Application Report
SLVA783A – June 2016 – Revised April 2017

TPS6598x FW Update From Embedded Controller Over I^2C

ABSTRACT

The TPS6598x is a stand-alone USB Type-C™ and Power Delivery (PD) controller providing cable plug and orientation detection at the USB Type-C connector. The TPS6598x contains a RAM-based processor and loads application code from an SPI Flash to operate. When a firmware (FW) or patch update for the TPS6598x is available, the FW and patch-bundle contained in the SPI Flash can be updated by an Embedded Controller (EC) communicating with the TPS6598x via I^2C. The TPS6598x application code includes a set of simple ASCII commands that enables the FW update process. The EC writes data to the TPS6598x, sends these Flash-specific ASCII commands in the correct sequence to overwrite the Flash memory, and then resets the TPS6598x to begin running the new, upgraded FW. This application note explains the procedure of updating the TPS6598x FW from an EC over I^2C and provides example C++ code that can be implemented on any EC.

Contents

List of Figures

List of Tables

1 Introduction

The goal of this application note is to provide an example of how the FW update procedure of the TPS6598x can be performed from an EC over I^{2}C. The TPS6598x is an *SPI Master* connected directly to an SPI Flash IC, containing the application code that runs on the RAM-based processor of the TPS6598x after it is loaded during the boot-up sequence. Occasionally, a TPS6598x FW update may be required to support new features of the USB PD specification or resolve interoperability issues with other products in the market. Therefore, in addition to loading application code from the SPI Flash, the TPS6598x can also erase sections of the SPI Flash memory and write new data into memory to upgrade from the current FW version to a newer FW version.

The TPS6598x is an I^{2}C slave which can be controlled by an EC - the I^{2}C master in the system - to modify USB Type-C and PD behavior dynamically in a system. In order for the TPS6598x to act as an SPI Master after boot and modify the Flash memory, the EC must use I^{2}C reads, writes, and special commands to pass a new FW binary file into the TPS6598x. Finally, the EC must force a hardware reset for the TPS6598x to re-boot, load the new FW version into memory, and execute the new application code.

Although this application note is written specifically for the TPS6598x as an example, it also applies to the TPS65986 and TPS65981.

With minor modifications, these steps and instructions can be used to upgrade the firmware patch on TPS65982D. The subsequent sections will list these differences wherever applicable.

2 Related Material

The TPS6598x data sheet, the *TPS6598x and TPS65986 Firmware User's Guide* (SLVUAH7), the TPS6598X Configuration Tool, and the TPS6598X Utilities Tool are resources required to implement this FW update in an application of the TPS6598x. Tables in Appendix A of the *TPS6598x and TPS65986 Firmware User's Guide* are required to understand I^{2}C Reads, Writes, and 4CC commands used in the FW update procedure, and the TPS6598X Utilities Tool should be used to test the I2C reads, writes, and commands before implementing C-code to perform the full FW update.

3 Background of Firmware Update Procedure

3.1 Flash Memory Organization

The firmware and patch-bundle of the TPS6598x is contained in the memory of the attached SPI Flash IC. The organization of this data in memory is explained in the TPS6598x data sheet (SLVSD02) using variables and relative locations because the TPS6598x also works in applications where the SPI Flash memory is shared with another IC. Memory organization might differ between the various variants of TPS6598x, and the EC application developers shall account for these differences when writing host-programming applications to update TPS6598x firmware and patch-bundle.

In this application note, the flash memory organization is explained based on the assumption that the SPI Flash is dedicated to the TPS6598x and not shared. For redundancy, the FW is copied into two regions, called Region 0 (low region) and Region 1 (high region). Before a FW update and after a FW update is complete, the data in Region 0 and Region 1 will be identical. If the full memory of the SPI flash is read directly, it will contain pointers and both Regions, which will simply be referred to as a "*flash.bin*" file.

A depiction of the full *"flash.bin"* flash memory organization for TPS65982 is shown in Figure 1.

