Dual-Rail Asynchronous Implementation of a VLSI Processor for Digital Cellular Neural Networks

Péter Keresztes, Timót Hídvégi

Department of Automation, Széchenyi István University
H-9026 Győr, Egyetem tér 1, Hungary
Email: keresztp@sze.hu, hidvegi@sze.hu

Abstract: An emulated digital Cellular Neural Network (CNN) Universal Machine chip was designed by a research group of Analogic and Neural Computing Systems Laboratory of HAS, almost 10 years ago. During the design period of the chip, particular attention had to be paid to the clocking and control system of the chip, since each architecture element, placed on a large silicon area chip (1 cm^2), operated in a totally synchonous mode, using a single global clock. The designer's manipulations for eliminating the consequences of the too high propagation delay of long interconnections resulted in a relatively large chip, and a significant part of the chip was totally superfluous from the point of view of logical operation. The clock rate was limited at a lower level than it would have been possible without the long interconnections. So the development of 'CASTLE' CNN processor array showed the most significant problems of the submicron VLSI, which arose from the too long interconnections of big size chips.

The elimination of clocking problems and the re-designing of the CASTLE architecture using delay-insensitive, self-synchronized, asynchronous logic elements were performed. The result of the re-designing work, which is presented in this paper, is a clockless, totally asynchronous architecture. The combination of the *DUAL-RAIL logic* and the methods of the well known 4-*phase asynchronous inter-register communication* were chosen. Obviously the timing and control unit of the synchronous version is unnecessary in the asynchronous one. So the tasks of re-designing several synchronous units to their dual-rail versions consisted in designing an application specific dual-rail arithmetic unit with feedback and dual-rail multidirection FIFOs. Several new dual rail logic elements were introduced, which were modeled and simulated as follows:

- Two-input dual-rail register with a common acknowledge output and priority order for inputs
- Two- or more-input dual rail register with independent acknowledge outputs and dual-rail selection inputs
- Dual-rail register with CLEAR single-rail control inputs, which enable setting the register into so called DATA-TOKEN and BUBBLE states.

The lecture presents the most interesting parts of the design process.

Keywords: emulated digital CNN-UM, clockless processors, Dual-rail logic, self-synchronization, delay-insensitive logic systems

1. Introduction: Experiments of designing a synchronous CNN processor array

The architecture of a VLSI chip containing a sub-array of digital CNN processors was published in 1999. [2], [11]. The chip operated with a two-phase global clock, so it demanded a very careful design of the central timing and control unit (TAC), which were based on a synchronous FSM. The cause of the difficulties is that the time delay of the relatively long metal interconnections exceeded the delay time of the logic gates in the applied submicron CMOS technology. A relatively large silicon area had to be sacrificed for the synchronity of the clock and the control signals in the different points of the chip.

The problem of clocking, which had to be solved 10 years ago, became the primary obstacle to the future development of VLSI technology. There are several concepts and methods intended to solve the clocking problem, as follows:

- The system consists of *smaller synchronous units*, which communicate *asynchronously* [3], [5],[6]
- Application of *Delay Locked Loop* (DLL) circuits to insert the required value additional delay into the paths of the clock signal [10].
- Application of *asynchronous units* which communicate *asyncronously* [1],[4], [7], [8], [9].

The re-designing of the former CNN processor architecture 'CASTLE' will be discussed in detail in this paper.

2. Dual-Rail asynchronous digital circuits and systems

The dual-rail asynchronous circuits and systems are based on two principles, as follows:

- The principle of 4-phase hand-shake communication.

The background of the classical four-phase handshaking asynchronous communication is that the data on the output of the transmitter (sender) has to be ready before the signal '*request*' rises. The idea is that the code of the output datum itself contains the *request*. The dual-rail codes can contain it. The application of this principle leads to the asynchronous inter-register communication of the dual-rail sytems.

- The principle of logic completeness.

