Application Report
SLVAE21A–July 2018–Revised April 2019

TPS65987 and TPS65988 SPI Flash Firmware Update Over I^2C

ABSTRACT

The TPS65987D(DX) and TPS65988(DX) is a fully-integrated USB power delivery (PD) management device providing cable plug and orientation detection for USB Type-C and PD plug or receptacle. Each Type-C port that is controlled by the device is functionally identical and supports the full range of the USB Type-C and PD standards. The device hosts the boot and application code on its internal ROM, and can accept patch bundles (containing the application configuration and/or code patch) from an external host over I2C or serial flash. The application code includes a set of ASCII commands that enables the host to either download the patch bundle onto the device's SRAM or program the content onto the attached flash memory using the host interface of the device

Contents

List of Figures

List of Tables

Trademarks

1 Purpose and Scope

This document details the flash-update process of the device to enable the external hosts program and the patch bundles onto the attached external flash memory using the host interface of the device. Code snippets, wherever applicable, are presented in the document to demonstrate this process.

The document also details the boot flow of the device and lists the various default configurations that define the operation of the device for the cases where the patch bundle is not loaded successfully because of the unavailability of an external host or serial flash, or the corruption of the patch bundle before or during the download process.

NOTE: The instructions and code snippets listed in this document are applicable only to the DX (Ex: DH) variant of this device.

2 Boot Code

The boot code of the device has two primary functions:

1. Device Initialization
2. Configuration and Patch Loading

When power is applied to the device through VIN_3V3 or VBUS, LDO_3V3 is enabled and a power-on-reset (POR) signal is issued. The digital core receives this reset signal and loads the boot code from the internal memory, and then begins initializing the device settings. This initialization includes enabling and resetting internal registers, loading initial values and configuring the I2C addresses of the device. The various BUSPOWERZ configurations, and the detailed boot sequence of the device is detailed in Appendix A.

3 Patch Bundle

A patch bundle is used for two purposes:

1. Providing code patch to replace functions inside the application ROM
2. Providing configuration for the device

As described in the previous section, the patch bundle can be loaded from an external flash memory or a host using the device's I2C host interface. The patch bundle shall be from a single source. If no device configurations are available, the boot code uses one of the factory set device configurations.

4 Flash Update

4.1 Overview

If an external flash is detected, the boot code first attempts to read the patch bundle from the low region of the attached flash memory. If any part of the read process yields invalid data, the device aborts the low region read and attempts to read from the high region. If both regions contain an invalid patch bundle, the boot firmware proceeds to check for the existence of an external host.

The device is designed to power the external flash from LDO_3V3 to support dead-battery or no-battery conditions, and therefore pull-up resistors used for the flash memory must be tied to LDO_3V3.

The flash memory IC must support 12 MHz SPI clock frequency. The size of the flash must be at least 64 kB. The SPI master interface of the device supports SPI Mode 0. For Mode 0, data delay is defined such that data is output on the same cycle as chip select (SPI_SS pin) becomes active. The chip select polarity is active-low. The clock phase is defined such that data (on the SPI_MISO and SPI_MOSI pins) is shifted out on the falling edge of the clock (SPI_CLK pin) and data is sampled on the rising edge of the clock. The clock polarity for chip select is defined such that when data is not being transferred the SPI_CLK pin is held (or idling) low. The minimum erasable sector size of the flash must be 4 kB. The W25X05CL or similar is recommended.

4.2 Flash Memory Organization

The patch-bundle of the device can be contained in the memory of the attached SPI Flash IC. The organization of this data in memory is explained in this section using variables and relative locations. Memory organization might differ between the various variants of device, and the application developers shall account for these differences when developing the flash-update applications for the device.

In this application note, the flash memory organization is explained based on the assumption that the SPI Flash is dedicated to the device and not shared with other ICs of the system. For redundancy, the patch-bundle is copied into two regions, called Region-0 (low region) and Region-1 (high region). If the full memory of the SPI flash is read directly, it will contain region-headers and regions, which will simply be referred to as a "flash.bin" file. A depiction of the full "flash.bin" flash memory organization for the device is shown in Figure 1.

