Approach to design of a general purpose dynamically reconfigurable microprocessor architecture

Piotr Amrozik, Rafał Kiełbik, Andrzej Napieralski

Lodz University of Technology

Abstract

Several dozen years ago first successful implementations of digital VLSI technologies began the general purpose computer era. Since then the technological advancement and the development of new computing architectures have the direct impact on the computational speed. Currently we are reaching the point where the possibilities of decreasing the feature size of the silicon VLSI technology are brought almost to its physical limits and other technologies are not yet developed enough to substitute it. Therefore the research on the new paradigm in computing architectures becomes even more important.

Dynamically reconfigurable circuits perfectly fit in today's trends of computer development. Their main advantage is an ability to merge simultaneously: versatility – as they can be configured to do different tasks, and dedication to a task – because they can instantiate an application specific hardware for a specific task. But due to their unique features, it is not enough to propose only the novel architecture which is expected to give better results in theory. There is a need of research on the solution which will be able to address known issues regarding dynamically reconfigurable devices, using a completely new approach to the current computing circuit implementation paradigms. Therefore, the realisation of the project requires dedicated approach to the physical synthesis allowing designation of the actual capabilities of proposed circuit.

This paper presents the approach proposed by the authors to design a dynamically reconfigurable microprocessor. Particularly it includes a discussion on possible architectures, EDA tools required during the design process and the results obtained in the course of an exemplary architecture implementation.

1. Introduction

1.1 Motivation for the research

Since the dynamic reconfiguration techniques were discovered and started to be supported by the commercially available Field-Programmable Gate Arrays (FPGAs), different concepts of using them for acceleration of many types of computations have appeared. The dynamically reconfigurable resources were initially used as stand-alone coprocessors for demanding computations like image processing, data encryption, etc. Later, the development of nanometer Complementary Metal-Oxide-Semiconductor (CMOS) technologies allowed to implement the sequential processors together with dynamically reconfigurable resources in a single die (system on chip). This technological progress allowed to develop plenty of different concepts utilizing dynamically reconfigurable resources in computations and even in custom-made processors [2], [6], [13], [14].

Other examples of coupling reconfigurable architectures with a classic processor can be found in Chimera [20], PRISC [15] or OneChip [5] systems. Tight integration with a host processor was a very interesting concept, but in such systems the coprocessor did not have a direct access to the memory and had to rely on the processor when fetching or storing data, what limited their performance. Other solutions based on loosely coupled systems like the Chameleon architecture [19] or MOLEN processor [17] allowed direct memory access for FPGA unit but at the cost of limited processor-coprocessor bandwidth.

Some research groups invented systems that acted like independent coprocessors but still needed to be controlled by an external PC. An example of such architecture was RaPiD [8]. However, it suffered from limited bandwidth and performed well only in computationally intensive applications. PipeRench [10] architecture is another example of such a system. It was able to execute some applications independently, but still needed a host processor for most of them. The architecture was composed of individually configurable pipeline stages. The executed application was first compiled into the set of virtual stages and then mapped onto physical stages in the chip.

All presented above publications show that the dynamically reconfigurable devices are able to greatly improve some easily parallelisable tasks.
Unfortunately, no one ever proposed to use them as a stand-alone processor. Probably, it is caused by the fact that design of a new dynamically reconfigurable microprocessor architecture from scratch, which is implementable on silicon in advanced process, requires development of sophisticated design methodology and wide knowledge of software engineering, computer architecture, dynamically reconfigurable circuits and design methodologies for integrated digital circuits. Moreover, it is hard to imagine better solution dedicated to general purpose programs than sequential processors. Although ready-to-use FPGA devices are widely available to researchers nowadays, they can be used for the research of a new processor architecture in very limited way. This will be explained better further.

1.2 Concept of a general purpose reconfigurable microprocessor

The concept of the general purpose dynamically reconfigurable microprocessor is described in [11]. It is an FPGA-like architecture (the matrix of dynamically reconfigurable cells) on which the programs are translated automatically by synthesis tools. They convert programs developed in high level languages in similar way as compilers do to the machine code for typical processors. The difference is that they produce bitstreams needed for configuration of the reconfigurable matrix. The concept is that these bitstreams are stored in the computer memory and used by operating system to execute programs. The programs (their threads) are split into the partitions which are implemented on the processor sequentially, as it is depicted in Fig. 1. When the processor executes the partitions of a given program “A” in one part of the matrix, the partitions of program “B” can be loaded into the matrix part that is currently free. When the partition of program “B” is loaded, both programs are executed completely separately and in the same time. The number of concurrently executed partitions is limited only by the size and reconfiguration capabilities of processor array.

