Freescale Semiconductor
Application Note

Document Number: AN4789
Rev 1, 3/2014

Generic Timer Module (GTM) Inter-Integrated Circuit (I2C) Bus Emulation

by: Inga Harris

Contents

1 Introduction

This Generic Timer Module (GTM) Inter-Integrated Circuit (I2C) Bus Emulation application note is intended to provide details of how to emulate I2C bus Serial Data (SDA) and Serial Clock (SCK) output signals using the GTM Multi-Channel Sequencer (MCS) submodule and the ARU connection Timer Output (ATOM) submodule. The assembly functions are portable to any product that has a GTM module. Porting the application code, which configures the chip and GTM, from one chip to another does require minor changes. Example code in this application note is based on the MPC5777M device. This application note should be read in conjunction with application note *AN4351, "MPC57xxM Generic Timer Module (GTM) Quick Start Guide."*

2 Overview

The I2C bus was designed by Philips (now NXP Semiconductors®) in the early 1980s to allow easy communication between components that reside on the same circuit board. It is now a common communication protocol used in many embedded applications.

An I2C interface is commonly emulated in software where a dedicated hardware peripheral is not available. The solution presented in this application note emulates the interface in the GTM MCS module with only a small amount of software running on the chip core, which means that the emulated interface does not consume CPU bandwidth and only consumes one MCS channel with two ATOM channels.

This example is for a single-master implementation, where the MPC5777M is the single master as the I2C bus does not need to do arbitration detection. The single-master implementation is often used in chip to external EEPROM memory configurations. A multi-master system is not implemented because it is more complicated to implement all the bus timing specifications because start and stop bit detection would be needed, involving the additional of use of a TIM submodule.

The given example transmits a 7-bit address followed by 16 bits of data (divided into 8-bit bytes) with a few control bits for communication start, end, direction, and acknowledgement as shown in Table 1. The address and data information is stored in the MCS RAM at compile time for simplicity, but it could also be moved to the RAM through DMA or read by the MCS through the PSM submodule in a full application environment.

Table 1. Message format

	START	Slave address	R/W	ACK	Data	ACK	Data	ACK	STOP
Length	1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit
SDA	Falling Edge	ADDR	R = High W = Low	Release / High	DATA	Low	DATA	Release / High	Rising Edge
SCL	High	Pulse	Pulse	Pulse	Pulse	Pulse	Pulse	Pulse	High

The transmission of ADDR and DATA is controlled by the master. The SDA signal can only be changed when the SCL signal is low. When the master sets the SCL signal high, the slave samples the SDA signal.

The figure below shows an example transmission from the system:

- ADDR = 0x70
- WRITE
- DATA = 0x0114

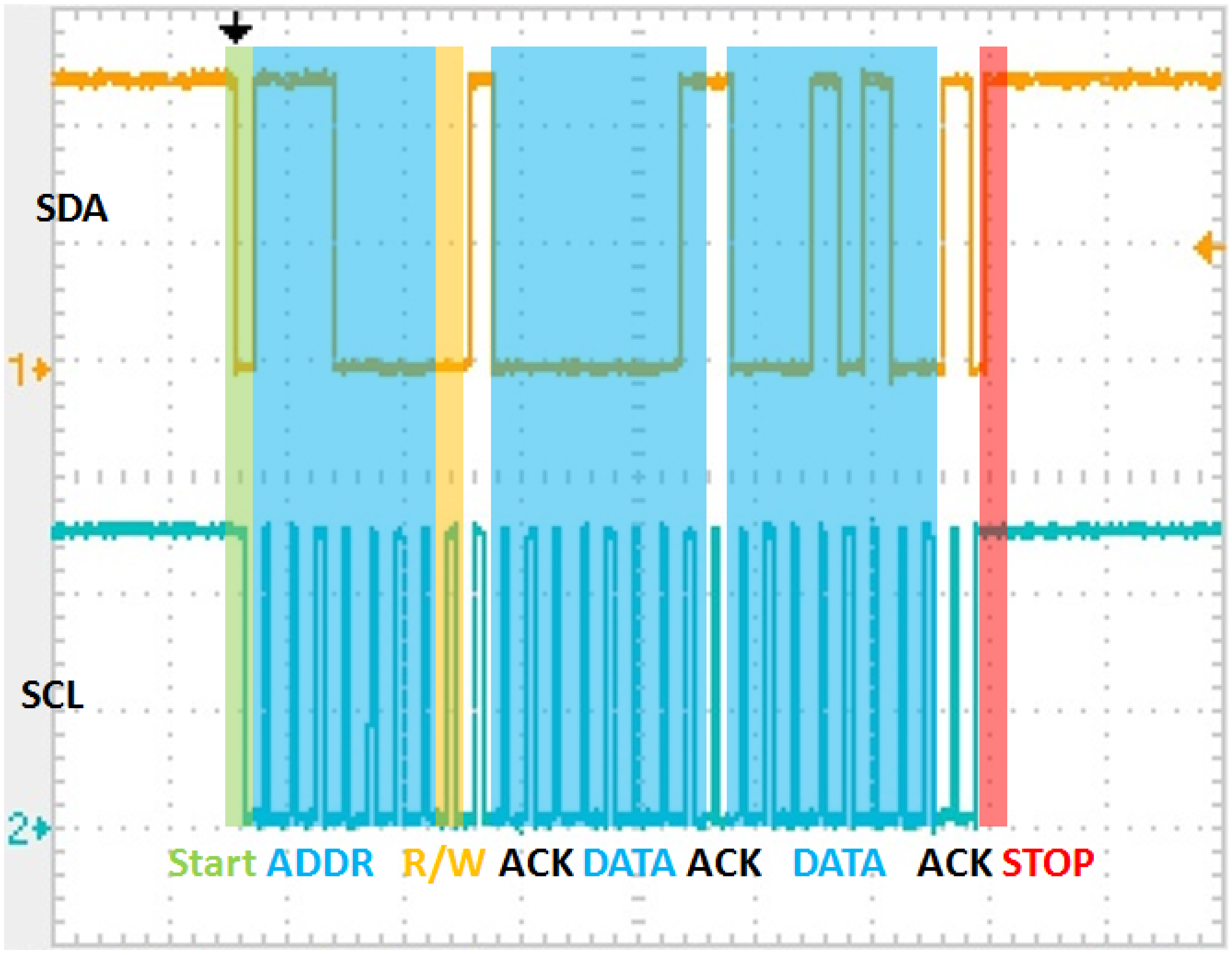


Figure 1. Example output waveform

NOTE

As this example does not require a slave to be connected to the master, the lack of the slave driving the SDA line low during the ACK clock is ignored.

3 Block Diagram

The I2C solution given in this application note uses the following GTM104 submodules:

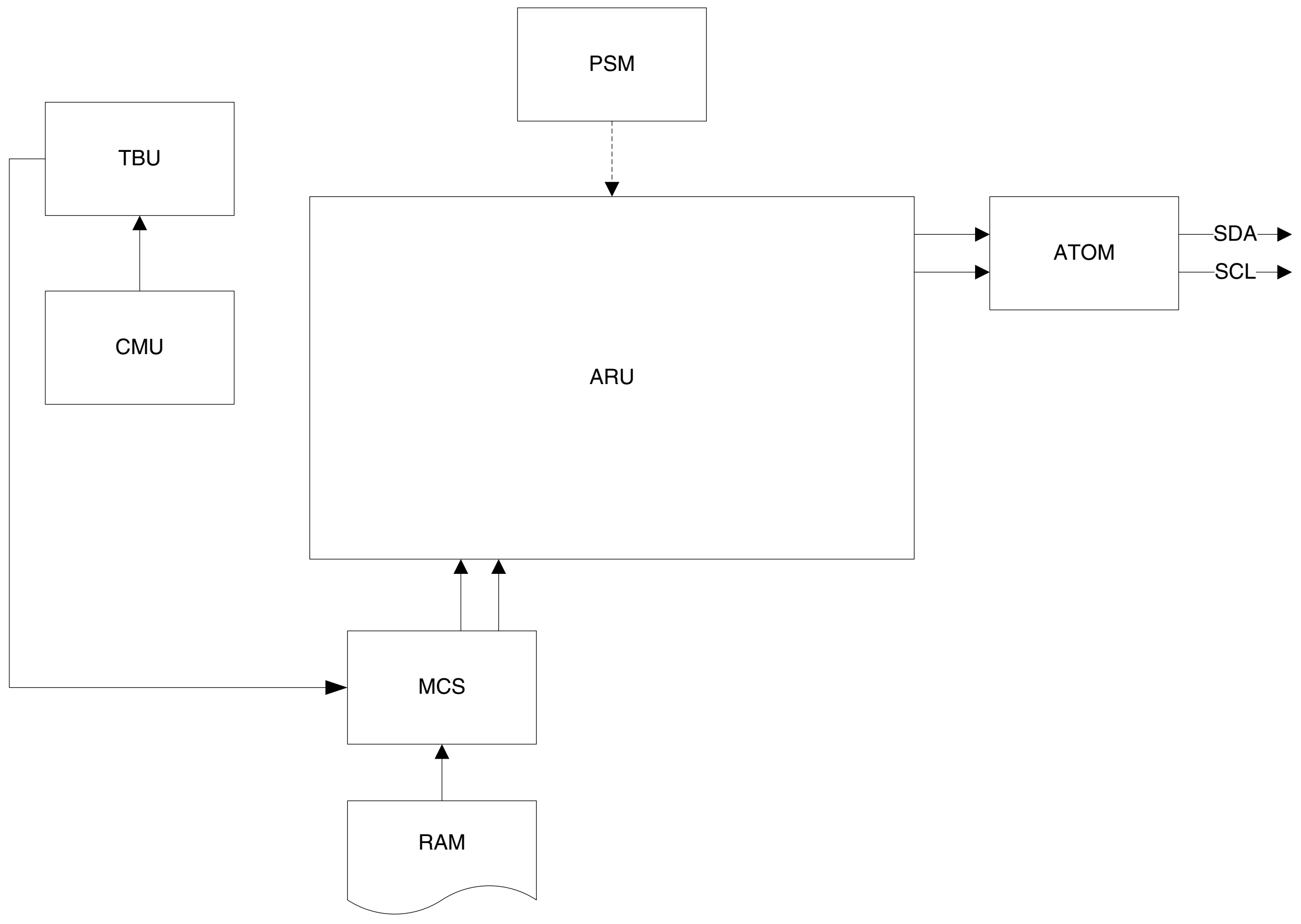


Figure 2. GTM configuration for I2C

Submodule	Purpose	Use case
Clock Management Unit (CMU)	Generates all of the clocks and counters for the GTM subsystem.	Controls the system clock speed from the chip level clocks.
Time Base Unit (TBU)	Provides a common timebase that can be used throughout the GTM subsystem.	TBU_TS0 uses CMU_CLK0 as source for the GTM global timebase.
Multi Channel Sequencer (MCS)	A generic data processing module that is connected to the ARU. It allows "programs" to be written to calculate complex output sequences that depend on timebase values.	MCS0 software state machine that calculates the timebase's values and output states for the ATOM channels.
Advanced Routing Unit (ARU)	Provides a mechanism for routing streams of data between data sources and transfer it to a destination. This is the heart of the GTM subsystem.	Complex output waveforms for the SDA and SCL as instructed by the MCS through the ARU.
ARU connected Timer Output Module (ATOM)	Capable of generating complex output signals through its interconnectivity with the ARU to other modules in the GTM subsystem.	ATOM0, channels 4 (SDA) and 5 (SCL) in SOMC mode reading values from MCS0.

Figure 2 also shows the Parameter Storage Module (PSM) which could be used to bring in DATA to the MCS from another peripheral in the MPC5777M device, such as an ADC or sensor. The MCS could also get its data directly from its RAM from another peripheral through DMA.

4 Chip Level Software Description

The configuration of the chip modes and clocks, and the GTM at a basic initialization is as described in AN4351. The specific configuration of the GTM submodules (TBU, ATOM, and programming of MCS RAM) is shown in Appendix B.

The MCS array, `MCS0_MEM`, contains both the data and the software for the I2C bus emulation. Only MCS0 channel 0 is used for the calculation; however, the channels write to two ARU ports for the SDA and SCL commands to be consumed by two ATOM channels. To start the I2C bus output after initialization, the MPC5777M core and the MCS do a handshake with the MCS's trigger mechanism as shown below. The MCS's half of the handshake can be seen in the following section describing the MCS assembly program operation.

```
/* Start the MCS Program */
GTM_MCS_0.CH0_CTRL.R = 0x00000001; // Enable Channel 0 of MCS0

/*Check that the ATOM channels are ready, MCS_STRG is set */
while((GTM_MCS_0.STRG.R & 0x2) == 0);

/*Next Trigger for MCS to signal "Port config finished"  */
GTM_MCS_0.STRG.R = 0x00000001;
```

When this handshake is complete, the MCS is running in an infinite loop.

5 MCS Software Description

The MCS's program and data must be written and pre-compiled before loading in to the MCS RAM block.

As described in the "Example 7: Writing, Compiling, and Programming MCS Code" section of the previously mentioned application note, AN4351, the structure of the assembly code includes some definitions, initialization of start addresses for each active channel, and initialization of data and stacks, followed by the subroutines themselves.

Figure 3 and Table 2 describe the general functionality of the MCS assembly code.

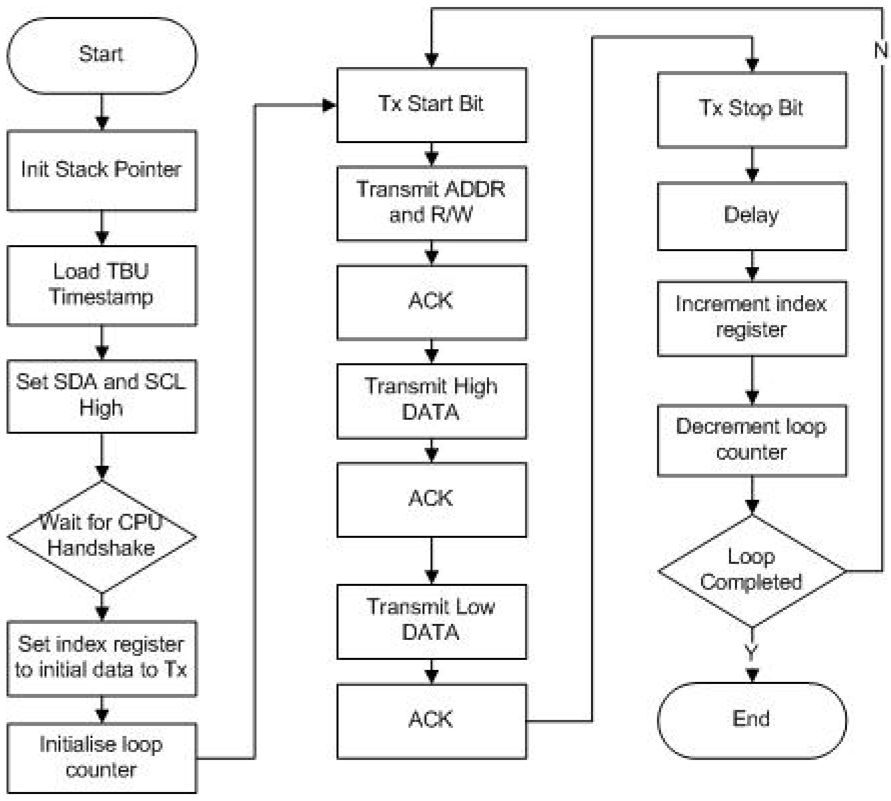


Figure 3. MCS code flow chart

Table 2. I2C bus emulation code blocks description

Step	Operation	Description	Code example / Section for further details
1	Start	After Reset the MCS channel program counter is at address 0 and must be moved to the start of the code.	JMP tsk0_init
2	Init stack pointer	Initialize the stack pointer to the start of the reserved memory space.	MOVL R7 0x000020
3	Load TBU time stamp	Read the current value of the TBU timestamp in to the MCS register, R3.	MOV R3 TBU_TS0
4	Set SDA and SCL high	Set both ATOM channels high, which is the I2C bus default state.	Set SDA and SCL high
5	Wait for CPU handshake	Handshake with the chip core to ensure the system is fully initialized.	Handshake with CPU
6 (**start_tx**)	Load message (index register)	Direct the MCS index register, R6, to the message to be transmitted.	MOVL R6 message_array
7	Initialize loop counter	Set the loop counter such that all the messages in the message array are sent and then repeated.	MRD R1 tsk0_counter

Table continues on the next page...

