

TI Designs

SMBus Design Using MSP430™ Design Guide



TI Designs

TI Designs provide the foundation that you need including methodology, testing and design files to quickly evaluate and customize the system. TI Designs help you accelerate your time to market.

Design Resources

TIDM-SMBUS	Tool Folder Containing Design Files
MSP430FR5969	Product Folder
MSP430G2553	Product Folder
TMP006	Product Folder



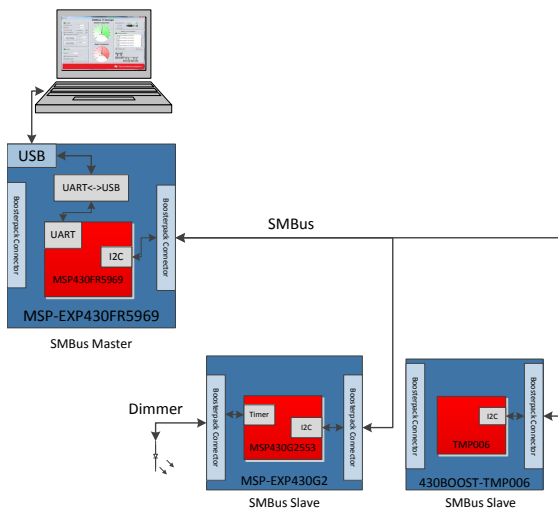
[ASK Our E2E Experts](#)
[WEBENCH® Calculator Tools](#)

Design Features

- Low Power
- Cost Effective
- Shows Implementation of SMBusLib as Master and Slave
- Uses TMP006 to Measure Temperature of Objects Remotely
- Simple LED Dimmer Implementation Using MSP430™ as SMBus Slave
- Easy-to-use GUI

Featured Applications

- Mobile Computing
- Battery-Powered Applications



An IMPORTANT NOTICE at the end of this TI reference design addresses authorized use, intellectual property matters and other important disclaimers and information.

All trademarks are the property of their respective owners.

1 System Description

This design utilizes existing Launchpad™ evaluation kits and BoosterPack™ plug-in modules to demonstrate the implementation of an SMBus-based system using TI's [MSP430™ SMBus Library](#).

The SMBus system consists of a master device (MSP-EXP430FR5969) that controls two slave devices, 430BOOST-TMP006 and MSP-EXP430G2.

The 430BOOST-TMP006, containing a TMP006 infrared thermopile sensor, can be used to measure the temperature of objects remotely. The MSP-EXP430G2, using a MSP430G2553 ultra-low-power value line device, implements a simple LED dimmer with three virtual registers.

The MSP-EXP430FR5969, acting as SMBus master and containing a MSP430FR5969 ultra-low power microcontroller with FRAM, works as a bridge between UART and SMBus, allowing a PC application to control the slave devices.

A GUI, provided as source code and executable, allows easy configuration and display of the system.

1.1 MSP-EXP430FR5969

The MSP-EXP430FR5969 Launchpad development kit is an easy-to-use Evaluation Module for the MSP430FR5969 microcontroller. The kit contains everything needed to start developing on the MSP430 ULP FRAM platform, including on-board emulation. This emulation can not only be used for programming, debugging, and Energy Measurements, but it can also provide a back-channel UART port to communicate with a PC.

The MSP-EXP430FR5969 is used as the SMBus master in this design, but it can also be used as SMBus slave (as explained in [Section 6](#)). This design was built on Revision 2.0 of the MSP-EXP430FR5969, but the version available on TI eStore may be different; users should check for compatibility issues.

The LaunchPad user's guide includes more details about this board, including schematics, description of jumpers, and a hardware change log.

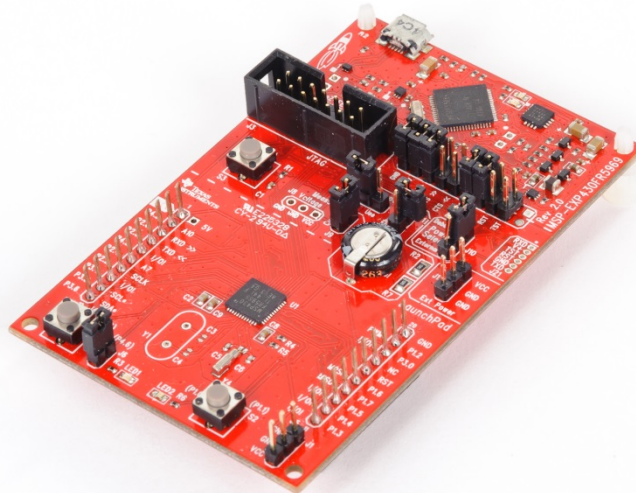


Figure 1. MSP430FR5969

1.1.1 MSP430FR5969

The MSP430 ultra-low-power (ULP) FRAM platform combines uniquely embedded FRAM and a holistic ultra-low-power system architecture, allowing innovators to increase performance at lowered energy budgets. FRAM technology combines the speed, flexibility, and endurance of SRAM with the stability and reliability of flash at much lower power.

Among other features, the MSP430FR59xx includes the following:

- Up to 64KB of FRAM memory
- A 12-bit analog-to-digital converter
- Two enhanced universal serial communication interfaces:
 - One supporting UART, IrDA, and SPI (eUSCI_A)
 - One supporting I²C and SPI (eUSCI_B)
- DMA
- Five 16-bit timers
- AES256 and CRC16 hardware accelerators
- Up to 40 I/O pins

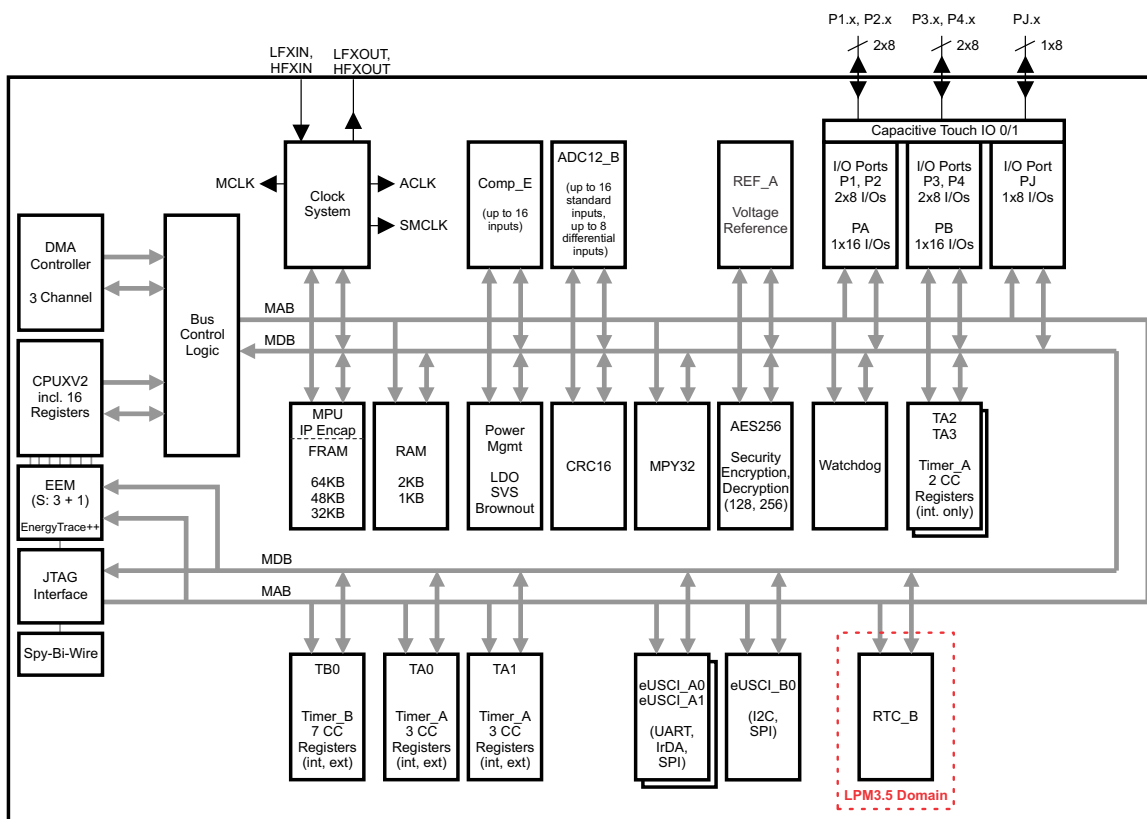


Figure 2. Functional Block Diagram—MSP430FR59xx

1.2 MSP-EXP430G2

The MSP-EXP430G2 LaunchPad™ is an easy-to-use flash programmer and debugging tool for the MSP430G2xx Value Line microcontrollers. The LaunchPad features everything needed to start developing on an MSP430 microcontroller device. The MSP-EXP430G2 features a 14 / 20-pin DIP socket, on-board buttons and LEDs, and BoosterPack™-compatible pinouts that support a wide range of plug-in modules for added functionality such as wireless, displays, and more. The included eZFET circuitry for on-board emulation not only allows for programming and debugging of the device, but it also provides a back-channel UART that can be used to communicate with a PC.

The MSP-EXP430G2 functions as an SMBus slave in this design, but it can also function as SMBus master as explained in [Section 6](#). This design was built on Revision 1.5 of the MSP-EXP430G2, but the version available on TI eStore may be different; users should check for compatibility issues.

The LaunchPad user's guide includes more details about this board, including schematics, a description of jumpers, and a hardware change log.

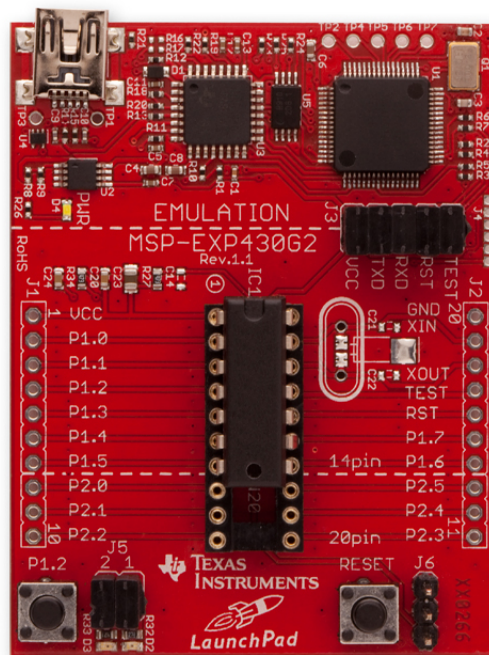


Figure 3. MSP-EXP430G2

1.2.1 MSP430G2553

The MSP430G2x13 and MSP430G2x53 series are ultra-low-power mixed-signal microcontrollers with built-in 16-bit timers, up to 24 I/O capacitive-touch enabled pins, a versatile analog comparator, and built-in communication capability using the universal serial communication interface. In addition, the MSP430G2x53 family members have a 10-bit analog-to-digital (A/D) converter.

Typical applications include low-cost sensor systems that capture analog signals, convert them to digital values, and then process the data for display or for transmission to a host system.

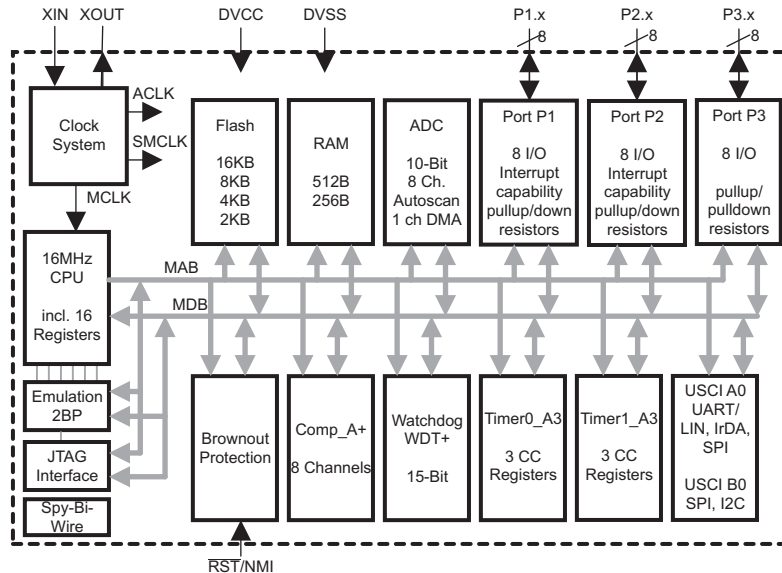


Figure 4. Functional Block Diagram—MSP430G2x53

1.3 430BOOST-TMP006

The TMP006 BoosterPack™ is a simple adapter board that allows the TMP006EVM circuit board to be used with the MSP430™ LaunchPad™. The TMP006EVM is an evaluation board designed to allow full evaluation of the TMP006 device. The TMP006EVM features the TMP006 device with the required specific PCB layout and some simple support circuitry.

The 430BOOST-TMP006, together with the TMP006EVM, acts as an SMBus slave capable of measuring temperature of objects remotely. This design was built on Revision A of the TMP006EVM, but the version available on TI eStore may be different; users should check for compatibility issues.

More information about both boards, including schematics and pinout, is available in the respective user's guides.



Figure 5. 430BOOST-TMP006 with TMP006EVM

1.3.1 TMP006

The TMP006 and TMP006B are the first in a series of temperature sensors that measure the temperature of an object without the need to make contact with the object. This sensor uses a thermopile to absorb the passive infrared energy emitted from the object being measured and uses the corresponding change in thermopile voltage to determine the object temperature. The thermopile voltage is digitized and reported with the die temperature through serial communication.

