FORMAL VERIFICATION SUPPORT FOR ELLA

IED 4/1/1357

Requirements Specification for Soft Logic Analyser *

Version 0.3

DRAFT

Howard Barringer Graham Gough Brian Monahan† Alan Williams ‡

Department of Computer Science
University of Manchester§
Oxford Road, Manchester, UK.
email: howard@cs.man.ac.uk

February 1993

†Brian Monahan is now with Harlequin Ltd., Alderley Edge

‡Alan Williams acknowledges additional support from DRA (Malvern) and Harlequin Ltd (Cambridge)

§The University of Manchester acknowledges support from the SERC via research grant GR/F 38174.

1 Introduction

The phrase 'soft logic analyser' (SLA) was coined during a discussion in an IED/ELLA project meeting[Mtg92a]. It was envisaged to be based around the design simulator, with enhanced functionality for signal input, output and monitoring, and a graphics user interface. There would be hooks between it and other tools already being developed for the ELLA environment, such as schematics and text editors. A separate examination of the requirements was deemed necessary due to the extension of tools in the environment to handle the full ELLA typing mechanism.

The purpose of this report is to expand the requirements already identified, so that the SLA can be developed, together with the consequent extensions to existing tools, such as the ELLA compiler, schematics/text editors, and the simulation/comparison tools. High level verification requirements in the project may also influence the design of the SLA. The project clearly imposes time and resource constraints on the development of the SLA, so the report concentrates on those requirements that can be realistically achieved. However, the impact of possible future extensions to the functionality of the SLA must be examined during its design.

It has been observed that some requirements are really just 'productisations' and do not enhance the core functionality of the proposed SLA tool. Although this report will attempt to identify core features that are required, it will also include suggestions concerning these more peripheral aspects. Suggestions concerning various aspects of the design of the SLA are also given.

The report begins by giving a summary of the original discussion on the SLA, and then outlines the present state of simulation tools in the environment. A broad description of the 'coarse' functionality of the SLA is given, followed by the 'fine' aspects concerning input generation, simulation, output capture/analysis, and data presentation. The impact of the SLA requirements on existing external tools will be examined.

1.1 Background

In the meeting[Mtg92a], the proposed ELLA environment was examined from the viewpoint of the designer — what does a designer want to see, whilst analysing and debugging a design? The following requirements were drawn up:

i. rule-based monitoring of signals,

ii. pattern matching to recognise signal patterns,

iii. backtracking, i.e., running a simulation backwards in time, perhaps from an identified error point,

iv. waveform input.

It should also be possible to link the SLA with schematics and text editors, so that any signal referenced could be identified with the original ELLA schematic/text.

Current experience of related tools in the project includes the design and use of the prototype ELLA environment, and of DHDW[BBE+92]. The author has used a Tektronix hardware digital analysis system. A brief examination of CAD tools for VHDL has also been possible. It is hoped that these and similar tools such as ELLAVIEW, can be tested further to assist in the design of the SLA.

1.2 Outline of Present Tools

It is useful to outline the present state of the relevant tools in the ELLA environment, to indicate the position from which the development of the SLA will begin.

The existing prototype environment, demonstrated at the end of Year 2 meeting[Mtg92b], provides a waveform display panel to control simulation and comparator tools. A new panel is created by selecting a menu option available in the library browser, schematics/text editors or comparator. When a simulation or comparison is requested, any recompilation of altered designs is performed silently. The comparator provides a set of counter-example traces, if any exist, extracted from the difference automaton generated during design comparison. The simulator operates on the IOA representing the design, simply by exercising the various input/output traces which it defines — initially all inputs are set low. The following facilities are then offered:

i. All input/outputs of the IOA are displayed — no selection of signals is possible.

ii. Each waveform in the display is identified by the signal-name, given in the IOA description.

iii. The user can specify the length of simulation in terms of the number of time-steps required.

iv. Inputs can be set to be all low or all high, and the simulator is then rerun. Individual values can then be changed by clicking on the appropriate part of the input trace. The simulator is then rerun to update the output waveforms in the display.

v. The window can be resized.

vi. Scrolling and panning of the window is possible.

vii. An information line gives the name of the design under simulation, and the current simulation length.

viii. Comparison Display:

- returns segment of trace up to the point where the behaviours of the two design differ,
- highlights outputs that differ,
- scrolls through sequence of difference traces provided by comparator,
- displays the names of the two designs under comparison.

The environment can incorporate other tools such as those developed in the related DHDW project at Manchester[BBE+92]. The environment also has a schematics editor, being developed in ClassWorks, which will be able to support useful annotations such as animation or highlighting of nodes.

