Hardwired Networks on Chip in FPGAs to Unify Functional and Configuration Interconnects

Kees Goossens[1,2], Martijn Bennebroek[3], Jae Young Hur[2], and Muhammad Aqeel Wahlah[2]
[1] Research, NXP Semiconductors, Eindhoven, The Netherlands, kees.goossens@nxp.com
[2] Computer Engineering Laboratory, Delft University of Technology, Delft, The Netherlands
[3] Research, Philips, Eindhoven, The Netherlands

Abstract

We propose that networks on chip (NOC) are hardwired in Field-Programmable Gate Arrays (FPGA). Although some area of the FPGA then has a fixed function, this loss of flexibility is outweighed by the following benefits. First, implementation cost is much reduced. Second, a hardwired NOC solves physical problems such as timing closure and high cost of global wiring. Third, dynamic partial reconfiguration can be better exploited. Compared to current soft or firm interconnects, a hardwired NOC poses fewer restrictions on the (re)placement of IP blocks in the FPGA. Finally, we also propose that the hardwired NOC is used for both the functional interconnect between the IP blocks and the configuration interconnect that transports the bitstreams. We give a detailed overview of our NOC architecture, and its configuration and programming. The proposed scheme enhances the on-line generation of bitstreams and the on-line verification of loaded bitstreams to detect tampering with the device. In our experiment, a hardwired NOC has acceptable ($\leq 10\%$) overhead for IP sizes with approximately 1400 lookup tables (LUT), enabling a fine-grained combined functional and configuration interconnect. A hardwired NOC offers significantly better functional performance than a soft NOC. Moreover, the configuration and programming of the hard NoC is much faster than when using a soft NOC.

1. Introduction

Field-programmable gate arrays (FPGA) are highly-programmable chips at the forefront of silicon technology scaling. Current FPGAs are divided in orthogonal configuration and functional regions, each with their own interconnect. In this paper we propose, first, to use a hardwired (or just: hard) network on chip (NOC) for the functional interconnect. Second, we advocate to use the same hard NOC as the configuration interconnect. In the remainder of the introduction we define some terminology, list the advantages of our approach, and give the outline of the rest of the paper.

1) Terminology: We call an intellectual property (IP) block *soft*, when after synthesis it is mapped on basic FPGA elements such as LUTs, before being placed and routed. A soft IP is independent of the device it is synthesised on. A *firm* IP is a soft IP that has optimised mapping, placement, and routing, taking into account the particular FPGA device implementation. An IP is *hard* when it is directly implemented in silicon. Existing examples are embedded multipliers, microprocessors, RAMs, and high-speed IO interfaces [37]. We define *(re)configuration* as the installation of new functionality (IPs) in the FPGA by loading reconfiguration regions with bitstreams. *Dynamic partial reconfiguration* means that only part of the FPGA is reconfigured and that this happens while part of its functionality remains operational. Only soft and firm IPs need to be configured. An IP is *programmed* after it is configured, if necessary, which entails changing the state of its registers when it is in functional mode. These registers are usually memory-mapped so that they can be accessed through the functional interconnect, such as a bus or a NOC. Note that in our case the NOC unifies the configuration interconnect used to transport bitstreams, the functional control interconnect through which IPs are programmed, and the functional data interconnect to transport data between IPs. The first is used when the system is (partially) reconfigured, and the latter two are used when the system is (partially) in functional mode. In the remainder, we will not further distinguish the control and data interconnects, calling them functional interconnect.

2) Advantages of Hard NOCs: Hard interconnects use the native silicon technology rather than configurable elements of the FPGA. They are reported to occupy 35 times less area, operate 3.5 times faster, and also use less energy [21]. However, the FPGA is less flexible because some silicon area is committed to a fixed function [37]. We argue that the cost:performance gain far outweighs the loss of flexibility. Modern FPGAs are complex embedded systems on chip that are not monolithic functions, but composed of many reconfigurable blocks. The IPs communicate using standard communication protocols, e.g. AXI [3], OCP [25], and DTL [27] implemented by a functional interconnect, such as busses, switches, and NOCs. Hence, a functional interconnect is required, whether it is soft, firm, or hard. A hard interconnect must be dimensioned for the worst-case, whereas a soft interconnect can be tailored to the running application. However, the performance:cost ratio of the hard interconnect is much better than of the soft interconnect. Thus the hard interconnect can be "over-dimensioned" significantly before we lose flexibility. Also, in contrast to soft interconnects, hard interconnects are not configured, which saves configuration footprint (bits) as well as configuration time. We refer to Section 4 for the entire analysis.