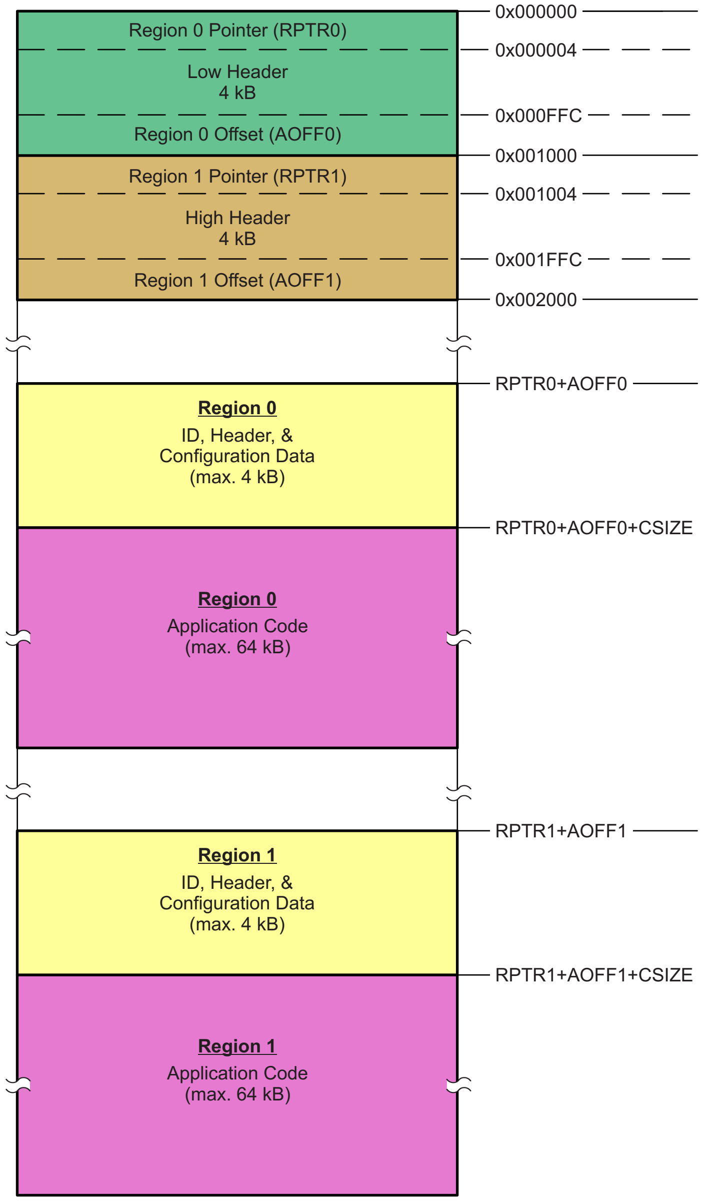


Figure 1. TPS65982 SPI Flash Memory Organization of Region 0 and Region 1

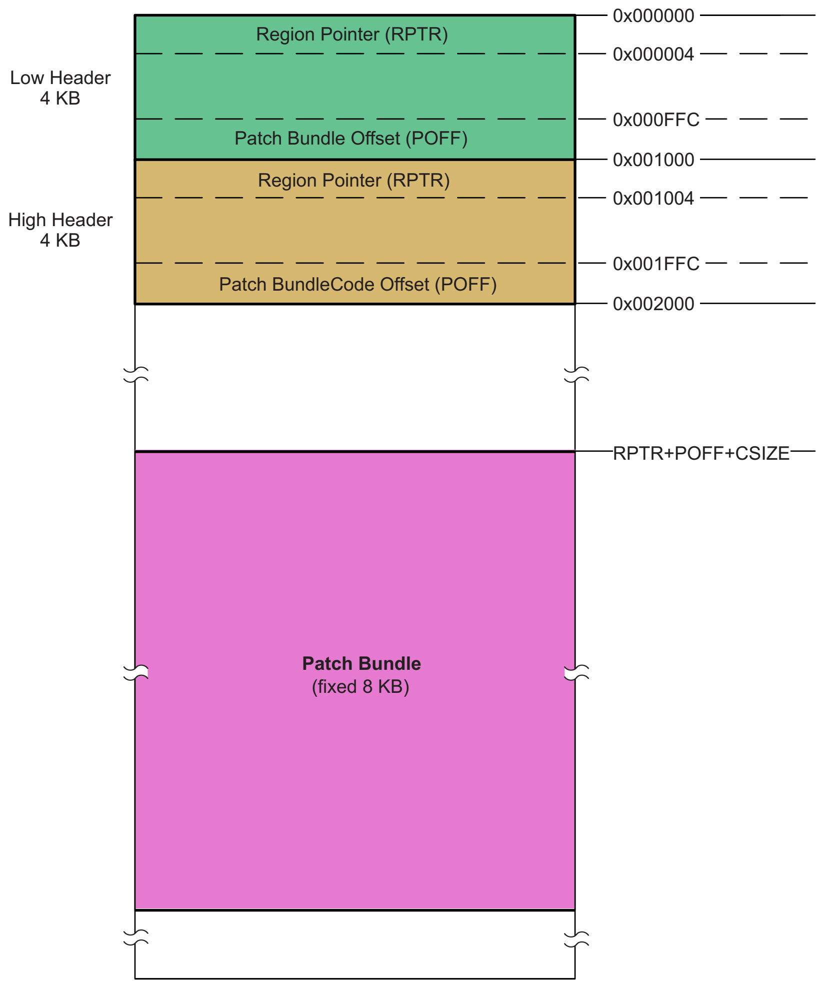


Figure 2. TPS65982D SPI Flash Memory Organization of Region 0

Since Region 0 and Region 1 are copies of each other, the binary data in each region is identical. The region header and the application code combined are referred to as the “low region” and the “high region”. When the FW update is performed over I^{2}C, the input file must be a *“low-region.bin”* which is placed into both the low region and high region in Flash memory.

3.2 ASCII Commands

The FW update process utilizes the special 4CC ASCII character commands listed in Table 1. All of the 4CC commands that pertain specifically to the Flash IC begin with the ASCII characters *‘FL''* and end with 2 additional ASCII characters that help identify the function of the command. The *‘GAID”* command is a unique type of 4CC command which requires no data entered into the DATA1/2 register to execute. The *‘GAID''* command is used at the end of the FW update procedure to re-boot the TPS6598x and reload the new FW version from non-volatile Flash memory.

Table 1. List of 4CC Commands Used in TPS6598x FW Update Over I^{2}C

Name of 4CC Command	ASCII	Hex Value	Data1/2 Input Length
Flash read region	‘FLrr’	0x46 0x4C 0x72 0x72	32-bits (4 Bytes)
Flash read region	‘FLem’	0x46 0x4C 0x65 0x6D	1-bit (1 Byte min.)
Flash address	‘FLad’	0x46 0x4C 0x61 0x64	32-bits (4 Bytes)
Flash write data	‘FLwd’	0x46 0x4C 0x77 0x64	64 Bytes
Flash verify	‘FLvy’	0x46 0x4C 0x76 0x79	32-bits (4 Bytes)

Table 1. List of 4CC Commands Used in TPS6598x FW Update Over I²C (continued)

Name of 4CC Command	ASCII	Hex Value	Data1/2 Input Length
Hardware Reset of TPS6598x	‘GAID’	0x47 0x41 0x49 0x44	None

When a 4CC command is given, the following steps must be performed in order:

1. Write input data to DATA1/2 register (0x09/0x11)
2. Write 4CC command characters to CMD1/2 register (0x08/0x10)
3. Read back data from CMD1/2 register (0x08/0x10) until it clears
   - Clear return value = 0x00 0x00 0x00 0x00
   - Incorrect command input return value = ‘!CMD’ = 0x21 0x43 0x4D 0x44
4. Read back data output and error return code from DATA1/2 register (0x09/0x11)
   - This allows the TPS6598x to clear the DATA1/2 buffer and accept new data

3.3 Firmware Update Flow

The FW update procedure follows the flow depicted in Figure 3. This FW flow shows at a high level the order in which the *‘FL__’* commands must be completed. Intermediate I²C Reads and Writes to the CMD1/2 and DATA1/2 registers are needed to ensure timing requirements are met and buffers are cleared, and these Reads and Writes are included in the example source code provided in the next section, Example C-Code.

