Timing Constraint Specification and Synthesis in Behavioral VHDL

Petru Eles*, Krzysztof Kuchcinski†, Zebo Peng†, and Alexa Doboli*

* Computer Science and Engineering Department
Technical University of Timisoara
Romania

† Dept. of Computer and Information Science
Linköping University
Sweden

Abstract

This paper describes two methods to specify timing constraints in behavioral VHDL for high-level synthesis purposes. The first method specifies timing constraints on sequences of statements by using predefined procedures. The second method provides support for specification of timing constraints across process borders based on concurrent assert statements on signal events. The paper discusses also an approach to synthesize hardware with timing constraints and concentrates in particular on how to ensure consistency between the behavior of the simulation model and that of the synthesized hardware.

1. Introduction

A high-level synthesis (HLS) system performs two main tasks: resource allocation and operation scheduling [6]. The time schedule of operations in the final hardware implementation generated by a HLS algorithm is determined by the optimization decisions made during synthesis. Therefore, a high level algorithmic description accepted for HLS very often contains no timing information. Some design requirements can, however, impose certain timing restrictions on the functionality of the specified system [10].

Requirements on the timing aspects of a design can be incorporated as *timing constraints* (TCs) in the behavioral specification submitted to the synthesis system. Verification of consistency and operation scheduling under timing constraints are discussed in [2] and [9]. In this paper we propose a notation for TC specifications in behavioral VHDL and discuss some aspects concerning synthesis with such constraints. We consider the following main requirements for a notation capturing TCs for high-level synthesis with VHDL:

- specification of minimal, maximal, exact, and range constraints between arbitrary statements;
- specification of nested TCs;
- simulation of the VHDL design before synthesis, with some estimated values for the TCs;
- high-level synthesis with TCs, preserving correspondence between the behavior of the synthesized hardware and that of the simulation model;
- back-annotation of the synthesized times for post-synthesis behavioral simulation;
- simplicity, readability, and clarity of the notation, and integration into the overall language concept.

The proposed notation is implemented in the VHDL front-end of the CAMAD high-level synthesis system designed at Linköping University [12, 4]. In Fig. 1 we show the overall structure of the system, including high-level synthesis from a behavioral VHDL specification with TCs to a register-transfer level hardware structure and back-annotation of the input specification for post-synthesis simulation. One important aspect of CAMAD is its ability to synthesize VHDL specifications consisting of several interacting processes [5]. Accepting for synthesis TCs in the context of VHDL processes interacting through signals generates difficult semantic problems. Some solutions will be presented for preserving correspondence between the behavior of the synthesized hardware and that of the simulation model, while accepting both TCs and interacting VHDL processes for synthesis.

This paper is divided into six sections. Section 2 discusses possible solutions for TC specification and evaluates their advantages and shortcomings. Section 3 presents our approach to TC specification with predefined procedures. Section 4 explores the problems concerning VHDL simula-

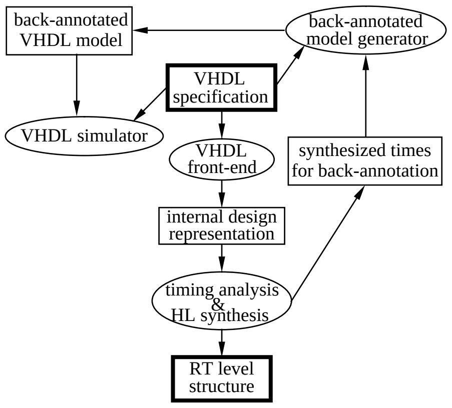

back-annotated VHDL model
back-annotated model generator
VHDL specification
VHDL simulator
VHDL front-end
synthesized times for back-annotation
internal design representation
timing analysis & HL synthesis
RT level structure

Fig. 1. High-level synthesis with timing constraints.

This work has been partially sponsored by the Swedish National Board for Industrial and Technical Development (NUTEK).

tion semantics in the context of synthesis with TCs and presents our solutions. In section 5 we introduce a mechanism for TC specifications across process borders. Finally, section 6 presents our conclusions.

2. Specification of TCs

Some hardware description languages, such as HardwareC [9] and DSL [2], include as part of their definitions a notation for TC specification. There is, however, no provision in VHDL [8] for the specification of a minimum, maximum or range of acceptable TC values for synthesis. According to VHDL standard semantics both *after* clauses in signal assignment statements and *wait* statements with time clauses are used to express strict simulation timing. Thus, if VHDL is used as an input language for high-level synthesis, some conventional notation has to be adopted for the TC specification.

