XDP™ XDP730-001 config file translation and programming guide

About this document

Scope and purpose

XDP™ XDP730-001 is a 30 A wide-input voltage smart e-Fuse with integrated ~1.5 mΩ R_{DS_ON} OptiMOS™ FET, current sensor, digital controller, and a built-in GATE driver. XDP™ XDP730-001 has an internal non-volatile memory (NVM) to store specific command settings. At power-up, during the READ_CFG state, all the settings saved in one-time programmable (OTP) are copied into the 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 non-volatile memory (NVM).

Intended audience

This document is intended for design and application engineers who use XDP™ XDP730-001.

About this product group

- Target applications Servers and datacenters
- 24 V – 48 V industrial systems
- Power distribution systems
- Network routers and switches
- Power supplies
- Test equipment

Product family

XDP™ product family industry's first wide-input voltage range protection IC controller with a programmable digital SOA control. This family of products comprises of 48 V hotswap controllers for both +48 V and -48 V application and has recently introduced the eFuse that has hotswap controller and the FET integrated into one package.

Table of contents

About this document **1**
About this product group **1**
Table of contents **2**
1 Introduction **3**
2 Hardware setup **4**
3 PMBus® commands **5**
3.1 PMBus® protocol diagram symbols and elements 5
3.2 PMBus® commands communication protocol 6
4 Production file format details and programming procedure **8**
4.1 Identifying the connected device for programming 8
4.1.1 Reading device identification 8
4.2 Production file format 9
4.3 PMBus® register configuration and programming to RAM 10
4.3.1 Interpreting PMBus® register values from production (.txt) file 10
4.3.2 Write PMBus® commands into RAM 13
4.3.3 Reading PMBus® register CRC and verifying with config file CRC 13
4.4 Storing commands to non-volatile memory (NVM) 14
4.4.1 Storing PMBus® commands settings to OTP 16
4.4.1.1 OTP programming when the WRITE_PROTECT is disabled 17
4.4.1.2 OTP programming when the WRITE_PROTECT is enabled 18
4.5 Handling PMBus® address change caused by configuration file 20
4.6 Memory check after rebooting 22
5 Related resources **25**
References **26**
Revision history **27**
Disclaimer **28**

1 Introduction

XDP™ XDP730-001 is a wide-input voltage 30 A smart eFUSE that is used in 48 V AI factories, robotics actuators, datacenter, etc. for charging capacitive loads, or hot-plugging events. Since this is a digital switch, all the configuration is done digitally via PMBus® and is stored on the internal memory of the device.

XDP™ XDP730-001 has the following types of memory for programmability:

- Volatile memory
- One-time programmable (OTP)

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.

In this document, the configuration file example is for XDP™ XDP730-001, and the programming procedure is explained based on it.

