIR.x	P1SEL1.x	P1SEL0.x
P1.3/TA1.2/UCB0STE/A3/C3	3	P1.3 (I/O)	I: 0; O: 1	0	0
		TA1.CCI2A	0	0	1
		TA1.2	1		
		UCB0STE	X[2]	1	0
		A3, C3[3][4]	X	1	1
P1.4/TB0.1/UCA0STE/A4/C4	4	P1.4 (I/O)	I: 0; O: 1	0	0
		TB0.CCI1A	0	0	1
		TB0.1	1		
		UCA0STE	X[5]	1	0
		A4, C4[3][4]	X	1	1
P1.5/TB0.2/UCA0CLK/A5/C5	5	P1.5(I/O)	I: 0; O: 1	0	0
		TB0.CCI2A	0	0	1
		TB0.2	1		
		UCA0CLK	X[5]	1	0
		A5, C5[3][4]	X	1	1

Figure 1. Port Pin Function eUSCI Example

Figure 1 shows an example from the data sheet *MSP430FR59xx Mixed-Signal Microcontrollers* with the necessary register settings of PxDIR, PxSEL0, and PxSEL1 when configuring a pin to function with a communication peripheral. See your specific target device's data sheet for a similar table as these settings vary between device families.

2.3 Source Clock and Low-Power Modes (LPMs)

When having difficulty debugging a communication peripheral, it helps to ensure the clock is being sourced from the correct location and the expected frequency is being seen. To check for the correct frequency, an approach similar to the one described in Section 2.2 can be used to output the clock to a GPIO pin. By setting the correct values of PxSEL and PxDIR, a source clock can be output for observation.

PIN NAME (P3.x)	x	FUNCTION	CONTROL BITS AND SIGNALS[1]		
			P3DIR.x	P3SEL1.x	P3SEL0.x
P3.4/TB0.3/SMCLK	4	P3.4 (I/O)	I: 0; O: 1	0	0
		TB0.CCI3A	0	0	1
		TB0.3	1		
		N/A	0	1	X
		SMCLK	1		
P3.5/TB0.4/COUT	5	P3.5 (I/O)	I: 0; O: 1	0	0
		TB0.CCI4A	0	0	1
		TB0.4	1		
		N/A	0	1	X
		COUT	1		
P3.6/TB0.5	6	P3.6 (I/O)	I: 0; O: 1	0	0
		TB0.CCI5A	0	0	1
		TB0.5	1		
		N/A	0	1	X
		Internally tied to DVSS	1		
P3.7/TB0.6	7	P3.7 (I/O)	I: 0; O: 1	0	0
		TB0.CCI6A	0	0	1
		TB0.6	1		
		N/A	0	1	X
		Internally tied to DVSS	1		

Figure 2. Port Pin Function Clock Example

Figure 2 shows another example from the data sheet *MSP430FR59xx Mixed-Signal Microcontrollers*. The figure shows the proper register settings to output the SMCLK on pin P3.4. When the clock signal has been output to the pin, a logic analyzer or oscilloscope can be used to measure the frequency of the signal. See your specific target device's data sheet for a similar table, as these settings vary between device families.

Finally, if your application is operating in a low-power mode (LPM), ensure that the clock being sourced for the USCI or eUSCI module is active in the respective LPM. The target device's data sheet contains information on which clocks are available in each LPM.