Some HLS systems accepting VHDL as an input language, such as CALLAS [1], rely on *exact* synthesis of a user specified number of clock cycles between certain operations. In the VHDL subset accepted by CALLAS, timing specification is based on a set of predefined procedures. The CALLAS approach in fact requires scheduling decisions to be taken by the designer and thus limits the freedom for synthesis optimization. In [7] a similar approach to timing specification is proposed. In addition, the so called “interface procedures”, for the specification of the interface part of the circuit, are introduced. They are synthesized by specialized tools, and signal assignments and *wait for* statements are accepted only inside such a procedure. The use of predefined procedures for TC specification is also advocated in [11].

Synopsis VHDL tools use both commented lines and attributes to specify TCs [13]. The main drawback with these two methods is that no simulation of the behavioral model is possible, either before or after synthesis. The TCs are ignored by VHDL simulators and are recognized only by the synthesis tools.

3. Predefined Procedures for TC Specification

In order to express TCs in a behavioral VHDL description for high-level synthesis, a conventional notation is needed to specify restrictions on the execution time of a sequence of statements after synthesis. According to our requirements we consider the following restrictions on the execution of a sequence of operations:

- *Minimal delay*: it has to take at least a certain time;
- *Maximal delay*: it has to take at most a certain time;
- *Range delay*: it has to take a time between two limits;
- *Exact delay*: it has to take exactly a given time.

For each of the above TC specifications a predefined procedure is implemented. To specify a certain restriction the user calls the corresponding predefined timing procedure

```
PACKAGE constraints IS
  SUBTYPE time_1 IS TIME RANGE 150 ns TO 150 ns;
  SUBTYPE time_2 IS TIME RANGE 0 ns TO 200 ns;
  SUBTYPE time_3 IS TIME RANGE 100 ns TO TIME'HIGH;
  SUBTYPE time_4 IS TIME RANGE 100 ns TO 1200 ns;
  CONSTANT constr_1:time_1;
  CONSTANT constr_2:time_2;
  CONSTANT constr_3:time_3;
  CONSTANT constr_4,constr_5:time_4;
END constraints;
PACKAGE BODY constraints IS
  -- the values are estimated for pre-synthesis simulation;
  -- they will be back-annotated after synthesis.
  CONSTANT constr_1:time_1:=150 ns;
  CONSTANT constr_2:time_2:=100 ns;
  CONSTANT constr_3:time_3:=130 ns;
  CONSTANT constr_4:time_4:=500 ns;
  CONSTANT constr_5:time_4:=650 ns;
END constraints;

. . .
PROCESS
  VARIABLE anchor1, anchor2: TIME;
  . . .
BEGIN
  . . .
  ANCHOR(anchor1);
  . . .
  EXACT_TIME(constr_1, anchor1);
  . . .
  IF cond THEN
    ANCHOR(anchor2);
    . . .
    MAX_TIME(constr_2, anchor2);
  ELSE
    . . .
    ANCHOR(anchor2);
    . . .
    MIN_TIME(constr_3, anchor2);
    . . .
  END IF;
  . . .
  RANGE_TIME(constr_4, anchor1);
  . . .
END PROCESS;
. . .
```

Fig. 2. Timing constraints on sequences of statements.

passing as an argument the value of the time interval associated with the constraint. The location of the call determines the end point (the *sink*) of the constrained sequence. The starting point of the statement sequence is defined by an *anchor*. To refer to the anchor, each call to a predefined timing procedure contains a second argument representing a variable of type *time*. The anchor point is defined by the precedent call to a timing procedure that contains as an argument the same time variable. An example of TC specifications is given in Fig. 2, where the constrained sequences are indicated by arrows.

The predefined package *time_restrict*, implemented as part of our design environment, exports the timing procedures *anchor*, *range_time*, *min_time*, *max_time*, and *exact_time*. It is natural that both the anchor and the sink of a TC have to be located in the same branch of an *if* statement, variant of a *case*, and body of a *loop*, *procedure*, or *process*.

The time values associated with the constraints are specified as constants of a subtype of type *time*. Similar to the approach proposed in [3], at synthesis the ranges of these subtypes are identified as the constraint limits. The values of the time constants are ignored by the synthesis tool.

