Multi-core Architectures and Streaming Applications

Gerard J.M. Smit, André B.J. Kokkeler, Pascal T. Wolkotte, Marcel D. van de Burgwal
University of Twente
Department of EEMCS
P.O. Box 217, 7500 AE Enschede, The Netherlands
G.J.M.Smit@utwente.nl

ABSTRACT

In this paper we focus on algorithms and reconfigurable multi-core architectures for streaming digital signal processing (DSP) applications. The multi-core concept has a number of advantages: (1) depending on the requirements more or fewer cores can be switched on/off, (2) the multi-core structure fits well to future process technologies, more cores will be available in advanced process technologies, but the complexity per core does not increase, (3) the multi-core concept is fault tolerant, faulty cores can be discarded and (4) multiple cores can be configured fast in parallel. Because in our approach processing and memory are combined in the cores, tasks can be executed efficiently on cores (locality of reference). There are a number of application domains that can be considered as streaming DSP applications: for example wireless baseband processing (for HiperLAN/2, WiMax, DAB, DRM, and DVB), multimedia processing (e.g. MPEG, MP3 coding/decoding), medical image processing, colour image processing, sensor processing (e.g. remote surveillance cameras) and phased array radar systems. In this paper the key characteristics of streaming DSP applications are highlighted, and the characteristics of the processing architectures to efficiently support these types of applications are addressed. We present the initial results of the Annabelle chip that we designed with our approach.

Categories and Subject Descriptors

C.1.3 [**Processor architectures**]: Other Architecture Styles—*Heterogeneous (hybrid) systems*; C.4 [**Computer Systems Organization**]: Performance of systems—*Design studies*

General Terms

Design, Measurement, Performance

Keywords

Multi-core SoC design, NoC design, streaming applications, system design

1. INTRODUCTION

This paper addresses the design issues of a reconfigurable multi-core System-on-Chip (SoC) platform for streaming DSP applications. In streaming DSP applications computations can be specified as a data flow graph with streams of data items (the edges) flowing between computation kernels (the nodes). Most signal processing applications can be naturally expressed in this modelling style [3]. Typical examples of streaming DSP applications are: wireless baseband processing, multi-media processing, medical image processing and sensor processing e.g. for remote surveillance cameras and phased array radars. In the 4S project we defined a dynamically reconfigurable heterogeneous multi-core SoC architecture (see section 2). In this architecture a core can either be: a bit-level reconfigurable unit (e.g. FPGA), a word-level reconfigurable unit (e.g. Montium tiles [7, 8]), or a general-purpose programmable unit (DSP or microprocessor). The cores on the SoC are interconnected by a reconfigurable Network-on-Chip (NoC). The programmability of the individual cores enables the system to be targeted at multiple application domains.

1.1 Holistic Approach

In the 4S project we take a holistic approach, which means that all aspects of systems design need to be addressed simultaneously in a systematic way. We believe that this is key for an efficient overall solution. An interesting optimization in a small corner of the design might lead to inefficiencies in the overall design. For example the design of the NoC should be coordinated with the design of the processing cores, and the design of the processing cores should be coordinated with the tile specific compilers. Eventually, there should be a tight fit between the application requirements and the SoC and NoC capabilities.

1.2 Predictable Solutions

To manage the complexity of streaming DSP applications, predictable techniques should be used. For example: the NoC should provide latency and throughput guarantees, and the real-time schedulers in core processors should provide latency guarantees. One reason for predictability is that the amount of data in streaming DSP applications is so high that even a large buffer would be too small to compensate for unpredictably behaving components and that the latency that these buffers would introduce is not acceptable in typical streaming DSP applications. A second reason for using predictable techniques is because of the composability issue. In case multiple applications are mapped on the same platform, the behaviour of one application should not influence another application.

Furthermore, in these applications there are often hard deadlines at the beginning of the chain (e.g. sampling rate of an A/D converter) or at the end of the chain (e.g. fixed rate of the D/A converter, or update rate of the screen). In other applications such as phased arrays applications individual paths of signals should be exactly timed before they can be combined. Also in these applications the data rate is so high (e.g. 100 Msamples/s) that buffering of data is not useful.

Unfortunately, future semiconductor technologies will introduce more uncertainty. Design techniques will have to include resiliency at the circuit and micro-architecture level to deal with these uncertainties and the variability at the device technology level. One of the future challenges is to design predictable systems with unpredictable components.

