XDP™ XDP711-001 config file translation and programming guide

About this document

Scope and purpose

XDP™ XDP711-001 is a wide-input voltage hot-swap and system-monitoring controller IC that drives single or multiple parallel N-channel MOSFETs. XDP™ XDP711-001 has two types of internal nonvolatile memory (NVM) to store specific command settings; at power-up, during the READ_CFG state. All the settings saved in one-time programmable (OTP) and multiple time programmable (MTP) memory are copied into volatile memory.

This document describes how to read the production configuration (.txt) file and store the commands to the volatile memory over the I^2C serial interface, and then store the settings in nonvolatile memory (NVM).

Intended audience

This document is intended for design and application engineers who use XDP™ XDP711-001.

Table of contents

About this document **1**
Table of contents **2**
1 Introduction **3**
2 Hardware setup **5**
3 Register difference between XDP™ XDP710-002 and XDP™ XDP711-001 ..... **6**
4 PMBus® commands **7**
4.1 PMBus® protocol diagram symbols and elements 7
4.2 PMBus® commands communication protocol 8
5 Production file format details and programming procedure **10**
5.1 Identifying the connected device for programming 10
5.1.1 Reading device identification 10
5.2 Production file format 11
5.3 PMBus® register configuration and programming to RAM 12
5.3.1 Interpreting PMBus® register values from production (.txt) file 12
5.3.2 Write PMBus® commands into RAM 15
5.3.3 Reading PMBus® register CRC and verifying with config file CRC 15
5.4 SOA register configuration and programming to RAM 17
5.4.1 SOA register interpretation from production (.txt) file 17
5.4.2 SOA data programming onto RAM 20
5.5 Storing commands to nonvolatile memory (NVM) 22
5.5.1 Storing PMBus® commands settings to OTP 23
5.5.2 Storing I_SNS_CFG settings to MTP 24
5.5.3 Storing SOA settings to SOA memory 25
5.6 Handling PMBus® address change caused by configuration file 26
5.7 Memory check after rebooting 28
References **31**
Revision history **32**
Disclaimer **33**

1 Introduction

XDP™ XDP711-001 is a wide-input voltage hot-swap and system-monitoring controller IC that is used in 48 V AI factories, robotics actuators, data center, etc., for charging capacitive loads or hot-plugging events. Since this is a digital hot-swap controller, all the configuration is done digitally via PMBus® and is stored on the internal memory of the device.

XDP™ XDP711-001 has the following types of memory for programmability:

- Volatile memory
- One-time programmable (OTP)
- Multiple-time programmable (MTP)
- Safe operating area (SOA) memory

The OTP memory can be used to fix and save specific command settings; at power-up, during the READ_CFG state and all the settings saved in the OTP memory are copied into volatile memory.

XDP™ XDP711-001 contains 10-pages of MTP for multiple-time programmability. When the number of reprogramming reaches 10 (indicated by the MTP_FULL bit in the `STATUS_MEM` command), the circuit keeps the latest programmed values. This memory section contains the `I_SNS_CFG` command that contains the CS_RNG, CS_RNG_TRIM, SOC_FAULT_LIMIT, and START_ILIM configuration bits.

In this document, the configuration file example is for XDP™ XDP711-001, and the programming procedure is explained based on it.

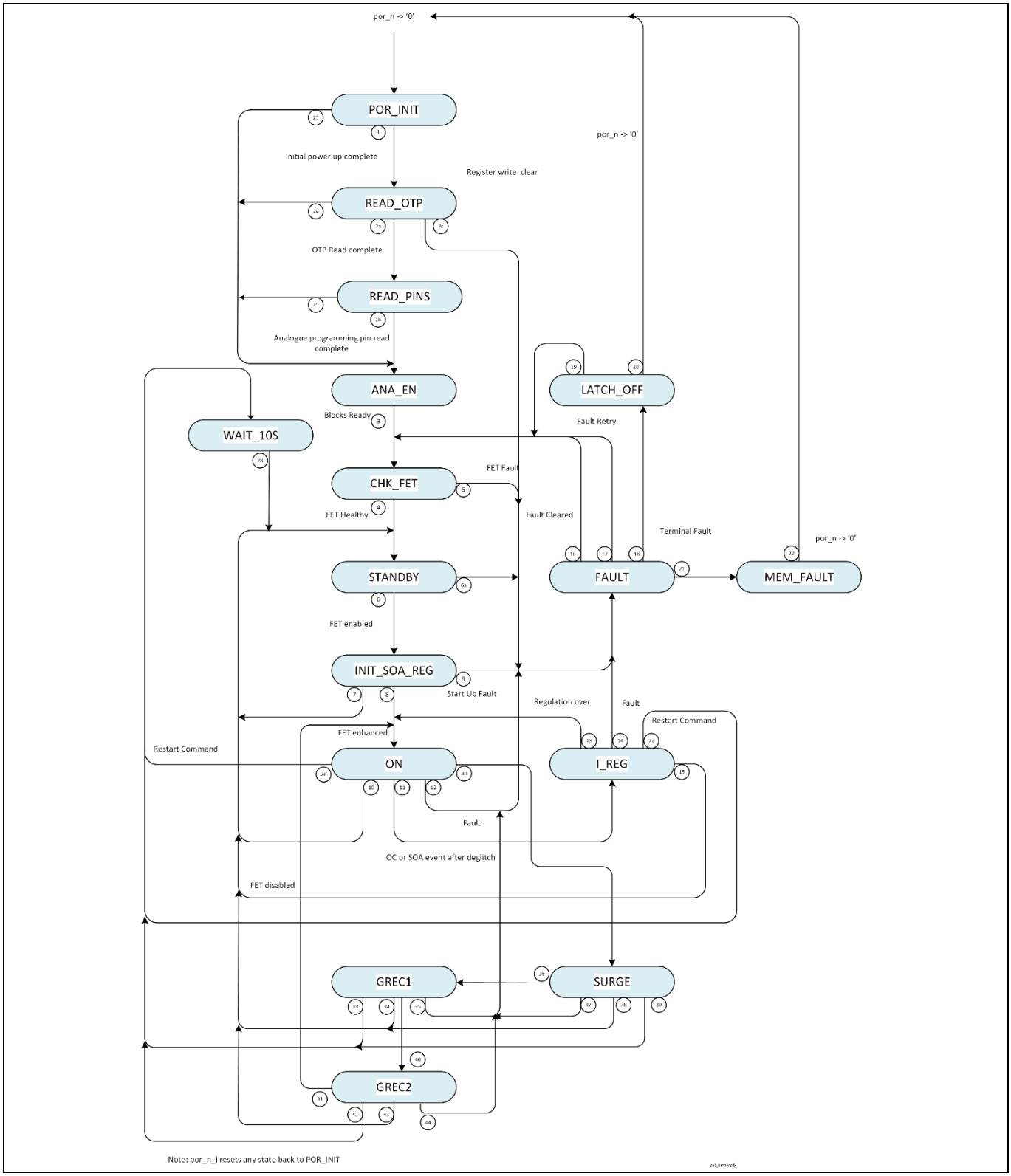


Figure 1 XDP™ XDP711-001 state-machine

2 Hardware setup

Set up the hardware as follows:

1. Apply a voltage at the VDD_VIN and ISNS_P pins:
a) At least 7 V to program commands
b) At least 20 V to program OTP or MTP memory
2. Keep the UV/EN pin at chip GND potential
3. Communication via PMBus® is possible as soon as the standby state is entered. At this point, commands, OTP, or MTP memory can be programmed
4. For successful programming, the internal temperature of the device must stay below 125°C

Figure 2 shows the board set up and the connection of the I²C to the PMBus® communication module.

