XDPP1100 firmware patch programming multiple partitions

Scope and purpose

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

Intended audience

This document is intended for engineers who are interested in programming configurations and firmware patches to XDPP1100's OTP.

Table of contents

Table of contents **1**
1 Introduction **2**
1.1 OTP organization 2
1.2 Programming multiple patches 3
2 Storing patch in same partition **4**
2.1 Invalidating a patch 4
2.1.1 Invalidating a patch through GUI 4
2.1.2 Invalidating a patch programmatically 5
3 Storing patches in multiple partitions **6**
3.1 Storing first patch in to partition 1 7
3.2 Storing a second patch in to partition 2 7
3.2.1 Storing different FW patches at different OTP partitions 7
3.2.2 STEP 1a: Update linker_config.sct file for ARM-CC compiler 7
3.2.3 STEP 1b: Update linker_config.ld for GCC compiler 10
3.2.4 STEP 2: Update Patch Entry 12
3.2.5 STEP 3: Modify Makefile 13
3.2.6 STEP 4: Build the project 14
3.2.7 Update patch_init.c file 15
3.2.8 Update partial_patch.cfg file 16
Revision history **17**

1 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).

1.1 OTP organization

A total of 64 KB of “OTP” nonvolatile memory space is available for the user to store trim, configurations and patches. By default 16 KB of memory is allocated for data partition to store trim and configurations. Remaining 48 KB 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. Therefore, to store a new patch in a partition, invalidating the patch before storing a new patch is necessary.

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.

Attention: ***Care must be taken not to exceed the memory allowance when uploading a new patch.***

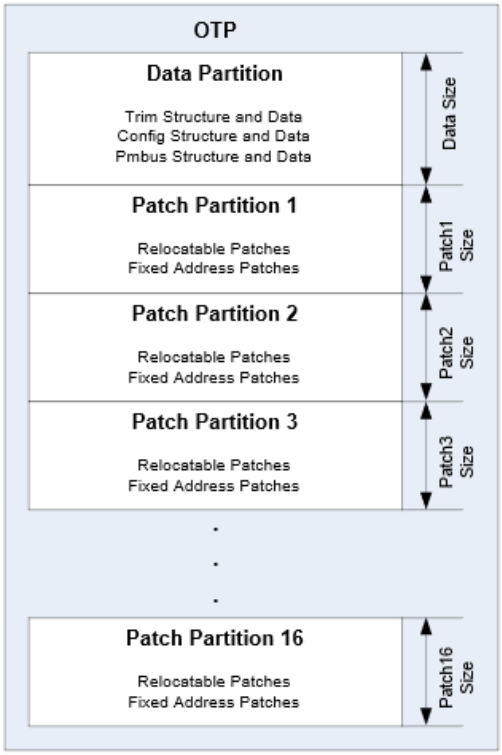

Figure 1 OTP partition

1.2 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.

2 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. **Figure 4** will demonstrate on storing a patch to OTP.

