# Architectural Synthesis of a Coarse-Grained Run-Time-Reconfigurable Accelerator for DSP Applications

Dem Fachbereich 18 der
Technischen Universität Darmstadt
zur Erlangung der Würde eines
Doktor–Ingenieurs (Dr.-Ing.)
genehmigte Dissertation

M.Sc. Eng.
**Abdulfattah Mohammad Obeid**
geboren am 13. März 1971

| | |
|---|---|
| Referent: | Prof. Dr. Dr. h. c. mult. Manfred Glesner |
| Korreferent: | Prof. Dr.-Ing. Andreas Koch |
| Tag der Einreichung: | 01.11.2005 |
| Tag der mündlichen Prüfung: | 13.02.2006 |

D17

Darmstädter Dissertationen

*Dedicated to my families from me,*

*as a grandson,*
*a son,*
*a husband*
*and a father...*

# Acknowledgments

All praise be to ALLAH the almighty creator, the most merciful, the supreme in knowledge, wisdom and power, to Whom my whole life is dedicated. Him I prase for His guidance and His care that brought me all the way through my life and I praise Him too for He has blessed me throughout my work for this Thesis. To Him I extend my prayers to forgive me for all my sins and shortcomings – that I all admit–, to grant me blessings, thankfulness and to shape me the way He likes a humble slave and a worshiper to be. I would like to thank both My parents for all their care, prayers and support. Their sincere prayers warmth I have felt for all my life. I also thank them for their care, advice and dedication that they have raised me with. Their guidance and advice were always very helpful and appreciable. I pray for them to be blessed and for my self to make them proud.

I would also like to thank my advisor Prof. Dr. Dr. h.c. Mult. Manfred Glesner for his experienced advising during my stay in Germany and work in his institute: the MES. As I worked in his institute, I have benefitted from him not only scientifically, but also, I have learned a lot from him on the personal level. The friendly working atmosphere in MES that all of us colleagues treasure is a reflection of his rare qualities, personality and care. Thanks also to Prof. Dr.-Ing. Andreas Koch for his fruitful discussion, and to Prof. Dr.-Ing. Jürgen Stenzel, Prof. Dr.-Ing. Peter Mutschler and Prof. Dr.-Ing. Alex Gerschman for their participation in my oral examination. Many thanks to all my colleagues in MES. In specific I thank Dr.-Ing. Alberto García Ortiz for the fruitful discussion in digital design issues, Dipl.-Ing. Tudor Murgan for the many fruitful discussions, Dr. Leandro Soares Indrusiak for his useful advice throughout the writing of this thesis. Special thanks are due to Dipl.-Ing. Heiko Hinkelmann for his reversion and correction of the German summaries and also to Dipl.-Ing. Massoud Momeni and Dipl.-Ing. Oliver Soffke. Thanks to Dr. Gilles Sassatelli for the many discussions on the area of RC.

Surely, many thanks are due to CERI of KACST that have financed my studies abroad. In particular I like to thank Prof. Dr. Fayez Alhargan, Dr. Rumaih Alrumaih and Dr. Mohammad Alkanhal for their support. Many thanks also to Mr. Saud Albattal and Mr. Ibrahim Almuneef for their continuous help. Thanks are due as well to the Saudi Arabian Cultural Bureau in Germany for their constant help and support and especially to Dr. Ahmed Ashi and Dr. Othman Omar. Thanks also to my friends: Eng. Wael Eddali, Dipl.-Ing. Omar Ben Maajouz, Mr. Mohammad El Haddad and Dipl.-Ing. Naseem Mujahed for helping me in proof reading the thesis and in the German language translation of the abstract and conclusions.

Last but by no means least, a whole lot of thanks to my wife who took a lot of hardships and was giving always her very best for me to successfully finish my studies. Many thanks also to my children: Mohammad Yousuf, Nuha and Doha for their patients for having a such busy father. I pray for them to be blessed and successful. Wa Alhamdu li ALLAH Rabbi Alalameen...

# Kurzfassung

Die Vorteile und Möglichkeiten Rekonfigurierbaren Rechnens haben zu einer großen Anzahl an Forschungsarbeiten in diesem Bereich geführt. Sowohl die Kosten der Rekonfiguration als auch spezifische Herausforderungen im Bereich des Rekonfigurierbaren Rechnens waren die Hauptgründe dafür, dass bisher keine optimalen Lösungen gefunden wurden.

Aufgrund der Flexibilität rekonfigurierbaren Rechnens gibt es eine große Anzahl an neuen Entwurfsparametern, wie z.B. dynamische Rekonfiguration, partielle Rekonfiguration, Kontextmanagement und HW/SW Probleme.

Abhängig von den Zielanwendungen und ihren Nebenbedingungen können verschiedene Entwurfsentscheidungen getroffen werden, um die rekonfigurierbare Lösung zu optimieren.

In dieser Dissertation wird *HPad*, eine effiziente grobkörnige dynamisch rekonfigurierbare Lösung für DSP Anwedungen, präsentiert. Die HPad Architektur wurde von veröffentlichten VLSI Architekturen für vielfältige DSP Anwendungen maßgeblich beeinflusst.

Basierend auf den Charakteristiken dieser DSP Algorithmen und ihren entsprechenden Architekturen wurde das HPad als heterogene und grobkörnige dynamisch rekonfigurierbare Lösung gewählt. Das HPad besitzt sowohl partielle als auch dynamische Rekonfigurationsfähigkeiten. Zudem wurde die Datenpfadarchitektur des HPads auf eine Art und Weise entworfen, die eine effiziente Realisierung der untersuchten DSP Anwendungen ermöglicht. Durch den Gebrauch lokaler Rekonfigurations-Schnittstellen wird das Problem der dynamischen Rekonfiguration partitioniert und effizient gelöst.

Das HPad wurde mit VHDL Code auf RTL Ebene modelliert und synthetisiert. Die Parametrisierung des Codes ist vorteilhaft, da sie eine einfache und schnelle Generierung neuer Entwürfe durch Anpassung der entsprechenden Konstanten und Rekompilierung gestattet. Das Modell besteht aus mehreren tausend Kodezeilen. Die Abbildung und die Verdrahtung verschiedener Pipeline- Architekturen von DSP Algorithmen wurde untersucht, um die Eignung und Validität des HPads für den beabsichtigten Anwendungsbereich zu demonstrieren.

# Abstract

Given all its merits and potential, Reconfigurable Computing has attracted lots of research work. Reconfiguration costs as well as new Reconfigurable Computing specific challenges have so far been the main obstacles hindering reaching optimal reconfigurable computing solutions.

Because of the flexibility offered by Reconfigurable Computing many new design parameters that were previously unknown now exist. Dynamic reconfiguration, partial reconfiguration, context management and HW/SW issues are among these.

Depending on the target set of applications, different design decisions can be made in order to optimize the reconfigurable solution according to the target application constraints.

In this thesis the *HPad*, an efficient coarse-grained dynamically reconfigurable solution targeted for DSP computation, is proposed. The HPad architecture was greatly influenced by reported VLSI architectures of a variety of DSP algorithms.

Based on observations of the characteristics of these DSP algorithms and their architectures the HPad was chosen to be a heterogeneous and dynamically reconfigurable coarse grained solution. The HPad features partial, dynamic, and background reconfiguration capabilities. In addition, the HPad data path architecture is tailored to efficiently realize the studied DSP applications. Through the use of local reconfiguration interface sockets around each processing element, the dynamic reconfiguration problem is partitioned and efficiently solved.

The HPad was modeled and synthesized with a parameterizable VHDL code written at the RTL level. Parameterizing the code was beneficial since it permitted generation of new designs simply by changing a few constants and recompiling. The model consisted of several thousand lines of code. Mapping and routing of several pipelined architectures of DSP algorithms were examined to demonstrate the suitability and validity of the HPad to the proposed scope of applications.

# Table of Contents

# List of Tables

# List of Figures

# Glossary

| | |
|---|---|
| $\mu p$ | *Micro P*rocessor |
| ALU | *A*rithmetic *L*ogic *U*nit |
| ACS | *A*dd *C*ompare *S*elect |
| ASIC | *A*pplication *S*pecific *I*ntegrated *C*ircuit |
| BMU | *B*ranch *M*etric *U*nit |
| CG | *C*oarse *G*rain |
| CGRC | *C*oarse *G*rain *RC* |
| CS | *C*urrent *S*tate |
| DCT | *D*iscrete *C*osine *T*ransform |
| DF FIR | *D*irect *F*orm FIR |
| DPA | *D*ata-*P*ath *A*rray |
| DIF FFT | *D*ecimation *I*n *F*requency FFT |
| DIT FFT | *D*ecimation *I*n *T*ime FFT |
| DSB | *D*ata *S*hare *B*us |
| DSP | *D*igital *S*ignal *P*rocessing |
| DTFT | *D*iscrete *T*ime *F*ourier *T*ransform |
| EPROM | *E*lectrically *P*rogrammable *ROM* |
| EEPROM | *E*lectrically *E*rasable *P*rogrammable *ROM* |
| FFT | *F*ast *F*ourier *T*ransform |
| FG | *F*ine *G*rain |
| FGRC | *F*ine *G*rain *RC* |
| FIFO | *F*irst *I*n *F*irst *O*ut |
| FIR | *F*inite *I*mpulse *R*esponse |
| FPGA | *F*ield *P*rogrammable *G*ate *A*rray |

| | |
|---|---|
| GALPE | *Gross ALU PE* |
| **HPad** | *H*eterogeneous *Pad* |
| HW | *H*ard*W*are |
| LBL | *L*ook *B*ack *L*evels |
| LUT | *L*ook *U*p *T*able |
| LSB | *L*east *S*ignificant *B*it |
| MAC | *M*ultiply-*AC*cumulate |
| MeMPE | *Me*mory *M*anipulation *PE* |
| MPGA | *M*ask *P*rogrammable *G*ate *A*rray |
| MSB | *M*ost *S*ignificant *B*it |
| NRE Costs | *N*on *R*ecurring *E*ngineering Costs |
| OFDM | *O*rthogonal *F*requency *D*ivision *M*ultiplexing |
| PE | *P*rocessing *E*lement |
| PS | *P*revious *S*tate |
| R2$^2$SDF | *R*adix $2^2$ *S*ingle *D*elay *F*eedback |
| R2$^2$MDC | *R*adix $2^2$ *M*ulti-Path *D*elay Commutator |
| R2MDC | *R*adix *2 M*ulti-Path *D*elay Commutator |
| R2MDF | *R*adix *2 M*ulti *D*elay *F*eedback |
| R2SDF | *R*adix *2 S*ingle *D*elay *F*eedback |
| RAM | *R*andom *A*ccess *M*emory |
| RC | *R*econfigurable *C*omputing |
| RE | *R*gister *E*xchange |
| RIS | *R*econfiguration *I*nterface *S*ocket |
| ROM | *R*ead *O*nly *M*emory |
| RTL | *R*igester *T*ransfer *L*evel |
| RTR | *R*un *T*ime *R*econfigurable |
| SFG | *S*ignal *F*low *G*raph |
| SoC | *S*ystem *o*n a *C*hip |
| SRAM | *S*tatic *RAM* |
| SW | *S*oft*W*are |
| TB | *T*race *B*ack |
| TDF FIR | *T*ransposed *D*irect *F*orm FIR |
| TW | *T*rellis *W*indow |

| | |
|---|---|
| VHSIC | *V*ery *H*igh *S*peed *I*ntegrated *C*ircuits |
| VHDL | *V*HSIC *H*ardware *D*escription *L*anguage |
| VLIW | *V*ery *L*ong *I*nstruction *W*ord |
| VLSI | *V*ery *L*arge *S*cale *I*negration |

# Chapter 1

# Introduction

## Contents



Nowadays, the consumer electronics market is growing with a variety of new products emerging every day. This constant increase of electronic products is maintained by the rapid advancements in process technology. New applications that were previously out of reach are now offered to consumers for affordable prices. In the realm of such a dynamic market many challenges are now rising. The increase of development and fabrication costs, higher demands of speed and computational power, the need for various standards support and lower power consumption considerations especially for mobile systems necessitate finding new flexible yet efficient and practical solutions that facilitate the realization of digital systems capable of meeting todays and tomorrows market demands.

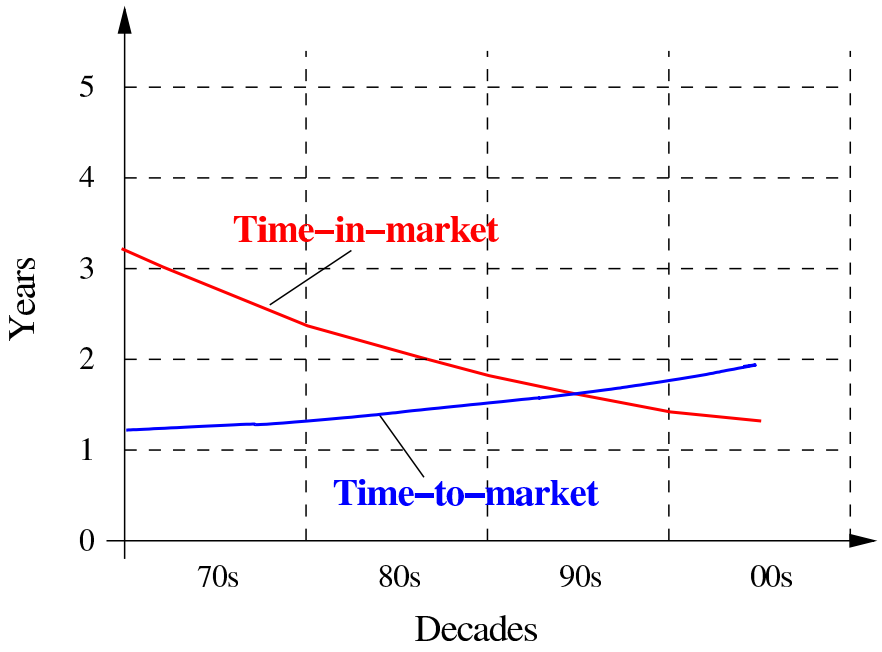


Figure 1.1: Expected time in market and time to market [55]

## 1.1 Current Challenges

The continuous growth of digital devices complexities, the constant increase in their clock frequencies and thus their power consumption and the highly competitive market bring about many design challenges. In the following we touch on the most important of them.

### 1.1.1 Time-to-Market

As fresh and advanced products are injected in the market every day the expected life time of many electronic devices is shortened and they turn obsolete at an increasing rate. Meanwhile, developing newer and more sophisticated products require considerably more design efforts as well as increasing verification and testing times. As a result, the expected *time-in-market* is steadily decreased as the the needed *time-to-market* increases as shown in Fig. 1.1 [55]. Such disturbing characteristics makes it difficult to introduce new successful products to the fiercely competing electronics market and emphasizes the importance of time-to-market reduction to reduce the dominance of Non Recurring Engineering (*NRE*) Costs. As shown in Fig. 1.1 the Time-to-market grew less than the expected Time-in-market in the middle of the last decade and the development period now approaches two years.

### 1.1.2 Computational Solutions Paradigms

With the constant demand on faster digital mobile systems the power consumption issue gains more importance especially when more functionality is sought and thus more processing power and extra peripherals are added on board. Different computational paradigms result in different power consumption per computational task characteristics. *A*pplication *S*pecific *I*ntegrated *C*ircuits (*ASICs*) for example may consume more power per cycle as compered to the functional units in a normal *M*icro *P*rocessor ($\mu p$) but while doing much more operations per cycle and thus finishing the assigned computations in considerably less amount of time. Consequently, only the measure of power consumption in *Watts* will not suffice in reflecting the total amount of energy typically needed under the considered computational paradigm for a given task. Rather, computational efficiency in *M*ega *O*perations *P*er *S*econd *P*er *W*atts (*MOPS per Watts*) can be used to give a better understanding of the power consumption needs.

