ABSTRACT

The universal asynchronous receiver transmitter (UART) interface and the inter-integrated circuit (I\textsuperscript{2}C) interface are two common serial communication interfaces. They both enable communication between the MSP430™ microcontroller (MCU) and another device, such as a personal computer (PC), another MCU, or a processor. Many devices support only one or the other interface, therefore some designs require communication between devices with these different serial protocols. This application report describe a program that can convert between UART and I\textsuperscript{2}C protocols. In this program, both hardware UART and I\textsuperscript{2}C modes are used, which can support baud rates up to 921600 and I\textsuperscript{2}C clock frequencies up to 400 kHz.

Table of Contents

1 Introduction ............................................................................................................................................................................2
2 Implementation ........................................................................................................................................................................2
3 UART Message Format ........................................................................................................................................................3
  3.1 Write N Bytes to Slave Device ................................................................................................................................................3
  3.2 Read N Bytes From Slave Device .........................................................................................................................................3
  3.3 Repeated Start (Read After Write) .................................................................................................................................4
  3.4 Repeated Start (Write After Write) .....................................................................................................................................4
  3.5 Write to Internal Register ......................................................................................................................................................4
  3.6 Read From Internal Register ..............................................................................................................................................5
4 Internal Registers Available ....................................................................................................................................................5
  4.1 Register Summary ...............................................................................................................................................................5
  4.2 Baud Rate Generator (BRG) .............................................................................................................................................5
  4.3 I\textsuperscript{2}C Bus Clock Rates (I2CClk) ................................................................................................................................6
5 Performance ..............................................................................................................................................................................6
6 Application Examples ..............................................................................................................................................................7
  6.1 Test With I\textsuperscript{2}C Slave Device ..................................................................................................................................7
  6.2 Read and Write EEPROM ..................................................................................................................................................7
7 Reference ..................................................................................................................................................................................9
8 Revision History ......................................................................................................................................................................9

Trademarks

MSP430™ and LaunchPad™ are trademarks of Texas Instruments. All trademarks are the property of their respective owners.
1 Introduction

The MSP430FR2311 MCU has two eUSCI modules that can be used as a low-cost UART-to-I\(^2\)C bridge (configured as an \(I^2C\) master) using the UART and \(I^2C\) modes. This bridge make it convenient for hardware devices that support only UART to access \(I^2C\) protocol devices for data transmission and exchange. To get started, download the project files and a code example that demonstrate this functionality.

2 Implementation

Figure 2-1 shows the block diagram for the UART-to-I\(^2\)C bridge. The MSP-TS430PW20 target development board was used to connect the peripherals to the MSP430FR2311 MCU. Set the jumpers as listed here:

- Populate jumpers JP14, JP15, JP17, and JP18
- Do not populate jumper JP13
- Set jumper JP16 to UART
- Set jumpers JP3 to JP8 all on 1-2
- Set jumper JP11 on 1-3 and 2-4

These jumper settings allow the back-channel UART interface on the MSP-FET programmer and debugger to simulate the UART device that will communicate with the bridge. At the same time, the 10k resistor is used as a pullup on the \(I^2C\) pins. To communication with an \(I^2C\) device, connect the SCL pin of the \(I^2C\) device to J4.19 (P1.3) and connect the SDA pin to J4.20 (P1.2). A simple \(I^2C\) slave project was implemented on an MSP430FR2311 LaunchPad™ development kit to demonstrate the functionality of the UART-to-I\(^2\)C bridge. The UART-to-I\(^2\)C bridge functions in \(I^2C\) master mode.

![UART-to-I\(^2\)C Bridge Block Diagram](image)

NOTE: Pullup resistors are required on the \(I^2C\) bus.

Using a PC, open a new serial connection with a terminal program, and connect to the back-channel UART interface on the MSP-FET by selecting the COM port called **MSP Application UART1**. In the terminal window, change the baud rate to 115200. To demonstrate the functionality of the UART-to-I\(^2\)C bridge, enter a string of bytes into the terminal window follow command formats and send it. It will be sent to the \(I^2C\) slave device, and whatever value was in the TX buffer of the \(I^2C\) slave device will be displayed in the serial terminal.

Figure 2-2 shows the flowchart for the \(I^2C\) and UART code.