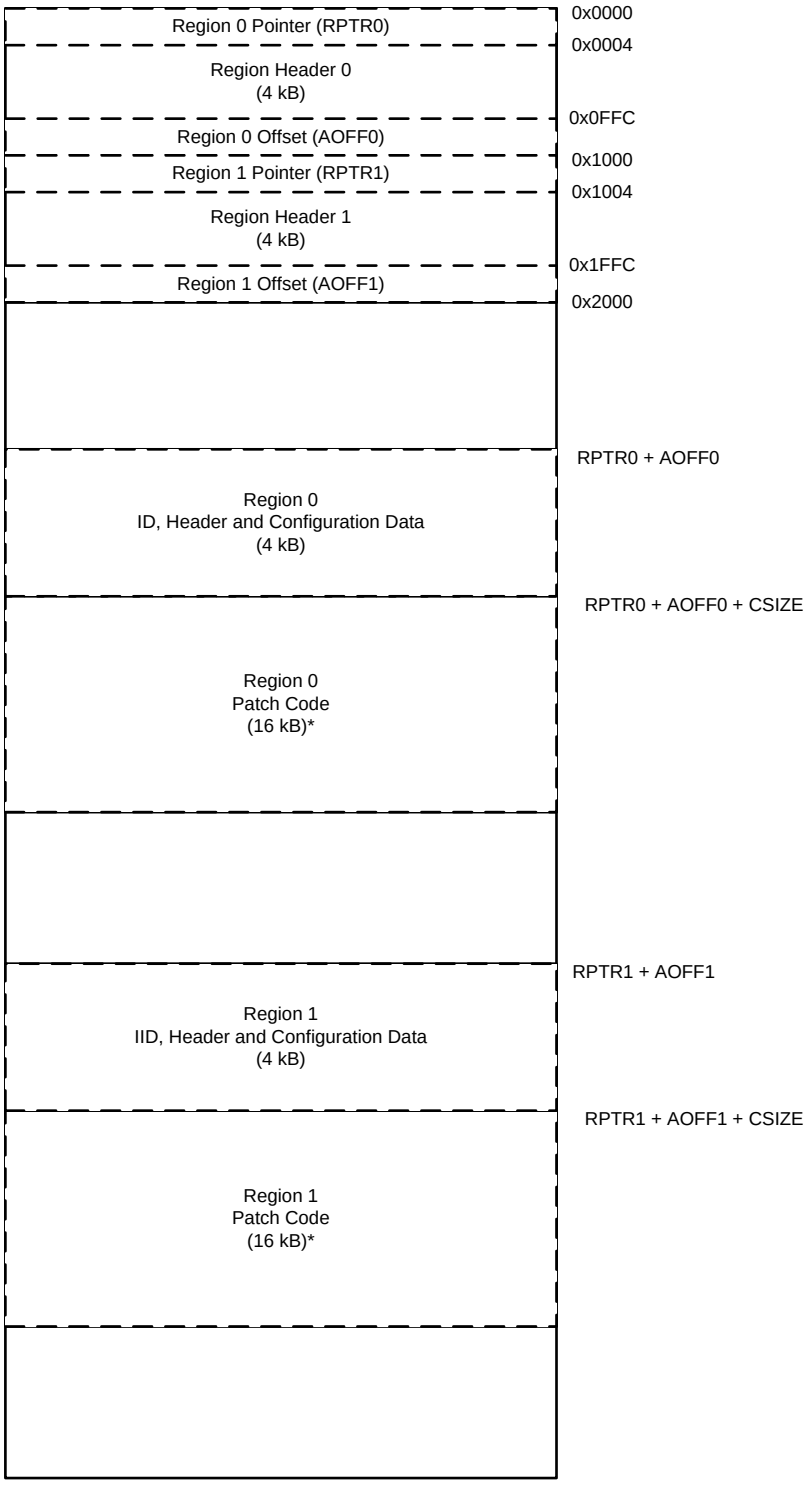

Figure 1. Flash Memory Organization of Region-0 and Region-1

The external SPI-Flash shall be programmed with a full 'flash.bin' the very first time the platform is powered up so that the region headers are set up correctly. The subsequent flash-updates can be executed by the external hosts by following the sequence detailed in subsequent sections. A full 'flash.bin' can be generated by the device's 'Application Customization Tool'. When generating a full 'flash.bin' using the GUI, 'Region offsets' for Region-0 and Region-1 shall be appropriately set, such that there is ample space between the regions to accomodate for the increased sizes of the future patch-bundles. For the Dx (Ex: DH) variant of this device, TI recommends offset 0x2000 and 0x9000 for Region-0 and Region-1 respectively.

