Analog Monitoring Tool

AMT 0.3b User Manual

1 Introduction

AMT (Analog Monitoring Tool) is a tool for checking the correctness of analog and mixed-signal simulation traces with respect to a formal specification expressed as an assertion.

The specification language supported by the tool is STL/PSL, an extension of the temporal logic inspired by the PSL language, which allows to express properties of real-valued continuous-time behaviors.

1.1 Installation

AMT is distributed as a binary executable, compiled for Debian/GNU Linux x86 machines. It requires the following libraries

- Qt 4.4 (http://www.trolltech.com)
- Qwt 5.1.1 (http://qwt.sourceforge.net/)

For the installation, the user needs to copy the binary file `amt` into the desired directory and execute `./amt`

2 Quick Start

Start the AMT tool with the command `./amt`

In the Property Edit frame, open the file `align.stl`

Add the property to the PropertyList frame

Import the property from the Property List frame to the Property Structure frame

From the Signal List frame, open the simulation dump file *align.raw*

Select and load all signals (*clk*, *nclk* and *data*)

Select the *offline* evaluation mode

Start monitoring the property

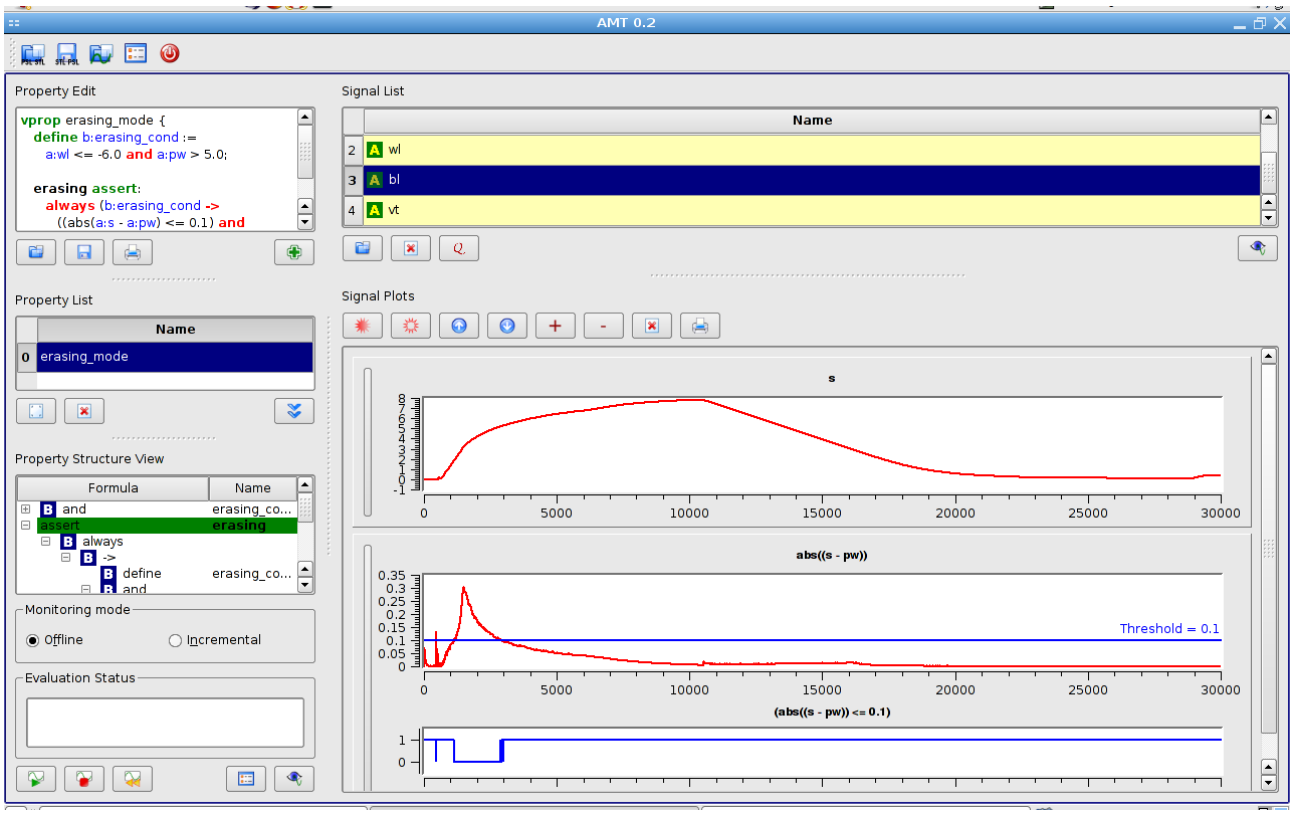
After the evaluation is done, you can select sub-formulae from the property parse tree, and visualize the results

