XDPP1100 Firmware Number Format

Getting Started in XDPP1100 Custom Firmware Development

About this document

Scope and purpose

This document shows on how to develop custom firmware on XDPP1100 to extend its capabilities beyond the default firmware.

Intended audience

Firmware engineers, Switched Mode Power Supplies (SMPS) engineers

Table of contents

About this document **1**
Table of contents **1**
1 Introduction **2**
2 Physical Quantities **3**
2.1 Integers 3
2.2 Rational Numbers 4
3 Binary Representation **5**
3.1 Unsigned Integers 5
3.1.1 Summary 6
3.2 Signed Integers (Two's Complement) 7
3.2.1 DEC2BIN Conversion Tips 8
3.2.2 BIN2DEC Conversion Tips 8
3.2.3 Summary 8
3.3 Q-Number Format 9
3.3.1 Summary 10
4 Q-Number Arithmetics **11**
4.1 Addition 11
4.2 Substraction 16
4.3 Multiplication & Division 20
5 Exponent-Mantissa Format **21**
6 Pre-Assigned Binary Points Registers **22**
7 PMBus Linear Data Format **23**
8 More Resources **24**
8.1 Firmware Documentation 24
8.2 Register Map Documentation 25
8.3 PMBus Documentation 26
9 References **27**
Revision history **28**

1 Introduction

The XDPP1100 is a digital power supply controller offering superior levels of integration and performance in a single-chip solution. The flexible nature of the IC makes it suitable for a wide variety of power conversion application. Multiple peripherals inside the device have been specifically optimized to enhance the performance of isolated DC-DC applications and reduce the solution component count in the IT and network infrastructure space.

At the core of the XDPP1100 controller are the digital control loop peripherals. Each implements a high-speed digital control loop consisting of a dedicated voltage Analog-to-Digital Converter (ADC), a high resolution current ADC, a PID-based digital compensator, and DPWM outputs with 78.125-ps pulse width resolution. The device also offers 6 channels of 9-bit, 1Msps general purpose ADC, timers, interrupt control, PMBus, and two I2C communications ports.

The device is based on a 32-bit, 100MHz ARM® Cortex™-M0 RISC microcontroller that performs real-time monitoring, configures peripherals, and manages communications. The ARM® microcontroller executes its program out of programmable OTP as well as on-chip RAM and ROM.

The default firmware in XDPP1100 comes with vast list of features such as support for multiple power topologies commonly found in DC/DC server/telecom bricks power conversion, built-in PMBus commands, voltage mode control (VMC), Peak Current Mode Control (PCMC), and many others.

The default firmware made several assumptions on the system with the good intention for minimum programming effort at the system level. However, it is still possible to develop custom firmware for XDPP1100 to fulfill different system requirements as well as for proprietary control algorithms.

ARM® Cortex™-M0 used as the CPU core of XDPP1100 has no Floating-Point Unit (FPU) and only support fixed-point arithmetics.

Critical peripherals in XDPP1100 such as ADC, PWM, PID Controller and FF (FeedForward) Block are designed to be tightly coupled and operated together to enable high performance DC/DC power conversion. Users only need to configure the right registers to enable certain features in the PID and FF block. The hardwares will take care of all the calculations.

For these reasons, there is a need to have a number formating to represent real-world values, such as voltage, current and temperature that are being processed in the hardware.

In addition, PMBus comes with its own formats which are elaborated briefly in this section. For more details, user is encouraged to refer to the PMBus standards directly from http://www.pmbus.org/Home.

This section discuss on the fixed-point arithmetics and number representation used in XDPP1100 and how it interacts with PMBus number format.

2 Physical Quantities

2.1 Integers

Examples of integers are -4, -3, -2, -1, 0, 1, 2, 3 and 4. Integers have no fractions/decimal points.

Integers can be thought of as **discrete**, equally spaced points on an infinitely long number line. It can visually be represented in the following diagram:

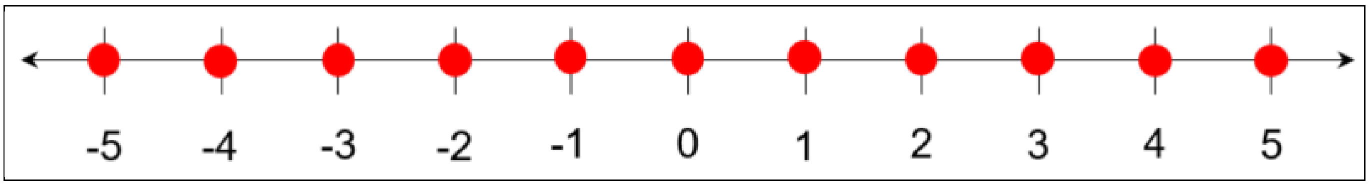


Figure 1 Integers Visualization

Positive integers, with zero included, are also known as **Whole Numbers.**

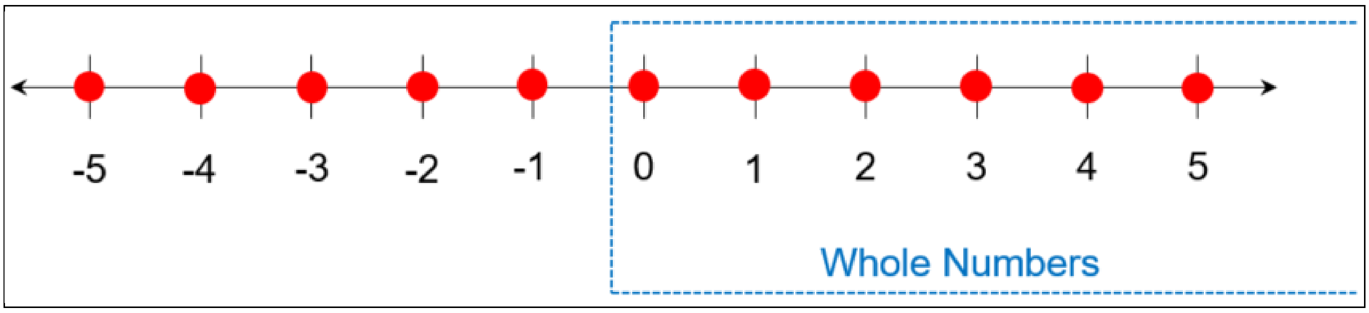


Figure 2 Whole Numbers Visualization

Some physical quantities quantified with integers/whole numbers are: number of steps taken per day, number of persons in the room.

Some physical quantities quantified with integers/whole numbers in digital controller:

- Number of ADC samples needed to trigger fault
- Number of data to be transmitted/received

2.2 Rational Numbers

Examples of rational numbers are -1.3, -1.0, -0.7, 0.0, 0.7 and 1.3.

