Design Tools and Reusable Libraries for FPGA-Based Digital Circuits

Valery Sklyarov, Iouliia Skliarova, Pedro Almeida, Manuel Almeida
skl@ieeta.pt, iouliia@det.ua.pt
Department of Electronics and Telecommunications
University of Aveiro
3810-193 Aveiro, Portugal

Abstract

This paper suggests tools that provide significant improvements in the design and verification of FPGA-based digital circuits. These tools include reusable specifications of hardware components (modules) that have been proposed for two types of CAD environments; Xilinx ISE 5.x and Celoxica DK1. The components can be employed to implement both application-specific blocks from the selected area (mainly from the scope of combinatorial computations) and a number of interfaces that are very useful for interaction and data exchange with devices attached to a FPGA, such as LCD and touch panels, bus controllers, etc. The designed modules can be easily integrated into any application-specific digital system and used for visualizing the results, fast data transfer, debugging of internal sub-circuits, etc. They were constructed in such a way that their functionality can be either fixed or modifiable (both statically and dynamically). The latter capability was provided with the aid of re-loadable RAM-based blocks. To illustrate the capabilities of the tools suggested, four design examples are discussed. Additional materials for this paper are available in the form of a number of tutorials and projects for FPGAs that can be accessed through the Internet.

1. Introduction

Today field programmable logic devices in general and FPGAs in particular are considered to be an alternative to ASICs, and they have already been very successfully used in a large number of practical applications, such as co-processors for general-purpose computers, problem-oriented digital systems, embedded controllers, and so on. A number of prototyping boards that provide support for various experiments with FPGA-based circuits have been fabricated. These permit digital systems to be implemented in FPGA and to interact with both onboard microchips and devices (such as static RAM, micro controllers, etc.) connected through expansion headers. This significantly simplifies the design of new FPGA-based applications and allows the development lead time to be shortened. Very often we can take full advantage range of the hardware capabilities of the prototyping boards if the relevant libraries are provided. In particular these libraries support interfacing FPGAs with external devices.

The paper discusses one approach to the organization of such libraries and suggests a number of reusable descriptions for HDL (VHDL) and system-level specification (Handel-C) design flows. This is especially interesting for experiments and comparisons that allow the most appropriate approach for the design of a particular system to be properly selected. The RC100 prototyping board with FPGA XC2S200 (Spartan-II family of Xilinx), the Celoxica DK1 environment [1] and the Alpha Data ADM-XPL PCI board [2] containing FPGA XC2VP7 from the Virtex-II Pro family were considered as primary tools for projects based on Handel-C. The development system TE-XC2Se with FPGA XC2S300E (the Xilinx Spartan-IIE family) from Trenz electronic [3] and Xilinx ISE 5.2 were chosen for the VHDL-based design flow. The hardware modules discussed allow the FPGA mentioned above (as well as other FPGAs for which the CAD systems can be employed) to be linked with a number of external devices. These modules can be considered to be components of a library that permits the development of FPGA-based systems communicating with computers, external memory and peripheral devices. They provide many very useful facilities, such as support for various interfaces; interaction with touch panels, graphical and textual LCDs; communications with a mouse, a keyboard, and VGA monitors; data exchange between prototyping boards; links with segment displays, pushbuttons, dipswitches, LEDs, and many others. The suggested tools enable designers to concentrate their efforts on the particular problem that has to be solved and to involve the components mentioned above just to provide such helpful facilities as data exchange, the visualization of final and intermediate results, debugging capabilities as well as supporting experiments, comparisons, etc.

This paper is organized in five sections. Section 1 is this introduction. Section 2 presents a general description of the proposed design tools. Section 3 is dedicated to reusable interface modules that can be employed for interactions between FPGAs and external devices. Section 4 discusses some examples of practical design and illustrates the applicability and effectiveness of reusable components. The conclusion is in section 5.

2. General characteristics of the design tools

In general the reusable circuits suggested can be divided into two groups. The first group provides for interaction with various external devices that support data input and output. Some of such devices are indicated in fig. 1 (see the left bottom and the right bottom rectangles). The second group is composed of application-specific circuits.

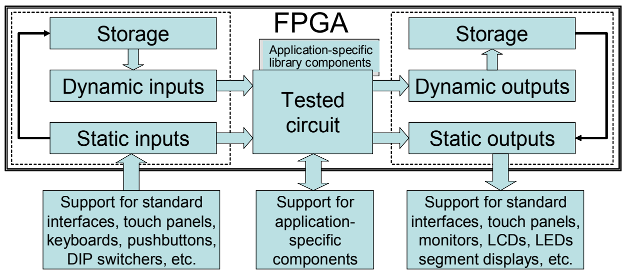
FPGA
Storage
Dynamic inputs
Static inputs
Application-specific library components
Tested circuit
Storage
Dynamic outputs
Static outputs
Support for standard interfaces, touch panels, keyboards, pushbuttons, DIP switchers, etc.
Support for application-specific components
Support for standard interfaces, touch panels, monitors, LCDs, LEDs segment displays, etc.

