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 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;
```

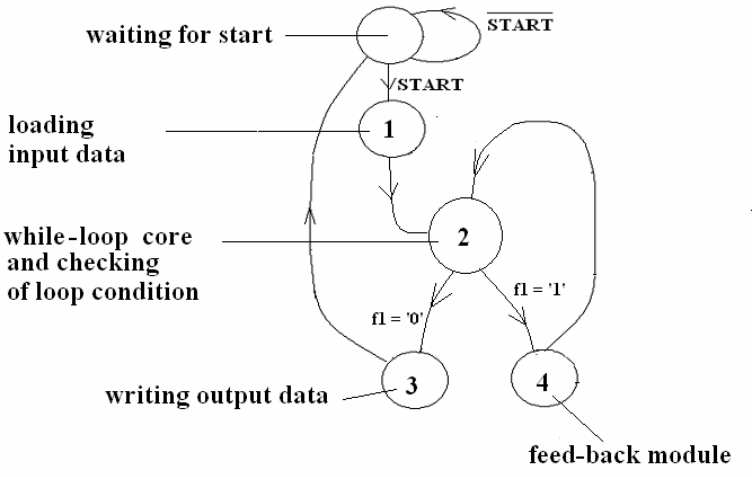


Figure 1
The decomposition of the square root algorithm into sequential modules

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 $\mathbf{v_1, v_2, \ldots v_j \ldots v_n}$ by the sequence. Pick out the value of $\mathbf{v_j}$ next in line, resulting from the next-in-line variable assignment. Formulate this in the following substitution expression:

$$v_{j(p+1)} = E[\ldots v_k / v_{k(p)} \ldots].$$

The 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 $v_{1(0)}, v_{2(0)}, \ldots v_{j(0)}, \ldots v_{n(0)}$ up to those terminal values of maximum indexes $v_{1(t)}, v_{2(t)}, \ldots v_{j(t)}, \ldots v_{n(t)},$ 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

$$w_{j(p+1)} <= E[\ldots v_k / w_{k(p)} \ldots],$$

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 $\mathbf{v_j}$ is equal to the initial value of the target $\mathbf{w_j}$, ordered to it, then the value of $\mathbf{v_j}$, 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 VTBs 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.

