

Vader Digital DFT Guide

Contents

[1. Overview 4](#_Toc144120375)

[2. References 5](#_Toc144120376)

[2.1. I](#_Toc144120377)[2](#_Toc144120377)[C specification 5](#_Toc144120377)

[2.2. Vader Register Map 5](#_Toc144120378)

[2.3. Vader Digital Specification 5](#_Toc144120379)

[2.4. eMemory OTP Documents 5](#_Toc144120380)

[3. Terminology 6](#_Toc144120381)

[4. I](#_Toc144120382)[2](#_Toc144120382)[C 7](#_Toc144120382)

[5. Scan ATPG 9](#_Toc144120383)

[6. Shadow Registers and OTP Data Format 12](#_Toc144120384)

[6.1. Page Loading Algorithm 14](#_Toc144120385)

[7. OTP Access 15](#_Toc144120386)

[7.1. Reload Shadow Registers 16](#_Toc144120387)

[7.1.1. Reload Test Options 16](#_Toc144120388)

[7.2. Read OTP Memory 16](#_Toc144120389)

[7.2.1. Read OTP Test Options 17](#_Toc144120390)

[7.3. Write OTP Memory 17](#_Toc144120391)

[7.3.1. Write OTP Test Options 18](#_Toc144120392)

[7.4. Committing Shadow Registers to OTP Memory 18](#_Toc144120393)

[8. Test Modes 20](#_Toc144120394)

[8.1. Analog Test Modes 20](#_Toc144120395)

[8.2. Overriding Digital-to-Analog Output Signals 21](#_Toc144120396)

[8.3. Forcing State Machines 22](#_Toc144120397)

[8.4. Diagnostic Digital Mux 23](#_Toc144120398)

[9. C Code to Calculate Error Correction Code 26](#_Toc144120399)

[10. Revision History 28](#_Toc144120400)

[Figure 1: Example Vader Application Diagram 4](#_Toc144126972)

[Figure 2: I2C Register Write and Read 7](#_Toc144126973)

[Figure 3: I2C Burst Write and Read 7](#_Toc144126974)

[Figure 4: Scan Pattern Simulation Waveform 9](#_Toc144126975)

[Figure 5: Scan Shift-Capture Diagram (image copied from PE24103 Digital DFT document) 10](#_Toc144126976)

[Figure 6: Scan Reset Diagram (image copied from PE24103 Digital DFT document) 10](#_Toc144126977)

[Figure 7: ATPG Fault Coverage 11](#_Toc144126978)

[Figure 8: Shadow and Volatile Registers 12](#_Toc144126979)

[Figure 9: Vader Shadow Registers I2C View 13](#_Toc144126980)

[Figure 10: Vader Shadow Registers OTP Memory 14](#_Toc144126981)

[Figure 11: Shadow Image Registers 15](#_Toc144126982)

[Figure 12: NVM ACCESS registers 16](#_Toc144126983)

[Figure 13: DFT Registers 21](#_Toc144126984)

[Table 1: Digital DFT Pins 4](#_Toc137132976)

[Table 2: Scan Pins 8](#_Toc137132977)

[Table 3: ATPG-Test Pattern Statistics 10](#_Toc137132978)

[Table 4: OTP Page Addressing 12](#_Toc137132979)

[Table 5: OTP Read Modes 14](#_Toc137132980)

[Table 6: NVM SIZE Field 14](#_Toc137132981)

[Table 7: Test Mode Signals/Registers 19](#_Toc137132982)

[Table 8: Output Signal Override Field 20](#_Toc137132983)

[Table 9: Output Overrides 20](#_Toc137132984)

[Table 10: Main State Codes 21](#_Toc137132985)

[Table 11: Power Control State Codes 21](#_Toc137132986)

[Table 12: Diagnostic Data 22](#_Toc137132987)

# Overview

Diagram, schematic

Description automatically generated

Figure : Example Vader Application Diagram

This document contains information about the digital Design-For-Test (DFT) features in the Vader design.

Table : Digital DFT Pins

|  |  |  |  |
| --- | --- | --- | --- |
| Pin | Mission | Scan | Diagnostic Data |
| **Boost\_Mode** | boost\_mode (sampled at startup) | scan clock | diagnostic data 1 |
| **Ratio0** | ratio (sampled at startup) | scan reset (active low) |  |
| **Ratio1** | ratio sampled at startup | scan enable |  |
| **RCL** | CP current limit setting resistor | scan data in |  |
| **RCLK** | CP switching frequency setting resistor (measured at startup) |  | diagnostic data 0 |
| **SCL** | I2C clock input |  |  |
| **SDA** | I2C open-drain, bidirectional | scan data out |  |

# References

The official version of each digital document is kept under version control, co-located with the actual digital design. The documents in the SOS project area should be considered as be the best, most up-to-date version. The Vader digital project can be found in SOS at

Server: DIG  
 Project: VADER\_DIGITAL

## I2C specification

/projects/VADER\_DIGITAL/work/<USER>/windows\_files/UM10204.pdf

## Vader Register Map

/projects/VADER\_DIGITAL/work/<USER>/windows\_files/vader\_register\_map.xlsm

Several register map snapshots are included in this document for readability. Please refer directly to the vader\_register\_map.xlsm spreadsheet itself to get the best version.

## Vader Digital Specification

/projects/VADER\_DIGITAL/work/<USER>/windows\_files/vader\_digital\_spec.docx

## eMemory OTP Documents

/tools/IP/OTP/eMemory/EO0128X8KA130BC02\_v1.1/Datasheet/EO0128X8KA130BC02\_v1.1.pdf

/tools/IP/OTP/eMemory/EO0128X8KA130BC02\_v1.1/Test\ Methodology/EO0128X8KA130BC02\_OTP\_Fuse\_Test\_Methodology\_v1.1.pdf

# Terminology

|  |  |
| --- | --- |
| ECC | Error Correction Code. |
| I2C Bus | Inter-IC Control bus. An industry standard two-wire serial bus created by Philips Semiconductors (aka NXP). |
| I2C Slave Address | Address of a slave device on the I2C bus. |
| NVM | Generic term for Non-Volatile Memory. Some examples are EEPROM, EPROM, Flash, OTP, FTP, MTP. |
| OTP | One Time Programmable. A type of non-volatile memory. |
| OTP Address | Physical address of a row in OTP memory. |
| OTP Block | An 8-byte span of OTP memory that includes 57 data bits and 7 ECC bits. |
| OTP Page | A group of OTP blocks that is sized to match the collective size of all the shadow registers. |
| Register Address | Address of an I2C accessible register. Each I2C accessible register has a unique address. |
| RO | Read Only |
| RW | Read Write |
| SCL | I2C Clock |
| SDA | I2C Data |
| Shadow Register | I2C accessible register that can be configured from OTP memory. After the preferred settings are captured in the OTP memory the shadow registers will be automatically reconfigured each time the chip powers up or the EN pin is toggled. |
| STIL file | ATPG test vectors in “IEEE P1450.1 Standard Test Interface Language” format. Often pronounced as “style”. |
| UV | Ultraviolet light. |
| Volatile Register | I2C accessible register that is **not** configured from OTP memory. The register data is lost whenever chip power is cycled or the EN pin is toggled. |
| W1C | Write 1 to Clear. |
| WGL file | ATPG test vectors in “Waveform Generation Language” format. Often pronounced as “wiggle”. |

# I2C

An I2C interface is provided to make test features available on ATE and in the lab. It is not intended to be fully compliant to the I2C standard. It supports standard (100KHz), fastmode (400KHz) and fastmode+ (1MHz) speeds. Vader has dedicated pins for the I2C **SCL** and **SDA** signals. The default 7-bit I2C Slave Address is 30h, but it will this address can be configured from a validly programmed OTP. For detailed information on the I2C interface standard, please refer to the I2C specification (see page 5 for location).