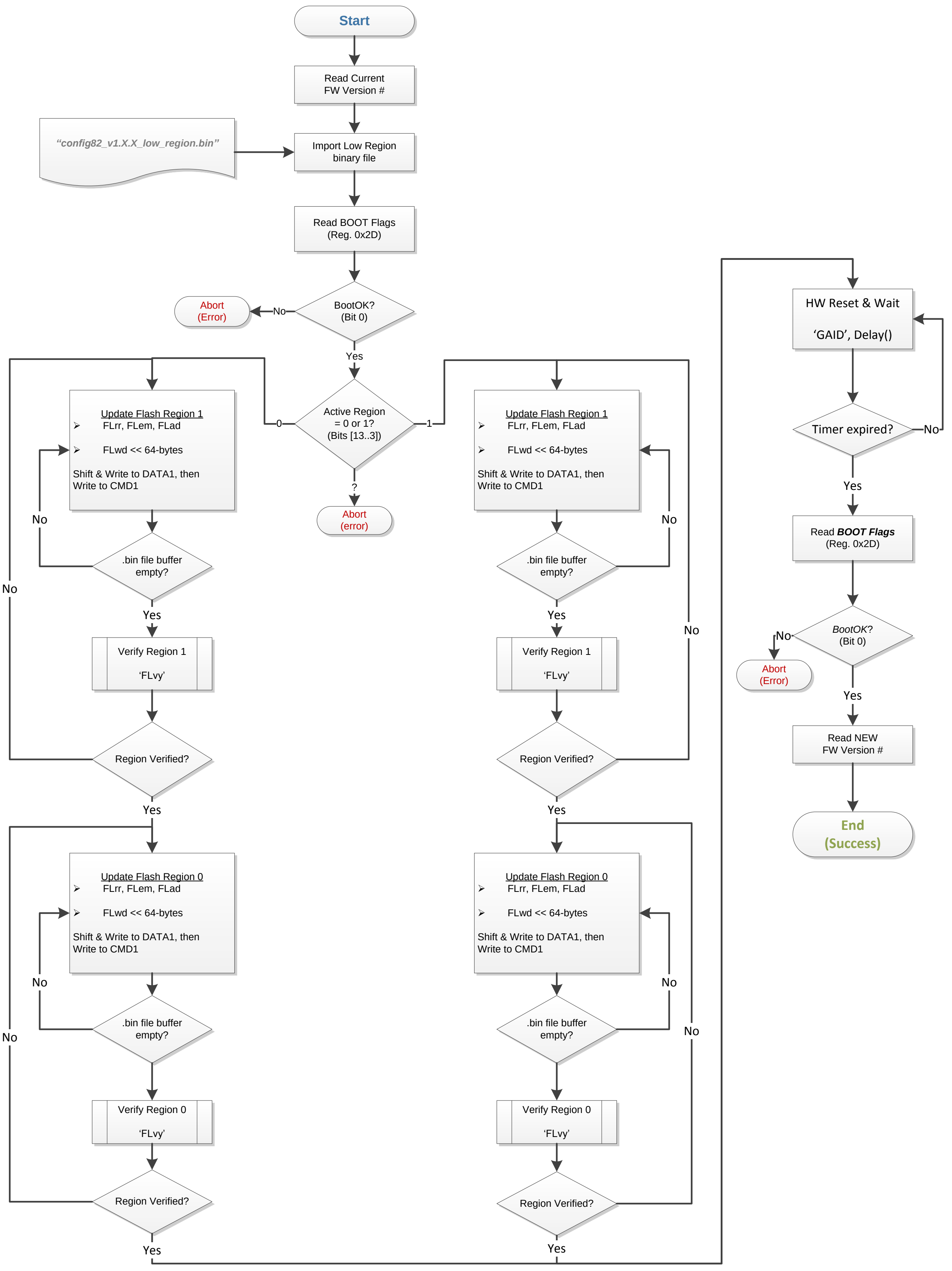


Figure 3. Flow Diagram of FW Update Process

4 Example C-Code

The following example code is copied from C++ source code written to test the TPS6598x FW update procedure from an embedded controller via I^{2}C. The code is broken up into 6 files: 3 .c files and 3 .h header files. This project was tested using the TMC123GXL Tiva-C processor from Texas Instruments, but the supporting code, drivers, and libraries that are specific to the Tiva-C are outside the scope of this document because they are not relevant when implementing the TPS6598x FW update procedure on other embedded controllers. Table 2 lists the relevant .c/.h files in the project and briefly explains their purpose for performing the FW update.

Table 2. List of .c/.h Files in the TPS6598x FW Update Project

File Name	Type	Description
hostIF82.h	Header	Header file to define Register Addresses and Register Lengths for the TPS6598x, define 4CC ASCII commands as variables, and create structures for selected Registers to define individual fields needed to perform the FW update
i2cFwUpdate82.c	C++	C file containing the core code of the FW Update procedure, the *FWUpdate82()* function and local functions such as *RegionUpdate82()* which accepts a Boolean input: '0' = Update Region 0 '1' = Update Region 1
i2cFwUpdate82.h	Header	Header file allowing the *FWUpdate82()* function to be called from *main.c*
i2cHandlerHostTo82.c	C++	C file containing the fundamental global functions used for controlling the TPS6598x: **ReadIICRegister()** → I^{2}C Read – A generic I2C read function that can be used for any register of the TPS6598x **WriteIICRegister()** → I^{2}C Write – A generic I2C write function that can be used for any register of the TPS6598x, using the register length and data as an input **ourCC_Command()** → 4CC Commands – A case statement uses the switch() operator using selected 4CC variables required to perform the FW update
i2cHandlerHostTo82.h	Header	Header file allowing the *"i2cHandlerHostTo82.c"* functions to be called from other .c files in the project
main.c	C++	C file that initializes the Tiva-C processor appropriately, then calls the *FWUpdate82()* function in the main while loop. Because the return value is fully handled, resulting in a break regardless of the outcome, the entire project is essentially a script that runs 1 time and exits.

4.1 Excerpt of Code From “hostIF82.h” Header File

The following code is an excerpt from the *“hostIF82.h”* file. Code not required for the FW update procedure has been redacted.

```
#ifndef __HOSTIF82_H__
#define __HOSTIF82_H__

#include <stdbool.h>
#include <stdint.h>

//******************** Enable/Disable UART Stream **********
#define UART_Stream_ON

//*********************************************************
//******************** Global 82 I2C Constants *************
//*********************************************************

//**************** TPS65982 Slave Address *****************
#define DefAddr1   0x38 //7-bit I2C Slave address 0 for primary interface
#define DefAddr2   0x3F //7-bit I2C Slave address 1 for primary interface

//Max Arugment Length [First Byte in I2C Read/Write is Length Byte]
#define MAX_ARG_LENGTH  65

//******************** Register #'s **********************
#define REG_CMD1                0x08  //used for the primary command interface.
#define REG_DATA1               0x09  //used for the primary command interface.
#define REG_VERSION             0x0F  //FW VERSION ##.##.##.##
#define REG_CMD2                0x10  //used for the secondary command interface.
#define REG_DATA2               0x11  //used for the secondary command interface.
…
#define REG_SYS_CONFIG          0x28  //System Config
…
#define REG_BOOT_FLAGS          0x2D  //Boot Flags

//******************** Register Lengths **********************
…
#define lenCMD1                 4  //used for the primary command interface.
#define lenDATA1                64  //used for the primary command interface.
#define lenVERSION              4  // FW VERSION ##.##.##.##
#define lenCMD2                 4  //used for the secondary command interface.
#define lenDATA2                64  //used for the secondary command interface.
…
#define lenSYS_CONFIG           10  //System Config
…
#define lenBOOT_FLAGS           2  //Boot Flags
…

//******************** 4CC Words **********************
// Convert 4CC to 32 bit word - Little Endian
#define CONV_4CC_TO_WORD(_A_, _B_, _C_, _D_) ((_D_ << 24) | (_C_ << 16) | (_B_ << 8) | _A_)

#define nCMD             CONV_4CC_TO_WORD('!','C','M','D')
…
#define FLrr       CONV_4CC_TO_WORD('F', 'L', 'r', 'r')
…
```

```
  #define FLem       CONV_4CC_TO_WORD('F', 'L', 'e', 'm')
  #define FLad       CONV_4CC_TO_WORD('F', 'L', 'a', 'd')
  #define FLwd       CONV_4CC_TO_WORD('F', 'L', 'w', 'd')
  …
  #define FLvy       CONV_4CC_TO_WORD('F', 'L', 'v', 'y')
  …
  #define GAID       CONV_4CC_TO_WORD('G', 'A', 'I', 'D')


   //************ 82 Selected Registers Structures ***********************
   //Read-ONLY Registers

   //Static - Set in FW image, so can only change after successful FW update
   typedef struct {
       uint32_t FW_VERSION_B0            :8; //Byte 0 = 0x00 as of 04-12-2016
       uint32_t FW_VERSION_B1            :8; //Byte 1 = Major Revision = 0x01 on 04-12-2016
       uint32_t FW_VERSION_B2            :8; //Byte 2 = Minor Revision = 0x07 on 04-12-2016
       uint32_t FW_VERSION_B3            :8; //Byte 3 = Bug Fix = 0x06 on 04-12-2016

       } tFWVersion82;

   //Dynamic - Set by BOOT Code based on conditions during Boot
   typedef struct {
       uint32_t BootOk              :1; // Bit 0
       uint32_t ExtPhvSwitch        :1; // Bit 1
       uint32_t DeadBatteryFlag     :1; // Bit 2
       uint32_t SpiFlashPresent     :1; // Bit 3
       uint32_t Region0             :1; // Bit 4
       uint32_t Region1             :1; // Bit 5
       uint32_t Region0Invalid      :1; // Bit 6
       uint32_t Region1Invalid      :1; // Bit 7
       uint32_t Region0FlashErr     :1; // Bit 8
       uint32_t Region1FlashErr     :1; // Bit 9
       uint32_t reserved1           :1; // Bit 10
       uint32_t UartCrcFail         :1; // Bit 11
       uint32_t Region0CrcFail      :1; // Bit 12
       uint32_t Region1CrcFail      :1; // Bit 13
       uint32_t CustomerOtpInvalid :1; // Bit 14
       uint32_t reserved2           :1; // Bit 15
   } tBootFlags82;

   //Read-Write Registers (excluding CMD1 & DATA1, which are special)

   // Note: Writing to SysConfig Register causes a Disconnect on the Type-C Port
   typedef struct {
       uint32_t  PortInfo                  :3;
       uint32_t  ReceptacleType            :3;
       uint32_t  TypeCCurrent              :2;
       uint32_t  VCONNsupported            :2;
       uint32_t                            :4;
       uint32_t  HighVoltageWarningLevel   :1;
       uint32_t  LowVoltageWarningLevel    :1;
       uint32_t  OvpTripPoint              :6;
       uint32_t  OvpUsage                  :2;
       uint32_t  PP_5V0config              :2;
       uint32_t  PP_HVconfig               :2;
       uint32_t  PP_HVEconfig              :3;
       uint32_t                            :1;
       uint32_t  BC12enable                :1;
       uint32_t  USBRPenable               :1;
       uint32_t  USBEPenable               :1;
```

```
        uint32_t  USB3rate                   :2;
        uint32_t  USB2Supported              :1;
        uint32_t  AudioAccessorySupport      :1;
        uint32_t  DebugAccessorySupport      :1;
        uint32_t  PoweredAccessorySupport    :1;
        uint32_t  RSense                     :1;
        uint32_t  TrySRCSupport              :1;
        uint32_t  BillboardAllowed           :1;
        uint32_t                             :2;
        uint32_t  PP_EXT_OC_Timeout          :5;
        uint32_t  ResetZTimeoutCount         :6;
        uint32_t  ResetZTimeoutClock         :2;
        uint32_t  Vout3V3SupThresh           :3;
        uint32_t  Vout3V3enable              :1;
        uint32_t                             :1;
        uint32_t                             :2;
        uint32_t  setUvpTo4P5V               :1;
        uint32_t  UvpTripPoint5V             :3;
        uint32_t  UvpUsageHV                 :3;
        uint32_t                             :7;
    } tSysConfig82;


    #endif //__HOSTIF82_H__
```

