Application Note

通过 I²C 进行 TPS25751 和 TPS26750 EEPROM 更新

Aya Khedr

摘要

TPS25751 和 TPS26750 是高度集成的独立式 USB Type-C® 和电力输送 (PD) 控制器，针对支持 USB-C PD 电源的应用进行了优化。PD 控制器应用程序二进制文件可以使用 I2Ct 端口通过 I2C 推送至 PD 控制器，也可以由 PD 控制器从连接到 I2Cc 端口、目标地址为 0x50 的外部 EEPROM 中进行读取。主机必须更新用于引导的补丁捆绑包时，必须遵循特定的流程。EEPROM 更新过程使用 4CC ASCII 命令来允许主机将补丁捆绑包下载到外部 EEPROM 上。

内容

商标

1 引言

本应用手册详细介绍了器件的 EEPROM 更新过程，方法是使外部主机使用器件的主机接口将补丁捆绑包编程到所连接的外部 EEPROM 上。

本文档还通过流程图和代码示例详细介绍了 EEPROM 引导流程、如何更新 EEPROM 映像以及 EEPROM 更新示例。

2 EEPROM 引导流程

在引导过程中，PD 控制器从连接到 I2Cc 端口的外部 EEPROM 加载补丁捆绑包，如下所述。在 PD 控制器处于 APP 模式（即 MODE 寄存器读数为 *APP*）后，主机可如下所述写入 EEPROM 以更新用于引导 PD 控制器的补丁捆绑包。在此过程中，先前的补丁捆绑包保持不变，因此在将新的补丁捆绑包写入 EEPROM 的过程中出现任何问题时，PD 控制器便可从旧的补丁捆绑包进行引导。

除了一组可设置主机接口 (HI) 寄存器默认值的配置外，补丁捆绑包还包含一个固件补丁映像。

EEPROM 分为两个区域，因此进行更新时，在完成新补丁捆绑包的验证之前不必使先前的补丁捆绑包失效。有效区域是包含最新补丁捆绑包的区域。

2.1 引导过程

在引导时，PD 控制器将首先从低区中的地址 LowRegionStart 和 LowAppConfigOffset 处读取 Header_ID。如果在读取低区 Header_ID 时出现任何错误，则 PD 控制器将从高区中的地址 HighRegionStart 和 HighAppConfigOffset 处读取 Header_ID。如果在读取高区 Header_ID 时出现任何错误，PD 控制器将返回并再次尝试低区。PD 控制器只会进行两次尝试，然后中止 EEPROM 加载过程。

如果 PD 控制器在低区中读取到正确的 Header_ID（预期为 0xACE0_0001），那么它将开始从低区读取补丁捆绑包。*如果在读取补丁捆绑包时出现 CRC 错误，PD 控制器不会尝试从高区读取*。如果 PD 控制器在高区中读取到正确的 Header_ID，那么它将开始从高区读取补丁捆绑包。如果在读取补丁捆绑包时出现 CRC 错误，PD 控制器会尝试从低区读取。但是，PD 控制器不会对任何区域进行超过两次的尝试。

因此，当更新 EEPROM 的其中一个区域时，在将区域起点指向该区域之前验证该区域中的新补丁捆绑包至关重要。

如果 EEPROM 加载过程中止，则 PD 控制器将相应地更新 BOOT_STATUS 寄存器并使 INT_EVENTx.ReadyForPatch 中断生效。然后，它将无限期等待主机通过 I2Cc 端口加载补丁或发出 *GAID* 4CC 命令以重新启动 PD 控制器。不存在 EEPROM 时，也会出现这种情况。

图 2-1 展示了 EEPROM 的存储器映射以及指针和偏移量所在的位置，其中假设 EEPROM 最初已在两个区域中写入了相同的补丁捆绑包。PD 控制器在地址 LowRegionStart 和 LowAppConfigOffset 处查找低区的 Header_ID，并在地址 HighRegionStart 和 HighAppConfigOffset 处查找高区的 Header_ID。

