# A Novel Processor for Artificial Intelligence Acceleration

ATANAS N. KOSTADINOV
Department of Computer Systems and Technologies
Technical University of Sofia, Plovdiv Branch
25 Tsanko Diustabanov Str., 4000 Plovdiv
BULGARIA

GUENNADI A. KOUZAEV
Department of Electronic Systems
Norwegian University of Science and Technology - NTNU
Gløshaugen, O.S. Bragstads plass 2B, 7491 Trondheim
NORWAY

*Abstract:* - A variable predicate logic processor (VPLP) is proposed for artificial intelligence (AI), robotics, computer-aided medicine, electronic security, and other applications. The development is realized as an accelerating unit in AI computing machines. The difference from known designs, the datapath of this processor consists of universal gates changing on-the-fly their logical styles-subsets of predicate logic according to the data type and implemented instructions. In this paper, the processor's reconfigurable gates and the main units are proposed, designed, modeled, and verified using a Field-Programmable Gate Array (FPGA) board and corresponding computer-aided design (CAD) tool. The implemented processor confirmed its reconfigurability on-the-fly performing testing codes. This processor is interesting in accelerating AI computing, molecular and quantum calculations in science, cryptography, computer-aided medicine, robotics, etc.

*Key-Words:* - Variable predicate logic processor (VPLP), predicate logic, artificial intelligence (AI), predicate RAM (PRAM), topological computing, Hilbert-space pseudo-quantum computers, hybrid quantum-classical computers, Field-Programmable Gate Array (FPGA).

Received: June 29, 2021. Revised: April 13, 2022. Accepted: May 11, 2022. Published: July 1, 2022.

## 1 Introduction

In artificial intelligence (AI), many data flows have very complicated structures requiring fast change of the logic processing styles. Partially, this idea is realized in FPGAs (Field-Programmable Gate Arrays), where a designed processor is modeled by programmed computing cells. Unfortunately, moving from one design to another requires an essential reconfiguration time [1]-[5]. Meanwhile, accelerated change of logic style requires fine-grain reconfigurability on the gate level [6],[7].

In this paper, a new specific approach to this reconfigurability is discussed. It is known that predicate logic (the logic of our intelligence) is general for many logic styles, including the Boolean one, for instance [8]-[10].

If universal predicate gates controlled by instructions are realized, they can fulfill particular logic operations of different styles. We have already published the first ideas and circuits in this field in Refs. [11]-[14]. These contributions describe only initial designs for several different logic styles.

Section 2 is on the theory and hardware realizations of predicate logic and its subsets that can be unified in a single processor. In Section 3, the proposed variable predicate logic processor is described in detail. Section 4 is on implementing the processor in FPGA and its verification. Concluding remarks are in Section 5.

## 2 Predicate Logic and Processing Units

In mathematics, predicate logic is a generic term for formal symbolic systems [8]-[10],[15],[16]. This predicate system is distinguished from others in the formula $S$ containing variables $A$ and quantifiers $T$ .

$$S=\left(A,T\right) \tag{1}$$

Thus, predicate logic operates with sentences $S$ similar to the atomic one (1) instead of truth tables

of propositional logic [8]. Some predicate logic applications are used in computer science. They could be found in AI modeling software, big data-based systems, circuit theory, hardware verification codes, etc. [17]-[21].

However, such programs are mostly executed on processors built on propositional logic gates. Depending on the number of quantifiers, this source-code-level simulation can increase the execution time in orders of magnitude compared to possible micro-parallel gates realized operations with these predicated data streams.

In some hardware, the predicate gates of fixed logic and even large units are implemented to enhance the processor parameters, as it was in the Itanium processor architecture [22]. Several ideas were published to modify the conventional computer modules for better processing Prolog programs [23]-[26] or enhance information exchange in multi-processor supercomputing systems [27].

Today, the computing devices involved in massive AI operations [28] require new designs called artificial intelligence accelerators [15], [29]-[34]. Some of them can be built on the combined use of propositional and predicate logic units [12] to improve AI computers' performance.

According to our best knowledge, the first application-specific instruction-set processor (ASIP) accelerating some AI operations was a predicate logic processor published in Refs. [35],[36].

The idea of computing the electromagnetic (EM) signals carrying predicated information relates to the 90$^{th}$ of the last century [11],[37]-[41]. The elementary binary predicate or atomic unit of knowledge [42]-[44] is a pair of logically coupled bits for the formula (1). They can be carried by two logically or even EM coupled wires [39].

Generally, predicate logic uses an extended set of logical and non-logical symbols. Among them are the quantifier ones, conjunction (AND), disjunction (OR), negation (NOT), and implications (*if-then*).

A reduced predicate logic in Ref. [36] and here uses only the AND, OR, and NOT logical operations applied to a predicate expression $S$ :

$$\begin{aligned} S &= \overline{S} \ \text{(NOT)}, \\ \mathrm{S} &= S_1 \wedge S_2 \ \text{(AND)}, \\ \mathrm{S} &= S_1 \vee S_2 \ \text{(OR)}. \end{aligned} \qquad (2)$$

After developing predicate gates according to the formula (2), an experimental 8-bit processor consisting of a predicate datapath and a conventional control unit was designed [36]. The datapath there implements the mentioned logically full set of predicate operations (2) in a parallel manner.

This processor, thought a predicate logic accelerator, was modeled by VHDL (Very High-Speed Integrated Circuit Hardware Description Language) and synthesized in FPGA board from Intel (formerly Altera) using Quartus II design software. The realized microprocessor works at a maximum clock frequency of 130.28 MHz. It consists of 5868 total logic elements, 3482 combinational functions, 4628 registers, and 10624 memory bits. The results of some testing programs were observed helped by the Quartus II tool and successfully compared with theoretical calculations.

Ref. [36] shows the need for further enhancement of the designed predicate processor. It was overly specific for some practical applications. As a rule, the data is not always organized in predicate form in knowledge-based applications. Many flows need Boolean, multi-valued, reversible, etc., operations. Performing them by fixed predicate gates requires an additional program code. In this way, it leads to a decrease in throughput.

As it was mentioned in the Introduction, the main idea of this paper is the development of a processor whose universal gates are controlled by instructions and realize several subsets of predicate logic. This possibility was noticed in the first works on spatially-modulated signals propagating along paired wires in Refs. [11],[37],[39]. There, one of the predicate logic units in the formula (1) can be assigned to control a logic type or realize the reversibility of gates [14]. Additionally, the paired wires can be used to model qubits in quantum computer emulators [11],[41].

In some applications, such as security-enhanced data processing, the paired wires can be used to avoid or diminish information leakage through irradiation from signal traces or/and power delivering wires. Again, this pair-wire style is a subset of the predicate logic set (2).

In all these cases, the signals propagating along the paired lines, formally in predicate form, require new universal reconfigurable gates and newly built arithmetic logic units (ALUs).

In this article, based on our experience in the development, design, simulation, and FPGA implementations, a novel flexible processor architecture tailored to modern artificial intelligence applications is considered prospective to boost AI operations. The predicate flows are combined with

conventional data representation in a specially designed microprocessor containing flexible ALU.

As a difference from all other microprocessors, the processor's datapath can perform operations logically equal to the results produced by seven types of logic. These logics allow new possibilities which have been not realized earlier in full:

(1) Predicate logic with the paired wires
(2) Conventional Boolean operations along each wire (depending on signal and instruction) [14]
(3) Multi-valued (with four logic levels) operations spatially mapped on two wires [45]-[47]
(4) Pseudo-quantum logic [13],[41],[45], [48]-[59]
(5) Reversible logic [60]-[61]
(6) Dual-rail operations [62],[63]
(7) Dual-rail single-spacer operations [64],[65]
(8) Dual-rail dual-spacer operations [66],[67]

The initial designs of universal gates performing the above-considered operations have already been published in Refs. [13],[14]. More information is needed on pseudo quantum gates, which are not widely known to the electronic community.

It is known that quantum computing can be powerful in some cases because of quantum parallelism when $n$ - particles are in $2^n$ states. We need $2^n$ classical electronic gates integrated into a $2^n-$ dimensional Hilbert-space processor to emulate a quantum computing unit. The initial idea in this field was from R.J.C. Spreeuw, who discussed building a Hilbert-space processor using photons of opposite polarization [50] paired into qubits. Unfortunately, the use of a multitude of bulky optical elements is a rather challenging problem.

Contemporary electronics integrating billions of gates allows emulating a several-ten-qubit quantum machine. In 1999, we proposed using the microwave or digital electronics when a sum of even and odd modes in coupled strip lines models a qubit state because they have topologically different electromagnetic field maps [41]. A logically full set of gates was designed and realized in hardware by us in those years [40],[48],[49].

In Fig. 1 (not published earlier), a PCB board for a $\sqrt{\text{CNOT}}$ gate and switch-controlled signal generator (designed with A. Ermakov in 1999) is shown as an example. The gate is described in detail in Ref. [40].

The interest in emulation of quantum computers has been strong for many next following years [13], [14],[51]-[55], considering the problems in the developments of full-scale fault-tolerant quantum processors. It was found that pseudo-quantum architectures, being still classical, can calculate the quantum algorithms used in cryptography, quantum physics, chemistry, and biology more effectively than ordinary computers [53]-[56]. It is known, emulating quantum computers, that not all operations are with qubits; then, a universal computer should have gates performing Boolean and other operations belonging to the predicate set. Besides, in AI applications, quantum algorithms are not always powerful.