The $\mu p$'s consumes more power per task because of the auxiliary operations carried out to execute a single instruction and also because of the additional power consumed by the $\mu p$ control resources. In the $\mu p$ domain a task is supplied to the $\mu p$ in the form of a sequence of instructions. A $\mu p$ goes through instruction-fetch, instruction-decode, instruction-execute and write-back cycles for each instruction to be computed. The above and other tasks (other than the instruction execute) obviously result in considerable overhead time and power consumption overhead costs. These overhead costs are the price of the high flexibility and versatility that the $\mu p$ computational paradigm possesses.

On the other hand, a typical ASIC designed to efficiently solve a specific problem executes many operations per cycle with minimal control and overhead costs. This results in ASICs computational efficiencies two to three orders of magnitude higher than those of $\mu p$ as illustrated in Fig. 1.2 [55]. Moreover, the speed of operation of ASICs is clearly far more superior than that of $\mu p$s, but unfortunately, at the expense of very low computational flexibility.

### 1.1.3 Integration Density Gap

In addition, because of the big amount of control circuitry (which result in irregular layouts) the hardware density gap between regular structures such as memories (that feature regular layouts) and those of $\mu p$s is growing wider and wider as the process technology is advancing over the years. Fig. 1.3 [7] shows the integration densities of memories and $\mu p$s. This integration density gap, as shown in Figure 1.3, is already several orders of magnitude and is still expected to grow.

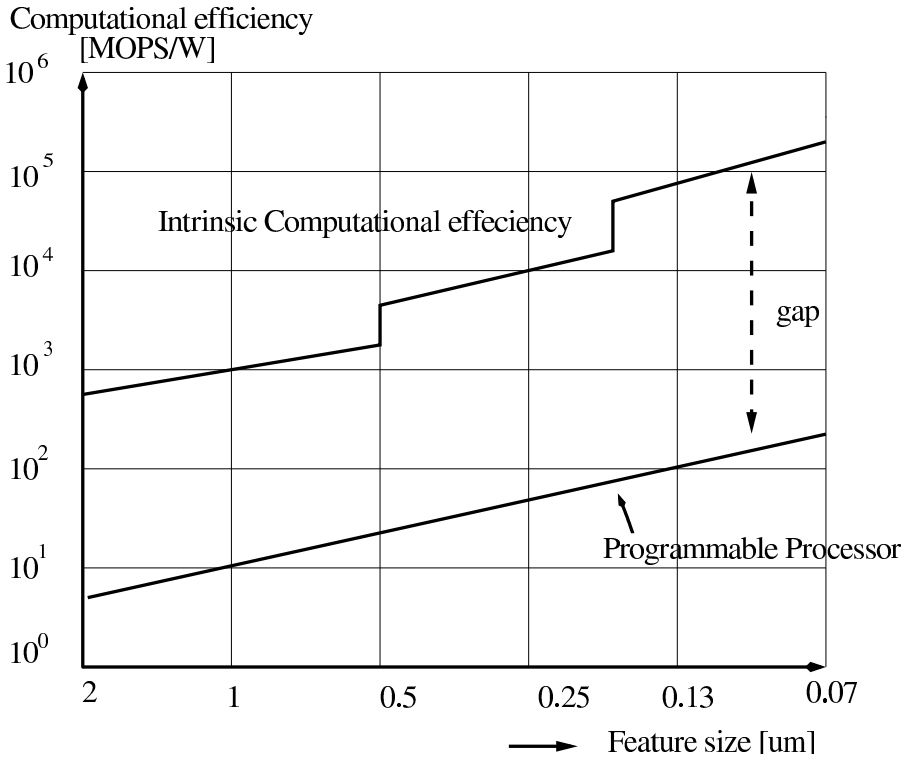


Figure 1.2: Computational efficiency gap between Silicon and programmable Processors [55]

### 1.1.4 Computational Flexibility and Cost Tradeoff

Between the two above computational paradigm extremes with the $\mu p$ trading computational efficiency for flexibility and ASICs trading computational efficiency and performance for flexibility there is clearly a design space gap that can be occupied by another computational paradigm: *R*econfigurable *C*omputing (*RC*) . By dedicating more general purpose hardware resources that can be interconnected and configured to realize different functions RC solutions aim at finding an optimal balance between pure software and pure hardware approaches by developing hardware structures that can be reconfigured to carryout different tasks at different points of time.

RC solutions can be classified according to their granularity as fine and coarse grain architectures. By having bit-level operations resources as the basic building tiles, *F*ine *G*rain *RC* (*FGRC*) solutions enjoy greater flexibility and can realize a wide spectrum of applications ranging from control-dominant to complicated Digital Signal Processing (*DSP*) applications. This flexibility comes naturally at the price of additional hardware costs which are directly reflected on performance and power consumption. Today many successful *F*ield *P*rogrammable *G*ate *A*rrays *FPGA*s which are good examples of FGRC solutions are available in the market.

FPGAs that have more regular structures have better integration densities than $\mu p$s but

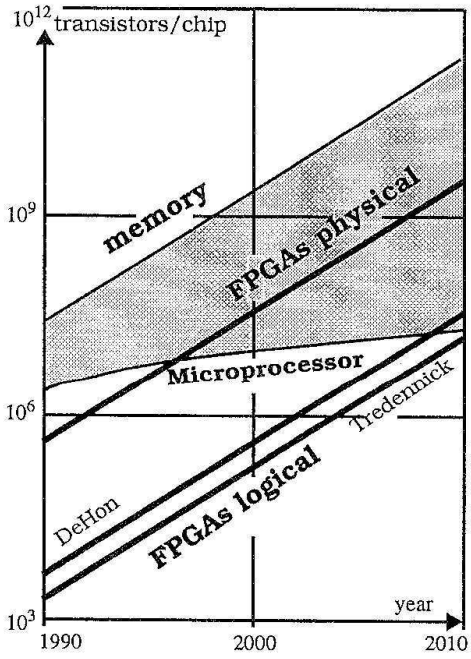


Figure 1.3: The integration density gap between memories and several computational solutions [7]

are still lower than those of memories because of the high fine grain reconfigurability overhead. Fig. 1.3 [7] depicts the integration densities of memories, $\mu p$s, and FPGAs that lie in the space between the two former.

Although very flexible, FPGAs' fine granularity makes them slower than ASICs and therefore not suitable for all applications that require high speeds of operation. Moreover, the reconfiguration costs (in terms of power, area and time) become higher because of the time required for reconfiguration and the size of reconfiguration bit stream. This makes dynamic switching between tasks –which is an important advantage of reconfigurable solutions– impractical. It was reported in [7], [93], [92] and [17] that only $0.5\% - 1\%$ is used for computational logic (See Fig. 1.4 [17]). In [17] it states that about $90\%$ of a typical FPGA area is consumed by interconnect resources which leaves only $10\%$ of the area to reconfigurable logic. Out of the configurable logic roughly $10\%$ is used as active logic yielding to only some $1\%$ utilization of the total area although $10$ times better performance as the normal $\mu p$.

This notable disadvantages in area inefficiency, power, area and time costs needed for reconfiguration dictates finding more efficient reconfigurable solutions that exhibit lower reconfiguration costs along with better area efficiency. In this context *C*oarse *G*rained *R*econfigurable *C*omputing (*CGRC*) solutions emerged as intermediate solutions optimized for vector computations and consisting of vector rather than bit processing based processing building blocks hence saving important costs. Moreover,

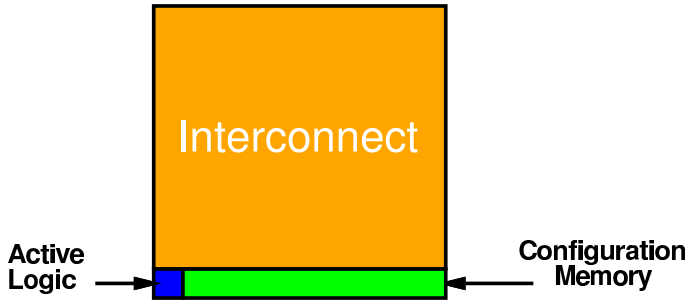


Figure 1.4: FPGA reconfiguration hardware overhead[17]

since CGRC solutions are more dense and require a smaller amount of reconfiguration resources, they are expected to have better integration densities than FPGAs and thus should lie in the area between FPGAs and memory densities in Fig. 1.3 [7].

## 1.2 Configurable Computing Potential

From the above discussion we can see that there are clear opportunities in the design space between ASICs and $\mu p$s and a lot of on chip real estate value that can be added. These clear opportunities can be exploited by RC solutions.

In [61] Tsugio Makimoto makes insight analysis based on his careful observation of the evolution of the semiconductor industry and its applications. As shown in Fig. 1.5 [61] Makimoto detected computational paradigms transitions based on the technology available in the market and how they interact with applications demands. Both the types of applications and the computational trends change as each is developed over time.

In the late 1950 discrete components were the mainstream trend of the industry. Using these discrete components customers built many devices which lead to the demand of newer solutions that can realize more advanced products. As the process technology improved, in the late 1960s, new specialized products that could efficiently realize customized products were produced and led the trend for about a decade. As explained in [7] this is the first wave that is characterized by fixed algorithms and resources. Then the $\mu p$ that enjoys flexibility and ability to realize a wide variety of applications customized by programming led the trend. However, because of its performance limitations ASICs that assist the computations of the $\mu p$ were produced to accelerate the system speed of operation. This wave is characterized by variable algorithm (software) and fixed resources ($\mu p$ & ASICs). RC is thus the last remaining wave that is expected

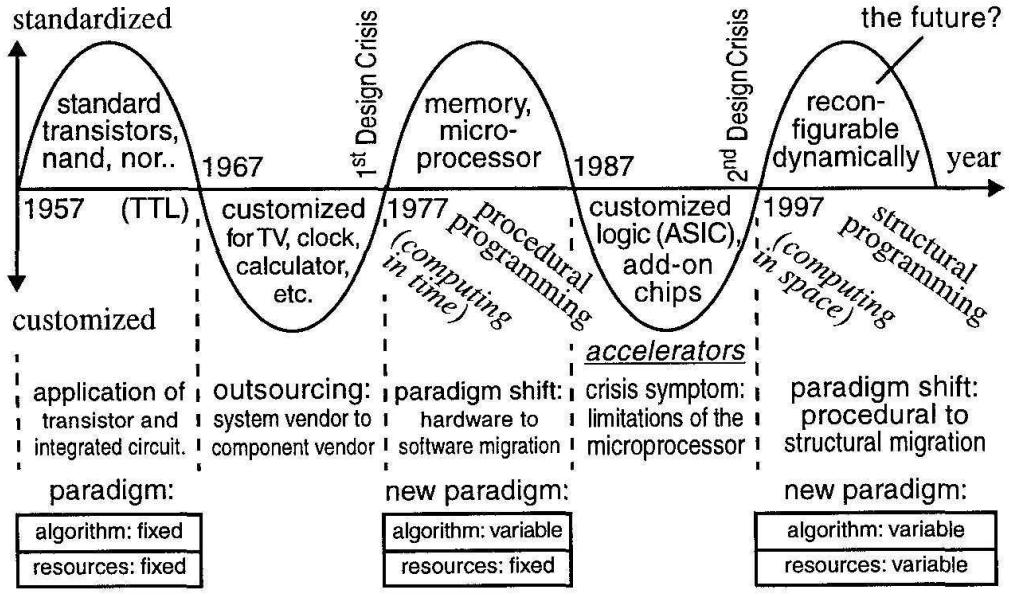


Figure 1.5: History of dominant computational paradigms shifts [61, 7]

to dominate the next computational paradigm where both the algorithm and resources are variable as advocated in [27].

Moreover, as process technology improves, the gate sizes approach physical limits and so developing more advanced RC architectures may be cheaper than designing classical solutions on smaller feature size technology.

## 1.3 Problem Statement

Clearly, each computational paradigm is best suited for a given range of tasks. Irregular tasks and those with complicated control flow are best solved by $\mu p$s when the speed of operation is not critical. When very efficient realizations in terms of both performance and power consumption are intended, ASICs are the most appropriate especially when flexibility is not an important issue. For tasks that require good performance, power consumption and flexibility RC solutions present the best suited. When irregular structures and control flow dominated applications and no dynamic reconfiguration is needed FGRC solutions are convenient but when more efficiency and dynamic reconfiguration is sought CGRC solutions is well fitted. Hence, a *S*ystem *o*n a *C*hip (*SoC*) encompassing $\mu p$(s), ASIC(s), FPGA(s) and CGRC solution(s) is expected to be both powerful and versatile a system for general purpose computing.

As Our choice of RC solutions change according to the type of task, the choice of CGRC solution will also change depending on the family of tasks it is intended to solve. In

[34] it is noted that different design choices can be made according to the target set of applications.

Many of CGRC solutions proposed in the literature or produced in the market today do not efficiently solve all types of computational problems. They can not efficiently realize well studied DSP algorithms and their well crafted architectures. Although most of them have some kind of dynamic reconfiguration capabilities, the dynamic configuration style does not in most cases fit nicely with DSP algorithms' architectures.

Withal, in some common DSP applications bit manipulation operations which is not supported by most proposed CGRC solutions are required.

As our goal was to reach an efficient CGRC solution for DSP applications and based on the aforementioned observations we decided that a pragmatic methodology to go about the design task is to study several DSP algorithms and their reported implementations extracting common and essential features to be considered when designing our proposed CGRC architecture.

Our design of the proposed CGRC solution took into consideration that it should be feasible and easy to be coupled in the future with a $\mu p$ or a more complete SoC computational solution as mentioned above.

## 1.4 Thesis Contributions

In this work several DSP algorithms architectures were realized and in some cases improved. Likewise, reconfigurable and modular architectures for FIR filters, FFTs, DCTs and Viterbi decoders were studied and realized.

Based on our findings, a CGRC architecture for DSP applications was developed. The proposed CGRC solution computational paradigm is based on a library computational model in order for it to be easily integrated with a $\mu p$ in a loosely coupled fashion.

The proposed CGRC solution (the *HPad*) was described in a Register Transfer Level (*RTL*) parameterizable VHDL model. Only by changing a few parameters in the VHDL package new copies of the design could be synthesized and simulated with no extra design efforts. This is a very useful feature that enabled experimentations with different design parameters. The synthesisable HPad model and all needed peripherals consisted of several thousand lines of VHDL code.

The HPad architecture can be categorized as coarse grained although it possesses limited bit manipulation capabilities but with only minimal overhead and therefore can be also categorized as mixed-grained. Furthermore, the proposed HPad can be categorized as heterogeneous. The heterogeneity of the HPad architecture allows efficient realizations of a good range of DSP algorithms. The HPad is also multi context dynamically reconfigurable with partial reconfigurability capability.

## 1.5 Thesis Organization

The remainder of this thesis is organized as follows: the next Chapter introduces RC technology by highlighting the main features of chosen examples of both FPGA and CGRC solutions available in the market or in the literature. This is followed by a general discussion of the previewed architectures and their different features and our observation of their development trends.

Chapter 3 then discusses FIR, FFT, DCT and the Viterbi algorithms their theory, their reported implementations and some of our contributions in the course of our study of their architectures. Afterwards is a discussion of the extracted features that are needed in a CGRC solution in order for it to efficiently realize such algorithms.

In Chapter 4 the features of the required CGRC architecture are studied and discussed in the light of the studied CGRC architectures in Chapter 2 and the DSP architectural requirements found in Chapter 3. The advantages and disadvantages and the degree of suitability of these architectures to the studied DSP implementations are pointed out. Thereupon, the goals of our work are stated leading the way for the design of our proposed solution.

The architecture and description of our proposed HPad is then discussed in Chapter 5. There, the general architecture is introduced and its different basic building blocks are described as well. Then proposed dynamic and partial reconfigurability mechanisms are discussed. Reconfiguration and multi context control and external peripherals are also described.

This leads to illustration examples of several DSP algorithms mapping on the HPad in Chapter 6. Finally the thesis is concluded in Chapter 7.

