On reconfigurable tiled multi-core programming
Processing cores evaluation

Kenneth C. Rovers, Marcel D. van de Burgwal, Jan Kuper, André B.J. Kokkeler and Gerard J.M. Smit
Computer Architecture for Embedded Systems group
CTIT, Department of EEMCS, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands
Email: K.C.Rovers@utwente.nl
http://caes.cs.utwente.nl/Research/?project=BeamForce

***Abstract*—For a generic flexible efficient array antenna receiver platform a hierarchical reconfigurable tiled architecture has been proposed. The architecture provides a flexible reconfigurable solution, but partitioning, mapping, modelling and programming such systems remains an issue. A semantic model has been presented to allow the development of the model for the specification, design and implementation. The semantic model is used for partitioning the application, evaluating the consequences and mapping to an architectures. Design space exploration allows us to adapt the partitioning and mapping to an architecture or visa-versa.**

With tiled reconfigurable cores as basis for the architecture, this paper explores the different options for processing cores and its suitability with respect to the design flow of the semantic model approach. Trade-offs with respect to granularity depending on flexibility and efficiency allow interesting design evaluations, especially for programability. This work therefore represent an important step forward in the design flow for designing and using multi-core tiled architectures.

***Index Terms*—Phased array, beamforming, hierarchical tiled architecture, semantic model, dataflow, functional programming**

I. Introduction

This paper will evaluate processing cores for use as an building block in a tiled multi-core architecture. The application domain for this architecture is high performance digital signal processing. The specific case is phased array beamforming processing. Beamforming is characterised by high rate streaming data and dependability requirements. The most important is performance, however, that doesn't mean energy efficiency or cost are not important. The goal is to define a generic phased array platform solution for applications areas like radar, radio astronomy or satellite receivers (DVB-S). A reconfigurable scalable system is envisioned as to achieve these goals. Therefore we will focus on (reconfigurable) digital processing. [1]

We will first summarise the beamforming application and the rationale for a reconfigurable tiled multi-core architecture. Section II will then discuss the design flow for using such architectures by partitioning and mapping. This will give us the environment in which the processing core will operate, which is used to evaluate different option in section III. Many of those have been used or are evaluated for use for the beamforming application. None of those however are suitable as a tile in the proposed architecture. Therefore a new core is proposed, the FlexCore, to be used to fulfil this task in section IV. As the signal processing is all sample based, we prefer course-grained reconfigurable hardware operating at word level. The goal is to achieve the efficiency of an FPGA/ASIC with the flexibility and ease of use of an DSP/GPP by applying a reconfigurable core matched to the tiled architecture and application domain.

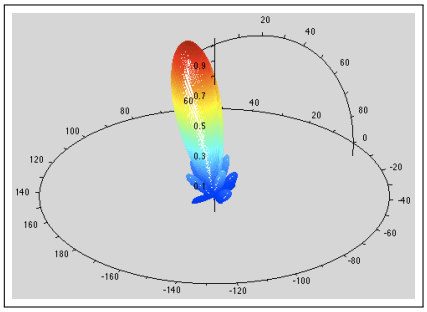

Fig. 1. Beamform or radiation pattern

A. Array antenna receiver platform

Phased array beamforming systems use multiple antennas in an array to make a receiver directional, i.e. form a electromagnetic beam into a certain direction (Fig. 1). The direction of the beam can be influenced by beamsteering. This allows one to point at the direction of interest in order to reduce interference or reject interferers, use less (wasted) energy or provide higher throughput for example.

A block diagram of a typical phased array receiver is shown in figure 2. Assume a single omni-directional wave source, emitting a spherical waveform in time and space $s(t,l) = A \cdot \cos(\omega t \pm kl)$, with A the amplitude, ω the frequency, k the wave number, t time and l the path length from the source. For a source in the far field perpendicular to the array, the wavefront is considered planar. If the plane of the array is not perpendicular, the wavefront arrives at different times at the antennas (see fig. 2). If the antennas are placed a distance d apart and the wavefront arrives at an angle ϑ incident to the array, the wavefront travels a distance $d \cdot \sin(\vartheta)$ further to the next antenna, resulting in a time delay $\Delta t = \frac{d \cdot \sin(\vartheta)}{c}$ between the signals (c is the propagation speed of radio waves). Depending on the frequency of the wave, this time delay results in a phase shift ($\Delta\psi = \omega \cdot \Delta t$) giving rise

