ODMB VME Baseline Firmware Guide

UCSB ODMB Team

October 20, 2020

1 Test Instructions

This is document is a guide to using the ODMB7 baseline firmware for testing VME communication and copper (x)DCFEB communication on the ODMB7 version 1 prototype boards. This firmware will confirm if VME signals are properly received and sent to the VME back plane as well as if JTAG signals are properly received and sent to the PPIB/(x)DCFEBs.

Physically, the ODMB should be inserted into an ODMB-compatible slot in a VME crate. There should be a computer with appropriate optical drivers and capable of running some version of Emulib that is connected to and able to control the crate's VCC. This computer should be loaded with the vme_cli software available at this repository. There should also be a computer equipped with the Vivado software and connected to the JTAG port of the ODMB via a Xilinx red box. This can be the same or separate from the computer handling VCC communication. Finally, the ODMB should be connected to the two skew-clear cables running to the PPIB. When testing communication with each (x)DCFEB, a powered (x)DCFEB will need to be plugged into the PPIB slot being tested. A diagram of this setup is shown in figure 1.

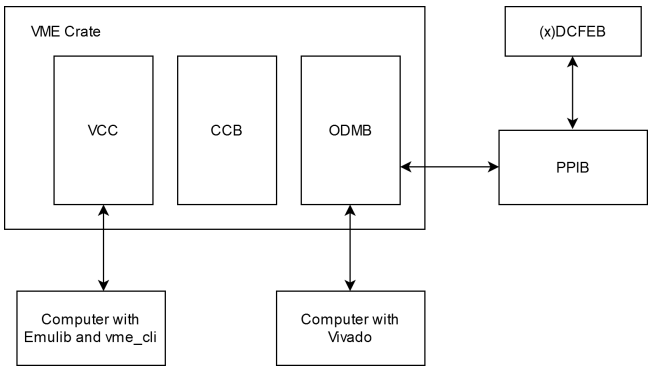

Figure 1: Test stand configuration for testing the ODMB7 VME baseline firmware.

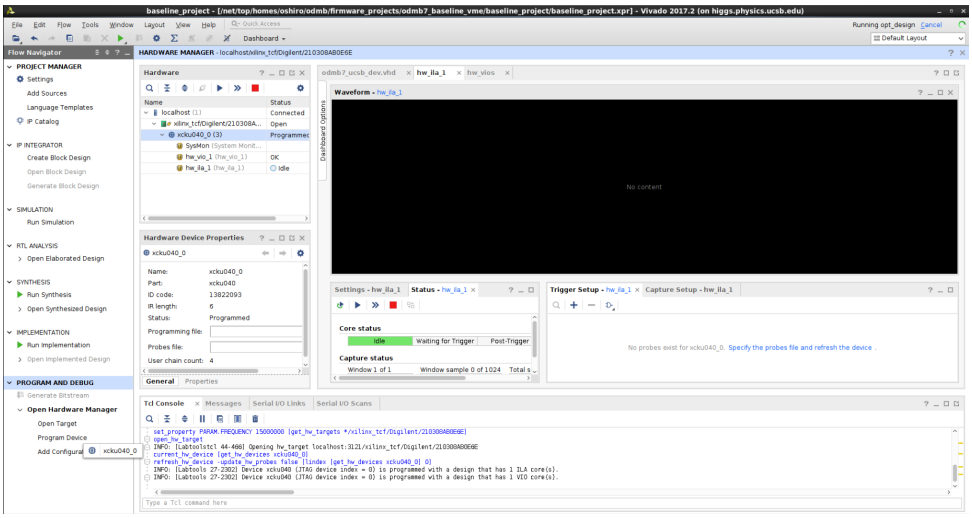

Figure 2: Programming device with Vivado, pictured for KCU105 evaluation board.

