FPGA implementation of a Prototype Hierarchical Control Network for Large-Scale Signal Processing Applications

Jérôme Lemaitre[1] and Ed Deprettere[2]

[1] ASTRON, Oude Hoogeveensedijk 4, 7991 PD Dwingeloo, The Netherlands
[2] LIACS, Leiden university, NielsBohrweg 1, 2333 CA Leiden, The Netherlands

Abstract. The performance of a high throughput and large-scale signal processing system must not be compromised by the control and monitoring flow that is inherently part of the system. In particular, the interfacing of data flow and control flow components should be such that control does not obstruct the signal flow that is of higher priority. We assume that the signal processing is modeled as a distributed hierarchy of data flow networks, and that the control and monitoring is modeled as a distributed hierarchy of communicating Finite State Machines. The interfaces between leaf-nodes of the control and monitoring network, and the signal processing nodes in the dataflow networks are specified in such a way that the semantics of both network types are preserved. In this paper, we present the prototyping of a control network and its interfacing with a data flow network in a FPGA-based platform, and we analyze the performance of the interfacing in a case study. The HDL code that is involved in the interfaces is generated in a semi-automated way.

1 Introduction

Large-scale signal processing systems such as phased array radio telescopes [15] typically comprise of a hierarchically distributed data flow network (DFN), a hierarchically distributed control network (CN), and an interfacing between these two networks. Depending on the nature of the astronomical source that is observed, the system must be able to operate in modes that range from spectroscopy, pulsar observation or searches for transients. Moreover, disturbances in the high throughput streaming data paths, which are mainly due to radio frequency interferences and changes in the ionosphere, must be monitored and mitigated [14] by re-configuring the dataflow processing at run-time. Thus, to each operational mode corresponds a different set of high-level dataflow processing parameters (e.g. frequency resolution and integration time) and control parameters (e.g schedule to update the number of beams and blanked channels).

We assume that signal processing tasks such as filtering, FFT, beamforming and correlation in the DFN are modeled as nodes of Kahn Process Networks (KPN [12]). The control data for re-configuring and monitoring the processes in the KPNs and/or the components onto which they are mapped is sent over the

CN that has a tree or lattice-like structure, until they reach CN leaf-nodes which interact with KPN nodes through specific interfaces. As shown schematically in Figure 1, the interaction between these nodes is synchronized by means of periodic pulse trains that are distributed to control nodes in a synchronization network. The periods of the pulse trains are so chosen that a command that is sent over the CN to a dataflow process reaches this process during the period that precedes its execution. The process will then execute this command concurrently and complementarily to the DFN data processing.

This paper focuses on the way the CN is modeled and reports on a prototype FPGA implementation. In [7], the interfaces between CN leaf-nodes and KPN processing nodes have been so modeled that the two networks that are designed separately can work together without compromising their individual semantics. This paper also demonstrates that these interfaces can be implemented in a semi-automated way, taking IP re-use and scalability constraints into account, and avoiding error prone and time consuming handcrafting of FPGA implementations [10].

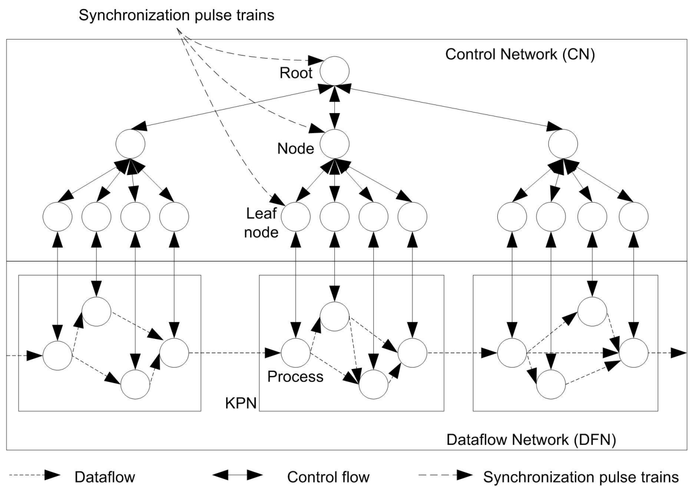


Fig. 1. Interface between a control network (CN) and a data flow network (DFN). The root, nodes and leaf-nodes receive periodic pulse trains in a synchronization network.