Rational numbers includes Integers as well as fractions/decimal points. It excludes special numbers such as π, *e* and roots (√). It can visually be represented in the following diagram:

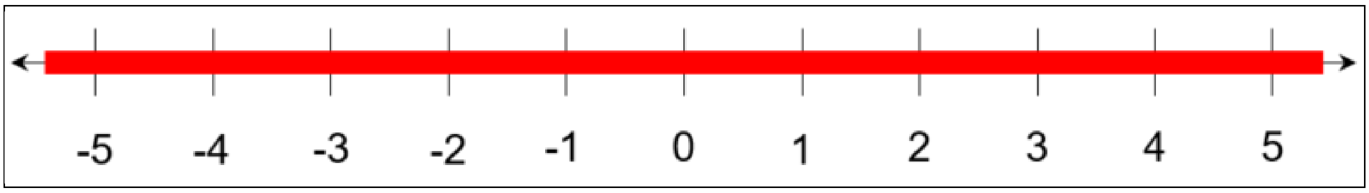


Figure 3 Rational Numbers Visualization

Most physical quantities related to electrical and power such as Voltage (V), Current (A), Power (W) and Temperature (T) can be represented by rational numbers.

Time (sec, nanosec), usually is represented by positive rational numbers.

3 Binary Representation

Representing the abovementioned physical quantities into the computer has its own challenges because computers only recognize and operate with **binary numbers** in a form of 1 and 0.

This section discuss on how binary numbers can be manipulated for representing the real-world physical quantities.

3.1 Unsigned Integers

When binary numbers (BIN) are grouped together with certain length (a.k.a. N-bit length), it can be used to represent Whole Number (DEC). In addition, binary numbers can be converted into Hexadecimal numbers (HEX) to shorten the need of writing a long numbers.

Table 1 BIN2DEC: Binary Representation of Whole Number using Unsigned Integers

	Min Value			Max Value		
Bit Length	**BIN**	**HEX**	**DEC**	**BIN**	**HEX**	**DEC**
4-bit	0000	0x0	**0**	1111	0xF	**15**
8-bit	0000 0000	0x00	**0**	1111 1111	0xFF	**255**

The following table shows some worked examples of Binary Numbers and its Hexadecimal equivalents.

Table 2 DEC2BIN: Worked examples of Unsigned Integers

Bit Length	**Value (DEC)**	**Value (BIN)**
4-bit	10	**1010**
	9	**1001**
8-bit	171	**1010 1011**
	205	**1100 1101**

Table 3 BIN2DEC: Worked examples of Unsigned Integers

Bit Length	**Value (BIN)**	**Value (DEC)**
4-bit	1101	**14**
	0100	**4**
8-bit	1000 1000	**136**
	0111 1010	**122**

It is also interesting to note that binary numbers have **discrete** property, the same to the real-world physical Integers. This can be observed when the right-most, i.e. **Least Significant Bit (LSB)**, of the binary number is incremented/decremented by 1.

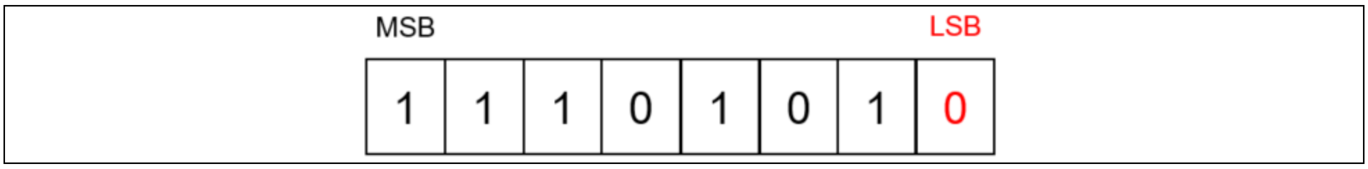


Figure 4 LSB Illustration

Table 4 **BIN numbers Discrete Property**

Bit Length	Value (BIN)	Value (DEC)
4-bit	1010	10
	1011	11
8-bit	1010 1010	170
	1010 1011	171

It can be seen that the **Whole Numbers (positive integers)** can be easily represented by a *grouped Binary Numbers/Hexadecimal Numbers*. In computer science, this representation is called as “**Unsigned Integer**”.

3.1.1 Summary

Mathematically, Unsigned Integers can be summarized as the following:

Table 5 **Unsigned Integers Formulae**

Property	Formula
Bit Length	N
Min Value (DEC)	0
Max Value (DEC)	$2^N - 1$

Unfortunately, this is still not enough to fully represent the negative range of real-world physical Integer number. For this reason, “**Signed Integer**” representation is introduced.

3.2 Signed Integers (Two's Complement)

A sign-bit can be embedded in the binary numbers to indicate positive/negative integer and complete the representation of real-world physical Integers number. The sign-bit is usually placed at the left-most, i.e. **Most Significant Bit (MSB)**, of the binary numbers at the expense of losing the number range.

Positive number has "0" as the sign-bit whereas negative number has "1" as the sign-bit.

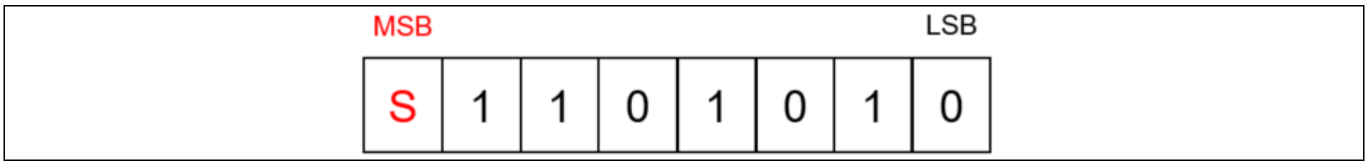


Figure 5 MSB Illustration with Sign-bit

In computer science, a mathematical operation called "Two's Complement" is used to represent the full range of real-world physical Integers. There is another operation called "One's Complement" which is not used in XDPP1100.

Table 6 BIN2DEC: Binary Representation of Integers using Signed Integers

	Min Value			Max Value		
Bit Length	**BIN**	**HEX**	**DEC**	**BIN**	**HEX**	**DEC**
4-bit	1000	0x8	**-8**	0111	0x7	**7**
8-bit	1000 0000	0x80	**-128**	0111 1111	0x7F	**127**

Some worked examples of Signed Integers usingTwo's Complement are shown below.

Table 7 DEC2BIN: Worked Examples of Signed Integers

Bit Length	**Value (DEC)**	**Value (BIN)**
4-bit	6	**0110**
	-6	**1010**
8-bit	100	**0110 0100**
	-100	**1001 1100**

Table 8 BIN2DEC: Worked Examples of Signed Integers

Bit Length	**Value (BIN)**	**Value (DEC)**
4-bit	0100	**4**
	1001	**-7**
	0000	**0**
	1111	**-1**
8-bit	0100 0101	**69**
	1101 1010	**-38**
	0000 0000	**0**
	1111 1111	**-1**

3.2.1 DEC2BIN Conversion Tips

Sometimes it is handy to know a quick number conversion tricks.

DEC2BIN quick example: Convert -6 to BIN.

1. Convert absolute value of Whole Number to binary:	*6 in binary is 0110*
2. Flip all the bits. 1 becomes 0 and 0 becomes 1:	*0110 becomes 1001*
3. Add 1 to LSB:	*1001 + 1 = 1010*

Therefore, -6 in Binary is 1010.