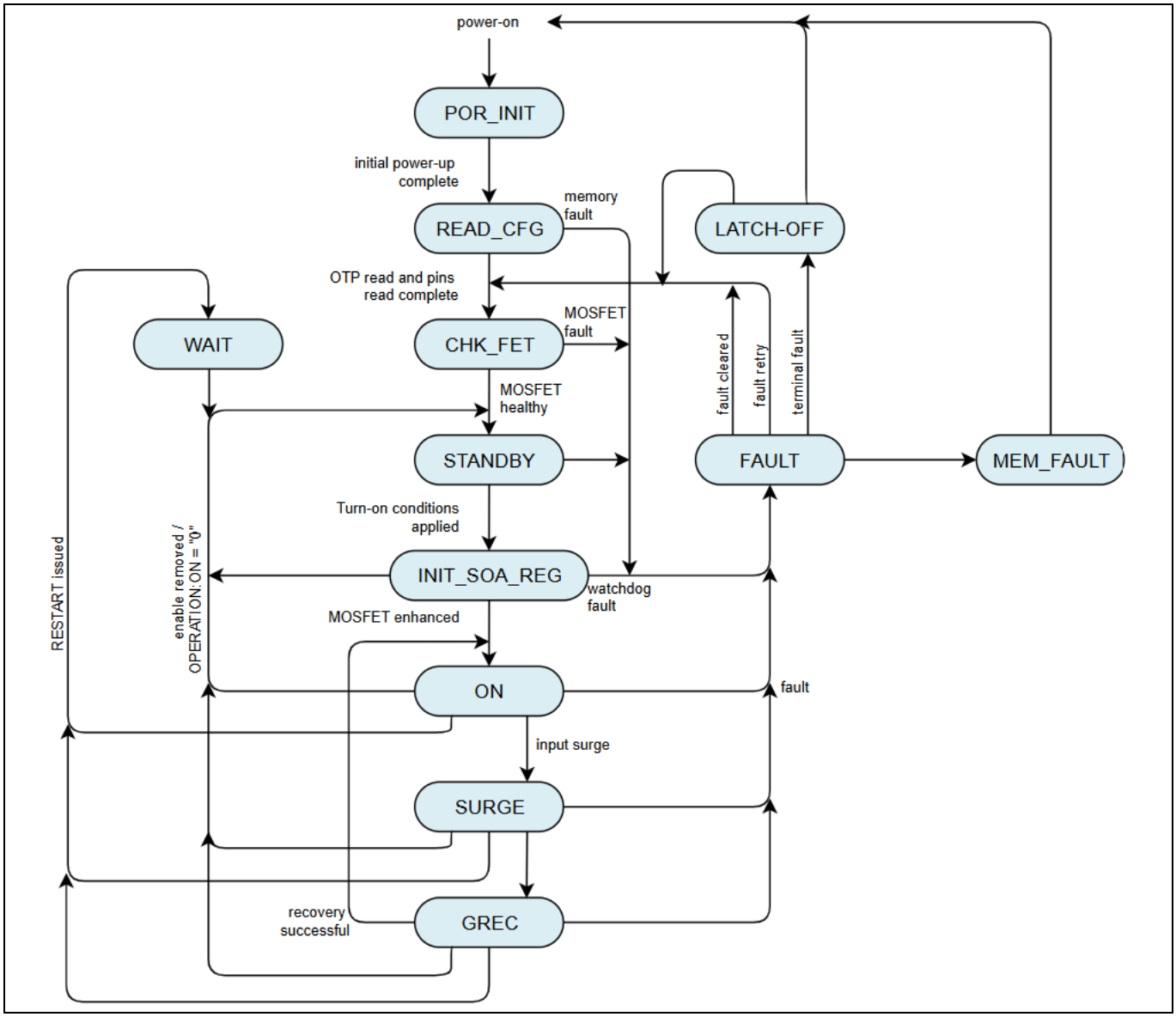


Figure 1 XDP™ XDP730-001 state-machine

2 Hardware setup

Set up the hardware as follows:

1. Apply a voltage at the VDD_VIN (Pin 20) and VINS (Pin 11) pins:
a) At least 7 V to program commands
b) At least 20 V to program OTP or MTP memory
2. Keep the EN pin (Pin 7) at chip GND potential
3. Communication via PMBus® is possible as soon as the standby state is entered. At this point, commands, OTP 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.

