PROTOTYPING OF EMBEDDED DIGITAL SYSTEMS FROM SDL LANGUAGE: A CASE STUDY

César A. M. Marcon, Fabiano Hessel, Alexandre M. Amory, Luís H. L. Ries, Fernando G. Moraes, Ney L. V. Calazans

{marcon, hessel, amory, ries, moraes, calazans}@inf.pucrs.br

ABSTRACT

The goal of this paper is to evaluate the performance of embedded digital systems generated from a system level description language. The target language is SDL, which is automatically synthesized with a codesign tool, resulting in VHDL and C descriptions. The codesign tool is responsible for software, hardware and communication synthesis. Two case studies are presented, exploring area and delay results. The results concern only the hardware synthesis, since the goal is to compare the performance of systems generated from hand coded HDL descriptions against a synthesized HDL. The analysis of the advantages and drawbacks of this automatic hardware design flow and the evaluation of the commercial tools integration are also reported.

1 Introduction

Embedded systems requirements are getting increasingly complex. This complexity requires modern design methodologies to prototype such systems, viewing the time-to-market reduction. In general, embedded systems are built with hardware and software parts developed concomitantly, a method usually named codesign. Coware N2C [1], Ptolemy [2], SEA [4], VCC [5], Seamless [6] and the work described in [7] are typical environments supporting codesign. These environments start from a system level specification with languages like SDL (Specification and Description Language) [8], Esterel [3], Java and C/C++ extensions.

A typical codesign flow starts with an informal specification of the whole system, generally in natural language. This specification contains requirements and constraints. It enables the elaboration of the system level description, which depicts functionality as a hierarchical mixed data/control flow diagram, and can be written in one of the above-cited languages.

The second step in the design flow consists in the refinement of the system level communication model, including protocol selection. Some design alternatives are examined to identify those meeting the system constraints, and the architectural choices are made. Once the architecture is determined, the functional specification is mapped into an abstract architectural model. At this stage, the system can be viewed as a technology independent multiprocessor architecture mixed with hardware components.

The third step comprises hardware/software partitioning and scheduling. Partitioning is the mapping of functional subsystems onto abstract processes. Scheduling defines the exact start time of each process. The software part is described in high-level programming language and the hardware part is written as HDL code. Besides the software-only functions, the software part includes low-level device drivers for interfacing with hardware components and eventually an OS synthesis step. In the same manner, the hardware part contains the interfaces and components to communicate with the software.

The fourth step comprises software, interface and hardware synthesis. The main difference between input and output descriptions in this step is that the output descriptions are targeted to a specific architecture, including the choice of processor(s), of low-level communication, and hardware modules interfaces. At this stage, the architectural model is detailed and the clock-cycle validation is obtained through RTL co-simulation.

The design flow proposed in this work is illustrated in Figure 1. It starts with a system level description in SDL. This description is alternatively obtained using three SDL environments, Telelogic TAU SDL™ [9], Cinderella SDL™ [10] and ObgectGeode™ [11]. These environments allow system validation through MSC (Message Sequence Chart) high level simulation. The goal of choosing more than one environment is to compare these tools and evaluating the portability of SDL descriptions. The tools produce SDL textual descriptions, which are input to Archimate™ [12], a codesign environment performing semi-automatic hardware-software partition, hardware and software synthesis and communication refinement. Archimate internal format is called BEEF. The synthesis steps generate C descriptions for software part and RTL VHDL descriptions for the hardware part. The final step is prototyping, where hardware is obtained through Leonardo Spectrum™ [13] and Foundation™ [14] tools. The software synthesis is not considered here, since our goal is to analyze the advantages and drawbacks of hardware flow description only. The systems are targeted to Xilinx Virtex FPGAs.

0-7803-7655-2/02/$17.00©2002 IEEE

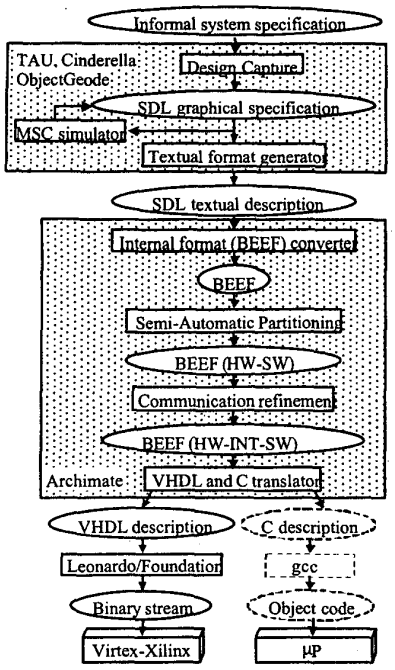

Figure 1 – Design flow.