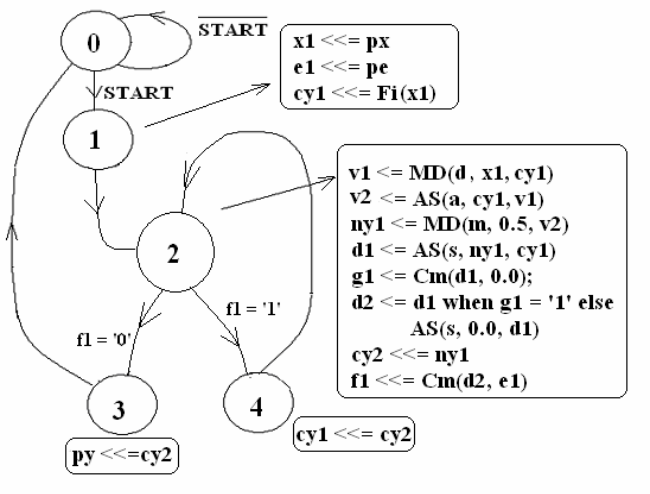


Figure 2
The canonical form of the square root calculation's specification

4 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

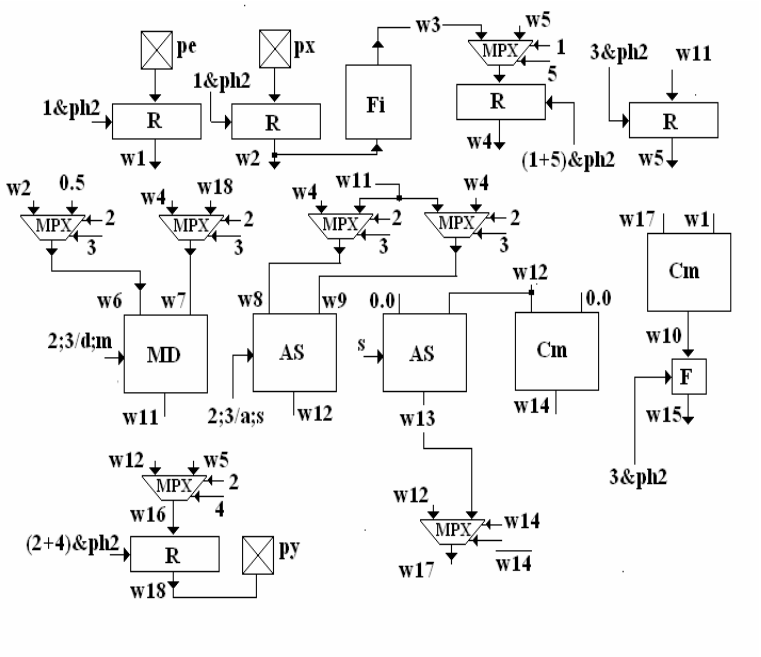
that we convert the modules into data-flow blocks composed of concurrent statements, and thus the canonical forms of equivalent sequential modules applying different orders of statements are also identical. The third characteristic derives from the fact that the states, whose number has been increased 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.

5 Process of Verification of a RT-Level Unit

The RT-level structure to be verified is shown in *Figure 3*. The description has to contain the structure of DATA-PATH (left side of the figure), and the state-transition graph of the FSM (right side of the figure). 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. The *Figure 4* shows the result of the initial step of transformations. The transactions are ordered in time intervals represented by the states of the FSM, and the clock phases (**ph2**) inside the states. There are state-independent transactions in the structure, and they are isolated in the left side of the list.



