A Canonical Form of RT-Level FSM Controlled Data Path Descriptions for Formal Verification

Péter Keresztes

Széchenyi István University
Egyetem tér 1, H-9026 Győr, Hungary
keresztp@sze.hu

Abstract: The paper proposes a new canonical form for RT-level descriptions, which can be systematically generated from both the specification and the structural description. The verification can be executed with the comparison of the two generated canonical form descriptions.

1 Introduction

When it comes to the designing of digital systems, a description in accordance with a well-chosen canonical form provides grounds for the efficient methods of the formal verification and the symbolic simulation, alike. The logic (gate-level) synthesis, along with the verification and the symbolic simulation are all based on the canonical forms, which borrows its tools from the classic switching algebra. In the aspect of their application on computer design systems, particularly successful was Roth's cube algebra, which is based on a new wording of Boole's canonical forms [1].

The descriptions of the register transfer level have up to the present lacked the universality and heuristic power, which characterises the switching algebra. Thus, the canonical forms employed on the register level could only be applied to a restricted scale of tasks. To this category belongs, for instance, the Taylor-polynomial method, which is capable of verifying the register-level structures of arithmetic expressions, but has its limits within this very class [2], [3].

The implementation of the register transfer level canonical description suggested by the author of the present paper is conditional on the same requirements as those forming the principle of the most part of designing methods. The data-path structure is controlled by a synchronous finite state machine (FSM), as a controller built around a core. The structure must clearly reflect that in a specific state of the FSM, as an interval:

1 Which sub-paths of the data-path are switched active by the multiplexers,

and

2 Into which registers and on what conditions occurs entering of data.

On condition that the structure's description meets the requirements above, the canonical form, as suggested by this paper, can be prepared.

At the same time, an identical canonical description is gained from the algorithm-level specification, which is a behavioural description, formulated in one of the high level programming languages. If the canonical description, gained from the structure, and the behavioural description are provably homomorphous, – even at the expense of certain permissible transformations – the verification process can be considered successful.

2 Decomposition of Sequential Behavioral Descriptions

We decompose the program, constituted by sequential statements, into a hierarchical structure of modules, between the statements modifying the control, as bordering points. In the sequential subset of VHDL-processes the control branch statements are the following:

begin . . . . end

wait until . . . .

for . . loop. . . . .end loop

while . . .loop . . . end loop

if . . . then . . .else . . . end if

The example below is the abstract style VHDL behavioural description of a hardware unit in charge of carrying out the algorithm of square root calculation. *Figure 1* shows the way we decompose the description into modules, and the way these modules and their attachments constitute the state-graph of an abstract state machine. It is important to formulate the variable-assignment statements of the description through functions that are implemented by the components (function-units) of the hardware structure.

```
library work; use
work.sqrtpack.all;
entity SQRT_UNIT is
  port ( START : in bit;
         READY : inout bit := '1';
         RESET : in bit;
         pe : in real := 0.0;
         px : in real:= 0.0;
         py : inout real := 0.0;
         ph1, ph2 : in bit);
end SQRT_UNIT;

architecture BEH of SQRT_UNIT
is
 begin
 process
   variable e, x, y, cy, ny, v : real :=
   0.0;
   variable d : real := 1.0;
   variable f : bit := '1';
   variable g : bit;
  begin
   wait until START = '1';
   READY <= '0';
   wait for 1 ns;
   e := pe; x := px;
   cy := Fi(x);
   wait for 1 ns;

while f = '1' loop
      v := MD(div, x, cy);
      v := AS(add, cy, v);
      ny := MD(mult, 0.5, v) ;
      d := AS(sub,ny,cy);
      g := Cm(d, 0.0);
      if g = '0' then
         d := AS(sub, 0.0, d);
      end if;
      cy := ny;
      f := Cm(d, e);
    end loop;
   wait for 1 ns;
    py <= cy;
    READY <= '1';
end process;
end BEH;
```

