Application Note

TPS25751 and TPS26750 EEPROM Update Over I²C

TEXAS INSTRUMENTS

Aya Khedr

ABSTRACT

The TPS25751 and TPS26750 is a highly integrated stand-alone USB Type-C® and Power Delivery (PD) controller optimized for applications supporting USB-C PD Power. The PD Controller application binary can be pushed over I2C to the PD controller using the I2Ct port, or the PD controller can read from an external EEPROM at target address 0x50 on the I2Cc port. When the host must update the Patch Bundle used for booting, the host must follow a particular process. The EEPROM-update process uses 4CC ASCII commands to enable the host to download the patch bundle onto the external EEPROM.

Table of Contents

Trademarks

USB Type-C® is a registered trademark of USB Implementers Forum.
All trademarks are the property of their respective owners.

1 Introduction

This application note details the EEPROM update process of the device by enabling external hosts to program the patch bundles onto the attached external EEPROM using the host interface of the device.

The document also details the EEPROM boot flow, how to update the EEPROM image, and an EEPROM update example through flow charts and code examples.

2 EEPROM Boot Flow

During the boot process, the PD controller can load the Patch Bundle from the external EEPROM connected to the I2Cc port as described in the following. After the PD controller is in the APP mode, that is MODE register reads as *APP*, the host can write to the EEPROM as described in the following to update the Patch Bundle used for booting the PD controller. During the process the previous Patch Bundle is kept intact so that the PD controller can boot from the Patch Bundle in case errors occur when writing the new Patch Bundle into the EEPROM.

The Patch Bundle contains a FW patch image in addition to a set of configurations that set the default value of the Host Interface (HI) registers.

The EEPROM is divided into two regions to allow for it being updated without invalidating the previous Patch Bundle until the new Patch Bundle has been verified. The active region is the one containing the latest Patch Bundle.

2.1 Boot Process

At boot, the PD controller can first read the Header_ID from the Low Region at the address LowRegionStart and LowAppConfigOffset. If any error occurs in reading the Low Region Header_ID, the PD controller can then read the Header_ID from the High Region at the address HighRegionStart and HighAppConfigOffset. If any error occurs in reading the High Region Header_ID, the PD controller can loop back and try the Low Region again. The PD controller can only make two attempts, after that PD controller aborts the EEPROM loading process.

If the PD controller reads the correct Header_ID (PD controller is expecting 0xACE0_0001) in the Low Region, then it can begin reading the Patch Bundle from the Low Region. *If there is a CRC error while reading the Patch Bundle, the PD controller does not attempt to read from the High Region*. If the PD controller reads the correct Header_ID in the High Region, then it can begin reading the Patch Bundle from the High Region. If there is a CRC error while reading the Patch Bundle, the PD controller can attempt to read from the Low Region. However, the PD controller does not make more than two attempts on any region.

Therefore, when updating one of the regions of the EEPROM it is critical to verify the new Patch Bundle in the region before pointing the Region Start to it.

If the EEPROM loading process is aborted, then the PD controller can update the BOOT_STATUS register accordingly and assert the INT_EVENTx.ReadyForPatch interrupt. the PD controller then waits indefinitely for the host to load a patch over the I2Cc port or to issue a *GAID* 4CC command to reboot the PD controller. This behavior also occurs when there is no EEPROM present.

Figure 2-1 shows the memory map of the EEPROM and where the pointers and offsets reside assuming that the EEPROM has initially been written with the same Patch Bundle in both regions. The PD controller looks for the Header_ID of the Low Region at address LowRegionStart and LowAppConfigOffset, and it looks for the Header_ID of the High Region at the address HighRegionStart and HighAppConfigOffset.

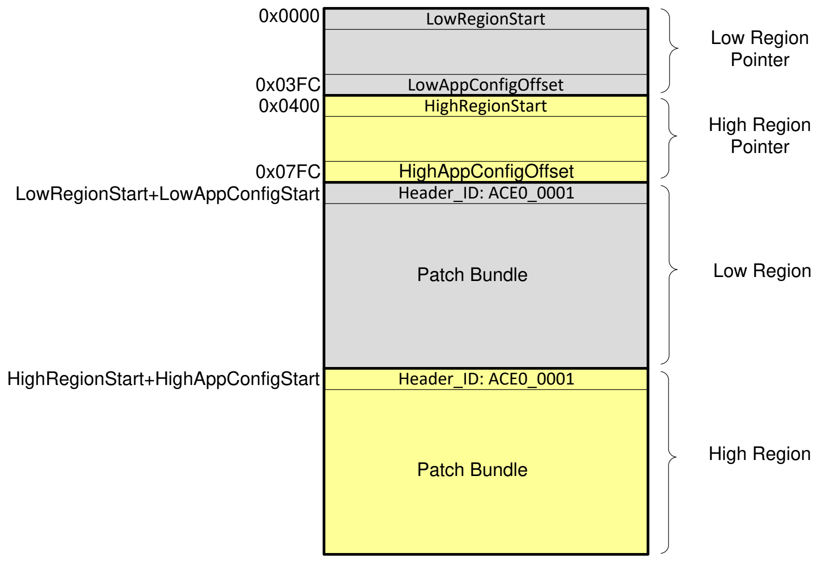

Figure 2-1. EEPROM Memory Map

Note

The external EEPROM shall be programmed with a *Full Flash* binary the very first time the platform is powered up so that the region headers are set up correctly. A *Full Flash* binary file can be generated from the USBCPD Application Customization Tool. The device tool can also generate a *Low Region* binary, and this file cam be used by the external host for the EEPROM update.