1.3 Energy-efficiency

Portable devices rely on batteries; the functionality of these devices is strictly limited by the energy consumption. There is an exponential increase in demand for streaming communication and processing for wireless protocol baseband processing and multimedia applications, but the energy content of batteries is only increasing at a pace of 10% per year.Also for high performance computing there is a need for energy-efficient architectures to reduce the cost for cooling and packaging.

In addition to that there are also environmental concerns that urge for more efficient architectures in particular for systems that run 24 hours per day such as wireless base stations and search engines (e.g. Google has an estimated server park of one million servers that run 24 hours per day).

Most components are fabricated using CMOS technology today. The dominant component of energy consumption (85 to 90%) in 130 nm CMOS technology is dynamic power consumption. However, when technology scales to lower dimensions, the static power consumption will become more and more important. A first order approximation of the dynamic power consumption of CMOS circuitry is given by the formula:

$$P_d = \alpha \cdot C_{eff} \cdot f \cdot V^2 \tag{1}$$

where P_d is the power in Watts, C_{eff} is the effective switch capacitance in Farads, V is the supply voltage in Volts, α the activity factor and f is the frequency of operations in Hertz.

Equation (1) suggests that there are essentially four ways to reduce power: reduce the capacitive load C_{eff}, reduce the supply voltage V, reduce the switching frequency f, or reduce the activity α. In the context of this paper we will mainly address reducing the capacitance by using locality of reference and using adaptivity.

1.3.1 Minimize Capacitance

As shown in equation (1) energy consumption in CMOS circuitry is proportional to capacitance. Therefore energy consumption can be reduced by minimizing the capacitance. This can not only be reached at the technological level, but much profit can be gained by an architecture that exploits locality of reference. Connections to external components typically have much higher capacitance than connections to on-chip resources. For example: on-chip capacitance is in the order of 10-50 fF, whereas off-chip capacitance is in the order of 10-20 pF. This means that the energy it takes to read a 32 bit value from an off-chip memory is far more than the energy it takes to perform e.g. a 32x32 multiplication. Therefore, to save energy, use few off-chip wires, and have them toggle as infrequently as possible. Consequently, it is beneficial to use on-chip memories like caches, scratchpads and registers.

1.3.2 Locality of Reference

References to memory typically display a high degree of temporal and spatial locality of reference. Temporal locality of reference refers to the observation that referenced data is often referenced again in the near future. Spatial locality of reference refers to the observation that once a particular location is referenced, a nearby location is often referenced in the near future. Accessing a small and local memory is much more energy-efficient than accessing a big and far distant memory. Transporting a signal over a 1 mm wire in a 45 nm technology will require more than 50 times the energy of a 32-bit operation in the same technology (the off-chip interconnect will consume more than a 1000 times the energy of an on-chip 32-bit operation). A multi-core architecture intrinsically encourages the usage of small and local on-core memories. Exploiting the locality of reference principle extensively improves the energy-efficiency substantially. Due to the locality of reference principle the communications within a core are more frequent than between cores.

1.3.3 Adaptivity

When the system can adapt (at run-time) to the environment, significant savings in computational costs can be obtained [13]. One of the main reasons for introducing reconfigurable hardware in a wireless terminal is to support multiple wireless communication standards [11]. The support of multiple wireless communication standards introduces a first level of adaptivity in the wireless terminal because the terminal can switch between standards. For example, when packet data transport is performed over Universal Mobile Telecommunications System (UMTS) and a Wireless Local Area Network (WLAN) hotspot becomes available the terminal can switch from UMTS to a WLAN standard. This is referred to as *standards level adaptivity.*

Although a wireless communication standard usually defines the DSP functionality, it usually does not define the algorithms that have to be used to implement these functions. Therefore, the communication system can "adapt the algorithms" that are used to implement the DSP functionality.

"Adapt the algorithms" means that the communication system selects an algorithm from a set of algorithms that implement the same DSP functionality. This second level of adaptivity is referred to as *algorithm-selection level adaptivity.* Within a specific algorithm, there are also opportunities for adaptivity by changing parameters of the algorithm. This third level of adaptivity is called *algorithm-parameter level adaptivity.*

Dynamic reconfiguration of hardware is required to achieve real adaptive systems. The reconfiguration rate is highly dependent on the operating environment. Changes within the standards level are due to interaction with the end-user whereas changes within the parameter-level are due to the physical environment. For instance, the standard selected by the user changes on a minute or hour rate, while the parameters of a standard can change on a sub-second rate, influenced by the quality of, e.g., the wireless channel.

1.4 Streaming Applications

In this paper the focus is on multi-core SoC architectures for streaming DSP applications where we can assume that the data streams are semi-static and have a periodic behaviour. This means that for a long period of time subsequent data items of a stream follow the same route through the SoC, often modelled as a data-flow graph.