Minor modifications apply to the previous data-structures if the application is being developed for TPS65982D. Details of these modifications follow:

Before

```
typedef struct {
    uint32_t BootOk :1; // Bit 0
    …
    …
```

After

```
typedef struct {
    uint32_t PatchHeaderErr :1; // Bit 0
    …
    …
```

Before

```
typedef struct {
    …
    uint32_t reserved1        :1; // Bit 10
    …
    …
```

After

```
typedef struct {
    …
    uint32_t PatchDownloadErr :1; // Bit 10
    …
    …
```

4.2 Excerpt of Code From “i2cFwUpdate82.c” C File

The following code is an excerpt from the *“i2cFwUpdate82.c”* file. Code not required to complete the FW update procedure, such as I²C Reads and Writes for debug only, has been redacted.

```
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
…
// I2C FW Update files - Start
#include "i2cFwUpdate82.h"
#include "i2cHandlerHostTo82.h"
#include "hostIF82.h"
// I2C FW Update files - End

//*********Global Variables********************
//Used in FW Update
tBootFlags82 BootFlags82;
tStatus82 Status82;
tSysConfig82 SysConfig82;
tFWVersion82 FWVersion82;
//*********************************************

//*********Local Variables*********************
//Used in FW Update
uint32_t tempData;
uint8_t i2cReadData[64];
uint8_t i2cCmdRtn[4];
uint8_t i2cWriteData[64];
uint8_t i,j,i2cDataData[64];
uint8_t i2cFlashData[64]; //Temp data
uint8_t     oldVerFW82[4];
uint8_t     newVerFW82[4];
uint32_t i2cDataFLemIn[2];
```

```
uint8_t count8bit;
uint32_t count32bit;
bool readError = false;
bool writeError = false;
bool fourCCerror = false;
bool rgn0Vrfy = false;
bool rgn1Vrfy = false;
bool rgnVrfy[2] = {false,false};
bool i2cFWUpdateDone = false;
//*********Local Constants*********************

//Used in FW Update
const uint8_t i2cDataClr[4] = {0x00, 0x00, 0x00, 0x00};
const uint8_t i2cDataFlrrRgn0[4] = {0x00, 0x00, 0x00, 0x00};
const uint8_t i2cDataFlrrRgn1[4] = {0x01, 0x00, 0x00, 0x00};
const uint32_t i2cDataFlrrRgn[2] = {0x0, 0x1};
const uint32_t rgnPntr1Loc[2] = {0x0, 0x1000};
const uint32_t rgnPntr2Loc[2] = {(0x1000-0x40), (0x2000-0x40)};
const uint32_t rgnPntr1Val[2] = {0x2000, 0x20000};
const uint32_t rgnPntr2Val[2] = {0x0, 0x0};
const uint32_t rgnPntr1ValNew[2] = {0x1234, 0x12345};
const uint32_t rgnPntr2ValNew[2] = {0xDCC, 0xDCBB};
const uint32_t rgnPntrVal[2] = {0x00002000, 0x00020000};
const uint32_t sectorCount = 17;

const uint32_t lowRgnFalsePntr = 0x0100E0AC;
//********************************************
…

void init64ByteDataFF(void)
{
          for (i = 0; i < 64; i++)
        {
            i2cFlashData[i] = 0xFF;
        }
}

void initCountUpDwn64(bool countArray[8])
{
     if (countArray[0]== 0)
     {
             for (i=0; i<8; i++)
             {
                     for (j=0; j<8; j++)
                     {
                             i2cFlashData[i*8+j] = i*8 + j;
                     }
             }
     }
     else
     {
             for (i=0; i<8; i++)
             {
                     for (j=0; j<8; j++)
                     {
                             i2cFlashData[i*8+j] = 63-(i*8 + j);
                     }
             }
     }
}

bool RegionUpdate82(bool rgnNum)
{
     //Write new Data in Flash IC for Region0/1
     //Determine Pointer0/1 value
     fourCCerror = fourCC_Command(FLrr, (uint8_t *)&i2cDataFlrrRgn[rgnNum]);
     readError = ReadIICRegister(I2C1_BASE, DefAddr1, REG_DATA1, 4, i2cReadData);

#ifdef UART_Stream_ON
```

```
     UARTprintf("Region 0 (or 1) Pointer =                0x%02x%02x%02x%02x\n\n", \
i2cReadData[3], i2cReadData[2], i2cReadData[1], i2cReadData[0]);
     DelayInMilliseconds(1000); //Wait for 1 seconds to read UART Stream in Putty terminal
#endif
     tempData = (i2cReadData[0] | (i2cReadData[1] << 8) | (i2cReadData[2] << 16) | \
(i2cReadData[3] << 24));

     if (tempData == rgnPntrVal[rgnNum])
     {
             i2cDataFLemIn[0] = rgnPntrVal[rgnNum];
             i2cDataFLemIn[1] = sectorCount;
             //Erase 64kB+4kB Header of Memory (17 4kB Sectors) starting at Region0/1 Pointer
             fourCCerror = fourCC_Command(FLem, (uint8_t *)&i2cDataFLemIn);
             //Point to location for Writing with Offset of Region Pointer. Same as result from FLrr
             fourCCerror = fourCC_Command(FLad, (uint8_t *)&rgnPntrVal[rgnNum]);

             count8bit = 0;
             //64kB+4kB = 65536Bytes + 4094Bytes = 69632/64 = 1088 times to write to DATA1
             for (count32bit = 0; count32bit < 1088; count32bit++)
             {
                     initCountUpDwn64((bool *) &count8bit);
                     //Write DATA in 64 Bytes chunks
                     fourCCerror = fourCC_Command(FLwd, (uint8_t *)&i2cFlashData);
                     count8bit += 1;
                     if (count8bit == 63)
                     {
                             UARTprintf("4kByte Block Written\n");
                             count8bit = 0;
                     }
             }


#if 0
//Updating Data Record is optional, if Flash Images are flipped for redundancy
//currently, this code has not been fully implemented
             i2cDataFLemIn[0] = rgnPntr1Loc[rgnNum];
             i2cDataFLemIn[1] = 1;
             //Erase Region0/1 Pointers in Data by Erasing 1 4kByte sector;
             fourCCerror = fourCC_Command(FLem, (uint8_t *)&i2cDataFLemIn);
             //Point to location of new Pointer 1
             fourCCerror = fourCC_Command(FLad, (uint8_t *)&rgnPntr1Loc[rgnNum]);

             init64ByteDataFF();
             i2cFlashData[0] = rgnPntr1Val [rgnNum] >> 0; //replaced rgnPntr1ValNew
             i2cFlashData[1] = rgnPntr1Val [rgnNum] >> 8; //replaced rgnPntr1ValNew
             i2cFlashData[2] = rgnPntr1Val [rgnNum] >> 16; //replaced rgnPntr1ValNew
             i2cFlashData[3] = rgnPntr1Val [rgnNum] >> 24; //replaced rgnPntr1ValNew
             //Write DATA for NEW Data Record #1
             fourCC_Command(FLwd, (uint8_t *)&i2cFlashData);
             //Point to location of new Pointer 2
             fourCCerror = fourCC_Command(FLad, (uint8_t *)&rgnPntr2Loc[rgnNum]);

             init64ByteDataFF();
             i2cFlashData[0] = 0xFF;
             i2cFlashData[1] = 0xFF;
             i2cFlashData[2] = 0xFF;
             i2cFlashData[3] = 0xFF;
             i2cFlashData[60] = rgnPntr2Val [rgnNum] >> 0; //replaced rgnPntr2ValNew
             i2cFlashData[61] = rgnPntr2Val [rgnNum] >> 8; //replaced rgnPntr2ValNew
             i2cFlashData[62] = rgnPntr2Val [rgnNum] >> 16; //replaced rgnPntr2ValNew
             i2cFlashData[63] = rgnPntr2Val [rgnNum] >> 24; //replaced rgnPntr2ValNew
             //Write DATA for NEW Data Record #2
             fourCCerror = fourCC_Command(FLwd, (uint8_t *)&i2cFlashData);
#endif

             fourCCerror = fourCC_Command(FLrr, (uint8_t *)&i2cDataFlrrRgn[rgnNum]); //Determine
Pointer for Region0/1 value using NEW record