It is supported by the dual-rail code of logical variables. Logic completeness means that a function unit should only give a valid output datum if it has valid data on all of its inputs, and should only switch its output to invalid, if all its inputs have turned invalid. It follows from this that a logic decision is valid and transmissible only, if all the premises necessary for it are valid. This enables

hazard-free logic implementations. A network consisting of such units bears the ability of self-synchronization, that is the data validated by specific, valid input data in a given time will not mix with those validated by the input data of a different time.

If two wires are ordered to one logic variable, it is possible to distinguish valid and invalid logic values. The value of the variable is valid, if the levels of wires are (L H) or (H L). The value of the variable is invalid, if both wires are at low level (L L). The convention is that a valid datum contains a request.

Dual-rail (DR) registers can be easily built up from the C-elements, which are introduced by Müller at the end of 50'ths. The simplest Müller element, the C-2 is shown in Figure 1. The DR-LATCH consisting of C-2 elements and the symbol of a DR-register are shown in the Figure 2.

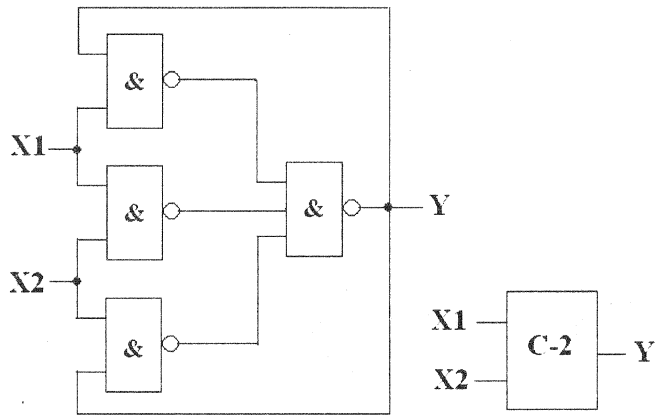

Figure 1. The scheme of Muller C-2 and its symbol

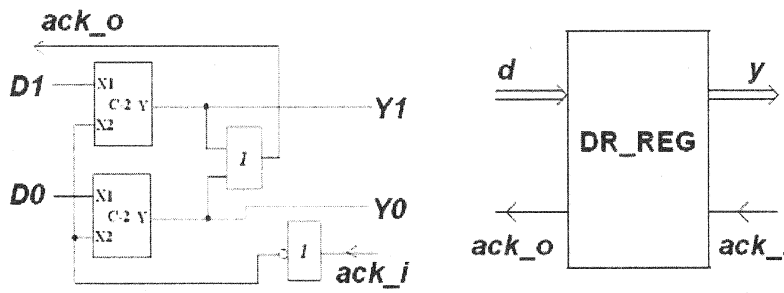

Figure 2. Scheme of a Dual-Rail latch and the symbol of Dual-Rail register

3. R-T level architecture of the synchronous CASTLE processor

The name of the architecture of the core processor of the fully synchronous array was a made-up name 'CASTLE'.

The operation of CASTLE was derived from the "full-signal-range model" of the CNN state-equations using the simplest forward Euler integration form:

$$x_{ij}(n+1) = \sum_{C(kl)\in N_r(i,j)} A'_{ij,kl} * x_{kl}(n) + g_{ij}$$

$$g_{ij} = \sum_{C(kl)\in N_r(i,j)} B'_{ij,kl} * u_{kl}(n) + h * z_{ij}$$

Here h is the time step, u_{kl} are the inputs, $x_{ij}(n+1)$ is the next state of the cell(i,j) before truncation, $x_{kl}(n)$ are the current states of the neighbouring cells, g_{ij} and z_{ij} are constants, supposing a constant input, and A' and B' are the so called modified template matrices. The R-T level architecture is shown in Figure 3.