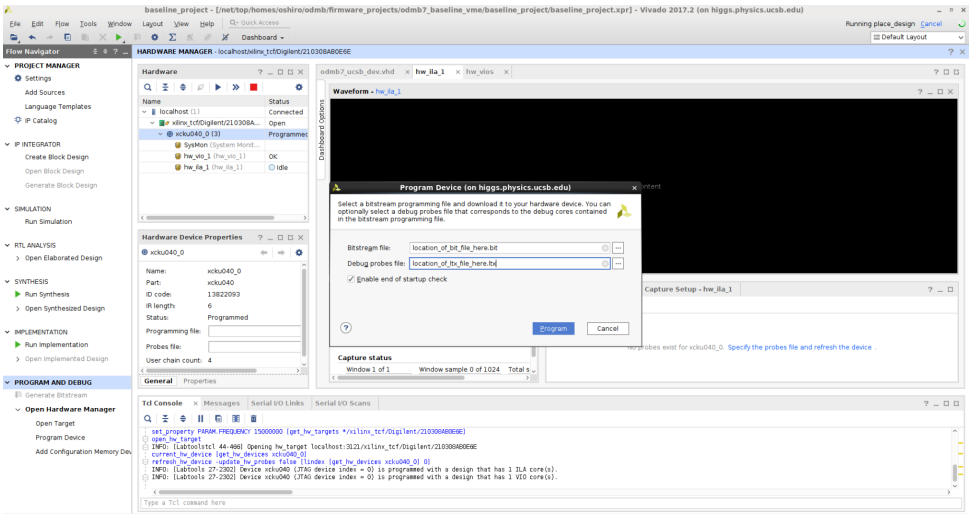

Figure 3: After clicking program device, the bit and ltx files can be specified.

To begin testing the VME firmware, the prototype boards should be loaded with the provided firmware and the Vivado program equipped with the appropriate test probes. The provided .bit and .ltx files can be loaded as shown in figures 2 and 3.

Once the firmware is loaded, VME commands can be sent using the vme_cli software. After compiling the software with make, a VME command can be issued to the ODMB in a format demonstrated by the following example.

```
./vme_cli --vcc_mac_address 02:00:00:00:00:4A --schar_port 2 --eth_name p5p2 --vme_write_read w
    --vme_command 1020 --vme_data ff --vme_slot 19
```

The VCC MAC address, `schar_port`, `eth_name`, and VME slot should be adjusted based on the configuration of the host computer and VME crate. The arguments `vme_write_read`, `vme_command`, and `vme_data` indicate the actual content of the command issued to the ODMB. The `vme_write_read` argument takes on the value `w` or `r` to indicate a write or read command respectively. In the case of read commands, no `vme_data` needs to be provided. The `vme_command` and `vme_data` take 16 bit values specified as 4 hex characters. `vme_data` values with fewer than 4 characters are interpreted by padding to the left with 0's. Below, commands will be listed in the format `<vme_write_read> <vme_command> (<vme_data>) (//comment on command)`.

As a first test, the command `R 4100` should be issued. The ODMB should respond with the return value `OD3B`, which will be displayed by the vme_cli program.

If this is successful, a more complicated series of commands can be issued to read the User code from the (x)DCFEBs. This is given in listing 1 for DCFEBs and listing 1 for xDCFEBs. The two `R 1014` commands will return the usercode, which should be `DCFEB###` where the #'s depend on the firmware version of the (x)DCFEB. Note that the lower 16 bits get returned first.

```
W 1018 0000 //reset DCFEBs
W 1020 #    //Select DCFEBs, use #=01, 02, 04, 08, 10, 20, or 40 to select DCFEB 1, 2, 3, 4, 5, 6, or
    7 respectively
W 191C 3C8  //shift instruction "read usercode"
W 1F04 0000 //shift 16 upper data bits
R 1014      //read retrieved data bits
W 1F08 0000 //shift 16 lower data bits
R 1014      //read retrieved data bits
```

```
W 1018 0000 //reset DCFEBs
W 1020 #    //Select DCFEBs, use #=01, 02, 04, 08, 10, 20, or 40 to select DCFEB 1, 2, 3, 4, 5, 6, or
    7 respectively
W 1934 3C8  //shift instruction "read usercode"
W 1F30 FFFF //bypass instruction to other xDCFEB devices
W 1F30 FFFF //bypass instruction to other xDCFEB devices
W 1F30 FFFF //bypass instruction to other xDCFEB devices
W 1338 F    //bypass instruction to other xDCFEB devices
W 1F04 0000 //shift 16 upper data bits
R 1014      //read retrieved data bits
W 1F08 0000 //shift 16 lower data bits
R 1014      //read retrieved data bits
```

If the returned values from `R 4100` or `R 1014` are not as expected, more information on debugging is given in the next section.

2 Debugging

2.1 Check Clocks

The first check to perform if the `R 4100` command fails is to check the ODMB clocks. If the CMS clock is not received from the CCB or if the clock manager is not functional, the ODMB firmware will not be able to handle VME communication. To diagnose these problems, one can first run the ILA un-triggered. If the ILA does not respond, this is indicative of no clock signal, which can be caused by either not receiving the CMS clock from the CCB or a malfunction of the clock manager in firmware.

