Exploring Energy-Efficient Reconfigurable Architectures for DSP Algorithms

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

Departments of Computer Science and Electrical Engineering,
University of Twente, Enschede, the Netherlands
email:heysters@cs.utwente.nl

Abstract— **Future hand-held multimedia terminals require a very high performance on a very small energy budged. Such devices can only be realized if their entire system is energy cognisant. In this paper a** *reconfigurable systems-architecture* **for mobile multimedia systems is introduced. The** *Field Programmable Function Array* **(FPFA) is discussed in detail. Several digital signal processing algorithms are discussed and mapped on the FPFA. Also, the** *power radius* **theory about low-power design is introduced.**

Keywords— **Energy efficiency; FPFA; power radius; mobile multimedia systems.**

I. 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 and are small in size, but require a performance that exceeds the level 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. Research has shown that adapting continuously the system and protocols can significantly improve the energy efficiency while maintaining a satisfactory level of performance [5].

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 [4]. 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 solu-

tion 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.

II. 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 [6]. 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 [4].

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 [5].

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.) [3].

- *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.

We intend to implement an energy-efficient mobile system based on this (reconfigurable) architecture. Consequently, a whole new QoS model that can balance issues like energy budget, required performance and communication costs is required [8]. 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. 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.

III. 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. Fig. 1 shows a FPFA with 25 tiles; each tile has five ALUs.

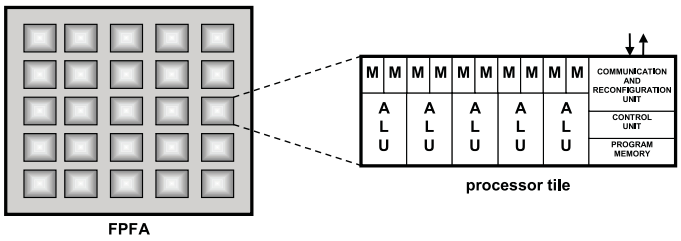

Fig. 1. FPFA architecture.

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.

A. Processor tile

A FPFA processor tile in Fig. 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 20-bit entries. A crossbar-switch makes flexible routing between the ALUs, registers and memories possible. Fig. 2 shows the crossbar interconnect between five blocks. This interconnect enables an ALU to write-back to any register or memory within a tile.

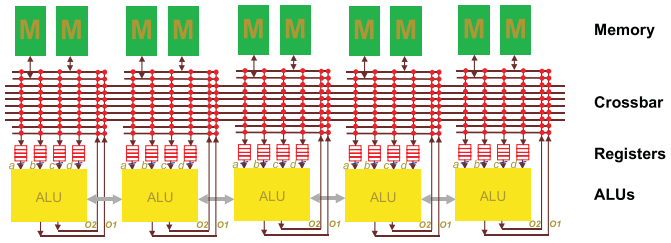

Fig. 2. Crossbar-switch.

Five blocks per processor tile seems reasonable. With five blocks there are ten memories available. This is convenient for the Fast Fourier Transform algorithm, which has six inputs and four outputs (see Section IV-C). Also, we now have the ability to use $5 \cdot 16 = 80$-bit wide numbers, which enable us to use floating-point numbers (although some additional hardware is required). Some algorithms, like the Finite impulse-response filter (Section IV-B) , 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.

B. ALU datapath

The datapath of the FPFA-ALU is depicted in Fig. 3. The ALU has 4 inputs (a, b, c, d) and 2 outputs $(OUT1, OUT2)$. The in- and outputs are 20-bit wide and use a sign-magnitude representation. In algorithms that work on small numbers, the sign-magnitude representation is more energy efficient than the 2-complement representation. In 2-complement the transition from a small positive number to a small negative number causes a lot of bit reversals. The internal datapaths are either 20 or 40-bit wide. The internal data representation is signed-magnitude for the multiplier and 2-complement for the adders. The datapath shown is only suitable for integer arithmetic. The FPFA-ALU will also support fixed point and cascaded arithmetic, but this is not discussed here.
In the ALU three different levels can be discriminated. These three levels require in total 33 control signals. We will now discuss the behaviour of each level.