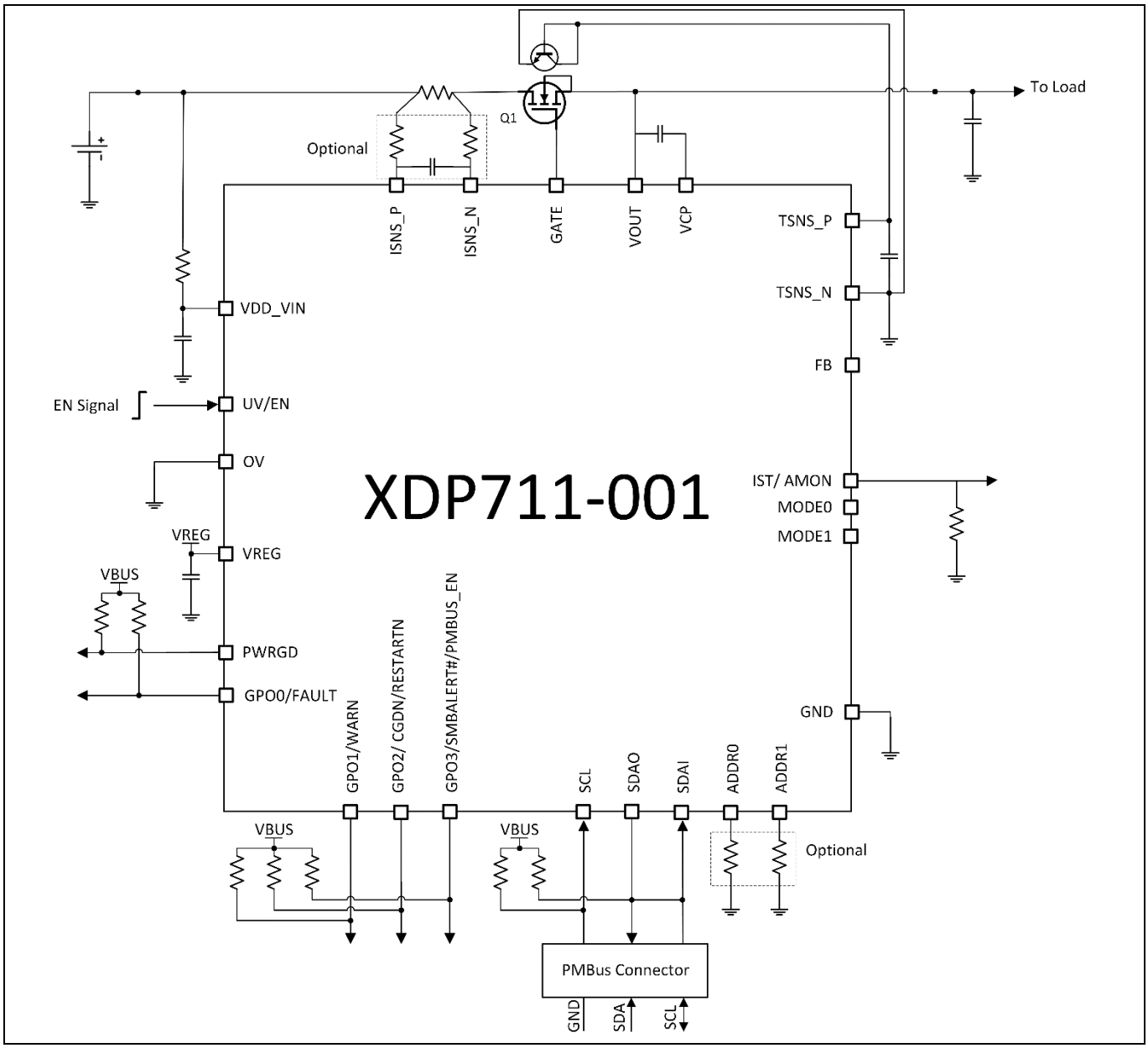


Figure 2 XDP™ XDP711-001 I²C/PMBus® connection

3 Register difference between XDP™ XDP710-002 and XDP™ XDP711-001

The command structure is the same between the two versions of devices, but there are two notable differences as shown in Table 1.

Table 1 Register difference between XDP™ XDP710-002 and XDP™ XDP711-001

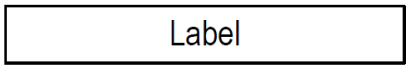
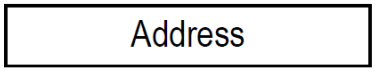
Command	XDP™ XDP710-002	XDP™ XDP711-001
WATCHDOG_TMR (0xD8)	Size: 1 byte	Size: 2 bytes (1 word)
CONFIG_ID (0xFA)	Non-existent	Size: 1 byte

4 PMBus® commands

4.1 PMBus® protocol diagram symbols and elements

Table 2 shows the details of PMBus® protocol symbols, see [1] for more details.

Table 2 PMBus® protocol symbols

Symbol	Description
	An non-shaded rectangle with a number over it represents one or more bits, as indicated by the number, sent from the master to the slave. A label describes the data and includes specific values
	A shaded rectangle with a number over it represents one or more bits, as indicated by the number, sent from the slave to the master. A label describes the data and includes specific values
	A seven-bit destination address is sent from the master to one or more devices (such as with the general call address) on the bus
	The START condition sent from a bus master device
	A REPEATED START condition sent from a bus master device
	A STOP condition sent by a bus master device
	The bit [0] of the address byte indicates the device addressed to write data into the device. Bit [0] will have a value of ‘0’
	The bit [0] of the address byte indicates the device addressed to read data from the device. Bit [0] will have a value of ‘1’
	An acknowledge (ACK) bit is sent from the host
	A negative acknowledgement (NACK) bit is sent from the host
	An acknowledge (ACK) bit is sent from a slave device
	A negative acknowledge (NACK) bit is sent from a slave device

4.2 PMBus® commands communication protocol

- PMBus® command write a byte:
  - PMBus® address+bit[0]] [command] [dataByte]
  - Example: Writing PMBus® command OPERATION (0x01) to value of 0x80
  - [0x20] [0x01] [0x80]

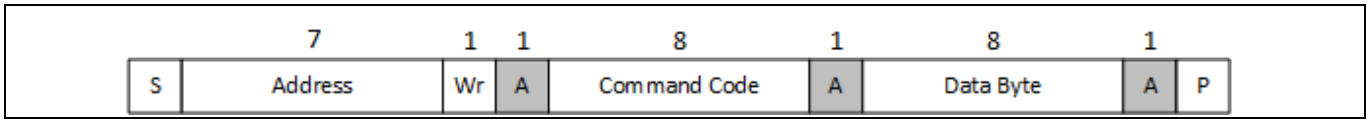


Figure 3 Write byte protocol

- PMBus® command read a byte:
  - [PMBus® address+bit[0]] [command] [PMBus® address+bit[0]] [data0]
  - Example: Reading a PMBus® command OPERATION (0x01). The read value is 0x80
  - [0x20] [0x01] [0x21] [0x80]

