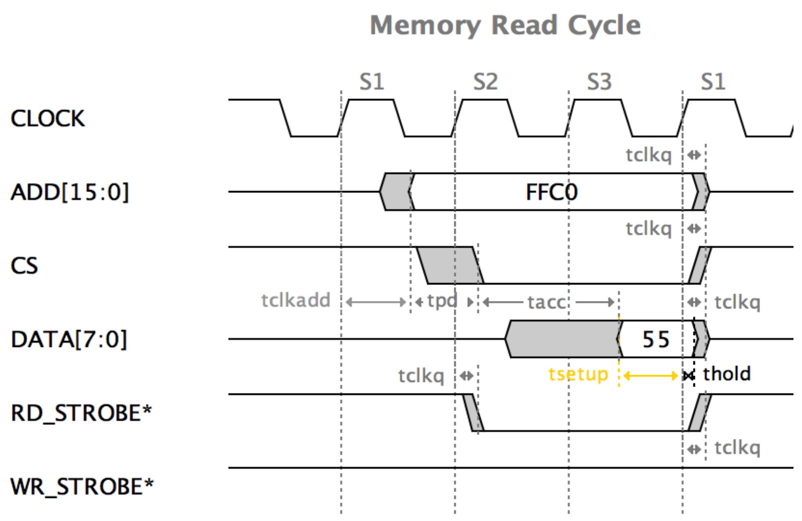
TimingAnalyzer Documentation

***Release 0.980**$_{b}eta$*

Dan Fabrizio

August 02, 2014



Using the TimingAnalyzer, you can quickly draw timing diagrams and build libraries of timing diagrams. Using the built-in analysis functions, you can quickly find speed related design issues since setup and hold violations are detected, reported, and shown in the diagrams.

You can draw the diagrams using the GUI or Python scripts. Timing diagrams can be created automatically from simulation VCD files so they can be annotated and used in documenation. Application notes included in the documentation desribe how to create Verilog or VHDL monitors that automatically generate timing diagrams.

SOME NEW CHANGES COMING IN BETA 0.980 SOON

A new app note, **"Intro to Timing Analysis"**, is included in the Documenation section. It describes how to analyze setup and hold paths with clock jitter and clock skew and much more. Included scripts can be used as is, or modified to analyze any user defined paths.

- New timing engine with accuracy to 1x10-15 seconds
- Added jitter margins
- Delays can show min or max annotations
- Web site moved to ReadTheDocs format

The new documentation can also be viewed and printed in various formats at:

- This Web Site.
- Reference.
- Scripting.
- Generating Timing Diagrams from Verilog and VHDL Simulations.
- Introduction to Timing Analysis.

CHAPTER 1

Features

- Generate timing diagrams directly from VHDL or Verilog simulations.
- Generate timing diagrams directly from VCD files.
- Easily draw and edit timing diagrams.
- Easily add pulses with one mouse click.
- Easily add clock synchronous pulses in any signal or bus
- Easily add automatically incrementing and decrementing pulses.
- Undo / Redo
- Timing analysis with min-max delay margins.
- User defined delays and constraints.
- Select transactions and move complete cycles synchronously
- Mouse moves edges and text.
- StateBars to view clock cycles in diagram.
- Text labels to make notes in diagram.
- Period labels to show clock periods and pulse widths.
- TimeWarps compress timing diagram in time.
- Logic simulations. DFF, Binary Counters, Inverter, Buffer, Differential Driver (More functions coming)
- Timing diagrams files are text formatted files for easy user control.
- Image preview showing diagram within common page sizes.
- Cross platform. Will run on any system with Java Virtual Machine JRE1.6.0 or newer.
- Recent file history to quickly load any of last 20 files
- Automatically loads timing diagrams left open from last session
- Save diagrams as JPG, GIF, PNG image files, and PS, PDF, and SVG formats.
- User control of color settings, fonts, font sizes, and more
- User scripts in Python.
- User scripts create custom features.
- User scripts generate test vectors and testbench files.

- User scripts quickly draw complex timing diagrams.

CHAPTER 2

Latest News

2.1 9/9/13 Beta version 0.970 is Released.

- New “Add Pulse” AP mode button in toolbar. When “Add Pulse” mode is enabled, you can add pulses or edges to any signal in the diagram. The Buttons for H L Z X B are disabled when “Add Pulse” mode is disabled.
- New “Show Hidden” button in the toolbar. Use this to show any hidden Pulse Width Labels, Constraints, or Delays. Use the pop-up menu to un-hide any selected hidden object. Use the pop-up menu to hide the object. The hidden objects are shown with a yellow background.
- Selected objects are shown with dashed line red rectangle around the border. Use toolbar buttons to change color, font, and line styles and see the results immediately
- New font select combobox and button in toolbar. You can easily change the font used for any text selected in the diagram with one click.
- Added Differential Driver. Given an input signal, it creates differential signals. Delays from the original input signal can be added to model path delays or skew on each signal.
- Added “Sync Clock” to DigitalSignal and DigitalBus. When adding pulses to a Signal or Bus, the clock above the signal is used by default if the Synchronous Clock is not specified. For example: How can I model clock tree skew? Add a Differential Driver with a DigitalClock named CLK as the input. Specify delays on both outputs CLK+ and CLK- to represent the different path delays and skew. Sounds like a good idea for another application note.
- Updated the Python API. Updated functions with more consistent argument positions for addDigitalClock, addDigitalBus, and addDigitalSignal. This will break older scripts but the changes are very simple.

The following functions are recommended when adding signals to a timing diagram.

- dclk = timDiagram.addDigitalClock(name, startState, freq)
- dclk = timDiagram.addDigitalClock(name, startState, freq, dutyCyle)
- dclk = timDiagram.addDigitalClock(name, startState, freq, dutyCyle, riseTime, fallTime)
- dsig = timDiagram.addDigitalSignal(name, startState)
- dsig = timDiagram.addDigitalSignal(name, startState, riseTime, fallTime)
- dbus = timDiagram.addDigitalBus(name, startState, stateFormat)
- dbus = timDiagram.addDigitalBus(name, startState, stateFormat, riseTime, fallTime)
- The following functions have been removed.

The following functions have been removed.

- timDiagram.addDigitalClock(dclk)
- timDiagram.addDigitalSignal(dsig)
- timDiagram.addDigitalBus(dbus)
- Fixed addTimeWarp script function.
- Plus other minor fixes and improvements.

2.2 5/2/13 Beta Version 0.960 is Released.

