A DFG PROCESSOR IMPLEMENTATION FOR DIGITAL SIGNAL PROCESSING APPLICATIONS

Ali Shatnawi, Osama Al-Khaleel and Hala Alzoubi

Department of Computer Engineering, Jordan University of Science and Technology, Irbid, Jordan

ABSTRACT

This paper proposes a new scheduling technique for digital signal processing (DSP) applications represented by data flow graphs (DFGs). Hardware implementation in the form of a specialized embedded system, is proposed. The scheduling technique achieves the optimal schedule of a given DFG at design time. The optimality criterion targeted in the proposed algorithm is the maximum throughput than can be achieved by the available hardware resources. Each task is presented in a form of an instruction to be executed on the available hardware. The architecture is composed of one or multiple homogeneous pipelined processing elements, designed to achieve the maximum possible sampling rate for several DSP applications. In this paper, we present a processor implementation of the proposed architecture. It comprises one processing element where all tasks are processed sequentially. The hardware components are built on an FPGA chip using Verilog HDL. The architecture requires a very small area size, which is represented by the number of slice registers and the number of slice lookup tables (LUTs). The proposed scheduling technique is shown to outperform the retiming technique, which is proposed in the literature, by 19.3%.

KEYWORDS

Data Flow Graphs, Task Scheduling, Processor Design, Hardware Description Language

1. INTRODUCTION

Digital signal processing (DSP), image processing, and communication tasks are computationally intensive. Hence, they demand very high-speed and high-throughput computing systems. Multiprocessing is normally very suitable for implementing DSP applications, since they are inherently parallel [1].

On the other hand, logic synthesis [2] is the process of designing digital systems that are described in HDL to get optimized hardware implementations. Logic synthesis uses standard cell libraries to produce the required design; these libraries include the basic logic gates and the macro cells (adder, multiplexer, memory and flip-flop). HDL description can be synthesized targeting different chip technologies, such as FPGA or ASIC.

DSP applications are commonly represented using signal flow graphs and DSP block diagrams [3]. A DFG representation of these applications can be easily obtained from the previous representation models. In a DFG model, operations of a DSP application are represented by nodes and the data dependencies (signal paths) between nodes are represented by arcs (edges) [4].
In this paper, we propose a new technique for scheduling DSP applications represented by DFGs. The schedule achieves the maximum throughput of a DFG for the available hardware resources. This technique is composed of two parts: task analysis of the DFG and hardware assignment of

Dhinaharan Nagamalai et al. (Eds) : WIMO, ICAIT, ICDIPV, NC - 2019
pp. 13-22, 2019. © CS & IT-CSCP 2019 DOI: 10.5121/csit.2019.90402

the tasks to achieve the desired algorithmic behavior. The two parts are combined to form a complete system.

The paper organization is as follows: Section 2 presents some related work, Section 3 presents scheduling and hardware allocation, Section 4 presents the proposed scheduling algorithm, Section 5 presents an example to demonstrate the application of the algorithm, Section 6 presents hardware implementation of the proposed systems, Section 7 presents results, and Section 8 concludes the paper.

2. RELATED WORK

Many algorithms for achieving optimal static schedules for task graphs modeled by DFG into multiprocessing systems, were presented in the literature.

In [5], the algorithm introduces an optimal scheduling technique of a cyclic data flow graph onto a homogeneous multiprocessing system. It depends on fixing the iteration period and keeping the number of processors variable until a solution is attained. The tasks are assigned to a set of homogeneous processing elements according to their computational delays. The optimality criteria considered include maximum throughput, minimum delay and minimum hardware resources.

There are some other techniques that were presented in the literature. An Incremental Subgraph Earliest Finish Time (INCSEFT) strategy is proposed in [6].

In [7], a scheduling algorithm for the task graph in multi-computing heterogeneous computing system is proposed.

There are scheduling algorithms that attempt to reach better performance in scheduling DSP applications, by shortening the execution time and achieving parallel scheduling, such as the one in [8].

