XDPP1100 Firmware Examples Documentation

Jumpstart XDPP1100 Firmware Developments

About this document

Scope and purpose

This document describes how to develop custom firmware on XDPP1100 to extend its capabilities beyond the default firmware. The firmware example codes are grouped together based on the common use case in digital power conversion firmware development.

Intended audience

Power supply design engineers who wish to evaluate XDPP1100.

Table of contents

1 Introduction

In order to create a custom XDPP1100 firmware or to modify existed features, there is a way to do so by patches' implementation. Patches are used to enhance or replace ROM functions which executable code stored in OTP.

Formatted: Justified

For exemplification the general patch project is presented in **example_user_app**. This example shows a generalic way of XDPP1100 code structure and relationship between different features, which are combined together to showcase it as a custom firmware. There are several features are implemented into the project for a learning purpose like "*Frequency Ramp for Peak Current Mode Control Startup*", "*Fan Control*", "*Frequency Dithering*", "*Board Trim (as MFR_BOARD_TRIM)*", "*Current Share*", etc.. Each of these features is described into its section in this document accordingly.

Other examples have own code implementation in a separated patch project according to their section name and description. As an example "*Hello World*" can be found in **example_hello_world**.

In projects with a single feature, PMBus Spreadsheet shasta_pmbus.xlxx was simplified (Picture below) to highlight new PMBus commands, which are in the use in the particular example project.

	A	B	C	D	E	F	G	L	U	AF
1	Opco	Command	Wr. Tx	Rd. Tx	#B	M	Loop 0 Suppo	Loop 1 Support	HAS FW HANDLE	Description
179	B1	USER_DATA_01	Block Write	Block Read	2		n	n	n	
180	B2	MFR_DEADTIME	Write Byte	Read Byte	1	y	y	y	y	Deadtime when the output voltage is at target. LSB = 1.25 ns
181	B3	MFR_DBG_DEADTIME	Write Byte	Read Byte	1	y	y	y	y	Deadtime debug
182	B4	USER_DATA_04	Block Write	Block Read	2		n	n	n	

Figure 1 Example of simplified shasta_pmbus.xlxx based on deadtime manipulation example

The user might unhide other PMBus commands to have them as an example to create a custom MFR PMBus command.

Formatted: Justified

There are several recommendations:

- Start explore with "hello" examples. After that, understand examples 5.1 "Efficiency Look-Up Table and Input Current Correction Look-Up Table" and 4.1 "Open-loop soft-startup", which are most intro descriptive. Then, discover other features through general example_user_app project and example projects of each particular feature.
- Read an example description in this document first, then dive into its example project. Additional description are presented in the code.
- Create custom PMBus MFR commands at the addresses 0xB1 to 0xBF in PMBus Spreadsheet shasta_pmbus.xlxx. Change column "Loop 0/1 support" from "y" to "n" to deactivate or from "n" to "y" to activate a respective MFR PMBus command. Explore other columns.
- Each example code has a pre-built image at /project/build/patch/patch.elf, which the user can upload to XDPP1100 directly - Plug-and-Play.

2 Basic examples

The following example codes show variation of "hello world" codes to illustrate the firmware patching that can be done on XDPP1100.

2.1 Hello World

This example shows how to configure and use the UART module to transmit "Hello World" during initialization.

UART module will transmit "Hello World" string when user_drv_init() function is called during initialization.

Example project name is: **example_hello_world**

The following additions and modifications in the project are summarized in the following table.

Table 1 Additions/Modifications for Hello World

Filename	Function Name
user_app.h/c	user_drv_init(void)~~Hello_world()~~
hello_world.h/c	hello_world()

Formatted Table

2.2 Hola Mundo

This example shows how to configure and use the UART module to transmit "Hola Mundo" periodically in regulation state machine.

Formatted: Justified

UART module is configured to transmit "Hola Mundo" and then attached to one of the state machine callback function "AT_TARGET_ENABLE" periodically.

Example project name is: **example_hola_mundo**

The following additions and modifications in the project are summarized in the following table.

Table 2 Additions/Modifications for Hola Mundo

Filename	Function Name
periodic_function.h/c	hHola_mundo()
r~~p~~egulation_state_machine_callbacks.h/c	Regulation_set_event_cb() Regulation_set_regulation_event_cb(REGULATION_STATE_AT_TARGET_VID, REGULATION_CONTROLS_ENABLE,AT_TARGET_ENABLE); AT_TARGET_ENABLE()

Formatted Table

2.3 Hallo Welt

This example shows how to configure and use the UART module to transmit "Hallo Welt" via custom PMBus command.

Formatted: Justified

UART module is configured to transmit "Hallo Welt" and then attached to PMBus MFR command 0xB1~~ED~~ MFR_HALLO_WELT. Whenever user set 0xFF to the PMBus data and perform Write, the "Hallo Welt" string will be transmitted out.

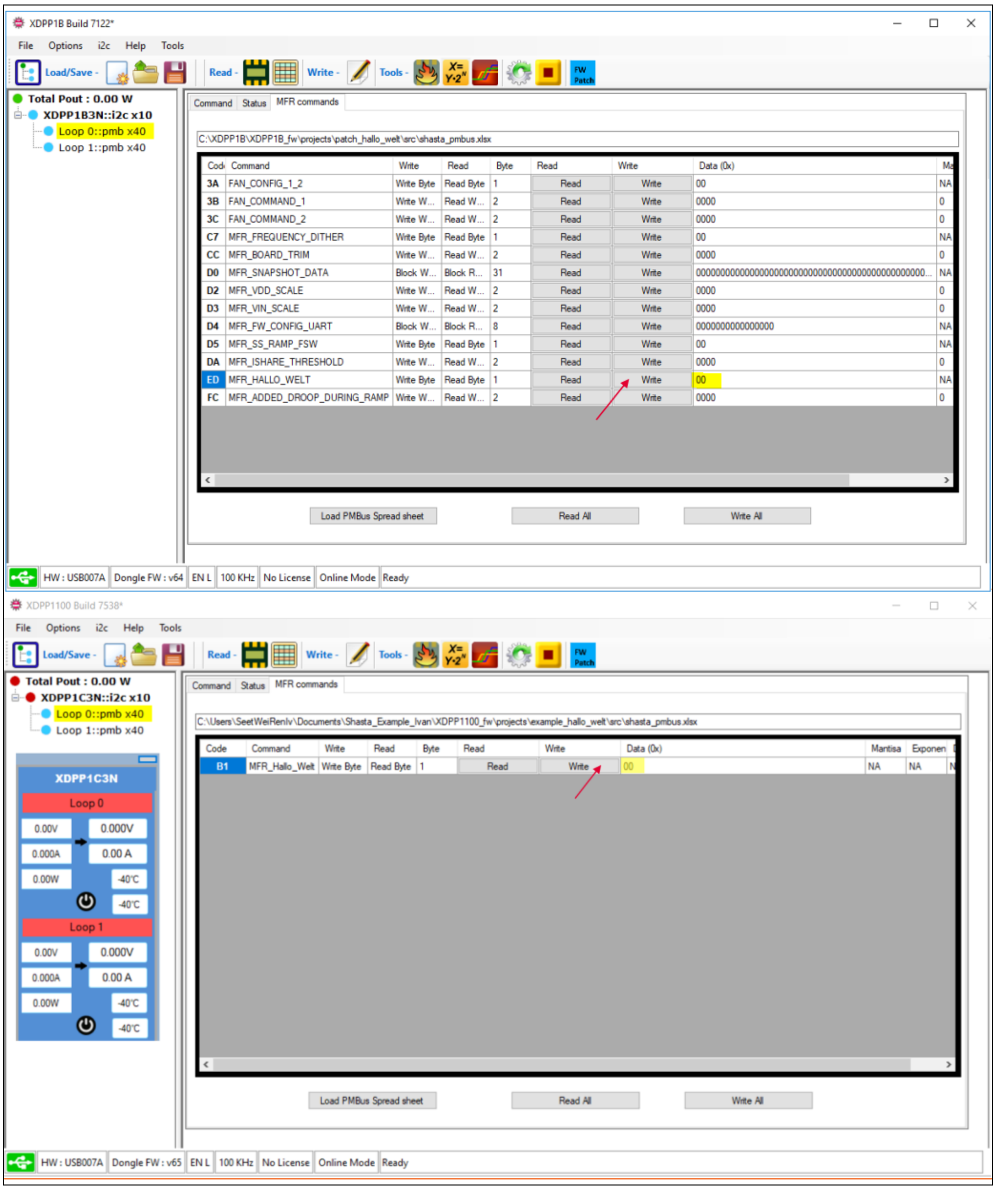


Figure 2 PMBus MFR Command MFR_HALLO_WELT

Example project name is: **example_hallo_welt**

The following additions and modifications in the project are summarized in the following table.

Table 3 Additions/Modifications For Hello Welt

Filename	Function Name
pmbus_mfr_specific_handler.h/c	~~Hallo_Welt()~~ PMBUS_HANDLE_MFR_HALLO_WELT()
hallo_welt.h/c	hallo_welt()

Formatted: English (United States)

Formatted: German (Germany)

3 Modulation

The following example codes show some of the existing Firmware patches designed for modulation related functionality.

Formatted: Justified

3.1 Adaptive Deadtime Adjustment

In bridge-type power topologies with digital controller, the power efficiency can be further optimized by manipulating deadtimes between the power switches across different operating loads. The following example codes show the operating loads can be divided into four operating regions. Each region will have its own set of deadtimes. Hysteresis zone is introduced in between each operating regions in order to perform smooth deadtimes transition.