For simulation, the constraint associated with a time constant (the range corresponding to its type) is not relevant, but its value is considered. This value is passed as a parameter to the procedure exported by the package *time_restrict* and is considered by the VHDL simulator. In our example these values are specified in the body of package *constraints*. For pre-synthesis simulation the values are estimated by the

```
PACKAGE BODY time_restrict IS
  PROCEDURE wait_delay(delay: TIME; t_anchor: INOUT TIME) IS
    VARIABLE to_wait: TIME;
  BEGIN
    to_wait:=delay-(NOW-t_anchor); -- time left for wait
    IF to_wait>=0 THEN
      WAIT FOR to_wait;
    ELSE
      ASSERT FALSE--already more time spent then expected
        REPORT "timing restriction error"
          SEVERITY WARNING;
    END IF;
    t_anchor:=NOW;     -- set time for anchor
  END wait_delay;
  PROCEDURE range_time(delay: TIME; t_anchor: INOUT TIME) IS
  BEGIN
    wait_delay(delay, t_anchor);
  END range_time;
  . . .
  PROCEDURE anchor(t_anchor: OUT TIME) IS
  BEGIN
    t_anchor:=NOW;     -- set time for anchor
  END anchor;
  . . .
END time_restrict;
```

Fig. 3. Part of the package *time_restrict.*

designer; after synthesis they are automatically replaced at back-annotation with the synthesized times which are then considered for post-synthesis simulation.

At simulation the four procedures *range_time*, *min_time*, *max_time*, and *exact_time* act in a similar way. We illustrate this with a sequence from the package body *time_restrict* in Fig. 3. Simulation of the delay on the constrained sequence is solved by the *wait for* statement in the procedure *wait_delay.* The actual wait is for the amount of time (*to_wait*) left after previous waits executed inside the constrained sequence corresponding to the specified anchor. This solution supports, according to our requirements, nested TCs.

The synthesis tool ignores the body of the predefined timing procedures. The procedures are recognized by their name and the corresponding constraint is translated into the internal design representation, with time limits corresponding to the range of the subtype associated to the time constant.

4. Simulation/Synthesis Correspondence

The most difficult issues concerning hardware synthesis of VHDL specifications originate from the VHDL semantics of signal assignments, wait statements, and the timing model, which are specified in terms of simulation. We have developed and implemented two strategies for high-level synthesis of behavioral VHDL descriptions containing interacting concurrent processes [5]. These strategies preserve the partial ordering relation of operations on signals and ports from the simulation model to the synthesized hardware structure. Thus we achieve *simulation/synthesis correspondence* which means that the simulation model and the synthesized hardware react with the same values (sequences of values) of the signals and ports to identical sequences of stimuli applied at the inputs.

For VHDL specifications containing TCs, simulation/ synthesis correspondence implies the conditions stated above and the correspondence between the timing behavior of the constrained sequences in the synthesized hardware and in the back-annotated simulation model.

4. 1. Design Representation

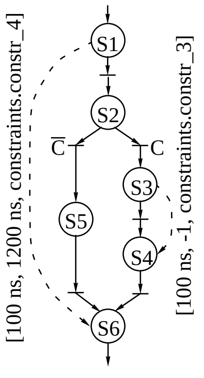
The internal design representation of CAMAD, called ETPN (Extended Timed Petri Net) [12], has been developed to capture the intermediate results during the high-level synthesis process. The ETPN representation, which VHDL specifications are translated to, consists of a control part and a data path. The *control part* is represented as a timed Petri net with restricted transition firing rules. Transfer of data in the data path is controlled by control signals coming from the control part. A control signal is generated when a token is deposited at a Petri net place. A Petri net transition may be guarded by one or several conditions produced from the data path. It may be fired when it is enabled (all its input places have a token) and the guarding condition is true. In Fig. 4 we show an ETPN control part and its corresponding VHDL sequence.

TCs are captured by the ETPN representation as additional arcs in the control part (represented as dotted lines in Fig. 4). They are attributed with the time limits associated to the constraint and with an identifier corresponding to the respective time constant (the identifier is provided for back-annotation). Such an arc is placed between the control places corresponding to the start and to the end of the constrained sequence.

