Register Transfer Level design

1.0 Introduction

A design at the 'Register Transfer Level' consists of a number of interconnected blocks e.g. registers, counters, multiplexers, state machines. The behaviour of each of these blocks is described ***functionally*** but the internal ***structure***, which will eventually consist of interconnected gates, is not yet defined.

This document describes the use of a set of CAD tools supplied with Cadence Design Framework II (DF2) to implement a Register Transfer Level design. The design referred to in the document is the 'GameChip' design which forms part of the CS224 laboratory exercise. Supplementary documentation on the project is available (The CS224 Lab Manual, October 1994). It includes a fully documented design example 'ScoreBoard'.

The main steps of the Register Transfer Level design are as follows:

- Enter a schematic showing the interconnections of the blocks used in the design.
- Enter the ***functional*** description of each cell which is not in the library CS224lib using the Hardware Description Language (HDL) ***Verilog.***
- Simulate the design using the test language STL to input test patterns. The recommended procedure is to test parts of the circuit independently before testing the circuit as a whole.

    A test program is written in STL for each cell or part of the circuit which is to be tested separately. The simulations are run using the ***Verilog-XL simulator*** and are repeated, after making corrections, until the designs are correct.

    The whole circuit is simulated in the same way until correct.

When the Register Transfer Level design is complete the Gate Level design is carried out as described in the document 'CADEN-4'.

Before starting the design walk-through you should have first carried out the Register Transfer Level design on paper. Functional/behavioural models for most of the blocks you will need are provided in the library ***CS224lib***.

It is also assumed that the walkthrough exercises in CADEN-1 and CADEN-2 have been completed.

If you wish to work through some of these stages described in this document using an existing design, you can use the design example '***ScoreBoard***' which may be found using your library browser.

2.0 Create a New Library

You will need to create a new library for your design.

2.1 Click on ***Open->Library*** on the CIW.

2.2 Enter a ***Library Name*** - it is most important that you use your group name here.

2.3 Click on ***OK*** on the Open Library form.

2.4 Accept the values in the next form presented.

Note: For a more detailed explanation look at the CADEN-1 document, section 7.

3.0 Copying Register Transfer Level blocks

Most of the Register Transfer Level blocks which you require for your design will be supplied in the Library ***CS224lib.*** This is a reference library with symbol, functional and schematic cellviews for each block. You will be able to reference the cells directly from the library in order to include them in your own design. However you will need to copy the ***Gameblock*** cell, and the ***MyGame*** cell into your own library because you will be creating your own version of GameBlock and MyGame.

3.1 Click on ***Design Manager->Library Browser*** on the CIW to open the Library Browser.

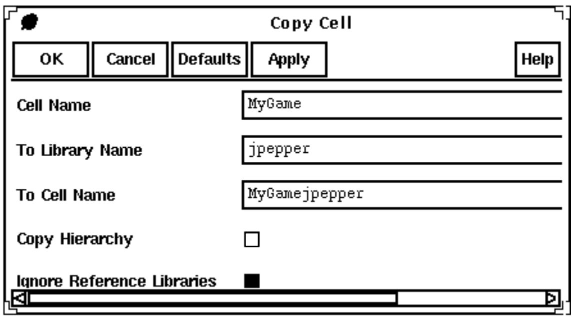
3.2 Click on the ***CS224lib*** library to show its contents.

3.3 Hold down the middle mouse button over the ***MyGame*** block you require, to reveal the menu.

3.4 Slide the mouse to the ***Copy*** option.

3.5 Release the mouse.

The Copy Cell form appears.



3.6 Fill in the ***Library Name*** with the library of ***your*** design. The form above has used the example jpepper-your Library Name will be different.