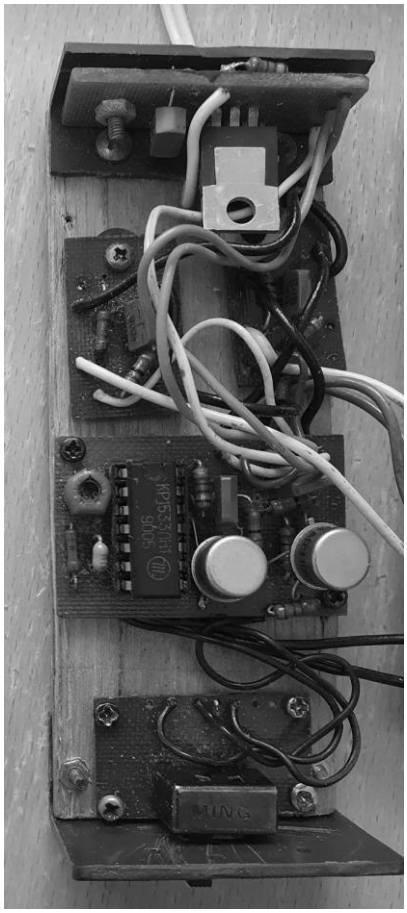

Fig. 1: A gate module (from G.A. Kouzaev's archive, see as well [40]).

The proposed here processor, called the variable logic one, can change its logic styles on-the-fly according to the incoming data flow and corresponding control signal. It will increase the effectiveness of data processing. Considering that all eight mentioned operations are the subsets of

predicate logic, the full name of our design is the *Variable Predicate Logic Processor (VPLP)*

## 3 Variable Predicate Logic Processor (VPLP) Design

This VPLP architecture [13] has been developed in three major steps. Initially, the design of the variable predicate logic gates is performed [14], which is not considered here. The PRAM (Predicate Random-Access Memory) is also composed and designed in the second step. Finally, the complete variable predicate logic processor has been realized and verified using an appropriate CAD (Computer-Aided Design) tool and an FPGA board (Fig. 2).

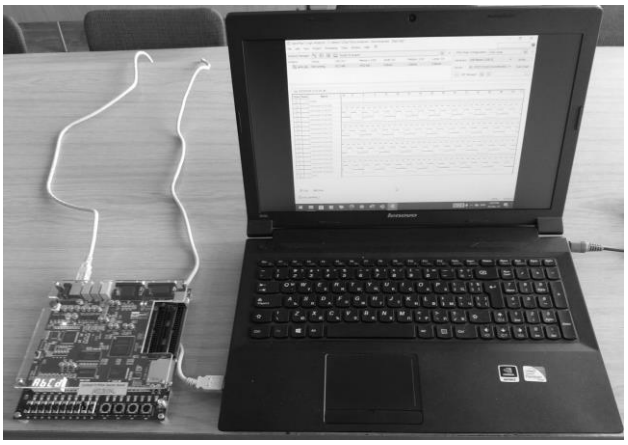

Fig. 2: Cyclone II FPGA Starter Development Board (Altera, now Intel) is connected to a computer to emulate the designed VPLP.

An 8-bit predicate processor is studied, i.e., each value in predicate expression (1) is represented by 8-bit digits.

A PRAM basic cell has been implemented in the second step of the processor development (Fig. 3). This basic cell has two inputs and two outputs for predicated signals. These signals can be of predicate information origin or contain the bits for control of logic of predicate gates.

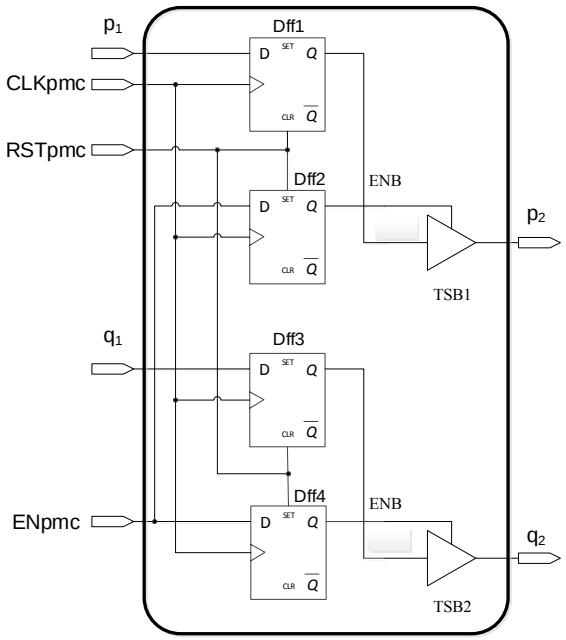


Fig. 3: Implemented PRAM cell.

This design uses four D flip-flops (from Dff1 to Dff4) and two three-state buffers (TSB1 and TSB2, denoted by triangles). The inputs *CLKpmc* and *RSTpmc* are for the clock and reset signals. The signal *ENpmc* enables the input of this PRAM unit. When the *ENpmc* signal is equal to the logic zero, then the three-state buffer outputs go to a high-impedance state. In this case, PRAM basic cell is disabled. In the opposite case, the *ENpmc* signal goes to logic one.

A new 8-bit PRAM module is designed (Fig. 4) when eight cells are combined. Two memory data buses have 8-bit width. All other signals are equal to the described ones in Fig. 3 (a basic predicate memory unit). Then, 256 8-bit PRAM cells are connected, and PRAM is organized as 256 words by 16 bits. An address decoder and a multiplexer have been added to this PRAM module (they are not included in Fig. 4 due to simplification reasons).

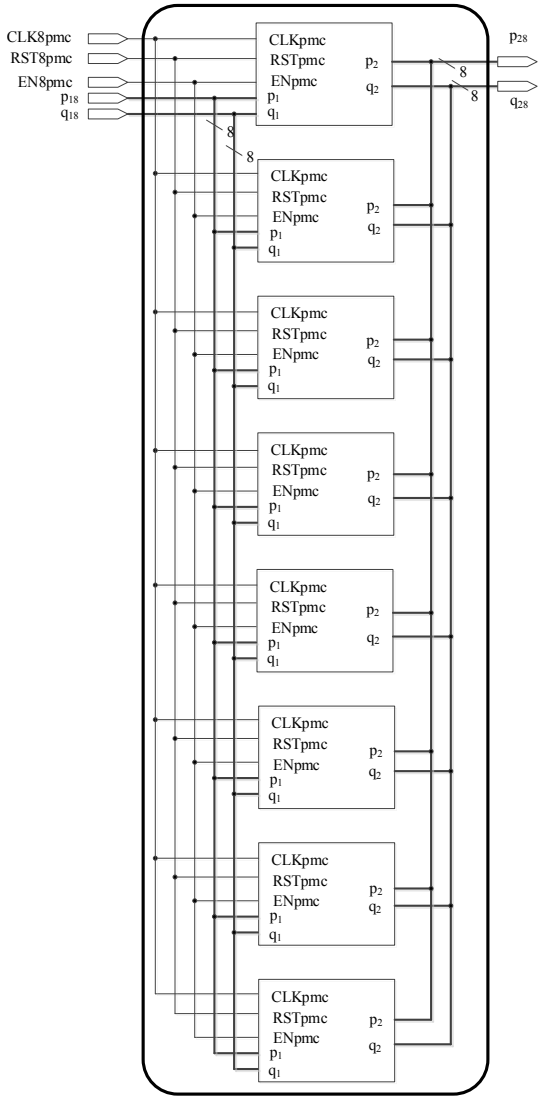


Fig. 4: Designed 8-bit PRAM cell.

The complete variable predicate logic processor has been realized and verified in the final step. VPLP is a successor of the PLP (Predicate Logic Processor) [36] and PBOP (Predicate and Boolean Operation Processor) [12] processor architectures. It extends the architectures mentioned above. The instruction set is enlarged with new instructions. It has been used term flexible processor to express its opportunity to tune to different types of incoming data. The synthesized block diagram of the variable predicate logic processor is shown in Fig. 5. The VPLP includes a reset circuit, datapath, and control unit.

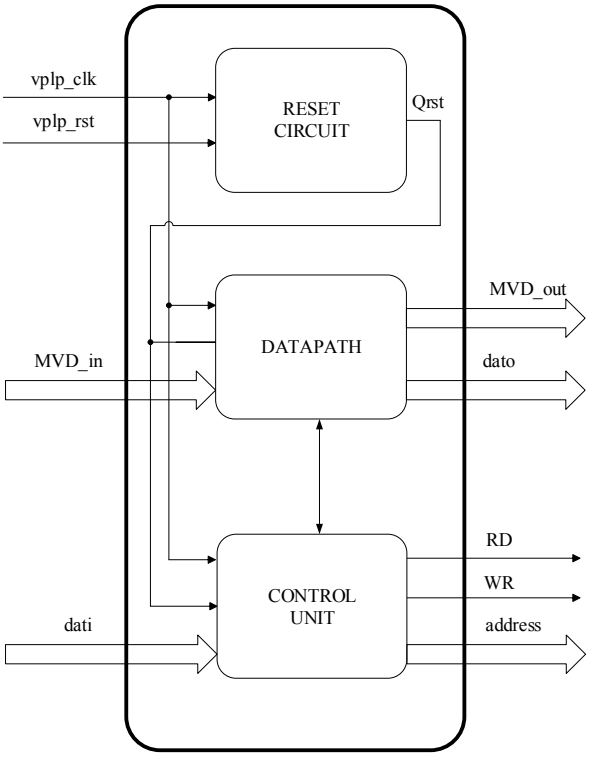


Fig. 5: Synthesized variable predicate logic processor (VPLP).