In the remaining of this section we give our problem statement, solution approach and related work. The rest of the paper is organized as follows. In Section 2, we explain how to map nodes of the CN onto soft-cores and how to interface CN leaf-nodes with KPN processing nodes that wrap hardware IPs. In Section 3, we present a case study for the control of processes in a KPN in the DFN from leaf-nodes in the CN, and discuss the results concerning the Hardware Description Language (HDL) semi-automation, the IP re-use, and the scaling in the design. Finally, conclusions and future work are drawn in Section 4.

1.1 Problem statement

The high throughput large-scale systems we are concerned with comprise of two networks that transport and process signal data and control data, respectively. The signals propagate and are processed in a hierarchy of distributed KPNs that together make up the Dataflow Network (DFN). The control data are passed and processed in a tree or lattice like structure of communicating Finite State Machines that make up the Control Network (CN). Signal processing tasks in the KPNs as well as the system components onto which they are mapped are re-configured and their behavior is monitored from the leaf-nodes in the CN for the system to operate in a particular mode. Because the two networks have different semantics as shown in Table 1, their interfaces have to be defined and modeled judiciously to avoid semantic corruption. Their definition and model can be found in [7]. The problem that is addressed in this paper is whether these interfaces can be implemented in a semi-automated way such that, indeed

– The semantics of the two models of computation remain respected, and
– The inclusion of control interfaces does not obstruct the performance of the DFN in terms of throughput and resource usage.

	Dataflow network (DFN)	Control network (CN)
Behavior	Deterministic	Sporadic
I/O data type	Streams	Messages
Scheduling	Local	Global
Synchronization	Blocking write and read	Periodic
Timing	Self-timed	Synchronous

Table 1. Characteristics of the data flow network and control network to interface

1.2 Solution approach

In the CN, all nodes are connected to other nodes above through a single port, and all nodes, except the leaf-nodes are connected to other nodes below through a single output port. They receive and send control packets from and to these ports, respectively. The leaf-nodes have three ports to below: a configuration-data output port, a command output port, and a monitoring-data input port as shown in Figure 2. Control packets that are received from above are unpacked (possibly after some processing) to separate commands from configuration data before being send downwards. By the same token, incoming monitoring data is packed in control packets that are sent upwards, possibly after some processing of the packets. The DFN is mapped onto networks of platforms that implement the KPNs in the DFN, and each platform has a single entry point for control data. CN leaf-nodes are internal to platforms and are interfaced to platform components onto which processes of a KPN in the DFN are mapped. As a result, all platform components must have a configuration-data input port, a command input port, and a monitoring data output port. A CN leaf-node that is associated with a dataflow component therefore unpacks component-specific configuration data, releases component-specific commands, and packs component-specific incoming monitoring data.

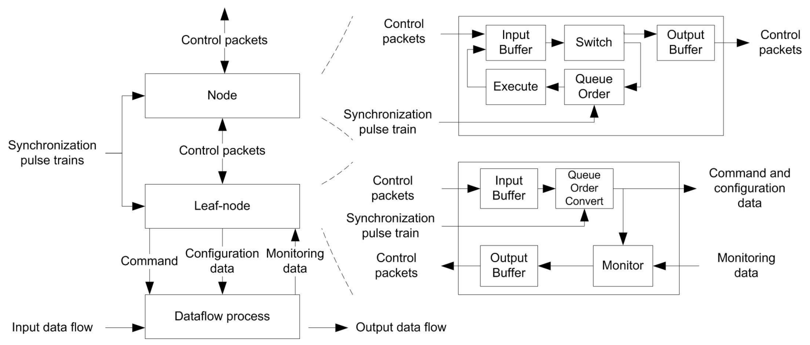


Fig. 2. Approach to interface CN nodes with CN leaf-nodes, and CN leaf-nodes with DFN processes.