If it is suspected that the CMS clock is not being received, an alternative firmware version with the ILA directly reading the CMS clock signal can be generated. Alternatively, the firmware can be replaced with an alternate version that uses the clock from the on-board clock synthesizer, which must be configured using appropriate software. Failure to see the CMS clock directly and success with the alternate firmware indicates a problem in receiving the clock from the CCB.

If the alternative firmware that reads the clock signal directly from the CCB demonstrates the clock is being received, then the clock manager module in the firmware is likely at fault and firmware debugging is required.

2.2 Check VME Signals

If the clock signals look as expected, the next debugging step is to check that VME signals are being sent/received. To diagnose VME problems, the VME crate should be restarted and the ODMB ILA equipped with the VME signals listed in table 1. This table also lists the expected behavior of the signals and their response to a `R 4100` command in simulation is shown in figure 4. Once the ILA is equipped with the appropriate signals, it should be triggered on `vme_as_b`=0. Once the ILA is armed, a `R 4100` command should be issued from vme_cli.

The first thing to check are the strobes and dtack. The VCC should pull `vme_as_b` low, then both bits of `vme_ds_b`. Shortly after `vme_ds_b` is pulled low, the `strobe` signal become 1. After some time, the ODMB should pull `vme_dtack_v6_b` low, and the VCC should de-assert `vme_ds_b`, then `vme_as_b`. If `vme_as_b` or `vme_ds_b` is never pulled low, and `vme_dir` is low, there may be some problem receiving the appropriate signal, which could be caused by ICs or traces on the ODMB. Further debugging may be performed with an oscilloscope on appropriate IC legs. If `vme_as_b` and `vme_ds_b` are pulled low but `strobe` never goes high, the other `vme_` signals listed in table 1 should be checked against their expected values. If these match their expected values, additional firmware will need to be generated to debug firmware issues.

If the correct sequence of strobes and dtack is observed, then the VME command is being received by the ODMB and the signals `vme_data_out`, `vme_tovme`, and `vme_doe_b` should be checked against their expected values. If these signals match their expected values, external ICs and signal traces may need to be debugged.

If `vme_data_out` is incorrect or if `strobe` is asserted but `vme_dtack_v6_b` never goes low, it is likely that the VME modules in the firmware are at fault. The `dummy_confregs` module that handles the `R 4100` command is very simple and thus not expected to fail. Information on debugging the `cfebjtag` module is given in the next section.

Signal	Long name	Expected behavior
`vme_data_in_buf`	Data in	Value passed by vme_cli for write commands.
`vme_data_out_buf`	Data out	Value returned by ODMB.
`vme_crate_addr`	Address	The slot number should match the bitwise inverse of `vme_ga_b` when `vme_as_b` goes low.
`vme_cmd`	Command	Should match value passed by vme_cli.
`vme_am`	Address Modifier.	When `vme_as_b` goes low, this should be 111X10 or 111X01.
`vme_gap_b`	Geographical address parity.	Should be 1 if an odd number of bits in ga are 0 and 0 otherwise.
`vme_ga_b`	Geographical address.	Should correspond to the VME slot used.
`vme_as_b`	Address strobe.	Should go low to indicate address ready to read.
`vme_ds_b`	Data strobe.	Both bits should go low after `vme_as_b` to indicate data ready to read.
`vme_sysfail_b`	Sysfail.	Should be 1.
`vme_berr_b`	Bus Error.	Should be 1, but doesn't matter for ODMB.
`vme_iack_b`	Interrupt acknowledge.	Should be 1.
`vme_lword_b`	Load word.	Should be 1 when `vme_as_b` goes low.
`vme_write_b`	Write.	Should be 0 for write commands and 1 for read commands.
`vme_dtack_v6_b`	Data acknowledge.	Should be issued by ODMB some time after receiving `vme_ds_b` low.
`vme_dir`	To VME	Should be go high when read commands are sent and low otherwise.
`vme_doe_b`	Output Enable	Should go low when read commands are sent.
`strobe`	Stobe	Should go high shortly after `vme_ds_b` goes low.