2.1 Invalidating a patch

2.1.1 Invalidating a patch through GUI

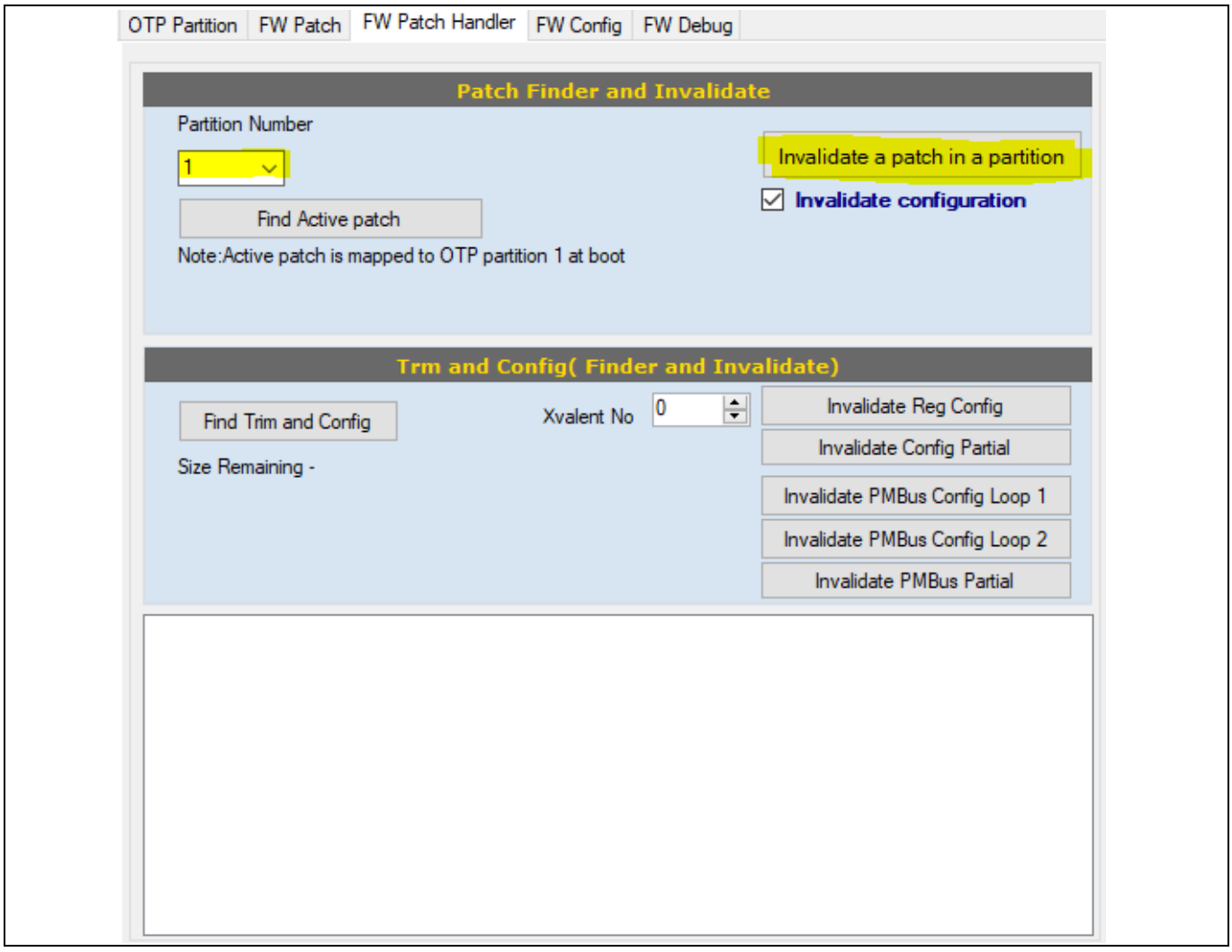

Figure 2 Invalidating a patch through GUI

2.1.2 Invalidating a patch programmatically

Code Listing 1

```
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
```

3 Storing patches in multiple partitions

We need to modify the patch partition sizes to store two or more patches in to OTP. Supposing, the 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 and the size of the errata patch is 0x800 therefore we store this patch to partition 1.

The development patch needs to be stored in patch partition 2. Thus, 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)

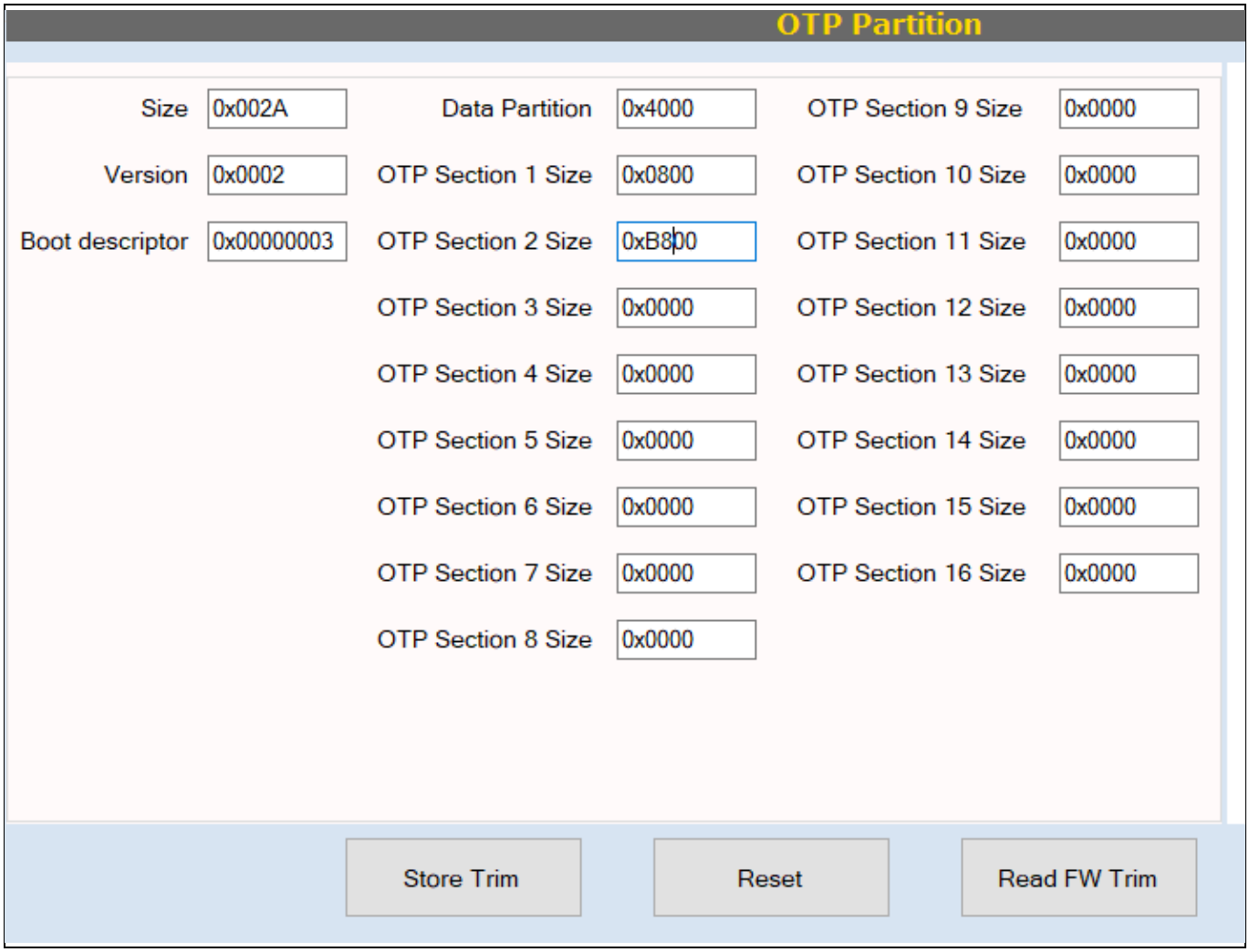

Figure 3 OTP partition GUI display

Click on store trim to modify the partition sizes.

