QUKU: A FPGA Based Flexible Coarse Grain Architecture Design Paradigm using Process Networks

Sunil Shukla[1,2], Neil W. Bergmann[1], Jürgen Becker[2]

[1]University of Queensland
Information Technology & Electrical Engg.
St. Lucia, QLD 4072, Australia
{sunil, n.bergmann}@itee.uq.edu.au

[2]Universität Karlsruhe
Institut für Technik der Informationsverarbeitung (ITIV)
76131 Karlsruhe, Germany
becker@itiv.uni-karlsruhe.de

Abstract

DSP applications can be suitably represented using Process Network Models. This paper uses a modification of Kahn Process Network to solve the problem of finding an optimum architectural template for coarse grain array on per application basis. By applying the model at architectural level in QUKU, better hardware efficiency is achieved for a wide domain of applications. A few widely used DSP algorithms have been presented to demonstrate the application of process network models into architectural template generation in QUKU.

1. Introduction

FPGAs have come a long way from just being used as a platform for implementing glue logic. The fine grained structure of FPGAs has been seen as being unsuitable for implementing coarse grain algorithms. Moreover FPGAs have been touted as power hungry devices suitable for implementing control logic only.

To overcome these disadvantages, CGRAs (Coarse Grained Reconfigurable Architectures) have been proposed [1]. CGRAs have advantages over conventional FPGAs in terms of ease of reconfigurability and power consumption. Despite all these advantages, CGRAs have failed to make a mark in the industry. The failure can be attributed to the factors like inflexibility, lack of unified design flow, commercial unavailability and lack of application domains.

ASICs and GPPs (General Purpose Processors) lie at the extreme ends of flexibility versus performance graph. The current demand is for performance with increased flexibility. FPGA lies in between ASICs and GPPs in the flexibility versus performance graphs. One of the attractive features of FPGA is the capability for dynamic and partial reconfiguration. There has been a lot of research to do partial dynamic reconfiguration on FPGA. Donthi and Haggard describe the extent of reconfigurability on the existing commercial FPGAs [2]. Despite the popularity and inherent capability of FPGAs for partial reconfiguration, the vendors have not supported it well commercially. Despite all the technological advances, run time reconfigurability is still hindered by the lack of tool support from vendors and the long reconfiguration time [3]. Refer to [4, 3, 5] to read more about partial run time reconfigurability on FPGA devices.

QUKU is a coarse grained overlay for FPGA. The idea is to develop an architecture based on a proven and affordable platform which doesnt have the inherent limitations of the underlying platform. The architecture has already been presented in [6, 7]. This paper aims in bringing out the design methodology of dynamic reconfiguration with and without using fine grained reconfiguration. We will also be discussing the KPN network model applicability as a MoC (Model of Computation) and architecture to QUKU.

The paper is formatted into following sections. The next section gives a short architectural overview of QUKU. Section3 describes the process network graphs and their analogy to PE array. Section 4 covers the dual layered reconfigurability feature of QUKU. Section 5 describes the design methodology discussing the implementation details of a few commonly used DSP algorithms. Section 6 presents the result followed by conclusion.

2. QUKU Architecture

QUKU is a merger of two technologies: CGRAs and FPGAs. The aim is to develop a system which is based on commercially available and affordable technologies but at the same time provides active support for fast and efficient