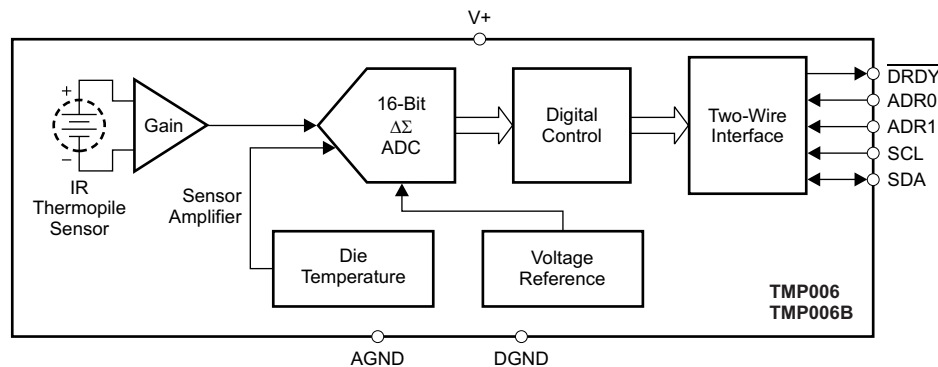


Figure 6. Functional Block Diagram—TMP006

2 Block Diagram

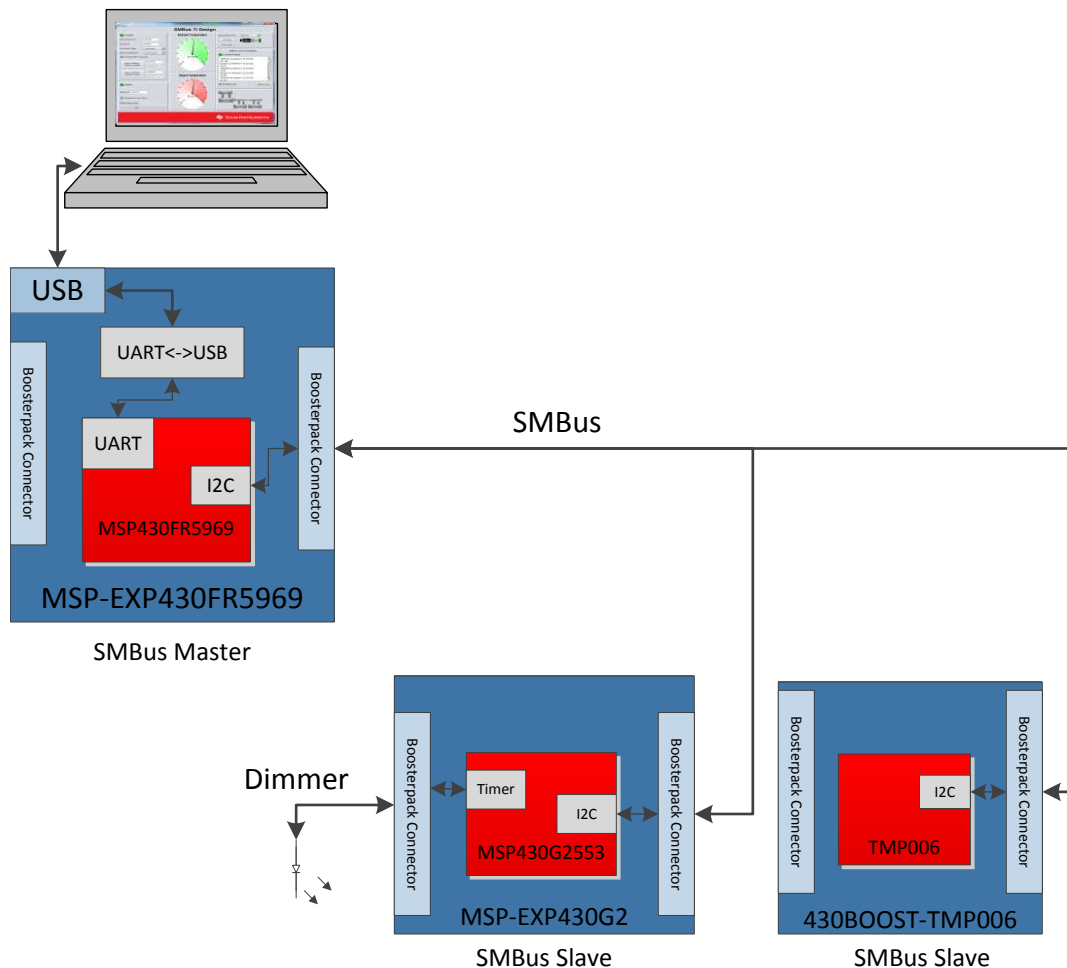


Figure 7. Block Diagram of the System

3 System Design Theory

The purpose of this TI Design is to show the implementation of an SMBus system using TI's MSP430™ SMBus Library. The design includes the implementation of a MSP430 as an SMBus slave dimmer, a TMP006 as an SMBus slave temperature sensor, a PC GUI displaying information about the devices and allowing control of them, and another MSP430 used as SMBus master and acting as a bridge between the GUI and the slave devices.

3.1 SMBus

The System Management Bus (SMBus) is a lightweight two-wire interface based on the principles of I²C, commonly used as a control bus and for power-management tasks in computing, mobile computing, and battery-operated applications. A device performing data transfers on the bus can be considered a master, which is the device initiating a transaction and drives the clock, or a slave, which is the target of an SMBus transaction driven by the master. Both the master and the slave can act as transmitters or as receivers.

SMBus 2.0 shares many similarities with I²C, but some of the most relevant differences include the following:

- Time-out detection occurs when a device stretches the clock for too long
- Packet Error Checking (PEC) can be optionally appended at the end of each transaction, allowing the bus to automatically validate packets
- I²C only defines a PHY and Data-Link layers, but SMBus defines a network layer with different SMBus protocols, which can be used to exchange data between devices
- Optional use of additional lines are available, such as SMBAlert# and SMBSUS#.

For more information about SMBus, please refer to <http://smbus.org/specs/>. For more information about I²C, please refer to http://www.nxp.com/documents/user_manual/UM10204.pdf

3.1.1 MSP430 SMBus Library

The MSP430 SMBus Library is a royalty-free API stack intended to enable easy and reliable communication for MSP430 in an SMBus system. The library provides support for applications where the MSP430 acts as the master or the slave. The library also includes the implementation of all layers of SMBus 2.0 as well as application examples allowing for a faster time to market. SMBus Library 1.0 includes support for MSP430FR5xx/FR6xx and MSP430G2xx3 devices.

For more information about SMBus Library, please refer to [TIDM-SMBUS](#).

3.2 SMBus Master

The SMBus master initializes the slave devices and acts as a bridge between them and the PC. This functionality is shown in [Figure 8](#).

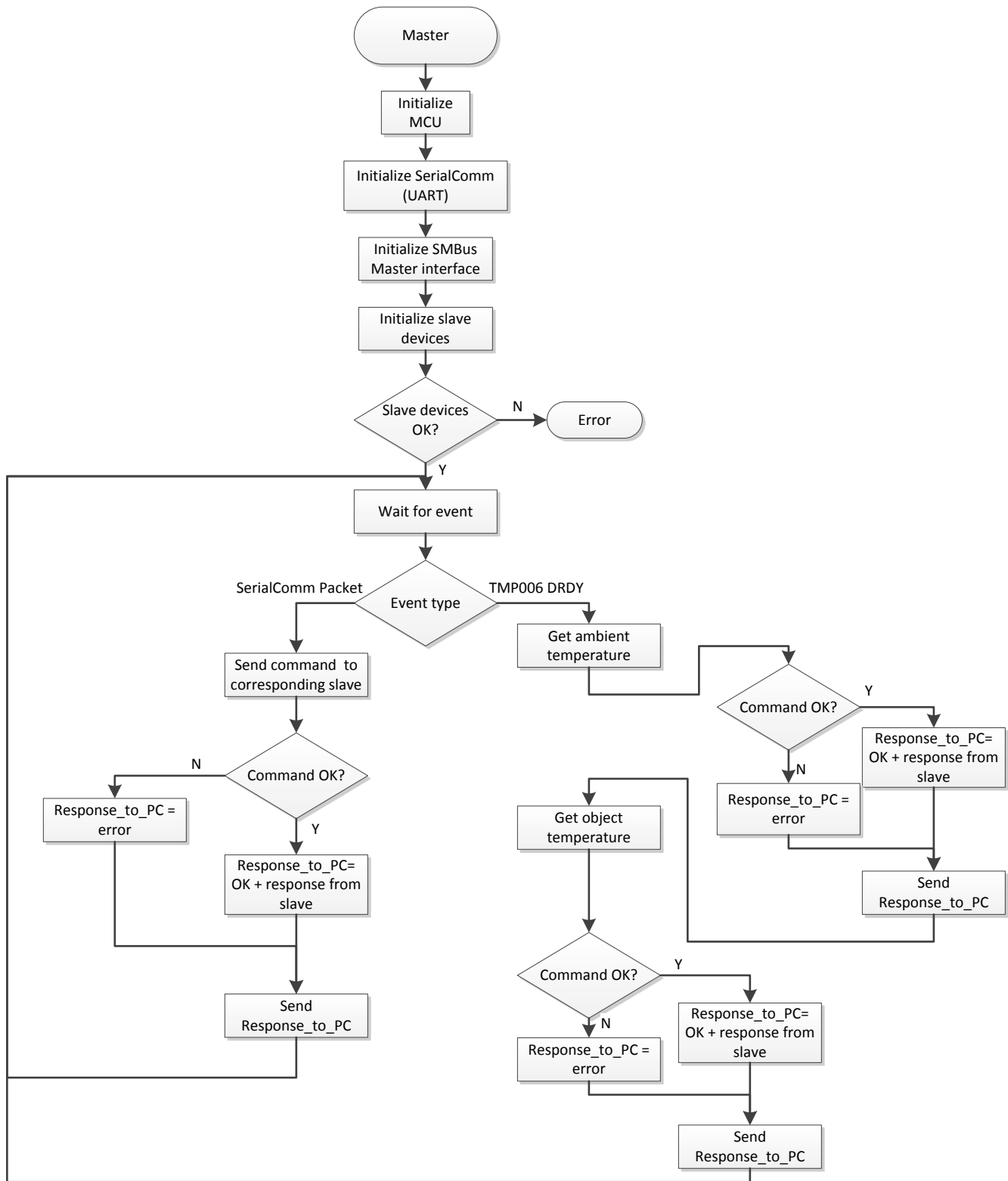


Figure 8. Flow diagram—Master Application

After initialization, the master device goes to an idle state in low power mode waiting for any of the following events:

- A packet from SerialComm (UART) that indicates that the PC is sending a new command
- An asynchronous event from one of the slaves, in this case signaled by the assertion of DRDY signal from TMP006

The implementation of TMP006 DRDY is explained in more detail in [Section 3.4.1](#), while the SerialComm protocol is explained in [Section 3.2.1](#).

3.2.1 SerialComm (UART) Protocol Between Master and PC

The MSP-EXP430FR5969 eZ-FET circuitry includes support for a backchannel UART, which is used to communicate with the PC. A custom protocol, shown in [Table 1](#), was implemented to exchange data with the PC GUI:

Table 1. Packet from PC to Master

HEADER	LENGTH	COMMAND	DATA
0x80	LEN	CMD	D1...Dn

Header— Indicates beginning of the frame. Always 0x80

LEN— Number of bytes, including Command and Data

CMD— Command sent to the master device, as explained in [Table 2](#)

D1...Dn— Data bytes 1 to n

The commands supported by the master are as follows:

Table 2. Commands Supported by Master Using SerialComm

COMMAND	CMD	DATA	SMBUS PROTOCOL
TMP006_Write_Config	'C' (0x43)	LSB,MSB	WriteWord
TMP006_Read_Config	'c' (0x63)	-	ReadWord
TMP006_Read_Manuf_ID	'm' (0x6D)	-	ReadWord
TMP006_Read_Device_ID	'd' (0x64)	-	ReadWord
TMP006_Read_Sensor_V	'v' (0x76)	-	ReadWord
TMP006_Read_Ambient_T	't' (0x74)	-	ReadWord
DIM430_Write_Config	'Y' (0x59)	LSB,MSB	WriteWord
DIM430_Read_Config	'y' (0x79)	-	ReadWord
DIM430_Read_Device_ID	'p' (0x70)	-	ReadWord
DIM430_Write_DutyCycle	'U' (0x55)	LSB,MSB	WriteWord
DIM430_Read_DutyCycle	'u' (0x75)	-	ReadWord

Upon receiving a packet, the master validates its contents and starts an SMBus transfer if needed. Each command access a particular register of the slave devices TMP006 or DIM430. More information about the slave registers can be found in [Section 3.4.3](#) and [Section 3.3.2](#) respectively.

The result of the SMBus transfer is reported back to the PC using the packet shown in [Table 3](#).

Table 3. Response from Master to PC

HEADER	LENGTH	SMB CONTROL	SMB STATUS	SMB TXLEN
0x80	LEN	Ctrl	Stat	TxLen
SMB TXADDR	SMB TXCMD	SMB TXDATA	RESPONSE ID	SMB RESPDATA
TxAddr	TxCMD	TxD1...TxDn	RespID	RxD1...RxDn

Header— Indicates beginning of the frame. Always 0x80.

LEN— Number of bytes, including all fields from SMB Control to SMB RespData

Ctrl— SMBus Control byte with the following contents:

Table 4. SMBus Control Byte

BIT7:BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
Reserved	Master	PHYEn	intEn	swackEn	pecEn

Check definition of SMBus_Ctrl in SMBusLib for more details.

Stat— SMBus Status byte with the following contents:

Table 5. SMBus Status Byte

BIT7:BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
Reserved	cmdErr	byteOvrErr	packOvrErr	packErr	toErr	pecErr

Check definition of SMBus_Status in SMBusLib for more details.

TxLen— Number of bytes sent to the slave device via SMBus including command and data

TxAddr— Address of SMBus slave device

TxCMD— SMBus command sent to slave device

TxD1...TxDn— Data sent to the slave device via SMBus

RespID— Response ID indicating if the device was sent correctly, as detailed in [Table 6](#).

RxD1...RxDn— Response data received from the slave device via SMBus.

Table 6. Response IDs Sent by Master Using SerialComm

RESPONSE	RESPID	RXD1...RXDN	DESCRIPTION
SC_OK	0x00	-	Command executed OK, no RX data.
SC_RX_ERROR	0x01	-	Incorrect packet received from PC
SC_SMB_ERROR	0x02	-	Error during SMBus communication
SC_TMP006_V	0x03	LSB,MSB	TMP006 Sensor_Voltage
SC_TMP006_T	0x04	LSB,MSB	TMP006 Ambient Temperature
SC_TMP006_CONFIG	0x05	LSB,MSB	TMP006 Configuration
SC_TMP006_MANID	0x06	LSB,MSB	TMP006 Manufacturer ID
SC_TMP006_DEVID	0x07	LSB,MSB	TMP006 Device ID
SC_DIM430_DEVID	0x08	LSB,MSB	DIM430 Device ID
SC_DIM430_CONFIG	0x09	LSB,MSB	DIM430 Configuration
SC_DIM430_DUTY	0x0A	LSB,MSB	DIM430 Duty cycle

For more details about the TMP006 or DIM430 register contents, please refer to [Section 3.4.3](#) and [Section 3.3.2](#) respectively.

Command Examples:

Table 7. PC Sends Command to Write DIM430 Duty Cycle to 50%

HEADER	LENGTH	COMMAND	DATA
0x80	0x03	'U' (0x55)	0x32 0x00

Table 8. Expected Response from Master

HEADER	LENGTH	SMB CONTROL	SMB STATUS	SMB TXLEN
0x80	0x08	0x1C	0x00	0x04
SMB TxAddr	SMB TxCmd	SMB TxData	Response ID	SMB RespData
0x43	0x01	0x32 0x00	0x00	-

Table 9. PC Sends Command to Read TMP006 Device ID

HEADER	LENGTH	COMMAND
0x80	0x01	'd' (0x64)

Table 10. Expected Response from Master

HEADER	LENGTH	SMB CONTROL	SMB STATUS	SMB TXLEN
0x80	0x08	0x1C	0x00	0x02
SMB TxAddr	SMB TxCmd	SMB TxData	Response ID	SMB RespData
0x40	0xFF	-	0x07	0x67 0X00

3.2.2 Software—Master

The master application consists of the following files:

```

SMBus Master
|--DriverLib           <- MSP430 DriverLib
|   |--MSP430FR5xx_6xx <-
|   |--inc             <-
|
|--smbuslib           <- SMBus Library
|   |--MSP430FR5xx_6xx <- PHY for FR5xx eUSCI
|   |--MSO430G2xx3    <- (ignored)
|
|--main.c             <- Main application
|--Dim430.c/h         <- DIM430 control
|--TMP006.c/h         <- TMP006 control
|--TI_SerialComm.c/h  <- SerialComm (UART to PC) protocol
|--TI_SerialComm_HAL_FR5969.c <- Hardware Abstraction for SerialComm
|--Master_HAL_FR5969.c <- Hardware Abstraction for application
|--Master_HAL.h       <- Header file for Hardware Abstraction
    
```

Figure 9. Software Files for SMBus Master Example

The application is structured in a modular way with a hardware abstraction layer (HAL), which allows easy migration to other devices. This procedure is explained in [Section 6](#).

3.2.3 Hardware—Master

The software for the master application was developed for MSP430FR5969 using the following resources:

Table 11. Master Hardware Resources—MSP430FR5969

FUNCTION	PERIPHERAL	GPIO	CONNECTION TO OTHER BOARDS
SMBus	eUSCI_B0	SDA: P1.6/UCB0SDA SCL: P1.7/UCB0SCL	SDA of both slaves SCL of both slaves
UART	eUSCI_A0	TXD: P2.0/UCA0TXD RXD: P2.1/UCA0RXD	-
LEDs	-	LED0: P1.0 LED1: P4.6	-
TMP006 DRDY	-	P3.5	DRDY of TMP006 slave

The application can be executed in practically any hardware, but the examples were developed and tested in the MSP-EXP430FR5969 LaunchPad. This LaunchPad is explained in more detail in the *MSP-EXP430FR5969 LaunchPad Development Kit User's Guide* ([slau535](#)).

