Reactive Test Bench Tutorial 1

© Copyright 1994-2004 SynaptiCAD, Inc.

Table of Contents

1. Overview

This tutorial introduces some of the optional reactive test bench feature set. This feature set is included with TestBencher Pro and can be optionally be added to Waveformer Lite, Waveformer Pro, Datasheet Pro, and BugHunter Pro. When running any of these products, documentation on these features can be found in Help->Reactive Test Bench Generation Help.

The following features will be covered in this tutorial:

– Cycle-based test bench generation
– "For Loop" Markers
– Point Samples for checking model output
– Sensitive Edges
– Bi-directional signals

All of the relevant files for this tutorial can be found in "<SYNCAD INSTALL>\Examples\TutorialFiles\ReactiveTestBench". At the end of this tutorial, you will have created one timing diagram that uses many different reactive features. There are also pre-made diagrams for each completed step allowing you to start at any step of the tutorial desired. These completed diagrams can be found in the "ReactiveTestBench\CompletedDiagrams" directory.

2. The Model Under Test (MUT)

We will use a simplified version of a PCI slave device as the model to be tested. The model is contained in "mymut.v" and the module is named mymut. No experience with PCI is required to perform and understand this tutorial. There is no arbitration, the MUT responds to all addresses, and the only valid commands are single reads and writes. It contains a memory that can be written to and read from and has the following ports (all control signals are active low):

– CLK (input) – device is clocked on the negative edge
– FRAME (input) – indicates start of transaction.
– WRITE (input) – indicates write transaction.
– IRDY (input) – stands for "initiator ready". Indicates when the master device is ready for transaction to complete (the master will be the test bench in this case).
– TRDY (output) – stands for "target ready". During a write, this indicates that the MUT has finished writing data to it's memory. During a read, this indicates that the MUT has read the data from memory and put it on the DATA bus.
– ADDR (output) – Address to write to or read from.
– DATA (inout) – Data to write to memory or data that is read from memory.

Each transaction consists of an address cycle and data cycle. During the address cycle, the WRITE and ADDR signals must be valid. During a write data cycle, the DATA signal must be valid before IRDY is asserted. Then the MUT indicates that it is finished

storing the data by asserting TRDY. During a read data cycle, the MUT must drive DATA before asserting TRDY. Then, the master asserts IRDY when it is finished reading the data. Once IRDY or TRDY is asserted, they must remain asserted until the transaction is finished which is indicated by the de-assertion of FRAME.

3. Create signals

3.1. Extract ports from MUT

If you're running TBP or BHP you can create a new project that contains the mymut.v source file and use the "Extract MUT ports into Diagram" button to create all of the signals. If you're using Libero, the ports will automatically be extracted into a new diagram when WFL is launched.

3.2. Create clock waveform

Once the ports are extracted, convert the signal named "CLK" to a Clock by right-clicking on the name of the signal and selecting "Signal <-> Clock". This will draw a clock waveform with a default frequency of 10 MHz.

3.3. Set default clocking signal and edge

Next we set the "clocking signal" and edge for all of the signals in the diagram so that the test bench will be cycle-based instead of time-based (this means the test bench stimulus will change after waiting on clock transitions instead of time delays). Right-click in the signal name list in the diagram window and select "Diagram Properties." Select "CLK" as the default clock to use and "pos" as the Edge. Then click "Update Existing" to set the clock for existing signals. Press OK to close the dialog.
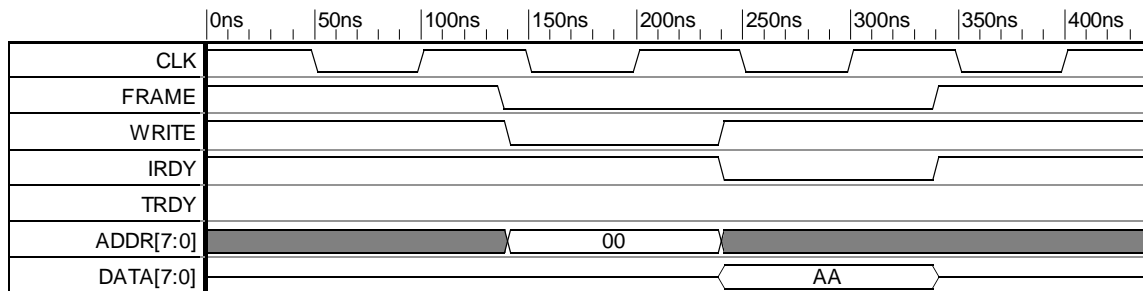
Following is an example of the difference between a cycle-based and time-based test bench. Both of these code segments were exported from the diagram you will be drawing in the next step. The example on the left is time-based and the example on the right is cycle-based.

```
                                     repeat (2)
                                     begin
                                       @(posedge CLK);
        #137;                        end
        FRAME_driver <= 1'b0;        FRAME_driver <= 1'b0;
        #3;
        WRITE_driver <= 1'b0;        WRITE_driver <= 1'b0;
        ADDR_driver <= 8'h00;        ADDR_driver <= 8'h00;
        #100;                          @(posedge CLK);
        WRITE_driver <= 1'b1;        WRITE_driver <= 1'b1;
        IRDY_driver <= 1'b0;          IRDY_driver <= 1'b0;
        ADDR_driver <= 8'hxx;        ADDR_driver <= 8'hxx;
        DATA_driver <= 8'hAA;        DATA_driver <= 8'hAA;
        #100;                          @(posedge CLK);
        FRAME_driver <= 1'b1;        FRAME_driver <= 1'b1;
        IRDY_driver <= 1'b1;          IRDY_driver <= 1'b1;
        DATA_driver <= 8'hzz;        DATA_driver <= 8'hzz;
        #101;                          @(posedge CLK);
```

