Design and Verification of Dynamically Reconfigurable Architecture

D. Kesler, S. Dautović, R. Struharik
University of Novi Sad, Faculty of Technical Sciences, Department of Electronics, Novi Sad, Serbia
kesler.daniel@gmail.com, dautovic@uns.ac.rs, rasti@uns.ac.rs

Abstract—This paper introduces a novel programmable reconfigurable architecture, called vCell Matrix. Proposed architecture is based on the Cell Matrix architecture with several important modifications that allow simpler and fast reconfiguration. The new architecture is optimized for the implementation using Xilinx FPGA devices. A special embedded system, based on Xilinx Virtex5 FPGA device, together with embedded Linux was also developed to test the performance of the vCell Matrix architecture.

I. INTRODUCTION

Until recently, single-core processor systems have dominated the field of personal computers. Usage of parallel computing architectures has been mainly limited to the scientific computation field. Scientists have used various parallel processing architectures, such as symmetric multi-processors (SMP) to speed-up the execution of computationally intensive applications. Recently, multiprocessor systems are being increasingly introduced and used even in the personal computer and mobile devices domains. Multiprocessing systems enable significant performance increase, but with the higher power demands.

Reconfigurable computing (RC) offers an interesting alternative to the von Neumann-type computers. In RC, hardware elements of the computing system can be reconfigured, possibly even dynamically. Reconfigurability of the computing system allows it to emulate a wide range of different computing architectures. This flexibility of reconfigurable architectures and their potential performance improvements, e.g. increased computational performance or lowered power consumption, have allowed the RC architectures to become an interesting alternative to traditional computing architectures in several computing domains.

There are many different classes of RC architectures. One possible classification has been proposed by Todman et al. [1]. According to Todman, who has continued the work of Compton and Hauck [2], there are five distinct classes of RC architectures, as shown on the Fig. 1. First four classes have the common feature, the presence of one physical processor that is controlling the operation of entire system. The difference between these four classes is in the way the communication between the processor and the reconfigurable fabric (RF) is achieved.

On Fig. 1a, RF is connected to the processor using the processor’s I/O bus. Although this structure is the easiest to implement, it offers the slowest data transfer speed between the processor and the RF.

Fig. 1b and 1c show two possible architectures of the system in which the RF is connected to the processor through its memory sub-system. By incorporating RF directly into the processor’s memory sub-system, greater data transfer speeds can be achieved, allowing for shorter configuration time, which is one of the most important characteristics of any RC architecture.

Fig. 1d shows the system in which RF is connected directly to the processor’s data path. This allows RF to be treated as one of the functional units of the processor. This type of connection allows RF to access all of the processor’s local information, for example the register file.

The fifth RC class, shown on the Fig. 1e, is the newest RC class. The existence of this class was made possible by the significant improvements in the programmable logic manufacturing technology. Instead of connecting the RF into a larger processor-based system, processor is now integrated into the RF itself. Integrated processor can be implemented as Hard-Processor, or, which is even more interesting, as a Soft-Processor that will be implemented using the RF itself. Soft-Processors provide the additional flexibility to the system designer, since now the type and the number of used processors can be specified. Examples of existing Soft-Processors include Xilinx’s PicoBlaze and MicroBlaze, and Altera’s Nios and Nios-II processors.

Figure 1. Reconfigurable computing classes

This work was partially supported by the EC FP7 REGPOT grant No. 256615 and by the Serbian MNTR grant No. TR32016.

One particularly interesting feature of the RC is the Partial Reconfiguration (PR). Partial reconfiguration is the
process of reconfiguration where only a part of the RF is being reconfigured. Based on the way this reconfiguration is performed, there are two types of PR [3]. First type of PR, Static Partial Reconfiguration (SPR), requires the device that is being reconfigured to suspend its operation until the reconfiguration process is completed. This type of PR requires some external configuration control and is not suitable for usage in the fifth RC class. The second type of PR is the Dynamic Partial Reconfiguration (DPR), also known as the Active Partial Reconfiguration. This type of PR allows the device that is being reconfigured to continue its operation during the reconfiguration process. Only the part of the device that is being reconfigured must cease its operation. The unaffected part of the system can continue working uninterrupted. This type of PR is responsible for the existence of fifth RC class. Improvement of the DPR is the so-called Dynamic Partial Self-Reconfiguration (DPSR), which allows the device to reconfigure itself, without the need of external configuration control module.

