MSP PMBus Library

Users Guide

DOCNUM-1.00.00.00

Contents

Chapter 1

Copyright

Texas Instruments
13532 N. Central Expressway
Dallas, TX 75243
www.ti.com

TEXAS INSTRUMENTS

Chapter 2

Introduction

The Power Management Bus (PMBus) is an industry-wide standard that can be viewed as a step toward unifying communication standards for power conversion and digital power-management devices. It was developed by the PMBus Implementers Forum (PMBus-IF). The standard uses the widely accepted Inter-Integrated Circuit (I2C) communication protocol for the hardware interface. A number of additional features serve to enhance the basic I2C communication protocol. The PMBus standard is also considered to be an extension to the System Management Bus (SMBus) protocol that was popularized by the SBS Implementers Forum for battery systems.

This document serves as a guide to Texas Instruments implementation of the PMBus master protocol on MSP430™ microcontroller.

2.1 Associated Files

The following files and folders are included in the package folder.

```
<install>
  |--release_notes.html   <- Release notes
  |--license.txt          <- Licensing text
  |--manifest.html        <- Library manifest
  |--docs
  |     |-- API_Guide      <- PMBus source code function documentation
  |     |-- Users_Guide    <- PMBus library users guide
  |--examples             <- Example projects
  |--smbuslib             <- SMBus library source code used by the PMBus library
  |--src                  <- PMBus library source code
```

Chapter 3

PMBus Protocol

The PMBus standard was developed by a group of companies that envisioned the need for a power-management protocol that has a well-defined physical communication layer and includes support for implementing commonly used power-management commands. I2C was considered the preferred standard for implementing the physical communication layer because it provided a simple means to relay data representing commands, control, and information over a serial bus.[4] In addition to the data and clock lines provided by I2C, the PMBus protocol implements the optional alert response (for host notification in the presence of a fault) and control line (for turning the slave device on/off). The PMBus command layer provides a set of 256 commands that the host microcontroller can use to instruct slave PMBus devices.

The PMBus protocol also implements a timeout feature (present in SMBus) that prevents slower slave devices from holding the clock line for longer than the specified timeout interval. This limits the minimum allowable frequency of I2C transactions to 10 kHz, similar to SMBus but absent in I2C. For further differences in timing and DC specifications between the protocols, see System Management Bus (SMBus) Specification V2.0.[2] A new feature that increases reliability and robustness of this protocol is Packet Error Checking (PEC). PEC was first introduced in version 1.1 of the SMBus protocol and is implemented by using a Cyclic Redundancy Check-8 (CRC-8) algorithm to validate the integrity of a transaction.

It should be noted that PMBus can function as a 2-wire protocol. The data and clock lines are retained from the I2C protocol for transmitting/receiving data and for controlling the clocking of data respectively. The alert response and control signals are optional implementations/enhancements for host microcontrollers that are willing to trade two GPIO pins for the ability to service a system fault and to turn the slave device on or off using software.

The figure below shows the architecture of the PMBus host microcontroller's communication model. The application layer is at the user-interaction level and also can be viewed as a data translation/interpretation layer. It performs application services and issues requests to the PMBus layer. The PMBus layer accepts commands from the application layer. It processes these commands and passes them to the SMBus layer which implements the SMBus communication state machine and instructs the hardware access (I2C) layer on how to execute the commands. The I2C layer, in conjunction with the alert response and control line, provides the physical interface needed for direct communication with the slave device.

