Using Tcl/Tk in ModelSim to create custom displays

A worked example

Jonathan Bromley, Doulos Ltd

Doulos Ltd wishes to thank Mentor Graphics staff who have helped in the creation of this example. However, any remaining errors or omissions are entirely the responsibility of Doulos Ltd.

Contact Doulos for information about other HDL training and project support services. Visit the Doulos website for tips, code samples and tutorials.

Doulos Ltd.
Church Hatch
22 Market Place
Ringwood
Hants. BH24 1AW

Tel: +44 (0)1425 471223
Fax: +44 (0)1425 471573
Email: info@doulos.com

www.doulos.com

The value of custom displays in simulation

HDL simulations are used for many purposes. In early stages of the design cycle, an HDL simulator can be a powerful tool for checking that a hardware design block is functioning in the expected way, providing a useful "sanity check" that the designer has correctly interpreted the original specification.

Sometimes, however, it's quite difficult to relate simulation output to the real-world behaviour of the system. For example, a subsystem that generates RGB video output will provide the red, green and blue digitised video data, and one or more sync signals. Except in the very simplest cases, it's extremely hard to imagine the picture that these signals will represent. Visualisation of this kind of output is therefore a powerful tool for gaining insight into the device-under-test's behaviour, and can be a useful additional weapon in the verification engineer's armoury.

Stand-alone visualisers

It is, of course, possible to extract output data from simulation as a file which can then be visualised later using a stand-alone program, perhaps written in C. This is often the most flexible approach, but it denies you one of the key potential benefits of visualisation: the ability to relate visualised behaviour directly to signals in the device under test, as the simulation progresses or as a cursor is moved in the simulator's waveform display.

Integrated visualisers with Tcl/Tk

ModelSim has the attractive feature that it comes with a built-in Tcl/Tk system. Tcl, originally designed as a script language, is in fact a highly competent programming environment; and Tk provides a flexible and easy-to-use toolkit for graphical user interface development. The tight integration between Tcl/Tk and ModelSim makes it exceptionally straightforward to add new display functionality to your ModelSim simulations.

An example: Phase/amplitude plot

As an example of how to create customised displays for your ModelSim simulations, we look at the task of providing a polar plot of the output from a phase/amplitude demodulator.

Quadrature Amplitude Modulation (QAM)

In many telecomms applications a carrier signal is modulated using complex amplitude modulation in which two separate data streams are modulated on to the carrier and a second carrier 90° out-of-phase with the first. The two modulation components are commonly known as the in-phase and quadrature or I and Q components. The demodulator's output therefore consists of two separate signals I and Q, each of which is an analog value digitised to some appropriate number of bits. To understand how the digital data has been encoded on these signals, it's usual to represent them on a polar diagram (sometimes called an Argand diagram) in which the X-axis represents the I component, and the Y-axis the Q component. The demodulator output at any given moment is then represented by a single point on the polar diagram. If these points are accumulated over time, we get a "constellation plot" which should show the I/Q points grouped in discrete "clouds", each cloud representing one combined value of the transmitted data. Naturally, if the clouds overlap then there is some uncertainty in the decoding; this overlap can be caused by noise or other artefacts of the transmission medium, or by some inadequacy of the demodulator itself.