3 Main Window

The main window of AMT is divided into Property and Signal sections, situated respectively on the left and right side of the application. The Property section contains frames for editing, managing and monitoring STL/PSL properties. The Signal section is composed of frames for loading input signals and visualizing (plotting) the results. All the frames can be resized and hidden by the user by moving the "splitters" that separate them.

There are five frames frames within the main window, as shown in Figure 1:

1. Property edit: text editor for writing STL/PSL properties
2. Property list: contains the list of the valid STL/PSL properties
3. Property evaluation: allows the user to monitor the correctness of an STL/PSL property
4. Signal list: contains the list of offline input signals
5. Signal plots: visualization of the signals



4 Property Management

4.1 Property Format

The syntax and semantic of the STL/PSL language supported by the tool is described in the file `stl-psl-language.pdf`. The following production rules are used to specify a valid STL/PSL property:

```
Stl_Psl_Vprop :==
vprop STRING { { Define } { Assert_Directive } }
Define :==
define Analog_Variable := Analog_Expression ;
| define Boolean_Variable := Stl_Psl_Property ;

Analog_Variable :== a:STRING
Boolean_Variable :== b:STRING

Assert_Directive :==
STRING assert : Stl_Psl_Property ;
/* See stl-psl-language.pdf for the list of Analog_Expression
   and Stl_Psl_Property operators                              */
```

Example of a valid STL/PSL property:

```
vprop example {
  define b:asig_cond := a:asig1 <= 3.5;

  first assert:
    always distance (a:asig1, a:asig2, 3.5);
  second assert:
    always (b:asig_cond ->
      eventually! [0:200] (a:asig2 > 5.5));
}
```

4.2 Property Edit Frame

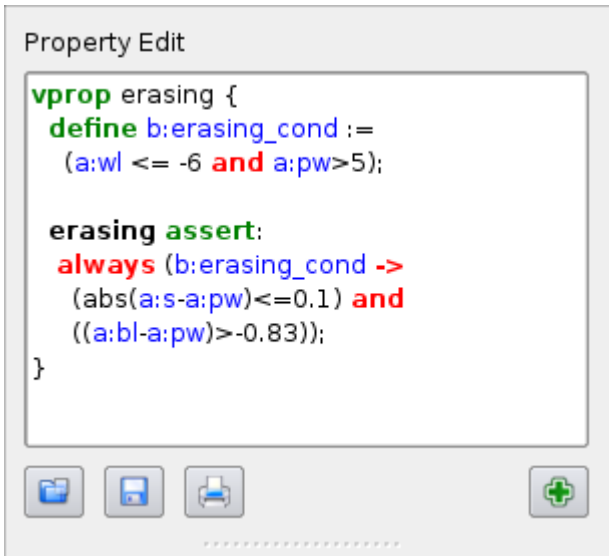


Illustration 1: Property Edit Frame

Property edit frame contains a simple editor for writing STL/PSL properties.