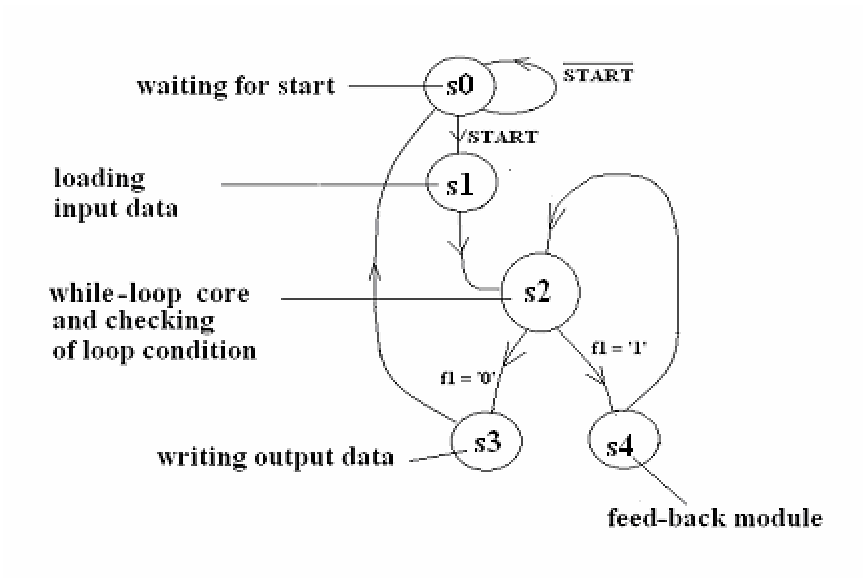

Figure 1
The decomposition of the square root algorithm into sequential modules

3 Generating Value-target Event-driven Data-flow Blocks from Behavioural Description

Consider the variable-assignment statements of a sequential module and the values ordered to the variables **v1, v2, . . . vj . . . .vn** by the sequence. Pick out the value of **vj** next in line, resulting from the next-in-line variable assignment. Formulate this in the following substitution expression:

$$vj(p+1) = E[\ldots v_k / vk(p) \ldots]$$

A value next in line of variable $\mathbf{v_j}$ can be calculated through the substitution of the present values of the variables of the right hand side into the variable-assignment statement. If we number the values of the variables of the sequence, from ***v1(0), v2(0), . . . vj(0), . . .vn(0)*** up to those terminal values of maximum indexes ***v1(t), v2(t), . . . vj(t), . . .vn(t),*** ordering one target to each and every value of each and every variable, and on the other hand, we order to each variable-assignment an event-driven concurrent statement,

$$wj(p+1) <= E[\ldots vk / wk(p) \ldots]$$

then from these statements we attain an event-driven dataflow-block, which can be ordered to the sub-sequence. This block is termed the value-target block (**VTB**) of the sequential module.

The **VTB** at rest is

$$w_j(t) = E[\ldots v_k / w_k(t) \ldots]$$

It is conceivable that if the initial value of the variable v_j is equal to the initial value of the target w_j, ordered to it, then the value of v_j, with which it leaves the sequence module, is also equal to the terminal value of the target of the maximal index. One sequence is therefore *value-equivalent* to the value-tracking block gained from it. See a simple example:

```
SEQ1: begin
        for i in 1 to 4 loop
          a := a + 1;
        end loop;
end;
```

```
VTB1 : block
   begin
     a1 <= a0 + 1;
     a2 <= a1 + 1;
     a3 <= a2 + 1;
     a4 <= a3 + 1;
end block;
```

A more complex one:

```
SEQ2 : begin
          if e < 0 then a := b * c;
                        d := a + b;
          elsif  e = 0 then
                        a := b;
          else
                        d := a;
          end if;
     end;
```

```
VTB2 :  block  begin
             a1 <= b0 * c0  when
                      e0 < 0  else
             b0 when e0 = 0  else
                      a0;
             d1 <= a1 + b0 when
                      e0 < 0 else
             d0 when e0 = 0  else
                      a0;
        end block;
```

Now complement the abstract state-transition graph, attained from the square root algorithm, with the **VTB**s of the particular modules. Hence will be obtained the description in accordance with *Figure 2*. Hereafter, this is regarded as the canonical form of the specification.

4 Notations

In *Figure 2* a possible form of FSM controlled value-target blocks are shown. The meaning af notations which are used in the blocks can be explained by the semantics of VHDL statements. *Table 1* shows that form of canonical description of the specification, which will be compared with the canonical form of the structure.

