



Powering Your Ideas.™

PMBus™ Implementers Forum

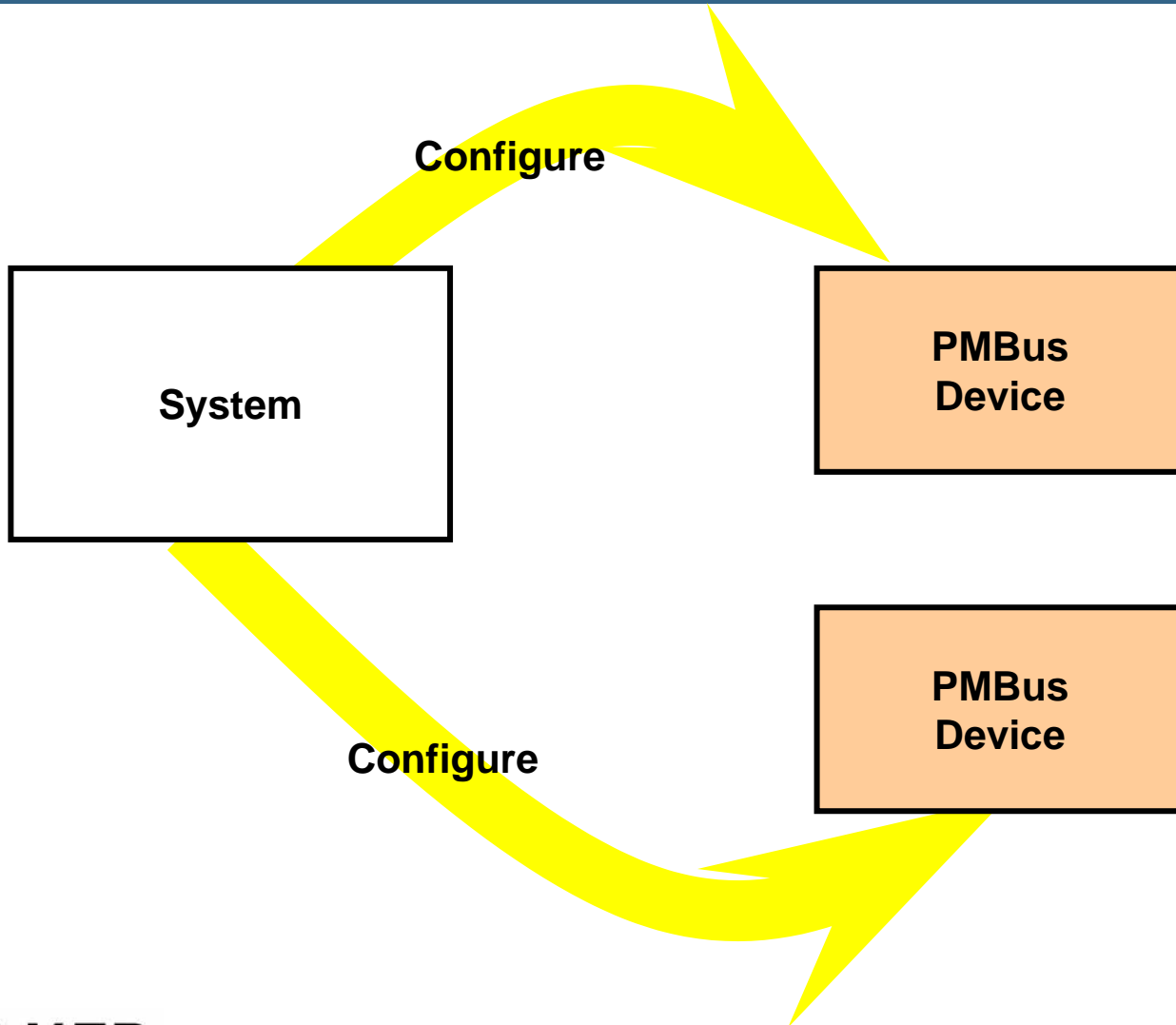
Kenneth W. Fernald
Chief Technical Officer
Zilker Labs, Inc.

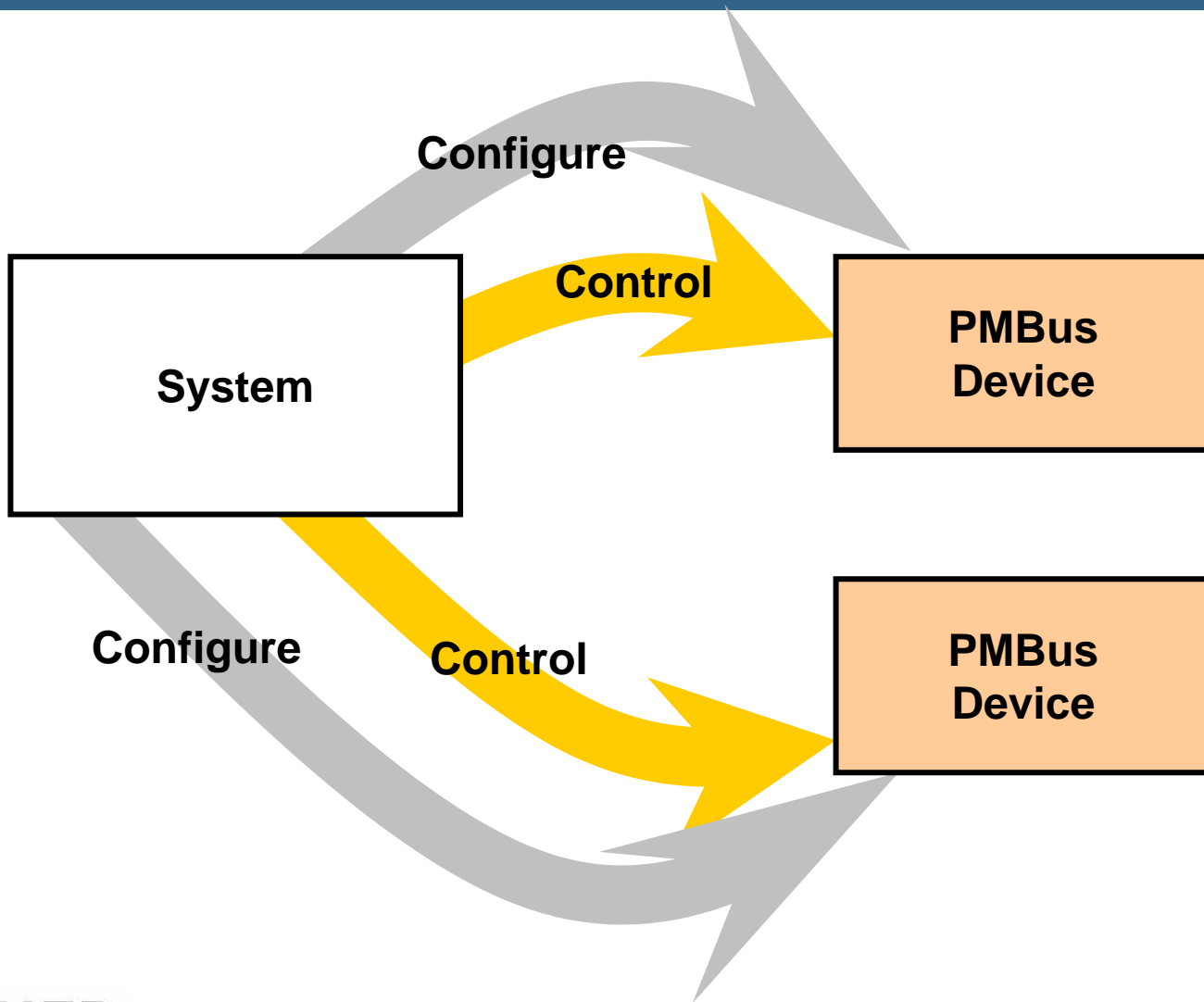
Presentation Overview

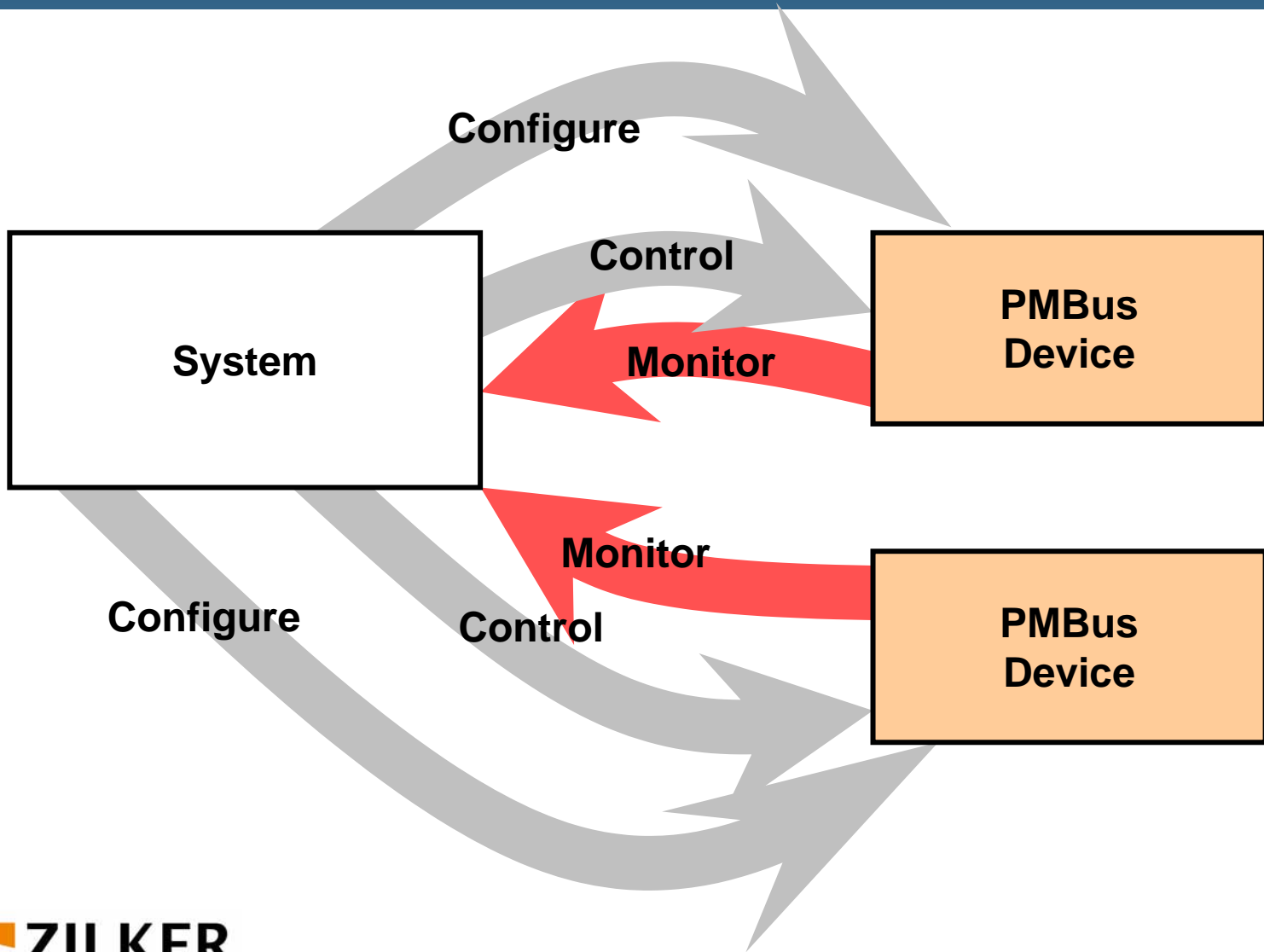
- What Is PMBus™?
- SMBus/I2C
- PMBus basics
- Using PMBus in the lab
- Implementing PMBus
- Command language overview
- Data formats
- Setting the output voltage
- On/Off control
- Sequencing
- Status reporting
- Fault management and reporting
- Monitoring voltage, current and temperature

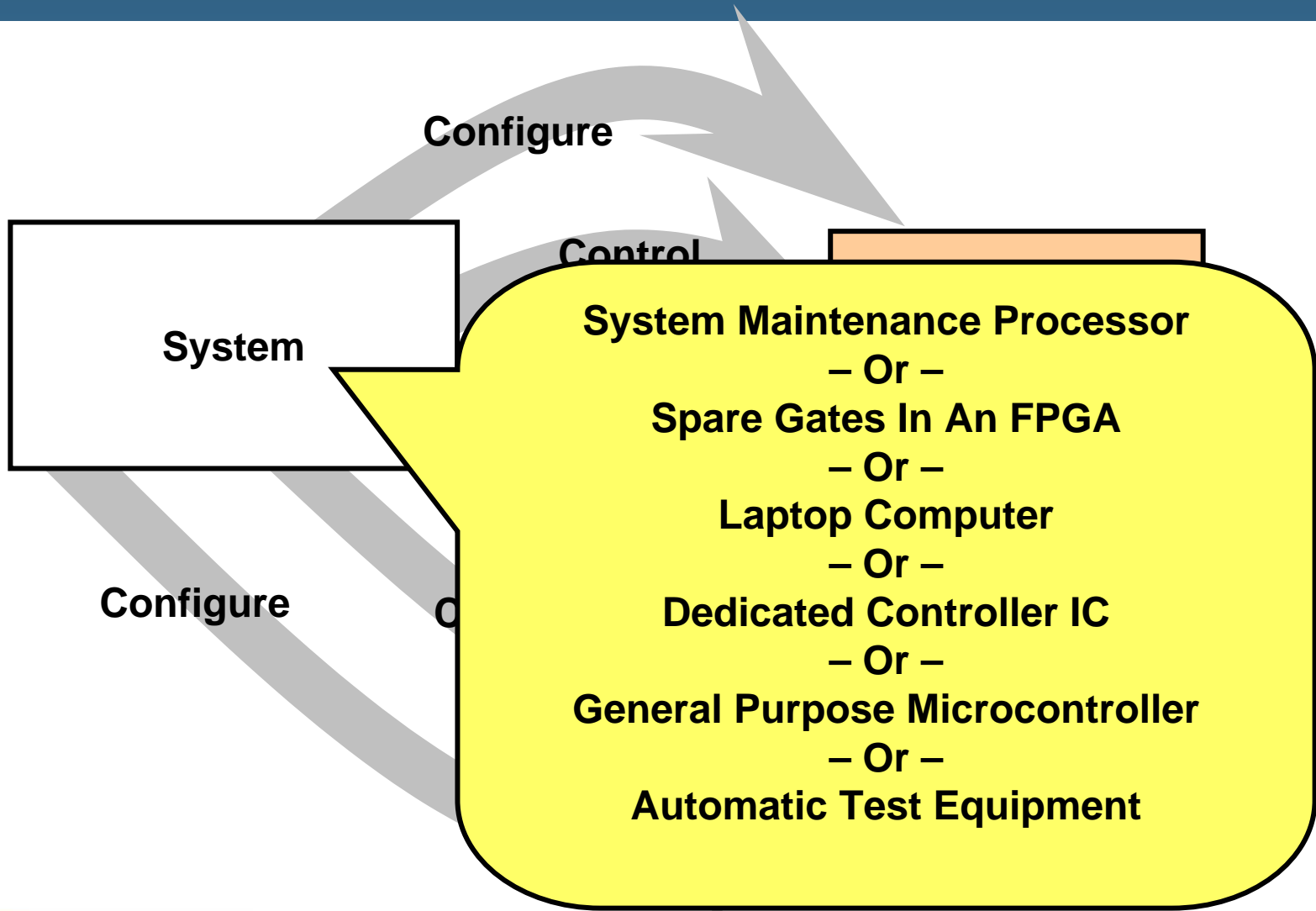
What Is PMBus?

A standard way
to communicate
with power converters
over a digital
communications bus









PMBus Is An Open Standard

- Owned by the System Management Interface Forum (SM-IF)
 - SM-IF membership is open to all
- Royalty free
- Released specifications freely available
- Works with all types of power converters
 - AC-DC power supplies
 - Isolated DC-DC and bus converters
 - Non-isolated point-of-load converters
 - Microprocessor power converters
 - Temperature sensors, fan controllers, monitors, etc.

PMBus: What It Is Not

- Not a product or product line
- Not a standard for a power supplies or DC-DC converters
 - No form factor, pin out, efficiency, etc.
 - Alliances like POLA and DOSA will define
- No converter-to-converter communication
 - Such as current share and analog voltage tracking
 - Left to the IC and power supply manufacturers
 - Including these would inhibit future innovation
 - PMBus devices should allow multi-master operation to allow converter-to-converter communications

Some Basic PMBus Requirements

- PMBus devices must start up safely without bus communication
- PMBus devices can be used with or without a power system manager/controller
- PMBus devices support “set and forget”
 - Can be programmed once at time of manufacture
 - Then operate without bus communication
- Defaults from either/or
 - Non-volatile memory
 - Pin programming
 - Hard-coded constants

Who Created PMBus?

PMBus Founders And Supporters

Founders

Artesyn Technologies

Emerson/Astec

Texas Instruments

Intersil

Microchip Technology

Summit Microelectronics

Volterra Semiconductor

Zilker Labs

Supporters



System Management Interface Forum, Inc.



System Management
Interface Forum

**SM-IF Membership
Open To Any And All**



Smart Battery System
Implementers Forum



Power Management Bus
Implementers Forum

www.powerSIG.org

Specification Structure

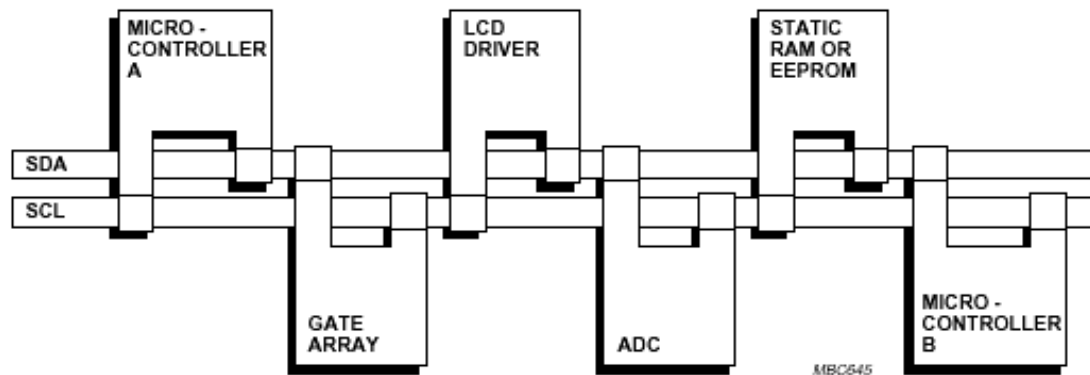
- Part I – Physical layer and transport
 - Physical bus details
 - Discrete signals
 - Electrical levels
- Part II – Command language
 - Commands
 - Data formats
 - Control, monitoring
 - Fault management
 - Status reporting
 - Information storage

PMBus vs SMBus vs I2C

- PMBus is based on SMBus
 - SMBus = System Management Bus
- SMBus is an existing standard developed initially for Smart Battery management
- Based on I²C

I2C protocol – Background

- Inter-Integrated Circuit Protocol
- Low-bandwidth, short-distance, two-wire interface for communication amongst ICs and peripherals
- Originally developed by Philips for TV circuits
- Many devices are available that have hardware interfaces compatible with this protocol



I2C Features

- Only two bus lines are required
 - The SDA(for data) and SCL(for clock)
- Simple master-slave relation amongst devices (either can be receiver or transmitter)
- Multi-bus master collision detection and arbitration is supported
- Serial, 8-bit oriented, bi-directional data transfer up to 100 kbit/s (400kbit/s and 3.4Mbit/s also supported)

I2C Benefits

- Simple 2-wire serial communication
- Reduces interconnection on the ICs (fewer pins) and hence cheaper PCBs
- Easy fault diagnosis
- Completely integrated I2C-bus protocol eliminates the need for address decoders
- Multi-master capability of the I2C-bus allows rapid configuration, testing, and debug
 - An external bus host/snooper can be attached without introducing contention
 - Can simply application-level bus host (no need for debug support)
- ICs can be added and removed from the system without affecting other ICs on the bus

The I2C-Bus

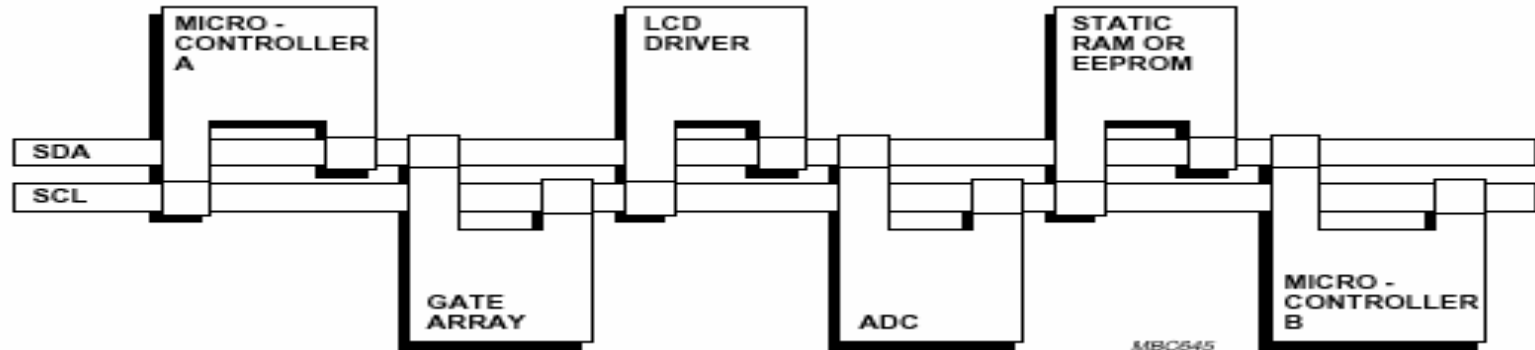
- Two wires SDA(serial data) and SCL(serial clock) carry information
- Each device is recognized by a unique address and can operate as either receiver or transmitter
- Devices have master-slave relation
- I2C bus is a multi-master bus
- Generation of clock signals in the I2C bus is always the responsibility of the master devices

Key terminology

- Transmitter: The device which sends data to the bus
- Receiver: The device which receives data from the bus
- Master: The device which initiates a transfer, generates clock signals and terminates a transfer
- Slave: The device addressed by a master

- Multi-master: More than one master can attempt to control the bus at the same time without corrupting the message
- Arbitration: Procedure to ensure that, if more than one master simultaneously tries to control the bus, only one is allowed to do so and the winning message is not corrupted
- Synchronization: Procedure to synchronize the clock signals of two or more devices

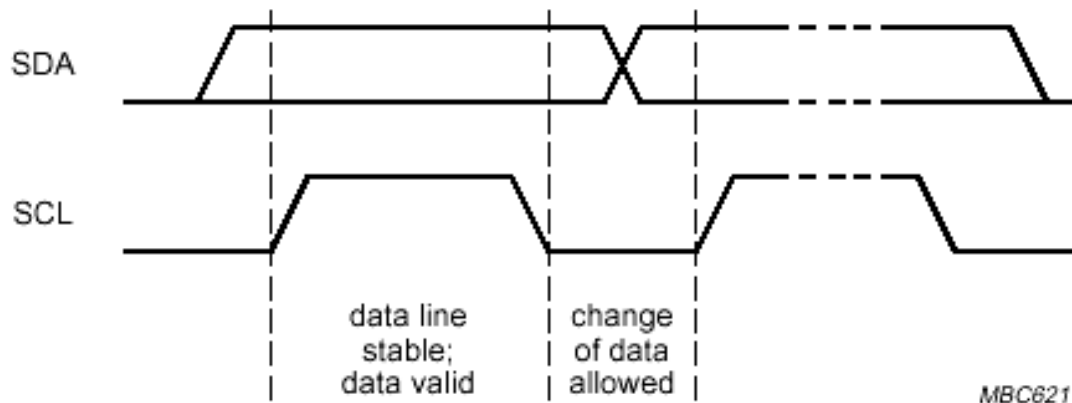
Key Steps: Overview



- Suppose Micro-controller A wants to send info to micro-controller B
 1. A (master) addresses B (slave)
 2. A (master-transmitter) sends data to B (slave-receiver)
 3. A terminates the transfer
- Suppose Micro-controller A wants to receive info from micro-controller B
 1. A (master) addresses B (slave)
 2. A (master-receiver) receives data from B (slave-transmitter)
 3. A terminates the transfer

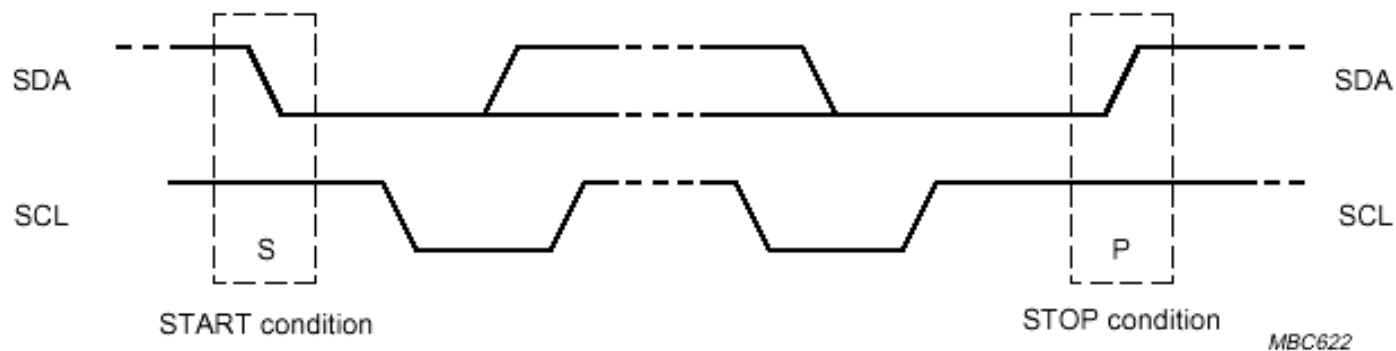
I2C Bit-Transfer

- One clock pulse is generated for each data bit that is transferred
- Data Validity
 - The data on the SDA line must be stable during the HIGH(1) period of the clock. The data line(SDA) can change data only when the clock signal (SCL) is LOW(0)



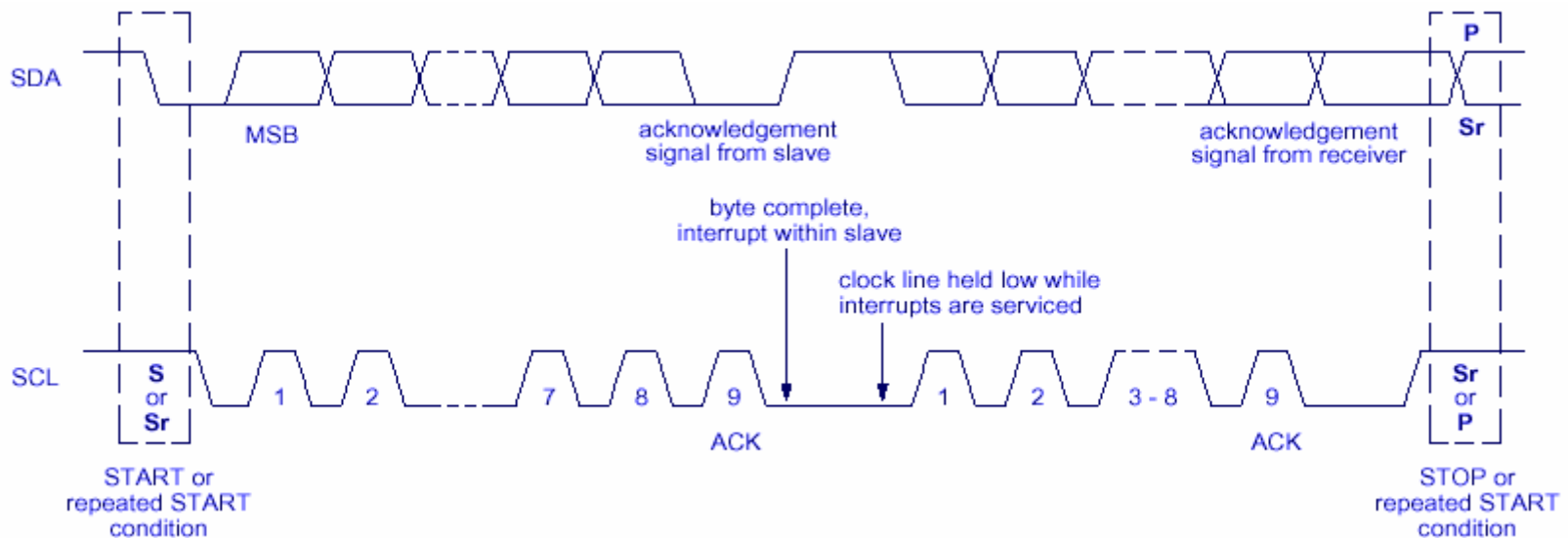
I2C START/STOP Conditions

- START condition: Signals begin of transfer
 - A HIGH to LOW transition on the SDA line while the SCL is HIGH
- STOP condition: Signals end of transfer
 - A LOW to HIGH transition on the SDA line while the SCL is HIGH
- START/STOP is always generated by the Master
- Repeated START condition is allowed
 - Repeated start is used for changing the slave, or changing the direction of data transfer (Send/Receive) for the same slave



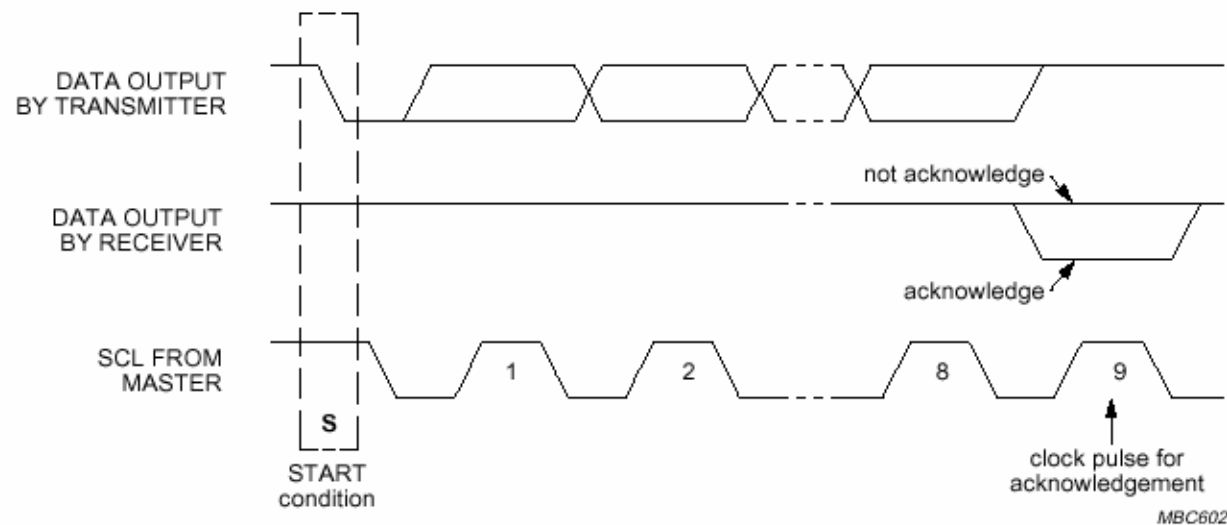
I2C Data Transfer

- Every byte on the SDA line must be 8-bits long
- Each byte must be followed by an acknowledgement from the receiver
- Data byte is transferred bit-wise with the MSB as the first bit sent
- A slave can force the master to wait by holding the clock line SCL LOW



Acknowledgement Scheme

- The acknowledge-related clock-pulse is generated by the master
- The transmitter (master or slave) releases the SDA line
- To acknowledge (ACK), the receiver must pull-down the SDA line during the acknowledge clock pulse



Acknowledgement Scheme

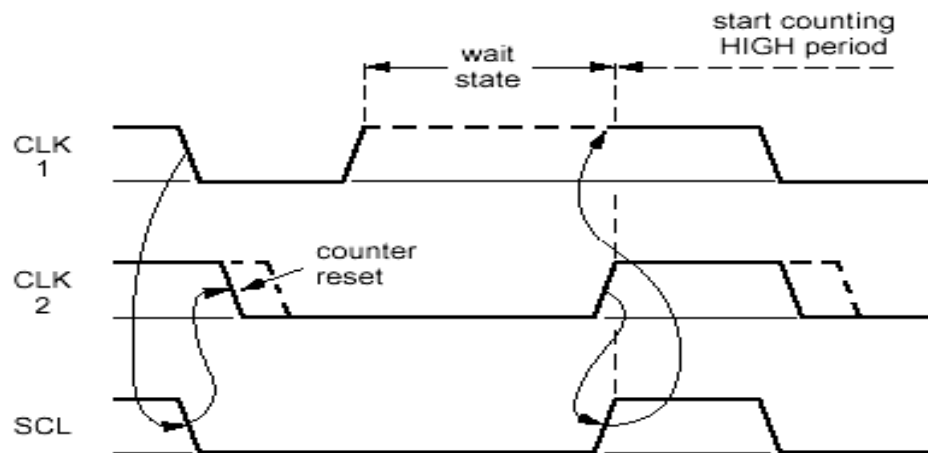
- The receiver is obliged to generate an acknowledge after each byte received
- A slave can not acknowledge (NACK) its address when busy
 - The master must issue a STOP or Repeated START
 - Not allowed by SMBus!
 - The slave can NACK later bytes if unable to proceed
- When receiving data, the master must signal the end of data to the slave by NACKing the last byte
 - The slave must release the data line to allow the master to generate a STOP or repeated START condition

7-Bit Addressing

- The slave address is contained in the first byte after the START
- The first 7 bits contain the address and LSB contains the direction of transfer(R/W' : 0 = write;1= read)
- When an address is sent, each device compares the first seven bits and considers itself addressed.
- A slave address can be made up of a fixed and a programmable part.
 - The programmable part allows multiply instances of the same device
- A 'general call' addressing is also available to address all the slaves at the same time.
 - PMBus uses a "Group" packet instead

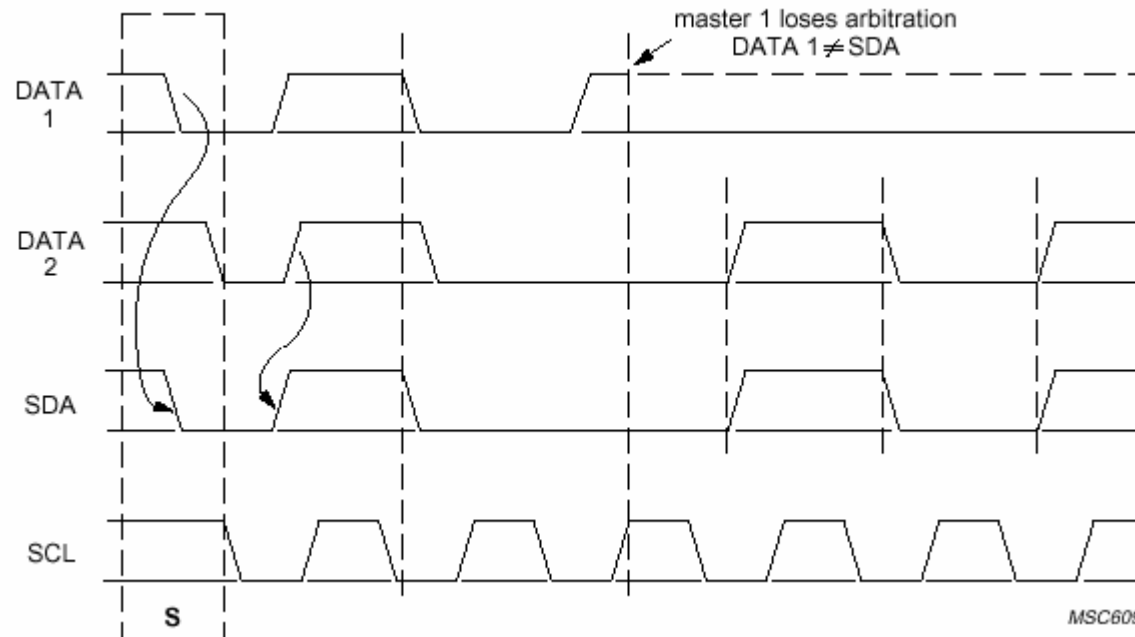
Multi-Master Clock Synchronization

- In the I2C bus, clock synchronization is performed using the wired-AND connection of the interfaces to SCL line.
 - If at least one master clock goes from HIGH to LOW, then the SCL is held LOW irrespective of the other masters' clock.
 - The SCL line goes to a HIGH state only when all the master clocks are in HIGH (this SCL HIGH is triggered by the last master clock that entered into its HIGH state).
- The synchronized clock is generated with its LOW period determined by the device with the longest clock LOW period and its HIGH period determined by the one with the shortest clock HIGH period.



