Remigiusz Wiśniewski

Synthesis of Compositional Microprogram Control Units for Programmable Devices

Ph.D. Thesis

Supervisor:
prof. Alexander Barkalov

University of Zielona Góra
Zielona Góra, Poland, 2008

To my lovely parents.

Abstract

The dissertation focuses on the structural decomposition of control units. Six new synthesis methods of compositional microprogram control units are introduced. Proposed solutions can be divided into two main groups. The first one deals with CMCUs with mutual memory, where the internal code of the controller is recognized by the microinstruction address. The second group of presented methods is based on control units with sharing codes, where the microinstruction address is formed as a concatenation of codes generated by the counter and by the register. The aim of all proposed solutions is to reduce the number of logic blocks of the destination programmable device.

The second scope of the dissertation is partial reconfiguration of CMCUs implemented on an FPGA. The idea is based on swapping the content of the control memory, while the rest of the system is not modified. Such an approach permits to decrease the size of a bit-stream that is sent to the device. Therefore, time needed for device configuration is shorter. Additionally, the proposed solution is much safer due to fewer errors that may occur during reconfiguration of FPGAs.

Contents

The list of the most important symbols

B - the set of operational vertices of the flow-chart; B=$\{b_1,\ldots,b_K\}$

X - the set of conditional vertices of the flow-chart; X=$\{x_1,\ldots,x_L\}$

Y - the set of microoperations executed by the controller; Y=$\{y_1,\ldots,y_N\}$

K - the number of all operational vertices in the flow-chart

L - the number of conditional vertices in the flow-chart

N - the number of microoperations that are executed by the controller (the total number of microoperations is equal to $N+2$ because of two additional microoperations: y_0 and y_K)

C - the set of operational linear chains; C=$\{\alpha_1,\ldots,\alpha_G\}$

C' - the subset of the set C, contains only such OLCs that are not connected with the final vertex of the flow-chart; $C' \subset C$

I - the set of inputs of OLCs; I=$\{I_1,\ldots,I_J\}$

O - the set of outputs of OLCs; O=$\{O_1,\ldots,O_G\}$

G - the total number of OLCs in the flow-chart

J - the number of all OLCs inputs

α_g - the operational linear chain g

I_j^t - the t-th input of the operational linear chain α_j

O_g - the output of the OLC α_g

M^g - the number of operational vertices that belongs to the OLC α_g

M_1 - the number of operational vertices of the longest OLC; M_1=M^g (here α_g is the OLC that contains the most operational vertices)

R_1 - the number of bits required to encode the longest OLC; R_1=$\lceil log_2 M_1 \rceil$

M_2 - the number of OLCs in the set C (equal to the parameter G)

R_2 - the number of bits required to encode OLCs; R_2=$\lceil log_2 M_2 \rceil$

M_3 - the number of all microinstructions kept in the control memory (equal to the parameter K and to the number of operational vertices in the flow-chart)

R_3 - the minimum number of bits required to address microinstructions; R_3=$\lceil log_2 M_3 \rceil$

M_Z - the number of all OLCs inputs (equal to the parameter J)

R_Z - the number of bits required to encode all OLCs inputs; R_Z=$\lceil log_2 M_Z \rceil$

R_{OI} - the minimum number of bits required for recognition OLCs outputs

T - the excitation function for the counter; consist of R_1 variables: T=$\{t_1,\ldots,t_{R_1}\}$

D - the excitation function for the register; consist of R_2 variables: D=$\{d_1,\ldots,d_{R_2}\}$; it is not generated in case of CMCUs with mutual memory

A - the function generated by the counter (in case of CMCUs with sharing codes it is treated as a minor-part of the microinstruction address, in all other CMCUs this function directly addresses microinstructions); A=$\{a_1,\ldots,a_{R_1}\}$

Q - the feedback function usually generated by the register (except CMCUs with mutual memory, where Q is generated by the counter and it is a sub-function of A: $Q \subset A$); consist of R_2 variables: Q=$\{q_1,\ldots,q_{R_2}\}$

Z - the excitation function for the function decoder; consist of R_Z variables: Z=$\{z_1,\ldots,z_{R_Z}\}$

V - the converted microinstruction address generated by the address converter; consist of R_3 variables: V=$\{v_1,\ldots,v_{R_3}\}$; usually implemented as a memory

y_0 - the additional microoperation, used for organization the addressing mode (increments counter and forbids changing state of the CMCU when equal to 0; loads counter and changes state when equal to 1)

y_K - the additional microoperation, used for indication that the final vertex of the flow-chart will be reached (at the next clock trigger); terminates fetching of microinstructions from the control memory

S_{CM} - the volume of the control memory:

- S_{CM}=(N+2)$*2^{R_1+R_2}$ - for CMCUs with sharing codes;
- S_{CM}=(N+2)$*2^{R_3}$ - for CMCUs with address converter;
- S_{CM}=(N+2)$*2^{R_1}$ - for all others CMCUs

S_{FD} - the volume of the function decoder:

- S_{FD}=$R_1*2^{R_Z}$ - for all CMCUs with mutual memory;
- S_{FD}=$(R_1+R_2)*2^{R_Z}$ - for all CMCUs with sharing codes

S_{CA} - the volume of the address converter; S_{CA}=$(R_1 + R_2)*2^{R_3}$

The list of the most important abbreviations

ASIC - Application Specific Integrated Circuit

BRAM - Block Random Access Memory

CLB - Configurable Logic Block

CPLD - Complex Programmable Logic Device

CU - Control Unit

CMCU - Compositional Microprogram Control Unit

FDC - Flip-flop with Data and asynchronous Clear

FDCE - Flip-flop with Data, asynchronous Clear and clock Enable

FPGA - Field Programmable Gate Array

FSM - Finite State Machine

HDL - Hardware Description Language

LUT - Look-Up Table

MCU - Microprogram Control Unit

PAL - Programmable Array Logic

PLA - Programmable Logic Array

PLD - Programmable Logic Device

PROM - Programmable Read Only Memory

SoC - System-on-a-Chip

SoPC - System-on-a-Programmable-Chip

SPLD - Simple Programmable Logic Device

The list of abbreviations of synthesis methods and CMCUs structures

MM - (synthesis method of a CMCU with) Mutual Memory

FD - Function Decoder

OI - Outputs Identification

OD - Outputs identification and function Decoder

SC - Sharing Codes

SD - Sharing codes and function Decoder

CA - Address Converter

CD - address Converter and function Decoder

CMCU U_{MM} - CMCU (represented as a Unit) with Mutual Memory

CMCU U_{FD} - CMCU with Function Decoder

CMCU U_{OI} - CMCU with Outputs Identification

CMCU U_{OD} - CMCU with Outputs identification and function Decoder

CMCU U_{SC} - CMCU with Sharing Codes

CMCU U_{SD} - CMCU with Sharing codes and function Decoder

CMCU U_{CA} - CMCU with Address Converter

CMCU U_{CD} - CMCU with Address converter and function Decoder

Chapter 1

Introduction

A control unit (CU) is one of the most important part of any digital system (De Micheli, 1994; Clements, 2000; Łuba, 2003; Bolton, 1990; Bursky, 1999). It can be found in almost all devices that contain microelectronics; such as computers (central processor unit, CPU), cellular phones, cars and even remote controllers. The control unit is responsible for managing all modules of the designed system - it sends adequate microinstructions that should be executed (Gajski, 1997).

Most of control units that are available on the market are created as a single-level finite-state-machine (FSM). This means that the control unit is formed as a simple Moore or Mealy automaton (Mealy, 1955; Moore, 1956). Such a solution was good for small systems. But the size of devices grows very fast, and now complex digital systems can be implemented using one digital board such as system-on-a-chip (SoC) or system-on-a-programmable-chip (SoPC). Especially SoPC systems, where logic functions are realized using programmable logic devices (PLDs),complex programmable logic devices (CPLDs) or field programmable gate arrays (FPGAs) are very popular nowadays. Such devices compacts all elements of the design on a single chip that contains built-in logic and dedicated memory blocks (Altera, 2006; Xilinx, 2000). Therefore, traditional methods of control units prototyping evolve. One of effective methods of the CU realization is an application of the model of the compositional microprogram control unit (CMCU).

The compositional microprogram control unit is a multi-level device, where the control unit is decomposed into two main units (Łuba, 2005; Baranov, 1994; Barkalov, 2002). The first is responsible for addressing of microinstructions that

are kept in the control memory. It is a simple finite-state-machine. The role of the second part is to hold and generate adequate microinstructions. Such a solution may lead to minimization of the number of logic elements that are used for implementation of the CU. Therefore, wider areas of the target device can be accessed by other modules of the designed system. The CMCU memory can be implemented using either logic elements or dedicated memory blocks of a chip (Wiśniewski, 2005; Xilinx, 2004; Altera, 2006).

1.1 The thesis and main goals of the dissertation

This dissertation is focused on proving that the following claim is true: **the appropriate modification of traditional structures of the compositional microprogram control unit permits to decrease the number of logic blocks that are required for implementation of the controller in the target FPGA device.**

There are two main goals of the dissertation. The first goal is to **reduce the number of logic blocks that are required for implementation of the compositional microprogram control unit in an FPGA**. The reduction is reached trough an application of additional internal blocks of the control unit.

The second aim of the dissertation is to **reduce the size of the bit-stream that is send to the FPGA during the physical implementation of the compositional microprogram control unit**. This task will be solved thanks to partial reconfiguration of programmable devices.

1.2 The structure of the dissertation

The dissertation is divided into eight chapters and two appendices. *Chapter 1* presents the thesis and main goals. It outlines the structure of the dissertation.

The work related to the dissertation is reviewed in *Chapter 2*. Information about integrated circuits, programmable devices and control units are briefly described.

Functional and structural decomposition of a control units is presented in *Chapter 3*. Moreover, compositional microprogram control units based on structural

decomposition are presented in more detail. Most important symbols and abbreviations related to the CMCU are defined too.

Chapter 4 introduces new synthesis methods of a CMCU implemented in the FPGA. All proposed methods are based on the CMCU with mutual memory. Each designing method is presented in detail and illustrated by an example.

Chapter 5 has the structure similar to Chapter 4. New synthesis methods of the CMCU implemented in the FPGA are presented. However, all the proposed CMCUs are based on the application of the idea of sharing codes. Each method is shown in detail and illustrated by an example.

The idea of partial reconfiguration of control units implemented in the FPGA is shown in *Chapter 6*. First, the mechanism that permits to replace a portion of the design implemented in the FPGA is briefly introduced. Next, the traditional design process of controllers is presented. Furthermore, a new prototyping flow based on an application of partial reconfiguration of CMCUs implemented in the FPGA is proposed.

Chapter 7 briefly describes a tool that was designed for automatic synthesis of the CMCU (ATOMIC). Main ideas used during implementation of ATOMIC are presented and reviewed. The tool implements all synthesis algorithms presented in Chapters 4 and 5. Therefore, its functionality aided the prototyping process of CMCUs and it was a very important link during experiments.

The most important results of experiments are presented in *Chapter 8*. The analysis of gained values was divided into two parts. The first deals with the effectiveness of proposed synthesis methods while the second concentrates on results achieved during partial reconfiguration. A detailed analysis of experimental results is concluded with an attempt to select a suitable method depending on the initial description of the controller.

Chapter 9 summarises the dissertation. Conclusions and plans for the future work are presented.

The structure of ATOMIC is shown in *Appendix A*. Input and output data formats are described. Furthermore, all ATOMIC modules are present in detail.

Appendix B describes detailed results of experiments. There are values gained during the verification of the effectiveness of proposed synthesis methods shown. Moreover, results of partial reconfiguration of CMCUs implemented in the FPGA are presented.

Chapter 2

Related work

2.1 Information about integrated circuits and programmable devices

2.1.1 Introduction

By the late 1940s the first transistor was created as a point-contact device formed from germanium. Such an invention was a base for the further digital circuits and programmable devices. In the next decade the development of transistors benefited with the first digital logic gates and circuits classified as the TTL (transistor-transistor logic). Devices had up to 16 pins and each could perform a simple logic function (for example the device 7400 contained four 2-input NAND gates, 7404 - six NOT inverters). Those small circuits were the first **application specific integrated circuits (ASICs)** where logic functions were fixed and unchangeable. It means that the ASIC contains dedicated logic values and it cannot be reconfigured. Up to the early 1970s ASICs were developed implementing more and more gates on one chip, however the main problem was fixed logic. Once manufactured device could not be changed, therefore there was not any possibility to correct any errors or bugs. The designer could not verify his prototype using the real physical circuit.

This problem was solved in the 1970s when the first programmable devices were introduced. Those circuits were named **programmable logic devices (PLDs)**. The PLD was built as a fixed array of AND (OR) functions driving a programmable

array of OR (AND) functions (Maxfield, 2004). Such a circuit was initially used to implement the simple combinational logic. Later, the registered and tri-state outputs were added. At the early 1980s the first **complex PLD (CPLD)** was presented. Basically the CPLD is an extended version of the PLD; it contains small PLD blocks linked to the interconnect arrays. CPLDs were highly configurable but it was impossible to implement large and complex functions. Thus in 1984 the first **field programmable gate array (FPGA)** was introduced. The device was made of a matrix of programmable logic blocks. Each logic block contained 3-input **look-up table (LUT)** that could perform any combinational function. Additionally, there were simple multiplexer and flip-flop inside the logic block.

Nowadays, FPGAs are still the best solution for the designers who want to verify their prototype of the device. However, FPGAs are too slow and too expensive to compete with ASICs in case of mass-production. On the other hand, features of FPGAs like partial reconfiguration offer new ideas and new markets for such devices.

Next subsections describe programmable devices in more detail.

2.1.2 Programmable logic devices (PLDs)

Programmable logic devices can be generally divided into two groups:

- **Simple programmable logic devices (SPLD)** that refer to the relatively small programmable devices. Three types of the SPLD: the PROM, the PLA and the PLA are briefly described in this subsection.
- **Complex programmable logic devices (CPLD)** that consist of multiple smaller devices (like the SPLD). Because of their structure, CPLDs will be shown in the next subsection in more detail.

The programmable read-only memory (PROM)

The first programmable logic device was made as the **programmable read-only memory (PROM)**. Generally the PROM performs a function of the fixed AND-array which drives the programmed OR gates (array). The idea of the PROM is shown in the fig. 2.1 (the device is in the unprogrammed state).

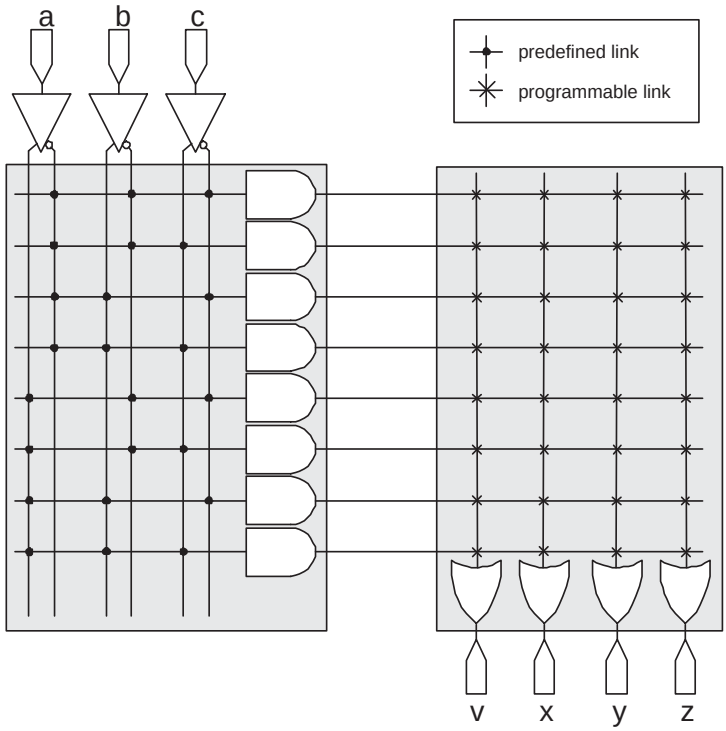

Figure 2.1: The unprogrammed PROM device

Here the predefined (fixed) AND array has 3 inputs (variables a,b,c) and eight outputs that are created as all combinations of inputs. Thus outputs of the AND matrix hold all terms of the input function. The OR-matrix is programmable and its inputs refer to outputs of the AND-matrix. Programmable links of the OR array permit to create any sum of the logic terms, therefore any logic function may be programmed using the AND-OR structure. An example of the programmed PROM device is shown in the fig. 2.2. The device performs simple combinational functions of three inputs and four outputs, where:

$$\begin{aligned} v &= a \cdot b \cdot \overline{c} + a \cdot b \cdot c, \\ x &= \overline{a} \cdot b \cdot \overline{c} + \overline{a} \cdot b \cdot c, \\ y &= \overline{a} \cdot b \cdot c + a \cdot \overline{b} \cdot c + a \cdot b \cdot c, \\ z &= \overline{a} \cdot b \cdot \overline{c} + a \cdot \overline{b} \cdot c + a \cdot b \cdot c. \end{aligned} \tag{2.1}$$

The AND-array produces the combination of all possible logic terms. Logic functions v,x,y,z are in fact realised by the OR-array via programmable lines.

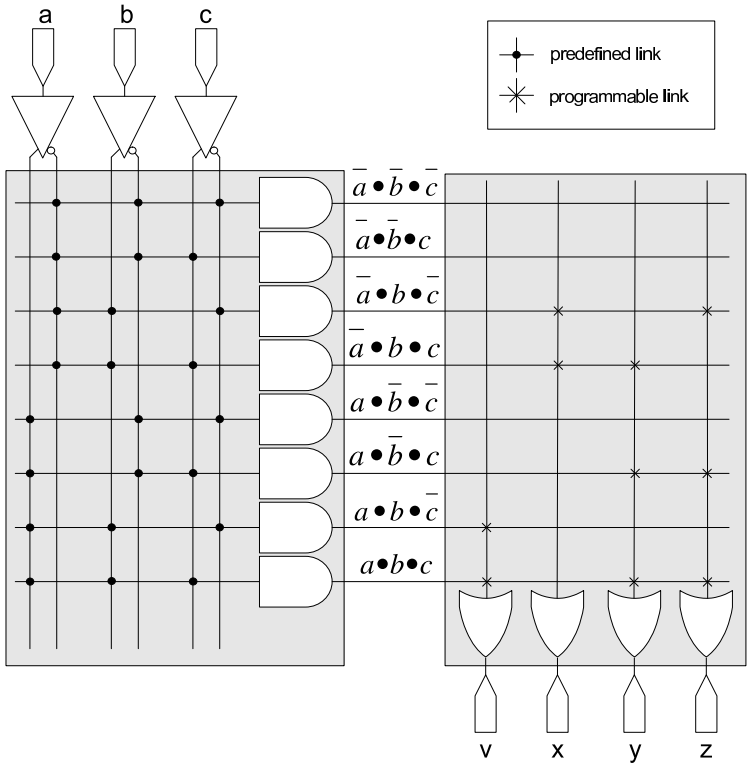

Figure 2.2: The programmed PROM device

It is clear that every combination of inputs is always decoded creating a possibility of the usage of all terms of the input function. Therefore, PROMs are useful for circuits that realise functions with large amount of product terms. The main problem in PROM-based PLDs is the number of inputs because each additional input requires twice wider memory volume.

The programmable logic array (PLA)

In the middle of 1970s, the new way of PLDs realisation was invented. In the **programmable logic arrays (PLAs)** both matrices: the AND and the OR are configurable. What is the most important, the number of inputs does not influence the number of terms performed by the AND-array. Therefore, the device based

on the PLA technology could perform functions with relatively higher number of inputs in comparison to the PROM. An example of the unprogrammed PLA device is illustrated in the figure 2.3.

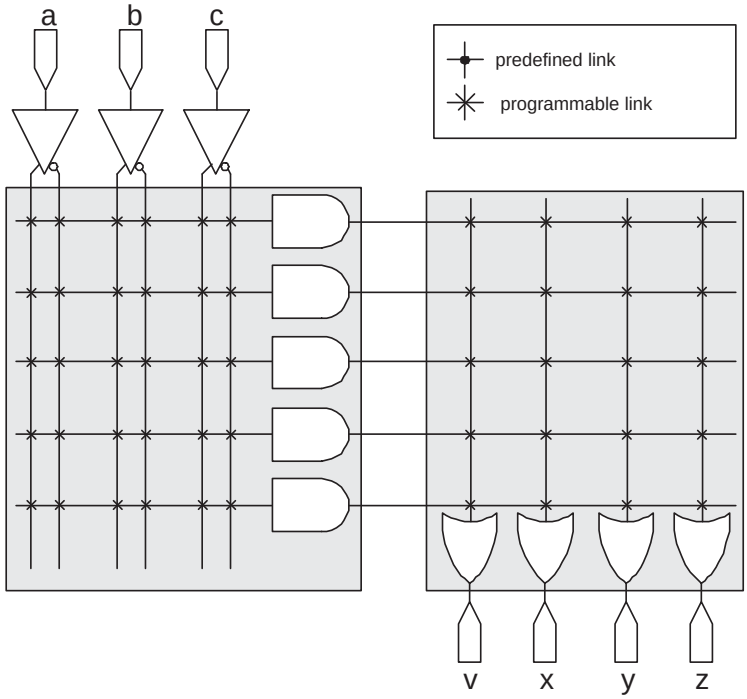

Figure 2.3: The unprogrammed PLA device

The cost of the device can be reduced by minimization of functions that ought to be realised (Dagless, 1983; Ciesielski and Yang, 1992; Yang and Ciesielski, 1989; Sasao, 1988). Because both matrices are programmable, the AND-matrix defines prime implicants while the OR-matrix sums all necessary implicants. Functions defined in the example (2.1) can be minimized and represented as:

$$\begin{aligned} v &= a \cdot b, \\ x &= \overline{a} \cdot b, \\ y &= a \cdot c + b \cdot c, \\ z &= a \cdot c + \overline{a} \cdot b \cdot \overline{c}. \end{aligned} \tag{2.2}$$

There are only four primary implicants needed to represent all functions. Therefore, the AND-matrix realises four minterms unlike in the PROM device where all eight lines were used. Furthermore, the OR-matrix sums the products generated by the AND-matrix using four programmable lines (fig. 2.4). Additionally, functions y and z use sharing of the minterm ac, thus the size of both matrices is reduced.

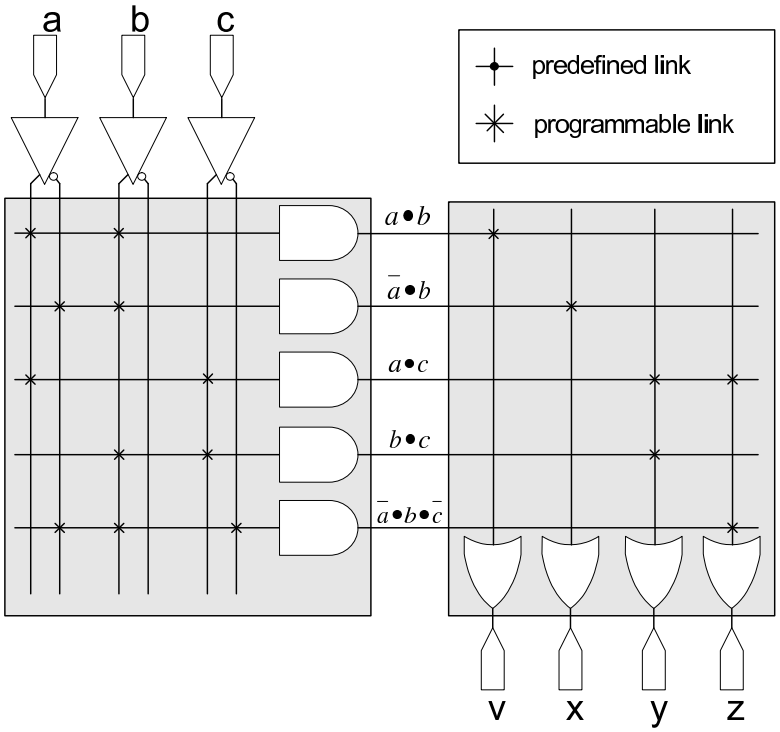

Figure 2.4: The programmed PLA device

The PLA is useful especially for large project where many common implicants are present in the design. However, because of its structure the PLA was relatively expensive to manufacture. Furthermore, the device was slow due to propagation delays that appeared in the programmable links. Therefore, in the late 1970s a new type of the PLD devices was proposed - the programmable array logic.

The programmable array logic (PAL)

The structure of the **programmable array logic (PAL)** is an opposite to the PROM structure. There is the programmable AND-matrix and the fixed OR-plane in the PAL device. An example of the unprogrammed PAL circuit is presented in the fig. 2.5.

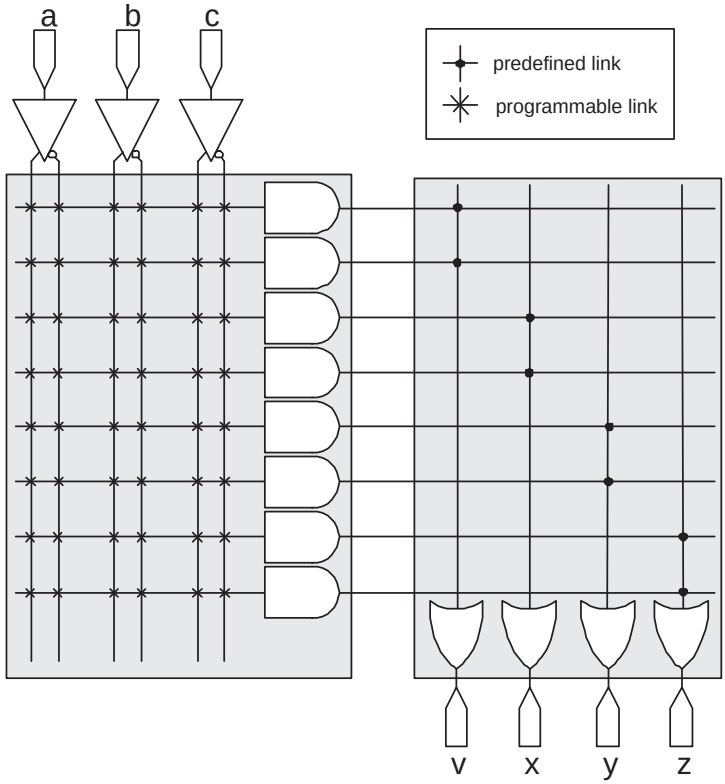

Figure 2.5: The unprogrammed PAL device

In the PAL only the first level of the device is configurable. Thus, any variation of input values in the AND-matrix may be defined by the designer. Such a combination creates product terms at outputs of the AND-plane. The OR-matrix is fixed so it can connect the restricted number of product terms for the realisation of logic functions. The figure 2.6 shows an exemplary implementation of functions 2.2. Similarly to the PLA, the AND-matrix is programmed to generate prime implicants on its outputs. Furthermore the OR-matrix generates proper functions using fixed OR-lines.

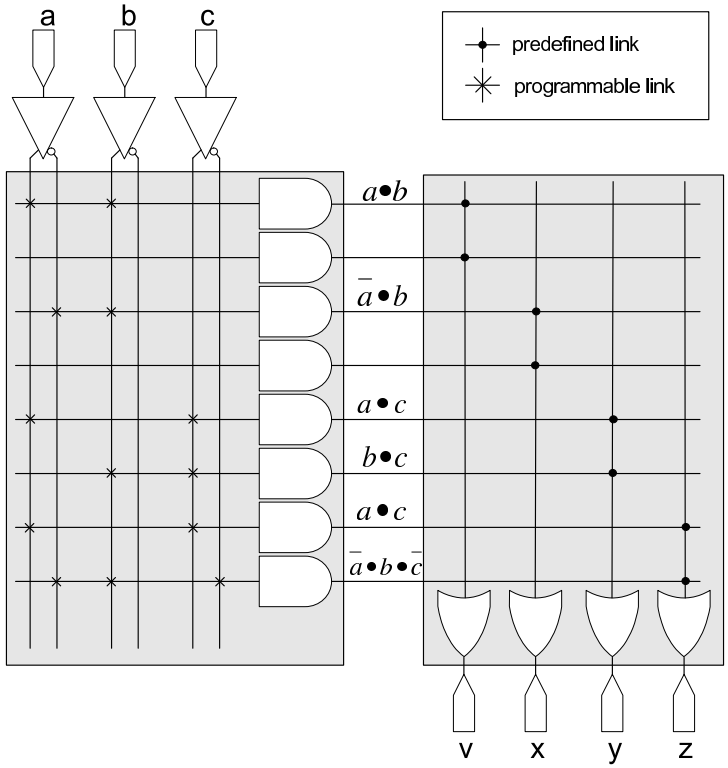

Figure 2.6: The programmed PAL device

PAL devices are not so flexible in configuration as PLAs. The structure of the device ought to be prepared and very often designers use some approaches to omit the restricted number of product terms that should be OR-ed (Kania, 2004; Kania and Kulisz, 2007; Hrynkiewicz et al., 1997). The main advantage of PAL devices in comparison to PLAs is their speed. There is only one programmable matrix in the PAL, thus the circuit is much faster in comparison with the PLA device. Another important benefit is price. Thanks to the low-cost and high speed of PAL devices became very popular in the late 1970s. However, the development of the prototyped systems effected appearance of the new branch in the programmable circuits: complex programmable logic devices.

2.1.3 Complex programmable logic devices (CPLDs)

The complex programmable logic device appeared at the beginning of the 1980s. The main idea was to connect several small PLD devices to create wider area for the programmable logic. First CPLDs have 100% connectivity to the inputs and outputs associated with each block (Maxfield, 2004). Therefore, the interconnection array was huge what made the whole device slow and expensive in production. The solution to this problem was the programmable interconnect array (PIA). Nowadays each vendor of CPLD devices has its own technology of CPLDs manufacturing, but the idea is similar: all SPLDs share the common PIA (Kania, 2004; Łuba, 2003; Maxfield, 2004). Figure 2.7 shows the structure of the typical CPLD device.

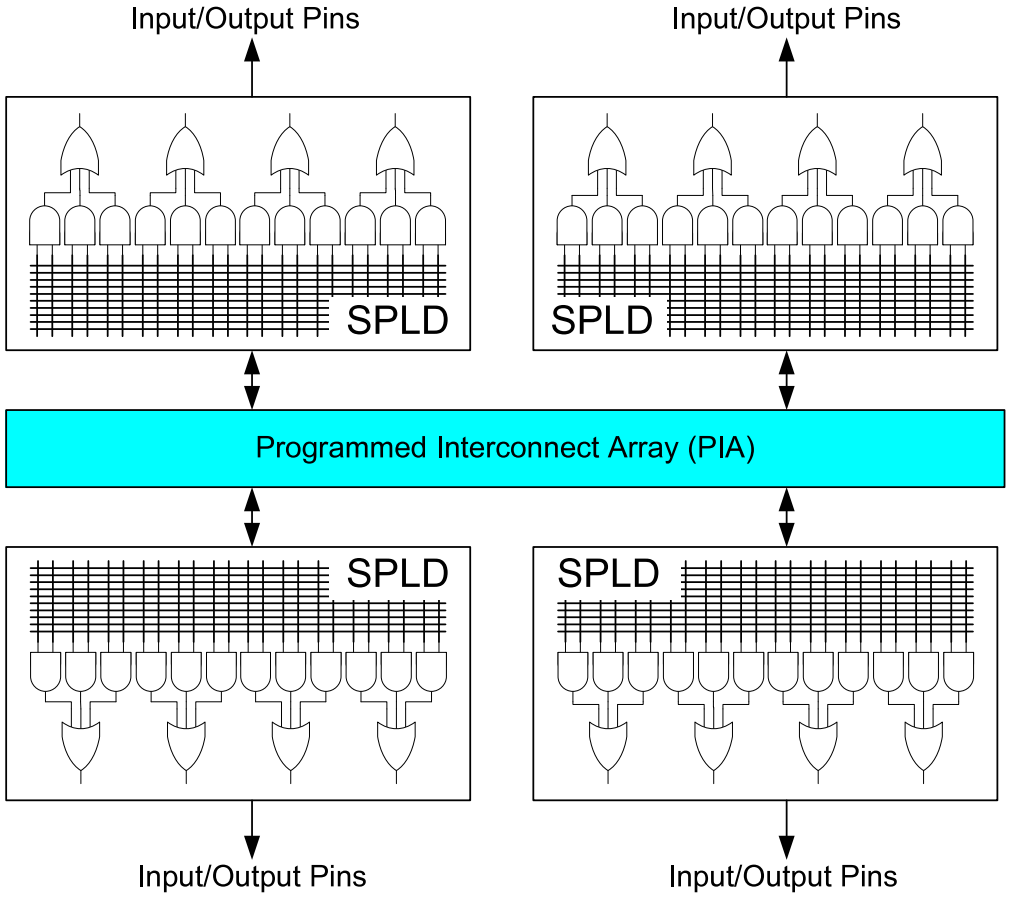

Figure 2.7: The CPLD structure

As it was mentioned, the typical CPLD consists of the programmable interconnect array surrounded by microcells (Kania, 2004; Łuba, 2003; Maxfield, 2004). The macrocell is built from AND-OR matrices (usually small PLDs, like PAL), programmable flip-flops and additional logic elements like multiplexers, OR and XOR gates. The most popular CPLDs that are currently available on the market are devices from Altera, Xilinx, Atmel, Lattice, Lucent, Cypres.

2.1.4 Field programmable gate arrays (FPGAs)

The first field programmable gate array appeared in the middle of the 1980s. Its structure was different in comparison to the CPLD (Xilinx, 2001; Altera, 2008; Maxfield, 2004; Jenkins, 1994). The main idea of FPGAs was to use programmable logic elements for implementation of simple logic functions. Such elements are called look-up tables (LUT) and can perform any logic function with the specific number of inputs and one output. Early FPGAs contained logic elements that could perform any logic function up to three inputs and one output. Additionally, each LUT was connected with a multiplexer and a register which created simplified programmable logic block. The idea of the early programmable logic block is illustrated in the fig. 2.8. As it was mentioned, the LUT could be configured to perform any combinational function with three inputs and one output. Sequential functions also could be realised thanks to the register connected with the LUT.

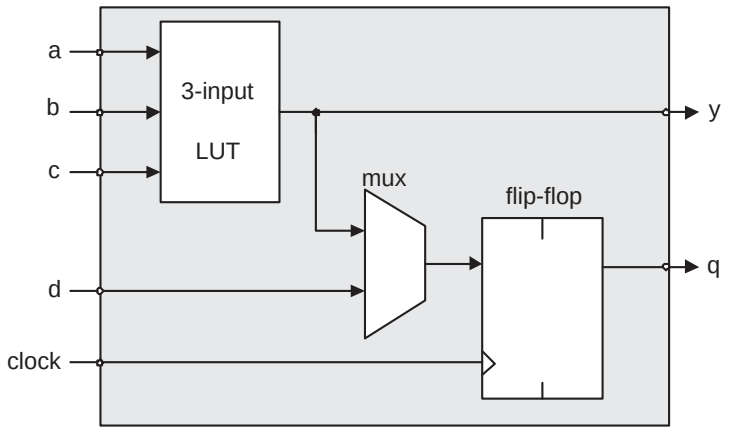

Figure 2.8: The early CLB structure

To perform functions that require more than one output (or more than three inputs) more logic blocks were used. Thanks to the structure of the FPGA, logic blocks are connected via interconnect matrix. Generally, the FPGA was created as "large number of logic blocks (islands), surrounded by a sea of programmable interconnections" (Maxfield, 2004).

Nowadays, there are many vendors of FPGA devices. The most popular are Xilinx, Altera, Lattice, Actel, Atmel. The structure of the FPGA depends on the vendor, however the idea is the same - usage of the array of logic blocks based on the LUT and the register. There are different names for logic blocks, interconnections and other elements of the FPGA because vendors ascribe new ideas in the branch to their inventions. In the dissertation, the structure of the FPGA will be described based on the Xilinx devices. Therefore, all references and information will concern Xilinx FPGAs.

The typical structure of the FPGA from Xilinx is shown in the fig. 2.9. The main elements of the device are: a matrix of configurable logic blocks (CLBs), configurable input/outputs blocks (IOBs) and dedicated memory blocks (BRAMs). All elements are connected via the programmable interconnect matrix.

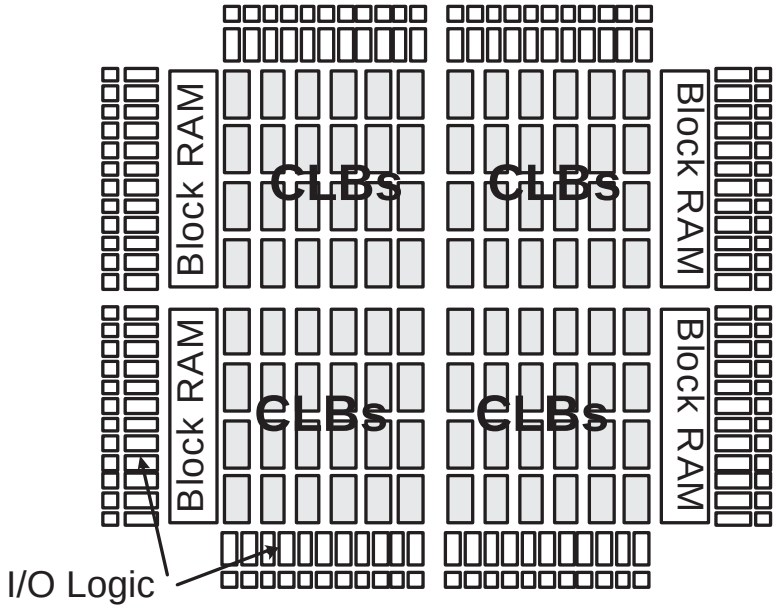

Figure 2.9: The structure of the typical FPGA device

The configurable logic block (CLB) consist of four **logic cells (LCs)**. Two logic cells are organized into a **slice**. Thus each CLB has two similar slices. Figure 2.10 illustrates the idea of the CLB structure.

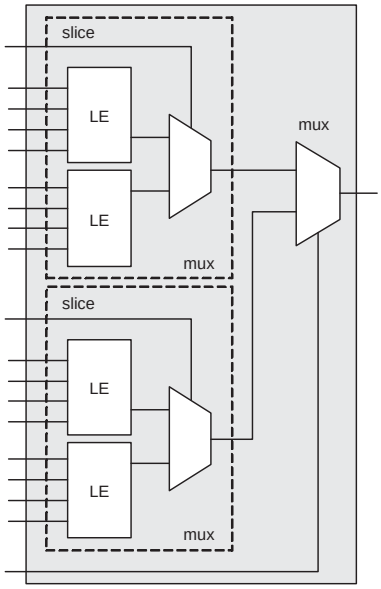

Figure 2.10: The structure of the CLB block

The main elements of the logic cell are: four-input and one-output LUT, multiplexer and flip-flop. Each family also has the additional logic (like carry logic for arithmetic operations) however, it is not important for further consideration in the dissertation. The simplified structure of the LE is presented in the fig. 2.11. The register can be configured either as the flip-flop or latch. What is important, the polarity of the clock may be rising-edge or falling-edge. Thus each LE could be configured as the active-low or active-high clock trigger.

The look-up table has four inputs and one output. Therefore it can realise any logic function that has maximum four input variables. Larger functions ought to be decomposed and more LUTs (or even slices) have to be used. Additionally, the look-up table can be configured as the 16x1 RAM (random-access memory) or 16-bit shift register.

The main role of **the configurable input/output block (IOB)** is to ensure the connectivity beetween the FPGA and other elements of prototyped systems. Thus very essential fact is the wide variety of the power supply standards

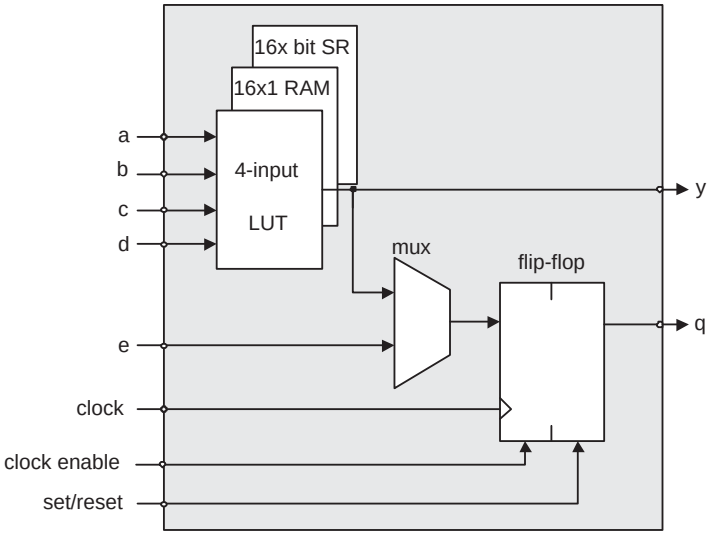

Figure 2.11: The structure the Logic Element (LE)

(Maxfield, 2004). IOBs are organized into banks (the number of banks depends on the FPGA). Each bank may be configured independently so the designer can connect to the FPGA other devices, with different input/output standards.

