UDC 004.383 DOI: 10.18372/2073-4751.84.20899

Molchanov O. A., PhD.
orcid.org/0000-0001-8384-0918
oleksii.molchanov@gmail.com
Sergiyenko A. M., D.Sci.,
orcid.org/0000-0001-5965-1789
aser@comsys.kpi.ua
Vinokurov A. I.
orcid.org/0009-0006-7861-5138
neyv2015@gmail.com

MAPPING CYCLO-DYNAMIC DATAFLOW INTO PIPELINED DATAPATH

National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

Introduction

The high-level synthesis (HLS) systems are distributed by companies producing CAD tools for integral circuits and FPGAs. They are intended for the 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. However, HLS still does not provide a decent minimization of hardware costs of synthesized pipelined datapaths compared to 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.

This work considers the known methods of the pipelined datapath design. Through these methods of analysis, it is possible to select a method for designing pipelined datapaths based on cyclic data flow graphs with dynamic scheduling that focuses on algorithm execution. The proposed method consists of mapping a cyclo-dynamic data flow graph into the datapath with a finite state machine (FSMD) which is described in the VHDL language.

Problem analysis

Typically, the datapath design involves describing the algorithm with a dataflow graph (DFG) and control flow graph, and mapping them to hardware. Three stages are sequentially executed: resource selection, operation scheduling, and operation assignment in such a mapping. Then, finite state machine (FSM) is derived along with the interconnection scheme. Because both the datapath and FSM are synthesized, this method is commonly referred to as the FSMD method [1]. However, the quality of the resulting device is strongly influenced by the efficiency of the various stages of synthesis, each of which is focused on a specific objective. A large set of algorithms can be implemented using this method; but it cannot be used for synthesis of pipelined datapaths that have high throughput.

The method of mapping the synchronous data flow graphs (SDF) has become widespread for the synthesis of pipelined computers [2]. Such a graph consists of operator nodes and directed edges connecting them. It is considered that the operator in the node is executed immediately as soon as data (tokens) appear at its inputs, and it outputs the results in the output edges. An edge serves as a data flow and has a FIFO buffer to store the data. It is represented by thick dashes across the edge, that correspond to register delays. SDF is synchronous, because there is a one-to-one correspondence between the data in the data flows, they usually have indices,

which depend on the iteration number. 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 single-rate SDF into a pipelined datapath that executes an algorithm with a period of one cycle. Then, 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 in which nodes execute the same operations in each period.

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

The parametrized SDF (PSDF) is a more general and sophisticated 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. But CSDF, SSDF, and PSDF are practically used only in the automatic programming [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. Such an algorithm has a dynamic schedule and the respective computation model is named 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.

Objective of research

The objective is to design of a new method of mapping CDDF into the pipelined datapath.

Method of mapping cyclo-dynamic dataflow

This method is intended for the design of pipelined datapaths in FPGAs. It must consider the FPGA architecture features. In contrast, a CDDF model arrangement must ensure that both correct hardware implementation and deadlock absence are ensured.

In our view, CDDF can be mapped into the pipelined datapath as well as 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]. By this condition, there are no dependency cycles in the data dependency graph and, accordingly, such a graph gives structural solutions without blocking. This is a condition for the absence of a latch in the resulting combination scheme [9].

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

4) CDDF operates in cycles, each of them executes a 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 to the initial state of this cycle. This assures that CDDF is period safe [10].

6) FSM, which generates the control tokens, must be determined and free of deadlocks.

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

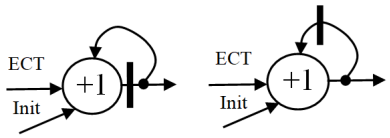
1) The logic scheme which is described in the VHDL language using the process operator, must use the IF-THEN-ELSE and CASE logical operators. And condition that deadlocks do not occur is that the ELSE alternative and all alternative branches of the CASE statement are enabled. The similar features have the WHEN-ELSE and WITH-SELECT operators as well [11].

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 [11].

To describe CDDF, each node together with the edges coming out of it are described by one process operator. However, a set of such operators can be combined into a single process operator [12].