- PulseWidthLabels, Text Labels, Constraints and Delays can moved across TimeWarps.
- New Toolbar for quick way to add and edit edges and bus values.
- Improved drawing algorithm with TimeWarps improves diagram load times and allows start times greater than 0.
- Diagrams can start at a time greater than 0. This fixes a problem related to the scripts for the VHDL and Verilog monitors that were described in the app notes. If you captured signals from time t1 to time t2, the first edge would be wrong if t1 was not 0.
- Added more error checking for edges that were moved and ended up at a time that is less than zero. This can happen when move multiple edges at one time.
- Removed number of edges for each signal in the .tim file.
- Other fixes and improvements

2.3 3/1/13 Beta Version 0.958 Is Coming.

Below is preliminary list of changes so far. Much more coming.

- PulseWidthLabel can moved across TimeWarps. Once verified by users, this will be added to Delays and Constraints.
- Added more error checking for edges that were moved and ended up at a time that is less than zero. This can happen when move multiple edges at one time.
- Removed number of edges for each signal in the .tim file.

2.4 8/16/11

The script below can be used to generate a differential signal. It will be included in the distribution of the next version beta 0.957. Just select clocks or signals in the timing diagram and then run the script. It will add the differential signals for each selected signal. Save the script below as diff_signal.py and put in the scripts directory so it can be executed from the program script panel:

```
import ta_utils
from org.dmad.ta import DigitalBus

td = taApp.getTimingDiagram()

selected_signal_list = td.getEditSigList()
```

```
for selected_signal in selected_signal_list:
    if not isinstance(selected_signal, DigitalBus):
        diff_signal = td.addDigitalSignal(selected_signal.getName() + "_diff",
                                          ta_utils.invert(selected_signal.getStartState()))
        for sig_edge in selected_signal.getEdgeList():
            diff_signal.addEdge(sig_edge.getPt2Min(),ta_utils.invert(sig_edge.getNextState()))
```

Add the following function to ta_utils.py. It is called in diff_signal.py above:

```
def invert(state):
    states = {
        "H": "L",
        "1": "L",
        "L": "H",
        "0": "H",
        "Z": "Z"
    }
    return states.get(state)
```

2.5 5/12/11

Started a new timing diagram library. A place to hold scripts that generate timing diagrams for commonly used interfaces.

2.6 5/1/11

Added a FAQ. A place to look for answers to commonly asked questions.

2.7 2/5/11

Added new documentation. A scripting reference manual reference manual. Both are available in html and pdf. A users guide is coming.

2.8 9/29/10

Added move signal up and down buttons on the toolbar.

Jython scripts now use the standard libraries without the need for installing jython. The standalone jython.jar archive includes the standard libs, and is included in the jlib directory. When executing scripts from the File → Scripts menu, the interpreter is started and the os and sys libs are imported before executing the selected script file. You can also run scripts from the command line shell or dos window using the following commands:

```
c:\Apps\TimingAnalyzer_b951> java -jar jlib\jython.jar
...
>>>execfile('start_app.py')
>>>execfile('./scripts/dump_edges.py')
```

2.9 2/6/10

The newest feature is ability to read VCD files and convert it to a timing diagram automatically. I have tested with Xilinx Chipscope and Modelsim VCD files. Modelsim VCD files don't seem to contain buses so I added a function, "Bus Signals" to make buses from sequentially ordered signals like ADIO<31>, ADIO<30>, ADIO<29> ... You have to hold the shift key and click on each signal to select multiple signals. Better signal group selections will be addressed in the next version. Selecting groups of signals will only take 2 clicks.

Refer to the documentation for more information

2.10 1/24/10

Added "Generate Timing Diagrams from Verilog Simulations" Application Note. This app note shows how to use Verilog to generate timing diagrams by writing text files which are Python scripts that the TimingAnalyzer executes to draw the diagram. A separate module in the example, sram_timing_diagram.v, includes all the source code use to generate the Python script. You can read it and download the example at: Generate Timing Diagrams from Verilog Simulations

2.11 12/15/09

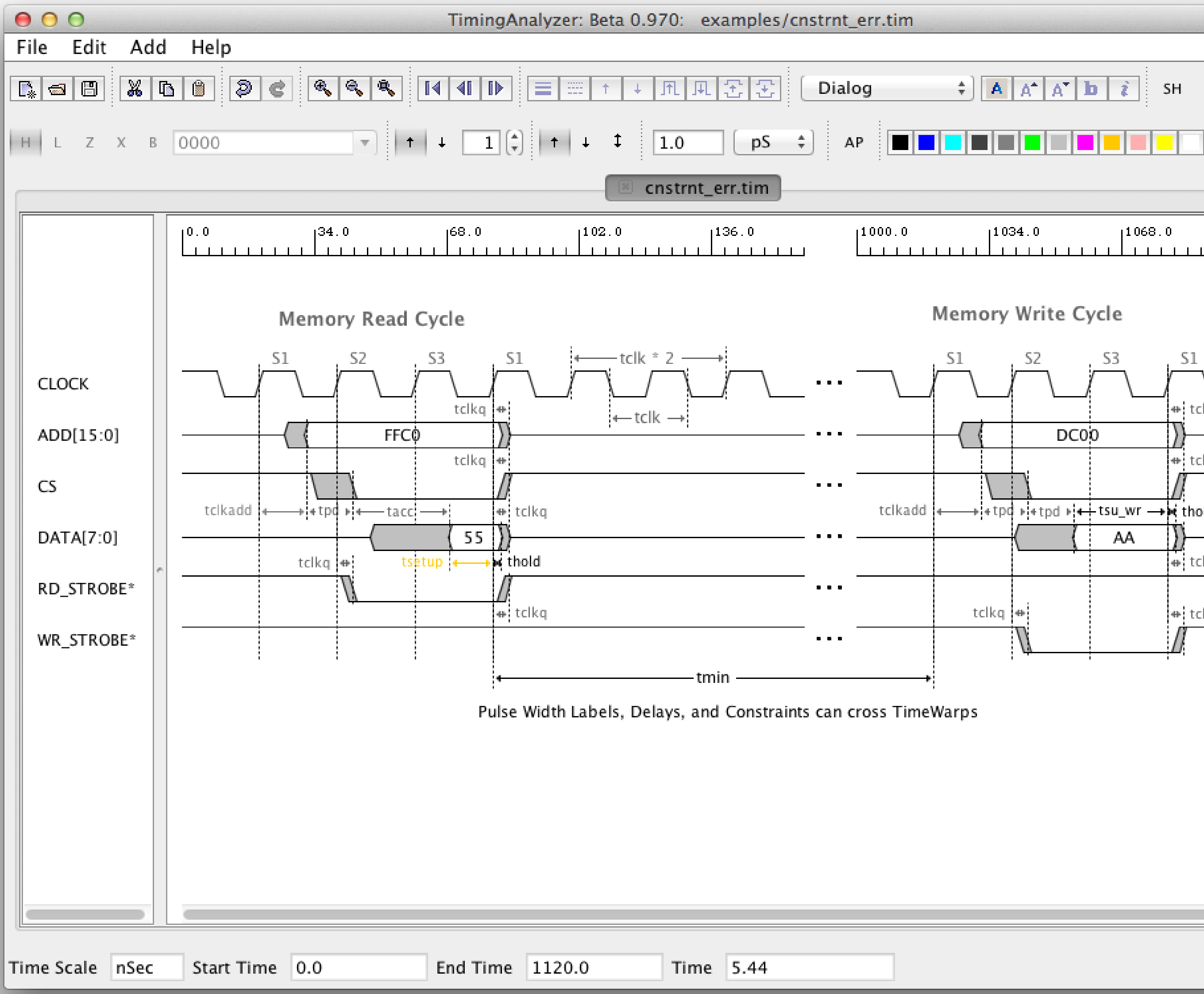
Updated the "Generate Timing Diagrams from VHDL Simulations" Application Note. A separate component, sram_timing_diagram.vhd, is now used to generate the timing diagram Python script. Start and end time are specified as generic parameters so you can make timing diagrams of any window in time from the simulation. This could be used as a template and modified to generate timing diagrams for any interface. A python script could developed that automatically builds this file for given list of signals, anyone interested?