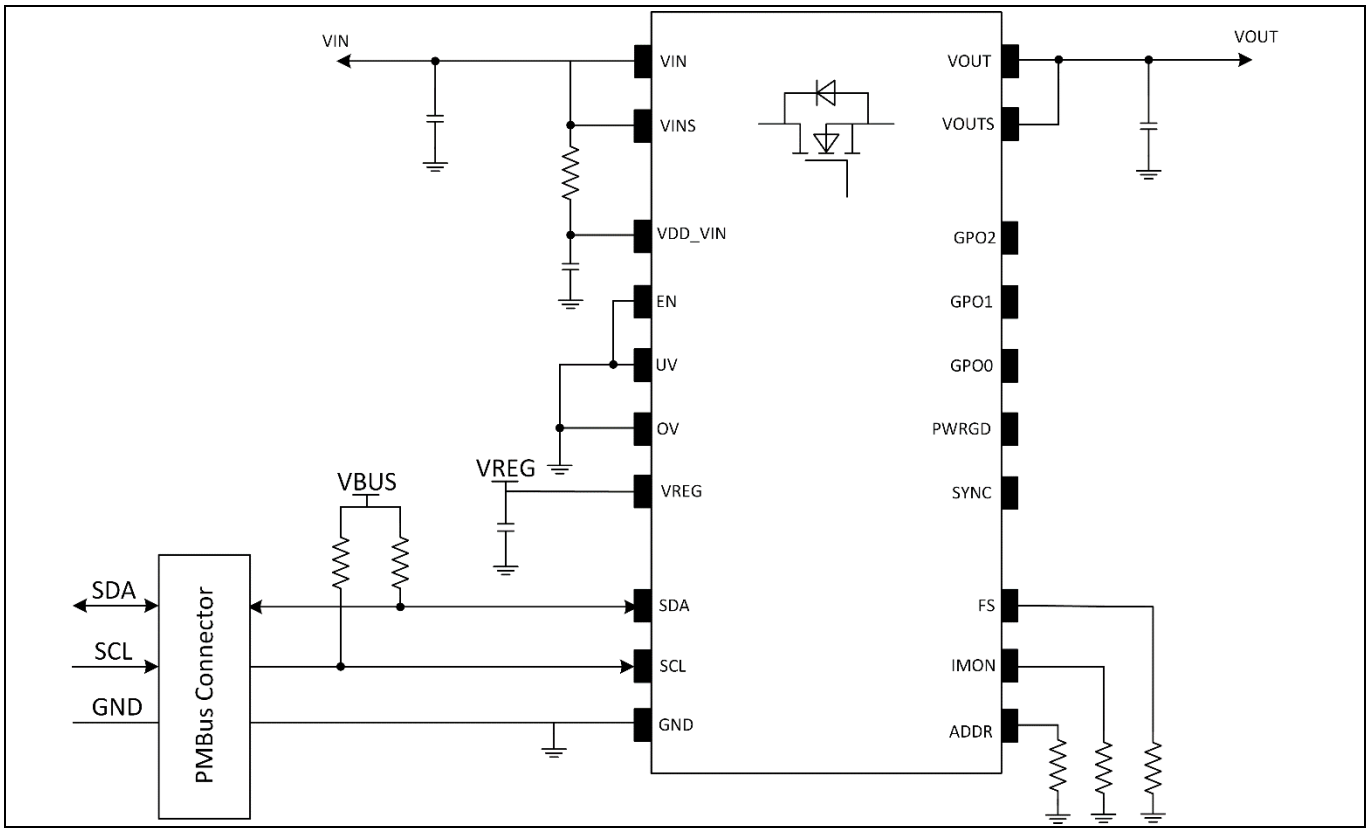


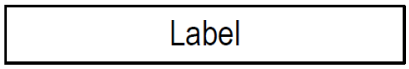
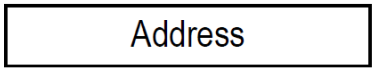
Figure 2 XDP™ XDP730-001 I²C/PMBus® connection

3 PMBus® commands

3.1 PMBus® protocol diagram symbols and elements

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

Table 1 PMBus protocol symbols

Symbol	Description
	A 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.

3.2 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]

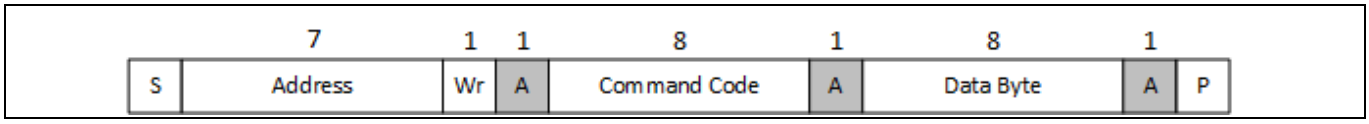


Figure 3 Write byte protocol

- 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]

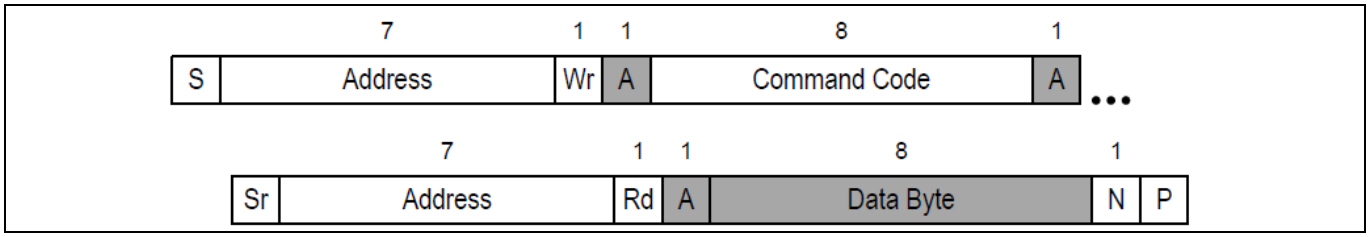


Figure 4 Read byte protocol

- 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]

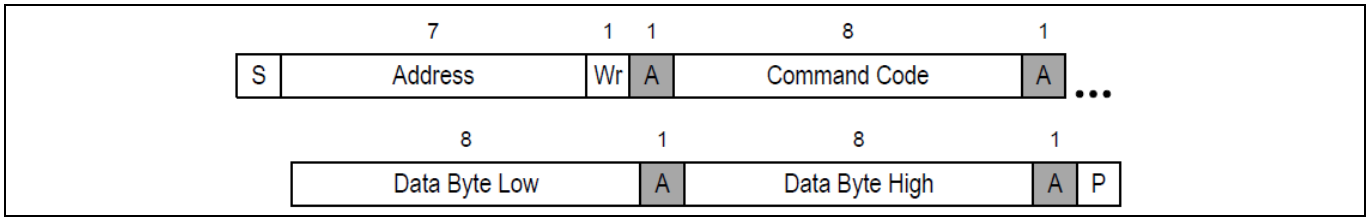


Figure 5 Write word protocol

- 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]

