VME TestBencher Pro Example

Table of Contents

1.Overview

This example was created for VHDL and Verilog and can be found in the following directories:

```
<INSTALL>\Examples\Verilog\VME
<INSTALL>\Examples\VHDL\VME
```

The VME example demonstrates how you would create bus-functional-models (BFMs) for the following VME components: arbiter, master, and slave. It also demonstrates how you might configure a set of slave BFM instances to respond to different address ranges. Each of these BFMs are represented by a TestBencher Pro project which are all instantiated in a project named VME.hpj. This entire example is composed of unclocked diagrams. Here is the full project hierarchy:

```
VME.hpj
    VME_arbiter.hpj
    VME_master.hpj
        VME_requester.hpj
    VME_slave.hpj
```

Some features demonstrated in this example include:

1. Component Signals & Ports dialog -used to specify ports of each BFM.
2. Edge Sensitive Signals – used throughout most of the transactors.
3. User-defined condition Samples – master write and read transactors.
4. Project Level Variable – VME_slave “memory”.
5. Store Sampled Value to Variable – VME_slave write transactor.
6. Multiple Delay Resolution – VME_requester
7. Diagram Input Variables – VME_slave – used to configure address range.

Open VME.hpj and look at the Project window to see the hierarchy of projects. The instantiations are displayed below the top level Component Model. You can browse the contents of master.hpj and slave.hpj via the Project Library folder.

Throughout the diagrams in this example, names for delays, setups, and holds were chosen based on the VME specification. Typically, the numbers used in the names match up with the timing parameters given in the spec. Although some of the delays don't have a corresponding timing parameter in the spec. VME also has a set of “rules” and you'll see names that contain “Rule” in them for these.

2.VME_arbiter.hpj

There are multiple arbitration schemes supported by the VME specification. This BFM could theoretically contain a transactor for each scheme. But, for this example it only implements one of them and has only one transaction diagram, sgl_arbiter.btim. This is a “Slave Transactor” and responds to requests put out on the BR3 control signal. It only grants requests to the bus when there are no transactions in progress. Here is a detailed sequence of what this transactor does when running.

2.1.sgl_arbiter.btim

1. WaitForRequest Sample: waits up to 50 ns for an assertion on BR3. If an assertion on BR3 is detected, it will trigger delay D0 which will assert BG3. If the Sample times out before BR3 asserts, then it will restart the diagram.
2. Wait for BBSY to assert. This is done by setting up BBSY as a “Falling Edge Sensitive” signal and drawing the falling edge where the arbiter should wait.
3. D1 Delay: this is triggered by the falling edge of BBSY. It will release grant of the bus by de-asserting BG3. This delay isn't strictly necessary since the diagram is already waiting on the falling edge on BBSY. It was put in place to make sure that the delay stays fixed to 10 ns when editing the diagram. It also makes clear that the delay between BBSY and BG3 is not arbitrarily chosen.
4. Wait for BBSY to de-assert. This is done by setting up BBSY as a “Rising Edge Sensitive” signal and drawing the rising edge where the arbiter should wait.
5. Repeat steps 1-4. This looping affect is achieved by making sgl_arbiter a “Slave Transaction”. This setting is set when adding the diagram to the project. But, it can also be changed by going to the “Diagram Setting” dialog.

3.VME_master.hpj

The master BFM consists of four transactors and one requester BFM instance. The requester BFM is explained in greater detail in its own section. Essentially, the master uses the requester to communicate with the arbiter to receive access to the VME bus. The requester could have been implemented as part of the master BFM, but there are other types of VME components (not used in this example) that need to request access to the bus, so the requester behavior was encapsulated in a separate BFM.

3.1.requestbus.btim and releasebus.btim

These are the two transactors that interact with the requester BFM. The “requestbus” transactor asserts DEVICE_WANTS_BUS, then waits for DEVICE_GRANTED_BUS to assert. The “releasebus” transactor *de-asserts* DEVICE_WANTS_BUS, then waits for DEVICE_GRANTED_BUS to *de-assert*.

3.2.write.btim and read.btim

These two transactors are similar, so only the write.btim will be explained in detail. The write transactor has two input arguments: $$addr and $$data. The read transactor has one input argument, $$addr, and one output argument, SampleData. Here is the sequence of events for write.btim:

1. Put address on “A” bus.
2. DTACK_highcheck Sample: waits until DTACK is de-asserted. This is a level sensitive wait. Full Expect is disabled, the multiplier is set to 1000 and Blocking is enabled. This combination causes the transactor to wait up to 1000 ns for DTACK to de-assert.
3. BERR_highcheck Sample: works just like the DTACK_highcheck sample, but waits for BERR to de-assert.
4. Assert AS after a delay of D4 (10 ns).
5. Put data on “D” bus. This is done by the AS_to_DS delay.
6. Assert DS0 and DS1. This is either done by the D8 delays or D10 delays, whichever will cause the latest transition. Double-click on the falling edge to open the “Edge Properties” dialog. Notice that “Latest Transitions” is selected. The Multiple Delay Resolution can be configured separately for each edge in the diagram.
7. Wait for DTACK to assert. DTACK is set up to be “Falling Edge Sensitive”.
8. Release all the control signals based on the appropriate delays.

During this diagram's execution, all of the Setup and Hold checks are running in parallel. If any of these checks fail and error message will be output to the simulator's log file.