Another part of the variable predicate logic processor is its interface. It includes the signal lines *vplp_clk*, *vplp_rst*, *MVD_in*, *MVD_out*, *dati*, *dato*, *RD*, *WR*, and *address*. The lines *vplp_clk* and *vplp_rst* interface the clock and reset (*Qrst* is produced by reset circuit) signals to various components of VPLP. Signal lines *MVD_in* and *MVD_out* are the input and output of multi-valued numbers to the variable predicate logic processor. At the input, the multi-valued numbers are converted to binary ones and vice versa to the output using convertors.

The rest signal lines (Fig. 5) connect VPLP to the PRAM module. *RD* and *WR* signals are utilized to perform the read and write memory operations. A signal line *address* is the address bus of the variable predicate logic processor. The data buses of the VPLP are formed by *dati* and *dato* signals.

In Fig. 6, the variable predicate logic processor datapath is shown. It is responsible for the manipulation of data. It consists of the storage units: register B, accumulator A, multi-valued register, flag register (FLAGs), and the combinational units: data multiplexer and variable predicate arithmetic logic unit (VPALU).

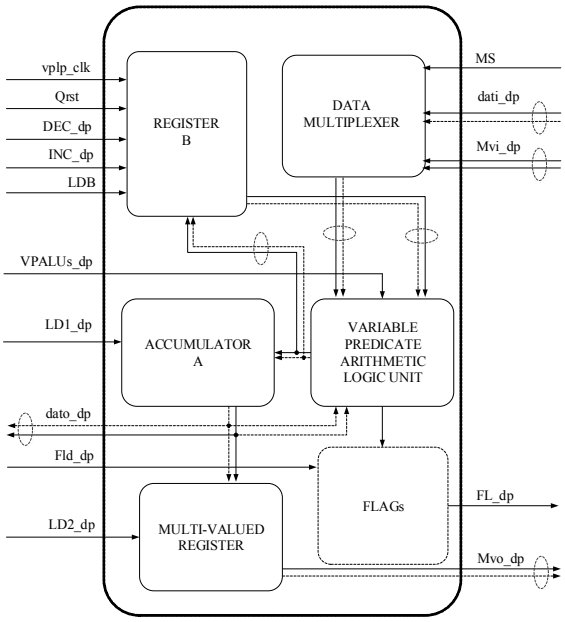


Fig. 6: Designed VPLP datapath.

The data multiplexer in Fig. 6 selects the data either from PRAM (*dati_dp*) or from multi-valued signal lines *Mvi_dp* and sends the selected input's data to VPALU. This data multiplexer has a select line, which is named *MS*. The VPALU control bits belong to signal line *VPALUs_dp*, which is of 5-bit size. Accumulator A and register B are provided to aid in executing instructions in VPALU.

Accumulator A is a 16-bit register (Fig. 6). It usually contains one of the two operands involved in actual instruction execution. The second operand is read from PRAM or register B. The result of an operation is again stored in accumulator A, either register B or both. The load signal line *LD1_dp* is applied to load/store operations.

Register B is 16-bit, too. It plays the same role as accumulator A in some instructions. The added letter B in the assembly instruction names (in mnemonics) specifies the application of register B in their execution.

Accumulator A and register B are both used in some operations. Clock *vplp_clk* and reset *Qrst* signal lines are distributed to other datapath components except for the data multiplexer and VPALU. The decrement *DEC_dp*, increment *INC_dp* and load *LDB* signal lines only apply to register B.

The multi-valued register (Fig. 6) is 16-bit that stores the processed multi-valued data. Load signal line *LD2_dp* is used for this operation to be performed. The multi-valued register output signal line is *Mvo_dp*. The output signal line of datapath for a different type of data, excluding multi-valued one, is *dato_dp*.

A flag register (FLAGs) is a 3-bit one containing three status flags. These bits are set to logic one or logic zero based on the results after completion of the comparison operation by VPALU. The FLAGs load signal line is *Fld_dp*. When logic one is applied to this load signal line *FLd_dp*, the VPLP flags are saved into the flag register. They are placed on the output signal line *FL_dp*.

Variable predicate arithmetic logic unit (VPALU) performs arithmetic and logical operations (Fig. 6). The current design extends these operations with the ones applicable to several types of logics: mainly the Boolean, predicate, multi-valued (multiple-valued), pseudo-quantum, reversible, and dual-rail (differential) logics and its modifications using a single spacer (all-zeroes state) or dual spacers (all-zeroes and all-ones states).

The corresponding processor instructions have been implemented. Table 1 shows new instructions belonging to the VPLP instruction set. The old ones, including a part of the Boolean and predicate instructions, are inherited from previous PLP and PBOP variants [12],[36].

Table 1. New VPLP instructions.

| Command | Semantic |
|---|---|
| **Arithmetic and load instructions** | |
| INCA | Increment the content of accumulator A by 1 |
| DECA | Decrement the content of accumulator A by 1 |
| INCB | Increment the content of register B by 1 |
| DECB | Decrement the content of register B by 1 |
| LDB | Load constant into register B |
| LBA | Load register B with the content of accumulator A |
| LAB | Load accumulator A with the content of register B |
| **Multi-valued logic instructions** | |
| INM | Load accumulator A with converted quaternary to binary number |
| OUTM | Load multi-valued register with processed binary number |
| **Pseudo-quantum logic instructions** | |
| CNB | The logic equivalent of controlled-NOT (CNOT) operation (according to the truth table) |
| SWB | The logic equivalent of the swap operation (with an identical truth table) |

Table 1. (continued)

| **Reversible logic instruction** | |
|---|---|
| FGO | Logic equivalent (with identical truth table) of Fredkin gate operation |
| **Dual-rail (differential) logic instructions** | |
| DOR | Dual-rail OR operation |
| DAND | Dual-rail AND operation |
| **Dual-rail (differential) logic instructions using a single spacer (all-zeroes state)** | |
| TNOT | Dual-rail NOT operation |
| TOR | Dual-rail OR operation |
| TAND | Dual-rail AND operation |
| **Dual-rail (differential) logic instructions using dual spacers (all-zeroes and all-ones states)** | |
| SNOT | Dual-rail NOT operation |
| SOR | Dual-rail OR operation |
| SAND | Dual-rail AND operation |
| **Reversible logic instruction** | |
| FGO | Logic equivalent (with identical truth table) of Fredkin gate operation |
| **Dual-rail (differential) logic instructions** | |
| DOR | Dual-rail OR operation |
| DAND | Dual-rail AND operation |
| **Dual-rail (differential) logic instructions using a single spacer (all-zeroes state)** | |
| TNOT | Dual-rail NOT operation |
| TOR | Dual-rail OR operation |
| TAND | Dual-rail AND operation |
| **Dual-rail (differential) logic instructions using dual spacers (all-zeroes and all-ones states)** | |
| SNOT | Dual-rail NOT operation |
| SOR | Dual-rail OR operation |
| SAND | Dual-rail AND operation |

Additional information about the VPLP instruction set is presented in the following lines:

- **INCA** – VPALU increments by one accumulator A. The result is stored in accumulator A.
- **DECA** – VPALU decrements by one accumulator A content. The result is saved in accumulator A.
- **INCB** – Variable predicate logic controller sets a logic one on datapath signal line *INC_dp*. The content of register B is incremented by one.
- **DECB** – Variable predicate logic controller sets a logic one on datapath signal line number *DEC_dp*. The content of register B is decremented by one.
- **LDB** – The program counter is incremented by one. From the next PRAM cell (each memory cell is 16-bits wide), the 16-bit operand is fetched. With VPALU pass-through operation, the operand is loaded to register B.
- **LBA** – 16-bit operand is initially stored in accumulator A. With a VPALU pass-through operation, the operand is transferred to register B.
- **LAB** – Register B contains a 16-bit operand initially. With VPALU pass-through operation, the operand is moved to accumulator A.
- **INM** – 16-bit value (converted quaternary to binary number) is placed on VPLP signal lines *MVD_in*. With a VPALU pass-through operation, the operand is loaded into accumulator A.
- **OUTM** – 16-bit operand is initially stored in accumulator A. With datapath load signal line *LD2_dp* the operand is transferred to a multi-valued register.
- **CNB** – Register B holds two 8-bit operands. According to the controlled-NOT (CNOT) truth table, the result is again stored in register B.
- **SWB** – Register B holds two 8-bit operands. The result, according to the SWAP truth table, is contained again in register B.
- **FGO** – The first 8-bit operand is stored in the most significant byte of accumulator A and the second two 8-bit operands – in register B. According to the Fredkin gate truth table, the result is kept again in the same registers (one 8-bit result in the most significant byte of accumulator A and two 8-bit results in register B).
- **DOR** – Two 8-bit dual-rail operands are loaded into accumulator A and register B. The dual-rail OR logic operation results are stored again in the same registers.
- **DAND** – Two 8-bit dual-rail operands are loaded into accumulator A and register B. Result of dual-rail AND logic operation is kept again in the same registers.

**TNOT**, **TOR**, **TAND, SNOT, SOR, and SAND** instructions perform the dual-rail NOT, OR, and AND logic operations. The operand and result for TNOT and SNOT instructions are in accumulator A only. The operands and results for TOR, TAND, SOR, and SAND instructions are in accumulator A and register B. A single spacer (all-zeroes state) or dual spacer (all-zeroes and all-ones

states) can be used during transmission for each of the two groups of three instructions, respectively.