3.2.2 BIN2DEC Conversion Tips

BIN2DEC quick example: Convert 1010 to DEC.

1. Flip all the bits. 1 becomes 0 and 0 becomes 1:	*1010 becomes 0101*
2. Add 1 to LSB:	*0101 + 1 = 0110*
3. Convert binary number to Whole Number, add the sign:	-(0110) in decimal is -6

Therefore, 1010 in Decimal is -6.

3.2.3 Summary

Mathematically, Signed Integers can be summarized as following:

Table 9 Signed Integers Formulae

Property	Formula
Bit Length	N
Min Value (DEC)	$-(2^{(N-1)})$
Max Value (DEC)	$2^{(N-1)}-1$

We are now able to fully represent real-world physical Integer number. Unfortunately, we are still unable to represent fractions and decimal points. This is the last piece of the puzzle in order to represent Rational Numbers. For this reason, **Q-Number format** representation is introduced.

3.3 Q-Number Format

Q-Number format enables fraction and decimal points representation on Signed Integers representation, making Rational Numbers possible in the CPU.

Q-Number format is usually denoted as **Qm.n** where:

- **m** represents the **Integers**
- **n** represents the **decimal points/fractions**

Using Q-Number format perspective:

- **Unsigned Integer** representation can be denoted as **Ux.0**
- **Signed Integer** representation can be denoted as **Sx.0**

Table 10 Unsigned Integer (U8.0) and Signed Integer (S8.0) in Qm.n

Format	Bit Length	Min Value		Max Value	
		BIN	DEC	BIN	DEC
U8.0	8	0000 0000	0	1111 1111	255
S8.0	8	1000 0000	-128	0111 1111	127

Adding decimal points/fraction can be done by setting the value of “n”. Setting **n** to **3** on the above U8.0 and S8.0 examples yield Q-Numbers of U8.3 and S8.3, respectively. Adding “n” will also lengthen the binary numbers from 8 to 11 and therefore the new Bit Length can be calculated as:

$$Bit\ Length\ (N) = m + n$$

In U8.0 and S8.0, every increment of LSB correspond to an increased value of 1 in DEC value. In U8.3 and S8.3 however, every increment of LSB correspond to different value. This property is called LSB Weight, which can be calculated as:

$$LSB\ Weight = 2^{-(n)}$$

Quick calculation will show that for U8.0 and S8.0, the LSB weight is 1 whereas for U8.3 and S8.3, the LSB weight is 0.125.

Table 11 Unsigned Integer (U8.3) and Signed Integer (S8.3) in Qm.n DEC

Format	Bit Length	LSB Weight	Min Value (DEC)	Max Value (DEC)
U8.3	8 + 3 = 11	2^(-3) = 0.125	0	$(2^{(11)} - 1) * (2^{(-3)}) = 255.875$
S8.3	8 + 3 = 11	2^(-3) = 0.125	$(-(2^{(11-1)})) * (2^{(-3)}) = -128$	$(2^{(11-1)} - 1) * (2^{(-3)}) = 127.875$

U8.3 and S8.3 representation in binary numbers:

Table 12 Unsigned Integer (U8.3) and Signed Integer (S8.3) in Qm.n BIN

Format	Bit Length	LSB Weight	Min Value (BIN)	Max Value (BIN)
U8.3	8 + 3 = 11	2^(-3) = 0.125	0000 0000.000	1111 1111.111
S8.3	8 + 3 = 11	2^(-3) = 0.125	1000 0000.000	0111 1111.111

Notice that the Binary representation of the Q-Number is still conformant to the usual Two’s Complement.

More worked examples of Q-Number formats below:

Table 13 Worked examples of Qm.n numbers Set 1

Format	Bit Length	LSB Weight	Min Value (DEC)	Max Value (DEC)
U8.3	8 + 3 = 11	2^(-3) = 0.125	0	255.875
U3.8	8 + 3 = 11	2^(-8) = 0.00390625	0	7.99609375
S8.3	8 + 3 = 11	2^(-3) = 0.125	-128	127.875
S3.8	8 + 3 = 11	2^(-8) = 0.00390625	-4	3.99609375

Table 14 Worked examples of Qm.n numbers Set 2

Format	Bit Length	LSB Weight	Min Value (DEC)	Max Value (DEC)
U8.0	8 + 0 = 8	2^(-0) = 1	0	255
U0.8	8 + 0 = 8	2^(-8) = 0.00390625	0	0.99609375
S8.0	8 + 0 = 8	2^(-0) = 1	-128	127
S0.8	8 + 0 = 8	2^(-8) = 0.00390625	-0.5	0.49609375

Table 15 Worked examples of Qm.n numbers Set 3

Format	Bit Length	LSB Weight	Min Value (DEC)	Max Value (DEC)
U8.-3	8 + (-3) = 5	2^(-(-3)) = 8	0	248
U3.-8	3 + (-8) = -5 (Invalid)			
S8.-3	8 + (-3) = 5	2^(-(-3)) = 8	-128	120
S3.-8	3 + (-8) = -5 (Invalid)			

Table 16 Worked examples of Qm.n numbers Set 4

Format	Bit Length	LSB Weight	Min Value (DEC)	Max Value (DEC)
U-8.3	-8 + 3 = -5 (Invalid)			
U-3.8	-3 + 8 = 5	2^(-8) = 0.00390625	0	0.12109375
S-8.3	-8 + 3 = -5 (Invalid)			
S-3.8	-3 + 8 = 5	2^(-8) = 0.00390625	-0.0625	0.05859375

3.3.1 Summary

Mathematically, Qm.n can be summarized as following:

Table 17 Qm.n Formulae

Property		Formula	
Bit Length (N)		m + n	
LSB Weight		$2^{(-n)}$	
Um.n	Min Value	0	Min Value of Um.0 * LSB Weight
	Max Value	$(2^{(N)} - 1) * (2^{(-n)})$	Max Value of Um.0 * LSB Weight
Sm.n	Min Value	$(-(2^{(N-1)})) * (2^{(-n)})$	Min Value of Sm.0 * LSB Weight
	Max Value	$(2^{(N-1)} - 1) * (2^{(-n)})$	Max Value of Sm.0 * LSB Weight

4 Q-Number Arithmetics

4.1 Addition

Table 18 **Example 1 (U8.0 + U8.0 = U9.0)**

	Op 1 (U8.0)	Op 2 (U8.0)	Result (U9.0)
BIN	0110 1001.	0110 1010.	1. Convert to binary: Operand 1 = 105. In binary: 0110 1001. (U8.0) Operand 2 = 106. In binary: 0110 1010. (U8.0) 2. Sign extend both Operands: Operand 1 = 0 0110 1001. (U9.0) Operand 2 = 0 0110 1010. (U9.0) 3. Add both Operands to get Result: Result = 1 0001 0100. (U9.0) Result = 211
DEC	105	106	211