Table 1: Signals in VME interface

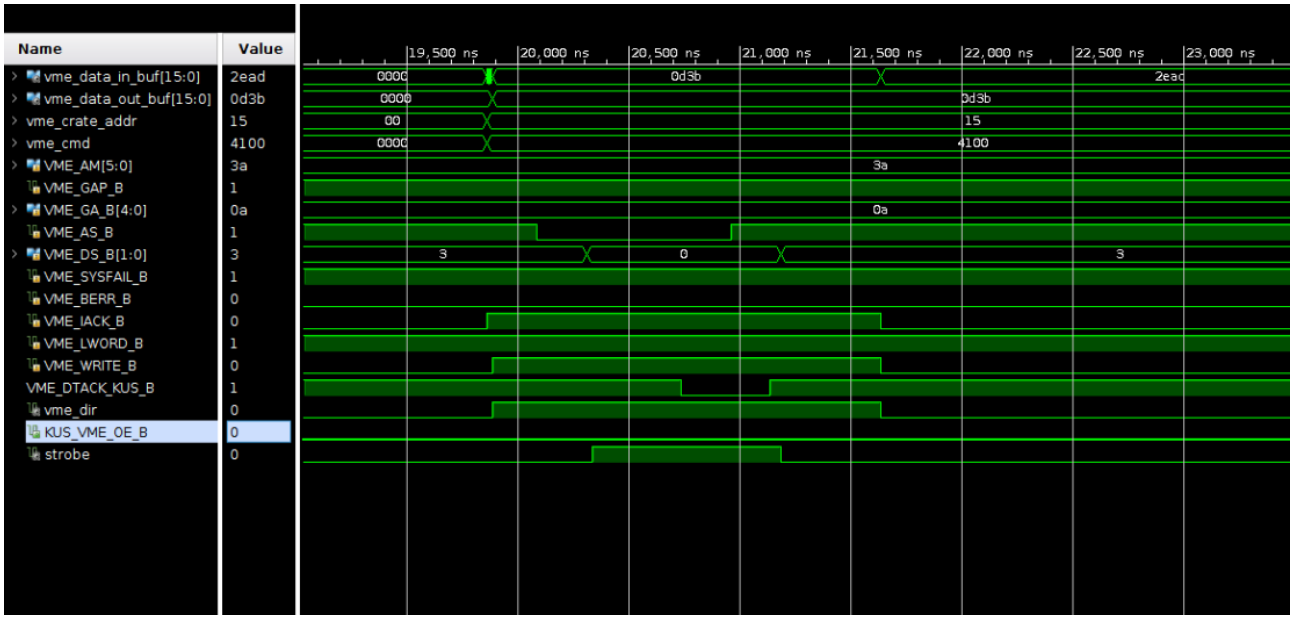

Figure 4: Simulated ODMB response to VME command `R 4100`. Note that in simulation `2EAD` appears on the `vme_data_in` line while in the real ODMB, a value of `0000` is expected.

2.3 Check CFEBJTAG

If the user code can not be successfully read back from the (x)DCFEBs, the first thing to check are the JTAG communication lines. These signals are `tms`, `tck`, `tdi`(selected (x)DCFEB index), and `tdo`(selected (x)DCFEB index), and are listed along with the other signals relevant to `cfebjtag` in table 2. Note that the signals in the ILA are prefixed with `dcfeb_`. Their expected behavior along with that of other `cfebjtag` signals is shown in figures 5 through 11.

As a first qualitative check, after each `W 1X0Y`, `W 1X1Y`, or `W 1X3Y` command, the `tms` signal and the `tck` only for the selected (x)DCFEB should respond with some pattern. This can be observed by again triggering on `vme_as_b`=0. If they do not respond, this indicates a bug in generating a `busy` signal or appropriate `tms` response and the appropriate paragraph below should be referenced. If other `tck` clocks are running, this indicates a problem in selecting the correct dcfeb and the `selfeb` signal should be monitored as discussed below.

If `tms` and `tck` seem to respond to appropriate commands, one should check that the correct `tdo` bits are being returned from the (x)DCFEBs. When the `W 1F04` and `W 1F08` commands are issued, one can check the value of `tdo` on rising edges of `tck` after `tms` has been 0 for two cycles up to and including the first `tck` edge for which `tms` is 1. The bits should be `DCFEB###` noting that the shifted bits on `tdo` appear in reverse order.