The control unit (CU) issues the appropriate signals to be executed the current command. The CU also performs instruction fetching and decoding. It consists of sequential components such as an instruction register, index counter, program counter, another register, variable predicate logic controller, and combinational units, which are the address multiplexer and adder. Fig. 7 shows the internal architecture of the VPLP control unit.

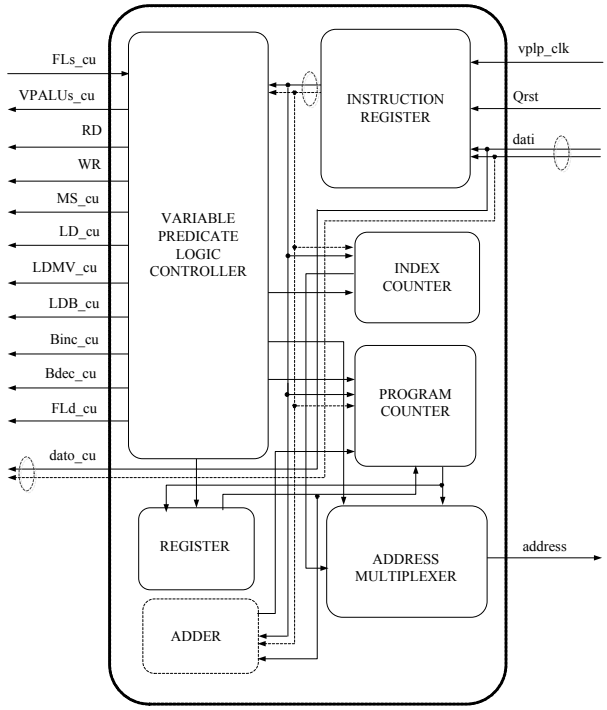


Fig. 7: Realized VPLP control unit.

The instruction register is 16-bit. It holds the instruction fetched from PRAM. Every instruction is encoded in eight bits. Clock *vplp_clk* and reset *Qrst* signal lines are distributed to other control unit sequential components. A variable predicate logic controller (VPLCl) is the main component of the control unit. The VPLCl is realized as a finite state machine (FSM). Each state of the FSM corresponds to a different instruction encoded in eight bits. Depending on decoded instruction, other control signals are issued and sent to datapath, control unit components, and PRAM. These signals are required for proper instruction execution.

PRAM data is transferred to the instruction register using signal lines *dati*. Data are also sent to signal lines *dato_cu*. The VPLCl output signal line *VPALU_cu* selects a VPALU operation.

PRAM read or write operation is performed when *RD* or *WR* signals are applied. The select signal line *MS_cu* is coupled to the data multiplexer signal line *MS* (in Fig. 6). The following output signal lines are the load ones. Signal line *LD_cu* is set to load the accumulator A. Signal line *LDMV_cu* is used to load the multi-valued register. The next signal line *LDB_cu* applies to register B. When logic one is set to *Binc_cu* or *Bdec_cu* control signals, it is possible to increment or decrement the value of register B.

The index counter (IC) can contain the operand address. The IC output line is connected to one address multiplexer inputs.

Adder is implemented to calculate the operand address when a branch instruction is executed. It adds the offset value to the current program counter content. The result is loaded into the program counter.

The additional register (located under VPLCl in Fig. 7) stores the program counter content. Later, this program counter could be loaded again with stored value.

Address multiplexer selects the address signals from the program counter or the index counter. The selected address will appear on the address bus *address*.

The Program Counter (PC) is an 8-bit digital component, and it holds the address of the next instruction, which must be executed. This PC needs to be incremented by one count for every instruction or two of them. Its output is the signal line *PCout*. Fig. 8 illustrates the Program Counter implementation.

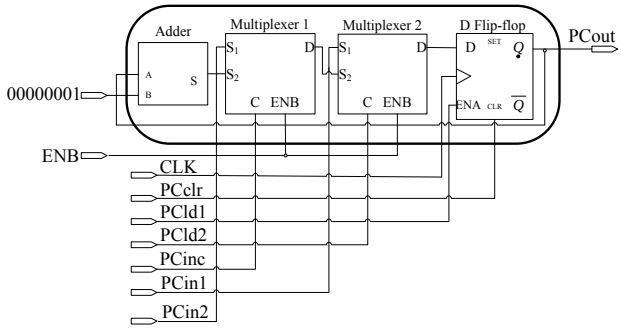


Fig. 8:Implemented VPLP's Program Counter.

This electronic circuit consists of an 8-bit adder, an 8-bit register (denoted as a D flip-flop in Fig. 8), and two multiplexers (Mux1, Mux2). The register is incremented by one helped by the adder and corresponding signal line *PCinc* (in logic high). For this purpose, the input operand is used equally to the value one (0000 0001) placed on input *B* of this adder. The 8-bit register could be loaded using

two multiplexers and corresponding load signal lines *PCld1* and *PCld2*. Two values are applied to 8-bit input buses *PCin1* and *PCin2*. For multiplexers to work properly, they must be enabled (ENB signal must be logic 1).

The signal lines *CLK* and *PCclr* (connected to *Qrst*) are clock and reset signals correspondingly. Fig. 9 shows the VPLP reset circuit [68].

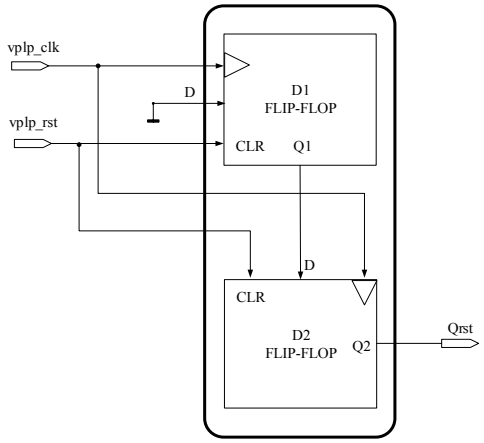


Fig. 9: VPLP reset circuit.

The main building blocks are two D flip-flops D1 and D2. Clock *vplp_clk* and reset *vplp_rst* signal lines are connected to both flip-flops. *D* input of the first D flip-flop is coupled to ground potential. Its output signal line (*Q1*) is connected to the *D* input of the second flip-flop. The output of the reset circuit is the signal line *Qrst*.

The reset circuit is used to synchronize asynchronous VPLP reset signals (*Qrst*). It is avoided any potential problems with asynchronous reset using this presented circuit.

# 4 Variable Predicate Logic Processor Testing

The designed VPLP is connected to the PRAM module, as it is shown in Fig. 10.

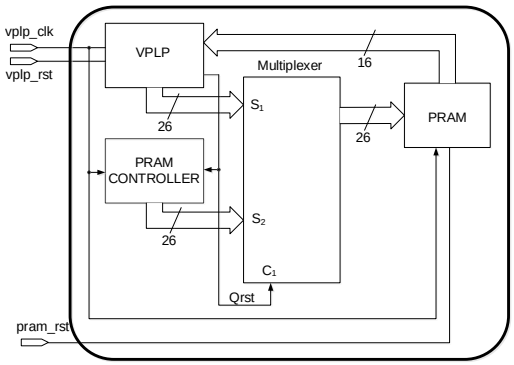


Fig. 10: Designed VPLP connected to PRAM.

A test program is coded and loaded into memory using an additional PRAM controller and a multiplexer. VPLP and PRAM controllers have reset signals with different logic levels. VPLP has an active-high reset signal, and the PRAM controller uses an active-low reset signal. The reset signal (*Qrst*) is connected to the select input of the multiplexer ($C_1$). In this way, when the PRAM controller works, VPLP is idle and vice versa. The PRAM reset signal is *pram_rst*.

VPLP address bus, data (*dato*), and read/write signals are coupled to the first data input of the multiplexer ($S_1$). The same signals of the PRAM controller are connected to the second data input of the multiplexer ($S_2$). Another part of VPLP data bus information (*dati*) is transferred directly to the processor. Clock signal *vplp_clk* is distributed to VPLP, PRAM controller, and PRAM. Reset signal *vplp_rst* is connected to VPLP's reset circuit input.

During the VPLP verification phase, the results obtained from the test programs used for the earlier designed computer architectures [12],[36] are compared with those obtained using this new architecture. Then, the implemented further instructions are checked for correct work. It is done with the SignalTap II Embedded Logic Analyzer, which is a part of Quartus II design software [69] and Cyclone II FPGA Starter Development Board **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**.

## 4.1 Test program 1

The final step in the VPLP verification process is executing several test programs. One example of them, with short comments, is given below.

- **LDB #$55AF** ; Register B is loaded with hexadecimal value "55AF" (timestamp – ts 11).
- **LAB** ; Accumulator A is loaded with hexadecimal value "55AF" (ts 15).
- **CNB** ; Operation, logically equal to control-NOT, is executed between 8-bit numbers located in register B (ts 22).
- **FGO** ; Operation, logically equal to Fredkin gate, is performed. 8-bit A operand is in the MSB of accumulator A. The other two 8-bit operands (B and C) are maintained in register B (ts 26).
- **LDB #$AACD** ; Register B is loaded with hexadecimal value "AACD" (ts 35).
- **LAB** ; Accumulator A is loaded with hexadecimal value "AACD" (ts 39).

- **CNB** ; Operation, logically equal to control-NOT one, is executed between 8-bit numbers in register B (ts 46).
- **FGO** ; Operation, logically equal to Fredkin gate one, is performed. 8-bit A operand is in the MSB of accumulator A. The other two 8-bit operands (B and C) are maintained in register B (ts 50).
- **NOP** ; There is no processor operation (ts 54).
- **HLT** ; VPLP stops execution of any instructions (ts 58).