A simple register access protocol is implemented with 8-bit register address and 8-bit data transfers. Figure 2 shows write and read transactions for a single 8-bit register.

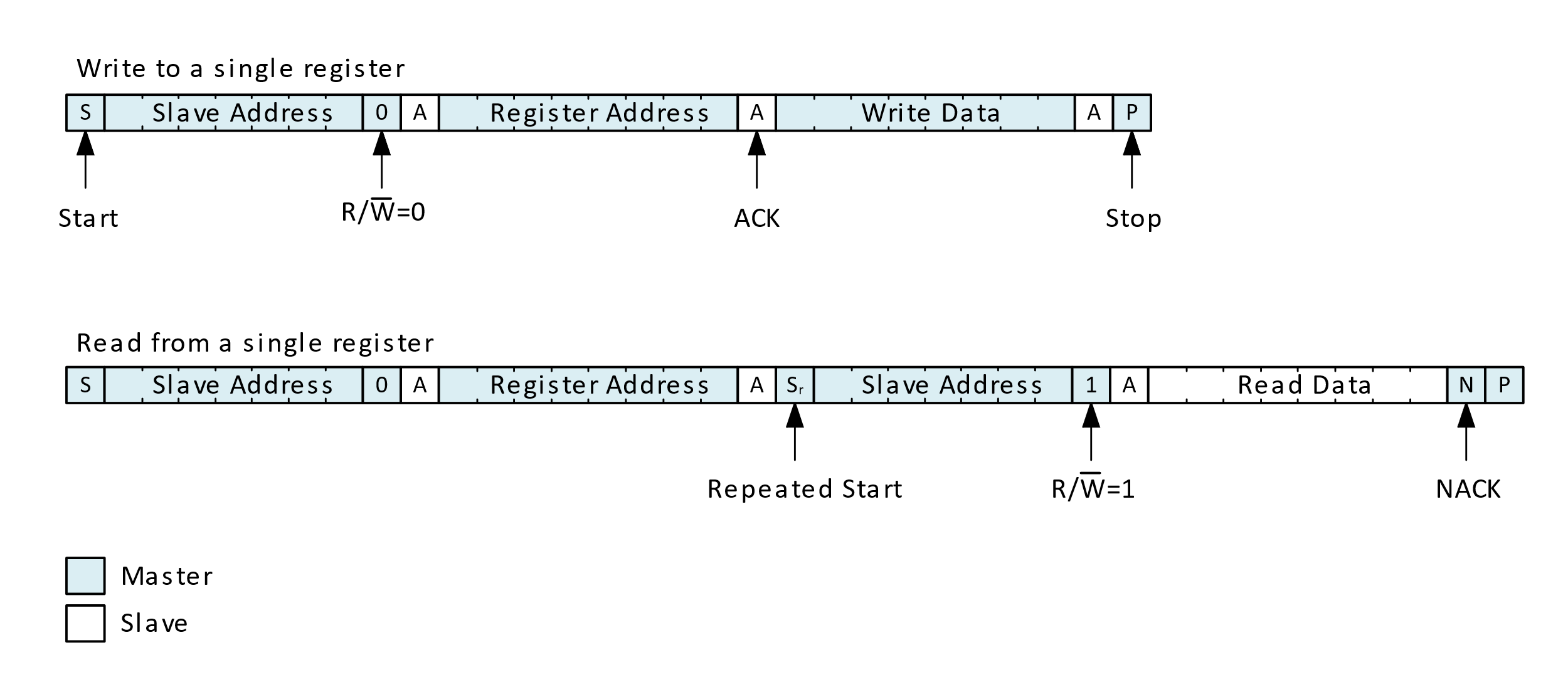


Figure : I2C Register Write and Read

The implemented protocol is capable of handling burst reads and writes. By default, burst data will sequentially read or write contiguous registers, where the register address is automatically incremented. If needed, the automatic address increment behavior can be disabled to enable repeated access of the same addressed register. Figure 3 shows write and read transactions for multiple registers.

