Journal of Computer Science 1 (2): 270-275, 2005
ISSN 1549-3636

VHDL Specification Methodology from High-level Specification

M. Benmohammed and S. Merniz
LIRE Lab., Computer Science Departmetn, University of Cne, 25000 Constantine, ALGERIA.

Abstract: Design complexity has been increasing exponentially this last decade. In order to cope with such an increase and to keep up designers' productivity, higher level specifications were required. Moreover new synthesis systems, starting with a high level specification, have been developed in order to automate and speed up processor design. This study presents a VHDL specification methodology aimed to extend structured design methodologies to the behavioral level. The goal is to develop VHDL modeling strategies in order to master the design and analysis of large and complex systems. Structured design methodologies are combined with a high-level synthesis system, a VHDL based behavioral synthesis tool, in order to allow hierarchical design and component re-use.

Keys words: CAD, VLSI, VHDL, High Level Synthesis, Hierarchical Design

INTRODUCTION

Due to the increasing complexity of designs within the last decades, designers had resort to higher level specifications. Moreover in order to cope with time-to-market constraints, various tools (both for simulation and synthesis) have been developed, and thereby promoting high level specification for VLSI [1].
The efficiency of such tools is increased on application of structured design methodologies. Besides, structured design methodology allows to handle very complex design with hierarchical approach. Hierarchical design proceeds by partitioning a system into modules [2]. During the design of the uppermost system, the implementation details of these modules are hidden. Proper partitioning allows independence between the design of the different parts. The decomposition is generally guided by structuring rules aimed to hide local design decisions, such that only the interface of each module is visible.
Structured design methodology for VLSI consists of 3 main steps in the design-flow:

* Partitioning of a whole system into sub-systems,
* Synthesis of each resulting sub-system, and
* Abstraction of each synthesized sub-system to be used as component during the synthesis of systems higher in the hierarchy.

Structured design methodologies for VLSI have been developed at different abstraction levels: physical or circuit [3], logic and register transfer [4] levels and some work has been achieved for the behavioral domain [5]. Such methodologies allow to handle complex designs with a hierachical approach. Fig.1 illustrates the structured design methodology applied at the behavioral level. Hierarchical decomposition or partitioning splits the system specification into simpler sub-systems. These units are to be defined according to the corresponding degree of re-utilization, and those that may share the same operators may be regrouped.

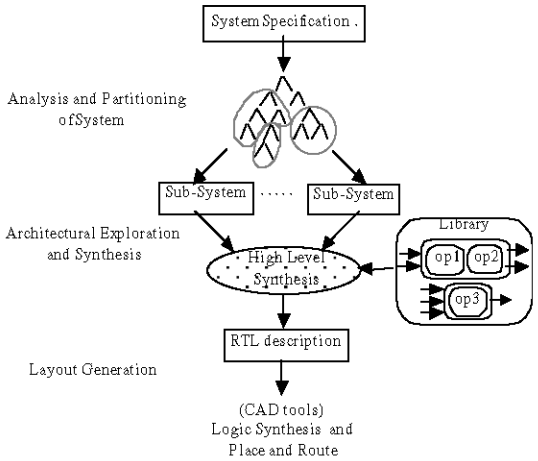

Fig. 1: Design-flow

When the hierarchical decomposition is done with respect to the regularity of the system design, the whole synthesis is simplified. Regularity implies that sub-systems or specific designs will be re-used more than once and therefore the total amount of designs needed will be reduced [2]. As a result, regularity allows an improvement in productivity, in general.

Models for Structured Design Methodology: The main models used for structured design methodology are those of the component and the system. A component model is a sub-system that will be re-used. A system is a full design made of an assembling of already designed components. These two concepts will be detailed in the case of a structured design methodology acting at the behavioral level, on a VHDL description.

