Modeling of Embedded Digital Systems from SDL Language: a Case Study

César A. M. Marcon, Fabiano P. 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 digital systems generated from a high-level description language. The target language in this work is SDL. The SDL description is automatically synthesized with a codesign tool, resulting in a VHDL description. The codesign tool is responsible for software, hardware and communication synthesis. The presented results concerns only the hardware synthesis, since the goal is to compare the performance of systems generated from manual VHDL descriptions against a synthesized VHDL. Two case studies are presented, exploring area and delay results.

1 Introduction

Embedded systems requirements are getting increasingly complex. This complexity requires modern design methodologies to prototype such systems in a competitive time-to-market. In general, embedded systems are constructed with hardware and software parts, leading to a design environment implementing the hardware and software concomitant design, or just codesign. Coware n2c [1], Ptolemy [2] and Seamless [3] are typical environments supporting such a codesign scheme. These environments start from a system-level specification with languages like SDL, C/C++/SystemC, Esterel and Java.

Figure 1 shows a typical codesign flow. The flow starts with an informal specification of the whole system, generally in natural language. This specification forms the basis for analyzing the system requirements and for the high-level description of the system functionality. From the informal specification a first high-level formal description is generated. This description depicts systems functionality, in most cases, as a hierarchical mixed data/control flow diagram, and can be written in one of the above-cited languages. System-level simulation or formal verification achieves validation and exploration of algorithms and systems functionality.

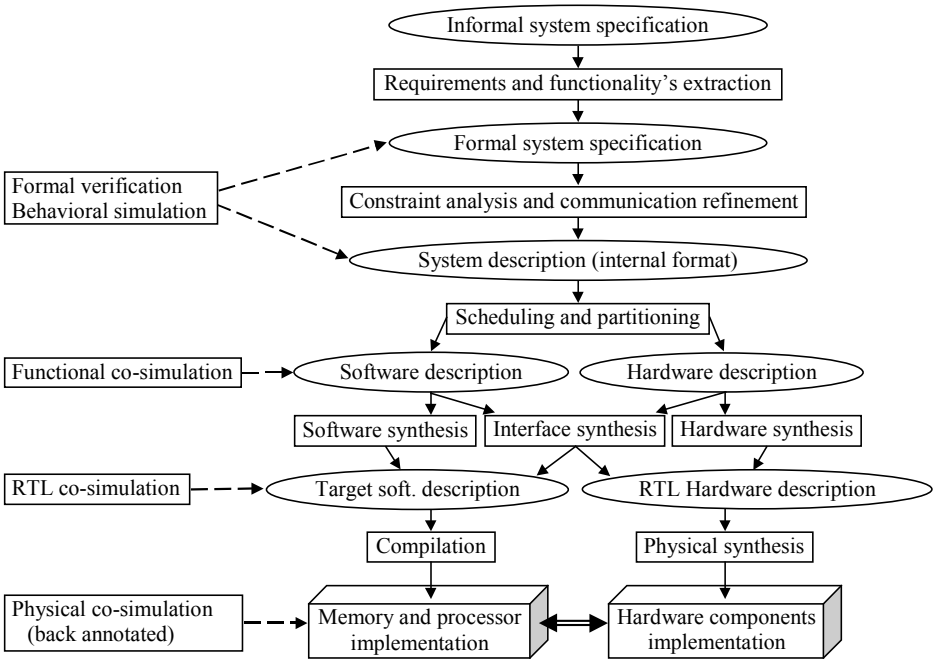

Figure 1 – Hardware/software integrated design flow.

The second step consists in the refinement of high-level inter-module communication, including protocol selection. Some design alternatives are examined to identify those that meet the system constraints, and the architectural choices are made, generally guided by the user. Once architecture is decided, the functional specification is mapped into an abstract architectural model. This model may include one or more processors and others components. At this stage, the system can be viewed as 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 an HDL description. Besides the software-only functions, the software part includes low-level device drivers for interfacing with hardware components. In the same manner, the hardware part contains the interfaces and components to communicate with the software. At this level, the system is validated through functional co-simulation.

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), communication model, and hardware modules interfaces. In this stage, the architectural model is detailed and the clock-cycle validation is obtained through RTL co-simulation.

The system can then be validated through physical co-simulation. After these steps, occurs the prototyping.

From the available system level languages, this work focuses on SDL (Specification and Description Language) [4]. The reason to use such language is the particular interest of the authors in the domain of telecom applications. SDL is widely used to specify such systems. SDL is an object-oriented language defined by ITU (International Telecommunication Union) [4], which allows to specifying systems in a hierarchical way. The system specification starts from a construction called *system*, where functional blocks are inserted. A *block* is a component composed by one or more *processes* an/or other blocks. A process contains a sequential behavior and concurrency is modeled by a set of processes.