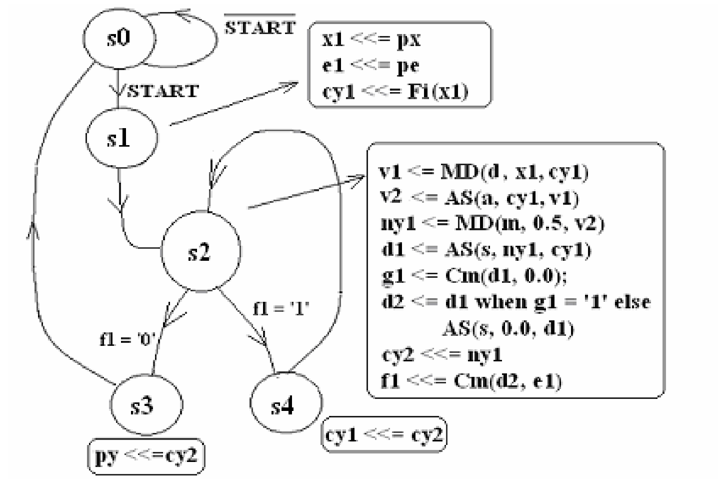

Figure 2
The canonical form without time-refinement of the square root calculation's specification

Target/state	**s0**	**s1**	**s2**	**s3**	**s4**
v1			**MD(d,x1,cy1)**		
ny1			**MD(m,0.5,py)**		
v2			**AS(a,cy1,v1)**		
d1			**AS(s,ny1,cy1)**		
g1			**Cm(d1, 0.0)**		
d2			**d1 when g1 = ’1’ else AS(s, 0.0, d1)**		
e1		**pe**			
x1		**px**			
cy1		**Fi(x1)**			**cy2**
cy2			**ny1**		
f1			**Cm(d2, e1)**		
py				**cy2**	

Table 1
Canonical form of specification. The simple- (<=) and the register-type (<<=) transactions are isolated parts of the first column.

The simple transaction **v1 <= MD(d, x1, y1)** given in a box ordered to state **s2** of **FSM** can be expressed as follows:

v1 <= MD(d, x1, y1) when fsm_state = s1 else anyvalue;

The transaction for **v1** is a *non-register-type* statement.

An other transaction, for example **cy1 <<= Fi(x1)** given in the fsm-state **s1** together with the other transaction with the same target in fsm-state **s4**, (**cy1 <<= cy2)** can be interpeted as follows:

```
cy1 <= Fi(x1) when fsm_state = s1 and fsm_phase = ph2 else
            cy2 when  fsm_state = s4 and fsm_phase = ph2 else
            cy1;
```

The two transactions for **cy1** constitute *register-type* statement. It has to be emphasided that each register type transaction is executed by a *phase* of an **FSM** state. The phase has to be an inner time interval of the of the **FSM** state.

5 Characteristics of the Proposed Canonical Form Description

Two sequential descriptions can be fully equivalent in spite of the number of variables or the order of statements within them being different. The canonical form described above shows some very important features. These are the following:

1 Unaffected by the number of the variables of the specification's equivalent forms.

2 Unaffected by the order of statements in the modules' equivalent forms.

3 Unaffected by the number of FSM states deriving from hardware limitations.

4 Unaffected by the allocations of function-unit, register and multiplexer, which derive from hardware limitations.

The first characteristic derives from the fact that the description orders the target-signals to the values of the variables, which means that the canonical forms of two equivalent sequential modules are identical, irrespective of the difference between the number of their respective variables. The second feature derives from the fact that we convert the modules into data-flow blocks composed of concurrent statements, and thus the canonical forms of equivalent sequential modules applying different orders of statements are also identical. The third characteristic derives from the fact that the states, whose number has been increased because of the necessity generated by the hardware limitations, can be contracted during the transformations of the canonical form that describes the structure. The fact that units lose their identities during the transformations of the structure-describing canonical form, and appear in the changed canonical description only through their functions (similarly to the way they do in the canonical description of the specification) accounts for the fourth characteristic.

6 Process of Verification of a RT-level Unit

The **RT**-level structure to be verified is shown in *Figure 3*, *Figure 4*, and *Table 2*. The description has to contain the structure of **DATA-PATH** (*Figure 3*) and the state-transition graph of the **FSM** (*Figure 4*). The fuction units of the **DATA-PATH**:

- One multiplier/divider unit (**M/D**)
- Two adder/subtractor units (**A/S**)
- Two comparators (**Cm**)
- A special look-up table unit for deriving of initial approximation of square-root. (**Fi**)

Above these components the **DATA-PATH** contains 6 registers and 7 multiplexers. *Table 2* shows the initial form of the structural description to be verified. There is a part of the targets which contain outputs of multiplexers and function units, while another part of them contain outputs of registers. These parts are isolated in the left side of the list. There are state-independent transactions in the structure, and they are isolated in the left side of the list.