The reconfigurable processor can be perceived as a typical processor with changeable number of cores and with cores that have no typical architecture (arithmetic logic unit, registers etc.), but performing computations by logic resources fitted to the task being executed. This is the sole difference between these two approaches and main advantage of the reconfigurable processor.

The presented concept indirectly explains why it is hard to use of-the-shelf FPGAs for this research. Namely, the number of reconfigurations required during program execution can be too large to make everything work effectively. Typical FPGAs have long bitstreams (even when using partial reconfiguration [7]), so their transfer to the device takes too much time. Moreover, manner of dynamic reconfiguration and granularity of their matrices are optimised for performing typical logic functions. Logic function is perceived as a big when it has i.e. 10 input signals. Nowadays many of arithmetic operations during execution of typical program concerns i.e. addressing of external devices, thus they must be 32 or 64 bit.

2. Approach to architecture design

Design of a new general purpose microprocessor architecture based on dynamically reconfigurable hardware requires development of dedicated design methodology. This methodology must include comprehensive knowledge on electronics and computer architecture. Particularly, it must cover operating systems, software development and compiler techniques, electronic hardware solutions and implementation, see Fig. 2.
The authors are interested in development of the electronic hardware implementation methodology. This is crucial task on early phase of project since it allows verifying feasibility of the proposed architecture as a real device in advanced silicon process. The approach proposed by the authors is depicted in simplified form in Fig. 3.

At the beginning initial specification of the architecture must be defined. It must include general assumptions regarding the computer architecture which will be built on the basis of developed microprocessor. In the concerned case, it is assumed that the processor is a general purpose one and must be able to substitute devices used nowadays in personal computers. This assumption imposes compatibility with existing software development and hardware standards. It means that proposed compilers (actually, in this case: synthesis tools) are able to automatically convert high-level code into bitstreams and support typical functionality of operating systems. All this issues are described in details in [11]. Another aspect of the architecture specification concerns definition of the microprocessor hardware. It must address several problems:

- dynamic reconfiguration of the circuit (bitstreams length, configuration memory),
- matrix granularity,
- architecture of reconfigurable cells,
- internal connection interfaces,
- memory interfaces, etc.

The next phase of the design methodology is realisation of the microprocessor in terms of hardware. It is done using typical methods of designing digital circuits – structural/behavioural description with Hardware Description Language (HDL), semi-custom implementation and verification methods [9] with many indispensable modifications.

The development of HDL representation of the device is especially difficult task. It lays in the heart of the whole project. It must take into account all aspects of the processor architecture, from the one hand requirements of chosen manufacturing technology, and from the second requirements of the operating system. This step of the project requires continuous verification from the both sides. Every implementation or modification of subsequent functional blocks of the processor (i.e. executive array, memory access block, etc.) requires completing physical synthesis and post-synthesis verification in order to check its feasibility and performance. That is why development of the effective implementation flow is crucial for this project.

Figure 3. Assumed hardware design methodology (simplified)

Relevant information about the part of the assumed methodology related to logic and physical synthesis of the device is presented further. First, the architecture of typical FPGA-like circuit is briefly described. Then, the problems that such architectures pose during implementation are discussed. Full details on this topic can be found in [1].

3. Architecture of typical FPGA

The most important feature of FPGA-like devices is configurability. They can be configured many times in order to perform designated tasks. It gives FPGA devices considerable flexibility which enables using them in various applications. Obviously, software applications, developed in high-level languages and compiled on general purpose processors, provides much more flexibility than FPGA, but at the cost of longer execution time and higher power consumption [7]. In comparison with ASICs, FPGAs are slower and more power-demanding but ASICs cannot be configured, i.e. they provide the only one functionality. Another matter
requiring comparison is cost of these three approaches. The cheapest is development of a software application for general purpose processor. The most expensive is development of an ASIC requiring specialised designers to be engaged, expensive Electronic Design Automation (EDA) tools and even more expensive access to the silicon technology. The FPGA approach is again somewhere in the middle - it requires designers skilled in HDL, not so expensive synthesis and mapping tools, and off-the-shelf hardware. That is why the FPGAs are often used for prototyping or even as a base hardware for various projects regarding the realisation of sophisticated computing systems.