Table 19 **Example 2 (U8.0 + U8.0 = U9.0)**

	Op 1 (U8.0)	Op 2 (U8.0)	Result (U9.0)
BIN	1010 1010.0	0110 1010.0	1. Convert to binary: Operand 1 = 170. In binary: 1010 1010. (U8.0) Operand 2 = 106. In binary: 0110 1010. (U8.0) 2. Sign extend both Operands: Operand 1 = 0 1010 1010. (U9.0) Operand 2 = 0 0110 1010. (U9.0) 3. Add both Operands to get Result: Result = 1 0001 0100. (U9.0) Result = 276
DEC	170	106	276

Table 20 Example 3 (U8.0 + U4.4 = U9.4)

	Op 1 (U8.0)	Op 2 (U4.4)	Result (U9.4)
BIN	1010 1010.	0110.1010	1. Convert to binary: Operand 1 = 170. In binary: 1010 1010. (U8.0) Operand 2 = 106. In binary: 0110.1010 (U4.4) 2. Align LSBs of both Operands: Operand 1 = 1010 1010.0000 (U8.4) Operand 2 = 0110.1010 (U4.4) 3. Sign extend both Operands: Operand 1 = 0 1010 1010.0000 (U9.4) Operand 2 = 0 0000 0110.1010 (U9.4) 4. Add both Operands to get Result: Result = 0 1011 0000.1010 (U9.4) Result = 176.625
DEC	170	6.625	176.625

Table 21 Example 4 (U8.0 + U0.8 = U9.8)

	Op 1 (U8.0)	Op 2 (U0.8)	Result (U9.8)
BIN	1010 1010.	.0110 1010	1. Convert to binary: Operand 1 = 170. In binary: 1010 1010. (U8.0) Operand 2 = 0.4140625. In binary: .0110 1010 (U0.8) 2. Align LSBs of both Operands: Operand 1 = 1010 1010.0000 0000 (U8.8) Operand 2 = 0000 0000.0110 1010 (U8.8) 3. Sign extend both Operands: Operand 1 = 0 1010 1010.0000 0000 (U9.8) Operand 2 = 0 0000 0000.0110 1010 (U9.8) 4. Add both Operands to get Result: Result = 0 1010 1010. 0110 1010 (U9.8) Result = 170.4140625
DEC	170	0.4140625	170.4140625

Table 22 Example 5 (U8.1 + U1.8 = U9.8)

	Op 1 (U8.1)	Op 2 (U1.8)	Result (U9.8)
BIN	1010 1010.1	1.0110 1010	1. Convert to binary: Operand 1 = 170.5. In binary: 1010 1010.1 (U8.1) Operand 2 = 1.4140625. In binary: 1.0110 1010 (U1.8) 2. Align LSBs of both Operands: Operand 1 = 1010 1010.1000 0000 (U8.8) Operand 2 = 0000 0001.0110 1010 (U8.8) 3. Sign extend both Operands: Operand 1 = 0 1010 1010.1000 0000 (U9.8) Operand 2 = 0 0000 0001.0110.1010 (U9.8) 4. Add both Operands to get Result: Result = 0 1010 1011. 1110.1010 (U9.8) Result = 171.9140625
DEC	170.5	1.4140625	171.9140625

Table 23 Example 6 (U8.4 + U-4.8 = U9.8)

	Op 1 (U8.4)	Op 2 (U-4.8)	Result (U9.8)
BIN	1010 1010.1010	.xxxx 0101	1. Convert to binary: Operand 1 = 170.625. In binary: 1010 1010.1010 (U8.4) Operand 2 = 0. 00390625. In binary: .xxxx 0101 (U-4.8) 2. Align LSBs of both Operands: Operand 1 = 1010 1010.1010 0000 (U8.8) Operand 2 = .xxxx 0101 (U-4.8) 3. Sign extend both Operands: Operand 1 = 0 1010 1010.1010 0000 (U9.8) Operand 2 = 0 0000 0000.0000 0101 (U9.8) 4. Add both Operands to get Result: Result = 0 1010 1010.1010 0101 (U9.8) Result = 170. 64453125
DEC	170.625	0.01953125	170.64453125

Table 24 Example 7 (U8.-4 + U-4.8 = U9.8)

	Op 1 (U8.-4)	Op2 (U-4.8)	Result (U9.8)
BIN	1010 xxxx.	.xxxx 0101	1. Convert to binary: Operand 1 = 80. In binary: 1010 xxxx. (U8.-4) Operand 2 = 0. 00390625. In binary: .xxxx 0101 (U-4.8) 2. Align LSBs of both Operands: Operand 1 = 1010 xxxx. (U8.-4) Operand 2 = .xxxx 0101 (U-4.8) 3. Sign extend both Operands: Operand 1 = 0 1010 0000.0000 0000 (U9.8) Operand 2 = 0 0000 0000.0000 0101 (U9.8) 4. Add both Operands to get Result: Result = 0 1010 0000.0000 0101 (U9.8) Result = 160.00390625
DEC	160	0.00390625	160.00390625

Table 25 Example 8 (S8.0 + 8.0 = S9.0)

	Op 1 (S8.0)	Op 2 (S8.0)	Result (S9.0)
BIN	1001 0110.	1100 1100.	1. Convert to binary: Operand 1 = -106. In binary: 1001 0110. (S8.0) Operand 2 = -52. In binary: 1100 1100. (S8.0) 2. Sign extend both Operands: Operand 1 = 1 1001 0110. (S9.0) Operand 2 = 1 1100 1100. (S9.0) 3. Add both Operands to get Result: Result = 1 0110 0010. (S9.0) Result = -158
DEC	-106	-52	-158

Table 26 **Example 9 (S8.0 + 8.0 = S9.0)**

	Op 1 (S8.0)	**Op 2 (S8.0)**	**Result (S9.0)**
BIN	0110 1010.	1100 1100.	1. Convert to binary: Operand 1 = 106. In binary: 0110 1010. (S8.0) Operand 2 = -52. In binary: 1100 1100. (S8.0) 2. Sign extend both Operands: Operand 1 = 0 0110 1010. (S9.0) Operand 2 = 1 1100 1100. (S9.0) 3. Add both Operands to get Result: Result = 0 0011 0110. (S9.0) Result = 54
DEC	106	-52	54

4.2 Substraction

Table 27 **Example 1 (U8.0 - U8.0 = U8.0)**

	Op 1 (U8.0)	Op 2 (U8.0)	Result (U8.0)
BIN	0110 1010.	0011 0100.	1. Convert to binary: Operand 1 = 106. In binary: 0110 1010. (U8.0) Operand 2 = 52. In binary: 0011 0100. (U8.0) 2. Sign extend both Operands: Operand 1 = 0 0110 1010. (U9.0) Operand 2 = 0 0011 0100. (U9.0) 3. Two’s complements of Operand 2: Operand 1 = 0 0110 1010. (U9.0) **Operand 2 = 1 1100 1100. (S9.0)** 4. Add both Operands to get Result: Result = 0 0011 0110. (U9.0) Result = 0 0011 0110. (S9.0) Result = 0011 0110. (U8.0) Result = 54
DEC	106	52	54

