Parallel Section RT. IoT, Real Time Systems.

UDC 004.383

Anatoly Sergiyenko, Anastasia Molchanova, Ivan Mozghoviy

METHOD OF MAPPING CYCLO-DYNAMIC DATA FLOWS INTO HARDWARE

The article focuses on the relevance of high-level synthesis (HLS) systems used for designing pipelined datapaths. The goal is to explore methods of mapping algorithms to the pipelined datapaths implementing the cyclic data flow graphs with dynamic schedules. The proposed method involves creating and optimizing cyclo-dynamic data flow graphs, describing them in VHDL. The method demonstrates its effectiveness through examples like run-length encoding decompression and can be implemented in HLS tools.

Keywords: data flow graph, field programable gate array, VHDL, pipeline, dynamic schedule.

Fig.: 3. Tabl.: 1. Bibl.:12.

Introduction. The high-level synthesis (HLS) systems are increasingly distributed by companies producing the CAD tools for integral circuits and FPGAs. They are intended for the computer-aided design of hardware devices that execute algorithms described in a high-level programming language such as C in parallel. Their use makes it possible to speed up the design process tenfold. But HLS still do not provide a decent minimization of hardware costs of synthesized pipelined datapaths in comparison with the manual design. Therefore, it is necessary to search for new methods of mapping algorithms into hardware computing devices. Existing methods of mapping the data flow graphs of various types have found application in the program compilation, but they are rarely used in hardware design.

In this work, the known methods of the pipelined datapath design are considered. These methods analysis make it possible to select the approach to create a method of designing the pipelined datapaths, which is focused on the execution of algorithms based on the cyclic data flow graphs with a dynamic schedule. The proposed method consists in mapping a cyclo-dynamic data flow graph into the datapath with finite state machine (FSMD) which is described in the VHDL language.

Methods of pipelined datapath design and dataflow graphs. A typical method of the datapath design consists in describing the algorithm with a dataflow graph (DFG) and control flow graph, and mapping them into hardware. Such a mapping consists in the sequential execution of three stages: resource selection, operation scheduling, and operation assignment. The synthesis is finished by the

deriving the interconnection scheme and the finite state machine (FSM) design [1]. Since this method synthesizes both the datapath and FSM controlling it, it is often called the FSMD method [2]. But the quality of the resulting device depends significantly on the performance of the stages of synthesis, each of which has a different goal. Although the method is used to implement a large set of algorithms, it is not directly adapted to the synthesis of pipelined datapaths with high throughput.

For the synthesis of pipelined computers, the method of mapping the synchronous data flow graphs (SDF) has become widespread. Such a graph consists of operator nodes and directed edges connecting them. It is considered that the operator in the node is executed immediately (fired for execution) as soon as data (tokens) appear at its inputs, and it outputs the results in the output edges. An edge serves as a dataflow and has a buffer to store the data. Most often, this is a FIFO buffer. It is represented by thick dashes across an edge that correspond to register delays. DFG is synchronous, i.e., SDF, if there is a one-to-one correspondence between the data in the dataflows, for example, they have indices, which depend on the iteration number. Therefore, the execution of the algorithm on SDF has a constant period during which each node consumes and generates the same number of tokens [3]. Because of this, SDF is classified as a statically scheduled dataflow graph (SSDF)[4].

In a single-rate SDF, inputs and outputs of nodes consume and produce the same number of tokens during the calculation period. Therefore, it is not difficult to map a a single-rate SDF into a pipelined datapath that executes a given algorithm with a period of one cycle. By such a mapping, the nodes correspond to the logical circuits that calculate operators and edges do the communication lines, their register delays do the pipeline registers [5]. The representation of SDF in a multidimensional space makes it possible to formally design the pipelined datapaths with a given period of algorithm execution [6]. The SDF use is limited by the set of algorithms which nodes execute the same operations in each period.

The cyclo-static dataflow (CSDF) considers that the actors can have different number of executed tokens in different firings, but the amount of these tokens in a single cycle is stable. Therefore, such a model provides the static schedule [7].