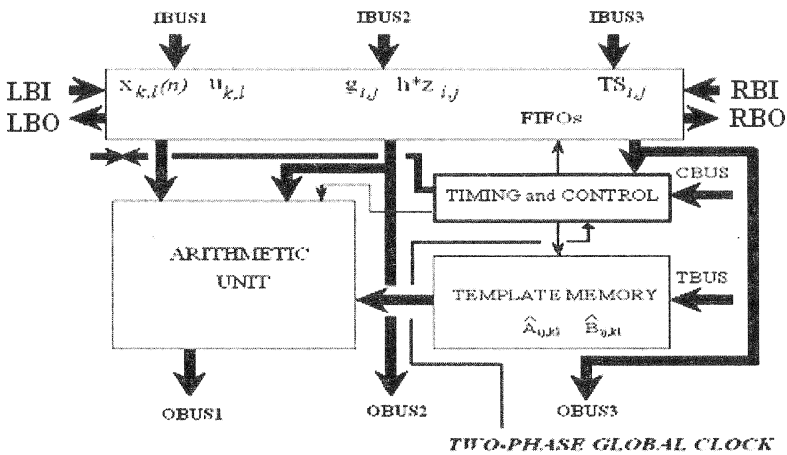

Figure 3. The original, global clock CASTLE architecture

The sub-unit TIMING and CONTROL controls not only one processor, but all the processors on the chip. This is the origin of most synchronization difficulties.

In Figure 4. it can be seen that the state variables (**IBUS1**), additive constants (**IBUS2**), and the bit-vectors of template selection (**IBUS3**) are continuously shifting in FIFO memories. The buses **LBI, LBO**, **RBI** and **RBO** are used for the communication with the left-side and right-side neighbouring processors. The FIFOs of state variables are particularly complicated circuits. These are so-called multi-direction FIFOs with horizontal and vertical shifting, combined with both horizontal and vertical rotation.

