



# PMBus™ Power System Management Protocol Specification Part III – AVSBus

Revision 1.3.1  
13 March 2015

[www.powerSIG.org](http://www.powerSIG.org)

© 2015 System Management Interface Forum, Inc. – All Rights Reserved

### **DISCLAIMER**

This specification is provided “as is” with no warranties whatsoever, whether express, implied or statutory, including but not limited to any warranty of merchantability, non-infringement, or fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification or sample.

In no event will any specification co-owner be liable to any other party for any loss of profits, loss of use, incidental, consequential, indirect, or special damages arising out of this specification, whether or not such party had advance notice of the possibility of such damages. Further, no warranty or representation is made or implied relative to freedom from infringement of any third party patents when practicing the specification.

Other product and corporate names may be trademarks of other companies and are used only for explanation and to the owner’s benefit, without intent to infringe.

### **REVISION HISTORY**

| <b>REV</b> | <b>DATE</b>   | <b>DESCRIPTION</b>                             | <b>EDITED BY</b>                       |
|------------|---------------|--|--|
| 1.3        | 17 March 2014 | Revision 1.3, First public release of Part III | Robert V. White<br>Embedded Power Labs |
| 1.3.1      | 13 March 2015 | Second public release                          | Robert V. White<br>Embedded Power Labs |

## Table of Contents

|   |    |
|---|----|
| 1. Introduction .....                                       | 6  |
| 1.1 Specification Scope .....                               | 6  |
| 1.1.1 Specification Structure .....                         | 6  |
| 1.1.2 What Is Included .....                                | 6  |
| 1.1.3 What Is Not Included In The PMBus Specification ..... | 6  |
| 1.2 Specification Changes Since The Last Revision .....     | 6  |
| 1.3 Where To Send Feedback And Comments .....               | 6  |
| 2. Related Documents .....                                  | 7  |
| 2.1 Scope .....   | 7  |
| 2.2 Applicable Documents .....                              | 7  |
| 2.3 Reference Documents .....                               | 7  |
| 3. Reference Information .....                              | 7  |
| 3.1 Device, Signal and Parameter Names .....                | 7  |
| 3.1.1 AVSBus Signal Names .....                             | 7  |
| 3.1.2 Frame Elements .....                                  | 7  |
| 3.2 Numerical Formats .....                                 | 7  |
| 3.2.1 Decimal Numbers .....                                 | 7  |
| 3.2.2 Floating Point Numbers .....                          | 7  |
| 3.2.3 Binary Numbers .....                                  | 7  |
| 3.2.4 Hexadecimal Numbers .....                             | 8  |
| 3.2.5 Examples .....  | 8  |
| 3.3 Bit And Byte Order .....                                | 8  |
| 3.4 Bit And Field Illustrations .....                       | 8  |
| 3.5 Abbreviations, Acronyms And Definitions .....           | 9  |
| 4. General Requirements .....                               | 11 |
| 4.1 Compliance .....  | 11 |
| 4.2 Start Up And Operation .....                            | 12 |
| 5. Transport .....  | 12 |
| 5.1 Topology .....  | 12 |
| 5.2 Operation .....   | 13 |
| 5.3 Modes .....   | 14 |
| 5.3.1 3-Wire Mode .....                                     | 14 |
| 5.3.2 2-Wire Mode .....                                     | 14 |
| 5.4 Frame Size .....  | 14 |
| 5.5 Idle Bus Condition .....                                | 14 |
| 5.6 Slave Resynchronization .....                           | 15 |
| 5.7 Bus Timeout .....                                       | 16 |
| 5.8 Clocking .....  | 16 |
| 5.9 Electrical Interface .....                              | 16 |
| 5.9.1 Timing .....  | 16 |
| 5.9.2 Electrical Drive Levels .....                         | 19 |
| 6. Protocol rules .....                                     | 20 |
| 6.1 Data Launch .....                                       | 20 |
| 6.2 Data Capture .....                                      | 20 |
| 6.3 Data Format .....                                       | 20 |
| 6.4 Unknown Resource Selector .....                         | 20 |

|        |   |    |
|--------|---|----|
| 6.5    | Unavailable Resource .....  | 21 |
| 6.6    | <StartCode> .....   | 21 |
| 6.7    | <SlaveAck> .....  | 21 |
| 6.8    | <StatusResponse> .....  | 22 |
| 6.9    | CRC Verification .....  | 22 |
| 6.10   | Data Validation.....  | 23 |
| 6.11   | Types Of Writes .....   | 23 |
| 6.11.1 | Write and Commit.....   | 23 |
| 6.11.2 | Write and Hold.....   | 23 |
| 7.     | Frame Definition .....  | 24 |
| 7.1    | The Write Frame .....   | 24 |
| 7.2    | The Read Frame.....   | 24 |
| 7.3    | Frame Alignment.....  | 26 |
| 7.4    | Status Response Frame .....   | 27 |
| 8.     | Data Types .....  | 28 |
| 8.1    | Voltage Read/Write (Command Data Type = 0000b).....                         | 28 |
| 8.2    | Vout Transition Rate Read/Write (Command Data Type = 0001b).....            | 29 |
| 8.3    | Current Read (Command Data Type = 0010b) .....                              | 29 |
| 8.4    | Temperature Read (Command Data Type = 0011b).....                           | 29 |
| 8.5    | Voltage Reset (Command Data Type 0100b) .....                               | 29 |
| 8.6    | Power Mode Read/Write (Command Data Type = 0101b).....                      | 30 |
| 8.7    | Reserved for future use (Command Data Types 0110b to 1101b) .....           | 30 |
| 8.8    | AVSBus Status Read/Write (Command Data Type = 1110b) .....                  | 30 |
| 8.9    | AVSBus Version Read (Command Data Type = 1111b).....                        | 31 |
| 8.10   | Manufacturer-Specific Read/Write (Group 1b, Data Types 0000b to 1111b)..... | 32 |
| 9.     | Communication From The AVSBus Slave To The AVSBus Master .....              | 32 |
| 10.    | PMBus™ And AVSBus Device Mapping.....                                       | 32 |
| 11.    | Accuracy .....  | 33 |
|        | Appendix I. Summary Of Changes .....  | 34 |

## Table of Figures

|  |    |
|--|----|
| Figure 1. Bit Order Within A Field .....   | 8  |
| Figure 2. Example AVSBus Topology With One Master And One Slave .....                      | 13 |
| Figure 3. Example AVSBus Topology With Multiple Links .....                                | 13 |
| Figure 4. AVSBus 3-Wire Mode Connections.....  | 14 |
| Figure 5. AVSBus 2-Wire Mode Connections.....  | 14 |
| Figure 6. Timing Diagrams Reference Schematic .....  | 17 |
| Figure 7. Timing diagram for AVS_MData .....   | 17 |
| Figure 8. Timing diagram for AVS_SData.....  | 18 |
| Figure 9. <StatusResponse> Composition .....   | 22 |
| Figure 10. Write Frame Structure .....   | 24 |
| Figure 11. Read Frame Structure .....  | 25 |
| Figure 12. Maximum Throughput.....   | 27 |
| Figure 13. Last Frame In A Sequence. ....  | 27 |
| Figure 14. Status Response Frame .....   | 28 |
| Figure 15. First Frame In A Sequence.....  | 28 |
| Figure 16. AVSBus_Status_Bits .....  | 31 |
| Figure 17. Example Power Management IC With Multiple PAGEs And Multiple AVSBus Ports ..... | 33 |

## Table of Tables

|   |    |
|---|----|
| Table 1. Bit And Field Representation For Frame Illustrations .....               | 8  |
| Table 2. Abbreviations, Acronyms and Definitions Used In This Specification ..... | 9  |
| Table 3. Electrical Characteristics.....  | 18 |
| Table 4. Electrical Drive Levels.....   | 19 |
| Table 5. Frame Fields .....   | 25 |

# 1. Introduction

AVSBus is an interface designed to facilitate and expedite point-to-point communication between an ASIC, FPGA, or other logic, memory, or processor devices and a POL control device on a system for the purpose of adaptive voltage scaling.

When integrated with PMBus, AVSBus is available for allowing independent control and monitoring of one or more rails within one slave.

The communication protocol has been designed with provisions for future growth which include a mechanism for an ASIC to query the version of AVSBus supported. This will allow firmware to make the best use of new features as AVSBus evolves to satisfy system needs, and over time it will permit combining different generations of devices on a single system.

For more information, please see the System Management Interface Forum Web site: <http://www.powerSIG.org>.

## 1.1 Specification Scope

### 1.1.1 Specification Structure

The PMBus™ specification is in three parts.

Part I includes the general requirements, defines the transport, electrical interface and timing requirements of hardwired signals for PMBus.

Part II defines the command language used with PMBus.

Part III, this document, defines the transport, electrical interface, timing requirements and command language for AVSBus.

### 1.1.2 What Is Included

This specification defines two protocols to manage power converters and a power system via communication over digital communication buses. PMBus can be used without the AVSBus extension, and AVSBus could be used on a simple system that does not implement the full PMBus spec, but the two are integrated seamlessly as AVSBus is a powerful extension to PMBus.

### 1.1.3 What Is Not Included In The PMBus Specification

The PMBus specification is not a definition or specification of:

- A particular power conversion device or family of power conversion devices.
- A specification of any individual or family of integrated circuits.

This specification does not address direct unit to unit communication such as analog current sharing, real-time analog or digital voltage tracking, and switching frequency clock signals.

## 1.2 Specification Changes Since The Last Revision

A summary of the changes between this revision and Revision 1.0 are shown in Section 12 at the end of this document.

## 1.3 Where To Send Feedback And Comments

Please send all comments by email to: [TechQuestions@smiforum.org](mailto:TechQuestions@smiforum.org).

## **2. Related Documents**

### **2.1 Scope**

If the requirements of this specification and any of the reference documents are in conflict, this specification shall have precedence unless otherwise stated.

Referenced documents apply only to the extent that they are referenced.

The latest version and all amendments of the referenced documents at the time the device is released to manufacturing apply.

### **2.2 Applicable Documents**

Applicable documents include information that is, by extension, part of this specification.

