Verification by Simulation Comparison using Interface Synthesis

Cordula Hansen[*], Arno Kunzmann[*], Wolfgang Rosenstiel[+]
FZI, Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany[*]
Universität Tübingen, WSI-TI, Sand 3, 72076 Tübingen, Germany[+]
{hansen, kunzmann}@fzi.de, rosen@peanuts.informatik.uni-tuebingen.de

Abstract

One of the main tasks within the high-level synthesis (HLS) process is the verification problem to prove automatically the correctness of the synthesis results. Currently, the results are usually checked by simulation. In consequence, both the behavioral specification and the HLS results have to be simulated by the same set of test vectors. Due to the HLS and the inherent changes in the cycle-by-cycle behaviour, the synthesis results require an adaption of the initial test vector set. This reduces the advantage gained by using the automated HLS process. In order to decrease these simulation efforts, in this paper a new method will be presented that enables the usage of the same simulation vectors at both abstraction levels and the execution of an automated simulation comparison.

1 Introduction

With high level synthesis (HLS) tools, the circuit behavior can be specified at an abstract behavioural level. The specification is realized as an algorithmic description usually by VHDL programs [8] and then transformed into a register transfer (RT) design. HLS is a well-known research area (e.g. [2], [3], [7]), and the first commercial tool [9] is already available. However, one open question in HLS domain is the verification of the synthesized RT design. Several verification methods are currently available, but usually, the results are checked by simulation. A set of simulation vectors is created for the simulation of the algorithmic specification. Due to the HLS and the resulting changes in the cycle-by-cycle behaviour, an adaption of the original vectors or a complete new set of simulation vectors for the generated RT design is necessary. Simulations at both levels have to be performed and the results have to be checked. In the case of the RT simulation, the examination of the results is a difficult task, because the RT design is generated code and, therefore, the designer has to analyse unknown source code. Assuming that simulation is one of the most time intensive steps, this significantly reduces the advantage gained by using the automated HLS process. In order to decrease these simulation efforts, automated simulation methods are necessary. One straightforward method is the usage of the same set of simulation vectors for the simulation of the algorithmic specification and the simulation of the generated RT design. If a common set is used, an automated simulation comparison is possible. During the creation of common simulation vectors, the different cycle-by-cycle behaviour has to be regarded. During the generation of the comparison, also the different semantics of the specification language and the realized interpretation of this semantics in HLS process with request to the simulation task has to be considered.

One of the first systems, dealing with these problems, is the Satya System [4]. In Satya, a new specification is generated by adding scheduling results from synthesis to the original specification. The generated specification and the synthesized RT implementation are then checked for equivalence. The disadvantage is obvious, since instead of the original specification the generated specification is checked for equivalence. Another verification approach based on simulation comparison is presented in [1]. Here, special hardware structures are created in the RT design which allow a comparison between the specification simulation and the RT simulation at some synchronization points. The applicability of this approach is limited due to the required internal change of the HLS process.

This paper presents a new approach for comparing simulation results of an algorithmic VHDL specification and a synthesized RT design, also represented as a VHDL description, using the interface synthesis of a HLS process. One common simulation set is used and the normal HLS process has not to be changed. In this paper, first, the boundary conditions for an efficient simulation comparison will be discussed. Next, an overview of the used design flow is given and the proposed simulation comparison using the interface synthesis is described. The enclosed sample design will illustrate the application of the proposed method and underline the efficiency and simplified portability onto other HLS systems.

2 VHDL for simulation and high-level synthesis

When using HLS the circuit behavior can be specified at an abstract level by synchronous VHDL programs. The data path as well as the controller part are synthesized starting at this behavioral specification. In HLS, an important step is the scheduling, which assigns the operations to control steps and determines the degree of parallelism and the timing of the circuit [6]. Therefore, the scheduling may change the cycle-by-cycle behavior of the initial specification. For example, an operation which needs one clock cycle in the specification may need several clock cycles in the synthesized RT design. To give the designer control over the scheduling step, and thus the possibility to determine the degree of change in the resulting cycle-by-cycle behaviour, different I/O scheduling modes have been developed (e.g. [7], [9], [11]). These modes represent alternative interpretations of the VHDL timing concepts.