4. Draw single write (without waiting on TRDY)

Draw the write transaction which is shown below. This transaction could be used as a simple test bench that just drives the input ports of the MUT, but it ignores the TRDY signal and doesn't verify that the data was actually written successfully to the MUT. We will add this functionality in the next couple of steps.
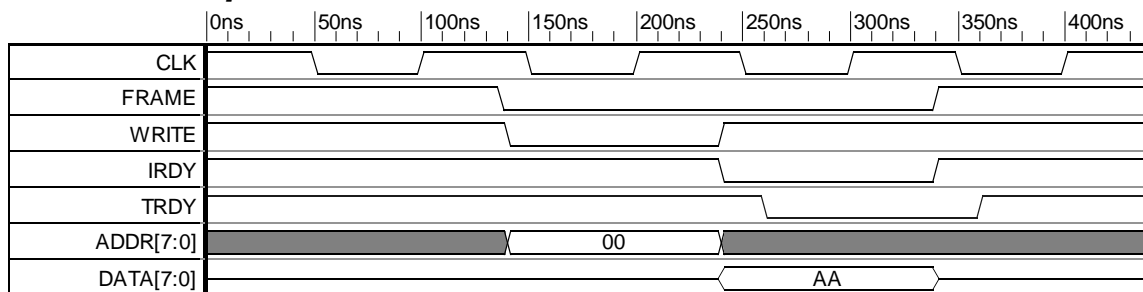
	0ns	50ns	100ns	150ns	200ns	250ns	300ns	350ns	400ns
CLK									
FRAME									
WRITE									
IRDY									
TRDY									
ADDR[7:0]				00					
DATA[7:0]						AA			

5. Add wait for TRDY assertion

There are two ways to perform this step. One method uses the Sensitive Edge feature and will wait indefinitely for TRDY to assert. The other method uses a Sample instead, where a timeout can be specified. Both methods are explained below. Before doing either method though, you need to draw the expected TRDY waveform shown below.

Note: TRDY's waveform is blue because it is an input, so the data shown is predicted data, not data to be driven. The direction of TRDY was automatically determined by the tool during the Extract Ports from MUT step.

5.1. Draw expected TRDY waveform

	0ns	50ns	100ns	150ns	200ns	250ns	300ns	350ns	400ns
CLK									
FRAME									
WRITE									
IRDY									
TRDY									
ADDR[7:0]				00					
DATA[7:0]						AA			

5.2. Wait method 1: Wait indefinitely using sensitive edges

Note: If you want to specify a timeout for this wait, skip this step and go to 5.3.

Double-click on TRDY to open the Signal Properties dialog, enable the "Falling Edge Sensitive" check box, and hit OK. When this is enabled, the test bench will wait on every drawn falling edge on TRDY. This is indicated graphically by an arrow on the falling edge. Make sure that the falling edge of TRDY is drawn after the falling edge of IRDY, otherwise the test bench will wait for TRDY to assert before asserting IRDY.

5.3. Wait method 2: Wait with a timeout using sample

Note: Skip this step if you performed step 5.2.

Depress the "Sample" button. To create a sample, left-click on the rising edge of CLK at 300 ns, then right-click on TRDY at 300ns. Double-click on the new sample's name to open the Sample Properties dialog. Change the name to "WaitForTRDY" then click on the "HDL Code" button to open the Code Generation Options dialog. Here is where you can control the behavior of the Sample once it is triggered to run. Make the following changes:

– Disable the "Full Expect" check box.
– Specify 100 for the Multiplier.
– Enable the "Blocking" check box.