The parametrized SDF (PSDF) is more general, sofisticated, and impressive model of the cyclic algorithms. It considers that the nodes can perform a set of different operations, which can be switched depending on the configuration of tokens in the node inputs. Moreover, the graph can be hierarchical one. But CSDF, SSDF and PSDF are practically used only in the automatic programming but not in the hardware design [4,8].

There is a wide class of cyclic algorithms which could not be represented by SDF, CSDF or SSDF. They distinguished in that the algorithm period depends on the data which it executes. For example it is the compression algorithm, each output code calculation period is variable and strongly depends on the data. Therefore, such an algorithm has the dynamic schedule and the respective computation model is named as cyclo-dynamic dataflow (CDDF) [9]. The hardware implementation of such an algorithm is performed usually by the FSMD method, and therefore, it is complex and often ineffective. Consider the design of a new method of mapping CDDF into the pipelined datapath.

Prerequisites for the method creation. The method is intended for the design of the pipelined datapaths performed in FPGA. So, it must take into consideration the FPGA architecture features like the operation hardware execution and the dataflow performing. On the other hand, the CDDF model arrangement must provide such algorithm shedule which assures both correct hardware implementation and deadlock absence.

The hypothesis is that CDDF can be mapped into the pipelined datapath as the homogeneous SDF can. Such a mapping is possible when a set of conditions is satisfied. Firstly, the necessary conditions of CDDF to be deterministic and free of deadlocks must be satisfied [9]. They are the following.

1) A set of values of the control token, which infer the dynamic behavior, must be limited, this token must be present in the same phase (iteration) in which it is used to determine which phase should be executed, the executed phase must not depend from the value or index of the input datum [7].

2) The graph should not have cycles of dependencies without any delay in the edges [7]. It should be noted that by this condition, there are no dependency cycles in the corresponding data dependency graph and, accordingly, such a graph gives structural solutions without blocking. This is also a condition for the absence of a loop in the corresponding combination scheme, which results the latch [9].

3) All the delays which load the edges must have the initial data or tokens, and this condition is named as the live cycle condition [10].

4) CDDF operates in cycles, each of them executes different number of iterations. The number of consumed and generated tokens by any node in each iteration must be stable as in the homogeneous SDF [7, 10].

5) After execution of a single cycle, CDDF must return in the initial state of this cycle. This assures that CDDF is period safe [10].

6) FSM, which generates the control tokens, must not be deadlocked.

Each CDDF node is mapped into respective logic scheme. And the dataflow represented by an edge weighted by a FIFO delay is mapped into respective register chain or FIFO buffer. When the resuling structure is described by some hardware description language like VHDL the following conditions must be satisfied.

1) The logic scheme which is described in the VHDL language using the process operator, must use the IF-THEN-ELSE and CASE logical operators. And condition that deadlocks do not occur is that the ELSE alternative and all alternative branches of the CASE statement are enabled. The similar features have the WHEN-ELSE and WITH-SELECT operators as well.

2) The dataflow is described in the VHDL language as a process that is triggered by the edge of a common clock signal, in which assignments are made to the signals that mark the FIFO registers.

To describe CDDF and for its convenient perception, each node together with the edges coming out of are described by one process operator. However, a set of such process operators can be combined in a single process operator [11].

For the convenience of compiling the algorithm, the CDDF elements have a symbolic representation, the examples of which are shown in Fig. 1. So, the counter is set using the node of incrementing and register delay as in Fig. 1,a. At the node inputs the enable, initialization control data as well the data stored in the register delay are inputted. The similar counter but with the separate output edge without a delay is shown in Fig. 1, b.

a) b)

Fig. 1. Counter model with output from the register (a) and from the node (b)

If the random access memory (RAM) is the mapping target then the specific subgraph of CDDF is related to it. The Block RAM (BRAM), which is used in FPGA, has a storage as the register set drive, writing data register, write and read address registers, and respective write address decoder and read data multiplexor. The corresponding subgraph has a node for data writing (MW), a storage and a reading data node (MR) (Fig. 2). Nodes MW, MR have the data D and addresses A inputs. The storage itself is

depicted by a long bar. The edge that passes through the storage is also thickened because it characterizes the *n* data buses that are attached to the *n* registers of the storage.