3.7 Type in the ***Cell Name*** with your group name appended to it.

e.g. ***MyGame<groupname>***

3.8 Make sure ***Copy Hierarchy*** is turned off.

3.9 Click on ***OK***.

3.10 Repeat the exercise for the ***Gameblock***, remembering to use the cell name ***Gameblock<groupname>.***

4.0 Editing the Gameblock schematic cellview.

4.1 Click on ***Open->Design*** on the CIW.

4.2 Fill in the form with

Library Name Enter the name of the library you created in section 2

Cell Name Gameblock<groupname>

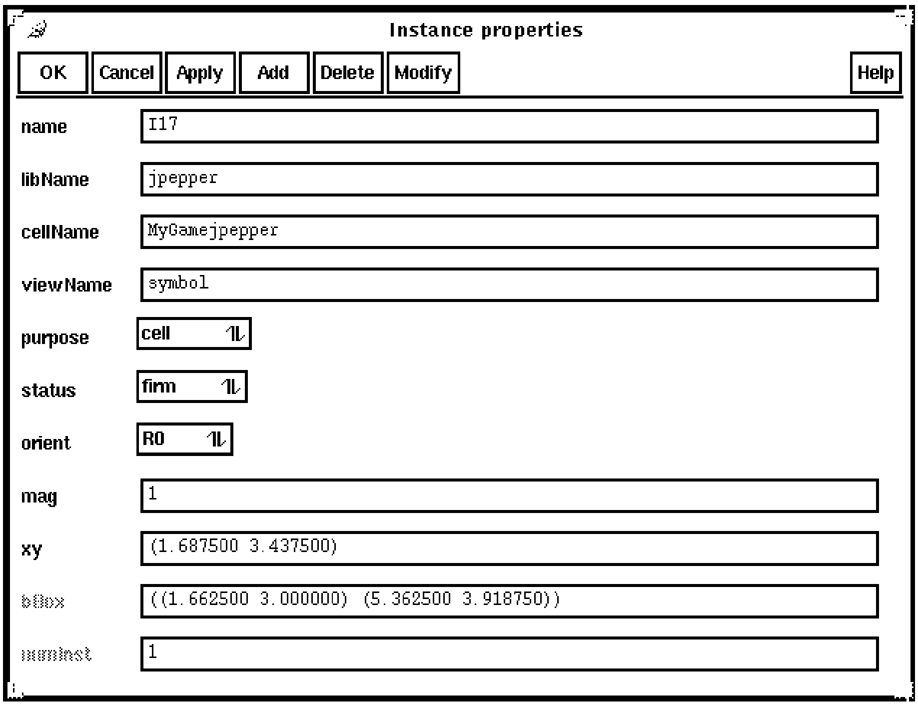
View Name schematic

4.3 Click on ***OK***.

The Gameblock schematic window appears; it will have in it the symbol for MyGame from the CS224lib library. This needs changing to the MyGame<groupname> symbol in your own library.

4.4 Select the MyGame symbol.

4.5 Click on ***Edit->properties*** on the schematic command banner.The following



4.6 Change ***libName*** from ***CS224lib*** to your own library name.

4.7 Change ***cellName*** from ***MyGame*** to ***MyGame<groupname>***.

4.8 Click ***OK.*** on the form.

4.9 Click ***Design->Check and Save*** on the schematic command banner.

4.10 Click ***Window->Close*** on the schematic command banner.

5.0 Creating the MyGame schematic cellview.

5.1 Click on ***Open->Design*** on the CIW.

5.2 Fill in the form with

Library Name Enter the name of the library you created in section 2

Cell Name Gameblock<groupname>

View Name schematic

5.3 Click on ***OK***.

The Gameblock schematic window appears; it will have in it the symbol for MyGame within which you will need to create a schematic.

5.4 Click on ***Create->Cellview->From Instance*** on the schematic command banner.

5.5 Check that the view name on the Cellview From Instance form is set to ***schematic***.

5.6 Click on the ***MyGame<groupname>*** instance in the Gameblock design to generate the schematic cellview.

5.7 Click on ***Window->Close***.

5.8 Click on ***Open->Design*** in the CIW

Fill in the form with:

Library Name Enter the name of the library you created in section 2

Cell Name MyGame<groupname>

View Name schematic

The MyGame<groupname> schematic window now appears ready for your design.

6.0 Entering your Design

The symbols for most of the blocks will be available in the CS224lib Library. However you will need to create the symbols of the FSM and the Multiplexer.

6.1 Follow the steps in the ***CADEN-2*** document section ***3*** to create the symbols for the FSM and Multiplexer.Use cellnames ***FSM<groupname>*** and ***Mult<groupname>***.

6.2 Place all the Instances you need on the schematic.

6.3 Add wires to the schematic so that the Blocks are connected and the input and output pins are connected.

6.4 Label any buses in the schematic.

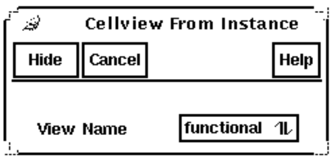
6.5 Check and Save your design.