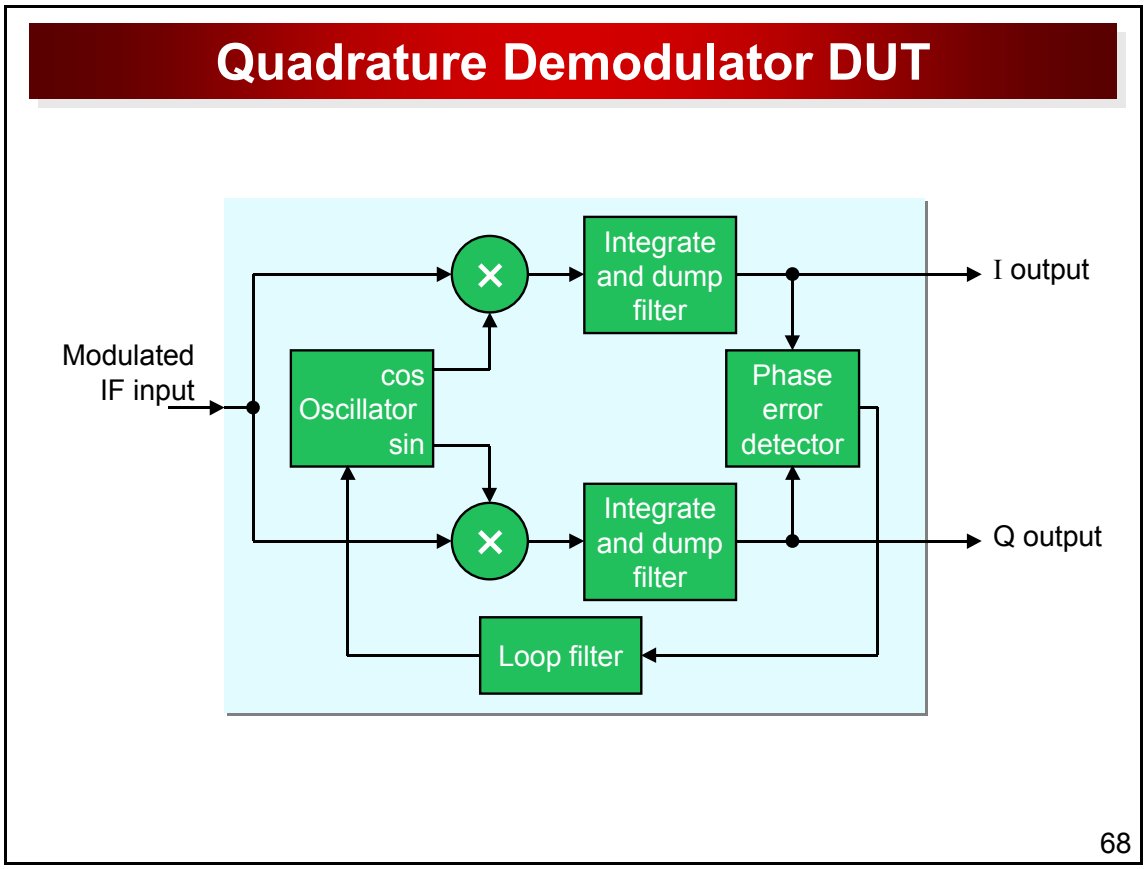
The constellation diagram gives a very clear and easily understood representation of the demodulated data stream, but it is almost impossible to visualise it given only the stream of numbers on the I and Q outputs.

We need to visualise the polar diagram!

To do this, we need to add some software – written in Tcl/Tk – to our ModelSim simulation. The following diagrams indicate how this could be done.