The codesign flow proposed in this work starts from an SDL system specification. This specification is done using three SDL environments, Telelogic TAU SDL™ [5], Cinderella SDL™ [6] and ObgectGeode™ [7]. The objective of choosing more than one environment is the evaluation of the state-of-art of these tools and the evaluation of the portability of SDL descriptions. These environments allow system validation by high level simulation. The hardware-software partition is done manually. The hardware synthesis, software synthesis and communication refinement are accomplished with Archimate™ [8], which generates C descriptions for software parts and RTL VHDL descriptions for hardware parts. The final step is the prototyping. The hardware description is accomplished with Leonardo Spectrum™ [9] for logic synthesis and Foundation™ [10] environment for physical synthesis. The software synthesis is not considered here, since our goal is to analyze the advantages and drawbacks of hardware flow description only. The whole systems are targeted on Xilinx Virtex Family FPGAs.

This work is organized as follows. Section 2 illustrates two digital system case studies. The results obtained are showed in section 3. Finally, some conclusions and directions for future work are presented in Section 4.

2 Case Studies

This Section presents the characteristics of two case studies starting form a SDL specification and going to until the FPGA physical synthesis step.

The first case study is an academic example, ***Polygon***, of an algorithm to fill non-concave polygons. The goal of this case study is the exploration of the heavy traffic of messages among system modules. The algorithm receives a set of coordinates, representing a polygon, and generates the horizontal lines to fill the polygon. The Polygon example is typically a dataflow-oriented asynchronous system.

The five SDL processes to implement the polygon-filling algorithm, is shown in Figure 2: (*i*) the processes *delta_x* and *delta_y* perform a subtraction; (*ii*) the *displac_y* process does a comparison; (*iii*) the *displac_x* process executes a division and (*iv*) *point_gen* is a process that generates horizontal lines to fill the polygon.

The second case study is an industrial application, called ***DropInsert***. The *DropInsert* is a telecom system that manipulates an E1 frame [11], dropping and inserting data and/or voice channels from/to 32 time slots (each E1 slot contains 8 bits). The operating frequency is 2048 kHz. This system is characterized by being 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 the available devices. The E1 protocol is essentially asynchronous, the first time slot is used to synchronize the frame operation. The DropInsert implementation of this control-flow asynchronous protocol is synchronous. Thirty concurrent processes were employed to implement DropInsert. Half of these processes are essentially related to bit manipulation.

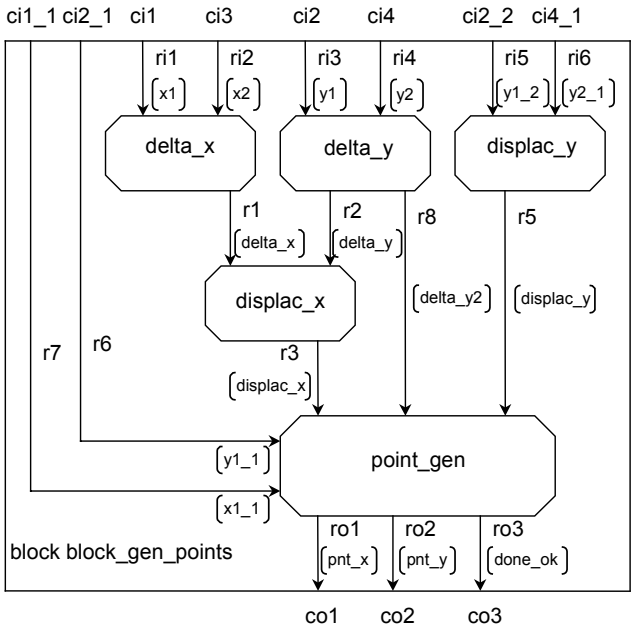

Figure 2 – Polygon-filling algorithm.

3 Results

This Section presents results obtained from DropInsert and Polygon case studies. Both systems are used to analyze the SDL to VHDL translation automatically performed by Archimate. The DropInsert system has two VHDL descriptions: (*i*) *Manual* and (*ii*) *Automatic*. The polygon system has three VHDL descriptions: (*i*) *Manual*; (*ii*) *Single_process* and (*iii*) *Five_processes*. In both case studies, the manual descriptions of DropInsert and Polygon were implemented using traditional digital system tools like manual design entry followed by HDL simulator and hardware synthesis tools. The co-synthesis tool automatically generated the other descriptions from SDL. The difference between *Single_process* and *Five_processes* Polygon description is the number of SDL processes.

