TEXAS INSTRUMENTS

December 2014

Examples Guide: MSP430 SMBus Library

MSP430

1 Introduction

This document is intended for the person evaluating the MSP430 SMBus Library. The API is accompanied by many examples that demonstrate its use; this document describes how to run those examples, and provides commentary on how they were written.

For detailed information on developing with the SMBus Library, please see the MSP430 SMBus Library User's Guide.

Table of Contents

1.1 Supported Devices

The SMBus library and examples were developed and tested in the following MSP430 families of devices:

Table 1: MSP430 Devices supported by SMBus Library examples

Device	SMBus/I^{2}C Interface
MSP430FR59xx	Enhanced Universal Serial Communication Interface (eUSCI)
MSP430G2xx3	Universal Serial Communication Interface (USCI)

The code examples and the SMBus library can be implemented in different derivatives of the same family with little effort; and they can be migrated to other MSP430 devices with the same SMBus/I^{2}C interface with some considerations such as pin usage, memory and timeout-timer availability.

1.2 Summary of the Examples

Each example in this software package is provided with full source code and including the necessary project files to build the project in IAR or CCS.

The examples, with the corresponding role and slave address are listed in Table 2.

Table 2: Examples implemented in SMBus Library

Name	Role	Slave Address
SMB_Master00_AllProtocols	Master	0x43
SMB_Slave00_AllProtocols	Slave	
SMB_Master01_ReadByte_Echo	Master	0x43
SMB_Slave01_ReadByte_Echo	Slave	
SMB_Master02_WriteWord_Dimmer	Master	0x43
SMB_Slave02_WriteWord_Dimmer	Slave	

The hardware used by each example is described in more detail in Section 2.2 and their functionality is explained in Section 3.

2 Running the Examples

2.1 Obtaining Code Composer Studio or IAR Kickstart

The SMBus Library and examples build and run on both the IAR and Code Composer Studio (CCS) environments for MSP430. See the Release Notes HTML file in the SMBus Library package for specific IAR/CCS version information.

IAR and CCS are both available in free, code-size-limited versions (8K and 16K, respectively, of object code). Applications that require less than r 8K of memory can be run on both free versions. Applications that require more than 8K of memory cannot be built using the free IAR Kickstart tool. Instead, the free version of CCS can be used, or a licensed version of either environment.

2.2 Hardware Support

Most of the examples can run on any hardware TI sells in the eStore, supporting the devices mentioned in Table 1. This includes:

- FR5969 LaunchPad (MSP-EXP430FR5969)
- MSP430 Value Line Launchpad (MSP-EXP430G2)
- Any FET target board for supported MSP430 derivatives

Every example has a HAL layer which can be used to initialize ports, clocks and any other hardware used by the application and SMBus layers.

The following tables list the resources used by the examples:

Table 3: Hardware resources for example 00_AllProtocols

	Description	MSP430FR59xx	MSP430G2xx3
SMB_Master **00_AllProtocols**	I2C	eUSCI_B0	USCIB0
	I2C_SDA	P1.6/UCB0SDA	P1.7/UCB0SDA
	I2C_SCL	P1.7/UCB0SCL	P1.6/UCB0SCL
	LED0	P1.0	P1.0
	LED1	P4.6	N/A
SMB_Slave **00_AllProtocols**	I2C	eUSCI_B0	USCIB0
	I2C_SDA	P1.6/UCB0SDA	P1.7/UCB0SDA
	I2C_SCL	P1.7/UCB0SCL	P1.6/UCB0SCL
	LED0	P1.0	P1.0
	LED1	P4.6	N/A
	ADC	P2.4/A7	P1.4/A4

	ADC trigger	TA0.1	TA1.0
	IOPort	P4	P1
	Input Switch	P4.5	P1.3

Table 4: Hardware resources for example 01_ ReadByte_Echo

SMB_Master	**Description**	**MSP430FR59xx**	**MSP430G2xx3**
01_ReadByte_Echo	I2C	eUSCI_B0	USCIB0
	I2C_SDA	P1.6/UCB0SDA	P1.7/UCB0SDA
	I2C_SCL	P1.7/UCB0SCL	P1.6/UCB0SCL
	LED0	P1.0	P1.0
	LED1	P4.6	N/A
SMB_Slave	I2C	eUSCI_B0	USCIB0
01_ReadByte_Echo	I2C_SDA	P1.6/UCB0SDA	P1.7/UCB0SDA
	I2C_SCL	P1.7/UCB0SCL	P1.6/UCB0SCL
	LED0	P1.0	P1.0
	LED1	P4.6	N/A

Table 5: Hardware resources for example 02_WriteWord_Dimmer

SMB_Master	**Description**	**MSP430FR59xx**	**MSP430G2xx3**
02_WriteWord_Dimmer	I2C	eUSCI_B0	USCIB0
	I2C_SDA	P1.6/UCB0SDA	P1.7/UCB0SDA
	I2C_SCL	P1.7/UCB0SCL	P1.6/UCB0SCL
	LED0	P1.0	P1.0
	LED1	P4.6	N/A
SMB_Slave	I2C	eUSCI_B0	USCIB0
02_WriteWord_Dimmer	I2C_SDA	P1.6/UCB0SDA	P1.7/UCB0SDA

	I2C_SCL	P1.7/UCB0SCL	P1.6/UCB0SCL
	LED0	P1.0	P1.0
	LED1	P4.6	N/A
	PWM	P1.4/TB0.1	P2.4/TA1.2

2.3 Opening and building the examples

The SMBus Library examples have the following structure:

```
<install>
  |--driverlib         <- MSP430 driverlib used to access MSP430 peripherals
  |--smbuslib          <- Contains the SMBus library itself
  |--examples
       |--MSP430*       <- MSP430 device supported by examples
       |     |--- CCS
       |     |     |-- SMB*.projectspec  <- Individual project files for CCS
       |     |
       |     |--- IAR
       |     |     |-- SMB*
       |     |     |    |-- *.ewd/ewp  <- Individual project files for IAR
       |     |     |    |-- *.eww      <- Workspace for individual projects
       |     |     |-- SMBus_Examples_Workspace.eww <- Workspace containing all
       |     |                                         examples
       |     |--- Src
       |     |     |-- SMB*
       |     |          |-- main.c      <- The example's main code
       |     |          |-- *_HAL.c/h   <- Hardware abstraction for example
       |     |          |-- *.c/h       <- Other source files used by application
```

2.3.1 IAR Projects

The IAR projects are grouped within workspaces. There is one workspace containing all examples and individual workspaces for each project.

To use the examples in IAR, open the corresponding workspace file (*.eww) by selecting File→Open→Workspace.