3.1 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.

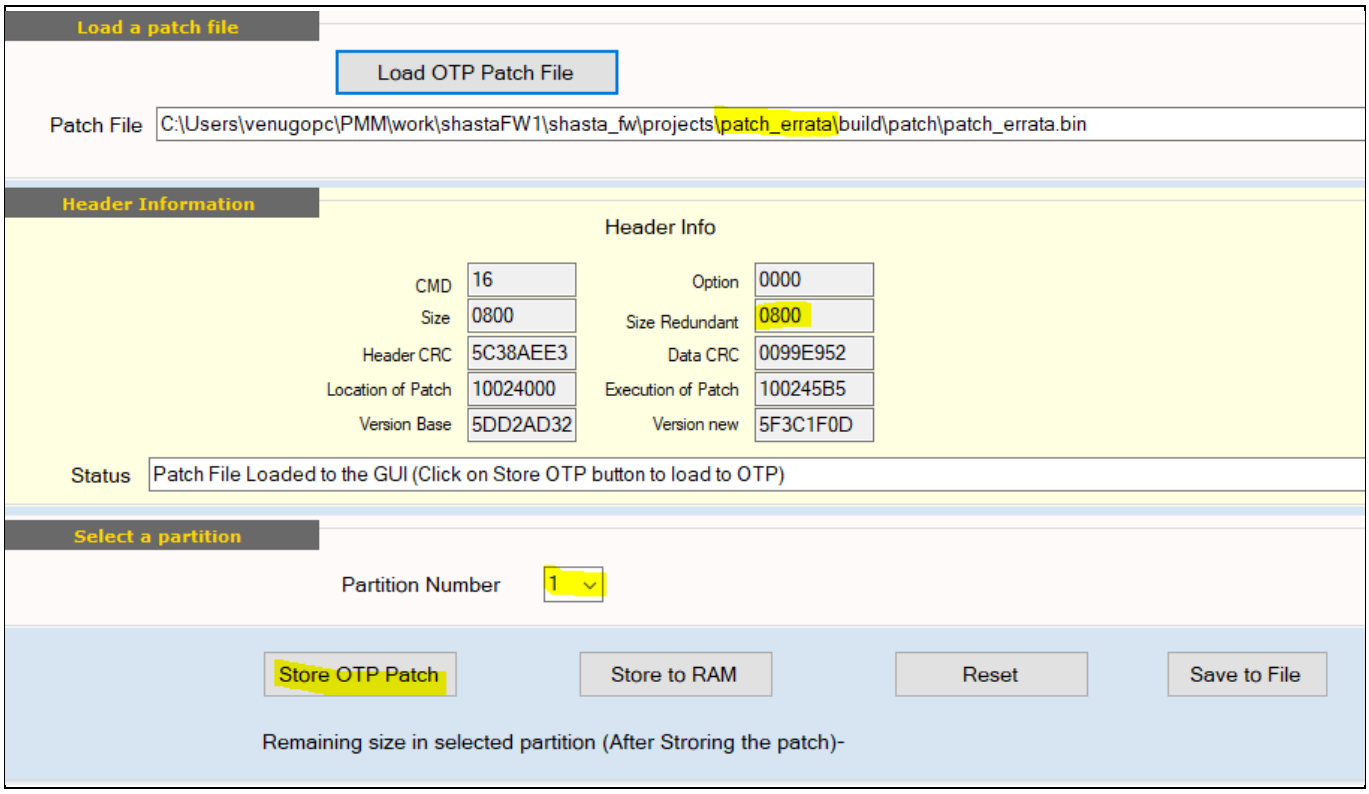

Figure 4 Store patch to partition 1 GUI display

3.2 Storing a second patch in to partition 2

The developed patch project needs to be updated for building the patch for partition 2. Because of 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.

3.2.1 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.

Note: 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.

3.2.2 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.