We have implemented and analysed several streaming applications. For example, physical layer processing for HiperLAN/2, DAB, DRM, WiMAX, and multimedia processing for MPEG-4 video decoding.

The common characteristics of typical streaming DSP applications are:

- They are characterized by relatively simple local processing but a huge amount of data. The trend is that energy costs for data communication dominates energy costs of processing.
- Data arrives at nodes at a rather fixed rate, which causes periodic data transfers between successive processing blocks. The resulting communication bandwidth is application dependent so a large variety in communication bandwidth is required.
- The size of the data items is application dependent, e.g. 14-bit samples for a sensor system, 64 32-bits words for HiperLAN/2 [4] OFDM symbols or 8 x 8 x 24-bits macro blocks for a video application. Also the data rate is application dependent e.g. 100 Msamples/sec after the A/D converter for a sensor system, 200k OFDM symbols per second for HiperLAN/2, 50 frames/sec for video.
- The data flows through the successive processes in a pipelined fashion. Processes might work in parallel on parallel processors or can be time-multiplexed on one or more processors. Therefore, streaming applications show a predictable temporal and spatial behaviour.
- For our application domains typically throughput guarantees (in data items per sec.) are required for the communication as well as for the processing. Sometimes also latency requirements are given.
- The life-time of a communication stream is semi-static, which means a stream is fixed for a relatively long time.

2. EFFICIENT MULTI-CORE ARCHITECTURES

Flexible and efficient SoCs can be realized by integrating reconfigurable hardware parts (called tiles or cores) of different granularities into heterogeneous reconfigurable SoCs. In this paper the term *core is used for processor-like hardware blocks and the term* tile is used for ASICs, fine-grain reconfigurable- and memory blocks. In this section we present techniques we use to design predictable and energy-efficient systems. In line with the holistic view, these techniques range from pure hardware- to software design techniques.

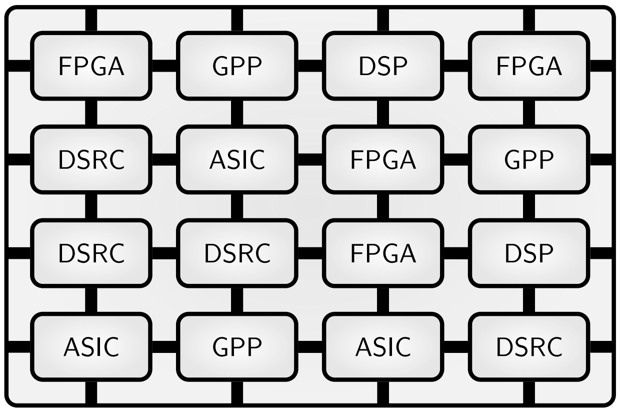

Figure 1: Heterogeneous tiled organization

In our approach we assume that the interconected building blocks can be heterogeneous (see figure 1), e.g.: bit-level reconfigurable tiles (e.g. embedded FPGAs), word-level reconfigurable cores (e.g. Montium tiles), general-purpose programmable cores (e.g. DSPs and microprocessor cores) and memory blocks.

From a systems point of view these architectures are heterogeneous multi-processor systems on a single chip. The programmability and reconfigurability of the architecture enables the system to be targeted at multiple applications.

2.1 Multi-core Architectures

Recently a number of Multi-core architectures have been proposed for the streaming DSP application domain. Examples are: Silicon Hive (an incubator of Philips Research) [6]; the PACT-XPP [2]; the Maya [3] and Pleiades [1] chips from Berkeley; and the Chameleon/Montium architecture from the University of Twente and Recore Systems [8]. For an overview we refer to [7].

A multi-core architecture has a number of advantages:

- It is a future-proof architecture, as the processing cores do not grow in complexity with technology. Instead, as technology scales, simply the number of cores on the chip grows.
- A multi-core organization can contribute to the energy-efficiency of a SoC. The best energy savings can be obtained by simply switching off cores that are not used, which also helps for reducing the static power consumption. Furthermore, the processing of local data in small autonomous cores abides by the locality of reference principle. Moreover, a core processor might not need to run at full clock speed to achieve the required QoS at a particular moment in time.
- When one of the cores is discovered to be defect (either due to a manufacturing fault or discovered at operating-time by the built-in-diagnosis) this defective core can be switched-off and isolated from the rest of the design.
- A multi-core approach also eases verification of an integrated circuit design, since the design of identical cores only has to be verified once. The design of a single core is relatively simple and therefore a lot of

effort can be put in (area/power) optimizations on the physical level of integrated circuit design.

