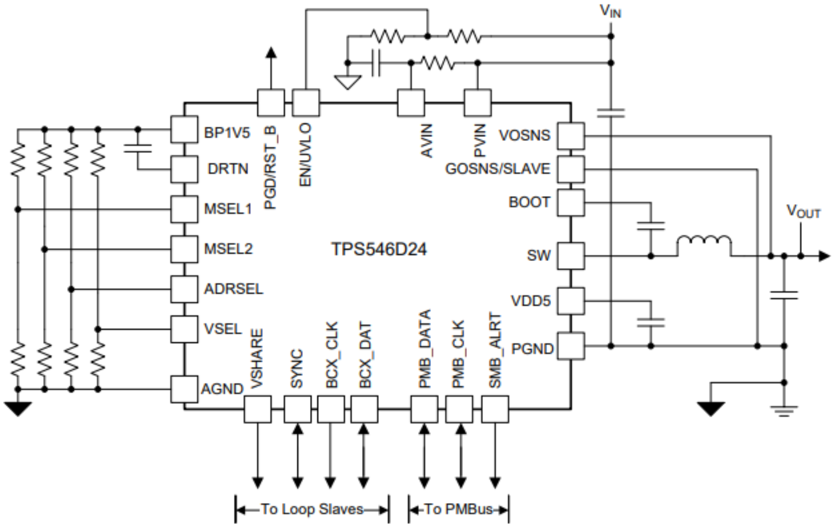
Technical Article

TPS546D24A Programming considerations

Jason Ren

This document was translated from a simplified Chinese source. (ZHCTA06)

TI's TPS546D24A is a highly integrated Buck Converter with a high control frequency that supports currents up to 40A. It supports 2/3/4 parallel outputs totaling 160A. The internal 5V LDO can be driven by an external 5V supply via VDD5 pin to improve efficiency and reduce the heat dissipation of the converter. Meanwhile, the device supports PMBus interface configuration. 1MHz PMBus interface rate allows TPS546D24A to be configured more conveniently and flexibly for the device parameters such as output voltage/current, and to monitor device junction temperature. It also provides Back-channel communication between stack devices, enabling stack devices to use the same device address, thereby simplifying software system complexity.



Users may have some confusion regarding programming methods and considerations when using the device. Therefore, the following section briefly outlines programming considerations and potential issues users may encounter during application.

Description of device PMBus configuration Stretching:

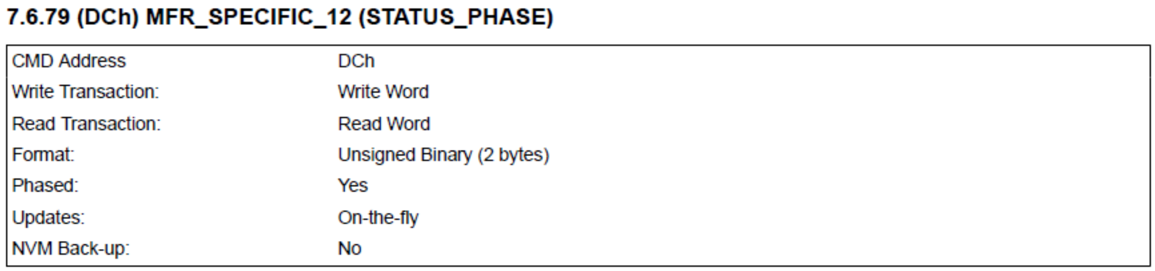
For specific protocol of the device PMBus, you may refer to part 1 on https://pmbus.org/. TPS546D24A supports 100kHz, 400kHz, 1Mhz communication rates. It is important to note that on certain data transmission bit, TPS546D24A reserves clock stretching to guarantee proper processing logic. Here is a brief overview of this stretching feature:

- Address BIT does not stretch
- Stretch occurs between command byte bit0 and ACK
- Stretch occurs after address bit0 of the read command
- Stretch occurs between bit0 and ACK of the last byte of the data

Programming Mode and considerations for programming NVM:

Enter Programming Mode for device software configuration. Here are a few points that must be noted during the programming phase:

1. When programming, ensure that AVIN is above 3.3V and VDD5 is above 3.5V, otherwise the internal charge pump may not function properly and generate enough voltage to support NVM programming.
2. During programming, after executing STORE_USER_ALL, wait at least 200ms to allow the internal NVM to complete the programming. As a debug purpose, if NVM still does not refresh to the expected configuration value at the next power-up cycle after STORE_USER_ALL occurs. You can try executing it again after STORE_USER_ALL to determine if the command is issued/executed successfully.
3. The configuration of the device is stored in RAM before NVM programming. Therefore, after a power cycle, this configuration is lost, and during the next power cycle, the device reverts to the configuration from the previous NVM programming cycle. Consequently, when reading registers after the device powers up, the values returned to the user are those read from RAM, specifically, the values programmed into the device by NVM during the previous power-up cycle.
4. During NVM programming, the device will not respond to new commands, such as STORE_USER_ALL, or to configuration attempts on other registers in RAM. Therefore, in user software, NVM programming and other commands must be mutually exclusive to ensure no conflicts occur.
5. During the debugging phase, in addition to performing write/read checks on the written registers to verify correctness, you can also use the following commands to check for programming communication failures or programming failures:

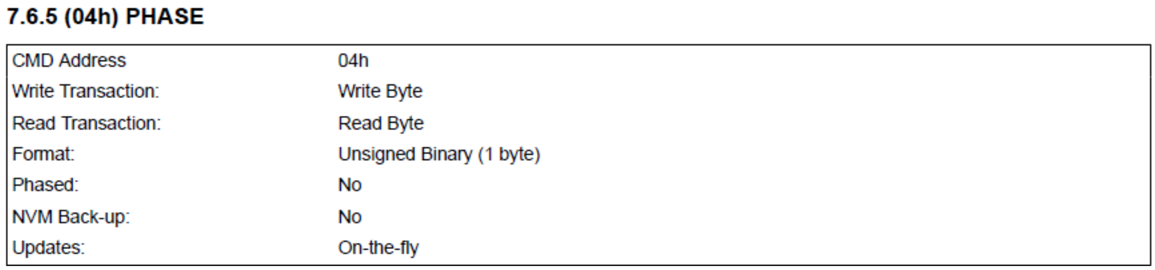


7.6.79 (DCh) MFR_SPECIFIC_12 (STATUS_PHASE)

CMD Address	DCh
Write Transaction:	Write Word
Read Transaction:	Read Word
Format:	Unsigned Binary (2 bytes)
Phased:	Yes
Updates:	On-the-fly
NVM Back-up:	No

6. In a multi-stack cascade scenario, the user needs to read the status read/write for a particular phase, select the phase number, and then read/write the corresponding phase device.

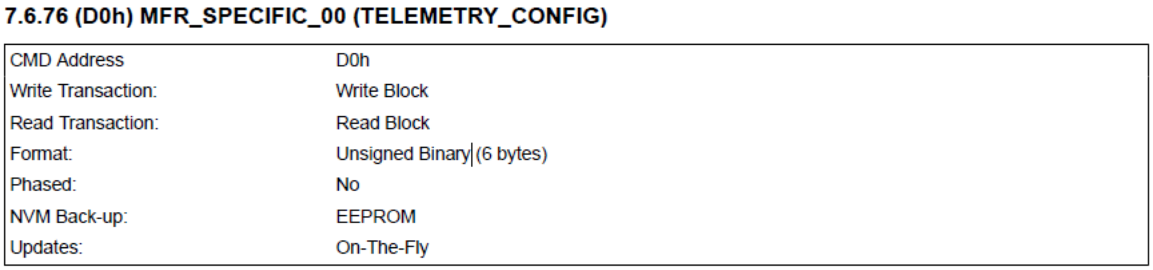
   It is important to note that after the operation of a separate phase device, it is recommended to write the phase register back to 0xff.



7.6.5 (04h) PHASE

CMD Address	04h
Write Transaction:	Write Byte
Read Transaction:	Read Byte
Format:	Unsigned Binary (1 byte)
Phased:	No
NVM Back-up:	No
Updates:	On-the-fly

7. Telemetry data is continuously reported as a background process to the master (via main PMBus) and the slave (via BCX channel), covering conditions such as overtemperature, undervoltage, and overcurrent. As a result, the user may also observe data continuously reported on PMBus or BCX channels while the user is not operating the device. This reporting process is mutually exclusive to the user's actions on the device, as determined by the device's internal logic, to prevent conflicts. Meanwhile, the priority of data reported by Telemetry is configurable in the following registers.



7.6.76 (D0h) MFR_SPECIFIC_00 (TELEMETRY_CONFIG)

CMD Address	D0h
Write Transaction:	Write Block
Read Transaction:	Read Block
Format:	Unsigned Binary (6 bytes)
Phased:	No
NVM Back-up:	EEPROM
Updates:	On-The-Fly

8. In a multi-stack cascade system, BCX communication helps users simplify the complexity of host and multiple slaves, so that there is only a single PMBus host and slave for the user system communication level. Write commands issued by BCX to the slave are dispatched in the order in which the master receives PMBus commands. However, the slave readback action is real-time. Once the host requests to read back the status/configuration of a particular slave via BCX, and BCX is idle, the slave will report the data promptly through BCX. It is important to note here that when writing and reading back to devices on BCX, we recommend adding a 4ms delay between writing and reading to ensure that the reading is correct.

This article focuses on describing the programming methods for TPS546D24A and highlights key considerations for software implementation. For more information, please refer to the device manual on TI's official website. For more detailed device configuration methods, you may use the software tool of the device, FUSION DIGITAL POWER DESIGNER, to learn which registers are configured/read for device-specific features in GUI interface.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025