2.12 12/1/09

Added app notes that explain how to build timing diagrams directly from VHDL or Verilog. Refer to Generate Timing Diagrams from VHDL Simulations and Generate Timing Diagrams from Verilog Simulations

CHAPTER 3

Screenshots and Videos



Time Scale nSec Start Time 0.0 End Time 1120.0 Time 5.44

3.1 Moving Transactions Synchronously

Moving transactions synchronously can make life much easier when you have add signal changes or transactions to an existing timing diagram. For example, if you decide to add a read cycle in between two write cycles you would need to move the 2nd write cycle to the right by maybe 10 clock cycles. This is done with synchronous moves. Simply select the 2nd write cycle by dragging the mouse around the transaction, then use Alt and Right Key combination to move it.

See the video shown below to see an example.

Move Synchronoulsy Video.

CHAPTER 4

Getting Started

4.1 Installing the TimingAnalyzer

- Download and install the JRE Version 1.6.0 or newer if needed.
- Download the zip file.
- Unzip the downloaded TimingAnalyzer_bxx.zip in any directory:

```
c:\Apps\TimingAnalyzer_bxx
      TimingAnalyzer.jar     --   The executable program
      start_app.py           --   Python script to start the program
      scripts                --   directory for user scripts
      examples               --   directory for timing diagram examples
      settings               --   directory for default and user settings
```

4.2 Running the TimingAnalyzer

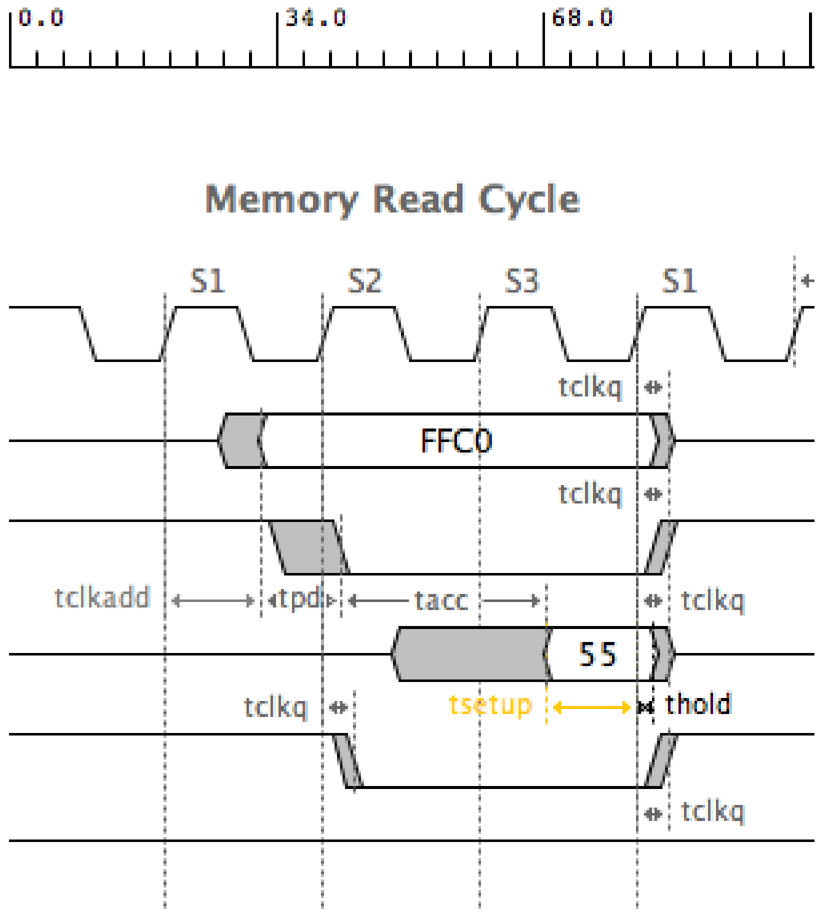
- Windows Desktop
  - Make shortcut icon of TimingAnalyzer.jar on Desktop.
  - Double click TimingAnalyzer.jar icon on Desktop.
- Dos Window
  - cd c:\Apps\TimingAnalyer_bxx
  - java -jar TimingAnalyzer.jar

4.3 System Requirements

- Oracle Java JRE1.6.0 or JDK1.6.0 or newer.
- Windows XP, Vista, Linux, and OS X.

Step by Step Example

‘‘



5.1 The Timing Diagram Specifications

‘‘

Signal Name	Freq	Duty Cycle	Edge Times	Start State	Format
CLOCK	50MHz	50%	2nS	H	
ADD[15:0]			2nS	Z	Hex
CS*			2nS	H	
DATA[7:0]			2nS	Z	Hex
RD_STROBE*			2nS	H	
WR_STROBE*			2nS	H	

“

Delays	Min	Typ	Max
tclkadd	7nS	10nS	12nS
tpd	7nS	10nS	12nS
tacc	15nS	20nS	25nS
tclk2q	2nS	3nS	4nS

“

Constraints	Min	Typ	Max
tsetup	12nS	12nS	12nS
thold	1nS	1nS	1nS

5.2 Start a New Timing Diagram

- To start a diagram, File Menu -> New or Ctrl-n
    1. Start a new timing diagram.

