DSP Algorithms in FPGA - Proposition of a New Architecture

Piotr Kolasinski[a], Wojciech Zabolotny[a]

[a]Institute of Electronic Systems, Nowowiejska 15/19, 00-665 Warszawa, Poland;

ABSTRACT

This paper presents a new reconfigurable architecture created in FPGA which is optimized for DSP algorithms like digital filters or digital transforms. The architecture tries to combine advantages of typical architectures like DSP processors and datapath architecture, while avoiding their drawbacks. The architecture is built from blocks called Operational Units (OU). Each Operational Unit contains the Control Unit (CU), which controls its operation. The Operational Units may operate in parallel, which shortens the processing time. This structure is also highly flexible, because all OUs may operate independently, executing their own programs. User may customize connections between units and modify architecture by adding new modules.

Keywords: DSP, VHDL, FPGA, architecture

1. INTRODUCTION

Field Programmable Gate Arrays (FPGA) are general-purpose devices. They are the most popular universal devices for many applications because of their programmability and reduced development costs. Today's FPGAs contain not only the logic blocks, but also many specialized modules like DSP blocks and memory blocks.[1] The FPGA allows the user to implement the arithmetical blocks well suited for the required precision of arithmetics (even though some "native" lengths of the data words are preferred). Additionally, the multilevel structure of internal interconnections allows to send multiple data in parallel, resulting in very high internal data throughput. These features makes the FPGA chips a very efficient platform for implementation of the DSP algorithms,[2] however it is not easy to build an architecture offering both - high throughput and high flexibility.

1.1. Typical DSP architectures

One of typical architectures used for DSP algorithms is a DSP processor. The performance of this architecture is limited by a few bottlenecks:

- single program memory bus allows to read only one code word in a single clock cycle
- small number of data buses (usually 1 or 2) allows to transfer only a few data words in a single clock cycle
- small amount of data processing blocks (like "multiple and accumulate" (MAC) blocks) allows to perform only a few operations in parallel

Another typical architecture used for the DSP applications is the "datapath" architecture. This structure offers high throughput (in every clock cycle new data words are read at the inputs, and new result is provided at the output). However this architecture is not flexible - each processing block is used only for one particular operation in the algorithm. The datapath architecture is also not suitable for algorithms which require iterative operations.

Therefore a need exists to create another architecture, able to combine highly parallel, pipelined operation of the "datapath" architecture with the flexibility of the DSP processor, where the same resource may be reused for different purposes in the different steps of the algorithm.

Further author information: (Send correspondence to W.M.Zabolotny)
W.M.Zabolotny: E-mail: wzab@ise.pw.edu.pl, Telephone: +48 22 2347717

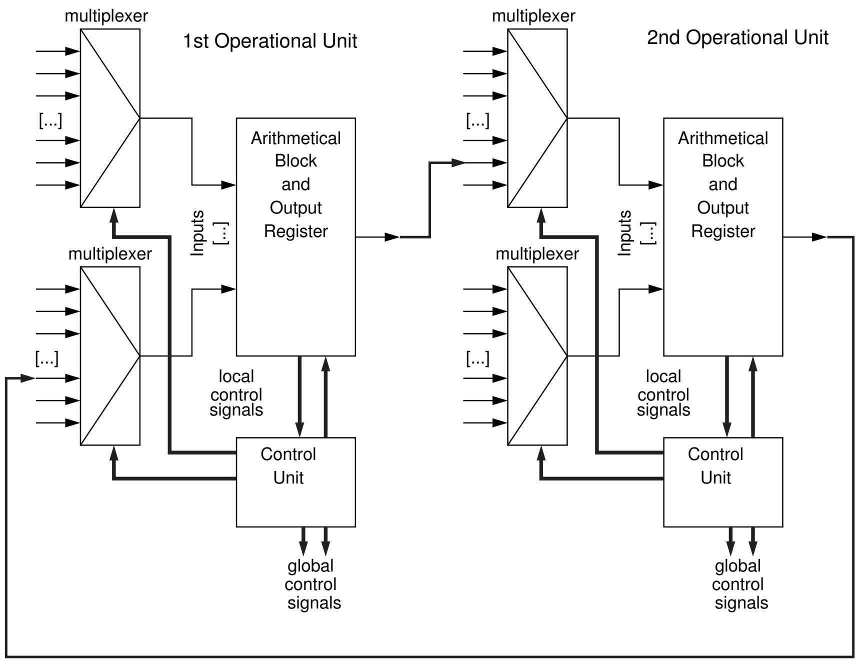

Figure 1. Two interconnected Operational Units with their internal structure.