After doing so, one of the projects in the workspace will be highlighted in bold; this is the active project. If a different example is desired, right-click on it and select “set as active”. If the workspace only has one project, this will be the only option.

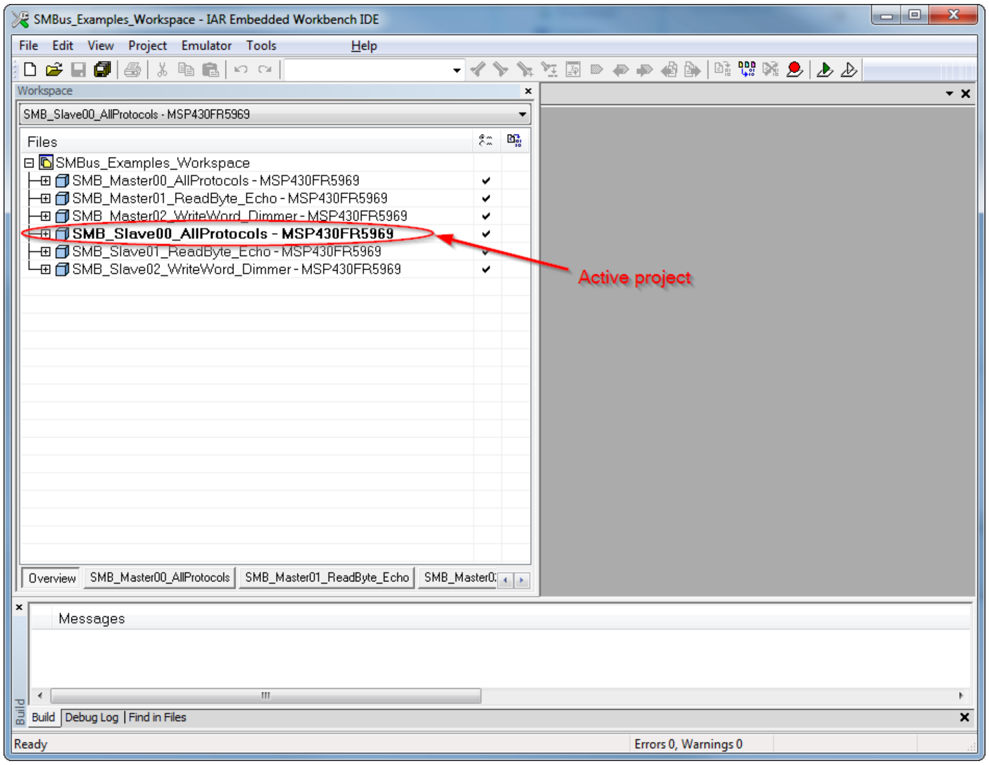

SMBus_Examples_Workspace - IAR Embedded Workbench IDE
File Edit View Project Emulator Tools Help
Workspace
SMB_Slave00_AllProtocols - MSP430FR5969
Files
SMBus_Examples_Workspace
SMB_Master00_AllProtocols - MSP430FR5969
SMB_Master01_ReadByte_Echo - MSP430FR5969
SMB_Master02_WriteWord_Dimmer - MSP430FR5969
SMB_Slave00_AllProtocols - MSP430FR5969
SMB_Slave01_ReadByte_Echo - MSP430FR5969
SMB_Slave02_WriteWord_Dimmer - MSP430FR5969
Active project
Overview
SMB_Master00_AllProtocols
SMB_Master01_ReadByte_Echo
SMB_Master0
Messages
Build
Debug Log
Find in Files
Ready
Errors 0, Warnings 0

Figure 1: Active project in IAR

If desired, the active project can also be selected by clicking on the corresponding project at the bottom of the workspace window. This will only show the files used by the Active project in the workspace window as shown in the following figure:

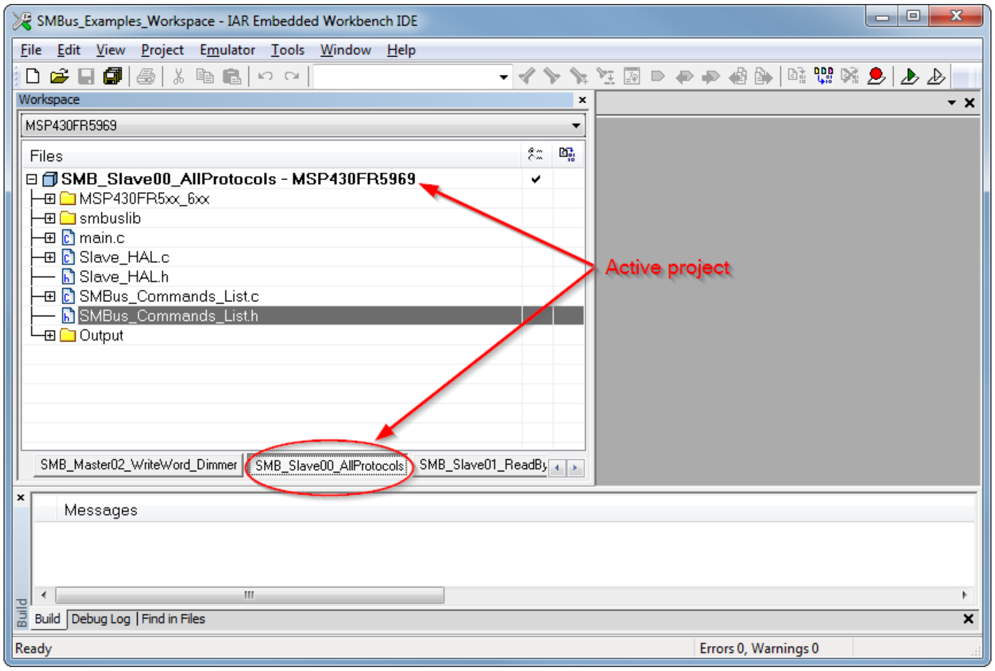


Figure 2: Individual project view in IAR

The project can be built using F7, Menu→Project→rebuild All, or ; and downloaded using Ctrl+D, Menu→Project→Download and Debug, or .

