Real-Time LUT-Based Network Topologies for Dynamic and Partial FPGA Self-Reconfiguration

Michael Huebner, Tobias Becker, Juergen Becker
University Karlsruhe (TH), Germany
http://www.itiv.uni-karlsruhe.de/
{huebner, beckert, becker}@itiv.uni-karlsruhe.de

Abstract

Xilinx Virtex FPGAs offer the possibility of dynamic and partial run-time reconfiguration. If a system uses this feature the designer has to take care, that no signal lines cross the border to other reconfigurable regions. Traditional solutions connecting modules on a dynamic and partial reconfigurable system use TBUF elements for connection and separation of the functional blocks. While automalically placing and routing the design, the routing-tool sometimes uses signal lines which cross the module border. The constraints given by the designer are ignored. To solve this problem we use slices instead of TBUF elements which leads to a benefit by using an automatic modular design flow. This paper gives an overview of the used technique and the complete system on a Xilinx XC2V3000 FPGA.

Categories and Subject Descriptors

B.4.3 [Input/ Output and data communications]: Interconnections (Subsystems) – Synchronous operation, interfaces, physical structures, topology

General Terms

Design

Keywords

Dynamic partial reconfiguration, Virtex

SBCCI'04, September 7–11, 2004, Pernambuco, Brazil.

1. Introduction

Xilinx Virtex FPGAs offer the possibility of dynamic and partial run-time reconfiguration. New approaches use this feature by outsourcing configuration data which makes it possible to use FPGAs with smaller configuration memory and consequently smaller chip size. Thus it is possible to save costs and reduce power consumption because not actually used modules of a complete system do not allocate configuration memory and corresponding power consuming hardware [1]. Nevertheless power dissipation during reconfiguration has to be considered [2]. [3] raise this issue with the main focus on energy saving and basis for new design methodology. The designer of a dynamic and partial reconfigurable system on an FPGA must be sure, that no signal lines of a module cross the border to another functional block. During reconfiguration such a signal line might cause a malfunction or a short-circuit, which destroys the FPGA. Because of this it is necessary to implement interfaces which are used as fixed routing resources. These interfaces, called BUS Macros, are placed in the same position for each functional block. Connecting the modules with signal lines on the same position grants the option, to substitute a module by another. Traditional techniques using TBUF elements as BUS Macro, are able to separate the modules and provide a resource for automatic routing. Unfortunately the autorouter of the Xilinx ISE design software ignores the constraints which disallow the use of routing resources across the border of a module area. This leads to a very time consuming and difficult manual debugging which is not acceptable. A more detailed schematic of the traditional TBUF interface is shown in figure 1.1. To send a signal from left to right, for example the LE(0) input of the Tristate gate is on enable level, while RE(0) switches the second gate into Tristate mode. The signal of LI(0) is now connected to the Output(0) line which connects the two sides. In the opposite direction communication is possible by changing the mode of the Tristate gates. For

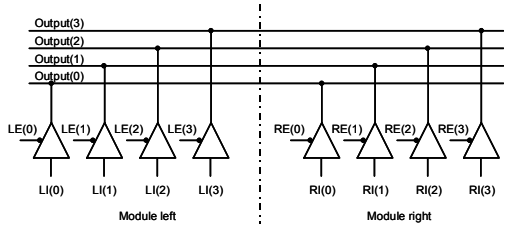

Figure 1.1 Tristate gates of CLB interface

more information about the design section 0 describes the target hardware and gives an overview of the complete system. The following subchapters 2.1 to 2.3 contain the description of the used modules in detail. Section 3 shows the implementation of the complete system after place and route procedure. Real-Time aspects are described in section 0 and in section 5 the conclusion and a preview of future work is described.

2. Target Hardware and Reconfiguration Techniques

The method which is introduced in this paper was used on a dynamic and partial reconfigurable system implemented on a Xilinx Virtex-II FPGA (XC2V3000). To get an overview of the complete system, figure 2.1 shows the schematic picture with all used components.