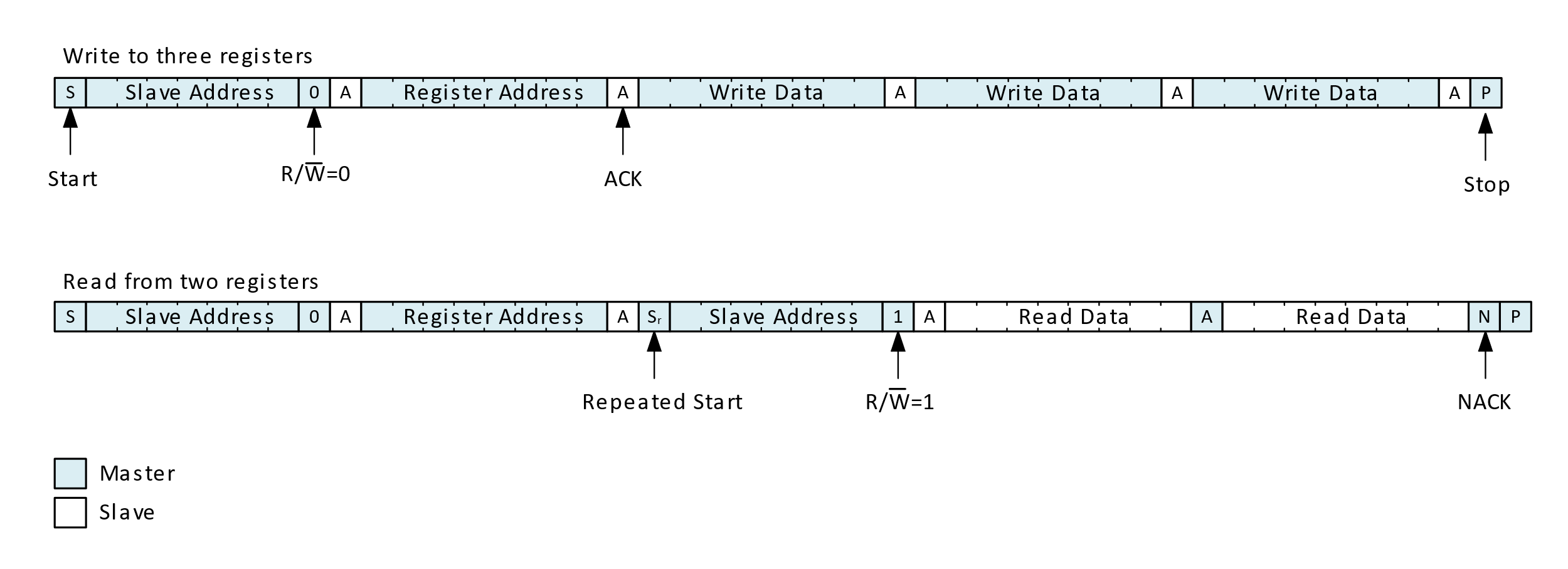


Figure : I2C Burst Write and Read

When bringing up I2C for the first time it may be useful to read the ID registers; which have known, fixed values.

1. Reading the DIE\_REV register (address = FDh) should return 00h.
2. Reading the CHIP\_ID register (address = Feh) should return 16h.
3. Reading the PSEMI\_ID register (address = FFh) should return 95h.

The NVM\_DATA\* registers (address = E8h through Efh) can be treated as scratch registers for writing and reading back.

# Scan ATPG

The Vader digital logic can be put into scan-mode by writing 80h to the TEST\_DFT register (address C1h). After entering scan-mode the digital cannot be used in mission mode again until the **EN** pin is toggled.

Table : Scan Pins

|  |  |
| --- | --- |
| Digital Scan Signal | External Pin |
| a\_scan\_clk | **Boost\_Mode** |
| a\_scan\_en | **Ratio1** |
| a\_scan\_reset\_n | **Ratio0** |
| a\_scan\_in | **RCL** |
| d\_scan\_out | **SDA** |

The scan vectors can be found in the VADER\_DIGITAL Cliosoft (SOS) repository in the ./modus/testresults directory. The **SDA** pin is open-drain and requires a pull-up. The **SDA** I/O also inverts the scan out signal. The scan vectors are generated to take this into account. There are Verilog, STIL and WGL formats available. The scan vector simulation environment is located in the ./modus/sim directory.

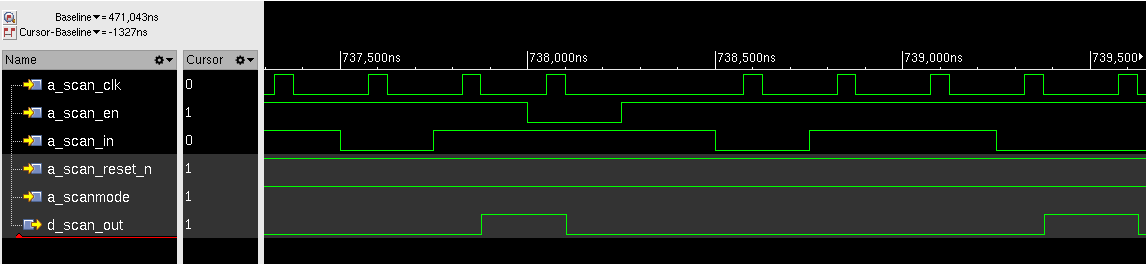


Figure : Scan Pattern Simulation Waveform

The scan pattern timing was copied from the BB8 and R2D2 projects. The scan patterns are generated for 4MHz timing, but will most likely need to be scaled down to 1MHz or slower.

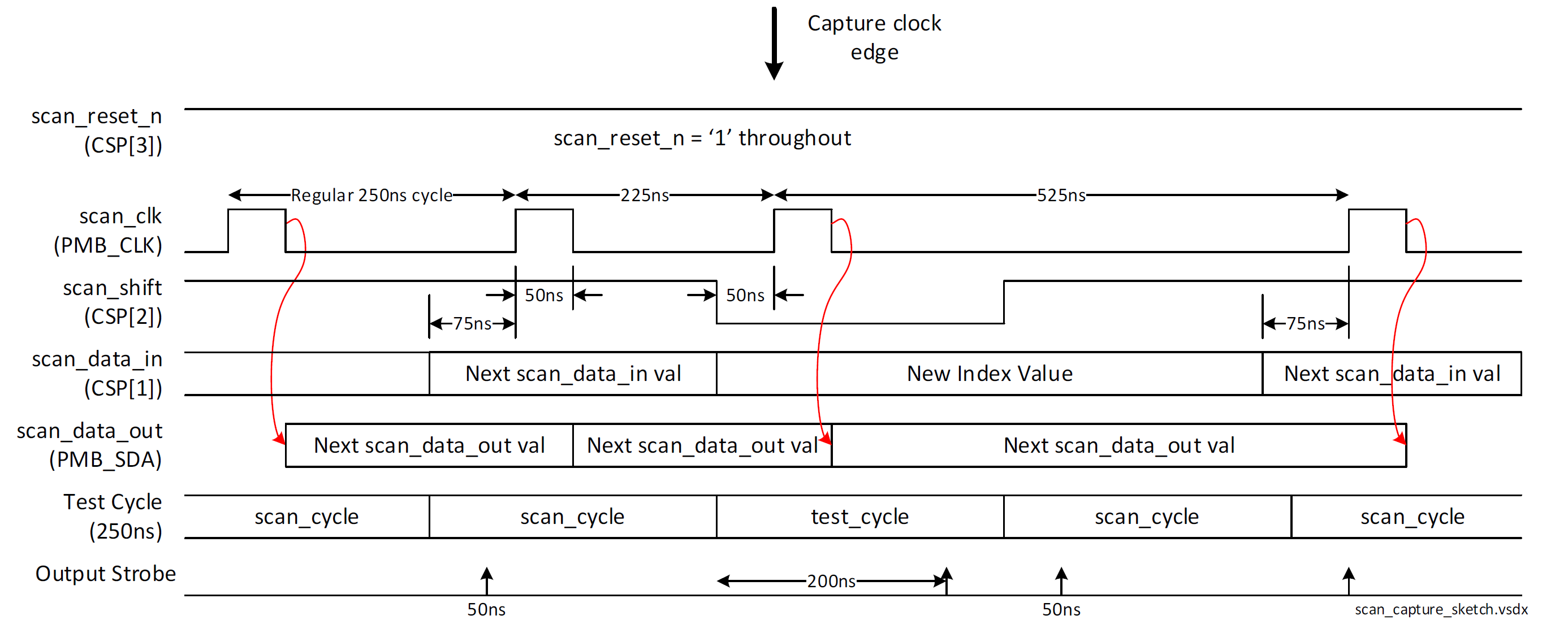


Figure : Scan Shift-Capture Diagram (image copied from PE24103 Digital DFT document)

Diagram

Description automatically generated with medium confidence

Figure : Scan Reset Diagram (image copied from PE24103 Digital DFT document)

Figure : ATPG Fault Coverage

Table : ATPG-Test Pattern Statistics

|  |  |
| --- | --- |
| Tests | 351 |
| Chain Length | 976 |
| Clock Cycles | 347,191 |

# Shadow Registers and OTP Data Format

Vader’s shadow registers are I2C accessible registers that are loaded from OTP memory when the device powers up. The shadow registers can be written and read like a normal, volatile registers. The OTP memory is large enough to support three collections of shadow register data (OTP pages). See Figure 8 for an overview of the OTP memory and I2C accessible registers.