2. PROPOSITION OF THE NEW ARCHITECTURE

To avoid limitations of the typical DSP processor, the resources of the FPGA chips are divided between multiple Operational Units (OU), which may operate independently, in parallel. To allow reuse of the same resources for different purposes in different steps of the algorithm, each OU is equipped with the Control Unit (CU) (solution similar to the one described in Ref. 3). The CU executes its own program and controls what operations are performed by the OU in each cycle, where are the data read from, and where are the results stored. This solution creates the distributed code memory, which allows to overcome limitations resulting from single program memory bus in the DSP processor. The intermediate results of calculations are stored in the registers (implemented with FlipFlops, or distributed RAM), which create the distributed data memory.

2.1. The Operational Unit

The Operational Unit contains arithmetical modules, registers and multiplexers. The general structure of the Operational Unit is shown in the Fig.1. The multiplexer at every input of the arithmetical module allows to select the source, from which the data should be received in the particular clock cycle.

The operations performed by the arithmetical module may be defined by the user, and depend on the hardware platform. In fact it is not necessary, that all OUs are the same, there may be a few OUs performing more complicated operations (e.g. division), and more OUs performing simple multiplications and additions (like “Elementary Mathematical Blocks” in Ref. 4).

The output of the arithmetical unit is connected with the register, and further with the inputs of multiplexers in the next blocks.

2.2. The Control Unit

Different operations performed by the OU may require different number of clock cycles. It is also possible, that the OU may perform more complex calculations (e.g. calculation of square root) working iteratively. To implement such operations it is necessary to select the data source and the operation to be performed by the arithmetical block in each clock cycle. This

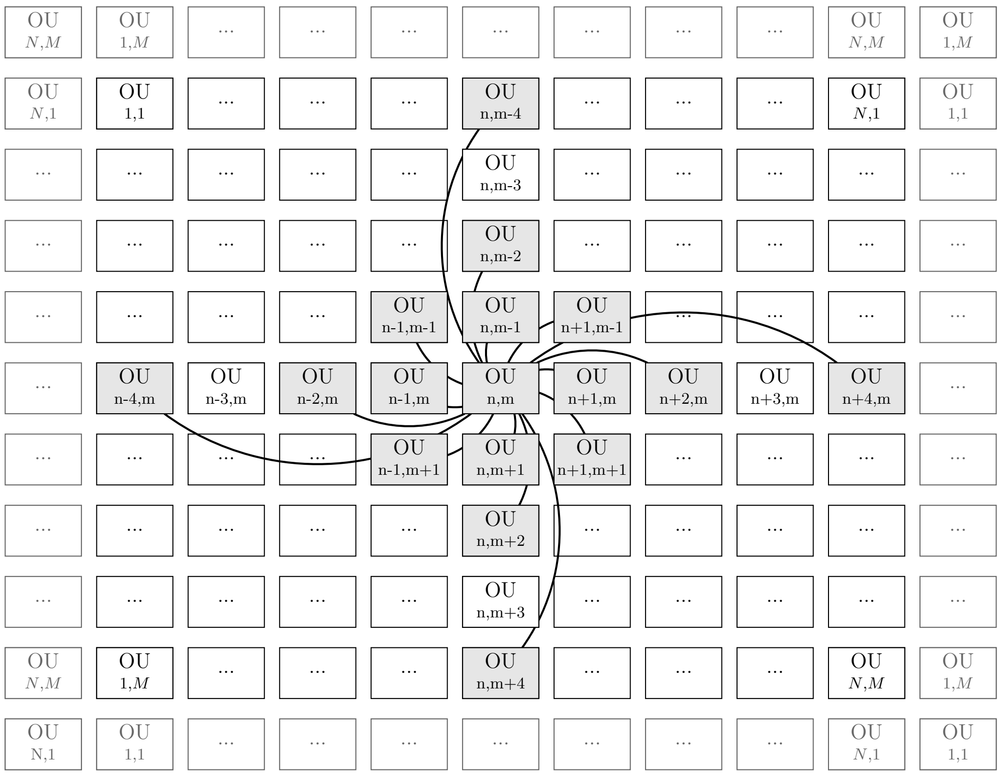
Figure 2. Interconnections in the proposed architecture. All Operating Units are organized in the rectangular matrix of size $N \times M$, wrapped to create a torus-like structure. The inputs of the OU with coordinates (n,m) are connected to the outputs of shaded neighboring OUs.