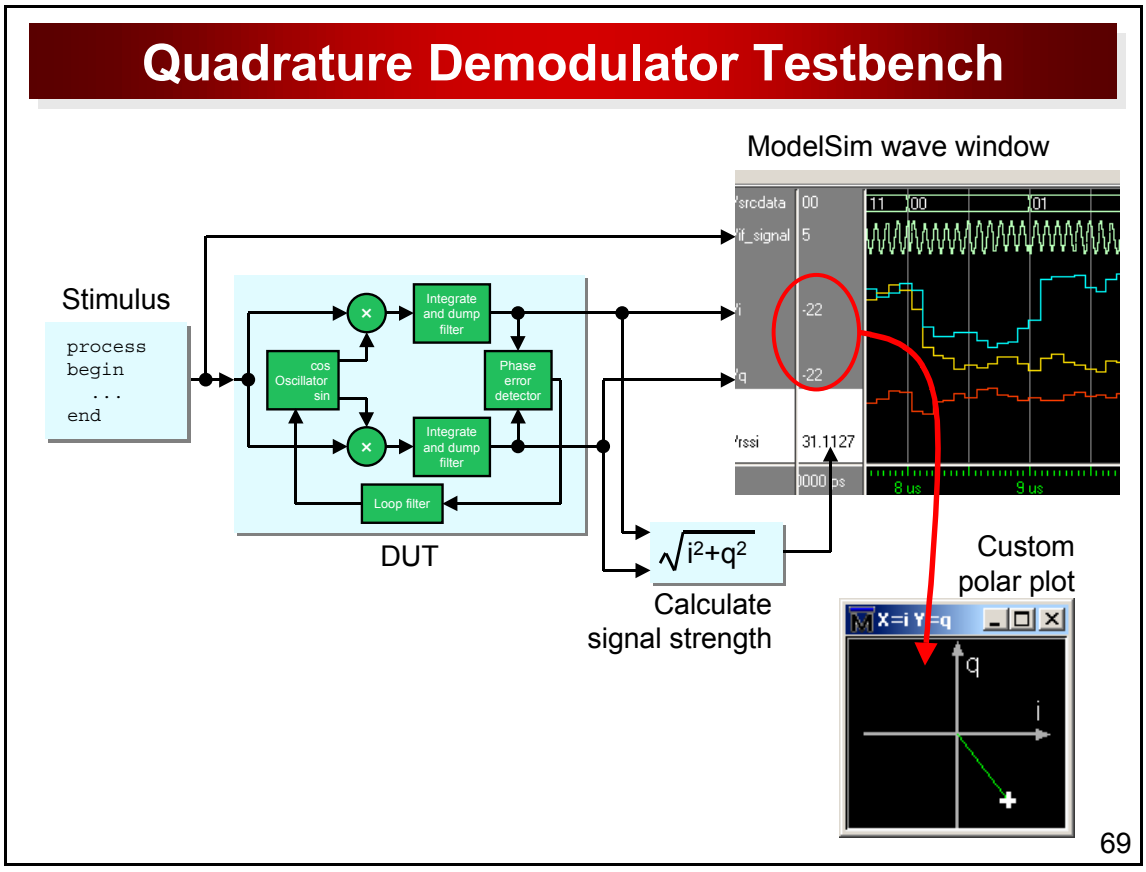
The Demodulator

The diagram below shhows the overall structure of a typical digital quadrature demodulator. The IF input is a digitised data stream, sampled at the demodulator's system clock rate. The I and Q outputs are also digital data streams. In our specimen device under test (DUT), I and Q are also sampled at the system clock rate. In a practical demodulator they may be sampled at a somewhat slower rate.



Testing the demodulator in simulation

A test bench for the demodulator needs to apply an appropriate modulated IF signal. Using VHDL's `ieee.math_real` library package it's easy to use `sin()` and `cos()` functions to generate the appropriate phase modulated signal as a `real` (floating point) value, and then "digitise" it using a type conversion to `integer`. The next diagram shows an overview of how a simple test bench might work, together with the polar diagram visualisation that we hope to achieve.