[A01] PMBus™ Power System Management Protocol, Part I, General Requirements, Transport And Electrical Interface, Revision 1.3

[A02] PMBus™ Power System Management Protocol, Part II, Command Language, Revision 1.3

### **2.3 Reference Documents**

Reference documents have background or supplementary information to this specification. They do not include requirements or specifications that are considered part of this document.

None in this revision.

## **3. Reference Information**

### **3.1 Device, Signal and Parameter Names**

#### **3.1.1 AVSBus Signal Names**

The names of AVSBus signals corresponding to physical connections are given as “AVS”, followed by an underscore (“\_”), followed by the signal name. An example is AVS\_Clock.

#### **3.1.2 Frame Elements**

Values for frame elements are represented by symbolic names shown in mixed-case monospaced font, and enclosed in angle brackets, like <CmdDataType> or <Select>.

### **3.2 Numerical Formats**

All numbers are decimal unless explicitly designated otherwise.

#### **3.2.1 Decimal Numbers**

Numbers explicitly identified as decimal are identified with a suffix of “d”.

#### **3.2.2 Floating Point Numbers**

Numbers explicitly identified as floating point are identified with a suffix of “f”.

#### **3.2.3 Binary Numbers**

Numbers in binary format are indicated by a suffix of “b”.

Unless otherwise indicated, all binary numbers are unsigned. All signed binary numbers are two's complement.

## 3.2.4 Hexadecimal Numbers

Numbers in hexadecimal format are indicated by a suffix of “h”.

## 3.2.5 Examples

255d ⇔ FFh ⇔ 11111111b

175d ⇔ AFh ⇔ 10101111b

1.2f

## 3.3 Bit And Byte Order

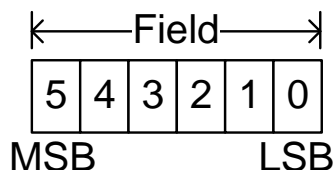
Considering that AVSBus interface frames have fields of multiple sizes, not limited to multiples of 8 bits, there is no concept of byte when determining the order in which information is transmitted.

- When a field of data is transmitted, the most significant bit (MSB) is sent first and the least significant bit (LSB) is sent last, regardless of the number of bits involved. This way the first bit transmitted corresponds to the sign bit in Two's Complement notation.
- Fields are transmitted in the order in which they are depicted in the frame illustrations throughout the document: <StartCode> is sent first, followed by subsequent fields and ending with the last field on the right hand side of the diagram.

## 3.4 Bit And Field Illustrations

The transmission of bits within a field is illustrated in this section.

In all cases, the least significant bit is indicated as Bit 0, as shown below.



**Figure 1. Bit Order Within A Field**

This part of the specification describes transactions over AVSBus. The symbols used to describe the details of those transactions and protocols are shown in Table 1.

**Table 1. Bit And Field Representation For Frame Illustrations**

| Symbol | Meaning   |
|--------|---|
|        | A rectangle with no shading indicates data sent from the host (bus master) to the slave device. |
|        | A rectangle with a shaded interior indicates data sent from the slave device to the bus master. |



| Symbol  | Meaning  |
|---|--|
| <div>3</div> <div>CRC</div> <div>3</div> <div>CRC</div> | <p>By default there is a rectangle for each field sent, not for each bit within a field. Typically the rectangle will have a number over it indicating the number of bits it represents.</p> <p>Shown here is the 3-bit CRC used for verifying the integrity of a transfer. It can be sent by the master or the slave, so it may be found shaded or clear.</p> |
| <div>2</div> <div>01</div>                              | <p>The &lt;StartCode&gt; sent from a bus master device to initiate a transfer is the 2-bit code 01b.</p>   |
| <div>2</div> <div>&lt;SlaveAck&gt;</div>                | <p>When slaves send acknowledgement back to the bus master, they send a 2-bit code, represented by &lt;SlaveAck&gt;.</p>   |
| <div>16</div> <div>&lt;Voltage&gt;</div>                | <p>Data fields are identified by a descriptive word, with the number of bits indicated above, as shown here for a 16-bit voltage value.</p>  |

### 3.5 Abbreviations, Acronyms And Definitions

**Table 2. Abbreviations, Acronyms and Definitions Used In This Specification**

| Term                       | Definition  |
|----------------------------|---|
| ACK,<br><SlaveAck>         | Acknowledge. The response from a slave indicating that it has received a transfer. The encoding used for the 2 bits is explained in Section 6.7.  |
| <AVSBus_Status>            | A 16-bit field composed by concatenating multiple status bits. The associated <CmdDataType> as explained in Section 8.8 describes those flags and how they can be read by the AVSBus Master.<br>It should not be confused with <StatusResponse>.                                  |
| Assert, Asserted           | A signal is asserted when the signal is true. See Negate.   |
| AVS                        | Adaptive Voltage Scaling. AVS is used by a device to control its supply voltage, generally to minimize power consumption for a given operating condition.   |
| Clear                      | When referring to a bit or bits, this means setting the value to zero.  |
| Command Data,<br><CmdData> | A generic term to refer to the actual data carried in a frame, which in the most typical case is a voltage setting. Command codes, selector fields, acknowledgement codes and CRC are not command data. They provide the context for the command data to be properly interpreted. |

| <b>Term</b>                     | <b>Definition</b>  |
|---------------------------------|--|
| Command Data Type,<CmdDataType> | An identifier used to differentiate between the types of data contained in the <CmdData> field of a given AVSBus frame, such as voltage or temperature.  |
| <CRC>                           | A 3-bit Cyclic Redundancy Code used to detect errors in a transfer, which allows the receiving device to discard presumably corrupted data.  |
| Default Store                   | A non-volatile memory store most typically used by the device manufacturer to store default values.  |
| Disable, Disable Output         | To instruct the device to stop the power conversion process and to stop delivering energy to the output. The device's control circuitry remains active and the device can communicate via control buses. |
| Enable, Enable Output           | To instruct the device to start the power conversion process and to start delivering energy to the output.   |
| Host                            | A host must be an AVSBus Master. There is only one host per AVSBus instantiation.  |
| Inhibit                         | To stop the transfer of energy to the output while a given condition, such as excessive internal temperature, is present.  |
| LSB                             | Least significant bit  |
| Master                          | A master is a device that issues commands, generates the clock, and terminates the transfer.   |
| MData                           | Master Data sent via the output port AVS_MData of an AVSBus Master for sending bits to its AVSBus Slave, which is connected to the input port AVS_SData of the slave device. See also SData.             |
| MSB                             | Most significant bit   |
| Negate, Negated                 | A signal is negated when the signal is false. See Assert.  |
| POL                             | Point-of-load.   |
| Rail                            | Generally refers to a power supply output  |
| Rail Selector,<RailSel>         | The identifier for a rail in the device, used as <Selector> in AVS Commands.   |
| SData                           | Slave Data sent via the output port AVS_SData of an AVSBus Slave for sending bits to its AVSBus Master, which is connected to the input port AVS_MData of the master device. See also MData.             |

| <b>Term</b>      | <b>Definition</b>   |
|------------------|---|
| <Selector>       | The identifier that indicates what instance of a device's resources is the target of a command. <Selector> typically refers to a rail number, in which case it is depicted as <RailSel>.  |
| Set              | When referring to a bit or bits, this means setting the value to one.   |
| Shut Down        | Disable or turn off the output. This generally implies that the output remains off until the device is instructed to turn it back on. The device's control circuit remains active and the device can respond to commands received via control buses.            |
| Slave            | A slave is a device that is receiving or responding to a command.   |
| <StartCode>      | A 2-bit field used to mark the beginning of a Master sub-frame.   |
| <StatusResponse> | A 5-bit field composed by concatenating multiple flags. There is no associated <CmdDataType>.<br>The Status Response field will be sent by the AVSBus Slave twice in every frame, as explained in Section 9.<br>It should not be confused with <AVSBus_Status>. |
| X                | When used to define a binary value, X means that the value of that bit is a "don't care" that can be safely ignored.<br>When used in examples, X means that the example applies to any value of that bit. It is a "don't care" for the purpose of the example.  |

## **4. General Requirements**

### **4.1 Compliance**

The AVSBus protocol is intended to cover a wide range of power system architectures and converters, as it can be used stand-alone or as a complement to PMBus.

AVSBus Slaves may not support all of the available features, functions and commands.

To be compliant to the AVSBus specification:

- If a device accepts an AVSBus command code, it must execute the associated function as described in Part III of the PMBus specification (this document).
- If a device does not accept a given AVSBus command code, it must respond as described in Section 6.
- A device must support voltage scaling through implementation of writing voltages as specified in Section 8.1.
- A device configured for 3-wire operation must support reading of <AVSBus\_Status> and sending of <StatusResponse>.

- A device must implement the Resynchronization mechanism described in Section 5.6.

A device may or may not support additional operations, such as reading or writing Vout Transition Rate, reading Temperature, or reading Current. This does not affect compliance.

### 4.2 Start Up And Operation

AVSBus devices, upon application of power, must start up and begin operation in a controlled manner, as programmed internally or externally, without requiring communication from either PMBus or AVSBus.

Furthermore, enabling and disabling the power conversion process must also occur independently of AVSBus communication. Control over enabling and disabling power conversion, as well as all operational parameters, is best achieved through PMBus, with AVSBus relegated to its intended purpose of controlling voltage scaling.

PMBus determines if the AVSBus Master gets control over the settings of the AVSBus Slave, and can take that control away. However, regardless of whether control is given to the AVSBus Master or not, PMBus can always read data, and the AVSBus Master can always read data.

In fact, the AVSBus Master may attempt to write data at any time, but any writes that occur while AVS has not been given control via PMBus will result in the AVSBus Slave responding with the <SlaveAck> for “Unavailable Resource”. See Section 6.5 for more information.

## 5. Transport

### 5.1 Topology

AVSBus is a 3-wire communication link designed to provide bidirectional communication between one powered device (such as an ASIC, FPGA or processor) and one slave for controlling voltage scaling. The three wires used for communication are AVS\_Clock, AVS\_MData, and AVS\_SData.

