

Design Procedure Based on VHDL Language Transformations

LÁSZLÓ VARGA, GÁBOR HOSSZÚ* and FERENC KOVÁCS

Department of Electron Devices, Technical University of Budapest, H-1111 Budapest, Goldmann Gy. t. 3., Hungary

(Received 20 July 1999; Revised 10 March 2000)

One of the major problems within the VHDL based behavioral synthesis is to start the design on higher abstraction level than the register transfer level (RTL). VHDL semantics was designed strictly for simulation, therefore it was not considered as high-level synthesis language. A novel synthesis procedure was developed, which uses the methodology of high level synthesis. It starts from an abstract VHDL model and produces an RTL VHDL description through successive language transformations while preserving the VHDL standard simulation semantics. The steps of the synthesis do not use graph representation or other meta-language, but apply the standard VHDL only. This VHDL representation is simulatable and accessible, functional verification can be performed by simulation at any time, and the simulation results can be used to guide the synthesis process. The output VHDL format is suitable to continue the design flow with RTL based synthesis tools.

Keywords: Language transformations; High-level synthesis; VHDL; Simulation

INTRODUCTION

The increasing complexity of integrated circuits and the aggressive time-to-market requirements have caused growing interest in the synthesis of digital systems from behavioral level specifications. Behavioral synthesis offers significant productivity increase by raising the abstraction level of digital design, and improvements in design quality by exploring high-level trade-off. A behavioral description represents an algorithmic specification of the functionality of the digital system. It contains no structural information, and the physical type “time” is not allowed, since the structural implementation, and the cycle-by-cycle behavior are introduced by the high-level synthesis tool.

The task of the synthesis is to convert the behavioral description into a structural, register transfer level architecture that is based on two separated but related parts: data path and control block [1]. The data path consists of macro blocks such as functional units (such as multipliers, shifters, adders, etc.), storage units (such as registers, memories, etc.), and interconnection units (such as multiplexers and buses). Behavioral synthesis tools typically compile a behavioral description into a suitable intermediate format, such as control-data flow graph (CDFG), and use the CDFG all along the synthesis process. The result of the behavioral synthesis process

provides a register transfer level (RTL) specification of the circuit typically described by a hardware description language (HDL). Among the different HDLs for digital circuit design, VHDL [2] is the most widely used and standardized. VHDL can capture the design at several abstraction levels and conveniently represents both the behavioral specification and the RTL design.

Automated hardware synthesis starting from the VHDL RTL description is well established. There are many commercial CAD tools starting with the RTL structural specification as input and yielding the complex documentation for fabricating the circuit. However, there are still open problems in starting the VHDL based design on a higher abstraction level than the RTL. The main difficulty lies in the fact that VHDL semantics are designed strictly for simulation, therefore high-level synthesis with VHDL has not found a common ground among the researchers, but other formal description techniques than VHDL have been considered (such as control/data flow graphs, timing graphs, and structural graphs).

To overcome the obstacle of high-level synthesis of digital circuits with VHDL, we have developed a series of transformation steps, which converts each design representation into a VHDL description utilizing the multi-level description capability of VHDL while preserving the VHDL standard semantics during synthesis. As a result,

*Corresponding author. Tel.: +361-463-4034. Fax: +361-463-2973. E-mail: hosszu@nimrud.eet.bme.hu

the digital system design can be simulated together with the behavioral model and/or RTL model of the macro blocks, which are also described in VHDL. Simulation is typical validation technique of current industry practice. Fast and accurate feedback in the early design flow is a key issue. A pure VHDL based high-level synthesis methodology is proposed in [3]. The design procedure presented in this paper starts from the abstract high-level functional model and produces an RTL description through successive language transformations. The main benefits of this approach are the following:

1. The purely behavioral VHDL provides consistency, and takes advantage of coarse-grained high level parallelism in the input language.
2. Only the VHDL as design representation is used during the synthesis process. This VHDL representation is simulatable and accessible to the designer. The behavior of a design can be verified at an early design phase, functional verification can be performed by simulation at any time, the simulation results can be used to guide the synthesis process.
3. Simulation runs at all levels of abstraction. The pre-simulation of the design behavior, the simulation in an intermediate step of synthesis and post-simulation after the synthesis are straightforward. The same mechanism/tool/environment can be used, and multi-level simulation is achieved. The design to simulate can be inserted into the original test bench without altering the test bench. The only modification in some cases that the designer will have to retime the test bench.
4. The methodology gives the designer a powerful set of tools, and preserves the designer's ability to provide control to the available options of synthesis algorithms in order to perform efficient design space exploration by interacting with the high-level synthesis process.