7.0 Functional Descriptions

All of the cells in the library CS224lib have a functional view which describes the behaviour of the block. However the FSM and Multiplexer do not have any functional view, so they cannot be simulated. You will first need to create a cellview functional for both blocks and then Descend Edit to add the text to the File.

7.1 Click on ***Create->Cellview->From Instance*** on the schematic command banner.

7.2 Change the View Name on the form to ***functional***.



7.3 Click on your FSM instance to create the functional cellview.

7.4 Press ***Esc*** to cancel the command.

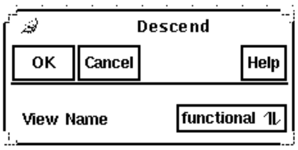
There is now a ***functional*** cellview for the instance. You need to edit the ***functional*** to add in the text to describe its behaviour.

7.5 Move the mouse onto the FSM instance.

7.6 Hold down the middle button and choose the ***Descend Edit*** option.

7.7 Click on the FSM instance.

7.8 Change the View Name to ***functional***.



The Textedit window now appears ready for you to enter the text. Once you have finished typing in the text save the file and close the window. The software will now compile your file.

Document CADEN-6 gives guidelines on Verilog HDL and a summary of the Verilog syntax.

If there are any errors in the file a dialog box appears:

7.9 Click on ***OK*** to view the error file and re-edit HDL.

7.10 Repeat this process until your HDL is correct.

7.11 Now repeat the whole process for the Multiplexer block.

8.0 Partition the Circuit to Test Individual Blocks

Before testing the whole circuit, it is sensible to partition it and test the parts independently. A schematic view must be created for each part that you wish to test. The following procedures are suggested:

For individual cells, such as the FSM and Multiplexer cells:

8.1 Open a schematic for the cell e.g. FSM_schem, as described in 5.8 above.

8.2 Place the symbol for the cell in the schematic.

8.3 Add input and output pins and connect them to the symbol.

8.4 Check and save the schematic.

For sections consisting of a number of interconnected cells, e.g a set of interconnected counters:

8.5 Copy the whole Register Transfer Level design into a new schematic e.g. Counter_schem.

8.6 Delete all the parts of the circuit not wanted.

8.7 Add suitable input and output pins and connect them to the circuit.

8.8 Check and save the schematic.

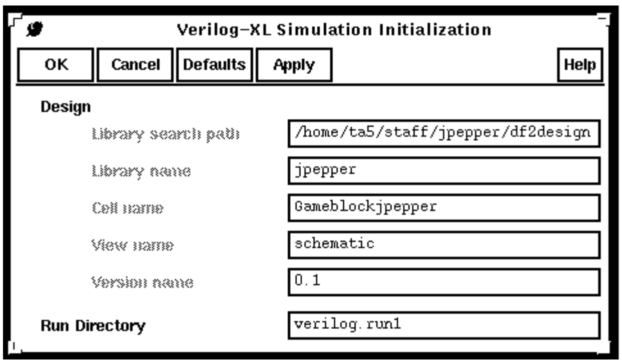
Each schematic can now be simulated following the procedures described below.

When each part has been checked, the whole circuit as entered in section 5, can be simulated.

9.0 Start the Simulation

9.1 Click on ***Tools->Simulation->Verilog-XL***.

The ***Verilog-XL Simulation Initialization*** form appears. This form asks you to specify the ***Run Directory***.



9.2 Enter a Run Directory. Change the default Run directory name so that you use a different name for each cell or block simulated e.g. verilog.fsm1. (Using the same name repeatedly can cause problems and confusion).

9.3 Click on ***OK***.

When you enter the Verilog-XL environment, the ***Simulate***, ***Stimulus***, and ***Session*** menu names appear in the menu banner of the Schematic Editing window.

10.0 Netlisting

First you will need to Netlist the schematic. A netlist describes the connectivity of the design, so that during simulation the simulator knows how the components are connected.

To netlist your design follow the steps below:

10.1 Click on ***Simulate->Netlist*** to netlist your design.

The results of the netlist will be shown on the CIW. If any errors occur then no netlist is produced, the errors will be shown on the CIW.

10.2 Correct any errors until a netlist is produced.

11.0 Creating an STL File

To test the design you need to create a Test Stimulus. We use the Simulation and Test Language (STL).