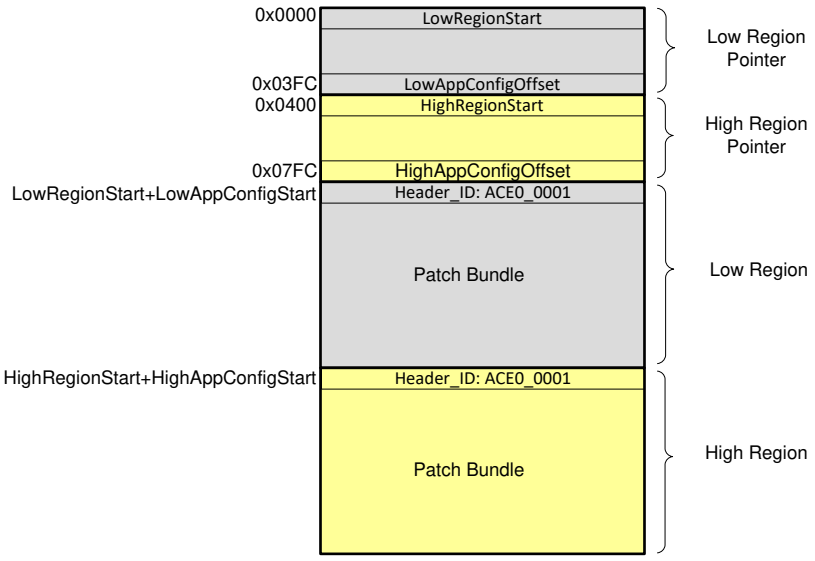

图 2-1. EEPROM 存储器映射

备注

首次为平台上电时，应使用完整的*全闪存*二进制文件对外部 EEPROM 进行编程，以便正确设置区域头部。可通过 USBCPD 应用程序自定义工具生成*全闪存*二进制文件。器件工具还可以生成*低区*二进制文件，供外部主机用于 EEPROM 更新。

2.2 更新 EEPROM 映像

当主机须更新用于引导的补丁捆绑包时，必须遵循本节中所述的过程。顶级流程是更新 PD 控制器未从其执行引导的区域，如图 2-2 所示。顶级流程执行图 2-3 中所示的函数 UpdateRegionOfEeprom()。

