Stoht - An SDL-to-Hardware Translator

Ivanil S. Bonatti

Faculty of Electrical Engineering
University of Campinas
Campinas, SP Brazil 13081-970
Fax: +55-192-391395
e-mail: ivanil@dt.fee.unicamp.br

Renato J.O. Figueiredo

Faculty of Electrical Engineering
University of Campinas
Campinas, SP Brazil 13081-970
Fax: +55-192-391395
e-mail: rjof@dt.fee.unicamp.br

Abstract— This article presents a language translator that allows the use of SDL as a front-end, high-level graphical description tool for hardware design. A subset of this language is proposed for hardware design, including a synthesisable model for SDL's signal-based communication. An algorithm to translate this subset to fully synthesisable VHDL is proposed and implemented in a public-domain software package.

I. Introduction

The gap between specification and implementation of digital circuits is responsible for a good deal of design effort spent in the mapping of a given specification into a real system. The development of behavioral Hardware Description Languages, remarkably VHDL [8] [9], and the availability of Electronic Design Automation (EDA) tools that make use of such languages have allowed the description of hardware at a higher level. Gate-level synthesis is straightforward from a VHDL description if the VHDL code respects synthesis constraints.

The goal of this work is to place the hardware design closer to its specification, without losing the implementation capability. This will be accomplished by using CCITT's SDL (Specification and Description Language [1]) as a graphical description tool.

A proposal of using SDL for VHDL-based hardware design was introduced in [4], and further refined in [7], covering simulation aspects, and in [3], covering synthesis. In [6] an intermediate language interfacing system-level (SDL) and implementation-level (VHDL) languages is presented, and a software/hardware co-design approach based on this language in proposed in [5].

The use of a specification language such as SDL for hardware design offers advantages when compared to VHDL-based hardware design: graphical capture, higher level of abstraction, better documentation and easier approach to a hardware/software co-design methodology, by using automatic code generation and high level synthesis.

This article proposes the use of a subset of the SDL language, consisting of its most usual elements (system, blocks, substructures, processes, states, signals, tasks and decisions) to the hardware design and the application of some constraints to allow efficient synthesis. This subset can be mapped to synthesisable VHDL with the mapping rules also proposed in this article.

The main contribution of this work is the specification of a flexible hardware model of SDL's signal exchanging scheme for communication synthesis. Low-cost implementations (in terms of hardware area) can be achieved with the simplification of SDL's finite queue model for signals arriving in processes. Such property is attractive for the design of embedded circuits using programmable logic. This algorithm can be used in the design of general-purpose digital circuits; it's not geared to any specific application.

The first part of this article focuses on which elements of SDL are supported and the constraints that are applied to this SDL subset to make the description efficiently synthesisable. The next part covers the translation algorithm itself that is implemented in the **Stoht** translator.

Along this text the SDL reserved words are written in bold lowercase, with the first letter uppercase (such as **Process**), while VHDL reserved words are written all uppercase (such as PROCESS), unless they can be distinguished by context.

II. The SDL scope

A. Elements supported

The most usual SDL elements are supported by the algorithm, and are well suited for the behavioral description of digital systems:

System: may contain blocks, channels, signals and data definitions.

Block: may contain processes, substructures, signal-routes, signals and data definitions.

Substructure: only block substructure is supported. It may contain blocks, channels, signals and data definitions.

Process: supported, but restricted to only one instance per process; dynamic creation of processes is not allowed. It may contain states, inputs, continuous signals, tasks, outputs, decisions, variables and data definitions.

Signal: Interchange of signals among processes via input/output primitives is allowed. The infinite-length queue abstraction of SDL isn't supported; finite queues are used instead. A signal must be conveyed by channels and signalroutes to its destination. Continuous signals and enabling conditions are also supported.

Channel, **Signalroute**: must be used to establish communication paths among **Processes**.

Variable: ordinary and **Revealed Variables** are supported.

