Concurrent Specification And Timing Analysis of Digital Hardware using SDL

Kenneth J. Turner, F. Javier Argul-Marin and Stephen D. Laing

Computing Science and Mathematics, University of Stirling
kjt@cs.stir.ac.uk, javargul@bbvnet.com, stephenl@reading.sgi.com

Abstract. Digital hardware is treated as a collection of interacting parallel components. The ANISEED method (Analysis In SDL Enhancing Electronic Design) uses SDL (Specification and Description Language) to specify and analyse timing characteristics of hardware designs. A library contains specifications of typical components in single/multi-bit and untimed/timed forms. Timing may be specified at an abstract, behavioural or structural level. Consistency of temporal and functional aspects may be assessed between designs at different levels of detail. Timing characteristics of a design may also be inferred from validator traces.

1 Introduction

Digital hardware can be viewed as a concurrent system whose components operate in parallel but synchronised via exchange of electrical signals. Although SDL (Specification and Description Language [9]) was developed for specifying communications systems, it is a general-purpose language of wide applicability. It is the contention of this paper that SDL is appropriate and useful for specifying and analysing digital hardware as collections of interacting parallel components. The approach particularly focuses on timing aspects, which are often tricky in hardware design. SDL is of interest to hardware specifiers because it offers rigorous specification, good system structuring features, high-level communication, and the possibility of hardware-software co-design.

Most uses of SDL for hardware description have aimed at synthesis using standard engineering tools (e.g. [2, 5, 6]). The authors are engaged in the project ANISEED (Analysis In SDL Enhancing Electronic Design [1, 4]). Its goals are complementary to those of others who have used SDL for hardware description. Specifically, translation to VHDL and/or C is assumed to be dealt with by other tools. Instead, the authors have concentrated on timing aspects of hardware specification and analysis. Timing constraints on circuits and components can be specified and analysed at various levels: *abstract* (overall sequencing constraints), *behavioural* (black-box viewpoint), and *structural* (internal design).

2 Approach

The behaviour of hardware components and their timing characteristics can be modelled naturally using SDL processes, since these run in parallel and communicate via signals. For timing analysis, ANISEED can use a standard SDL simulator or validator. However, the authors have also implemented a discrete event simulation by automatically modifying the scheduling strategy of a standard SDL simulator.

A hardware signal is modelled as an SDL signal with parameters:

Time-stamp optionally records the time at which a signal is considered to have been generated. This is necessary partly because an output signal may not be consumed immediately; it still, however, carries the time of its generation.

Value is mandatory and may simply be a bit. However, in general it may be multi-bit (i.e. a list of bits). This is appropriate at a high level of specification where a bus or group of wires is to be specified as a whole. In a very abstract specification it might even be desirable to carry arbitrary values such as data structures.

ANISEED allows single-bit/multi-bit and untimed/timed specifications. Library components are available in all four bit/time combinations. It is useful to write an untimed specification first in order to check the functional correctness of a design. Timing constraints can then be added (by a change of library component names), allowing timing issues such as race conditions and hazards to be studied.

ANISEED is supported by a library of common components and circuits (i.e. designs consisting of a number of components.) These are stored in SDL packages, forming a modular and easily extended library. Some of the packages depend on others (for example, all packages use bits). A type of component is modelled as a block type in SDL. The motivation for choosing block types rather than process types is mainly that the internal construction of a component should be invisible. A further consideration is that library block types are instantiated statically. A block type is parameterised by its signal names and its gates (in the SDL sense). Timed components are also parameterised by characteristics such as their propagation delay or setup time.

Components are interconnected by no-delay channels. Like real wires these are considered to convey signals instantaneously. If it is necessary to model the propagation delay of a wire, a delay component can be used. Where multi-bit components are interconnected with single-bit components, a split 'component' is used to separate the bits. Correspondingly a merge 'component' is used to combine single-bit signals into a multi-bit signal.

It would have been possible to specify all the library components individually. However this would have been very tedious. As a more pragmatic solution, all variants are generated automatically from an SDL template that is parameterised by the logic function, the number of inputs, whether timed and whether multi-bit. The template is an outline PR (SDL Phrasal Representation) specification that is pre-processed to yield the required variants. The approach using templates makes the library much smaller (10%–15% in size compared to the generated PR) and more maintainable.