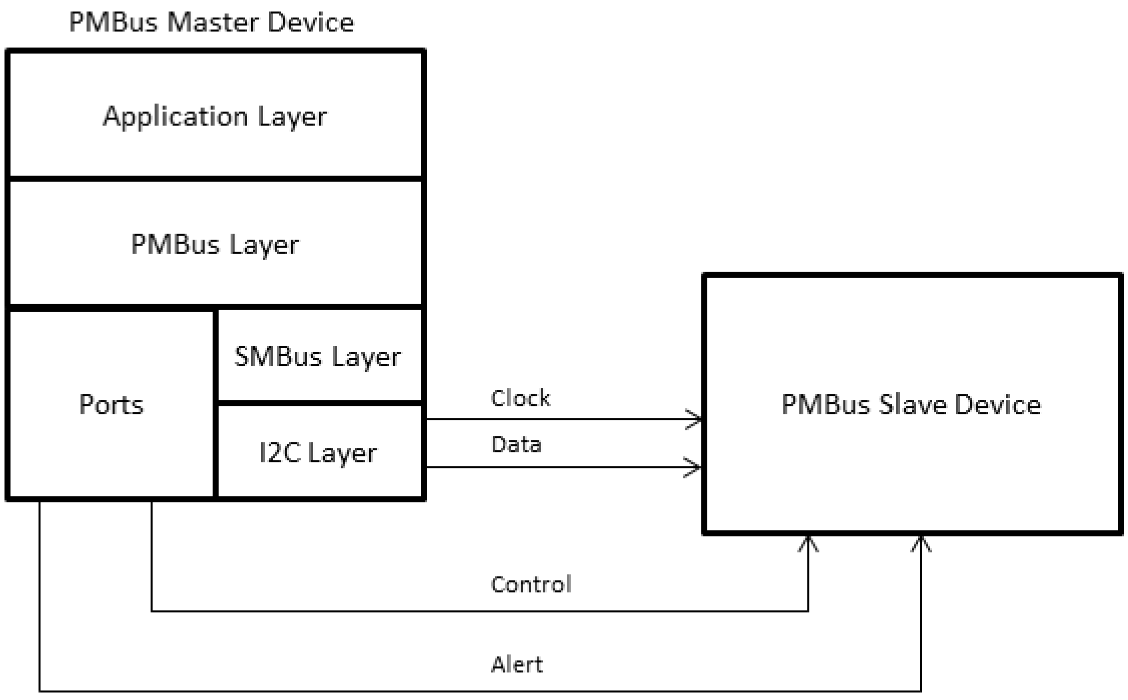


Figure 3.1: PMBus Communication Model

3.1 PMBus Layer

The following significant features uniquely identify the PMBus protocol and establish its position as a valuable part of any power-management system.

3.1.1 Alert Signal (SMBALERT)

The SMBALERT line is used by slave devices to notify the master in the presence of a system fault. This line is a wired-AND signal and is tied high at reset similar to I2C clock and data lines. The slave device can pull it low at any time to notify the host microcontroller that it needs to communicate. When the master PMBus device receives an alert from the slave, the master can be configured to read the status registers of the slave device to determine the type and location of the fault. Once the fault has been read and recorded, the slave device can be notified and its status registers reinitialized. In systems that have multiple slaves tied to the bus, the master uses the Alert Response Address (ARA) to determine which slave pulled the alert line low. If multiple slaves are asserting the alert simultaneously, the slave with the lowest address wins. This application report, however, deals with alert response implementation for single slave only; for multi-slave implementation with the alert response, see System Management Bus (SMBus) Specification V2.0.[2]

3.1.2 Control Signal (CONTROL)

On/off control of the slave device is achieved by using the control line in combination with commands on the bus. The control line is configured as active high or active low based on the ON_OFF_CONFIG command. The OPERATION command can then be used to turn the slave device on/off through software.

3.1.3 Packet Error Checking (PEC)

PEC is optionally implemented in PMBus devices but is highly recommended due to the critical nature of data validity in power-management systems. Packet Error Code (also PEC) bytes are generated using the popular CRC-8 algorithm that is based on performing XOR operations on the input bit stream with a fixed CRC polynomial. The PEC byte is calculated on all bytes in the I2C transaction including device address and read/write. PEC does not include start, stop, ACK/NACK, and repeated start bits.

3.2 PMBus Procotol Formats

PMBus defines a set of standard commands for interacting with power management devices. Each command has a defined command code and protocol format. The protocol formats are standard SMBus command protocol formats and the specific protocols for each PMBus command are defined in the Appendix of the PMBus specification[3].

PMBus allows for device specific commands. The SMBus protocol for device specific commands are specified by the device manufacturer.

See the SMBus specification[2] for definition of the command protocols.

Chapter 4

MSP PMBus Library

The MSP430™ PMBus software library is based on PMBus v1.2, but does not support the following features defined in the PMBus v1.2 specification:

- Group and Extended command protocols are not supported
- Block reads and writes are supported up to a maximum of 32 data bytes
- 400kHz bus speed is not supported (optional PMBus v1.2 feature)
- Address resolution protocol(ARP) is not supported (optional PMBus v1.2 feature)

4.1 Library structure

The MSP430™ PMBus software library is implemented as a level on top of the MSP430™ SMBus library.