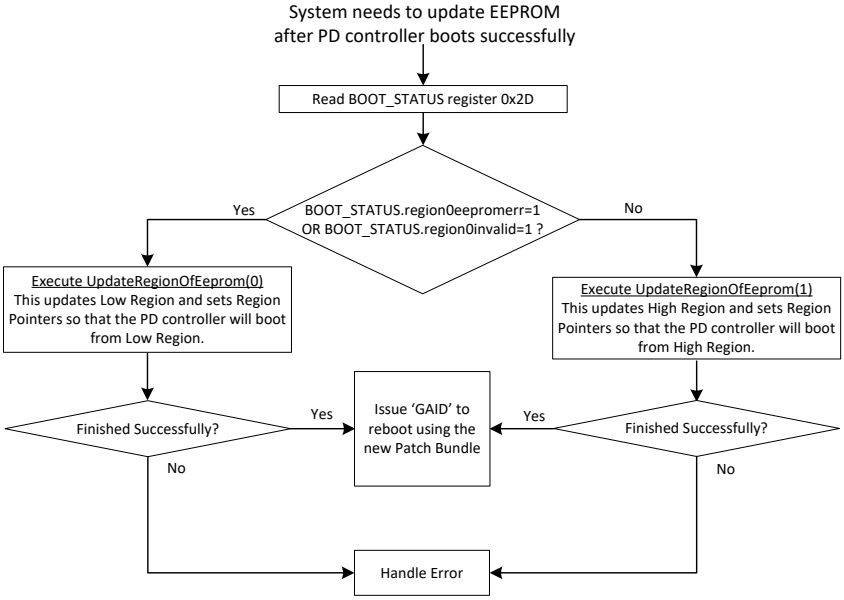

图 2-2. 更新 EEPROM 的流程

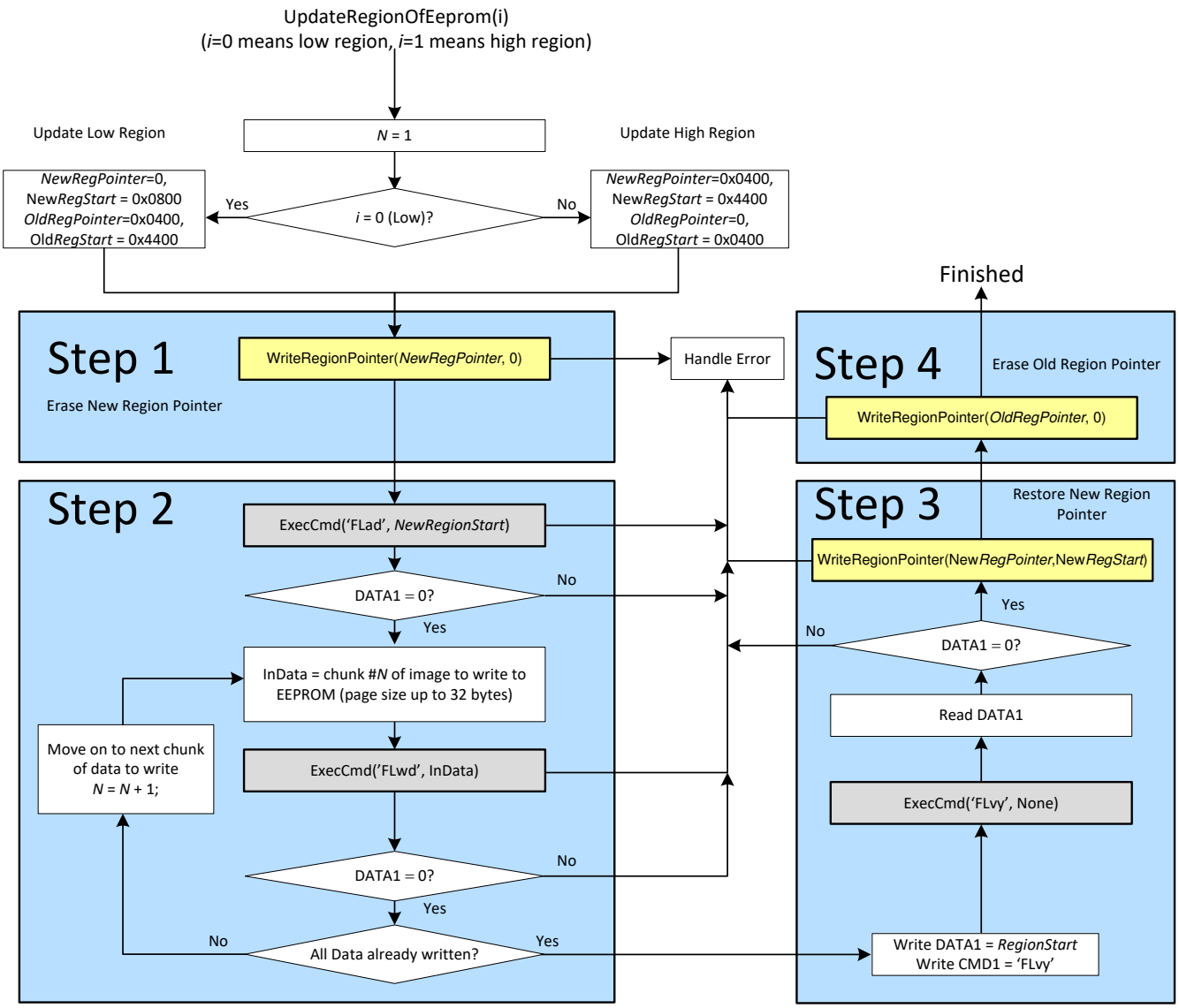

图 2-3. 用于更新 EEPROM 的 UpdateRegionOfEeprom() 函数的详细信息

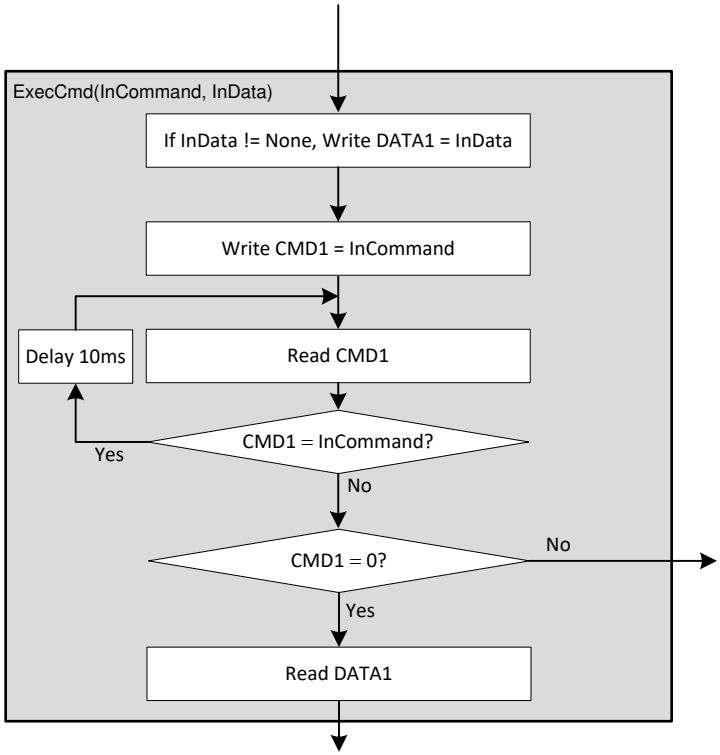

图 2-4. ExecCmd() 块的详细信息

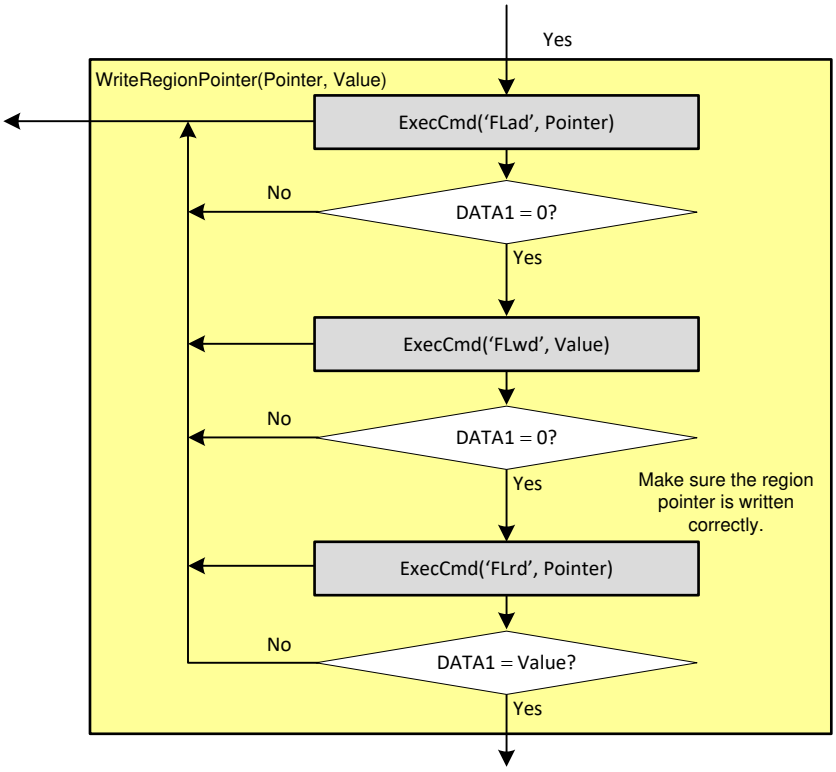

图 2-5. WriteRegionPointer() 块的详细信息

2.3 命令

EEPROM 更新过程使用表 2-1 中列出的 4CC ASCII 命令。

表 2-1. 4CC ASCII 命令

4CC 命令的名称	ASCII	输入 DataX 长度（以字节为单位）	输出 DataX 长度（以字节为单位）	说明
闪存存储器读取	FLrd	4	16	*FLrd* 命令读取指定地址处的闪存
闪存存储器写入起始地址	FLad	4	1	*FLad* 命令设置起始地址以准备进行闪存写入
闪存存储器写入	FLwd	64	1	*FLwd* 命令从 *FLad* 命令定义的闪存起始地址开始写入数据。该地址自动递增
闪存存储器验证	FLvy	4	1	*FLvy* 命令验证补丁或配置是否有效
冷复位请求	GAID	0	0	*GAID* 命令使 PD 控制器处理器冷重启。*GAID* 命令用于在固件更新过程结束时重新启动 TPS25751/TPS26750，并从非易失性闪存存储器重新加载新的固件版本

