Efficient Architectures for Streaming DSP Applications

Gerard J.M. Smit[1], Andre B.J. Kokkeler[1], Pascal T. Wolkotte[1], Marcel D. van de Burgwal[1], Paul M. Heysters[2]

***Abstract* — In this paper we focus on algorithms and reconfigurable tiled architectures for streaming DSP applications. The tile concept has a number of advantages: (1) depending on the requirements more or less tiles can be switched on/off, (2) the tile structure fits well to future IC process technologies, more tiles will be available in advanced process technologies, but the complexity per tile stays the same, (3) the tile concept is fault tolerant, faulty tiles can be discarded and (4) tiles can be configured in parallel. Because processing and memory is combined in the tiles, tasks can be executed efficiently (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, color 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.[3]**

***Index Terms* — Streaming applications, SoC design, NoC design, system design.**

I. Introduction

This paper addresses the design issues of a re-configurable System-on-Chip (SoC) platform and the supporting software tools for streaming DSP applications. Streaming DSP applications (sometimes also called Synchronous Data Flow programs) express computation as a signal flow graph with streams of data (the edges) flowing between computation kernels (the nodes). Most signal processing applications can be naturally expressed in this style [1].

In the Chameleon project [2] we defined a dynamically reconfigurable tiled heterogeneous SoC architecture for streaming DSP applications (see Section 2). In this architecture a tile can either be: a bit-level reconfigurable unit (e.g. FPGA), a word-level reconfigurable unit (e.g. Montium tiles [2][4]), or a general-purpose programmable unit (DSP or microprocessor). The tiles on the SoC are interconnected by a reconfigurable Network-on-Chip (NoC). The programmability of the architecture enables the system to be targeted at multiple application domains.

A. Holistic approach

In the Chameleon project we take a holistic approach, which means that all aspects of systems design need to be addressed simultaneously in a structured way. We believe that this is necessary for an efficient overall solution. An 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 tiles, and the design of the processing tiles should be coordinated with the tile specific compilers. Eventually, there should be a good fit between the application requirements and the SoC and NoC capabilities.

B. Predictable solutions

In order to manage the complexity of streaming DSP applications the solutions have to show deterministic behavior. The SoC and the NoC together with the real-time schedulers in the tile processors should provide latency guarantees. The reasons for predictability are 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 the latency these buffers would introduce is not acceptable in typical streaming DSP applications.

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 refresh rate of the screen). In other applications such as phased arrays individual paths of signals should be exactly predictable 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.

C. 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 computation for wireless protocol processing and multimedia applications, but the energy capacity of batteries is only increasing 10% per year. For high performance computing there is also a need for energy-efficient architectures to reduce costs for cooling and packaging.

In addition, also environmental concerns 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. the energy costs for 4 years of a typical low-end server is already more than 40 percent of the hardware costs [5]).

[1] University of Twente, dept. EEMCS
[2] Recore Systems, http://www.recoresystems.com
[3] This work has been partly supported by the Sixth European Framework Programme as part of the 4S project under project number IST 001908.5

D. CMOS technology

Most components are fabricated using CMOS technology today. The dominant component of energy consumption (85 to 90%) in 130 nm CMOS technology is dynamic. 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:

$$P_d = \alpha\, C_{eff} V^2 f \quad (1)$$

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 and activity by using locality of reference.

1) Minimize capacitance

Energy consumption in CMOS circuitry is proportional to capacitance. Therefore an option to reduce energy consumption is to minimize 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 and regularity. Connections to external components typically have much greater capacitance than connections to on-chip resources. Therefore, in order to save energy, use few external outputs, and have them switch as infrequently as possible. For example, accessing external memory consumes much energy. So, a way to reduce the effect of large capacitance is to reduce external accesses and optimize the system by using on-chip resources like caches and registers.