Table 28 **Example 2 (U8.0 – U8.0 = S9.0)**

	Op 1 (U8.0)	Op 2 (U8.0)	Result (S9.0)
BIN	0011 0100.	0110 1010.	1. Convert to binary: Operand 1 = 52. In binary: 0011 0100. (U8.0) Operand 2 = 106. In binary: 0110 1010. (U8.0) 2. Sign extend both Operands: Operand 1 = 0 0011 0100. (S9.0) Operand 2 = 0 0110 1010. (S9.0) 3. Two’s complements of Operand 2: Operand 1 = 0 0011 0100. (S9.0) **Operand 2 = 1 1001 0110. (S9.0)** 4. Add both Operands to get Result: Result = 1 1100 1010. (S9.0) Result = -54
DEC	52	106	-54

Table 29 **Example 3 (S8.0 - S8.0 = S9.0)**

	Op 1 (S8.0)	**Op 2 (S8.0)**	**Result (S9.0)**
BIN	0110 1010.	0011 0100.	1. Convert to binary: Operand 1 = 106. In binary: 0110 1010. (S8.0) Operand 2 = 52. In binary: 0011 0100. (S8.0) 2. Sign extend both Operands: Operand 1 = 0 0110 1010. (S9.0) Operand 2 = 0 0011 0110. (S9.0) 3. Two's complements of Operand 2: Operand 1 = 0 0110 1010. (S9.0) **Operand 2 = 1 1100 1100. (S9.0)** 4. Add both Operands to get Result: Result = 0 0011 0110. (S9.0) Result = 54
DEC	106	52	54

Table 30 **Example 4 (S8.0 - S8.0 = S9.0)**

	Op 1 (S8.0)	**Op 2 (S8.0)**	**Result (S9.0)**
BIN	0011 0100.	0110 1010.	1. Convert to binary: Operand 1 = 52. In binary: 0011 0100. (S8.0) Operand 2 = 106. In binary: 0110 1010. (S8.0) 2. Sign extend both Operands: Operand 1 = 0 0011 0100. (S9.0) Operand 2 = 0 0110 1010. (S9.0) 3. Two's complements of Operand 2: Operand 1 = 0 0011 0100. (S9.0) **Operand 2 = 1 1001 0110. (S9.0)** 4. Add both Operands to get Result: Result: 1 1100 1010. (S9.0) Result = -54
DEC	52	106	-54

Table 31 Example 5 (S8.0 – S4.4 = S9.4)

	Op 1 (S8.0)	**Op 2 (S4.4)**	**Result (S9.4)**
BIN	0011 0100.	0110.1010	1. Convert to binary: Operand 1 = 52. In binary: 0011 0100. (S8.0) Operand 2 = 6.625. In binary: 0110.1010 (S4.4) 2. Align LSBs of operands: Operand 1 = 0011 0100.0000 (S8.4) Operand 2 = 0110.1010 (S4.4) 3. Sign extend both Operands: Operand 1 = 0 0011 0100.0000 (S9.4) Operand 2 = 0 0000 0110.1010 (S9.4) 4. Two’s complements of Operand 2: Operand 1 = 0 0011 0100.0000 (S9.4) **Operand 2 = 1 1111 1001.0110 (S9.4)** 5. Add both Operands to get Result: Result = 0 0010 1101.0110 (S9.4) Result = 45.375
DEC	52	6.625	45.375

Table 32 Example 6 (S4.4 - S8.0 = S9.4)

	Operand 1 (S4.4)	Operand 2 (S8.0)	Result (S9.4)
BIN	0110.1010	0011 0100.0	1. Convert to binary: Operand 1 = 6.625. In binary: 0110.1010 (S4.4) Operand 2 = 52. In binary: 0011 0100. (S8.0) 2. Align LSBs of operands: Operand 1 = 0110.1010 (S4.4) Operand 2 = 0011 0100.0000 (S8.4) 3. Sign extend both Operands: Operand 1 = 0 0000 0110.1010 (S9.4) Operand 2 = 0 0011 0100.0000 (S9.4) 4. Two's complements of Operand 2: Operand 1 = 0 0000 0110.1010 (S9.4) Operand 2 = 1 1100 1100.0000 (S9.4) 5. Add both Operands to get Result: Result = 1 1101 0010.1010 (S9.4) Result = -45.375
DEC	6.625	52	-45.375

4.3 Multiplication & Division

This document will not discuss binary multiplication and division as they are not the core focus of this document. In general:

Table 33 Multiplication and Division

Op 1	Op 2	Operation	Result
Sm1.n1	Sm2.n2	Op1 * Op2	S(e + m1 + m2).(n1 + n2)
		Op1 / Op2	S(e + m1 – m2).(n1 – n2)

5 Exponent-Mantissa Format

Rational Numbers can also be represented in Exponent-Mantissa format, for example:

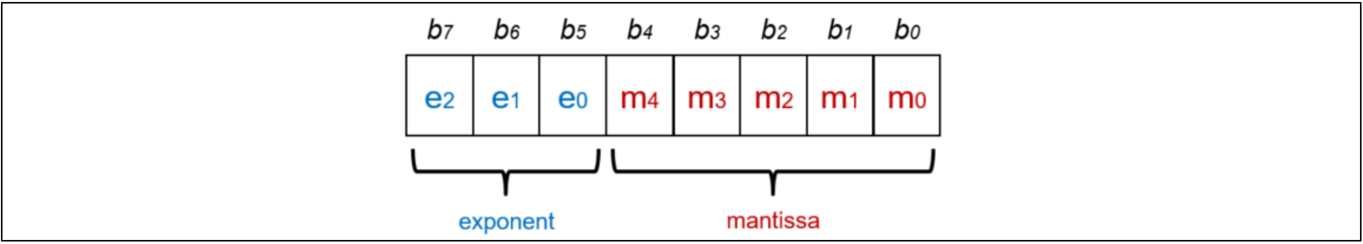

b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0
e_2 e_1 e_0 m_4 m_3 m_2 m_1 m_0
exponent
mantissa

Figure 6 **Exponent-Mantissa Representation**

The formulae can be summarized as following. Let:

e = Exponent bit-length

m = Mantissa bit-length

w = Weighting factor

For a binary-represented Rational Numbers R denoted by eEmM:

Table 34 **Exponent-Mantissa Format Formulae**

Property	Formula	Representation
Bit Length	e + m	R[(e + m - 1) : 0]
Exponent		R[(e + m - 1) : m]
Mantissa	2^m + R[(m - 1) : 0]	R[(m - 1) : 0]
Rational Numbers	Mantissa * 2^{Exponent} * 2^w	R[(e + m + 1) : 0]

6 Pre-Assigned Binary Points Registers

Some registers that are directly interfaced to the hardware may already have pre-assigned binary point and LSB weighting. In general:

1. *Voltages referenced to the VS ADC's (e.g., internal vout, vrect and vcontrol) have a LSB at 1.25mV. This was chosen to match the VS ADC LSB weight.*
2. *Other voltages generally place the binary point at 1V to match PMBus.*
3. *Currents generally place the binary point at 1A to match PMBus.*
4. *Resistances (e.g., for Droop/loadline) generally place the binary point at 1mOhm to match PMBus.*
5. *Temperatures generally place the binary point at 1C to match PMBus.*
6. *Powers generally place the binary point at 1W to match PMBus.*
7. *Time parameters generally place the binary point based on the HW clock period associated with the parameter. Some variations in binary points are possible such as 5ns, 10ns, 20ns, etc. For some longer time parameters, binary points at 1ms is possible in order to match PMBus.*

7 PMBus Linear Data Format

PMBus Linear Data Format is represented as following:

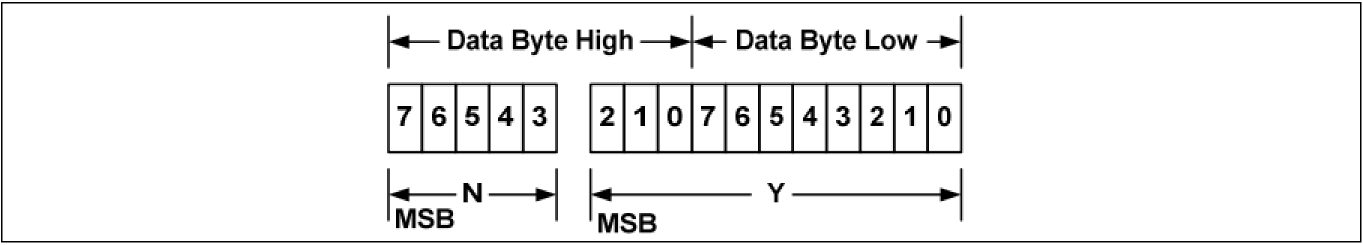


Figure 7 PMBus Linear Format Representation

The relation between Y, N and the “real world” value X is:

$$X = Y \cdot 2^N$$