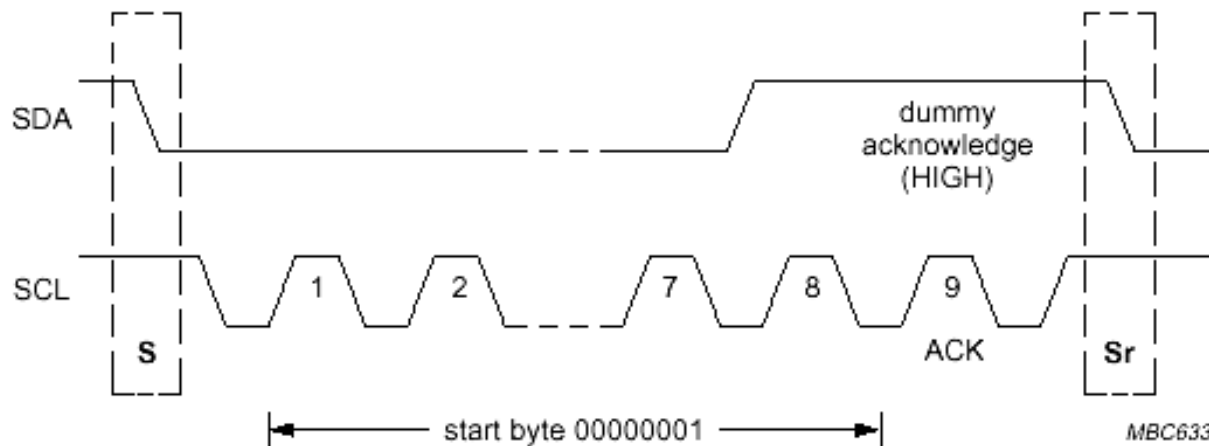
Multi-Master Arbitration Using the Clock Syn.

- If more than one device is capable of being a master, arbitration is needed to choose the master that takes control of the bus
- Arbitration takes place on the SDA, while the SCL is HIGH,
 - the master which transmits a HIGH level when another master is transmitting LOW level will switch off its DATA output stage because the level on the bus does not correspond to its own level



Software Interface to I2C Bus

- It is also possible to use an I2C software interface (bit-banged)
- Software constantly monitors the bus. Hence, these controllers are inherently slower as they have to spend time for polling
- The START process is lengthier also.
 - START ; START byte (00000001);ACK clock pulse; Repeated START
- Not supported in SMBus/PMBus



Common I2C Implementation Errors

- I2C is deceptively complicated
 - Wired-AND nature
 - Arbitration
 - Clock synchronization
- Sensitive to SCL/SDA noise
 - Use analog (or preferably digital) filters on SCL/SDA
 - Implement SMBus Error detection and timeout features
- Improper Master arbitration & synchronization
 - Never assume SCL returns HIGH when released
 - measure HIGH time based on sensed HIGH
 - Compare against transmitted SDA when SCL goes HIGH
 - Watch for SCL to go LOW when transmitting a HIGH

Why SMBus vs. I²C?

- Low cost like I²C
- More robust than I²C
- More features than I²C
 - SMBALERT# line for interrupts
 - Packet error checking (PEC)
 - Packet timeouts
 - Erroneous START, missing STOP protection
 - Host Notify Protocol
- Generally electrically compatible with I²C
 - Uses TTL levels rather than proportional

SMBus Improvements

- I²C “Noise Sensitivity” – False START/STOP
 - Packet Error Checking protects against erroneous mid-packet START/STOP
 - 50usec timeout protects against rogue STARTs and missing STOP
- I²C “Noise Sensitivity” – Corrupt Data
 - Data Rates Permit Analog/Digital Filtering
 - Packet Error Checking (PEC)
 - In PMBus, every value that can be written can be read

SMBus Improvements

- I²C slave device hangs the bus
 - 25msec timeouts force device reset
- I²C requires retrieving device information by polling
 - SMBALERT# line acts as a wired-AND interrupt
 - Alert Response uses arbitration to implement interrupt priority management
 - Device with lowest address gets priority
 - Host Notification allows a device to report information to the bus host
 - Device acts a master using a specific packet format including its own address

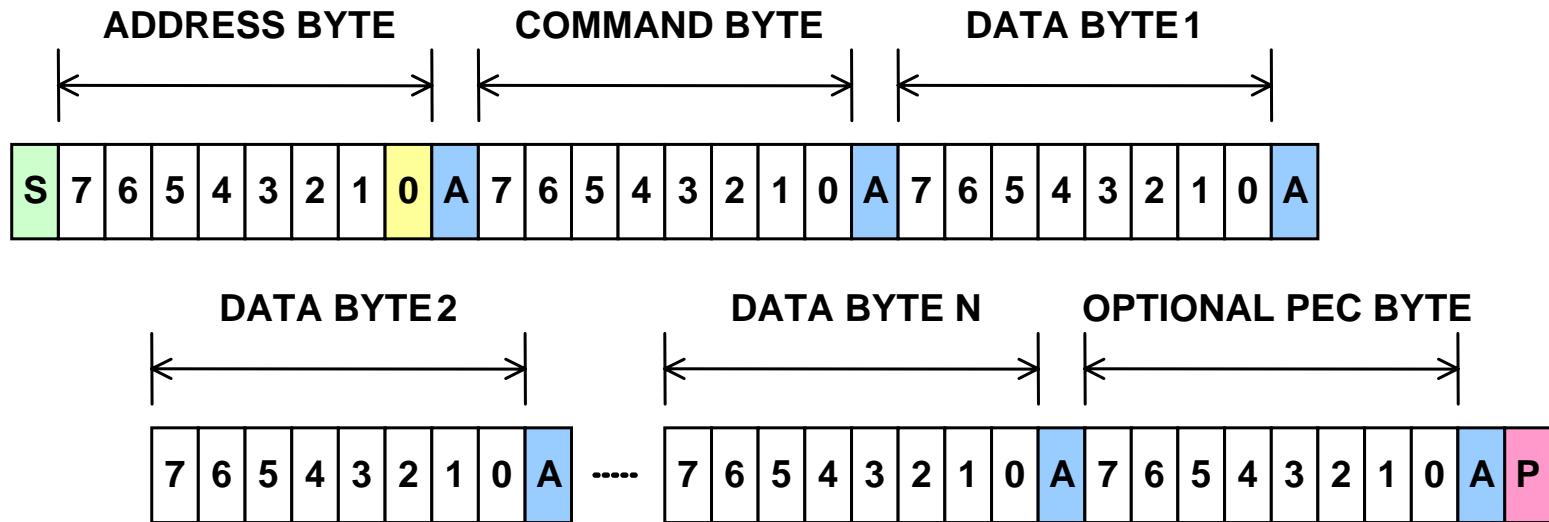
SMBus Limitations

- SMBus and PMBus specifications say 100 kHz
 - I²C allows 400 kHz - PMBus Rev 1.1 will allow 400kHz
- Capacitance is a concern
 - No explicit maximum
 - Excessive capacitance causes a violation of bus timing by slowing rise times
 - See SMBus specification for details

Addressing

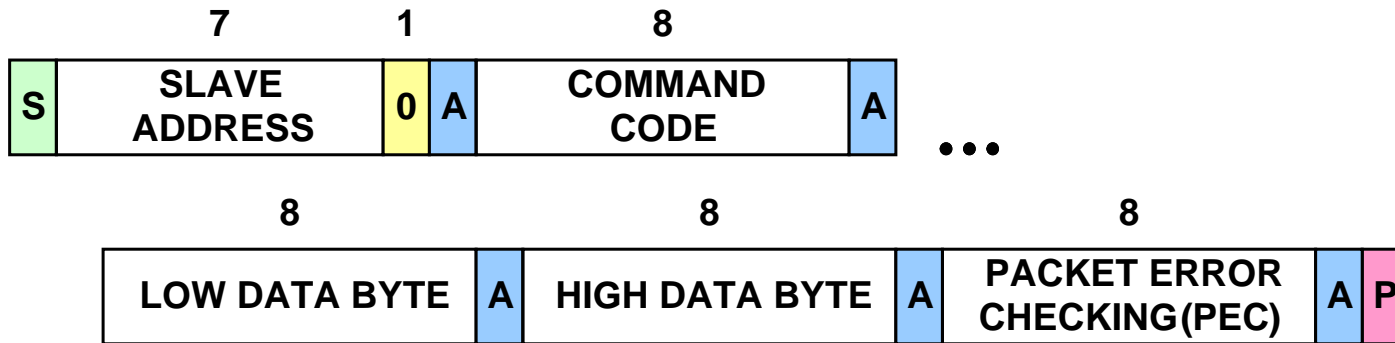
- PMBus devices use a 7 bit address
Per The SMBus Specification
 - Provides more than 100 possible device addresses after allowing for reserved addresses
- PMBus users can expect device addresses to be set by a mix of:
 - Hardwired address pins
 - High order address bits set by the PMBus device manufacturer
- PMBus device manufacturers will trade off cost of pins versus address flexibility

Basic Packet Structure



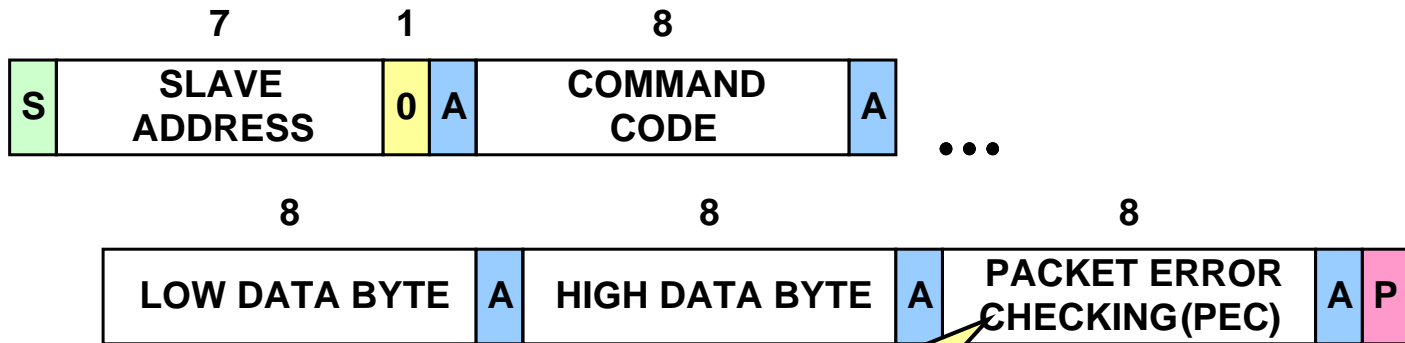
S START Signal From Host System
 0 READWRITE# Bit
 A ACKNOWLEDGE Signal From Converter
 P STOP Signal From Host System

Write Word Packet



S START Signal From Host System
 0 READWRITE# Bit
 A ACKNOWLEDGE Signal From Converter
 P STOP Signal From Host System

Write Word Packet

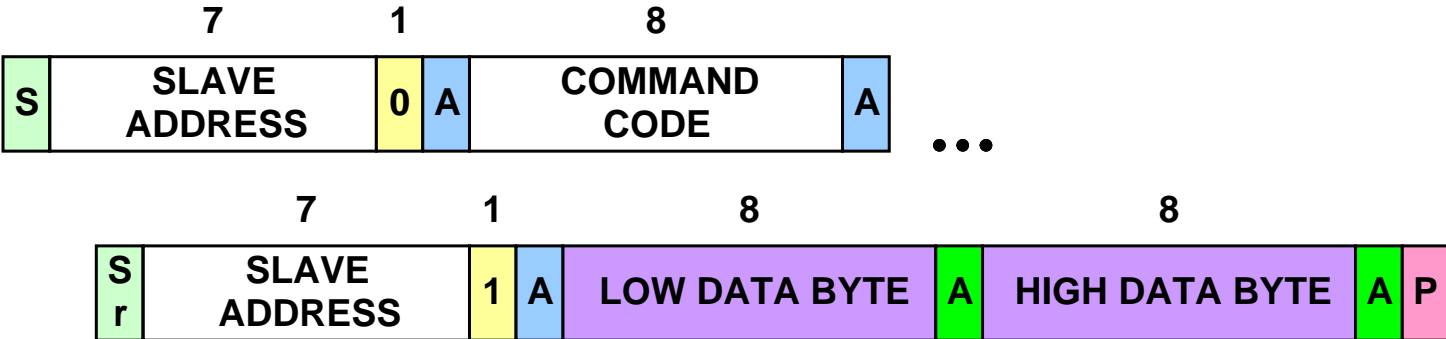


S START Signal From **0** READWRITE KNOWLEDGE From Converter **P** STOP Signal From Host System

**Packet Error Checking (PEC)
is optional in the specification**

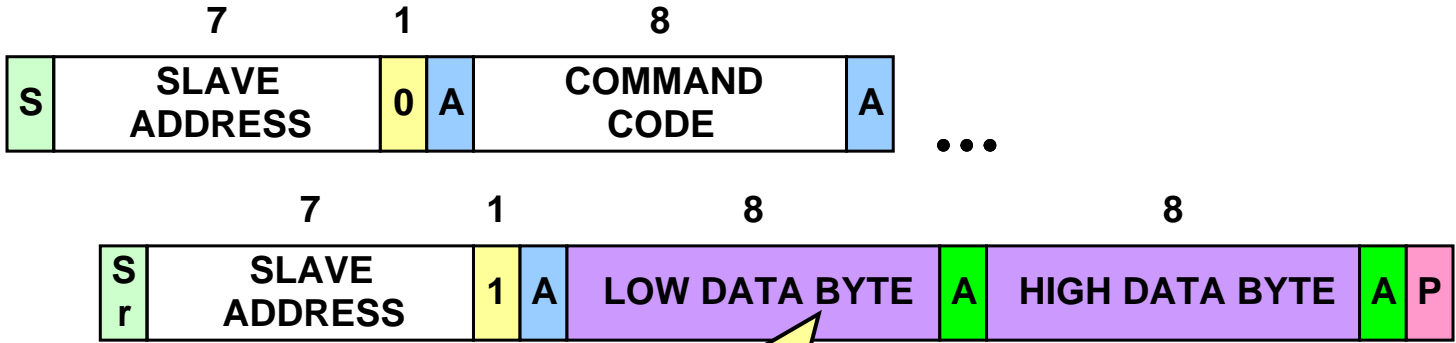
**But it is expected to be
very popular with system OEMs!**

Read Word Packet



S	START Signal From Host System	0	READWRITE# Bit	A	ACKNOWLEDGE Signal From Converter	P	STOP Signal From Host System
Sr	Repeated START Signal From Host System	1	READWRITE# Bit	A	ACKNOWLEDGE Signal From Host System		

Read Word Packet



**This Data Is Being Transmitted
By The Slave Device
To The Host**

S START Signal From
0 READ Signal From Converter
ACKNOWLEDGE Signal From Host System
P STOP Signal From Host System

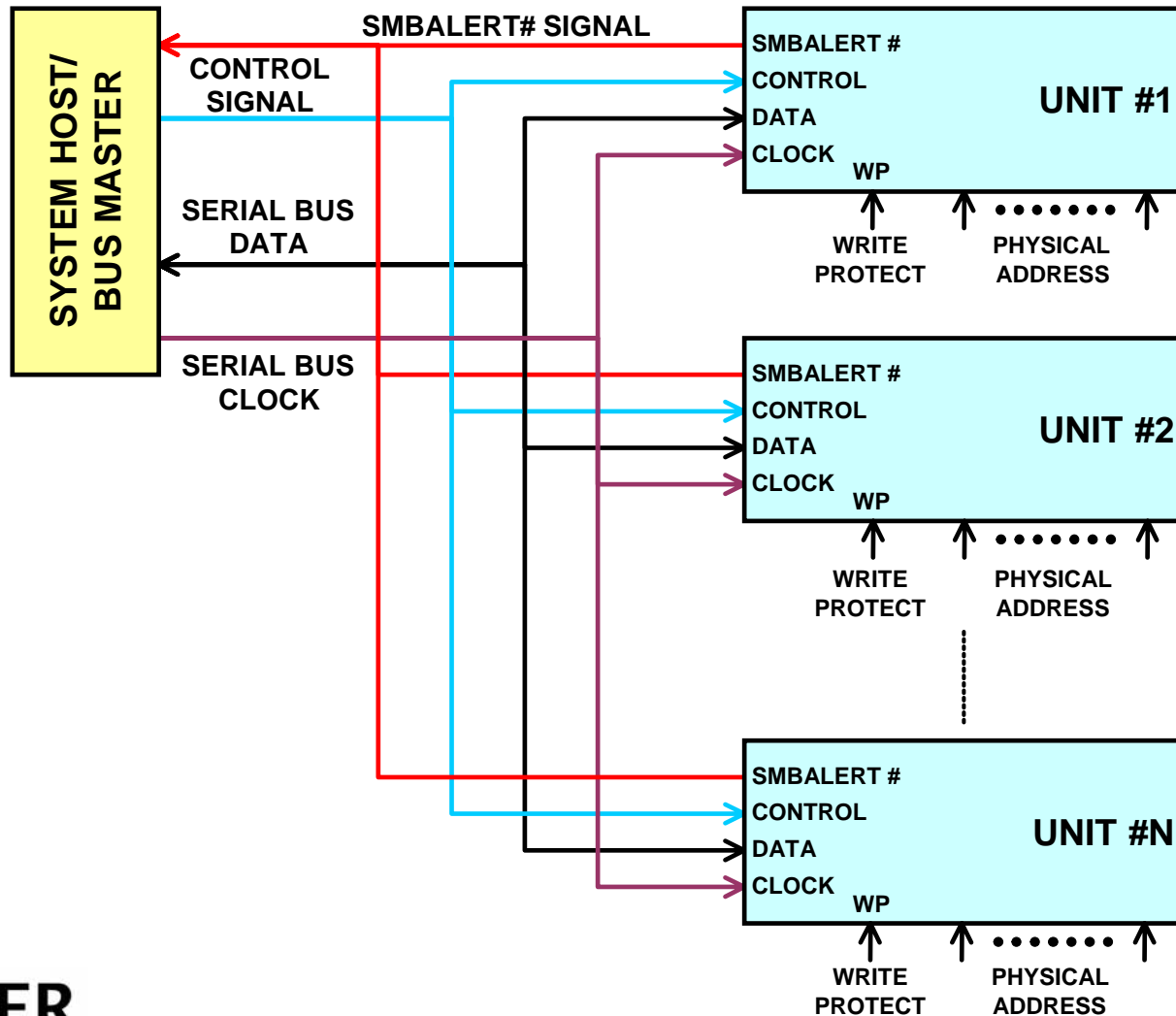
Packet Error Checksum

- Uses the CRC-8 polynomial
 - Be careful, there are multiply CRC-8 polynomials out there!
- $C(x) = x^8 + x^2 + x + 1$
- Easily implemented in software
 - Bit Shift+XOR = 135 instructions/byte
 - Partial look-up table = 100 instructions/byte
 - Full look-up table (256 bytes) = 25 instructions/byte
- www.smbus.org/faq/crc8Applet.htm

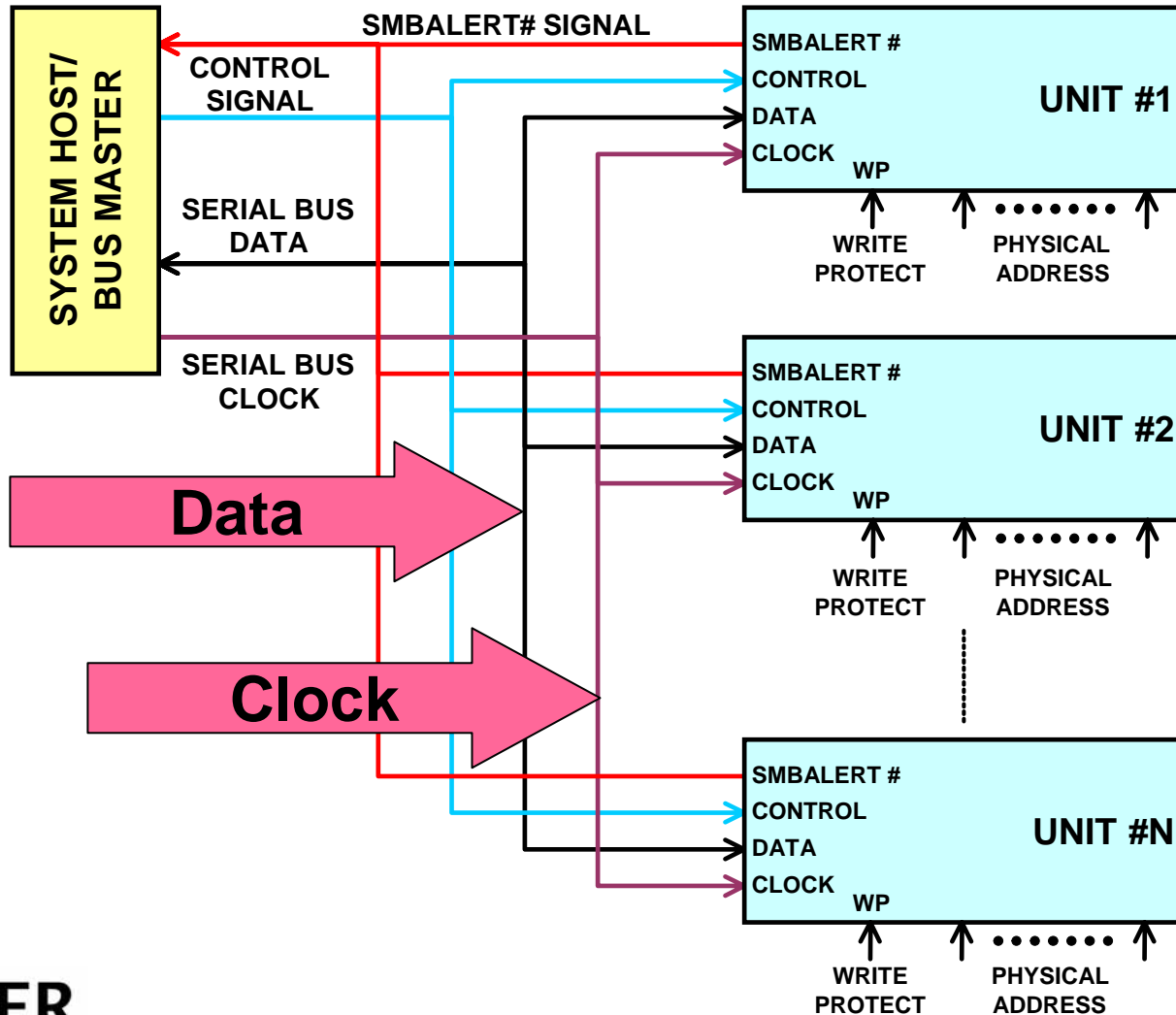
Group Commands/Operation

- Used when multiple units need to execute a command simultaneously
- One SMBus transaction used to send commands to multiple addresses
 - Sent in one large packet using repeated STARTs
- Can be same or different commands
 - Example: command one unit to margin low and all others to margin high
- Commands are executed when SMBus STOP condition received!

