Analysis of Different Protocol Description Styles in VHDL for High-Level Synthesis

L. Pirmez[1,2,3] M. Rahmouni[1] P. Kission[1]
A. Pedroza[3,4] A. Mesquita[3] A. A. Jerraya[1]

[1] TIMA Laboratory Grenoble, France
[2] NCE/UFRJ Electronic Computer Center RJ, Brazil
[3] Coppe/UFRJ - Program of Electric Engineering RJ, Brazil
[4] EE/UFRJ - Department of Electronics RJ, Brazil

Abstract

When synthesizing control-flow dominated descriptions based on VHDL, different styles of semantically equivalent descriptions may differ significantly in quality. This paper discusses the effect of the input description on High-Level Synthesis when using VHDL. In order to show this effect, a high speed protocol based on the ISO reference protocol Abracadabra *is used. Five VHDL descriptions styles of the same protocol have been synthesized using AMICAL, a VHDL based behavioral synthesis tool. Discussions of the different results leads to a VHDL based methodology for protocol modelling in order to produce efficient designs.*

1. Introduction

The emergence of new technologies in VLSI, the high efficiency of nodes (commuters), the reduction of memory and processor costs and the development of optical fibers allowed the development of networks of increasing speed and reliability. The bottleneck in the hardware development of complex distributed system is the design of the communication protocols.

On the other side, High Level Synthesis (HLS) tools (also called behavioral or architectural synthesis) are maturing and may provide an issue to the design bottleneck. These tools allow to produce complex designs starting from a behavioral description.

Protocol design require specific HLS tools, generally called control flow dominated behavioral compilers[13]. These tools are able to handle large control structure including sophisticated handshaking and non-structured sequences. However most of these compilers present an inescapable drawback, the compilation results depends on the quality of the input description. This comes from the fact that the specification may include explicit synchronization points (e.g. a wait in a VHDL description) which restricts the scope of the transformation and optimization usually performed in behavioral descriptions[1].

A communication protocol is a set of rules and procedures that allow the interaction between peers entities and, consequently, they are crucial to the functionallity of the computers networks and of the distributed systems. A communication protocol is structured on such concepts as hierarchy, concurrence, distributiveness and communication.

A protocol can be described as a set of communicating processes where the behavior of each process can be represented by one or more communicating FSMs. Then, a communication protocol can be characterized by multiple, hierarchical, concurrent and communicating FSM's.

The FSM model is traditionally used to describe a communication protocol. Each FSM can be described using standard programming language statements, such as loops, if and case statements, and assignment statements. Unstructured statements such as exceptions may also be used. Moreover, the normal course of the actions can be disrupted at any point in time by events signaling exceptions (resets, time-out, error situations, etc.). Thus, an exception procedure is treated immediately after the corresponding wait statement. Reset conditions may be defined by adding an outer loop to the process and including an *exit outer loop* into the corresponding exception clause.

This paper discusses the effect of the input description on HLS when using VHDL[3]. We will use AM-

EURO-DAC '96 with EURO-VHDL '96
0-89791-848-7/96 $4.00

ICAL, a VHDL behavioral compiler for control flow dominated machines[8].

In order to show this effect, a high speed protocol based on the ISO reference protocol Abracadabra called Abracadabra_HS was devised. This protocol includes the parameters of most of the existing high performance protocols such as the XTP[5] and the DATAKIT[6].

Section 2 presents the preliminary concepts concerning high level synthesis and AMICAL, a VHDL based behavioral synthesis tool. This section also presents the scheduling algorithm used in AMICAL. The knowledge of this algorithm is crucial to the understanding of the next sections. The different styles that can be used to describe a protocol in VHDL are discussed in section 3 . In section 4 the proposed different description styles are applied to the high performance protocol Abracadabra_HS and the results are compared. The last section contains the conclusions and the results obtained so far, are summarized.

2. High Level Synthesis and Scheduling

HLS is a procedure that maps a behavioral description into a structural RTL description. The main tasks involved in HLS are scheduling and allocation. In this section the scheduling step will be focused.

Designers can write a behavioral description in many different styles. As will seen in the next section, from the HLS point of view, each different style of semantically equivalent descriptions, may differ significantly in quality. This quality will depend on the type of the constructs used in the description and on their order. This is known as the problem of syntactic variance[7].