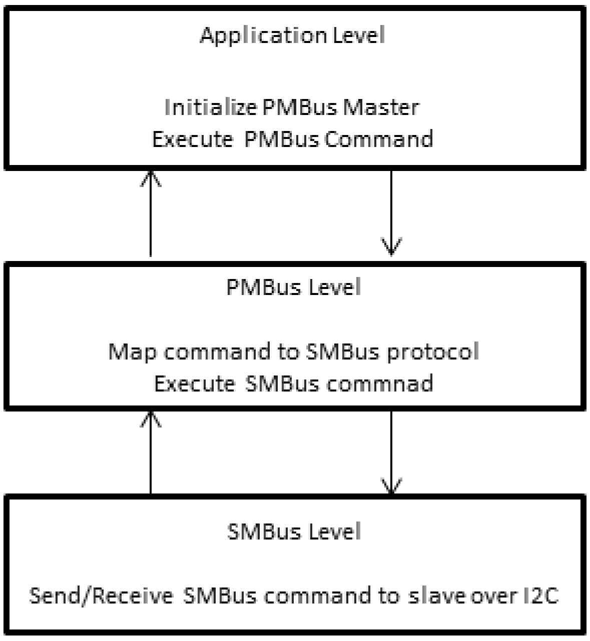


Figure 4.1: MSP430 PMBus Software Layers

The application consists of the code to initialize the MSP430™ hardware and makes calls to the PMBus functions. The PMBus level process the commands it receives from the application level and makes the required function calls to the SMBus library. The SMBus level interfaces with the I2C peripheral to communicate with the PMBus slave device.

See the examples included with the release package for further details on using the library.

4.2 Device Support

The PMBus level source code is device independent, but the SMBus level interacts with the microcontroller I2C peripheral and is device dependent. The PMBus libraries and examples were developed for the following MSP430™ microcontroller families:

- MSP430G2xx3
- MSP430FR5xx_6xx

4.3 Initialization

The initialize the PMBus library for use the application must first initialize the I2C peripheral and, if desired GPIO pins for SMBALERT and CONTROL lines. Then the application must call the PMBus_init function before any PMBus commands.

4.3.1 PEC

The PMBus library implements PEC as an optional feature. Applications that require PEC must enable this feature by calling the PMBus_enablePEC function before any PMBus commands are sent to the slave device.

Generating the PEC byte involves invoking the CRC-8 algorithm provided in a function. The CRC function performs an XOR operation on the input data stream with the CRC-8 polynomial C(x) = x8 + x2 + x1 + 1. This calculation is done in the order bits are received and also can be considered a brute force polynomial division technique. The algorithm used to calculate the CRC byte is modified from the application report CRC Implementation With the MSP430.[3]

PMBus master devices that implement PEC have either prior knowledge of whether or not the slave supports the feature, or they are expected to determine this by reading information from the slave. The master transmitter generates the PEC byte and inserts it at the end of the transmit data stream. If the PEC byte is not acknowledged by the slave, then the transaction is considered invalid.

If the master is receiving, then it generates the extra clock cycles needed to receive the PEC byte from the slave as the last byte of the transaction. Master receivers can then check the validity of the PEC byte by generating their own PEC and comparing it to the one received from the slave device. PMBus slave devices that implement PEC must be able to respond to master devices with or without PEC support. Slave transmitters must have the capability of generating PEC bytes and inserting them as the final byte of any transaction if requested. Slave receivers must be able to accept the PEC byte, check to see if the byte is valid or not, and respond with an ACK/NACK accordingly.[2]

4.4 Initialization

The PMBus specification defines set of command for interfacing with power management devices. Each command is assigned a one byte command code. Some commands have defined transmit and receive protocols for interfacing with the slave device. There are some command codes whose protocols are left as device specific. The MSP430™ PMBus library supports both types of commands

See the `API Guide` for detailed documentation of all the PMBus functions.

4.4.1 PEC

For all the pre-defined protocol PMBus commands, applications can use the PMBus_cmdRead and PMBus_cmdWrite functions. The PMBus library will translate the passed in command code to the required protocol automatically. Define's for all the standard command codes can be found in the PMBus library header file.