The application can execute with just the external connections to the other two boards as shown in [Table 11](#); or the three boards can be stacked on top of each other. The software takes special considerations to avoid electrical conflicts when stacking the boards and this procedure is used in the Getting Started guide included in [Section 4](#).

The BoosterPack pinout when using the MSP-EXP430FR5969 as a master is shown in [Figure 10](#).

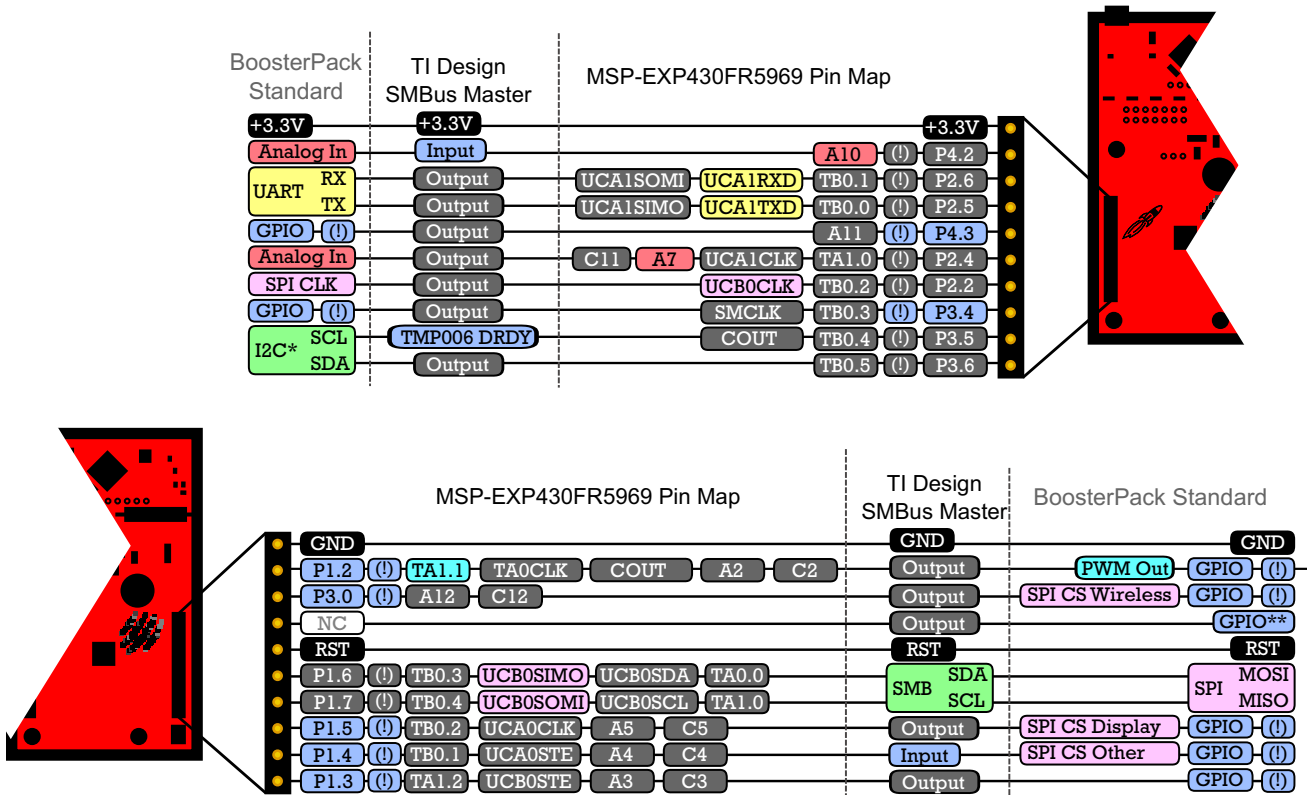


Figure 10. BoosterPack Pinout for MSP-EXP430FR5969 as Master

- “Output” pins are actively driven low by the application. These pins are not used by the master or any of the slaves, and users must take care when changing the configuration of these pins to avoid a conflict with the other boards.
- “Input” pins are floating because they are not used by the master but they are driven by a slave.
- “TMP006 DRDY” is connected to TMP006’s DRDY signal and is used by the master as explained in [Section 3.4.1](#).
- “SMB SDA” and “SMB SCL” signals are connected between the master and both slaves. Note that this configuration differs from the BoosterPack standard.

The board includes several jumpers that must be configured properly to download, debug, and execute the application. [Table 12](#) shows the configuration:

Table 12. MSP-EXP430FR5969 Configuration as Master

JUMPER	PROGRAMMING / DEBUGGING	EXECUTION – STACKED
J1	OFF	OFF
J2	Bypass	Bypass
J6	ON	ON
J9	ON	ON
J10	Debugger	Debugger
J11	OFF	OFF
J12	OFF	OFF
J13 – GND	ON	ON
J13 – 5V	ON	ON
J13 – V+	ON	ON
J13 – RTS	OFF	OFF
J13 – CTS	OFF	OFF
J13 – RXD	ON	ON
J13 – TXD	ON	ON
J13 – RST	ON	OFF
J13 – TST	ON	OFF

Note that the application can actually execute using the “Debugging” configuration but there could be problems if the three boards are stacked on top of each other. The step-by-step procedure on how to debug and execute the application is shown in [Section 4](#) and [Section 5](#).

3.3 SMBus Slave – LED Dimmer (DIM430)

This reference design shows an implementation of an SMBus slave using MSP430G2553 acting as a simple LED Dimmer. The purpose of this example is to show a simple implementation of an SMBus slave that can be used as a starting point for more complex implementations. Developers can implement more commands, registers, and functionality as needed.

The flow diagram for the slave application is shown in Figure 11.

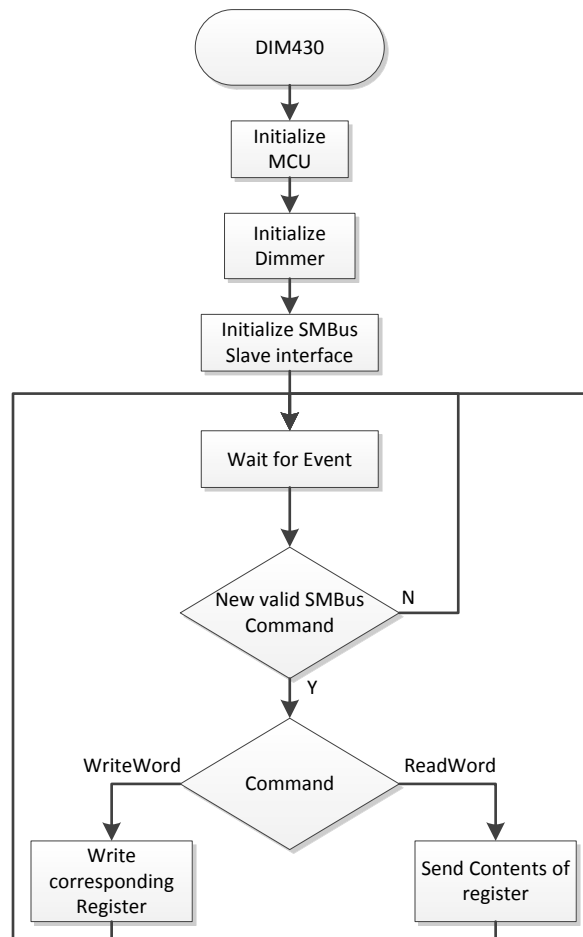


Figure 11. Flow diagram—Slave DIM430 Application

The dimmer is implemented using a Timer_A instance that is sourced by ACLK to consume low power even when the PWM is enabled. VLO (Very-Low Power Low-Frequency Oscillator) is used to source ACLK to avoid using an external crystal.

The PWM is configured to run at ~200Hz using Timer_A in Toggle/Set mode as shown in the following figure:

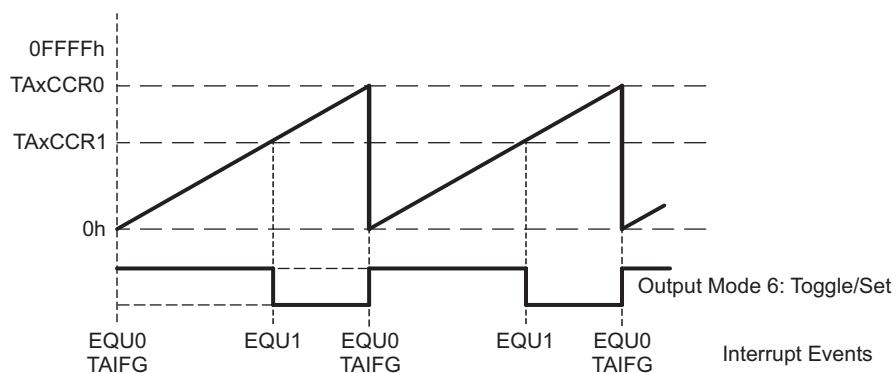


Figure 12. Timer_A Toggle/Set Mode

The granularity and tolerance of the PWM is limited by the clock source being ACLK, which is sourced by VLO. Because VLO is expected to have a typical frequency of 12 KHz and the output frequency of the PWM is 200 Hz, the granularity is calculated as follows:

$$PWMSteps = \frac{TimerCLKFreq}{PWMFreq} = \frac{12,000Hz}{200Hz} = 60 \tag{1}$$

This indicates that there are 60 steps between a 0% and a 100% duty cycle. As a result, some duty cycle percentages will result in repeated values of the duty cycle as shown in Figure 13.

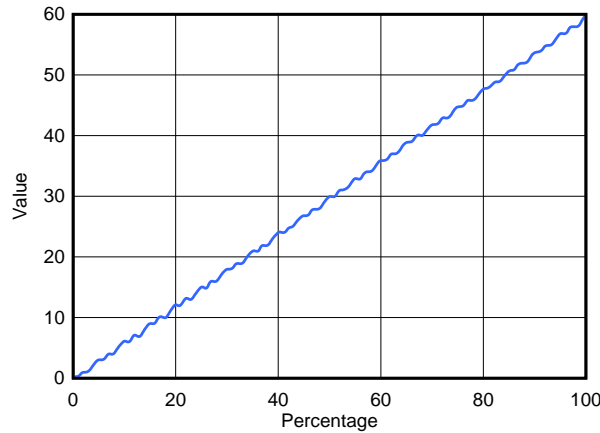


Figure 13. Duty Cycle, Percentage vs Value

It's also important to remark that the frequency of VLO can vary from 4 KHz to 20 KHz, which indicates that the PWM frequency can vary as shown in Table 13:

Table 13. PWM Frequency

PARAMETER	MIN	TYP	MAX	UNIT
PWM Frequency (from VLO, G2553)	66.66	200	333.33	Hz

Applications requiring more tolerance or better granularity on the PWM output can use an external clock source or the DCO.

The device implements three virtual registers that can be read and written by the master device to control the DIM430:

- CONFIG register enables the PWM output using CONFIG.EN0
- DUTY_CYCLE register is used to set the duty cycle percentage from 0% to 100%
- DEV_ID register returns the device ID.

These registers are explained in more detail in Section 3.3.2, and they can be read or written as explained in the following section.

3.3.1 SMBus Communication

This address of the DIM430 can be customized, but the default value is as follows:

DIM430 Slave Address	0x43
----------------------	------

Figure 14. DIM430 Slave Address

A master device can write the contents of the DIM430 registers using the SMBus Write Word protocol, and it can read their contents using the Read Word protocol:

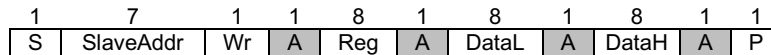


Figure 15. Write Word Protocol

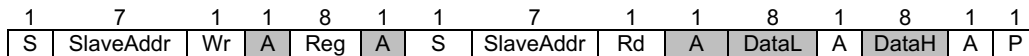
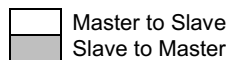


Figure 16. Read Word Protocol



- S:** Start / Repeated Start
- SlaveAddr:** Slave address
- Wr:** Write bit (0)
- Rd:** Read bit (1)
- A:** Acknowledge ('1' = NACK, '0' = ACK)
- P:** Stop
- Reg:** Register being addressed as described in [Section 3.3.2](#)
- DataL:** Low byte of Data
- DataH:** High byte of Data

Examples:

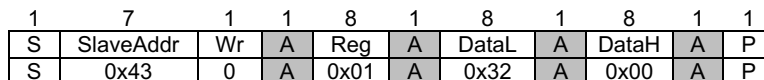


Figure 17. Master Writes Duty Cycle to 50%

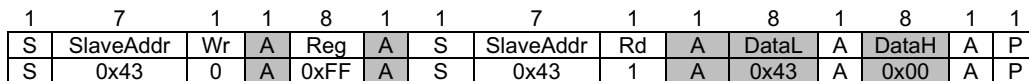


Figure 18. Master Reads DIM430 Device ID

3.3.2 Registers – Slave – DIM430

The following registers are implemented in DIM430:

Table 14. DIM430 Registers

REGISTER	ADDRESS	TYPE	RESET
CONFIG	0x00	Read / Write	0x0000
DUTY_CYCLE	0x01	Read / Write	0x0000
DEV_ID	0xFF	Read	0x0043

3.3.2.1 CONFIG Register

The CONFIG register is a read-and-write register that enables the PWM output. Setting CONFIG.EN0=1 will enable the PWM output at the current duty cycle, while clearing this bit has the opposite effect.

Address = 0x00

Reset value = 0x0000

15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-
R	R	R	R	R	R	R	R
7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	EN0
R	R	R	R	R	R	R	R/W

Figure 19. DIM430 CONFIG Register

Bits [15:1] Unused

Read-only, always returns to 0

Bit [0] EN0

0 = Disable PWM0 output

1 = Enable PWM0 output

3.3.2.2 DUTY_CYCLE Register

The DUTY_CYCLE register is a read-and-write register that sets the PWM duty cycle percentage. The duty cycle can be configured to go from 0% (0x0000) to 100% (0x0064).

Address = 0x01

Reset value = 0x0000

15	14	13	12	11	10	9	8
Duty15	Duty14	Duty13	Duty12	Duty11	Duty10	Duty9	Duty8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
7	6	5	4	3	2	1	0
Duty7	Duty6	Duty5	Duty4	Duty3	Duty2	Duty1	Duty0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Figure 20. DIM430 DUTY_CYCLE Register

Bits [15:0] Duty[15:0]

Sets the Duty Cycle percentage of PWM0. The value is limited from 0% to 100%.

3.3.2.3 DEV_ID Register

The DEV_ID register is a read-only register that returns the device ID of DIM430. The Device ID is set to 0x0043.

Address = 0xFF

Reset value = 0x0043

15	14	13	12	11	10	9	8
ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8
R	R	R	R	R	R	R	R
7	6	5	4	3	2	1	0
ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
R	R	R	R	R	R	R	R

Figure 21. DIM430 DEV_ID Register

Bits [15:0] ID[15:0]

Contains the Device ID for DIM430, fixed at 0x0043.

3.3.3 Software – Slave – DIM430

The DIM430 slave application consists of the following files:

```

SMBus Slave
|--DriverLib           <- (ignored for MSP430G2553)
|   |--MSP430FR5xx_6xx
|   |   |--inc
|   |   |--inc
|   |
|--smbusLib           <- SMBus Library
|   |--MSP430FR5xx_6xx <- (ignored)
|   |--MSO430G2xx3    <- PHY for G2xx3 USCI
|   |
|--main.c             <- Main application
|--Slave_HAL_G2553.c <- Hardware Abstraction for application
|--Slave_HAL.h        <- Header file for Hardware Abstraction
  
```

Figure 22. Software Files for SMBus DIM430 Slave Example

The application is structured in a modular way with a hardware abstraction layer (HAL), which allows easy migration to other devices. This procedure is explained in [Section 6](#).