The device's 'Application Customization Tool' can also generate a 'Low Region' binary, and this file shall be used by the external hosts for doing the flash updates.

4.3 Commands

The flash-update process shall utilize the 4CC ASCII commands listed in Table 1.

Table 1. (1)

Name of 4CC Command	ASCII	Input DataX Length (In Bytes)	Output DataX Length (In Bytes)	Description
Flash Load Read Regions	FLrr(1)	1	4	The 'FLrr' Command loads the address of the flash memory for the selected region into Output DataX
Flash Memory Erase	FLem(1)	4	1	The 'FLem' Command erases the number of segments specified
Flash Memory Write Start Address	FLad(1)	1	1	The 'FLad' Command sets start address in preparation the flash write
Flash Memory Write	FLwd(1)	64	1	The 'FLwd' Command writes data beginning at the flash start address defined by the 'FLad' Command. The address is auto-incremented
Flash Memory Verify	FLvy(1)	1	1	The 'FLvy' Command verifies if the patch/configuration is valid
Cold reset request	GAID(1)	0	0	The 'GAID Command causes a cold restart of the PD Controller processor. 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

(1) Details in HI-TRM (SLVUBH2)

To execute a 4CC command, the host application shall follow the below sequence:

1. If the 4CC command requires an input, the application shall first write the input data into the DataX (0x09 or 0x11) register.
2. The application shall then write the 4CC command characters into the corresponding CmdX (0x08 or 0x10) register.
3. The application shall wait until the four byte content of CmdX register reads the following – Applications can either poll, or set and use the 'CmdXComplete' event:
   - '0x00' indicating that the command is successfully executed.

- or, 'CMD' indicating that the command's execution has failed.

If the command is successfully executed, the application shall proceed to read the 'n' byte content of the DataX register which will contain the output. Refer the device's Host Interface TRM for more details on the 4CC commands SLVUBH2.