Dedicated memory blocks are very important components of the FPGA. Nowadays, a lot of designs contain memory that can be implemented with dedicated memory blocks (Bursky, 1999). Each vendor has its own concept of dedicated memory blocks. In case of Xilinx devices there are **Block-RAMs (BRAMs)**. Such elements are synchronous, therefore clock signal ought to be delivered. The number of BRAMs and the number of logic cells per each block depends on the particular device. BRAMs are organized into columns and each BRAM can be used separately. The main advantage of BRAMs is their size and reconfigurability. Depending on the device, there can be stored even up to three mega bits in the memory (the XC2V8000 device from the Virtex-II family). Additionally, BRAMs may be very easily reconfigured by the process of partial reconfiguration. The content of one (or more) BRAM is replaced while the rest of the device remains unchanged. Such a solution reduces the size of the destination bit-stream used to configure the FPGA. Moreover, the designing and configuration time is highly reduced as well. Partial reconfiguration of the control units implemented on FPGAs is described in Chapter 6 in more detail.

2.2 Control units

A digital system may be represented by a composition of the **control unit (CU)** and **operational unit (OU)** also known as a data-path (Gajski, 1997; De Micheli, 1994; Barkalov and Węgrzyn, 2006). The idea of such a defined digital system is illustrated in the fig. 2.12.

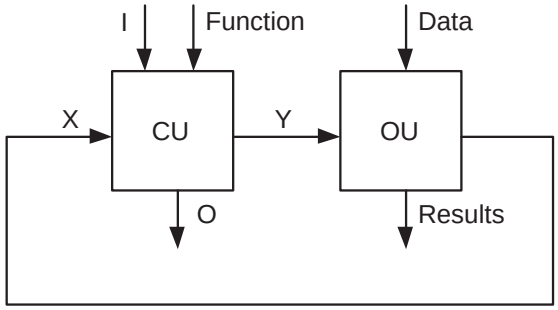

Figure 2.12: The model of the digital system

Based on the set of input values (**I**) and set of logic conditions (**X**), the CU sends proper microoperations (**Y**) to the OU. Additionally, a set of output values (**O**) is generated. The set of inputs (I) and set of outputs (O) are used for communication with the environment of the digital system (Łuba, 2001; Molski, 1986).

The operational unit executes microoperations (Y) by processing proper input (**Data**) and generating results (**Results**). Additionally the OU generates logic conditions (X) for the control unit.

2.2.1 Single-level control units (finite state machines)

The most popular realisation of control units nowadays is an **finite state machine (FSM)** also known as the finite state automaton (Łuba, 2003; Baranov, 1994; Barkalov and Węgrzyn, 2006; Traczyk, 1982; Adamski et al., 2007; Adamski and Węgrzyn, 2003; Adamski, 1980). The FSM is a model of behavior that consists of the set of states, set of transitions between states and set of actions (microoperations). Formally the FSM can be described as a 6-tuple vector:

$$M =< S, X, Y, f, h, s_0 >, \tag{2.3}$$

where:

- $S = \{s_0, s_1, \ldots, s_K\}$ is non-empty finite set of states;
- $X = \{x_0, x_1, \ldots, x_L\}$ is finite set of inputs;
- $Y = \{y_0, y_1, \ldots, y_N\}$ is finite set of outputs;
- $f : S \times X \rightarrow S$ is the transition function, this function determines the next state $s_s \in S$ depending on the current state $s_m \in S$ and on the value of input $x_l \in X$;
- $h : S \times X \rightarrow Y$ is the output function, this function determines the current output $y_n \in Y$, based on the current state (in case of Moore automaton) or depending on the current state and the current input (in case of Mealy automaton);
- $s_0 \in S$ is the initial state of automaton.

Figure 2.13 shows the typical realization of the finite state machine (Baranov, 1994; Barkalov and Węgrzyn, 2006). There are two main units in the FSM. The combinational circuit CC generates proper output values (microinstruction) and indicates excitation functions for the register RG which is in charge of holding internal state $s_m \in S$ of an FSM.

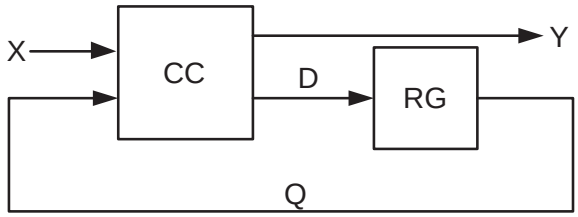

Figure 2.13: The model of the finite state machine

The FSM can be realised as **Moore** or **Mealy** automaton. If the control unit is described as **Moore** FSM, then outputs depend on the current state of the automaton (Moore, 1956). Microinstruction is represented as:

$$Y_t = f(s_m), \tag{2.4}$$

where Y_t means the value of the output and $s_m \in S$ represents the current state of the FSM. The main advantage of such a realisation is simplification of the behaviour of the control unit. States are clearly tied with the action generating proper microinstruction while inputs (conditions) influence only transitions between states.

The second way of implementation of the FSM is **Mealy** automaton (Mealy, 1955). The value of outputs depends not only on the current state but also on input signals:

$$Y_t = f(s_m, X), \tag{2.5}$$

where X is a set of input values (conditions).

The main benefit offered by Mealy FSM is the reduction of the number of internal states of the automaton in comparison with Moore FSM. Both Moore and Mealy automata are classified as **single-level control units**.

The optimization of the FSM is one of the most popular tasks nowadays. There are many ideas focused on improving the prototyping process and encoding of internal states of the automaton (Sentovich et al., 1992*a*; Hrynkiewicz et al., 1997; Ashar et al., 1990; Ashar et al., 1992; Kubátová, 2005; Perkowski et al., 2001; Rawski et al., 2003; Barkalov, 1998; Ahmad et al., 2000). Above researches benefited in appearance of computer-aided design systems, like *Sequential Circuit Synthesis, SIS* (Sentovich et al., 1992*b*). It contains algorithms for state assignments (*NOVA*, *JEDI*) and state minimization (*STAMINA*).

The next subsection deals with microprogram control units where outputs of the controller are organized in microinstructions.

2.2.2 Microprogram control units

The idea of microprogramming was introduced by M. V. Wilkes in 1951 as an intermediate level to execute computer program instructions (Wilkes, 1951; Molski, 1986; Traczyk, 1982; Husson, 1970; Kravcov and Chernicki, 1976; Misiurewicz, 1982; Papachristou, 1979; Stalings, 1996). Microprograms were organized as a sequence of microinstructions and stored in the special control memory (CM). The algorithm for the MCU is usually specified by the **flow-chart (FC)** description (Łuba, 2005; Barkalov and Węgrzyn, 2006). Such a flow-chart algorithm consists

of four main types of vertices (start, end, operational vertex, conditional vertex) that are interpreted by the control unit. More information about the flow-chart algorithm can be found in Chapter 3.

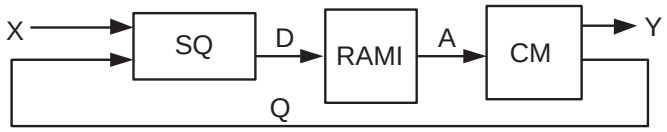

Figure 2.14: The model of the microprogram control unit (MCU)

Typical structure of the MCU is presented in the fig. 2.14. There are three main blocks that consist of the MCU: a sequencer SQ, a register of the microinstruction address RAMI and the control memory CM (Łuba, 2005; Barkalov and Węgrzyn, 2006). The sequencer is the combinational circuit that forms the excitation function for the RAMI:

$$D = f(X, T). \tag{2.6}$$

Here X is a set of logic conditions of the system and T is a feedback function generated by the control memory. Based on this function, the RAMI generates the proper microinstruction address A. The control memory CM holds microprogram that is further executed by the operational part of the system. There are different methods of microinstructions addressing (Barkalov and Palagin, 1997; Łuba, 2003), however in most cases the CM also keeps addresses of next microinstructions that should be executed. The feedback function T is analysed by the sequencer which based on the condition from the set X selects the proper address A.

There are many designing ways of the MCU (Łuba, 2005; Adamski and Barkalov, 2006). One of the most popular is to perform the sequencer as the multiplexer, and the RAMI as the counter (Łuba, 2005). Then the structure of the MCU may be interpreted as the system shown in the fig. 2.15.

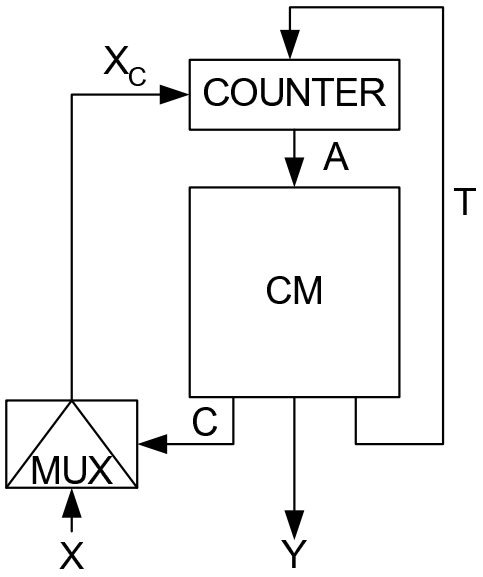

Figure 2.15: The model of the microprogram control unit with counter

In the MCU presented in the fig. 2.15, the CM generates two feedback functions - T for the counter and C for the multiplexer. Such a realisation is especially fruitful in case of long segments (chains) of microinstructions (Łuba, 2001). Then the chain of microinstructions that are not separated by the condition may be replaced by one state (block). Chapter 3 deals with the usage of chains of microinstructions in more detail.

The main advantage of the microprogram control unit is simplicity of its structure. Outputs of the controller are organized in microinstructions and they can be easily replaced. Additionally, the control memory may be implemented using dedicated memory blocks of the FPGA reducing the area of used logic elements.

Apart from its benefits, the MCU has some disadvantages. The control memory holds not only microoperations but also information for calculation of the address of the next microinstruction. Very often the size of the control memory exceeds the size of the dedicated memory block of an FPGA. To eliminate these disadvantages, the control unit may be decomposed and designed as a **Compositional Microprogram Control Unit (CMCU)**.

Chapter 3

Compositional microprogram control units

Any flow-chart of algorithm can be interpreted as the compositional microprogram control unit (Barkalov, 2002). In the CMCU the control unit is decomposed into two main parts. The first is responsible for addressing microinstructions that are kept in the control memory. The role of the second part is to hold and generate adequate microinstructions.

The control unit may be decomposed in two ways. The first one is **functional decomposition**. Here the controller is decomposed basing on its internal functions and states. The second method is **structural decomposition** where the task of the decomposition is reached thanks to the modification of the structure of the control unit. Both ideas of the decomposition of the control unit lead to the **compositional microprogram control unit**.

3.1 Functional decomposition of control units

Functional decomposition is the process that splits the complex function into the smaller sub-functions (Kania et al., 2005; Łuba, 2005; Devadas et al., 1988; Sasao, 1999; McCluskey, 1986; Rawski et al., 2005*b*; Scholl, 2001). Such a realisation is often used as a part of logic synthesis of designs implemented with programmable devices. Functional decomposition is widely expanded especially by academic organizations (Sentovich et al., 1992*b*; Kania and Kulisz, 2007; Kania, 2007; Łuba

et al., 2002; Rawski et al., 2001). The dissertation focuses on the decomposition of control units implemented on the FPGA. Optimization of SPLDs and CPLDs can be found in (Kania, 2004; Kania, 1999; Ciesielski and Yang, 1992; Devadas et al., 1988; Devadas et al., 1989; Muthukumar et al., 2007; Sasao, 1999; Sasao, 1999; Sentovich, 1993; Solovjev, 1996).

In the FPGA, the limited number of inputs and only one output of LUT elements make functional decomposition very effective (Scholl, 2001; Łuba, 2005; Rawski et al., 2003; Łach et al., 2003; Pasierbiński and Zbysiński, 2001). The idea of functional decomposition is widely used either by commercial (Xilinx, Altera, Synplicity, etc.) and non-commercial organisations (Universities). It should be pointed out that the best results are achieved by non-commercial projects such as DEMAIN (Technical University of Warsaw) or SIS (University of Berkeley).

Functional decomposition may be realised as **serial decomposition** or **parallel decomposition**. In the first one, the set X of input variables is split into two subsets U and V (Łuba, 2003).

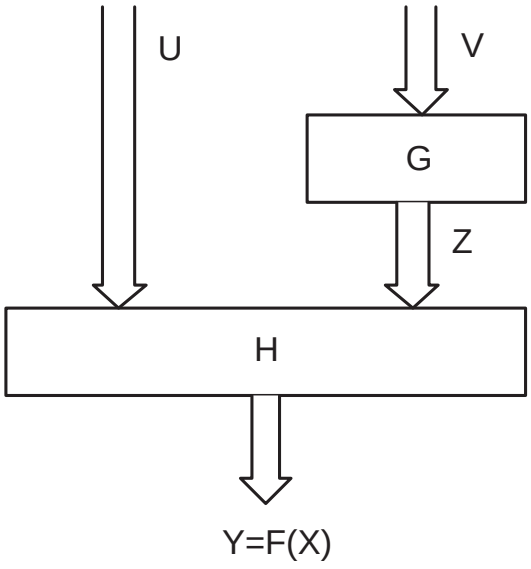

Figure 3.1: The idea of serial functional decomposition

The set V forms inputs for the function G which generates the set of outputs Z=$G(V)$. Of course the method has sense only if the number of outputs of the function G is fewer than the number of its inputs. Furthermore, outputs Z gener-

ated by G and the set U form inputs for the function H (3.1). Finally, function F is represented as follows:

$$F = H(U, G(V)). \tag{3.1}$$

The aim of parallel decomposition is to decompose the initial function F into two separate sub-functions G and H (Łuba, 2005). The main idea is to split the set of outputs Y into two subsets Y_G and Y_H. Here Y is the set of outputs of the function F. Similarly Y_G is the set of outputs of the function G and Y_H - the set of outputs of H. The method has sense if either one of functions G or H has fewer input variables than the initial function F. The idea of parallel decomposition is illustrated in the 3.2.

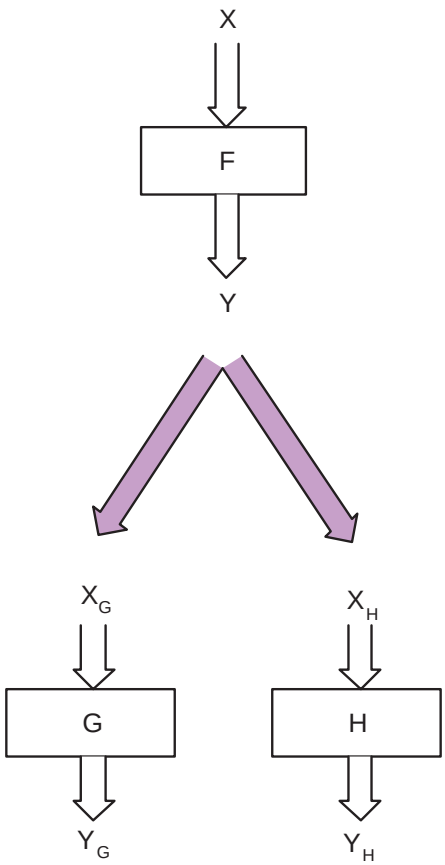

Figure 3.2: The idea of parallel functional decomposition

Serial and parallel decompositions are very often combined. **Balanced decomposition** joins both methods (Łuba, 2005; Rawski et al., 2005*a*; Selvaraj and Luba, 1995). The whole process may be divided into steps. In each step either serial or parallel decomposition is performed. The process is repeated until the satisfactory result is reached (Łuba, 2005).

Presented methods of decomposition are fruitful for combinational blocks of the system. However they can also be used in decomposition of control units (Łuba, 2005; Rawski et al., 2003; Borowik, 2004; Borowik, 2005). The controller may be realised as the sequential circuit shown in the fig. 3.3. The main idea is to use the control memory to hold microinstructions. Such a memory is implemented with dedicated memory blocks of the FPGA, which reduces the logic resources of the device.

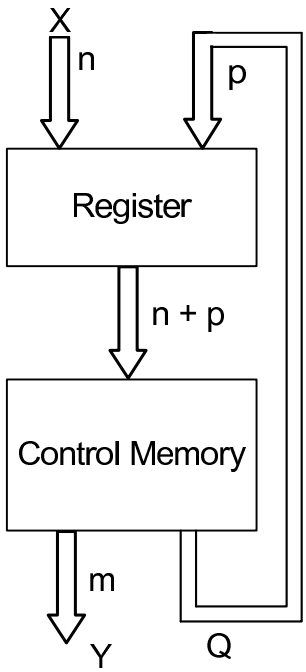

Figure 3.3: The control unit realised as the sequential circuit

Each microinstruction of the control unit presented in the fig. 3.3 consists of two fields. The first one holds the code Q of internal states of the automaton, while the second contains output variables from the set Y. The next state of the controller is determined by input variables X and by the current state of the control unit.

The width of the address of the control memory may be calculated as $|A|=n+p$, where n means the number of the input variables and p represents the number of bits that are used for encoding internal states of the controller (Łuba, 2005). The volume of the control memory depends on the width of its address. Each additional bit doubles the volume of the memory. Thus, very often such a volume exceeds the volume of dedicated memory blocks of the FPGA. The solution to this problem may be functional decomposition of the control unit (fig. 3.4).

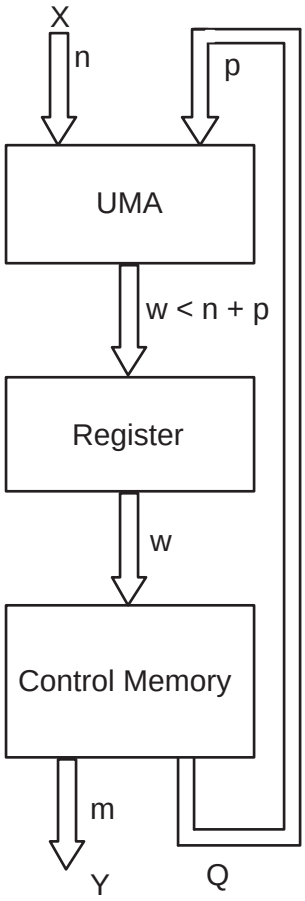

Figure 3.4: Functional decomposition of the control unit

In the system shown in the fig. 3.4 the control memory is decomposed into two parts: block of the address modification (CAM) and smaller memory that may be realised using the dedicated memory block of the FPGA. There are many variants

of such a decomposition (Rawski et al., 2003; Borowik, 2004; Borowik, 2005). The main aim of all methods is to decrease the size of the control memory using the minimum number of logic blocks of the FPGA.

The main benefit of the functional decomposition of the control unit is very high effectiveness. The memory may be decomposed in such a way that dedicated memory blocks of the FPGA are used to the maximum. In the other words the minimum number of logic blocks are used to realise the circuit of the address modification. On the another hand, only a part of a microinstruction is held in the memory after the decomposition. Therefore, there is no possibility to apply the idea of partial reconfiguration which can significantly accelerate the prototyping process of the control unit (Chapter 6).

3.2 Structural decomposition of control units

This section deals with structural decomposition of control units and it is an introduction to the main part of work presented in the dissertation. Thus, the idea of CMCUs created as structural decomposition of the controller will be described in more details. The section presents main definitions and base structure of the CMCU, which was inspiration and motivation of a PhD Thesis. Next sections show Author's methods of CMCUs synthesis, however all structures are based on ideas shown below.

3.2.1 Main definitions

This section introduces some definitions that will be needed later in order to describe the CMCU more formally.

Let the control algorithm (De Micheli, 1994) of a digital system (Adamski and Barkalov, 2006; Barkalov and Węgrzyn, 2006; Gajski, 1997) be represented as the flow-chart Γ (Baranov, 1994) with a set of operational vertices $B = \{b_1, \ldots, b_K\}$ and a set of edges E. Each vertex $b_k \in B$ contains microoperations $Y(b_k) \in Y$, where $Y = \{y_1, \ldots, y_N\}$ is the set of microoperations. Each conditional vertex of the flow-chart contains one element from the set of logic conditions $X = \{x_1, \ldots, x_L\}$.

Defininition 3.1. The operational linear chain (OLC) of the flow-chart Γ is a finite sequence of operational vertices $\alpha_g = \langle b_{g1}, \ldots, b_{gFg} \rangle$ such that for any pair of adjacent components of the vector α_g there is an edge $\langle b_{gi}, b_{gi+1} \rangle \in E$, where i is the number of the component in the vector α_g $(i = 1, \ldots, F_g - 1)$.

Defininition 3.2. The vertex $b_q \in B$ is called as an **input of the OLC** α_g if there is the edge $\langle b_t, b_q \rangle \in B$, where b_t is either initial or conditional vertex of the flow-chart Γ or operational vertex that does not belong to the OLC α_g.

Defininition 3.3. The vertex $b_q \in B$ is named as an **output of the OLC** α_g if there is the edge $\langle b_q, b_t \rangle$, where b_t is either conditional or final vertex of the flow-chart Γ or operational vertex that does not belong to the OLC α_g.

Defininition 3.4. The parameter M_1 is equal to the number of vertices in the longest OLC α_g of the flow-chart Γ.

Defininition 3.5. The minimum number of bits required to encode the variable M_1 is represented as R_1 and it is equal to: $R_1 = \lceil log_2 M_1 \rceil$.

Defininition 3.6. The parameter M_2 is equal to the number of all operational chains presented in the flow-chart Γ.

Defininition 3.7. The minimum number of bits required to encode the variable M_2 is represented as R_2 and it is equal to: $R_2 = \lceil log_2 M_2 \rceil$.

Defininition 3.8. The parameter M_3 is equal to the number of all operational vertices in the flow-chart Γ. This parameter also indicates the total number of microinstructions of the CMCU.

Defininition 3.9. The minimum number of bits required to encode the variable M_3 is represented as R_3 and it is equal to: $R_3 = \lceil log_2 M_3 \rceil$.

3.2.2 The CMCU with base structure

Let D^g be a set of operational vertices that are included in the chain α_g. Then let $C = \{\alpha_1, \ldots, \alpha_G\}$ be a set of OLCs of the flow-chart Γ satisfied to the condition:

$$\begin{aligned} &D^g \cap D^q = \emptyset \ (g \neq q; g, q \in \{1, ..., G\}); \\ &B = D^1 \cup D^2 \cup ... \cup D^G; \\ &D^g \neq \emptyset \ (g = 1, ..., G). \end{aligned} \tag{3.2}$$

Let natural addressing of microinstructions be executed for each α_g:

$$A(b_{gi+1}) = A(b_{gi}) + 1 \ (i = 1, ..., F_{g-1}), \tag{3.3}$$

where $A(b_g)$ is an address of the microinstruction corresponding to the vertex $b_g \in B$. In this case the flow-chart Γ can be interpreted as a CMCU with base structure denoted in future as the CMCU U_{BS} (fig. 3.5).

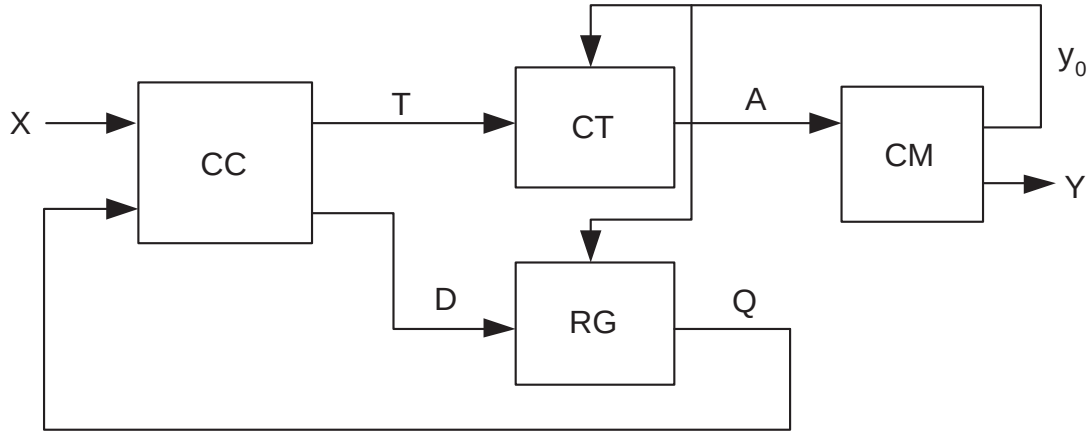

Figure 3.5: The compositional microprogram control unit with base structure

There are four main modules in the CMCU U_{BS}: the combinational circuit CC, the register RG, the counter CT and the control memory CM. The combinational circuit and the register represent the simplified FSM of microinstructions addressing S_1. Furthermore, the counter and the control memory form the microprogram control unit S_2. The RG keeps a code $K(a_m)$ of the current state $s_m \in S$ of the CMCU, where S=$\{s_1, \ldots, s_M\}$ is a set of internal states. The register has $\lceil log_2 M_2 \rceil$ flip-flops and their outputs $q_r \in Q$ are used to encode states $s_m \in S$, here Q=$\{q_1, \ldots, q_{R_2}\}$, $|Q| = R_2$. The transition from the state $s_m \in S$ to the state $s_s \in S$ is executed by switching the register from the code $K(a_m)$ to the code $K(a_s)$. Such a switching is determined by the excitation function $Q_r \in Q$. The CT keeps the address $A(b_t)$ of the microinstruction $Y(b_t)$ that is executed by a data-path (Barkalov, 2002). Variables $a_r \in A$ are used for the representation of the addresses $A(b_k)$. Microinstructions are kept in the CM having 2^{R_1} words. Each word (microinstruction) has N+2 bits in a case of one-hot encoding of microoperations (Barkalov et al., 2005e; Barkalov and Wiśniewski, 2004a). One of additional

bits is used to keep an variable y_0 to organize the mode of addressing (3.3). The second bit keeps a variable y_K to terminate the fetching of microinstructions from the control memory (to clarify CMCUs structures, y_K is not marked in figures presented in the dissertation).

The presented CMCU U_{BS} operates in the following manner: at the beginning the register is set to the value that corresponds to the initial state of the FSM. Similarly, the counter is set to the address of the first microinstruction. If transitions are executed inside the OLC $\alpha_g \in C$, then $y_0 = 0$ which causes the increment of the CT and forbids changing the state of the CMCU. When the output of the OLC $\alpha_g \in C$ is reached then $y_0 = 1$ and the combinational circuit forms the excitation function for the register setting it into the proper state (Barkalov and Wiśniewski, 2004b; Barkalov et al., 2004; Barkalov and Wiśniewski, 2004d). Similarly the counter is set with the proper value as well:

$$D = f(Q, X), \tag{3.4}$$

$$T = f(Q, X). \tag{3.5}$$

Here X means the set of conditions, Q is the set of internal variables used to encode the current state of the CMCU, D is a set of variables that form an excitation function for the register D=$\{d_1, \ldots, d_{R_2}\}$ and T is a set of variables that form an excitation function for the counter T=$\{t_1, \ldots, t_{R_1}\}$.

Functions (3.4) and (3.5) form a code $K(s_m)$ of the state of the transition in the register and an address of the input of the next OLC$\alpha_g \in C$. If the CT contains the address of the microinstruction $Y(b_k)$ such as $\langle b_k, b_E \rangle \in E$, then $y_K = 1$. In this case the operation of the CMCU U_{BS} is finished (Barkalov and Wiśniewski, 2004f; Barkalov and Wiśniewski, 2004c; Barkalov et al., 2005*a*).

The main benefit of the realisation of the controller as the compositional microprogram control unit U_{BS} is a possibility of implementation of the circuit CM using dedicated memory blocks (Wiśniewski, 2005). Other blocks of the prototyping system U_{BS} are implemented with the logic blocks (flip-flops and LUT elements) of the FPGA (Łuba, 2003; Xilinx, 2005; Altera, 2008). Such an idea lead to reduction of the number of logic blocks in comparison with the realisation of the controller as a traditional finite state machine and thus, the designer can allocate wider area of the FPGA for another blocks of the prototyping system.

The effectiveness of the CMCU is especially high if the controller interprets the linear flow-chart. Such flow-chart contains 75% of operational vertices or includes long linear chains (segments) of operational vertices.

The second advantage of the CMCU is possibility of selecting the implementation method of the control memory. The designer can decide if the circuit CM should be realised with logic blocks or with dedicated memory blocks. It is important especially in case of designs, which consumes large area of the memory. Then the whole CMCU is implemented with logic blocks of the FPGA.

In opposition to functional decomposition, structural decomposition of a control unit permits to apply the idea of **partial reconfiguration** (Wiśniewski, 2005; Barkalov et al., 2006; Mesquita et al., 2003). In this case, only a part of the controller (the control memory) can be replaced while the rest of the system remains untouched. Partial reconfiguration of control units implemented in the FPGA is widely described in Chapter 6.

3.3 Conclusions

There were two ideas of the decomposition of control units presented in this Chapter. The aim of both methods is to decompose the controller into two main modules. The first is in charge of addressing microinstructions that are hold in the control memory. Functional decomposition of the CU bases on the realisation of circuit addressing using divisions of Boolean functions. In structural decomposition additional internal blocks are added to the structure of the control unit.

The idea of structural decomposition of the control unit presented in this Chapter was a base for development of the new synthesis methods of the CMCU. In the (Barkalov, 2002) Professor A. Barkalov introduced two new significant ways of the control units: the CMCU with mutual memory and the CMCU with sharing codes. The aim of both methods was to reduce the number of logic blocks required for implementation of the controller. Experiments showed that either implementation of the CMCU with mutual memory or the CMCU with sharing codes instead of the traditional CMCU with base structure, benefited with less area usage of programmable devices (Barkalov and Węgrzyn, 2006; Adamski and Barkalov, 2006; Barkalov, 2002). However, thanks to the modification in the structure, there is still a possibility to reduce the number of logic blocks required for

implementation of the controller. Thus, both methods - the CMCU with mutual memory and the CMCU with sharing codes - were an inspiration for researches of new designing ways of control units. Six new structures and synthesis methods of the CMCU are shown in the dissertation. All presented methods are divided into two parts. The first group is based on the CMCU with mutual memory and it is described widely in Chapter 4. The second group is based on the CMCU with sharing codes and it is shown in the Chapter 5.

Chapter 4

Compositional microprogram control units with mutual memory

This Chapter deals with the compositional microprogram control unit with mutual memory. The main idea is to recognize each operational linear chain by an address generated by the counter. Now the code produced by the counter indicates the current state of the control unit. Therefore, usage of the register in the CMCU with mutual memory is unnecessary.

The first section describes the traditional synthesis method of the CMCU with mutual memory that was initially proposed by in (Barkalov and Palagin, 1997). Next sections show three new synthesis methods of the CMCU with mutual memory. The main idea was to reduce the number of logic blocks (especially LUT elements) that are required for implementation of the system in the FPGA.

4.1 The CMCU with mutual memory

The structure of the CMCU U_{MM} with mutual memory is presented in the fig. 4.1. There are three main blocks in the CMCU U_{MM}: the combinational circuit CC, the counter CT and the control memory CM (Barkalov et al., 2005*b*).

Distinct from the CMCU U_{BS} with base structure, in the CMCU U_{MM} the combinational circuit generates the excitation function only for the counter:

$$T = f(X, A), \tag{4.1}$$

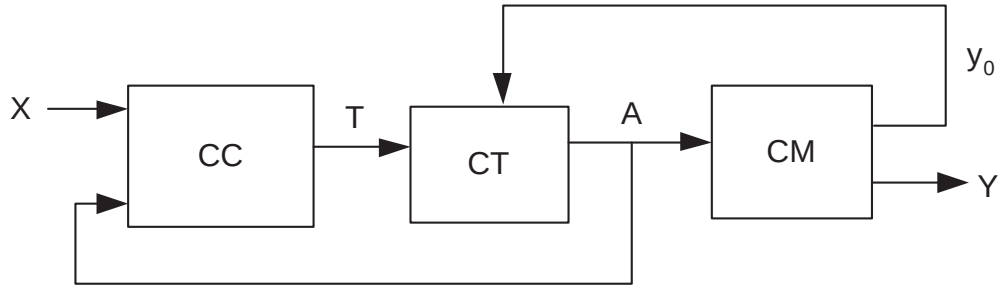

Figure 4.1: The structure of the CMCU with mutual memory

where X means the set of conditional vertices and A means the code that is determined by the counter. Such a code is also the address of the microinstruction that is kept in the control memory. The number of logic functions is decreased in comparison with the CMCU U_{BS}, because the circuit CC doesn't generate the excitation function for the register. Thus the number of logic blocks of the destination programmable device is reduced (Adamski and Barkalov, 2006; Wiśniewski and Barkalov, 2007; Barkalov et al., 2006*c*; Wiśniewski et al., 2007).

4.1.1 The main idea of the method

In the CMCU U_{MM} transitions between internal states of the controller are performed in the different way than it is in the CMCU with base structure. Here the address generated by the counter is used to recognize the proper state of the control unit.

The controller operates as follows: at the beginning, the counter is set to the value that corresponds to the initial state of the FSM which is equal to the address of the first microinstruction. If transitions are executed inside the $\alpha_g \in C$, then $y_0 = 0$. It causes the incrementation of the CT and forbids to change the current state of the control unit. When the output of the $\alpha_g \in C$ is reached, $y_0 = 1$ and the circuit CC forms the excitation function for the counter (4.1). This function forms the code $K(s_s)$ of the state of transition and the address of the input of the next OLC $\alpha_g \in C$ as well. If the controller reaches an address of the microinstruction $Y(b_k)$ such as $\langle b_k, b_E \rangle \in E$, then $y_K = 1$. In this case, operation of the CMCU U_{MM} is finished.

4.1.2 Synthesis of the CMCU with mutual memory

The method of synthesis of the CMCU with mutual memory includes the following steps:

1. ***Formation of the set of OLCs.*** At the beginning, the set of operational linear chains is created. For each OLC all outputs and inputs are determined. This step is executed according to the definition 3.2 and 3.3.

2. ***Formation of the content of the control memory.*** In order to perform the formation of the content of the control memory, microinstructions and their addresses ought to be encoded. In case of the CMCU U_{MM} the natural binary codes will be used. Next the control memory is formed. Each microinstruction consists of all (N) microoperations that belong to the initial flow-chart and two additional bits: y_0 and y_K (value 1 means that microoperation belongs to the microinstruction). Therefore, the volume of the control memory can be calculated as S_{CM}=(N+2)$*2^{R_1}$, where R_1 is the width of the microinstruction address generated by the counter.

3. ***Formation of the transition table of the CMCU U_{MM}, formation of the excitation function for the counter.*** At this stage, the table of transitions between the OLCs is created. The table contains the following columns: O_g,$SA(O_g)$,X_h,I_j^t,$K(I_j^t)$,T,h, where:

   - O_g is the output from which the transition is executed;
   - $SA(O_g)$ is the address of the output O_g;
   - X_h is the input signal causing the transition $< O_g, I_t^j >$ and it is equal to the conjunction of the elements from the set X;
   - I_j^t is the input of the destination chain, $\alpha_j \in C$, where the transition is executed;
   - $K(I_j^t)$ is the address of the input I_j^t;
   - T is the set of variables that form the excitation function for the counter;
   - h is the number of the transition (h=1, ..., H).

Based on this table the excitation function T for the counter is formed:

$$T_r = \bigvee_{h=1}^{H} C_{rh} E_g^h X_h \ (r = 1, \ldots, R_1). \tag{4.2}$$

Here C_{rh} is a Boolean variable that is equal to 1, if and only if the function T_r is written in the h-th line of the table of transitions; E_g^h is a conjunction of internal variables $A_r \in A$ corresponding to the address $SA(O_g)$ of the output O_g from the h-th line of the table of transitions.

4. ***Implementation of the CMCU*** U_{MM}**.** The controller may be implemented on the FPGA in two ways. The first one is to realize the control memory with dedicated memory blocks of programmable devices. Such a solution permits to decrease the number of used logic blocks of the FPGA. The second way is to implement the whole system using logic blocks of the device. This option is applied usually if the size of the control memory exceeds the size of available dedicated memory blocks of the FPGA.

4.1.3 Example of the synthesis method of the CMCU U_{MM}

To bring closer the idea of the CMCU with mutual memory, the synthesis method of such a controller will be illustrated by a simple example. Figure 4.2 illustrates the hypothetical algorithm of the control unit U_1. There are eleven operational $B = \{b_1, \ldots, b_{11}\}$ and three conditional $X = \{x_1, x_2, x_3\}$ vertices in the flow-chart Γ_1. Thus, the circuit should generate eleven microinstructions that consist of five microoperations $Y = \{y_1, \ldots, y_5\}$.

In order to design the CMCU with mutual memory, first the set C of operational linear chains ought to be formed (fig. 4.3). In the presented example there are four OLCs $C = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$, where $\alpha_1 = \langle b_1, b_2 \rangle$, $\alpha_2 = \langle b_3, \ldots, b_7 \rangle$, $\alpha_3 = \langle b_8, b_9 \rangle$, $\alpha_4 = \langle b_{10}, b_{11} \rangle$. All OLCs except α_2 have one input: for α_1 it is vertex b_1, for α_3 - b_3 and for α_4 - b_4. The OLC α_2 has two inputs: vertex b_3 and vertex b_6. Therefore the set of inputs contains five elements: $I = \{I_1^1, I_2^1, I_2^2, I_3^1, I_4^1\}$, where $I_1^1 = b_1$, $I_2^1 = b_3$, $I_2^2 = b_6$, $I_3^1 = b_8$, $I_4^1 = b_{10}$. Each OLC may have only one output, thus there are four outputs in the set of OLCs: $O = \{O_1, \ldots, O_4\}$, where $O_1 = b_2$, $O_2 = b_7$, $O_3 = b_9$, $O_4 = b_{11}$.

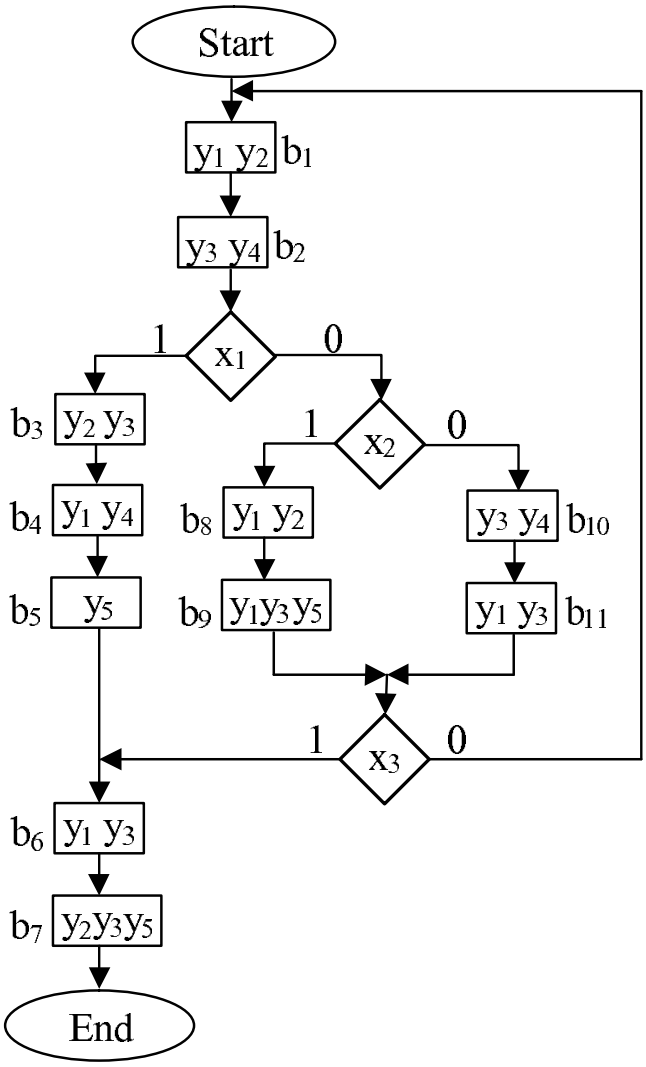

Figure 4.2: The flow-chart Γ_1

In the next step of the designing process, the content of control memory should be formed. To perform this task addresses of all microinstructions have to be encoded. In case of control unit with mutual memory the encoding method is not important, therefore according to the (3.3) natural binary codes will be used. There are eleven operational vertices in the flow-chart Γ_1, so microinstructions will be encoded using four bits. In the presented example microinstructions are addressed as follows: $A(b_0) = 0000$, $A(b_1) = 0001$, ..., $A(b_{11}) = 1010$.