The current ANISEED library contains over 400 standard components such as might be found in a typical logic family. The library includes packages for arithmetic units, bit values, decoders and encoders, demultiplexers and multiplexers, flip-flops and latches, logic gates, sequencing constraints, and tri-state devices. Explanatory comments are automatically generated by the templates. Since GR (SDL Graphical Representation) is often preferred by SDL specifiers, the library templates can also automatically generate comments in the style of CIF (SDL Common Interchange Format).

The authors use version 3.5 of the TAU/SDT toolset. This has some restrictions that affect its suitability for ANISEED. The library modules generate two PR variants: one has axioms, and the other has C code for simulation/validation. A more severe problem is that commercial SDL tools (SDT, ObjectGeode) do not currently support context

parameters fully. These are essential for block types since the actual timing parameters and signal names are not known until a block type is instantiated in a particular context. To overcome tool limitations, ANISEED automatically instantiates context parameters in the PR generated from a graphical description.

3 Validation and Verification

SDL tends to be used in pragmatic ways, so validation is usually the method of choice. What would normally be termed verification (proof, model-checking) is comparatively rare for SDL [3, 7]. The SDL community uses the term validation to mean automated checking of an SDL specification. A specification may be validated in isolation; such a check is useful but does not confirm functional correctness. Alternatively a specification may be validated against an MSC (Message Sequence Chart [8]). This may be written by the specifier, derived from a validation run, or generated automatically from a higher-level specification. An MSC can be used to confirm correct refinement of a specification.

Most SDL validation is oriented towards checking functional correctness. However, SDL allows timing aspects to be specified and validated. In ANISEED, interactive simulation can be used to check for the expected timing behaviour. However, automated validation is preferred using exhaustive analysis or random-walk validation.

For hierarchical timing specifications, the validator is useful in checking consistency between different design levels. For example the abstract and behavioural specifications may be compared, or the behavioural and structural specifications. The MSC traces produced by the validator are useful in deriving timing characteristics. When many components are interconnected, a range of high-level timing properties for a circuit can be derived from the validator output.

During SDL simulation, the signal with the earliest time-stamp must be consumed first, even if other signals have been placed before it in the input queue of a process. The SDT Master Library was modified to achieve this effect, with ANISEED supplying its own scheduling functions for discrete event simulation.

4 Abstract Sequencing Constraints

Constraints at the highest level may be given without regard to functionality. This defines gross sequencing relationships among the inputs and outputs of a component. The constraints may be given in untimed form, but are most useful when used to express timing restrictions. It is valuable to check for timing inconsistencies before any more detailed functional design is undertaken. Once high-level sequencing properties have been validated, their specifications are replaced by those of the real behaviour.

The following examples of sequencing constraints are drawn from the ANISEED library; the constraints exist in untimed and timed forms. An *N-Of* constraint requires an input event to occur *N* times before output occurs. As an example without a timing constraint, a divide-by-4 counter produces one output pulse for every four input pulses. A period during which counting occurs may optionally be given. A *One-Of* constraint accepts just one input before producing output. For example, a bus arbiter must service just one client request during a bus cycle of some period. A second input within this

period is retained until the next cycle. A variant of this constraint discards additional inputs before the period has elapsed. An *All-Of* constraint requires all inputs of a component to be received before output is produced. For example, the inputs to an adder must be received before its output can be calculated. The order in which inputs arrive is unimportant, but all inputs may be required in some period. Unless all the inputs arrive in time, the whole constraint is re-enforced.

5 A Typical Component: A Delay Flip-Flop

As an example of hierarchical specification, a DFF (Delay or D Flip-Flop) is the basic storage element in many hardware designs. The conventional symbol for a delay flip-flop is shown in figure 1 (a). The variant to be described here is positive edge-triggered, which means that the data input *D* is stored when the clock *C* goes from 0 to 1. After the data has been clocked in, it appears at the output *Q* after some propagation delay that depends on the hardware family. Apart from the obvious propagation delay *TProp*, a D flip-flop also imposes two other timing constraints. It is required that the data input be steady for a period *TSetup* before it can be clocked in. Immediately after a clock trigger, the data input must remain steady for a period *THold*. The timing constraints are shown graphically in figure 1 (b).