3.1 Functional blocks

In general, the structure of simple FPGA is based on two basic functional blocks: logic block, usually called Configurable Logic Block (CLB) and routing block often called routing resources or switch box. In general, their architecture can be defined as a modular dynamically reconfigurable circuit.

CLBs are composed of combinational and sequential resources, and constitute executive elements of FPGA. Their common logic resource is Look-Up Table (LUT). It consists of a set of Flip-flops (FFs) and a multiplexer as it depicted in Fig. 4. This simple architecture enables performing any n-input logic function. That is, realisation of 4-input logic function requires the 4-input LUT (4-LUT) composed of the set of 16 FFs and one 16x1 multiplexer.

In order to extend capabilities of CLB to perform combinational and sequential functions, one FF and a 2 x 1 multiplexer must be added, as it is depicted in Fig. 5. Typical LUT sizes which are usually utilised in CLBs are 3- or 4-LUT (sometimes even 6-LUT) [7]. To realise more complex logic functions, CLBs are grouped together by additional logic and routing resources. Typically, CLB can be configured in several different operating modes. It can act as adder, counter, shift register, state machine, distributed Static Random Access Memory (SRAM), etc.

The main task of routing resources is to establish global connections between FPGA input/output ports and CLBs, and distributing clock signals inside the chip. Moreover, they are often intended for creating local connections between CLBs in order to compose bigger blocks of logic. They are mainly comprised of multiplexers, tri-state buffers and signal buses.

The example of physical distribution of FPGA basic functional blocks and global routing is showed in Fig. 6. CLBs and switch boxes are evenly distributed throughout the silicon, but routing resources take even up to 80-90% of total chip area [7]. It looks like the CLBs are floating in the sea of interconnections. That is why this type of block distribution is usually called an island style.

Since modern FPGAs are intended to perform complex computations, they are much more advanced than it is described here and consist of various sophisticated blocks. To increase performance and save capacity, several additional functional blocks are embedded into the array, like big SRAM blocks, fast-carry logic, multipliers, serial I/O blocks, or microprocessor cores. With these
resources, FPGAs can be used as a complete system on chip.

### 3.2 Reconfiguration

As it was briefly mentioned earlier, FPGAs can be configured many times. It is possible thanks to the reconfiguration resources which are often referred to as the configuration memory. These resources in modern FPGAs are usually based on SRAM. In case of FPGAs utilising this volatile solution, there is a necessity of reprogramming the device after each power-up.

The configuration memory is attached to every functional block of the device. Simple CLB, depicted in Fig. 5, based on 4-LUT, requires 17 configuration memory bits. That is, 16 bits for 4-LUT and one bit for multiplexer choosing the operating mode between the sequential and combinational one. It is worth noting that in typical FPGAs one CLB requires tens or even hundreds of configuration bits [7]. The routing resources, since they are based mainly on big multiplexers and tri-state buffers, also require a significant amount of configuration bits.

The configuration is loaded to the FPGA through the dedicated configuration bus or typically Joint Test Action Group (JTAG – it is the common name for the IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture). Since FPGAs contain hundreds of CLBs, switch boxes and other blocks, and each of these functional blocks require hundreds of configuration bits, the total number of bits to be loaded is counted in megabits. The configuration bits that are ready to be sent to the FPGA are called bitstream.

The bitstream is prepared by logic synthesis and mapping tools (usually referred to as synthesis tools). These tools are provided by the FPGA vendors. The process of generating bitstream can be divided into four most important steps:

- **logic synthesis**,  
- **mapping the design onto the device primitives**,  
- **placing and routing the design including optimisation**,  
- **generating bitstream**.

Modern FPGA can be configured statically or dynamically. Static configuration requires stopping current operation of the device and loading the whole bitstream at once. After the configuration of the device, its operation can be started again. There is no possibility to share the result data between subsequent configurations by means of the device resources. This can be done with external hardware (e.g. external SRAM).