In our approach to prototype the hierarchical CN we start from Figure 1. `Root` is mapped on a PC, and the rest of the figure is mapped on an FPGA-based platform as if it were a DFN platform onto which a KPN was mapped. Thus `Node` is the platform's single entry point. `Node` and `Leaf-node` are mapped onto soft-cores (e.g., the `microblaze` from Xilinx, or the `nios II` from Altera) of the FPGA-based platform [16] [17]. `Leaf-node` is controlling a specific DFN platform component. The control of a dataflow component is kept separated from the dataflow and from the synchronization mechanism by means of three concurrent FSMs. A first FSM synchronizes the execution of commands issued from a leaf-node with the (periodic) execution of the signal processing function. A second FSM controls the dataflow `Read` and `Write` ports in a SBF (Stream Based Functions [8]) dataflow model of computation. A third FSM controls the IP `Execute` function repertoire in the SBF dataflow model of computation. These FSMs are generated automatically in HDL from a high-level specification.

1.3 Related work

The PSDF model [1] separates the dataflow specification from the control specification, with the objectives of staying within one model of computation and modeling parameterized dataflow in such a way that it remains possible to derive quasi-static schedules [2]. We are dealing with with a large DFN and a large CN instead of a single process network. The DFN is a network of distributed KPNs and the CN is a network of communicating FSM. The interfacing of the two is not done as in [1] because the behavior of the CN is sporadic. Thus we cannot schedule and we cannot afford blockings read at the interface.

The FunState model [3] unifies many models of computation and allows to verify scheduling constraints as well. However, a state transition must take place in a FunState component before a function starts a new execution. Waiting for this transition may obstruct a high throughput streaming dataflow processing. In our approach to respecting the dataflow integrity, pre-defined control procedures are executed in a FSM in a process, while a complementary FSM concurrently controls the dataflow distribution in this process.

In [4], applications are modeled using Process Networks and SBF with non-static parameters. These applications are also mapped onto a FPGA and get configuration data from outside. However, the re-configuration is only possible after a complete network cycle. We want to be able to re-configure each individual periodic dataflow process during any period at run-time, without stopping the entire DFN.

An approach to dynamically reconfiguring a streaming application in a hierarchical SoC with a multiprocessor subsystem is presented in [5]. Processing tasks can be reconfigured through inserting reconfiguration tokens in the data streams. We avoid such insertions by physically separating dataflow and control paths in our implementations. Nevertheless, combining the generic services offered by the shell described in [5] with a standard task-level specification as in [6] would lead to optimized SoC implementations and re-usable modules.

2 Interface mapping

In this Section we first briefly review the modeling paradigm to abstract platform-specific processing, communication, and synchronization mechanisms to specifying Control and Dataflow processing applications in the large-scale system. Then we detail the mapping of the CN and the DFN into soft-cores and hardware of an FPGA-based platform, respectively, and the interfacing of the two networks.

2.1 Modeling paradigm

From a separation of concerns viewpoint it is interesting to specify the DFN and CN models independently from each other and to progressively refine them for HW/SW implementation on networks of platforms as shown in Figure 3. Thus, optimized implementations of the separated networks can be re-used from a high-level specification. The first step consists of refining the mechanisms that are involved in the communication between nodes in the models based on generic, platform-independent services of an abstract HW/SW task transaction level interface [6]. These services may be provided with functions (`Read, Write, Execute`) that manipulate vectors of arbitrary data types. These data types may be dataflow tokens in the DFN, or control packets in the CN, which consist of two parts: a header that indicates which command should be executed at what time by which node or leaf-node, and eventually control data information that comes with a specific command.

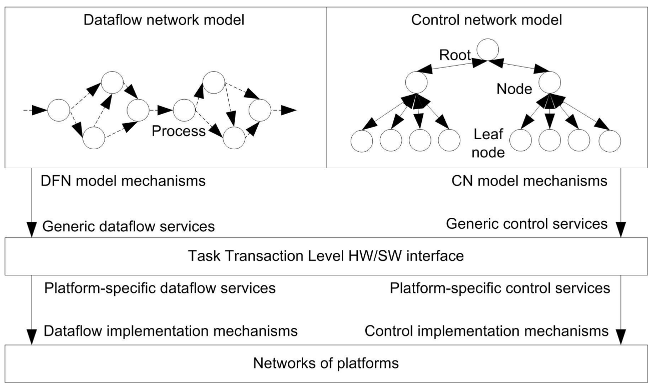

Dataflow network model
Control network model
Root
Node
Process
Leaf node
DFN model mechanisms
CN model mechanisms
Generic dataflow services
Generic control services
Task Transaction Level HW/SW interface
Platform-specific dataflow services
Platform-specific control services
Dataflow implementation mechanisms
Control implementation mechanisms
Networks of platforms