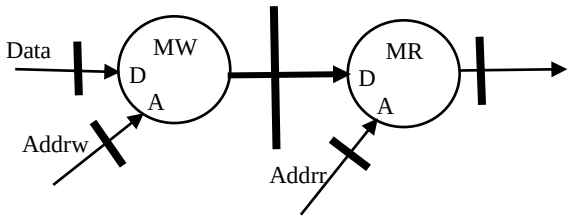

Fig. 2. Subgraph of BRAM

The subgraph of FSM consists of the node which forms the next state (NS) based on the input signal X, the output state (OS) node, which forms the output data Y, and the state register (ST) which loads the edge connecting both nodes (Fig.3).

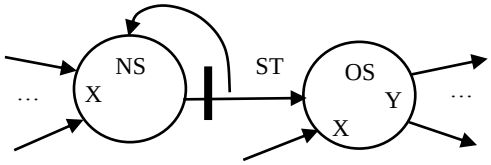

Fig. 3. Subgraph of FSM

CDDF Optimization. When synthesizing pipeline datapath, SDF is optimized by shortening the critical path in the corresponding datapath circuit, and then, optimized SDF is mapped to the datapath. The retiming is recognized as an universal method of optimizing SDF, and it consists in such a permutation of delays in edges that does not disrupt the general execution of the algorithm. The pipelining method is most often used for SDFs, according to which the same number of delays is inserted into the edges, which are directed in the same direction relative to the graph intersection. The pipelining is similar to the retiming in that it shortens the critical path. But at the same time, the latent delay of the algorithm increases [12].

Likewise, CDDF can be optimized using the retiming and pipelining methods.

Method of mapping CDDF to the pipelined datapath. The method is intended for the design of pipelined datapaths, which are configured in FPGAs with a period of one cycle.

Initial data for design are:

— a dataflow algorithm that is represented as CDDF and that can use access to single or multi-port memory, which has the control part like FSM;

— optimality criterion t_C as the minimum period of the clock interval;

— a library of FPGA elements of a certain series, which includes registers, adders, BRAM with specific delays.

Design results: the device description in VHDL or Verilog, which is ready for synthesis and further configuration in FPGA.

1. Representation of the algorithm in the form of CDDF. The functions performed by the logic circuits are represented by the corresponding nodes. The data transfer between the nodes along with the corresponding delays for the required number of clock cycles are represented by edges loaded by the respective delays. The functions of storing data into BRAM and reading these data are represented by the subgraph like one in Fig. 2. The control FSM is represented by the subraph like one in Fig. 3. The derived CDDF must satisfy the necessary conditions to be deterministic and free of deadlocks mentioned above.

2. Optimization of CDDF using pipelining and retiming. The goal of optimization is to minimize the value of t_C. It is equal to the critical path in the CDDF.

3. Mapping the optimized CDDF to the hardware. At the same time, nodes with outputted edges incident to them are described by VHDL language process operators or Verilog language Always constructs. The FSM is described as the separate process. The resulting VHDL or Verilog program is a description of the functional scheme at the level of register transfers of a pipeline computer that executes a given algorithm with a period of one cycle, which is minimized in terms of duration.

The following should be taken into account during the optimization of CDDF. The critical path t_C in CDDF is the maximum path delay. It is determined as a sum of delays in the logic circutits which are relevant to the nodes that belong to the path between two edges loaded by registered delays:

$$t_C = \max \sum_i t_{Pi}, \quad (1)$$

where t_{Pi} is the delay of the i-th node of the P-th type which belongs to the considered path. It should also be noted that in modern FPGAs, the share of delay in the interconnection lines reaches 60-90%, and the delay in logic circuits accounts for 10-40% of the total delay, respectively. Therefore, the final decision to obtain an optimized project should be made after processing the VHDL file with a compiler-synthesizer, placer and router of the FPGA CAD tool.⇒

