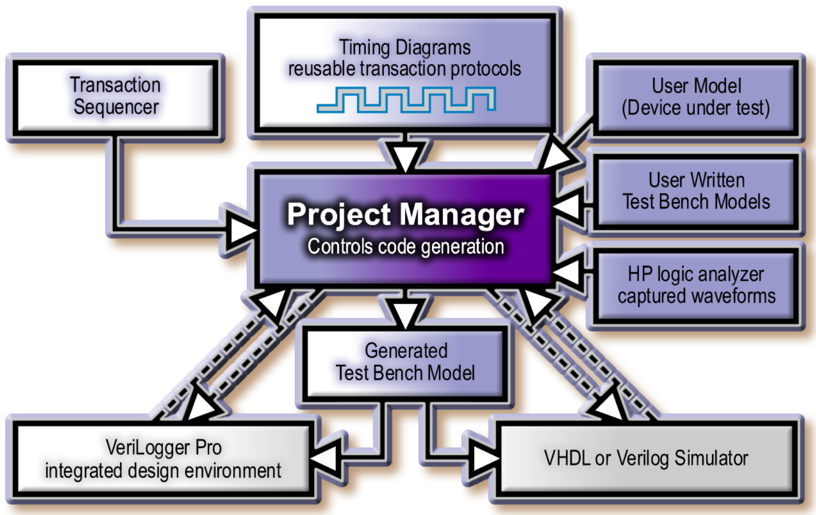
TestBencher Pro

Graphical Test Bench Generation for VHDL and Verilog

TestBencher Pro is a VHDL and Verilog test bench generator that dramatically reduces the time required to create and maintain test benches. One of the most time consuming tasks for users of HDL languages is coding test benches to verify the operation of their designs. In his book "Writing Testbenches," Janick Bergeron estimates that 70% of design time is spent verifying HDL code models and that the test bench makes up 80% of the total HDL code generated during product development.

TestBencher Pro Design flow


Transaction Sequencer
Timing Diagrams reusable transaction protocols
User Model (Device under test)
Project Manager
Controls code generation
User Written Test Bench Models
HP logic analyzer captured waveforms
Generated Test Bench Model
VeriLogger Pro integrated design environment
VHDL or Verilog Simulator

TestBencher Pro automates the most tedious aspects of test bench development, allowing you to focus on the design and operation of the test bench. This is accomplished by representing each bus transaction graphically and then automatically generating the code for each transaction. TestBencher makes use of the powerful features of the language that is being generated and the engineer does not have to hand-code each transaction. When hand coding, the designer would have to take the time to deal with the specifics of the design (port information, monitoring system response, etc) as well as common programming errors (race conditions, minor logic errors, and code design problems). This removes a considerable amount of time from the test bench design process because TestBencher manages the low-level details and automatically generates a valid test bench.

Communicates System-Level Design Concepts

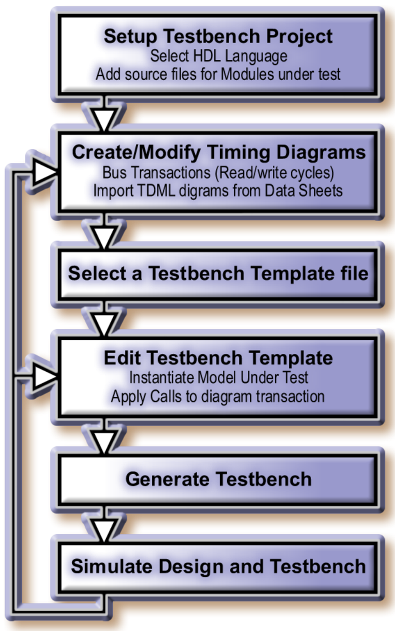
Translating design specifications into working systems and hand coded test benches is a difficult and error prone process. With TestBencher the translation is a natural conversion since the test bench is created directly from the design specification timing diagrams. System-level design engineers can create timing diagrams that describe a system's interfaces and hand them directly to the verification engineers who will use TestBencher Pro to generate bus-functional models from the diagrams.

