XDPP1100 number format and conversion

Getting started in XDPP1100 custom firmware development

About this document

Scope and purpose

This document explains the number format used in XDPP1100 firmware, showing the fixed-point numbers used in registers as well as their conversion to PMBus LINEAR11 format.

Intended audience

This application note is intented for firmware engineers and 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 Subtraction 16
4.3 Multiplication and division 20
5 Exponent-Mantissa format **21**
6 Pre-assigned binary points registers **22**
7 PMBus linear data format **23**
8 Number conversion **24**
8.1 LINEAR11 to Q-number 24
8.2 Q-number to LINEAR11 24
9 References **25**
Revision history **26**

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 applications. 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 six channels of 9-bit, 1 Msps general-purpose ADCs, timers, interrupt control, PMBus and two I^2C communications ports.

The device is based on a 32-bit, 100 MHz 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 the XDPP1100 comes with vast list of features such as support for multiple power topologies commonly found in DC-DC server/telecom brick power conversion, built-in PMBus commands, voltage mode control (VMC), peak current mode control (PCMC) and many others.

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

An ARM® Cortex™-M0 is used as the CPU core of the XDPP1100; it has no floating-point unit (FPU) and only supports fixed-point arithmetics.

Critical peripherals in the XDPP1100 such as ADC, PWM, PID controller and feed-forward (FF) 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 hardware blocks will take care of all the calculations.

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

In addition, PMBus comes with its own formats, which are described briefly in this section. For more details, the user is encouraged to refer to the PMBus standards directly in [1].

This section discusses the fixed-point arithmetics and number representation used in the XDPP1100, and how this interacts with the 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. They can be represented visually in the following diagram:

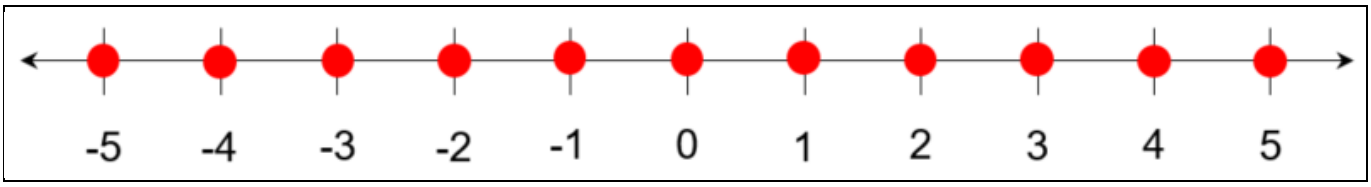

Figure 1 Integers visualization

Positive integers, including zero, are also known as **whole numbers**.

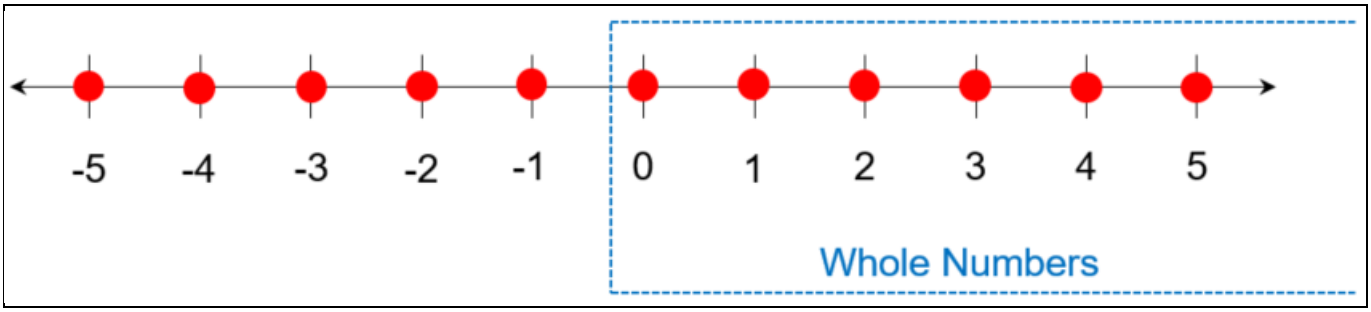

Figure 2 Whole numbers visualization

Some physical quantities quantified with integers/whole numbers include the number of steps taken per day, or the number of people in a room.

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

- Number of ADC samples needed to trigger a fault
- Amount 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 include integers as well as fractions/decimal points. They exclude special numbers such as π, *e* and roots (√). They can be visually represented in the following diagram:

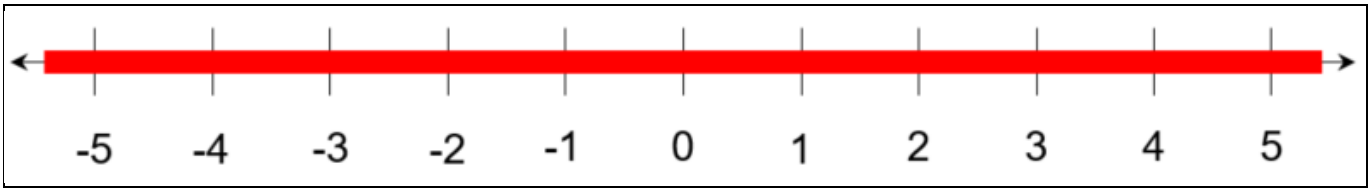

Figure 3 Rational numbers visualization

Most physical quantities related to electricity and power such as voltage (V), current (A), power (W) and temperature (T) can be represented by rational numbers.

