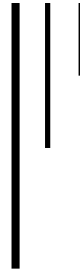

University of Piraeus – Department of Informatics

Master of Science

“Advanced Systems”

Master Thesis

Title	**Optimization of ReRAM write latency and comparison with SRAM technology**
Full Name	**Eirini-Panagiota Douka**
Father’s Name	**Efstratios**
Identification Number	**MPSP 10052**
Supervisor	**Mihalis Psarakis, Assistant Professor**

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Master Thesis

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Three-Member Committee

Mihalis Psarakis	Dimitris Apostolou	Panayiotis Kotzanikolaou
Assistant Professor	Assistant Professor	Lecturer

Contents

1. Introduction **8**

2. Analysis of SRAM and ReRAM modeling **3**

2.1 SRAM 3

2.2 ReRAM 6

2.2.1 Generally 7

2.2.2 ReRAM cell characteristics and system requirements: 7

3. PROCESSOR ARCHITECTURE **9**

3.1 ADRES 9

3.1.1 Generally 9

3.1.2 ADRES Architecture 9

3.2 DRESC COMPILER 11

3.3 BoADRES 11

4. RERAM Implementation For Data Memory Replacement . **115**

4.1 Experimental Setup 12

4.2 Simulation Results 22

4.3 Energy Modeling and Results 28

4.3 Discussion 29

5. Conclusions and Future Work **30**

References **32**

Figures

Figure 1 SRAM cell 4
Figure 2 4

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Figure 3 4T SRAM cell 5
Figure 4 6T SRAM cell 5
Figure 5 TFT SRAM cell 6
Figure 6 Basic ReRAM cell 8
Figure 7 Decimation in Time 12
Figure 8 Decimation in Frequency 13
Figure 9 15

Source Code

Source Code 1 SRAM entry process 14
Source Code 2 SRAM_banked module definition 14
Source Code 3 Instances of SRAM_banked module 15
Source Code 4 SRAM_banked process for bank selection 17
Source Code 5 p_sel_delayed process 18
Source Code 6 ReRAM writing process 19
Source Code 7 ReRAM reading process 19
Source Code 8 ReRAM stall 20
Source Code 9 20
Source Code 10 21
Source Code 11 22

Screenshots

Screenshot 1 SRAM 23
Screenshot 2 SRAM Banked 23
Screenshot 3 ReRAM 24
Screenshot 4 ReRAM Banked (version 1) 24
Screenshot 5 ReRAM Banked (version 2) 25
Screenshot 6 SRAM 25
Screenshot 7 SRAM Banked 26
Screenshot 8 ReRAM 26
Screenshot 9 ReRAM Banked (version 1) 27
Screenshot 10 ReRAM Banked (version 2) 27

Tables

Tabel 1 Energy Numbers for SRAM and ReRAM 28
Tabel 2 Computational Energy 28
Tabel 3 2-Stages FFT 29
Tabel 4 Full-Stages FFT 29

Charts

Chart 1 31
Chart 2 31

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Abbreviations

ADRES	Architecture for Dynamically Reconfigurable Embedded Systems
ASIC	Application-Specific Integrated Circuit
ASIP	Application Specific Instruction Set Processor
CGA	Coarse-Grain-Array
CGRA	Coarse-Grained Reconfigurable Architecture
CMOS	Complementary Metal-Oxide Semiconductor
DFT	Discrete Fourier Transform
DRAM	Dynamic Random-Access Memory
DSP	Digital Signal Processors
FFT	Fast Fourier Transformation
FPGA	Field-Programmable Gate Array
FU	Functional Units
NMOS	Negative Metal Oxide Semiconductor
NVM	Non-Volatile Memory
RAM	Random-Access Memory
ReRAM	Resistive Random-Access Memory
RF	Register Files
SoC	System-on-Chip
SRAM	Static Random-Access Memory
VCC	Voltage
VLIW	Very Long Instruction Word

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ACKNOWLEDGMENTS

This thesis represents work completed over a two year period. I am thankful for all the support and efforts from professors, staff members, fellow graduate students, and industrial partners.

First of all, I would like to express my sincere appreciation and thankfulness to my professor Mr. Mihalis Psarakis for his support to this great step I wanted to do abroad. Moreover I would like to thank him for his help and guidance wherever this was needed.

This work has been conducted under a project with the Wireless Communication Group at IMEC Research Center. Sincere appreciation goes to my promoter on this project, Dr.Francky Catthoor for his trust on my knowledge and abilities. Also I would like to thank my supervisor Matthias Hartmann, for his patience to show to me and teach me all the needed, for being able to succeed on this project.

I would like to thank current fellow students at the Master Advanced Systems and also some past fellow students at my Bachelor Informatics at the University of Piraeus, for their actual and very important support and help through all of these years.

Finally, I would like to thank my family, especially my parents and sister, and also my friends for their never ending support and encouragement. This work would never have been accomplished without their true guidance, patience, and love.

Abstract – Recently many different kinds of non-volatile memories have been proposed in order to replace the already existed SRAM-based memories. Despite the multiple advantages of those memories like low energy consumption, improvement of the area needed, nevertheless there also some drawbacks like longer write latencies and lower endurance. In this Master Thesis, different methods of architectural design of memory for embedded systems are proposed. The performance of two different kinds of memory, the already widely known SRAM and the emerging ReRAM is evaluated. We created two different implementations for each one of them, as data memory of an FFT application of an ASIP (Application Specific Instruction Processor). The first one is the simple implementation of each one of them and the second is the sub-banked scheme. Also architectural solutions are proposed in order to conceal in the most effective way, the long write latencies of ReRAM and the affection that they have in the performance of the whole system.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Περίληψη – Πρόσφατα πολλά διαφορετικά είδη μη-πτητικών μνημών έχουν προταθεί με σκοπό να ατνικαταστήσουν τις υπάρχουσες SRAM-based μνήμες. Παρόλα τα πολλαπλά πλεονεκτήματα αυτών των μνημών, όπως η χαμηλή κατανάλωση ενέργειας, η μείωση της απαιτούμενου χώρου, ωστόσο υπάρχουν και μειονεκτήματα όσον αφορά στους μεγάλους χρόνους εγγραφής όπως και στη χαμηλή αντοχή. Σ΄αυτή τη Διπλωματική Εργασία, προτείνονται διαφορετικές μέθοδοι σχεδιασμού της μνήμης ενός ενσωματωμένου συστήματος. Η απόδοση δύο διαφορετικών ειδών μνήμης, της ήδη ευρέως γνωστής SRAM και της αναδυόμενης ReRAM εξετάζονται. Δημιουργήθηκαν δύο διαφορετικές υλοποιήσεις για κάθε μία από αυτές, ως μνήμη δεδομένων μιας FFT εφαρμογής ενός ASIP (Application Specific Instruction Processor). Η πρώτη αφορά στην απλή υλοποίηση κάθε μνήμης και η δεύτερη είναι η sub-banked υλοποίηση της. Επίσης προτείνονται αρχιτεκτονικές λύσεις, με σκοπό να απαλειφθούν όσο το δυνατόν και με τον πιο αποτελεσματικό τρόπο, οι μεγάλοι χρόνοι εγγραφής της ReRAM και η επίπτωση τους στην συνολική απόδοση του συστήματος.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

1. Introduction

Memory is any physical device capable of storing information temporarily or permanently. For example, Random Access Memory (RAM), is a type of volatile memory that stores information on an integrated circuit, and is used by the operating system, software, hardware, or the user. Memory can be either volatile or non-volatile memory.

Embedded non-volatile memories (NVM) are generally integrated on a chip, to store programs and data for embedded applications. The term non-volatile is referred to the fact that the content of these kinds of memory is not needed to be periodically refreshed. In this type of memory all read-only memories such as programmable read-only memory (PROM), erasable programmable read-only memory (EPROM), electrically erasable programmable read-only memory (EEPROM) and flash memory are included. It also includes random access memory (RAM) which is powered with a battery. The NVMs require high-voltage devices, and cannot achieve high speed operations due to long write latencies. The high-performance embedded applications require fast-access embedded NVMS.

SRAM-based memories are characterized from lower write latencies when compared to emerging NVMs technologies such as Phase-change (PCRAM), MRAM (Magnetoresistive RAM), STT-MRAM (Spin Transfer Torque MRAM), CBRAM (Conductive Bridging RAM), FeRAM (Ferroelectric RAM) and the most recent RERAM (Resistive RAM), as well as, lower power consumption and higher endurance.

Despite all the previous advantages they suffer from some drawbacks too, such as leakage power in the memory cells, which is a growing concern in advanced CMOS nodes. Also they are susceptible to read/write failure with Dynamic Voltage Scaling (DVS) schemes or a low supply voltage. Moreover SRAM-based memories show a large scaling between different technology nodes, as well as, low area density. As an outcome of the rising number of problems that arise from the traditional SRAM-based memories, significant effort and resources have

been spent on the research and development of these emerging memories technologies that were referred above.