"I have recently purchased the TestBencher (Pro) product… I have been able to generate Verilog code models for various peripherals connected to an ASIC under test in record time without writing much Verilog code. I plan on using TestBencher for generating comprehensive test benches that I had previously generated manually and were prone to many errors. I can generate the various waveform files for the various models connected to the chip-under test with self-checking capability (samples) using the integrated waveform drawing tool and TestBencher generates the Verilog test bench. TestBencher is the most complete and comprehensive Verilog design tool that I have ever used."

- George Lysy, Hewlett-Packard Company

Generates Test Benches from Timing Diagrams

TestBencher generates reactive VHDL and Verilog test benches and bus-functional models from language-independent timing diagrams. The generated test benches are capable of applying different stimulus vectors depending on simulation response so that the test bench functions as a behavioral model of the environment in which the system being tested will operate. The generated transaction can be either cycle-based or time-based. TestBencher also provides features to quickly change from time-based to cycle-based in a single step.

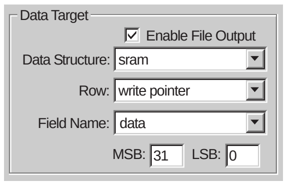

Setup Testbench Project
Select HDL Language
Add source files for Modules under test
Create/Modify Timing Diagrams
Bus Transactions (Read/write cycles)
Import TDML digrams from Data Sheets
Select a Testbench Template file
Edit Testbench Template
Instantiate Model Under Test
Apply Calls to diagram transaction
Generate Testbench
Simulate Design and Testbench

Transaction Based Modeling

Using the built-in timing diagram editor you can quickly model common bus activities such as driving control signals, sampling and verifying values from the model under test (reporting unexpected values or transitions), waiting for handshaking signals before completing a transaction, and aborting a transaction under a reset condition. TestBencher timing diagrams can be used to model synchronously clocked (cycle-based) and asynchronous transactions. Some examples of a bus-transaction would be a PCI read or write cycle.

Data Structures and Memory Interfaces

TestBencher Pro provides the capability of easily creating complex data structures that can be used to supply or store state information. These data structures can represent memories, for example, or they can be used to read from or write to a file using a spreadsheet format. By using the memory features, data can be stored from one transaction and then used by another. Once a data structure has been defined for a project, multiple instances of that data structure can be created. For file-based representations, TestBencher takes care of all the handshaking and wraps the transaction code around the generated file I/O code.


Data Target

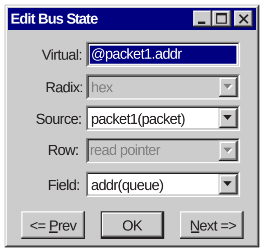
☑ Enable File Output

Data Structure: sram

Row: write pointer

Field Name: data

MSB: 31 LSB: 0


Edit Bus State

Virtual: @packet1.addr

Radix: hex

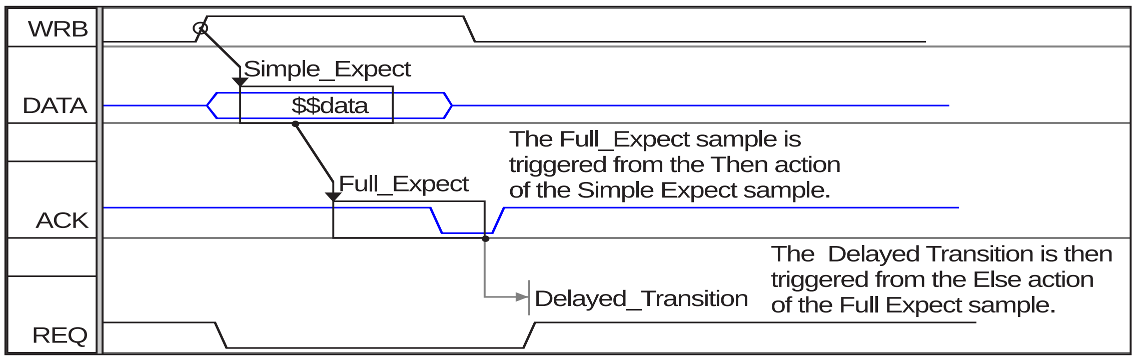
Source: packet1(packet)

