Application Report
SLAA703A – May 2016 – Revised July 2018

Software I^{2}C on MSP430™ MCUs

MSP430 Applications

ABSTRACT

Hardware and layout limitations often require smaller package sizes that could potentially create a tradeoff for the number of serial peripherals available to the user. With the I^{2}C protocol, special software can be created to use a pair of simple GPIO ports to emulate an I^{2}C master or slave device. This allows programmers to be flexible with pin assignments and enables smaller package devices to overcome the hardware limitation of scarce number of hardware I^{2}C peripherals. Having a software-emulated I^{2}C implementation enables programmers to communicate effectively and fully control attached slave devices without excessive overhead. This application report describes a software-based I^{2}C solution for the MSP430FR2111 microcontroller (MCU), and the implementation can be expanded to work on any MSP430™ MCU with a timer. This software solution supports transactions on I^{2}C bus with an SCL clock frequency of up to 100 kHz for master and slave.

Related software can be downloaded from http://www.ti.com/lit/zip/slaa703.

Contents

List of Figures

Trademarks

1 Introduction

I^{2}C communication protocol is widely used in various applications and some applications even require more than one I^{2}C interface. However, small package, low pin count, and low cost could potentially create a tradeoff for the number of serial peripherals available to the user. For example, the MSP430FR2111 is an FRAM-based low-pin-count (16 pins) microcontroller of the MSP430FRx family. Because of its low pin count, the hardware I^{2}C module is not integrated in this MCU. For this kind of MCU, software I^{2}C using a pair of GPIOs to emulate I^{2}C master or slave is a good choice for the user. Both software I^{2}C master and slave solutions are implemented using a small amount of MCU resources. The interface to I^{2}C bus only uses two GPIO pins, which are quite flexible for users to choose. The software solution supports transactions on the I^{2}C bus with an SCL clock frequency up to 100 kHz for master and slave. The code size is small, so that this solution can also be implemented on low memory footprint MSP430 MCUs.

2 I^{2}C Theory of Operation

2.1 I^{2}C Overview

The I^{2}C bus is a two-wire bidirectional serial bus that requires a serial data (SDA) line and a serial clock (SCL) line to communicate. A pullup resistor is required for each of the lines. When the bus is free, both lines are high. All connected devices can be either a master or slave device. The master device generates the serial clock and initiates communication on the bus. The slave device is addressed by master and responds to communication on the bus. To communicate with a specific device, each slave device must have an address that is unique on the bus. The I^{2}C bus supports either a 7-bit or a 10-bit address mode, allowing up to 128 or 1024 devices, respectively, to be on the bus. The frequency of the I^{2}C serial clock can be up to 100 kHz in standard mode and up to 400 kHz in fast mode.

2.2 I^{2}C Protocol

All I^{2}C transfers begin with a START condition and end with a STOP condition. Figure 1 shows that a START condition is defined as a high-to-low transition on the SDA line while SCL is high, and a STOP condition is defined as a low-to-high transition on the SDA line while the SCL is high. When the START condition occurs on I^{2}C bus, the bus is considered busy and cannot be used by another master until a STOP condition is detected.