Formatted: Justified

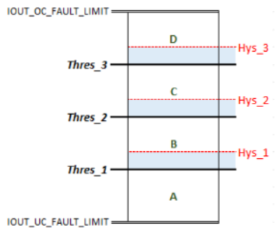


Figure 3 Operating Load Regions

Custom PMBUS MFR commands are defined for each set of deadtimes as well as Threshold level (THRES_1, THRES_2, THRES_3) as shown in the following figure.

Formatted: Justified

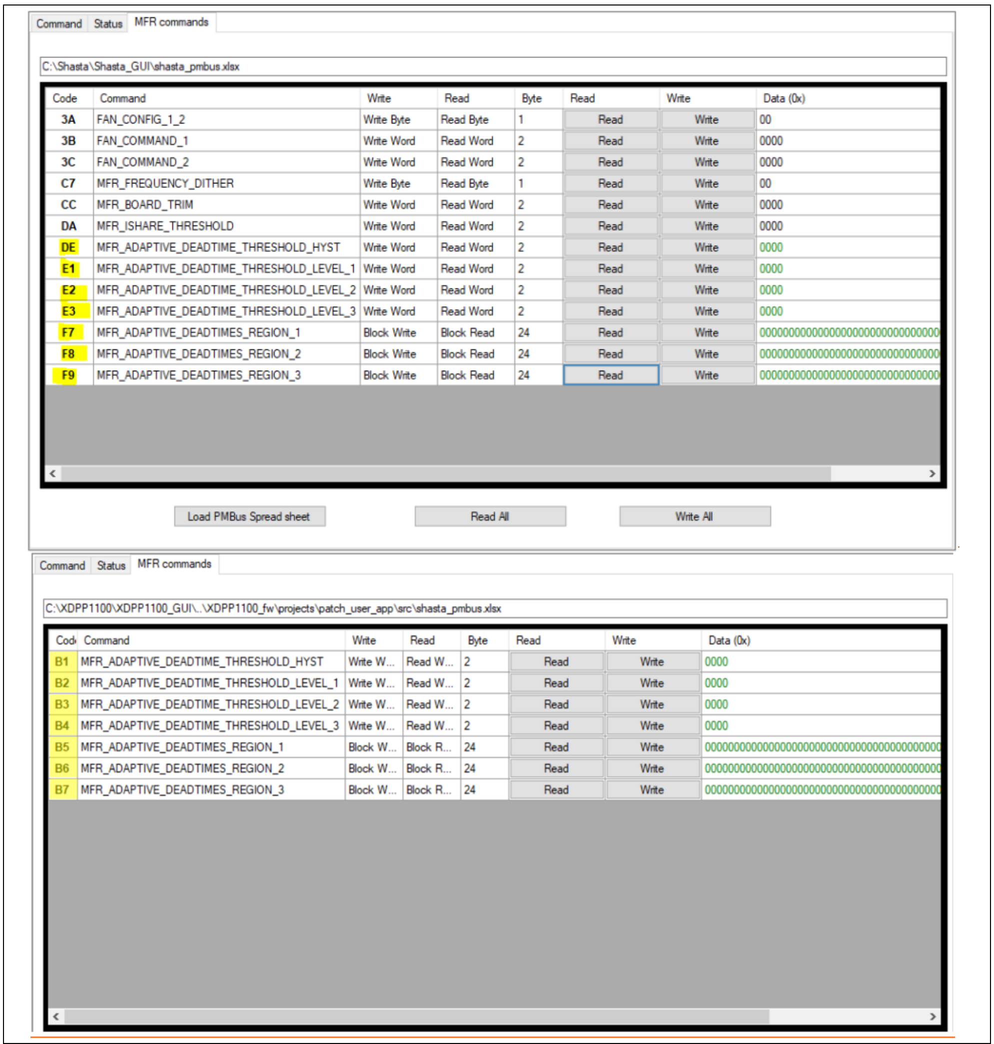


Figure 4 **Custom PMBus MFR Commands**

Each set of deadtime follows the same format as PMBus command 0xCF.

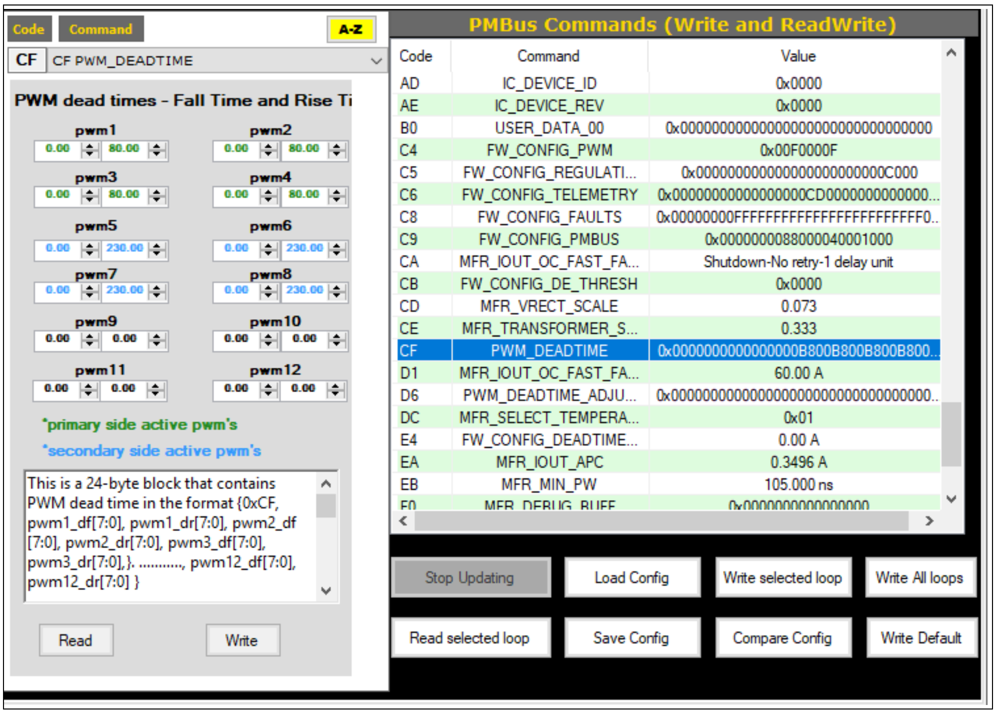


Figure 5 PMBus command 0xCF

Converting the deadtime decimal number (in ns) to integer representation can be done by dividing the respective decimal number with deadtime resolution defined as 1.25ns.

Formatted: Justified

The algorithm can be summarized in the following steps:

1. Get all the operating load thresholds, hysteresis and output current limit from PMBus
2. Check whether thresholds and hysteresis configuration make sense otherwise do nothing.
3. If thresholds and hysteresis configuration make sense, read output current
4. Find out at which band the output current currently and get deadtimes from respective PMBus command.
5. Update the deadtime registers.

Formatted: Justified, Indent: Left: 0.22"

Example project name is: **example_adaptive_deadtime_adjustment**

Formatted: Justified

The following additions and modifications in the project are summarized in the following table.

Table 4 Additions/Modifications for Adaptive Deadtime Adjustment

Filename	Function Name
adaptive_deadtime_adjustment.h/c	Adaptive_deadtime_adjustment() Adaptive_deadtime_init()
regulation_state_machine_callbacks.h/c	Regulation_set_regulation_event_cb(REGULATION_STATE_TON_DELAY, REGULATION_CONTROLS_ENABLE, adaptive_deadtime_init) Regulation_set_regulation_event_cb(REGULATION_STATE_AT_TARGET_VID, REGULATION_CONTROLS_TELEMETRY_UPDATED, adaptive_deadtime_adjustment)
~~pmbus_mfr_specific_handler.h/c~~	~~PMBUS_HANDLE_MFR_ADAPTIVE_DEADTIME_THRESHOLD_HYST()~~ ~~PMBUS_HANDLE_MFR_ADAPTIVE_DEADTIME_THRESHOLD_LEVEL_1()~~ ~~PMBUS_HANDLE_MFR_ADAPTIVE_DEADTIME_THRESHOLD_LEVEL_2()~~ ~~PMBUS_HANDLE_MFR_ADAPTIVE_DEADTIME_THRESHOLD_LEVEL_3()~~

Formatted Table

Commented [SWRI(DPSS1]: Function is remove as handler is not use and it set to “n” in the PMBus excel setting

3.2 Kickstart Pulse Pre-Startup

XDPP1100 is designed to control isolated bridge power topologies and always placed in the secondary-side. In certain situation, it is important to sense the input voltage at the primary-side before the power topologies are started. Due to the isolated nature of the topology, primary side sensing can only be achieved indirectly via secondary side voltage sensing. The following example codes show how some pulses can be injected (“kickstart”) into the power topologies from secondary side to enable indirect primary voltage sensing.

Formatted: Justified

This feature can be implemented by inserting some additional functions in OFF state and TON_DELAY state of the regulation state machine.

In XDPP1100, the duty cycle output is influenced by two hardware modules, namely PID Controller and FeedForward Controller (FF), as shown in the following diagram:

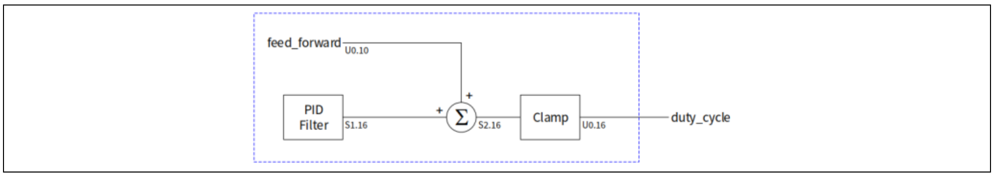


Figure 6 Duty Cycle Generation Block

In this implementation, PID module is chosen as the kickstart pulse generator by forcing it to generate a fixed duty cycle. At the same time, the FF module must be forced to 0 to remove its influence on the generated duty cycle. Of course in theory, it is possible to use FF module as the duty cycle generator and force the PID to 0.

