XDPP1100 firmware patch programming multiple partitions

# Scope and purpose

This document provides the details on procedure to program configuration and Firmware patch into OTP of XDPP1100 device.

# Intended audience

This document is intended to Engineer interested in programming configurations and firmware patches to XDPP1100’s OTP.

Table of contents

[Table of contents 1](#_Toc62217473)

[1 Introduction 2](#_Toc62217474)

[1.1 OTP Organization 2](#_Toc62217475)

[1.2 Programming multiple patches 3](#_Toc62217476)

[2 Storing patch in same partition 4](#_Toc62217477)

[2.1 Invalidating a patch 4](#_Toc62217478)

[2.1.1 Invalidating a patch through GUI 4](#_Toc62217479)

[2.1.2 Invalidating a patch programmatically 5](#_Toc62217480)

[3 Storing patches in multiple partitions 6](#_Toc62217481)

[3.1 Storing first patch in to partition 1 7](#_Toc62217482)

[3.2 Storing a second patch in to partition 2 7](#_Toc62217483)

[3.2.1 Storing different FW patches at different OTP partitions 7](#_Toc62217484)

[3.2.2 STEP 1a: Update linker\_config.sct file for ARM-CC compiler 7](#_Toc62217485)

[3.2.3 STEP 1b: Update linker\_config.ld for GCC compiler 10](#_Toc62217486)

[3.2.4 STEP 2: Update Patch Entry 12](#_Toc62217487)

[3.2.5 STEP 3: Modify Makefile 13](#_Toc62217488)

[3.2.6 STEP 4: Build the project 14](#_Toc62217489)

[3.2.7 Update patch\_init.c file 15](#_Toc62217490)

[3.2.8 Update partial\_patch.cfg file 16](#_Toc62217491)

[Revision history 17](#_Toc62217492)

# Introduction

The Infineon “XDPP” family of Digital Controllers contains a built-in micro-controller for feature enhancement or quick bug fixes. The micro-controller firmware (FW) can be updated or “patched” over the industry standard I2C serial interface and saved in internal non-volatile memory (NVM). This document describes how to upload a firmware patch over the I2C serial interface and also store a configuration to non-volatile memory (NVM).

## OTP Organization

A total of 64KB of “OTP” nonvolatile memory space is available for the user to store trim, configurations and patches. By default 16KB of memory is allocated for data partition to store trim and configurations. Remaining 48KB is allocated for patch partitions to store the patches. These sizes of data partition and patch partitions are configurable and should be kept unchanged after a patch is stored in OTP.

Patch partitions can further partitioned up to 16 for storing multiple patches. In a given partition there can only be one active patch. So, to store a new patch in a partition, invalidate the patch before storing a new patch.

The OTP memory space is shared with partial configuration data and depending on the size of the patch, the procedure described in this document can be used to load a firmware patch into devices which may or may not already contain a previously loaded patch. Care must be taken not to exceed the memory allowance when uploading a new patch.

|  |
| --- |
|  |

1. OTP partition

## Programming multiple patches

XDPP1100 makes use of OTP (One-Time Programmable) memory. Due to the nature of OTP, XDPP1100 memory has to be partitioned in order to optimize the space. OTP is partitioned in several partitions to support storing the configurations and also FW patches. Data partition is the first partition by default set to 16 KB of space and then a patch partition of 48 KB. User can change the patch partition sizes and also add a new patch partition. User can perform up to 16 partitions in OTP memory.

Each OTP partition can only store one active firmware patch. In order to store a new firmware patch to the OTP, user has two options:

1. Update the existing patch project with the newly added features, invalidate the old active patch and then store the updated patch in to the OTP. The procedure for this option are described in chapter 2.
2. Create a new patch project for the newly added features only, store this new patch into a different partition while keeping the old patch active in Partition 1. This process keeps two active patches in two different partitions. The procedure for this option are described in chapter 3

# Storing patch in same partition

Each OTP partition can only store one active firmware patch. Each partition may have multiple invalidated patches but can only handle one active patch. In order to store a new firmware patch in to the same partition, user has to invalidate the existing patch and then perform storing a new patch in to the same partition. The steps are below

* Update the existing patch project with the newly added features and create a patch.bin file.
* Invalidate the old active patch in the partition.
* Check if there is enough space remaining in the partition to store the newly created patch. If there is not enough space, increase the partition size by writing the new OTP partition sizes in to OTP.
* Store the updated patch in to the OTP in the same partition.

## Invalidating a patch

### Invalidating a patch through GUI

|  |
| --- |
|  |

1. Invalidating a patch through GUI

### Invalidating a patch programmatically

|  |
| --- |
| Try  **// 1. Disable the MMU**  **//Store the boot descriptor in to temp variable , to load back again at the end of the function**              status = i2cRead4Bytes(boot\_descrptor\_addr, data2)  **//disabling mmu by writing 0xF in to boot descriptor register**              status = i2cWrite4Byteswithdata("0000000F", boot\_descrptor\_addr)  **// 2 .Reset after disabling the MMU**              status = pmbWriteByte (cmdMFR\_FIRMWARE\_COMMAND, CByte(&HE))  **// 3 .pass the parameters to FIRMWARE COMMAND DATA - PMBUs command to invalidate the patch**              writeData(0) = &H10         **// patch file command type ( 0x10 )**              writeData(1) = 0              writeData(2) = 0              writeData(3) = partition\_number **// partition number – to invalidate partition – “1”**              status = pmbWriteBlock(cmdMFR\_FIRMWARE\_COMMAND\_DATA, writeData1)  **// 4 .Execute the PMBUs command (FIRMWARE COMMAND) to invalidate the patch – pass the argument as 0x12 to invalidate**              status = pmbWriteByte(cmdMFR\_FIRMWARE\_COMMAND, &H12)  **// 5. Read back the status to check if the invalidate is success or fail. If status returned is 0 then success else failed.**              data = pmbReadBlock(cmdMFR\_FIRMWARE\_COMMAND\_DATA, 4)              If (data(0) <> 0) Then                  MessageBox.Show("Invalidating the patch failed")             End If          Catch ex As Exception     MessageBox.Show("Invalidating patch failed")          Finally  **// 6. Enable the MMU by restoring the boot descriptor – which is saved in the first step in to data2.**              status = i2cWrite4Bytes(data2, boot\_descrptor\_addr, &H10)  **// 7 .Reset after enabling the MMU**              status = pmbWriteByte(&H80, cmdMFR\_FIRMWARE\_COMMAND, CByte(&HE))    End Try |

