PMBus®
Power System Management Protocol Specification
Part III – AVSBus

Revision 1.5
5 Oct 2025

www.powerSIG.org

Filename: PMBus_Specification_Part_III_Rev_1_5_20251005.docx
Last saved: 05 Oct 2025, 14:52

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

REV	DATE	DESCRIPTION	EDITED BY
1.3	17 March 2014	Revision 1.3, First public release of Part III	Robert V. White Embedded Power Labs
1.3.1	13 March 2015	Revision 1.3, Second public release	Robert V. White Embedded Power Labs
1.4	12 Jan 2022	Revision 1.4; Third public release	Robert V. White Embedded Power Labs
1.4.1	18 Apr 2022	Revision 1.4.1, Error correction	Robert V. White Embedded Power Labs
1.5	5 Oct 2025	Revision 1.5, Fourth public release	Robert V. White Embedded Power Labs

Table of Contents

Table of Figures

Table of Tables

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 target.

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 fout 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.

Part IV that defines the use of the PMBus security commands to implement Secure PMBus

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 the previous revision are shown in Appendix I at the end of this document.

1.3 Where To Send Feedback And Comments

Please send all comments by email to: 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. Unless otherwise specified the latest version of the listed documents is applicable.

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

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

[A03] PMBus Power System Management Protocol, Part IV, PMBus Security Commands And Processes

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.

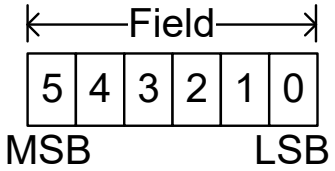
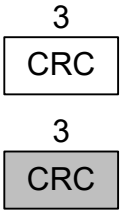
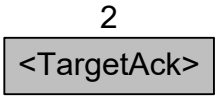
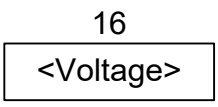


Figure 1. Bit Order Within A Field

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
0	A rectangle with no shading indicates data sent from the host (bus controller) to the target device.

Symbol	Meaning
	A rectangle with a shaded interior indicates data sent from the target device to the bus controller.
	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. Shown here is the 3-bit CRC used for verifying the integrity of a transfer. It can be sent by the controller or the target, so it may be found shaded or clear.
	The <StartCode> sent from a bus controller device to initiate a transfer is the 2-bit code 01b.
	When targets send acknowledgement back to the bus controller, they send a 2-bit code, represented by <TargetAck>.
	Data fields are identified by a descriptive word, with the number of bits indicated above, as shown here for a 16-bit voltage value.

3.5 Abbreviations, Acronyms And Definitions

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

Term	Definition
ACK, <TargetAck>	Acknowledge. The response from a target 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 Controller. 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.
Controller	A controller is a device that issues commands, generates the clock, and terminates the transfer.

Term	Definition
CData	Controller Data sent via the output port AVS_CData of an AVSBus Controller for sending bits to its AVSBus Target, which is connected to the input port AVS_TData of the target device. See also TData.
Command Data, <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.
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 Controller. 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
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.

Term	Definition
<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.
<StartCode>	A 2-bit field used to mark the beginning of a Controller sub-frame.
<StatusResponse>	A 5-bit field composed by concatenating multiple flags. There is no associated <CmdDataType>. The Status Response field will be sent by the AVSBus Target twice in every frame, as explained in Section 9. It should not be confused with <AVSBus_Status>.
Target	A target is a device that is receiving or responding to a command.
TData	Target Data sent via the output port AVS_TData of an AVSBus Target for sending bits to its AVSBus Controller, which is connected to the input port AVS_CData of the controller device. See also CData.
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. 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 Targets 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 Resynchonization 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.

When AVSBus is used in conjunction with PMBus, it is PMBus that determines if the AVSBus Controller gets control over the settings of the AVSBus Target, and can take that control away from the Controller-Target. However, regardless of whether control is given to the AVSBus Controller or not, PMBus can always read data, and the AVSBus Controller can always read data.

In fact, the AVSBus Controller 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 Target responding with the <TargetAck> 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 target for controlling voltage scaling. The three wires used for communication are AVS_Clock, AVS_CData, and AVS_TData.

- AVS_CData is driven by the AVSBus Controller and carries data to the target,
- AVS_TData is driven by an AVSBus Target and carries data to the controller, and
- AVS_Clock is driven by the controller and clocks data for both the AVS_CData and AVS_TData lines.

An optional 2-wire unidirectional implementation is allowed using only AVS_Clock and AVS_CData. In this limited implementation, the AVSBus Controller will never receive acknowledgments, status or data back from a target device.

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

- AVS_CData and AVS_TData 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 Target and its controlling AVSBus Controller.

