Bachelor of Engineering THESIS

Run-time Partial Re-configuration

Kees van der Bok

Abstract

Run-Time Partial Re-configuration (RPR) enables a FPGA (Field Programable Gate Array) to be partially re-configured during run-time (i.e. part of the array can be over written with another re-configuration, while the remaining part is active).
An extra ordinary form of RPR is self re-configuration. In this special case the re-configuration process is controlled from within the chip. The re-configuration is performed by the static part of the logic, while the dynamic part is re-configured.
To enable self re-configuration, actually self run-time partial reconfiguration, the target FPGA should feature on-chip access to the re-configuration logic. Furthermore logic has to be designed controlling the re-configuration process (i.e. fetching the re-configuration data and writing it to the re-configuration logic).
In this thesis a solution enabling self re-configuration is presented. Although the solution is designed for integration in the MOLEN framework it has a broader application range.
The MOLEN framework is a, framework composed of a general purpose processor augmented with a re-configurable processor. The re-configurable processor can be re-configured using the solution presented in this thesis.

Rijswijk Institute of Professional Education
Lange Kleiweg 80
2288 GK Rijswijk
The Netherlands
http://www.thrijswijk.nl/

Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Mekelweg 4
2628 CD Delft
The Netherlands
http://ewi.tudelft.nl/

Computer Engineering Group
http://ce.et.tudelft.nl/

MOLEN Research Theme
http://ce.et.tudelft.nl/molen/

Delft, October 9, 2007

Run-time Partial Re-configuration
An Implementation for the MOLEN Framework

THESIS

submitted in partial fulfillment of the
requirements for the degree of

BACHELOR OF ENGINEERING

in

ELECTRICAL ENGINEERING

by

Kees van der Bok
born in Dirksland, The netherlands

Acknowledgements

During my thesis work, I received support from many persons. I am thankful towards all those people for there direct help, critical comment on my work and writing, or any other form of support.

There are a few people I want to thank personally. Both my supervisors, Th. Koreneef and A.J. van Genderen. Both gave me practical advice and commented on early versions of this writing. There support and guidance was really helpful. Both supervisors gave me freedom in my design choices, although the did comment on it critically. I have appreciated this professional attitude, wich has motivated my during my thesis work.

Beside my official supervisors, some other people showed there interest in my work. I want to thank R.F. Chaves, G.K. Kuzmanov, F. Duarte, B. Donchev and W. van Ooien in no particular order for there support, advice and comment.

Special thanks go to B. Meijs, system administrator of the Computer Engineering Group. I am very thankful for his support during my thesis period, especially for installing the ISE suite twice.

Furthermore I want to thank all other members of the CE group, it was a pleasure to work in this friendly though professional atmosphere.

Lastly I want to thank my relatives and friends. Although they where not directly involved there support, care and love motivated me throughout my study.

Kees van der Bok
Delft, The Netherlands
October 9, 2007

Summary

PR (Partial Re-configuration) is a technique that enables changing (i.e. re-configuration) part of a FPGA configuration, on the fly. Usual a re-configurable system is composed out of a fixed and a dynamic part. By means of PR the dynamic part can be re-configured on the fly. Run-time updates is one of the applications of PR. An even more advanced method of PR is Self PR. In this special case the re-configuration is performed by the static part of the design, thus by the system it self.

The MOLEN polymorphic processor framework, from now on referred to as MOLEN framework, is composed of a fixed processor and a re-configurable processor, intends to use the self re-configuration methodology to enable run-time re-configuration of the re-configurable processor.

For efficiency reasons multiple re-configurable processors are proposed. A number of re-configurable areas also referred to as slots are reserved for the re-configurable processors. The current design enables static mapping (i.e. the destination slot is determined at compile time). An additional core, manipulating the re-configuration data needs to be implemented to enable dynamic mapping (i.e. the destination slot is determined at run-time).

Self re-configuration of a re-configurable processor is realised by the PRMU (Partial Re-configuration Management Unit). The PRMU is an independent core designed for integration in the MOLEN framework. The PRMU is composed of three components the re-configuration unit, the repository selector and one or more repository interfaces.

The re-configuration unit handles the re-configuration process. The re-configuration data is stored in a repository accessible using the repository interface. A repository interface adapts the data transfer protocol of the repository (i.e. memory) to the protocol used within the PRMU. The design supports multiple repositories. The repository selector, an advanced mux, is used to select the proper repository.

The internal data transfer protocol of the PRMU is a handshaking type protocol. This protocol is used for two reasons namely. To simplify the integration of other repositories as currently supported. Furthermore the protocol enables the integration of re-configuration data processing units.

The PRMU is thoroughly verified and tested. By testing and verification processes have proven the proper operation. Furthermore the tests are used to determine the performance parameters of the PRMU. Using the tests we concluded that the PRMU is able of performing a re-configuration at the maximum speed on both the Virtex 2 Pro and Virtex 4 FPGA series.

The current implementation supports only a small repository. This repository is able of storing only a limited amount of re-configuration data. For useful applications support of a larger repository is necessary. Future research is necessary to find a solution for this problem.

To realise dynamic mapping of re-configurable processors a core capable of altering the re-configuration data needs to be implemented. The current design enables rapid integration of such core.

Contents

List of Figures

List of Tables

1 Introduction

This chapter gives a brief overview of the MOLEN project. Furthermore the purpose and aims of this thesis work are given. This chapter also describes the context of this thesis work within the MOLEN project.

The next section describes the MOLEN research theme, while section 1.2 describes the MOLEN polymorphic processor framework from now on referred to as MOLEN framework. This framework is one of the main projects of the research theme and is the context of this thesis work. Section 1.4 describes the ISA (Instruction Set Architecture) extension applied to a standard GPP (General Purpose Processor) necessary to realise the polymorphic behavior of the MOLEN Framework. Section 1.5 describes in brief the aims for this thesis work. The last section describes the principle of micro-code. Micro-code is a method for implementing control of digital systems. The micro-code principle is discussed because it is a key feature of the MOLEN framework.

1.1 What is MOLEN

MOLEN is a research theme of the Computer Engineering group at the Delft University of Technology. The MOLEN theme research is mainly focused on re-configurable computing. A more specific direction of research within this field is adaptable processors. The MOLEN framework described in this section is an example of an adaptable processor.

These adaptable processors also referred to as polymorphic[1] , can be partially adapted. The benefit of a polymorphic processor is that it can dynamically (i.e. run-time) adapt to a specific task. The processor can be adapted either dynamic (i.e. run-time) or static (i.e not run-time).

Adaptable processors can be implemented using re-configurable fabric like FPGAs. Nowadays state of the art FPGAs even offer partial re-configuration. This feature enables the FPGA to re-configured only partial. The PR (Partial Re-configuration) feature enables implementation of fixed en adaptable parts on a single chip. Single chip systems also known as SOCs (System On Chip) have significant advantages over systems distributed on multiple chips. Rapid development and early prototyping are the most relevant of these advantages. Thus RP enables implementation of the MOLEN Framework on a single FPGA.

[1]Polymorphic: Composition of poly (many) an morph(form). Meaning: An object capable of occurring in many forms and thus capable of changing form

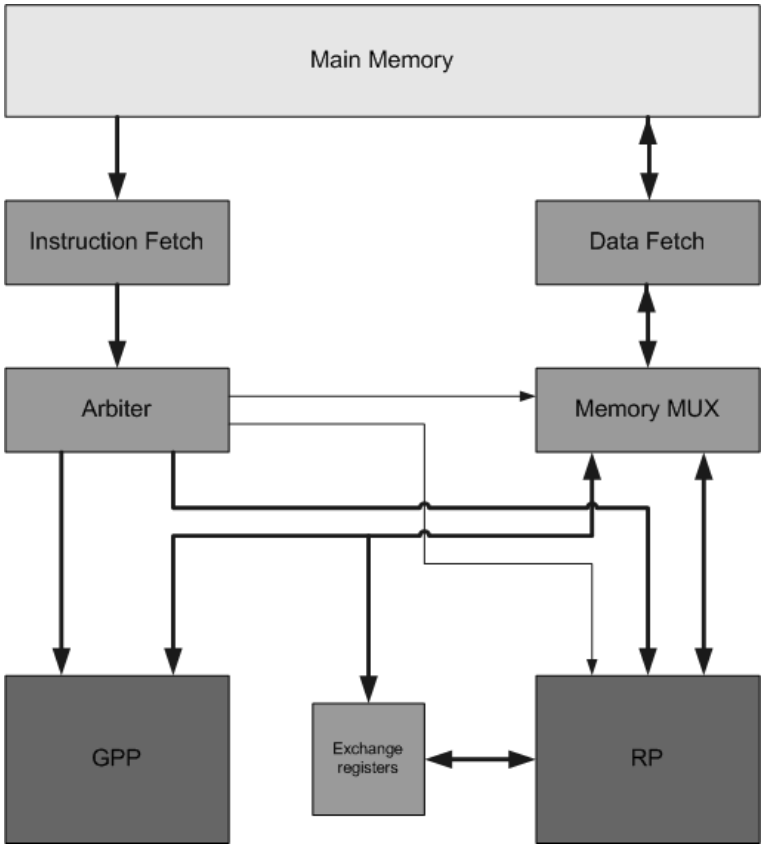

Figure 1.1: Schematic representation of the *Polymorphic Processor Framework*

The aim of this thesis work is the implementation of RPR applied to the MOLEN framework. The MOLEN framework is a composition of a general purpose processor (GPP) and a re-configurable processor (RP). The framework is discussed in detail in the next section.

1.2 The MOLEN Polymorphic Processor Framework

The MOLEN Framework architecture shown in figure 1.1, contains multiple components which purpose and relations are discussed hereafter.

The GPP (General Purpose Processor) and the RP (Re-configurable Processor) are the main components of the MOLEN framework, these two components are responsible for the actual execution of program code. The GPP executes the ordinary instructions (i.e. instructions which are members of the GGP's instruction set). The RP executes "custom" instructions. These instructions are actually micro-instructions[2] instead of machine-instructions. The re-configurable RP is responsible for the polymorphic

[2]Section 1.6 discusses these micro-instructions as well as the general concept of micro-code in more detail

behaviour of the framework.

The other components have a supporting character, one part of the components is responsible for fetching and directing machine-instructions or micro-instructions to the correct processor. Another part is responsible for arbitration of the memory access of both processors, while a third part takes care of the inter-processor communication.

The components managing the instruction fetching and multiplexing are the instruction fetch and the arbiter . The instruction fetch component fetches instructions from the main memory and passes them to the arbiter. The arbiter determines whether the instruction is native to the GPP or a extension instruction. Based on this analysis the instruction is either fed to the GGP or actions are taken to control the RP.

Because both processors need to access the main memory some sort of arbitration is required. The data bus is multiplexed to either the RP or the GPP. The arbiter controls the memory multiplexer. The data fetch component fetches the data from the main memory.

The Exchange registers are used for inter-processor communication i.e. data transfers between the GPP and RP. The exchange registers are accessible by both processors. The exchange registers are meant to transfer small amounts of data.

1.3 The Re-configurable Processor

The re-configurable processor or also referred to as CCU (Custom Configurable Unit) is the very essence of the MOLEN framework and needs some further explanation. The RP can be any hardware core, from an advanced ALU (Arithmetic and Logic Unit) up to a processor as complex as a GPP. The RP can use the principle of micro-code, but doesn't have to do so.

The RP has a dedicated interface to the rest of the MOLEN Framework, this interface is specified in depth by appendix B. Every RP has to comply to this interface.

1.4 The Polymorphic ISA

ISA (Instruction Set Architecture) is the set of instructions a particular processor supports. in order to support additional features like for example a RP one could extend the ISA with new instructions. The policy of adding instructions to the ISA for each hardware function or RP has proven to be inefficient. Because it leads to so called instruction set explosion. Large instruction sets with complex instructions make it hard

to develop efficient compilers. Thus instruction set explosion is undesired[3]

To prevent a instruction set explosion a one-time instruction set extension is applied in the MOLEN framework. This instruction set extension exists of a limited number of instructions enabling a virtual infinite extension of functions. The complete extension exists of eight instructions listed hereafter.

set The set instruction initiates a re-configuration of the RP. There are two forms of this instruction namely *p-set* and *c-set*. The *p-set* is a partially set and configures only frequently used functions, while the *c-set* is used to configure all the functions. Thus the *p-set* partial re-configures the RP. The general format of the set instructions are *set <address>* where address is the header address of the re-configuration data.

execute The execute instruction is meant to execute a micro-code program on the RP, the format of this instruction is *execute <address>* where address is the start address of the micro-program.

set prefetch This instruction is used to prefetch the configuration data into a fast local memory to diminish loading times.

execute prefetch Prefetches the micro-program to a local memory.

break The break instruction is used to synchronize the two processors, RP and GPP. The *break* instruction completes a phase of parallel execution of both processors. When a break instruction is fetched the program execution is postponed until both the RP and GPP have finished there current process. Thus the break statement is used to enable parallel execution of a hardware and software threat and forbids it in case of dependency between the two threats

movtx Used to transfer the contents of an general purpose register to a exchange register.

movfx Used to transfer the content of an exchange register to a general purpose register.

The described instructions are handled by the arbiter, except for the *movtx* and *movfx* instructions which are special instructions offered by the GPP[4]. The other instructions are executed by asserting signals of the RP interface. Native GGP instructions are supplied to the GPP.

[3]The fact that smaller instruction sets enable development to more efficient compilers was discovered in the seventies, the architecture referred to as RISC (Reduced Instruction Set Computer) became popular since this discovery. However GPPs used in nowadays PCs are still CISC (Complex Instruction Set Computer). This less efficient architecture is still used in order to maintain backwards binary compatibility.

[4]The GGP of the current prototype is a PowerPC 405, embedded as hardcore in the Virtex II Pro FPGA

Not all the instructions are supported by the current prototype. The prefetch and break commands are not supported, neither is the set instruction.

The solution developed during this thesis enables the set command, although only the c-set command will be supported.

start-op This signal is use to request either the execution of the micro-program or a re-configuration of the RP depending on the set and execute signal.

end-op This signal tells the arbiter that the requested execution has been performed.

set The set signal is used as qualifier, when the set and start-op are asserted a re-configuration is requested. When the set signal is not asserted a micro-program execution is requested.

execute This qualifier is used to indicate a micro-program execution request.

micro-code address This bus is used to pass the start address of either the micro-code or re-configuration data.

1.5 Aims and Context of This Thesis Work

The current prototype does not enable re-configuration of the RP, the main aim of this thesis is to implement RPR (Run-time Partial Re-configuration) and integrate it in the MOLEN framework prototype. The current prototype platform is the Virtex II pro based XUP (Xilinx University Program) board.

The unit to be develop must interface with the arbiter of the MOLEN framework as shown in figure 1.2. The following paragraph describes the proposed design and its operation in brief.

The arbiter will request a re-configuration by asserting the set and start-op signal as well as supplying the start address of the re-configuration data using the micro-code address bus. The re-configuration data is stored in repositories (i.e. memory). The current design foresees two repositories as shown in figure 1.2. The ICAP (Internal Configuration Access Port) is an internal Virtex II Pro component enabling run-time partial re-configuration of the FPGA.

1.6 About Micro-code

Although nowadays FSMs (Finite State Machines) are popular for control logic implementation, it has some disadvantages when it comes to control of complex systems. An other method for implementation control is micro-code, in contrast to the method FSM

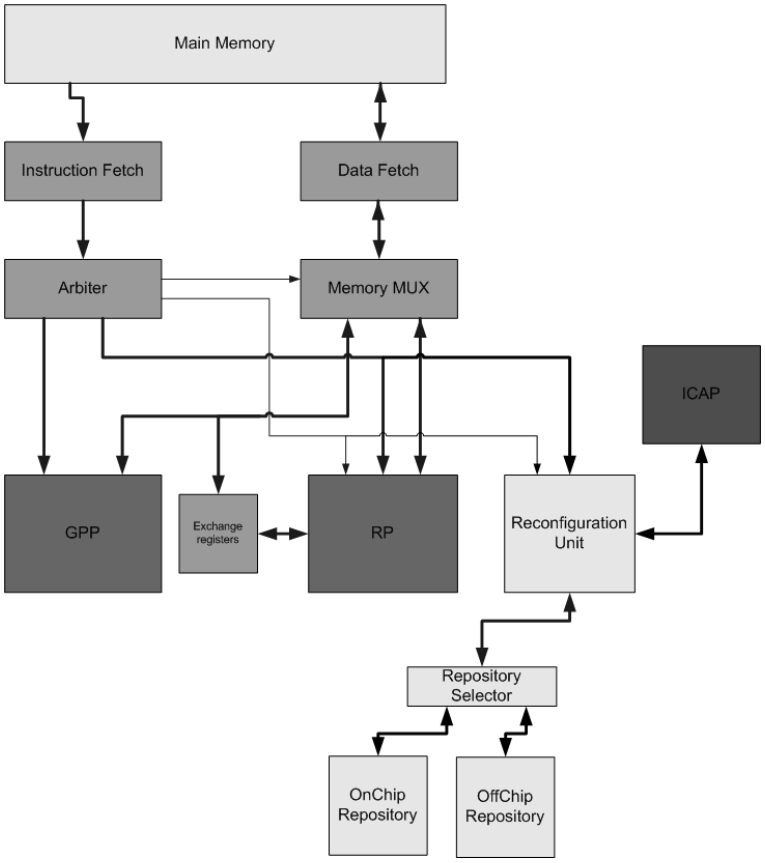

Figure 1.2: Schematic representation of the polymorphic processor framework, including units for partial re-configuration

it has proven to be convenient especially in complex systems.

The just mentioned characteristic is the main reason for featuring the micro-code principle in the MOLEN framework.

Digital systems are usually divided into two distinct parts, namely datapath and control. Computer architectures are usual designed using this principle. There are various ways to implement the control part, a FSM is a commonly used method.

Some digital systems like computers require complex control logic. When using a FSM this will lead to a large number of states. Such FSMs are hard to implement and even harder to debug. Instead of using a FSM another method know as micro-code is an attractive alternative for implementation of control logic.

The micro-code principle dates back to the 1950's and was first introduced by Wilkes [12]. Since its introduction the micro-code principle has involved and is widely used in many computer architectures. Micro-code control is based on small programs,

taking care of the control. These programs are called micro-programs. Usually there is a single micro-program for each machine-instruction, covering the control for that specific instruction.

Thus the control of each instruction can be designed independently. This is an advantage over the FSM method, using one huge FSM covering each instruction. Furthermore micro-code is more flexible, because the micro-program(s) can be adapted afterwards.

FSM control can be divided over multiple FSM's to minimise the complexity. Although large FSMs are prevented, it doesn't offer the flexibility and extendibility of the micro-code principle.

We can state that micro-code based control can be advantageous for implementation of complex control, although it requires a non constant[5] amount of memory to store the micro-programs.

Similar to machine code, micro-code is constructed out of binary codes. Machine codes are usually divided into a few field (e.g. the first 6 bits contain the op-code, the next 5 bits contain the target register address et cetera) as is micro-code. The format of micro-code usually exists of two fields, one field is used for control while the other is used for sequencing (i.e. determination of which micro-instruction is to be executed next). Thus a single micro-instructions contains information for control as well as information determining which instruction must follow the current one.

Furthermore there are two distinct types of micro-code, horizontal and vertical. De horizontal controls the hardware directly (i.e. The content of the "control" field is directly supplied to the hardware, hence that the content represents the control signals) , the vertical type includes a decode stage (i.e. the content of the "control" field is decoded before the actual hardware is controlled).

Figure 1.3 shows a typical implementation of micro-code control. The micro-code storage is usual implemented using a certain type of memory: RAM, ROM etc depending on the desired degree of flexibility. A specific range of the output data is supplied to the "logic under control", either direct (horizontal micro-code) or via a decoder (vertical micro-code).

The remaining bits of the output data contain the sequencing information and are supplied to the address select logic. This block controls the micro-program counter, which serves a similar purpose as a program counter known from computer architectures. The micro-program counter selects the next micro-instruction to be executed.

Micro-progams are usually executed in sequential order, although unconditional

[5]The memory usage will grow when control complexity increases

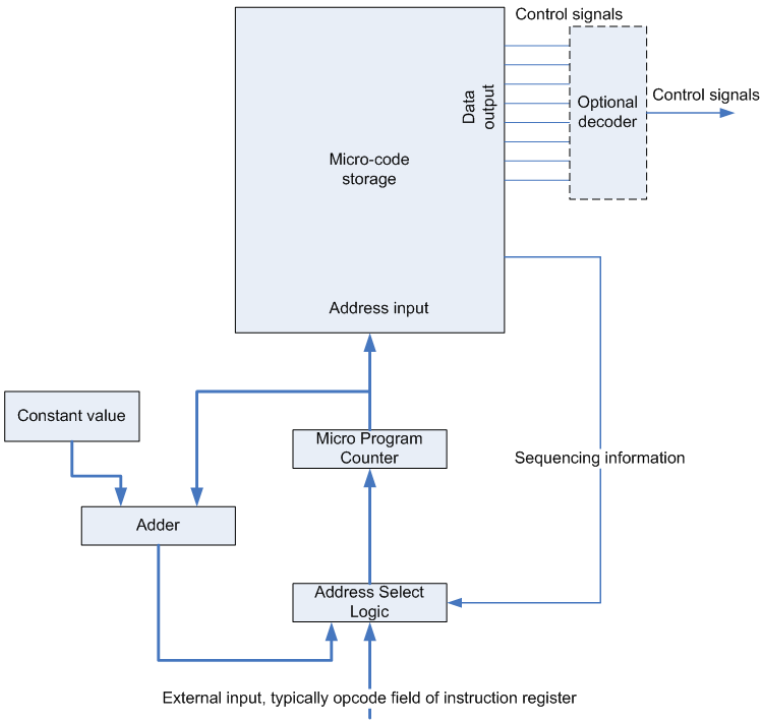

Figure 1.3: Micro-code control logic

jumps are also supported. As mentioned before the sequencing information determines the next micro-code to be executed. In case of a sequential execution the address generated by the feedback loop is supplied to the micro-program counter. In case of a jump the sequencing bits are used to to determine the next address to generate (e.g. The sequencing bits represent the relative jump). Thus the number of bits reserved as sequencing information determine the maximum jump.

Furthermore there is a sequencing code serving as end instruction meant to stop micro-code execution at the end of the micro-program. The address select logic can also select an address based on information supplied from outside the micro-code control logic This external supplied information triggers the execution of a specific micro-program. In micro-code based GPPs the external signal is usually the op-code field of the machine-instruction. The content of the op-code field is used to determine the start address of the correct micro-program (i.e. the micro program handling the control of the instruction identified width the supplied op-code).

The MOLEN framework uses a slightly different implementation of micro-code control, although the principle is the same. The implementation shown in figure 1.3 is used for clarity and to eliminate MOLEN framework specific details, irrelevant for the general description given in this section.

Micro-code is supported by the MOLEN Framework for implementation of RPs

with complex control. The micro-programs are store in the main memory. The micro-instructions are redirected to the RP by the arbiter. Thus by using micro-code a unlimited number of special instructions can be implemented. These instructions are executed by a RP

For a more detailed description the interested reader is referred to [8] in particular section 5.7. Another another interesting document regarding this topic is [11].

Study into Partial Re-configuration

2

This chapter discusses the matter of partial re-configuration. The discussion is meant to explore the the principle and how we can use it to enable run-time re-configuration of the CCU (Custom Configurable Unit) in the MOLEN framework.

Section 2.1 discussed RPR (Run-time Partial Re-configuration) in brief while section 2.2 defines the desired usage of the RPR principle in the MOLEN framework. This section states what we expect from the RPR principle. The remaining sections are mainly focused on how to use the RPR principle in order to implement CCU swapping in the MOLEN framework.

The Xilinx supported design flows for RPR are discussed in section 2.3. These design flows are develop by Xilinx to simplify RPR design implementation.

Section 2.4 discusses the Virtex Pro organisation, furthermore this section gives information about the process of re-configuration. The composition and format of the re-configuration data is discussed in section 2.5. The re-configuration data is wrapped in a file. The format of this file is also discussed in 2.5.

The matter of bitstream manipulation is discussed in section 2.6. Bitstream manipulation is used to enable re-location. The ICAP (Internal Configuration Access Port) is described in section 2.7, the ICAP gives access to the re-configuration logic from within the FPGA.

Section 2.9 and 2.8 are about the future developments of RPR and the current support among FPGA's. Design proposals are discussed in section 2.10. This section also includes a trade off. The last section outlines the proposed functional design strategy.

2.1 Run-time Partial Re-configuration

Run-time partial re-configuration is a feature offered by todays state of the art FPGAs including the Virtex II pro. The feature enables hardware designs to be partially re-configured. It is used for run-time upgrades or for reliability in high radiation environments[1]. Hardware task swapping is a third application of RPR, although it is rarely used.

[1]Radiation can harm the current configuration, by performing frequent re-configurations malfunctioning of the system is prevented

Although partial re-configuration is supported it is not often used, especially not in the industry. Academic users have generally more interest in this feature. Due to the fact that the run-time partial re-configuration is relatively slow and limited, most of the academic research concentrates on the conceptual level. Some experimental so called "self configuring systems" are developed, but they haven't proven to be very efficient.

Many papers discussing the matter of partial re-configuration conclude that the feature is not mature enough to exploit its potential. The current limitations of partial re-configuration is seen as major obstacle to a totally new application of FPGAs, namely self reconfiguring systems on chip.

A fast and more mature version of partial re-configuration enables the development of efficient self reconfiguring systems. Which would be a complete new and promising concept. Efficient self reconfiguring systems combine the flexibility of software and the performance of hardware.

2.2 RPR Usage in the MOLEN Framework

This section discusses a general concept of the desired usage of the partial re-configuration feature of the Virtex II pro in the MOLEN processor. Although this concept is naive it is good to have an ideal concept before considering possible limitations. By this approach one is not influenced or actually controlled by the limitations, but can anticipate to them by adapting the naive design to a feasible version. Which should result in a design close to what is initially intended.

Implementation of real-time hardware unit swapping is one of the phases of the development of the MOLEN framewok. RPR is supported by the Virtex II pro FPGA. The Virtex II pro is currently used as prototyping platform for the MOLEN project. Hardware unit swapping can be realized by using the RPR principle.

Using the RPR principle we want to implemented a form of hardware task swapping. For efficiency reasons we consider multiple hardware tasks (i.e CCUs) to operate in parallel. Using RPR we want to enable "loading" of hardware tasks. Furthermore we want the loaded hardware tasks to be mapped to a free space of the FPGA. When there is no free space available, we want other tasks to be removed to make place for the new task.

The dynamic (i.e. run-time) mapping of the hardware tasks also referred to as re-location, would highly improve the efficiency and user friendliness of the MOLEN framework. The remainder of the chapter describes the technical details of RPR. Furthermore the feasibility of our proposal is determined.

2.3 Run-time Partial Re-configuration Design Flows

Xilinx offers two basic methodologies for implementation of partial re-configuration [13]. Module based and difference based partial re-configuration. In this section the two Xilinx supported methodologies are described.

A third possibility is the manipulation of the configuration data. This last option is not supported by Xilinx ,but there are some academic projects using this unorthodox methodology see [4], [9] and [3]. This method is not discussed in this section, but in section 2.6 because it requires more detailed knowledge of the re-configuration process.

2.3.1 Module Based Partial Re-configuration

The module based partial re-configuration methodology integrates partial re-configuration in the "Xilinx Modular Design flow". This methodology is originally intended to manage projects in which multiple designers participate.

The modules of a modular design are separate projects. A module can be implemented independent. The integration of modules is done after successful verification of each module.

Xilinx has extended the modular design flow in order to support partial re-configuration. A module based partial re-configurable design is composed of multi modules. Each module can either be static or re-configurable.

The "Modular design flow" is based on the following steps, which are derived from Xilinx documentation see [13].

1. A top-level design needs to be constructed which defines the interfaces of each module. Furthermore each module must be assigned a fixed area on the FPGA by means of a floorplan.

2. The separated modules can be developed and tested independently. The implementation must meet the constrains specified by the toplevel design and floorplan.

3. Finally the intended system can be realized by intergrading the separately implemented modules.

The described design flow can be extended by making one or more modules re-configurable. In order to make a module re-configurable one has to take care of signals crossing the module border. Signals crossing the re-configurable modules boundaries must do so through fixed routing bridges referred to as "bus macros".

The "bus macros" function as fixed interfaces at the module boundaries. Due to these components, a module can safely being overwritten by another module compatible

to the interface. Without losing inter-modular communication.

The module based design flow does not support dynamic mapping of a module. The area that will be occupied by the module must be determined during implementation. Thus dynamic mapping or re-loaction is not supported by this standard design flow.

The preceding discussion leads to the conclusion that the "Module based partial re-configuration" methodology does not support re-location of hardware units. The "Module based partial re-configuration" methodology must be extended or another methodology must be used to enable re-location.

2.3.2 Difference Based Partial Re-configuration

The "Difference based partial re-configuration" is the other Xilinx supported methodology for partial re-configuration. This difference based methodology is meant to switch to another implementation which slightly differs from the original.

The advantages of this method is that only the differences between the two implementations are re-configured. The re-configuration data for the partial re-configuration is formed by the difference between both implementations. Due to the fact that the redundant parts of the implementation are preserved and only the changes are altered there is a gain in re-configuration latency above a complete re-configuration.

Although the discussed methodology offers a superb way to perform a runtime upgrade, it is not suitable in the foreseen usage of partial re-configuration in the MOLEN processor. Because CCU's typically use a significant area of the chip. The difference based flow is not beneficial in such case. However the difference based flow can be advantageous when we consider two or more CCUs which slightly differ. Using the difference based method in these cases can result in a significant performance gain, although the logic controlling the re-configuration process will increase in complexity.

Further research must be done in order to determine whether the it is worth supporting difference based design in the MOLEN framework.

2.4 Virtex II Pro, Organisation and Re-configuration

In this section the re-configuration of the Virtex II pro is discussed. The principle of re-configuration and the organization of the re-configurable components are discussed. The information in this section is mainly intended to understand the re-configuration process and the format of the re-configuration data.

An FPGA is a chip containing logic components and programmable interconnects. During configuration the components are programmed and interconnections between

these components are made in order to realize a digital system.

A logic component typically contains LUTs (Look Up Tables) as well as flip flops. The former is used to emulate combinatorial behavior while the later is used to construct sequential logic. By means of the programmable interconnects the logic components are connected in order to realize a digital system.

A FPGA is re-configured using re-configuration data, generated by implementation tools. Xilinx refers to the re-configuration data as bitstream. The bitstream is usually created by synthesizing a hardware description, formulated in a HDL (Hardware Description Language) like VHDL.

Nowadays FPGAs offer specialized hardware units in addition to the standard logic components. These dedicated hardware blocks are usually multipliers, dividers, RAM or other specialized units.

Although FPGAs are commonly used for rapid prototyping, new application fields are foreseen due to development of concepts like RPR.

2.4.1 Configurable units of the Virtex II pro

The Virtex II pro FPGAs has several programmable logic components as described hereafter. The descriptions are intentionally kept brief, because they are meant to give a general understanding prior to the discussion of the architectural organization of the FPGA.

CLB The *Configurable Logic Blocks* are the basic components of the FPGA. Combinatorial as well as sequential logic can be constructed using these blocks. Furthermore these blocks contain multiplexers.

Multipliers The FPGA is equipped with a number of dedicated multipliers. The dedicated multipliers are much more efficient in terms of footprint than a multiplier constructed using CLBs.

BRAM *Block RAM* Memory for data storage are encapsulated in the FPGA.

SelectRAM These components are used to configure a interface to the BRAMs. Memory blocks of a desired capacity and width[2] can be configured using BRAM and BRAM interconnect components.

IOB *Input Output blocks*, contain logic to configure a pin of the FPGA as input or output. Furthermore the IOB can be used to configure a specific I/O standard.

IOI The *Input Output Interface* components are used to configure the I/O pins behavior e.g. tristate, weak, strong etc.

[2]There are some restrictions regarding the width an capacity

PowerPC hardcore The version of Virtex II pro FPGA used during this project contains two hardcore PowerPC general purpose processors. The cores are mapped within the configurable array, but are not re-configurable themselves.

MGT The *Multi Gigabit Transceiver* core is a programmable core used for high speed data communication with the outside world. The MGT cores are not used within this project.

DCM The *Digital Clock Manager* The DCM primitives are generating clock signals for the design. A DCM can produce an number of clock signals, DCMs can either multiply or divide a source clock signal. Furthermore a DCM can adapt the phase ad the duty cycle of the input clock cycle etc.

2.4.2 Interconnection Infrastructure

The interconnection infrastructure is formed by routing wires. These wires run through the chip in vertical and horizontal direction. The wires can cross the entire chip or only part of it, respectively local and global routing wires.

During configuration, routing wires are connected to each other in order to create paths. These paths interconnect the various components of the FPGA. This process is referred to as routing. The connections are made using the interconnection switches.

2.4.3 Virtex II pro Architecture

In general the array of an FPGA is organized in a way that enables efficient routing. Because the organization of the array is crucial to determine if bitstream manipulation is feasible, a discussion follows.

The array is organized in columns of equal components see figure 2.1. Each column is divide in multiple frames. In general there are two types of columns: CLB columns and BRAM/MUL columns. The former contain CLB elements while the later contain multipliers, memory and selectRAM components. BRAM/MUL columns are followed by CLB columns, this pattern is repeated throughout the entire array. The two hardcore PowerPc units are located half way the third[3] an seventh column see figure 2.1. The PowerPc units stretch two BRAM and one CLB row. IOB elements surround the entire array.

Re-configuration is the process of "programming" the FPGA elements and routing the interconnection between them. Information about the "programming" of elements and interconnection paths is given by the re-configuration data or bitstream.

The Virtex II Pro is a SRAM based FPGA. SRAM based FPGAs feature a SRAM memory to re-configure the FPGA. The bits stored in this memory configure the under

[3]The columns are counted from left to right

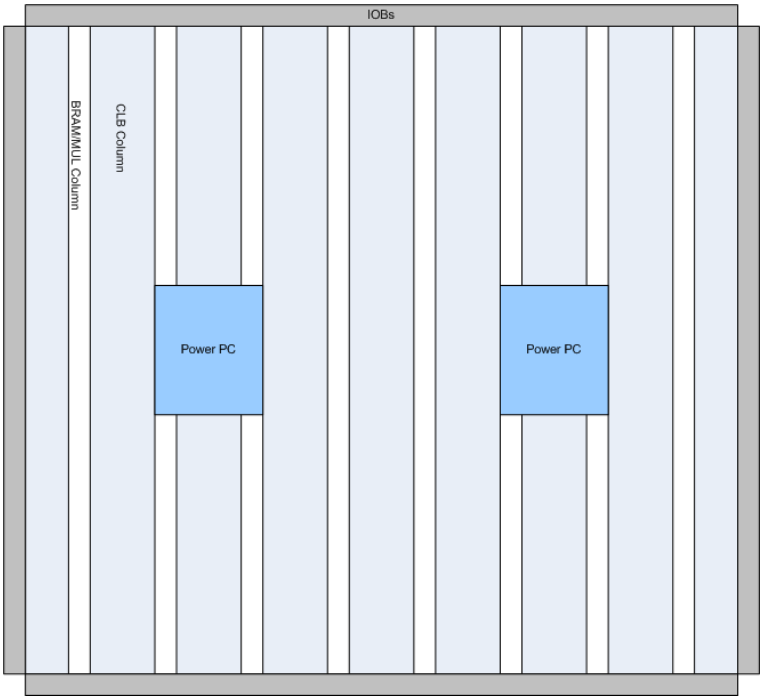

Figure 2.1: Organization of Virtex II Pro array

laying components such as CLB's et cetera. The memory is written in a vertical way segment by segment. These segments also referred to as frames are one bit wide and are the smallest addressable portion of the configuration memory.

2.5 Re-configuraton Data

The re-configuration data referred to as bitstream is basically a sequence of commands and data. The re-configuration logic encodes this bitstream and configures the FPGA according to these commands and data.

The following section describes the composition of a bitstream. An example bitstream is given and discussed in detail. The second section handles the format of a bit file. A bit file is a file generated by the Xilinx tools. A bit file is composed out of the bitstream and some additional information.

2.5.1 Bitstream Composition

A bitstream starts with a unique synchronisation word, this word is unique within the bitstream. The rest of the bitstream is composed out of headers followed by a number of data words. The header contains the destination of the data as well as the number of

data words that follow. The composition of header and data is called a package.

There are two forms of the header word, the Type 1 and the Type 2 header. A Type 2 header is optional and follows the Type 1 header. An additional type 2 header must be used when the word count (i.e the number of data words following the header) is to large to fit in the type 1 header.

When a type 1 header is used the ten least significant bits of the header word are used to represent the word count (The maximum word count value is $2^{10} = 1,024$) When using a additional Type 2 header the 27 lower bits of this word represent the word count (The maximum word count is $2^{27} = 134,217,728$).

There are two other commands, namely the *de-synchronisation* word and the *no-operation* word. The *de-synchronisation* word is used to end the re-configuration, a few additional *no-operation* words may follow to flush the re-configuration logic's buffer. Table 2.1 shows a example Bitstream, this Bitstream is not complete, but it shows how a Bitstream is composed.

In the example bitstream a few specific registers are addressed. These registers are embedded within the re-configuration logic and accessible through the ICAP. The address registers and there purpose are described hereafter.

CRC Cyclic Redundancy Check register, this register is used to store the CRC word of the bitstream.

CMD The command register

FAR Frame Address Register

FDRI Frame Data Register Input

2.5.2 Bitstream File

The Xilinx Bitgen tool is used during the design flow to generate Bitstreams as described in the paragraph 2.5. Bitgen produces a ".bit" file, composed of the Bitstream and and some additional information regarding the Bitstream. Because the Bit file format is not documented some reverse engineering was use to analyse the actual format. Understanding the Bit File format is necessary because when applying on-chip partial re-configuration we need to store the Bitstream in On-chip memory. Furthermore the on-chip re-configuration logic doesn't except the additional information, only the raw bitstream must be written to the configuration port.

The Bitgen tool generates the Bit file using the ncd file, the ncd file is a description of the design implementation in respect to the target device. The ncd file is the output of the PAR (Place And Route) process. The Xilinx implementation flow is given by the following enumeration.

Table 2.1: Example Bitstream

Configuration Data	**Explanation**
0xAA995566	Synchronisation word
0x30000001	Type 1 Header: Write 1 word to the CRC register
0x00009A32	Packet Data: CRC Word
0x30008001	Type 1 Header: Write 1 word to CMD register
0x00000001	Packet Data: WCFG command
0x30002001	Type 1 Header: Write 1 word to the FAR register
0x02000000	Packet Data: Frame Address
0x3000401A	Type 1 Header: Write 26 words to the FDRI register
0x????????	Packet Data: Reconfig. Data word 1
...	Packet Data: Reconfig. Data n
0x????????	Packet Data: Reconfig. Data 26
0x0000A53B	Auto CRC word
0x0000000D	De-synchronisation Word
0x20000000	Optional No Operation word
0x20000000	Optional No Operation word

1. HDL coding
2. Synthesise
3. Translate
4. Map
5. Place and Route
6. Programming File Generation (i.e. Bitstream)

As mentioned in the preceding paragraph, the Bit File is composed of a header followed by the Raw Bitstream. The Bit File exists of a few sections each containing specific information. The header information is used by Xilinx programming tools to validate whether the Bitstream suites the target device.

The sections of the Bit File contains a key, length field and a data field. The key is a single ascii character, a to e where a is the fist section b the second etc. The length field is always followed by NULL byte (i.e. hex 00). The Length field is a single byte representing the length of the data field in bytes. The information or header sections contain ascii formatted strings terminated with a NULL word[4]. The last section, section e contains the Bitstream and some additional information. Interesting detail is that the first section is not preceded with a character.

[4]Equivalent to C strings

Table 2.2: Bit file section composition

Key	Length	Content

Table 2.2 shows the typical section composition all sections comply to this format. Except for the first section referred to a preamble. The e section containing the actual has bitstream has multiple byte length field in contrast to the other sections al having a one byte length field.

preamble Nine byte section containing undefined data, probably a fixed header saying this is a bit file.

NULL word ascii NULL word (Hex 00)

Length field Length of the preamble section (hex 09)

Data field probably an ID content: (hex 0F F0 0F F0 0F F0 0F F0 00)

Extra NULL word

Section 'a' Source file

Key (ascii 'a')

NULL word

Length field Content: Variable

Data field Source ncd file, content example: PRM2_partial.ncd

Section 'b' Target device

Key ascii 'b'

NULL word

Length field

Data field Target device, content example: 2vp30ff896

Section 'c' Date of generation

Key ascii 'c'

NULL word

Length field

Data field Date of generation, content example: 2007/07/26

Section 'd' Time of generation

Key ascii 'd'

NULL word

Length field

Data field Time of generation, content example: 15:15:19

Section 'e' Bitstream

Key ascii 'e'

NULL word

Length field The length of the bitstream in bytes[5]

Dummy word The dummy word is apparently used to flush the buffer of the re-configuration logic, content : hex FF FF FF FF

Data field bitstream

- content start: hex AA995566
- content end: hex 00 00 00 0D or hex 20 00 00 00

2.6 Re-location by Bitstream Manipulation

The two supported methodologies have proven to be insufficient in order to implement the desired concept of relocation. Because of that other ways must be explored. At first the possibility and feasibility of bitstream manipulation is discussed. The discussion is based on the following question.

Is it possible to manipulate the configuration data in such a way that the module will be mapped to an other area as it was originally intended to ?.

A brief search for papers concerning this subject revealed that the idea of configuration data manipulation had been risen to the minds of several researchers. In these papers concepts are proposed. Some of them even claim functional implementations. For more detail see [3], [4] and [9].

Based on this information the answer to the earlier question is yes.

The question remains what exactly must be manipulated, for bitstream is a rather vage term which can interpret in many ways. In section 2.5 we saw that the addressing of the destination frames is incapsulated within the bitstream. If we alter the frame addresses in the bitstream by adding an offset, the re-configuration is applied to a other area of the FPGA. Thus by doing so we are able to configure various areas of the FPGA using a single source bitstream.

The implementation of this relocation must scan the bitstream for the hearder word addressing the FAR (Frame Address Register) and alter the word following this header. Hence that the word following the header is the frame address.

[5] In contrast to the other length fields this length field is three bytes wide

Table 2.3: ICAP pins

Name	Direction	Description
CLK	Input	Clock signal
CE	Input	Config. Enable, set 'high' to pause reconfig. operation.
Write	Input	Read, write signal write='0' read ='1'
BUSY	Output	Signal, asserted 'high' when re-configuration is pending
O[7:0]	Output	Config. data output (read)
I[7:0]	Input	Reconfig. data input (write)

2.7 ICAP

The ICAP is one of the interfaces to the re-configuration logic. The other interfaces are: JTAG, serial slave and select map. The ICAP is the only interface that allows on-chip re-configuration (i.e. re-configuration controlled from within the chip).

The ICAP interface is a subset of the select map interface. The ICAP does not allow a complete re-configuration and there fore does not need some of the pins the select map interface is equipped with.

Both select map and ICAP allow re-configuration data to be written to or read from the re-configuration logic in byte wide portions. In addition to the data bus, both interfaces are equipped with some pins used to control the re-configuration process. The pins of the ICAP interface are described by means of table 2.3.

The ICAP interface is based on a handshaking principle. the CE and BUSY signal are used to control the data flow. The BUSY pin is asserted when the buffer is full. When the BUSY is asserted the data on the data-bus is ignored. The CE pin is used to pause the re-configuration process. Supplied data is only accepted when the CE signal is asserted and the BUSY signal is not.

There is a alternative way of controlling the ICAP, namely by keeping the ce asserted and using the clock pin to perform a write or read action. This method of controlling is based on the fact that a write or read action is performed on the falling edge of the clock signal if and only if the ce pin is asserted.

An important detail of the ICAP interface is that the re-configuration data needs to be bit swapped on a byte bases. Thus the MSB (Most Significant Bit) of a re-configuration byte is the LSB (Least Significant Byte) of the data-bus. This implies that bit 7 must be connected to bit 0 of the data-bus, bit 6 to bit 1 etc. Figure 2.2 gives a graphical representation of the swapping.

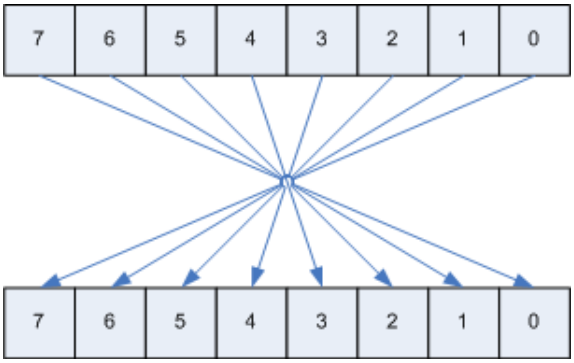

Figure 2.2: Bitswapping of a single byte

The ICAP can be instantiated as a primitive in a HDL based design. Furthermore Xilinx offers a core called the HW_ICAP. This HW_ICAP is a OPB (Onchip Peripheral Bus) device. The OPB interface allows the ICAP to be controlled by means of a microprocessor.

2.8 Support of Partial Re-configuration

Although some sources claim that partial re-configuration is also supported by non Xilinx FPGAs, documentation provide by other FPGA vendors regarding this feature is poor. Because of this reason the discussion is limited to Xilinx FPGAs only.

The MOLEN will likely be mapped to the Virtex IV or Virtex V FPGA in the near future. Because of that it is useful to know about the differences between these devices and the Virtex II pro.

The Virtex IV and Virtex V are both equipped with a ICAP interface similar to the one of the Virtex II pro. The width of the data-busses and the data-rate of these busses are the only differences among the Virtex series ICAP primitives. Thus it is rather easy to develop a generic solution compatible width all Virtex FPGAs.

2.9 Future Developments

Partial re-configuration is a feature which enables the development of self-reconfiguring systems. In nowadays FPGAs the feature is not mature enough to make these systems efficient. There are two major disadvantages to modern FPGAs supporting partial re-configuration.

Re-configuration latencies Re-configuration of a FPGA is relatively slow and will

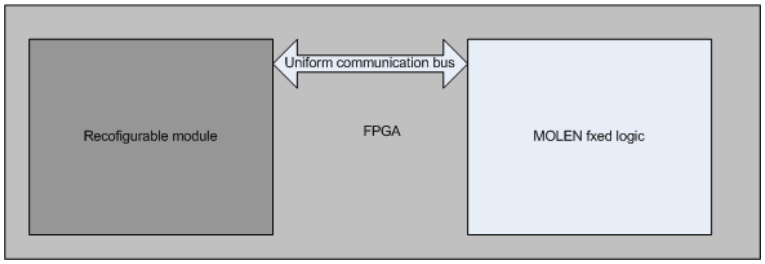

Figure 2.3: Single configurable module concept

therefore delay the system significantly. It is possible that the execution time diminish by the dedicate hardware is lost due to the re-configuration delay.

Array organization The array of most modern FPGAs is not uniform enough to easily implement re-location of hardware units.

To fully exploit the RPR concept FPGAs needs to be improved especially regarding the mentioned RPR limiting issues.

Whether future FPGAs will be designed for RPR systems depends mainly on the economic benefits for FPGA manufacturers.

2.10 Design proposals

Study of the Virtex II pro device and discussion with people of the CE group resulted in three design proposals. This section handles the advantages as well as disadvantages of the each proposal. Furthermore a tradeoff is included, based on this tradeoff a plan is made for the functional design.

2.10.1 Single Re-configurable Module

The most basic implementation of RPR is a "Single configurable module" design. This design is based on a two module approach, where one module is fixed(MOLEN framework) and the other one is re-configurable (RP) see figure 2.3. Note that in this discussion the term unit and module are considered to be synonym.

The re-configurable module and fixed module are equipped with a fixed interface. The re-configurable module can be swapped with any other module supporting the same interface and which is mappable to the area reserved for the re-configurable module. Figure 2.4 shows the module swapping principle.

The main advantage of the described concept is the simplicity, which implies that it is rather easy to implement. However the concept lags in flexibility, because there

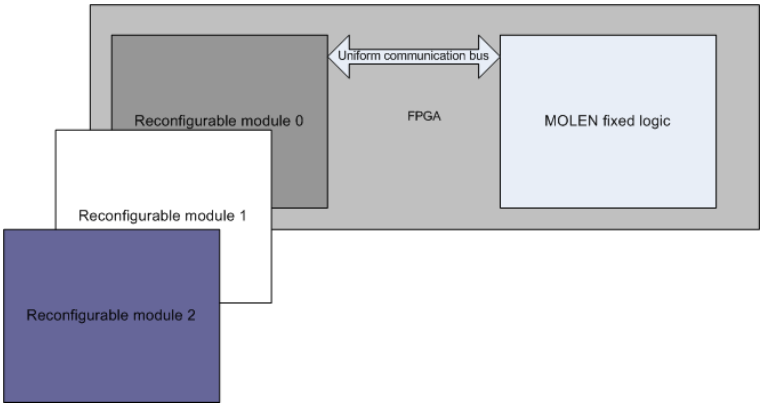

Figure 2.4: Module swapping, conceptual view

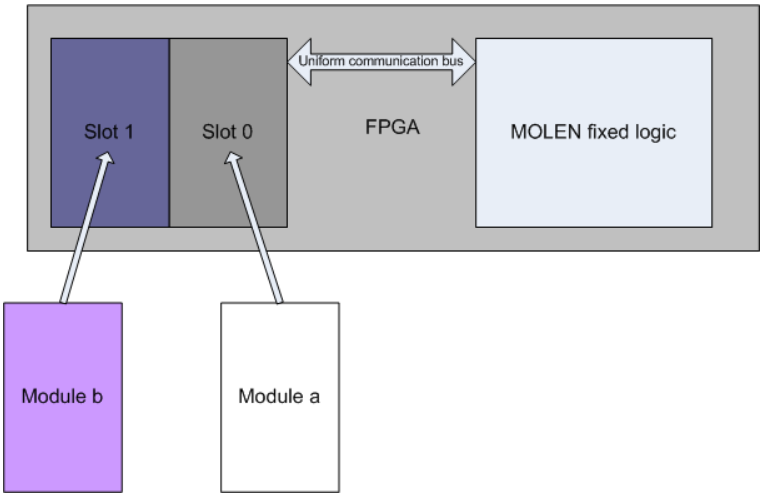

Figure 2.5: Multiple non-relocatable module concept

can be only one module present at a time. Every time the MOLEN needs another module a time consuming re-configuration needs to be performed. This can contribute to a significant decrease of performance especially when two modules are used in an alternating order.

2.10.2 Multiple Re-configurable Non re-locatable Modules

A more flexible approach is by offering multiple re-configurable areas instead of one. In contrast to the earlier one, this concept enables swapping one module while the other one(s) remain(s).

In this conceptual discussion we limit the number of independently re-configurable areas to two. These re-configurable areas are called slots throughout this document. Both slots can contain one single module. A slot can be re-configured without interfering

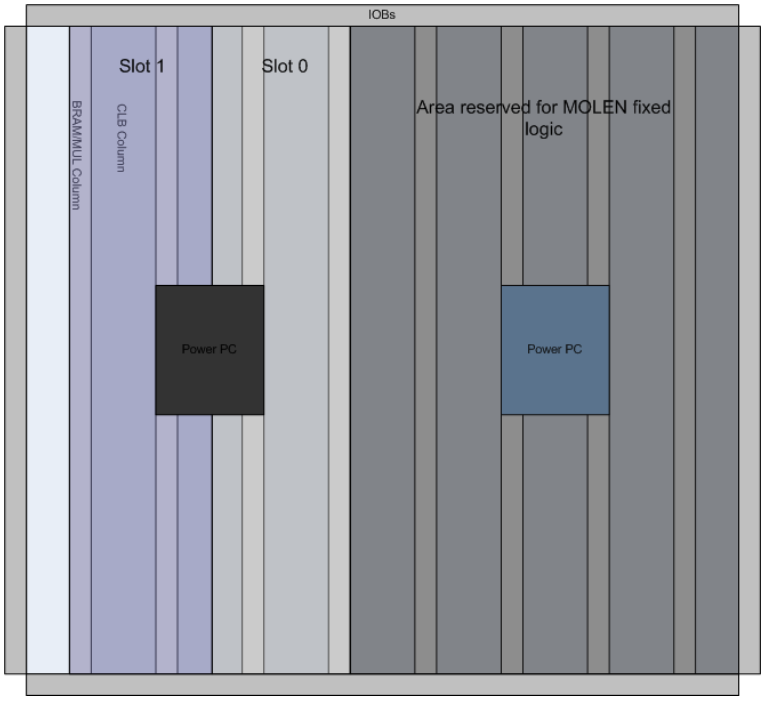

Figure 2.6: Multiple non-relocatable module floorplan

with the other module. Figure 2.5 gives a graphical representation of this concept.

Due to the fact that the module bitstreams are location fixed a module can only be mapped to one of the slots. The slot a module is mapped to can only be determined during design time and not during run-time. One can overcome this restriction by generating a separate bitstream for each slot. In the current discussed concept this means that two bitstreams per module are required. When increasing the number of slots the number of bitstreams increases linear, because each module must have a separate bitstream for each slot. This work around results in multiple bitstreams all serving the same function. Because these bitstreams must all be present in memory it is obvious that this work around dramatically decreases the memory usage efficiency.

Due to its physical architecture the FPGA can only be configured frame by frame, each frame spans the complete device height. Based on this observation the proposed slots can at best be aligned vertically, making the slots independently re-configurable.

Although the advantages of this choice is obvious it implies a new problem. Due to the very nature of module based design, modules can only communicate to modules adjacent to them self. By aligning the slots vertically the slot adjacent to the fixed logic isolates the other slot from the fixed logic as shown in figure 2.6.

The problem immersed in the preceding paragraph can be tackled by extending the

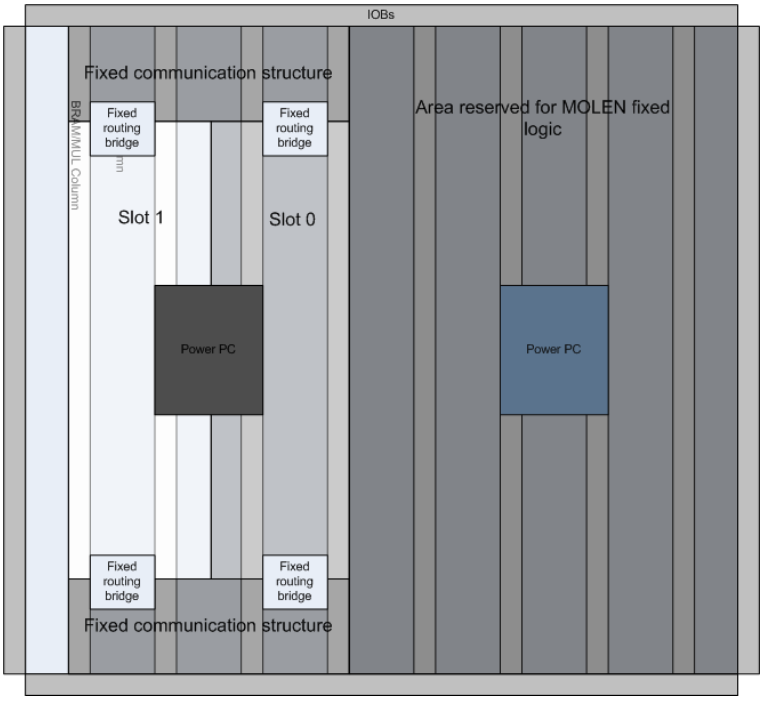

Figure 2.7: Communication structure

fixed logic by a communication structure interconnecting the slots and the fixed part of the MOLEN. Figure 2.7 shows this principle.

During and after re-configuration the communication structure must remain unchanged. Due to the fact that re-configuration is glitch less the logic can safely be overwritten by the original configuration leaving the logic unharmed.

The just mentioned principle can be used to prevent the communication structure from being influenced during re-configuration. The principle can be used by including the original configuration data of the communication structure vertically adjacent to the slot area within the partial bitstream.

The concept discussed in this section enables swapping on a module bases, although the description is based on two slots the concept can easily be extended to support more slots. The main limitation of the concept is that a module can only be mapped to a specific slot. Random placement of a module can be enabled by generating a bitstream for each slot. This method implies an significant increase of bitstreams to be stored in the main memory, which has a negative influence on the memory usage efficiency.

2.10.3 Multiple Re-configurable Re-locatable modules

The multiple re-locatable module concept is a extension of the previous discussed one. The currently described concept is based on the slot based architecture discussed earlier.

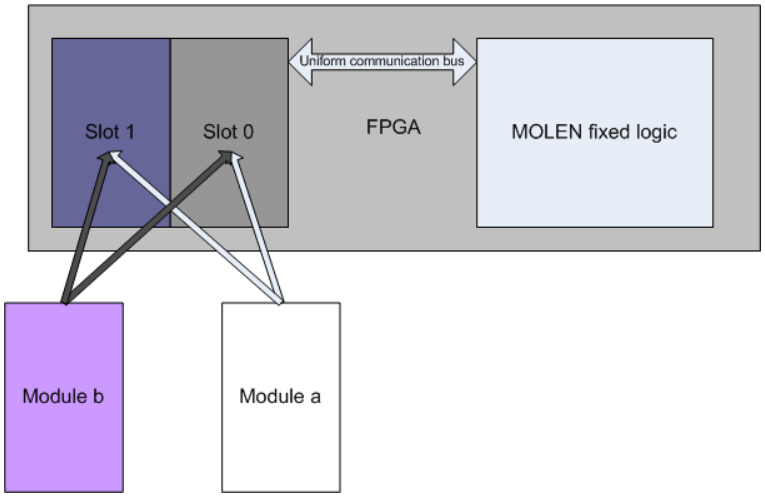

Figure 2.8: Multiple relocatable re-configurable modules

The Multiple re-locatable module concept enables a module to be mapped to any slot. The re-locatable mapping is realized by performing bitstream manipulation.

Re-locatable mapping by bitstream manipulation requires slots that are completely homogenous. To create a significant number of homogenous slots the current MOLEN floorplan needs some changes. The proposed changes are graphically represented by Figure 2.9.

Figure 2.9 shows that the FPGA organization is a limiting factor while planning homogenous slots. An other FPGA could be more convenient.

Other slot organizations could be proposed but the one described is the most feasible option. Due to the fact that the array can only be configured per frame it is beneficial to keep the slots narrow (horizontally seen). Because narrow slots contain less frames the re-configuration latency is minimized.

Furthermore multiple slots in the vertical direction will increase the complexity of a re-configuration operation, because overwriting of the other slots vertically adjacent to the one re-configured must be prevented.

This issue can be tackled by reading out the entire vertical area first. After that merging the read out data with those of the module to be replaced. By writing the merged data back only the intended module is changed This sequence of read, merge and write back is too time consuming to be effective. Figure 2.10 shows an other though less efficient floorplan.

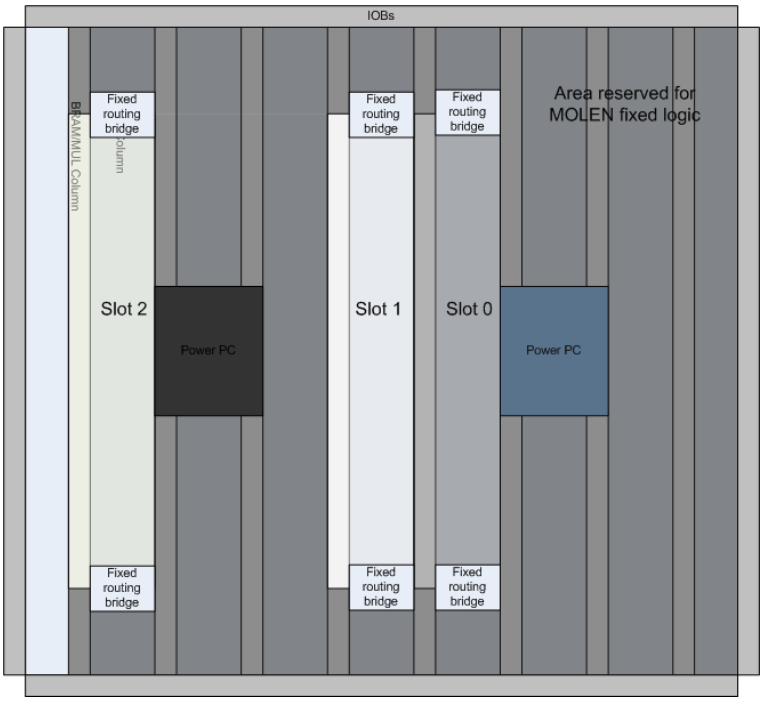

Figure 2.9: Floorplan modification proposal

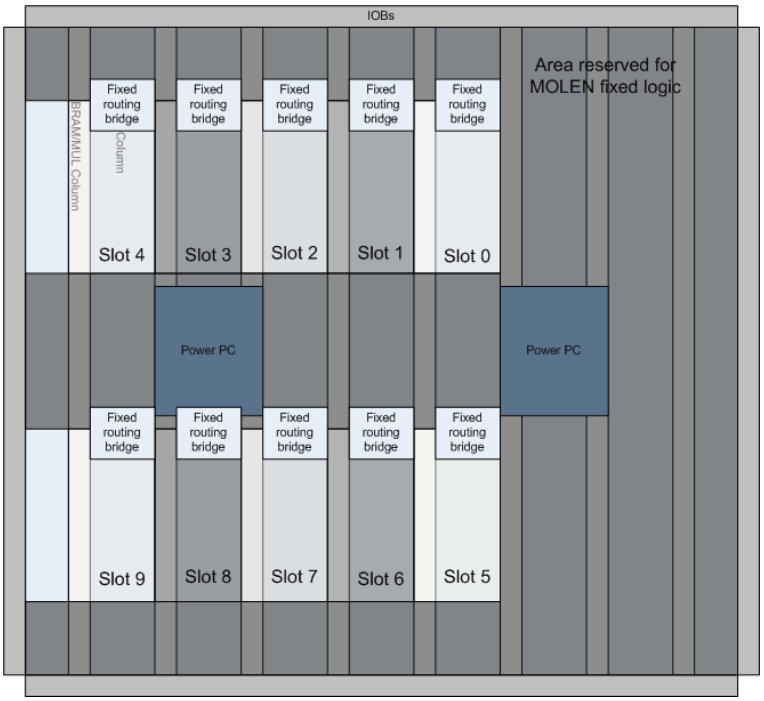

Figure 2.10: Inefficient floorplan proposal

Table 2.4: Score table

Proposal	**Design complexity**	**Flexibility**	**Performance**	**Appraisal**
Single	+	-	-	Useful
Multi non-relocatable	+	+	+	Useful
Multi relocatable	++++	+++	+++	Final goal

2.10.4 Trade Off

In this section three design proposals are discussed, namely "Single re-configurable module" figure 2.3, "Multiple non-relocatable re-configurable modules" figure 2.5 and "Multiple re-locatable re-configurable modules" figure 2.8. The advantages and disadvantages of each proposal has been discussed. A trade off needs to be performed to determine the best solution. A rating table is used for this tradeoff, see table 2.4.

By means of table 2.4 the three design proposals are rated. The proposals are rated on implementation complexity, flexibility and performance. Furthermore a total appraisal is given.

From table 2.4 we conclude that the design of both the *single module* and the *Multi non-relocatable* are equal in complexity. Because the *single module* design lags in flexibility and performance, mainly due to constant swapping, it is given a lower appraisal. The single module design is however a useful intermediate stage during the development.

The *Multi non-relocatable* leads to more flexibility and performance with about the same amount of effort. A disadvantages of the this design however is the necessity to schedule the mapping of modules during compile time. The *Multi relocatable* overcomes this disadvantages because the mapping is dynamic (i.e. the destination slot is run-time determined).

The multi re-locatable design enables the modules to be mapped to a free slot, if there are no free slots left a replacement algorithm will determine which other module must be dropped. Thus the dynamic or re-locatable mapping implies an administration of slot usage to be kept, as well as the implementation of a replacement algorithm. This will increase the design complexity significantly.

In conclusion we state that the *Single Module* option is a useful intermediate stage, the *Non-relocatable* option on the other hand is considered a valuable addition to the MOLEN framework. In fact some members of the Computer Engineering group can use it in there current research, the *relocatable* option is the by far the most advantaged application of *Partial Re-configuration.*

2.11 Functional Design Strategy

Based on the tradeoff a plan or strategy for the functional design can be made. As discussed in the preceding sections at first a one slot solution is implemented. In the second phase this implementation is extended to support multiple slots. The last phase of the design implements re-location of units.

The main advantage of the three phase approach is a spreading of complexity. Furthermore the second phase implementation is not just a intermediate version, but is actually useful for other MOLEN project members.

3 Specifications

In both the plan of approach and the introduction, we have briefly defined the purpose of the logic to be designed. In this chapter we are defining some key requirements, in order to determine some design consideration. Based on the design considerations the functionality is divided among a few functional units.

The remainder of this chapter is organized as follows. Section 3.1 discusses the functionality of the PRMU, this section gives a definition of the PRMU. In section 3.2 the initial demands are given, these are demands the design must comply to in order to enable integration into the MOLEN Framework. The initial demands lead to a few design considerations discussed in section 3.4. Based on the design considerations a more detailed definition of the PRMU is given in section 3.5. The maximum data rates of some key interfaces are discussed in section 3.6, while the storage of the bitstream in the repositories is discussed in section 3.7. The last section discusses the interface of the CCU, this interface has no relation with the PRMU design. The interface is discussed because it determines the signals of the communication structure between the fixed and re-configurable parts of the design.

3.1 Functionality

Based on the conclusions of the preceding chapter, a draft diagram is made. This block diagram is shown in figure 3.1. The PRMU (Partial Re-configuration Management Unit) functional block covers the main functionality of this thesis work. The other blocks are either hardware components or IP-cores.

The PLB (Processor Local Bus) including related components are used to access the off chip DDR-memory. The PLB is composed of Xilinx supplied ip-cores, these IP-cores can be instantiated without significant effort.

The key functionality of the design is: Partial re-configuration of the FPGA using configuration data from a storage device. The storage device can either be implemented using internal FPGA resources or physical memory external to the chip.

3.2 Initial Demands

There are four demands tot the design, these demands are a result of the implementation platform chosen and the foreseen integration within the MOLEN Framework.

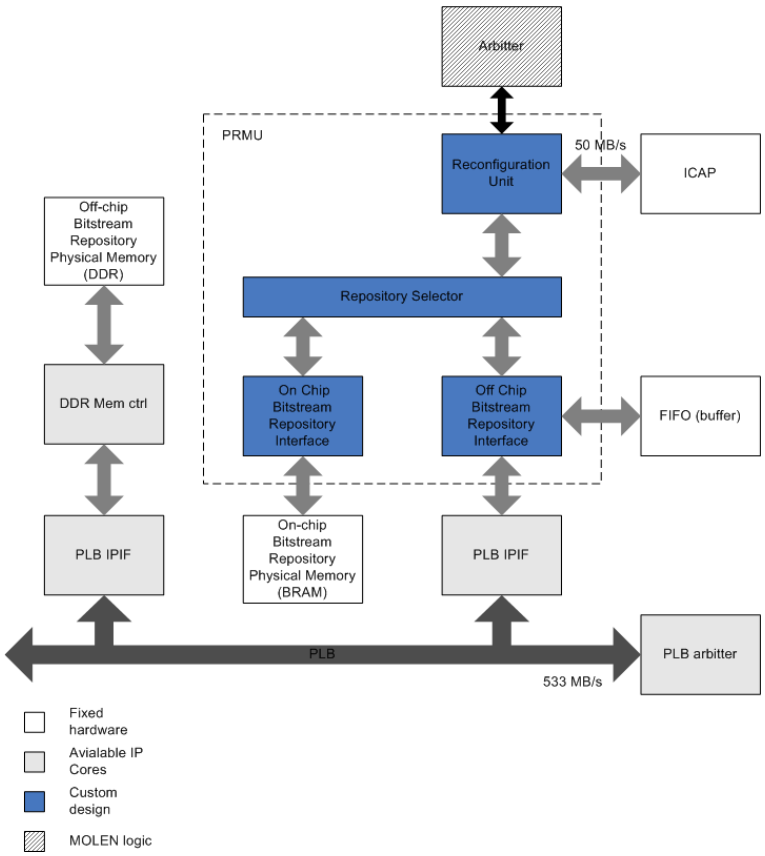

Figure 3.1: Total design simple version

- The implementation target is the XUP (Xilinx University Program) Board. The XUP Board is equipped with a Virtex 2 pro FPGA, type V2P30. The board contains a few other chips and expansion slots, the most relevant of these is the DDR (Double Data Rate) Memory.
- The On-chip Repository must be implemented using the FPGAs BRAM components.
- The Off-chip Repository must be implemented using DDR memory available on the XUP (Xilinx University Program) board.
- Interfacing with the DDR memory must be done using the PLB (Processor Local Bus).

Furthermore performance, re-configuration latency in particular, is an important issue. The design should enable re-configuration at the maximum speed. The design should also be portable to the Virtex 4 device, because this devices has several advantages a higher re-configuration data-rate is one of them. The following concrete

demand is extracted from the preceding discussion.

- The design must enable re-configuration at the highest possible data-rate on both the Virtex 2 Pro and the Virtex 4.

3.3 Memory Demands for Bitstream Storage

As mentioned earlier the bitstream must be stored in memory. Based on the specifications in the Virtex 2 Pro datasheet [1] we can estimate the required amount of memory.

According to the Virtex 2 Pro datasheet [1] a complete bitstream contains approximately $12 * 10^6$ bits. In the two slot design proposed in chapter 2 a single slot covers about a quarter of the FPGA area. Thus a bitstream for such slot contains about $3 * 10^6$ bits. This implies that storing a bitstream requires at most 366 KB.

The Virtex 2 Pro, type XCVP30 contains a total amount of 2,448 Kb or 306 KB. Thus the on-chip memory resources are limited, moreover it is not even enough to store a single bitstream.

In the current prototype the bitstream address is 24 bits wide, allowing a addressing range of $2^{24} = 16,777,216$. In case of the 64 bits addressable DDR memory this means that 428 bitstreams can be stored as proven by the following calculations.

The total addressable bytes:
$16,777,216 * 8 = 134,217,728B = 131072kB$
The total number of bitstreams:
$\frac{131072}{306} = 428$

3.4 Design Considerations

Based on the available resources and resources used by the current MOLEN prototype an on-chip and off-chip Storage of re-configuration data is proposed, from now on referred to as repositories. Besides the two supported repositories, the design must be extendable to support other repositories as well.

In the initial design the re-configuration data is directly written to the ICAP. In order to implement the relocation (see chapter 2) there is a need to modify the re-configuration data. A unit, either software or hardware, needs to be developed capable of performing this process.

From the preceding two paragraphs we can derive two main design considerations.

Easy integration of repositories The design must be extendable to support other storage devices to function as re-configuration data repositories.

Easy integration of bitstream processors The design must enable the integration of elements processing the bitstream.

There are two options for implementation of on-chip re-configuration. One can either implement a full hardware design or use the OPB_HWICAP to control the reconfiguration process. The OPB_HWICAP wraps the ICAP so it can be used as an OPB device. Using this OPB device requires a micro-processor fetching the re-configuration data from memory and sending it to the OPB_HWICAP.

Typically software solutions are more flexible, while hardware solutions often achieve a better performance. Especially bitstream manipulation is easily implemented using a micro-controller. There is however a price to be paid, the micro-controller based solution lags in performance due to overhead caused by bus transactions. Furthermore the performance will decline even more when the number of devices connected to the shared bus increase.

An other disadvantage of the micro-controller based approach is the resources usage. The micro-controller and the various bus component will most likely use much more resources than a dedicated hardware solution.

Because both performance and resources usage are two key parameters, we propose a dedicated hardware solution. The process of bitstream manipulation outlined in chapter 2 is rather straight forward, implementing it in hardware should be feasible.

3.5 PRMU Definition

The PRMU (Partial Re-configuration Management Unit) is the collection of components controlling the re-configuration process see figure 3.1. The PRMU is composed of the re-configurator the repository selector and the repository interfaces, the physical memories are not part of the PRMU.

The re-configurator, controls the ICAP (Internal Configuration Access Port). The re-configuration data is retrieved from a repository via the repository selector. The repository selectors task is to select the appropriate repository. It's a multiplexer which redirects the data from one repository to the re-configurator.

A repository interface functions as adapter adapting the data format and protocol to the transfer protocol used within the PRMU.

In conclusion we state that the PRMU is a functional block which controls the partial re-configuration. The block is controlled by the arbiter, the arbiter triggers

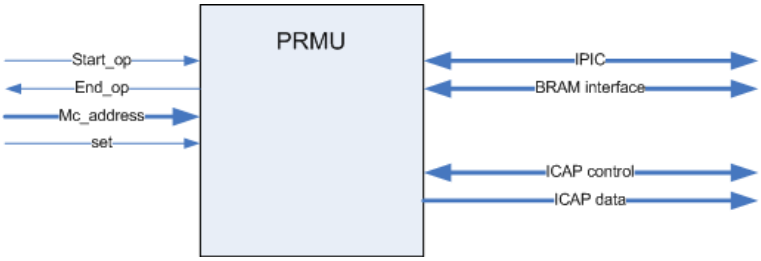

Figure 3.2: PRMU primitive

Table 3.1: Data rates

Connection	**Data rate**
PLB	533 MB/s
PRMU to ICAP	50 MB/s

the PRMU to perform a partial re-configuration. The arbiter also provides a pointer to the re-configuration data (i.e. a memory address). When the arbiter requests a partial re-configuration, the PRMU reads the re-configuration data from one of the two repositories (memory) and writes it to the ICAP.

The preceding describes the primary functionality of the PRMU. In order to enable dynamically placement to a free slot (i.e. Re-location), the design needs to be extended.

The extension needs to keep track of the slots occupation and needs to implement a replacement algorithm. Furthermore bitstream manipulation must be performed by the extension.

3.6 Data Rates

This section discusses the data rate of all the connections between the ICAP and the DDR-memory. By means of this analysis bottlenecks can be discovered.

From table 3.1 the conclusion is drawn that there is a significant difference in data rate between the IPIC bus and the ICAP data bus. For this reason a FIFO is implemented within the PRMU to buffer the data. The data is transferred through the PLB in 16 words long bursts. For this reason the FIFO size is specified to buffer two bursts. The FIFO size (two bursts) prevents the FIFO from becoming empty, leading to an increase in efficiency.

The PLB is a shared bus, which implies the given data-rate is in fact the total bandwidth. The achievable data-rate when fetching the bitstream depends on the number of devices and there load on the bus. The actual data-rate is much lower than then given value. It is not unlikely that it temporally drops under 50 MB/s.

Table 3.2: Bitstream Storage Format

Length Field (64 bits)
Actual Bitstream
...
...
...
...

3.7 Bitstream Storage

The bitstream stored in a repository exists of a bitstream generated by a Xilinx tool preceded by a length field. This length field is a 64 bit wide word containing the length of the bitstream in binary unsigned format. The length field contains the number of 32 bits words the bitstream is composed of. Table 3.2 shows the storage format in a tabular form.

3.8 CCU Communication Structure

CCU (Custom Configurable Units) need to comply to a certain interface. This interface is described in the following paragraph. The interface specified is the interface between the MOLEN framework and the RP or CCU.

The communication infrastructure is a set of buses, serving as interface between the CCUs and other MOLEN components. The communication structure is composed of four busses. Namely the *clock and reset*, *micro-code*, *exhange register* and *memory* bus.

The *clock and reset* bus provides three clock signals and a reset signal to the CCUs. The *micro-code* bus is a control bus, the CCUs are controlled through this bus by means of micro-instructions. The *exchange register* bus is an interface for argument exchange between the CCU and the PowerPC. For large blocks of data the CCU can access the memory through the *data memory* bus. The communication structure is specified in detail in appendix B.

4 Functional Design

This Chapter describes the functional design of the hardware capable of performing RPR (Run-time Partial Re-configuration) of CCUs (Custom Configurable Units) also referred to as RPs (Re-configurable Processors) and its integration in the MOLEN framework.

The description of the design is done by using a top-down approach. The discussion starts at the top-level and narrows down towards the level of basic blocks (e.g. registers, multiplexers et cetera) commonly used in digital design. The design covers the functional behaviour of the system, the actual implementation might defer due to optimisations and restrictions.

The chapter is organised as follows. In section 4.1 the Top-level is described. The designed data bus is discussed in section 4.2. Section 4.3 Discusses the top-level components of the design, the low level components are also discussed. The repositories are discussed in a separate section namely 4.4. The last section discuses in brief a tool designed to generate memory initialisation files. These initialisation files are used to load the bitstreams in the on-chip repository.

4.1 The Top-level

The top-level contains three main parts namely the re-configuration unit, controlling the re-configuration process, the repository selector which selects a repository (i.e. memory) and lastly the repository interfaces serving as adapters. The selector selects a specific repository based on the address supplied by the arbiter. The address is the start address of the bitstream (i.e. the re-configuration data). The selector selects the repository in which the bitstream is stored, there can be multiple repositories. The current design features two repositories, a small on-chip repository and a larger off-chip repository.

The re-configuration unit writes the data supplied to its input ports to the ICAP described in chapter 2. The width of both the input and output data bus is generic and can be adapted. The input is standard defined to be 32 bits wide while the output data is 8 bit wide. The input is 32 bits because the bitstream is 32 bits oriented. The 8 bits output is compatible with the Virtex II Pro ICAP, although it can easily be changed to conform to the 32 bits ICAP of the Virtex IV.

A specific data bus is used between the repository interfaces and the re-configuration

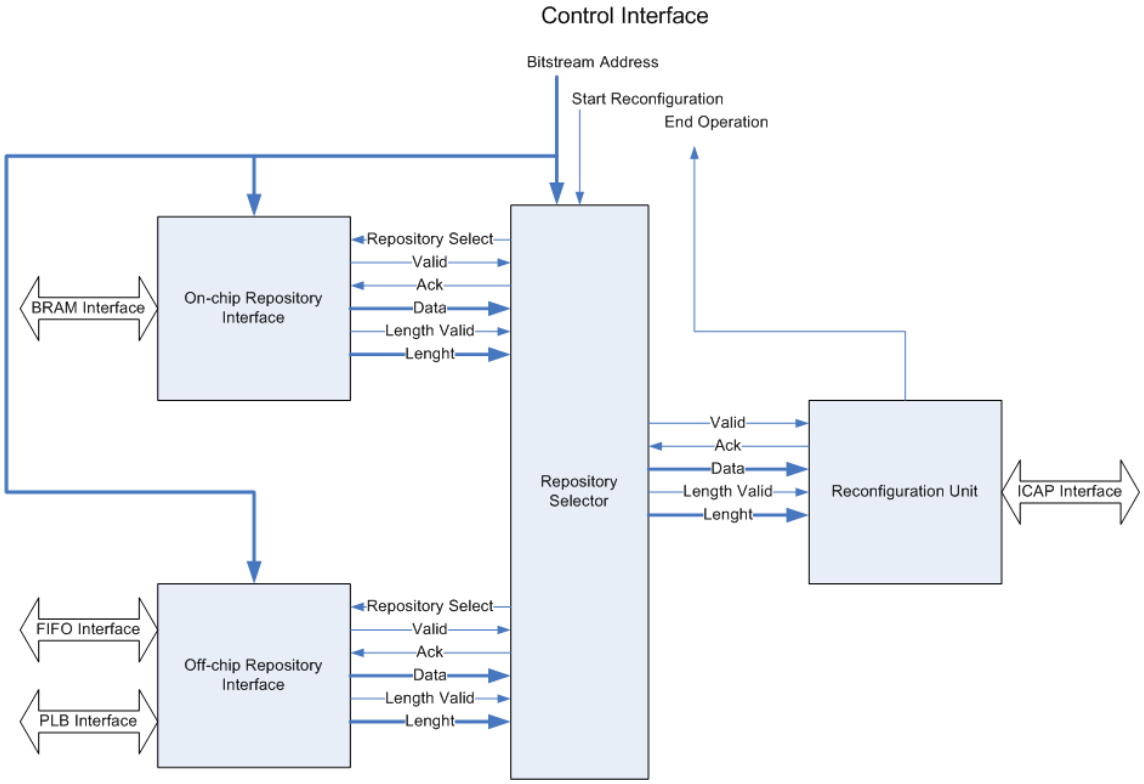

Figure 4.1: Schematic overview of the Top-level logic

unit. The repository selector functions as multiplexer of this bus. The data bus behaves like a hand shaking FIFO, more details about the protocol are given in section 4.2.

The Repository selector can be extended to support a virtual infinite number of repositories. A specific repository is selected based on the address, the selection will be updated when the start re-configuration signal is asserted. The repository selector functions basically as a multiplexer.

4.1.1 Interfaces of the Top-level

The PRMU top-level has three distinct interfaces. The PRMU module interfaces to the arbiter of the MOLEN Framework, to one or more repositories and to the ICAP primitive.

The interface to the the arbiter is meant for control purpose, through this interface the arbiter instantiates a re-configuration. The bitstream is read from one of the repositories, using the repository selector. The third interface is used to control the ICAP.

4.2 Data Bus Protocol

The data bus protocol has been mentioned multiple times in the preceding paragraphs, although the actual operation and possible implementation was not covered. The specific databus, discussed in the following paragraph is design to fulfil the earlier mentioned demands. Namely

1. Easy integration of repositories.
2. Easy integration of processing elements.

In order to fulfil the first demand the necessary adapter logic, in order to comply to the protocol, should be minimal. Furthermore the protocol should be able to pause the data stream. The pause feature is a effect of the second demand and prevents a processing element form choking.

This section discusses the design of the data bus protocol in depth, including a justification. At first the Principle is discussed using a trivial analogy. Furthermore a feasible implementation is proposed, lastly the choice for the type of data bus protocol is justified.

4.2.1 Data Bus Principle

In this section the datapath protocol is discussed in more detail. As mentioned earlier the data-path is constructed using a hand-shaking principle. The advantage of this principle is that it enables us to construct chains of components, each capable of performing actions on the passing Bitstream (e.g. components checking the Bitstream or components manipulating the Bitstream).

The principle is comparable with a row of persons, passing on packages along that row from one end to the other. A particular person in the row can receive a packages from the preceding person and pass it on the the next. By doing so the packages propagates from one end of the row to the other.

Its clear that a person in the line can only accept a package when one is currently not holding a packages. A person holding a packages offers the package to the next person in line, hence the analogy with raising a valid signal. A person not holding a package offers his hands towards its predecessor, which is similar to assertion of the acknowledge signal.

4.2.2 Data Bus Implementation

How can we implemented the principle described in the previous section? We can do so using standard D-flipflops with some additional logic to generate the valid and acknowledge signals. Figure 4.2 shows this principle. The registers can be cascaded in order to construct a chain, through which a data word can propagate. The registers holding the data are standard D-ff, featuring an additional enable signal. As long as

the enable signal is asserted, the data supplied to the input is stored. A store operation is performed on the rising edge of the clock signal.

This signal is used to store the data on the input in the register. The control logic controls this enable signal.

The valid signal indicates, whether the data on the input is valid. When the register content is not valid (i.e the current content is already acknowledged by the next chain element) or the register has not jet contained valid data (i.e. initial state) then the enable signal is asserted. The effect is that the data appears on the output during the next clock-cycle, the valid signal is asserted to state that the data is valid.

In the preceding paragraph we assumed a single word propagating through a chain of registers, but the chain is also capable of handling a multiple word burst. In the case of a burst the register to register hop takes a single clock-cycle. Thus a sequence of words propagates as fast as a single word.

The maximum flow through the proposed chain is one word per clock cycle, there is however a initial delay before the first word appears at the chain output. This initial delay is proportional to the number of elements within the chain, when considering bulk transfers the initial delay is not of any relevance to the performance.

Until now we assumed the chain to exist of registers. In the design however we use multiple components in the chain. Furthermore it should be possible to add devices in the chain capable of processing the bitstream. These bitstream processors (e.g. bitstream manipulator) could require more than one clock-cycle processing time, harming the maximum flow significantly. We can overcome this issue by increasing the clock-frequency of the chain elements, until we achieve the required throughput.

Figure 4.3 and 4.4 show two wave forms of possible transfers using the handshaking principle. The former shows a start of a multiple word burst, while the later shows the end of such transfer. The protocol functions at 50 MHz, in figure 4.3 the time between two vertical lines is 20 ns, while in figure 4.4 this time slot spans 10 ns.

The transfer start shown in figure 4.3 begins when the valid signal is asserted, the valid data is asserted simultaneous. After some time the acknowledge signal is asserted. The next word is made available exactly one clock-cycle after the acknowledgement. The valid signal remains asserted in this situation.

In figure 4.4 the valid signal is de-asserted after the second acknowledgement. This means that there's no valid data at the moment. The valid goes 'low' exactly one clock-cycle after the assertion of the acknowledge signal.

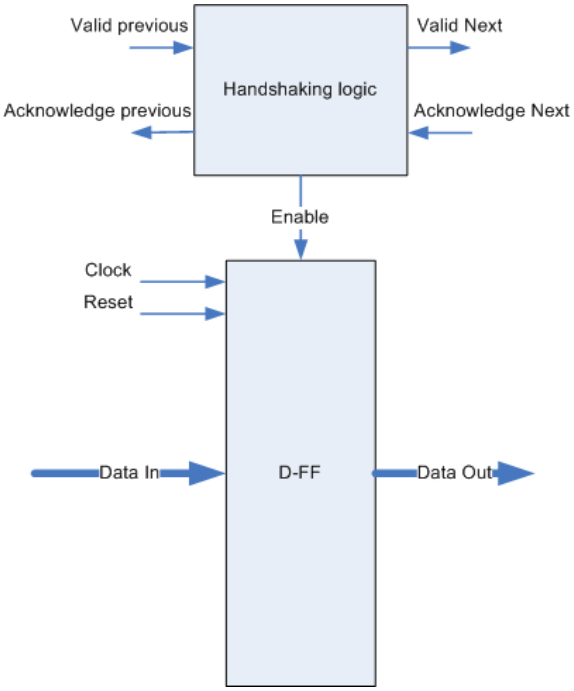

Figure 4.2: Possible implementation of the Handshaking Register

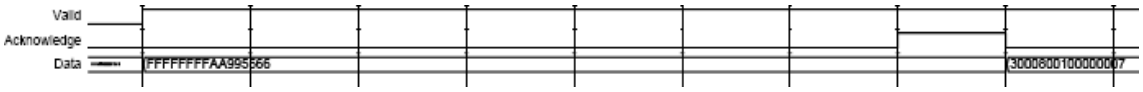

Figure 4.3: Handshaking protocol, start of a burst

4.2.3 Justification

Based on the specifications given by chapter 3 we conclude that: Integration of new repositories in the design should be easy and the design should be augmentable with devices preforming actions on the bitstream (e.g. a bitstream manipulator to feature re-location of CCUs). Figure 4.5 shows how a bitstream processor can be inserted.

Both demands require a uniform data bus, the later demand requires a mechanism to suspend the data, in order to prevent the device from choking. The data must propagate through the bus automatically, we don't want data to stay inside the data bus when no new data is pushed in on the other side.

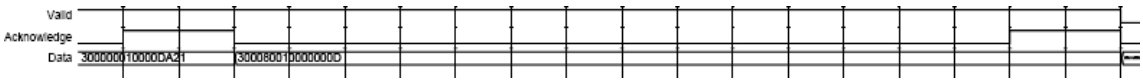

Figure 4.4: Handshaking protocol, end of burst

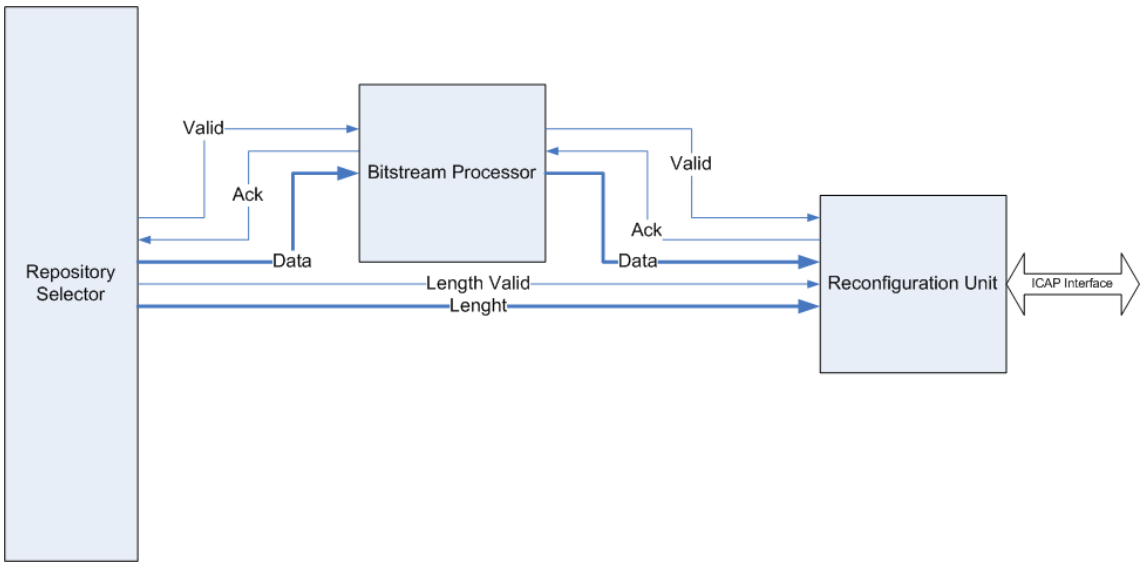

Figure 4.5: Insertion of a bitstream processor within the handshaking chain

The proposed hand-shaking data bus protocol fulfils all the demands. The Data Bus is uniform and a device can suspend the data stream by not acknowledging the received data to its predecessor. Any word will eventually appear at the chains output, there is no need for a flush action.

4.3 Top-level Components

The following sections discusses the mentioned top-level components in more detail. The re-configuration unit as well as the repository selector are described including their sub-components.

4.3.1 re-configuration unit

In the design we can address the re-configuration unit as being the sink of the handshaking chain. The re-configuration unit converts the data to a size suitable for the ICAP (8 bits when using the Virtex 2 Pro). Furthermore the re-configuration unit handles the control signals, it even covers the abort sequence used to cancel a pending re-configuration. This last feature is meant to end the re-configuration process when illegal activities have been detected.

The re-configuration unit exists of two main components, one managing the with adaptation of the supplied data and the other controlling the ICAP interface.

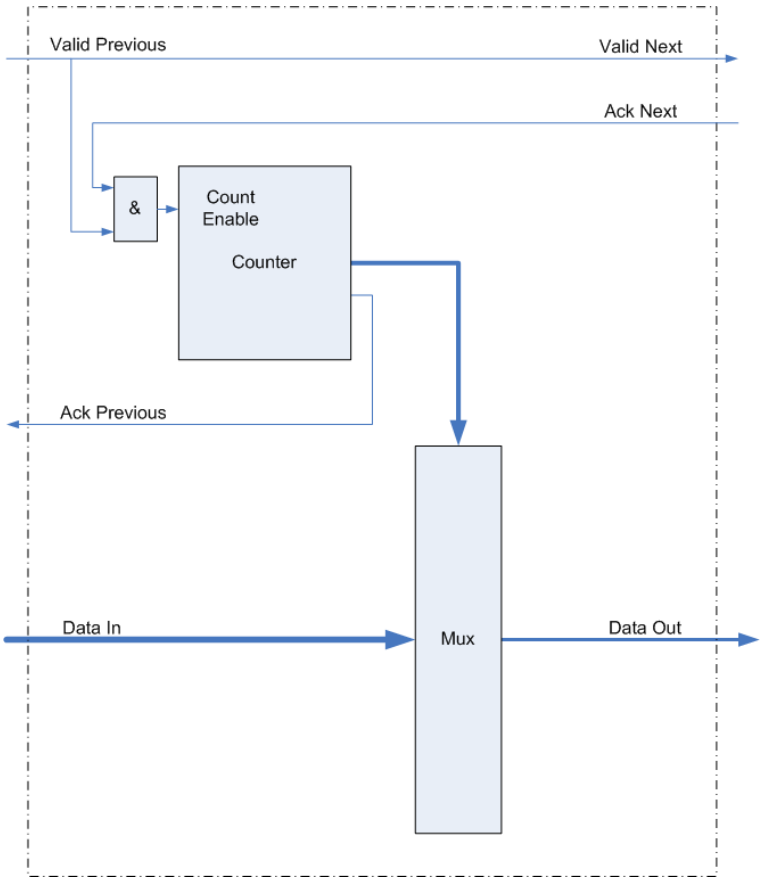

Figure 4.6: Possible implementation of the Width Adapter using standard components

4.3.1.1 Width Adapter

The width adapter is a special form of the discussed handshaking registers. The component puts the data on its output sub-word by sub-word. If the current sub-word is acknowledged the next one will appear during the following clock-cycle. When all sub-words are acknowledged the width adapter acknowledges the word driven on its input.

For example we consider the width adapter to have a 64 bits input and a 8 bit output. Initially the first byte of the 8 byte word (64 bits = 8 bytes) appears on the output. When the byte is acknowledged the next byte is selected, until all 8 bytes are acknowledged. On acknowledgement of the last byte the width adapter acknowledges the 64 bits word to its predecessor. When fresh data appears on the width adapters inputs, the described process repeats.

On the implementation level we can construct the width adapter using two common components, namely a counter and a multiplexer see figure 4.6. We use the counter value as the selection input of the multiplexer. The input data is driven to the input

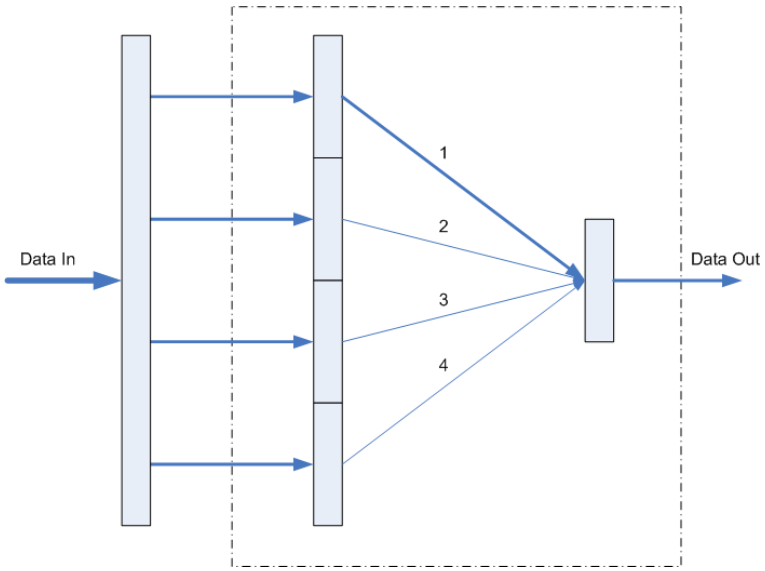

Figure 4.7: Abstract principle of the Widthadapter operation

of the multiplexer while the output of the multiplexer functions as output for the unit. The multiplexer has a number of inputs equal to the width of the input data, while the output has the width we desire. For proper operation the input width must be a divisor of the output width. The multiplexer can connect a specific sub-word to the output. The multiplexer selection is controlled by the value of the counter, thus by incrementing the counter value we can "walk through" the word in sequential order as shown in figure 4.7.

4.3.1.2 Re-configuration Controller

The second component of the re-configuration unit is the re-configuration controller. This unit is slightly more complex as the earlier described width adapter and is composed out of multiple sub-components to cover all the functions of this unit. A schematic block diagram of the re-configuration controller is shown in figure 4.8. Figure 4.9 gives a simplified state-diagram of the main controller.

The implementation of the re-configuration controller can be done using a few standard components. A multiplexer, counter and two controllers are incorporated in this component. The controllers are implemented using the finite state machine methodology.

One of the controllers referred to as the main controller functions as master within the unit and controls the other components either directly or indirectly. The second controller is the abort sequence generator, this component generates the "wave form" on the ICAP interface necessary to abort the pending re-configuration. This component is summoned by the main controller to generate the "abort sequence". The completion of this abort sequence is reported to the main controller.

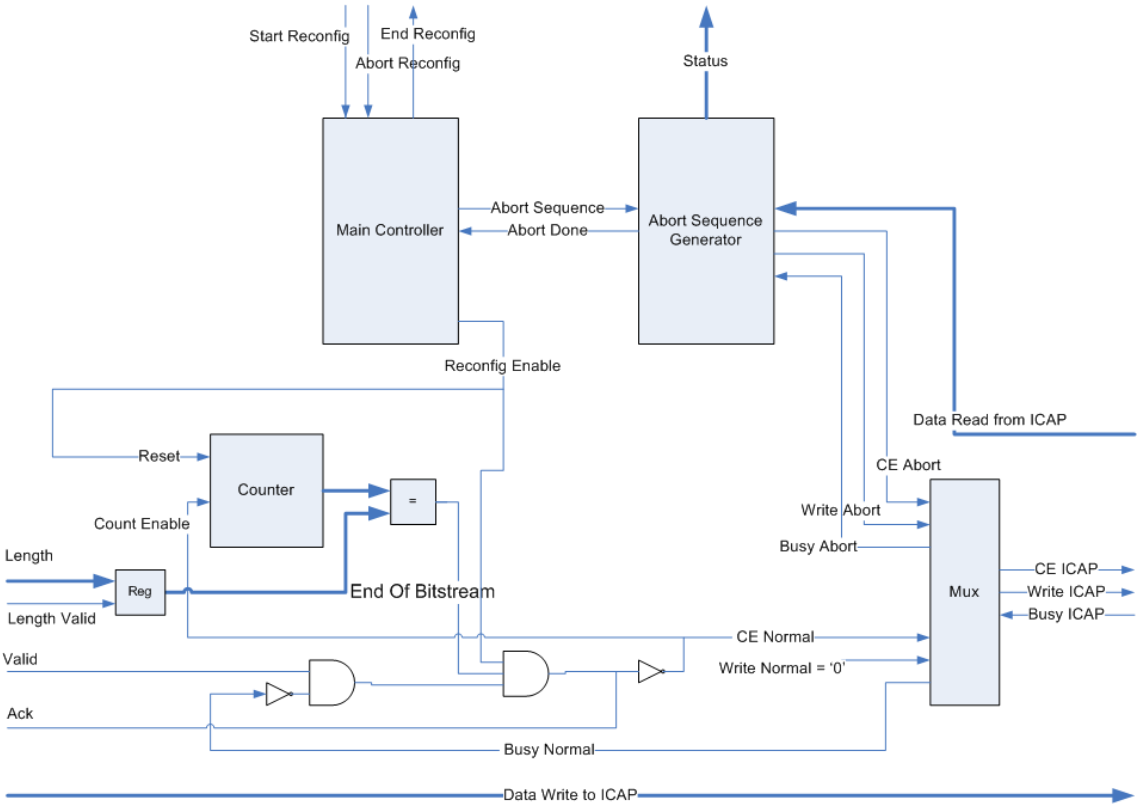

Figure 4.8: Schematic representation of the Re-configuration Controller

The main controller is used to perform an initialization procedure, after the initialization the re-configuration controller operates in normal re-configuration. In this state the data supplied through the handshaking chain is written to the ICAP. During the initialization state the length of the requested re-configuration is retrieved. This parameter is necessary to determine when the re-configuration is completed. When the re-configuration is completed the Main Controller asserts the End Operation signal and goes to the idle state, waiting for another re-configuration request.

During the normal re-configuration state the abort signal can be asserted by a external component. When this happens the main controller switches to the abort mode, the abort sequence generator is ordered to perform the abort sequence. After completion of this sequence the main controller goes to the Idle state.

A multiplexer is used to multiplex the ICAP interface to either the abort sequence generator or the "normal" operation data path. The multiplexer is controlled by the main controller.

We also mentioned a counter in one of the preceding paragraphs. This component

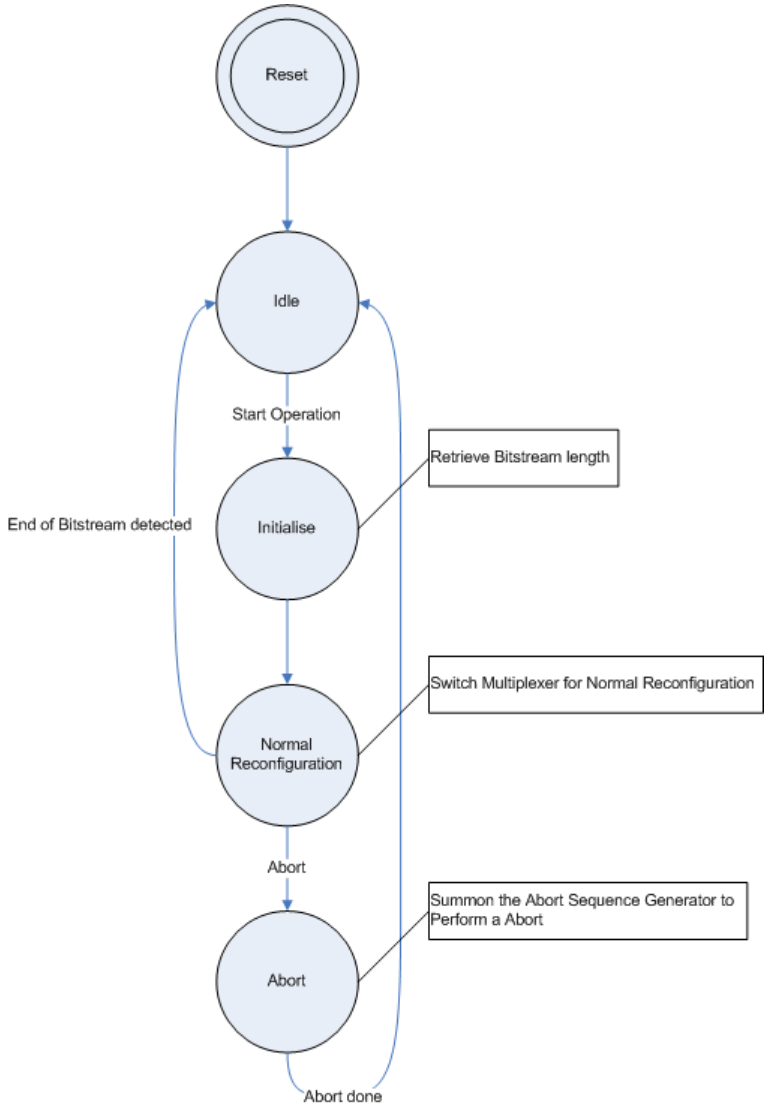

Figure 4.9: Simplified state-diagram of the Main Controller

is used to determine the end of the bitstream. The counter counts the number of words written to the ICAP and compares them against the length parameter retrieved in the initialization state. When the end of the bitstream is detected the main controller is informed, the main controller returns to its Idle state and signals the completion by asserting the end operation signal.

4.3.2 Repository Selector

As mentioned earlier the repository selector is used to select a source for fetching the bitstream. The selection is updated when a re-configuration is requested, the selection is

based on the supplied start address of the bitstream. The repository selector multiplexes the signals of a specific repository to its output.

The repository selector can be adapted to support as many repositories as resources allow. The number of repositories are usually limited, the current design offers two repositories. Despite the limited number of repositories an expandable principle is designed. In order to enable any storage device to be used as a repository. This implies that a repository must comply to the described handshaking protocol, which requires some additional adapter logic.

The XUP (Xilinx University Program) prototyping board, the implementation target of this project, offers various types of memory which could function as repositories. The current flexibility makes it rather easy to implement and integrate one of the memories as a repository. Although we only consider two repositories at the moment there might be a need to use one of the other repositories in future applications.

The current design features two repositories, an on-chip repository implemented using BRAM (Block RAM) and an off-chip repository using DDR memory. The BRAMs are components of the FPGA, which can be used to implemented various types of storage elements. The DDR is supported by the XUP board. Interconnection with the DDR is achieved through the PLB (Processor Local Bus), in order to enable sharing of the memory with other MOLEN components.

4.4 Repositories

A repository is composed of a storage element and some additional logic meant for initialization purpose. A repository offers a handshaking port to retrieve the bitstream and a separate bus to get the length of the bitstream. Thus the data transferred using the handshaking protocol is the raw bitstream, whithout any aditional data. The bitstream stored in the memory is composed of a header and the actual bitstream. The header is a 64 bits word representing the length of the bitstream.

The header (i.e. length field) and the re-configuration data (i.e. bitstream) are separated by the repository interface in order to achieve that only the actual re-configuration data is transfer by the handshaking chain. Thus any element in the chain receives data formatted according to the bitstream standard.

When selected, a repository must return the length field using the BSLength (Bitstream Length) bus and BSLengthValid signal. Extracting and returning the length parameter is called the initialisation stage. After the initialisation stage the re-configuration data can be fetched using the handshaking port of the repository. The repository will behave in a FIFO a like manner.

4.4.1 Repository Interfaces

A storage device functioning as bitstream repository must comply to a certain interface. This interface is specified as follows.

RepositorySelect : This signal is used to initialise the repository, the StartAddress must be valid when this signal is asserted.

StartAddress : The start address of the bitstream.

BSLength : The length of the bitstream.

BSLengthValid : Signal used to indicate that the BSLength content is valid.

Valid : Signal indicating that the content of DataOut is valid.

Ack : Signal to acknowledge the data supplied through the DataOut signal.

DataOut : The actual data, hence bitstream only.

In addition to a uniform interface the bitstream repositories must conform to a typical behaviour as described hereafter.

1. On assertion of the repository select signal the repository starts the initialisation or re-intialisation.

2. The repository returns the length parameter.

3. The first word is set on the data bus and the valid signal is asserted.

4. The bitstream can be retrieved through the handshaking port of the repository.

4.4.2 On-chip Repository

The on-chip repository of the design is implemented using BRAM (Block Random Access Memory). The BRAM are standard components of the Virtex II pro FPGA, used to implement storage devices. The BRAM has a one-clock cycle access time, thus re-configuration can be done at full speed. The main drawback of using BRAM is that this resource is limited.

The storage element used for the on-chip repository is in fact a single port memory, including a enable signal. This storage element is constructed out of a asynchronous memory and a register, the register is connected to the output of the memory in order to prevent timing problems. When asserting the enable signal the memory contents will be available at the next positive clock edge as shown in figure 4.10.

BRAM is random accessible memory. This type of memory is used often for FIFO implementation. A memory based FIFO is usually implemented using a part or the full

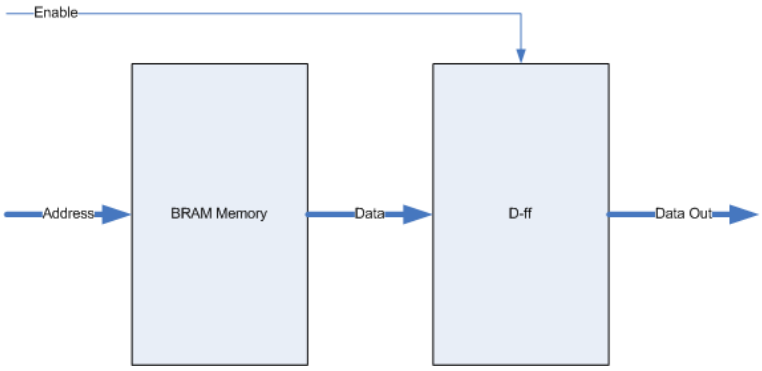

Figure 4.10: BRAM implemented single port memory

range of the memory. Furthermore two pointers are used, one being the write pointer while the other one serves as read pointer.

Initially the two pointers point to the same address. In this state the FIFO is empty. A word written to the FIFO is stored at the address the write pointer points to. Consequently the pointer is incremented. The read pointer points always to the first word written to the FIFO. The read operation is composed of putting the data addressed by the read pointer on the data out port and incrementing the read pointer. When one of the pointers meets the end of the allocated address range, the pointer is assigned to the first address of the range.

When both pointers point to the same address, the FIFO is either empty or full. The pointer is empty when a read operation caused the pointers to become equal and full when it is caused by a write operation. The preceding shows that a FIFO can be constructed using an addressable memory augmented with some additional logic.

The principles out lined in the preceding paragraphs are used to implement the on-chip repository. The on-chip repository only has to support read operations. Thus it will function as a "read only FIFO". As mentioned earlier the FIFO is configured during an initialisation stage. In this stage the length field is read from the repository. Based on the length the end address of the bitstream is determined (i.e. the range of the FIFO). After this stage the repository behaves as a FIFO.

As long as the read pointer is not equal to the end pointer the valid signal is asserted. When the acknowledge signal is asserted the read pointer is incremented.

4.4.3 Off-chip Repository

For implementation of the off-chip repository a DDR (Double Data Rate) memory is used. The implementation target, the XUP board enables DDR-memory usage. This DDR memory can be accessed by the FPGA. In the design a less direct method is

used to access the DDR, namely the PLB (Processor Local Bus). The main reason for accessing the memory through this multi device bus is to enable sharing of the DDR memory.

The PLB can transfer data at a significant higher rate than the ICAP can handle. Therefore the data is transferred using the burst capability of the PLB. Furthermore a FIFO is used to buffer the data transfer.

The main reason for using the DDR is that it offers a repository capable of storing a significant number of bitstreams. The bitstream storage capacity of this repository will most likely suite the needs for most of the current re-configurable designs.

Because the repository interface logic directly interfaces to a FIFO this logic can be rather simple. The only thing this logic must take care of is that the first word is directed to the BSLenght port, while the remaining words must be supplied to the DataOut port. Because the handshaking signals are similar to FIFO control signals this requires only some multiplexers and optionally some inverters.

Another feature of the off-chip repository is fetching of the data from the DDR-memory to the FIFO. The fetching of the data is described in the following section.

4.4.4 Off-chip Data Fetching

For interfacing the PLB bus we propose to use the IPIF (Intellectual Property Inter-Face), which is a interface commonly used to connect a IP to the PLB. The IPIF allows two types of data transfers namely. Single word transfers and burst transfers.

When a fetch is requested the length field is read first using a single word transfer. Based on the content of the length field the number of burst is calculated. A burst contains a fixed number of words.

The FIFO used as buffer must be capable of storing at least two burst. Furthermore a burst is started only when the FIFO can store the whole burst.

4.5 Memory Initialisation File Tool

When using the on-chip repository the BRAM implemented memory must be initialised with the re-configuration data. For this purpose an initialisation file is used. These initialisation files are composed of a ascii representation of the memory content. A tool is developed to generate such a file.

The tool takes as input multiple bit files and produces a coe file and a map file. The coe file is the initialisation file, while the map file reports the start addresses of the bitstreams and the required memory size.

5 Verification and Testing

This chapter discusses the means of both verification and testing. Verification is conducted on various levels of the design using VHDL test-benches and the ModelSim functional simulator. Tests are applied using the XUP board as target and the Chipscope toolbox. So we verification is the process of proving the correct operation by functional simulation, while testing is done using the actual prototype.

This chapter is composed out of three sections. The verification process is discussed in section 5.1, while the performed test are covered by section 5.2. The last section describes the test performed to measure the re-configuration latency. The results of these tests are also incorporated in this chapter.

5.1 Testbench Driven Verification

A testbench is used to verify the proper operation of the components referred to as PRMU. The testbench generates the reset, clock and control signals, Furthermore it simulates the ICAP. For simulation of the BRAM, the verification model provided by Xilinx is used.

The testbench used for the verification is a directed testbench. Directed testbenches simulate a specific action or actions, where the UUT (Unit Under Test) was designed for. A directed testbench does not test every possible input combination, like a total testbench. A testbench testing every input combination is useful when verifying a arithmetic unit or some other data processor.

In the PRMU case testing every possible input combination is not useful because the PRMU is not altering the data. The data is only transferred by the PRMU, thus the data has no influence on the operation. A bitstream processor however does alter data, in such case a full testbench might be useful.

In order to simplify the verification it is wise to verify the standard PRMU (i.e. the PRMU without bitstream altering devices) independently. Bitstream manipulating devices can at best be verified using a separate testbench.

The situations or test cases tested by the directed testbench are:

Single Bitstream Re-configuration using one bitstream

Two bitstreams, successive Re-configuration using two Bitstreams, stored in a uninterrupted sequence(i.e. bitstream 1 is directlly followed by bitsream 2)

Two bitstreams, gap Re-configuration using two bitstreams, bitstream 1 is not directlly followed by bitstream 2.

The verification process proofed the proper operation of a re-configuration using the on-chip repository. The abort sequence was also covered by the verification process. The testbench used during verification generates the signals used to start a reconfiguration, the ICAP is simulated by a model. This ICAP model stores the data written to it in an array, which can is compared against the original bitstream. Furthermore the waveform generated by ModelSim were analysed by hand in order to proof the correct operation of the design.

When analysing the waveform, a few time slots in particular require our attention. These moments are the start of the re-configuration and the end. We carefully checked whether the correct number of bytes were written to the ICAP. As mentioned earlier a bitstream should start with a dummy word followed by a synchronisation word and must end using a de-synchronisation or no-operation word. The particular situations mentioned were verified manually.

5.2 Testing Using XUP Board

Testing is done using a test design mapped to the XUP Board. The test design is a simple re-configurable design, containing a fixed module and a re-configurable module, set up using [14] and [5].

The fixed part of the design contains a component to generate test vectors to drive the re-configurable module. Furthermore the PRMU is instantiated in the fixed module. The re-configuration logic and the logic for stimuli generation can be controlled using the push bottons of the XUP Board.

The re-configurable design contains two simple re-configurable modules. One module is a pure combinatorial unit. This unit executes logic operation on the four bits stimuli vector. The other component is a four bits counter, controllable using one of the push buttons.

Figure 5.1 shows the re-configurable design used for testing the PRMU. The PRMU, the unit under test in this case, is situated in the lower right of the figure. The BRAM functioning as on-chip repository is placed to the left of the PRMU, while the ICAP is

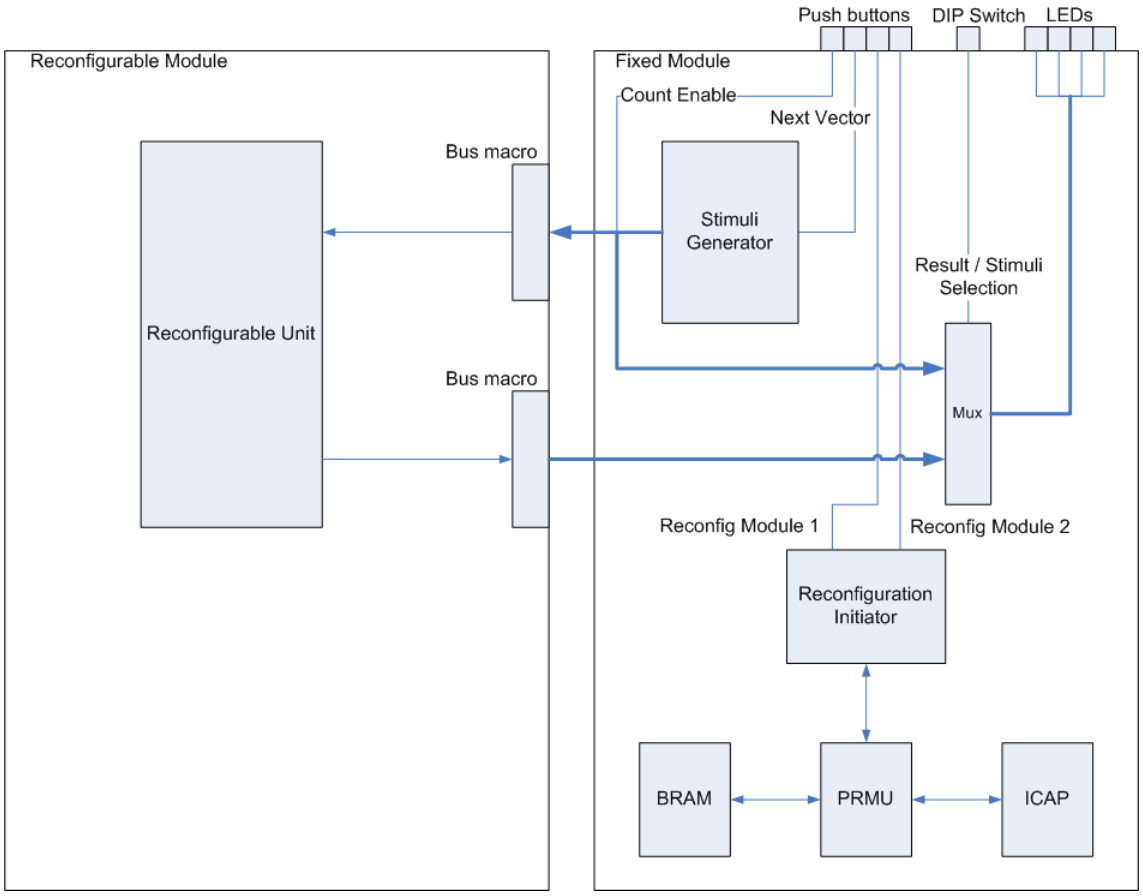

Figure 5.1: Schematic of the re-configurable design

on the right side.

All other components are meant for performing the test. The re-configuration initiator simulates the arbiter of the MOLEN framework. This component initiates a re-configuration in the test framework. The re-configuration using either bitstream 1 or bitstream 2 can be forced externally using the push buttons. The other two push buttons are used for the generation of stimuli. The most left button functions as the count enable when the counter is loaded. The next button is used to force the Stimuli Generator to generate the next stimuli vector. The DIP Switch is used to switch either the stimuli vector or the result to the LEDs, multiplexing is necessary in this case because the XUP Board features only four LEDs.

All the buttons are de-bounced, the components for de-bouncing are not shown in the schematic as well as reset and clock signals.

The logic operations are chosen in such manner that the proper operation can easily be verified. The first column of the results are those of a bit-wise AND operation. The

Table 5.1: Truth Table of PRM module 1

Stimuli				**Result**			
a	**b**	**c**	**d**	$a \wedge b \wedge c \wedge d$	$\neg(\neg a \vee \neg b \vee \neg c \vee \neg d)$	$a \vee b \vee c \vee d$	$\neg(\neg a \wedge \neg b \wedge \neg c \wedge \neg d)$
0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	1
0	1	0	0	0	0	1	1
0	1	0	1	0	0	1	1
0	1	1	0	0	0	1	1
0	1	1	1	0	0	1	1
1	0	0	0	0	0	1	1
1	0	0	1	0	0	1	1
1	0	1	0	0	0	1	1
1	0	1	1	0	0	1	1
1	1	0	0	0	0	1	1
1	1	0	1	0	0	1	1
1	1	1	0	0	0	1	1
1	1	1	1	1	1	1	1

second column is equivalent to first, which can be proven using Morgans law. The same goes for the third and fourth column, the third column is a bit wise OR operation, the forth is its Morgan equivalent. Table 5.1 gives a truth table of the combinatorial module.

The second module is a ordinary four bits counter with a count enable. By pressing one of the push buttons the counter counts up in the usual order. The Result value and the Stimuli are displayed using the four LEDs of the XUP Board.

5.3 Re-configuration Latency

The re-configuration latency is a key performance parameter in our design. Because the design is able to supply the re-configuration data at 50 MB/s, which is the maximum data rate the ICAP can handle, we expect the measured data-rate to approach the theoretical maximum.

Furthermore we want to investigate whether either the area, also referred to as slot, or the actual logic determines the re-configuration latency. Does it make any difference how much area a module occupies? When the re-configurartion latency is not proportional to the module "size", small modules will likely perform very inefficient. The inefficiency is caused by the relative large re-configuration latency compared to the functionality of the module.

5.3.1 Measuring Re-configuration Latencies

By extending the test framework, figure 5.1, with some components we are able to measure the re-configuration latency of a specific module. To measure the re-configuration latency we use a counter running on the same frequency as the ICAP. The counter is started when the start operation signal is asserted and stops when the end operation is asserted. This enables us to calculate the exact time of the re-configuration process. The counter value is read out using the ChipScope logic analyser. Using the retrieved information the reconfiguration throughput or data-rate is easily calculated as described hereafter.

5.3.2 Maximum Data-rate

As described we use a counter to determine the time consumed by a re-configuration operation. Using this parameter we can determine the throughput of the re-configuration data.

In order to calculated the data-rate we need two parameters namely: The size of the bitstream in bytes and the re-configuration time in seconds.

Using the memory instantation file generation tool, the bitstream size is determined. The tool reports the bitstream size to be 19134 bytes. This are actual the number of bytes written to the ICAP. The re-configuration timer count is 19143 clock-cycles. Thus 19134 bytes are written in 19134 clock-cycles. In other words one byte per clock-cycle. Because de design runs on a 50 Mhz clock it is easily verified that the design has a throughput of 50 MB/s. The following equations repeats the proof in a mathematical form.

Definition of throughput:
$Throughput = \frac{\#Data(Bytes)}{\#Time(Seconds)}$

Solution:
$Throughput = \frac{\#Data(Bytes)}{\#Time(Seconds)}$
$Throughput = \frac{19134}{19134*20*10^{-9}}$
$Throughput = \frac{1}{20*10^{-9}}$
$Throughput = 50,000,000B/s = 50MB/s$

The discussion in previous paragraph and the more mathematical formulation given, state that we can achieve the maximum throughput on the target FPGA. Because the solution is scalable and can run at 100 MHz the design can also handle the Virtex 4 ICAP throughput. The scaling is realised by means of generic parameters. A version of the design scaled to the Virtex 4 has been implemented in order to verify whether the timing constraints are still met.

Table 5.2: Bitstream size

Slot size	Bitstream size
1	19 kB
2	23 kB

The Virtex 4 ICAP has a 32 bit databus and it can run on a 100 MHz clock rate. Thus, four bytes at 100 MHz which lead to a data-rate of 400 MB / s.

In conclusion we can state that our solution is capable of performing re-configurations at the maximum data-rate on both Virtex 2 Pro and Virtex 4 devices. The design has been tested on the Virtex 2 Pro, the statements regarding the Virtex 4 are based on extrapolation of the results. Due to the fact that a Virtex 4 device is not available the tests could not be performed.

5.3.3 Relation between the Slot Size and Re-configuration Latency

As mentioned in the introduction of this section we would like to analyse what is of main influence to the bitstream size. Is it either the slot size or the size of the actual module. When the slot size determines the bitstream size, the re-configuration latency is fixed no matter the size of the module. It is obvious that this might lead to very bad performance in case of a small module mapped to a relative large slot. It would be more advantageous when the module size would determine the bitstream size. Because performed tests have proven that the re-configuration latency is linear[1] to the bitstream size.

In order to find out whether the bitstream is either determined by the slot size or the module size we conducted the following test. We implemented a partial re-configurable design twice. Before the second implementation doubled the size of the re-configurable slot. Afterwards we compared the size of the generated partial bitstreams. The result are shown in table 5.2.

Table 5.2 shows that doubling the slot size causes a slight increase in bitstream size. However the bitstream size will not double. The result shows us that a small module mapped to a relative large slot is less efficient as mapping it to a smaller slot. The slot size is a factor that determines the bitstream size, although it is not the main factor.

[1]This has been proven by measuring the reconfiguration latency using multiple bitstreams differing in size

At first it sounds odd that bitstreams used to re-configure the same area might differ in size. One expects them to be equal sized, because they need to configure the same amount of re-configuration memory. The difference in size is caused by a compression applied to the bitstream. This compression is done by including a byte that is repeated sequentially, only ones in the bitstream. Information regarding the length of the repetition needs to be include. It is obvious that this compression technique can shorten the bitstream significantly.

The compression reduces no only the bitstream size, but also the re-configuration time. Although it is not documented in detail, the re-configurtation logic is apparently capable of simultaneously writing equal values to multiple locations.

6 Results

This chapter describes the results of the implementation process. Resource usage and maximum clock frequencies are discussed. Furthermore this chapter compares our design against related work.

The chapter is organised as follows. Section 6.1 discusses the resource usage, while section 6.2 discusses the maximum clock frequency. Lastly a performance comparison is conducted by means of section 6.3

6.1 Resource Usage

In VLSI design a main performance parameter is the resource usage. The resource usage is vitally important when one considers integration in a larger collective. The resource usage is reported in percent of the proposed platform, furthermore the absolute resource usage is given. In this section we discuss the components earlier identified as PRMU (see 3.5). Memory required for bitstream storage is covered by 3.3.

The PRMU, composed of the re-configuration unit, repository selector and repository interface use only a fraction of the FPGA resources. As shown in table 6.1 the PRMU uses only 2 % of the available slices.

6.2 Maximum Clock Frequency

The maximum clock frequency is one of the main performance parameters in a VLSI design. This parameter gives the maximum operation frequency of the design. Therefore the this parameter is closely related to the maximum "speed" of a VLSI design.

As presented in Table 6.1, the PRMU, composed of the reconfiguration unit, repository selector, and repository interface, is capable of achieving an operating frequency of 100 MHz, which is more than sufficient for the used FPGA.

Table 6.1: Slices and Maximum Clock Frequency

Unit	**Slices**	**Max. Freq.**
PRMU	336 (2%)	102 MHz

Implementation results suggest that the proposed system can also achieve the maximum reconfiguration throughput using the Virtex 4 FPGA, which has an ICAP operating at 100MHz with 32-bit input data.

6.3 Related Work

There are numerous so called self re-configuring systems using the micro-controller based solution. The disadvantages of this approach are discussed in 3.4. In this section we compare some other self re-configuring systems against our design.

Two publications regarding self re-configuring systems are discussed and compared against our design. The related work is introduced in brief hereafter.

Möller et al [7], Darmstadt University of Technology Two solutions for enabling partial re-configuration are discussed in this paper. The fist is based on the OPB_HWICAP, while the second is a mainly hardware based design like ours.

Claus et al [2], Munich University of Technology This paper compares the performance of the OPB_HWICAP against a PLB based HWICAP. This PLB_HWICAP is developed at the Munich University of Technology.

In order to analyze the functionality and the hardware costs of the proposed units, a Xilinx Viretex 2 Pro (XC2VP30-6) on a XUP prototyping board has been used. Given that on a Virtex 2 Pro the ICAP received 8-bit data at 50 MHz, a maximum reconfiguration throughput of 50 MB/s is available in this FPGA.

Several self reconfiguring systems using a micro-controller have been proposed. In such approach a micro-controller is used to write the reconfiguration data to the ICAP.

Table 6.2 summarizes the reconfiguration throughput (Reconf. ThrPut.) and hardware requirements of the solutions we compare against ours.

Table 6.2: Performance Comparison

Solution	**Reconf. ThrPut.**	**Slices**
Claus(OPB) [2]	5 MB/s	933
Claus(PLB) [2]	50 MB/s	919
Möller(OPB) [7]	2 MB/s	973
Möller(NoC) [7]	10 MB/s	1661
Ours	50 MB/s	336

The solutions proposed in the mentioned publications have a similar structure, which is composed of a reconfiguration unit and a memory interface. The design proposed

in this thesis uses a direct memory interface, referred to as repository interface, while the other use a Processor Local Bus (PLB), an OPB, or a NOC (Network On Chip) as memory interface.

A PLB and an OPB based approach are presented in [2]. Both approaches require, in contrast to our design, an additional micro-controller to control the data flow. Both approaches require more that 900 slices, 170% more than our design. The OPB approach is significantly slower than ours, while the PLB approach can achieve a reconfiguration throughput of 50 MB/s , imposed by the ICAP interface. It should be noted that the throughput of the PLB bus can significantly vary, depending on the utilization of the PLB.

The number of slices of [2], depicted in table 6.2, are composed of the values presented in [2] and the slices required for the additional bus and micro-controller. Claus et al. [2] only present the slices required for the reconfiguration unit. The number of slices required for the bus and micro-controller were retrieved from [7].

The NOC based approach proposed by Möller et al. [7] uses a NOC to fetch the bitstream from memory. Identically to our approach, the design using the NOC also does not require a micro-controller; however, the use of the NOC requires 1661 slices and is only able to achieve a reconfiguration throughput of 10 MB/s.

Conclusions and Recommendations

7

In this chapter the final conclusions are drawn and recommendations are stated. The recommendations and conclusion apply to the PRMU core, and the design flow for implementation of MOLEN framework applications.

The PRMU core is discussed in section 7.1. This section gives some specific recommendations regarding the off-chip repository. Section 7.2 gives recommendations for a design flow tailored to the MOLEN project. Future work is covered by section 7.3.

7.1 PRMU Conclusions

Although the current implementation does not support an off-chip repository and re-location, it is a strong basses for further development of RPR in the MOLEN prototype. Furthermore integration of the design in the MOLEN framework is an open item.

The PRMU complies to the interface of the MOLEN framework. Furthermore the resource usage is limited, so integration is relatively . The only limiting factor is the lag of an off-chip repository. However the PRMU is designed to be extended with multiple repositories.

The implementation of a PLB master, capable of fetching a bitstream autonomously (i.e. without micro-processor intervention) is a labour extensive task. By sides it doubtful whether it will bring much benefit, because the OPB/PLB designs we compared against performed poor. The lag of performance is most likely caused by the shared nature of the OPB and PLB bus.

Based on the results presented by Claus et al [2], an autonomous PLB master might be beneficial for a Virtex 2 Pro implementation. However we can not achieve the minimum re-configuration time when we migrate to the Virtex 4 series. In fact when we consider the results of Claus et al [2], we can only achieve a quarter of the maximum data-rate. Furthermore the throughput of the shared bus decreases by the number of bus devices.

In order to minimise the influence of the re-configuration latency on the overall performance, so called re-configuration latency hiding is proposed. Hiding of the re-configuration latency means planning the re-configuration of a CCU ahead of its usage. For the MOLEN programmer it seems that the re-configuration process is time-less. Re-configuration latency hiding can be realised by a set of measures. One

could consider compile time measures (e.g. planning the re-configuration well ahead of the hardware task execution) and run-time measures (e.g. enabling simultaneous GPP program execution and re-configuration).

Fetching the bitstream using the PLB will influence the GPP program execution if it also accesses the PLB. Thus the re-configuration will harm the performance of the software execution. For this reason it is better to use a separate and directly accessible memory as off-chip repository. The main advantages of this approach is that software execution and re-configuration can be performed purely in parallel. A separate and non-shared re-pository will increase the performance and portability, furthermore developing an interface is much less labour intensive.

The XUP board is not equipped with a storage device which we can serve as repository. The board does supply a compact flash expansion slot, however this type of memory can not provide the data-rate desired in our application. Accessing the memory directly is another option. This implies that the memory can not be used by other devices, because only a fraction of the memory is needed for bitstream storage it is clearly a waste of memory.

7.2 Recommendations Regarding Design Flow

The Xilinx tools do not support implementation of partial re-configurable designs. There is however a extension patch to ISE 8.2, enabling implementation of partial re-configurable designs. The patch provides some perl scripts to eas the implementation flow. Although the patch simplifies the the implementation it requires most steps to be done manually.

In order to enable rapid development of applications for the MOLEN framework, a design flow needs to be developed. Furthermore mechanisms must be developed to enable co-simulation. Using co-simulation the RP, MOLEN Framework and the developed software can be simulated and debugged simultaneous.

7.3 Future Work

There are four main open items for future work namely: Integration in the MOLEN Polymorphic Processor Framework, creating a design flow tailored to the MOLEN project, implementation of a fast off-chip repository, implementation of re-location and development of a generic repository interface. Each of the four items is outline hereafter.

Integration in the MOLEN Polymorphic Processor Framework: The design must be integrated in the current MOLEN framework prototype. The integration requires the floorplan of the prototype to be adapted. Furthermore the communication structure outlined in appendix B, meant for communication between the framework and the CCU, must be routed via bus macro's. The bus macro's must be placed at the slot bounds as discussed in chapter 2.

Implementation of design flow: As mentioned before the design flow should not only automate the implementation of a re-configurable design, but also feature co-simulation.

Implementation of an off-chip repository: Research needs to be performed to find a solution for the off-chip repository. The autonomous PLB proposed in this thesis is a possible solution, although it should be verified which data-rate it can offer. A independent non shared repository would enable true latency hiding and prevent interference between the GGP execution and CCU reconfiguration.

Implementation of re-location: Re-location or dynamic mapping can be integrated in the current design by development of a bitstream processor as proposed in this thesis, see 4.2 .

Implementation of a generic repository interface: Developing a generic or a few generic repository interfaces simplify the integration of repositories. The user only needs to adapt some parameters in order to build a repository for a specific storage device.

Bibliography

[1] *Virtex-ii pro and virtex-ii pro x platform fpgas: Complete data sheet (ds083 v4.6)*, Tech. report, March 2007.

[2] Christopher Claus, Florian H. Müller, Johannes Zeppenfeld, and Walter Stechele, *A new framework to accelerate virtex-ii pro dynamic partial self-reconfiguration*, (2007).

[3] Heiko Kalte and Mario Porrmann, *Replica2pro: Task relocation by bitstream manipulation in virtex-ii/pro fpgas*, Tech. report, University of Western Australia School of Computer Science & Software Engineering, Heinz Nixdorf Institute System and Circuit Technology University of Paderborn, 2006.

[4] Yana E. Krasteva, Ana B. Jimeno, Eduardo de la Torre, and Teresa Riesgo, *Straight method for reallocation of complex cores by dynamic reconfiguration in virtex ii fpgas.*

[5] Gregory Mermoud, *A module-based dynamic partial reconfiguration tutorial*, November 2004.

[6] Abhishek Mitra, Zhi Guo, Anirban Banerjee, and Walid Najjar, *Dynamic co-processor architecture for software acceleration on csocs*, (2006).

[7] Leandro Möller, Ismael Grehs, Ewerson Carvalho, Rafael Soares, Ney Calazans, and Fernando Moraes, *Anoc-based infrastructure to enable dynamic self reconfigurable systems*, (2007).

[8] David A. Patterson and John L. Hennessy, *Computer organisation and design*, third edition ed., Morgan Kaufmann, 2005.

[9] P. Sedcole, B. Blodget, T. Becker, J. Anderson, and P. Lysaght, *Field programmable logic and applications modular dynamic reconfiguration in virtex fpgas*, IEEE Proc.-Comput. Digit. Tech., Vol. 153, No. 3, May 2006 (2006), 157– 164.

[10] Heng Tan and R. F. DeMara, *A multi-layer framework supporting autonomous runtime partial reconfiguration*, (2007).

[11] S. Vassiliadis, S. Wong, and S. D. Cotofana, *Microcode processing: Positioning and directions*, IEEE Micro, vol. 23, no. 4 (2003), 21–30.

[12] M. V. Wilke, *The best way to design a automatic calculating machine*, Ferranti LTD, 1952.

[13] Xilinx, *Two flows for partial reconfiguration: Module based or difference based*, September 2004.

[14] Xilinx, inc., *Early access partial reconfiguration user guide for ise 8.1.01i*, March 2006.

Plan of Approach

A

This document describes the approach to the Bachelor of engineering thesis of Kees van der Bok. The thesis subject is the implementation of partial reconfiguration in the MOLEN project.

More information regarding the MOLEN project can be found in section A.1. In section A.2 the assignment is given. Section A.3 discusses the chosen approach. In succession the progress meetings are discussed, see section A.4. Parties responsible for support and supervision as well as the persons representing those parties are mentioned in section A.5. The deliverables are covered by section A.6 while section A.7 describes the layout of the thesis report. Planning including time table are covered by section A.8. The last section describes the presence, which can be useful for supervisors, advisors etc.

A.1 The MOLEN Project

The MOLEN polymorphic processor also referred to as the MOLEN framework consists out of a general purpose processor linked to a co-processor. The co-processor can be re-configured to execute a specific task. The GPP and co-processor can operate in parallel. Both processors communicate using dedicated exchange registers for small amounts of data. For large chunks of data memory sharing is used.

Implementation of time consuming operations in dedicated hardware is an often used technique, to accelerate a initially pure software implementation. Moving functionality from software to hardware is generally referred to as co-design. The MOLEN project intends to extend this co-design methodology by making runtime re-configuration of the hardware implementation possible. Basically the MOLEN can be seen as a processor extended with a reconfigurable co-processor.

A.2 Assignment

The MOLEN research project at the Delft University of Technology is developing a design system for a re-configurable processor. The reconfigurable processor consists of a (general) core processor as well as some special - application dependent - hardware acceleration units.

The MOLEN system takes as input

1. An application program in which functions are identified that require hardware acceleration
2. HDL descriptions for the hardware units that implement these functions Based upon this input the MOLEN system generates
   (a) A configuration file for a reconfigurable design platform that implements the reconfigurable processor
   (b) An executable file that can run on this processor. Currently a prototype for the MOLEN system is available for reconfigurable design platforms based on the Xilinx Virtex-II Pro technology. For more details on the project see http://ce.et.tudelft.nl/MOLEN

To fully use the potential of the MOLEN system, the idea is that the hardware acceleration units are configured (and deleted to make space for new ones) during run-time. This should be possible to implement using the partial reconfiguration techniques that are available within the Xilinx Virtex-II Pro technology and using the ICAP controller that allows the self-configuration in Xilinx devices. The task of this project is.

1. To find how partial re-configuration works and how ICAP can be used for this.
2. To implement this in the MOLEN system.

Example application programs and HDL descriptions are available for the the MOLEN system that can be used to verify the new implementation.

A.3 Approach

The tasks described in the assignment are the two main tasks of the project. Based on the assignment the following phases are derived.

1. Study of papers, articles etc. to achieve some general understanding of the operation of the MOLEN.
2. Writing a plan of approach.
3. Literature study into partial re-configuration options of the Xilinx Vertex-II pro FPGA.
4. Derive specifications, requirements for the design.
5. Propose a possible implementation of partial. re-configuration in the MOLEN prototype by means of a functional design.

6. Implement and realize the functional design.
7. Verify and test the design.
8. Solve possibly discovered errors.
9. Finishing of documentation etc.

Almost all the mentioned phases will result in one or more chapters to be added to the Thesis report. The intention of this approach is to achieve a better final result. Furthermore the finished chapters can be useful to inform the project supervisors and advisors about the progress being made.

A.4 Progress Meetings

To keep track of progress, progress meetings are planned on a weekly bases. During these meetings progress and other matters are discussed. The progress meetings will be attended by the supervisor and student. If necessary other people might be invited to this meetings.

Based on the timetables of both supervisor and student a certain day and time will be reserved for this meeting.

A.5 Thesis Support and Supervision

This Bachelor of engineering thesis is supervised and supported by two parties. The school responsible for offering education and relating issues. As well as the company offering the thesis work.

The parties and persons representing them:

School: ***Rijswijk, University of Professional Education*** Represented by: *ing. Th. Koreneef*

Company: ***Delft, University of Technology, Research group: Computer Engineering*** Represented by: *Dr. ir. A.J. van Genderen*

A.6 Deliverables

The deliverables of the thesis work are:

Thesis report Report covering all the required documentation.

Implementation and realization Source code, project files of used tools etc.

A.7 Report Layout

The report layout is based on the earlier specified phases and due to this closely related to the planning, described in the next section.

The report writing will be a continues process during the thesis period resulting in a final report at the end. Each phase of the thesis will result in one or more chapters to be added to the report. The finished chapters will function as intermediate reports which can be used by supervisors and advisors to stay informed about the progress made.

Report Layout

Preface

Summary

Main part This part is produced during the mentioned phases.

Introduction Brief description of the MOLEN project and information about the assignment.

Study into Partial Re-configuration Description of the operation and possibilities of the partial reconfiguration feature of the vertex II pro FPGA.

Specifications Description of requirements the design has to apply to.

Functional Design Description of the design, from general concept down to separate modules(blocks) to be implemented.

Verification and Testing Description of verification and test performed. Furthermore a discussions covering the test results.

Results Discussion of the results. Performance, maximum clock frequentcy and resource usage are covered.

Conclusions and Recommendations The main conclusions and recommendations.

Appendix

A.8 Planning

Because of following a bridging program parallel to this thesis work, the regular time span of twenty weeks is spread over a longer period. The thesis has started in January and will end in late August.

The time spend on the thesis work, is about twenty hours per week during the spring semester. During the summer break a regular work week is planned.

Table A.1 shows a week based planning. Tasks to accomplish for the specific week are given as well as the estimated time to invest during that week.

Italic week numbers indicate milestones during this weeks intermediate reports or chapters are planned to be finished. The following milestones are defined.

Week 3 Jan-19-2007 *hand in plan of approach for review.*

Week 6 Feb-9-2007 *Finish and make available conclusions and recommendations based on study of partial re-configuration.*

Week 12 Mar-23-2007 *Finish and make available documentation etc. concerning functional design and requirements phase 1.*

Week 29 Jul-20-2007 *Finished design and test frame work*

Week 31 Aug-3-2007 *Handing in concept report for review*

Week 37 Sep-20-9-2007 *Handing in final report for review*

A.9 Presence

The given planning is based on the amount of hours invested. Because of courses to attend in parallel to this thesis it isn't possible to work full working weeks. The presence is described in the following paragraphs.

During the first quarter *5-2-2007, 23-3-2007*

Monday Until 13.00 *4 hours*

Wednesday All day *8 hours*

Friday Starting at 11.00 *5 hours*

Total hours per week: *17*

During the second quarter *9-4-2007, 8-6-2007*

Monday All day *8 hours*

Wednesday or Tuesday Afternoon *4 hours*

Friday All day *8 hours*

Total hours per week: *20*

Week number	Tasks	Spendable hours
First semester exam period 3 weeks		
3	Plan of approach	32
4	Literature study "Partial re-config"	28
5	Literature study "Partial re-config"	32
First quarter 7 weeks		
6	Writing summary of "Partial re-config." study	17
7	Study of available tools, by reading handbooks etc.	17
8	Requirements, interface specifications	17
9	Requirements, interface specifications	17
10	Functional design	17
11	Functional design	17
12	Finishing Functional design incl. doc.	17
Exams first quarter 2 weeks		
13	White week	0
14	Exams	0
Second quarter 7 weeks		
15	Change of functional design	9
16	Change of functional design	20
17	Change of functional design	20
18	Implementation and realisation	32
19	Implementation and realisation	20
20	Implementation and realisation	12
21	Implementation and realisation	20
22	Implementation and realisation	12
23	Implementation and realisation	20
Exams second quarter 4 weeks		
24	Exams	0
25	Exams	0
26	Exams	0
27	Exams	0
Summer break		
28	Implementation and realisation	44
29	Verification and testing	44
30	Verification and testing	44
31	Finishing concept report	44
32	Verification and testing	44
33	Rehabilitation due to car crash	0
34	Rehabilitation due to car crash	0
Period extension		
35	Verification and testing	20
36	Finishing thesis report	20
37	Finishing thesis report	20

Table A.1: Time table of thesis period

Communication structure specification

B

clock and reset bus: All clock and reset signals

Reset: Global asynchronous reset signal

CCU_clk: Clock signal for the CCU

mem_clk: Clock signal for the memory interface

PPC_clk: Clock signal of the PowerPC

micro-code bus: The MIR *microcode instruction register*

CCU_start: Signal to start the CCU

CCU_end: Signal to end the CCU

MIR(63:0): Microcode instruction register

status(63:0): Status register, feedback from CCU to the MOLEN logic

Exchange register file bus: The exchange register file is a set of read/write registers. The exchange register file can be accessed through a memory alike interface.

addr_XREG(31:0): The address of the exchange register

write_XREG(31:0): Data from CCU to exchange register

read_XREG(31:0): Data read from the exchange register

we_XREG: Write enable signal, actually read/write signal

Data memory bus Interface to the data memory

addr_data(31:0): Address

write_data(63:0): Data from CCU to memory

read_data(63:0): Data from memory to CCU

brw_data(3:0): Memory control signals