Dynamic reconfiguration, in general, consist in reconfiguration of the device without stopping its current operation, and in particular cases without erasing current operation data. It can be achieved using several techniques. One of the techniques, introduced for the first time by Xilinx (one of the biggest vendors of FPGAs) is run-time partial reconfiguration. It relies on the possibility to reconfigure only part of the device without interrupting operation of the remaining parts. More information about partial reconfiguration can be found in [7].

More advanced type of dynamic reconfiguration is the multi-context dynamic reconfiguration. It consists in multiplication of configuration resources in such a way that it enables the possibility of switching the configuration in a single clock cycle. Such an approach introduces new possibilities e.g. sharing the data between contexts. It can be obtained by dedicated FFs which states can be saved during context switching from being erased. This technique is often utilised by sophisticated hardware in order to hide reconfiguration time overhead. Off-the-shelf FPGAs are not equipped with multi-context dynamic reconfiguration so far. The reason is that the additional configuration resources consumes too much space on silicon to be used in commercial devices. In addition, implementation of this feature greatly complicates synthesis tools. Therefore, the multi-context dynamic reconfiguration is still in a research phase.

### 4. The problem of implementing modular dynamically reconfigurable circuits

Design of an integrated modular dynamically reconfigurable circuit requires sophisticated implementation flow which employs hierarchical approach along with cloning techniques [12]. This approach solves the critical problems related to Static Timing Analysis (STA) [3]. But one of the specific feature of this flow is that it makes the design very sensitive to the quality of a floorplan. This is a typical problem for the hierarchical designs [9] [18].

In case of FPGA-like architectures most part of a chip floorplan is comprised of a matrix of identical CLBs. Since CLB constitute partitions to be cloned in the chip layout [12], the size and shape of these logic blocks significantly affect the design – its timings, size and power consumption. Therefore, the designers need something more than general early chip estimations in such case. They rather have to explore the design space of CLB before the start of the floorplan preparation in order to carry out the whole process effectively.

The early constraint-based design space exploration of a CLB is an iterative process, which consists in determining design space points by manipulating timing constraints [3] [16]. Each point is composed of three values: worst path delay, silicon area utilization and power consumption of the circuit. The result of this process is a set of design
space points with timing constraints correlated to them.

Determination of the worst path delay (timing characterization) of a CLB requires analysis of path delays in relation to each of its configuration modes. It can be realized with every kind of simulations which allow collecting data on module timings, for example with back-annotated functional simulations or STA. The first approach seems to be a natural candidate for this purpose. However, it requires additional effort from designers (i.e. preparation of appropriate testbenches) and it has to be done using dedicated software, outside of the synthesis tools. Therefore, back-annotated functional simulations are not suitable for early constraint-based design space exploration of the design. Fortunately, STA, which is used by every synthesis tool, can be employed for this purpose as well. Such an approach gives a significant advantage since it allows performing timing characterization of a CLB during its synthesis. In consequence, it enables to explore a design space of the module in one logic synthesis session.

Timing characterization with STA of a CLB requires detailed analysis of reports produced by STA what is a quite laborious task. Moreover, the architecture of reconfigurable logic blocks causes serious problems for STA engine [12]. These obstacles can be overcome by appropriate timing constraining of the circuit. Therefore, the matter of timing constraining of CLB for their timing characterization with STA is very important and requires careful consideration – it will be discussed further in this paper.

Concluding, an early constraint-based design space exploration and timing constraining of CLB are crucial moments during the logic synthesis stage of modular reconfigurable integrated circuit design. These steps of the methodology are very time-consuming and quite difficult even for experienced designers. Unfortunately, up to now, there haven’t been any EDA tools that support designers in that field. Constant interest of researchers in integrated DR systems became the motivation for starting the work on TCT (Timing Characterisation and constraining Tool) [1] – the tool which is intended to make the designers’ life easier and encourage researchers to implement custom dynamically reconfigurable integrated circuits.

The solution of the problem of early constrain-based design space exploration became the main field of research for the authors. They proposed the technique of timing constraining of reconfigurable circuits, which eliminates the problem of combinational loops and simultaneously enable to use STA for the module timing characterization. This technique allows automating the timing constraining of reconfigurable circuits and constraint-based design space exploration. These algorithms were implemented using Tool Command Language (Tcl), which is widely supported by synthesis tools. TCT was integrated with CADENCE Encounter RTL Compiler [4].