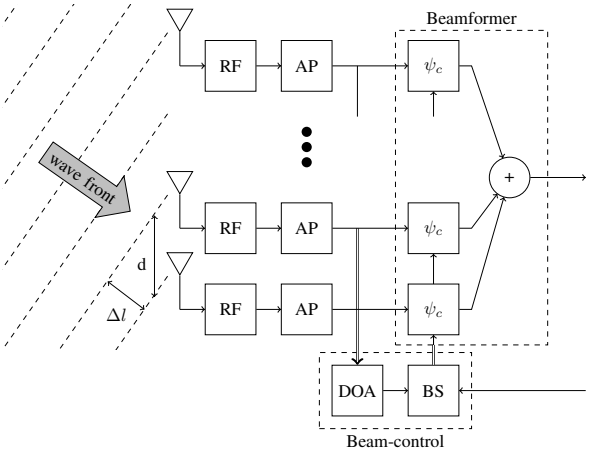

Fig. 2. Phased array receiver

to the term "phased array". By correcting the delay we can steer the direction of maximum sensitivity [2].

Thus, signals at the receiving antennas have different time delays and when combined the interference pattern results in a beam in the radiation pattern.After the RF (radio frequency) front end for each antenna, antenna processing (AP) may be applied for calibration or equalisation purposes (to correct for electrical or mechanical distortions of the front-end). The signals are then combined by the beamforming processing (beamformer). Beamsteering (BS) applies time delay or phase shift corrections to the antenna signals to add the antenna signals coherently in the direction of interest. Gain tapering/windowing can be used to change the shape of the beam. Note that multiple beams in different directions can be formed by duplicating the antennas signals and apply the beamforming for each beam with different correction parameters. To calculate these parameters, the beamsteerer needs to know in which angle (direction) to point the beam. This information is provided by the beam control process. It can be set by the user, based on an algorithm (for example for tracking a source) or based on an estimation of the angles of the strongest sources available. This latter estimation is provided by the direction of arrival (DOA) estimation process.

B. Reconfigurable tiled architectures

Phased array processing can be characterised as a streaming application with high data rates and processing requirements, but a regular processing structure. Because of costs, complexity, dependability and scalability reasons a design with mostly identical components is preferred, but because of functionality with different requirements and use it will be heterogeneous. We would like to limit the data rate as soon as possible through beamforming, because I/O is expensive. This implies that the processing is moved closer to the antennas. However, combined data cannot be separated later on, so we lose flexibility. Furthermore, the distributed processing must be synchronised. Because a scalable and dependable solution is needed, a tiled architecture is proposed with reconfigurable

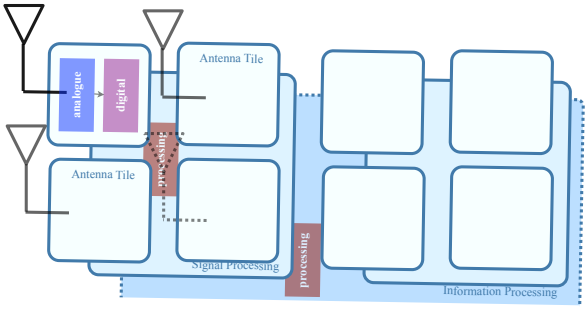

Fig. 3. Hierarchical tiled architecture

hardware to regain flexibility. Processing tiles are combined on multiple hierarchical levels. A multi-processor system-on-chip (MPSoC) can be extended to multiple chips on a board (MCoB) and multiple boards in a system (MBiS) giving a heterogeneous hierarchical tiled architecture (as in figure 3). We aim at a processing architecture which is flexible enough to support multiple methods of beamforming, as well as beamsteering and beam-control. [1]

The phased array application has a high data rate (i.e. around 100MHz 16-bit for radar and astronomy). Because of the number of antennas is in the range of 128 to 4096, the signal processing gives huge processing requirements (400 Gops 50 Tops [1]). Thus, a large number of tiles are needed. However the amount of processing per data sample is relatively low and the application is therefore data stream driven and characterised as a streaming application.

A reconfigurable hierarchical processing array can provide flexibility and has a number of advantages. We can use only part of the array or create multiple sub-arrays to save energy or increase the lifetime. Reconfigurability (also in I/O routing) supports graceful degradation if tiles break down. Reconfigurability inherently leads to having an adaptable system, that adapts to changing environments while maintaining the quality of service.