3.1 SDL to VHDL Translation

The DropInsert system was originally described with Cinderella SDL [6]. In order to evaluate SDL portability, the Cinderella description was used as an entry for two other SDL environments, Telelogic TAU SDL [5], and ObgectGeode [7]. It was observed that the graphical representation is not the same and the generated textual description, used by the co-design tool, was slightly different among the tools. In addition, TAU SDL is the only tool generating an acceptable code by Archimate. This fact shows SDL is partially portable.

The VHDL code generated by the co-synthesis tool [8] has the following basic template:

The *entity* of each module description depends on the protocol selected in the co-synthesis process. For handshake protocol, for example, a SDL channel is represented by send, acknowledge and data ports in VHDL. Each entity has just one input interface (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 process (see process *delta_y* in Figure 2, which sends data to two processes). This entity's implementation depends on the codesign tool and is not related to the SDL model. The constraint of one input interface generates an input serialization leading to a communication overhead, reducing the performance.

SDL *data types* used to describe digital systems are: (*i*) integer to carry data and (*ii*) non-valued signals to represent events. SDL integer is translated to VHDL integer and SDL non-valued signals is translated to VHDL *bit* data type. In hardware, it is sometimes interesting to restrict the range of data values, to optimize the implementation, e.g. 8-bit buses to carry ASCII characters. As SDL does not support this kind of range limited data types, the resulting VHDL description uses the full integer range (32 bits in VHDL), resulting in area overhead.

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 state machine (FSM) that represents the system behavior. This FSM is generally very large due to the number of input signals implemented with asynchronous protocol. For example, there are 15 states in the FSM of *delta_x* SDL process, depicted in Figure 2, to implement a subtraction. Just one of these states is used to execute the operation and others are used to implement the handshake protocol. This fact also leads to an overhead in area and speed.

Due to the large number of signals in the *sensitivity list* of combinational VHDL processes, the hardware synthesis tool requires a long time to synthesize the modules. For example, the same SDL process *delta_x,* which implements a subtraction, has a *sensitivity list* with 12 signals, being 5 defined as integer.

3.2 Code Generation Analysis

The goal of this Section is to compare some quantitative metrics like number of lines, number of processes and code style. These parameters are used to illustrate the complexity of the generated description, not for performance measurements. Table 1 and Table 2 present the number of VHDL lines and VHDL sequential processes in each description.

Table 1 - Quantitative metrics for the Polygon benchmark.

	Polygon synthesis		
	Manual	Single process	Five processes
Number of lines	473	520	2239
Number of processes	14	1	5

Table 2 - Quantitative metrics for the DropInsert benchmark.

	DropInsert synthesis	
	Manual	Automatic
Number of lines	1278	34.457
Number of processes	28	36

The manual VHDL descriptions were written to optimize area and execution time. In this case, the designer is free to choose the code style, using pipelines, parallel modules, and other design techniques. In an automatic VHDL generation the code style uses the fixed template discussed in Section 3.1. These two tables show two facts: (*i*) the fixed template imposes several constraints to the generated code and (*ii*) these constraints imply in a huge area and performance degradation, to be presented in Sections 3.4 and 0, respectively.

3.3 Comparison of SDL and VHDL Simulations

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 Message Sequence Chart (MSC) formalism. 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 communication processes and the other lines represent the communication, carrying the value of the signal.

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. VHDL simulations give much more detailed information.

A system level simulation, as MSC, should be used to (*i*) abstract some implementation details and (*ii*) to help the designer to validate the system more quickly. The first feature is fully accomplished. Due to the size of the case studies implemented, it was not possible to evaluate the second feature, since the designers were much more used to VHDL simulation than to MSC simulation. However, this feature is expected to become relevant for large systems, since the waveform simulation becomes much more complicated as the system complexity increase.

3.4 Functional Performance

Table 3 shows the number of clock cycles necessary to perform the RTL simulation on three different Polygon implementations. It can be noted that there is a very large communication overhead due to the asynchronous protocol and the interface serialization. As explained earlier, the SDL process model may have several inputs, but, since there is only one input interface in the VHDL entity generated from SDL, the reception of these signals is serialized. Table 3 also shows that when the number of processes in an SDL description increases, the overhead associated to the protocol also increases. The performance is 15 or 30 times slower with regard to the manual description.

The DropInsert system results are not presented in this Table because its dimensions have not allowed the simulation tool to operate upon the automatically synthesized code (due to memory restrictions).

Table 3 - Simulation data of different implementations for the Polygon system.