1-4244-0910-1/07/$20.00

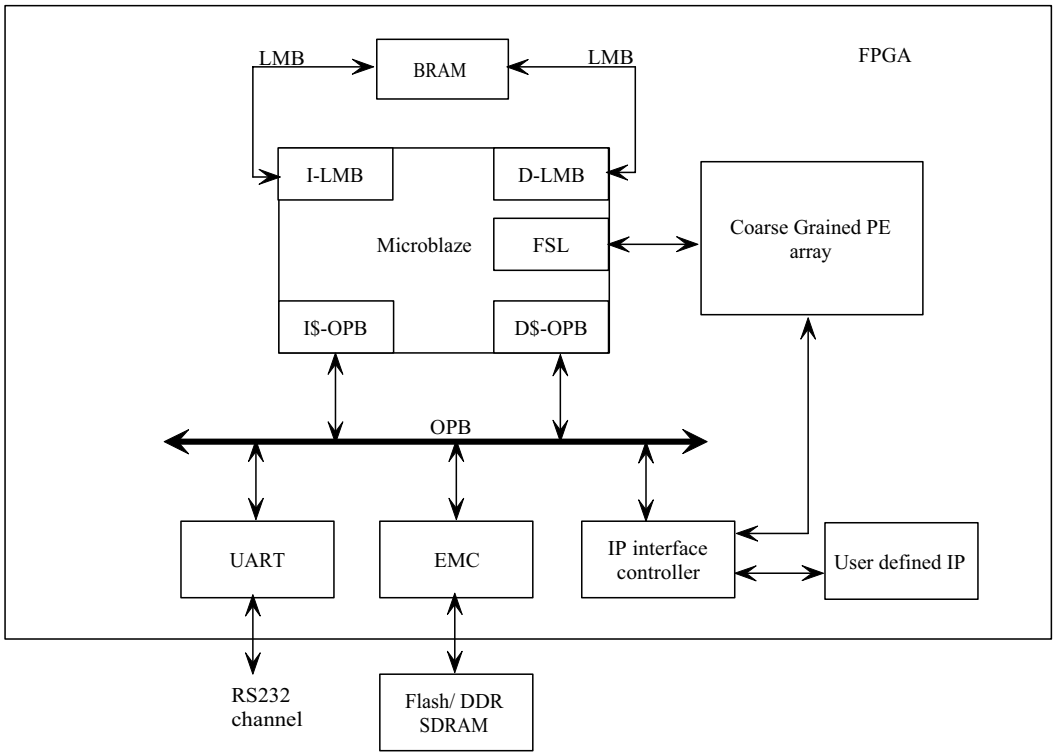


Figure 1. Block diagram of QUKU

dynamic reconfiguration. This implementation will enable us to use the same platform for applications which are a mix of control flow and computationally intensive applications. Our system is unique in that it provides two levels of application-specific reconfigurability. This dual level reconfigurability is discussed in Section 4. QUKU is a complete SoC solution consisting of a coarse-grained PE matrix overlaid on a FPGA. Fig. 1 shows a detailed block diagram of QUKU. The PE array is coupled with Microblaze based soft processor core using FSL (Fast Simplex Link). Microblaze is responsible for running software processes and scheduling algorithm. OPB (On-chip Peripheral Bus) is used to connect to peripherals like external memory controllers, UART, timer device and other user defined IP cores.

2.1. Coarse Grain Programmable Array

The coarse gained programmable matrix consists of a dynamically reconfigurable PE array, CMM and AMM. Refer to fig. 2 for a block level description and acronyms. Each PE consists of a LCC and LAC besides containing a functional unit and a memory sub-system. CMM is responsible for loading the configuration data onto the LCC of the PEs. AMM loads address parameters onto the LAC of the PEs. LAC controls read and write access to data and result memory of the respective PE. It also generates address for accessing each memory.