Field Programmable Gate Array (FPGA) devices are the most important enabler of the RC. Majority of the RC architectures, both academic and commercial, are based on the FPGA devices as the key technology for the actual implementation. There are of course different approaches also, but they are still limited mainly to the academic community: GARP [4], OneChip [5], Chimaera [6], Pleiades [7] to mention only some. For a more detailed survey please see [8]. One of these different approaches is also the Cell Matrix architecture [9], which has served as the basis for the development of new reconfigurable architecture presented in this paper.

Cell Matrix architecture is composed of homogenous array of cells. Each cell is directly connected only to its nearest neighbors whom it can also reconfigure. Internally, the cell is built using a Look-up Table (LUT), which is a much simpler design when compared with the FPGA’s Configurable Logic Block (CLB) building blocks. Beside LUTs, CLB typically includes flip-flops, multiplexers, carry logic for implementation of fast arithmetic operations, etc. Contemporary FPGA device also includes the RAM memory, DSP blocks and complex interconnect resources, making it much more complex and heterogeneous architecture when compared with the Cell Matrix.

II. OVERVIEW OF THE NEW RECONFIGURABLE ARCHITECTURE

Since the proposed reconfigurable architecture is based on the existing Cell Matrix architecture, first a brief overview of the most important characteristics of the Cell Matrix architecture will be presented.

A. Main Characteristics of the Cell Matrix Architecture

Cell Matrix architecture is composed of identical cells, arranged in a two-dimensional array. Each cell is connected only to its nearest neighbours (west, north, east and south). Reconfiguration process is distributed amongst the cells, meaning that any cell can initiate the reconfiguration process, by configuring its nearest neighbours. This means that this architecture doesn’t require external configuration controller, i.e. it supports the DPSR type of partial reconfiguration.
The major difference is in the structure of the individual cells that are used to build the 2D array. In vCell Matrix architecture, each cell, which will be called vCell, can reconfigure only its east neighbor. Also, the cell cannot initiate the configuration process by itself.

The interface of vCell and the structure of vCell Matrix 2D array is shown in Fig. 3.

![Figure 3. vCell Matrix cell overview and 2D array structure](image)

vCell interface is shown on Fig. 3a. It has the same data ports as the Cell Matrix cell (one input and one output data port on each of four sides), but the number of configuration control ports is reduced. There are only two ports. One input port, located on the west side and designated as C_{IN}, and one output port, C_{OUT}, located on the east side of the cell. This is a significant reduction compared with eight configuration ports in the original Cell Matrix cell architecture.

Internally, C_{IN} and C_{OUT} ports are physically connected so the value of C_{IN} port of the first cell in each row is transferred to all the others cells located in that row. This can be clearly visible on the Fig. 3b, showing the hypothetical 3x3 array composed of vCells. This means that no cell within the array can initiate the configuration process and the configuration is limited to configuration of individual rows in the array. This is a significant reduction in the configuration flexibility when compared with the Cell Matrix, but the gains are in much simpler configuration mechanism and smaller size of the LUT table located inside each vCell. vCell requires LUT table of size 16x4 bits, compared to 16x8 bits required by the cell from the Cell Matrix architecture. This is a reduction of 50% in size of the basic cell. This means that twice as much vCells can be fitted into the same silicon area when compared with the CellMatrix cells.