Formatted: Justified

The following diagram shows the overall startup process.

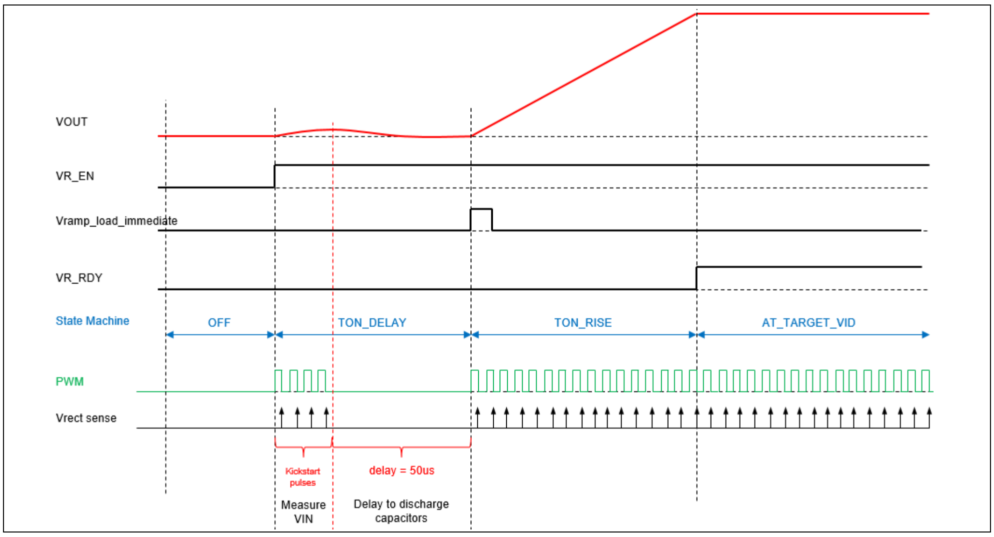


Figure 7 **Startup process with kickstart pulse**

The following diagram shows the startup process when regulation state is at OFF.

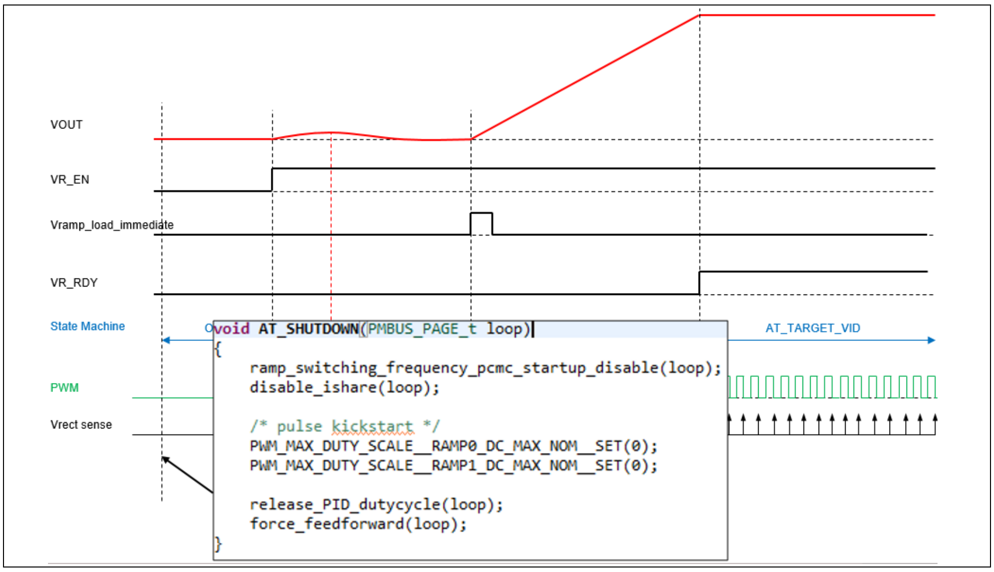


Figure 8 **Startup process when STATE = OFF**

The following diagram shows the startup process when regulation state is at TON_DELAY.

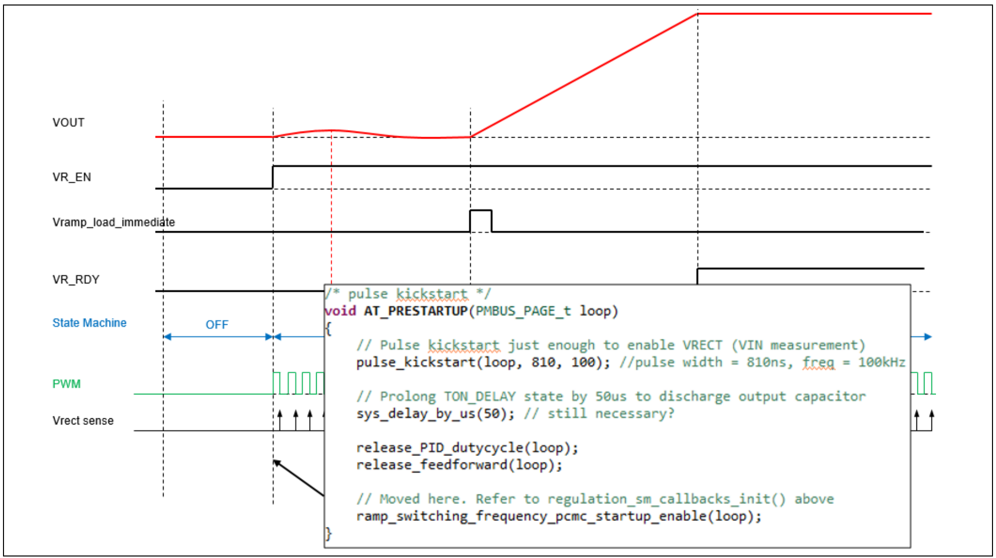


Figure 9 Startup process when STATE = TON_DELAY

Example project name is: **example_kickstart_pulse_pre_startup**

The following additions and modifications in the project are summarized in the following table.

Table 5 Additions/Modifications for Kickstart Pulse Pre-Startup

Filename	Function Name
vin_pulse_kickstart.c/h	force_PID_dutycycle() pulse_kickstart() release_PID_dutycycle() force_feedforward() release_feedforward()
regulation_state_machine_callbacks.c/h	Regulation_set_regulation_event_cb(REGULATION_STATE_TON_DELAY, REGULATION_CONTROLS_TON_TOFF_DELAY_TIMER, AT_PRESTARTUP); Regulation_set_regulation_event_cb(REGULATION_STATE_OFF, REGULATION_CONTROLS_SHUTDOWN, AT_SHUTDOWN); Regulation_set_regulation_event_cb(REGULATION_STATE_OFF, REGULATION_CONTROLS_SHUTDOWN_IMMEDIATE, AT_SHUTDOWN); AT_PRESTARTUP() AT_SHUTDOWN()

Formatted Table

3.3 Frequency Dithering

Adding some dithering in the switching frequency helps to randomize quantization error, which in turn gives a perceived smoother regulation performance. This example codes shows how to add frequency dithering in the PWM.

Formatted: Justified

Frequency dithering function in XDPP1100 is implemented in the firmware and it is configurable via PMBus MFR commands 0xC7.

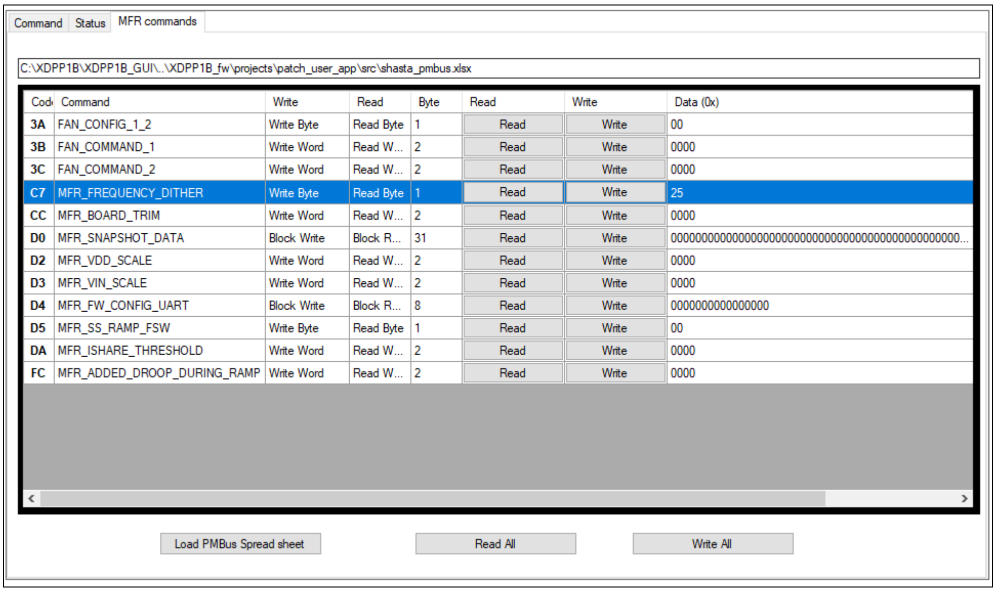


Figure 10 **PMBus MFR 0xC7 for Frequency Dithering**

The configuration consist of 8-bit data where:

Formatted: Justified

- [7:4] dithering update rate. The resolution is 64 switching period unit.
- [3:0] percentage of frequency to be dithered. The resolution is the percentage number, ±15% maximum.