Mode	AM		LPM0	LPM1	LPM2	LPM3	LPM4	LPM3.5	LPM4.5	
	Active	Active, FRAM Off[1]	CPU Off[2]	CPU Off	Standby	Standby	Off	RTC only	Shutdown with SVS	Shutdown without SVS
Maximum System Clock	16 MHz		16 MHz	16 MHz	50 kHz	50 kHz	0[3]	50 kHz	0[3]	
Typical Current Consumption, T_A = 25°C	103 µA/MHz	65 µA/MHz	70 µA at 1 MHz	35 µA at 1 MHz	0.7 µA	0.4 µA	0.3 µA	0.25 µA	0.2 µA	0.02 µA
Typical Wake-up Time	N/A		instant	6 µs	6 µs	7 µs	7 µs	250 µs	250 µs	1000 µs
Wake-Up Events	N/A		all	all	LF I/O Comp	LF I/O Comp	I/O Comp	RTC I/O	I/O	
CPU	on		off	off	off	off	off	reset	reset	
FRAM	on	off[1]	standby (or off[1])	off	off	off	off	off	off	
High-Frequency Peripherals	available		available	available	off	off	off	reset	reset	
Low-Frequency Peripherals	available		available	available	available	available[4]	off	RTC	reset	
Unclocked Peripherals[5]	available		available	available	available	available[4]	available[4]	reset	reset	
MCLK	on		off	off	off	off	off	off	off	
SMCLK	optional[6]		optional[6]	optional[6]	off	off	off	off	off	
ACLK	on		on	on	on	on	off	off	off	
Full Retention	yes		yes	yes	yes	yes	yes	no	no	
SVS	always		always	always	optional[7]	optional[7]	optional[7]	optional[7]	on[8]	off[9]
Brownout	always		always	always	always	always	always	always	always	

Figure 3. Active Clocks in Low-Power Modes Example

Figure 3 shows an example from data sheet *MSP430FR59xx Mixed-Signal Microcontrollers* that details the modes in which MCLK, SMCLK, and ACLK are operational. Before selecting a clock source for your USCI or eUSCI module, see the specific device's data sheet for a similar table detailing active clocks in low-power modes. This table can usually be located in the *Detailed Description* section of the data sheet under the subsection *Operating Modes*.

3 Common UART Communication Issues

UART (or Universal Asynchronous Receiver/Transmitter) is a form of serial communication in which the transmitter does not send a clock signal to the receiver. Instead, the two devices agree on timing parameters, including the baud rate, prior to communication and extra bits are added to each word transmitted to synchronize the sending and receiving. This section describes common UART issues on MSP430 MCUs, using UART with a low-power mode and issues that arise when using the peripheral with a terminal application on a PC.

The following list of the most common issues experienced when debugging UART should be given special attention when evaluating a communication issue:

- ❑ Verify both devices agree on presence of a parity bit and number of stop bits
- ❑ Tools for achieving desired baud rate:
  - See the table of common baud rates and settings in the target devices family user's guide.
  - Online tools: Baud Rate Calculator
- ❑ When developing IrDA communication based on UART, see *Implementing IrDA With the MSP430*.
- ❑ Check all peripheral initialization, including:
  - Physical pin selection (RX and TX pins)
  - Communication register settings
  - Enabling of interrupts (both local and global)
- ❑ Ensure you are connecting one device's RX pin to the other device's TX pin and the reverse.
- ❑ Use the tools available (for example, IDE debugger, logic analyzer, and oscilloscope) to prove that both ends are adhering to the UART software protocol.

- ❑ Review the MSP430 device errata sheet for UART errata that could be involved.

3.1 Using UART With a Terminal Application

UART communication is commonly used to interface with a PC and is typically done with the help of a terminal application. Most commonly, the application will interpret the received data as ASCII format and display accordingly. However, there is often a need to view this information in hex, decimal, or even binary format. In this scenario, the user needs to decide whether the MSP430 MCU or the terminal application will convert the data to the appropriate viewing format.

There are a large variety of terminal applications available and each offers varying ways to view the data received. If the terminal application will be performing the conversion, it must have settings that can be configured to view received data in the desired format. This is the simplest solution for solving this issue so choose your terminal program with this in mind.

Conversely, the MSP430 MCU can perform the conversion and then send and receive all data in ASCII format. However, this process can be slow and consume precious time and resources available in your application. Take this into consideration when deciding to use this approach.

3.2 Approaches to Using UART With Low-Power Modes

The asynchronous nature of UART communication can cause issues when an MSP430 MCU is operating in a LPM and attempting to receive data. Depending on the LPM, the USCI or eUSCI module's source clock may be disabled and require activation. A clock activation request is automatically sent by the USCI or eUSCI module when UART is transmitting or when a byte is received. Conversely, when no data is being transmitted or received, the USCI or eUSCI module de-requests the clock, and it can become inactive. Therefore, between each byte received in a LPM, the source clock can become inactive and require an activation request.