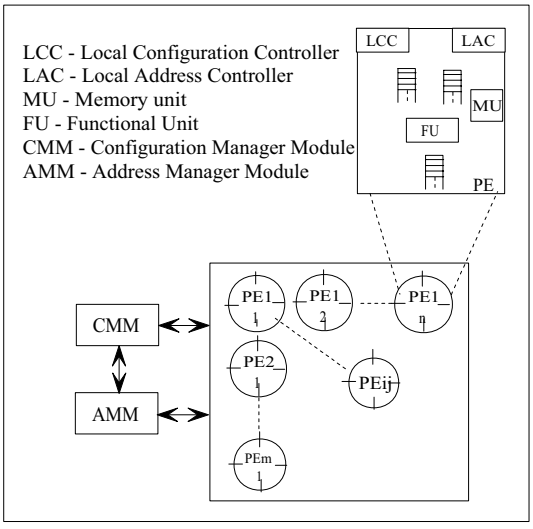
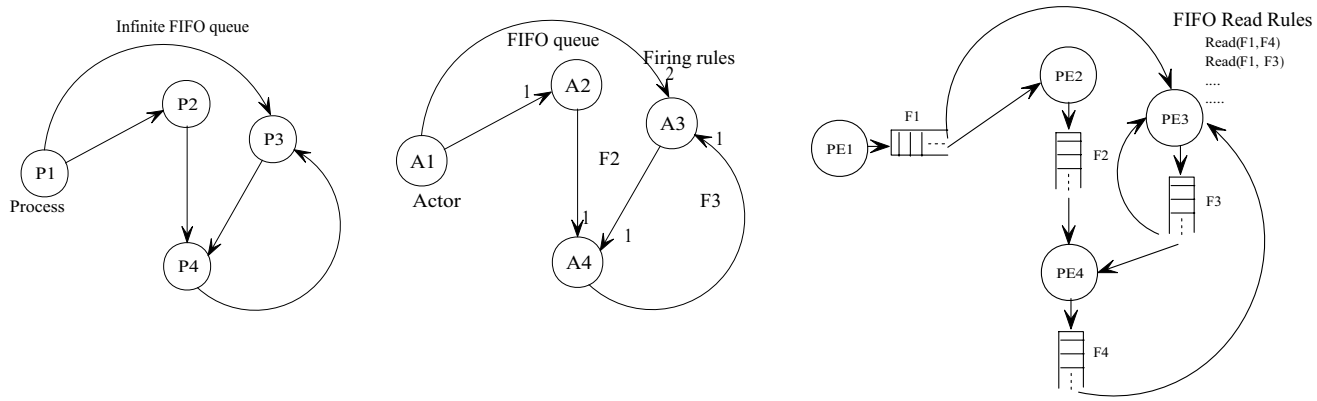


Figure 2. Block diagram of Coarse Grain PE Array



a) Kahn process network b)Dataflow process network c) QUKU PE array process network

Figure 3. Process Network diagrams

2.2. PE Array Organization

QUKU follows MIMD style architecture with distributed memory. In MIMD, it is not feasible to provide a point-to-point connection between all PEs, especially for a large network of PEs. Hence a PE is connected to a fixed number of neighboring PEs. Two of the commonly used topologies are mesh and hypercube [8]. Hypercube has the disadvantage of requiring the number of PEs which is a power of 2. Although mesh based topology has no such limitation, the diagonal length is larger as compared to hypercube resulting in longer worst case interconnection delay.

In any type of interconnection it is sometimes required to configure an intermediate PE as a pass through. So, a conventional MIMD/SIMD architecture suffers from this data routing problem as any multi-processor architecture has a fixed inter-connection topology. QUKU doesnt suffer from this problem of finding an optimal interconnect strategy as during design time an application is first divided into processes and a network of process is created which is a special case of KPN (Kahn Process Network). This process network provides an insight into how different processes are mapped. Depending on this knowledge an appropriate interconnect strategy is used and implemented.

3. Process Networks

3.1. Introduction

Process Networks is a Model of Computation (MoC) that was originally developed for modeling distributed systems but has proven its convenience for modeling signal processing systems. Kahn introduced a formal process model in [9], commonly known as Kahn Process Network (KPN). It is a natural model for describing signal processing systems where infinite streams of data are incrementally transformed by processes executing in sequence or parallel. In a KPN, concurrent processes communicate only through one-way FIFO channels with unbounded capacity. Writes to the channels are non-blocking but reads are blocking. It is not feasible to implement KPN in hardware due to infinite queue length. Lee and Parks proposed DataFlow Networks (DFN), a modification over KPN, where processes are replaced by actors and there are set of firing rules for each actor [10] . When an actor fires, it consumes a finite number of tokens and produces a finite number of output tokens.