A example usage of the PMBus "VIN_ON" command on a TPS544C20 device might look like:

```
// Write VIN_ON 4.25 V
Command_Buff[0] = 0x11; //LSB
Command_Buff[1] = 0xF0; //MSB
int8_t ret = PMBus_cmdWrite(0x1B,           // TPS544C20 Device Address
                            PMB_VIN_ON,     // #define of the VIN_ON 0x35 command code - see pmbus.h
                            Command_Buff,   // TX buffer
                            2,              // TX bytes sent
                            100000);        // Software response timeout cycles
```

4.4.1.1 Device specific PMBus command interface

The MSP430™ PMBus library provides functions to call the low level SMBus command protocols. These functions enable applications to access the device-specfic PMBus commands. The function names map directly to SMBus protocol names to enable easy mapping from the slave device's PMBus documentation to MSP430™ PMBus function calls.

For example, the PMBus DEVICE_CODE (0xFC) command is manufacturer specific. The TPS544C20 device device defines this command's read transaction to use the SMBus "ReadWord" protocol.

```
// Verify the ID of the device (0x0153 for TPS544C20 Rev 3)
// Note that DEVICE_CODE (0xFC) is manufacturer specific but it's accessed
// with Read_Word/Write_Word commands.
// So, we call the specific ReadWord function directly
uint8_t Resp_Buff[PMB_MAX_PACKET_SIZE + 4];
if ((PMBus_cmdReadWord(TPS544C20_ADDRESS,
                       0xFC,
                       Resp_Buff,
                       timeout)  != PMBUS_RET_OK)
   || ( Resp_Buff[0] != 0x53)
   || ( Resp_Buff[1] != 0x01) )
{
    // Stay in loop if device didn't respond OK
    while(1) ;
}
```

4.5 SMBALERT and Alert Response Address (ARA)

In the event that a SMBALERT was detected, the PMBus library provides a PMBus_ReceiveByteARA function to enable applications to send the receive byte packet to the alert response address to determine which device triggerd the alert. Support for SMBALERT and ARA is optional, and manufacturer specific. Applications must consult the device documentation for usage details for their PMBus slave device.

An example application showing the usage of SMBALERT and ARA with the TPS544C20 is included in the examples included in the installation package.

4.6 CONTROL

Applications may define a CONTROL line to provide on/off control for PMBus devices. An example application showing how to configure a MSP430™ microcontroller GPIO pin as a CONTROL line is included in the examples directory.

Chapter 5

Examples

This section describes the examples included with the PMBus library.

5.1 Supported Devices

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

Device	I2C Interface
MSP430FR59xx	Enhanced Universal Serial Communication Interface (eUSCI)
MSP430G2xx3	Universal Serial Communication Interface (USCI)

MSP430 devices supported by PMBus library examples

The code examples and PMBus 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 I2C interface with some considerations such as pin usage, memory and timeout-timer availability.

5.2 Summary of the examples

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

Each of the examples are written as PMBus master application communicating to a `TPS544C20`.

Name	Description
PMBus_Master01_TPS544C20	Basic PMBus communication
PMBus_Master02_TPS544C20_SMBAlert	PMBus with SMBALERT
PMBus_Master03_TPS544C20_Control	PMBus with device on/off through CONTROL line

PMBus library examples

The hardware setup and their funtionality are described in more detail in the following sections.

5.3 Running the examples

5.3.1 Obtaining Code Composer Studio (CCS) or IAR Kickstart

The PMBus library and examples build and run on both the `IAR and CodeComposer Studio(CCS)` environments for MSP430™ microcontrollers. See the Release Notes in the PMBus library package for specific

IAR and 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 8K of memory can be run on both free versions. Applications that require more than 8K of memory cannot be build using the free IAR Ickstart tool. Instead, the free version of CCS can be used, or a licensed version of either environment.

5.3.2 Hardware support

Most of the examples can run on any hardware TI sells in the eStore, supporting the devices mentioned in the Supported DevicesSupported Devices"". 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 PMBus/SMBus layers.

