

Mapping of DSP Algorithms on Field Programmable Function Arrays

Paul M. Heysters, Jaap Smit, Gerard J.M. Smit, Paul J.M. Havinga

University of Twente, depts. Computer Science & Electrical Engineering,
Enschede, the Netherlands
heysters@cs.utwente.nl

Abstract. This position paper¹ discusses reconfigurability issues in low-power handheld multimedia systems. A *reconfigurable systems-architecture* is introduced, with a focus on a *Field Programmable Function Array* (FPFA). Application domain specific algorithms determine the granularity of FPFA processor tiles. Several algorithms are discussed and mapped onto a FPFA processor tile.

1 Introduction

In the next decade two trends will definitively play a significant role in driving technology: the development and deployment of personal mobile computing devices and the continuing advances in integrated circuit technology. The semiconductor technology will soon allow the integration of one billion transistors on a single chip [1]. This is an exciting opportunity for computer architects and designers; their challenge is to come up with system designs that efficiently use the huge transistor budget and meet the requirements of future applications. The development of personal mobile devices will give an extra dimension, because these devices have a very small energy budget, are small in size but require a performance that exceeds the levels of current desktop computers. The functionality of these mobile computers will be limited by the required energy consumption for communication and computation.

The way out is *energy efficiency*: doing more work with the same amount of energy. Traditionally, designers have been focused on low-power techniques for VLSI design. However, the key to energy efficiency in future mobile multimedia devices will be at the higher levels: energy-efficient system architectures, energy-efficient communication protocols, energy-cognisant operating systems and applications, and a well designed partitioning of functions between wireless device and services on the network.

Mobile computers must remain usable in a variety of environments. They will require a large amount of circuits that can be customized for specific applications to stay versatile and competitive. *Reconfigurability* is thus an important requirement for mobile systems, since the mobiles must be flexible enough to accommodate a variety of multimedia services and communication capabilities and adapt to various operating conditions in an (energy) efficient way.

¹ This research is supported by PROGRESS, the embedded systems research program of the Dutch organisation for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the Technology Foundation STW.

Research has shown that adapting continuously the system and protocols can significantly improve the energy efficiency while maintaining a satisfactory level of performance [4].

Reconfigurability also has another more economic motivation: it will be important to have a fast track from sparkling ideas to the final design. If the design process takes too long, the return on investment will be less. It would further be desirable for a wireless terminal to have architectural reconfigurability whereby its capabilities may be modified by downloading new functions from network servers. Such reconfigurability would also help in field upgrading as new communication protocols or standards are deployed, and in implementing bug fixes [3]. One of the key issues in the design of portable multimedia systems is to find a good balance between flexibility and high-processing power on one side, and area and energy-efficiency of the implementation on the other side.

Finally, a major obstacle to designing one billion transistor systems is the physical design complexity, which includes the effort devoted to the design, verification and testing of an integrated circuit. A possible solution is to work with a highly regular structure since they only require the design and replication of a single processor tile and an interconnection structure. We have designed a reconfigurable architecture that is suitable for many DSP-like algorithms and yet is energy-efficient. In this paper we will show how various algorithms map on this architecture.

2 Reconfigurable systems architectures

The design of energy efficient hand-held multimedia computers cannot be done in isolation. The energy problem has to be considered at all layers of a system.

The interconnect of a system contributes significantly to the total energy consumption of a system. Experiments have demonstrated that in chip-designs, about 10 to 40% of the total power is dissipated in buses, multiplexers and drivers [5]. This amount can increase dramatically for systems with multiple chips due to large off-chip bus capacitance. Measurements on a Xilinx XC4003 FPGA show that at least 65% of a design's power is dissipated in the collection of interconnect resources and logic cell interface circuitry [3].

Multimedia applications have a high computational complexity. They also have regular and spatially local computations. Exploiting such locality of reference improves the energy efficiency of a system.

With high-speed wireless networks, many different architectural choices become possible, each with different partitioning of functions not only between the resources of the hand-held itself, but also between servers resident in the network. Partitioning is an important architectural decision, which dictates where applications can run, where data can be stored, the complexity of the mobile and the cost of communication services [4].

Our approach to cope with the challenges mentioned above is to have a reconfigurable systems-architecture, in combination with a QoS driven operating system. In our architecture locality of reference is exploited at several levels. The main philosophy used is that operations on data should be done at the place where it is most energy efficient and where it minimizes the required communication. This can be achieved by matching computational and architectural granularity.