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

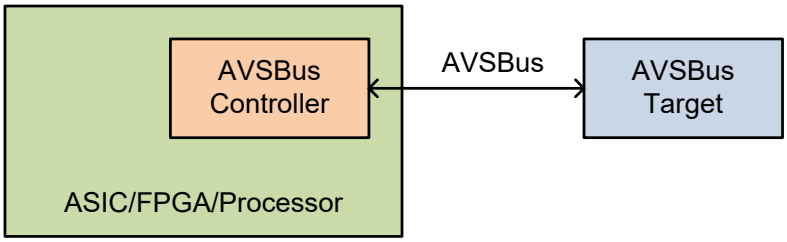


Figure 2. Example AVSBus Topology With One Controller And One Target

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

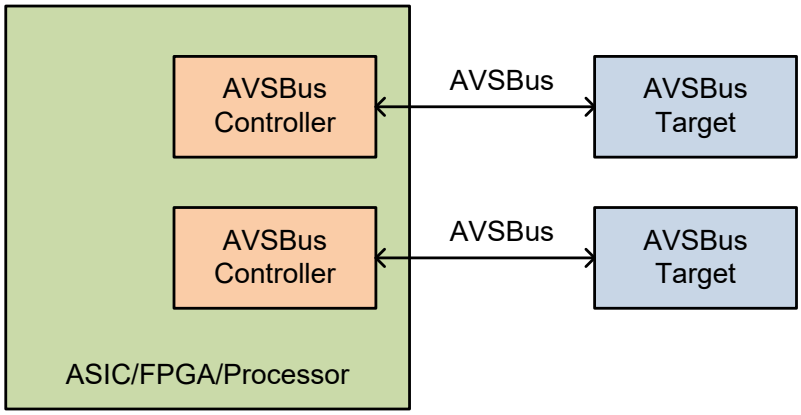


Figure 3. Example AVSBus Topology With Multiple Links

5.2 Operation

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

1. The AVSBus Controller must initiate all data transfers.

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

   The controller must guarantee that AVS_CData 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_CData is not allowed to be at a logic value ‘0’ when the clock starts for a frame.

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

   In general the target must guarantee that AVS_TData 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 target is not sending data

to the controller. An exception to this rule is that the AVSBus Target will set AVS_TData to a logic value '0' when it needs to alert the AVSBus Controller 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 controller and target 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 controller to receive acknowledgements from targets 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_TData.

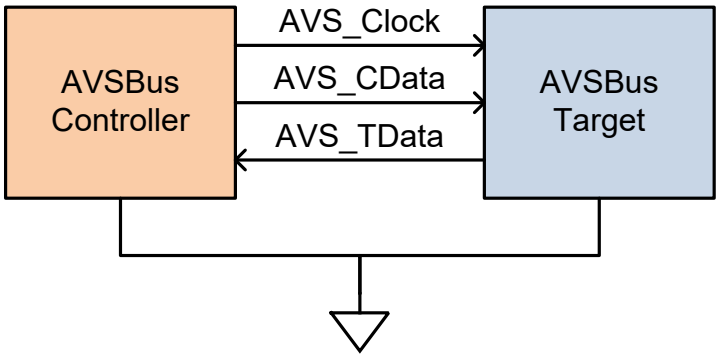


Figure 4. AVSBus 3-Wire Mode Connections

5.3.2 2-Wire Mode

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

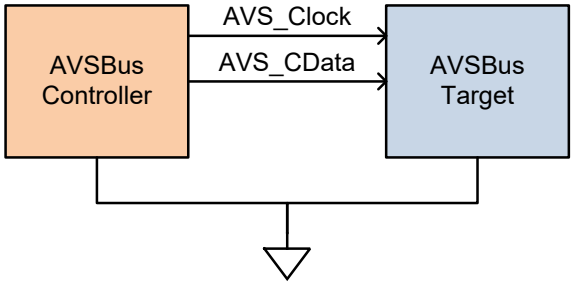


Figure 5. AVSBus 2-Wire Mode Connections

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_CData is a “Don’t care” to the target, but it is required that the line be kept high to enable proper detection of <StartCode> when the clock resumes.

   The state of AVS_TData is relevant to the controller even when the clock is suspended. Considering that the target has no control over when the controller may start providing a clock, a target that wants to alert the controller to the fact that there has been an important status change will simply change AVS_TData from its default state (Logic 1) to a signaling state (Logic 0), and keep the signaling state until the controller initiates a frame. Except for this interrupt scenario, the target 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_CData must be held high (Logic 1) if the AVSBus Controller 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_TData should be held high (Logic 1) when no frames are being transmitted by the AVSBus Target. Just like when the clock is suspended, a target that wants to alert the controller to the fact that there has been an important status change will simply change AVS_TData from its default state (Logic 1) to a signaling state (Logic 0), and keep the signaling state until the controller initiates a frame.

To effect this, the AVSBus Target must be designed to monitor AVS_CData continuously, not just for the purpose of receiving a new frame, but to guarantee that the state of AVS_TData does not change between the 0b and the 1b of the 01b sequence for starting a frame. Conversely, the AVSBus Controller must be designed to ignore the value of AVS_TData 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 Target Resynchronization

An AVS Target 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_CData is held high will cause a target to resynchronize its communication interface, and wait for the next <Start_Code>.

For the resynchronization mechanism to work correctly, the AVSBus Target 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 controller 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 controller to resynchronize the target at startup, and to do so periodically afterwards.