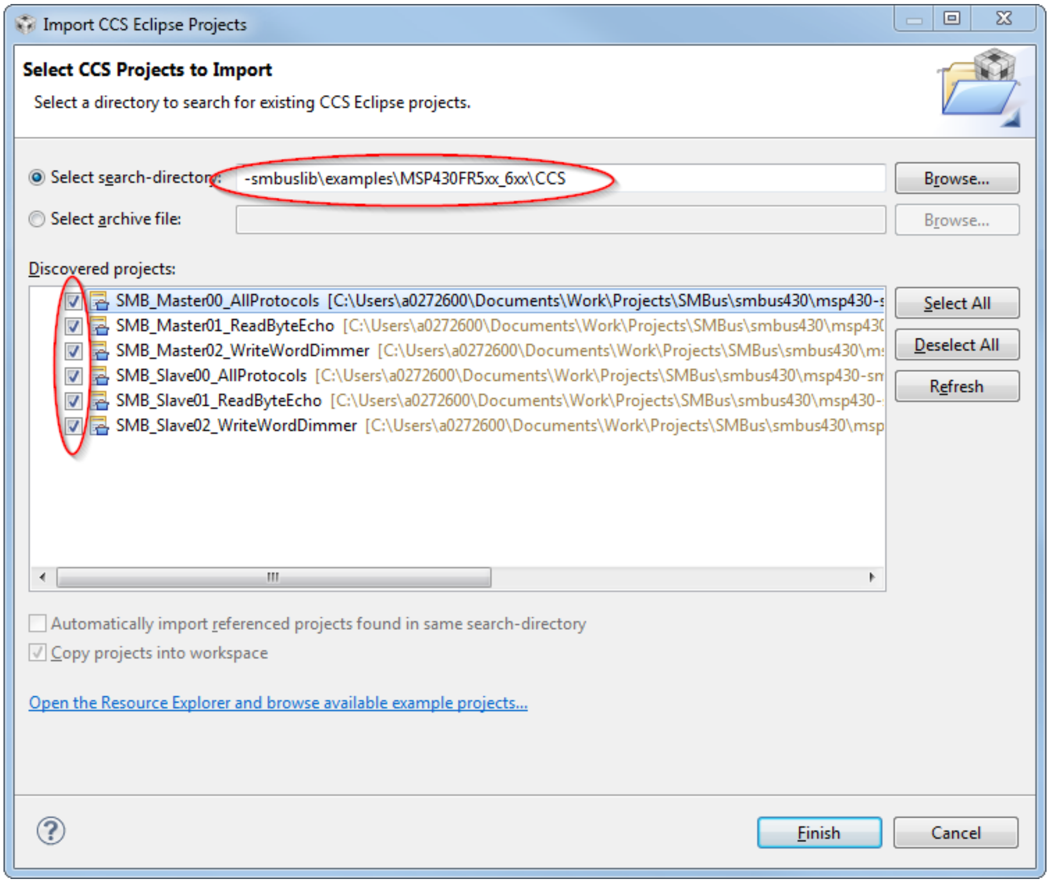
The example can now be executed.

2.3.2 CCS projects

The CCS projects are not grouped in workspaces. They need to be imported into a workspace of your choosing. The projects are defined by *.projectspec files, which contain the information CCS needs to import the project.

Open CCS, and choose Project → Import CCS Eclipse Project. Browse to the \CCS directory (containing the .projectspec files) of the example you wish to open. CCS will show a list of all the projects that were discovered in this directory.

Select all or any of the projects and click on "Finish". The project(s) should appear in the *Project Explorer.*



Import CCS Eclipse Projects

Select CCS Projects to Import

Select a directory to search for existing CCS Eclipse projects.

● Select search-directory: -smbuslib\examples\MSP430FR5xx_6xx\CCS

○ Select archive file:

Discovered projects:

☑ SMB_Master00_AllProtocols [C:\Users\a0272600\Documents\Work\Projects\SMBus\smbus430\msp430-s

☑ SMB_Master01_ReadByteEcho [C:\Users\a0272600\Documents\Work\Projects\SMBus\smbus430\msp430

☑ SMB_Master02_WriteWordDimmer [C:\Users\a0272600\Documents\Work\Projects\SMBus\smbus430\ms

☑ SMB_Slave00_AllProtocols [C:\Users\a0272600\Documents\Work\Projects\SMBus\smbus430\msp430-sn

☑ SMB_Slave01_ReadByteEcho [C:\Users\a0272600\Documents\Work\Projects\SMBus\smbus430\msp430-

☑ SMB_Slave02_WriteWordDimmer [C:\Users\a0272600\Documents\Work\Projects\SMBus\smbus430\msp

□ Automatically import referenced projects found in same search-directory

☑ Copy projects into workspace

Open the Resource Explorer and browse available example projects...

Figure 3: Importing CCS projects

The active project can be selected by simply left-clicking on the project and it will be highlighted with bold letters as shown in the following figure:

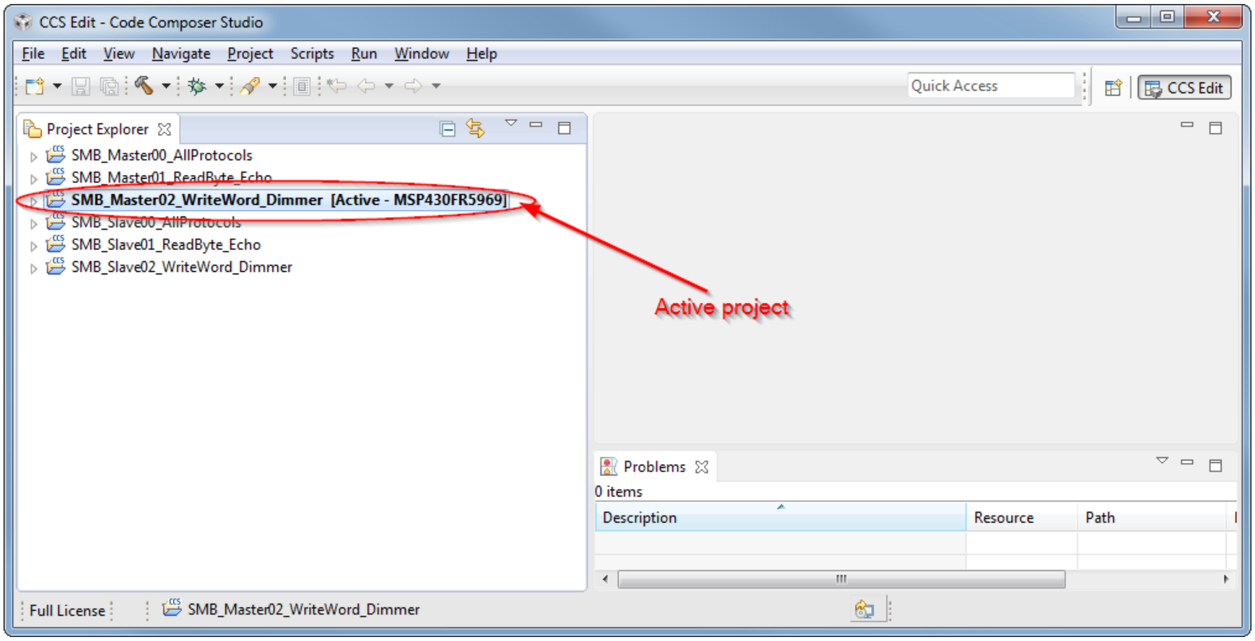

CCS Edit - Code Composer Studio
File Edit View Navigate Project Scripts Run Window Help
Quick Access
CCS Edit
Project Explorer
SMB_Master00_AllProtocols
SMB_Master01_ReadByte_Echo
SMB_Master02_WriteWord_Dimmer [Active - MSP430FR5969]
SMB_Slave00_AllProtocols
SMB_Slave01_ReadByte_Echo
SMB_Slave02_WriteWord_Dimmer
Active project
Problems
0 items
Description
Resource
Path
Full License
SMB_Master02_WriteWord_Dimmer