In our architecture, we have an organization of a programmable *communication switch* surrounded by several autonomous modules. Modules communicate without involvement of

the main processor. For example, an audio stream from the network can be sent directly to the audio module. In a system we differentiate three grain-sizes of operations:

- *fine grained* operations in the modules that perform functions like multiply and addition.
- *medium grained* operations are the functions of the modules. The functional tasks are allocated to dedicated (reconfigurable) modules (e.g. display, audio, network interface, security, etc.) [2].
- *course grained* operations are those tasks that are not specific for a module and that can be performed by the CPU module, or even on a remote compute server. This partitioning is mainly a task of the operating system.

In the remaining part of this paper we will focus on the fine-grained reconfigurable processing modules, and more specifically on the Field Programmable Function Array.

3 Field Programmable Function Array

Field-Programmable Function Arrays (FPFAs) are reminiscent to FPGAs, but have a matrix of ALUs and lookup tables [7] instead of Configurable Logic Blocks (CLBs). Basically the FPFA is a low power, reconfigurable accelerator for an application specific domain. Low power is mainly achieved by exploiting locality of reference. High performance is obtained by exploiting parallelism. A FPFA consists of interconnected *processor tiles*. Multiple processes can coexist in parallel on different tiles. Within a tile multiple data streams can be processed in parallel. Each processor tile contains multiple *reconfigurable ALUs*, local memories, a *control unit* and a *communication unit*. Figure 1 shows a FPFA with 25 tiles; each tile has five ALUs.

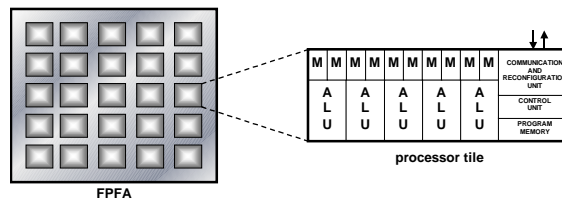


Figure 1: FPFA architecture.

The ALUs on a processor tile are tightly interconnected and are designed to execute the (highly regular) inner loops of an application domain. ALUs on the same tile share a control unit and a communication unit. The ALUs use the locality of reference principle extensively: an ALU loads its operands from neighboring ALU outputs, or from (input) values stored in lookup tables or local registers.

The FPFA concept has a number of advantages:

- The FPFA has a highly regular organisation, it requires the design and replication of a single processor tile, and hence the design and verification is rather straightforward. The verification of the software might be less trivial. Therefore, for less demanding applications we use a general-purpose processor core in combination with a FPFA.
- Its scalability stands in contrast to the dedicated chips designed nowadays. In FPFAs, there is no need for a redesign in order to exploit all the benefits of a next generation CMOS process or the next generation of a standard.

- The FPFA can do media processing tasks such as compression/decompression efficiently. Multimedia applications can for example benefit from such energy-efficient compression by saving (energy-wasting) network bandwidth.

3.1 Typical FPFA algorithms

In this section we will study several widely used algorithms. These algorithms provide a means to bring together a set of requirements for the functionality of an FPFA-ALU. We will look into some algorithms from the digital signal-processing domain: linear interpolation, finite-impulse response filter and Fast Fourier Transform. Other algorithms of our interest include Viterby decoding, Turbo decoding and various computer graphics algorithms.

Linear interpolation

It may be convenient (and energy efficient) to use tables for difficult calculations like square roots, division and sine. In order to keep the energy consumption low, such a table should be kept local and small. Interpolation between the function values from a table can be used to approximate a function value that is not in the table. The algorithm works as follows. For a value x ($x_0 \leq x < x_1$), two surrounding function values $F(x_0)$ and $F(x_1)$ are looked up in a table. With these values the in between function value $F(x)$ is calculated with:

$$F(x) = [F(x_1) - F(x_0)] \times x_{fraction} + F(x_0) \quad (x_0 \leq x < x_1) \quad (1)$$

$x_{fraction} = (x - x_0) / (x_1 - x_0)$ determines the distance of x relative to x_0 and x_1 . For example $x_{fraction}$ is $1/2$ if x is exactly between x_0 and x_1 . In many applications the $x_{fraction}$ values are known constants. The algorithm for linear interpolation is shown in Figure 2.

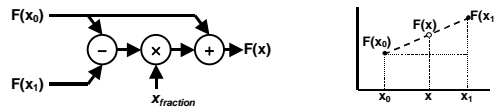


Figure 2: Linear interpolation.