2.2 Updating the EEPROM Image

When the host must update the Patch Bundle used for booting it must follow the process described in this section. The top-level flow is to update the region that the PD controller did NOT boot from as shown in Figure 2-2. The top-level flow executes the function UpdateRegionOfEeprom() illustrated in Figure 2-3.

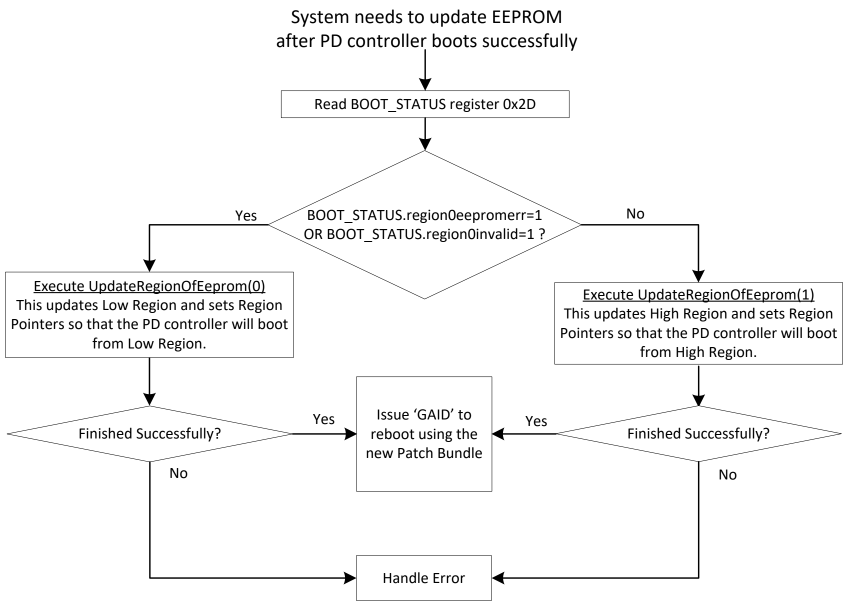
System needs to update EEPROM after PD controller boots successfully
Read BOOT_STATUS register 0x2D
BOOT_STATUS.region0eepromerr=1 OR BOOT_STATUS.region0invalid=1 ?
Yes
No
Execute UpdateRegionOfEeprom(0)
This updates Low Region and sets Region Pointers so that the PD controller will boot from Low Region.
Execute UpdateRegionOfEeprom(1)
This updates High Region and sets Region Pointers so that the PD controller will boot from High Region.
Finished Successfully?
Yes
Issue 'GAID' to reboot using the new Patch Bundle
Yes
Finished Successfully?
No
No
Handle Error