Table 2. I2C bus emulation code blocks description (continued)

Step	Operation	Description	Code example / Section for further details
8	Send Start Bit	Transmit the Start Bit (falling edge) on ATOM0 CH4 while ATOM0 CH5 is high.	Send Start bit
9	Send ADDR and R/W	Send the 8 bit ADDR and R/W contents of the memory address pointed by index register.	Send message
10	Wait for ACK	Send one clock pulse while the SDA line is high.	Acknowledgement from slave
11	Send 1st byte of DATA (high)	Send the next 8 bits which is the upper byte of the DATA.	Send message
12	Wait for ACK	Send one clock pulse while the SDA line is high.	Acknowledgement from slave
13	Send 2nd byte of DATA (low)	Send the next 8 bits which is the lower byte of the DATA.	Send message
14	Wait for ACK	Send one clock pulse while the SDA line is high.	Send message
15	Send Stop Bit	Transmit the Stop Bit (rising edge) on ATOM0 CH4 while ATOM0 CH5 is high.	Send Stop bit
16	Delay Routine	Pause between data transmissions.	MCS delay routine
17	Increment index register	Move the index register to the next message to be transmitted.	ADDL R6 0x000004
18	Decrement loop counter	Adjust the loop counter for the completed message.	SUBL R1 0x000001
19	Loop Completed?	Test R1 for zero.	JBC STA Z next_message
20	End	In this example the message array repeats from the start.	JMP **start_tx**

The Acknowledging data from slave and Send repeated Start bit routines described in this application note are not used in this basic example, which does not assume that any slaves or other masters are connected to the I2C bus.

The routines can be configured to use different ARU ports and ATOM channels, to send different ARU commands, and to point to different message address spaces by altering the definitions at the start of the assembly file. This examples uses the configuration listed below.

```
.set ARU_PORT0, 0x0000
.set ARU_PORT1, 0x0001
.set PIN_HI, 0x000009
.set PIN_LO, 0x00000A
.set ATOM0_CH4, 0x0123
.set ATOM0_CH5, 0x0124
.set message_array, 0x70
```

There are also variables that are set up in the assembly code and referred to through the routine for the number of messages to be transmitted, the clock rate, and the length of the delay between messages as shown below. The clock is four times the tsk0_clock because it is used four times for each bit transmitted: three for the SCL clock timing and once for the SDA transition.

```
tsk0_counter: .lit24 68  # number of messages to transmit
tsk0_clock:   .lit24 62  # clock rate / 4
tsk0_delay:   .lit24 250 # length of delay between messages
```

The full assembly code in the Hightec™ format is provided in Appendix A. To modify the assembly code for the CASPR-MCS assembler, refer to AN4351.

5.1 Set SDA and SCL high

The default state of the I2C bus SDA and SCL transmission lines are "pulled up" and undriven. To emulate this pre-driven state, the ATOM channels must be set to output 1.

Table 3. Set SDA and SCL high

Step	Operation	Description	Parameters	Code snippet
1	Configure the ACB for SCL	Set on match event (Compare in CCU0 only, use timebase TBU_TS0).	ACB = 0x09	MOVL ACB PIN_HI
2	Place the data for the ATOM channel associated with SDA into the ACBO register	Move R3 to the ARU port.	ARU Read port = 0x0000	AWR R3 R3 ARU_PORT0
3	Read the ATOM channel to reactivate it after the match	In SOMC mode the channel is disabled after the match event.	ARU Read address = 0x0124	ARD R0 R4 ATOM0_CH5
4	Configure the ACB for SDA	See step 2.	ACB = 0x09	MOVL ACB PIN_HI
5	Place the data for the ATOM channel associated with SDA into the ACBO register	See step 3.	ARU Read port = 0x0001	AWR R3 R3 ARU_PORT1
6	Read the ATOM channel to reactive it after the match	See step 4.	ARU Read address = 0x0123	ARD R0 R4 ATOM0_CH4

5.2 Handshake with CPU

To ensure that both the CPU and the GTM are in the initialized state and ready to start the I2C communication, a handshake routine can be used. Both the CPU and the GTM MCS have access to the STRG and CTRG registers inside the MCS memory map.

The description of the assembly routine that runs inside of the GTM is described below in Table 4.

Table 4. Handshake with CPU

Step	Operation	Description	Code Snippet
1	Set the Channel 1 trigger	Set the trigger bit to indicate the routine has started to the CPU	MOVL STRG 0x000002
2	Load R0 with 1	—	MOVL R0 0x000001

Table continues on the next page...

Table 4. Handshake with CPU (continued)

Step	Operation	Description	Code Snippet
3	Wait until bit 0 of the STRG register is same as R0	Wait until the CPU signals back that the handshake was seen	WURM R0 STRG 0x0001
4	Load R3 with the current timebase value	—	MOV R3 TBU_TS0
5	Load R0 with 25,000	—	MOVL R0 0x0061A8
6	Add R0 and R3 in R0	—	ADD R0 R3
7	Wait until TS0 == R0	—	WURM R0 TBU_TS0 0xFFFF
8	Clear the triggers	—	MOVL CTRG 0x000003

The code below is the CPU's side of the handshake code.

```
/*Check that the ATOM channels are ready, MCS_STRG is set */
while((GTM_MCS_0.STRG.R & 0x2) == 0);

/*Next Trigger for MCS to signal "Port config finished"  */
GTM_MCS_0.STRG.R = 0x00000001;
```

5.3 Send Start bit

Before any message contents can be transmitted on the I2C bus, the start condition needs to be issued on the bus. A start bit is a high-to-low transmission on the SDA line while the SCL is high.

Table 5. Load message and send start bit

Step	Operation	Description	Parameters	Code snippet
1	Load TBU timestamp	Read the current value of the TBU timestamp in to the MCS register, R3.	n/a	MOV R3 TBU_TS0
2	Load R2 with the I2C clock speed	The I2C baud rate is controlled using the value stored at address 0x68.	n/a	MRD R2 tsk0_clock
3	Move index register to R5	Load the full message to be transmitted in to R5	n/a	MRDI R5 R6
4	Add R2 and R3 in R2	Set the match value at 0x68 from the current timestamp.	n/a	ADD R3 R2
5	Configure the ACB for SDA	Clear on match event (Compare in CCU0 only, use time base TBU_TS0).	ACB = 0x0A	MOVL ACB PIN_LO
6	Place the data for the ATOM channel associated with SDA into the ACBO registerCH4	Move R3 to the ARU port.	ARU Read port = 0x0001	AWR R3 R3 ARU_PORT1
7	Read the ATOM channel to reactivate it after the match	In SOMC mode the channel is disabled after the match event.	ARU Read address = 0x0123	ARD R0 R4 ATOM0_CH4
8	Add R2 and R3 in R2	See step 4.	n/a	ADD R3 R2

Table continues on the next page...