The checked instructions in the above-presented test program are the **LDB**, **LAB**, **CNB**, **FGO**, **NOP**, and **HLT** ones. The SignalTap II Embedded Logic Analyzer's captured data is compared with one based on theoretical calculations. The conclusion is that the designed VPLP works appropriately. The basic signals used in the verification process are shown in Table 2.

Table 2. VPLP test signal legend.

| Key | Signal name | Signal explanation |
|---|---|---|
| 0 | *vplp_rst* | VPLP reset signal (It is assigned pushbutton KEY [0] to *vplp_rst*) |
| 1 | PRAM\|*address* | PRAM address |
| 2 | PRAM\|*dati* | PRAM input data |
| 3 | PRAM\|*dato* | PRAM output data |
| 4 | PRAM\|*RD* | PRAM read signal |
| 5 | PRAM\|*WR* | PRAM write signal |
| 6 | control_unit\|VPLPcontroller \|*OPCODE* | Operation code of executed instruction |
| 7 | control_unit\|PC\|*PCout* | Program counter output data |
| 8 | datapath\|ACCA\|*Q1* | Accumulator A output data |
| 9 | datapath\|REGB\|*regBout* | Register B output data |

The clock frequency of this variable predicate logic computer prototype in Figure 9 is 50 MHz (PIN_L1 of the Cyclone II FPGA Starter Development Board is connected to the *vplp_clk* signal **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**).

An essential part of the collected data is included in Table 3. It is a portion of the created SignalTap II Embedded Logic Analyzer list file. The data in this Table 3 is captured using the clock (*vplp_clk*) as an acquisition signal. The VPLP reset signal (*vplp_rst*) is a trigger one. The signal keys from Table 2 are in the first row of Table 3.

The sample depth of the SignalTap II Embedded Logic Analyzer data buffer is specified to get 128 samples. Table 3 presents half of them. The first column contains the time in which the logic value of the test signals is registered. Minus sign (-) denotes a period before a trigger signal appears.

Table 3. A part of SignalTap II Embedded Logic Analyzer list file.

| *t* | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
|---|---|---|---|---|---|---|---|---|---|---|
| -2 | 1 | 00h | 0000h | 0000h | 0 | 0 | 07h | 00h | 0000h | 0000h |
| -1 | 1 | 00h | 0000h | 0000h | 0 | 0 | 07h | 00h | 0000h | 0000h |
| 0 | 0 | 00h | 0000h | 0000h | 0 | 0 | 07h | 00h | 0000h | 0000h |
| 1 | 0 | 00h | 0000h | 0000h | 0 | 0 | 07h | 00h | 0000h | 0000h |
| 2 | 0 | 00h | 0000h | 0000h | 0 | 0 | 07h | 00h | 0000h | 0000h |
| 3 | 0 | 00h | 0000h | 0000h | 1 | 0 | 07h | 00h | 0000h | 0000h |
| 4 | 0 | 00h | 0000h | 171Dh | 0 | 0 | 07h | 00h | 0000h | 0000h |
| 5 | 0 | 00h | 0000h | 171Dh | 0 | 0 | 07h | 00h | 0000h | 0000h |
| 6 | 0 | 00h | 0000h | 171Dh | 0 | 0 | 17h | 00h | 0000h | 0000h |
| 7 | 0 | 00h | 0000h | 171Dh | 0 | 0 | 17h | 00h | 0000h | 0000h |
| 8 | 0 | 00h | 0000h | 171Dh | 0 | 0 | 17h | 00h | 0000h | 0000h |
| 9 | 0 | 01h | 0000h | 171Dh | 1 | 0 | 17h | 01h | 0000h | 0000h |
| 10 | 0 | 01h | 0000h | 55AFh | 0 | 0 | 17h | 01h | 0000h | 0000h |
| 11 | 0 | 01h | 0000h | 55AFh | 0 | 0 | 17h | 01h | 0000h | 55AFh |
| 12 | 0 | 01h | 0000h | 55AFh | 0 | 0 | 1Dh | 01h | 0000h | 55AFh |

Table 3. (continued)

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 13 | 0 | 01h | 0000h | 55AFh | 0 | 0 | 1Dh | 01h | 0000h | 55AFh |
| 14 | 0 | 01h | 0000h | 55AFh | 0 | 0 | 1Dh | 01h | 0000h | 55AFh |
| 15 | 0 | 01h | 55AFh | 55AFh | 0 | 0 | 1Dh | 01h | 55AFh | 55AFh |
| 16 | 0 | 02h | 55AFh | 55AFh | 1 | 0 | 1Dh | 02h | 55AFh | 55AFh |
| 17 | 0 | 02h | 55AFh | 1B20h | 0 | 0 | 1Dh | 02h | 55AFh | 55AFh |
| 18 | 0 | 02h | 55AFh | 1B20h | 0 | 0 | 1Dh | 02h | 55AFh | 55AFh |
| 19 | 0 | 02h | 55AFh | 1B20h | 0 | 0 | 1Bh | 02h | 55AFh | 55AFh |
| 20 | 0 | 02h | 55AFh | 1B20h | 0 | 0 | 1Bh | 02h | 55AFh | 55AFh |
| 21 | 0 | 02h | 55AFh | 1B20h | 0 | 0 | 1Bh | 02h | 55AFh | 55AFh |
| 22 | 0 | 02h | 55AFh | 1B20h | 0 | 0 | 1Bh | 02h | 55AFh | 55FAh |
| 23 | 0 | 02h | 55AFh | 1B20h | 0 | 0 | 20h | 02h | 55AFh | 55FAh |
| 24 | 0 | 02h | 55AFh | 1B20h | 0 | 0 | 20h | 02h | 55AFh | 55FAh |
| 25 | 0 | 02h | 55AFh | 1B20h | 0 | 0 | 20h | 02h | 55AFh | 55FAh |
| 26 | 0 | 02h | 55AFh | 1B20h | 0 | 0 | 20h | 02h | 55AFh | 50FFh |
| 27 | 0 | 03h | 55AFh | 1B20h | 1 | 0 | 20h | 03h | 55AFh | 50FFh |
| 28 | 0 | 03h | 55AFh | 171Dh | 0 | 0 | 20h | 03h | 55AFh | 50FFh |
| 29 | 0 | 03h | 55AFh | 171Dh | 0 | 0 | 20h | 03h | 55AFh | 50FFh |
| 30 | 0 | 03h | 55AFh | 171Dh | 0 | 0 | 17h | 03h | 55AFh | 50FFh |
| 31 | 0 | 03h | 55AFh | 171Dh | 0 | 0 | 17h | 03h | 55AFh | 50FFh |
| 32 | 0 | 03h | 55AFh | 171Dh | 0 | 0 | 17h | 03h | 55AFh | 50FFh |
| 33 | 0 | 04h | 55AFh | 171Dh | 1 | 0 | 17h | 04h | 55AFh | 50FFh |

Table 3. (continued)

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 34 | 0 | 04h | 55AFh | AACDh | 0 | 0 | 17h | 04h | 55AFh | 50FFh |
| 35 | 0 | 04h | 55AFh | AACDh | 0 | 0 | 17h | 04h | 55AFh | AACDh |
| 36 | 0 | 04h | 55AFh | AACDh | 0 | 0 | 1Dh | 04h | 55AFh | AACDh |
| 37 | 0 | 04h | 55AFh | AACDh | 0 | 0 | 1Dh | 04h | 55AFh | AACDh |
| 38 | 0 | 04h | 55AFh | AACDh | 0 | 0 | 1Dh | 04h | 55AFh | AACDh |
| 39 | 0 | 04h | AACDh | AACDh | 0 | 0 | 1Dh | 04h | AACDh | AACDh |
| 40 | 0 | 05h | AACDh | AACDh | 1 | 0 | 1Dh | 05h | AACDh | AACDh |
| 41 | 0 | 05h | AACDh | 1B20h | 0 | 0 | 1Dh | 05h | AACDh | AACDh |
| 42 | 0 | 05h | AACDh | 1B20h | 0 | 0 | 1Dh | 05h | AACDh | AACDh |
| 43 | 0 | 05h | AACDh | 1B20h | 0 | 0 | 1Bh | 05h | AACDh | AACDh |
| 44 | 0 | 05h | AACDh | 1B20h | 0 | 0 | 1Bh | 05h | AACDh | AACDh |
| 45 | 0 | 05h | AACDh | 1B20h | 0 | 0 | 1Bh | 05h | AACDh | AACDh |
| 46 | 0 | 05h | AACDh | 1B20h | 0 | 0 | 1Bh | 05h | AACDh | AA67h |
| 47 | 0 | 05h | AACDh | 1B20h | 0 | 0 | 20h | 05h | AACDh | AA67h |
| 48 | 0 | 05h | AACDh | 1B20h | 0 | 0 | 20h | 05h | AACDh | AA67h |
| 49 | 0 | 05h | AACDh | 1B20h | 0 | 0 | 20h | 05h | AACDh | AA67h |
| 50 | 0 | 05h | AACDh | 1B20h | 0 | 0 | 20h | 05h | AACDh | 22EFh |
| 51 | 0 | 06h | AACDh | 1B20h | 1 | 0 | 20h | 06h | AACDh | 22EFh |
| 52 | 0 | 06h | AACDh | 0807h | 0 | 0 | 20h | 06h | AACDh | 22EFh |
| 53 | 0 | 06 | AAC | 0807 | 0 | 0 | 20 | 06 | AAC | 22EF |

| | | h | Dh | h | | | h | h | Dh | h |
|---|---|---|---|---|---|---|---|---|---|---|
| 54 | 0 | 06 h | AAC Dh | 0807 h | 0 | 0 | 08 h | 06 h | AAC Dh | 22EF h |