4. 2. Synthesis of Concurrent VHDL Processes

In [5] we presented two models for specifying interacting VHDL processes, the unrestricted model and the reduced synchronization model. The *unrestricted model* offers the freedom to express process interaction at the level of signal assignments and wait statements. From the point of view of synthesis the model implies practically the hardware implementation of the simulation cycle in order to preserve simulation/synthesis correspondence. This means that processes have to wait for each other, until all of them are executing a wait statement, before updating the signal values.

The synthesis strategy corresponding to the *reduced synchronization model* does not reproduce the simulation cycle in hardware, while maintaining simulation/synthesis corre-


[100 ns, 1200 ns, constraints.constr_4]
S1
S2
C̄
C
S3
S5
S4
S6
[100 ns, -1, constraints.constr_3]

```
. . .
ANCHOR(anchor1);
x:=z+1;             -- S1
IF x>0 THEN         -- S2
  ANCHOR(anchor2);
  x:=a+2;           -- S3
  y:=x-b;           -- S4
  MIN_TIME(constr_3,anchor2);
ELSE
  x:=a*2;           -- S5
END IF;
y:=y*x;             -- S6
RANGE_TIME(constr_4,anchor1);
. . .
```

Fig. 4. An ETPN controll part and its corresponding VHDL code.

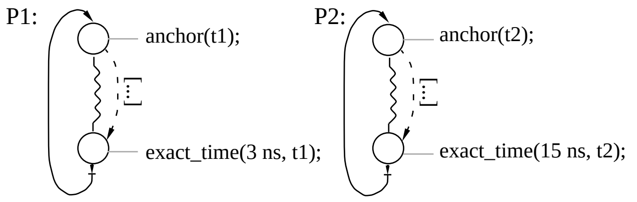


Fig. 5. Parallel processes with timing constraints.

spondence. According to this model VHDL processes interact using a synchronous message passing mechanism with predefined *send*/*receive* commands. Communication channels are represented by VHDL signals. Assignment of a value to a signal is done with a *send* command. Processes that refer to the signal will wait until a value is assigned to it, by calling a *receive* command.

The correct behavior of the hardware described conforming to the reduced synchronization model does not rely on the implicit synchronization enforced by the simulation cycle. Thus the synthesis strategy corresponding to this model does not need to implement the simulation cycle in the synthesized hardware [5].

4. 3. Simulation/Synthesis Correspondence with the Unrestricted Model

The consequence of the synthesis strategy entailed by the unrestricted model on the behavior of a hardware specification with TCs is illustrated by processes *P1* and *P2* with their control part depicted schematically in Fig. 5. Both processes consist of a statement sequence constrained by an exact TC. At simulation, according to VHDL semantics, the statement sequences will be executed in 0 (simulation) time before the processes are stopped in the wait state (as result of the wait statement executed in procedure *wait_delay* defined by package *time_restrict*); process *P1* will be restarted after 3 ns simulation time and thus the constrained statement sequence inside its body will be activated four more times before reactivation of process *P2* (after 15 ns simulation time).

After synthesis the generated structure needs 3 ns and 15 ns respectively for the execution of the sequences in *P1* and *P2*. However, as result of the synthesis strategy discussed in the previous section, after reaching the wait statement, process *P1* will have to wait for process *P2* to reach the same state. Hence, both processes will execute their sequences stepwise the same number of times. A shorter execution time for *P1* will result in a longer time to wait for global synchronization with process *P2*. It is obvious that such a circuit works differently both from simulation behavior and from the designer’s intuitive expectation. The difference is due to the delay time interpretation at simulation and at synthesis: at simulation the prescribed time is entirely spent in the wait state while the synthesized hardware spends this time along the constrained sequence and thus prevents all other processes from leaving a wait state. The consequence for our synthesis strategy, if we are accepting TCs, has to be the following:

Inside a constrained sequence a process should behave as if it would be in a wait state; this means that, from the point of view of global synchronization, the process executing a constrained sequence should not prevent the update of signal values and reactivation of other processes.