11.1 Click on ***Stimulus->STL->Edit Stimulus*** to create or edit the input stimulus file (***input.stl***) in your current run directory.

The Edit STL Stimulus File form appears. Click on ***Compile STL after editing*** if you want to compile the input.stl file

If the input.stl file does not exist then a template is created for you. The Textedit window now appears with the input.stl file ready for you to edit.

11.2 Once you have finished entering the text ***Save*** the file.

11.3 ***Close*** the Textedit window by choosing the ***Quit*** command.

The input.stl file will now be compiled. Any errors that occur will be shown in the CIW. Correct any errors until the file is compiled successfully.

12.0 Simulation and Test Language (STL)

STL is a high-level language for the generation of simulation test vectors. It is based on the Cadence-developed IL programming language. It enables concise test vector definitions, and the ability to use high-level programming language features such as variables, iteration, conditional statements and procedures.

This section gives a brief introduction to the use of STL, via a worked example for the ScoreBoard. An explanation of the STL instructions used in the file is given; it must be stressed that these are a small subset of STL. For more details, consult the STL Command Reference Manual in the laboratory.

```
; STL Source Program Template

stlinit

; note that clock inputs should be of type “clk”
defpin Clkclk
defpin GameEnablein
defpin GameResetin
defpin InpBus<3:0>in
defpin OutData<8:0>out
defpin OutSel<4:0>out
defpin Strobeout

deftiming 1e-11 1e-07 1e-06 ; define basic time units
; define any clocks or timing (defclock, defstrobe)

defclock “1111.....1” Clk
defstrobe out “.......%..”
defformat InpBus:bin GameReset:bin GameEnable:bin Strobe:bin
OutSel:bin OutData:bin
deftest

; define simulation stimulus values using xv command
; use binary truth table data, in the order of the
; defpin commands entered above. (eg- “xv 0 1 0”).
xv X X 0 Z Z Z
xv 0b0000 0 1 X X X
xv 0b0000 1 1 X X X
xv 0b0000 0 1 0 0b00000 0b000000000
xv 0b0000 0 1 0 0b00110 0b000000000
xv 0b0000 0 1 1 0b00110 0b000000000
xv 0b0000 0 1 0 0b00101 0b000000000
xv 0b0000 0 1 1 0b00101 0b000000000
xv 0b0000 0 1 0 0b01001 0b000000001
xv 0b0000 0 1 1 0b01001 0b000000001
```

```
xv 0b1000 0 1 1 0b01011 0b000000000
xv 0b1000 0 1 0 X X
..................................
..................................
..................................
xv 0b0000 0 1 0 0b00110 0b010010011
xv 0b0000 0 1 1 0b00110 0b010010011
xv 0b0000 0 1 0 0b00101 0b010011001
xv 0b0000 0 1 1 0b00101 0b010011001
xv 0b0000 0 1 0 0b01001 0b000000001
xv 0b0000 0 1 1 0b01001 0b000000001
xv 0b0000 0 1 0 0b01011 0b000000000
xv 0b0000 0 1 1 0b01011 0b000000000
xv 0b0000 0 1 0 X X
endtest
```

STL Instructions

- ***Comments*** are introduced by a semi-colon.
- ***Stlinit*** initialises STL.
- ***Defpin*** declares pin names, specifies whether they are of type input (***in***), output (***out***) or clock (***clk***), and assigns them (arbitrary) pin numbers.
- The syntax of the ***defformat*** command is:

  defformat <name>[:<radix>]...<name>[:<radix>]

  Name is the user-defined name for the device pin group which was previously defined in a defpin statement. Radix specifies the number base for each signal in the xv (and similar) instructions which follow. The default is decimal; other number bases are given after a colon. For example, hex stands for hexadecimal data, oct for octal and dec for decimal.

  The order in which the signals are specified in the defformat instruction determines the order in which they appear in the xv statement below.

  The ***clock*** signal does not appear in the defformat statement, nor in the xv statement.

- ***Deftiming*** specifies three timing intervals for the simulation:

- The first 1e-10 is the ***resolution*** of the simulation. ***Do not change this value***
- The second 1e-06 is the length required for the ***subdivision*** of a clock cycle.
- The third 1e-05 is the duration of a ***clock cycle.***

- ***Defstrobe*** defines input and output strobe windows. Input strobe windows define the time during the test at which input data should be applied. Output strobe windows define the time during the test period at which the circuit output pins should be sampled in order to compare the output data with the expected data.