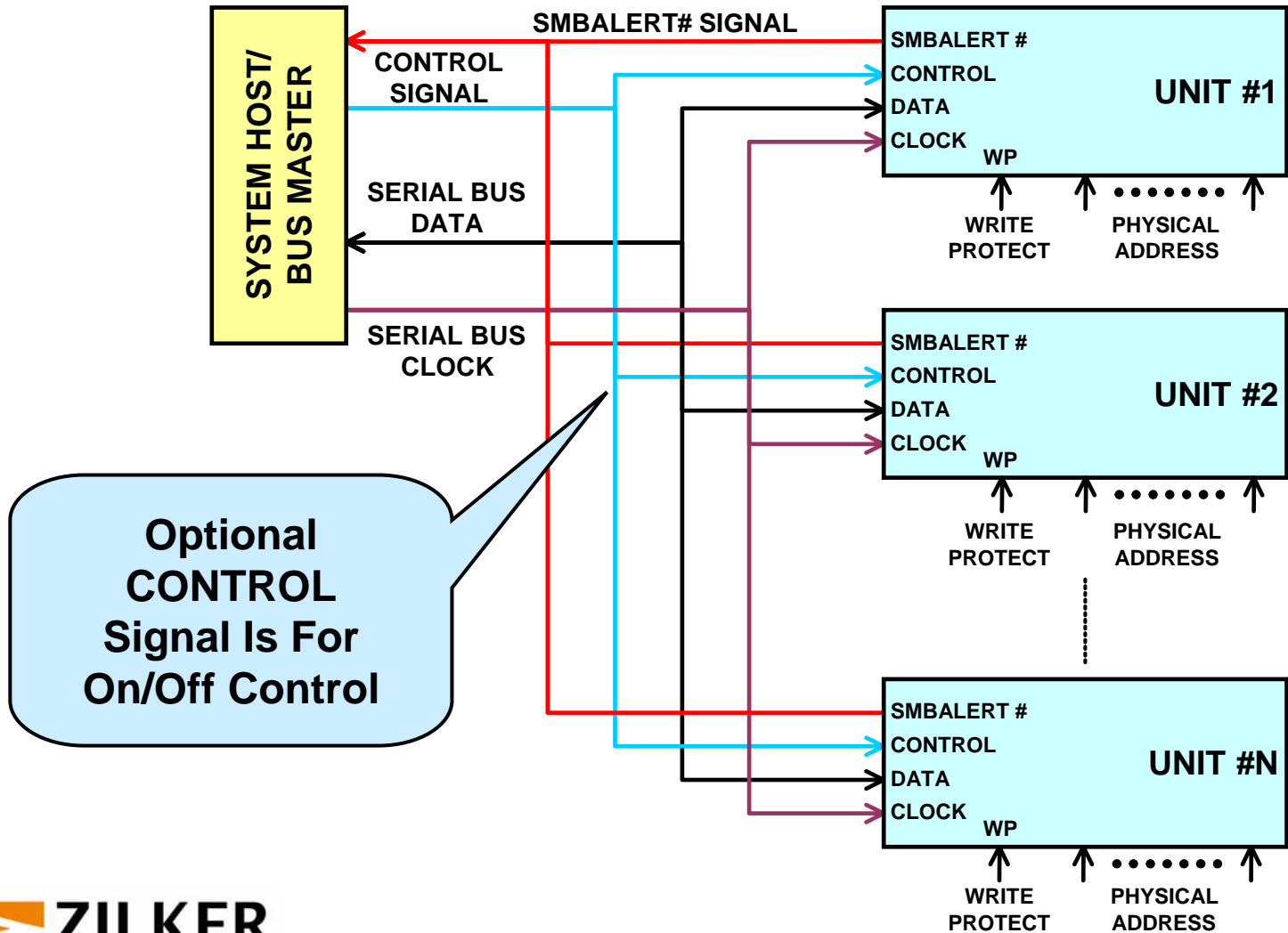
PMBus™ Connections



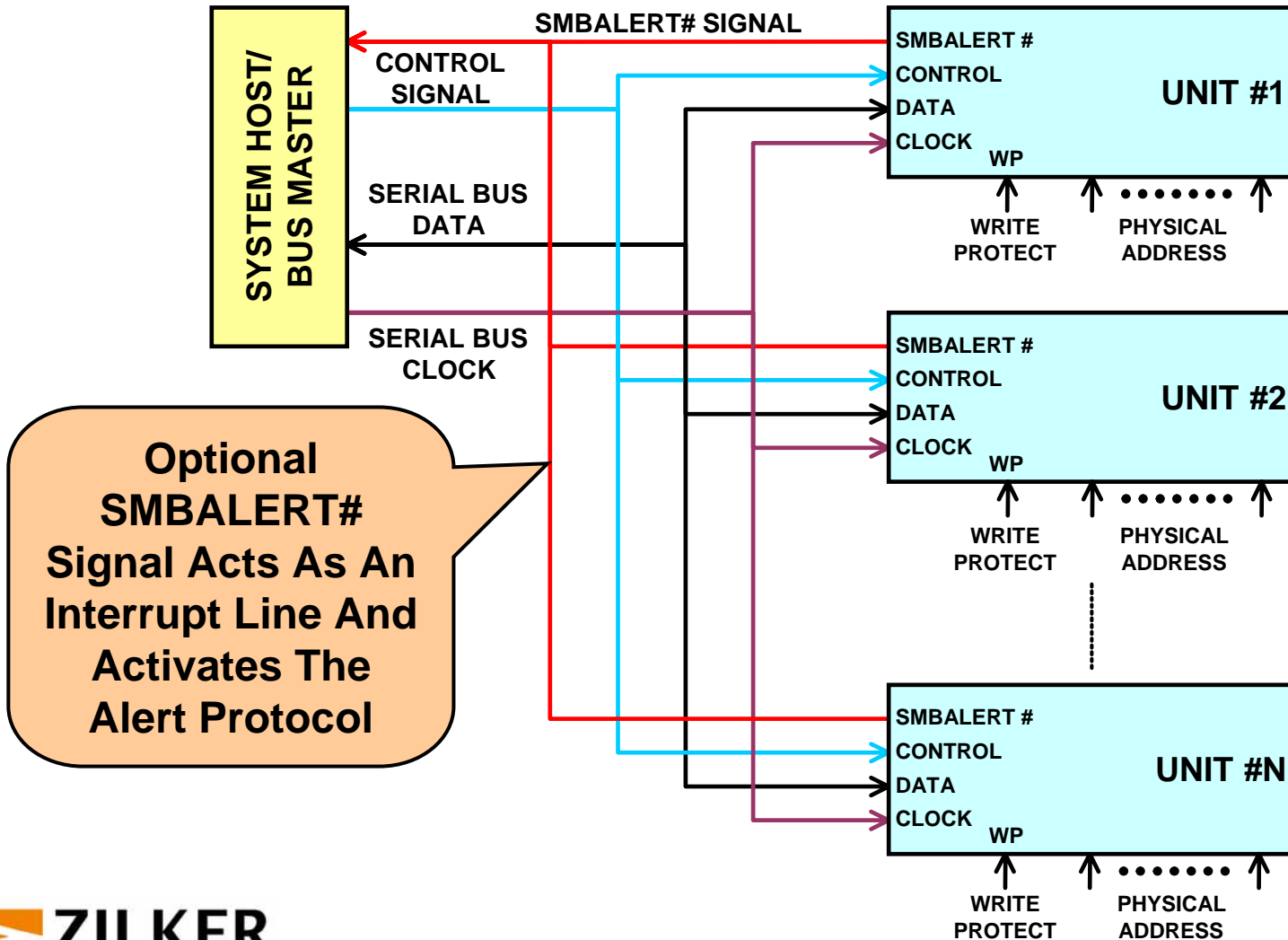
PMBus™ Connections



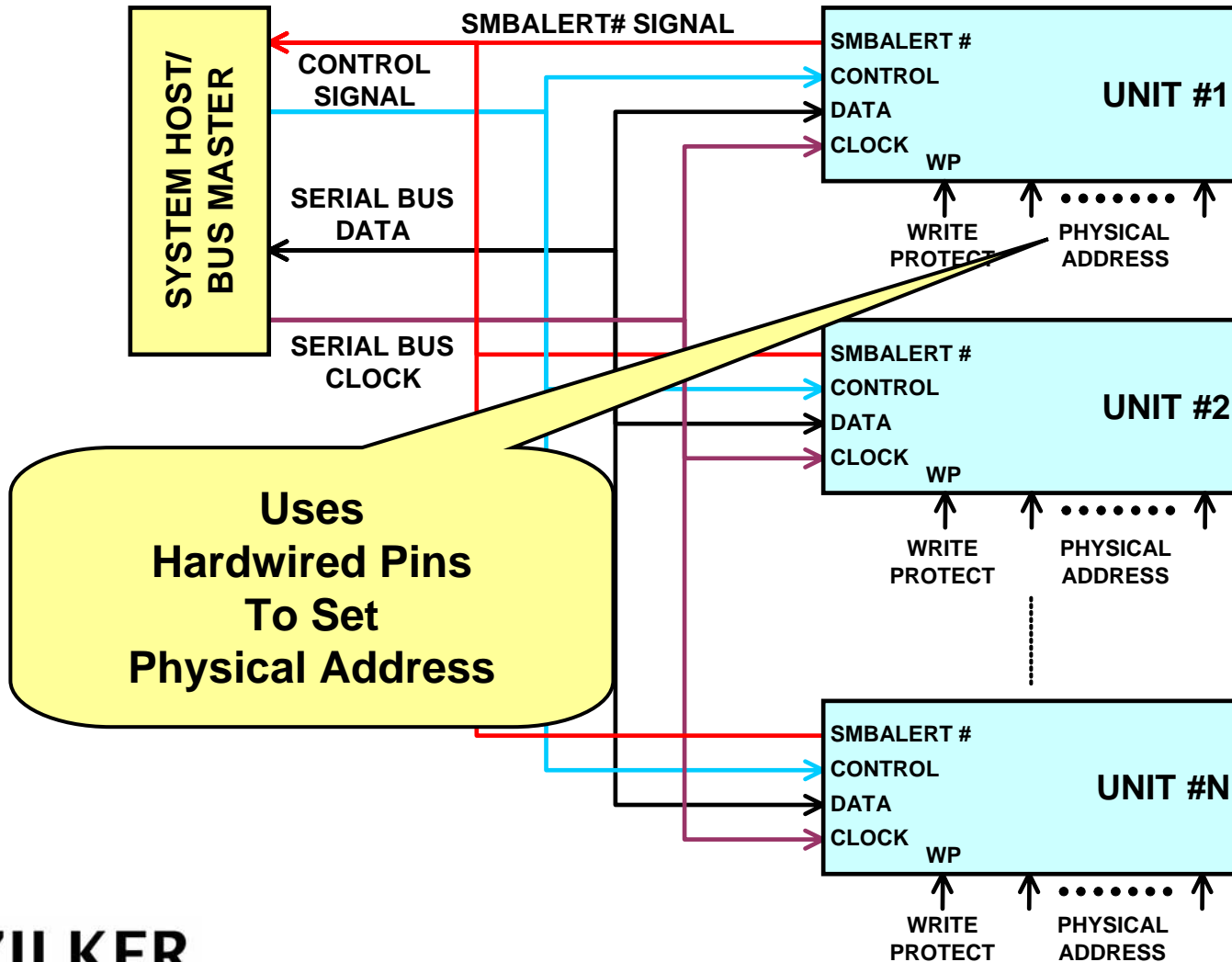
PMBus™ Connections



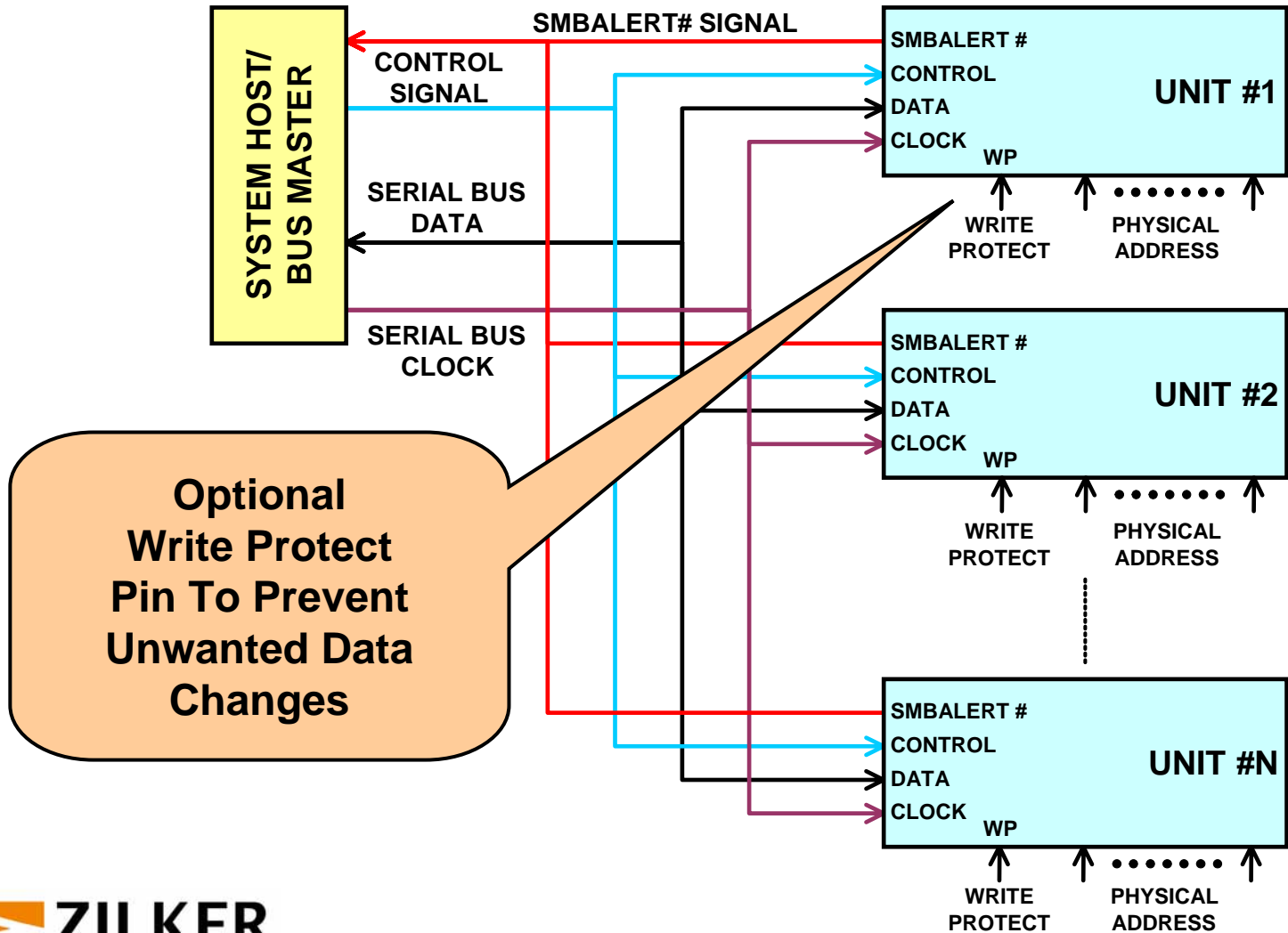
PMBus™ Connections



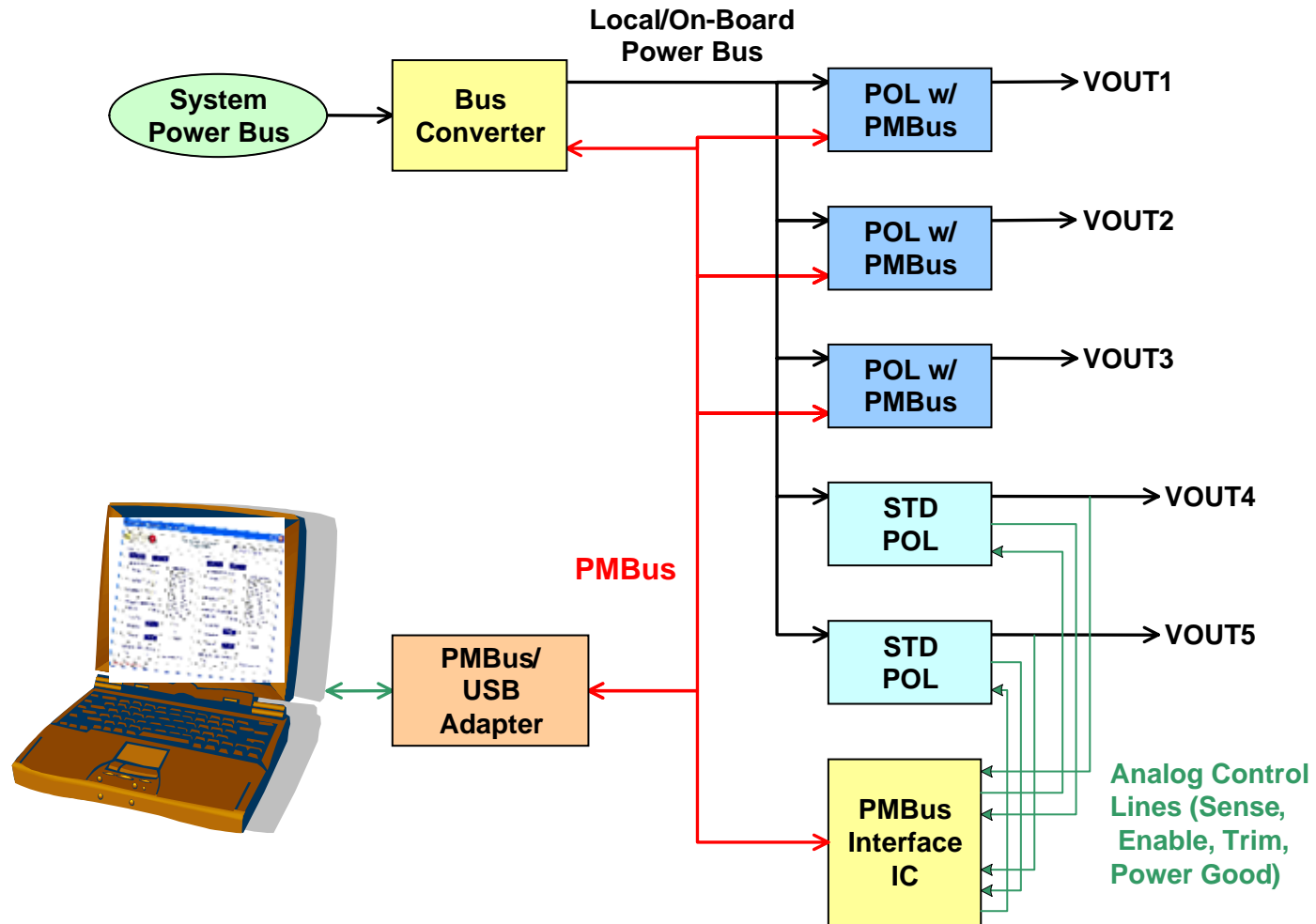
PMBus™ Connections



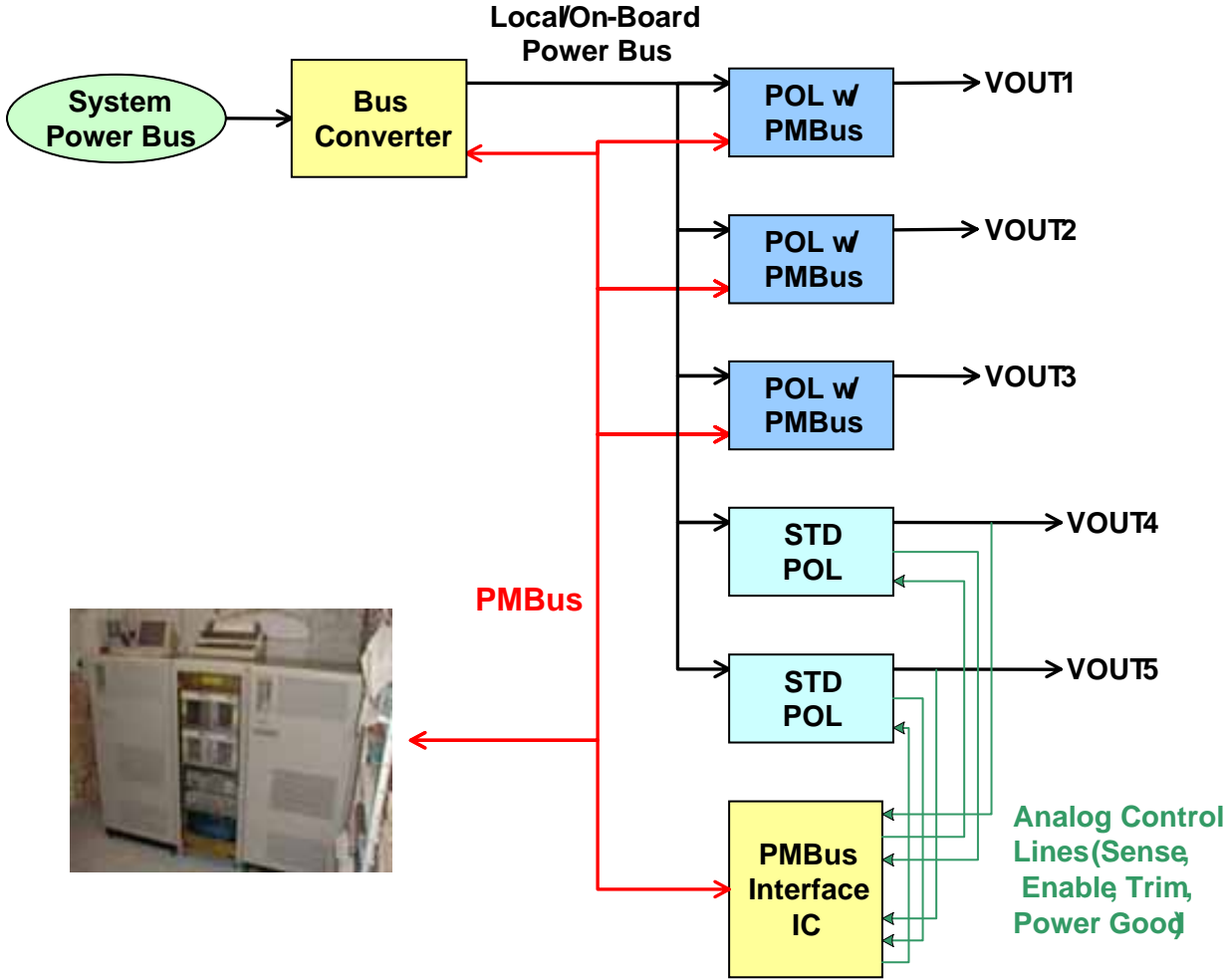
PMBus™ Connections



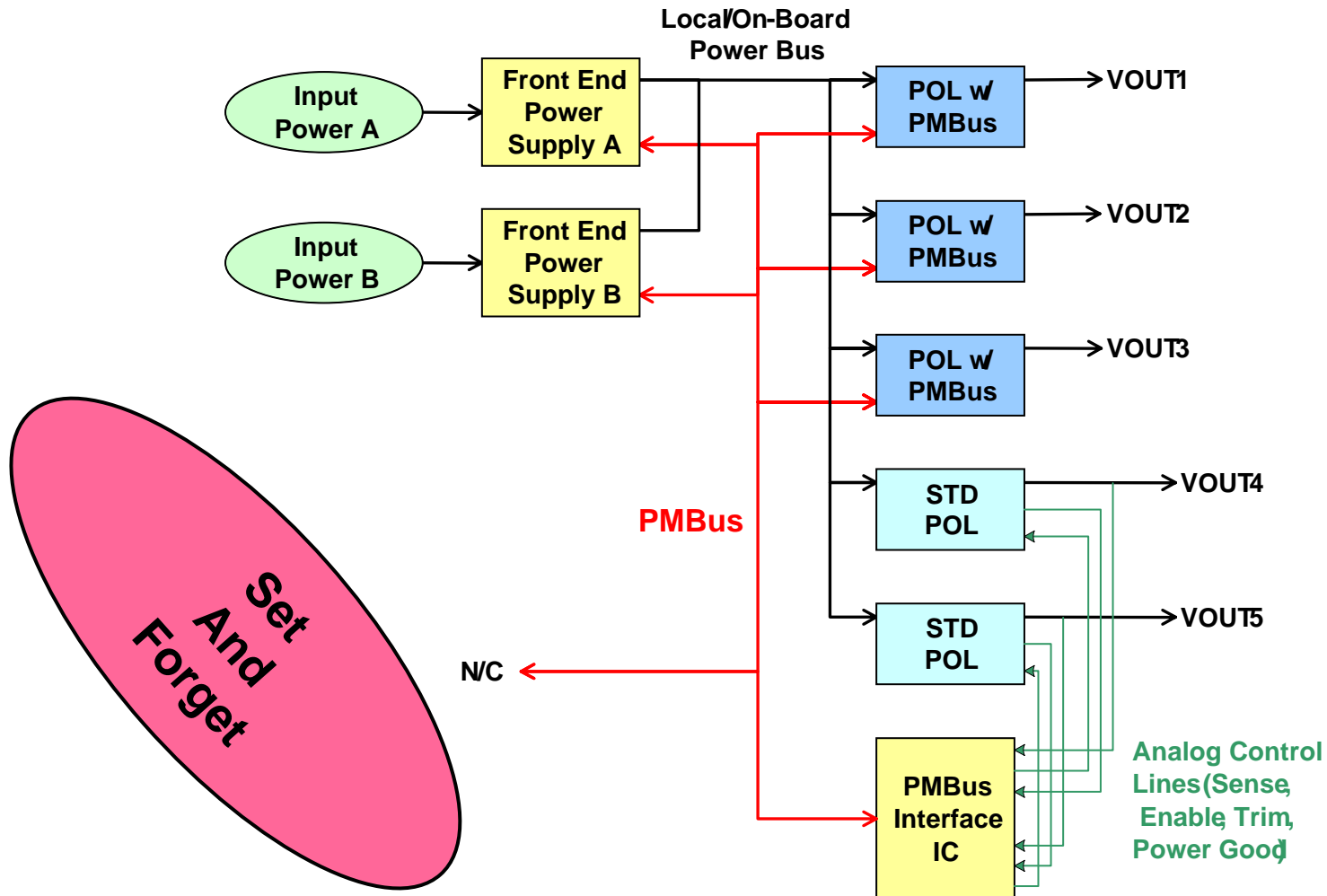
Using PMBus In The Lab



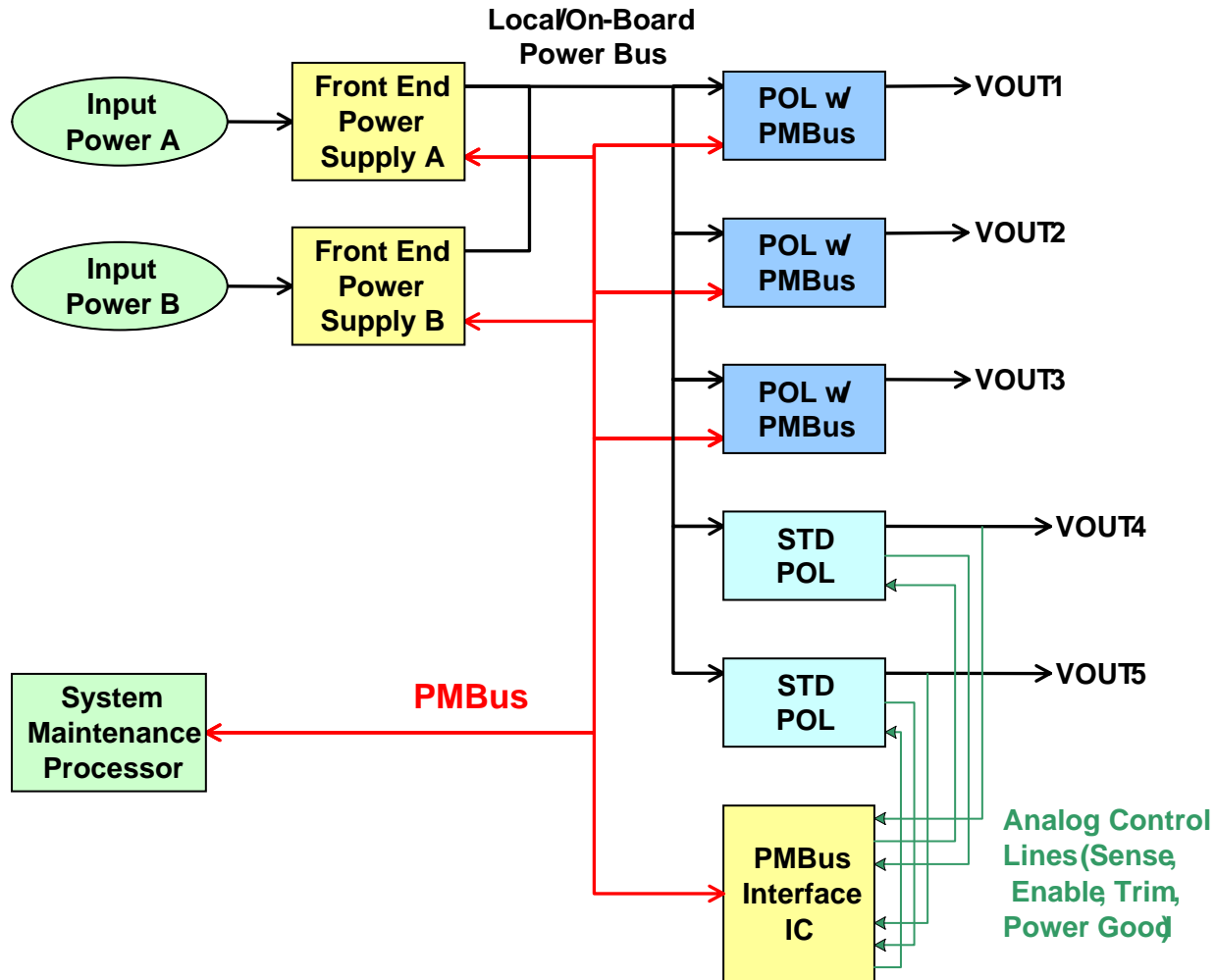
Using PMBus In The Factory



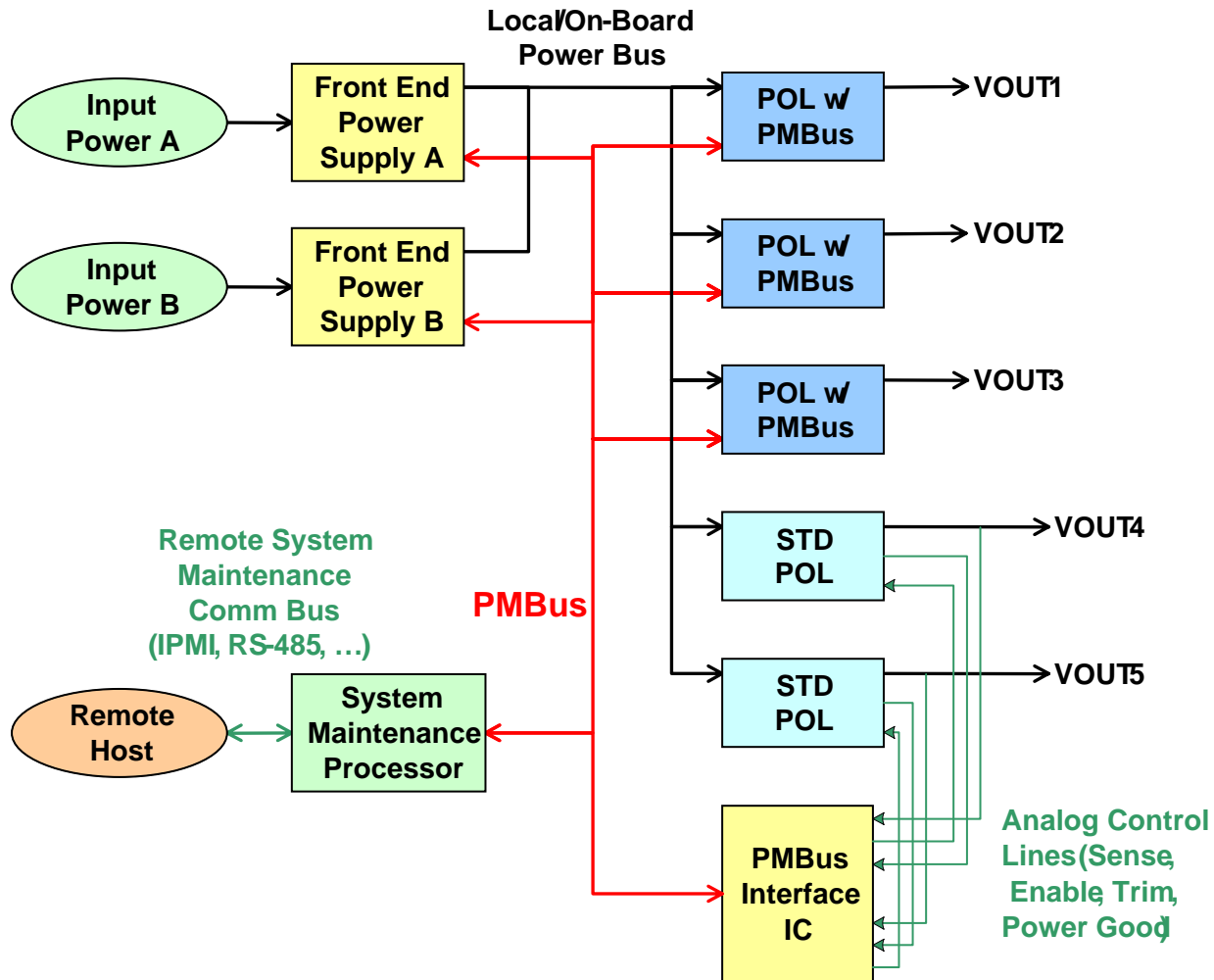
Using PMBus In A System



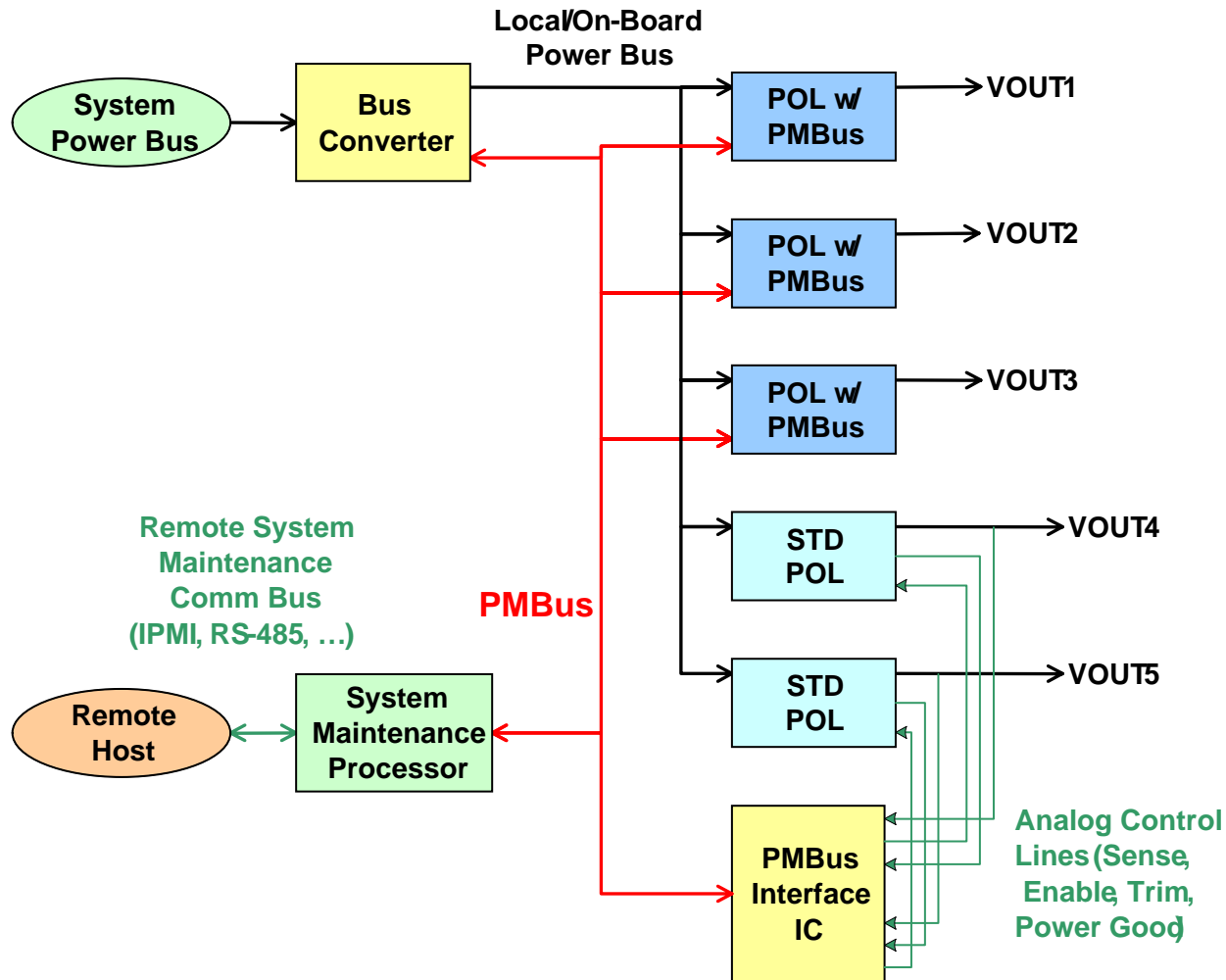
Using PMBus In A System



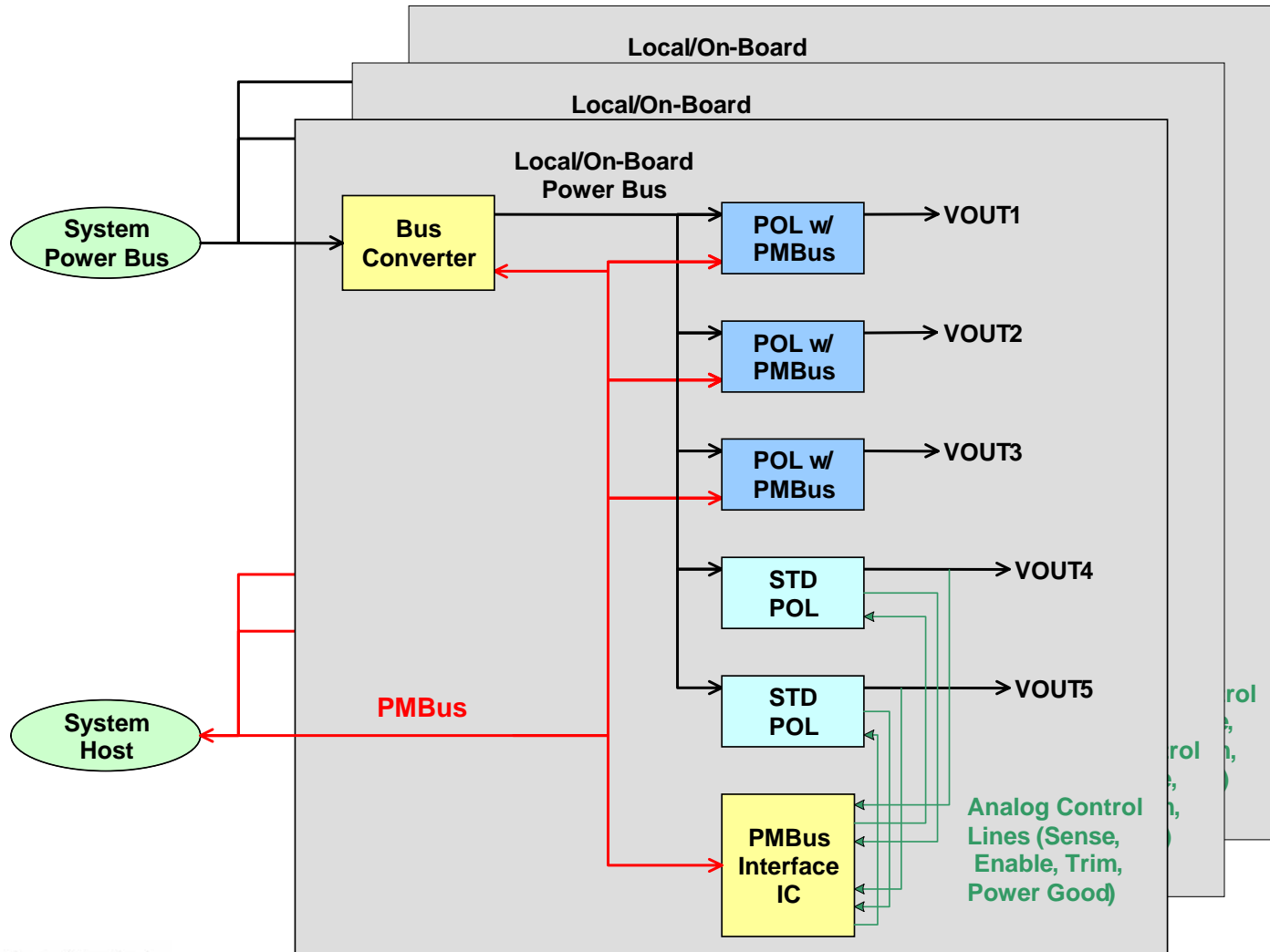
PMBus In A System



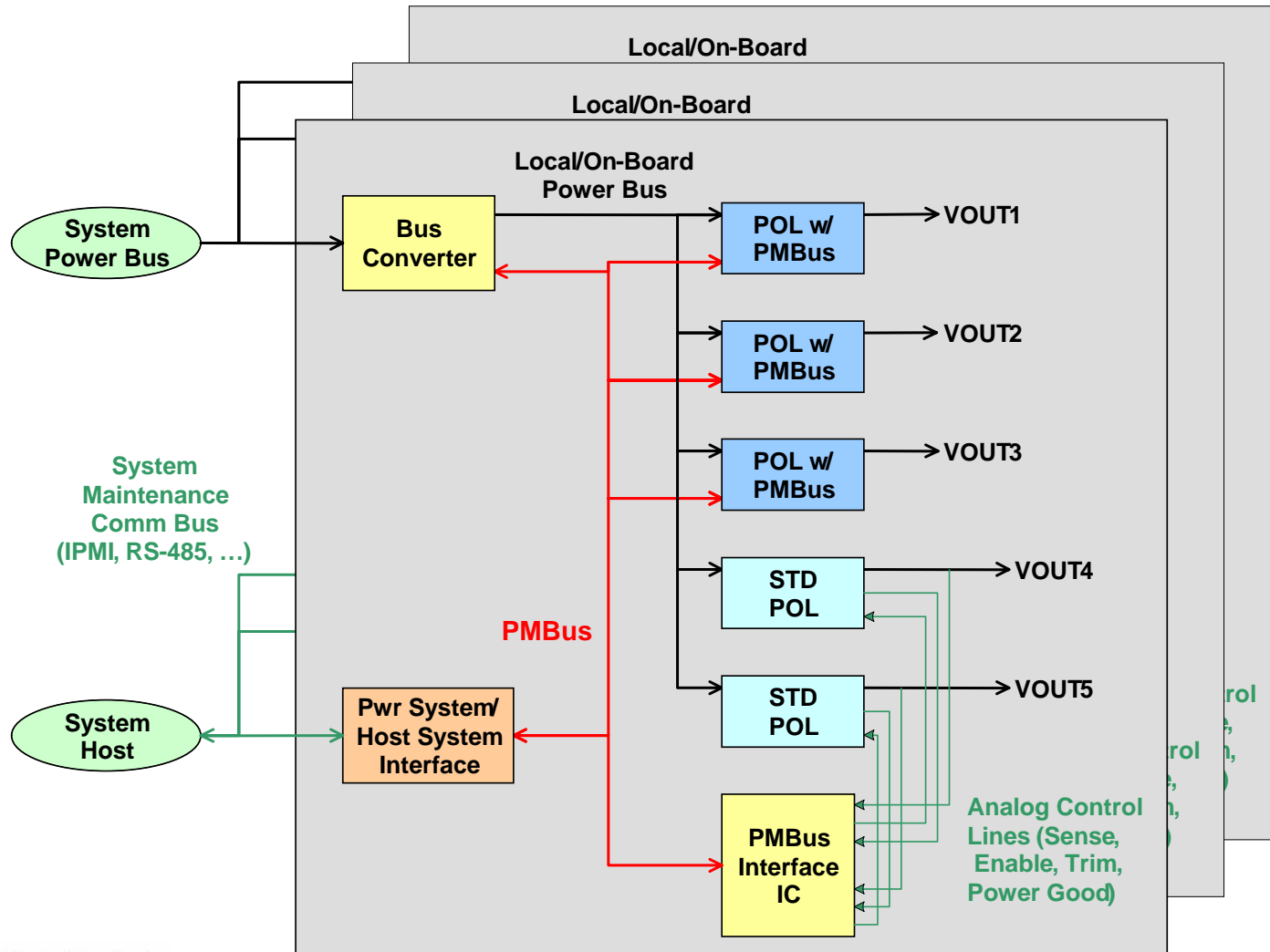
PMBus In A System



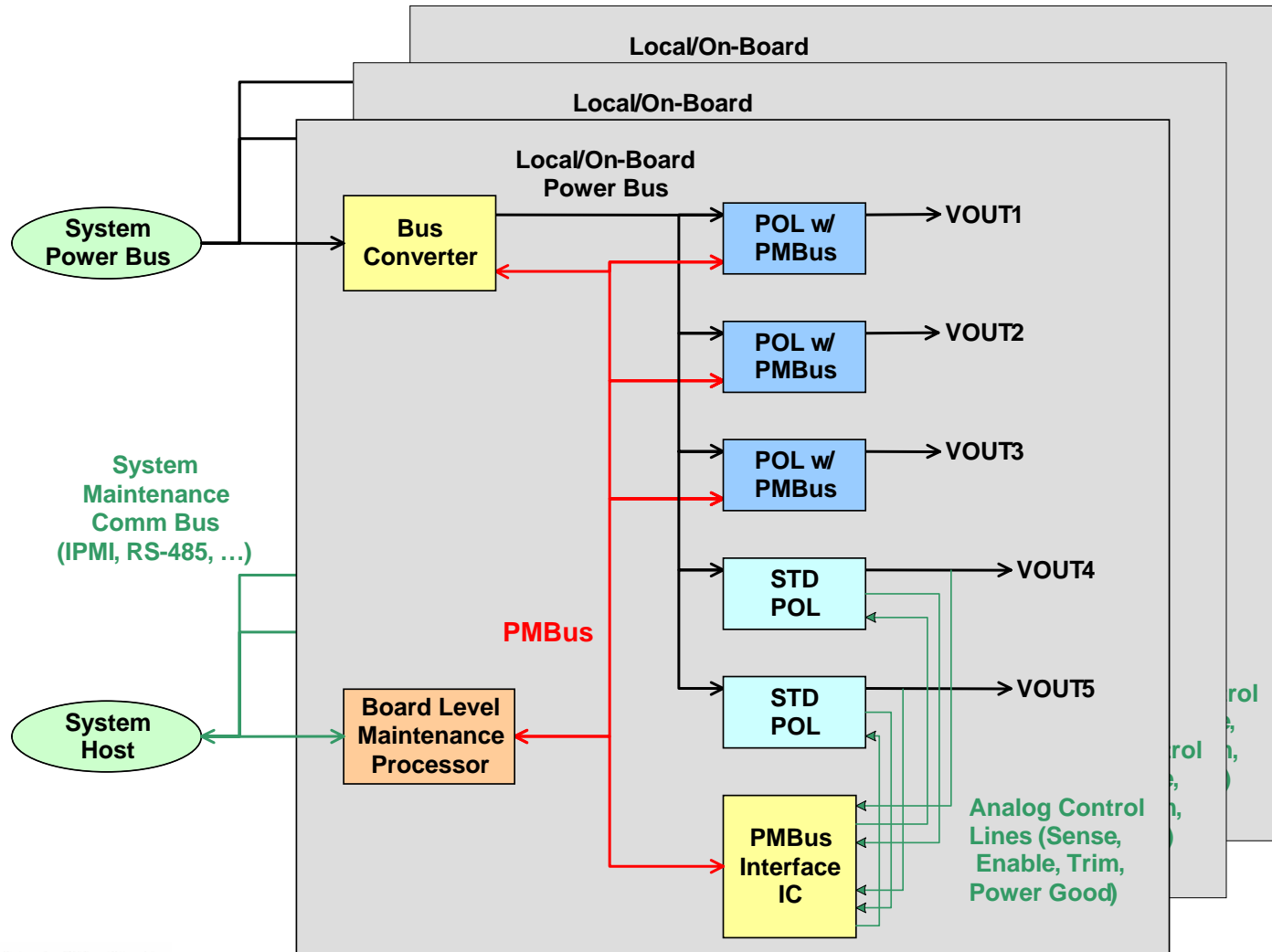
PMBus In A System



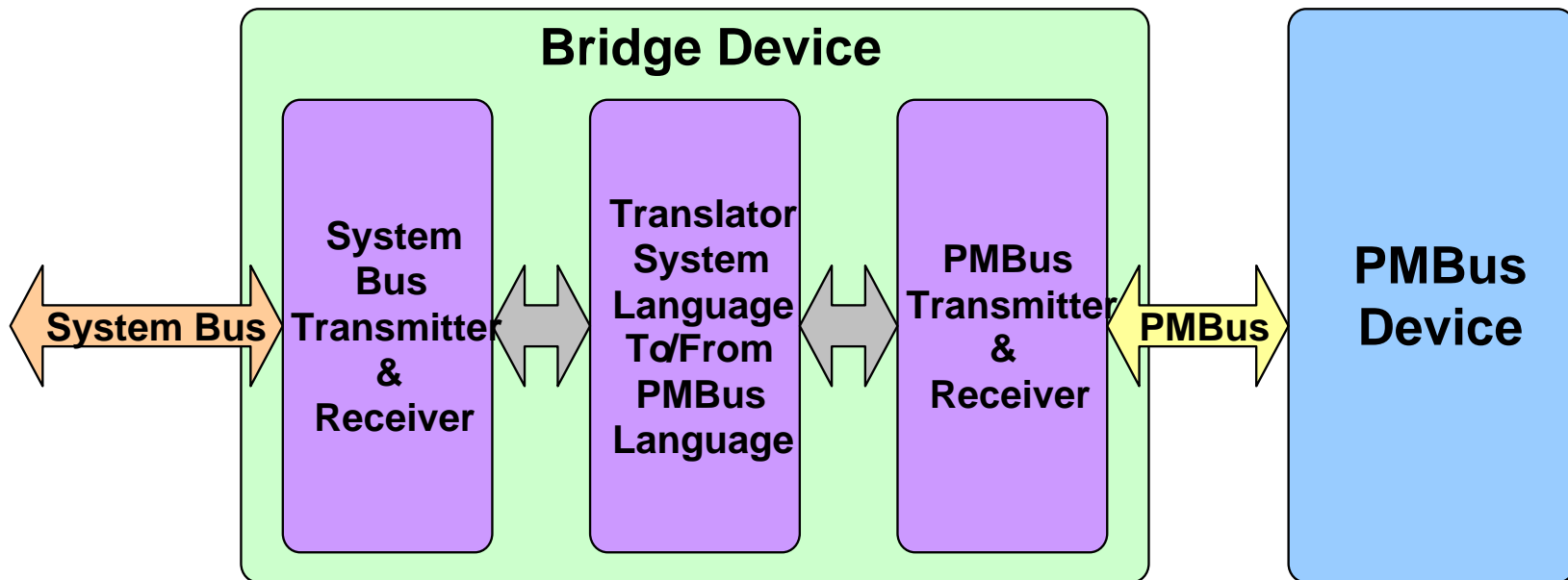
PMBus In A System