This relaxation of the global synchronization of processes can produce a nondeterministic behavior of the synthesized hardware, if no restrictions on the use of signals in specifications containing TCs are introduced. We use processes *Q1* and *Q2* in Fig. 6 to illustrate this problem. According to the simulation semantics, the value of signal *s* will be updated only when both processes *Q1* and *Q2* are executing a wait statement; the first two references to signal *s* in process *Q2* will always result in the same value. If processes are synthesized according to the rules discussed above, the value of *s* will be updated if *Q1* executes the wait and *Q2* executes any statement in the constrained sequence. Thus the behavior of the synthesized circuit differs from that of the simulation model and, in *Q2*, the value of signal *s* in the second reference can differ from that in the first one; this will happen if at the moment that *Q1* reaches the wait, *Q2* executes a statement of the constrained sequence preceding the reference to signal *s*.

Determinism in standard VHDL semantics is based on the fact that process interaction is strictly concentrated to the states when all processes are executing a wait. To preserve such a determinism in the circumstances of our synthesis strategy with TCs we have to exclude any operation that is related to process interaction from the constrained sequences. This results in the following restriction on the TC specifications:

No access to signals is allowed inside a constrained sequence; these sequences execute internal operations with local variables, which neither influence nor are affected by other processes. Interaction between processes (signal assignments, references to signals, wait statements) should be executed outside the constrained sequences.

4. 4. Simulation/Synthesis Correspondence with the Reduced Synchronization Model

As mentioned in section 4.2, synthesis with the reduced synchronization model does not implement global synchronization in hardware. The correct behavior of a hardware specified in VHDL according to this model does not rely on

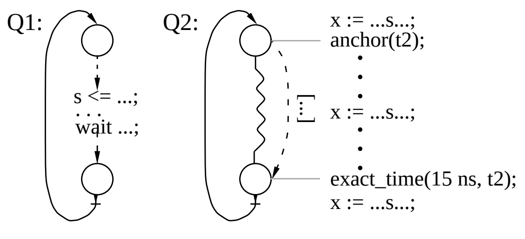


Fig. 6. Operations on signals in constrained sequences.

the global synchronization imposed by the simulation cycle, but only on the synchronization explicitly stated with the *send*/*receive* operations. As a consequence, the synchronization mechanism and timing are orthogonal and no additional rules or restrictions have to be introduced in order to guarantee simulation/synthesis correspondence with TCs.

This results in no restriction on the statements in a constrained sequence. However, if some of these statements (like *send* and *receive*, for instance) require a data dependent delay, it is possible that synthesis which satisfies the required constraints under any circumstances has to be rejected [9].

5. Specification of TCs Across Process Borders

The previous sections have focussed on TC specifications on sequences of statements inside a process. However, design restrictions can also be expressed as constraints on the time interval between certain events on signals and it is likely that these events are produced by different processes.

In order to reason about synchronization and communication across process borders, and specifically about the relative timing of operations, process interaction has to be specified at a higher level than that using VHDL signal assignment and wait statements. Hence, we have based the mechanism for timing specification across process borders on our reduced synchronization model.

We have decided to use for our notation the *concurrent assert statement* and to specify constraints as follows:

```
ASSERT condition
  REPORT "timing constraint violation"
    SEVERITY WARNING;
```

The *condition* has to be a call to one of the four predefined boolean functions exported by package *time_restrict*: *range_assert*, *exact_assert*, *min_assert*, or *max_assert*. The function will be selected according to the kind of the constraint that has to be specified; function parameters define the two signals affected by the constraint and the constraint limits. In the example presented in Fig. 7, the *assert* statement specifies that a transaction on signal *b* has to come not earlier than 100 ns and not later than 800 ns after a transaction on signal *a*. It has to be mentioned that in the reduced synchronization model, a transaction or event on a signal can be produced only as result of a *send* operation.

During simulation the concurrent *assert* statement will be triggered at each *send* executed on signal *a* or *b*, and an assert violation occurs if the constraint is not satisfied. We present, for illustration, the predefined function *range_assert*, as defined in the body of package *time_restrict*:

```
FUNCTION range_assert (SIGNAL a,b:IN BIT; t1,t2:IN TIME)
         RETURN BOOLEAN IS
BEGIN
  RETURN NOT b'ACTIVE OR
    (NOW-a'LAST_ACTIVE>=t1 AND NOW-a'LAST_ACTIVE<=t2);
END range_assert;
```

The synthesis tool translates the *assert* statement into a constraint represented as one or several edges in the internal design representation. If a time constraint has been expressed