Figure 4: Active project in CCS

The project can be built using Ctrl+B, Menu→Project→Build All, or 🔨.; and downloaded using F11, Menu→Run→Debug, or 🐞).

The example can now be executed.

3 Example Descriptions

3.1 General Instructions

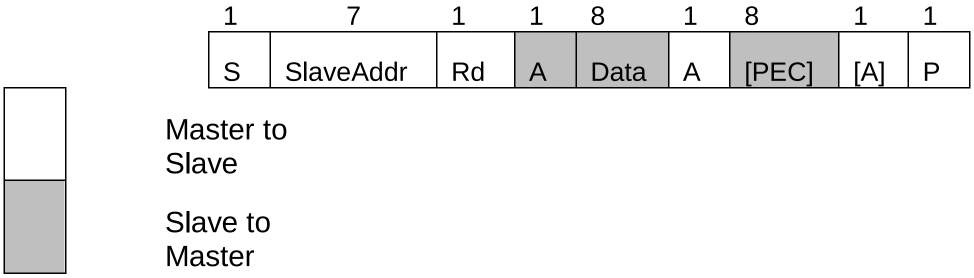
This section contains instructions that are common to all SMBus examples.

Specific instructions for each example are provided after these general instructions.

3.1.1 Nomenclature

The SMBus protocols shown for all examples have the following syntax:

Example:



1	7	1	1	8	1	8	1	1
S	SlaveAddr	Rd	A	Data	A	[PEC]	[A]	P

□ Master to Slave

▨ Slave to Master

S: Start/Repeated Start

SlaveAddr: Slave address (found in Table 2)

Wr: Write bit (0)

Rd: Read bit (1)

A: Acknowledge (‘1’ = NACK, ‘0’ = ACK)

P: Stop

[PEC]: Optional PEC if enabled

[A]: Optional ACK/NACK slot if PEC is enabled

3.1.2 Hardware Connection

A basic connection shown in Figure 5 between Master and Slave is required in order to execute the examples.

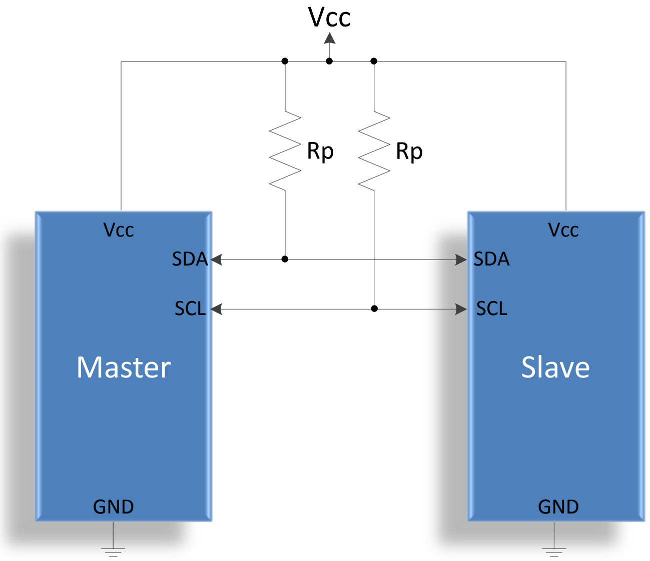


Figure 5: Basic connection between Master and Slave

The pins used for SDA and SCL for the supported derivatives are described in Section 2.2.

The value of the pull-up resistors (Rp) will depend on the system's VDD and the bus' capacitance. Typical values are in the range of 1KΩ-10KΩ.

3.1.3 Building and executing the examples

1. Select an example from Table 2. All examples include projects and source code for both a master device and a slave device. These examples are named as follows:

    - SMB_Master*XX_NameOfProject*
    - SMB_Slave*XX_NameOfProject*

2. Select a MSP430 device for the master and slave. All projects can be executed in the supported devices listed in Table 1. Note that the master and slave don't have to be from the same family of devices necessarily.
3. Connect master and slave devices as described in Section 3.1.2.
4. Open, build and download the corresponding projects as described in Section 2.3.
5. Check the results as described in the "Executing the Example" section of the individual example.

3.2 00_AllProtocols

The purpose of this example is to show the implementation of all the SMBus 2.0 protocols.

3.2.1 Master Implementation

The example sends each protocol one by one as shown in the following flow diagram:

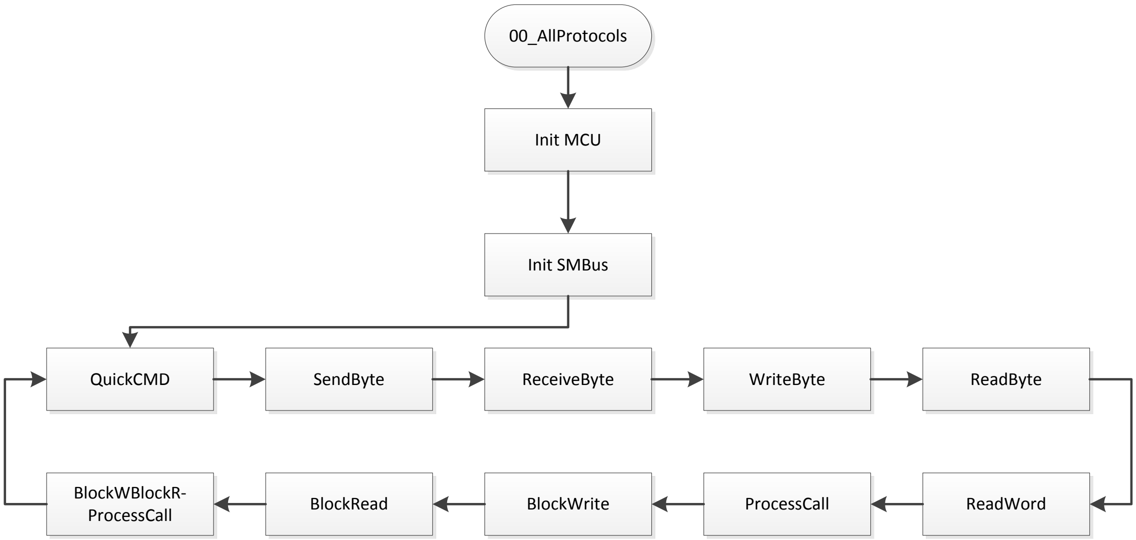


Figure 6: Master00_AllProtocols flow diagram

The data sent for each command can be modified as needed, but the default example expects the following behavior:

- *QuickCMD*: The slave should simply toggle its LED0
- *SendByte*: Sets the pointer to read the Demo Version
- *ReceiveByte*: Reads the pointer which is expected to be the Demo version set by *SendByte*
- *WriteByte*: Writes the value 0xAA to the slave’s IOPort (more details in Section 3.2.2)
- *ReadByte*: Read IOPort (more details in Section 3.2.2)
- *ReadWord*: Read ADC result from slave (more details in Section 3.2.2)
- *ProcessCall*: Gets the result of multiplying 02h x 10h
- *BlockWrite*: Write a string on the slave
- *BlockRead*: Reads the string written in *BlockWrite*
- *BlockWBlockR-ProcessCall*: Calculate CRC of three bytes: 01h, 02h, 03h

3.2.2 Slave Implementation

The slave implements different commands using various SMBus protocols. The list of commands is shown in the following table:

Table 6: Slave00_AllProtocols – Slave commands

Function	CMD	SMBus Protocol
Quick_Led_Toggle	-	QUICK_COMMAND
Read_Reg_Ptr	-	RECEIVE_BYTE
Set_Reg_Ptr	00h-03h	SEND_BYTE
Write_Reg	10h-13h	WRITE_BYTE
Read_Reg	20h-23h	READ_BYTE
Read_ADC_ch	30h	READ_WORD
Mult_Bytes	40h	PROCESS_CALL
Write_String	50h	BLOCK_WRITE
Read_String	51h	BLOCK_READ
Calc_CRC	60h	BLOCKW_BLOCKR_PROCESS_CALL

* For a description of the SMBus Protocols, please refer to the SMBus 2.0 specification.

The functions of the commands are very simple, but they are intended to be used for demonstration purposes or as a template for developers. These functions are described in more detail in the following sections.

3.2.2.1 Quick_Led_Toggle

Description: Toggles LED0.

Protocol: Quick Command.

Format:

1	7	1	1	1
S	SlaveAddr	Wr	A	P

3.2.2.2 Read_Reg_Ptr

Description: Read the contents of a virtual register pointer, initialized to Reg0, or set by *Set_Reg_Ptr*. The slave defines the following 4 virtual registers:

Table 7: Virtual registers of Slave00_AllProtocols example

Register	Description
Reg0	SMBus Status (check definition of SMBus_Status in SMBus Library documentation)
Reg1	SMBus Control (check definition of SMBus_Control in SMBus Library documentation)
Reg2	IOPort. When written, it writes to PxOUT register of the IOPort defined in Table 3; when read, it reads PxIN register of the same IOPort. Note that an Input switch defined in Table 3 is implemented in the same IOPort, allowing users to read the status of the switch by reading Reg2.
Reg3	Demo Version. Read-only register returning the version.

Protocol: Receive Byte

Format:

1	7	1	1	8	1	8	1	1
S	SlaveAddr	Rd	A	**Resp**	A	[PEC]	[A]	P

Resp: Contents of virtual register

3.2.2.3 Set_Reg_Ptr

Description: Changes the virtual register pointer. The contents of the register can then be read using *Read_Reg_Ptr.*

Protocol: Send Byte

Format:

1	7	1	1	8	1	8	1	1
S	SlaveAddr	Rd	A	**CMD**	A	[PEC]	[A]	P

CMD:

0x00: Change pointer to Reg0

0x01: Change pointer to Reg1

0x02: Change pointer to Reg2

0x03: Change pointer to Reg3

3.2.2.4 Write_Reg

Description: Writes directly to a virtual register defined in Table 7.

Protocol: Write Byte

Format:

1	7	1	1	8	1	8	1	8	1	1
S	SlaveAddr	Rd	A	**CMD**	A	**Data**	A	[PEC]	[A]	P

CMD:

0x10: Write Reg0

0x11: Write Reg1

0x12: Write Reg2

0x13: Write Reg3

Data: Byte written to virtual register.

3.2.2.5 Read_Reg

Description: Return the contents of a virtual register defined in Table 7.

Protocol: Read Byte

Format:

1	7	1	1	8	1	1	7	1	1	8	1	8	1	1
S	SlaveAddr	Wr	A	**CMD**	A	S	SlaveAddr	Rd	A	**Resp**	A	[PEC]	[A]	P

CMD:

0x20: Read Reg0

0x21: Read Reg1

0x22: Read Reg2

0x23: Read Reg3

Resp: Contents of the virtual register.

3.2.2.6 Read_ADC_ch

Description: Return the latest ADC result of the channel specified in Table 3. Note that this example performs continuous conversions of the ADC channel triggered by a timer.

Protocol: Read Word

Format:

1	7	1	1	8	1	1	7	1	1	8	1	8	1
S	SlaveAddr	Wr	A	**CMD**	A	S	SlaveAddr	Rd	A	**RespL**	A	**RespH**	A

8	1	1
[PEC]	[A]	P

CMD: 0x30

RespL/RespH: Latest ADC result.

3.2.2.7 Mult_Bytes

Description: Returns the result of the multiplication of two bytes.

Protocol: Process Call.

Format:

1	7	1	1	8	1	8	1	8	1	1	7	1	1
S	SlaveAddr	Wr	A	**CMD**	A	**Data1**	A	**Data2**	A	S	SlaveAddr	Rd	A

8	1	8	1	8	1	1
RespL	A	**RespH**	A	[PEC]	[A]	P

CMD: 0x40

Data1/Data2: Multiplier and multiplicand

RepL/RespH: Result of multiplication

3.2.2.8 Write_String

Description: Write a string in memory.

Protocol: Block Write

Format:

1	7	1	1	8	1	8	1	8	1	…	8	1	8	1	1
S	SlaveAddr	Wr	A	**CMD**	A	**Len**	A	**Data1**	A	…	**DataN**	A	[PEC]	[A]	P

CMD: 0x50

Len: Length of string (1 to 32)

Data1...DataN: 1 to 32 bytes of data which will be written to string

3.2.2.9 Read_String

Description: Return a string stored in memory.