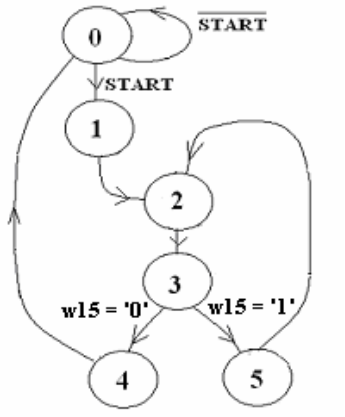


Figure 3

The RT-level structure of the square-root calculation to be verified

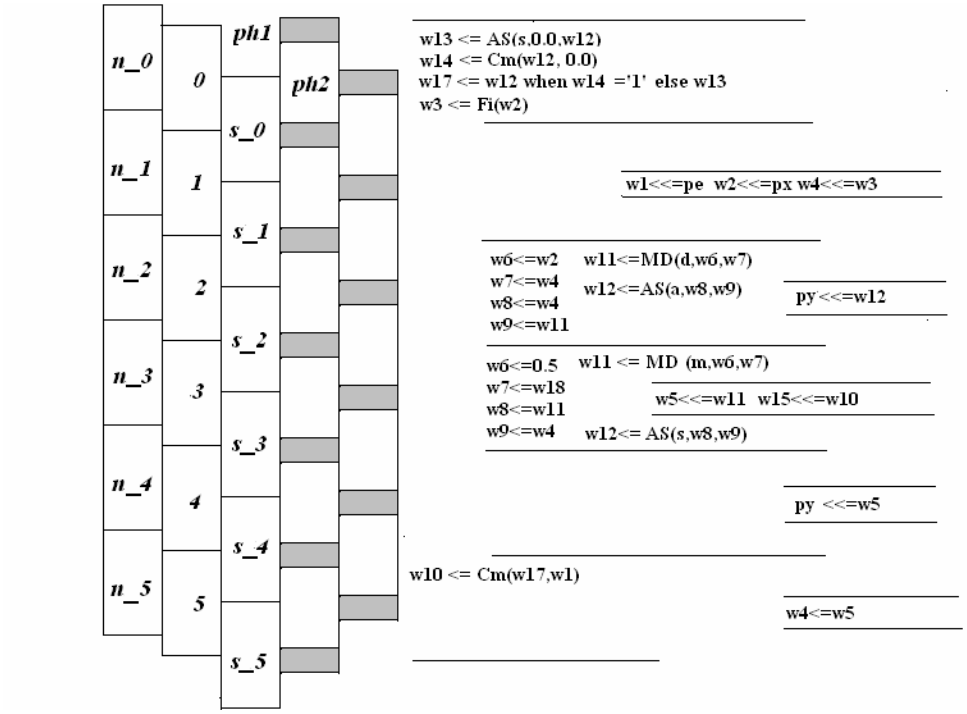


Figure 4
Initial canonical form of the structure of the square-root calculation unit

6 Transformation Steps of the Verification Process

1 *Simplification within a state:*

sk : wi <= wj ; wj <= driver1 → wi <= driver1;

sk: wi <<= wj; wj <= driver1 → wi <<= driver1;

2 *Placement of a state-independent transaction into states, if the target of the transaction has to be storaged only in one state (**si**).*

wi <= driver1 si: wj <<= wi → ~~wi <= driver1~~ si: wj <<= driver1

3 *Placement of a state-independent transaction into states, if the target of the transaction has to be storaged in more than one state.*

wi <= driver1 si: wj <<= wi sj: wj <<= wi

→ ~~**wi <= driver1**~~ **si: wj_si <<= driver1 sj : wj_sj <<= driver1**

4 *Distinguishing transactions, which have the same target in different states*