- AVS\_MData is driven by the AVSBus Master and carries data to the slave,
- AVS\_SData is driven by an AVSBus Slave and carries data to the master, and
- AVS\_Clock is driven by the master and clocks data for both the AVS\_MData and AVS\_SData lines.

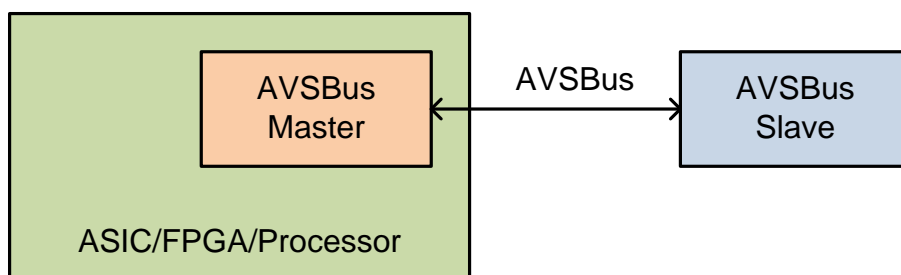
An optional 2-wire unidirectional implementation is allowed using only AVS\_Clock and AVS\_MData. In this limited implementation, the AVSBus Master will never receive acknowledgments, status or data back from a slave device.

Note: AVSBus is behaviorally and electrically similar to SPI bus without chip select lines:

- AVS\_MData and AVS\_SData are equivalent to MOSI and MISO.
- AVS\_Clock is equivalent to CLK of the SPI bus.

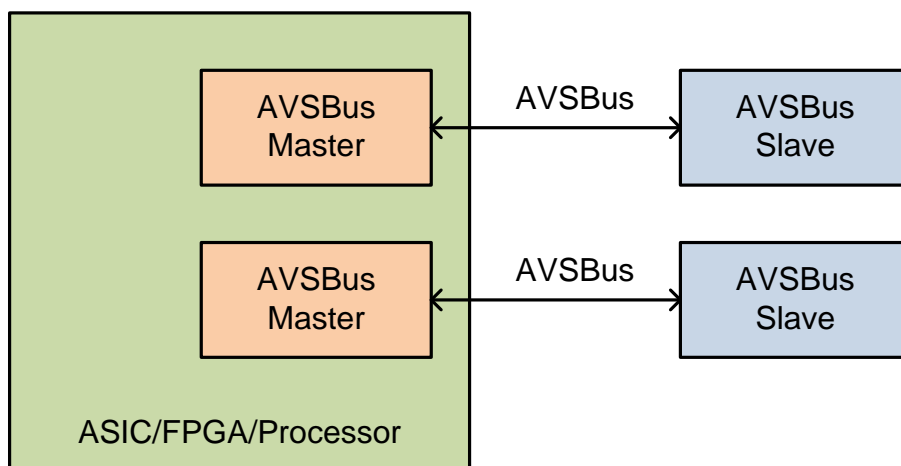
The only topology supported is one-to-one dedicated link between an AVSBus Slave and its controlling AVSBus Master.

In the example shown in Figure 2 the ASIC (or FPGA or processor) is the AVSBus Master and the other device in the link is the AVSBus Slave. The AVSBus Slave may support one or more voltage rails, each one individually addressable by the AVSBus Master. In either case each rail may be composed of multiple phases.



**Figure 2. Example AVSBus Topology With One Master And One Slave**

In the example shown in Figure 3 one ASIC (or FPGA or processor) includes two AVSBus Masters, each of which communicates by a dedicated link to one AVSBus slave.



**Figure 3. Example AVSBus Topology With Multiple Links**

## 5.2 Operation

Each type of device in a link performs specific complementary functions:

1. The AVSBus Master must initiate all data transfers.

Besides setting voltages, a master may also use AVSBus to request data of various types from the slave.

The master must guarantee that AVS\_MData is held at a logic value '1' when the clock is not running: during initialization prior to any bus transfers, as well as during idling between frames. AVS\_MData is not allowed to be at a logic value '0' when the clock starts.

2. The AVSBus Slave listens for and may respond to master commands but cannot initiate a transfer under any circumstances.

In general the slave must guarantee that AVS\_SData is held at a logic value '1' when the clock is not running: during initialization prior to any bus transfers, between frames, as well as at any time during a transfer when the slave is not sending data to the master. An exception to this rule is that the AVSBus Slave will set AVS\_SData to a logic value '0' when it needs to alert the AVSBus Master to start a frame so that it can send <StatusResponse>.

See the Section 9 for more details.

## 5.3 Modes

There are two different modes of operation for AVSBus, 3-wire mode and 2-wire mode. In both modes the pin names are identical in master and slave and matching pins are connected to each other.

### 5.3.1 3-Wire Mode

The 3-wire mode is the complete implementation of AVSBus that allows a master to receive acknowledgements from slaves in response to write operations, to read back data and configuration from them, and to receive <StatusResponse> in every frame. The figure below illustrates a typical implementation.

This mode requires AVS\_SData.

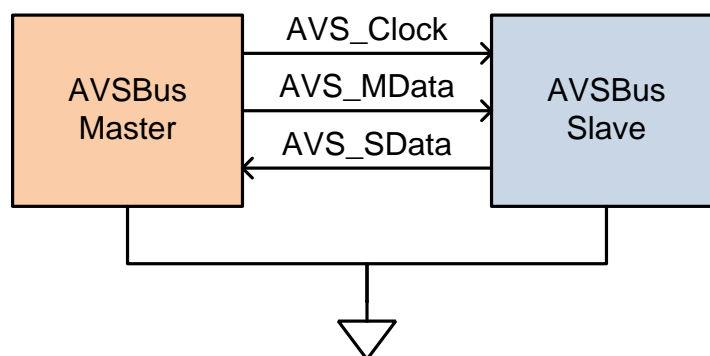


Figure 4. AVSBus 3-Wire Mode Connections

### 5.3.2 2-Wire Mode

The 2-wire mode is a partial implementation of AVSBus in which a slave does not come with the optional AVS\_SData output, thus rendering it incapable of sending any data back to its master.

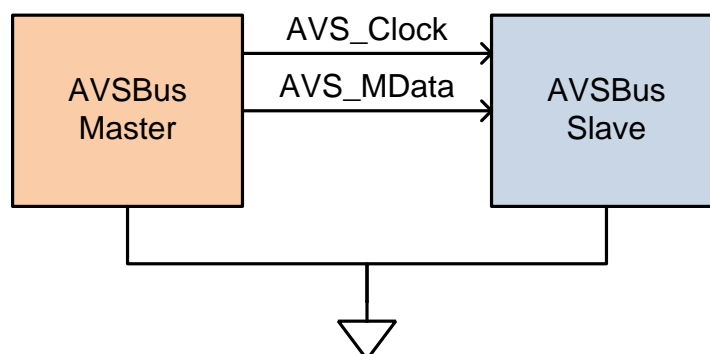


Figure 5. AVSBus 2-Wire Mode Connections

## 5.4 Frame Size

AVSBus frames are 64 bits long. There are two different types of frames, as explained in Section 7.

## 5.5 Idle Bus Condition

The bus is considered idle in two cases:

1. When the clock is suspended between frames.

In this case the state of AVS\_MData is a “Don’t care” to the slave, but it is required that the line be kept high to enable proper detection of <StartCode> when the clock resumes.

The state of AVS\_SData is relevant to the master even when the clock is suspended. Considering that the slave has no control over when the master may start providing a clock, a slave that wants to alert the master to the fact that there has been an important status change will simply change AVS\_SData from its default state (Logic 1) to a signaling state (Logic 0), and keep the signaling state until the master initiates a frame. Except for this interrupt scenario, the slave must keep the line high when the clock is suspended.

As a consequence both lines must be held high while the clock is suspended, to support detection of start codes and for requesting a frame.

2. When, even in the presence of a clock, no frames are being transmitted.

Whenever the clock is running AVS\_MData must be held high (Logic 1) if the AVSBus Master is not sending any data. Again that level is a precondition for issuing the <StartCode> which consists of the two bit sequence 01b. In effect, the first logic 0 encountered while the bus has been in idle condition marks the beginning of a transfer, as there is no concept of persistent 8-bit boundaries when the clock runs in idle condition.

Likewise, AVS\_SData should be held high (Logic 1) when no frames are being transmitted by the AVSBus Slave. Just like when the clock is suspended, a slave that wants to alert the master to the fact that there has been an important status change will simply change AVS\_SData from its default state (Logic 1) to a signaling state (Logic 0), and keep the signaling state until the master initiates a frame.

To effect this, the AVSBus Slave must be designed to monitor AVS\_MData continuously, not just for the purpose of receiving a new frame, but to guarantee that the state of AVS\_SData does not change between the 0b and the 1b of the 01b sequence for starting a frame. Conversely, the AVSBus Master must be designed to ignore the value of AVS\_SData while it sends the 01b sequence, as long as it receives either 00b (alert issued) or 11b (no alert issued). Receiving 01b or 10b would constitute an error.

### 5.6 Slave Resynchronization

An AVS Slave must implement the resynchronization mechanism to recover in case noise in the line or some other artifact has caused it to reach an incorrect state. Receiving 34 clock pulses while AVS\_MData is held high will cause a slave to resynchronize its communication interface, and wait for the next <Start\_Code>.

For the resynchronization mechanism to work correctly, the AVSBus Slave must keep counting ones as long as it is receiving clock pulses, and reset the count whenever it receives a zero. This will ensure the earliest possible detection of a sequence of 34 ones.

In order to prevent aborting a valid frame through resynchronization when a frame ends with a few consecutive ones, it is imperative that the counter be reset at the successful completion of a master sub-frame. In this context successful means that <CRC> verification passed, ensuring there were no communication errors, independently of whether the data contained in the frame is valid or not.

It is recommended for the master to resynchronize the slave at startup, and to do so periodically afterwards.

### 5.7 Bus Timeout

AVSBus does not require that slaves implement bus timeout functions. Such a feature would be useful for detecting when AVS\_Clock has stalled so that the slave can take some protective action, but it would impose additional burdens on the slave which may be too costly.