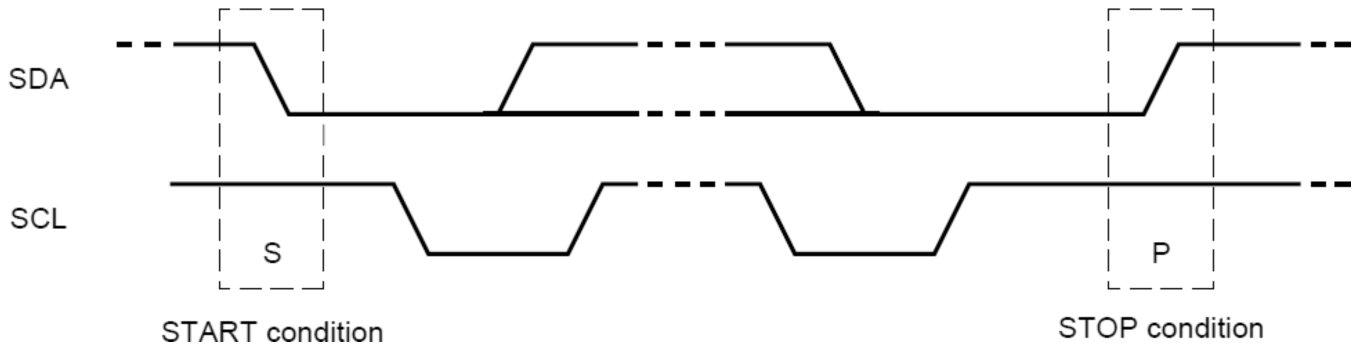


Figure 1. START and STOP Conditions

The START condition is always followed by the address and then by a data direction bit (R/W bit). If the R/W bit is 0, the master will write to the slave device. If the R/W bit is 1, the master will read from slave device. After the R/W bit is sent, the master releases the bus and allows the slave to acknowledge (ACK) the request. The slave device that was addressed acknowledges to the master by holding SDA low for one clock cycle. Then the master or slave transmits data on the SDA line, depending on the R/W bit value. The SDA line can change only when SCL is low, and SDA must be stable when SCL is high. Data on the I^{2}C bus is transferred in 8-bit packets (bytes). There is no limitation on the number of bytes, but each byte must be followed by an ACK bit. If the slave device does not acknowledge transfer, this means that there is no more data or that the device is not ready for the transfer yet. The master device must generate either a STOP or a repeated START condition. For details on the I^{2}C protocol, see the I^{2}C-bus specification and user manual [4].

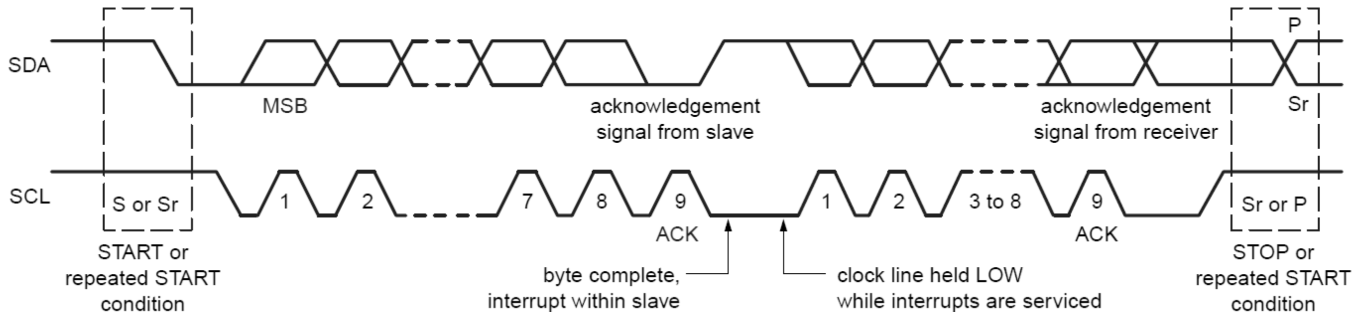

SDA
SCL
MSB
acknowledgement signal from slave
acknowledgement signal from receiver
P
Sr
S or Sr
1
2
7
8
9
ACK
1
2
3 to 8
9
ACK
Sr or P
START or repeated START condition
byte complete, interrupt within slave
clock line held LOW while interrupts are serviced
STOP or repeated START condition

Figure 2. I²C Data Transfer

3 Software I²C Implementation

3.1 Master I²C Software Implementation

3.1.1 Software Implementation

Software implementation of the master I²C requires only two GPIOs and one timer. The two GPIOs are used to emulate SDA and SCL signals. Any ordinary GPIO can be used, and there are no special requirements such as interrupt capability. The configured pins should be reserved exclusively for I²C operation. If another peripheral or function uses or reconfigures the selected pins for a different purpose, the behavior is unreliable. The timer generates the I²C clock. I²C clock frequency relies on the frequency of the timer clock source. Most MSP430 MCUs integrate Timer_A, but the MSP430FR2111 integrates only Timer_B. There is no difference between Timer_A and Timer_B for this software I²C application.