Finite-impulse response filter

The finite-impulse response filter (FIR) is a frequently used algorithm in digital signal processing applications. Figure 3 shows two implementations of a 4-tap FIR filter.

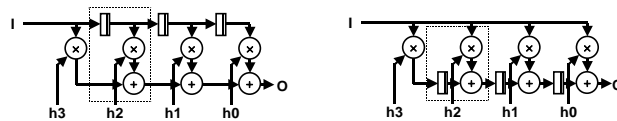


Figure 3: FIR filter.

Each section contains a multiplication, an addition and a delay. As shown in Figure 3, the left FIR filter uses three inputs and two outputs for every tap. The right FIR filter uses three inputs and one output. However, in the latter case, the input is routed to more than one tap.

Fast Fourier Transform

Fourier transform enables the conversion of signals from the time domain to the frequency domain (and vice versa). For digital signal processing, we are particularly interested in the Discrete Fourier Transform (DFT). The Fast Fourier Transform (FFT) can be used to calculate a DFT efficiently. FFT recursively divides a DFT into smaller DFTs. Eventually only basic DFTs remain. These DFTs have a number of inputs that is equal to the radix of the FFT. This is illustrated in Figure 4 for a radix 2 FFT with N=8 input signals.

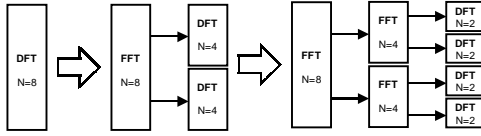


Figure 4: Recursion of a radix 2 FFT with 8 inputs.

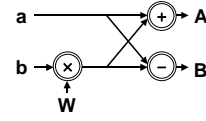


Figure 5: The radix 2 FFT butterfly.

$$\begin{aligned} A &= a + W \times b \equiv (a_{re} + a_{im}) + ((W_{re} \times b_{re} - W_{im} \times b_{im}) + (W_{re} \times b_{im} + W_{im} \times b_{re}))_{im} \\ B &= a - W \times b \equiv (a_{re} + a_{im}) - ((W_{re} \times b_{re} - W_{im} \times b_{im}) + (W_{re} \times b_{im} + W_{im} \times b_{re}))_{im} \end{aligned} \quad (2)$$

The resulting basic DFTs can be calculated by a structure called a *butterfly*. The butterfly is the basic element of a FFT. Figure 5 depicts the radix 2 butterfly; **a** and **b** are complex inputs and **A** and **B** are complex outputs. **W** is a complex constant called the *twiddle factor*. The radix 2 butterfly consists of a complex multiplication, a complex addition and a complex subtraction.

The FFT butterfly depicted in Figure 5 can be written as Equation (2). A hardware algorithm for the radix 2 FFT butterfly has six inputs ($a_{re}, a_{im}, b_{re}, b_{im}, W_{re}, W_{im}$) and four outputs ($A_{re}, A_{im}, B_{re}, B_{im}$). Each input is used two times. Three subtraction, four multiplication and three addition operations are used.

3.2 ALU datapath design

After investigating the functionality required by an application domain, a suitable ALU can be designed. Based upon [9] and [7], we introduce an FPFA-ALU that can be used to implement the algorithms discussed in the previous section. This ALU – which is depicted in Figure 6 – has 4 inputs (**a**, **b**, **c**, **d**) and 2 outputs (**O1**, **O2**). The input operands are all 16-bit, 2-complement numbers. The internal ALU datapaths are either 20 or 40 bits wide.

As can be seen in Figure 6, three different levels can be distinguished in the ALU:

- *Level one* is a reconfigurable function block. The function blocks f_1, f_2 and f_3 in Figure 6 each can perform five operations: add, subtract, absolute value, minimum and maximum. The result of level one is:

$$Z1 = f_3(f_1(a,b), f_2(c,d))$$

This means that most functions with four operands and five operations can be performed, e.g. $Z1 = abs((a+b) - max(c,d))$. The internal operands have a width of 19-bits + a sign-bit.²