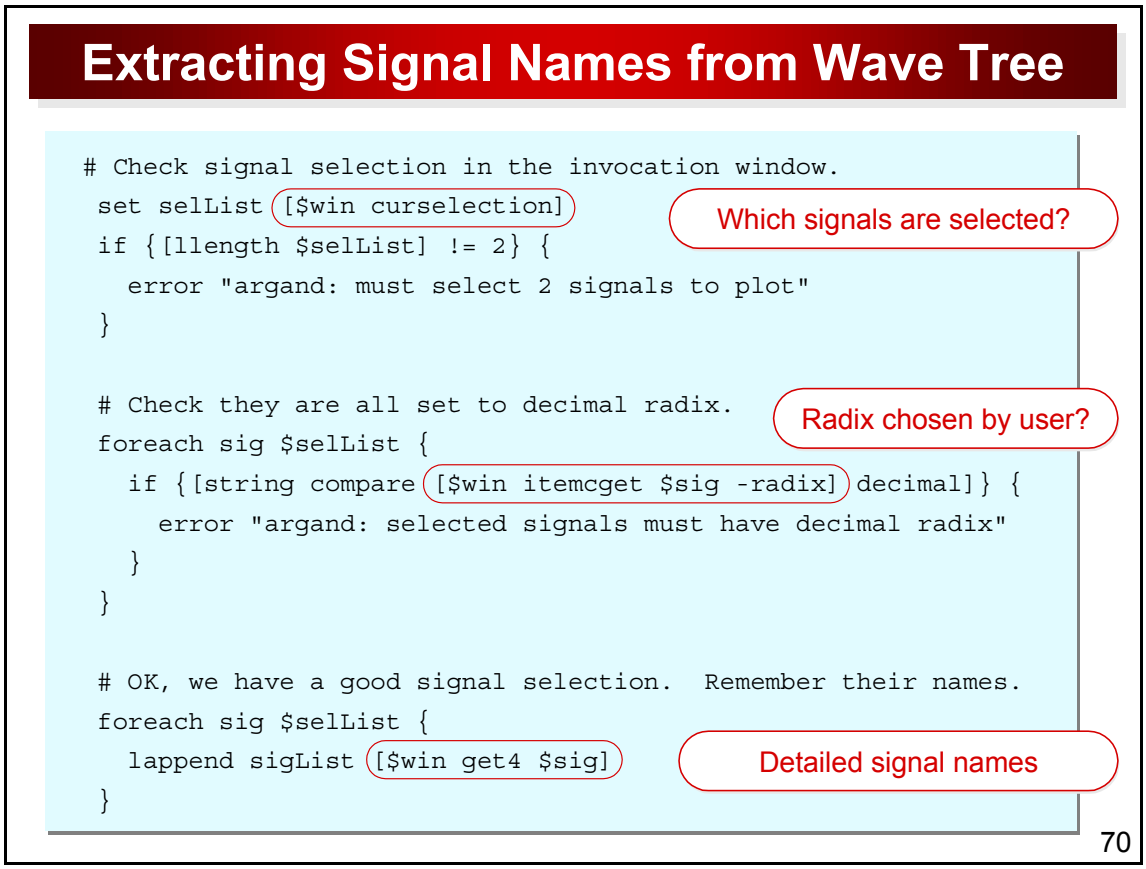


Key Challenges

Any GUI-based programming task like this requires careful attention to detail in order to give a good impression to the user, with all the required functionality being available in an appropriate way regardless of the user's sequence of actions. However, when we try to integrate custom displays into a ModelSim simulation there are several new challengest that are specific to the ModelSim environment.

Get signal names and values from ModelSim wave window

Once the user has set up a simulation, they are likely to use the ModelSim wave display window as a first "point of contact" when trying to visualise the DUT's behaviour. We need to link our new display to the wave window, picking the signal names and their required values from the wave display. To achieve this we need to make extensive access to ModelSim's *WaveTree* widget, which is the object used by ModelSim to set up the list of signal names and values that you see on the left of a wave window. There are numerous Tcl commands giving access to all aspects of this widget. The code fragments in the next diagram give an indication of the kind of operations you'll need to perform.



In all these examples, `win` is a Tcl variable containing the name of the WaveTree widget we're interrogating. Typically, for the default wave window `.wave`, this will be `.wave.tree` but in general it will be `<windowname>.tree` where `<windowname>` is the specific wave window in use.

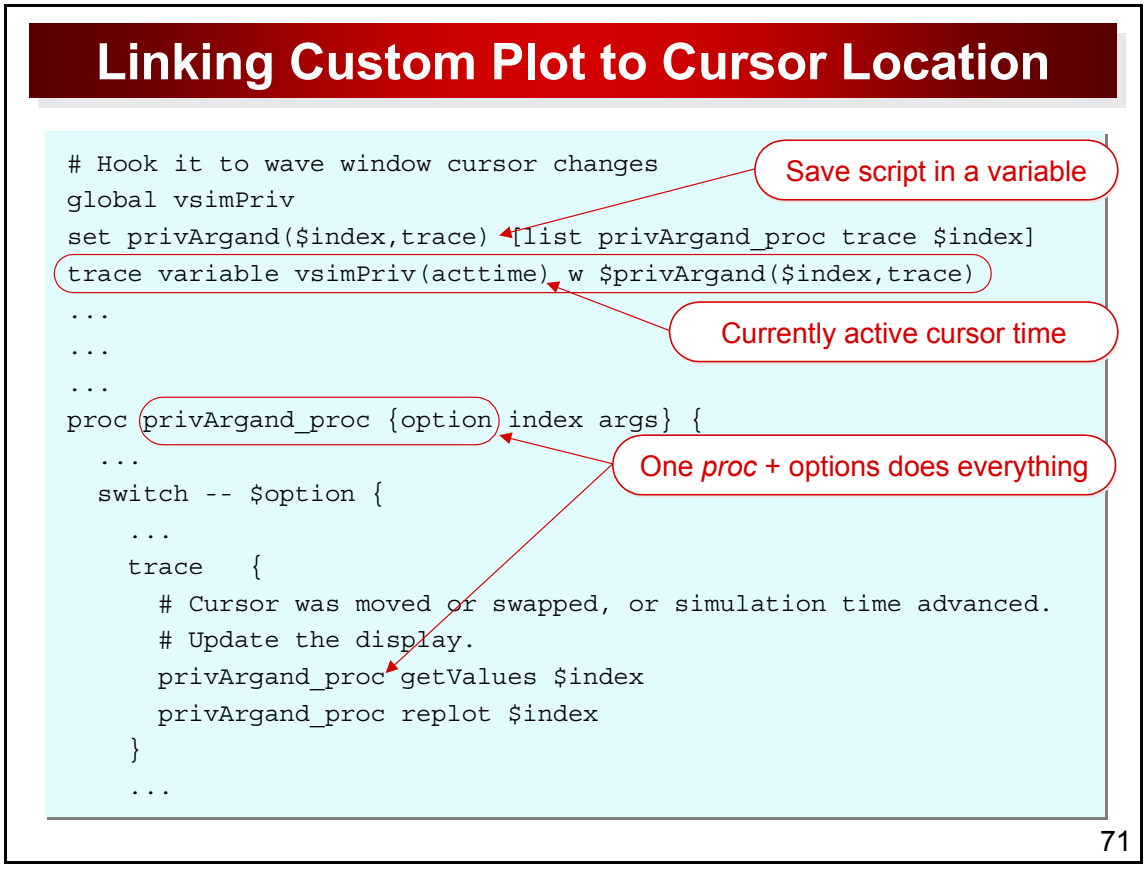
Note that, as with most Tk standard widgets, the name of a WaveTree widget acts as a command. Followed by one of several subcommands such as `itemcget`, it allows us to access many properties of the widget.