The time required to repeatedly activate the source clock can be significant when compared to the configured baud rate and can cause the MCU to miss several bits or even a full byte. The activation time is dependent on several factors including the MSP430 device variant, the selected clock source, and the current LPM. If the DCO is used as the clock source, the activation time is approximately the wake-up time as specified in the device-specific data sheet. The issue can be further exacerbated when sourcing from the DCO across temperature range due to the possibility of frequency drift that is specified in device-specific data sheets.

There are several methods that can be used to avoid missing bits or bytes including:

- Operating at a lower baud rate
- Using a GPIO interrupt to wake device before data transmission starts
- Transitioning to a higher LPM when expecting to receive data through UART
- Utilizing dummy bytes to wake up the MSP430 MCU and maintain active clock throughout data transmission

The following subsections describe these approaches in more detail. When experiencing this issue, see Section 3.3 as well. Erratum UCS6 affects MSP430F5xx and MSP430F6xx devices operating in LPM3 or LPM4 while using UART communication and is directly related to the issue described in this section.

3.2.1 GPIO Interrupt Approach

When needing to maintain a relatively high baud rate and low power consumption, the best method to avoid missing data is to use a GPIO interrupt based approach. Using this method, the transmitting device signals to the MSP430 before data transmission starts. This can be accomplished by using a GPIO on the transmitting device and toggling its logic value. The signal can be received by the MSP430 through a GPIO pin configured to generate an interrupt on a rising or falling edge. Then, the interrupt will wake the device and can be used to activate the appropriate USCI or eUSCI source clock. This method comes with the disadvantage of consuming a GPIO on both the MSP430 MCU and the interfaced device.

3.2.2 Transitioning to a Higher LPM Approach

If the MSP430 is receiving data through UART on regular intervals or in bursts, transitioning to a higher LPM could be a solution to the issues described in this section. For example, if communication is scheduled to occur every 500 ms, a periodic timer interrupt could be used to bring the MCU out of a LPM where the source clock is inactive and into a higher LPM where the source clock is available. Then, at the end of communication, the device could be placed back into a deeper LPM. This approach works best when the communication interval is predictable and is performed in bursts.

3.2.3 Dummy Bytes Approach

Padding communication with dummy bytes can be used to solve the issue of missing bits when receiving UART data while operating in a LPM. Using this method, the device transmitting data to the MSP430 MCU needs to send a dummy byte to start communication. Then, to keep the USCI or eUSCI source clock active, the MSP430 MCU needs to continuously transmit dummy bytes back to the transmitting device until all data has been received. This approach adds extra time to any UART communication in a LPM and requires the transmitting device to ignore any dummy bytes received. Alternatively, the dummy byte can be used to transition the MSP430 MCU to a higher LPM as described in Section 3.2.2.

3.3 Erratum UCS6 Affecting MSP430F5xx/6xx Devices

When UART is idle and the MSP430 MCU is in LPM3 or LPM4, the USCI source clock is supposed to turn off. On older revisions of MSP40F5xx and MSP430F6xx devices, the USCI clock source remains enabled in LPM3 and LPM4 when configured in UART mode and communication is idle. This can lead to higher than expected current consumption and the device also reacts faster to incoming UART communication because the clock is already on. Typically, the device needs to activate the clock when in a LPM before UART receives can properly occur and this adds some delay to the communication.

This erratum has been fixed on newer device revisions. See your device-specific errata sheet for a table describing which device revisions experience the issue and for a workaround. An example of this erratum can be found in *MSP430F5529 Device Erratasheet.*

4 Common SPI Communication Issues

SPI (or Serial Peripheral Interface) is a form of serial communication that synchronously transfers data between devices at speeds up to a few MHz. This protocol has become a popular communication method and learning to use SPI communication with MSP MCUs has become essential for helping engineers develop their applications. This section explains the SPI software protocol, calculating the maximum communication speed, and how to implement a chip select (CS) signal on MSP MCUs.