- The computational power of a multi-core architecture scales linearly with the number of cores. The more cores there are on a chip, the more computations can be done in parallel (providing that the network capacity scales with the number of cores and there is sufficient work).

- Although cores operate together in a complex system, an individual tile operates quite autonomously. In a multi-core architecture every processing core is configured independently. In fact, a core is a natural unit of partial reconfiguration. Unused cores can be configured for a new task, while at the same time other cores continue performing their tasks. That is to say, a multi-core architecture can be reconfigured dynamically.

2.2 Heterogeneous Multi-core SoC

The reason for heterogeneity is that typically, some algorithms run more efficiently on bit-level reconfigurable architectures (e.g. PN-code generation), some on DSP-like architectures and some perform optimal on word-level reconfigurable platforms (e.g. FIR filters or FFT algorithms). Application designers or high-level compilers can choose the most efficient processing core for the type of processing needed for a given application task. Such an approach combines performance, flexibility and energy-efficiency. It supports high performance through massive parallelism, it matches the computational model of the algorithm with the granularity and capabilities of the processing entity, it can operate at minimum supply voltage and clock frequency and hence provides energy-efficiency and flexibility at the right granularity only when and where needed and desirable.

A thorough understanding of the algorithm domain is crucial for the design of an (energy-) efficient reconfigurable architecture. The architecture should impose little overhead to execute the algorithms in its domain. Inter-processor 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 inter-processor communication. These are again motivations for a holistic approach.

2.3 Dynamic Reconfiguration

Reconfigurable systems offer the flexibility and adaptivity needed for future streaming DSP applications. The flexibility of a platform is revealed as the ease of upgrading the system with a new or enhanced application or standard (standard-level adaptivity) as well as the ability of the system to adapt dynamically to changing environmental conditions (algorithm-level and parameter-level adaptivity).

There are quite a number of good reasons for using reconfigurable architectures in future mobile systems:

- When the system can adapt – at run-time – to the environment significant power-saving can be obtained. For example: depending on the distance of the receiver and transmitter or cell occupation, more or less processing power is needed.

- Standards evolve quickly; this means that future systems have to have the flexibility and adaptivity to adapt to changes in the standards. By using reconfigurable architectures instead of ASICs costly re-designs can be avoided.

- The design costs complex ASICs is growing rapidly; in particular the mask costs of these chips are very high. With reconfigurable processors it is expected that less chips have to be designed, so companies can save on mask costs.

- Reconfigurability reduces risks; because these systems can adapt to standards that may change during and after product development, and the time to market can be shortened.

- Dynamically reconfigurable architectures allow to experiment with new concepts such as software-defined radios, multi-standard terminals, cognitive radio, adaptive turbo decoding, adaptive equalizer modules and adaptive interference rejection modules.

Reconfigurability also has another more economic motivation: it will be important to have a fast track from sparkling ideas to the final design. Time to market is crucial. If the design process takes too long, the return on investment will be less.

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 systems.

2.4 Network-on-Chip

A multi-core architecture has to be supported by a predictable inter-core communication network e.g. NoC. A NoC that routes data items has a higher bandwidth than an on-chip bus, as it supports multiple concurrent communications. The well-controlled electrical parameters of an on-chip interconnection network enable the use of high-performance circuits that result in significantly lower power dissipation, higher propagation velocity and higher bandwidth than is possible with conventional circuits.

To describe the network traffic in a system, we adopt the notation used in [12]. According to the type of services required, the following types of traffic can be distinguished in the network:

- Guaranteed throughput (GT) is the part of the traffic for which the network has to give real-time guarantees (i.e. guaranteed bandwidth, bounded latency).

- Best effort (BE) is the part of the traffic for which the network guarantees only fairness but does not give any bandwidth and timing guarantees.

The main-stream of the communication we foresee has a guaranteed throughput character; a minor part (assumed to be less then 5%) is of best effort nature e.g. control, interrupts and configuration data. This communication has more relaxed requirements for the network and hence can use the best effort services.

If diverse wireless standards are to be supported by the SoC it requires the support of several levels of granularity for individual data-streams. We already observed that the required bandwidth between the processes of the different applications varies widely from several kbit/s (DRM) up to more than 0.5 Gbit/s (HiperLAN/2).