3.3.4 Hardware – Slave – DIM430

The software for the DIM430 slave application was developed for MSP430G2553 using the following resources.

Table 15. DIM430 Slave Hardware Resources—MSP430G2553

FUNCTION	PERIPHERAL	GPIO	CONNECTION TO OTHER BOARDS
SMBus	USCI_B0	SDA: P1.7/UCB0SDA SCL: P1.6/UCB0SCL	SDA of master and TMP006 SCL of master and TMP006
Dimmer PWM	TA1 TA1.2	PWM: P2.4/TA1.2	LED1 of TMP006 BoosterPack

The application can be executed in practically any hardware, but the examples were developed and tested in the MSP-EXP430G2 LaunchPad. For more information about this LaunchPad, visit the *MSP-EXP430G2 LaunchPad Evaluation Kit User's Guide* ([slau318](#)).

The application can execute with just the external connections to the other two boards as shown in [Table 15](#); or the three boards can be stacked on top of each other. The software takes special considerations to avoid electrical conflicts when stacking the boards and this procedure is used in the Getting Started guide included in [Section 4](#).

The BoosterPack pinout when using the MSP-EXP430G2 as a slave is shown in [Figure 23](#).

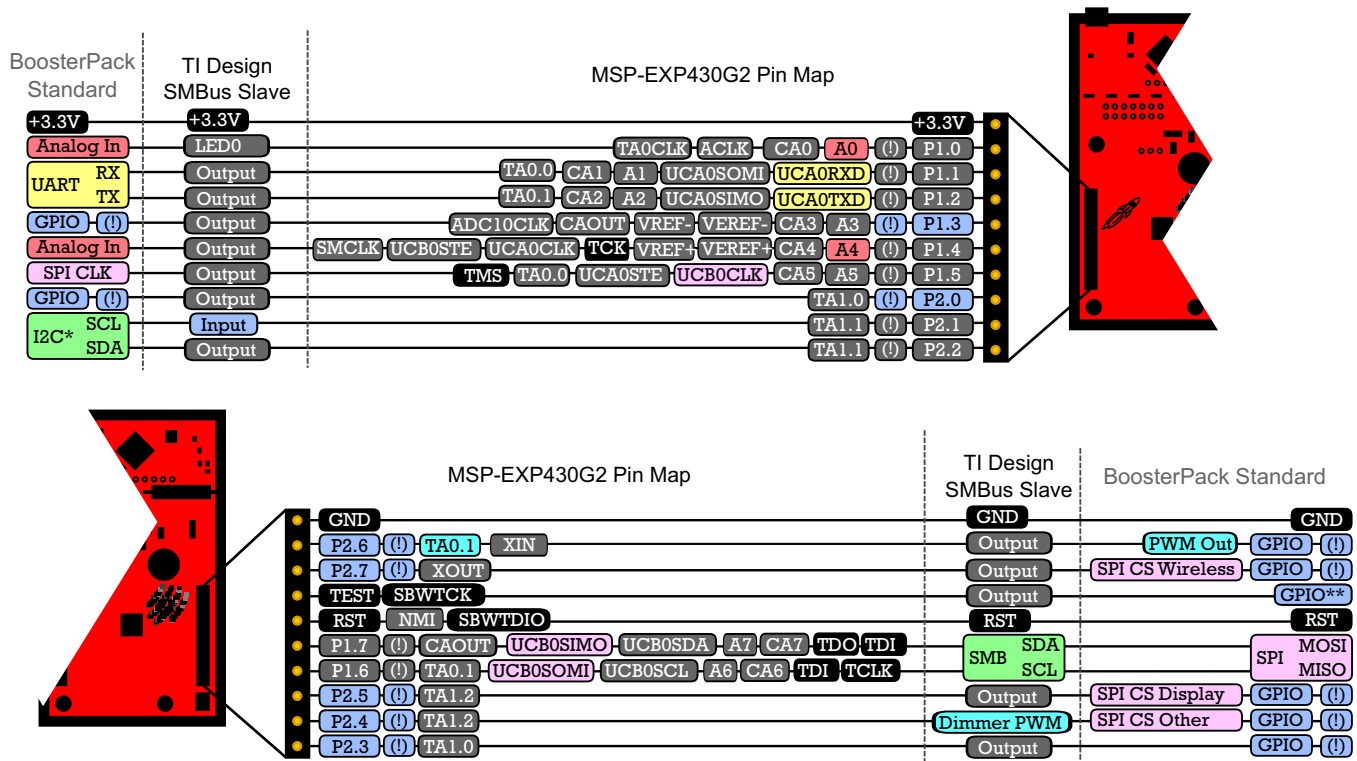


Figure 23. BoosterPack Pinout for MSP-EXP430G2 as Slave

- “Output” pins are actively driven low by the application. These pins are not used by the master or any of the slaves, and users must take care when changing the configuration of these pins to avoid a conflict with the other boards.
- “Input” pins are floating because they are not used by this slave, but they are used by the other boards.
- “DIMMER PWM” is the PWM output signal that connects to LED1 from TMP006 BoosterPack.
- “LED0” is connected directly to an LED in this Launchpad and can be used by the application.
- “SMB SDA” and “SMB SCL” signals are connected between the master and both slaves. Note that this configuration differs from the BoosterPack standard.

The board includes several jumpers that must be configured properly to download, debug, and execute the application. The jumpers and the default configuration are shown in Table 16:

Table 16. MSP-EXP430G2 Jumper Configuration as Slave

JUMPER	PROGRAMMING / DEBUGGING	EXECUTION—STACKED
J3 – TXD	OFF	OFF
J3 – RXD	OFF	OFF
J3 – RST	ON	OFF
J3 – TEST	ON	OFF
J3 – VCC	ON	OFF
J5 – P1.0	ON	ON
J5 – P1.6	OFF	OFF
J6	OFF	OFF

Note that the application can execute using the “Debugging” configuration, but problems may occur if the three boards are stacked on top of each other. The step-by-step procedure on how to debug and execute the application is shown in Section 4 and Section 5.

3.4 SMBus Slave—TMP006

This design uses a TMP006 as a second slave device to read the temperature of objects remotely. This device has five registers that the master device can read and write to control the functionality:

- The Sensor Voltage Result register is a 16-bit register in binary two's complement format containing the result of the last object temperature measurement. Data from this register is used in conjunction with data from the Local Temperature register to calculate the object temperature.
- The Temperature register is a 14-bit register that stores the result of the most recent conversion for the die temperature.
- The Configuration register determines the operational mode, conversion rate, and DRDY control; initiates a single conversion; performs a software reset; or puts the device into shutdown mode.
- The Manufacturer ID register always reads 0x5449
- The Device ID register always reads 0x0067

These registers are explained in more detail in the [TMP006 datasheet](#). The method to calculate the object temperature is described in the *TMP006 User's Guide* ([sbou107](#)).

3.4.1 DRDY Functionality

The TMP006 can use the DRDY (Data Ready) pin to inform the master when a new conversion of voltage and temperature is available. This event allows a master to use an interrupt-based approach instead of polling.

This implementation is independent from SMBus specification, but the master device used for this TI Design uses an input pin as an interrupt to detect falling edges on this pin. The master device also triggers a read transaction of both the Sensor Voltage Result register and the Temperature register. The pin used by the master is mentioned in [Section 3.2.3](#).

The DRDY is automatically de-asserted when the master reads the contents of these registers, allowing for automatic periodic conversions.

3.4.2 SMBus Communication

This address of the TMP006 can be modified depending on the settings of ADR1 and ADR0 pins, but the address used in this design is as shown in [Figure 24](#):

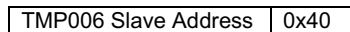


Figure 24. TMP006 Slave Address

A master device can write the contents of the TMP006 registers using the SMBus Write Word protocol, and it can read their contents using the Read Word protocol:

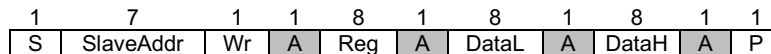


Figure 25. Write Word Protocol

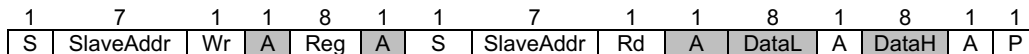
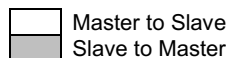


Figure 26. Read Word Protocol



S: Start / Repeated Start
SlaveAddr: Slave address
Wr: Write bit (0)
Rd: Read bit (1)
A: Acknowledge ('1' = NACK, '0' = ACK)
P: Stop
Reg: Register being addressed as described in [Section 3.3.2](#)
DataL: Low byte of Data
DataH: High byte of Data

Examples:

- Master enables TMP006 in Sensor and die continuous conversion mode, with a conversion rate of 1 conversion per second, and with DRDY enabled:

1	7	1	1	8	1	8	1	8	1	1
S	SlaveAddr	Wr	A	Reg	A	DataL*	A	DataH*	A	P
S	0x40	0	A	0x02	A	0x75	A	0x00	A	P

* Note that the TMP006 expects the MSB first.

- Master reads TMP006 Device ID:

1	7	1	1	8	1	1	7	1	1	8	1	8	1	1
S	SlaveAddr	Wr	A	Reg	A	S	SlaveAddr	Rd	A	DataL*	A	DataH*	A	P
S	0x40	0	A	0xFF	A	S	0x40	1	A	0x00	A	0x67	A	P

* Note that the TMP006 sends the MSB first.

3.4.3 Registers—Slave—TMP006

The following registers are implemented in TMP006 as shown in [Table 17](#):

Table 17. TMP006 Registers

REGISTER	ADDRESS	TYPE	RESET
SENSOR VOLTAGE RESULT	0x00	Read	0x0000
TEMPERATURE	0x01	Read	0x0000
CONFIGURATION	0x02	Read / Write	0x7400
MANUFACTURER ID	0xFE	Read	0x5449
DEVICE ID	0xFF	Read	0x0067

3.4.4 Hardware—Slave—TMP006

The application can be executed in practically any hardware, but the examples were developed and tested in the 430BOOST-TMP006 BoosterPack using TMP006EVM. The BoosterPack is explained in more detail in the *430BOOST-TMP006 BoosterPack™ User's Guide* ([slau440](#)), and the EVM is explained in the *TMP006EVM User Guide and Software Tutorial* ([sbou109](#)).

The Boosterpack can be stacked on top of the LaunchPads. The software takes special considerations to avoid conflicts when stacking boards as explained in the Getting Started guide included in [Section 4](#).

The 430BOOST-TMP006 can use two TMP006EVM simultaneously, but the software included in this TI Design only supports one board placed in Input #1.

The pinout of the BoosterPack when the TMP006EVM is placed in Input #1 is shown in [Figure 27](#).

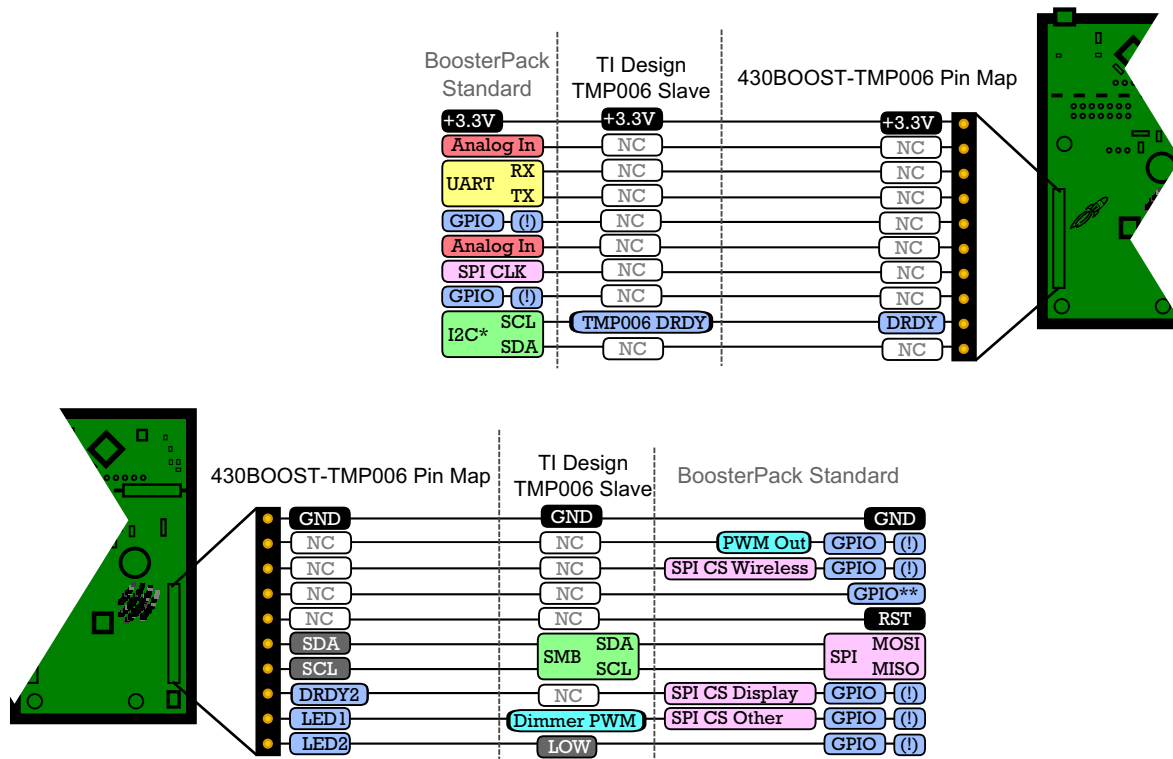


Figure 27. BoosterPack Pinout for 430BOOST-TMP006 with TMP006 in Input #1

- “NC” pins are not connected in this Boosterpack. These pins are driven low by the LaunchPads.
- “LOW” pins are driven low by the LaunchPads, but these pins are not used by application.
- “TMP006_DRDY” is connected to the master to indicate when a new conversion is available as explained in [Section 3.4.1](#).
- “DIMMER PWM” is a PWM signal driven by DIM430 Launchpad connected to LED1 in this BoosterPack.
- “SMB SDA” and “SMB SCL” signals are connected between the master and both slaves. Note that this configuration differs from the BoosterPack standard.

The step-by-step procedure on how to connect and use this BoosterPack is shown in [Section 4](#).