Designs can only use boolean types in this prototype, and so any problems with visual representation of simulation traces for more complex data types have not yet been encountered or addressed.

From brief initial use of the prototype tools, the following observations have been made:

i. tools limited to boolean values,

ii. no indication of when output is out of date, after input has been edited on-screen,

iii. no cursors or grid,

iv. limited highlighting facility (for comparator only),

v. not scalable/no zoom,

vi. no selection of inputs/outputs,

vii. input always initialised to either $(0,..,0)$ or $(1,..,1)$, and changed manually on-screen (although this is a desirable option to have),

viii. signal-names are lost off the screen after a pan,

ix. time-step scale is lost off the screen after a scroll,

x. no selection of comparator error-traces displayed possible.

1.3 Functional Role of SLA

The SLA will be used to generate suitable input traces, and to drive a simulation. It analyses results of the simulation or comparison/verification, especially for debugging purposes. It can be used to explore design characteristics and properties, perhaps using external analysis tools. The SLA will therefore assist in tackling a major obstacle in design analysis — the difficulty of generating the large amounts of test input data required to drive design simulation and then extracting pertinent segments of the resulting output data, for analysis by the user. Information must then be displayed in a clear manner via the user interface — this may involve interaction with other external tools such as schematics/text editors. Apart from analysing values on the input/outputs defined by the design description, the SLA enables internal nodes to be examined, or to be set to particular values, by declaring monitors and probes.

2 SLA Requirements

The SLA is composed of a facility for displaying traces, external tools for input generation and output analysis, additional functionality in other existing tools in the environment, and interfaces between these components. The existing tools include the simulator, comparator, schematics/text editors, and compiler.

One method for calling the SLA from the environment will be via menu options in the editors or browser, replacing the present options which invoke simulation or comparison; these analysis tools will be run from the SLA. For example, the current waveform displays for the simulator and comparator will be replaced by the SLA display discussed below2.4.

Figure 1 shows the components of an SLA. The double box represents the interface between the various parts of the SLA, indicated by the single solid boxes. The dotted boxes indicate facilities provided by existing tools in the environment.

The overall design of the SLA should be capable of accommodating changes in the external tools with which it must interact, and in the representations of the various objects involved. For example, in the present environment, the semantic objects used are explicitly enumerated IOAs — for convenience throughout this report, a semantic object has still been assumed to be an IOA. However in future systems, even if IOAs are used, then their representation is likely to change; current work at Manchester is concerned with the development of alternative methods of representation, for example using algebraic expressions. It may indeed be the case that different semantic objects altogether are used.

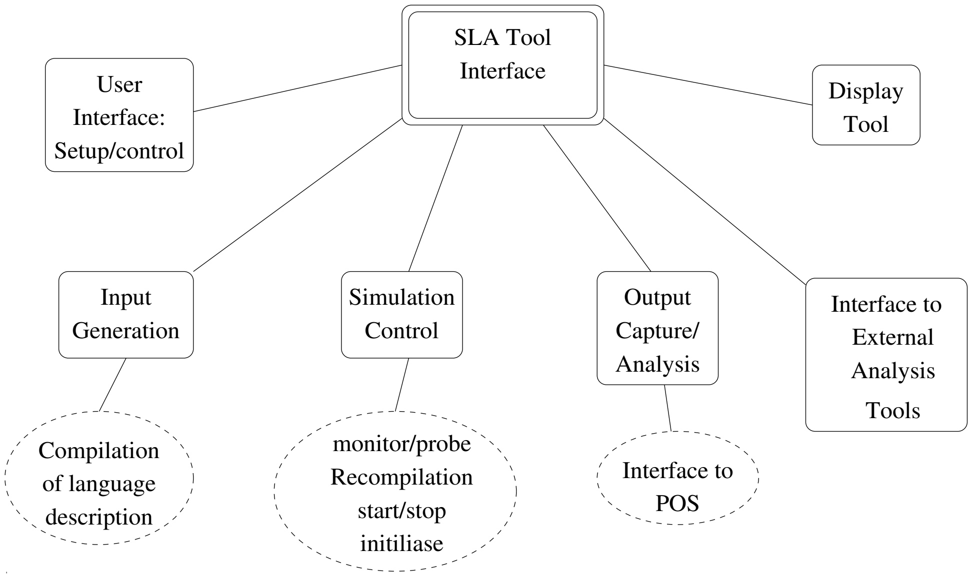

Figure 1: Proposed Components of SLA

In all areas, it is important to be able to perform operations in terms of the original text and to relate results back to it — this requirement is emphasised in [Hil92].