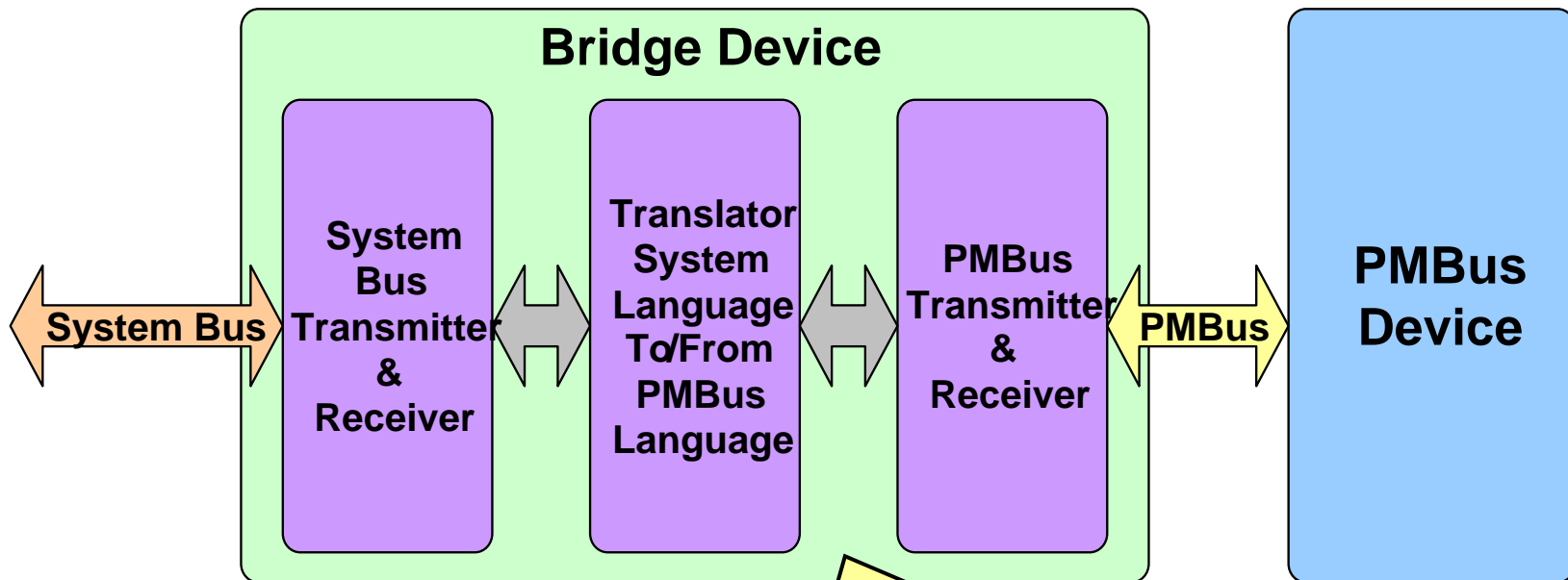
PMBus In A System



PMBus Bridge To Other Buses

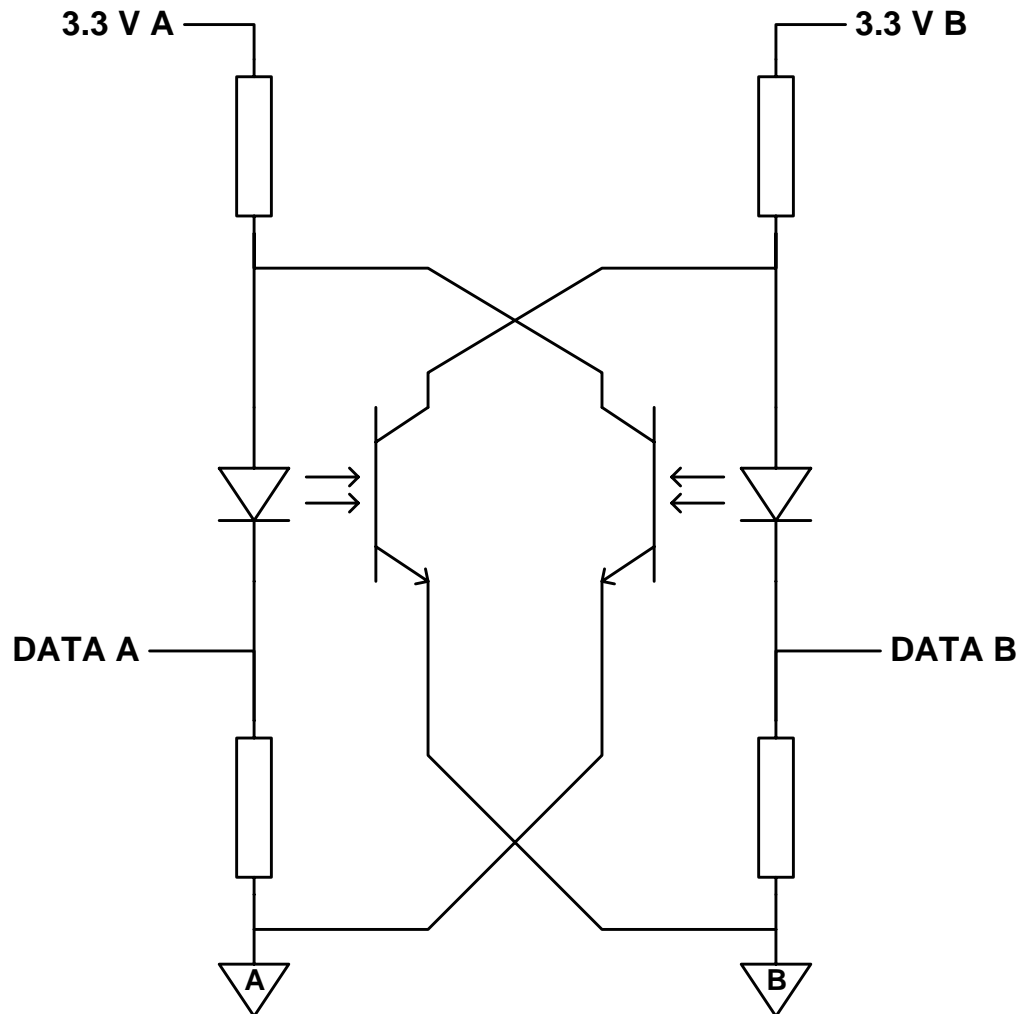


PMBus Bridge To Other Buses

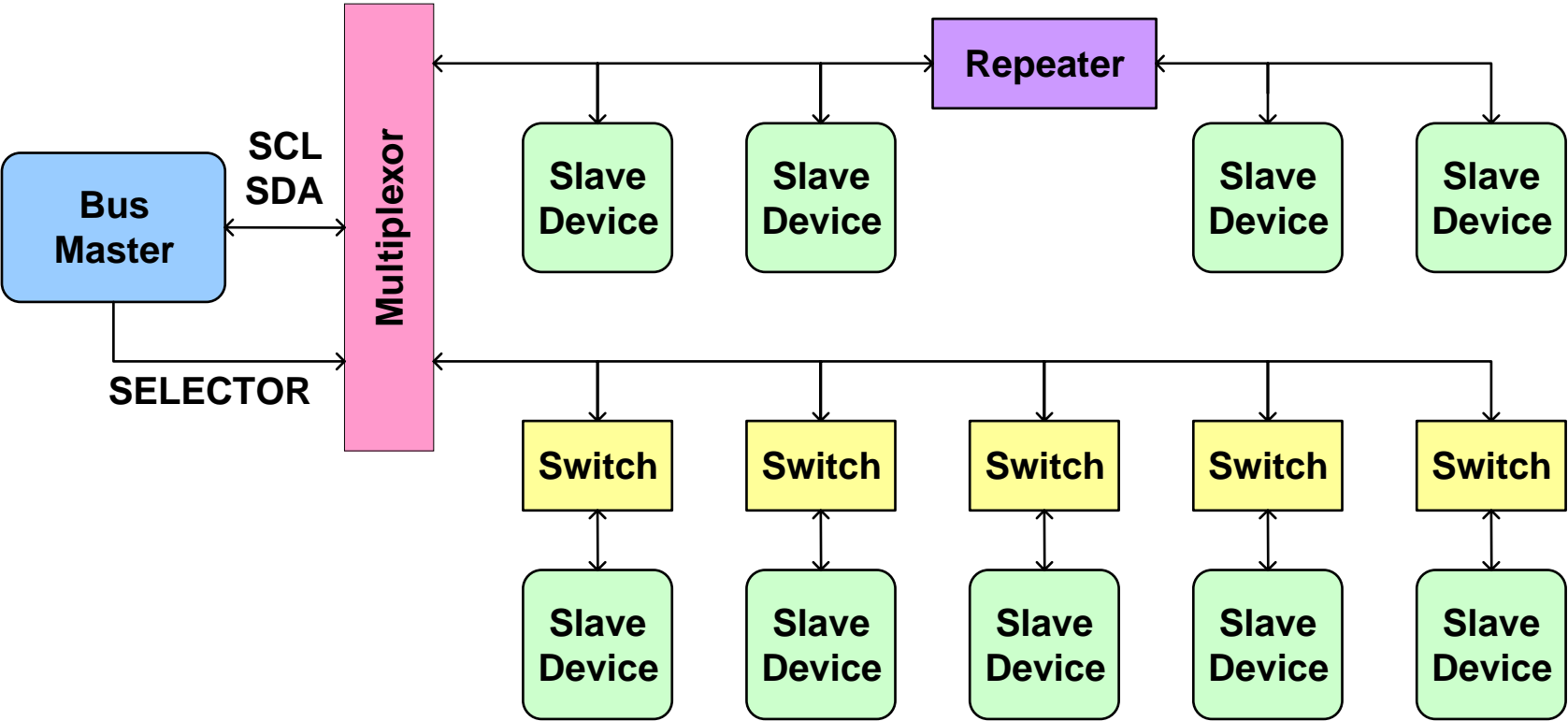


Extra Gates In An FPGA
– Or –
General Purpose Microcontroller
– Or –
Application Specific IC

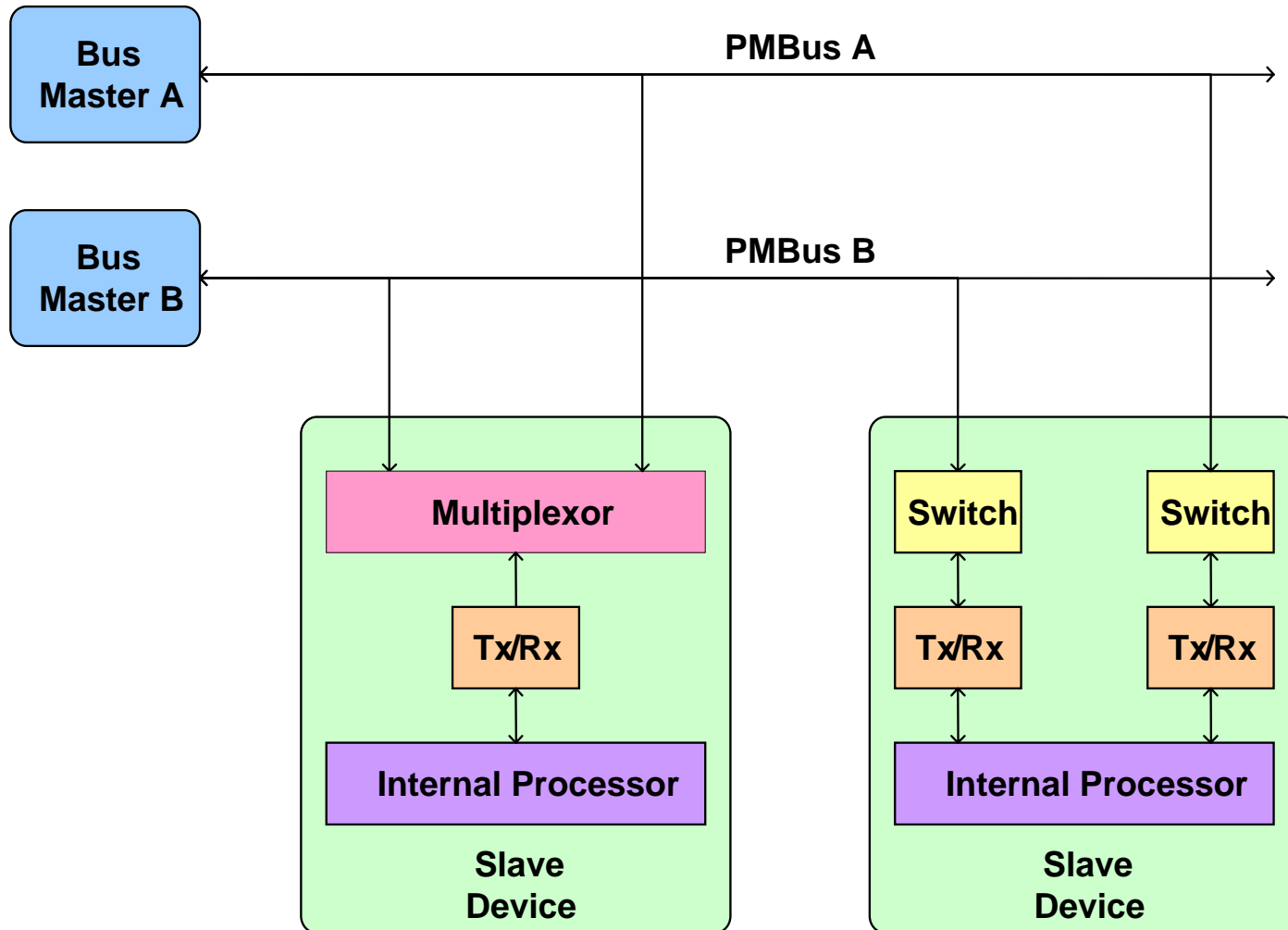
Simple Electrical Bi-Directional Isolation



Bus Extensions



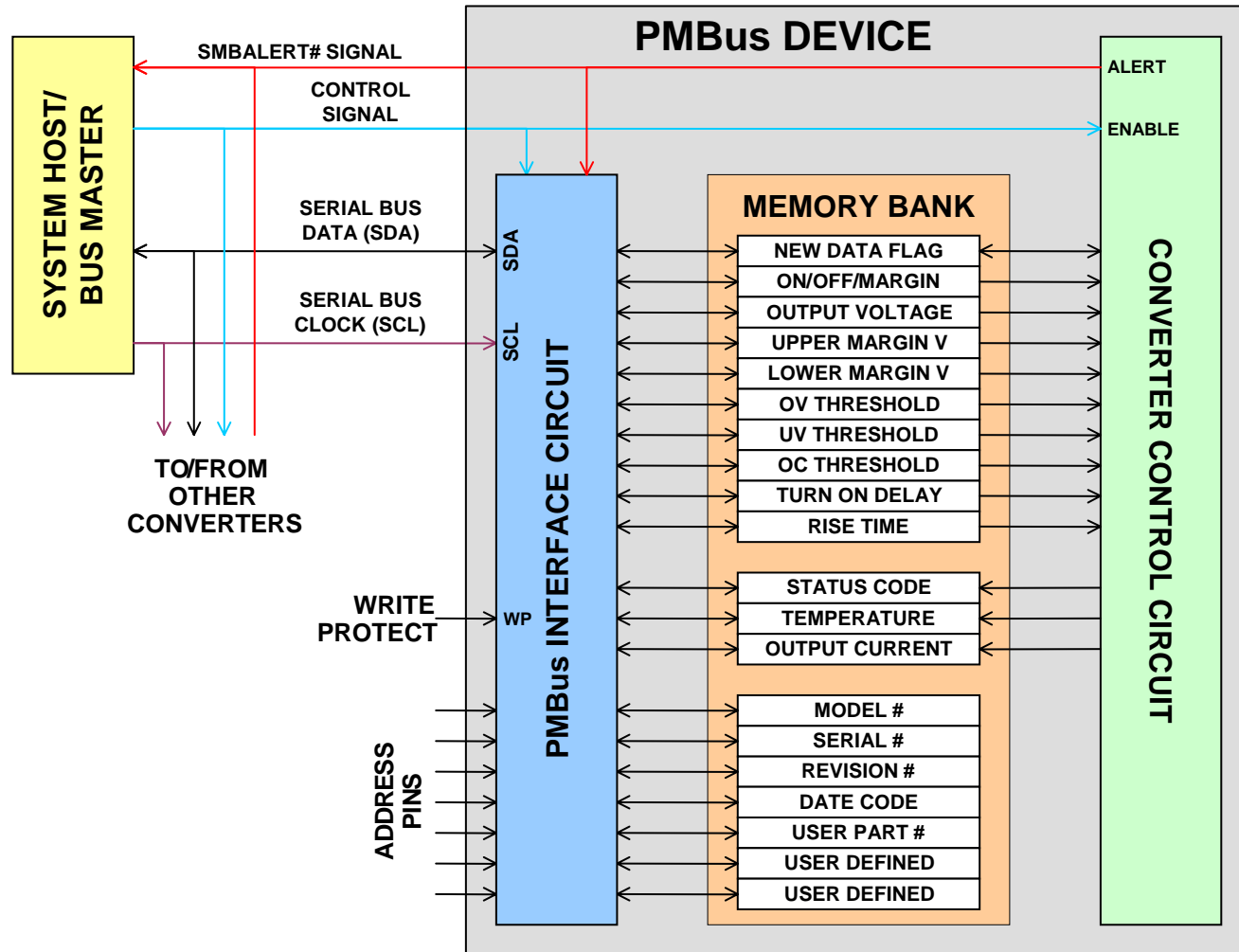
Redundant Buses



What's Needed To Make A PMBus Device?

- Physical/Data link layer to receive & send data over the bus
 - SCL, SDA, CONTROL, SMBALERT#, WP
 - Address pin interface
- Memory
 - Received configuration
 - Device status and parametric information
- The rest of the device
 - Such as power control and conversion circuits that use/supply stored information
 - Note that PMBus does not depend on the type of controller
 - analog, digital, hybrid