If the `tdo` bits are not correct, it may be necessary to inspect each `W 1X0Y`, `W 1X1Y`, and `W 1X3Y` command issued in detail. Figures 5 through 11 show the expected behavior for each command. To compare to the figures, check the value of `tdi` and `tms` against the values shown in the figures for each rising edge of `tck`. If the `tdi` or `tms` are incorrect, this indicates a bug in the logic that generates the `tms` patterns or transfers the input to `tdi`.

If the `tms` clock never begins running after a (x)DCFEB communication command is issued, this indicates either a problem in generating a `busy` signal or a failure to select any (x)DCFEBs. To diagnose the prior issue, check the `strobe`, `load`, `busy`, and `tck_global` signals after issueing a command. The load should go high once after receiving `strobe` high. If this does not happen, check that `vme_cmd` is correct. After load goes high, `busy` should go high until the communication is finished and while `busy` is running, `tck_global` should be running. If this does not happen, more debugging will be required for the `busy` logic.

If multiple `tms` clocks are running or if `tck_global` is running but no other `tck` signals, this indicates a problem with selecting (x)DCFEBs. One should check the `selfeb`signal after issuing a `W 1020` to see if updates to reflect selected (x)DCFEBs. The command `R 1024` can also be used to read back the currently selected (x)DCFEBs to check. After issuing a `W 1020` command, one run the ILA untriggered or issue `R 1024` to see if the selected (x)DCFEBs are still selected. If not, it is likely the ODMB received a spurious reset, and additional firmware should be made to investigate spurious reset signals.

If `tms` is not generated properly for a particular command, signals involved in `tms` pattern generation should be investigated. In particular, for `W 1Y1C` and `W 1Y34`, the `shihead` signal should be asserted to shift the instruction header before shifting data. For the `W 1Y04` command, the `shdhead` signal should be asserted to shift the data header. For all JTAG shift commands, the `shdata` signal should be asserted to shift the `tdi`/`tdo` data. Then, for `W 1Y1C`, `W 1Y38`, and `W 1Y08` commands, the `shtail` signal should be asserted to shift the tailer at the end of the shifting. If any of these signals are not asserted, additional firmware should be generated to analyze the VHDL logic generating the signal in question. If the signals are generated, but the `tms` or `tdi` patterns are not correct, additional firmware should be generated to analyze the shifting logic.

Signal	Description	Expected behavior
`dcfeb_tms`	JTAG Control	While `tck` is running, this should various values depending on command issued. See figures 5 through 11 for expected behavior for DCFEB commands
`dcfeb_tck`	JTAG Clock	After a `W 1X0Y`, `W 1X1Y`, or `W 1X3Y` command, the TCK for the selected (x)DCFEB should oscillate for several cycles
`dcfeb_tdi`	JTAG Input	When `tck` is running, the bits matching the provided `vme_data` should be seen.
`dcfeb_tdo`	JTAG Output	During the final `W 1F04` and `W 1F08` commands, the `tdo` signal for the selected (x)DCFEB should be the bits for its usercode.
`load`	Load	Should go high once for each (x)DCFEB communication command issued.
`busy`	Busy	Should be asserted while JTAG shifts in progress.
`tck_global`	Global JTAG clock.	Should be sent to selected (x)DCFEB(s) as `tck`.
`selfeb`	Selected (x)DCFEB	Should match selected (x)DCFEB(s) after issuing `W 1020` command.
`shihead`	Shift instruction header.	Should go high for `W 1Y1C` and `W 1Y34` commands.
`shdhead`	Shift data header.	Should go high for `W 1Y04` commands.
`shdata`	Shift data.	Should go high for all JTAG shift commands commands.
`shtail`	Shift tailer.	Should go high for `W 1Y08`, `W 1Y1C`, and `W 1Y38` commands.

Table 2: Signals related to JTAG debugging

Figure 5: Simulated ODMB response to VME command `W 1018 0000`. Note that this JTAG reset commands causes `tck` for each (x)DCFEB to run.