It is easy to generate START and STOP conditions with two GPIOs driving the lines high or low. When the GPIO is set as input, the hardware pullup pulls the line high. When the GPIO is set as output, the GPIO drives the line low. After a START condition, the master transfers the 7-bit address following by one R/W bit and then detects the ACK signal. For bit 1, the SDA line is driven high. For bit 0, the SDA line is driven low. 7 address bits are shifted out one by one. Only 7-bit address mode is supported in the provided code, but users can change it to 10-bit address mode if needed. Write-in data transfer is similar to address transfer that the master drives SDA line high or low for each bit. Read-out data transfer is different in that the master listens to the SDA line and shifts bit 1 or 0 into thr receive data buffer. The number of transfer bytes is controlled by the master.

3.1.2 Functions Description

The following functions are defined in the master I²C software code (fr2111_swi2c_master.c).

void SWI2C_initI2C(void)

This function initializes the software I²C master. It configures two selected GPIOs to output high on SDA and SCL lines. For MSP430FR4x2x family MCUs, the GPIO power-on default high-impedance mode should be disabled to activate configured GPIO settings. This function also initializes Timer CCR register for SCL clock frequency setting.

bool SWI2C_writeData(…)

This function performs write-in transaction. It sends out address and data to the slave. It also handles ACK signal from the slave. If there is no ACK, it will stop the transaction. The parameters passed as arguments are uint8_t addr, uint8_t *outputArray, uint_fast16_t size, and bool sendStop. Parameter *addr* is the slave address. The pointer *outputArray* points to the memory location the data to be sent is stored. Parameter *size* is the number of data bytes to be sent. Parameter *sendStop* controls whether to send out STOP condition or not.

bool SWI2C_readData(…)

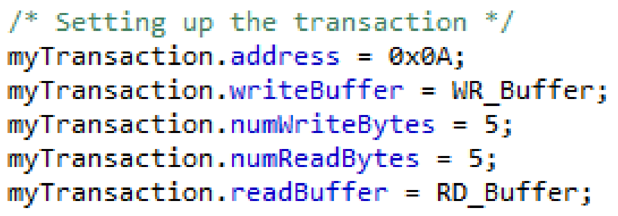
This function performs read-out transaction. It sends out address to the slave and receives data from the slave. It not only detects ACK of the address but also sends out ACK for received data to the slave. The parameters passed as arguments are uint8_t addr, uint8_t *inputArray, and uint_fast16_t size. Parameter *addr* is the slave address. The pointer *inputArray* points to the memory location the received data is stored. Parameter *size* is the number of data bytes to be received.

bool SWI2C_performI2CTransaction(…)

This function performs write-in transaction firstly and then performs read-out transaction. In this function, SWI2C_writeData function and SWI2C_readData function are called as subfunctions. The parameter passed as argument is SWI2C_I2CTransaction *i2cTransaction. *SWI2C_I2CTransaction* is the configuration structure for performing an I^{2}C transaction. This structure is used to pass parameters to SWI2C_writeData and SWI2C_readData functions.

3.1.3 Software Example Code

This application report provides software example code for software master I^{2}C in folder FR2111_SW_I2C_Master of associated source. There are three files called FR2111_SW_I2C_Master_main.c, fr2111_swi2c_master.c, and fr2111_swi2c_master.h. fr2111_swi2c_master.c contains the functions discussed in previous paragraph. fr2111_swi2c_master.h contains functions and structure declarations and pin definitions. User experience code can be added in FR2111_SW_I2C_Master_main.c. In this example code, the master writes 5 data bytes into the slave and then reads 5 data bytes from the slave. Slave address is 0x0A. Figure 3 shows the configuration structure.



```
/* Setting up the transaction */
myTransaction.address = 0x0A;
myTransaction.writeBuffer = WR_Buffer;
myTransaction.numWriteBytes = 5;
myTransaction.numReadBytes = 5;
myTransaction.readBuffer = RD_Buffer;
```

Figure 3. Structure Configuration

To migrate this example code into other application code, the configuration structure must be modified to set slave address and data length. Users also need set target I^{2}C clock frequency which is controlled by parameter SWI2C_TIMER_PERIOD in fr2111_swi2c_master.h. By changing the pin definitions, this master I^{2}C can be ported to any other GPIO. Besides pin definitions, the GPIO power-on default high-impedance mode should be disabled to activate configured GPIO settings in MSP430FR4x2x MCUs. For some MCUs that integrate only Timer_A, migration from Timer_B to Timer_A should be completed. If the timer is already used for another function, thr real-time clock (RTC) counter can also be used to generate I^{2}C SCL clock. This software master I^{2}C solution requires a small memory size (approximately 1KB FRAM and 160 bytes SRAM) so that it can be easily migrated to other low-end MSP430 MCUs.