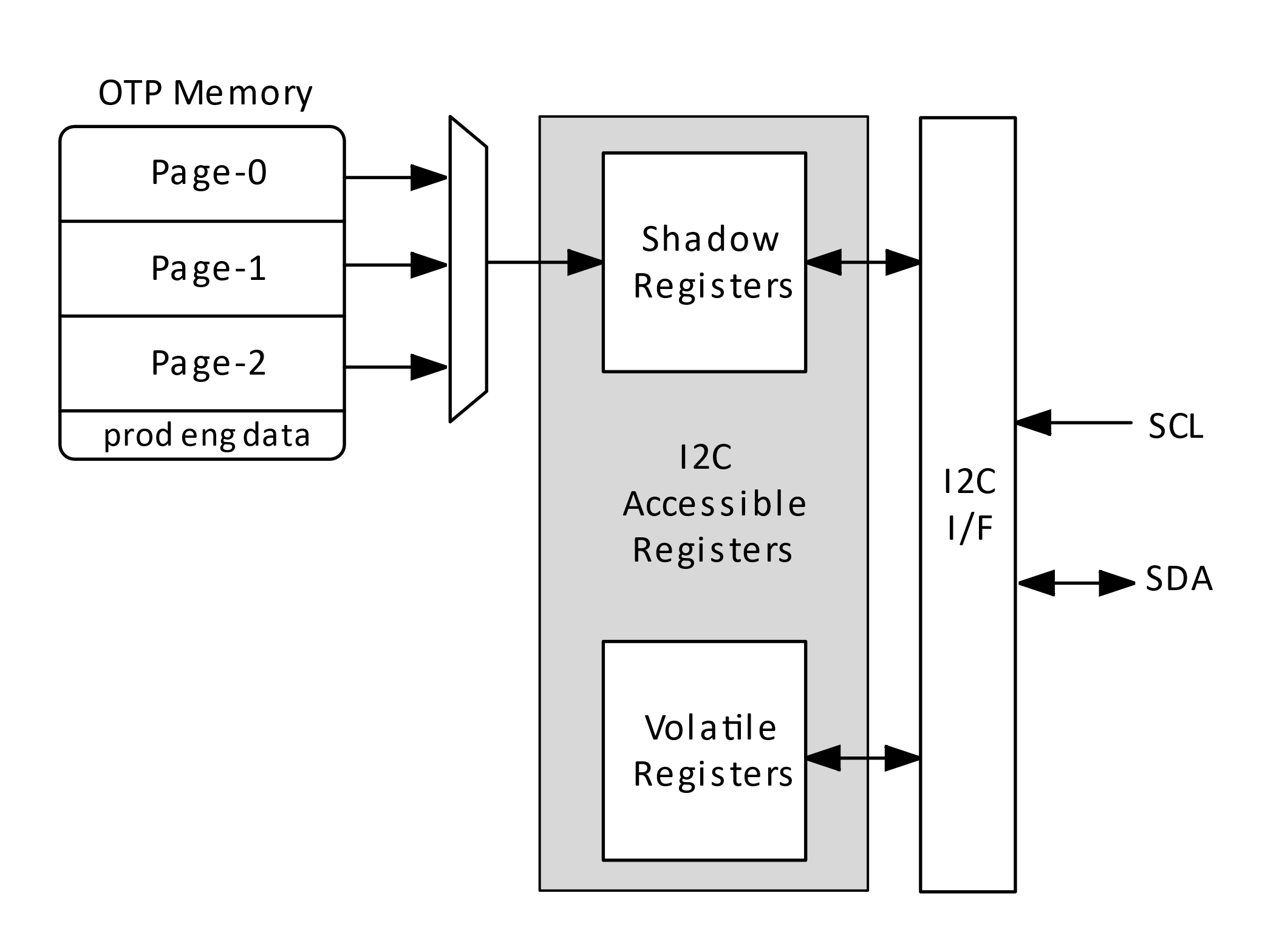


Figure 8: Shadow and Volatile Registers

The I2C view of the shadow registers is different from the data arrangement in OTP memory. First, the raw OTP memory is also organized into 64-bit blocks with 7-bit ECC checksums (OTP blocks). When the shadow registers are loaded these checksums are used for error correction, but they are not loaded into registers. Secondly, the shadow registers may contain empty bits, but the OTP memory is packed to save memory. Refer to Figure 8 and Figure 9 to see how the shadow registers are implemented in the I2C view and in the condensed OTP memory.

Please note that unprogrammed OTP bits are 0.[[1]](#footnote-1) Once an OTP bit is programmed to 1 it cannot be changed back to 0, unless the OTP memory is erased with ultraviolet (UV) light exposure.[[2]](#footnote-2)

A screenshot of a computer screen

Description automatically generated with low confidence

Figure : Vader Shadow Registers I2C View[[3]](#footnote-3)

A screenshot of a computer

Description automatically generated with medium confidence

Figure : Vader Shadow Registers OTP Memory[[4]](#footnote-4)

The entire collection of shadow registers consumes only 40 bytes of the 128-byte OTP memory. Refer to Table 4 to see how the OTP memory is divided into three 40-byte pages, plus space for product engineering data. Each 40-byte OTP Page is further subdivided into five 8-byte OTP Blocks.

Table : OTP Page Addressing

|  |  |  |  |
| --- | --- | --- | --- |
| Page | Block | NVM\_ADDR | IMAGE\_ADDR |
| 0 | 0 | 00h | 00h |
| 1 | 08h | 08h |
| 2 | 10h | 10h |
| 3 | 18h | 18h |
| 4 | 20h | 20h |
| 1 | 0 | 28h | 00h |
| 1 | 30h | 08h |
| 2 | 38h | 10h |
| 3 | 40h | 18h |
| 4 | 48h | 20h |
| 2 | 0 | 50h | 00h |
| 1 | 58h | 08h |
| 2 | 60h | 10h |
| 3 | 68h | 18h |
| 4 | 70h | 20h |
| Reserved for Product Engineering Data | 0 | 78h | N/A |

A condensed view of the shadow registers is available via the IMAGE\_ADDR and eight IMAGE\_DATA\* registers. These registers present a single 8-byte block of shadow register data in the same format as the OTP memory. These may help to simplify OTP programming. Refer to Figure 10 to see the IMAGE\_ADDR and IMAGE\_DATA\* registers.

