

# Reconfigurable System Design: The Control Part

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

*University of Twente, Electrical Engineering / Computer Science*

*PO box 217, 7500 AE Enschede*

e-mail: [heysters@cs.utwente.nl](mailto:heysters@cs.utwente.nl)

**Abstract** – Advancement in low-power hand-held multimedia systems requires exploration of novel system architectures. In conventional computer architectures, an increase in processing power also implies an increase in energy consumption<sup>1</sup>. In a mobile system this results in a shorter operating-time. Limiting the generality of an architecture can improve its energy efficiency. However, application specific architectures have a number of drawbacks and are too restricted. Therefore, an *application domain* specific architecture is proposed. As part of an application domain specific architecture for the digital signal-processing domain, the *Field Programmable Function Array* (FPFA) was conceived. The FPFA is a reconfigurable device with a data-path that can be configured to implement a number of DSP algorithms energy efficiently [1]. The flexibility in the FPFA results in a data-path that requires many control signals. To reduce the vast amount of control signals, the FPFA control-path uses a combination of *configuration registers* and *vertical microprogramming*.

**Keywords** – reconfigurable computing, energy efficiency, mobile multimedia systems, dynamic reconfiguration

## I. INTRODUCTION

The design of low-power hand-held multimedia systems requires exploration of novel system architectures. The most flexible processing architectures are general purpose processors (GPPs), including the large class of digital signal processors (DSPs). To achieve performance for a wide range of applications, GPPs dedicate a substantial amount of die area to overhead such as speculative execution and branch prediction. These complex mechanisms can extract a moderate amount of parallelism but not the large amount available in many compute-intensive applications.

At the other end of the flexibility spectrum are the application specific integrated circuits (ASICs) which can be used to achieve higher performance at lower area

and energy cost than GPPs. High performance can be achieved since the architecture can be tailored for a specific application to extract the parallelism, while optimizing for power, speed or area. However the drawback of ASICs are their lack of flexibility, their high design cost and high design time. This makes them unattractive except for very well-defined and wide-spread applications.

Field programmable gate arrays (FPGAs) promised to bridge the flexibility and performance gap between GPPs and ASICs. FPGAs are more flexible than ASICs and they can exploit the parallelism in algorithms better than GPPs. Unfortunately, to implement arbitrary circuits, FPGAs have to be very fine-grained (the logic blocks must be small and regular) and the overhead of this generality can be expensive in both area and performance. While GPPs use highly optimized functional units that operate on long data words, FPGAs are only efficient for complex bit-oriented computations or complicated bit-level masking and filtering [2].

In the Chameleon project [4], a coarse grain reconfigurable architecture is defined, as we believe a reconfigurable architecture might be the key to flexible low-power hand-held systems. In this project, as part of an application domain specific architecture for the digital signal-processing domain, the *Field Programmable Function Array* (FPFA) was conceived. The FPFA is a reconfigurable device with a data-path that can be configured to implement a number of DSP algorithms energy efficiently [1]. In this paper we mainly discuss the control part of the FPGA. Because of the flexibility of the FPFA data-path there are an enormous number of control signals which leads to a rather complex control structure.

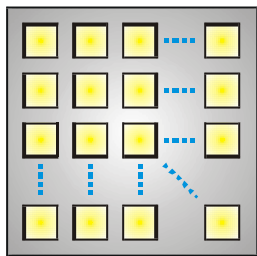
## II. FIELD PROGRAMMABLE FUNCTION ARRAY

Field-Programmable Function Arrays (FPFAs) are reminiscent to FPGAs, but have a matrix of ALUs and

---

<sup>1</sup> 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.

lookup tables [3] 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 processing parts* that share a *control unit*. Figure 1 shows a FPFA with 25 tiles.



**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 FPFA's, 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.

### III. TYPICAL FPFA ALGORITHMS

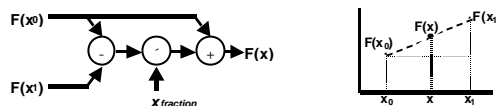
In this section we will portray several widely used algorithms from the digital signal-processing domain: linear interpolation, finite-impulse response filter and Fast Fourier Transform. These algorithms are typical examples of algorithms that can execute efficiently on an FPFA.