System level descriptions allow faster system validation and technology independence, when compared to RTL abstraction. From the available system level languages, this work focuses on SDL. The goal is to compare SDL synthesized hardware to hand-coded VHDL descriptions. Comparison considers area and delay figures. It is also possible to identify problems related to the SDL underlying model and/or to the codesign synthesis tool.

This paper is organized as follows. Section 2 discusses the SDL underlying model, emphasizing its usage in digital systems. Section 3 illustrates two digital system case studies. The obtained results are discussed in section 4. Finally, some conclusions and directions for future work are presented in Section 5.

2 SDL Underlying Model

SDL is an object-oriented language standardized by ITU-T [8], which allows the hierarchical description of systems. The description starts from a construction called *system*, where functional blocks are inserted. A *block* is a component composed by one or more *processes* and/or other blocks. A process contains a sequential behavior and concurrency is modeled by a set of processes.

2.1 Process Model

The Z.100 ITU-T standard defines that the SDL underlying model is EFSMs (Extended Finite State Machines), where all processes are ESFMs. A more generally accepted terminology to EFSM is to call it a Non-deterministic Finite Automaton (NFA) [15]. For each process, a finite number of states, inputs and outputs determine its behavior. Non-determinism capability allows representing *spontaneous transitions*, which are transitions without any signal causing them. This is useful to describe unpredictable system characteristics. An NFA allows that each state may consume and generate multiple signals. However, in SDL only one input signal can be consumed (evaluated) at each instant. This means that each input signal consumed corresponds to one state transition in an SDL description.

2.2 Communication Model

The concurrency model used in SDL allows independent and asynchronous processes operation. There is no guaranteed relative ordering of operations in distinct processes, except the ordering created by explicit synchronization among processes through the use of shared signals. Shared signal events are then the means by which a precise ordering of events in distinct process can be achieved.

The communication between processes is reliable. It is assured that the receiving process will consume every signal produced by a sender process. However, it is not guaranteed that the ordering of the signals generated by all processes is the same of their consumption. This model is adequate to describe events without precise ordering, like systems that can have their normal flow interrupted. Handshaking or unlimited queues (in practice bounded queues) are used to implement the communication model. For both cases, each SDL state results in a set of protocol communication signals and area overhead to implement the protocol. This characteristic may cause large communication overhead, which can penalize the implementation.

2.3 Synchronous Modeling Proposal for SDL

The SDL model for processes assumes single input signal consumption for each SDL state. SDL provides a construction, called SAVE, to describe processes that exchange more than one signal event during an SDL state. This construction stores signal events that may be available in instants where the process is not able to consume them, as illustrated in Figure 2. The SAVE construction represented graphically by a parallelogram, stores signal A value to be consumed in the next sate.

The SDL code showed in Figure 2 illustrates that, if the process is in state S0, it may deal with two situations. The first is B occurring before A, the second is A occurring before B. To avoid losing the A value, if it occurs before B, the SAVE construction is used to temporarily store its contents to be evaluated by another state.

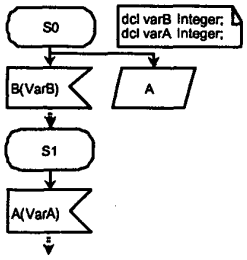

Figure 2 – SDL SAVE construction.

The main drawback of SAVE is area and delay overhead. The SDL standard recommends reducing the use of SAVE. This is the reason why some synthesis tools do not support this construction, including Archimate. To overcome this problem, an approach to simulate SAVE in synchronous systems is proposed.

This approach, illustrated in Figure 3, assumes that each SDL state represents a clock cycle. Therefore, all signals inside a single clock period can be generated without relative ordering, as long as they remain stable during the clock transition, which is exactly what is expected in a synchronous system. What determines state transition is a clock event, instead of any other input signal event. In the example illustrated in Figure 3, any transition of signals A or B will be stored respectively in varA or varB local process variables. Even with this approach, the SDL concurrency model implies a communication overhead for synchronous systems.