# Chapter 2

# Evolution of Reconfigurable Computing

## Contents

System's versatility and flexibility features have always inspired scientists and engineers. In the area of digital computing, finding flexible platforms that can support the realization of a variety of digital systems has attracted the attention of design engineers since the early days of the digital age. It all started with the mask programmable ROM and evolved to the current state of the art multi-million gate FPGAs and complex Coarse Grained arrays capable of several billion of operations per second. In this chapter we introduce –in brief– the evolution of reconfigurable devices showing examples of their their main features and architecture and we also touch on some of the possible areas of improvements.

In the first section we discuss the the evolution of fine-grained FPGAs beginning by introducing different configuration techniques. Later in Section 2.3 we show some of the main features and architectures of some of the popular FPGA devices in the market today. Thereafter, section 4.2.2 introduces some of the main achievements in the area of Coarse-Grained Reconfigurable arrays, their structures and how they aim to solve data flow applications.

## 2.1 FPGA Evolution

Based on the fact that every logic function can be expressed as a sum of products, ROM presented a solution that can realize the truth table of a given function by having the input lines specify the addresses and the output lines delivering the stored bits corresponding to those inputs. This solution denoted PROM (Programmable Read Only Memory) laid the first stone in a new area of computing: Programmable Logic or later known as Configurable Computing. In the early days, configuring the PROM was finalized during the fabrication phase through masking. Later the EPROM Electrically Programmable ROM) and EEPROM (Electrically Erasable Programmable ROM) technologies emerged allowing users to directly customize the popular ROM devices and, more importantly, paving the way, along with other configuration technologies, for the era of Reconfigurable Computing. The remainder of this section introduces the main programming technologies and the main features of various FPGA architectures are presented.

### 2.1.1 Configuration Technologies

The proper organization of hardware resources along with the ability to change its configuration are the two main tools that make configurable computing possible. Configuration technologies started in the very beginning by putting the configuration information on the masks allowing the configuration of ROMs or s *M*asked *P*rogrammable *G*ate *A*rrays (*MPGA*) at fabrication time, reducing by that to some extent the NRE cost and time to market. The appearance of several hardware programming technologies

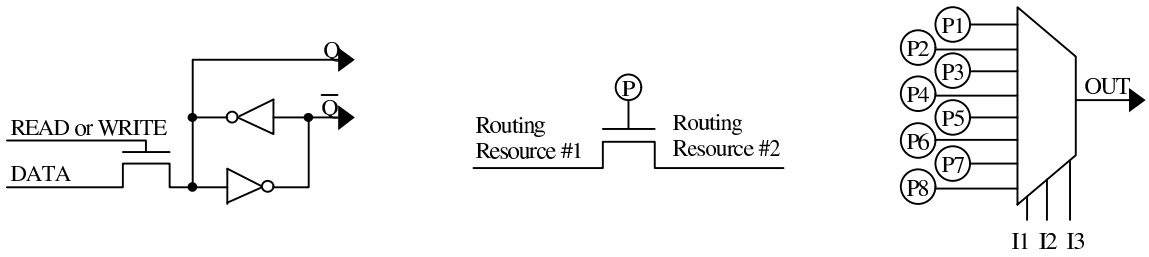


Figure 2.1: The SRAM reconfiguration technology [15]

uncovered the wide range of possibilities that could be exploited by configurable computing.

Configuration of hard ware is achieved by changing the inputs and internal connectivity of its different components. The configuring elements should have a very high OFF resistance, a low ON resistance and a low parasitic capacitance. Moreover, for efficient and feasible realizations, these reconfiguration resources should consume as small area as possible and should be easy to fabricate. There are three main configuration technologies: *S*tatic *RAM* (*SRAM*), *EPROM*, *EEPROM* and *Anti fuse*.

- *SRAM*
  The SRAM configures the internal connections by controlling pass transistors or transmission gates. An SRAM cell connected to the input of a pass transistor can switch it on or off according to the stored bit in the SRAM cell (see Figure 2.1). In practice, a combination of SRAM cells and multiplexors is used as Figure 2.1. Such a system is volatile since the SRAM cells will loose its stored data at power down which means that a system having an SRAM configurable device on board must also have some sort of permanent storage device on board to load the configuration from say an EPROM or a hard disk.

  The two main disadvantages of the SRAM configuration technique are the large area required for the SRAM cells and the the need of on board permanent storage to carry the configuration information. On the other hand, reconfigurability (which is relatively fast) makes is attractive for special applications such as testing and prototyping. Moreover, fabrication is easy since it is carried out completely in a standard CMOS process.

- *EPROM & EEPROM*
  The EPROM configuration technology is similar to those used in EPROM memories. The idea is a modified MOS transistor with an additional "floating" gate not connected to any wire (Figure 2.2) . By passing a high current through the transistor, some charges will be trapped in the floating gate causing the transistor to be permanently switched OFF. as such, the EEPROM acts now as a configuration resource. By shining ultraviolet light on the EPROM transistor the electrons

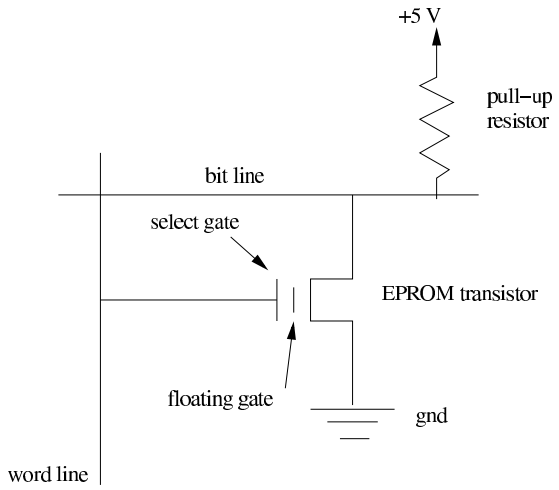


Figure 2.2: The EPROM programming technology [13]

trapped in the floating gate gain enough energy to tunnel and escape from the floating gate.

The EEPROM technology is similar to that of the EPROM except that configuration can be electrically erased of course at the expense of extra chip area. The main disadvantage of of the EPROM technology is the slow reconfiguration time which make it not suited for dynamic reconfiguration. Another disadvantage of the EPROM technique is the need of multiple voltage sources for configurations that are otherwise not required.

In both the EEPROM and EPROM techniques the configurable transistors can be used to pull up or down the inputs of logic blocks as depicted in Figure 2.2 which, in tern, results in extra power consumption. The main advantages of the EPROM and EEPROM configuration technologies is that they are reprogrammable and there is no need for extra on board memory resources since the techniques are non-volatile.

- *Anti Fuse*
  A very interesting configuration approach is the Anti Fuse technology. Here the Anti Fuse which have a very high impedance acting like an open circuit can be permanently switched to a permanent low resistance state acting as a permanent connection. There are two main Anti Fuse techniques: The PLICE introduced by XILINX and the ViaLink introduced by QuickLogic. The PLICE as shown in Figure 2.3 is composed of a top layer made of poly Silicon, a middle layer which is an Oxygen-Nitrogen-Oxygen dielectric and an n+ doped Silicon forming the bottom layer. A high voltage applied through the Anti Fuse terminals causes it to breakdown forming a permanent connection between the poly Silicon and the n+ diffusion.

  The ViaLink Anti Fuse consists of a top and bottom metal layers separated by a

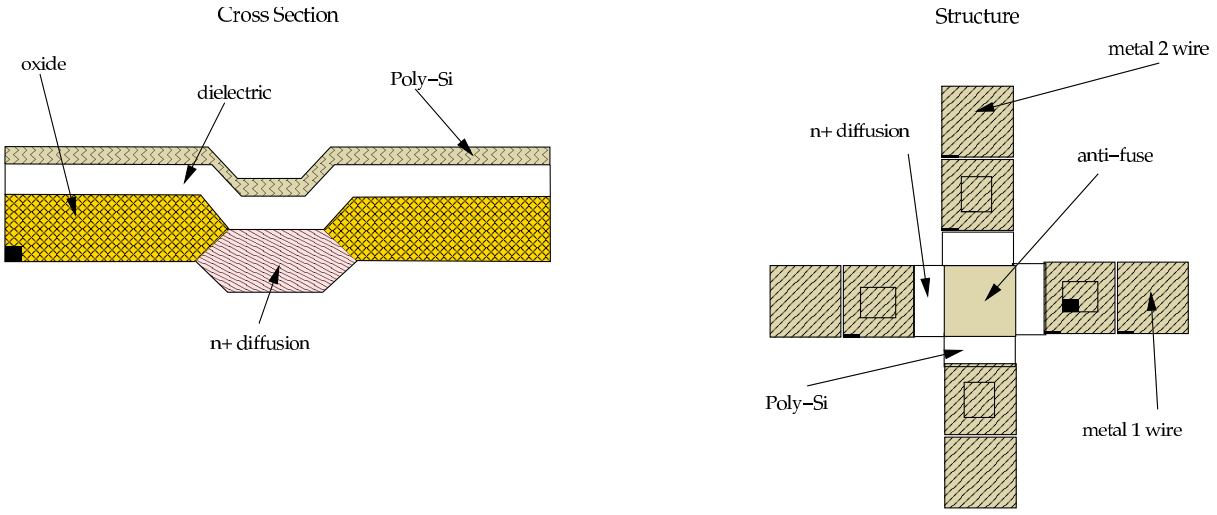


Figure 2.3: The PLICE Anti Fuse structure [13]

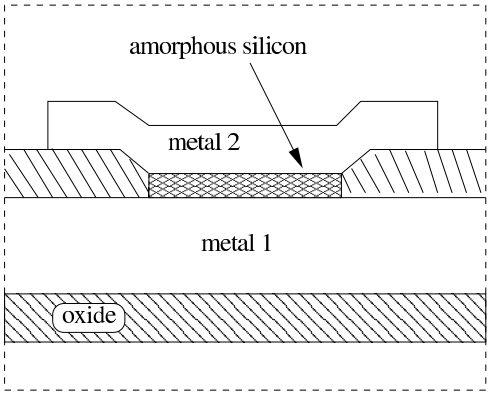


Figure 2.4: The ViaLink Anti Fuse structure [13]

poly amorphous Silicon of an unprogrammed resistance in the gigaohm range and altered to low resistance forming a permanent connection between the two metal layers after applying a high voltage across the amorphous Silicon terminals. Figure 2.4 shows the structure of the ViaLink anti fuse.

Although both Anti Fuse configuration technologies require less chip area than both the SRAM and EPROM technologies, their main disadvantages is that they require extra high voltage transistors needed to deal with the configurations high voltages and currents and also they need 3 more mask layers than the standard CMOS process.

## 2.2 Reconfigurable Architectures Classifications

A variety of reconfigurable architectures can be found today in the market or in literature. It is therefore beneficial to classify them according to various characteristics in order to have a better understanding of their different features and capabilities.

Reconfigurable architectures can be classified according to the smallest granularity, the type of reconfiguration supported and how the reconfigurable block is attached to the on board microprocessor (if any). In the following we introduce briefly these classifications.

### 2.2.1 Granularity

Reconfigurable architectures are generally composed of an array of processing elements. These processing elements can be as small as a 3-input 1-output *L*ook *U*p *T*able (*LUT*) or as big as a 16-bit *A*rithmetic *L*ogic *U*nit (*ALU*) . The term *granularity* refers to the size of the operands and output of the processing elements. In the case of the 3-input LUT the reconfigurable architecture is classified as a very *fine-grained* architecture and the 16-bit ALU can be classified as a very *coarse-grained architecture* . Fine-grain architectures can realize a very wide range of designs but they suffer from expensive routing and reconfiguration resources and thus the net area efficiency of designs realized on them is usually low. Coarse-grained architectures on the other hand can efficiently realize data flow designs but are not suitable for designs that involve control or irregular and fine-grained operations.

Most FPGAs in the market such as XILINX, Actel and Altera FPGAs are classified as fine-grained architectures. Examples of course-grained architectures include Elixent, PACT, MATRIX, KressArray and RAW where the former two are examples of products available in the market and the others are still research projects in different universities.

### 2.2.2 Heterogeneity

Most configurable devices use a single processing element as the main building block. An other approach is to use different types of processing elements instead of only one. The idea is to gain more processing power and flexibility yet maintaining a small area requirements. This way the smaller processing elements can be used distributing data processing over the array rather than having only one type big processing element that have a lot of processing power.

An example use of that is to use functional units that implement functions that are expensive or impractical to realize using other processing elements such as multipliers [101] and [15]. An other example is embedding storage resources to enable more efficient realizations for applications that need memory for storage. This may in many

cases increase the area efficiency of the reconfigurable device since forming registers from basic logic blocks or using the flip flop in the logic block my result in inefficient mapping of some special architectures.

Moreover, some FPGA vendors provide on chip microprocessors to assist the designer to partition the most control dominated parts of his design in the $\mu p$ for example in the Virtex-II pro platform. [101].

### 2.2.3 Reconfiguration Features

The configuration technology, the number of reconfigurations and the speed of reconfigurability are properties that classify reconfigurable architectures.

According to the configuration technology configurable devices can be classified into *volatile* and *non-volatile*. Non volatile device are one-time-programmable and therefore can not be referred to as reconfigurable. Although non-volatile devices do not enjoy reprogramming capabilities, they posses better area efficiency and power consumption characteristics. Examples of non-volatile devices are produced by Actel and QuickLogic both use their own *Anti-fuse* configuration technology as was discussed in this Section 2.1.

Volatile devices can be further classified according to *Statically* and *Dynamically* reconfigurable . Statically reconfigurable devices are those that need relatively long time –usually in the order of milliseconds or more which may cause unacceptable delays when run-time-reconfiguration is required– to reconfigure. Most of reconfigurable devices in the market now are statically reconfigurable. Dynamically reconfigurable (also referred to as *R*un *T*ime *R*econfigurable (*RTR*)) devices can be reconfigured in considerably shorter amount of time. Currently all RTR devices are based on *SRAM* reconfiguration technology since *EEPROM* devices need fairly long time for reconfiguration.

RTR devices are classified according to their reconfiguration resources and method of reconfiguration. A *Single-Context* device is a device that is fully configured with a single stream of configuration bits. These configuration bits are referred to as the *Configuration-Context*. When a new design or configuration is to be assigned a new context is swapped into the reconfigurable device. Clearly the time required to swap in and out the contexts may make it difficult for such a device to operate under strict RTR conditions unless an efficient reconfiguration mechanism is implemented. To speed up the reconfiguration time several techniques such as configuration compression and configuration caching can be used [15].

*Multi-Context* devices have memory resources that can hold more than one configuration allowing the reconfigurable device to quickly switch between them. In this context the reconfigurable device can be viewed as a multiplexed set of single context reconfigurable devices.

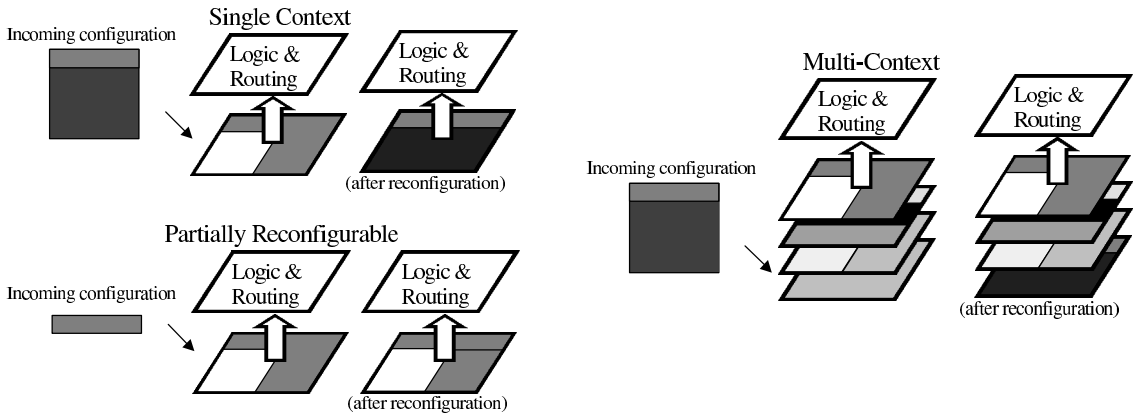


Figure 2.5: Partial reconfigurability cases [15]

RTR devices can be also classified as *Partially Reconfigurable* when portions of the reconfigurable device can be reconfigured without interfering with the operation of the rest of the device. This may be helpful in reducing or virtually eliminating the time of reconfiguration. Figure 2.5 illustrates single, multi context and partial methods of reconfiguration.