Storing patches in multiple partitions

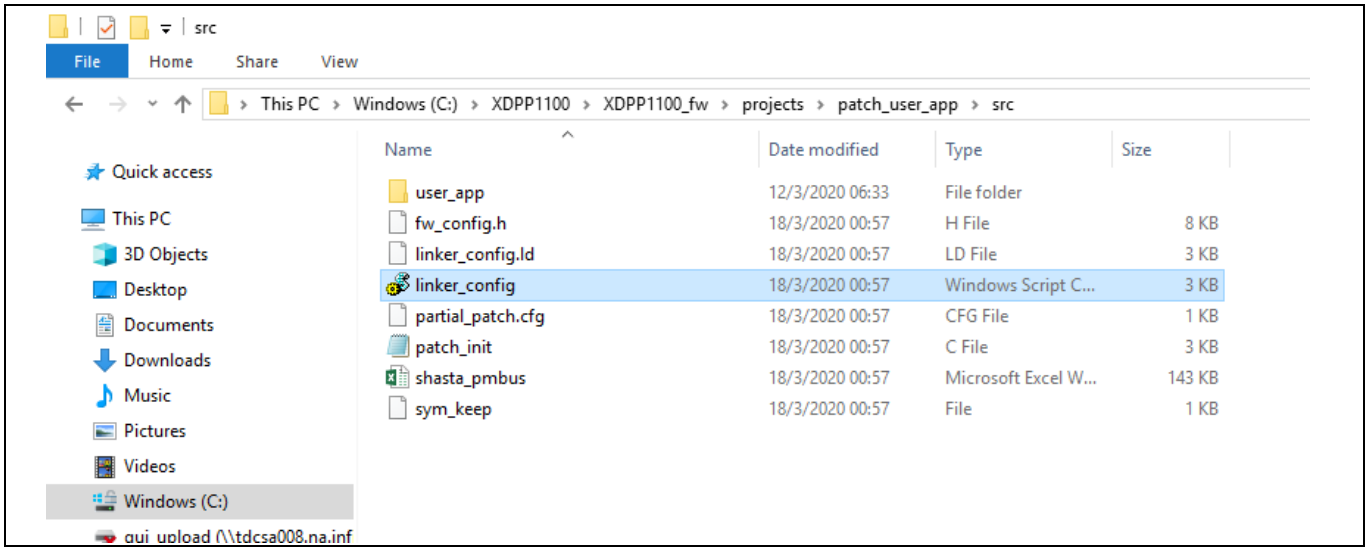

Figure 5 File location of linker_config.sct

Scroll down until you see the following lines of codes.

```
; Here we define the patch size we allocate 4 patch regions, each 4k in size this means we
; 16k for data
#define otp_data_base   otp_base
#define otp_data_size   0x4000
; 16k each for patches
#define otp_patch1_base   otp_data_base + otp_data_size
#define otp_patch1_size 0x4000
#define otp_patch1_effective_size otp_patch1_size - otp_versioned_patch_header_size
#define otp_patch2_base   otp_patch1_base + otp_patch1_size
#define otp_patch2_size 0x4000
#define otp_patch3_base   otp_patch2_base + otp_patch2_size
#define otp_patch3_size 0x4000
#define otp_patch4_base   otp_patch3_base + otp_patch3_size
#define otp_patch4_size 0x0


; Needs to be filled out according to the needs of the patch
PATCH_LOAD otp_patch1_base + otp_versioned_patch_header_size otp_patch1_effective_size
{

    OTP_PATCHES otp_patch1_base + otp_versioned_patch_header_size otp_patch1_effective_size
    {
        * (+RO)         ; (.text)
    }

    RAM_INIT LINKER_RAM_RANGE_START LINKER_RAM_RANGE_SIZE
    {
        * (+RW, +ZI)    ; (.data | .bss)
    }

        RAM_EXEC +0
    {
                * (RAM_EXEC)
    }
}


; "*" means: from arbitrary objects
```

Figure 6 Linker_Config.sct

Storing patches in multiple partitions

Update the linker_config.sct to build for Partition 2:

```
; Here we define the patch size we allocate 4 patch regions, each 4k in size this means we
; 16k for data
#define otp_data_base   otp_base
#define otp_data_size   0x4000
; 16k each for patches
#define otp_patch1_base   otp_data_base + otp_data_size
#define otp_patch1_size 0x4000
#define otp_patch1_effective_size otp_patch1_size - otp_versioned_patch_header_size
#define otp_patch2_base   otp_patch1_base + otp_patch1_size
#define otp_patch2_size 0x4000
#define otp_patch2_effective_size otp_patch2_size - otp_versioned_patch_header_size
#define otp_patch3_base   otp_patch2_base + otp_patch2_size
#define otp_patch3_size 0x4000
#define otp_patch4_base   otp_patch3_base + otp_patch3_size
#define otp_patch4_size 0x0


; Needs to be filled out according to the needs of the patch
PATCH_LOAD otp_patch2_base + otp_versioned_patch_header_size otp_patch2_effective_size
{

    OTP_PATCHES otp_patch2_base + otp_versioned_patch_header_size otp_patch2_effective_size
    {
        * (+RO)         ; (.text)
    }

    RAM_INIT LINKER_RAM_RANGE_START LINKER_RAM_RANGE_SIZE
    {
        * (+RW, +ZI)    ; (.data | .bss)
    }

                RAM_EXEC +0
    {
                                * (RAM_EXEC)
    }
}


; "*" means: from arbitrary objects
```

Figure 7 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.

3.2.3 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.