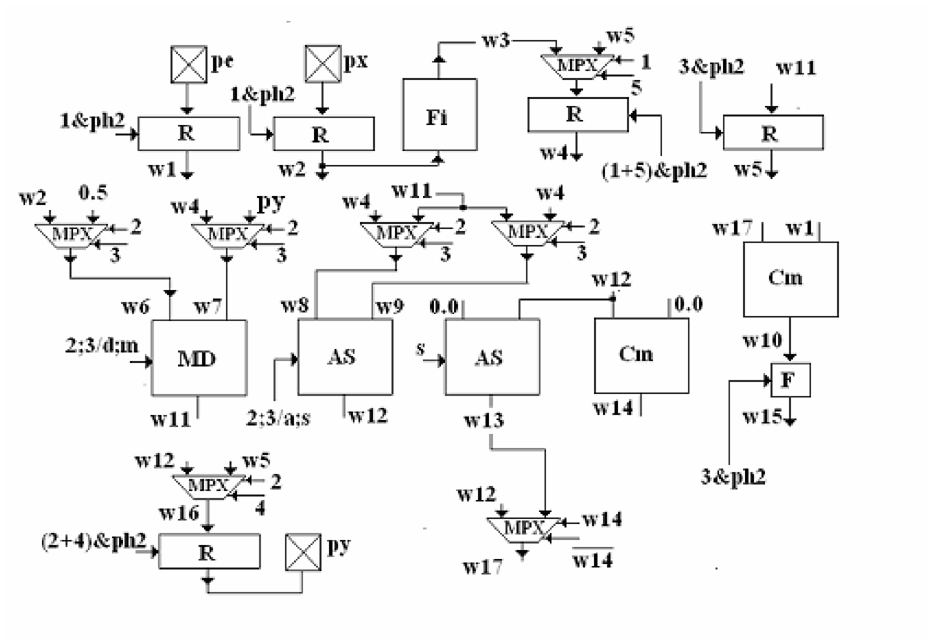

Figure 3
RT-level structure of square-root calculation unit

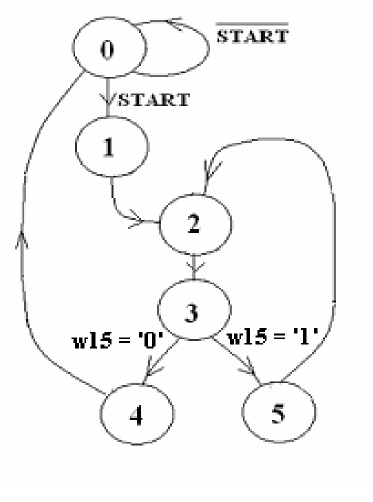

Figure 4
The graf of counter-based FSM, which controls the data-path of square-root calculation unit

Target/state	**0**	**1**	**2**	**3**	**4**	**5**
w3	**Fi(w2)**					
w6			**w2**	**0.5**		
w7			**w4**	**py**		
w8			**w4**	**w11**		
w9			**w11**	**w4**		
w10	**Cm(w17,w1)**					
w11			**MD(d,w6,w7)**	**MD(m,w6,w7)**		
w12			**AS(a,w8,w9)**	**AS(s,w8,w9)**		
w13	**AS(s,0.0,w12)**					
w14	**Cm(12, 0.0)**					
w16			**w12**		**w5**	
w17	**w12 when w14 = '1' else w13**					
w1		**pe**				
w2		**px**				
w4		**w3**				**w5**
w5				**w11**		
w15				**w10**		
py			**w16**		**w16**	

Table 2
The source of the structural canonical description, which is derived from the implementation

7 Transformation Steps of the Verification Process

The application of the following transformation steps leads to the canonical form of structural description which can be compared with the canonical form of the specification. They are based on the semantical equivalency of some parts of the structural description and corresponding abstract data-flow expressions. The name of each step is a reference to the structural analogy of the given transformation.

7.1 Placement of State-Independent Transactions into States

The first step of transformation is the placement of the state-independent transactions into those states, in which the target of the transaction, or the target of another transaction which is driven by it is stored in a register. This step is based on the recognition that the target-value of a state-independent transaction is *'don't care'* in those states, in which it is not stored.

For example, given a state independent transaction

wi <= EXPi

and **wi** is used in state **nl** as follows:

nl : wj <= EXPj(. . . wi . . .), wk <<= wj.