5.7 Bus Timeout

AVSBus does not require that targets implement bus timeout functions. Such a feature would be useful for detecting when AVS_Clock has stalled so that the target can take some protective action, but it would impose additional burdens on the target 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 target 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 target ready to properly detect a future <Start_Code>. Otherwise, when the controller recovers and tries to start sending a brand new frame, the target 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 controllers do not simply rely on targets' implementation of Bus Timeout. Instead, they should trigger the Target 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 controller. For periods with no bus activity, the controller 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 Controller always provides AVS_Clock and AVS_CData (Controller Data), and it may or may not have an input for AVS_TData (Target 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_CData lines are always driven by the controller and AVS_TData is always driven by the target, 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 Target-to-Controller 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 Controller and the Target.

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

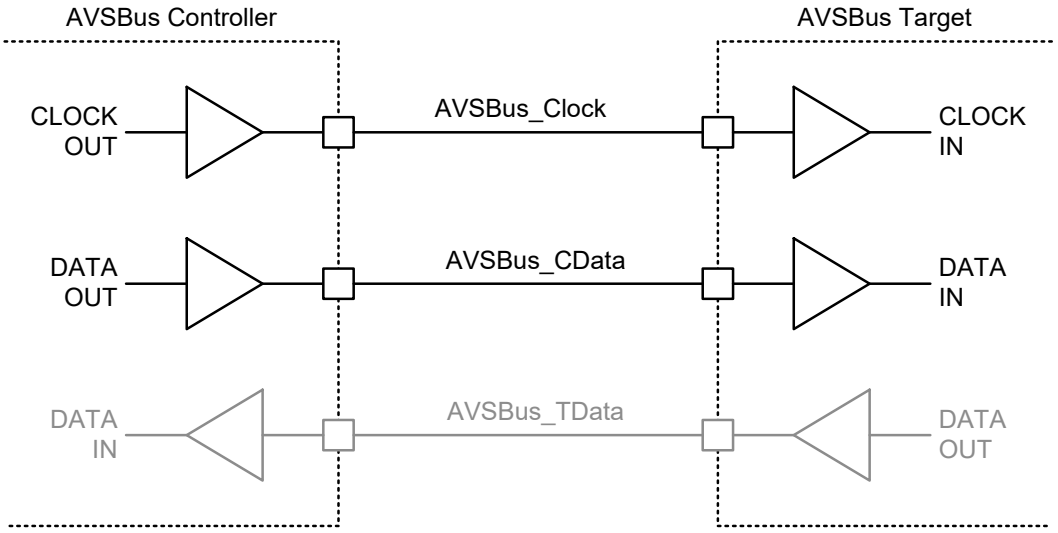


Figure 6. Timing Diagrams Reference Schematic

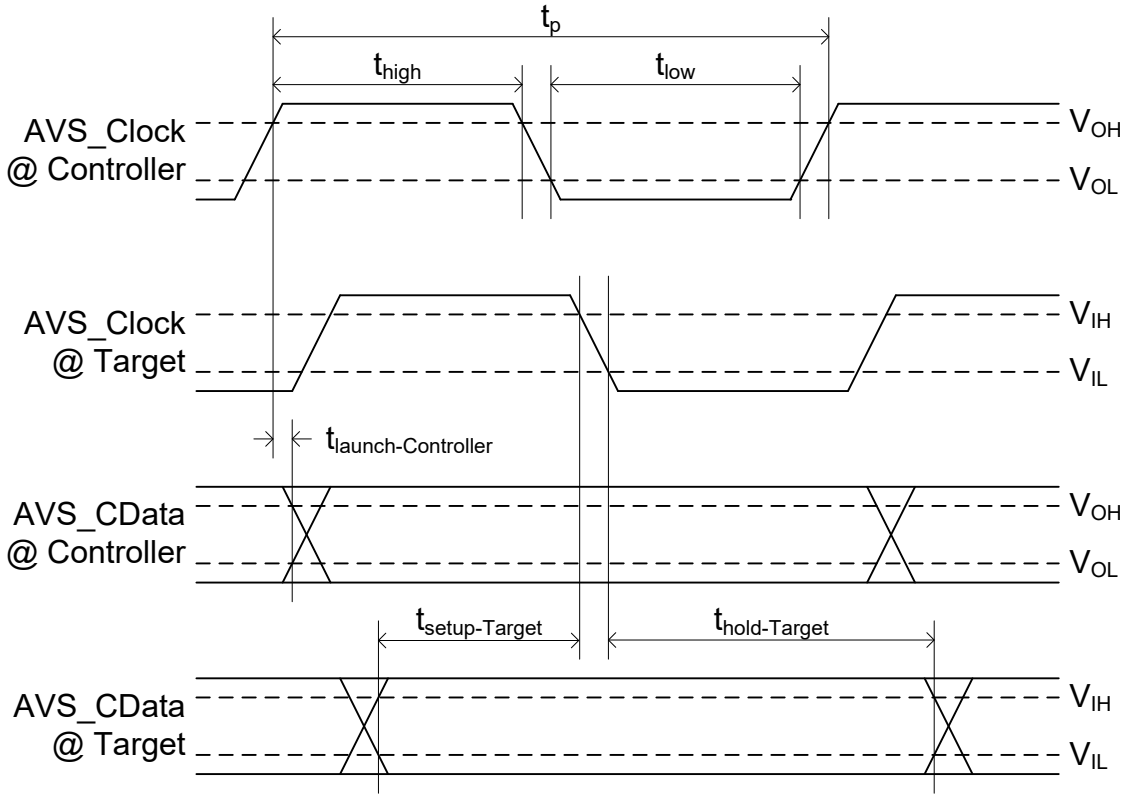


Figure 7. Timing diagram for AVS_CData

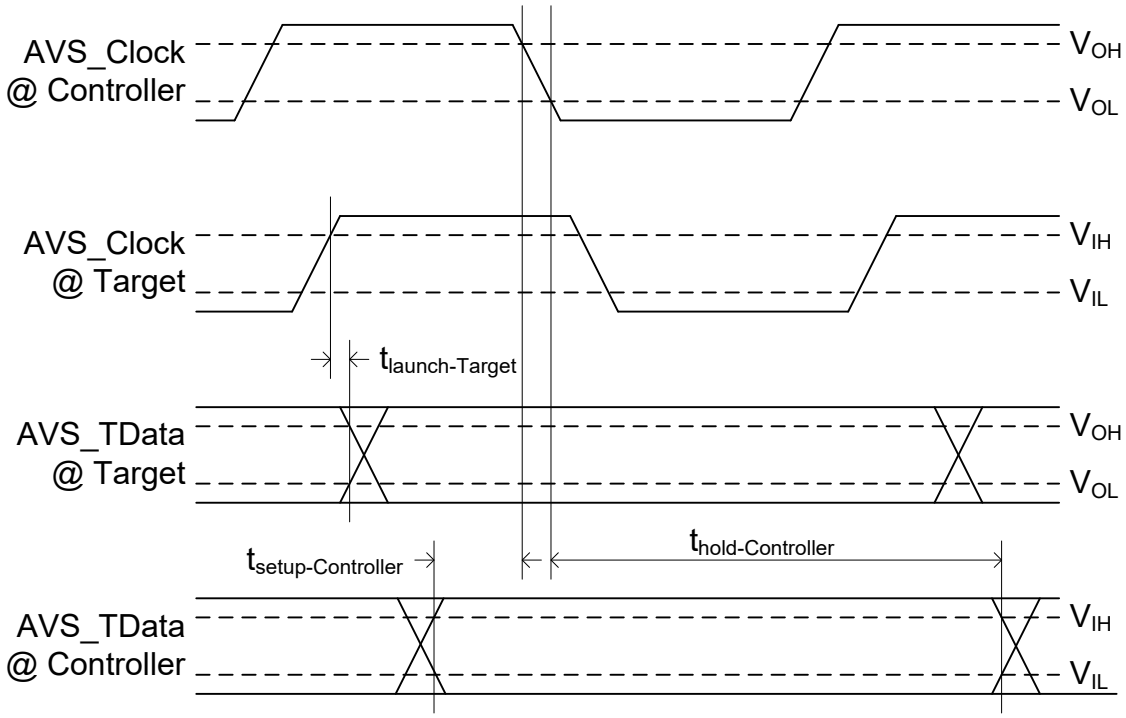


Figure 8. Timing diagram for AVS_TData:
Controller Capturing Data On Clock Falling Edge