The following tables list the resources used by the examples:

Description	Resource
I2C	eUSCI_B0
I2C_SDA	P1.6/USCB0SDA
I2C_SCL	P1.7/USCB0SCL
SMBAlert	P1.5
Control1	P1.4
Control2	P1.3
LED0	P1.0
LED1	P4.6

MSP430FR59xx examples

Description	Resource
I2C	USCI_B0
I2C_SDA	P1.7/USCB0SDA
I2C_SCL	P1.6/USCB0SCL
SMBAlert	P2.5
Control1	P2.4
Control1	P2.3
LED0	P1.0
LED1	P2.0

MSP430G2xx3 examples

5.3.3 Obtaining and building the examples

The PMBus Library examples have the following structure:

```
<install>
  |--smbuslib           <- SMBus Library source code
  |    |--driverlib     <- MSP430 driverlib used to access the microcontroller peripherals
  |--src                <- Contains the PMBus Library source code
  |--examples
       |--MSP430*       <- MSP430 device supported by examples
           |--CCS
           |    |--PMB*.projectspec  <-- Individual project files for CCS
           |
           |--IAR
```

```
|     |--PMB*
|          |--*.{ewd,ewp}  <-- Individual project files for IAR
|          |--*.eww        <-- Project workspace file for IAR
|--Src
     |--PMB*
          |-- main.c            <-- The example main code
          |-- Master_HAL.{h,c} <-- Hardware abstraction layer for example
          |-- *.{h,c}           <-- Other source files used by example
```

5.3.3.1 IAR projects

Each IAR project has a workspace. To use the examples in IAR, open the corresponding workspace file (∗.eww) by selecting File->Open->Workspace.

The project can be build using the F& key, Menu->Project->rebuild All, or clicking on the *Make* icon. Projects can be downloaded to the MSP430™ microcontroller using Ctrl+D, Menu->Project->Download, or by clicking the *Download and Debug* icon.

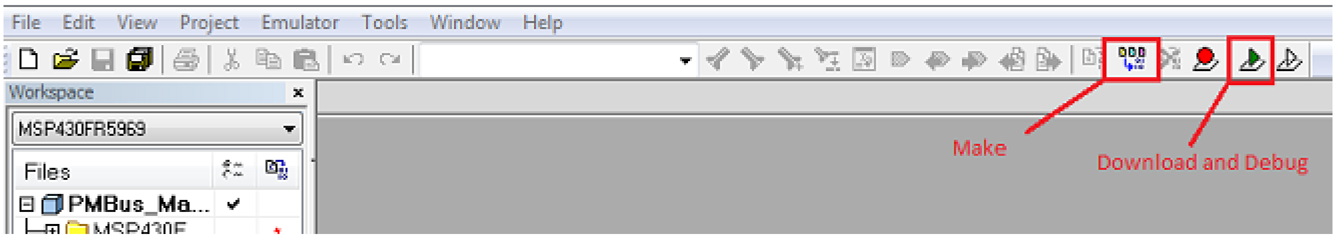


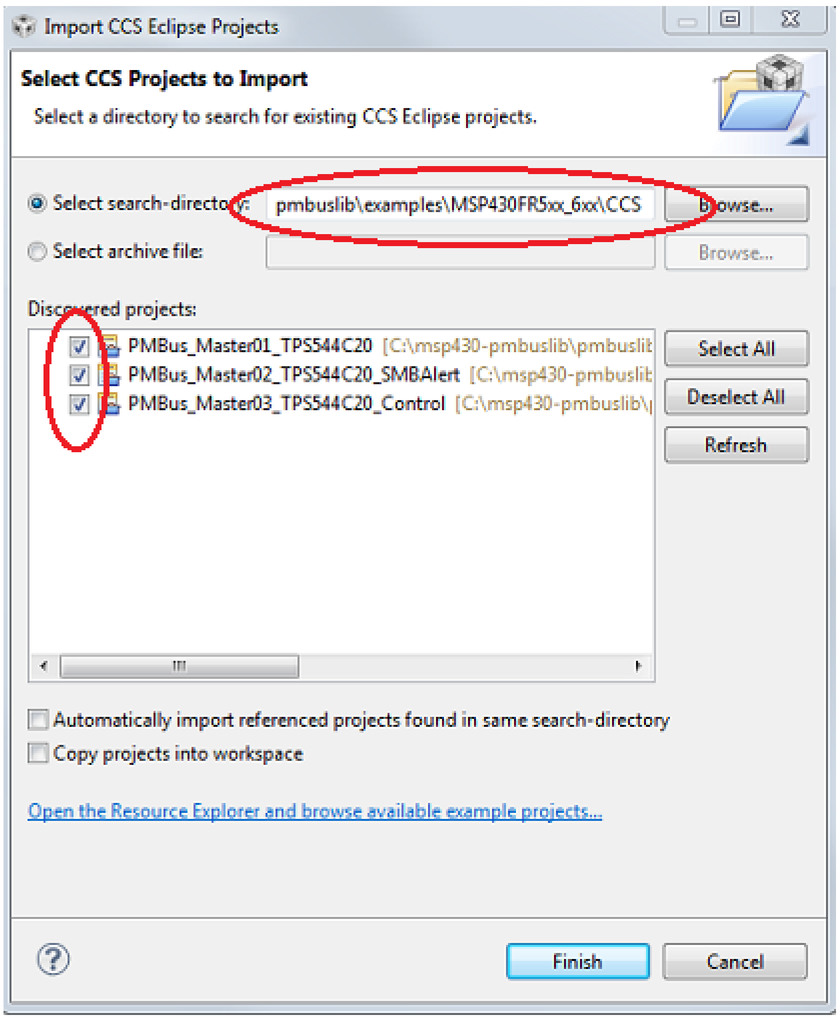
Figure 5.1: IAR project build and debug icons