The work in [9] proposes a list-based scheduling technique based on a heterogeneous architecture and referred to as ``Improved Predict Earliest Finish Time (IPEFT)".

In [10], a scheduling of parallel tasks in heterogeneous platform is proposed and in [11], simple DSP applications presented by DFGs are directly mapped into FPGA.

3. SCHEDULING AND HARDWARE ALLOCATION

3.1. Scheduling

In this process, each operation in a DFG is assigned to a sequence of control steps. These control steps represent clock cycles in a synchronous system. A scheduling process must preserve precedence constraints between tasks. A program presented by a graph may be executed once or repeated many times. The target of the scheduling algorithm is either to minimize the iteration time to meet the speed requirements or the implementation area to meet the resource constraints.
Precedence constraints should be met to preserve the data integrity of the algorithm specifications. There are two types of precedence constraints: intra-iteration precedence constraints and inter-iteration precedence constraints [12]. The intra-iteration precedence constraints are represented by nodes connected with edges that have no ideal delays [12]. (One ideal delay is represented in the Z-transform domain by Z^{-1}.) The inter-iteration precedence constraints are represented by nodes that are connected with edges having non-zero ideal delays.

3.2 Hardware Allocation

In the hardware allocation phase, function units are assigned to execute the operations represented by the nodes of the graph, storage units (registers) are used to save intermediate computational results, and the interconnection network is used for data movement. The goal of the hardware allocation technique is to reduce the number of processing elements, memories, registers, multiplexers, and wires [13].

There are two types of hardware architectures based on the functional units used: homogeneous, when all the functional units are identical and can perform all operations, and heterogeneous, when not all functional units are identical [14]. The homogeneous architectures have higher flexibility and scalability than heterogeneous architectures. On the other hand, the heterogeneous architectures are more resource-efficient and normally require less area than homogeneous ones.

4. SCHEDULING ALGORITHM

A new scheduling technique for DSP applications represented by DFGs is proposed. It produces a schedule with maximum throughput using the available hardware resources. This technique is composed of two parts: task analysis of the DFG operations and hardware assignment of these tasks to achieve the algorithmic behavior. The two parts are combined to form a complete system. In the first part, the DFG nodes are ordered in a node queue according to their inter-related data dependencies. This queue is then used to create a compound task queue. Each compound task is created by combing two nodes together into one task. These tasks are expressed in instruction format to be used in a hardware system. The second part of the scheduling technique is the allocation of the tasks of the DFG on a general pipelined hardware architecture. This system stores the DFG operations as instructions in the system memory. These operations are stored in a sequence ready for execution, as many iterations as needed. The proposed system has one processing element where all tasks are processed sequentially. The first step in the scheduling algorithm is the cyclic to acyclic conversion.

CYCLIC TO ACYCLIC CONVERSION

The first step in constructing the queue is to convert a cyclic DFG to an acyclic graph, DAG by keeping the edges representing the intra-iteration dependencies and removing all other ones. The resulting graph will definitely be an acyclic one.

NODE QUEUE CONSTRUCTION

Once a cyclic DFG is converted into a DAG, the nodes of a DAG are arranged into a queue depending on the data dependencies between the graph nodes as depicted in Fig. 1.