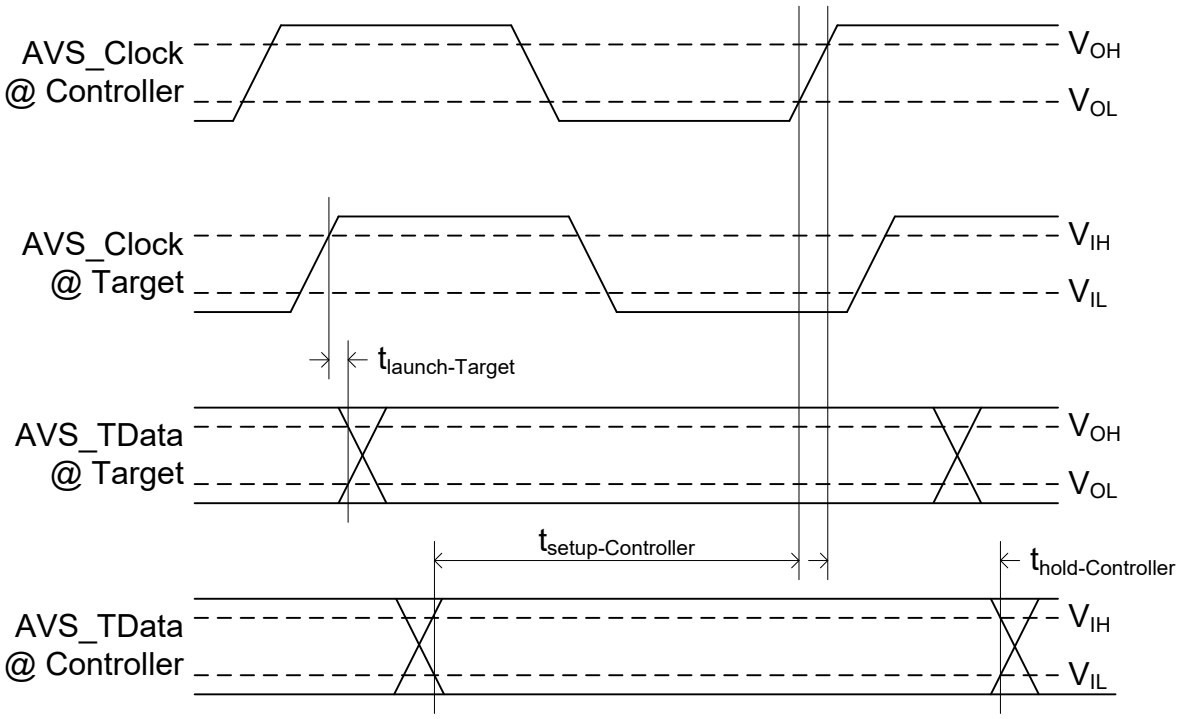


Figure 9. Timing diagram for AVS_TData:
Controller Capturing Data On Clock Rising Edge

All timing measurements are made at the pads of the respective devices.

The timing diagrams, Figure 7 and Figure 8, show the effects of (but do not specifically define) the unspecified propagation delays of the clock and data from the Controller to the Target and of the data from the Target to the Controller. These figures also show the unspecified rise and fall times of the clock and data. The propagation delays and rise and fall times are unspecified because these are dependent on the system design including, but not limited to, the length of the clock and data traces and the capacitance of those traces.

Table 3. Electrical Characteristics

Parameter	Description	Min.	Typ.	Max.	Units
t_p	Period for active clock	20			ns
				10	µs
t_{high}	Duration of the high-phase of clock	40			% t_p
t_{low}	Duration of the low-phase of clock	30			% t_p
Controller-to-Target data path					
$t_{launch\text{-}Controller}$	Time from rising clock edge in Controller to data-out transition at Controller's data-out port.	0		15	% t_p
$t_{setup\text{-}Target}$	Time from data-out edge in Controller to capture clock edge in Target.	10			% t_p
$t_{hold\text{-}Target}$	Time from capture clock edge in Target to data-out edge in Controller (for next bit).	10			% t_p
Target-to-Controller data path					
$t_{launch\text{-}Target}$	Time from rising edge of clock at the Target until data out transition at the Target	0		15	% t_p
$t_{setup\text{-}Controller}$	Time from data-out edge in Target to capture clock edge in Controller.	0			% t_p
$t_{hold\text{-}Controller}$	Time from capture clock edge in Controller to data-out edge in Target (for next bit).	20			% t_p

Controller devices must specify their available t_p and clock high duty cycle (both minimum and maximum) in the product literature.

Target devices must specify their minimum t_p in the product literature.

Note that the clock used by the target is a delayed version of the clock in the controller. For that reason, launching data from the target starts later than launching from the controller, and relatively speaking, capturing by the controller comes earlier. If the

propagation delay is large on a given board, it may be necessary to increase t_{high} to compensate and give more time for the data to go from the target to the controller.

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

Table 4. Electrical Drive Levels

Parameter	Description	Limits		Units	Comments
		Min	Max		
V_{DD}	AVS Controller bus voltage	0.9	3.63	V	1 V to 3.3 V ±10%
V_{IL}	AVSBus signal Input Low voltage		40% V_{DD}	V	
V_{IH}	AVSBus signal Input High voltage	60% V_{DD}		V	
V_{OL}	AVSBus signal Output Low voltage		20% V_{DD}	V	
V_{OH}	AVSBus signal Output High voltage	80% V_{DD}		V	
$I_{LEAK-PIN}$	Input Leakage per pin	-10	+10	μA	

6. Protocol rules

For the rules outlined below there is no distinction between controller and target 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

Target devices capture data on the falling edge of the clock.

Controllers may capture data from a target device on either the falling edge of the clock or the next rising edge of the clock. It is the responsibility of the system designer to assure that all data setup and hold time specifications are satisfied.

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 Target 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 Target that receives a command for an unknown resource will respond with a special `<TargetAck>` 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 Target that receives a read or a write frame for one of its resources which is not available will respond with a special `<TargetAck>` 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_CData having a default state of '1' sets the conditions for the detection by a target 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 <TargetAck>

Whenever an AVSBus Controller sends a command, the AVSBus Target will send back a response frame which carries a 2-bit acknowledgment.

The two bits encode the following four states, shown here in order of precedence:

- 10b: Bad CRC. Any apparent errors like invalid selector or resource not available are irrelevant, as the frame is unreliable.
  No action is taken.