3.5 GUI

The graphical user interface (GUI) for this TI design is intended to provide an easy-to-use interface for users and developers to control the slave devices and observe the activity on SMBus. The project for the GUI was implemented in Java using Netbeans IDE as a development platform, using the following versions:

- JDK 7 Update 71
- Netbeans IDE 8.0.2

A pre-built executable .jar and full source code is provided with the software package. The GUI is divided in different panels as shown in [Figure 28](#):

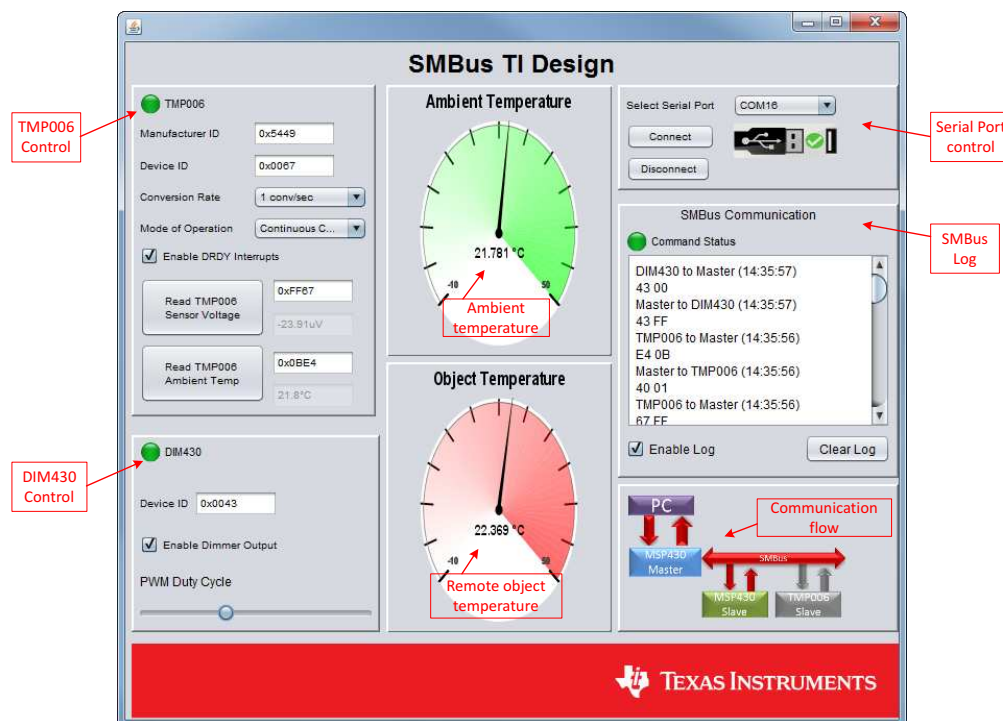


Figure 28. GUI Panels

The panels consist of the following:

- **TMP006 Control:** Shows when a TMP006 slave is connected and allows users to control this device. This panel contains the following fields:
 - **TMP006 LED:** shows if the TMP006 is connected and responding correctly to periodic packets
 - **Manufacturer ID:** Result of reading Manufacturer ID register in hexadecimal format
 - **Device ID:** Result of reading Device ID register in hexadecimal format
 - **Conversion Rate:** Configures the conversion rate to the 5 possible values: 4, 2, 1, 0.5, or 0.25 conversions/sec
 - **Mode of Operation:** Configures the mode of operation as Power-down mode, or Sensor and die continuous conversion
 - **Enable DRDY interrupts:** Enables/disables the DRDY pin in TMP006. More details about this functionality are discussed in [Section 3.4.1](#)
 - **Read TMP006 Sensor Voltage:** Performs manual read of Sensor Voltage Result register. The result is shown in hexadecimal and converted to volts.
 - **Read TMP006 Ambient Temp:** Performs manual read of Temperature register. The result is shown in hexadecimal and converted to Celsius degrees.
- **DIM430 Control:** Shows when a DIM430 slave is connected and allows users to control this device. This panel contains the following fields:
 - **DIM430 LED:** Shows if the DIM430 is connected and responding correctly to periodic packets
 - **Device ID:** Result of reading DEV_ID register in hexadecimal format
 - **Enable Dimmer Output:** Enables/disables the PWM output by setting/clearing CONFIG.EN0
 - **PWM Duty Cycle:** Controls the duty cycle from 0% to 100%
- **Ambient temperature:** graphical representation of the latest result of ambient temperature as read by TMP006
- **Remote object temperature:** graphical representation of the latest result of object temperature, calculated with TMP006's ambient temperature and sensor voltage

- Serial Port control: allows connection to the serial port. This panel contains the following fields:
 - Select Serial port: Shows ports available and allows users to select the corresponding port
 - Connect: Connects to the selected port
 - Disconnect: Disconnects from the selected port
- SMBus Communication: Shows the results of SMBus communications
- Communication Flow: Graphically represents the flow of communication.
 - Blocks are highlighted when communication occurs in the corresponding channel.
 - Communication from PC to master is performed via SerialComm (UART)
 - Communication between master and slaves is performed via SMBus.

The state diagram of the application is as shown in [Figure 29](#):

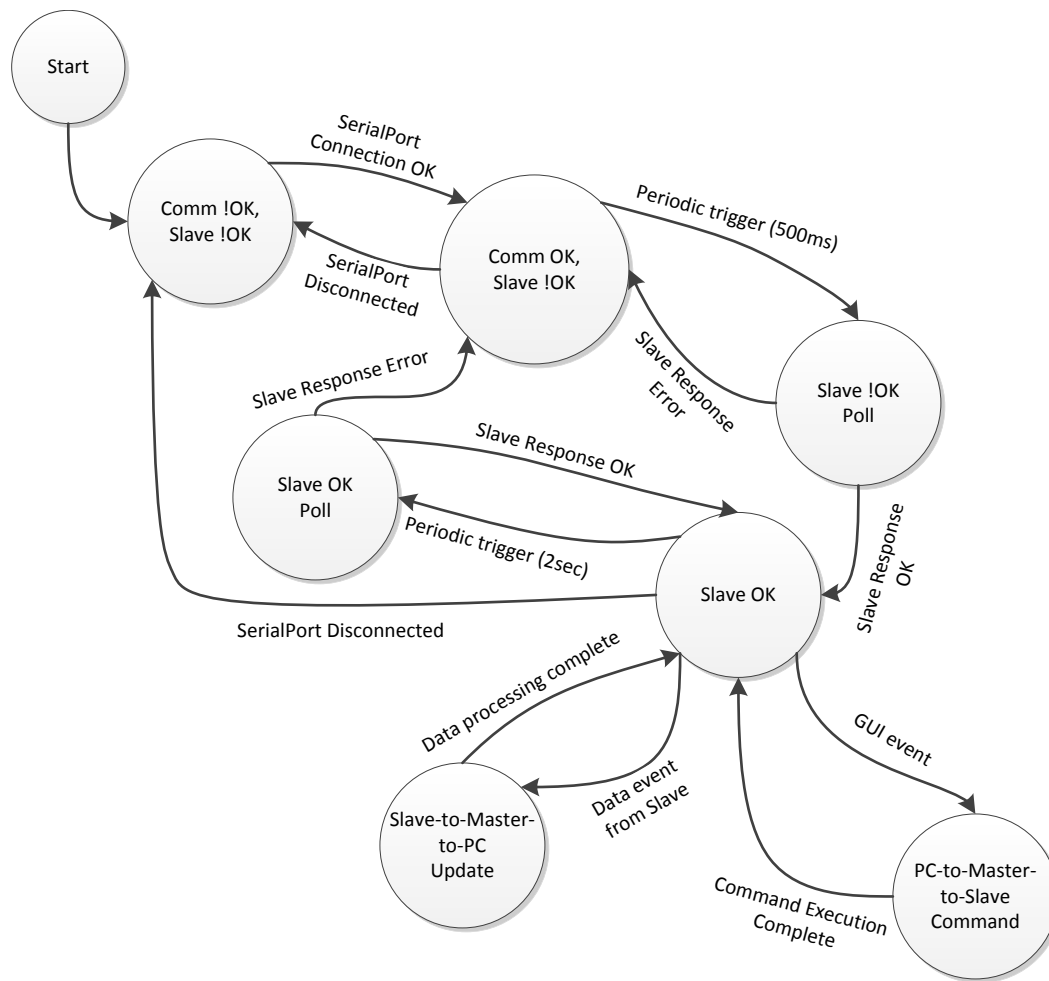


Figure 29. GUI State Diagram

The diagram applies for communication with both slave devices. During this communication, the GUI will poll both the TMP006 and DIM430 periodically to check if they are connected and enabled. If one of the slaves does not respond as expected, this slave will move to the “Communication Enabled, Slave Disconnected” state, but the other device will stay in “Slave Connected” state if it keeps responding as expected.

The states of the application are as follows:

- Comm !OK, Slave !OK: Initial state of the GUI since the SerialPort is disconnected. Disconnecting the SerialPort will return the application to this state.
- Comm OK, Slave !OK: Opening a valid SerialPort connection moves the application to this state indicating that Serial communication is valid, but communication with slave devices has not been established.
- Slave !OK Poll: To establish valid communication with the slave device, the PC will poll the device periodically every 500 ms, checking for a valid response.
 - For TMP006, the GUI reads the Device ID, Manufacturer ID, and Configuration, in that order. If successful, the GUI goes to “Slave OK” state and populates the corresponding fields in the TMP006 Control panel. If unsuccessful, the GUI goes back to “Comm OK, Slave !OK” state.
 - For DIM430, the GUI reads the Device ID and Configuration, in that order. If successful, the GUI goes to “Slave OK” state and populates the corresponding fields in the DIM430 Control panel. If unsuccessful, the GUI goes back to “Comm OK, Slave !OK” state.
- Slave OK: This state indicates that valid serial connections from the PC to the master device and from the master to slave device were established. The corresponding slave device is fully functional and the corresponding fields in the control panel are enabled.
- Slave OK Poll: The GUI checks for disconnection from the slave devices periodically (every 2 seconds).
 - For TMP006, the GUI reads the Device ID. If successful, the application stays in “Slave OK” state and populates the corresponding fields in the TMP006 Control panel. If unsuccessful, the GUI goes back to “Comm OK, Slave !OK” state.
 - For DIM430, the GUI reads the Device ID. If successful, the application stays in “Slave OK” state and populates the corresponding fields in the DIM430 Control panel. If unsuccessful, the GUI goes back to “Comm OK, Slave !OK” state.
- PC-to-Master-to-Slave Command: Any user-triggered event using the GUI sends a command to the master device, which in turn will send it to the corresponding slave.
 - For TMP006, the available GUI events are as follows:
 - Changing the conversion rate
 - Changing mode of operation
 - Enabling and disabling DRDY interrupts
 - Manually reading sensor voltage
 - Manually reading ambient temperature
 - For DIM430, the available GUI events are as follows:
 - Enabling/disabling dimmer output
 - Changing the PWM duty cycle
- Slave-to-Master-to-PC update: When data events from slave are detected as asynchronous events or in response from previous command, the host will inform the GUI that will process the data, updating the corresponding fields in the respective control panel.
 - For TMP006, the data events include the following:
 - Sensor voltage: Triggered by manual read events or DRDY interrupt events. This data event updates TMP006 Control and Remote Object Temperature panels.
 - Ambient temperature: Triggered by manual read events or DRDY interrupt events. This data event updates TMP006 Control and Ambient Temperature panels.
 - For DIM430, there are no data events.

3.5.1 Software—GUI

The GUI source code consists of the following files:

```

src
|- Resources          <- Images used by GUI
|- SMBusDemo         <- Source code
  |-- GUI.form        <- Used by IDE
  |-- GUI.java        <- Main panel including all GUI components
  |-- MainFrame.form  <- Used by IDE
  |-- MainFrame.java  <- Main Frame of application
  |-- Communicator.java <- Class controlling Serial communication
  |-- MyThermometer.java <- Class implementing gauge for graphical
  |                       representation of temperature
  |--Slave_Comm.java  <- Class handling communication with slave
  |                       devices
  |--DIM430_Comm.java <- Class extended from Slave_Comm to control
  |                       DIM430
  |--TMP006_Comm.java <- Class extended from Slave_Comm to control
  |                       TMP006
  |

```

Figure 30. Software Files for GUI

4 Getting Started Hardware

4.1 Programming the Boards

4.1.1 Master—MSP-EXP430FR5969

1. Disconnect board from USB
2. If stacked, unstack MSP-EXP430FR5969 from MSP-EXP430G2.
3. Set jumpers of MSP-EXP430FR5969 as shown in the “Programming / Debugging” configuration of [Table 12](#)
4. Connect USB to PC
5. Build program in CCS or IAR and download as explained in [Section 5](#).

4.1.2 DIM430 Slave—MSP-EXP430G2

1. Disconnect board from USB.
2. If stacked, unstack MSP-EXP430G2 from MSP-EXP430FR5969.
3. Set jumpers of MSP-EXP430G2 as shown in the “Programming / Debugging” configuration of [Table 16](#).
4. Connect USB to PC.
5. Build program in CCS or IAR and download as explained in [Section 5](#).

4.2 Executing the Application

1. Ensure that the master and DIM430 slave are programmed according to [Section 4.1](#).
2. Disconnect both boards from USB.
3. Set jumpers of MSP-EXP430FR5969 according to “Execution—Stacked” configuration of [Table 12](#).
4. Set jumpers of MSP-EXP430G2 according to “Execution—Stacked” configuration of [Table 16](#).
5. Connect TMP006EVM to Input#1 of 430BOOST-TMP006.
6. Stack MSP-EXP430FR5969 on top of MSP-EXP430G2 (note that MSP-EXP430G2 usually only has a male connector, while MSP-EXP430FR5969 has a connector with both female headers and male leads).
7. Stack 430BOOST-TMP006 on top of MSP-EXP430FR5969 (note that 430BOOST-TMP006 has a female connector, while MSP-EXP430FR5969 has a male/female connector).

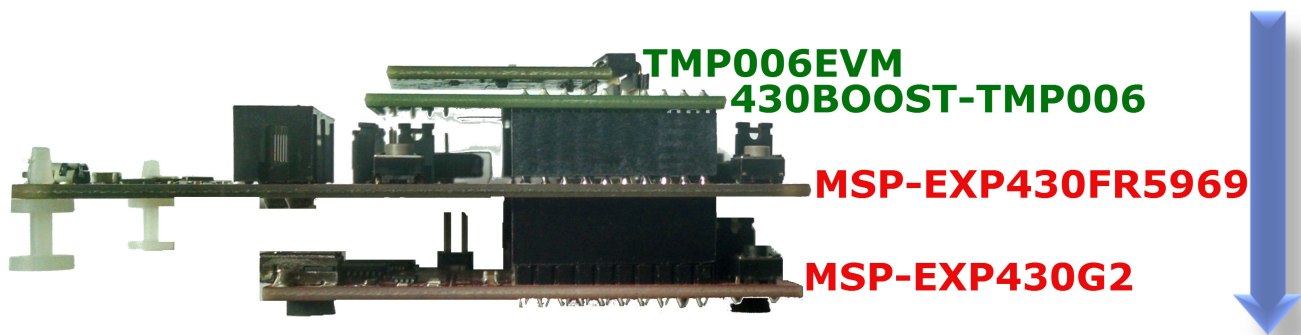


Figure 31. Boards Stacked for Demo

8. Connect MSP-EXP430FR5969 USB to PC (note that the other boards will get power from the master).
9. Check the COM port in the Device Manager (device will be installed as “MSP Application UART” under “Ports (COM & LPT)”).

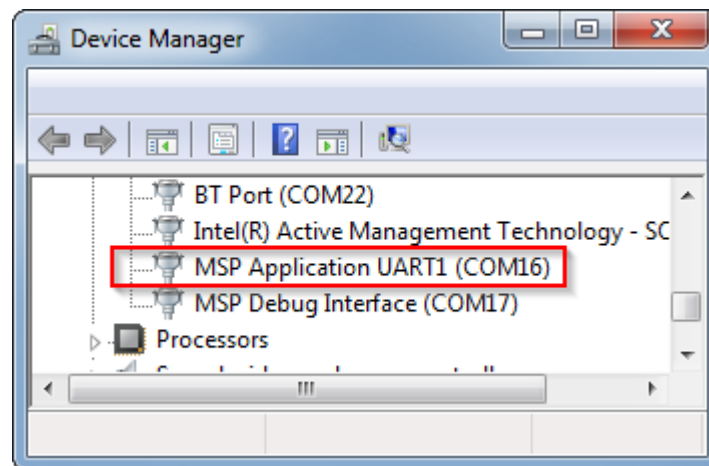


Figure 32. COM Port in Device Manager

10. Execute the application
 - .\PC_App\TID SMBus GUI\dist\TID_SMBus_GUI.jar
11. Select the corresponding COM port in the GUI.
12. Click the “Connect” button.
13. The GUI should connect with the master. The master will try to establish communication with slaves, and if successful, all the GUI components will be enabled.