若要执行 4CC 命令，主机应用程序应遵循以下顺序：

1. 如果 4CC 命令需要输入，应用程序应首先将输入数据写入 Data1 (0x09) 寄存器。
2. 应用程序随后应将 4CC 命令字符写入相应的 Cmd1 (0x08) 寄存器。
3. 应用程序应等到读取到 Cmd1 寄存器的以下四个字节内容，才能轮询或者设置并使用 *Cmd1Complete* 事件：
   - *0x00*，表示命令执行成功。
   - 或 *CMD*，表示命令执行失败。

如果命令成功执行，应用程序将继续读取包含输出的 Data1 寄存器 *n* 字节内容。有关 4CC 命令的更多详细信息，请参阅 *TPS25751 技术参考手册* 器件的主机接口 TRM。

2.4 EEPROM 更新示例

在以下示例中，假设 EEPROM 的初始状态是低区和高区都具有相同的补丁捆绑包。PD 控制器从低区进行引导。图 2-6 展示了针对该初始条件的 EEPROM 存储器映射。

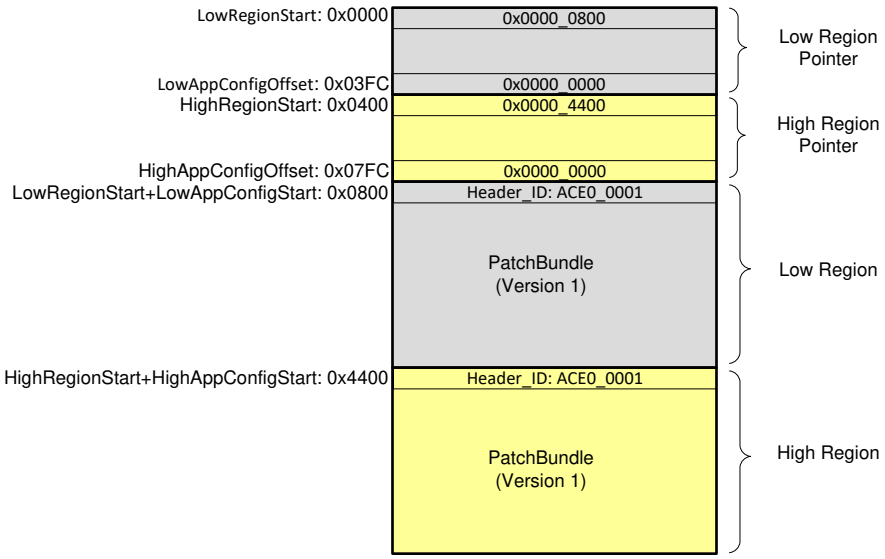

图 2-6. EEPROM 的初始状态

当主机必须更新 PD 控制器用于引导的补丁捆绑包时，首先会擦除高区指针，这样一来，如果在将新的补丁捆绑包写入高区时出现中断，可以保证 PD 控制器不会尝试加载这个补丁捆绑包。具体而言，会执行图 2-3 中的 UpdateRegionOfEeprom(1) 函数的步骤 1，图 2-7 展示了成功擦除高区指针后的存储器映射。如果在此状态下引导，则会从低区加载补丁捆绑包（版本 1）。