The SLA can be set up and driven from external software or command files, so that it performs automatic analysis as a background process. This is useful if repetitive or long tests are required. If an operation is aborted, then as far as possible, some meaningful diagnostics should be provided, e.g., current state in simulation, signal values, time elapsed. As with other tools, the SLA should be equipped with on-line help facilities, session recording, option display, and so on.

As indicated above, the requirements of the SLA can be grouped into a number of 'coarse' areas. The following sections give 'fine' details for each area.

2.1 Input

The aim of the input generation tool is to allow the user to easily describe the input traces to be applied to the design representation, either using an explicit description, or by defining the characteristics of the input traces. In some circumstances then randomly generated input may be the most suitable. For a set of input signals then a combination of these methods may be useful, e.g., explicitly describe one component, give the characteristics of a second, and have random values for a third.

A common method used for input description is simply an extension of the existing system, where the user explicitly constructs the input stream. This can of course be performed manually or automatically. An input description language or even a test harness written in the HDL could be used as shorthands for the description. A second method is to describe the *characteristics* of the input, using perhaps a specification language related to temporal logic. The SLA should provide facilities covering both methods.

With ELLA, there is a simple universal timing model where the unit of time is a single simulation step — each input vector corresponds to one step in the simulation. A more flexible approach would allow the input data and simulation steps to be independent. This may be necessary when more complex timing models are required. However, a simple retiming mechanism for the input data stream would be useful so that for example, a single input vector could cover a number of simulation steps.

The following is a list of suggestions for input generation;

i. random input generation (e.g., *simulate* tool in DHDW), default initialisation as in current simulator,

ii. manual input via user interface (current), button pressing and further elaborations (e.g., set values of individual signals),

iii. user-generated input vectors, from ASCII files stored in the Persistent Object Store (POS).

iv. input specification language (e.g., regular expressions, imperative language, temporal logic-based language, 'waveform' specification language)

v. user-specified (temporal) constraints for automatically generated input, e.g., *paths* tool[BBE$^+$92] generates the set of uni-repeating paths for an IOA which satisfy a temporal property. With the *paths* tool, constraints can be applied to both the input and output.

vi. test harness, written in HDL (used in tools for VHDL); in this case the SLA does not need to generate an input, but simply to compose the IOAs for the test harness and the design. Alternatively the SLA could read input traces from the test harness IOA and apply them to the design IOA (see (vii)).

vii. language represented by an IOA.

viii. external software input generation.

ix. apply transformations to data, or introduce retiming.

2.2 Simulation

The aim of a simulation controller is to enable the shortest simulation that exercises the desired design components, and to allow the designer to monitor and modify the course of a simulation. This is to be used in conjunction with careful selection of input.

The current simulator operates on the semantic object representing a design. It first checks that the design IOA is up to date — if it is not, then it instigates a recompilation. It then simply applies the supplied input trace to the IOA and produces the corresponding output trace. The proposed simulator will also operate on the semantic object, and perform silent recompilation if necessary — the user should have control over this, since the operations will take a long time if the design to be compiled and simulated is large. For non-deterministic automata a random choice between paths would be required, with an indication to the user that a choice has been made — the user should be able to specify the trace to be taken.

Other options involve restricting, at the simulation stage, the permissible input/output traces exercised, by using temporal constraints, e.g., *paths*[BBE$^+$92]. Again this could also be performed by applying filters to the input/output of the usual simulation.

The initial state for simulation should be specifiable using the different representations of the design (e.g., schematic, text, IOA). The simulator should be able to be run 'backwards' from a specified state, which may be an error point.

A simulation run should be controllable by specification of the length of a run, user interrupt, or by setting break points (the control mechanisms for these will also be used for capturing output). After an interrupt, the user will be able to interrogate the current simulation state, and perhaps alter it for continuation of the run. Again, alterations can be defined using the different design representations. The simulation state can be saved, perhaps for a later run. Clearly these features can be produced using a small set of simulator control operations, in conjunction with input generation.

Another important feature, required in [Hil92] is the introduction of test-points — monitors and probes — for forcing the value on internal nodes and for reading their value, respectively. Once a probe has been declared then its values are set during input generation. Once a monitor has been declared, then its values are processed during output capture. In the current environment, the addition of probes or monitors requires a recompilation of the design, since the output signature is altered by adding monitors, and the connectivity of the design is altered by adding probes. One possible method for adding monitors was sketched in [BGM+92]. The actual mechanism for introducing probes and monitors into the design IOA will depend on the exact representation of semantic object used.