In the example above, the PMBus MFR 0xC7 is set to **0x25**:

- The step rate is **2*64 = 128 switching period unit**. Assuming the switching frequency used is 100 kHz, the dithering update rate is 128 * (1/100 kHz) = **1280 us**.
- The percentage of frequency to be dithered is **±5%**

Example project name is: **example_user_app / Frequency Dithering**

The following additions and modifications in the project are summarized in the following table.

Table 6 Additions/Modifications for Frequency Dithering

Filename	Function Name
periodic_functions.c/h	Frequency_dither_enable() Frequency_dither_disable() Frequency_dither_irq_callback()
pmbus_mfr_specific_handlers.c/h	Pmbus_handle_mfr_frequency_dither()
regulation_state_machine_callbacks.c	Regulation_set_regulation_event_cb(REGULATION_STATE_TOFF_FALL, REGULATION_CONTROLS_SHUTDOWN, frequency_dither_disable); Regulation_set_regulation_event_cb(REGULATION_STATE_TOFF_FALL, REGULATION_CONTROLS_SHUTDOWN_IMMEDIATE, frequency_dither_disable); Regulation_set_fsw_irq_event_cb(fsw_irq_idx_3, frequency_dither_irq_callback); AT_TARGET_ENABLE()
regulation_state_machine_callbacks.h	USER_DATA{}

3.4 Deadtime Manipulation

Before reading this section, go ahead to read “Adaptive Deadtime Adjustment” in 2.1 for better understanding of how deadtime works in XDPP1100.

Formatted: Justified

Certain applications requires to use a larger deadtime for startup and a normal set of deadtime when the output voltage is at its target. This example shows how to manipulate by deadtime during different states.

For the purpose of this example, PWM2 and PWM4 deadtimes are manipulated depends on states of XDPP1100 operation. The startup deadtime is taken from 0xCF PWM_DEADTIME PMBus command for the startup state (refer to the picture below). For the steady state (at the output voltage target), the deadtime can be configurable via the respective new MFR PMBus command 0xB2 MFR_DEADTIME with LSB = 1.25 ns.

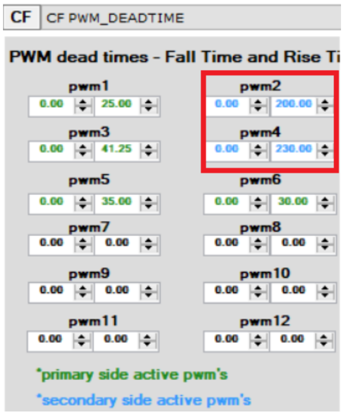


Figure 11 PMBus MFR 0xCF PWM_DEADTIME

Example project name is: **example_deadtime_manipulation**

The following additions and modifications in the project are summarized in the following table.

Table 7 Additions/Modifications for Deadtime Manipulation

Filename	**Function Name**
deadtime.c/h	void Set_StartUp_Deadtime() void Set_SteadyState_Deadtime() void PMBUS_HANDLE_MFR_DEADTIME () void PMBUS_HANDLE_MFR_DBG_DEADTIME ()
add_on_features.h	#define deadtime
regulation_state_machine_callbacks.c	regulation_sm_callbacks_init() TON_DELAY_ENABLE() TON_RISE_VID_REACHED() AT_SHUTDOWN()
regulation_state_machine_callbacks.h	USER_DATA{}

4 Regulation

The following examples code show some of the existing firmware patches designed for regulation related functionality.

4.1 Open-loop soft-startup

XDPP1100 is designed to operate in close-loop regulation and configurable via GUI. However, it is also possible to intentionally operate XDPP1100 in open-loop mode. The following example shows how to set the open-loop regulation with XDPP1100 and how to enable soft-startup in open-loop mode.

Example project name is: **example_open_loop _three_slope**

4.1.1 Open-loop regulation

The following diagram shows XDPP1100 modules that enable closed-loop regulation. In order to enable open-loop regulation, some modules have to be disabled, i.e. PID compensator, feed-forward compensator and flux-balancing. In addition, PWM can be forced to operate at specific values through *pwm_ramp* register. For example, by setting pwm.RAMP_FORCE_DUTY (format U0.8) to value 0x80, it will force the PWM to operate at 50% duty cycle.

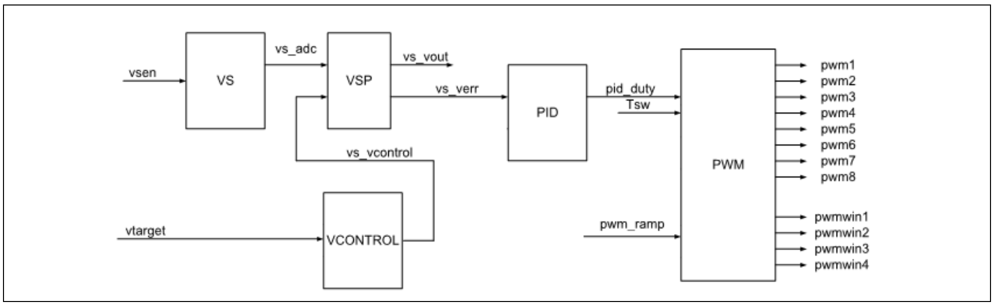


Figure 12 Voltage mode control diagram

4.1.2 Open-loop soft-startup

Open-loop soft-startup can be implemented by forcing PWM to operate from a small duty-cycle to a target duty-cycle. This ensures a smooth output voltage ramp-up during startup in the open-loop operation. PWM duty-cycle adjustment can be done using interrupt callback function which occurs periodically.

XDPP1100 has the possibility to execute user's code periodically for several regulation states (examples are REGULATION_STATE_TON_RISE, REGULATION_STATE_TON_DELAY, etc.) with a programmed numbers of callback cycles: from 2 to 64 cycles.

4.1.3 Implementation

In this example, a three slope open loop soft startup was designed to provide a flexibility to create various startup functions. Pleaes, refer to the Figure 13.

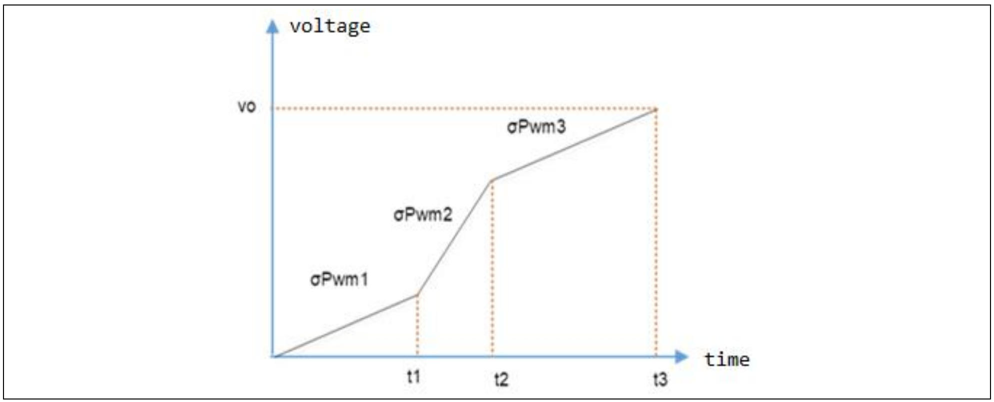


Figure 13 Three slope schema

The user is available to set his unique PWM and time actuation configuration via 0xB1, 0xB2 and 0xB3 PMBus MFR commads. Please, refer to example_open_loop_three_slop/src/shasta_pmbus.xlsx for additional information.

Formatted: Justified, Space Before: 8 pt, After: 8 pt

Moreover, the user might perform an initial calculation and check our configuration example in example_open_loop_three_slop/doxy/Open_Loop_Softstart_Calculation.xlsx.

The following additions and modifications in the project are summarized in the following table.

Table 8 Additions/Modifications for Open-Loop Soft-Startup

Filename	Function Name
open_loop_llc.c/h	#define SOFTSTART_MAX_STAGE 3 #define NUM_FORMAT_DIFF_RATIO_SCALE 5 #define NUM_FORMAT_DIFF_RATIO 41 update_switching_period() update_force_duty() void open_loop_llc_soft_start_enable() void open_loop_llc_soft_start_disable() void open_loop_llc_soft_start_irq_handler()
regulation_state_machine_callbacks.c/h	regulation_sm_callbacks_init() AT_TON_DELAY() AT_SHUTDOWN()~~regulation_sm_callbacks_init()~~

Commented [SWRI(DPSS2]: To make all example explaination the same for regulation state machine callback where it indicate the additional regulation cb

4.2 Frequency Ramp for Peak Current Mode Control Startup

In order to ensure smooth performance for peak current mode control during startup, the switching frequency can be ramped down from very high starting frequency to the desired frequency. A similar concept was discussed in 3.1 paragraph.

Formatted: Justified

Example project name is: **example_user_app**

The following additions and modifications in the project are summarized in the following table.

