Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2007, Article ID 78082, 10 pages
doi:10.1155/2007/78082

Research Article

The Chameleon Architecture for Streaming DSP Applications

Gerard J. M. Smit,[1] André B. J. Kokkeler,[1] Pascal T. Wolkotte,[1] Philip K. F. Hölzenspies,[1] Marcel D. van de Burgwal,[1] and Paul M. Heysters[2]

[1] *Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, Drienerlolaan 5, 7522 NB Enschede, The Netherlands*
[2] *Recore Systems, Capitool 22, 7521 PL Enschede, The Netherlands*

Received 15 May 2006; Revised 20 December 2006; Accepted 20 December 2006

Recommended by Neil Bergmann

We focus on architectures for streaming DSP applications such as wireless baseband processing and image processing. We aim at a single generic architecture that is capable of dealing with different DSP applications. This architecture has to be energy efficient and fault tolerant. We introduce a heterogeneous tiled architecture and present the details of a domain-specific reconfigurable tile processor called Montium. This reconfigurable processor has a small footprint (1.8 mm^2 in a 130 nm process), is power efficient and exploits the locality of reference principle. Reconfiguring the device is very fast, for example, loading the coefficients for a 200 tap FIR filter is done within 80 clock cycles. The tiles on the tiled architecture are connected to a Network-on-Chip (NoC) via a network interface (NI). Two NoCs have been developed: a packet-switched and a circuit-switched version. Both provide two types of services: guaranteed throughput (GT) and best effort (BE). For both NoCs estimates of power consumption are presented. The NI synchronizes data transfers, configures and starts/stops the tile processor. For dynamically mapping applications onto the tiled architecture, we introduce a run-time mapping tool.

1. INTRODUCTION

Streaming DSP algorithms are becoming more common in portable embedded systems and require an efficient processing architecture. Typical streaming DSP examples are found in signal processing for phased array antennas (for radar and radio astronomy), wireless baseband processing (for HiperLAN/2, WiMax, DAB, DRM, DVB, UMTS [1, 2]), multimedia processing (encoding/decoding), MPEG/TV, medical image processing, and sensor processing (e.g., remote surveillance cameras and automotive). Streaming DSP algorithms (sometimes modeled as synchronous dataflow programs) express computation as a dataflow graph with streams of data items (the edges) flowing between computation kernels (the nodes). Most signal processing applications can be naturally expressed in this style [3].

Analyzing the common characteristics of typical streaming DSP applications, we made the following observations.

(i) These applications are characterized by relatively simple local processing on a huge amount of data. The trend is that energy costs for data communication dominate the energy costs of processing.

(ii) Data blocks arrive at nodes at a fixed rate, which causes periodic data transfers between successive processing elements. The rate at which blocks arrive is application dependent, for example, 4 μs for HiperLAN/2 and 20 milliseconds for DRM.

(iii) The size of the data blocks transported over the edges is application dependent, for example, 14-bit samples for a sensor system, 64 32-bit words for HiperLAN/2 [1] OFDM symbols or 1,024 $\times$ 768 $\times$ 24 bit frames for a video application. The required communication bandwidth for the edges is also application dependent so a large variety in communication bandwidth is needed.

(iv) Data flows through the successive nodes in a pipelined fashion. Nodes work in parallel on parallel processors or can be time multiplexed on one or more processors. Thus, streaming applications show a predictable temporal and spatial behavior.

(v) For our application domains, throughput guarantees (in data items per second) are typically required for the communication as well as for the processing.

(vi) In general, the amount of processing is fixed for each sample, however, in some applications the amount of processing per sample is data dependent (nonmanifest) [4].

(vii) The lifetime of a communication stream is semi-static, which means that a stream is fixed for a relatively long time.

In the examples mentioned above, streaming DSP algorithms dominate the use of the processing and communication resources.

This paper focuses on our research activities concerning efficient architectures for streaming DSP applications. In Section 2, we present our vision on the design of architectures and deduct the design paradigms. Our ideas have led to a heterogeneous tiled architecture resulting in a System-on-Chip (SoC) named Annabelle (see Section 3). One part of this architecture is a domain specific reconfigurable Core (DSRC) which is described in Section 4. The Network-on-Chip (NoC) interconnects the different parts of the tiled architecture and is described in Section 5. A network interface connects the DSRC to the NoC (see Section 6). Section 7 reflects our ideas on mapping applications onto a tiled architecture.

2. REQUIREMENTS AND DESIGN PARADIGMS

In our vision, architectures for streaming DSP applications have to satisfy the following requirements.

(i) A single architecture has to be capable of dealing with multiple DSP applications efficiently.
(ii) The architecture has to be fault tolerant.
(iii) The architecture has to be energy efficient.

Based on these requirements, we developed paradigms for the design of a SoC for streaming DSP applications. Below we elaborate on the requirements and the design paradigms in more detail.

2.1. Capable of dealing with multiple DSP applications

The set of applications that will run on a future processing architecture is not fixed but changes over time. The architecture has to be reconfigurable to allow different mixtures of applications without realizing all possible mixtures in hardware.