In the HLS system used in this approach four different scheduling modes are realized. Three of them are based on the approach represented in [9], the last one was presented in [7]. In [9], a VHDL specification style was defined with three different I/O scheduling modes. These modes are the "cycle-fixed" I/O scheduling mode, the "superstate" I/O mode and the "free-floating" I/O scheduling mode. When using the "cycle-fixed" I/O scheduling mode, the simulation vectors need not to be adapted since the cycle-by-cycle behavior of the specification is unchanged. The I/O operations, this means the read and write signals of the design, are tied to particular cycles. Only other operations, e.g. additions, register reads and writes, could be shifted in time. Therefore, in this scheduling mode, the main advantage is, that the designer synthesize the exact timing behaviour of the initial specification. Thus, a direct simulation comparison is possible. However, the designer has to have an exact know-how about the required circuit, and only few optimization possibilities are left for the scheduler. When using the "superstate" I/O mode, the scheduler has some more optimization alternatives and as a result, the cycle-by-cycle behaviour may change. The scheduler can shift I/O operations in time, but only within the boundaries between two adjacent clock edge expressions. These two clock statements form the boundary of one superstate in which all I/O operations remain. Now, the scheduler can add clock cycles to lengthen the superstate (Figure 1). Consequently, a direct cycle-by-cycle comparison is not possible and would lead to incorrect simulation results. Thus, some transformations have to be performed to use the initial simulation vectors. When using the "free-floating" I/O mode, the permutation of I/O operations and clock edge expressions are possible. The timing constraints are explicitly defined by the designer without any reference to the simulation behaviour. Therefore, it is impossible to reuse the same simulation vectors. With this scheduling mode, the designer convey the most optimization possibilities to the scheduler.

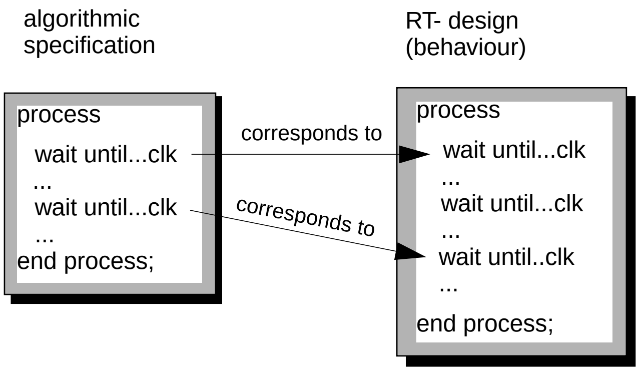

Figure 1: Adding clock cycles in the "superstate" I/O scheduling mode

In [7], a VHDL description style is defined using a "min/max constraint" I/O scheduling mode. The cycle-by-cycle behaviour of the specification is defined by minimum and maximum time conditions in absolute timing values. Therefore, the designer has not to have the exact know-how about the required circuit. Just as the "cycle-fixed" I/O scheduling mode, the synthesized result has exactly the same timing behaviour as the original specification. Thus, a direct simulation comparison is also possible.

In Table 1, the different scheduling modes are classified corresponding to there comparison capability.

direct cycle-by-cycle comparison	cycle-by-cycle comparison with transformation	cycle-by-cycle comparison impossible
cycle-fixed scheduling mode	superstate scheduling mode	free-float scheduling mode
min/max constraint scheduling mode		

Table 1: Cycle-by-cycle comparison and scheduling modes

In summary, the "cycle-fixed" and the "min/max constraint" I/O scheduling mode can be compared directly. With the "free-float" I/O scheduling mode a cycle-by-cycle comparison is impossible, due to the permutation of I/O operations and clock edge expressions. Consequently, the main research work concerning verification by simulation comparison is concentrated on the "superstate" I/O scheduling mode, where a cycle-by-cycle comparison is possible by some transformations.

3 Simulation Comparison by Using Interface Synthesis

3.1 Design flow

The simulation comparison realized in this design flow is based on the "superstate" I/O scheduling mode. This mode is implemented in the CADDY II synthesis system. A detailed overview of the methodology used in CADDY II is given in [10]. In the following, the design flow is described and the integration of the scheduling mode generation and the simulation comparison is shown in Figure 2.