We distinguish between two mechanisms for transferring data between the NoC and the core processor: block-mode and streaming-mode. Some processes require all the input data to be in the local memories of the core before the execution can be started. This operation mode is called *block-mode*. Typically, a block-mode operation is done in three stages: the input data is loaded into the local memories, the process is executed and the result is fetched from the local memories and sent to another core. During the data transfers, the core is halted to make sure that the execution is not started until all data is valid. Some cores support reading input data and writing output data while they are processing, using the network interface as a slave for performing data transfers. This operation mode is called *streaming-mode*.

Depending on the standard we can use block-based communication (e.g. OFDM) or streaming-mode (e.g. UMTS) for example because the blocks get too large. Typically, during the execution of a process in streaming-mode, connections for the input data and output data remain open. This is an advantage for both the sender as well as for the receiver, since the overhead for packet assembly and re-assembly is avoided.

Whether block-mode or streaming-mode should be used, is determined by the application programmer and strongly depends on the characteristics of the application process. When the application operates in block-mode, no computation and communication occurs simultaneously. This increases the ease of programming at process level, but gives overhead at application level. For applications in streaming-mode the programmer has to carefully plan how and when the communication takes place. In general streaming-mode reduces the required buffer size.

2.5 Run-time Mapping of Streaming Applications to Multi-core Architectures

As discussed above, today's multi-core SoC architectures are composed of commercially of-the-shelf available reconfigurable intellectual property (IP) blocks.

Ultimately, we would like to design a generic heterogeneous multi-core SoC architecture that is flexible enough to run different applications (within a certain application domain). However, mapping an application to such a heterogeneous SoC is more difficult compared to mapping to a homogeneous one [5].

Today, common practice is to map applications to the architecture at design-time. In our approach we perform the mapping at run-time. Run-time mapping offers a number of advantages over design-time mapping. It offers the possibility:

- to adapt to the available resources. Only at run-time the available resources are known to the mapping algorithm. Moreover, the available resources may vary over time for example due to applications running simultaneously or adaptation of algorithms to the environment or QoS parameters (e.g. video frame rate, screen size) set by the user.

- to enable unforeseeable upgrades after first product release time, e.g. new applications and new or changing standards.

- to avoid defective parts of a SoC. Larger chip area means lower yield. The yield can be improved when the mapper is able to avoid faulty parts of the chip. Also aging can lead to faulty parts that are unforeseeable at design-time.

We assume that the mapping algorithm runs as a software process on a central (on-chip) general purpose processor that has an overview of the entire SoC which is needed for the earlier mentioned holistic approach. This mapping algorithm also generates the routes in the NoC and does the (re)configuration of the processing tiles.The mapping algorithm requires a description of the streaming applications, a library of process implementations, a description of the architecture and the current status of the system.

The objective of the run-time mapping algorithm is to determine at run-time a near-optimal mapping of the application(s) to the architecture using the library of process implementations and the current status of the system. The mapping algorithm should minimize the energy consumption and has to satisfy all the constraints of the application and the architecture e.g. real-time guarantees or bandwidth constraints. The considered problem is a combination of several optimization problems (which, on their own, are already hard) that have to be solved by light weighted methods.

The problems we consider differ from multi-processor load-balancing or scheduling mechanisms [5] because: (1) we not only consider processing but also inter-core communication. Inter-core communication is becoming a major source of energy consumption, and by optimizing the inter-core communications (i.e. placing frequently communicating processes close together on neighbouring cores) considerable energy can be saved, (2) we target at heterogeneous architectures and not just at homogeneous multi-processors, (3) we optimize for energy and not just for (time-) performance and (4) we perform the mapping at run-time. As a consequence often used scheduling techniques (such as ILP, branch and bound/price and dynamic programming [5, 10]) are not applicable and for existing heuristics (such as priority rules and local search) we carefully have to evaluate whether they can be adapted for solving at least some of the sub problems of the overall problem or not.

3. CASE STUDY: ANNABELLE SOC

As an example we will now present the Annabelle chip as depicted in figure 2. The Annabelle SoC has been developed within the 4S project and was processed with ATMEL's proprietary process (130 nm process). In November 2007 we received the first samples of the chip. A quick inspection of the samples revealed that the most important elements of the chip are fully functional. The performance figures mentioned below are still estimations; we just started to collect real measurement data. We will show how we obtained a predictable and energy-efficient SoC/NoC.

In the Annabelle SoC a conventional ARM926 architecture is complemented by ASIC blocks (for example Viterbi and DDC) and four domain specific coarse-grain reconfigurable Montium cores [8] (see figure 2). The key issue in the design of future SoC platforms for streaming applications 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.