3.2. Analogy of QUKU PE Array with Process Networks

PEs in QUKU are analogous to actors in DFN. The FU in a PE acts as an actor and the firing rules are stored in LCC. One firing rule results in consumption of one or more input tokens producing a finite number of tokens at the output. The dataflow network of PE array in QUKU is shown in Fig. 3(c). In many signal processing applications the firing sequence can be determined statically at compile time. This class of dataflow process networks are called synchronous dataflow networks.

In fig. 3(c), it is shown that one channel FIFO may be connected with multiple actors whereas a typical dataflow network implements separate channels to support this. Instead, in QUKU, it is implemented as one channel with multiple outputs. There is some intelligence built in the channel to verify a legitimate read among all the possible candidates. This condition occurs because QUKU supports multiple configurations where ingress and egress actors may change in each clock cycle. The disadvantage of this model is that if actor PE2 is blocked then the channel F1 will wait

for PE2 before PE3 can access the channel. But the advantage is that there is no need of having as many FIFOs as the number of channels connecting one actor to others. This is a huge advantage in terms of hardware.

4. Reconfiguration in QUKU

Conventional CGRAs support very fast dynamic reconfiguration using micro-codes as opposed to Megabytes of configuration data in FPGAs. But CGRAs have fixed physical layout which ties their usability to a particular application domain. The uniqueness of QUKU lies in the fact that it supports dual layered reconfiguration. The dual layered reconfiguration capability overcomes the slow reconfiguration of FPGAs as well as overcomes the problem of finding an optimal architecture which suits a wide variety of application domains. The two reconfiguration planes are:

- FPGA level/fine grain reconfiguration
- PE level/coarse grain reconfiguration

4.1. Fine Grain Reconfiguration

This is an infrequently happening reconfiguration which typically takes a few milli-seconds. This method, although takes a few milli-seconds, serves the important function of physically reconfiguring the array. This includes a change in data width, change in numeric representation of data from fixed point to floating point or even changing the physical structure of the array itself. There is no single PE layout which can be described as best match for all the applications. Hence this reconfigurability provides designer an excellent way to select a PE layout which can serve the target applications in the best possible way. A practical scenario could be the case where the radio receiver is tuned to DAB transmission and then after a while the user wants to tune to the alternative transmission which may follow DRM (digital Radio Mondial). In such case, it's worth reconfiguring the complete FPGA to reload the new configuration which is more optimized for the application.

4.2. Coarse Grain Reconfiguration

The very frequently happening reconfiguration is at PE level which configures the PE array to perform the new functionality, without changing the physical aspects of the array. The very fast occuring dynamic reconfiguration of the PE array provides an excellent option to overcome the slow partial dynamic reconfiguration methodology used in FPGAs. Huebner et al have done experiments on FPGA wherby the FPGA is divided into slots and each slot can hold a module. This slot based reconfiguration took about 1.5 milli-seconds per slot [4]. QUKU overcomes this long reconfiguration time by using short micro-codes which reconfigures the PE array in few nano-seconds. The PE array can change its coarse grained configuration every clock cycle.

5. Design Methodology

This section describes the QUKU design methodology. The design process can be divided into three phases:

- Design phase
- Compile Phase
- Execution Phase

We have taken the case study of DFT (Discrete Fourier Transform), FIR filter, complex multiplication and FFT. With the aid of these algorithms we would be discussing the design methodology.