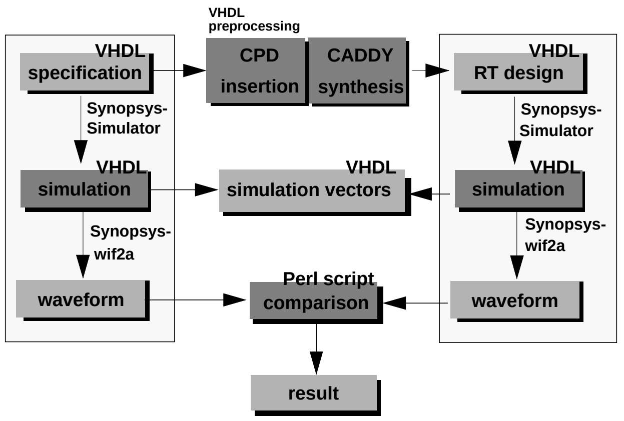

Figure 2: Overview of designflow with synthesis and simulation comparison

In the first step, the VHDL preprocessing is started. One main task during the preprocessing is the realization of the different scheduling modes ("constraint realization"). The CADDY II synthesis process is based on the "free-float" I/O scheduling mode. Other modes can be realized by setting corresponding constraints. In the VHDL specification an attribute has to be set which defines the required scheduling mode. Then the preprocessor generates the necessary constraints for the scheduler. Due to this strategy, any scheduling mode between the "cycle-fixed" and the "free-float" mode could be generated. The second main task during the preprocessing is the preparation of the simulation comparison, e.g. the insertion of the comparison point detection (CPD) signal. This task is described in more detail in section 3.2. Finally, the preprocessing generates a flow graph description, the interchange format for the CADDY II system, and the synthesis process is started without any change for the simulation comparison. The resulting RT design is represented by a VHDL description. As soon as the synthesis process is finished, the simulation of the specification and the simulation of the RT design can be started. In this case, the Synopsys simulator was used, but also other VHDL simulator can be applied. After simulation, the results are automatically stored in a Synopsys specific data format and then transformed into ASCII files. Finally, these files are compared with a simple Perl script and the results are displayed. Assuming the interface component used for the simulation comparison is correctly specified, the designer can then decide whether there are any faults in the synthesis process. If any differences occur, the synthesis process has generated an invalid RT description.

3.2 Realization of the comparison point detection

When realizing a simulation comparison using the "superstate" I/O scheduling mode, the following requirements has to be fulfilled:

- common simulation vectors (simulation environment) have to be used for the simulation of the algorithmic specification and the RT design
- a handshake protocol has to be implemented because of the different timing behaviour of the specification and the synthesized design (section 3.2.1)
- the simulation comparison points of the specification and the RT design have to be defined (section 3.2.2)
- a signal has to be generated that displayed the comparison points in the RT design (section 3.2.3)

3.2.1 Implementing a handshake protocol

To implement a common simulation environment for simulations with different timing behaviour, a handshake protocol has to be implemented. This ensures that the values of the input signals can not change faster than the design can calculate the results. Basically, several handshake protocols are possible. The most commonly used types are the two phase handshake protocol and the four phase handshake protocol. In the following example implementing a greatest common divider (*gcd*) algorithm a four phase handshake protocol is realized:

```
ARCHITECTURE behaviour OF gcd IS
BEGIN
    PROCESS
```

```
        VARIABLE x1, x2 : UNSIGNED(7 DOWNTO 0);
    BEGIN
        WAIT until clock‘event and clock = ‘1‘ and
                in_ready = "1";
        x1 := in1;
        x2 := in2;

        IF (x1 /= SIGNED‘("00000000") and
            x2 /= SIGNED‘("00000000")) THEN
            WHILE x1 /= x2 LOOP
                IF x1 < x2 THEN
                    x2 := x2 - x1;
                ELSE
                    x1 := x1 - x2;
                END IF;
            END LOOP;
            outp  <= x1;
        ELSE
            outp <= "00000000";
        END IF;

        out_ready <= "1";

        WAIT UNTIL clock‘event and clock = '1' and
                        out_request = "0“;

        out_ready <= "0";
    END PROCESS;
END;
```

First, the simulation environment provides an *in_ready* signal which indicates that the *gcd* processing can be started. As soon as the outputs are calculated the design set an *out_ready* signal and indicates that the results can be picked up. The environment confirm the pick-up by setting an *out_request* signal to the value zero and in the last step, the design reconfirm this by setting the *out_ready* signal back to the value zero.

Further, the example contains two "wait until" statements. In the "superstate" I/O scheduling mode, these two boundaries define one superstate, in which the whole *gcd* algorithm is executed.