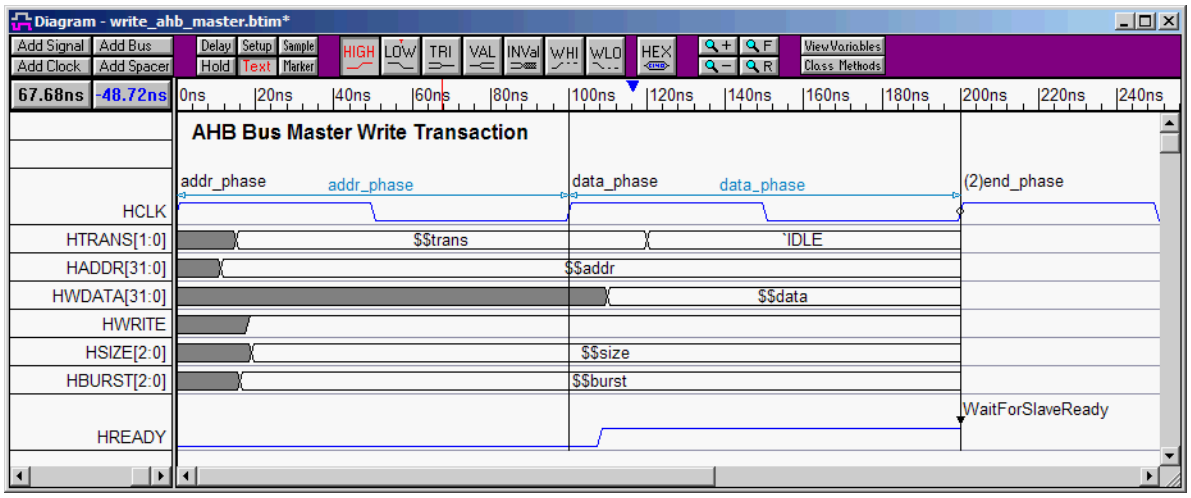
Row: read pointer

Field: addr(queue)

Response Checking with Samples

Sample parameters generate self-testing code that is used to store and verify data produced by the system being tested. A sample can be defined to trigger in a transaction at a specified time or when a edge transition takes place. When a sample triggers, it collects data about a signal's state at that specific time, or over a window of time (for example to check for stability or for an edge transition). The value stored by the sample can be exported to the top-level module, output to a file, or stored in a data structure or memory interface for further use later.


WRB
DATA
ACK
REQ
Simple_Expect
$$data
Full_Expect
The Full_Expect sample is triggered from the Then action of the Simple Expect sample.
Delayed_Transition
The Delayed Transition is then triggered from the Else action of the Full Expect sample.



▲ *This diagram models a write transaction for an ARM master device. The pipeline nature of the bus is represented using Pipeline Boundary Markers, where the user simply indicates the boundaries of each pipeline phase.*

Pipelining Transactions

Modeling pipelined transactions is simple in TestBencher. The user is only required to indicate where pipeline phases begin and end, using Pipeline Boundary Markers. When the transaction is applied it will automatically be pipelined based on these Markers. Each Pipeline phase is given a name and the namces can be shared between multiple diagrams. This allows you to model different transaction types that are to be executed in the same pipeline.

Sequence Recognition

Sequence Recognition provides the ability to create a state machine within a timing transaction. This state machine can be used to ensure that a specific sequence of events occurs on one or more signals prior to the transaction proceeding. Sequence Recognition allows test benches to be designed using a dataflow (event-driven) architecture instead of a sequential architecture.