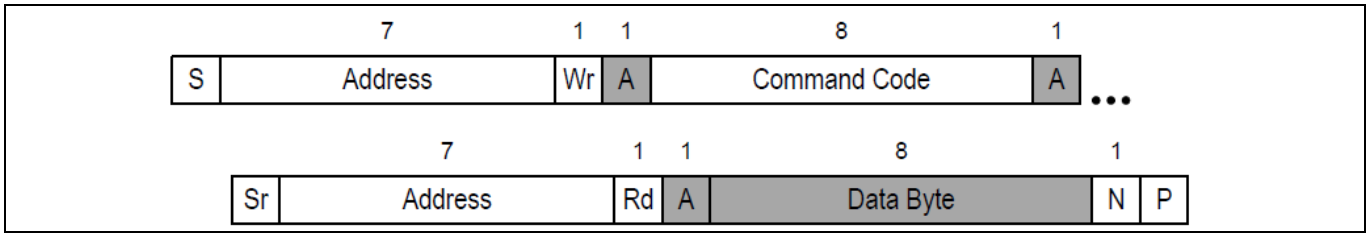


Figure 4 Read byte protocol

- PMBus® command write a word:
  - [PMBus® address+bit[0]] [command] [dataByte0] [dataByte1]
  - Example: Writing PMBus® command VOUT_OV_WARN_LIMIT (0x42) to value of 0x0FFF
  - [0x20] [0x42] [0xFF] [0x0F]

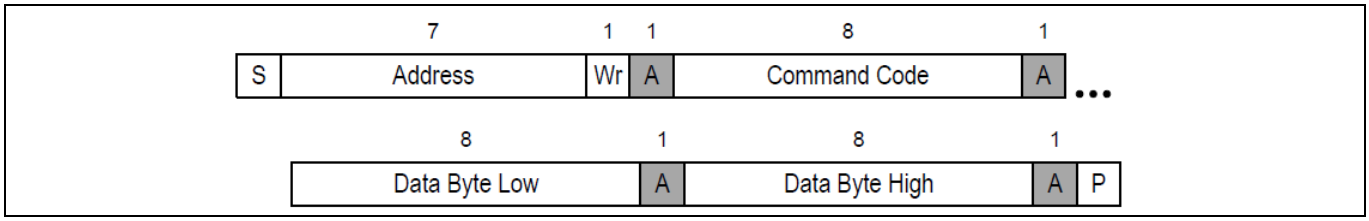


Figure 5 Write word protocol

- PMBus® command read a word:
  - [PMBus® address+bit[0]] [command] [PMBus® address+bit [0]] [dataByte0] [dataByte1]
  - Example: Reading a PMBus® command VOUT_OV_WARN_LIMIT (0x42). The read value is 0x0FFF
  - [0x20] [0x42] [0x21] [0xFF] [0x0F]

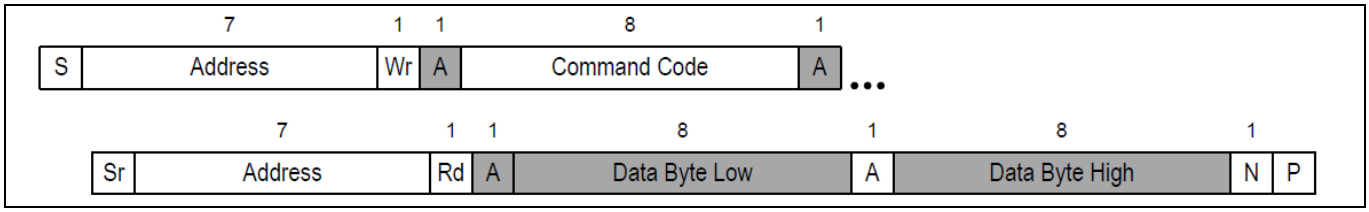


Figure 6 Read word protocol

- PMBus® read a block command:
  - [PMBus® address+bit[0]] [command] [PMBus® address+bit[0]] [number of bytes] [dataByte0] [dataByte1] … [dataByteN]
  - Example: Reading a PMBus® command MFR_ID (0x99). The read value is [0x49] [0x46] [0x00]
  - [0x20] [0x99] [0x21] [0x03] [0x49] [0x46] [0x00]

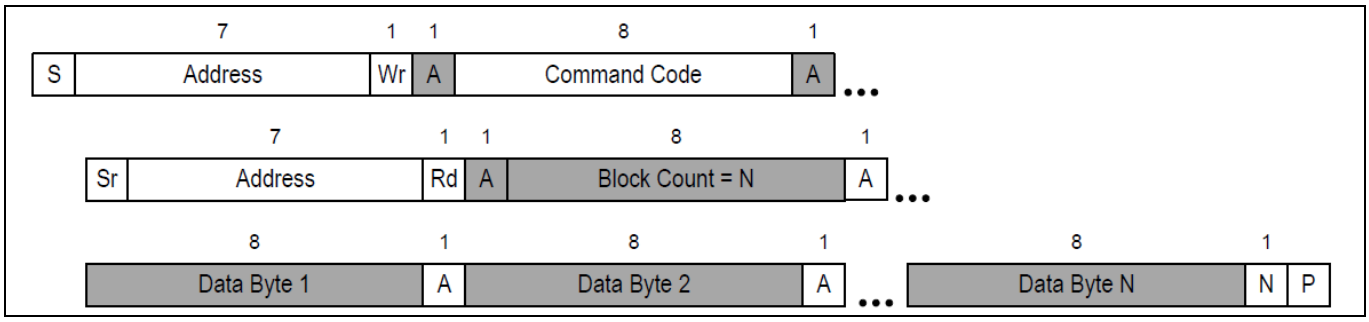


Figure 7 Block-read protocol

5 Production file format details and programming procedure

5.1 Identifying the connected device for programming

Identify the connected device by reading the PMBus® commands 0x99 (MFR ID) and 0x9A (MFR MODEL).

5.1.1 Reading device identification

Use the below commands to identify the connected device.

- Reading a PMBus® command MFR_ID (0x99). This is a block read command. The read value is 0x004649
  - [0x20] [0x99] [0x21] [0x03] [0x49] [0x46] [0x00]
- Reading a PMBus® command MFR_MODEL (0x9A). This is a block read command. The read value is 0x0000313137504458
  - [0x20] [0x9A] [0x21] [0x08] [0x58] [0x44] [0x50] [0x37] [0x31] [0x31] [0x00] [0x00]
- Reading a PMBus® command MFR_REVISION (0x9B). This is a read byte command. The read value is 0x0003
  - [0x20] [0x9B] [0x21] [0x03] [0x00]

Code Listing 1 Reading connected device ID

```
//1. Check MFR_ID
       data = PMBus read block (0x99, 4)
        data_check = data[4] + data[3]<<8 + data[2]<<16
       IF data_check == 0x004649
              True    //MFR identified as Infineon
       Else
              False  //Different Manufacturer

//2. Check MFR_MODEL
       data = PMBus read block (0x9A, 8)
       data_check = data[9] + data[8]<<8 + data[7]<<16 + data[6]<<24 +
data[5]<<32 + data[4]<<40  + data[3]<<48 + data[2]<<56
       IF data_check == 0x000313137504458
              True    //Device identified is XDP711
       Else
              False  //Incorrect device

//3. Check MFR_REVISION
       data = PMBus read byte (0x9B)
       IF data == 0x0003
              True    //Device identified is XDP711-001
       Else
              False  //Incorrect device
```

5.2 Production file format

The production file (.txt) generated from XDP™ Designer GUI includes all the PMBus® commands for storing a complete configuration via volatile registers to OTP and MTP, and SOA data stored into SOA.

