Application Report
SLVA972A–May 2018–Revised April 2019

TPS65987 and TPS65988 SPI Less Host Programming Over I²C

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.

Contents

List of Figures

List of Tables

Trademarks

All trademarks are the property of their respective owners.

1 Purpose and Scope

This document details the boot flow of the device, and also includes the instructions for the external hosts to download the patch bundles to the SRAM of the device over I^2C. Code snippets, wherever applicable, are included in the document to demonstrate the patch download process.

The document also 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 because of the corruption of the patch bundle before or during the download process.

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

2 Boot Code

The boot code of the device has two primary functions:

- Device initialization
- 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, then begins initializing the device settings. This initialization includes enabling and resetting internal registers, loading initial values, and configuring the I^2C addresses of the device. Table 1 lists the various BUSPOWERZ configurations. Section 4.1 details the boot sequence.

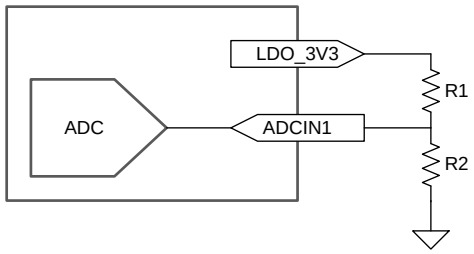


Figure 1. BUSPOWERZ

Table 1. BUSPOWERZ Configurations

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
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.
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.

(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 1. BUSPOWERZ Configurations (continued)

DIV = R2 / (R1 + R2)		SPI-MISO	Configuration	Description
DIV MIN	DIV MAX			
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.

3 Patch Bundle

A patch bundle is used for two purposes:

- Providing code patch to replace functions inside the application ROM
- Providing configuration for the device

As detailed in Section 2, the patch bundle can be loaded from an external flash memory or a host using the I^2C host interface of the device. The patch bundle must be from a single source. If no device configurations are available, the boot code uses one of the factory set device configurations.

3.1 Patch Bundle from Host over I^2C

The 4CC commands and sequence for uploading a patch bundle to the device from an external host over I^2C is listed in Section 3.1.1.

3.1.1 Commands

- PTCs: The PTCs task starts the patch loading sequence. This command initializes the firmware in preparation for a patch bundle load sequence and is used to transfer the patch header structure from the host to the PD controller.
- PTCd: After a successful PTCs command, patch binary data can be transferred 64 bytes at a time using the PTCd task command.
- PTCc: The PTCc task ends the patch loading sequence. This command shall be sent after the patch data has been completely transferred through the PTCd command. This command initiates the checksum check on the binary patch data that has been transferred, and if the checksum is successful, the patch_init function contained within the patch will be completed. If this command is sent prior to a PTCs start command, it indicates to the PD controller that no patch is available and bypasses the patch process.
- PTCq: The PTCq task can query the status of the patch process.
- PTCr: This task resets the patch firmware to the no patch state.