Table 5. Load message and send start bit (continued)

Step	Operation	Description	Parameters	Code snippet
9	Configure the ACB for SCL	See step 5.	ACB = 0x0A	MOVL ACB PIN_LO
10	Place the data for the ATOM channel associated with SDA into the ACBO register	See step 6.	ARU Read port = 0x0000	AWR R3 R3 ARU_PORT0
11	Read the ATOM channel to reactivate it after the match	See step 7.	ARU Read address = 0x0124	ARD R0 R4 ATOM0_CH5
12	Return from subprogram	The program counter PC is loaded with current value on the top of the stack.	n/a	RET

5.4 Send repeated Start bit

During an I2C transmission there is often the need to send a command and then immediately read back an answer. This has to be done without the risk of another device interrupting this atomic operation. The I2C protocol defines a repeated start condition. After having sent the address byte (address and read/write bit), the master may send any number of bytes followed by a stop condition. Instead of sending the stop condition it is also allowed to send another start condition again, followed by an address (and of course including a read/write bit) and more data.

Table 6. Send repeated start bit

Step	Operation	Description	Parameters	Code snippet
1	Move index register to R5	Load the full message to be transmitted in to R5	n/a	MRDI R5 R6
2	Add R2 and R3 in R2	Set the match value at 0x68 from the current timestamp.	n/a	ADD R3 R2
3	Configure the ACB for SDA	Set on match event (Compare in CCU0 only, use time base TBU_TS0).	ACB = 0x09	MOVL ACB PIN_HI
4	Place the data for the ATOM channel associated with SDA into the ACBO register	Move R3 to the ARU port.	ARU Read port = 0x0001	AWR R3 R3 ARU_PORT1
5	Read the ATOM channel to reactivate it after the match	In SOMC mode the channel is disabled after the match event.	ARU Read address = 0x0123	ARD R0 R4 ATOM0_CH4
6	Add R2 and R3 in R2	See step 2.	n/a	ADD R3 R2
7	Configure the ACB for SCL	See step 3.	ACB = 0x09	MOVL ACB PIN_HI
8	Place the data for the ATOM channel associated	See step 4.	ARU Read port = 0x0000	AWR R3 R3 ARU_PORT0

Table continues on the next page...

Table 6. Send repeated start bit (continued)

Step	Operation	Description	Parameters	Code snippet
	with SDA into the ACBO register			
9	Read the ATOM channel to reactive it after the match	See step 5.	ARU Read address = 0x0124	ARD R0 R4 ATOM0_CH5
10	Add R2 and R3 in R2	See step 2.	n/a	ADD R3 R2
11	Configure the ACB for SDA	See step 3.	ACB = 0x0A	MOVL ACB PIN_LO
12	Place the data for the ATOM channel associated with SDA into the ACBO register	See step 4.	ARU Read port = 0x0001	AWR R3 R3 ARU_PORT1
13	Read the ATOM channel to reactivate it after the match	See step 5.	ARU Read address = 0x0123	ARD R0 R4 ATOM0_CH4
14	Add R2 and R3 in R2	See step 2.	n/a	ADD R3 R2
15	Configure the ACB for SCL	See step 3.	ACB = 0x0A	MOVL ACB PIN_LO
16	Place the data for the ATOM channel associated with SDA into the ACBO register	See step 4.	ARU Read port = 0x0000	AWR R3 R3 ARU_PORT0
17	Read the ATOM channel to reactivate it after the match	See step 5.	ARU Read address = 0x0124	ARD R0 R4 ATOM0_CH5
18	Return from subroutine	The program counter PC is loaded with current value on the top of the stack.	n/a	RET

5.5 Send message

The message to be transmitted consists of 3 bytes: 7 bits of address, 1 bit R/W, and 2 bytes of data.

The messages in this example are stored in the MCS RAM in the format 0x00aadddd, where 'aa' is the address and the R/W bit, and 'dddd' is the 16-bit data. For example, 0x00e00114 is the address 0x70, with the R/W clear (indicating a write), the high byte data is 0x01, with the low byte data is 0x14 as shown in Figure 1.

As the message is transmitted MSB first, the message in memory must be shifted to get the bits out in the correct order.

The routine is designed to be run multiple times for each message, it preserves and moves the bits around so that each major and minor loop transmits the correct byte, and bit, in the correct order, using the MCS channel's R0 and R5 registers as illustrated in Figure 5.

Figure 4 shows the flow of the assembly routine that is executed to calculate the ATOM and timing of the ATOM channels for each byte. This routine is run once for each byte: ADDR and R/W, high DATA, and low DATA.

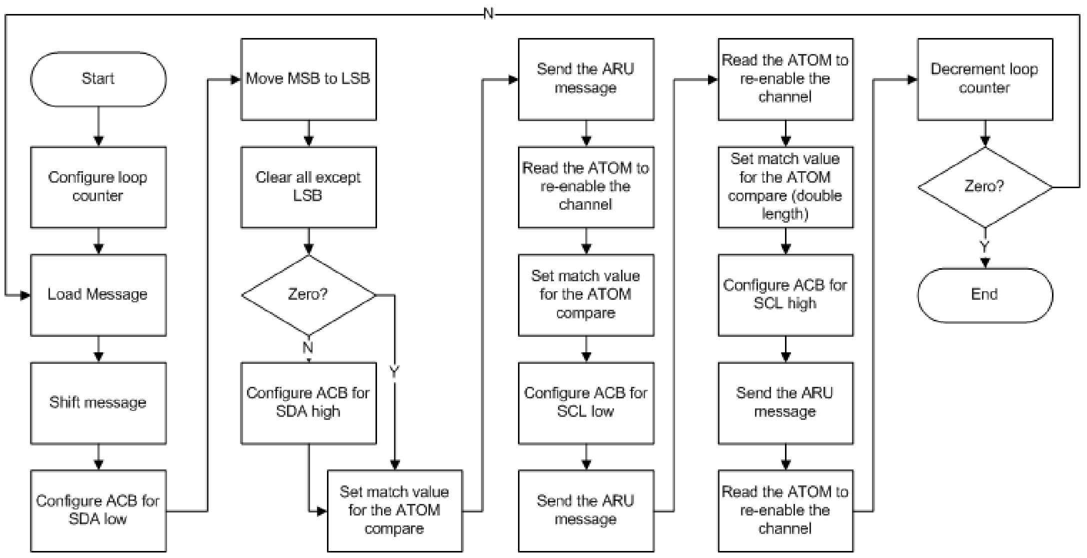

Start
Configure loop counter
Load Message
Shift message
Configure ACB for SDA low
Move MSB to LSB
Clear all except LSB
Zero?
N
Y
Configure ACB for SDA high
Set match value for the ATOM compare
Send the ARU message
Read the ATOM to re-enable the channel
Set match value for the ATOM compare
Configure ACB for SCL low
Send the ARU message
Read the ATOM to re-enable the channel
Set match value for the ATOM compare (double length)
Configure ACB for SCL high
Send the ARU message
Read the ATOM to re-enable the channel
Decrement loop counter
Zero?
N
Y
End