Parameterize Both State and Timing Information

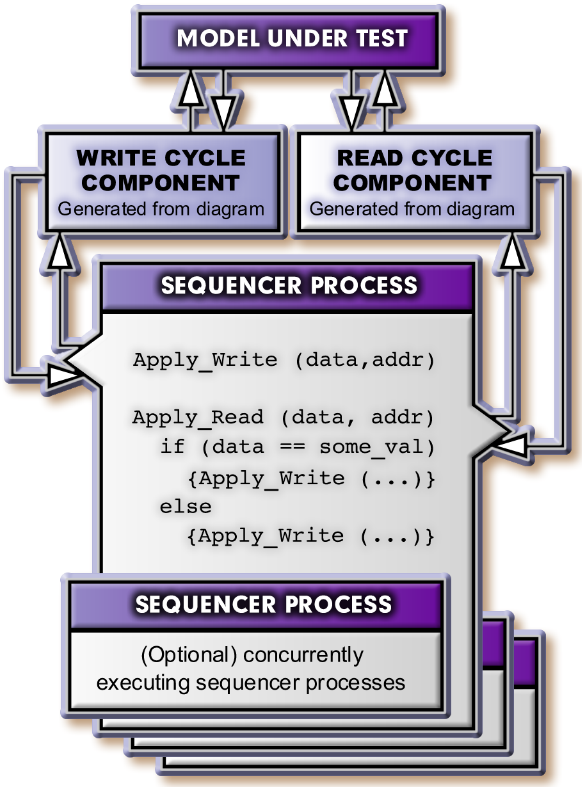
To create reusable timing transactions, TestBencher allows both state and timing variables to be parameterized so that new values can be passed into the transaction each time it is called. State and timing variables can be passed into a transaction through a transaction call or read in from a file using Data Structures. The Data Structure editor makes it easy to transparently change the data sources and targets for a testbench's transactions as needed during testbench development.

Markers used for Control and Looping

Markers can be added to timing diagrams to specify actions to be taken by the transaction during execution. These actions can include signifying the end of a transaction, creating loops in the transaction, waiting for a condition (or group of conditions) to become true, and inserting a call to an HDL subroutine.

Control Execution of Sequential and Concurrent Transactions

A top-level template file provides the top module of the test bench. Execution mode selections determine whether a transaction executes once or if it loops continuously, and whether the transaction executes sequentially or concurrently. The top-level module contains a Sequencer Process where the transaction calls are placed. Native HDL code can also be placed in the Sequencer Process. This enables you to place conditions on whether or not a timing transaction is called, or on the parameter values that you wish to have applied, for example. The figure demonstrates how this feature can be used with read and write transactions. In this case, two bus transaction timing diagrams were created: a write cycle and a read cycle. TestBencher automatically generated a read transactor and a write transactor model from these diagrams and functions that can be used to start these transactions (Apply_Write and Apply_Read). The functions in this case have two parameters: the address and data values to be used during each transaction. The sequencer process in this example contains several function calls added by the user to execute the transactions sequentially. TestBencher provides an "Insert Diagram Call" palette that makes it easy to auto-insert templated function calls from the list of available functions (constructed from the list of timing diagrams defined in the TestBencher Project).



```
Apply_Write (data,addr)

Apply_Read (data, addr)
  if (data == some_val)
    {Apply_Write (...)}
  else
    {Apply_Write (...)}
```

Native Language Code Generation

TestBencher Pro generates native VHDL or Verilog test bench code. This allows the generated code to be compiled with your design files and simulated using all major VHDL and Verilog simulators. Debugging the resulting system is easy since the test bench is structured into transactions and all of the generated code uses the same language.

TestBencher Pro for SystemC code generation is also available now!

VeriLogger Pro

TestBencher Pro

DataSheet Pro

WaveFormer Pro

TimingDiagrammer Pro

SYNAPTICAD

Tools for the Thinking Mind

Site:
www.syncad.com

E-mail:
sales@syncad.com

Direct:
540 953 3390

Sales:
800 804 7073

Fax:
540 953 3078

520 Prices Fork Rd. Suite C4
Blacksburg, Virginia 24060