Special Design Tools on FIFO-based VLSI Target Architectures and their Application

Péter Keresztes
Széchenyi István University
Egyetem tér 1.Győr,
Hungary
keresztp@sze.hu

Timót Hídvégi
Széchenyi István University
Egyetem tér 1. Győr,
Hungary
hidvegi@sze.hu

Gheorghe Sebestyen
Technical University of Cluj
str. G. Baritiu nr. 26-28 Cluj,
Romania
gheorghe.sebestyen@ cs.utcluj.ro

Abstract

This paper proposes a new target architecture, mainly for the implementation of digital signal processing algorithms, which consists of multi-direction FIFO memories as main components, and a design-environment, which makes a semi-switch level logic simulation of extracted schematic descriptions possible. This target architecture can be reached through a special value-trace method, which was proposed by the author in an earlier paper. The efficiency of the FIFO-based architectures depends on the simplicity of the basic cell of FIFO. A very simple scheme for the FIFO cell utilises the charge-storage effect of an input of the CMOS gates, and it is composed by CMOS transmission gates and inverters. If the final verification of the FIFO-based VLSI circuits is a logical simulation of the schematic description previously proved by an LVS (Layout Versus Schematic) program, special multi-level logical models are necessary. These models have to be such as to enable their simulation to detect the typical design errors. A new VHDL package described in the paper consists of a new multi-value model for the type 'bit', and semi-switch level component models. The package was tested in a design process of a digital CNN array processor chip.

I. FIFO-BASED STRUCTURES, AS I/O-CONSTRAINED TARGET ARCHITECTURES FOR VALUE-TRACE BASED HIGH LEVEL VLSI SYNTHESIS

The concept of value trace was introduced by Snow and McFarland [1] [2] , and it is used in one of HLS systems [3], in the CMU's System Architect Workbench (SAW), which is one of the first HLS systems. In SAW, the input is an ISPS description, and the value trace (VT) is defined as a combined control- and data-flow graph. VT-graph is an intermediate representation transformed from the ISPS behavioural description, and it is the source of scheduling and allocation. The value trace method itself means the transformation from ISPS behavioural description into VT-graph. The method introduced by an earlier work of author is similar, but the target description is a pure, event driven data-flow description, given properly in a concurrent VHDL block [4]. Consider now a very simple example as illustration.

```
program EXAMPLE (a, b, c, d : in;
                 x, y : out );
variable u, v ;
 begin
  read (a, b, c, d);
   u := a + b;
   u := 2 * u;
   v := b * d;
   v := b - v;
   x := u * c;
   y := v * d;
  write(x, y);
end
```

```
VT_DFB_OF_EXAMPLE : block
 begin
   u1 <= a0 + b0;
   u2 <= 2 * b0;
   v1 <= b0 * d0;
   v2 <= b0 - v1;
   x1 <= u2 * c0;
   y1 <= v2 * d0;
end block;
```

Program 1. A simple algorithm description and its VALUE-TRACE DATA-FLOW BLOCK

Naturally, there are many tasks, in which to obtain a pure data-flow block is impossible. At the same time, many digital signal assignment algorithms consisting of iteration steps of the same sequence of computations, and can be described with *for* or *while* type loops, are easy to transform into pure VT-DFB descriptions.

Target architectures play a fundamental role in the high-level and system level synthesis of digital systems. A target architecture defines a hardware-structure in terms of particular units, their parameters and the interconnections between them. A target architecture can be considered to be an optimum for a given behavioural description and a given set of constraints. The selected target architecture depends on the abstract level behavioural description and the constraints. At the same time, the target architecture determines the algorithm of synthesis procedure. Consider the implementations of the EXAMPLE with two kinds of value-trace based target architectures applied for implementation of EXAMPLE given in Fig.1. In the first implementation, the number of function units is taken into account as primary constraint, while the recovery time in the second one. The disadvantage of the second architecture is the great number of function units, but the short time between two tasks (recovery-time) is a significant advantage.