Reconfigurability is defined as fixing functionality (instructions) for a longer time (multiple clock cycles) in order to be more energy efficient. For the beamforming application reconfiguration can be performing at different scales with respect to the time required for reconfiguration and the impact to the system. Small scale reconfiguration can for example consist of changing the beamsteering parameters. For medium scale reconfiguration, the beamforming or tracking method can be changed. Large scale reconfiguration for example consists of changing to direction of arrival estimation, using subarrays or going to multi-function radar.

II. Design flow

For the signal processing application domain, the functionality to be performed is captured in mathematical equations. Furthermore, it is part of a higher level mixed signal system design. We would like to model this system at various degrees of sophistication in order to simulate and verify the design. The system specification used for analysis also often consists of

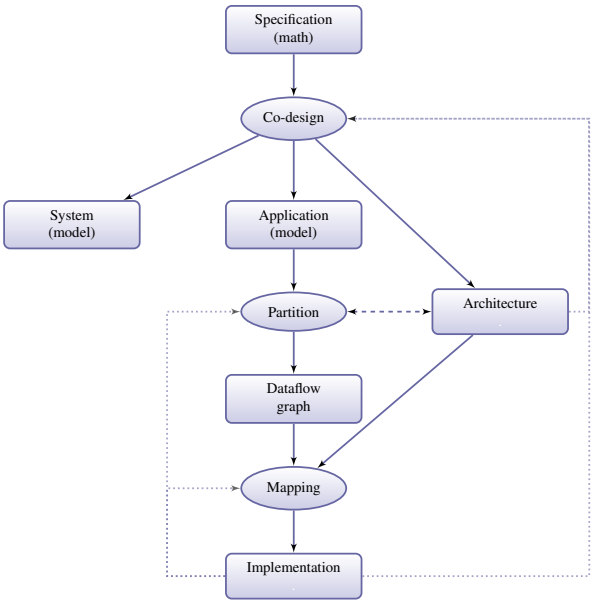

Fig. 4. Design flow for tiled architectures

math. Signal processing implementations, however, are often described in an imperative programming language as a sequence of commands changing state. This hinders the use of a model-based design approach to refine the system specification modelled in a mathematical tool such as MATLAB into an implementation.

In [3] and [4] the concept of a semantic model is introduced as a single model which can be used with different levels of detail. It is based on using a functional programming language for modelling. A functional language treats computation as the evaluation of mathematical functions and avoids state and mutable data. It is therefore much more suited to the signal processing application domain. Mathematical rules can be used for formal verification and model and program transformations aiding design space exploration. It can also be used to define different models of computation (MoCs), such as continuous and discrete time in our mixed signal system. Because it is a programming language, it can be run for simulation and evaluation at each stage of the design up to providing an implementation for the architecture.

To go from system specification to implementation with a tiled architecture as basis, the design flow of figure 4 is used. During the co-design stage, the system specification is split into the different MoCs. For example, analogue/digital co-design is used to define where to go from continuous time to discrete time, and hardware/software co-design is used to define the application. The application is then partitioned to define the communication between the different parts of the application and to match with the architecture. After partitioning the application is modelled as a dataflow graph to capture the communication. The processes of the dataflow graph are assigned to hardware blocks by the mapping.

We will elaborate on this design flow with the beamforming application as an example.

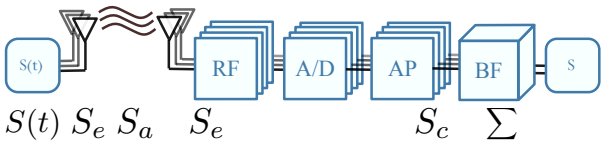

Fig. 5. Simplified system block diagram

A. Semantic model and co-design

A simplified beamforming system block diagram is shown in figure 5. This block diagram corresponds to the system specification detailed next.

Based on the radar equation [5], the resulting signal after beam-forming can be represented by the source signal $S(t)$, an element factor depending on the sensitivity or gain of each antenna element S_e, an array factor depending on the element positions S_a, a correction (steering) factor S_c and a combining sum:

$$\begin{aligned} S &= \sum S(t) \cdot S_\mathrm{e}(\theta,\varphi) \cdot S_\mathrm{a}(l) \cdot S_\mathrm{c}(\theta_0,\varphi_0) \\ &= \sum a \cdot e^{j(\omega t \pm \psi_\mathrm{e}(\theta,\varphi) \pm kl \pm \psi_\mathrm{c}(\theta_0,\varphi_0))} \end{aligned} \tag{1}$$

$$\psi_\mathrm{c}(\theta,\varphi) = k \cdot (-\Delta l(\theta_0,\varphi_0)) = \omega \cdot (-\Delta t(\theta_0,\varphi_0)) \tag{2}$$

$$\Delta l = \vec{r} \cdot \vec{R} = dx \cdot u + dy \cdot v + dz \cdot w \tag{3}$$

with ψ the phase, $\vec{r}$ the element position, $\vec{R}$ the plane wave direction, u, v, w the direction cosines and $-\Delta t(\theta_0,\varphi_0)$ the time delay correction [2], [5], [6].

