Enhancing a VHDL Based Design Methodology with Application Specific Data Abstraction

Lars Lindqvist

DSC Communications A/S
NKT Allé 85
DK-2605 Brøndby
DENMARK

E-mail LLq@ne.dk

Center for Integreret Elektronik,
Institut for Datateknik, DTU
DTU - Bygning 344
DK-2800 Lyngby, DENMARK

E-mail LLq@id.dtu.dk

Abstract: VHDL has successfully been introduced into the design methodology for VLSI ASICs. This paper describes a high-level data abstraction and supporting tool that enhance this methodology in the telecommunication application domain. A significant performance gain was obtained by introducing the data abstraction outside the VHDL simulator. The enhanced methodology has been used in current ASIC designs with good results.

I. Introduction

The introduction of VHDL in the development methodology for VLSI ASICs shifted the design paradigm from capture-simulate to describe-validate-synthesize [1]. Much effort has been put into the research on synthesis [2]. As a result, today synthesis systems have sufficient quality for designing ASICs, when the utmost performance of the technology is not needed [3]. When ASICs are the end product, for example for microprocessor vendors, the deterioration caused by synthesis tools is not always acceptable. The introduction of synthesis raised the level of functional abstraction from gate-level schematics to RTL. This paradigm shift made it possible to use a longer part - of the overall shorter - development time on the function rather than on implementation aspects. I.e., instead of drawing schematics more time can be spent on validating the behavior of the design.

This paper concentrates on validation of ASIC designs at the RTL and gate-level. This work is related to the telecommunication application domain, specifically SDH[1] [4] for telecommunication networks. In this application domain RTL VHDL is quite convenient for capturing the functional aspects of the design. The natural data types bit and bit_vectors [5] are convenient for the functional aspects but are cumbersome to use for representation of data from the environment. This is caused by the large amounts of data needed. Thus, seen from a designer's viewpoint, the main problem is presently handling stimuli and response data.

SDH transmission networks are the backbone for transport of all kinds of telecommunication traffic. The novelties of SDH compared to the previous PDH[2] are inherent management and surveillance capabilities. These features have caused increased interest in design methodologies for VLSI ASICs. SDH network elements process huge amount of data. An SDH terminal multiplexer can have an aggregate rate of 2.5 Gbit/s, whereas the more complex cross-connects process hundreds of Gbit/s. Thus, even if the basic processes are still multiplexing and demultiplexing, the new SDH functions has introduced functionally more complex processing in the equipment.

II. ASIC Design Methodology

This section elaborates on where in the ASIC design flow the data manipulation is needed. In a strict Phase-Review methodology [6] each phase is performed as a separate task. The next phase is not started before the previous phase has been concluded with an approval at the final review.

Figure 1 shows an implementation of the Phase-Review process for ASIC design. The situation is idealized; it does not capture the inevitable iterations or the concept of concurrent engineering [7]. For the purpose of this paper the model is, albeit, sufficient to demonstrate the benefits of the developed methodology and tools.

Figure 1. Phase-Review methodology for ASIC development.

RTL validation is required to ensure consistency between ASIC specification and RTL VHDL(Review II). Furthermore the RTL validation is the basis on which the gate-level implementation is validated (Review III).

The following section elaborates on the options available for design reviews and especially reviews II and III in Figure 1. Also, the reason for using validation and not verification is discussed in the next section.

III. Design Reviews

There are basically two methods for checking consistency; formal verification and simulation [8,9]. Formal verification requires a description of the specification and

[1] SDH, Synchronous Digital Hierarchy, is used in this paper, but the ideas and results can easily be transferred to SONET.

[2] PDH, Plesiochronous Digital Hierarchy

implementation in a formal system. The formal system needs to include proper behavioral and temporal abstractions. The behavioral abstractions should, for instance, cope with the left-outs in the specification; i.e., the don't care set. The temporal abstractions should handle, for example, the case of a sequential algorithm specification versus its scheduled implementation. Furthermore an automatic proof-system is needed. The proof-system should be capable of proving certain properties of a design (specification verification) and to prove equivalence between design descriptions (implementation verification). It should perform the proofs automatically and find a descriptive counter-example in case of contradiction. Such a formal system and accompanying tool are not available yet, as for all but the simplest designs. Nevertheless, the theoretical basis is on its way, and formal verification will certainly play a role in future design methodologies when the tools and interfaces mature.

Simulation requires an executable model of the design; e.g., a VHDL simulator and a VHDL description of the design. VHDL has been selected as basis for the development methodology, because of the broad range of tools available. The simulator and the model form the basis but are only useful with a suitable set of stimuli and proper analysis of the response. A suitable set of stimuli is the first hurdle. In theory for a combinational circuit all possible input combinations are needed and for sequential circuits all possible input combinations with the circuit in all possible states are required. This is of course not realistic for all but the smallest designs, and is the reason for the interest in formal verification. Sufficing with less than the theoretical requirement is the working solution today. This means that only validation and not verification is done. Thus, insight by the designers is required to select the stimuli set.