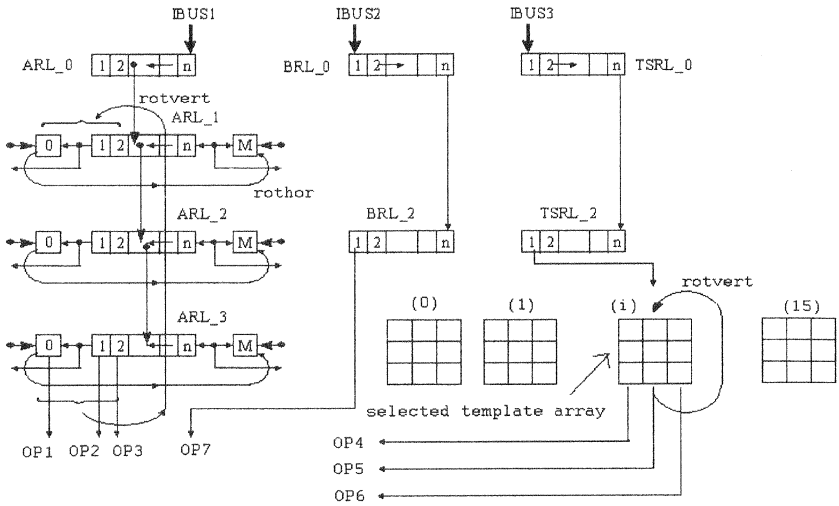

Figure 4. Traditional FSM controlled multidirection FIFOs of CASTLE

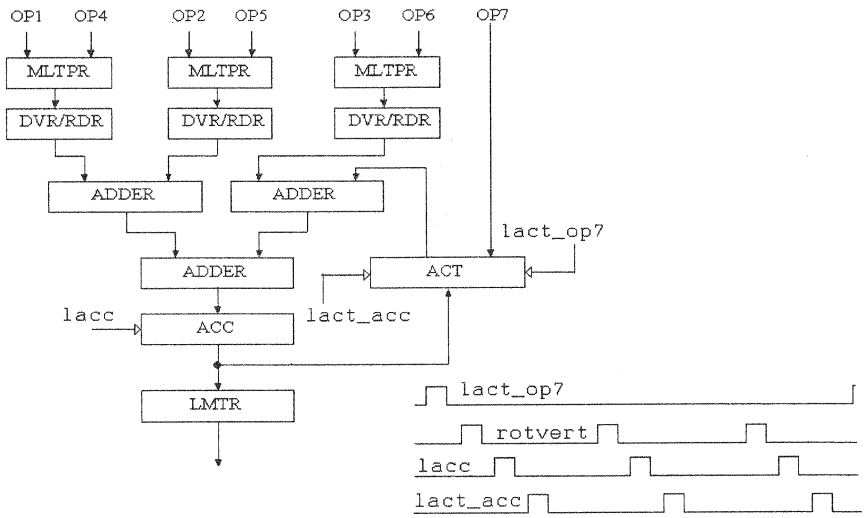

Figure 5. The FSM controlled arithmetic unit of the CASTLE processor

The left-side 9 cells with indexes 0, 1, 2 of the three state-FIFOs take part in the next-state calculation, along with the left-side cell of the constant-FIFO and the template values selected by the left-side cell of template-select FIFO. Figure 5. shows the three-multiplier arithmetic unit, for the inputs of which the state-FIFO rotate the state- and template-operands. There is a feedback via an ACCUMULATOR and a TEMPORARY

ACCUMULATOR. The latter receives the constant to be added to the sum-of- products. The control signals of the register-tranfers are marked in the figure, and a timing diagram shows the three-cycle rotation (*rotvert, lacc, lact_acc*), preceded by the loading of an additive constant (*lact_op7*).

4. Architecture elements of a Dual-Rail asynchronous CNN processor

A DR asynchronous R-T level system can be considered as a composition of DR-stages. A DR-stage consists of DR-registers, which communicate with the registers of other stages in the way detailed above, and DR function units, the operation of which fulfils the criteria of logic completeness.

Figure 6. demonstrates how the R-T level DR units are connected to each other, using DR buses and acknowledge signals.

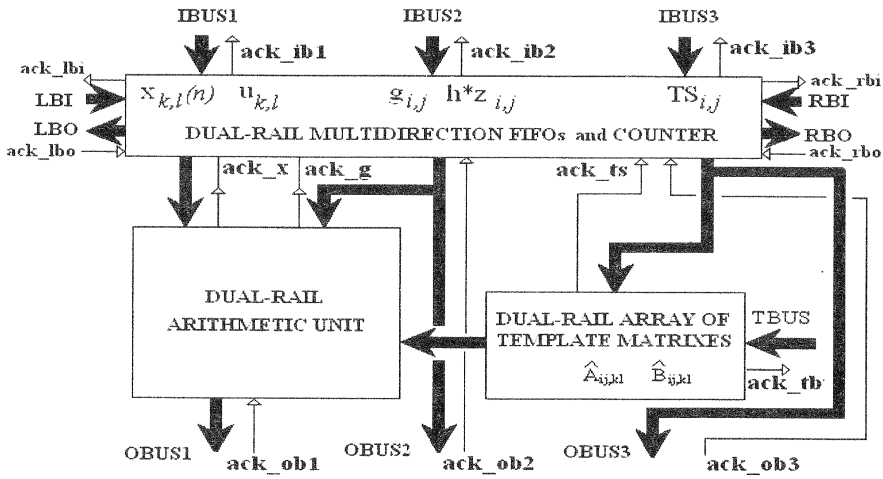

Figure 6. The architecture of DR-CASTLE processor

The two most interesting DR-stages are presented in this paper. They are interesting from the point of view that several new solutions had to be invented, which had not existed until then in the literature. These are as follows:

- DR-FIFO compositions for multidirection shifting and rotations
- CNN arithmetic unit consisting of DR multipliers and adders and DR registers

4.1. Dual-Rail, multidirection FIFOs for asynchronous CNN processor

Sparso established the scheme of how to build up a shift-register from DR-latches. [9]. He called the state of a latch BUBBLE, when its output is invalid, and output *ack_out* and input *ack_in* are low. In the BUBBLE state the latch can be immediately loaded from its valid input. The opposite state is called TOKEN. There are two variants of the

TOKEN state. The fisrt is called DATA-TOKEN, in which the *Y* output is valid, *ack_out* is high, and *ack_in* low. The second one is called EMPTY-TOKEN, when the *Y* output is invalid, *ack_in* are high, and *ack_out* is low. The different state registers and a part of a DR-shift-register is shown in Figure 7.