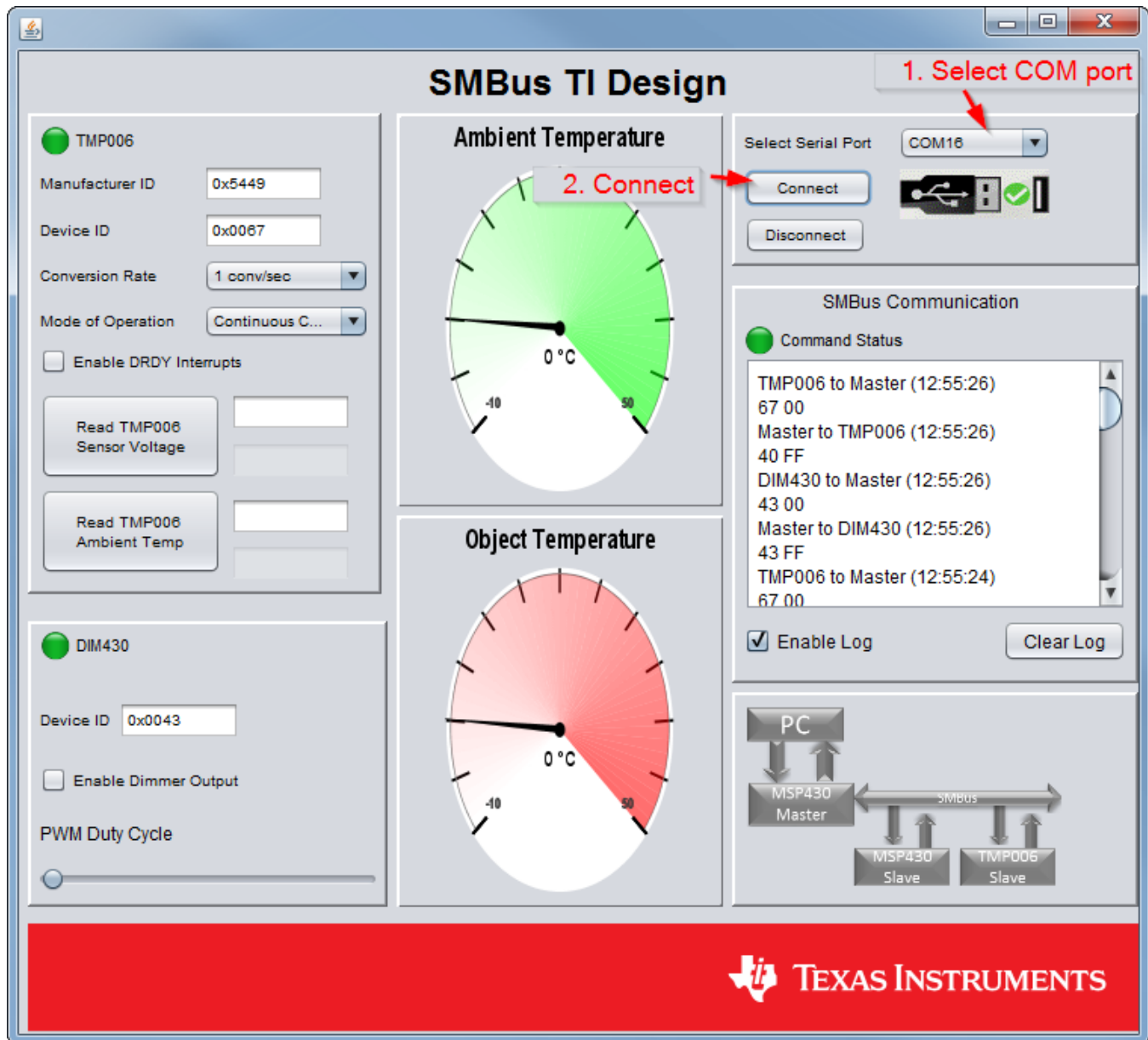


Figure 33. Steps to Connect with Master Using GUI

4.3 Testing USBKBD Configuration

1. Follow steps described in [Section 4.2](#) to execute the application.
2. Using the GUI, click the “Enable DRDY Interrupts” in TMP006 Control panel.
 - This will enable continuous conversions of the TMP006.
 - The master will obtain the latest conversion automatically and will send it to the PC.
 - The GUI will update all corresponding fields including the graphical gauges.
3. In the GUI, click the “Enable Dimmer Output” checkbox in the DIM430 Control panel.
4. Move slider “PWM Duty Cycle” slider to control the intensity of the LED.
 - Note that the LED is located in the 430BOOST-TMP006 board, but it is controlled by the DIM430 slave (MSP-EXP430G2).
5. Modify other GUI fields as needed.

6. Check the SMBus Log for details on SMBus packets.

5 Getting Started Firmware

The firmware included in this reference design has the following structure:

```

tidm-smbus-msp430
|--TID_App                <- Firmware for master and slave
|  |--smbuslib            <- Source code for SMBus Library
|  |  |--MSP430FR5xx_6xx  <- SMBus PHY for FR5xx/6xx eUSCI
|  |  |--MSP430G2xx3     <- SMBus PHY for G2xx3 USCI
|  |  |
|  |  |--driverlib       <- MSP430 Driverlib
|  |  |  |--MSP430FR5xx_6xx <- Driverlib for FR5xx/6xx
|  |  |  |--deprecated
|  |  |  |--inc
|  |  |
|  |  |--CCS             <- CCS project folder
|  |  |  |--SMBus_Master  <- CCS project for master
|  |  |  |--SMBus_Slave   <- CCS project for slave
|  |  |
|  |  |--IAR            <- IAR project folder
|  |  |  |--SMBus_Master  <- IAR project for master
|  |  |  |--SMBus_Slave   <- IAR project for slave
|  |  |
|  |  |--src            <- Source code for application
|  |  |  |--Master       <- Source code for master
|  |  |  |--Slave        <- Source code for slave

```

Figure 34. Firmware File Structure

The projects included in the software package have been built and tested in the following IDEs:

- Code Composer Studio 6.0.1
- IAR for MSP430 6.20.1

The procedure to build code for these IDEs is explained in the following sections.

5.1 Building Projects in IAR

1. Open the IAR workspace for the corresponding project:

```
tidm-smbus-msp430\TID_App\IAR\SMBus_TID_Workspace.eww
```

2. Select the SMBus_Master Project and the MSP430FR5969 target configuration.

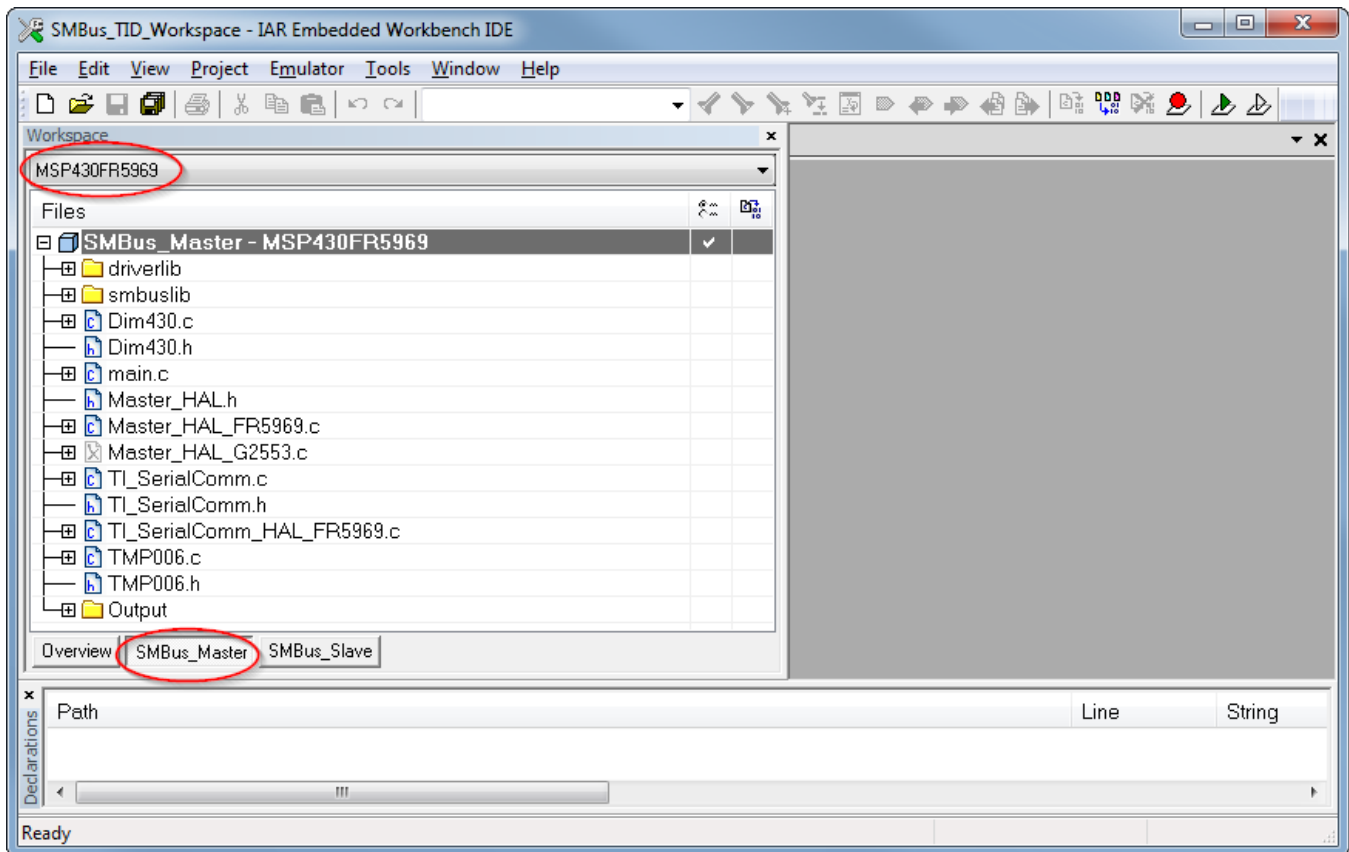




Figure 35. SMBus_Master Project in IAR

3. Connect MSP-EXP430FR5969 board as described in [Section 4.1.1](#).
4. Build project (F7, Menu → Project → rebuild All, or )
5. Download project to device (Ctrl + D, Menu → Project → Download and Debug, or )
6. Close Debugger.
7. Select the SMBus_Slave Project and the MSP430G2553 target configuration.
8. Connect MSP-EXP430G2 as described in [Section 4.1.2](#).
9. Repeat steps 4 through 6 to flash the slave device 1.
10. Follow procedure in [Section 4.2](#) to execute the example.