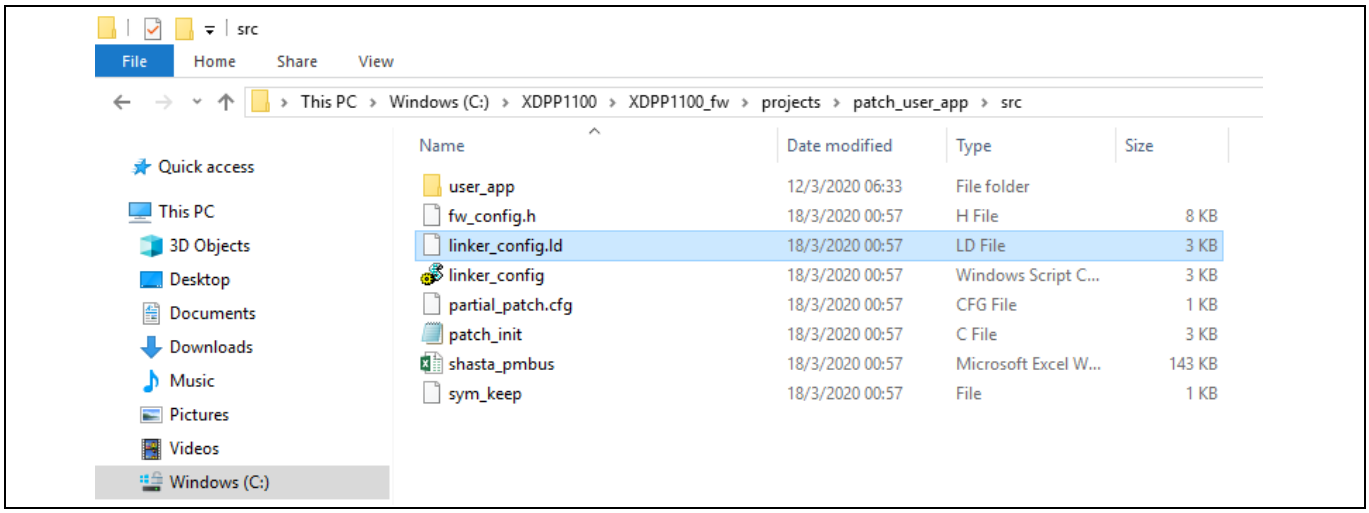

Figure 8 File location of linker_config.ld

Scroll down until you see the following line:

```
    /* Reserved space for the patch header.
           The patch region 1 follows the data region */
        .otp_header :
        {
          . += _otp_patch_header_size;
        } >otp

        /* Reserved space for the patch code */
        .txt :
    {
            *(.text*)
                *(.rodata*)
                *(.ARM.extab* .gnu.linkonce.armextab.*)
                *(.ARM.exidx* .gnu.linkonce.armexidx.*)
      _end_otp_patch1 = .;
           _end_text = .;
    } >otp

        /* Reserved space for functions to be executed in RAM instead of OTP */
        _ram_exec_source = ALIGN (4);

        .ram_exec : AT (_ram_exec_source)
        {
                _start_ram_exec = .;
                *(RAM_EXEC*)
                _end_ram_exec = .;
        } > ram

    . = ALIGN(4);

        /* These are examples for additional patch partitions
           They only need to be defined if needed. */
        .otp_patch2 :
    {
          _otp_patch2_base = .;
          . += _otp_patch2_size;




      _end_otp_patch2 = .;

    } >otp

        .otp_patch3 :
```

Figure 9 Linker_config.ld

Storing patches in multiple partitions

Update the linker_config.ld to build for Partition 2:

```
    /* Reserved space for the patch header.
           The patch region 1 follows the data region */
        .otp_header :
        {
          . += _otp_patch_header_size;
        } >otp

        /* Reserved space for the patch code */
        .txt :
    {
      _otp_patch1_base = .;
          . += _otp_patch1_size;


      _end_otp_patch1 = .;

    } >otp

        /* Reserved space for functions to be executed in RAM instead of OTP */
        _ram_exec_source = ALIGN (4);

        .ram_exec : AT (_ram_exec_source)
        {
                _start_ram_exec = .;
                *(RAM_EXEC*)
                _end_ram_exec = .;
        } > ram

    . = ALIGN(4);

        /* These are examples for additional patch partitions
           They only need to be defined if needed. */
        .otp_patch2 :
    {
          _otp_patch2_base = .;
          *(.text*)
          *(.rodata*)
          *(.ARM.extab* .gnu.linkonce.armextab.*)
          *(.ARM.exidx* .gnu.linkonce.armexidx.*)
      _end_otp_patch2 = .;
      _end_text = .;
    } >otp

        .otp_patch3 :
```

Figure 10 Modification of linker_config.ld for Partition 2

3.2.4 STEP 2: Update Patch Entry

Open patch_init.c. The file can be found in $PROJ/src/ folder.