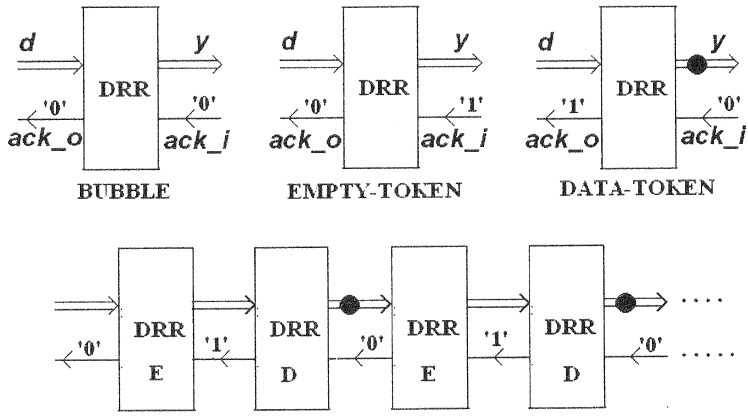

Figure 7. The states of DR-registers and the initial state of a register-chain

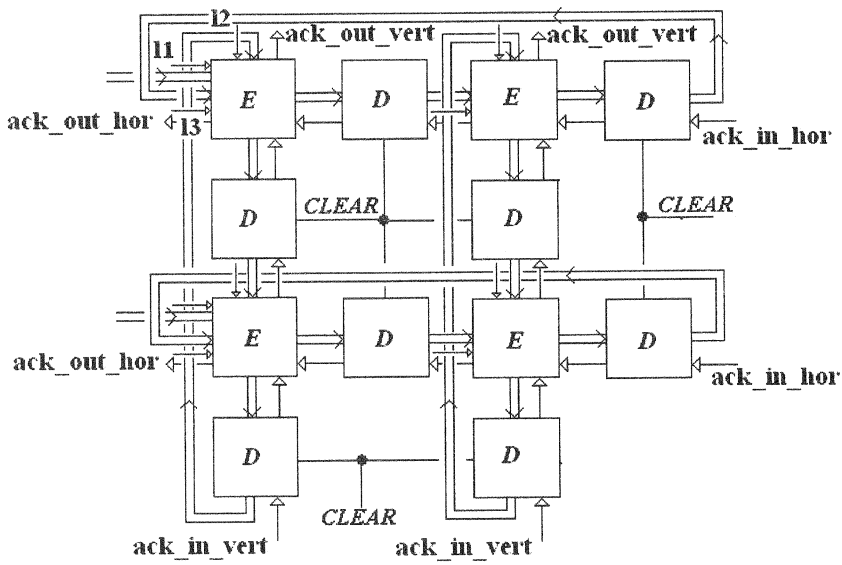

Figure 8. A simplified model of multidirection DR-FIFO

A chain of DR-LATCHES can be considered an *n*-stage shift-register, if

- there are 2n latches chained,
- the initial state of the chain has to be special, namely an EMPTY-TOKEN latch has to be followed by a DATA-TOKEN one.

Starting the DR-SR from this state with a valid input and rising the last *ack_in*, BUBBLE state runs along the chain from the last latch to the first one. So all the data will be shifted and the state of each latch will change.

4.1.1. DR-register with clear

The DR-register equipped with CLEAR input is a necessary component of DR-FIFO. The CLEAR is a single-rail control signal, H-level of which results in valid intial data on the output. This initial data in our case is the valid code of the integer 0. If signal CLEAR is raised, the initial data is stored, and the register will be in state DATA-TOKEN. It can be seen in Figure 8 that each second register is an instance of this component. The symbol of DR-register with CLEAR is shown in Figure 9., and in Figure 10. the scheme of its element is presented. The VHDL behavioural description of this unit is given in the Appendix.