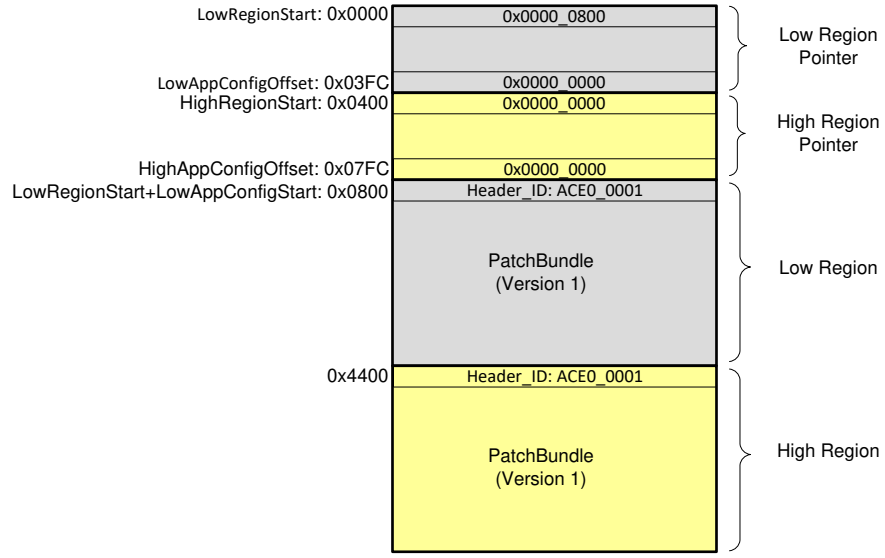

图 2-7. 执行 UpdateRegionOfEeprom(1) 步骤 1 之后的 EEPROM 状态

接下来，主机将新的补丁捆绑包（版本 2）写入高区。具体而言，会执行图 2-3 中的 UpdateRegionOfEeprom(1) 函数的步骤 2，图 2-8 展示了执行此步骤后的存储器映射。请注意，如果 PD 控制器使用此状态下的 EEPROM 进行引导，则仍会从低区加载补丁捆绑包（版本 1）。

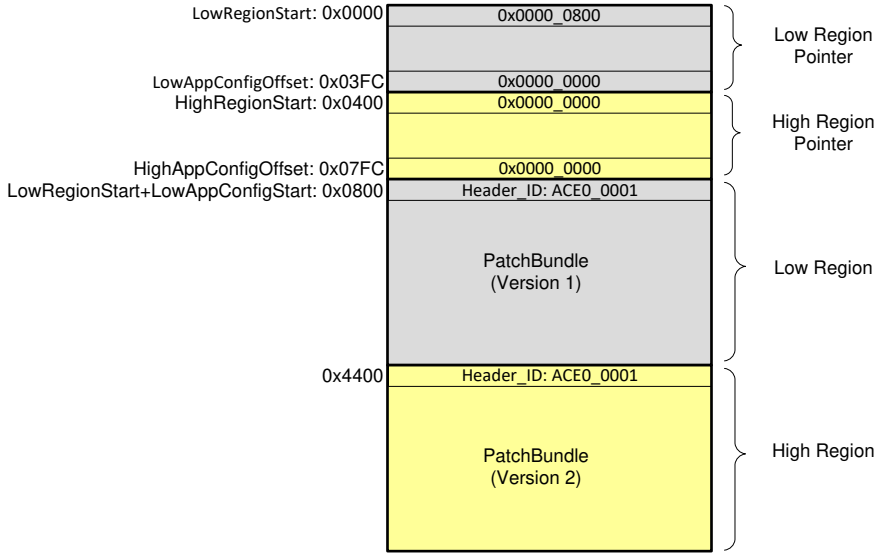

图 2-8. 执行 UpdateRegionOfEeprom(1) 步骤 2 之后的 EEPROM 状态

接下来，主机使用 *FLvy* 命令验证 EEPROM 高区的内容。如果成功，则主机将 0x4400 写入到地址为 0x0400 的 HighRegionStart。此过程发生于图 2-3 中的 UpdateRegionOfEeprom(1) 的步骤 3，图 2-9 展示了执行此步骤后的存储器映射。如果 PD 控制器使用此状态下的 EEPROM 进行重新启动，则仍会先尝试从低区进行引导。