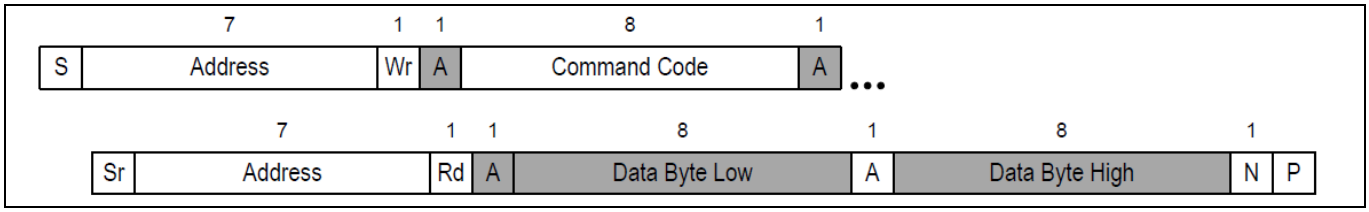


Figure 6 Read word protocol

- 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]

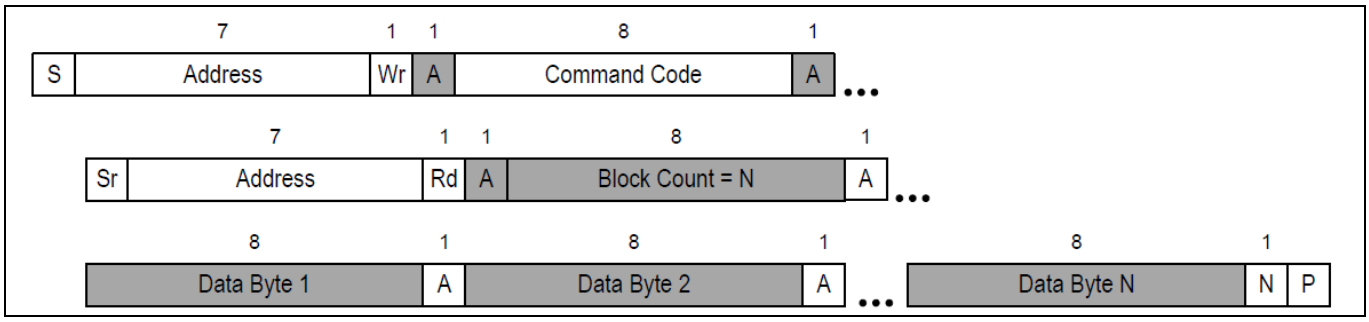


Figure 7 Block-read protocol

4 Production file format details and programming procedure

4.1 Identifying the connected device for programming

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

4.1.1 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 0x0000303337504458.
  - [0x20] [0x9A] [0x21] [0x08] [0x58] [0x44] [0x50] [0x37] [0x33] [0x30] [0x00] [0x00]
- Reading a PMBus® command MFR_REVISION (0x9B). This is a read byte command. The read value is 0x0001.
  - [0x20] [0x9B] [0x21] [0x01] [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 == 0x000303337504458
              True    //Device identified is XDP730
       Else
              False  //Incorrect device