3.2.2 Definition of the simulation comparison point

The next question that has to be discussed is, where could the specification simulation and the simulation of the RT design be compared. In the "superstate" I/O scheduling mode two adjacent clock edge statements are defined as a superstate [9]. The scheduler can insert new clock edges in a superstate, however, the initial clock edges will remain (Figure 1). Thus, the simulation comparison points are already specified by this definition (Figure 3). As an illustration, the *gcd* example contains two "wait until clock" statements which forms one superstate. Hence, these two clock edge statements are the points where the simulations could be compared.

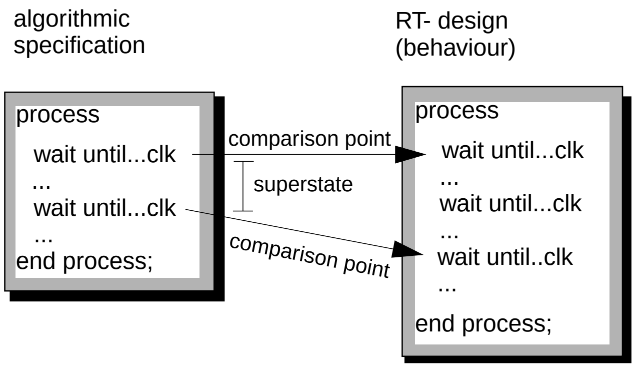

Figure 3: Comparison points in the "superstate“ I/O scheduling mode

One last open question concerning this comparison point definition has to be answered. In a VHDL simulation, all signals have to be updated at the clock edges. Therefore, the output signals in the specification are stable until the next assignment is executed. The synthesis process now interprets this simulation behaviour and transforms it into the behaviour of a real circuit with some delays. Currently, four different interpretations are used for output signals:

1. After a defined setup time, the output signals are stable until the next assignment is executed
2. At an exact defined hold time, the output signals are stable until the next assignment is executed
3. After a defined setup time, the output signals are stable for the rest of the clock period
4. At an exact defined hold time, the output signals are stable for the rest of the clock period

Using interpretation 1 or 2 in the synthesis process, the output signals are stable until the next assignment is executed. Therefore, in the generated RT design, the signals are stable until the next comparison point is executed (Figure 4). The necessary registers for the output signals are already generated and the simulation results could be compared without any change to the synthesis process. Using interpretation 3 or 4, the output signals are only stable for the rest of the clock period. As a result, the signals are not stable until the next comparison point is executed (Figure 4). Synthesis systems using these interpretations have to generate extra registers for the simulation comparison. Most of the synthesis systems including the CADDY II system realize interpretation 1 or 2 and, therefore, a re-implementation of the synthesis system is not necessary.

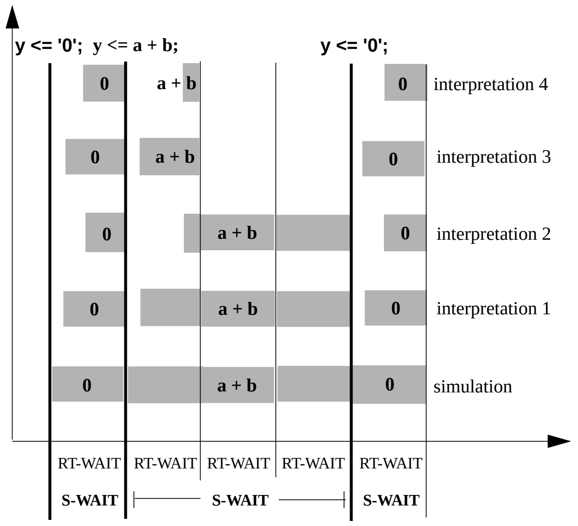

Figure 4: Interpretations used for output signals in the synthesis process

3.2.3 Generation of the comparison point detection signal

Finally, the comparison points have to be displayed in the RT design. The easiest way to perform the comparison is by generating a special internal signal [1]. This signal is used to determine the comparison points between the simulation results of the specification and the simulation results of the RT design. In this approach, the signal is named CPD (comparison point detection signal) and can be generated by using the interface synthesis. The advantage of this method is especially the efficiency and the simplified portability onto other HLS systems. A change in the synthesis process in not necessary. The signal can simply be generated by defining a special procedure and the corresponding component. Further necessary properties, e.g. a correct interpretation for output signals, are already integrated in the normal interface synthesis process of most HLS systems.