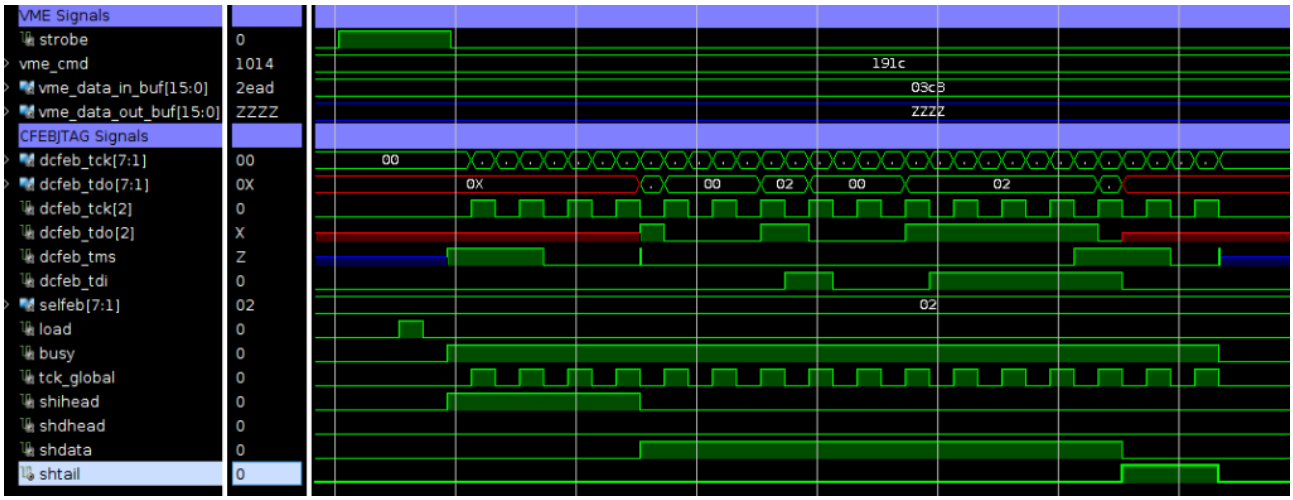

Figure 6: Simulated ODMB response to VME command `W 191C 03C8`. In this example, DCFEB 2 is selected. The `tdo` signals are unimportant and may differ for real DCFEBs.

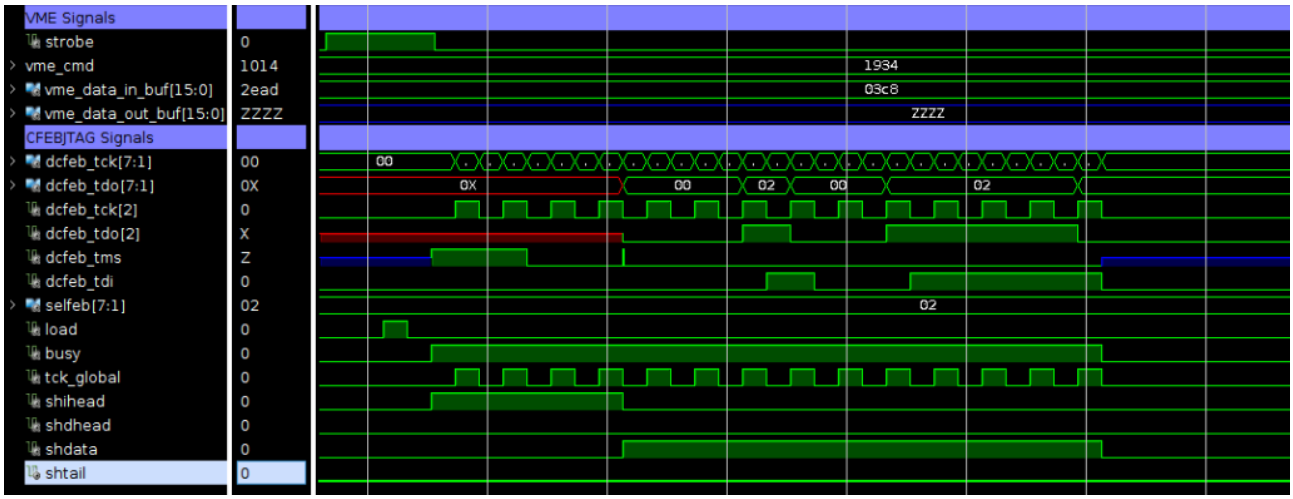

Figure 7: Simulated ODMB response to VME command `W 1934 03C8`. In this example, xDCFEB 2 is selected. The `tdo` signals are unimportant and may differ for real xDCFEBs.