When an \(I^2C\) packet is received, the UCRXIFG interrupt flag is set, and the data is read from the \(I^2C\) RX buffer and stored in the internal buffer (default size is 100 bytes). Next when the data is receive completed, write this data to UART TX buffer and send it through the UART.

When a UART packet is received, it is stored in the internal buffer (the default size is 100 bytes), and then the application analyzes the UART packet command format. Different steps are performed for these two command formats: read or write data through the \(I^2C\) interface, or read or write internal register data.
3 UART Message Format

The host initiates an I²C bus data transfer, or reads from and writes to internal registers through a series of ASCII commands. Table 3-1 lists the supported ASCII commands and their hexadecimal value representation. Unrecognized commands are ignored by the device.

Table 3-1. ASCII Commands

<table>
<thead>
<tr>
<th>ASCII</th>
<th>Hex Value</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0x53</td>
<td>I²C start</td>
</tr>
<tr>
<td>P</td>
<td>0x50</td>
<td>I²C stop</td>
</tr>
<tr>
<td>R</td>
<td>0x52</td>
<td>Read from internal register</td>
</tr>
<tr>
<td>W</td>
<td>0x57</td>
<td>Write to internal register</td>
</tr>
</tbody>
</table>

3.1 Write N Bytes to Slave Device

The host issues the write command by sending an S character followed by an I²C bus slave device address, the total number of bytes to be sent, and I²C bus data which begins with the first byte (DATA 0) and ends with the last byte (DATA N). The frame is then terminated with a P character. Once the host issues this command, the MSP430 MCU will access the I²C bus slave device and start sending the I²C bus data bytes.

Note that the second byte sent is the I²C bus device slave address. The least significant bit (W) of this byte must be set to 0 to indicate this is an I²C bus write command.
3.3 Repeated Start (Read After Write)

The bridge also supports ‘read after write’ command as specified in the I²C bus specification. This allows a read command to be sent after a write command without having to issue a STOP condition between the two commands.

The host issues a write command as normal, then immediately issues a read command without sending a STOP (P) character after the write command.

![Figure 3-2. Read N Bytes From Slave Device](image)

3.4 Repeated Start (Write After Write)

The bridge also supports ‘write after write’ command as specified in the I²C bus specification. This allows a write command to be sent after a write command without having to issue a STOP condition between the two commands.

The host issues a write command as normal, then immediately issues a second write command without sending a STOP (P) character after the first write command.

![Figure 3-3. Repeated Start : Read After Write](image)

3.5 Write to Internal Register

The host issues the internal register write command by sending a W character followed by the register and data pair. Each register to be written must be followed by the data byte.

The frame is then terminated with a P character.
The values of the written registers are stored in the internal FRAM of the MSP430 MCU and have a memory function.

### 3.6 Read From Internal Register

The host issues the internal register read command by sending an R character followed by the registers to be read. The frame is then terminated with a P character.

---

### 4 Internal Registers Available

#### 4.1 Register Summary

Table 4-1 lists the internal registers of the MSP430FR2311 that control this application.

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>BRG0</td>
<td>bit7</td>
<td>bit6</td>
<td>bit5</td>
<td>bit4</td>
<td>bit3</td>
<td>bit2</td>
<td>bit1</td>
<td>bit0</td>
<td>R/W</td>
</tr>
<tr>
<td>0x01</td>
<td>BRG1</td>
<td>bit7</td>
<td>bit6</td>
<td>bit5</td>
<td>bit4</td>
<td>bit3</td>
<td>bit2</td>
<td>bit1</td>
<td>bit0</td>
<td>R/W</td>
</tr>
<tr>
<td>0x02</td>
<td>I2CClkL</td>
<td>bit7</td>
<td>bit6</td>
<td>bit5</td>
<td>bit4</td>
<td>bit3</td>
<td>bit2</td>
<td>bit1</td>
<td>bit0</td>
<td>R/W</td>
</tr>
<tr>
<td>0x03</td>
<td>I2CClkH</td>
<td>bit7</td>
<td>bit6</td>
<td>bit5</td>
<td>bit4</td>
<td>bit3</td>
<td>bit2</td>
<td>bit1</td>
<td>bit0</td>
<td>R/W</td>
</tr>
</tbody>
</table>

#### 4.2 Baud Rate Generator (BRG)

This register sets the UART baud rate (the default is 115200). Table 4-2 lists the supported rates.