Where X is “read world” value,

Y is an 11-bit, two’s complement integer (also called Mantissa)

N is a 5-bit, two’s complement integer (also called Exponent).

Some worked examples below:

Table 35 PMBus Linear Data Format Worked Examples

Real World value (X)	Exponent (N)		Mantissa (Y)		16-bit Representation
	DEC	BIN	DEC	BIN	
0.5	-1	1 1111	1	000 0000 0001	0xF801
0.5	-9	1 0111	256	001 0000 0000	0xB900
0.5625	-4	1 1100	9	000 0000 1001	0xE009
0.5625	-5	1 1011	18	000 0001 0010	0xD812
-45.375	-3	1 1110	-363	110 1001 0101	0xEE95
-45.375	-4	1 1100	-726	101 0010 1010	0xE52A

As can be seen from the above table, one particular real world value can be represented with different combinations of Linear Format representations. The choice is up to the developer. Typically, the choice depends on the Exponent selection followed by Mantissa calculation.

8 More Resources

More programming resources can be accessed easily from GUI main interface:

8.1 Firmware Documentation

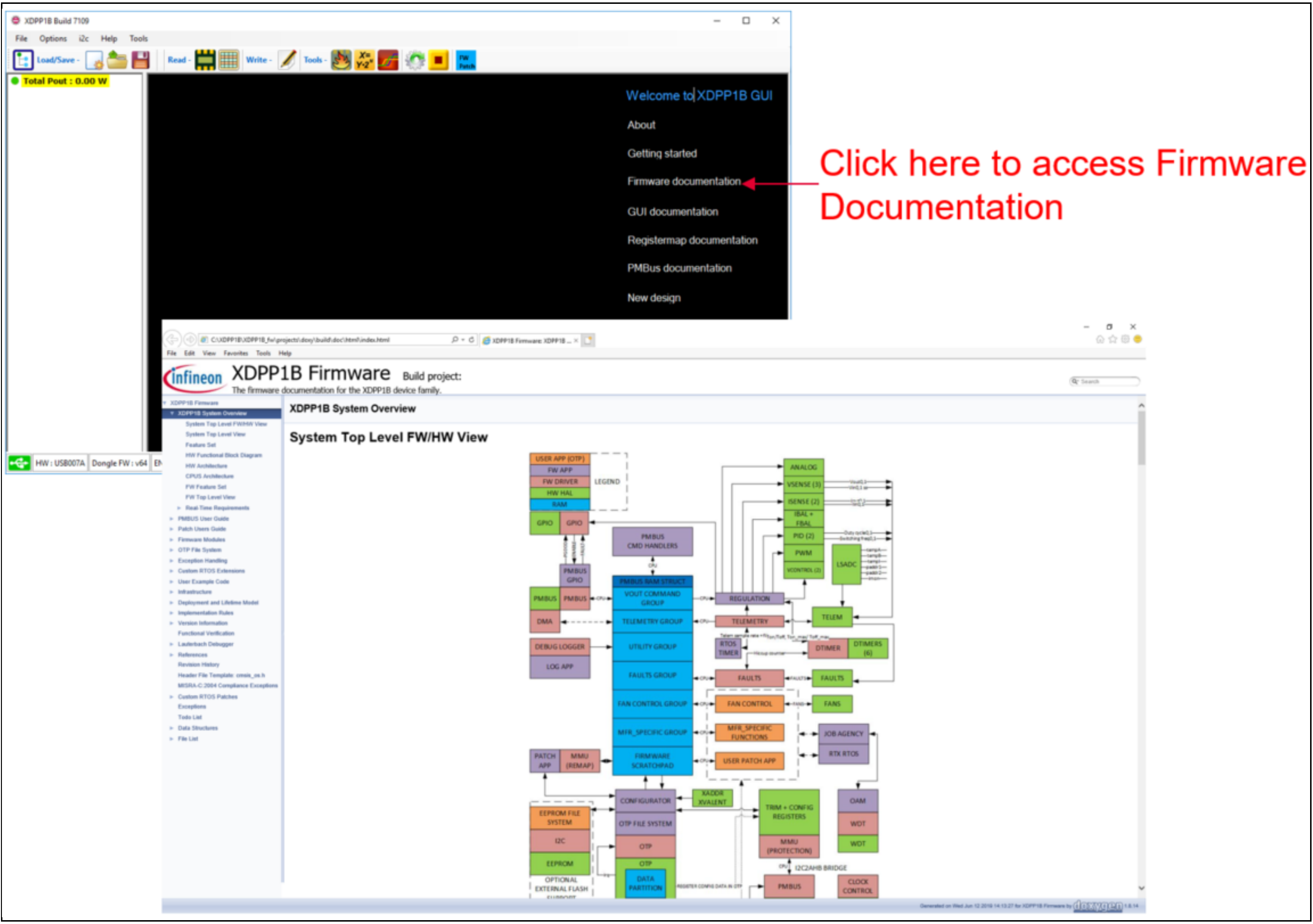
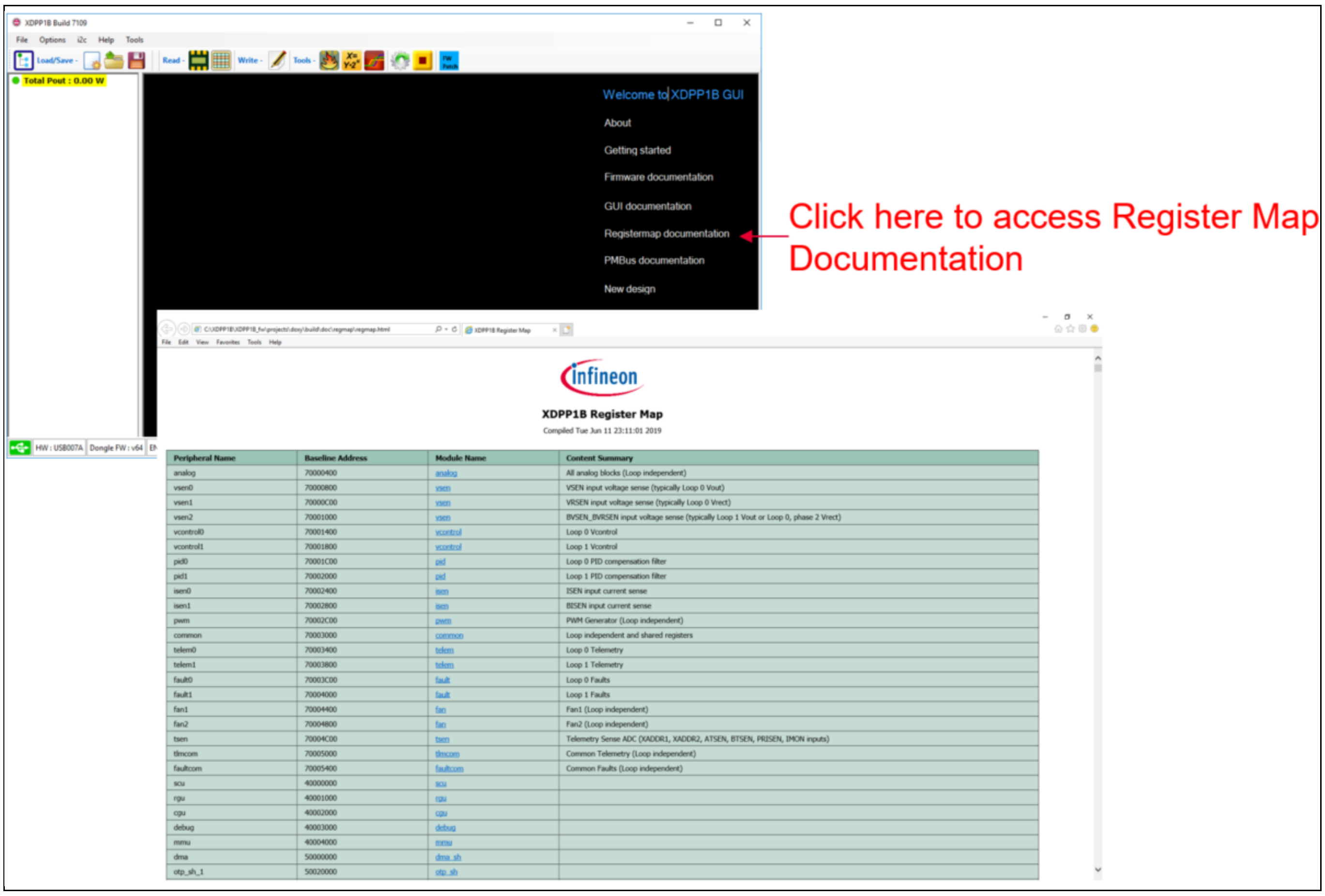