Time (s, ns) is usually represented by positive rational numbers.

3 Binary representation

Representing the above-mentioned physical quantities on a computer has its own challenges, because computers only recognize and operate with **binary numbers** in the form of 1 and 0.

This section discusses how binary numbers can be manipulated to represent real-world physical quantities.

3.1 Unsigned integers

When binary numbers (BIN) are grouped together with a certain length (a.k.a. N-bit length), it can be used to represent whole numbers (DEC). In addition, binary numbers can be converted into hexadecimal numbers (HEX) to remove the need to write long numbers.

Table 1 BIN2DEC: binary representation of whole numbers 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 their 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	**13**
	0100	**4**
8-bit	1000 1000	**136**
	0111 1010	**122**

It is also interesting to note that binary numbers have **discrete** properties, the same as real-world physical integers. This can be observed when the right-most number, i.e. the **least significant bit (LSB)**, of the binary number is incremented/decremented by 1.

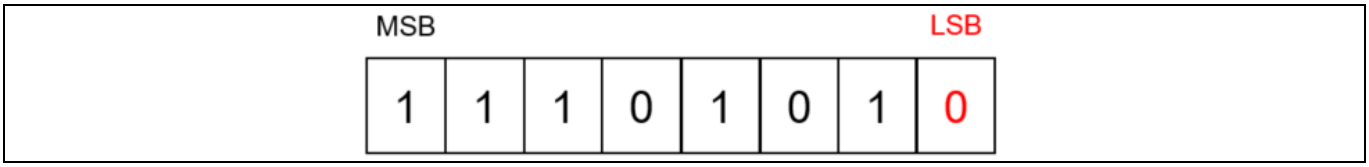

Figure 4 LSB illustration

Table 4 BIN numbers discrete properties

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 *grouped binary numbers/hexadecimal numbers*. In computer science, this representation is called an "**unsigned integer**".

3.1.1 Summary

Mathematically, unsigned integers can be summarized as follows:

Table 5 Unsigned integers formulas

Property	Formula	
Bit length	N	
Min. value (DEC)	0	*This is also known as **Un.0** format – discussed in the next chapter.*
Max. value (DEC)	$2^N - 1$	

Unfortunately, this is still not enough to fully represent the negative range of real-world physical integer numbers. For this reason, "**signed integer**" representation is introduced.

3.2 Signed integers (two's complement)

A sign-bit can be embedded into the binary numbers to indicate a positive/negative integer and complete the representation of real-world physical integer numbers. 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.

A positive number has "0" as the sign-bit, whereas a negative number has "1" as the sign-bit.

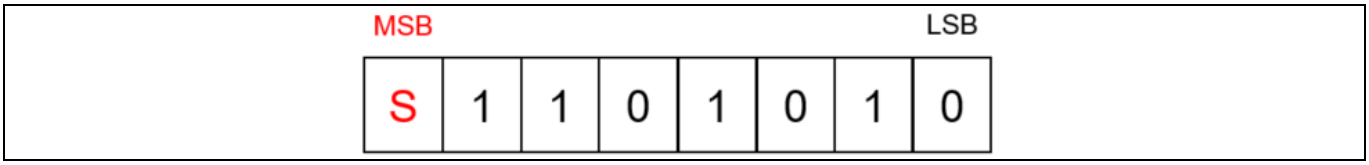

Figure 5 MSB illustration with sign-bit

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 the 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 using two'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 quick number conversion tricks.

DEC2BIN quick example: Convert -6 to BIN.

1. Convert the absolute value of a 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 the 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 the LSB: *0101 + 1 = 0110*
3. Convert the binary number to a 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 follows:

Table 9 Signed integers formulas

Property	**Formula**	
Bit length	N	
Min. value (DEC)	$-(2^{(N-1)})$	*This is also known as **Sn.0** format – discussed in the next chapter.*
Max. value (DEC)	$2^{(N-1)}-1$	

We are now able to fully represent real-world physical integer numbers. 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 representation of fractions and decimal points by signed integers, 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/fractions can be done by setting the value of "n". Setting **n** to **3** on the above U8.0 and S8.0 examples yields 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 corresponds to an increased value of 1 in DEC value. In U8.3 and S8.3, however, every increment of LSB corresponds to a different value. This property is called "LSB weight", which can be calculated as:

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

A 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 still conforms 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 follows:

Table 17 Qm.n formulas

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 = 0 1101 0011. (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 (xxxx refers to “don’t care”): 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 (assume xxxx is filled with 0000): 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 (xxxx refers to “don’t care”): 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 (assume xxxx is filled with 0000): 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	0.01953125	160.01953125

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 Subtraction

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 and division

This document will not discuss binary multiplication and division.