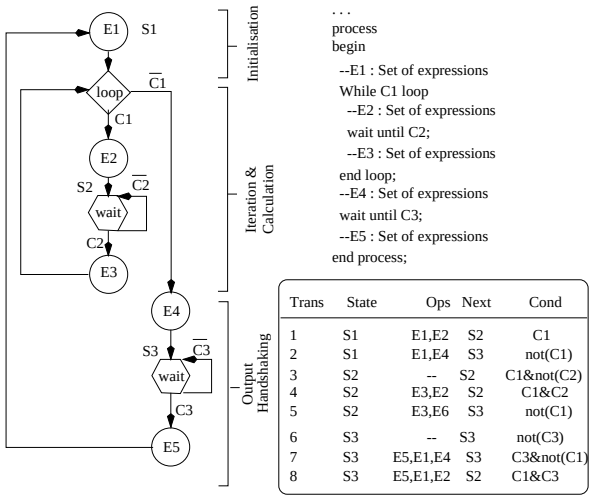
When dealing with real time applications where the control sequence is based on external conditions and the corresponding algorithmic description contains few arithmetic operations, i.e., they are specified basically by control statements, the control dependency may be conveniently represented by Control Flow Graphs(CFG).

Starting from a CFG, scheduling assigns operations to control steps. A control step corresponds to a basic machine cycle in a controlling FSM.

2.1. Dynamic Loop Scheduling

Dynamic Loop Scheduling (DLS)[12] is a native VHDL scheduling algorithm optimised for the treatment of control-flow dominated descriptions. It is based on the principles of path-based scheduling[2] but significantly reduces the number of the generated paths and hence, the computation cost. DLS has been implemented and integrated with the AMCAL high-level synthesis tool.

The main representation used by DLS is a control-flow graph (CFG)[11]. This kind of representation is well-suited to control-flow dominated circuits. It provides a convenient means to represent inherent properties of these circuits that are described by nested loops, unstructured control such as loop exits and synchronization constructs such as wait statements. Perhaps the most significant characteristic of this CFG is the introduction of the wait node. A wait node always means that a change of state will take place. The wait statements plays a fundamental role in describing control-flow. Therefore, we allow the use of asynchronous signals in these statements. As will be seen in the next section, the use of multiple wait statements in the input description will significantly reduce the complexity of the resulting FSM by reducing the number of paths generated. To show how DLS works, consider



Trans	State	Ops	Next	Cond
1	S1	E1,E2	S2	C1
2	S1	E1,E4	S3	not(C1)
3	S2	--	S2	C1&not(C2)
4	S2	E3,E2	S2	C1&C2
5	S2	E3,E6	S3	not(C1)
6	S3	--	S3	not(C3)
7	S3	E5,E1,E4	S3	C3&not(C1)
8	S3	E5,E1,E2	S2	C1&C3

Figure 1. Control-flow graph, VHDL process outline and transition table representation for a typical handshaking process.

any circuit that needs to perform handshaking operations with another circuit. Such functions are needed in almost every controller-like circuit that synchronizes with external devices. A classic example is outlined in figure 1. In this model one can distinguish three steps; an initialisation phase, a calculation phase which consists of a loop that waits for input data on each iteration and performs some calculations on these data, and a final handshaking phase to output the result and perhaps receive an acknowledgement from processes requesting the result. The wait statements are used to

synchronize with one or more external processes. The first step in the DLS, consists in computing the set of possible execution paths from the CFG. A path consists of a sequence of operations which may execute in parallel. The paths are generated starting from the first node in the CFG. The generation of the current path is stopped and a new path begins when a *Wait statement* is encountered or the new node encountered is data dependent on one of the nodes already in the path. The next step is to generate the corresponding FSM. Each generated path corresponds to a transition in the FSM. The *And* of all condition nodes will constitute the condition for the corresponding transition. All paths starting with the same node are bundled together in the same state. For this example, a total of 3 states and 8 transitions are extracted, as shown in figure 1.

3. Styles the description of protocols in VHDL

A protocol can be described in VHDL in many different styles. To illustrate this fact, consider a protocol model that has a state machine performing handshaking operations with another state machine. This model has two phases: the initialization phase and the protocol processing phase. This last one consists of a loop that waits for an input signal on each interaction, processes the input data and outputs the corresponding result through a signal. Thus, the processing phase has 3 steps: input, treatment and output. We assume that there are no data dependencies in this model. The initialization phase corresponds to node E1 and the treatment steps to nodes E2,E3, respectively. The input step is represented by the wait statement and the output step by the assignment statement $x <= value$. Each node in figures 2, 3 and 4 may contain one or more statements. This example will be used to illustrate the several description styles and the consequences on the HLS results. The analysis will be restricted to the scheduling results since this gives the complexity of the controller. In fact, all these styles will result into nearly equivalent data_paths. Moreover, in a protocol design the data-path is generally smaller than the controller.