             readError = ReadIICRegister(I2C1_BASE, DefAddr1, REG_DATA1, 4, i2cReadData);
#ifdef UART_Stream_ON
            UARTprintf("Region 0 (or 1) Pointer =                0x%02x%02x%02x%02x\n\n",\
i2cReadData[3], i2cReadData[2], i2cReadData[1], i2cReadData[0]);
            DelayInMilliseconds(1000); //Wait for 1 seconds to read UART Stream in Putty terminal
#endif

            tempData = (i2cReadData[0] | (i2cReadData[1] << 8) | (i2cReadData[2] << 16) |
(i2cReadData[3] << 24));
            fourCCerror = fourCC_Command(FLvy, (uint8_t *)&tempData); //Verify Flash is valid
            readError = ReadIICRegister(I2C1_BASE, DefAddr1, REG_DATA1, 4, i2cReadData);

#ifdef UART_Stream_ON
            UARTprintf("Flash Verify Return Value =  0x%02x\n\n", i2cReadData[0]);
            DelayInMilliseconds(1000); //Wait for 1s to read UART Stream in Putty terminal
#endif

            if (i2cReadData[0] == 0x00)
            {
                    rgnVrfy[rgnNum] = true;
                    UARTprintf("SUCCESS: Flash Update was successful and Verified\n");
            }
            else
            {
                    rgnVrfy[rgnNum] = false;
                    UARTprintf("FAIL: Flash Update was either unsuccessful or Not capable of being \
verified\n");
                    UARTprintf("FAIL: in v5 of this code, it is expected for Region1 to fail Flvy\n");
            }
     }
     else if (tempData == lowRgnFalsePntr)
     {
            UARTprintf("ABORT: Low-Region File found with offset 0x0. This is not a valid 2-region \
flash image\n");
     }
     else
     {
            UARTprintf("ABORT: Un-defined Flash Region pointer found and 'Flash-Image.bin' \
from TI Config Tool not installed correctly with offset 0x0\n");
     }

     return rgnVrfy[rgnNum];
}

extern bool FWUpdate82(void)
{
     readError = false;
     writeError = false;
     fourCCerror = false;

#ifdef UART_Stream_ON
     UARTprintf("\nWelcome to the TPS65982 I2C FW Update program...\n\n");
#endif
     DelayInMilliseconds(2000); //Wait for 2 seconds to read UART Stream in Putty terminal



#if 1
     // Read Version # of FW image running in code
     ReadIICRegister(I2C1_BASE, DefAddr1, REG_VERSION, lenVERSION, (uint8_t *)&FWVersion82);
#ifdef UART_Stream_ON
     UARTprintf("OLD Version # (0xDD.MM.mm.BB) = 0x%02x.%02x.%02x.%02x.\n", \
FWVersion82.FW_VERSION_B3, FWVersion82.FW_VERSION_B2, FWVersion82.FW_VERSION_B1, \
FWVersion82.FW_VERSION_B0);
#endif

     // Read BOOT Flags 1st to determine active region
     ReadIICRegister(I2C1_BASE, DefAddr1, REG_BOOT_FLAGS, lenBOOT_FLAGS, (uint8_t *)&BootFlags82);
#ifdef UART_Stream_ON
     UARTprintf("BootOk (Bit 0): %d \n", BootFlags82.BootOk);
     UARTprintf("Region0 Attempted (Bit 4): %d \n", BootFlags82.Region0);
     UARTprintf("Region1 Attempted  (Bit 5): %d \n", BootFlags82.Region1);
#endif

     if (BootFlags82.BootOk == 1)
     {

            // Change PortType in SysConfig to Disabled to safely disable Type-C port before FW
Update
            readError = ReadIICRegister(I2C1_BASE, DefAddr1, REG_SYS_CONFIG, lenSYS_CONFIG, \
(uint8_t *)&SysConfig82);
            SysConfig82.PortInfo = 7; //Port Disabled
            writeError = WriteIICRegister(I2C1_BASE, DefAddr1, REG_SYS_CONFIG, lenSYS_CONFIG, \
(uint8_t *)&SysConfig82);

            i2cFWUpdateDone = false;
            do
            {
                   if (BootFlags82.Region1 == 0) //Region0 is Valid and running because Region1 was
not attempted
                   {
                          rgn1Vrfy = RegionUpdate82(1);
                          if (rgn1Vrfy == 1)
                          {
                                 rgn0Vrfy = RegionUpdate82(0);
                                 if (rgn0Vrfy == 1)
                                 {
                                        i2cFWUpdateDone = true;
                                 }
                                 else
                                 {
                                        UARTprintf("\nFAIL: Region 0 FW Update attempted and
Failed. Region 1 was successful\n");
                                        break;
                                 }
                          }
                          else
                          {
                                 UARTprintf("\nFAIL: Region 1 FW Update attempted and Failed. Region
0 never attempted\n");
                                 break;
                          }
                   }
                   //Region1 is running because both Regions were attempted and Region1 has no
Warning flags
                   else if ((BootFlags82.Region1 == 1) && (BootFlags82.Region0 == 1) &&\
                          ((BootFlags82.Region1CrcFail == 0) && (BootFlags82.Region1FlashErr == 0)
&& (BootFlags82.Region1Invalid == 0)))
                   {
                          //Update & verify Region0, then repeat for Region1
                          rgn0Vrfy = RegionUpdate82(0);
                          if (rgn0Vrfy == 1)
                          {
                                 rgn1Vrfy = RegionUpdate82(1);
                                 if (rgn1Vrfy == 1)
                                 {
                                        i2cFWUpdateDone = true;
                                 }
                                 else
                                 {
                                        UARTprintf("\nFAIL: Region 1 FW Update attempted and
Failed. Region 0 was successful\n");
                                        break;
                                 }
                          }
                          else
                          {
                                 UARTprintf("\nFAIL: Region 0 FW Update attempted and Failed. Region
1 never attempted\n");
```

```
                                        break;
                                }
                        }

                }while((i2cFWUpdateDone==false) && (!((readError == true) || (writeError == true) ||
   (fourCCerror == true))));



                // Validate the new FW Image is running after a HW Reset
                fourCCerror = fourCC_Command(GAID, (uint8_t *)&i2cDataClr);


                ReadIICRegister(I2C1_BASE, DefAddr1, REG_BOOT_FLAGS, lenBOOT_FLAGS, (uint8_t
*)&BootFlags82);
#ifdef UART_Stream_ON
     UARTprintf("BootOk (Bit 0): %d \n", BootFlags82.BootOk);
#endif

                // Read Version # of NEW FW image running after FW Update
                ReadIICRegister(I2C1_BASE, DefAddr1, REG_VERSION, lenVERSION, (uint8_t *)&FWVersion82);
#ifdef UART_Stream_ON
     UARTprintf("NEW FW Version # (0xDD.MM.mm.BB) = 0x%02x.%02x.%02x.%02x.\n", \
FWVersion82.FW_VERSION_B3, FWVersion82.FW_VERSION_B2, FWVersion82.FW_VERSION_B1, \
FWVersion82.FW_VERSION_B0);
#endif

     }
     else
     {
                    UARTprintf("\nBootOK was False before FW Update attempted. FW is not valid\n");
     }

     return i2cFWUpdateDone;

}
```