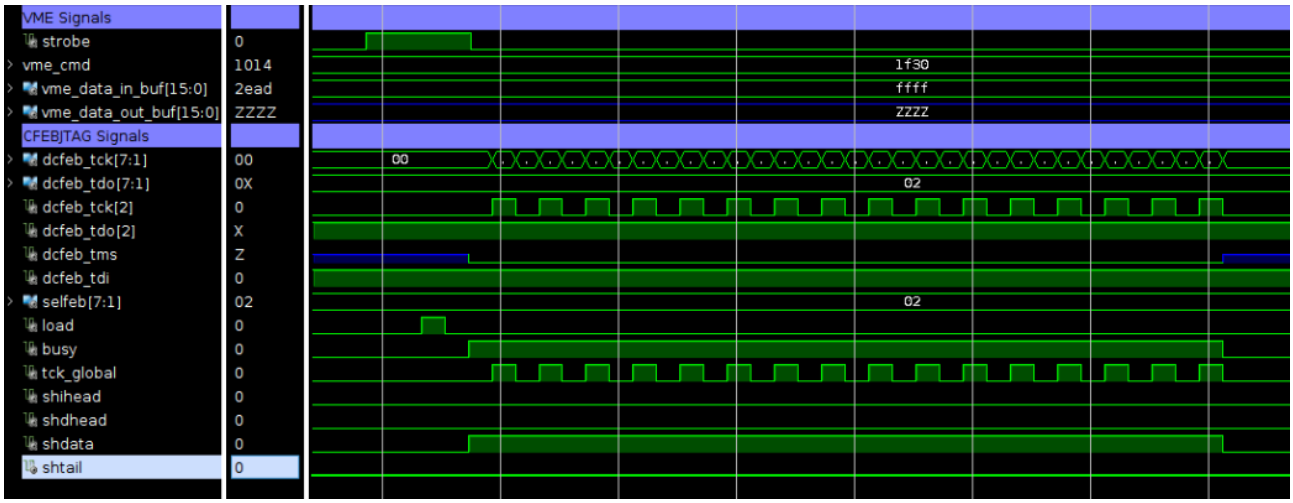

Figure 8: Simulated ODMB response to VME command `W 1F30 FFFF`. In this example, xDCFEB 2 is selected. The `tdo` signals are unimportant and may differ for real xDCFEBs.

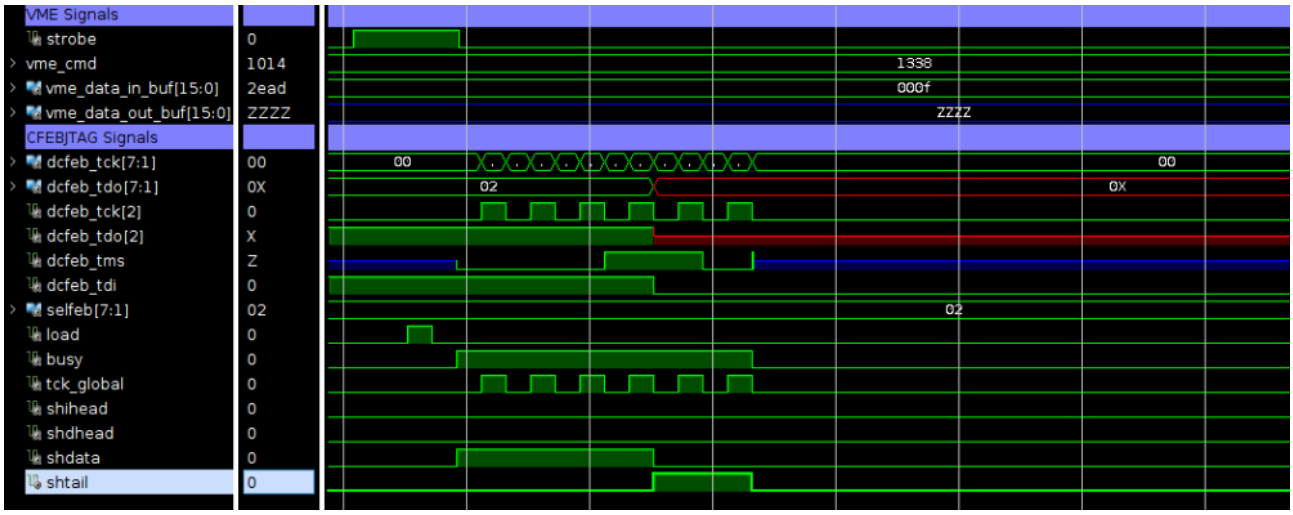

Figure 9: Simulated ODMB response to VME command `W 1338 000F`. In this example, xDCFEB 2 is selected. The `tdo` signals are unimportant and may differ for real xDCFEBs.