5.3 Add the User Defined Constraints and Delays.

- To add a User Delay, Add Menu -> Delay or Ctrl-7, select User Delay tab in panel.
- To add a User Constraint, Add Menu -> Constraint or Ctrl-8, select User Constraint tab in panel.
    1. Add the Delays shown above.
    2. Add the Constraints shown above.

5.4 Adding the Signals

- To add a Clock, Add Menu -> Clock or Ctrl-1
- To add a Signal, Add Menu -> Signal or Ctrl-2
- To add a Bus, Add Menu -> Bus or Ctrl-3
    1. Add Clock CLOCK.
    2. Add Bus ADD.
    3. Add Signal CS*.
    4. Add Bus DATA.
    5. Add Signal RD_STROBE*.
    6. Add Signal WR_STROBE*.

5.5 Adding the Pulses to the Signals

- To enable “Add Pulse” mode, click on AP button. Button background color turns dark.
- To disable “Add Pulse” mode, click on AP button. Button background color turns light.

1. Enable “Add Pulse” mode.
2. Set “Auto Increment” in toolbar to 0.
3. Set signal value in toolbar to FFC0.
4. Click in ADD bus at 30ns,50ns, and 70nS. This will make a FFC0 pulse for 3 clock cycles.
5. Select “L”. Click in CS* at 50nS and 70nS. This will make a low pulse for 2 clock cycles.
6. Change signal value to 55. Click in DATA bus at 70nS. This will make a 55 pulse for 1 clock cycle.
7. Select “L”. Click in RD_STROBE* at 70nS. This will make a low pulse for 1 clock cycle.
8. Disable “Add Pulse” mode.

5.6 Adding the Delays and Constraints

- To add a Delay, Add Menu -> Delay or Ctrl-7 or right button click for pop-up menu
- To add a Constraint, Add Menu -> Constraint or Ctrl-8 or right button click for pop-up menu
- To add a previously used Delay, right button click for pop-up menu and select “Add Used Delay”
- To add a previously used Constraint, right button click for pop-up menu and select “Add Used Constraint”

1. Select 1st rising edge of CLOCK and 1st edge of ADD. Add delay tclk2add.
2. Select 1st edge of ADD and 1st edge of CS*. Add delay tpd.
3. Select 1st edge of CS* and 1st edge of DATA. Add delay tacc.
4. Select 1st edge of DATA and 4th rising edge of CLOCK.
5. Add Constraint. Select Max-Min Type. Add constraint tsetup.
6. Select 4th rising edge of CLOCK and 2nd edge of ADD. Add delay tclk2q.
7. Select 4th rising edge of CLOCK and 2nd edge of CS*. Add previously used delay tclk2q
8. Select 4th rising edge of CLOCK and 2nd edge of DATA. Add previoulsy used delay tclk2q
9. Select 4th rising edge of CLOCK and 2nd edge of DATA. Select Max-Min. Add constraint thold.
10. Select 4th rising edge of CLOCK and 2nd edge of RD_STROBE*. Add delay tclk2q
11. Click in delay and contraint labels in diagram and then use the arrow key combinations to position the labels.

5.7 Adding the StateBars

- To add a StateBar, Add Menu -> StateBar or Ctrl-4

1. Select 1st rising edge of CLOCK. Add StateBar S1 using dialog.
2. Select 2nd rising edge of CLOCK. Add StateBar S2 using dialog already open.

3. Select 3nd rising edge of CLOCK. Add StateBar S3 using dialog already open.
4. Select 4nd rising edge of CLOCK. Add StateBar S1 using dialog already open.

5.8 Timing Analysis

You could increase the CLOCK frequency or change any of the delays to quickly check for constraint violations. This will indicate problems with fast clock rates or slower parts. Drag a Delay or Constraint edge to see the results immediately.

CHAPTER 6

Documentation

6.1 Manuals

- Reference.
- Scripting.

6.2 Application Notes

- Generating Timing Diagrams from Verilog and VHDL Simulations.
- Introduction to Timing Analysis.

CHAPTER 7

Timing Diagram Library

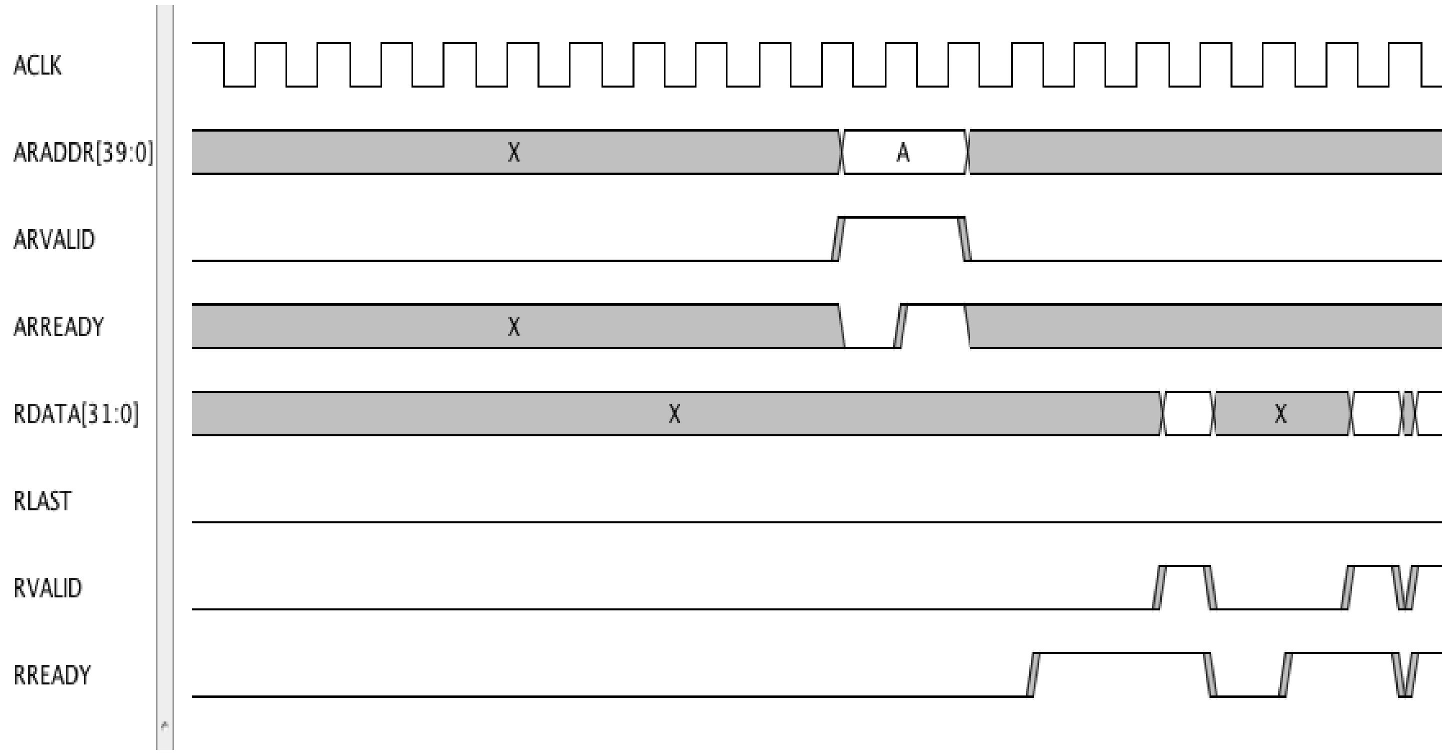
A work in progress. Keep checking back for more diagrams. Feel free to request timing diagrams you would like to see added to the library.