In this case the result of the placement is the following:

S[nl] : wi <=EXPi wj <= EXPj(. . . wi . . .) wk <<= wj

7.2 Node Elimination

In the second phase of the transformations those targets are eliminated inside a given **FSM** state, which are not stored in the given state, and they are used at the right side of another target. It can be shown, that this step can eliminate all the nodes, the signals represented by which do not belong to a behavioural description. Assume that in fsm-state **nl** the following transactions are given:

nl : wi <= wj wk <= EXP(. . . wi . . .)

The result of the node elimination is as follws:

S[nl] : wk <= EXP(. . . wj . . .)

7.3 Merging Subsequent Loopless States

The number of **FSM** states in canonical specifications is minimum, but in the structural description because of the hardware constraints it can be much higher. The **FSM** states that are introduced only because of the contraints of the number of function units are subsequent, and there is no control feedback between them.

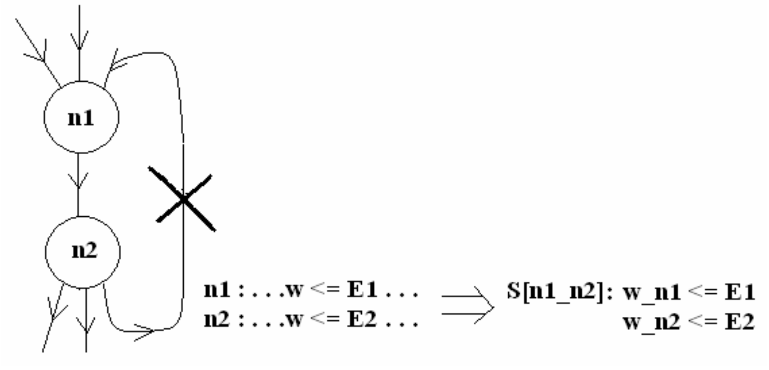

Figure 5
Merging subsequent FSM states with data-independent transactions

To get closer to the canonical form, a merging of these states is proposed. *Figure 5* shows the simpler case, when there is no such signal which is stored in state **n1** and used in state **n2**. In a more complex case, when a value of a signal is stored in a register, and it is used in the next state, the register after merging is eliminated. (*Figure 6*)

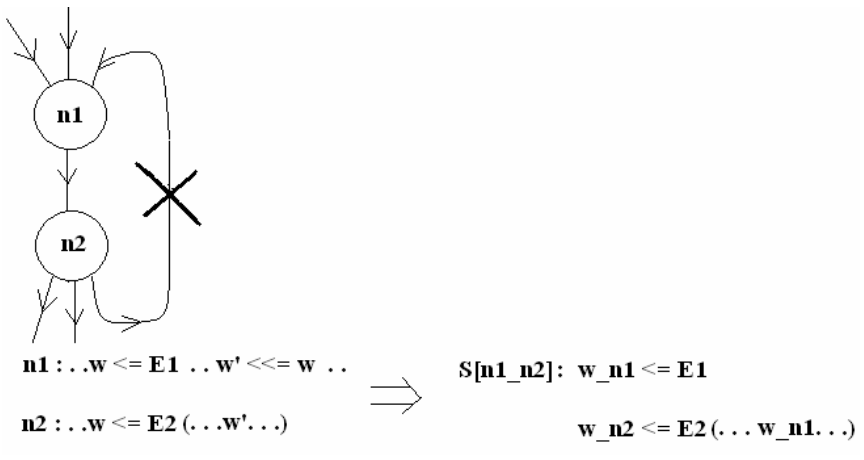
n1
n2
n1 : . .w <= E1 . . w' <<= w . .
n2 : . .w <= E2 (. . .w'. . .)
S[n1_n2]: w_n1 <= E1
w_n2 <= E2 (. . . w_n1. . .)

Figure 6

Merging subsequent FSM states with a common signal, which is stored in state **n1** for state **n2**

8 Generation of the Structural Canonical Form of Square-Root Unit

The following series of tables from *Table 3* to *Table 7* illustrates the verification flow of the hardware implementation of square-root procedure. The intermediate forms and the application of the three transformation steps lead to the structural canonical description.

Table 3 is the result of placement of state-independent transactions. For example the transaction **w3 <= Fi(w2)** was placed in **state 1**, because **w3** is stored in **ph2** phase of the **state 1**. The result of node eliminations is shown in *Table 4*. For example **w6** is eliminated, since **w6** is driven by **w2** in **state 2**, and **w6** is used in the driver **MD(d, w6, w7)** in the same state. So **w2** substitutes **w6** in driver of **w11**.