Synthesis example. The run-length encoding (RLE) decompressor is a simle example of the module which executes the CDDF algorithm. Consider a bit sequence 11100001100. Then, it is encoded by the RLE algorithm as a code sequence 3422. The decompression algorithm must be performed cyclically, i.e., for each input symbol, a sequence of zeros or ones of the appropriate length is generated. Thus, each

cycle has a variable number of iterations that depends on the data. So, the algorithm can be represented by CDDF. The algorithm is written in C language as follows.

```
if (start){
pw = 0;    //pointer to write in a buffer
while (~eos){ //  main cycle finishes by the end-of-sequence signal
                    while (pw ≠ N) { // cycle of writing  codes in a buffer
                                B(pw) = yi;
                                pw++;
                    }
                    pr = 0;  //pointer to read from a buffer
                    fl = 0;   // flag of the bit value
          while (pr ≠ N) { cycle of reading codes from a buffer
                                c = B(pr);
                               while (c ≠ 0){  // iterations of the bit sequence generation
                                          c--;
                                          xj = fl;
                                }
                                fl = ~ fl;
                                pr++;
                    }
                    pr = 0;
}
```

The input codes yi are written to the buffer memory B(pw) at the pointer pw address in the write cycle. The resulting sequence xj is generated in the cycle of reading from the buffer memory codes yi at the pointer pr address. Both cycles are relatively independent and can run in parallel until the eos signal arrives.

When designing the pipelined datapath for computing the sequences of the undefined length, the buffer memory B must be organized as FIFO implemented as a circular buffer. Then, the FIFO depth and code yi tracking period must be such that the buffer memory does not overflow. When FIFO is implemented in BRAM, then its volume is selected as $M = 2^{nb}$, where *nb* is the memory address bit width. Then, the writing pointer state pw must outrun the reading pointer state pr at least to the value of *N*, i.e., $pw - pr \geq N$. Note, that both pointers are incremented modulo *M*.

The corresponding CDDF is shown in Fig.4. The FSM diagram of the decompressor is shown in Fig. 5.

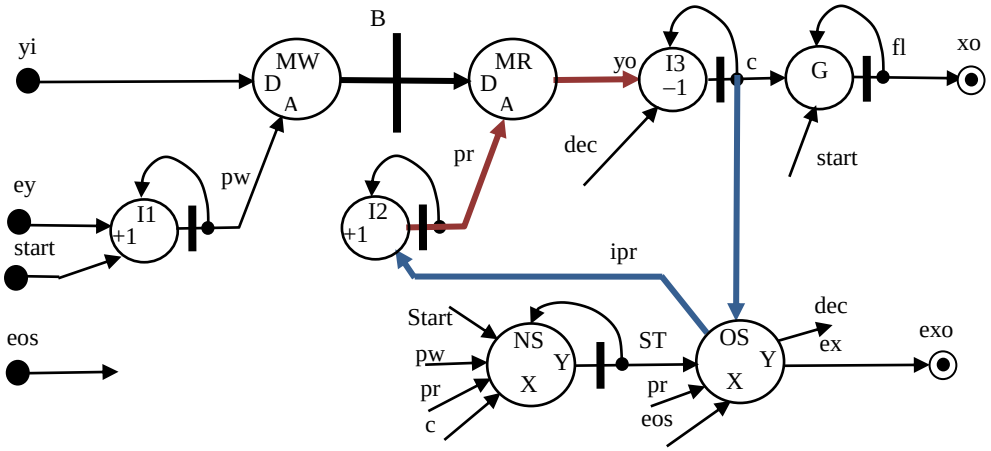

Fig. 4. CDDF of the decompressor

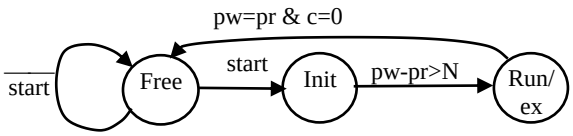

Fig. 5. FSM diagram

Here, the bold points and marked bold points represent the input and output nodes, nodes MW, MR with the storage B form the buffer memory, I1,I2 are the incrementer nodes of the pointers pw and pr, node I3 is the incrementer for deriving code *c* of the running length, node G generates the output sequence xo, nodes NL, OL with the state register ST form FSM. One can see that all conditions of the CDDF correctness are satisfied.