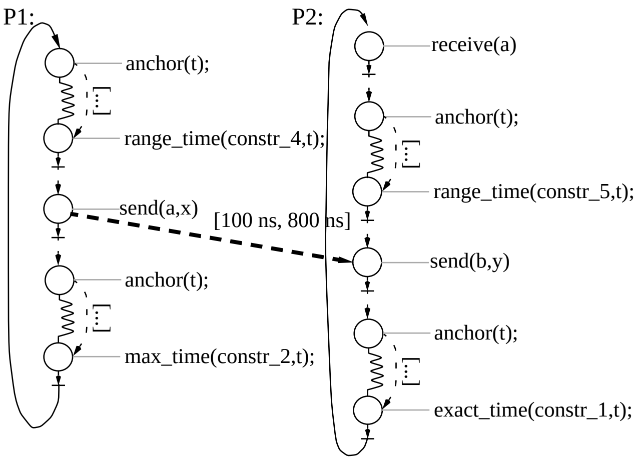


Fig. 8. Representation of timing constraints across process borders.

between signals *a* and *b* then an arc has to be generated between each control place corresponding to a *send* on *a* and each place corresponding to a *send* on *b*. In Fig. 8 we show a representation corresponding to the example in Fig. 7.

A VHDL design that contains no other TCs than those on signals (specified by *assert* statements) results in a correct synthesis, according to the specified timing requirements (if timing analysis and operation scheduling are possible), but no simulation with the TCs can be performed. During such a simulation, the *assert* statement would report a violation after each *send* on the constrained signals, because there is no operation in the VHDL model that produces any progress of the simulation time.

Such a shortcoming will be eliminated with a complete design, like that in Fig. 7. Both TCs on sequences of statements inside the processes and constraints on signals have

```
ARCHITECTURE synth OF synthesis IS
  USE constraints.ALL;  -- package constraints is defined in Fig. 2
  USE time_restrict.ALL;
  SIGNAL a,b: INTEGER;
BEGIN
  ASSERT RANGE_ASSERT(a'TRANSACTION,b'TRANSACTION,100 ns, 800 ns)
    REPORT "timing constraint violation"
      SEVERITY WARNING;
  P1:PROCESS
    VARIABLE t: TIME; VARIABLE x: INTEGER;
  BEGIN
    ANCHOR(t);
    . . .
    RANGE_TIME(constr_4,t);
    . . .
    SEND(a,x);
    . . .
    ANCHOR(t);
    . . .
    MAX_TIME(constr_2,t);
  END PROCESS P1;
  P2:PROCESS
    VARIABLE t: TIME; VARIABLE y: INTEGER;
  BEGIN
    RECEIVE(a);
    . . .
    ANCHOR(t);
    . . .
    RANGE_TIME(constr_5,t);
    . . .
    SEND(b,y);
    . . .
    ANCHOR(t);
    . . .
    EXACT_TIME(constr_1,t);
  END PROCESS P2;
END synth;
```

Fig. 7. Timing constraints across process borders.

```
ASSERT RANGE_ASSERT(inp'TRANSACTION,outp'TRANSACTION,t1,t2)
  REPORT "timing constraint violation"
    SEVERITY WARNING;
              -- this constraint is provided only when
              -- process P1 is interacting with P2
. . .
P1: PROCESS                                   P2: PROCESS
  . . .                                       . . .
BEGIN                                         BEGIN
  RECEIVE(inp);                               . . .
  ANCHOR(t);                                  SEND(data,inp);
  -- algorithm (ell. filt. or DFT)              . . .
  RANGE_TIME(constr, t);                      END PROCESS P2;
  SEND(outp, result);                         . . .
END PROCESS P1;
```

Fig. 9. VHDL skeleton used for experiments

been provided. The global constraints, expressing timing requirements on signals, are significant in this context both at simulation and synthesis. Simulation verifies if local constraints satisfy global requirements on signals, expressed by assert statements. At synthesis the global constraints are interpreted as additional restrictions that have to be considered when exploring of the design space spanned by the constraints on sequences of statements.

6. Experimental Results

Table 1 gives the results obtained for two benchmarks, which have been synthesized with TCs. We performed synthesis of both the elliptic filter [12] and the 256-point Discrete Fourier Transform (DFT) circuit [10] in two different contexts (Fig. 9). First we synthesized each benchmark as a process containing a local constraint (*P1* in Fig. 9). The input in this case is received from port *inp*. We then considered the process containing the benchmark algorithm in interaction with another process (*P2* in Fig. 9) from which it gets its input data through signal *inp*. In addition to the previous local constraint inside the benchmark specification, we specified a constraint for the time interval between the execution of a send on signal *inp* (when *P2* sends the data) and on signal *outp* (when the result is produced by *P1*).