Having selected the stimuli and run the simulation the second hurdle occurs, analyzing the response. For small response sets this can be done by inspection or better by comparing with an expected response set. Inspection is based on human operation and thus requires a compressed representation of the information. Manual inspection of large response sets is not a secure way of detecting errors. Using the comparison approach requires a reference. This reference can be a direct reference where a one-to-one equivalence can be determined, but this demands creation of such a reference set. Indirect comparison is done by checking properties in the response according to a reference set with a higher level of abstraction.

The consequence of the above is that simulation of VLSI ASICs for SDH at the RTL and gate-level requires a substantial amount of input data and generates a correspondingly large set of output data. A solution to the problem of generating input for simulations and analyzing simulation output will be discussed in the next section.

IV. SDH DATA MODEL

The solution to handling the large amounts of data is abstraction. A formalism for describing SDH data has been established. ETSI[3] recommendations [10] define the SDH data structures and network element functions in a layered fashion much like a software protocol stack. In this work, a set of SDH tokens and values have been defined based on the ETSI terminology. Tokens correspond to the information types present in each layer. Values define the actual information. To incorporate the notion of time, rules for specifying SDH data by sequences of SDH tokens have been defined. The resulting ASCII files are called SDHdm scripts. The SDHdm scripts and the supporting program SDHdm constitute the SDH data model.

The SDHdm program performs translation between SDHdm scripts and SDH data. The reason for not using the SDHdm scripts directly as stimuli for the VHDL simulations was ease of implementation and diversity. Lexical analyzers and parsers are easily created with *lex*/*yacc* tools. The implementation also benefits from an object oriented implementation; e.g., multiple instances of a layer are created as multiple object instances of the corresponding layer class. The diversity is obtained by having both a C++ (SDHdm) and a VHDL interpretation (the design) of the SDH recommendations. Because the source code differs in both structure and syntax, errors caused by cut-and-paste are eliminated; i.e., error replication is avoided. Finally, execution of binaries compiled directly from C++ was expected to be faster than VHDL simulation. The experimental results listed later confirm this assumption.

The paradigm for using the SDH data model is shown in Figure 2. The expand process takes an SDHdm script as input and produces the corresponding stimuli set in bit or byte format as needed. The response files generated from either RTL or gate-level simulations are interpreted by a condense process. This process condenses the response file by translating the data to the SDHdm script syntax.

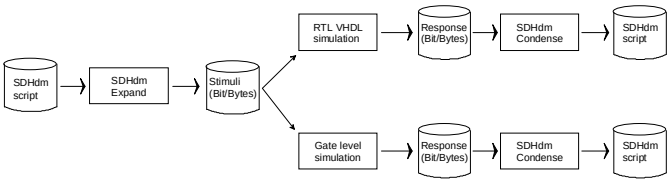

Figure 2. Paradigm for using the SDH data model.

V. EXAMPLE

A part of SDH, termed multiplex section adaptation, is used as an example of the application of the SDH data model. The basic unit in SDH signals transmitted between network elements is the STM-1[4] frame. An STM-1 frame is shown in Figure 3. It has 9 rows and 270 columns. Each position contains one byte. Rows are transmitted one after

[3] ETSI, European Telecommunication Standards Institute

[4] STM, Synchronous Transport Module.

another. The STM-1 is synchronous to the network element clock, which generates the STM-1. In SDH the STM-1 frame contains one VC-4[5]. The VC-4 is not necessarily synchronous to the network clock. To absorb phase and frequency variations between the VC-4 clock and the STM-1 frame clock, the VC-4s are floating within the AUG[6] area of the STM-1. As shown in Figure 3, a pointer in the AUG points to the beginning of a VC-4. The pointer is always in the same position of the STM-1 frame. Note that a VC-4 will normally be divided over two STM-1 frames. The pointer in the current STM-1 marks the beginning of a VC-4 that ends just before the position pointed to by the pointer in the following STM-1 frame.

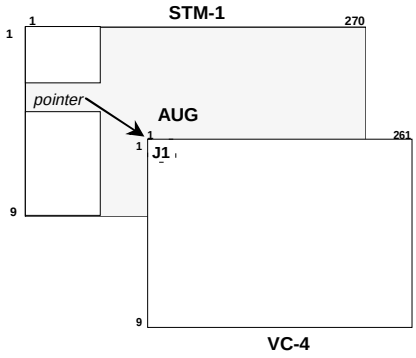

Figure 3. STM-1 and VC-4.