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 1mm 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!). A tiled architecture intrinsically encourages the usage of small and local in-tile memories. The length of a wire is bounded by the dimensions of a tile. Exploiting the locality of reference principle extensively will improve the energy-efficiency substantially. Due to the locality of reference principle communication within a tile dominates communication between tiles.

E. Streaming applications

In this paper the focus is on a tiled SoC for streaming DSP applications where we can assume that the data streams are semi-static and show a periodic behavior. This means that for a long period of time subsequent data items of a stream follow the same route through the SoC. These applications are often modeled as a synchronous data-flow graph.

Typical streaming DSP application examples are signal processing for wireless baseband processing (for HiperLAN/2, WiMax, DAB, DRM, DVB, UMTS [6]), multi-media processing (encoding/decoding): MPEG/TV, medical image processing, phased array antennas (for radar and radio astronomy) and sensor processing (for remote surveillance cameras and automotive).

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 [7][8] OFDM symbols or 1,024 x 1,024 x 24-bits frames for a video application. Also the data rate is application dependent e.g. 100 Msamples/sec after the A/D converter for a radar system, 250k OFDM symbols per second for HiperLAN/2, 50 frames/sec for video. The streams typically last for seconds and more. For example, a user listens to its radio or has a phone conversation for considerable time. However, the control system might change some settings of processes due to changing environmental conditions.

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

- 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 fixed rate, which causes periodic data transfers between the successive processing blocks. The rate of the blocks is application dependent (e.g. 4 µs for HiperLAN/2 and 26 ms for DRM). The resulting communication bandwidth is application dependent so a large variety in communication capacity is required.
- 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. Streaming applications show a predictable temporal and spatial behavior.
- For our application domains generally throughput guarantees (in data items per sec.) are required for the communication as well as for the processing.
- In general the amount of processing is fixed for each sample, however, in some applications the amount of processing per sample is data dependent (non-manifest).
- The life-time of a communication stream is semi-static, which means a stream is fixed for a relatively long time.

Our work is aiming at application domains where streaming DSP algorithms take 80 to 100% of the processing and communication resources in a system. This is the case for the examples mentioned above.

II. HETEROGENEOUS TILED ARCHITECTURES

Flexible and adaptive SoCs can be realized by integrating reconfigurable hardware parts of different granularities into heterogeneous reconfigurable SoCs. These systems typically

have a so-called 'tiled architecture', they contain domain specific processor tiles interconnected by an on-chip interconnection network. In our approach we assume that the set of tiles is heterogeneous (see Figure 1), e.g.: bit-level reconfigurable tiles (e.g. embedded FPGAs), word-level reconfigurable tiles (e.g. Montium tiles), and general-purpose programmable tiles (e.g. DSPs and microprocessors cores).

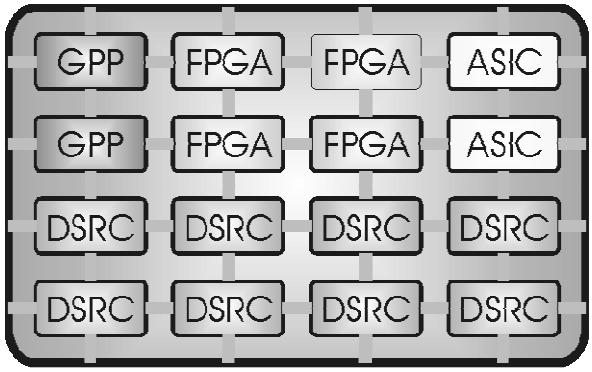

Figure 1: Heterogeneous tiled organization

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

A. *Tiled architectures*

In this section the benefits of tiled architectures are discussed. Recently a number of heterogeneous reconfigurable SoC architectures have been proposed for the streaming DSP application domain. Examples are: the Avispa from Silicon Hive (an incubator of Philips Research) [9]; the PACT-XPP [10]; the Maya [1] and Pleiades [11] chips from Berkeley and the Montium architecture from the University of Twente/Recore Systems [2]. For an overview we refer to [4].