A third kind of value trace based target architecture is derived from the primary constraint, which is given for the number of I/O ports. In this case, the VT-DFB is decomposed into generations. Each generation of value-signals is represented by a processor-stage. A stage contains an *INPUT-FIFO* (*IF*), a *VALUE-FIFO* (*VF*), an *OPERAND-FIFO* (*OF*) and a controlled function unit (*CFU*). A more complex solution is needed, if the generations of the value signals are decomposed into sub-blocks because of the necessity of further decreasing of the input-output buses. The consequence is the extension of *VALUE-FIFO* with a new, depth dimension.

Figure 2. shows the simplest version of this architecture for the implementation of EXAMPLE. The input data (*a0, b0, c0, d0*) enter into the IF of the first stage. When the IF is full, the data are transferred into the OF. Simultaneously and synchronously with filling of the results of the first operations (*u1, v1*) from the output of CFU, the input data of the first task are loaded into the IF of the second stage, and input data of a new (second) task are loaded into the IF of the first stage. Figure 2. shows the stages of the third type VT target architecture of EXAMPLE in successive phases in operation. It is shown that at the output of the third *CFU*, the values of the third-generation (*x1, y1*) are obtained as output data. The most applied building component of the I/O constrained VT architectures is the multi-direction FIFO.

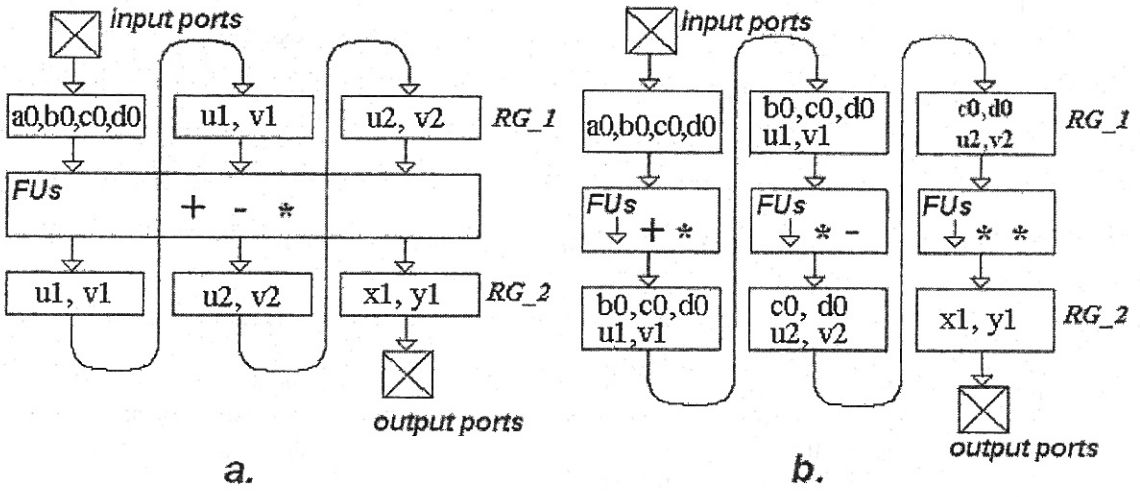

Figure 1. Two implementations of EXAMPLE. A FU constrained target architecture (a), and a recovery-time constrained target architecture (b).

III.A SIMPLE CHARGE-STORAGE FIFO CELL

Figure 4. shows a bit of the basic cell of a multi-direction FIFO. Inserting this stage into an array makes it possible to shift data vertically from two sources (*D1, D2*) and horizontally from source *D3*. The operation of this type of FIFO cells is based on the charge storage of capacitor *C*, which represents the input capacitance of the first inverter. The control of a shifting-step consists of a cycle of non-overlapping pairs of impulses. Between the shift and the inverted-shift, and between the hold and the inverted hold pairs, the sampled logical level is stored by capacitor *C*. The basic problems of a logical-level simulation model are as follows:

- The behavioural model of the transmission gate has to mirror the behaviour of the interconnected outputs. The solution of this problem is the introduction of a multi-state, new bit-type with a strength-controlled resolution function.
- The behavioural model has to mirror also the act of faulty, not completely controlled transmission gate. For example, if the p-MOSFET