The First Description style associates *wait until ...* statements to each branch of a *case state* statement. Figure 2 shows the VHDL description of the first style and the FSM resulting from the schedule of this description using the DLS algorithm. During the path generation the DLS algorithm stops the generation of the current path, for example the E1_Loop_Case path, when a Wait statement is found and a new path, for

```
process
begin
1   -E1
2   FSM loop
3     case state is
       when state_x =>
4       next <= '1';
5       wait until inputx = '1';
6       next <= '0';
7       -E2
8       outx <= '1';
9       wait until rising_edge(clk);
10       outx <= '0';
       when state_y =>
11       next <= '1';
12       wait until inputy = '1';
13       next <= '0';
14       -E3
15       outy <= '1';
16       wait until rising_edge(clk);
17       outy <= '0';
      end case;
    end loop;
end process;
```

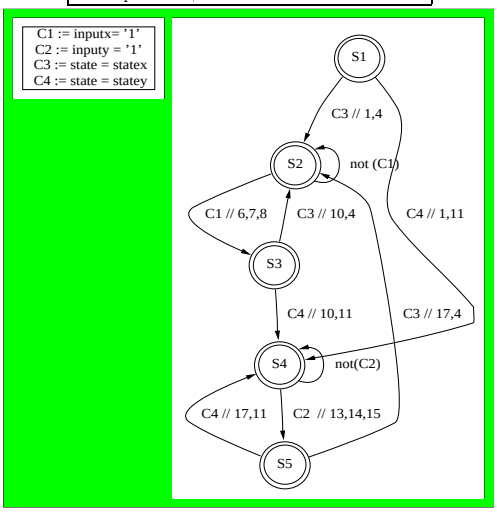


Figure 2. VHDL process model and FSM of the First Description

example wait_E2_out_Loop_Case path, begins. It is important to note that there are, at least, two possibilities to begin a new path when there is a *wait until input_condition* statement in each *case state* statement option.

The second description style for implementing a state machine is a variation of the first one. In this case, a *wait until input_condition* statement is placed outside the case statement. Figure 3 shows the VHDL description of the second style and the FSM resulting from the schedule of this description. In this style, it is important to note that there is only one possibility to begin a new path when there is only a *wait until input_condition* statement before the *case state* state-

```
process
begin
1   -E1
2   FSM loop
3     next <= '1';
4     wait until inputx = '1' or inputy = '1';
5     next <= '0';
6     case state is
      when state_x =>
7       if inputx = '1' then
8         -E2
9         outx <= '1';
10          wait until rising_edge(clk);
11          outx <= '0';
        end if;
      when state_y =>
12      if inputy = '1' then
13        -E3
14        outy <= '1';
15        wait until rising_edge(clk);
16        outy <= '0';
        end if;
    end case;
  end loop;
end process;
```

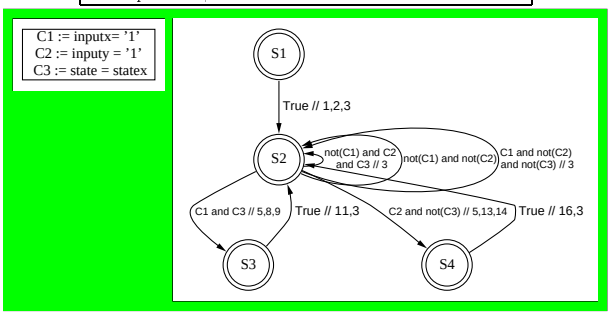


Figure 3. VHDL process model and FSM of the Second Description

```
process
begin
1   -E1
2   FSM loop
3     next <= '1';
4     wait until inputx = '1' or inputy = '1';
5     next <= '0';
6     case state is
      when state_x =>
7       if inputx = '1' then
8         -E2
9         outx <= '1';
        end if;
      when state_y =>
10      if inputy = '1' then
11        -E3
12        outy <= '1';
        end if;
      end case;
13    wait until rising_edge(clk);
14    outx <= '0';
15    outy <= '0';
  end loop;
end process;
```

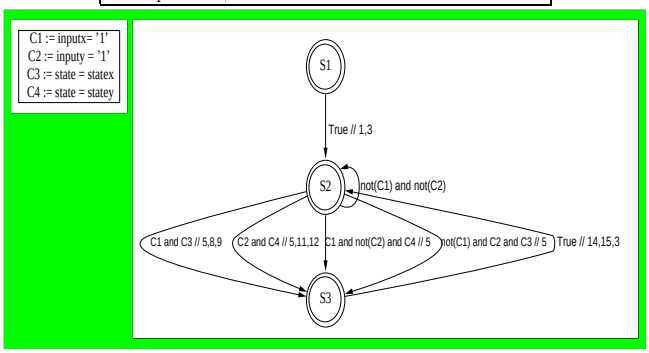


Figure 4. VHDL process model and FSM of the Fifth Description

ment.