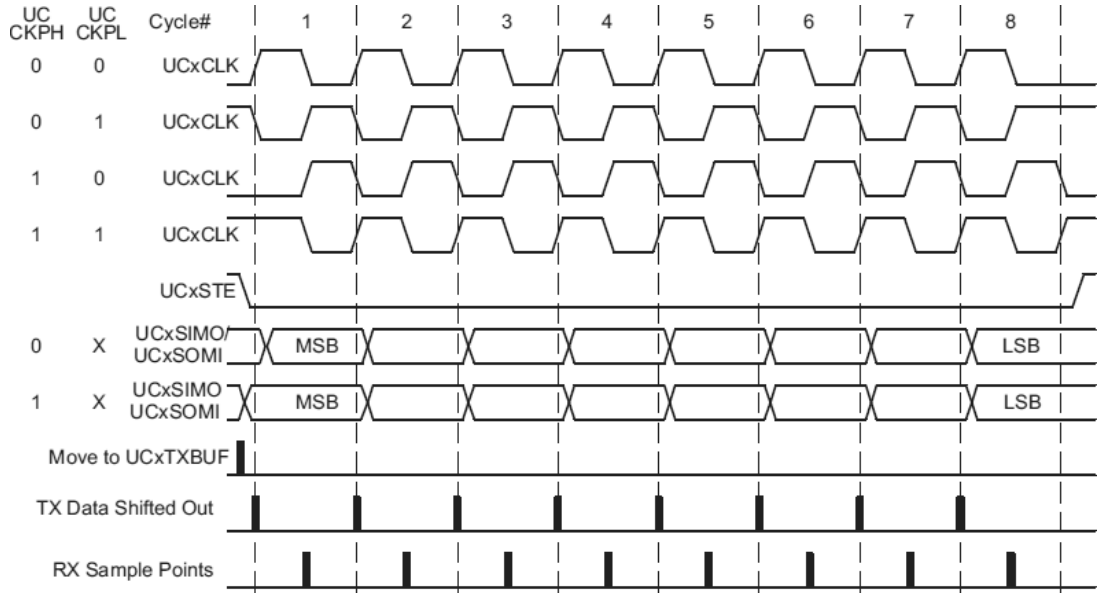
The following encompasses the most common issues experienced when debugging SPI and should be given special attention when evaluating a communication issue:

- ❑ Verify the MSP430 MCU and all slave devices agree on the following:
  - Clock polarity and phase
  - Chip select polarity
  - MSB or LSB first
- ❑ Check all peripheral initialization, including:
  - Physical pin selection (SCLK, MOSI, MISO, and CS)
  - Communication register settings
  - Enabling of interrupts (both local and global)
- ❑ Use the tools available (for example, IDE debugger, logic analyzer, and oscilloscope) to prove that both ends are adhering to the SPI software protocol.
- ❑ Review the MSP430 device errata sheet for SPI errata that could be involved.

4.1 SPI Software Protocol

The SPI protocol uses separate clock and data lines often denoted as Master-In/Slave-Out (MISO), Master-Out/Slave-In (MOSI), and Serial Clock (SCLK). Optionally, a chip select (CS) signal can be implemented, enabling the connection of multiple slaves on the same bus. The master then controls a slave's CS signal to start and stop communication with that device. The master is responsible for setting the clock frequency, polarity (CKPL), and phase (CKPH) as well. When deciding on a SCLK frequency, take into consideration the maximum SPI communication speed of both the MSP430 MCU and the slave. Also, ensure the slave has enough time to process its receive buffer before new data arrives.

The clock polarity is the logic value of the clock when it is idle. For example, if the clock polarity on an MSP430 MCU is 0, when no data is being sent, the clock line is logic 0. The clock phase decides which clock edge the data is captured on. For example, if the clock phase on an MSP430 MCU is 0, the data is captured on the first clock edge, regardless of whether it is rising or falling. When implementing SPI on an MSP430 MCU, ensure all devices involved in communication agree on the clock polarity and phase. Some slave devices follow a different definition for clock phase where CKPH = 0 means data is *changed* on the first clock edge as opposed to *captured*. Always read the definition for how CKPH and CKPL are implemented on a slave device and ensure the matching settings are used on both the master and slave. Figure 4 shows the timing for each case of CKPH and CKPL on an MSP430 MCU.