Regarding the maximum operating frequencies of CellMatrix and vCell cells they are identical. The reason for this is the fact that both architectures use the 16x1 memories in their cores resulting in identical signal delay times. If we analyze the configuration times, vCell based architectures will have two times shorter configuration time when compared with the original CellMatrix based architectures. Once more, this is the result of the smaller LUT table in the core of the vCell when compared with the CellMatrix cell.

vCell can operate in two modes of operation, data and configuration, similar to the Cell Matrix cell. The value of C_{IN} port determines the mode of operation of every cell. While C_{IN} is held low vCell operates in data mode, using its LUT table and current values of four input data ports to determine the required values of four output data ports. If C_{IN} is held high, vCell enters the configuration mode of operation. LUT table now works in the shift register mode, with serial input data coming in from D_{A} input data port and serial output data going out through D_{C} output data port. Matrix reconfiguration is done by rows, where multiple rows of cell can be configured simultaneously.

Fig. 4 presents the details of the internal structure of the vCell cell.

![Figure 4. Architecture of the vCell](image)

vCell is composed of one 16x4 bit LUT table which can be configured serially, using SI input. Additionally, one 2-to-1 1-bit multiplexer is required to supply the correct data to the D_{C} data output port, depending on the value of the C_{IN} port. LUT table is clocked, meaning that new data become available on the outputs of the LUT on every rising edge of the system clock. During the configuration process, on every rising clock edge, content of the LUT table is shifted by one bit. New bit arrives through the SI input port and is stored in the LUT memory location 0, while the content of the LUT memory location 63 is being transmitted through the SO output port.

Suitable applications for the vCell based reconfigurable architecture are the ones with the regular data-path and simple control-path structures, like the DSP applications. Also applications that can be heavily pipelined are also suitable for the implementation using vCell Matrix, since LUT inside vCell can be used as the pipelining register also.

### C. Configuration Control Unit

As mentioned earlier, to configure the vCell Matrix, one must use an external configuration control unit (CCU). This is similar with the FPGA devices. Unconnected signals from the boundary vCells (located on the west, south, east and north edges of the 2D array) create four access ports, designated as Port A, Port B, Port C and Port D on Fig. 5. These ports are used to transfer data to and from vCell Matrix array. This data can be configuration data or application specific, depending on the current value of the C_{IN} signal of each matrix row.

![Figure 5. Connection of the CCU with the vCell Matrix](image)

CCU unit is implemented as an IP core which can be connected to the surrounding modules using various communication protocols. On Fig. 5, CCU unit uses the PLB bus interface, allowing it to connect with IBM’s PowerPC processor family. Besides configuration function, CCU unit also enables the access to the vCell Matrix array during normal operation. Since processor using the vCell Matrix array and the array itself does not have to work on the same clock frequency, and the Operating System (OS) cannot guarantee that the two consecutive operations will finish in a predetermined number of system clock cycles, CCU must handle this clock domain crossing. Instead of measuring time or
number of elapsed clock cycles, CCU can be configured with the number of clock cycles during which it should supply the stimulus data (configuration or user) to the vCell Matrix array.

III. PROGRAMMING TOOLS

In order to simplify the mapping of user-defined circuits onto the vCell Matrix architecture, a whole range of software tools have been developed. They will be only briefly introduced in this section.

A. vCell Matrix Cell Compiler (VCMCC)

In order to map the target circuit onto the vCell Matrix architecture the functionality of every cell in the 2D array must be defined. This is done by defining the content of the LUT located inside the cell. One possible approach would be to define the LUT content by hand, but this approach would be time inefficient and prone to errors. The task of defining the LUT content for every cell would be much easier if one could define the intended functionality using Boolean logical expressions and have some tool that would then automatically map these expressions into the corresponding LUT content.

To serve this purpose a cell compiler, called VCMCC, has been developed. This compiler accepts the desired functionality, described by Boolean functions, and transforms them into the LUT content. Compilation process can be divided into several steps including lexical analysis, pre-processing, semantic analysis, code generation, optimization, etc. Although VCMCC compiler doesn’t include all the previously mentioned steps, its implementation was far from trivial. Luckily, there exist software tools that ease the compiler development process. One such tool is the Bison [10] tool that is a part of GNU software project. Using Bison it is possible to automate the process of generating tools for lexical and semantic analysis. All that is needed to generate these tools for some new language is to describe its grammar. Using this information Bison will automatically generate the code for above mentioned compilation steps.