In Imec significant effort on research of STT-MRAM and ReRAM has been spent until now. Especially ReRAM, which is the main type that is examined within this Master Thesis, utilizes materials that change their resistance in response to the applied voltage. This in turn facilitates its data retention capability which is considered to be one of their key advantages. In addition, they also have the ability to read/ write data at high speeds by utilizing limited voltage. These features have largely motivated the industry participants to further study the technology and develop products for the mass market.

This Master Thesis focused only on ReRAM, which gives some really promising results and shows some significant advantages when compared to SRAM based memories, Hence it is attempted, some ways to be found for eliminating the long write latencies of the ReRAM and thus eliminating the energy that is consumed.

This Thesis focuses on the reorganization of the structure of the memory, so as a way to be found for eliminating the stalls arose from the addressing in memory. For example the pure version of ReRAM, follows the memory access pattern. Thus a big number of stalls is caused. The reorganization of the memory structure eliminates this number of stalls by separating the memory in two different banks, which can handle in a more efficient way the memory addressing. Thus the stalls caused by writing in the memory are less.

So 5 different architectural solutions were created in order to compare the performance gain, due to replacement of the SRAM memory with ReRAM memory. The two solutions are the simple SRAM implementation and the sub-banked scheme of it. The third is the Drop-in replacement of the SRAM module with ReRAM module and the 2 different implementations of the ReRAM sub-banked schemes. The first one of the latter does not follow the memory access pattern whereas the second take the memory access pattern into consideration. For each one of them simulations were run on an ASIP environment and the number of cycles were counted. Briefly it was observed that the last implementation (ReRAM sub-banked scheme) which follows the memory access pattern reduces significantly the total number of cycles and gives some promising results that the advantages of the ReRAM can be exploited in an effective way.

Structure of the Thesis

Now a general plan and organization of the Thesis is given. In the *"Analysis of SRAM and ReRAM modeling*" section, there is the definition of what is SRAM and ReRAM and the comparison of them. The second one is an emerging technology that is not widely known so an explanation of how it works actually, starting from the structure of the cell and continuing with the system requirements for embedding ReRAM memories, is quoted.

Afterwards the *"Processor Architecture Part"* regards to some general description for the processor which was used for our experiments and is called ADRES. Then the definition of every individual part of it follows.

The next Chapter *"ReRAM implementation for Data Memory Replacement*" is the main part of the Thesis. Initially the experiment is set up. Also the particular application and the ASIP used for our results is described. Afterwards a brief description of the different implementations that are designed as well as the comparison of them is referred. In *"Further Analysis"* some parts of the source code are quoted, so as to be easier for the reader to see the differences in the implementations of the modules in every different design. This part is followed by the *"Simulation Results"* part. There, there are some screenshots with the results of the simulation of every different implementation of the memory. Using these results and taking some indicative

energy numbers for the computational energy the total energy consumption and the energy leakage in every case is counted. Then the "Discussion" follows, where all the results are analyzed.

Finally in the last part there is the *"Conclusions and Future Work"* where all the results are summarized and some proposals about this topic are given.

Language used for describing the different types of memories

The language that would be more helpful for implementing our memories is SystemC as it the most suitable language for the architecture of complex systems that are a hybrid between hardware and software. SystemC is an ANSI standard C++ class library for system and hardware design.

SystemC is a set of C++ classes and macros which provide an event-driven simulation interface in C++. These facilities enable a designer to simulate concurrent processes, each described using plain C++ syntax. SystemC processes can communicate in a simulated real-time environment, using signals of all the datatypes offered by C++, some additional ones offered by the SystemC library, as well as user defined. In certain respects, it also deliberately mimics the hardware description languages VHDL and Verilog, but is more aptly described as a system-level modeling language.

SystemC is applied to system-level modeling, architectural exploration, performance modeling, software development, functional verification, and high-level synthesis. SystemC is often associated with electronic system-level (ESL) design, and with transaction-level modeling (TLM).

SystemC language has also some strong features that set it as the most possible candidate for our application like the user definition of modules, ports, exports, processes, interfaces, events, data types etc.

The language used for the bullding of the firmware which is described below is C++.

2. Analysis of SRAM and ReRAM modeling

2.1 SRAM

2.2 General Description

Static random-access memory (SRAM) is a type of semiconductor memory that uses bistable latching circuitry to store each bit. The term static differentiates it from dynamic RAM (DRAM) which must be periodically refreshed. SRAM exhibits but it is still volatile in the conventional sense that data is eventually lost when the memory is not powered.

SRAM is a random access memory which can store the data as long as power is on. The SRAM cell consists of a bi-stable flip-flop connected to the internal circuitry by two access transistors. When the cell is not addressed, the two access transistors are closed and the data is kept to a stable state, latched within the flip-flop.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

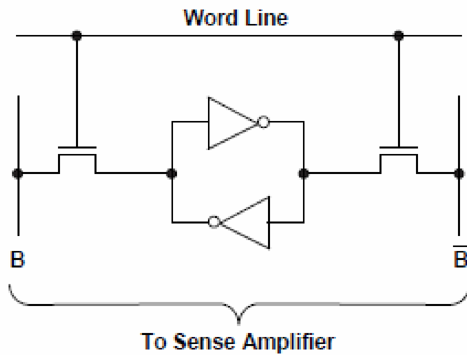

Figure 1 SRAM cell

The flip-flop needs the power supply to keep the information. The data in an SRAM cell is volatile (i.e., the data is lost when the power is removed). However, the data does not "leak away" like in a DRAM, so the SRAM does not require a refresh cycle. Below the read and the write operations are presented:

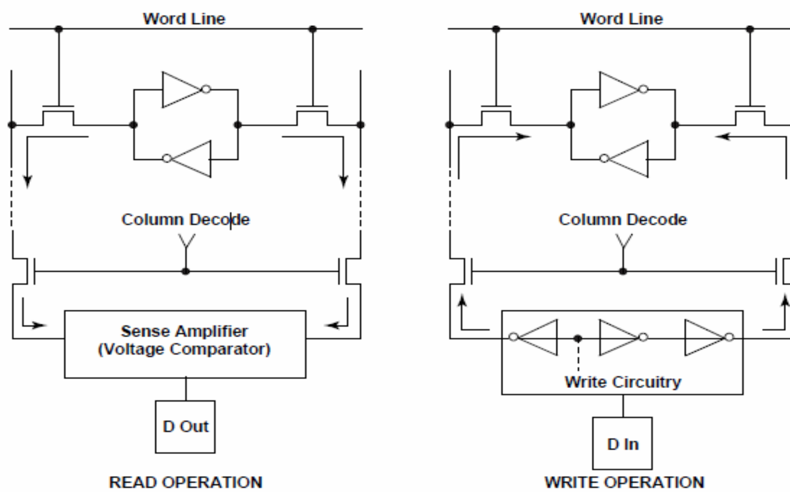

Figure 2

To select a cell, the two access transistors must be "on" so the elementary cell (the flip-flop) can be connected to the internal SRAM circuitry. These two access transistors of a cell are connected to the word line (also called row or X address). The selected row will be set at VCC. The two flip-flop sides are thus connected to a pair of lines, B and B. The bit lines are also called columns or Y addresses.

During a read operation these two bit lines are connected to the sense amplifier that recognizes if a logic data "1" or "0" is stored in the selected elementary cell. This sense amplifier then transfers the logic state to the output buffer which is connected to the output pad. There are as many sense amplifiers as there are output pads.

During a write operation, data comes from the input pad. It then moves to the write circuitry. Since the write circuitry drivers are stronger than the cell flip-flop transistors, the data will be forced onto the cell.

When the read/write operation is completed, the word line (row) is set to 0V, the cell (flip-flop) either keeps its original data for a read cycle or stores the new data which was loaded during the write cycle [1], [4], [5].

2.2 Types of SRAM

Different types of SRAM cells are based on the type of load used in the elementary inverter of the flip-flop cell. There are currently three types of SRAM memory cells:

• The 4T cell (four NMOS transistors plus two poly load resistors)

This is the most common cell. It consists of four transistors, to reduce the required cell area. The challenge for the 4T transistors is to make the resistor high enough, in order the current needed to be eliminated, but it must be up to one level, otherwise the good functionality is not guaranteed.

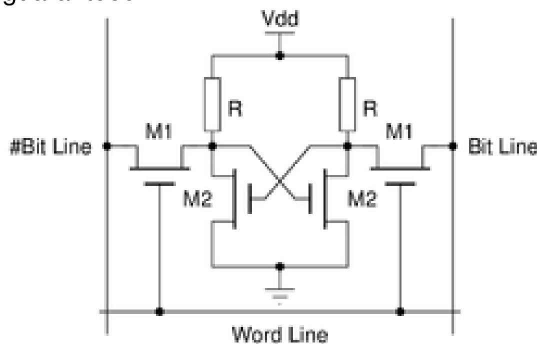

Figure 3 4T SRAM cell

• The 6T cell (six transistors—four NMOS transistors plus two PMOS transistors)

Generally 6T SRAM cell is used in many of commercial chips due to low leakage current and short operation latency which is typically < 1 ns

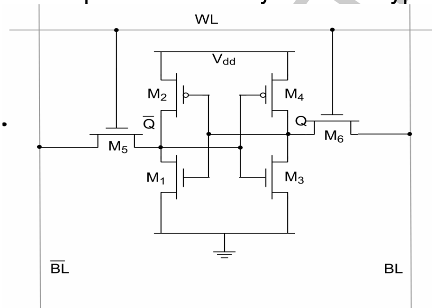

Figure 4 6T SRAM cell

n-channel MOSFET (NMOS) transistor is used as an access transistor due to higher mobility and lower on resistance. Memory cells that use less than six MOS transistors are

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

available as well. As the cost of processing a silicon wafer is partially fixed, using scaled cells and thus packing more bits on one wafer will end up with reduction of the cost per bit of memory. But due to aggressive CMOS scaling there are potentially several issues which can limit the progress of scaling.

One of the key factors which limit the SRAM scaling is less noise margin of 6T SRAM bit-cells in sub-threshold voltage. In many memory designs, to have less dynamic power consumption, lower supply voltage has been used. Also for static leakage reduction, source-body biasing has been utilized. The analysis shows that in these conditions the noise margin of the SRAM cell is reduced as well. Hence, the range of applicability of scaling is limited by the noise margin requirements for a safe read and write operations.

• The TFT cell (four NMOS transistors plus two loads called TFTs) view of six MOS SRAM cell shown in Figure. Each bit is stored on four transistors and two supplemental access transistors are used to control the access to a memory cell during read and write operations. Due to its low fabrication cost and unique feature of flexible substrate, various emerging devices use it. But most TFT transistors used today are weak enough devices that although reduce the current needed do not improve the cell stability.

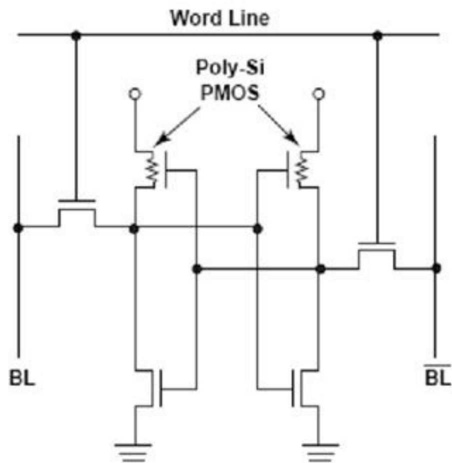

Figure 5 TFT SRAM cell

SRAM Timing Diagram

Synchronous or Asynchronous — SRAMs come in a variety of architectures and speeds, and in synchronous and asynchronous designs. Asynchronous SRAMs respond to changes at the device's address pins by generating a clock signal that is used to time the SRAM’s internal circuitry during a read or write operation. Although commonly used, this type of design runs into limitations at the high end of the performance range. For this reason, the fastest SRAMs are generally synchronous. Synchronous SRAMs use one or more external clocks to time the SRAM's operations. Because of the improved timing control possible with this method, access times and cycle times can be reduced to match the clock cycles of the fastest PC and RISC processors on the market today

2.2 ReRAM

2.2.1 General Description

Resistive random-access memory (ReRAM/RERAM) is a new non-volatile memory type that changes its resistance based on the applied voltage. Although ReRAM was initially designed to be a low energy Flash alternative, ReRAM can also mimic Dynamic RAM (DRAM) or a hard drive. Some reports claim that,[7], [10] ReRAM could in fact be used as a universal storage conduit (by using different materials with different characteristics tuned for the particular applications).

The ReRAM memory technology itself can potentially be used in two different contexts, based on the intended applications: Embedded and Standalone. Stand-alone memories are essentially supporting the main memory and the higher cache layers in general purpose computing platforms. They are optimized for yield and manufactured in high volume so the area concern is typically the most sensitive cost dimension. As a result, the design costs are spread across large unit volumes. Several recent ReRAM activities have been proposed to address this general-purpose market. However, stand-alone memories don't often come in the exact configuration required and moreover, their access times and energy per read/written word is far from minimal.

In contrast, embedded memories are integrated on-chip and they realize the lowest cache or scratchpad memory layers. They are tightly connected with the foreground memory (e.g. register files) that provides the data access for the logic core to accomplish the intended functions from the program being executed. High-performance embedded memory is a key component in modern system-on-chip (SoC) design, because of its high-speed and wide bus-width capability, which eliminates inter-chip communication and which allows high access bandwidth for peak workloads in the algorithm execution. As systems become implemented on a single chip, embedded memory has become the choice of many designers developing for such **SoCs** because it can enable additional capability in the end product due to its higher speed or lower power at the same cost or less than stand-alone memory.

2.2.2 ReRAM cell characteristics and system requirements:

The basic idea is that a dielectric, which is normally insulating, can be made to conduct through a filament or conduction path formed after application of a sufficiently high voltage. The conduction path formation can arise from different mechanisms, including defects, metal migration, etc. Once the filament is formed, it may be reset (broken, resulting in high resistance) or set (re-formed, resulting in lower resistance) by an appropriately applied voltage. Recent data suggest that the paths are best described as lineups of oxygen vacancies.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

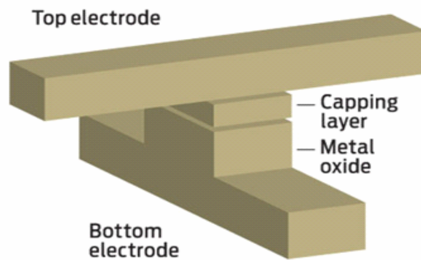

Figure 6 Basic ReRAM cell

A memory cell can be deduced from the basic memory cell in three different ways. In the simplest approach, the pure memory element can be used as a basic memory cell, resulting in a configuration where parallel bitlines are crossed by perpendicular wordlines with the switching material placed between wordline and bitline at every cross-point. This configuration is called a cross-point cell. Since this architecture will lead to a large "sneak" parasitic current flowing through non selected memory cells via neighboring cells, the cross-point array has a very slow read access. A selection element can be added to improve the situation, but this selection element consumes extra voltage and power. A series connection of a diode in every cross-point allows to reverse bias, zero bias, or at least partial bias non selected cells, leading to negligible sneak currents. This can be arranged in a similar compact manner as the basic cross-point cell. Finally a transistor device (ideally a MOS Transistor) can be added which makes the selection of a cell very easy and therefore gives the best random access time, but comes at the price of increased area consumption.

For random access type memories, transistor type architecture is preferred while the cross-point architecture and the diode architecture open the path toward stacking memory layers on top of each other and therefore are ideally suited for mass storage devices. The switching mechanism itself can be classified in different dimensions. First there are effects where the polarity between switching from the low to the high resistance level (reset operation) is reversed compared to the switching between the high and the low resistance level (set operation). These effects are called bipolar switching effects. On the contrary, there are also unipolar switching effects where both set and reset operations require the same polarity, but different voltage magnitude.

Another way to distinguish switching effects is based on the localization of the low resistive path. Many resistive switching effects show a filamentary behavior, where only one or a few very narrow low resistive paths exist in the low resistive state. In contrast, also homogenous switching of the whole area can be observed. Both effects can occur either throughout the entire distance between the electrodes or happen only in proximity to one of the electrodes. Filamentary and homogenous switching effects can be distinguished by measuring the area dependence of the low resistance state.

A set of typical system-level requirements for the ReRAM when used in an embedded wireless or multimedia application, include the following:

- Read access speeds below 1 ns, write access speeds can tolerate a larger latency but not more than 2 cycles in L1 and 8 cycles in L2 normally.
- Energy per read/write access should be reduced as much as possible because of the portable context, preferably in the same range as the foreground memory access

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

so around 100fJ/read word (writing happens less frequently so it can be more expensive).

- Area should remain low but due to the limited sizes (16-128Kb for L1 and 128Kb-8Mb for L2) the cell area can be a bit relaxed compared to standalone contexts.
- Endurance should be high: preferably up to 10^{13} for read access, but retention in the L1-L2 layers can be relaxed to a few days or even few hours. The limited amount of data that has to be stored longer can easily be backed up in an off-chip NVM with long retention. In order to meet these requirements we will have to heavily tune the ReRAM cell and periphery circuits, because standalone configurations do not even come close to the combination of these critical requirements.

3. PROCESSOR ARCHITECTURE

3.1 ADRES

3.1.1 Generally

Nowadays due to the increase of wireless systems, Digital Signal Processors (DSPs) and ASIC do not meet with the requirements of high-performance devices. DSP-based solutions cannot provide high speed and enough efficiency, whereas ASIC's design requires high engineering cost and slow time-to-market due to the specific application that has to be designed.

Coarse-Grained Reconfigurable Architecture (CGRA) is a good proposal as a solution, since it can offer higher performance, less power requirements and reduced chip area in typical communication applications. This is feasible due to coarse-grained operations. CGRAs include an array of basic functional units, which can execute either word level or sub-word level operations instead of bit-level operations found in common FPGAs .