B.1 Level one

Level one is a reconfigurable function block. Each function f_1, f_2 and f_3 in Fig. 3 returns the following result:

$$\begin{aligned} f_n \;=\;& 0 \\ &\mid\; [abs]\ [-]Op_{left} \\ &\mid\; [abs]\ [-]Op_{right} \\ &\mid\; [abs]\ +([-]Op_{left}, [-]Op_{right}) \\ &\mid\; [abs]\ min([-]Op_{left}, [-]Op_{right}) \\ &\mid\; [abs]\ max([-]Op_{left}, [-]Op_{right}) \end{aligned} \tag{1}$$

The result of level one is:

$$Z1 \;=\; f_3(f_1(a,b), f_2(c,d)) \tag{2}$$

B.2 Level two

Level two contains a $19 \cdot 19$-bit unsigned multiplier and a 40-bit wide adder. The *east-west interconnect* connects neighbouring ALUs. A value in the *east* input can be added to the multiplier result. The result of level two ($Z2$) is used as input for level 3 and for the ALU connected to the *west* output. We can express the result of level two as:

$$\begin{aligned} Z2 \;=\;& Z1 \\ &\mid\; +(a|b|c|d \cdot a|b|c|d|Z1, [-]\ 0|c_{se}|d_{se}|east) \end{aligned} \tag{3}$$

The *se* subscript means that the numbers are *sign extended*. Note that it is possible to bypass level two.

B.3 Level three

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

$$\begin{aligned} o2 \;&=\; +(0|c_{se}|d_{se}|cd, -Z2) \\ o1 \;&=\; +(0|c_{se}|d_{se}|cd, Z2) \end{aligned} \tag{4}$$

Two 20-bit words can be selected as the final result of the ALU:

$$\begin{aligned} out2 \;&=\; o1_{high} \mid o1_{low} \mid o2_{high} \mid o2_{low} \\ out1 \;&=\; o1_{high} \mid o1_{low} \mid o2_{high} \mid o2_{low} \end{aligned} \tag{5}$$

IV. Mapping of DSP algorithms on the FPFA

This section shows how several widely used algorithms from the digital signal-processing domain can be mapped on the FPFA-ALU structure presented above. Below, mapping for linear interpolation, finite-impulse response filter and Fast Fourier Transform are given. Other algorithms of our interest include Viterby decoding, Turbo decoding [10] and various computer graphics algorithms.