3.1.2 Execution Flow

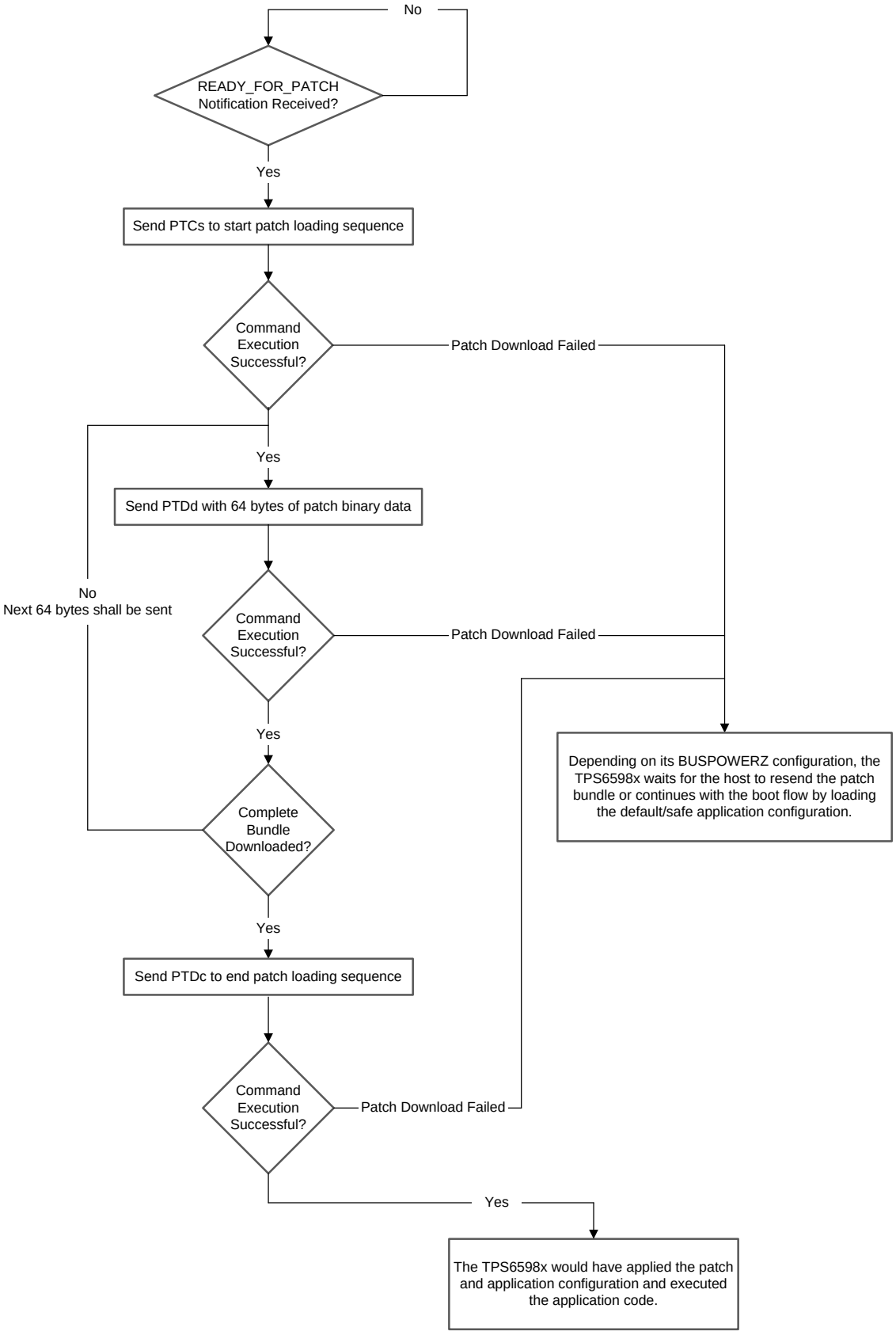


Figure 2. Code Flow

1. The device generates an I²C event, READY_FOR_PATCH, indicating that it is ready for patch. The host shall start the patch download process after receiving this notification from the device.
2. The host shall start the patch download process after receiving this notification from the device. The host shall then respond with a PTCs message within a finite time (which is one-time OTP configuration). If the device does not get any response from the host within the configured time, the PD controller assumes that no host is available and proceeds to the next step in the boot process. Depending on the BUSPOWERZ configuration, the boot code will wait indefinitely for the host to provide patch or proceed with the default configuration as listed in Table 1.
3. The host shall then start the actual patch download sequence by sending PTCd commands (64 bytes each) until the complete patch bundle is loaded successfully.
4. The host shall then send the PTCc command to indicate the PD Controller that the patch load sequence is compete. After a successful boot sequence, the PD controller begins application execution.

NOTE: If the host intends to load a different patch bundle (either apllication configuration and/or device patch) after the device has already transitioned to the 'Application' mode (either by loading the default configuration or after a previous patch download sequence), the above patch download sequence shall be preceeded by a 'PTCr' (Patch Reset) and succeeded by a 'Gaid' (Warm Restart) 4CC command.

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 repeatedly read the four byte content of CmdX register until it reads:
   A. '0x00' indicating that the command is successfully executed.
   B. or, 'CMD' indicating that the command's execution has failed.
4. 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 toSLVA the device's Host Interface TRM for more details on the 4CC commands.