5.2 Building Projects in CCS

1. Import the projects in CCS (Menu → Project → Import CCS Project).

tidm-smbus-msp430\TID_App\CCS

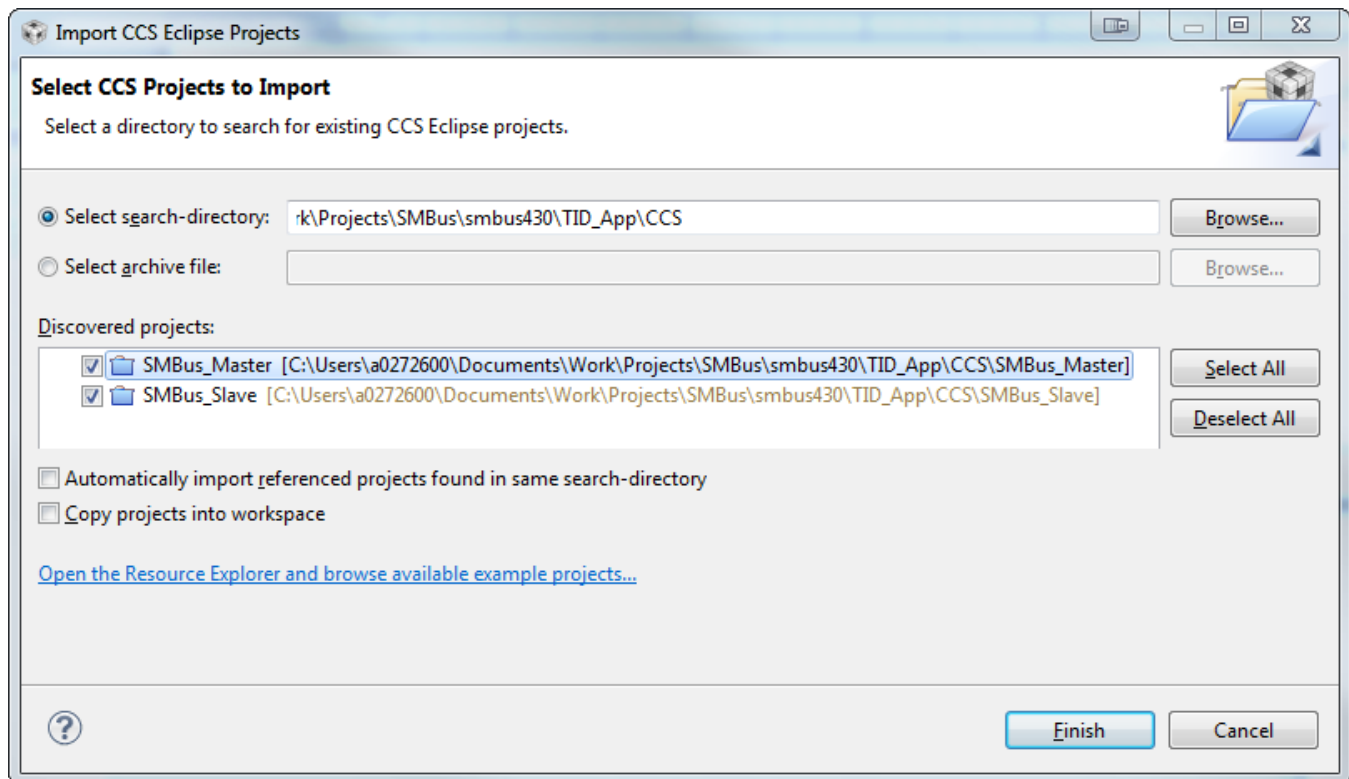


Figure 36. Importing SMBus Projects in CCS

2. Select the SMBus_Master Project and the MSP430FR5969 target configuration.

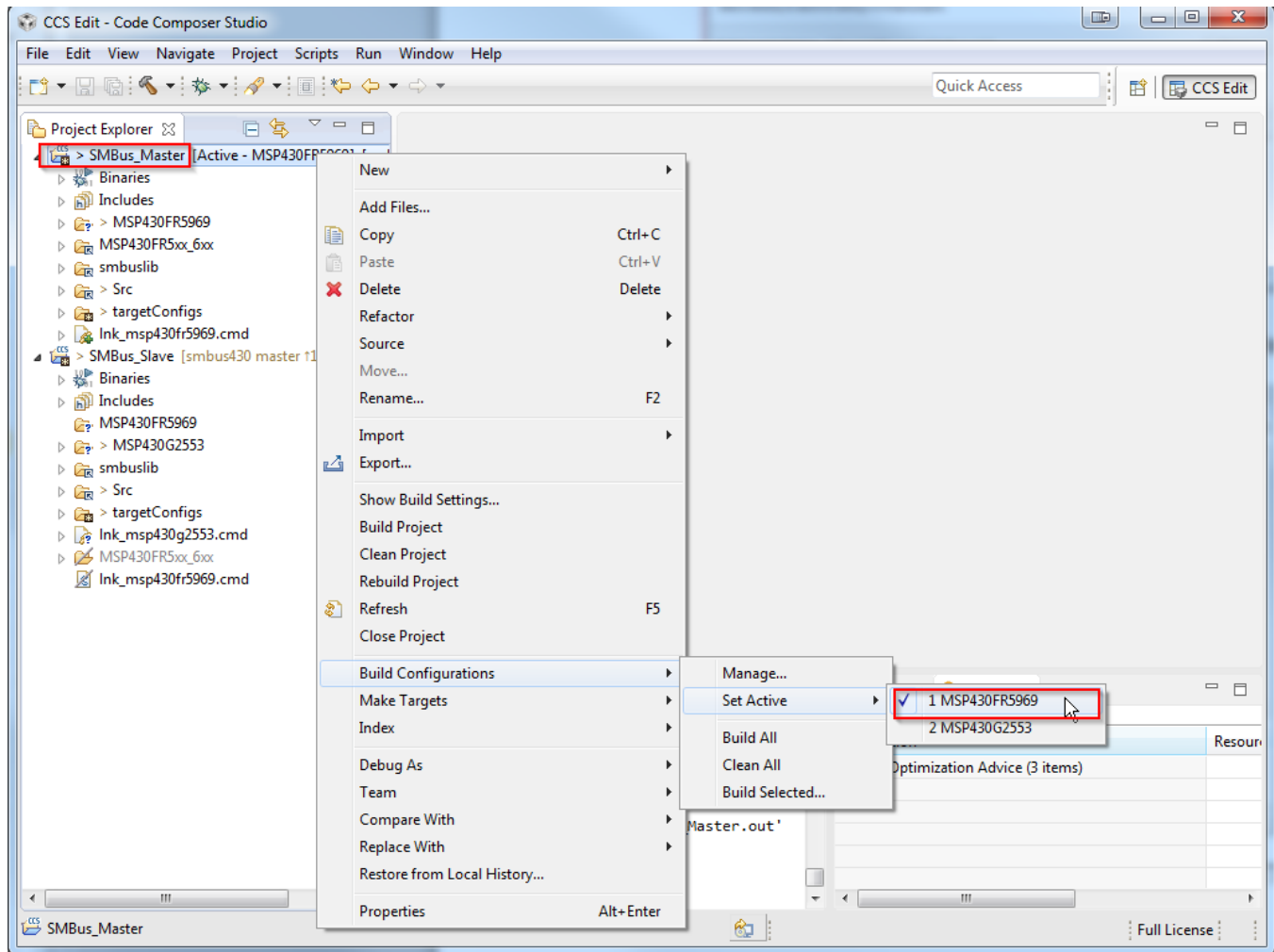




Figure 37. SMBus_Master Project in CCS

3. Connect MSP-EXP430FR5969 board as described in [Section 4.1.1](#).
4. Build project (Ctrl+B, Menu → Project → Build All, or ).
5. Download project to device (F11, Menu → Run → Debug, or ).
6. Close Debugger.
7. Select the SMBus_Slave Project and the MSP430G2553 target configuration.
8. Connect MSP-EXP430G2 as described in [Section 4.1.2](#).
9. Repeat steps 4–6 to flash the slave device 1.
10. Follow procedure in [Section 4.2](#) to execute the example.

6 Changing Functionality of MSP430™ Devices

The default configuration of this TI Design uses the MSP430FR5969 as master and MSP430G2553 as DIM430 slave, but the application is modular enough to allow swapping the functionality of these devices.

6.1 Using MSP430G2553 as Master

The resources used from MSP430G2553 when configured as master are as shown in [Table 18](#):

Table 18. Master Hardware Resources—MSP430G2553

FUNCTION	PERIPHERAL	GPIO	CONNECTION TO OTHER BOARDS
SMBus	USCI_B0	SDA: P1.7/UCB0SDA SCL: P1.6/UCB0SCL	SDA of both slaves SCL of both slaves
UART	USCI_A0	TXD: P1.2/UCA0TXD RXD: P1.1/UCA0RXD	-
LEDs	-	LED0: P1.0 LED1: X	-
TMP006 DRDY	-	P2.1	DRDY of TMP006 slave

Figure 38 shows the pinout of the MSP-EXP430G2 board when used as master:

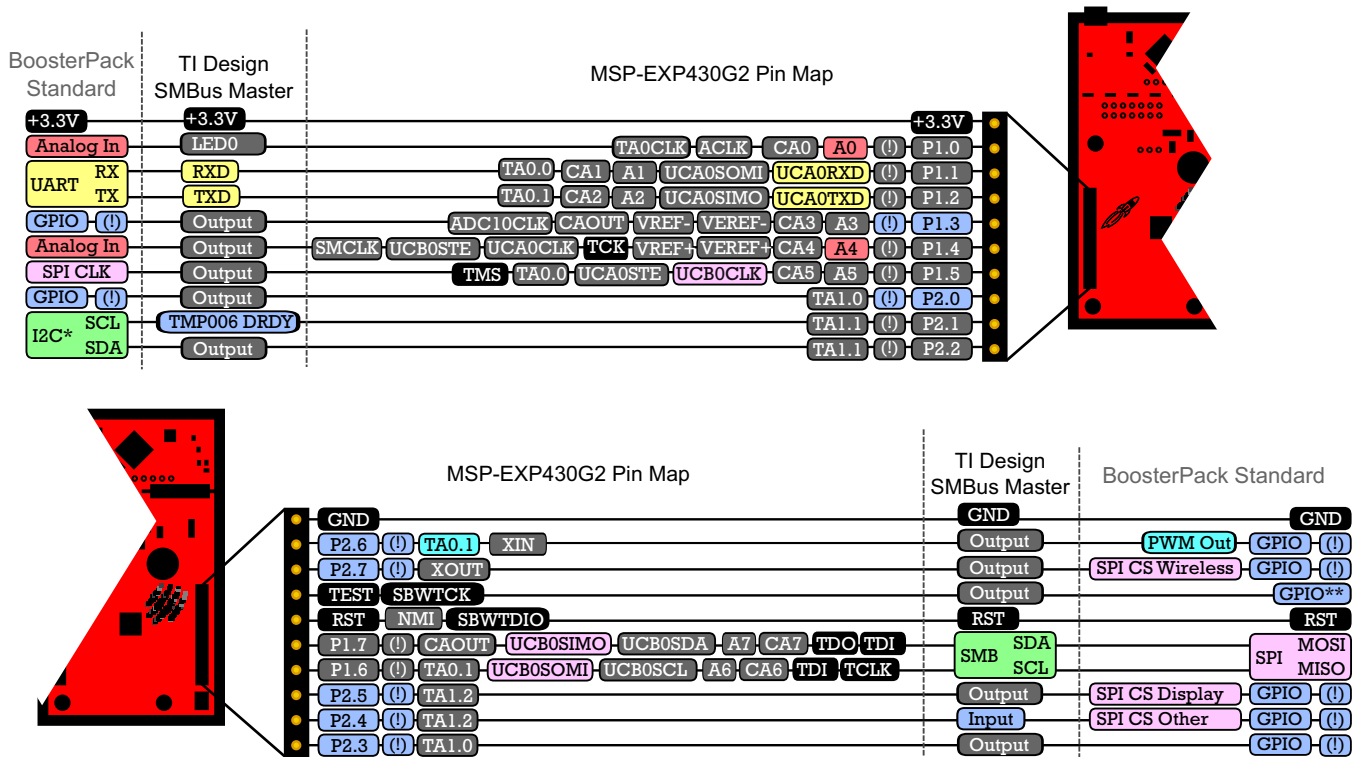


Figure 38. BoosterPack Pinout for MSP-EXP430G2 as Master

Table 19 shows the jumper configuration:

Table 19. MSP-EXP430G2 Jumper Configuration as Master

JUMPER	PROGRAMMING / DEBUGGING	EXECUTION—STACKED
J3 – TXD	ON – HW UART	ON – HW UART
J3 – RXD	ON – HW UART	ON – HW UART
J3 – RST	ON	OFF
J3 – TEST	ON	OFF
J3 – VCC	ON	ON
J5 – P1.0	ON	ON
J5 – P1.6	OFF	OFF
J6	OFF	OFF

6.2 Using MSP430FR5969 as Slave

Table 20 shows the resources used from MSP430FR5969 when configured as slave:

Table 20. DIM430 Slave Hardware Resources—MSP430FR5969

FUNCTION	PERIPHERAL	GPIO	CONNECTION TO OTHER BOARDS
SMBus	eUSCI_B0	SDA: P1.6/UCB0SDA SCL: P1.7/UCB0SCL	SDA of master and TMP006 SCL of master and TMP006
Dimmer PWM	TB0 TB0.1	PWM: P1.4/TB0.1	LED1 of TMP006 BoosterPack

Figure 39 shows the pinout of the MSP-EXP430FR5969 board when used as master:

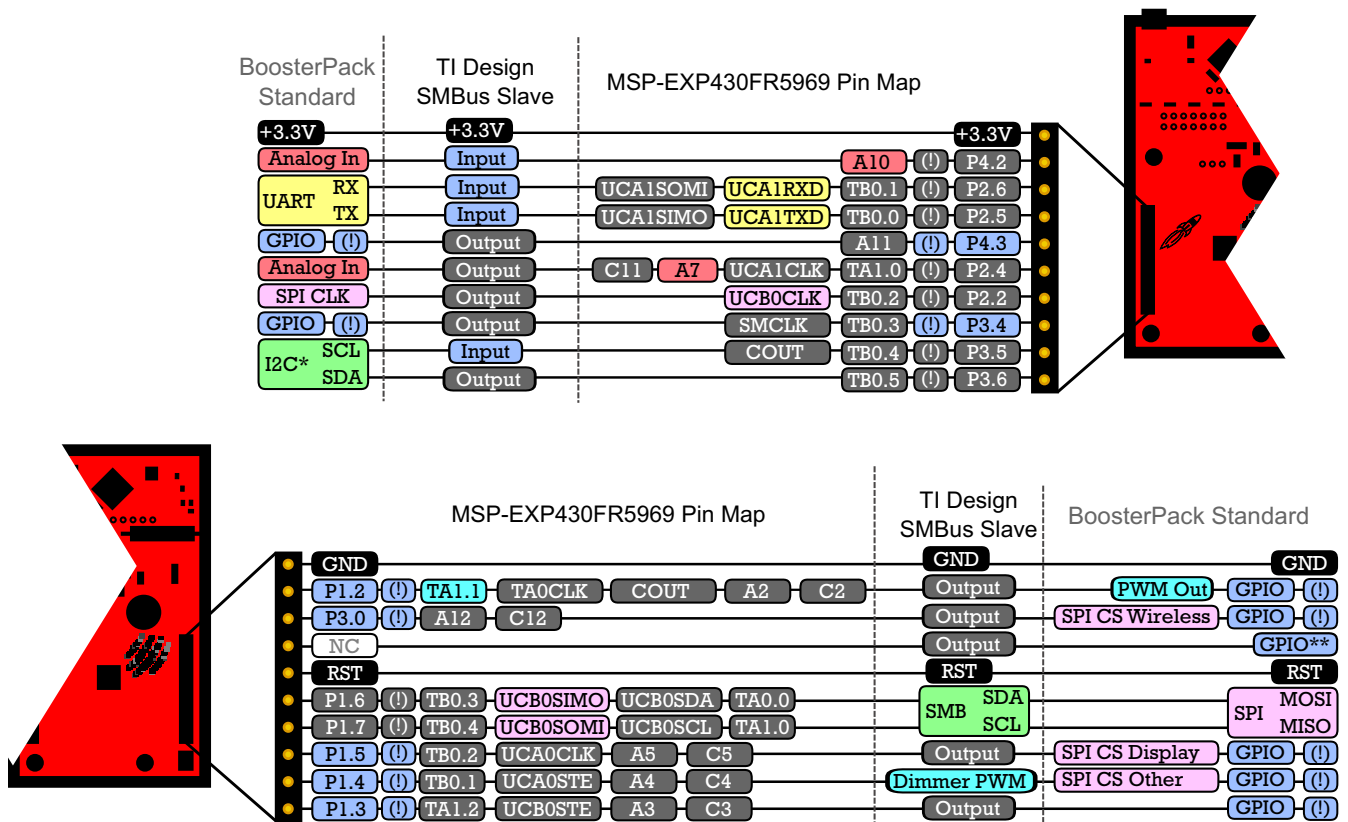


Figure 39. BoosterPack Pinout for MSP-EXP430FR5969 as Slave

Table 21 shows the jumper configuration:

Table 21. MSP-EXP430G2 Jumper Configuration as Master

JUMPER	PROGRAMMING / DEBUGGING	EXECUTION – STACKED
J1	OFF	OFF
J2	Bypass	Bypass
J6	ON	ON
J9	ON	ON
J10	Debugger	External
J11	OFF	OFF
J12	OFF	OFF
J13 – GND	ON	ON
J13 – 5V	OFF	OFF
J13 – V+	ON	OFF
J13 – RTS	OFF	OFF
J13 – CTS	OFF	OFF
J13 – RXD	OFF	OFF
J13 – TXD	OFF	OFF
J13 – RST	ON	OFF
J13 – TST	ON	OFF

6.3 Building and Testing the Project

- Use the same procedure explained in [Section 5](#) to build the project with the following changes:
 - Use MSP430G2553 target configuration for the SMBus_Master Project and program the MSP-EXP430G2 LaunchPad.
 - Use MSP430FR5969 target configuration for the SMBus_Slave Project and program the MSP-EXP430FR5969 LaunchPad.
- Program the boards following the procedure mentioned in [Section 4.1.1](#) and [Section 4.1.2](#), but using the corresponding LaunchPad.
- Use the procedure mentioned in [Section 4.2](#) and [Section 4.3](#) to execute and test the application with the following change:
 - In step 8 of [Section 4.2](#), connect MSP-EXP430G2 USB to PC (instead of MSP-EXP430FR5969).

7 Test Data

7.1 Test Setup

The board should be connected following the guidelines described in [Section 4](#).

To measure power consumption of MSP-EXP430FR5969:

- Disconnect J9 and connect an ammeter in series

To measure power consumption of MSP-EXP430G2:

- Isolate the VCC for this board from the rest and connect an ammeter in series.
- This procedure will probably require unstacking the board and using jumper wires to connect the required signals.