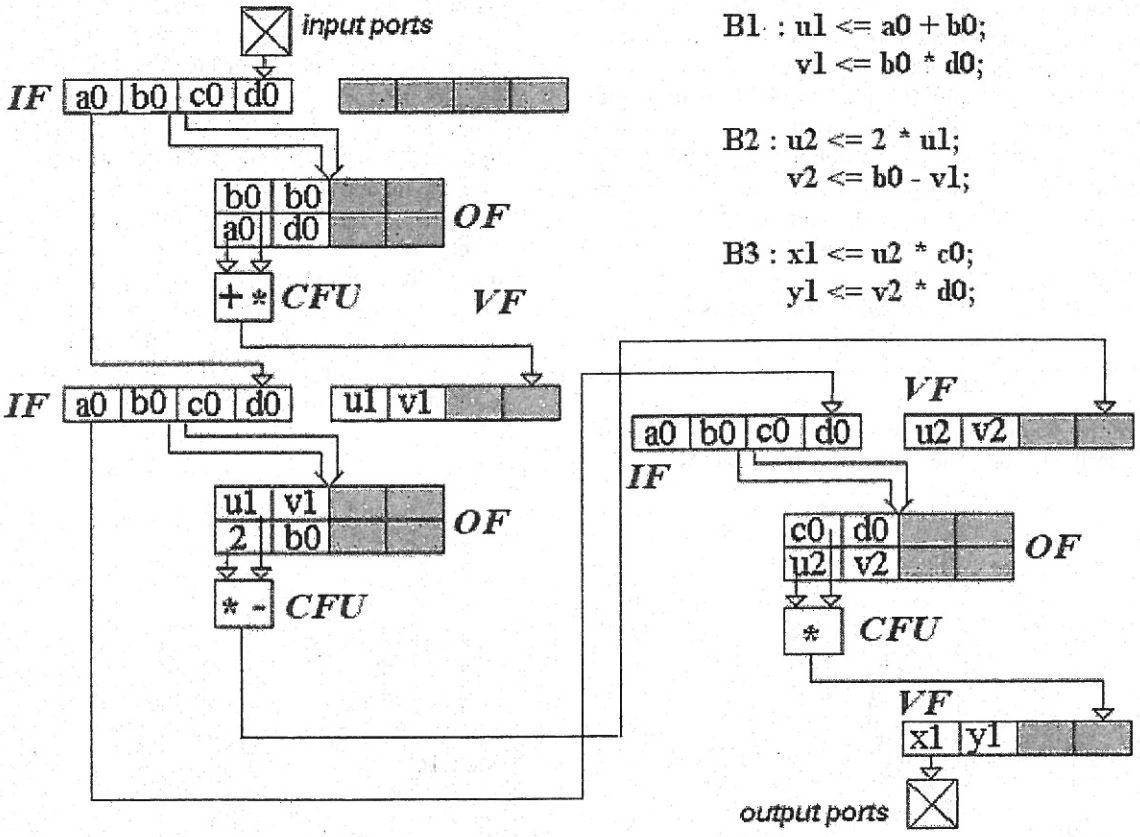

Figure 2. The implementation of EXAMPLE with an I/O constrained target architecture

is opened with a low level, but the high-level signal's connection to the gate-electrode of the n-MOSFET is missed, and the input to be transferred is low, the output has to be at a weak logical low level (*w0*) .

- Among the values of the type 'newbit' so called 'stored logical levels' (*s1, s0*) have to be inserted into a position of sufficient strength.
- The input gate electrodes of the logical gates have to act as capacitors with a given current discharging parasitic conductance.