    - The symbol **%** means that the window is open for that subdivision period; the symbol . means that the window is closed.
    - The qualifier ***edge*** means that signals are edge-triggered at the start of the window, the qualifier window means that signals maintain their levels throughout the window.

- ***Defclock*** defines the profile of the clock signal, in terms of the ***subdivision*** of a clock cycle in the deftiming statement.

  The symbol. means that the clock is low for that subdivision period; The symbol 1 means that the clock is high.

- ***xv*** (execute vector), or v in a defined sequence, specifies a set of simulation vectors, which is applied for one clock cycle.

    - The order of the signals is as defined in the defformat instruction.
    - For single-bit signals, 0 indicates driven low; 1 indicates driven high; Z indicates tristate and X unknown.
    - For the bus signals, the number base used must correspond to that given in the deformat statement. The prefixes which specify the number bases are as follows:

      0b binary

      0 octal

      0x hexadecimal

      The default is Decimal

- For bi-directional signals, two, signal specifications are required: one for input and the other for the expected output.

- ***Rptv*** repeats an xv command a specified number of times.

  The clock signal is not specified in the xv instruction - it follows the pattern set up in the defclock statement.

- ***Endtest*** indicates the end of the simulation.

13.0 Viewing the input Waveforms

The Waveforms of the input.stl file can be viewed before you simulate the design to verify that they are correct.

13.1 Click on ***Stimulus->STL->Show Stimulus Waveforms*** on the schematic menu banner.

The Waveform window appears with the input signals displayed.

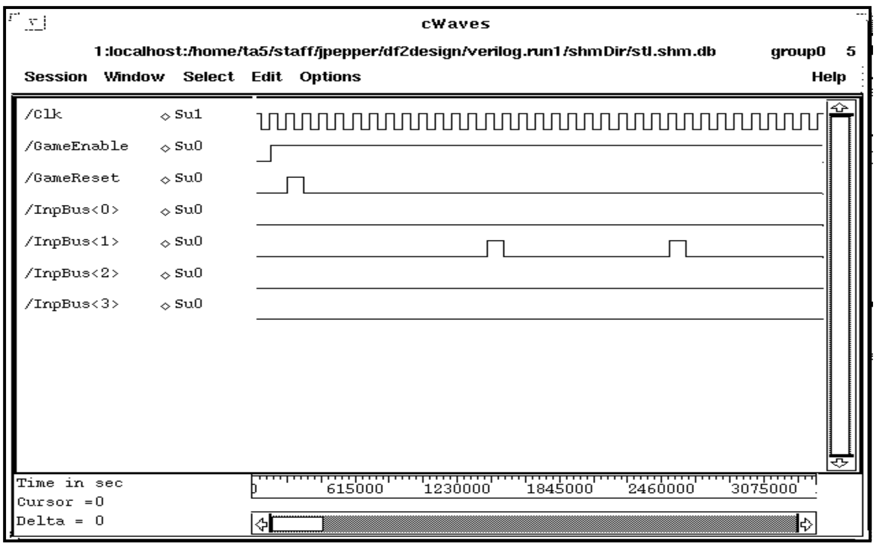
13.2 Move the mouse to one end of the time block.

13.3 Hold down the left mouse button.

13.4 Drag the mouse to increase or reduce the size of the block.

To quit the Waveform window:

13.5 Click on ***Session->Quit*** on the Waveform window.

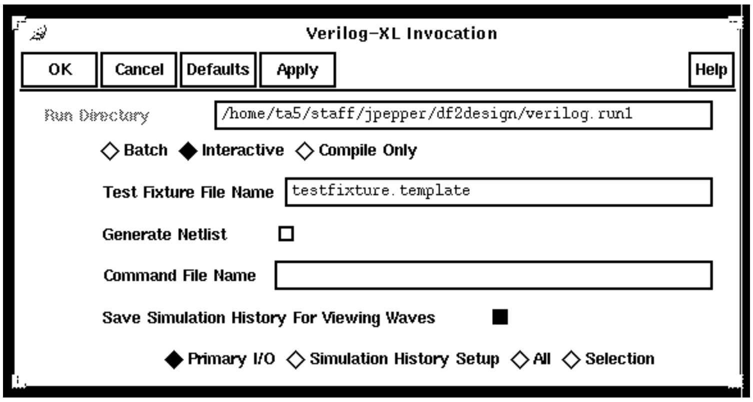


14.0 Running the simulation

Once you have netlisted your design and created an STL input file for the text fixture, then you are ready to simulate the design.