task is performed by the Control Unit. Each OU is equipped with the CU. The CU is a simple state machine, controlled by the microcode stored in its code memory (which may be either ROM or RAM, depending on the option set by the user when compiling the architecture).

The CU provides also synchronization with the other OUs, connected to its OU. To make it possible, the communication channels between OUs must also provide some handshake signals like "data available" and "data read acknowledge"

2.3. Interconnections between Operational Units

Theoretically, the best flexibility is assured by the structure, where each OU is fully interconnected with all others OUs. However such a structure is usually not possible to implement in the FPGA, because the number of connections is very high (equal to $N^2 - N$ for N Operating Units). Therefore another layout of interconnections has been proposed.

All OUs are considered to form a rectangular structure with the edges joined to create a "torus", as shown in the Figure 2.

The inputs of each OU are connected to the outputs of its 8 direct neighbors, with 4 neighbors located in the distance of 2 units, and with 4 neighbors located in the distance of 4 units. In this configuration each input multiplexer must have 16 inputs.

If the amount of OUs which can be implemented in the FPGA is small (e.g. less than 64), another interconnection scheme may be used (see Figure 3. In this configuration the inputs of each OU are connected to the outputs of its 8 direct neighbors, and with 4 neighbors located in the distance of 2 units. In this configuration 12-input multiplexers are needed.

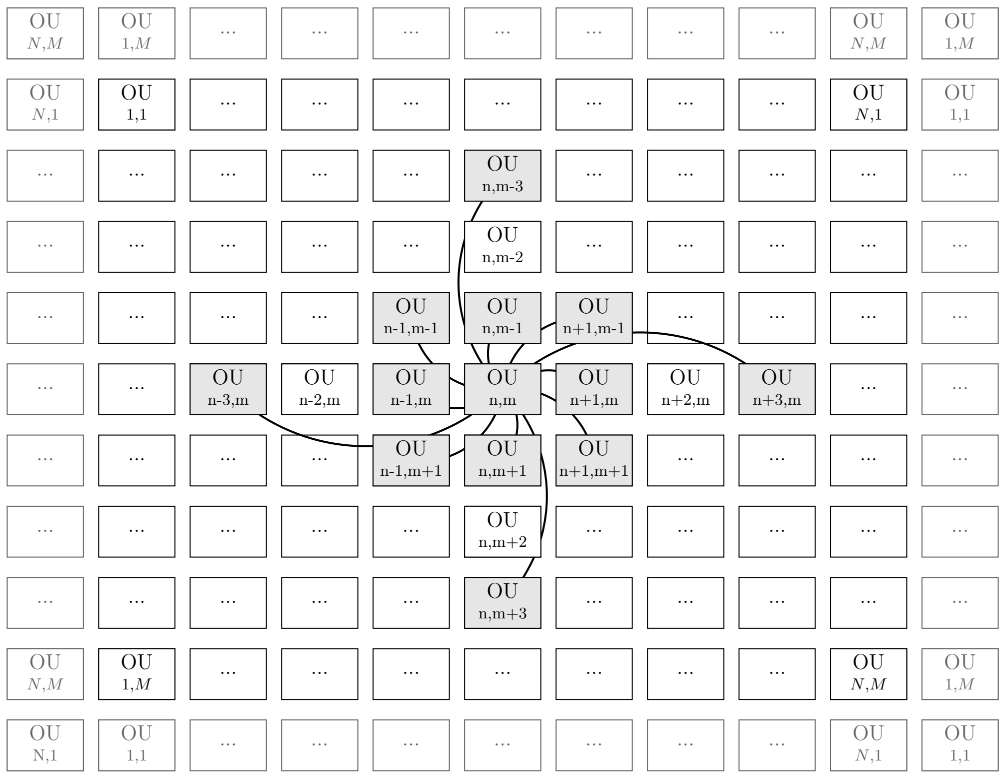

Figure 3. Interconnections in the proposed architecture for lower amount of OUs. All Operating Units are organized in the rectangular matrix of size $N \times M$, wrapped to create a torus-like structure. The inputs of the OU with coordinates (n,m) are connected to the outputs of shaded neighboring OUs.