Figure 2-2. Flow for Updating the EEPROM

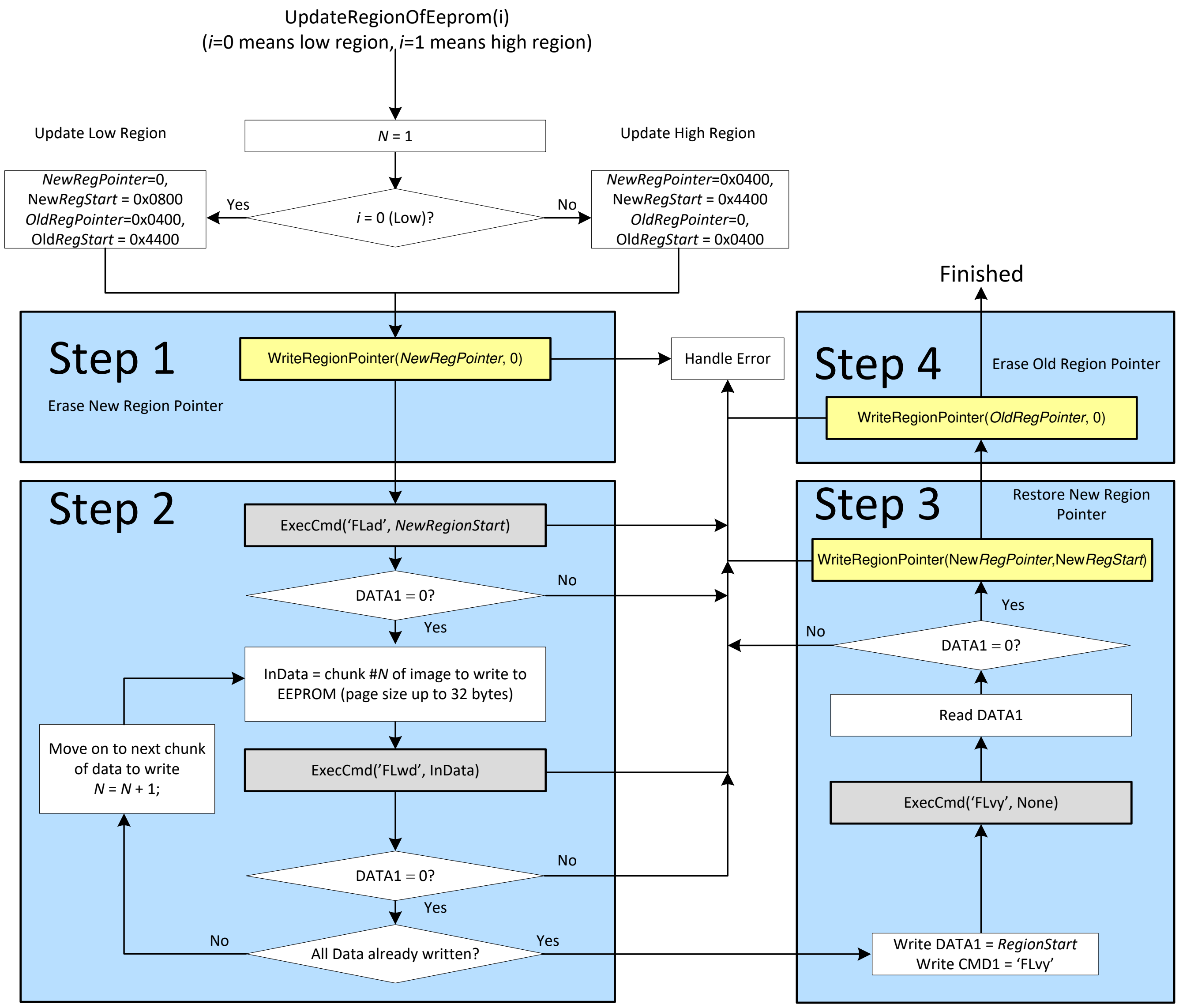
UpdateRegionOfEeprom(i)
(i=0 means low region, i=1 means high region)
N = 1
Update Low Region
Update High Region
i = 0 (Low)?
Yes
No
NewRegPointer=0, NewRegStart = 0x0800 OldRegPointer=0x0400, OldRegStart = 0x4400
NewRegPointer=0x0400, NewRegStart = 0x4400 OldRegPointer=0, OldRegStart = 0x0400
Step 1
Erase New Region Pointer
WriteRegionPointer(NewRegPointer, 0)
Handle Error
Finished
Step 4
Erase Old Region Pointer
WriteRegionPointer(OldRegPointer, 0)
Step 2
ExecCmd('FLad', NewRegionStart)
DATA1 = 0?
No
Yes
InData = chunk #N of image to write to EEPROM (page size up to 32 bytes)
Move on to next chunk of data to write N = N + 1;
ExecCmd('FLwd', InData)
DATA1 = 0?
No
Yes
All Data already written?
No
Yes
Step 3
Restore New Region Pointer
WriteRegionPointer(NewRegPointer,NewRegStart)
Yes
No
DATA1 = 0?
Read DATA1
ExecCmd('FLvy', None)
Write DATA1 = RegionStart Write CMD1 = 'FLvy'