### 2.2.4 Coupling

A reconfigurable device can be used as a stand-alone unit or it may be –in many cases– coupled with a $\mu p$. Different levels of couplings are possible each with its own advantages and drawbacks. A *very tightly coupled* reconfigurable device can be integrated with the $\mu p$ as deep as a *functional unit* or a *coprocessor*. In such a case an advantage is that the normal programming scheme can be still used with the addition of extra special instructions for the reconfigurable functional unit. A drawback may be the need to modify the $\mu p$ to attach the reconfigurable functional unit. Also such a coupling scheme may not be suitable for very heavy computations that require many cycles of operation since the $\mu p$ may be waiting for the conclusion of computation. An intermediate solution between the stand-alone and the *tightly-coupled* coupling schemes is the *loosely-coupled* coprocessor scheme. The *loosely-coupled* processing unit can be coupled through the memory bus and in that case referred to as an *attached processing unit* or can be connected to the system through I/O ports as an *external processing unit*. As shown in Figure 2.6 in a coprocessor coupling scheme the $\mu p$ issues an instruction to the reconfigurable device either passing to it data or telling it where to find it and then the reconfigurable device works independently with no need of $\mu p$'s control until it finishes and writes the result.

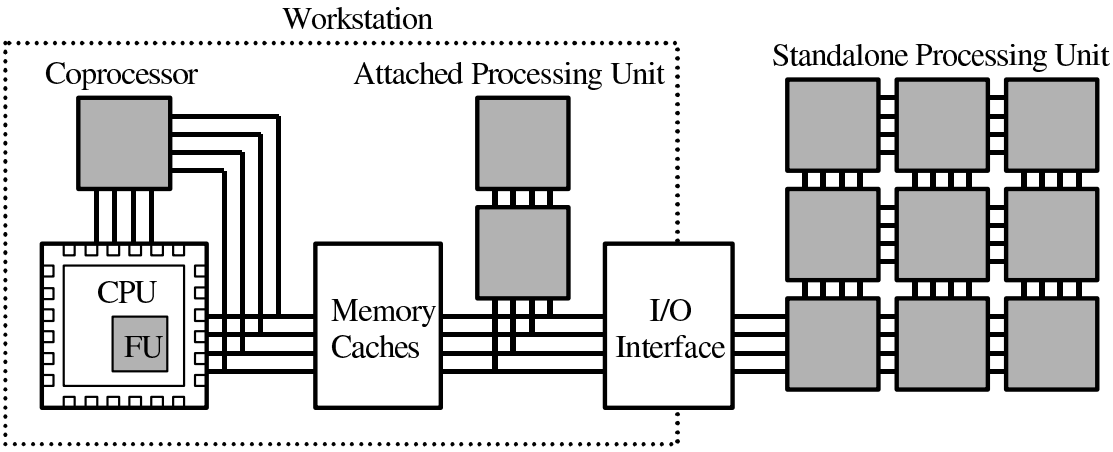


Figure 2.6: Tight, functional and loose coupling classes [15]

## 2.3 FPGA Architecture Examples

FPGA manufacturers have adopted different architecture techniques for their FPGA aiming at higher efficiencies and suitability for various designs. In the following the architectures of three main FPGA manufacturers representing different architecture and technology strategies are presented.

### 2.3.1 XILINX FPGAs

In 1985 XILINX [101] produced the first FPGA. Based on a symmetrical array architecture as simplified in Figure 2.7, and SRAM configuration technology, XILINX FPGAs was well received by consumers.

- XC2000 Series
  The XILINX XC2000 FPGA [13] series was the first produced by XILINX. The XC2000 – now extinct – was composed of several *C*onfigurable *L*ogic *B*locks (*CLBs*) interconnected via an interconnection matrix to form at the end a square matrix. The CLBs were based on an *L*ook *U*p *T*able (*LUT*) and a register and a few multiplexors permitting the realization on any four input boolean function or two functions of three variables. The CLB has 2 outputs which can both be combinational or one of them can be registered. Figure 2.8 shows the architecture of the XC2000 CLB.

  The outputs of each CLB had a direct connection with 3 of its neighbors: the top, the bottom and left neighbors. For medium range connectivity, additional general purpose wiring resources spanning only one CLBs were provided. Longer

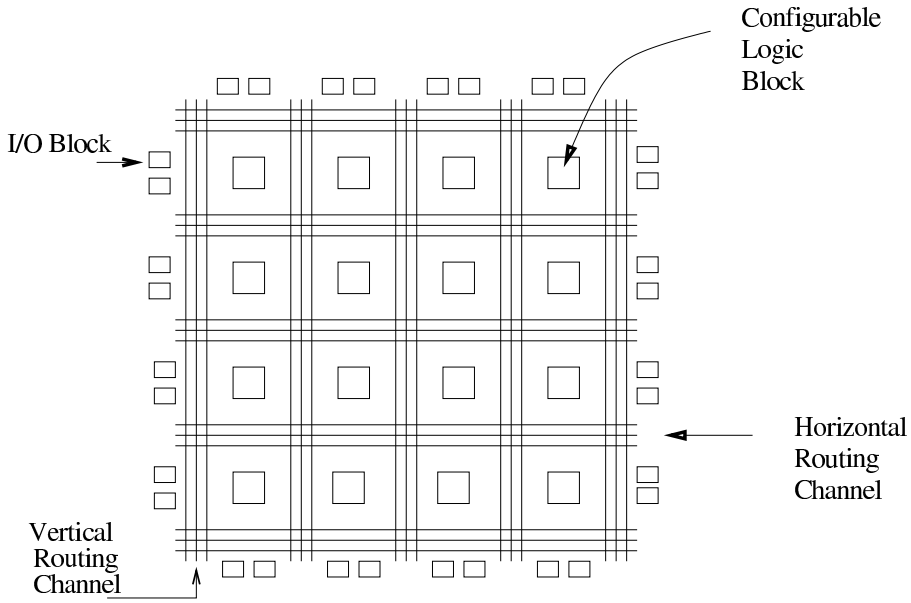


Figure 2.7: General architecture of XILINX XC2000 FPGAs [13]

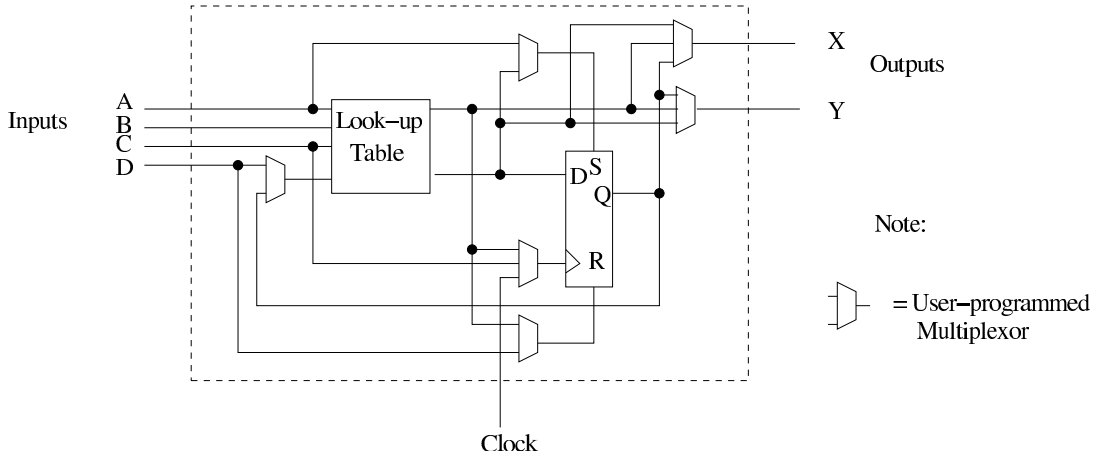


Figure 2.8: The XC2000 CLB structure [13]

connections were made possible by connecting more than one general purpose wiring resource together through a switch matrix. Long lines connections spanning the complete FPGA were also provided for very long connections. All multiplexors, interconnection resources configurations as well as the inputs of the truth tables were configured by SRAM cells that are volatile and loose their contents at power down, a feature making XILINX FPGAs perfect for prototyping and use in research and academia.

- Virtex4 Series
  Nearly a couple of decades later, Xilinx's Virtex4 series [101] was produced by

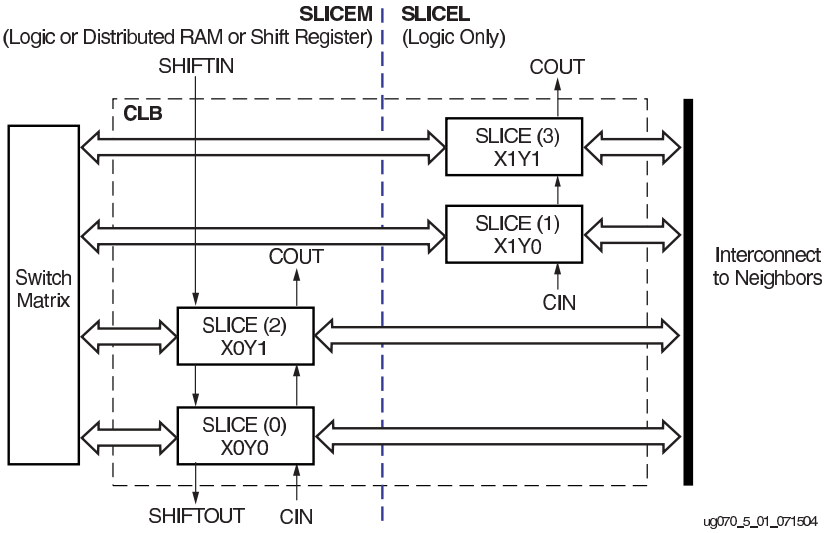


Figure 2.9: General structure of the Virtex4 CLB [101]

XILINX. The column based architecture with bigger and more complicated CLBs shows an evolved design strategy still based on LUTs. Now each CLB is composed of four slices as depicted in Figure 2.9 with support of more logic functions with two truth tables of four inputs per slice with the outputs also optionally registered. A new added feature that was not available in older FPGAs from XILINX is the Variable-Input LUT Architecture where more functions can be supported by connecting more than one slice with dedicated multiplexors supporting up to 32 input logic functions . Two types of slices are available in the CLB: the SLICEM shown in Figure 2.10 and SLICEL in Figure 2.11. Both SLICEM and SLICEL are the same with exception that SLICEM has added features shift features facilitating the implementation of distributed RAM or shift registers.

Additionally, the Virtex4 has block RAM designed to be configured in different depths and widths and can also support FIFO implementations. The XtremeDSP slices contain a dedicated $18 \times 18$ bit multiplier an adder and an accumulator along with truncation and saturation capabilities. The XtremeDSP block is designed to efficiently implement high-speed DSP applications. Some of the Virtex4 versions have even PowerPC processor and Transceiver blocks cores. Moreover, XILINX also provides soft IP (Intellectual Property) cores of some popular applications designed to efficiently suite the Virtex4 architecture. A simplified illustration of the XtremeDSP block is given in Figure 2.12.

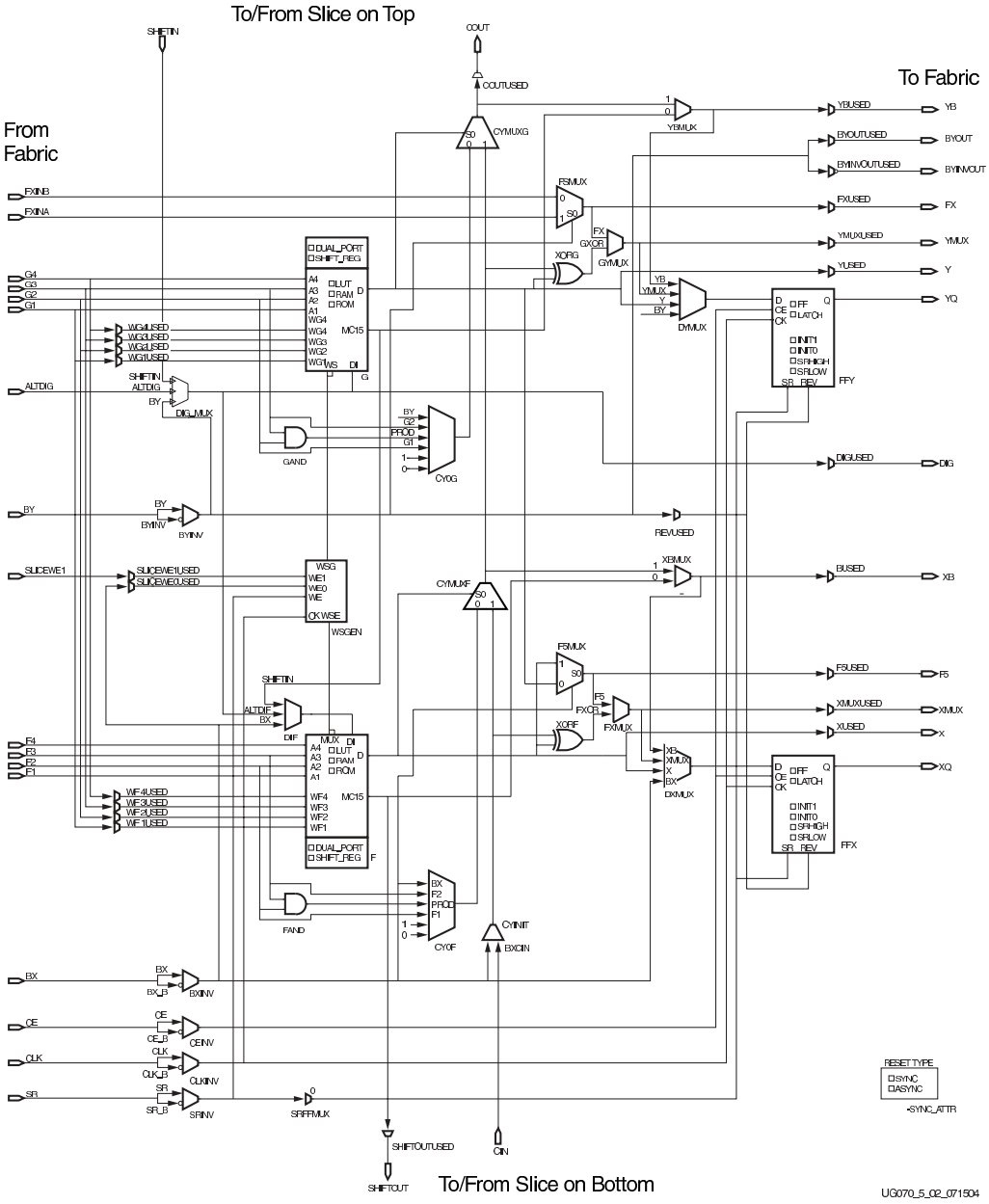


Figure 2.10: Architecture of SLICEM of the Virtex4 CLB

### 2.3.2 Actel FPGAs

Actel was one of the first companies to produce Anti Fuse FPGA. The Anti Fuse technology, although non-reconfigurable, enjoins better performance because of the better electrical characteristics of the Anti Fuse elements in contrast to the SRAM or EPROM alternatives. Anti Fuse FPGA thus represent a suitable solution furnishing reduced