Behavioral Components: A behavioral component is an entity able to execute a set of operations invoked in the behavioral description. It also acts as a black box linking the behavioral and register transfer levels. The operation(s) executed by the behavioral component may be as simple as predefined operations (+, -, *, ...) or as complex as input/output operations with handshaking or memory access with complex addressing functions. A component may correspond to a design produced by external tools and methods or to a sub-system resulting from an early design session. Complex operations can be invoked through procedure and function calls in the behavioral description. Allowing the use of procedures and functions within a HDL (Hardware Description Language) is a kind of extension of this HDL. This concept is similar to the concept of system function library in programming languages [6]. This way a language is composed of two parts:

* A fixed part which includes the predefined constructs, and,
* An exchangeable part which includes a set of procedures and functions that can be used within the language. These need not to be part of the language itself.

Modeling for Re-use: the Behavioral Components: In order to allow re-use, behavioral components have to be abstracted. The concept of behavioral component is a generalisation of functional unit concept, it allows the use of existing macro-blocks in the behavioral specification. A functional unit may execute standard operations or new customized operations introduced by the user. Each functional unit can be specified at three different abstraction levels: the conceptual view, the behavioral view and the implementation view. Figure 2 shows these three views for a memory cell that can achieve the 2 operations: mread and mwrite.

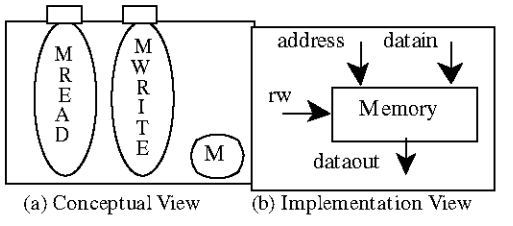

Fig. 2: The Three Abstraction Levels

From the conceptual point of view, the functional unit is an object that can execute one or several operations which may share some data (M). The implementation shows an external view of a possible realization of the functional unit; it thus includes the different connections of the functional unit: inputs, outputs and selection commands (selecting the procedure to execute). At the behavioral level, the functional unit is described through the operation that can be called from the behavioral description. These may correspond to standard operations or procedures and functions.

In order to use this kind of model for high level synthesis, a fourth model will be necessary. It will be called the high-level synthesis view. Its goal is to link the behavioral and implementation views. It includes the interface of the functional unit, its call-parameters (corresponding to the operation parameters), the operation set executed by the functional unit as well as the parameter passing protocol for each operation. High level synthesis algorithms impose that such protocols make use of static clock cycles: each operation needs to have a fixed predictable execution time. In order to overcome this constraint and to enable the use of complex functional units that may execute operations with data-dependent execution time, the methodology used consists in splitting the operation into a set of atomic operations with fixed execution time. The behavioral description will then be written according to the atomic operations introduced.

Modular and Flexible Architectural Model for Behavioral Design for Re-use: A system is viewed as an assembling of sub-systems coordinated through a top controller. This is a modular and flexible architecture model as shown in Fig. 3. It is composed of a top controller, a set of functional modules and a communication network. These last two constitute the datapath. Functional units can be of any degree of complexity and can themselves be the result of a synthesis process, as we described above in the case of a fixed-point unit synthesized by AMICAL and used as a functional unit to build a PID.

The network is built in order to allow the communication between functional modules, and with the external world.The top controller sequences the operations executed by the functional units and the communication network. Modular design can be achieved as the functional modules can be designed separately using different design methods. This model is flexible, it allows several configurations of functional modules and different communication schemes.

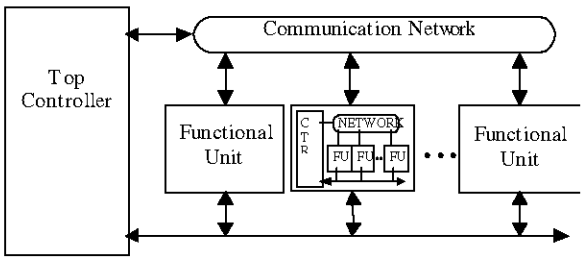

Fig. 3: Target Architecture

System Level Specification with VHDL: A behavioral description of the architecture model will give only a view of the top control, e.g. the coordination of the different sub-systems. In VHDL this may be described as a process that may make use of complex sub-systems through procedure and function calls. In other words, the functional units are used as black boxes. The only pieces of information required about each functional unit are the list of procedures executed by the functional unit and some informations about parameter passing protocols.