Figure 2-3. Details of the UpdateRegionOfEeprom() Function Used to Update EEPROM

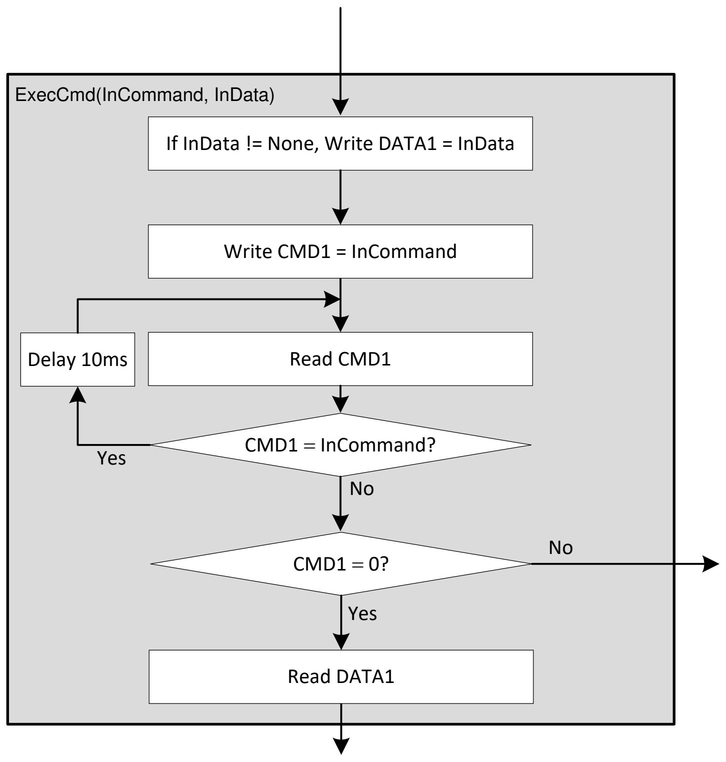

Figure 2-4. Details of the ExecCmd() Block

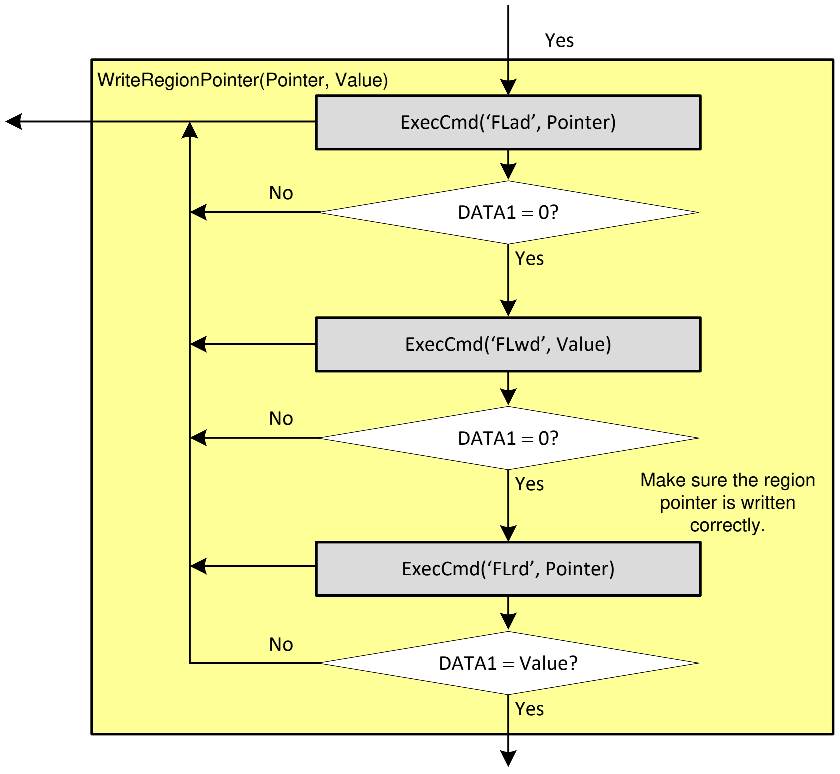

Figure 2-5. Details of the WriteRegionPointer() Block

2.3 Commands

The EEPROM-update process uses the 4CC ASCII commands listed in Table 2-1.

Table 2-1. 4CC ASCII Commands

Name of 4CC Command	ASCII	Input DataX Length (In Bytes)	Output DataX Length (In Bytes)	Description
Flash Memory Read	FLrd	4	16	The *FLrd* Command reads the flash at the specified address
Flash Memory Write Start Address	FLad	4	1	The *FLad* Command sets start address in preparation the flash write
Flash Memory Write	FLwd	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	4	1	The *FLvy* Command verifies if the patch or configuration is valid
Cold reset request	GAID	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 TPS25751/TPS26750 and reload the new FW version from non-volatile Flash memory

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 Data1 (0x09) register.
2. The application shall then write the 4CC command characters into the corresponding Cmd1 (0x08) register.
3. The application shall wait until the four byte content of Cmd1 register reads the following – Applications can either poll, or set and use the *Cmd1Complete* 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 Data1 register which will contain the output. Refer the device's Host Interface TRM for more details on the 4CC commands *TPS25751 Technical Reference Manual*.

2.4 EEPROM Update Example

In the following example, the assumption is that the initial state of the EEPROM is that the Low Region and the High Region both have the same Patch Bundle. The PD controller can boot from the Low Region. Figure 2-6 shows the EEPROM memory map for this initial condition.

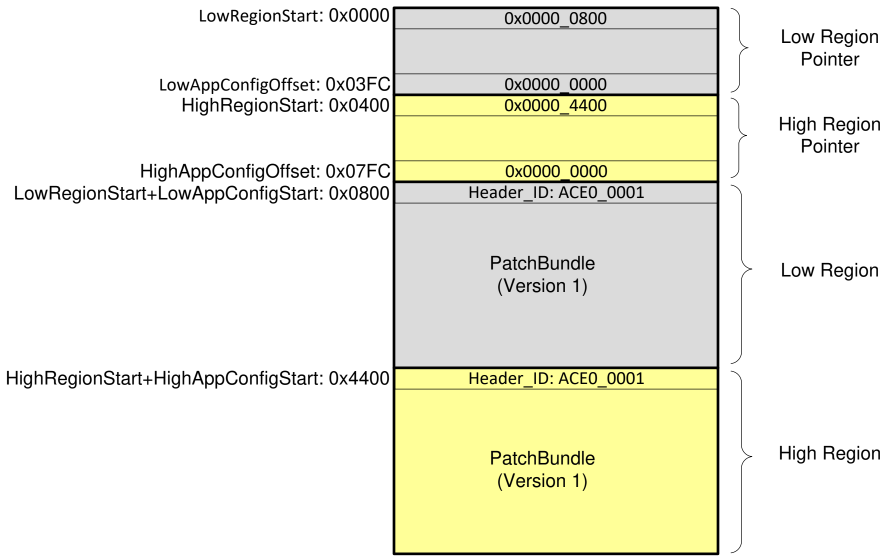

Figure 2-6. Initial State of the EEPROM

When the host must update the Patch Bundle that the PD controller uses for booting, it first erases the High Region pointer so that if there is an interruption while writing the new Patch Bundle to the High Region, it is guaranteed the PD controller does not attempt to load it. Specifically, Step 1 of the function UpdateRegionOfEeprom(1) from Figure 2-3 is executed, and Figure 2-7 shows the memory map after the High Region pointer has been erased successfully. If booted in this state, Patch Bundle (Version 1) is loaded from Low Region.