Generally, the CPD signal generation can be divided in four steps:

1. Generation of a procedure *write1_cpd.*
2. Inserting the procedure *write1_cpd* at every clock edge in the specification.
3. Implementation of an interface component *WRITE_CPD*.
4. Synthesizing the specification with the usual synthesis process

First, a procedure *write1_cpd* has to be generated. Currently, there are two possibilities available: the VHDL preprocessor generates the procedure or the designer develops it on it‘s own. The procedure *write1_cpd* has one output signal CPD. This signal is set to the value one for one clock period and is then set to the value zero.

Second, this procedure has to be inserted in the specification. One possibility is that the designer inserts the procedure at every clock edge statement in the specification:

```
ARCHITECTURE behaviour OF gcd IS
BEGIN
    PROCESS
        VARIABLE x1, x2 : UNSIGNED(7 DOWNTO 0);
    BEGIN
        WAIT until clock‘event and clock = '1' and
                    in_ready = "1";
        write1_cpd(CPD);

        -- algorithmic specification of the gcd
        ...
        out_ready <= "1";

        WAIT UNTIL clock‘event and clock = '1' and
                        out_request = "0“;
        -- second comparison point
        write1_cpd(CPD);

        out_ready <= "0";
    END PROCESS;
END;
```

The other possibility is, that the VHDL preprocessor inserts the *write1_cpd* procedure at every clock edge during the generation of the intermediate format for the CADDY II system. In this case, the *write1_cpd* procedure would not appear in the algorithmic specification. Now, the procedure is inserted at every clock edge in the specification and set the CPD signal to the value one for one clock period. As a result, during a specification simulation, the CPD signal has always the value one.

The third step is the implementation of an interface component *WRITE_CPD*. During the CADDY II synthesis process, every procedure has to be replaced by a component. These components are defined in the CADDY II library. Now, a new component for the *write1_cpd* procedure has to be defined in this library*:

```
Component "WRITE_CPD"
    Interface
        In  1 "Start" Ctrl 1
        Out 2 "CPD" Data 1
    Gates 1200
    Area  2000 1000
    Power 1000
```

* In the meantime, components for the CADDY II library could also be described in VHDL.

```
        Operation *write1_CPD
              Fix 1 "Start" 1 0
    End
```

Specifying this component, a first synthesis property has to be considered: the synthesis process is able to generate an input signal *Start* that indicates exactly when this component is activated. This property has to be realized from every HLS system as soon as it supports sequential components that are multi-cyclic and non pipelined. Hence, this property can be used, to realize the comparison point detection. In the RT design, the signal *Start* is set to the value one as soon as the component *WRITE_CPD* is activated. As mentioned before, the procedure corresponding to this component was inserted at every clock edge in the specification. As a result, the component is activated at every clock edge in the RT design (RT-WAIT) that corresponds to a clock edge in the specification (S_WAIT). Therefore, this activation indicates the comparison points (Figure 5).