Fig. 1. General use of the library components

The circuits can be employed as either components of more complicated devices implemented in an FPGA or as auxiliary blocks that simplify debugging, testing, and experiments with FPGA-based digital systems. In the second case they can be connected to another device (see the block “tested circuit” in fig. 1) in order to provide interfaces with a variety peripheral equipment for supplying input data (see the rectangle “static inputs” in fig. 1) to the tested circuit and displaying output data (see the rectangle “static outputs” in fig. 1) generated by the circuit. Fig. 1 shows that input data can be packaged in order to build a control sequence (see the rectangle “dynamic inputs” in fig. 1), which represents a test bench. Outputs from the circuit that are generated in response to the control sequence (see the rectangle “dynamic outputs” in fig. 1) can be saved in storage for future examination and analysis.

Fig.2 demonstrates two potential applications of the library components. The first example (see fig. 2,a) enables an application-specific circuit to communicate with a touch panel, such as EA KIT 240-7 that is fabricated by Electronic Assembly [4]. The second example illustrates how an LCD (such as LCD module 4x20 EA DIP204-4 [4]) can be controlled and various kinds of data can be sent to this LCD.

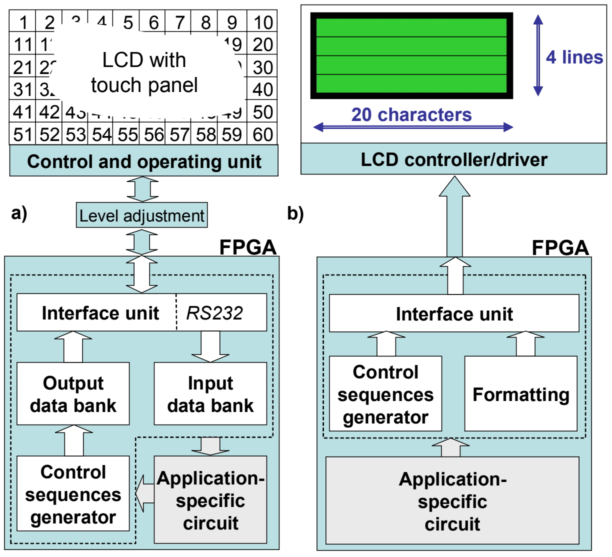

Fig. 2. Library components, which provide data exchange with a touch panel (a) and an LCD (b)

There are 4 reusable blocks in fig. 2,a. The first is an interface unit that provides data exchange in accordance with a given protocol. The RS232 serial interface is used for the example here. When interaction with a different type of external device is required, the interface unit can be replaced with another block from the library that supports the necessary data exchange. Two data banks for input and output are considered to be buffers to store the receiving/sending data. The control sequences generator permits a high-level interface to be provided. It supports a number of high-level instructions available for the LCD with touch panel, such as drawing graphical shapes, constructing menus, loading bitmaps, scaling, etc. (see [4] for details). Tutorial 7 from [5] gives many useful examples that are illustrated by ISE 5.2 projects for FPGAs, which demonstrate how to interact with the device EA KIT 240-7 [4].

Note that software tools available for the product [4] allow a variety of control sequences to be tested using a general-purpose programming language (such as C/C++). In this case interaction with an LCD can be examined through the PC serial port and then a set of LCD instructions can be generated from a C/C++ program and saved in RAM blocks belonging to the control sequences generator (see fig. 2,a).

There are 3 reusable blocks in fig. 2,b. The first is an interface unit. The second block builds control sequences for the LCD, such as those used for scrolling, changing fonts, inverting text, etc. (see [4] for details). The last block carries out any required formatting of the text, i.e. it specifies different types of data that should be displayed. For example, it will select positions on the LCD for displaying fixed strings and variable data that is received from the application-specific circuit. Tutorial 9 that is available in the public domain of [5] gives a number of useful examples, which are augmented by ISE 5.2 projects for FPGAs that demonstrate how to interact with the device [4] and the LCD panel available on the board [3].

Fig.3 demonstrates two application-specific library components. Many digital systems require the execution of various operations over Boolean and ternary matrices (see, for example, [6]). Such systems deal with two-dimensional arrays with elements having either two (0 and 1) or three (0, 1, and don't care) values. As a rule, access has to be provided to individual row/columns of the matrices and this requires dual-address storage.

