

LTC2978 Firmware Programmer's Getting Started Guide – Reading Input Voltage (VIN) Telemetry

Revision 0.1

Author: Mike Holloway, Linear Technology

Date: 1/25/2010, 1:40 PM

Revision History:

0.2	Terminology Change: changed 'command byte' to 'command code'
0.1	Initial Draft

Purpose

The LTC2978 provides a great deal of power and configurability for custom applications. The part additionally provides onboard non-volatile memory (NVM) or EEPROM to store and recall configuration parameters.

This architecture allows the chip to power up and load the desired customer configuration **autonomously with no I2C/firmware interaction required.**

However, there are some circumstances where a customer may wish to author firmware to communicate with the part. This document provides a quick overview of one common firmware task – How to read the VIN input voltage using the READ_VIN telemetry command.

LTpowerPlay™ is a powerful tool that can be used to understand how to author the firmware to talk to the LTC2978. You can download LTpowerPlay™ from <http://ltpowerplay.com>.

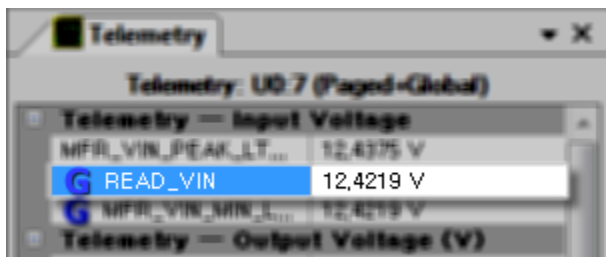
Reading the Input Voltage

The input voltage as measured by the ADC can be read over the PMBus using the global READ_VIN 16-bit 'word' register. This register uses the standard PMBus "Linear" format for non-output voltage related registers. We call this format LinearFloat5_11, and it represents a floating point value as a 5-bit signed exponent (upper 5 bits) and a signed 11-bit mantissa (lower 11 bits).

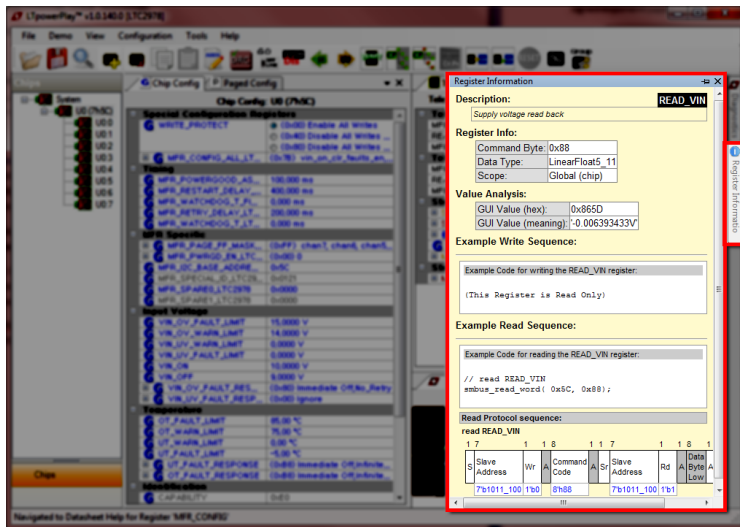
A Note on 'Linear' Format: The term 'Linear format' as defined in the PMBus specification actually refers to two distinctly different numeric formats. All output voltage related settings (like READ_VOUT, VOUT_COMMAND, etc) use what we call LinearFloat16 format. This is a 16-bit value that represents an unsigned mantissa which must be multiplied by a separate exponent (2^{-13} for the LTC2978) to determine the value. The LinearFloat16 format is **not** discussed in this document.

All **non output-voltage-related** registers using 'Linear' format use what we call LinearFloat5_11 format. This is a distinctly different numeric representation. In this case, the 16-bit value represents a self-contained floating point number, with both a 5-bit signed (two's complement) exponent (upper 5 bits) and a signed 11-bit mantissa (lower 11 bits), as discussed below.