Table 3. (continued)

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 55 | 0 | 06 h | AAC Dh | 0807 h | 0 | 0 | 08 h | 06 h | AAC Dh | 22EF h |
| 56 | 0 | 06 h | AAC Dh | 0807 h | 0 | 0 | 08 h | 06 h | AAC Dh | 22EF h |
| 57 | 0 | 06 h | AAC Dh | 0807 h | 0 | 0 | 08 h | 06 h | AAC Dh | 22EF h |
| 58 | 0 | 06 h | AAC Dh | 0807 h | 0 | 0 | 07 h | 06 h | AAC Dh | 22EF h |
| 59 | 0 | 06 h | AAC Dh | 0807 h | 0 | 0 | 07 h | 06 h | AAC Dh | 22EF h |
| 60 | 0 | 06 h | AAC Dh | 0807 h | 0 | 0 | 07 h | 06 h | AAC Dh | 22EF h |
| 61 | 0 | 06 h | AAC Dh | 0807 h | 0 | 0 | 07 h | 06 h | AAC Dh | 22EF h |

### 4.2 Test program 2

Another test program example is given below (with short comments).

o **LDA #$0000** ; Register A is loaded with hexadecimal value “0000”.
o **LDB #$4FB1** ; Register B is loaded with hexadecimal value "4FB1".
o **DECB** ; The value in register B is reduced by one.
o **SWB** ; A SWAP operation is executed. The result, according to the SWAP truth table, is contained again in register B.
o **CNB** ; Operation, logically equal to control-NOT one, is executed between 8-bit numbers in register B.
o **NOP** ; There is no processor operation.
o **HLT** ; VPLP stops execution of any instructions.

The checked instructions in the above-presented test program are the **LDA**, **LDB**, **DECB**, **SWB**, **CNB**, **NOP**, and **HLT** ones. The SignalTap II Embedded Logic Analyzer’s captured data is compared again with one based on theoretical calculations. The conclusion is the same as the previous one that the designed VPLP works appropriately. The basic signals (in particular the contents of registers A and B) used in the verification process are shown in Fig 11.

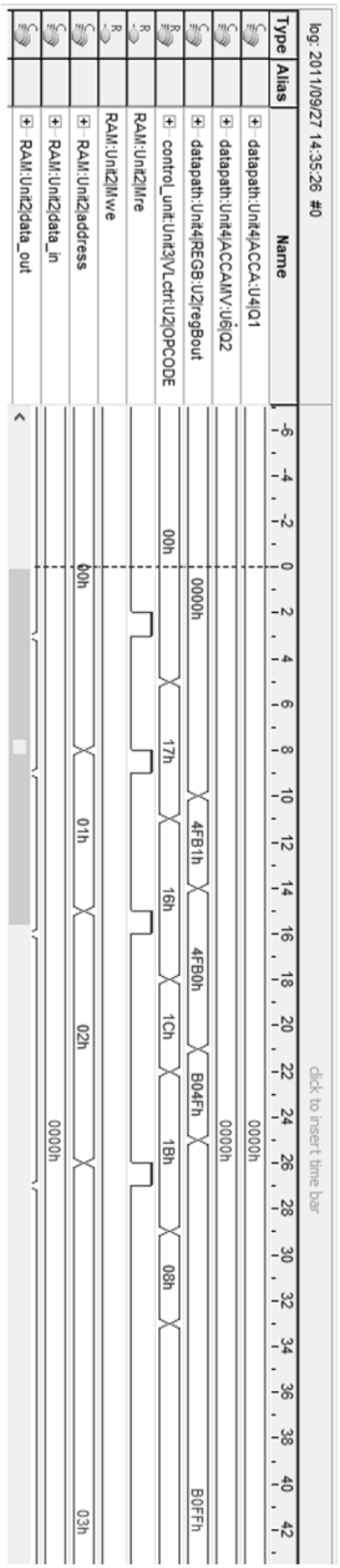

Fig. 11: SignalTap II wave diagram.

### 4.3 Test program 3

The third test program example is given below (with short comments).

- **LDC $0B** ; Index counter (IC) is loaded with hexadecimal value "0B".
- **INC** ; IC is incremented by 1.
- **LDAC** ; Register A is loaded with hexadecimal value "55AA".
- **INC** ; IC is incremented by 1.
- **CPAH** ; The content of register A (high byte) is compared (subtracted) with the value of the memory cell addressed by the IC.
- **BIG $02** ; Branch if flag Greater is equal to one.
- **NOP** ; There is no processor operation.
- **HLT** ; VPLP stops execution of any instructions.
- **INC** ; IC is incremented by 1.
- **STAC** ; The content of register A is stored in a cell with an address specified by the IC.
- **DEC** ; IC is decremented by 1.
- **LDAC** ; Register A is loaded with hexadecimal value "00AA".
- **DEC** ; IC is decremented by 1.
- **STAC** ; The content of register A is stored in a cell with an address specified by the IC.
- **INC** ; IC is incremented by 1.
- **INC** ; IC is incremented by 1.
- **LDAC** ; Register A is loaded with hexadecimal value "55AA".
- **DEC** ; IC is decremented by 1.
- **STAC** ; The content of register A is stored in a cell with an address specified by the IC.
- **HLT** ; VPLP stops execution of any instructions.

Memory cell with address $0C has an initial content $55AA.

Memory cell with address $0D has an initial content $00AA.

The checked instructions in the third test program are the LDC, INC, LDAC, CPAH, BIG, NOP, STAC, DEC, and HLT ones. The SignalTap II Embedded Logic Analyzer's captured data is compared with one based on theoretical calculations. The conclusion is that the designed VPLP works well as it is shown in Fig 12 (in Appendix 1).

During VPLP testing have been used more than ten test programs of different lengths. All instructions belonging to the instruction set of the microprocessor have been checked and their operation is correct.

## 5 Conclusions

In this article, a novel variable predicate logic processor has been presented. The designed VPLP consists of a variable-logic datapath, control unit, reset circuit, and PRAM module to store information.

Depending on the data and generated instructions, the datapath units perform the logical operations belonging to eight subsets of reduced predicate logic, including the predicate, Boolean, multi-valued (4-level), pseudo-quantum, reversible, and pair-wire logic styles. The logic change is realized on-the-fly if it is required. The processor can emulate in hardware many algorithms, including the AI operations and $2^n$ - dimensional Hilbert-space pseudo-quantum computing.

The proposed microprocessor architecture has been developed in three steps: the variable predicate logic gates design, PRAM realization, and final VPLP implementation in an FPGA board (Altera's Cyclone II FPGA Starter Development Kit) with subsequent verification using several test codes.

The invented variable predicate logic processor is interesting in accelerating artificial intelligence applications, enhancing hybrid quantum-classical architectures, molecular and pseudo-quantum calculations used in science, cryptography, computer-aided medicine, robotics, electronic security, etc.

*Acknowledgments:*

The authors thank their colleagues, Drs. M. Olavsbraten (NTNU, Norway) and V. Guitberg (ATSS, Canada) who took part in the earlier stages of the research.

The authors would like to thank the Research and Development Sector at the Technical University of Sofia for the financial support.

*References:*

[1] C. Bobda, *Introduction to Reconfigurable Computing Architectures, Algorithms, and Applications*, Springer, 2007.

[2] *Reconfigurable Computing: From FPGAs to Hardware/Software Codesign*, Cardoso J.M.P and M. Hübner, (Eds.), Springer, 2011.

[3] I. Pérez and M. Figueroa, A Heterogeneous Hardware Accelerator for Image Classification in Embedded Systems, *Sensors,* Vol. 21, Issue 8, 2637, 2021. https://doi.org/10.3390/s21082637

[4] R. Chen,T. Wu,Y. Zheng, and M. Ling, MLoF: Machine Learning Accelerators for the Low-Cost FPGA Platforms, Appl. Sc., Vol. 12, Issue 1, 89, 2021. https://doi.org/10.3390/app12010089

[5] K. Seng, P. Lee, and L. Ang, Embedded Intelligence on FPGA: Survey, Applications and Challenges, *Electronics,* Vol. 10, Issue 8, 895, 2021. https://doi.org/10.3390/electronics10080895

[6] K. Rajagopalan, B. Phillips, and D. Abbott, On-the-fly reconfigurable logic, *SPIE Proc., Smart Structures, Devices, and Systems II*, Vol. 5649, 2005, pp. 101-109. https://doi.org/10.1117/12.582429

[7] M.A. Iqbal and S.A. Khan, *Run-time reconfigurable instruction set processor (RT-RISP): Design and simulation using Verilog-HLD*, Lap Lambert Acad. Publ., 2012.

[8] A.A. Stolyar, *Introduction to Elementary Mathematical Logic*, Dover Publ. Inc., 1983.

[9] E.J. Lowe, *Forms of Thought. A Study in Philosophical Logic*, Cambridge Univer. Press, 2013.

[10] A. Iacona, *Logic: Lecture Notes for Philosophy, Mathematics, and Computer Science*, Springer, 2021.