Minor modifications apply to the previous code-snippet if the application is being developed for TPS65982D – Details of these modifications follow:

Before

```
    …
    const uint32_t sectorCount = 17;
    …
    …
```

After

```
    …
    const uint32_t sectorCount = 3;
    …
    …
```

Before

```
    …
    //64kB+4kB = 65536Bytes + 4094Bytes = 69632/64 = 1088 times to write to DATA1
    for (count32bit = 0; count32bit < 1088; count32bit++)
    {
       …
       …
```

After

```
    …
    //8kB+4kB = 8192Bytes + 4094Bytes = 12288/64 = 192 times to write to DATA1
    for (count32bit = 0; count32bit < 192; count32bit++)
    {
       …
       …
```

Before

```
    …
    if (BootFlags82.BootOk == 1)
    {
        …
        …
```

After

```
    …
    if(!BootFlags82.PatchHeaderErr)
    {
        …
        …
```

Before

```
     …
#ifdef UART_Stream_ON
    UARTprintf("BootOk (Bit 0): %d \n", BootFlags82.BootOk);
    …
    …
```

After

```
     …
#ifdef UART_Stream_ON
    UARTprintf("PatchHeaderErr(Bit 0): %d \n", BootFlags82.PatchHeaderErr);
    …
    …
```

Note: **The changes listed in this box are applicable to all references of the above 'print' statement in the sample code-snippet**

4.3 Code From *“i2cFwUpdate82.h”* Header file

The following is the full code from the *“i2cFwUpdate82.h”* file, which is used to allow the *“main.c”* file to call the functions in the *“i2cFwUpdate82.c”* file.

```
#ifndef __I2CFWUPDATE82_H__
#define __I2CFWUPDATE82_H__

#include <stdbool.h>
#include <stdint.h>

extern bool FWUpdate82(void);

#endif //__I2CFWUPDATE82_H__
```