Figure 8 shows a snapshot of the production file .txt generated by the XDP Designer GUI.

XDP711-001_example-0x0000AA85.txt - Notepad

File Edit Format View Help

```
//*********************************
// Do not edit any line below CRC32
//*********************************
System Design File CRC32 : 0xFECC540A
System Design File Ver : 1.1
Culture : en-US.
Created Date : 2025-03-30T00:16:09.748Z
GUI Version : 2.1.1Dev_Build-5163
Created by : AgarwalNitis
Company : Infineon Technologies
i2c Address Format : 7-Bit
Number of Devices : 1
//*************************
//Device 0
//*************************
Part Number : XDP711-001
i2c Address : 0x0
PMBus Address : 0x11
Firmware ID : 0x00000000
Configuration Checksum : 0x0000AA85
[Configuration Data]
// pmbus (crc: AA85)
80 00 00 5D 06 74 05 6F 05 22 08 A0 07 E8 0A 8B
0A 45 07 E8 06 59 31 10 3F 02 00 40 00 77 A1 00
00 BF 07 06 00 80 00 00 00 00 00 00 00 FF 7A FF
7A FF 1F FF 1F AF 03 00 06 00 C2 3F FF FF 00 00
00
// soa (crc: 45F1)
FF D4 8E 5B 41 31 26 1F 1A 16 13 11 0F 0D 0C 0A
0A 09 08 07 07 06 06 05 05 05 04 04 04 04 04 03
03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03
03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03
03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03
50 01 8E 00 E4 75
[End Configuration Data]
[End]
```

Figure 8 Production file in .txt format

The production file (.txt) is categorized into two segments of data: PMBus® configuration and safe operating area (SOA) data. All the numbers in the .txt file are hex values.

5.3 PMBus® register configuration and programming to RAM

5.3.1 Interpreting PMBus® register values from production (.txt) file

Figure 9 shows the PMBus® configuration segment of the file. The line 'CRC:' value calculates a checksum of the complete PMBus® configuration segment.

```
// pmbus (crc: AA85)
80 00 00 5D 06 74 05 6F 05 22 08 A0 07 E8 0A 8B
0A 45 07 E8 06 59 31 10 3F 02 00 40 00 77 A1 00
00 BF 07 06 00 80 00 00 00 00 00 00 00 FF 7A FF
7A FF 1F FF 1F AF 03 00 06 00 C2 3F FF FF 00 00
00
```

Figure 9 PMBus® configuration of production file

The CRC (CRC16-CCITT) calculation is from the start of the first data byte (0x80) of the PMBus® register configuration to the last data byte (0x00). The CRC value of the PMBus® commands for the production file is **crc: 0xAA85** as mentioned in the header of the PMBus® configuration Section highlighted in yellow.

The values written in the .txt file are 8-bit hex values. These are the data values of all the PMBus® commands available for OTP and MTP programming. They are written in ascending order of commands, i.e., the first data byte in Figure 9 corresponds to command 0x01 while the last data byte 0x00 corresponds to command 0xFA. For the command with the protocol write byte, only an 8-bit hex value is considered as data while for a command with write word protocol, the first 8-bit hex value is considered as a data byte LOW and the one next to it is considered as data byte HIGH.

Table 3 shows some examples of how to interpret the corresponding command data from the .txt file shown in Figure 9.

Table 3 Command data interpretation example (XDP711-001)

Address/PMBus® command code (hex)	Command name	PMBus® command protocol	Color of the box in Figure 9	Data from .txt file	
0x57	VIN_OV_WARN_LIMIT	Write word	Red	Data byte LOW	Data byte HIGH
				0x8B	0x0A
0xD1	MODE	Write byte	Green	Data byte	
				0x3F	
0xD5	I_SNS_CFG	Write word	Blue	Data byte LOW	Data byte HIGH
				0x77	0xA1

Table 4 shows the corresponding command values translated from the .txt file shown in Figure 9.

Table 4 PMBus® command data translated from .txt file (XDP™ XDP711-001)

Address/PMBus® command code (hex)	PMBus® command name	PMBus® command protocol	Data[n]	Data from .txt file	Command data from .txt file
0x01	OPERATION	Write byte	pmb_data[1]	0x80	0x80
0x42	VOUT_OV_WARN_LIMIT	Write word	pmb_data[2]	0x00	0x0000
			pmb_data[3]	0x00	
0x43	VOUT_UV_WARN_LIMIT	Write word	pmb_data[4]	0x5D	0x065D
			pmb_data[5]	0x06	
0x44	VOUT_UV_FAULT_LIMIT	Write word	pmb_data[6]	0x74	0x0574
			pmb_data[7]	0x05	
0x4A	IOUT_OC_WARN_LIMIT	Write word	pmb_data[8]	0x6F	0x056F
			pmb_data[9]	0x05	
0x4F	OT_FAULT_LIMIT	Write word	pmb_data[10]	0x22	0x0822
			pmb_data[11]	0x08	
0x51	OT_WARN_LIMIT	Write word	pmb_data[12]	0xA0	0x07A0
			pmb_data[13]	0x07	
0x55	VIN_OV_FAULT_LIMIT	Write word	pmb_data[14]	0xE8	0x0AE8
			pmb_data[15]	0x0A	
0x57	VIN_OV_WARN_LIMIT	Write word	pmb_data[16]	0x8B	0x0A8B
			pmb_data[17]	0x0A	
0x58	VIN_UV_WARN_LIMIT	Write word	pmb_data[18]	0x45	0x0745
			pmb_data[19]	0x07	
0x59	VIN_UV_FAULT_LIMIT	Write word	pmb_data[20]	0xEB	0x06EB
			pmb_data[21]	0x06	
0x6B	PIN_OP_WARN_LIMIT	Write word	pmb_data[22]	0x59	0x3159
			pmb_data[23]	0x31	
0xD0	PMBUS™ _CFG	Write byte	pmb_data[24]	0x10	0x10
0xD1	MODE	Write byte	pmb_data[25]	0x3F	0x3F
0xD3	REG_CFG	Write word	pmb_data[26]	0x02	0x0002
			pmb_data[27]	0x00	
0xD4	V_SNS_CFG	Write word	pmb_data[28]	0x40	0x0040
			pmb_data[29]	0x00	
0xD5	I_SNS_CFG	Write word	pmb_data[30]	0x77	0xA177
			pmb_data[31]	0xA1	
0xD6	I_SNS_OFFSET_COMP	Write word	pmb_data[32]	0x00	0x0000
			pmb_data[33]	0x00	
0xD7	TSNS_LVL_CTRL	Write word	pmb_data[34]	0xBF	0x07BF
			pmb_data[35]	0x07	
0xD8	WATCHDOG_TMR	Write byte	pmb_data[36]	0x06	0x0006