Options:

Open STL/PSL property from a file (*.stl)

Save STL/PSL property to a file

Print the current content of the editor

Transform the textual description of a (valid) property into its parse tree and add it to the Property List frame

4.3 Property List Frame

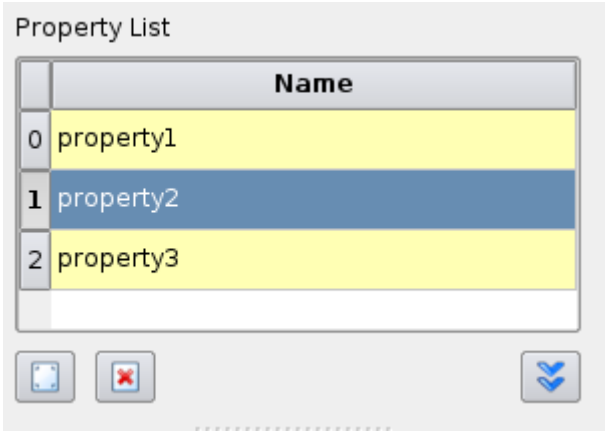


Illustration 2: Property Edit Frame

The Property List frame stores the list of STL/PSL property parse trees. The textual description of the property can be loaded back into the editor, deleted or set for evaluation.

Options:

Edit the selected property

Delete the selected property

Set the property for evaluation in the Property Structure View Frame

4.4 Property Structure View Frame

4.4.1 Property Evaluation

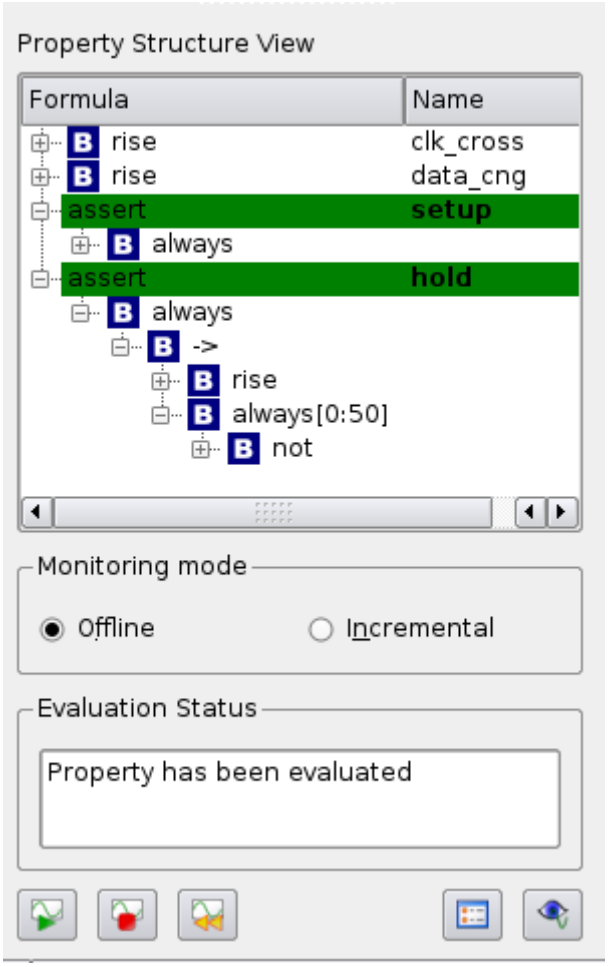


Illustration 3: Property structure view

The Property Evaluation frame allows the user to monitor the correctness of a set of assertions defined in an STL/PSL **vprop** verification unit with respect to the input traces. The frame shows the set of assertions in a tree view, following the parse structure of the formula. The user can choose between offline and incremental evaluation of the property. The **vprop** unit is shown in the evaluation frame in a form of a tree, where nodes correspond to different sub-formulae.
Each node contains three distinctive information:

- The icon showing the type of the sub-formula
  - Analog sub-formula A
  - Temporal or Boolean sub-formula B
- The name of the operator with its parameters, if any

- The name of the sub-formula, if any (only assertions and variables have a specific name).

The assertions are considered as particular nodes in the vprop unit parse tree, as they represent the top-level formulae. The user is interested in their correctness. We show visually whether an assertion within a vprop is true by highlighting it using the following color scheme:
- **Yellow:** Marks an undetermined assertion. An assertion is undetermined if it has not been evaluated yet, either because the monitoring procedure has not yet been started, some input variables defined in the assertion are missing or the **vprop** unit is in the process of evaluation in the incremental mode, and there has not been yet enough input to determine the correctness of the assertion.
- **Green:** The assertion is correct, ie. its evaluation signal is true at time 0.
- **Red:** The assertion is incorrect, ie. its evaluation signal is false at time 0.