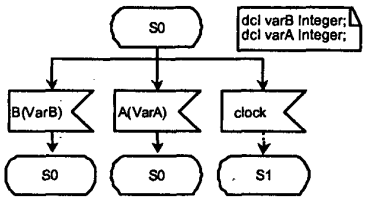

Figure 3 – An approach to simulate SAVE construction.

3 Case Studies

This Section presents the characteristics of two case studies, used as benchmarks in the next Section. Both systems were described in SDL and VHDL languages. VHDL descriptions were implemented in FPGAs. The first case study is an industrial application and the second is an academic example.

3.1 DropInsert Benchmark

The DropInsert is a telecom system that manipulates an E1 multiframe (MF) [17], dropping and inserting data and/or voice channels. MF is a structure made up of two sub-multiframes (SMF). An SMF contains 8 frames; each frame has 32 slots and each slot has 8 bits, for a total of 256 bits per frame. The basic structure of a MF is depicted in Figure 4. The first slot of each frame contains MF control information. The rest of the slots (1 to 31) are responsible for carrying voice and data. As the frame repetition rate is 8KHz, the basic frequency of an E1 frame is 2048 KHz.

Frame no.		time slot 0 (ts 0)								ts 1			···	ts 31		
↓		1	2	3	4	5	6	7	8	9	···	16	···	249	···	256
Multiframe	Sub-multiframe I	0	C1	0	0	1	1	0	1	1						
		1	0	1	A	S4	S5	S6	S7	S8						
		2	C2	0	0	1	1	0	1	1						
		3	0	1	A	S4	S5	S6	S7	S8						
		4	C3	0	0	1	1	0	1	1						
		5	1	1	A	S4	S5	S6	S7	S8						
		6	C4	0	0	1	1	0	1	1						
		7	0	1	A	S4	S5	S6	S7	S8						
	Sub-multiframe II	8	C1	0	0	1	1	0	1	1						
		9	1	1	A	S4	S5	S6	S7	S8						
		10	C2	0	0	1	1	0	1	1						
		11	1	1	A	S4	S5	S6	S7	S8						
		12	C3	0	0	1	1	0	1	1						
		13	E	1	A	S4	S5	S6	S7	S8						
		14	C4	0	0	1	1	0	1	1						
		15	E	1	A	S4	S5	S6	S7	S8						

Figure 4 - Basic E1 Multiframe structure.

This system is part of the physical level of OSI reference model. Voice transmission implies real time operation. DropInsert needs to evaluate one bit in less than 490 ns, meaning the implementation is not time-critical for most devices. Because E1 frames may carry data, it is necessary to guarantee reliable communication. The E1 protocol is essentially asynchronous, the first time slot is used to synchronize the frame operation. The DropInsert implementation of this control-flow protocol is synchronous, using the method proposed in Section 2.3. Thirty concurrent VHDL processes were employed to implement DropInsert in the hand coded description. Half of these processes are essentially related to bit manipulation.

3.2 Polygon Filling Benchmark

The second case study is an academic example of an algorithm to fill non-concave polygons. The algorithm receives a set of coordinates, representing a polygon, and generates the horizontal lines to fill it. Figure 5 illustrates a polygon and a filling table data structure that stores the horizontal line coordinates. The Y coordinate is used as an index for the filling table. For example, row Y_4 of the filling table has the X_3 (begin) and X_4 (end) coordinates of a horizontal line used to fill the Y_4 line of the polygon. The input image can have thousands of polygons, and consequently, the same number of filling tables is created. This behavior involves a large amount of memory accesses. Distinct from the DropInsert system, Polygon is typically a dataflow-oriented system.

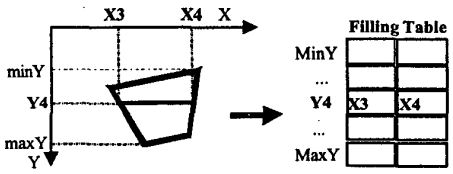

Figure 5 - Example of polygon and filling for the polygon-filling benchmark.