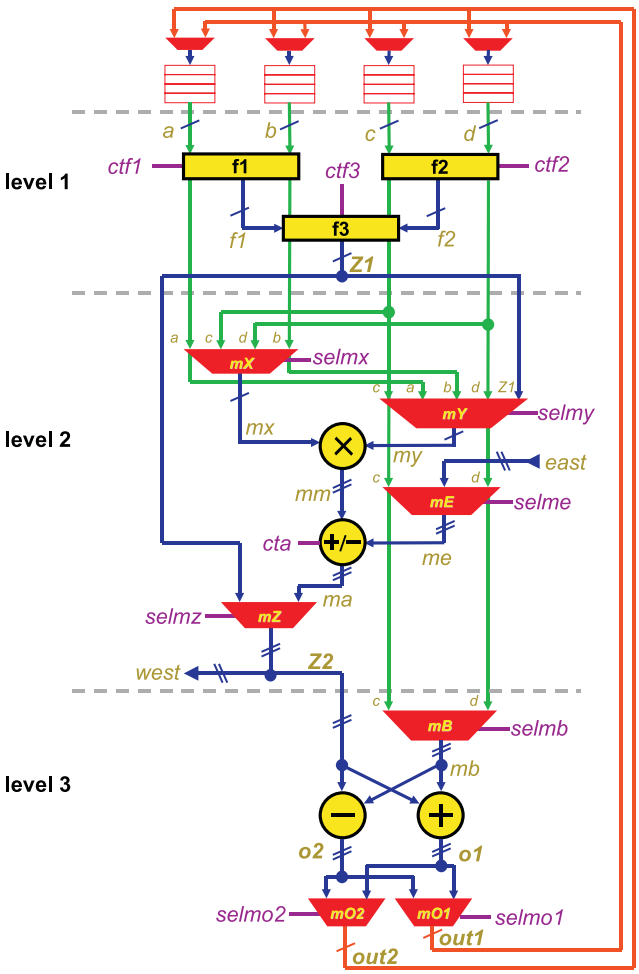

Fig. 3. ALU datapath.

A. 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 (with $x_0 \leq x < x_1$), 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:

$$f(x) \;=\; \Big(f(x_1) - f(x_0)\Big) \cdot x_{fraction} + f(x_0) \tag{6}$$

where: $x_0 \leq x < x_1$ and $x_{fraction} = (x - x_0)/(x_1 - x_0)$. $x_{fraction}$ determines the distance of x relative to x_0 and x_1. For example $x_{fraction}$ is $\frac{1}{2}$ 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 Fig. 4.

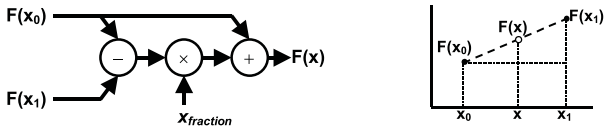

Fig. 4. Linear interpolation.

The algorithm of Eq. 6 can be mapped to the ALU depicted in Fig. 3. Assume that operands $f(x_0)$, $f(x_1)$ and $x_{fraction}$ are available at inputs d, c and a respectively. Then $f(x) = (c - d) \cdot a + d$, which is calculated with the following behaviour:

$$\begin{aligned} Z1 &= c - d \\ Z2 &= a \cdot Z1 + d \\ O1 &= Z2 \end{aligned}$$

B. Finite impulse-response filter

The finite-impulse response filter (FIR) is a frequently used algorithm in digital signal processing applications. Fig. 5 shows the *direct form* (left) and the *transposed direct form* (right) implementation of a 4-tap FIR filter. In finite precision implementations, it is better to use the transposed direct form. Mathematically a FIR filter can be expressed as:

$$O_j = \sum_{n=0}^{N-1} h_{N-n-1} \cdot I_{j-n} \qquad (7)$$

where:

I	:	input symbol.
O	:	output symbol.
N	:	number of taps.
j	:	discrete time.

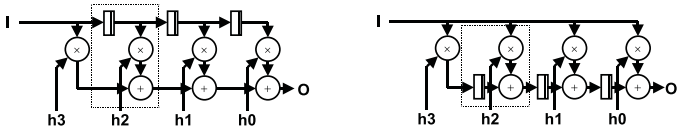

Fig. 5. FIR filter.

Fig. 6 shows the transposed implementation for a five-tap FIR 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.

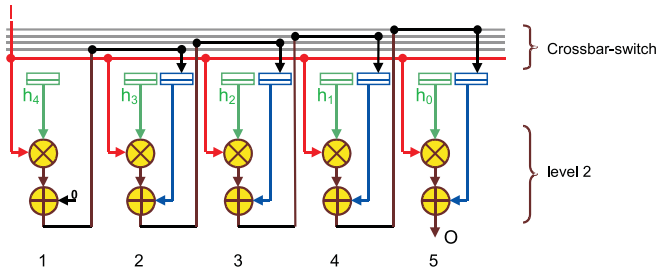

Fig. 6. Mapping of a five-tap FIR filter.

C. 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 Fig. 7 for a radix 2 FFT with $N = 8$ input signals.

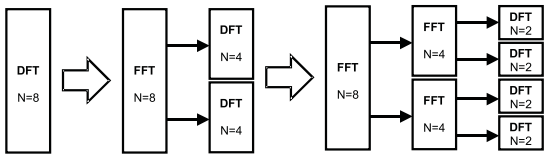

Fig. 7. Recursion of a radix 2 FFT with 8 inputs.