PMBus Device Concept



How to Make A PMBus Device

Integrated Solution

ASSP

ASIC

Piece Part Solution

Bus Interface

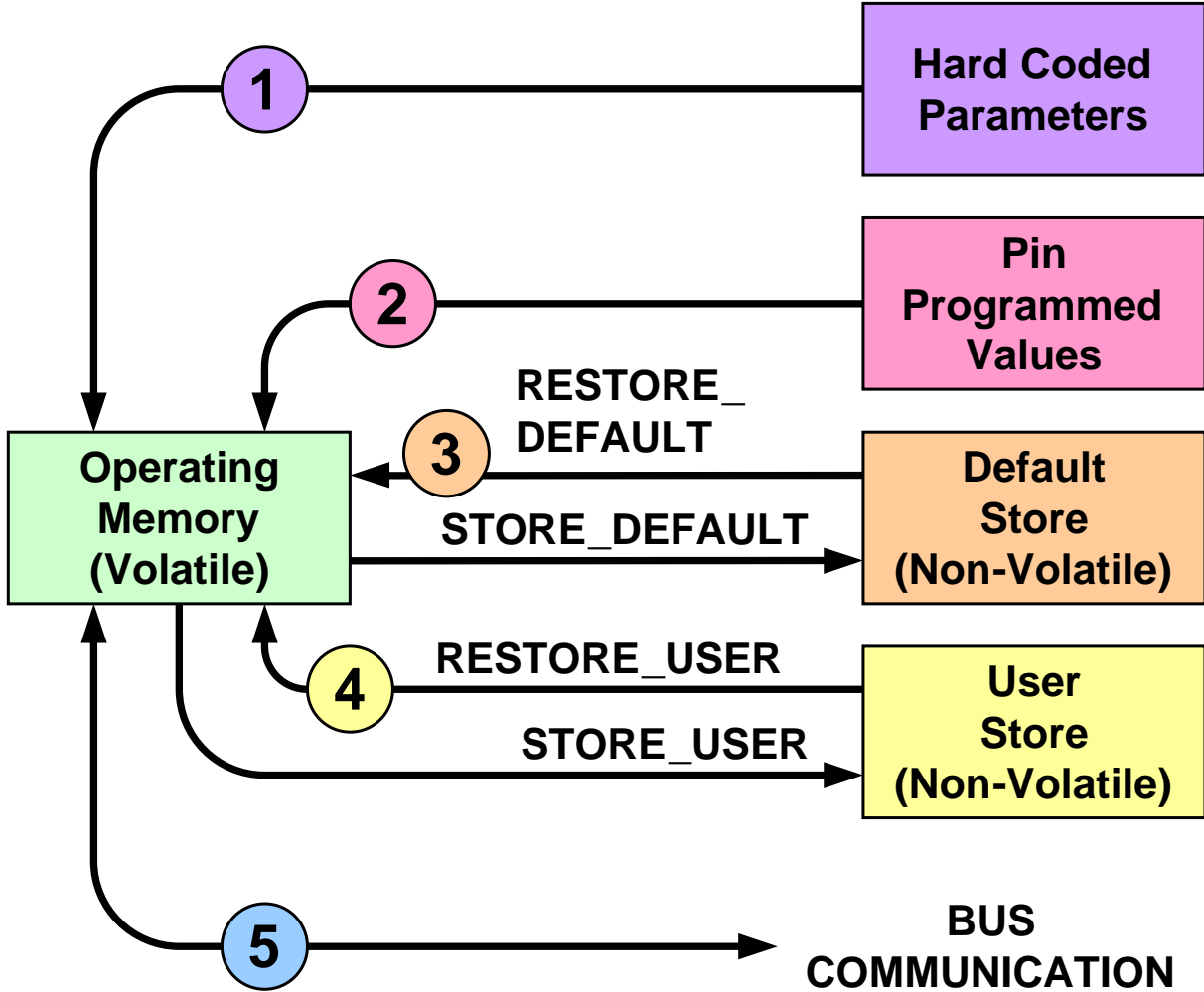
Control & Monitor

ASIC

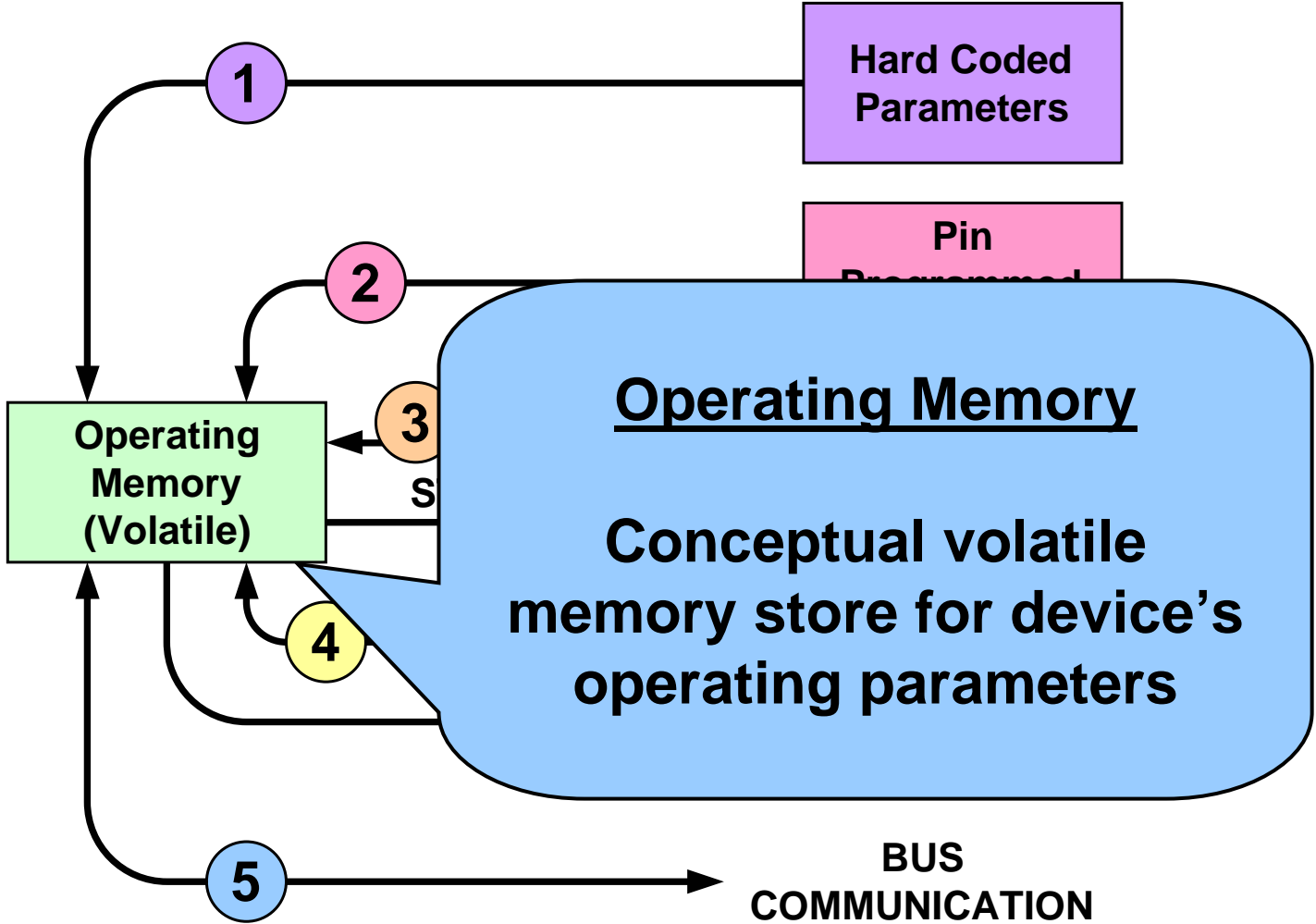
FPGA

GP Microcontroller

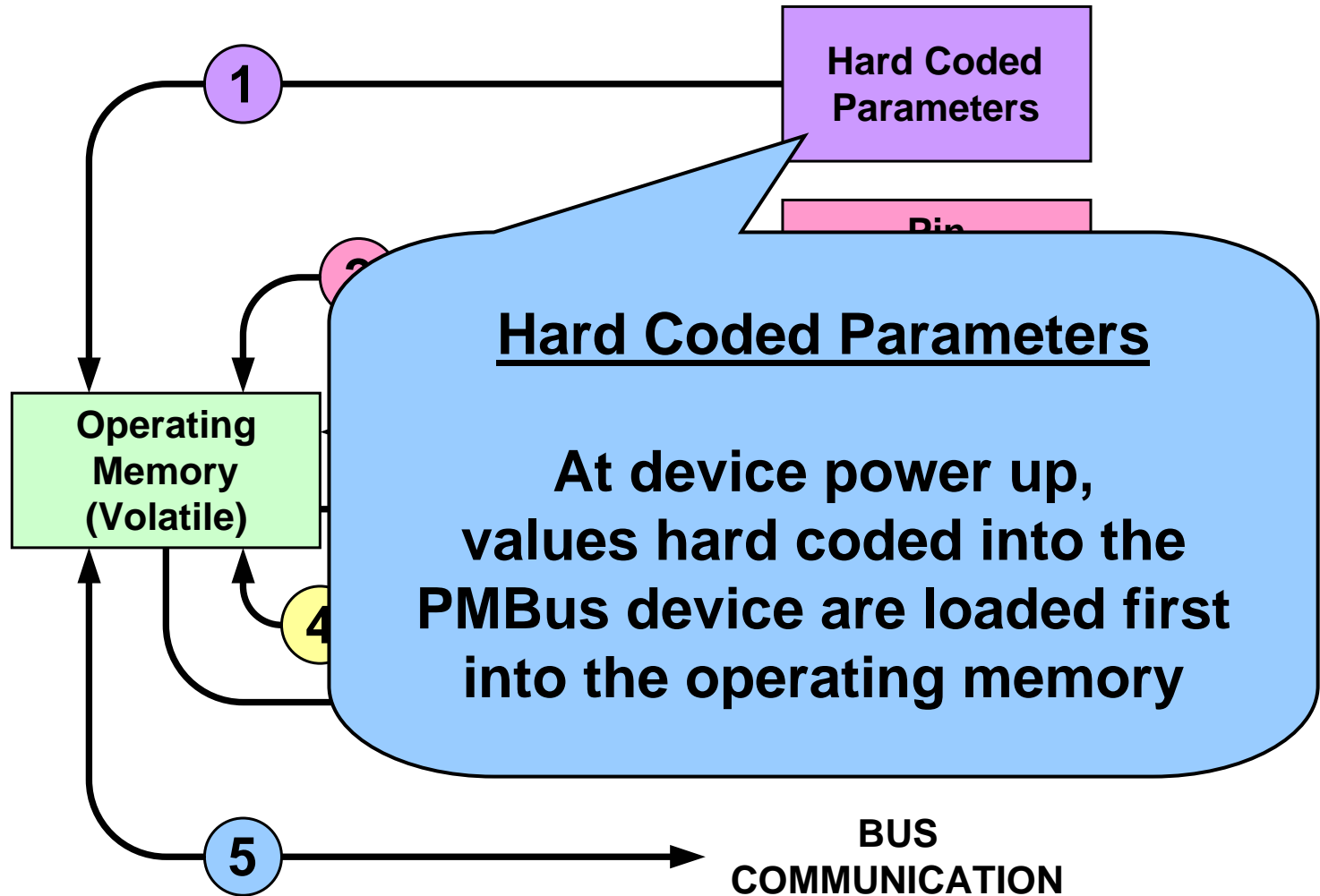
Conceptual View Of Memory And Startup



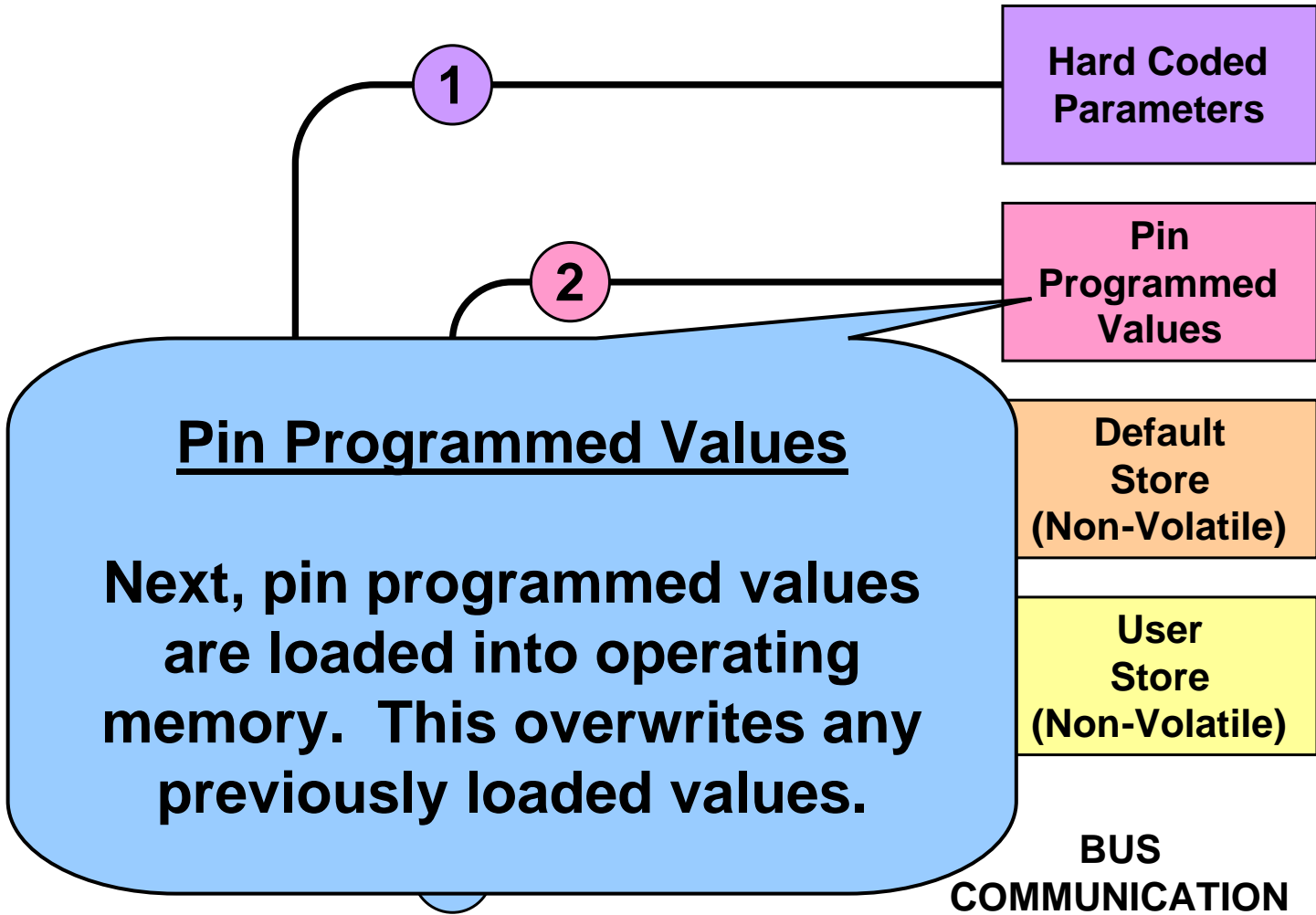
Conceptual View Of Memory And Startup



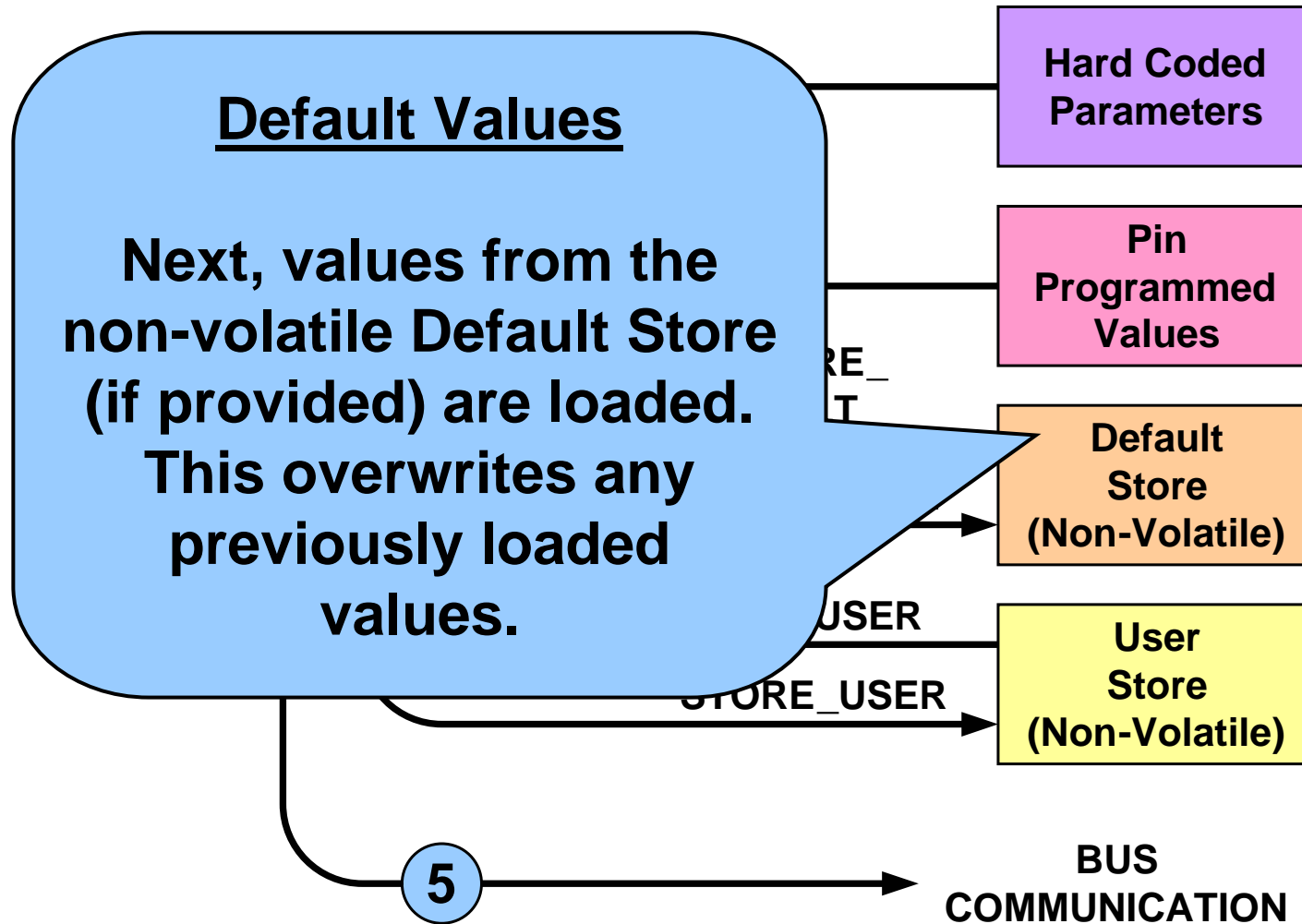
Conceptual View Of Memory And Startup



Conceptual View Of Memory And Startup



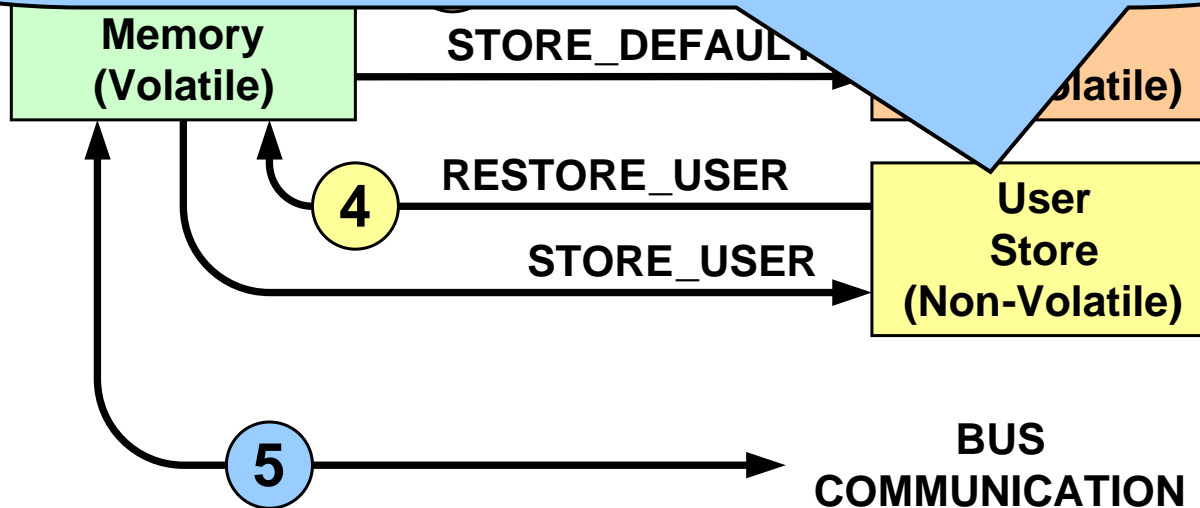
Conceptual View Of Memory And Startup



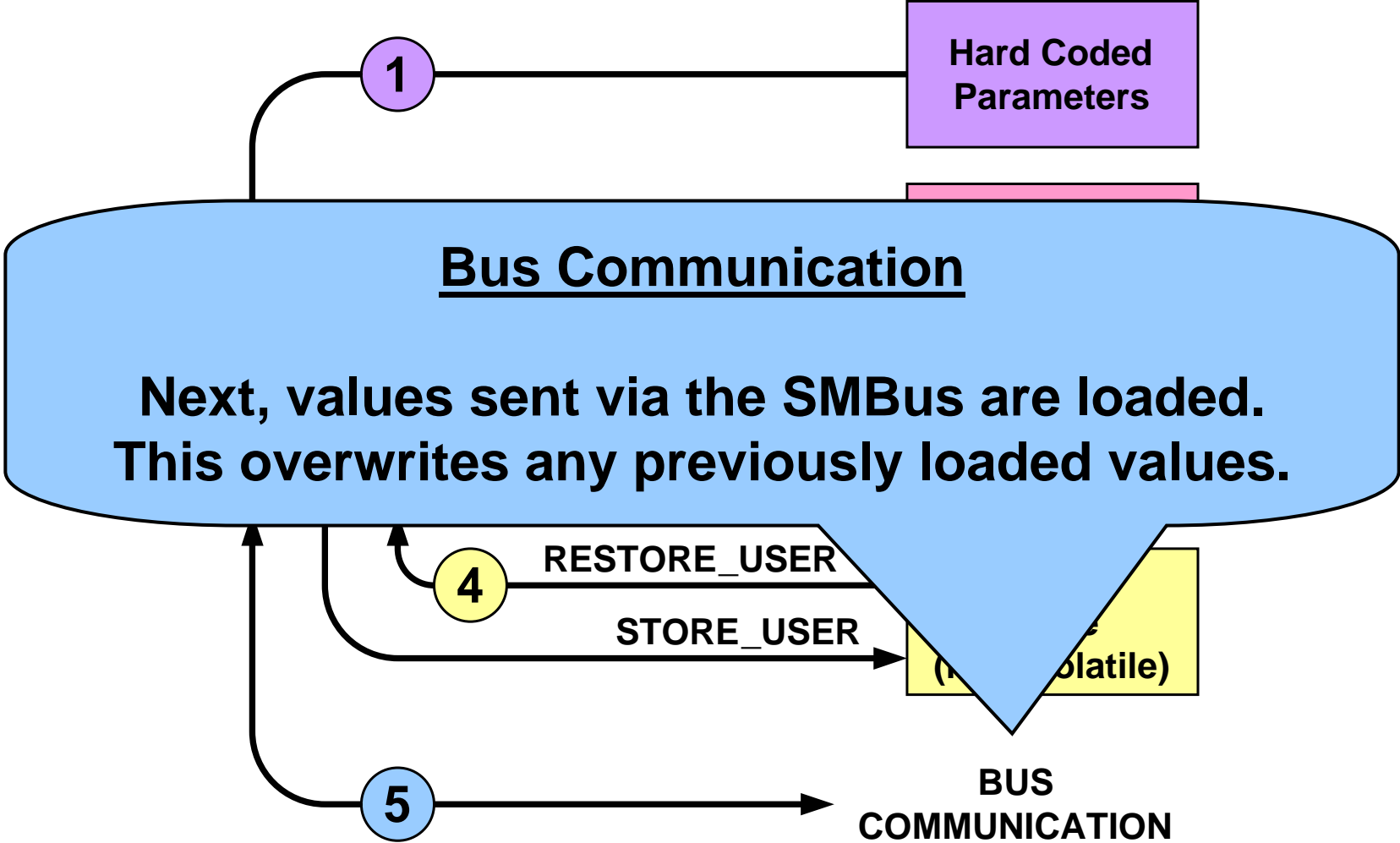
Conceptual View Of Memory And Startup

User Stored Values

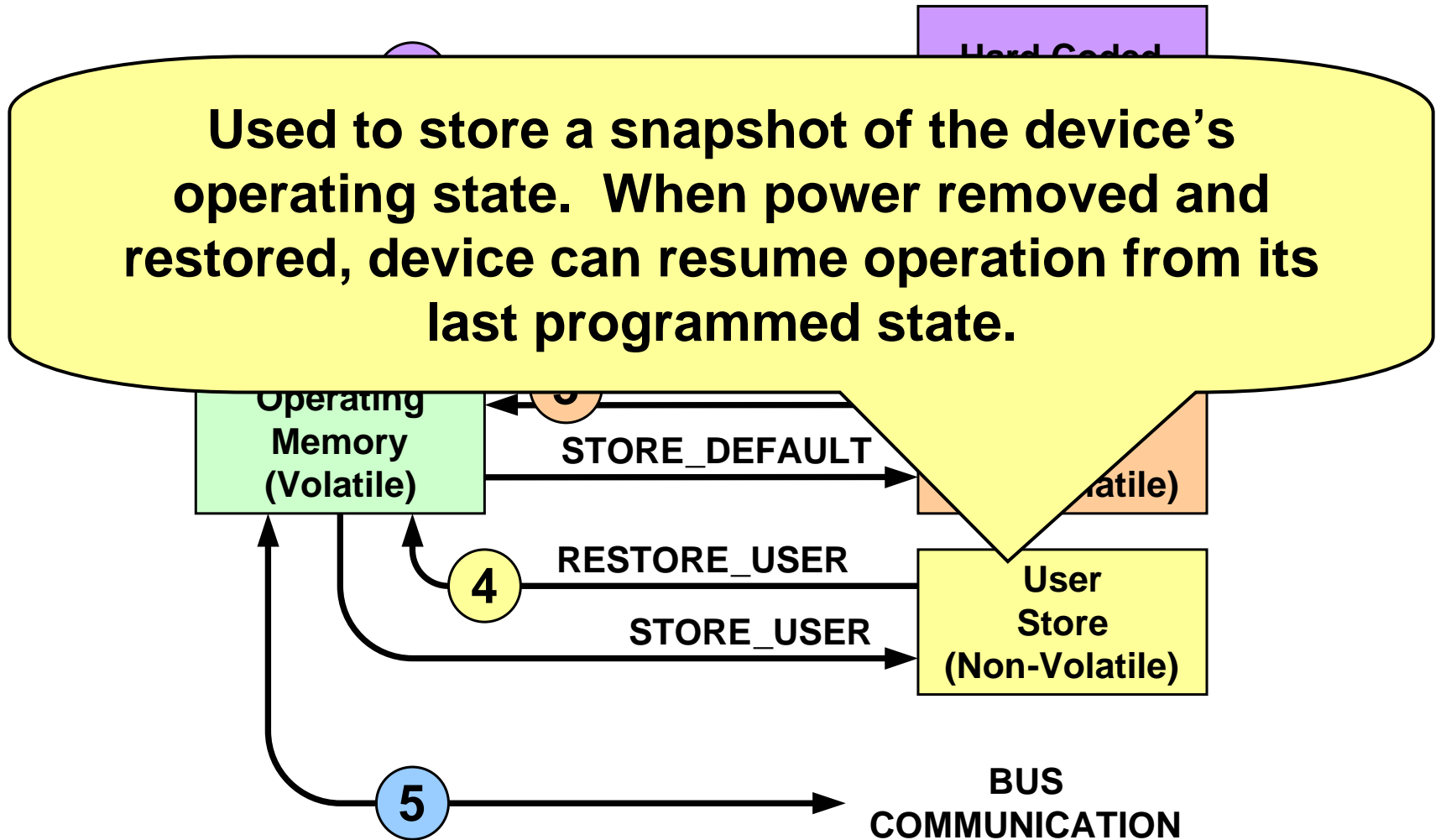
Next, values from the non-volatile User Store (if provided) are loaded. This overwrites any previously loaded values.



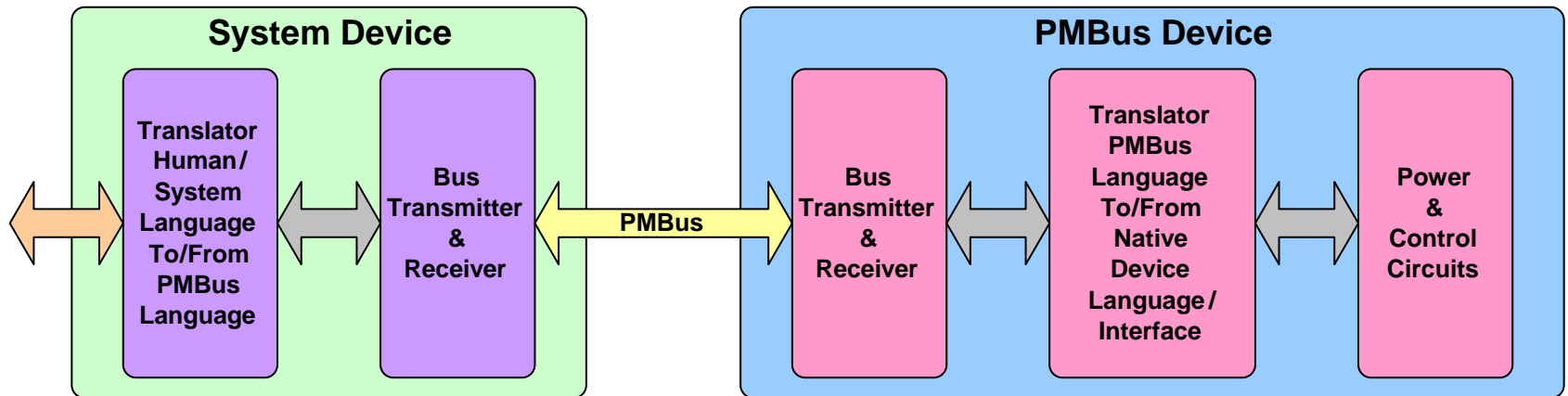
Conceptual View Of Memory And Startup



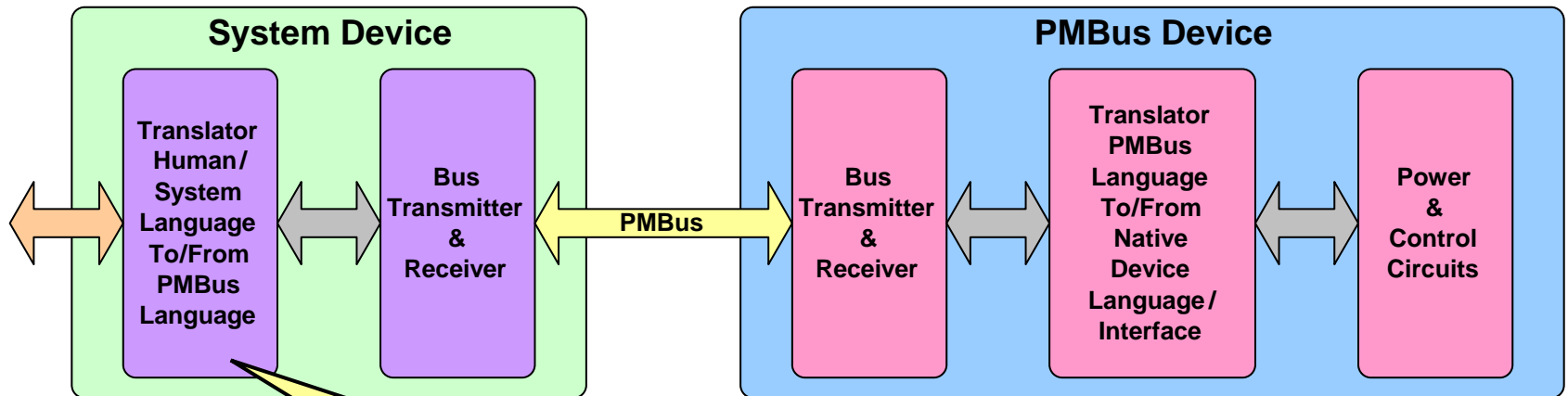
Conceptual View Of Memory And Startup



PMBus-Host Interface

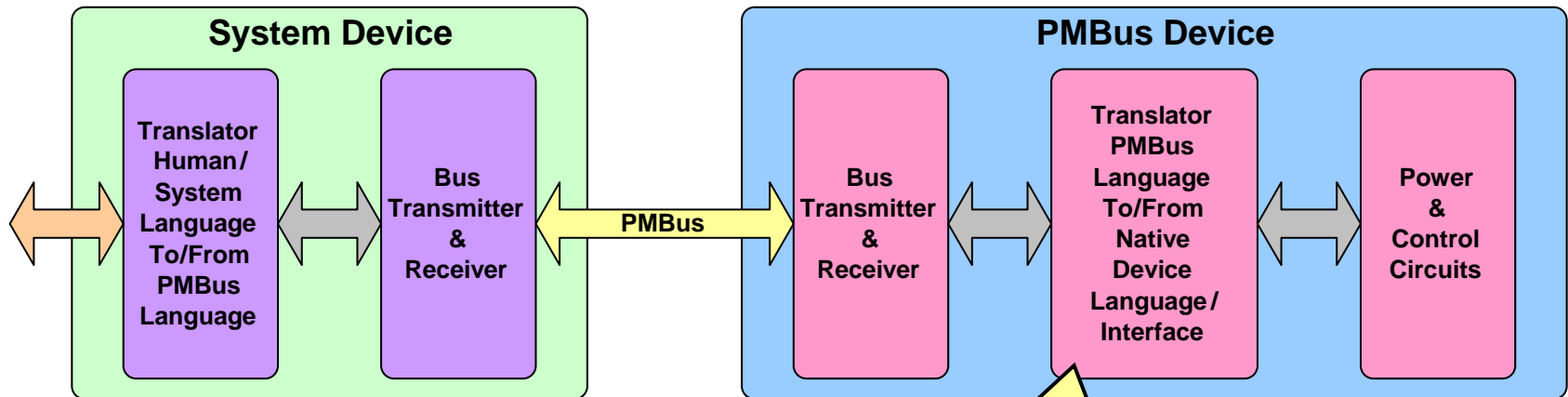


PMBus-Host Interface



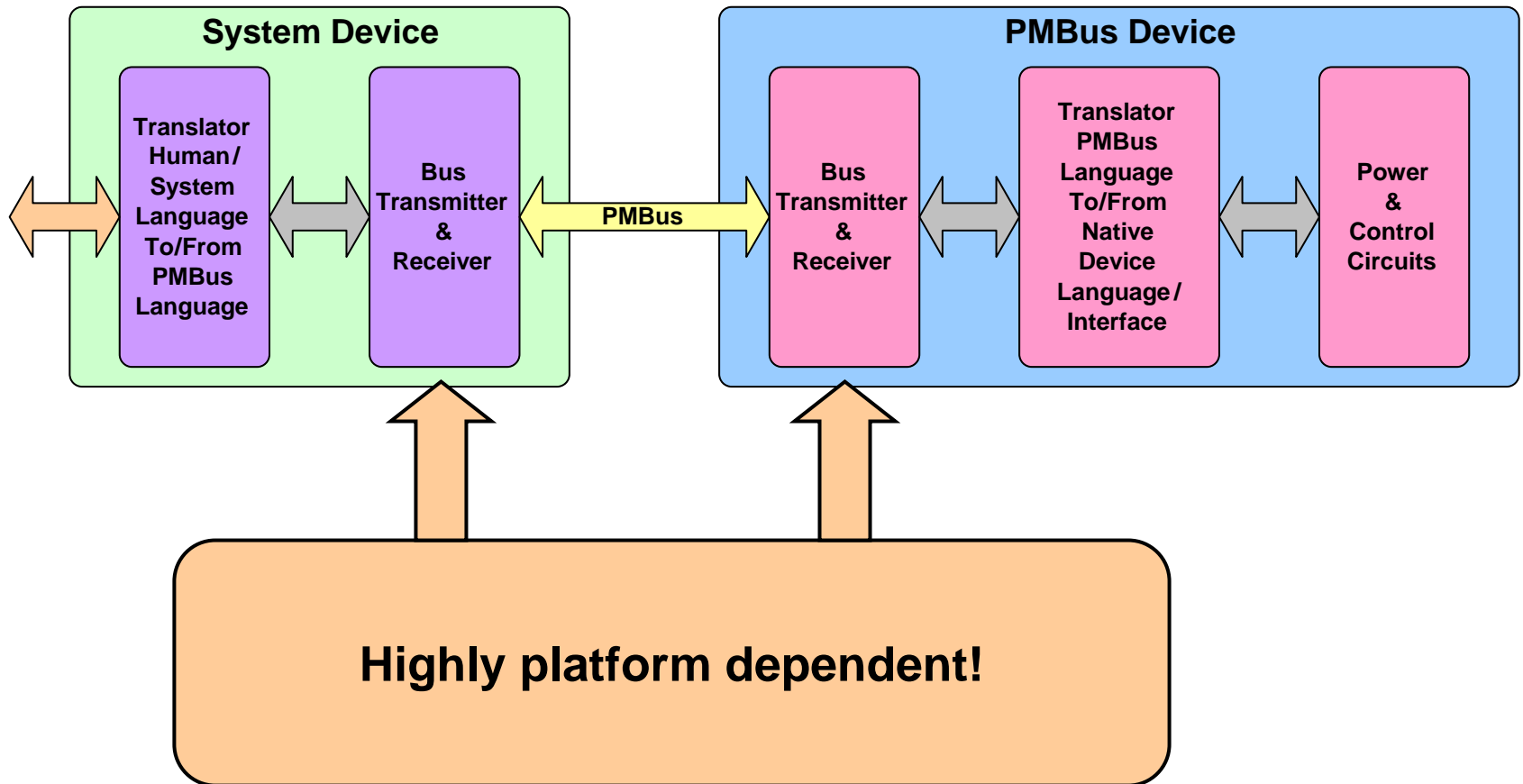
Someone will have to write code for this

PMBus–Host Interface



**Calculation and conversion
for input to D/A converters
and output from A/D converters**

PMBus-Host Interface



PMBus/SMBus Interface

- “Bit banging” with a general purpose I/O port on a microcontroller
 - Can be done – Using interrupts pins can help!
 - Pay attention to the specification
 - Clock synchronization
 - Setup/Hold timing
 - Avoid excessive bus stalls
- Integrated into silicon
 - Many microcontrollers have an I²C port that can be used to drive SMBus
 - General-purpose timers can often be used to implement SMBus time-out features

Command Language

- Commands consist of:
 - A command code
 - 256 command codes (00h To FFh)
 - Zero or more data bytes
- Command code
 - Not a register location!
 - Devices must map command code to memory location themselves
- Data byte(s)
 - Defined in the specification

Data Formats

- More time spent on this by specification working group than any other topic!
- Challenges
 - Wide range of values (millivolts to kiloamperes)
 - Wide range of resolution
 - Millivolts for microprocessors
 - Volts and amperes for AC power
 - Positive and negative values
 - Limited computing power in PMBus devices
 - Think low-cost!