The first byte in the VC-4 is called J1, and contains one byte of a multiframed trail trace identifier. A trail trace identifier is a 16 byte string in E.164 format [10]. The AUG pointer can take values from 0 to 782 corresponding to the 783 possible locations of the J1 byte within the AUG. Only every third position in columns 10 to 270 of the STM-1are legal J1 positions.

One way to test that the RTL VHDL model of the pointer interpreter (PI) finds the correct position of the VC-4 is: Generate stimuli that contain STM-1s with embedded VC-4s that have a recognizable trail trace identifier. Since the trail trace identifier is 16 bytes at least 16 frames are required. Sixteen frames contains 311 kbits. This test will in the following by referred to as the *small test.*

The pointer value is continuously adjusted based on buffer fillings in the network element generating the STM-1. The algorithm for adjusting the pointer values prescribes a minimum spacing of 3 frames between pointer operations. Increment and decrement pointer operations adjust the pointer value and the VC-4s are moved correspondingly in the AUG by inserting stuffing or extra data, respectively. To perform a thorough test of the pointer interpreter all pointer values should be entered via both increment and decrement operations. Such a test requires 2*4*783 frames; i.e., 6264 frames containing 122 Mbits. This test will in the following by referred to as the *large test.*

[5] VC-4, Virtual Container of 4'th order.

[6] AUG, Administrative Unit Group.

By now it should be clear that generating these stimuli sets by hand is out of the question. Thus, prior to the conception of the SDH data model, simulation of a connected pointer generator (PG) and pointer interpreter was the only solution to perform the validation. But the pointer generator will not during normal operation perform the required pointer operations. Even if forced to do the required operations, still no VC-4 data will be included without a VHDL model of the VC-4 generator. Furthermore the back-to-back interface of a pointer interpreter and pointer generator through a pointer buffer (PB) should also be checked. The complete test scenario is shown in Figure 4.

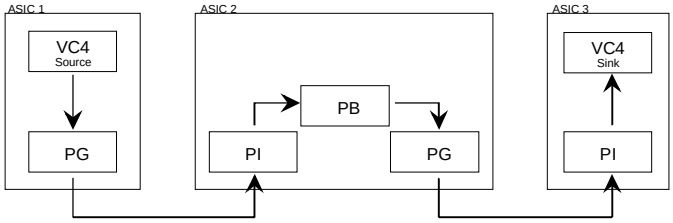

Figure 4. Test scenario for pointer interpreter and generator.

The SDH data model breaks the connections between the ASICs as shown in Figure 5.

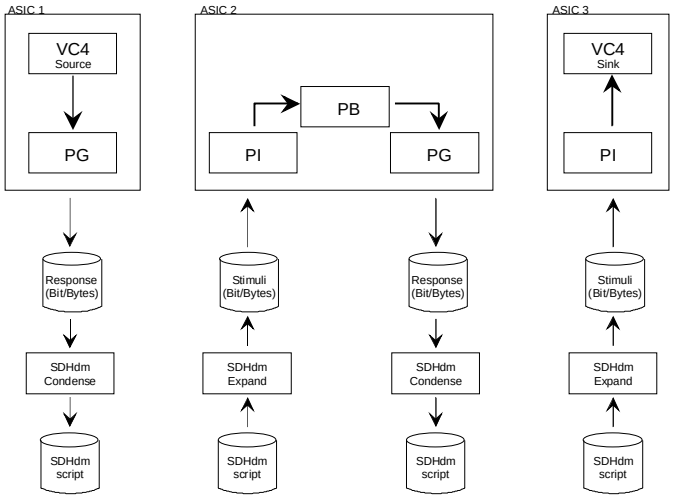

Figure 5. Test scenario for pointer interpreter and generator using the SDH data model.

The solutions for the two mentioned tests are quite simple using the SDH data model. The input script for the *small test* is shown in Figure 6

Expanding the input scripts results in two data files in byte format. As can be seen in the Table 1 the raw size ratios between input script and the stimuli file range from 10^2 to 10^5. The generated stimuli can be used as input for simulation of both ASICs 2 and 3 in Figure 5.

Condensing the stimuli files generated by the above expand operations corresponds to analyzing the response from simulations of ASIC 1 and 2 in Figure 5. The results are presented in Table 2. The condense ratios are not as impressing as the expand ratios, but the ratios only cover the size relationship between the two representations.

Table 1 Expand results				
Test	Expand script [bytes]	Stimuli file [bytes]	Expand ratio	SDHdm execution time[7] [CPU sec.]
Small	209	131,400	629	0.6
Large	494	45,851,300	92,820	199.7

Table 2 Condense results				
Test	Response file [bytes]	Condense script [bytes]	Condense ratio	SDHdm execution time[7] [CPU sec.]
Small	131,400	587	224	2.5
Large	45,851,300	449,525	102	1006.1

The post processing and interpretation are much easier on the condensed version of the data than on the actual bits and bytes. Figure 6 shows the condense result from the *small test*. For the small test the main check is that the J1 eventually becomes " VC-4 trail trace". For the large test the J1 remains " VC-4 trail trace" while the pointer values and types, indicated by the AUPTR token, change as expected.