4.4 Flash Update Execution Flow

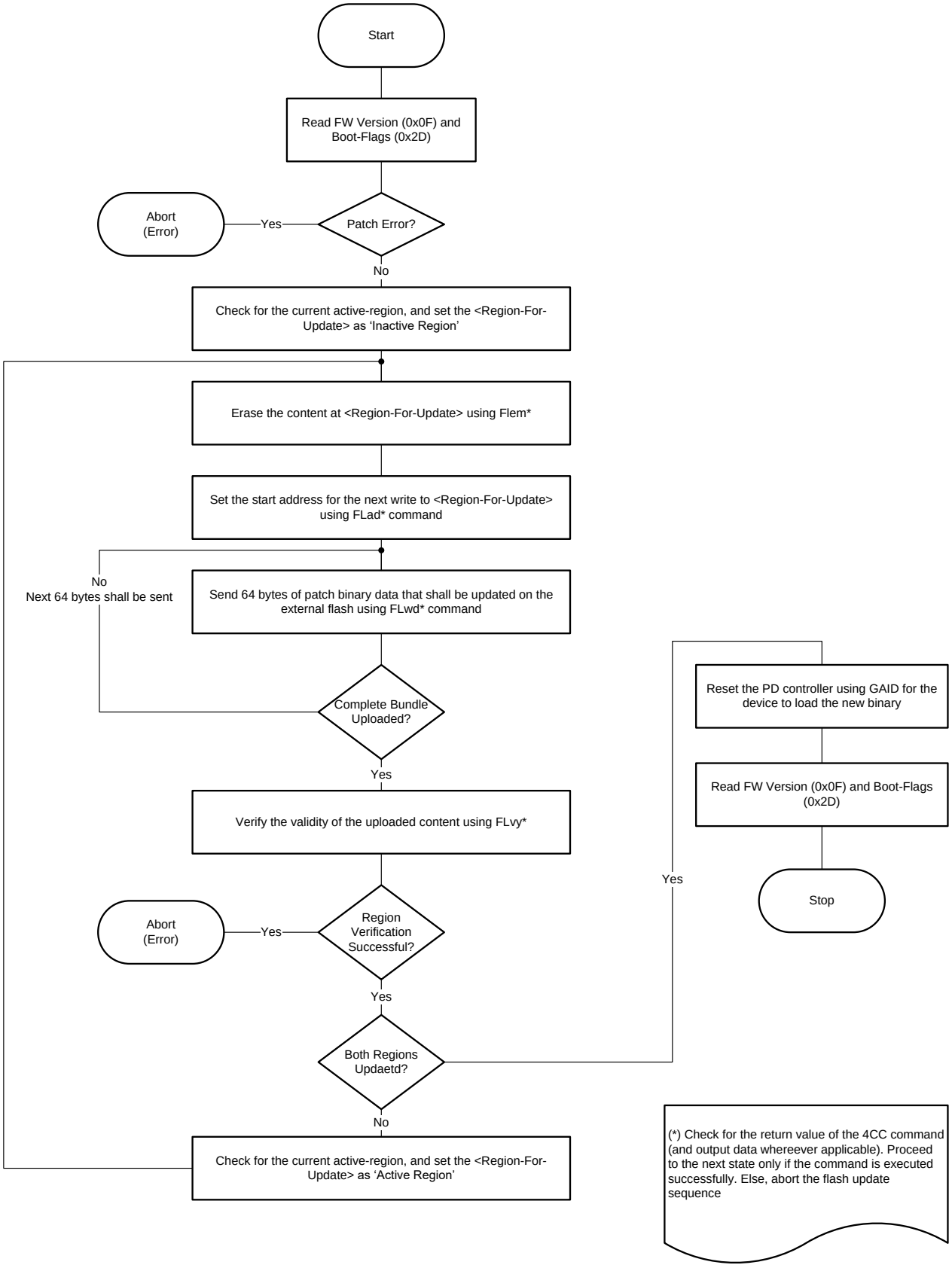

Figure 2. Flow Diagram of FW Update Process

1. The flash update process can be initiated by the host when the device is fully functioning in the application firmware, and the port shall be disabled during the update process.
2. The host shall first read the Boot-Flags (0x2D) register of the device to know what the current active region is – The active region is the region on external flash from where the device successfully loaded the patch bundle during the current boot, and it could either be Region-0 or Region-1.
3. The host shall first attempt to update the contents at the inactive region. Only after the inactive region is successfully updated, the host shall attempt to update the contents at the active region. This process ensures that a redundant and valid copy of the patch bundle is available on the external flash at any point in time for a successful boot of the device, thereby making the flash update process 'fail-safe'.
4. The host shall implement the below sequence to update the patch-bundle:
   - Using 'FLrr' command, the host shall query the device for the address of the region on external flash that is to be updated.
   - The host shall then command the device to erase 'n' number of 4kB segments starting from the above address using 'FLem' command, where 'n' is the total number of 4kB segments required to hold the maximum size of the patch-bundle.
   - The host shall then set the start address for the next write using 'FLad' command, and start sending the patch-bundle 64 bytes at a time using 'FLwd' command.
   - The device automatically increments the write-address after the successful execution of 'FLwd'– The host needn't set the start address for every write request.
   - After both regions are updated, the host shall cold-reset the device using 'GAID' – The device will go through its boot sequence all over again, and load the updated patch-bundle.