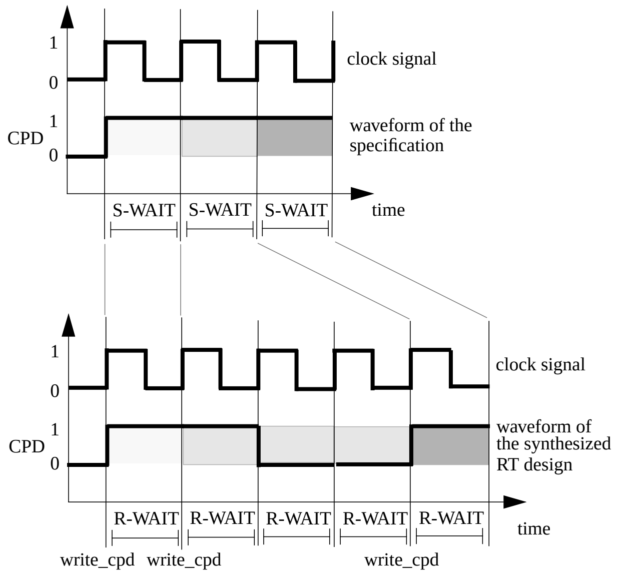

Figure 5: write_cpd before and after synthesis process

A second synthesis property used here is the interpretation of the component behaviour. The component behaviour is interpreted exactly as the procedure behaviour. This means, a component is activated exactly when the corresponding procedure is called. For example, the CPD signal of the procedure *write1_cpd* is set to the value one for one clock cycle. The CPD signal of the component *WRITE_CPD* has exactly the same behaviour. As soon as the component is activated, the signal is set to the value one for one clock period. During the scheduling, the component behaviour does not change. For every new clock cycle inserted due to the "superstate" I/O scheduling mode, the CPD signal is automatically set to the value zero. In most other HLS systems, the CPD signal is also set to the value one as soon as the component is activated, however, for every new clock cycle inserted the CPD signal is undefined. Therefore, in this cases, the designer has to define explicitly that the CPD signal has to be set to the value zero. In the fourth and final phase, the synthesis process is executed. The procedure *write1_cpd* and the component *WRITE_CPD* are used as a normal interface component taking advantage of the synthesis properties realized in CADDY II. Consequently, changes of the synthesis process are not necessary. Afterwards, the resulting RT design can be simulated

3.2.4 Generation of the CPD signal with other HLS systems

To implement the generation of the CPD signal the designer has to specify a procedure and a corresponding component. The procedure as well as the component can be realized with any other HLS system. For these HLS systems, there exist only the requirements concerning the output signal interpretation and the interface synthesis properties that have already been described. In more detail, the following steps have to be performed:

1. The simulation behaviour of the output signals has to be interpreted accordingly interpretation 1 or 2 as described in Figure 4. Especially interpretation 2 is commonly used, and thus, is generally available.
2. Concerning the interface synthesis, two properties are required: the HLS system must be able to generate a signal that indicates when the component is activated and it has to be allowed to define a default value of this signal when the component is activated and currently in its processing phase. These properties are available in all HLS systems which support the definition of sequential components that are multi-cyclic and non pipelined.

3.3 Comparing simulation results

In this approach, the Synopsys Simulator is used for the simulation of the specification and of the generated RT design. After synthesis, both simulations are started and the results are stored in a Synopsys specific waveform format. With the Synopsys wif2a program, this format is transformed into an ASCII format. Now, the results can be compared, e.g. with a simple Perl script. According to the definition of the CPD signal, the signal values can be compared when the CPD signal is set to the value 1 at a rising clock edge. Therefore, the comparison algorithm is quite simple. As long as the last simulation cycle is not executed, the CPD signal is scanned. As soon as the CPD signal is 1 at a rising clock edge, the signal values of the specification and of the RT design are compared in the following cycle; otherwise a comparison is not possible.

For illustration purposes, the waveforms of the specification and the synthesized RT design implementing the *gcd* algorithm are given in Figure 6. Here, the transformation results are shown for seven clock cycles and the first two phases of the specification, respectively.