From these block diagram and equations we can identify the following blocks: a single source source $S(t)$, a transmitter S_e, channel S_a, receiver S_e, RF front-end and AP block S_c per antenna and a beamformer $\sum$ combining the signals. These parts correspond to functions in the semantic model shown in listing 1. Furthermore, the `chain`, `frontend` and `systm` model compositions (explained below). We defined types to represent a signal (`Sig`) with parameters for the frequency, amplitude, gain, and time, a direction of arrival (`DOA`) with range, azimuth, and elevation parameters, an element position (`Pos`) with cartesian coordinates, and a beam-steer direction (`BSt`) with two angles. The `map` function applies a function to each argument of a list. By mapping the source signal over a list of time instants, we create a list of the signal over time, which we can use as input to the system to perform a simulation. The listing can be run, thereby performing a simulation with results as expected. A single source goes to a separate `transmitter`, `channel`, and `receiver chain` for each element. All chains for the elements together form the `frontend`, which uses the `map` function to create such a chain for each element. The `fmap` function is used to provide the same source signal (`s::Sig`) to each chain by mapping the list of front end chains over the source function. The output of the frontend is provided as input to the `beamformer` block with the pipe operator (`>>`, see listing 2), which simply performs a function composition. The `frontend` and the `beamformer` form the `systm`, which expects a signal as

```
type Sig = S (Num -> Num -> Num) Num Num
type DOA = D (Num, Num, Num)
type Pos = P (Num, Num, Num)
type BSt = B (Num, Num)

source t = S (sine f a) g t
simulation = map systm (map source ts)

systm :: Sig -> [Num]
systm s = ( frontend d ps >> beamformer bs ps ) s
frontend :: DOA -> [Pos] -> Sig -> [Num]
frontend d ps s = fmap (map (chain d) ps) s

chain :: DOA -> Pos -> Sig -> Num
chain d p s = (transmitter d p >> channel d p >> receiver d
    p >> adc) s
```

Listing 1. Phased array semantic model

```
(f >> g) x = (g . f) x = g (f (x))
```

Listing 2. Pipe operator

input and gives a beamformed result for each beamsteering vector provided.

B. Partitioning

The next part is partitioning the application into parts as shown in figure 6. In this step we add communication details to part of the system, the antenna processing and beamforming parts. This is done by going to a dataflow model. The dataflow model provides a MoC for stream processing with explicit communication. Processes in the dataflow model encompass computation and channels between processes encompass communciation. This model provides us with automatic synchronisation by using back-pressure and automatic parallelism by using referentially transparent data. Different analysis techniques provide quality-of-service (QoS) bounds on the latency, throughput and buffer sizes [7].

For the phased array beamforming application we partition the application into a process for each antenna processing part and a number of processes for the beamforming part. After some design space exploration, we settle at beamforming 4 inputs into a single output per process. For 64 antennas this results in a hierarchical beamformer with 21 processes. The 64 antenna processing processes and the 21 beamforming processes thus define the partitioning. The dataflow model and the partitioning are still part of the semantic model and can be simulated.

C. Mapping

After partitioning each process is assigned to a tile and each channel to a network connection by the mapping as shown in figure 7. The dataflow model is annotated with performance requirements and the architecture with constraints and costs. The mapping tool uses heuristic algorithm to determine the mapping. More information can be found in [8]. The result is that functionality is assigned to tiles and network interfaces and links.