Though not required, a mechanism for detecting AVSBus timeout should be given serious consideration and implemented whenever possible. The purpose of such a timeout function is for a slave to reset its AVSBus interface when it determines that an ongoing transfer has been effectively aborted because the clock stopped running as expected. Doing this will get the slave ready to properly detect a future <Start\_Code>. Otherwise, when the master recovers and tries to start sending a brand new frame, the slave would process the bits it receives as a continuation of whatever frame had been interrupted. This will make the new frame fail.

It is suggested that AVSBus masters do not simply rely on slaves' implementation of Bus Timeout. Instead, they should trigger the Slave Resynchronization mechanism as often as practical for their application.

### 5.8 Clocking

The bus can transfer data continuously when needed, which would require an uninterrupted clock from the master. For periods with no bus activity, the master can suspend AVS\_Clock and then restart it later when needed.

A suspended or inactive clock will be held in a logic '0' state, to ensure that whenever the clock starts, the first detectable edge is the rising edge.

The following section describes the timing requirements of AVS\_Clock.

### 5.9 Electrical Interface

The AVSBus Master always provides AVS\_Clock and AVS\_MData (Master Data), and it may or may not have an input for AVS\_SData (Slave Data), depending on whether 2-wire or 3-wire Mode is implemented.

The logic level for data and clock lines must be constant for any given application. An AVSBus-compliant device will have a predetermined I/O logic level within the specified range. Since the AVS\_Clock and AVS\_MData lines are always driven by the master and AVS\_SData is always driven by the slave, there is no need for pull-up resistors for normal operation. However, it is recommended that the system be designed with weak pull-up resistors for robustness, so that all three lines are at known voltage levels in case one of the devices is not powered up or not present.

#### 5.9.1 Timing

The following diagrams and tables illustrate the timing requirements for AVSBus. Notice that the Slave-to-Master data path is grayed-out in the schematic, since this is optional functionality. For the same reason, the timing tables have separate figures for launch and capture by the Master and the Slave.

In this interface, both master and slave launch data from the rising edge of the clock, and they both capture data using the falling edge.



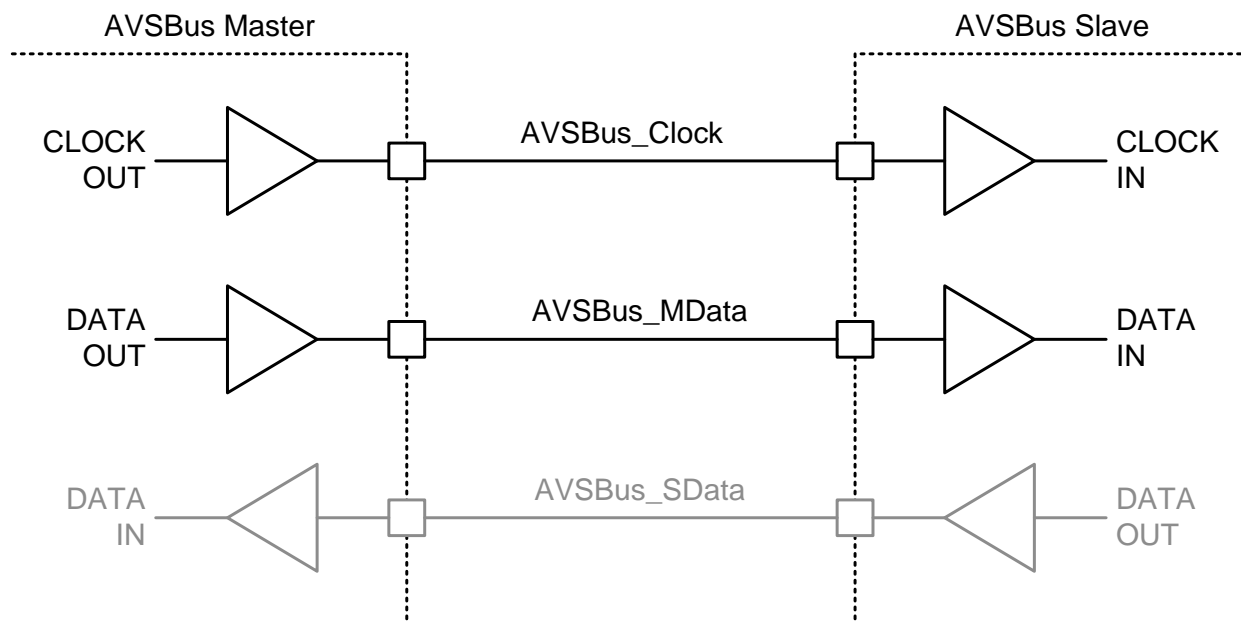


Figure 6. Timing Diagrams Reference Schematic

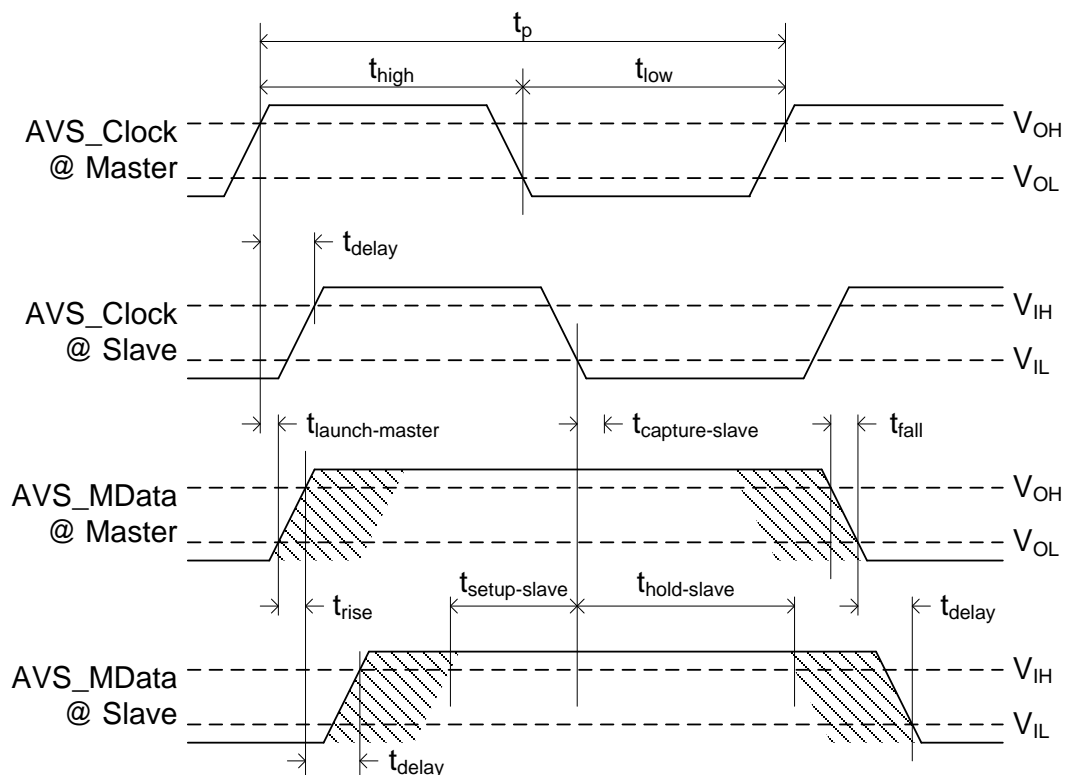
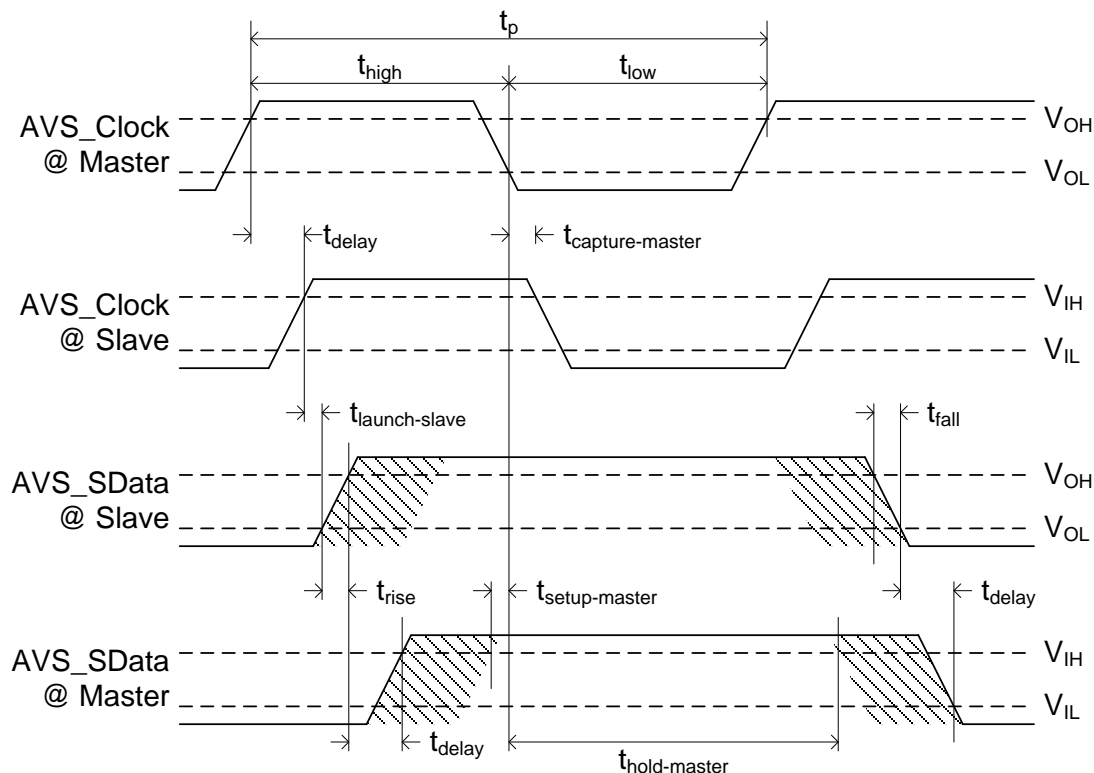


Figure 7. Timing diagram for AVS\_MData



**Figure 8. Timing diagram for AVS\_SData**

The timing diagrams, Figure 7 and Figure 8, show a skew in the clock and data lines, labeled  $t_{delay}$ , which corresponds to transmission delays on the board. A comparable delay affects the data lines. Setup and hold times are measured at the pads of the master or the slave, as pertinent.