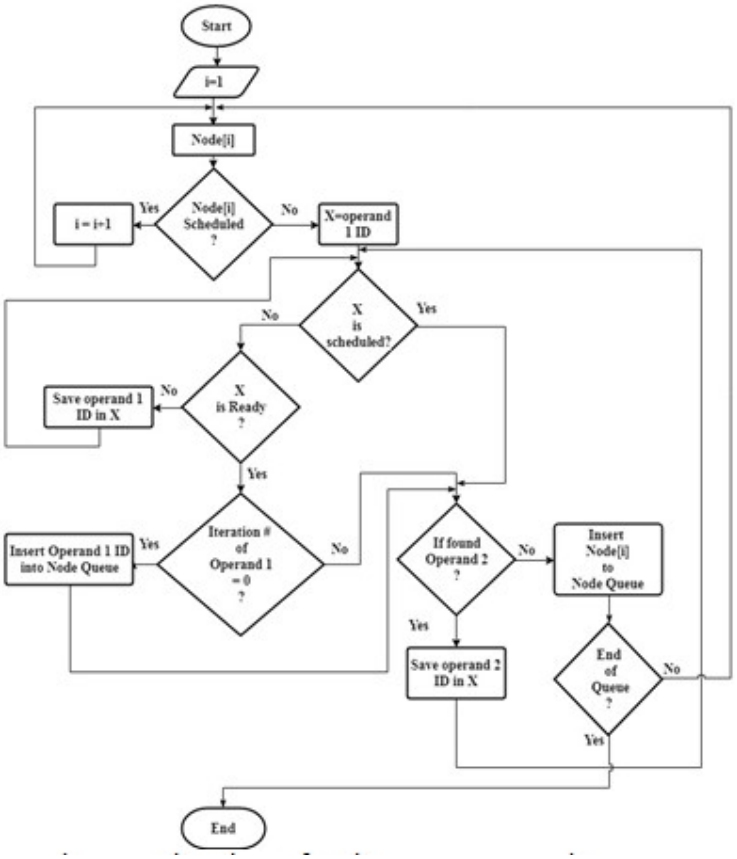

Figure 1: Flowchart of node queue construction

TASK CREATION

To minimize the number of nodes in a DFG, some pairs of nodes are combined into compound nodes. The multiplication accumulation operation (MAC) is derived from merging a multiplication node with the following addition node into one node. The execution of a compound node would require less number of cycles for execution than the sum of cycles needed to execute the two nodes separately.

COMPOUND TASK QUEUE CONSTRUCTION

Some development processes are applied at this stage to improve the throughput of the system. A task creation process is applied to the node queue until the task queue is produced. If there exists a multiplication node followed by an addition node in the node queue, these nodes are combined into a compound task. This algorithm works as shown in Fig. 2.

5. EXAMPLE

For the sake of keeping this paper concise and suitable for a conference paper, only one benchmark is presented, the DSP filter known as All-Pole Lattice Filter.

The DFG of the all-pole lattice filter is shown in Fig. 3. The node queue shown in Table 1 results from the first part of the scheduling phase. It shows the node order due to data dependencies

among them. According to the technique in Fig 1, the DFG nodes are ordered into a node queue. Node ordering starts from node 1, which is inserted directly to the node queue because its operands are ready from the previous iteration. The rest of the nodes 2 to 15 are then placed in the queue in order, since each node will have its operands ready upon executing all preceding nodes in the queue.