A summary of features required during simulation follows:

i. basic simulation controls:
   - start, pause/stop, continue, step,
   - set initial state and break points,
   - backtracking,
   - specify length of run in terms of (a) simulation time/steps, (b) cpu time.

ii. apply generated inputs (if any) to semantic object (IOA), collect output for processing by analyser stage (use internal data structure/pipe, or a set of circular buffers to hold output data),

iii. keep a simulation clock counter

iv. save simulation state on pause/stop; allow change of state/simulation settings,

v. add monitors and probes; perform recompilation when necessary (see [Hil92], p5):
   - only monitor specified inputs/outputs,
   - define test-points at different levels in the design,
   - set/display state of test-points: for example, in ELLA a designer can show current value, action, condition (e.g., ELLA allows actions: (*monitor, display, stop, keep*) and conditions (*always, changes, becomes, equals*) — see [ELL91]).

vi. indicate random/non-deterministic choice (compare with discussion of simulation/semantics relationship in [BGMW92]); allow user to choose a different value when non-deterministic,

vii. only generate output that satisfies constraint.

2.3 Output

The aim of output capture is to cut down on the quantity of data stored from a simulation, by selecting only that data which is going to be useful to the designer. In a realistically-sized design then it is very likely that simulations will take a significant amount of time, and produce large amounts of raw data. Additional facilities are required so that only data which will be useful to the design task at hand is produced and saved. For example, selection of monitored outputs and retention of only a window of data are two obvious candidates for reducing storage needs. Use will be made of the POS to store information, and to maintain its relationship with other design objects.

Another approach is to examine characteristics of the output traces generated, and indicate only when certain are met by the data. If this is possible, then this certainly fulfils the above criteria of cutting down data storage requirements and returning only that information which is useful to the designer. Both methods should be included in the SLA.

The output capture tool will interact with other tools in the environment, e.g., by pointing to a wire in the schematics editor, the corresponding node is selected for monitoring.

As before, a list of useful facilities is now given:

i. store a window of simulator input/output vectors,

ii. retain output data for future analysis, e.g., comparison with further simulation runs,

iii. only store information about selected nodes,

iv. trigger mechanism: only begin data storage when trace satisfies a constraint, e.g., a trigger word has been received; only continue data storage while trace satisfies constraint — perhaps a series of data windows is stored in this way,

v. sampling/clock: time-dependent version of above,

vi. examine properties of design: temporal properties, stability/hazards, deadlock, e.g., *paths* tool,

vii. apply transformations to data, retiming (see §2.1),

viii. data comparison (e.g., with previous run) — display differences,

ix. assertion outputs generation: if input/output satisfies condition then execute declared action, e.g., print statement,

x. supply data to external tools.

2.4 Display

The challenge in constructing the SLA display is to compactly and clearly display a large range of data objects, whilst highlighting features that are of interest to the user. Of course, by careful application of the preceding tools for input, simulation and output capture, then the task of display will be simplified.

An important aspect of the display is that it is flexible and easily configurable — this is related to *object presentation*[Arc91]. For example, the user should be able to specify maps between screen symbols and data values, and impose a particular interpretation on colours. Configurations can be set via the window and menu options, saved and then reloaded.

i. waveform-style display (an enhancement of the current version, extended to types) with graphic encoding of data types:

- decompose/expand/hide trace segments for complex data types
- cursors and grid to negotiate window, and assist designer in interpreting wave-forms, e.g., multiple cursors, display data-values/time-step under cursor, show data/time differences between two cursors, move window relative to cursor
- user-specified map between data type and display symbols
- signal selection and reordering, for display purposes
- multiple windows
- zoom/pan, magnification windows
- use of colour (e.g., VHDL tools use it for signal 'strength')
- highlighting signal changes,
- highlighting points of difference after comparison
- hard-copy output (snapshot of display)

ii. alternative display styles:

- textual display of data (e.g., used in EASE (ELLA Application Support Environment)[ELL91])
- display results of data analysis, e.g., traces that satisfy/violate conditions,
- *diff*-style display, from comparison
- statistics: e.g., numbers of paths satisfying temporal property; counts of paths, etc.,

iii. 'animation' of text, schematic or IOA representation of design,

2.5 User Interface: Setup and Control

The setup and control may be seen to be distinct from the display tool described above. It provides the menus and records necessary to configure and control the SLA manually as well as responding to information from setup files or external tools. It will provide the user with details about various aspects of configuration as requested. For example, the user may wish to monitor the status of a design node, its present value, whether it is currently monitored/probed, and so on. This request may be made from either the setup and control panel for the SLA, or it may come from the schematic editor in response to a user gesture.