The alternative critical paths in Fig. 4 are shown in bold color lines. So, according to (1) the minimum clock period is equal to

$$t_C = \max((t_{MR} + t_{I3}),\ (t_{OS} + t_{I2})), \qquad (2)$$

where t_{MR}, t_{I3}, t_{OS}, t_{I2} are delays of the logic circuits implemented in the respective nodes.

The FSM diagram (Fig.5) has the idle state Free, initialization state Init and the operation state Run. In the state Init the buffer memory loads N input data. In the state Run along with loading next input data the codes yi are read from the buffer and loaded to the register c. After that the code c is decremented yi times which derives the length of the output sequence of equal bits. The flag trigger fl is inverted each time when the code c is zeroed.

The optimization of CDDF consists in using the methods of retiming and pipelining. The goal is to add the pipelined registers at the inputs and outputs of CDDF and exchange the delay position for getting the subraph as in Fig. 2 which is mapped into BRAM and trying to minimize the critical path.

The resulting optimized CDDF is shown in Fig. 6. Its description in VHDL is the following.

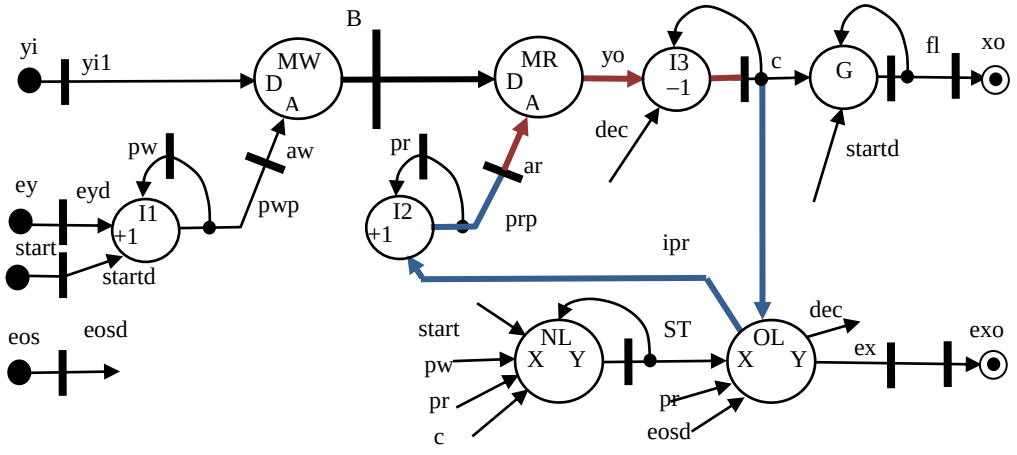

Fig.6. Optimized CDDF

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.Numeric_STD.all;
entity RLDecompressor is generic(nb: natural:= 9; -- address bit width
                N:natural:=256);      -- FIFO deep
        port(
                CLK : in STD_LOGIC;
                RST : in STD_LOGIC;
                EY : in STD_LOGIC;                -- input data enable
                START : in STD_LOGIC; -- start of the input sequence
                EOS : in STD_LOGIC;               -- end of input sequence
                YI : in STD_LOGIC_VECTOR(7 downto 0); -- input sequence
                XO : out STD_LOGIC;               -- output sequence
                EXO : out STD_LOGIC               -- enable of output sequence
                );
end RLDecompressor;
architecture synt of RLDecompressor is
        type ram_type is array (0 to 2**nb-1) of unsigned(7 downto 0);
        signal B : ram_type:=(others=>x"01");
        type state is (Free,Init, Run);
        signal ST: state;
```