5.3.3.2 CCS projects

CCS project must be imported into a CCS workspace. The projects are defined by ∗.projectspec files, which contain the information CCS needs to import the project.

Open CCS, and choose Project->Import CCS Projects. Browse to the examples/MSP430∗/CCS directory (containing the ∗.projectspec) files). CCS will show a list of all the projects that were discovered in this directory.

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


Import CCS Eclipse Projects

Select CCS Projects to Import

Select a directory to search for existing CCS Eclipse projects.

(●) Select search-directory: pmbuslib\examples\MSP430FR5xx_6xx\CCS

( ) Select archive file:

Discovered projects:

- [✓] PMBus_Master01_TPS544C20 [C:\msp430-pmbuslib\pmbuslib
- [✓] PMBus_Master02_TPS544C20_SMBAlert [C:\msp430-pmbuslib
- [✓] PMBus_Master03_TPS544C20_Control [C:\msp430-pmbuslib\

[ ] Automatically import referenced projects found in same search-directory

[ ] Copy projects into workspace

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

Figure 5.2: 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:

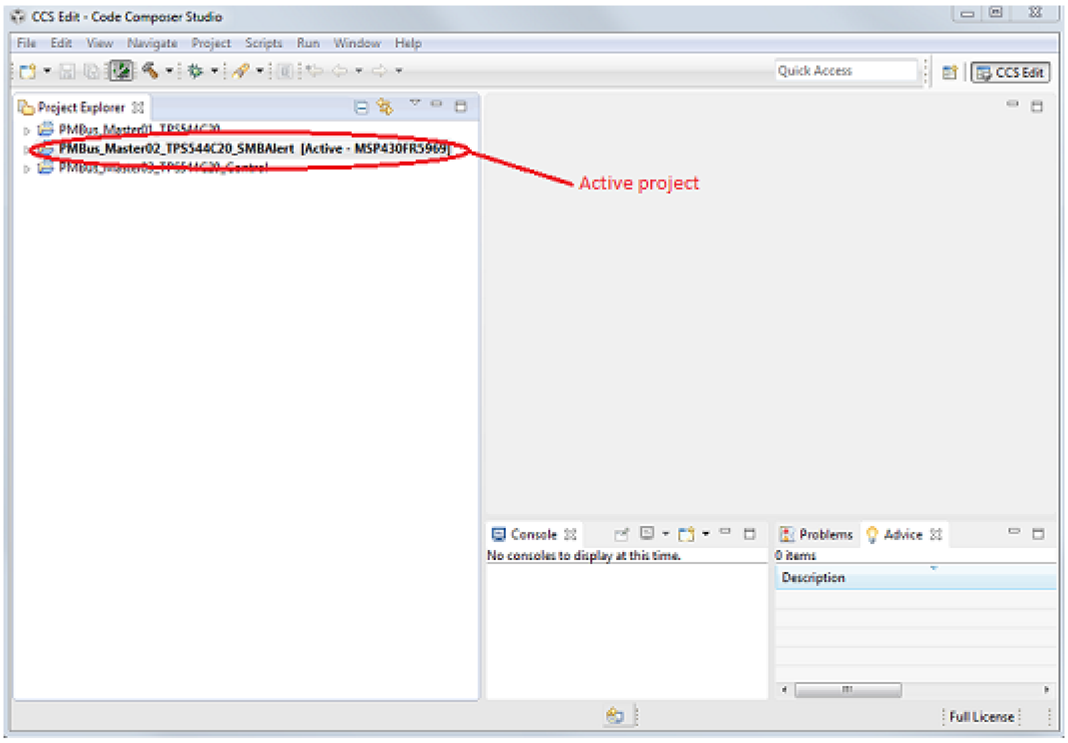