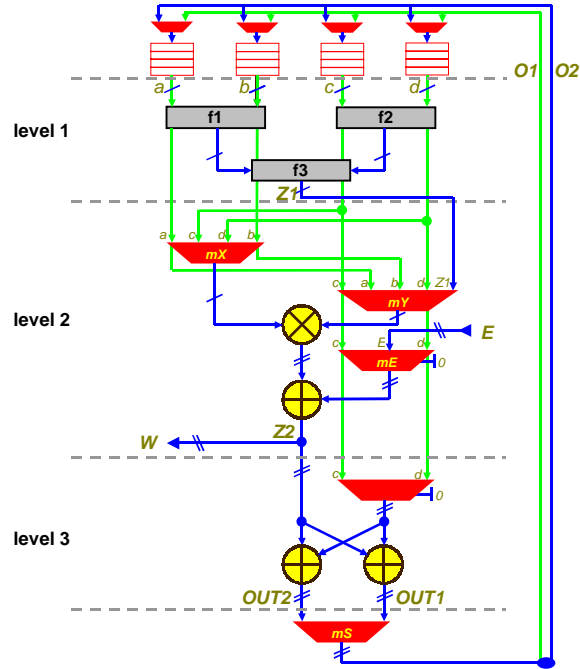


Figure 6: ALU datapath design.

- Level two contains a 19×19-bit unsigned multiplier and a 40-bits wide adder. The multiplier result is thus 38-bits + a sign-bit. An East-West interconnect connects neighbouring ALUs, which makes multiple precision arithmetic possible. A value on the East input can be added to the multiplier result. The result of a multiply-add (Z2) can be used as input for level 3 and for the ALU connected to the West output. We can represent the behaviour of level two as:

$$Z2 = m_x[a|b|c|d|nop] \times m_y[a|b|c|d|Z1|nop] [+|-] m_e[0:c|c:0|0:d|d:0|c:d|E|0|nop]$$

m_x , m_y and m_e are multiplexers (the output is one of its parameters and a colon denotes concatenation). In algorithms that do not use a multiply-add operation, the *nop* options can be used to bypass level 2.

- Level three can be used as a 40-bit wide adder or as a butterfly structure:

$$OUT2 = m_o[0:c|0:0:c|d:0|0:d|c:d|nop] - Z2$$

$$OUT1 = m_o[0:c|0:0:c|d:0|0:d|c:d|nop] + Z2$$

OUT2 and *OUT1* are both 40-bit wide.

² The carry signals are propagated to neighbouring ALUs to support extended precision of the operands. For clarity reasons these carry signals are *not* depicted in Figure 6.

The final multiplexer (m_3) selects which two 20-bit results are written back into the registers via its outputs **O1** and **O2**.

3.3 Processor tile datapath design

A FPFA processor tile in Figure 1 consists of five identical blocks, which share a control unit and a communication unit. An individual block contains an ALU, two memories and four register banks of four 20-bit wide registers. Because of the locality of reference principle, each ALU has two local memories. Each memory has 256 16-bit entries. A crossbar-switch makes flexible routing between the ALUs, registers and memories possible. Figure 7 shows the crossbar interconnect between five blocks. This interconnect enables an ALU to write-back to any register or memory within a tile.

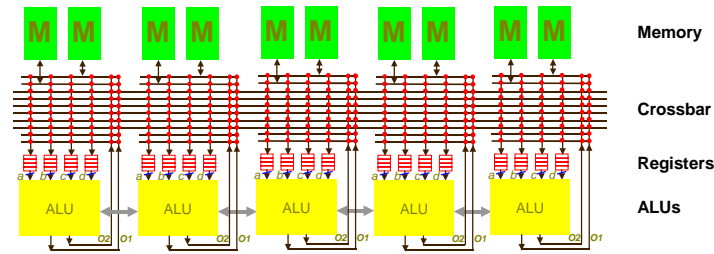


Figure 7: Crossbar-switch.

Five blocks per processor tile seems reasonable. With five blocks there are ten memories available. This is convenient for the FFT algorithm, which has six inputs and four outputs. Also, we now have the ability to use $5 \times 16 = 80$ -bit wide numbers, which enable us to use floating-point numbers (although some additional hardware is required). Some algorithms, like the FIR filter, can benefit substantially from additional ALUs. With five ALUs, a five-tap FIR filter can be implemented efficiently. The fifth ALU can also be used for complex address calculations and other control purposes.

3.4 Mapping of algorithms

The FPFA-ALU has been designed with well-know DSP algorithms in mind. As an example, we will show how the algorithms presented in Section 3.1 can be mapped on the FPFA.

Linear interpolation

The linear interpolation algorithm of equation (1) can be mapped to the ALU depicted in Figure 6. Assume that operands $F(x_0)$, $F(x_1)$, and $x_{fraction}$ are available at inputs **d**, **c** and **a** respectively. Then $F(x) = [c - d] \times a + d$, which is calculated with the following behaviour:

$$\begin{aligned} Z1 &= c-d \\ Z2 &= a \times Z1 + 0:d \\ OUT1 &= Z2 \end{aligned}$$

Finite-impulse response filter

Figure 8 shows the implementation for a five-tap finite-impulse response filter. When the FIR filter is started, every clock cycle an additional ALU is used, until all five ALUs are in use.