ADRES in cooperation with DRESC combines a VLIW processor (Very Long Instruction Word) processor and a CGRA. DRESC handles the switching between these two different execution modes so the programming of the processor has been significantly simplified. Due to this combination ADRES can offer both instruction level and loop level parallelism.

3.1.2 ADRES Architecture

ADRES (Architecture for Dynamically Reconfigurable Embedded Systems) [3] is a processor architecture designed for wireless and multimedia processing in single and multiprocessor systems. These processors are suitable for future mobile terminals like software-defined radios. They combine power efficiency, great performance and flexibility.

Moreover ADRES is a prototype defined by an Extensible Mark-up Language (XML)-based architecture, language that allows the building of the processor with a scalable number of Functional Units (FUs), Register Files (RFs) and wires. The applications for an ADRES can be exclusively programmed in high-level language (C) and are compiled with the DRESC compiler.

In the higher level, it combines a VLIW processor and a CGRA (Coarse Grained Reconfigurable Array) in the same structure. In general the loops are mapped to the Reconfigurable Array, whereas the rest code is mapped to the VLIW processor. The exchange of data between the VLIW processor and the Reconfigurable Array is resolved through the common Register Files and the common accesses in memory.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

The execution of CGRA is controlled by the control unit CGA, whereas the VLIW processor has a certain way of pipeline with 3 stages (fetching, decode and execution) which are controlled by the instruction memory. The configuration memory is loaded through the interface, after the system reset.

Basic blocks in the ADRES architecture are explained below:

- Functional Units

The core element in the ADRES architecture is a functional unit. The instruction set that is already defined for the FUs by the compiler, can be extended by instructions which user can define. Each FU can be specified according to its functionality like memory load and store
unit, logic unit, arithmetic unit

- Register Files

Register Files are used to store intermediate data. In ADRES architecture there are two types:

- ✓ Predicate register files (1-bit to store the predicate signal)
- ✓ Data register files (same bit width as FUs)

When there is loop level parallelism, corruption of data can happen due to overlapped execution of different iterations of the loop body. Therefore the life-time of the same variable may overlap over different iterations. In order to accommodate this situation, each of the simultaneously live instances needs different register. Furthermore, the name of the used register has to be defined clearly, either in hardware or in software.

ADRES uses two kinds of register naming; rotating register file (RRF) and modulo variable expansion (MVE). In the first kind which is hardware-based renaming method, the physical address is calculated by combining the virtual address with the iteration number. The MVE which is software-based technique unrolls the loop body and renames the register access to insure that there is no name confliction. Since this solution expands significantly the loop, which means more reconfigurable contexts ADRES architecture adopts mostly the RRF solution.

- Routing Networks

The routing Network consists of a data network and a predicate network. The first one routes the normal data among FUs and RFs, whereas the predicate network directs 1-bit predicate signals. These two networks do not have necessarily the same topology and cannot overlap due to different data widths.

Since ADRES architecture is designed to be compiler-friendly and software-like, it is expected the clock speed to be already determined and the compiler does not need to do timing analysis as it happens with FPGA. This is the reason why there are some constraints on how the routing networks are constructed. As far as ADRES is concerned, most of the routing is done by direct point-to-point interconnections, consisting of wires and multiplexors. Therefore, the timing can be statically analyzed at design time. When the compiler maps different kernels, the mapping will not change the timing behavior for a given ADRES fabric. If long interconnections are needed, a register can be introduced in the data network to limit the delay of the critical path. FUs can be viewed as part of the routing network.

- Memories

ADRES architecture consists of a global memory which is connected to VLIW unit. Some configuration memories are also used, so as to support intermediate operations on the CGRA mode. These configuration memories are used to feed source ports of CGRA FUs. These memories will be analyzed further in BoADRES section.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

3.2 DRESC COMPILER

A design starts from a C-language description of the application. The profiling/partitioning step identifies the candidate loops for mapping on the CGA based on the execution time and possible speed-up. The source-level transformation step tries to rewrite the kernel so as to make it able for pipeline and with aim to maximize the performance. In the next step IMPACT is used. IMPACT is a VLIW compiler framework, to parse the C code and do analysis and optimization. IMPACT also has an intermediate representation, called Lcode, which is used as the input for the scheduling. As far as architecture is concerned the target architecture is described in an XML-based language. The parser and abstraction steps transform the architecture to an internal graph representation. Taking this representation as an input, a novel modulo scheduling is applied to achieve parallelism for kernels, whereas ILP scheduling techniques are applied to discover the available moderate parallelism for the non-kernel code. The communication between these two parts is automatically identified and handled by some tools. Finally, the tools generate scheduled code for both the reconfigurable array and the VLIW. The outcome is simulated by a co-simulator.

3.3 BoADRES

BoADRES is a scalable baseband processor template for Gbps radios. It is a new generation power-efficient, high performance and flexible processor architecture, designed to achieve the data processing challenges for future mobile terminals. It is a dual-core implementation that adds thread-level parallelism to data-level parallelism. BoADRES can be implemented in 40-nm and can run at 700 MHz clock frequency.

It correlates a very-long instruction word (VLIW) processor with a coarse-grain-array (CGA) accelerator through a shared central register file. Number and type of memories in the processor listed as below:

- 2 scalar, 1 global scalar memories
- 4 vector (scratch pad) memories
- 2 levels of configuration memory

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

4. RERAM Implementation for Data Memory Replacement

4.1 Experimental Setup

In order to evaluate the performance of the two different types of memories presented above, we created *5 different architectural solutions* of the FFT processor. The ASIP used for the validation is optimized to execute radix 8 FFT stages and supports several types of FFT's ranging from a 64 point FFT to a 4K point FFT.

The used FFT is a 2K point FFT of 11 stages. The stages are named from 0 to 10. They are combined by 2, except from the last stage that is single. We executed two different simulations for every architectural design. One with only the first two stages of the FFT and one with full stages.

We focused on checking the cycles that kernel runs in order to evaluate and compare the performance of every architectural design that was created. Moreover we created in every design counters for the read and write accesses. In this way we can also provide some energy numbers.

4.1.1 Brief Description of the FFT Algorithm

There are many different FFT algorithms involving a wide range of mathematics, from simple complex-number arithmetic to group theory and number theory. The DFT is obtained by decomposing a sequence of values into components of different frequencies. This operation is useful in many but computing it directly from the definition is often too slow to be practical. An FFT is a way to compute the same result more quickly: computing the DFT of N points in the naive way, using the definition, takes $O(N^2)$ arithmetical operations, while a FFT can compute the same DFT in only $O(N \log N)$ operations. The difference in speed can be enormous, especially for long data sets where N may be in the thousands or millions. In practice, the computation time can be reduced by several orders of magnitude in such cases, and the improvement is roughly proportional to $N / \log(N)$. This huge improvement made the calculation of the DFT practical; FFTs are of great importance to a wide variety of applications, from digital signal processing and solving partial differential equations to algorithms for quick multiplication of large integers.

"Decimation-in-time" [5] and "Decimation-in-frequency" [5] fast Fourier transforms (FFTs) are the simplest FFT algorithms. Like all FFTs, they gain their speed by reusing the results of smaller, intermediate computations to compute multiple DFT frequency

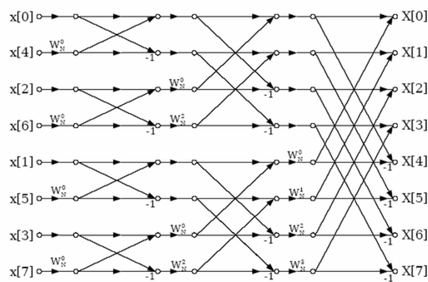

Figure 7 Decimation in Time

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

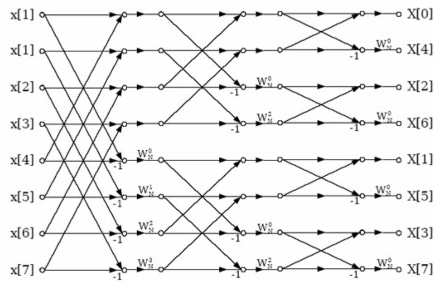

Figure 8 Decimation in Frequency

4.1.2 Description of the Architectural Designs

As referred above, 5 different architectural designs were created for our experiments. The first architectural design ***"SRAM"***, which is the reference architecture, is the design with no additional logic; since SRAM modules can be accessed within one cycle for both read and write operations. The second one ***"Drop-In"*** replaces all the data memories with ReRAM modules with additional logic. A stall mechanism is used since ReRAM modules can be accessed within one clock cycle for the write, whereas more cycles are needed for the read operation. The rest three designs are sub-banking schemes. The third design is the ***"sub-banked scheme of SRAM"*** design. The rest two are the ***"sub-banked schemes of ReRAM designs".*** Two different ways of choosing in which bank every read or write operation goes were designed. In the first case the last bit of the address is used to choose the bank. In this way all the odd addresses are mapped to bank0 and all the even to bank1. In the other case the 6th bit is checked, in order to pick up in which of the two banks the operation is applied.

Further Analysis:

<u>SRAM Design</u>: This is the reference design as mentioned above. Here there is a simple process called p_entry(), which handles both the read and the write operations

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

```
void mem_SRAM::p_entry()
{
  wait();
  read_counter = 0;
  write_counter = 0;
  while(1) {
  // READ
  if (rd_en.read() == 1)  {
          int read_addr = address.read();
          data_out.write(data[read_addr]);
          read_counter++;
                              }
  // WRITE
  if (wr_en.read() == 1) {
          int write_addr = address.read();
          data[write_addr] = data_in.read();
          write_counter++;
                             }
  counter++;
  wait();
          }
}
```

Source Code 1 SRAM entry process

SRAM_banked Design: For the design of the SRAM_banked scheme we used as template the SRAM design. We created a new module called SRAM_banked which has identical ports with the SRAM module.

```
SC_MODULE(mem_SRAM_banked)
{
  sc_in<bool>                              clock;
  sc_in<bool>                              reset;
  sc_in<sc_uint<VDM_ADDR_WIDTH> >          address;      // word
  sc_out<sc_biguint<VDM_DATA_WIDTH> >      data_out;     // vword
  sc_in<bool>                              rd_en;
  sc_in<sc_biguint<VDM_DATA_WIDTH> >       data_in;      // vword
  sc_in<bool>                              wr_en;
}
```

Source Code 2 SRAM_banked module definition

Two instances of the SRAM module, the bank0 and bank1 instance were created too, as well as some internal signals for every bank. These signals were mapped to the already existing ports of SRAM.

```
SC_HAS_PROCESS(mem_SRAM_banked);
  mem_SRAM_banked(sc_module_name name) {

  mem_SRAM* bank0 = new mem_SRAM("bank0");
  bank0->clock(clock);
  bank0->reset(reset);
  bank0->address  (bank0_address);
  bank0->data_in  (bank0_data_in);
  bank0->data_out (bank0_data_out);
  bank0->rd_en    (bank0_rd_en);
  bank0->wr_en    (bank0_wr_en);

  mem_SRAM* bank1 = new mem_SRAM("bank1");
  bank1->clock(clock);
  bank1->reset(reset);
  bank1->address  (bank1_address);
  bank1->data_in  (bank1_data_in);
  bank1->data_out (bank1_data_out);
  bank1->rd_en    (bank1_rd_en);
  bank1->wr_en    (bank1_wr_en);
}
```

Source Code 3 Instances of SRAM_banked module

The figure that follows visualizes the above instantiations. In this figure the ports and the connections between the original memory and the sub-banked scheme are represented. For selecting between the signals of the banks some muxes2x1 are used. Different colors have been used to make it easier to distinguish every signal that is connected to the original port of the memory. Clock and reset ports of the bank modules, are directly connected to the port of the SRAM module respectively.

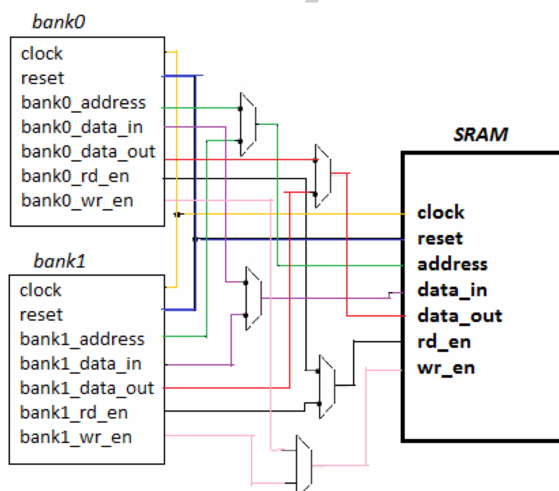

Figure 9

For the SRAM_banked module a new process called p_sel() was created. In this process the selection between the two banks is being held. The selection is based on the last bit of the address so all the even addresses go to the bank0 and all the uneven to the bank1. Depending on which bank is selected, the relative signals take the values of the ports and the signals of the other bank are reset. The addresses of every bank are shorter for one bit, as the original address is shifted to the right by 1. Moreover there is another signal created which is called sel_delayed. Since signal assignments do not have an immediate effect on the signal, but happen only after a default time, which is called delta delay, there is a need to create a kind of buffer which will keep the previous value of the data_out, when a write operation occurs. If the sel_delayed signal has the value 0 then the first bank is selected for writing, otherwise the bank1 is selected.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

```
void mem_SRAM_banked::p_sel()
{
  wait();
  while(1) {
  int read_addr = address.read();

  if ((address.read() % 2) == 0) {              // if address even
          bank0_address.write(read_addr>>1);
          bank0_data_in.write(data_in);
          bank0_rd_en.write(rd_en);
          bank0_wr_en.write(wr_en);
          bank1_address.write(0);
          bank1_data_in.write(0);
          bank1_rd_en.write(0);
          bank1_wr_en.write(0);
 } else {
          bank0_address.write(0);              // if address uneven
          bank0_data_in.write(0);
          bank0_rd_en.write(0);
          bank0_wr_en.write(0);
          bank1_address.write(read_addr>>1);
          bank1_data_in.write(data_in);
           bank1_rd_en.write(rd_en);
          bank1_wr_en.write(wr_en);
  }

  if (sel_delayed == 0) {
    data_out.write(bank0_data_out.read());
  } else {
    data_out.write(bank1_data_out.read());
  }
  wait();
 }
}
```

Source Code 4 SRAM_banked process for bank selection

In order to give value to the sel_delayed signal we created another process called p_sel_delayed(). It works in the same way as it happens with the selection of the banks. So if the address is even, the signal is assigned with the value 0, otherwise with the value 1.

```
void mem_SRAM_banked::p_sel_delayed() {
 sel_delayed = 0;
 wait();
 while (1) {
  if ((address.read() % 2) == 0) {
        sel_delayed.write(0);
  } else {
   sel_delayed.write(1);
  }
  wait();
 }
}
```

Source Code 5 p_sel_delayed process

ReRAM Design: In this design two different processes for the write and read operations are created called p_write() and p_read() respectively. Moreover a stall mechanism is created in order to handle the long write latency of the ReRAM. For this reason there is another process called p_stall(), which will be presented below. For the p_write() process some additional signals are created. The signal write_status represents the status in which the write operation is at a certain moment, whereas write_addr_buffer and write_data_buffer, operate as buffers for the address and the data_in respectively. Write_status signal, is given initially the value 9 which means that a new write operation occurs. As the write operation evolves, it is subtracted by 1 in every cycle. When the write_status is 1 and the wr_en is high, then the write operation is completed, so the data from the write_data_buffer signal are written in the memory.

So three different status of the memory occur. 1) Writing in the memory 2) Memory Idle because a new write instruction has occurred 3) Memory Idle due to proceeding of a read operation. As we can see in the following source code part the write_status signal defines in which of the three is the memory status. If write_status > 0 but not 1 the write_status is subtracted by 1 to achieve the delay of 10 cycles.

Normally a delay of 10 cycles is caused by this way of handling the write operations per every operation. In the two different sub-banked schemes we will try to handle in the most effective way the delay of two sequential writings, by dividing the memory in two parts having two different stall signals that do not affect one the other. In this way we can achieve to reduce the delay cycles from 20 to 11 for two sequential write operations.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΕΙ

```
void mem_ReRAM::p_write() {
  wait();
  write_status = 0;
  write_addr_buffer = 0;
  write_data_buffer = 0;
  write_counter = 0;
  while(1) {
        if (write_status > 0) {
          if (write_status == 1) {          // write
                int write_addr = write_addr_buffer;
                data[write_addr] = write_data_buffer;
                write_counter++;
          }
        write_status--;
        } else {                            // memory idle
          if ((wr_en.read() == 1)) {        // new write instruction
                write_addr_buffer = address.read();
                write_data_buffer = data_in.read();
                write_status = WRITE_LATENCY;
          } else {                          // memory idle or memory read
          write_status = 0;
          }
        }
   wait();
  }
}
```

Source Code 6 ReRAM writing process

As for the p_read() process is needed just to examine if the rd_en is high and the stall signal is low. Then the read operation can occur.

```
void mem_ReRAM::p_read() {
  wait();
  read_counter = 0;
  while(1) {
   // READ
    if ((rd_en.read() == 1) and (stall.read() == 0)) {
          int read_addr = address.read();
         data_out.write(data[read_addr]);
         read_counter++;
    }
   counter++;
   wait();
  }
}
```

Source Code 7 ReRAM reading process

As far as the p_stall process is concerned, the write_status signal is examined. If it is positive it means that a write operation is still in progress so the stall signal goes high. Otherwise the stall signal becomes zero.

```
void mem_ReRAM::p_stall() {
  wait();
  while(1) {
    if (write_status > 0)
          stall.write(1);
    else
          stall.write(0);
   wait();
  }
}
```

Source Code 8 ReRAM stall

ReRAM banked-version1: Similarly with the SRAM_banked version, the ReRAM module was used as template and two instances of this module, the bank0 and bank1, were created. The selection pattern in this version is similar to this one of the SRAM_banked version. There is a difference in the p_stall process in which, the two stall signals of the banks are examined in order some value to be given at the stall signal. If either bank0_stall or bank1_stall is 1 then the stall signal becomes 1, otherwise it takes the value 0.

```
void mem_ReRAM_banked::p_stall() {
  wait();
  while(1) {
    if ((bank0_stall.read()== 1) or (bank1_stall.read() == 1))
          stall.write(1);
    else
          stall.write(0);
  wait();
  }
}
```

Source Code 9

ReRAM banked-version2: This design is very similar with the previous one at many points. The only thing that changes is the way of the selection between the two banks. As it is referred above, in the first version the access pattern is not taken into consideration, so we use the last bit to select the bank. In this version the access pattern plays an essential role. Since the addresses are of 64 bit, we examine the 6th bit for the bank selection, since this is the bit that changes from iteration to iteration. The new address of every bank consists of the original address, without the 6th bit that is excluded.

ΠΑΝΕΠΙΣΤΗΜΙΟ

```
void mem_ReRAM_banked::p_sel() {
  wait();
  while(1) {
  sc_uint<VDM_ADDR_WIDTH> c;
  c = ((address.read() >> 6) & 0x1);
 int read_addr;
 read_addr = (((address.read() >> 7) << 6) xor (address.read() & 0x3F));
   if (c == 0) {
          bank0_address.write(read_addr);
          bank0_data_in.write(data_in);
          bank0_rd_en.write(rd_en);
          bank0_wr_en.write(wr_en);
          bank1_address.write(0);
          bank1_data_in.write(0);
          bank1_rd_en.write(0);
          bank1_wr_en.write(0);
    } else {
          bank0_address.write(0);
          bank0_data_in.write(0);
          bank0_rd_en.write(0);
          bank0_wr_en.write(0);
          bank1_address.write(read_addr);
          bank1_data_in.write(data_in);
          bank1_rd_en.write(rd_en);
          bank1_wr_en.write(wr_en);
    }

    if (sel_delayed == 0){
          data_out.write(bank0_data_out.read());
    } else {
          data_out.write(bank1_data_out.read());
    }
  wait();
  }
}
```

Source Code 10

Finally similarly the p_sel_delayed() process have change to follow the way pattern of selection.

```
void mem_ReRAM_banked::p_sel_delayed() {
  sel_delayed = 0;
  wait();
  while (1) {
  sc_uint<VDM_ADDR_WIDTH> c;
  c = ((address.read() >> 6) & 0x1);
    if (c == 0)
          sel_delayed.write(0);
    else
          sel_delayed.write(1);
   wait();
  }
}
```

Source Code 11

4.2 Simulation Results

Below the screenshots of every simulation are quoted. The screenshots of the two stages FFT arose after we changed the firmware in order only the first two stages of the FFT to be executed.

Except from the main part of simulation where the read and write operation happen there are also the initialization of memory and at the end the dumping of memory. For our results we will take into consideration only the cycles for the kernel, ignoring the number of the whole cycles that test took. Moreover some debugging statements where created so as to watch the read and write signals in our simulation as well as the stall signal for the ReRAM implementation. These are not obvious in the screenshots that follow as they do not give any useful results for our research and they were used only for debugging.

Two Stages FFT

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡ

```
xterm (on mammut01)
MSG :   -1 : Stack range [0..255] has been loaded from elf executable for stack
memory vVDM1 and stack pointer VSP1 : loadprogram (elf)
MSG :   -1 : Stack range [0..255] has been loaded from elf executable for stack
memory vVDM2 and stack pointer VSP2 : loadprogram (elf)
Starting sim...
@0 s: reset received
Reset (Triggered inside module VDM2)
Reset (Triggered inside module VDM1)
Reset (Triggered inside module tb_VDM2)
Reset (Triggered inside module tb_VDM1)
@20485 us: memories initialized
@26485 us: Simulation terminated normally, report follows...
Simulation terminated normally, report follows...
Cycle count report
------------------
Test took 2650 cycles
 + kernel[1] ran 593 cycles
Memory dump (Triggered inside module tb_VDM2)
Memory dump (Triggered inside module tb_VDM1)
Memory Dump successful

Info: /OSCI/SystemC: Simulation stopped by user.
SUCCESS: test complete
douka@mammut01>
```

Screenshot 1 SRAM

Simulation of the reference design SRAM for the two stages of FFT algorithm without additional logic.

```
xterm (on mammut01)
MSG :   -1 : Stack range [0..255] has been loaded from elf executable for stack
memory vVDM2 and stack pointer VSP2 : loadprogram (elf)
Starting sim...
@0 s: reset received
Reset (Triggered inside module VDM2.bank1)
Reset (Triggered inside module VDM2.bank0)
Reset (Triggered inside module VDM1.bank1)
Reset (Triggered inside module VDM1.bank0)
Reset (Triggered inside module tb_VDM2)
Reset (Triggered inside module tb_VDM1)
@20485 us: memories initialized
@26485 us: Simulation terminated normally, report follows...
Simulation terminated normally, report follows...
Cycle count report
------------------
Test took 2650 cycles
 + kernel[1] ran 593 cycles
Memory dump (Triggered inside module tb_VDM2)
Memory dump (Triggered inside module tb_VDM1)
Memory Dump successful

Info: /OSCI/SystemC: Simulation stopped by user.
SUCCESS: test complete
douka@mammut01>
```

Screenshot 2 SRAM Banked

Simulation of the banked scheme of the reference SRAM design without additional logic too.

```
xterm (on mammut01)
MSG :   -1 : Stack range [0..255] has been loaded from elf executable for stack
memory vVDM1 and stack pointer VSP1 : loadprogram (elf)
MSG :   -1 : Stack range [0..255] has been loaded from elf executable for stack
memory vVDM2 and stack pointer VSP2 : loadprogram (elf)
Starting sim...
@0 s: reset received
Reset (Triggered inside module VDM2)
Reset (Triggered inside module VDM1)
Reset (Triggered inside module tb_VDM2)
Reset (Triggered inside module tb_VDM1)
@204625 us: memories initialized
@233755 us: Simulation terminated normally, report follows...
Simulation terminated normally, report follows...
Cycle count report
------------------
Test took 23377 cycles
 + kernel[1] ran 2897 cycles
Memory dump (Triggered inside module tb_VDM2)
Memory dump (Triggered inside module tb_VDM1)
Memory Dump successful

Info: /OSCI/SystemC: Simulation stopped by user.
SUCCESS: test complete
douka@mammut01>
```

Screenshot 3 ReRAM

Simulation of the ReRAM implementation for the two stages FFT. In this implementation a stall mechanism was created, so as to handle the big latency caused in write operations (10 cycles).

```
xterm (on mammut01)
Starting sim...
end_of_elaboration memoryVDM1
end_of_elaboration memoryVDM2
@0 s: reset received
Reset (Triggered inside module VDM2.bank1)
Reset (Triggered inside module VDM2.bank0)
Reset (Triggered inside module VDM1.bank1)
Reset (Triggered inside module VDM1.bank0)
Reset (Triggered inside module tb_VDM2)
Reset (Triggered inside module tb_VDM1)
@225085 us: memories initialized
@254845 us: Simulation terminated normally, report follows...
Simulation terminated normally, report follows...
Cycle count report
------------------
Test took 25486 cycles
 + kernel[1] ran 2960 cycles
Memory dump (Triggered inside module tb_VDM2)
Memory dump (Triggered inside module tb_VDM1)
Memory Dump successful

Info: /OSCI/SystemC: Simulation stopped by user.
SUCCESS: test complete
douka@mammut01>
```

Screenshot 4 ReRAM Banked (version 1)

Simulation of the banked scheme of the previous ReRAM design.. For the selection of banks the last bit of address is used.

```
Starting sim...
end_of_elaboration memoryVDM1
end_of_elaboration memoryVDM2
@0 s: reset received
Reset (Triggered inside module VDM2.bank1)
Reset (Triggered inside module VDM2.bank0)
Reset (Triggered inside module VDM1.bank1)
Reset (Triggered inside module VDM1.bank0)
Reset (Triggered inside module tb_VDM2)
Reset (Triggered inside module tb_VDM1)
@201835 us: memories initialized
@219015 us: Simulation terminated normally, report follows...
Simulation terminated normally, report follows...
Cycle count report
------------------
Test took 21903 cycles
 + kernel[1] ran 1711 cycles
Memory dump (Triggered inside module tb_VDM2)
Memory dump (Triggered inside module tb_VDM1)
Memory Dump successful

Info: /OSCI/SystemC: Simulation stopped by user.
SUCCESS: test complete
douka@mammut01>
```

Screenshot 5 ReRAM Banked (version 2)

One other banked scheme of the ReRAM implementation. For selecting each bank we use the 6th bit of the address.