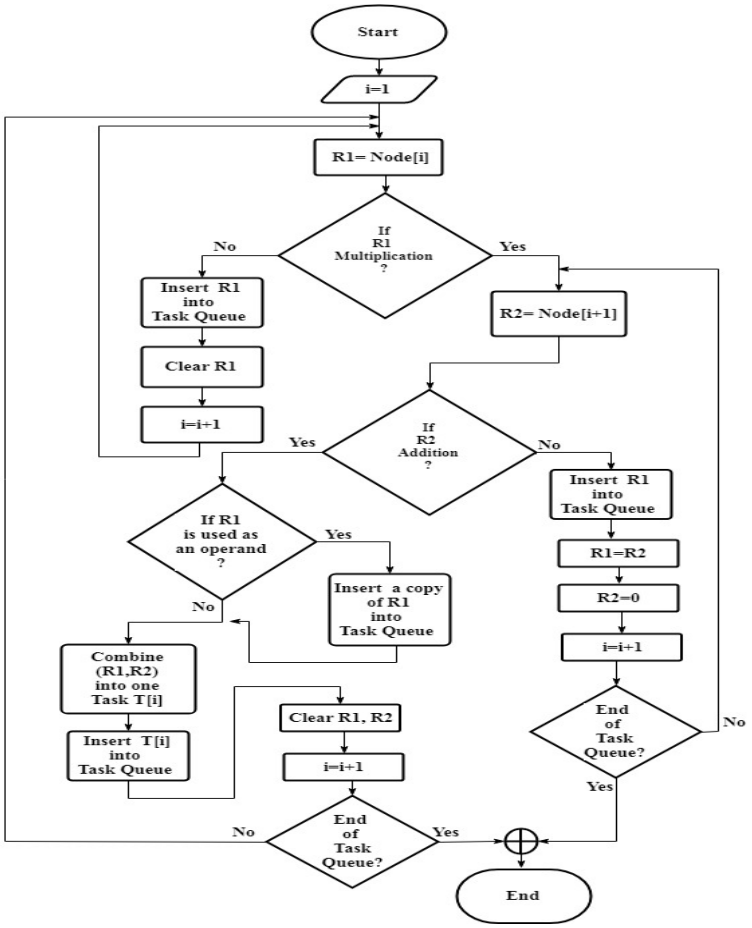

Figure 2: Flowchart of compound task queue construction.

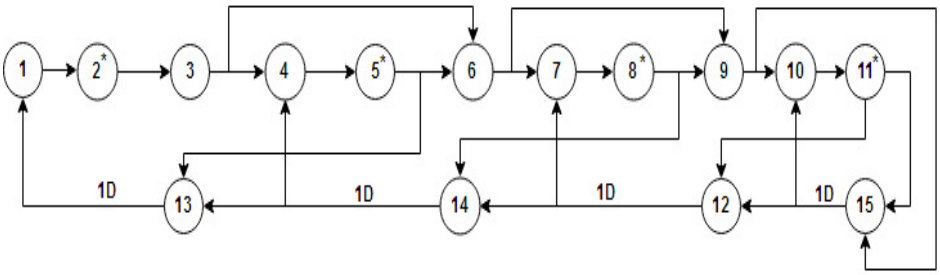

Figure 3: The DFG of the all-pole lattice filter [15]

Table 1: The node queue of the all-pole lattice filter.

Node ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Some nodes from the node list are combined to create compound tasks; thus resulting in the list shown in Table 2, which lists the task ID and the corresponding node representing it. As shown from the task queue in Table 2, when merging nodes into a compound tasks, its operation should be multiplication followed by addition.

Table 2: The Compound task queue of the all-pole lattice filter.

Task ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Nodes	1	2&3	4	5	5&6	7	8	8&9	10	11	11&12	13	14	15

As in the case of task 2, it is composed of the multiplication (node 2) and addition (node 3). Similarly, each pair of the nodes 5 and 6, 8 and 9, 11 and 12 is combined into a compound task. In case of nodes 5, 8 and 11, each is duplicated to appear as a stand-alone task and a task to be combined with a succeeding task. Nodes 13, 14 and 15 are inserted into the task queue as single operation tasks.

6. HARDWARE IMPLEMENTATION

The proposed system processes the data sequentially using a single pipelined processor. A parallel implementation of this system will appear in the extended paper of this manuscript.
Recall that the proposed system is dedicated for executing applications that are represented by DFGs. The hardware implementation of the system has been carried out using Verilog HDL and targeting different Xilinx FPGAs. The pipeline of the proposed system consists of four stages: Instruction fetch, instruction decode, execution, and write back.

Instruction Format

Once the nodes in a DFG are arranged into a task queue, the queue is loaded into the main memory of the system. Each task is represented in the form of an instruction. The instruction has a width of 41 bits. It describes and contains all the data needed for execution. The instruction consists of eight different fields as follows:

- Task Identifier (6 bits).
- Operand1 identifier: represents the ID of the task that produces operand1 (6 bits).
- Operand2 identifier: represents the ID of the task that produces operand2 (6 bits).
- Operand1 iteration number: the iteration index of operand1 (2 bits).
- Operand2 iteration number: the iteration index of operand2 (2 bits).
- Factor: immediate data used as a multiplication factor (16 bits).
- Operation code: represents the operation in the task: addition, multiplication or add-multiply (2 bits).
- Last_task bit, which is set to one in the last task of the task queue of the DFG.