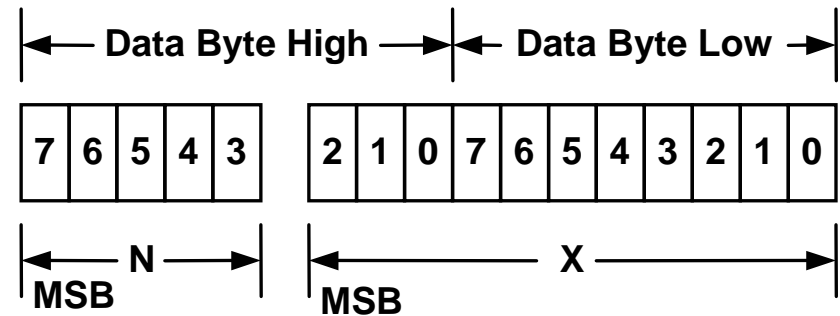
Data Format Choices

- LITERAL – Low-resolution signed data
 - Input voltages and currents
 - Output currents
 - Temperatures
- VOUT_MODE – High-resolution unsigned data
 - All output voltages
- DIRECT – Allows computation requirements to be pushed back into the System Host Device

General Purpose Literal Format

- “Floating Point” format

- 16 bits total
- Mantissa: 11 bit, 2’s complement
- Exponent: 5 bit, 2’s complement



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

- Wide range of values possible

- Maximum positive: $1023 \times 2^{15} = 33.5217 \times 10^6$
- Minimum value: $\pm 1 \times 2^{-16} = \pm 1.526 \times 10^{-5}$
- Maximum negative: $-1024 \times 2^{15} = -33.5544 \times 10^6$

Example Conversion

- $Y = 3.3 \text{ V}$;
Calculate X and N
- Divide maximum value of X by Y
- Find largest power of 2 less than the result. This gives N
- Multiply Y by 2^{-N} and round the result
- Convert to an 11 bit binary number. This gives X .

$$\frac{X_{MAX}}{Y} = \frac{1023}{3.3} = 310.0$$

$$\max(2^N) < 310.0 = 256$$

$$\Rightarrow N = -8 = 11000\text{b}$$

$$3.3 \times 2^{-N} = 3.3 \times 2^8$$

$$= 3.3 \times 256$$

$$= 844.8 \Rightarrow 845$$

$$845 \Rightarrow 01101001101\text{b} = X$$

Example Conversion

- Result – The 16 bit string to be sent to the PMBus device

11000-01101001101

- Think of N as telling the device how many binary places to move the binary point

11.01001101b

- Can think of the binary fraction as sum of powers of 2

$$1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + \dots + 1 \cdot 2^{-8}$$

Example Conversion

- Result – The 16 bit string to be sent to the PMBus device

11000-01101001101

- Think of N as telling the device how many binary places to move the binary point

11.01001101b

- Can think of the binary fraction as sum of powers of 2

$$1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + \dots + 1 \cdot 2^{-8}$$

- Result = 3.30078

Example Conversion

- Result – The 16 bit string to be sent to the PMBus device

11000-01101001101

- Think of N as telling the device how many binary places to move the decimal point

11.01001101b

Error Of (0.0236%) Is From Rounding 844.8 To 845

- Can think of the fraction as sum of powers of 2

$$1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + \dots + 1 \cdot 2^{-8}$$

- Result = 3.30078

Example Decode

■ Received value: 1110001101100111

■ Separate in X And N

$$X = 01101100111b = 871$$

$$N = 11100b = -4$$

■ Calculate Y

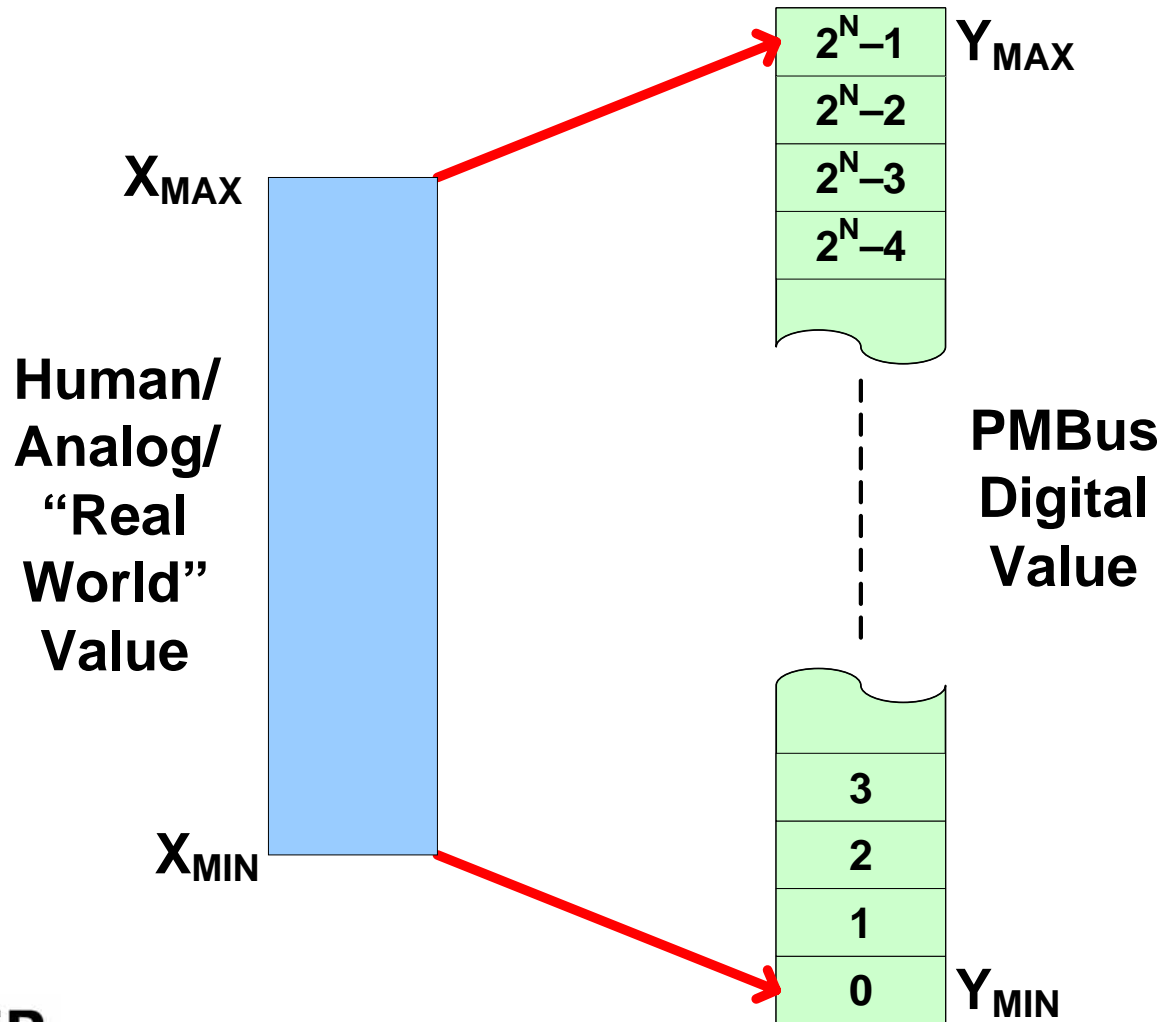
$$Y = X \cdot 2^N = 871 \cdot 2^{-4}$$

$$= 871/16 = 54.438$$

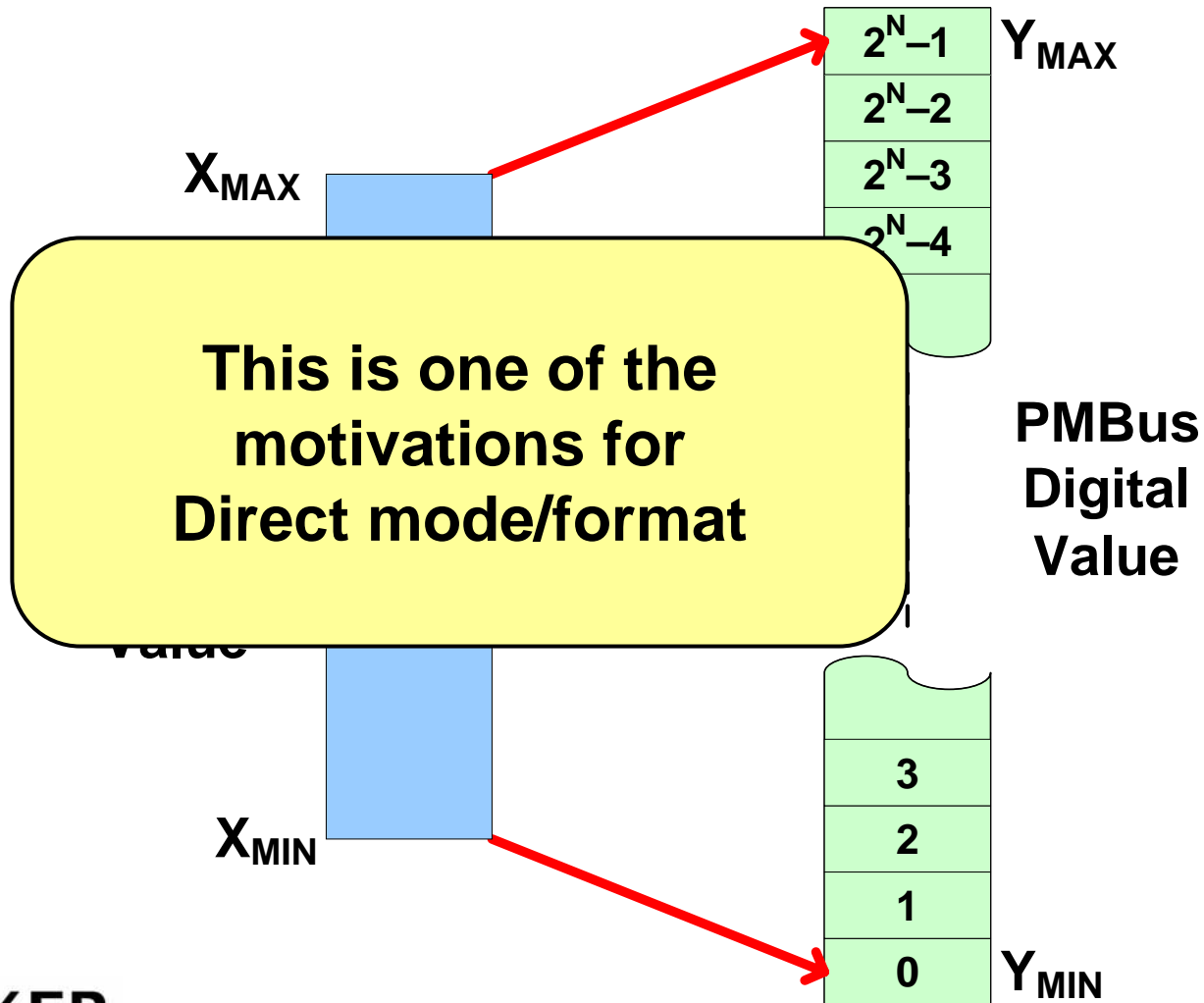
Another Example

- Suppose full scale is 60 V
- Resolution:
 $60 \text{ V} \div 1023 = 58.65 \text{ mV/bit}$
- One application, telecomm rectifier, needs a much finer resolution
 - Typically 10-20 mV/bit
- But range of interest is not 0 V to 60 V, more like 42 V to 58 V

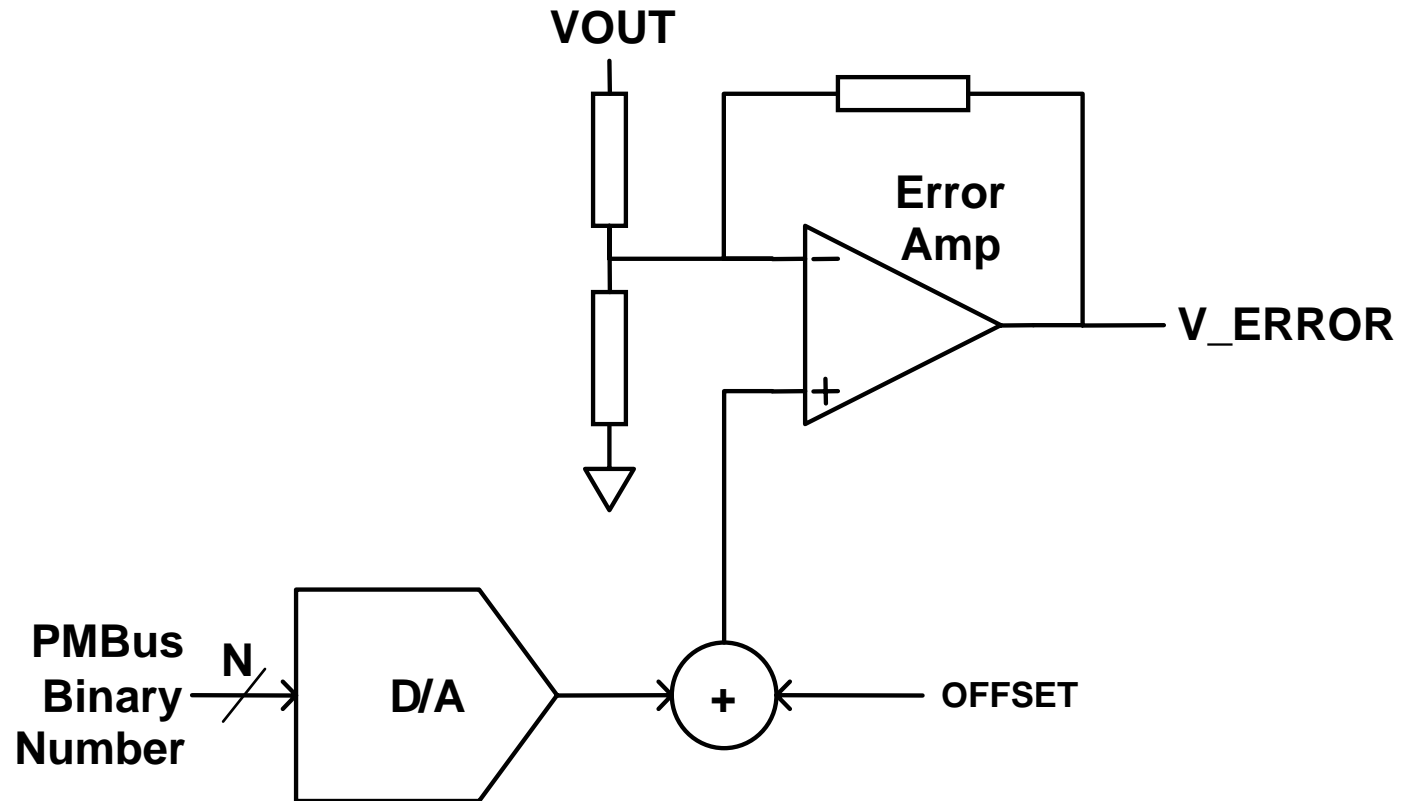
Scaling With Offset



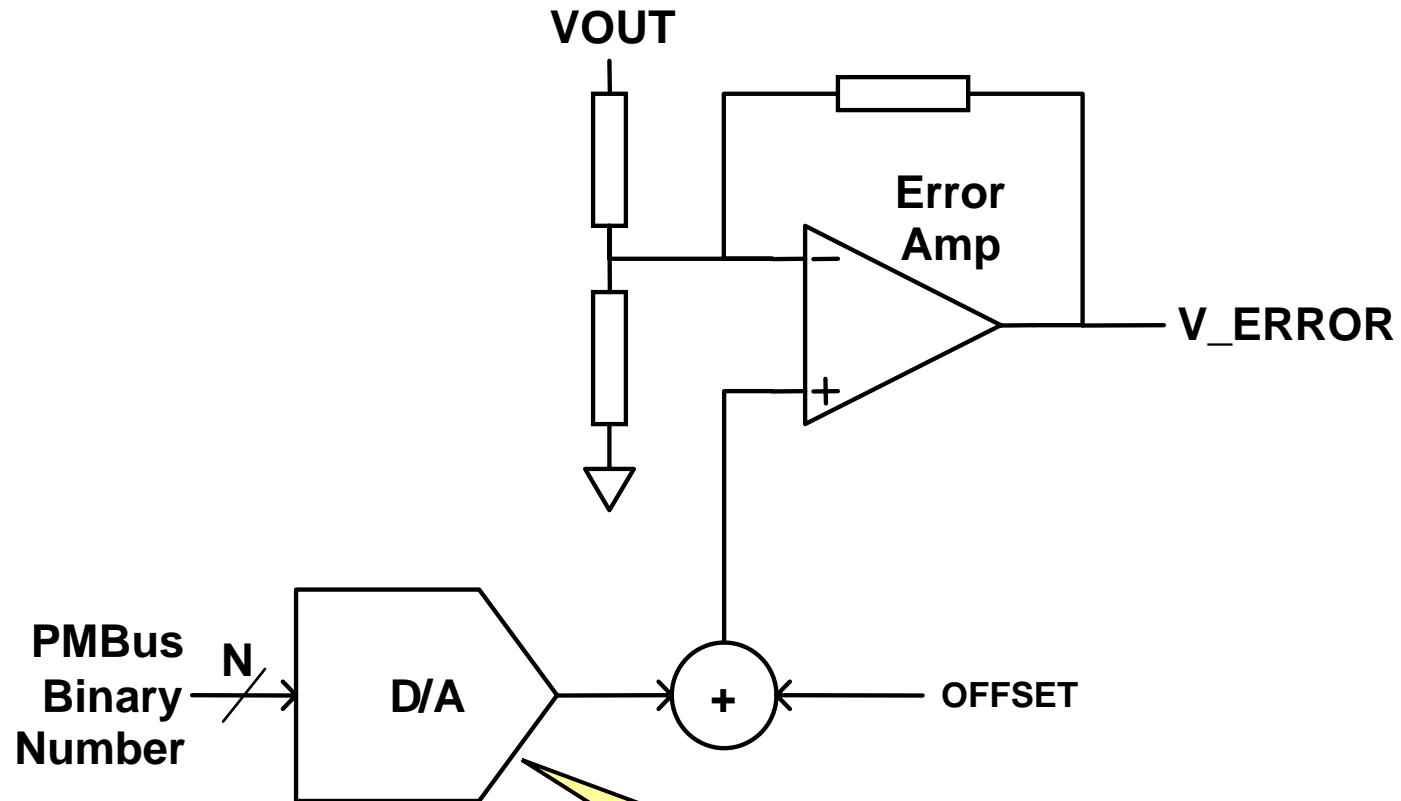
Scaling With Offset



More Direct Mode Motivation



More Direct Mode Motivation



**No "Floating Point"
computation!
Value from PMBus
is used directly**

Direct Mode Equation

- The Direct Mode uses an equation as follows:

$$Y = (mX + b) \cdot 10^R$$

- Where:
 - Y is the value transmitted to or received from the PMBus device (16 bits, signed)
 - X is the “human” value to be encoded
 - m is the scaling coefficient (16 bits, signed)
 - b is the offset coefficient (16 bits, signed)
 - R is the scaling coefficient (8 bits, signed)

Direct Mode Equation

- The Direct Mode Uses An Equation As Follows:

NOTICE!

**This is the form that will appear
in the PMBus Specification Revision 1.1**

**This is “backwards” from what is in
Specification 1.0 Section 7**

Direct Mode: m , b And R

- m , b And R are known as the coefficients
- They are supplied by the PMBus device manufacturer
- Preferred:
Coefficients stored in the device and retrieved by the host with the COEFFICIENTS command
- Alternative:
Coefficients are provided in the product literature (data sheet, application note)

Calculating The Coefficients

- Problem
 - 3 unknowns (m, b, R)
 - 2 constraints
- The two constraints
 - $X_{min} \Rightarrow Y_{min}$ And $X_{max} \Rightarrow Y_{max}$
- Solution procedure
 - Assume R is known and fixed
 - Solve for m and b in terms of $X_{min}, X_{max}, Y_{min}, Y_{max}$
 - Use a tool like Excel to solve for m and b for several values of R
 - Choose largest possible values of m and b

Calculating The Coefficients

- The constraints
- Substituting into the Direct Mode equation
- Solving for m and b

$$X_{\min} \Rightarrow Y_{\min} = 0$$

$$X_{\max} \Rightarrow Y_{\max} = 2^n - 1$$

$$Y_{\min} = (mX_{\min} + b) \cdot 10^R$$

$$Y_{\max} = (mX_{\max} + b) \cdot 10^R$$

$$m = \left(\frac{Y_{\max} - Y_{\min}}{X_{\max} - X_{\min}} \right) \cdot 10^{-R}$$

$$b = \left(Y_{\min} - \frac{Y_{\max} - Y_{\min}}{X_{\max} - X_{\min}} X_{\min} \right) \cdot 10^{-R}$$

$$= \left(Y_{\max} - \frac{Y_{\max} - Y_{\min}}{X_{\max} - X_{\min}} X_{\max} \right) \cdot 10^{-R}$$

Calculating The Coefficients

Example

- AC-DC rectifier for telecom applications
 - Wide range of output voltage to control battery charging
 - Resolution in range of 10–20 mV
- Assume the device contains a 10-bit DAC which spans the range of 44Vdc to 58Vdc.
- Available resolution:
 - Resolution: $(58 - 44)/1024 = 13.67 \text{ mV/bit}$

Calculating The Coefficients Example

- Using Microsoft Excel to solve for m and B for various values of R yields:

R	m (calculated)	b (calculated)	m (rounded)	b (rounded)
-4	730714.2857	-32151428.57	730714	-32151429
-3	73071.42857	-3215142.857	73071	-3215143
-2	7307.142857	-321514.2857	7307	-321514
-1	730.7142857	-32151.42857	731	-32151
0	73.07142857	-3215.142857	73	-3215
1	7.307142857	-321.5142857	7	-322
2	0.730714286	-32.15142857	1	-32
3	0.073071429	-3.215142857	0	-3
4	0.007307143	-0.321514286	0	0
5	0.000730714	-0.032151429	0	0

Calculating The Coefficients Example

- Using Microsoft Excel to solve for m and B for various values of R yields:

R	m (calculated)	b (calculated)	m (rounded)	b (rounded)
-4	730714.2857	-32151428.57	730714	-32151429
-3	73071.42857	-3215142.857	73071	-3215143
-2	7307.142857	-321514.2857	7307	-321514
-1	730.7142857	-32151.42857	731	-32151
0	73.07142857	-3215.142857	73	-3215
1	7.307142857	-321.5142857	7	-322
2	0.730714286	-32.15142857	0	-32
3	0.073071429	-3.215142857	0	-3
4	0.007307143	-0.3215142857	0	-0
5	0.000730714	-0.03215142857	0	0

Values in red exceed the range of values available to a 16 bit signed integer (+32,767 To -32,768)

Calculating The Coefficients Example

- Use m and b for various values of R

For best resolution,
choose largest possible
values of m and b

$$m = 731$$

$$b = -32151$$

$$R = -1$$

R			m (rounded)	b (rounded)
-4			730714	-32151429
-3	7307.142857	-32151.42857	73071	-3215143
-2	7307.142857	-321514.2857	7307	-321514
-1	730.7142857	-32151.42857	731	-32151
0	73.07142857	-3215.142857	73	-3215
1	7.307142857	-321.5142857	7	-322
2	0.730714286	-32.15142857	1	-32
3	0.073071429	-3.215142857	0	-3
4	0.007307143	-0.321514286	0	0
5	0.000730714	-0.032151429	0	0

Calculating The Coefficients Example

- Chosen solution

$$m = 731$$

$$b = 32151$$

$$R = -1$$

- Double check calculation

$$\begin{aligned} Y_{\min} &= (mX_{\min} + b) \cdot 10^R \\ &= (731 \cdot 44 - 32,151) \cdot 10^{-1} \\ &= 1.3 \neq 0 \end{aligned}$$

$$\begin{aligned} Y_{\max} &= (mX_{\max} + b) \cdot 10^R \\ &= (731 \cdot 58 - 32,151) \cdot 10^{-1} \\ &= 1024.7 \neq 1023 \end{aligned}$$

More Rounding Errors!

Calculating The Coefficients Example

- Example
 - Minimum voltage (X_{min}): 44 V
 - Maximum voltage (X_{max}): 58 V
 - PMBus device resolution: 16 bits

R	m (calculated)	b (calculated)	m (rounded)	b (rounded)
-2	468107.1429	-20596714.29	468107	-20596714
-1	46810.71429	-2059671.429	46811	-2059671
0	4681.071429	-205967.1429	4681	-205967
1	468.1071429	-20596.71429	468	-20597
2	46.81071429	-2059.671429	47	-2060
3	4.681071429	-205.9671429	5	-206
4	0.468107143	-20.59671429	0	-21

Calculating The Coefficients Example

- Chosen solution

$$m = 468$$

$$b = -20597$$

$$R = 1$$

- Double check calculation

$$\begin{aligned} Y_{\min} &= (mX_{\min} + b) \cdot 10^R \\ &= (468 \cdot 44 - 20,597) \cdot 10^{+1} \\ &= -5 \neq 0 \end{aligned}$$

$$\begin{aligned} Y_{\max} &= (mX_{\max} + b) \cdot 10^R \\ &= (468 \cdot 58 - 20,597) \cdot 10^{+1} \\ &= 65,470 \neq 65,535 \end{aligned}$$

Still Have Rounding Errors!

Decoding Direct Mode Example

- Example of reading the output current of an isolated DC-DC bus converter
- Using COEFFICIENTS command returns values for m , b and R as:
 - $m = 850$
 - $b = 0$
 - $R = -2$
- Using READ_IOUT command returns the value 0000000001101001b \Rightarrow 105d

Decoding Direct Mode Example

- Use the inverse of the equation used to encode
- Substitute values and solve
- Output current = 12.35 A

$$Y = (mX + b) \cdot 10^R$$

$$X = \frac{1}{m} (Y \cdot 10^{-R} - b)$$

$$\begin{aligned} X &= \frac{1}{850} (105 \cdot 10^{-(2)} - 0) \\ &= \frac{10500}{850} = 12.35 \end{aligned}$$

Note that these calculations are done in the host and not the PMBus device!

Setting The Output Voltage

Step 1
Which data format?
(aka which mode)

VOUT_MODE command

Linear

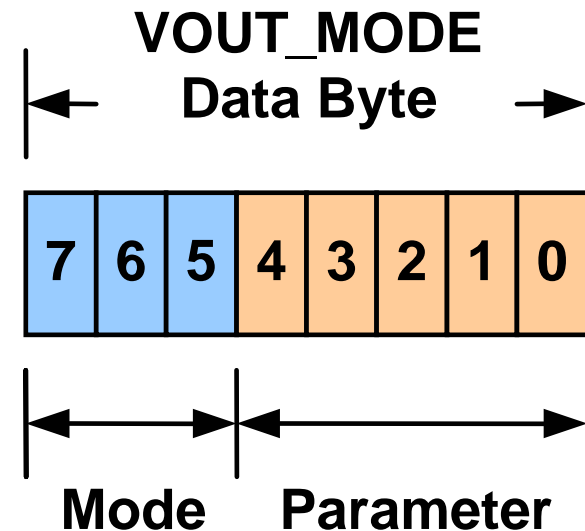
Direct

VID

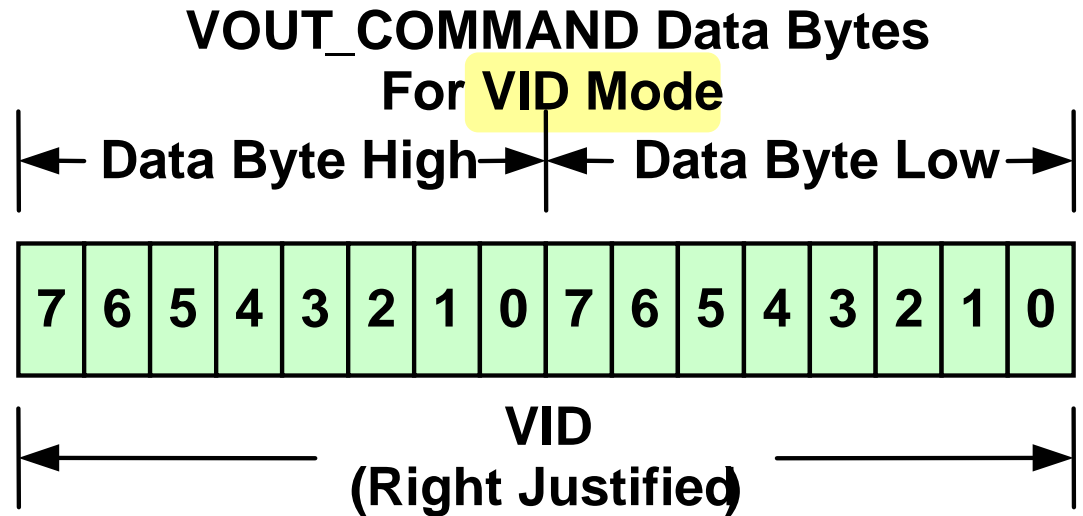
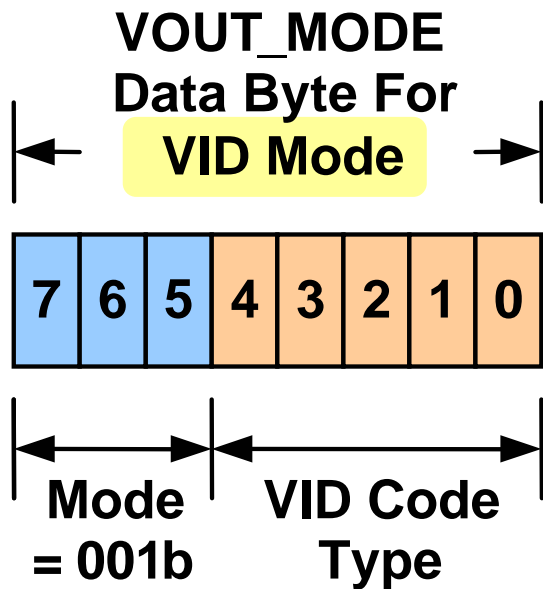
Step 2
Set the output voltage using the
VOUT_COMMAND command

VOUT_MODE Command

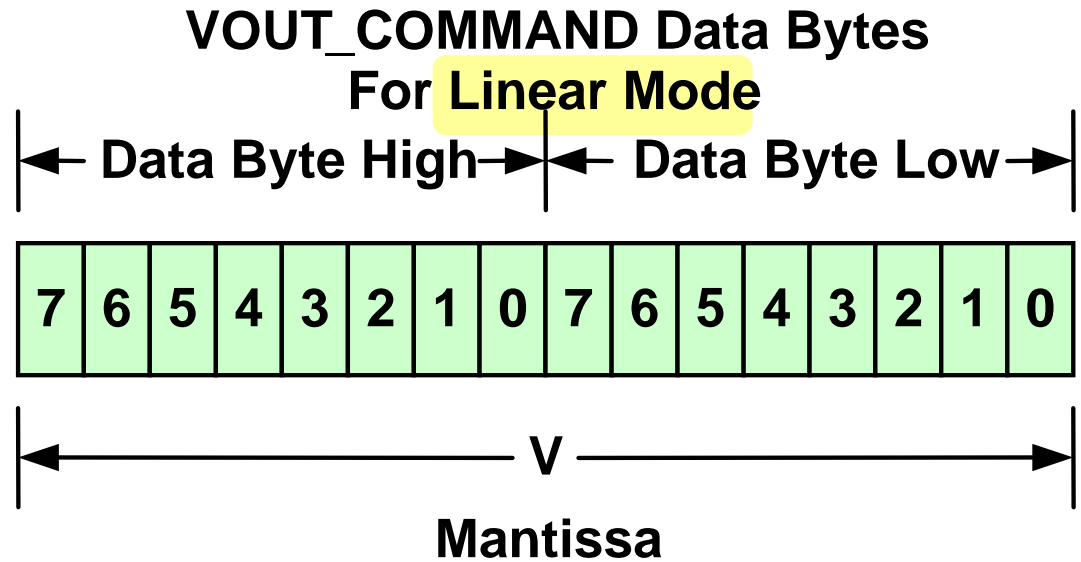
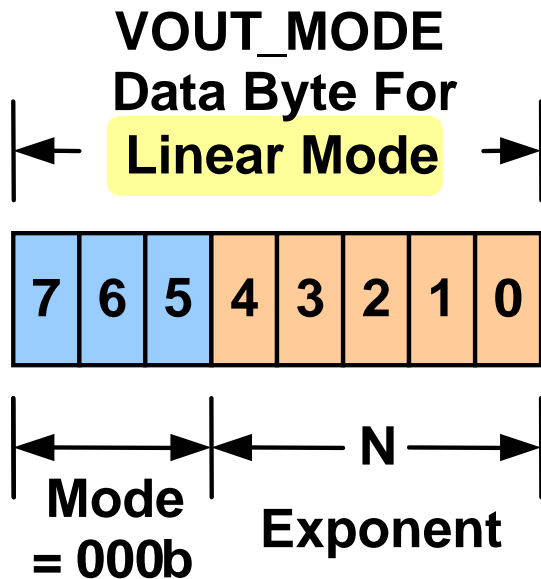
- VOUT_MODE command is sent separately from any other command, such as VOUT_COMMAND
- Sent only when necessary to change the mode
 - Only once?
 - Often VOUT_MODE will be read-only
- Applies for all output voltage related commands



VOUT_MODE & VOUT_COMMAND

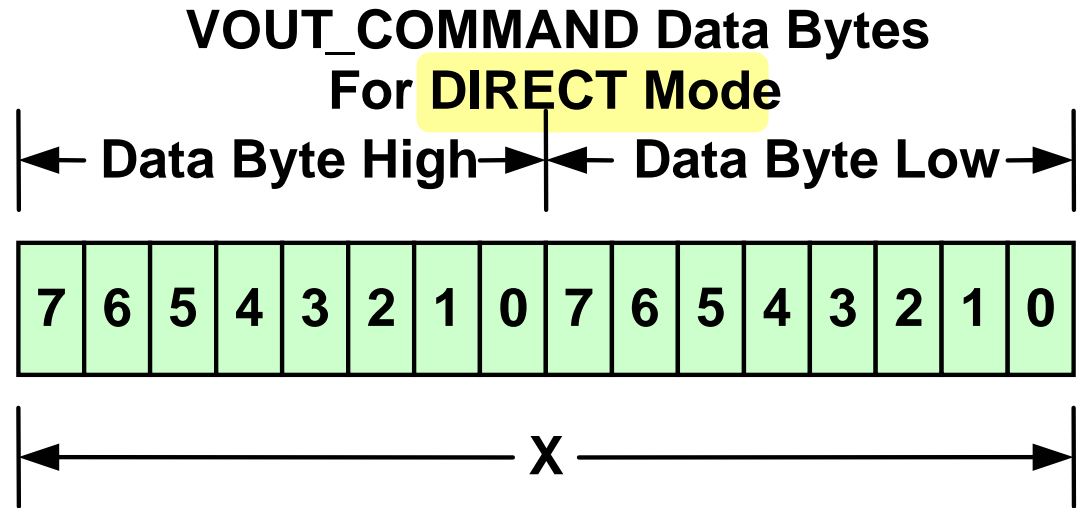
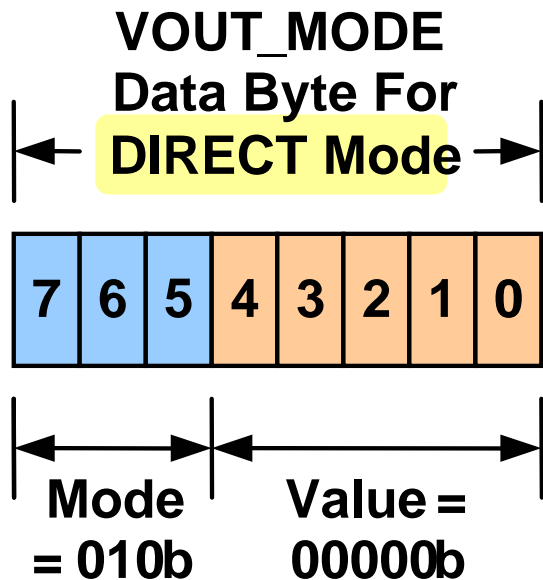