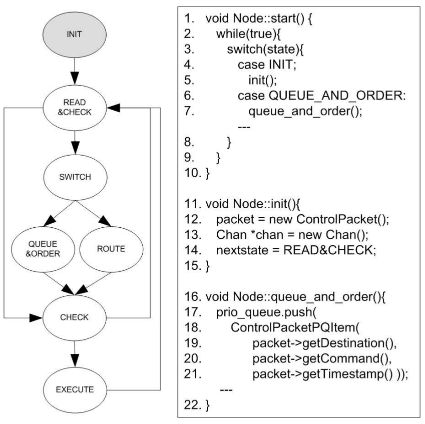
Fig. 3. Modeling Paradigm. The DFN and CN models are separated and gradually refined to be implemented in networks of platforms through a task transaction level interface.

The second step converts task-level representations to implementations through services that abstract platform-specific intricacies. When targeting a FPGA, each control node is assigned to a soft-core for which we generate code. Synchronization pulse trains are handled as interrupts through a real-time operating system. Control packets are defined in structures whose elements can easily be manipulated individually. These soft-cores communicate through embedded memories. Each dataflow process is assigned to a re-configurable processor that is generated in HDL. These dataflow processors exchange tokens through embedded memories, which are not shared with the memories that are used in the control network. Hardware IPs are wrapped in these dataflow processors because we do not want to get involved in low level functions design.

2.2 Node mapping

The state diagram of a control node executed in a soft-core is shown on the left-hand side in Figure 4 and the corresponding platform-independent code is given on the right-hand side (lines 1-10). The default state is represented in grey (`INIT`, lines 11-15) and corresponds to the definition of a packet and initialization of the communication channels in the CN. These channels are first checked for the presence of packets (`READ&CHECK`). When a packet is received, there are two possibilities (`SWITCH`): it is either sent to the appropriate destination in the hierarchy (`ROUTE`), or it is inserted in a priority queue (`QUEUE&ORDER`, lines 16-22). Finally, there are again two alternatives (`CHECK`): the command that is in the packet on top of the queue must be executed during the current period

(EXECUTE), else the communication channels are checked again until a new packet enters the node or a new command must be executed.



```
void Node::start() {
   while(true){
      switch(state){
         case INIT;
            init();
         case QUEUE_AND_ORDER:
            queue_and_order();
            ---
      }
   }
}

void Node::init(){
   packet = new ControlPacket();
   Chan *chan = new Chan();
   nextstate = READ&CHECK;
}

void Node::queue_and_order(){
   prio_queue.push(
      ControlPacketPQItem(
            packet->getDestination(),
            packet->getCommand(),
            packet->getTimestamp() ));
   ---
}
```

Fig. 4. Mapping a node as a FSM onto a soft-core: state diagram and corresponding platform-independent pseudo code.

As detailed in [9], the hardware realization of a KPN node (process) is made of four components: a `Dataflow Read Unit` that gets tokens from dataflow input channels, multiplexes and transmits them to an `Execute Unit`, which consumes these tokens, performs computation using an IP `Function Repertoire` and produces output tokens towards a `Dataflow Write Unit`. This unit demultiplexes the output tokens and sends them to output dataflow channels. The fourth component is the `Controller` that supervises the execution of the three other units. All these units are shown in Figure 5.

2.3 Interfacing a leaf-node with a process

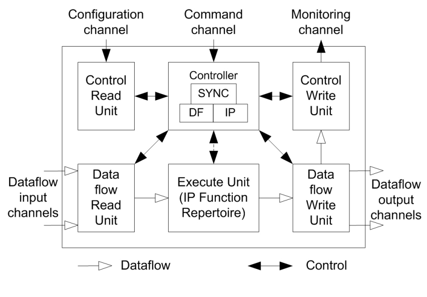


Fig. 5. Hardware implementation of the interface between a leaf-node and a process.