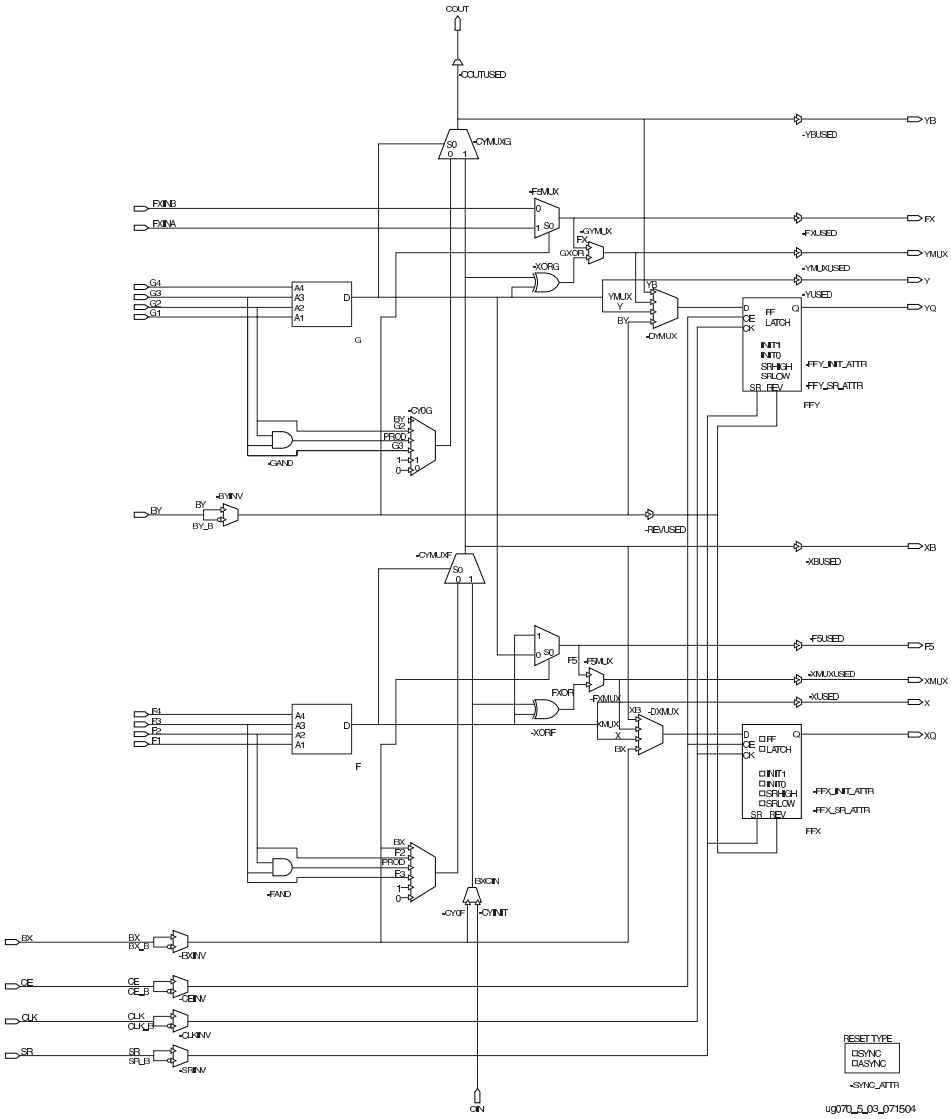


Figure 2.11: Architecture of SLICEL of the Virtex4 CLB

NRE costs, time to market and lower production costs for small volume numbers.

- Act1 Series
  Act1 series [13] – Actel's first production – was a row based architecture (shown in Figure 2.13) with relatively small and simple *L*ogic *M*odules (*LM*s) consisting as shown in Figure 2.14 of three multiplexors capable of realizing a total of 702 logic functions of two up to four input variables. Half the inputs of each LM had access to wiring resources above and the other half had access to the wiring segments below.

  The output of each LM was also connected to wiring segments both above and be-

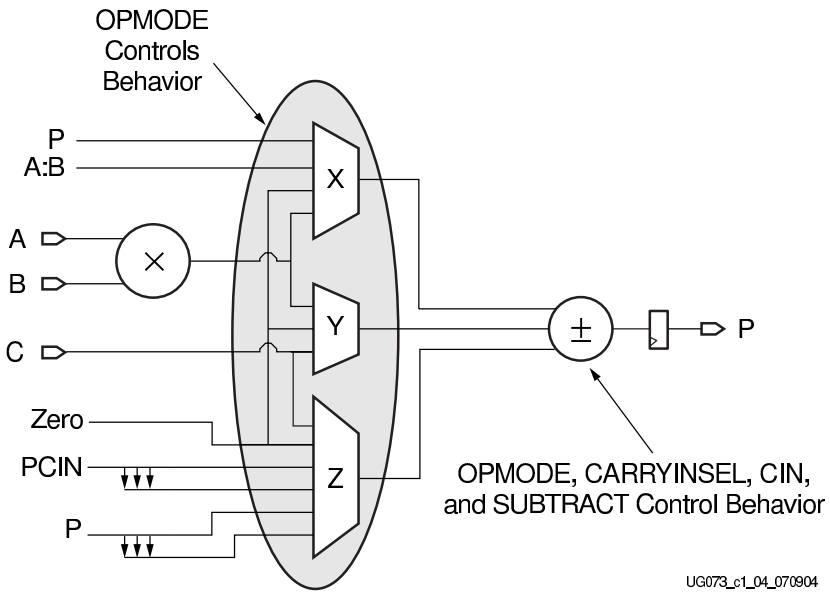


Figure 2.12: A simplified illustration for the Virtex4 XtremeDSP architecture [101]

low the LM. Wiring segments which were metal interconnects of different lengths were connected via programmable Anti Fuses and could also form longer segments by connecting more than one of them together. For inter row connectivity, a number vertical wiring tracks were also available for each column.

- Axcelerator Series
  Recent Actel FPGAs include Anti Fuse and EEPROM versions. The more advanced LM and the introduced RAM/FIFO blocks are notable. For the Axcelerator series [3], programmability is based on the metal to metal Anti Fuse technology. The Axcelerator's architecture is based on a Sea of Modules topology where the interconnection resources are fabricated on an overlaying layer on top of the logic modules. The chip (as shown in Figure 2.15) is divided into a symmetrical array of Core Tiles which are assembled from several RAM/FIFO blocks and an array of SuperClusters. A SuperCluster is composed of two Clusters that include three LM cells and several input and output buffers. There are two types of LM: the C-Cell shown in Figure 2.16 which is multiplexor-based structures in the SuperCluster permitting the implementation of over 4,000 logic functions of up to five inputs. The R-Cell as shown in Figure 2.17 is a register- based cell which can have various control configurations. Actel claims that this configuration of the SuperCluster permits efficient implementations of various logic functions and carry logic arithmetic with the ability to and register the outputs all at minimum interconnect delays.

  The RAM/FIFO can be structured into different RAM widths and lengths configurations or as asynchronous FIFOs with no use of other LMs. The FIFO

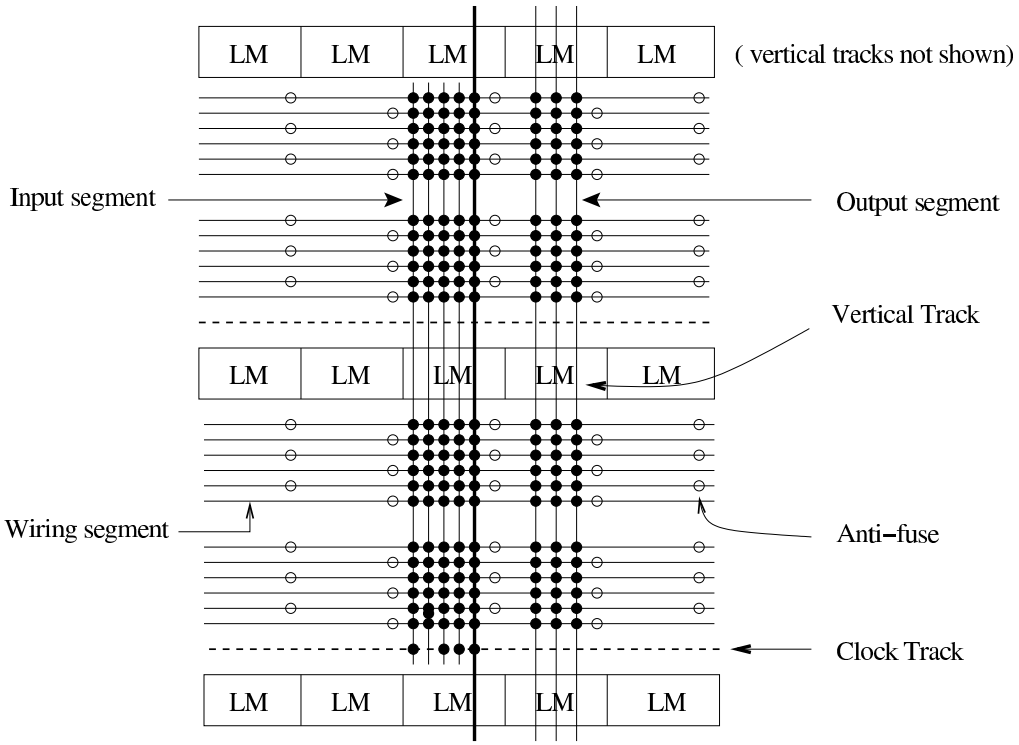


Figure 2.13: Routing architecture of the Act1 FPGA [13]

configurations allow different read and write data widths and can provide FULL/ALMOST-FULL and EMPTY/ALMOST-EMPTY flags. In addition, all control and counter resources needed for SRAM of FIFO implementations are included in the RAM/FIFO blocks.

Internal wiring resources are available in different hierarchies: DirectConnects which are relatively fast connect a C-Cell with an R-Cell. FastConnects connect different cells within a SuperCluster and also vertically to the SuperCluster below it. CarryConnects are used to rout carry logic between adjacent SuperClusters. In the Core Tile level vertical and horizontal tracks span the Core Tile and there are also Horizontal and vertical tracks spanning the entire length and width of the device.

### 2.3.3 Altera FPGAs

- EMP Series
  The EMP series [13] – Altera's first FPGAs – were constructed as hierarchal groupings of *P*rogrammable *L*ogic *D*evices (*PLD*s). The basic configurable PLD – named *L*ogic *A*rray *B*locks (*LAB*s) – were connected together through *P*rogrammable *I*nterconnect *A*rray (*PIA*s) as shown in Figure2.18. Each LAB was

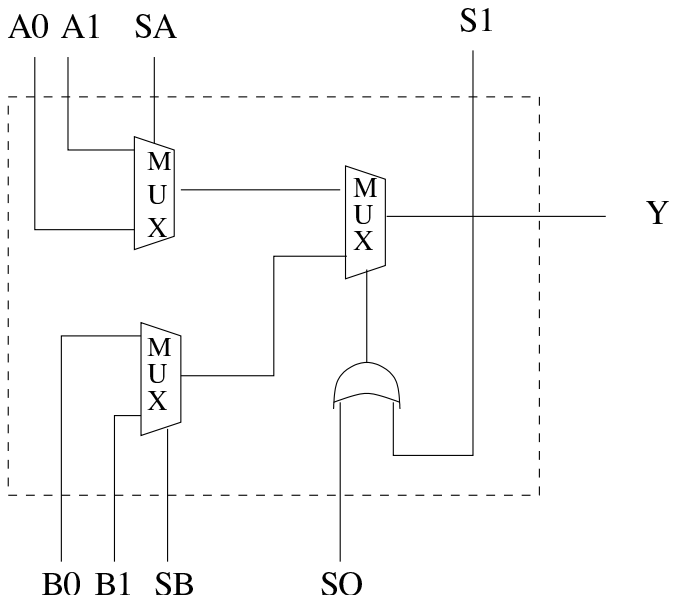


Figure 2.14: The Act1 LM [13]

made up from an array of Macrocells was interconnected with an expander product array. The Macrocell (illustrated in Figure 2.19) consisted of a flipflop and three wide input and gates whose inputs can be chosen from any signal of the PIA, any of the Macrocells in the array or any of the Expander array outputs.

As depicted in Figure 2.20 the Expander product term element is composed of a number of product terms that can be produced as in the Macrocell from any of the Expander array outputs, Macrocell array outputs or the PIA interconnects. The PIA is assembled of long wiring segments passing by every LAB thus providing full connectivity between LABs.

- Stratix II Series
  The Stratix II devices [4] are Altera's latest FPGA product. It is assembled of an array of LABs interleaved with DSP and memory blocks. Each LAB consists of eight *A*daptive *L*ogic *M*odule (*ALM*) that are have access to local interconnects for minimal local delays. ALMs as illustrated in Figure 2.21 are composed of several LUTs and a couple of adders with carry signal connections and a couple of registers allowing for optional registering of data. Several operation modes are defined for ALMs for logic and arithmetic operations. There is also a control logic block that provide configurable control signals such as clocks and resets to each ALM.

  The RAM blocks can be configured into different RAM or FIFO modes with the ability to use the closely interconnected ALMs to generate some of the needed control signals. The DSP blocks shown in Figure 2.22 can perform a variety of multiply and add operations of different vector sizes. The outputs can also be

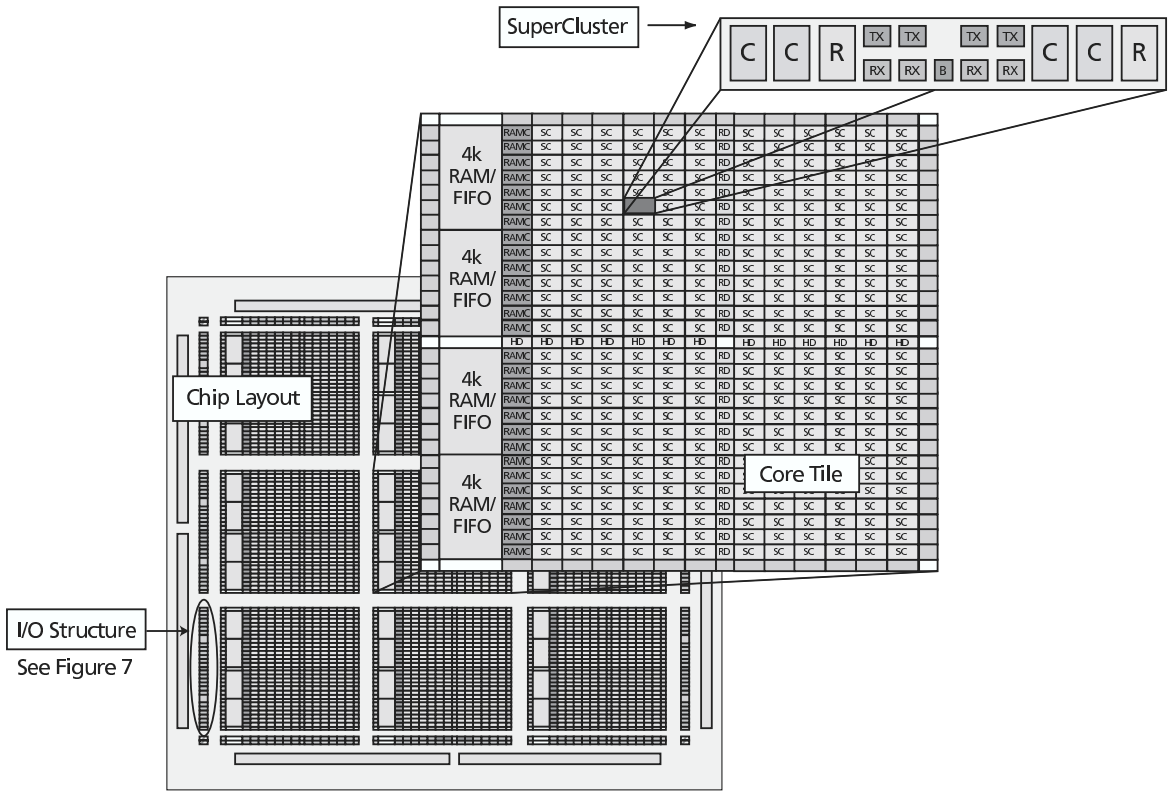


Figure 2.15: General architecture of the Axcelerator FPGA [3]

rounded or saturated as desired with the aid of several round/saturate blocks.