3.1 Montium

The Montium targets the 16-bit DSP algorithm domain.

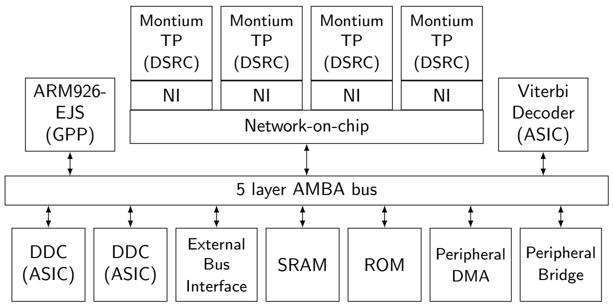

Figure 2: Block diagram of the Annabelle chip

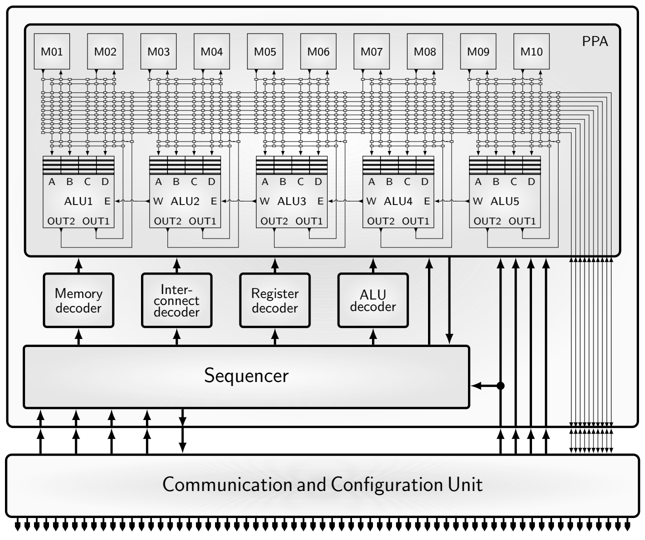

Figure 3: The Montium core

A single Montium core is depicted in figure 3. 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.

The lower part of figure 3 shows the Communication and Configuration Unit (CCU) and the upper part shows the reconfigurable Tile Processor (TP). The CCU implements the interface for off-tile communication. The off-tile interface depends on the interconnect technology that is used in the SoC.

The TP is the computing part that can be configured to implement a particular algorithm. figure 3 reveals that the hardware organization of the TP is very regular. The data path of the ALUs has a width of 16-bits and the ALUs support both signed integer and signed fixed-point arithmetic. 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 local memories imply a good locality of reference. A relatively simple sequencer controls the entire tile processor. The sequencer selects configurable tile instructions that are stored in the decoders (see figure 3).

Each local SRAM is 16-bit wide and has a depth of 1024 positions, which adds up to a storage capacity of 16Kbit per local memory. A reconfigurable Address Generation Unit (AGU) accompanies each memory. The AGUs can generate the most frequently used address patterns, but when needed also an ALU can generate address patterns. 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.

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. Neighbouring ALUs can also communicate directly: the West-output of an ALU connects to the East-input of the ALU neighbouring on the left.

3.2 CCU Network Interface

Each core processor in the SoC requires a customized network interface that is used to provide a footprint that exactly fits the NoC. As each processor operates independently, they need to be controlled separately. The ARM926 processor, the control processor, controls the other cores by sending configuration messages to their network interface (NI). Since the cores might not be running at the same clock speed as the NoC, the network interface synchronizes the data transfers.

The CCU is an interface developed to serve the Montium [14]. It provides the functionality to configure the Montium, to manage the memories by means of direct memory access (DMA) and to start/wait/reset the computation of the algorithm configured.

Both streaming-mode and block-mode communication are supported by the CCU. In block-mode, the DMA interface is used to manage data transfers. The network interface acts as a master for the tile processor, while the data transfers are directed by the control processor. In streaming-mode the algorithm's designer implicitly describes how the data transfers are performed and, therefore, the Montium controls the network interface to take care of the data flow from and to the NoC.

3.3 Predictable Network-on-Chip

For the Annabelle SoC we developed a predictable circuit-switched NoC [15] that interconnects the four Montium cores. Circuit switching has been chosen as it simplifies the NI, because it does not have to embed the data in specific network protocol and include the routing information. This is an advantage for both the sender and receiver, since there is no overhead during the communication for packaging of data (assembly or re-assembly of packets). For the Annabelle it was expected that the mapped applications, for example DRM and DAB, would be rather static with fixed information streams. Due to the semi-static behaviour of streaming applications, the connections for the input data and output data remain open during their execution.