Table 9 Additions/Modifications for Peak Current Mode Control Startup

Filename	Function Name
periodic_functions.c/h	Ramp_switching_frequency_pcmc_startup_enable() Ramp_switching_frequency_pcmc_startup_disable() Ramp_switching_frequency_pcmc_startup_irq_handle()
regulation_state_machine_callbacks.c/h	regulation_sm_callbacks_init() TON_DELAY_ENABLE() AT_SHUTDOWN()

4.3 Fan Control

Fans are integrated in any switched mode power supply that output very high current. They are used for thermal regulation for blowing the hot air out of the power supply casing. XDPP1100 supports up to two fan controls and it can be controlled via PMBus 0x3A, 0x3B and 0x3C.

Formatted: Justified

Example project name is: **example_user_app**

The following additions and modifications in the project are summarized in the following table.

Table 10 Additions/Modifications for Fan Control

Filename	Function Name
pmbus_mfr_specific_handlers.c/h	PMBUS_HANDLE_FAN_CONFIG_1_2() PMBUS_HANDLE_FAN_COMMAND_1() PMBUS_HANDLE_FAN_COMMAND_2()

4.4 Modifying Transformer Scaling

XDPP1100 is designed to specifically support step-down, buck-kind of power topologies. As a result, the transformer scaling (primary-to-secondary turns ratio, expressed in Ns/Np) is limited from 0.0 to 1.0.

However, certain topologies that employs transformer for voltage and current scaling, such as -48V to 50Vdc conversion, may require transformer scaling such that Ns/Np = 5/3 = 1.67 which is out of the design limit.

This patch shows on how to overcome this design limit by defining a new PMBus MFR command 0xB1~~E5~~ MFR_ADJ_TURN_RATIO in which the new turns ratio can be filled in. Take note that the default 0xCE MFR_TRANSFORMER_SCALE must be set to 1.0.

Example project name is: **example_trafo_scaling**

The following additions and modifications in the project are summarized in the following table.

Table 11 Additions/Modifications for Transformer Scaling

Filename	Function Name
buckboost_telem.c/h	patch_Telemetry_change_scales()~~pmbus_handle_mfr_adj_turn_ratio()~~

Filename	Function Name
pmbus_mfr_specific_handlers.c /h	PMBUS_HANDLE_MFR_ADJ_TURN_RATIO()~~patch_Telemetry_change_scales()~~
user_app.c	user_drv_init()

4.5 Active Current Sharing

This example can be useful in terms of XDPP1100 Current Sharing feature modification and debugging. Refer to XDPP1100 datasheet for "*Current Share*" feature clarification.

Example project name is: **example_user_app**

The following additions and modifications in the project are summarized in the following table.

Table 12 Additions/Modifications for Active Current Sharing

Filename	Function Name
add_on_features.c/h	#ifdef en_ishare void added_droop_disable() void added_droop_enable() void remove_added_droop_irq_callback() void enable_ishare() void disable_ishare() void patch_Regulation_Shutdown_Sequence() #endif
regulation_state_machine_callbacks.c/h	regulation_sm_callbacks_init() TON_RISE_ENABLE() AT_SHUTDOWN() TON_RISE_VID_REACHED()

5 Telemetry

The following example codes show some of the existing Firmware patches designed for telemetry related functionality.

5.1 Using PTC Temperature Sensor instead of NTC Temperature Sensor

Sensing temperature in power supplies is usually done with Thermistor, i.e. a device whose resistance change as change in its temperature. There are two types of thermistor namely NTC (Negative Temperature Coefficient) and PTC (Positive Temperature Coefficient).

NTC resistance decreases as the temperature goes higher. However, the relationship between the resistance and temperature is not exactly linear. The following figure illustrates the Temperature vs Resistance relationship on NTC thermistor:

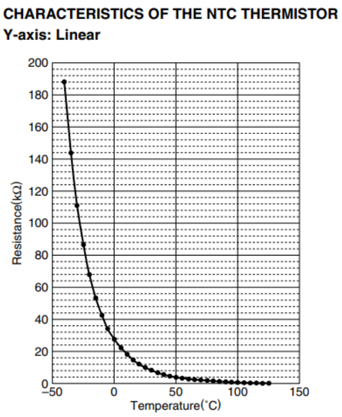


Figure 14 NTC Temperature vs Resistance

When using digital controller, it is common to denote this non-linear relationship using LookUp Table (LUT).

XDPP1100 supports NTC (Negative Temperature Coefficient) temperature sensor by default. LookUp Table is given in user_ntc_temp_lut.h.

In some cases, PTC (Positive Temperature Coefficient) is used over NTC sensor. PTC has the opposite behavior from NTC. The resistance increases as temperature increase. However, the relationship is not exactly linear either.

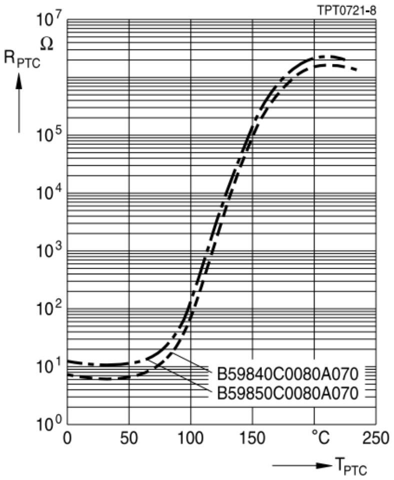


Figure 15 **PTC Temperature vs Resistance**

For this reason, a lookup table for PTC has to be redefined.

Example project name is: **example_ptc_temperature_sensor**

The following additions and modifications in the project are summarized in the following table.

Table 13 **Additions/Modifications for PTC Temperature Sensor**

Filename	Function Name
ptc_temp_sens_telemetry_get~~patch_Telemetry_get_adc_code~~.c/h	patch_telemetry_get_adc_code() patch_Telemetry_get()
user_app.c/h	user_drv_init() ~~patch_Telemetry_Get() // line 257~~

5.2 Modifying Output Current Scales

XDPP1100 support output current scaling up to 127A. This example codes show how to modify the output current scalings by double, up to 255A. Details on registers and PMBus commands patched can be found in iout_range_patch.xlsx attached in the /src folder.

Example project name is: **example_iout_scale**

The following additions and modifications in the project are summarized in the following table.

Table 14 **Additions/Modifications to modify Output Current Scales**

Filename	Function Name
iout_range_pmbus_mfr_handler.c/h	patch_PMBUS_HANDLE_IOUT_CAL_OFFSET() patch_PMBUS_HANDLE_IOUT_OC_FAULT_LIMIT()

Filename	Function Name
	patch_PMBUS_HANDLE_IOUT_UC_FAULT_LIMIT() patch_PMBUS_HANDLE_IOUT_OC_WARN_LIMIT() patch_PMBUS_HANDLE_IIN_OC_FAULT_LIMIT() patch_PMBUS_HANDLE_IIN_OC_WARN_LIMIT() patch_PMBUS_HANDLE_VOUT_DROOP() patch_PMBUS_HANDLE_FW_CONFIG_REGULATION() patch_PMBUS_HANDLE_MFR_IOUT_OC_FAST_FAULT_LIMIT() patch_PMBUS_HANDLE_MFR_IOUT_APC()
iout_range_telemetry_sample.c/h	patch_Telemetry_Sample()
user_app.c/h	user_drv_init() patch_pmbus_mfr_autogen_iout_range()

5.3 Efficiency Look-Up Table and Input Current Correction Look-Up Table

XDPP1100 is usually mounted on the secondary side of an isolated power topologies. In this case, certain parameters such as Input Voltage and Input Current at primary side can only be estimated due to the existence of galvanic isolation which disable direct measurements. For Input Voltage (Vin) measurement, it can be done indirectly by sensing the secondary side transformer voltage via Vrect sensing.

For Input Current (Iin), it has to be estimated with the equation below. To implement a modified Input Current Estimation, refer to section "5.4 Input Current Estimation" in this document.

$$Iin_{est} = \frac{Iout * Vout}{Vin} = \frac{Pout}{Vin}$$

This equation assumes unity efficiency, i.e. input power is the same to output power. In practice, efficiency varies with operating conditions.

For this reason, it is necessary to compensate the above Input Current Estimation to reflect more accurate readings. This can be done by measuring the actual efficiency at varying operating conditions and listing them in a table (thus "Efficiency Table") and use this table to compensate for the input current reading.

This patch implement both Efficiency LUT and Input Current Correction for more accurate telemetry reporting.

5.3.1 Efficiency LUT

The following table shows an example of the Efficiency Table measured directly from the system. This table assume close-to-ideal system with very tight component tolerance.

Table 15 Efficiency LUT Example (decimal)

Vin\Pout	4W	8W	16W	32W	64W	80W
36Vdc	0.941	0.944	0.951	0.96	0.961	0.96
48Vdc	0.941	0.945	0.952	0.963	0.964	0.958
60Vdc	0.94	0.944	0.955	0.964	0.966	0.952

The values in the table above can be represented as an unsigned U0.8 where:

Table 16 U0.8 Representation

Format	DEC	HEX
Bit Length	8 bit	
LSB Weight	2^(-8) = 0.00390625	
Min Value	0	0x00
Max Value	255 * LSB Weight = 0.99609375	0xFF

Using U0.8 representation, the efficiency LUT can be represented as following:

Table 17 Efficiency LUT in U0.8 Representation

Vin\Pout	4W	8W	16W	32W	64W	80W
36Vdc	241	242	243	246	246	246
48Vdc	241	242	244	247	247	246
60Vdc	240	242	245	247	247	244

Reporting efficiency figure to the user can then be implemented via custom PMBus MFR command. In this example patch, ~~0xE5~~ 0xB1 slot is chosen.