4.4 Excerpt of Code From *“i2cHandlerHostTo82.c”* C File

The following code is an excerpt from the *“i2cHandlerHostTo82.c”* file. Code not required to complete the FW update procedure, such as code that is only relevant specifically to the Tiva-C processor, has been redacted.

```
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
…

//I2C FW Update files - Start
#include "i2cHandlerHostTo82.h"
#include "hostIF82.h"
…
//I2C FW Update files - End

bool ReadIICRegister(uint32_t i2cPeripheral,uint8_t i2cSlaveAddress, uint8_t registerAddress, \
                    uint8_t readLength, uint8_t* i2cData)
{
    tI2cMsg i2cMessage;
    uint32_t result,i;
            bool error = false;

    // Send I2C Command to read which event we have received
    i2cMessage.bAllAce = false;
    i2cMessage.bRead = true;
    i2cMessage.includeLengthByte = true;
    i2cMessage.host = i2cPeripheral;
    i2cMessage.speed = 100;
    i2cMessage.targetAddress = i2cSlaveAddress;
    i2cMessage.registerAddress = registerAddress;
    i2cMessage.ucDataSize = readLength;
    result = HostCommandSend(&i2cMessage);
    if (result != 0)
    {
                           error = true;
    }
    for (i = 0; i < i2cMessage.ucDataSize; i++)
        i2cData[i] = i2cMessage.ucData[i];
    return error;
}

bool WriteIICRegister(uint32_t i2cPeripheral,uint8_t i2cSlaveAddress, uint8_t registerAddress, \
                    uint8_t writeLength, uint8_t* i2cData)
{
    tI2cMsg i2cMessage;
    uint32_t result,i;
            bool error = false;

    // Send I2C Command to read which event we have received
    i2cMessage.bAllAce = false;
    i2cMessage.bRead = false;
    i2cMessage.includeLengthByte = true;
    i2cMessage.host = i2cPeripheral;
    i2cMessage.speed = 400;
    i2cMessage.targetAddress = i2cSlaveAddress;
    i2cMessage.registerAddress = registerAddress;
    i2cMessage.ucDataSize = writeLength;
    for (i = 0; i < i2cMessage.ucDataSize; i++)
        i2cMessage.ucData[i] = i2cData[i];
    result = HostCommandSend(&i2cMessage);
    if (result != 0)
    {
                           error = true;
    }
    return error;
}

bool fourCC_Command(uint32_t fourCC, uint8_t* dataData)
{
     uint8_t fourCCData[4];
     bool error = false;
     bool writeError = false;
     uint32_t event = 0xFFFFFFFF;
     uint8_t rtnCMD[4];
    uint8_t i, rtnDATA[4];
    uint8_t i2cDataDataOUT[4];
    bool readError = false;


    DelayInMilliseconds(40); //Wait for 40ms


    switch (fourCC)
    {
            case FLrr:
            {

                    //Write 'FLrr' Input DATA to DATA1
                    writeError = WriteIICRegister(I2C1_BASE, DefAddr1, REG_DATA1, 4, dataData);
                    // Send 'FLrr' CMD to CMD1
                    fourCCData[0] = 'F';
                    fourCCData[1] = 'L';
                    fourCCData[2] = 'r';
                    fourCCData[3] = 'r';
                    writeError = WriteIICRegister(I2C1_BASE, 0x38, 0x08, 4, fourCCData);
                    //Handle 'FLrr' Output DATA by reading from DATA1
                    break;
            } //FLrr
            case FLem:
            {
                    //Handle 'FLad' Input DATA and write to DATA1
                    writeError = WriteIICRegister(I2C1_BASE, DefAddr1, REG_DATA1, 8, dataData);
                    // Send 'FLem' CMD
                    fourCCData[0] = 'F';
                    fourCCData[1] = 'L';
                    fourCCData[2] = 'e';
                    fourCCData[3] = 'm';
                    writeError = WriteIICRegister(I2C1_BASE, 0x38, 0x08, 4, fourCCData);
                    break;
            } //FLem
            case FLad:
            {
                    //Handle 'FLad' Input DATA and write to DATA1
                    writeError = WriteIICRegister(I2C1_BASE, DefAddr1, REG_DATA1, 4, dataData);
                    // Send 'FLad' CMD
                    fourCCData[0] = 'F';
                    fourCCData[1] = 'L';
                    fourCCData[2] = 'a';
                    fourCCData[3] = 'd';
                    writeError = WriteIICRegister(I2C1_BASE, 0x38, 0x08, 4, fourCCData);
                    break;
            } //FLad
            case FLwd:
            {
                    //Handle 'FLwd' Input DATA and write to DATA1
                    writeError = WriteIICRegister(I2C1_BASE, DefAddr1, REG_DATA1, 64, dataData);
                    DelayInMilliseconds(50); //Wait for 50ms for FW to Process 'FLwd' data
                    //Send 'FLwd' CMD
                    fourCCData[0] = 'F';
                    fourCCData[1] = 'L';
                    fourCCData[2] = 'w';
                    fourCCData[3] = 'd';
                    writeError = WriteIICRegister(I2C1_BASE, 0x38, 0x08, 4, fourCCData);
                    break;
            } //FLwd
            case FLvy:
            {
                    //Handle 'FLvy' Input DATA and write to DATA1
                    writeError = WriteIICRegister(I2C1_BASE, DefAddr1, REG_DATA1, 4, dataData);
                    // Send 'FLvy' CMD
                    fourCCData[0] = 'F';
                    fourCCData[1] = 'L';
                    fourCCData[2] = 'v';
```

```
                        fourCCData[3] = 'y';
                        writeError = WriteIICRegister(I2C1_BASE, 0x38, 0x08, 4, fourCCData);

                        //Handle 'FLvy' Output DATA by reading from DATA1
                break;
                } //FLvy
                case GAID:
                {
                        //Write 'GAID' Input DATA (Null) to DATA1
                        writeError = WriteIICRegister(I2C1_BASE, DefAddr1, REG_DATA1, 4, dataData);
                        //***Send 'GAID' CMD with Null input DATA
                        fourCCData[0] = 'G';
                        fourCCData[1] = 'A';
                        fourCCData[2] = 'I';
                        fourCCData[3] = 'D';
                        writeError = WriteIICRegister(I2C1_BASE, 0x38, 0x08, 4, fourCCData);
                        DelayInMilliseconds(2000); //Wait for 2s for FW to re-load
                        break;
                } //GAID
…
        }
        //Read Command Register
        do
        {
                event = 0;
                readError = ReadIICRegister(I2C1_BASE, 0x38, 0x08, 4, rtnCMD);

                if (readError != false)
                {
                        error = true;
                        return error;
                }
                else
                {
                        for(i = 0; i < 4; i++)
                                event |= rtnCMD[i] << (i*8);
                }
        }while(!((event == 0) || (event == nCMD)));

        //Read Data Register
        readError = ReadIICRegister(I2C1_BASE, 0x38, 0x09, 4, rtnDATA);
        if (readError != false)
        {
                error = true;
                return error;
        }
        else
        {
                for(i = 0; i < 4; i++)
                i2cDataDataOUT[i] = rtnDATA[i];
        }
        i2cDataDataOUT[0] &= 0x03;
        if ((i2cDataDataOUT[0] == 1) || (i2cDataDataOUT[0] == 3))
                error = true;
        else if (i2cDataDataOUT[0] == 0)
                error = false;
        else
                error = true;

        if ((writeError == true) || (readError == true))
                error = true;
        else
                error = false;

                return error;
}
```

4.5 Code From “i2cHandlerHostTo82.h” Header File

The following is the full code from the “ *i2cHandlerHostTo82.h*” file, which is used to allow the *“i2cFwUpdate82.c”* file to call the functions in the *“i2cHandlerHostTo82.c”* file.

```
#ifndef __TIVAHOSTI2CHANDLER_H__
#define __TIVAHOSTI2CHANDLER_H__

#include <stdbool.h>
#include <stdint.h>

extern bool WriteIICRegister(uint32_t i2cPeripheral,uint8_t i2cSlaveAddress, \
uint8_t registerAddress, uint8_t writeLength, uint8_t* i2cData);

extern bool ReadIICRegister(uint32_t i2cPeripheral,uint8_t i2cSlaveAddress, \
uint8_t registerAddress, uint8_t readLength, uint8_t* i2cData);

extern bool fourCC_Command(uint32_t fourCC, uint8_t* dataData);

#endif //__TIVAHOSTI2CHANDLER_H__
```

4.6 Excerpt of Code From “main.c” C File

The following code is an excerpt from the *“main.c”* file. Code not required for the FW update procedure has been redacted. In *“main.c”*, the embedded processor’s I/O pins must be initialized correctly for I^{2}C communication to work properly with the TPS6598x.