<table>
<thead>
<tr>
<th>BRG (hex)</th>
<th>UART Baud Rate (dec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0023</td>
<td>35</td>
</tr>
<tr>
<td>0x0046</td>
<td>70</td>
</tr>
<tr>
<td>0x008A</td>
<td>138</td>
</tr>
<tr>
<td>0x0116</td>
<td>278</td>
</tr>
<tr>
<td>0x0682</td>
<td>1666</td>
</tr>
</tbody>
</table>

The rate is programmed through the BRG register and the baud rate can be calculated as Equation 1.

\[
\text{Baud rate} = \frac{16 \times 10^6}{(BRG1, BRG0)}
\]
Note
For the new baud rate to take effect, both BRG0 and BRG1 must be written with new values simultaneously. The new baud rate takes effect immediately after BRG0 or BRG1 are written.

4.3 I²C Bus Clock Rates (I2CClk)
This register sets the serial clock frequency (the default is 100 kHz). Table 4-3 lists the supported serial rates.

<table>
<thead>
<tr>
<th>I2CClk (hex)</th>
<th>I2CClk (dec)</th>
<th>I²C Bus Clock Frequency (kHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0028</td>
<td>40</td>
<td>400</td>
</tr>
<tr>
<td>0x0050</td>
<td>80</td>
<td>200</td>
</tr>
<tr>
<td>0x00A0</td>
<td>160</td>
<td>100</td>
</tr>
<tr>
<td>0x0140</td>
<td>320</td>
<td>50</td>
</tr>
<tr>
<td>0x0280</td>
<td>640</td>
<td>25</td>
</tr>
</tbody>
</table>

The frequency can be determined using Equation 2.

\[
I²C \text{ Bit Frequency} = \frac{16 \times 10^6}{(I2CClkH, I2CClkL)}
\]  

(2)

Note
For the new bit frequency to take effect, both I2CClkH and I2CClkL must be written with new values simultaneously. The new baud rate takes effect immediately after I2CClkH or I2CClkL are written.

5 Performance
The firmware supports full duplex UART communication. It also supports UART packets with eight data bits, least significant bit (LSB) first, no parity bit, and one stop bit. Because two serial interfaces are required, this firmware can only be used on MSP430 MCUs that have at least two USCI modules.

The firmware creates two 100-bytes buffer to hold the data in the conversion. After the data is stored in the buffer, the next conversion is performed. Therefore, the maximum number of bytes sent in a single transmission cannot exceed 100 bytes (including the frame header). If you require packets larger than 100 bytes of data in single transmission, modify the two parameters I2C_TO_UART_BUFF_SIZE and UART_TO_I2C_BUFF_SIZE in the firmware to meet the requirements.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Maximum Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>I²C</td>
<td>400-kHz bit clock</td>
</tr>
<tr>
<td>UART</td>
<td>921600-bps baud rate</td>
</tr>
</tbody>
</table>

To change the baud rate of the UART interface or the clock frequency of the I²C interface, change the UCA0BRW or UCSB0BRW register in the code. In this application, SMCLK operates at 16 MHz. To change this, see the MSP430FR4xx and MSP430FR2xx Family User’s Guide for the proper configuration and the MSP430FR231x Mixed-Signal Microcontrollers datasheet for the maximum I²C clock value. After making these changes, close the serial terminal, rebuild the code, reprogram and reset the MCU, and then reopen the terminal with the new baud rate.

The I²C clock speed and the UART baud rate can also be changed using UART commands (see Section 3 and Section 4). When the baud rate of the UART or the clock frequency of the I²C is changed by the UART, the change takes effect immediately and is saved after power off and restart.
To reduce power consumption while the MCU is not receiving or transmitting data, LPM0 is used. Other low-power modes can achieve lower power consumption, but they might require an external crystal oscillator and can limit the maximum baud rate due to increased wake-up times.

6 Application Examples

6.1 Test With I2C Slave Device

In this example, another MSP430FR2311 is used as the I2C slave device, and the I2C slave program is loaded internally. You can download the slave program for reference, and you need to add RX interrupts and buffers for correct operation. The slave address is 0x48. When receiving a write command sent by the host device, the data is stored in the buffer area. When receiving a read command sent by the host device, the slave device sends incremental data starting at 0x00.

Figure 6-1 shows the serial port command sent by the I2C master to write data and read the data.