When a system is dynamically partially reconfigured, the functional interconnect must be updated, i.e. reconfigured and/or reprogrammed. This requires that IPs and interconnect are decoupled in several senses: physically (e.g. to avoid glitches), in placement (reconfiguration and functional regions of IP and interconnect must be disjoint), and logically (there should be no communication to/from IPs that are reconfigured, and communication between other IPs should not be affected). Hard interconnects are per definition disjoint from soft IPs and reduce layout restrictions. Programmable interconnects such as busses and NOCs only require reprogramming. Hence, communications between IPs that are not reconfigured are not disturbed during partial reconfiguration from a physical and placement point of view, and from a logical point of view too, if the interconnect offers guaranteed communication services. Moreover, programming is faster than reconfiguration, which is required by e.g. dedicated through-routed point-to-point wires. By virtue of their better performance, hard busses and NOCs are also reprogrammed quicker than the soft counterparts (see Section 4.3).

Busses and switches are single-hop interconnects, i.e. their arbitration does not scale with the number of attached IPs, unlike NOCs. Moreover, interconnects are physically distributed over the chip and deep-submicron problems related to long wires (such as low speed, signal degradation, etc.) complicate timing closure between IPs. NOCs can address these issues by a globally-asynchronous locally-synchronous design style and by replacing long global wires with optimised segmented wires between routers [31]. This is only possible when the NOC is hard (and to a lesser extent when firm). Finally, multi-hop interconnects, such as NOCs, are mandatory to offer transparent multi-chip or multi-board communication.

3) Unified Interconnects: We advocate merging configuration and functional interconnects because the configuration interconnect can benefit from the higher performance of the hard NOC. However, the NOC should support real-time streaming of bitstreams, i.e. avoid interference from other (functional) traffic. Unifying the configuration and functional NOCs allows configuration bitstreams to be treated like normal data. This allows many new applications. For example, bitstreams can be generated at run-time, e.g. for run-time tuning of IP configuration. Bitstreams can also be encrypted, decrypted, and check-summed to detect bitstream tampering. In addition, IP can be relocated from one part of the FPGA to another, rather than reloading from the configuration memory (flash). For example, we can load and relocate embedded built-in-self-test (BIST) engines to structurally test the FPGA, by generating test patterns on the FPGA itself and by using the hard functional NOC as a test-access mechanism (TAM) [2][1].

The remainder of this paper is structured as follows. The contributions of this paper are described when we review related work. In Section 3, we explain our NOC architecture. After describing how a system is configured and programmed, we combine all elements in a complete system. Various possible extensions and new applications of the hard NOC are then discussed. In Section 4, we compare a soft, firm, and hard implementation of the Æthereal NOC [12]. Finally, conclusions are drawn in Section 5.

2. Related Work

Scores of *soft* interconnects ranging from point-to-point [19], busses [5][30], cross-bars [24][34], to NOCs [22][20][28] have been presented in the literature. They implement different cost:performance trade-offs. Larger and faster FPGAs can contain many IPs and multi-hop network solutions are sure to gain popularity. Fewer *firm* interconnects have been presented [5][18][6], where the interconnect is implemented using native on-chip resources. This increases the performance and lowers the cost of the interconnect compared to soft interconnects, but their implementations are dedicated to particular FPGA architectures. [5] presents four communication schemes, including bus macros, shared memories, linear array multiple bus, and external crossbars that used for different communication modes. In [18], a reprogrammable interconnect is implemented based on a LUT-based bus macro and is used to dynamically reconfigure the attached IPs. In [6], a large (928×928 bits) crossbar using native programmable interconnects and LUTs is presented. However, standard IP communication protocols such as AXI [3] contain several hundred of wires per IP. Few IPs can therefore be connected to a single switch.