Figure 4. Flow Chart

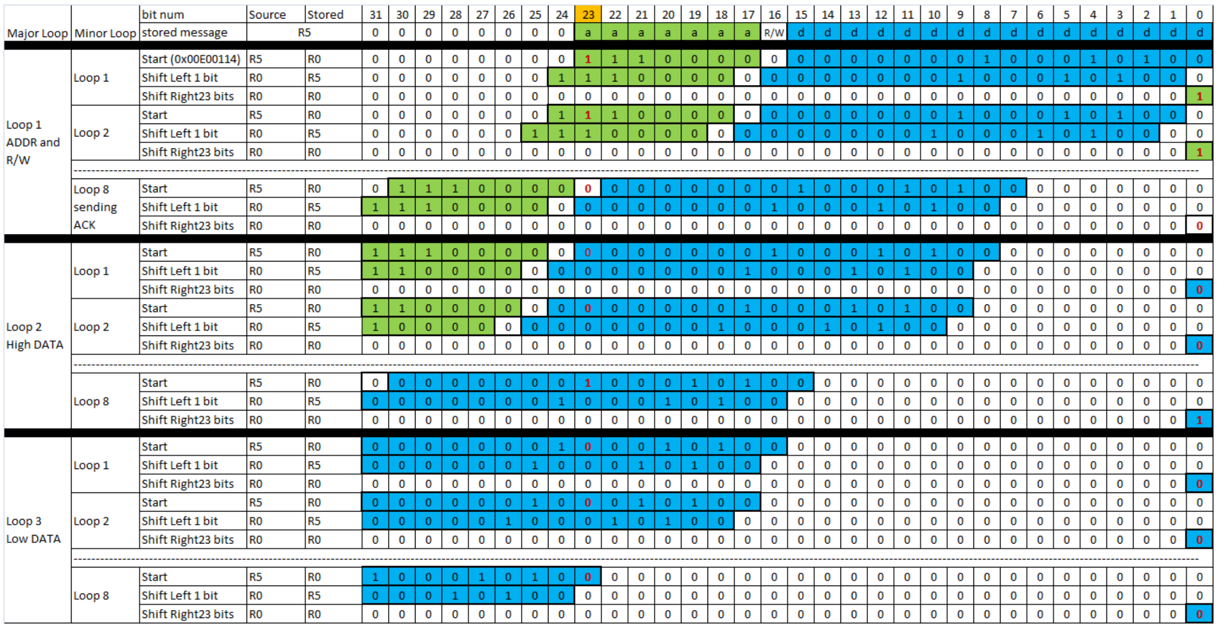
Table 7. Send byte (minor loop)

Step	Operation	Description	Parameters	Code snippet
1	Configure counter for loop	Set up a loop counter of 8 for the full 8-bit transmission, using R0. Copy the counter value to memory address 0x60.	n/a	MOVL R0 0x000008 MWR R0 0x0060
2 **(loop)**	Load message to transmit from R5	R5 was loaded with the message to be transmitted before the start bit was sent. Read it in to R0.	n/a	MOV R0 R5
3	Shift left the message in R5 once	Shift the message left so that the next shift right (step 6) accesses the correct bit each loop rotation.	n/a	SHL R5 0x0001
4	Configure the ACB for SDA	Clear on match event (Compare in CCU0 only, use time base TBU_TS0).	ACB = 0x0A	MOVL ACB PIN_LO
5	Move the byte MSB to the LSB position	Shift right the message in R0 23 spaces.	n/a	SHR R0 0x0017
6	Clear all bits except LSB bits in R0	Only interested in the LSB, and if it's zero.	n/a	ANDL R0 0x000001
7	If R0 is zero jump to step 10	Test R0 for zero.	n/a	JBS STA Z **tx_low**
8	Reconfigure the ACB for SDA	Set on match event (Compare in CCU0 only, use time base TBU_TS0) because a 1 is to be transmitted.	ACB = 0x09	MOVL ACB PIN_HI

Table continues on the next page...

Table 7. Send byte (minor loop) (continued)

Step	Operation	Description	Parameters	Code snippet
9 **(tx_low)**	Add R2 and R3 in R2	Set the match value at 0x68 from the current timestamp.	n/a	ADD R3 R2
10	Place the data for the ATOM channel associated with SDA into the ACBO register	Move R3 to the ARU port.	ARU Read port = 0x0001	AWR R3 R3 ARU_PORT1
11	Read the ATOM channel to reactive it after the match	In SOMC mode the channel is disabled after the match event.	ARU Read address = 0x0123	ARD R0 R4 ATOM0_CH4
12	Add R2 and R3 in R2	See step 10.	n/a	ADD R3 R2
13	Reconfigure the ACB for SCL	See step 9.	ACB = 0x09	MOVL ACB PIN_HI
14	Place the data for the ATOM channel associated with SDA into the ACBO register	See step 11.	ARU Read port = 0x0000	AWR R3 R3 ARU_PORT0
15	Read the ATOM channel to reactive it after the match	See step 12.	ARU Read address = 0x0124	ARD R0 R4 ATOM0_CH5
16	Add R2 and R3 in R2	See step 10.	n/a	ADD R3 R2
17	Add R2 and R3 in R2	See step 10.	n/a	ADD R3 R2
18	Reconfigure the ACB for SCL	See step 9.	ACB = 0x0A	MOVL ACB PIN_LO
19	Place the data for the ATOM channel associated with SDA into the ACBO register	See step 11.	ARU Read port = 0x0000	AWR R3 R3 ARU_PORT0
20	Read the ATOM channel to reactivate it after the match	See step 12.	ARU Read address = 0x0124	ARD R0 R4 ATOM0_CH5
21	Read and decrement the counter value. Then save it back.	Move the counter value from the RAM storage area to R0. Subtract 1 from the loop counter. Copy the counter value to memory address 0x60.	n/a	MRD R0 0x0060 SUBL R0 0x000001 MWR R0 0x0060
22	Test loop count	Jump back to step 3 is not complete.	n/a	JBC STA Z **loop**
23	Return from subprogram.	The program counter PC is loaded with current value on the top of the stack.	n/a	RET



		bit num	Source	Stored	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Major Loop	Minor Loop	stored message	R5		0	0	0	0	0	0	0	0	a	a	a	a	a	a	a	R/W	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d
Loop 1 ADDR and R/W	Loop 1	Start (0x00E00114)	R5	R0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0
		Shift Left 1 bit	R0	R5	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0
		Shift Right23 bits	R0	R0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	Loop 2	Start	R5	R0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0
		Shift Left 1 bit	R0	R5	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0
		Shift Right23 bits	R0	R0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	Loop 8 sending ACK	Start	R5	R0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
		Shift Left 1 bit	R0	R5	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
		Shift Right23 bits	R0	R0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Loop 2 High DATA	Loop 1	Start	R5	R0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
		Shift Left 1 bit	R0	R5	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
		Shift Right23 bits	R0	R0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Loop 2	Start	R5	R0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
		Shift Left 1 bit	R0	R5	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
		Shift Right23 bits	R0	R0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Loop 8	Start	R5	R0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		Shift Left 1 bit	R0	R5	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		Shift Right23 bits	R0	R0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Loop 3 Low DATA	Loop 1	Start	R5	R0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		Shift Left 1 bit	R0	R5	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		Shift Right23 bits	R0	R0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Loop 2	Start	R5	R0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		Shift Left 1 bit	R0	R5	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		Shift Right23 bits	R0	R0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Loop 8	Start	R5	R0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		Shift Left 1 bit	R0	R5	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		Shift Right23 bits	R0	R0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5. Message manipulation

5.6 Acknowledgement from slave

An I2C bus master releases the SDA line (high) and provides a pulse on the clock line straight after transmitting the last bit of the byte to allow a slave to acknowledge (pull low) the SDA line.

Table 8. Release Bus for ACK from slave

Step	Operation	Description	Parameters	Code snippet
1	Add R2 and R3 in R2	Set the match value at 0x68 from the current timestamp.	n/a	ADD R3 R2
2	Configure the ACB for SDA	Set on match event (Compare in CCU0 only, use time base TBU_TS0).	ACB = 0x09	MOVL ACB PIN_HI
3	Place the data for the ATOM channel associated with SDA into the ACBO register	Move R3 to the ARU port.	ARU Read port = 0x0001	AWR R3 R3 ARU_PORT1
4	Read the ATOM channel to reactivate it after the match	In SOMC mode the channel is disabled after the match event.	ARU Read address = 0x0123	ARD R0 R4 ATOM0_CH4
5	Add R2 and R3 in R2	See step 1.	n/a	ADD R3 R2
6	Configure the ACB for SCL	See step 2.	ACB = 0x09	MOVL ACB PIN_HI
7	Place the data for the ATOM channel associated with SDA into the ACBO register	See step 3.	ARU Read port = 0x0000	AWR R3 R3 ARU_PORT0

Table continues on the next page...