UCxSTE is unique to MSP430 MCUs. For more information about this signal, see Section 4.3 or your device's user's guide.

Figure 4. Example of MSP430 SPI Timing With MSB First

When communication starts, full duplex data transmission occurs with the master transmitting data on the MOSI line and the slave transmitting data on the MISO line. The master and slave will always send data when the clock is running, regardless of whether they have meaningful data to transmit. Therefore, if the master wants to read data from a slave, it may send command to signify a read, then send an agreed upon number of 'dummy' bytes to receive the information from the slave.

4.2 Calculating the Maximum SPI Communication Speed

When implementing SPI communication on an MSP430 MCU, the timing requirements of both the master and the slave need to be considered. While you can source the clock for the USCI or eUSCI module up to the system frequency, this does not mean that the data is valid at that speed. This depends on several factors including the setup and hold times of both the master and slave. Device-specific data sheets provide a formula for calculating the maximum SPI clock frequency at which data is still valid and it is denoted as f_{UCxCLK}.

To calculate f_{UCxCLK}, first find the section of your device-specific data sheet concerning the SPI timing specifications. This is typically located in the *Specifications* section under the *Timing and Switching Characteristics* subsection. There are two tables with corresponding diagrams that detail the timing characteristics of the MSP430 MCU in both SPI slave mode and SPI master mode. Figure 5 shows an example of the eUSCI SPI master mode timing specifications from MSP430FR59xx Mixed-Signal Microcontrollers.

over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted) (see note (1))

	PARAMETER	TEST CONDITIONS	V_{CC}	MIN	TYP	MAX	UNIT
$t_{STE,LEAD}$	STE lead time, STE active to clock	UCSTEM = 1, UCMODEx = 01 or 10		1			UCxCLK cycles
$t_{STE,LAG}$	STE lag time, Last clock to STE inactive	UCSTEM = 1, UCMODEx = 01 or 10		1			UCxCLK cycles
$t_{STE,ACC}$	STE access time, STE active to SIMO data out	UCSTEM = 0, UCMODEx = 01 or 10	2.2 V, 3.0 V			60	ns
$t_{STE,DIS}$	STE disable time, STE inactive to SOMI high impedance	UCSTEM = 0, UCMODEx = 01 or 10	2.2 V, 3.0 V			60	ns
$t_{SU,MI}$	SOMI input data setup time		2.2 V	35			ns
			3.0 V	35			ns
$t_{HD,MI}$	SOMI input data hold time		2.2 V	0			ns
			3.0 V	0			ns
$t_{VALID,MO}$	SIMO output data valid time(2)	UCLK edge to SIMO valid, C_L = 20 pF	2.2 V			10	ns
			3.0 V			10	ns
$t_{HD,MO}$	SIMO output data hold time(3)	C_L = 20 pF	2.2 V	0			ns
			3.0 V	0			ns

(1) $f_{UCxCLK} = 1/2t_{LO/HI}$ with $t_{LO/HI} = \max(t_{VALID,MO(eUSCI)} + t_{SU,SI(Slave)}, t_{SU,MI(eUSCI)} + t_{VALID,SO(Slave)})$.
For the slave parameters $t_{SU,SI(Slave)}$ and $t_{VALID,SO(Slave)}$, refer to the SPI parameters of the attached slave.

Figure 5. Example of MSP430FR59xx SPI Master Mode Timing Specifications

The current item of interest from Figure 5 is the footnote. As can be seen, the following equation describes the maximum frequency of the clock provided for valid SPI communication:

$$f_{UCxCLK} = \frac{1}{2t_{LO/HI}}$$

$$\text{Where } t_{LO/HI} = \text{MAX}\left(t_{VALID,MO(eUSCI)} + t_{SU,SI(Slave)}, t_{SU,MI(eUSCI)} + t_{VALID,SO(Slave)}\right) \quad (1)$$

For example, assume an application contains two MSP430FR59xx MCUs communicating through SPI. One MCU is the master and the other is the slave. Inserting the necessary numbers from Figure 5 into the equations results in:

$$t_{LO/HI} = \text{MAX}\left(10\text{ ns} + 4\text{ ns}, 35\text{ ns} + 35\text{ ns}\right) = \text{MAX}\left(14\text{ ns}, 70\text{ ns}\right) = 70\text{ ns}$$

$$f_{UCxCLK} = \frac{1}{2(70\text{ ns})} = \frac{1}{140\text{ ns}} \cong 7.14\text{ MHz} \quad (2)$$

In this hypothetical application, SPI communication could operate up to 7.14 MHz while still receiving and transmitting valid data from both devices. When developing your own application, be sure to calculate this maximum speed using the MSP430 and slave devices' SPI timing specifications.