4.5 Example Code

```
/**/
static int32_t PreOpsForFlashUpdate()
{
    s_AppContext        *const pCtx     = &gAppCtx;

    s_TPS_bootflag      *p_bootflags    = NULL;
    s_TPS_portconfig    *p_portconfig   = NULL;

    uint8_t outdata[MAX_BUF_BSIZE]  = {0};
    uint8_t indata[MAX_BUF_BSIZE]   = {0};

    int32_t retVal  = 0;

    /*
     * Read BootFlags (0x2D) register:
     *  - Note #1: Applications shouldn't proceed w/ flash-update if the device's
     *         boot didn't succeed
     *  - Note #2: Flash-update shall be attempted on the inactive region first
     */
    retVal = ReadReg(REG_ADDR_BOOTFLAG, REG_LEN_BOOTFLAG, &outdata[0]);
    RETURN_ON_ERROR(retVal);

    /*
     * Note #1
     * Error during patch load - Don't attempt flash-update as the device wouldn't
     * be able to process the 4CC commands
     */
    p_bootflags = (s_TPS_bootflag *)&outdata[1];
    if(0 != p_bootflags->patchheadererr)
    {
        ERR_PRINT( p_bootflags->patchheadererr);
        SignalEvent(APP_EVENT_ERROR);
        retVal = 0; /* For the state-machine */
        goto error;
    }
```

```
    /*
     * Note #2
     * Region1 = 0 indicates that device didn't attempt 'Region1',
     * which implicitly means that the content at Region0 is valid/active
     */
    if(0 == p_bootflags->region1)
    {
        pCtx->active_region = REGION_0;
        pCtx->inactive_region = REGION_1;
    }
    else if ( (1 == p_bootflags->region1) && \
              (1 == p_bootflags->region0) && \
              ((0 == p_bootflags->region1crcfail) && \
               (0 == p_bootflags->region1flasherr) && \
               (0 == p_bootflags->region1invalid)) )
    {
        pCtx->active_region = REGION_1;
        pCtx->inactive_region = REGION_0;
    }

    /*
     * Keep the port disabled during the flash-update
     */
    retVal = ReadReg(REG_ADDR_PORTCONFIG, REG_LEN_PORTCONFIG, &outdata[0]);
    RETURN_ON_ERROR(retVal);
    memcpy(&indata[0], &outdata[1], REG_LEN_PORTCONFIG); /* outdata[0] holds the register length
*/
    p_portconfig = (s_TPS_portconfig *)&indata[0];
    p_portconfig->typecstatemachine = DISABLE_PORT;
    retVal = WriteReg(REG_ADDR_PORTCONFIG, REG_LEN_PORTCONFIG, &indata[0]);
    RETURN_ON_ERROR(retVal);

error:
    return retVal;
}


/**/
static int32_t StartFlashUpdate()
{
    s_AppContext    *const pCtx     = &gAppCtx;
    int32_t retVal  = 0;

    UART_PRINT("\n\rActive Region is [%d] - Region being updated is [%d]\n\r",\
                                        pCtx->active_region, pCtx->inactive_region);

    /*
     * Region-0 is currently active, hence update Region-1
     */
    retVal = UpdateAndVerifyRegion(pCtx->inactive_region);
    if(0 != retVal)
    {
        UART_PRINT("Region[%d] update failed.! Next boot will happen from Region[%d]\n\r",\
                                pCtx->inactive_region, pCtx->active_region);
        retVal = 0;
        goto error;
    }

    /*
     * Region-1 is successfully updated.
     * To maintain a redundant copy for a fail-safe flash-update, copy the same
     * content at Region-0
     */
    retVal = UpdateAndVerifyRegion(pCtx->active_region);
    if(0 != retVal)
    {
```

```
            UART_PRINT("Region[%d] update failed.! Next boot will happen from Region[%d]\n\r",\
                                    pCtx->active_region, pCtx->inactive_region);
            retVal = 0;
            goto error;
        }

    error:
        SignalEvent(APP_EVENT_END_UPDATE);
        return retVal;
    }


    /**/
    static UpdateAndVerifyRegion(uint8_t region_number)
    {
        s_TPS_flrr  flrrInData = {0};