In streaming DSP applications, parts of the application may be executed in parallel which implies that an architecture may consist of multiple processing cores. Because we aim to support a wide variety of DSP applications in an efficient way, we need a heterogeneous architecture where most processing cores require configurability/programmability. Some parts of an application run more efficiently on bit-level reconfigurable architectures (e.g., PN-code generation), some on general purpose architectures and some perform optimal on word-level reconfigurable platforms (e.g., FIR filters or FFT algorithms). For the design of processing architectures for streaming DSP applications, it is crucial that multiple processing cores (tiles) are present on one SoC to enable parallel execution of different parts of the application and that these tiles show different levels of configurability/programmability and granularity. This results in a tiled heterogeneous SoC where the tiles are interconnected via an NoC (see also [5]).

There are basically two timescales for reconfiguration: long term and short term. Long term reconfiguration eases upgrading a system with a new or enhanced application or standard. Short-term (or dynamic) reconfiguration refers to the ability of a system to adapt dynamically to changing environmental conditions. Short-term reconfiguration can, for example, be applied in RAKE receivers where, depending on the radio channel conditions, the receiver switches to different configurations [6]. Dynamic reconfiguration poses more stringent requirements and requires a run-time mapping tool which is described in Section 7.

2.2. Fault tolerant

The processing architecture has to be fault tolerant to improve the yield of the production process and to extend the lifetime of the system once operational. Faults in one tile should not lead to a malfunctioning SoC device but should only lead to limited performance of a functionally correct device (graceful degradation). An example of a fault-tolerant reconfigurable architecture can be found in [7].

When one of the tiles on a tiled heterogeneous SoC is discovered to be defect (either due to a manufacturing fault or discovered at operating time by built-in-diagnosis) this defective tile can be switched off and isolated. The dataflow can be rerouted to another tile which can take over the tasks of the faulty tile. Also for graceful degradation, a runtime mapping of tasks to tiles is required. A tiled approach also eases verification of an integrated circuit design since the design of identical tiles only has to be verified once.

2.3. Energy efficiency

Portable devices very often run streaming DSP applications, for example, for wireless baseband or multimedia processing. 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 computation for wireless protocol processing and multimedia applications, but the energy content of batteries is only increasing 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 server clusters (e.g., Google has an annual energy budget of 50 million dollars).

General purpose processors (GPPs) are in general not suitable for applications that require energy efficiency because of the need to fetch every instruction from memory and because of the extra hardware overhead to improve performance. Even though digital signal processors (DSPs) are tailored towards executing the algorithms of streaming applications, their energy efficiency is limited because they also

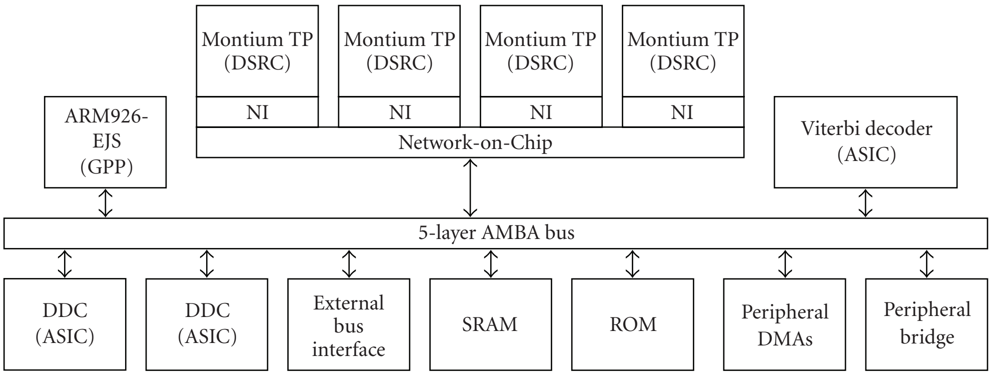

Figure 1: Blockdiagram of the Annabelle chip.

have to fetch and decode every instruction from memory. Field programmable gate arrays (FPGAs) do not need to fetch instructions but the bit-level programmability causes the word-level operations of streaming applications to become relatively inefficient.

In general, to reduce power consumption of a processor, the off-chip access to, for example, main memory should be reduced. Even on-chip data transfer should be limited: transporting a signal over a 1 mm wire in a 50 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 a 32-bit operation!) [8]. Since references to memory in streaming applications typically display a high degree of temporal and spatial locality, a tiled architecture where each tile contains its own local memory exploits the locality of reference principle and improves the energy efficiency.

Energy is also saved by simply switching off tiles that are not being used. This also helps to reduce the static power consumption. Moreover, a tile processor might not need to run at full clock speed to achieve the required QoS at a particular moment in time, also reducing power consumption.

The requirements and design paradigms have led to an architecture for streaming DSP applications which is presented in the next section.

3. HETEROGENEOUS TILED ARCHITECTURES

Recently, a number of heterogeneous, reconfigurable SoC architectures have been proposed for the streaming DSP application domain. Examples are the Avispa [9]; the PACT-XPP [10]; the Maya chip from Berkeley [3, 11], and the Chameleon/Montium architecture from the University of Twente/Recore Systems [12]. For an overview we refer to [13].

In the 4S project [14], we have developed a prototype chip, called Annabelle, for streaming DSP applications (see Figure 1).

It consists of an ARM926 processor with a 5-layer AMBA bus, 4 Montium TPs (Montium tile processors), a Viterbi decoder, two digital down converters (DDCs), memory and external connections. The Montium TPs are connected to a Network on Chip (NoC) via a network interface (NI). The SoC is fabricated in 130 nm CMOS technology and occupies 50 mm^2. The size of the 4 Montium TPs (without SRAM), the 4 NIs, and the NoC is 12 mm^2. The layout of the Annabelle chip has been finalized and we expect the first prototype chips to be delivered in spring 2007.