NOTE: These equations calculate the maximum bus speed and not necessarily the maximum data transmission. Each device must be able to fill or empty its SPI buffers at a speed faster than data is being transmitted.

4.3 Various Implementations of Chip Select (CS)

There are several ways to implement a chip select signal on an MSP430 MCU. The USCI or eUSCI module can be configured for 3-pin or 4-pin SPI communication. In some applications, the slave's CS pin can simply be grounded or pulled to VCC. The 3-pin mode implements the MOSI, MISO, and SCLK pin functionalities, leaving the CS functionality to be implemented by the user.

In 4-pin mode, the MCU implements a Slave Transmit Enable (STE) signal. The main purpose of the STE signal is to prevent conflict between multiple SPI masters on the same bus. However, some MSP430 MCUs allow this pin to be configured as an enable signal for a 4-wire slave. This can be accomplished by properly configuring the UCMODEx and UCSTEM bits in the appropriate USCI or eUSCI registers. Note that even though the STE signal can be used to enable a slave during transmission, it is not equivalent to a standard CS signal implementation. The difference is that the STE signal returns to its original logic state between each byte transmitted, and a CS signal maintains polarity throughout transmission until all bytes have been sent or received. See a device's user guide to see if this functionality is available.

The final method of implementing a CS signal is by toggling a GPIO bit when communication starts or ends. Any available GPIO on an MSP430 MCU can be used for this purpose. Always ensure that the slave and master agree on the polarity of this signal as well.

5 Common I²C Communication Issues

I²C (or I2C, for Inter-Integrated Circuit) is a form of two-line bidirectional communication used 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 section explains I²C connections, the software protocol, and implementation advice.

The following encompasses the most common issues experienced when debugging I²C and should be given special attention when evaluating a communication issue:

- ❑ Verify the pullup resistances and slave address value using the slave device data sheet
- ❑ Check all peripheral initialization, including:
  - Physical pin selection (SDA and SCL)
  - Communication register settings
  - Enabling of interrupts (both local and global)
  - When developing SMBUS communication based on I²C, see *SMBUS Design Using MSP430 Design Guide*.
- ❑ Use the tools available (for example, IDE debugger, logic analyzer, and oscilloscope) to prove that both ends of the bus are adhering to the I²C software protocol.
- ❑ Review the MSP430 device errata sheet for I²C errata that could be involved

5.1 The Physical I²C Bus

The I²C bus consists of two wires, SCL and SDA. SCL is the clock line used to synchronize all data transfers whereas 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 pullup 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 V_{CC}. The value of the pullup resistors is traditionally 4.7 kΩ, but this value ranges from less than 1 kΩ to over 10 kΩ, depending on the slave devices used. See the device's data sheet to use the correct pullup resistance. Multiple slave devices can share an I²C bus using single pullup resistors for the SCL and SDA lines. Also, ensure that the total pullup resistance and capacitance on the I²C line is within limits defined in the I²C standard.

5.2 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 the R/W bit low
3. Send the register number
4. Send the data bytes
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, the master resends 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.

For a read, the transaction sequence is as follows:

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

Figure 6 and Figure 7 show the control and data sequence from a MSP430 master to communicate with an I²C slave device during both transmission and reception.