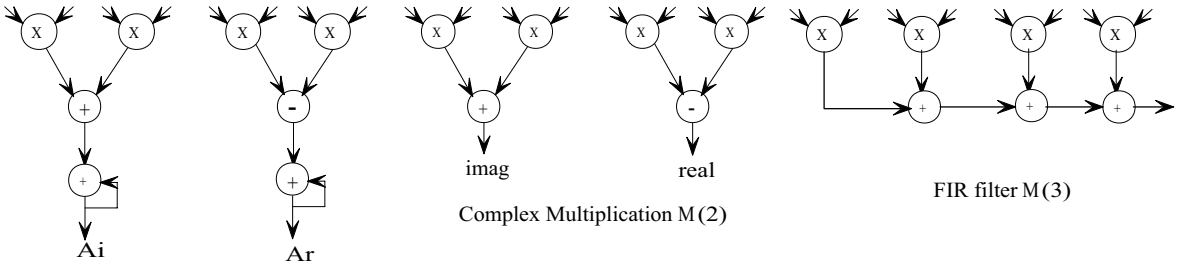
5.1. Design Phase

During design phase, a manual/automatic analysis of the application is done. The decision is taken about all the computationally intensive part of the application which needs to be offsourced to the PE array. Once all the modules which are to be mapped on the PE array are decided then a data flow diagram is prepared. At this phase, the designer is not bothered about the physical aspects of the PE array. The user perceives the PE array as a homogeneous function rich array with no interconnection bottleneck. The data flow diagram of the four algorithms are shown in fig. 4.

5.2. Compile Phase

After design phase, QUKU compiles the process network to generate a heterogeneous array of PEs, with each PE optimized for just the range of operations it is required to perform for a given set of applications. In compile phase, the aim is to find an optimum PE layout and do hardware-software partitioning of tasks.

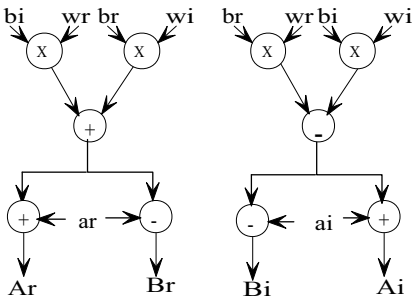
Fig. 4 shows the process network diagram for the four algorithms. The problem of finding an optimum match can be simplified using mathematical set theory. The individual modules, M(1), M(2), M(3) and M(4) can be viewed as a set of records. Each record contains elements which define a PE for that module. The definition consists of PE configuration data and the interconnect structure. The set of records make up a module. These modules then define the final PE layout using one of the rules given in equation 1.



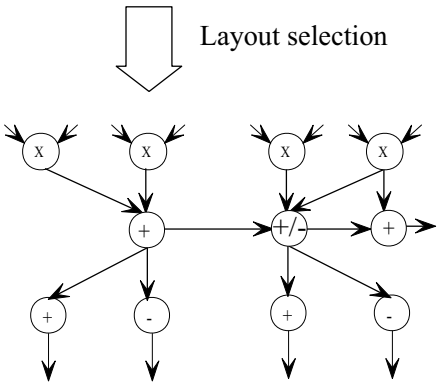
Discrete Fourier Transform M(1)

Complex Multiplication M(2)

FIR filter M(3)



Fast Fourier Transform M(4)



PE layout (superset of all the above data flow graphs)

Figure 4. Process Network diagram for algorithms

The chosen rule depends on the factors like area, concurrency, processing efficiency, etc.

$$PE_layout = \{M(1) \cup M(2) \cup M(3) \cup M(4)\} \quad (1a)$$
$$PE_layout \supset \{M(1), M(2), M(3), M(4)\} \quad (1b)$$
$$PE_layout = \{M(i) \cup M(j)\} \quad (1c)$$
$$PE_layout = M(k) \quad (1d)$$

where:
$\supset$ indicates superset,
$\cup$ indicates union of two sets,
i, j and k are arbitrary module indices

5.3. Execution Phase

During the execution phase, the central software controller runs the software processes and controls the loading of different modules on the PE array and provides inter-module synchronization. If not active, PEs remain in reset state to save power.