# Storing patches in multiple partitions

We need to modify the patch partition sizes to store two or more patches in to OTP. For example, if user wants to store two patches, errata patch (bug fixes) and patch user app (development patch). The errata patch has the firmware fixes of the errata. The size of the errata patch is 0x800 and we store this patch to partition 1.

The development patch needs to be stored in patch partition 2. So, we set the remaining size to partition 2, which is 0xB800. In the below picture, we set

* data partition to store the configurations of size 0x4000 ( 16 KB)
* patch partition 1 to store the errata patch of size 0x0800 ( 2 KB)
* patch partition 2 to store the development patch of size 0xB800 ( 46 KB)

|  |
| --- |
|  |

1. OTP partition GUI display

Click on store trim to modify the partition sizes.

## Storing first patch in to partition 1

Load the patch\_errata.bin file to the GUI and select the partition 1 and click on “Store OTP Patch”. This saves the errata patch to partition 1.

|  |
| --- |
|  |

1. Store patch to partition 1 GUI display

## Storing a second patch in to partition 2

The developed patch project needs to be update for building the patch for partition 2. So that, User could store to the partition 2. By default the project provided builds for partition 1.

We need to update this project to build for partition 2 following the below steps.

### Storing different FW patches at different OTP partitions

This chapter describe on how to create a firmware patch as per described in Option 2 above.

1. The following steps are done after user has implemented the desired features in a newly created patch project. It is therefore assumed that user already know how to create a new firmware patch.

### STEP 1a: Update linker\_config.sct file for ARM-CC compiler

The file can be found in $PROJ/src/ folder as shown in the following figure.

|  |
| --- |
|  |

1. File location of linker\_config.sct

Scroll down until you see the following lines of codes.

|  |
| --- |
|  |

1. Linker\_Config.sct

Update the linker\_config.sct to build for Partition 2:

|  |
| --- |
|  |

1. Modification of linker\_config.sct for Partition 2

Note: The above example is for 4 equal partitions, and we use second partition to store the patch.

### STEP 1b: Update linker\_config.ld for GCC compiler

Similarly, this file can be found in $PROJ/src/ folder as shown in the following figure.

|  |
| --- |
|  |

1. File location of linker\_config.ld

Scroll down until you see the following line:

|  |
| --- |
|  |

1. Linker\_config.ld

Update the linker\_config.ld to build for Partition 2:

|  |
| --- |
|  |

1. Modification of linker\_config.ld for Partition 2

### STEP 2: Update Patch Entry

Open patch\_init.c. The file can be found in $PROJ/src/ folder.

|  |
| --- |
|  |

1. File location of patch\_init.c

Scroll down until you can find the following lines:

|  |
| --- |
|  |

1. Patch\_init.c

Update patch\_entry:

|  |
| --- |
|  |

1. Modification of patch\_init.c

### STEP 3: Modify Makefile

Open Makefile. The file can be found in $PROJ/ folder.

|  |
| --- |
|  |

1. File location of Makefile

Search for the following lines:

|  |
| --- |
|  |

1. Makefile

Modify as following:

|  |
| --- |
|  |

1. Modification of Makefile

### STEP 4: Build the project

Build the project to generate the patch file.

|  |
| --- |
|  |

1. Build project

After building the patch for partition 2. Store the patch in to partition 2 using the GUI tool like Figure 18.

|  |
| --- |
|  |

1. Store Patch to OTP partition 2

### Update patch\_init.c file

This update is needed to update the base version of the developed patch to match against the new version of the patch errata. Change the version number like below shown in bold.

|  |
| --- |
| void patch\_entry(void)  {  // Initialize the .bss and .data section  memset(SECTION\_BASE\_ZI, 0, SECTION\_LENGTH\_ZI);  memcpy(SECTION\_BASE\_RW\_DESTINATION, SECTION\_BASE\_RW\_SOURCE, SECTION\_LENGTH\_RW);  // Initialize ram execution section  memcpy(SECTION\_BASE\_RAM\_EXEC\_DESTINATION, SECTION\_BASE\_RAM\_EXEC\_SOURCE, SECTION\_LENGTH\_RAM\_EXEC);  SCU\_SPARE\_FF\_\_SET(**0x5971BBBBUL**); // write the patch id to SCU spare for simple test that patch loaded  user\_drv\_init();  } |

### Update partial\_patch.cfg file

Update this file so the base version of the patch match to the new version of the errata patch. and create a new version with UTC ( to track for the date and time).

|  |
| --- |
| [Patch]  startaddress = 0x20063C00  endaddress = 0x20063fff  minramsize = 0x300  [Version]  baseversion = 0x5F3C1F0D  newversion = 0x5971BBBB |

Revision history

| Document version | Date of release | Description of changes |
| --- | --- | --- |
| 1.0 | 2020-08-24 | Initial version on the firmware Patch programming to OTP partition 2 |