```
        signal yi1,yo,c:unsigned(7 downto 0);
        signal pw,aw,pr,ar,pwp,prp:unsigned(nb-1 downto 0);
        signal eosd,startd,ipr,eyd,fl,dec,decd,ex: std_logic;
begin
        REGS:process(CLK,RST) begin  -- register set
                if RST='1' then
                        eyd<= '0';      startd<='0';    eosd<='0';
                        pw<=(others=>'0');  pr<=(others=>'0');
                        fl<='0';        decd<='0';      c<=x"01";  EXO<='0';
                elsif rising_edge(CLK) then
                        pw<=pwp;    pr<=prp;        decd<=dec;
                        eyd<=ey;        eosd<=eos;    EXO<=ex;
                        if dec ='1' and ST=Run then
                                c<= c - 1;
                        elsif c = 0 and ipr='1'  then
                                c<=yo;
                        end if;
                        if STARTd = '1' then
                                fl<='0';
                        elsif c = 0 then
                                fl<= not fl;
                        end if;
                        XO<=fl;
                end if;
        end process;
        I1:     pwp <=(others=>'0') when startd ='1' else     -- node I1
                        pw + 1 when eyd ='1' else pw;
        I2:     prp <=  pr + 1 when ipr ='1' else pr;                -- node I2

        FSM:process(CLK,RST,pw,pr,eosd,c) begin  -- finite state machine
                if RST='1' then
                        ST<=Free;                       -- state register
ex<='0';
                elsif rising_edge(CLK) then
```

```
                if ipr='1' then
                        ex<='1';
                end if;
                case ST is
                        when Free => if START='1' then
                                        ST<=Init;
                                end if;
                        when Init => if pw-pr>N     then
                                        ST<= Run;
                                end if;
                        when Run => if  pr = pw    and c=0 then
                                        ST<=Free;
                                        ex<='0';
                                end if;
                        when others => null;
                end case;
        end if;
        dec<= '0';      ipr<= '0';
        if ST = Run    then
                if (( pr(pr'high) = '1' and pw(pw'high) ='0' and pw + not pr >N) or
                        (pr(pr'high) = '0' and pw-pr>N) or eosd = '1')
                        and c=0  then
                        ipr<= '1';
                elsif c/=0 then
                        dec<='1';
                end if;
        end if;
end process;

RAMB:process (CLK)  begin                       -- circular buffer
        if clk'event and clk = '1' then
                aw<=  pwp;   ar <=   prp;    yi1<= unsigned(YI);
                if eyd ='1' then
                        B(to_integer(aw)) <= yi1;    -- writng data
```