A tiled architecture has a number of advantages:

- It is a future-proof architecture, as the processing tiles do not grow in complexity with technology. Instead, as technology scales the number of tiles on the chip grows.
- A tiled organization can contribute to the energy-efficiency of a SoC. The best energy savings can be obtained by simply switching off tiles that are not being used which also helps reducing static power consumption. Furthermore, the processing of local data in small autonomous tiles abides by the locality of reference principle. Moreover, a tile 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 tiles is discovered to be defect (either due to a manufacturing fault or discovered at operating-time by the build-in-diagnosis) this defective tile can be switched-off and isolated from the rest of the design.
- A tiled approach also eases verification of an integrated circuit design, since the design of identical tiles only has to be verified once. The design of a single tile is relatively simple and therefore a lot of effort can be put in (area/power) optimizations at the physical level of integrated circuit design.
- The computational power in a tiled architecture scales linearly with the number of tiles. The more tiles there are on a chip, the more computations can be done in parallel (providing that the network capacity scales with the number of tiles).
- Although tiles operate together in a complex system, an individual tile operates quite autonomously. In a tiled architecture every processor tile is configured in-dependently. In fact, a tile is a natural unit for partial reconfiguration. Unused tiles can be configured for a new task, while at the same time other tiles are performing other tasks. That is to say, a tiled architecture can be reconfigured dynamically.

B. *Heterogeneous 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 entity 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 of the processing entity, it can operate at minimum supply voltage and clock frequency and so provides energy-efficiency and flexibility at the right granularity only 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 run 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.

C. *Dynamic reconfiguration*

Reconfigurable systems offer the flexibility and adaptiveness needed for future streaming DSP applications. Reconfigurable computing can enhance the efficiency of these systems: doing more work with the same amount of energy resources. The flexibility of this platform is revealed as the ease of upgrading the system with a new or enhanced application or standard (long-term reconfiguration) as well as the ability of the system to adapt dynamically to changing environmental conditions (short-term reconfiguration).

Conventional reconfigurable processors are bit-level reconfigurable and are far from energy-efficient. However, 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: dependent 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 slight changes in the standards. By using reconfigurable architectures instead of ASICs costly re-designs can be avoided.
- The cost of designing 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.
- Dynamically reconfigurable architectures allow to experiment with new concepts such as software-defined radios, multi-standard terminals, 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 multimedia terminals.

D. Network on Chip

A tiled architecture has to be supported by a predictable NoC. A NoC that routes data items has a higher bandwidth than an on-chip bus, as it supports multiple concurrent communication streams. 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.

Beside the main-stream of the communication with guarantees we foresee a minor part (assumed to be less then 5%) of BE communication, for example control, interrupts and configuration data. This communication can have more relaxed requirements for the network and hence can use the best effort services.

If all the wireless standards are to be supported by the SoC, it immediately requires the support of several levels of granularity for the individual of data-streams. If we compare the required bandwidth between the processes of the different applications there is a large variety from several kbit/s (DRM) up to more than 0.5 Gbit/s (HiperLAN/2).

E. Streaming-mode and block-mode communication

We distinguish between two mechanisms for transferring data between the NoC and the tile processor. Some processes require all the input data to be in the local memories 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 tile processor. During the data transfers, the tile processor is halted to make sure the execution is not started until all data is 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 standard we can use block-based communication (OFDM) or have to use streaming-mode (UMTS) because the blocks get too large. Typically, during the execution of a streaming-mode process, 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 re-assembly of packets is reduced.

Whether block-mode or streaming-mode is 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 at the same time. This increases the ease of programming at process level, but gives some overhead at application level. For streaming applications the programmer has to carefully plan how and when the communication takes place.

III. RUN-TIME MAPPING OF STREAMS TO ARCHITECTURE

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

Ultimately, we would like to have a 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 a homogeneous architecture [13].