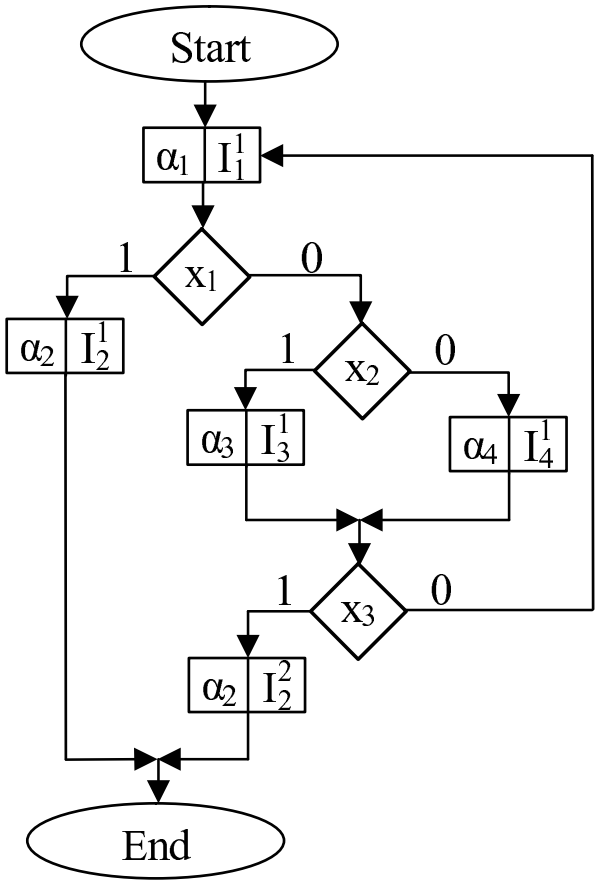

Figure 4.3: The OLCs flow-chart of the CMCU U_1

Each microinstruction executed in the vertex b_k consists of microoperations that are written in this vertex. There are two additional microoperations that are necessary for proper functionality of the CMCU: y_0 and y_K. The first one is set (y_0=1) if vertex b_k belongs to the set of outputs O. In other cases $y_0 = 0$. In the proposed example, y_0 will be produced by vertices b_2, b_7, b_9 and b_{11}. The microoperation y_K is equal to 1 only if vertex b_k is connected with the final vertex of the flow-chart. For the flow-chart Γ_1, y_k will be set only in the vertex b_7.

Next, microinstructions are encoded and the table of control memory is formed. Table 4.1 shows the content of the CM of the control unit U_1.

To determine the excitation function T for the counter, the table of transitions of the CMCU U_1 should be formed. This table describes transitions between all

Table 4.1: The content of the control memory of the CMCU U_1

Vertex	Address	Microinstruction							Comment
		y_0	y_1	y_2	y_3	y_4	y_5	y_K	
b_1	0000	0	1	1	0	0	0	0	I_1^1
b_2	0001	1	0	0	1	1	0	0	O_1
b_3	0010	0	0	1	1	0	0	0	I_2^1
b_4	0011	0	1	0	0	1	0	0	–
b_5	0100	0	0	0	0	0	1	0	–
b_6	0101	0	1	0	1	0	0	0	I_2^2
b_7	0110	1	0	1	1	0	1	1	O_2
b_8	0111	0	1	1	0	0	0	0	I_3^1
b_9	1000	1	1	0	1	0	1	0	O_3
b_{10}	1001	0	0	0	1	1	0	0	I_4^1
b_{11}	1010	1	1	0	1	0	0	0	O_4

operational linear chains depending on input values (set of operational vertices X). In the presented example, the table of transitions has $H = 8$ lines (tab. 4.2).

Table 4.2: The table of transitions of the CMCU U_1

O_g	$SA(O_g)$				X_h	I_j^t	$K(I_j^t)$				T	h
	a_4	a_3	a_2	a_1			t_4	t_3	t_2	t_1		
O_1	0	0	0	1	x_1	I_2^1	0	0	1	0	t_2	1
O_1	0	0	0	1	$\overline{x_1}\ x_2$	I_3^1	0	1	1	1	$t_3\ t_2\ t_1$	2
O_1	0	0	0	1	$\overline{x_1}\ \overline{x_2}$	I_4^1	1	0	0	1	$t_4\ t_1$	3
O_2	0	1	1	0	–	–	-	-	-	-	–	4
O_3	1	0	0	0	x_3	I_2^2	0	1	0	1	$t_3\ t_1$	5
O_3	1	0	0	0	$\overline{x_3}$	I_1^1	0	0	0	0	–	6
O_4	1	0	1	0	x_3	I_2^2	0	1	0	1	$t_3\ t_1$	7
O_4	1	0	1	0	$\overline{x_3}$	I_1^1	0	0	0	0	–	8

Based on the address $SA(O_g)$ (which is represented by the set of variables $A = \{a_1, \ldots, a_4\}$) and on the set of conditional vertices X the excitation function T for the counter is formed:

$$\begin{aligned} t_4 &= \overline{a_4}\cdot\overline{a_3}\cdot\overline{a_2}\cdot a_1\cdot\overline{x_1}\cdot\overline{x_2}, \\ t_3 &= \overline{a_4}\cdot\overline{a_3}\cdot\overline{a_2}\cdot a_1\cdot\overline{x_1}\cdot x_2 + a_4\cdot\overline{a_3}\cdot\overline{a_1}\cdot x_3, \\ t_2 &= \overline{a_4}\cdot\overline{a_3}\cdot\overline{a_2}\cdot a_1\cdot(x_1 + \overline{x_1}\cdot x_2), \\ t_1 &= \overline{a_4}\cdot\overline{a_3}\cdot\overline{a_2}\cdot a_1\cdot\overline{x_1} + a_4\cdot\overline{a_3}\cdot\overline{a_1}\cdot x_3. \end{aligned} \tag{4.3}$$

Now the CMCU U_1 can be easily prototyped using hardware description languages like Verilog (Lee, 1999; Thomas and Moorby, 2002) or VHDL (Bibilo, 1999; Brown and Vernesic, 2000; Pęcheux et al., 2005; Salcic, 1998; Skahill et al., 1996; Zwoliński, 2003). Based on this description the CMCU can be logically synthesised and finally implemented in the FPGA. Chapters 7 and 8 deal with the description of the CMCU in HLDs and implementation of the CMCU in the FPGAs in more detail.

4.1.4 Summary

In this section the synthesis method of the CMCU with mutual memory was described in detail. The presented example was prepared and implemented using the real FPGA device (XC2VP30 from Xilinx, Virtex-II Pro family). Figure 4.4 presents the simplified technological schematic of the controller. Initially the schematic was generated after the logic synthesis, by the Xilinx XST tool. It was modified to clarify the logic structure of the circuit U_1. Here 10 LUTs, that corresponds to the combinational circuit were replaced by one block. Similarly 4 LUTs and 4 flip-flops that form counter are represented by further two blocks. The *FDC* is a Xilinx primitive, and it represents a D-type flip-flop with asynchronous reset. Additionally, main nets were named (in the example T, A) to show the similarity to the logic schematic.

There are two blocks of the CMCU U_1 that are synchronous: the counter and the control memory. Therefore the clock signal *Clk* ought to be delivered to such modules. The counter is triggered by the rising edge of the clock signal. However, because of feedback signals, the control memory is active on the falling edge of the *Clk*. Now an address of a microinstruction is formed on a positive edge,

while outputs are generated when the negative edge of a clock signal occurs. Of course critical timing paths should be checked to avoid timing skews in the circuit (placement and timing paths are automatically verified by Xilinx tools during logical implementation of the design).

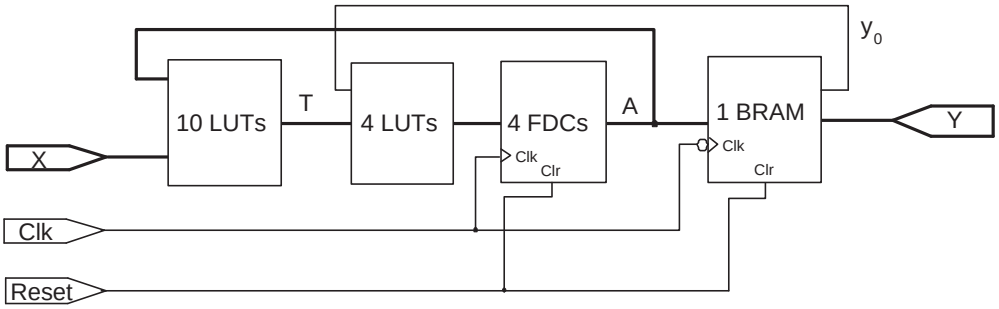

Figure 4.4: Technological structure of the CMCU U_1

The realization of the CMCU U_1 took 14 LUT elements and 1 dedicated memory block of the FPGA resources. In comparison, the controller prepared as a traditional finite-state-machine required 14 LUT elements and 1 dedicated memory block as well (here microinstructions were also implemented using the dedicated memory). Such a simple example shows that the controller designed as the CMCU with mutual memory may not give better results than the equivalent FSM-based circuit.

Achieved results of more tests (presented in Chapter 8 in detail) showed that for controllers that interprets linear flow-chart, the CMCU with mutual memory requires less logic blocks than traditional FSM. However, the benefit is very small (about 3%) and such results were an inspiration for searching new designing ideas of control units. The aim of all researches was to reduce the number of logic elements that are required in order to implement the controller using programmable devices. Next sections present Author's synthesis methods of compositional microprogram control units. All methods shown in this Chapter are based on the modification in the structure of the CMCU U_{MM}.

4.2 The CMCU with function decoder

The microprogram control unit with function decoder U_{FD} is an extended structure of the CMCU with mutual memory (Wiśniewski and Barkalov, 2007; Barkalov et al., 2007b). In comparison to the controller U_{MM} there is additional circuit (function decoder, FD) introduced. Figure 4.5 illustrates the CMCU with function decoder.

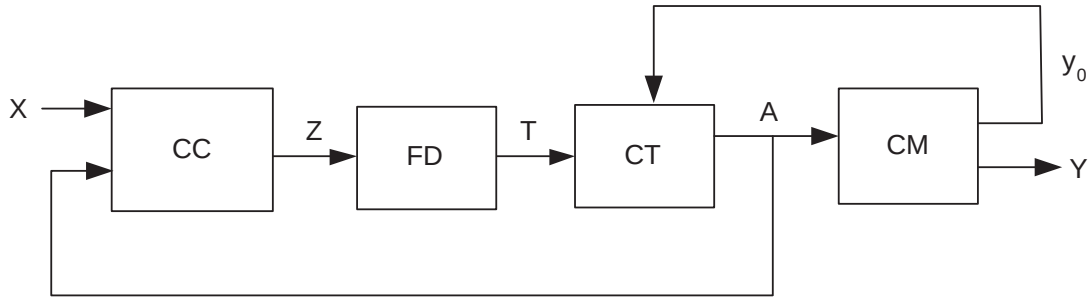

Figure 4.5: The structure of the CMCU with function decoder

The main idea of the method is to reduce the number of logic blocks of the destination FPGA due to the usage of additional block (function decoder) which may be implemented using dedicated memories. Therefore, fewer LUT elements are used during the realisation of the control unit in comparison with the CMCU with mutual memory.

4.2.1 The main idea of the method

In the CMCU U_{FD} variables that form excitation function for the counter are encoded with the minimum number of bits. To solve this task all inputs of operational linear chains ought to be encoded. Now the circuit CC generates the function Z:

$$Z = f(X, A). \tag{4.4}$$

Function Z contains encoded addresses $E(I)$ of all inputs in the set of OLCs. They are further decoded by the circuit FD which indicates the proper code for the counter:

$$T = f(Z). \tag{4.5}$$

The number of bits that are required to encode all inputs can be calculated as $R_Z = \lceil log_2 M_Z \rceil$, where $M_Z = |I|$ is equal to the number of all inputs in the set of OLCs.

Presented solution permits to reduce the number of outputs generated by the circuit CC. Additional block of the function decoder is implemented with dedicated memories of FPGAs. Therefore, the number of logic elements that are needed to implement whole controller is reduced.

4.2.2 Synthesis of the CMCU with function decoder

The designing method of the CMCU U_{FD} includes the following steps:

1. ***Formation of the set of OLCs, encoding their inputs and microinstructions addresses.*** Formation of the set of OLCs is executed in the same manner as it was shown during synthesis of the CMCU with mutual memory. Next, addresses A of all microinstructions are calculated. The encoding style is not important, so natural binary codes may be used. Finally, addresses $K(I_j^t)$ of all inputs of the set of OLCs are encoded with the minimum R_Z number of bits. Now each input has the unique code $E(I_j^t)$.

2. ***Formation of the control memory content.*** According to the addresses calculated in the previous stage, the content of the control memory is formed. As it was mentioned, the encoding style is not important here.

3. ***Formation of the table of the CMCU transitions and formation of the excitation function for the function decoder.*** This table is a base for formation of the system (4.4) and synthesis of the circuit CC. This table contains only transitions for such OLCs that $\alpha_g \in C'$, where $C' \subset C$. Subset C' contains OLCs $\alpha_g \in C$ if their outputs are not connected with the final vertex of the flow-chart. The transition table contains the following columns: O_g,$SA(O_g)$,X_h,I_j^t,$E(I_j^t)$,Z,h. Here:

- O_g is the output from which the transition is executed;
- $SA(O_g)$ is the address of the output O_g;
- X_h is the input signal causing the transition $< O_g, I_t^j >$ and it is equal to the conjunction of the elements from the set X;
- I_j^t is the input of the chain $\alpha_j \in C$ to which the transition is executed;
- $E(I_j^t)$ is the encoded address of the input I_j^t;
- Z is the set of variables that form the excitation function for the function decoder;
- h is the number of the transition (h=1, ..., H).

Based on the transition table, the excitation function Z can be determined. The system (4.4) is represented as:

$$Z_r = \bigvee_{h=1}^{H} C_{rh} F_g^h X_h (r = 1, \ldots, R_1), \tag{4.6}$$

where C_{rh} is a Boolean variable that is equal to 1, if and only if the function Z_r is written in the h-th line of the table of transitions; F_g^h is a conjunction of internal variables $A_r \in A$ corresponding to the address $SA(O_g)$ of the output O_g from the h-th line of the table of transitions.

4. ***Formation of the truth table of the function decoder.*** Based on the code $E(I_j^t)$, function decoder generates the proper address $K(I_j^t)$ of the OLC input. The set of addresses $K(I_j^t)$ form the excitation function T for the counter. The table of function decoder contains the following columns I_j^t, $K(I_j^t)$, $E(I_j^t)$, T, m:

   - I_j^t is the input of the chain $\alpha_j \in C$;
   - $E(I_j^t)$ is the encoded address of the input I_j^t;
   - $K(I_j^t)$ is the code of the input I_j^t;
   - T is the set of variables that form the excitation function for the counter;
   - m is the consecutive line in the truth-table of the function decoder (m=1, ..., M).

Based on this table, the circuit of the function decoder can be implemented with dedicated memory blocks. The code $E(I_j^t)$ forms inputs and $K(I_j^t)$ forms outputs of the function decoder. The volume of the memory that is required for implementation of the function decoder can be calculated as S_{FD}=R_1*2^{R_Z}, where R_1 counts the number of variables that form the excitation function for the counter and R_Z means the number of bits that are required for the OLCs inputs encoding.

5. ***Implementation of the CMCU*** U_{FD}***.*** The memory of the controller is implemented with dedicated memory blocks. However, in case of the CMCU U_{FD} the circuit of function decoder may be realised with dedicated memory blocks as well. In comparison to the CMCU U_{MM}, the number of output variables and the excitation function generated by the circuit CC is reduced. Because the circuit FD is implemented as the memory, the total number of logic blocks that is used for implementation of the controller may highly be reduced. The gain mainly depends on the total number of all inputs in the set of OLCs (see Chapter 8).

4.2.3 Example of the designing method of the CMCU U_{FD}

To demonstrate the designing method of the CMCU U_{FD} with function decoder, the flow-chart Γ_1 will be used as a description of the exemplary controller U_2. As it was shown in the section 4.1.3 there are four OLCs: $C = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$. The circuit should generate eleven microinstructions that consist of five microoperations Y=$\{y_1, \ldots, y_5\}$.

According to the synthesis rules of the CMCU with function decoder all OLCs inputs ought to be encoded. There are five inputs therefore the minimum number of bits that are needed for encoding is $R_Z = 3$. In the presented example natural binary code will be used and inputs will be encoded as follows: $E(I_1^1) = 000$, $E(I_1^2) = 001$, $E(I_2^2) = 010$, $E(I_1^1) = 011$, $E(I_1^1) = 100$.

In the second step the content of the control memory is formed. This stage is performed in the same way as it was shown in the section 4.1.3 (the content of the CM is shown in the tab. 4.1).

Next the table of transitions of the CMCU U_2 is prepared. The table is similar to the table shown in 4.2 but now all inputs of OLCs are encoded (tab. 4.3).

Table 4.3: The table of transitions of the CMCU U_2

O_g	$SA(O_g)$				X_h	I_j^t	$E(I_j^t)$			Z	h
	a_4	a_3	a_2	a_1			z_3	z_2	z_1		
O_1	0	0	0	1	x_1	I_2^1	0	0	1	z_1	1
O_1	0	0	0	1	$\overline{x_1}\ x_2$	I_3^1	0	1	1	$z_2\ z_1$	2
O_1	0	0	0	1	$\overline{x_1}\ \overline{x_2}$	I_4^1	1	0	0	z_3	3
O_2	0	1	1	0	–	–	-	-	-	–	4
O_3	1	0	0	0	x_3	I_2^2	0	1	0	z_2	5
O_3	1	0	0	0	$\overline{x_3}$	I_1^1	0	0	0	–	6
O_4	1	0	1	0	x_3	I_2^2	0	1	0	z_2	7
O_4	1	0	1	0	$\overline{x_3}$	I_1^1	0	0	0	–	8

Based on the table of transitions, the excitation function Z for the circuit FD is formed:

$$\begin{aligned} z_3 &= \overline{a_4}\cdot\overline{a_3}\cdot\overline{a_2}\cdot a_1\cdot\overline{x_1}\cdot\overline{x_2}, \\ z_2 &= \overline{a_4}\cdot\overline{a_3}\cdot\overline{a_2}\cdot a_1\cdot\overline{x_1}\cdot x_2 + a_4\cdot\overline{a_3}\cdot\overline{a_1}\cdot x_3, \\ z_1 &= \overline{a_4}\cdot\overline{a_3}\cdot\overline{a_2}\cdot a_1\cdot(x_1 + \overline{x_1}\cdot x_2). \end{aligned} \tag{4.7}$$

In order to decode the proper excitation function for the counter, the table of function decoder has to be prepared. Table 4.4 shows the content of the function decoder of the CMCU U_2.

Table 4.4: The table of the function decoder of the CMCU U_2

I_j^t	$E(I_j^t)$			$K(I_j^t)$				T	m
	z_3	z_2	z_1	t_4	t_3	t_2	t_1		
I_1^1	0	0	0	0	0	0	0	–	1
I_2^1	0	0	1	0	0	1	0	t_2	2
I_2^2	0	1	0	0	1	0	1	$t_3\ t_1$	3
I_3^1	0	1	1	0	1	1	1	$t_3\ t_2\ t_1$	4
I_4^1	1	0	0	1	0	0	1	$t_4\ t_1$	5

The circuit FD may be implemented either using dedicated memories or with logic blocks of the FPGA as well. In case of realisation with LUT elements, additionally the excitation function T is formed:

$$\begin{aligned} t_4 &= z_3 \cdot \overline{z_2} \cdot \overline{z_1}, \\ t_3 &= \overline{z_3} \cdot z_2, \\ t_2 &= \overline{z_3} \cdot z_1, \\ t_1 &= \overline{z_3} \cdot z_2 + z_3 \cdot \overline{z_2} \cdot \overline{z_1}. \end{aligned} \tag{4.8}$$

Finally, the CMCU U_2 may be implemented in the FPGA. As it was mentioned an additional module of the function decoder may be realized either with dedicated memory blocks or LUT elements of the programmable device.

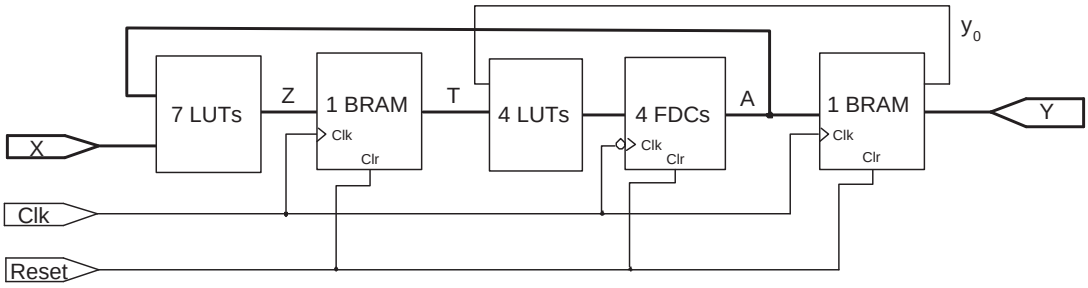

Figure 4.6: Technological structure of the CMCU U_2

Figure 4.6 shows the schematic of the CMCU U_2. The usage of function decoder permitted to decrease the number of LUT elements to 11. It means that the area used by logic blocks of the circuit U_2 was reduced 21% compared to the CMCU U_1.

There is an additional synchronous component in the CMCU U_2 in comparison to the CMCU U_1. The function decoder is implemented with dedicated memory blocks of an FPGA and it should be triggered by a clock signal. Therefore the FD is active on a positive edge, while the counter is synchronized with the falling edge of a *Clk*. Finally the control memory is triggered by a positive clock signal. Such a solution ensures proper functionality of the controller. On the other hand it should be emphasised, that microinstructions are formed by a half clock pulse later in comparison to the CMCU U_1.

4.2.4 Summary

The CMCU with function decoder was presented in the section. Here the excitation function for the counter was decoded on the minimum number of bits, thus the number of logic elements that are used to realize the circuit CC is reduced. The additional block - function decoder - may be realized with dedicated memory blocks of the FPGA. Such a solution preserves additional area of logic elements of the programmable device.

Detailed experiments showed that realization of the controller with application of the CMCU with function decoder reduces the number of used LUT elements average by 19% (see Chapter 8 for more detail).

4.3 The CMCU with outputs identification

The structure of the CMCU U_{OI} with outputs identification is illustrated by the fig. 4.7. The main idea is to use the part of the address A for the identification of the internal states of the control unit. Now the set of variables Q ($Q \subset A$) represents the code of the current state of the controller (Wiśniewski et al., 2007; Barkalov and Wiśniewski, 2005c).

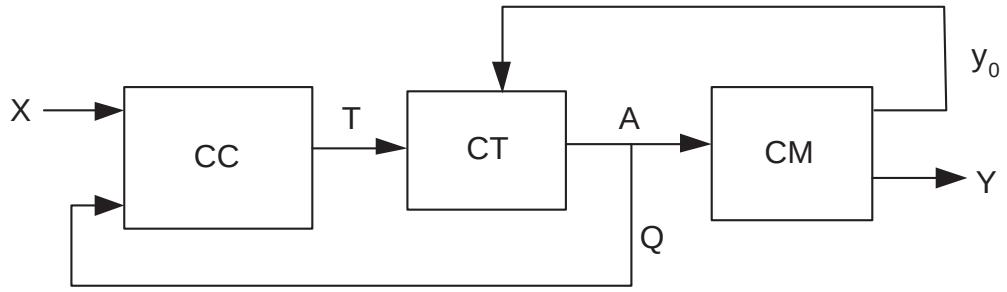

Figure 4.7: The structure of the CMCU with outputs identification

4.3.1 The main idea of the method

In the CMCU U_{OI} the set of feedback variables A that are used for the identification of the current state of the controller is reduced to the minimum. Outputs of the operational linear chains may be recognized using R_{OI} bits thanks

to the special encoding of microinstructions (Wiśniewski et al., 2006a; Barkalov et al., 2007a; Barkalov et al., 2006a; Barkalov and Wiśniewski, 2005b; Wiśniewski, 2006a). Therefore, the combinational circuit generates the function T for the counter (Wiśniewski et al., 2006a; Barkalov et al., 2006b):

$$T = f(X, Q), \tag{4.9}$$

where $Q \subseteq A$, $|Q| = R_{OI}$, $Q = \{Q_1, \ldots, Q_{R_{OI}}\}$.

4.3.2 Synthesis of the CMCU with outputs identification

The synthesis of the CMCU with mutual memory includes the following steps:

1. ***Formation of the set of OLCs.*** The set of operational linear chains is created. For each OLC its output and all inputs are determined. According to definitions 3.4 and 3.6 there are M_2 operational linear chains and the length of the longest is specified by the M_1 value. The total number of microinstructions is equal to M_3 (see definition 3.8).

2. ***Addressing of microinstructions and encoding of OLCs outputs.*** Let $Q \subseteq A$ be a set of variables that are sufficient for one-to-one identification of the OLC $\alpha_g \in C$ and $R_{OI} = |Q|$. Microinstructions addressing of the CMCU is executed in a following manner:

   (a) At the beginning all microinstructions are encoded using natural binary codes.

   (b) The value of R_{OI} is set to $R_{OI} = R_2$, where $R_2 = \lceil log_2 M_2 \rceil$.

   (c) The table of addressing is created. The table has $2^{R_{OI}}$ columns marked by R_{OI} major address bits and $2^{R_3 - R_{OI}}$ lines marked by $R_3 - R_{OI}$ junior address bits. Here $R_3 = \lceil log_2 M_3 \rceil$.

   (d) If outputs of two different OLCs $\alpha_i, \alpha_j \in C$ are situated in the same column and both outputs are not connected with the final vertex of the flow-chart then the information is shifted to the right starting from the first vertex of the OLC α_j (j>i). The releasing cells of the table are filled by symbols "$*$" . This operation is performed until outputs O_i and O_j will be in different columns of the table.

(e) If outputs of all OLCs have one-to-one identification by R_{OI} bits then the algorithm moves on to the point (g).

(f) If the address of any vertex is beyond the actual addressing space then $R_{OI} := R_{OI} + 1$. Next the algorithm is repeated from the point (c).

(g) End.

Finally all microinstructions are encoded. Now the code of each microinstruction is formed as a concatenation of major (columns) and minor (lines) addresses of created table. Outputs of OLCs are encoded using only major bits of the address. Such an encoding will be further used in the formation of the transitions table of the CMCU.

For better understanding, presented algorithm of microinstructions addressing will be illustrated later by an example.

3. ***Formation of the control memory content.*** The content of the control memory is formed. Addresses of microinstructions are created according to the algorithm presented in the previous step.

4. ***Formation of the transitions table of the CMCU U_{OI}, formation of the excitation function for the counter.*** At this stage the table of transitions between OLCs is created. The table contains the following columns: O_g,$MA(O_g)$,X_h,I_j^t,$K(I_j^t)$,T,h, where:

   - O_g is the output from which the transition is executed;
   - $MA(O_g)$ is the major part of an address of the output O_g, this address was calculated at the stage 2;
   - X_h is the input signal causing the transition $< O_g, I_t^j >$ and it is equal to the conjunction of the elements from the set X;
   - I_j^t is the input of the chain $\alpha_j \in C$ where the transition is executed;
   - $K(I_j^t)$ is the address of the input I_j^t;
   - T is the set of the variables that form the excitation function for the counter;
   - h is the number of the transition (h=1, ..., H).

Based on this table the excitation function T for the counter is formed:

$$T_r = \bigvee_{h=1}^{H} C_{rh} E_g^h X_h (r = 1, \ldots, R_{OI}). \tag{4.10}$$

Here C_{rh} is a Boolean variable that is equal to 1 if and only if the function T_r is written in the h-th line of the table of transitions; E_g^h is a conjunction of internal variables $Q_r \in Q$ corresponding to the address $MA(O_g)$ of the output O_g from the h-th line of the table of transitions.

5. ***Implementation of the CMCU*** U_{OI}***.*** This step is executed in the same manner as it was shown during the designing process of the CMCU U_{MM}. The combinational circuit and the counter are implemented using LUT elements while the control memory is realised with dedicated memory blocks of FPGAs.

4.3.3 Example of the synthesis of the CMCU with outputs identification

To bring closer the idea of the OLCs encoding, the designing process of the CMCU U_{OI} with outputs identification will be illustrated by an example. Once more the flow-chart Γ_1 will be used as the initial description of the controller U_3. As it was shown in the subsection (4.1.3), there are $M_3 = 11$ operational vertices and $M_2 = 4$ operational linear chains. The longest OLC is α_2 and it contains $M_1 = 5$ elements. According to the algorithm of microinstructions addressing, the initial value of the variable R_{OI} is equal to $R_2 = \lceil log_2 M_2 \rceil = 2$. Thus at the beginning, the table of addressing has $2^{R_{OI}} = 2$ columns and $2^{R_3 - R_{OI}} = 2$ lines (fig. 4.8).

Initially all addresses of microinstructions are encoded in natural binary code. In the presented example outputs O_3 of α_3 and O_4 of α_4 are located in the same column. Because neither O_3 nor O_4 are connected with the final vertex of the flow-chart Γ_1, thus all components that have higher addresses than the output O_3 are shifted. This movement is performed while the output O_4 is in the same column as O_3.

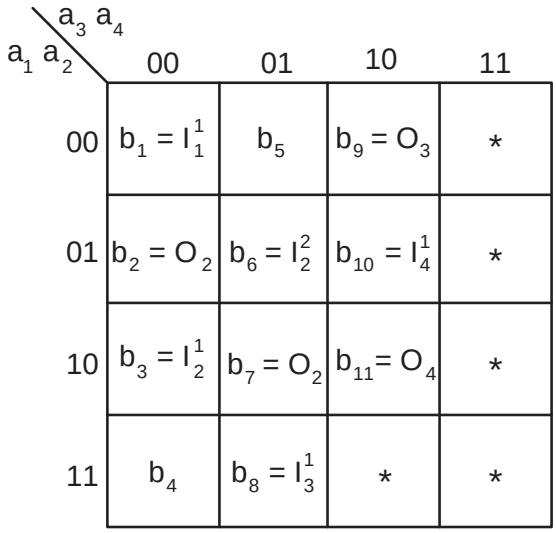

Figure 4.8: The initial table of addressing

Figure 4.8 presents the table after the shift operation. Now each OLC output is situated in the different column and there are no vertices beyond the addressing space. It means that all addresses are encoded and the algorithm is finished.

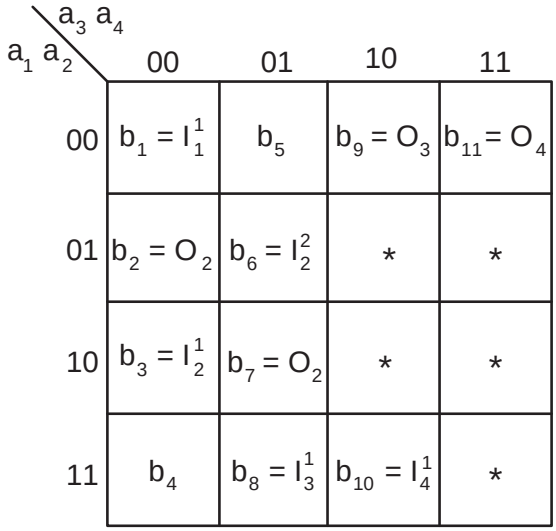

Figure 4.9: The table of addressing after shift operations

The table of encoding shows that each OLC output may be recognized using $|Q| = 2$ major bits, where the set Q is the subset of A and contains only variables $Q = \{a_3, a_4\}$. Now OLCs outputs are encoded as follows: $K(O_1) = 00$, $K(O_2) = 01$, $K(O_3) = 10$, $K(O_4) = 11$. The presented algorithm of encoding is used during creation of the control memory. Addresses of microinstructions are formed as concatenations of major (columns) and minor (lines) bits of the table of addressing. The content of the control memory is presented in the table 4.5.

Table 4.5: The control memory content of the CMCU U_3

Vertex	Address	Microinstruction							Comment
		y_0	y_1	y_2	y_3	y_4	y_5	y_K	
b_1	0000	0	1	1	0	0	0	0	I_1^1
b_2	0001	1	0	0	1	1	0	0	O_1
b_3	0010	0	0	1	1	0	0	0	I_2^1
b_4	0011	0	1	0	0	1	0	0	–
b_5	0100	0	0	0	0	0	1	0	–
b_6	0101	0	1	0	1	0	0	0	I_2^2
b_7	0110	1	0	1	1	0	1	1	O_2
b_8	0111	0	1	1	0	0	0	0	I_3^1
b_9	1000	1	1	0	1	0	1	0	O_3
b_{10}	1011	0	0	0	1	1	0	0	I_4^1
b_{11}	1100	1	1	0	1	0	0	0	O_4

In the next step, the table of transitions of the CMCU U_3 is created. The table is similar to the table that was created for the CMCU with mutual memory, however now there are only two major bits of the address used as the OLC output identification (tab. 4.6).

Table 4.6: The transitions table of the CMCU U_3

O_g	$MA(O_g)$		X_h	I_j^t	$K(I_j^t)$				T	h
	a_4	a_3			t_4	t_3	t_2	t_1		
O_1	0	0	x_1	I_2^1	0	0	1	0	t_2	1
O_1	0	0	$\overline{x_1}\ x_2$	I_3^1	0	1	1	1	$t_3\ t_2\ t_1$	2
O_1	0	0	$\overline{x_1}\ \overline{x_2}$	I_4^1	1	0	1	1	$t_4\ t_2\ t_1$	3
O_2	0	1	–	–	-	-	-	-	–	4
O_3	1	0	x_3	I_2^2	0	1	0	1	$t_3\ t_1$	5
O_3	1	0	$\overline{x_3}$	I_1^1	0	0	0	0	–	6
O_4	1	1	x_3	I_2^2	0	1	0	1	$t_3\ t_1$	7
O_4	1	1	$\overline{x_3}$	I_1^1	0	0	0	0	–	8

Based on the address $MA(O_g)$ (represented by the set of variables $Q = \{a_3, a_4\}$) and on the set of conditional vertex X, the excitation function T for the counter is formed:

$$\begin{aligned} t_4 &= \overline{a_4}\cdot\overline{a_3}\cdot\overline{x_1}\cdot\overline{x_2}, \\ t_3 &= \overline{a_4}\cdot\overline{a_3}\cdot\overline{x_1}\cdot x_2 + a_4\cdot x_3, \\ t_2 &= \overline{a_4}\cdot\overline{a_3}, \\ t_1 &= \overline{a_4}\cdot\overline{a_3}\cdot\overline{x_1} + a_4\cdot x_3. \end{aligned} \tag{4.11}$$

Now the CMCU U_3 can be prototyped using HDL languages. The excitation function T contains fewer variables and shorter equations in comparison to the excitation function formed for the controller with mutual memory (see 4.8). Therefore it is expected that the CMCU U_3 should consume fewer logic elements than the CMCU U_1. Implementation of the controller in the FPGA showed that in fact, the CMCU U_3 with output identification requires 11 LUT elements (fig. 4.10), which means the reduction by 21% in comparison to the CMCU U_1 with mutual memory.

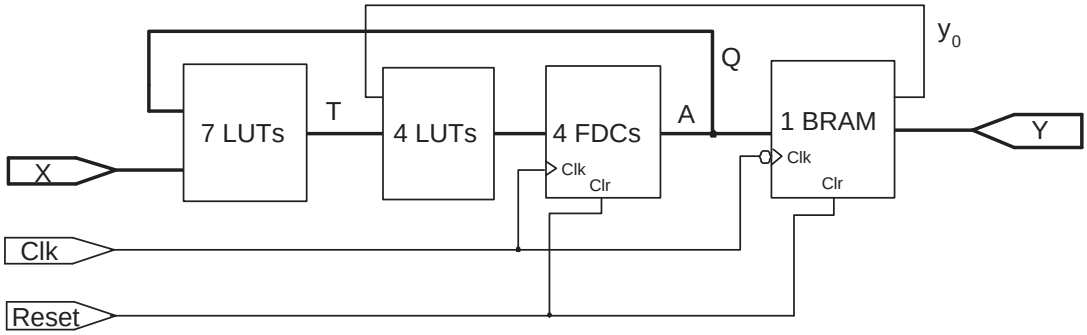

Figure 4.10: Technological structure of the CMCU U_3

4.3.4 Summary

The CMCU with outputs identification was described in this section. The main advantage of the presented solution is reduction of the number of variables that keep the actual code of the state of the controller. Thus, the number of logic elements of the FPGA is reduced in comparison with the traditional CMCU with mutual memory. It should be pointed out that special addressing of microinstructions is required. Therefore additional designing step has to be performed in comparison to the synthesis process of the CMCU U_{MM}.

4.4 The CMCU with outputs identification and function decoder

The CMCU U_{OD} with outputs identification and function decoder (fig. 4.11) is a conjunction of two structures presented in previous sections. There is a special addressing of microinstructions used in the CMCU U_{OD}. Moreover, maximal encoding of the set of variables A is performed as well.

4.4.1 The main idea of the method

To improve the reduction of LUT elements of the implementation of CMCUs U_{FD} and U_{OI}, both methods may be combined. Now the combinational circuit generates the excitation function Z for the circuit FD:

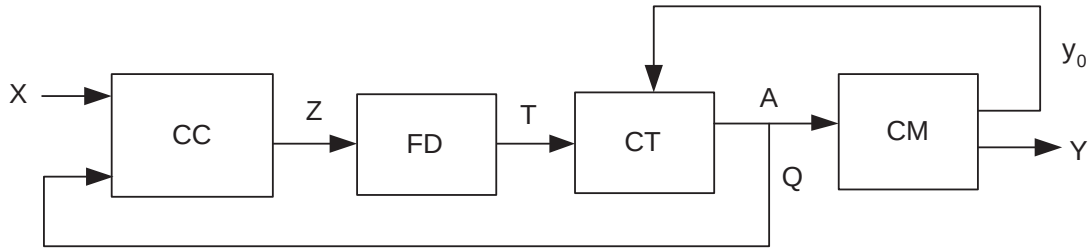

Figure 4.11: The structure of the CMCU with outputs identification and function decoder

$$Z = f(X, Q), \tag{4.12}$$

where X means the set of input variables of the CMCU (conditional vertices) and $Q \subseteq A$ is a feedback function generated by the counter. The function decoder generates proper addresses of microinstructions:

$$T = f(Z), \tag{4.13}$$

where T means the set of variables that form the excitation function for the counter.

4.4.2 Synthesis of the CMCU with outputs identification and function decoder

The designing method the CMCU U_{FD} includes the following steps:

1. ***Formation of the set of OLCs and encoding their inputs.*** This step is executed by the same procedure as it was described in the section of synthesis of the CMCU with function decoder. All inputs of OLCs are encoded with the minimum R_Z number of bits, so each input has the unique code $E(I_j^t)$.

2. ***Addressing microinstructions and encoding OLCs outputs.*** Addresses of microinstructions are represented using the algorithm shown in the previous section. Outputs of OLCs are encoded using only major bits of addresses. Such an encoding will be further used in the formation of the table of transitions of the CMCU.

3. ***Formation of the control memory content.*** According to addresses calculated in the previous stage, the content of the control memory is prepared.

4. ***Formation of the transitions table of the CMCU and formation of the excitation function for the function decoder.*** The table of transitions is the base for formation of the system (4.12) and synthesis of the circuit CC. This table contains only transitions for such OLCs that their outputs are not connected to the final vertex of the flow-chart. The table of transitions contains the following columns: O_g,$MA(O_g)$,X_h,I_j^t,$E(I_j^t)$,Z,h. Here:

   - O_g is the output from which the transition is executed;
   - $MA(O_g)$ is the major part of an address of the output O_g, this address was calculated at the stage of microinstruction addressing;
   - X_h is the input signal causing the transition $< O_g, I_t^j >$ and it is equal to the conjunction of the elements from the set X;
   - I_j^t is the input of the chain $\alpha_j \in C$ to which the transition is executed;
   - $E(I_j^t)$ is the address of the input I_j^t;
   - Z is the set of the variables that form the excitation function for the function decoder;
   - h is the number of the transition (h=1, ..., H).