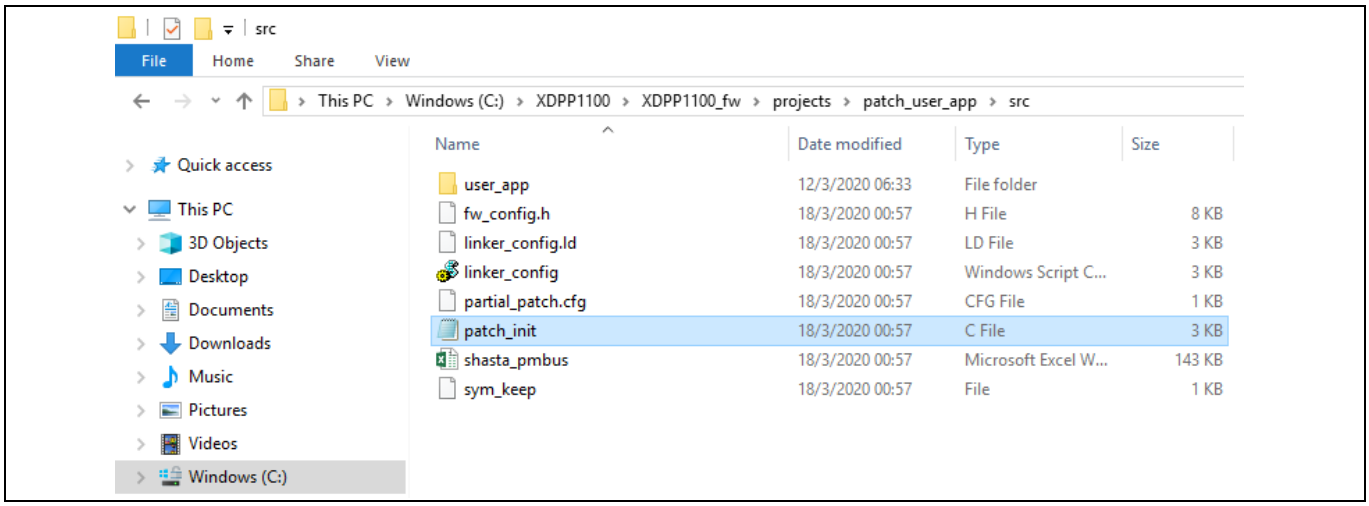

Figure 11 File location of patch_init.c

Scroll down until you can find the following lines:

```
43  * ------------------------------------------------------------------------- */
44  void patch_entry(void);
45  //STATIC void patch_install(void);
46
47  /* ------------------------------------------------------------------------- */
48
49  /*lint -esym(714, patch_entry)  Suppress: Symbol 'patch_entry' not referenced */
50  /*lint -e(957)  Suppress: Function 'patch_entry' defined without a prototype in scope. */
51  /*lint -esym(765, patch_entry)  Suppress: external 'patch_entry(void)' could be made static.
52   * No, its called from startup_opus. */
53  /**
54   * This is the patch entry point, which is called after the patch got
55   * successfully loaded into RAM.
56   */
57  void patch_entry(void)
58  {
59      // Initialize the .bss and .data section
```

Figure 12 Patch_init.c

Update patch_entry:

```
39  * ------------------------------------------------------------------------- */
40  void patch2_entry(void);
41  //STATIC void patch_install(void);
42
43  /* ------------------------------------------------------------------------- */
44
45  /*lint -esym(714, patch_entry)  Suppress: Symbol 'patch_entry' not referenced */
46  /*lint -e(957)  Suppress: Function 'patch_entry' defined without a prototype in scope. */
47  /*lint -esym(765, patch_entry)  Suppress: external 'patch_entry(void)' could be made static.
48   * No, its called from startup_opus. */
49  /**
50   * This is the patch entry point, which is called after the patch got
51   * successfully loaded into RAM.
52   */
53  void patch2_entry(void)
54  {
```