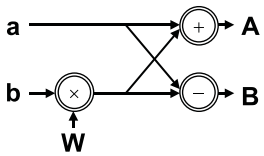

Fig. 8. The radix 2 FFT butterfly.

$$\begin{aligned} A &= a + W \cdot b \\ &\equiv (a_{re}+a_{im}) + ((W_{re} \cdot b_{re} - W_{im} \cdot b_{im})_{re} + (W_{re} \cdot b_{im} + W_{im} \cdot b_{re})_{im}) \\ B &= a - W \cdot b \\ &\equiv (a_{re}+a_{im}) - ((W_{re} \cdot b_{re} - W_{im} \cdot b_{im})_{re} + (W_{re} \cdot b_{im} + W_{im} \cdot b_{re})_{im}) \end{aligned} \qquad (8)$$

The resulting basic DFTs can be calculated by a structure called a *butterfly*. The butterfly is the basic element of a FFT. Fig. 8 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 Fig. 8 can be written as Eq. 8. 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. Fig. 9 shows how the FFT butterfly of Eq. 8 can be calculated on four FPFA-ALUs.

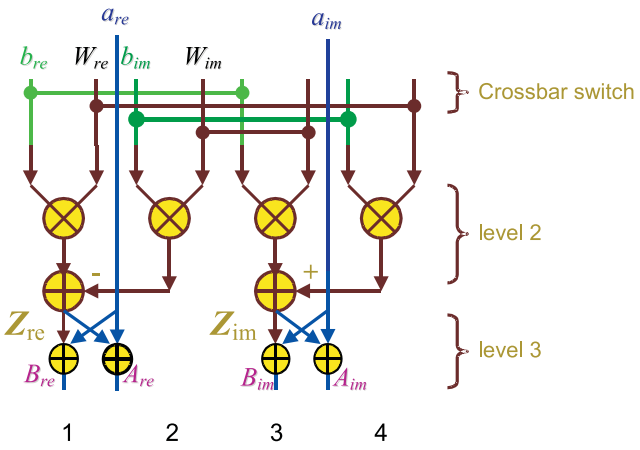

Fig. 9. Mapping of the radix 2 FFT butterfly.

$$\begin{aligned} A_{re} &= a_{re} + W_{re} \cdot b_{re} - W_{im} \cdot b_{im} \\ A_{im} &= a_{im} + W_{re} \cdot b_{im} + W_{im} \cdot b_{re} \\ B_{re} &= a_{re} - W_{re} \cdot b_{re} - W_{im} \cdot b_{im} \\ B_{im} &= a_{im} - W_{re} \cdot b_{im} + W_{im} \cdot b_{re} \end{aligned} \tag{9}$$

V. Low-power design

The FPFA processor tile is being designed with DSP algorithms like the ones discussed above in mind. From the study of these application domain specific algorithms, we can determine the *functionality* that should (at least) be supported by the processor tile. A mere functional specification is not enough to design a FPFA processor tile. There are other concerns as well. In this section we will discuss some of the concepts used in the design of the FPFA.

A. Balanced energy consumption

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. We will illustrate this by introducing the *power radius* concept. Later, we will show what impact the power radius has on the design of the FPFA processor tile.

A.1 Power radius

Most components are currently fabricated using CMOS technology. Main reasons for this bias is that CMOS is cost efficient and inherently lower power than other technologies. The sources of energy consumption on a CMOS chip can be classified as *static* and *dynamic* power dissipation. The dominant component of energy consumption (85 to 90 %) is dynamic [3]. The *dynamic energy consumption* $\mathcal{E}_{/m}$ due to a change in the input signal of a wire with a length of one meter is given by:

$$\mathcal{E}_{/m} = \frac{1}{2} \cdot C_{/m} \cdot V_{DD}^2 \tag{10}$$

where:

$C_{/m}$	:	wire capacitance per meter.
V_{DD}	:	supply voltage.

In many CMOS-VLSI systems much energy is dissipated in the wires connected to the terminals of a cell. The quest to keep the amount of energy dissipated in wires to a minimum has led us to the *power radius* model for low power design. The power radius gives a method to calculate the power dissipation in a VLSI design without knowledge of the average switching frequency of arithmetic elements and the associated interconnect.

Power radius $\mathcal{R}_{pd}$*:* the external wire radius of a cell at which the average dynamic energy consumed to drive the capacitance of all external wires equals the average dynamic energy consumed to execute the cells function.

$$\mathcal{R}_{pd} = \frac{\mathcal{E}_f}{\sum_{1}^{n} \alpha_n \cdot \mathcal{E}_{/m}} \tag{11}$$

where:

$\mathcal{E}_f$	:	average dynamic energy consumption to execute the cells function.
$\mathcal{E}_{/m}$	:	dynamic energy consumed to drive the capacitance of the cells external wiring per meter wire. $\mathcal{E}_{/m}$ is assumed to be equal for all wires.
n	:	number of external wires of the cell.
α_n	:	probability that a transition occurs on external wire n.

The power radius is defined in terms of the average switching energy in both the cell and its wiring. In typical applications, it is reasonable to assume that the wires of cells like adders and multipliers switch with a probability of $\frac{1}{2}$. If we assume that the switching probability $\alpha_n = \frac{1}{2}$ for every n, then we can simplify Eq. 11 to:

$$\mathcal{R}_{pd} = \frac{\mathcal{E}_f}{\frac{1}{2} \cdot n \cdot \mathcal{E}_{/m}} \tag{12}$$

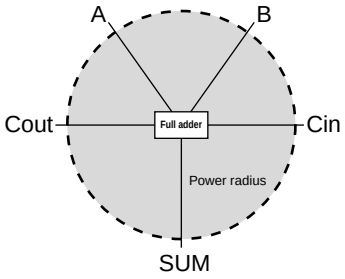

Fig. 10. Power radius.

Fig. 10 illustrates the power radius concept. Consider a full adder realization in a 1 μm process at 5V, with an average energy consumption of 2.56 pJ and a $\mathcal{E}_{/m}$ of 1.44×10^3 pJ/m. The power radius of this adder is:

$$\begin{aligned} \mathcal{R}_{pd} &= \frac{2.56\ pJ}{5 \cdot \frac{1}{2} \cdot 1.44 \times 10^3\ pJ/m} \\ &= 0.714 \times 10^{-3}\ m \end{aligned}$$

The power radius of 0.714 mm is about 10 times the height and 30 times the width of the full adder in the example. We will explain below, how the power radius can be used to identify an energy efficient layout of a datapath in which the full adder is used.
The energy consumption of small arithmetic blocks, like the full adder, has to be determined before the power radius can be calculated. This energy consumption can for example be obtained by detailed computer simulations. The energy consumption of small arithmetic blocks can be used to calculate the energy consumption of larger building blocks, like adders, multipliers, ALUs and application specific ALUs. The introduction of a new process may result in a more energy efficient realization of the small arithmetic blocks. The determination of the energy consumption of the small arithmetic blocks may therefore be repeated when a new process is introduced. It serves as a point of calibration for the *arithmetic energy complexity* of a design.

A.2 Arithmetic energy complexity

Arithmetic energy complexity: the lower bound of the average amount of energy dissipated to execute all arithmetic operations of an algorithm.

Note that the arithmetic energy complexity does not include the amount of energy needed to execute data transfers.
The arithmetic energy complexity gives a lower bound for the amount of energy consumed by the datapath of an algorithm. If a layout for a datapath can be identified that interconnects the arithmetic cells without introducing much extra wire length - e.g. less than 5% of the power radius -, then the datapath may execute the algorithm at energy dissipation levels close to the arithmetic energy complexity. Consider a datapath in which the full adder from Fig. 10 is used. If we want to remain within 10% of the power radius $\mathcal{R}_{pd}$, then the *accumulated length* of the external wires should not exceed 0.071 mm. In summary: the power radius enables to design layouts for energy efficient datapaths. That is to say, that almost all of the energy dissipated in a datapath is due to the actual calculation.

A.3 Random access power radius

The long-term storage of data consumes a negligible amount of energy. To transport data to and from the location where it is stored, does take a finite amount of energy. This energy consumption can be considered as a special case of energy consumption due to wiring. The *random access power radius* can be used to determine a power-optimal partition of processing elements and memories like register banks, SRAMs or DRAMs.

Random access power radius $\mathcal{R}_{RAM}$*:* the radius of a cell at which the average dynamic energy consump-

tion involved in the transport to or from a storage equals the average energy consumed to execute the cells function.

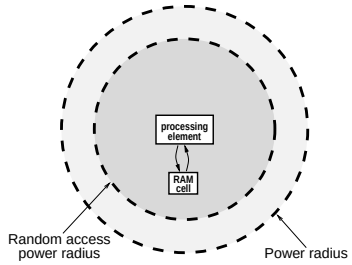

Fig. 11. Random access power radius.

The random access power radius is depicted in fig. 11. The power radius is a property of a cell and is dependent of the cell. The random access power radius is dependent of both the power radius and a specific storage. If a storage element itself does not introduce any additional energy dissipation, then the random access power radius of the cell that accesses this storage element equals its power radius. The random access power radius can be expressed in terms of the power radius:

$$\mathcal{R}_{RAM} = \mathcal{R}_{pd} \cdot \eta_{acc} \cdot \eta_{ov} \tag{13}$$

where:

η_{acc}	:	access efficiency of the memory system.
η_{ov}	:	overhead related to the application of the memory system.

The *access efficiency* η_{acc} of a random access storage system is dominated by the energy dissipation introduced by state transitions on the bit-lines used to access the individual storage cells. Another point of concern is for the access efficiency is the capacitive load of access transistors on the bit-lines. Typical values for η_{acc} range from $\frac{1}{8}$ to $\frac{1}{2}$. The need to supply address information causes extra overhead. The associated efficiency is:

$$\eta_{ov} \approx \frac{w}{log^2(N) + w} \tag{14}$$

where:

N	:	number of ram cells.
w	:	width of the databus.

A.4 Power radius conclusion

The power radius concept considers the energy required for the arithmetic operations as invariant. However, on the data transfers energy can possibly be saved. Applying the concept allows to keep the power dissipation due to wiring within acceptable boundaries. The storage of data does not contribute to the arithmetic complexity of an algorithm. Storage elements should therefore be kept local and small. The concept of power radius can be used through the whole design hierarchy. This can be broken down into the following levels:

- individual transistors within a cell,
- cells within an arithmetic primitive,
- arithmetic primitives within a module,
- modules on a chip and
- chips on a printed circuit board.

B. FPFA design

The ALU described in Section III-B is constituted by three vertically stacked levels. Instead of stacking the arithmetic logic blocks, they could be arranged in a more horizontal way. They could for example be connected via a bus, like in [9]. Consider a multiply-add in such a horizontal architecture. In order to perform the multiply-add, the result from a multiplier has to be passed to an adder via the bus. The power radius of a single adder or multiplier is fairly small. It is very unlikely that the length of a bus is within the power radius of such a cell. However, the power radius of an entire ALU might justify access to a (short) bus. Similarly, the random access power radius of an entire ALU justifies the use of larger memories.
In a combinatorial datapath, there is another advantage in having adders below a multiplier. In our ALU, the multiplication and additions of level two and three can be done largely in parallel. This is illustrated in Fig. 12. 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.

A vertically leveled ALU has a slightly longer signal propagation time, which implies a relatively slow clock. We showed above, that the FPFA-ALU can

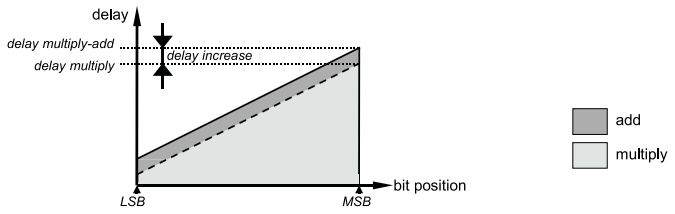

Fig. 12. Delay increase for vertical leveled multiply-add operation.

perform very complex functions in just one clock-tick The multiply-add functionality of level two of the ALU is not used in every algorithm (e.g. Turbo 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. When the multiplier of an ALU is disabled, it serves no function anymore. However, it does still take space. In effect, the external wires of some function cells have become longer. But even in this case, it can be expected - depending on the algorithm executed - that the ALU performs with an energy efficiency that is only a few factors worse than the arithmetic energy complexity. 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 will not have any dynamic energy consumption. Note, that the critical delay path of the ALU becomes shorter if the multiplier is disabled. Therefore, reconfigurable clock frequencies or asynchronous design methods should also be explored.

Each local memory in a FPFA processor tile has a size of 256 20-bit words. If energy efficient SRAM is used, then these memories will fit within the random access power radius of a FPFA-ALU. The small memories are a tradeoff between energy consumption by calculation and energy consumption by communication. To illustrate this, consider the implementation of a sine function. We could put a pre-calculated sine in a table in memory. This memory would be large and thus consume a lot of area and energy. In stead, we could calculate the entire sine. The required memory is minimized, however a lot of calculations need to be done. It is better to put a course sine table in a small memory and use linear interpolation to approximate the values which are not in the table.

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. To increase the flexibility of level one, the level one functionality could be implemented by a reconfigurable block that is similar to Xilinx Configurable Logic Blocks [11]. This also gives us the possibility to do bit-level logic functions. Note that such a reconfigurable block must be configured before the ALU can be used. We intend to reduce the number of control bits by using configuration caches, like in [2].

VI. Summary and conclusion

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. 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, communication and data storage be balanced.

The 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 well known *FPGA*, which is aimed at bit level logic functions, the FPFA is aimed at fine-grained word operations. We expect that on a FPGA operations like multiply-add are more expensive than on a FPFA (i.e. they take a larger area and as a consequence consume a larger 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-ALU is compact, energy efficient and has a good performance.

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

Acknowledgement

This research is supported by the PROGram for Research on Embedded Systems & Software (PROGRESS) of the Dutch organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the technology foundation STW.

References

[1] Burger D., Goodman J., "Billion-Transistor Architectures", *IEEE Computer*, September 1997.

[2] Hauser J.R., Wawrzynek J., "Garp: A MIPS Processor with a Reconfigurable Coprocessor", *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 16-18 1997.

[3] Havinga P.J.M.,"Mobile Multimedia Systems", *Ph.D. thesis, University of Twente*, February 2000.

[4] Kuse E.A., "Analysis and circuit design for low power programmable logic modules", *Ms.Thesis, University of California at Berkeley*, December 1997.

[5] Lettieri P., Srivastava M.B., "Advances in wireless terminals", *IEEE Personal Communications*, pp. 6-19, February 1999.

[6] Mehra R., Rabaey J., "Exploiting regularity for low-power design", *Proceedings of the international conference on computer-aided design*, 1996.

[7] Smit J. et al, "Low Cost & Fast Turnaround: Reconfigurable Graph-Based Execution Units", *Proceedings Belsign Workshop*, 1998.

[8] Smit L.T. et.al., "Run-time Energy Management for Mobiles", *to be published in: Proceedings Progress 2000*, October 2000.

[9] Rixner S., Dally W.J., et al, "A Bandwidth-Efficient Architecture for Media Processing", *Micro-31*, 1998.

[10] Valenti, M.C., "Turbo codes and iterative processing", *Proceedings IEEE New Zealand Wireless Commun. Symp. '98*, Auckland New Zealand, November 1998.

[11] Xilinx, "Virtex^{tm}-E 1.8V Field Programmable Gate Arrays", *Advance Product Specification*,December 1999