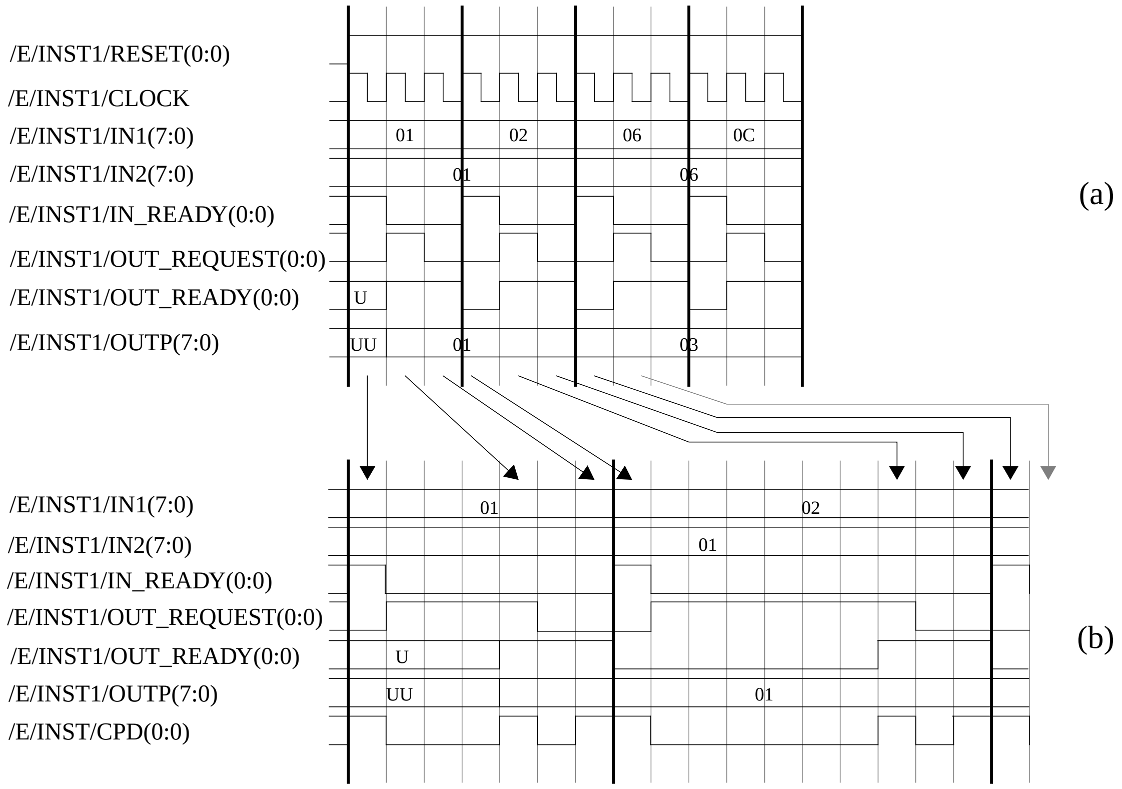

Figure 6: The CPD signal during the *gcd* simulation at (a) specification level and (b) RT level

4 Conclusion

In this paper, a new verification technique has been presented for verifying the HLS results through a simulation comparison. A key feature of this approach is that no changes of the HLS process are required. For an easier comprehension of the approach, the paper first provided a classification of different scheduling modes and their suitability for a cycle-by-cycle simulation comparison. Based on the "superstate" I/O scheduling mode, the detection of the required comparison points are described. This detection is realized by using the interface synthesis. Therefore, the designer has only once to define a new interface procedure and the corresponding interface component such that the HLS system generates automatically the necessary signal for the comparison point detection. The related synthesis step does not require any specific property for the generation of this signal. The user defined interface component is handled in the same way as a common library component. Finally, the automated simulation comparison process is described. The proposed verification technique has been implemented in the CADDY II system and several examples have been synthesized and compared successfully.

5 References

[1] Bergamaschi, R. A.; Raje, S.: *"Observable Time Windows: Verifying the Results of High-Level Synthesis“*, Proceedings of the European Design & Test Conference, March 1996.

[2] Camposano, R.; Kunzmann, A.; Rosenstiel, W.: *"Automatic Data Path Synthesis from DSL*

Specifications“, International Conference on Computer Design, ICCAD 1984.

[3] DeMicheli, G.: *"Synthesis and Optimization of Digital Circuits*“, McGraw-Hill Inc., 1994.

[4] Ernst, E.; Bhasker, *"Simulation-based verification for high-level synthesis"*, IEEE Design & Test of Computers, vol. 8, pp. 14-20, March 1991.

[5] Gajski, D.; Dutt, N.; Wu, A.; Lin. S.: *"High-Level Synthesis*“, Kluwer Academic Publishers, 1992.

[6] Gutberlet, P.; Krämer, H.; Rosenstiel, W.: *"CASCH - a Scheduling Algorithm for High Level -Synthesis*“, Proceedings of the EDAC, February 1991.

[7] Gutberlet, P.; Rosenstiel, W.: *"Timing Preserving Interface Transformations for the Synthesis of Behavioural VHDL*“, Proceedings of EURO-DAC, September 1994.

[8] *"IEEE Standard VHDL Language Reference Manual*“, IEEE Std 1076-1993, June 1994.

[9] Knapp, D.; Ly, T.; MacMillen, D.; Miller, R.: *"Behavioural Synthesis Methodology for HDL-Based Specification and Validation*“, Proccedings of the DAC, June 1995.

[10] Krämer, H.; Rosenstiel, W.: *"System Synthesis using Behavioural Descriptions*“, Proceedings of the EDAC, March 1990.

[11] Stoll, A.; Duzy, P.: *"High-Level Synthesis from VHDL with Exact Timing Constraints"*, Proceedings of the DAC, 1992.