        s_TPS_flem  flemInData = {0};
        s_TPS_flad  fladInData = {0};
        s_TPS_flvy  flvyInData = {0};

        uint8_t     outdata[MAX_BUF_BSIZE]  = {0};

        uint32_t    patchBundleSize = 0;
        uint32_t    regAddr = 0;

        int32_t     idx     = -1;
        int32_t     retVal  = -1;

        patchBundleSize = sizeof(tps6598x_lowregion_array);

        /*
         * Get the location of the region 'region_number'
         */
        flrrInData.regionnum = region_number;
        retVal = ExecCmd(FLrr, sizeof(flrrInData), (int8_t *)&flrrInData, \
                         OUTPUT_LEN_FLRR, &outdata[0]);
        RETURN_ON_ERROR(retVal);

        regAddr = (outdata[4] << 24) | (outdata[3] << 16) | \
                  (outdata[2] << 8) |  (outdata[1] << 0);

        /*
         * Erase #'numof4ksector' sectors at address 'regAddr' of the sFLASH
         *  - Note: The below snippet assumes that the total number of 4kB segments
         *          required to hold the maximum size of the patch-bundle is 4.
         *          Ensure its validity for the TPS6598x being used for your
         *          application.
         */
        flemInData.flashaddr = regAddr;
        flemInData.numof4ksector = TOTAL_4kBSECTORS_FOR_PATCH;
        retVal = ExecCmd(FLem, sizeof(flemInData), (int8_t *)&flemInData, \
                         TASK_RET_CODE_LEN, &outdata[0]);
        RETURN_ON_ERROR(retVal);

        /*
         * Set the start address for the next write
         */
        fladInData.flashaddr = regAddr;
        retVal = ExecCmd(FLad, sizeof(fladInData), (int8_t *)&fladInData, \
                         TASK_RET_CODE_LEN, &outdata[0]);
        RETURN_ON_ERROR(retVal);

        /**/
        UART_PRINT("Updating [%d] 4k chunks starting @ 0x%x \n\r", flemInData.numof4ksector, regAddr);
        for (idx = 0; idx < patchBundleSize/PATCH_BUNDLE_SIZE; idx++)
        {
```

```
            UART_PRINT(".");

            /*
             * Execute FLwd with PATCH_BUNDLE_SIZE bytes of patch-data
             * in each iteration
             */
            retVal = ExecCmd(FLwd, PATCH_BUNDLE_SIZE,\
                             (int8_t *)&tps6598x_lowregion_array[idx * PATCH_BUNDLE_SIZE],
                             TASK_RET_CODE_LEN, &outdata[0]);
            RETURN_ON_ERROR(retVal);

            /*
             * 'outdata[1]' will contain the command's return code
             */
            if(0 != outdata[1])
            {
                UART_PRINT("\n\r");
                UART_PRINT("Flash Write FAILED.!\n\r");
                retVal = -1;
                goto error;
            }
        }

        UART_PRINT("\n\r");

        /*
         * Write is through. Now verify if the content/copy is valid
         */
        flvyInData.flashaddr = regAddr;
        retVal = ExecCmd(FLvy, sizeof(flvyInData), (int8_t *)&flvyInData, \
                         TASK_RET_CODE_LEN, &outdata[0]);
        if(0 != outdata[1])
        {
            UART_PRINT("Flash Verify FAILED.!\n\r");
            retVal = -1;
            goto error;
        }

    error:
        return retVal;

    }


    /**/
    static int32_t ResetPDController()
    {
        int32_t retVal = -1;

        /*
         * Execute GAID, and wait for reset to complete
         */
        UART_PRINT("Waiting for device to reset\n\r");
        ExecCmd(GAID, 0, NULL, 0, NULL);
        Board_IF_Delay(1000);

        retVal = ReadMode();
        RETURN_ON_ERROR(retVal);

        retVal = ReadVersion();
        RETURN_ON_ERROR(retVal);

        retVal = ReadBootStatus();
        RETURN_ON_ERROR(retVal);

        return 0;
```