3.1.3 Example Code

1.
```
/**/
void AsyncEvtHandlerP11()
{
    s_AppContext *const pCtx = &gAppCtx;

    I2C_IF_TPS6598xIntClear(pCtx->i2cPort);
    I2C_IF_TPS6598xIntDisable(pCtx->i2cPort);

    ProcessEvent();
}

/**/
static int32_t ProcessEvent()
{
    s_AppContext    *const pCtx     = &gAppCtx;
    s_AppData       *const pData    = &gAppData[pCtx->i2cPort];

    s_TPS_IntEvent  *pSetEvent      = NULL;
    s_TPS_IntEvent  *pClrEvent      = NULL;

    int32_t     retVal = -1;

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

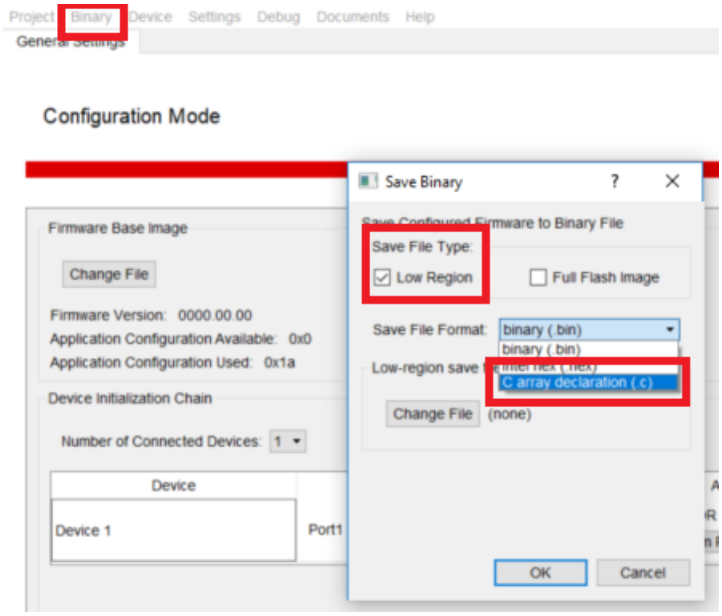
```
    /*
     * 'pCtx->p_output'/'g_output' will contain the register's content
     * after the successful execution of 'ReadReg'.
     *
     * pData->event_reg is either 0x14 or 0x15 depending on which I2C
     * is being used to communicate w/ TPS6598x
     */
    retVal = ReadReg(pData->event_reg, REG_LEN_INTEVENT1, &outdata[0]);
    RETURN_ON_ERROR(retVal);

    /*
     * Couldn't read the register.
     * Device could still be in MiniHI mode.!
     * Wait for some time and reattempt the read
     */
    if(0 == outdata[0])
    {
        Board_IF_Delay(75000); /* in uS */
        retVal = ReadReg(pData->event_reg, REG_LEN_INTEVENT1, &outdata[0]);
        RETURN_ON_ERROR(retVal);
    }

    /*
     * output[0] = Register's size in bytes
     * output[1..REG_LEN_INTEVENT1] = Register's content, and
     *                                hence '&g_output[1]' below.!
     */
    pSetEvent = (s_TPS_IntEvent *)&outdata[1];
    pClrEvent = (s_TPS_IntEvent *)&indata[0];

    /* If READY_FOR_PATCH is set */
    if(0 != pSetEvent->readyforpatch)
    {
        pClrEvent->readyforpatch = 1;
        SignalEvent(APP_EVENT_READY_FOR_PATCH);
    }

    /* If PATCH_LOADED is set */
    if(0 != pSetEvent->patchloaded)
    {
        pClrEvent->patchloaded = 1;
        SignalEvent(APP_EVENT_PATCH_LOADED);
    }

    /*
     * Application shall process 'APP_EVENT_READY_FOR_PATCH'
     * The device's 'READY_FOR_PATCH' event can now be cleared
     */
    retVal = WriteReg(pData->clear_reg, REG_LEN_INTCLEAR1, &indata[0]);
    RETURN_ON_ERROR(retVal);

    return 0;
}
```

2.
```
/**/
static int32_t StartPatchDownload()
{
    s_AppContext    *const pCtx     = &gAppCtx;

    s_PTCs_InData   ptcsInData      = {0};
    s_PTCs_OutData  *p_ptcsOutData  = NULL;

    int32_t retVal  = -1;
```

```
    UART_PRINT("Starting Patch Download\n\r");

    I2C_IF_TPS6598xIntEnable(pCtx->i2cPort);

    /*
     * Fill the input, and execute 'PTCs'
     */
    ptcsInData.config_start = true;
    ptcsInData.patch_start  = true;
    retVal = ExecCmd(PTCs, sizeof(ptcsInData),
                     (int8_t *)&ptcsInData, PTCs_OUTPUT_SIZE);
    RETURN_ON_ERROR(retVal);