Note also that we've assumed that the user has already selected (highlighted) the two signals in the wave window that should be plotted on our diagram.

Dynamically link polar display to current wave cursor position

The most important use of the new polar plot is to allow the designer to find situations where the DUT is giving the wrong results (as visualised on the polar plot), and then inspect other DUT signals at the same moment of simulation time, to understand what's gone wrong. To do this we need to arrange for the polar plot to be updated dynamically to reflect the chosen signal values at the current wave cursor position. Luckily we can "see inside" ModelSim to find the current cursor location. ModelSim maintains a great deal of information about its internal state in a Tcl array variable `vsimPriv()`, and although much of this data is of no concern to us, it's useful to know that `vsimPriv(acttime)` contains the time position of the currently active wave window cursor. Better still, we can use Tcl's `trace` facility to monitor any updates to this variable – for whatever

reason – and whenever it's updated, we can re-plot our custom display to keep pace with it. Check out online help for the Tcl `trace` command for more details.



This code fragment also illustrates another idea that's very useful when creating add-ons for ModelSim. The ModelSim environment creates and uses a very large number of new variables, and it would be painfully easy for our code to re-define or re-use those variable names. To minimise this risk, we use a Tcl array variable with the suitably unlikely name `privArgand()` to store all data related to our custom plot. Similarly, we avoid defining too many new Tcl `proc`s by creating just one `proc` to do all the work, and giving it a range of subcommands so that it can provide a wide range of functionality.

Create polar plot using new menu item in wave window

So that our users have easy access to the new functionality, we need to add a new menu item to each wave window. ModelSim makes this very easy: it provides utility commands `add_menu` and `add_menuitem` to simplify the task of inserting an item into ModelSim's rather complicated menu structures (don't try doing it using the traditional Tcl/Tk menu widgets – ModelSim's menus add an extra layer of complexity!). Better still, the `PrefWave(user_hook)` variable contains a user-specified Tcl list of commands that should be executed whenever any new Wave window is created. In the `modelsim.tcl` file, which is automatically read by ModelSim at startup, we can include commands to build this list. At the same time, it's useful to `source` the main script `argand.tcl` that provides most of the new plotting functionality.

Hooking at ModelSim Startup

modelsim.tcl

```
# Read the constellation plot code into Tcl
source argand.tcl

# Proc to add the appropriate menu item to any wave window
proc wave_hook {w} {
  add_menu $w plots
  add_menuitem $w plots Polar [list argand $w]
}

# Hook this stuff into every new ModelSim wave window
lappend PrefWave(user_hook) wave_hook
```

ModelSim tells us which window invoked the hook

73

Other Coding Techniques

Full Tcl code for this example is available from the Doulos web site at `www.doulos.com/tcl` together with a sample DUT and testbench written in VHDL.

- ***Please note***: The DUT is ***not*** intended to be a good example of how to build a demodulator! It has very poor performance, in part because we wanted to show how the polar plot can illustrate this behaviour.

The Tcl code has been very extensively commented to try to explain the techniques used. It includes examples of a range of useful tricks:

- using complicated strings to form array variable subscripts that are guaranteed to be unique
- using an index integer to construct array subscripts that are unique to each instance of the polar plot widget
- using Tcl's `catch` command to handle things that might possibly fail at run time
- using canvas tags to make it easy to keep track of items that have been drawn on a graphics canvas

We welcome your feedback by email to `info@doulos.com` and hope that this example will prompt you to experiment with other extensions to ModelSim.