	Polygon synthesis		
	Manual	Single process	Five processes
Clock cycles	30	530	900

3.5 Synthesis Performance

Table 4 presents some results obtained from the Polygon system. Logic synthesis was performed using Leonardo Spectrum, followed by physical synthesis with Xilinx proprietary tool set, targeting a Xilinx Virtex Family FPGA, the 300-thousand 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 represents maximal operating frequency, as predicted by the physical synthesis tool. The last line represents the necessary time to complete the synthesis on a workstation Sun Ultra10, 333 MHz with 256 MB RAM. The last line is important to show the different descriptions complexity under the point of view of the synthesis tool.

Table 4 - Implementation data of different versions for the Polygon system synthesis.

	Polygon synthesis		
	Manual	Single process	Five processes
Number of CLBs	130	401	1008
Number of D flip-flops	133	417	1095
Operating frequency (MHz)	76.4	70.2	48.3
Time for synthesis (seconds)	33.33	123.07	354.42

Table 4 shows that the manual implementation is 30 times faster than the five processes implementation. If the same performance is required among both implementations, the five processes implementation should increase 30 times its clock frequency. But this is probably not possible to accomplish a working hardware, because the maximum clock frequency estimate decreases 1.58 times in the five processes implementation.

DropInsert results are not illustrated because the available physical synthesis tools (Leonardo and Foundation) could not synthesize all processes due to the number of states. Some processes of automatic VHDL description have more than 500 states, which is 25 times more than the manual description. This gives a measure of how far is the underlying SDL model of computation for hardware synchronous systems, what is confirmed by other similar works like [12].

4 Conclusions

It is possible to enumerate some SDL language advantages, such as (*i*) steep learning curve, as it has a graphical input format; (*ii*) easy system description; (*iii*) possibility to express non-determinism due to its concurrency model; (*iv*) fast system simulation due to the abstract communication model. The main drawback is the lack of constructions to express synchronous systems, like constructions to simultaneously evaluate more than one signal transition.

Although SDL is standard, descriptions created with the available environments are only partially portable. Automatic SDL to VHDL translation is not mature, as can be observed in the result sections. The main problems are the performance degradation, due to the communication protocol serialization (up to 30 times slower), and area increase (up to 8 times). These results show that much more effort is required in system-level synthesis tools research in order to generate hardware descriptions competitive with manually implementation.

5 References

[1] K. VAN ROMPAEY, D. VERKEST, I. BOLSENS and H. DE MAN – Coware, A Design Environment for Heterogeneous Hardware/Software Systems. European Design Automation Conference. Geneve. Sept. 1996.

[2] B. LEE and A. LEE – Hierarchical Concurrent Finite State Machines in Ptolemy. Proc. of International Conference on Application of Concurrency to System Design. Pp. 34-40. 1998.

[3] R. KLEIN – A Hardware Software Co-Simulation Environment. RSP'96. IEEE CS Press. Pages 173-177. Miami, 1996.
[4] ITU-T – Programming Languages: SDL Methodology Guidelines, SDL Bibliography. ITU-T Recommendation Z.100 – Appendices I and II, March 1993.
[5] TELELOGIC TAU – A SDL tools for real time systems development. Telelogic, Inc. Available at homepage: http://www.telelogic.com, 2002.
[6] CINDERELLA – A visual SDL tools for systems development. Available at homepage: http://www.cinderella.dk, 2002.
[7] TELELOGIC OBJECTGEODE – A SDL tools for real time systems development. Telelogic, Inc. Available at homepage: http://www.telelogic.com, 2002.
[8] AREXSYS ARCHIMATE – A SDL synthesis tool. Arexsys, Inc. Available at homepage: http://www.arexsys.com, 2002.
[9] LEONARDO SPECTRUM. Available at homepage: http://www.exemplar.com, 2002.
[10] XILINX FOUNDATION – A system synthesis tool. Xilinx, Inc. Available at homepage: http://www.xilinx.com, 2001.
[11] NEY L. V. CALAZANS, FERNANDO G. MORAES, CÉSAR A. M. MARCON, VITOR H. BLAUTH, RONALDO VALIATI, ÉDISON MANFROI – Effective Industry-Academia Cooperation in Telecom: a Method, a Case Study and Some Initial Results. In: XIX Simpósio Brasileiro De Telecomunicações, Fortaleza. 2001.
[12] ANNETTE MUTH, GEORG FÄRBER – SDL as a System Level Specification Language for Application-Specific Hardware in a Rapid Prototyping Environment. ISSS 2000. Pp. 157-162. Sept, 2000.