Memory with complex addressing function and specific embedded computation and control can be easily described with this scheme. The addressing functions may be realized by an independent functional unit or may be integrated within the memory unit. In the same way complex I/O units may be used. They are also accessed through function and procedure calls. These may execute complex protocol or data conversion.

The association between the operations of the behavioral description (standard operators such as + and -, and procedure and function calls) and the functional units is made during the synthesis process, it may be made through names. Operations may have different execution schemes on different functional units within the library. The number of functional units selected (allocated or instanciated) will depend on the parallelism allowed by the initial description. Of course the synthesis process tries to share as much as possible the use of the functional units.

AMICAL: High Level Synthesis for Hierarchical Design: AMICAL is a high level synthesis system allowing structured design methodology [5]. It starts with two kinds of information: a behavioral specification given in VHDL and an external functional unit library and allows architectural exploration and synthesis. This corresponds to the second step of the methodology introduced as above. The first step, system analysis and partitioning, is performed manually.

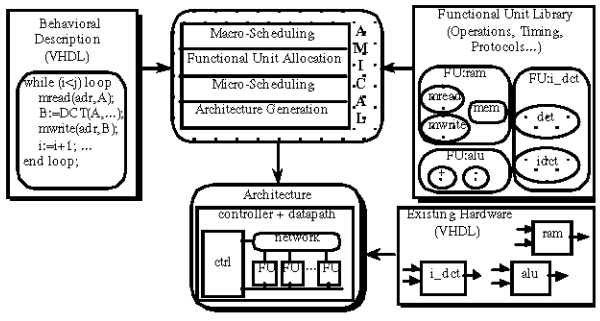

Fig. 4: AMICAL Design-flow

The AMICAL design-flow is illustrated by Fig. 4. The behavioral description is a VHDL process that may make use of complex sub-systems through procedure and function calls; in this case the behavioral description makes use of a complex function DCT. However for each procedure or function used, the library must include at least one functional unit able to execute the corresponding operation. In this case, the library includes a RAM able to execute the mread and mwrite procedures, a DSP-unit able to execute the dct and idct functions, and an ALU. During the different steps involved in the high-level synthesis, the functional units are used as black boxes, that may execute a list of procedures. However to complete the description at the register transfer level, the details of the functional units, about its implementation view are required.

The different steps involved in the synthesis process are: macro-scheduling, allocation, micro-scheduling and architecture generation. The macro-scheduler produces a finite state machine presented as a transition table from the initial behavioral description. Each transition corresponds to the execution of a control step under a given condition.

At this stage, each operation may take several clock cycles to execute. A transition is also called macro-cycle or macro-step. In fact a transition corresponds to a simple data-flow graph that has to be further synthesized using scheduling and allocation. The goal of these steps is to refine each macro-cycle into a set of basic control steps executing in one clock cycle each. These basic control steps are also called micro-cycles.

After scheduling, allocation starts with two kinds of information, namely the scheduled description (a set of data-flow graphs) and an external functional unit library. During the following steps, both allocation and binding are performed. The functional unit allocation step associates a functional unit with each operation in the state table. A second scheduling step (called micro-scheduling) is then performed according to the execution scheme for each operation. Each operation is decomposed into a set of transfers, which are scheduled into micro-cycles. Each micro-cycle contains a set of parallel transfers that take one basic clock cycle to execute.

The last synthesis step is a classic architecture generation. The clock cycle level description is mapped onto an architecture composed of a datapath and a controller. The communication network, within the datapath, may be composed of buses, multiplexers and registers. The number of buses and multiplexers is fixed according to the parallel transfers required by the architecture.

A Design Example: In order to illustrate hierarchical design and component re-use at the behavioral level we will use a design example: a PID (Proportional Integral Derivative -or Differential-).

The PID: A PID controller usually applies a control function to an analog input and generates an analog output. This kind of device is generally implemented as an analog device. The use of a digital solution allows to have a more flexible device.