Protocol: Block Read

Format:

1	7	1	1	8	1	1	7	1	1	8	1	8	1	…
S	SlaveAddr	Wr	A	**CMD**	A	S	SlaveAddr	Rd	A	**Len**	A	**Resp1**	A	…

8	1	8	1	1
RespN	A	[PEC]	[A]	P

CMD: 0x51

Len: Length of string (1 to 32)

Resp1…RespN: 1 to 32 bytes of data with the contents of the string

3.2.2.10 Calc_CRC

Description: Calculates and returns the CRC of an array of bytes.

Protocol: Block Write Block Read Process Call

Format:

1	7	1	1	8	1	8	1	8	1	…	8	1	1	7	1	1
S	SlaveAddr	Wr	A	**CMD**	A	**Len**	A	**Data1**	A	…	**DataN**	A	S	SlaveAddr	Rd	A

8	1	8	1	…	8	1	8	1	1
RespLen	A	**Resp1**	A	**…**	**RespN**	A	[PEC]	[A]	P

CMD: 0x60

Len: Length of the array of bytes (1 to 32)

Data1...DataN: Array of data

RespLen: Array of data in response (2 bytes for CRC)

Resp1…RespN: Result of CRC

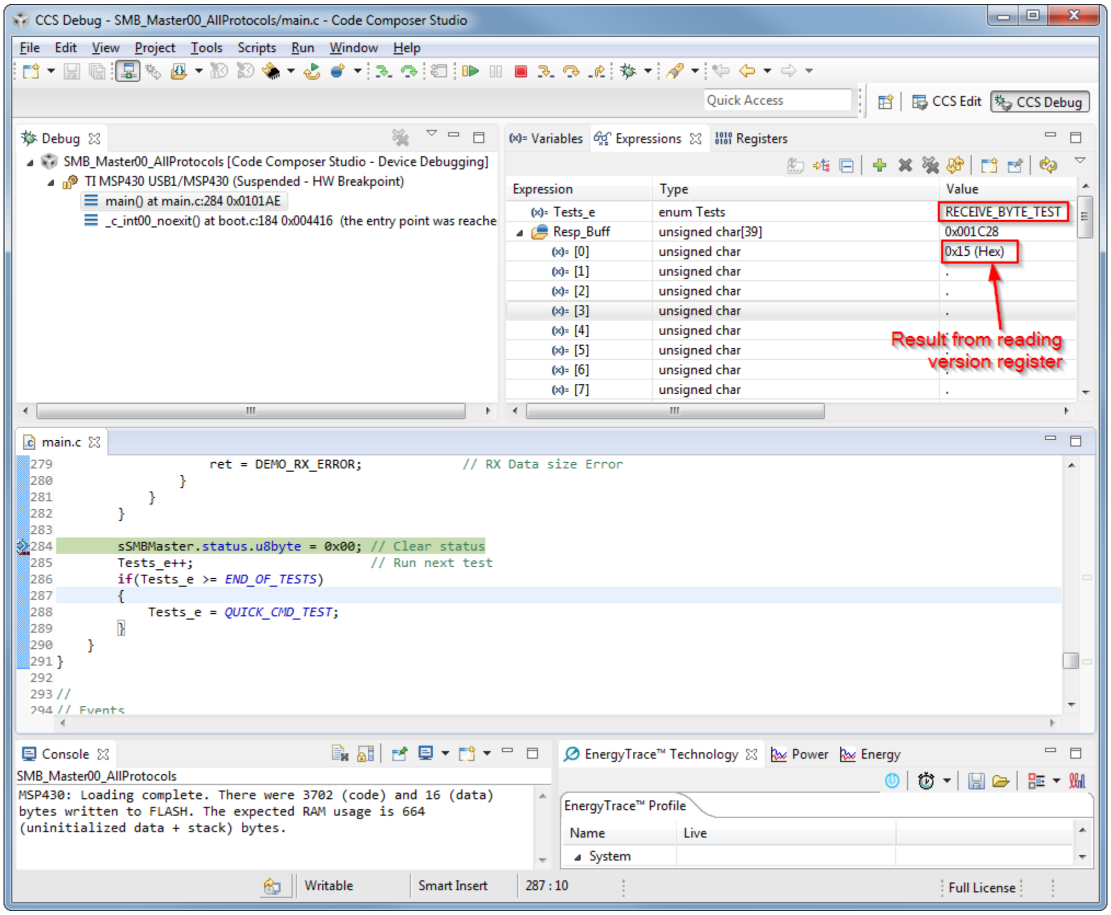
3.2.3 Executing the Example

Follow the steps described in Section 3.1.3 to load the example to both devices.

Executing the example should result in the Slave device toggling LED0 at a fast frequency due to *QuickCMD* and/or *WriteByte.*

The results can be better observed by debugging the devices or using a logic analyzer. Using the first approach, the master application can be executed step-by-step in IAR or CCS observing the responses.

Figure 7 shows the results of executing the “ReceiveByte” command which will return the contents of the Reg3 (Demo Version).



SMB_Master00_AllProtocols [Code Composer Studio - Device Debugging]
TI MSP430 USB1/MSP430 (Suspended - HW Breakpoint)
main() at main.c:284 0x0101AE
_c_int00_noexit() at boot.c:184 0x004416 (the entry point was reache

Expression	Type	Value
Tests_e	enum Tests	RECEIVE_BYTE_TEST
Resp_Buff	unsigned char[39]	0x001C28
[0]	unsigned char	0x15 (Hex)
[1]	unsigned char	.
[2]	unsigned char	.
[3]	unsigned char	.
[4]	unsigned char	.
[5]	unsigned char	.
[6]	unsigned char	.
[7]	unsigned char	.