Such dense net of interconnections allows to send many data in one cycle between the directly interconnected OUs. It is also possible to send data between more distant OUs, using other OUs as "relays", but it requires more clock cycles. Due to this property of the proposed architecture, the speed of the processing significantly depends on the distribution of the processing tasks between different OUs. The tasks requiring intensive data exchange should be assigned to neighboring OUs. The decomposition of the algorithm into simple tasks and assignment of tasks to the particular OUs is a complex job, which requires specialized software support.

3. SOFTWARE SUPPORT

The DSP algorithm, is typically written and tested in a C-like or script (Matlab or Scilab) language. To execute this algorithm on the proposed architecture, it is necessary to decompose it to the tasks which may be executed by a single OU, to assure the proper synchronization between different OUs, and finally to generate programs for particular CUs. This task could be done "by hand" in a simpler architecture,[4] where the latencies introduced by arithmetical blocks were predictable and constant, and where the algorithm did not use any conditional, data-dependent operations.

However the new architecture offers possibilities to implement algorithms using the iterative operations, where latency introduced by the operational unit is neither constant nor predictable (i.e. it may depend on the processed data).

The translator responsible for this task is currently under development. The translator must perform the following tasks:

- Decompose the algorithm into simple operations which may be performed independently, in parallel by the OUs

- Assign the operations to be performed to different OUs, so that the expected numbers of cycles required by all OUs to execute their tasks are equal (load balancing)
- Lay out the OUs so that as much data as possible may be transferred in the single cycle
- Generate the code for each CU

The described procedure does not make any assumptions regarding the particular properties of the algorithm to be performed by the proposed architecture. However, if the user wants the architecture to perform only a particular class of algorithms, it should be possible additionally to optimize the architecture on the synthesis stage.

3.1. Synthesis stage optimization

The proposed architecture is implemented in VHDL, however theoretically it can be ported to another Hardware Description Languages (Verilog, SystemC or others).

The architecture may be synthesized in a fully configurable form. In this case all CUs are controlled by the RAM stored code, all inputs of the multiplexers are available.

Sometimes the algorithms which will be implemented in the architecture do not require the dynamic change of the code executed by some CUs. In this case the code for this CUs may be stored in the inferred ROM instead of block RAM. If the CU code is fixed, some inputs of the multiplexers may be never used. In this case this inputs and associated logic may be optimized out, resulting in overall decrease of resources consumption and increase of performance. However it is still unclear how to implement the translator for such ”partially fixed” architecture, or how to write a translator which could automatically generate the ”partially fixed” architecture for the particular class of algorithms.

4. CONCLUSION

The proposed architecture may find numerous applications in implementation of the DSP algorithms. It allows for optimal usage of the resources of FPGA, and simultaneously makes possible to perform multiple operations in parallel. The architecture is also flexible. The user may create his own modules which can be added to the architecture. The efficient system of interconnections allows sending much data in the same cycle between the neighboring nodes. Architecture can work in parallel systems and because of its flexibility it can be very quickly adopted to new applications and algorithms.

REFERENCES

1. “Stratix III Device Handbook .” http://www.altera.com/literature/hb/stx3/stratix3_handbook.pdf.
2. H. Lee and G. Sobelman, “Digit-serial reconfigurable FPGA logic block architecture,” *Signal Processing Systems, 1998. SIPS 98. 1998 IEEE Workshop on* , pp. 469–478, 1998.
3. P. Sinha, A. Sinha, and D. Basu, “A novel architecture of a re-configurable parallel DSP processor,” *IEEE-NEWCAS Conference, 2005. The 3rd International* , pp. 71–74, 2005.
4. W. M. Zabolotny, K. Bunkowski, T. Czarski, T. Jezynski, K. Pozniak, P. Rutkowski, S. Simrock, and R. Romaniuk, “FPGA based cavity simulator for tesla test facility,” *Proceedings of SPIE* **5484**, pp. 139–147, 2004.