For dynamic partial reconfiguration of IPs, the functional interconnect and IPs must be disjoint, otherwise they must be reconfigured simultaneously. By *reprogramming* the functional interconnect, IPs can be dynamically added and removed through partial reconfiguration, e.g. [5][30][22][18]. Only few works *reconfigure* the soft interconnect itself. [19] dynamically reconfigures a point-to-point soft interconnect and [28] shows how a soft NOC can be partially reconfigured by adding or removing router modules at run time. A hard NOC is only (re)programmed. The cost of the hard interconnect is much lower than a soft functional interconnect, lowering the need for dynamically changing the topology. Moreover, because the hard NOC is per definition disjoint from soft IP, eliminating any interference during programming and/or reconfiguration.

Only two groups have reported on *hard* NOCs in FPGAs [17][16][8][11]. In [17][16], a system model is explored with SystemC simulation, and no hardware architecture is presented. In addition, the configuration and programming of the NOC are not defined in [17][16]. The work of [8][11] proposes to use a hard NOC as the functional interconnect. Although the basic idea of a hard NOC is introduced, no architecture details are provided. Our work differs from [8][11] in the following. Foremost, we combine the configuration and data interconnects in the hard NOC, which has not been proposed by any prior work. Furthermore, our partitioning of the network interface (NI) in hard and soft regions draws the distinction more clearly at the network versus transport layer (see Section 3.2). For example, we firmly place routing in the hard NI kernel domain. Moreover, in [16] the architecture is based on tiles (functional regions), which can communicate only through the NOC. As explained in Section 3.3, we do not partition the FPGA in distinct functional regions and also keep them orthogonal to the configuration regions. Next, we define the (re)configuration and (re)programming steps required to boot a system. The requirement for guaranteed communication services (GS) to support real-time streaming of bitstreams is not met by their NOC. Also, we support arbitrary topologies without inducing any extra (routing) complexity [15], which is useful, as shown in Section 3.4. Finally, we present a comprehensive analysis of cost, performance, and programming footprint, based on experimental results on a Xilinx Virtex-4, which is lacking in [17][11].

NOCs are also (naturally) emerging to deal with inter-chip communication due to their intrinsic multi-hop nature [32][26][7]. [32] motivates extending the functionality of an ASIC by connecting the (hard) NOC on the ASIC to a soft NOC on an FPGA companion chip. [26] goes one step further and introduces a three-level hierarchical network. Intra-FPGA point-to-point links are implemented with Fast Simplex Links, inter-FPGA communication uses Rocket IO, and an ethernet switch implements inter-board communication. The Berkeley Emulation Engine [7] employs a parallel bus for the inter-FPGA communications. They use three types of global communication networks, a low-latency global communication tree, a high-bandwidth nonblocking crossbar, and a 10/100 Base-T Ethernet. Multi-gigabit transceivers (MGTs) also were used for 2D-mesh inter-FPGA communications. Inter-chip and inter-board NOC are extensions of the work presented here, as discussed in Section 3.4.

3. Hard NOC Architecture

3.1. Overview

Figure 1 shows the differences between the FPGA architectures with conventional (a) and new (b) interconnects in a system context. Figure 1(a) illustrates a conventional FPGA with orthogonal configuration (thin lines, dark grey blocks) and functional (fat lines, light grey blocks) interconnects. The minimum coherent reconfiguration region is the minimum number of reconfiguration and functional regions that coincide. For example, in the case of Virtex4, it is equal to 22 frames, which cover 16 CLBs [36]. Figure 1(b) shows