We focus on the design of the Montium TP DSRC. Besides the development of the Montium TP, we discuss the design of the NoC and the NI and briefly address our activities concerning the mapping of applications onto a heterogeneous tiled architecture.

4. THE MONTIUM TP

The key issue in the design of future 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. Our effort to find such a balance resulted in the Montium architecture. The Montium is described in detail in [13] and in this section, we only discuss its general structure. The Montium architecture is an example of a domain specific reconfigurable core (DSRC). It is a parameterizable architecture, described in a hardware description language where, for example, memory depth and width of the data paths are important parameters which have to be fixed just before fabrication. A single Montium processing tile, including network interface (NI) is depicted in Figure 2.

The lower part of Figure 2 shows the NI which deals with the off-tile communication and configuration of the upper part, the reconfigurable tile processor (TP). The definition of the NI depends on the interconnect technology that is used in the SoC (see Section 6).

The TP is the computing part that can be dynamically reconfigured to implement a particular algorithm. At first glance the TP has a VLIW structure. However, the control structure of the Montium is very different. For (energy) efficiency it is imperative to minimize the control overhead. This is, for example, accomplished by scheduling instructions statically at compile time. A relatively simple sequencer

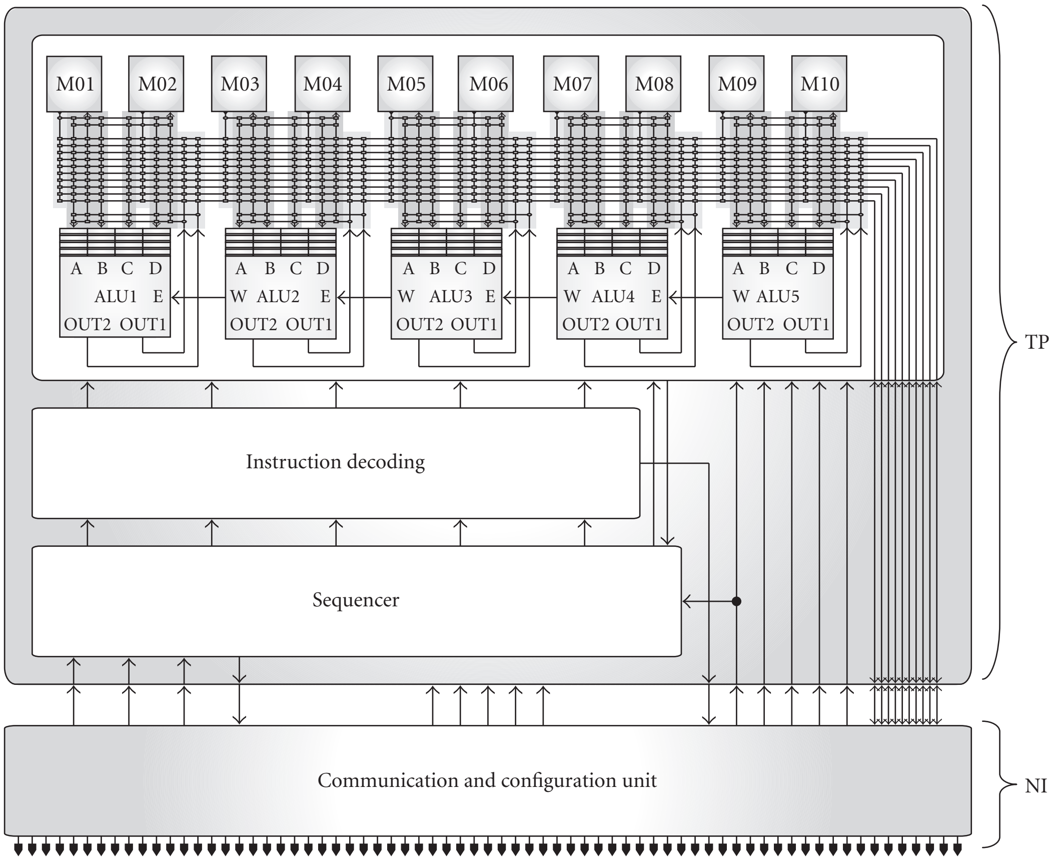

Figure 2: The Montium tile processor and network interface.

controls the entire tile processor. The sequencer selects configurable tile instructions that are stored in the instruction decoding block (see Figure 2).

Furthermore, we see multiple ALUs (ALU1, . . . , ALU5) and multiple memories (M01, . . . , M10). A single ALU has four inputs (A, B, C, D). Each input has a private input register file that can store up to four operands. The input register file cannot be bypassed, that is, 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 outputs (OUT1, OUT2), which are connected to the interconnect. The ALU is entirely combinational and consequently there are no pipeline registers within the ALU. Neighboring ALUs can also communicate directly: the west output (W) of an ALU connects to the east input (E) of the ALU neighboring on the left.

The ALUs support both signed integer and signed fixed-point arithmetic. The five identical ALUs 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 in parallel.

An address generation unit (AGU, not shown in Figure 2) accompanies each memory. The AGU can generate the typical memory access patterns found in common DSP algorithms, for example, incremental, decremental, and bit-reversal addressing. It is also possible to use the memory as a lookup table for complicated functions which 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.