Each timing diagram in the library includes the python script source. You can change the clock frequency parameter in each script to meet your design requirements.

7.1 AXI Burst Read

Save the following script as axi_burst_read.py

```
# start a new file
if taApp.getFileName() != "axi_burst_read.tim":
    taApp.fileNew("TimingDiagram")

# td is the current TimingDiagram Frame
td = taApp.getTimingDiagram()

aclk    = td.addDigitalClock("ACLK", "H", 100.0e6)
aclk.setRiseTime(0.0)
aclk.setFallTime(0.0)

araddr  = td.addDigitalBus("ARADDR[39:0]", "X", "Hex")
arvalid = td.addDigitalSignal("ARVALID", "L")
arready = td.addDigitalSignal("ARREADY", "X")
rdata   = td.addDigitalBus("RDATA[31:0]", "X", "Text")
rlast   = td.addDigitalSignal("RLAST", "L")
rvalid  = td.addDigitalSignal("RVALID", "L")
rready  = td.addDigitalSignal("RREADY", "L")

td.addEdge(araddr, 103.0e-9, "A")
td.addEdge(araddr, 123.0e-9, "X")

td.addEdge(arvalid, 102.0e-9, 103.0e-9, "H")
td.addEdge(arvalid, 122.0e-9, 123.0e-9, "L")

td.addEdge(arready, 103.0e-9, "L")
td.addEdge(arready, 112.0e-9, 113.0e-9, "H")
td.addEdge(arready, 123.0e-9, "X")

td.addEdge(rdata, 154.0e-9, "D(A0)")
td.addEdge(rdata, 162.0e-9, "X")
td.addEdge(rdata, 184.0e-9, "D(A1)")
td.addEdge(rdata, 192.0e-9, "X")
td.addEdge(rdata, 194.0e-9, "D(A2)")
td.addEdge(rdata, 202.0e-9, "X")
td.addEdge(rdata, 224.0e-9, "D(A3)")
td.addEdge(rdata, 232.0e-9, "X")

td.addEdge(rlast, 223.0e-9, 224.0e-9, "H")
td.addEdge(rlast, 231.0e-9, 232.0e-9, "L")

td.addEdge(rvalid, 153.0e-9, 154.0e-9, "H")
td.addEdge(rvalid, 161.0e-9, 162.0e-9, "L")
td.addEdge(rvalid, 183.0e-9, 184.0e-9, "H")
td.addEdge(rvalid, 191.0e-9, 192.0e-9, "L")
td.addEdge(rvalid, 193.0e-9, 194.0e-9, "H")
td.addEdge(rvalid, 201.0e-9, 202.0e-9, "L")
td.addEdge(rvalid, 223.0e-9, 224.0e-9, "H")
td.addEdge(rvalid, 231.0e-9, 232.0e-9, "L")


td.addEdge(rready, 133.0e-9, 134.0e-9, "H")
td.addEdge(rready, 161.0e-9, 162.0e-9, "L")
td.addEdge(rready, 173.0e-9, 174.0e-9, "H")
td.addEdge(rready, 191.0e-9, 192.0e-9, "L")
td.addEdge(rready, 193.0e-9, 194.0e-9, "H")
td.addEdge(rready, 211.0e-9, 212.0e-9, "L")
td.addEdge(rready, 223.0e-9, 224.0e-9, "H")
td.addEdge(rready, 231.0e-9, 232.0e-9, "L")

td.setEndTime(300.0e-9)
td.doAnalysisCommand("margins")
```