Figure 13 Modification of patch_init.c

3.2.5 STEP 3: Modify Makefile

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

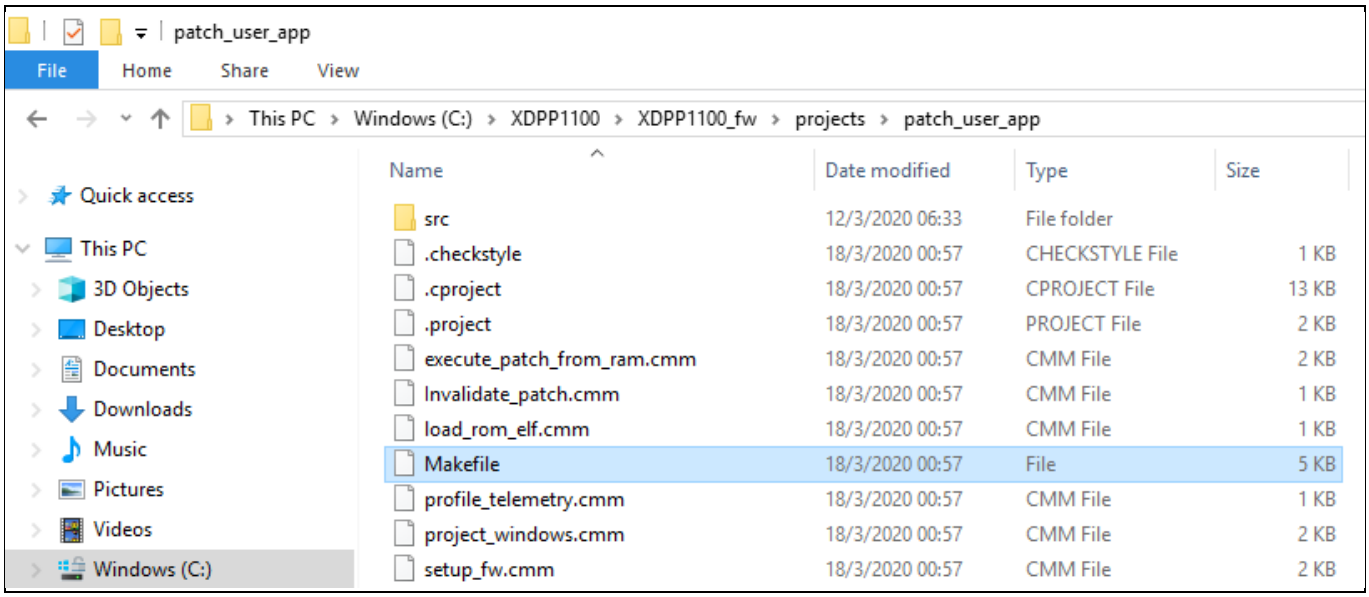

Figure 14 File location of Makefile

Search for the following lines:

```
# Reference Image for the patch targets. Not relevant for the image target.
PATCH_LINK_RANGE_CONFIG_FILE := ./src/partial_patch.cfg

# In case an entry point is needed by e.g. a debugger
LINKER_PARAMS += --entry=patch_entry

# Plugin for the targets "simvision"
include $(REPO_ROOT_DIR)/common/MakefileSimVision.mk
include $(REPO_ROOT_DIR)/common/MakefilePatching.mk
```

Figure 15 Makefile

Modify as following:

```
# Reference Image for the patch targets. Not relevant for the image target.
PATCH_LINK_RANGE_CONFIG_FILE := ./src/partial_patch.cfg

# In case an entry point is needed by e.g. a debugger
LINKER_PARAMS += --entry=patch2_entry

# Plugin for the targets "simvision"
include $(REPO_ROOT_DIR)/common/MakefileSimVision.mk
include $(REPO_ROOT_DIR)/common/MakefilePatching.mk
```

Figure 16 Modification of Makefile

3.2.6 STEP 4: Build the project

Build the project to generate the patch file.