**Table 3. Electrical Characteristics**

| Parameter   | Description   | Min. | Typ.        | Max. | Units |
|-------------|---|------|-------------|------|-------|
| $t_p$       | Period for active clock   | 20   |             | 200  | ns    |
| $f_{clock}$ | Frequency for active clock                                      |      | $1/t_p$     |      |       |
| $t_{high}$  | Duration of the high-phase of clock                             | 10   |             |      | ns    |
| $t_{low}$   | Duration of the low-phase of clock                              |      | $t_p/2$     |      | ns    |
| $t_{rise}$  | Rise time for data.   |      | $1.5\% t_p$ | 3    | ns    |
| $t_{fall}$  | Fall time for data.   |      | $1.5\% t_p$ | 3    | ns    |
| $t_{delay}$ | Time for signals to propagate from one device to the other one. |      | 4           |      | ns    |

| Parameter                        | Description   | Min.                   | Typ.                   | Max.   | Units |
|----------------------------------|---|------------------------|------------------------|--------|-------|
| <b>Master-to-Slave data path</b> |   |                        |                        |        |       |
| $t_{\text{launch-Master}}$       | Time from rising clock edge in Master to data-out transition at Master's data-out port. | 0                      | 2                      |        | ns    |
| $t_{\text{capture-Slave}}$       | Time from falling clock edge in Master to data capture inside Slave.                    | $t_{\text{delay}}$     | $2 + t_{\text{delay}}$ |        | ns    |
| $t_{\text{setup-Slave}}$         | Time from data-out edge in Master to capture clock edge in Slave.                       | $2 + t_{\text{delay}}$ |                        |        | ns    |
| $t_{\text{hold-Slave}}$          | Time from capture clock edge in Slave to data-out edge in Master (for next bit).        | 2                      | $t_{\text{low}}$       |        | ns    |
| <b>Slave-to-Master data path</b> |   |                        |                        |        |       |
| $t_{\text{launch-Slave}}$        | Time from rising clock edge in Master to data-out transition at Slave's data-out port.  | $2 + t_{\text{delay}}$ | $4 + t_{\text{delay}}$ | Note 1 | ns    |
| $t_{\text{capture-Master}}$      | Time from falling clock edge in Master to data capture inside Master.                   | 0                      | 2                      |        | ns    |
| $t_{\text{setup-Master}}$        | Time from data-out edge in Slave to capture clock edge in Master.                       | $2 + t_{\text{delay}}$ |                        |        | ns    |
| $t_{\text{hold-Master}}$         | Time from capture clock edge in Master to data-out edge in Slave (for next bit).        | 2                      | $t_{\text{low}}$       |        | ns    |

Note 1: The clock used by the slave is a delayed version of the clock in the master. For that reason, launching data from the slave starts later than launching from the master, and relatively speaking, capturing by the master comes earlier. If  $t_{\text{delay}}$  is large on a given board, it may be necessary to increase  $t_{\text{high}}$  to compensate and give more time for the data to go from the slave to the master.

## 5.9.2 Electrical Drive Levels

The electrical drive levels for AVSBus are independent of those for PMBus. Special care should be taken to ensure that the power sequencing is robust enough that all interoperability is not compromised.

The general requirement is that the master will determine the drive levels and the slave must work with those levels. The simplest way to do this is to have a common  $V_{\text{DD}}$  for the master and the slave I/O circuitry. There are also adaptive detect and drive level adjustment mechanisms that could be used by the slave to match the drive levels supplied by the master.

**Table 4. Electrical Drive Levels**

| Parameter | Description | Limits |     | Units | Comments |
|-----------|-------------|--------|-----|-------|----------|
|           |             | Min    | Max |       |          |

| Parameter             | Description                       | Limits              |                     | Units   | Comments               |
|-----------------------|-----------------------------------|---------------------|---------------------|---------|------------------------|
|                       |                                   | Min                 | Max                 |         |                        |
| V <sub>DD</sub>       | AVS Master bus voltage            | 0.9                 | 3.63                | V       | 1 V to 3.3 V $\pm$ 10% |
| V <sub>IL</sub>       | AVSBus signal Input Low voltage   |                     | 40% V <sub>DD</sub> | V       |                        |
| V <sub>IH</sub>       | AVSBus signal Input High voltage  | 60% V <sub>DD</sub> |                     | V       |                        |
| V <sub>OL</sub>       | AVSBus signal Output Low voltage  |                     | 20% V <sub>DD</sub> | V       |                        |
| V <sub>OH</sub>       | AVSBus signal Output High voltage | 80% V <sub>DD</sub> |                     | V       |                        |
| I <sub>LEAK-PIN</sub> | Input Leakage per pin             | -10                 | +10                 | $\mu$ A |                        |

## 6. Protocol rules

For the rules outlined below there is no distinction between master and slave unless indicated. Instead, the role of transmitter or receiver that they play at any given time is what matters.

### 6.1 Data Launch

A transmitter launches data on the rising edge of the clock.

### 6.2 Data Capture

A receiver captures data on the falling edge of the clock.

### 6.3 Data Format

Some AVSBus data types represent numeric data, others represent bit fields with custom encoding. Those representing numeric data use standard Two's Complement representation, with data getting sign-extended to fit the data type size. Those that consist of bit fields are padded with zeroes. In either case the resulting data can be described as LSB-aligned.

When a PMBus system is configured to use any data format other than Two's Complement (for example, the DIRECT data format), it is up to the manufacturer of the AVSBus Slave to determine how to make the data conversion to and from Two's Complement, and to specify in the documentation any limitations.

### 6.4 Unknown Resource Selector

An AVSBus Slave that receives a command for an unknown resource will respond with a special <SlaveAck> code that indicates that no action was taken because the resource does not exist. See Section 6.7

An unknown resource may be an unsupported <CmdDataType> (e.g., attempting to read temperature on a device that does not provide temperature values, or to write a read-

only data type like temperature or current) or a selector for a non-existing instance (e.g., referring to rail 3 when there are only 2 rails).

### 6.5 Unavailable Resource

An AVSBus Slave that receives a read or a write frame for one of its resources which is not available will respond with a special <SlaveAck> code that indicates that no action was taken because the operation cannot be carried out at the present time. See Section 6.7

A resource that is not available may be in a disabled state (e.g., attempting to set a target voltage on a rail that is turned off) or be busy (e.g., attempting to read temperature when the A/D converter is occupied in some other measurement and there is no temperature available to send back).

### 6.6 <StartCode>

AVS\_MData having a default state of '1' sets the conditions for the detection by a slave of the start of a new frame through a <StartCode> since the first bit of that code is a '0' as indicated in Table 5.

### 6.7 <SlaveAck>

Whenever an AVSBus Master sends a command, the AVSBus Slave will send back a 2-bit acknowledgment. The two bits encode the following four states, shown here in order of precedence:

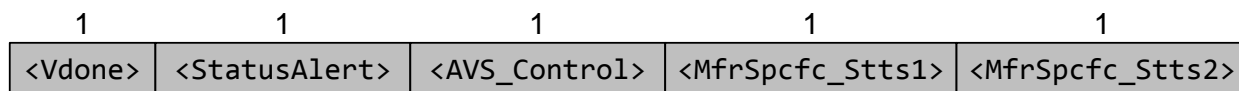
- 10b: Bad CRC, no action is taken.
- 11b: Good CRC, but invalid selector (e.g. unknown resource, or use of the broadcast selector when not supported), invalid data type (e.g. non-existent command), incorrect data (e.g. out of range data value) or incorrect action (e.g. attempting to read a data type that is not readable, or to write to a data type that is read-only). No action is taken.
- 01b: Good CRC, valid data (selector, data type, data value are good), but no action is taken due to resource being unavailable (busy or not allocated to AVSBus). It should be noted that a value that is inherently valid but may be unworkable for the device must be treated as a good value rather than invalid data. For instance, a voltage setting that is too high for the current voltage limits and must be clamped will result on a <SlaveAck> of 01b since effectively the command was executed: The voltage changed to the clamp value.
- 00b: Good CRC, valid data (selector, data type, data value are good) and resource available. Action taken.

The 11b value is effectively a catch-all for multiple conditions that may prevent the slave from carrying out a command. It is up to the device manufacturer to provide means for the master to discern the real cause by perhaps setting up a custom data type with more granularity on those conditions that can make a command fail.

Likewise there may be cases in which a 01b value may not tell the whole story, as in the example above with voltage clamping. Again, it is up to the device manufacturer to determine if such conditions warrant issuing special alerts through one of the bits in <MfrSpcfc\_8>.

## 6.8 <StatusResponse>

Every time an AVSBus Master initiates a frame, the slave must send back a response that can be used by the master to assess the general condition of the slave. This is a 5-bit field formed like this:



**Figure 9. <StatusResponse> Composition**

Where:

- <VDone> corresponds to the combined <VDone> bits of all active rails, ANDed together as described in Section 8.8.
- <StatusAlert> indicates that at least one of the status bits in Section 8.8 is asserted, excluding <VDone>. The AVSBus Master should consider issuing a read for AVS Status when <StatusAlert> is asserted.  
Notice that since <VDone> is sent by itself in the response, it is excluded from the conditions that cause <StatusAlert> to be set, so as to avoid triggering unnecessary read operations.
- <AVS\_Control> indicates whether AVSBus is controlling at least one of the device's output or not. When AVSBus is controlling at least one of the outputs, it returns 1b. When AVSBus is not in control of any output, it returns 0b.
- <MfrSpcfc\_Stts1> a status bit whose definition is left up to the AVSBus Slave's manufacturer. It must be defined with positive logic: the status is considered active (set) when the value returned is 1b; otherwise it will be returned as 0b.
- <MfrSpcfc\_Stts2> another status bit whose definition is left up to the AVSBus Slave's manufacturer. It must follow the same rules.