VOUT_MODE & VOUT_COMMAND

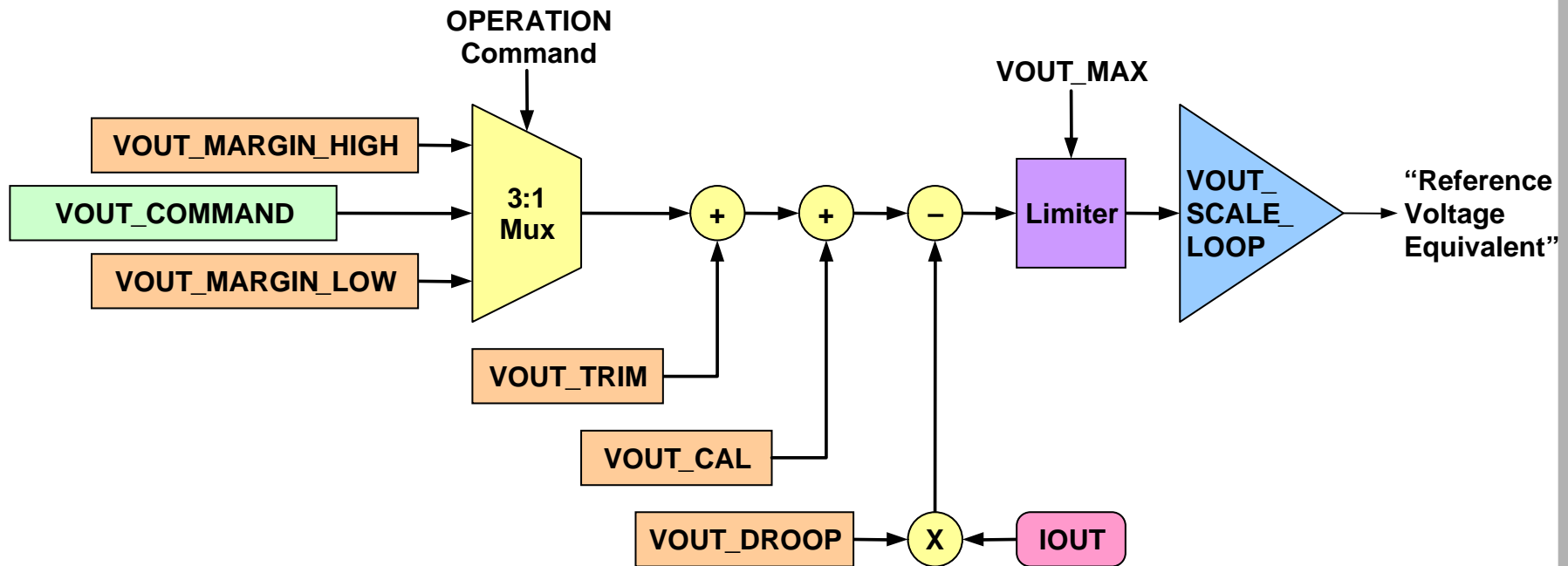


$$V_{out} = V \cdot 2^N$$

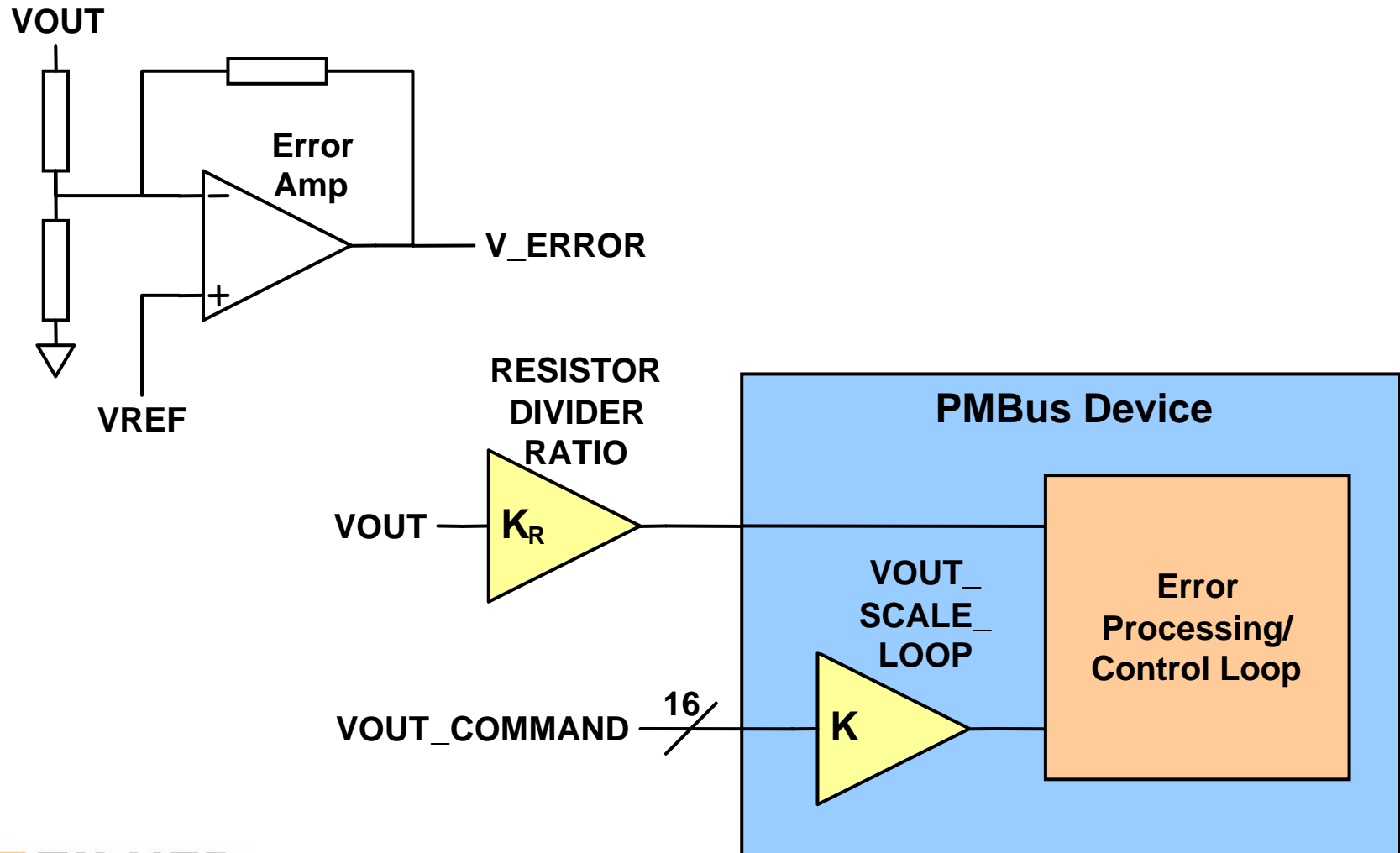
VOUT_MODE & VOUT_COMMAND



Fine Tuning The Output Voltage



Using And External Divider

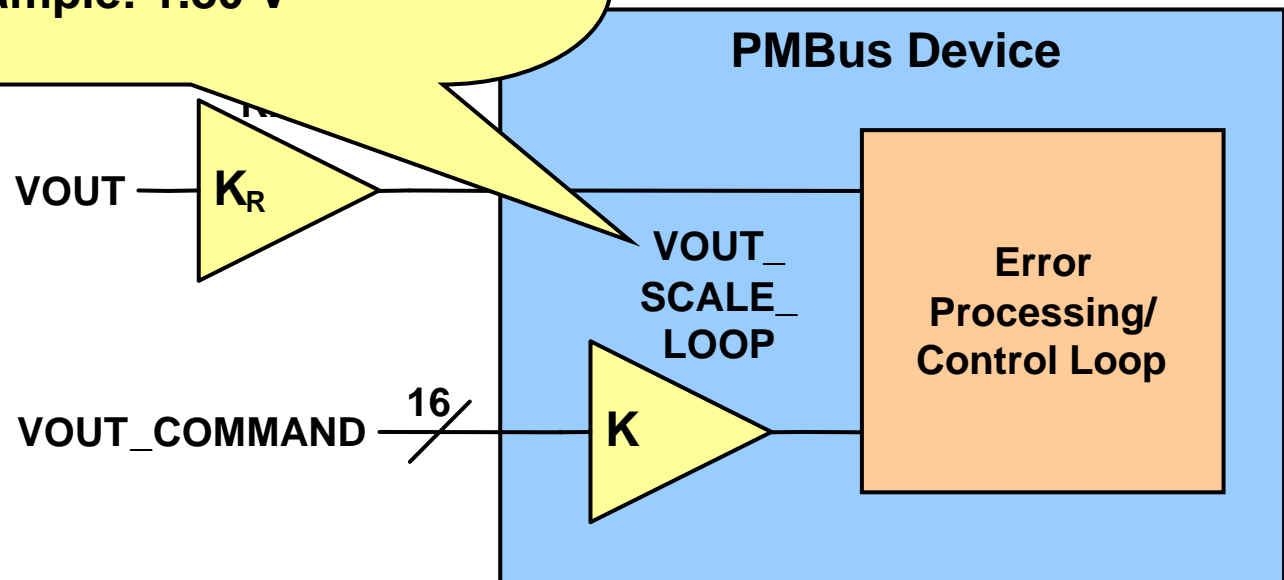


Using And External Divider

**Simplifies life
for the end user**

**They do not need to think about the
voltage divider**

**Just send command voltage as they
want it
Example: 1.80 V**



On/Off Control

- Two inputs control whether a PMBus device is operating or not
 - Hardwired CONTROL pin (programmable polarity)
 - OPERATION command from the bus
- On/Off control totally programmable
- CONTROL pin options
 - Active high or active low
 - Followed programmed ramp-down or disable immediately

ON_OFF_CONFIG

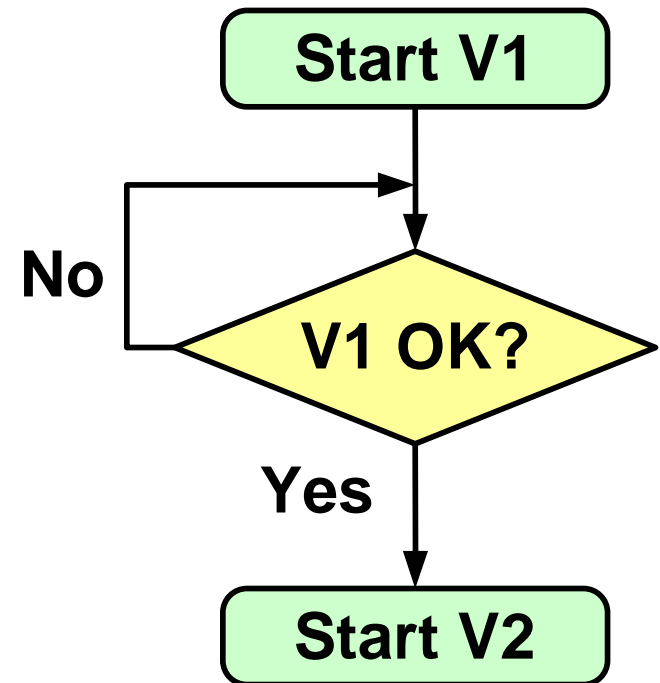
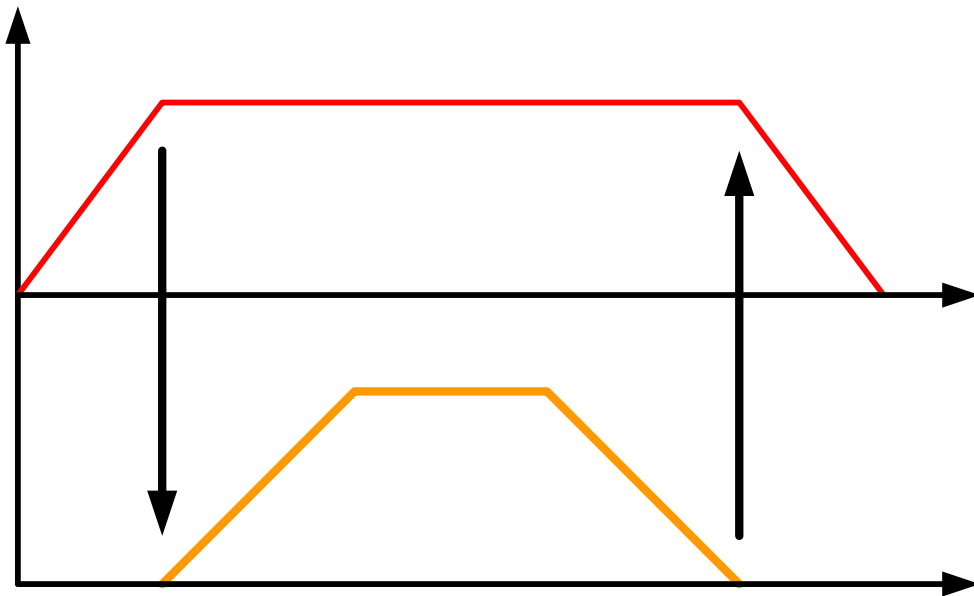
On/Off Control Mode	Device Power	CONTROL Input	Bus Command
Always ON	If Power, Then ON	X	X
Respond To CONTROL Only	If Power, Respond To CONTROL And Bus Commands As Programmed	Active High	Ignore Bus Commands
		Active Low	
Respond To Bus Only		Ignore CONTROL	Respond To Bus Commands
Respond To Both CONTROL And Bus		Active High	
	Active Low		

OPERATION Command

Bits [7:6]	Bits [5:4]	Bits [3:2]	Bits [1:0]	Unit On Or Off	Margin State
00	XX	XX	XX	IMMEDIATE OFF (No Sequencing)	N/A
01	XX	XX	XX	OFF (With Sequencing)	N/A
10	00	XX	XX	ON	OFF
10	01	01	XX	ON	MARGIN LOW (Ignore Fault)
10	01	10	XX	ON	MARGIN LOW (Act On Fault)
10	10	01	XX	ON	MARGIN HIGH (Ignore Fault)
10	10	10	XX	ON	MARGIN HIGH (Act On Fault)

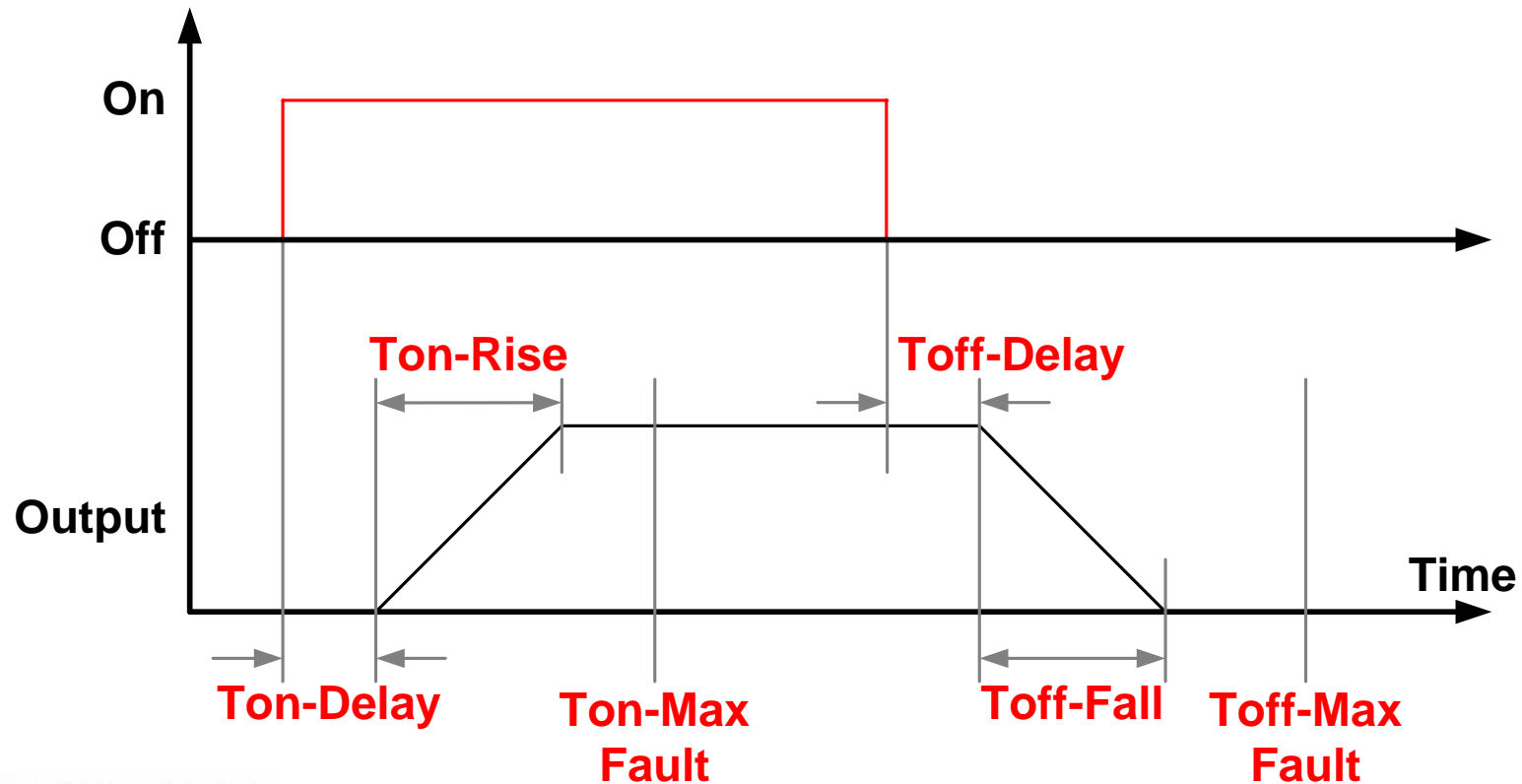
Sequencing: Event Driven

- Event driven sequencing is closed loop
- Requires power system manager to close the loop

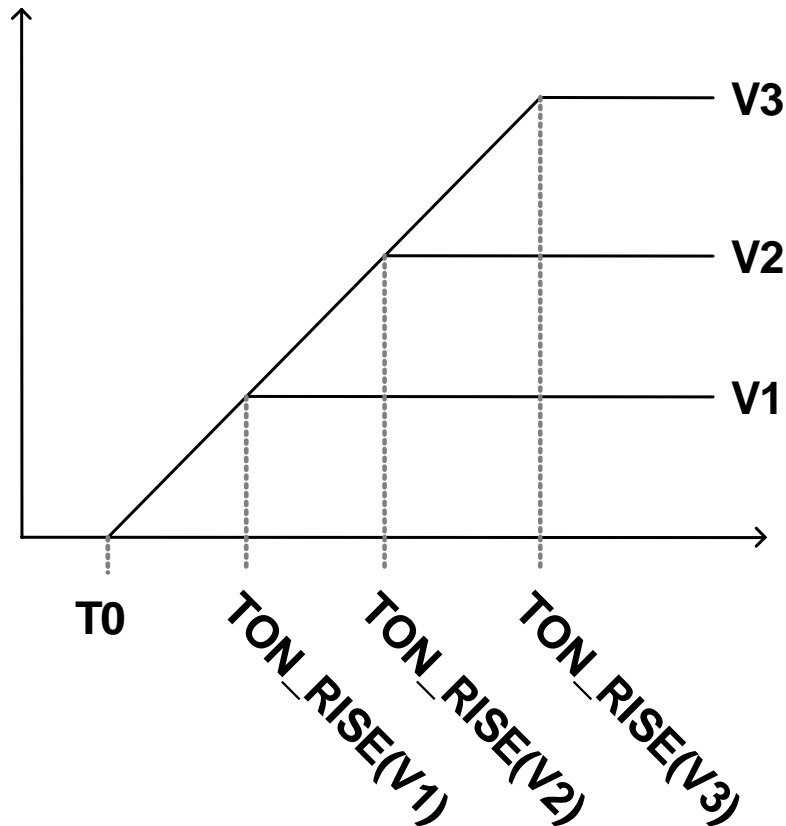


Sequencing: Time Driven Commands

- Open loop: does not require power system manager



Open Loop Tracking



- To implement an open loop tracking, need to know:
 - Each output voltage
 - Desired rise time (TON_RISE) for just one output voltage
- Calculate TON_RISE of all other outputs as follows:

$$TON_RISE(V2) = TON_RISE(V1) \cdot \frac{V2}{V1}$$

$$TON_RISE(V3) = TON_RISE(V1) \cdot \frac{V3}{V1}$$

- Use CONTROL or Group operation

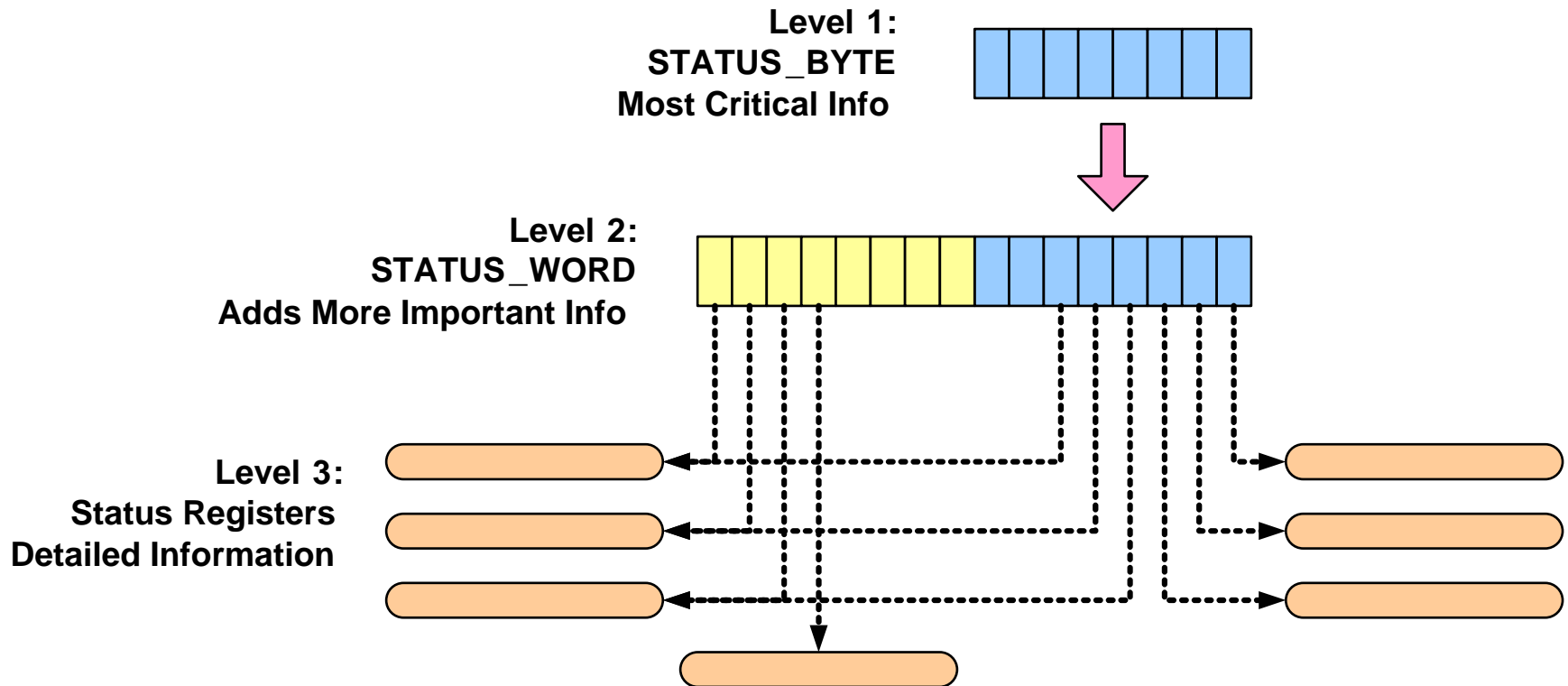
Status Reporting And Fault Management

- The PMBus protocol supports two alarm levels
 - Warnings (minor alarms)
 - Faults (major alarms)
- Warnings only result in host being notified that attention is needed
- Faults cause the PMBus device to respond and take action internally as programmed
- Parametric information (e.g. voltage) can also be read from PMBus devices

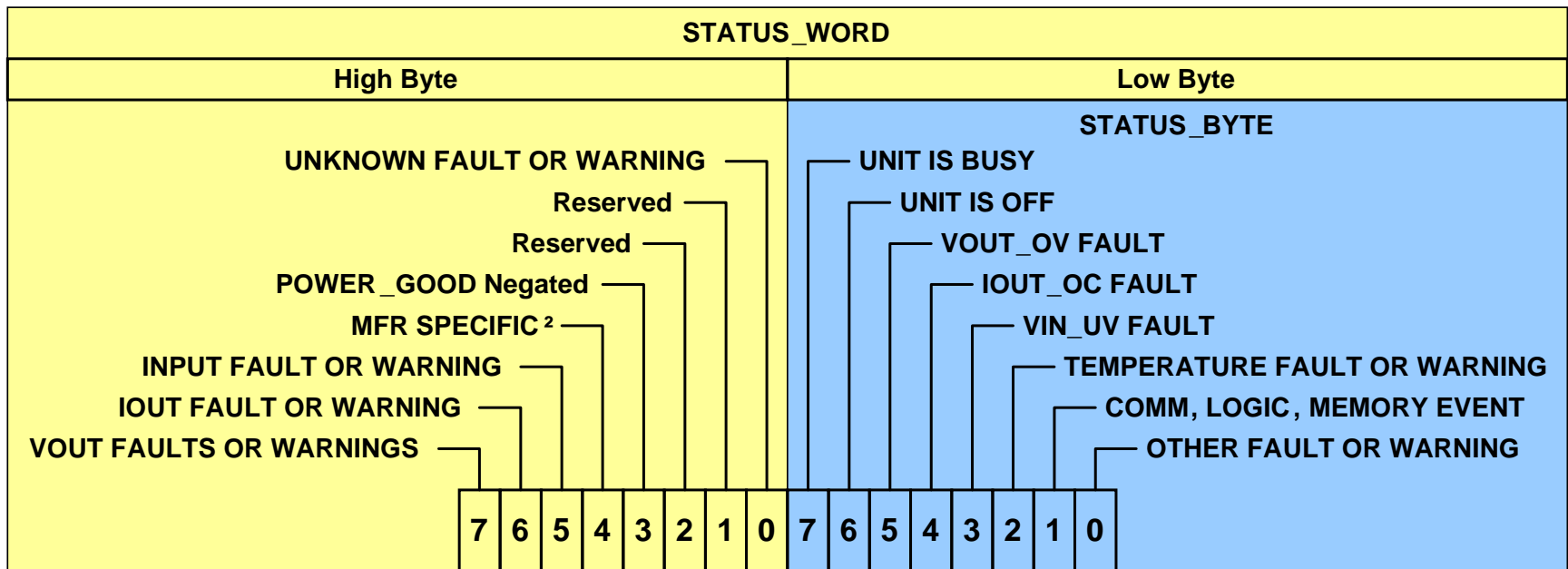
Notifying The Host Of A Fault

- Host can continuously poll PMBus devices
- PMBus device can send an interrupt
 - SMBALERT# signal is optional
 - See the SMBus Specification for details
- PMBus device can become a bus master and transmit notice to system host
 - Optional
 - Requires a more sophisticated host and more sophisticated PMBus devices

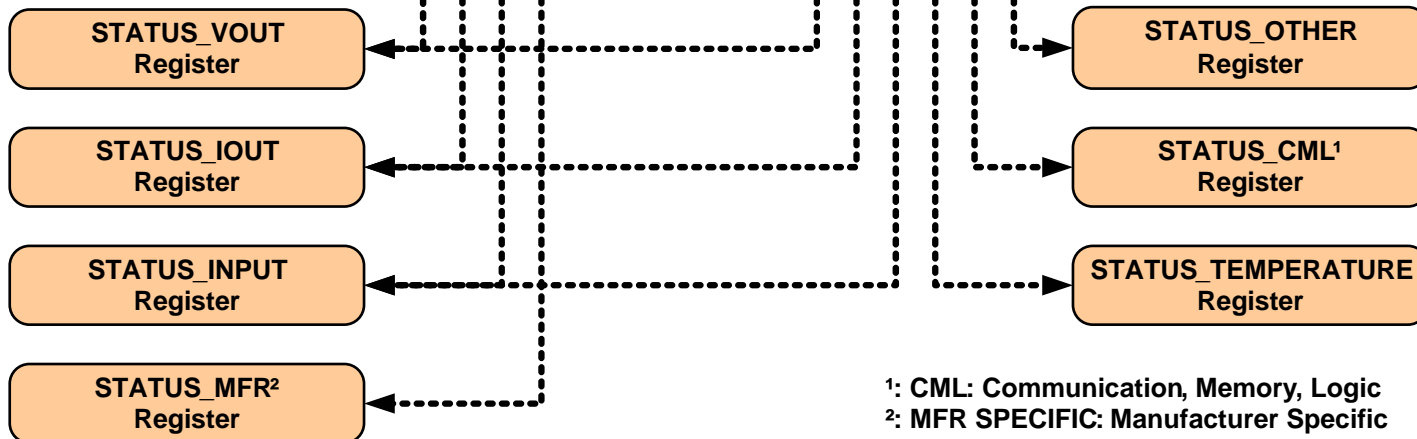
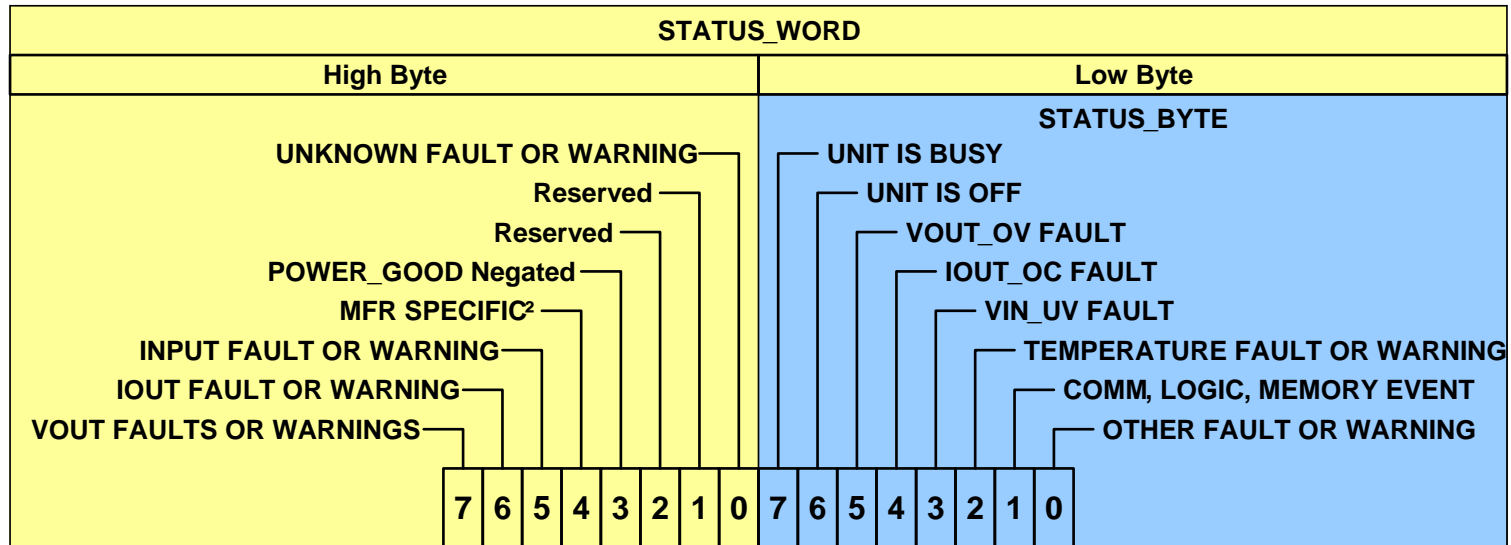
Status Reporting: 3 Levels Of Detail