These three options work together to achieve the "wait with timeout" behavior we want. With "Full Expect" off and the "Multiplier" set to 100, this Sample will wait for up to 100 clock cycles for TRDY to assert. The "Blocking" check box causes the rest of the transaction to wait on the Sample to finish. Otherwise, the Sample would be run in parallel with the stimulus. More details on these options can be found in the Reactive Export Help. Here's what the diagram should look like at this point:
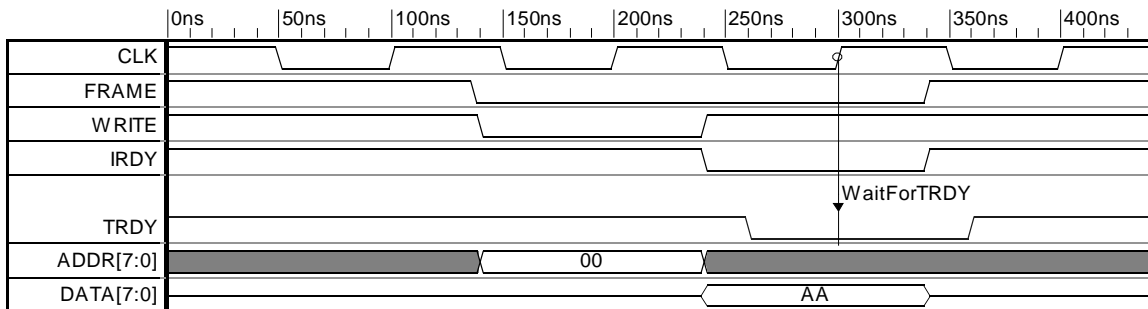
6. Draw single read

6.1. Draw the waveforms

Draw a complete read transaction following the write transaction. Here is what the waveforms should look like (assuming you used the edge sensitive wait; the sample version will look slightly different, of course):
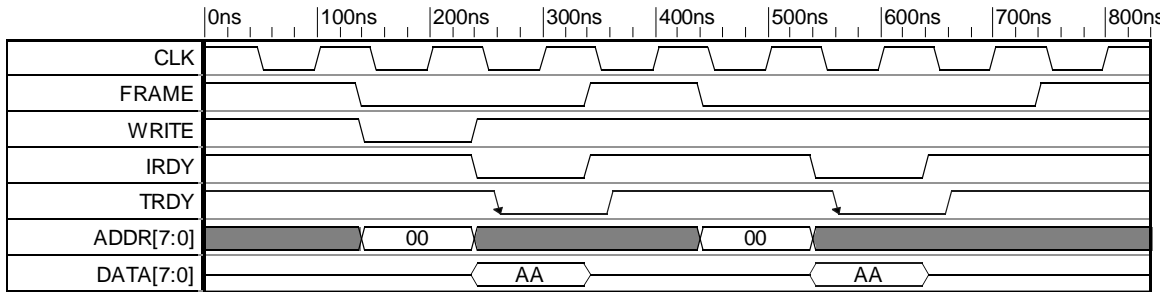
## 6.2. Disable "Drive" for the DATA segment

The test bench must not drive the DATA bus during the read cycle to avoid contention as the MUT will be driving it then. Since the DATA bus is a bi-directional signal, you can specify which parts of the waveform are driven by the test bench and which are not. One way to do this is to draw the bus with the TRI state, but in this case we need to specify the expected data on the bus, so the TRI state can't be used. Instead, double-click on the waveform segment of DATA that happens during the read. Disable the "Driven" check box and hit OK. The segment will be drawn in blue now, indicating that the DATA signal will *not* be driven by the test bench during this time period (i.e. just like the entire TRDY signal).

7. Add a Sample to verify data read from MUT

Depress the Sample button, then left-click on the positive clock edge at 600 ns and right-click on the DATA segment directly below it. This will place a Sample that will trigger at that clock edge and verify that the data read from the MUT is what we expect (indicated by the waveform drawn under the Sample). This is the default behavior of the Sample. Next, make the following changes to the Sample:

- Double-click on the Sample name and change its name to "VerifyDataRead"
- Click the "HDL Code" button to open the "Code Generation Options" dialog.
- Select "Display Message" for the "Then Action". Select "Note" for the severity level of this action. This will make the Sample display a note during simulation when it succeeds.
- Hit OK to close these dialogs.