```
                    end if;
                end if;
        end process;
        yo<= B(to_integer(ar));                    -- reading data from the buffer
end synt;
```

The CDDF before the optimization was described in VHDL as well. Both projects were compiled by the Xilinx ISE tool. The results of compiling, placing and routing are shown in Table 1.

Table 1. **Features of the RLE decompressor configured in Xilinx Virtex-7 FPGA**

Decompressor	**Hardware volume**				**Maximum clock frequency, MHz**
	LUTs	**Triggers**	**CLBS**	**18k BRAMs**	
Before optimization	219	37	78	0	358
After optimization	132	53	53	1	332

These results show the following. The decompressor before the optimization has the FIFO buffer implemented on the base of look-up tables (LUTs) which are configured as the distributed RAM. This is the cause of increased hardware volume of the unoptimized decompressor. Such a RAM has less delay in the MR node which provides in 8% higher maximum clock frequency. The optimized decompressor contains one BRAM module, which provides the hardware minimization in 70% in LUTs and in 47% in the configurable logic block slices (CLBS).

So, the use of the proposed method simplifies the design process of the pipelined datapaths, comparing to the most popular method of FSMD design. It optimizes the project both in clock frequency and in hardware volume, providing the use of the specific blocks of FPGA like BRAM.

Conclusions

The graph models of data flow processing algorithms are analyzed and a class of cyclo-dynamic data flow graphs is selected. A method of designing the pipelined datapath that perform cyclic algorithms with a dynamic schedule is proposed. The

results of designing a lossless decompression device using this method are given. The method can be used manually as well as be implemented in the HLS systems.

References

1. Gajski D. D., Abdi S., Gerstlauer A., Schirner G. (2009). *Embedded System Design. Modeling, Synthesis and Verification.* Springer. 352 p.

2. Schaumont P. (2011). *A Practical Introduction to Hardware/Software Codesign*. Springer. 396 p.

3. Lee E. A., Messerschmitt D. G. (1987). *Synchronous data flow*. in Proceedings of the IEEE, vol. 75, no. 9, pp. 1235-1245, Sept. 1987, doi: 10.1109/PROC.1987.13876.

4. Lee E. A, Neuendorffer S. (2005). *Concurrent models of computation for embedded software.* IEE-INST ELEC ENG. IEE Proceedings – Computers and Digital Techniques, Vol. 152. No. 2, pp. 239-250.

5. Khan S. A. (2011). *Digital Design of Signal Processing Systems. A Practical Approach*. UK: Wiley.

6. Sergiyenko A., Serhienko A., Simonenko A. (2017). *A method for synchronous dataflow retiming.* 2017 IEEE First Ukraine Conference on Electrical and Computer Engineering (UKRCON), Kyiv, Ukraine, april 2017, pp. 1015-1018, doi: 10.1109/UKRCON.2017.8100404.

7. Parks T. M., Pino J. L., Lee E. A. (1995). *A comparison of synchronous and cycle-static dataflow*. Conference Record of The Twenty-Ninth Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, USA, pp. 204-210 vol.1, doi: 10.1109/ACSSC.1995.540541.

8. Bhattacharya B., Bhattacharyya S. (2001). *Parameterized dataflow modeling for DSP systems*. IEEE Transactions on Signal Processing. V. 49. No. 10, pp. 2408–2421.

9. Wauters P., Engels M., Lauwereins R., Peperstraete J. A. (1996). *Cyclo-dynamic dataflow*. Proceedings of 4th Euromicro Workshop on Parallel and Distributed Processing, Braga, Portugal, 1996, pp. 319-326, doi: 10.1109/EMPDP.1996.500603.

10. Fradet P., Girault A., Poplavko P. (2012). SPDF: *A schedulable parametric data-flow MoC*. Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 2012, pp. 769-774, doi: 10.1109/DATE.2012.6176572.

11. Сергиенко А. М. (2004). *VHDL для проектирования вычислительных устройств.* Киев: Диасофт. 205 с.

12. Woods R., McAllister J., Lightbody G., Yi Y. FPGA-based Implementation of Signal Processing Systems. Wiley, 2d Ed. 2017, 447 p.

AUTHORS

Anatoly Mykhailovych Sergiyenko, Dr. Tech. Sciences, Senior Scientist, professor of the Computer Engineering Department of the National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.

E-mail: asera@comsys.kpi.ua

Molchanova Anastasia Anatoliivna, assistant lecturer of the Computer Engineering Department of the National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.

Ivan Mozghoviy, postgraduate of Computer Engineering Department of National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.

E-mail: mozg.v34@gmail.com

РОЗШИРЕНА АНОТАЦІЯ

А. М. Сергієнко, А. А. Молчанова, І. В. Мозговий

МЕТОД ВІДОБРАЖЕННЯ В АПАРАТУРУ АЛГОРИТМІВ ОБРОБЛЕННЯ ЦИКЛІЧНИХ ПОТОКІВ ДАНИХ З ДИНАМІЧНИМ РОЗКЛАДОМ

Актуальність теми дослідження. Системи високорівневого синтезу (СВС) все більше поширюються фірмами-виробниками засобів САПР мікросхем та програмовних логічних інтегральних схем (ПЛІС). Вони призначені для автоматизованого проєктування апаратних пристроїв, які паралельно виконують алгоритми, що описані високорівневою мовою програмування такою як С. Їх використання дає змогу у десятки разів прискорити процес проєктування.