}

A.1 BusPowerZ Configurations

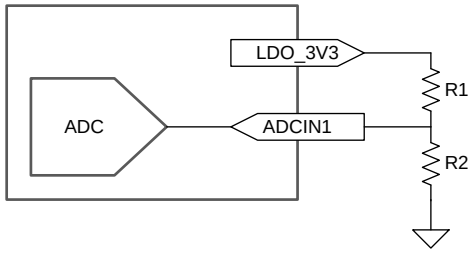

Figure 3. BUSPOWERZ

Table 2.

DIV = R2 / (R1 + R2)		SPI-MISO	Configuration	Description
DIV MIN	DIV MAX			
0	0.18	1	BP_NoResponse	Safe Configuration(1) : No power switch is enabled and the device does not start up until VIN_3V3 is present.
0.20	0.28	1	BP_WaitFor3V3_Internal	Safe Configuration(1) : The internal power switch from VBUSx to PP_HVx is enabled for the port receiving power. The device does not continue to start-up or attempt to load device configurations until VIN_3V3 is present.
0.30	0.38	1	BP_ECWait_Internal	Safe Configuration(1) : The internal power switch from VBUSx to PP_HVx is enabled for the port receiving power. The device will indefinitely wait for the EC to download the patch bundle.
0.40	0.48	1	BP_WaitFor3V3_External	Safe Configuration(1) : The external power switch from VBUSx to PP_HVx is enabled for the port receiving power. The device does not continue to start-up or attempt to load device configurations until VIN_3V3 is present

(1) PD is disabled in 'Safe Configuration', and the configuration is applied as indicated above only when the sFLASH is empty or does not host a valid patch bundle.

Table 2. (continued)

0.50	0.58	1	BP_ECWait_External	Safe Configuration[1] : The external power switch from VBUSx to PP_EXTx is enabled for the port receiving power. The device waits indefinitely for the EC to download the patch bundle.
0.60	1	1	BP_NoWait	Safe Configuration[1] : No power switch is closed and the device attempts to load the patch bundle from sFLASH.
0	0.08	0	BP_NoResponse	Configuration0: DFP only (internal switch) 5 V at 3 A source capability TBT Alternate Modes enabled DisplayPort Alternate Mode enabled
0.10	0.18	0	BP_NoResponse	Configuration1: DFP only (internal switch) 5 V at 3-A source capability TBT Alternate Modes not enabled DisplayPort Alternate Mode not enabled
0.20	0.28	0	BP_NoWait	Configuration2: UFP only (internal switch) 5 V at 0.9-A to 3-A sink capability TBT Alternate Modes not enabled DisplayPort Alternate Mode not enabled
0.30	0.38	0	BP_ECWait_Internal	The internal power switch from VBUSx to PP_HVx is enabled for the port receiving power. The device waits indefinitely for the EC to download the patch bundle.
0.40	0.48	0	BP_NoWait	Configuration3: UFP only (internal switch) 5 V to 20 V at 0.9-A to 3-A sink capability TBT Alternate Modes not enabled DisplayPort Alternate Mode not enabled
0.50	0.58	0	BP_ECWait_External	The external power switch from VBUSx to PP_EXTx is enabled for the port receiving power. The device waits indefinitely for the EC to download the patch bundle.

Table 2. (continued)

0.60	0.68	0	BP_NoWait	Configuration4: DRP only (internal source or external sink) 5 V at 0.9-A to 3-A sink capability 5 V at 3-A source capability TBT Alternate Modes not enabled DisplayPort Alternate Modes not enabled Accepts data and power role swaps, but does not initiate.
0.70	0.78	0	BP_NoWait	Reserved
0.80	0.88	0	Reserved(BP_NoResponse)	Safe: No power switch is closed and the device attempts to load the patch bundle from sFLASH.
0.90	1	0	BP_NoWait	Configuration5: DRP only (internal source or external sink) 5 V to 20 V at 0.9-A to 3-A sink capability 5 V at 3-A source capability TBT Alternate Modes not enabled DisplayPort Alternate Modes not enabled Accepts data and power role swaps, but does not initiate.