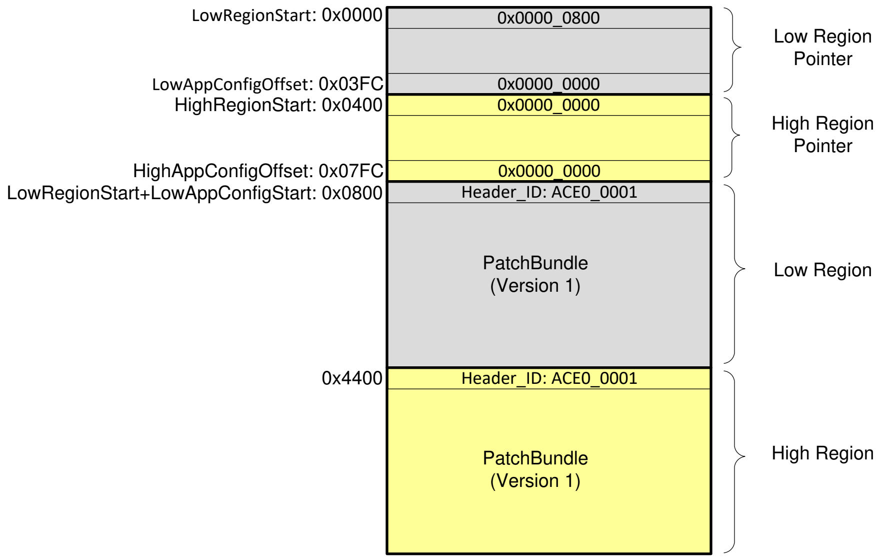

Figure 2-7. State of the EEPROM following UpdateRegionOfEeprom(1) Step 1

Next, the host writes the new Patch Bundle (Version 2) to the High Region. Specifically, Step 2 of the function UpdateRegionOfEeprom(1) from Figure 2-3 is executed, and Figure 2-8 shows the memory map following this step. Note that if the PD controller boots with the EEPROM in this state, the controller does still load Patch Bundle (Version 1) from the Low Region.

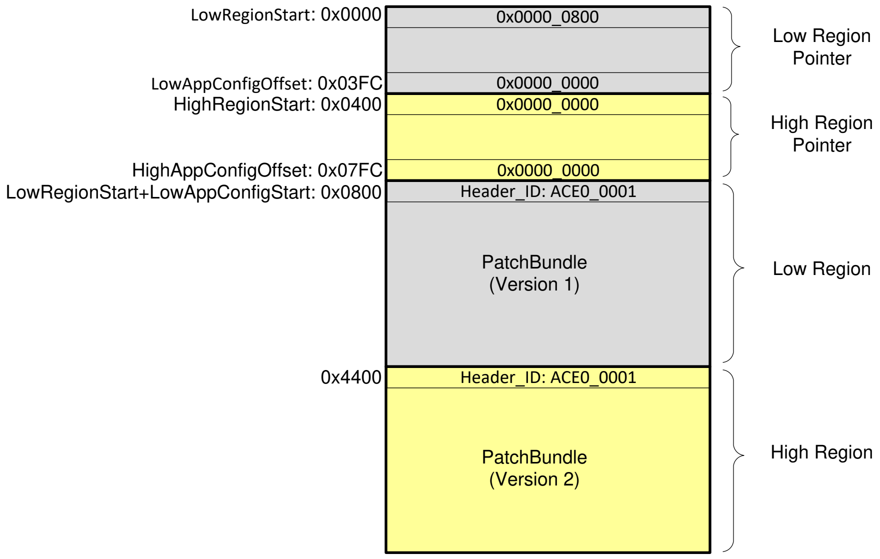

Figure 2-8. State of the EEPROM following UpdateRegionOfEeprom(1) Step 2

Next, the host can verify the contents of the EEPROM High Region using the *FLvy* command. If that succeeds, then the host can write 0x4400 into HighRegionStart at address 0x0400. This happens in Step 3 of UpdateRegionOfEeprom(1) in Figure 2-3, and Figure 2-9 shows the memory map after this is done. If the PD controller reboots with the EEPROM in this state, it can still first attempt to boot from the Low Region.