The polygon filling benchmark was partitioned in five SDL processes. The main feature of this particular partition is the great number of messages exchanged between processes. These processes are illustrated in Figure 6. The *delta_x* and *delta_y* processes perform subtractions. The *displac_y* process does comparisons. The *displac_x* process executes divisions. The *point_gen* process generates horizontal lines to fill the polygon.

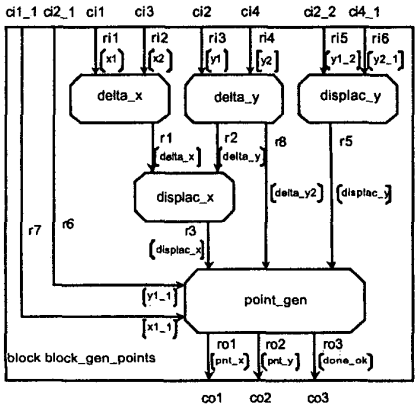

Figure 6 - SDL code for the polygon-filling benchmark.

4 Results

This Section presents results obtained from DropInsert and Polygon case studies. Both systems are used to analyze the integration of SDL design capture commercial tools and the SDL to VHDL translation automatically performed by Archimate.

The Polygon benchmark has three VHDL descriptions: (*i*) *Hand-coded*; (*ii*) *Single_process*, obtained from a single process SDL description, without concurrency; (*iii*) *Five_processes*, explores concurrency and communication. The DropInsert benchmark is more complex than Polygon, having just two VHDL descriptions: (*i*) *Hand-coded* and (*ii*) *Automatic*. In both case studies, the hand coded descriptions of DropInsert and Polygon were implemented using traditional digital system tools like manual design entry followed by an HDL simulator and hardware synthesis tools. The codesign tool automatically generated the other VHDL descriptions from SDL.

4.1 SDL to VHDL Translation

The DropInsert benchmark was first described with Cinderella SDL [10]. To evaluate SDL portability, the Cinderella description was used as an entry for two other SDL environments, Telelogic TAU SDL [9], and ObgectGeode [11]. It was observed that the graphical representation is not the same and the generated textual description, used by the codesign tool, was slightly different among the tools. In addition, TAU SDL is the only tool generating input code accepted by Archimate. This fact shows that either SDL is only partially portable, or that Archimate accepts only a subset of SDL.

The VHDL code generated by the co-synthesis tool [12][16] has the following basic template:

- The *entity* of each synthesized description (module) is a function of the selected communication protocol. For the handshake protocol an SDL channel is represented by send, acknowledge and data ports in VHDL. Each entity has just one input interface, with a set of ports necessary to implement a given protocol, and may have several output interfaces. The number of output interfaces depends on the number of target processes, such as process *delta_y* in Figure 6, which sends data to two processes. The constraint of one input interface generates an input serialization, leading to a communication overhead, which in turn reduces performance. This entity implementation depends on the codesign tool and it is strongly related to the SDL model, which allows just a single input consumption at each state.
- SDL *data types* used to describe digital systems are integer to carry data and non-valued signals to represent events. SDL integer type is translated to VHDL integer and SDL non-valued signals are translated to VHDL *bit* data type. In hardware, it is interesting to restrict the range of data values, to optimize the implementation, e.g. 8-bit buses to carry ASCII characters. Archimate does not support this kind of range limited data types. This results in a VHDL description using the full integer range (32 bits in VHDL), generating large area penalty.
- An SDL process is translated to one entity/architecture in VHDL. Two VHDL processes represent the architecture: a combinational and a sequential process. These two processes implement the FSM that represents the system behavior. This FSM is generally very large due to the number of states induced by the serialization protocol. For example, there are 15 states in the FSM of *delta_x* SDL process, depicted in Figure 6, to implement a subtraction. Just one of these states is used to execute the operation, the others are needed to implement the handshake protocol. This fact also leads to an overhead in area and speed, and corroborates the idea that SDL is not appropriated to describe synchronous digital systems.

- Due to the large number of signals in the *sensitivity list* of combinational VHDL processes, the hardware synthesis tool requires a large amount of time to synthesize the modules, and the area and delay obtained are not optimized. For example, the same SDL process *delta_x*, which implements a subtraction, has a *sensitivity list* with 12 signals. This problem could be reduced if Archimate translated the combinational parts to concurrent assignments instead a single process.