The interconnections network contains horizontal and vertical wiring resources of different lengths. Several lengths of interconnects connecting the elements of each LAB together, or interconnection tracks connecting labs and other neighboring blocks including other LABs, DSP or memory blocks, interconnection tracks spanning four LABs or interconnection tracks spanning the entire device are available in a flexible interconnection scheme.

## 2.4 Coarse-Grained Reconfigurable Arrays

Fine-grained FPGAs although provide great flexibility and considerable speedups when used with a $\mu p$ still suffers from a number of drawbacks when compered with CGRC solutions. These drawbacks can be lumped up in the large consumption of area, power and time needed for reconfiguration. While CGRC solutions do not exhibit the same flexibility as in FGRC solutions they enjoy the following advantages compered to their FGRC solutions counterparts:

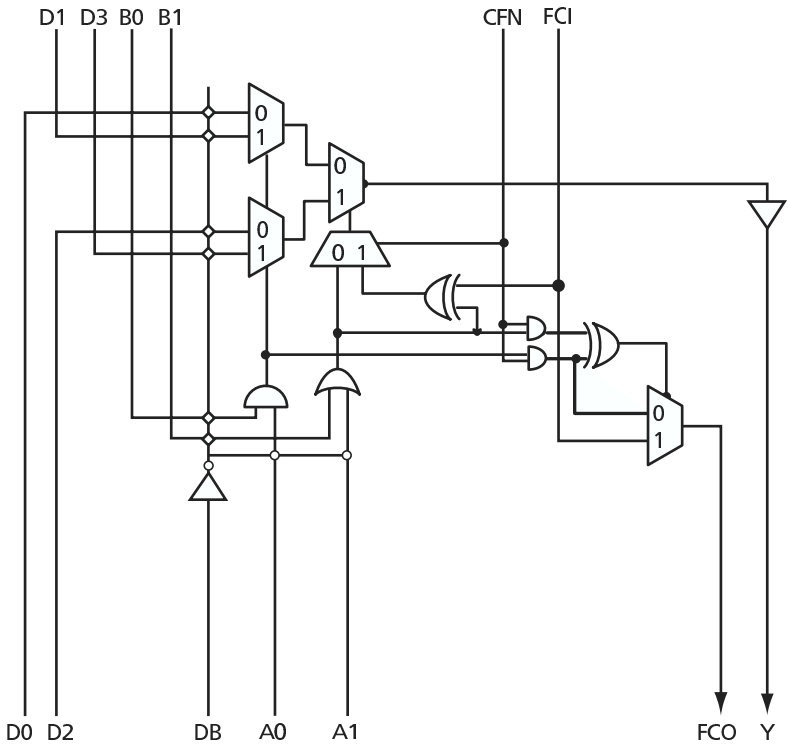


Figure 2.16: Structure of Actel's Axcelerator C-Cell [3]

- Orders of magnitude less amount of reconfiguration data is needed.
- Considerably less amount of control needed for the routing signals since vectors are routed rather than bits.
- The area efficiency is much higher than that of FGRC solutions since efficient processing elements are used as the basic building blocks.
- The architecture of CGRC solutions can be optimized for efficient implementations of data flow structures.
- Mapping and routing should be easier since coarse grained algorithms can be more easily mapped on the CGRC solution where processing elements represent mathematical operators.

Both FGRC and CGRC solutions have their usages. FGRC solutions are very effective in prototyping and low volume production of digital systems. As a matter of fact that capability of FGRC solution has been main reason for the competitive stand point that commercial FPGA manufacturers hold in the dynamic electronics market today. It has also given researchers in the industry and academia a good opportunity to design, experiment with architectures and prototype many systems a matter that must have

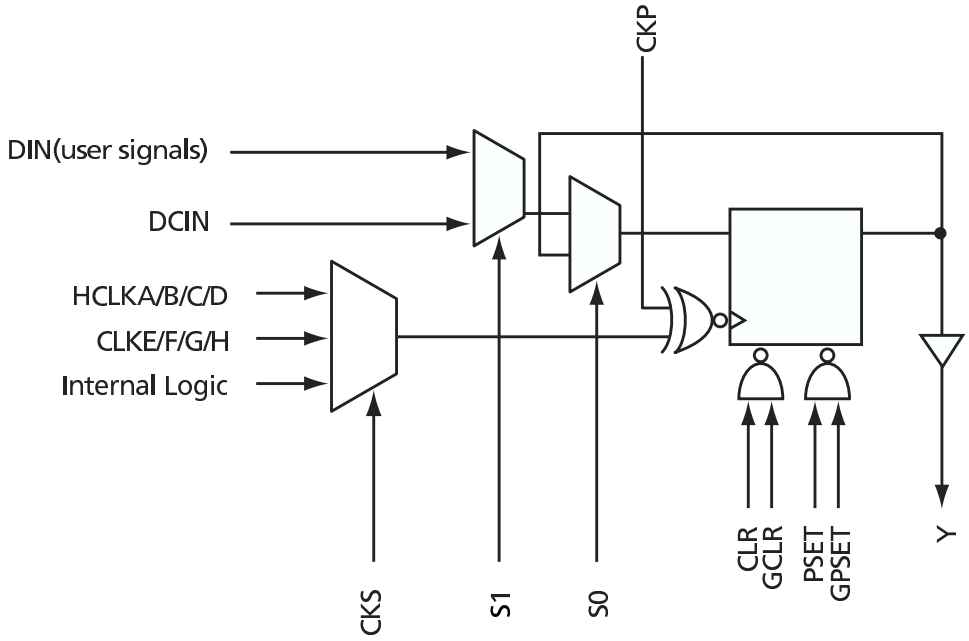


Figure 2.17: Structure of Actel's Axcelerator R-Cell [3]

had a positive impact on the quality of products we see in the market now. FGRC solutions can also be a good candidate for replacing many pieces of on-board glue logic with the more efficient FPGAs.

CGRC solutions on the other hand although not yet mature are good candidates to replace ASICs and help in the shift to the RTR paradigm by the use of dynamic reconfiguration techniques.

In the following a brief introduction to some of the CGRC solutions reported in the literature or available in the market will be introduced showing their general organization, their *P*rocessing *E*lements (*PE*) architectures, their interconnect structures and their reconfiguration characteristics.

### 2.4.1 The KressArray

- *General Organization:*
  The KressArray [36, 34, 37, 43] is a dynamically reconfigurable regular array of 32-bit *r*econfigurable *D*ata-*P*ath *U*nits (*rDPUs*). The operation of rDPUs is data driven. The KressArray went through several phases of development. Below we touch on the most important features of the KressArray-III.

- *Processing Elements Architecture:*
  The rDPU is the basic processing element of the KressArray. Each processing

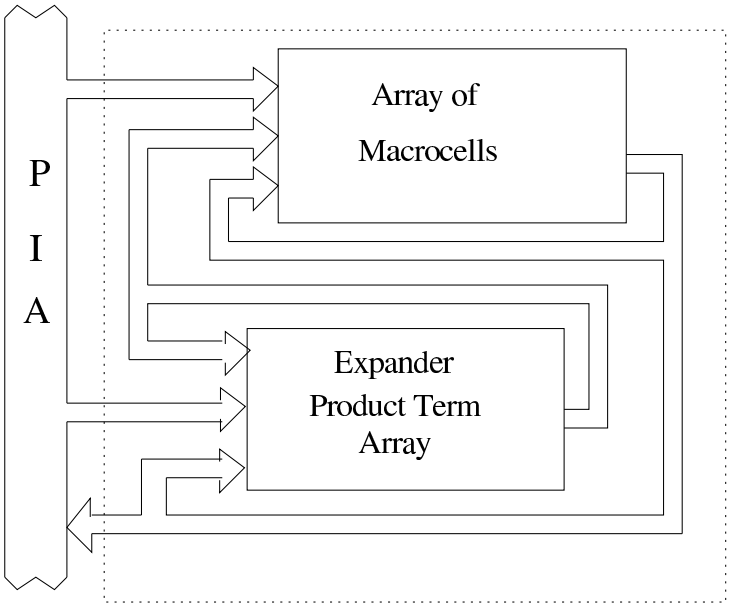


Figure 2.18: Altera's EMP LAB [13]

element is capable of all the basic integer *C* language operators support where simple operations are carried out directly and more expensive ones such as multiplication and division are carried out in a microprogrammed sequence style. rDPUs can also be used for routing data through them. An rDPU (see Figure 2.23 [34]) consists of an ALU, a register file and a number of multiplexors facilitating full connectivity. The register file can be used to store constants, intermediate results or frequently used inputs. The operations of the rDPU is data driven and is carried out independently of the rest of the array.

- *Interconnect Structure:*
  There are three levels of interconnects in the KressArray. In the bottom most level data is transferred between rDPUs to propagate and process intermediate results. Connection by abutment between rDPUs simplifies routing and is more suited to coarse grained applications. Global busses facilitate long distance connectivity between rDPUs and/or the higher level input/output busses. The input/output busses are interfaced with the internal global busses through switches. This style of hierarchical bus routing allows input data transferred to and from rDPUs not located at the edge of the array. Figure 2.24 [34] illustrates a 9-rDPU Array structure. Several sub arrays similar to the one shown in Figure 2.24 can be interconnected to form larger arrays. To reduce the number of input/output pins, serial mode connections are provided between local interconnects between sub-arrays. This serial connectivity is transparent to the programmer [37].

- *Reconfiguration:*
  The KressArray architecture is data driven allowing each rDPU to execute a given

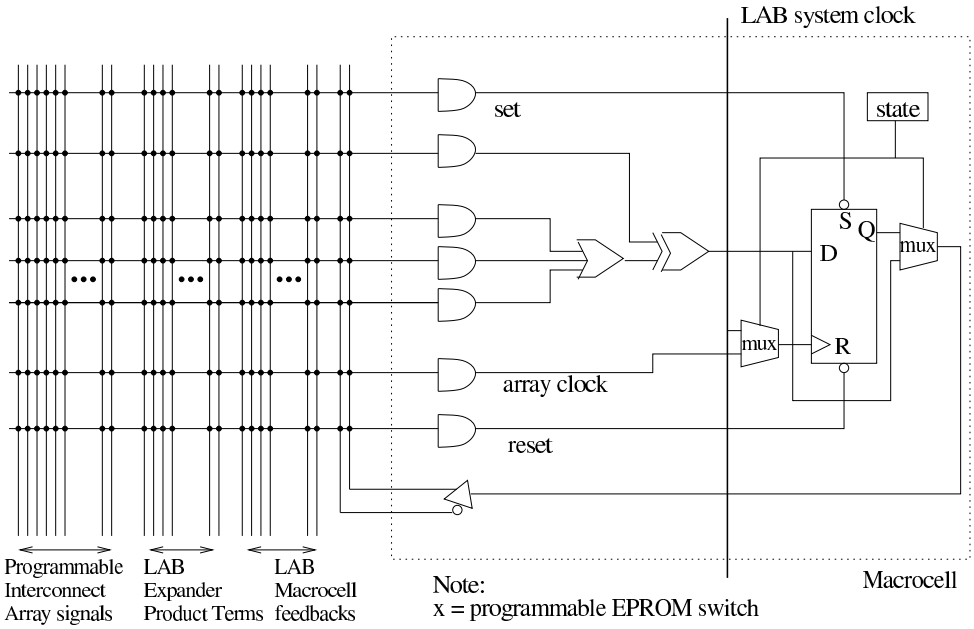


Figure 2.19: Altera's EMP Macrocell [13]

instruction on its operands as they are ready. This implies dynamic and partial reconfigurability. The KressArray is also a multi-context reconfigurable architecture. Each rDPU has a configuration memory (context) storing the operation and routing information. The configuration memory holds four layers of configurations. This implies that the register file has to be implemented in four layers as well. Switching between configurations provides very fast reconfiguration capability.

### 2.4.2 MATRIX

- *General Organization:*
  MATRIX [62, 62, 17] is an array of 8-bit Basic Functional Units (*BFU*s) interconnected via 3 levels of hierarchical interconnection resources. The MATRIX architecture allows it to operate in several modes of operation.

- *Processing Elements Architecture:*
  The BFU is the basic building block of the KressArray. As shown in Figure 2.25 the BFU consists of the following components:

  - An 8-bit ALU that is capable of several arithmetic and logic operations in one cycle and a multiply operation in two cycles. Wider word operations can be implemented by cascading several BFUs making use of the dedicated carry logic lines.

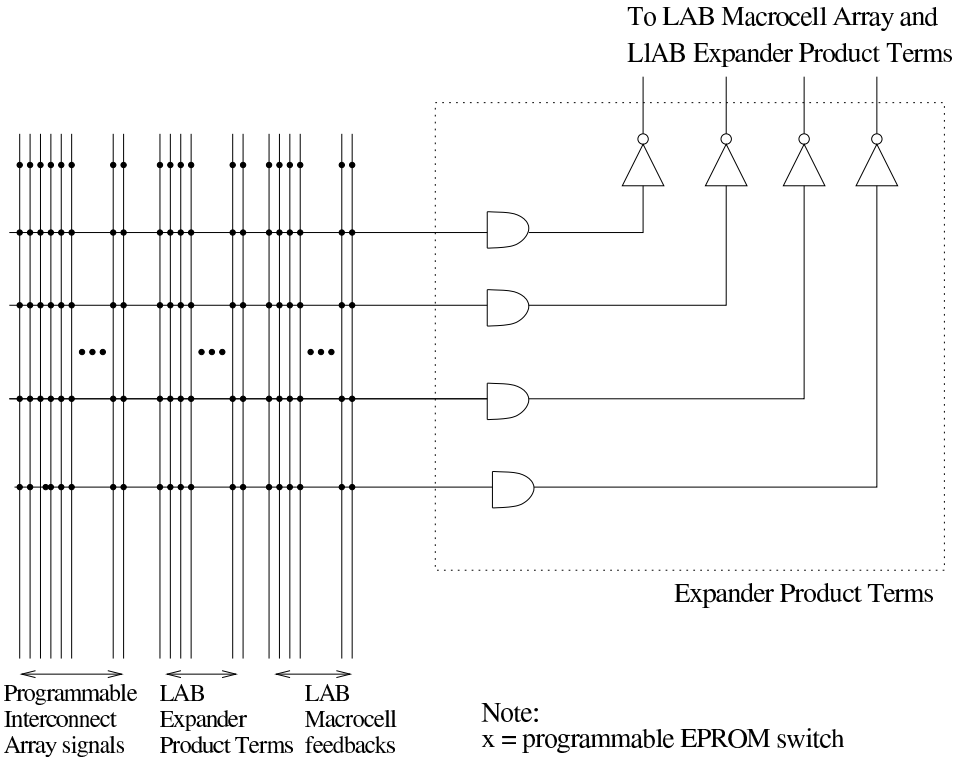


Figure 2.20: The expander product term of ALTERA EMP [4]

- A 256x8-bit Memory that can be used as a single port 256x1-byte memory or a two port 128x1-byte register file. This memory can be used to store micro instructions or data.
- Control Logic which serves mainly the function of generating a control bit by detecting a pattern condition such as a zero, negative, etc. result from the ALU. The generated bit is used to choose between contexts.

Each BFU can be configured as (1) a context storage unit, (2) a Data Memory unit, (3) ALU and Register file combination or (4) an independent ALU.

- *Interconnect Structure:*
  There are three levels of interconnects in the MATRIX architecture. Nearest neighbor connections, length four bypass connection as depicted in Figure 2.26 and (not shown in Figure 2.26) four global lines per row/column.

- *Reconfiguration:*
  The basic BFUs can be viewed as primitive $\mu ps$. At every cycle the program counter is incremented and the new program count is produced. At a configuration switch condition, the bit produced from the BFU controller is used to switch between a previously defined or stored instruction address or keep the new incremented program count. Each BFU generates its own switch conditions.