THE METHOD OF LANGUAGE TRANSFORMATION

There are three main tasks to be performed in transforming a high-level behavioral specification into an RTL architecture. The first is the determination of the cycle-by-cycle behavior of the design by scheduling of each operation into control steps. The second one is the allocation, which decides on the number and type of hardware units that will be used to implement the behavioral description. The third task is the binding, which assigns each operation to one of the allocated functional unit, each variable to one of the allocated register, and creation of interconnections to complete the data path.

For synthesis, two kinds of information are necessary: a behavioral specification and a functional unit library. The input behavioral specification is written in VHDL, which describes an algorithm that the designer wants to

implement on a chip. This input is much more abstract than a VHDL structural description that would be used as the input to a logic synthesizer. The functional unit library consists of the available elementary units, and provides information about different parameters (attributes) of the functional units [4]. Each unit in the library is described by a VHDL entity. The entity declaration specifies the name and the input/output port structure of the component together with possible parametrized information. An architecture body is used to specify the semantics of the component.

Our synthesis methodology starts from a behavioral VHDL description and reaches a RTL description through the step-by-step transformations using the tools of the high-level synthesis. The generated RTL architecture is based on constraints imposed by the designer. The designer can perform architectural design trade-off by executing different algorithms and evaluates its result. The simulation results, the performance, the estimated implementation cost, or a specific objective function (etc.) can be used to guide the synthesis process by interacting with the design algorithms in order to obtain better results. In addition to the algorithmic optimization techniques, manual optimizations are also possible by altering the VHDL code.

We have made some basic assumptions concerning the input description style. The circuit to be synthesized consists of a VHDL process, which will be mapped to hardware in the form of a data path and a control finite state machine [5]. In the next section we describe the steps of the proposed methodology in details.

THE STEPS OF THE DESIGN PROCESS

The first step is the creation of the behavioral description. The behavioral VHDL description is a purely algorithmic description, which contains high-level operations. Most algorithms have a common behavior in that they read some input data, perform some calculation, produce some output data and start over again. The behavioral description can be written in many different styles, which are semantically equivalent, but syntactically different, and designs synthesized from semantically equivalent and syntactically different descriptions may differ in quality [6]. We provided with some basic rules concerning the input description style, in order to describe models that can be efficiently synthesized in hardware. The overall structure of the design model comprises entity declaration and architecture body. A process body embodies the data and control flow of the design specification for synthesis purposes, which will be mapped to hardware in the form of a data path and a control finite state machine (FSM). We assume that the input data is loaded when the "start" signal is issued. After completion of the calculations, when the result are at the output ports, the circuit issues the "ready" signal. This control construct provides the entity level timing for the

```

program example (a, b, c, d : in integer;
  y : out integer) is
  variable va, vb : integer;
begin
  va := a + b;
  if c > d then
    vb := c - d;
  else
    vb := d - c;
  end if;
  va := va * vb;
  y := va;
end;
end program;

```

FIGURE 1 Abstract functional model.

design. The designer can write such a model in VHDL, or can use a supplemented VHDL subset to derive a simulatable architecture. A supplemented VHDL subset called Abstract Functional Model (AFM) was developed to provide a restricted design style to the designer corresponding to our basic assumptions.

We demonstrate the methodology in detail with the aid of an example. Consider the arithmetic expression $y = (a + b) \times \text{abs}(c - d)$. An example AFM input file for this expression we use throughout this paper is shown in Fig. 1. The high-level operations are respect to the functional units in the library. We use integer data type, however any types that can be mapped onto finite size bit patterns are allowed. A tool was developed which converts the Abstract Functional model into a VHDL description. The tool generates also a test-bench skeleton along with the design entity. The test-bench skeleton consists of a declaration of the test-bench entity and its architecture, which consists of a signal component corresponding to the design. The I/O ports of the design entity will be connected to signals, which will be used to capture the simulation results. The designer can fill in some waveform specification for the input ports of the design entity so that specific test patterns can be used to check the desired functionality. The test-bench skeleton also includes a clock signal generator and the “start”, “ready” control construction to control the simulation process.