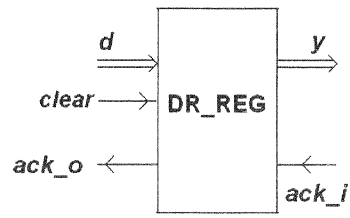

Figure 9. Symbol of DR-register with clear.

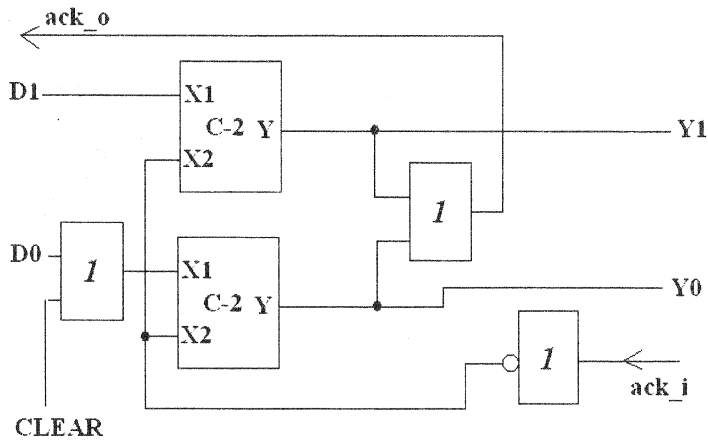

Figure 10. Scheme of DR-LATCH with clear

4.1.2. Multi-input DR-register with DR selectors (Figure 11.)

Multiple-input DR-registers with selectors are also needed for the multidirection FIFOs. A selector-wire, which is a 'single-rail' belongs to each data-input. The condition of

storing the value of a given data-input is that the selector value belonging to the data-input is 'H'. It means that the 'H' is the valid-value on a single-rail. The behavioural description can be seen in the Appendix.

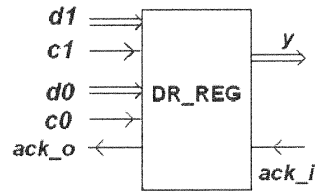

Figure 11. Scheme of two-input DR-REG with single-rail selector wires

4.1.3. Multi-input DR-register with priority order for inputs (Figure 12.)

In the case of the third type of a multiple-input register a priority order is defined for the inputs. For the two-input DR-register given in Figure 12. input **d1** dominates the input **d2**. The VHDL behavioural description of this unit is given in the Appendix.

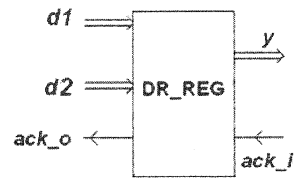

Figure 12. Scheme of two-input DR-REG with priority order for the inputs

4.2. Dual-Rail arithemetic unit with feedback

The arithmetic unit is a classical DR stage, consisiting of an arithmetic core and registers. The core is a compostion of DR multipliers and adders, and these components are dual-rail combinational networks. The input DR-registers make the asynchronous communication with the outputs of the multidirection FIFOs. The chain of three DR registers constitutes the feedback for accumulating the sum of subproducts. Sparso showed that for a DR-ring without a dead-lock the presence of at least three latches is needed. The third register of the loop receives not only the accumulated sum of the subproducts, but in the initial step of each cycle the additive constant as well. This register has two data-input with a common acknowledge signal. One of the data-input is connected to FIFO cell which stores the additive constant. Since a valid code of this input starts a new calculation cycle, it has a priority over the other data-input, which is the closing point of the loop.