#### Linear interpolation

For a point  $x$  ( $x_0 \leq x < x_1$ ), two surrounding sample values  $F(x_0)$  and  $F(x_1)$  are looked up in a table. With these values the in between sample value  $F(x)$  is estimated with:

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

$x_{fraction} = (x - x_0) / (x_1 - x_0)$  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.

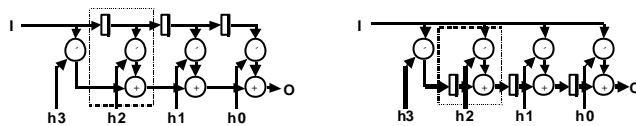


**Figure 2:** Linear interpolation

#### Finite-impulse response filter

Figure 3 shows the *direct form* (left) and the *direct transposed form* (right) implementation of a 4-tap FIR filter. Mathematically an  $N$ -tap FIR filter can be expressed as:

$$O_j = \sum_{n=0}^{N-1} h_{N-n-1} \times I_{j-n}$$



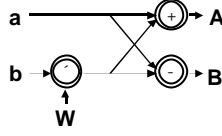
**Figure 3:** FIR filter

#### Fast Fourier Transform

The *Fast Fourier Transform* (FFT) can be used to calculate a *Discrete Fourier Transform* (DFT) efficiently. FFT recursively divides a DFT into smaller DFTs. Eventually only basic DFTs remain. These basic DFTs can be calculated by a structure called a *butterfly*. The butterfly is the basic element of a FFT. Figure 4 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 FFT butterfly can be written as the following equation:

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



**Figure 4:** The radix 2 FFT butterfly

#### IV. FPFA ALU

Based upon [5] and [3], 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 5 – has 4 inputs ( $a, b, c, d$ ) and 2 outputs ( $OUT1, OUT2$ ). The input operands are all 16-bit, 2-complement numbers. The internal ALU data-paths are either 20 or 40 bits wide. The ALU supports both integer and fixed point arithmetic.

As can be seen in Figure 5, 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 5 each can perform five operations: add, subtract, absolute value, minimum and maximum. The result of level one is:

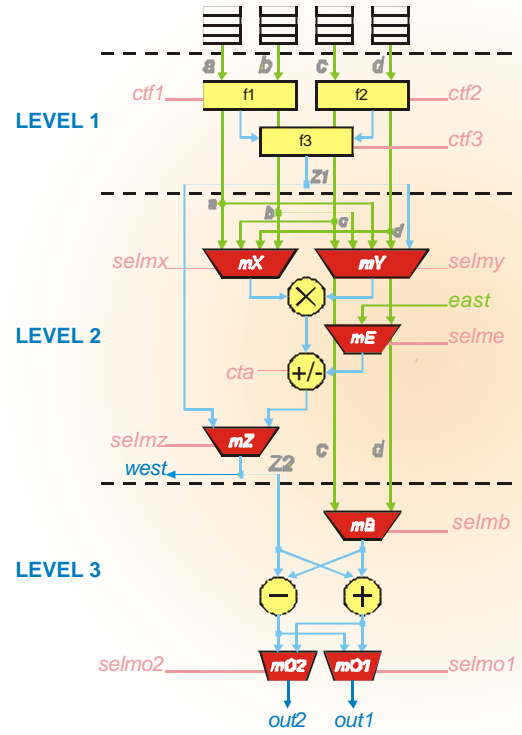
$$Z1 = f_3(f_1(a, b), f_2(c, d))$$

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

- *Level two* contains a  $19 \times 19$ -bit unsigned multiplier and a 40-bits wide adder. The multiplier result is thus 38-bits + a sign-bit. The *East-West* interconnect connects neighbouring ALUs, which enables multiple precision arithmetic. 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. The behaviour of level two as can be specified as:<sup>2</sup>

$$\begin{aligned}
 Z_2 &= Z_1 \\
 &| \quad +((a|b|c|d) \times (a|b|c|d|Z1), [-](0|d_{fp}|d_{se}|east))
 \end{aligned}$$

<sup>2</sup> The subscript *se* stands for *sign extended* and the subscript *fp* stands for *fixed point*.



**Figure 5:** ALU data-path

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

$$out1 = o1_{fp} | o1_{low} | o1_{high} | o2_{low}$$

$$out2 = o2_{fp} | o2_{low} | o2_{high} | o1_{high}$$

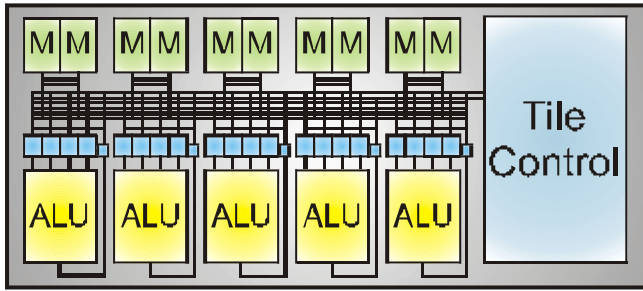
Where:

$$o1 = +(0|c_{se}|c \& d|c_{fp}, Z2)$$

$$o2 = +(0|c_{se}|c \& d|c_{fp}, -Z2)$$

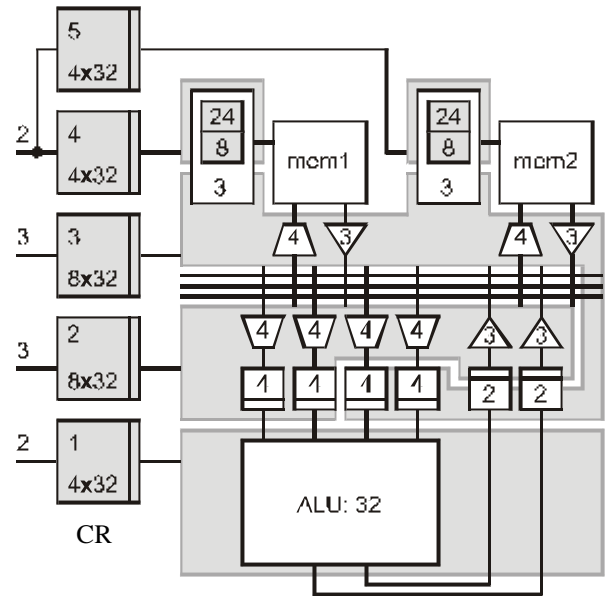
#### V. FPFA CONTROL

A FPFA *processor tile* consists of five identical *processing parts*, which share a *control unit* (see Figure 6). An individual processing part contains an ALU, two memories, four input register banks of four 20-bit wide registers and two 20-bit wide bypassable output 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. The crossbar enables an ALU to write-back to any register or memory within a tile.



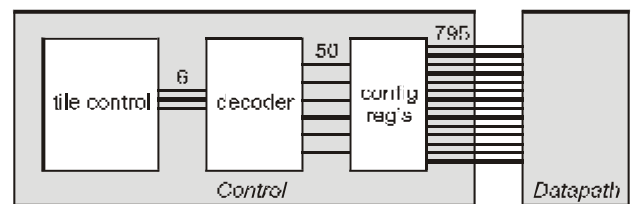
**Figure 6:** Processor tile

The flexibility in the FPPA results in a data-path that requires many control signals. To reduce the vast amount of control signals, the FPPA control-path uses a combination of *configuration registers* and *vertical microprogramming*. A configuration register is allocated to a data-path entity such as an arithmetic logic unit (ALU) or local memory. A configuration register contains a set (typically 4 or 8) of control signals for its associated data-path entity. The idea is illustrated in the FPPA *processing part* that is depicted in Figure 7. The left side of Figure 7 shows five configuration registers. Each configuration register is associated with a particular entity of the processing part. In Figure 7 it can be seen that configuration register 1 (CR1) is allocated to the ALU. CR1 can hold four different configurations of the ALU data-path. For example, ALU configuration CR1[0] can contain the control signals to configure the ALU as an adder and CR1[1] can contain the control signals to configure the ALU as a multiplier and so on. Before a processing part can be used, the configuration registers have to be filled with the configurations of the associated processing entities. On a more abstract level, this filling of the configuration registers can be considered as (a part of) *configuring the FPPA device*. After configuration of the FPPA device, an algorithm can be executed by selecting a particular sequence of configurations from the configuration registers. The configuration time of the FPPA device is amortized over the execution time of a particular algorithm (e.g. a filter). As a matter of speaking, the configuration registers determine a *temporal instruction set* for the processing part. The configuration registers reduce the number of control signals from 159 per processing part to 10 per processing part.



**Figure 7:** Configuration registers

To reduce the number of control signals even more, vertical microprogramming is performed on the configuration register selection signals. FPPAs group five processing parts on a *tile*. The five processing parts on a tile share one control unit. The configuration registers on a tile require  $5 \cdot 10 = 50$  control signals. By vertical microprogramming this number of control bits is reduced to six. This is illustrated in Figure 8. The decoder translates FPPA *tile instruction codes* into control signals for the configuration registers. The decoder can store  $2^6 = 64$  tile instruction codes. This provides enough flexibility for a wide range of DSP algorithms. A sequencer in the *tile control* part of Figure 8 generates the six control signals of the decoder.



**Figure 8:** Control-path

## VI. CONCLUSION AND FUTURE WORK

Looking at the trade-off between flexibility, speed and energy consumption, an FPFA seems the logical step ahead, providing a new and better way for the implementation of highly repetitive, computationally intensive DSP algorithms. On an FPFA many computations can be performed in parallel. It has a reconfigurable data-path to create flexibility and it has short design times, because execution of another application does not require the design of a new integrated circuit.

Three often-used DSP algorithms have been studied: linear interpolation, FIR and FFT. Of course, these are only a few of the algorithms that the FPFA should be able to handle. These three algorithms have been mapped onto the FPFA successfully.

The data-path of an FPFA tile contains five processing parts. Each processing part uses an ALU, four register files, a crossbar and two memories. Control consists of three parts: configuration registers, decoder and Tile control. Vertical programming is used because the rather complex data-path with many control signals requires a relatively small number of different combinations. The FPFA has been designed and implemented and the FPFA architecture is specified in a high-level description language (VHDL).

Logic synthesis has been performed and a one FPFA tile design fits on a Xilinx Virtex XCV1000. The Virtex can run at least at 6.5 MHz. In CMOS18 one FPFA tile is predicted to have an area of 2.6 mm<sup>2</sup> and it can run at least at 23 MHz. The FFT algorithm has been simulated successfully.

The implementation of an algorithm has shown that programming an FPFA by hand is not trivial. The FPFA control-path differs severely from the control part of conventional microprocessors. It is crucial that the control part of the FPFA is designed together with a programming model. The prospect of reconfigurable devices depends on the ability to automate the design flow. Rapid application development tools (like high-level language compilers) for reconfigurable architectures are even more complex than for conventional microprocessors. For example, contemporary compilers are incapable to cope with temporal instruction sets or massive parallel architectures.

Already it can be concluded that reconfigurable devices should not be designed in isolation, but in a multidisciplinary environment. A compelling

understanding of application algorithms, software design, system design and hardware design is essential to the progression of reconfigurable system design.

## VII. REFERENCES

- [1] Heysters P.M. et.al.: "Exploring Energy-Efficient Reconfigurable Architectures for DSP Algorithms", *Proceedings of the first PROGRESS workshop*, pp. 37-45, October 2000.
- [2] Mehra R., Rabaey J., "Exploiting regularity for low-power design", *proceedings of the international Conference on computer-aided design*, 1996.
- [3] Smit J., et al, "Low Cost & Fast Turnaround: Reconfigurable Graph-Based Execution Units", *Proceedings Belsign Workshop*, 1998.
- [4] Smit G.J.M., Havinga P.J.M., Bos M., Smit L.T., Heysters P.M.: "Reconfiguration in mobile multimedia systems", *proceedings PROGRESS 2000 workshop*, pp. 95-106, ISBN 90-73461-22-7, Utrecht, the Netherlands, Oct. 2000.
- [5] Stekelenburg M., "Optimization of a Field Programmable Function Array", *Ms. thesis, University of Twente*, March 1999.