In the interface between a CN leaf-node with a DFN process, the additional configuration-data port is connected to a `Control Read Unit` as shown in Figure 5. The Dataflow Read Unit, Dataflow Write Unit and Function Repertoire get their own configuration parameters under the supervision of the Controller. The additional command port is connected to the Controller and the monitoring-data port is connected to a `Control Write Unit`, which probes the dataflow in the Dataflow Write Unit, as well as the state of the node in the Controller. Zooming in into the Controller, three distinct FSMs are executed. The behavior of these FSMs is depicted in Figure 6 (initial states are represented in grey).

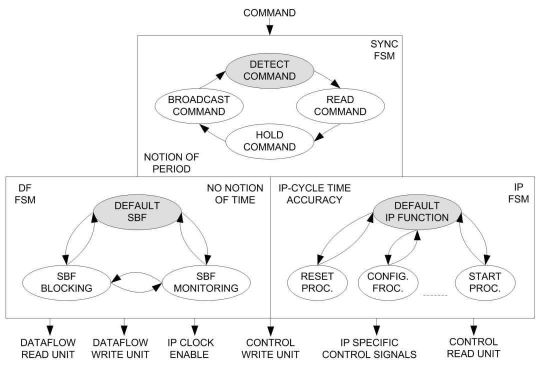


Fig. 6. Separation of concerns with three FSMs in a controller (`SYNC` for synchronization, `DF` for dataflow distribution and `IP` for IP-function control).

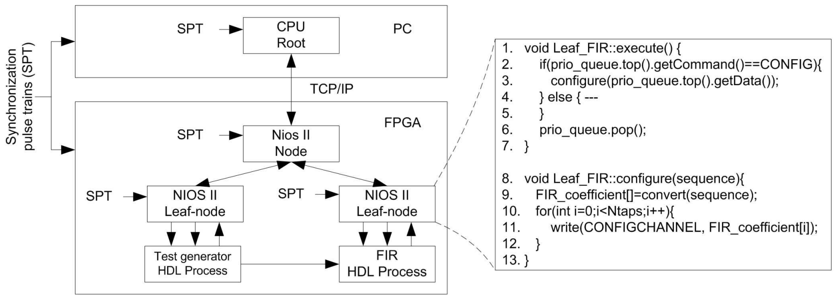
A `SYNC FSM` gets a command issued from a CN leaf-node and synchronizes its broadcasts to the two other FSMs when starting a new period. A `DF FSM` implements the behavior of an SBF [8] dataflow model of computation in a DFN process, without any notion of time, but a notion of order. It controls the Dataflow Read Unit, the Dataflow Write Unit and the clock enable signal of the IP Function Repertoire. The only command it can execute is a monitoring command. In this case it permits the Control Write Unit getting data from the Dataflow Write Unit. Concurrently, an `IP FSM` executes commands in states that encapsulate the corresponding IP-specific pre-defined control procedures, with an IP-cycle accurate notion of time. It generates control signals for the IP Function Repertoire and controls the flow of re-configuration data.

3 Case study

In Section 2 we detailed our approach to map a hierarchical CN and the interface with a DFN onto a FPGA. In this Section we present such an implementation and discuss the results concerning the separation of the two networks, the semi-automation, IP re-use and design-scaling.

3.1 Application

In this case study, the DFN is limited to a single KPN with two processes as shown in Figure 7. The first process generates periodic dataflow patterns (e.g. impulses, ramps or sinewaves) that can be re-configured (e.g. amplitude, frequency) by a leaf-node. The second process wraps an 8-taps FIR filter IP whose taps can be re-configured (e.g. low-pass, band-pass, high-pass characteristics depending on the operational mode) from a leaf-node as well. The two leaf-nodes and the node above in the hierarchical CN are executed in nios II soft-cores in a Stratix II FPGA [16], and support a portable real-time kernel (MicroC/OS-II [13]) to handle the synchronization pulse trains as interrupts. The node is the single entry point of the FPGA-based platform. It supports a portable light-weight TCP/IP stack from Opencores [18] to communicate with the root that is mapped on a CPU in a PC.



```
1.  void Leaf_FIR::execute() {
2.      if(prio_queue.top().getCommand()==CONFIG){
3.         configure(prio_queue.top().getData());
4.      } else { ---
5.      }
6.      prio_queue.pop();
7.  }

8.  void Leaf_FIR::configure(sequence){
9.     FIR_coefficient[]=convert(sequence);
10.    for(int i=0;i<Ntaps;i++){
11.       write(CONFIGCHANNEL, FIR_coefficient[i]);
12.    }
13. }
```

Fig. 7. FPGA implementation of a hierarchical control network to control and monitor a test generator and a FIR filter.