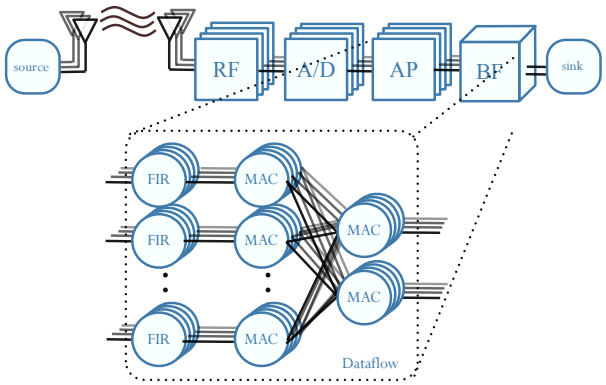

Fig. 6. Partioning

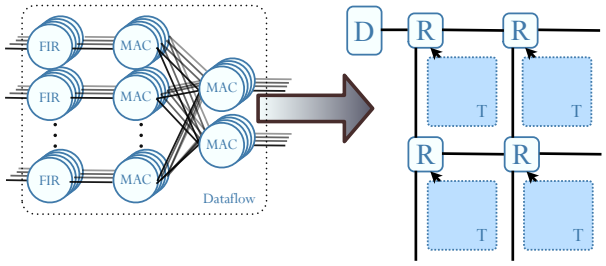

Fig. 7. Mapping

III. Processing Cores

In this section different options for the processing are evaluated. For simplicity we will assume a single processing core for each tile in the architecture. After the requirements, the available solutions are discussed, which will give us enough information to define the "ideal" core for our case.

A. Requirements

After going through the design flow of section II, the application is partitioned and mapped to the tiled multi-core architecture. However, during the mapping process we claimed the mapped is based on constraints and cost of the architecture. These constraints and cost are for a large part defined by the choice of processing cores and network part in the architecture. The mapping process and also the partitioning are therefore very much a back and forth process, indicated by the dotted arrows in figure 4. For the relatively simple case provided this process could be performed without making detailed architectural choices besides choosing a tiled design.

So the starting point is a dataflow model mapped to a tiled architecture. Each tile in the architecture is a process in the dataflow model. The beamforming operation mostly consists of multiply-accumulates (MAC) and much streaming IO with a fixed route. The beamsteering operation are mostly in the area of complex number algebra. Direction-of-arrival (DOA) or tracking algorithms consist of matrix operations. Calibration consists of logical and comparison operation, with equalisation and filtering again being MACs. All these have mainly fixed

routing requirements. However, control, supervision and test consists of control instructions and requires flexible IO routing.

The major part of the work of the front-end is in the AP and BF block. Most of the needed performance is determined by those parts. Thus, in this paper we will focus on the processing and assume the network is configured and transparent for the application.

B. Available solutions

A number of options are available in the trade-off between efficiency and flexibility. Advantages and disadvantages of each are discussed and summarised.

1) ASIC: At one side of the scale is an ASIC, which is a natural choice if performance is of uttermost importance. A complete hardware solution for beamforming is fast, efficient and has little overhead. However, functionality is fixed at design time. So needed flexibility must be design in. Furthermore, ASICs are relatively difficult to design and very costly, both in development as in production of limited amount (less than millions).

In the past these costs were lower and they were used for their performance. However, for dependability and control, flexibility in the design is required and the costs are too high.

2) Fine grained reconfigurable hardware (FPGA): A field-programmable-gate-array (FPGA) can be used as a stand-in for an actual ASIC. Digital hardware is "emulated" by allowing the designer to connect gates. Therefore, the same design tools as for ASICs can be used and the overhead is limited. But more flexibility is gained because the FPGA can be reconfigured in the field to change its functionality. However, this reconfiguration can not be performed easily at run-time. Unfortunately, as the same design tools are used they remain difficult to design or program. And while they are much cheaper than ASICs they are still too expensive for consumer products. A more generic building block achieving the numbers needed for an ASIC is preferred there. An FPGA also has quite some overhead (in area and power) because the interconnect between the gates is quite extensive.

FPGAs are the main processing hardware used for signal processing in radar and radio astronomy, but it is too expensive for a satellite receiver. Furthermore, the shorter product cycles means the development effort needed for FPGAs becomes limiting.

3) Coarse-grained reconfigurable hardware (Montium): An example of a course-grained reconfigurable processor is the Montium [9]. The Montium has 5 ALUs and 10 memories with address generation units (AGUs) connected by an interconnect. The functionality of the ALUs, AGUs and interconnect is configured and fixed while processing data. The Montium can be run-time reconfigured and is designed for energy efficiency by exploiting locality of reference and avoiding unnecessary bit-flips. The design is balanced with approximately equal parts for processing, memory and control and communication. The application domain is digital signal processing applications with streaming data and it is well suited for running FIR filters or FFTs.