- 11b: Good CRC, but the command cannot be executed because of one of the following conditions:
  - Invalid selector (e.g. unknown resource, or use of the broadcast selector when not supported).
  - Invalid data type (e.g. non-existent command, including those currently reserved).
  - Incorrect data (e.g. out of range data value).
  - 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, or to use Write and Hold when not supported by the target).

  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). If a Write and Commit command applies to the target, but there is no value waiting for commit, that is a special case of the resource not being available.
- 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 target from carrying out a command. It is up to the device manufacturer to provide means for the controller 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.

It should be noted that a value that is inherently valid but may be unworkable for the device could be handled differently depending on the design of the target. For instance, a target that is designed to clamp the voltage to a pre-defined setting would react to a voltage setting that is too high by clamping, and will return a `<TargetAck>` of `00b` since effectively the command was executed: The voltage changed to the clamp value. Alternatively, a device that does not support clamping will reject the command altogether, resulting on a `<TargetAck>` of `01b`.

There may be cases in which a 01b value may not tell the whole story, as in the example above with voltage clamping. 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 Controller initiates a frame, the target must send back a response that can be used by the controller to assess the general condition of the target. This is a 5-bit field formed like this:

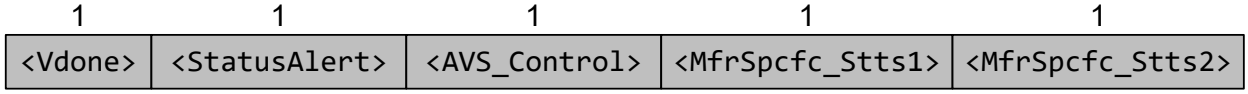


Figure 10. `<StatusResponse>` Composition

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 Controller 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 Target'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 Target'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:

CRC(x) = x^0 + x^1 + x^3

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

- An AVSBus Controller that receives a frame with an incorrect <CRC> from a target will discard the data. It should initiate the transfer once more.
- An AVSBus Target that receives a frame with an incorrect <CRC> from a controller will send the corresponding <TargetAck> 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 Target 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 Target perform a range check for each voltage write. If the value is larger than the maximum or smaller than the minimum provided, the target will respond with the <TargetAck> 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 Controller 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 Controller 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 Target 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 Controller segment (sent over AVS_CData), which always begins with <StartCode>, and the Target segment (sent over AVS_TData).

There is a causal relationship between sub-frames:

- A Controller Write sub-frame is immediately followed by a Target Write sub-frame.
- A Controller Read sub-frame is immediately followed by a Target 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 Controller sub-frame, carried in the AVS_CData line:

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

As well as the Target sub-frame, carried in the AVS_TData line:

- <TargetAck>
- 0b
- <StatusResponse>
- <Reserved_21> (Always all 1's)
- <CRC>

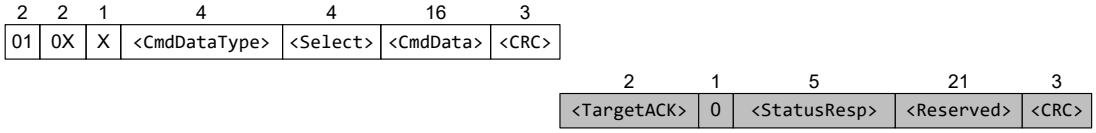


Figure 11. Write Frame Structure

7.2 The Read Frame

This type of frame contains the following fields in the Controller sub-frame, carried in the AVS_CData line:

- <StartCode>
- <Cmd> (Always 11b in read frames)
- <CmdGroup>
- <CmdDataType>
- <Select>
- <Reserved_16> (Always all 1's)
- <CRC>

As well as the Target sub-frame, carried in the AVS_TData line:

- <TargetAck>
- 0b
- <StatusResponse>
- <CmdData>
- <Reserved_5> (Always all 1's)
- <CRC>

Figure 12. 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 controller requires: • 11b: Read data. • 10b: Reserved. • 01b: Write data and hold, but do not commit (leave pending). • 00b: Write data and commit all pending writes.
<CmdGroup>	1	Qualifier to distinguish between two groups of data types: • 0b for fully defined AVSBus data types. • 1b for manufacturer-specific data types.
<CmdDataType>	4	Type of data to which <Cmd> applies. For <CmdGroup> = 0b, the data types are: • 0000b: Target rail voltage. • 0001b: Target rail Vout transition rate. • 0010b: Rail current (read only). • 0011b: Rail temperature (read only). • 0100b: Reset rail voltage to default value (write only). • 0101b: Rail power mode. • 0110b to 1101b: Reserved command data types. • 1110b: AVSBus Status • 1111b: AVSBus Version For <CmdGroup> = 1b the definition of the data types is found in the device's product literature.