Figure 8 Firmware Documentation

8.2 Register Map Documentation



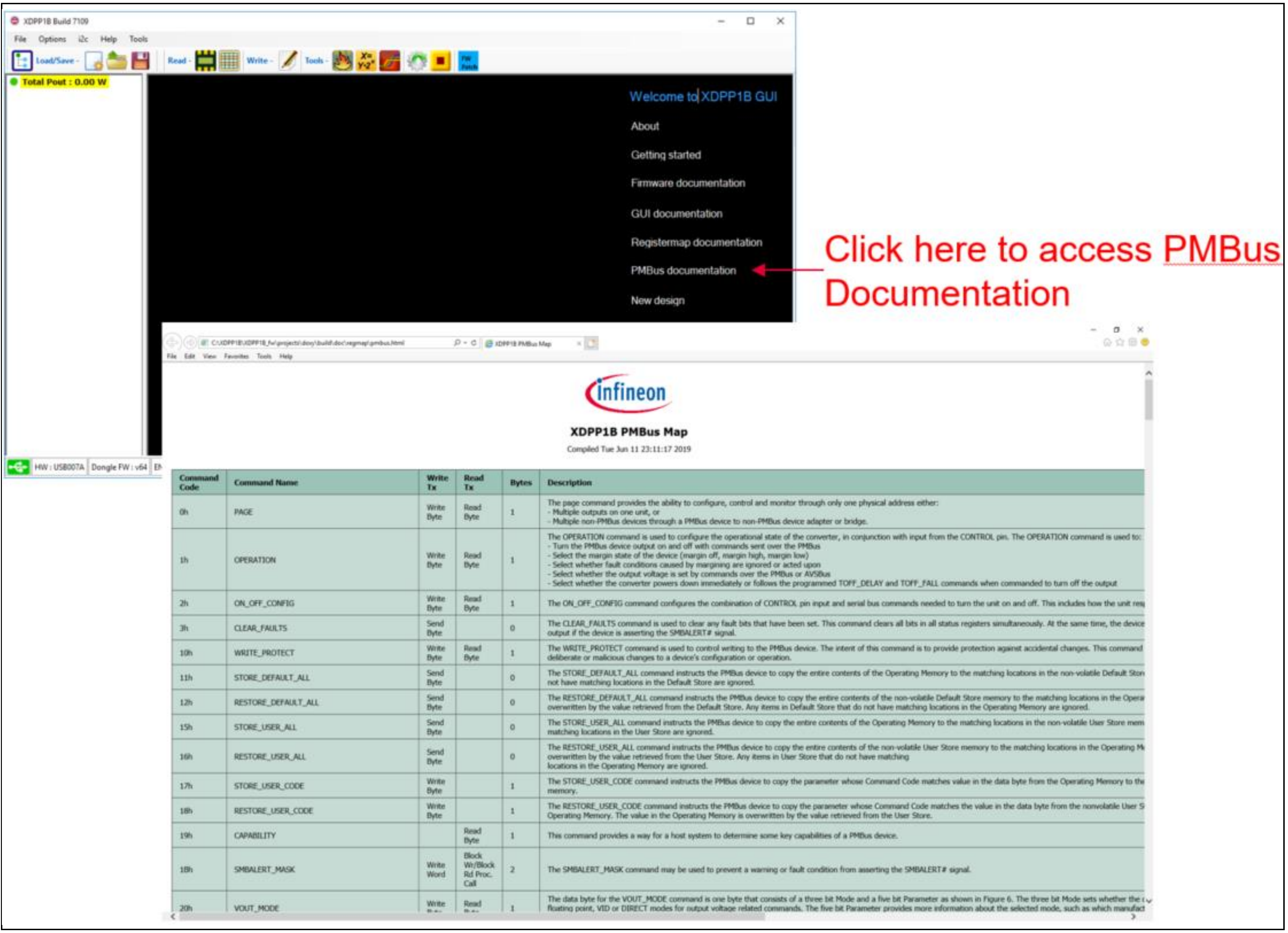
XDPP1B Register Map

Compiled Tue Jun 11 23:11:01 2019

Peripheral Name	Baseline Address	Module Name	Content Summary
analog	70000400	analog	All analog blocks (Loop independent)
vsen0	70000800	vsen	VSEN input voltage sense (typically Loop 0 Vout)
vsen1	70000C00	vsen	VRSEN input voltage sense (typically Loop 0 Vrect)
vsen2	70001000	vsen	BVSEN_BVRSEN input voltage sense (typically Loop 1 Vout or Loop 0, phase 2 Vrect)
vcontrol0	70001400	vcontrol	Loop 0 Vcontrol
vcontrol1	70001800	vcontrol	Loop 1 Vcontrol
pid0	70001C00	pid	Loop 0 PID compensation filter
pid1	70002000	pid	Loop 1 PID compensation filter
isen0	70002400	isen	ISEN input current sense
isen1	70002800	isen	BISEN input current sense
pwm	70002C00	pwm	PWM Generator (Loop independent)
common	70003000	common	Loop independent and shared registers
telem0	70003400	telem	Loop 0 Telemetry
telem1	70003800	telem	Loop 1 Telemetry
fault0	70003C00	fault	Loop 0 Faults
fault1	70004000	fault	Loop 1 Faults
fan1	70004400	fan	Fan1 (Loop independent)
fan2	70004800	fan	Fan2 (Loop independent)
tsen	70004C00	tsen	Telemetry Sense ADC (XADDR1, XADDR2, ATSEN, BTSEN, PRISEN, IMON inputs)
tlmcom	70005000	tlmcom	Common Telemetry (Loop independent)
faultcom	70005400	faultcom	Common Faults (Loop independent)
scu	40000000	scu	
rgu	40001000	rgu	
cgu	40002000	cgu	
debug	40003000	debug	
mmu	40004000	mmu	
dma	50000000	dma_sh	
otp_sh_1	50020000	otp_sh	

Figure 9 Register Map Documentation

8.3 PMBus Documentation


XDPP1B Build 7109
File Options i2c Help Tools
Load/Save Read Write Tools
Total Pout : 0.00 W
Welcome to XDPP1B GUI
About
Getting started
Firmware documentation
GUI documentation
Registermap documentation
PMBus documentation
New design
Click here to access PMBus Documentation
HW : USB007A Dongle FW : v64

XDPP1B PMBus Map

Compiled Tue Jun 11 23:11:17 2019

Command Code	Command Name	Write Tx	Read Tx	Bytes	Description
0h	PAGE	Write Byte	Read Byte	1	The page command provides the ability to configure, control and monitor through only one physical address either: - Multiple outputs on one unit, or - Multiple non-PMBus devices through a PMBus device to non-PMBus device adapter or bridge.
1h	OPERATION	Write Byte	Read Byte	1	The OPERATION command is used to configure the operational state of the converter, in conjunction with input from the CONTROL pin. The OPERATION command is used to: - Turn the PMBus device output on and off with commands sent over the PMBus - Select the margin state of the device (margin off, margin high, margin low) - Select whether fault conditions caused by margining are ignored or acted upon - Select whether the output voltage is set by commands over the PMBus or AVSBus - Select whether the converter powers down immediately or follows the programmed TOFF_DELAY and TOFF_FALL commands when commanded to turn off the output
2h	ON_OFF_CONFIG	Write Byte	Read Byte	1	The ON_OFF_CONFIG command configures the combination of CONTROL pin input and serial bus commands needed to turn the unit on and off. This includes how the unit res
3h	CLEAR_FAULTS	Send Byte		0	The CLEAR_FAULTS command is used to clear any fault bits that have been set. This command clears all bits in all status registers simultaneously. At the same time, the device output if the device is asserting the SMBALERT# signal.
10h	WRITE_PROTECT	Write Byte	Read Byte	1	The WRITE_PROTECT command is used to control writing to the PMBus device. The intent of this command is to provide protection against accidental changes. This command deliberate or malicious changes to a device's configuration or operation.
11h	STORE_DEFAULT_ALL	Send Byte		0	The STORE_DEFAULT_ALL command instructs the PMBus device to copy the entire contents of the Operating Memory to the matching locations in the non-volatile Default Stor not have matching locations in the Default Store are ignored.
12h	RESTORE_DEFAULT_ALL	Send Byte		0	The RESTORE_DEFAULT_ALL command instructs the PMBus device to copy the entire contents of the non-volatile Default Store memory to the matching locations in the Opera overwritten by the value retrieved from the Default Store. Any items in Default Store that do not have matching locations in the Operating Memory are ignored.
15h	STORE_USER_ALL	Send Byte		0	The STORE_USER_ALL command instructs the PMBus device to copy the entire contents of the Operating Memory to the matching locations in the non-volatile User Store mem matching locations in the User Store are ignored.
16h	RESTORE_USER_ALL	Send Byte		0	The RESTORE_USER_ALL command instructs the PMBus device to copy the entire contents of the non-volatile User Store memory to the matching locations in the Operating M overwritten by the value retrieved from the User Store. Any items in User Store that do not have matching locations in the Operating Memory are ignored.
17h	STORE_USER_CODE	Write Byte		1	The STORE_USER_CODE command instructs the PMBus device to copy the parameter whose Command Code matches value in the data byte from the Operating Memory to the memory.
18h	RESTORE_USER_CODE	Write Byte		1	The RESTORE_USER_CODE command instructs the PMBus device to copy the parameter whose Command Code matches the value in the data byte from the nonvolatile User S Operating Memory. The value in the Operating Memory is overwritten by the value retrieved from the User Store.
19h	CAPABILITY		Read Byte	1	This command provides a way for a host system to determine some key capabilities of a PMBus device.
1Bh	SMBALERT_MASK	Write Word	Block Wr/Block Rd Proc. Call	2	The SMBALERT_MASK command may be used to prevent a warning or fault condition from asserting the SMBALERT# signal.
20h	VOUT_MODE	Write	Read	1	The data byte for the VOUT_MODE command is one byte that consists of a three bit Mode and a five bit Parameter as shown in Figure 6. The three bit Mode sets whether the floating point, VID or DIRECT modes for output voltage related commands. The five bit Parameter provides more information about the selected mode, such as which manufact

Figure 10 PMBus Documentation

9 References

http://www.pmbus.org/Home

http://www.superkits.net/whitepapers/Fixed%20Point%20Representation%20&%20Fractional%20Math.pdf

https://www2.keil.com/mdk5/cmsis/rtx

http://eguruchela.com/math/calculator/twos-complement

Revision history

Document version	Date of release	Description of changes

Edition 2020-01-13

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
XDPP1100 FW DEV