//3. Check MFR_REVISION
       data = PMBus read byte (0x9B)
       IF data == 0x0001
              True    //Device identified is XDP730-001
       Else
              False  //Incorrect device
```

4.2 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.

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

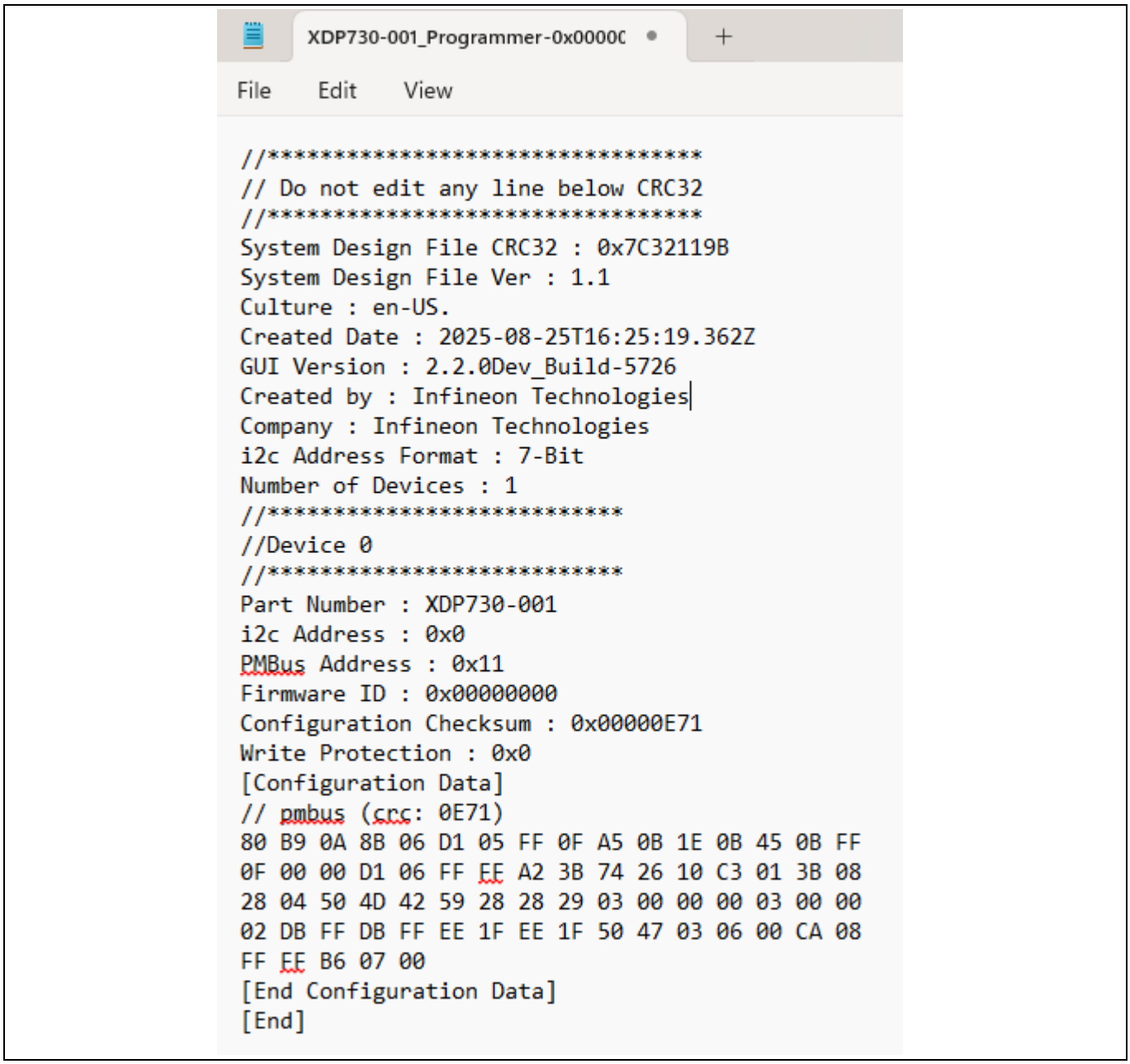


Figure 8 Production file in .txt format

4.3 PMBus® register configuration and programming to RAM

4.3.1 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: 0E71)
80 B9 0A 8B 06 D1 05 FF 0F A5 0B 1E 0B 45 0B FF
0F 00 00 D1 06 FF FF A2 3B 74 26 10 C3 01 3B 08
28 04 50 4D 42 59 28 28 29 03 00 00 00 03 00 00
02 DB FF DB FF EE 1F EE 1F 50 47 03 06 00 CA 08
FF FF B6 07 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: 0x0E71** 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 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 2 shows some examples of how to interpret the corresponding command data from the .txt file shown in Figure 9.

Table 2 Command data interpretation example (XDP730-001)

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
				0xFF	0x0F
0xD0	PMBUS_CFG	Write byte	Green	Data byte	
				0x10	
0xD5	I_SNS_CFG1	Write word	Blue	Data byte LOW	Data byte HIGH
				0x50	0x4D

Table 3 shows the corresponding command values translated from the .txt file shown in Figure 9.

Table 3 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]	0xB9	0x0AB9
			pmb_data[3]	0x0A	
0x43	VOUT_UV_WARN_LIMIT	Write word	pmb_data[4]	0x8B	0x068B
			pmb_data[5]	0x06	
0x44	VOUT_UV_FAULT_LIMIT	Write word	pmb_data[6]	0xD1	0x05D1
			pmb_data[7]	0x05	
0x4A	IOUT_OC_WARN_LIMIT	Write word	pmb_data[8]	0xFF	0x0FFF
			pmb_data[9]	0x0F	
0x4F	OT_FAULT_LIMIT	Write word	pmb_data[10]	0xA5	0x0BA5
			pmb_data[11]	0x0B	
0x51	OT_WARN_LIMIT	Write word	pmb_data[12]	0x1E	0x0B1E
			pmb_data[13]	0x0B	
0x55	VIN_OV_FAULT_LIMIT	Write word	pmb_data[14]	0x45	0x0B45
			pmb_data[15]	0x0B	
0x57	VIN_OV_WARN_LIMIT	Write word	pmb_data[16]	0xFF	0x0FFF
			pmb_data[17]	0x0F	
0x58	VIN_UV_WARN_LIMIT	Write word	pmb_data[18]	0x00	0x0000
			pmb_data[19]	0x00	
0x59	VIN_UV_FAULT_LIMIT	Write word	pmb_data[20]	0xD1	0x06D1
			pmb_data[21]	0x06	
0x6B	PIN_OP_WARN_LIMIT	Write word	pmb_data[22]	0xFF	0xFFFF
			pmb_data[23]	0xFF	
0xCD	VIN_OV_FAULT_LIMIT_S	Write word	pmb_data[24]	0xA2	0x3BA2
			pmb_data[25]	0x3B	