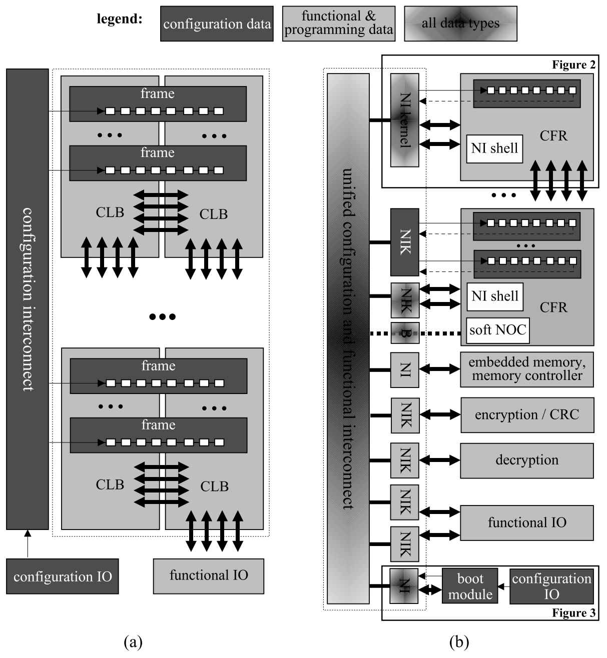


Figure 1. (a) Conventional and (b) new configuration and programming.

an example instance of the proposed architecture, where the configuration and functional IO are connected to the same hard NOC, which now has two shades of grey. For simplicity, the configuration and functional regions (frames and CLBs) have been drawn together as a CFR (configuration and functional region). However, just as in the conventional FPGA, they can be orthogonal. The local configuration interconnect/region (drawn as a chain of configuration registers) connects to the NOC to receive bitstreams. CLBs are connected to their neighbouring CLBs in the same or different functional regions, as usual, but a functional region (containing a number of CLBs, BRAMs, etc.) is also connected to the NOC. The configuration IO, usually connected to external flash memory, is shown at the bottom right. The boot module (Section 3.3) bootstraps the system by configuring and programming it through the unified interconnect.

We describe NOCs generally, before detailing network interfaces (Section 3.2), and their essential role in system configuration and programming (Section 3.3). Section 3.4 combines all these elements in our proposed unified hard configuration and data interconnect. NOCs contain two kinds of components: routers that move data around (usually packets), and network interfaces (NI) that convert the NOC-internal data format (e.g. packets) to the protocol required by the NOC clients (e.g. AXI, OCP, DTL). Direct mesh topologies, where every router is connected to a NI, are popular. However, it makes sense to adapt the NOC topology to the resources of the particular platform FPGA to provide ample bandwidth to high-speed off-chip IO links, configuration IO, external memory interfaces, or large shared on-chip memories. As illustrated in Figure 1(b), NI kernels and shells can be hard or soft. One or more NIs may be attached to one IP, such as functional IO.

Real-time applications often require communication with a guaranteed minimum bandwidth, and maximum or even fixed latency. A number of NOCs offer these guaranteed communication services (GS, often also called quality of service) [12][23][4], and almost all NOCs offer basic lossless communication but without real-time guarantees (best effort, or BE). Most NOCs implement transaction-based protocols, such as AXI and OCP, and some also implement streaming data protocols, such as DTL's peer-to-peer streaming data (PPSD) [27]. The former uses a distributed-shared-memory communication model requiring read/write commands, data, and address, each of which uses a valid/ready handshake. The latter is a much simpler basic data pipe, containing only the write data group, usually with a valid/ready handshake. This has consequences for which part of a NI is hard or soft (see Section 3.4).

Bitstreams are delivered to the region being reconfigured as a continuous data stream that should not be interrupted. However, during a dynamic partial reconfiguration the NOC may be in use by an application which may interfere with the bitstream's timing. Hence, the FPGA reconfiguration interconnect should offer a GS (fixed latency) communication between the bitstream source (e.g. off-chip flash) and the reconfigured region. Similarly, NOCs with GS have been proposed to replace SOC test access mechanisms (TAM) that transport test patterns and responses to test the IP for errors for manufacturing test or for field testing [1]. The alternative, stalling the test or configuration bitstream when the BE communication is interrupted, requires expensive holdable state elements.