Abstract types: Predefined supported types are **Integer** and **Boolean**. Integer ranges are declared by using **Syntype** constructs. Unidimensional arrays of boolean data are declared using **Newtype** constructs. **Synonyms** are supported for constant declaration.

B. Synthesis constraints

Many SDL features would lead to a very costly implementation if they were to be represented in hardware. For example, the dynamic creation of processes, floating-point arithmetic and infinite-length signal queues would require a great number of components for the implementation. They are not supported.

C. Inter-process communication

The communication model adopted by SDL contains abstractions that are not easily mapped into digital circuits; the infinite-length queue for incoming signals and the asynchronous nature of signals are two examples.

The mapping algorithm used by **Stoht** simplifies this model, in order to obtain synthesisable hardware at a low cost in terms of number of gates.

The first simplification is applied to the infinite-length queue for incoming signals; it is replaced by a finite queue that can only hold a limited number of signals. The lowest-cost implementation uses only one position per process for this queue; queues with more than one position are available to the designer for processes that may need it, significantly increasing the cost of the resulting hardware.

Also, asynchronous **Signals** are made synchronous to a global clock that's automatically assigned by the translator. Each sending or receiving of a **Signal** is evaluated in the rising edge of this master clock.

The algorithm implements the communication model in a VHDL ENTITY that is separated from the VHDL ENTITY that models behavior. Thus, each SDL **Process** is mapped into two separated VHDL ENTITYs: one modeling the behavior and the other one modeling the communication.

The VHDL ENTITY that models the communication of an SDL **Process** is referenced as "protocol" in the rest of this text. The interfaces between the protocol and the VHDL ENTITYs modeling behavior are defined in such a way that protocols can be replaced without changes in the behavior. The algorithm currently supports three types of protocols.

D. Protocols

The protocol that uses a queue of only one position for *all* incoming signals is called "Protocol 1". It's the most simple protocol, requiring less circuitry to be implemented.

"Protocol 2" allocates a queue of one position for *each* incoming signal that an SDL **Process** may receive.

"Protocol 3" uses a finite queue of N positions for *all* incoming signals (N is a parameter declared by the designer). It's the most expensive protocol in terms of hardware.

Any of these protocols interface with the behavioral part of the **Process** in the same way, guaranteeing flexibility for the designer. Furthermore, new protocols can be added to the algorithm.

E. Input handling and priority

Since finite queues are used in the implementation of SDL's communication scheme, conflicts may occur when more than one signal arrive in the same time-slot (clock cycle) and only one position is available in the queue. To solve this conflict, each signal has to be assigned a priority by the designer. Also, if a queue is allocated and more than one signal are in this queue, the signals are consumed in the order given by their priorities.

III. The mapping algorithm

The hardware implementation of the SDL system is synchronous to a global clock that triggers every transition inside processes and every sending or receiving of SDL signals. Synchronous machines are less sensitive to timing errors than asynchronous ones, and are more easily represented in synthesisable VHDL.

With this synchronization scheme every transition body in an SDL process takes one period of the master clock to be processed, and every signal sent from one process to another also takes a clock period to arrive in its destination.

The master clock is automatically allocated by the translation algorithm, as well as a master asynchronous reset that initializes all variables and starting states of processes. Both VHDL signals above, named **clock** and **reset**, are input PORTs of each VHDL ENTITY generated by the algorithm.

A. Structure

The SDL **System** generates a highest-level VHDL ENTITY that instantiates all of its **Blocks** as VHDL components and has input/output PORTs according to the

Signals sent to or received from the environment. Similarly, an SDL **Block** generates a VHDL ENTITY that instantiates all of its **Processes** as VHDL components and has input/output PORTs according to the **Signals** sent to or received from **Channels**.

An SDL **Process** generates two separate VHDL ENTITYs. The first ENTITY holds the behavior of the correspondent SDL **Process**. The second one implements the communication protocol associated with this **Process**.