Address/PMBus® command code (hex)	PMBus® command name	PMBus® command protocol	Data[n]	Data from .txt file	Command data from .txt file
			pmb_data[37]	0x00	
0xD9	V_TMR	Write word	pmb_data[38]	0x80	0x0080
			pmb_data[39]	0x00	
0xDA	PIN_POLARITY	Write byte	pmb_data[40]	0x00	0x00
0xDB	GPO_CFG	Write word	pmb_data[41]	0x00	0x0000
			pmb_data[42]	0x00	
0xDC	IOUT_UC_WARN_LIMIT	Write word	pmb_data[43]	0x00	0x0000
			pmb_data[44]	0x00	
0xDD	ONCHIP_TSD_FAULT_LIMIT	Write byte	pmb_data[45]	0x00	0x00
0xDE	ENABLE_FAULTS	Write word	pmb_data[46]	0xFF	0x7AFF
			pmb_data[47]	0x7A	
0xDF	MASK_FAULTS	Write word	pmb_data[48]	0xFF	0x7AFF
			pmb_data[49]	0x7A	
0xE1	ENABLE_WARNS	Write word	pmb_data[50]	0xFF	0x1FFF
			pmb_data[51]	0x1F	
0xE2	MASK_WARNS	Write word	pmb_data[52]	0xFF	0x0000
			pmb_data[53]	0x1F	
0xE4	PWRGD_DG_TMR	Write byte	pmb_data[54]	0xAF	0xAF
0xE5	SOA_TMR	Write word	pmb_data[55]	0x03	0x0003
			pmb_data[56]	0x00	
0xE6	TURN_OFF_CTRL	Write word	pmb_data[57]	0x06	0x0006
			pmb_data[58]	0x00	
0xE7	RETRY	Write word	pmb_data[59]	0xC2	0x3FC2
			pmb_data[60]	0x3F	
0xE8	TELEMETRY_EN	Write word	pmb_data[61]	0xFF	0xFFFF
			pmb_data[62]	0xFF	
0xE9	TELEMETRY_CFG	Write word	pmb_data[63]	0x00	0x0000
			pmb_data[64]	0x00	
0xFA	CONFIG_ID	Write byte	pmb_data[65]	0x00	0x00

5.3.2 Write PMBus® commands into RAM

The PMBus® commands configuration must be written to RAM before storing it to OTP/MTP. Do the following:

1. Write all the PMBus® commands to RAM registers as shown in Code Listing 2
   Table 4 shows PMBus® commands that can be stored in OTP and MTP
2. Write these PMBus® commands with user values using the commands protocol described in Section 4

- Example: Write PMBus® command 0x55 VIN_OV_FAULT_LIMIT:
- [0x20] [0x55] [0xE8] [0x0A]

Code Listing 2 XDP™ XDP711-001 steps for writing PMBus® config to RAM

```
// 1. Write the PMBus configuration (PMBus commands) to RAM
          // Write all PMBus commands to according to Table 3
       count = 1
       For command code = start to end

             If protocol == Write byte
                   PMBus write byte (command code, pmb_data[count])
                    count = count + 1
            ELSE
                    data_word = pmb_data[count + 1]<<8 | pmb_data [count]
                    PMBus write word (command code, data_word)
                    count = count + 2

      Next
```

5.3.3 Reading PMBus® register CRC and verifying with config file CRC

All the PMBus® registers that are OTP/MTP programmable are read and then the CRC calculates for these read values. Code Listing 3 shows how the CRC (CCITT-16 FALSE) is performed. The calculated CRC is compared with the PMBus® section CRC in the configuration file. If both match, then the device RAM is programmed correctly and can proceed with the OTP programming.

Code Listing 3 Calculate CRC

```
// 1. Reading OTP/MTP Programmable PMBus registers and calculating CRC
       count = 0
       For command code = start to end
             If protocol == Read byte
                   byte_data =  PMBus read byte (command code)
                    PMBus_data [count] = byte_data
                    count = count + 1
```

Code Listing 3 Calculate CRC

```
              ELSE //Read Word
                     word_data = PMBus read word (command code) //High Byte +
Low Byte<<8
                    PMBus_data [count] = word_data & 0x00FF //Low Byte
                     PMBus_data [count + 1] = (word_data 0xFF00) >> 8 //High
Byte
                     count = count + 2
       Next

// 2. Calculate CRC
       // CRC16-CCITT algorithm reference, this is crc16-ccitt (bad)
implementation as described in
       // http://srecord.sourceforge.net/crc16-ccitt.html
       crcPoly = 0x1021U;
       msb = 0x8000U;
       crc = 0xFFFFU;

       For i = 0 to PMBus_data.size - 1
              ch = PMBus_data[i]
              For j = 0 to 7
                     xor_flag = ((crc ^ (ch << 8)) & msb) != 0U;
                     crc <<= 1;
                     if xor_flag
                            crc ^= crcPoly;
                     ch <<= 1
               Next
        Next
       crc = crc & 0xffff;

// 3. Matching CRC with file CRC
        If crc == file crc
             //Ok to proceed to OTP Programming
        ELSE
             //CRC mismatch, reload config
```

5.4 SOA register configuration and programming to RAM

5.4.1 SOA register interpretation from production (.txt) file

Figure 10 shows the SOA configuration segment of the file. The line 'CRC:' value calculates a checksum of the complete SOA configuration segment.

```
// soa (crc: 45F1)
FF D4 8E 5B 41 31 26 1F 1A 16 13 11 0F 0D 0C 0A
0A 09 08 07 07 06 06 05 05 05 04 04 04 04 04 03
03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03
03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03
03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03
50 01 8E 00 E4 75
```

Figure 10 SOA configuration of production file

The values written in the .txt file are 8-bit hex values. These are the data values of the SOA register. These represent the 43-words that need to be sent out to the PMBus® register 0xD2 in a FIFO as explained in section 5.4.2. The first two bytes represent the 1st word that needs to be sent out. The 1st byte (0xFF) of the 1st word (highlighted in red color box) is the low byte in it and the 2nd byte (0xD4) is the high byte in it. Similarly, the 1st byte (0xE4) of the 43rd word (SOA CRC) (highlighted in blue color box) is the low byte in it and the 2nd byte (0x75) is the high byte in it. The last two bytes, i.e., the 43rd word represents the SOA CRC; therefore, the user does not have to calculate this value and can send this value directly into the controller.

Table 5 shows some examples of how to interpret the corresponding command data from the .txt file shown in Figure 10.

Table 5 SOA data to be programmed in SOA register (XDP711-001)

PMBus® command write word	soa_data[n]	Hex value from SOA config file in ascending order (soa_data[n])	PMBus® write word bit position	Data
1st word	soa_data[1]	0xFF	LOW BYTE	0xD4FF
	soa_data[2]	0xD4	HIGH BYTE	
2nd word	soa_data[3]	0x8E	LOW BYTE	0x5B8E
	soa_data[4]	0x5B	HIGH BYTE	
3rd word	soa_data[5]	0x41	LOW BYTE	0x3141
	soa_data[6]	0x31	HIGH BYTE	
4th word	soa_data[7]	0x26	LOW BYTE	0x1F26
	soa_data[8]	0x1F	HIGH BYTE	
5th word	soa_data[9]	0x1A	LOW BYTE	0x161A
	soa_data[10]	0x16	HIGH BYTE	
6th word	soa_data[11]	0x13	LOW BYTE	0x1113
	soa_data[12]	0x11	HIGH BYTE	
7th word	soa_data[13]	0x0F	LOW BYTE	0x0D0F
	soa_data[14]	0x0D	HIGH BYTE	

PMBus® command write word	soa_data[n]	Hex value from SOA config file in ascending order (soa_data[n])	PMBus® write word bit position	Data
8th word	soa_data[15]	0x0C	LOW BYTE	0x0A0C
	soa_data[16]	0x0A	HIGH BYTE	
9th word	soa_data[17]	0x0A	LOW BYTE	0x090A
	soa_data[18]	0x09	HIGH BYTE	
10th word	soa_data[19]	0x08	LOW BYTE	0x0708
	soa_data[20]	0x07	HIGH BYTE	
11th word	soa_data[21]	0x07	LOW BYTE	0x0607
	soa_data[22]	0x06	HIGH BYTE	
12th word	soa_data[23]	0x06	LOW BYTE	0x0506
	soa_data[24]	0x05	HIGH BYTE	
13th word	soa_data[25]	0x05	LOW BYTE	0x0505
	soa_data[26]	0x05	HIGH BYTE	
14th word	soa_data[27]	0x04	LOW BYTE	0x0404
	soa_data[28]	0x04	HIGH BYTE	
15th word	soa_data[29]	0x04	LOW BYTE	0x0404
	soa_data[30]	0x04	HIGH BYTE	
16th word	soa_data[31]	0x04	LOW BYTE	0x0304
	soa_data[32]	0x03	HIGH BYTE	
17th word	soa_data[33]	0x03	LOW BYTE	0x0303
	soa_data[34]	0x03	HIGH BYTE	
18th word	soa_data[35]	0x03	LOW BYTE	0x0303
	soa_data[36]	0x03	HIGH BYTE	
19th word	soa_data[37]	0x03	LOW BYTE	0x0303
	soa_data[38]	0x03	HIGH BYTE	
20th word	soa_data[39]	0x03	LOW BYTE	0x0303
	soa_data[40]	0x03	HIGH BYTE	
21st word	soa_data[41]	0x03	LOW BYTE	0x0303
	soa_data[42]	0x03	HIGH BYTE	
22nd word	soa_data[43]	0x03	LOW BYTE	0x0303
	soa_data[44]	0x03	HIGH BYTE	
23rd word	soa_data[45]	0x03	LOW BYTE	0x0303
	soa_data[46]	0x03	HIGH BYTE	
24th word	soa_data[47]	0x03	LOW BYTE	0x0303
	soa_data[48]	0x03	HIGH BYTE	
25th word	soa_data[49]	0x03	LOW BYTE	0x0303
	soa_data[50]	0x03	HIGH BYTE	
26th word	soa_data[51]	0x03	LOW BYTE	0x0303

PMBus® command write word	soa_data[n]	Hex value from SOA config file in ascending order (soa_data[n])	PMBus® write word bit position	Data
	soa_data[52]	0x03	HIGH BYTE	
27th word	soa_data[53]	0x03	LOW BYTE	0x0303
	soa_data[54]	0x03	HIGH BYTE	
28th word	soa_data[55]	0x03	LOW BYTE	0x0303
	soa_data[56]	0x03	HIGH BYTE	
29th word	soa_data[57]	0x03	LOW BYTE	0x0303
	soa_data[58]	0x03	HIGH BYTE	
30th word	soa_data[59]	0x03	LOW BYTE	0x0303
	soa_data[60]	0x03	HIGH BYTE	
31st word	soa_data[61]	0x03	LOW BYTE	0x0303
	soa_data[62]	0x03	HIGH BYTE	
32nd word	soa_data[63]	0x03	LOW BYTE	0x0303
	soa_data[64]	0x03	HIGH BYTE	
33rd word	soa_data[65]	0x03	LOW BYTE	0x0303
	soa_data[66]	0x03	HIGH BYTE	
34th word	soa_data[67]	0x03	LOW BYTE	0x0303
	soa_data[68]	0x03	HIGH BYTE	
35th word	soa_data[69]	0x03	LOW BYTE	0x0303
	soa_data[70]	0x03	HIGH BYTE	
36th word	soa_data[71]	0x03	LOW BYTE	0x0303
	soa_data[72]	0x03	HIGH BYTE	
37th word	soa_data[73]	0x03	LOW BYTE	0x0303
	soa_data[74]	0x03	HIGH BYTE	
38th word	soa_data[75]	0x03	LOW BYTE	0x0303
	soa_data[76]	0x03	HIGH BYTE	
39th word	soa_data[77]	0x03	LOW BYTE	0x0303
	soa_data[78]	0x03	HIGH BYTE	
40th word	soa_data[79]	0x03	LOW BYTE	0x0303
	soa_data[80]	0x03	HIGH BYTE	
41st word	soa_data[81]	0x50	LOW BYTE	0x0150
	soa_data[82]	0x01	HIGH BYTE	
42nd word	soa_data[83]	0x8E	LOW BYTE	0x008E
	soa_data[84]	0x00	HIGH BYTE	
43rd word (SOA CRC)	soa_data[85]	0xE4	LOW BYTE	0x75E4
	soa_data[86]	0x75	HIGH BYTE	

5.4.2 SOA data programming onto RAM

Write the SOA data to RAM for SOA-related storing commands.

1. Write word 0xD2 43 times inclusive of CRC as shown in Code Listing 4
   Table 5 shows an example of 43 write word commands
2. Use the PMBus® write word protocol described in Section 4
   - [0x20] [0xD2] [0xFF] [0xD4] … 1st word
   - [0x20] [0xD2] [0x8E] [0x5B] … 2nd word
   - …similarly write PMBus® word from 3rd to 42nd …
   - [0x20] [0xD2] [0xE4] [0x75] … 43rd word
3. Read byte PMBus® command 0xEB STATUS_MEM to check SOA_CRC and SOA_FULL. Verify write SOA register is successful. For details of the command, see Table 6 and Table 7.
   - [0x20] [0xEB] [0x21] [0x03]

Table 6 STATUS_MEM (0xEB)

	7	6	5	4	3	2	1	0
Field	PROG_BLOCK	OTP_FAIL	MTP_FULL	MTP_USER	OTP_USER	SOA_PRG	SOA_FULL	SOA_CRC

Table 7 STATUS_MEM command bits description

Field	Bit	Description	Values
SOA_CRC	0	CRC checks the status of the data inside the FIFO.	0x0 = CRC has not been performed or did not pass 0x1 = CRC passed
SOA_FULL	1	Indication if the FIFO (data + CRC) is full or not.	0x0 = SOA FIFO not full 0x1 = SOA FIFO is full, and CRC has been performed
SOA_PRG	2	SOA section of the OTP if it has been programmed or is virgin.	0x0 = Section is virgin 0x1 = Section has been programmed
OTP_USER	3	Indication if the configuration commands in OTP have been programmed or not.	0x0 = Configuration commands have not been programmed 0x1 = Configuration commands have been programmed
MTP_USER	4	Indication if MTP is in use or not.	0x0 = MTP is not in use 0x1 = MTP is in use
MTP_FULL	5	Indication if all the MTP memory has been used.	0x0 = Not all MTP has been used 0x1 = All MTP has been used