The connections in the NoC, i.e. the routers' configuration, is controlled via the AHB bridge. The routers' control interfaces are included in the memory map of the bridge. The original proposed NoC architecture used a serialization

Table 1: Static power consumption of one Montium on Annabelle

Module	Static power [mW]
Datapath	0.09
Memories	0.06
Sequencer	0.01
Decoders	0.03
CCU	0.01
Total	0.20

Table 2: Dynamic power consumption of one Montium on Annabelle

	Energy [mW/MHz]		
Module	FIR-5	FFT-512	FFT-288
Datapath	0.19	0.24	0.15
Memories	0.0	0.27	0.21
Sequencer	0.02	0.07	0.05
Decoders	0.0	0	0.0
CCU	0.02	0.02	0.02
Total	0.23	0.60	0.43

to reduce the link wires, but this part is omitted for this small network. Furthermore, it would create a significant bottleneck compared to the large internal communication bandwidth offered by the Montium.

4. IMPLEMENTATION RESULTS

The ASIC synthesis of the Annabelle was performed using ATMEL's proprietary 130 nm process technology. For the local data memories and sequencer instruction memory of the Montium TP, embedded SRAMs are used. The embedded SRAM is an optimized component from a cell library.

For ASIC synthesis, worst case military conditions are assumed. In particular, the supply voltage is 1.1 V and the temperature is 125 °C. Results obtained with the synthesis are:

- The area of one Montium core is 3.5 mm^2 of which 0.2 mm^2 for the CCU and 3.3 mm^2 for the Montium TP.
- With Synopsys® tooling we estimated that the Montium TP, within the Annabelle ASIC realization, can implement an FIR filter at about 100 MHz or an FFT at 50 MHz. The worst case clock frequency of the Annabelle chip is 25 MHz.
- With the Synopsys Prime Power tool, we estimated the energy consumption using placed and routed netlists under nominal conditions (supply voltage is 1.2 V and temperature is 25 °C). The following section provides some of the results.

4.1 Average Power Consumption

To determine the average power consumption of the Annabelle we performed a number of power estimations on the placed and routed netlist using the Synopsis Prime Power. Table 1 gives the static power consumption. Table 2 provides the dynamic power consumption in mW/MHz of various Montium blocks for three well-known DSP algorithms. These figures show that the overhead of the sequencer and decoder is low; <16% of the total dynamic power consumption. Finally, table 3 compares the energy consumption of the Montium and the ARM926 on Annabelle. For the FIR-5 algorithm the memory is not used.

4.2 Locality of Reference

As mentioned above locality of reference is an important design parameter. One of the reasons for the excellent energy figures of the Montium is the use of locality of reference. To illustrate this, table 4 gives the amount of memory references local to the cores compared to the amount of off-core communications. These figures are as expected algorithm dependent. Therefore we chose in this table three well-known algorithms in the streaming DSP application domain: a 1024p FFT, a 200 tap FIR filter, and a part of a Turbo decoder (SISO algorithm [9]). The results show that for these algorithms 80-99% of the memory references are local (within a tile).

Table 3: Energy comparison Montium/ARM926

Algorithm	Montium [μJ]	ARM926 [μJ]	Ratio
FIR-5	0.243	–	–
FFT-112	0.357	9	25
FFT-176	0.616	16	26
FFT-256	0.707	14	20
FFT-288	1.001	23	23
FFT-512	1.563	30	19
FFT-1920	5.054	168	33

4.3 Partial Dynamic Reconfiguration

One of the advantages of a multi-core SoC organization is that each individual core can be reconfigured while the other cores are operational. In the Montium the configuration memory is organized as a RAM memory. This means that to reconfigure the Montium, not the entire configuration memory needs to be rewritten, but only the parts that are changed. Furthermore, because the Montium has a coarse-grained reconfigurable architecture, the configuration memory is relatively small. The Montium has a configuration size of only 2.6 Kbytes. Because the configuration

Table 4: Internal and external memory references per execution of an algorithm

	Number of memory references		
Algorithm	Internal	External	Ratio
1024p FFT	51200	4096	12.5
200 tap FIR	405	2	202.5
SISO alg. (N softbits)	18*N	3*N	6

Table 5: Reconfiguration of algorithms on the Montium

Algorithm	Change	Size	# cycles
1024p FFT	Scaling factors	≤150 bits	≤ 10
to iFFT	Twiddle factors	16384 bits	512
200 tap FIR	Filter coefficients	≤3200 bits	≤80

memory can be accessed as a RAM memory the system allows for dynamic partial reconfiguration. Table 5 gives some examples of reconfigurations.