    /*
     * 'pCtx->p_output'/'g_output' will contain the command's output
     * after the successful execution of PTCs.
     */
    p_ptcsOutData = (s_PTCs_OutData *)&g_output[1];
    if(p_ptcsOutData->patch_status != PTCs_SUCCESS)
    {
        ERR_PRINT(p_ptcsOutData->patch_status);
        retVal = -1;
        goto error;
    }

error:
    return retVal;
}
```

3. Low region binary 'C' array format can be generated using the Application Customization GUI. tps6598x_lowregion_array and gSizeLowregionArray referenced in the below snippet are taken from this generated file.

Figure 3. C Array Low Region File Format




```
/**/
static int32_t DownloadDataChunk()
{
    s_PTCd_OutData  *p_ptcdOutData  = NULL;

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

```
    uint32_t    bytesUpdated = 0;
    uint32_t    idx = 0;
    int32_t     retVal = -1;

    /*
     * tps6598x_lowregion_array and gSizeLowregionArray are defined
     * in the 'C' array file that could be created using the
     * Application customization GUI
     */
    for (idx = 0; idx < gSizeLowregionArray/PATCH_BUNDLE_SIZE; idx++)
    {
        UART_PRINT(".");

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

        /*
         * 'pCtx->p_output'/'g_output' will contain the command's output
         * after the successful execution of PTCd.
         */
        p_ptcdOutData = (s_PTCd_OutData *)&outdata[1];
        if(p_ptcdOutData->transfer_status != PTCd_SUCCESS)
        {
            ERR_PRINT(p_ptcdOutData->transfer_status);
            retVal = -1;
            goto error;
        }

        bytesUpdated += PATCH_BUNDLE_SIZE;
    }

    /* Push more bytes if any */
    if(gSizeLowregionArray > bytesUpdated)
    {
        retVal = ExecCmd(PTCd, gSizeLowregionArray - bytesUpdated,\
                         (int8_t *)&tps6598x_lowregion_array[idx * PATCH_BUNDLE_SIZE],
                         PTCd_OUTPUT_SIZE, &outdata[0]);
        RETURN_ON_ERROR(retVal);

        /*
         * 'pCtx->p_output'/'g_output' will contain the command's output
         * after the successful execution of PTCd.
         */
        p_ptcdOutData = (s_PTCd_OutData *)&outdata[1];
        if(p_ptcdOutData->transfer_status != PTCd_SUCCESS)
        {
            ERR_PRINT(p_ptcdOutData->transfer_status);
            retVal = -1;
            goto error;
        }
    }

    /*
     * Proceed to the next step after the patch is successfully downloaded
     */
    UART_PRINT("\n\r");
    SignalEvent(APP_EVENT_PATCH_DOWNLOADED);

error:
    return retVal;
```

```
}
```

4.
```
/**/
static int32_t PatchDownloadComplete()
{
    s_PTCc_OutData  *p_ptccOutData  = NULL;
    int32_t retVal = -1;

    UART_PRINT("Patch Download Complete\n\r");

    /*
     * Execute PTCc to indicate to the device that the patch
     * is successfully downloaded
     */
    retVal = ExecCmd(PTCc, 0, NULL, PTCc_OUTPUT_SIZE);
    RETURN_ON_ERROR(retVal);

    /*
     * 'pCtx->p_output'/'g_output' will contain the command's output
     * after the successful execution of PTCc.
     */
    p_ptccOutData = (s_PTCc_OutData *)&g_output[1];
    if(p_ptccOutData->patch_complete_status != PTCc_SUCCESS)
    {
        ERR_PRINT(p_ptccOutData->patch_complete_status);
        retVal = -1;
        goto error;
    }

    /*
     * Read mode and version just to ensure the device is running
     * the downloaded patch/configuration
     */
    retVal = ReadMode();
    RETURN_ON_ERROR(retVal);
    retVal = ReadVersion();
    RETURN_ON_ERROR(retVal);

error:
    return retVal;
}
```

3.2 Patch Bundle from External Flash

3.2.1 Execution Flow

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 invalid patch bundle, the boot firmware proceeds to check for the existence of a host.

The boot code checks for any errors while reading the external SPI flash and enters a recovery state to attempt to restart the load process and retry for a second time. If the second time fails, the boot flow proceeds to the next phase.

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

The SPI master of the device supports SPI Mode 0. For Mode 0, data delay is defined so that data is output on the same cycle as chip select (SPI_SSZ 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 so that when data is not being transferred the SPI_CLK pin is held (or idles) low.