The reconfigurable elements within the Montium are the sequencer, the instruction decoding block, and the AGUs. Their functionality can be changed at run-time. The Montium is programmed in two steps. In the first step, the AGUs are configured and a limited set of instructions is defined by configuring the instruction decoding block. The sequencer is then instructed to sequentially select the required instructions. A predefined instruction set is available using an assembly type of mnemonics (Montium assembly). A compiler has been constructed to convert a Montium assembly program into configuration data for both the instruction decoding block and the sequencer.

4.1. Implementation results

The ASIC synthesis of the Montium TP was performed using a 130 nm CMOS process technology. The figures given in this section are from [13] and are based on a different technology than the figures for the Annabelle chip (see Section 3), although the feature size is equal. For the local data memories and sequencer instruction memory of the Montium TP, embedded SRAMs are used. Since the Montium targets the

16-bit digital signal processing domain, the width of the data paths of the ALUs is set to 16-bits. Each local memory is 16-bit wide and has a depth of 1024 positions, which adds up to a storage capacity of 16 Kbit per local memory.

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 following.

(i) The area of the Montium TP is about 1.8 mm^2.
(ii) With Philips tools we estimated that the Montium TP ASIC realization can implement an FIR filter at about 140 MHz or an FFT at 100 MHz. The maximum frequencies differ for different applications because each application leads to a specific configuration of the ALUs which results in different lengths of critical paths.

4.1.1. Average power consumption

Figure 3 depicts an energy comparison between various architectures (see [15–17]) executing FFT butterflies. The figures are normalized to 1 MHz clock rate and one FFT butterfly per clock cycle. The average absolute dynamic power consumption of a particular architecture can be obtained by multiplying the normalized average power with the clock frequency (in MHz). The results can also be found in [13].

This figure clearly shows the energy advantage of coarse-grain reconfigurable architectures. Note that this can also be observed from the Xilinx Virtex-II Pro implementations. The use of the coarse-grain multiplier blocks (see Figure 3, design A) improves the energy consumption of the FFT computation considerably compared to an implementation without coarse-grain multiplier blocks (see Figure 3, design B).

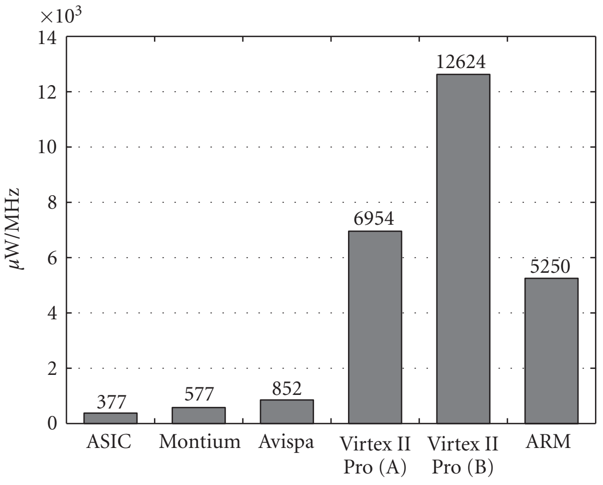

Figure 3: Average dynamic power consumption when FFT butterflies are computed at a rate of 1 MHz.

4.1.2. Locality of reference

Exploiting locality of reference is an important design paradigm. The Montium therefore contains 10 local memories. These memories are used to limit the number of off-tile memory operations. Table 1 gives the amount of memory operations local to the tile processors (on-tile operations) and the amount of off-tile operations.

These figures are algorithm dependent. Therefore, we chose in this table three algorithms in the streaming DSP application domain: a 1024 point FFT, a 200 tap FIR filter, and a part of a turbo decoder (SISO algorithm [18]). The results show that for these algorithms most memory references are local (within a tile) because of the presence of 10 local memories.

4.1.3. Differential reconfiguration

In a tiled SoC, each individual tile can be reconfigured while the other tiles are operational. In the Montium, the configuration memory is organized as RAM. This means that to reconfigure the tile, not the entire configuration memory needs to be written but only the parts that are changed, facilitating differential reconfiguration. Before reconfiguration, the NI freezes the program counter of the sequencer. The configuration memories are updated and the NI starts the sequencer again. Furthermore, because the Montium has a coarse-grained reconfigurable architecture, the configuration memory is relatively small (2.6 Kbytes). So, completely reconfiguring a Montium requires less than 1350 clock cycles using 16-bit reconfiguration words. A typical reconfiguration file contains 1 Kbytes. Table 2 gives some examples of reconfigurations.

To reconfigure a Montium from executing a 1024 point FFT to executing a 1024 point inverse FFT requires updating the scaling and twiddle factors. Updating these factors requires less than 522 clock cycles in total. To change the coefficients of a 200 tap FIR filter requires less than 80 clock cycles.

5. NETWORK-ON-CHIP

A tiled architecture for streaming DSP applications has to be supported by a predictable Network-on-Chip (NoC). In a NoC, each processing tile is connected to a router. Routers of different processing tiles are interconnected. Communication between two processing tiles involves at least the two routers of the corresponding processing tiles but other routers might be involved as well. 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 a bus (see also [19]). To describe the network traffic in a system, we adopt the notation used in [20]. According to the type of services required, the following types of traffic can be distinguished in the network.

Table 1: On-tile and off-tile memory operations per execution of an algorithm.

Algorithm	On-tile memory ops. no.			Off-tile memory ops. no.		
	Read	Write	Total	Read	Write	Total
1024p FFT	30720	20480	51200	2048	2048	4096
200 tap FIR	400	5	405	1	1	2
SISO alg. (N softbits)	10*N	8*N	18*N	2*N	N	3*N

Table 2: Reconfiguration of algorithms on the Montium.

Algorithm	Change	Size	Cycles no.
1024p FFT to inverse FFT	Scaling factors	$\leq 10*15 = 150$ bits	≤ 10
	Twiddle factors	$2*512*16 = 16384$ bits	512
200 tap FIR	Filter coefficients	$\leq 200*16 = 3200$ bits	≤ 80

(i) 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).

(ii) 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.

For streaming DSP applications most traffic is in the GT category. Besides the main stream of GT communication we foresee a minor part (assumed to be less then 5%) of BE communications, for example, control, interrupts, and configuration data.

For the NoC we first defined a virtual channel wormhole router packet-switched network [21] and later we developed a circuit-switched network [22] with a separate best-effort network [23]. Both NoCs support GT traffic as well as BE traffic. For the GT traffic, guaranteed latencies are supported.

First, we discuss the effects of varying load on GT and BE traffic for the packet-switched network in a 6×6 configuration. Second, we underpin our decision to develop a circuit-switched network and third we present simulation results concerning power consumption for both the packet-switched and circuit-switched NoCs.

5.1. *Packet-switched NoC*

In the context of our work on heterogeneous reconfigurable SoC architectures, we developed a predictable virtual channel wormhole router packet-switched NoC [21]. This NoC supports both GT traffic as well as BE traffic. For the GT traffic, guaranteed latencies are supported. Figure 4 presents simulation results for a 6×6 NoC.

The graph shows how the latency of the GT and BE messages depends on the offered BE load. For the GT traffic, the mean and the maximal latency of packets are given. When the offered BE load is low, the latency of the GT packets is lower than the guaranteed (or allowed) latency. The reason is that the GT traffic utilizes the bandwidth unused by the BE traffic. The latency of the GT packets is higher than the latency of the BE traffic because the GT packets are larger (256 bytes against 10 bytes for BE packets). With the increase of the BE load, the latency of the GT traffic increases too until the maximum delay reaches the guarantee. Further increase of the BE load increases the GT mean latency but the GT maximum latency never exceeds the guaranteed latency.

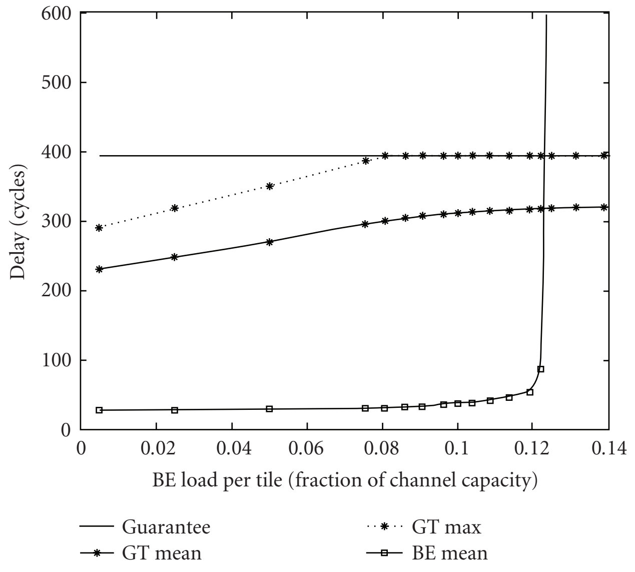

Figure 4: Message delay of the GT and BE traffic versus BE load for 6-by-6 network.

5.2. *Circuit-switched NoC*

The streaming applications, as mentioned in Section 1, show that the amount of expected GT traffic is much larger than the amount of BE traffic. Fundamentally, sharing resources and giving guarantees are conflicting, and efficiently combining guaranteed traffic with best-effort traffic is hard [24]. Using dedicated techniques for both types of traffic, we aim for reducing the total area and power consumption by means of a circuit-switched NoC.

As mentioned, we defined a circuit-switched NoC after we developed a packet-switched NoC. The reasons for reconsidering circuit switching are that the flexibility of packet switching is not needed, because a connection between two tiles will remain open for a long period (e.g., seconds or longer). This in contrast with connections in a local area network. Furthermore, large amounts of the traffic between tiles will need a guaranteed throughput, which is easier to guarantee in a circuit-switched connection. Circuit switching

Table 3: Scenario definitions.

No.	No. of streams	Comment
1	0	The router is idle
2	1	Stream from and to other router
3	1	Stream from other router to processing tile
4	1	Stream processing tile to other router
5	2	Combination of 3 and 4
6	3	Combination of 2, 3, and 4
7	5	Combination of 5 and three times 2
8	10	Two times the number of streams of 7
9	15	Three times the number of streams of 7
10	20	All the lanes/virtual channels are occupied

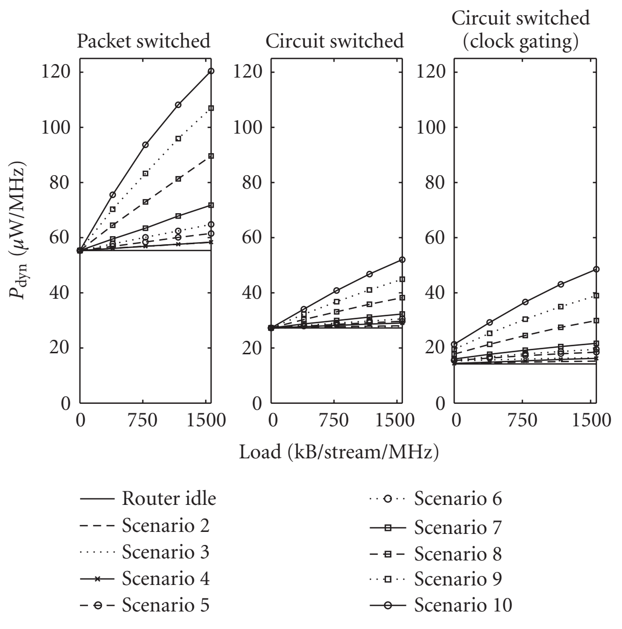

Figure 5: Energy consumption of routers for typical data (random data with 50% bit-flips).

also eases the implementation of asynchronous communication techniques, because data and control can be separated and circuit switching has a minimal amount of control in the data path (e.g., no arbitration). This increases the energy efficiency per transported bit and the maximum throughput. Further, we see some benefits of a circuit-switched NoC when GT traffic has to be scheduled. Scheduling communication streams over nontime multiplexed channels is easier because, by definition, a stream will not have collisions with other communication streams. The Æthereal [25] and SoCBUS [26] routers have large interaction between data streams (both have to guarantee contention free paths). Determining the static time slots table for these systems requires considerable effort. Because data streams are physically separated in a circuit-switched NoC, collisions in the crossbar do not occur. Therefore, we do not need buffering and arbitration in the individual router. An established physical channel can always be used. Because of the above reasons we used a circuit-switched NoC in the Annabelle chip, which was introduced in Section 3.

5.3. Power measurements

For both NoCs, we analyzed the power consumption under variable network loads. We defined a set of test scenarios for traffic patterns. We used random data with 50% bit-flips. Furthermore, to vary the amount of traffic which concurrently traverses the router of a processing tile, we defined ten scenarios. The scenarios have a variable number of concurrent data streams with a variable load between 0% and 100%. The ten scenarios are listed in Table 3.

The first scenario is a situation where no data traverses the router during the time of the simulation. This gives the static offset in the dynamic power consumption. The other scenarios simulate one or more concurrent data streams.

Power estimation of the packet-switched and circuit-switched networks is performed by modeling the designs in VHDL. The synthesized VHDL design is then annotated via a set of test scenarios. We can estimate the power consumption per scenario using Synopsis power compiler and the annotated design. For both network solutions all 10 scenarios are applied. In each scenario, the data streams use the guaranteed throughput protocol of the router. The power consumption of the router is measured over 20 kB of data that is offered to the router in a variable time interval. The variable interval is used to change the average load of the link. We measured the power consumption per MHz [μW/MHz] for each scenario and different load.

The left graph of Figure 5 depicts the dynamic power consumption of the packet-switched network depending on the offered load for typical data. The middle graph of Figure 5 depicts the dynamic power consumption of the circuit-switched network + best-effort router depending on the offered load for typical data. The power consumption of the extra required best-effort network is measured with a separate testbench [23]. The power consumption of this small extra router varied between 8.4 and 12.3 μW/MHz. In this paper, we use the measurement of the GT traffic and added the worst-case power consumption of the BE network (12.3 μW/MHz) to find the worst-case power consumption of the combination. We noticed a relatively high offset in the dynamic power consumption. This could be reduced by including clock gating to switch off the inactive lanes. This resulted in the right graph of Figure 5, where the remaining offset is mainly determined by the best-effort network.

6. HYDRA NETWORK INTERFACE

A tile that is connected to a NoC requires a customized network interface (NI) that provides a footprint that exactly fits the NoC (see Figure 1). The Hydra is a NI, developed to serve the Montium within the Annabelle chip [27].

The Montium tiles in the Annabelle chip operate independently so they need to be controlled separately. Since the tile processors can be operated at different clock frequencies, the NIs synchronize the data transfers between the tile processors and the NoC.

The Hydra uses a light-weight message protocol that includes 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 configured algorithm.

6.1. Communication modes

We distinguish two mechanisms for transferring data between the circuit-switched NoC and the Montium: block mode and streaming mode. Some applications require all the input data to be stored in the local memories before the execution can be started. This operation mode is called block mode. Typically, a block-mode operation is performed in three stages: the input data is loaded into the local memories (DMA load), the process is executed, and the result is fetched from the local memories (DMA retrieve) and sent to another tile processor. During the data transfers, the tile processor is halted to make sure the execution is not started until all data are valid. Some tile processors 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*.

Dependent on the application, we can use block-mode communication or we may have to use streaming-mode communication because the blocks get too large to fit in the local memories. Due to the semistatic behavior of streaming applications (see Section 2), the connections for the input data and output data remain open during their execution. This is an advantage for both the sender and receiver, since there is no overhead during the communication for packaging of data (assembly or reassembly of packets).

Whether block-mode or streaming-mode is used is determined by the application programmer by properly configuring the Hydra NI. Clearly, this strongly depends on the characteristics of the application process. When the application operates in block mode, no computation and communication can be done at the same time. This increases the ease of programming at process level, but gives some overhead at application level. For streaming-mode communication, however, the application designer has to embed the communication patterns inside the instructions such that the Montium controls the NI to take care of the data flow from and to the NoC. In streaming mode, the communication pattern is part of the instructions that are programmed in the Montium and, therefore, directly related to the program flow.

6.2. Implementation results

In the implementation of the Hydra NI on the Annabelle chip, at most four 16-bit input streams and four 16-bit output streams can be handled in parallel. So, four 16-bit words can be read from the network and four 16-bit words can be written to the network simultaneously at the rate of the Montium clock.

For the synthesis of the Hydra, we used the same ASIC technology as mentioned before. To be sure that the Hydra would not become a bottleneck in clock frequency, the maximum clock frequency for the synthesis was constrained to 200 MHz. With this constraint, the area is 0.106 mm^2, which is less than 6% of the area of the Montium.

7. RUNTIME MAPPING OF STREAMS TO ARCHITECTURE

Ultimately, we target a dynamically reconfigurable heterogeneous 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 than mapping to a homogeneous architecture [28, 29]. Today, it is common practice to map the applications to the architecture at design time. An alternative approach is to map applications at run time. In [30], a run-time partitioning system is introduced, aiming at the run-time hardware/software partitioning and mapping of a single application (thread). In [31], run-time scheduling of multiple threads onto reconfigurable hardware is presented. In our approach we also want to address spatial mapping.

Currently, we are ramping up our research efforts on run-time mapping and as reported in [32] the first results are promising. In this section we describe our plans on how to perform the mapping at run time.

Run-time mapping offers a number of advantages over design-time mapping. It offers, for example, the possibility to adapt to the available resources. If we allow applications to run simultaneously and if we allow adaptation of algorithms to the environment or QoS parameters set by the user or the applications (e.g., video frame rate, screen size), the required resources may vary over time. Then, only at run time the required resources are known and to optimally use the available resources, we need a run-time mapping algorithm.

Furthermore, run-time mapping offers the possibility to increase yield. The yield of an SoC can be improved when the run-time mapping tool is able to avoid faulty parts of the chip. Also aging can lead to faulty parts that are unforeseeable at design time.

In our approach, the mapping algorithm maps applications to a heterogeneous SoC architecture at run time. We assume that the mapping algorithm runs as a software process on a central coordination node (CCN). This CCN 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 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, for example, real-time guarantees or bandwidth constraints. The considered problem is a combination of several optimization problems (which, on their own, are already hard) that has to be solved by light-weighted methods.

The problems we consider differ from multiprocessor scheduling or load-balancing mechanisms [33] because:

(1) besides processing, we also consider inter-tile communication. Inter-tile communication is becoming a major source of energy consumption, and by optimizing the inter-tile communications (i.e., placing frequently communicating processes close together) considerable energy can be saved,
(2) we target at heterogeneous architectures and not just at homogeneous multiprocessors,
(3) we optimize for energy and not just for (time) performance. As a consequence, often-used scheduling techniques such as ILP, branch and bound/price, and dynamic programming [20, 28], are not applicable and for existing heuristics such as priority rules and local search, we carefully have to evaluate whether they are adaptable for solving at least some of the sub problems of the overall problem.

8. CONCLUSION

This paper addresses the design issues of a reconfigurable SoC platform and the supporting software tools for streaming DSP applications. Streaming DSP applications can be modeled as a dataflow graph with streams of data items (the edges) flowing between computation kernels (the nodes). Typical examples of streaming DSP applications are wireless baseband processing, multimedia processing, medical image processing, and sensor processing. These application domains need flexible and energy-efficient architectures. This can be realized with a tiled architecture, in which tiles are interconnected by a Network-on-Chip (NoC). Energy efficiency is realized with locality of reference and dynamic reconfigurations. To keep the design manageable, we have a NoC that supports both guaranteed throughput traffic (GT) as well as best effort traffic (BE). Furthermore, future operating systems have to support runtime mapping of streaming tasks onto a tiled SoC.

ACKNOWLEDGMENT

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

REFERENCES

[1] ETSI, "Broadband Radio Access Networks (BRAN); HiperLAN type 2; Physical (PHY) layer," ETSI TS 101 475 V1.2.2 (2001-2002), 2001.

[2] T. Ojanperä, *Wideband CDMA for Third Generation Mobile Communications*, The Artech House Universal Personal Communications Series, Artech House, Norwood, Mass, USA, 1998.

[3] W. J. Dally, U. J. Kapasi, B. Khailany, J. H. Ahn, and A. Das, "Stream processors: programmability with efficiency," *ACM Queue*, vol. 2, no. 1, pp. 52–62, 2004.

[4] O. Mansour, *High level synthesis for non-manifest digital signal processing applications*, Ph.D. thesis, University of Twente, Twente, The Netherlands, 2006.

[5] R. Tessier and W. Burleson, "Reconfigurable computing for digital signal processing: a survey," *The Journal of VLSI Signal Processing*, vol. 28, no. 1-2, pp. 7–27, 2001.

[6] M. Latva-aho, M. Juntti, and I. Oppermann, "Reconfigurable adaptive RAKE receiver for wideband CDMA systems," in *Proceedings of the 48th IEEE Vehicular Technology Conference (VTC '98)*, vol. 3, pp. 1740–1744, Ottawa, Ontario, Canada, May 1998.

[7] P. K. Lala and A. Walker, "An on-line reconfigurable FPGA architecture," in *Proceedings of the 15th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT '00)*, pp. 275–280, Yamanashi, Japan, October 2000.