0xCE	VIN_UV_FAULT_LIMIT_S	Write word	pmb_data[26]	0x74	0x2674
			pmb_data[27]	0x26	
0xD0	PMBUS_CFG	Write byte	pmb_data[28]	0x10	0x10
0xD1	MODE	Write byte	pmb_data[29]	0xC3	0x01C3
			pmb_data[30]	0x01	
0xD3	RESERVED	Write word	pmb_data[31]	0x3B	0x083B
			pmb_data[32]	0x08	
0xD4	V_SNS_CFG	Write word	pmb_data[33]	0x28	0x0428
			pmb_data[34]	0x04	
0xD5	I_SNS_CFG1	Write word	pmb_data[35]	0x50	0x4D50
			pmb_data[36]	0x4D	

Address/PMBus® command code (hex)	PMBus® command name	PMBus® command protocol	Data[n]	Data from .txt file	Command data from .txt file
0xD6	I_SNS_CFG2	Write word	pmb_data[37]	0x42	0x5942
			pmb_data[38]	0x59	
0xD7	UV_TH_TMR	Write word	pmb_data[39]	0x28	0x2828
			pmb_data[40]	0x28	
0xD8	WATCHDOG_TMR	Write byte	pmb_data[41]	0x29	0x0329
			pmb_data[42]	0x03	
0xD9	V_TMR	Write word	pmb_data[43]	0x00	0x0000
			pmb_data[44]	0x00	
0xDB	GPO_CFG	Write word	pmb_data[45]	0x00	0x0300
			pmb_data[46]	0x03	
0xDC	IOUT_UC_WARN_LIMIT	Write word	pmb_data[47]	0x00	0x0000
			pmb_data[48]	0x00	
0xDD	ONCHIP_TSD_FAULT_LIMIT	Write byte	pmb_data[49]	0x02	0x02
0xDE	ENABLE_FAULTS	Write word	pmb_data[50]	0xDB	0xFFDB
			pmb_data[51]	0xFF	
0xDF	MASK_FAULTS	Write word	pmb_data[52]	0xDB	0xFFDB
			pmb_data[53]	0xFF	
0xE1	ENABLE_WARNS	Write word	pmb_data[54]	0xEE	0x1FEE
			pmb_data[55]	0x1F	
0xE2	MASK_WARNS	Write word	pmb_data[56]	0xEE	0x1FEE
			pmb_data[57]	0x1F	
0xE4	PWRGD_DG_TMR	Write byte	pmb_data[58]	0x50	0x50
0xE5	SOA_TMR	Write word	pmb_data[59]	0x47	0x0347
			pmb_data[60]	0x03	
0xE6	TURN_OFF_CTRL	Write word	pmb_data[61]	0x06	0x0006
			pmb_data[62]	0x00	
0xE7	RETRY	Write word	pmb_data[63]	0xCA	0x08CA
			pmb_data[64]	0x08	
0xE8	TELEMETRY_EN	Write word	pmb_data[65]	0xFF	0xFFFF
			pmb_data[66]	0xFF	
0xE9	TELEMETRY_CFG	Write word	pmb_data[67]	0xB6	0x07B6
			pmb_data[68]	0x07	
0xFA	CONFIG_ID	Write byte	pmb_data[69]	0x00	0x00

4.3.2 Write PMBus® commands into RAM

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

1. Write all the PMBus® commands to RAM registers as shown in Code Listing 2
   Table 3 shows PMBus® commands that can be stored in OTP
2. Write these PMBus® commands with user values using the commands protocol described in Section 3

- Example: Write PMBus® command 0x55 VIN_OV_FAULT_LIMIT:
- [0x20] [0x55] [0x45] [0x0B]

Code Listing 2 XDP™ XDP730-001 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
```