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 as the data stored in the register delay are inputted. A similar counter but with a separate output edge without a delay is shown in Fig. 1, b.

The Block RAM (BRAM), which is used in FPGA, has storage as the register set, 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), storage, and reading data node (MR) (Fig. 2). Nodes MW and MR have the data D and address A inputs. The storage itself is depicted by a long bar.



a) b)

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

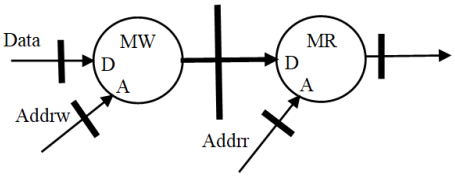


Fig. 2. Subgraph of BRAM

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

The subgraph of FSM consists of the node that 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).

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 procedure is recognized as a universal method of optimizing SDF, and it consists of 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. However, the latent delay of the algorithm increases [13].

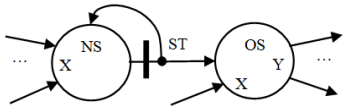


Fig. 3. Subgraph of FSM

Likewise, CDDF can be optimized using the retiming and pipelining methods.The proposed method is intended for the design of pipelined datapaths, which are configured in FPGAs and execute the algorithm using a period of one clock 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;

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

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

Design results are the device descripttion 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 the one in Fig. 2. The control FSM is represented by the subgraph like the 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 CDDF.

3. Mapping 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 a 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 circuits which are relevant to the nodes that belong to the path between two edges loaded by registered delays:

$$t_C = \max , \qquad (1)$$

where t_{Pi} is the delay of the i-th node of the P-th type which belongs to the 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 examples

The run-length encoding (RLE) decompressor is simple example of the module which executes the CDDF algorithm. The RLE algorithm is used to compress data, in this case a bit sequence, by representing repetitive consecutive bits with a code sequence. In a specific example, the bit sequence "11100001100" is encoded as "3422" using RLE.

To decompress the encoded data and retrieve the original bit sequence, the reverse process is performed. Fig. 4. shows a decompression algorithm. The

decompression algorithm operates cyclically, meaning that for each symbol in the input code sequence, a sequence of zeros or ones is generated based on the appropriate lengths. The number of iterations in each cycle can vary depending on the data being decompressed.

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. Note, that the pointers *pw, pr* count modulo *M*.

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

Fig. 4. RLE decompression algorithm

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 $M = 2n_b$, where n_b 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., $pw - pr \geq N$.

The corresponding CDDF is shown in Fig.5. The FSM diagram of the decompressor is shown in Fig. 6. 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 incrementing nodes of the pointers *pw* and *pr*, node I3 is incrementing for deriving code *c*, node G generates the output sequence *xo*, nodes NL, OL with the state register ST form FSM. The output sequence *xo* is synchronized by the impulses *exo* generated by FSM. Note, that all conditions of the CDDF correctness are satisfied.

The alternative critical paths in Fig. 5 are shown in bold colored lines. So, according to (1) the minimum clock period is equal to

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

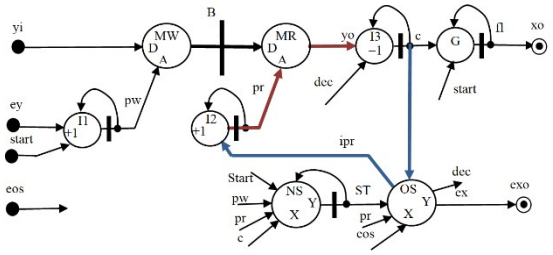


Fig. 5. CDDF of the decompressor

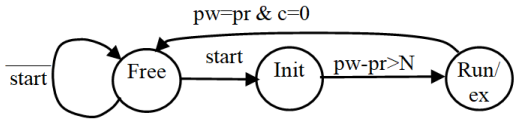


Fig. 6. FSM diagram

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

The FSM diagram (Fig.6) has the idle state Free, the 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 the next input data the codes *yi* are read from the buffer and loaded to the register *c*. After that, the code *c* is decremented y_i 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 CDDF optimization consists of 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 subgraph as in Fig. 2 which is mapped into BRAM and trying to minimize the critical path.