5.3.2 Input Current Correction LUT

Given Output Power (Pout) and Input Voltage (Vin) conditions, an efficiency value (eff) can be interpolated from the table and therefore, the adjusted Input Current estimation can be compensated with the as following:

$$Iin_{est_{adj}} = Iin_{est} * \frac{1}{eff}$$

Division operation in embedded processor takes a lot of CPU load and therefore it is undesirable to implement the above equation as it is. A straightforward workaround can be devised by manually calculating the (1/eff) component, as shown in the following table.

Table 18 Updated Correction factor LUT with (1/eff)

Vin\Pout	4W	8W	16W	32W	64W	80W
36Vdc	1.06382978723404	1.05820105820106	1.05263157894737	1.04166666666667	1.04058272632674	1.04166666666667
48Vdc	1.06382978723404	1.05820105820106	1.05042016806723	1.03842159916926	1.03734439834025	1.04275286757039
60Vdc	1.06496272630458	1.05932203389831	1.04602510460251	1.03734439834025	1.0351966873706	1.04931794333683

As can be seen from the above table, taking the manually calculated (1/eff) component as and plug it in to the equation will not be a good solution because these values requires a very lengthy data representation to maintain good accuracy. This is also undesirable in embedded processor as the efficiency table will consume more memory.

It is also important to notice that the values are always conforming to "1.0xxxxxx". For this reason, it is possible to modify and represent the (1/eff) as a Correction Factor:

$$corr = \left(\frac{1}{eff}\right) - 1$$

And the original equation can be modified as following:

$$Iin_{est_{adj}} = Iin_{est} * (corr + 1)$$

By doing so, the efficiency table will be updated as following:

Table 19 Correction Factor LUT

Vin\Pout	4W	8W	16W	32W	64W	80W
36Vdc	0.06382978723404	0.05820105820106	0.05263157894737	0.04166666666667	0.04058272632674	0.04166666666667
48Vdc	0.06382978723404	0.05820105820106	0.05042016806723	0.03842159916926	0.03734439834025	0.04275286757039
60Vdc	0.06496272630458	0.05932203389831	0.04602510460251	0.03734439834025	0.0351966873706	0.04931794333683

Assuming the lowest efficiency possible is at 33% and accuracy representation has to be less than 3%, the values in the Correction Factor LUT has to be represented with U1.9 representation, where:

Table 20 U1.9 Representation

Format	DEC	HEX
Bit Length	10 bit	
LSB Weight	2^(-9) = 0.001953125	
Min Value	0	0x00
Max Value	1.998046875	0x3FF

The updated Correction Factor LUT is shown below:

Table 21 Correction Factor LUT in U1.9 Representation

Vin\Pout	4W	8W	16W	32W	64W	80W
36Vdc	33	30	27	21	21	21
48Vdc	33	30	26	20	19	22
60Vdc	33	30	24	19	18	25

A telemetry function called 'Telemetry_~~sample~~Sample()' can then be patched to calculate the corrected input current value.

5.3.3 Implementation

The following figure shows Efficiency LUT and Input Current Correction LUT.

```
const uint8_t efficiency_table [VIN_MAX_IDX][POUT_MAX_IDX] = {
                {0, 241, 242, 243, 246, 246, 246},
                {0, 241, 242, 244, 247, 247, 246},
                {0, 240, 242, 245, 247, 247, 244},
```

```
              {0, 239, 241, 244, 246, 247, 243}, // as an example an additional line can be presented for Vin 75V
};
```

Figure 16 Efficiency LUT

```
const uint16_t correction_table [VIN_MAX_IDX][POUT_MAX_IDX] = {
              {0, 33, 30, 27, 21, 21},
              {0, 33, 30, 26, 20, 19, 22},
              {0, 33, 30, 24, 19, 18, 25},
              {0, 36, 32, 26, 21, 18, 28},
}
```

Figure 17 Input Current Correction LUT

A new function called '~~InputCurrent~~input_current_~~correction(~~correction ()' is defined to implement and combine both Efficiency LUT and Input Current Correction LUT. To implement a modified Input Current Estimation, refer to section "4.4 Input Current Estimation" in this document.

It is also interesting to take note on the arithmetic operation that is happening in the following line:

```
uint16_t iin_corr = input_current + SHIFT_EXPONENT_UNSIGNED( input_current * correction, correction_exp );
uint16_t iin_corr = InputCurrent + SHIFT_EXPONENT_UNSIGNED( InputCurrent * correction, correction_exp );
```

Formatted: Font color: Auto

Figure 18 Input Current Correction fixed-point implementation

With original input current format of U6.4, the firmware operation can be described as the following steps:

Table 22 Input Current Correction fixed-point arithmetic

Equation	$InputCurrent_{corr} = InputCurrent + (InputCurrent * corr) \gg correction\_exp$
Representation	$U6.4 = U6.4 + (U6.4 * U1.9) \gg 9$
	$U6.4 = U6.4 + (U6.13 \gg 9)$
	$U6.4 = U6.4 + (U6.4)$

Example project name is: **example_efficiency_table_current_correction**

The following additions and modifications in the project are summarized in the following table.

Table 23 Additions/Modifications for Efficiency LUT and Input Current Correction

Filename	Function Name
input_c~~C~~urrent_~~Correction~~correction.c/h	c~~C~~alculate_i() c~~C~~alculate_j() i~~I~~nput_c~~C~~urrent_c~~C~~orrection()
input_current_telemetry_sample.c	patch_Telemetry_Sample()
efficiency_table.h	c~~C~~orrection_table[][] e~~E~~fficiency_table[][]
user_app.c/h	user_drv_init()

Filename	Function Name
	~~Patch_Telemetry_Sample()~~

5.4 Input Current Estimation

In order to calculate the input current from the relation between the output power and the input voltage directly, the user can apply this example.

Example project name is: **example_input_current_estimation**

The following additions and modifications in the project are summarized in the following table.

Table 24 Additions/Modifications for Input Current Estimation

Filename	Function Name
add_on_features.c/h	input_current_estimation~~inputCurrentCorrection~~() patch_Telemetry_Sample()
user_app.c/h	user_drv_init()

5.5 Telemetry Interrupt

Telemetry interrupt (or Telemetry IRQ) is essential key in the XDPP1100 to trigger certain events to be executed when observed telemetry data exceed/fell down below certain threshold.

This example shows how to use the telemetry IRQ for a frequency switch adjust based on the input voltage (VIN) source.

Example project name is: **example_telemetry_irq**

The following additions and modifications in the project are summarized in the following table.

Table 25 Additions/Modifications to create a custom VIN Telemetry IRQ

Filename	Function Name
add_on_features.c/h	void add_on_features_init() void Telemetry_IRQ() void Telemetry_IRQ_VIN_HANDLE() custom_frequency_update()
regulation_state_machine_callbacks.h	USER_DATA[]

Another example of Telemetry IRQ usage ~~using~~ can be found in the section 6.1 “Adding Extra Level of Firmware protection”.

5.6 Telemetry Sense ADC – Custom VDAC with XADDR

XDPP1100 is available to be used for an external resistance decoding to pre-program XDPP1100 settings. For instance, a frequency switch can be tuned by using an external resistor.

This example shows how to measure an external resistor at pin XADDR1 with TSADC (VDAC) and how to set an output target voltage according to a measured resistance value with a custom Look-Up Table (LUT).

5.6.1 Implementation

In this example, an external resistor at pin XADDR1 is measured and respective output voltage target is set once XDPP1100 regulation starts. In addition, the user can change a current source value idac at XADDR pin and implement own Look-Up table to pick up needed output voltage target according to desired measured resistance.

Take a note, that measured resistance is measured based on the following formula:

$$res_{meas} = vdac/idac$$

where res_meas – measured (calculated) resistance, vdac – measured voltage at pin XADDR and idac – initiated current source value.

0xB1 MFR PMBus command MFR_VOUT_RES_MEAS was implemented to observe a measured resistance at pin XADDR1 with LSB 0.1172 kOhm.

Example project name is: **example_xaddr_usage**

The following additions and modifications in the project are summarized in the following table.

Table 26 Additions/Modifications to create a custom VDAC measurements with XADDR

Filename	Function Name
regulation_state_machine_callbacks.h	USER_DATA[]
custom_vout_command_lut.h	Look-Up Table
custom_vdac.c/h	xaddr_resistance_measurement() vdac_xaddr_measurement() vout_command_set()

6 Faults & Protection

The following example codes show some of the existing Firmware patches designed for faults & protection related functionality.

6.1 Adding Extra Level of Firmware protection

Some applications require more stringent, multi-layer protections to ensure system reliability. This example codes show how to add extra level for firmware protection for over voltage protection.

Example project name is: **example_soft_voltage_protection**

The following additions and modifications in the project are summarized in the following table.

Table 27 Additions/Modifications to set Firmware Protection

Filename	Function Name
ovp_fault.c/h	Telemetry_IRQ_VOUT_MFR_OVP_FAULT_HANDLE() PMBUS_HANDLE_MFR_OVP_FAULT_LIMIT()
vin_uvp.c/h	Telemetry_IRQ_VIN_MFR_UVP_FAULT_HANDLE() PMBUS_HANDLE_MFR_VIN_UV_FAULT_LIMIT()

6.2 Status Bit Clean

In order to clean up an unwanted fault register, the user can use this example.

For instance, the user wants to clear the PMBus STATUS bit “NONE OF THE ABOVE” if it is due to a Power Good fault. The “NONE OF THE ABOVE” should not be cleared if it is due to other trigger. The picture below shows example’s performance.