4.3.3 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
```

4.4 Storing commands to non-volatile memory (NVM)

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

Production file format details and programming procedure

Follow the steps below 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 4 and Table 5.
   - [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 4 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
```

4.4.1 Storing PMBus® commands settings to OTP

XDP730 also offers the Write Protect feature where the write access to the device is not allowed. When WR_PROTECT feature is enabled then it is not possible to send the command to the controller to initiate the OTP programming of the device and thus the indication to enable the WR_PROTECT is stored in a separate section of the configuration file to ensure that it will not interfere with the OTP programming of the device. The section indicating the write protect is highlighted in red box. If the indicator has a value of 0x1 then the WR_PROTECT is enabled and if it has a value of 0x0 then WR_PROTECT is disabled.

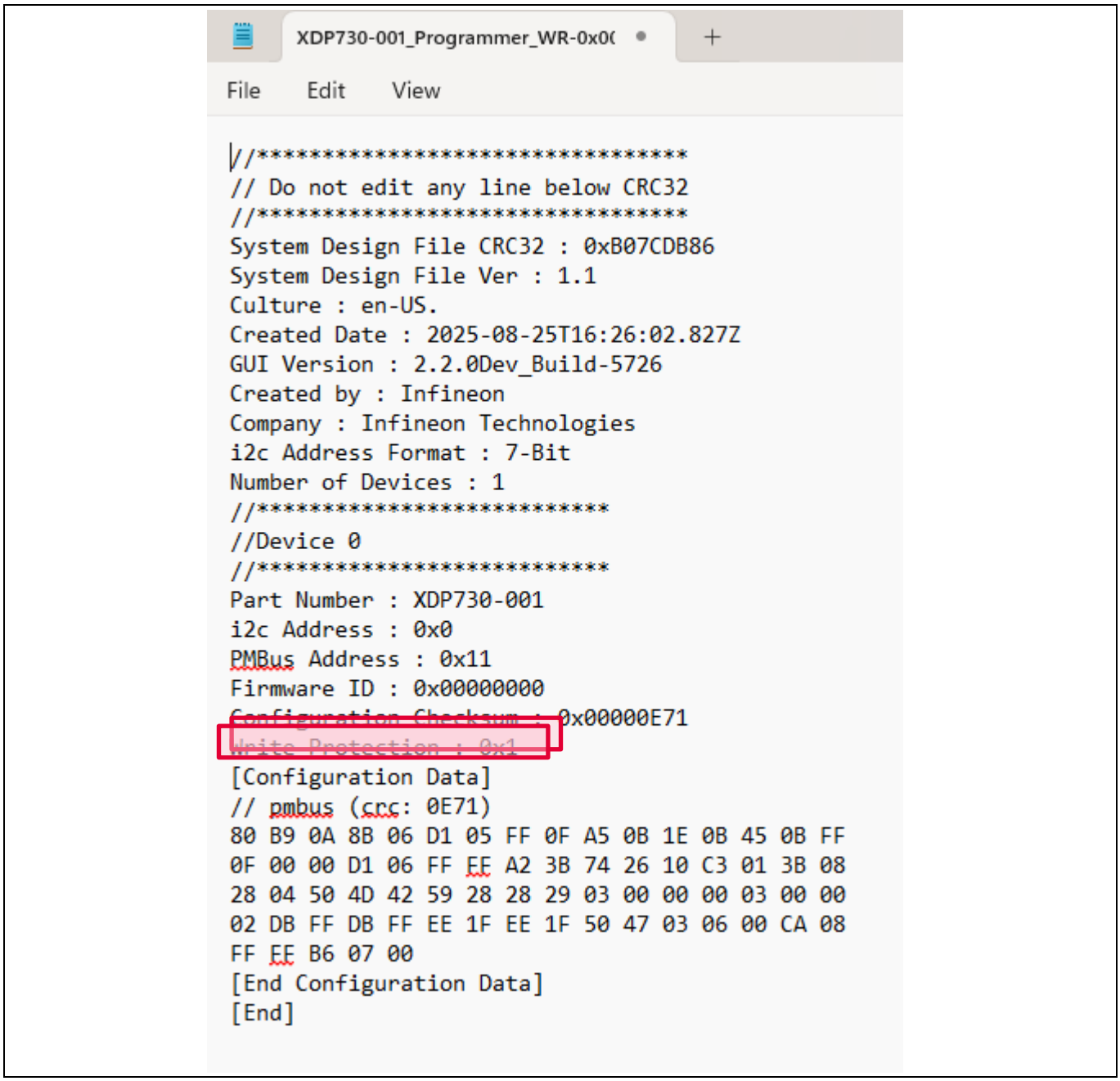


Figure 10 Write protection indicator in production file

4.4.1.1 OTP programming when the WRITE_PROTECT is disabled

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 I^2C 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 4 and Table 5.
   - [0x20] [0xEB] [0x21] [0x08]

Table 4 STATUS_MEM (0xEB)

	7	6	5	4	3	2	1	0
Field	PROG_BLOCK	OTP_FAIL	RESERVED	RESERVED	OTP_USER	RESERVED	RESERVED	RESERVED

Table 5 STATUS_MEM command bits description

Field	Bit	Description	Values
RESERVED	2:0	RESERVED	-
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.
RESERVED	5:4	RESERVED	-
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 5 Storing data to OTP with WR_PROTECT disabled

```
//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
```

4.4.1.2 OTP programming when the WRITE_PROTECT is enabled

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 = 0x09
     - The following data bytes appear on the I²C bus:
       - [0x20] [0xEA] [0x09]
2. Read byte PMBus® command 0xEB STATUS_MEM to check OTP_USER. Verify programming is successful. For details of the command, see Table 4 and Table 5.
   - [0x20][0xEB] [0x21][0x08]
3. Read byte PMBus® command 0xEA WRITE_OTP to check OTP_USER. Verify the write protect is enabled.
   - [0x20][0xEA] [0x21][0x08]