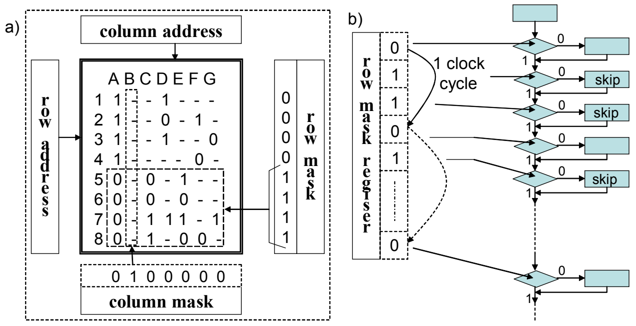

Fig. 3. Application-specific library components: dual-address storage (a) and a selection unit that skips unnecessary rows/columns (b)

Fig. 3,a depicts the respective reusable block, which includes RAM-based memory, mask registers that permit some rows and columns to be excluded from the matrix to select a minor, and columns/rows registers that provide dual-address access. Fig. 3,b demonstrates the functionality of a reusable circuit that selectively generates the necessary RAM addresses. The latter allows the rows/columns that are not required for the minor that is being extracted during the current step to be skipped. Let us assume that a matrix is composed of 8 rows *1,2,…,8* and 7 columns *A,B,…,G* (see fig. 3,a). We want to remove all the rows that have "0" in column *A* and all the columns that have just don't care values ("-"). Fig. 3,a shows the contents of the mask registers for this case. Suppose that all the rows that are indicated by ones in the row mask register have to be skipped. Fig. 3,b demonstrates how the relevant reusable block implements this requirement.

The mentioned above and the other designed library components can be employed for both practical design and educational purposes. For example, a number of tutorials that explain how to work with various tools available for FPGAs and how to use these tools to design different types of FPGA-based circuits are available in [5]. They are organized in such a way that any circuit being tested is considered to be the core of some working environment and the remainder of the working environment provides for interaction of the circuit with the external world. Fig. 4 demonstrates the basic idea of the tutorial 5 [5], which explains how various synthesizable constructions of VHDL for ISE 5.2 can be used. The tutorial lists these constructions alphabetically (such as A – *aggregates*, A – *architecture*, etc. up to W - *while*) and describes examples that can be tested in ISE 5.2 with the aid of the circuit in fig. 4 in such a way that input data can be entered from peripheral devices and the result can be visualized on peripheral devices. Interfaces and communications with these devices are provided through the designed library components.

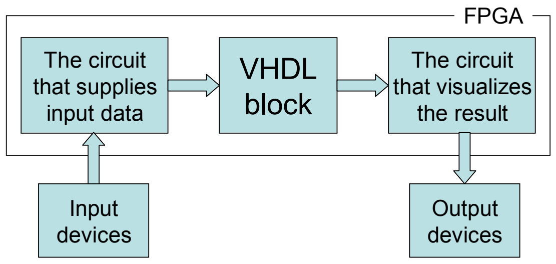
FPGA
The circuit that supplies input data
VHDL block
The circuit that visualizes the result
Input devices
Output devices

Fig. 4. Using the designed reusable components for testing different VHDL constructions

A similar technique was used for the other tutorials available in [5]. Note that the majority of the modules are based on distributed and dedicated FPGA RAM-blocks that can be programmed either before or during run-time. This permits both static and dynamic changes to module's functionality.

At the beginning of this section, we mentioned that two groups of library components have been suggested (see fig. 1-3). Components from the first group were described in VHDL and tested within ISE 5.2. They can be reused as either fragments of VHDL code that are inserted in the design (much like we invoke Xilinx ISE synthesis templates) or library modules that are attached to the design. Components from the second group provide support for handling Boolean and ternary matrices. They were implemented in VHDL and in Handel-C in such a way that the respective code can be copied to any other project. The Celoxica DK1 design suite [1] was used for synthesis from Handel-C. The results are presented in EDIF format. Further processing of the EDIF files was carried out in the ISE 5.2 environment. The circuits were tested for the Xilinx Spartan-II/Spartan-IIE/Virtex-II Pro families of FPGAs that are available on the prototyping boards [1-3]. Details of many projects using these components can be found in [5] and they augment the material presented in this paper.

3. Interface modules

The designed interface modules support serial (type 1) and parallel (type 2) modes. For each type two different circuits have been constructed that provide fixed and modifiable functionality of the modules respectively.

Modules with fixed functionality were mainly described using behavioral VHDL (ISE 5.2) and state transition diagrams constructed using Xilinx StateCAD software. Modules with modifiable functionality were built on the basis of RAM/ROM blocks, parameterizable structural VHDL code with *generic* and *generate* statements and reprogrammable finite state machines (RFSM) [7,8]. Fig. 5 shows a simple example demonstrating the capabilities of a reprogrammable circuit that receives data from an RS232 serial interface (similar circuits were used for communicating with a mouse and a keyboard).