The Third Description style employs an *If ... then ... elsif ... end if* construct instead of a *case state* statement into a *process* statement. This construct uses the state as the outer variable. A *wait until input_condition* statement is placed outside the nested if clauses.

The fourth description style is a variation of the third one. It also employs an *if ... then ... elsif ... end if* structure but with the input as the outer signal. The Fifth alternative to describe a state machine is a variation of the Second Description style: the different output control signals are reseted immediately after the *case state* statement. Figure 4 shows the VHDL description of the fifth style and the FSM resulting from the schedule of this description.

If these different descriptions are synthesized by AMICAL, different results will be produced by the Dynamic Loop Scheduling (DLS) algorithm, as will be seen in the next section.

4. A complex protocol specification

The five alternative styles to describe a protocol in VHDL discussed previously, will be applied to an example, the Abracadabra_HS protocol.

4.1. The High Performance Protocol : Abracadabra_HS

The Abracadabra_HS protocol is a proposed high speed version of the ISO reference protocol Abracadabra[4]. Considerable attention was payed on its design in order to shorten the length of the instructions, transmission path and reception of messages, when no errors occurs. This agrees with the design philosophy of minimizing the processing needs, even if bandwidth is sacrified, optimizing the protocol processing in the case of normal (error-free) communication, and including more effective control-flow algorithms.

4.2. Internal architecture of Abracadabra_HS

The Abracadabra_HS protocol is structured into five modules, as shown in Figure 5, namely the Signaling, Data, Generation-ctrl, Critic_Region and Rate-Control module. Each module is represented by a finite state

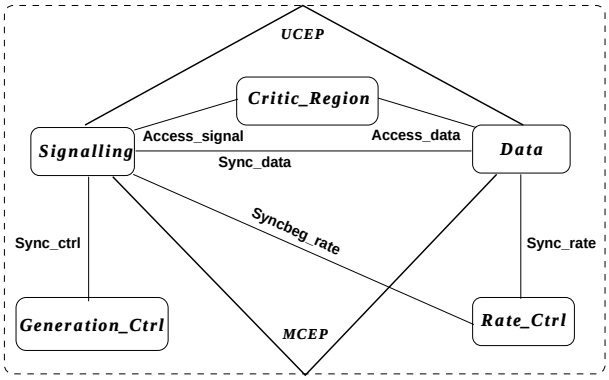


Figure 5. Architecture of Abracadabra_HS provider

machine. The Signaling module (Data module) is responsible for the message Signaling (Data) management. The Generation-Ctrl module (Rate_Ctrl module), when active, is responsible for the periodic generation of a signal to the Signaling module (Data module). The Critic-Region module is responsible for assuring the mutual exclusion in the access of the global variable by the Signaling and Data modules. These modules are connected by internal channels as Sync_data, Sync_ctrl, Syncbeg_rate, Sync_rate, Access_signal, and Access_data. A channel is implemented by two FIFOs queues, one in each channel side. Moreover, there are two external channels, the UCEP and the MCEP, and they connect the Abracadabra_HS entity to the UPPER level and to the LOWER level, respectively. A full specification of the protocol Abracadabra_HS can be found in[9, 10].

4.3. Results

The example discussed in this section is the Signaling state machine, the most complex module of the Abracadabra_HS protocol. All the different styles of description in VHDL were applied to this machine. The scheduling results obtained by the DLS algorithm, for each description, are displayed in table 1. In the first two columns of this table the number of paths and the number of states generated by each description are shown. The next five columns contain the number of operations, the number of loops, the number of waits, the number of ifs and the number of cases, respectively. In the last two columns the number of VHDL lines of code in the input description and an estimation of the area of the controller, in number of transistors, are given.

Description	Path	States	Operations	Loops
First	614	88	400	12
Second	138	70	406	12
Third	136	69	413	12
Fourth	137	69	451	12
Fifth	104	36	325	12

Description	Wait	Ifs	Case	Lines	Area
First	54	34	1	653	206.183
Second	45	42	1	622	46.473
Third	44	51	0	625	45.904
Fourth	44	52	1	616	46.237
Fifth	11	42	1	557	35.641

Table 1. DLS Results

The five descriptions have very complex control structures including wait statements with several conditionals, procedure calls, loops, nested ifs and nested loops. All of them took less than one second to execute on a SPARC II workstation. It can be observed from table 1 that globally the Fifth Description presented the best results. It significantly reduces the number of *wait until input_condition* and the number of *wait until rising_edge (clk)*. As a consequence, the number of paths generated is reduced and, hence, the computation cost. This is due to the structure of DLS algorithm. This algorithm, as dicussed earlier, generates a new path each time a wait statement is found. It can observed that the Second and Third Descriptions had also good results, because they significantly reduce the number of *waits until input_condition* reducing consequently the number of paths generated. The Second and Third Description produce similar results. This is in agreement with the fact that the if ... then ...elsif ... endif and case statement are similar constructs.