5. CONCLUSIONS

In this paper we address the design issues of a reconfigurable multi-core SoC platform for streaming DSP applications. Streaming DSP applications express computation as a data flow graph with streams of data items (the edges) flowing between computation kernels (the nodes). Typical examples of streaming DSP applications are: wireless baseband processing, multi-media processing, medical image processing and sensor processing. These application domains require flexible and energy-efficient architectures. This can be realized with a multi-core architecture, in which cores are interconnected by a NoC. Energy-efficiency is realized with locality of reference and adaptivity (dynamic reconfiguration). To keep the design manageable we take a holistic view and we apply deterministic principles, for example we have a NoC that supports guaranteed throughput. Further, we presented a heterogeneous reconfigurable multi-core SoC architecture, called Annabelle, for streaming DSP applications. We show how locality of reference and partial reconfiguration works out in this architecture.

6. ACKNOWLEDGEMENTS

This work has been partly supported by the Sixth European Framework Programme as part of the 4S project under project number IST 001908.5.

The Software described in this document is furnished under a license from Synopsys International Limited. Synopsys and the Synopsys product names described herein are trademarks of Synopsys, Inc.

7. REFERENCES

[1] A. Abnous. *Low-Power Domain-Specific Processors for Digital Signal Processing*. PhD thesis, University of California Berkeley, Berkeley, CA, USA, 2001.

[2] V. Baumgarte, G. Ehlers, F. May, A. Nückel, M. Vorbach, and M. Weinhardt. PACT XPP – A self-reconfigurable data processing architecture. *The Journal of Supercomputing*, 26(2):167–184, 2003.

[3] W. J. Dally, U. J. Kapasi, B. Khailany, J. H. Ahn, and A. Das. Stream processors: Progammability and efficiency. *Queue*, 2(1):52–62, 2004.

[4] European Telecommunication Standard Institute (ETSI). *Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Physical (PHY) layer*, ETSI TS 101 475 v1.2.2 edition, February 2001.

[5] Y. Guo, G. J. M. Smit, and P. M. Heysters. Template generation and selection algorithms. In W. Badaway and Y. Ismail, editors, *Proceedings of International Workshop on System-on-Chip for Real-Time Applications*, pages 2–5, Los Alamitos, California, June 2003. IEEE Computer Society.

[6] I. Held and B. Vandewiele. AVISPA-CH embedded communications signal processor for multistandard digital television, March 29-30 2006.

[7] P. M. Heysters. *Coarse-Grained Reconfigurable Processors - flexibility meets efficiency*. PhD thesis, University of Twente, September 2004.

[8] P. M. Heysters and G. J. M. Smit. Mapping of dsp algorithms on the montium architecture. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 180.2, Washington, DC, USA, 2003. IEEE Computer Society.

[9] P. M. Heysters, L. T. Smit, G. J. M. Smit, and P. J. M. Havinga. Max-log-map mapping on an fpfa. In *Proceedings of the 2005 International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'02)*, pages 90–96, Las Vegas, NV, USA, June 2002. CSREA Press.

[10] M. L. Pinedo. *Planning and Scheduling in Manufacturing and Services*. Springer Series in Operations Research and Financial Engineering. Springer, 2005.

[11] G. K. Rauwerda, P. M. Heysters, and G. J. M. Smit. Towards software defined radios using coarse-grained reconfigurable hardware. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(1):3–13, January 2008.

[12] E. Rijpkema, K. Goossens, A. Rădulescu, J. Dielissen, J. van Meerbergen, P. Wielaga, and E. Waterlander. Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip. *IEE Proceedings: Computers and Digital Techniques*, 150(5):294–302, sep 2003.

[13] L. T. Smit. *Energy-Efficient Wireless Communication*. PhD thesis, University of Twente, Enschede, The Netherlands, December 2003.

[14] M. D. van de Burgwal, G. J. M. Smit, G. K. Rauwerda, and P. M. Heysters. Hydra: an energy-efficient and reconfigurable network interface. In *Proceedings of the 2006 International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'06)*, pages 171–177, Las Vegas, NV, USA, June 2006. CSREA Press.

[15] P. T. Wolkotte, G. J. M. Smit, G. K. Rauwerda, and L. T. Smit. An energy-efficient reconfigurable circuit-switched network-on-chip. In *Proceedings of the 12th Reconfigurable Architecture Workshop (RAW 2005)*, page 155, Washington, DC, USA, April 2005. IEEE Computer Society.