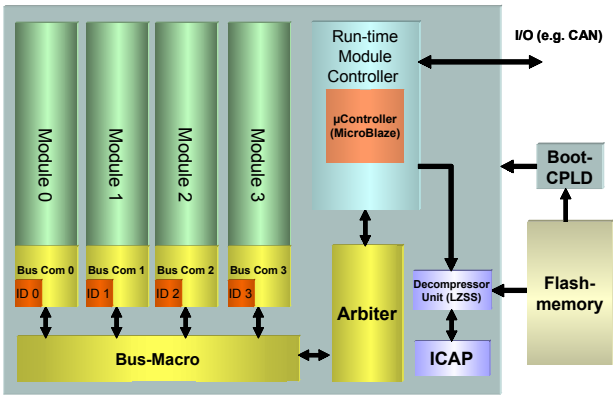

Figure 2.1 Dynamic reconfigurable system

As a controlling element, the Xilinx MicroBlaze [11] Soft-Core controller was used. The internal block RAM elements of the FPGA contains the run-time system which controls the activities inside the complete system. A detailed description of the run-time system and the tasks of the MicroBlaze controller can be found in [7]. Via an arbiter the controller is connected to the bus macro. The arbiter controls the dataflow for sending or receiving messages to or from the modules. The bus was designed for our test scenario for automotive control functions but it also can easily be adapted to other usage. To connect the modules to the FPGA bus system, an interface module called BUS-COM Module was designed. This module enables a unified connection to functional modules of other applications. Designers only have to adapt their implementations to the specifications. The BUS-COM Module receives signals from the arbiter to connect data and signal lines of the modules to the bus system. The BUS-COM module also contains the slot identification number which allows to assign a unique address for each module. This id number won't be overwritten if reconfiguration occurs. The BUS-COM Module is used like a connector for multi-purpose applications which are designed in the slot-modules.

The FPGA of the Virtex-II series grants the option to use the internal configuration access port (ICAP) [4] which allows a self-reconfiguration without demand of external wiring. The ICAP interface provides nearly the same features as the SelectMap interface for external usage. Figure 2.2 shows the schematic of the modules used for reconfiguration procedure. Partial bitstreams are stored in an external flash memory. The run-time system sends the start address and endadress to the decompressor module. Afterwards a start command enables the decompressor to read the compressed bitstream data from the flash, to decompress it and write it through the ICAP interface. While this process is running, the controller is able to work on further tasks. A signal from the decompressor reports the end of the configuration process. The complete system is connected via a CAN interface to its environment. The interface of the decompressor is designed in a way that it can easily be integrated into an FPGA based run-time system for partial reconfiguration. The decompressor consists of three parts: A controller, a RAM block and a module that changes the bit width of the output data. A detailed description of this decompressor system can be found in [5].

If more communication channels are needed it is possible to implement more CAN controllers or other interfaces to the FPGA. A detailed overview of the system can be found in [2].

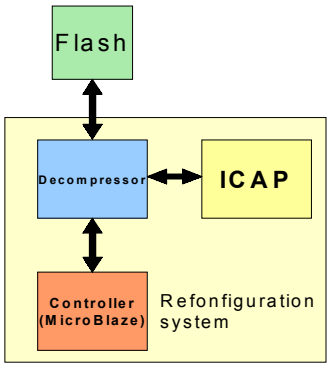

Figure 2.2 Modules for reconfiguration using ICAP interface

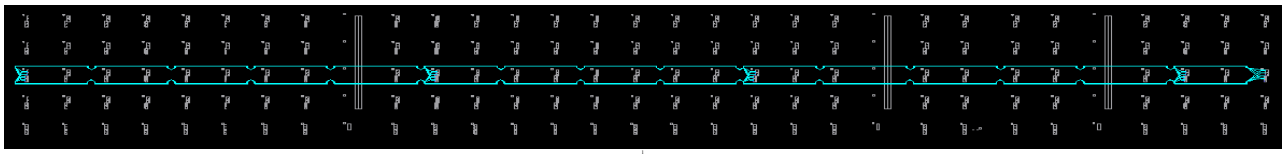

Figure 2.3 Complete Bus Macro for all slots

2.1. BUS-COM Module

To provide the stand-alone functionality of the modules on the bus, the BUS-COM module is necessary. The main task of this module is to buffer input data, signalize the arbiter the occurrence of new output data from the functional-modules and transmit the write enable signal from the arbiter. The BUS-COM module contains a module id input, which is connected to pre-programmed flip-flops. These flip-flops are not overwritten during the reconfiguration and thereby the modules have a unique address. If a module is addressed, the data word of the bus is wrapped and connected to the module's input. This data is stored until the module is addressed again. If the output data of a module changes, the BUS-COM module enables the request output. The arbiter recognizes this request and enables the module to send data via the bus by addressing it. Afterwards the request signal is disabled.