The Montium therefore seems well suited to perform the beamforming processing. The beamforming operation was implemented on a Montium, but because of the large amount of choices for running an application on a Montium, the implementation on the Montium is a labour intensive manual process. This corresponds with the effort required to implement other applications [10]. Furthermore, the 3-stage ALU give overhead as only the MAC stage is needed. It also is not flexible enough for the control parts.

To mitigate some of the disadvantages we have cut-out the unnecessary stages in the ALUs. Unfortunately this does achieve a speed-up as the balance in the design breaks down. Therefore more extensive design changes would be needed.

4) DSP/GPU: A digital signal processor (DSP) or a graphics processing unit (GPU) can be seen as general purpose processors (GPP) optimised for an application domain, signal processing and graphics respectively. Both could be suitable for the beamforming application. An advantage is they are much more flexible with respect to control operation than the previous options, especially the DSP. An DSP has support for complex numbers, saturated computations and MACs. An GPU support operations on large amount of data at the same time. All of which are useful for the beamforming application. Another large advantage is the support for higher level programming languages and tools.

Of course, this added flexibility comes at a cost of overhead and performance. Because they are easy to program, DSPs are used but later in the chain for lower data rates, as the performance is not enough. GPUs have been evaluated but do not provide enough flexibility to be effectively used.

5) GPP/CPU: The general purpose processor (GPP) or central processing unit (CPU) provides a lot of flexibility at a low cost. It can support any application of beamforming. It also has extensive programming languages and tool support. The performance is not sufficient for beamforming. Also the energy efficiency is worse than the other options because of years of increasing the clock frequency and performance at any cost. However, with the advances over the years, the GPP has been moving forward in the phased array beamforming processing domain. It is used for simple system and for general control. A disadvantage that the GPP shares with the DSP is that the design is based on a sequential Von Neumann machine. While beamforming is embarrassingly parallel and data oriented, the Von Neumann machine is control oriented [11], [12].

6) Comparison: The different options discussed are summarised in table I. This table shows that an ASIC is very strong in performance and efficiency, while the GPP is very strong in cost, programability and flexibility. A good trade-off could be found in the Montium if it would be more programmable and flexible or the DSP if it would have a higher performance and efficiency.

C. Approach

A solution could be to more extensively redesign the Montium. A survey around a group of practical users of the

TABLE I
PROCESSOR COMPARISON

	ASIC	FPGA	Montium	DSP	GPP
performance	++	+	+	-	- -
efficiency	++	-	+	-	- -
overhead	++	-	+	-	- -
cost	- -	-	+	+	++
programmability	- -	- -	-	+	++
flexibility	- -	+	-	+	++

Montium identified the following pros and cons:

- The reconfiguration abilities together with the address generation units effectively exploit locality of reference to achieve an competitive design from an performance/energy perspective. This is especially true for applications such as a FIR filter or FFT.
- The five parallel ALUs within one Montium provide some performance, but it is limited by either the connections between them or the decoders of the sequencer or both.
- The first ALU stage is not used for some applications.
- There is a communication bottleneck to and from the Montium and thus between tiles. Data can not be delivered to the Montium fast enough for streaming applications.

So a redesign is feasible with a simpler ALU optimised for MACs, a simpler non-limiting decoder and a simpler crossbar with better communication interface between tiles. However, the problem remains that it is very complex to effectively configure the Montium. For example, to continuously keep the processor processing, we must fill a local memory with new data and read out the processed data at the same time as processing and we must switch when ready. It is also a challenge to find a mapping to use all the processing parts available, because of the many choices and interdependencies.

Therefore, we propose to take the strengths of the Montium and design a new processor avoiding the weaknesses and better matched to the design flow detailed above.

IV. FLEXCORE

A design flow for the digital signal processing application domain is presented with a focus beamforming. Starting point is a mathematical specification, which we use to define a system model, an application and an architecture. The architecture is assumed to be a system-on-chip (SoC) with tiles connected by a network-on-chip (NoC), because of the requirements of the application. However, a suitable processing core to fit this model has not been found. In this section we will detail the characteristics and requirements for a processor intended to solve the problems found. This processor is called the FlexCore.

A. Design

The main goal is to propose a design that fits the presented design flow. In the design flow processes from the dataflow model are mapped to tiles on the architecture. The processing tile must therefore accept data from channels and output data to channels, with the channels themselves providing the connections between the tiles. This fits well with the streaming application domain. A natural choice is then to use a dataflow processor. Dataflow processors however, as well as functional languages, are deemed to be inefficient. We will discus this in more detail below. First we will discuss the requirements.