Full Stages FFT

```
MSG :   -1 : Stack range [0..255] has been loaded from elf executable for stack
memory vVDM1 and stack pointer VSP1 : loadprogram (elf)
MSG :   -1 : Stack range [0..255] has been loaded from elf executable for stack
memory vVDM2 and stack pointer VSP2 : loadprogram (elf)
Starting sim...
@0 s: reset received
Reset (Triggered inside module VDM2)
Reset (Triggered inside module VDM1)
Reset (Triggered inside module tb_VDM2)
Reset (Triggered inside module tb_VDM1)
@20485 us: memories initialized
@67765 us: Simulation terminated normally, report follows...
Simulation terminated normally, report follows...
Cycle count report
------------------
Test took 6778 cycles
 + kernel[1] ran 4721 cycles
Memory dump (Triggered inside module tb_VDM2)
Memory dump (Triggered inside module tb_VDM1)
Memory Dump successful

Info: /OSCI/SystemC: Simulation stopped by user.
SUCCESS: test complete
douka@mammut01>
```

Screenshot 6 SRAM

Simulation of the reference design SRAM for the full stages of FFT algorithm without additional logic.

```
MSG :   -1 : Stack range [0..255] has been loaded from elf executable for stack
memory vVDM2 and stack pointer VSP2 : loadprogram (elf)
Starting sim...
@0 s: reset received
Reset (Triggered inside module VDM2.bank1)
Reset (Triggered inside module VDM2.bank0)
Reset (Triggered inside module VDM1.bank1)
Reset (Triggered inside module VDM1.bank0)
Reset (Triggered inside module tb_VDM2)
Reset (Triggered inside module tb_VDM1)
@20485 us: memories initialized
@67765 us: Simulation terminated normally, report follows...
Simulation terminated normally, report follows...
Cycle count report
------------------
Test took 6778 cycles
 + kernel[1] ran 4721 cycles
Memory dump (Triggered inside module tb_VDM2)
Memory dump (Triggered inside module tb_VDM1)
Memory Dump successful

Info: /OSCI/SystemC: Simulation stopped by user.
SUCCESS: test complete
douka@mammut01>
```

Screenshot 7 SRAM Banked

Simulation of the banked scheme of the reference SRAM design without additional logic too.

```
MSG :   -1 : Stack range [0..255] has been loaded from elf executable for stack
memory vVDM1 and stack pointer VSP1 : loadprogram (elf)
MSG :   -1 : Stack range [0..255] has been loaded from elf executable for stack
memory vVDM2 and stack pointer VSP2 : loadprogram (elf)
Starting sim...
@0 s: reset received
Reset (Triggered inside module VDM2)
Reset (Triggered inside module VDM1)
Reset (Triggered inside module tb_VDM2)
Reset (Triggered inside module tb_VDM1)
@204625 us: memories initialized
@390235 us: Simulation terminated normally, report follows...
Simulation terminated normally, report follows...
Cycle count report
------------------
Test took 39025 cycles
 + kernel[1] ran 18545 cycles
Memory dump (Triggered inside module tb_VDM2)
Memory dump (Triggered inside module tb_VDM1)
Memory Dump successful

Info: /OSCI/SystemC: Simulation stopped by user.
SUCCESS: test complete
douka@mammut01>
```

Screenshot 8 ReRAM

Simulation of the ReRAM implementation for the two stages FFT. In this implementation a stall mechanism was created, so as to handle the big latency caused in write operations (10 cycles).

```
xterm (on mammut01)
Starting sim...
end_of_elaboration memoryVDM1
end_of_elaboration memoryVDM2
@0 s: reset received
Reset (Triggered inside module VDM2.bank1)
Reset (Triggered inside module VDM2.bank0)
Reset (Triggered inside module VDM1.bank1)
Reset (Triggered inside module VDM1.bank0)
Reset (Triggered inside module tb_VDM2)
Reset (Triggered inside module tb_VDM1)
@225085 us: memories initialized
@411955 us: Simulation terminated normally, report follows...
Simulation terminated normally, report follows...
Cycle count report
------------------
Test took 41197 cycles
 + kernel[1] ran 18671 cycles
Memory dump (Triggered inside module tb_VDM2)
Memory dump (Triggered inside module tb_VDM1)
Memory Dump successful

Info: /OSCI/SystemC: Simulation stopped by user.
SUCCESS: test complete
douka@mammut01>
```

Screenshot 9 ReRAM Banked (version 1)

Simulation of the banked scheme of the previous ReRAM design. For the selection of banks the last bit of address is used.

```
xterm (on mammut01)
Starting sim...
end_of_elaboration memoryVDM1
end_of_elaboration memoryVDM2
@0 s: reset received
Reset (Triggered inside module VDM2.bank1)
Reset (Triggered inside module VDM2.bank0)
Reset (Triggered inside module VDM1.bank1)
Reset (Triggered inside module VDM1.bank0)
Reset (Triggered inside module tb_VDM2)
Reset (Triggered inside module tb_VDM1)
@201835 us: memories initialized
@338355 us: Simulation terminated normally, report follows...
Simulation terminated normally, report follows...
Cycle count report
------------------
Test took 33837 cycles
 + kernel[1] ran 13645 cycles
Memory dump (Triggered inside module tb_VDM2)
Memory dump (Triggered inside module tb_VDM1)
Memory Dump successful

Info: /OSCI/SystemC: Simulation stopped by user.
SUCCESS: test complete
douka@mammut01>
```

Screenshot 10 ReRAM Banked (version 2)

One other banked scheme of the ReRAM implementation. For selecting each bank we use the 6th bit of the address.

4.3 Energy Modeling and Results

In the first two tables that follow we have concluded all the necessary numbers for the energy model that we want to examine. These numbers have been extracted from a previous Master Thesis and were used in order to give an estimation of the energy consumption in every case [8]. In the first table the energy numbers for every read and write access, for both SRAM and ReRAM model are quoted. Moreover the energy leakage per cycle is given.

	SRAM	ReRAM
read energy per access	0,86	0,66
write energy per access	1,13	2,60
energy leakage per cycle	1,73	0,35

Tabel 1 Energy Numbers for SRAM and ReRAM

In the second table there are the numbers of the computational energy that is consumed for the specific application of the 11-stages FFT algorithm. This is the dynamic energy that is consumed and it changes every time that the application changes. Using this number, we counted the computational energy consumed for the 2-stages FFT, with the rule of three.

	Processor
dynamic	15.594,06
static per cycle	1,00

Tabel 2 Computational Energy

In the rest two tables we have concluded the results of all the simulations that we ran for every different implementation. We have included both the 2-stages and the full-stages implementation. In the first column there are the *"Cycles in Kernel".* In the second and the third column we counted the *"Write and Read Accesses*" for every case. These numbers are used to count the results of the fourth and fifth column according to the energy numbers that we have received for the two different kinds of memory and are quoted above.

For computing the 5th column of these two tables we used the energy leakage number for every one of the two different models. We used the numeral Energy ***Leakage = energy leakage per cycle * Cycles in* Kernel** for every different implementation. The 4th column is the total energy that is consumed for the specific application. For the Full-stages FFT the *computational energy* is 15.594,06 whereas for the 2-stages FFT is 2.835,28. The numeral for computing this total energy that is consumed is ***Energy = Computational Energy + (Read***

Accesses * read energy per access) + (Write Accesses * write energy per access) + Leakage energy.

	Cycles in Kernel	Write Accesses	Read Accesses	Energy	Energy Leakage
SRAM	593	256	329	4.433,39	1.025,89
SRAM_banked	593	256	329	4.433,39	1.025,89
ReRAM	2.891	256	329	4.729,87	1.011,85
ReRAM_banked (version 1)	2.960	256	329	4.754,02	1.036,00
ReRAM_banked (version 2)	1.711	256	329	4.316,87	598,85

Tabel 3 2-Stages FFT

	Cycles in Kernel	Write Accesses	Read Accesses	Energy	Energy Leakage
SRAM	4.721	1.536	1.755	27.006,37	8.167,33
SRAM_banked	4.721	1.536	1.755	27.006,37	8.167,33
ReRAM	18.152	1.536	1.755	27.099,16	6.353,20
ReRAM_banked (version 1)	18.671	1.536	1.755	27.280,76	6.534,85
ReRAM_banked (version 2)	13.645	1.536	1.755	25.521,71	4.775,75

Tabel 4 Full-Stages FFT

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

4.3 Discussion

In this section we will analyze and explain the effects caused by the different implementations and organization of the memory, as they are presented in the previous table.

SRAM and SRAM_banked versions:

The number of cycles that kernel runs for the cases of SRAM and SRAM_banked is the same. That can be explained by the fact that no additional logic was used for the implementation of the SRAM module. Hence, it makes no difference that the addresses go to two different banks. By this it is clear, that since the SRAM consumes only 1 cycle per writing (as it happens with the reading too), we cannot improve the total cycles running by changing the memory. For having some improvement we should change the firmware. Something like that is not examines within this Master Thesis.

ReRAM-"Drop-In" version

In the “Drop-In” replacement, we can notice that the number of cycles is quite bigger than this one of the SRAM case. This is of course caused by the ReRAM write latency in the actual memory. For handling this latency, we used some additional logic in the implementation of the ReRAM module. We created a stall mechanism so as to handle the 10 cycles’ delay that is added because of the long write latency of ReRAM.