The circuit contains a RAM-based RFSM, a data register, and a baud rate circuit. The number of stop bits (1 or 2) for *Wait* state (see node 1), the number *n* of data bits (see node 2) and a presence or an absence of a *parity bit* (see node 3) can all be set by varying RFSM functionality. Trivial changes in the algorithm enable handshaking to be implemented through the use of request to send (RTS) and clear to send (CTS) bits. The functionality of the RFSM can be modified by reloading the RAM blocks that are used as basic components of the RFSM combinational circuit. Methods [7] were applied for such purposes. The circuit "*Baud rate*" takes an input from a clock generator and sets up the required output frequency that can be changed by modifying a division coefficient that is read from RAM.

The considered method, which provides for modification of a circuit, has some advantages compared to traditional devices, such as a UART. Firstly, this method can be used for any similar circuit that supports either a standard or a customized (application-specific) interface. Secondly, it only implements the functionality that is required for a particular application. This permits the necessary hardware resources to be reduced. And finally, the mechanisms considered that provide for modification of the circuit's functionality are very well suited for recent architectures of FPGAs that contain a large number of built-in memory blocks supporting the dual-port access that is needed for reprogrammability.

A serial port was used for data exchange with the PC and for an interaction with some peripheral devices, such as that shown in fig. 2.

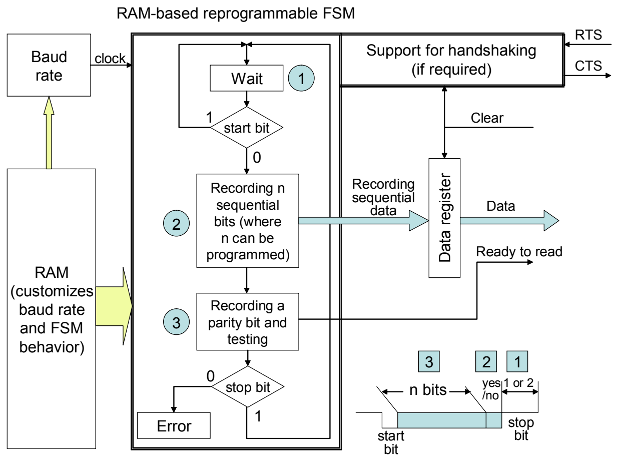
RAM-based reprogrammable FSM
Baud rate
clock
Wait
1
start bit
1
0
Recording n sequential bits (where n can be programmed)
2
Recording a parity bit and testing
3
stop bit
0
1
Error
RAM (customizes baud rate and FSM behavior)
Support for handshaking (if required)
RTS
CTS
Clear
Recording sequential data
Data register
Data
Ready to read
3
2
1
n bits
yes /no
1 or 2
start bit
stop bit

Fig. 5. Using a reprogrammable circuit for implementing serial interface

Fig. 6 shows a simple example demonstrating the capabilities of a reprogrammable circuit that controls a textual LCD panel (see also fig. 2,b). The RFSM enables the circuit to be customized for any LCD with a different number of lines and/or characters per line, specific control sequences, etc. Depending on the application, any mode can be used, such as write to LCD, read from LCD, and write/read to/from LCD. This is achieved either through direct data transfer from FPGA circuit to bi-directional LCD bus or with the aid of an intermediate dual-port RAM that buffers the data. In the last case the first RAM port (see fig. 6) is used for interaction with an application-specific circuit implemented in FPGA. The second port provides communication with an LCD through a bi-directional data bus. The RFSM can control this bus and permits various preliminary saved in RAM control sequences for the LCD to be specified. For example, we can provide for shifting data on the LCD panel, scrolling lines, etc. Note that the same circuit, depicted in fig. 6, can be used for interactions with the LCD panel available on the TE-XC2Se board [3] as well as with any other externally connected LCD panel. Tutorial 9 from [5] includes some ISE 5.2 projects for LCDs.

A similar technique was used for the design of other interface circuits. They provide data exchange between FPGAs and a variety of external devices. For example, two prototyping boards [1,3] were connected through expansion headers. For such purposes a special parallel interface has been implemented. This permits resources available on the boards and peripheral equipment that might be connected to the boards to be combined [1,3]. As a results more complicated devices can be constructed. Besides, this is the easiest way to test projects that were created using tools such as Handel-C system-level specification language [1] and CAD software (such as ISE 5.2) that supports a traditional HDL-based design flow.

A parameterizable VHDL code for a RFSM can be found in [8]. Tutorial 8 available from [5] includes ISE 5.2 projects for a RFSM that were tested with simple datapath.