Another interesting point can be observed in the results of the Third and the Fourth Description styles. In the Third Description style, the outer *If ... then ... elsif ... end if* construct test the state variable and in the Fourth description style, the outer *If ... then ... elsif ... end if* construct tests the input conditions. The results show that the Third Description is better than the Fourth description because in this example the processing is more dependent on the state than the input. In situations where the processing is more dependent on the input than on the state one should expect a

better result from the Fourth Description style.

In the case of protocol design, in order to reduce the controller complexity, the following rules should be adopted;

- The FSM protocol and its signals are identified. The FSM model corresponds to a VHDL process. The process is composed of two loops: an external loop that models the restart of the process and an internal that models the protocol FSM.
- The *wait until ...* statement is combined with the INNER_LOOP where the control signals are checked.
- In order to avoid duplications in the code when specifying exceptions, the exceptions signals (reset, error situations) should be created and added to the sensitivity list of the wait statement. Thus, the exceptions signals (reset, error situations) are immediately checked after this wait statement.
- A state machine can be built using a case statement or *if ... then ... elsif ... end if* constructs. In both solutions, the choice between initially to test the state or the input depend on if the processing is more dependent on the state than the input or vice-versa.
- After a *case...* or after an outer *if ... then ... elsif ... end if* statement, the output control signals should be set to zero.

5. Final Considerations

Five different styles that can be used to describe a protocol in VHDL and the constraints that determine the most appropriate to synthesis tools have been discussed. This analysis was done with the aid of an example, the Abracadabra_HS protocol. The main contributions of this study are:

- the use of multiple input conditions in a single wait statement in the input description allow us to significantly reduce the complexity of the transition table by reducing the number of paths generated by the scheduling algorithm of AMICAL
- the choice of the state as the outer variable, instead of the input, when the processing is state dominated, allows us to reduce the number of states generated by the scheduling
- the generation of exception signals in such a way that the exceptions codes are treated immediately after a wait statement, allow to avoid the duplication of exception codes.

References

[1] J. Bhasker and H. Lee. An optimizer for hardware synthesis. *IEEE Design and Test of Computers*, pages 20–36, 1990.

[2] R. Camposano. Path-based scheduling for synthesis. *IEEE trans. on CAD*, 10(1):85–93, Jan. 1991.

[3] R. Camposano, L. Saunders, and R. Tabet. VHDL as input for high-level synthesis. *IEEE Design and Test of Computers*, pages 43–49, Mar. 1991.

[4] W. Doeringer. A survey of light-weight transport protocols for high-speed networks. *IECO*, 388(11), November 1990.

[5] W. S. et al. *XTP: The Xpress Transfer Protocol.* Addison-Wesley, 1985.

[6] A. Fraser and W. Marshall. Data transport in a byte stream network. *IESAC*, 7(7), September 1989.

[7] D. D. Gajski and L. Ramachandran. Introduction to high-level synthesis. *IEEE Design and Test of Computers*, pages 44–54, Winter 1994.

[8] A. Jerraya, E. Berrebi, H. Ding, P. Kission, M. Rahmouni, and P. V. Raghavan. A pragmatic approach to behavioural synthesis. *Electronic Engineering*, May 1995.

[9] L. Pirmez. *A Methodology to the Implementation of Distributed System in Hardware from a Formal Description.* PhD thesis, COPPE/UFRJ Brazil, 1996.

[10] L. Pirmez, A. Pedroza, and C. Mesquita. A methodology to the implementation of distributed system in hardware from a formal description. In *International Symposium on Protocol Specification Testing and Verification*, Varsow, Poland, 1995.

[11] M. Rahmouni and A. Jerraya. Formulation and evaluation of scheduling techniques for control flow graphs. In *Proceedings of the European Design Automation Conference*, Brighton, UK, Sept. 1995.

[12] M. Rahmouni, K. O'Brien, and A. Jerraya. A loop-based scheduling algorithm for hardware description languages. *Parallel Processing Letters*, 4(3):351–364, 1994.

[13] R. Walker and R. Camposano. *A Survey of High-Level Synthesis Systems.* Kluwer Academic Publishers, 1991.