The PID used in this study forms part of a speed control system detailed [7]. The speed control system includes an ALU which performs elementary and logic operations, and memories to store the state variables and coefficients. The PID algorithm is given by:

$$Irefk <= (Kp*Ek) + Ki*\int(Ek)dt + Kd*dEk/dt$$

where Kp, Ki, Kd are constants and Ek is the error change. However only the close approximation given as:

Irefk <= (Kp*Ek) + Ki*∑(Ek*deltaT) + Kd*Δ(Ek)/deltaT will be developped in order to be synthesized at the behavioral level by AMICAL, for digital implementation

System Level Analysis and Partitioning: The goal of the system level analysis and partitioning step is to structure the description in order to allow hierarchical description and component re-use. The result of such a step is a behavioral description and its corresponding functional unit library. The computation makes use of two complex fixed-point operations (/ and *). At this step the designer has to choose between using basic operators from the library (+, -, shift) and building specific units to perform these computations.

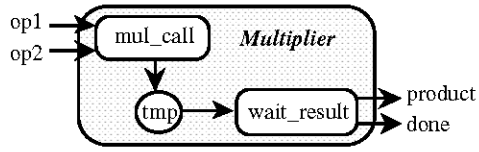

Fig. 5: Component Re-use

As the PID to be designed has no severe timing constraint, the multiplication and division operators will be implemented by sequential procedures using basic operators from the library (+, -, shift). In order to share the basic operators, we decided to gather all the fixed-point operators (*, /, +, -) within a fixed-point unit that will be synthesized by AMICAL and re-used in order to build the PID.

For the execution of the complex operations by the fixed-point unit, as functional unit, a 2-step protocol is applied. Each operation is controlled through 2 procedure calls :

* Starting the operation (or computation by the functional unit) through a first procedure call with the corresponding parameters, and
* Recovering the computation results through outputs of a second procedure call when their validity is indicated.

During its computation, the functional unit will be blind to any external command. These characteristics have to be taken into account while defining the components or functional units used for synthesis as well as when writing the behavioral description of the whole design. The 2-step protocol applied to the multiplication operation may be summarized by the Fig. 5.

Such an operation decomposition into atomic procedures allows component re-use and has been applied to both multiplication and division operations, for the design of the fixed-point unit. The design-flow of the PID is shown in Fig. 6.

Specifications: The behavioral description of the PID is given in Fig. 7. The global organization of this description is made of an entity/architecture pair. Unlike most existing synthesized architectures, the architecture accepted by AMICAL is made of a main process and a set of non-synthesizable statements.

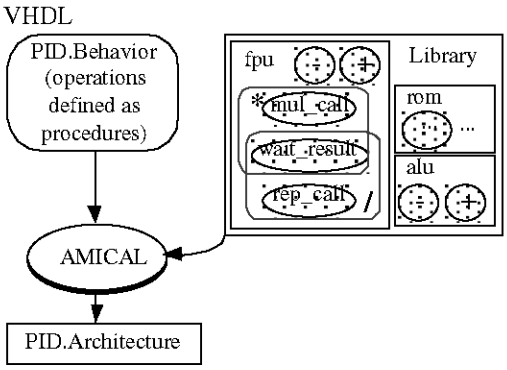

Fig. 6: Partitioning results for the PID

These elements may correspond to instances of functional units and other specific units. The main process accesses the functional units through procedure calls. As explained earlier the multiplication and division calls are split into a 2-step protocol. Lines 12 to 16 describe the sequence of statements used to execute a multiplication.

The multiplication call is made using a first procedure call ("mul_call") using as input parameters the corresponding input values (Kp and Ek). When the multiplication will be over, the result will be memorized within the functional unit itself. A second procedure ("wait_result") will be called to bring back the product obtained, Irefk. In order to point out when this value is ready, a validity signal called "done" is used. This results in writing a "mul_call" followed by a loop of "wait_result" until the result is ready.

The fixed-point unit itself is described by a separate VHDL entity that may be designed using AMICAL itself or some other specific tools. Figure 8a shows the behavioral description of the fixed-point unit. The corresponding synthesis view, described in Fig. 8b, after abstraction for re-use, gives the protocol exchange format between the top control and the functional unit. Each operation is decomposed into a set of scheduled cycles made of transfers to and from the functional unit.