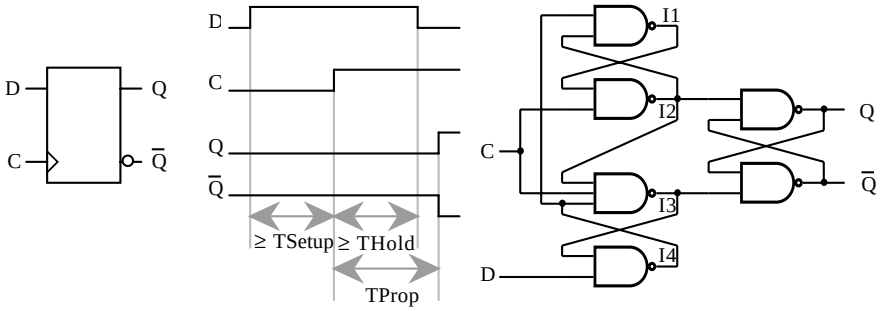

Fig. 1. D Flip-Flop

The timing rules of figure 1 (b) can be readily represented in an abstract SDL specification. Once the flip-flop has committed to output (i.e. after setup and hold periods), a separate process instance is created to produce output after the propagation delay. During this period it is necessary to allow for further inputs in parallel with the output delay. The flip-flop therefore consists of a single input process instance plus output process instances as required.

In hardware description terminology, a behavioural specification treats a circuit or component as a black box. Since the D flip-flop has very little functionality, timing considerations dominate its specification. The behavioural specification of the flip-flop thus differs little from the abstract one. The main addition for the flip-flop behavioural specification is how to calculate the new output value.

A structural specification concerns the internal design of a component. Structural specifications may form a hierarchy of designs at progressive levels of detail. At each level of the design hierarchy, ANISEED may be used to specify both timing and functionality.

As an example, a typical design for a D flip-flop is shown in figure 1 (c). The internal signals *I1* to *I4* are shown. This circuit uses a number of *nand* gates (the D-shaped symbols); other flip-flop designs are possible. The SDL equivalent of the flip-flop design is shown in figure 2. This is closely related to the circuit diagram in figure 1 (c). Since SDL allows channel details to be inferred from block inputs and outputs, these do not strictly need to be drawn and hence are shown as gray in figure 2. For convenience the instantiations of each block type are listed separately in the figure. The context parameters of a *nand* gate are *delays*, *inputs*, *output*. For a junction the parameters are: *input*, *outputs*. Primed signals refer to the outputs of a junction (e.g. output I' would correspond to input I).

N1 : Nand2T < CMOSDel1, CMOSDel0, I4`I2`, I1>
N2 : Nand2T < CMOSDel1, CMOSDel0, I1, C`, I2>
N3 : Nand3T < CMOSDel1, CMOSDel0, I2`, C`, I4`, I3>
N4 : Nand2T < CMOSDel1, CMOSDel0, I3`, D, I4>
N5 : Nand2T < CMOSDel1, CMOSDel0, I2`, I6`, I5>
N6 : Nand2T < CMOSDel1, CMOSDel0, I5`, I3`, I6>

J1 : Junction2T <C, C`, C`>
J2 : Junction3T <I2, I2`, I2`, I2`>
J3 : Junction2T <I3, I3`,I3`>
J4: Junction2T <I4, I4`, I4`>
J5 : Junction2T <I5, I5`, Q>
J6 : Junction2T <I6, I6`, QBar>

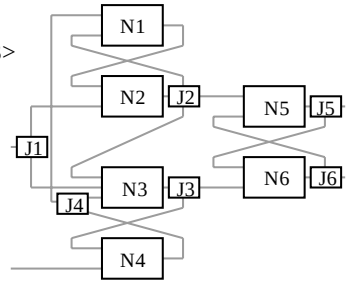

Fig. 2. SDL Specification of D Flip-Flop Design

Like each of the ANISEED library components, the D flip-flop was simulated and validated using the SDT toolset. Using the validator it was shown that the different levels of abstraction for the D flip-flop are equivalent in the sense that they respect the same traces. Exhaustive validation of the gate-level specification takes about one minute and 412 states. Random-walk validation of the gate-level specification takes about two minutes to explore over 211,000 states.

6 A Simple Circuit: The Single Pulser

The Single Pulser is a standard hardware verification benchmark circuit [10]. It is a clocked device with a one-bit input and a one-bit output. The purpose of the circuit is to debounce a push-button. The circuit must sense the depression of the button and assert an output signal for one clock pulse. The circuit does not allow further output until the operator has released the button.

The circuit specification was interactively simulated and mechanically checked. An example of the circuit behaviour appears in figure 3, generated automatically from a

validator trace (MSC). The time base corresponds to 100 ns per clock cycle. Although the functionality is correct, it was found that there is a flaw in the supposedly proven benchmark circuit! The first output pulse after power-up is longer than expected (110 ns, from 67 ns to 177 ns) instead of lasting one clock cycle. The reason for this is that on the first pulse, one of the flip-flops in the design does not need to complete its setup time. This causes the first output pulse to appear 10 ns early. On subsequent pulses this flip-flop allow for the setup delay so the output pulse length is correct at 100 ns.