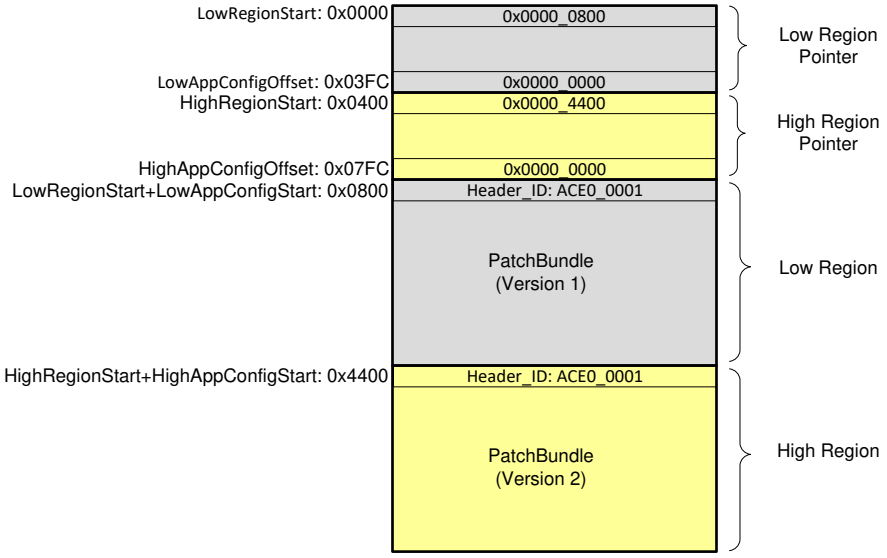

图 2-9. 执行 UpdateRegionOfEeprom(1) 步骤 3 之后的 EEPROM 状态

最后一步是擦除 LowRegionStart 值，以便 PD 控制器从高区引导。主机可以使用图 2-3 中所示的 UpdateRegionOfEeprom(1) 步骤 4 中提到的 WriteRegionPointer() 功能。图 2-10 展示了步骤 4 完成后的存储器映射。因为 LowRegionStart 现在为 0，所以低区的内容对 PD 控制器的引导方式没有影响。

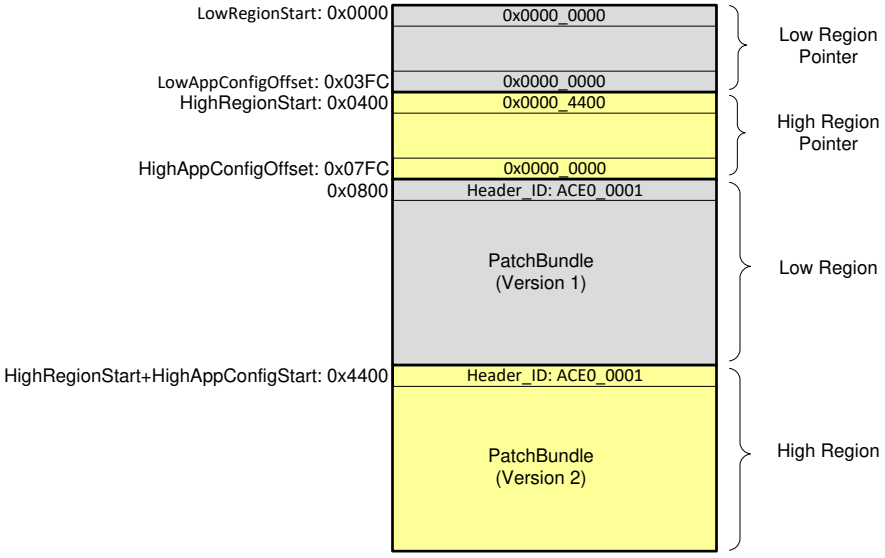

图 2-10. 执行 UpdateRegionOfEeprom(1) 步骤 4 之后的 EEPROM 状态

下次主机须更新 EEPROM 映像时，它会执行 UpdateRegionOfEeprom(0)，该过程以类似的方式进行。主机可以选择使用刚刚写入高区的相同新补丁捆绑包立即执行 UpdateRegionOfEeprom(0)。

3 源代码示例

本节给出了实现上述 EEPROM 流程的源代码示例。

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 从 EEPROM 故障中恢复

如果 EEPROM 加载过程因没有有效补丁捆绑包而终止，则 INT_EVENTx.ReadyForPatch 中断将生效。主机必须读取 BOOT_STATUS 寄存器 0x2D 才能发现从 EEPROM 引导失败的原因。然后，主机必须通过使用 ***PBMx*** 命令推送补丁来强制 PD 控制器进入 APP 模式（有关详细信息，请参阅技术参考手册）。这个补丁可以是通常位于 EEPROM 中的完整补丁捆绑包。PD 控制器处于 APP 模式后，主机可以使用 ***FLxx*** 命令来写入 EEPROM 并纠正问题。图 4-1 显示了推荐的引导流程。

此引导流程要求主机能够纠正 EEPROM。如果主机需要 PD 控制器在引导前启用受电路径，则必须通过 ADCINx 引脚选择相应的电池电量耗尽配置。在这种情况下，SafeMode 电池电量耗尽配置不适用。