B. Data definition

Sorts can be derived from the supported predefined sorts (**Boolean** and **Integer**) by using either the **Syntype** (for constrained integers) or the **Newtype** construct (only for the definition of constrained vectors of boolean data); both map into VHDL TYPEs. SDL **Synonyms** can also be defined, and are mapped into VHDL CONSTANTs.

Sorts defined in an SDL **System**, **Block** or **Block Substructure** are placed in VHDL PACKAGEs, one for each scope, so they can be further referenced by the lower-level **Blocks** and **Processes**.

Sorts declared in an SDL **Process** are defined in the correspondent VHDL ENTITY modeling the **Process** behavior.

C. Variables

Variables declared in an SDL **Process** are declared as VARIABLEs in the behavior-modeling VHDL PROCESS.

Variables declared **Revealed** are made visible to the other **Processes** in the same **Block** by creating a VHDL output PORT having a copy of their value. **Processes** that view **Revealed Variables** have input PORTs that are read when a **View** access is made. The SDL-92 Z.100 [2] recommendation allows the resolution by name of the viewed **Variable** inside a **Block**.

D. Process Communication

Each SDL **Process** has one corresponding VHDL ENTITY modeling its signal-based communication. This ENTITY, named "protocol", provides an interface between the incoming **Signals** and the behavior of the **Process** as follows:

- There is one input PORT and one output PORT for each **Signal** that is received by the **Process**. For each parameter that each **Signal** might carry, one input PORT and one output PORT are added.

- The protocol receives a request of **Signal** sending by sensing a logic inversion in the respective input PORT. At this time, the **Signal** parameters (if any) are available in the respective input PORTs.

- The protocol gives notice of an incoming **Signal** by holding a positive logic pulse for one cycle of the master clock in the respective output PORT. It also holds the parameters (if any) for this period of clock in the respective output PORTs.

Thus, the action of sending an SDL **Signal** via the **Output** primitive is performed with the following actions:

- Assignment of the values of each parameter, if any, to the respective VHDL output PORTs;

- Logic inversion of the VHDL output PORT that represents the action of sending a signal.

A transition is initiated when the receiving VHDL PROCESS senses the presence of this positive pulse in the VHDL input PORT written by the protocol. This test is accomplished through the use of an IF-ELSE-ELSIF structure, ordered according to the priorities assigned by the designer to each incoming signal.

Fig. 1 shows examples of VHDL code for the communication synthesis for the sending process, the protocol and the receiving process. The SDL signal **signal1** used in this example carries one parameter. The protocol used in this example does not store signals in a queue; it just converts the input logic inversion (sending request) to a clock-wide positive logic pulse for the receiving process.

```
  -- Sending process
send_par1_signal1 <= SOME_VALUE;
send_signal1 <= NOT(send_signal1);
  -- Action of sending an SDL signal

  -- Protocol
latch : PROCESS(clock)
BEGIN
  IF (clock'EVENT AND clock='1')
    int_signal1 <= rec_signal1;
  END IF;
END PROCESS latch;
p_send_signal1<=int_signal1 XOR rec_signal1;
p_send_par1_signal1 <= rec_par1_signal1;
  -- The simplest protocol is used here

  -- Receiving process
IF (p_rec_signal1='1') THEN    -- Priority 1
  some_variable := p_rec_par1_signal1;
  -- transition body
ELSIF (p_rec_signal2='1') THEN -- Priority 2
  -- transition body
END IF;
  -- The receiving process tests the
  -- presence of signals according to
  -- their priorities
```

Fig. 1: Example of the communication synthesis in VHDL

The final implementation of the communication model depends on the protocol chosen by the designer. There are

currently three kinds of protocols supported by the algorithm that are modeled in VHDL. The simplest protocol requires only one flip-flop and an XOR gate for each signal received by an SDL process.