Code Listing 6 Storing data to OTP with WR_PROTECT enabled

```
//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, 0x09)

//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

//4. Check Write Protect
      data  = PMBus read byte  (0xEA)
       data_check = data & 0x08
       IF data_check == 0x08
              RETURN true     //Write protection successful
       Else
              RETURN false    //Write protection not successful, DISCARD!
```

4.5 Handling PMBus® address change caused by configuration file

The device and base address of the XDP730-001 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. For details of the command, see Table 6 and Table 7
   - [0x20] [0xF9] [0x21] [0xC0] [0x03]
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 6 STATUS_MODE (0xF9)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	GATE_STATUS		PWRGD_STATUS	RESERVED			SELECTED_DEVICE_ADDRESS				RESERVED					

Table 7 STATUS_MODE command bits description

Field	Bits	Description	Value
RESERVED	[5:0]	Reserved	–
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
RESERVED	[12:10]	RESERVED	–
PWRGD_STATUS	[13]	Shows PWRGD signal status	–
GATE_STATUS	[15:14]	Shows status of gate pin	–

Code Listing 7 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
```

Code Listing 7 Updating PMBus® address based on configuration file value

```
IF dev_addr == 0xF
      new_addr =  (mod_addr & 0x7F)

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

4.6 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

Code Listing 8 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 8 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 Related resources

- XDP730-001 product page
- XDP730-001 Community
- XDP™ Designer

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: *XDP730-001 product page*; Available online

[3] CRC16-CCITT; Available online

Revision history

Document revision	Date	Description of changes
V 1.0	2026-03-17	Initial release

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

Edition 2026-03-17

Published by

Infineon Technologies AG
Am Campeon 1-15
85579 Neubiberg
Germany

© 2026 Infineon Technologies AG. All Rights Reserved.

Do you have a question about this document?

Email:

erratum@infineon.com

UG092939

Important Notice

Products which may also include samples and may be comprised of hardware or software or both ("Product(s)") are sold or provided and delivered by Infineon Technologies AG and its affiliates ("Infineon") subject to the terms and conditions of the frame supply contract or other written agreement(s) executed by a customer and Infineon or, in the absence of the foregoing, the applicable Sales Conditions of Infineon. General terms and conditions of a customer or deviations from applicable Sales Conditions of Infineon shall only be binding for Infineon if and to the extent Infineon has given its express written consent.

For the avoidance of doubt, Infineon disclaims all warranties of non-infringement of third-party rights and implied warranties such as warranties of fitness for a specific use/purpose or merchantability.

Infineon shall not be responsible for any information with respect to samples, the application or customer's specific use of any Product or for any examples or typical values given in this document.

The data contained in this document is exclusively intended for technically qualified and skilled customer representatives. It is the responsibility of the customer to evaluate the suitability of the Product for the intended application and the customer's specific use and to verify all relevant technical data contained in this document in the intended application and the customer's specific use. The customer is responsible for properly designing, programming, and testing the functionality and safety of the intended application, as well as complying with any legal requirements related to its use.

Unless otherwise explicitly approved by Infineon, Products may not be used in any application where a failure of the Products or any consequences of the use thereof can reasonably be expected to result in personal injury. However, the foregoing shall not prevent the customer from using any Product in such fields of use that Infineon has explicitly designed and sold it for, provided that the overall responsibility for the application lies with the customer.

Infineon expressly reserves the right to use its content for commercial text and data mining (TDM) according to applicable laws, e.g. Section 44b of the German Copyright Act (UrhG).

If the Product includes security features:

Because no computing device can be absolutely secure, and despite security measures implemented in the Product, Infineon does not guarantee that the Product will be free from intrusion, data theft or loss, or other breaches ("Security Breaches"), and Infineon shall have no liability arising out of any Security Breaches.

If this document includes or references software:

The software is owned by Infineon under the intellectual property laws and treaties of the United States, Germany, and other countries worldwide. All rights reserved. Therefore, you may use the software only as provided in the software license agreement accompanying the software.

If no software license agreement applies, Infineon hereby grants you a personal, non-exclusive, non-transferable license (without the right to sublicense) under its intellectual property rights in the software (a) for software provided in source code form, to modify and reproduce the software solely for use with Infineon hardware products, only internally within your organization, and (b) to distribute the software in binary code form externally to end users, solely for use on Infineon hardware products. Any other use, reproduction, modification, translation, or compilation of the software is prohibited. For further information on the Product, technology, delivery terms and conditions, and prices, please contact your nearest Infineon office or visit https://www.infineon.com