To understand the protocol/firmware required to read this register, we will utilize the "Register Information" tab in the LTpowerPlay™ GUI. To view this tab connect a demo board and launch LTpowerPlay™. Then select the READ_VIN register in the Telemetry pane.



Now go over to the right and activate the 'Register Information' tab.



This tab shows you more detailed information about the selected register which can be a great help in understanding how to author the firmware. Below is an example of the Register information for the READ_VIN register:

Example Read Sequence:

```

Example Code for reading the READ_VIN register:

// read READ_VIN
smbus_read_word( 0x5C, 0x88 );
    
```

Read Protocol sequence:

read READ_VIN

1 7	1	1 8	1 1 7	1	1 8	1 8	1 1		
S	Wr	A	A	Sr	Rd	A	A	N	P
Slave Address		Command Code		Slave Address		Data Byte Low	Data Byte High		
7'b1011_100	1'b0	8'h88		7'b1011_100	1'b1				

Note that the information in this tab is 'context sensitive'. In other words, it shows you detailed information based on what is selected in the GUI. In this particular case it shows the steps required to read the global READ_VIN register for the chip at 7-bit I2C address 0x5C.

Because the READ_VIN is a global register, writing the PAGE register is not required, and the examples reflect this. The view shows example C code as well as detailed SMBus protocol diagrams. The protocol

diagrams follow the conventions established in the SMBus specification. The protocol diagram legend or element key is shown in the figure below:

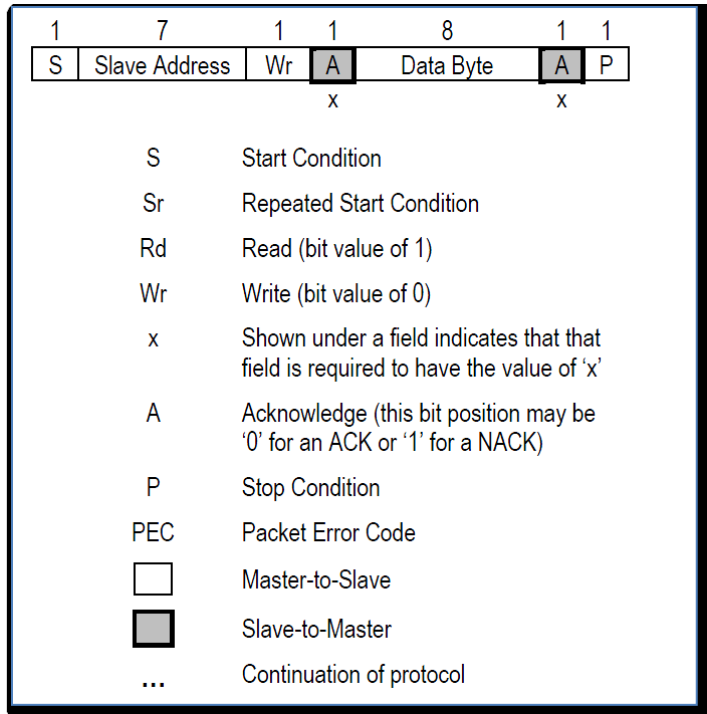


Figure: SMBus packet protocol diagram element key

The SMBus read word protocol starts out like a normal I2C write transaction. That is the address+Wr is sent first, **not address+Rd**. Then the command code follows. A repeated start follows, and **then the address+Rd** is sent to signal that the slave should return data for the specified command code. The slave responds by transmitting the low byte first, and the host acknowledges this byte. Then the slave returns the second byte (high byte) of the data, and the host NAKs (not acknowledges) this byte. The LTpowerPlay Register Information tab above reveals all of this protocol detail in a compact visual protocol diagram.

NOTE: A very common mistake made by firmware engineers familiar with I2C but unfamiliar with SMBus is attempting to read PMBus registers by immediately sending the address+Rd after the first start condition. **This is incorrect.**

The most common error made by I2C-experienced firmware engineers shown here:

17	1
S Slave Address	Rd
7'b1011_100	1'b1

Though the above protocol would be a valid I2C read, it is **NOT a valid SMBus Read**. SMBus/PMBus read byte/word transactions **always start with the address+Wr** as shown in the LTpowerPlay™ protocol diagrams above, **not** with address+Rd.