Field Name	Size (bits)	Description
<Select>	4	Selector field to differentiate between instances of a command data type on a device. • For <CmdGroup> = 0b, this is a rail selector: <RailSel>. • For <CmdGroup> = 1b, this is left up to the device manufacturer to specify. 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.
<TargetAck>	2	Response from a target.
<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 Controller will continually send frames back to back. This is what such a sequence would look like:

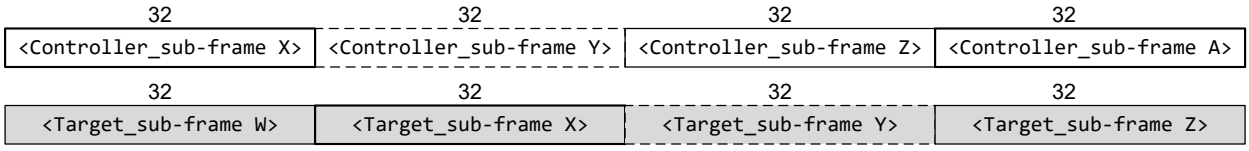


Figure 13. Maximum Throughput

Notice that the two pieces of Frame_X 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 Controller initiated <Controller_sub-frame_X> while the AVSBus Target was sending <Target_sub-frame_W>.

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

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

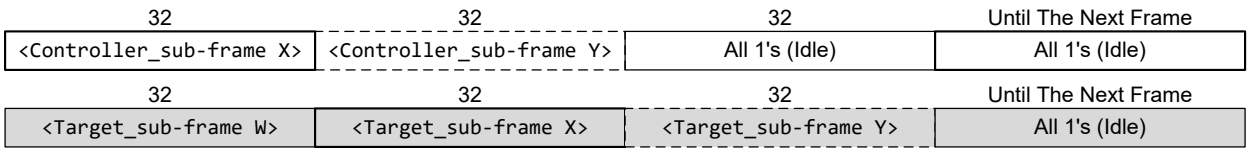


Figure 14. Last Frame In A Sequence.

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 Controller. With the frame definitions we have so far, there would be no alternative for the AVSBus Target but to keep AVS_TData high while it receives the first controller sub-frame in a sequence. However, there is a special type of target sub-frame created to take advantage of that bandwidth: the Status Response Frame.

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

The Status Response Frame is structured as shown in Figure 15 in the context of a generic controller 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>.

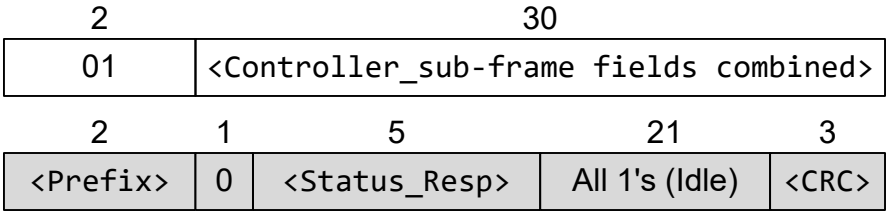


Figure 15. Status Response Frame

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

- If the target signals an interrupt by lowering AVS_TData prior to the appearance of <StartCode>, it would naturally be expected to be 00b, providing continuity.
- If the target did not signal an interrupt, AVS_TData 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 Target may not be ready for the <StartCode>, it is necessary that the AVSBus Controller be designed to watch out for the AVS_TData 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 target to a command from the controller, with two differences:

- <Prefix> is replaced with <TargetAck> when the target is responding to a controller command.
- The all 1's idle field corresponds to the rest of the target'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 16.

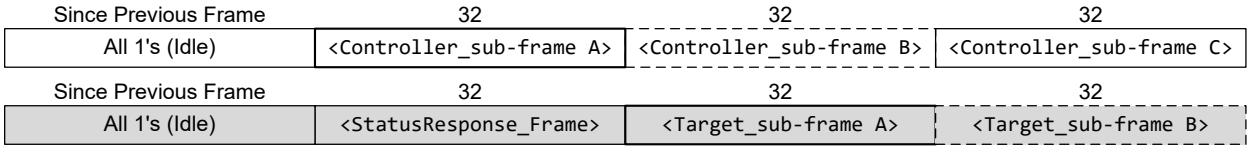


Figure 16. First Frame In A Sequence

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 Controller to read or write a new voltage target for a rail on an AVSBus Target. 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 Target may respond with the <TargetAck> for incorrect data and do nothing, or take action by clamping the voltage and responding with the <TargetAck> that indicates the action was taken.

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

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

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 Target 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 Target manufacturer and given in the product literature.

8.3 Current Read (Command Data Type = 0010b)

This type allows the AVSBus Controller to read the current for a rail on an AVSBus Target. 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 Target'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 Controller to read the temperature for a rail on an AVSBus Target. 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 Target'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 Controller to force a "Predetermined Value" for an AVSBus Target 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, or 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 Target.

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 target 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 Controller 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 Controller to set the mode of operation for the output of an AVSBus Target 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 Controller to read the <AVSBus_Status> of an AVSBus Target, or to clear the target'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 17:

- <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 Target, 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 Target. 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.