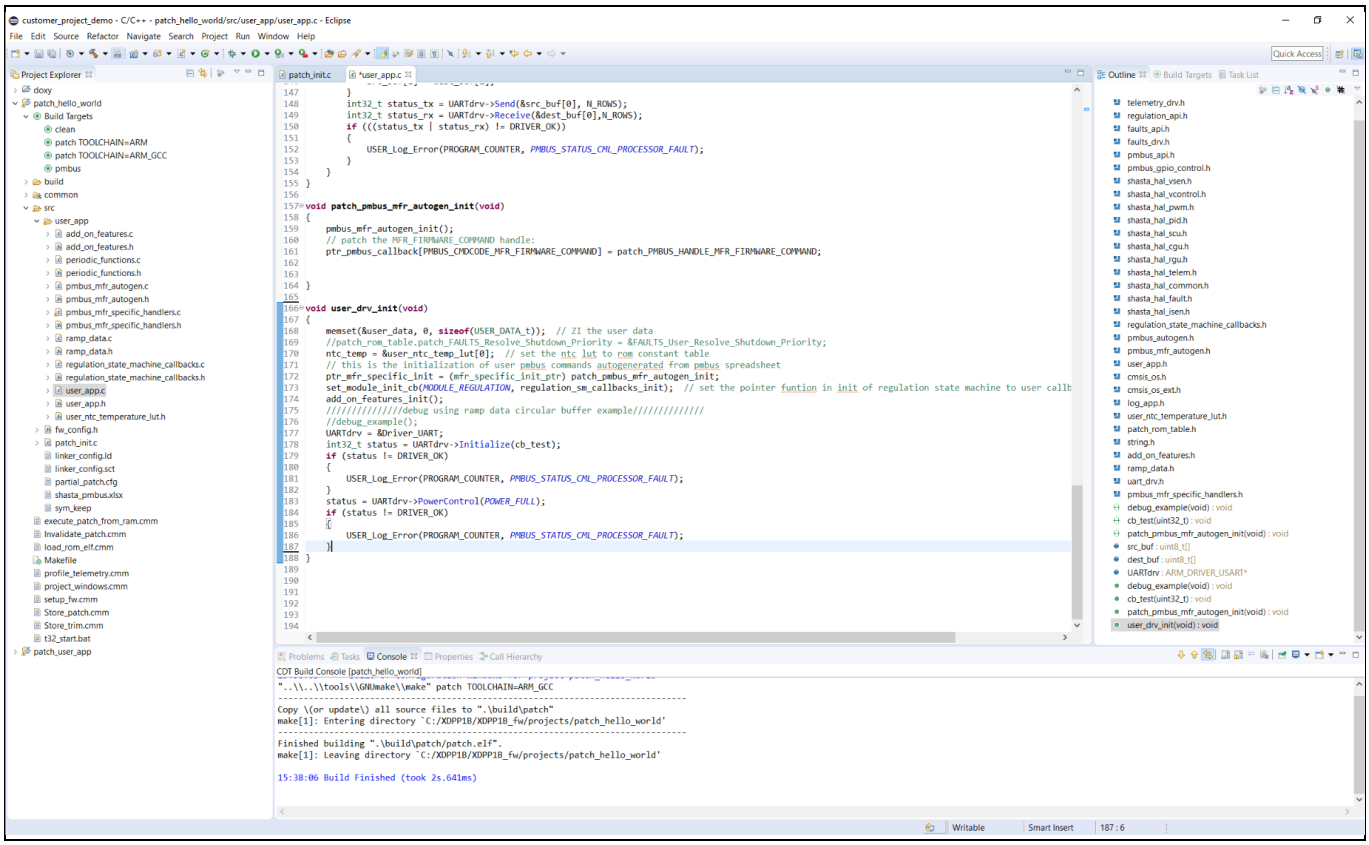

Figure 17 Build project

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

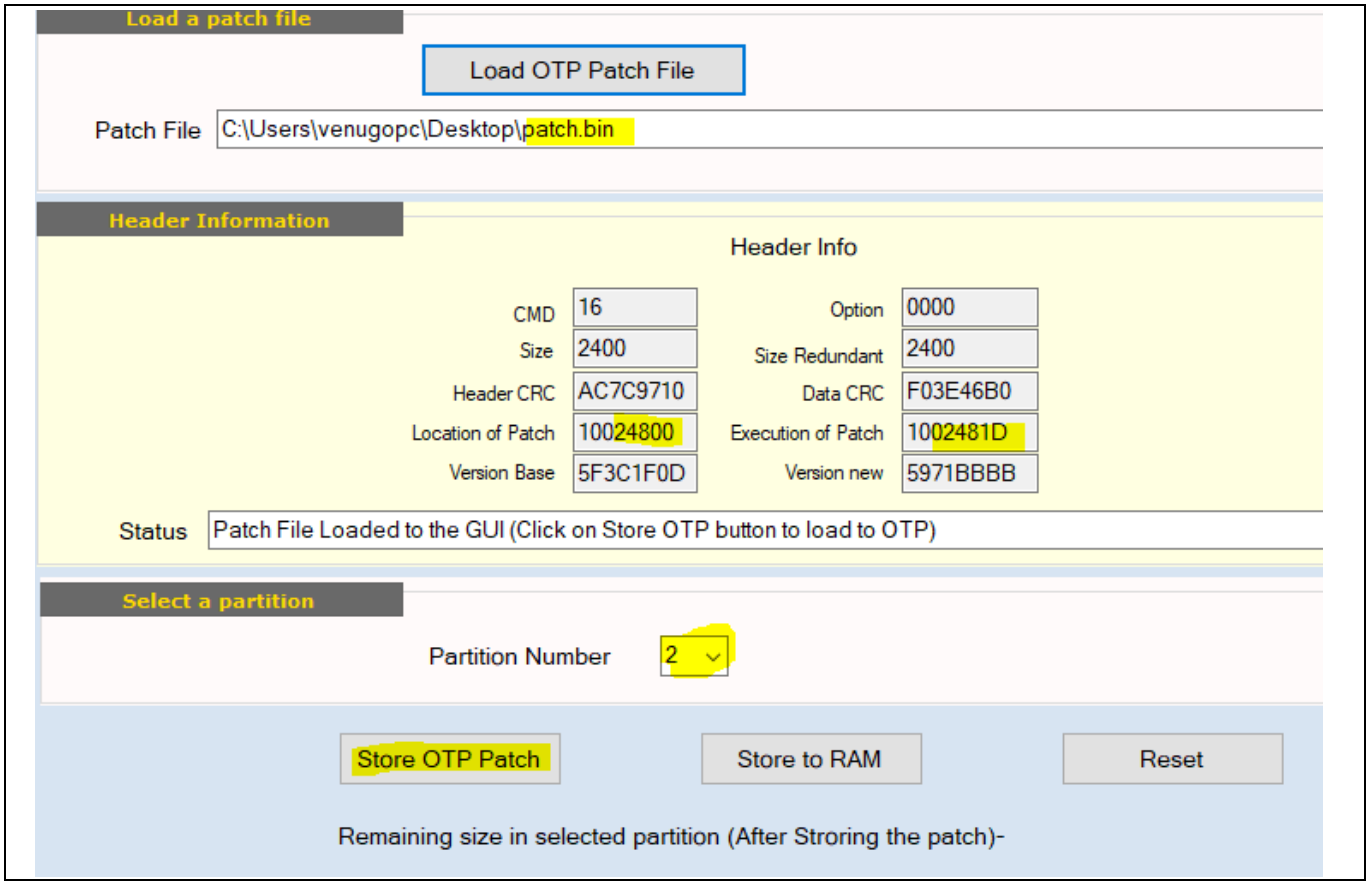

Figure 18 Store Patch to OTP partition 2

3.2.7 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.

Code Listing 2

```
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();
}
```

3.2.8 Update partial_patch.cfg file

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

Code Listing 3

```
[Patch]
startaddress = 0x20063C00
endaddress = 0x20063fff
minramsize = 0x300

[Version]
baseversion = 0x5F3C1F0D
newversion = 0x5971BBBB
```

Revision history

Document version	Date of release	Description of changes
V 1.0	05-02-2021	First release

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2021-02-05

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
AN_2101_PL88_2102_201614

IMPORTANT NOTICE

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.