Re-configuring the filter coefficients requires converting the new coefficients to the filter-specific format because coefficients are stored in partial order and distributed in embedded memory segments. This is done in the Execute state of the leaf-node that controls the filter as shown in the pseudo-code in Figure 7. On the occurrence of a synchronization pulse train, a control packet is read from the priority queue. If the command requests re-configuring the filter, then a conversion program is called (lines 2-4) that converts the configuration data to the filter-specific sequence. This sequence is sent to the re-configuration channel (lines 10-12) and the control packet is removed from the queue after its execution (line 6). The leaf-node may then send a command to activate the re-configuration of the process as detailed in Section 2.

3.2 Results

Each nios II soft-core has been implemented in approximately 1,000 (Adaptive) Look-Up Tables (LUT) and 64kB of memory in this application. The middleware library took approximately 1.3MB of external SDRAM for each processor. These results could be improved by mapping nodes and leaf-nodes onto Application Specific Instruction-Set Processors (ASIP [11]) and by sharing the implementation of the middleware between all soft-cores, respectively. Figure 8 shows the impact of the FIR function-scaling on throughput (maximum frequency sustained by the dataflow in the process after synthesis of the process, on the left-hand side) and resource usage (LUT, on the right-hand side). Results are given for the interface presented in this paper (CN-DFN) and for a manufacturer-dependent dataflow only interface (Atlantic [16]). Our interface needs a few more resources since it includes both dataflow and control, and the loss in the throughput is still acceptable (less than 10%) in this case study.

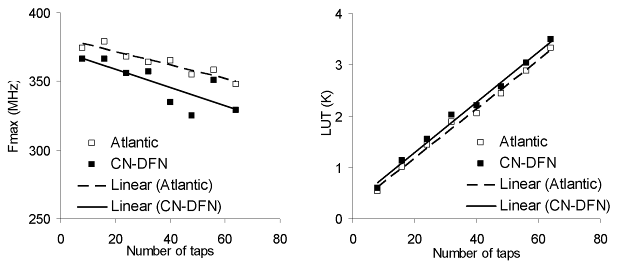


Fig. 8. Impact of design-scaling on throughput and resource usage.

Portable and speed-optimized HDL has been generated from high-level graphical specifications in StateCAD [17] for the three FSMs that are executed in the controllers of the DFN processes to wrap the hardware IPs. Thus, we avoided time consuming and error prone HDL handcrafted development. Our interfaces permit de-coupling low-speed clock domain(s) in the CN (soft-cores hardly run faster than 150MHz) from the high-speed clock domain(s) in the DFN. However, finding optimal buffer sizes to minimize the chance of blocking in the dataflow communication channels remains problematic. Moreover, the granularity of the dataflow processes in our prototype is large enough for the control network not to be critical since dataflow processes use more resources than a soft-core. Nevertheless their periods should not be shorter in future implementations so that soft-cores in the CN are not overwhelmed handling synchronization pulse trains as interrupts.

Although dataflow processes can be re-configured without the loss of data as in the case of the filter, re-configuring functions may induce transients in the data itself. Thus, the current practise is still to discard the data that is temporarily corrupted due to the transition.

4 Conclusion

We presented a prototype FPGA implementation of a hierarchical control network (modeled as communicating Finite State Machines) and its interfacing with a distributed dataflow network (modeled as communicating Kahn Process Networks). Nodes of the control network have been mapped onto soft-cores and interfaced with the nodes of the dataflow network, without sharing hardware resources, and based on HDL FSMs that isolate the synchronization mechanisms from the dataflow distribution and from the monitoring and control of the dataflow processing tasks, which are executed in hardware IPs. The performance of the interfaces in term of speed and resource usage allowed not to obstruct the performance of the (dominant) dataflow network in a case study. In addition, we anticipated design scaling and design re-use constraints by semi-automating HDL code generation in the mapping of the interfaces.