3.2 Slave I²C Software Implementation

3.2.1 Software Implementation

This application report implements the software slave I²C solution based on a state machine. I²C transaction is divided into different states. Two GPIOs with interrupt capability are required to monitor the SDA and SCL lines. In the provided software slave I²C example code, P1.0 pin is defined as the SCL pin, and P2.0 pin is defined as the SDA pin. The state machine is managed in interrupt routines, and both port 1 and port 2 interrupt vectors are used.

When the master sends the address to the slave, the I²C ISR executes the corresponding sequence of states to process the master write or read request. The same software routine is used to process both addresses and write-in data sent by the master. A different sequence of states is used to response to read command from the master. There are 14 states defined in the example code. Table 1 lists all the states and the description of each state.

Table 1. State Description

State	Triggering Edge	Description
I2C_START	↓	Enable SCL INT, set the rising edge trigger of SDA.
I2C_STOP	↑	Reinitialize GPIOs for START detection. Reset status for address detection.
SCL_W1LH	↑	The most-significant bit, bit 7, is detected and shifted into buffer. Enable SDA rising edge INT, which enables STOP condition detection.
SCL_W1HL	↓	Disable SDA INT, which disables STOP condition detection.
SCL_W2to6LH	↑	Bit 6 to bit 2 for address and write-in data are shifted into buffer.
SCL_W7LH	↑	Bit 1 is shifted into the buffer and the address is compared.
SCL_W8LH	↑	Bit 0 detect; R/W bit detect; normal write-in data byte received.
SCL_W8HL	↓	Set SDA output for ACK. If a write command is received, record data in buffer. If a read command is received, go to read-out data states.
SCL_W9HL	↓	Release SDA line.
SCL_R1HL	↓	Set the following SCL ISR trigger to be rising edge. Send out the first bit.
SCL_R1LH	↑	SDA rising edge INT is enabled, which enables the detection of a STOP condition. The SDA INT is enabled until SCL sends out a falling edge. The state SCL_R2to8HL disables the SDA INT and continues to read out data. This state provides an alternative way to stop communication with ACK at the end.
SCL_R2to8HL	↓	Rotate one bit out of a data byte. Set the SDA output according to the bit.
SCL_R9HL	↓	Release control of SDA. Set the rising edge trigger of SCL, prepare for detection of the ACK bit.
SCL_R9LH	↑	Detect the ACK/NACK bit. If ACK is received, falling edge of SCL is set. Next data is loaded. If NACK is received, indicating the end of data reading, SCL triggering is set to rising edge, and next state is set to SCL_W1LH, which enables detection of a STOP condition

3.2.2 State Machine Description

Figure 4 shows the state machine of the software I²C slave. All states are handled in an interrupt routine. In I2C_STOP state, the SDA and SCL pins are reinitialized to detect a START condition. At the end of a write or read transaction, I2C_STOP is entered. If the received address does not match the slave address, I2C_STOP is entered. Address is detected in SCL_W7LH state. State SCL_W8HL checks the R/W bit and goes to the corresponding sequence of states for write or read transaction. SCL_W2to6LH and SCL_R2to8HL are repeated states used to deal with similar bits of the data.