Fig. 4 shows the instruction format of a DFG tasks.

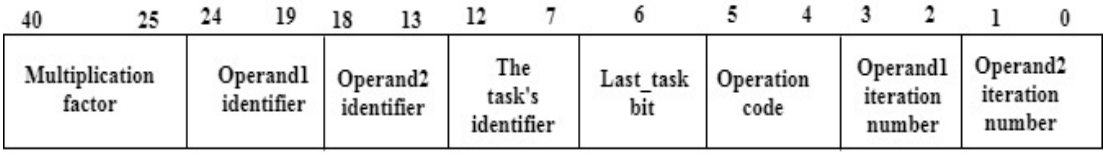

Figure 4: The instruction format of the DFG tasks.

Pipeline Stages

The pipeline in the proposed system consists of four stages:

- IF (Instruction Fetch): an instruction is read from the main memory at a specific address that is generated by an address generator unit.
- ID (Instruction Decode): operands of the task are read from the buffer memory into registers to be ready for the next stage.
- Execution: the task is executed using the processing unit.
- WB (Write Back): the result is written back to the multi way unit buffer, and the state table entry is updated for that task.

Hardware Implementation

In this system, only one task is executed every clock cycle because only one processing element exists. The system consists of seven main units: the main memory, state table, multiway function unit buffer, execution array, processing element, instruction buffer, and address generation unit. All units are driven by an external clock. Fig. 5 shows a block diagram for the hardware system. This system executes the tasks of the DFG as follows:

- Initially, the address counter is cleared to 0. A descriptive information of a DFG is loaded into the main memory, and the state table and the function unit buffer are cleared.
- The address counter generates sequential addresses to the memory. A task is loaded to the instruction buffer.
- Instruction buffer checks the task_id and directly loads the first four tasks of every iteration without saving them into its queue. (Other tasks are saved).
- Once the instruction buffer is full, it sends a buffer_full signal to the address generator to stop counting.
- When the instruction buffer receives a ready signal from the execution array unit, it sends one task to the state table and the execution array register.
- The task operands are checked for availability using the state table. Ready operands are sent to the multi-way function unit buffer.
- A function unit buffer receives operand IDs and their iteration numbers, followed by extracting their corresponding values and sending them to all stages of the execution array unit.
- At every clock cycle and when a stage receives a ready signal from the first stage in this unit, a task is shifted up to the other stages.
- The execution array stages wait for any values coming from the function unit buffer or from the processing element unit and compare them with its task operands IDs until a match is found.
- Operand values are stored within a task as soon as an Alu_Ready signal reaches to the last stage of the execution array. Then, a task is loaded to the processing element.
- After execution, the result is sent to the state table, function unit buffer, and the execution array.
- The state table and the function unit buffer update their content for the executed task. When the task with last_task bit set, is encountered, an end of iteration is indicated.
- At the end of each iteration, the state table and the instruction buffer are cleared, and the buffer contents are updated to be ready for executing the next iteration. Rb and Rd signals from the function unit buffer and the state table respectively are sent to reset the address generator to start a new iteration.