   Now, according to the (4.6), the set of variables Z can be formed.

5. ***Formation of the table of the function decoder.*** This step is executed in the same manner as during the designing of the CMCU with function decoder (see section 4.2).

6. ***Implementation of the CMCU*** U_{OD}***.*** The main advantage of the CMCU with outputs identification and function decoder is a possibility to implement both blocks - FD and CM - with dedicated memory blocks. Moreover, thanks to outputs identification the number of feedback functions for the combinational circuit decreases in comparison to the CMCU U_{MM}. Therefore, implementation of the CMCU U_{OD} consumes the least logic elements

of programmable devices in comparison with CMCUs U_{MM}, U_{FD} and U_{OI}. However, it should also be pointed out, that presented controller uses at least two dedicated memory blocks of the FPGA.

4.4.3 Example of synthesis of the CMCU with outputs identification and function decoder

To illustrate the synthesis of the CMCU U_{OD}, the flow-chart Γ_1 will be used as the initial description of the controller. The prototyping process of the CMCU U_4 with outputs identification and function decoder is a conjunction of the designing of CMCUs U_2 and U_3. At the beginning, the set of OLCs is formed and all OLCs inputs are encoded. As it was presented in previous sections, there are four OLCs which have five inputs. Thus OLCs inputs may be encoded using $|Z|$=3 bits. In the presented example natural binary code will be used: $E(I_1^1)$=000, $E(I_1^2)$=001, $E(I_2^2)$=010, $E(I_1^1)$=011, $E(I_1^1)$=100.

At the next stage, addressing microinstructions and encoding OLCs outputs should be performed. According to the algorithm presented in the subsection 4.3.2, microinstructions corresponding to vertices b_1,...,b_9 are addressed consecutively in natural binary code: $A(b_1)$=0000, $A(b_2)$=0001, $A(b_3)$=0010,..., $A(b_9)$=1000. Addresses of two last microinstructions are shifted, thus their codes are $A(b_{10})$=1011 and $A(b_{11})$=1100. Outputs of OLCs are encoded with $|Q|$=2 major bits of an address, therefore $MA(O_1)$=00, $MA(O_2)$=01, $MA(O_3)$=10, $MA(O_4)$=11. The content of the control memory is identical to the CMCU U_3 shown in the previous section (tab. 4.5).

Next the transition table of the CMCU is prepared. The table contains transitions from the output O_i (which is encoded using $Q \subset A$ bits) to the input I_j^t (which is encoded using Z bits). Table 4.7 shows the transition table for the CMCU U_4.

From the table of transitions the excitation function Z for the function decoder is formed:

$$\begin{aligned} z_3 &= \overline{a_4}\cdot\overline{a_3}\cdot\overline{x_1}\cdot\overline{x_2}, \\ z_2 &= \overline{a_4}\cdot\overline{a_3}\cdot\overline{x_1}\cdot x_2 + a_4\cdot x_3, \\ z_1 &= \overline{a_4}\cdot\overline{a_3}\cdot(x_1 + \overline{x_1}\cdot x_2). \end{aligned} \tag{4.14}$$

Table 4.7: The table of transitions of the CMCU U_{OD}

O_g	$MA(O_g)$		X_h	I_j^t	$E(I_j^t)$			Z	h
	a_4	a_3			z_3	z_2	z_1		
O_1	0	0	x_1	I_2^1	0	0	1	z_1	1
O_1	0	0	$\overline{x_1}\cdot x_2$	I_3^1	0	1	1	$z_2\ z_1$	2
O_1	0	0	$\overline{x_1}\cdot\overline{x_2}$	I_4^1	1	0	0	z_3	3
O_2	0	1	–	–	-	-	-	–	4
O_3	1	0	x_3	I_2^2	0	1	0	z_2	5
O_3	1	0	$\overline{x_3}$	I_1^1	0	0	0	–	6
O_4	1	1	x_3	I_2^2	0	1	0	z_2	7
O_4	1	1	$\overline{x_3}$	I_1^1	0	0	0	–	8

In the last step the table for function decoder is formed. For the CMCU U_4, this table is identical as it was presented during synthesis of CMCU U_2 (tab. 4.4). Finally, the controller may be designed with HDL languages and implemented in the programmable device. Figure 4.12 shows the technological schematic of realisation of the CMCU U_4 in the FPGA. As it was expected, the CMCU U_4 took the fewest logic blocks of the device between all CMCUs presented in this Chapter. Conjunction of the OLCs output identification and application of the function decoder resulted in reduction of used LUT elements to 10. This means that the initial CMCU with mutual memory was decreased by 26%.

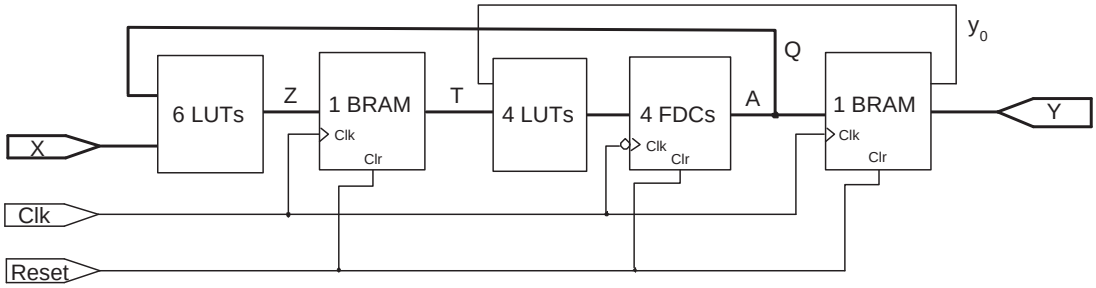

Figure 4.12: Technological structure of the CMCU U_4

4.4.4 Summary

This section presented the designing method of the CMCU U_{OD} with outputs identification and function decoder. The main idea was to use special identification of OLCs outputs what resulted in reduction of the size of the internal feedback function. Additionally, the application of the function decoder permitted to decrease the number of outputs of the combinational circuit to reduce finally the number of used LUT elements of the destination FPGA device. Detailed experiments showed that the designing method of the CMCU U_{OD} always gives better results than the CMCU with mutual memory, average by 32% (see Chapter 8 for more detail).

4.5 Conclusions

There were four designing methods of the CMCU with mutual memory presented in this Chapter. The first one was initially proposed by prof. Alexander Barkalov and it was the inspiration for further methods. The main idea was to reduce the number of logic blocks that are required to implement the CMCU in the FPGA.

The CMCU U_{FD} includes new block - function decoder. Now the excitation function for the counter is encoded with the minimum number of bits. The additional circuit is realised with dedicated memory blocks of the FPGA, thus the application of the function decoder reduces the total number of LUT elements in comparison to the CMCU U_{MM}. Experiments showed that application of the function decoder may be ineffective. Such a situation may happen when the initial-flow chart contains many short OLCs. It should also be pointed out that the CMCU with function decoder usually consumes 19% less of LUT elements compared to the traditional CMCU with mutual memory.

The reduction of the number of variables that keep actual state of the controller was introduced in the CMCU U_{OI} with outputs identification. Here special encoding of microinstructions is used and each OLC output has the unique code that usually (but not always - it depends on the structure of the flow-chart that describes the controller) requires fewer bits than in the traditional CMCU with mutual memory. Experiments showed that the CMCU with OLCs outputs identification consumes fewer LUTs than CMCU with mutual memory by 14%.

The last presented solution of the CMCU designing combines two methods of synthesis. In the CMCU U_{OD} with outputs identification and function decoder the excitation function for the counter is encoded and special addressing of microinstructions is performed as well. As it was expected, such a solution gained the best results of all CMCUs presented in this Chapter. The CMCU U_{OD} **always requires fewer LUT elements than CMCUs** U_{MM} **and** U_{FD}. It is also almost always better than CMCU U_{OI} (in 98% cases). **The area used by CMCU with outputs identification and function decoder is on average 32% smaller than traditional CMCU with mutual memory.**

Detailed interpretation of results of experiments are presented in Chapter 8. Next Chapter shows four alternative methods of the CMCU designing where the idea of sharing codes is used.

Chapter 5

Compositional microprogram control units with sharing codes

This Chapter refers to synthesis methods of a CMCU with sharing codes. The aim of proposed structures is to determine the microinstruction address by both codes generated by the counter and by the register.

In the first section, the traditional synthesis method of the CMCU U_{SC} with sharing codes is presented. The structure of the CMCU U_{SC} was proposed by prof. A. Barkalov in (Barkalov and Palagin, 1997). Next sections deal with three new methods of synthesis of the CMCU with sharing codes. The aim of proposed methods is different. The first - the CMCU with sharing codes and function decoder - concentrates on the reduction of logic blocks of the FPGA. The aim of the second designing method (the CMCU with address converter) is to reduce the width of the control memory address and thus the volume of the memory. This method is very useful if the volume of the control memory exceeds the volume offered by the dedicated memory blocks of the FPGA. The third method mixes the usage of address converter and function decoder.

5.1 The CMCU with sharing codes

Figure 5.1 shows the CMCU U_{SC} with sharing codes (Barkalov and Wiśniewski, 2004e; Wiśniewski, 2004; Wiśniewski, 2006b; Wiśniewski et al., 2006c). The main idea is to use both codes generated by the counter and by the register to form

the microinstruction address. Therefore, the number of variables that are used for encoding of the excitation functions for the counter is reduced in comparison to the CMCU U_{BS}.

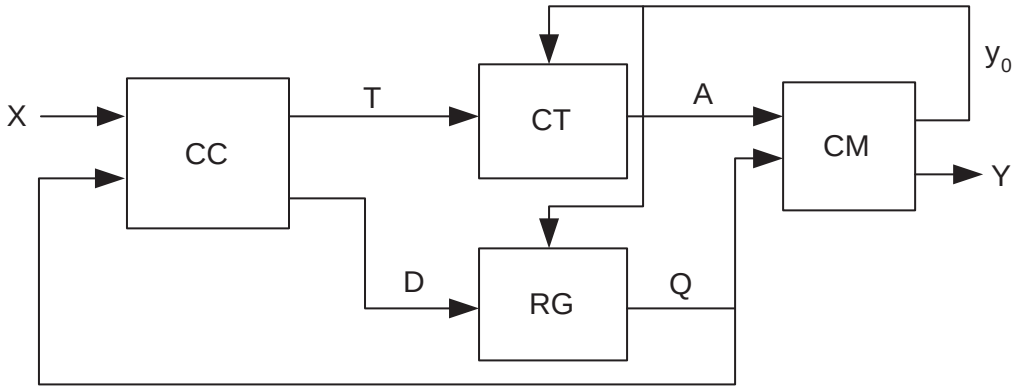

Figure 5.1: The structure of the CMCU with sharing codes

In the CMCU with sharing codes the microinstruction address $A(b_t)$ is represented as a concatenation (Barkalov and Palagin, 1997):

$$A(b_t) = K(\alpha_g) * K(b_t). \tag{5.1}$$

Here $K(\alpha_g)$ is a code of the OLC $\alpha_g \in C$ with $R_2 = \lceil log_2 M_2 \rceil$ bits, where M_2 defines the number of OLCs in the initial flow-chart Γ; $K(b_t)$ is a code of a component of the OLC $\alpha_g \in C$ corresponding to the vertex $b_t \in B$. Code $K(b_t)$ has $R_1 = \lceil log_2 M_1 \rceil$ bits, where M_1 is equal to the maximum amount of components in the OLC $\alpha_g \in C$. Sign $(*)$ in (5.1) is used for concatenation operation.

5.1.1 The main idea of the method

In the CMCU U_{SC} the combinational circuit CC generates excitation functions for the counter CT and for the register RG:

$$T = f(X, Q), \tag{5.2}$$

$$D = f(X, Q). \tag{5.3}$$

The RG is in charge of holding the code of the current OLC. Additionally it generates an upper part of the microinstruction address. The CT keeps only the number of the active component (block) in the current OLC. Therefore it determines the lower part of the microinstruction address.

5.1.2 Synthesis of the CMCU with sharing codes

The designing method of the CMCU U_{SC} includes the following steps:

1. ***Formation of the set of OLCs, encoding OLCs and their components.*** Based on definitions 3.2 and 3.3, the set of operational linear chains is formed. Next, OLCs are encoded. Each OLC forms the code $K(\alpha_g)$. Finally all components in OLCs are encoded with the code $K(b_t)$. Both, the OLC and its components are encoded with natural binary code.

2. ***Addressing microinstructions and formation of the control memory.*** The microinstruction address is determined as concatenation of the code of the OLC and its component, according to the (5.1). Then, the content of the control memory is formed. The volume of the control memory can be calculated as S_{CM}=(N+2)$*2^{R_1+R_2}$, where R_1 is the size of the code generated by the counter and R_2 is the size of the code generated by the register.

3. ***Formation of the transitions table of the CMCU and formation of excitation functions for the counter and for the register.*** This table is the base for the formation of system functions (5.2 and 5.3) and synthesis of the circuit CC. The table of transitions contains the following columns: α_g,$K(\alpha_g)$,X_h,α_t,$K(\alpha_t)$,b_j,$K(b_j)$,D,T,h. Here:

   - α_g is the OLC from which the transition is executed;
   - $K(\alpha_g)$ is the code of the OLC α_g;
   - X_h is the input signal causing the transition and it is equal to the conjunction of elements from the set X;
   - α_t is the destination OLC where the transition is executed;
   - $K(\alpha_t)$ is the code of the OLC α_t;

- b_j is the destination component in the OLC α_t where the transition is executed;
- $K(b_j)$ is the code of the component b_j;
- D is the set of variables that form an excitation function for the register;
- T is the set of variables that form an excitation function for the counter;
- h is the number of the transition (h=1, ..., H).

Based on this table, the excitation function T for the counter is formed:

$$T_r = \bigvee_{h=1}^{H} T_{rh} E_g^h X_h (r = 1, \ldots, R_1). \tag{5.4}$$

Here T_{rh} is a Boolean variable that is equal to 1 if and only if the variable T_r is written in the h-th line of the table of transitions; E_g^h is the conjunction of internal variables $Q_r \in Q$ corresponding to the code $K(\alpha_g)$ from the h-th line of the table of transitions.

Similarly the excitation function D for the register is determined:

$$D_r = \bigvee_{h=1}^{H} D_{rh} F_g^h X_h (r = 1, \ldots, R_2), \tag{5.5}$$

where D_{rh} is a Boolean variable that is equal to 1 if and only if the variable D_r is written in the h-th line of the table of transitions; F_g^h is the conjunction of internal variables $Q_r \in Q$ corresponding to the code $K(\alpha_g)$ from the h-th line of the table of transitions.

4. ***Implementation of the CMCU*** U_{SC}**.** Three modules of the CMCU U_{SC} - the combinational circuit, the counter and the register - are implemented with logic blocks of the destination programmable device. The control memory may be realized in two ways, with dedicated memory blocks or with logic elements as well.

5.1.3 Example of synthesis of the CMCU with sharing codes

To bring closer the idea shown in this section, the designing method of the CMCU U_5 with sharing codes will be illustrated by the example. Let use the description of the controller presented in the fig. 5.2. There are 13 operational and 2 conditional vertices in the flow-chart Γ_2.

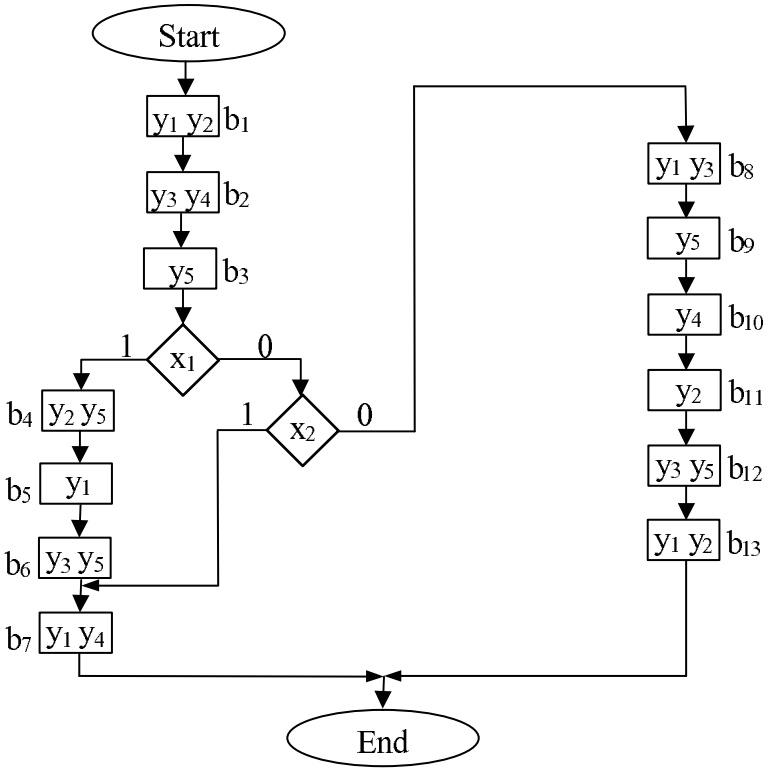

Figure 5.2: The flow-chart Γ_2

According to the designing rules defined in the previous subsection, initially the set of operational linear chains ought to be formed. In the presented example there are three OLCs: $C=\{\alpha_1, \alpha_2, \alpha_3\}$. Here $\alpha_1=\langle b_1, \ldots, b_3\rangle$, $\alpha_2=\langle b_4, \ldots, b_7\rangle$, $\alpha_3=\langle b_8, \ldots, b_{13}\rangle$. All OLCs except α_2 have one input: for α_1 it is vertex b_1 while for α_3 - b_8. The OLC α_2 has two inputs: vertex b_4 and vertex b_7. There are three outputs in the set of OLCs $O=\{O_1, O_2, O_3\}$, where $O_1=b_3$, $O_2=b_7$, $O_3=b_{13}$. The set of operational linear chains is shown in the fig. 5.3.

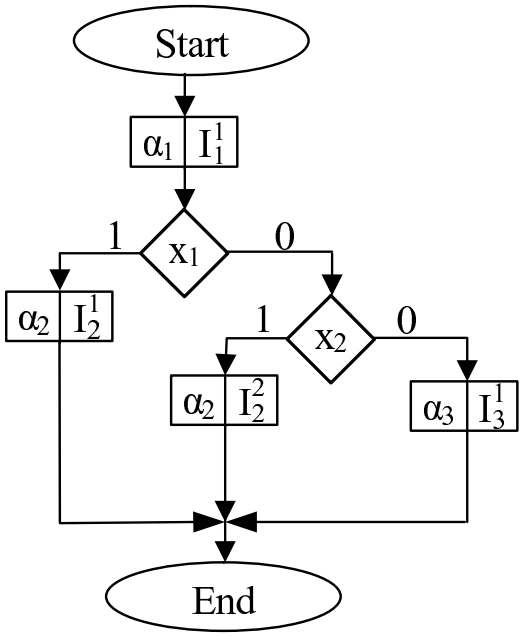

Figure 5.3: The OLCs flow-chart of the CMCU U_5

Operational linear chains and their components are encoded using natural binary codes. There are M_2=3 OLCs thus R_2=$\lceil log_2 M_2 \rceil$=2 bits will be used for encoding: $K(\alpha_1)$=00, $K(\alpha_2)$=01, $K(\alpha_1)$=10. To encode OLCs components, firstly the length M_1 of the longest OLC ought to be determined. For the presented example α_1 contains M^1=3 components, α_2 includes M^2=4 vertices and α_3 has M^3=6 blocks. Thus the longest OLC is α_3 with M_1=M^3 and it is equal to 6. It means that OLCs components will be encoded using R_1=$\lceil log_2 M_1 \rceil$=3 bits. Table 5.1 illustrates the encoding of OLCs and their components.

The address of each microinstruction is formed as concatenation of both codes: $A(b_t)$=$K(\alpha_g) * K(b_t)$. In the presented example, vertex b_1 has the address $A(b_1)$=$K(\alpha_1) * K(b_1)$=00000. Vertex b_9 is encoded as $A(b_9)$=$K(\alpha_3)$*$K(b_9)$=10001 and so on. Such encoding is necessary during formation of the control memory content (tab. 5.2).

In order to form excitation functions for the counter and for the register, the transition table of the CMCU ought to be prepared. The table is presented in the tab. 5.3.

Table 5.1: Encoding of OLCs and their components of the CMCU U_5

α_g	$K(\alpha_g)$	b_t	$K(b_t)$	Address $K(\alpha_g) * K(b_t)$
α_1	00	b_1	000	00 000
		b_2	001	00 001
		b_3	010	00 010
α_2	01	b_4	000	01 000
		b_5	001	01 001
		b_6	010	01 100
		b_7	011	01 101
α_3	10	b_8	000	10 000
		b_9	001	10 001
		b_{10}	010	10 010
		b_{11}	011	10 011
		b_{12}	100	10 100
		b_{13}	101	10 101

The transitions table is the base for formation of the excitation function D for the register. OLCs are encoded with R_2=2 bits, thus two variables ought to be calculated from the tab. 5.3:

$$\begin{aligned} d_2 &= \overline{q_2}\cdot\overline{q_1}\cdot\overline{x_1}\cdot\overline{x_2}, \\ d_1 &= \overline{q_2}\cdot\overline{q_1}\cdot(x_1+\overline{x_1}\cdot x_2). \end{aligned} \tag{5.6}$$

Similarly, the excitation function T for the counter is formed:

$$\begin{aligned} t_3 &= 0, \\ t_2 &= t_1 = \overline{q_2}\cdot\overline{q_1}\cdot\overline{x_1}\cdot\overline{x_2}). \end{aligned} \tag{5.7}$$

Finally, the CMCU U_5 is designed using hardware description languages. The control memory is realised using dedicated memory blocks of the FPGA. Figure 5.4 shows the logic schematic of the prototyped controller. Similarly to the technological schematic of CMCUs with mutual memory presented in the previous Chapter, the combinational circuit is implemented with LUTs, the counter is formed from

Table 5.2: The content of the control memory of the CMCU U_5

Vertex	Address	Microinstruction							Comment
		y_0	y_1	y_2	y_3	y_4	y_5	y_K	
b_1	00000	0	1	1	0	0	0	0	I_1^1
b_2	00001	0	0	0	1	1	0	0	–
b_3	00010	1	0	0	0	0	1	0	O_1
b_4	01000	0	0	1	0	0	1	0	I_2^1
b_5	01001	0	1	0	0	0	0	0	–
b_6	01010	0	0	0	1	0	1	0	–
b_7	01011	1	1	0	0	1	0	1	I_2^2 O_2
b_8	10000	0	1	0	1	0	0	0	I_3^1
b_9	10001	0	0	0	0	0	1	0	–
b_{10}	10010	0	0	0	0	1	0	0	–
b_{11}	10011	0	0	1	0	0	0	0	–
b_{12}	10100	0	0	0	1	0	1	0	–
b_{13}	10101	1	1	1	0	0	0	1	O_3

Table 5.3: The table of transitions of the CMCU U_5

α_g	$K(\alpha_g)$		X_h	α_t	$K(\alpha_t)$		b_j	$K(b_j)$			D	T	h
	q_2	q_2			d_1	d_1		t_3	t_2	t_1			
α_1	0	0	x_1	α_2	0	1	b_4	0	0	0	d_1	–	1
α_1	0	0	$\overline{x_1}x_2$	α_2	0	1	b_7	0	1	1	d_1	t_2 t_1	2
α_1	0	0	$\overline{x_1}$ $\overline{x_2}$	α_3	1	0	b_8	0	0	0	d_2	–	3

LUTs and FDC flip-flops, and the control memory is realised as a BRAM. Additionally, the register is implemented with *FDCE* blocks. Such elements are D-type flip-flops with clock enable and asynchronous clear.

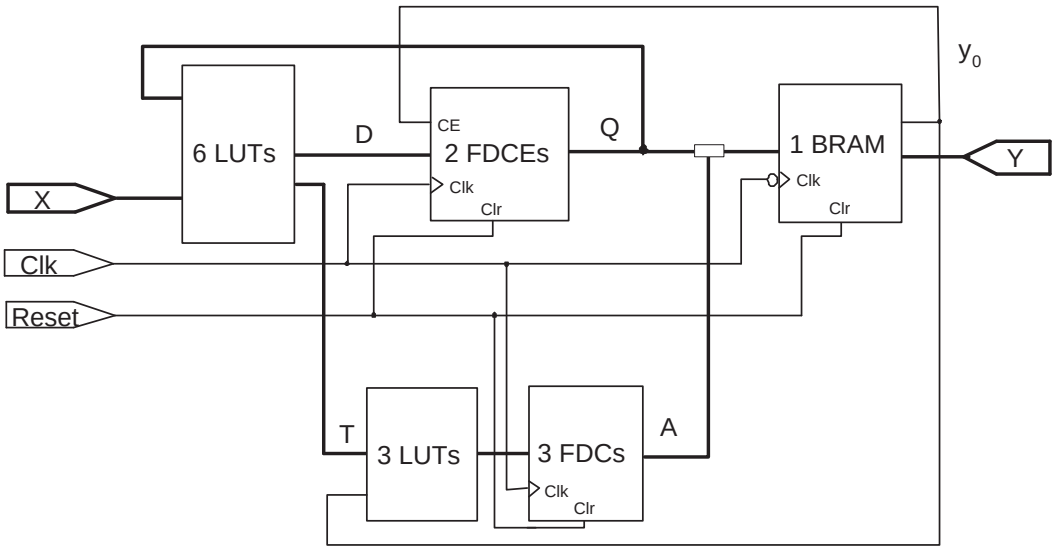

Figure 5.4: Technological structure of the CMCU U_5

5.1.4 Summary

The designing method of the CMCU with sharing codes was presented in this section. The prototyping process of the exemplary CMCU U_5 was shown to bring closer presented ideas. Implementation of such an exemplary controller in the FPGA resulted in two main facts that should be pointed out.

Firstly, the realisation of the CMCU U_5 took 9 LUT elements. Detailed experiments showed that realisation of the controller as the CMCU with sharing codes is almost always better than realisation as a traditional FSM. However, in this particular case the FSM also requires 9 LUT elements of the FPGA. Achieved results were inspiration for researching new methods (modifications) of the presented solution.

Another very important fact is volume of the control memory of the CMCU U_5. There are 13 microinstructions, thus the minimum width of the address is equal to 4. Each microinstruction contains 7 microoperations (see tab. 5.2). Therefore expected volume of the memory is equal to S_{CM}=7*2^4=112. On the other hand, the CMCU U_5 requires 5 bits to encode addresses of microinstructions thus the total volume of the memory took 224 bits. Of course, such a small example does not influence device resources because the area offered by one dedicated memory block

of the FPGA (in this case Xilinx Virtex-II Pro family) is much larger. However, the volume of the dedicated memory block is limited, therefore method of sharing codes may be ineffective because the control memory ought to be decomposed. Application of the address converter solves this problem (Wiśniewski et al., 2006b). The additional block determines the microinstruction address, which is encoded with the minimum number of bits.

Next sections present new structures of the CMCU with sharing codes. All methods are based on the traditional CMCU U_{SC}. The main goal of proposed methods is to reduce the number of logic elements that are used for implementation of the controller, however in the CMCU with address converter, the reduction of the volume of the control memory is performed as well.

5.2 The CMCU with sharing codes and function decoder

The CMCU U_{SD} with sharing codes and function decoder is shown in the fig. 5.5. The main idea is to reduce the number of outputs of the combinational circuit thanks to the encoding of the excitation functions for the counter and the register. Therefore, the number of logic blocks required for implementation of the CMCU is reduced. The additional block - function decoder - decodes and sends proper values for the counter and for the register. Function decoder can be implemented with dedicated memory blocks.

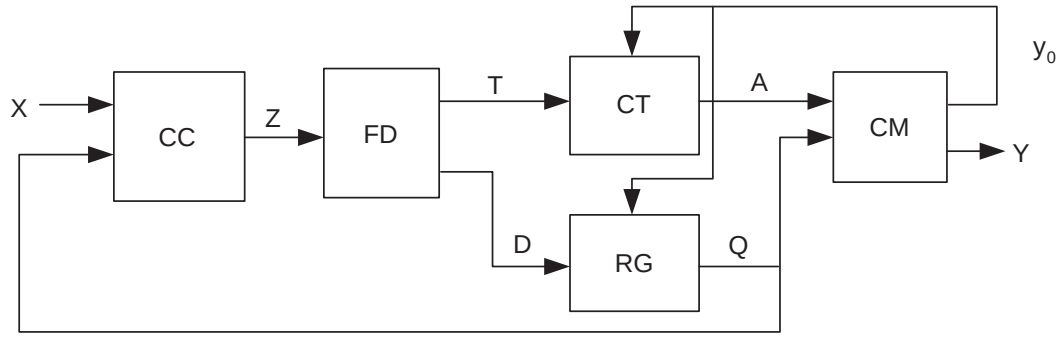

Figure 5.5: The CMCU with sharing codes and function decoder

5.2.1 The main idea of the method

In the CMCU U_{SD}, the set of variables that form the excitation function T for the counter and the set of variables that form the excitation function D for the register are encoded. Similarly to the CMCUs U_{FD} and U_{OF} shown in the previous Chapter, all inputs of the set of OLCs are encoded. The combinational circuit generates the excitation function Z for the function decoder:

$$Z = f(X, Q). \tag{5.8}$$

Function Z contains encoded addresses Q of all inputs I in the set of OLCs. They are further decoded by the circuit FD which indicates the proper code for the counter and for the register:

$$T = f(Z), \tag{5.9}$$

$$D = f(Z). \tag{5.10}$$

5.2.2 Synthesis of the CMCU with sharing codes and function decoder

The method of synthesis of the CMCU U_{SD} includes the following steps:

1. ***Formation of the set of OLCs, encoding OLCs inputs and their components.*** First, the set of OLCs is formed. Then, similarly to the synthesis of the CMCU U_{SC} all OLCs and their components are encoded. Additionally, each input I is encoded with natural binary codes. Finally, each input has the unique code $K(I_j^t)$.

2. ***Addressing microinstructions and the control memory formation.*** The microinstruction address is determined as concatenation of the code of the OLC and its component, according to the (5.1). Then the content of the control memory is formed.

3. ***Formation of the transitions table of the CMCU and formation of the excitation function for the function decoder.*** This table is a base

for the formation of the function (5.8) and synthesis of the circuit CC. The table of transitions contains the following columns: α_g,$K(\alpha_g)$,X_h,I_j^t,$K(I_j^t)$,Z,h. Here:

- α_g is the OLC from which the transition is executed;
- $K(\alpha_g)$ is the code of the OLC α_g;
- X_h is the input signal causing the transition and it is equal to the conjunction of the elements from the set X;
- α_t is the destination OLC where the transition is executed;
- I_j^t is the input of the chain $\alpha_j \in C$ to which the transition is executed;
- $K(I_j^t)$ is the address of the input I_j^t;
- Z is the set of variables that form an excitation function of the function decoder;
- h is the number of transition (h=1, ..., H).

Based on this table, the excitation function Z for the function decoder is formed:

$$Z_r = \bigvee_{h=1}^{H} T_{rh} E_g^h X_h (r = 1, \ldots, R_2), \tag{5.11}$$

where T_{rh} is a Boolean variable that is equal to 1 if and only if the variable T_r is written in the h-th line of the table of transitions; E_g^h is the conjunction of internal variables $Q_r \in Q$ corresponding to the code $K(\alpha_g)$ from the h-th line of the table of transitions.

4. ***Formation of the table of the function decoder.*** Based on the code of each input, function decoder generates excitation functions for the counter and for the register. The table of the function decoder contains the following columns I_j^t, $K(I_j^t)$, α_t,$K(\alpha_t)$,b_j,$K(b_j)$,T,D,m:

   - I_j^t is the input of the chain $\alpha_j \in C$;
   - $K(I_j^t)$ is the code of the input I_j^t;
   - α_t is the destination OLC where the transition is executed;

- $K(\alpha_t)$ is the code of the OLC α_t;
- b_j is the destination component in the OLC α_t where the transition is executed;
- $K(b_j)$ is the code of the component b_j;
- T is the set of variables that form the excitation function for the counter;
- D is the set of variables that form the excitation function for the register;
- m is the consecutive line in the truth-table of the function decoder (m=1, ..., M).

Based on this table, the circuit of function decoder can be implemented with dedicated memory blocks. Outputs of the circuit FD form excitation functions T for the counter and D for the register. The volume of the memory required for the realization of the function decoder can be calculated as $S_{FD}=(R_1+R_2)*2^{R_Z}$, where R_1 is the size of the code generated by the counter, R_2 is the size of the code generated by the register, and R_Z is the number of variables used for the OLCs inputs encoding.

5. ***Implementation of the CMCU*** U_{SD}***.*** The circuit of function decoder is realized as a memory. Thus it can be implemented with dedicated memory blocks of the destination FPGA. The remaining blocks of the CMCU U_{SD} are implemented in the same manner as in case of the CMCU U_{SC}: the CC, the CT and the RG are realized with logic blocks while the CM is implemented using dedicated memory blocks.

5.2.3 Example of synthesis of the CMCU with sharing codes and function decoder

To bring closer the synthesis method of the CMCU U_6 with sharing codes and function decoder, a controller described by the flow-char Γ_2 (fig. 5.2) will be designed. As it was already shown in the previous section, there are M_2=3 OLCs that have M_Z=4 inputs: $I_1^1=b_0$, $I_2^1=b_4$, $I_2^2=b_7$, $I_3^1=b_8$. OLCs are encoded using natural binary code, using $R_2=\lceil log_2 M_2 \rceil$=2 bits: $K(\alpha_1)$=00, $K(\alpha_2)$=01, $K(\alpha_1)$=10. OLCs components are encoded with R_1=3 bits as it was presented in the tab. 5.1.

According to the synthesis rules shown in the previous subsection all inputs are encoded using R_Z=$\lceil log_2 M_Z \rceil$=2 bits: $K(I_1^1)$=00, $K(I_2^1)$=01, $K(I_2^2)$=10, $K(I_3^1)$=11.

In the next step, addresses of microinstructions are encoded and formation of the control memory of the CMCU U_6 is done. This stage is performed exactly in the same manner as it was presented during the synthesis of the CMCU U_5 (see tab. 5.2).

In the third step of the designing process, the table of transitions is formed. Distinct from the table of transitions of the CMCU U_5, now inputs of OLCs are used as destination of transitions. The transitions table of the CMCU U_6 is presented in the tab. 5.4.

Table 5.4: The transition table of the CMCU U_6

α_g	$K(\alpha_g)$		X_h	α_t	I_t^j	$K(I_t^j)$		Z	h
	q_2	q_1				z_2	z_1		
α_1	0	0	x_1	α_2	I_2^1	0	1	z_1	1
α_1	0	0	$\overline{x_1}\ x_2$	α_2	I_2^2	1	0	z_2	2
α_1	0	0	$\overline{x_1}\ \overline{x_2}$	α_3	I_3^1	1	1	$z_2\ z_1$	3

The transition table is the base for formation of the excitation function Z for the function decoder. This function includes two variables Z=$\{z_1$,$z_2\}$:

$$\begin{aligned} z_2 &= \overline{q_2}\cdot\overline{q_1}\cdot\overline{x_1}, \\ z_1 &= \overline{q_2}\cdot\overline{q_1}\cdot(x_1+\overline{x_2}). \end{aligned} \tag{5.12}$$

Function Z keeps the code of the proper input that is selected by the combinational circuit. This code is further decoded by the block FD. Therefore, function decoder generates the address of microinstruction which is concatenation of the proper code $K(\alpha_t)$ of the OLC and the code of its component $K(b_j)$. Table 5.5 presents the table of the function decoder for the CMCU U_6.

The circuit of the function decoder is usually implemented using dedicated memory blocks. However, it can be also realised with logic elements of the FPGA.

Table 5.5: The table of the function decoder of the CMCU U_6

I_j^t	$K(I_j^t)$		α_t	$K(\alpha_t)$		b_j	$K(b_j)$			D	T	h
	z_2	z_1		q_2	q_1		t_3	t_2	t_1			
I_1^1	0	0	α_1	0	0	b_1	0	0	0	–	–	1
I_2^1	0	1	α_2	0	1	b_4	0	0	0	d_1	–	2
I_2^2	1	0	α_2	0	1	b_7	0	1	1	d_1	$t_2\ t_1$	3
I_3^1	1	1	α_3	1	0	b_8	0	0	0	d_2	–	4

In this case functions T and D are formed:

$$\begin{aligned} t_3 &= 0, \\ t_2 &= t_1 = z_2 \cdot \overline{z_1}, \\ d_2 &= z_2 \cdot z_1, \\ d_1 &= z_2 \cdot \overline{z_1} + \overline{z_2} \cdot z_1. \end{aligned} \tag{5.13}$$

Now the CMCU U_6 may be prototyped using HDLs. Both, the control memory and the function decoder are implemented with dedicated memory blocks, while the combinational circuit, the counter and the register are realised with logic blocks of the FPGA. Figure 5.6 shows the logic schematic of the controller.

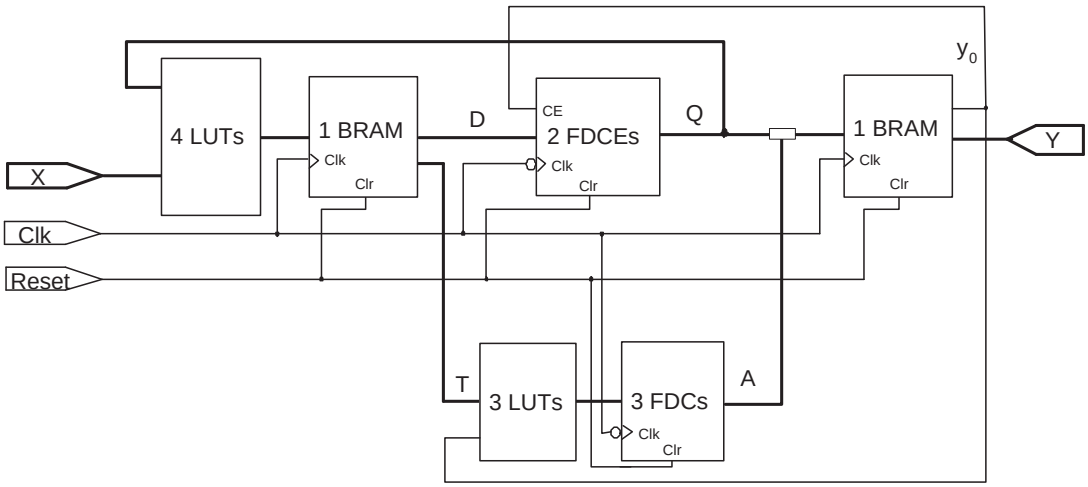

Figure 5.6: Technology structure of the CMCU U_6

5.2.4 Summary

The CMCU with sharing codes and function decoder was introduced in this section. The additional block reduces the number of variables of logic functions that are formed by the combinational circuit. Experiments proved the effectiveness of proposed method. The CMCU U_{SD} average consumes 6% fewer LUT elements than the CMCU U_{SC}.

5.3 The CMCU with address converter

The method of sharing codes makes sense only if the size of codes generated by the register RG and by the counter CT is equal to the width of the microinstruction address (Barkalov, 2002). Then the following condition is fulfilled:

$$R_1 + R_2 = R_3. \tag{5.14}$$

In most cases the total number of bits generated by the register and by the counter exceeds the width of the microinstruction address. The condition (5.14) is violated because $R_1 + R_2 > R_3$ and the volume of the control memory grows drastically. The minimum volume of the memory can be calculated as:

$$S_{CM} = (N + 2) * 2^{R_3}, \tag{5.15}$$

where S_{CM} means the total volume of the control memory, $N + 2$ counts the total number of microoperations kept in the control memory (N is the number of microoperations while two additional bits are formed by y_0 and y_K), and R_3 defines the minimum width of the address. It is clear that **each additional bit in the microinstruction address doubles the total volume of the memory**.