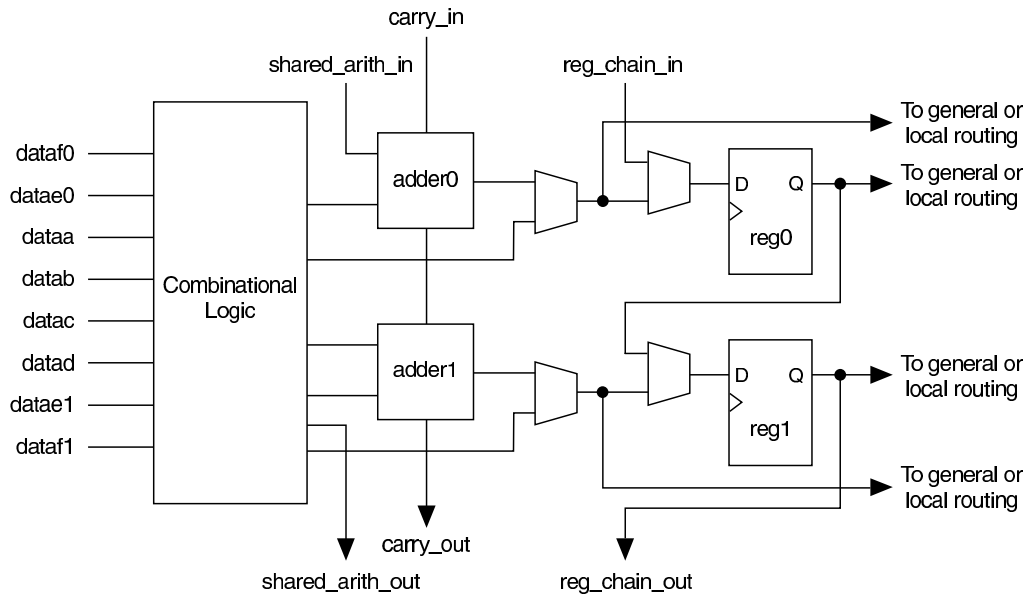


Figure 2.21: StratixII ALM [4]

Therefore MATRIX can be classified as dynamically and partially reconfigurable device with multi context support.

### 2.4.3 RAW

- *General Organization:*
  The RAW machine [90, 97] is an array of 16 identical programmable tiles. Each tile contains a MIPS style $\mu p$ with instruction memory and other peripherals. The tiles are interconnected using four routing networks. The RAW machine is designed to achieve very high speeds by tailoring the size of the tiles to the time delay needed for the electrical signal to tavel through them. The RAW machine is considered by its developers as a $\mu p$ with its proposed scalable instruction set architecture that can describe parallelism of the implemented application.

- *Processing Elements Architecture:*

  As shown in Figure 2.27 each tile is composed of

  - One static communication router,
  - Two dynamic communication routers,
  - An eight-stage in-order single issue MIPS-style processor,
  - A 32-Kbyte Data cache and

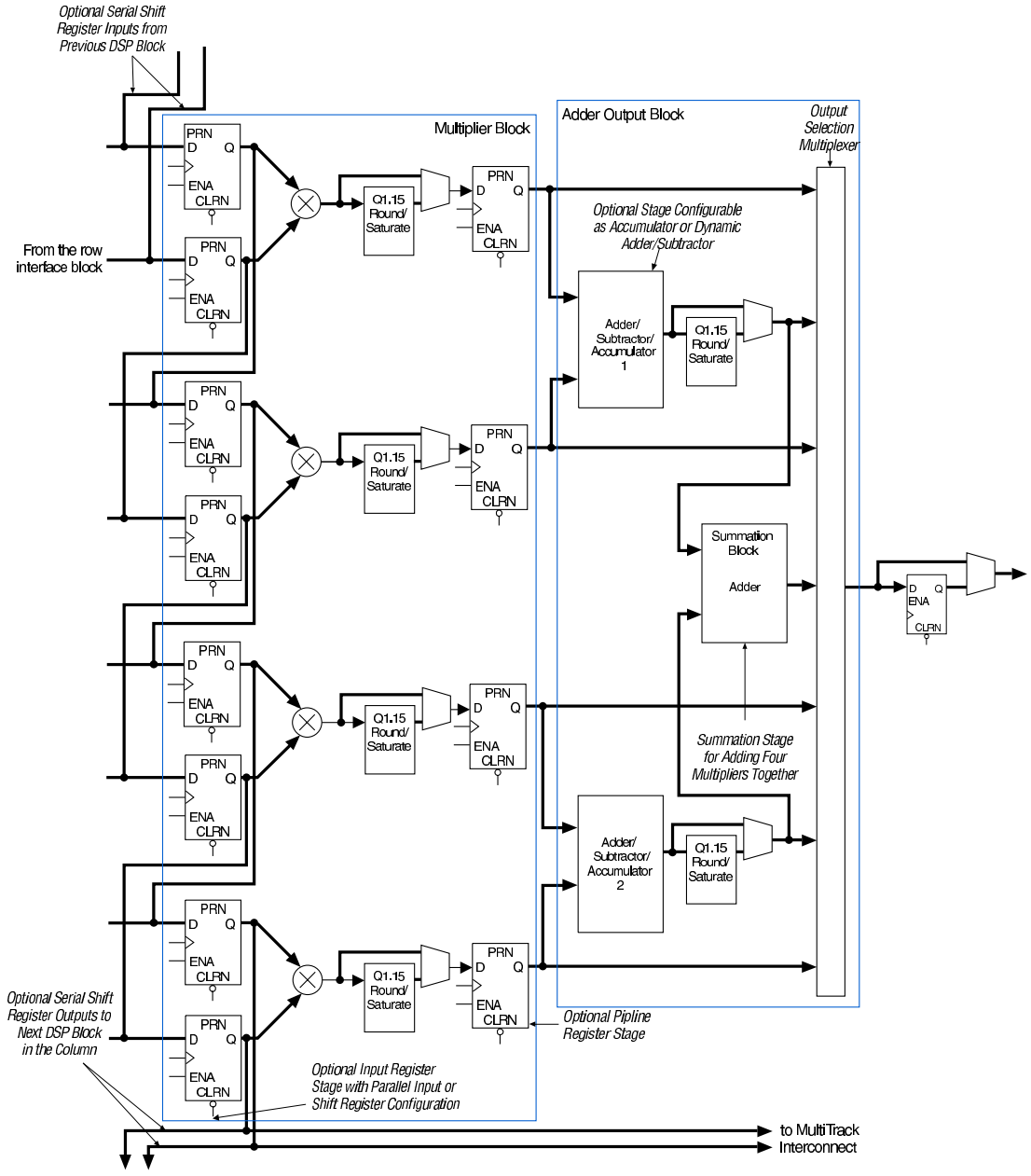


Figure 2.22: StratixII DSP block [4]

- A 96-Kbyte Instruction cache.

Each tile has input and output registers to achieve maximum pipelining speedup.

- *Interconnect Structure:*
  As pointed above, tiles are interconnected through static and dynamic networks. The routing provide full duplex connections to only four nearest neighbors. All

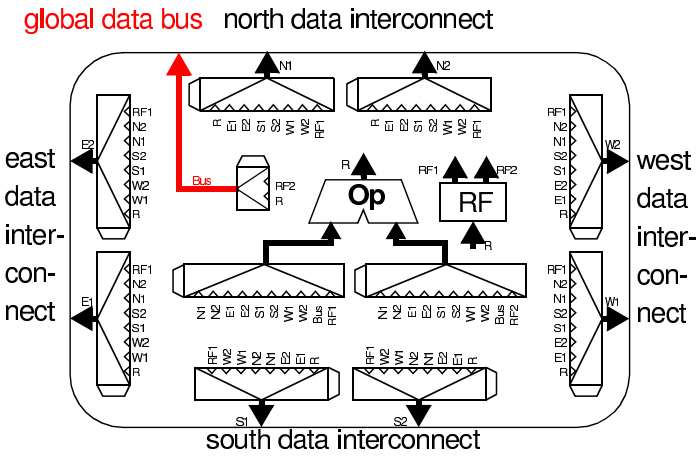


Figure 2.23: The general architecture of the KressArray rDPU [34]

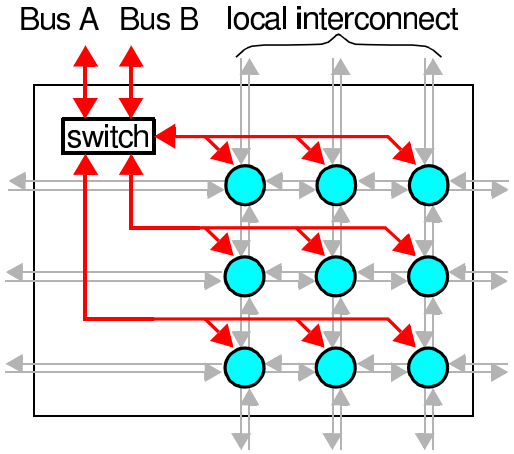


Figure 2.24: Inter-routing of a KressArray of 9 rDPUs [34]

wires are registered at the inputs and outputs of the tile to achieve high clock rates. Static routing is specified at compile time while dynamic routing is determined at run time. At the edge of the array the high number of input/output wires is reduced by multiplexing so that full duplex connections to any of the tiles located at the side of the array is possible as shown in Figure 2.28.

- *Reconfiguration:*
  The RAW machine can be considered as multi context since contexts are fed to the machine as instructions. RAW is also partially configurable since each tile operation is independent from the others and can have its own program running. Also, different tiles can be cascaded to achieve higher throughputs. Dynamic reconfiguration is facilitated through programming and dynamic switching.

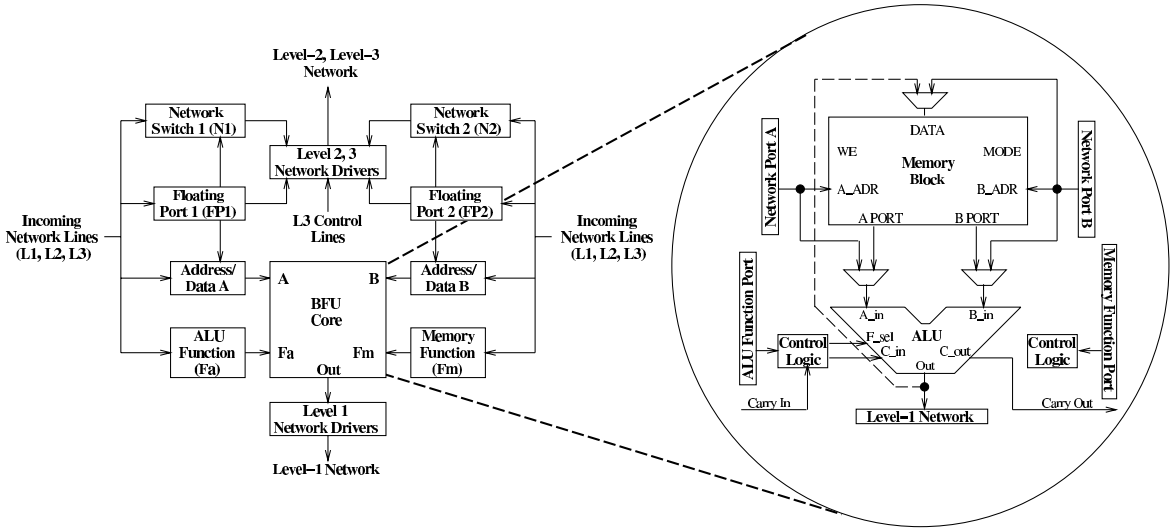


Figure 2.25: Structure of MATRIX BFU [62]

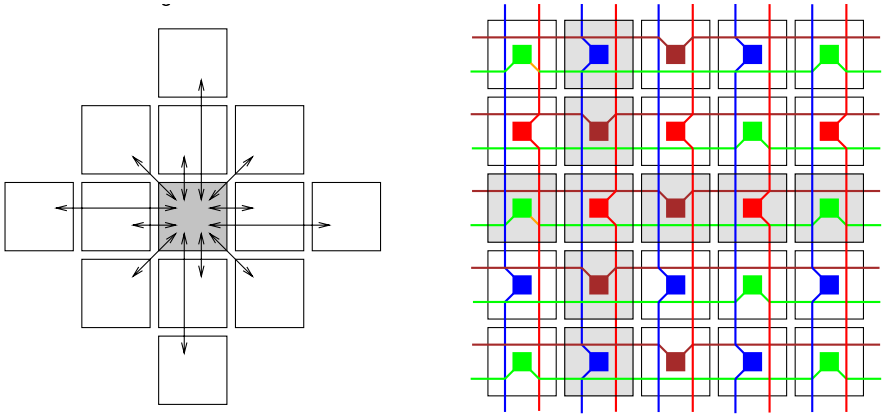

Figure 2.26: Interconnect topology of MATRIX [62]

### 2.4.4 MorphoSys

- *General Organization:*
  The MorphoSys [56, 82, 81, 60, 57] (shown in Figure 2.29) is a tightly coupled array of *Reconfigurable Cells* with a TinyRISC $\mu p$. The *Reconfigurable Cells* array is composed of four $4 \times 4$ quadrants. The TinyRISC processor is responsible of general purpose operations as well as controlling the operation of the *Reconfigurable Cells* array. The frame buffer provides two sets of data that the *Reconfigurable Cells* array can access simultaneously.

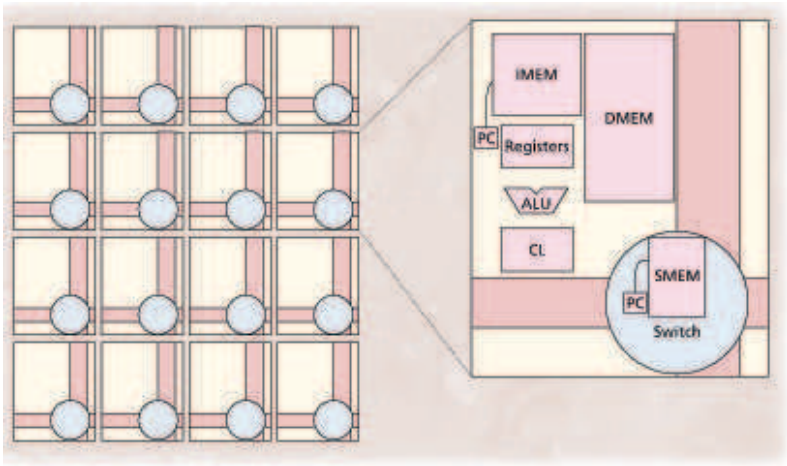


Figure 2.27: The architecture of RAW tiles [97]

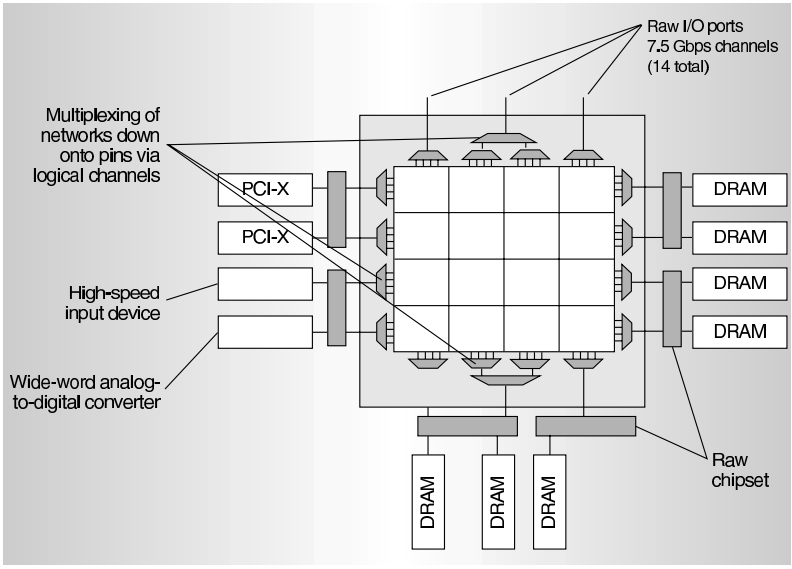


Figure 2.28: RAW routing architecture [90]

- *Processing Elements Architecture:*
  The *Reconfigurable Cells* (shown in Figure 2.30) is a $\mu p$ like block with a register file, input multiplexors, a context register an ALU/mult unit and a shifter. The *Reconfigurable Cells* is thus capable of MAC and other basic arithmetic and logic operations. The input multiplexors choose operands from the the ports of the *Reconfigurable Cells* or from the register file.