4.VME_slave.hpj

The slave BFM contains three transactors: gluelogic, read and write. All of these transactors are configured to be “Slave Transactors” (set in Diagram Settings). This will cause them all to run continuously in a looping mode when applied. The read and write diagrams are set up to respond to VME control signals. The gluelogic diagram defines a signal that represents DS0 “and” DS1. This signal is internal to the VME_slave project and is used in the write and read diagrams. Only the write diagram will be explained in detail here since the read is very similar. There is a variable named “memory” defined in this project. The “D_tomemory” Sample in the write diagram stores values in this variable. And this variable is used to drive the “D” bus in the read diagram.

Both the write and the read transactors have two input variables defined: lowValidAddress and highValidAddress. These define precisely what address range to respond to and are passed in via the transactor apply call.

4.1.write.btim

1. Wait for falling edge on AS (Address Strobe). AS is set up as “Falling Edge Sensitive”.
2. address Sample: This sample has a user-defined condition set to “(A >= write_args.lowValidAddress) && (A <= write_args.highValidAddress)”. This condition makes sure that the address on “A” is in the slave's address range. If it isn't then it will restart the diagram. To see how this is set up, double-click on the sample's name to open the Sample Properties dialog, then click on the HDL Code button to open the Code Generation Options dialog.
3. Wait for both DS0 and DS1 to assert. This is achieved by setting DS_and to be “Falling Edge Sensitive.”
4. CheckForWrite Sample: Since this is the write transactor, this sample checks to make sure that WRITE is asserted before continuing. If WRITE is *de-asserted* then the Sample will restart the diagram.
5. D28 Delay: assert DTACK after 30 ns.
6. Latch data while data strobes (DS0 and DS1) are asserted. This is done by the DS0_latch and DS1_latch signals. For example, the DS0_latch was created by setting the following in the Signal Properties dialog:
   a) Boolean Equation = D[7:0]
   b) Clock = DS0
   c) Edge/Level = low
7. Wait for both DS0 and DS1 to de-assert. This happens since DS_and is also a “Rising Edge Sensitive” signal.
8. D_tomemory Sample: This stores the latched data in a variable array named “memory”. In the Sample's Code Generation Options, “@memory[address-write_args.lowValidAddress]” was specified in the “Store Sampled Value to Variable” edit box. Open the Classes and Variables dialog for the VME_slave project to see how the “memory” variable was created.
9. Release DTACK after D30 delay.

5.VME_requester.hpj

The requester BFM contains one transactor, rwd_requester. This BFM is instantiated by the VME_master BFM and could be used by any VME BFM that needs to request access to the bus.

5.1.rwd_requester.btim

1. Wait for assertion of DEVICE_WANTS_BUS. This signal is set as “Falling Edge Sensitive”. The master's requestbus transactor will assert this when it needs access to the bus.
2. Assert BR3 after delay of D0 (5 ns). This is the bus request signal that the arbiter will respond to.
3. Wait for assertion of BG3IN, the bus grant signal. This is set as “Falling Edge Sensitive”.
4. Assert DEVICE_GRANTED_BUS, which will is what the master is waiting for.
5. Release BR3.
6. Assert BBSY after D1 (5 ns) to indicate that the bus is busy. The arbiter will continue to grant the master access to the bus as long as this busy signal is asserted.
7. Wait for BG3IN to de-assert. This is set as “Rising Edge Sensitive”.
8. Wait for DEVICE_WANTS_BUS to de-assert.
9. Release DEVICE_GRANTED_BUS.
10. Release BBSY. Note that there are four delays that drive this release: Rule3.9, Rule3.7, D3, and Rule3.10 This is to ensure that all the VME rules are being honored. The “Multiple Delay Resolution” for this edge is set to “Latest Transition”, which means the transactor will detect (during simulation) which delay will cause the latest transition to occur for this edge, then use that delay to drive the edge. You can edit this setting by double-clicking on the edge itself to bring up the “Edge Properties” dialog. Here's what each delay ensures:
    a) Rule3.9 – release of BBSY must be no less than 30 ns after BG3IN is asserted.
    b) Rule3.7 – release of BBSY must be no less than 90 ns after BBSY is asserted.
    c) D3 – release of BBSY must not occur before DEVICE_WANTS_BUS is de-asserted.
    d) Rule3.10 – release of BBSY must not occur before BG3IN is de-asserted.

6.VME.hpj

This is where all of the BFMs are connected together. There is no Model Under Test (MUT) in this example since it's purpose is to demonstrate how to create BFMs. Instead, the BFMs interact with each other (i.e. when the master performs a write the slave reacts and stores the data written to the bus). There are three slaves and four masters instantiated along with the arbiter instantiation. The VME bus grant daisy chain is achieved by connecting a series of slot*n*_BG3OUT signals to each master instance. The actual port connection is shown in the project window. If you want to see the formal port names, double-click on an instance to bring up the "Signals and Ports" dialog.

6.1.Sequencer Process (in Component Model)

Double-click on the Component Model in the project tree and scroll to the sequencer process, which is located near the bottom. Here is a breakdown of what the sequencer process does:

1. Starts up each slave with a given address range. In the VME_slave's component model, there are two user-defined tasks: StartRunning and StopRunning. The StartRunning task was written to take the address range as input, which then starts each transactor in a "no-wait" mode using that address range.
2. Starts up the requester for each master.
3. Starts up the arbiter.
4. Applies the reset transactor.
5. Performs writes and reads using the different masters.
6. Performs an invalid write, which results in a logged error during simulation.
7. Stops all "Slave Transactors": arbiter, requesters, and slaves.