3.2. Network Interface Architecture

The details of router architectures are not relevant here, details can be found in e.g. [4][12]. NIs are often split in two parts, see Figure 2: the kernel (performing network layer functions) and the shell (for transport layer functions) [29]. NOCs with GS communication require resource reserva-

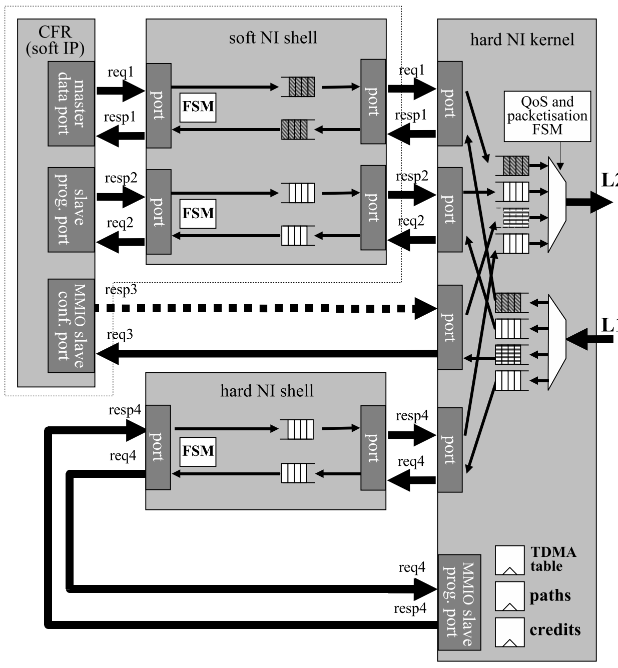


Figure 2. IP that cannot reprogram/reconfigure, and NI.

tions, in the form of *channels*, to move data from a source NI to a destination NI. Two IPs communicate using a request channel and a response channel. Each channel has its own quality of service (QoS), i.e. the allocated bandwidth and latency. The NI kernel is responsible for receiving packets from the router on link L1 and depacketising them. Conversely, it packetises data and sends them to the router on link L2, according to the channel's QoS. Our example NOC, Æthereal [12] uses a virtual-circuit TDMA scheme to offer strict GS bandwidth and latency guarantees, and also offers BE communication. The kernel therefore contains a programmable TDMA table, credit counters for end-to-end flow control, and per channel the programmable path that its packets take through the NOC. The kernel contains a MMIO port through which these registers are programmed. In fact, the kernels are programmed using the NOC itself [14], by looping an output port to the programming MMIO port, as shown at the bottom of Figure 2. Note that the loop-back and NI shell are omitted from Figure 1, for lack of space. More details on programming follow in Section 3.3.

The ports on the NI kernel are streaming data ports (PPSD), i.e. a fixed word-width with valid/ready handshake to cross from NOC to IP clock domains. The response is optional. Streaming IP can connect directly to these ports. It is the job of the NI shell to convert the data stream to a distributed-shared-memory protocol, such as AXI. The three FSMs implement the valid/ready handshakes per command, read/write data groups, and their serialization to/from the NI kernel ports, etc.

The IP shown in Figure 2 has a master data port, on which it sends read/write requests to a slave somewhere on the NOC, and a slave MMIO programming port, like the NI kernel, over which read/write requests are received. The slave configuration port is a new addition proposed in this paper. Because configuration data is streaming data and not shared memory communication, the IP configuration port is connected directly to the NI kernel. From the NOC perspective, it is just another port, over which data is communicated with fixed latency. This is achieved by programming sufficient bandwidth allocation in the kernel's TDMA slot table and by sizing the buffers in the NI kernel/shell to absorb any jitter from unevenly-spaced TDMA slots [9].

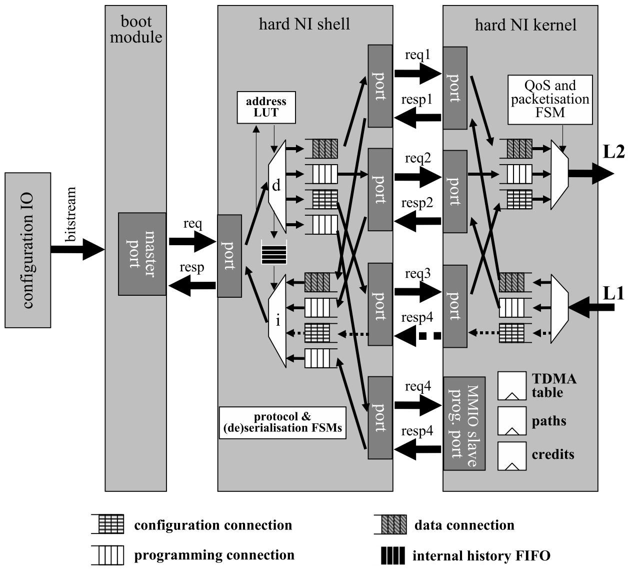


Figure 3. Boot module able to reprogram/-reconfigure and its NI shell & kernel.