[8] I. Bolsens, "Challenges and opportunities for FPGA platforms," in *Proceedings of the 12th International Conference on Field-Programmable Logic and Applications (FPL '02)*, pp. 391–392, Montpellier, France, September 2002.

[9] I. Held and B. VanderWiele, "Avispa-CH - embedded communications signal processor for multi-standard digital television," GSPx TV to Mobile, March 2006.

[10] V. Baumgarte, F. May, A. Nuckel, M. Vorbach, and M. Weinhardt, "PACT XPP—a self-reconfigurable data processing architecture," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA '01)*, pp. 64–70, Las Vegas, Nev, USA, June 2001.

[11] A. Abnous, "Low-power domain-specific processors for digital signal processing," Ph.D dissertation, University of California, Berkeley, Calif, USA, 2001.

[12] P. M. Heysters and G. J. M. Smit, "Mapping of DSP algorithms on the MONTIUM architecture," in *Proceedings of Reconfigurable Architectures Workshop (RAW '03)*, Nice, France, April 2003.

[13] P. M. Heysters, *Coarse-grained reconfigurable processors - flexibility meets efficiency*, Ph.D. thesis, University of Twente, Twente, The Netherlands, 2004.

[14] G. J. M. Smit, E. Schuler, J. E. Becker, J. Quevremont, and W. Brugger, "Overview of the 4S project," in *Proceedings of the International Symposium on System-on-Chip (SoC '05)*, pp. 70–73, Tampere, Finland, November 2005.

[15] G. Burns, P. Gruijters, J. Huisken, and A. van Wel, "Reconfigurable accelerator enabling efficient SDR for low-cost consumer devices," in *SDR Technical Forum*, Orlando, Fla, USA, November 2003.

[16] *Virtex II Pro and Virtex II pro X FPGA UserGuide*, March 2005, Xilinx.

[17] K. Yarlagadda, "ARM refocuses DSP effort," Microprocessor report, Micro-design resources, Consorci de Biblioteques Universitàries de Catalunya, Barcelona, Spain, June 1999.

[18] 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 International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA '02)*, pp. 90–96, Las Vegas, Nev, USA, June 2002.

[19] H. Zhang, M. Wan, V. Gearge, and J. Rabaey, "Interconnect architecture exploration for low-energy reconfigurable single-chip DSPs," in *Proceedings of the IEEE Computer Society Workshop on VLSI (WVLSI '99)*, p. 2, Orlando, Fla, USA, April 1999.

[20] K. Goossens, J. van Meerbergen, A. Peeters, and R. Wielage, "Networks on silicon: combining best-effort and guaranteed services," in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE '02)*, pp. 423–425, Paris, France, March 2002.

[21] N. Kavaldjiev, G. J. M. Smit, P. G. Jansen, and P. T. Wolkotte, "A virtual channel network-on-chip for GT and BE traffic," in *Proceedings of IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI '06)*, pp. 211–216, Karlsruhe, Germany, March 2006.

[22] 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 Architectures Workshop (RAW '05)*, Denver, Colo, USA, April 2005.

[23] P. T. Wolkotte, G. J. M. Smit, and J. E. Becker, "Energy-efficient NOC for best-effort communication," in *Proceedings of the 15th International Conference on Field Programmable Logic and Applications (FPL '05)*, pp. 197–202, Tampere, Finland, August 2005.

[24] J. Rexford and K. G. Shin, "Support for multiple classes of traffic in multicomputer routers," in *Proceedings of the 1st International Workshop on Parallel Computer Routing and Communication (PCRCW '94)*, pp. 116–130, Springer, Seattle, Wash, USA, May 1994.

[25] J. Dielissen, A. Rădulescu, K. Goossens, and E. Rijpkema, "Concepts and implementation of the phillips network-on-chip," in *IP-Based SOC Design*, Grenoble, France, November 2003.

[26] D. Wiklund and D. Liu, "Socbus: switched network on chip for hard real time embedded systems," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS '03)*, p. 78a, Nice, France, April 2003.

[27] 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 International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA '06)*, pp. 171–177, Las Vegas, Nev, USA, June 2006.

[28] Y. Guo, *Mapping applications to a coarse-grained reconfigurable architecture*, Ph.D. thesis, University of Twente, Twente, The Netherlands, 2006.

[29] M. Koester, M. Porrmann, and H. Kalte, "Task placement for heterogeneous reconfigurable architectures," in *Proceedings of IEEE International Conference on Field Programmable Technology (FPT '05)*, pp. 43–50, Singagore, December 2005.

[30] L. A. Smith King, M. Leeser, and H. Quinn, "Dynamo: a run-time partitioning system," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA '04)*, pp. 145–151, Las Vegas, Nev, USA, June 2004.

[31] W. Fu and K. Compton, "An execution environment for reconfigurable computing," in *Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '05)*, pp. 149–158, Napa, Calif, USA, April 2005.

[32] L. T. Smit, J. L. Hurink, and G. J. M. Smit, "Run-time mapping of applications to a heterogeneous SoC," in *Proceedings of the International Symposium on System-on-Chip (SoC '05)*, pp. 78–81, Tampere, Finland, November 2005.

[33] G. Aggarwal, R. Motwani, and A. Zhu, "The load rebalancing problem," in *Proceedings of the 15th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '03)*, pp. 258–265, San Diego, Calif, USA, June 2003.