5 Exponent-Mantissa format

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

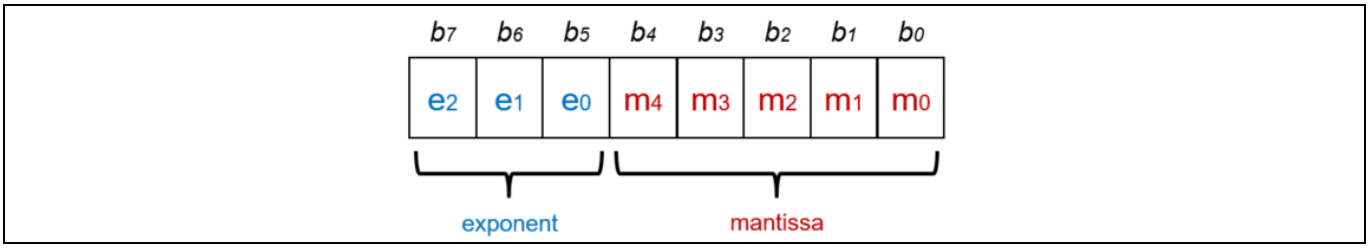

Figure 6 Exponent-Mantissa representation

The formulas can be summarized as follows. Let:

e = Exponent bit-length

m = Mantissa bit-length

w = weighting factor

For binary-represented rational numbers R denoted by eEmM:

Table 33 Exponent-Mantissa format formulas

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 block may already have pre-assigned binary points and LSB weighting. In general:

1. *Voltages referenced to the VS ADCs (e.g., internal V_{out}, V_{rect} and $V_{control}$) have a LSB at 1.25 mV. This was chosen to match the VS ADC LSB weight.*
2. *Other voltages generally place the binary point at 1 V to match PMBus.*
3. *Currents generally place the binary point at 1 A to match PMBus.*
4. *Resistances (e.g., for droop/load-line) generally place the binary point at 1 mΩ to match PMBus.*
5. *Temperatures generally place the binary point at 1°C to match PMBus.*
6. *Powers generally place the binary point at 1 W 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 5 ns, 10 ns, 20 ns, etc. For some longer time parameters, binary points at 1 ms are possible in order to match PMBus.*

7 PMBus linear data format

PMBus linear data format is represented as follows:

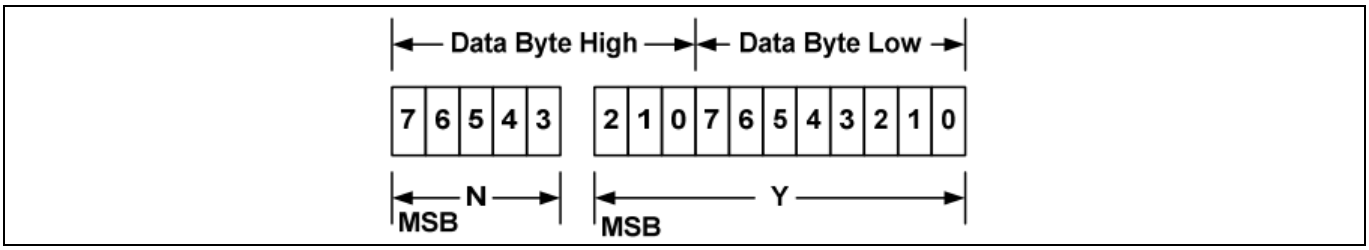

Figure 7 PMBus linear format representation

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

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

Where X is the real-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)

There are some worked examples below:

Table 34 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 the Mantissa calculation.

8 Number conversion

This section showcases some examples of how numbers can be converted easily from LINEAR11 to Q-number and vice-versa using XDPP1100 firmware drivers.

8.1 LINEAR11 to Q-number

```
// This is equivalent to 36.0 in decimal.
uint16_t LIN11_num = 0xE920;

// Get the Mantissa
int32_t qNum_man = LINEAR11_TO_MANTISSA(LIN11_num);
// Get the Exponent
int32_t qNum_exp = LINEAR11_TO_EXPONENT(LIN11_num);

// Shift mantissa based on exponent and get original number
int32_t qNum = SHIFT_EXPONENT(qNum_man, qNum_exp);

printf("%d", qNum); // print out 36
```

Figure 8 LINEAR11 to Q-number

8.2 Q-number to LINEAR11

```
// qNum is the Q number to be converted.
int32_t qNum = 36;
int8_t exponent = -3;

// Calculate LINEAR11 exponent
uint8_t LIN11_expo = TWOS_COMPLEMENT(5, exponent);
// Calculate LINEAR11 mantissa
int16_t LIN11_mant = qNum << -LIN11_expo;

// Shift mantissa based on exponent and get original number
uint16_t LIN11_num = LIN11_expo << 11 | LIN11_mant & 0x7FF;

printf("%x", LIN11_num); // print out 0xE920
```

Figure 9 Q-number to LINEAR11

9 References

[1] http://www.pmbus.org/Home

Revision history

Document version	Date of release	Description of changes
V 1.0	09-06-2020	First release

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

Edition 2020-06-09

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
AN_2005_PL88_2006_073720

IMPORTANT NOTICE

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

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

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

WARNINGS

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

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