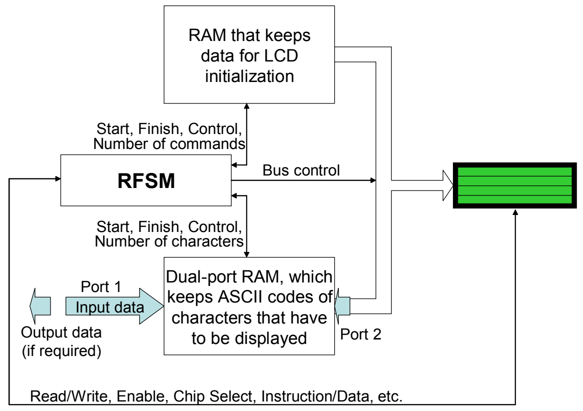
RAM that keeps data for LCD initialization
Start, Finish, Control, Number of commands
RFSM
Bus control
Start, Finish, Control, Number of characters
Port 1
Input data
Output data (if required)
Dual-port RAM, which keeps ASCII codes of characters that have to be displayed
Port 2
Read/Write, Enable, Chip Select, Instruction/Data, etc.

Fig. 6. Using a reprogrammable circuit to implement a parallel interface with an LCD

4. Design examples

This section describes four projects for the DK1 and ISE 5.2 environments that demonstrate how the reusable components considered above can be involved in the design. The first three projects were developed in Handel-C and they demonstrate an arithmetical circuit (RC100 prototyping board [1]) and two combinatorial processors (RC100 prototyping board [1] and ADM-XPL PCI board [2]). The fourth project for ISE 5.2 allows a recursive sorting procedure to be performed (TE-XC2Se prototyping board of Trenz electronic [3]). All these projects are available in [5].

4.1. Design of an arithmetical circuit in Handel-C

The circuit implements four arithmetical operations (+: addition, -: subtraction, x: multiplication, and /: division) and interacts with a VGA monitor and a mouse connected to the FPGA. The functionality was tested in a Spartan-II XC2S200 FPGA (the RC100 board).

The Handel-C specification contains the following blocks (see fig. 7):

The *Initialization* block permits initial values to be assigned to the variables that are used. After that the 7 blocks shown as rectangular nodes in fig. 7 will be executed in parallel. This is supported by a Handel-C *par* statement.

The *Data processing* block calculates the result of the selected arithmetical operation.

The *Preparing data to display* block interacts with the block *Defining color for each pixel* and supplies two operands, an operator, and the result.

The *Moving screen windows* block permits a dialog box to be moved using the mouse drag and drop option. Thus, interaction with a mouse driver has to be provided. The dialog box displays the operands, the arithmetic operator, and the result of the operation. Fig. 8 illustrates how the operands (234 and 132), the operator (x - multiplication), and the result (30888) will appear on the monitor screen. The dialog box containing this information can be moved to any position within the working area of the screen.

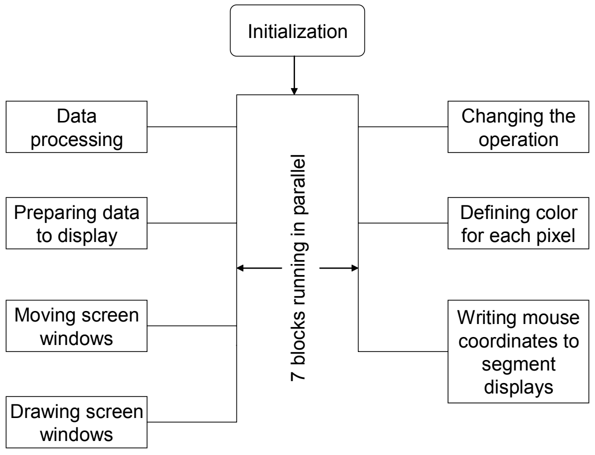
Initialization
Data processing
Preparing data to display
Moving screen windows
Drawing screen windows
7 blocks running in parallel
Changing the operation
Defining color for each pixel
Writing mouse coordinates to segment displays

Fig. 7. Parallel blocks of Handel-C specification

The *Drawing screen windows* block communicates with the block *Defining color for each pixel* to provide for data visualization, i.e. to display the dialog box (see fig. 8).

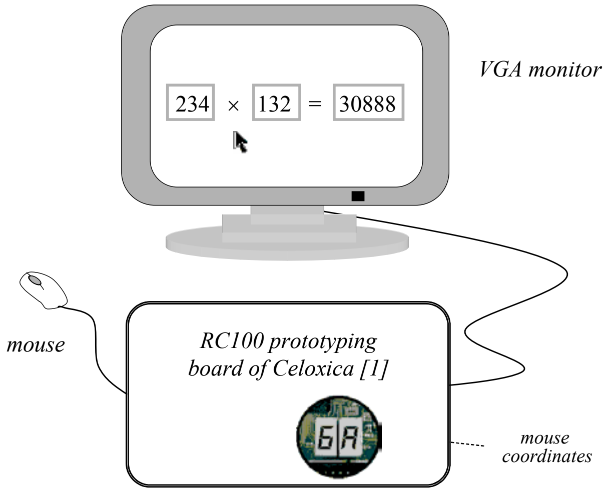

Fig. 8. Experiments with the arithmetical circuit designed from a specification in Handel-C

The *Changing the operation* block enables the desired operation (+, -, x, /) to be selected. This is done by moving the mouse cursor over a sign position on the screen and selecting an operation by clicking the left mouse button. The button permits operations to be switched in a cyclical sequence: 1) +; 2) –; 3) x; 4) /. Thus, again it is necessary to interact with the mouse driver.