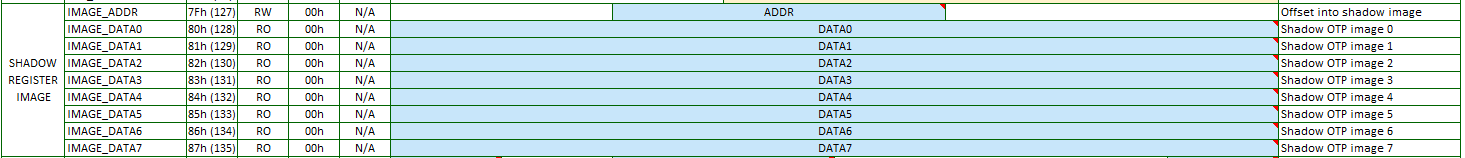


Figure : Shadow Image Registers[[5]](#footnote-5)

## Page Search Algorithm

On each power-up the shadow registers are updated from OTP memory. The page search algorithm scans the OTP memory looking for a valid OTP page. Vader starts with page-0, followed by page-1 and then page-2. Vader will stop searching after a valid page is found. Vader will also stop searching if none of the three pages are valid. Two things are needed for an OTP Page to be valid:

1. the first four bits of the page are set to 1010b.
2. no uncorrectable errors are detected across the whole page.

# OTP Access

On startup the shadow registers are automatically loaded from OTP memory. Afterwards the OTP memory can be manually accessed with three commands:

1. Reload shadow registers
2. Read OTP memory
3. Write OTP memory

The NVM\* and OTP\* I2C registers are used to access the OTP memory. Refer to Figure 11 to see these registers.

A screenshot of a computer

Description automatically generated with low confidence

Figure : NVM ACCESS registers[[6]](#footnote-6)

Table : OTP Read Modes

|  |  |
| --- | --- |
| OTP Read Mode | PTM |
| Normal Read Access | 000b |
| Margin-1 Read Mode | 110b |
| Margin-2 Read Mode | 111b |
| Off State Margin Read Mode | 101b |

Table : NVM SIZE Field

|  |  |
| --- | --- |
| SIZE Field | OTP Bytes |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |

The following sections show steps for OTP memory access.

## Reload Shadow Registers

The shadow registers may be refreshed from OTP memory on demand by performing the following steps:

1. Set the NVM\_CMD.RELOAD\_CMD bit to 1. This will start a reload of the shadow registers from OTP. When the reload operation completes the NVM\_STATUS.RELOAD\_DONE bit will go high.
2. Clear the NVM\_CMD.RELOAD\_CMD bit to 0. This handshake permits the NVM state machine to return to the IDLE state, where it can then recognize subsequent OTP access commands.

The NVM\_STATUS register provides feedback on the completed reload operation.

* The NVM\_STATUS.LOAD\_ERR bit indicates if the reload operation was successful.
* The NVM\_STATUS.LOAD\_PAGE field tells which of the three OTP pages was loaded into the shadow registers.
* The NVM\_STATUS.DBE and NVM\_STATUS.SBE bits indicate if there were any ECC errors during the reload operation. The SBE bit will be 1 when a correctable, single-bit error is encountered. The DBE bit will be 1 when an uncorrectable, dual-bit error occurs. ECC checking/correction is done on each of the five 64-bit blocks, so it is possible for both SBE and DBE to be set to 1.

With a fresh, unprogrammed device no valid page will be found. In this case the LOAD\_ERR bit is expected to be 1 and the LOAD\_PAGE field is expected to be 2.

Also, an alternative starting page may be specified in NVM\_CMD.START\_PAGE.

### Reload Test Options

The reload operation performs a series of OTP reads. Each OTP read can be made to use an alternative read mode. The OTP\_CTRL0.PTM field enables these alternative modes. Refer to Table 5 for a list of the OTP read options.

The entire reload operation can made to continuously repeat by setting the NVM\_CMD.LOOP bit to 1. This may be useful when making current measurements. Vader will keep reloading the shadow registers until the LOOP bit is cleared.

## Read OTP Memory

The OTP memory can be read without affecting the shadow registers. The NVM\_CMD.READ\_CMD bit is used to kick off an OTP read operation. The OTP memory is read into a 64-bit buffer. The 64-bit buffer is accessible via the NVM\_DATA\* registers. A single read operation can access 8, 16, 32 or 64 bits, depending upon the NVM\_CTRL.SIZE setting. The NVM\_ADDR register points to the location in OTP memory to read. Reads are done along natural boundaries, again depending upon the NVM\_CTRL.SIZE setting. After the read operation the NVM\_DATA\* registers will contain the OTP data.

1. Write the OTP address to the NVM\_ADDR register.
2. Write the access size to the NVM\_CTRL.SIZE register field.
3. Set the NVM\_CMD.READ\_CMD bit to 1.
4. Wait for the NVM\_STATUS.READ\_DONE bit to go high.
5. Clear the NVM\_CMD.READ\_CMD bit to 0 to complete the handshake with the NVM state machine.
6. Read the NVM\_DATA\* registers to access the 64-bit buffer data.

Please note that ECC error correction is enabled when the NVM\_CTRL.ECC bit is set to 1 and the NVM\_CTRL.SIZE field is set to 11b.

### Read OTP Test Options

Alternative read modes are available for testing the OTP. The OTP\_CTRL0.PTM field enables these read options.[[7]](#footnote-7) Refer to Table 5 for a list of the read options.

The read operation can made to continuously repeat by setting the NVM\_CMD.LOOP bit to 1. Vader will repeat the read operation until the LOOP bit is cleared. Note that the repeated read operation may consist of multiple actual OTP reads, depending upon NVM\_CTRL.SIZE.

The low-level timing of the OTP read pulse (PRD signal) can be manually controlled via the OTP\_CTRL0.TM\_PRD bit.

## Write OTP Memory

Writing to OTP memory is typically only done once. It also requires that the PGOOD pin level is raised to 7.5V.