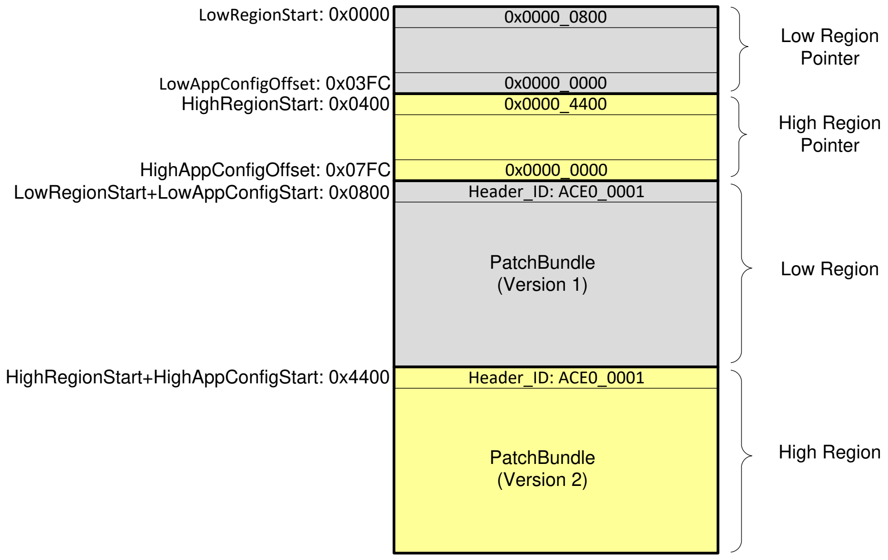

Figure 2-9. State of EEPROM following UpdateRegionOfEeprom(1) Step 3

The last step is to erase the LowRegionStart value so that the PD controller can boot from the High Region. The host can use the WriteRegionPointer() functionality as shown in Step 4 of UpdateRegionOfEeprom(1) as illustrated in Figure 2-3. Figure 2-10 shows the memory map after Step 4 is complete. Because the LowRegionStart is now 0, the contents of the Low Region have no impact on how the PD controller boots.

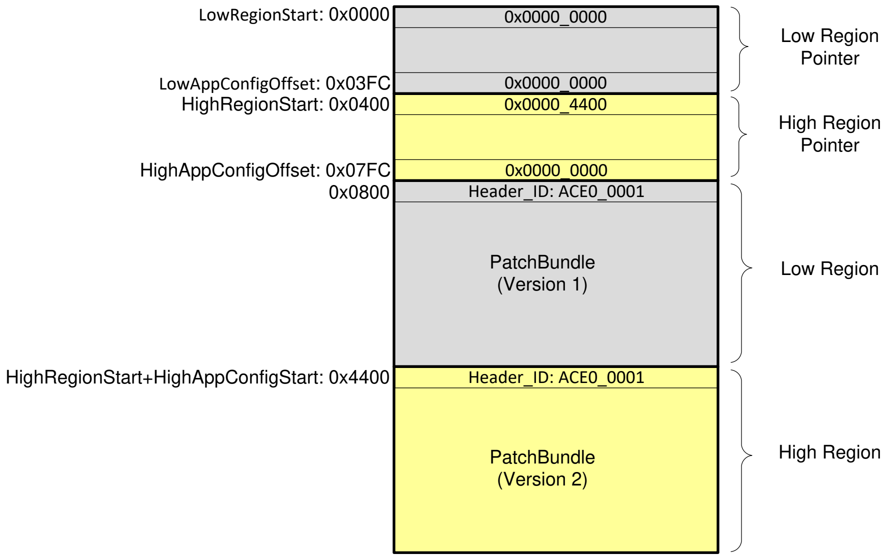

Figure 2-10. State of EEPROM following UpdateRegionOfEeprom(1) Step 4

The next time the host must update the EEPROM image the host can execute UpdateRegionOfEeprom(0), and the process can proceed in a similar manner. The host can optionally execute UpdateRegionOfEeprom(0) immediately with the same new Patch Bundle it has just written to the High Region.

3 Source Code Example

This section gives example source code to implement the EEPROM flow described previously.

3.1 UpdateRegionOfEeprom()

```
const uint32_t region_ptr_start[NUM_OF_REGIONS] = {0x0 , 0x400 };
const uint32_t region_ptr_appconfig_offset[NUM_OF_REGIONS] = {0x3FC, 0x7FC };
const uint32_t region_addr_patchbundle[NUM_OF_REGIONS] = {0x800, 0x4400};
static int32_t UpdateRegionOfEeprom()
{
s_AppContext *const pCtx = &gAppCtx;
int32_t retVal = -1;
UART_PRINT("\n\rActive Region is [%d] - Region being updated is [%d]\n\r",\
pCtx->active_region, pCtx->inactive_region);
/*
* Region-0/Region-1 is currently active, hence update Region-1/Region-0 respectively
*/
retVal = UpdateRegionOfEeprom_Step1(pCtx->inactive_region);
if(0 != retVal)
{
UART_PRINT("Region[%d] update Step 1 failed.! Next boot will happen from Region[%d]\n\r",\
pCtx->inactive_region, pCtx->active_region);
goto error;
}
/*
* Region-0/Region-1 is currently active, hence update Region-1/Region-0 respectively
*/
retVal = UpdateRegionOfEeprom_Step2(pCtx->inactive_region);
if(0 != retVal)
{
UART_PRINT("Region[%d] update Step 2 failed.! Next boot will happen from Region[%d]\n\r",\
pCtx->inactive_region, pCtx->active_region);
goto error;
}
/*
* Write is through. Now verify if the content/copy is valid.
* Update the corresponding region-pointer point to the new region.
*/
retVal = UpdateRegionOfEeprom_Step3(pCtx->inactive_region);
if(0 != retVal)
{
UART_PRINT("Region[%d] update Step 3 failed.! Next boot will happen from Region[%d]\n\r",\
pCtx->inactive_region, pCtx->active_region);
goto error;
}
/*
* Invalidate the region-pointer of the old region.
*/
retVal = UpdateRegionOfEeprom_Step4(pCtx->active_region);
if(0 != retVal) {goto error;}
error:
SignalEvent(APP_EVENT_END_UPDATE);
return retVal;
}
```