The flash memory IC must support a 12-MHz SPI clock frequency. The size of the flash must be at least 16kB to hold the maximum ROM patch and configuration code. The minimum erasable sector size of the flash must be 4kB. The W25X05CL or similar is recommended.

Refer to *TPS6598x FW Update From Embedded-Controller Over I²C* application report for more details on programming and erasing the attached flash memory over SPI using the host interface of the device.

4 Appendix

4.1 Boot Flow

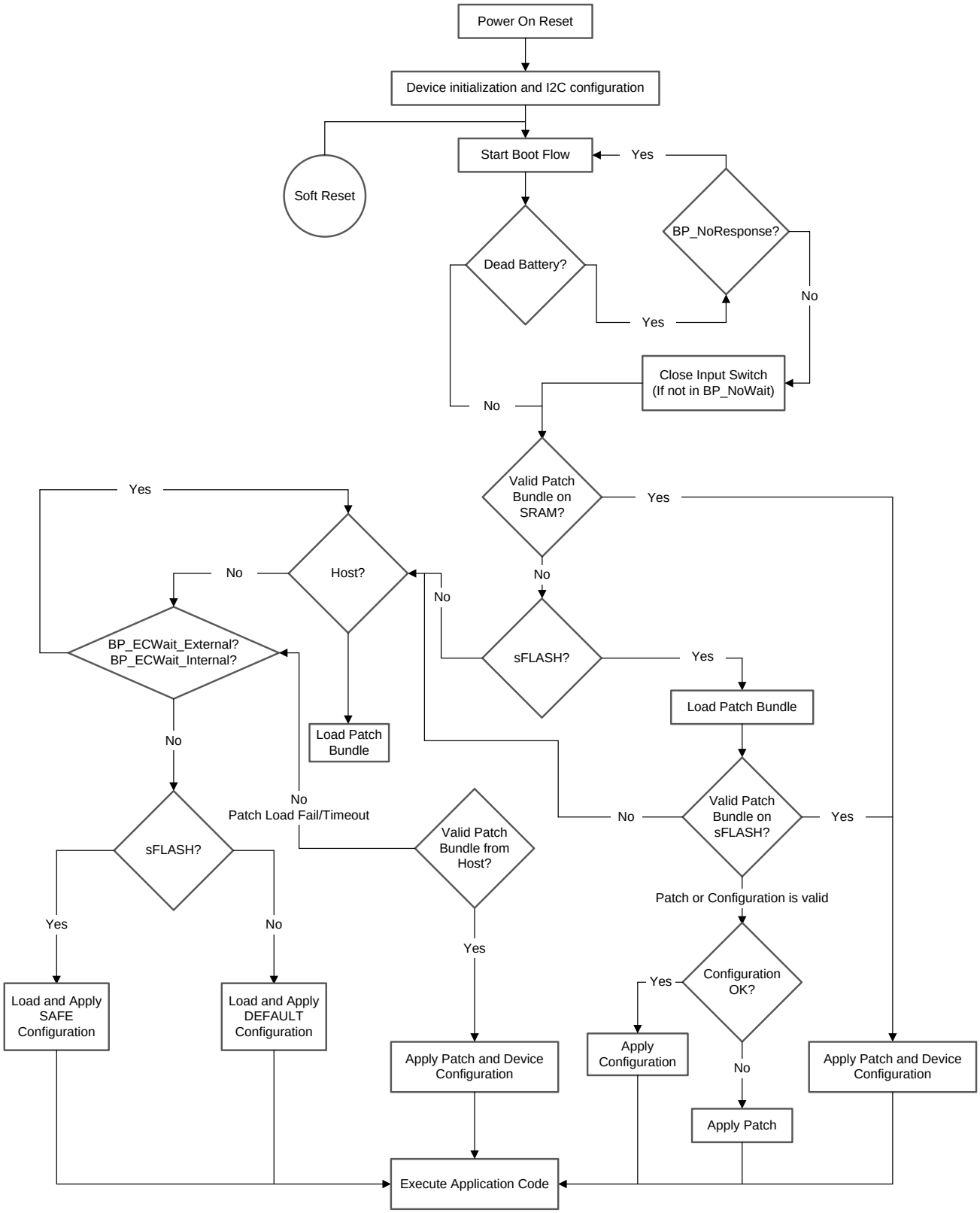


Figure 4. Boot Flow

- 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 I^2C or internal ROM, depending on the BUSPOWERZ configuration.
- 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 dead-battery 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.
- 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.
- 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.
- 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 1.
- 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