The IP in Figure 2 can neither program its NI nor influence the configuration data that it receives. Figure 3 illustrates a boot module that can do this. It can communicate with the local NI and four remote NIs or IP ports using distributed-shared-memory transactions. The address lookup table demultiplexes requests to one of the four channels, based on the address. The FIFO between the request demultiplexer "d" and the response multiplexer "i" ensures that split pipelined transactions to different slaves are presented in the right order to the master (for details see [29]). Using the top three channels in Figure 3, the boot module can exchange data with IPs. However, they can also be used to program remote NIs and IPs (functional regions), and to configure a configuration region that is attached to a remote NI. Hence, the importance of the demultiplexer "d" is its type conversion role, i.e. that the data produced by the boot module in functional mode is interpreted as "normal" data, MMIO programming data, or as a configuration bitstream, depending on which port the channel is connected to at the receiving side. The next section describes how a system is programmed and configured.

3.3. System Configuration & Programming

The key to system programming is the channel at the bottom of Figure 3 that is hardwired to the NI kernel's MMIO programming port. Unlike the IP in Figure 2, the boot module can bootstrap the system by programming the NOC and by configuring and programming other IPs. Normally, a CPU or a hard (secure) boot module boots the system [10]. First, the local NI is programmed with a channel to a remote NI. The new channel is used to program the remote NI. In this manner, the whole NOC can be programmed, i.e. programming and data channels are set up. Following this, the IPs are programmed (initialised and started) on their MMIO ports[29][14][13]. At this point the application runs.

1) Soft or Firm NOC: The scenario sketched above is sufficient for a hard ASIC. But for chips with configurable components (FPGAs, but also ASICs with embedded FPGA) a configuration phase is required. Figure 4 shows how an FPGA application is bootstrapped conventionally. First (dot-dashed lines), the boot module (if not a hard processor, such as a PowerPC), interconnect, and IP are all configured by copying a bitstream from a configuration memory (e.g. flash) using the conventional configuration IO and interconnect. After a functional reset, the boot module programs the NOC and then the IP. A solid line indicates that the component is functionally active, and a dashed line that it is being programmed. Finally, the application runs.

(Partial) reconfiguration of the system, shown after the vertical bar, operates identically. However, care must be taken that those parts of the system that continue to operate are shielded from parts that are reconfigured [28]. As shown in grey text, the interconnect is reconfigured too. The IP and NOC must occupy different reconfiguration regions (e.g. CLB columns). In fact, one of the reasons for using a programmable soft or firm interconnect, such as a NOC, instead of a non-programmable soft interconnect (e.g. point-to-point wires) is that it can be left in place, and reprogrammed only, before IP are reconfigured and reprogrammed. The configuration interconnect is marked as pervasive because it reaches all configurable elements in the FPGA from the configuration IO connected to the (off-chip) bitstream memory.

2) Hard NOC with GS: Figure 5 shows how to configure a system when a hard NOC is used as the configuration interconnect. First, notice that the NOC is no longer configured (no dot-dashed line). Second, IPs are configured only after the NOC has been programmed (dashed line) because the bitstreams are transported using the NOC in functional mode. For a (partial) reconfiguration, the following steps are required. First, the IPs to be reconfigured are stopped by programming their MMIO ports. Then, via the NOC they

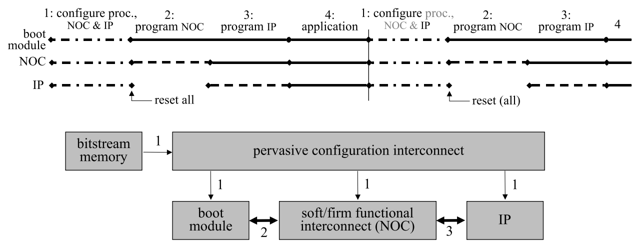


Figure 4. Current configuration & programming.