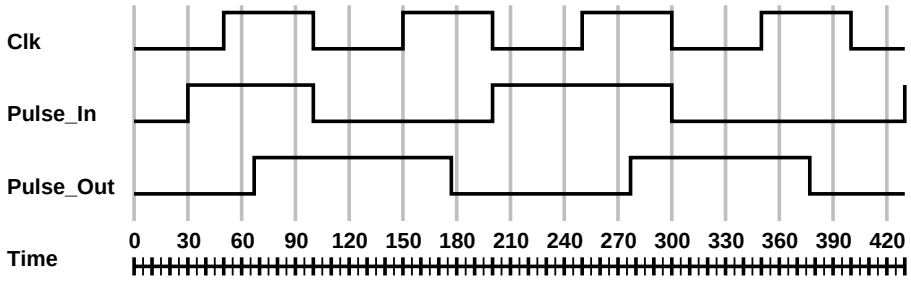

Fig. 3. Timing Behaviour of Single Pulser

7 A More Complex Circuit: A Bus Arbiter

The Bus Arbiter is another standard hardware verification benchmark circuit [10]. The purpose of the Bus Arbiter is to grant access on each clock cycle to a single client among a number of clients requesting use of a bus. The inputs to the arbiter are a set of request signals from each client. The outputs are a set of acknowledge signals, indicating which client is granted access during a clock cycle.

As shown in the structural specification of figure 4, each cell of the arbiter is moderately complex. The whole circuit consists of a number of such cells connected cyclically, e.g. three as shown in figure 5.

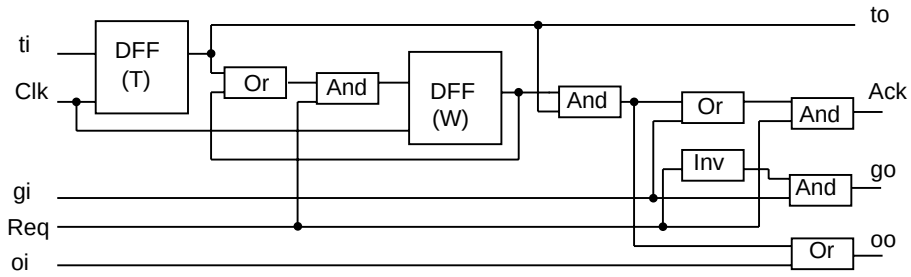

Fig. 4. Design of A Bus Arbiter Cell

The design of the circuit will be explained only briefly here. The *ti* (token in) and *to* (token out) signals are for circulation of the token. The *to* output of the last cell is connected to the *ti* input of the first cell to form a token ring. The *gi* (grant in) and *go* (grant out) signals are related to priority. The grant of cell *i* is passed to cell *i+1*,

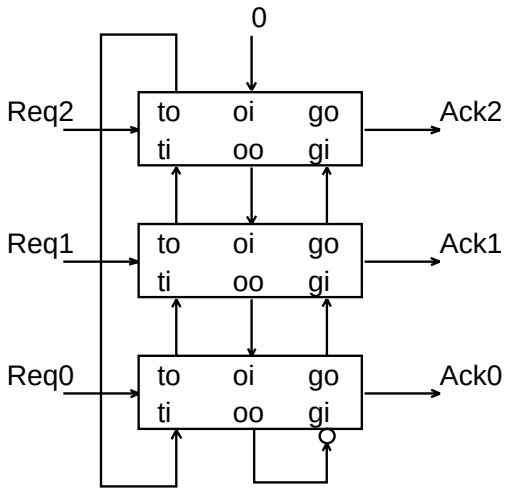

Fig. 5. Interconnection of Multiple Bus Arbiter Cells

indicating that no client of index less than or equal to *i* is requesting. Hence a cell may assert its acknowledge output if its grant input is asserted. The *oi* (override in) and *oo* (override out) signals are used to override the priority. When the token is in a persistently requesting cell, its corresponding client will get access to the bus; the *oo* signal of the cell is set to 1. This signal propagates down to the first cell (numbered 0) and resets its grant signal through an inverter. As a consequence the *gi* signal of every cell is reset, in other words the priority has no effect during this clock cycle. Within each cell, flip-flop *T* stores 1 when the token is present, and flip-flop *W* (waiting) is set to 1 when there is a persistent request. Initially the token is assumed to be in the first cell.