The controller will be used to invoke analysis by the various tools available in the environment — currently these include the simulator and comparator. The display will be comprised of the usual hierarchical tree of menus, and windows displaying further buttons and records for the user to complete, setting up and controlling the required tool. Some degree of checking of the information entered by the user would be desirable.

2.6 Interface to Existing and External Tools

As will have become apparent during this report, the SLA will require extensive interfacing between other tools in the environment, and additional external analysis tools. For example, it will need to be able to fully control the operation of the simulator as outlined in §2.2. If the task of control of recompilation prior to simulation is assigned to the SLA, then it will need to access the compiler, so that a new semantic object can be produced for simulation. The SLA will need to accept input from the editors so that it can modify the status of monitors and probes. Results presentation may also involve animation of the design schematic or text. Finally, the POS will be used to store and access results and information from the SLA.

i. control simulation, IOA compilation; process results produced

ii. receive information from schematic/text editors about selection of monitors and probes

iii. animate schematic/text

iv. correlate states in IOA with values held by registers in design schematic/text (`DELAY, SAMPLE, RAM`)

v. correlate inputs/outputs in IOA with node names in (hierarchical) design schematic/text

vi. store/recover information from POS

It is also possible that the environment will interact with existing design tools, such as EASE or ELLAVIEW[ELL91]. In this case, the SLA will need to exchange information with the simulation and analysis tools in these systems.

3 Future Work

A set of requirements for the SLA has been proposed, together with suggestions for design. The next step in the development of the SLA is for the project partners to review this report and to provide feedback:

i. are these requirements desirable, and should they be graded into an order of importance, for implementation purposes?

ii. are there any requirements which have been omitted?

iii. can the requirements be implemented, or are there major technical barriers which cannot be overcome in the lifetime of the project?

The features identified spread across a range of the existing tools in the environment, and also require additional tools to be developed. An assignment of the various design and implementation tasks will have to be agreed among the project partners.

The following gives an outline of modifications and additional features for the existing tools in the environment:

i. ELLA-text, Semantics, Compiler and IOA:

  - devise notation for indicating test-points in ELLA-text
  - modify semantics and compiler to accommodate monitors and probes
  - relate states in IOA with 'states' in ELLA-text

ii. Schematic/text Editors:

- animation: display values on monitored/probed nodes and inputs/outputs — how to display complex data types
- mechanism for editing monitors/probes in (hierarchical) design

iii. Simulator:

- acts on semantic object
- now invoked from SLA controller
- specify initial state
- specify monitored nodes (subset of output signature)
- input is supplied by SLA
- run for *n* steps
- run forwards/backwards
- indicate if path choice is non-deterministic — user should have control over chosen path
- return output trace and final state
- current waveform display is replaced by SLA display

iv. Comparator:

- from the SLA viewpoint, the comparator is an external tool to be controlled, and effectively simply one option for supplying input to the simulator (via the counter-example IOA, or its set of traces)
- uses facilities provided by simulator
- now invoked from SLA controller
- user control over order in which traces are displayed
- indicate where traces differ

4 Acknowledgements

The author wishes to thank Andy Carpenter (University of Manchester) for a useful discussion about VHDL CAD tools.

References

[Arc91] M Arcus. The ELLA Environment: Discussion Document. Harlequin Ltd, 7th January 1991.

[BBE+92] B Bani-Eqbal, H Barringer, D Edwards, T Longshaw, B Mitchell, B Monahan, and M Peim. DHDW Project Final Report. Technical report, University of Manchester, October 1992.

[BGM+92] H Barringer, G Gough, B Monahan, M Peim, and A Williams. ELLA Verification and Semantics Report - Year 2. Technical Report D2.2, University of Manchester, May 1992.

[BGMW92] H Barringer, G Gough, B Monahan, and A Williams. A semantics for Core ELLA. Technical Report D2.3b, University of Manchester, November 9th 1992.

[ELL91] Computer General Electronic Design Ltd. *ELLA Manuals, Issue 5.1*, 1991.

[Hil92] M Hill. High Level Verification Requirements. Technical Report D1.3c, DRA, Malvern, November 1992.

[Mtg92a] Minutes of Project Technical Meeting, 3rd April 1992.

[Mtg92b] Minutes of 7th Technical Meeting, 15th May 1992.