Table 8. Release Bus for ACK from slave (continued)

Step	Operation	Description	Parameters	Code snippet
8	Read the ATOM channel to reactivate it after the match	See step 4.	ARU Read address = 0x0124	ARD R0 R4 ATOM0_CH5
9	Add R2 and R3 in R2 (twice)	See step 1.	n/a	ADD R3 R2 ADD R3 R2
10	Configure the ACB for SCL	See step 2.	ACB = 0x0A	MOVL ACB PIN_LO
11	Place the data for the ATOM channel associated with SDA into the ACBO register	See step 3.	ARU Read port = 0x0000	AWR R3 R3 ARU_PORT0
12	Read the ATOM channel to reactivate it after the match	See step 4.	ARU Read address = 0x0124	ARD R0 R4 ATOM0_CH5
13	Return from subprogram.	The program counter PC is loaded with current value on the top of the stack.	n/a	RET

5.7 Acknowledging data from slave

An I2C slave releases the SDA line (high) while the master provides a pulse on the clock line after transmitting (slave)/ receiving (master) the last bit of the byte.

Step 2 is the only step that is different from the routine described in Acknowledgement from slave.

Table 9. Send ACK

Step	Operation	Description	Parameters	Code snippet
1	Add R2 and R3 in R2	Set the match value at 0x68 from the current timestamp.	n/a	ADD R3 R2
2	Configure the ACB for SDA	Clear on match event (Compare in CCU0 only, use time base TBU_TS0).	ACB = 0x0A	MOVL ACB PIN_LO
3	Place the data for the ATOM channel associated with SDA into the ACBO register	Move R3 to the ARU port.	ARU Read port = 0x0001	AWR R3 R3 ARU_PORT1
4	Read the ATOM channel to reactivate it after the match	In SOMC mode the channel is disabled after the match event.	ARU Read address = 0x0123	ARD R0 R4 ATOM0_CH4
5	Add R2 and R3 in R2	See step 1.	n/a	ADD R3 R2
6	Configure the ACB for SCL	See step 2.	ACB = 0x09	MOVL ACB PIN_HI

Table continues on the next page...

Table 9. Send ACK (continued)

Step	Operation	Description	Parameters	Code snippet
7	Place the data for the ATOM channel associated with SDA into the ACBO register	See step 3.	ARU Read port = 0x0000	AWR R3 R3 ARU_PORT0
8	Read the ATOM channel to reactivate it after the match	See step 4.	ARU Read address = 0x0124	ARD R0 R4 ATOM0_CH5
9	Add R2 and R3 in R2 (twice)	See step 1.	n/a	ADD R3 R2 ADD R3 R2
10	Configure the ACB for SCL	See step 2.	ACB = 0x0A	MOVL ACB PIN_LO
11	Place the data for the ATOM channel associated with SDA into the ACBO register	See step 3.	ARU Read port = 0x0000	AWR R3 R3 ARU_PORT0
12	Read the ATOM channel to reactivate it after the match	See step 4.	ARU Read address = 0x0124	ARD R0 R4 ATOM0_CH5
13	Return from subprogram.	The program counter PC is loaded with current value on the top of the stack.	n/a	RET

5.8 Send Stop bit

To completed transmission of the message, the stop condition needs to be issued on the bus. A stop bit is a low-to-high transmission on the SDA line while the SCL is high.

Table 10. Send stop bit

Step	Operation	Description	Parameters	Code snippet
1	Add R2 and R3 in R2	Set the match value at 0x68 from the current timestamp.	n/a	ADD R3 R2
2	Configure the ACB for SDA	Clear on match event (Compare in CCU0 only, use time base TBU_TS0).	ACB = 0x0A	MOVL ACB PIN_LO
3	Place the data for the ATOM channel associated with SDA into the ACBO register	Move R3 to the ARU port.	ARU Read port = 0x0001	AWR R3 R3 ARU_PORT1
4	Read the ATOM channel to reactivate it after the match	In SOMC mode the channel is disabled after the match event.	ARU Read address = 0x0123	ARD R0 R4 ATOM0_CH4
5	Add R2 and R3 in R2	See step 1.	n/a	ADD R3 R2

Table continues on the next page...

Table 10. Send stop bit (continued)

Step	Operation	Description	Parameters	Code snippet
6	Configure the ACB for SCL	See step 2.	ACB = 0x09	MOVL ACB PIN_HI
7	Place the data for the ATOM channel associated with SDA into the ACBO register	See step 3.	ARU Read port = 0x0000	AWR R3 R3 ARU_PORT0
8	Read the ATOM channel to reactivate it after the match	See step 4.	ARU Read address = 0x0124	ARD R0 R4 ATOM0_CH5
9	Add R2 and R3 in R2	See step 1.	n/a	ADD R3 R2
10	Configure the ACB for SDA	See step 2.	ACB = 0x09	MOVL ACB PIN_HI
11	Place the data for the ATOM channel associated with SDA into the ACBO register	See step 3.	ARU Read port = 0x0001	AWR R3 R3 ARU_PORT1
12	Read the ATOM channel to reactivate it after the match	See step 4.	ARU Read address = 0x0123	ARD R0 R4 ATOM0_CH4
13	Return from subprogram.	The program counter PC is loaded with current value on the top of the stack.	n/a	RET

5.9 MCS delay routine

A delay routine is a useful code snippet to have for any software development. In this example, a delay is used to create a time space between messages.

The MCS has direct access to the TBU timestamp counter and also has a "wait until register match" instruction, WURM, which can be used to hold the MCS program counter for a predetermined amount of time or to wait until a trigger event from another channel occurs. WURM suspends the MCS channel until the two registers (with a bit mask) match.

```
WURM A B C

Wait until A = (B & C)
```

A commonly used delay routine often involves a variable "duration" that is decremented in a loop, until it is zero. Within that loop, a known finite time can be included by using a wait operation.

In the example given in this application note, the "duration" variable is stored in the MCS RAM with other data such as the clock frequency and message loop counter at address 0x6C (0xFA = 250 in this particular case).

The timebase value is read and the match value is set at 2,500 clocks after "now." If the GTM TBU is running from an 80 MHz clock, the delay is 31.25 µs around each loop, 7.8125 ms in total.

```
delay:
    MOV     R3    TBU_TS0           # Load timestamp to R3
    MRD     R4    0x006C            # Load loop counter to R4
    ATUL    R4    0x000000          # Is R4 Zero?
    JBS     STA   Z  exit           # If R4 is Zero jump to exit
    MOVL    R0    0x0009C4          # Load R0 with 2,500
```

```
  continue: ADD    R3   R0          # Add R0 to the Timebase saved in R3
    WURM   R3   TBU_TS0  0xFFFF     # Wait until the timebase matches R3
    SUBL   R4   0x000001            # Decrement the loop counter in R4
    JBC    STA  Z  continue         # If not zero continue
  exit: RET                         # Exit subroutine
```

Appendix A Core Code to Initialize the GTM for I2C

The C code below configures the TBU, the ATOM channels, and then loads the MCS software and message array into MCS RAM memory and the handshake with the MCS itself.