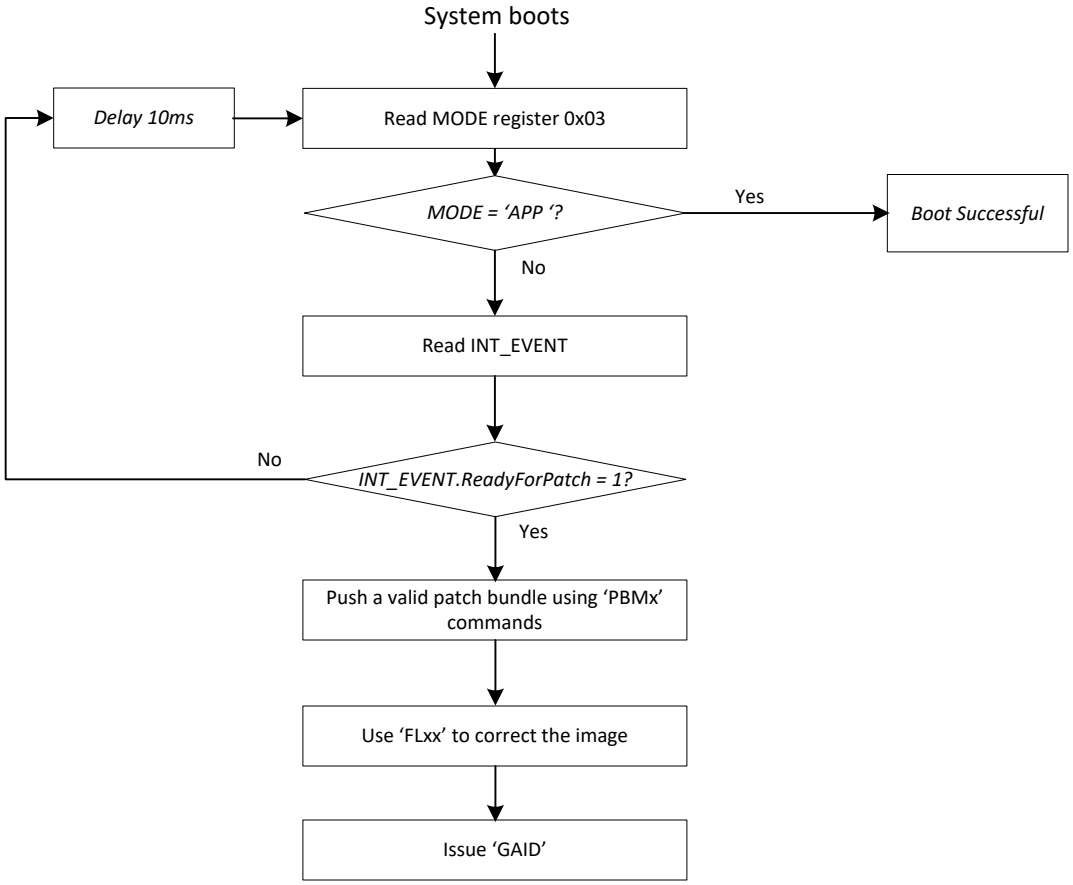

图 4-1. 推荐的 EEPROM 引导流程

5 结语

TPS25751 和 TPS26750 应用程序二进制文件可以使用 I2Ct 端口通过 I2C 推送到 PD 控制器，也可由 PD 控制器从外部 EEPROM 读取。主机须更新用于引导的补丁捆绑包时，必须遵循一定的顺序。

主机应按以下顺序来更新补丁捆绑包：

- 使用 *FLrd* 命令，在器件上查询外部 EEPROM 上待更新区域的地址。
- 然后，使用 *FLad* 命令设置下一次写入的起始地址，并使用 *FLwd* 命令开始一次性发送补丁捆绑包 32 个字节。
- 成功执行 *FLwd* 后，器件会自动递增写入地址，主机无需为每个写入请求设置起始地址。
- 然后，使用 *FLvy* 命令验证 EEPROM 的内容。
- 更新这两个区域后，应使用 *GAID* 命令对器件进行冷复位。器件可以再次执行引导序列，并加载更新后的补丁捆绑包。

6 参考资料

- 德州仪器 (TI)，*TPS25751 技术参考手册*。
- 德州仪器 (TI)，*TPS26750 技术参考手册*。

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265