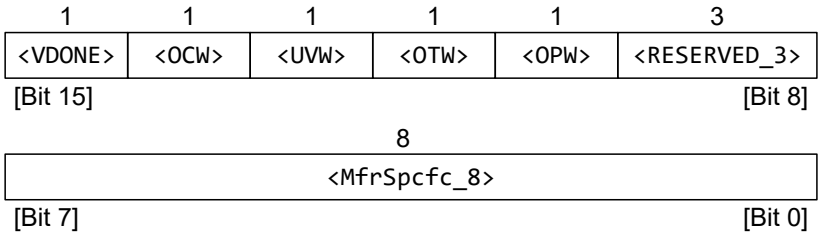


Figure 17. AVSBus_Status_Bits

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 Target 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 Controller to read the version of AVSBus implemented by an AVSBus Target. For PMBus 1.4, the <AVSBus_Version> value is 0001b.

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 Target 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 Target To The AVSBus Controller

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

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

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

10. PMBus And AVSBus Device Mapping

To illustrate how a device with both a PMBus interface and AVSBus ports works in an arrangement with multiple output voltages a hypothetical configuration is shown in Figure 18. Note that Figure 18 is meant to be an example of a possible configuration and is not in any way to be considered definitive.

The power management IC (PMIC) in Figure 18 has four power stage managers that 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 PMIC also has a PMBus interface that uses the PAGE command to individually address the four power stage managers. The PMIC also has two AVSBus ports. AVSBus Port 1 is routed within the PMIC to power stage managers 2 and 3. This means AVSBus Controller 1 is supporting two rails. AVSBus Port 2 is routed only to power stage manager 4. Power stage manager 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 power stage managers (and their PAGE numbers) are controlled only by the PMBus interface and which are controlled by both PMBus and AVSBus. There could also be, although not shown in this example, outputs that are controlled only by an AVSBus port. How PMBus interfaces and AVSBus ports are connected to and interact with power stage managers or other internal functions of the PMIC 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 controller or an AVSBus controller. 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.

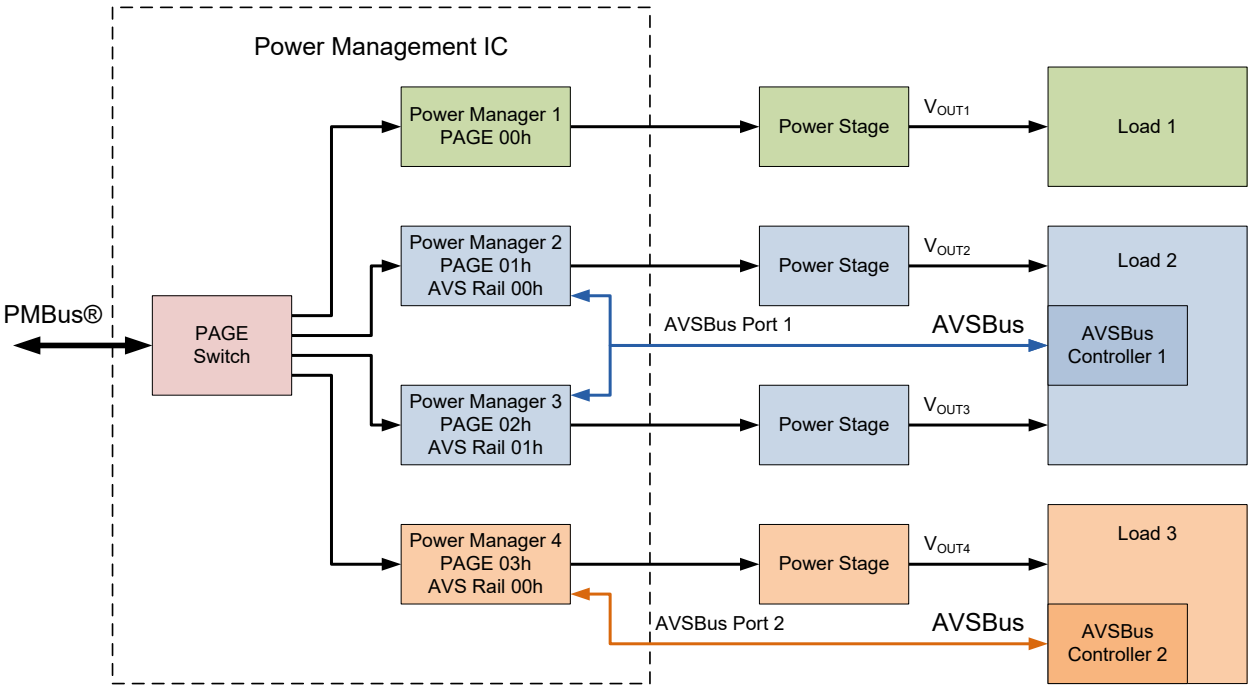


Figure 18. Example Power Management IC With Multiple PAGEs And Multiple AVSBus Ports

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 1.4.1 to this revision, 1.5 is given below.

- Section 6.2: Updated to allow the controller to latch data from a target device at any time from the falling to the rising edge of the clock at the controller.
- Figure 8: Slight update to the figure. Caption updated to indicate this is the case for the controller capturing data on the falling edge of the clock.
- Figure 9: New figure to show the timing for the case when the controller captures data from a target device on the rising edge of the clock.
- Figures 9 through 17 were renumbered to Figures 10 through 18 as the result of adding the new Figure 9.