1) Requirements: The requirements are defined by three factors involved: the requirements from the application and application domain, the requirements following from the design flow and the requirements following from the evaluation of processing cores.

From the application, where we focus on implementing beamforming and beam-control, we found that:

- The performance is dominated by the beamforming processing, which mainly consists of (complex number) multiply-accumulate operations.
- Flexibility is required for control and testing. Much of the flexibility is in the communication and therefore the NoC.
- Functionality is changed, but there is not much data dependence. Therefore reconfiguration is sufficient.
- The architecture and tiles must be scalable and modular to be used as a building block in multiple designs.
- The processing cores must be general enough to be used by multiple applications to save cost.
- Energy efficiency is important because of the different requirements between applications and because of the number required for phased array beamforming. This means overhead must be limited when possible.

From the design flow, with the main focus on the usability of the architecture, we found that:

- The processing cores executes a process.
- The communication to and from the core is based on streaming data channels.
- The processing core preferably directly runs sub-parts of the application following from the partitioning and mapping.
- It is therefore directly usable in the semantic and dataflow models.
- We would like to stay close to the math also for the processing core.
- State and memory is in principle separated, shared memory can be implemented on top of it.

From the processing core evaluation, where we focus on the strengths, we found that:

- We would like a coarse-grained reconfigurable processor with the efficiency of an FPGA/ASIC and the flexibility and ease of use of an DSP/GPP.
- The core is optimised for the signal processing application domain, but general enough not to limit any application. Ideally it must be able to run any application and perform well for signal processing applications.
- The number and mix of execution units is preferably customisable. This enables one to tweak the core for the intended application domain and performance at design

time, while relieving the burden of designing and verifying a complete processor at once.
- Energy efficiency is achieved by exploiting locality of reference with reconfiguration. For these kind of applications with streaming processing, locality is in the instructions.

2) Dataflow processors: Dataflow processors fit well with the dataflow model resulting from the design flow, which in turn fit well with the application domain. Signal processing applications have a mathematical specification as base. Mathematical functions operate on input and generate output, just as functions in a functional programming language and processes in the dataflow model.

This approach has a number of advantages. Functions have no side-effects other than the output. This means that state is explicit and therefore probably intended. Thus only dependencies that are intended and in the mathematics are created. Therefore, parallelism is not unnecessarily restricted. Furthermore as there is no central state to continuously update, shared data is also explicit, distribution of memory is much easier. However, dataflow processors also have there share of disadvantages. Because matching on tags is used in order to connect the results of computations and substitute the result, matching is performed for each data value. This limits performance and efficiency an expensive associative memory is needed. Associative memory is about twice the area and energy of normal memory. We intend to use configurations to fix and thus disable the matching for a number of clock cycles in order to drastically reduce this cost. Dataflow processors also have difficulties handling larger data structures because matching is performed on words and not on larger data structures. We intend to use a local memory with a address generation unit to linearise memory accesses and allow matching on larger data structures. Furthermore, dataflow processors do not match very well with the imperative programming languages used by the majority of software engineers. However, the functional language we are using matches very well and vice versa the functional language is also expected to perform better on a dataflow processor than on a Von Neumann architecture.

Dataflow processors have failed to succeed in the past because the incredible improvements of RISC processors. RISC processors could ride Moore's law to get faster with each improvement in process technology, while dataflow processors could less easily. However, now limits have been reached in the improvement of single core processors and multi-core programming because necessary for performance improvements and this is where dataflow processors shine. The final flaw of previous dataflow processors has been that they were positioned as general purpose processors, while they are by nature much more suitable for data oriented applications such as ours, than for control oriented applications.

B. Architecture

The FlexCore architecture consists of a local memory, a address generation unit, a dispatcher, execution units with corresponding queues and a router for communication. The FlexCore dispatcher uses tags and matching to perform substitutions of computational results, as in done in mathematics. This matching is the communication in the application. In order to limit the (energy) cost of matching, configurations are used to fix the matching operation for a number of data values set by a loop counter. Address generation units are used to linearise memory access of common data structures. This limits the amount of data dependent control instructions.