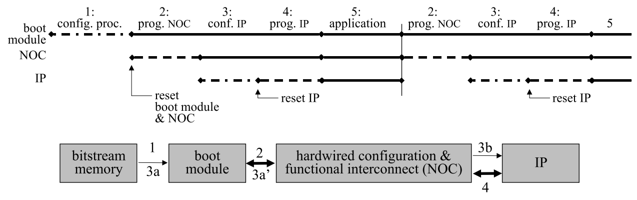


Figure 5. New configuration & programming.

receive new bitstreams and are reset on their MMIO ports. The NOC may be reprogrammed with the new application mode as well. Recall that bitstreams are streaming data and are usually not interruptable during their transport from bitstream memory to the IP (reconfiguration region). The fixed-latency GS communication service of the NOC is essential to avoid any (temporal) interference, because during the partial reconfiguration other IPs continue to operate and communicate using the NOC. Thus, our hard NOC offers two essential qualities: reprogramming instead of reconfiguration and guaranteed (fixed-latency) communication.

The configuration interconnect is now split in two parts. First, the single link between configuration IO and the boot module (Figure 3 and arrow 3a in Figure 5). The boot module connects to the hard NOC. Second, once the bitstream arrives on the other side of the NI (e.g. the CFR of Figure 2) it enters the local conventional configuration interconnect, to configure the configuration region (e.g. one or more frames) connected to the NI port (arrow 3b in Figure 5).[1] The FPGA is divided in configuration regions, each of which is reachable only via a single NI kernel of the hard NOC. But our functional regions are connected to each other, like CLBs are, and are also connected to the NOC's hard NI kernels. We do not divide the FPGA in functional regions (also called tiles) that can communicate only via the NOC, as advocated by [5][16][11]. This unnecessarily causes problems, such as optimally packing multiple IP in multiple regions, or restricting the IP size to that of a region.

3.4. Proposed FPGA Overview

In this section, we discuss Figure 1(b) in more detail. The configuration IO is connected to the boot module, which may be a simple hardwired DMA and/or secure programmable processor. The boot module shown in Figure 1(b) is attached to the hard NI shown in Figure 3. The functional IO is connected to a hard NI kernel because it produces and/or consumes streaming data. This is not the case for all hard functional blocks. For example, the embedded memories, memory controllers, and hard processors (not shown) connect to a hard NI kernel *and shell* because they use distributed-shared-memory transactions. The local configuration interconnect/region (drawn as a chain of configuration registers) connects to a hard NI kernel to receive bitstreams. CLBs are, as usual, connected to their neighbouring CLBs in the same or different functional regions; but a functional region (containing a number of CLBs, BRAMs, etc.) is also connected to the NOC with a hard NI kernel. The NI shell is soft and will be implemented in the functional region. The two CFRs illustrate that configuration and functional regions may share the same NI kernel, and that the number of NIs and their number of ports may vary. The sizes of configuration and functional regions are discussed in Section 4.

1) Hard versus Soft: In Figure 1(b), we discuss which parts of the NOC can be hard, soft or firm. Routers are best implemented hard. NI kernels (cf. Figures 2 and 3) have fixed hardware, such as (de)packetisation hardware. Channel FIFOs, per-channel administration (paths, credit counters, etc.), and TDMA slot tables can all be implemented soft. This makes no sense for FIFOs because the cost of LUT-based FIFOs is prohibitive, compared to BRAM-based FIFOs. Given that a number of hard FIFOs within reasonable distance of the hard (de)packetisation will be fixed, it is best to dimension the TDMA and channel administration tables for this worst case, to hardwire the entire NI kernel.

The NI shell, however, is soft for the following reasons. First, the port protocol depends on the IP and a single system often contains a number of different port protocols. Second, the depth of a channel FIFO depends on the required bandwidth and latency (e.g. it must hide jitter due to the distribution of TDMA slots and the round-trip latency of end-to-end flow control), which depends on the application. The channel FIFO is for a small part in the NI kernel, where it has a fixed size, and for the remainder in the NI shell. The only NI shells that are hard are the following. First, those connected to the NI kernel's MMIO programming port, because it is always required and uses a fixed protocol. Those NI shells that connect to hard IPs that use shared-memory protocols, such as the boot module, embedded memories, memory controllers, hard embedded processors, etc.