II A VHDL PACKAGE FOR MODELLING CHARGE-STORAGE FIFO CELLS

The logical simulation of charge-storage FIFO cells does not require bidirectional switch level VHDL models, one-directional switching models are

satisfactory. At the same time, the modelling of charge-storage effect is necessary.
The logical levels of the type '*newbit*' ordered by the binary relation 'dominates on' are shown in Figure 3. Level *A* dominating *B* means that a driver which is forcing *A* on the node, on which another driver is forcing *B*, *A* is the stronger driver. Logical level '*U*' represents a contradictorily undefined level. Logical levels '*0*' and '*1*' correspond to the classical logical levels, and with the exception of '*U*', they are the strongest levels. The weaker levels '*w0*' and '*w1*' represent such logical levels, whose values of voltage have changed toward the voltage of the complementary level, but the so called comparison level is not exceeded. (See the case of a partly-opened transfer-gate above). With the weak logical levels also the low-current drivers can be described. In this way, '*quasi-n*' and '*quasi-p*' CMOS gates, and their *precharge-evalution* variants can be modelled. Levels '*s0*' and '*s1*' represent the stored values.
A node modelled with the stored levels is an input of a logical gate. The concurrent signal assignment statement for target '*INPUT*' holds a stored level below the corresponding strong or weak level, but when the drivers goes into a stored level, the node '*INPUT*' goes in 'Z' after a delay-time, which represents a storage time in this case. The weakest logical level '*Z*' represents the well known floating state.

```
INPUT <=   s1 when (INPUT = '1' or  INPUT = w1) else
           s0 when (INPUT = '0' or  INPUT = w0) else
           Z after 100 ns when INPUT = s1 or INPUT = s0  else
           Z;
```

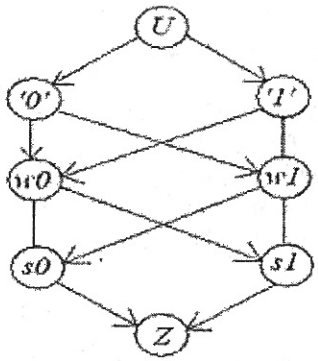

Figure 3. The levels of 'newbit' ordered by the relation 'dominates on'

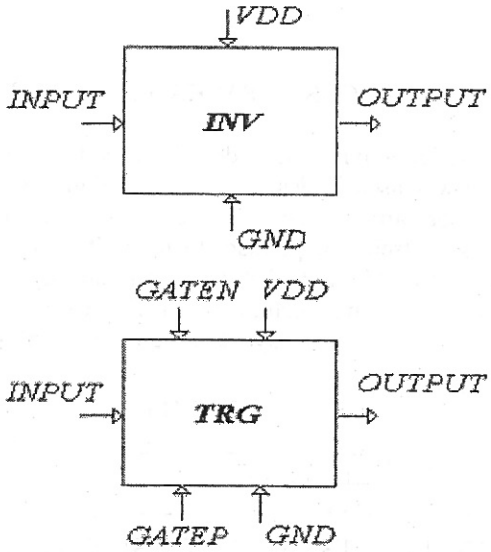

Figure 5. Basic logical building elements of FIFOs

The VHDL description of the package is given in the APPENDIX.

IV. VHDL MODELLING OF FIFO-CELL

The package '*newnstd*' and the component models, which are based on it, are capable of detecting the most common control and instance errors in charge-storage type FIFO arrays. Not only the typical control errors listed in point 2 can be observed but also the unbounded power-supply nodes in the simulation. The latter is made to be possible by the fact that the power-supply nodes of the components are considered as logical inputs.
(See Figure 5.)