The Design Process: The architectural synthesis of both fixed-point unit and PID are realized by AMICAL. The result of the synthesis of the fixed-point unit will be used twice:

* It will be used to create the corresponding behavioral component that will be used for the synthesis of the PID (abstraction of functional unit).
* It will be used during the logic synthesis of the PID as the required structural description of the functional unit: fixed-point unit.

```
use work.types_and_functions.all;
entity pid is
port (      clock, Fsignin                          : in bit;
            HostInterrupt, PositionChange           : in bit;
                  Irefkout                  : out str32b);
end pid;
architecture behavioral of pid is
      component fpu
      port (      clock                     : in bit;
                  input1, input2            : in str32b;
                  sel           : in bit;
                  com                       : in int3bits;
                  output                    : out str32b;
                  outdone                   : out bit);
      end component;
      signal sig_in1, sig_in2, sig_out      : str32b;
      signal sig_sel, sig_done              : bit;
      signal sig_com                        : int3bits;
begin
      Inst_FU : fpu
      port map (clock, reset, sig_in1, sig_in2,
                  sig_com, sig_out, sig_done);
      main : process
                  variable Ik, Ek, Kp, Ki, Fref, N, Fk : str32b;
                  variable Ek_1, Dek, Irefk, Temp : str32b;
                  variable done : bit;
                  variable val_rom : ROM;
                  procedure mul_call (a, b : in str32b) is
                  begin
                        sig_in1 <= a; sig_in2 <= b;
                        sig_sel <= '1'; sig_com <= 2;
                        wait until rising_edge(clock);
                  end mul_call;
                  procedure rep_call (a : in str32b) is
                  begin
                     sig_in1 <= a; sig_sel <= '1'; sig_com <= 1;
                     wait until rising_edge(clock);
                  end rep_call;
                  procedure wait_result (x: out str32b; y: out bit) is
                  begin
                        x := sig_out; y := sig_done;
                        wait until rising_edge(clock);
                  end wait_result;
                  ...
      begin
[1]   ...
[2]   wait until (HostInterrupt = '0');
[3]   while (HostInterrupt = '0') loop
[4]         ...
[5]         rep_call(N);
[6]         wait_result(Fk, done);
[7]         while (done /= '1') loop
[8]               wait_result(Fk, done);
[9]         end loop;
[10]        if (Fsignin = '0') then Ek := Fref-Fk;
[11]        else Ek := Fref+Fk; end if;
[12]        mul_call(Kp, Ek);
[13]        wait_result(Irefk, done);
[14]        while (done /= '1') loop
[15]              wait_result(Irefk, done);
[16]        end loop;
[17]        ...
[18]        Irefkout <= Irefk + Temp;
[19]  end loop;
[20]  ...
      end process main;
end behavioral;
```

Fig. 7: PID Algorithm (VHDL Description)

```
use work.types_and_functions.all;
entity fixedpointunit is
port (      clock                           : in bit;
            input1, input2          : in str32b; -- input values
            sel                     : in bit; -- enable signal
            com                     : in int3bits; -- operation asked
            output                  : out str32b; -- output value
            outdone                         : out bit); -- validation signal
end fixedpointunit;
architecture behavior of fixedpointunit is
begin
      process
            variable tmp      : integer;              -- result buffer
            variable val1, val2       : integer; -- input value buffers
            procedure mul_call(A,B: in integer) is begin
                  -- shift and add algorithm; tmp:= A * B;
            end mul;
            procedure rep_call(A: in integer)is begin
                  -- restoring division algorithm; tmp:= 1/A;
            end rep;
      begin
                  wait until sel='1';
                  case com is
                  when 1 => -- rep_call
                     outdone <= '0';      val1:=input1;      rep(val1);
                  when 2 => -- mul_call
                     outdone <= '0';      val1:= input1; val2:= input2;
                     mul(input1,input2);
                  when 3 => -- "+"
                              Z = A + B;
                  when 4 => -- "-"
                              Z <= A - B;
                  when 5 => -- wait_result
                     Z <= tmp;            outdone <= '1';
                  end case;
      end process;
end behavior;
```