This circuit is relatively challenging. In the detailed design, the SDL specification contains 56 components (over 60 concurrent processes) and 93 signals. Nonetheless, the structure of the SDL specification closely resembles the circuit diagram and was easily produced. A behavioural specification of the intended behaviour was also written, so that it might be compared to the structural specification. The behavioural specification reflects the arbitration algorithm of the circuit.

Validation of the supposedly proven benchmark circuit uncovered a problem. Suppose that client 0 requests the bus in the first three clock cycles. In the fourth cycle, client 0 cancels its request but client 1 begins to request access. At this point the structural specification does not grant access to client 1 whereas the behavioural specification does. After interactive simulation of this case, it was discovered that the circuit provided in the benchmark (figure 4) may not properly reset the *oo* (override out) signal. The required correction is to connect the *Req* signal to the *And* gate that follows flip-flop *W*. The output of the *And* gate guarantees that the *oo* signal is always correctly set or reset according to the request signal in the current clock cycle.

A further problem was then discovered during automated validation using the random-walk approach. Analysis with the interactive timing simulator showed that this is due to the arbiter misbehaving when three clients simultaneously request access. In such a case the given arbiter design grants requests to two of the clients concurrently!

8 Conclusions

It has been seen how ANISEED can successfully model digital hardware as a collection of interacting parallel components. The emphasis is on timing specification and analysis. The authors were gratified to find that the approach can genuine problems with the Single Pulser and the Bus Arbiter – standard circuits that might have been supposed to be well verified. Work is continuing on the ANISEED library and tools. A GUI editor will be written to produce SDL hardware descriptions more directly from circuit diagrams. SDL verification techniques are also being developed for analysing hardware timing.

Acknowledgements

Financial support from NATO under grant HTECH.CRG974581 is gratefully acknowledged for collaboration with Dr. G. Adamis, Dr. Gy. Csopaki (who contributed to section 4 of this paper) and Mr. T. Kasza of the Technical University of Budapest. F. J. Argul-Marin thanks the Faculty of Management, University of Stirling, for supporting his work. Mrs. Ji He, University of Stirling, discovered and analysed the first arbiter design problem mentioned in section 7.

References

1. F. J. Argul Marin and K. J. Turner. Extending hardware description in SDL. Technical Report CSM-155, Department of Computing Science and Mathematics, University of Stirling, UK, Feb. 2000.
2. I. S. Bonatti and R. J. O. Figueiredo. An algorithm for the translation of SDL into synthesizable VHDL. *Current Issues In Electronic Modeling*, 3, Aug. 1995.
3. E. Bounimova, V. Levin, O. Başbuğoğlu, and K. İnan. A verification engine for SDL specification of communication protocols. In S. Bilgen, M. U. Çağlayan, and C. Ersoy, editors, *Proc. 1st. Symposium on Computer Networks*, pages 16–25, Istanbul, Turkey, 1996.
4. G. Csopaki and K. J. Turner. Modelling digital logic in SDL. In T. Mizuno, N. Shiratori, T. Higashino, and A. Togashi, editors, *Proc. Formal Description Techniques X/Protocol Specification, Testing and Verification XVII*, pages 367–382. Chapman-Hall, London, UK, Nov. 1997.
5. J.-M. Daveau, G. F. Marchioro, C. A. Valderrama, and A. A. Jerraya. VHDL generation from SDL specifications. In C. Delgado-Kloos and E. Cerny, editors, *Proc. Computer Hardware Description Languages and their Applications XIII*, pages 20–25. Chapman-Hall, London, UK, Apr. 1997.
6. T. Hadlich and T. Szczepanski. The ODE system – An SDL based approach to hardware-software co-design. In C. Müller-Schlör, F. Geerinckx, B. Stanford-Smith, and R. van Riet, editors, *Embedded Microprocessor Systems*, pages 269–281. IOS Press, Amsterdam, Netherlands, 1996.
7. G. J. Holzmann. Practical methods for the formal validation of SDL. *Computer Communications*, 15(2):129–134, 1992.
8. ITU. *Message Sequence Chart (MSC)*. ITU-T Z.120. International Telecommunications Union, Geneva, Switzerland, 1996.
9. ITU. *Specification and Description Language*. ITU-T Z.100. International Telecommunications Union, Geneva, Switzerland, 1996.
10. J. Staunstrup and T. Kropf. IFIP WG10.5 benchmark circuits. http: //goethe. ira. uka.de/hvg/ benchmarks.html, July 1996.