The execution units are separate autonome components, processing if instruction are available in the queue and there is space for the output. The dispatcher fills the queues as instructions are ready to be processed. The type and number of execution units is a design parameter which can depend on the required performance and application. Furthermore, the number of cores in a tile is scalable as well as the number of tiles in a chip. Thus the multi-core tiled architecture is scalable at three levels, which we can use to balance the design. Performance is then achieved by exploiting parallelism and distributed processing and the scalability of the design.

The FlexCore is easy to use and program, because it corresponds well to the functional language used in the semantic model based design flow. It can directly run the subpart of the application, while the matching takes care of connection everything together. Because reconfiguration fixes the matching, most data transfers are for getting the data to the processing units and for explicit communication in the application. Therefore the execution units are expected to be well utilised, resulting in an efficient design.

V. Conclusion

In this paper the design flow for a signal processing application on a tiled reconfigurable multi-core architecture is presented, based on a semantic model. This semantic model is is a single model for system specification, design, evaluation and implementation. The design flow is demonstrated by performing co-design, partitioning and mapping for a phased array beamforming application. The result is a dataflow model mapped to a tiled architecture.

A dataflow model is used as a model to effectively and efficiently use tiled multi-core architecture, because computation is partitioned, communication is made explicit and synchronisation is provided by back-pressure. Analysis techniques can then provided quality-of-service measures.

Different processing cores are evaluated for use as a tile in the tiled architecture for the beamforming application. Coarse-grained reconfigurable processors are found to be most suitable for their flexibility. However, their ease of use is found to be lacking which is also a major argument against using FPGA. DSPs on the other hand lack the required performance.

Thus a new core is proposed called the FlexCore, an easy to use high performance, energy efficient, reconfigurable processor. It is based on dataflow principles and using the strengths of the evaluated cores while avoiding their weaknesses. Reconfigurability is used to achieve efficiency, dataflow principles are used for programmability and ease of use. The FlexCore

is scalable at three levels allowing one to balance the design and achieve high performance.

ACKNOWLEDGMENTS

This research is partly funded by Thales Netherlands and STW projects CMOS Beamforming (07620) and NEST (10346).

REFERENCES

[1] K. C. Rovers, M. D. van de Burgwal, A. B. J. Kokkeler, and G. J. M. Smit, "Rationale for and design of a generic tiled hierarchical phased array beamforming architecture," in *Proceedings of the 18th Annual Workshop on Circuits Systems and Signal Processing (ProRISC), Veldhoven, the Netherlands*. Utrecht: Technology Foundation, November 2007, pp. 160–168.

[2] H. J. Visser, *Array and phased array antenna basics*. Chichester: Wiley, 2005.

[3] K. C. Rovers, J. Kuper, and G. J. M. Smit, "Semantic programming model-based design," in *Proceedings of the 19th Annual Workshop on Circuits Systems and Signal Processing (ProRISC), Veldhoven, the Netherlands*, November 2008, pp. 83–88.

[4] K. C. Rovers, M. D. van de Burgwal, J. Kuper, and G. J. M. Smit, "Towards effective modeling and programming multi-core tiled reconfigurable architectures," in *Proceedings of the 2009 International Conference on Engineering of Reconfigurable Systems & Algorithms, Las Vegas, Nevada, USA*. USA: CSREA Press, July 2009, pp. 167–174.

[5] M. I. Skolnik, *Introduction to Radar Systems*, 3rd ed. New York, NY, USA: McGraw-Hill, 2001.

[6] H. L. van Trees, *Optimum array processing*. New York: Wiley-Interscience, 2002, vol. Detection, estimation and modulation theory.

[7] M. H. Wiggers, "Aperiodic multiprocessor scheduling for real-time stream processing applications," Ph.D. dissertation, University of Twente, Enschede, The Netherlands, June 2009.

[8] T. ter Braak, "Run-time spatial mapping in heterogeneous mpsocs," Master's thesis, University of Twente, August 2009.

[9] P. M. Heysters, "Coarse-grained reconfigurable processors – flexibility meets efficiency," Ph.D. dissertation, University of Twente, Enschede, The Netherlands, Sep 2004.

[10] G. K. Rauwerda, "Multi-standard adaptive wireless communication receivers – adaptive applications mapped on heterogeneous dynamically reconfigurable hardware," Ph.D. dissertation, University of Twente, Enschede, The Netherlands, Jan 2008.

[11] J. Backus, "Can programming be liberated from the von neumann style?: a functional style and its algebra of programs," *Commun. ACM*, vol. 21, no. 8, pp. 613–641, 1978.

[12] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 4th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.