The *Defining color for each pixel* block constructs a 24-bit RGB vector for each pixel, which is composed of three 8 bit fields for each color (*R – red*, *G – green* and *B – blue*) and establishes links with the VGA monitor driver.

The *Writing mouse coordinates to segment displays* block communicates with the mouse and segment display drivers and displays the horizontal and vertical positions of the mouse (see fig. 8, where the segment displays show the less significant digits *6A* of the mouse coordinates in hexadecimal notation, where the value *6* gives a horizontal and *A* - a vertical coordinate). For the considered example Celoxica mouse and VGA monitor drivers [1] were used. However the circuit can also interact with the described above reusable components to provide interfaces with other peripheral devices.

This is a very simple example but it would become much more complicated if the drivers that establish the interfaces for data input and output were not available.

4.2. Design of a combinatorial processor for a stand-alone board

The second project is to discover a minimal column cover of an arbitrary Boolean matrix, i.e. to find a minimal number of columns that have at least one value "1" in each row of the given matrix. The project implements the algorithm [9] and the hardware architecture described in [6].

The basic operations of the algorithm involve the application of so-called reduction and selection rules. The reduction rules enable an initial matrix and intermediate matrices (that appear on each iteration through the algorithm) to be simplified. For the covering problem the following reduction rules were applied: 1) if for $i \neq j$ row_i & row_j = row_j then row_i is removed from the matrix; 2) if for $i \neq j$ $column_i$ & $column_j$ = $column_i$ then $column_i$ is removed from the matrix; 3) if any column contains just "0" values it is removed from the matrix; 4) if there is a row that does not have any "1" value then covering cannot be found; 5) if a row has just one "1" value then the respective column must be included into the covering; 6) if all rows have more than one "1" value then the first row from the top of the matrix that contains a minimum number of ones is selected. For this row it is necessary to examine all possible branches and the number of such branches is equal to the number of "1" values in the row.

Analysis of different combinatorial search algorithms has shown that the following four types of application-specific blocks are required: storage for matrices, a stack memory, a functional unit for operations over discrete vectors (i.e. rows/columns of matrices), and a control circuit. Fig. 9 shows how different hardware resources were used and lists the basic architectural blocks of the processor that were described in Handel-C. All these blocks are parameterizable and the respective Handel-C code can be reused for similar applications.

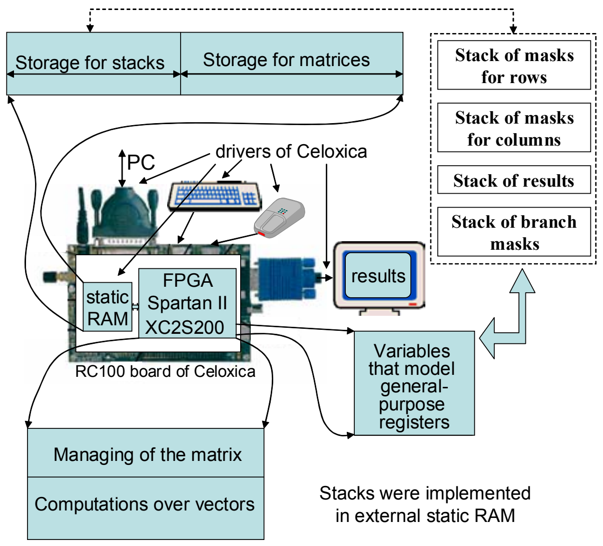

Fig. 9. Reusable application-specific blocks for the combinatorial processor

The algorithm requires the execution of operations over individual rows and columns of the matrix. Thus dual-address access to the matrix (see section 2 and fig. 3) permits these operations to be accelerated significantly. In order to minimize the required hardware resources just one matrix is kept in memory. Extracting any minor is achieved by masking some rows and columns. Thus the techniques considered in section 2 (see fig. 3) become very helpful because it significantly accelerates traversal and examination of rows/columns that are not masked. There are two blocks of static RAM available on the RC100 board (256K x 36 bit each). Storage for matrices was allocated in the first block and the second block was used as a memory for a VGA monitor connected to the board (see fig. 9). Different stacks required for the chosen architecture [6] of the processor were also implemented in static RAM (in the first block). The control circuit is hidden in Handel-C code because this language uses pre-defined specific synchronization mechanisms [1]. Reusable VHDL code for such circuit is considered in [5] (see tutorial 10). Computations over vectors are carried out through logical operations available in Handel-C. An initial matrix can be loaded either from the PC through the parallel port or from a keyboard. The results are displayed on a VGA monitor.