Our initial VHDL model, called High-level Behavioral Model (HBM) (see Fig. 2.) can be automatically derived from the AFM. Predefined “start”, “ready” control construction is used in the description of HBM to timing the operation of the design. Variable equivalents for input and output ports are created. The process is started with a WAIT statement. The inputs are read, the algorithm is calculated using variables, and the result is assigned to the variable equivalents of the outputs. Finally, the output ports are assigned from the variable equivalents. This VHDL description is simulatable. The designer can perform functional verification of VHDL specifications before synthesis to check if the description has the intended behavior. The HBM will be mapped to an RTL description in the form of a data path and a control FSM through successive transformations. Each transformation

```

entity example is
  port (pa, pb, pc, pd : in integer;
    py : out integer;
    start : in bit;
    ready : out bit);
end design;
architecture HBM of example is
begin process
  variable a, b, c, d: integer;
  variable va, vb : integer;
begin
  wait until start = '1';
  ready <= '0';
  a := pa; b := pb;
  c := pc; d := pd;
  wait until start = '0';
  va := a + b;
  if c > d then
    vb := c - d;
  else
    vb := d - c;
  end if;
  va := va * vb;
  py <= va;
  ready <= '1';
end process;
end HBM;

```

FIGURE 2 High-level behavioral model.

incrementally augments the model with structural information until a complete RTL design has evolved.

The first transformation step separates the data path and the control part of the design. (See Fig. 3.) The separation makes it possible to handle the basic operations by algorithmic synthesis tools. The control of the timing will be concentrated in the process, whereas the data path will be collected into a block outside of the process. The block is “guarded”, wherein “guarded” concurrent signal assignments are present. The concurrent signal

```

architecture VTB of example is
  signal a0, b0, c0, d0 : integer;
  signal va0, va1, vb0 : integer;
  signal cnt0 : bit;
  type pltype is (empty, token);
  signal df0 : pltype;
  constant t1 : time := 50 ns;
begin process
  variable a ,b ,c ,d : integer;
  variable va : integer;
begin
  wait until start = '1';
  ready <= '0';
  a := pa; b := pb; c := pc; d := pd;
  wait until start = '0';
  a0 <= a; b0 <= b; c0 <= c; d0 <= d;
  df0 <= token; wait for t1; df0 <= empty;
  va := va1; py <= va;
  READY <= '1';
end process;
dfb0 : block (df0 = token)
begin
  va0 <= guarded add (a0, b0);
  cnt0 <= guarded comp (c0, d0);
  vb0 <= guarded sub (c0, d0) when cnt0 = '1'
    else sub (d0, c0);
  va1 <= guarded mul (va0, vb0);
end block;
end VTB;

```

FIGURE 3 Value-trace block.

```

dfb1 : block (df1 = token) begin
  va0 <= guarded add (a0, b0) after 20 ns;
  cnt0 <= guarded comp (c0, d0) after 20 ns;
end block;
dfb2 : block (df2 = token) begin
  vb0 <= guarded sub (c0, d0) after 20 ns when
    cnt0 = '1' else sub (d0, c0) after 20 ns;
end block;
dfb3 : block (df3 = token) begin
  val <= guarded mul (va0, vb0) after 40 ns;
end block;

```

FIGURE 4 Scheduling and allocation.

assignments statements describes the data dependency among the operations. The targets of concurrent assignments are ordered to the values of variables of the HBM. The assignments are executed if the “guard” expression changes to a true value or if the “guard” expression is true and in the same time there is an event on the signal in the right side. The “guard” expressions are controlled by the instructions of the process body. The functions in the right-side expressions of the transactions cover their implementations among the functional unit library. For faster simulation, however, the according behavioral model can be used, too. For example, the behavioral model of the addition function may be as simple as the VHDL plus operator, but in the case of non-simple operator, at least its behavioral function is required. We call this model as Value Trace Block (VTB) [7]. The signals had to be declared for the data exchange among the process and block, respectively. The type *pltype* is defined as an enumerated type with the block control elements *empty* and *token*. The signal *df0* is used to control the block *dfb0*. High-level synthesis systems generally have the CDFG behavioral representation. The graph representation has a special format, they do not provide the possibility of simulation, or provide it in an adjunct simulation environment to a high-level synthesis system, which is different from an HDL environment. The VTB model is given in VHDL, it can be simulated, and describes the data- and control-flow with the same expressivity as the CDFGs.