3.2 UpdateRegionOfEeprom_Step1

```
static int32_t UpdateRegionOfEeprom_Step1(uint8_t region_number)
{
int32_t retVal = -1;
/*
* First erases the region-pointer so that if there is an interruption while writing
* the new Patch Bundle, it is guaranteed the PD controller won't try to load it.
*/
retVal = WriteRegionPointer(region_number, 0);
RETURN_ON_ERROR(retVal);
error:
return retVal;
}
```

3.3 UpdateRegionOfEeprom_Step2()

```
static int32_t UpdateRegionOfEeprom_Step2(uint8_t region_number)
{
uint8_t outdata[MAX_BUF_BSIZE] = {0};
s_TPS_flad fladInData = {0};
uint32_t bytesUpdated = 0;
int32_t retVal = -1;
int32_t idx = -1;
/*
* Set the start address for the next write
*/
fladInData.flashaddr = region_addr_patchbundle[region_number];
retVal = ExecCmd(FLad, sizeof(fladInData), (uint8_t *)&fladInData,
TASK_RET_CODE_LEN, &outdata[0]);
RETURN_ON_ERROR(retVal);
if(0 != outdata[1]) {retVal = -1; goto error;}
for (idx = 0; idx < gSizeLowregionArray/PATCH_BUNDLE_SIZE; idx++)
{
/*
* Execute FLwd with PATCH_BUNDLE_SIZE bytes of patch-data
* in each iteration
*/
retVal = ExecCmd(FLwd, PATCH_BUNDLE_SIZE,\
(uint8_t *)&tps6598x_lowregion_array[idx * PATCH_BUNDLE_SIZE],
TASK_RET_CODE_LEN, &outdata[0]);
RETURN_ON_ERROR(retVal);
if(0 != outdata[1]) {retVal = -1; goto error;}
bytesUpdated += PATCH_BUNDLE_SIZE;
Board_IF_Delay(75); /* in uSecs */
}
/* Push more bytes if any */
if(gSizeLowregionArray > bytesUpdated)
{
retVal = ExecCmd(FLwd, gSizeLowregionArray - bytesUpdated,\
(uint8_t *)&tps6598x_lowregion_array[idx * PATCH_BUNDLE_SIZE],
TASK_RET_CODE_LEN, &outdata[0]);
RETURN_ON_ERROR(retVal);
if(0 != outdata[1]) {retVal = -1; goto error;}
}
error:
return retVal;
}
```

3.4 UpdatingRegionOfEeprom_Step3()

```
static int32_t UpdateRegionOfEeprom_Step3(uint8_t new_region_number)
{
uint8_t outdata[MAX_BUF_BSIZE] = {0};
s_TPS_flvy flvyInData = {0};
int32_t retVal = -1;
flvyInData.flashaddr = region_addr_patchbundle[new_region_number];
retVal = ExecCmd(FLvy, sizeof(flvyInData), (uint8_t *)&flvyInData, \
TASK_RET_CODE_LEN, &outdata[0]);
if(0 != outdata[1]) {retVal = -1; goto error;}
retVal = WriteRegionPointer(new_region_number, region_addr_patchbundle[new_region_number]);
RETURN_ON_ERROR(retVal);
error:
return retVal;
}
```

3.5 UpdatingRegionOfEeprom_Step4()

```
static int32_t UpdateRegionOfEeprom_Step4(uint8_t old_region_number)
{
int32_t retVal = -1;
retVal = WriteRegionPointer(old_region_number, 0);
RETURN_ON_ERROR(retVal);
error:
return retVal;
}
```

3.6 WriteRegionPointer()

```
static int32_t WriteRegionPointer(const uint8_t region_number, const uint32_t value)
{
uint8_t outdata[MAX_BUF_BSIZE] = {0};
s_TPS_flad fladInData = {0};
uint32_t regionVal = 0;
int32_t retVal = -1;
fladInData.flashaddr = region_ptr_start[region_number];
retVal = ExecCmd(FLad, sizeof(fladInData), (uint8_t *)&fladInData, TASK_RET_CODE_LEN, &outdata[0]);
RETURN_ON_ERROR(retVal);
if(0 != outdata[1]) {retVal = -1; goto error;}
retVal = ExecCmd(FLwd, sizeof(uint32_t), (uint8_t *)&value, TASK_RET_CODE_LEN, &outdata[0]);
RETURN_ON_ERROR(retVal);
if(0 != outdata[1]) {retVal = -1; goto error;}
retVal = ExecCmd(FLrd, sizeof(uint32_t), (uint8_t *)&region_ptr_start[region_number],
sizeof(s_TPS_flrdassert), &outdata[0]);
RETURN_ON_ERROR(retVal);
regionVal = (outdata[4] << 24) | (outdata[3] << 16) | (outdata[2] << 8) | (outdata[1] << 0);
if(value != regionVal) {retVal = -1; goto error;}
error:
return retVal;
}
```

4 Recovering From EEPROM Failure

If the EEPROM loading terminates without a valid Patch Bundle, then the INT_EVENTx.ReadyForPatch interrupt gets asserted. The host must read the BOOT_STATUS register 0x2D to discover why booting from EEPROM failed. Then the host must force the PD controller into the APP mode by pushing a patch using the *PBMx* commands (see Technical Reference Manual for details). This can be the full Patch Bundle that is normally in EEPROM. After the PD controller is in the APP mode, the host can write to the EEPROM using the *FLxx* commands and correct the problem. Figure 4-1 shows the recommended boot flow.