4.2 Comparison of SDL and VHDL Verification

Waveforms are a typical output generated by a VHDL simulation. All signals are time tagged, what means that there is a total ordering of signals. The time is one of the waveform coordinates, the other coordinate is the signal value.

SDL descriptions have an abstract model of simulation implemented in the MSC formalism as shown in Figure 7. The MSC shows the data signal exchange between processes, omitting all communication protocol signals. There is a total ordering of signals, but the precise time of an event is not important. The vertical lines represent the processes and the other lines represent the communication, carrying the signal values. In this example, there are two processes: *data* and *add*. The *data* process has one state called *waiting* while the *add* process has two states called *recv1* and *recv2*. Initially, the *data* process sends two signals, called *data1* and *data2* with value 1 and 2 respectively, to the *add* process. This cause in the *add* process a state transition from *recv1* to *recv2*. After that, the *add* process answers to *data* sending a signal called *res* with value 3

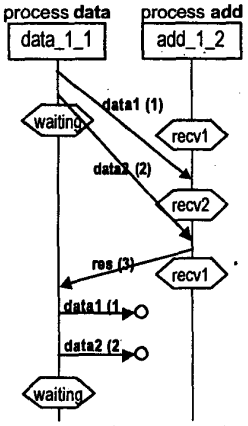

Figure 7 – MSC formalism simulation.

VHDL simulators are, in general, implemented with the Discrete Event (DE) model. The DE model implies scheduling all signal events in discrete instants of time. The abstraction implied by the use of the MSC model leads to faster simulation, which is useful for high-level validation. On the other hand, VHDL simulations give much more detailed information.

An MSC simulation is used to abstract some implementation details and to help the designer to validate the system more quickly. These features are expected to become relevant for large systems, as waveform analysis becomes unfeasible as system complexity increases.

4.3 Functional Performance Analysis

This Section analyzes the effects of the communication and concurrency models over functional performance. Table 1 shows the number of clock cycles necessary to perform the RTL simulation on three different Polygon implementations.

Table 1 - Simulation data of different implementations for the Polygon system.

	Hand Coded	Single process	Five processes
Clock cycles	30	530	900

The use of modularity is a well-known method to manage design complexity. So, it is reasonable to expect that modularity usage in SDL would not penalize performance. However, comparing *Single_process* and *Five_processes* implementations, it can be noted that increasing the number of processes led to a performance reduction. When the number of processes in an SDL description increases, the overhead associated to the handshake protocol also increases. Comparing automatic VHDL with hand coded VHDL description, it can be observed that it is necessary 15 to 30 times more clock cycles to do the same job. The asynchronous protocol and the interface serialization are responsible for this communication overhead.

4.4 Synthesis Performance Analysis

Table 2 presents some results obtained from the Polygon benchmark, targeted to the 300,000 equivalent gates XCV300BG352. The first two lines contain the number of Configurable Logic Blocks (CLBs) and the number of D flip-flops, roughly representing the area to implement the system. The third line shows maximum operating frequency estimation. The last line displays the CPU time to complete the synthesis on a Sun Ultra10, 333 MHz/256 MB RAM.

Table 2 - Polygon benchmark synthesis data.

	Hand Coded	Single process	Five processes
# of CLBs	130	401	1008
# of D flip-flops	133	417	1095
Frequency (MHz)	76.4	70.2	48.3
Time for synthesis	33 s	123 s	355 s

Table 1 shows that hand coded implementation is 30 times faster than the *Five_processes* implementation. If the same performance level is to be attained for both implementations, the *Five_processes* should have its clock frequency increased 30 times. Besides this, Table 2 shows that automatic synthesis penalizes area (3 to 7 times) and operating frequency (up to 1.58).

DropInsert results are not shown because the available physical synthesis tools could not synthesize all processes due to the excessive number of states. Some processes of the automatic generated VHDL description have more than 500 states, which is 25 times more than the hand coded description. This gives a measure of how far is the underlying SDL model of computation from synchronous systems, a result similar to what is reported in [18].

5 Conclusions

It is possible to enumerate some SDL language advantages for hardware design, such as:

- Steep learning curve, as it has graphical as well as textual input formats;
- Easy system description, due to a relatively small number of languages primitives;
- Possibility to express non-determinism due to its concurrency model;
- Fast system verification due to the abstract communication model used in MSC simulation.