Result from reading version register

```
                ret = DEMO_RX_ERROR;            // RX Data size Error
            }
        }
    }

    sSMBMaster.status.u8byte = 0x00; // Clear status
    Tests_e++;                       // Run next test
    if(Tests_e >= END_OF_TESTS)
    {
        Tests_e = QUICK_CMD_TEST;
    }
  }
}

//
// Events
```

SMB_Master00_AllProtocols

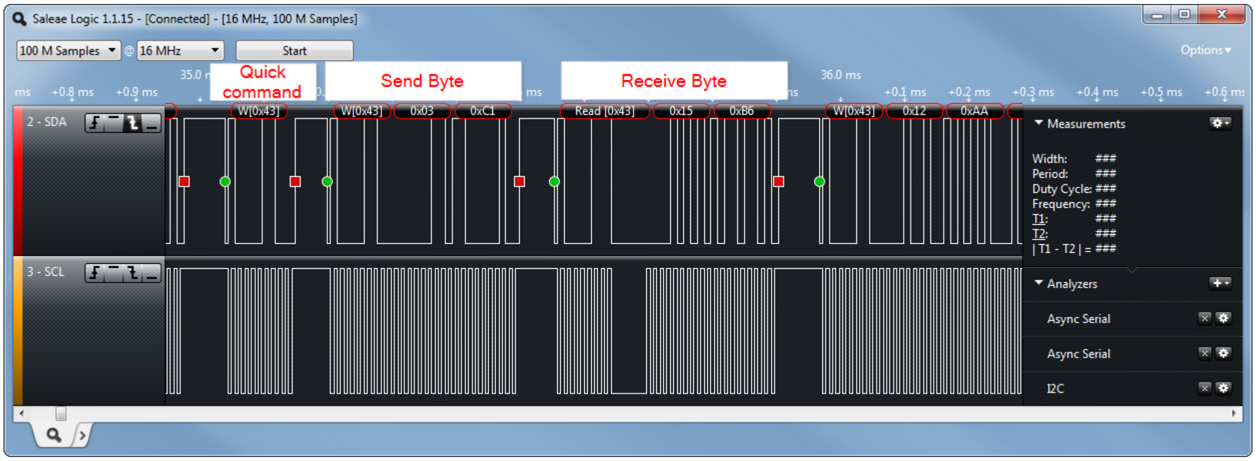
```
MSP430: Loading complete. There were 3702 (code) and 16 (data)
bytes written to FLASH. The expected RAM usage is 664
(uninitialized data + stack) bytes.
```

Name	Live
System	

EnergyTrace™ Profile

Figure 7: Debugging 00_AllProtocols example in CCS

The results can also be observed with a logic analyzer as shown in Figure 8.



Quick command

Send Byte

Receive Byte

Figure 8: Results of 00_AllProtocols example using Logic Analyzer

3.3 01_ ReadByte_Echo

This example shows a very basic implementation of ReadByte commands with a slave echoing the byte received from the master.

3.3.1 Master Implementation

The master sends an incrementing command using the Read Byte protocol. Waits for the response and checks if the response is the same as the command sent. This is shown in the following flow diagram:

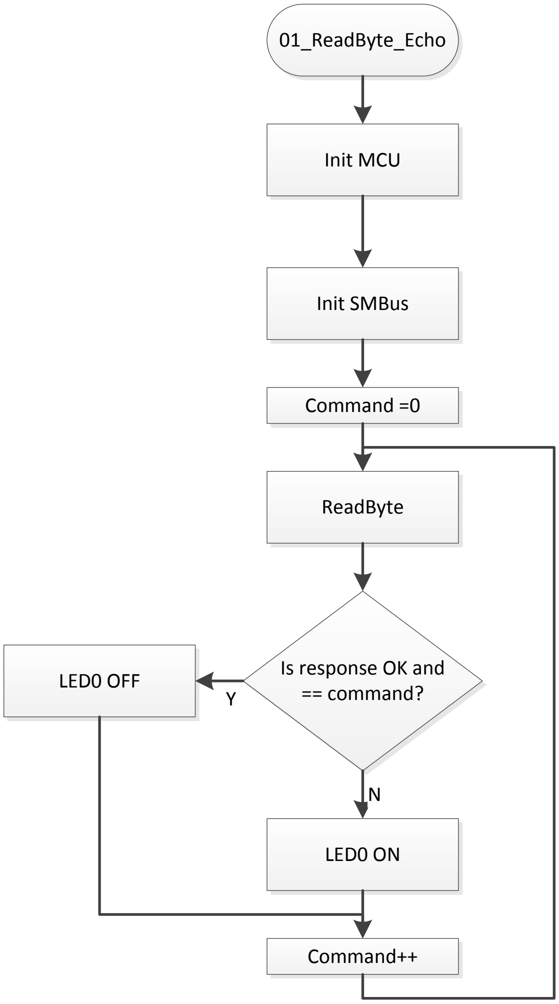


Figure 9: Master01_ReadByteEcho flow diagram

3.3.2 Slave Implementation

The slave device in this application simply echoes the command sent by the master. It doesn't have any specific commands assigned or makes any distinction between them.

For a description of the SMBus Read Byte protocol, please refer to the SMBus 2.0 specification.

3.3.2.1 Echo Command

Description: Returns an echo of the Command

Protocol: Read Byte

Format:

1	7	1	1	8	1	1	7	1	1	8	1	8	1	1
S	SlaveAddr	Wr	A	**CMD**	A	S	SlaveAddr	Rd	A	**Resp**	A	[PEC]	[A]	P

CMD: Any value

Resp: Echo of CMD

3.3.3 Executing the Example

Follow the steps described in Section 3.1.3 to load the example to both devices.

The results of the execution can be observed on LED0 of the master which should stay off; unless the communication with the slave is interrupted (i.e. pressing the reset button on the slave).

The pin used for LED0 depends on the device and it's listed on Table 4.

The results can also be observed using the debugger or a logic analyzer as shown in the following figure:

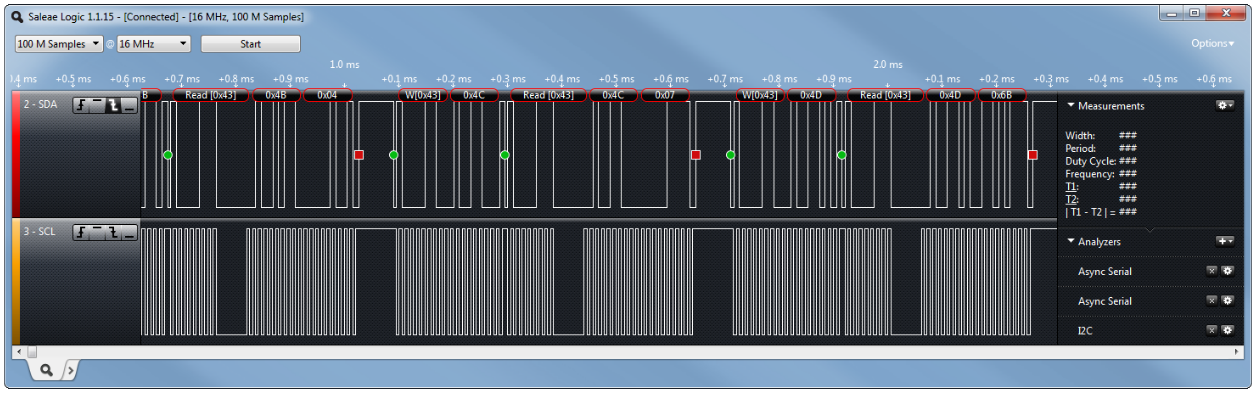


Figure 10: Results of 01_ReadByte_Echo example using Logic Analyzer

3.4 02_ WriteWord_Dimmer

This example shows a very basic implementation of a WriteWord command controlling a LED dimmer on the slave.