CCS Edit - Code Composer Studio
File Edit View Navigate Project Scripts Run Window Help
Quick Access
CCS Edit
Project Explorer
PMBus_Master01_TPS544C20
PMBus_Master02_TPS544C20_SMBAlert [Active - MSP430FR5969]
PMBus_Master03_TPS544C20_Control
Active project
Console
No consoles to display at this time.
Problems
Advice
0 items
Description
Full License

Figure 5.3: Active project in CCS

The project can be built using Ctrl+B, Menu->Project->Build All, or by clicking the *Build* icon. Projects can be downloaded to the MSP430™ microcontroller using F11, Menu->Run->Debug, or by by clicking the *Debug* icon.

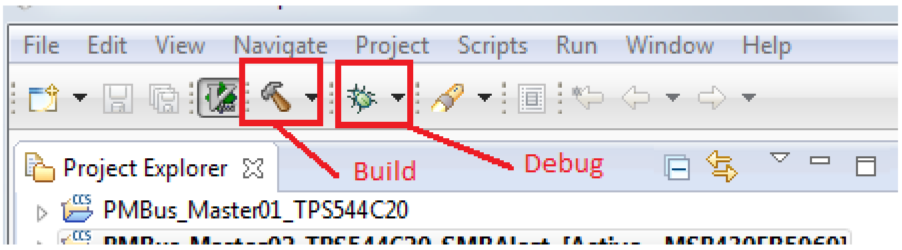


Figure 5.4: CCS project build and debug icons

5.4 Example Descriptions

This section contains information that is common to all PMBus examples. Specific information for each example is provided after this section.

5.4.1 Hardware connection

The required basic connection between Master and Slave PMBus devices is shown below.

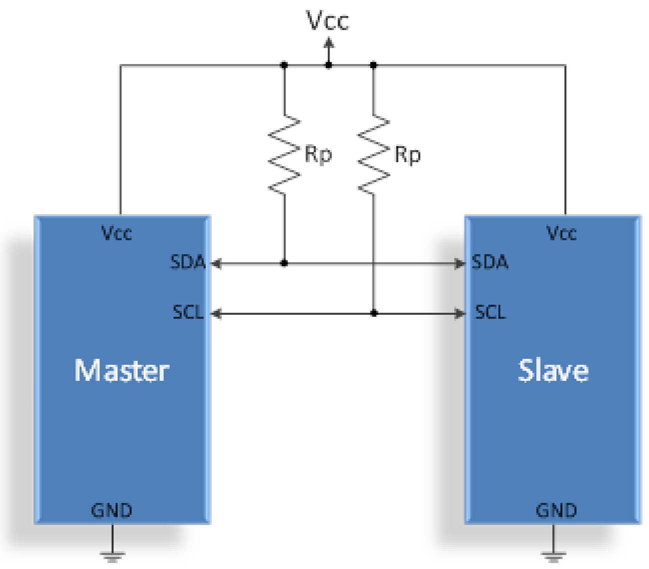


Figure 5.5: Basic connection between Master and Slave

The pins used for SDA and SCL for the supported devies are described in the _Hardware Support_ section.