[11] G.A. Kouzaev, Topological computing, *WSEAS Trans. Comp. Res*., Vol. 5, Issue 10, 2006, pp. 2221-2224. https://www.researchgate.net/journal/WSEAS-Transactions-on-Computer-Research-1991-8755

[12] A.N. Kostadinov and G.A. Kouzaev, Predicate and binary operations processor, *Proc. 8th WSEAS Int. Conf. Appl. El. Eng*., WSEAS, Houston, 2009, pp. 199-204, 2009. https://www.researchgate.net/publication/316495127_Predicate_and_Boolean_operations_processor

[13] G.A. Kouzaev, A.N. Kostadinov, M. Olavsbraten, and V. Guitberg, Variable predicate logic computer architectures, *UK Pat. Appl. GB2508162 dated on 21.11.2012, Searchable Pat. J. 6523*, online published on 28.05.2014, Publ. # GB2508162.

[14] A.N. Kostadinov, V. Guitberg, M. Olavsbraten, and G.A. Kouzaev, Multi-logics gates, *Proc. IEEE Int. Sem. Electron. Dev. Design Production*, Prague, 2019. pp. 1-3. https://doi.org/10.1109/SED.2019.8798452

[15] A.G. Hamilton, *Logic for Mathematicians*, Cambridge Univer. Press, 1988.

[16] Microsoft Corp., Project Brainwave, 2018 (accessed June 26, 2021). https://blogs.microsoft.com/ai/build-2018-project-brainwave/?utm_source=press&utm_campaign=75592,

[17] I. Bratko, *Prolog Programming for Artificial Intelligence*, 4th Edition, Pearson Educ., 2011.

[18] S.P. Vingron, *Switching Theory: Insight through Predicate Logic*, Springer, Berlin, 2004.

[19] V.D. Shet, M. K. Singh, C. Bahlmann, V. Ramesh, S. P. Masticola, J. Neumann, T. Parag, M. A. Gall, and R. A. Suarez, Predicate logic based image grammars for complex visual pattern recognition, *US Pat. 8548231 B2*, 2013 (accessed June 26, 2021). http://www.google.com/patents/US8548231

[20] G. Tzimpragos, D. Vasudevan, N. Tsiskaridze, G. Michelogiannakis, A. Madhavan, and J. Volk, A computational temporal logic for superconducting Accelerators, *Proc. 25th Int. Conf. Arch.. Supp. for Prog. Lang. and Oper. Syst.*, Lausanne, ACM, New York, 2020, pp. 435–448. https://dl.acm.org/doi/10.1145/3373376.3378517

[21] A. Dutt, C. Wang, A. Nazi, S. Kandula, V. Narasayya, and S. Chaudhuri, Selectivity estimation for range predicates using lightweight models, *Proc. VLDB Endowment*, Vol. 12, issue 5, 2019, pp. 1044-1057. https://dl.acm.org/doi/10.14778/3329772.3329780

[22] H. Sharangpani and H. Arora, Itanium processor microarchitecture, *IEEE Micro.*, Vol. 20, 2000, pp. 24-43. https://ieeexplore.ieee.org/document/877948

[23] M. Umemura and M. Yokota, Prolog processing system *US Pat. 4546432 A*, 1986 (accessed June 26, 2021). https://www.google.com/patents/US4546432

[24] K. Kobayashi and M. Sasaki, System for processing data using logic language, *US Pat.*

*5129081 A*, 1992 (accessed June 26, 2021). http://www.google.com.na/patents/US5129081

[25] R.I. Baum, G.A. Brent, D.H. Gibson, and D.B. Lindquist, Database engine predicate evaluator, *US Pat. 5590362 A*, 1996 (accessed June 26, 2021). http://www.google.ch/patents/US5590362

[26] T. Yokota and K. Seo, Pegasus - an ASIC implementation of high-performance Prolog processor, *Proc. EURO ASIC'90*, IEEE, Paris, 1990, pp. 156-159. https://doi.org/10.1109/EASIC.1990.207928

[27] P. R. Pietzuch, K. H. Tsoi, I. Papagiannis, M. Migliavacca, and W. Luk Accelerating publish/subscribe matching on reconfigurable supercomputing platforms, *Proc. Many-core and Rec. Supercomp. Conf.*, Vol. 3, Rome, MRSC, Rome, 2010. https://www.semanticscholar.org/paper/Accelerating-Publish%2FSubscribe-Matching-on-Pietzuch-Tsoi/d9ab550bf483b9adcc4583025e0c44905bea1809

[28] G.F. Luger, *Artificial intelligence: structures and strategies for complex problems solving*, 6th Edition, Pearson Education Inc., Boston, 2009.

[29] D. Monroe, Chips for artificial intelligence, *Commun. ACM*, Vol. 61, 2018, pp. 15-17. https://doi.org/10.1145/3185523

[30] R. Kumar and S. Baul, Artificial intelligence chip market outlook – 2025, 2019 (accessed June 26, 2021). https://www.alliedmarketresearch.com/artificial-intelligence-chip-market,

[31] S. Harini, A. Ravikumar, and D. Garg, VeNNus: An artificial intelligence accelerator based on RISC-V architecture, *Proc. Int. Conf. Comp. Intell. Data Eng.* Singapore, 2020, *In: Lect. Notes Data Eng. Commun. Techn.*, Vol. 56, Springer, pp. 287-300. https://doi.org/10.1007/978-981-15-8767-2_25

[32] A. Shawahna, S. Sait, and A. El-Maleh, FPGA-based accelerators of deep learning networks for learning and classification: A review, *IEEE Access*, Vol. 7, 2019, pp. 7823-7859. https://doi.org/10.1109/ACCESS.2018.2890150

[33] Y. Chi, Z. Zheng, R. Liu, and W. Cui, Design of hardware acceleration system based on FPGA and deep learning algorithm, *Proc. IEEE Int. Conf. Art. Intell. Comp. Apps.*, Dalian, IEEE, New York, 2020, pp. 1332-1337. https://doi.org/10.1109/ICAICA50127.2020.9182658

[34] M. Talib, S. Majzoub, Q. Nasir, and D. Jamal, A systematic literature review on hardware implementation of artificial intelligence algorithms, *J. Supercomp.*, Vol. 77, 2021, pp. 1897-1938. https://doi.org/10.1007/s11227-020-03325-8

[35] G.A. Kouzaev and A.N. Kostadinov, Predicate logic processor of spatially patterned signals, *Proc. WSEAS Int. Conf. Recent Advances in Systems Eng. Appl. Math.*, 2008, pp. 94-96.

[36] G.A. Kouzaev and A.N. Kostadinov, Predicate gates, components and a processor for spatial logic, *J. Circ. Syst. Comp.*, Vol. 40, No. 7, 2010, pp. 1517-1541. https://doi.org/10.1142/S0218126610006888

[37] V.I. Gvozdev and G.A. Kouzaev, Microwave flip-flop for topological computers, *Russian Federation Pat.*, No 2054794, dated May 26, 1992.

[38] G.A. Kouzaev and V.I. Gvozdev, Topological pulse modulation of field and new microwave circuits design for superspeed operating devices, *Proc. ISSE'95 – Int. Symp. Signals, Systems Electron.*, 1995, pp. 383-384. https://doi.org/10.1109/ISSSE.1995.498014

[39] G.A. Kouzaev, Topologically modulated signals and predicate gates for their processing, 2001. https://arxiv.org/abs/physics/0107002v1

[40] G.A. Kouzaev, *Applications of Advanced Electromagnetics. Components and Systems*, Springer, 2013. https://doi.org/10.1007/978-3-642-30310-4

[41] G.A. Kouzaev, I.V. Nazarov, and A.V. Kalita, Unconventional logic elements on the base of topologically modulated signals, 1999. https://arxiv.org/abs/physics/9911065v1

[42] M. Houška, L. Dömeová, and R. Kvasnička, Unary operations with knowledge units, *Proc. 2nd Int. Conf. Software Techn. Eng.*, Vol. 1, San Juan, IEEE, San Juan, 2010, pp. 237-241. https://doi.org/10.1109/ICSTE.2010.5608840

[43] M.H. Zack, Managing codified knowledge, *Sloan Manag.*, Vol. 40, 1999, pp. 45-58.

[44] R. Kowalsky, Predicate logic as programming language, *Proc. IFIP Congress.*, North-Holland Publ. Comp., Amsterdam, pp. 569-574, 1974.