## 6.9 CRC Verification

Support for 3-bit CRC is required for all AVSBus devices. The transmitter uses the CRC-3 polynomial on the first 29 bits in a sub-frame, and it generates the CRC code to send. The receiver then takes the entire sub-frame, including the CRC itself, and uses the same polynomial to confirm integrity of the data by verifying the CRC.

The polynomial used to calculate the 3-bit CRC is:

$$\text{CRC}(x) = x^0 + x^1 + x^3$$

If data being transferred does not pass CRC verification, the action taken by the receiver depends on what device is playing that role. Generation and validation of the <CRC> requires that the shift register be set to all-zeroes prior to the first shift.

An AVSBus Master that receives questionable data from a slave will discard the data. It should initiate the transfer once more.

An AVSBus Slave that receives questionable command/data from a master will send the corresponding <SlaveAck> code indicating that the specific action requested was not performed due to a bad <CRC> and ignore the command/data.

### 6.10 Data Validation

It is crucial that an AVSBus Slave be able to protect itself and its loads by determining if a voltage setting is unacceptable. To that effect it is required that a compliant AVSBus Slave perform a range check for each voltage write. If the value is larger than the maximum or smaller than the minimum provided, the slave will respond with the <SlaveAck> code for “Incorrect Data”, described in Section 6.7

The PMBus commands VOUT\_MAX and VOUT\_MIN shall be used for providing the range used by AVSBus for voltage validation.

### 6.11 Types Of Writes

AVSBus offers two types of writes, one of which offers functionality that allows the AVSBus Master to synchronize changes across rails in a device. It is important to notice that the PMBus setting of WRITE\_PROTECT does not have any effect on AVSBus writes. The PMBus WRITE\_PROTECT command is not meant to prevent a device from managing its own power.

#### 6.11.1 Write and Commit

The traditional write operation: This type of write will also commit values held for other instances of the same command data type as described for “Write and Hold” below.

Every <CmdDataType> that supports writing must support Write and Commit which is encoded as 00b in <Cmd>.

#### 6.11.2 Write and Hold

A very flexible write operation in which the value carried in the command data field is stored in a holding area but does not take effect (is not committed). What happens to the stored value depends on future commands, as follows:

- If a value is written with Write and Hold for a given instance of a command data type, a subsequent Write and Commit command for any instance of the same command data type will cause the value being held to be committed.
- If a value is written with Write and Hold for a given instance and then again later with another Write and Hold command for the same instance, the first value is discarded and never used. The new value is held to be committed later.
- If a value is written with Write and Hold and then a new value is written for the same instance with a Write and Commit command, the first value is discarded and never used.

For example, an AVS Master could use Write and Hold for programming new transition rates and voltages in 2 rails of a device, and have them there ready for later use. At a later time it would issue a Write and Commit command for the transition rate of the third rail, which would also commit the transition rates of the other two, while the voltages remain uncommitted.

For a simple AVSBus Slave with only one instance of a command data type, use of the Write and Hold command is pointless: the only way to commit a value written with it is to use a command that replaces it.

Supporting Write and Hold, which is encoded as 01b in <Cmd> is not mandatory, unless clearly stated in this specification for individual command data types.

7. Frame Definition

The communications protocol consists of two frame layouts: a Write frame and a Read frame. The frames themselves have two segments (sub-frames): the Master segment (sent over AVS\_MData), which always begins with <StartCode>, and the Slave segment (sent over AVS\_SData).

There is a causal relationship between sub-frames:

- A Master Write sub-frame is immediately followed by a Slave Write sub-frame.
- A Master Read sub-frame is immediately followed by a Slave Read sub-frame.

According to that description, the two sub-frames of a frame happen sequentially one after the other. This is the default behavior, and it is depicted in the illustrations that accompany the frame descriptions below, which are followed by detailed field definitions in Table 5.

7.1 The Write Frame

This type of frame contains the following fields in the Master sub-frame, carried in the AVS\_MData line:

- <StartCode>
- <Cmd> (Either 00b or 01b in write frames)
- <CmdGroup>
- <CmdDataType>
- <Select>
- <CmdData>
- <CRC>

As well as the Slave sub-frame, carried in the AVS\_SData line:

- <SlaveAck>
- 0b
- <StatusResponse>
- <Reserved\_21> (Always all 1's)
- <CRC>

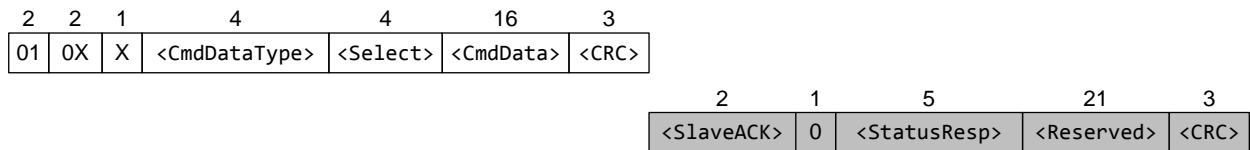


Figure 10. Write Frame Structure

7.2 The Read Frame

This type of frame contains the following fields in the Master sub-frame, carried in the AVS\_MData line:

- <StartCode>
- <Cmd> (Always 11b in read frames)
- <CmdGroup>



- <CmdDataType>
- <Select>
- <Reserved\_16> (Always all 1's)
- <CRC>

As well as the Slave sub-frame, carried in the AVS\_SData line:

- <SlaveAck>
- 0b
- <StatusResponse>
- <CmdData>
- <Reserved\_5> (Always all 1's)
- <CRC>

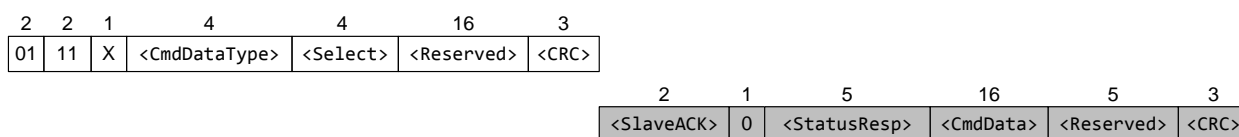


Figure 11. Read Frame Structure

Table 5. Frame Fields

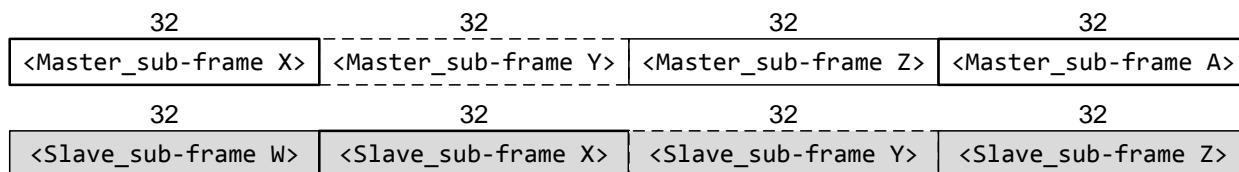
| Field Name  | Size (bits) | Description   |
|-------------|-------------|---|
| <StartCode> | 2           | 01b is the code that activates the frame-decoding logic.  |
| <Cmd>       | 2           | Command code that determines the action that the master requires: <ul style="list-style-type: none"> <li>• 11b: Read data.</li> <li>• 10b: Reserved.</li> <li>• 01b: Write data and hold, but do not commit (leave pending).</li> <li>• 00b: Write data and commit all pending voltage writes.</li> </ul> |
| <CmdGroup>  | 1           | Qualifier to distinguish between two groups of data types: <ul style="list-style-type: none"> <li>• 0b for fully defined AVSBus data types.</li> <li>• 1b for manufacturer-specific data types.</li> </ul>  |

| <b>Field Name</b> | <b>Size (bits)</b> | <b>Description</b>  |
|-------------------|--------------------|---|
| <CmdDataType>     | 4                  | Type of data to which <Cmd> applies. For <CmdGroup> = 0b, the data types are: <ul style="list-style-type: none"> <li>• 0000b: Target rail voltage.</li> <li>• 0001b: Target rail Vout transition rate.</li> <li>• 0010b: Rail current (read only).</li> <li>• 0011b: Rail temperature (read only).</li> <li>• 0100b: Reset rail voltage to default value (write only).</li> <li>• 0101b: Rail power mode.</li> <li>• 0110b to 1101b: Reserved command data types.</li> <li>• 1110b: AVSBus Status</li> <li>• 1111b: AVSBus Version</li> </ul> For <CmdGroup> = 1b the definition of the data types is found in the device's product literature. |
| <Select>          | 4                  | Selector field to differentiate between instances of a command data type on a device. <ul style="list-style-type: none"> <li>• For &lt;CmdGroup&gt; = 0b, this is a rail selector: &lt;RailSel&gt;.</li> <li>• For &lt;CmdGroup&gt; = 1b, this is left up to the device manufacturer to specify.</li> </ul> The value 1111b is called a "Broadcast Frame". It applies to all instances. For example, it applies to all rails when the selector corresponds to <RailSel>. Broadcast only makes sense for writes, not reads, with the exception of AVSBus Status Read. See Section 8.8.   |
| <CmdData>         | 16                 | Data being transferred.   |
| <CRC>             | 3                  | A 3-bit field used to detect the presence of errors in the transmission of a sub-frame.   |
| <Reserved_N>      | N                  | A number of bits reserved for future use. Reserved bits must be sent as all 1's.  |
| <SlaveAck>        | 2                  | Response from a slave.  |
| <StatusResponse>  | 5                  | AVSBus status bits providing a high-level view of the condition of the device. For more details, see Section 8.8.   |

### 7.3 Frame Alignment

Frames can happen in complete isolation from each other, sequentially one after the other, or overlapping. How frames occur with respect to each other over time affects the overall throughput of the bus and determines how the gaps are filled.