Sections 5.3 and 5.4 show new synthesis idea of the CMCU with sharing codes. The method is based on the application of the additional block (address converter) in the CMCU structure (fig. 5.7). Such an approach has sense only if the condition (5.14) is violated and the total quantity of codes generated by the register and by the counter is greater than the width of the address of the control memory.

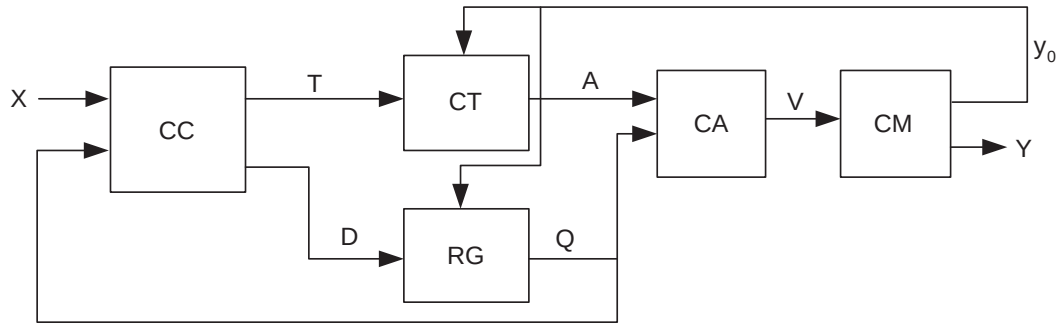

Figure 5.7: The structure of the CMCU with address converter

5.3.1 The main idea of the method

Let $K(\alpha_g)$ be the state code of the register and $K(b_t)$ the state code of the counter. According to the (5.1), the microinstruction address $A(b_t)$ is calculated as the concatenation of these codes:

$$A(b_t) = K(\alpha_g) * K(b_t).$$

In the CMCU U_{CA} the address generated by the register and by the counter is converted by the address converter.

Now the circuit CC forms the system of functions:

$$T = f(X, Q), \tag{5.16}$$

$$D = f(X, Q), \tag{5.17}$$

and the circuit CA converts generated addresses, forming the new function V:

$$V = V(Q, X). \tag{5.18}$$

Here $V = \{v_1, \ldots, v_{R_3}\}$ is the set of addresses of the control memory.

Presented solution permits to combine the positive features of the traditional CMCU with base structure (U_{BS}) and with sharing codes (U_{SC}) such as:

- minimal number of inputs and outputs of the combinational circuit CC (compared with the U_{SC});

- minimal width of an address of the control memory (in comparison with the U_{BS}).

It is clear that application of a given method makes sense only if the implementation of the CMCU with additional address converter requires fewer memory blocks of the destination FPGA than CMCUs based on the standard structure U_{SC}.

5.3.2 Synthesis of the CMCU with address converter

The design method of the CMCU with sharing codes and address converter includes the following steps:

1. ***Formation of the set of OLCs, encoding OLCs and their components***. This step is executed in the same manner as it was presented in previous sections.

2. ***Natural addressing of microinstructions and formation the of control memory content***. Here microinstructions are simply encoded with the natural binary codes. Then the content of the control memory is formed. The volume of the control memory can be calculated as S_{CM}=(N+2)$*2^{R_1+R_2}$.

3. ***Formation of the transitions table of the CMCU***. According to (5.16) and (5.17), this table is the base for the formation of excitation functions for the counter and for the register. Here outputs of the register (function Q) and of the counter (function T) are inputs of the address converter.

   This table contains the following columns: α_g, K(α_g), α_t, K(α_5), I_t^j, $A(\mathrm{I}_t^j)$, X_h, T, D, h, where:

   - $\alpha_g \in C$ is the initial chain of the flow-chart Γ;
   - K(α_g) is the code of the OLC $\alpha_g \in C$;
   - $\alpha_t \in C$ is the destination OLC of the transition;
   - K(α_t) is the code of the destination OLC of the transition;
   - I_t^j is the j-th input of the OLC $\alpha_t \in C$ in which the transition from the output O_g of the OLC $\alpha_g \in C$ is occurred;

- $A(\mathrm{I}_t^j)$ is the address of the component in the OLC α_t corresponding to the input I_t^j;
- X_h is the input signal causing the transition $< O_g, I_t^j >$ and it is equal to the conjunction of the elements from the set X;
- T is the set of variables that form the excitation function for the counter CT (formed with the code $A(\mathrm{I}_t^j)$);
- D is the set of variables that form the excitation function for the register RG (formed with the code $\mathrm{K}(\alpha_t)$);
- h is the number of the transition (h=1, ..., H).

4. ***Formation of the table of the address converter***. In this step the truth-table for the address converter is created. Based on functions generated by the counter and by the register, the microinstruction address is formed.

   The table contains the following columns: α_g, $\mathrm{K}(\alpha_g)$, b_t, $\mathrm{K}(\mathrm{b}_t)$, A_t, V_t, m. Here:

   - $\alpha_g \in C$ is the chain of the flow-chart Γ;
   - $\mathrm{K}(\alpha_g)$ is the code of the OLC $\alpha_g \in C$;
   - $b_t \in B$ is the operational block of the flow-chart Γ;
   - $\mathrm{K}(\alpha_g)$ is the code of the block $b_t \in B$;
   - A_t is the address of the microinstruction encoded in natural binary code;
   - V_t is the column containing the variables $v_r \in V$ that are equal to 1 in the address A_t.
   - m is the consecutive line in the truth-table of the address converter.

5. ***Design and implementation of the logic circuit of the CMCU*** U_{SC}. The circuit CC is implemented using systems (5.16) and (5.17) that are formed from the table of transitions. Depending on the demands, address converter may be implemented using either logic elements or memory blocks of the FPGA. The volume of the memory that is required for the address converter realization can be calculated as $S_{CA}=(R_1 + R_2)*2^{R_3}$.

5.3.3 Example of synthesis of the CMCU with address converter

The synthesis method of the CMCU with address converter will be illustrated by the example. There is the flow-chart Γ_3 shown in the fig. 5.8.

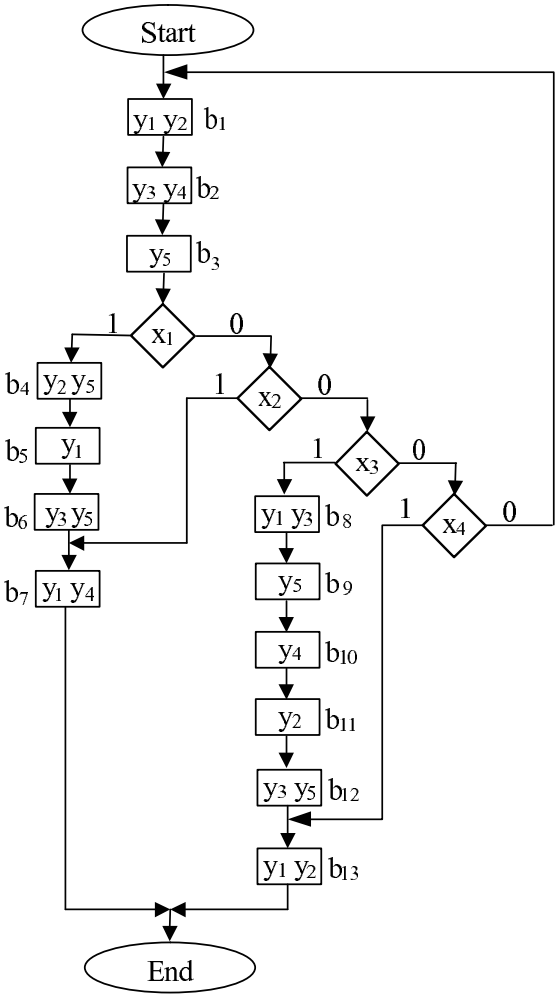

Figure 5.8: The flow-chart Γ_3

Flow-chart Γ_3 describes a hypothetical CMCU U_7. There are M_2=3 OLCs in the set of operational linear chains: $C=\{\alpha_1,\ldots,\alpha_3\}$, where $\alpha_1=\langle b_0,\ldots,b_2\rangle$, $\alpha_2=\langle b_3,\ldots,b_7\rangle$ and $\alpha_3=\langle b_8,\ldots,b_{13}\rangle$. The OLCs flow-chart of the CMCU U_7 is illustrated in the fig. 5.9.

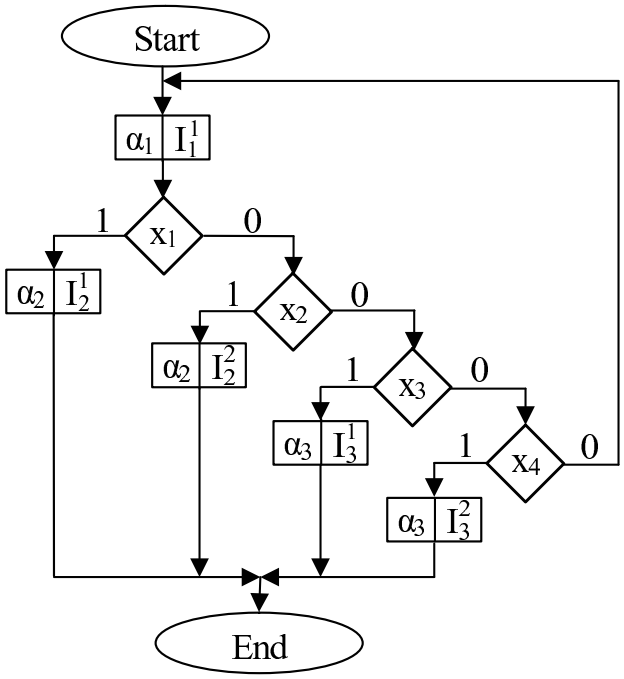

Figure 5.9: The OLCs flow-chart of the CMCU U_7

The longest operational linear chain is α_3 that contains M_1=6 vertices. Therefore, there are R_1=$\lceil log_2 M_1 \rceil$=3 bits required to implement the counter. There are M_2=3 OLCs, thus R_2=$\lceil log_2 M_2 \rceil$=2 variables will be used for encoding of internal states of the controller. The CMCU U_7 contains M_3=13 operational vertices. It means that microinstructions can be addressed with the minimum number of R_3=$\lceil log_2 M_3 \rceil$=4 bits. On the other hand, the total width of an address generated by the counter and by the register is equal to R_1+R_2=5. Therefore, the condition (5.14) is violated (R_1+R_2>R_3) and application of the address converter has sense.

According to the designing rules, OLCs and their components should be encoded. Let's use natural binary codes. Table 5.6 illustrates the encoding of OLCs and their components.

In opposite to the traditional method with sharing codes, the address of each microinstruction is encoded with R_3=4 bits in natural binary code. Therefore, microinstruction corresponding to vertex b_1 has the address $A(b_1)$=0000. Similarly, the address of microinstruction executed in vertex b_9 is encoded as $A(b_9)$=1000 and so on. Now, the content of the control memory can be formed (tab. 5.7).

Table 5.6: Encoding of OLCs and their components of the CMCU U_7

α_g	$K(\alpha_g)$	b_t	$K(b_t)$	α_g	$K(\alpha_g)$	b_t	$K(b_t)$
α_1	00	b_1	000	α_3	10	b_8	000
		b_2	001			b_9	001
		b_3	010			b_{10}	010
α_2	01	b_4	000			b_{11}	011
		b_5	010			b_{12}	100
		b_6	011			b_{13}	101
		b_7	010				

Table 5.7: The content of the control memory of the CMCU U_7

Vertex	Address	Microinstruction							Comment
		y_0	y_1	y_2	y_3	y_4	y_5	y_K	
b_1	0000	0	1	1	0	0	0	0	I_1^1
b_2	0001	0	0	0	1	1	0	0	–
b_3	0010	1	0	0	0	0	1	0	O_1
b_4	0011	0	0	1	0	0	1	0	I_2^1
b_5	0100	0	1	0	0	0	0	0	–
b_6	0101	0	0	0	1	0	1	0	–
b_7	0110	1	1	0	0	1	0	1	I_2^2 O_2
b_8	0111	0	1	0	1	0	0	0	I_3^1
b_9	1000	0	0	0	0	0	1	0	–
b_{10}	1001	0	0	0	0	1	0	0	–
b_{11}	1010	0	0	1	0	0	0	0	–
b_{12}	1011	0	0	0	1	0	1	0	–
b_{13}	1100	1	1	1	0	0	0	1	I_3^2 O_3

In the next step, the transition table of the CMCU should be formed. This table is a base for excitation functions for the counter and for the register. Table 5.8 presents the transition table of the CMCU U_7.

Table 5.8: The transition table of the CMCU U_7

α_g	$K(\alpha_g)$		X_h	α_t	$K(\alpha_t)$		b_j	$K(b_j)$			D	T	h
	q_2	q_1			d_2	d_1		t_3	t_2	t_1			
α_1	0	0	x_1	α_2	0	1	b_4	0	0	0	d_1	–	1
α_1	0	0	$\overline{x_1}\ x_2$	α_2	0	1	b_7	0	1	1	d_1	$t_2\ t_1$	2
α_1	0	0	$\overline{x_1}\ \overline{x_2}\ x_3$	α_3	1	0	b_8	0	0	0	d_2	–	3
α_1	0	0	$\overline{x_1}\ \overline{x_2}\ \overline{x_3}\ x_4$	α_3	1	0	b_{13}	1	0	1	d_2	$t_3\ t_1$	4
α_1	0	0	$\overline{x_1}\ \overline{x_2}\ \overline{x_3}\ \overline{x_4}$	α_1	0	0	b_1	0	0	0	–	–	5

The transition table is a base for formation of the excitation functions D for the register and T for the counter, according to the (5.19).

$$\begin{aligned} d_2 &= \overline{q_2}\cdot\overline{q_1}\cdot\overline{x_1}\cdot\overline{x_2}\cdot(x_3+\overline{x_3}\cdot x_4),\\ d_1 &= \overline{q_2}\cdot\overline{q_1}\cdot(x_1+\overline{x_1}\cdot x_2),\\ t_3 &= \overline{q_2}\cdot\overline{q_1}\cdot\overline{x_1}\cdot\overline{x_2}\cdot\overline{x_3}\cdot x_4,\\ t_2 &= \overline{q_2}\cdot\overline{q_1}\cdot\overline{x_1}\cdot x_2,\\ t_1 &= \overline{q_2}\cdot\overline{q_1}\cdot\overline{x_1}\cdot(x_2+\overline{x_2}\cdot\overline{x_3}\cdot x_4). \end{aligned} \tag{5.19}$$

At the next stage, the truth table for the address converter is prepared. An address of microinstruction is formed based on the OLC code and its component (tab. 5.9).

Finally the CMCU U_7 is designed using hardware description languages. The control memory and the address converter are realised with dedicated memory blocks of the FPGA. Figure 5.10 shows the logic schematic of prototyped controller. There are 13 LUTs required in order to implement the controller.

The main problem of the CMCU U_7 is realisation of the address converter, which is synchronous. Such a module has to form proper address based on codes delivered from the counter and from the register. Both blocks are triggered by the rising edge of a clock signal *Clk*. Additionally, the address ought to be prepared before falling edge of a *Clk*, which synchronizes the control memory. Therefore to ensure proper functionality of the CMCU, additional clock signal *Clk2* was introduced. The address converter should be triggered between rising and falling edges of the *Clk*.

Table 5.9: The table of the address converter

α_g	$K(\alpha_g)$	b_t	$K(b_t)$	A_t	V_t	m
α_1	00	b_1	000	0000	–	1
		b_2	001	0001	v_1	2
		b_3	010	0010	v_2	3
α_2	01	b_4	000	0011	$v_2\ v_1$	4
		b_5	001	0100	v_3	5
		b_6	010	0101	$v_3\ v_1$	6
		b_7	011	0110	$v_3\ v_2$	7
α_3	10	b_8	000	0111	$v_3\ v_2\ v_1$	8
		b_9	001	1000	v_4	9
		b_{10}	010	1001	$v_4\ v_1$	10
		b_{11}	011	1010	$v_4\ v_2$	11
		b_{12}	100	1011	$v_4\ v_2\ v_1$	12
		b_{13}	101	1100	$v_4\ v_3$	13

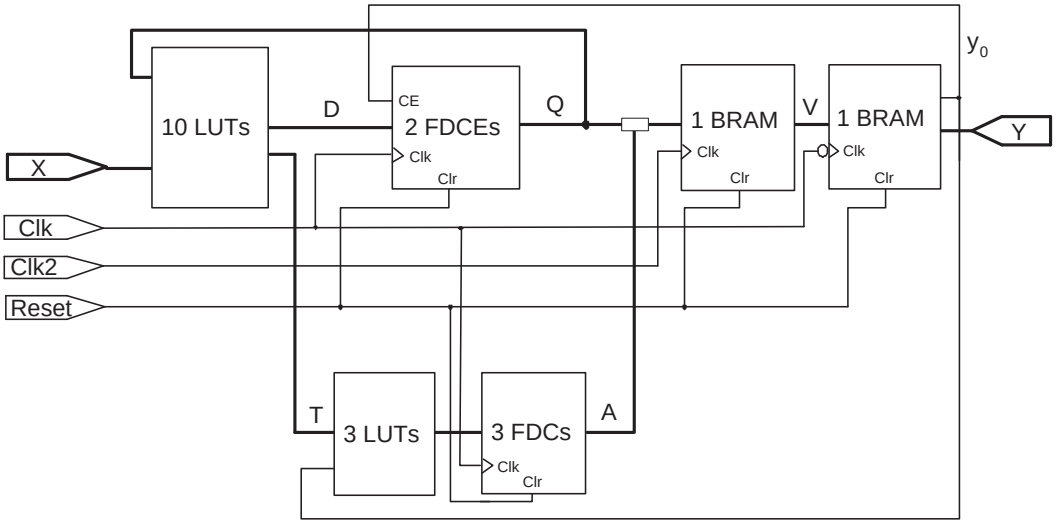

Figure 5.10: Technological structure of CMCU U_7

5.3.4 Summary

There was the CMCU with address converter presented in this section. Application of the additional block has sense only if the total size of codes generated by the

counter and by the register exceeds the width of the address of the control memory. **Results of experiments showed that proposed method average permits to reduce the number of required dedicated memory blocks of an FPGA by 46% in comparison to the traditional CMCU with sharing codes. It should also be emphased out that the number of other resources of an FPGA (LUTs, Flip-Flops, Slices) is the same.**

5.4 The CMCU with address converter and function decoder

This section presents the last method of synthesis of the CMCU with sharing codes that is proposed in the dissertation - the CMCU U_{CD} with address converter and function decoder. Such a controller combines two ideas presented in previous sections. Application of the address converter permits to minimize the volume of control memory if the condition (5.14) is violated, while the additional function decoder reduces required logic elements for implementation of the CMCU.

5.4.1 The main idea of the method

The CMCU U_{CD} with address converter and function decoder is shown in the fig.5.11. Excitation functions T for the counter and D for the register are encoded with the minimum number of bits. Now the combinational circuit CC generates the excitation function Z for the function decoder:

$$Z = f(X, Q). \tag{5.20}$$

Function Z contains encoded addresses Q of all inputs I in the set of OLCs. They are further decoded by the circuit FD which indicates the proper code for the counter and for the register:

$$T = f(Z), \tag{5.21}$$

$$D = f(Z). \tag{5.22}$$

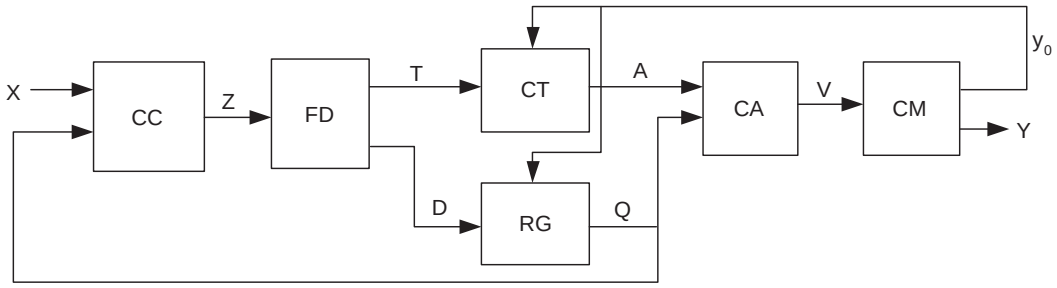

Figure 5.11: The CMCU with address converter and function decoder

Finally the address indicated by the counter and by the register is converted via the circuit CA:

$$V = f(T, D). \tag{5.23}$$

5.4.2 Synthesis of the CMCU with address converter and function decoder

The synthesis process of the CMCU U_{CD} is a combination of designing flows of CMCUs U_{SD} and U_{CA}. Therefore only the most important stages will be presented (more details were shown in the previous sections).

The designing method of the CMCU with address converter and function decoder includes the following steps:

1. ***Formation of set of OLCs, encoding OLCs inputs and OLCs components.*** First, the set of OLCs is formed. Then, similarly to the synthesis of the CMCU U_{SC}, all OLCs and their components are encoded. Additionally, each input I is encoded with natural binary codes. Finally, each input has an unique code $K(I_j^t)$

2. ***Natural addressing of microinstructions and formation of the control memory content.*** Here microinstructions are encoded with natural binary codes. Then the content of the control memory is formed.

3. ***Formation of the transition table of the CMCU*** U_{CD}***.*** According to the (5.20) this table is the base for formation of the excitation function for the function decoder.

4. ***Formation of the table of the function decoder.*** Based on the code of each OLC input, the function decoder generates the proper excitation functions for the counter and for the register.

5. ***Formation of the table of address converter.*** In this step the truth-table for the address converter is created. Based on functions generated by the counter and by the register an address of the microinstruction is formed.

6. ***Design and implementation of the logic circuit of the CMCU*** U_{SC}***.*** Three blocks (CC, CT and RG) of the CMCU are implemented with logic elements of the FPGA while circuits CM, FD and CA are realized with dedicated memory blocks.

5.4.3 Example of synthesis of the CMCU with address converter and function decoder

To bring closer the synthesis method of the CMCU U_8 with address converter and function decoder, the controller described by the flow-char Γ_3 (fig. 5.8) will be designed once more. There are M_2=3 OLCs. The encoding of OLCs and their components is performed in the same manner at it was shown in the previous section (see tab. 5.6). There are M_Z=5 OLCs inputs: I_1^1=b_0, I_2^1=b_4, I_2^2=b_7, I_3^1=b_8 and I_3^2=b_{13}. Therefore, R_Z=$\lceil log_2 M_Z \rceil$=3 bits are required to encode all inputs: $K(I_1^1)$=000, $K(I_2^1)$=001, $K(I_2^2)$=010, $K(I_3^1)$=011, $K(I_3^2)$=100.

At the second stage, microinstructions are addressed with natural binary code. Then the content of the control memory is formed. This step is executed identically as it was shown during synthesis method of CMCU U_7 with address converter (see tab. 5.7).

Next, the transitions table of the CMCU U_8 is formed. The table is a base for the excitation function for the module FD. Table 5.10 presents the content of the table of transitions of the CMCU U_8.

Table 5.10: The transition table of the CMCU U_8

α_g	$K(\alpha_g)$		X_h	α_t	I_t^j	$K(I_t^j)$			Z	h
	q_2	q_1				z_3	z_2	z_1		
α_1	0	0	x_1	α_2	b_4	0	0	1	z_1	1
α_1	0	0	$\overline{x_1}\ x_2$	α_2	b_7	0	1	0	z_2	2
α_1	0	0	$\overline{x_1}\ \overline{x_2}\ x_3$	α_3	b_8	0	1	1	$z_2\ z_1$	3
α_1	0	0	$\overline{x_1}\ \overline{x_2}\ \overline{x_3}\ x_4$	α_3	b_{13}	1	0	0	z_3	4
α_1	0	0	$\overline{x_1}\ \overline{x_2}\ \overline{x_3}\ \overline{x_4}$	α_1	b_{13}	0	0	0	–	5

From the table of transitions, excitation function Z for the function decoder is formed. The function consist of $|Z|$=3 variables:

$$\begin{aligned} z_3 &= \overline{q_2}\cdot\overline{q_1}\cdot\overline{x_1}\cdot\overline{x_2}\cdot\overline{x_3}\cdot x_4, \\ z_2 &= \overline{q_2}\cdot\overline{q_1}\cdot\overline{x_1}\cdot(x_2+\cdot\,\overline{x_2}\cdot x_3), \\ z_1 &= \overline{q_2}\cdot\overline{q_1}\cdot(x_1+\overline{x_1}\cdot\overline{x_2}\cdot x_3). \end{aligned} \tag{5.24}$$

The truth table of the function decoder is prepared at the next stage. Based on the function Z, the module FD indicates proper values for the counter and for the register (tab. 5.11).

Table 5.11: The truth table of the function decoder of the CMCU U_8

I_j^t	$K(I_j^t)$			α_t	$K(\alpha_t)$		b_j	$K(b_j)$			D	T	h
	z_3	z_2	z_1		d_2	d_1		t_3	t_2	t_1			
I_1^1	0	0	0	α_1	0	0	b_1	0	0	0	–	–	1
I_2^1	0	0	1	α_2	0	1	b_4	0	0	0	d_1	–	2
I_2^2	0	1	0	α_2	0	1	b_7	0	1	1	d_1	$t_2\ t_1$	3
I_3^1	0	1	1	α_3	1	0	b_8	0	0	0	d_2	–	4
I_3^2	1	0	0	α_3	1	0	b_{13}	1	0	1	d_2	$t_3\ t_1$	4

At the last step, the content of the address converter should be determined. The truth table for the module CA is exactly the same as it was shown in the previous section (tab. 5.9). Finally the CMCU U_8 can be designed. Now three

blocks: the function decoder, the address converter and the control memory are implemented with dedicated memory blocks of an FPGA. Figure 5.12 shows the technological structure of the CMCU U_8.

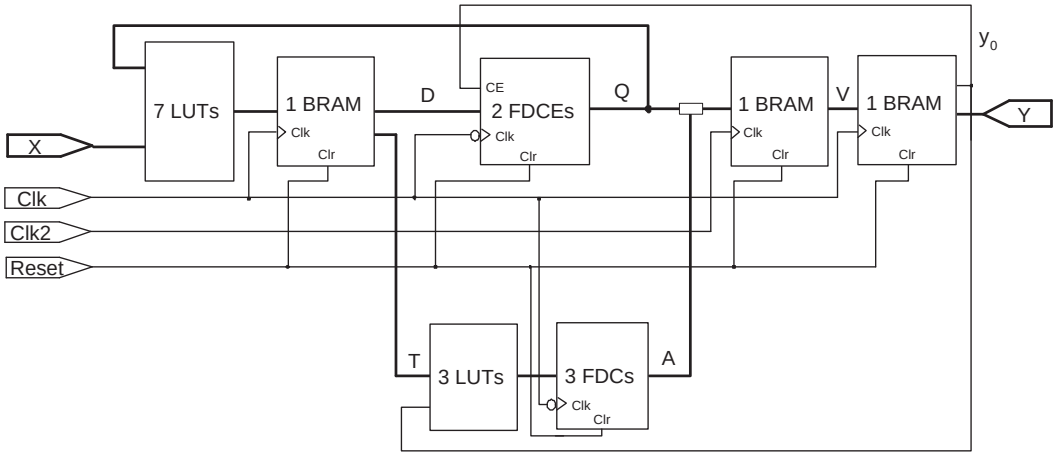

Figure 5.12: Technological structure of the CMCU U_8

The implementation of the CMCU U_8 requires 10 LUTs. Therefore, the number of such elements was reduced by 23% in comparison to the CMCU U_7 with address converter.

5.4.4 Summary

There was the designing method of the CMCU U_{CD} with address converter and function decoder presented in this section. The reduction of the number of LUTs is reached thanks encoding of the excitation functions for the counter and for the register. Additionally, application of an address converter permits to keep the minimum volume of the control memory. It should be pointed out, that performed experiments proved the effectiveness of proposed method. The CMCU U_{CD} average permits to reduce the number of logic blocks of the destination FPGA by 50% in comparison to the traditional FSM.

5.5 Conclusions

There were four CMCUs designing methods presented in this Chapter. The first one - the CMCU U_{SC} with sharing codes - was initially proposed by prof. A. Barkalov and oriented to CPLDs. Therefore, it ought to be adapted to the FPGA. All remaining synthesis methods are based on the CMCU with sharing codes. The aim of all methods is to reduce the number of logic blocks or - in case of application of the address converter - reduction of the control memory volume.

In the CMCU U_{SD} with sharing codes and function decoder, excitation functions for the counter and for the register are encoded with the minimum number of bits. These functions are further decoded by the function decoder. An additional circuit is realized with dedicated memory blocks of an FPGA. Therefore implementation of the CMCU U_{SD} requires less logic blocks than CMCU U_{SC}. Detailed experiments (presented in Chapter 8 proved the effectiveness of an application of the function decoder. **Realisation of the controller as a CMCU U_{SD} reduces the number of required logic blocks of an FPGA average by 13%**.

Two remaining synthesis methods refer to application of an address converter. The additional circuit is in charge of keeping the minimum volume of the control memory. Application of the address converter has sense only if the total length of codes generated by the counter and by the register exceeds the minimum width of microinstructions addresses. Conducted investigations showed that in case of controllers, where the control memory ought to be decomposed (their volume exceeds the volume offered by dedicated memory blocks of the FPGA) the CMCU U_{CA} with address converter **reduces the number of dedicated memory blocks of the FPGA on average by 46%** in comparison to the CMCU U_{SC}.

Ideas presented in CMCUs U_{SD} and U_{CA} were combined in the last synthesis method of the compositional microprogram control unit with address converter and function decoder. In the CMCU U_{CD} both address converter and function decoder are applied. Results of experiments showed that the **CMCU with address converter and function decoder** requires the least resources of an FPGA of all methods where the operation of sharing codes was applied. It should be pointed out that the **presented method permits to reduce the number of logic blocks of the destination device on average by 49% in comparison to the FSM**. On the other hand, realisation of the controller as a CMCU U_{CD} requires

more dedicated memory blocks than traditional automaton by 86%. Therefore, the proposed synthesis method is the best solution if the total volume offered by dedicated memory blocks of an FPGA does not exceed the volume of the control memory of the CMCU. In the other cases, one of synthesis methods of the CMCU with mutual memory presented in the previous Chapter should be used instead.

Chapter 6

Partial reconfiguration of CMCUs implemented in the FPGA

This Chapter deals with partial reconfiguration of compositional microprogram control units implemented in the FPGA. In traditional prototyping methods of the CMCU, the content of the control memory is realised with logic elements of the FPGA. However, the most newer FPGAs offer additionally blocks of dedicated memory that are integrated with the device. Therefore, the content of the control memory can be easily implemented with dedicated memories of the FPGA (Wiśniewski, 2005; Barkalov et al., 2005*d*). The functionality of the CMCU prepared in such a way can easily be changed. This Chapter introduces the new idea of the partial implementation of the CMCU. Designers are able to modify only a few microinstructions of the CMCU. In case of traditional implementation the whole content of the FPGA ought to be replaced. Partial reconfiguration of the CMCU permits to change only the content of the control memory while the rest of the system is not modified.

Partial reconfiguration of FPGA devices is a relatively new idea. Therefore, not all programmable devices offer the reconfiguration of the part of their resources. Such a solution refers especially to devices from Xilix, Altera and Atmel.

The XC2VP30 device (Virtex-II Pro family) from Xilinx was selected as the base FPGA for the further analysis (Xilinx, 2007). Such a device permits partial reconfiguration and it is available at the University of Zielona Góra. All presented FPGA structures, researches and performed experiments refer to this FPGA.

6.1 Introduction to partial reconfiguration of FPGA devices

This section introduces partial reconfiguration of FPGA devices. From the functionality of the design, partial reconfiguration can be divided into two groups:

- ***Dynamic partial reconfiguration*** - also known as an **active** partial reconfiguration - permits to change the part of the device while the rest of the FPGA is still running.
- ***Static partial reconfiguration*** - the device is not active during the reconfiguration process. While the partial data is sent into the FPGA, the rest of the device is stopped (in the shutdown mode) and brought up after the configuration is completed.

There are two styles of partial reconfiguration of FPGA devices from Xilinx: module-based and difference-based.

- ***Module-based partial reconfiguration*** permits to reconfigure distinct modular parts of the design. To ensure the communication across the reconfigurable module boundaries, special bus macros ought to be prepared. It works as a fixed routing bridge that connects the reconfigurable module with the rest part of the design. Module-based partial reconfiguration requires to perform a set of specific guidelines during the stage of design specification that is detailed in (Xilinx, 2004). Finally, for each reconfigurable module of the design, separate bit-stream is created. Such a bit-stream is used to perform the partial reconfiguration of the FPGA.
- ***Difference-based partial reconfiguration*** can be used when a small change is made to the design. It is especially useful in case of changing LUT equations or the dedicated memory blocks content. The partial bit-stream contains only information about differences between the current design structure (that resides in the FPGA) and the new content of the FPGA. There are two ways of difference-based reconfiguration known as a front-end and back-end. The first one is based on the modification of the design in the hardware description languages (HDLs). It is clear that such a solution requires

full repeating of the synthesis and implementation processes. The back-end difference-based partial reconfiguration permits to make changes at the implementation stage of the prototyping flow. Therefore, there is no need for re-synthesis of the design. The usage of both methods (either front-end or back-end) leads to creation of a partial bit-stream that can be used for a partial reconfiguration of the FPGA.

All researches and experiments presented in the dissertation are based on the **static difference-based partial reconfiguration**. Such a method was chosen because of the structure of the CMCU. Difference-based partial reconfiguration permits to change the content of the control memory at the implementation stage. Therefore, most steps of the prototyping flow can be omitted. Moreover, the designer can prepare more than one partial bit-streams with alternative versions of the content of control memory. They can be very easily switched in the FPGA (the full bit-stream is sent only once).

Next section presents the organization of dedicated memory blocks of Xilinx FPGAs. Such an organization is very important for understanding of a mechanism of partial reconfiguration.

6.2 The mechanism of partial reconfiguration of Xilinx FPGAs

Figure 6.1 presents the structure of the typical FPGA device from Xilinx. As it was already shown in Chapter 2, main elements of the device are *Configurable Logic Blocks (CLB)* which create matrix of connected blocks. Each CLB contains two logic elements called *Slices*. Furthermore, each Slice is built from two *Look-Up Tables (LUT)* that perform all logic functions. Therefore, all logic elements of the CMCU such as combinational circuit, register and counter are implemented using CLBs. Moreover, the FPGA contains dedicated memory blocks called *Block-RAMs (or just BRAMs)*.

Block-RAMs are organized in *columns*. The number of columns and BRAMs in each column is different and it depends on the particular FPGA. For an example, the device XC2VP30 (Virtex II Pro family) contains 136 dedicated memories which are grouped in 8 columns organized as 2x20 (two columns containing 20 BRAMs),

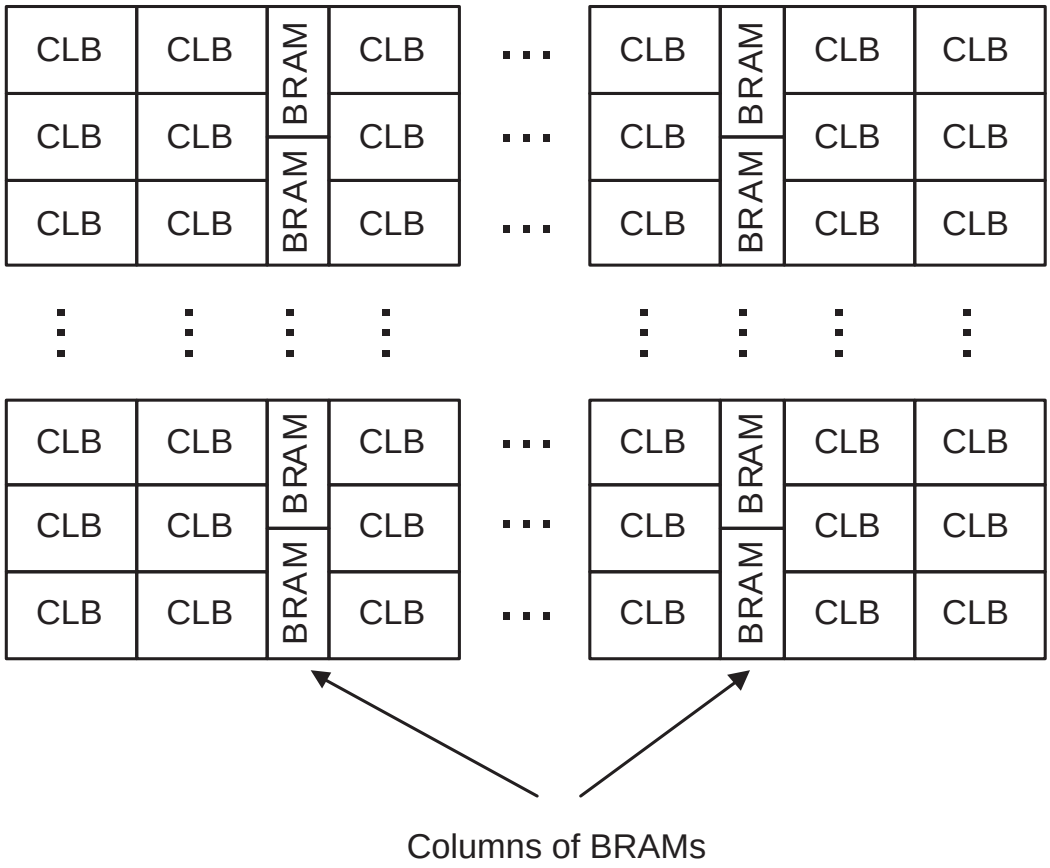

Figure 6.1: The structure of the FPGA device

2x18, 2x16 and 2x14 (Xilinx, 2007). Additionally each BRAM is divided into *lines* (also called as *INITs*). They are used for initialization, configuration and partial reconfiguration of the block. There are 64 lines per each BRAM (counted hexadecimal from *INIT_00* to *INIT_3F*).

Both full and partial bit-stream that is used for configuration of the device consist of *frames*. Each frame contains a portion of information about the implemented design. In case of partial reconfiguration, there are only different frames sent to the FPGA. What is very important, partial reconfiguration of Xilinx devices operates on the whole column of BRAMs. It means that modification of one microoperation in one BRAM causes reconfiguration of all dedicated memories that belong to the same column. In case of the XC2VP30 device, each column of

BRAMs is divided into 64 frames. One frame corresponds to one line (INIT) in all BRAMs in the column (for example modification of two frames means reconfiguration of two lines in all blocks that belongs to the column). Therefore, each frame contains a portion of information about all BRAMs that are organized in the column (fig. 6.2).

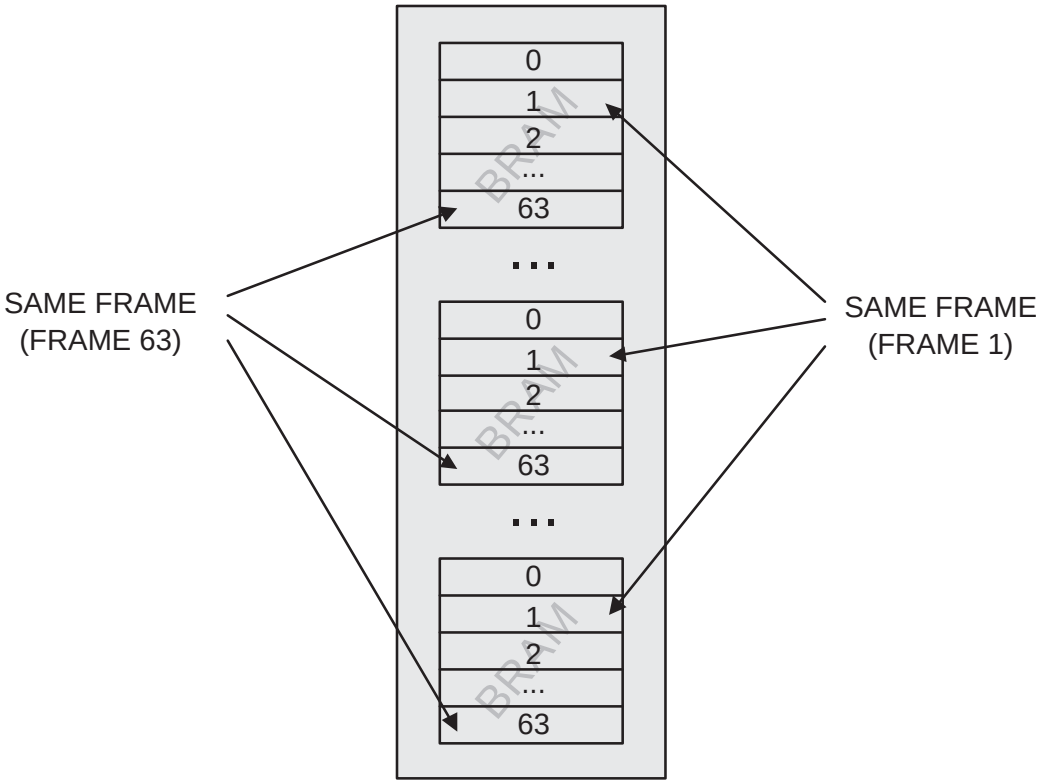

Figure 6.2: Organization of BRAMs

Next section presents the current prototyping flow of CMCUs. Such a designing process does not include the idea of partial reconfiguration. Therefore, further section introduces modified prototyping flow based on the partial reconfiguration of CMCUs implemented in the FPGA.