1. Place the new OTP data values in the 64-bit buffer by writes to the NVM\_DATA\* registers.
2. Configure the NVM\_CTRL.SIZE and NVM\_CTRL.ECC fields as needed. For example, write 13h to NVM\_CTRL for all 64-bit of data and automatic ECC parity calculation.
3. Configure the OTP target address by writing to the NVM\_ADDR register.
4. Set the OTP\_CTRL0.PROG bit to 1.
5. Apply 7.5V to the **PGOOD** pin.
6. Connect the PGOOD pin to the OTP VPP signal. This is done by setting the OTP\_CTRL1.VPP\_ON bit to 1.
7. Set the NVM\_CMD.WRITE\_CMD bit to 1 to start the OTP write operation. Depending upon the NVM\_CTRL.SIZE setting 1, 2, 4 or 8 rows of OTP memory will be programmed. When the write operation finishes the NVM\_STATUS.WRITE\_DONE bit will go high.
8. Clear the NVM\_CMD.WRITE\_CMD bit to 0 to complete the handshake with the NVM state machine.
9. Disconnect the PGOOD pin from the OTP VPP signal. This is done by clearing the OTP\_CTRL1.VPP\_ON bit to 0.
10. Clear the OTP\_CTRL0.PROG bit to 0.
11. Remove 7.5V from the **PGOOD** pin

Please note that ECC error correction is enabled when the NVM\_CTRL.ECC bit is set to 1 and the NVM\_CTRL.SIZE field is set to 11b. When ECC error correction is enabled the ECC checksum will be automatically generated and bits [63:57] will be overwritten with the ECC checksum.

### Write OTP Test Options

The write operation can be made to automatically repeat by setting NVM\_CMD.LOOP to 1. Vader will repeat the write operation until the LOOP bit is cleared.

The low-level timing of the OTP write pulse (PWE) can be manually controlled via the OTP\_CTRL0.TM\_PWE bit.

The eMemory OTP cells defines test mode called “IPP mode”. This can be implemented by performing a write sequence with OTP\_CTRL0.TM\_PWE set to 1 and OTP\_CTRL1.VPP\_ON kept cleared to 0 (with VDD/VPP set to 2.5V). The PWE pulse width is recommended by eMemory to be over 30ms in IPP mode.

## Committing Shadow Registers to OTP Memory

The shadow register contents can be written to OTP memory with five 64-bit OTP write operations. The basic idea is to copy each block of shadow register data from the IMAGE\_DATA\* registers to the equivalent NVM\_DATA\* registers and then perform a 64-bit write.

1. Ensure that all shadow registers are configured with the correct values. Please note that the PAGE.PAGE\_VALID field should be set to 1010b.
2. Configure for 64-bit writes with ECC calculation enabled (write 13h to NVM\_CTRL).
3. Program BLOCK0
   1. write 00h to IMAGE\_ADDR and NVM\_ADDR
   2. copy IMAGE\_DATA0 … IMAGE\_DATA7 to NVM\_DATA0 ... NVM\_DATA7
   3. perform a 64-bit OTP write
4. Program BLOCK1
   1. write 08h to IMAGE\_ADDR and NVM\_ADDR
   2. copy IMAGE\_DATA0 ... IMAGE\_DATA7 to NVM\_DATA0 ... NVM\_DATA7
   3. perform a 64-bit OTP write
5. Program BLOCK2
   1. write 10h to IMAGE\_ADDR and NVM\_ADDR
   2. copy IMAGE\_DATA0 ... IMAGE\_DATA7 to NVM\_DATA0 ... NVM\_DATA7
   3. perform a 64-bit OTP write
6. Program BLOCK3
   1. write 18h to IMAGE\_ADDR and NVM\_ADDR
   2. copy IMAGE\_DATA0 ... IMAGE\_DATA7 to NVM\_DATA0 ... NVM\_DATA7
   3. perform a 64-bit OTP write
7. Program BLOCK4
   1. write 20h to IMAGE\_ADDR and NVM\_ADDR
   2. copy IMAGE\_DATA0 ... IMAGE\_DATA7 to NVM\_DATA0 ... NVM\_DATA7
   3. perform a 64-bit OTP write

This sequence of steps will transfer the shadow register configuration to the 1st page of the OTP memory. Alter the NVM\_ADDR values to instead target the 2nd page or 3rd page. Refer to Table 4 for a list of addresses to use when programming the various OTP pages.

To invalidate an existing OTP page with minimal updates, write 0Fh to the 1st byte of the page (PAGE\_VALID field). Alternatively, overwrite all 40 bytes of the existing page with FFh in each byte. This should mark the page as invalid and has the side-effect of creating correct ECC checksums.

# Test Modes

A screenshot of a computer

Description automatically generated with low confidence

Figure : DFT Registers[[8]](#footnote-8)

## Analog Test Modes

Support for analog test-mode, in the Vader digital logic, consists mostly of digital outputs back with volatile, I2C-accessible registers.

Table : Test Mode Signals/Registers

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Register | Field | Notes |
| d\_tm\_cp[7:0] | TM\_CP0 | \* |  |
| d\_tm\_cp[15:8] | TM\_CP1 | \* |  |
| d\_tm\_cp]23:16] | TM\_CP2 | \* |  |
| d\_tm\_en\_pmux\_a[1:0] | TM\_PMUX | PMUX\_A |  |
| d\_tm\_en\_pmux\_d[1:0] | TM\_PMUX | PMUX\_D |  |
| d\_tm\_i2c[1:0] | TM\_I2C | TM\_I2C |  |
| d\_tm\_porb | TM\_REF | PORB |  |
| d\_tm\_refbias | TM\_REF | REFBIAS |  |
| d\_tm\_testaddr\_a[7:0] | TM\_ADDR\_A | ADDR\_A |  |
| d\_tm\_testaddr\_d[7:0] | TM\_ADDR\_D | ADDR\_D |  |

## Overriding Digital-to-Analog Output Signals

Each vader\_digtop output can be individually controlled via writes to I2C -accessible registers. For dynamically changing outputs an override capability is provided via the TM\_OVER0, TM\_OVER1, TM\_OVER2 and TM\_OVER3 I2C registers. The override register field for each overridden signal consists of two bits, where the MSB enables the signal override and the LSB gives the forced value.

Table : Output Signal Override Field

|  |  |
| --- | --- |
| Register Override Field | Output Signal State |
| 00b | normal operation |
| 01b | normal operation |
| 10b | force output low |
| 11b | force output high |

Table : Output Overrides

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Registers | Field | Notes |
| d\_boost\_mode | TM\_OVER0 | OV\_BOOST\_MODE |  |
| d\_ratio[0] | TM\_OVER0 | OV\_RATIO0 |  |
| d\_ratio[1] | TM\_OVER0 | OV\_RATIO1 |  |
| d\_cp\_discharge | TM\_OVER0 | OV\_DISCHARGE |  |
| d\_state[0] | TM\_OVER1 | OV\_STATE0 |  |
| d\_state[1] | TM\_OVER1 | OV\_STATE1 |  |
| d\_state[2] | TM\_OVER1 | OV\_STATE2 |  |
| d\_en\_cp | TM\_OVER1 | OV\_EN\_CP |  |
| d\_en\_digital | TM\_OVER2 | OV\_EN\_DIGITAL |  |
| d\_pgate\_en | TM\_OVER2 | OV\_PGATE\_EN |  |
| d\_pgood\_valid | TM\_OVER2 | OV\_PGOOD\_VALID |  |
| d\_pgood\_pd\_n | TM\_OVER2 | OV\_PGOOD\_PD\_N |  |
| d\_cpclkm\_en | TM\_OVER3 | OV\_CPCLKM\_EN |  |
| d\_cpclkm\_start | TM\_OVER3 | OV\_CPCLKM\_START | Existing CP-clock measurement not affected |
| d\_dig\_read | TM\_OVER3 | OV\_DIG\_READ | Boost\_Mode and Ratio[1:0] pins will be resampled whenever d\_dig\_read is high |
| d\_softsw | TM\_OVER3 | OV\_SOFTSW | does not affect CP-clock frequency |