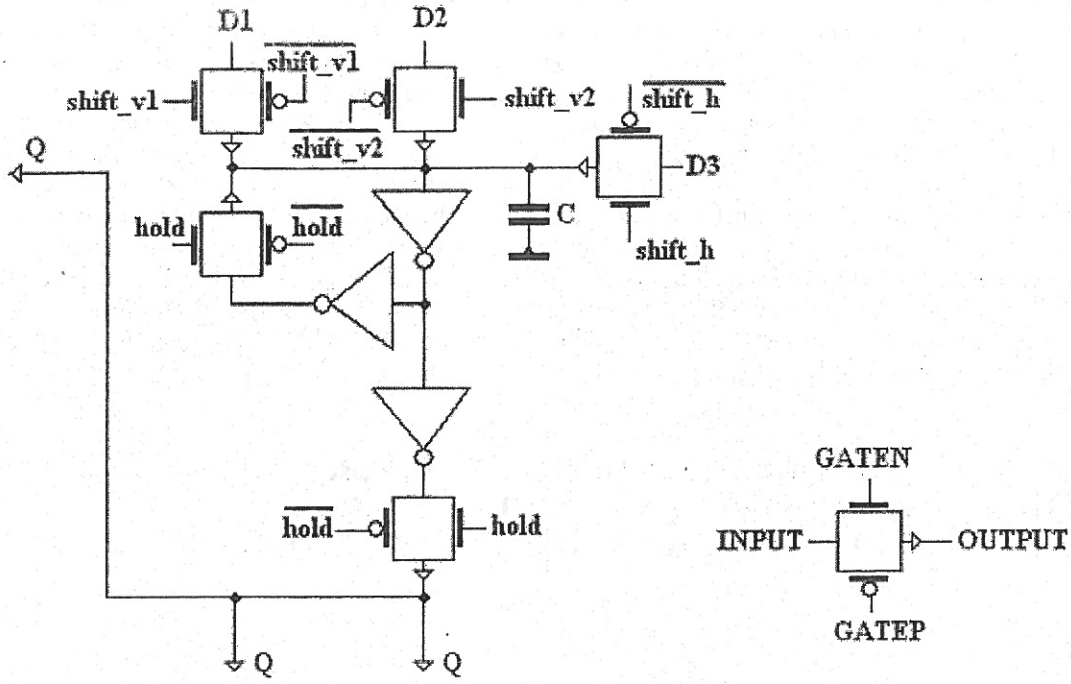

Figure 4. Scheme of the charge-storage FIFO cell

V. APPLICATIONS OF PACKAGE 'NEWNSTD'

The design environment described in the above points was used in designing a 3 x 2 digital CNN processor array. The architecture of a single processor itself was published earlier. It is easy to recognize in Figure 6. that the most frequently used parts of the architecture are FIFO units. The schematic entry and layout design parts of the CADENCE–OPUS system were applied in chip design. After the proving of the equivalence between the extracted schematic and the layout descriptions, a final VHDL simulation was executed to detect the design errors. The package and the models described above and given below in the APPENDIX proved to be suitable.