7.2 Power Consumption

7.2.1 Master Application

Figure 40 shows the expected power profile for the master application:

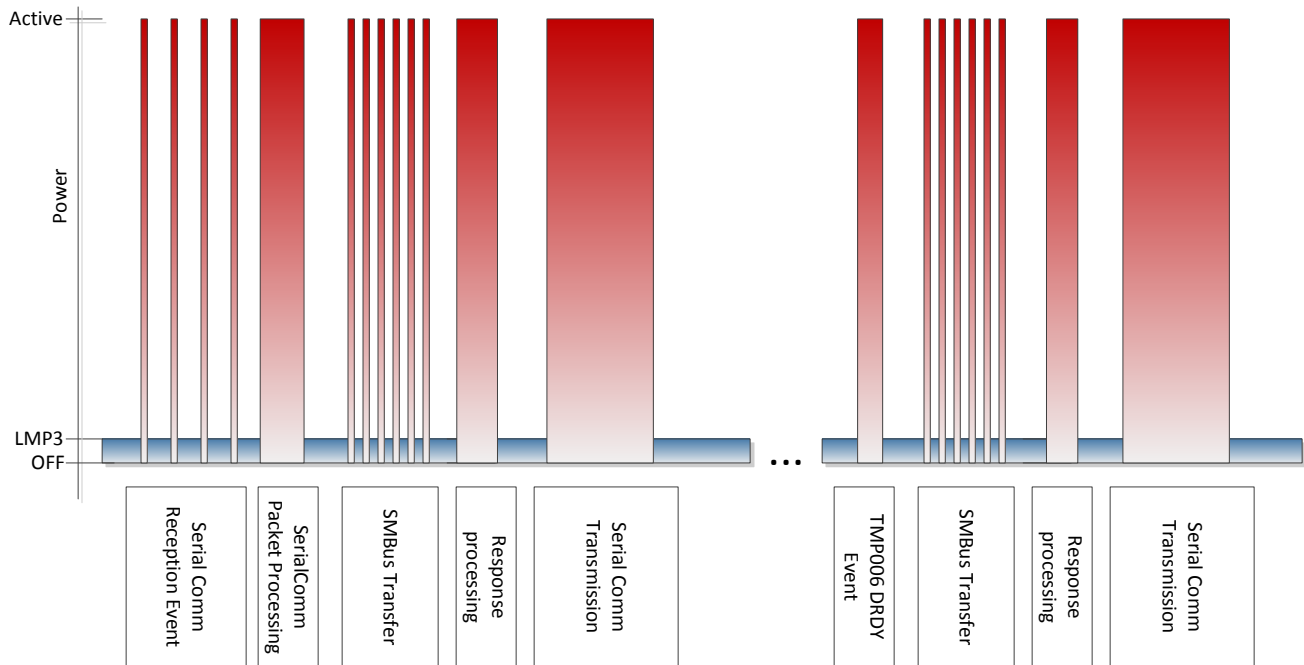


Figure 40. Expected Power Profile for Master

The following profile in Figure 41 was observed for a serial packet reception from the PC when running the MSP430FR5969 as a master at 8 Mhz:

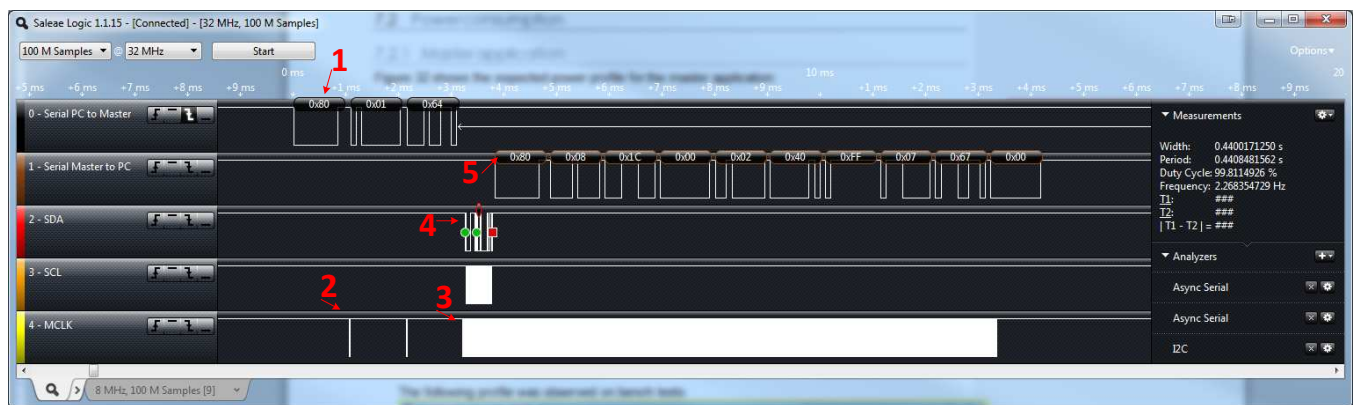


Figure 41. Power Profile for SerialComm Reception—Master Using MSP430FR5969

The labels from Figure 41 show the different steps of the process:

1. Serial packet is received from the PC.
2. MCU wakes-up periodically to get each byte.
3. When the packet is complete, the devices processes it.
4. Device sends a command to the corresponding slave via SMBus.
5. After the SMBus packet is sent, the response is sent to the PC.

Similarly, a DRDY event generates the following profile seen in [Figure 42](#) when using the same MSP430FR5969 at 8 Mhz:

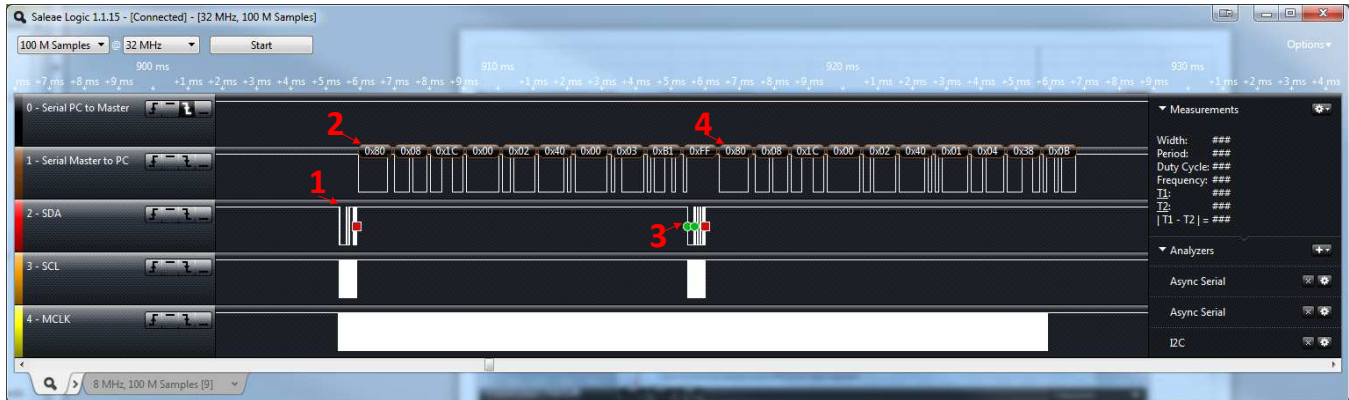


Figure 42. Power Profile for DRDY Event—Master Using MSP430FR5969

The labels from [Figure 42](#) show the different steps of the process:

1. Device wakes up and sends command via SMBus to TMP006 (sensor voltage)
2. Result from SMBus transfer is sent to PC
3. A second packet is sent to TMP006 (ambient temperature)
4. Result from SMBus transfer is sent to PC

[Table 22](#) shows the power consumption measured on MSP-EXP430FR5969:

Table 22. Power Consumption Measurements for MSP430FR5969 as Master

DEVICE	MODE	CURRENT	POWER
MSP430FR5969	Active VCC = 3.3 V MCLK = SMCLK = DCO = 8 Mhz ACLK = VLO	1.032 mA	3.4 mW
	LPM3 VCC = 3.3 V ACLK = VLO, SVS Enabled	0.8 μ A	2.64 μ W

The average power consumption will depend on the bus load, both from PC to host, and from the slaves through SMBus. As an example, with two periodic serial commands every two seconds and periodic TMP006 DRDY events every 0.5 seconds, the average power consumption was measured as follows:

Table 23. Average Power Consumption for MSP430FR5969 as Master

DEVICE	MODE	CURRENT (AVG)	POWER (AVG)
MSP430FR5969	Periodic updates 2 TMP006 DRDY events x sec 1 TMP0006 DEV_ID read x 2 secs 1 DIM430 DEV_ID read x 2 secs	60.4 μ A	199.32 μ A

7.2.2 Slave Application

Figure 43 shows the expected power profile for the slave application:

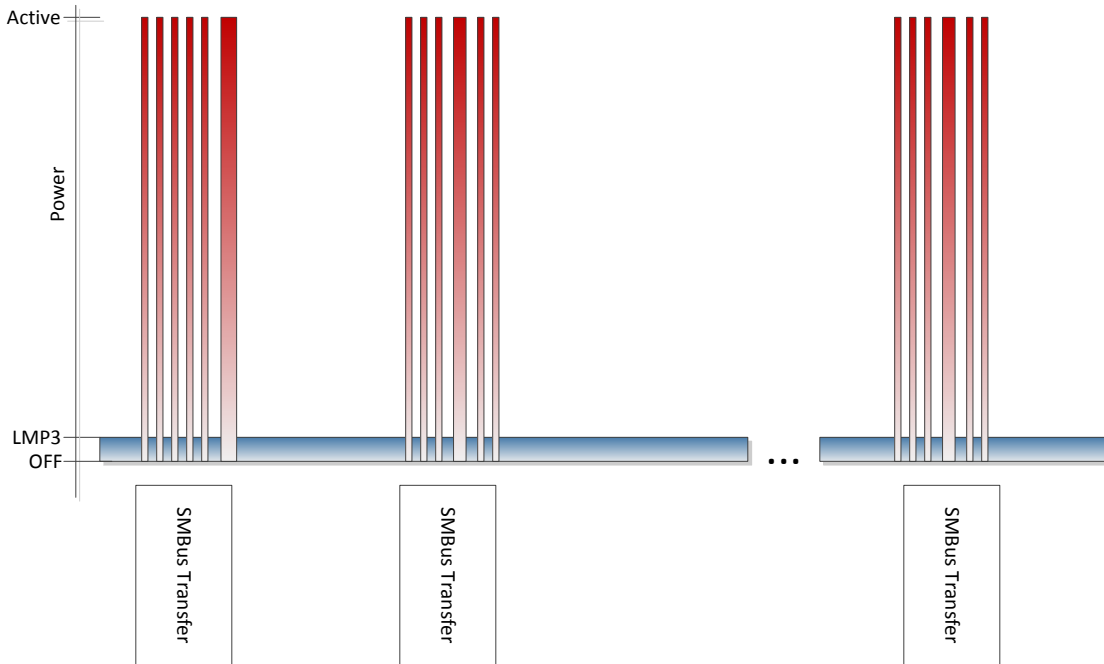


Figure 43. Expected Power Profile for Slave

The profile in Figure 44 was observed for a Read Word packet using MSP430G2553 as a slave at 8 Mhz:



Figure 44. Power Profile for ReadWord Packet—Slave Using MSP430G2553

The labels from Figure 44 show the different steps of the process:

1. Master sends a packet via SMBus
2. Slave wakes periodically to get each byte
3. During Repeated Start, the slave processes the packet and prepares response
4. Slave wakes to send each response byte

Similarly, a Write Word packet shows the profile in Figure 45:



Figure 45. Power Profile for WriteWord Packet—Slave Using MSP430G2553

The steps shown in [Figure 45](#) are:

1. Master sends a packet via SMBus
2. Slave wakes periodically to get each byte
3. Slave processes end of packet

[Table 24](#) shows the power consumption measured on MSP-EXP430G2:

Table 24. Power Consumption Measurements for MSP430G2553 as Slave

DEVICE	MODE	CURRENT	POWER
MSP430G2553	Active VCC = 3.3 V MCLK = SMCLK = DCO = 8 Mhz ACLK = VLO	2.45 mA	8.08 mW
	LPM3 VCC = 3.3 V ACLK = VLO	0.6 μ A	1.98 μ W

The average power consumption depends on the SMBus activity. However, with a periodic Write Word packet every 500 ms, and a periodic Read Word packet every two seconds, the average power consumption measured is as shown in [Table 25](#):

Table 25. Average Power Consumption for MSP430G2553 as Slave

DEVICE	MODE	CURRENT (AVG)	POWER (AVG)
MSP430G2553	Periodic updates 2 WriteWord packets \times sec 1 ReadWord packet \times 2 secs	1.8 μ A	5.94 μ W

7.3 Memory Footprint

The memory footprint in [Table 26](#) was obtained using IAR for MSP430 6.20.1 using optimization level “High-Balanced”:

Table 26. Memory Footprint

	MSP430FR5969 MASTER	MSP430G2553 SLAVE
CODE	4,786B	1,796B
SMBusLib	1,528B	1,116B
driverlib	902B	-
App	2,200B	570B
others	156B	110B
CONSTANTS	284B	256B
SMB CRC lookup_table	256B	256B
Other	28B	-
DATA	357B	178B
Stack	160B	80B
Heap	80B	-
App	117B	98B

8 Design Files

8.1 Schematics

To download the Schematics for each board, see the design files at <http://www.ti.com/tool/TIDM-SMBUS>.

8.2 Bill of Materials

To download the bill of materials (BOM), see the design files at <http://www.ti.com/tool/TIDM-SMBUS>.

8.3 Layer Plots

To download the layer plots, see the design files at <http://www.ti.com/tool/TIDM-SMBUS>.

8.4 Altium Project

To download the Altium project files, see the design files at <http://www.ti.com/tool/TIDM-SMBUS>.

8.5 Layout Guidelines

To download the layout guidelines, see the design files at <http://www.ti.com/tool/TIDM-SMBUS>.

8.6 Gerber Files

To download the Gerber files, see the design files at <http://www.ti.com/tool/TIDM-SMBUS>.

8.7 Assembly Drawings

To download the assembly drawings, see the design files at <http://www.ti.com/tool/TIDM-SMBUS>.

8.8 Software Files

To download the software files, see the design files at <http://www.ti.com/tool/TIDM-SMBUS>.

9 References

1. SMBus 2.0 specification (<http://www.smbus.org/>)
2. *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide* ([slau367](#))
3. *MSP430FR59xx datasheet* ([slas704](#))
4. *MSP-EXP430FR5969 LaunchPad Development Kit User's Guide* ([slau535](#))
5. *MSP430x2xx Family User's Guide* ([slau144](#))
6. *MSP430G2xx3 datasheet* ([slas735](#))
7. *MSP-EXP430G2 LaunchPad Evaluation Kit User's Guide* ([slau318](#))
8. *TMP006 User's Guide* ([sbou107](#))
9. *TMP006 datasheet* ([sbos518](#))
10. *430BOOST-TMP006 BoosterPack User's Guide* ([slau440](#))
11. *TMP006EVM User's Guide* ([sbou109](#))
12. *MSP430 DriverLib for MSP430FR5xx 6xx Devices User's Guide* (<http://www.ti.com/tool/msp430driverlib>)
13. *MSP430 Smbus Library User's Guide* (<http://www.ti.com/tool/TIDM-SMBUS>)

10 About the Author

LUIS REYNOSO is an Applications Engineer at Texas Instruments. He has taken multiple customer-facing roles in the embedded industry, and during this time he has published several Applications Notes and papers for microcontrollers. He joined the MSP430™ Applications team in 2010.

IMPORTANT NOTICE FOR TI REFERENCE DESIGNS

Texas Instruments Incorporated ("TI") reference designs are solely intended to assist designers ("Buyers") who are developing systems that incorporate TI semiconductor products (also referred to herein as "components"). Buyer understands and agrees that Buyer remains responsible for using its independent analysis, evaluation and judgment in designing Buyer's systems and products.

TI reference designs have been created using standard laboratory conditions and engineering practices. **TI has not conducted any testing other than that specifically described in the published documentation for a particular reference design.** TI may make corrections, enhancements, improvements and other changes to its reference designs.

Buyers are authorized to use TI reference designs with the TI component(s) identified in each particular reference design and to modify the reference design in the development of their end products. HOWEVER, NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY THIRD PARTY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT, IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI REFERENCE DESIGNS ARE PROVIDED "AS IS". TI MAKES NO WARRANTIES OR REPRESENTATIONS WITH REGARD TO THE REFERENCE DESIGNS OR USE OF THE REFERENCE DESIGNS, EXPRESS, IMPLIED OR STATUTORY, INCLUDING ACCURACY OR COMPLETENESS. TI DISCLAIMS ANY WARRANTY OF TITLE AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, QUIET ENJOYMENT, QUIET POSSESSION, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO TI REFERENCE DESIGNS OR USE THEREOF. TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY BUYERS AGAINST ANY THIRD PARTY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON A COMBINATION OF COMPONENTS PROVIDED IN A TI REFERENCE DESIGN. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, SPECIAL, INCIDENTAL, CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ON ANY THEORY OF LIABILITY AND WHETHER OR NOT TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT OF TI REFERENCE DESIGNS OR BUYER'S USE OF TI REFERENCE DESIGNS.

TI reserves the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI's terms and conditions of sale of semiconductor products. Testing and other quality control techniques for TI components are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers' products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers' products and applications, Buyers should provide adequate design and operating safeguards.

Reproduction of significant portions of TI information in TI data books, data sheets or reference designs is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards that anticipate dangerous failures, monitor failures and their consequences, lessen the likelihood of dangerous failures and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in Buyer's safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI's goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed an agreement specifically governing such use.

Only those TI components that TI has specifically designated as military grade or "enhanced plastic" are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components that have **not** been so designated is solely at Buyer's risk, and Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.