Additionally, the BUS-COM module contains a command decoder which allows a context save or load beside the normal bus communication. For that purpose a demultiplexer is switched to connect context in- or output signals to the bus system. This feature is used to store or load necessary data of the modules before substituting them by reconfiguration, or reload data after successful configuration on the slot. The commands for context load- or save are connected via special bus signals.

2.2. Multifunctional Arbiter

The arbiter is responsible for controlling the incoming bus. Outgoing module data is solely synchronized by a register. There are no control functions necessary to transmit both normal module data and context data.

The main task of the arbiter is to execute the requests of the modules in a reasonable sequence and to transmit the received data to the run time system together with the module address. For that purpose the arbiter enables writing permission to the bus for the module which has been requesting. Hereupon the module data arrives two clock cycles later at the arbiter. The module address which has been delayed for two clock cycles is now joined together with the data word and transmitted to the run time system. If the run time system detects the need for a reconfiguration it sends an instruction word to the arbiter to interrupt the normal routine. Then the arbiter inquires which modules are busy by polling each one. By the time the state of each module is available the arbiter sends a status word to the run time system. The run time system is now able to choose an idle module for replacing it. Afterwards the arbiter returns to the normal routine.
In order to save the context of a module the normal routine has to be interrupted as well. Thereupon the context data is transmitted cohesively to the run time system. After finishing the context save the arbiter returns to his normal task.

2.3. Basics of Bus Macro

Because of unsymmetry of the routing resources at the positions with block-ram elements, we designed a bus macro which connects all module slots on the reconfiguration area. Using a macro which is dedicated to only one module, leads to routing and placing problems because slots can be positioned and routed while containing block-ram elements.

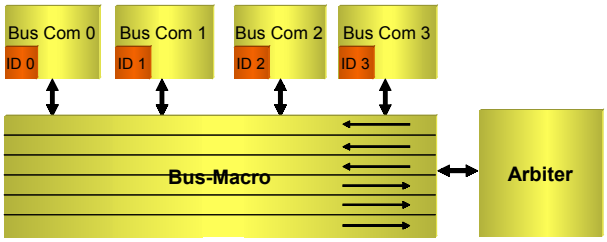

Figure 2.4 Bus Macro overview

In this case the unsymmetry causes a failure while routing. Figure 2.3 shows the macro with the connection to the arbiter in the position completely on the right.

Each macro is able to transport 8 bits of data unidirectional. If more than 8 bits are needed, another macro can be placed above or below. To enable a bidirectional bus an input and output macro was designed. The disadvantages of using unidirectional bus macros and the need of an input and output channel are minor, versus the benefit of using an automatic designflow without a time consuming debugging and the search for signal lines crossing module borders. In our implementation an input bus with the width of 32 bit and an output bus with 16 bit were used. The difference of these buses in comparison to buses using TBUF elements is that macro external pins are placed on slices. Each macro uses 20 slices as resource. The utilization of resources within a slot and the arbiter is 4 slices. The complete bus consists of 6 macros. This means that 24 slices per module slot are occupied by the bus system. In [6] an approach using a serial connection as communication path is described.

Investigations in using serial protocols for communication have been done and will be adapted to the existing system.

2.3.1. Input Communication Macro

The macro consists of 20 slices whereas 4 slices are used for connection to each module respectively to the arbiter. The look up tables of the slices are pre-programmed to route input signals through. The macro has the height of one CLB row and contains all the connection points for each module. As an example figure 2.6 shows the usage of the slices. Figure 2.5 shows the schematic of the complete macro used for the input signals. The right slices are used to connect the arbiter to the bus. 8 signal lines are routed to every module and are connected via 4 slices to the BUS-COM module.