Input to Expand

```
{
  MS-AI = NORMAL ;

  S4-CI() = NORMAL ;
  J1() = " VC4 trail trace" ;

  {
    AUPTR() = NDF ;
    AUPTR() = 0 ;
  } FOR 1 FRAME ;
  {
     AUPTR() = NORMAL ;
  } FOR 17 FRAMES ;

} FOR 18 FRAMES ;
```

Output from Condense

```
{
  MS-1-AI = NORMAL ;
  AUPTR(1) = UNKNOWN ;
  ...
} FOR 2394 BYTES
{
  ...
} FOR 2394 BYTES
{
  MS-1-AI = NORMAL ;
  AUPTR(1) = NORMAL 0 ;
  {
    ...
  } FOR 33669 BYTES
  {
    S4-CI(1) = NORMAL ;
    J1(1) = " VC4 trail trace";
  } FOR 3915 BYTES
} FOR 38304 BYTES
```

Figure 6. SDHdm scripts.

By using the standard UNIX utility *grep* it is possible to filter the condense scripts very effectively. In this case searching for the pointer token AUPTR and the trail trace identifier token J1 is useful. Extending the filtering to remove NORMAL pointers and J1s with the expected values further reduces the data to inspect. Table 3 lists the file sizes after post processing with *grep*.

Table 3 Post processed condense results				
Test	Filter	Extracted [bytes]	Condense ratio	Grep script execution time[7] [CPU sec.]
Small	J1	139	945	1.0
	AUPTR	25	5,256	
Large	J1	139	329,900	10.5
	AUPTR	44,530	1030	

The execution times for the SDHdm software are considerably shorter than for a corresponding VHDL simulation. For example testing ASIC 2 can be done as indicated in Figure 4 using VHDL models for all three ASICs. The alternative shown in Figure 5, using the SDHdm to generate and analyze data, is much faster. Typical simulations of the ASICs in this work at RTL take in the order of minutes per frame. The duration of course depends on the number of blocks included in the simulation but reflects actual experience. Therefore the execution times in Table 1 and Table 2 indicate a performance gain in the order of 10^3.

[7] Test executed on a Sun SPARCstation 20 model 61.

VI. Conclusions

To evaluate the applicability of the SDH data model, it was introduced to ASIC designers working on commercial projects. The designers were developing ASIC of sizes up to several hundred thousands gates. The confrontation of the model with a real life design scenario of course resulted in numerous additional features, but also led to a very thorough evaluation of the applicability. By now simulations of all but the simplest blocks use the SDH data model extensively. Part of the success was due to the abstraction of data it self. Designers were able to alter their stimuli for even large simulations in a few minutes, and to analyze large response sets more thoroughly than before. Another cause of its success was the point of diversity as mentioned before. Because the software represents an independent implementation of the recommendations a number of bugs were revealed.

The flexibility and performance of the SDH data model make it possible to validate designs more thoroughly and faster than before. This is a key factor in securing the design process and to reduce time-to-market for products.

The SDH data model covers the SDH relevant parts of the multiplexing hierarchy. It would be fairly simple to extend the model to cover SONET. The basic idea of a domain specific data model should also be applicable to other domains than telecommunication. The conclusion is thus, that a significant performance gain can be obtained by handling high-level data abstraction outside the VHDL simulator.

VII. References

[1] D. Gajski and L. Ramachandran. Introduction to high-level synthesis, *IEEE Design & Test of Computers*, Winter 1994.

[2] R. Camposano and Wayne Wolf, editors. *High Level VLSI Synthesis*, Kluwer Academic Publishers, 1991.

[3] L. Lindqvist. "Evaluating the applicability of current VHDL synthesis tools to an industrial top-down development procedure", in *Proceedings of VHDL International Users' Forum Fall 93 conference*.

[4] M. Sexton and A. Reid. *Transmission Networking: SONET and the Synchronous Digital Hierarchy*. Artech House, 1992.

[5] IEEE. *IEEE Standard VHDL Langauge Reference Manual*, 1987 edition.

[6] R. E. Kmetovicz. Five steps to on-time software development. In *Engineering Software (a supplement to Electronic Design)*, October 3, 1994.

[7] Donald E. Carter and Barbara Stilwell Backer. *Concurrent engineering: The product development environment for the 1990s*, Mentor Graphics, 1991.

[8] J. Staunstrup. *A Formal Approach to Hardware Design*, Kluwer Academic Publishers, 1994.

[9] M. Yoeli, editor. *Formal Verification of Hardware Design*, IEEE Computer Society Press, 1990.

[10] ETSI. prETS 300 417 (former draft ETS DE/TM-1015).