3.4.1 Master Implementation

The master sends a WriteWord command which changes the duty cycle of a PWM controlled by the slave. The duty cycle starts at 0%, increases gradually to 100% and restarts.

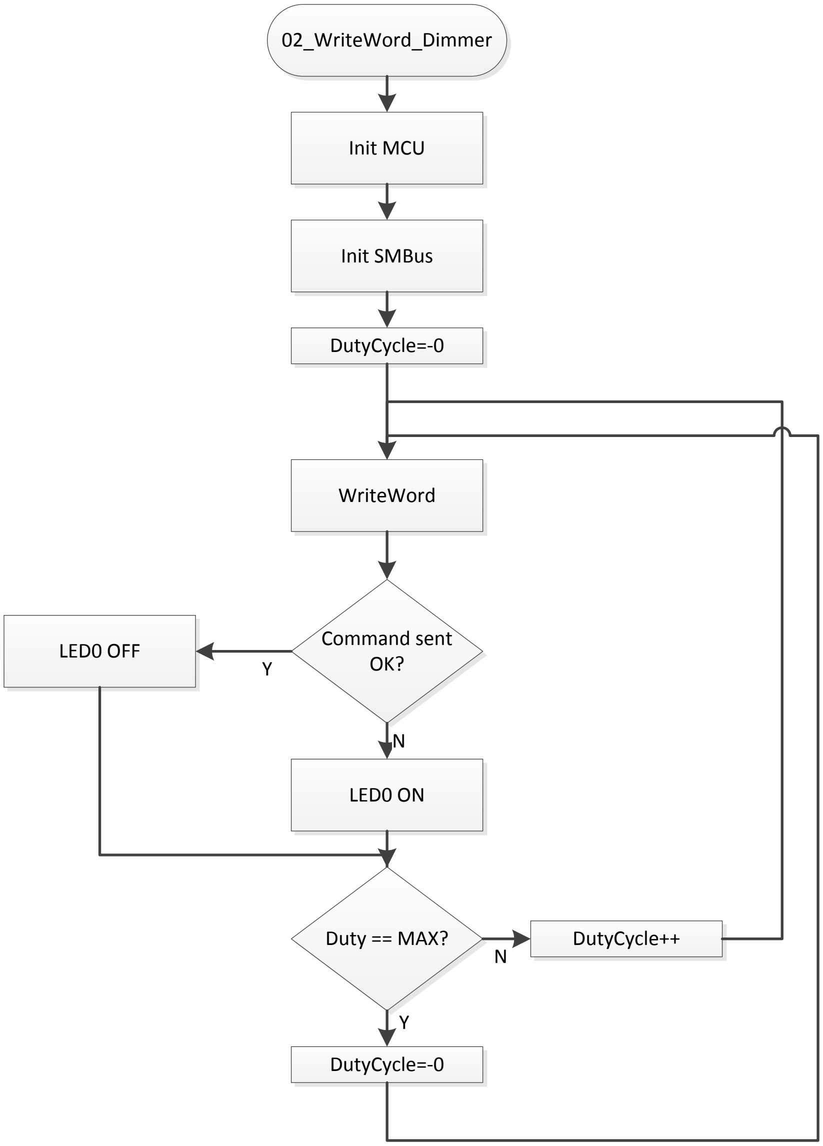


Figure 11: Master02_WriteWord_Dimmer flow diagram

3.4.2 Slave Implementation

The slave device in this application implements a single Write Word command which controls the PWM duty cycle. The PWM output depends on the device and is described in Table 5.

For a description of the SMBus Write Word protocol, please refer to the SMBus 2.0 specification.

3.4.2.1 Set_Duty_Cycle

Description: Sets the duty cycle of the PWM

Protocol: Write Word

Format:

1	7	1	1	8	1	8	1	8	1	8	1	1
S	SlaveAddr	Rd	A	**CMD**	A	**DataL**	A	**DataH**	A	[PEC]	[A]	P

CMD: 0x10

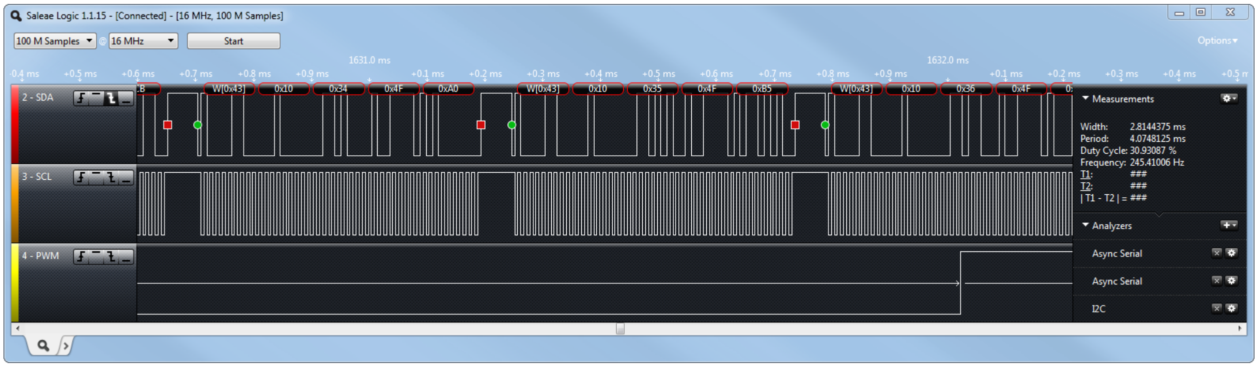
DataL/DataH: Duty cycle from 0x0000 to 0xFFFF. With 0x0000 being 0% and 0xFFFF being 100%.

3.4.3 *Executing the Example*

Follow the steps described in Section 3.1.3 to load the example to both devices.

The results of the execution can be observed if an LED is connected to the PWM output shown in Table 5. The LED will increase gradually until it reaches 100% and then it will restart.

The results can also be observed using a logic analyzer as shown in the following figure:



Saleae Logic 1.1.15 - [Connected] - [16 MHz, 100 M Samples]

100 M Samples @ 16 MHz

Measurements

Width: 2.8144375 ms
Period: 4.0748125 ms
Duty Cycle: 30.93087 %
Frequency: 245.41006 Hz
T1: ###
T2: ###
| T1 - T2 | = ###

Figure 12: Results of 02_WriteWord_Dimmer example using Logic Analyzer