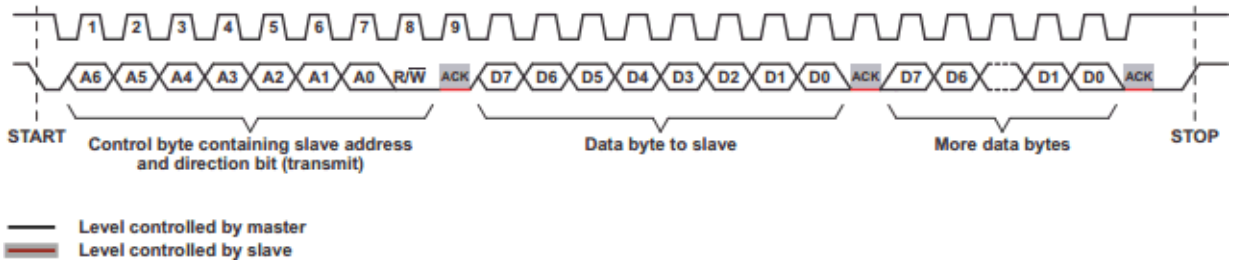


Figure 6. MSP430 Master Transmitter

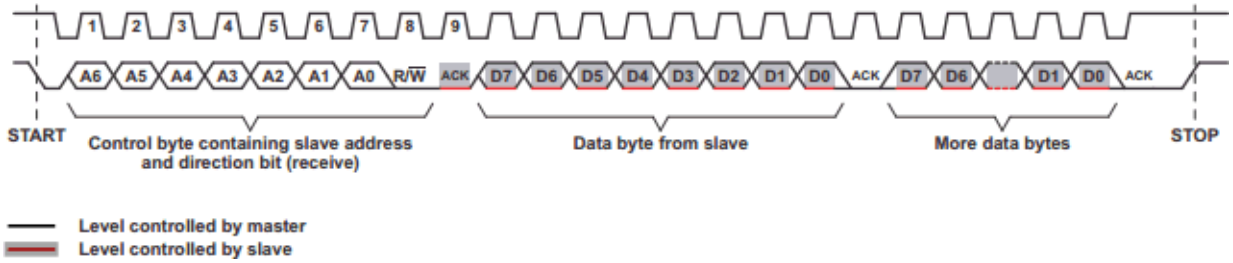


Figure 7. MSP430 Master Receiver

5.3 Tips for Implementing I²C with MSP Devices

When attempting to communicate between a peripheral IC and an MSP MCU 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 (www.ti.com → product page → Tools & software → Software → Examples). Review the changes made to the I²C registers with that of the device 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 pullup resistance and address specified by the slave's data sheet (can sometimes vary based

on input pins and other 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, so the slave address register value needs to be set such that it already represents the 7 most significant bits (MSbs).

- Begin by writing to a register and monitoring the MSP device for an ACK to confirm a valid communication link. Leverage fault flags and bench equipment to alert of communication failure. Use the debugging tools offered by Code Composer Studio™ IDE or IAR Embedded Workbench® IDE to understand how the code operates, what registers are being accessed, and when or how functions or ISRs are accessed. After doing this it will become easier to add functionality for reading from a register.
- The UCTR bit decides whether the R/W bit is high or low and should be determined when setting the UCTXSTT bit, as this command causes the MSP430 device to send the start sequence followed by the slave address and R/W bit automatically.
- The USCI or 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 $N - 1^{th}$ byte. This ensures that the stop sequence is sent immediately after receiving the last byte.
- Research the issue on the TI E2E community forum to see if any similar cases have been solved. Try various combinations of I^2C-related key words and take advantage of the search filters.

6 Tips When Posting to the TI E2E™ Forum

If proper debugging and research methods have failed to find a solution, 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™ development kit or TI target board being used, or schematic if a custom board
- Other devices attempting to be communicated with
- Precise description of issue or problem being seen
- Behavior observed while using the debugger (Code Composer Studio IDE or IAR Embedded Workbench IDE)
- Peripheral initialization and function or ISR code snippets (not full code unless simple)
- Logic analyzer and oscilloscope images with appropriate labels

7 References

1. *Migrating from the USCI Module to the eUSCI Module*
2. MSP430Ware for MSP Microcontrollers
3. *MSP430FR59xx Mixed-Signal Microcontrollers*
4. *MSP430F5529 Device Erratasheet*
5. *Implementing IrDA With the MSP430*
6. *SMBUS Design Using MSP430 Design Guide*

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources 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.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

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