The user has the option to conduct the evaluation of an STL/PSL property either *offline* or *incrementally*. In the offline monitoring case, the input signals are taken from the available signals in the Signal View Frame and the property is evaluated with respect to them. If one or more input signals are missing, the monitoring procedure tries to evaluate the **vprop** unit anyways, but without the guarantee to bring any conclusive result. However, in some cases a definite response can be given even if some inputs are missing. Consider as an example the assertion `always (b:p and b:q)`. If the signal p is false at some time, we don't need the input signal q to conclude that the assertion is false. The incremental procedure requires an online feed of input signals, through TCP/IP protocol. In the incremental mode there are three manners to end the evaluation:
- If all the assertions become determined, the monitor stops and closes the connection with the simulator
- If the termination packet is received from the simulator
- If the user presses the Stop Monitor button

Options:


Start monitoring. In the incremental evaluation case, it starts a server that listens for a new client (simulator) that will provide inputs (if the current **vprop** was already evaluated, the old results will be reset)


Stop monitoring (this button is effective only when in incremental mode)


Reset the monitoring results


Open the Configuration Dialog (see Section 4.4.2)


Plot the satisfaction signal of the selected sub-formula

4.4.2 Configuration Dialog

Some of the options available via the Configuration Dialog are:

- Port: the TCP communication port on which the tool will listen for an incoming connection
- Timout: in the incremental mode, the monitor receives inputs dynamically from the simulator and evaluates the formula on-the-fly. However, reevaluating the formula after each new input is highly inefficient. Instead, the tool waits for some time during which it accumulates new input, and reevaluates the formula every timout milliseconds. Note that small timeout may allow to detect errors with smaller delays. However, it can be inefficient on slower machines or when the simulator does not send new inputs fast enough.
- Keep history: The monitoring procedure separates each evaluation signal into two parts, the first part that has already been used by and is not relevent anymore to the parent formula, and the second part that is still needed by the parent formula in order to make its own updates. By checking this option, the user enforces the tool to keep in memory portions of signals that are not needed anymore by the monitoring procedure. This option may require a larger amount of RAM memory but is useful for understanding the reasons of an evaluation that succeeds/fails by providing the complete history of the monitoring execution.

Note that the configuration options can be saved and hence automatically reloaded the next time that the application is run.

5 Signal Management

The input signals in AMT can be imported either statically from a file or dynamically, using a simple protocol built on top of the TCP/IP.

5.1 Signal List Frame

Static (offline) import of signals is done via the Signal List frame. Several file formats are supported:

1. **out:** Nanosim simulation files
2. **vcd:** Value Change Dump files
3. **raw:** Berkeley Spice3 raw files (ASCII and binary)
4. CosmosScope ASCII output files

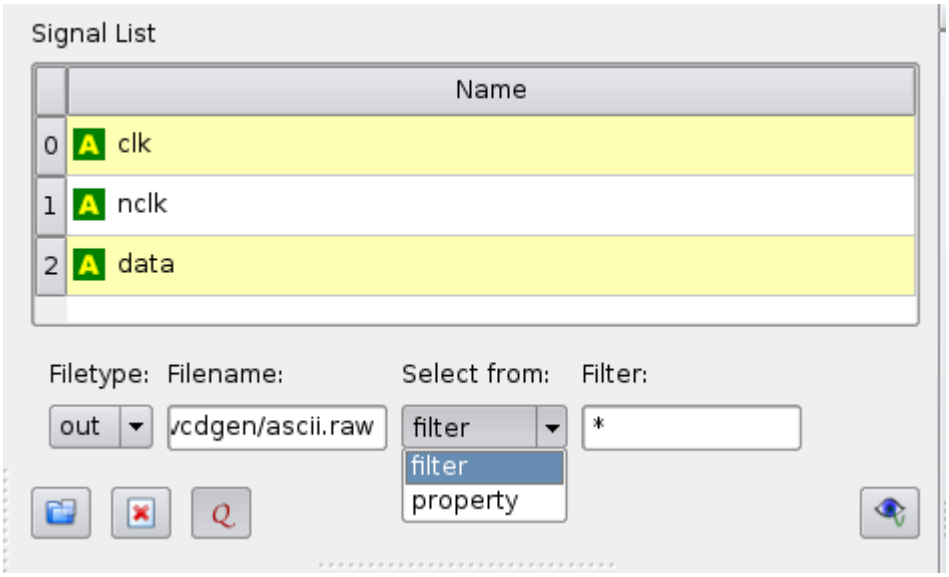


Illustration 4: Signal list frame

Note: Current and voltage signals are loaded from a Nanosim out file. Logical signals are ignored