The resulting optimized CDDF is shown in Fig. 7. The CDDF before and after optimization was described in VHDL. Both projects were compiled by the Xilinx Vivado tool. The results of compiling, placing, and routing are shown in Table I.

The decompressor before the optimization has the FIFO buffer implemented on the look-up tables (LUTs). This increased hardware volume of the non-optimized decompressor. Such a RAM has less delay in the MR node providing an 8% higher clock frequency.

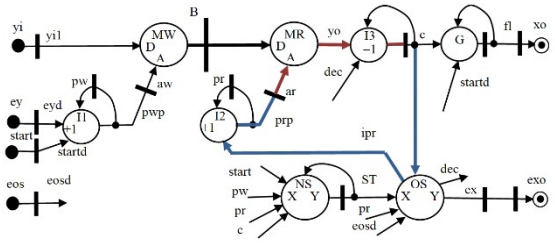


Fig. 7. Optimized CDDF

Table 1. Features of the RLE decompressor configured in AMD-Xilinx Kintex-7 FPGA

Deco m-pressor	Hardware volume				Max. clock freq. MHz
	LUTs	FFs	CLBS	18k BRAM	
Before optimi-zation	219	37	78	0	358
After optimi-zation	132	53	53	1	332

The optimized decompressor contains one BRAM module, which supports hardware minimization in 70% of LUTs and 47% in the configurable logic block slices (CLBS).

Consider an example of the module design that performs decompression using the LZW algorithm. The paper [14] illustrates both the LZW algorithm and its execution by hardware-software approach in detail. The LZW algorithm is represented by CDDF. It has to get from the dictionary the output symbols in sequence for each input code, which is variable depending on the input code.

The results of the synthesis of the decompressor are presented in Table 2. The results of the synthesis of similar devices are given there for comparison. These devices are built using the traditional method of design and inventions of their authors. The proposed design is implemented in Xilinx Kintex-7 FPGA, and the analogous devices are in Xilinx Virtex-7 series. It should be taken into account that the Virtex-7 FPGAs have slightly better timing characteristics than Kintex-7, however, they are manufactured using the same architecture and technology.

Table 2. Features of the LZW decompressor configured in AMD-Xilinx FPGA

Deco m-pressor	Hardware volume			Max. clock freq., MHz	Through-put, MB/s
	LUTs	FFs	18k BRAM		
Propo-sed	154	162	7	360	205
Zhou [15]	307	278	13	301	200
Kaga-wa [16]	213	224	13	296	295

Table 2 analysis shows that the de-compressor described in [16] has the highest throughput due to excessive BRAM memory volumes. The decompressor developed according to the proposed method has a 21% higher clock frequency and 28% lower hardware costs,

as well as a 46% smaller memory volume compared to the device [16], which has the same dictionary volume, and bit sizes of input and output data.

It should be noted that BRAM memory is a valuable resource, because one block of BRAM in an AMD-Xilinx FPGA, on average, accounts for 800 LUTs. Therefore, the developed decompressor is certainly profitable in terms of hardware costs. The received throughput achieves 205 decompressed megabytes per second with an average compression ratio of two times.

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

Conclusion

A method of designing the pipelined datapath that performs cyclic algorithms with a dynamic schedule is proposed. The method is based on a class of cyclo-dynamic data flow graphs. 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. The next investigation steps will be devoted to probing this method for more complex algorithm mapping into FPGA and to designing an automatic framework for the pipelined datapath development. This method can be adapted to programming the parallel computer systems as well.