In the most efficient use of bandwidth, an AVSBus Master will continually send frames back to back. This is what such a sequence would look like:

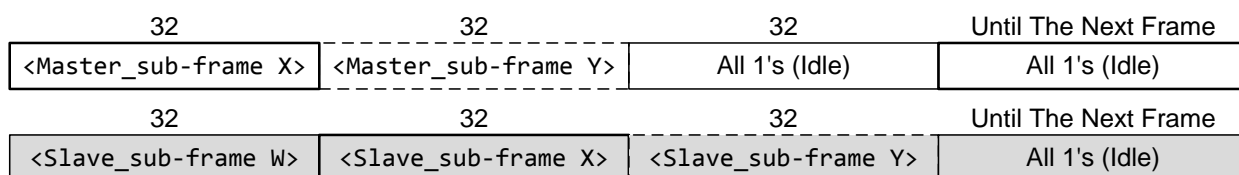


**Figure 12. Maximum Throughput**

Notice that the two pieces of Frame\_Y in the middle of the diagram happen back to back, as illustrated earlier for read and for write frames. However, unlike those illustrations, the AVSBus Master initiated <Master\_sub-frame\_Y> while the AVSBus Slave was sending <Slave\_sub-frame\_X>.

Note that when master and slave sub-frames overlap as described, <StartCode> for the upcoming frame occurs exactly at the same time as <SlaveAck> for the frame currently in progress.

This overlapping can continue for as long as there are more frames for the master to initiate, but sooner or later the AVSBus Master will have nothing to send. When that occurs, there is no <StartCode> overlapping the last <SlaveAck>. Instead, AVS\_MData must be returned to its default state (high) until the end of the slave sub-frame. Then since the channel goes idle, both AVS\_MData and AVS\_SData stay high indefinitely. This is what the end of a sequence would look like:



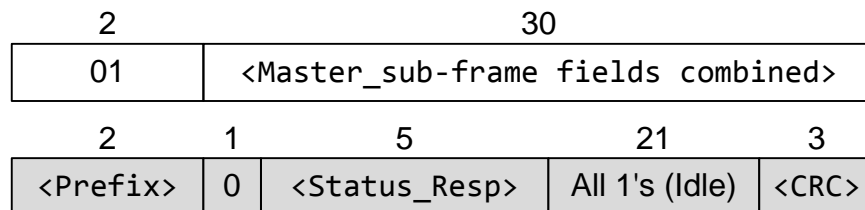
**Figure 13. Last Frame In A Sequence.**

## 7.4 Status Response Frame

At some point, after the AVSBus has been idle, there will be more frames to send, and the first <StartCode> is issued by the AVSBus Master. With the frame definitions we have so far, there would be no alternative for the AVSBus Slave but to keep AVS\_SData high while it receives the first master sub-frame in a sequence. However, there is a special type of slave sub-frame created to take advantage of that bandwidth: the Status Response Frame.

Since this frame is not sent as a response to a master sub-frame, there is no <SlaveAck> to send. <SlaveAck> is always part of the reply to a master sub-frame, and in this situation the AVSBus Master is not awaiting a reply. There is no <CmdData> either, since there is no read command pending completion. But <StatusResponse> can bring valuable and timely information to the master, and that is the purpose of the Status Response Frame.

The Status Response Frame is structured as shown in Figure 14 in the context of a generic master sub-frame. The zero at the 3rd bit of the frame is followed by the five bits that make up <StatusResponse> which in turn are followed by all 1's padding to preserve the frame size, and finally a standard <CRC>.



**Figure 14. Status Response Frame**

The <Prefix> is a 2-bit placeholder at the beginning of the frame. Only two values are valid for this prefix.

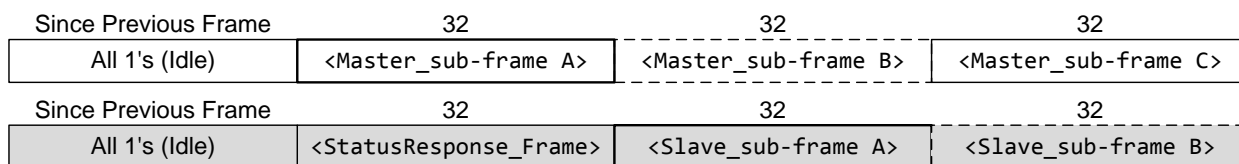
- If the slave signals an interrupt by lowering AVS\_SData prior to the appearance of <StartCode>, it would naturally be expected to be 00b, providing continuity.
- If the slave did not signal an interrupt, AVS\_SData is at its default high state when <StartCode> appears, and it would retain that value so it can be expected to be 11b.

Considering that it is conceivable that the AVSBus Slave may not be ready for the <StartCode>, it is necessary that the AVSBus Master be designed to watch out for the AVS\_SData set low during the 3rd bit of the first byte of a Status Response frame. As long as the line is low during that bit, it should accept <StatusResponse> as valid.

<StatusResponse> is formed by concatenating <VDone>, <StatusAlert> and other flags as outlined in Section 6.8. The format of the Status Response Frame shown here is directly compatible with the reply of a slave to a command from the master, with two differences:

- <Prefix> is replaced with <SlaveAck> when the slave is responding to a master command.
- The all 1's idle field corresponds to the rest of the slave's response, which is different for a read than for a write.

Incorporating the concept of the Status Response Frame, the first frame in a sequence of overlapping frames would appear as shown in Figure 15.



**Figure 15. First Frame In A Sequence**

## 8. Data Types

This section describes in detail each one of the commands supported by AVSBus.

### 8.1 Voltage Read/Write (Command Data Type = 0000b)

This type allows the AVSBus Master to read or write a new voltage target for a rail on an AVSBus Slave. See Part II, Section 12.1 for a full description of the handoff between AVSBus and PMBus control of the output voltage.

In this case <Select> is <RailSel>, and <CmdData> is <Voltage> which is a 16-bit unsigned integer field with 1 LSB = 1 mV.

If the new voltage is too high or too low, as determined by the values set through the PMBus commands VOUT\_MIN and VOUT\_MAX, the AVS Slave will respond with the <SlaveAck> for incorrect data.

### 8.2 Vout Transition Rate Read/Write (Command Data Type = 0001b)

This type allows the AVSBus Master to read or write a new transition rate for a rail on an AVSBus Slave.

In this case <Select> is <RailSel>, and <CmdData> is <TransRates> which is a 16-bit field composed of two fields: <RiseTransRate> and <FallTransRate>. Each field is an 8-bit unsigned integer with 1 LSB = 1 mV/μs.

The fields are concatenated in the order they were described: <RiseTransRate> is sent first, followed by <FallTransRate>.

The AVSBus Slave will use <RiseTransRate> when the rail voltage is increasing, and <FallTransRate> when the rail voltage is decreasing. When AVSBus is enabled, the initial transition rates will be specified by the AVSBus Slave manufacturer and given in the product literature.

### 8.3 Current Read (Command Data Type = 0010b)

This type allows the AVSBus Master to read the current for a rail on an AVSBus Slave. Writes are not supported.

In this case <Select> is <RailSel>, and <CmdData> is <Current> which is a 16-bit unsigned integer field with 1 LSB = 10 mA.

AVSBus does not dictate whether the value being read is filtered in any particular way, leaving it up to each specific AVSBus Slave's manufacturer product literature to clearly describe if filtering is used, and whether it is configurable by PMBus.

### 8.4 Temperature Read (Command Data Type = 0011b)

This type allows the AVSBus Master to read the temperature for a rail on an AVSBus Slave. Writes are not supported.

In this case <Select> is <RailSel>, and <CmdData> is <Temperature> which is a 16-bit signed integer field with 1 LSB = 0.1 °C.

AVSBus does not dictate whether the value being read is filtered in any particular way, leaving it up to each specific AVSBus Slave's manufacturer product literature to clearly describe if filtering is used, and whether it is configurable by PMBus.

### 8.5 Voltage Reset (Command Data Type 0100b)

This type allows the AVSBus Master to force a "Predetermined Value" for an AVSBus Slave rail, particularly while handling exceptions that could otherwise cause damage. The device manufacturer must decide how the value is predetermined. Some of the choices are: Through PMBus, through an AVSBus manufacturer-specific data type, through external pins. Whatever the method chosen, it must be clearly specified in the documentation.

Since there is no associated data for this command, <CmdData> must be set to all 0's. The predefined value is determined by the manufacturer of the AVSBus Slave.

Resetting the rail's voltage through this data type is deemed an emergency reaction to a problem. As such, it can be assumed that a slave would use the fastest transition rate it is capable of, disregarding the current Vout Transition Rate setting.

It is envisioned that this command would be particularly useful when used in broadcast mode to all the rails in a device (by setting <RailSel> to all 1's), giving the AVSBus Master the ability to reset all rails to their own individual reset values in a single frame.

In this case <Select> is <RailSel>, and <CmdData> is 00h.

### 8.6 Power Mode Read/Write (Command Data Type = 0101b)

This type allows the AVSBus Master to set the mode of operation for the output of an AVSBus Slave rail. There are 8 possible settings: four predefined settings referred to as the "AVSBus Standard Power Modes", and four manufacturer-specific modes.

In this case, <Select> is <RailSel> and <CmdData> is <PowerMode> which is a 3-bit field with the following encoding:

- 000b – Maximum Efficiency
- 001b – Reserved
- 010b – Reserved
- 011b – Maximum Power
- 100b to 111b: Manufacturer-specific settings

Following the bit justification rule, the 3-bit <PowerMode> field is aligned on the LSB of the <CmdData> field, with all preceding bits filled with zeroes.

Power mode is also accessible through the PMBus interface. See Part II, Section 14.13 for more details.

### 8.7 Reserved for future use (Command Data Types 0110b to 1101b)

### 8.8 AVSBus Status Read/Write (Command Data Type = 1110b)

This type allows the AVSBus Master to read the <AVSBus\_Status> of an AVSBus Slave, or to clear the slave's <AVSBus\_Status> by a write. When writing to this data type, all status bits written with a 1b are cleared. All others remain unchanged. Persistent faults will be immediately re-asserted.

A clear operation for AVSBus shall not affect the corresponding status bits for PMBus status registers. This implies that the bits in <AVSBus\_Status> that reflect PMBus status bit must either be copies of those in bits in their own register or that some form of a mask must be used to present a cleared value to the AVSBus.