Постановка проблеми. СВС досі не забезпечують гідної мінімізації апаратних витрат синтезованих конвеєрних обчислювачів у порівнянні з ручним проєктуванням. Тому необхідний пошук нових методів відображення алгоритмів у апаратні обчислювальні засоби. Існуючі методи відображення графів потоків даних різних видів знайшли застосування у автоматизованому програмуванні, але вони рідко використовуються у проєктуванні апаратури.

Аналіз останніх досліджень і публікацій. Найбільш поширеним тепер методом проєктування апаратних пристроїв для обробки потоків даних є метод синтезу блоку обробки даних з керуючим автоматом, який має труднощі з автоматизацією через низький рівень формалізації. Існуючі формалізовані методи синтезу конвеєрних обчислювачів орієнтовані на вузький клас алгоритмів обробки синхронних потоків даних зі статичним розкладом.

Виділення недосліджених частин загальної проблеми. Є широкий клас алгоритмів, які представляються графами циклічних потоків даних з динамічним розкладом, які ще недостатньо досліджені з боку удосконалення їх відображення у конвеєрні апаратні засоби і впровадження у СВС.

Постановка завдання. Завданням є створити метод проєктування конвеєрних обчислювачів, які орієнтовані на виконання алгоритмів оброблення циклічних потоків даних з динамічним розкладом.

Викладення основного матеріалу. Новий метод полягає у створенні графу циклічних потоків даних з динамічним розкладом, його оптимізації та описі мовою VHDL стилем для синтезу. Приклад синтезу декомпресора

кодування довжини рядків повторюваних символів показує дієвість та ефективність методу. Також показані результати синтезу декомпресора за алгоритмом LZW та їх порівняння з іншими пристроями.

Висновки. Проаналізовані існуючі графові моделі представлення алгоритмів оброблення потоків даних і виділено клас графів циклічних потоків даних з динамічним розкладом. Запропоновано метод проєктування конвеєрних обчислювачів, що виконують циклічні алгоритми з динамічним розкладом. Наведені результати проєктування за допомогою цього метода пристрою для безвтратної декомпресії. Метод може бути впроваджений у СВС.

Ключові слова: граф потоків даних, програмовна логічна інтегральна схема, VHDL, конвеєр, динамічний розклад.

Relevance of the research topic. Systems of high-level synthesis (HLS) are increasingly distributed by companies producing CAD tools for integral circuits and FPGAs. They are intended for the computer-aided design of hardware devices that execute algorithms described in a high-level programming language such as C in parallel. Their use makes it possible to speed up the design process tenfold.

Formulation of the problem. HLS still do not provide a decent minimization of hardware costs of synthesized pipeline computers in comparison with manual design. Therefore, it is necessary to search for new methods of mapping algorithms to hardware computing devices. Existing methods of displaying data flow graphs of various types have found application in automated programming, but they are rarely used in hardware design.

Analysis of recent research and publications. The most common method of designing hardware devices for data flow processing is the method of synthesizing a datapath with FSM, which has difficulties with its implementation due to a low level of its formalization. The existing formalized methods of synthesis of pipelined datapaths are focused on a narrow class of algorithms for processing synchronous data flows with a static schedule.

Highlighting unexplored parts of the general problem. There is a wide class of algorithms, which are represented by the cyclo-dynamic data flow graphs, which have not yet been sufficiently explored in terms of improving their mapping in the pipelined datapathse and implementation them in HLS.

Setting objectives. The task is to create a method of designing the pipelined datapaths, which are focused on the execution of algorithms for processing the cyclic data flow graphs with a dynamic schedule.

Presentation of the main material. The new method consists in creating a cyclo-dynamic data flow graph, optimizing it and describing it in the VHDL language in a synthesis style. An example of the synthesis of a run-length encoding decompressor shows the effectiveness of the method. The results of decompressor synthesis using the LZW algorithm and their comparison with other devices are also shown.

Conclusions. The graph models of data flow processing algorithms are analyzed and a class of cyclo-dynamic data flow graphs is selected. A method of designing the pipelined datapath that perform cyclic algorithms with a dynamic schedule is proposed. The results of designing a lossless decompression device using this method are given. The method can be implemented in the HLS tools.

Keywords: data flow graph, field programable gate array, VHDL, pipeline, dynamic schedule.