TCT requires some input data that has to be prepared by the designer. Among other things it is the specification of reconfigurable circuit configurations – list of its configuration registers (configuration memory) and their states. The format of the input data has to be compliant with Comma-Separated Values (CSV).

TCT generates reports about all parameters of the circuit, which are required for accelerating the floorplanning process. Moreover, it prepares all files: netlists and corresponding timing constraints in Synopsys Design Constraints (SDC) format for every designated design space point. These files can be used directly by the physical synthesis tools during next stages of the design implementation. The time required by TCT to generate the results is of magnitude of hours with a typical desktop computer.

5. Results

The general purpose reconfigurable microprocessor was successfully designed with the proposed design methodology. Currently, it is implemented in Taiwan Semiconductor Manufacturing Company (TSMC) 65 nm process.

According to the concept described in section 1.1, the processor is built as a typical FPGA device. The silicon die with size of 6.25 mm² contains executive matrix built of 400 CLBs and other required functional blocks, see Fig. 6. So far, it can perform only basic tasks effectively and not all of the requirements of the operating system are supported.

The CLBs do not have typical for sequential processors executive blocks like arithmetic logic units or registers. All operations are realised on LUT-like hardware. Bitstream required for
configuration is short in relation to the CLB of typical FPGA. The processor matrix can be configured very flexibly, that is each CLB can be configured independently and without influence of adjacent cells. This is indispensable for the assumed processor architecture bearing in mind that program partitions must be able to be loaded in short time and in any part of the device.

Simple operations like adding or multiplying of two 32-bit operands requires 8-32 CLBs. Therefore, time needed for reconfiguration of partition realising these tasks is acceptable. Comparing to the sequential processor, these operations do not require so many clock cycles, and at the same time the clock frequency is much lower too. However, this phase of the project is not focused on the performance verification thus it is too early to fairly judge whether the new processor architecture is better than the sequential one.

At this stage of the project more details about the microprocessor architecture cannot be published since it is in the process of patenting.

6. Conclusions

In this paper the approach to design of a new general purpose microprocessor architecture is concisely presented. It is focused mainly on assumed methodology of hardware development. This approach employs available synthesis tools and authors’ solutions.

Silicon implementation of FPGA-like circuits is difficult. Such architectures pose serious problems to the STA and synthesis tools. Moreover they are not directly supported by currently available EDA tools. This motivates authors to develop TCT – the EDA tool which facilitates and accelerates implementation of dynamically reconfigurable circuits.

Development of the microprocessor requires taking into consideration results of its implementation on silicon. This is necessary in order to obtain reliable information whether theoretical architecture is feasible and gives assumed performance in reality. Result of silicon implementation gives feedback to the architecture initial specification and behavioural/structural HDL representation.

Presented design methodology is used for successful implementation of the processor in TSMC 65nm technology. Obtained results allow verification of the performance of the real device and will enable its comparison with currently used processors. They confirmed usefulness and reliability of the proposed methodology and allow further developing of the idea of general purpose dynamically reconfigurable microprocessor.

Certainly, presented approach to design of the reconfigurable processor will evolve during realisation of the project. This is driven by continuous progress in silicon technologies and EDA tools, as well as requirements of the developed architecture.

Acknowledgements

This work was funded by the Polish National Science Centre on the basis of the decision number DEC-2011/01/N/ST7/05242.

Bibliography

Authors:

Dr Piotr Amrozik
Lodz University of Technology
ul. Wólczańska 221/223
90-924 Łódź
tel. +48 (042) 631 27 22
fax +48 (042) 631 03 27

email: pamrozik@dmcs.p.lodz.pl

Dr Rafał Kielbik
Lodz University of Technology
ul. Wólczańska 221/223
90-924 Łódź
tel. +48 (042) 631 27 22
fax +48 (042) 631 03 27

email: rkielbik@dmcs.p.lodz.pl

Prof. Andrzej Napieralski
Lodz University of Technology
ul. Wólczańska 221/223
90-924 Łódź
tel. +48 (042) 631 27 22
fax +48 (042) 631 03 27

email: napier@dmcs.p.lodz.pl