Note: Real and wire var signals are loaded from a **vcd** file. All the other var types are ignored. The user can rescale the time axis of the signal by choosing the appropriate time unit when asked during the loading process. Wire signals are assumed to have only 0 and 1 values. X and Z values are ignored

Note: Input signal names have to match those used in the property. To change the signal name doubleclick on it and press Enter to validate the new name.

Options:


Open input signals from a file


Delete the selected signal


Plot the selected signal


QuickOpen button allows to specify directly the path of the file to open, and select either the signals that match the wildcard filter, or that match the variable names in the property that is currently loaded into the Property Structure View frame.

5.1.1 Signal Load Page

When the signal input file is opened in the standard way (not via the QuickOpen button), the Signal Load Page allows the user to see all the available signals within the file and select the ones to load. Selection is done either by mouse(`Click`, `Ctrl+Click` and `Shift+Click`) or by a wildcard filter. The user can see all the selected signals by enabling the View selection button. For some file formats, the user can choose the time scaling of the signals.

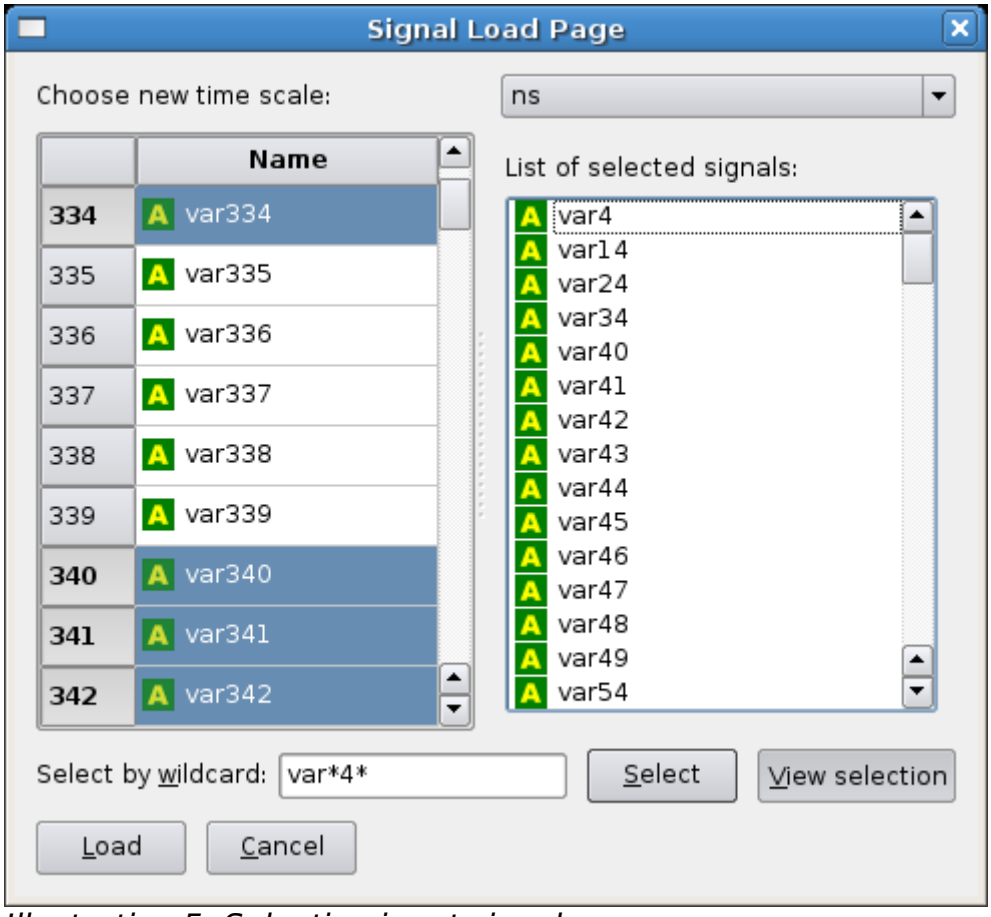


Illustration 5: Selecting input signals

5.2 Signal Plot Frame

Each operator in STL/PSL (Boolean, temporal or analog) contains an associated satisfaction signal (the signal that shows the truth value of the operator at any time). The signals corresponding to different operators are visualized as plots in the Signal Plots Frame. AMT provides four types of plots:

- Analog plots
- Boolean plots
- Threshold abstraction plots
- Distance plots

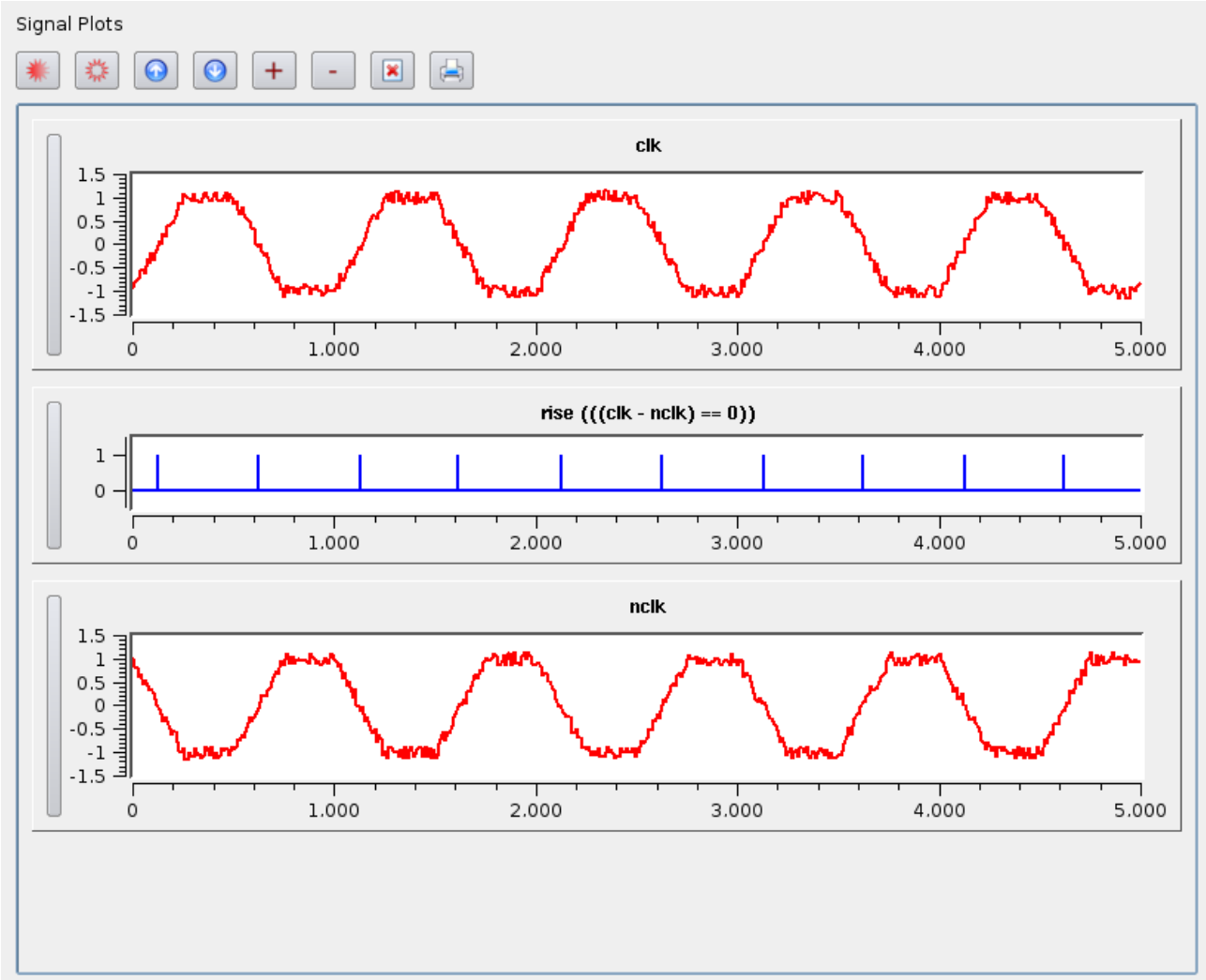


Illustration 6: Signal plots frame

Options:

To select/unselect a plot, click on the vertical barto the left of the plot.

Select All/Clear Selection buttons

Move Up/Down the selected plots

Remove the selected plots

Print all the selected plots (on a printer or to a PostScript file)

Additionally, the user can zoom, unzoom and shift the plots. To zoom in, hold the `Left Button` of the mouse and drag it over the region to zoom. To unzoom the plot to the previous zooming level, press `Ctrl+Middle Button`. To unzoom the plot to the original zooming level, press `Ctrl+Right Button`. To shift the plot, hold the `Middle Button` and drag the plot.

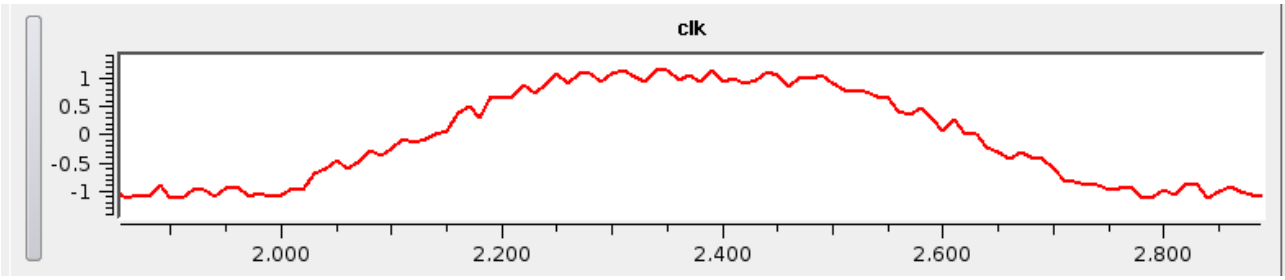


Illustration 7: Zoomed in signal

5.2.1 Incremental Signal Input

The input signals can be imported into AMT incrementally via a simple protocol built on top of the TCP/IP network. A simulator that provides the input signals needs to act as a client that connects to AMT when an STL/PSL property is set for incremental evaluation. For more details about the connection to the server, refer to the section 4.4. The client can send to the tool both Boolean and analog signals. They are encoded as a sequence of TCP/IP packets representing adjacent intervals in the