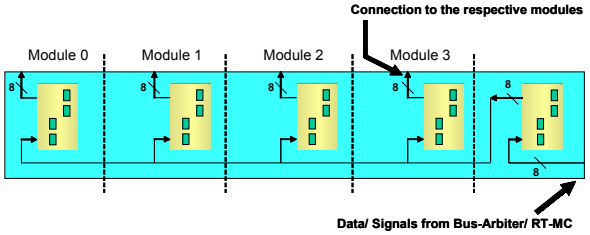

Figure 2.5 Schematic of Input Macro

The bus macro has a delay of maximal 5.5 ns (measured from the arbiter to the last module). Thus the bus can be clocked with the maximum possible clock frequency of 66 Mhz.

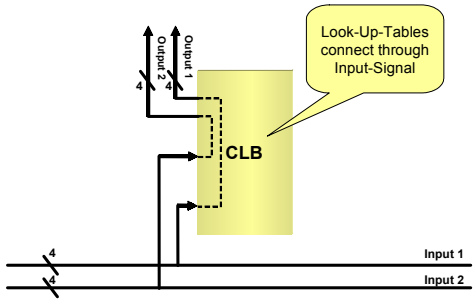

Figure 2.6 Usage of slices for input macro

2.3.2. Output Communication Macro

Similar to the input macro the output macro also uses 20 slices. However the look up tables are initialized to realize a multiplexer (see figure 2.7). This is necessary to grant the option of writing to the bus system with more than one module. The output macro has a height of one CLB row and connects all modules. Also the output macro has a delay time of 5.5 ns. This makes it possible to use the full clock speed for the complete bus system. Figure 2.8 shows the schematic of the implemented output bus macro. The slices on the right are used to connect to the arbiter. If a module is allowed to write data on the bus, the select signal switches the multiplexer inputs to the module's side. Now Predecessor modules are not able to use the bus. After the bus transfer of data the select signal switches the inputs to the predecessor module and the bus can now be used for other modules.

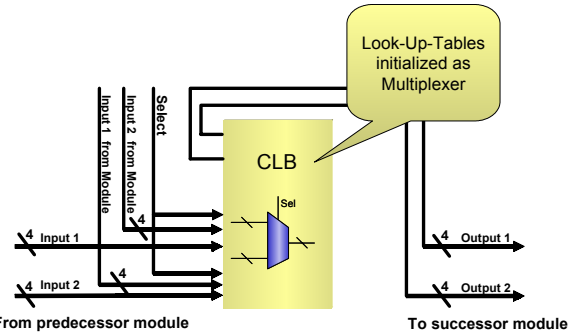

Figure 2.7 Usage of slices for output macro

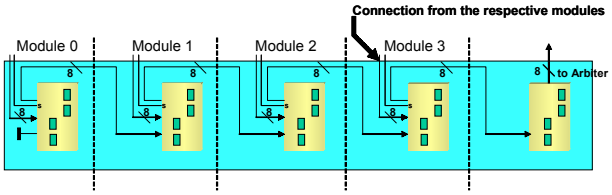

Figure 2.8 Schematic of output macro

3. System Integration

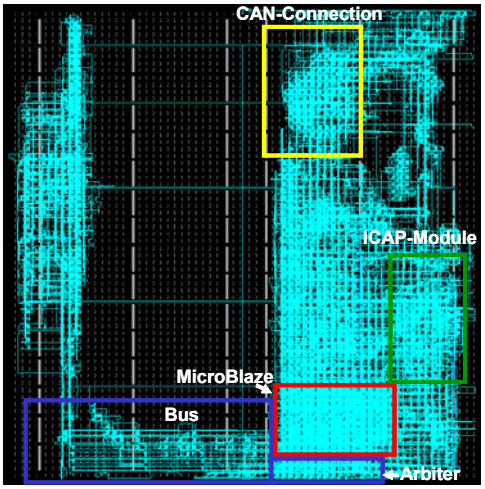

Figure 3.1 Complete system after placement and routing

By using the Xilinx design software ISE [9] and the tool EDK (Embedded Development Kit) [10] the complete system was implemented on the FPGA. Figure 3.1 shows the complete system placed on a Xilinx Virtex-II XC2V3000 FPGA. The modular designflow [12] was used to generate the partitioned design. Modular design

flow allows the creation of dynamic reconfigurable systems. Figure 3.1 shows the separation of the used modules in detail. For example one function on the left is implemented to show the first slot of the system. On the bottom the bus system with its signal lines connecting all modules is visible. In this system a width of 8 columns was used for the separated modules. Therefore a value of 2048 slices can be utilized in each module slot. For the BUS-COM Module and the logic for the bus-macro only 74 slices are utilized within the slot.