Scheduling of each operation and functional unit allocation are performed in the second transformation step. The result of this transformation step is shown in Fig.

4. (The control description is omitted for brevity.) Each block represents one control step in the schedule. The operations scheduled in different control steps will be placed into different blocks. The process is responsible of the scheduled start of each block. The after clause in a signal assignment represents the delay times of each operation, if its behavioral model is used. Most of the well known scheduling and allocation algorithm can be used. The designer has the task to choose among the available algorithms and its parameters according to the actual constraints. The designer set the demands for the restarting time, has the specification of the number of architectural components such multipliers, adders, etc. that are allowed in the design along with their delay times, but other constraints, such as testability or power issues [8] can be considered, too. According to this, there are several behavioral architectures among this transformation step. If the actual design do not satisfactory, re-scheduling or behavioral transformations [9] can be applied in this step to meet the design constraints. Different implementations of the operations are distinguished. A component with many different implementations can exist simultaneously in the library which have different characteristic, and selection of components during the synthesis process is possible. For example, the selected implementation (such as serial ripple-carry or parallel look-ahead-carry) of an addition function is distinguished according the name of the implementation in the library.

After scheduling and allocation, the behavioral model of the controller is created using state register denoted as FSM. In the third transformation step register and multiplexer allocation and final binding are performed. (See Fig. 5.) The allocated registers are connected to the operator units (through multiplexers if necessary) to complete the data path. We use the point-to-point interconnection model, however, interconnections, which are not used simultaneously, can be merged into buses to reduce the required number of multiplexers. For synthesis, the behavioral models of operators are substituted with their selected library implementation.

After completing the data path, we substitute the behavior model of the control part with an FSM model. The FSM is generated according to the data path. This is accomplished by inserting a specific part into a general

```

DATAPATH : block
begin
  a0 <= pa when fsm = st_load else a0;
  b0 <= pb when fsm = st_load else b0;
  c0 <= pc when fsm = st_load else c0;
  d0 <= pd when fsm = st_load else d0;
  py <= val;
  va0 <= add (a0, b0) after 20 ns when fsm = st1 else va0;
  cnt0 <= comp (c0, d0) after 20 ns when fsm = st1 else cnt0;
  vb0 <= sub (m1, m2) after 20 ns when fsm = st2 else vb0;
  val <= mul (va0, vb0) after 40 ns when fsm = st3 else va0;
  m1 <= c0 when cnt0 = '1' else d0 when cnt0 = '0' else m1;
  m2 <= d0 when cnt0 = '1' else c0 when cnt0 = '0' else m2;
end block;

```

FIGURE 5 RTL architecture.

RTL template using interactive tools. The data path and the controller model together form the RTL implementation of the input specification. This RTL code is suitable to feed into commercial logical synthesis tools in order to obtain a gate-level implementation of the circuit.

IMPLEMENTATION

The language transformations described in the previous chapter can be automated by elaborating and following an appropriate strategy, which is applied in a program system called Synthesis Based on Language Transformation (SYLANT). The SYLANT software generates step-by-step the required RTL model by starting from the initial description. Its output model, and the transformation step from the output model of the previous step to its output model define each step. A program in the SYLANT system parses the VHDL construct according the previous output model, performs the task of the transformation step, and outputs the result according to its output model. The transformation from AFM to HBM, and from HBM to VTB model typically requires no interaction from the designer. These transformations are well automated and correct by construction. The most significant part of the system is the scheduling and allocation algorithm, which requires interaction in order to meet the requirements. It may be cycles in the synthesis flow, iterative transformations can be applied to obtain better result. The methodology is open to apply many scheduling and allocation algorithm. A power aware scheduling technique [10] was adopted and used on the example presented in this paper. The register and multiplexer allocation and final binding step are automated, however, they can be performed under the control of the designer.