Fig. 6 illustrates the process of generating vCell content using the VCMCC compiler. On the Fig. 6a intended cell functionality is defined using a block diagram approach. This functionality is the described using appropriate Boolean functions, shown on Fig. 6b. These functions are the input to the VCMCC compiler that automatically generates the necessary LUT content of the vCell, shown on the Fig. 6c.

B. vCell Matrix Compiler (VCMMC)

Once the functionality of every vCell has been defined, the configuration streams that will be used to configure the vCell Matrix 2D array must be devised. To do this, first the location of every required cell within the array and their relative placement to each other must be specified. This process can be done by hand, but once again, this approach would be too time consuming and prone to errors. For example, if the size of the vCell Matrix array is 32x32 cells, then one has to properly assign the LUT contents for 1024 cells.

To automate this task, a new software tool has been developed, called VCMMC. As with the VCMCC compiler, Bison package has been used to generate much of the compiler’s source code. One major difference between the grammar definition in case of VCMCC and VCMMC is that the in the latter case grammar must be able to express the information about the cells positions within the 2D array. Since the commonly used hardware modules are often regular in their structure, one of the main goals was to simplify the modelling of these regular, repetitive structures.

C. vCell Matrix Bitstream Downloader (VCMPROG)

Once the configuration bitstream using the VCMMC compiler has been generated, it must be programmed into the vCell Matrix architecture. This requires properly configuring the CCU unit and transmitting the configuration data. To ease this process, VCMPROG application has been developed. VCMPROG application takes the configuration bitstream and using the appropriate functions in the vCell Matrix device driver, downloads the bitstream into the vCell Matrix 2D array. The programming algorithm is presented on the Fig. 7.

In the first step, bitstream file (download.mbit), consisting from vCell content data and several control fields is loaded. One of the control fields is the required number of clock cycles (clock_count) to transfer all the configuration data to the matrix. Next, required files are
opened to allow the communication with the vCell Matrix device driver and configuration mask is set. This mask activates $C_{IN}$ signals of selected rows in the vCell Matrix 2D array. This allows the user to specify the sections of vCell Matrix that should be reconfigured.

In the second step, VCMPROG application checks if the specified number of clock cycles has elapsed and if so proceeds with the step 3. In this step values of the first configuration bits for the selected rows are being written to the Port A of the CCU and the clock_count variable is decremented. Step 4 waits for one clock cycle while this configuration data is being written into the vCell Matrix array. Step 5 checks if all the configuration data has been transferred to the vCell Matrix and if so, application proceeds to the step 6, in which opened files are closed and activated $C_{IN}$ signals are deactivated.

IV. EXPERIMENTS

For the purpose of testing the vCell Matrix architecture, an embedded system has been developed. This embedded system has been designed to be implemented using the Xilinx Virtex5 FPGA devices [11]. In this section an overview of the embedded system that was used for testing will be presented, as well as some sample circuits that were implemented on the vCell Matrix architecture to test its operation in some real application.

A. Structure of the embedded system

Embedded system that was used for testing comprises from the Xilinx MicroBlaze processor as the central unit of the system, several standard peripheral cores necessary for booting the embedded Linux OS and the vCell Matrix core, implemented as array of 32x32 vCell cells. Fig. 8 shows the overall structure of the developed embedded system.

![Figure 8. Structure of the embedded system used for testing](image)

One of the goals during the testing was to demonstrate the possibility of integrating vCell Matrix core into the embedded system running some operating system. Embedded Linux was chosen for this purpose. This required porting the Linux to the MicroBlaze based system and writing a Linux device driver for the vCell Matrix core.

Furthermore, to simplify the application development targeting vCell Matrix core, a special C code library, called vCM, has been developed. vCM library contains the following set of functions: setPortValue, getPortValue, triggerClock, isBusy, waitUntilDone, loadBitstream.