14.1 Click on ***Simulate->Invoke***.

The ***Verilog-XL Invocation*** form appears.

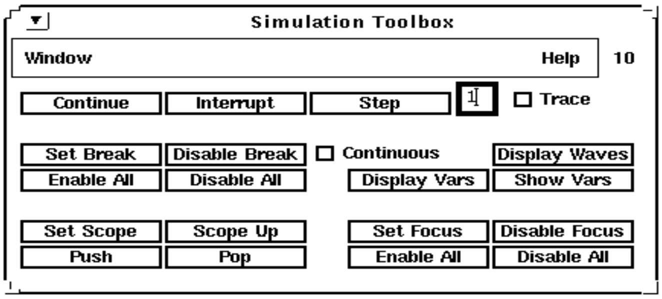


14.2 Select ***Save Simulation History For Viewing Waves***.

14.3 Click on ***OK***.

The Verilog-XL window now appears.This is used to display the Verilog-XL simulation output and lets you interact with the simulator.

14.4 Click on ***Utilities->Toolbox***.



The Toolbox window appears. With this window you can run the simulation and display the signals as waves. Before simulation you need to bring up the cWaves window so that the results of the simulation will be displayed as waves. To bring up the cWaves.

14.5 First select the wires/buses on your design that you want to display as waves.

14.6 Click on ***Display Waves*** on the ***Simulation Toolbo***x window to bring up the cWaves window.

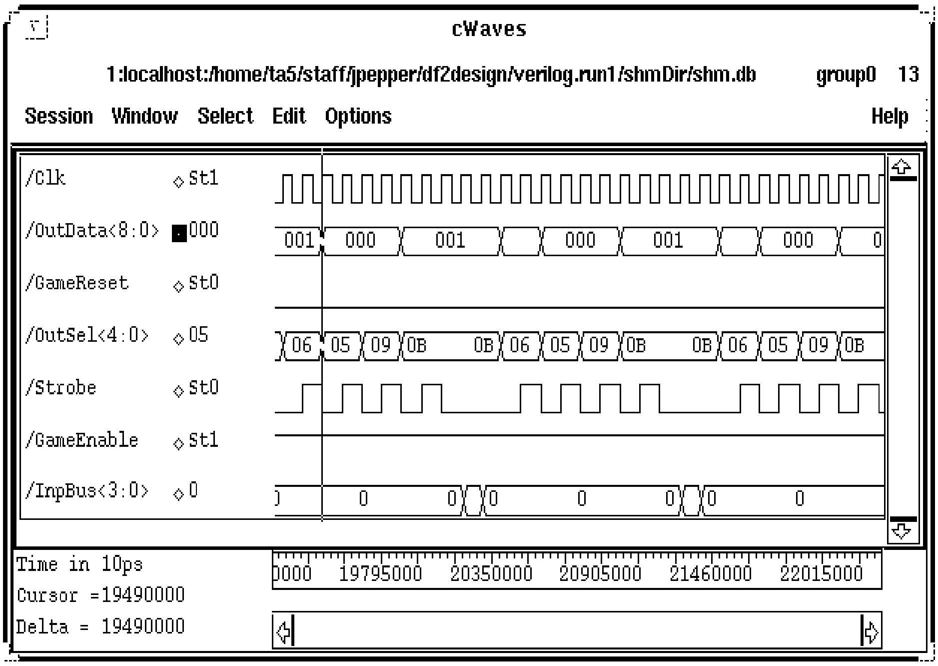
Once the cWaves window is open then you can run the simulation.

14.7 Click on ***Continue*** on the Toolbox window. This will continue the simulation until it reaches the end of the test fixture or an interrupt occurs.

The output will be displayed in the cWaves window.

15.0 Using cWaves

Once the cWaves window has been loaded with the results of the simulation you may find that the scale it too large to see any changes in the signals. There are several ways that you can Zoom out and View more of the results. The first is to use the Zoom Out command in the cWaves Window menu.