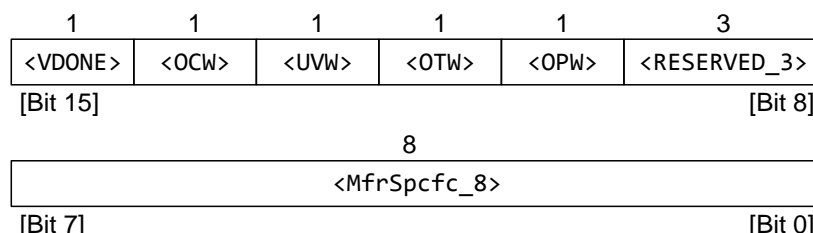
The status consists of 16 bits concatenated in the following order as shown in Figure 16:

- <VDone> - A single bit flag that will be 0b while the rail is off or powering up, it will change to 1b as soon as the voltage has reached the set operating point, and will again transition to 0b when a new target is committed.

The threshold for determining when the output has reached the target is defined by the manufacturer of the AVSBus Slave, and must be clearly described in the product documentation.

- <OCw> - The IOUT\_OC\_WARNING (Output over-current) flag that is found as bit [5] of PMBus command STATUS\_IOUT.

- <UVW> - The VOUT\_UV\_WARNING (Output under-voltage) flag that is found as bit [5] of PMBus command STATUS\_VOUT.
- <OTW> - The IOUT\_OT\_WARNING (Over-temperature) flag that is found as bit [6] of PMBus command STATUS\_TEMPERATURE.
- <OPW> - The POUT\_OP\_WARNING (Output over-power) flag that is found as bit [0] of PMBus command STATUS\_IOUT.
- <Reserved\_3> - Three bits reserved for future use. Until defined this field should be returned as 000b.
- <MfrSpcfc\_8> - Eight bits whose definition is left up to the AVSBus Slave. They must be defined with positive logic: the status is considered active (set) when the value returned is 1b; otherwise it will be returned as 0b.



**Figure 16. AVSBus\_Status\_Bits**

For this data type <Select> is <RailSel>, and <CmdData> is <AVSBus\_Status>. This is the only command in which it is possible to set <Select> to all 1's for a read, making it the only supported broadcast read. The AVSBus Slave simply combines the values of each status bit for all active rails using a predetermined function, and returns the resulting value, effectively providing a quick overview of the entire device.

The function depends on the purpose of each status bit, as follows:

- <VDone> bits from all rails are ANDed together so that the resulting value is set to 1b only when all rails have it set to 1b.
- <OCW> bits from all rails are ORed together so that the resulting value is set to 1b if any any rail has it set to 1b.
- <UVW> bits from all rails are ORed together so that the resulting value is set to 1b if any any rail has it set to 1b.
- <OTW> bits from all rails are ORed together so that the resulting value is set to 1b if any any rail has it set to 1b.
- <OPW> bits from all rails are ORed together so that the resulting value is set to 1b if any any rail has it set to 1b.
- <Reserved\_3> bits are undefined.
- <MfrSpcfc\_8> bits will be combined as determined by the manufacturer. The device documentation must clearly specify the function used.

## 8.9 AVSBus Version Read (Command Data Type = 1111b)

This type allows the AVSBus Master to read the version of AVSBus implemented by an AVSBus Slave. For PMBus 1.3, the <AVSBus\_Version> value is 0000b.

Following the bit justification rule, the 4-bit <AVSBus\_Version> field is aligned on the LSB of the <CmdData> field, with all preceding bits filled with zeroes. There is no selector for AVSBus version. <Select> must be set to all 1's for this command.

### 8.10 Manufacturer-Specific Read/Write (Group 1b, Data Types 0000b to 1111b)

The definition of <Select> and <CmdData> for each one of these data types is specific to each device. In all other regards, these commands follow the same frame structure and protocol rules that apply to Group 0b data types.

The product literature of an AVSBus Slave must clearly identify the manufacturer-specific data types it supports, together with the size of the fields, their encoding and whether they allow Read-only, Write-only or Read/Write commands.

## 9. Communication From The AVSBus Slave To The AVSBus Master

AVSBus devices will never become bus masters for communication with the host. All transfers are initiated by the master. However, a mechanism for AVSBus Slaves to issue an interrupt to the AVSBus Master is supported so that the slave can send timely status information to the master. Implementation of this mechanism is not mandatory for AVSBus Slaves, although support for the status information (<StatusResponse>) is mandatory for 3-wire mode implementations. See Section 5.3 for more information.

When an AVSBus Slave that supports interrupts determines that it needs the AVSBus Master to be aware of any special condition, it can simply signal to the AVSBus Master that it needs a frame to be started. It does this by pulling the AVS\_SData line low while the bus is idle.

It is up to the AVSBus Master to initiate such frame as soon as practical.

## 10. PMBus™ And AVSBus Device Mapping

Figure 16 shows a hypothetical power management IC. This IC has four controllers that can provide a control signal, typically a pulse width modulated (PWM) signal, to four different power stages. As shown, each power stage generates an output voltage but it is possible that two or more of the power stages could be operated as phases of one interleaved multi-phase power stage.

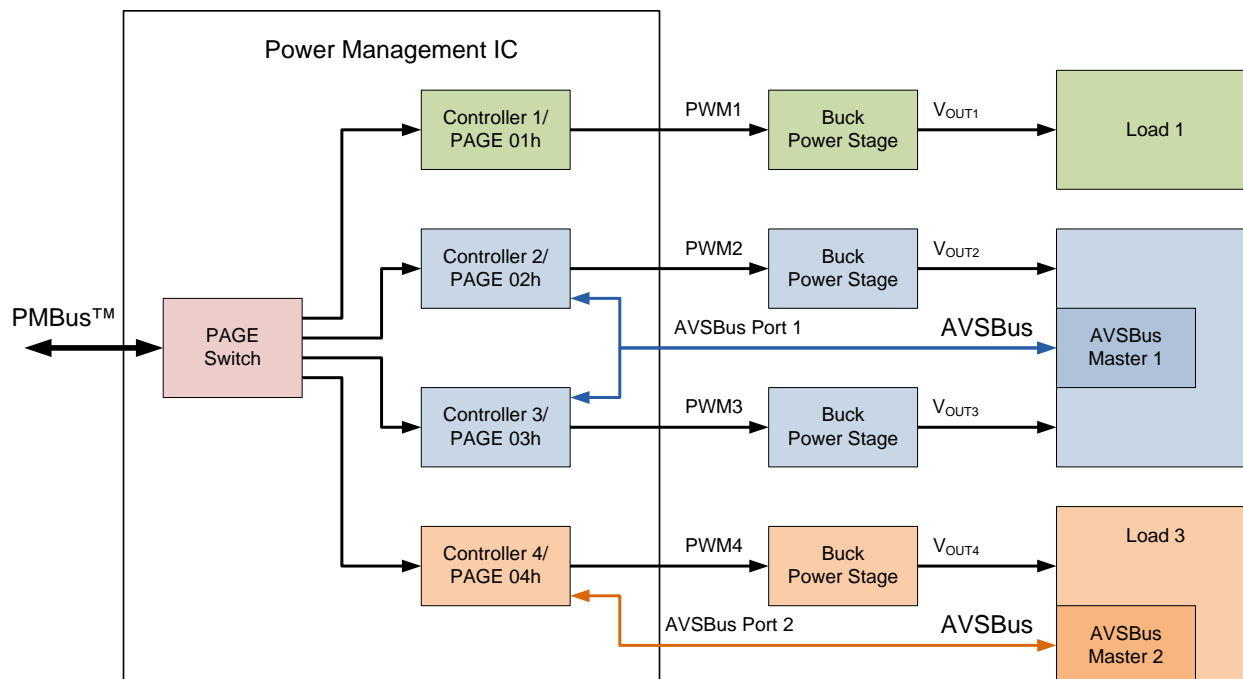
This hypothetical IC also has a PMBus™ interface that uses the PAGE command to individually address the four controllers. The IC also has two AVSBus interface ports. AVSBus Port 1 is routed within the power management IC to controllers 2 and 3. This means AVSBus Master 1 is supporting two rails. AVSBus Port 2 is routed only to controller 4. Controller 1 is controlled only through the PMBus and does not have an AVSBus port connection.

In order to properly design the power system, the system engineer needs to know which of the controllers (and their PAGE numbers) are controlled only by the PMBus interface and which are controlled by both the PMBus and AVSBus interfaces. There could also be, although not shown in this example, outputs that were controlled only by an AVSBus interface. The information describing the connection of internal elements or controllers or power management ICs or device to externally connected PMBus and AVSBus shall be fully described in the product literature.

At this time there is no standard mechanism provided for the details of these connections to be discoverable by either a PMBus system host or an AVSBus master. This capability



may be included in a future release of the PMBus and AVSBus standards. Device manufacturers may provide this capability through manufacturer specific commands.



**Figure 17. Example Power Management IC With Multiple PAGES And Multiple AVSBus Ports**

## 11. Accuracy

Each AVSBus device will specify in its product literature the accuracy with which the target voltage and other parameters can be set and reported.

## **Appendix I. Summary Of Changes**

DISCLAIMER: The section is provided for reference only and for the convenience of the reader. No suggestion, statement or guarantee is made that the description of the changes listed below is sufficient to design a device compliant with this document.

A summary of the changes made in Part III of the PMBus specification from Revision 1.3 to this revision, 1.3.1, is given below. This is not an exact list of every change made between the two documents; rather, it is a summary of the changes deemed significant by the editor.

- Added clarification to Idle Bus Condition (Section 5.5)
- Added clarification to Slave Resynchronization (Section 5.6)
- Added clarification to Bus Timeout (Section 5.7)
- Added clarification to <SlaveAck> (Section 6.7)
- Added clarification to Status Response Frame (Section 7.4)
- Added clarification to Voltage Reset (Command Data Type 0100b) (Section 8.5)
- Added clarification to AVSBus Status Read/Write (Command Data Type = 1110b) (Section 8.8)