[45] G.A. Kouzaev, V.V Cherny, and T.A. Lebedeva, Multivalued processing spatially modulated discrete electromagnetic signals, *Proc. 30th Europ. Microw. Conf., Paris*, Oct. 2000, pp. 209-213. https://doi.org/10.1109/EUMA.2000.338807

[46] V. Patel and K.S. Gurumurthy, Arithmetic operations in multivalued logic, *Int. J. VLSICS,*

Vol. 1, 2010, Issue 1, pp. 21-32. https://doi.org/10.5121/vlsic.2010.1103

[47] M. Huang, X. Wang, G. Zhao, P. Coquet, and B. Tay, Design and implementation of ternary logic integrated circuits by using novel two-dimensional materials, *Appl. Sci. J.*, Vol. 9, 2019, pp. 1-13. https://doi.org/10.3390/app9204212

[48] G.A. Kouzaev and T.A. Lebedeva, New logic components for processing complex measurement data, *Measurement Tech.*, Vol. 43, 2000, pp. 1070-1073. https://doi.org/10.1023/A:1010948020127

[49] G.A. Kouzaev, Qubit logic modeling by electronic gates and electromagnetic signals, 2001. https://arxiv.org/abs/quant-ph/0108012v2

[50] R.J.C. Spreeuw, A classical analogy of entanglement, *Found. Phys.*, Vol. 28, 1998, pp.361-374. https://doi.org/10.1023/A:1018703709245

[51] S. O'uchi, M. Fujishima, and K. Hoh, An 8-qubit quantum circuit processor, Proc. *IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2002,pp.V-209-212. https://doi.org/10.1109/ISCAS.2002.1010677

[52] L.B. Kish, Quantum computing with analog circuits: Hilbert space computing, *Proc. SPIE Conf. Smart Electron., MEMS, BioMEMS, and Nanotechnology*, March 3, 2003. http://dx.doi.org/10.1117/12.497438

[53] B.R. La Cour and G.E. Ott, Signal based classical emulation of a universal quantum computer, *New J. Phys.*,Vol. 17, 2015, pp. 053017(1-19). http://iopscience.iop.org/1367-2630/17/5/053017/article

[54] M. Halid, N.I. Muhammad, U.M. Khokhar, A. Jafri, and H. Choi, An FPGA based hardware abstraction of quantum computing system, *J. Comput. Electron.*, Vol. 20, 2021, pp.2001-2018. https://doi.org/10.1007/s10825-021-01765-w

[55] M. Borgarino, Circuit-based compact model of electron spin qubit, electronics, Vol. 11, 2022, pp. 526 (1-14). https://www.mdpi.com/2079-9292/11/4/526#

[56] D. O'Shea, Nvidia expands efforts to support hybrid classical-quantum computing, *Fierce Electronics*, March 25, 2022. https://www.fierceelectronics.com/embedded/nvidia-expands-efforts-support-hybrid-classical-quantum-computing

[57] C.P. Williams, *Explorations in Quantum Computing*, 2nd Edition, Springer, London, 2011.

[58] R. Stárek, M. Mičuda, M. Miková, I. Straka, M. Dušek, M. Ježek, and J. Fiurášek, Experimental investigation of a four-qubit linear-optical quantum logic circuit, *Sci. Rep. J.*, Vol. 6, 2016, pp. 1 – 11. https://doi.org/10.1038/srep33475

[59] T. Chattopadhyay, All-optical modified Fredkin gate, *IEEE J. Sel. Top. Quant. Electron.*, Vol. 18, 2012, pp. 585-592. https://doi.org/10.1109/JSTQE.2011.2106111

[60] H.G. Rangaraju, U. Venugopal, K. Muralidhara, and K.B. Raja, Low power reversible parallel binary adder/subtractor, *Int. J. VLSICS*, Vol. 1, 2010, pp. 23-34. https://doi.org/10.5121/vlsic.2010.1303

[61] J. Rice, Project in Reversible Logic, 2005 (accessed June 26, 2021). http://www.cs.uleth.ca/~rice/publications/TR-CSJR1-2005.pdf

[62] J. Waddle and D. Wagner, Fault attacks on dual-rail encoded systems, *Proc. 21st Annual Comp. Security Appl. Conf., IEEE*, Tucson, 2005,pp.483–494. https://doi.org/10.1109/CSAC.2005.25

[63] Z. Xia, M. Hariyama, and M. Kameyama, Asynchronous domino logic pipeline design based on constructed critical data path, *IEEE Trans., VLSI Syst.*, Vol. 23, 2014, pp. 619-630. https://doi.org/10.1109/TVLSI.2014.2314685

[64] K. Tiri and I. Verbauwhede, A digital design flow for secure integrated circuits, *IEEE Trans. Comp.-Aided Des. Int. Circ. Syst.*, Vol. 25, 2006, pp. 1197-1208. https://doi.org/10.1109/TCAD.2005.855939

[65] F. Huemer and A. Steininger, Novel approaches for efficient delay-insensitive communication, *J. Low Pow. Electron. Appl.*, Vol. 9, 2019, Art. no. 16. https://doi.org/10.3390/jlpea9020016

[66] D. Sokolov, J. Murphy, A. Bystrov, and A. Yakovlev, Improving the security of dual-rail circuits, *Proc. Crypt. Hardw. Emb. Syst.*, Springer, Cambridge, 2004, pp. 282-297. https://doi.org/10.1007/978-3-540-28632-5_21

[67] D. Sokolov, J. Murphy, A. Bystrov, and A. Yakovlev, Design and analysis of dual-rail circuits for security applications, *IEEE Trans. Comp.*, Vol. 54, 2005, pp. 449-460. https://doi.org/10.1109/TC.2005.61

[68] C. Cummings, D. Mills, and S. Golson, Asynchronous & synchronous reset design techniques - part deux, 2003 (accessed June 26, 2021). https://trilobyte.com/pdf/CummingsSNUG2003Boston_Resets_rev1_2.pdf

[69] *Intel Corp., Quartus II Subscription Edition Software*, 2011 (accessed June 26, 2021). https://fpgasoftware.intel.com/13.0sp1/?edition=subscription&platform=windows

[70] *Intel Corp., Cyclon II FPGA Starter Development Kit*, 2016 (accessed June 26, 2021).
https://www.intel.cn/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_cii_starter_board.pdf

Appendix 1

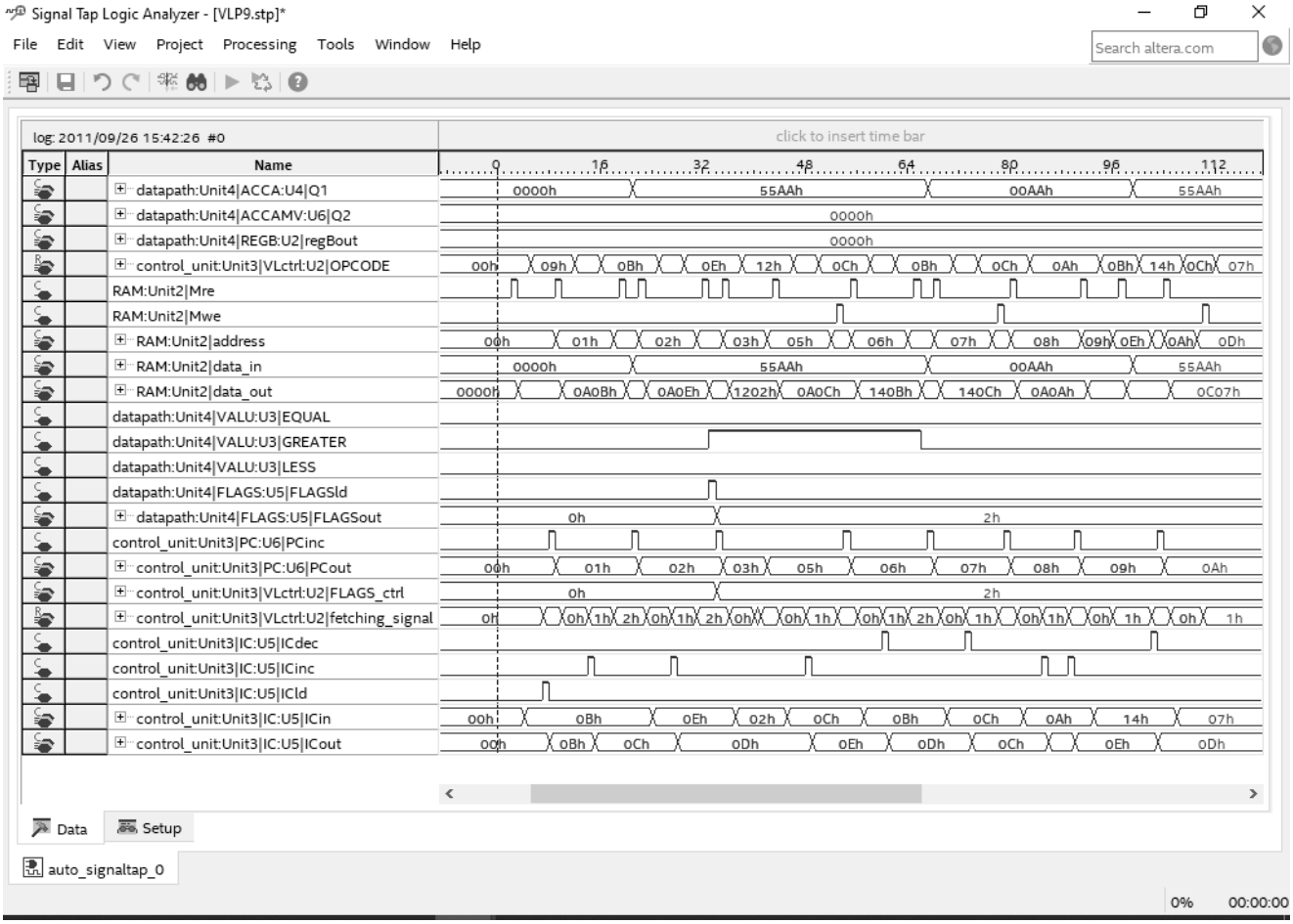


Fig. 12: SignalTap II wave diagrams for the third test program.

**Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)**

-Guennadi A. Kouzaev proposed the initial ideas as well formulation of overarching research goals and aims. He was also responsible for the research activity planning and execution, including mentorship to the core team.

-Atanas N. Kostadinov has implemented the microprocessor and performed testing of the design.

**Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself**

The European Research Consortium for Informatics and Mathematics (ERCIM), Faculty of Information Technology and Electrical Engineering (NTNU), and Department of Electronic Systems (NTNU) supported the initial stage of this research.

**Conflict of Interest**

The authors have no conflicts of interest to declare that are relevant to the content of this article.