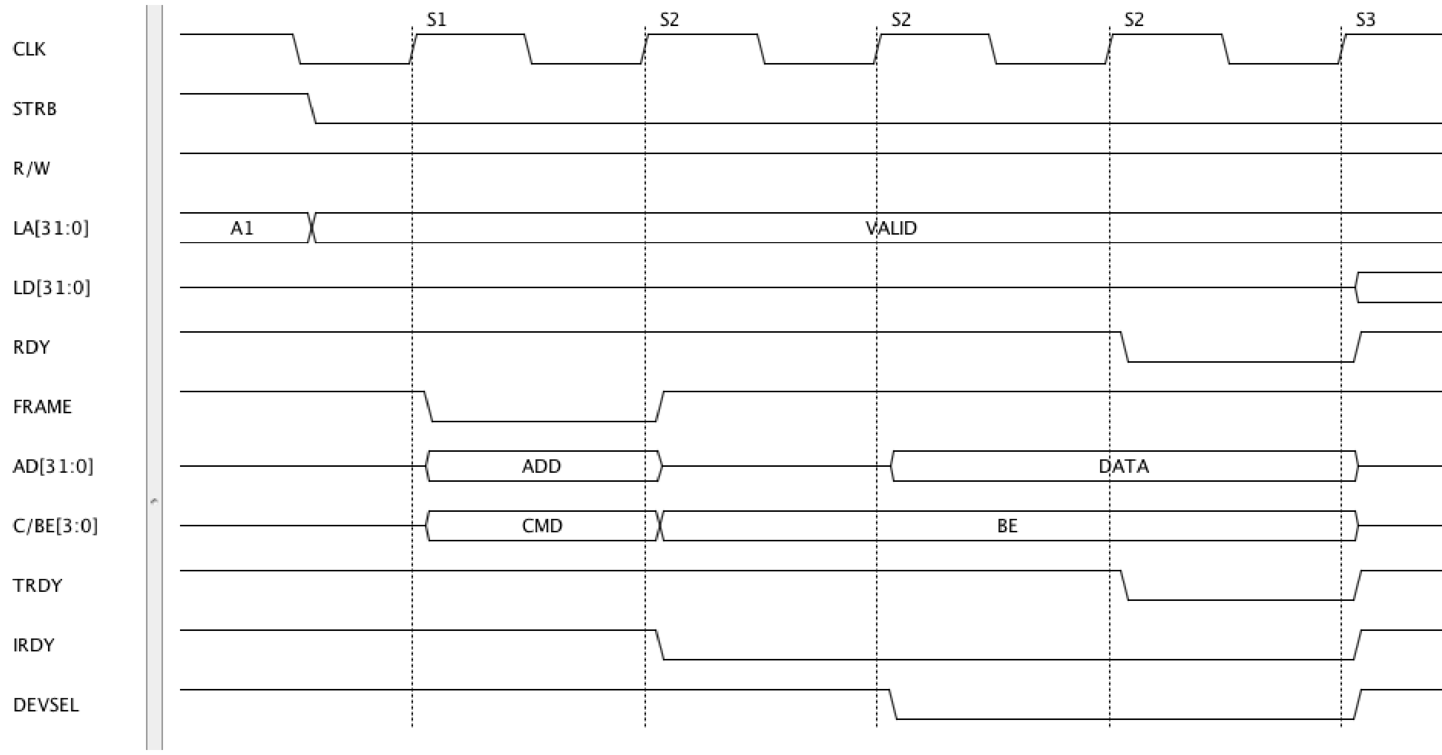


7.2 PCI IO Read

Save the following script as pci_io_read.py

```
taApp.fileNew("TimingDiagram")
td = taApp.getTimingDiagram()

clk_freq = 33.0e6
clk_per  = 1.0/clk_freq

clk     = td.addDigitalClock("CLK", "H", clk_freq)
strb    = td.addDigitalSignal("STRB", "H")
r_w     = td.addDigitalSignal("R/W", "H")
la      = td.addDigitalBus("LA[31:0]", "A1", "Text")
ld      = td.addDigitalBus("LD[31:0]", "Z", "Text")
rdy     = td.addDigitalSignal("RDY", "H")
frame   = td.addDigitalSignal("FRAME", "H")
ad      = td.addDigitalBus("AD[31:0]", "Z", "Text")
c_be    = td.addDigitalBus("C/BE[3:0]", "Z", "Text")
trdy    = td.addDigitalSignal("TRDY", "H")
irdy    = td.addDigitalSignal("IRDY", "H")
devsel  = td.addDigitalSignal("DEVSEL", "H")


def add_edges(clk_cycle):
   fe_time = (clk_cycle*clk_per) + (clk_per/2.0) + 2.0e-9
   re_time = clk_cycle*clk_per + 2.0e-9
   if clk_cycle == 0:
       td.addEdge(strb,fe_time, "L")
       td.addEdge(la,  fe_time, "VALID")
   elif clk_cycle == 1:
       td.addEdge(frame,re_time, "L")
       td.addEdge(ad,re_time, "ADD")
       td.addEdge(c_be,re_time, "CMD")
   elif clk_cycle == 2:
       td.addEdge(frame,re_time, "H")
       td.addEdge(ad,re_time, "Z")
       td.addEdge(c_be,re_time, "BE")
       td.addEdge(irdy,re_time, "L")
   elif clk_cycle == 3:
       td.addEdge(ad,re_time, "DATA")
       td.addEdge(devsel,re_time, "L")
   elif clk_cycle == 4:
       td.addEdge(rdy,re_time, "L")
       td.addEdge(trdy,re_time, "L")
   elif clk_cycle == 5:
       td.addEdge(strb,fe_time, "H")
       td.addEdge(la,fe_time, "A1")
       td.addEdge(ld,re_time, "VALID")
       td.addEdge(rdy,re_time, "H")
       td.addEdge(ad,re_time, "Z")
       td.addEdge(c_be,re_time, "Z")
       td.addEdge(trdy,re_time, "H")
       td.addEdge(irdy,re_time, "H")
       td.addEdge(devsel,re_time, "H")
   elif clk_cycle == 6:
       td.addEdge(ld,re_time, "Z")
```

```
for clk_cycle in range (0,7):
    add_edges(clk_cycle)

state_list = ['NU','S1','S2','S2','S2','S3','S0']

clk_cycle = 0
for clk_edge in clk.getEdgeList():
    if clk_cycle == 7:
        break
    if clk_edge.getNextState() == "H":
        td.addStateBar(clk_edge,state_list[clk_cycle],"Dashed",0,0)
        clk_cycle+=1

td.setEndTime(300.0e-9)
```