6. Results

The data flow graphs for each algorithm is shown in fig. 4 and also shown is a resultant PE layout which is a superset of all the individual data flow graphs. To compare the different implementation options, synthesis was done for the alternative implementations listed in equation 1. Fig. 5 shows the hardware area for different implementations. The applications were implemented on Xilinx ML401 board which has XC4VLX25 device. The slice count shown is just for the coarse grained PE array and not the complete system. The %age figure repersents the ratio of actual slice count and the available slice count in Virtex4 XC4VLX25 device.

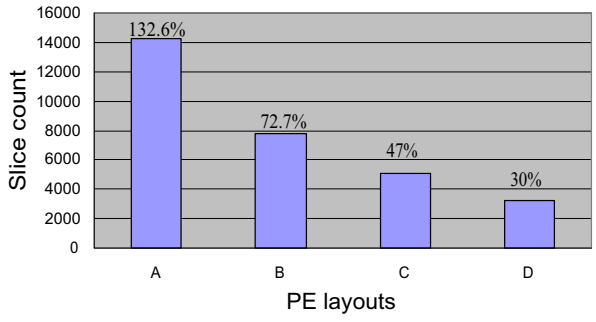


Figure 5. Implementation results for different layout options.

The different PE layout options i.e. A, B, C and D are defined below.

$$A = \{M(1) \cup M(2) \cup M(3) \cup M(4)\}$$
$$B = \{M(3) \cup M(4)\}$$
$$C \supset \{M(1), M(2), M(3), M(4)\}$$
$$D = M(3)$$

7. Conclusion

In this paper, we have described the dual layered reconfiguration capability of QUKU and the design methodology using process network graphs. This process network based architectural exploration is unique to QUKU and enables the designer to comprehend the performance during design time and select the most optimum design. We also presented the various implementation methods of an application. Future work includes making a GUI for automatic extraction of computationally intensive kernels from the source code and deriving the configuration code for the PEs.

References

[1] Reiner W. Hartenstein. A decade of reconfigurable computing: a visionary retrospective. In *Proc. of the Design Automation and Test in Europe (DATE'01)*, pages 642–649.

[2] S. Donthi and R. L. Haggard. A survey of dynamically reconfigurable FPGA devices. In *Proc. of the 35th Southeastern Symposium on System Theory, 2003*, pages 422–426.

[3] Shannon Koh and Oliver Diessel. COMMA: A communications methodology for dynamic module-based reconfiguration of FPGAs. In *ARCS Workshops*, pages 173–182.

[4] M. Huebner, C. Schuck, and J. Becker. Elementary block based 2-dimensional dynamic and partial reconfiguration for virtex-II FPGAs. In *Proc. of the 20th International Symposium on Parallel and Distributed Computing (IPDPS'06)*, pages 8–16.

[5] C. Bobda, M. Majer, A. Ahmadinia, T. Haller, A. Linarth, J. Teich, and J. V. D. Veen. The Erlangen slot machine: A highly flexible FPGA-based reconfigurable platform. In *Proc. of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM05)*, pages 319–320.

[6] Sunil Shukla, Neil W. Bergmann, and Juergen Becker. QUKU: A two level reconfigurable architecture. In *Proc. of IEEE Computer Society Annual Symposium on VLSI (ISVLSI06)*, pages 109–116.

[7] Sunil Shukla, Neil W. Bergmann, and Juergen Becker. QUKU: A fast run time reconfigurable platform for image edge detection. In *Lecture Notes in Computer Science*, volume 3985, pages 93–98, 2006.

[8] Mimd architectures, online at http://carbon.cudenver.edu/ galaghba/mimd.html.

[9] G. Kahn. The semantics of a simple language for parallel programming. In *Information Processing*, pages 471–475, August 1974.

[10] Edward Lee and Thomas Parks. Dataflow Process Networks. In *Proc. of the IEEE*, pages 773–799, May 1995.