The MCS software is compiled in to a binary file and loaded into the MCS RAM as described in AN4351. The MCS message array is given as an array of integers and copied into RAM at the specified location so that it is easier to manipulate without the need to reassemble the MCS software each time.

```
unsigned int * dest, src;
extern int __MCS0_ADDR; /* Label of location of the raw data set in the linker */

void memcpy_swap_word(unsigned int *, unsigned int *, signed int);
void I2C()
{
int i;
gtm_ptr p;

// Configure TBU
GTM_TBU.CH0_CTRL.R = 0x00000000;// Select CMU_CLK0
GTM_TBU.CHEN.R = 0x00000002;// Switch on TBU0

/*********************************************************************************
*ATOM0 CH5 = SCL
*ATOM0 CH4 = SDA
*********************************************************************************/

/* Program MCS. First check whether the RAM RESET is complete.
WAIT until RAM_RST == 0, wait RAM Reset after startup. */
while(GTM_MCS_0.CTRL.R == 0x00010000);

GTM_ATOM_0.CH5_RDADDR.R = MCS0_WRADDR0;// used for reading
GTM_ATOM_0.CH4_RDADDR.R = MCS0_WRADDR1;  // used for reading
GTM_ATOM_0.CH5_CTRL.R = 0x00000009;// SOMC, ARU_EN=1, SL=0
GTM_ATOM_0.CH4_CTRL.R = 0x00000009;// SOMC, ARU_EN=1, SL=0

/*ATOM0_CH4 + CH5 switch on*/
GTM_ATOM_0.AGC_OUTEN_CTRL.R = 0x00000A00;
GTM_ATOM_0.AGC_ENDIS_CTRL.R = 0x00000A00;

GTM_ATOM_0.AGC_FUPD_CTRL.R = 0x00000000;
GTM_ATOM_0.AGC_INT_TRIG.R = 0x00000000;

GTM_ATOM_0.AGC_GLB_CTRL.R = 0xAAAA0001;// Host Trigger to start ATOM

// load raw bin data in to MCS0 RAM = 0xFFD38000
dest = (int)&MCS0_MEM; /* CPU view of the address of the MCS memory space */
src = (int)&__MCS0_ADDR; /* Label of location of the raw data set in the linker */
memcpy_swap_word(dest, src, 270);

p = &MCS0_MEM + 0x1C;
for(i=0;i<=67;i++)
{
/* Copying the content of the array mcs0i2c_messages[i] into MCS0 RAM0 */
p[i]=mcs0i2c_messages[i];
}

/* Start the MCS Program */
GTM_MCS_0.CH0_CTRL.r = 0x00000001;// Enable Channel 0 of MCS module 0

/*Check if the Channel program is ready and MCS_STRG is set, then start configure the Ports.
WAIT until STRG == 0x00000002
MCS --> ATOM Output finished when STRG == h#00000002 */
while((GTM_MCS_0.STRG.R & 0x2) == 0); // STRG != 2

/*Next Trigger for MCS to signalize "Port config finished"  */
GTM_MCS_0.STRG.R = 0x00000001;// Port configuration finished, MCS running
/*Now the MCS is running in a infinite loop. */

}/*END of function I2C()*/

void memcpy_swap_word(unsigned int * dst, unsigned int * src, signed int size)
{
while (size-- > 0)
{
*dst++ = SWAPW(*src);
src++;
}
}
```

Below is the I2C message array used in this example.

```
int mcs0i2c_messages[68] = {0x00e00101, 0x00e00102, 0x00e00104, 0x00e00108, 0x00e00110,
0x00e00120, 0x00e00140, 0x00e00180, 0x00e00101, 0x00e00102, 0x00e00104, 0x00e00108,
0x00e00110, 0x00e00120, 0x00e00140, 0x00e00180, 0x00e00101, 0x00e00102, 0x00e00104,
0x00e00108, 0x00e00110, 0x00e00120, 0x00e00140, 0x00e00180, 0x00e00155, 0x00e001aa,
0x00e00155, 0x00e001aa, 0x00e00155, 0x00e001aa, 0x00e00100, 0x00e001ff, 0x00e00100,
0x00e001ff, 0x00e00100, 0x00e001ff, 0x00e00180, 0x00e00141, 0x00e00122, 0x00e00114,
0x00e00108, 0x00e00114, 0x00e00122, 0x00e00141, 0x00e00180, 0x00e00141,
0x00e00122, 0x00e00114, 0x00e00108, 0x00e00114, 0x00e00122, 0x00e00141, 0x00e00180,
0x00e00141, 0x00e00122, 0x00e00114, 0x00e00108, 0x00e00114, 0x00e00122, 0x00e00141,
0x00e00180, 0x00e00141, 0x00e00122, 0x00e00114, 0x00e00108, 0x00e00114, 0x00e00122,
0x00e00141};
```

The Hightec assembler generates the binary in the little endian, whereas the MPC57xx is big endian. The endianness can be swapped using the following macro.

```
#define SWAPW(w) \
(((w & 0xff) << 24) | ((w & 0xff00) << 8) \
| ((w & 0xff0000) >> 8) | ((w & 0xff000000) >> 24)) /* change endianness */
```