E. Process Behavior

Each SDL **Process** has one corresponding VHDL PROCESS inside an ENTITY that models its behavior. The sensitive list of this PROCESS is solely the master clock and reset automatically allocated by the algorithm.

The current state of a **Process** is stored in a VHDL SIGNAL inside this ENTITY. An enumerated TYPE is used to store each possible **State** of the **Process** plus an initial state artificially created by the algorithm.

State transitions are modeled with a CASE/WHEN structure. Each possible state has a corresponding entry in a WHEN clause. The body of each WHEN clause has an IF-ELSIF-THEN construct that tests the presence of an incoming **Signal**. The body of each IF-ELSIF-THEN construct has the transition body (actions) for the corresponding incoming **Signal**; they are ordered according to the priorities assigned by the designer

Assignment **Tasks** are mapped into sequential VHDL assignments. **Decisions** are mapped into IF-ELSE-THEN constructs.

IV. Stoht

The translation algorithm has been implemented in a software called **Stoht** (**S**DL-**to**-**H**ardware **T**ranslator).

Stoht translates textual SDL descriptions to VHDL behavioral models. It is designed to work together with other CASE/CAE tools. It has been successfully tested with **Telelogic**'s **SDT** graphical SDL editor, **Mentor Graphics**' VHDL compiler and synthesizer.

Stoht reads a single input file containing an SDL-PR description and analyzes its syntax and semantics according to the synthesis restrictions that the algorithm introduces. The program keeps the original SDL structure in the generated directory tree. **Blocks** are placed under their **System** or **Substructure** upper scope, **Substructures** and **Processes** are placed under their respective **Blocks** subdirectories.

Stoht also generates library mapping files and a compiling batch script that make the compiling procedure straightforward.

Stoht is available as public domain software, and can be installed in several UNIX platforms, from workstations to PCs. More information on the software can be found in the Internet World Wide Web link "http://dt.fee.unicamp.br".

V. Conclusion

The specification of digital systems at higher levels provides faster development cycles, technology-independent circuits and better documentation. The proposed algorithm allows hardware developers to use a subset of the high-level graphical language SDL as a front-end to their designs, by automatically generating fully synthesisable and syntactically correct VHDL. The algorithm is implemented in the program **Stoht**, available as public-domain software. Further research will include aspects of hardware/software co-design and cover more features of the SDL language.

References

[1] CCITT. *Recommendation Z.100: Specification and Description Language SDL*, volume X.1-X.5. CCITT, 1988.

[2] CCITT. *Recommendation Z.100: Specification and Description Language SDL*, volume X-R25-X-R32. CCITT, 1992.

[3] W. Glunz, T. Kruse, T. Rössel, and D. Monjau. Integrating SDL and VHDL for System-Level Hardware Design. In *CHDL 93 - Computer Hardware Description Languages and their Application*, Ottawa, Canada, April 1993.

[4] W. Glunz and G. Venzl. Using SDL for Hardware Design. In *Proceedings of the Fifth SDL Forum*, Glasgow, 1991.

[5] T.B. Ismail, M. Abid, and A. Jerraya. COSMOS: A Codesign Approach for Communicating Systems. In *Co-Design, Computer Aided Hardware/Software Engineering*. IEEE Press, 1994.

[6] A.A. Jerraya and K. O'Brien. SOLAR: An Intermediate Format for System-Level Modelling and Synthesis. In *Co-Design, Computer Aided Hardware/Software Engineering*. IEEE Press, 1994.

[7] B. Lutter, W. Glunz, and F.J. Rammig. Using VHDL for Simulation of SDL Specifications . In *Proceedings of the EURO-VHDL*, 1992.

[8] The Institute of Electrical and Electronic Engineers. *IEEE Standard VHDL Language Reference Manual*. IEEE, 1987.

[9] D.L. Perry. *VHDL*. McGraw-Hill Inc, 1991.