An instance of the covering problem is presented in fig. 10. A graph in fig. 10 explains the sequence of steps that have to be carried out. The rows are indicated by digits 1-10 and the columns by letters A-I. If any element is removed, it is crossed by a horizontal line. Circles show an initial matrix **U** and how the matrix columns have been selected step by step. Rectangles indicate the branch points. This task may require the examination of more than two alternatives for any branch point. For this reason a stack of branch masks has to be implemented (see fig. 9). This enables the branches of the decision tree that have already been examined to be avoided. Stop points appear when the number of columns in the covering obtained at the current step is equal to any previously discovered covering. In this case it cannot be improved upon and we can terminate traversing this branch.

$$\mathbf{U} = \begin{array}{ccccccccc|l} A & B & C & D & E & F & G & H & I & \\ \hline 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 2 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 3 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 4 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 5 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 6 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 7 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 8 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 9 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 \end{array}$$

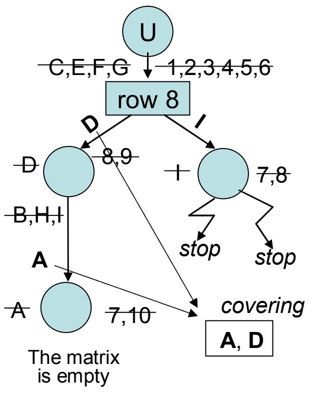

Fig. 10. Execution of the considered combinatorial search algorithm

Note that supplementary blocks (such as those that provide visualization of the results, support for debugging and experiments, etc.) consume lots of hardware resources, which could otherwise be used for extending the dimensions of the considered problem. For many practical applications it is reasonable to separate the primary and supplementary resources in such a way that one FPGA is solely responsible for the problem that has to be solved and the other for all the required auxiliary functions. The latter could be displaying and scrolling matrices, highlighting (or selecting by color) elements considered in any intermediate step (such as any minor), visualizing the contents of stacks, checking any currently executing operation, and many others. Hardware components that execute these operations can also be included into the library, which on the one hand is application-specific, but on the other hand can be reused for a large number of practical applications that handle discrete matrices. Thus the tools considered in sections 2, 3 can be employed directly because they provide data exchange between primary and auxiliary blocks, interactions with peripheral devices and support for application-specific operations. It is important that this functional separation enables any available tools for the design of FPGA-based circuits to be combined. Indeed, the primary block implemented in a separate FPGA might be designed with the aid of a system-level specification language (such as Handel-C) and the supplementary tools can be developed in a hardware description language.

4.3. Design of a combinatorial processor for PCI board

The Virtex-II Pro family FPGA XC2VP7 (ADM-XPL PCI board of Alpha Data [2]) is much more powerful than FPGAs of Spartan series and it permits the dimensions of the matrices to be significantly increased. Besides, data exchange with a PC through a PCI bus makes it possible to increase the speed of execution of complex algorithms that require distribution and parallel execution of different sub-algorithms in general-purpose (such as PC) and application-specific (such as combinatorial accelerator) processors.

Fig. 11 depicts the basic architecture of the system that has been implemented. Compared to the previous project (see sub-section 4.2), the Handel-C code for the combinatorial processor considered only changes the sizes of the objects (variables and arrays) that are involved. A set of C++ classes and functions provide auxiliary operations supporting input/output data for matrices, debugging facilities and a user-friendly graphical interface realized through dialog boxes for working with the combinatorial processor. These classes and functions are very helpful because the ADM-XPL PCI board is installed inside the PC and does not provide communications with additional peripheral devices (except for the RS232 serial port). Thus the C++ library is responsible for such operations.

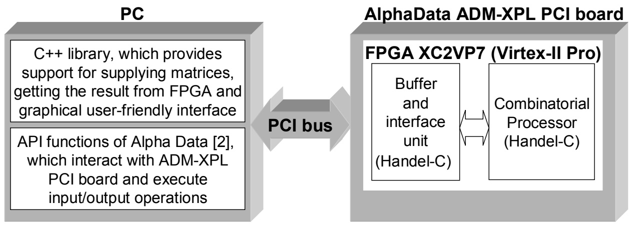
PC
C++ library, which provides support for supplying matrices, getting the result from FPGA and graphical user-friendly interface
API functions of Alpha Data [2], which interact with ADM-XPL PCI board and execute input/output operations
PCI bus
AlphaData ADM-XPL PCI board
FPGA XC2VP7 (Virtex-II Pro)
Buffer and interface unit (Handel-C)
Combinatorial Processor (Handel-C)

Fig. 11. Software/hardware tools for the combinatorial processor

API functions supplied by Alpha Data [2] are used to drive the board and to provide data exchange between the PC and the combinatorial processor through PCI.