Thus, the first result is ready after five clock cycles. Every subsequent cycle returns the next result.

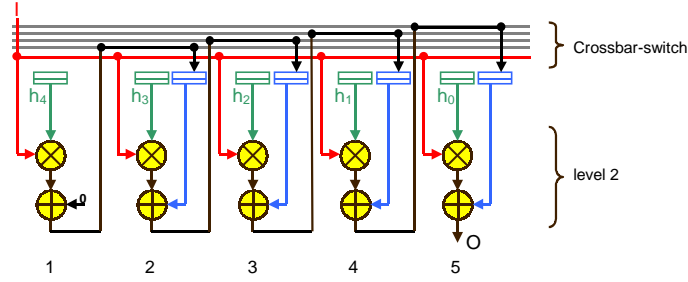


Figure 8: Five-tap finite-impulse response filter.

Fast Fourier transform

Figure 9 shows how the FFT butterfly of Equation (2) can be calculated on four FPFA-ALUs.

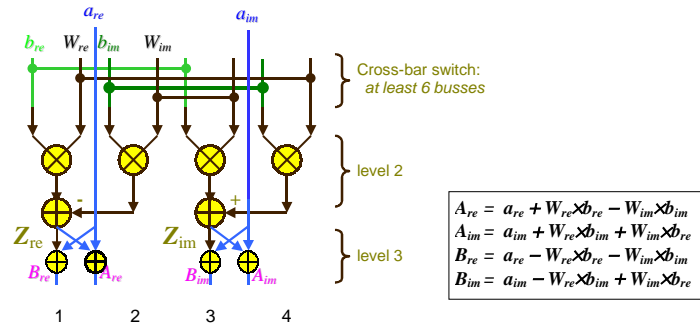


Figure 9: The radix 2 FFT butterfly.

4 Discussion and related work

We intend to implement an energy-efficient mobile system based on the (reconfigurable) architecture presented in Section 2. Consequently, a whole new QoS model that can balance issues like energy budget, required performance and communication costs is required. We are also working on operating system extensions for energy efficiency. The operating system has to provide the flexibility required by the QoS model and it must exploit the flexibility of the systems architecture. The FPFA is part of the systems architecture. We believe that the architecture of the FPFA implies a good energy-efficiency. The minimization of the energy consumption of an architecture like the FPFA requires that the energy consumption due to arithmetic (logic), communication (wiring) and data storage (RAM) be balanced. It is shown in [8] how these concepts can be balanced to find a lower bound for the energy consumption of the FFT. We applied the design concepts again to design the FPFA such that it is energy efficient for a wide class of signal processing applications. In our current research, we are investigating the energy efficiency of various algorithms when executing on a FPFA architecture. We plan

to build an experimental chip after we have demonstrated the efficiency of the architecture and evaluated various architectural alternatives. The interconnection network between the processor tiles is another topic of research.

To support a wide range of algorithms requires a lot of flexibility in the ALU. This has particular impact on level one of the FPFA-ALU datapath. The control of level one would require 15 control bits per ALU. To save on control bits, and to have more flexibility, we plan to implement the level one functionality by a reconfigurable block that is similar to Xilinx CLBs [11]. This also gives us the possibility to do bit-level logic functions. Having a reconfigurable block in level one means that the functionality of the ALU becomes partly reconfigurable. Note that such a reconfigurable block must be configured before the ALU can be used.

Unfortunately, space restrictions restrain us from giving a literature overview of the impetuous area of reconfigurable systems-architecture. However, we do want to mention the *Raw architecture* [10] of MIT and the *Imagine stream architecture* [6]. Raw has a mesh of identical processor tiles. Each tile has a tile processor, a static switch processor and a dynamic router. The tile processor is related to a 32-bit MIPS processor. The RAW group abandoned the idea to include reconfigurable logic in their processor tile, mainly because of the added complexity of the reconfigurable logic, its interface, and the software interface.