Today, it is common practice to map the applications to the architecture at design-time. In this section we consider how to 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 and required resources. Only at run-time the available resources are known to the mapping algorithm. Moreover, the available resources may vary over time due to applications running simultaneously, adaptation of algorithms to the environment or QoS parameters (e.g. video frame rate, screen size) set by the user or the applications.

- 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.

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 the central coordinating node (CCN); a central general purpose processor that has an overview of the entire SoC, which is needed for the earlier mentioned holistic approach. 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 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 has to be solved by light weighted methods in a limited time.

The problems we consider differ from multi-processor scheduling or load-balancing mechanisms [14] because: (1) we not only consider processing but also 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 multi-processors, (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 [15]) are not applicable and for existing heuristics (such as priority rules and local search) we carefully have to evaluate if they are adaptable for solving at least some of the sub problems of the overall problem.

IV. CASE STUDY: CHAMELEON SOC

As an example we will now present the Chameleon/ Montium architecture. We will show how we obtained a predictable and energy-efficient SoC/NoC. The key issue in the design of such future streaming architectures 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.

In the Chameleon SoC organization conventional architectures are complemented by domain specific coarse-grain reconfigurable architectures [4]. The Montium architecture is an example of a coarse-grained reconfigurable processing tile.

The Montium targets the 16-bit digital signal processing (DSP) algorithm domain. A single Montium Processing 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 different. For (energy) efficiency it is imperative to minimize the control overhead. This is for example accomplished by statically scheduling instructions as much as possible at compile time.

The lower part of Figure 2 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 definition of the off-tile interface depends on the interconnect technology that is used in the SoC.

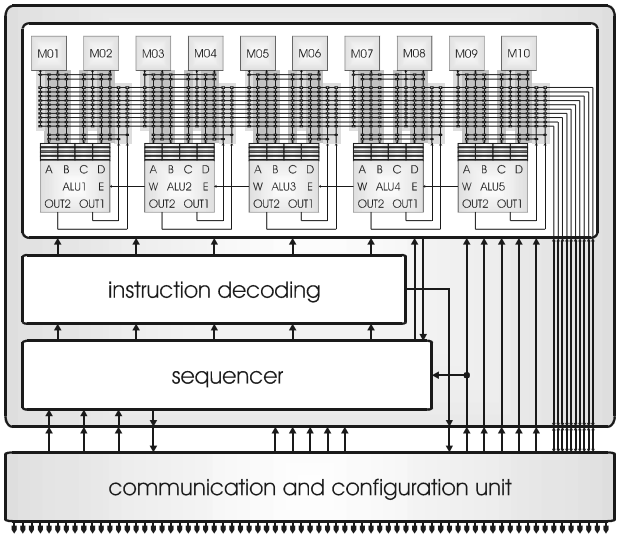

Figure 2: The Montium Tile Processor

The TP is the computing part that can be configured to implement a particular algorithm. Figure 2 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. A relatively simple sequencer controls the entire tile processor. The sequencer selects configurable tile instructions that are stored in the decoders (see Figure 2).

Each local SRAM is 16-bit wide and has a depth of 1024 positions, which adds up to a storage capacity of 2KB per local memory. A reconfigurable Address Generation Unit (AGU) accompanies each memory. 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. Neighboring ALUs can also communicate directly: the West-output of an ALU connects to the East-input of the ALU neighboring on the left.

A. *Hydra Network interface*

Each tile processor in the SoC requires a customized network interface that is used to provide a footprint that exactly fits to the NoC. As the processors operate independently, they need to be controlled separately. One of the tile processors, the CCN, controls the other tile processors by sending configuration messages to their network interfaces. Since the tile processors might not run at the same clock speed as the NoC, the network interface synchronizes the data transfers.

The Hydra network interface is an interface developed to serve the Montium [16]. 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 Hydra. In block-mode, the DMA interface is used to manage data transfers. The Hydra acts as a master for the tile processor, while the data transfers are directed by the CCN or other tiles. In streaming-mode the Montium is the master over the Hydra and takes care of the data flow from and to the NoC. Therefore, the algorithm's designer has to describe how the data transfers are performed.

B. *Implementation Results*

The ASIC synthesis of the Montium TP was performed using the Philips CMOS12 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 the Montium TP according to the synthesis tool is about 1.8 mm^2.
- 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.

1) *Average power consumption*

Figure 3 depicts an energy comparison between various architectures normalized to one FFT butterfly per clock cycle. It shows the power consumption per MHz of FFT butterflies. The average power consumption of a particular architecture is obtained by multiplying the normalized power consumption with the architecture's clock frequency.

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 usage of the coarse-grain multiplier blocks (Figure 3, design A) considerably improves the energy consumption of the FFT computation.

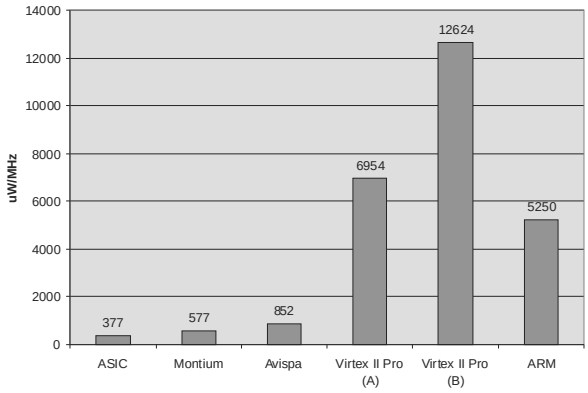

Figure 3: Average dynamic power consumption when FFT butterflies are computed at a rate of 1 MHz

2) *Locality of Reference*

As mentioned above, locality of reference is an important design parameter. One of the reasons for good energy-efficiency figures of the Montium is the use of locality of reference. To illustrate this, Table 1 gives the amount of memory references local to the tile processors compared to the amount of off-tile communications. These figures are 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 [17]). The results show that for these algorithms most memory references are local (within a tile).

TABLE 1: INTERNAL AND EXTERNAL MEMORY REFERENCES

Algorithm	#Internal mem. refs			#External mem. refs		
	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

3) *Partial dynamic reconfiguration*

One of the advantages of a tiled SoC organization is that each individual tile can be reconfigured, while the other tiles are operational. In the Montium the configuration memory is organized as a RAM memory. This means that to reconfigure a tile, not the entire configuration memory needs to be written, 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 memory can be accessed as a RAM, the system

allows for dynamic partial reconfiguration.

Table 2 gives some examples of reconfigurations. The **Size** column indicates the total number of bits that are reconfigured and the **#cycles** column gives an estimation of the number of configuration clock cycles that are required to perform the reconfiguration.

TABLE 2: RECONFIGURATION OF ALGORITHMS

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

C. *Predictable NoC*

For the Chameleon SoC we developed a predictable NoC [12]. This NoC supports both GT traffic as well as BE traffic. For the GT traffic guaranteed latencies are supported. Figure 4 presents the simulation results for a 6x6 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. In this experiment we increased the BE traffic load of the SoC.

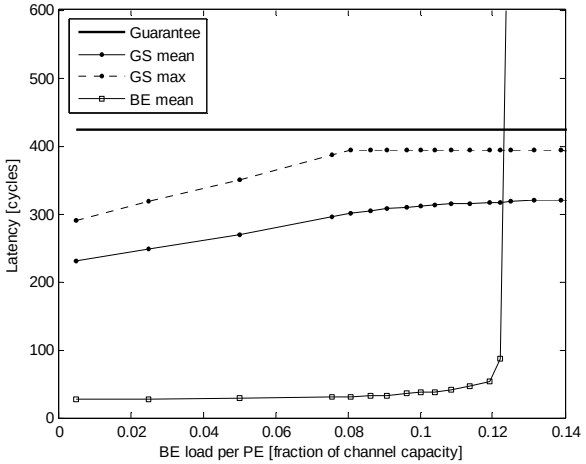

Figure 4: Message delay of the GT and BE traffic vs. BE load for 6-by-6 network

When the offered BE load is low, the latency of the GT packets is smaller than the guaranteed 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 for the BE traffic because the GT packets are larger (256 bytes) than the BE packets (10 bytes). With the increase of the BE load, the latency of the GT traffic increases too and at some point it saturates. Further increase of the BE load increases the GT mean latency but the GT maximum latency does not increase and never exceeds the guaranteed latency.