Two new blocks that support communications of the FPGA with external hardware were designed in Handel-C. The first block establishes the interface with the ADM-XPL bus. The second block buffers the initial matrices and stores the result to make it ready for transmission via the PCI bus. Obviously these blocks can also be considered as reusable components for future applications.

Initial matrices have pre-defined dimensions and can be either randomly generated using the C++ program with a given percentage of zeros/ones or loaded from a file. The latter permits experimentation using existing benchmarks.

Reusing pre-designed fragments of Handel-C code made it possible to develop, implement and test a quite complicated digital system with additional software support in less than two weeks.

Note that each reusable component can be considered from two different points of view. Firstly it has a particular specification such as in Handel-C or in VHDL. Secondly it has a unique reusable structure, which, as a rule, can be described using any relevant tool (language). A reusable (particular) code can only be compiled (synthesized) in environments that permit this code to be used. On the other hand any reusable architecture (structure) is independent of any environment. In order to prove reusability of the considered application-specific circuits (such as that shown in fig. 3) at a structural level, a combinatorial accelerator for solving the Boolean satisfiability problem has been described in VHDL, implemented in a Virtex-EM family FPGA and tested [12]. The results obtained show that the application-specific blocks mentioned above are actually reusable for a variety of digital systems that require handling of discrete matrices.

4.4. Recursive sorting

The last project implements in hardware two recursive procedures considered in [10], where they were described in C. Similar C++ code can be found in tutorial 10 available from [5]. The first procedure receives incoming external data (such as integers, strings, etc.) and constructs a binary tree that simplifies many computations, such as ordering the data, finding minimum or maximum values, etc. The second procedure sorts data on the basis of the binary tree. The hardware architecture that might be used for such purposes was proposed in [11]. Tutorial 10 in [5] extends the architecture [11] and gives detailed explanations of the algorithms and their hardware implementations in ISE 5.2 with a number of ISE 5.2 examples (mainly in VHDL).

The project includes VHDL code for a hierarchical FSM (HFSM) described in [11] that executes recursive modular algorithms and carries out all control functions for the two procedures considered above. This component is obviously reusable and it is needed for solving different tasks such as those formulated on binary trees (see, for example, [10]), composed of relatively autonomous modules, etc. ISE 5.2 projects that include HFSM can be found in tutorial 10 [5]. Obviously all the debugging tools discussed previously can be used for this project. Some of them are invoked in the tutorial 10 [5].

5. Conclusion

The paper discusses reusable modules for the design and debugging of FPGA-based digital circuits. They permit the exchange of data between FPGAs and a personal computer through various interfaces, interaction with external memories, data visualization, etc. Two general types of reusable components have been described. They provide support for debugging and reflect the specifics of some selected applications.

The library modules that have been developed can easily be integrated into any application-specific system that is going to be implemented on the basis of FPGA. Four design examples demonstrating such facilities have been described.

This work was partially supported by the grants FCT-PRAXIS XXI/BD/21353/99 and POSI/43140/CHS/2001.

6. References

[1] Handel-C, DK1, RC100. [Online]. Available: http://www.celoxica.com/.

[2] Alpha Data, [Online]. Available: http://www.alpha-data.com.

[3] Spartan-IIE Development Platform, [Online]. Available: www.trenz-electronic.de.

[4] Electronic Assembly products, [Online], Available: http://www.lcd-module.de.

[5] http://webct.ua.pt, "2 semester", the discipline "Computação Reconfigurável", public domain is indicated by the letter "*i*" enclosed in a circle. Login and password for access to the protected section can also be provided (via e-mails: skl@ieeta.pt, iouliia@det.ua.pt).

[6] V. Sklyarov and I. Skliarova, "Architecture of Reconfigurable Processor for Implementing Search Algorithms over Discrete Matrices", Proceedings of ERSA'2003, Las Vegas, USA, 2003.

[7] V. Sklyarov, "Reconfigurable models of finite state machines and their implementation in FPGAs", Journal of Systems Architecture, 2002, 47, pp. 1043-1064.

[8] V. Sklyarov and I. Skliarova, "Design of Digital Circuits on the Basis of Hardware Templates", Proceedings of ESA'2003, Las Vegas, USA, 2003.

[9] A.D. Zakrevski, *Logical Synthesis of Cascade Networks*, Moscow: Science, 1981.

[10] B.W. Kernighan and D.M.Ritchie, *The C Programming Language*, Englewood Cliffs, NJ: Prentice-Hall, 1988.

[11] V. Sklyarov, "Hierarchical Finite State Machines and Their Use for Digital Control", IEEE Transactions on VLSI, Vol. 7, No 2, June, 1999, pp. 222-228.

[12] I. Skliarova and A.B. Ferrari, "A hardware/software approach to accelerate Boolean satisfiability", Proc. of IEEE DDECS'2002, Brno, Czech Republic, pp. 270-277.