These experiments have been performed on both benchmarks and the results given in Table 1 show how the additional constraints imposed on signals influence the synthesis process. Usually, additional functional units are used in order to meet these constraints, as indicated in Table 1.

Table 1: Synthesis results

Benchmark	Constraints (in µs)	Synthesis results[1]				CPU time (s)[2]
		exec. cycles	cycle time (ns)	exec. time (µs)	funct. units	
elliptic filter	local: 0.1..0.25	19	11.63	0.220	2 +, 1 *	78.4
	local: 0.1..0.25 signals: 0.1..0.2	15	11.63	0.174	3 +, 2 *	20.6
DFT	local: 1000..2000	68559	27.12	1859.320	1 ALU, 1 *	3.7
	local: 1000..200 signals: 1000..1500	52175	27.12	1414.986	1 ALU, 1 +, 1 *	2.6

1. The LSI 10K technology is used.
2. These experiments have been carried out on a SPARCstation 10.

7. Conclusions

We have presented a notation for TC specifications in VHDL for high-level synthesis. Based on the requirements adopted in the first section, we have decided to use predefined procedures for the specification of constraints on sequences of statements. Our notation accepts nested TCs with specification of minimum, maximum, range, and exact limits and supports back-annotation for post-synthesis simulation. We have also provided support for specification of TCs across process borders.

One of the major problems of defining and implementing a mechanism for TC specifications in the context of VHDL synthesis is how to preserve consistency between the behavior of the simulation model and that of the synthesized hardware. We have proposed solutions to this problem in the frame of our synthesis strategies and have shown in the paper how simulation/synthesis correspondence can be achieved for VHDL specifications containing both concurrent processes and TCs.

References

[1] J. Biesenack, et. al., *The Siemens High-Level Synthesis System CALLAS*, IEEE Trans. on VLSI, vol. 1, no. 3, Sept. 1993, 233-243.

[2] R. Camposano, A. Kunzmann, *Considering Timing Constraints in Synthesis from a Behavioral Description*, Proc. ICCD, 1986, 6-9.

[3] W. Ecker, S. März, *Subtype Concept of VHDL for Synthesis Constraints*, Proc. of EURO-DAC/VHDL'92, 1992, 720-725.

[4] P. Eles, K. Kuchcinski, Z. Peng, M. Minea, *Compiling VHDL into a High-Level Synthesis Design Representation*, Proc. of EURO-DAC/VHDL'92, 1992, 604-609.

[5] P. Eles, K. Kuchcinski, Z. Peng, M. Minea, *Synthesis of VHDL Concurrent Processes*, Proc. of EURO-DAC/VHDL'94, 1994, 540-545.

[6] D. Gajski, N. Dutt, A. Wu, S. Lin, *High-Level Synthesis, Introduction to Chip and System Design*, Kluwer Academic Publisher, 1992.

[7] P. Gutberlet, W Rosenstiel, *Timing Preserving Interface Transformations for the Synthesis of Behavioural VHDL*, Proc. of EURO-DAC/VHDL'94, 1994, 618-623.

[8] *IEEE Standard VHDL Language Reference*, IEEE Std. 1076-1993, IEEE Comp. Soc. Press, 1993.

[9] D.C. Ku, G. De Micheli, *Relative Scheduling Under Timing Constraints: Algorithms for High-Level Synthesis of Digital Circuits*, IEEE Trans. on CAD, vol 11, no. 6, June 1992, 696-717.

[10] P. Michel, U. Lauther, P Duzy, *The Synthesis Approach to Digital System Design*, Kluwer Academic Publisher, 1992.

[11] K. Nordqvist, *Timing Specification and Back-Annotation in High-Level Synthesis*, Technical Report - Swedish Institute of Microelectronics, 1993.

[12] Z. Peng, K. Kuchcinski, *Automated Transformation of Algorithms into Register-Transfer Level Implementation*, IEEE Trans. on CAD, vol. 13, no. 2, Feb. 1994, 150-166.

[13] Synopsys, *VHDL Compiler Reference Manual*, V. 3.0, Nov. 1992, Chapter 11, VHDL Compiler Directives.