ReRAM_banked versions

In the first version of ReRAM sub-banked scheme, the sub-banks do not take into account the address access pattern. So the last bit is used to choose in which bank every operation (read or write) is applied. As we can notice from the log files, the addresses are written continuously in the same bank. This causes a stall of 20 cycles for two sequential writes. Therefore there is no difference in the number of cycles, so it doesn’t offer any optimizing in the performance.

In the second version of ReRAM sub-banked scheme, the sub-banks take into account the address access pattern. The offset of sequential writes is 64, hence the 6th bit is used for the bank selection. The outcome of the applied technique is two sequential writes to go to two different banks, one in bank0 and one in bank1. This causes a stall of 11 cycles, instead of 20, since every bank has its own stall mechanism. In this way the optimizing in the number of cycles is quite significant (approximately 25% for the full stages version and 50% for the 2-stages version).

As far as the energy models are concerned we wanted to examine the differences between the SRAM and ReRAM model. As it is obvious from the *Energy Numbers for SRAM and ReRAM* table, the read energy that is consumed per cycle for the ReRAM model is less than the SRAM one, but the write energy is significantly more. That is the big problem of the ReRAM model. For this reason the total energy that is consumed for the simple ReRAM or the ReRAM_banked version1 model is bigger than these of the SRAM or SRAM_banked model. But when it comes to the ReRAM_banked version2 model the total energy number seems to give an improved outcome which is more obvious in the full stages FFT where the Cycles in Kernel and consequently the read and write accesses grow in number. This happens mainly due to the reduced number of Cycles that Kernel runs.

Below we quote some charts that illustrate the results referred above:

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

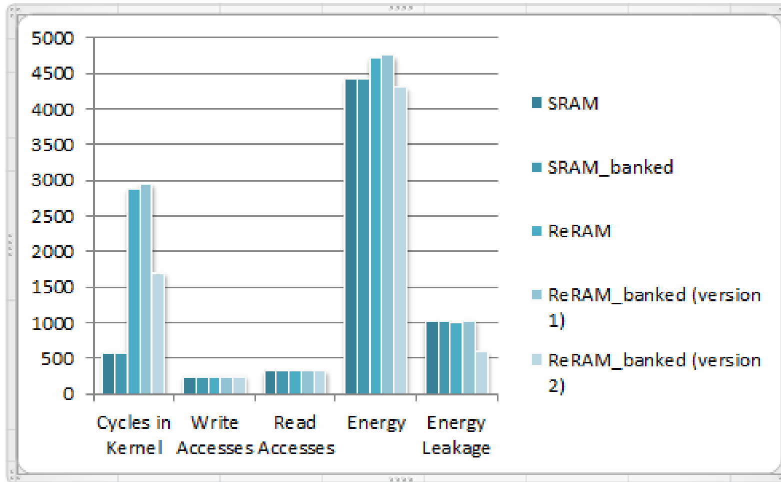

Chart 1

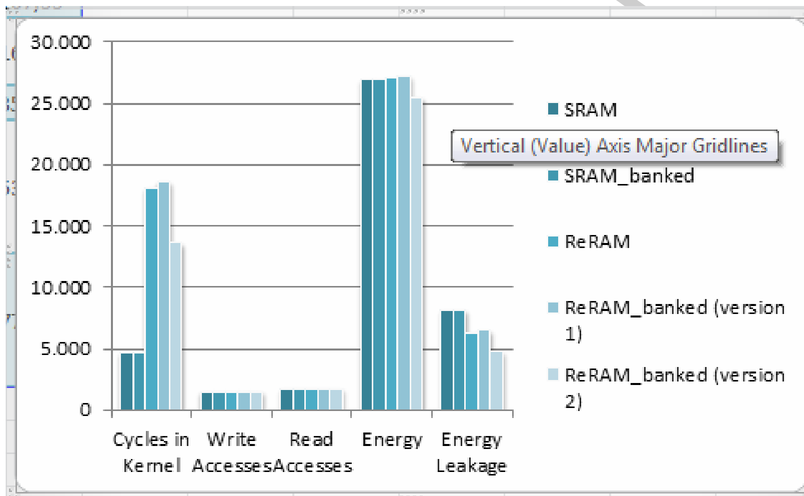

Chart 2

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

5. Conclusions and Future Work

As a conclusion, it is noticed that an emerging NVM ReRAM, offers some significant advantages in the energy as shown above. Especially a banked version, which takes into consideration the memory access pattern, gives some promising results. Aim of this Master Thesis was to examine the effect of different implementations of the data memory in an ASIP.

Therefore it has to be underlined, that the banked solution does not hide the extra cycles that the long write latency of ReRAM causes yet, in a very effective way. Hence, the overhead is still large enough, to create a large total energy gain, even though the energy cost (both dynamic and leakage) is better for the reads in the ReRAM implementation.

For the reasons mentioned above, some other architectural options of the ReRAM is being examined in Imec now. For example, architectures based on “line buffers” or “Very Wide Register” interfaces. These architectural solutions aim at reducing the energy cost that is caused because of the long write latency of the ReRAM and hence to reduce the total energy cost of the whole application running on an ASIP.

References

[1] Arash Azizi Mazreah, Mohammad T. Manzuri Shalmani, Hamid Barati, and Ali Barati; “A Novel Four-Transistor SRAM Cell with Low Dynamic Power Consumption”; 2008; World Academy of Science, Engineering and Technology

[2] Bingfeng Mei, S erge Vemalde, Diederik Verkes, Hugo De Man, Rudy Lauwereins; “DRESC: A Retargetable Compiler for Coarse-Grained Reconfigurable Architectures”; Department of Electrical Engineering, Kdtholic Universiteit Leuven, Belgium, Department of Electrical Engineering, Vrije Universiteit Brussel, Belgium; 2002; IEEE

[3] B. Mei, M. Berekovi, J-Y. Mignolet; ADRES & DRESC: Architecture and Compiler for Coarse-Grain Reconfigurable Processors; Chapter 6; 2007; Springer

[4] Chen-Wei Lin, Chih-Hsiang Ho, Chao Lu, Mango C.-T. Chao, and Kaushik Roy; “A Process/Device/Circuit/System Compatible Simulation Framework for Poly-Si TFT Based SRAM Design”; Department of Electronics Engineering, National Chiao Tung University, Taiwan, 30010

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

[5] Craig Lage, James D. Hayden, Chitra Subramanian; "Advanced SRAM Technology - The Race Between 4T and 6T Cells"; Advanced Products Research and Development Laboratory Motorola Inc., 3501 Ed Bluestein Blvd., Austin, Tx. 78721

[6] Describing Synthesizable RTL in SystemC; Version 1.0; May 2001; Synopsys, Inc.

Understanding Static RAM Operation; Applications Note; IBM

[7] Hiroyuki Akinaga, Hisashi Shima; "Resistive Random Access Memory (ReRAM) Based on Metal Oxides"; 12 December 2010; Proceedings of the IEEE

[8] Manu Perumkunnil Komalan, Francky Catthoor; "ReRAM circuit Exploration and Optimization"; IMEC Reports

[9] Linh Hong; "Applications Emerging to Employ Embedded Non---Volatile Memory"; May

2011

[10] Matthias Hartmann; "Low Energy Memristor-based (ReRAM) Data Memory Architecture in ASIP Design"; Proposed on Date 2012 Conference

[11] Nikos Andrikos, Luciano Lavagno; "Optimal and Heuristic Scheduling Algorithms for

Asynchronous High-Level Synthesis"; Department of Electronic Engineering, Politecnico di Torino, Italy; 2011; IEEE

[12] SRAM Technology; Integrated Circuit Engineering Corporation

Stefan Cosemans, Wim Dehaene, Francky Catthoor; "A Low Power Embedded SRAM for Wireless Applications"; 2006; ESAT-MICAS, K.U. Leuven Kasteelpark Arenberg 10, B-3001 Leuven, Belgium; IEEE

[13] SystemC Tutorial; Version 2.0 User's Guide; 1996-2002

[14] Target Compiler Technologies NV; "Checkers Simulator Manual"; Confidential, Technologielaan 11-0002 B-3001, Leuven, Belgium

[15] T. Widhe, J. Melander, L. Wanhammar; "Design of Efficient Radix-8 Butterfly PEs for VLSI"; Department of Electrical Engineering, Linköping University, Linköping, Sweden

[16] Weidong Li, Lars Wanhammar; "Efficient Radix-4 and Radix-8 Butterfly Elements"; Electronics Systems, Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden

Web References

[17] Nonvolatile Memory; http://searchstorage.techtarget.com/definition/nonvolatile-memory

[18] Reconfigurable Array Processor Satisfies Multi-Core Platforms; http://chipdesignmag.com/display.php?articleId=950&issueId=19

[19] Fast Fourier Transform (FFT); http://www.relisoft.com/Science/Physics/fft.html

[20] IEEE Standard for Standard SystemC Language Reference Manual; IEEE Standards Association; http://standards.ieee.org/findstds/standard/1666-2011.html

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

[21] Decimation-in-time (DIT) Radix-2 FFT; http://cnx.org/content/m12016/latest/

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