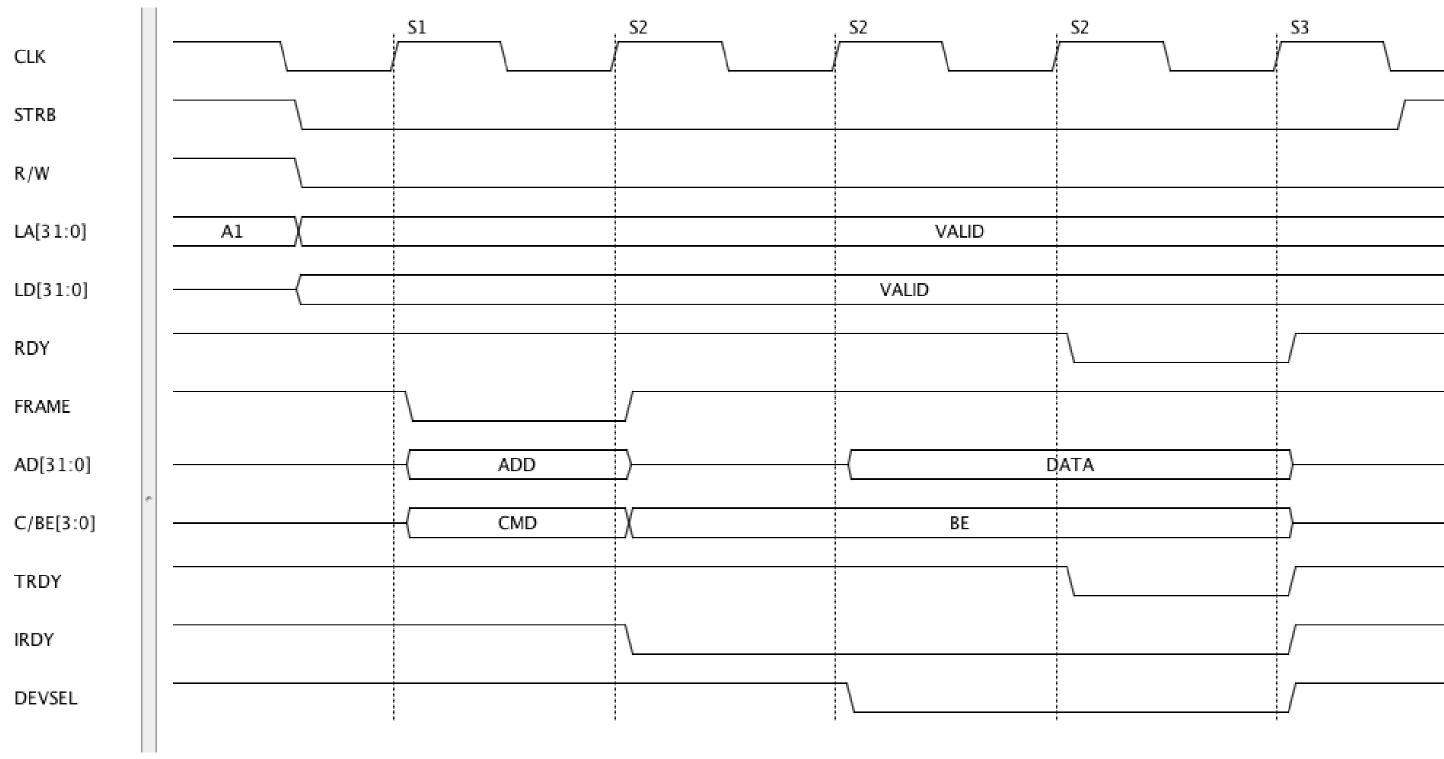
7.3 PCI IO Write

Save the following script as pci_io_write.py

```
taApp.fileNew("TimingDiagram")
td = taApp.getTimingDiagram()

clk_freq = 33.0e6
clk_per  = 1.0/clk_freq

clk     = td.addDigitalClock("CLK", "H", clk_freq)
strb    = td.addDigitalSignal("STRB", "H")
r_w     = td.addDigitalSignal("R/W", "H")
la      = td.addDigitalBus("LA[31:0]", "A1", "Text")
ld      = td.addDigitalBus("LD[31:0]", "Z", "Text")
```

```
rdy      = td.addDigitalSignal("RDY", "H")
frame    = td.addDigitalSignal("FRAME", "H")
ad       = td.addDigitalBus("AD[31:0]", "Z", "Text")
c_be     = td.addDigitalBus("C/BE[3:0]", "Z", "Text")
trdy     = td.addDigitalSignal("TRDY", "H")
irdy     = td.addDigitalSignal("IRDY", "H")
devsel   = td.addDigitalSignal("DEVSEL", "H")


def add_edges(clk_cycle):
   fe_time = (clk_cycle*clk_per) + (clk_per/2.0) + 2.0e-9
   re_time = clk_cycle*clk_per + 2.0e-9
   if clk_cycle == 0:
       td.addEdge(strb,fe_time, "L")
       td.addEdge(la,   fe_time, "VALID")
       td.addEdge(ld,   fe_time, "VALID")
       td.addEdge(r_w,  fe_time, "L")
   elif clk_cycle == 1:
       td.addEdge(frame,re_time, "L")
       td.addEdge(ad,re_time, "ADD")
       td.addEdge(c_be,re_time, "CMD")
   elif clk_cycle == 2:
       td.addEdge(frame,re_time, "H")
       td.addEdge(ad,re_time, "Z")
       td.addEdge(c_be,re_time, "BE")
       td.addEdge(irdy,re_time, "L")
   elif clk_cycle == 3:
       td.addEdge(ad,re_time, "DATA")
       td.addEdge(devsel,re_time, "L")
   elif clk_cycle == 4:
       td.addEdge(rdy,re_time, "L")
       td.addEdge(trdy,re_time, "L")
   elif clk_cycle == 5:
       td.addEdge(strb,fe_time, "H")
       td.addEdge(rdy,re_time, "H")
       td.addEdge(ad,re_time, "Z")
       td.addEdge(c_be,re_time, "Z")
       td.addEdge(trdy,re_time, "H")
       td.addEdge(irdy,re_time, "H")
       td.addEdge(devsel,re_time, "H")
   elif clk_cycle == 6:
       td.addEdge(r_w, re_time, "H")
       td.addEdge(ld,re_time, "Z")
       td.addEdge(la,fe_time, "A1")

for clk_cycle in range (0,7):
    add_edges(clk_cycle)

state_list = ['NU','S1','S2','S2','S2','S3','S0']

clk_cycle = 0
for clk_edge in clk.getEdgeList():
    if clk_cycle == 7:
        break
    if clk_edge.getNextState() == "H":
        td.addStateBar(clk_edge,state_list[clk_cycle],"Dashed",0,0)
        clk_cycle+=1
```

```
td.setEndTime(300.0e-9)
```



CHAPTER 8

Download

Download Version Beta0.971.

- Fix Signal names missing in save images.
- Fix Last directory remembered with saving images
- Fix Delay and Constraint edge positions used.
- Fix Added Max Constraint back in checks.
- Changed User Delay to Part Delay
- Changed User Constraint to Part Constraint.
- Added Part Delay GUI panel.
- Added Part Constraint GUI panel.

Download Version Beta0.970.

- New “Add Pulse” AP mode button in toolbar. When “Add Pulse” mode is enabled, you can add pulses or edges to any signal in the diagram. The Buttons for H L Z X B are disabled when “Add Pulse” mode is disabled.
- New “Show Hidden” button in the toolbar. Use this to show any hidden Pulse Width Labels, Constraints, or Delays. Use the pop-up menu to un-hide any selected hidden object. Use the pop-up menu to hide the object. The hidden objects are shown with a yellow background.
- Selected objects are shown with dashed line red rectangle around the border. Use toolbar buttons to change color, font, and line styles and see the results immediately
- New font select combobox and button in toolbar. You can easily change the font used for any text selected in the diagram with one click.
- Added Differential Driver. Given an input signal, it creates differential signals. Delays from the original input signal can be added to model path delays or skew on each signal.
- Added “Sync Clock” to DigitalSignal and DigitalBus. When adding pulses to a Signal or Bus, the clock above the signal is used by default if the Synchronous Clock is not specified. For example: How can I model clock tree skew? Add a Differential Driver with a DigitalClock named CLK as the input. Specify delays on both outputs CLK+ and CLK- to represent the different path delays and skew. Sounds like a good idea for another application note.
- Updated the Python API. Updated functions with more consistent argument positions for addDigitalClock, addDigitalBus, and addDigitalSignal. This will break older scripts but the changes are very simple. I will add a note in the announcement that shows how to fix the scripts.

  The following functions are recommended when adding signals to a timing diagram.

  - dclk = timDiagram.addDigitalClock(name, startState, freq)
  - dclk = timDiagram.addDigitalClock(name, startState, freq, dutyCyle)
  - dclk = timDiagram.addDigitalClock(name, startState, freq, dutyCyle, riseTime, fallTime)
  - dsig = timDiagram.addDigitalSignal(name, startState)
  - dsig = timDiagram.addDigitalSignal(name, startState, riseTime, fallTime)
  - dbus = timDiagram.addDigitalBus(name, startState, stateFormat)
  - dbus = timDiagram.addDigitalBus(name, startState, stateFormat, riseTime, fallTime)

  The following functions have been removed.