## Forcing State Machines

The main and power state machines can be forced to a given state via the TM\_STATE0 and TM\_STATE1 I2C registers. Write to the TM\_STATE1 register to set the forced state variables and write to the TM\_STATE0 register to force or release the Main and/or Power Control state machines. Refer to Table 10 and Table 11 to see the state codes.

When released the state machine will resume operation, starting from the forced state.

Table : Main State Codes

|  |  |
| --- | --- |
| Main State | Code |
| MAIN\_RESET | 0h |
| MAIN\_LOAD\_OTP | 1h |
| MAIN\_MEASURE\_CPCLK | 2h |
| MAIN\_IDLE | 3h |

Table : Power Control State Codes

|  |  |
| --- | --- |
| Power Control State | Code |
| PWR\_RESET | 00h |
| PWR\_INIT\_SETTLE | 01h |
| PWR\_CPCLKM\_WAIT | 02h |
| PWR\_PGATE | 03h |
| PWR\_FAULT\_CHECK | 04h |
| PWR\_WAIT\_CHECK | 05h |
| PWR\_CP\_START | 06h |
| PWR\_DISCHG\_PHASE | 07h |
| PWR\_DISCHG\_FLYCAP | 08h |
| PWR\_UP\_SOFTSTART | 09h |
| PWR\_UP\_ACTIVE | 0Ah |
| PWR\_DOWN\_BSTCHARGE | 0Bh |
| PWR\_DOWN\_SOFTSTART | 0Ch |
| PWR\_DOWN\_ACTIVE | 0Dh |
| PWR\_HICCUP | 0Eh |
| PWR\_SHUTDOWN | 0Fh |
| PWR\_OFF | 10h |
| PWR\_LATCH\_FAULT | 11h |

## Diagnostic Digital Mux

Internal information can be brought out to the **RCLK** and/or **Boost\_Mode** pins on the Vader device.

To show diagnostic data on the **RCLK** pin, do the following steps:

1. Write 63h to the TM\_ADDR\_D register
2. Write 10h to the TM\_PMUX register
3. Update the TM\_DIAG0 register to select the desired diagnostic data

To show diagnostic data on the **Boost\_Mode** pin, do the following steps:

1. Write 63h to the TM\_ADDR\_D register
2. Write 20h to the TM\_PMUX register
3. Update the TM\_DIAG1 register to select the desired diagnostic data

To enable diagnostic data on both the **RCLK** and **Boost\_Mode** pins, write 30h to the TM\_PMUX register.

See Table 12 for the list of selectable internal signals.

Table : Diagnostic Data

|  |  |  |
| --- | --- | --- |
| TM\_DIAG\*.SELECT | Signal | Source |
| 0 | LOW | N/A |
| 1 | HIGH | N/A |
| 2 | NVM\_DBE | nvm\_ctrl |
| 3 | NVM\_SBE | nvm\_ctrl |
| 4 | MAIN\_RESET | main\_ctrl |
| 5 | MAIN\_LOAD\_OTP | main\_ctrl |
| 6 | MAIN\_MEASURE\_CPCLK | main\_ctrl |
| 7 | MAIN\_IDLE | main\_ctrl |
| 8 | PWR\_RESET | power\_ctrl |
| 9 | PWR\_INIT\_SETTLE | power\_ctrl |
| 10 | PWR\_CPCLKM\_WAIT | power\_ctrl |
| 11 | PWR\_PGATE | power\_ctrl |
| 12 | PWR\_FAULT\_CHECK | power\_ctrl |
| 13 | PWR\_WAIT\_CHECK | power\_ctrl |
| 14 | PWR\_CP\_START | power\_ctrl |
| 15 | PWR\_DISCHG\_PHASE | power\_ctrl |
| 16 | PWR\_DISCHG\_FLYCAP | power\_ctrl |
| 17 | PWR\_UP\_SOFTSTART | power\_ctrl |
| 18 | PWR\_UP\_ACTIVE | power\_ctrl |
| 19 | PWR\_DOWN\_BSTCHARGE | power\_ctrl |
| 20 | PWR\_DOWN\_SOFTSTART | power\_ctrl |
| 21 | PWR\_DOWN\_ACTIVE | power\_ctrl |
| 22 | PWR\_HICCUP | power\_ctrl |
| 23 | PWR\_SHUTDOWN | power\_ctrl |
| 24 | PWR\_OFF | power\_ctrl |
| 25 | PWR\_LATCH\_FAULT | power\_ctrl |
| 26 | a\_cp\_status[0] | input |
| 27 | a\_cp\_status[1] | input |
| 28 | a\_cp\_status[2] | input |
| 29 | a\_cp\_status[3] | input |
| 30 | a\_cp\_status[4] | input |
| 31 | a\_cp\_status[5] | input |
| 32 | a\_cp\_status[6] | input |
| 33 | a\_cp\_status[7] | input |
| 34 | a\_cp\_status[8] | input |
| 35 | a\_cp\_status\_dgl[0] | deglitcher |
| 36 | a\_cp\_status\_dgl[1] | deglitcher |
| 37 | a\_cp\_status\_dgl[2] | deglitcher |
| 38 | a\_cp\_status\_dgl[3] | deglitcher |
| 39 | a\_cp\_status\_dgl[4] | deglitcher |
| 40 | a\_cp\_status\_dgl[5] | deglitcher |
| 41 | a\_cp\_status\_dgl[6] | deglitcher |
| 42 | a\_cp\_status\_dgl[7] | deglitcher |
| 43 | a\_cp\_status\_dgl[8] | deglitcher |
| 44 | a\_cpclkm\_out | input |
| 45 | a\_cpclkm\_out\_dgl | deglitcher |
| 46 | a\_boost\_mode | input |
| 47 | a\_en\_det | input |
| 48 | a\_en\_det\_dgl | deglitcher |
| 49 | a\_pgate\_on | input |
| 50 | a\_pgate\_on\_dgl | deglitcher |
| 51 | a\_ratio[0] | input |
| 52 | a\_ratio[1] | input |
| 53 | a\_tsd | input |
| 54 | a\_tsd\_dgl | deglitcher |
| 55 | d\_boost\_mode | output |
| 56 | d\_clk\_cpa | output |
| 57 | d\_clk\_cpb | output |
| 58 | d\_clkss | output |
| 59 | d\_cp\_discharge | output |
| 60 | d\_cpclkm\_en | output |
| 61 | d\_cpclkm\_start | output |
| 62 | d\_dig\_read | output |
| 63 | d\_en\_cp | output |
| 64 | d\_en\_digital | output |
| 65 | d\_pgate\_en | output |
| 66 | d\_pgood\_pd\_n | output |
| 67 | d\_pgood\_valid | output |
| 68 | d\_ratio[0] | output |
| 69 | d\_ratio[1] | output |
| 70 | d\_softsw | output |
| 71 | d\_state[0] | output |
| 72 | d\_state[1] | output |
| 73 | d\_state[2] | output |
| 74 | reg\_clk\_req | regclk\_enable |
| 75 | vout\_fault | power\_ctrl |