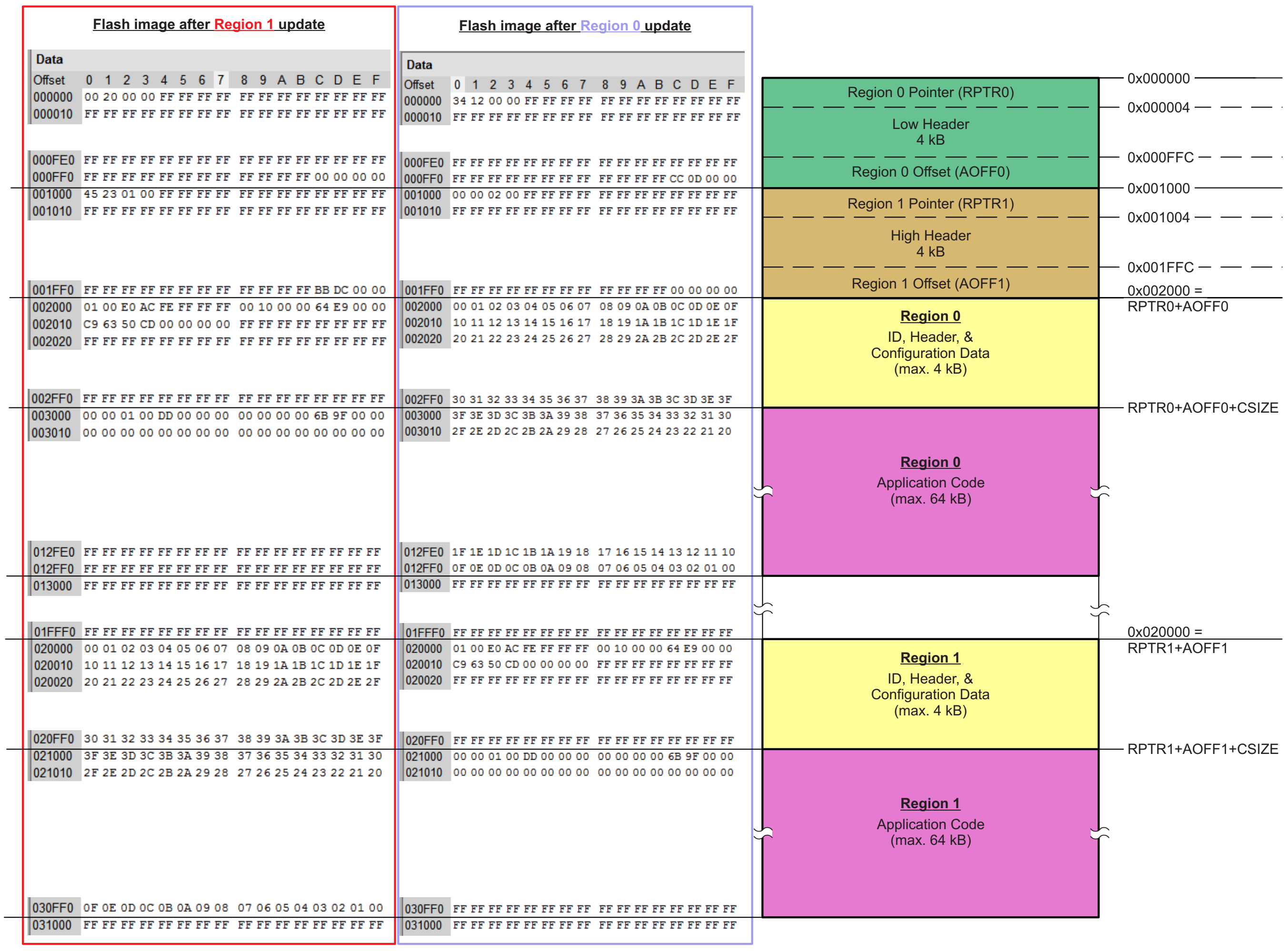
```
#include <stdbool.h>
#include <stdint.h>
…
//I2C FW Update files - Start
#include "i2cFwUpdate82.h"
//I2C FW Update files - End

bool fwUpdateComplete = false;

int main()
{
    HostI2c1Init();
    HostI2c2Init();
    while (1)
    {
            fwUpdateComplete = FWUpdate82(); //I2C FW Update code
            if (fwUpdateComplete == true)
            {
                    UARTprintf("Program exited successfully. FW Update complete!\n");
                    DelayInMilliseconds(100); //Wait for 100ms for UART print
                    break;
            }
            else
            {
                    UARTprintf("Program did not exit cleanly. FW Update failed. Try turning UART\
Stream ON to debug\n");
                    DelayInMilliseconds(100); //Wait for 100ms for UART print
                    break;
            }
    }
}
```

5 Summary

The results of the example code provided are shown in Figure 4. In the final implementation of the FW update procedure required in the field, the EC must receive the new binary file data from the Operating System (OS) or application processor of the product. This method varies from one system to another and is outside the scope of this application note.

In this example, instead of real data from the *“low-region.bin”* binary file, a function is used to generate fake data and place it in the redundant region of the Flash memory. The maximum file size of 68kB is used, so this function executes exactly 1024 times, but in the final code this function executes fewer times using the maximum length of the .bin file to determine when the ‘FLwd’ while() loop exits. The *regionUpdate82()* function can only execute twice if the ‘FLvy’ 4CC command returns a ‘0’ in the least-significant bit of the DATA/12 register; as a result, using fake data always runs once and exits with an error flag that is ‘true’.

Successful implementation of this code using real data from a *“low-region.bin”* file will pass the ‘FLvy’ test, execute the *regionUpdate82()* function twice, and exit cleanly with an error flag of ‘false’. Reading back the FW image from the SPI flash directly is recommended to manually verify the results the first time the code is tested, even if the program exits cleanly.


Flash image after Region 1 update

```
Data
Offset  0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
000000  00 20 00 00 FF FF FF FF  FF FF FF FF FF FF FF FF
000010  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF

000FE0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
000FF0  FF FF FF FF FF FF FF FF  FF FF FF FF 00 00 00 00
001000  45 23 01 00 FF FF FF FF  FF FF FF FF FF FF FF FF
001010  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF

001FF0  FF FF FF FF FF FF FF FF  FF FF FF FF BB DC 00 00
002000  01 00 E0 AC FE FF FF FF  00 10 00 00 64 E9 00 00
002010  C9 63 50 CD 00 00 00 00  FF FF FF FF FF FF FF FF
002020  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF

002FF0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
003000  00 00 01 00 DD 00 00 00  00 00 00 00 6B 9F 00 00
003010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00

012FE0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
012FF0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
013000  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF

01FFF0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
020000  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F
020010  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F
020020  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F

020FF0  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F
021000  3F 3E 3D 3C 3B 3A 39 38  37 36 35 34 33 32 31 30
021010  2F 2E 2D 2C 2B 2A 29 28  27 26 25 24 23 22 21 20

030FF0  0F 0E 0D 0C 0B 0A 09 08  07 06 05 04 03 02 01 00
031000  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
```

Flash image after Region 0 update

```
Data
Offset  0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
000000  34 12 00 00 FF FF FF FF  FF FF FF FF FF FF FF FF
000010  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF

000FE0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
000FF0  FF FF FF FF FF FF FF FF  FF FF FF FF CC 0D 00 00
001000  00 00 02 00 FF FF FF FF  FF FF FF FF FF FF FF FF
001010  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF

001FF0  FF FF FF FF FF FF FF FF  FF FF FF FF 00 00 00 00
002000  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F
002010  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F
002020  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F

002FF0  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F
003000  3F 3E 3D 3C 3B 3A 39 38  37 36 35 34 33 32 31 30
003010  2F 2E 2D 2C 2B 2A 29 28  27 26 25 24 23 22 21 20

012FE0  1F 1E 1D 1C 1B 1A 19 18  17 16 15 14 13 12 11 10
012FF0  0F 0E 0D 0C 0B 0A 09 08  07 06 05 04 03 02 01 00
013000  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF

01FFF0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
020000  01 00 E0 AC FE FF FF FF  00 10 00 00 64 E9 00 00
020010  C9 63 50 CD 00 00 00 00  FF FF FF FF FF FF FF FF
020020  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF

020FF0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
021000  00 00 01 00 DD 00 00 00  00 00 00 00 6B 9F 00 00
021010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00

030FF0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
031000  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
```

Figure 4. Results of TPS6598x FW Update C-Code

6 References

- TPS65982 data sheet (SLVSD02)
- *TPS6598x and TPS65986 Firmware User's Guide* (SLVUAH7)
- TPS6598X Configuration Tool
- TPS6598X Utilities Tool
- TPS65986 data sheet (SLVSD13)
- TPS65981 data sheet (SLVSDC2)

Revision History

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

Changes from Original (June 2016) to A Revision **Page**

- Changed device name globally from TPS65982 to TPS6598x. 1
- Included language for software and firmware patches. 1
- Added *TPS65982D SPI Flash Memory Organization of Region 0* image. 4
- Added TPS65982D modifications for data structures in the *Excerpt of Code From "hostIF82.h" Header File*section 10
- Added TPS65982D modifications for data structures in the *Excerpt of Code From "i2cFwUpdate82.c" C File*section. 16