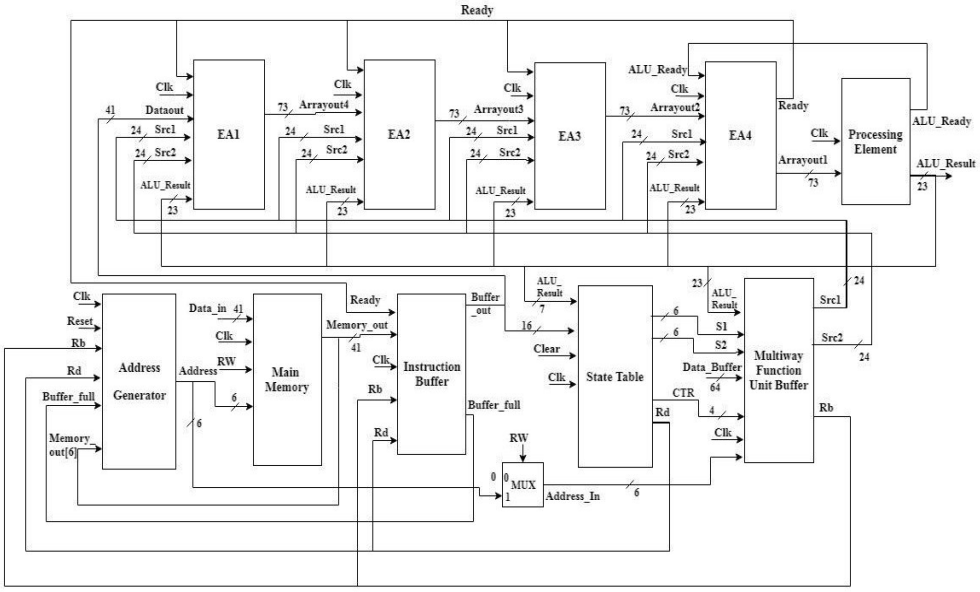

Figure 5: The hardware model.

7. RESULTS

The proposed system is synthesized targeting vertix7 XC7VX690T-FFG1157 FPGA device from Xilinx. The implementation has been carried out for 64 instructions (tasks), 64-bit state table, and 64X64 bit buffer in the function unit to maintain tasks information for four iterations. The multiplier size is 16-bit by 16-bit. Also, a 16-bit adder is used.

Synthesis results show that the design can run at a frequency of 355.29 MHz. Table 3 shows a summary of the hardware requirements and resulting sampling period.

Table 3: Summary of implementation requirements and performance

Number of Slice Registers Utilized	4818
Number of Slice LUTs Utilized	8867
Minimum Period Time	2.815 ns

The parallel implementation of the system, which is not presented here, achieves the best performance with clock frequency of 266 MHz, whereas the presented sequential system achieves a 210 MHz clock rate. However, the retiming technique presented in [11] achieves a maximum clock frequency of 176 MHz. So, the performance of the parallel system is faster than the retiming technique by 51.2%, and the presented sequential system performance is faster than the retiming technique by 19.3%.

8. CONCLUSION

This paper has proposed a new scheduling technique that is targeting DSP applications, represented by data flow graphs (DFG). This technique is composed of two parts: task analysis of the data flow graph and hardware assignment of the tasks to achieve the desired algorithmic behavior. That is, the two parts are combined to form a complete system. The scheduling

technique is designed to minimize the execution time of a single iteration of the DSP application and thus maximizing the system performance. In the task analysis part, the nodes are arranged in a queue of nodes such that when the tasks associated with these nodes are executed in order, data dependencies are preserved. Two consecutive tasks may be combined to form a compound task. In many cases, the compound task can be executed in a time that is less than the sum of the execution times of the individual tasks. The second part of the scheduling technique is the allocation of the tasks of the DFG on a general purpose pipelined hardware architecture. The proposed system comprises one processing element where all tasks are processed sequentially. The parallel version of the proposed system has achieved the best performance; it uses four homogeneous parallel processing elements for concurrent task execution. It has been shown that our system outperforms the retiming technique in terms of the iteration time. In terms of the system area, the retiming technique and the serial version of our system require the same number of DSP slices when FPGA hardware implementation is used.

REFERENCES

[1] DeFatta D, Lucas J, Hadgkiss W. Digital signal processing, a system design approach. John Wiley & Sons. 1988.

[2] Trevillyan L. An overview of logic synthesis systems. Conference on Design Automation. IEEE; 1987; 166-172.