6.3 The traditional prototyping flow of control units

In order to show the idea of the partial reconfiguration of CMCUs implemented in the FPGA, the traditional prototyping flow of controllers will be presented. Figure 6.3 illustrates the designing process of a typical digital system (Wiśniewski and Węgrzyn, 2005; Parnell and Mehta, 2003), which can be applied in case of CMCUs prototyping flow.

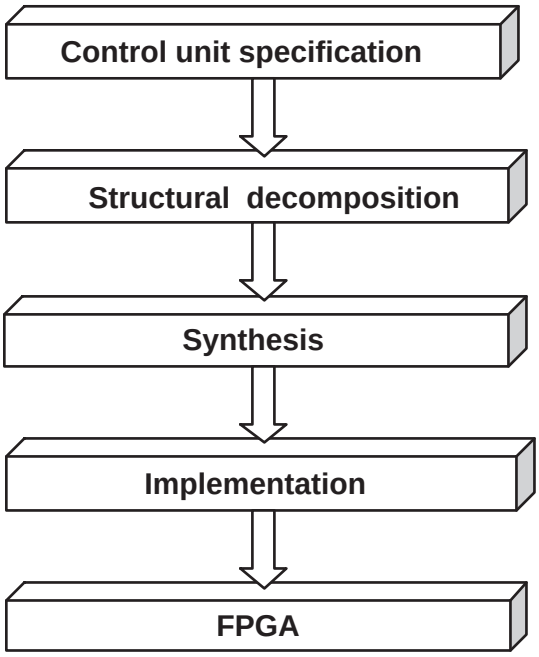

Figure 6.3: The traditional prototyping flow

At the beginning, the specification and structure of the CMCU ought to be prepared. The structure of the controller is prepared according to synthesis methods presented in previous Chapters. Finally, the CMCU may be designed according to the following steps:

1. ***Description of the compositional microprogram control unit prepared with HDL languages.*** At this stage, all modules (combinational circuit, register, counter and control memory) of the further control system are created. The specification of the control memory content is not required now, however the designer can specify initial values for the controller. Next

description of the compositional microprogram control unit should be verified at the software simulator. It allows to avoid most functional errors of the design.

2. ***Logical synthesis of the design.*** The synthesis process converts the design described with HDLs into the gate level. There are gates, logic blocks and connections between them created as a result of synthesis (also known as a "netlist"). This process is the same as it is in the traditional prototyping flow.

3. ***Logical implementation of the design.*** At this stage, the logic implementation of the CMCU is performed. As a result of the implementation process, the bit-stream is produced. It contains full description of the design that will be sent to the device for configuration of the FPGA.

4. ***Hardware implementation of the design.*** The FPGA is configured with the bit-stream produced in the previous step.

It is clear that any modification of the content of the control memory requires repeating the full prototyping flow. Therefore, if there is a need to implement another version of the CMCU, all steps ought to be performed, even if the designer wants to change only one bit of the control memory.

Next section presents the new idea of the prototyping flow of the CMCU. The method is based on the partial reconfiguration of the FPGA devices.

6.4 Partial reconfiguration of CMCUs implemented in the FPGA

The prototyping flow for the control unit that should be prepared for further reconfiguration is similar to the traditional prototyping process. Therefore at the beginning, the design should be described using Hardware Description Languages (HDL) like Verilog or VHDL. Then it should be verified to avoid further functional errors. After the verification, the design is synthesized. The difference between proposed and traditional prototyping flows is implementation process. At this stage, the content of the further control memory is prepared. As the result of

the implementation process, the bit-stream is created. It contains full information about configuration of destination FPGA device. Therefore, the size of the file is respectively large. That also means the long FPGA configuration time (Barkalov et al., 2005*f*; Barkalov and Wiśniewski, 2005a; Barkalov et al., 2005*c*).

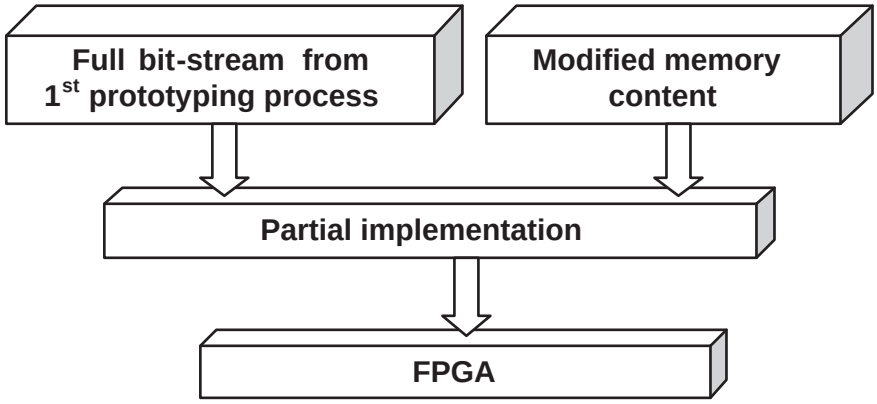

Figure 6.4: The modified prototyping flow including the operation of partial reconfiguration of CMCUs implemented in the FPGA

The method of partial reconfiguration of the control unit includes the following steps:

1. ***Description of the compositional microprogram control unit prepared with HDL languages.*** This step is performed in the same manner as it is in the traditional prototyping flow. Next, the CMCU should be verified at the software simulator. It allows to avoid most functional errors of the design.

2. ***Logical synthesis of the design.*** This process is the same as it is in the traditional prototyping flow.

3. ***Formation of the control memory content.*** Now the content of the control memory is created. The designer can prepare as many versions of the control memory content as it is necessary.

4. ***Logical implementation of the first version of the design.*** As the result of the logic implementation process, the bit-stream is produced. It

contains first description of the design that will be sent to the device for configuration of the FPGA.

5. ***Hardware implementation of the design.*** At this stage the FPGA is configured for the first time. Therefore, whole description about the device must be specified in the bit-stream.

6. ***Modification of the control memory content.*** At this step, the content of the control memory should be replaced with alternative values that were previously prepared at stage 3. The modification is performed during logic implementation. The content of the control memory can be specified in many ways - by an *.ucf* file or via Xilinx tools like *FPGA Editor*, see (Xilinx, 2004) for more detail.

7. ***Preparation of the difference bit-stream.*** Now the new bit-stream is created. It contains only the differences between the new version of the design and previous one, that is already implemented in the FPGA. In fact, the bit-stream will contain only information about modified elements of the control memory (Xilinx, 2007).

   Steps 6 and 7 should be repeated for each version of the control memory content that was prepared at stage 3.

8. ***Partial reconfiguration of the device.*** Using bit-streams that were produced in step 3, the device can be partially reconfigured. The functionality of the control unit can be changed very easily and very fast, because only different frames between the modified and already implemented designs are sent to the FPGA.

6.5 Example of partial reconfiguration of the CMCU implemented in the FPGA

The idea of partial reconfiguration of the compositional microprogram control unit will be shown by the example of the traffic lights driver. It is a simplified version of the control unit just to show the benefits of the partial reconfiguration of the CMCU.

The driver controls the traffic lights for vehicles and pedestrians on the crossroad. It is assumed that the CMCU works in the following rules:

- the crossroad is completely collision-free,
- each road has three independent traffic lines and three independent traffic lights, each for vehicles turning left, going straight and turning right,
- to make the design more clear, yellow lights are not considered. There are only two signals for vehicles and pedestrians. Green light means "go/walk" and red one says "wait/stop".

The main goal for designers of such traffic light drivers is to minimize traffic bottlenecks and jams. In case of the daytime (morning, before noon, after noon, etc.) more privileges ought to be admitted for vehicles or pedestrians.

In the example, two versions of traffic driver are proposed. The first one gives more privileges for vehicles. The simplified model of the design can be described by four main states.

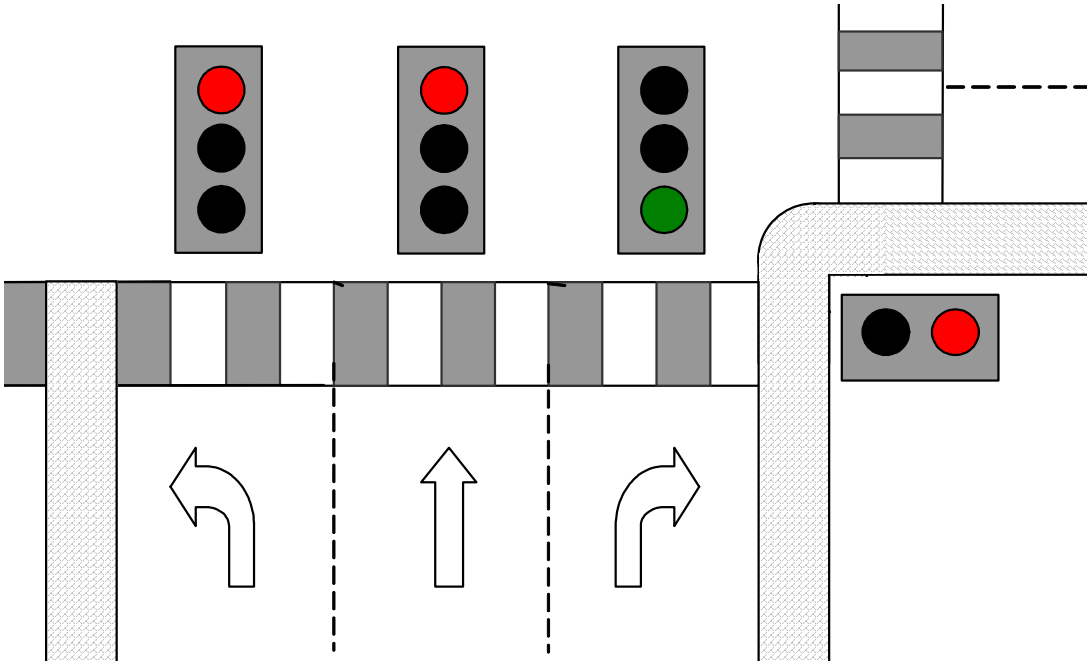

Figure 6.5: The first version of the 3^{rd} state of the traffic light driver

In the first state green light is set for vehicles turning left. The "Stop" signal is shown for another vehicle lines. Pedestrians also have to wait. At the second stage, vehicles that are going straight and turning right may pass trough the crossroad. The light for cars turning left is changed to red. Pedestrians still wait. The third step allows turning right while another lights are set to red. It means that none of drivers turning left, going straight or pedestrians can't go. This stage was illustrated by the fig. 6.5. Finally, at the last stage pedestrians can walk safely across the street because all three lights for vehicles are set to red.

It has to be pointed out that such a solution of the traffic lights is very comfortable for vehicles but pedestrians can across the street only in one of the four states. Therefore, the second version of the driver assumes more privileges for pedestrians. In this design only the third state was modified (fig. 6.6). Now all lights for vehicles are set to red. It means, that green light for pedestrians is shown and they can walk across the street. Such a small modification changes the whole traffic cycle, because now pedestrians may safely cross the street twice as frequent as it was before.

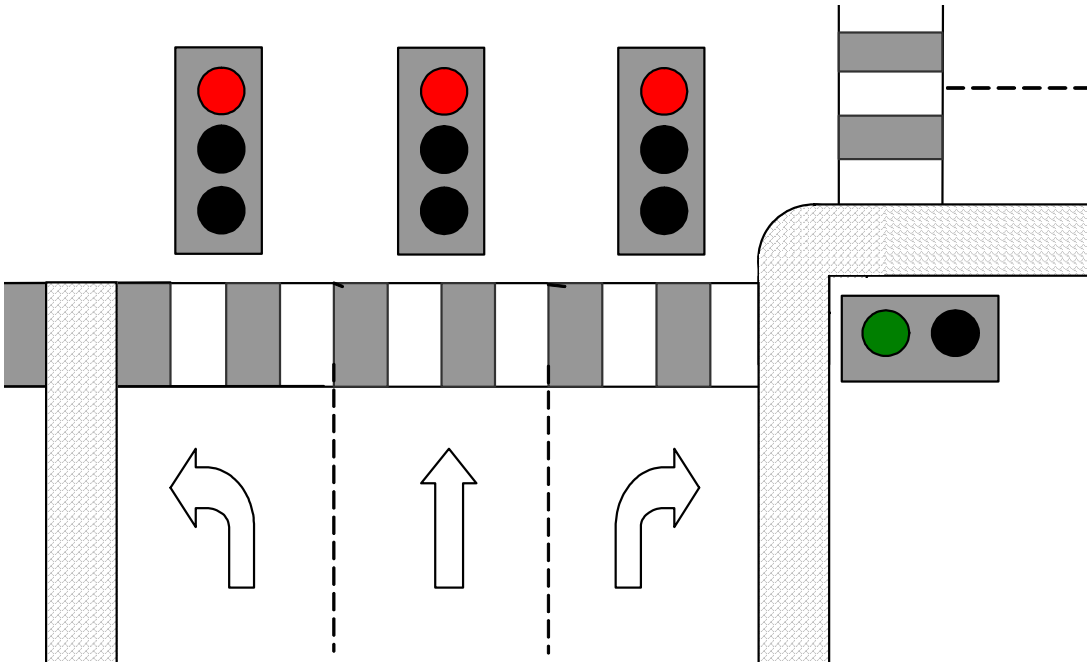

Figure 6.6: The second version of the 3^{rd} state of the traffic light driver

The design of the traffic light driver was prepared and implemented using the XC2VP30 device (Xilinx Virtex-II Pro family). For both versions full bit-streams and partial reconfiguration data were prepared. Table 6.1 shows the results that were achieved during configuration. The size of the bit-stream and approximate time that is needed for device configuration are presented.

Table 6.1: Results achieved during implementation of the traffic light driver

	Full bit-stream	Partial-bit stream
Size [bytes]	1 448 816	**2 696**
Time [s]	4.5	>0.1

Above table shows that size of the original bit-stream was highly reduced. During the first configuration of the FPGA over 1448K bytes have to be sent. In case of partial reconfiguration, only 2,5K bytes were required. It means that original size of the bit-stream was reduced by over 99,81%.

6.6 Conclusions

The idea of partial reconfiguration of CMCUs implemented in the FPGA was shown in this Chapter. Moreover, the new prototyping flow of CMCUs were proposed. The modified designing method is based on partial reconfiguration of a controller implemented in the FPGA. There is only the control memory content replaced while the rest of the system is not modified. In the presented prototyping flow, logic synthesis and implementation are performed only once. Therefore, such a realisation highly accelerates whole prototyping process.

Performed experiments showed that the **original bit-stream that is sent to the FPGA can be reduced even over 500 times**. Detailed results of investigations of the effectiveness of partial reconfiguration of CMCUs implemented in the programmable devices are presented in Chapter 8.

Chapter 7

The CAD-Tool for Automatic synthesis of CMCUs (ATOMIC)

This Chapter introduces the dedicated CAD-Tool that was prepared to perform the **A**u**TOM**atic synthes**I**s of **C**MCUs (ATOMIC). Based on the description of the controller as a flow-chart, ATOMIC produces a code in the hardware description language (Verilog). Such a code is ready for the logic synthesis and further implementation in the FPGA. There are main features shown in this Chapter. The detailed description of input and output data formats, switches and parameters are presented in Appendix A.

7.1 Overview of ATOMIC

ATOMIC implements all 8 methods presented in Chapters 5 and 6. Based on the description of the controller, ATOMIC generates the code in the Verilog-HDL. There are three main modules that consist of ATOMIC (fig. 7.1).

The first module (***fc2olc***) analyses the structure of the flow-chart and produces the set of operational linear chains. This step is common for all implemented methods. The second module (***olc2mcu***) based on the description of OLCs and the chosen method performs the structural decomposition process. All required data (excitation functions, description of the control memory, etc.) are stored using intermediate format (see Appendix A). Such a format may be a base for various ways of the CMCU description; for example the Verilog or VHDL code

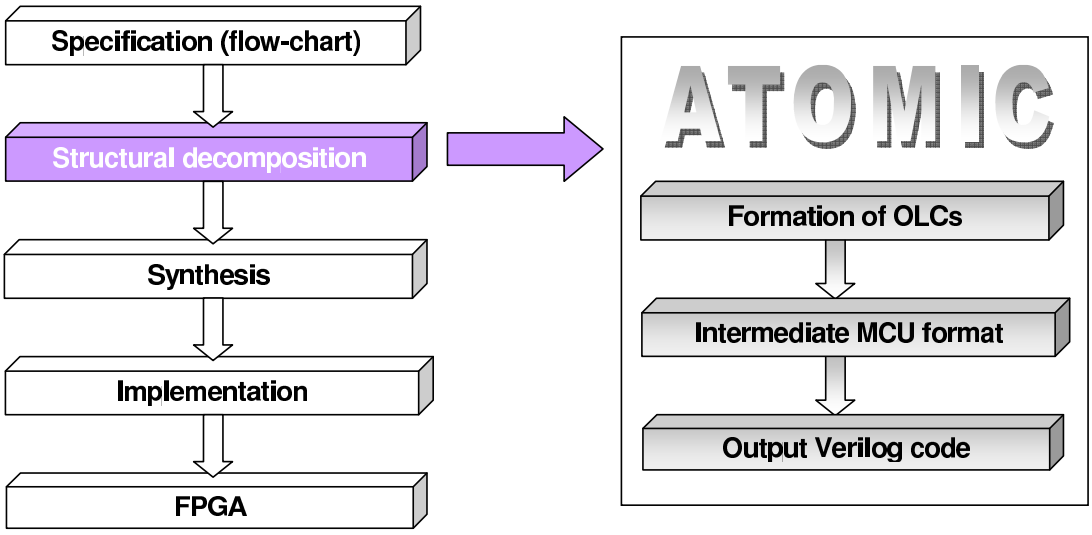

Figure 7.1: The structure of ATOMIC

may be very easily produced. The last module of ATOMIC - (**mcu2verilog**) - generates direct description of the CMCU using the Verilog-HDL language. The description is ready for logic synthesis and implementation.

ATOMIC was prepared as the module-based tool in order to improve its performance. At each stage the description of the prototyped controller may be changed. Furthermore, once prepared OLC description may be commonly used as the input for all eight implemented synthesis methods.

The very important feature is the possibility of the external tools usage for further analysis. **Each excitation function that is produced by the *olc2mcu* module may be decomposed with another systems that are based on functional decomposition like SIS, DEMAIN, etc. Therefore, both structural and functional decomposition can be used in the prototyping flow of the CMCU**. The control unit is initially decomposed with structural procedures and then excitation functions produced for internal blocks of the CMCU are optimized with functional decomposition. **Such a solution saves the structure of the CMCU which leads to possibility of partial reconfiguration of the controller (see Chapter 8)**. Obviously to perform this task, the excitation function has to be converted into the proper format, however it is a relatively easy process. It also has to be pointed out that the idea of joined structural and

functional decomposition in the prototyping flow of CMCUs will be investigated in the future in more details and it is not within the scope of the dissertation.

7.2 Realisation of ATOMIC

ATOMIC was implemented using the standard C++ language (Kernighan and Ritchie, 1977; Stroustrup, 1986). It does not include any additional libraries or external units, thus it can be easily compiled under any system and any platform that supports the C++ language (Windows, Solaris, Unix, etc.).

From the mathematical point of view, ATOMIC is based on the graph theory and implements methods of searching graphs (Berge, 1985; Wilson, 1979; Harary, 1994; Aho et al., 1974). Both modules *fc2olc* and *olc2mcu* read the input data and dynamically form the adequate oriented graph that contains all information about the flow-chart (or the OLC flow-chart). Such a graph can easily be searched using standard graph searching methods. ATOMIC operates on the modified algorithm of the DFS (Cormen et al., 2001). Such a method was used during realisation of the module *fc2olc* and *olc2mcu* as well. The original algorithm was modified and fit to perform adequate operations. In case of the unit *fc2olc*, operational vertices are replaced by the adequate OLC during the searching process. Here traditional stack is used. In case of the *olc2mcu* module, adequate excitation functions are formed during the searching operation. It has to be pointed out, that functions for all the modules (counter, register, function decoder) are formed during the same step, thus the whole process is executed once only. The complexity of implemented algorithms is linear; $\Theta(|V| + |E|)$, where $|V|$ means the total number of vertices (this number corresponds to the total number $|B|$ of all operational vertices in the initial flow-chart) and $|E|$ is the total number of edges between vertices.

The presented tool was prepared to aid the prototyping process of CMCUs and it was applied to perform all experiments. Next Chapter shows the detailed results of implementation of the control unit that was prepared with all 8 methods shown in the dissertation. The analysis of results of experiments is presented as well. Moreover, the detailed description of ATOMIC like intermediate formats and command-line parameters can be found in Appendix A.

Chapter 8

Results of experiments

This Chapter presents results of experiments that were made to check the effectiveness of proposed synthesis methods. Moreover, benefits of partial reconfiguration will be shown as well. The first section deals with results gained during implementation of prepared CMCUs in an FPGA. Achieved values are analysed in detail and finally concluded with an attempt to select the proper synthesis method depending on the initial flow-chart description. Experiments of the partial reconfiguration of CMCUs are shown in the second section. Finally, the last section concludes achieved results.

8.1 Results of experiments of investigations with prepared synthesis methods

This section presents results of experiments that were achieved during implementation of CMCUs designed according to the rules shown in Chapters 4 and 5. At the beginning, there is short introduction to the library of test modules (benchmarks) that were used for the verification of prepared synthesis methods. Further, the simulation of the functionality of the CMCU will be presented. The third subsection shows the main results of experiments made to check the effectiveness of proposed synthesis methods. Finally, all the achieved values are analysed and concluded.

8.1.1 The library of test modules

All synthesis methods presented in Chapters 4 and 5 were verified with over 100 test modules (benchmarks). Each test module was prepared in the text-format that contains description of the CMCU as the flow-chart (see Appendix A for more detail). The library of test modules was initially created at Doneck University (Ukraine). Now it is expanded in University of Zielona Góra. Most of prepared benchmarks describe hypothetical flow-chart however some of them contain the description of real-devices (for example the traffic light controller, systems with arithmetic operations, etc.).

8.1.2 Verification of prepared methods

Verification of the functionality of prepared CMCUs was performed with the software Simulator (here Active-HDL from Aldec and ModelSim from Mentor Graphics were used). The simulation was performed for each synthesis method. The verification of each module was similar. First, a Verilog code was generated for each synthesis method using ATOMIC (see Chapter 7). Next, the controller was simulated and its functionality was verified.

The verification process will be illustrated by an example. There is an exemplary flow-chart Γ_4 of the CMCU U_9 shown in the fig. 8.1. The flow-chart Γ_4 contains $|B|$=10 operational and $|X|$=2 conditional vertices. There are $|Y|$=5 microoperations that are generated by the controller. Let's design the U_9 as the CMCU with sharing codes. The set of operational linear chains contains $|C|$=3 OLCs: α_1=$\langle b_1, b_2, b_3\rangle$, α_2=$\langle b_4, \ldots, b_7\rangle$ and α_3=$\langle b_8, b_9, b_{10}\rangle$. Therefore, there are R_2=2 variables Q=$\{q_2$,$q_1\}$ required to keep the state of the controller. The longest OLC is α_2 and it contains M_1=4 operational blocks. It means that two variables T=$\{t_1$,$t_2\}$ will form the excitation function for the counter. Both codes generated by the counter A=$\{a_1$,$a_2\}$ and by the register Q=$\{q_1$,$q_2\}$ are encoded using two variables, thus the address of each microinstruction has the width equal to R_3=R_1+R_2=4. According to the (5.1), such address is formed as the concatenation of codes generated by the counter and by the register: $A(b_t)$=$K(\alpha_g)$*$K(b_t)$.

Figure 8.2 shows results of the software verification of the U_9 designed as the CMCU with sharing codes. Here, the microoperation y_0 increments the counter and it is a feed-back signal for the counter and for the register. The additional

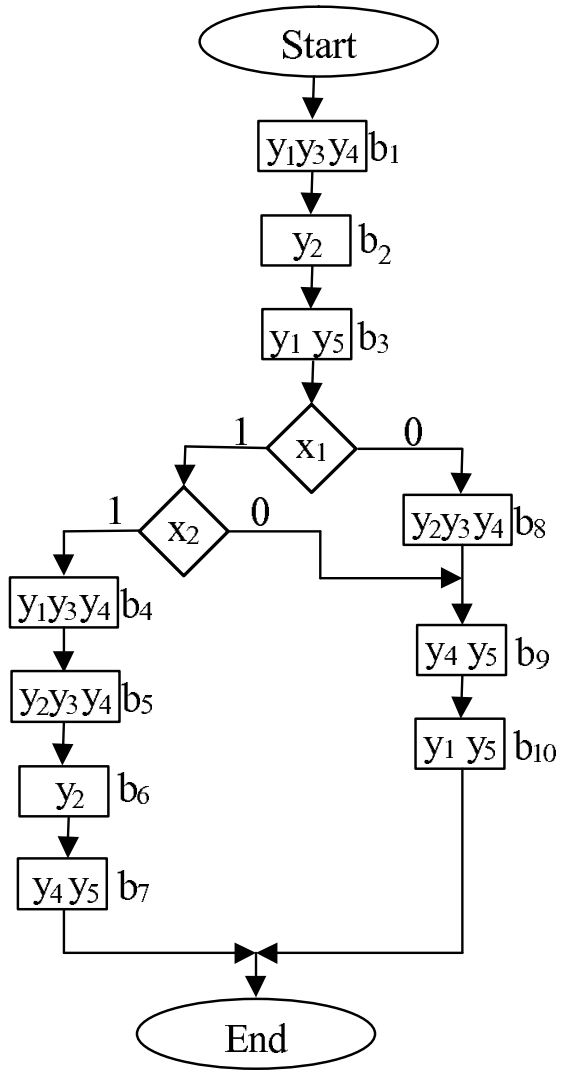

Figure 8.1: The flow-chart Γ_4

variable y_K is set to 1 if the final vertex of the flow-chart is reached.

The counter and the register are active on the positive edge of the clock signal. Dedicated memory blocks that were used for implementation of the control memory are synchronous and the clock signal ought to be connected as well. Because the variable y_0 drives the counter and the register, thus microinstructions are generated when the negative edge of the clock signal appears. Such a solution ensures the proper functionality of the CMCU.

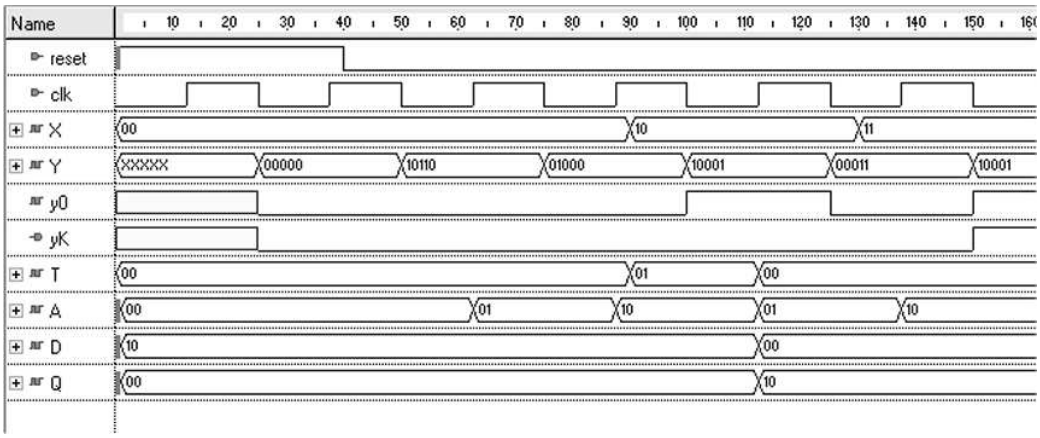

Figure 8.2: Results of the simulation of the CMCU U_9

8.1.3 Results of experiments

This section presents results that were achieved during the logic synthesis and implementation of CMCUs. All synthesis methods were verified with over 70 benchmarks. Additionally, there was an FSM model prepared for each test. The automaton was created according to the rules presented in (IEEE, 2001; Thomas and Moorby, 2002). It should be pointed out that all FSMs were prepared in such a way, that during implementation, all microoperations were realised with **dedicated memory blocks of the FPGA**.

The prototyping process for each benchmark was similar. Based on the flow-chart description (*.fc* file), the controller was structurally decomposed with all 8 synthesis methods presented in the dissertation. Additionally, there was an equivalent FSM produced. Achieved Verilog codes were finally synthesised and implemented with the Xilinx XST tool.

Table 8.1 presents average results of the CMCU implementation designed with the particular synthesis method in comparison to the FSM and to the traditional CMCU with mutual memory. To clarify the presentation of results, detailed values that were achieved during experiments are presented in Appendix B. As the destination, the FPGA XC2VP30 (Xilinx Virtex-II Pro family) was selected. The device contains 27392 Flip-Flops, 27392 LUTs (13696 Slices) and 136 dedicated memory blocks (Block-RAMs). The device was selected because of its structure (it can be partially reconfigured) and its availability at University of Zielona Góra.

Table 8.1: Average results of experiments

	FPGA resources	Designing method								
		FSM	MM	FD	OI	OD	SC	SD	CA	CD
Comparison to the FSM	Slices	100%	91%	73%	76%	60%	57%	51%	53%	**50%**
	FF	100%	100%	105%	102%	108%	120%	127%	122%	125%
	LUTs	100%	91%	71%	78%	60%	57%	50%	54%	**49%**
	BRAMs	100%	100%	136%	102%	126%	279%	320%	151%	186%
Comparison to the CMCU with mutual memory	Slices	110%	100%	82%	84%	68%	62%	**57%**	60%	**57%**
	FF	100%	100%	105%	102%	108%	120%	127%	122%	125%
	LUTs	110%	100%	81%	86%	68%	63%	**57%**	62%	**57%**
	BRAMs	100%	100%	136%	102%	126%	279%	320%	151%	186%

FSM - realisation of the controller as the FSM;

MM - realisation of the controller as the CMCU with mutual memory;

FD - realisation of the controller as the CMCU with function decoder;

OI - realisation of the controller as the CMCU with outputs identification;

OD - realisation of the controller as the CMCU with outputs identification and function decoder;

SC - realisation of the controller as the CMCU with sharing codes;

SD - realisation of the controller as the CMCU with sharing codes and function decoder;

CA - realisation of the controller as the CMCU with address converter;

CD - realisation of the controller as the CMCU with address converter and function decoder.

8.2 Analysis of results of experiments

The detailed analysis of results presented in tables 8.1 and B.1 (Appendix B) shows that **the number of logic blocks that are required for implementation of the controller in the FPGA is strongly tied with the number of microinstructions that are held in the control memory**.

From the tab. B.1 that is presented in Appendix B we can see that **in case of relatively small devices where the control memory may be implemented with one dedicated memory block, the realisation of the controller as the CMCU U_{SD} with sharing codes and function decoder gives the best results**. Firstly, it requires average the fewest number of logic blocks among all presented methods. Furthermore, the control memory is implemented with one dedicated memory block, thus there is no need for application of the address converter. Obviously application of the function decoder is optional - its usage reduces the number of logic blocks but increases the number of dedicated memories.

According to the (5.14), if the total number of bits generated by the register and counter exceeds the width of the microinstruction address, the CMCU U_{CD} with address converter and function decoder ought to be selected (see Chapter 5). It should be pointed out that results gained during realisation of the controller as the CMCU U_{SD} are similar to the values achieved for the CMCU U_{CD}. The number of required logic blocks for implementation of both controllers are almost the same. However, in case of control units that contain memories that ought to be decomposed (their volume exceeds the volume of one dedicated memory block), the **CMCU with address converter requires on average by 46% fewer dedicated memory blocks than the CMCU with sharing codes** (see benchmarks *Test031*, *Test036*, *TestAW02* and further presented in tab. B.1 in Appendix B). **These results prove the effectiveness of application of the address converter in case of CMCUs, where the address indicated by the counter and by the register is wider than the minimum number of bits needed for microinstructions addressing**.

The CMCU U_{SD} with address converter and function decoder consumes the fewest number of logic blocks of the destination FPGA in case of controllers where the control memory is decomposed (which means that more than one BRAM is used). Such a realisation requires only 49% LUTs in comparison to the FSM and

57% in comparison to the CMCU with mutual memory. It means that the proposed **synthesis method with address converter and function decoder reduces the number of logic blocks that are used for implementation of the controller over two times in comparison to the traditional automaton**. On the other hand, there are more dedicated memory blocks required for realisation of the control unit. The number of dedicated memory blocks increases on average by 86%, therefore the CMCU U_{CD} is the best solution for implementation of the controller in FPGAs that contain enough dedicated memory blocks.

Finally from the tab. B.1 we can see, that **among controllers that produce more than 150 microinstructions, the CMCU U_{OD} with outputs identification and function decoder gives the best results**. In this case, such a realisation on average requires the fewest dedicated memory blocks and usually the fewest logic blocks as well.

Concluding, it should be pointed out that performed experiments proved the effectiveness of proposed synthesis methods. The criteria of all experiments was to reduce the number of logic blocks that are required for the controller implementation. The detailed analysis of the results of experiments showed that selection of the proper synthesis method may be tied with the structure of the CMCU. There are three typical situations when the proper synthesis algorithm can be proposed:

- In case of relatively small systems (where the number of microinstructions does not exceed 150 and the control memory can be implemented with one dedicated memory block), the **CMCU with sharing codes and function decoder** seems to be the best solution. However, it should be pointed out that such a realisation consumes at least two dedicated memory blocks of the FPGA. Therefore, if a number of available dedicated memory blocks is limited, the method with **outputs identification** should be used.

- In case of controllers where the volume of the control memory exceeds the volume of one dedicated memory block and the total number of microinstructions is fewer than 150, the **CMCU with address converter and function decoder** gives the best results.

- In case of controllers where the total number of microinstructions exceeds 150, the **CMCU with outputs identification and function decoder** ought to be selected.

8.3 Results of experiments of partial reconfiguration of CMCUs implemented in the FPGA

This section presents analysis of results of partial reconfiguration of CMCUs implemented in the FPGA. Detailed values gained during experiments are shown in Appendix B.

The partial reconfiguration process of CMCUs was performed on the *XC2VP30* device. As it was already mentioned in Chapter 6, such FPGA contains 136 dedicated memory blocks organized in 8 columns. Each column can be configured with 64 frames independently (one frame configures one line (INIT) in all BRAMs that belong to the column).

Analysis of results of experiments showed that the way of realisation of the control memory of the CMCU in the FPGA is very important. Figure 8.3 presents three variants of implementation of a hypothetical CMCU where two microinstructions A and B are partially reconfigured. In the first mode, both microinstructions are implemented in the separate BRAMs that are placed in the same column. Both A and B are located in the line *INIT_ 00* of its BRAM. Therefore, during partial reconfiguration only one frame will be sent to the FPGA. Such a frame covers lines of both BRAMs, because they are situated in one column. In the second mode, both A and B are implemented in the same BRAM. However there are two lines required (A is initialized with *INIT_ 00* while B with *INIT_ 01*. It means that two frames are required for reconfiguration. In the third mode, A and B are implemented in two BRAMs. Now it is not important that both microinstructions are configured with the same line (*INIT_ 00*), because they are located in the different columns. Therefore, two frames are sent during reconfiguration.

Table 8.2 shows that the best results were achieved during implementation of the first variant of the controller. Despite the fact that two lines are modified, only one frame is sent to reconfigure the device and the original bit-stream was reduced over 500 times. Very interesting results were achieved during implementations of two remaining variants. Both versions required two frames for partial reconfiguration, however the size of partial bit-streams are different. In case of the second variant, where both microinstructions were located in the same BRAM, the bit-stream was reduced over 400 times. The worse gain was achieved during third mode, where A and B were realised with BRAMs located in the different columns.

Table 8.2: Results of three variants of reconfiguration of 2 microinstructions

Variant	Modified BRAMs	Modified lines	Modified columns	Modified frames	Size of partial bit-stream [bytes]	Reduction (% of original)	Reduction (times smaller)
1	2	2	1	1	2696	0,19%	527
2	1	2	1	2	3520	0,24%	417
3	2	2	2	2	4360	0,30%	333

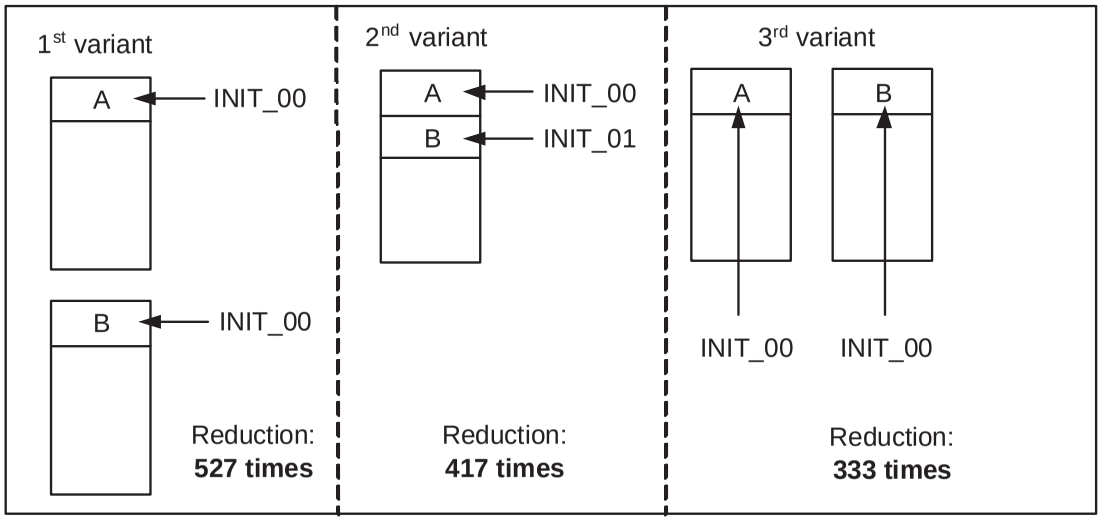

Figure 8.3: Three variants of reconfiguration of 2 microinstructions

The detailed analysis of performed experiments indicates that the reduction of the size of the original bit-stream strongly depends on the placement of the control memory in dedicated memory blocks of an FPGA. The best gain is reached in case of implementation of the control memory with BRAMs that are located in the same column. Partial reconfiguration of such a organization requires the least amount of configuration frames. Experiments showed that even replacement of the content of the control memory that was implemented with 13 BRAMs (organized in one column) **permits to reduce the original bit-stream over 50 times**. Furthermore, the worst results were achieved in case of implementation of the control memory with BRAMs located in separate columns. Partial reconfiguration of the control memory that was realised with 13 BRAMs placed in 8 different columns reduces the size of the bit-stream over 8 times.

Concluding, it should be pointed out that **partial reconfiguration of compositional microprogram control units implemented on the FPGA reduces the size of the original bit-stream even over 500 times**. In case of controllers, where the control memory ought to be decomposed into more than one BRAM, the best gain is reached during realisation of the memory with blocks located in the same column. The placement of each BRAM can be easily modified with tools delivered from Xilinx, which additionally checks routings and timing paths. Detailed values that were gained during partial reconfiguration of CMCUs implemented on the FPGA are presented in Appendix B.

Chapter 9

Conclusions

The development of microelectronics benefited in appearance of the system-on-a-programmable-chip that can be used for implementation of complex digital systems. The main part of SoPCs is an FPGA. Such a device contains logic blocks for implementation of the combinational logic and dedicated memory blocks that offer additional area for data storage. Therefore, traditional methods of the design prototyping evaluate. The aim of such modifications is the reduction of the number of logic blocks of the FPGA. This task is very often solved by application of the design decomposition.