Here is what the diagram should look like after adding the Sample:
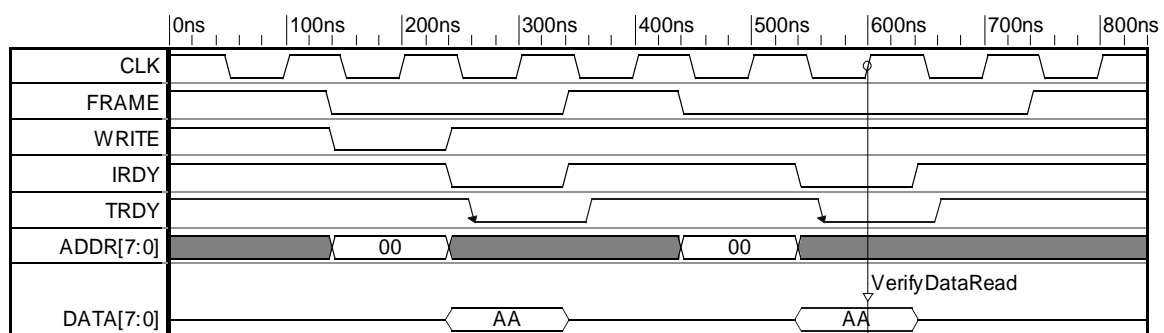
8. Drive data using a "Test Vector Spreadsheet" file

This step will use an input file to drive the DATA bus during the write cycle. This will increase the effectiveness of the test bench by writing different patterns to different addresses. The basic idea is to create a user-defined array variable that is initialized from a file. Here are the steps to create the variable.

- Click the "View Variables" button in the diagram to open the "Variable List" dialog.
- Click the "New Variable" button, then click on the name and change it to "inputData". This name is important because it must match a column name in the input file that we choose.
- Under "Structure" select "array."
- Set "Size" to 256.
- Set "Data Type" to "2_state" then change MSB to 7.
- Near the bottom of the dialog, enable "Initialize Structure With File". Browse to the "inputData" directory and select "inputData.txt". Hit OK.

Now that the variable is created, the next step is to refer to this array to drive and verify data. So, both of the "AA" states need to be changed. For the two "AA" states, do the following:

- Double-click on the state to open the "Edit Bus State" dialog.
- Type "@inputData[address]" and hit OK. The '@' symbol is used to refer to a variable defined in the "Variable List" dialog.
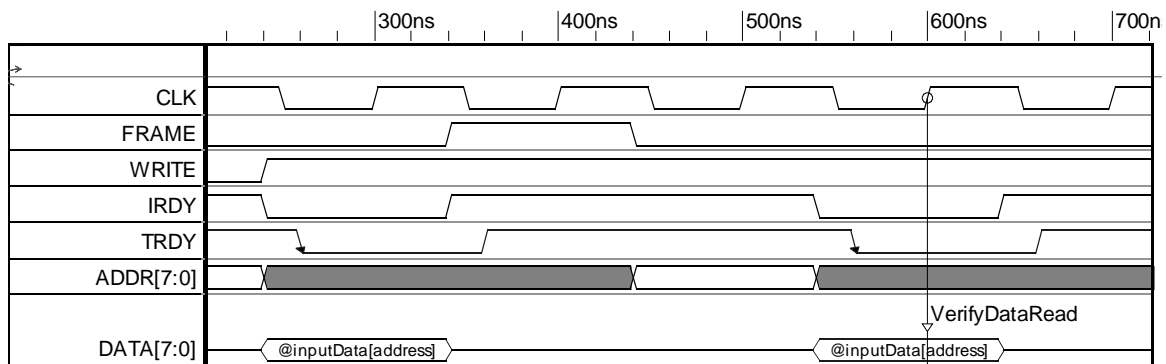

# 9. Create for-loop to perform multiple writes and reads

This step sets up the diagram to perform multiple writes and reads. Note: If you are creating a TestBencher transactor then the next step should be performed, TBP Transactor – Add address argument. Perform this step if you are unsure as it is also valid for TestBencher transactors.

- Depress the "Marker" button, left-click the positive clock edge at 100 ns, then right-click to place the Marker.