# C Code to Calculate Error Correction Code

#include "stdio.h"  
#include "stdint.h"  
  
void show\_parity(uint64\_t dw, uint8\_t expected);  
uint8\_t ecc\_parity(uint64\_t dw);  
uint8\_t calc\_parity64(uint64\_t d);  
uint8\_t calc\_parity8(uint8\_t d);  
  
main() {  
 show\_parity(0x0000000000000000LL, 0x00);  
 show\_parity(0x01ffffffffffffffLL, 0x7f);  
 show\_parity(0x000605961e037fd4LL, 0x4a);  
 show\_parity(0x01354723bda118f5LL, 0x40);  
 show\_parity(0x004fb53740d8d1bcLL, 0x10);  
 show\_parity(0x000a6d394fe2bcc1LL, 0x03);  
 show\_parity(0x01d2b047ef0b9292LL, 0x63);  
 show\_parity(0x013b86a8b5050892LL, 0x26);  
 show\_parity(0x00261110e3f07b93LL, 0x51);  
 show\_parity(0x01b3c39871268a34LL, 0x3a);  
 show\_parity(0x01eb8679c7c9fafaLL, 0x22);  
 show\_parity(0x002b788ed20d0a2fLL, 0x6c);  
 show\_parity(0x0052cf869544fb7cLL, 0x56);  
 show\_parity(0x00a811eb6cab8b46LL, 0x7a);  
 show\_parity(0x0093a6d5aef89bcbLL, 0x76);  
 show\_parity(0x01fa776c4aa239fbLL, 0x6c);  
 show\_parity(0x01761e20a4edd615LL, 0x30);  
 show\_parity(0x013ddfaac7bb4c07LL, 0x1b);  
 show\_parity(0x00cd69c0aa131784LL, 0x30);  
 show\_parity(0x015913991f6fcfe4LL, 0x43);  
 show\_parity(0x00f61c26b9c8c7beLL, 0x37);  
 show\_parity(0x0084220baa626c45LL, 0x3d);  
}  
  
void show\_parity(uint64\_t dw, uint8\_t expected) {  
 uint8\_t ecc;  
 ecc = ecc\_parity(dw);  
 printf("data=%16.16llx expected=%2.2x calculated=%2.2x %0s\n",  
 dw, expected, ecc, (ecc !expected) ? "ERR" : "OK");  
}

uint8\_t ecc\_parity(uint64\_t dw) {  
 uint8\_t p;  
 uint8\_t result;  
 uint64\_t temp;  
 const uint64\_t mask\_6 = 0x01fffffffc000000LL;  
 const uint64\_t mask\_5 = 0x01fffe0003fff800LL;  
 const uint64\_t mask\_4 = 0x01fe01fe03fc07f0LL;  
 const uint64\_t mask\_3 = 0x01e1e1e1e3c3c78eLL;  
 const uint64\_t mask\_2 = 0x019999999b33366dLL;  
 const uint64\_t mask\_1 = 0x0155555556aaad5bLL;  
 const uint64\_t mask\_0 = 0x01ffffffffffffffLL;  
 result = 0;  
 temp = mask\_6 & dw; p = calc\_parity64(temp); result = (result<<1) | p;  
 temp = mask\_5 & dw; p = calc\_parity64(temp); result = (result<<1) | p;  
 temp = mask\_4 & dw; p = calc\_parity64(temp); result = (result<<1) | p;  
 temp = mask\_3 & dw; p = calc\_parity64(temp); result = (result<<1) | p;  
 temp = mask\_2 & dw; p = calc\_parity64(temp); result = (result<<1) | p;  
 temp = mask\_1 & dw; p = calc\_parity64(temp); result = (result<<1) | p;  
 temp = mask\_0 & dw; p = calc\_parity64(temp); result = (result<<1) | p;  
 result = (result & 0xfe) | calc\_parity8(result);  
 return result;  
}

uint8\_t calc\_parity64(uint64\_t d) {  
 uint8\_t i;  
 uint8\_t count;  
 uint64\_t b;  
 count = 0;  
 b = 1LL;  
 for(i=0;i<64;i++) {  
 if(d&(b<<i)) {  
 count++;  
 }  
 }  
 return(count%2);  
}

uint8\_t calc\_parity8(uint8\_t d) {  
 uint8\_t i;  
 uint8\_t count;  
 uint8\_t b;  
 count = 0;  
 b = 1;  
 for(i=0;i<8;i++) {  
 if(d&(b<<i)) {  
 count++;  
 }  
 }  
 return(count%2);  
}

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Author | Date | Notes |
| 0.1 | Robert Macomber | 6/8/2023 | Initial version. |
| 0.2 | Robert Macomber | 8/28/2023 | Clarifications and corrections. |

1. Normally, an unprogrammed EPROM bit will default to 1. The eMemory OTP cell inside Vader inverts the data when it is read. Some pSemi devices perform a second inversion on the inverted read data; which Vader does not do. Thus, unprogrammed bits are 0 in Vader. [↑](#footnote-ref-1)
2. eMemory specifies UV erase conditions at 254nm wavelength at 18mW/cm2. Erase time can vary from 30 minutes to 120 minutes, depending upon UV erase tools. [↑](#footnote-ref-2)
3. Snapshot taken from vader\_register\_map.xlsm. [↑](#footnote-ref-3)
4. Snapshot taken from vader\_register\_map.xlsm. [↑](#footnote-ref-4)
5. Snapshot taken from vader\_register\_map.xlsm. [↑](#footnote-ref-5)
6. Snapshot taken from vader\_register\_map.xlsm. [↑](#footnote-ref-6)
7. The OTP\_CTRL0.PTM field affects the PTM[2:0] input signal on the OTP cell. Detailed information is available in the OTP cell documentation. See page 5 for documentation references. [↑](#footnote-ref-7)
8. Snapshot taken from vader\_register\_map.xlsm. [↑](#footnote-ref-8)