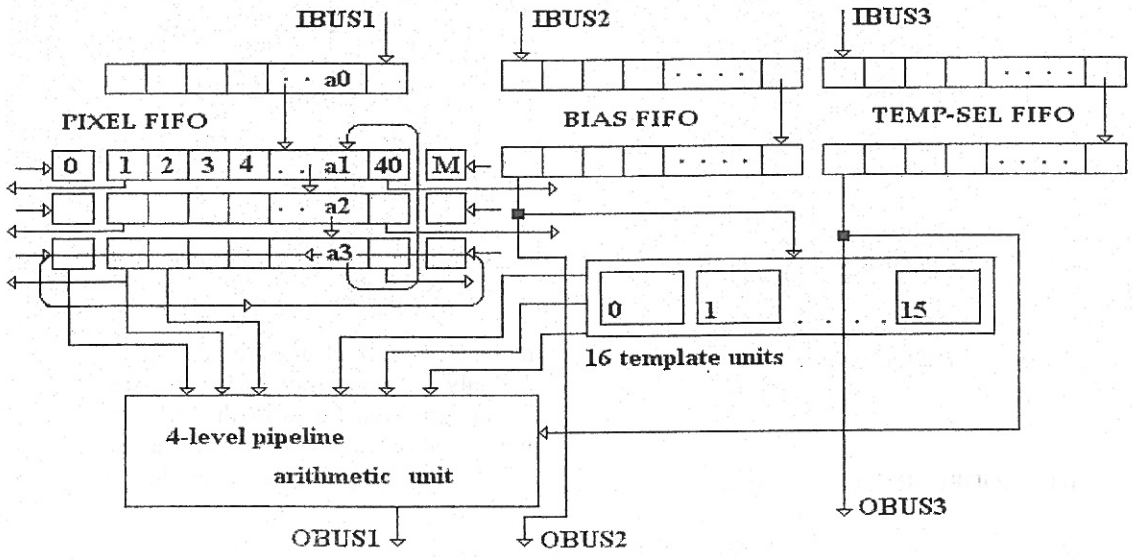

Figure 6. An emulated digital CNN processor unit

VI. REFERENCES

1.P. Keresztes
Value Trace VHDL Blocks and Their Application in High Level Synthesis
Proc. of 2nd Workshop on Libraries, Component Modeling and Quality Assurance , Toledo, Spain, April 1997. pp. 225-232

2. K. Kordoc, M. Biotteau, E. Cerny
Switch-Level Models in Multi-Level VHDL Simulations
Proc. of the First European Conference on VHDL, Marseilles, Sept.1990.

3. A.G. Stanculescu, A.S. Tsay, A.N.D. Zamfirescu, D.L. Perry
Switch Level VHDL Descriptions, ICCAD89.

4 A.Zarándy, P. Keresztes, T.Roska, P.Szolgay,
An emulated digital architecture implementing the CNN Universal Machine
Proc of IEEE CNNA'98, London, pp. 249-252, 1998.