In the near future we would like to map such interfaces onto a network of re-configurable platforms, mainly consisting of FPGAs and CPUs, so as to evaluate its adaptiveness to larger systems. This mapping could be facilitated by keeping the signal processing separated from the control and monitoring in the architecture exploration of these large systems.

Acknowledgements

We would like to thank J. Bol, M. Lammertink and M. van der Kooij for their contribution to the implementation of the prototype.

References

1. B. Bhattacharya and S. Bhattacharyya, "Parameterized Dataflow Modeling of DSP Systems", In Proc. of the Intl. Conf. on *Acoustics, Speech, and Signal Processing (ASSP 2000)*, Istanbul, Turkey, June 2000.
2. B. Bhattacharya and S. Bhattacharyya, "Quasi-static Scheduling of Reconfigurable Dataflow Graphs for DSP Systems", In Proc. of the Intl. Workshop on *Rapid System Prototyping (RSP 2000)*, Paris, France, June 2000.
3. K. Strehl, L. Thiele, M. Gries, D. Ziegenbein, R. Ernst and J. Teich, "FunState - An Internal Design Representation for Codesign", *IEEE Trans. on VLSI Systems*, 2001.
4. H. Nikolov, T. Stefanov and E. Deprettere, "Modeling and FPGA Implementation of Applications using Parameterized Process Networks with Non-Static Parameters", In Proc. of the Symposium on *Field-Programmable Custom Computing Machines (FCCM'05)*, Napa, California, USA, Apr. 2005.
5. M. Rutten, E-J Pol, J. van Eijnhoven, K. Walters and G. Essink, "Dynamic reconfiguration of streaming graphs on a heterogeneous multiprocessor architecture", *SPIE Electronic Imaging: Embedded processors for Multimedia and Communications II*, vol. 5683, San Jose, CA, USA, January 2005.
6. P. van der Wolf, E. de Kock, T. Henriksson, W. Kruijtzer, G. Essink, "Design and Programming of Embedded Multiprocessors: An Interface-Centric Approach", *CODES+ISS'04*, Stockholm, Sweden, september 2004.

7. J. Lemaitre, S. Alliot and E. Deprettere, "Behavioral specification of control interface for signal processing applications", In Proc. of the Intl. Conf. on *Application-Specific Systems, Architectures, and Processors (ASAP'05)*, Samos, Greece, July 2005.
8. B. Kienhuis and E. Deprettere, "Modelling stream-based applications using the SBF model of computation", *Journal of VLSI Signal Processing-Systems for Signal, Image, and Video Technology*, 34(3):291-300, 2003.
9. S. Derrien, A. Turjan, C. Zissulescu, B. Kienhuis and E. Deprettere, "Deriving Efficient Control in Process Networks with Compaan/laura", *International Journal of Embedded Systems*, vol.1, issue 7, 2005.
10. J. Lemaitre, S. Alliot and E. Deprettere, "Requirements for interfacing IP-components in re-configurable platforms", *Journal of VLSI Signal Processing-Systems for Signal, Image and Video Technology*, vol.43, number 2, 2006.
11. L. L'Hours, "Generating Efficient Custom FPGA Soft-Cores for Control Dominated Applications", in Proc. of the *IEEE conference on Application-Specific Systems, Architectures, and Processors (ASAP'05)*, Samos, Greece, July 2005.
12. G. Kahn, "The semantics of a simple language for parallel programming", in Proc. of the *IFIP Congress 74*, North-Holland Publishing Co., 1974.
13. J. Labrosse, "MicroC/OS-II, The Real-Time Kernel, 2nd edition", CMP Books, 2002.
14. A.J. Boonstra, "Radio Frequency Interference Mitigation in Radio Astronomy", PhD thesis, T.U. Delft, The Netherlands, June 2005.
15. SKA, The Square Kilometre Array radiotelescope: www.skatelescope.org
16. Altera: www.altera.com
17. Xilinx: www.xilinx.com
18. Opencores: www.opencores.org