STATUS_BYTE & STATUS_WORD



Status Registers

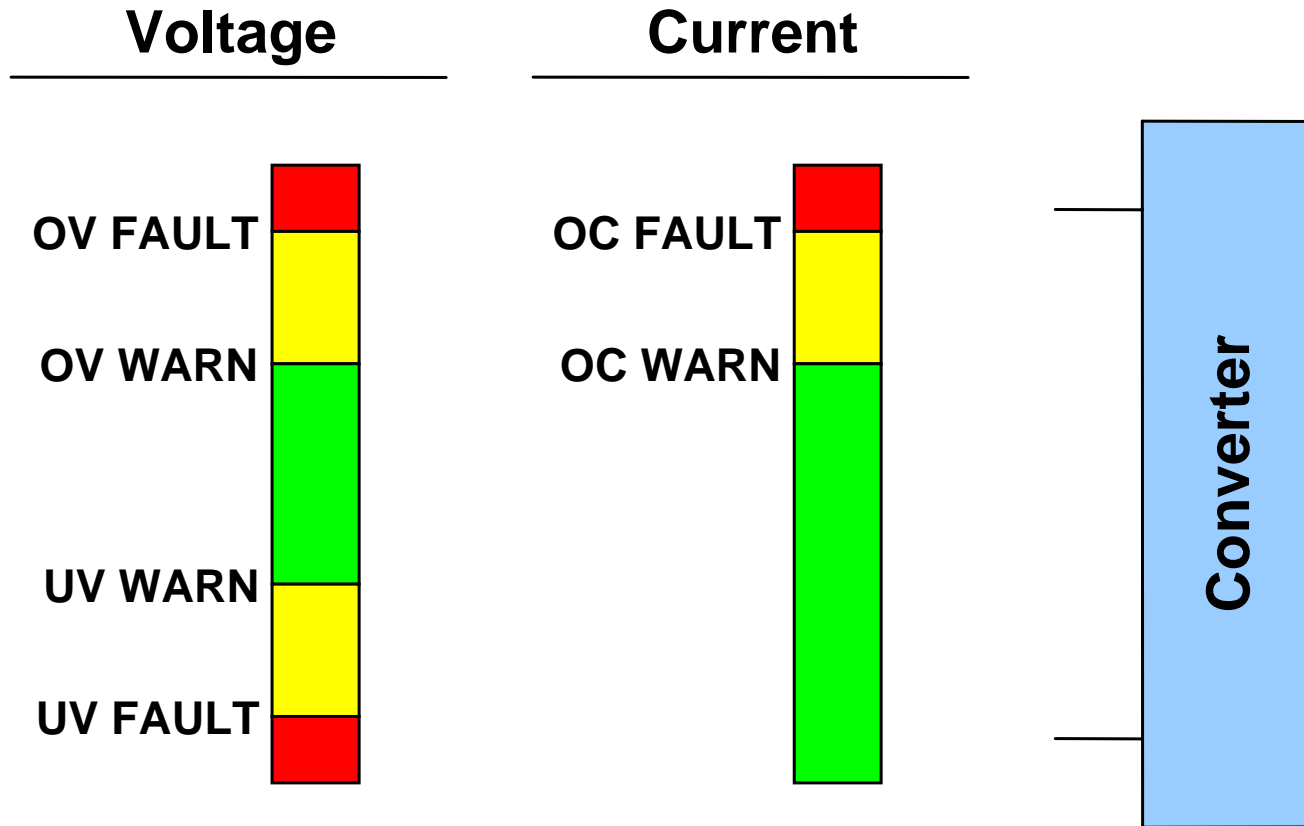


¹: CML: Communication, Memory, Logic
²: MFR SPECIFIC: Manufacturer Specific

Clearing Status Bits

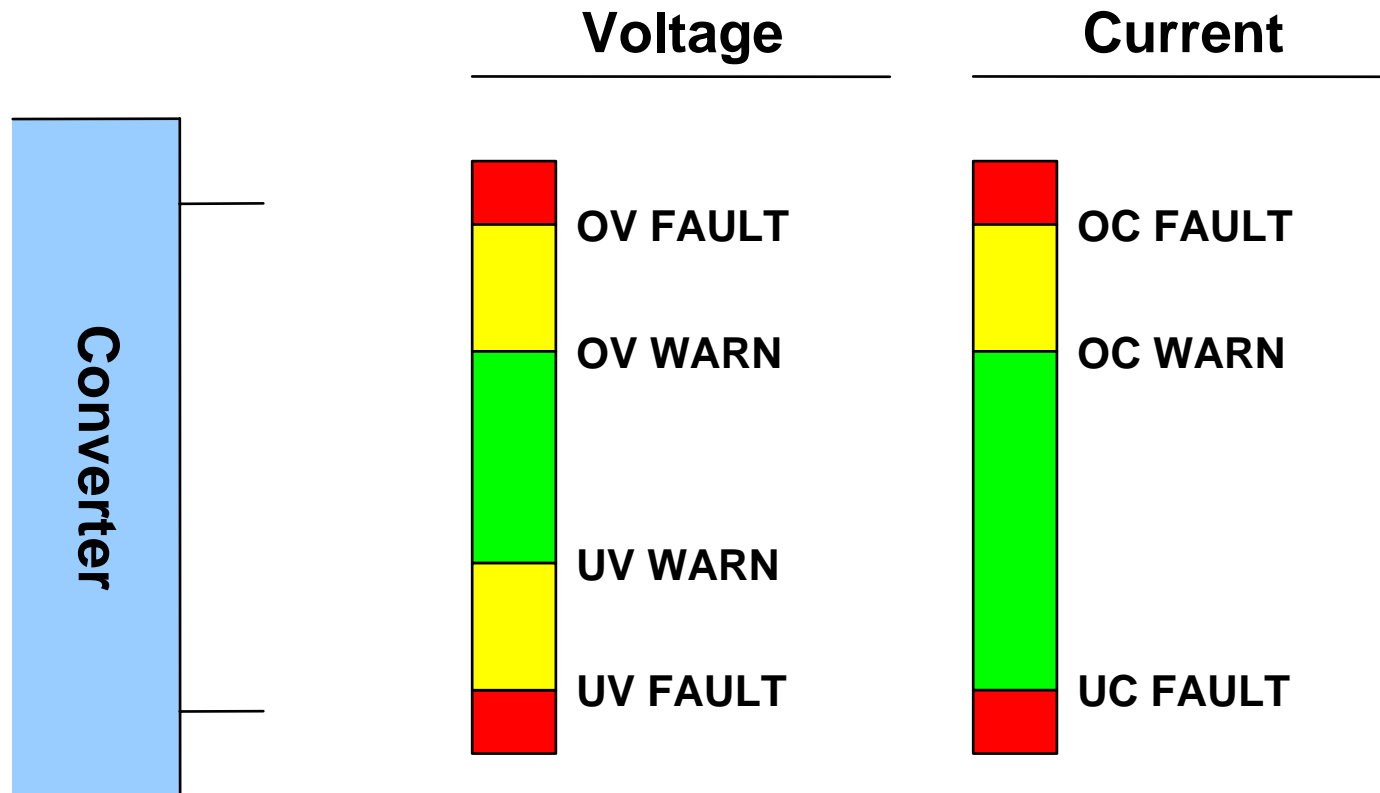
- Any warning or fault bits set in the status registers remain set, even if the fault or warning condition is removed or corrected, until:
 - The device receives a CLEAR_FAULTS command,
 - A RESET signal (if one exists) is asserted,
 - The output is commanded through the CONTROL pin, the OPERATION command, or the combined action of the CONTROL pin and OPERATION command, to turn off and then to turn back on
 - Bias power is removed from the PMBus device.
- If the warning or fault condition is present when the bit is cleared, the bit is immediately set again. The device shall respond as described in Section 10.2.1 or Section 10.2.2 as appropriate.

Fault Management: Input



Related Commands `VIN_ON`, `VIN_OFF`

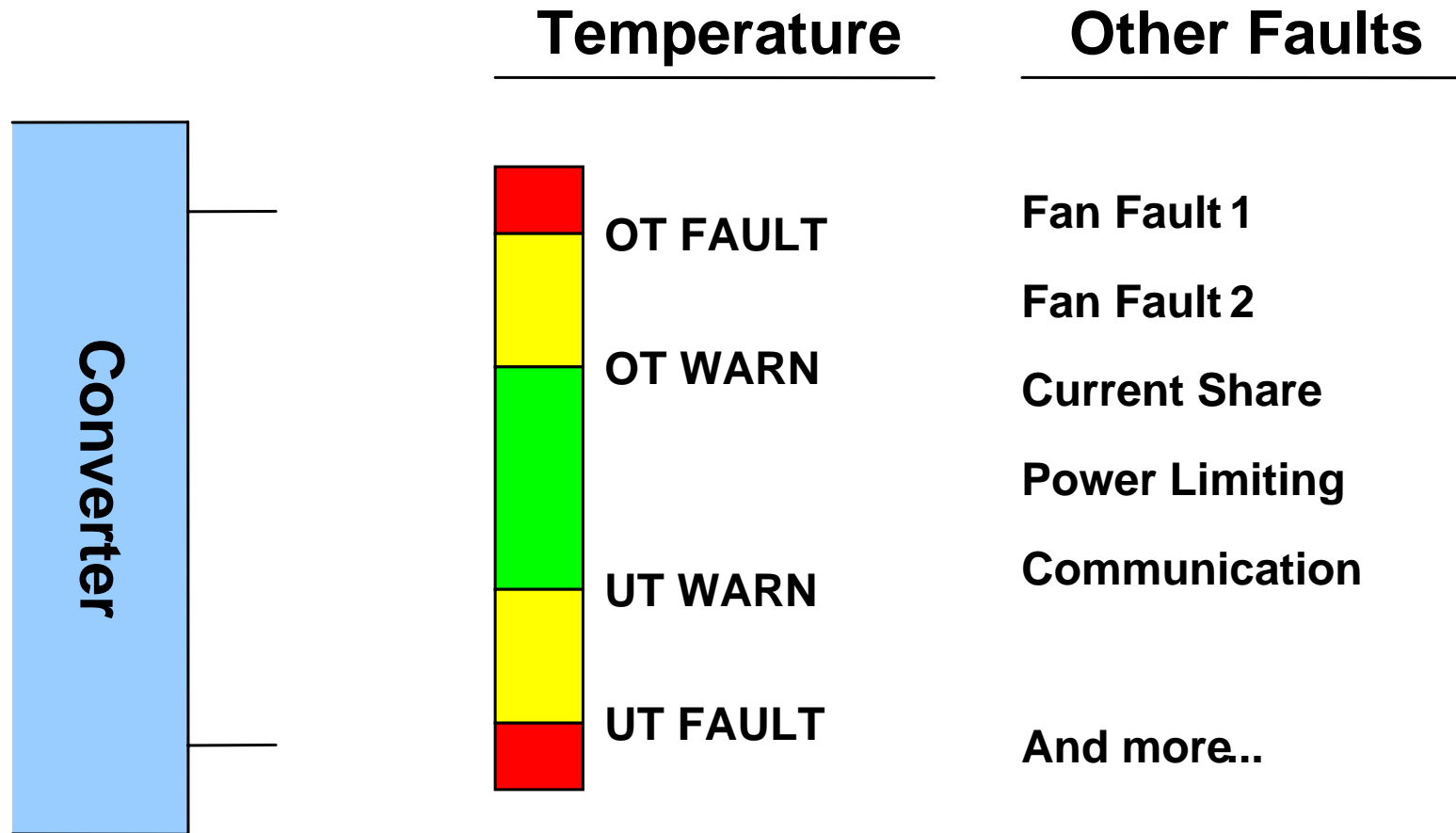
Fault Management: Output



Related Commands

POWER_GOOD_ON, POWER_GOOD_OFF

Other Fault Management



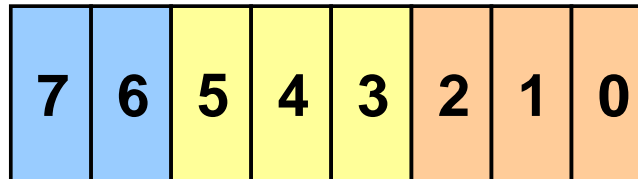
Voltage Or Temperature Fault Response Programming Byte

RESPONSE

00 - CONTINUE
01 - DELAYED OFF
10 - SHUTDOWN & RETRY
11 - INHIBIT

DELAY TIME

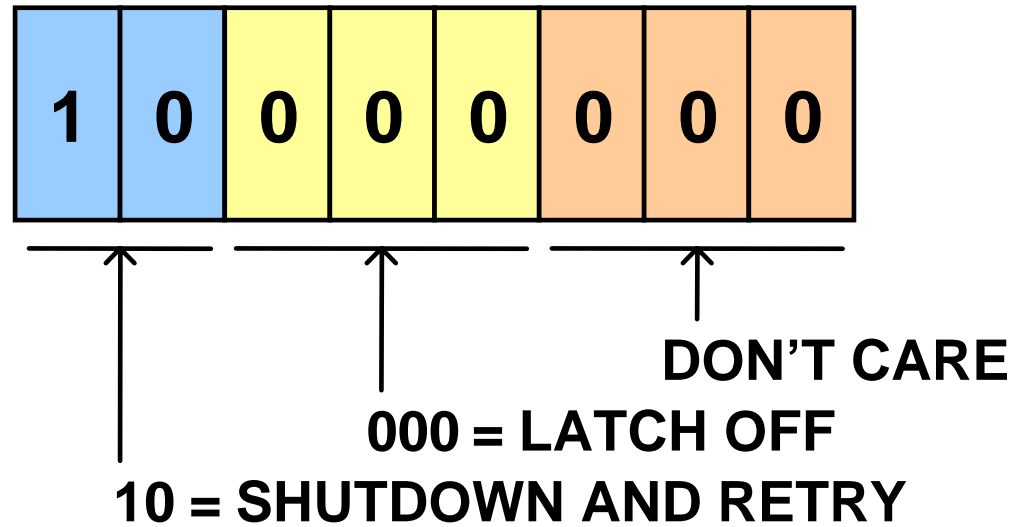
XXX - NUMBER OF DELAY
TIME UNITS



RETRY

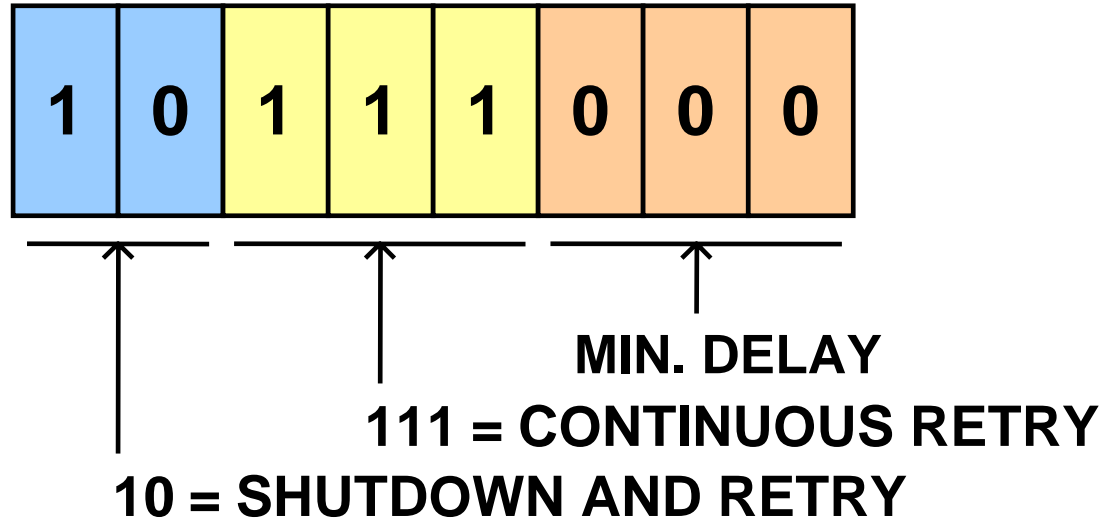
000 - LATCH OFF
001 - 110: RETRY COUNT
111 - CONTINUOUS

Fault Response Examples



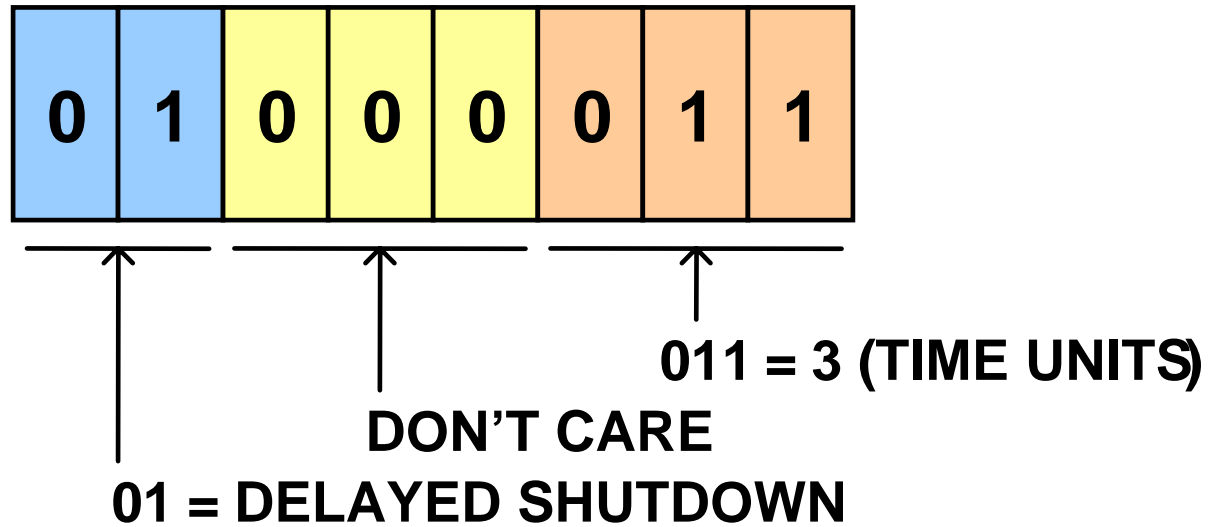
Shut Down And Latch Off

Fault Response Examples



Continuous Hiccup Mode

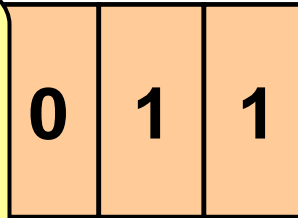
Fault Response Examples



**Keep operating for 3 time units.
If fault still exists at that time,
shut down and latch off**

Fault Response Examples

“Time units” are defined in each device’s product literature



011 = 3 (TIME UNITS)

01 = DELAYED SHUTDOWN

Keep operating for 3 time units.
If fault still exists at that time,
shut down and latch off

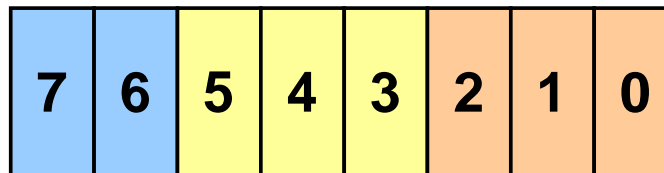
Current Fault Options

RESPONSE

00 - CONTINUE
01 - CONTINUE WITH
 LOW VOLTAGE SHUTDOWN
10 - DELAYED OFF
11 - SHUTDOWN& RETRY

DELAY TIME

XXX - NUMBER OF DELAY
TIME UNITS



RETRY

000 - LATCH OFF
001 - 110: RETRY COUNT
111 - CONTINUOUS

Parametric Information

- Input voltage (READ_VIN)
- Input current (READ_IIN)
- Output voltage (READ_VOUT)
- Output current (READ_IOUT)
- Hold up capacitor voltage (READ_VCAP)
- Temperature (READ_TEMPERATURE_1, _2, _3)
 - Up to 3 sensors
- Fan speed (READ_VFAN_1,_2)
 - Up To 2 Fans
- Duty cycle (READ_DUTY_CYCLE)
- Switching frequency (READ_FREQUENCY)

Data Integrity And Security

- Protecting against corrupted transmissions
 - Packet error checking can be used
- Unwanted or unintentional data changes
 - Write protect pin
 - WRITE_PROTECT command
- Expect device vendors to provide additional security features as demanded by specific applications

Manufacturer And User Data

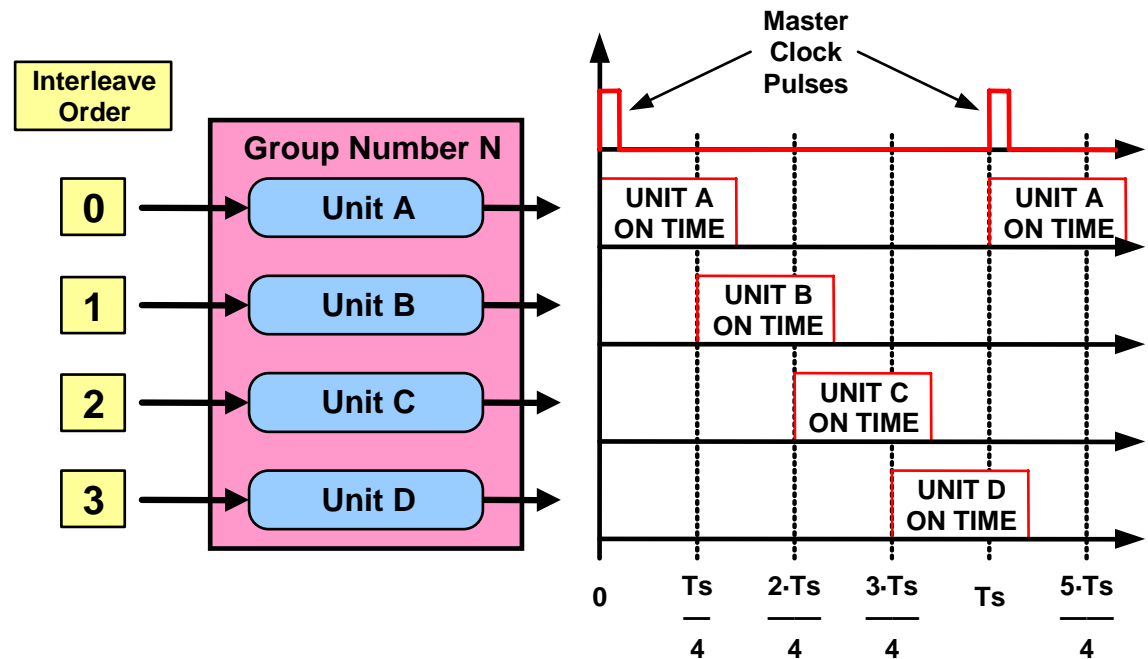
- Manufacturer's information
 - Inventory information (model number, etc.)
 - Ratings information (input voltage range, etc.)
- User data
 - 32 command codes for PMBus device makers to support user inventory and configuration data
 - Example: installation tracking information
- Manufacturer specific commands
 - 45 command codes reserved for PMBus device makers to implement manufacturer specific commands

Interleaving

■ INTERLEAVE Command sets

- Group number
- Number of units in the group
- Switching order within the group

Example Of INTERLEAVE Command Operation

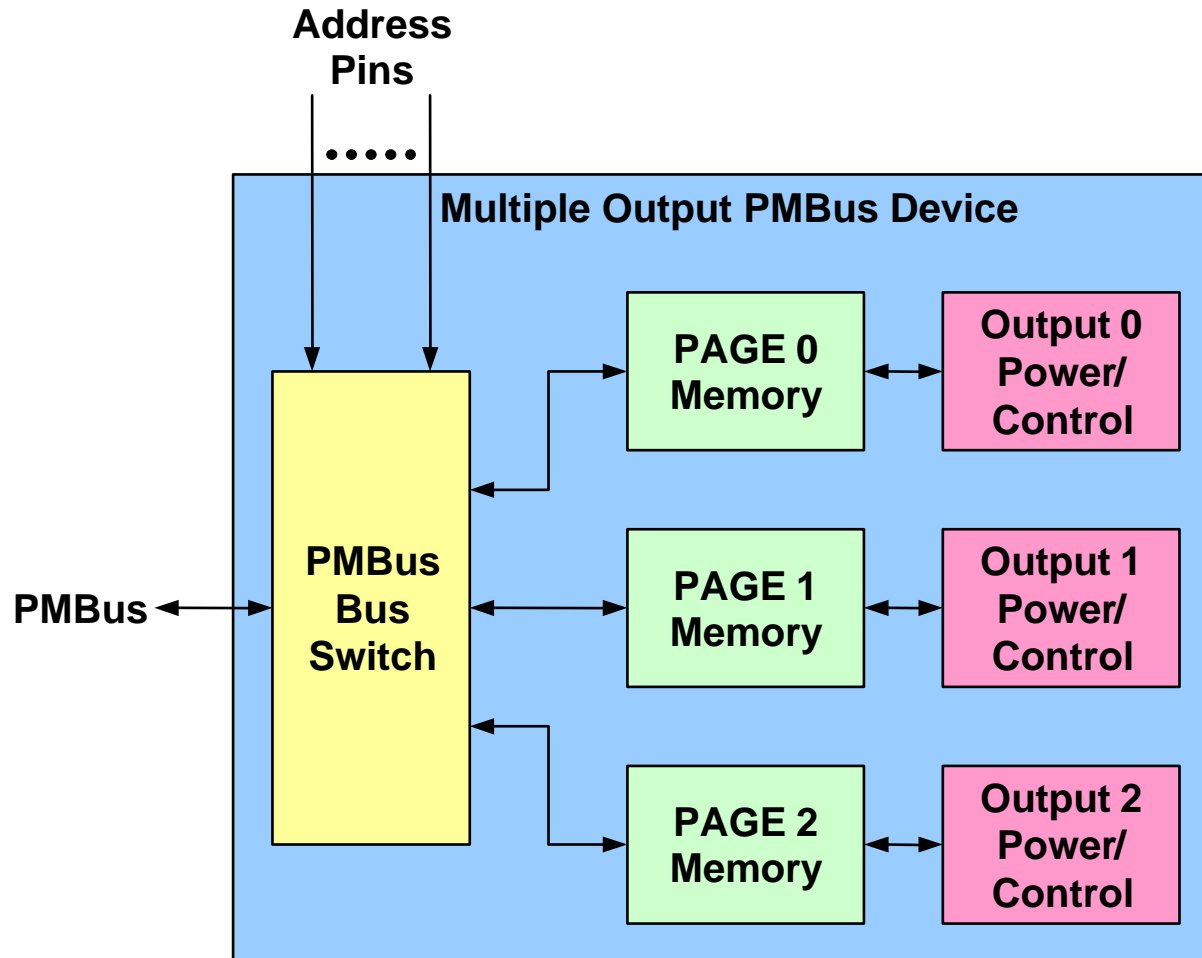


$$T_{\text{delay}}(\text{Unit } X) = \frac{\text{Interleave Order Of Unit } X}{\text{Number In Group}} \cdot T_s$$

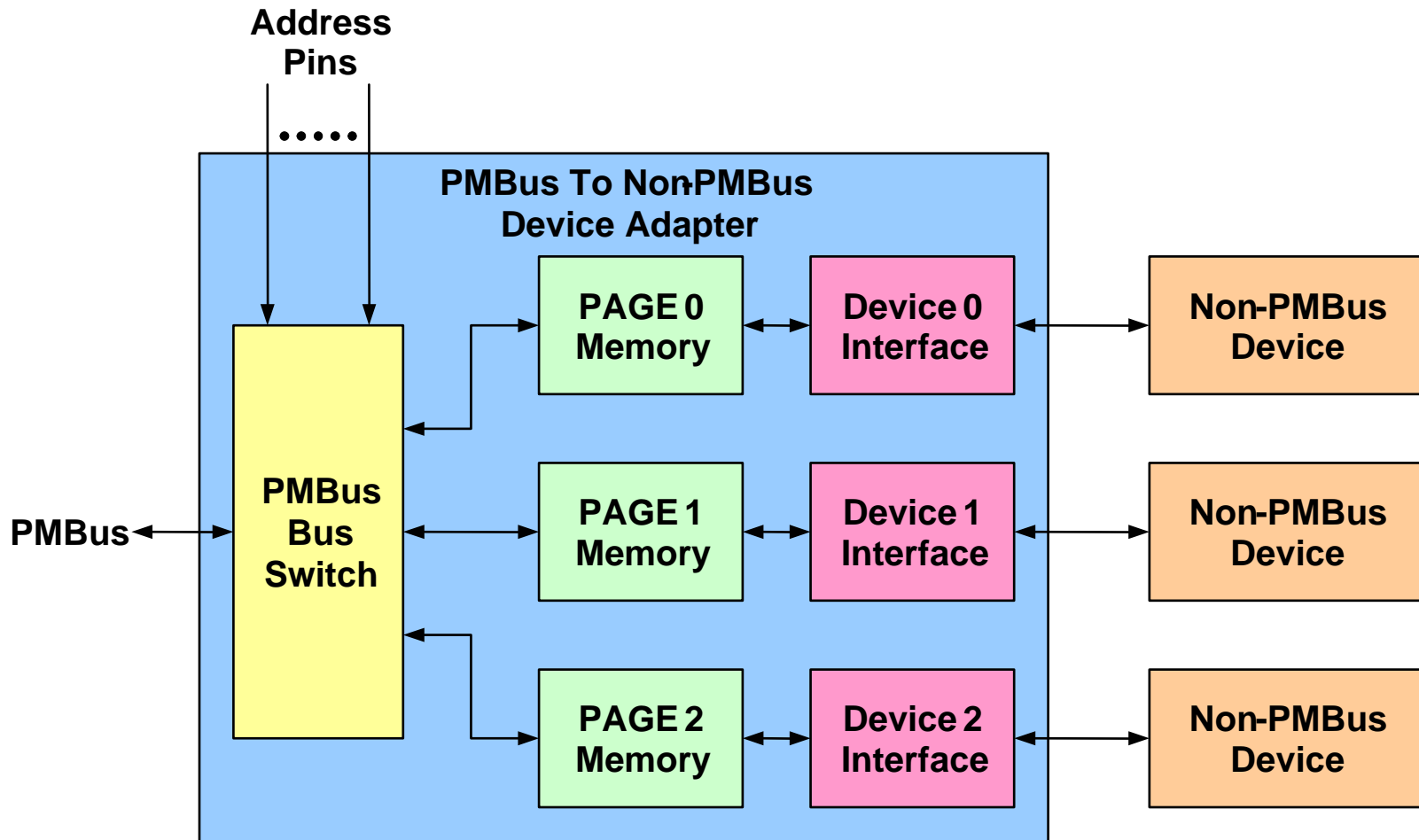
Multiple Output Units And Paging

- Paging allows one physical address to be used to control multiple outputs
 - One address per physical unit
 - One page per output
 - Pages contain all the settings of each output
- Paging process
 - Set page for output of interest
 - Send commands
 - Configure, control, read status

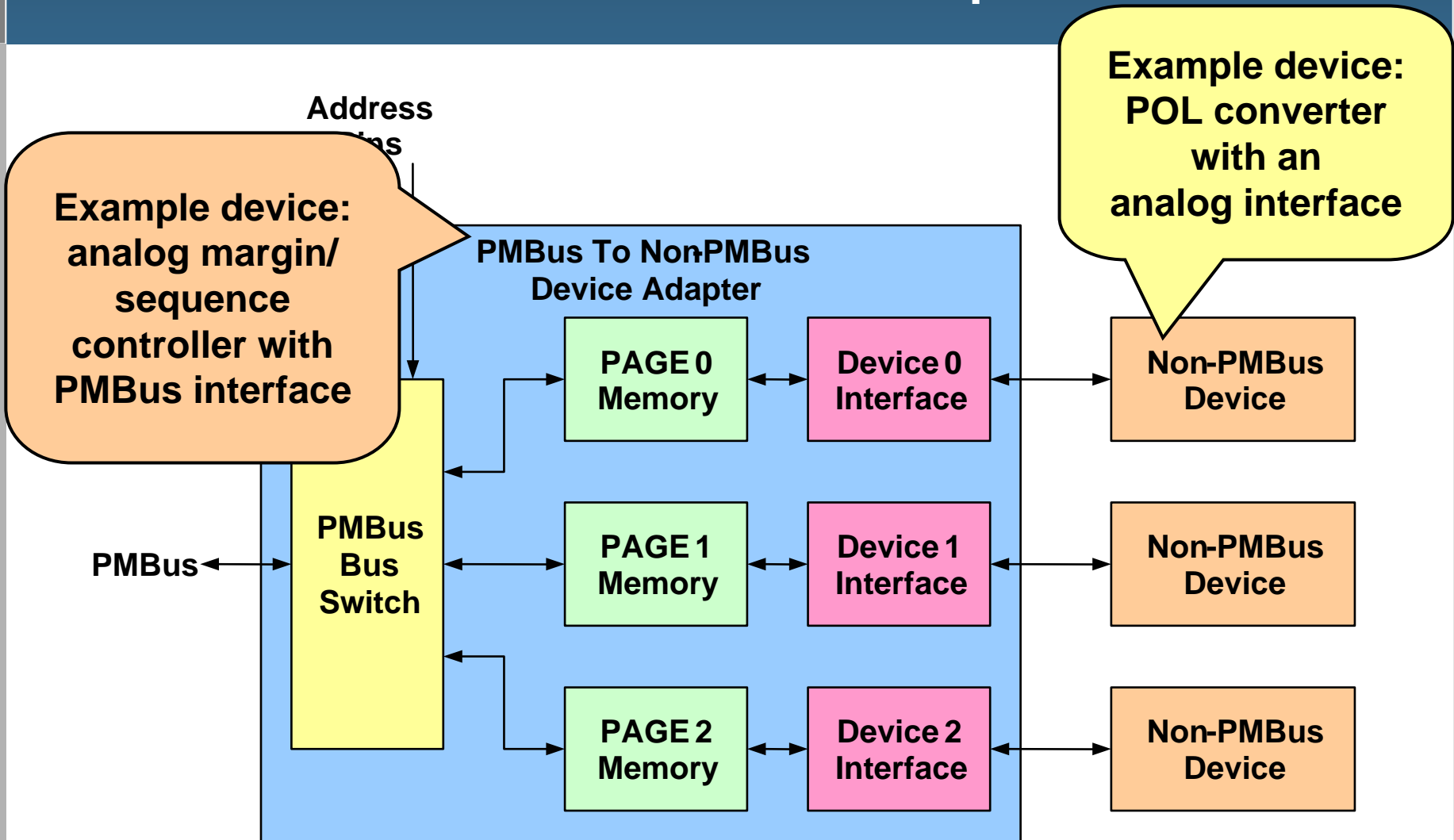
Paging: Multiple Output Units



Paging: Non-PMBus Device Adapter



Paging: Non-PMBus Device Adapter



Many Other Configuration Commands

- Maximum output voltage
- Maximum output power
- Voltage scale for external output monitor path
- Maximum duty cycle
- Switching frequency
- Turn on/off levels for input voltage
- Current scale for current sense resistance
- Current measurement calibration

For More Information

www.PMBus.org

info@PMBus.org

**Thank You
For Your Time
And Attention!**