A.2 Boot Flow

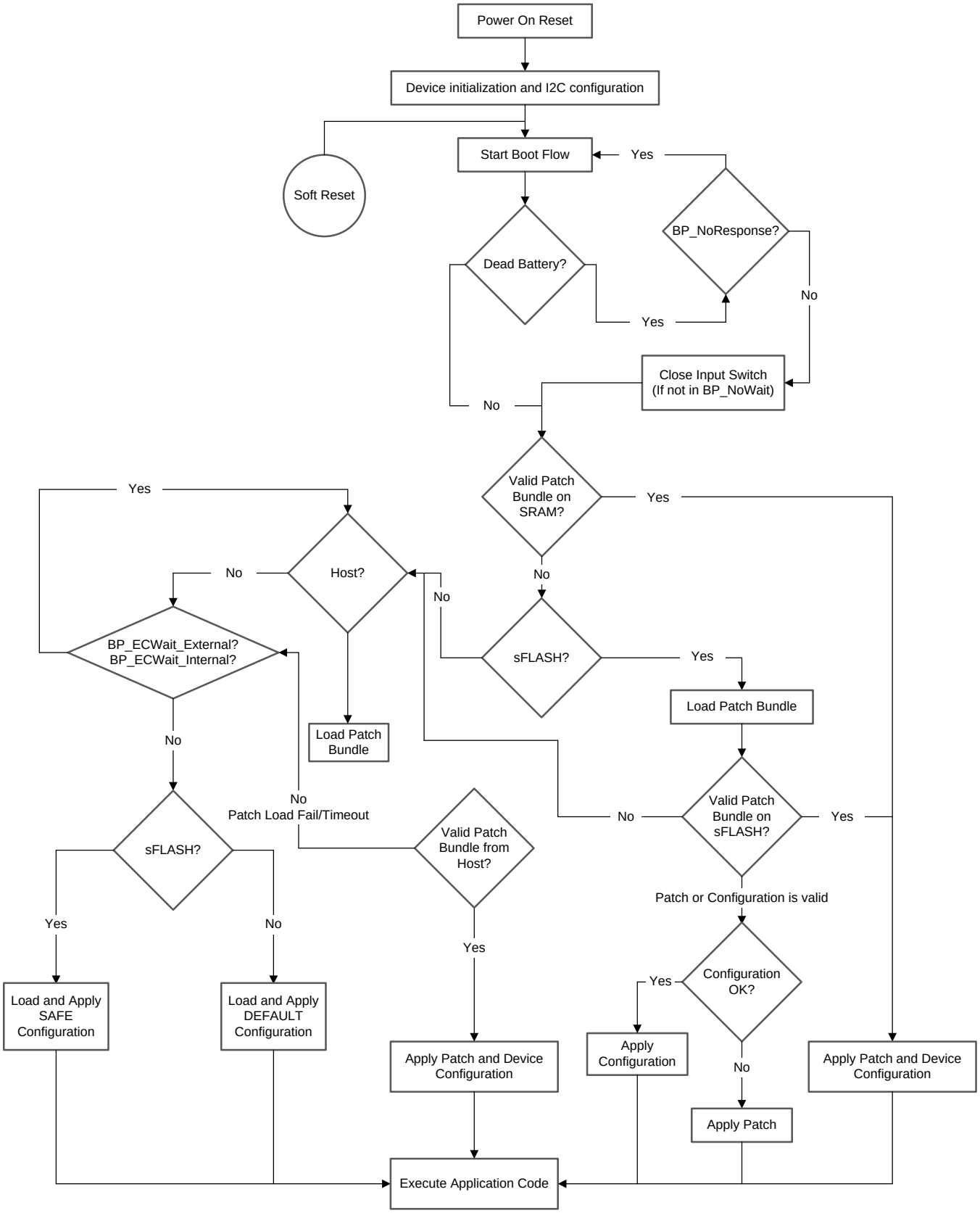

Figure 4. Boot Flow

1. When the initial device configuration is complete, the boot code samples the BUSPOWERZ pin to determine the start-up behavior of the device. The device can operate with or without an external flash. On the systems where the external flash is not available, the code patch and device configuration is downloaded to the SRAM of the device from the external host over I2C or internal ROM, depending on the BUSPOWERZ configuration.
2. The boot code then determines if the device is booting under dead-battery condition (for example, VIN_3V3 invalid and VBUS valid). If the device is booting under dead-battery condition, the deadbattery flag is set and the device continues through the boot flow depending on the BUSPOWERZ configuration. If BUSPOWERZ configuration is set to BP_NoResponse, the device does not start up until VIN_3V3 is available.
3. The boot code checks the system sequentially for an available patch bundle. It first checks to see if the SRAM has a valid patch bundle and, if available, the patch bundle is applied and the device transitions to APP mode.
4. If the content on SRAM is not valid, the boot code checks if an external flash is present on the system. If it is present, the boot code checks for a valid patch bundle and, if available, the patch bundle is loaded into the device SRAM and the device transitions to the APP mode.
5. If no patch bundle is available, or no external flash is present, the boot code proceeds to check for the presence of a host. Depending on the BUSPOWERZ configuration, the boot code waits indefinitely for the host to provide patch or proceeds with the default configuration as listed in Table 2.
6. If no host is present or a patch bundle is not available, then it is assumed that no patch bundle is to be loaded. At this point, the device uses the code that exists in the current ROM version.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265