4. Real-Time Application

As described in sections 2.3.1 and 2.3.2 the bus can run at full clock speed of 66 Mhz. In our system it is possible to transfer 252 MByte/s with the input bus and 125 MByte/s with the output bus. The used applications configured in the slots need to save maximal 20 words with counter states or actual states of internal finite state machines. Thus for a reconfiguration process the time of 20 clock cycles (303 ns) is required for saving and loading the data. The time of about 606 ns is a fixed value for every context load or save process. Certainly additional time is needed for writing the configuration bitstream to the ICAP interface. In our case the amount of configuration data is about 118 Kb. With the speed capacity of 66 Mbyte/s it is possible to reconfigure one slot in 1.8 ms. This leads to a maximum reconfiguration frequency of about 555 Hz. Of course the time for reconfiguration depends of the chosen size of the slots. Timing measurements with the CAN-Simulation programme CANOE [8] shows, that response times shorter that 3 µs without reconfiguration are possible. If a slot has to be reconfigured, the response time is smaller than 10 ms. These results are sufficient for the automotive application we used for testing. Also the topology we designed with this bus structure is ideal for the real scenario of the test application. The bus topology and the possibility to use the run-time system as a central for managing the incoming and outgoing messages enables a fast and efficient transfer of data.

5. Conclusions and Future Work

This paper shows an implementation of a reconfigurable system using slices as connecting resources instead of TBUF elements. Using this design method an automatic design flow without manual debugging is possible. The system is implemented on a rapid prototyping system and runs at full clock speed of 66 Mhz. The bus system can easily be adapted to other demands e.g. higher in- or output width. Using the universal BUS-COM modules makes it possible to design easy connectable functions and allows saving important data before functions are substituted by others. Also data can be reloaded after reconfiguring a function in order to start for example with identical counter states. The number of module slots can easily be adapted to the required amount by changing the bus macro. Flexibility was the main focus while designing this system. Further work will be to design a tool which allows implementing this design methodology into the ISE design flow. This grants the option for a fast design for a dynamic and partial reconfigurable system. Further on the bus system can be adapted to the demands of run-time in future. At the moment the topology and the width of the bus is adapted to the requirements of the system. This feature of a network on chip can help to save energy because it uses only necessary resources of the FPGA. A new possibility of reconfiguring rectangular shaped areas for a better utilization of the area will be developed on the basis of this work. This grants the option to configure smaller functions without wasting precious reconfiguration area.

6. References

[1] J. Becker, M. Huebner, M. Ullmann: “Power Estimation and Power Measurement of Xilinx Virtex FPGAs: Trade-offs and Limitations”, SBCCI03, Sao Paulo, Sep. 03

[2] J. Becker, M. Huebner, M. Ullmann: “Real-Time Dynamically Run-Time Reconfiguration for Power-/Cost-optimized Virtex FPGA Realizations”, VLSI03, Darmstadt, Sep. 03

[3] L. Benini, G. De Micheli: “Networks on Chip: A New Paradigm for Systems on Chip Design”, Date 02, March 3~7, Paris France

[4] B. Blodget, S. McMillan: “A lightweight approach for embedded reconfiguration of FPGAs”, Date03, Munich Germany

[5] M. Huebner, M. Ullmann, F. Weissel, J. Becker: “Real-time Configuration Code Decompression for Dynamic FPGA Self-Reconfiguration”, RAW04, Santa Fee

[6] J.C. Palma, A. Vieira de Melo, F. G. Moraes, N. Calazans, "Core Communication Interface for FPGAs", SBCCI02, Porto Alegre BRAZIL

[7] M. Ullmann, M. Huebner, B. Grimm, J. Becker: “An FPGA Run-Time System for Dynamical On-Demand Reconfiguration”, RAW04, Santa Fee

[8] http://www.vector-cantech.com

[9] http://www.xilinx.com/ise/design_tools/

[10] http://www.xilinx.com/ise/embedded/edk.htm

[11] http://www.xilinx.com/ipcenter/processor_central/microblaze/literature.htm

[12] XAPP290: Two Flows for Partial Reconfiguration: Module Based or Small Bit Manipulations