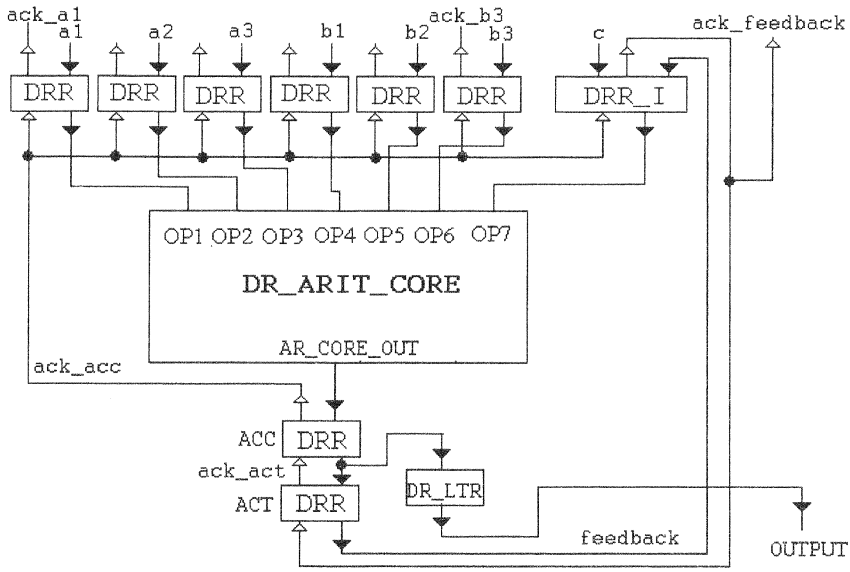

Figure 11. The scheme of the Dual-Rail arithmetic unit

5. Conclusions

The original synchronous architecture of CASTLE can easily be transformed into a Dual-Rail asynchonous version. Above the well known elements, the elaboration of several new Dual-Rail logic elements is necessary. They are as follows:

- Dual-rail register with CLEAR single-rail control input, which enables setting the register into so called DATA-TOKEN state.
- Two- or more-input dual-rail register with sigle-rail selection inputs
- Two input dual-rail register with priority order for inputs

Using these new RT elements, fully asynchronous CNN processor arrays can be realized, and the results can be applied in hardware-implementation of other types of neural networks too. Designing Dual-Rail asynchronous processors for digital VLSI implementations of neural networks helps avoid the clocking problem of submicron VLSI technology.

References

[1] Fant, K. M, Brandt, S. A.: *Null Convention Logic: A complete and consistent logic for asynchronous digital circuit Synthesis,* International Conference of Application Specific Sytems, Architectures and Processor, (1996) pp. 261-273

[2] Keresztes, P., Zarándy, Á., Roska, T., Szolgay, P., Bezák, T., Hídvégi, T., Jónás, P. and Katona, A.: *An emulated digital CNN implementation*, Journal of VLSI Signal Processing Volume 23, No 2/3, (1999) pp. 291-303

[3] Lavagno, L., Sangiovanni-Vincentelli, A.: *Algorithms for synthesis and testing of asynchronous circuits,* Kluwer Academic Publishers, (1993) pp.18-25

[4] Muller, D. E., Bartky, W. C.: *A theory of asynchronous circuits*, Annals of Computing Laboratory of Harward University, (1959) pp.204-243

[5] Nowick, S. M., Dill, D. L.: *Automatic synthesis of locally clocked asynchronous state machines*, Proc. of the International Conference on Computer Aided Design, November, (1991)

[6] Saphiro, D. M.: *Globally-Asynchronous, Locally Synchronous Systems*, PhD Thesis, Stanford University, October (1984)

[7] Smith, S. C., DeMara, R. F., Yuan, J. S., Ferguson, D. , Lamb, D.: *Optimization of Null convention self timed circuits*, INTEGRATION, the VLSI Journal 37 (2004) pp. 135-165

[8] Sutherland, I. E.: *Micropipelines*. Communications of the ACM, June, 1989. Turing Award Lecture

[9] Sparso, J., Furber, S.: *Principles of asynchronous circuit design – A system perspective,* Kluwer Academic Publisher, (2001) pp. 11-13