B. Testing the vCell Matrix Architecture

To test the functionality of vCell Matrix architecture two example designs, adder and multiplier, have been developed. These two modules have been selected because they are readily found as the building blocks in digital filters, DSP algorithms, etc.

To test both designs a special application has been developed. This application uses the vCM API to test the vCell Matrix core. Application can work in three different modes. First mode allows the user to specify the test values of the input arguments, while the second mode enables the collection of calculated result. Application also has the “test” mode, in which it exhaustively verifies the core by executing addition and multiplication operations for all possible operand values on both MicroBlaze and vCell Matrix core and then compares the results. Both test circuits have been successfully verified, correctly performing addition and multiplication operations for all possible operand values.

C. 16-bit Adder Test Design

First test design that was used to verify the operation of vCell Matrix core was the 16-bit Ripple-carry adder design. Ripple-carry adder has a very regular structure composed of two types of blocks, Half-adder (HA) and Full-adder (FA). Fig. 9 shows the functional descriptions for HA and FA circuits, required by the VCMCC compiler.

![Figure 9. Functional descriptions of HA and FA cells ready for VCMCC compiler](image)

To implement the Ripple-carry adder on the vCell Matrix architecture, three blocks of vCell cells are required, as shown on the Fig. 10.

![Figure 10. Basic building blocks for the Ripple-carry adder](image)

The first group, shown on Fig. 9a, is used to implement the Half-adder circuit, used to add the LSB bits. Circuit shown in the Fig. 9b is used to add the remaining bits up to the MSB bits. To add the MSBs circuit shown on the Fig. 9c is used.

Complete design, in case of 6-bit Ripple-carry adder, is shown on Fig. 11. Adder that was actually implemented in hardware was 16-bit.

![Figure 11. 4-bit Ripple-carry adder implemented on the vCell Matrix](image)
D. 8-bit Multiplier Test Design

Second test circuit that was implemented was the 8-bit array multiplier. Binary array multiplier uses multiple addition and shift operations to perform the multiplication. To implement the array multiplier on the vCell Matrix array four groups of cells are required, as shown on Fig. 12.

![Figure 12. Basic building blocks for the array multiplier](image)

Comparing the building blocks for adder and multiplier it can be seen that the multiplier building blocks are twice as large as the adder building blocks. Furthermore, to implement the array multiplier many more blocks are required.

On Fig. 13, functional descriptions of these four building blocks are presented. These functional descriptions are the input to the VCMCC cell compiler.

![Figure 13. Functional descriptions of multiplier building blocks ready for VCMCC compiler](image)

Complete design, in case of 4-bit array multiplier, is shown on the Fig. 14. In actual hardware testing, 8-bit array multiplier has been used.

![Figure 14. 4-bit array multiplier implemented on the vCell Matrix](image)

V. SUMMARY AND DISCUSSIONS

This paper presented a new programmable reconfigurable architecture, called vCell Matrix. Proposed architecture is based on the existing Cell Matrix architecture with several modifications that allow easier and quicker reconfiguration.

To allow easy mapping of custom circuits on the vCell Matrix architecture, a set of software tools has been developed. These tools include the VCMCC cell compiler, VCMCC matrix compiler, Linux device driver and VCM API library.

Also, CCU IP core has been developed to enable the easy integration of the vCell Matrix core into the larger system.

To demonstrate and test the operation of the vCell Matrix architecture, a custom MicroBlaze-based embedded system, targeting Xilinx Virtex5 FPGA device with the embedded Linux OS has been developed.

The performance of the new architecture was tested with two example designs, 16-bit Ripple-carry adder and 8-bit array multiplier.

This paper presents only the first results of the work aimed at developing new dynamically reconfigurable architectures. Future work will include development of whole range of dynamically reconfigurable architectures with different reconfiguration schemes in order to find the most efficient one. Also further improvement of the VCMCC and VCMCC compilers will be done directed toward the development of complete hardware synthesis platform for dynamically reconfigurable architectures.

REFERENCES