References

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

2. Schaumont P. A Practical Introduction to Hardware/Software Codesign. *Springer*. —2011.

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

4. Lee E. A., Neuendorffer S. Concurrent models of computation for embedded software. *IEE-INST ELEC ENG. IEE Proceedings ~ Computers and Digital Techniques,* vol. 152. No. 2, —2005, —pp. 239–250. https://doi.org/10.1049/ip-cdt:20045065

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

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

7. Parks T. M., Pino J. L., Lee E. A. A comparison of synchronous and cycle-static dataflow. *29th Asilomar Conference on Signals, Systems and Computers,* Pacific Grove, CA, USA, vol. 1, —1995. —pp. 204–210 https://doi.org/10.1109/ACSSC.1995.540541

8. Bhattacharyya B., Bhattacharyya S. Parameterized dataflow modeling for DSP systems. *IEEE Transactions on Signal Processing,* vol. 49, no. 10, 2001, pp. 2408–2421. https://doi.org/10.1109/78.950795

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

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

11. Keating M., Brikaud P. Reuse Methodology Manual for System-on-a-Chip Designs, 3d Ed. *Kluwer.* —2007.

12. Sergiyenko A. M. HDL dlya projectirovanija vychislitelnych ustroystv. Kyiv: *Diasoft.* 2004. (In Russian).

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

14. Romankevych V. O., Mozghovyi I. V., Serhiienko P. A. Zacharioudakis L. Decompressor for hardware applications. *Applied Aspects of Information Technology.* Vol.6, No.1. —2023, —pp. 74–83. https://doi.org/10.15276/aait.06.2023.6

15. Zhou X., Ito Y., Nakano K. An Efficient Implementation of LZW Decompression in the FPGA. *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. Chicago, IL, USA, 2016. pp. 599607, https://doi.org/10.1109/IPDPSW.2016.33.

16. Kagawa H., Ito Y., Nakano K. Throughput-Optimal Hardware Implementation of LZW Decompression on the FPGA. *2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW),* Nagasaki, Japan, —2019. —pp. 7883. https://doi.org/10.1109/ CANDARW.2019.00022.

Молчанов О. А., **Сергієнко А. М.**, **Вінокуров А. І.**

ВІДОБРАЖЕННЯ ЦИКЛОДИНАМІЧНОГО ГРАФУ ПОТОКІВ ДАНИХ У КОНВЕЙЕРНИЙ ОБЧИСЛЮВАЛЬНИЙ БЛОК

У статті обговорюється актуальність систем високорівневого синтезу для проектування конвеєрних обчислювальних блоків. Метою дослідження є методи відображення алгоритмів у конвеєрний блок обробки даних, який реалізує алгоритми, що задаються циклічним графом потоків даних, що має динамічний розклад. Запропонований метод включає створення та оптимізацію циклодинамічного графу потоків даних (ЦДГПД) та його опис мовою VHDL. Запропоновано набір правил для опису коректного ЦДГПД та відповідного керуючого автомата. Отриманий ЦДГПД відображається у структуру конвеєрного обчислювального блоку за правилами, які є такими самими, як при відображенні графа синхронний потоків даних. Продемонстровано позитивну ефективність методу на прикладах проєктування пристроїв декомпресії файлів з кодуванням довжини ланцюжків та за алгоритмом LZW, які реалізовані в програмовних логічних інтегральних схемах. Запропонований метод можна використовувати вручну або реалізувати в САПР високорівневого синтезу інтегральних схем.

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

Molchanov O. A., **Sergiyenko A. M., Vinokurov A. I.**

MAPPING CYCLO-DYNAMIC DATAFLOW INTO PIPELINED DATAPATH

The paper discusses the relevance of high-level synthesis (HLS) systems in designing pipelined datapaths. The goal of the research is to explore methods for mapping algorithms into a pipelined datapath that implements cyclic data flow graphs with dynamic schedules. The proposed method involves creating and optimizing cyclo-dynamic data flow graphs (CDDFG) and describing them in VHDL. A set of rules for arranging the correct CDDFG and respective finite state machine is proposed. The derived CDDFG is mapped into the pipelined datapath, as well as the synchronous dataflow. The method demonstrates its effectiveness through examples of run-length encoding decompression and LZW file decompression devices, which are implemented in field programmable gate arrays. The proposed method can be used manually or be implemented in HLS tools.

***Keywords:** data flow graph, FPGA, VHDL, datapath, pipeline, dynamic schedule.*

Стаття подана до редакції: 30/11/2025
Стаття прийнята до опублікування:14/12/2025
Стаття опублікована: 30/12/2025