This boot flow requires the host to be able to correct the EEPROM. If the host requires the PD controller to enable the sink path before it can boot, then the appropriate dead-battery configuration must be selected by the ADCINx pins. In this case, the SafeMode dead-battery configuration is not applicable.

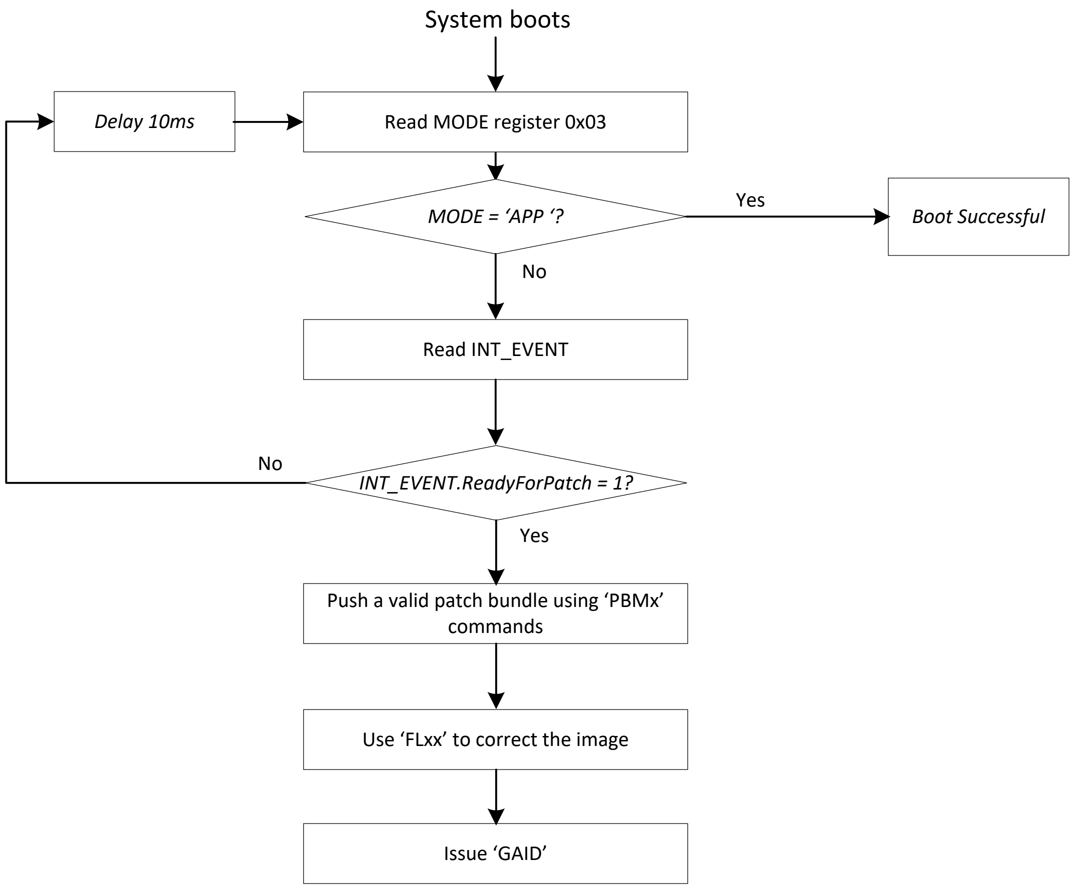

Figure 4-1. Recommended Boot Flow for EEPROM

5 Summary

The TPS25751 and TPS26750 application binary can be pushed to the PD controller over I2C using the I2Ct port, or the PD controller can read it from an external EEPROM. When the host must update the Patch Bundle used for booting, it must follow a particular sequence.

The host shall implement the below sequence to update the patch-bundle:

- Using *FLrd* command, the host shall query the device for the address of the region on external EEPROM that is to be updated.
- The host shall then set the start address for the next write using *FLad* command, and start sending the patch-bundle 32 bytes at a time using *FLwd* command.
- The device automatically increments the write-address after the successful execution of *FLwd*– The host does not need to set the start address for every write request.
- The host shall then verify the contents of the EEPROM using *FLvy* command.
- After both regions are updated, the host shall cold-reset the device using *GAID* – The device can go through the boot sequence all over again, and load the updated patch-bundle.

6 References

- Texas Instruments, *TPS25751 Technical Reference Manual*.
- Texas Instruments, *TPS26750 Technical Reference Manual*.