si : wk <= driver1 sj : wk <= driver2; → si : wk_si <= driver1 sj: wk_sj <=driver2

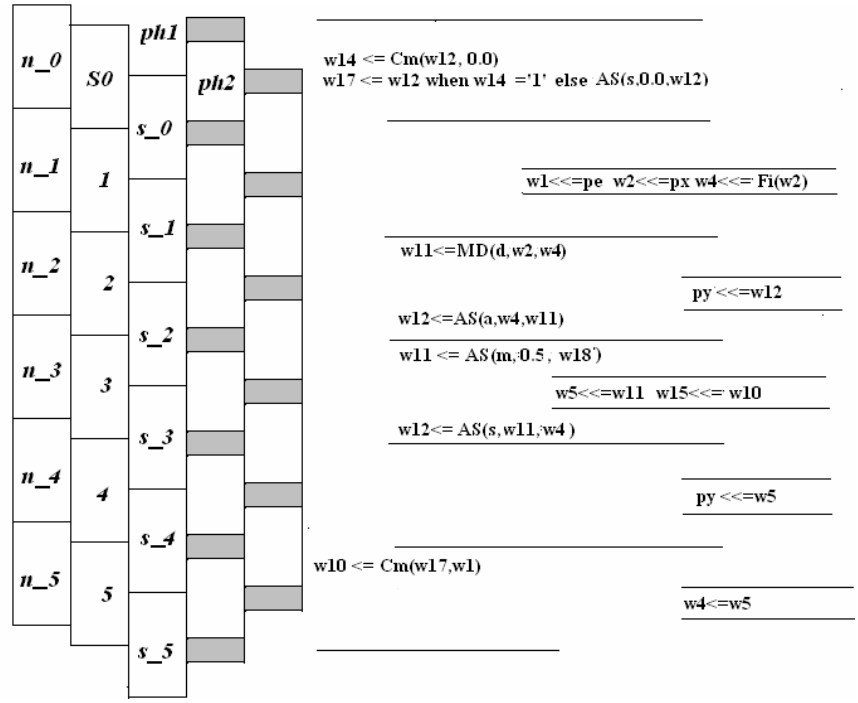


Figure 5

Transformation of the description deriving from Figure 4, based on the following transformation rules:

sk: wi<=wj; wj <=driver1 → sk: wi <=driver1;

sk: wi <<= wj ; wj <= driver1 → sk: wi<=driver1;

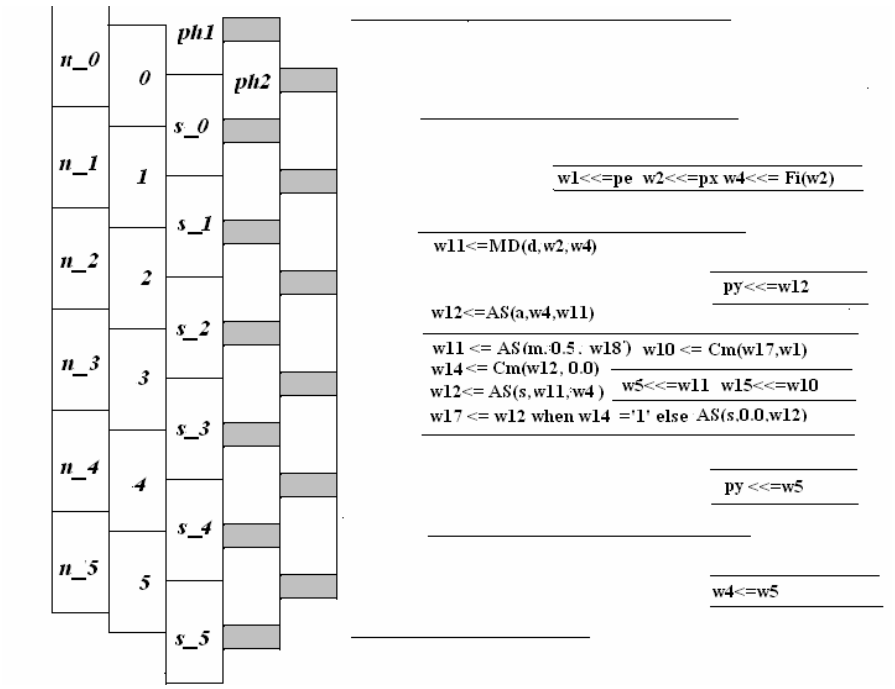


Figure 6
Transformation of the description deriving from Figure 5, based on the rule

wi <=driver1; si: wj<<= wi → si: wj <<= driver1

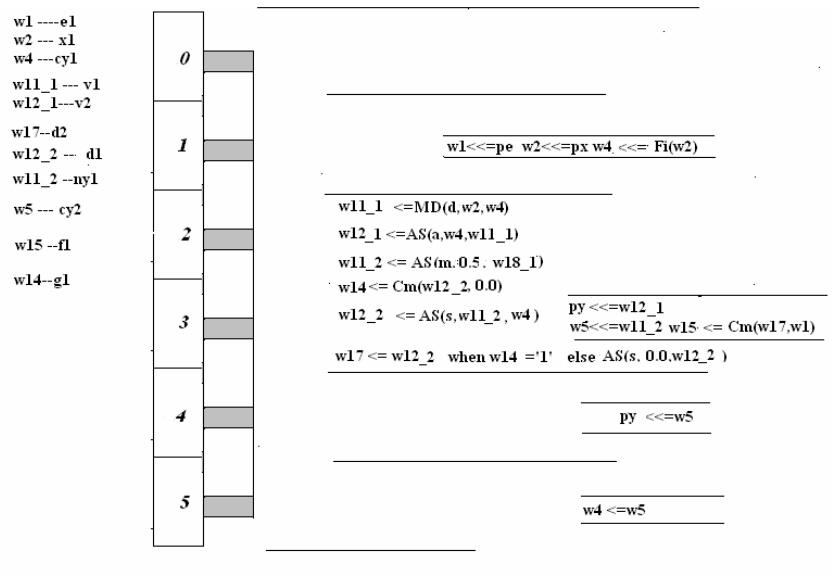


Figure 7

The transformation of the description in accordance with Figure 6, based on the recognition that states 2 and 3 are contractible. It is to conceive that if the objects of the description are corresponded with those of the specification, as seen on the left-hand side of the figure, then a canonical description identical to the canonical form of the specification is obtained.

Conclusions

The new canonical form detailed above seems to be capable of developing an algorithm and an automatic verification system. The work intended to elaborate 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