- *Interconnect Structure:*
  As depicted in Figures 2.31 and 2.32 the *Reconfigurable Cells* interconnect topology is of 3 levels: in the first level each *Reconfigurable Cells* connects to its four nearest

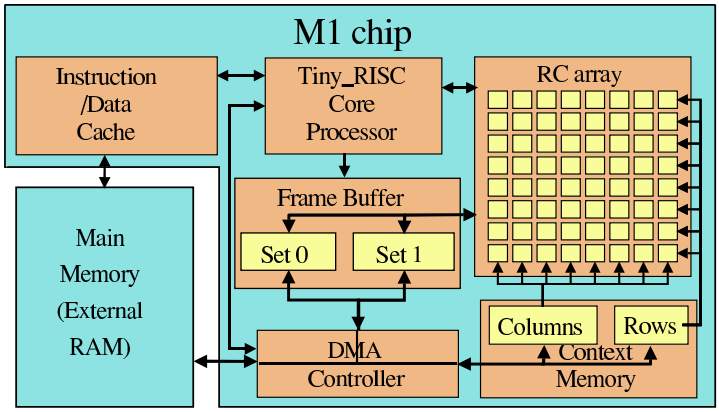


Figure 2.29: Overview of the MorphoSys architecture [60]

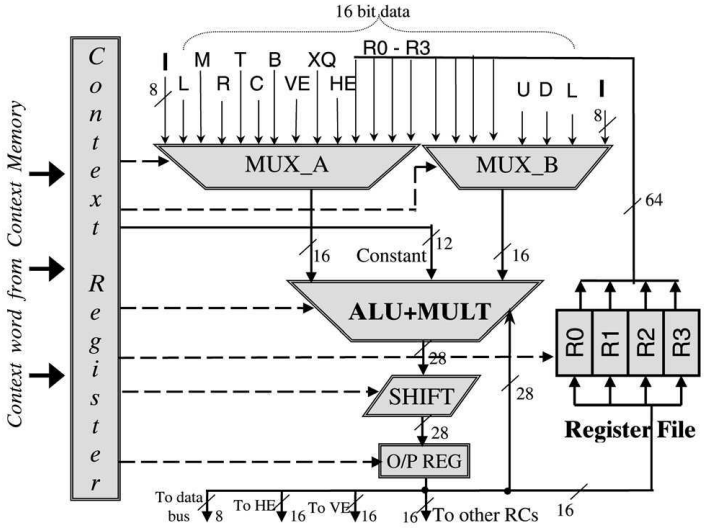


Figure 2.30: Structure of the MorphoSys *Reconfigurable Cell* [81]

neighbors, in the second level complete quadrant row and column connections is provided and in the third inter-quadrant connectivity is implemented.

- *Reconfiguration:*
  To increase context efficiency, contexts can be broadcasted on columns or rows with the *Reconfigurable Cells* sharing the same column or row share the contexts and thus performing the same function forming a *S*ingle *I*nstruction *M*ultiple *D*ata (*SIMD*) topology over the column or row. There are 16 context words per column and row that can be chosen from according to the control signals provided by the TinyRISC processor. Dynamic reconfiguration is achieved by updating the contexts of the *Reconfigurable Cells*. This can be done in the background

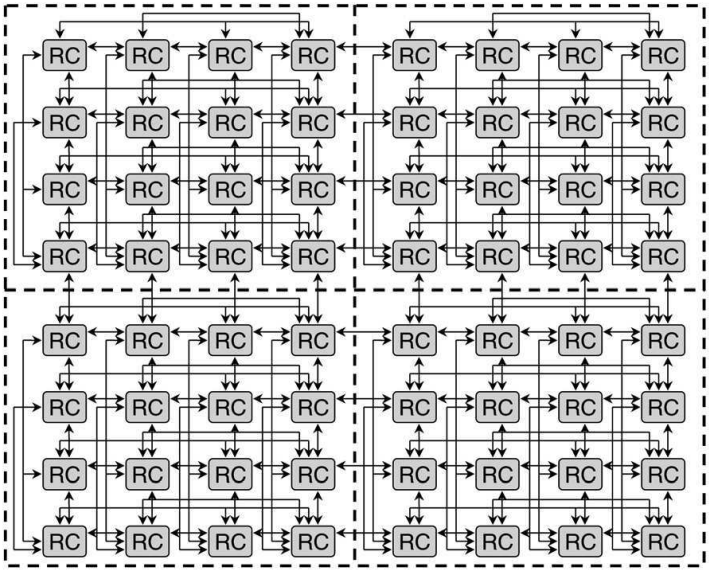


Figure 2.31: Intra-quadrant routing between the MorphoSys *Reconfigurable Cells* [81]

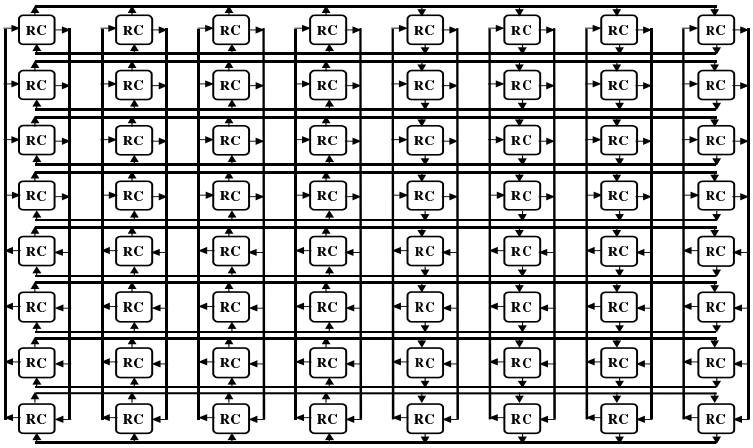


Figure 2.32: Inter-quadrant routing of MorphoSys [56]

as the *Reconfigurable Cells* are running allowing thus RTR. Here again since rows and columns hold a number of context words that are selected from and since they can be reconfigured independently the MorphoSys architecture can be considered multi context and partially reconfigurable.

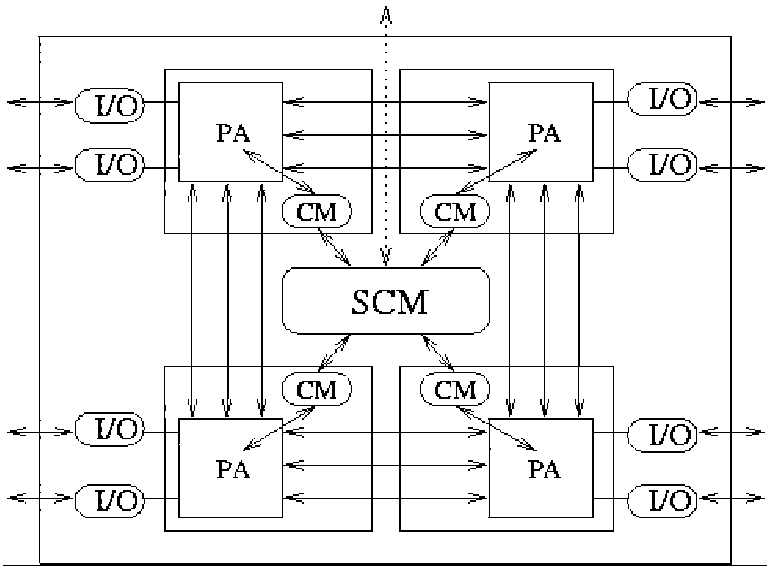


Figure 2.33: Hierarchical structure of an XPP device of four PAs [69]

### 2.4.5 PACT XPP

- *General Organization:*
  PACT XPP (eXtreme *P*rocessing *P*latform) [70, 69, 6, 71] is a composition of *P*rocessing *A*rrays (*PA*s) each composed of *P*rocessing *A*rray *E*lements (*PAE*) and a *C*onfiguration *M*anager (*CM*). As shown in Figure 2.33, a *S*upervising *CM* (*SCM*) manages individual CMs. The complete I/O interfaces are available at the edges of the XPP device. The XPP is a data-driven architecture with partial and dynamic configuration support.

- *Processing Elements Architecture*
  Figure 2.34 shows the architecture of a PA. Two types of PAE are used: an ALU-PAE 2.35 and a RAM-PAE 2.36. Both the ALU-PAE and the RAM-PAE are composed of three sub-blocks and each block contains an ALU. The FREG and BREG blocks are mainly used to connect the corresponding PAE forwards or backwards to the top or bottom interconnect resources. The ALUs of the FREG and BREG are used for routing and control of data and can also be used for arithmetic operations. The ALU sub-block in the ALU-PAE features also a multiplier and other specific DSP operations. The RAM sub-block can be configured as a dual port 512x16-bit RAM or as a FIFO.

  Processing in the XPP is event and data driven. A data packet and an event packet are sent simultaneously. The processing of the appropriate PAE is stalled until all operands to be consumed are ready. The PAE is also stalled if the destination inputs are busy.

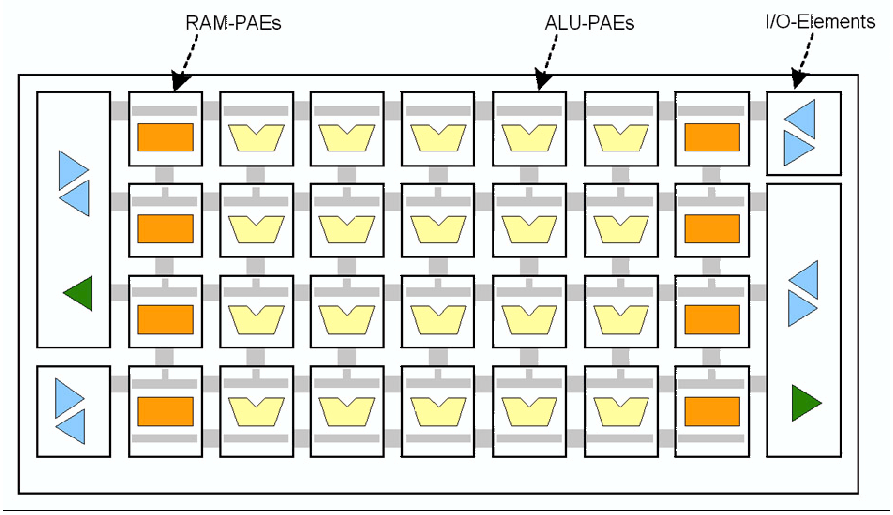


Figure 2.34: Structure of a $4 \times 5$ PA [70]

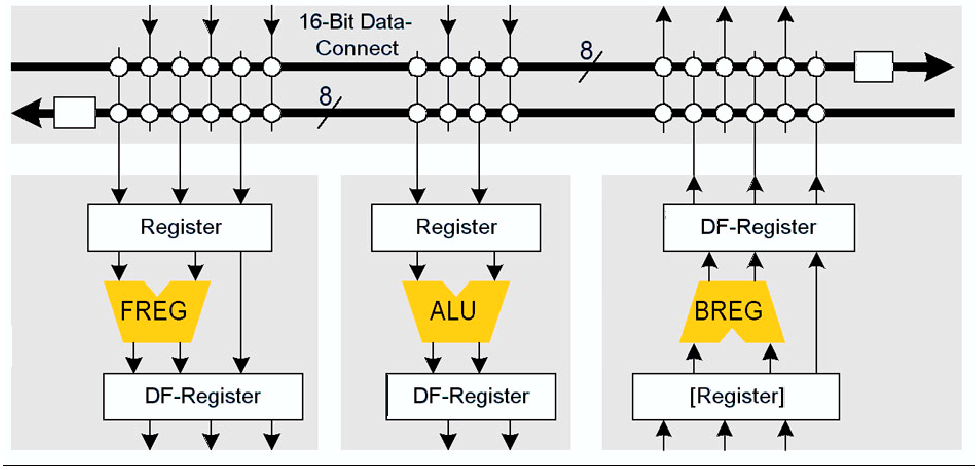


Figure 2.35: Structure and routing of the ALU-PAE [70]

- *Interconnect Structure:*
  As illustrated in Figures 2.35 and 2.36, connectivity is mainly via segmented horizontal 16-bit interconnects. As mentioned above, FREG an BREG assist in transferring data vertically in the PA.

- *Reconfiguration:*
  PACT XPP is partially and dynamically reconfigurable. The CM sends reconfiguration data addressed to the individual PAE. This data travels through the array until it reaches its destination. Once configured the PAE changes its state to "configured". The CM can hold multiple configurations making the XPP thus

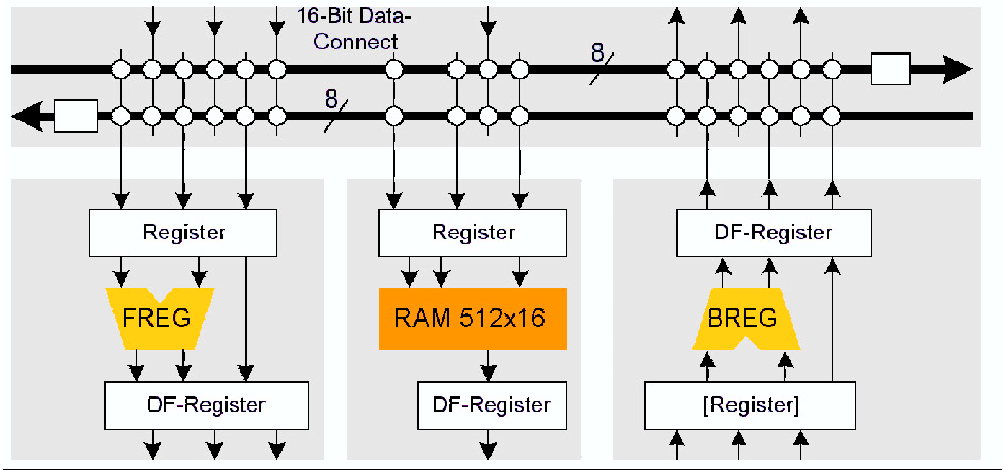


Figure 2.36: Structure and routing of the ALU-PAE [69]

multi-context. New contexts can be loaded in the background at run time to save context fetching time. Moreover, complete configurations could be switched between in only a small number of cycles. Because of the event packet driven operation of the XPP no loss of data is expected to occur with dynamic configuration. In addition, special hardware protocols are implemented to ensure immunity against dead-locks in case of overlapping partial configurations.

## 2.5 Concluding Remarks

In this Chapter, a short introduction on reconfigurable technologies was presented. To give a more comprehensive idea about reconfigurable computing features and architectures, a collection of FGRC and CGRC solutions was introduced.

Although RC possesses many attractive features and potential, they are not suited for all types of applications. The more flexible the computational solution is, the less power efficient and fast it is per computational task.

Since better efficiencies and more flexibility is still demanded, the set of RC solutions is now polarized to two major subsets: CGRC and FGRC solutions. As a matter of fact, within each of the aforementioned subsets, architectures are still evolving to better their flexibilities and efficiencies.

Some very interesting observations could be deduced after studying the introduced old and new FGRC solutions in Section 2.3: the move towards coarse-granularity and heterogeneity yet with maintenance of more or less the same degree of flexibility. XILINX, Actel and ALTERA new FPGAs have now larger basic logic blocks capable of more FG functionality. They all have also dedicated configurable RAM blocks i.e. heterogeneity.

Moreover, both XILINX and ALTERA new FPGAs exhibit DSP blocks implementing in high efficiency multiplication and accumulations operations.

A similar observation was made when studying the CGRC solutions: the CGRC architecture should relate to the target application. The SIMD based applications have inspired the design of the MorphoSys for example. In [36] the authors noted that the sought application field should reflect on the design of the CGRC architecture.