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

5.4.2 Building and excuting the examples

1. Select and example and import the project into CCS or IAR
2. Connect the master and slave devices. All of the examples are written as PMBus master application communicating to a TPS544C20. With minor modifications, each example could be modified to work with any PMBus compliant device.

```
                            /|\  /|\
                            Rp    Rp
               Master        |    |        Slave
               MSP430        |    |        TPS544C20
         -----------------   |    |      -------------
        |             SDA*|<-|----+----> |DATA        |
        |                 |  |           |            |--> VOUT
        |             SCL*|<-+---------> |CLK         |
        |       /|\       |              |            |
LED0 <--|Px.y*   |---Pb.c*|<-------------|SMBAlert    |
LED1 <--|Pz.a*            |              |            |
        |                 |              |            |
        |         Control*|------------->|CNTL        |
```

1. Open, build and download the example project to the MSP430™ device
2. Check the results as described in the individual example section below.

Check the comments in the source code for each example for detailed information of the example functions.

5.4.3 PMBus_Master01_TPS544C20

This example configures some parameters of the TPS544C20 to illustrate some basic calls of the PMBus Library.

After initializing the MSP430™ device and GPIO the example will:

1. Initialize the PMBus library
2. Verify the DEVICE_CODE for the TPS544C20
3. Write the PMBus VIN_OFF command to set the value of the input voltage at wich the unit should stop power conversion
4. Write the PMBus VIN_ON command to set the value of the input voltage at wich the unit should start power conversion
5. Write the PMBus IOUT_OC_WARN_LIMIT command to set the value of the output current that cause an output overcurrent warning
6. Write the PMBus ON_OFF_CONFIG and PMB_OPERATION commands to turn on the TPS544C20

5.4.4 PMBus_Master02_TPS544C20_SMBAlert

This example configures the TPS544C20 and enables SMBAlert functionality. When an alert is detected by the slave device, the MSP430 reads the fault and clears the flag. If enabled, MSP430 will send an ARA packet to double-check if the TPS544C20 is the device asserting the SMBAlert line.

The example can be tested by lowering the TPS544C20's VIN.

After initializing the MSP430™ device and GPIO the example will:

1. Initialize the PMBus library
2. Clear SMBAlert interrupt flags and eanble SMBAlert interrupts
3. Verify the DEVICE_CODE for the TPS544C20
4. Write the PMBus VIN_OFF command to set the value of the input voltage at wich the unit should stop power conversion
5. Write the PMBus VIN_ON command to set the value of the input voltage at wich the unit should start power conversion
6. Write the PMBus IOUT_OC_WARN_LIMIT command to set the value of the output current that cause an output overcurrent warning
7. Write the TPS544C20 device specific PMBus command to unmask all alerts, and enable auto-ARA
8. Write the PMBus ON_OFF_CONFIG and PMB_OPERATION commands to turn on the TPS544C20
9. Enter LPM and wake up on a SMBAlert detection. When an alert is detected
    - use ARA to check if the TPS544C20 was the device calling the alert,
    - send PMBus STATUS_WORD to retrieve the fault
    - send PMBus CLEAR_FAULTS to clear the TPS544C20 fault status

5.4.5 PMBus_Master03_TPS544C20_SMBControl

This example configures the TPS544C20 using the CONTROL line. The CONTROL line turns the output ON/OFF together or independently of the OPERATION command. The signal is optional per PMBus spec but it is implemented in many PMBus devices.

After initializing the MSP430™ device and GPIO the example will:

1. Initialize the PMBus library
2. Clear SMBAlert interrupt flags and eanble SMBAlert interrupts
3. Verify the DEVICE_CODE for the TPS544C20

4. Write the PMBus VIN_OFF command to set the value of the input voltage at wich the unit should stop power conversion
5. Write the PMBus VIN_ON command to set the value of the input voltage at wich the unit should start power conversion
6. Write the PMBus IOUT_OC_WARN_LIMIT command to set the value of the output current that cause an output overcurrent warning
7. Write the TPS544C20 device specific PMBus command to unmask all alerts, and enable auto-ARA
8. Write the PMBus ON_OFF_CONFIG and PMB_OPERATION commands to COONTROL-only mode.
9. Set the GPIO CONTROL line to turn the TPS544C20 on.

Chapter 6

References

1. MSP430™ System Management Bus (SMBus) library (`www.ti.com/tool/msp430-smbus`)
2. System Management Bus (SMBus) Specification V2.0 (`www.smbus.org`)
3. CRC Implementation With the MSP430 (`SLAA221`)
4. Power Management Bus (PMBus) Specification Part I and II, Version 1.2 (`www.pmbus.org`)
5. Using the USCI I2C Master (`SLAA382`)

Chapter 7

Disclaimer