In the write example, the UART data command writes 8 bytes to the I2C slave. The command is composed of these bytes:

- The first byte is ‘S’ (0x53) to indicate I2C start
- The second byte indicates the slave address and write command (0x48 << 1 | 0 = 0x90)
- The third byte indicates the length of the write data (0x08)
- The next 8 bytes are the data to write to the slave (0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88)
- The last byte is ‘P’ (0x50) to indicate I2C stop

In the read example, the UART data command reads 8 bytes from I2C slave.

- The first byte is ‘S’ (0x53) to indicate I2C start
- The second byte indicates the slave address and the read command (0x48 <<1 | 1 = 0x91)
- The third byte indicates the length of the read data (0x08)
- The last byte is ‘P’ (0x50) to indicate I2C stop

Figure 6-2 shows the waveform of the I2C write data. From the figure, we can see the start bit, slave address, write command, write data and the end bit in SDA line.

Figure 6-3 shows the waveform of the I2C read data. From the figure, we can see the start bit, slave address, read command, read data and the end bit in SDA line.

Figure 6-1. I2C Write and Read

Figure 6-2. I2C Write Data Waveform

Figure 6-3. I2C Read Data Waveform

6.2 Read and Write EEPROM

This firmware was tested using EEPROM as the I2C slave device to verify the correctness of the firmware by reading and writing the EEPROM. The EEPROM address is 0x50. Figure 6-4 to Figure 6-7 show the read and write format.
The firmware first sets the I²C clock frequency to 100 kHz and sets the UART baud rate to 115200 by writing the internal registers (Figure 6-8 shows the UART commands to read and write the internal register values). For the write test, the firmware writes 64 bytes of data to the EEPROM starting at address 0x0000. For the read test, the firmware reads 64 bytes of data starting at address 0x0000.

Figure 6-8 shows the UART data commands to write to and read from the internal registers.

The write command is composed of these bytes:

- The first byte is 'W' (0x57) to indicate a write to internal registers
- The second byte is the internal register start address to write
- The next four bytes are the data to write
- The last byte is 'P' (0x53) to indicate stop

The read command is composed of these bytes:
The first byte is 'R' (0x52) to indicate a read from internal registers
• The next four bytes are the addresses to read (0x00, 0x01, 0x02, 0x03)
• The last byte is 'P' (0x53) to indicate stop

The final four bytes in the last row of Figure 6-8 are the values of the registers specified in the read command.

Figure 6-8. Write and Read Internal Register (Set I²C Clock Frequency and UART Baud Rate)

Figure 6-9 shows the UART commands to read and write the EEPROM.

The write command is composed of these bytes:
• The first byte is 'S' (0x53) to indicate an I²C start
• The second byte is the EEPROM I²C address and write command (0x50 << 1 | 0 = 0xA0)
• The next three bytes is the length of the write data (0x42), which includes the 2-byte address plus the number of bytes to write (64 in this case)
• The next two bytes are the start address in the EEPROM to write
• The next 64 bytes are the data
• The last byte is 'P' (0x50) to indicate stop

The read command is composed of these bytes:
• The first byte is 'S' (0x53) to indicate an I²C start
• The second byte is the EEPROM I²C address and write command (0x50 << 1 | 0 = 0xA0)
• The third byte is the length of the data (0x02), which is for the 2-byte address
• The next two bytes are the start address in the EEPROM to read
• The next byte is 'S' (0x53) to indicate an I²C restart
• The next byte is the EEPROM I²C address and the read command (0x50<<1 | 1 = 0xA1)
• The next byte is the length of the read data (64 bytes) (0x40)
• The last byte is 'P' (0x50) to indicate stop

Figure 6-9 shows that the read data is the same as the write data.

7 Reference
2. MSP430FR231x Mixed-Signal Microcontrollers data sheet
3. UART-to-SPI Bridge Using Low-Memory MSP430™ MCUs
4. UART-to-UART Bridge Using Low-Memory MSP430™ MCUs
5. Microchip 128Kb I²C compatible 2-wire Serial EEPROM (24LC128)
6. NXP Master I²C bus Controller with UART Interface (SC18IM700IPW)

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

Changes from Revision * (20190912) to Revision A (20210929)  
• Updated the numbering format for tables, figures, and cross references throughout the document.................1
• Corrected Figure 2-2 I²C and UART Code Flow .................................................................2
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.