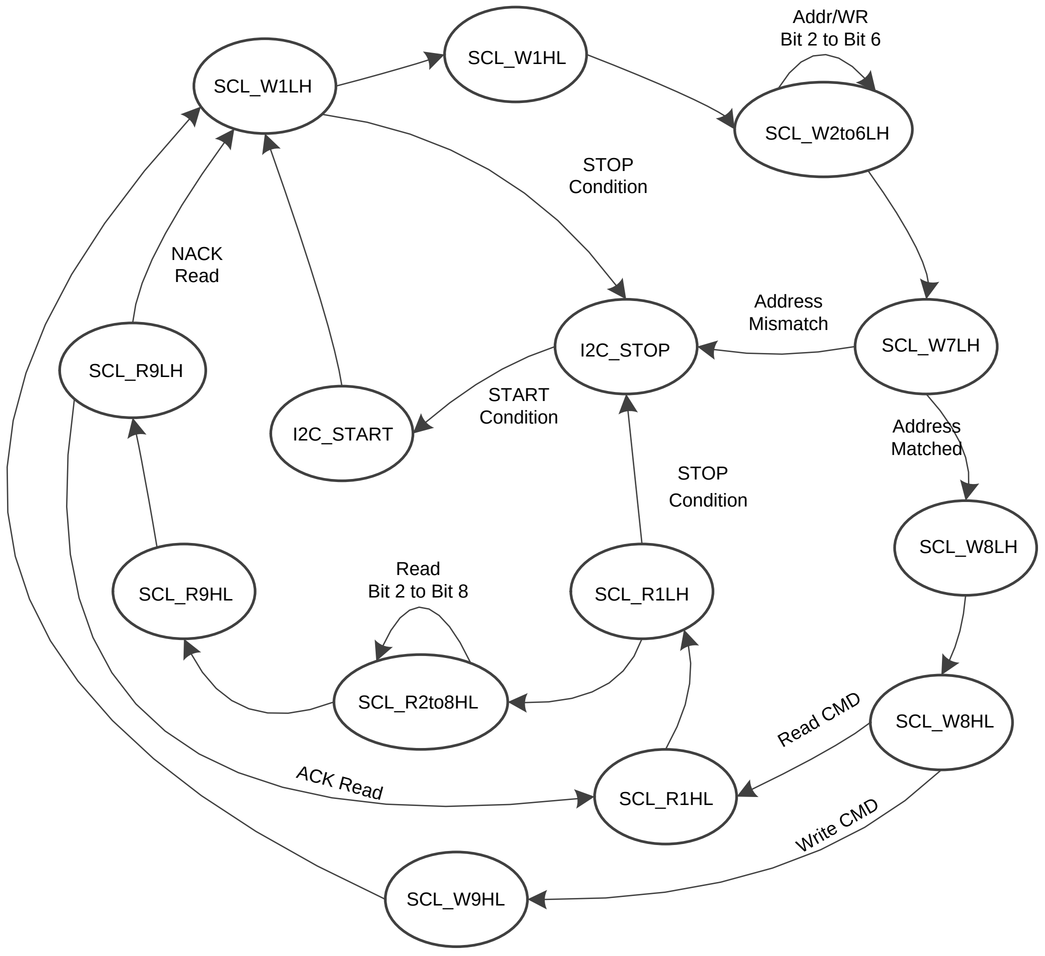


Figure 4. Slave I²C State Machine

3.2.3 Software Example Code

The software slave I²C state machine is implemented with both C code and assembly code. It is easier to migrate the C code into user application code, but assembly is more efficient and can achieve higher clock frequency of the slave I²C. The C code is FR2111_SW_I2C_Slave.c in folder FR2111_SW_I2C_Slave. The assembly code is FR2111_slave_i2c_isr.s43 in folder FR2111_SW_I2C_Slave _withAssemblyCode. This assembly code is packaged as function INIT_I2C and called by the C code FR2111_SW_I2C_Slave_withASMcode.c . This software slave I²C can receive data from master and send out data to master. A 16-byte buffer is defined in the code for read and write transactions.

To migrate this example code into other application code, the macro definition I2COA must be modified to set the slave address. Pin definitions SDA and SCL are used to select GPIOs for I²C function. In the MSP430FR2111 MCU, P1.4, P1.5, P1.6, and P1.7 do not support interrupts, so these four pins cannot be selected for slave I²C.

In the software slave I²C example code, MCLK frequency is set at 8 MHz. With 8-MHz MCLK, software slave I²C can achieve 100-kHz SCL clock frequency using assembly code and 43-kHz SCL clock frequency using C code. Higher SCL clock frequency can be achieved with higher frequency MCLK. The FRAM memory size for the C code and the assembly code is different. The C code is approximately 1.5KB of FRAM, while the assembly code is less than 1KB of FRAM. The SRAM size cost for both C code and assembly code is approximately 180 bytes. Users can choose assembly code or C code based on application requirements.

4 Testing Software I²C Master and Slave

4.1 Hardware Setup

To validate this software I²C solution, a hardware I²C device is required to communicate with the software I²C device. This application report uses the MSP430FR2311IPW16, which integrates a hardware I²C module, to communicate with MSP430FR2111IPW16. The test platform is based on two MSP-TS430PW20 target socket boards, which support both MSP430FR2311 and MSP430FR2111. A logic analyzer is used to measure I²C communication waveform. Figure 5 shows the test platform.

The MSP-TS430PW20 target socket board has two 10-kΩ pullup resisters (R7 and R15) for I²C communication. To enable these two pullup resistors, set jumper JP16 to the I²C side instead of the UART side. Figure 5 shows how to set jumpers JP17 and JP18. The MSP-TS430PW20 target socket board supports both PW20 and PW16 packages. Set jumper J11 to the PW16 side, because PW16 package MCUs are used. The two boards are connected by four wires: two wires for I²C pins and two wires for power supply and ground pins. See the *MSP430 Hardware Tools User's Guide* for MSP-TS430PW20 target socket board schematic and layout.

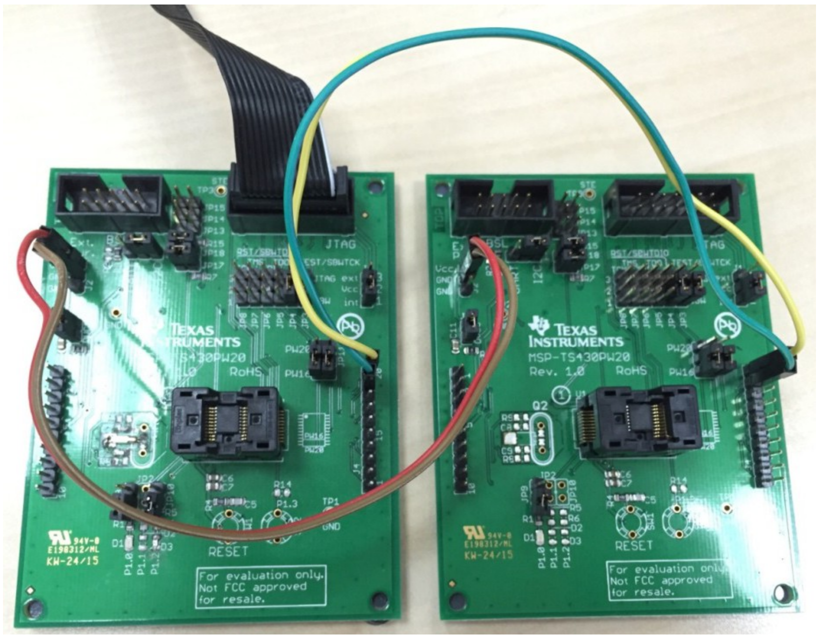


Figure 5. Test Platform

4.2 Software Setup

This application report provides both MSP430FR2111 software I²C codes and MSP430FR2311 hardware I²C codes for test. There are five folders that contain I²C source codes for different MCUs in different I²C modes. MSP430FR2111 code should be used with MSP430FR2311 code by pairs to include one I²C master and one I²C slave.

For the test of MSP430FR2111 software I^{2}C master mode, code in FR2111_SW_I2C_Master folder and FR2311_HW_I2C_Slave folder should be programmed into the target MCUs. Figure 6 shows the software master I^{2}C waveform measured by a logic analyzer. As shown in the waveform, MSP430FR2111 writes five data into the slave with address 0x0A and then reads out five data from the slave.