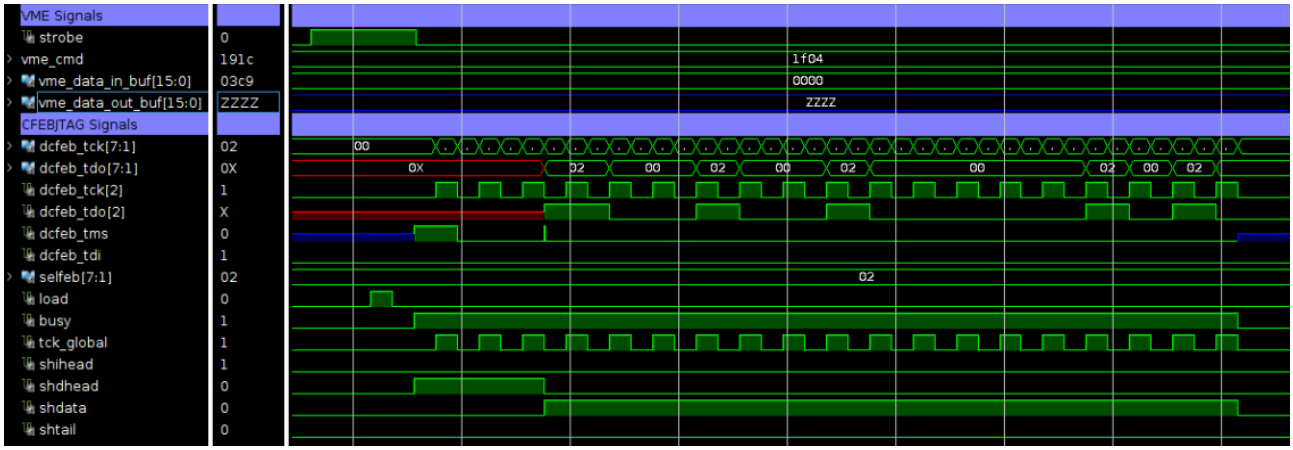

Figure 10: Simulated ODMB response to VME command `W 1F04 0000`. In this example, xDCFEB 2 is selected. The `tdo` signal in simulation is `A093` rather than `B###` expected for a real (x)DCFEB.

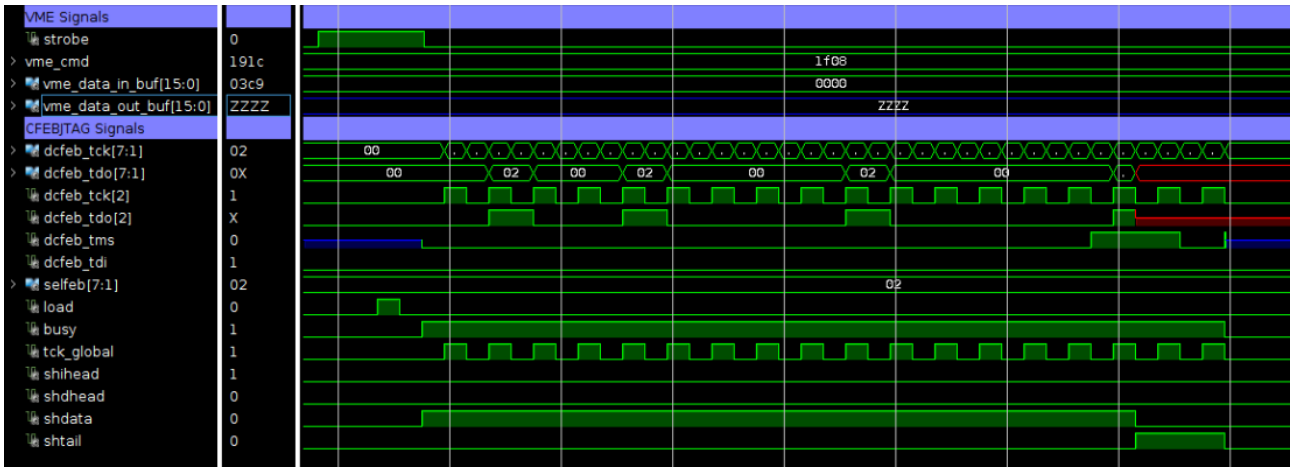

Figure 11: Simulated ODMB response to VME command `W 1F08 0000`. In this example, xDCFEB 2 is selected. The `tdo` signal in simulation is `0424` rather than `DCFE` expected for a real (x)DCFEB.