However, the SDL communication model is unsuitable to represent many classes of embedded systems, due to the lack of constructions to express synchronous processes, like the possibility to simultaneously evaluate more than one signal transition at each SDL state. The communication model directly affects the choice of the number of processes to implement a system, compromising design modularity. Using many communicating processes may strongly compromise the quality of the hardware implementation.

The SDL specification using multiple commercial environments revealed that available tools are not fully compatible, although all claim to be compliant to the ITU-T SDL standard. This is particularly true in the case of SDL graphical format. This incompatibility is not observed when dealing with VHDL tools.

The Archimate automatic translation of SDL to VHDL generates an implementation that end up using much more area and operates at a considerably lower frequency than the equivalent hand coded design. There are two reasons for this:

- The main reason is the SDL communication model. To represent this model, Archimate groups all process inputs into a single input, with a serial protocol identifying each input;
- The VHDL generated by Archimate reflects the same structure of the input SDL description, i.e., a set of process, mostly without optimization.

These results show that much more effort is required in higher-level synthesis tools research in order to generate hardware descriptions competitive with hand coded implemented systems. The authors suggest that hardware parts of computational systems be described with HDLs, leaving SDL to describe software and the communication between asynchronous subsystems.

References

[1] K. Van Rompaey, D. Verkest, I. Bolsens and H. De Man. *Coware, A Design Environment for Heterogeneous Hardware/Software Systems*. In: European Design Automation Conference. 1996.

[2] B. Lee and A. Lee. *Hierarchical Concurrent Finite State Machines in Ptolemy*. In: International Conference on Application of Concurrency to System Design. pp. 34-40. 1998.

[3] G. Berry. *The Foundations of Esterel*. Available at: http://www.esterel.org. INRIA.

[4] B. Kleinjohann. *Multilanguage Formalism*. MEDEA/Esprit Conference HW/SW Codesign. pp. x.1.1-x.1.22, 1998.

[5] Virtual Component Co-Design (VCC). Available at: http://www.cadence.com.

[6] R. Klein. *A Hardware Software Co-Simulation Environment*. In: RSP'96. pp. 173-177. 1996.

[7] D. E. Thomas, J. K. Adams and H. Schmit. *A model and methodology for Hardware-Software Codesign*. IEEEDTC, vol. 10(3), pp. 16-28. 1993.

[8] ITU-T. *Programming Languages: SDL Methodology Guidelines, SDL Bibliography*. ITU-T Recommendation Z.100 – Appendices I and II, March 1993.

[9] Telelogic TAU. *A SDL tools for real time systems development*. Telelogic, Inc. Available at: http://www.telelogic.com.

[10] Cinderella. *A visual SDL tools for systems development*. Available at: http://www.cinderella.dk.

[11] Telelogic Objectgeode. *A SDL tools for real time systems development*. Telelogic, Inc. Available at: http://www.telelogic.com.

[12] Arexsys Archimate. *A SDL synthesis tool*. Arexsys, Inc. Available at: http://www.arexsys.com.

[13] Leonardo Spectrum. Available at: http://www.exemplar.com.

[14] Xilinx Foundation. *A system synthesis tool*. Xilinx, Inc. Available at: http://www.xilinx.com.

[15] D. Cohen, *Introduction to Computer Theory*. Wiley & Sons, 1991.

[16] J. M. Daveau, G. Marchioro, A. Jerraya. *VHDL generation from SDL specification*. In: CHDL. pp. 182-201, 1997.

[17] N. Calazans, F. Moraes, C. Marcon, V. Blauth, R. Valiati, E. Manfroi. *Effective Industry-Academia Cooperation in Telecom: a Method, a Case Study and Some Initial Results*. In: XIX Simpósio Brasileiro de Telecomunicações, 2001.

[18] A. Muth, G. Färber. *SDL as a System Level Specification Language for Application-Specific Hardware in a Rapid Prototyping Environment*. In: ISSS 2000. pp. 157-162. 2000.

Acknowledgements: F. Moraes, N. Calazans and F. Hessel are partially financed by CNPq (project numbers 522939/96-1, 520091/96-5, 680117/01-6) and FAPERGS (project number 01/0565.5).