APPENDIX

```
package newnstd is
type nnewbit is ( Z, '0', '1', w0, w1, s0, s1, U);

-- '0', '1' : strongest classical logical levels;
-- s0, s1 : stored logical levels;
-- w0, w1 : weak logical levels;
-- Z : floating (third) state;
-- U : undefined state; ( The signal having at U, is not Z , but
uknown )

type nnewbit_vector is array (natural range<>) of nnewbit;
function resolved (srcs : nnewbit_vector) return nnewbit;
subtype newbit is resolved nnewbit;
type newbit_vector is array (natural range<>) of newbit;
subtype oldbit is newbit range Z to '1';
type oldbit_vector is array (natural range<>) of oldbit;
end newnstd;

package body newnstd is
function resolved ( srcs : nnewbit_vector) return nnewbit is
   variable num0, num1, s0num , s1num, w0num, w1num,
znum, unum : natural := 0;
   variable v : newbit := Z;
    begin
    for i in srcs'range loop
     if srcs(i) =  '0' then num0 := num0 + 1;
     elsif srcs(i) = '1' then num1 := num1 + 1;
     elsif srcs(i) = s0 then s0num := s0num + 1;
     elsif srcs(i) = s1 then s1num := s1num + 1;
     elsif srcs(i) = w0 then w0num := w0num +1;
     elsif srcs(i) = w1 then w1num := w1num +1;
     elsif srcs(i) = Z then znum := znum + 1;
     else unum := unum + 1;
      end if;
    end loop;
 if unum > 0 then v := U; elsif
   unum = 0 and num0 = 0 and num1 > 0  then
 v := '1'; elsif
   unum = 0 and  num1 = 0 and num0 > 0  then
 v := '0'; elsif
   unum = 0 and num1 > 0 and  num0 > 0  then
 v := U; elsif
 unum = 0 and num1 = 0 and num0 = 0  and  w0num > 0 and
            w1num = 0 then v := w0; elsif
   unum = 0 and num1 = 0 and num0 = 0 and w0num = 0 and
             w1num > 0 then v := w1; elsif
  unum = 0 and num1 = 0 and num0 = 0 and w0num = 0 and
w1num = 0 and s1num = 0
            and s0num > 0 then v := s0; elsif
  unum = 0 and num1 = 0 and num0 = 0 and w0num = 0 and
w1num = 0 and s1num > 0
             and s0num = 0 then v := s1; elsif
   unum = 0 and znum = srcs'length then v := Z;
else v:= Z;
    end if;
    return v;
end resolved;

library work; use work.newnstd.all;
 entity INV  is
 port  ( INPUT :  inout newbit;
         OUTPUT : inout newbit;
         VDD : in oldbit;
          VSS : in oldbit);
 end;
library work; use work.newnstd.all;
entity TRG  is
 port
   ( INPUT : in newbit;
     OUTPUT :inout newbit;
     GATEP : in oldbit;
     GATEN : in oldbit;
     VSS : in oldbit;
     VDD: in oldbit);
end;

library work; library work; use work.newnstd.all;
entity FIFO_CELL is
   port( VDD, VSS : in oldbit;
      D1, D2, D3 : inout newbit;
      shift_v1, shift_v2, shift_h, hold,
      nshift_v1, nshift_v2, nshift_h, nhold : in
      oldbit;
      Q : inout newbit);
end FIFO_cell;

architecture BEH of INV is
begin
  INPUT <=  s1 when (INPUT = '1' or
                        INPUT = w1) else
         s0 when (INPUT = '0' or INPUT = w0) else
         Z after 100 ns when INPUT = s1 or
                        INPUT = s0 else
         Z;

  OUTPUT <= '0' after 100 ps when ((INPUT = '1' or INPUT
= w1 or input = s1) and VDD = '1' and VSS = '0')  else
     '1' after 100 ps when ((INPUT = '0' or INPUT = w0 or
INPUT = s0) and VDD = '1'
                        and VSS = '0') else
          U;

end BEH;

architecture BEH  of TRG is
begin
```

```
 OUTPUT <= U  when (VDD = '0' or
    VDD = Z or VSS = '1' or VSS = Z)
                                    else
    INPUT after 100 ps when (INPUT /= s1 and
                                INPUT /= s0 and
    GATEN = '1' and GATEP = '0')
                                    else
     Z  after 100 ps when  ((INPUT = s1 or
                               INPUT = s0) and
     GATEN = '1' and GATEP = '0')
                                    else
    '1' after 100 ps when  (INPUT = '1' and
     GATEN = '0' and GATEP = '0')
                                    else
    '0' after 100 ps when  (INPUT = '0' and
     GATEN = '1' and GATEP = '1')
                               OUTPUT = s0) and
     GATEN = '0'and GATEP = '1')
                                    else
     U after 100 ps when GATEN = Z or
                               GATEP = Z
                                    else

     Z;
end BEH;

architecture STRUCT of FIFO_CELL is
component TRG
 port
   ( INPUT : in newbit;
     OUTPUT :inout newbit;
     GATEP : in  oldbit;
     GATEN : in  oldbit;
     VSS : in oldbit := '0';
     VDD: in oldbit := '1');
end component;

component INV
 port
                                    else
    w1 after 100 ps when (INPUT = '1' and
     GATEN = '1' and GATEP = '1')
                                    else
    w0 after 100 ps when  (INPUT = '0' and
      GATEN = '0' and GATEP = '0')
                                    else
     s1 after 100 ps when ((OUTPUT = '1' or
                                OUTPUT = w1) and
       GATEN = '0' and GATEP = '1')
                                    else
       s0 after 100 ps when ((OUTPUT = '0' or
                                OUTPUT = w0) and
    GATEN = '0' and GATEP = '1')
                                    else
     Z after 1 ns when   ((OUTPUT = s1 or

   ( INPUT :  inout newbit;
     OUTPUT : inout newbit;
     VDD : in oldbit;
     VSS : in oldbit);
end component;

signal SIG1, SIG2, SIG3, SIG4 : newbit;
begin
U1 : TRG port map (D1, SIG1, nshift_v1,
                   shift_v1, VSS, VDD);
U2 : TRG port map (D2, SIG1, nshift_v2,
                    shift_v2, VSS, VDD);
U3 : TRG port map (D3, SIG1, nshift_h, shift_h,
                   VSS, VDD);
U4 : TRG port map (SIG3, SIG1, nhold, hold,
                  VSS, VDD);
U5 : TRG port map (SIG2, SIG4, nhold, hold,
                   VSS, VDD);
U6 : INV port map (SIG1, SIG2, VDD, VSS);
U7 : INV port map (SIG2, SIG3, VDD, VSS);
U8 : INV port map (SIG4, Q, VDD, VSS);
end STRUCT;
```