The result of merging state **’2’** and state **‘3’** are shown in the *Table 5*. The *Figure 5* which shows the state-transition graf of the implemntation, proofs that ‘**2**’ and ‘**3**’ are subsequent and loopless states. Since the targets **w11** and **w12** are driven in both states, after the merging both of them have to duplicated.

Target/state	**0**	**1**	**2**	**3**	**4**	**5**
w3		**Fi(w2)**				
w6			**w2**	**0.5**		
w7			**w4**	**py**		
w8			**w4**	**w11**		
w9			**w11**	**w4**		
w10				**Cm(w17,w1)**		
w11			**MD(d,w6,w7)**	**MD(m,w6,w7)**		
w12			**AS(a,w8,w9)**	**AS(s,w8,w9)**		
w13				**AS(s,0.0,w12)**		
w14				**Cm(12, 0.0)**		
w16			**w12**		**w5**	
w17				**w12 when w14 = ’1’ else w13**		
w1		**pe**				
w2		**px**				
w4		**w3**				**w5**
w5				**w11**		
w15				**w10**		
py			**w16**		**w16**	

Table 3

Result o the placement of the state independent transactions

Target/state	**0**	**1**	**2**	**3**	**4**	**5**
w11			**MD(d, w2, w4)**	**MD(m, 0.5, py)**		
w12			**AS(a, w4, w11)**	**AS(s, w11, w4)**		
w14				**Cm(12, 0.0)**		
w17				**w12 when w14 = ’1’ else AS(s, 0.0, w12)**		
w1		**pe**				
w2		**px**				
w4		**Fi(w2)**				**w5**
w5				**w11**		
w15				**Cm(w17,w1)**		
py			**w12**		**w5**	

Table 4

Result of the node elimination

Table 6 is the result of the attempt, the goal of which to find a consistent cross-reference list between the nodes of the canonical description of the structure and the signals of the canonical specification. If the nodes of *Table 5* are replaced by the signals of the cross-regference list, *Table 7* is derived. It is obvious, that the structural canonical description covers the canonical specification, because the corresponding signals appear in every correspondig cells of the table, where the driver is specified.

Target/state	**0**	**1**	**2 3**	**4**	**5**
w11_1			**MD(d, w2, w4)**		
w11_2			**MD(m, 0.5, py)**		
w12_1			**AS(a, w4, w11_1)**		
w12_2			**AS(s, w11_2, w4)**		
w14			**Cm(w12_2, 0.0)**		
w17			**w12_2 when w14 = ’1’ else AS(s, 0.0, w12_2)**		
w1		**pe**			
w2		**px**			
w4		**Fi(w2)**			**w5**
w5			**w11_2**		
w15			**Cm(w17,w1)**		
py			**w12_1**	**w5**	

Table 5
Result of the merging state ‘2’ and state’3’

Signals from the structural desription	**Signals from the canonical behavioural description**
w1	**e1**
w2	**x1**
w4	**cy1**
w11_1	**v1**
w12_1	**v2**
w17	**d2**
w12_2	**d1**
w11_2	**ny1**
w5	**cy2**
w15	**f1**
w14	**g1**

Table 6
Equivalence between the signals of the structure and the signals of specification

Target/state	**s0**	**s1**	**s2**	**s3**	**s4**
v1			**MD(d,x1,cy1)**		
ny1			**MD(m,0.5,py)**		
v2			**AS(a,cy1,v1)**		
d1			**AS(s,ny1,cy1)**		
g1			**Cm(d1, 0.0)**		
d2			**d1when g1 = '1' else AS(s,0.0,d1)**		
e1		**pe**			
x1		**px**			
cy1		**Fi(x1)**			**cy2**
cy2			**ny1**		
f1			**Cm(d2,e1)**		
py			**v2**	**cy2**	

Table 7

Application af signal equivalences as a final step of verification

Conclusions

The new canonical form detailed above seems to be capable of developing an algorithm and an automatic verification system. The work intended to work out an implementation of the algorithm has been started.

References

[1] M. A. Breuer: Design Automation of Digital Systems, Prentice-Hall Inc, 1972

[2] M. Ciesielsky, P. Kalla, Z. Zeng and B. Rouzeyre: Taylor Expansion Diagrams: A new Representation for RTL Verification, IEEE Intl. High Level Design Validation and Test Workshop (HLDVT'01), 2001, pp 70-75

[3] P. Kalla, M. Ciesielsky, E. Boutillon, E. Martin: High Level Design Verification Using Taylor Expansion Diagrams: First Results, IEEE Intl. High Level Design Validation and Test Workshop (HLDVT'02), 2002, pp 13-17