Field	Bit	Description	Values
OTP_FAIL	6:7	Indication if the programming or the readout of the OTP has been successful or not.	0x00 = Programming successful 0x01 = OTP programming started but failed during programming because of the OTP issue. Part must be discarded 0x10 = OTP programming has been blocked because temperature or VIN is out of range 0x11 = OTP programming started but failed during programming because temperature or voltage is out of range during programming. Part must be discarded

Code Listing 4 XDP711-001 steps for writing SOA data onto RAM

```
// 1. Write the SOA data onto PMBus command 0xD2
       // Write PMBus command 0xD2 43 times according to Table 5 values
       command code = 0xD2
       count = 1
       While count != 86
              data_word = soa_data[count + 1]<<8 | soa_data [count]
              PMBus write word (command code, data_word)
              count = count + 1
       Next
// 2. Check SOA written fully and CRC correct
       data = PMBus read byte (0xEB)
       data_check = data & 0x03
       IF data_check == 3
              RETURN true   //SOA CRC successful
       Else
              RETURN false //SOA CRC mismatch
```

5.5 Storing commands to nonvolatile memory (NVM)

The following section explains how to store command configurations to OTP, MTP, and SOA after copying configurations to RAM. Checking necessary programming conditions before programming to OTP, MTP, and SOA

Follow these steps to ensure the temperature and input voltage are within limits for programming the device.

1. Read byte PMBus® command 0xEB STATUS_MEM to check PROG_BLOCK
2. Verify programming conditions favorable for programming by reading the bit 7 of the returned value. For details of this command, see Table 6 and Table 7
   - [0x20] [0xEB] [0x21] [0x00]
3. If bit 7 is high, then programming conditions are not favorable and check for input voltage and temperature to ensure they are within the limits

Code Listing 5 Check to see if programming is blocked or not

```
//1. Check if programming not blocked
       data = PMBus read byte (0xEB)
       data_check = (data & 0x80)
       IF data_check == 0x00
              True    //Programming conditions favorable
       Else
              False  //Programming blocked; VIN or temperature out of range
```

5.5.1 Storing PMBus® commands settings to OTP

Follow the below steps to write the register configuration from RAM to OTP.

1. Write WRITE_OTP (0xEA) PMBus® command with the data option as follows:
   - PMBus® write byte command (WRITE_OTP, writeData)

- WRITE_OTP = 0xEA
- writeData = 0x01

The following data bytes appear on the I²C bus:

- [0x20] [0xEA] [0x01]

2. Read byte PMBus® command 0xEB STATUS_MEM to check OTP_USER. Verify programming is successful. For details of the command, see Table 6 and Table 7.
   - [0x20] [0xEB] [0x21] [0x08]

Code Listing 6 Storing data to OTP

```
//1. Check OTP empty
       data = PMBus read byte (0xEB)
       data_check = (data & 0x08) >> 3
       IF data_check == 0
              True    //OTP available for programming
       Else
              False  //OTP not available for programming

// 2. Write the PMBus command WRITE_OTP
       // Write all PMBus commands 0xEA
       IF True
              PMBus write byte  (0xEA, 0x01)

//3. Check OTP programmed
       data  = PMBus read byte  (0xEB)
       data_check = data & 0xC8
       IF data_check == 0x08
              RETURN true     //OTP Programming successful
       Else
              RETURN false    //OTP Programming failed
```

5.5.2 Storing I_SNS_CFG settings to MTP

Follow the below steps to write the register configuration from RAM to MTP.

1. Write WRITE_OTP PMBus® command with the data option as follows:
   - PMBus® write byte command (WRITE_OTP, writeData)
     - WRITE_OTP = 0xEA
     - writeData = 0x05
   - The following data bytes appear on the I²C bus (see Table 7 for the description of the data byte):
     - [0x20] [0xEA] [0x05]
2. Read byte PMBus® command 0xEB STATUS_MEM to check MTP_USER. Verify programming is successful.
     - [0x20] [0xEB] [0x21] [0x10]

Code Listing 7 Storing I_SNS_CFG to MTP

```
// 1. Check MTP empty
       data = PMBus read byte (0xEB)
       data_check = (data & 0x20) >> 4
       IF data_check == 0
              True    //MTP available for programming
       Else
              False   //MTP not available for programming

// 2. Write all PMBus commands 0xEA
       IF True
              PMBus write byte  (0xEA, 0x05)

// 3. Check MTP programmed
       data  = PMBus read byte  (0xEB)
       data_check = (data & 0xD0) >> 4
       IF data_check == 1
              RETURN true     //MTP Programming successful
       Else
              RETURN false    //MTP Programming failed
```

5.5.3 Storing SOA settings to SOA memory

Follow the below steps to write the register configuration from RAM to SOA OTP.

1. Read byte PMBus® command 0xEB STATUS_MEM to check SOA_CRC and SOA_FULL. Verify write SOA register is successful. For details of the command, see Table 6 and Table 7
   - [0x20] [0xEB] [0x21] [0x03]
2. Write WRITE_OTP PMBus® command with the data option as follows:
   - PMBus® write byte command (WRITE_OTP, writeData)
     - WRITE_OTP = 0xEA
     - writeData = 0x03
   - The following data bytes appear on the I^2C bus (see Table 7 for the description of the data byte):
     - [0x20] [0xEA] [0x03]
3. Read byte PMBus® command 0xEB STATUS_MEM to check SOA_PRG. Verify programming is successful
     - [0x20] [0xEB] [0x21] [0x04]

Code Listing 8 Storing SOA data to SOA memory

```
// 1. Check SOA written fully and CRC correct
       data = PMBus read byte (0xEB)
       data_check = data & 0x03
       IF data_check == 3
              RETURN true   //SOA CRC successful
       Else
              RETURN false //SOA CRC mismatch

// 2. Check SOA empty
       data = PMBus read byte (0xEB)
       data_check = (data & 4) >> 2
       IF data_check == 0
              RETURN True     //SOA available for programming
       Else
              RETURN False    //SOA not available for programming

// 3. Write all PMBus commands 0xEA
       IF True
              PMBus write byte  (0xEA, 0x03)

// 4. Check SOA programmed
       data  = PMBus read byte  (0xEB)
       data_check = data & 0xC4
       IF data_check == 4
```

Code Listing 8 Storing SOA data to SOA memory

```
            RETURN true    //SOA Programming successful
      Else
            RETURN false   //SOA Programming failed
```

5.6 Handling PMBus® address change caused by configuration file

The device and base address of the XDP711-001 device can be modified on the fly via command 0xD0. If the value of this register is modified to be different than the default value of [0x10], then communication with the host device will be lost and the programming will be terminated.