[3] Schafer R, Oppenheim A. Digital Signal Processing. 1st ed. Englewood Cliffe, New Jersey: Prentice Hall; 1975; 31-32.

[4] Shatnawi A. Compile-time scheduling of digital signal processing data flow graphs onto homogeneous multiprocessor systems. Ph.D. Thesis Department of Electrical and Computer Engineer, Concordia University, Montreal Canada, 1996.

[5] Shatnawi A. Optimal Scheduling of Digital Signal Processing Data-flow Graphs using Shortest-path Algorithms. The Computer Journal. 2002; 45(1):88-100.

[6] Wang G, Wang Y, Liu H, Guo H. HSIP: A Novel Task Scheduling Algorithm for Heterogeneous Computing. Scientific Programming. 2016; 2016:1-11.

[7] Ullah Munir E, Mohsin S, Hussain A. SDBATS: A Novel Algorithm for Task Scheduling in Heterogeneous Computing Systems. Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW). IEEE; 2013; 43-53.

[8] Liu G, He Y, Guo L. Static Scheduling of Synchronous Data Flow onto Multiprocessors for Embedded DSP Systems. Third International Conference on Measuring Technology and Mechatronics Automation. IEEE. 2011; 338–341.

[9] Zhou N, Qi D, Wang X, Zheng Z, Lin W. A list scheduling algorithm for heterogeneous systems based on a critical node cost table and pessimistic cost table. Concurrency and Computation: Practice and Experience. 2016;29(5):1-11.

[10] Kang Y, Lin Y. A Recursive Algorithm for Scheduling of Tasks in a Heterogeneous Distributed Environment. 2011 4th International Conference on Biomedical Engineering and Informatics (BMEI). IEEE. 2011; 2099-2103.

[11] Woods R, McAllister J, Lightbody G, Yi Y. FPGA-Based Implementation of Signal Processing Systems. Chichester, United Kingdom: John Wiley & Sons 2009; 145-169.

[12] Parhi K, Messerschmitt D. Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding. IEEE Transactions on Computers. 1991;

[13] McFarland M, Parker A, Camposano R. Tutorial on high-level synthesis, 25th Design Automat. 1988. p. 330-336.

[14] Hurson A, Milutinović V, Advances in computers. Waltham, MA : Academic Press, 2015;(96):1-45.

[15] De Groot S, Gerez S, Herrmann O. Range-chart-guided iterative data-flow graph scheduling. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications. 1992; 39(5):351-364.

AUTHORS

Ali Shatnawi is a professor of computer engineering. He received the B.Sc and M.Sc in electrical and computer engineering from the Jordan University of Science and Technology (JUST) in 1989 and 1992, respectively; and the Ph.D degree in electrical and computer engineering from Concordia University, Canada, in 1996. He has been on the faculty of the Jordan University of Science and Technology since 1996. He served as the director of computer centre 1996-1999, Vice-dean 2001-2002, Dean of IT at Hashemite University 2002-2005 and dean of Computer and Information Technology, JUST, 2016-2018. His present research includes algorithms and optimizations, hardware scheduling, computer architecture and high level synthesis of DSP applications.

Osama Al-Khaleel is an associate professor of Computer Engineering in the Department of Computer Engineering of Jordan University of Science and Technology (Irbid, Jordan), received his B.S in Electrical Engineering from Jordan University of Science and Technology in 1999, M.Sc. and Ph.D. in Computer Engineering from Case Western Reserve University, Cleveland, OH, USA in 2003 and 2006 respectively. Currently, his main research interests are in embedded systems design, reconfigurable computing, computer arithmetic, and logic design.

Hala AL-Zu'bi received her B.SC. in Computer Engineering from Yarmouk University in 2012, and M.Sc. in Computer Engineering from Jordan University of Science & Technology in 2018. Her research interests include computer architecture, hardware description language, task scheduling and data flow computing.