CONCLUSIONS

We presented a method of high-level synthesis from behavioral VHDL, which uses language transformations to reach a structural architecture while preserving standard VHDL simulation semantics. The most significant advantage of the presented method is, that the steps of the synthesis do not use graph representation or other meta-language, but apply the standard VHDL only. The method does not take the control of the architectural design process from the designer and does not restrict the designer to produce the best possible design. The intermediate design representations are in fact VHDL models, their simulation can be performed to verify the functionality and, more importantly the timing of the system. The designer can use the simulation results to guide the synthesis process by interacting with the synthesis algorithms which gives an opportunity for creative exploration in the design space. In addition to the algorithmic optimization, manual optimizations are also

possible. Another advantage of the language transformation method is that the generated RTL VHDL output can be used directly as input to a logic synthesis tool which accepts VHDL as input specification language.

Acknowledgements

The authors wish to thank to P. Keresztes for his encourage of the research. This work was supported by projects No. T 029331 and No. T 023963 of the Hungarian Academy of Sciences.

References

- [1] Gajski, D. and Ramachandran, L. (1994) "Introduction to high-level synthesis", *IEEE Design and Test of Computers*, 44–54.
- [2] *IEEE Standard VHDL Reference Manual*, New York, 1988.
- [3] Hosszú, G., Kovács, F., Varga, L. (1999). "Design procedure based on VHDL language transformations", *IEEE Int. Symposium on Circuit and Systems, ISCAS'99*, Orlando.
- [4] Jerraya, A.A., Ding, H., Kission, P. and Rahmouni, M. (1997) Behavioral Synthesis and Component Reuse With VHDL (Kluwer Academic Publishers, Boston).
- [5] Gajski, D., Vahid, F., Narayan, S. and Gong, J. (1994) Specification and Design of Embedded Systems (Prentice Hall, Englewood Cliffs, NJ).
- [6] Pirmez, L., Rahmouni, M., Kission, P., Pedroza, A., Mesquita, A. and Jerraya, A.A. (1996) "Analysis of different protocol description styles in VHDL for high-level synthesis", *Proceedings of European Design Automation Conference*.
- [7] Keresztes, P. and Ágotai, I. (1993) "The concept of superprocesses for high level synthesis and their VHDL Modeling", *Proceedings of European VHDL Conference, Hamburg*, 480–485.
- [8] Varga, L., Hosszú, G. and Kovács, F. (1999) "Resource sharing for low-power in high-level synthesis", *Proceedings of Electronic Devices and Systems Conference, Brno*.
- [9] Chandrakasan, A., Potkonjak, M., Mehra, R., Rabaey, J. and Brodersen, R.W. (1995) "Optimizing power using transformations", *IEEE Trans. On Computer Aided Design of Integrated Circuit and Systems* Jan..
- [10] Monteiro, J., Devadas, S., Ashar, P. and Mauskar, A. (1996) "Scheduling techniques to enable power management", *Proceedings of Design Automation Conference, Las Vegas*.

Authors' Biographies

László Varga received his MS degree in electrical engineering from the Technical University of Budapest, Hungary. Currently he is working toward on his PhD degree in electrical engineering at the Budapest University of Technology and Economics. He presented several conference papers about his work on high-level design methodologies. His research interests include high-level synthesis with VHDL, design automation and synthesis for low-power systems.

Gábor Hosszú received the ME degree from Technical University of Budapest in electrical engineering and the Academic degree of Technical Sciences (PhD) in 1992. After graduation he received a three-year grant of the Hungarian Academy of Sciences, then he continued his research in the Microelectronics Co. In 1990 he joined the Electron Devices Department of the Budapest University

of Technology and Economics, where he is currently Associate Professor. He published several papers on CAD methods and VHDL modeling. His further research fields are multicasting and media streaming technologies.

Ferenc Kovács received the ME degree from the Technical University of Budapest in 1959, and the Academic Degree of Doctor of Technical Sciences in

1999. He joined the Research Institute for Electronics in 1959 where he worked in the field of microelectronic design and testing. Since 2000 he has been Professor of the Electron Devices Department of the Budapest University of Technology and Economics. He published more than 100 technical papers and three books on semiconductor and IC applications. His further areas of research are real-time control and signal processing.