One of main blocks of the digital system is a control unit. It can be designed as the compositional microprogram control unit, where the controller is decomposed into two main parts. The first one is in charge of the proper address formation of microinstructions that are kept in the control memory. The main advantages of such realisation is a possibility of implementation of the controller using logic elements and dedicated memory blocks offered by FPGAs. Moreover, thanks to its structure, part of the CMCU that is already implemented in the FPGA can be easily reconfigured.

The structural synthesis of compositional microprogram control units was the main scope of the dissertation. Six new designing methods of the CMCU were proposed. The aim of all methods is to reduce the number of logic blocks that are required for implementation of the controller in the FPGA. Prepared algorithms were divided into two groups. The first one deals with the CMCU with mutual memory, where the microinstructions address is used for recognition of internal

states of the controller. The second group is oriented on the formation of the microinstruction's address by codes generated by the register and by the counter. Additionally, the CMCU with address converter permits to decrease the volume of the control memory to the required minimum.

The effectiveness of presented methods was proved thanks to the prepared design aided system. ATOMIC performs the automatic structural decomposition of CMCUs. The output code (generated in the Verilog-HDL language) is ready for further logic synthesis and implementation. Modular structure of the designed system permits to modify the structure of the CMCU at any level of the prototyping process.

The second task of the dissertation was partial reconfiguration of CMCUs implemented in the FPGA. The designer can replace only the control memory content of the controller, that already resides in the programmable device. Therefore, there is no need to repeat the whole CMCU prototyping process for each version of the control unit. Full designing process is done only once. For further versions of the CMCU, only the reduced prototyping flow ought to be performed. Therefore, partial reconfiguration reduces the size of data that are sent to the FPGA. Additionally, the configuration time is shorter, what in consequence benefits in reduction of the risk of errors that may occur during the FPGA configuration.

The most important innovations introduced in the dissertation are:

- **preparation of new synthesis methods of CMCUs oriented on the reduction of the number of logic blocks that are required for implementing the controller in the FPGA**,
- **preparation of new synthesis methods of CMCUs oriented on the reduction of the number of dedicated memory blocks that are required for implementing the controller in an FPGA**,
- **designing of the dedicated tool that aids the prototyping process of CMCUs implemented in the FPGA**,
- **preparation of the new CMCU prototyping flow, based on the partial reconfiguration of controllers implemented in the FPGA**,
- **verification of the effectiveness of prepared methods by adequate experiments (implementation of benchmarks in the FPGA)**.

Performed experiments proved the effectiveness of proposed synthesis methods. Criteria of all researches were to reduce the number of logic blocks that are required for the controller implementation. The detailed analysis of the results of experiments showed that the proper synthesis method may be chosen at the specification stage.

There are three main future directions of the presented work. The first one is an attempt of conjunction of both CMCUs decomposition methods. Excitation functions of internal blocks of CMCUs that are formed during the structural decomposition are further decomposed by commercial synthesis tools. Therefore, there is an idea to apply the external functional decomposition. This task is successfully developed by academic organizations for both, CPLDs (Kania, 2004; Kania, 1999; Devadas et al., 1988) and FPGAs (Selvaraj et al., 2005; Jóźwiak and Chojnacki, 2003; Rawski et al., 2003). Additionally, the functional decomposition can be applied in reduction of the volume of the controller memory. This idea is already been developed and was outlined in (Wiśniewska and Wiśniewski, 2005).

The second aim of future work is improvement of the encoding of internal states in case of CMCUs based on sharing codes. Such controllers contain simplified automaton for microinstructions addressing which can be optimized with algorithms like NOVA or JEDI (Sentovich et al., 1992*b*).

Proposed synthesis methods of the CMCU can be expanded to improve the safety of the controller (Halang and Krämer, 1994; Adamski, 1999; Halang and Krämer, 1992; Halang and Adamski, 1997; Adamski et al., 2007). There are ideas to implement the combinational circuit with dedicated memory blocks of the FPGA. Additionally, such a realisation allows partial reconfiguration of the combinational circuit. Therefore, it means that the functionality of the controller can be easily modified. Presented idea has been already developed as a common project with prof. W.A.Halang (FernUniversität of Hagen).

The presented work benefited publications in journals (10 international and 3 domestic). Moreover, conducted investigations were presented at various workshops and conferences (16 international and 5 domestic). Totally, there were 34 articles published that directly refer to the dissertation. Presented solutions were honoured by four awards (two submitted at OWD and KNWS conferences, and two Rector's group awards). The work was realised as a part of the Integrated Regional Operational Programme and as a part of *KBN* grant no 3 T11C 046 26.

Appendix A

Description of ATOMIC

Consecutive sections present the detailed description of ATOMIC. Firstly, the structure of ATOMIC is presented. In the second section, formats of input and output data are shown. Finally, the last section describes ATOMIC command-parameters and switches. The overview of the tool is shown in Chapter 7.

A.1 The structure of ATOMIC

Figure A.1 presents the idea of prototyping flow of CMCUs. Such a flow is performed by ATOMIC's modules.

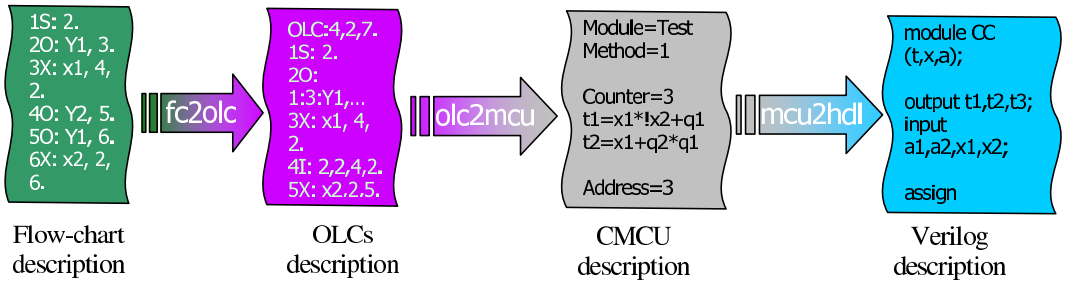

Figure A.1: The detailed structure of ATOMIC

Based on the flow description the module *fc2olc* generates the *.olc* file. Such a file contains the description of operational linear chains. The operation is performed in 5 main steps:

1. Reading and analysis of input data.
2. Searching for OLCs.
3. Searching for OLCs inputs.
4. Generating output data.
5. Writing the output file.

Next the main structural decomposition is performed by the *olc2mcu* unit. Here one of the eight of the CMCU prototyping methods is used (command-line switches are described in section A.3). The structural synthesis is made in 5 main steps:

1. Reading and analysis of input data.
2. Encoding OLCs and their components (with the selected method).
3. Expanding the memory content (conversion to the binary format).
4. Creating excitation functions.
5. Writing the output file.

The structural synthesis performed by ATOMIC was improved, thus the transition table of the CMCU is not formed. Due to the structure of ATOMIC, excitation functions are generated directly from the graph that represents the OLCs flow-chart. Therefore, the synthesis process is faster and additionally less computational memory is used (there is no need to represent the transition table in the memory).

Finally, the description of the CMCU is converted to the Verilog language with the *mcu2verilog* module. This process is performed in only two stages. The input file is read, and then it is converted to the Verilog-HDL format. Finally, the CMCU

is written as the Verilog source code and it is ready for further logic synthesis and implementation.

It should be pointed out that each operation of execution of each presented modules is logged into a log-file. Such a log-file is additionally written to the screen (however, it can be turned off, see section A.3).

The next section presents the detailed input and output data format of each ATOMIC module.

A.2 Input and output data formats of ATOMIC

This section describes all data formats that are used by ATOMIC modules. At the beginning the input format of ATOMIC is shown. It is also the input for the module *fc2olc*. The second subsection presents all intermediate formats that are exchanged by ATOMIC modules. Finally, the last subsection deals with the output data of ATOMIC that is generated by the *mcu2verilog* module.

A.2.1 The input data format of ATOMIC

The input for ATOMIC is specified as a text-file that contains the description of the flow-chart (*.fc*) file. Such a description was initially proposed in (Baranov, 1994). The input file is divided into two sections: the flow-chart description and microinstructions definition.

The first part contains the description of the flow-chart structure. Figure A.2 shows the graphic and text version of the CMCU U_9. Each line corresponds to one block of the flow-chart. The line must begin with the number of the vertex. Next the symbol of the vertex appears, where:

- **S** - start - initial vertex of the flow-chart. After this symbol there is a number of next vertex of the flow-chart.
- **O** - operational vertex. After the symbol, a name of pseudo-microinstruction is declared (i.e. Y_1). Such a pseudo-microinstruction is defined in the second part of the file. Then, there is a number of the next block where the transition should be performed.

- **X** - conditional vertex of the flow-chart. After the symbol there is a definition of the block name (x_1, x_2, etc.). Next numbers of destination vertices appear - first number means the vertex where the transition should be executed in case if the condition is true while the second one shows the destination in case of being false.
- **E** - end - final vertex of the flow-chart.

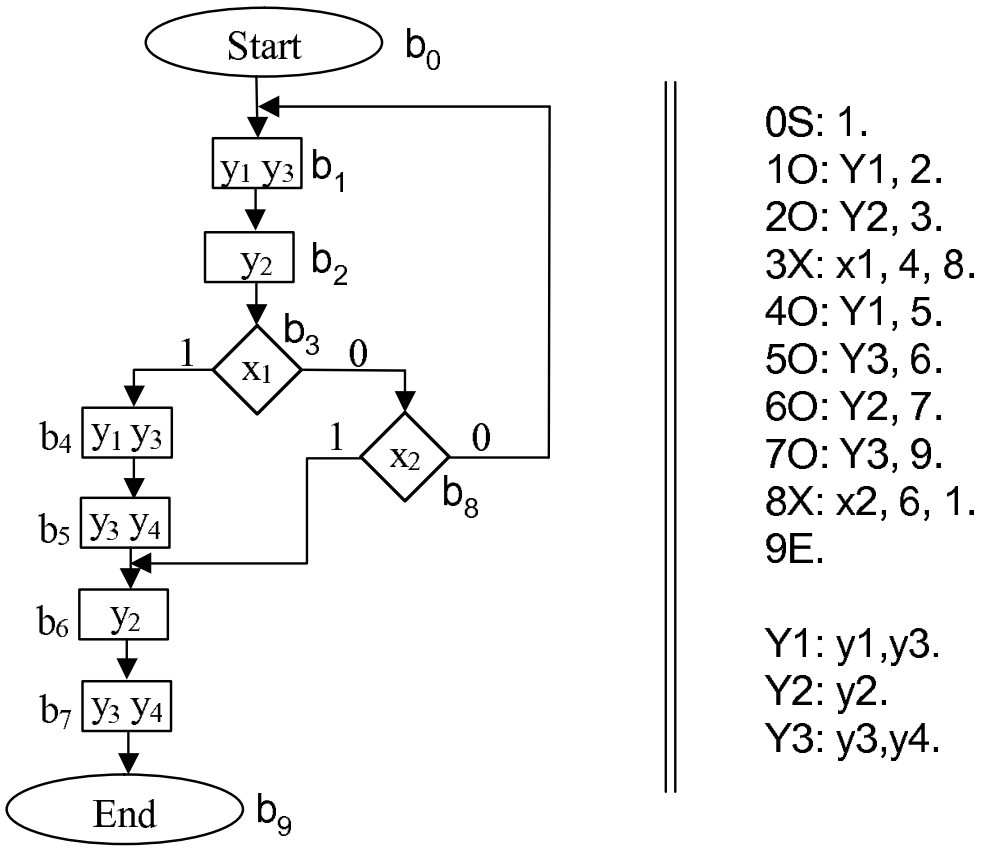

Figure A.2: The graphic and text description of the exemplary CMCU U_9

The second part of the file contains the definition of pseudo-microinstructions that were already declared by operational vertices of the flow-chart. There is a name of pseudo-microinstruction that is followed by the list of microoperations that are executed in such a pseudo-microinstruction.

The description of microinstructions is compacted, thus two or more different vertices may define the same pseudo-microinstruction. Of course such compacted information is expanded in further steps executed by ATOMIC and finally each vertex of the flow-chart will execute one microinstruction. For example, in the CMCU U_9 there are two pseudo-microctions named Y_1 and Y_2. Here Y_1 consists of microoperations y_1 and y_3 that are executed in vertices b_1 and b_5 while Y_2 includes y_2 that is executed in the vertex b_4.

A.2.2 The intermediate data formats of ATOMIC

ATOMIC consists of three modules thus each module has its own data format. The description of the flow-chart presented in the previous subsection is either input for ATOMIC and for the module *fc2olc*. Such a module generates intermediate data format (*.olc* file) that contains the description of the set of operational linear chains. Of course, additionally information about the content of the control memory is specified. The structure of the *.olc* file is similar to the *.fc* structure. The set of operational linear chains is also described as a flow-char, however the meaning of the symbols are now the following:

- **S** - start - initial vertex of the OLC flow-chart. After this symbol there is a number of next vertex of the OLC flow-chart.
- **O** - main input and description of the OLC. After the symbol there are four numbers specified separated by commas, consecutive: the number g of the OLC $\alpha_g \in C$, the number t of the input I_g^t in the OLC $\alpha_g \in C$, the position of vertex b_i in the OLC $\alpha_g \in C$, the number of next vertex b_j in the OLC flow-chart. Next, besides colons, there is a number of microinstructions that are executed inside the chain (sum of all operational vertices inside the OLC $\alpha_g \in C$). Finally there are pseudo-microinstructions specified, separated by commas. The meaning of pseudo-microinstructions is the same as it was explained during the *.olc* file description.
- **I** - input of the OLC (appears only if the OLC has more than one input). After the symbol there are four numbers specified, consecutive: number of the OLC, number of the input in the OLC, position in the OLC, number of the next vertex in the OLC flow-chart.

- **X** - conditional vertex of the OLC flow-chart. After the symbol there is a definition of the name of the block (x_1, x_2, etc.). Next, numbers of destination vertices appear - first number means vertex, where the transition should be executed in case if the condition is true while the second one shows the destination in case of being false.
- **E** - end - final vertex of the OLC flow-chart.

The exemplary *.olc* file for the CMCU U_9 is presented in the A.3.

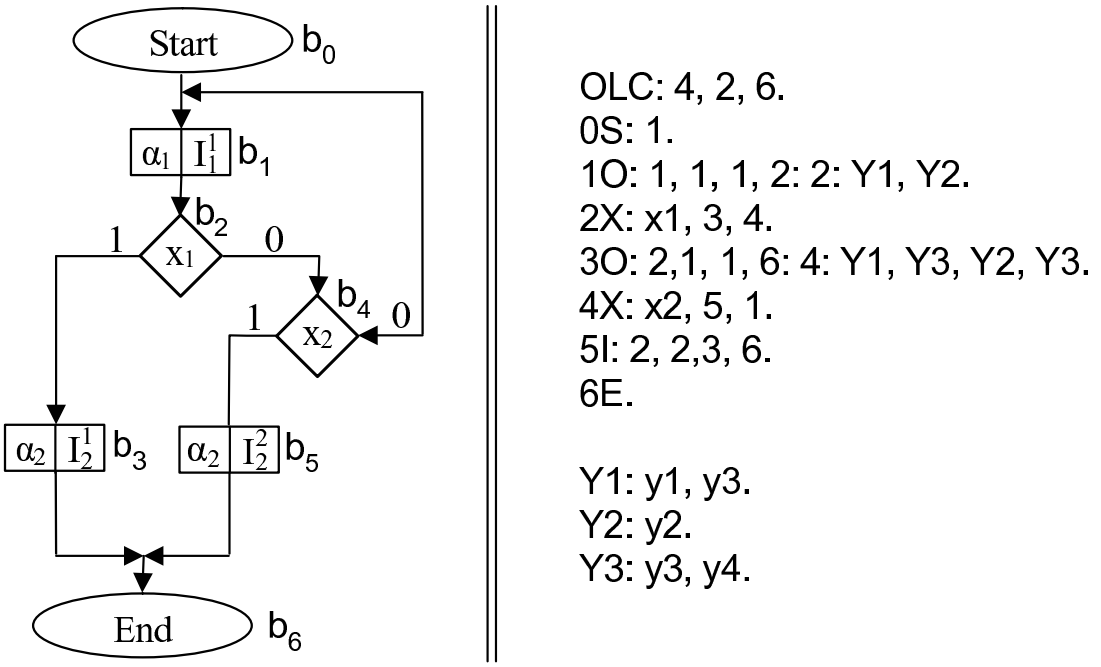

Figure A.3: The OLCs description of the CMCU U_9

The *.olc* file contains all necessary data for the main synthesis of the CMCU. The file is read by the module **olc2mcu** which produces the description of the CMCU (*.mcu* file) by one (of eight) selected synthesis method. There are three main sections specified in the *.mcu* file: *Module name and parameters*, *Excitation functions*, *Control memory content*. The first section contains information about the CMCU (its name, number of inputs, outputs, etc). The second section defines excitation functions for the counter, register and/or function decoder. The last section contains the description of the control memory (specified as a table of

microoperations). In case of applying the circuit of function decoder or address converter, additional description of such a block is written as well. The exemplary *.mcu* file content for the CMCU U_9 realised as a controller with mutual memory is presented in the listing A.1.

Listing A.1: The MCU description of the CMCU U_9

```
//Module name and parameters:
Module=CMCU_U9_mm
Method=1
Inputs=2
Outputs=6
Microinstructions=6
InputNames=x1,x2
OutputNames=y0,y1,y2,y3,y4,yK

//Excitation functions:
Counter=3
t1=0
t2=a1*!a2*!a3*x1
t3=a1*!a2*!a3*!x1*x2

//Control Memory content:
Address=3
O0=010100
O1=101000
O2=010100
O3=000110
O4=001000
O5=100111
DefaultMemoryValue=000000
```

Finally, the code in Verilog-HDL is generated. Such a code is the output file produced by ATOMIC and it is ready for the further logic synthesis and implementation. The detailed description of the Verilog code generated by ATOMIC is shown in the next subsection.

A.2.3 The output data format of ATOMIC

ATOMIC produces the code in the Verilog-HDL as the output. Depending on the selected method, the output file is different. Listing A.2 presents the exemplary Verilog code of the CMCU U_9 generated for the method with mutual memory.

Listing A.2: The Verilog code for the flow-chart Γ_9

```
//-------------------------------------------------------
//Description of CMCU
//-------------------------------------------------------
module CMCU_U9_mm (y1,y2,y3,y4,yK,clk,reset,x1,x2);
output y1,y2,y3,y4,yK;
input clk;
input reset;
input x1,x2;

        wire [3:1] t; //excitation function for counter
        wire [3:1] a; //address generated by counter
        wire y0;

        CC combinational (t,x1,x2,a);
        CT counter (a,clk,t,reset,y0);
        CM memory ({y0,y1,y2,y3,y4,yK},~clk,~yK,reset,a);
endmodule

//-------------------------------------------------------
//Description of COMBINATIONAL CIRCUIT
//-------------------------------------------------------
module CC (t,x1,x2,a);
output [3:1] t;
input [3:1] a;
input x1,x2;

        assign t[1]=0;
        assign t[2]=a[1]&~a[2]&~a[3]&x1;
        assign t[3]=a[1]&~a[2]&~a[3]&~x1&x2;
endmodule
```

```
//-----------------------------------------------------
//Description of COUNTER
//-----------------------------------------------------
module CT (q,clk,data,reset,load);
output reg [3:1] q;
input reset, load, clk;
input [3:1] data;

always @(posedge reset or posedge clk)
begin
        if (reset == 1'b1) q = {3{1'b0}};
        else if (load == 1'b1) q = data;
        else q=q+1;
end
endmodule

//-----------------------------------------------------
//Description of CONTROL MEMORY
//-----------------------------------------------------
module CM (y, clk, oe, reset, address);
output reg [6:1] y;
input  [3:1] address;
input clk, oe, reset;

// synthesis attribute bram_map of CM is yes
always @(posedge clk)
begin
        if (reset) y=0;
        else if (oe)
        case (address)
                0:y=6'b010100;
                1:y=6'b101000;
                2:y=6'b010100;
                3:y=6'b000110;
                4:y=6'b001000;
                5:y=6'b100111;
```

```
                    6 : y=0;
                    7 : y=0;
                    default : y=6'b000000 ;
            endcase
end
endmodule
```

A.3 Arguments of ATOMIC modules

This section shows the usage of ATOMIC. All modules are independent of the platform and system. Therefore, ATOMIC tools operate under the command-line interface. Each module ought to be run with at least one argument where the name of the input file is specified. Furthermore other arguments separated by the space character may be specified. Consecutive subsections presents available arguments for each ATOMIC module.

A.3.1 The usage of the fc2olc module

The module is executed by the following command:

$$\textbf{\textit{fc2olc inputfile.fc}}\ -o\ outputfile\ -q$$

The meaning of arguments:

- *inputfile.fc* is the input file name that contains the flow-chart description;
- *-o outputfile* (optional) is the name of the output file, where results are written (the default output name is the same as *inputname* with the *.olc* extension);
- *-q* (optional) runs in the quiet mode (the log is not displayed on the screen).

Example of the execution: ***fc2olc test.fc -q***

A.3.2 The usage of the olc2mcu module

The module is executed by the following command:

$$\textbf{\textit{olc2mcu inputfile.olc}} \ -m[1-8] - o\ outputfile\ -q$$

The meaning of arguments:

- *inputfile.olc* is the name of the input file that contains the OLCs description;
- *-m[No]*, where No is a number of the synthesis method $(1 \ldots 8)$:
  1. The CMCU with mutual memory.
  2. The CMCU with function decoder.
  3. The CMCU with outputs identification.
  4. The CMCU with outputs identification and function decoder.
  5. The CMCU with sharing codes.
  6. The CMCU with sharing codes and function decoder.
  7. The CMCU with address converter.
  8. The CMCU with address converter and function decoder.
- *-o outputfile* (optional) is the name of the output file, where results are written (the default output name is the same as *inputname* with the *.mcu* extension;
- *-q* (optional) runs in the quiet mode (the log is not displayed on the screen).

Example of the execution: ***olc2mcu test.olc -m7 -o cmcu_conv.mcu***

A.3.3 The usage of the mcu2verilog module

The module is executed by the following command:

$$\textbf{\textit{mcu2verilog inputfile.mcu}} \ -o \ outputfile \ -q$$

The meaning of arguments:

- *inputfile.olc* is the name of the input file that contains the description of the CMCU in the *.mcu* format;
- *-o outputfile* (optional) is the name of the output file, where results are written (the default output name is the same as *inputname* with the *.v* extension;
- *-q* (optional) runs in the quiet mode (the log is not displayed on the screen).

Example of the execution: ***mcu2verilog cmcu_ conv.mcu -o cmcu.v***

Appendix B

Detailed results of experiments

Detailed results of experiments are presented in this Appendix. First section analyses the effectiveness of prepared synthesis methods, while the second one shows results of experiments performed with partial reconfiguration of CMCUs.

B.1 Detailed results of experiments of effectiveness of prepared synthesis methods

Table B.1 shows results of implementation. To clarify the presentation of achieved results there were only most important benchmarks selected. It should be pointed out that the detailed analysis and calculation of average results of the whole collection of benchmarks are described in Chapter 8.

The table contains the following columns:

- *Benchmark* - the name of the benchmark;
- X - the number of inputs (logic conditions) of the controller;
- Y - the number of outputs (microoperations) of the controller;
- M_1 - the length of the longest OLC (such OLC that contains most operational vertices);
- M_2 - the total number of OLCs;
- M_3 - the number of microinstructions (total number of operational vertices);

- *FPGA resources* - the name of the presented FPGA resource (Slice, Flip-Flop, Look-Up Table or dedicated Block-RAM);
- *FSM* - results achieved for implementation of the controller as the FSM;
- *MM* - results achieved for implementation of the controller as the CMCU with mutual memory;
- *FD* - results achieved for implementation of the controller as the CMCU with function decoder;
- *OI* - results achieved for implementation of the controller as the CMCU with outputs identification;
- *OD* - results achieved for implementation of the controller as the CMCU with outputs identification and function decoder;
- *SC* - results achieved for implementation of the controller as the CMCU with sharing codes;
- *SD* - results achieved for implementation of the controller as the CMCU with sharing codes and function decoder;
- *CA* - results achieved for implementation of the controller as the CMCU with address converter;
- *CD* - results achieved for implementation of the controller as the CMCU with address converter and function decoder.

The method that gives the best results (the CMCU that requires the fewest LUT elements of an FPGA) is marked by gray color. Additionally, in case of benchmarks where an application of address converter reduces the number of required dedicated memory blocks in comparison to traditional sharing codes, the number of BRAMs used by CMCUs U_{SC} and U_{SD} are marked in red.

Table B.1: Results of implementation - Part 1

Benchmark	X	Y	M_1	M_2	M_3	FPGA resources	FSM	MM	FD	OI	OD	SC	SD	CA	CD
MK_01	10	9	15	15	85	Slices	45	48	33	26	22	19	20	19	20
						FF	7	7	7	7	7	8	8	8	8
						LUTs	80	83	59	50	40	34	35	34	35
						BRAMs	1	1	2	1	2	1	2	2	3
MK_02	10	9	10	15	89	Slices	52	47	30	31	24	19	19	19	19
						FF	7	7	7	7	7	8	8	8	8
						LUTs	91	85	53	54	43	35	34	35	34
						BRAMs	1	1	2	1	2	1	2	2	3
MK_03	7	10	8	12	48	Slices	29	24	19	20	17	13	14	13	14
						FF	6	6	6	6	6	7	7	7	7
						LUTs	48	41	31	36	30	22	26	22	26
						BRAMs	1	1	2	1	2	1	2	2	3
MK_04	9	9	10	13	56	Slices	25	27	28	26	15	17	14	17	14
						FF	6	6	6	6	6	8	8	8	8
						LUTs	45	45	48	47	25	29	24	29	24
						BRAMs	1	1	2	1	2	1	2	2	3
MK_06	8	13	7	14	58	Slices	32	31	21	20	16	14	13	14	13
						FF	6	6	6	6	6	7	7	7	7
						LUTs	55	53	37	36	29	26	23	26	23
						BRAMs	1	1	2	1	2	1	2	2	3

Table B.2: Results of implementation - Part 2

Benchmark	X	Y	M_1	M_2	M_3	FPGA resources	FSM	MM	FD	OI	OD	SC	SD	CA	CD
MK_07	11	10	8	19	70	Slices	30	52	34	46	31	22	19	22	19
						FF	7	7	7	7	7	8	8	8	8
						LUTs	46	90	57	83	55	41	35	41	35
						BRAMs	1	1	2	1	2	1	2	2	3
MK_08	7	9	7	12	49	Slices	22	25	19	22	16	11	11	11	11
						FF	6	6	6	6	6	7	7	7	7
						LUTs	39	44	34	41	28	21	21	21	21
						BRAMs	1	1	2	1	2	1	2	2	3
MK_09	7	9	7	17	56	Slices	34	33	28	28	23	18	17	18	17
						FF	6	6	6	6	6	8	8	8	8
						LUTs	63	58	48	50	39	33	33	33	33
						BRAMs	1	1	2	1	2	1	2	2	3
MK_10	13	12	7	21	92	Slices	69	55	44	48	30	28	22	28	22
						FF	7	7	7	7	7	8	8	8	8
						LUTs	126	94	75	84	54	50	39	50	39
						BRAMs	1	1	2	1	2	1	2	2	3
MK_11	8	9	6	14	48	Slices	32	24	21	24	19	14	17	14	17
						FF	6	6	6	6	6	7	7	7	7
						LUTs	49	43	36	45	32	27	31	27	31
						BRAMs	1	1	2	1	2	1	2	2	3

Table B.3: Results of implementation - Part 3

Benchmark	X	Y	M_1	M_2	M_3	FPGA resources	FSM	MM	FD	OI	OD	SC	SD	CA	CD
MK_12	11	10	6	23	71	Slices	55	56	38	36	27	24	20	24	20
						FF	7	7	7	7	7	8	8	8	8
						LUTs	100	96	67	64	49	43	35	43	35
						BRAMs	1	1	2	1	2	1	2	2	3
MK_14	8	9	5	17	53	Slices	26	33	27	31	26	18	17	18	17
						FF	6	6	6	6	6	8	8	8	8
						LUTs	47	59	46	54	47	34	28	34	28
						BRAMs	1	1	2	1	2	1	2	2	3
MK_15	8	12	8	18	68	Slices	37	41	29	25	22	18	15	18	15
						FF	7	7	7	7	7	8	8	8	8
						LUTs	49	71	50	48	40	31	28	31	28
						BRAMs	1	1	2	1	2	1	2	2	3
MK_17	9	9	7	19	72	Slices	43	50	34	30	27	19	17	19	17
						FF	7	7	7	7	7	8	8	8	8
						LUTs	81	88	57	54	48	32	31	32	31
						BRAMs	1	1	2	1	2	1	2	2	3
MK_19	6	9	6	13	46	Slices	21	28	19	21	13	10	10	10	10
						FF	6	6	6	6	6	7	7	7	7
						LUTs	38	49	33	38	24	18	18	18	18
						BRAMs	1	1	2	1	2	1	2	2	3

Table B.4: Results of implementation - Part 4

Benchmark	X	Y	M_1	M_2	M_3	FPGA resources	FSM	MM	FD	OI	OD	SC	SD	CA	CD
Test014	2	7	14	3	30	Slices	10	10	8	5	6	4	5	4	5
						FF	5	5	5	5	5	6	6	6	6
						LUTs	21	20	15	12	11	10	9	10	9
						BRAMs	1	1	2	1	2	1	2	2	3
Test017	2	6	4	3	11	Slices	4	6	6	4	5	4	4	4	4
						FF	4	4	4	4	4	4	4	4	4
						LUTs	10	12	10	9	9	8	7	8	7
						BRAMs	1	1	2	1	2	1	2	2	3
Test024	6	17	3	5	9	Slices	10	9	7	9	7	6	5	6	5
						FF	4	4	4	4	4	5	5	5	5
						LUTs	20	18	14	18	14	13	12	13	12
						BRAMs	1	1	2	1	2	1	2	2	3
Test025	65	18	14	54	153	Slices	366	161	131	174	128	130	122	130	122
						FF	8	8	8	8	8	10	10	10	10
						LUTs	642	283	229	309	224	224	211	224	211
						BRAMs	1	1	2	1	2	1	2	2	3
Test027	35	152	16	29	100	Slices	206	99	79	103	82	67	66	66	66
						FF	7	7	7	7	7	9	9	9	9
						LUTs	361	176	138	181	142	123	117	121	117
						BRAMs	5	5	6	5	6	5	6	6	7

Table B.5: Results of implementation - Part 5

Benchmark	X	Y	M_1	M_2	M_3	FPGA resources	FSM	MM	FD	OI	OD	SC	SD	CA	CD
Test031	52	222	11	51	151	Slices	276	120	111	134	95	99	106	99	105
						FF	8	8	8	8	8	10	10	10	10
						LUTs	482	208	199	240	172	173	187	173	186
						BRAMs	7	7	8	8	9	13	14	8	9
Test032	8	12	4	2	7	Slices	5	6	5	6	5	4	5	4	5
						FF	3	3	5	3	5	3	4	3	4
						LUTs	11	13	11	13	11	9	10	9	10
						BRAMs	1	1	2	1	2	1	2	2	3
Test033	52	152	11	51	151	Slices	279	119	112	134	95	99	106	98	105
						FF	8	8	8	8	8	10	10	10	10
						LUTs	486	207	200	240	172	173	187	172	186
						BRAMs	5	5	6	5	6	9	10	6	7
Test036	52	452	11	51	151	Slices	279	119	112	134	95	98	105	98	105
						FF	8	8	8	8	8	10	10	10	10
						LUTs	486	207	200	240	172	173	190	172	186
						BRAMs	13	13	14	14	15	27	26	14	15
TestAW01	11	42	36	17	154	Slices	63	33	26	27	19	21	22	21	22
						FF	8	8	8	8	8	11	11	11	11
						LUTs	114	58	49	51	38	41	39	41	39
						BRAMs	2	2	3	2	3	5	6	3	4

Table B.6: Results of implementation - Part 6

Benchmark	X	Y	M_1	M_2	M_3	FPGA resources	FSM	MM	FD	OI	OD	SC	SD	CA	CD
TestAW02	11	42	68	17	188	Slices	67	31	29	28	22	27	27	27	27
						FF	8	8	8	8	8	12	12	12	12
						LUTs	119	57	55	51	42	47	49	47	49
						BRAMs	2	2	3	2	3	11	12	4	5
TestAW03	11	42	36	19	188	Slices	60	43	35	30	28	29	26	29	26
						FF	8	8	8	8	8	11	11	11	11
						LUTs	108	76	62	56	52	53	48	53	48
						BRAMs	2	2	3	2	3	5	6	3	4
TestAW05	12	44	70	17	188	Slices	70	39	29	27	21	28	24	28	24
						FF	8	8	8	8	8	12	12	12	12
						LUTs	126	68	56	51	40	49	43	49	43
						BRAMs	2	2	3	2	3	11	12	4	5
TestAW06	14	44	70	17	188	Slices	77	37	30	36	24	31	23	31	23
						FF	8	8	8	8	8	12	12	12	12
						LUTs	141	67	56	67	47	55	42	55	42
						BRAMs	2	2	3	2	3	11	12	4	5
TestAW07	14	44	70	18	207	Slices	75	40	31	37	25	31	24	31	24
						FF	8	8	8	8	8	12	12	12	12
						LUTs	134	71	57	69	50	56	45	56	45
						BRAMs	2	2	3	2	3	11	12	4	5

B.2 Detailed results of partial reconfiguration of CMCUs implemented on an FPGA

This section presents results gained during partial reconfiguration of CMCUs. To verify the effectiveness of partial reconfiguration of controllers implemented in the FPGA, there were four CMCUs selected:

- *Test002* that contains control memory implemented with one BRAM,
- *Lights* that contains control memory implemented with 4 BRAMs,
- *Test027* that contains control memory implemented with 5 BRAMs.
- *Test036* that contains control memory implemented with 13 BRAMs.

All benchmarks except *Test002* were checked in three different modes. In the first mode, all BRAMs were organized according to the automatic placement and routing executed by the Xilinx implementation tool. In this case BRAMs are usually mixed - some of them are placed in the same column, some of them are located in different columns.

In the second mode, all BRAMs were placed in the one column. Such an operation was performed with Xilinx *FPGA Editor* that additionally checks routing and timing paths. According to the configuration rules presented in Chapter 6, the content of BRAMs located in the same column is modified with the same configuration frames. It means, that partial reconfiguration of such implemented memories should require the fewest frames, and therefore the size of the partial bit-stream should be reduced to the minimum.

The third mode organizes all BRAMs in different columns (except *Test036* which requires 13 BRAMs, which exceeds the total number of 8 columns, thus some memory blocks were located in the same column). It is expected, that partial reconfiguration of CMCUs realized in this mode should give the worse gain. The control memory is implemented with BRAMs located in separate columns and different frames are required for partial reconfiguration. Therefore, the reduction of the bit-stream should be the worst of all presented modes.

Table B.2 presents results of experiments of partial reconfiguration of CMCUs implemented in the FPGA. There are three main columns in the table:

- *Benchmark* - contains information about the Benchmark (name, the number of microinstructions, the number of microoperations, the number of BRAMs that are required for implementation and the size of the full bit-stream).
- *Modification* - contains information about modifications that were done to the Benchmark (the number of modified microinstructions, the number of BRAMs where microinstructions were modified, the total number of modified lines (INITs), the number of columns that contain modified BRAMs).
- *Results* - contains information about achieved results (the number of different frames, the size of the partial bit-stream, the percentage of the comparison between full and reduced bit-streams and achieved reduction - the number of times that original bit-stream was reduced).

As it was expected the best results were achieved in the second mode, where all BRAMs were placed in the same column. Even in case of *Test036* which requires 13 BRAMs, swapping the memory content is performed using only 32 frames and **size of the partial bit-stream is over 50 times smaller** compared to the bit-stream containing full FPGA configuration data.

The worst results were achieved in the third mode, where BRAMs were located in the different columns. In case of Benchmark *Test036*, 256 frames have to be sent to the FPGA to replace the whole memory content. However, it should be pointed out that the **bit-stream is still reduced over 8 times**.

Table B.7: Results of partial reconfiguration of CMCUs - Part 1

Benchmark					Modification				Results		
Name	Micro-instr.	Micro-oper.	BRAMs	Full bit stream	Modified microinstr.	Modified BRAMs	Modified lines	Modified columns	Different frames	Partial bit stream	% of original	Reduction
Test002 (default)	11	13	1	1448816	1	1	1	1	1	2696	0,19%	537,39
					2	1	1	1	1	2696	0,19%	537,39
					2	1	2	1	2	3520	0,24%	411,60
					4	1	2	1	2	3520	0,24%	411,60
					8	1	2	1	2	3520	0,24%	411,60
					11	1	1	1	1	2696	0,19%	537,39
					11	1	2	1	2	3523	0,24%	411,24
Lights (default) all BRAMs in 2 columns	53	32	4	1448816	1	1	1	1	1	2696	0,19%	537,39
					1	2	2	1	1	2696	0,19%	537,39
					1	3	3	2	2	4360	0,30%	332,30
					1	4	4	2	2	4360	0,30%	332,30
					13	1	2	1	2	3520	0,24%	411,60
					26	2	4	1	2	3520	0,24%	411,60
					39	3	6	2	4	6008	0,41%	241,15
					53	4	8	2	4	6008	0,41%	241,15
Lights (min) all BRAMs in 1 column	53	32	4	1448816	1	1	1	1	1	2696	0,19%	537,39
					1	2	2	1	1	2696	0,19%	537,39
					1	3	3	1	1	2696	0,19%	537,39
					1	4	4	1	1	2696	0,19%	537,39
					13	1	2	1	2	3520	0,24%	411,60
					26	2	4	1	2	3520	0,24%	411,60
					39	3	6	1	2	3520	0,24%	411,60
					53	4	8	1	2	3520	0,24%	411,60
Lights (max) all BRAMs in 4 columns	53	32	4	1448816	1	1	1	1	1	2696	0,19%	537,39
					1	2	2	2	2	4360	0,30%	332,30
					1	3	3	3	3	6024	0,42%	240,51
					1	4	4	4	4	7688	0,53%	188,45
					13	1	2	1	2	3520	0,24%	411,60
					26	2	4	2	4	6008	0,41%	241,15
					39	3	6	3	6	8496	0,59%	170,53
					53	4	8	4	8	10984	0,76%	131,90

Table B.8: Results of partial reconfiguration of CMCUs - Part 2

Benchmark					Modification				Results			
Name	Micro-instr.	Micro-oper.	BRAMs	Full bit stream	Modified microinstr.	Modified BRAMs	Modified lines	Modified columns	Different frames	Partial bit stream	% of original	Reduction
Test027 (default) all BRAMs in 3 columns	100	152	5	1448816	1	1	1	1	1	2699	0,19%	536,80
					1	2	2	2	2	4360	0,24%	411,13
					1	3	3	2	2	4360	0,24%	411,60
					1	4	4	3	3	6024	0,30%	333,29
					1	5	5	3	3	6024	0,30%	333,06
					20	1	16	1	16	15060	1,04%	96,20
					40	2	32	2	32	29091	2,01%	49,80
					60	3	48	2	32	29088	2,01%	49,81
					80	4	64	3	48	42488	2,93%	34,10
					100	5	68	3	48	42491	2,93%	34,10
Test027 (min) all BRAMs in 1 column	100	152	5	1448816	1	1	1	1	1	2699	0,19%	536,80
					1	2	2	1	1	2696	0,19%	537,39
					1	3	3	1	1	2696	0,19%	537,39
					1	4	4	1	1	2696	0,19%	537,39
					1	5	5	1	1	2696	0,19%	537,39
					20	1	16	1	2	3520	0,24%	411,60
					40	2	32	1	4	5168	0,36%	280,34
					60	3	48	1	8	9292	0,64%	155,92
					80	4	64	1	16	15055	1,04%	96,23
					100	5	68	1	16	15060	1,04%	96,20
Test027 (max) all BRAMs in 5 columns	100	152	5	1448816	1	1	1	1	1	2696	0,19%	537,39
					1	2	2	2	2	4360	0,30%	332,30
					1	3	3	3	3	6024	0,42%	240,51
					1	4	4	4	4	7688	0,53%	188,45
					1	5	5	5	5	9352	0,65%	154,92
					20	1	16	1	16	15060	1,04%	96,20
					40	2	32	2	32	29088	2,01%	49,81
					60	3	48	3	48	43116	2,98%	33,60
					80	4	64	4	64	54048	3,73%	26,81
					100	5	68	5	68	54984	3,80%	26,35

Table B.9: Results of partial reconfiguration of CMCUs - Part 3

Benchmark					Modification				Results			
Name	Micro-instr.	Micro-oper.	BRAMs	Full bit stream	Modified microinstr.	Modified BRAMs	Modified lines	Modified columns	Different frames	Partial bit stream	% of original	Reduction
Test036 (default) all BRAMs in 5 columns	151	452	13	1448816	1	1	1	1	1	2696	0,19%	537,39
					1	2	2	1	1	2696	0,19%	537,39
					1	4	4	2	2	4360	0,30%	332,30
					1	8	8	4	4	7688	0,53%	188,45
					1	13	13	5	5	9352	0,65%	154,92
					30	1	30	1	30	26697	1,84%	54,27
					60	2	60	2	60	52273	3,61%	27,72
					90	4	64	3	48	43351	2,99%	33,42
					120	8	128	4	64	57519	3,97%	25,19
					151	13	416	5	160	118263	8,16%	12,25
Test036 (min) all BRAMs in 1 column	151	452	13	1448816	1	1	1	1	1	2696	0,19%	537,39
					1	2	2	1	1	2696	0,19%	537,39
					1	4	4	1	1	2696	0,19%	537,39
					1	8	6	1	1	2696	0,19%	537,39
					1	13	13	1	1	2696	0,19%	537,39
					30	1	30	1	30	26701	1,84%	54,26
					60	2	60	1	30	26701	1,84%	54,26
					90	4	64	1	30	26701	1,84%	54,26
					120	8	128	1	32	28349	1,96%	51,11
					151	13	416	1	32	28349	1,96%	51,11
Test036 (max) all BRAMs in 8 columns	151	452	13	1448816	1	1	1	1	1	2696	0,19%	537,39
					1	2	2	1	1	2696	0,19%	537,39
					1	4	4	3	3	6143	0,42%	235,85
					1	8	8	5	5	9352	0,65%	154,92
					1	13	13	8	8	14495	1,00%	99,95
					30	1	30	1	30	26702	1,84%	54,26
					60	2	60	2	60	52273	3,61%	27,72
					90	4	64	4	64	57531	3,97%	25,18
					120	8	128	5	80	71427	4,93%	20,28
					151	13	416	8	256	180449	12,45%	8,03

Bibliography

Adamski, M. (1980). Programowane asynchroniczne układy sterujące z samosynchronizacją, *Prace VIII Krajowej Konferencji Automatyki KKA'80*, Szczecin, Polska, pp. 203–208.