V. CONCLUSION

This paper addressed the design issues of a reconfigurable System-on-Chip (SoC) platform and the supporting software tools for streaming DSP applications. Streaming DSP applications express computation as a synchronous data flow graph with streams of data (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 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 reconfiguration. To keep the design manageable we take a holistic view and we apply deterministic principles, for example we have a NoC that supports both Guaranteed Throughput (GT) as well as Best-Effort Traffic (BE). In this paper we present a heterogeneous reconfigurable SoC architecture for streaming DSP applications. We show how locality of reference and partial reconfiguration works out in this architecture.

REFERENCES

[1] W. Dally et al.: “Stream Processors: Programmability with Efficiency” ACM Queue, March 2004, pp. 52-62.

[2] “Reconfigurable computing in hand-held multimedia computers”, http://chameleon.ctit.utwente.nl/

[3] P.M. Heysters, G.J.M. Smit: “Mapping of DSP Algorithms on the Montium Architecture”, Reconfigurable Architectures Workshop (RAW 2003), Nice, France, April 2003.

[4] P.M. Heysters: “Coarse-grained reconfigurable Processors – flexibility meets efficiency”, PhD thesis University of Twente, ISBN 90-365-2076-2, 2004.

[5] L.A. Barroso: “The Price of Performance”, ACM Queue vol. 3, no. 7, September 2005

[6] T. Ojanperä: “Wideband CDMA for Third Generation Mobile Communications.” The Artech House Universal Personal Communications Series. Artech House, 1998. ISBN: 0-89006-735-X.

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

[8] Software-Defined-Radio project, “A Bluetooth-HiperLAN/2 SDR receiver”, http://nt5.el.utwente.nl/sdr/.

[9] I. Held, B. VanderWiele: “Avispa-CH – embedded communications signal processor for multi-standard digital television”, GSPx TV to Mobile March 29-30, 2006.

[10] V. Baumgarte, F. May, et al.: “PACT XPP – A Self-Reconfigurable Data Processing Architecture”, Proceedings Engineering of Reconfigurable Systems and Algorithms, pp. 64-70, Las Vegas, USA, June 2001.

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

[12] N.K. Kavaldjiev, G.J.M. Smit, P.G. Jansen, P.T. Wolkotte: “A Virtual Channel Network-on-Chip for GT and BE traffic”, proceedings of IEEE Annual Symposium on VLSI, pp 211 – 216, March 2006.

[13] Y. Guo, G.J.M. Smit, P.M. Heysters, “Template Generation and Selection Algorithms for High Level Synthesis”, Proc. the 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications, Calgary, Canada, June 30 - July 2, 2003.

[14] G. Aggarwal, R. Motwani, A. Zhu: “The load rebalancing problem”, Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures, pages 258-265, 2003.

[15] M.L. Pinedo: “Planning and Scheduling in Manufacturing and Services”, Springer Series in Operations Research and Financial Engineering, ISBN: 0-387-22198-0, 2005.

[16] M.D. van de Burgwal, G.J.M. Smit, G.K. Rauwerda, P.M. Heysters: “Hydra: an Energy-Efficient and Reconfigurable Network Interface”, submitted for publication, 2006.

[17] P.M. Heysters, L.T. Smit, et al.: “Max-Log-MAP Mapping on an FPFA”, Proceedings of Engineering of Reconfigurable Systems and Algorithms, pages 90-96, Las Vegas, USA, June 2002.