(a) Behavioral description

```
(FU fpu
      (AREA 30000)
      (PARAMETER (DataIn A B) (DataOut Z done))
      (CONNECTOR
            (DataIn input1 (BIT 0 32) input2 (BIT 0 32))
            (DataOut output (BIT 0 32) outdone (BIT 0 1))
            (ControlIn sel (BIT 0 1))
            (ControlIn com (BIT 0 3)))
      (OpType + (commutative A B)
            (Cycle 1 (Transfer A input1) (Transfer B input2)
            (Transfer output Z)           (active sel 1) (active com 3)))
      (OpType -
            (Cycle 1 (Transfer A input1) (Transfer B input2)
            (Transfer output Z) (active sel 1) (active com 4)))
      (OpType mul_call (commutative A B)
            (Cycle 1 (Transfer A input1)
            (Transfer B input2) (active sel 1) (active com 2)))
      (OpType rep_call
            (Cycle 1 (Transfer A input1)
            (Transfer B input2) (active sel 1) (active com 1)))
      (OpType wait_result
            (Cycle 1 (Transfer output Z)
            (Transfer outdone done) (active sel 1) (active com 5))))
```

(b) Synthesis view

Fig. 8: Fixed-point Unit

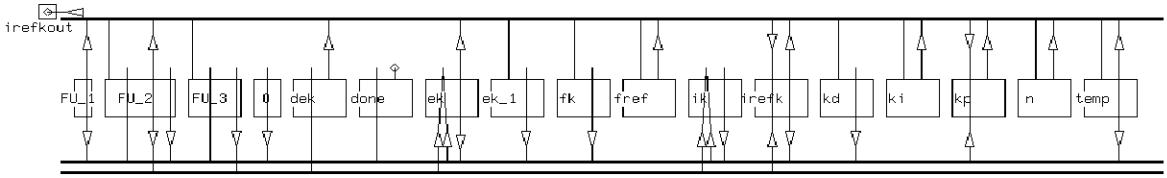

Fig. 9: Layout of PID

The synthesis of the PID produced an architecture where the controller is a 22-state and 33-transition finite state machine. The datapath obtained after some interactive architectural transformations is made up of 3 functional units and 3 buses. Within the PID datapath, one of the components is an instance of the fixed-point unit compiled previously.

The resulting RTL description has been fed to the commercial logic synthesis and place and route tools. The synthesis results obtained for the PID is composed of around 50000 transistors. The full design stands on 10.5 mm square when mapped onto a 0.8 CMOS technology. The Fig. 9 gives the layout of the final chip.

CONCLUSION

This study dealt with structuring design in order to allow hierarchical design and synthesis using VHDL at the behavioral level. The use of behavioral design to build more complex design corresponds to the re-use of existing components for the design and synthesis at the behavioral level.

The scheme detailed above is powerful as it allows for hierarchical design based on behavioral VHDL descriptions. This structured method enables the use of complex sub-systems as functional units in the library during architectural synthesis.

ACKNOWLEDGEMENTS

We gratefully acknowledge the help of the following researchers at TIMA/INPG (Grenoble, FRANCE): P.Vijay Radhavan, Maher Rahmouni, Hong Ding, Mohamed Romdhani, Clifford Liem, Elisabeth Berrebi and Volker Zens.

REFERENCES

1. Courtois, B., 1994. CAD and testing of ICs and systems: Where are we going?. J. Microelectronics Systems Integration, 2: 3.
2. Weste, N.H.E. and K. Eshraghian, 1993.Principles of CMOS VLSI Design. Addison-Wesley (Publ).
3. Trimberger, S., J.A. Rowson, C.R. Lang and J.P. Gray, 1981. A structured design methodology and associated software tools. IEEE Transactions on Circuits and Systems, 28: 7.
4. Girczyc, E. and S. Carlson, 1993. Increasing design quality and engineering productivity through design reuse. 30th ACM/IEEE Design Automation Conference.
5. Kission, P., H. Ding and A.A. Jerraya, 1994. Structured design methodology for high-level design. 31st ACM/IEEE Design Automation Conference.
6. Backus, J., 1978. Can programming be liberated from the Von Newman style? A functional style and its algebra of programs. CACM, 21: 8.
7. Kinniment, D.J., P.P. Acarnley and A.G. Jack, 1991. An integrated circuit controller for brushless dc drives. European Power Electronics Conference Proceedings, 4: 111-116, EPE Firenze.