  - timDiagram.addDigitalClock(dclk)
  - timDiagram.addDigitalSignal(dsig)
  - timDiagram.addDigitalBus(dbus)

- Fixed addTimeWarp script function.
- Plus other minor fixes and improvements.

Download Version Beta0.963.

- Fixed Edge fall time – Signals were using one value for rise and fall times.
- Fixed Locale settings to english and US. This should override any settings made by the JVM. This Locale is displayed if started from the command line.
- Added VCD Analog Signals. Try sine_waves.vcd in example dir. VCD files get converted automatically to timing diagrams so this is the start of AnalogSignal for timing diagrams. More capablities will be added in future versions. Recommendations and suggestions welcome from users. Looking for VCD file examples from users to help test this feature.
- Started the clean-up of the examples directory. There where many examples from previous versions of the TimingAnalyzer that were not working.
  - cnstrnt_err.tim
  - cram_read.tim
  - pci_bus_master_mem_read.tim
  - pci_bus_master_mem_write.tim
  - pci_io_read.tim
  - pci_io_write.tim
  - sine_waves.vcd

Download Version Beta0.962.

- Fixed VCD file loading
- Fixed Python scripts not executing from GUI

Download Version Beta0.961.

- Fixed update UserDelay options
- Added constraint margin time in display when “show constraint time” is selected.
- Fixed Clock, Signal, and Bus edge values only using pS when adding signals

Download Version Beta0.960.

- PulseWidthLabels, Text Labels, Constraints, and Delays can cross TimeWarps
- Improved drawing routines allow timing diagram to start at a time other than 0
- Fixes VHDL and Verilog monitors so simulation waveform views can start at a time other than 0
- Add and Edit Pulses and Edges with new toolbar .
- Many other improvements and fixes

Download Version Beta0.957.

- Included images directory in TimingAnalyzer.jar to clean up the install directory
- Fixed bus value select when the bus value did not contain a number.
- Fixed clock edge times displayed that were rounded to ns. Now accurate to +/- 1.0E-15.
- Fixed user delay / constraint panel textfields position on OS X

CHAPTER 9

Frequently Asked Questions

9.1 Why won’t it start?

In the settings directory, there is a file called ta_open_files_list. This file contains the names of the timing diagram files that were left open when you last closed the program. If one of the files is missing or it was moved, this would cause the program to hang when starting. If one of the files is saved on a network drive and the network drive is not mapped or not responding, this would also cause the program to hang when starting.

9.2 How do I start the program from the command line?

You might want to start the program from the command line if your developing python scripts and want to see the debug information that is output to the shell.

Refer to getting_started.

You might also want to start the program from the command line and specify more heap memory if your converting larger vcd waveforms diagrams to timing diagrams.

Refer to Working with VCD Files

9.3 How do I perform a timing analysis?

In older versions of the program, you had to select the analysis mode using a menu. You had a choice of min, typ, max, or margins and you couldn’t edit the timing diagram when margins was selected. Now, it is always in the margins mode and you can edit the diagram. You can also move the min and max edges in an edge margin or delay with the mouse or arrow keys in combination with Crtl, Shft, and Alt keys.

Basic procedure for timing analysis:

- Add delays that represent real part delays specified by the manufacturer.
- Add constraints that represent real part constraints specified by the manufacturer. Constraint errors are shown in orange, otherwise they are gray. A report of the delays and constraints will be coming a new version soon.
- Move the edges that connect to one of the delays in a path with the mouse and see how it affects the constraint. If there is a constraint error, you can move one of the edges to the left decreasing the delay time. If the constraint error is eliminated (color changed to gray), a faster part with less delay is one way to fix the problem. If there isn’t a constraint error and you move one of the edges increasing the delay time and a constraint error occurs, then you might be able to use a slower part with more delay and still meet the timing requirements.

9.4 What new features are planned

- generate reports listing actual delays and constraint values
- VCD output from timing diagram for use as simulator test vectors.
- bus enumerations for state machines
- Min/Max VCD diagrams for gate level simulation results
- more timing diagram generator scripts (common bus protocols)
- Define transactions. Copy and paste transactions
- Printing diagrams.

9.5 What improvements are planned

- change to individual User Delay and User Constraint panels.
- edit TimeWarps.
- generate timing diagrams directly from SystemVerilog app note
- generate timing diagrams directly from SystemC app note
- multi line text labels
- horizontal dividers to label groups of signals
- statebar grid shows vertical bars on all rising or falling clock edges
- output python script commands instead of command text.
- User guide and intro to timing analysis app note

9.6 What known issues will be addressed

- larger bus values. You can specifiy bus values greater than 32 bits, but any math functions are limited to 32 bit signed numbers.
- arrows don't look right when delays are very small
- same constraint/delay name shown multiple times in gui combobox
- when bus values are separated by a timewarp, you can not edit the value on the left
- lines are not aligned to the edges for 0.0 rise and fall times
- undo "delete signal" does not bring the signal back sometimes
- undo moving objects sometimes don't go back to the original position
- text labels are drawn below the image boundaries when signal heights are increased
- colors in image do not match the signal line and edge colors
- SaveAs adds multiple file extensions to the file name when "All Files" filter is selected

9.7 I'm having other problems, what should I do?

If your problem is not shown in the list above, send an email that includes a dump of the command line text logs.

Start the program from the command line as described in getting started, copy the information that is displayed in the shell window, and send to timinganalyzer@gmail.com with an explanation of the issue your having.

CHAPTER 10

Google Group

A place to discuss using the TimingAnalyzer, reporting problems, new feature requests, or suggestions for improvements. A google account is required.

TimingAnalyzer Google Group.

Contact Author

Feel free to email and make suggestions for new features and improvements, report problems, or discuss the future plans for the program.

Dan Fabrizio timinganalyzer@gmail.com

CHAPTER 12

License

The TimingAnalyzer is free to use by anyone without any limitations and is currently licensed as “Freeware” while beta testing. When the first final version 1.0 is released, the license will change to a “Commercial” license. The “Commercial” license will require businesses to purchase the program, but it will continue to be free for personal and academic use. The estimated cost of the program will be low and affordable for small companies.

During beta testing, no warranties or guarantees are provided by the author of the program. If you decide to use the program, you should be aware that it is a work in progress and issues occur. Issues reported are fixed as soon as possible. Many users have been building timing diagrams with the program and it is considered to be stable and reliable and ready for everyday use.