Adamski, M. (1999). Application specific logic controllers for safety critical systems, *14th World Congress of IFAC International Federation of Automatic Control*, Vol. Q: Transportation Systems: Computer Control, Oxford, International Federation of Automatic Control, Beijing, China, pp. 519–524.

Adamski, M. and Barkalov, A. (2006). *Architectural and Sequential Synthesis of Digital Devices*, University of Zielona Góra Press, Zielona Góra.

Adamski, M. and Węgrzyn, M. (2003). Reprogrammable controllers for reactive embedded systems, *in* M. W. ed. M. Colnaric, M. Adamski (ed.), *Real-time programming 2003 (WRTP 2003) : a proceedings volume from the 26th IFAC/IFIP/IEEE Workshop, ISBN: 0-08-044203 X*, Elsevier, Oxford, pp. 39–44.

Adamski, M., Węgrzyn, M. and Węgrzyn, A. (2007). Safe reconfigurable logic controllers design, *in* ed. by J. Korbicz (ed.), *Measurements models systems and design, ISBN: 978-83-206-1644-6*, Wydaw. Komunikacji i Łączności, Warszawa, pp. 343–370.

Ahmad, I., Ali, F. and Ul-Mustafa, R. (2000). *An Integrated State Assignment and Flip-Flop Selection Technique for FSM Synthesis*, 141–152, Microprocessors and Microsystems.

Aho, A. V., Hopcroft, J. E. and Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass.

Altera (2006). *Embedded memory in Altera FPGAs*, Altera, http://www.altera.com/technology/memory/embedded/mem-embedded.html.

Altera (2008). *Altera Devices Website*, http://www.altera.com/products/devices/dev-index.jsp.

Ashar, P., Devadas, S. and Newton, A. R. (1990). A unified approach to the decomposition and re-decomposition of sequential machines, *DAC '90: Proceedings of the 27th ACM/IEEE conference on Design automation*, ACM, New York, NY, USA, pp. 601–606.

Ashar, P., Devadas, S. and Newton, A. R. (1992). *Sequential Logic Synthesis*, Kluwer Academic Publishers, Norwell, MA, USA.

Baranov, S. I. (1994). *Logic Synthesis for Control Automata*, Kluwer Academic Publishers, Boston.

Barkalov, A. (1998). Principles of optimization of logical circuit of Moore finite-state-machine, *Cybernetics and System Analysis* **No. 1**: 65–72.

Barkalov, A. (2002). *Synthesis of Control Units on PLDs*, DonNTU, Donetsk.

Barkalov, A., Bukowiec, A. and Wiśniewski, R. (2005*a*). Sintez mikroprogrammnogo avtomata s predstavleniem termov funkcij vozbuzdenija kak par mikrokomand, *Radiotechnika* **nr 142**: 92–96. Charkivs'kij nacional'nij universitet radioelektroniki, Carkiv.

Barkalov, A., Efimenko, K. and Wiśniewski, R. (2006*a*). Optimizaciâ shemy adresacii kompozicionnogo ustrojstva upravleniâ, *Naukovi Praci Donec'kogo Nacional'nogo Tehnièogo Universitetu : Problemi Modeljuvannja ta Avtomatizacii Proektuvannja Dinamicnich Sistem* **nr 5**: 156–161. Doneckij nacional'nyj technicnyj universitet, Doneck.

Barkalov, A. and Palagin, A. (1997). *Synthesis of Microprogram Control Units*, IC NAC of Ukraine, Kiev.

Barkalov, A., Titarenko, L. and Wiśniewski, R. (2005*b*). Optimization of the amount of LUT-elements in compositional microprogram control unit with mutual memory, *Proceedings of IEEE East-West Design & Test Workshop - EWDTW '05; ISBN: 966-659-113-8*, Ministry of Education and Science of Ukraine, Kharkov National University of Radioelectronics, Odessa, IEEE EWDTW, Odessa, Ukraina, pp. 75–79.

Barkalov, A., Titarenko, L. and Wiśniewski, R. (2005*e*). Synthesis of compositional microprogram control units with transformation of the numbers of inputs, *The experience of designing and application of CAD systems in microelectronics : proceedings of the VIIIth International Conference CADSM 2005; ISBN: 966-553-431-9*, CAD departament of the Institute of Computer Science and Information Technologies, Lviv Polytechnic National University, Lviv, Publishing House of Lviv Polytechnic National University, Lviv - Polyana, Ukraina, pp. 181–184.

Barkalov, A., Titarenko, L. and Wiśniewski, R. (2007a). Optimization of the circuit of compositional microprogram control unit with mutual memory, *The experience of designing and application of CAD systems in microelectronics : proceedings of the IXth International Conference CADSM 2007; ISBN: 978-966-553-587-4*, CAD departament of the Institute of Computer Science and Information Technologies, Lviv Polytechnic National University, Lviv, Publishing House of Lviv Polytechnic National University, Lviv - Polyana, Ukraina, pp. 251–255.

Barkalov, A., Titarenko, L. and Wiśniewski, R. (2007b). Synthesis of compositional microprogram control units with function decoder for telecommunication systems, *Radiotehnika : Problemy telekommunikacij* **nr 151**: 106–111. Charkivskij nacionalnij universitet radioelektroniki, Charkiv.

Barkalov, A. and Węgrzyn, M. (2006). *Design of Control Units with Programmable Logic*, University of Zielona Góra Press, Zielona Góra.

Barkalov, A., Węgrzyn, M. and Wiśniewski, R. (2006). Partial reconfiguration of compositional microprogram control units implemented on FPGAs, *Programmable Devices and Embedded Systems - PDeS 2006 : proceedings of IFAC*

workshop; ISBN: 80-214-3130-X, Departament of Control and Instrumentation Faculty of Electrical Engineering and Communication Brno University of Technology Czech Republic, Brno, Brno, Czechy, pp. 116–119.

Barkalov, A., Węgrzyn, M. and Wiśniewski, R. (2006*c*). Optimization of LUT-elements amount in cotrol unit of system-on-chip, *Discrete-Event System Design - DESDes '06 : a proceedings volume from the 3rd IFAC Workshop; ISBN: 83-7481-035-1*, Institute of Computer Engineering and Electronics University of Zielona Góra, Zielona Góra, International Federation of Automatic Control by University of Zielona Góra Press, Rydzyna, Polska, pp. 143–146.

Barkalov, A. and Wiśniewski, R. (2004a). Design of compositional microprogram control units with maximal encoding of inputs, *Radioelektronika i Informatika* (no 3): 79–81.

Barkalov, A. and Wiśniewski, R. (2004b). Design of compositional microprogram control units with transformation of the number of transactions, *Mixed Design of Integrated Circuits and Systems - MIXDES 2004 : Proceedings of the 11th International Conference; ISBN: 83-919289-7-7*, Departament of Microelectronics and Computer Science, Technical University of Lodz, Lodz, Szczecin, Polska, pp. 172–175.

Barkalov, A. and Wiśniewski, R. (2004c). Design of control units with transformation of the number of transaction, *Radiotechnika* **nr 138**: 110–113. Charkivskij nacionalnij universitet radioelektroniki, Charkiv.

Barkalov, A. and Wiśniewski, R. (2004d). Optimization of compositional microprogram control unit with elementary operational linear chains, *Upravljuscije Sistemy i Masiny* **No. 5**: 25–29.

Barkalov, A. and Wiśniewski, R. (2004e). Optimization of compositional microprogram control units with sharing of codes, *Avtomatizacija proektirovanija diskretnych sistem : materialy pjatoj mezdunarodnoj konferencii; ISBN: 985-6744-06-7*, Vol. T. 1, Nacional'naja Akademija Nauk Belarusi, Minsk, Minsk, Bialorus, pp. 16–22.

Barkalov, A. and Wiśniewski, R. (2004f). Synthesis of compositional microprogram control units with transformation of the numbers of inputs, *Discrete -*

Event System Design - DESDes' 04 : Proceedings of the International Workshop; ISBN: 83-89712-15-6, Institute of Computer Science and Electronics University of Zielona Góra, Zielona Góra, University of Zielona Góra Press, Dychów, Polska, pp. 145–148.

Barkalov, A. and Wiśniewski, R. (2005a). Implementation of compositional microprogram control unit on FPGAs, *Proceedings of IEEE East-West Design & Test Workshop - EWDTW '05; ISBN: 966-659-113-8*, Ministry of Education and Science of Ukraine, Kharkov National University of Radioelectronics, Odessa, IEEE EWDTW, Odessa, Ukraina, pp. 80–83.

Barkalov, A. and Wiśniewski, R. (2005b). Optimization of compositional microprogram control units implemented on system-on-chip, *Informatyka Teoretyczna i Stosowana* **R. 5**(nr 9): 7–22.

Barkalov, A. and Wiśniewski, R. (2005c). Sintez kompozicionnogo mikroprogrammnogo ustrojstva upravlenija s optimal'nym kodirovaniem elementarnych operatornych linejnych cepej, *Informatika* (no 1): 95–102.

Barkalov, A., Wiśniewski, R. and Babakov, R. (2004). Optimizacija kompozicionnogo mikroprogramnogo ustrojstva upravlenija s elementarnymi operatornymi linejnymi celjami, *Naukovi Praci Donec'kogo Nacional'nogo Technicnogo Universitetu : Obcisljuval'na Technika ta Avtomatizacija* **nr 77**: 210–216. Doneckij nacional'nyj technicnyj universitet, Doneck.

Barkalov, A., Wiśniewski, R. and Efimenko, K. (2005*d*). Realizacija kompozicionnogo mikroprogramnogo ustrojstva upravlenija na FPGA, *Radioelektronika Informatika Upravlinnja* **nr 2**: 127–131. Zaporiz'kij nacional'nij technicnij universitet, Zaporizzja.

Barkalov, A., Wiśniewski, R. and Greczko, E. (2005*f*). Synteza układów mikroprogramowanych w strukturach FPGA, *Reprogramowalne Układy Cyfrowe - RUC 2005 : materiały VIII krajowej konferencji naukowej; ISBN: 83-87362-69-7*, Politechnika Szczecińska, Szczecin, Politechnika Szczecińska, Szczecin, Polska, pp. 17–23.

Barkalov, A., Wiśniewski, R., Kovalyov, S. and Efimenko, K. (2006*b*). Optimizaciâ èisla LUT-elementov v ustrojstve upravleniâ sistemy na kristalle, *Mašinostroenie i tehnosfera XXI veka : sbornik trudov XIII meždunarodnoj nauènotehnièeskoj konferencii; ISBN: 966-7907-20-1*, Vol. T. 1, Ministerstvo obrazovaniâ i nauki Ukrainy, Doneck, DonNTU, Sevastopol, Ukraina, pp. 75–80.

Barkalov, A., Wiśniewski, R. and Titarenko, L. (2005*c*). Synthesis of compositional microprogram control unit on FPGA, *Mixed Design of Integrated Circuits and Systems - MIXDES 2005 : proceedings of the 12th International Conference; ISBN: 83-919289-9-3*, Vol. Vol. 1, Technical University of Lódz, Warsaw University of Technology, AGH University of Science and Technology, Kraków, Kraków, Polska, pp. 205–208.

Berge, C. (1985). *Graphs and Hypergraphs*, Elsevier Science Ltd.

Bibilo, P. (1999). *Background of VHDL*, Solon-R, Moscow.

Bolton, M. (1990). *Digital Systems Design with Programmable Logic*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Borowik, G. (2004). Synteza układów sekwencyjnych w siechiach wbudowanych matryc logicznych struktur FPGA, *Materiały VI Międzynarodowych Warsztatów Doktoranckich OWD'04*, Vol. 19 of *Archiwum Konferencji PTETiS*, Wisła, Polska, pp. 361–366.

Borowik, G. (2005). FSM coding for optimal serial decomposition, *Materiały VII Międzynarodowych Warsztatów Doktoranckich OWD'05*, Vol. 21 of *Archiwum Konferencji PTETiS*, Wisła, Polska, pp. 243–248.

Brown, S. and Vernesic, Z. (2000). *Fundamentals of digital logic with VHDL design*, McGraw Hill.

Bursky, D. (1999). Embedded logic and memory find a home in FPGAs, *Electronic Design* **Vol. 47**(No. 14): 43–56.

Ciesielski, M. and Yang, S. (1992). PLA de: a two-stage PLA decomposition., *IEEE Trans. on CAD of Integrated Circuits and Systems* **11**(8): 943–954.

Clements, A. (2000). *The principles of computer hardware*, Oxford University Press, New Jersey.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. (2001). *Introduction to Algorithms*, 2nd edn, The MIT Press.

Dagless, E. L. (1983). PLA and ROM based design, *in* P. J. Hicks (ed.), *Semi-Custom IC Design and VLSI*, IEE Digital Electronics and Computing Series 2, Peter Peregrinus Ltd., Herts, pp. 121–135.

De Micheli, G. (1994). *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York.

Devadas, S., Wang, A., Newton, R. and Sangiovanni-Vincentelli, A. L. (1989). Boolean decomposition in multilevel logic optimization, *IEEE Journal of solid-state circuits* pp. 399–408.

Devadas, S., Wang, A. R., Newton, A. R. and Sangiovanni-Vincentelli, A. L. (1988). Boolean decomposition of programmable logic arrays, *CICC'88* .

Gajski, D. (1997). *Principles of Digital Design*, Prentice Hall, New Jersy.

Halang, W. A. and Krämer, B. (1992). Achieving high integrity of process control software by graphical design and formal verification, *Softw. Eng. J.* **7**(1): 53–64.

Halang, W. A. and Krämer, B. J. (1994). Safety assurance in process control, *IEEE Softw.* **11**(1): 61–67.

Halang, W. and Adamski, M. (1997). A programmable electronic system for safety related control applications, *Advances in safety and reliability : proceedings of the International Conference - ESREL '97, ISBN: 0-08-042835-5*, Vol. Vol. 1, Pergamon, Oxford, pp. 349–355.

Harary, F. (1994). *Graph Theory*, Addison-Wesley, Reading, MA.

Hrynkiewicz, E., Pucher, K. and Kania, D. (1997). The input partitioning and coding problem in PAL -based CPLD s, *XX National Conference Circuit Theory and Electronic Networks* pp. 145–152.

Husson, S. (1970). *Microprogramming - Principles and Practices*, Prentice Hall, New York.

IEEE (2001). *IEEE Standard Verilog Hardware Description Language 1364-2001*, New York.

Jenkins, J. (1994). *Designing with FPGAs and CPLD s*, Prentice Hall, New Jersy.

Jóźwiak, L. and Chojnacki, A. (2003). Effective and efficient FPGA synthesis through general functional decomposition, *J. Syst. Archit.* **49**(4-6): 247–265.

Kania, D. (1999). Two-level logic synthesis on PAL -based CPLD and FPGA using decomposition, *Procedings of 25-th Euromicro Conference*, IEEE Computer Society Press, pp. 278–281.

Kania, D. (2004). *Synteza logiczna przeznaczona dla matrycowych struktur programowalnych typu PAL*, Zeszyty Naukowe Politechniki 'Sl,askiej, Gliwice.

Kania, D. (2007). A new approach to logic synthesis of multi-output boolean functions on PAL -based CPLD s, *GLSVLSI '07: Proceedings of the 17th ACM Great Lakes symposium on VLSI*, ACM, New York, NY, USA, pp. 152–155.

Kania, D. and Kulisz, J. (2007). Logic synthesis for PAL-based CPLD-s based on two-stage decomposition, *J. Syst. Softw.* **80**(7): 1129–1141.

Kania, D., Kulisz, J., Milik, A. and Czerwiński, R. (2005). Modele dekompozycji przeznaczone dla struktur matrycowych, *RUC'2005* pp. 77–84.

Kernighan, B. W. and Ritchie, D. M. (1977). *The C Programming Language*, Prentice-Hall, Englewood Cliffs, NJ.

Kravcov, L. and Chernicki, G. (1976). *Design of microprogram control units*, Energia, Leningrad. in Russian.

Kubátová, H. (2005). Finite state machine implementation in FPGAs, *in* M. Adamski, A. Karatkevich and M. Węgrzyn (eds), *Design of Embedded Control Systems*, Springer, New York, pp. 177–187.

Łach, J., Sapiecha, E. and Zbierzchowski, B. (2003). Synteza układów sekwencyjnych w strukturach FPGA z wbudowanymi blokami pamięci, *Przegląd Telekomunikacyjny i Wiadomości Telekomunikacyjne* **Nr 2-3**: 81–86.

Lee, J. M. (1999). *VERILOG QuickStart: A Practical Guide to Simulation and Synthesis in VERILOG*, Kluwer Academic Publishers, Norwell, MA, USA.

Łuba, T. (2001). *Synteza układów logicznych*, WSISiZ, Warszawa.

Łuba, T. (2003). *Synteza układów cyfrowych*, WKŁ, Warszawa.

Łuba, T. (2005). *Synteza układów logicznych*, Oficyna Wydawnicza PW, Warszawa.

Łuba, T., Rawski, M. and Jachna, Z. (2002). Functional decomposition as a unversal method of logic synthesis for digital circuits, *Proceedings of the 9th International Conference Mixed Design of Integrated Circuits and Systems MixDes'02*, Wrocław, Poland, pp. 285–290.

Maxfield, C. (2004). *The Design Warrior's Guide to FPGAs*, Academic Press, Inc., Orlando, FL, USA.

McCluskey, E. (1986). *Logic design principles*, Prentice Hall, Englewood Cliffs.

Mealy, G. (1955). A method for synthesizing sequential circuits, *BSTJ* **34**: 1045–1079.

Mesquita, D., Moraes, F., Palma, J., Möller, L. and Calazans, N. (2003). Remote and partial reconfiguration of FPGAs: Tools and trends, *Proc. International Parallel and Distributed Processing Symposium (IPDPS'03)*, IEEE, pp. 177–185.

Misiurewicz, P. (1982). *Podstawy techniki cyfrowej*, WNT, Warszawa.

Molski, M. (1986). *Modułowe i mikroprogramowalne układy cyfrowe*, WKŁ, Warszawa.

Moore, E. (1956). Gedanken experiments on sequential machines, *in* C. E. Shannon and J. McCarthy (eds), *Automata Studies*, PUP, pp. 129–153.

Muthukumar, V., Bignall, R. J. and Selvaraj, H. (2007). An efficient variable partitioning approach for functional decomposition of circuits, *J. Syst. Archit.* **53**(1): 53–67.

Papachristou, C. A. (1979). A scheme for implementing microprogram addressing with programmable logic arrays, *Digital Processes* **Vol. 5**(No. 3-4): 235–256.

Parnell, K. and Mehta, N. (2003). *Programmable Logic Design Quick Start Hand Book*, Xilinx.

Pasierbiński, J. and Zbysiński, P. (2001). *Uklady programowalne w praktyce*, WKL, Warszawa.

Pęcheux, F., Lallement, C. and Vachoux, A. (2005). VHDL-AMS and Verilog-AMS as alternative hardware description languages for efficient modeling of multidiscipline systems, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* **24**(2): 204–225.

Perkowski, M., Jóźwiak, L. and Zhao, W. (2001). Symbolic two-dimensional minimization of strongly unspecified finite state machines, *Journal of Systems Architecture* **Vol. 47**: 15–28.

Rawski, M., Jóźwiak, L. and Łuba, T. (2001). Functional decomposition with an efficient input support selection for sub-functions based on information relationship measures, *Journal of Systems Architecture* **Vol. 47**: 137–155.

Rawski, M., Łuba, T., Jachna, Z. and Tomaszewicz, P. (2005*b*). The influence of functional decomposition on modern digital design process, *in* M. Adamski, A. Karatkevich and M. Węgrzyn (eds), *Design of Embedded Control Systems*, Springer, Boston, pp. 193–206.

Rawski, M., Selvaraj, H. and Luba, T. (2003). An application of functional decomposition in rom-based fsm implementation in FPGA devices, *DSD '03: Proceedings of the Euromicro Symposium on Digital Systems Design*, IEEE Computer Society, Washington, DC, USA, p. 104.

Rawski, M., Tomaszewicz, P., Selvaraj, H. and Łuba, T. (2005*a*). Efficient implementation of digital filters with use of advanced synthesis methods targeted

FPGA architectures, *DSD '05: Proceedings of the 8th Euromicro Conference on Digital System Design*, IEEE Computer Society, Washington, DC, USA, pp. 460–466.

Salcic, Z. (1998). *VHDL and FPLDs in Digital Systems Design, Prototyping and Customization*, Kluwer Academic Publishers, Boston.

Sasao, T. (1988). Multiple-valued logic and optimization of programmable logic arrays, *Computer* **21**(4): 71–80.

Sasao, T. (1999). Totally undecomposable functions: Applications to efficient multiple-valued decompositions, *ISMVL '99: Proceedings of the Twenty Ninth IEEE International Symposium on Multiple-Valued Logic*, IEEE Computer Society, Washington, DC, USA, p. 59.

Scholl, C. (2001). *Functional decomposition with application to FPGA synthesis*, Kluwer Academic Publishers.

Selvaraj, H. and Luba, T. (1995). A balanced multilevel decomposition method, *EDTC '95: Proceedings of the 1995 European conference on Design and Test*, IEEE Computer Society, Washington, DC, USA, p. 594.

Selvaraj, H., Tomaszewicz, P., Rawski, M. and Luba, T. (2005). Efficient application of modern logic synthesis in FPGA-based designing of information and signal processing systems, *ITCC '05: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, IEEE Computer Society, Washington, DC, USA, pp. 22–27.

Sentovich, E. M. (1993). *Sequential Circuit Synthesis at the Gate Level*, PhD thesis. Chair-Robert K. Brayton.

Sentovich, E., Singh, K. J., Moon, C. W., Savoj, H., Brayton, R. K. and Sangiovanni-Vincentelli, A. L. (1992*a*). Sequential circuit design using synthesis and optimization, *ICCD '92: Proceedings of the 1991 IEEE International Conference on Computer Design on VLSI in Computer & Processors*, IEEE Computer Society, Washington, DC, USA, pp. 328–333.

Sentovich, E., Singh, K., Lavagno, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Stephan, P. R., Brayton, R. K. and Sangiovanni-Vincentelli, A. (1992*b*). Sis: A system for sequential circuit synthesis, *Technical Report UCB/ERL M92/41*, U.C. Berkeley.

Skahill, K., Legenhausen, J., Wade, R., Wilner, C. and Wilson, B. (1996). *VHDL for Programmable Logic*, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

Solovjev, V. (1996). *Design of the Functional Units of Digital Systems Using Programmable Logic Devices*, Bestprint, Minsk.

Stalings, W. (1996). *Computer organization and architecture*, Prentice Hall, New Jersey.

Stroustrup, B. (1986). *The C++ Programming Language*, Addison-Wesley, Reading, MA.

Thomas, D. and Moorby, P. (2002). *The Verilog hardware description language*, 5th edn, Kluwer Academic Publishers, Norwell, MA, USA.

Traczyk, W. (1982). *Układy cyfrowe. Podstawy teoretyczne i metody syntezy*, WNT, Warszawa.

Wilkes, M. V. (1951). The best way to design an automatic calculating machine, *Manchester University inaugural conference*, Manchester, England.

Wilson, R. J. (1979). *Introduction to Graph Theory*, Academic Press, New York, USA.

Wiśniewska, M. and Wiśniewski, R. (2005). Zastosowanie transwersali hipergrafów do minimalizacji rozmiaru pamięci jednostek sterujących, *Informatyka - sztuka czy rzemiosło - KNWS' 05 : materiały II konferencji naukowej; ISBN: 83-89712-62-8*, Instytut Informatyki i Elektroniki Uniwersytetu Zielonogórskiego, Zielona Góra, Oficyna Wydaw. Uniwersytetu Zielonogórskiego, Złotniki Lubańskie, Polska, pp. 23–32.

Wiśniewski, R. (2004). Design of compositional microprogram control units with sharing of the codes, *VI Międzynarodowe Warsztaty Doktoranckie - OWD 2004; ISBN: 83-915991-8-3*, Vol. Archiwum Konferencji PTETiS, Vol. 19, z. 4, Polskie Towarzystwo Elektrotechniki Teoretycznej i Stosowanej, Wydzial Elektryczny Politechniki Slaskiej w Gliwicach, Gliwice, Wisla, Polska, pp. 217–220.

Wiśniewski, R. (2005). Projektowanie ukladów mikroprogramowanych z wykorzystaniem wbudowanych bloków pamięci w matrycach programowalnych, *Informatyka - sztuka czy rzemiosło - KNWS' 05 : materialy II konferencji naukowej; ISBN: 83-89712-62-8*, Instytut Informatyki i Elektroniki Uniwersytetu Zielonogórskiego, Zielona Góra, Oficyna Wydaw. Uniwersytetu Zielonogórskiego, Zlotniki Lubańskie, Polska, pp. 33–38.

Wiśniewski, R. (2006a). Design of compositional microprogram control units with elementary operational linear chains, *Discrete-Event System Design - DESDes '06 : a proceedings volume from the 3rd IFAC Workshop; ISBN: 83-7481-035-1*, Institute of Computer Engineering and Electronics University of Zielona Góra, Zielona Góra, International Federation of Automatic Control by University of Zielona Góra Press, Rydzyna, Polska, pp. 191–194.

Wiśniewski, R. (2006b). Synteza mikroprogramowanych układów sterujących ze współdzieleniem kodów z wykorzystaniem dekodera adresów, *Pomiary Automatyka Kontrola* (nr 6, wyd. spec.): 38–40.

Wiśniewski, R. and Barkalov, A. (2007). Synthesis of compositional microprogram control units with function decoder, *International Workshop Control and Information Technology - IWCIT 2007; ISBN: 978-80-248-1567-1*, VSB - Technical University of Ostrava Faculty of Electrical Engineering and Computer Science, Departament of Measurement and Control Czech Rpublic, Ostrava, VSB - Technical University of Ostrava, Ostrava, Czechy, pp. 229–232.

Wiśniewski, R., Barkalov, A. and Titarenko, L. (2006a). Optimization of address circuit of compositional microprogram unit, *Proceedings of IEEE East-West Design & Test Workshop - EWDTW '06; ISBN: 966-659-124-3*, Kharkov National University of Radioelectronics, Kharkov, Sochi, Rosja, pp. 167–170.

Wiśniewski, R., Barkalov, A. and Titarenko, L. (2006b). Synteza mikroprogramowanych układów sterujących ze współdzieleniem kodów z wykorzystaniem konwertera adresów, *Pomiary Automatyka Kontrola* (nr 7, wyd. spec.): 121–123.

Wiśniewski, R., Barkalov, A. and Titarenko, L. (2006c). Synthesis of compositional microprogram control units with sharing codes and address decoder, *Mixed Design of Integrated Circuits and Systems - MIXDES 2006 : proceedings of the international conference; ISBN: 83-922632-1-9*, Departament of Microelectronics and Computer Science, Technical University of Łódź, Łódź, Gdynia, Polska, pp. 397–400.

Wiśniewski, R., Barkalov, A. and Titarenko, L. (2007). Synthesis of compositional microprogram control units with olc output identification, *Computer-Aided Design of Discrete Devices - CAD DD '07 : proceedings of the Sixth International Conference; ISBN: 978-985-6744-34-4*, Vol. T. 2, Minsk, Minsk, Białoruś, pp. 81–86.

Wiśniewski, R. and Węgrzyn, M. (2005). Hardware acceleration and verification of systems designed with hardware description languages (hdl), *Proceedings of SPIE* **Vol. 5775**: 365–376.

Wiśniewski, R. (2005). Częściowa rekonfiguracja mikroprogramowanych układów sterujacych implementowanych z wykorzystaniem struktur FPGA, *VII Międzynarodowe Warsztaty Doktoranckie - OWD 2005; ISBN: 83-922242-0-5*, Vol. Archiwum Konferencji PTETIS, Vol.21, Polskie Towarzystwo Elektrotechniki Teoretycznej i Stosowanej, Wydzial Elektryczny Politechniki Slaskiej w Gliwicach,, Gliwice, Wisla, Polska, pp. 239–242.

Xilinx (2000). *Using Block SelectRAM+ Memory in Spartan-II FPGAs*, www.xilinx.com/bvdocs/appnotes/xapp130.pdf.

Xilinx (2001). *ASIC alternatives*, http://www.xilinx.com/.
*http://www.xilinx.com/

Xilinx (2004). *Two flows for partial reconfiguration*, http://direct.xilinx.com/bvdocs
/appnotes/xapp290.pdf.

Xilinx (2005). *Using Block RAM in Spartan-3 Generation FPGAs*, www.xilinx.com/bvdocs/appnotes/xapp463.pdf.

Xilinx (2007). *Virtex-II Pro and Virtex-II Pro X FGPA User Guide*, http://www.xilinx.com/support/documentation/user_guides/ug012.pdf.

Yang, S. and Ciesielski, M. (1989). PLA decomposition with generalized decoders, *Proc. ICCAD*, IEEE CS Press, IEEE Computer Society Press, pp. 312–315.

Zwoliński, M. (2003). *Digital System Design with VHDL*, second edn, Prentice Hall, New Jersy.

List of Figures

List of Tables

Metody syntezy układowej części sterującej w mikrosystemie cyfrowym

Streszczenie

Jednostka sterująca jest jednym z najważniejszych elementów układu cyfrowego. Bardzo szybki rozwój w dziedzinie techniki cyfrowej spowodował pojawienie się zintegrowanych układów takich jak System-on-a-Chip (*SoC*) czy System-on-a-Programmable-Chip (*SoPC*), w których bloki funkcjonalne projektowanego układu implementowane są z wykorzystaniem matryc programowalnych FPGA (*Field Programmable Gate Array*). Takie podejście wymusza modyfikację klasycznych metod projektowania jednostek sterujących. Główną cechą matryc FPGA jest wykorzystanie elementów LUT (*Look-Up Table*) do realizacji funkcji logicznych. Liczba wejść elementu LUT jest ściśle ograniczona, co wiąże się z zastosowaniem dekompozycji w projektowanym układzie.

Jedną z metod zmniejszenia liczby wykorzystanych elementów LUT jest strukturalna dekompozycja jednostki sterującej. Wówczas system jest realizowany jako układ wielopoziomowy, w którym mikrooperacje mogą zostać zaimplementowane w dedykowanych blokach pamięci układu FPGA. Ponieważ pojemność dedykowanych bloków pamięci układów programowalnych jest ograniczona, pojemność pamięci jednostki sterującej powininna być jak najmniejsza. Wymagania te spełnia mikroprogramowany układ sterujący, w którym zastosowano dekompozycję sterownika na część zarządzającą (adresującą) oraz pamięć, w której przechowywane są mikroinstrukcje kontrolera. Należy tu jednakże zaznaczyć, że stosowanie mikroprogramowanych układów sterujących ma sens w przypadku, gdy sterownik może zostać zinterpretowany opisem tzw. liniowej sieci działań. W sieci takiej liczba bloków operacyjnych stanowi co najmniej 75% wszystkich bloków sieci.

W rozprawie zaproponowano sześć autorskich metod syntezy mikropogramowanych układów sterujących. Celem opracowanych algorytmów było zmniejszenie liczby wykorzystanych elementów logicznych docelowego układu FPGA. Ze względu na strukturę, przedstawione metody syntezy zostały podzielone na dwie grupy. Pierwsza dotyczy mikroprogramowanych układów sterujących o adresowaniu wspólnym, gdzie adres mikroinstrukcji jest wykorzystywany do rozpoznania stanów wewnętrznych sterownika. Zaproponowano trzy nowe metody syntezy układu o adresowaniu wspólnym:

- *Układ z dekoderem funkcji* - w którym wprowadzono dodatkowy blok dekodera funkcji. Ideą metody jest zakodowanie funkcji wzbudzeń dla licznika, która jest następnie dekodowana przez dekoder funkcji. Dodatkowy blok jest implementowany z wykorzystaniem dedykowanych bloków pamięci matryc FPGA, co pozwala zredukować liczbę wykorzystanych elementów LUT w porównaniu do tradycyjnej realizacji sterownika o adresowaniu wspólnym.

- *Układ z identyfikacją wyjść* - w którym stany wewnętrzne sterownika kodowane są z wykorzystaniem minimalnej, niezbędnej liczby bitów. Dzięki temu zmniejszona zostaje liczba wejść układu kombinacyjnego, a co za tym idzie liczba bloków logicznych, niezbędnych do implementacji tego modułu oraz całego sterownika w matrycach FPGA.

- *Układ z identyfikacją wyjść oraz dekoderem funkcji* - w którym zastosowano połączenie obu wyżej opisanych rozwiązań.

Druga grupa metod bazuje na idei współdzielenia kodów. W tym przypadku adres mikroinstrukcji jest wyznaczany na podstawie kodów generowanych zarówno przez licznik, jak i przez rejestr. W obrębie sterownika ze współdzieleniem kodów także zaproponowano trzy nowe rozwiązania:

- *Układ z dekoderem funkcji* - w którym wprowadzono dodatkowy blok dekodera funkcji. Ideą metody jest zakodowanie funkcji wzbudzeń dla licznika oraz dla rejestru. Funkcje te są następnie dekodowane przez dodatkowy blok. Dekoder funkcji jest realizowany z wykorzystaniem dedykowanych bloków pamięci matryc FPGA, co pozwala zredukować liczbę wykorzystanych elementów LUT w porównaniu do tradycyjnej realizacji sterownika o adresowaniu wspólnym.

- *Układ z konwerterem adresów* - w którym zastosowano dodatkowy blok konwertera adresów mikroinstrukcji. Metoda ma sens w przypadku, gdy rozmiar kodów generowanych przez licznik oraz rejestr jest większy niż minimalny rozmiar adresu mikroinstrukcji. Każdy nadmiarowy bit oznacza podwojenie pojemności pamięci układu mikroprogramowanego. Zastosowanie konwertera adresów pozwala utrzymać minimalny rozmiar adresu mikroinstrukcji.

- *Układ z konwerterem adresów oraz dekoderem funkcji* - w którym zastosowano połączenie obu prezentowanych powyżej idei. Konwerter adresów umożliwia zastosowanie metody ze współdzieleniem kodów w przypadku, gdy rozmiar kodów generowanych przez licznik oraz przez rejestr jest większy niż minimalny rozmiar adresu mikroinstrukcji. Dodatkowo dekoder funkcji pozwala zmniejszyć liczbę bloków LUT niezbędnych do realizacji układu w matrycach FPGA.

W celu weryfikacji skuteczności algorytmów zaproponowanych w rozprawie, opracowany został system automatycznej syntezy mikroprogramowanych układów sterujących (*ATOMIC*). Środowisko składa się z trzech niezależnych modułów, które realizują kolejne etapy projektowe jednostki sterującej. Na podstawie specyfikacji sterownika przedstawionej w formie tekstowej, *ATOMIC* przeprowadza automatyczny proces dekompozycji strukturalnej. Wynikiem działania systemu jest opis mikroprogramowanego układu sterującego w językach opisu sprzętu. Tak przygotowany sterownik może zostać w konsekwencji zaimplementowany w docelowym układzie FPGA. Warto podkreślić fakt, że *ATOMIC* jest niezależny od platformy oraz systemu operacyjnego. Program może zostać uruchomiony zarówno w środowisku Windows, Unix jak i Solaris.

Przeprowadzone eksperymenty potwierdzają skuteczność zaproponowanych metod syntezy mikroprogramowanych układów sterujących. W badaniach porównano opracowane algorytmy z tradycyjnym sposobem projektowania jednostek sterujących, realizowanych jako skończony automat stanów. Średnio największą skuteczność uzyskano podczas realizacji układu jako mikroprogramowany układ sterujący z konwerterem adresów oraz dekoderem funkcji. Taki sposób implementacji sterownika pozwala średnio zmniejszyć liczbę bloków LUT o 50% w porównaniu do realizacji układu jako skończony automat stanów.

Szczegółowa analiza wyników badań pozwoliła określić kryteria doboru metody, w zależności od specyfikacji jednostki sterującej:

- W przypadku relatywnie małych sterowników (w których liczba mikroinstrukcji nie przekracza 150, a pamięć układu może zostać zrealizowana z wykorzystaniem jednego bloku pamięci FPGA), należy zastosować układ ze współdzieleniem kodów oraz dekoderem funkcji.

- W przypadku sterowników, w których liczba mikroinstrukcji nie przekracza 150, jednakże pojemność pamięci jednostki sterującej jest większa niż pojemność jednego bloku pamięci układu FPGA, należy zastosować układ z konwerterem adresów oraz dekoderem funkcji.

- W przypadku sterowników, w których liczba mikroinstrukcji przekracza 150, najlepszym rozwiązaniem jest zastosowanie układu z identyfikacją wyjść oraz dekoderem funkcji.

Badania związane z częściową rekonfiguracją mikroprogramowanych układów sterujących implementowanych w matrycach FPGA pokazały, że redukcja strumienia danych jest silnie związana z rozlokowaniem pamięci realizowanego sterownika w matrycy FPGA. Najlepsze rezultaty zostały osiągnięte w przypadku implementacji modułu pamięci jednostki sterującej w blokach pamięci układu programowalnego, które są położone w tej samej kolumnie. Takie rozwiązanie pozwala zmniejszyć rozmiar strumienia przesyłanego do układu FPGA nawet ponad 500 razy.

Najważniejsze wyniki badań zaprezentowano na konferencjach oraz w czasopismach, w tym w dwudziestu dziewięciu o zasięgu międzynarodowym oraz w pięciu o zasięgu krajowym. Ponadto wymiernym efektem prowadzonych badań jest nawiązana współpraca z Uniwersytetem w Hagen (zespół pod kierunkiem profesora Halanga) oraz plany opatentowania opracowanych metod.

Część prowadzonych prac została zrealizowana w ramach projektu badawczego finansowanego ze środków Zintegrowanego Programu Operacyjnego Rozwoju Regionalnego z udziałem Europejskiego Funduszu Społecznego oraz grantu KBN nr 3 T11C 046 26.