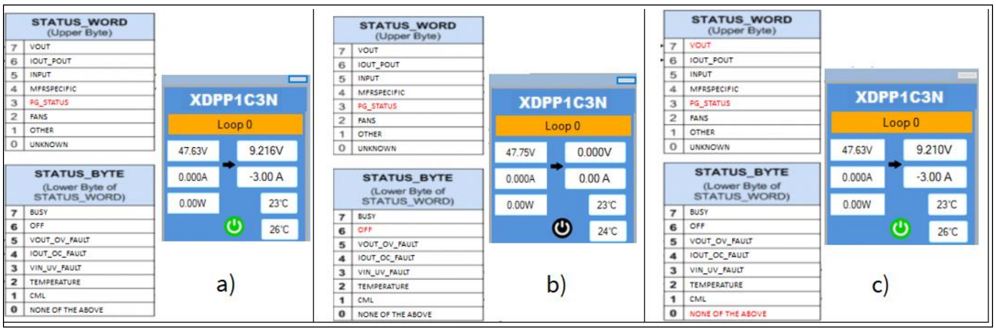


Figure 19 a) Active state (ON mode) operation with only POWER_GOOD fault; b) standby state (OFF mode); c) active state with an additional fault (VOUT – the output voltage)

Picture 20 a) shows that NONE_OF_THE_ABOVE is cleared, when only PG_STATUS (or POWER_GOOD) is triggered. In b), NONE_OF_THE_ABOVE is cleared as well, because there are only PG_STATUS and OFF faults are triggered. In c) case the additional fault VOUT is presented during the active state, so NONE_OF_THE_ABOVE is active.

A function status_word_update() checks if PG_GOOD is high and if PG_GOOD is only one triggered fault in the upper byte of status word bits [12:8]. If so, NONE_OF_THE_ABOVE is cleared and it is written back to the status word and status byte data.

The tricky part is that the XDPP1100 hardware does the latching, so the status bit update is always needed to be executed. A function status_word_update() is executed every 1 ms time in a internal RTOS user thread USER_Thread().

Example project name is: **example_pmbus_flag_fault**

The following additions and modifications in the project are summarized in the following table.

Table 28 Additions/Modifications to clean up NONE_OF_THE_ABOVE fault status

Filename	Function Name
user_app.c/h	USER_Thread() user_drv_init() status_word_update()

6.3 Fault Masking

There might be cases, when a certain fault should be masked or hidden. To do so, the user might disable/enable the output voltage under voltage fault with the following code:

Code Listing 1 Faults_Mask_Fault() use

```
/**
* Faults_Mask_Fault function does a masking of the fault types
* @param loop: pmbus page to update status on
* @param hw_fault 1: hw; 0:fw; 2: common mask applied to hardware (0-31) or firmware (32-63) fault_types; an enum
in faults_api.h to define which type of fault to mask
* @param set_mask: set 1 to desired faults to be enabled
* @param clear_mask: set 1 to clear enable mask bits
*/

        // To disable vout_uv faults //
        Faults_Mask_Fault(loop, 1, //hw fault
                          0, //set_mask
                          ((1u<<(uint32_t)FAULT_TYPE_VOUT_UV_FAULT) |
                                  (1u<<(uint32_t)FAULT_TYPE_VOUT_UV_WARN)));


        // To enable vout_uv faults //
        Faults_Mask_Fault(loop, 1, //hw fault
                       ((1u<<(uint32_t)FAULT_TYPE_VOUT_UV_FAULT) |
                              (1u<<(uint32_t)FAULT_TYPE_VOUT_UV_WARN)), //set_mask
                                     0); //clear_mask
```

7 Communication

The following example codes show some of the existing Firmware patches designed for communication related functionality.

7.1 Making PMBus stays on when no regulation

XDPP1100 assumes that there is no need for PMBus reporting when the power converter is off. In certain situation, this behavior is undesirable and there is still a need for PMBus reporting the telemetry. This firmware example codes show how to enable PMBus when the power converter is off.

Example project name is: **example_pmbus_stays_on**

The following additions and modifications in the project are summarized in the following table.

Table 29 Additions/Modifications for PMBus Stays ON

Filename	Function Name
~~patch_~~pmbus_stays_on.c/h	patch_Regulation_Shutdown_Sequence() patch_Regulation_Power_Mode()
user_app.c/h	user_drv_init()

8 Memory/Storage

The following example codes show some of the existing Firmware patches designed for memory/storage related functionality.

8.1 Storing different FW patches at different OTP partitions

In some applications, different firmwares need to be stored at different memory partition. 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.

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.
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 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.*

8.1.1 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.

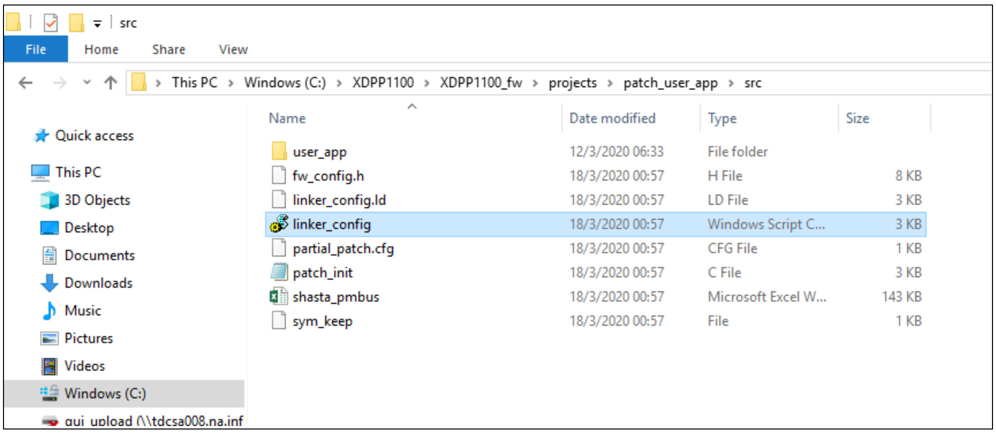


Figure 20 File location of linker_config.sct

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 21 **Linker_Config.sct**

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 22 **Modification of linker_config.sct for Partition 2**

8.1.2 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.

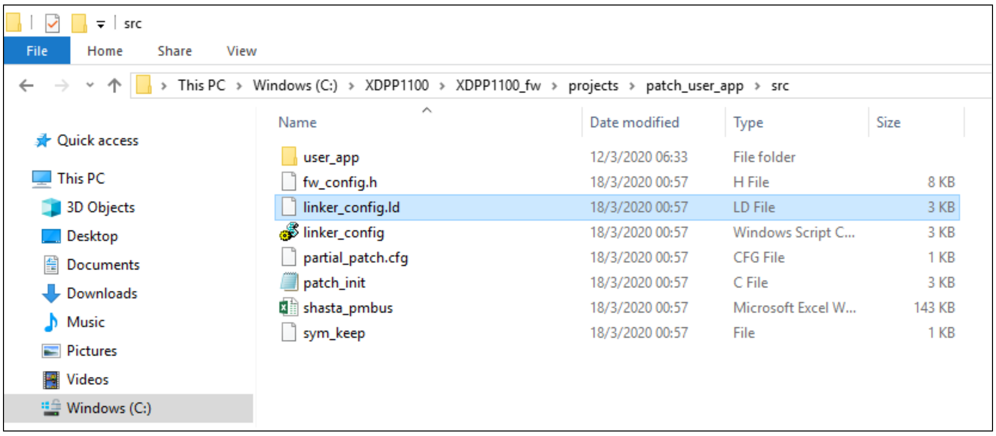


Figure 23 File location of linker_config.ld

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 24 **Linker_config.ld**

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 25 **Modification of linker_config.ld for Partition 2**

8.1.3 STEP 2: Update Patch Entry

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

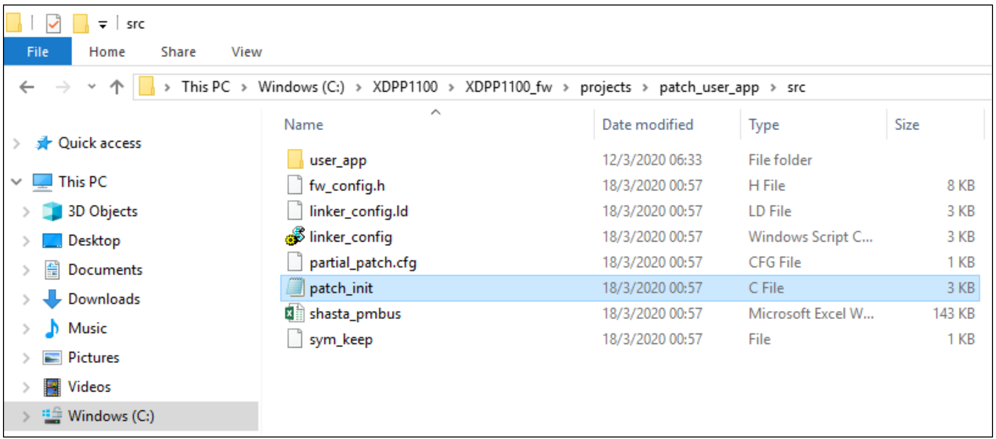
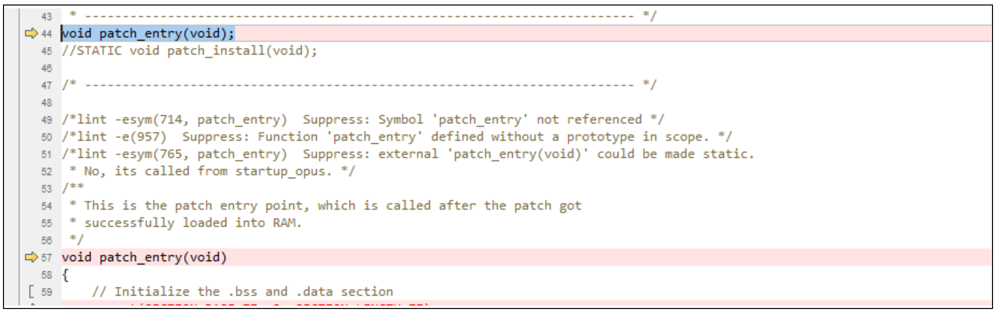