We distinguished three vertical levels in the ALU datapath. There are also architectures that use a horizontal approach, such as the Imagine stream architecture. This architecture has clusters of functional units. One cluster can for example contain three adders, two multipliers, one divide/square root unit, one 128-entry scratch-pad register file and one communication unit. All functional units in a cluster are connected to busses and can operate in parallel. In this architecture a multiply-add is performed by first completing a multiply and subsequently sending the result to an adder. So, the delay of a multiply-add is equal to the sum of the delay of a multiply and the delay of an addition. In our ALU, the multiplication and additions of level two and three can be done largely in parallel. This is illustrated in Figure 10. The lesser significant bits of the result of a multiplier are known (or stable) prior to the more significant bits. As soon as the next bit of the multiplication result is known, the adder can calculate the next bit of its result. Clearly, the delay increase of a multiply-add in a leveled ALU is smaller than the delay of an individual addition.

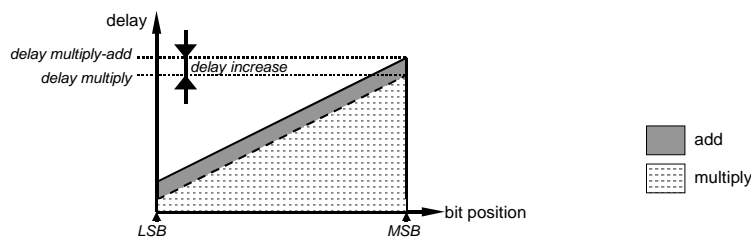


Figure 10: Delay increase for vertical leveled multiply-add operation.

A vertically leveled ALU has a relatively long signal propagation time, which implies a relatively slow clock. The slower clock is a tradeoff; a vertically leveled ALU requires less clock ticks for particular operations (like the multiply-add) than a flat ALU. We showed above, that the FPFA-ALU can perform very complex functions in just one clock-tick.

The multiply-add functionality of level two of the ALU is not required for every algorithm (e.g. Viterby decoding). An unused multiplier consumes chip area and might even consume

some energy. However, if we omit the multiplier, a severe performance penalty has to be paid for the algorithms that do require a multiplication. In future one billion transistor chip designs, the area of a multiplier is small compared to the area used for communication and local memory. Therefore, we believe that the benefits outweigh the penalty and that a multiply-add unit should be integrated in the FPFA-ALU. However, when not used, the multiply-add unit should not consume much energy.

5 Conclusion

The introduced FPFA architecture is aimed at fine-grained operations in hand-held multimedia computers. The architecture has a low design complexity, is scalable and can execute various algorithms energy efficiently, while maintaining a satisfactory level of performance. Several non-trivial algorithms have been mapped successfully on the FPFA processor tile. Examples from the digital signal-processing domain include linear interpolation, FIR filter and FFT.

In contrast to the FPFA, which is aimed at fine-grained 16-bit wide operations, the FPGA is aimed at bit level logic functions. As a consequence, operations like multiply-add are relatively expensive on a FPGA (i.e. they take a large area and thus consume a large amount of energy).

The ALU of a FPFA processor tile has four input operands. Most standard ALUs have two input operands. The extra inputs increase the functionality of the ALU and enable it to compute powerful functions like the linear interpolation efficiently.

The FPFA design has still many open issues, which include the control of a processor tile and the interconnect between processor tiles.

References

1. Burger D., Goodman J., "Billion-Transistor Architectures", *IEEE Computer*, September 1997.
2. Havinga P.J.M., "Mobile Multimedia Systems", *Ph.D. thesis, University of Twente*, Februari 2000.
3. Kuse E.A., "Analysis and circuit design for low power programmable logic modules", *Ms. Thesis, University of California at Berkeley*, December 1997.
4. Lettieri P., Srivastava M.B., "Advances in wireless terminals", *IEEE Personal Communications*, pp. 6-19, February 1999.
5. Mehra R., Rabaey J., "Exploiting regularity for low-power design", *proceedings of the international Conference on computer-aided design*, 1996.
6. Rixner S., Dally W.J., et al, "A Bandwidth-Efficient Architecture for Media Processing", *Micro-31*, 1998.
7. Smit J., et al, "Low Cost & Fast Turnaround: Reconfigurable Graph-Based Execution Units", *Proceedings Belsign Workshop*, 1998.
8. Smit J., Huijsken, J.A, "On the energy complexity of the FFT", *Proceedings of the Patmos Conference, Oldenberg, Germany*, 1995.
9. Stekelenburg M., "Optimization of a Field Programmable Function Array", *Ms. thesis, University of Twente*, March 1999.
10. Taylor M., "The Raw Prototype Design Document", *Massachusetts Institute of Technology*, V1.3, February, 2000.
11. Xilinx, "Virtex[™]-E 1.8V Field Programmable Gate Arrays", *Advance Product Specification*, December, 1999.