Our specific task is to read the READ_VIN register for the device at 7-bit I2C address 0x5C. The required I2C primitives in the order they would occur on the bus are as follows:

I2C protocol required to read the READ_VIN register (0x88)

- <Start Condition>
- Host transmits byte 0xB8 (0x5C<<1 with Write bit set to 0)
- Host transmits byte 0x88 (command code for READ_VIN register)
- <Repeated Start Condition>
- Host transmits byte 0xB9 (0x5C<<1 with Read bit set to 1)
- Slave Responds with low data byte for READ_VIN, Host Acknowledges this byte
- Slave Responds with high data byte for READ_VIN, Host NAKs this byte
- <Stop Condition>

Now let's say that the host executes the above protocol and the device replies with the 16-bit hex value 0xD317. Remember that the slave will return the lower byte (0x17) first on the wire and then the upper byte (0xD3).

How do we translate the hex value 0xD317 into an actual voltage? The READ_VIN register, as shown by the Register Information tab is formatted as a LinearFloat5_11 value. This standard PMBus 'Linear' format is used to represent all **non-output voltage** related telemetry registers, like READ_VIN, READ_TEMPERATURE*, READ_IIN, etc.

The LinearFloat5_11 formatted value is a 16-bit value containing both a 5-bit signed (two's complement) exponent (in the upper 5 bits) and a signed 11-bit mantissa (in the lower 11 bits). This format allows representation of negative values as well as covering a very large range of values.

Let's break the hex value that the slave returned (0xD317) into individual bits and look at the exponent and mantissa fields:

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Hex Nibbles	D				3				1				7			
Binary Value	1	1	0	1	0	0	1	1	0	0	0	1	0	1	1	1
Field	Exponent					Mantissa										
Decimal Value	-6					+791										

So the LinearFloat5_11 formatted 16-bit word 0xD317 represents the numeric value $791 * 2^{-6} \approx 12.359V$ (units for the READ_VIN register are volts).

More Help on 2's Complement Representation

Those unfamiliar with two's complement representation may be confused as to how we derived the decimal value -6 from the 5-bit binary value 11010.

The process is quite simple. In two's complement numeric representation, the most significant bit is called the sign bit. When it is a 1 (as in this case of our exponent), this indicates that the numeric quantity represented is negative.

We can more easily see the magnitude of a negative number by negating it. Negating a two's complement value is simple. To negate a value, simply invert all the bits and then add 1. Negation works in either direction -- going from negative to positive or vice versa.

In our case, we start with a negative value 11010. To negate this we invert all the bits to get 00101, and then add 1 to get 00110. This is equivalent to decimal +6.

So the 5-bit signed (two's complement) representation of decimal -6 is 11010. Note that we can use the same method to negate +6 (binary 00110), and we get binary 11010, confirming that **the two's complement representation for -6 is binary 11010**.

Example C Code

The following code listing shows an example C routine that converts a 16-bit LinearFloat5_11 formatted word into a floating point value.

Example Code Listing:¹ Converting a LinearFloat5_11 Formatted Word into a Floating Point Value

```
// convert a LinearFloat5_11 formatted word into a floating point value
float lin5_11ToFloat(uint16_t wordValue)
{
    int8_t exponent = wordValue>>11;           // extract exponent as MS 5 bits
    int16_t mantissa = wordValue & 0x7ff;      // extract mantissa as LS 11 bits

    if( exponent > 0x0F ) exponent |= 0xE0;    // sign extend exponent
    if( mantissa > 0x03FF ) mantissa |= 0xF800; // sign extend mantissa

    return mantissa * pow(2,exponent);        // compute value as
                                              // mantissa * 2^(exponent)
}
```

¹ Linear Technology provides this Example Code “as is” without warranty of any kind.