Follow the below steps to point the host to the correct/modified address of the device to maintain the communication and ensure successful programming, for explanation purpose assume the value stored in command 0xD0 is 0x32:

1. First, record the device current 7-bit PMBus® address which for this example is '0x10' and store it in local host memory and call it "cur_addr"
2. Before sending in command 0xD0, read byte PMBus® command 0xF9 STATUS_MODE to check the bit [9:6] for selected device address, verify write SOA register is successful. For details of the command, see Table 8 and Table 9
   - [0x20] [0xF9] [0x21] [0x03] [0xC0]
3. Save the read bits [9:6] for the selected device address in the host local memory and call it "dev_addr"
4. Read the contents of the command 0xD0 and store it in the host memory somewhere and call it "mod_addr"
5. Write the value of the command 0xD0 to the device using the original PMBus® address as follows:

[0x20] [0xD0] [0x32] After writing the command 0xD0, the communication between the host and the device will be lost if the value of 0xD0 is not 0x10 or 0x90. Then, it is based on the 'dev_addr' and 'mod_addr' and the device will be relocated to the new address to establish the communication as follows:

- If 'dev_addr' is equal to 0xF, then the new address will be calculated as new_addr = (mod_addr & 0x7F)
- If "dev_addr" is not equal to 0xF, then the new address will be calculated as new_addr = (mod_addr & 0x70) + dev_addr

With this newly calculated 'new_addr', re-establish the communication with the device and complete the programming procedure.

Table 8 STATUS_MODE (0xF9)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	GATE_ STATUS		PWRGD_STATUS	SELECTED_IST			SELECTED_DEVICE_ ADDRESS				SELECTED_FET				STATUS_ MODE	

Table 9 STATUS_MODE command bits description

Field	Bits	Description	Value
STATUS_MODE	[1:0]	Indicates the current operating mode	2'b00 = FDM with Digital Comparators (DCM) 2'b01 = FDM with Analog Comparators (ACM) 2'b10 = AADM 2'b11 = RESERVED
SELECTED_FET	[5:2]	Shows configured FET	–
SELECTED_DEVICE_ADDRESS	[9:6]	LSB part of the PMBus® address [3:0]	DEVICE ADDRESS 0: 4'b0000 DEVICE ADDRESS 1: 4'b0001 DEVICE ADDRESS 2: 4'b0010 DEVICE ADDRESS 3: 4'b0011 DEVICE ADDRESS 4: 4'b0100 DEVICE ADDRESS 5: 4'b0101 DEVICE ADDRESS 6: 4'b0110 DEVICE ADDRESS 7: 4'b0111 DEVICE ADDRESS 8: 4'b1000 DEVICE ADDRESS 9: 4'b1001 DEVICE ADDRESS 10: 4'b1010 DEVICE ADDRESS 11: 4'b1011 DEVICE ADDRESS 12: 4'b1100 DEVICE ADDRESS 13: 4'b1101 DEVICE ADDRESS 14: 4'b1110 DEVICE ADDRESS programmed in PMBUS_CFG.PMBUS_DEVICE_ADDR: 4'b1111
SELECTED_IST	[12:10]	Shows selected IST	–
PWRGD_STATUS	[13]	Shows PWRGD signal status	–
GATE_STATUS	[15:14]	Shows status of gate pin	–

Code Listing 9 Updating PMBus® address based on configuration file value

```
// 1. Store the device current 7-bit PMBus & store value of 0xD0 command
       cur_addr = Address
       mod_addr = PMBus_data[24]

// 2. Read status 0xF9 and store device address
       data = PMBus read word (0xF9)
       dev_addr = data & 0x03C0

// 3. Write register 0xD0 with original PMBus address
       PMBus write byte  (0xD0, PMBus_data [24])

// 4. Calculate new PMBus address
       IF dev_addr == 0xF
              new_addr =  (mod_addr & 0x7F)

       Else
              new_addr =  (mod_addr & 0x70) + (dev_addr)
```

5.7 Memory check after rebooting

There could be a situation where the data while loading from volatile memory to non-volatile memory could get corrupted. To ensure that the data is copied correctly, do the following:

1. After OTP programming, reduce the voltage at the VDD_VIN pin down to 0 V and then apply 20 V onto the VDD_VIN to reboot the device
2. Use the read byte PMBus® command 0x7E STATUS_CML to check MEM_FAULT. Verify that there is no memory fault

   [0x20] [0x7E] [0x21] [0x00]
3. Run a CRC check on all the data read from the non-volatile memory and compare it with the CRC of the loaded file to ensure that the correct data is programmed onto the OTP and loaded correctly into the volatile memory
4. If the SOA register was programmed, then use the read byte PMBus® command 0xEB STATUS_MEM to check SOA_CRC and SOA_FULL. Verify that valid SOA data is copied from the SOA memory to the volatile memory

   [0x20] [0xEB] [0x21] [0x07]

Code Listing 10 Memory check after reboot

```
// 1. Check memory error
       data  = PMBus read byte  (0x7E)
       data_check = data & 0x10
       IF data_check == 0x10
              RETURN true    // Memory fault occurred, discard the chip
       Else
              RETURN false   // No memory fault


// 2. Reading OTP/MTP Programmable PMBus registers and calculating CRC
        count = 0
        For command code = start to end
               If protocol == Read byte
                     byte_data =  PMBus read byte (command code)
                      PMBus_data [count] = byte_data
                      count = count + 1
              ELSE //Read Word
                      word_data = PMBus read word (command code) //High Byte +
Low Byte<<8
                    PMBus_data [count] = word_data & 0x00FF //Low Byte
                      PMBus_data [count + 1] = (word_data 0xFF00) >> 8 //High
Byte
                      count = count + 2
       Next

// 3. Calculate CRC
       // CRC16-CCITT algorithm reference, this is crc16-ccitt (bad)
implementation as described in
       // http://srecord.sourceforge.net/crc16-ccitt.html
       crcPoly = 0x1021U;
       msb = 0x8000U;
       crc = 0xFFFFU;

       For i = 0 to PMBus_data.size - 1
              ch = PMBus_data[i]
              For j = 0 to 7
                     xor_flag = ((crc ^ (ch << 8)) & msb) != 0U;
                     crc <<= 1;
```

Code Listing 10 Memory check after reboot

```
                    if xor_flag
                          crc ^= crcPoly;
                    ch <<= 1
               Next
        Next
       crc = crc & 0xffff;

// 4. Matching CRC with file CRC
        If crc == file crc
             //data copied correctly to OTP
        ELSE
             //CRC mismatch, memory error

// 5. Check SOA programmed
       data  = PMBus read byte  (0xEB)
       data_check = data & 0xC4
       IF data_check == 4
             RETURN true    //SOA memory programmed
       Else
             RETURN false   //SOA memory un-programmed

 // 6. Check SOA data copied correctly from SOA memory (if programmed)
       data = PMBus read byte (0xEB)
       data_check = data & 0x03
       IF data_check == 3
             RETURN true   //SOA data copied correctly
       Else
             RETURN false //SOA data copied incorrectly; memory error
```

References

[1] Robert V. White, Embedded Power Labs System Management Interface Forum, Inc; *System Management Bus (SMBus) Specification Version 3.1*; 19 Mar 2018; Available online

[2] Infineon Technologies AG: *XDP711-001 product page*; Available online

[3] *CRC16-CCITT*; Available online

Revision history

Document revision	Date	Description of changes
V 1.0	2025-05-30	Initial release

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

Edition 2025-05-30

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2025 Infineon Technologies AG. All Rights Reserved.

Do you have a question about this document?

Email:

erratum@infineon.com

Document reference

UG023442

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.