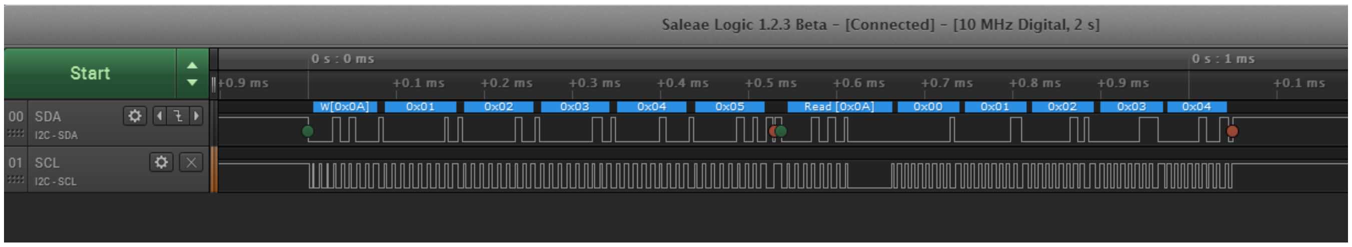


Figure 6. Software Master I^{2}C Waveform

For the test of MSP430FR2111 software I^{2}C slave mode, users can choose C code in folder FR2111_SW_I2C_Slave or assembly code in folder FR2111_SW_I2C_Slave _withAssemblyCode. Use the MSP430FR2311 as the hardware master device. The hardware master code is in folder FR2311_HW_I2C_Master. Figure 7 shows the software slave I^{2}C waveform measured by the logic analyzer. As shown in the waveform, MSP430FR2111 receives four data from the master and then sends out five data to the master.

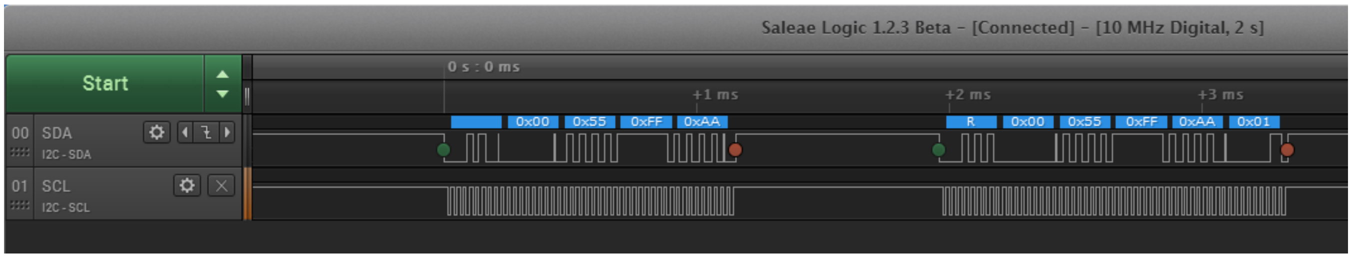


Figure 7. Software Slave I^{2}C Waveform

The C code included with this application report compiles without issue on different IDEs such as Code Composer Studio™ IDE and IAR Embedded Workbench® IDE. The provided assembly code supports only IAR Embedded Workbench IDE.

5 Conclusion

This application report describes a method to implement software I^{2}C master and slave functionality on an MSP430FR2111 MCU. Both software I^{2}C master and slave functionality are implemented using a small amount of MCU resources. The software slave I^{2}C uses only two GPIOs to emulate the SDA and SCL lines. For software master I^{2}C, just one timer is used to generate the I^{2}C SCL clock in addition to the two GPIOs. The provided code supports transactions on the I^{2}C bus with an SCL clock frequency up to 100 kHz for software master and software slave when using an 8-MHz MCLK. Higher SCL clock frequencies can be achieved with higher MCLK frequencies. The code size is approximately 1KB of FRAM and less than 200 bytes of SRAM for both software master and software slave. Because of the small resource cost, this software I^{2}C solution can also be implemented on other low-end MSP430 MCUs.

6 References

1. MSP430FR4xx, MSP430FR2xx Family User's Guide
2. MSP430FR211x Mixed-Signal Microcontrollers
3. MSP430 Hardware Tools User's Guide
4. I^{2}C-bus specification and user manual (http://www.nxp.com)

Revision History

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

Changes from May 19, 2016 to July 2, 2018	Page
• Changes throughout document to describe both I^{2}C master and slave implementation	1