[10] Xilinx: *Using Delay-Locked Loops in Spartan-II/IIE FPGAs*, Application Note, http://www.xilinx.com/support/documentation/application_notes/xapp174.pdf

[11] Zarándy, Á, Keresztes, P., Roska, T., and Szolgay, P.: *An emulated digital architecture implementing the CNN Universal Machine*, Proc. of the fifth IEEE Int. Workshop on Celluler Neural Networks and Their, Applications, London, (1998) pp. 249-252

Appendix

```
package DR_PACK is

  type WIRE  is (L, H);
  subtype DR_REAL is real range -10.000000 to +9.999999;

end  DR-PACK;


-- behavioural description of DR-REGISTER with CLEAR

library work;
use work.DR_PACK.all;
entity DR_REG_cl is
 port ( clear : in bit;
        d : in DR_REAL;
        y : inout  DR_REAL;
        ack_i : in bit;
        ack_o : out bit);
end;

architecture BEH of DR_REG_cl is
constant td : time := 1 ns;
constant REALNULL : DR_REAL := -10.000;
begin

process(clear, d, ack_i, y)
 begin
  if clear = '1' then
           y <= 0.0 after td;
           ack_o <= '1' after td;

  elsif clear = '0' and Y = REALNULL and ack_i = '0' and
           d /= REALNULL then
           y <= D after td;
           ack_o <= '1' after td;

  elsif clear = '0' and Y /= REALNULL and ack_i = '1' and
            d = REALNULL then
            y <= REALNULL after td;
            ack_o <= '0' after td;
  end if;
 end process;
end BEH;
```

```
-- behavioural description of TWO-INPUT DR-REGISTER with
SELECTOR wires

library work;
use work.DR_PACK.all;
entity DR_REG_D1_D2 is
 port ( d1, d2 : in DR_REAL;
        y : inout  DR_REAL;
        ack_i : in bit;
        ack_o : out bit;
        c1, c2 : in WIRE);
end;

architecture BEH of DR_REG_D1_D2 is
constant td : time := 1 ns;
constant REALNULL : DR_REAL := -10.000;
begin
process(d1, d2, c1, c2, ack_i, y)
   begin
    if y = REALNULL and ack_i = '0' and d1 /= REALNULL and
       c1 = H then
             y <= d1 after td;
             ack_o <= '1' after td;

      elsif y = REALNULL and ack_i = '0' and d2 /= REALNULL
and
             c2 = H then
             y <= d2 after td;
             ack_o <= '1' after td;
      elsif y /= REALNULL and ack_i = '1' and d1 = REALNULL
and
             d2 = REALNULL and
             c1 = L and c2 = L then
             y <= REALNULL after td;
             ack_o <= '0' after td;
      end if;
  end process;
end BEH;


-- behavioural description of TWO-INPUT DR-REGISTER with
priority order  -- for the inputs

library work;
use work.DR_PACK.all;
entity DR_REG_DD is
 port ( d1 : in DR_REAL;
        d2 : in DR_REAL;
        y : inout  DR_REAL;
```

```
        ack_i : in bit;
        ack_o : out bit);
end;

architecture BEH of DR_REG_DD is
constant td : time := 2 ns;
constant REALNULL : DR_REAL := -10.000;
begin
process(d1, d2, ack_i, y)
   begin
    if y = REALNULL and ack_i = '0' and d1 /= REALNULL then
           y <= d1 after td;
           ack_o <= '1' after td;

      elsif y /= REALNULL and ack_i = '0' and d1 /=
REALNULL then
            y <= d1 after td;
            ack_o <= '1' after td;

      elsif y = REALNULL and ack_i = '0' and d2 /= REALNULL
then
            y <= d2 after td;
            ack_o <= '1' after td;

      elsif y /= REALNULL and ack_i = '1' and d1 = REALNULL
and
            d2 = REALNULL then
            y <= REALNULL after td;
            ack_o <= '0' after td;
      end if;
  end process;
end BEH;
```