format shown in the table below (it is up to the user to ensure the consistency of the data sent such as adjacency of intervals for subsequent Boolean packets). All packets have a header containing the size of the packet and its type which determines the type of subsequent data fields:

- **Analog Packet:** a sample of the analog signal, containing its name, time and the value
- **Boolean Packet:** a Boolean interval, contains the name of the corresponding signal, the end points, the type and the value of the interval
- **Termination Packet:** Informs the tool that the simulation is over and no further packets are sent

Boolean packet						
quint16 size remaining size	quint16 type 'b'	qreal begin	qreal end	quint16 value	quint16 itype	char *name
Analog packet						
quint16 size remaining size	quint16 type 'a'	qreal time	qreal value	char *name		
Termination packet						
quint16 size remaining size	quint16 type 't'					

Note: Boolean packet interval type itype

- 0: left-open right-open
- 1: left-open right-closed
- 2: left-closed right-open
- 3: left-closed right-closed

Note: quint16 = 2 bytes and qreal = 8 bytes

Note: Remaining size (quint16 size) = size in bytes that remain in the packet. Example for a Boolean packet with variable named "boolvar":

size = 3*sizeof(quint16) + 2*sizeof(qreal) + strlen("boolvar") = 6 + 16 + 7 = 29 bytes remaining in the packet.