15.1 Hold down the middle mouse button on the waves to reveal the menu.

15.2 Choose the ***Zoom Out*** option.

It will Zoom Out by the Scale Factor set in the ***Options->Zoom Factor*** form. The default Scale Factor for Zoom Out is 2.

An easier way is to use the scroll bar on the bottom of the window.

15.3 Click the left mouse button in the middle of the scroll bar.

This will bring the ***Scroll Bar Block*** to the middle. This block can now be stretched to display more of the results.

15.4 Move the mouse to one ***end*** of the ***block***.

15.5 Hold down the left mouse button.

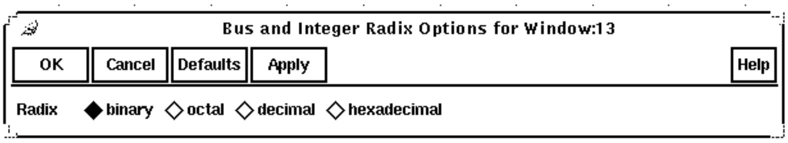
15.6 Drag the mouse to ***increase*** or ***reduce*** the size of the ***block***.

15.7 Release the mouse button.

You may also want to change the Radix for the signals (i.e. binary, octal, hex, decimal)

15.8 Click on the ***diamond*** of the signal which you want to change.

15.9 Click on ***Window->Set Radix.***



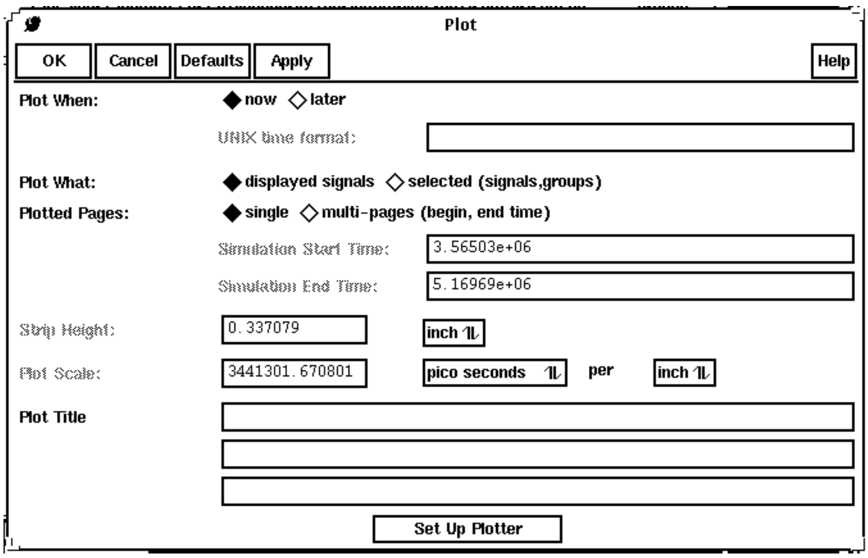
15.10 Select the Radix for the signal.

15.11 Click on ***OK***.

To Plot the cWaves window on the default laser printer follow the steps below:

15.12 Click on ***Session->Plot.***

This will bring up the plot form. To print the cWaves that are displayed in the window:



15.13 Click on ***OK*** to plot the cWaves.

16.0 Re-running the Simulation

- Before re-running the simulation use ***quit*** in the cwaves ***session*** menu.
- To re-probe the same simulation use ***Reinvoke*** in the ***Verilog-XL Session*** menu, followed by ***Display waves.***
- To re-simulate with a modified STL test program you need to
  - netlist
  - STL edit stimulus
  - Reinvoke
- To re-simulate with modified design, quit the session and start again.

17.0 SimDiff

SimDiff is used to compare the simulation results with the expected results specified in the STL file. Any mismatches that are found are displayed. To run SimDiff you first must have entered the expected results in the STL file ***input.stl.*** You then must have simulated the design using the same file. Once these two things have been done the SimDiff will compare the input.stl file with the simulation results from the current run directory. To use SimDiff:

17.1 Click on ***Simulate->Results->SimDiff*** on the Verilog-XL Environment Menu Banner.

If the SimDiff was successful then the following message will appear in the CIW:

‘STL simulation output comparison completed’

‘No simulation comparison mismatches detected’

18.0 Checking In

When you are satisfied that the design is correct, checkin your design as described in CADEN-2, section 12.