Figure 26 **File location of patch_init.c**

Scroll down until you can find the following lines:



```
 * ----------------------------------------------------------------------- */
void patch_entry(void);
//STATIC void patch_install(void);

/* ----------------------------------------------------------------------- */

/*lint -esym(714, patch_entry)  Suppress: Symbol 'patch_entry' not referenced */
/*lint -e(957)  Suppress: Function 'patch_entry' defined without a prototype in scope. */
/*lint -esym(765, patch_entry)  Suppress: external 'patch_entry(void)' could be made static.
 * No, its called from startup_opus. */
/**
 * This is the patch entry point, which is called after the patch got
 * successfully loaded into RAM.
 */
void patch_entry(void)
{
    // Initialize the .bss and .data section
```

Figure 27 **Patch_init.c**

Update patch_entry:

```
 * ------------------------------------------------------------------------- */
void patch2_entry(void);
//STATIC void patch_install(void);

/* ------------------------------------------------------------------------- */

/*lint -esym(714, patch_entry)  Suppress: Symbol 'patch_entry' not referenced */
/*lint -e(957)  Suppress: Function 'patch_entry' defined without a prototype in scope. */
/*lint -esym(765, patch_entry)  Suppress: external 'patch_entry(void)' could be made static.
 * No, its called from startup_opus. */
/**
 * This is the patch entry point, which is called after the patch got
 * successfully loaded into RAM.
 */
void patch2_entry(void)
{
```

Figure 28 Modification of patch_init.c

8.1.4 STEP 3: Modify Makefile

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

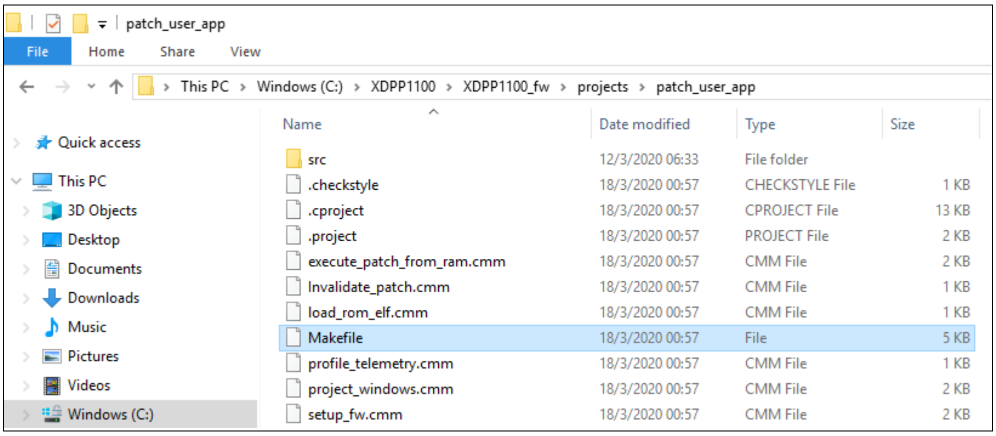


Figure 29 File location of Makefile

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 30 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 31 Modification of Makefile

8.1.5 STEP 4: Build the project

Build the project to generate the patch file.

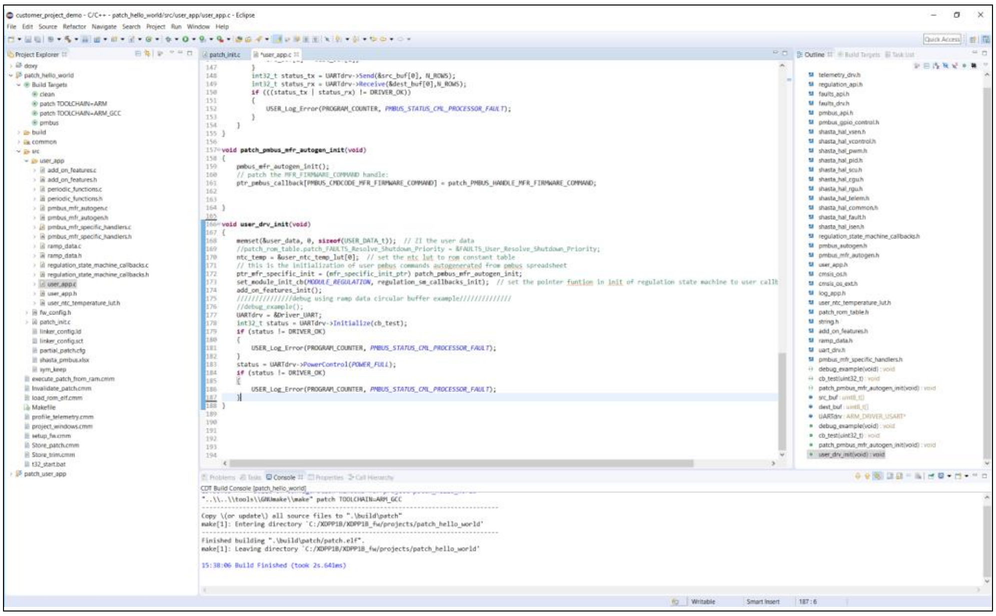


Figure 32 Build project

8.1.6 STEP 5: Store the patch project

Open the GUI and update the OTP partitions as shown below.

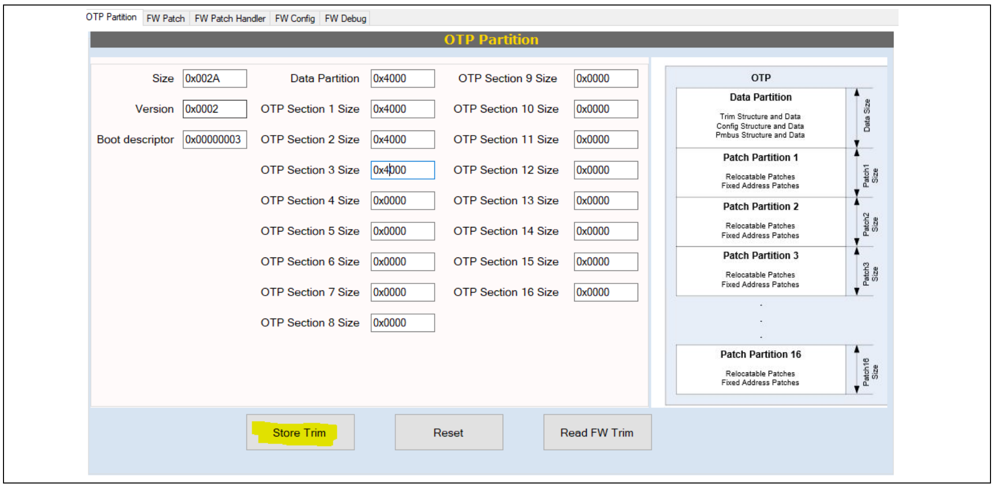


Figure 33 **OTP Partition configuration**

Set the OTP partition to the correct one before storing the patch.

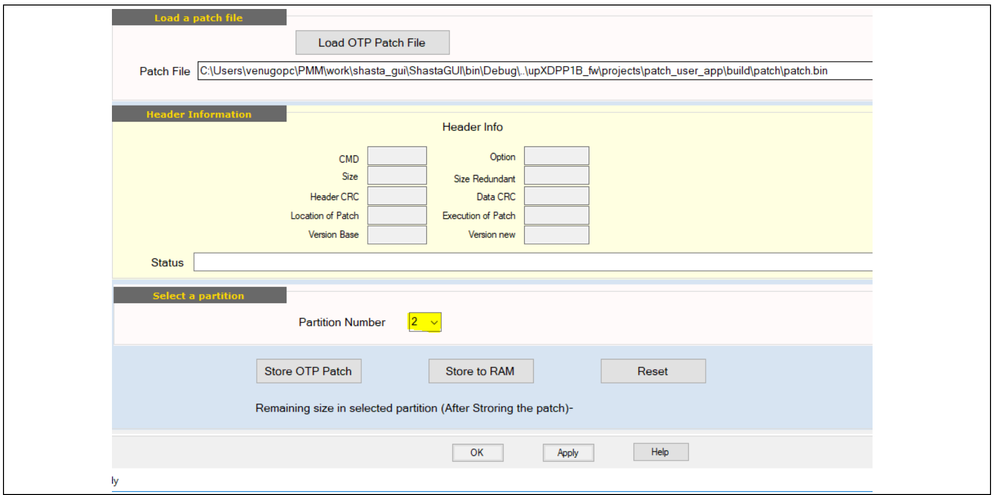


Figure 34 **Setting Partition Number to the correct one**

Revision history

Document version	Date of release	Description of changes
1.0	2020-07-23	Initial 12 examples code; PMBus configuration.
~~2~~1.1~~0~~	2020-~~11~~10-01	Added ~~7~~4 examples. 4 in dev.

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

Edition 2020-07-23

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference