Flexibility of the Montium Word-Level Reconfigurable Processing Tile

Paul M. Heysters, Gerard J.M. Smit, Bert Molenkamp, Gerard K. Rauwerda
University of Twente / Electrical Engineering, Mathematics and Computer Science
P.O. Box 217, 7500 AE Enschede, The Netherlands
Phone: +31 (0)53 489 4178 Fax: +31 (0)53 489 4590
E-mail: p.m.heysters@utwente.nl

Abstract— **Primary requirements of wireless multimedia handheld computers are high-performance, flexibility, energy-efficiency and low costs. In this paper an integral approach to tackle these contradicting requirements is outlined. A System-on-Chip (SoC) template that comprises heterogeneous processing tiles is proposed. This SoC template offers a balance between flexibility, efficiency and performance. The coarse-grained reconfigurable Montium processing tile is introduced. The implementation of a C++ compiler for the Montium is also discussed. High-level language compilers are crucial for the economic viability of a reconfigurable architecture, since they accelerate product development and thus save costs. Software defined radio is used as a sample application for the dynamically reconfigurable Montium processing tiles. The same hardware can be used to implement the baseband processing of both a HiperLAN/2 and a Bluetooth receiver. The inherent overhead of reconfigurable architectures compared to dedicated architectures is amortized by the benefits of flexibility.**

Keywords— **dynamic reconfiguration; energy-efficiency; System-on-Chip**

I. INTRODUCTION

Future wireless terminals must meet several (often conflicting) requirements: high performance, good energy-efficiency, high flexibility, support for Quality of Service (QoS), small size, and low cost. In addition to that, the time to market is a crucial factor. This paper addresses the design of a reconfigurable platform for wireless multimedia communication within the Chameleon project [14]. The flexibility offered by this platform allows upgrading the system with new or enhanced applications or standards and enables the system to adapt dynamically to changing environmental conditions.

Reconfigurable systems offer the flexibility and adaptability needed for future QoS based systems. Reconfigurable computing can enhance the efficiency of these systems: doing more work with the same amount of resources. Reconfigurability reduces risks: these systems can adapt to standards that may change during and after product development and the time to market can be shortened. It will enable systems with true multi-standard capability that extends the product's life cycle. When a good architecture is chosen at design time, it is possible to enhance the system later for applications the system was not originally designed for.

We expect that the combination of high-level design tools and reconfigurable hardware architectures will enable designers to develop highly flexible, efficient and adaptive systems and applications for future multimedia terminals. Performance and power consumption will improve by applying dynamically reconfigurable (heterogeneous) architectures. In this approach application-specific chip-design is replaced by dynamic reconfiguration and reprogramming.

In the Chameleon project we defined an architectural template for a heterogeneous SoC for mobile computing. A sample Chameleon architecture is given in Figure 1. Such a heterogeneous SoC combines performance, flexibility and energy-efficiency. In the template four processor types are distinguished:

- *general-purpose* – i.e. general-purpose processors and digital signal processors,
- *fine-grained reconfigurable* – i.e. FPGAs,
- *coarse-grained reconfigurable* – e.g. Montium tiles (see below) and
- *application specific* – i.e. ASICs.

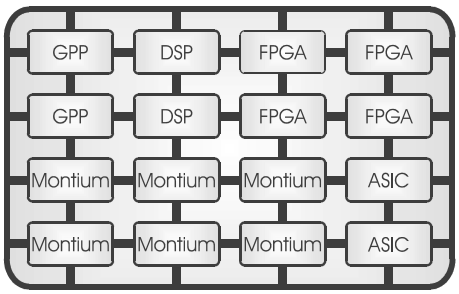

Figure 1:*The Chameleon SoC template.*

The programmability of the architecture enables the system to be targeted at multiple applications. Achieving a high clock speed is not a primary goal for energy-efficient architectures. Instead, massive parallelism is used to achieve a high performance.

II. RELATED WORK

Both academy and industry show interest in coarse-grained reconfigurable architectures. The Pleiades project at UC Berkeley [1] focuses on an architectural template for ultra low-power high-performance multimedia computing. In the Pleiades architecture template a general-purpose microprocessor is surrounded by a heterogeneous array of autonomous, special-purpose satellite processors. The Pleiades SoC design methodology assumes a (very) specific algorithm domain. A processor for speech coding applications, called Maia, was designed using the Pleiades architecture template. Quicksilver's adaptive computing machine (ACM) [7] technology is intended for low-power mobile devices. Their key observation is that algorithms are heterogeneous by nature. The architecture of the ACM comprises heterogeneous nodes of different granularities. The extreme processor platform (XPP) of PACT [2] is based on clusters of coarse-grained processing array elements (PAEs), which is either an ALU or a memory. Actual PAEs are tailored to the algorithm domain of a particular XPP processor.

III. THE MONTIUM ARCHITECTURE

The Montium – a descendent of the Field Programmable Function Array [9][11][12] – targets the 16-bit digital signal processing (DSP) algorithm domain. A single Montium tile is depicted in Figure 2. At first glance the Montium architecture bears a resemblance to a VLIW processor. However, the control structure of the Montium is very different. For (energy-) efficiency it is imperative to minimize the control overhead. This can be accomplished by statically scheduling instructions as much as possible on compile time.

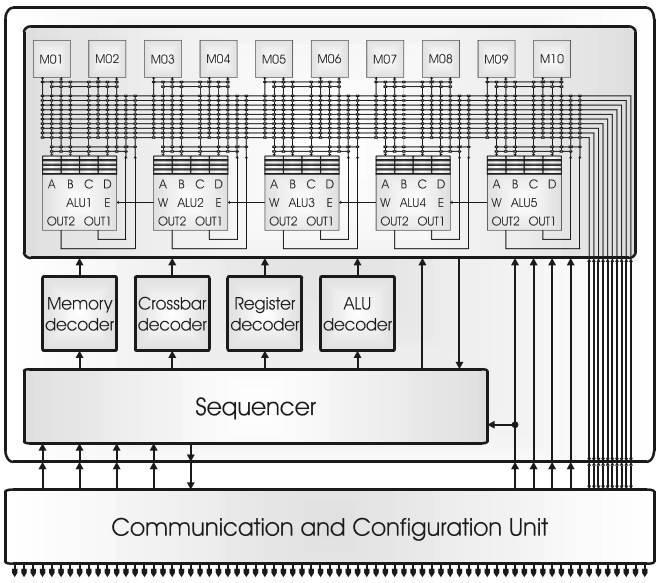

Figure 2*: The Montium processing tile.*

A Montium processing tile is depicted in Figure 2. The lower part of the figure shows the Communication and Configuration Unit (CCU) and the upper part shows the reconfigurable Tile Processor (TP). The TP is the computing part that can be configured to implement a particular algorithm.

A. Communication and Configuration Unit

The CCU implements the interface for off-tile communication. The definition of the off-tile interface depends on the interconnect technology that is used in the SoC. It is very well possible that the TP has its own clock domain and that the CCU needs to provide synchronization between the clock domains. The on-chip communication network for our experimental SoC is still subject of study. Currently we have an FPGA-based CCU prototype that mimics a standard SRAM interface [15]. Note that this interface was solely chosen for quick verification of concepts.

The tasks of the CCU are best illustrated by an example. Suppose that a Montium TP is to perform a particular algorithm. First, a remote configuration manager sends a configuration binary that implements the algorithm to the CCU. The CCU uses the binary to configure the TP. It may even need to reconfigure parts of itself if there is configuration information regarding communication. The CCU also receives input data from the controller. The CCU uses direct memory access to write this data into the local memories of the TP. The TP is now configured and the input data is stored in its local memories. The CCU signals the sequencer of the TP that it can start computing. The CCU then waits until the TP signals that the algorithm has finished. It retrieves the

results from the local memories and forwards them to their off-tile destination. Subsequently, the CCU is ready to receive either new input data and run the algorithm again or to receive new configuration information for another algorithm. Alternatively, the data can remain resident in the local memories while the tile is reconfigured for the next algorithm. This particular example operates on blocks of data. The Montium can also stream 16-bit samples in case latency is an important quality of service parameter. The CCU can access the communication busses within the TP during the execution of an algorithm. In this streaming mode the CCU synchronizes to the TP, but it can suspend the TP in case of a full or empty buffer.

The TP has a special configuration interface that the CCU uses for configuration. The Montium TP contains the following configurable entities:

- *configuration registers* that can hold local instructions (607.5 bytes),
- *initialization registers* for the memory address generation units (157.5 bytes),
- *decoders* that can hold tile instructions (680 bytes) and
- *instruction memory* for the sequencer of the TP (1.25Kbyte).

The configuration space that contains all configurable entities is memory mapped. The configuration interface comprises a 12-bit address port and a 16-bit write-only data port. The size of the entire TP configuration space is about 2.6Kbyte. However, for a particular algorithm only a fraction of the configuration space is actually used. For example, a 5-tap FIR filter that filters 512 samples from a local memory and stores the results in another memory requires a configuration binary of only 240 bytes. Unsurprisingly, it takes 120 clock cycles to load this particular configuration using the configuration interface.

B. Tile Processor

The Tile Processor is tailored for a specific algorithm domain. We chose the rather loose domain of 16-bit DSP algorithms, because we are particularly interested in the flexibility that can be achieved in a coarse grained reconfigurable architecture. The algorithms that we used to refine the datapath of the Montium TP include: FIR filters, FFT, correlation, turbo decoding, DCT, matrix multiplication and linear interpolation [8][9]. A thorough understanding of the algorithm domain is crucial to design an (energy-) efficient reconfigurable architecture. The architecture should impose little overhead to run the algorithms in its domain. Communication is in essence also overhead, as it does not contribute to the computation of an algorithm. Therefore, there needs to be a sound balance between computation and communication. To achieve this balance, the Montium has comparatively large combinational ALUs. The ALUs can perform a relatively complicated computation before the communication interconnect is accessed. The ALUs are not pipelined, as they introduce significant overhead to handle conditional execution and complicate a compiler tremendously. Achieving a high clock speed is not a primary goal for energy-efficient architectures. Instead, massive parallelism can be used to achieve a high performance.

Figure 2 reveals that the hardware organization within a tile is very regular. The five identical ALUs (ALU1…ALU5) in a tile can exploit spatial concurrency to enhance performance. This parallelism demands a very high memory bandwidth, which is obtained by having 10 local memories (M01…M10) in parallel. The small local memories are also motivated by the locality of reference principle. The datapath has a width of 16-bits and the ALUs support both signed integer and signed fixed-point arithmetic. The ALU input registers provide an even more local level of storage. Locality of reference is one of the guiding principles applied to obtain energy-efficiency in the Montium. A vertical segment that contains one ALU together with its associated input register files, a part of the interconnect and two local memories is called a Processing Part (PP). The five Processing Parts together are called the Processing Part Array (PPA). A relatively simple sequencer controls the entire PPA.

1) Sequencer

The sequencer selects configurable PPA instructions that are stored in the decoders of Figure 2. In Figure 3 a miniscule sequencer program is listed. The program implements the earlier mentioned 5-tap FIR filter example. It filters 512 input samples from a local memory M1 and stores the results in memory M9. First the sequencer waits until the CCU has finished loading the 512 samples and the filter coefficients. The instruction at address 1 reads the first sample from memory M1 and at the same time one of the ALUs generates a zero, which will be used to clear the “FIR-delay” registers. The next instruction (at address 2) loads the first sample and the generated zero into the ALU input registers. The same instruction also tells the ALUs to start computing the first five multiply-accumulate (MAC) operations. In the next instruction the first result

is written to memory M9. At the same time a loop counter is set in the sequencer. The "loop" instruction at address 4 will loop until the loop counter is zero. In this case, the loop will be executed 510 times. At the end of the loop all input samples have been read (the sharp-eyed reader will notice that M1 is read 513 times, however the instruction at address 4 does not load the value that was read last into the input registers). When the last result has been stored, the sequencer signals the CCU that it is ready. When the handshake has been acknowledged, the sequencer program starts over again and waits for the next block of samples to be filtered.

```
Addr Sequencer instruction PPA instruction
  0  Wait for data valid   NOP
  1  NOP                   rd_M1,alu=0
  2  NOP                   rd_M1,ld_reg,alu=MAC
  3  LoopCounter:= 509     rd_M1,ld_reg,alu=MAC,wr_M9
  4  Loop to address 4     rd_M1,ld_reg,alu=MAC,wr_M9
  5  Signal "done"         wr_M9
  6  Wait for data invalid NOP
  7  Clear "done"          NOP
  8  Jump to address 0     NOP
```

Figure 3: *FIR filter pseudo code for the sequencer.*

2) Local memories

Each local SRAM is 16-bit wide and has a depth of 512 positions, which adds up to a storage capacity of 8Kbit per local memory. A memory has only a single address port that is used for both reading and writing. A reconfigurable Address Generation Unit (AGU) accompanies each memory. The AGU contains an address register that can be modified using base and modify registers. The AGU also supports cyclic buffers and bitreversal. It is also possible to use the memory as a lookup table for complicated functions that cannot be calculated using an ALU, such as sine or division (with one constant). A memory can be used for both integer and fixed-point lookups.

3) Interconnect

The interconnect provides flexible routing within the PPA. The configuration of the interconnect can change every clock cycle. There are ten busses that are used for inter-PP communication. These busses are called global busses. Note that the span of global busses is only the PPA within a single TP. The CCU is also connected to the global busses. The CCU uses the global busses to access the local memories and to handle data in streaming algorithms. Communication within a PP uses the more energy-efficient local busses.

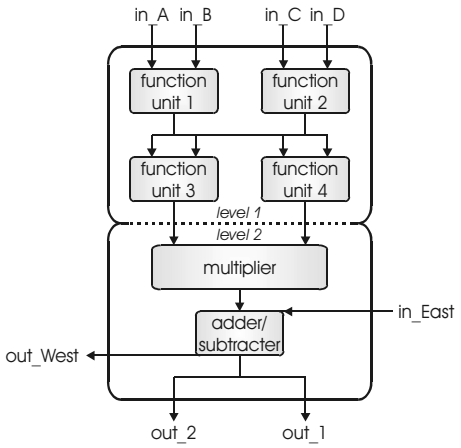

Figure 4*: The Montium ALU.*

4) ALUs

A single ALU has four 16-bit inputs. Each input has a private input register file that can store up to four operands. The input register file cannot be bypassed, i.e. an operand is always read from an input register. Input registers can be written by various sources via a flexible interconnect. An ALU has two 16-bit outputs, which are connected to the interconnect. The ALU is entirely combinational and consequentially there are no pipeline registers within the ALU.

The Montium ALU, which is depicted in Figure 4, is divided into two levels. The upper level, level 1, contains four function units. A function unit implements general arithmetic and logic operations that are available in languages like C. Most arithmetic operations apply saturation on the result. Saturation can be detected by means of the overflow flag. Normal addition/subtraction (i.e. without saturation) is also supported. The lower level, level 2, contains the MAC unit and is optimised for algorithms such as FFT and FIR. Additions or subtractions on level 2 always use saturated arithmetic. Levels can be bypassed (in software) when they are not needed. If it is known beforehand that a level is never going to be used in the aimed algorithm domain, then a Montium TP can be made in which an entire level is removed. For example, if the algorithm domain is limited to FFT and FIR filters only, then level 1 is not used and a TP realization with only level 2 will suffice. The level interface has been designed in such a way that a level can be left out easily. It is in principle also possible to replace a level or to insert a new one. The exact organization of the ALU should be optimised for the aimed algorithm domain. The current level 1 of the Montium ALU is rather general purpose, because we did not want to restrict the algorithm domain of our experimental processor too much.

Neighbouring ALUs can also communicate directly on level 2. The West-output of an ALU connects to the East-input of the ALU neighbouring on the left (the West-output of the leftmost ALU is not connected and the East-input of the rightmost ALU is always zero). The 32-bit wide East-West connection makes it possible to accumulate the MAC result of the right neighbour to the multiplier result (note that this is also a MAC operation). This is particularly useful when performing a complex number multiplication, or when adding a large amount of numbers (up to 20 in one clock cycle). The East-West connection does not introduce a delay or pipeline, as it is not registered.

IV. THE MONTIUM COMPILER

An architecture with a reconfigurable instruction set poses a challenge for developing a compiler. Yet, the availability of high-level design entry tools is essential for the (economic) viability of any reconfigurable architecture. It is essential that new applications can be implemented quickly and at low cost. The Montium has a straightforward control mechanism that has moderate overhead and enables the development of a compiler.

A C++ based compiler is currently being developed for the Montium TP. A four-phase decomposition is used for mapping a C++ program on the Montium TP.

A. Phase 1: Translation

The translation step generates a Control Dataflow Graph (CDFG) from the C++ source code [13].

B. Phase 2: Clustering

In the clustering phase the CDFG is partitioned and mapped to an unbounded number of fully connected Montium ALUs, which can perform inter-ALU communication simultaneously. Clusters are selected such that they can be executed on one ALU in one clock-cycle [6].

C. Phase 3: Scheduling

In the scheduling phase, the graph obtained from the clustering phase is scheduled according to the maximum number of ALUs (in our case 5). The configurations for all five ALUs of one clock cycle form a *5-ALU configuration* [5].

D. Phase 4: Allocation

In the allocation phase, the scheduled graph is mapped to the resources in such a way that locality of reference is exploited. The latter is important for performance and energy reasons [5].

V. MONTIUM TP SAMPLE APPLICATION

In this section, the feasibility of implementing a multi-standard communication system on the Montium TP is analysed. The mappings of the most important baseband functions of both a HiperLAN/2 [4] and a Bluetooth [3] receiver are discussed.

In the HiperLAN/2 receiver, information in the time domain has to be translated to the frequency domain. In order to synchronize, preambles are detected with matched filters. After synchronization, the inverse Orthogonal Frequency Division Multiplexing (OFDM) function takes care of the time-to-frequency-domain translation. This function is performed by a Fast Fourier Transform (FFT), which is applied on 64 complex samples. Finally, the hard output bits are determined by comparing the soft output values with values in a lookup table. The size of the lookup table depends on the used modulation scheme (i.e. BPSK, QPSK, 16-QAM or 64-QAM).

Table 1 shows the number of clock cycles and the number of ALU configurations required for receiving one HiperLAN/2 symbol. These numbers were acquired by mapping the individual function blocks on the Montium TP. The required processing time of de-mapping depends on the used modulation scheme. Assuming the tiles run at a clock frequency of 100 MHz, the receiver part can be implemented in 2 or 3 Montium TP tiles, depending on the modulation type.

The Bluetooth receiver contains an FM-discriminator, a Low Pass Filter (LPF) and a threshold detector. In the FM-discriminator FM-to-AM conversion is performed by multiplying the received signal with its delayed version. After FM-to-AM conversion, the signal is passed through a low pass FIR filter. This FIR filter blocks all the high frequencies that were introduced as a side effect of the multiplication. Finally, a threshold detector detects the consecutive bits.

Functional block	# clock cycles	# ALU configurations
Prefix removal	272	2
Frequency offset correction	64	2
Inverse OFDM	198	1
Phase offset correction	128	2
Channel equalization	64	2
De-mapping	48 – 384	1

Table 1: *Computational requirements for receiving one HiperLAN/2 symbol.*

Functional block	# clock cycles	# ALU configurations
FM-discriminator	1	1
N-taps FIR filter	ceil{N/5}+2	2
Threshold detector	1	1

Table 2: *Computational requirements for receiving one Bluetooth bit.*

The functionality of the Bluetooth receiver is rather simple. It can be implemented on the Montium TP quite well. Table 2 shows that the computation delays of the different functional blocks are a few clock cycles. The processing for receiving one bit takes about 130 ns, when a FIR filter with 50 coefficients is applied and the clock frequency of the Montium TP is 100 MHz. These examples illustrate the flexibility of the Montium TP tiles: 2 or 3 tiles (depending on the modulation type) can implement the baseband processing of both HiperLAN/2 and Bluetooth.

VI. THE COST OF FLEXIBILITY

We believe that reconfiguration is inevitable for future mobile terminals. However, a price has to be paid for reconfigurability in both chip area and energy consumption. In the Montium architecture we have tried to reduce the cost (in terms of area, energy and speed) of flexibility in several ways:

A. Area overhead

The area of an entire Montium Tile Processor in a 0.12 μm process is about at 2 mm^2. Roughly speaking, 1/3 is used for the local memories and 1/3 is used for the actual datapath. The remaining 1/3 of the area is used for control and configuration, which is the area overhead that has to be paid for flexibility.

B. Configuration overhead

The size of the entire Montium Tile Processor configuration space that contains all configurable entities is 2.6 Kbyte. However, for a particular algorithm only a fraction of the configuration space is actually used. The configuration memory is randomly addressable; only the changed configuration words need to be written. For example: for the 5-tap FIR filter example only 240 bytes of configuration data are required.

C. Control overhead

The Montium architecture makes a trade-off between control overhead and flexibility. The control mechanism has been kept simple and straightforward. Therefore, the instruction decoding does not result in excessive switching of control signals. For example: during the FIR filter operation all the control signals remain stable; only the data changes.

D. Performance

The large combinational paths of the ALUs determine the critical path. The worst-case clock frequency of the current implementation of the Montium TP is about 50 MHz in a 0.12 μm process. However, the critical path of an actual algorithm is in general much shorter. For example, the same realization could compute a FIR filter at a clock speed of about 125 MHz. This is because a FIR filter uses only level 2 of an ALU and no East-West interconnects are used.

VII. FUTURE WORK

The next step in our research is to determine the energy-efficiency of the Montium architecture through power simulations. Energy-efficiency is pursued by reducing the overhead of both communication and control. We expect that substantial power, performance and area improvements can be made by abandoning our high-level hardware design approach and making a highly optimised design by hand. This tedious effort is feasible because the design of a Montium tile is small and it will be replicated many times in a SoC.

VIII. CONCLUSION

Mobile multimedia devices need to be energy-efficient, flexible and computationally powerful. To achieve this there must be synergy between algorithms and hardware. In a mobile system a variety of algorithms is used. Hardware architectures for low-power devices should acknowledge this heterogeneity. Algorithms should be executed on domain specific processing units, which can be small and lightweight. Hardware that targets a limited algorithm domain can make a much better trade-off between efficiency and flexibility than general purpose or dedicated hardware. Besides the GPPs, ASICs and FPGAs, also coarse-grained reconfigurable architectures are needed.

The Montium is an experimental coarse-grained reconfigurable architecture. In this paper the design of the Montium processing tile was outlined. The design focuses on a trade-off between (energy-) efficiency, flexibility and performance. Performance can be obtained by the parallelism in a tiled architecture. The mappings of various algorithms have shown the flexibility of the architecture.

A reconfigurable architecture like the Montium is only viable if high-level design entry tools accompany it. The design of a high level compiler for coarse-grained reconfigurable architectures was also briefly discussed.

The approach looks very promising and we believe that in the end the entire flow (from C++ to a configuration binary) can be automated.

Finally, we showed that a SoC containing Montium processing tiles is flexible and powerful enough to implement a HiperLAN/2 receiver as well as a flexible Bluetooth receiver on the same architecture.

REFERENCES

[1] Abnous A.: "Low-Power Domain-Specific Processors for Digital Signal Processing", *Ph.D Dissertation, University of California*, Berkeley, USA, 2001.

[2] Baumgarte V., May F., Nückel A., Vorbach M. & Weinhardt M.: "PACT XPP – A Self-Reconfigurable Data Processing Architecture", *Proceedings Engineering of Reconfigurable Systems and Algorithms*, pp. 64-70, Las Vegas, USA, June 2001.

[3] Bluetooth SIG, "Specification of the Bluetooth System – Core", Technical Specification Version 1.1, 22 February 2001.

[4] ETSI, "Broadband Radio Access Networks (BRAN); HiperLAN type 2; Physical (PHY) layer", ETSI TS 101 475 V1.2.2 (2001-02), 2001.

[5] Guo Y., Smit G.J.M., Broersma H., Rosien M.A.J. & Heysters P.M.: "Mapping Applications to a Coarse Grain Reconfigurable System", ***accepted for*** *The Eighth Asia-Pacific Computer Systems Architecture Conference*, 2003.

[6] Guo Y., Smit G.J.M., Heysters P.M. & Broersma H., "A Graph Covering Algorithm for a Coarse Grain Reconfigurable System", *Proceedings of the 2003 ACM SIGPLAN Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'03)*, pp.199-208, San Diego, USA, June 2003, ISBN 1-58113-647-1.

[7] Heidari G. & Lane K.: "Introducing a Paradigm Shift in the Design and Implementation of Wireless Devices", *Proceedings Wireless Personal Multimedia Communications*, vol.1 pp. 225-230, Aalborg, Denmark, September 2001.

[8] Heysters P.M. & Smit G.J.M.: "Mapping of DSP Algorithms on the Montium Architecture", *Proceedings of the 17th International Parallel & Distributed Processing Symposium Reconfigurable Architectures Workshop (RAW 2003)*, Nice, France, April 2003, ISBN 0-7695-1926-1.

[9] Heysters P.M., Smit L.T., Smit G.J.M. & Havinga P.J.M.: "Max-Log-MAP Mapping on an FPFA", *Proceedings of Engineering of Reconfigurable Systems and Algorithms*, pp. 90-96, Las Vegas, USA, June 2002.

[10] Heysters P.M., Smit J., Smit G.J.M. & Havinga P.J.M.: "Exploring Energy-Efficient Reconfigurable Architectures for DSP Algorithms", *Proceedings of the 1st PROGRESS workshop on Embedded Systems*, pp. 43-51, Utrecht, The Netherlands, October 2000, ISBN 90-73461-25-1.

[11] Heysters P.M., Bouma H., Smit J., Smit G.J.M. & Havinga P.J.M.: "Reconfigurable System Design: The Control Part", *Proceedings of the 2nd PROGRESS workshop on Embedded Systems, pp. 89-93*, Veldhoven, The Netherlands, October 2001, ISBN 90-73461-27-X.

[12] Heysters P.M., Smit G.J.M. & Molenkamp B.: "Reconfigurable Architecture for Handheld Devices", *Proceedings of the 3rd PROGRESS workshop on Embedded Systems*, pp. 66-70, Utrecht, The Netherlands, October 2002, ISBN 90-73461-34-0.

[13] Rosien M.A.J., Guo Y., Smit G.J.M., Krol T.: "Mapping Applications to an FPFA Tile", *Proceedings of DATE 2003*, pp. 1124-1125, Munich, Germany, March 2003.

[14] Smit G.J.M., Havinga P.J.M., Smit L.T., Heysters P.M., Rosien M.A.J.: "Dynamic Reconfiguration in Mobile Systems", *Proceedings of Field-Programmable Logic and Applications*, pp. 171-181, Montpellier, France, September 2002.

[15] Vellinga H.H.: "Communication and Configuration Unit", MSc. Thesis, University of Twente, Enschede, The Netherlands, February 2003.