- Place another marker at the positive clock edge at 800 ns.
- Double-click the first Marker to open the "Edit Time Marker" dialog.
- Select "For Loop" in the "Type" drop down list.
- Set "Name" to "AddressLoop".
- Set "Index" to "address".
- Set "End" to "10".
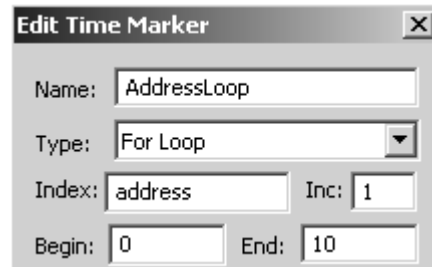
- Hit OK to close the dialog.
- Double-click the second Marker to open the "Edit Time Marker" dialog.
- Change it's "Type" to "Loop End" and hit OK.
- The two markers should now be connected graphically as shown below.
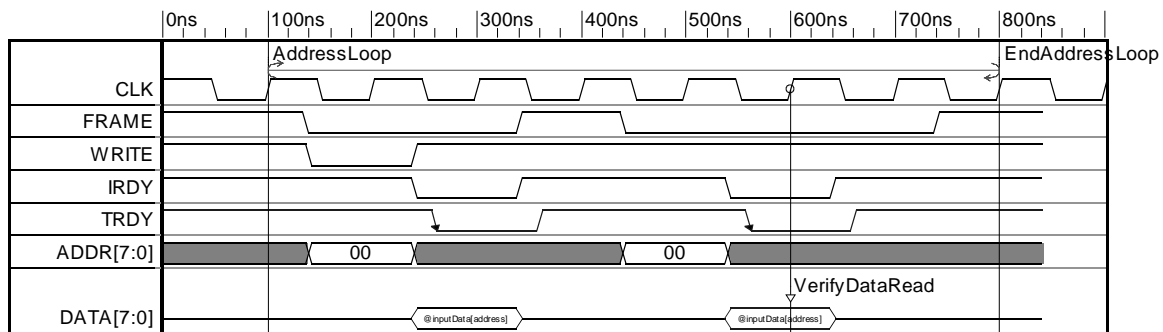
10. TBP Transactor – Add address argument

This step is optional and should only be performed if you are creating a TestBencher transactor. In this case, the for-loop can be omitted from the diagram and an argument can be set up for the address (i.e. the address can be passed in via the diagram apply call). It's not invalid to create a for-loop as performed in the previous step, but avoiding the for-loop gives the transactor greater flexibility.

Note: The primary purpose of this tutorial is to demonstrate various features available to all "reactive_tb" users. So, there are several steps that may not make as much sense for TestBencher users. For instance, two transactors could have been created instead of one: one for the write cycle and one for the read cycle. Also, the data could have been passed in as an argument to the diagram apply call (a function call that causes the transactor to perform a transaction with a given set of transaction arguments).
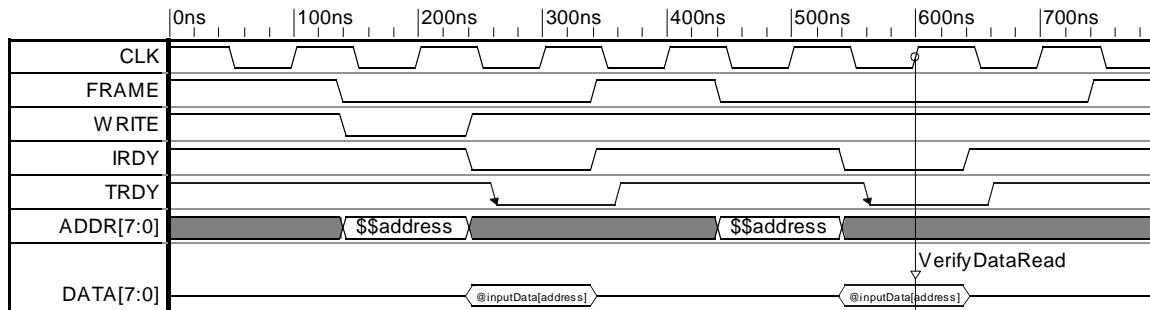
To add an "address" argument, do the following for the two address states (which

currently are set to 00):

– Double-click on the state to open the "Edit Bus State" dialog.
– Enter "$$address" for the state value and hit OK.

Here's what the final transactor should look like:

Now when an apply call in inserted for this transactor in the sequencer process, you will be able to specify which address to use.

11. Alternatives

11.1. Consecutive writes followed by consecutive reads

If you wanted to perform multiple writes concurrently, followed by multiple concurrent reads, then two for-loops are needed. The array of data can be referenced in each loop in the same manner already demonstrated.

11.2. Random data

In Verilog, you could "$random()" as the state value for DATA during the write transaction. A user-defined function can also be embedded into the generated test bench using the "Class Methods" dialog which could be used to generate data values. In both of these cases, you would need to modify the state value under the "VerifyDataRead" sample since the inputData array is no longer used. A Sample must be placed on the driven DATA segment to capture the expected data. For example, you could create a Sample named "ExpectedData" that is triggered from the clock edge at 300 ns. Then the state under the "VerifyDataRead" Sample would be set to "ExpectedData" instead of "@inputData[address]".