Appendix B Assembly Code for I2C Example

```
#============================================================================
# Project Name    : AN4789
# Company         : Freescale
# Author          : Inga Harris
#============================================================================
.section .mcs.text,"axw",@progbits
.include "mcs.inc"
.set memid, 0
.set memsize, 0x1800

# Define the values of the symbols used
.set ARU_PORT0, 0x0000     # MCS ARU port number 0
.set ARU_PORT1, 0x0001     # MCS ARU port number 0
.set PIN_HI, 0x000009      # ACB = 0x09 set high when compare in CCU0 with TBU_TS0
.set PIN_LO, 0x00000A      # ACB = 0x0A clear high when compare in CCU0 with TBU_TS0
.set ATOM0_CH4, 0x0123     # ATOM0_CH4 ARU write address
.set ATOM0_CH5, 0x0124     # ATOM0_CH5 ARU write address
.set message_array, 0x70   # offset address of the I2C messages


# initialize reset vectors of different tasks
# -------------------------------------------
.org 0x0
jmp tsk0_init
```

```
# allocate stack frames ( each task has 16 memory locations )
# -----------------------------------------------------------
.org 0x20
tsk0_stack:.lit24 0


# allocate and initialize memory variables
# ----------------------------------------
.org 0x64
tsk0_counter: .lit24 68    # number of messages to transmit
tsk0_clock:   .lit24 62    # clock rate / 4
tsk0_delay:   .lit24 250   # length of delay between messages


#*************************************************************
#   tsk0: I2C master
#*************************************************************
.org 0x180
tsk0_init:
movl   R7,   0x000020               # Init stack pointer
mov    R3,   TBU_TS0                # Load timestamp
movl   ACB,  PIN_HI                 # Set ACB value
awr    R3,   R3,   ARU_PORT0        # Send timestamp and ACB to ARU
ard    R0,   R4,   ATOM0_CH5        # Read ATOM channel
movl   ACB,  PIN_HI                 # Set ACB value
awr    R3,   R3,   ARU_PORT1        # Send timestamp and ACB to ARU
ard    R0,   R4,   ATOM0_CH4        # Read ATOM channel
movl   STRG, 0x000002               # Set channel 1 trigger
movl   R0,   0x000001               # Load R0 with 1
wurm   R0,   STRG, 0x0001           # Wait until channel 0 trigger is set by core
mov    R3,   TBU_TS0                # reload the current timestamp
movl   R0,   0x0061A8               # Set R0 to 25,000
add    R0,   R3                     # Add R0 and R3
wurm   R0,   TBU_TS0,  0xFFFF       # Wait until the timstamp reaches that value
movl   CTRG, 0x000003               # Clear the triggers
start_tx: movl   R6,   message_array # Initialize index register
mrd    R1,   tsk0_counter           # Initialise loop counter
next_message: call   start_bit      # Send Start Bit
call   byte_tx                      # Send ADDR and R/W
call   wait_for_ack                 # Release SDA for ACK
call   byte_tx                      # Send DATA high byte
call   wait_for_ack                 # Release SDA for ACK
call   byte_tx                      # Send DATA low byte
call   wait_for_ack                 # Release SDA for ACK
call   stop_bit                     # Send Stop Bit
call   delay    # Wait tsk0_delay * 2,500 clocks
addl   R6,   0x000004               # Increment index register
subl   R1,   0x000001               # Deccrement loop counter
jbc    STA,  Z,  next_message       # Is the loop counter zero? No = next_message
jmp    start_tx                     # Loop ended. Start from beginning

#*************************************************************
#   start_bit
#*************************************************************
start_bit:
mov    R3,   TBU_TS0                # Load timestamp
mrd    R2,   tsk0_clock             # Load clock rate
mrdi   R5,   R6                     # Read index register
add    R3,   R2                     # Add clock rate to timestamp
movl   ACB,  PIN_LO                 # Set ACB value
awr    R3,   R3,   ARU_PORT1        # Send timestamp and ACB to ARU
ard    R0,   R4,   ATOM0_CH4        # Read ATOM channel
add    R3,   R2                     # Add clock rate to timestamp
movl   ACB,  PIN_LO                 # Set ACB value
awr    R3,   R3,   ARU_PORT0        # Send timestamp and ACB to ARU
ard    R0,   R4,   ATOM0_CH5        # Read ATOM channel
ret                                 # Return from subroutine

#*************************************************************
```

```
#   repeated_start
#************************************************************
repeated_start:
mrdi    R5,    R6
add     R3,    R2                   # Add clock rate to timestamp
movl    ACB,   PIN_HI               # Set ACB value
awr     R3,    R3,    ARU_PORT1     # Send timestamp and ACB to ARU
ard     R0,    R4,    ATOM0_CH4     # Read ATOM channel
add     R3,    R2                   # Add clock rate to timestamp
movl    ACB,   PIN_HI               # Set ACB value
awr     R3,    R3,    ARU_PORT0     # Send timestamp and ACB to ARU
ard     R0,    R4,    ATOM0_CH5     # Read ATOM channel
add     R3,    R2                   # Add clock rate to timestamp
movl    ACB,   PIN_LO               # Set ACB value
awr     R3,    R3,    ARU_PORT1     # Send timestamp and ACB to ARU
ard     R0,    R4,    ATOM0_CH4     # Read ATOM channel
add     R3,    R2                   # Add clock rate to timestamp
movl    ACB,   PIN_LO               # Set ACB value
awr     R3,    R3,    ARU_PORT0     # Send timestamp and ACB to ARU
ard     R0,    R4,    ATOM0_CH5     # Read ATOM channel
ret                                 # Return from subroutine

#************************************************************
#   byte_tx
#************************************************************
byte_tx:
movl    R0,    0x000008             # Set loop counter for 8 bits
mwr     R0,    0x0060               # Copy back to address 0x60
loop: mov      R0,    R5            # Load message frpm R5 to R0
shl     R5,    0x0001               # Shift message left once in R5
movl    ACB,   PIN_LO               # Set ACB value
shr     R0,    0x0017               # Shift R0 right 23 times
andl    R0,    0x000001             # Clear all bit LSB
jbs     STA,   Z,    tx_low         # If result is zero jump next instruction
movl    ACB,   PIN_HI               # Set ACB value
tx_low: add      R3,    R2          # Add clock rate to timestamp
awr     R3,    R3,    ARU_PORT1     # Send timestamp and ACB to ARU
ard     R0,    R4,    ATOM0_CH4     # Read ATOM channel
add     R3,    R2                   # Add clock rate to timestamp
movl    ACB,   PIN_HI               # Set ACB value
awr     R3,    R3,    ARU_PORT0     # Send timestamp and ACB to ARU
ard     R0,    R4,    ATOM0_CH5     # Read ATOM channel
add     R3,    R2                   # Add clock rate to timestamp
add     R3,    R2                   # Add clock rate to timestamp
movl    ACB,   PIN_LO               # Set ACB value
awr     R3,    R3,    ARU_PORT0     # Send timestamp and ACB to ARU
ard     R0,    R4,    ATOM0_CH5     # Read ATOM channel
mrd     R0,    0x0060               # Move bit counter to R0
subl    R0,    0x000001             # Decrement bit counter
mwr     R0,    0x0060               # Save back to R0
jbc     STA,   Z,    loop           # If bit counter is zero jump back to loop
ret                                 # Return from subroutine

#************************************************************
#   send_ack
#************************************************************
send_ack:
add     R3,    R2                   # Add clock rate to timestamp
movl    ACB,   PIN_LO               # Set ACB value
awr     R3,    R3,    ARU_PORT1     # Send timestamp and ACB to ARU
ard     R0,    R4,    ATOM0_CH4     # Read ATOM channel
add     R3,    R2                   # Add clock rate to timestamp
movl    ACB,   PIN_HI               # Set ACB value
awr     R3,    R3,    ARU_PORT0     # Send timestamp and ACB to ARU
ard     R0,    R4,    ATOM0_CH5     # Read ATOM channel
add     R3,    R2                   # Add clock rate to timestamp
add     R3,    R2                   # Add clock rate to timestamp
movl    ACB,   PIN_LO               # Set ACB value
awr     R3,    R3,    ARU_PORT0     # Send timestamp and ACB to ARU
ard     R0,    R4,    ATOM0_CH5     # Read ATOM channel
```

```
ret                                 # Return from subroutine

#*************************************************************
#   wait_for_ack
#*************************************************************
wait_for_ack:
add     R3,    R2                   # Add clock rate to timestamp
movl    ACB,   PIN_HI               # Set ACB value
awr     R3,    R3,    ARU_PORT1     # Send timestamp and ACB to ARU
ard     R0,    R4,    ATOM0_CH4     # Read ATOM channel
add     R3,    R2                   # Add clock rate to timestamp
movl    ACB,   PIN_HI               # Set ACB value
awr     R3,    R3,    ARU_PORT0     # Send timestamp and ACB to ARU
ard     R0,    R4,    ATOM0_CH5     # Read ATOM channel
add     R3,    R2                   # Add clock rate to timestamp
add     R3,    R2                   # Add clock rate to timestamp
movl    ACB,   PIN_LO               # Set ACB value
awr     R3,    R3,    ARU_PORT0     # Send timestamp and ACB to ARU
ard     R0,    R4,    ATOM0_CH5     # Read ATOM channel
ret                                 # Return from subroutine

#*************************************************************
#   stop_bit
#*************************************************************
stop_bit:
add     R3,    R2                   # Add clock rate to timestamp
movl    ACB,   PIN_LO               # Set ACB value
awr     R3,    R3,    ARU_PORT1     # Send timestamp and ACB to ARU
ard     R0,    R4,    ATOM0_CH4     # Read ATOM channel
add     R3,    R2                   # Add clock rate to timestamp
movl    ACB,   PIN_HI               # Set ACB value
awr     R3,    R3,    ARU_PORT0     # Send timestamp and ACB to ARU
ard     R0,    R4,    ATOM0_CH5     # Read ATOM channel
add     R3,    R2                   # Add clock rate to timestamp
movl    ACB,   PIN_HI               # Set ACB value
awr     R3,    R3,    ARU_PORT1     # Send timestamp and ACB to ARU
ard     R0,    R4,    ATOM0_CH4     # Read ATOM channel
ret                                 # Return from subroutine

#*************************************************************
#   delay
#*************************************************************
delay:
mov     R3,    TBU_TS0              # Load timestamp
mrd     R4,    tsk0_delay           # Load loop counter
atul    R4,    0x000000             # Is it zero?
jbs     STA,   Z,   exit            # If zero exit subroutine
movl    R0,    0x0009c4             # Load R0 with 2,500
continue: add     R3,    R0         # Add 2,500 to timestamp
wurm    R3,    TBU_TS0,   0xFFFF    # Wait until timebase matches R3
subl    R4,    0x000001             # Decrement loop counter
jbc     STA,   Z,   continue        # If not zero jump to continue
exit: ret                           # return from subroutine
```

Document Number AN4789
Revision 1, 3/2014