2) Hard NOC Extensions: The first extension is to allow the hard NOC to be expanded by soft routers and NIs. This is useful when functional regions are large and more NI kernel ports are required than are present on the hard NI kernels near the region. This is implemented by the fat dashed line from the hard router network to the lower CFR in Fig-

[1]In a first instance we only upload bitstreams from boot processor to IPs, and the response channels are then absent. i.e. we omit the dashed channels at the boot module's NI (Figure 3), and the slave IP NI (Figure 2). We re-introduce the response channel and alternative 3a' in Section 3.4.

ure 1(b). However, because the hard NOC will be running at higher frequencies than can be achieved with a soft NOC, it passes through the bridge marked "B," which implements a clock domain crossing between hard NOC and the functional region. The bridge must be hardwired. When the soft extension is not used, the link from the functional region to the bridge can easily be "tied off" by setting the packet-type side-band signal to "empty" [29]. In the opposite direction, towards the functional region, no connection will be programmed along the unused link. A related extension is the use of the functional IO to connect the NOCs on multiple FPGAs, to create a multi-FPGA NOC [32][26][7]. The NI kernel on one NOC converts packets to streaming data which is transported over the functional IO to the other FPGA, where it is repacketised by the NI kernel on the other NOC, which can be soft, firm, hard.

3) Dynamic Bitstreams: The proposed architecture already allows dynamic embedded bitstream generation, in the sense that any IP can functionally generate a bitstream at run time. Modern FPGAs support the on-line partial configuration, for example using Xilinx ICAP(internal reconfiguration access port). However, these conventional configuration interconnects constitute dedicated physical point-to-point networks, which suffer from scalability problems and increased wire delays. To upload the bitstream to a configuration region, the boot module must program the NOC to connect the data port of the IP to the configuration port of the configuration region. (As briefly discussed in Sections 3.2 and 3.3; cf. arrow 3a' in Figure 5.) We envisage computing or modifying a hardware accelerator. For example, optimising a FIR filter by configuring multipliers optimised for the coefficients instead of generic (larger, slower) multipliers and programming the coefficients via MMIO. In fact, bitstreams are normal data and can, for example, be loaded in encrypted form from the configuration IO, be sent to a (hard) decryption engine on the NOC, before being sent to a configuration region.

The proposed extension goes one step further, by allowing bitstreams to be read or downloaded from as well as uploaded to configuration regions. This is implemented by the dashed arrows from the reconfiguration regions to the NOC in Figures 2, 3, and 1(b). Apart from speculative applications, such as embedded genetic algorithms operating on bitstreams [33], it allows dynamic relocation of IPs by moving the bitstreams, in conjunction with techniques like those of [22] where bitstreams are reloaded rather than moved. As an example, we can load and relocate embedded built-in-self-test (BIST) engines that generate test patterns on the FPGA that are distributed to functional regions (e.g. using multicast). We use the hard NOC as a test access mechanism (TAM), which requires adaptations to the normal test shells, described in [1]. In Figure 2 this would entail adding a streaming data port on the CFR for test data. The BIST engines check the test responses that are streamed back over the hard NOC. In this way, the costly test time is reduced by testing at higher (functional) speed, and by reducing the amount of data that must be streamed to and from the automated test equipment (ATE) over limited IO [2].

Functionally operating on bitstreams also can be used to check if IP configurations have been tampered with since they were installed. This can be achieved by streaming the bitstream to a (hard) encryption or CRC IP and comparing the output with the original encrypted bitstream or a smaller checksum. The use of a functional interconnect for decrypted bitstreams is in general not safe, because it allows a malicious IP to capture secret information. However, first note that a NOC is a (virtual) point-to-point and not a broadcast interconnect. Second, all channels are either hardwired (i.e. the path between source-destination pair is fixed), in which case only the source and destination IP have access to the data. Or, the channel is programmed at run time. By bootstrapping from a (single) secure boot module, no IP can communicate (send or listen) until the boot module creates a channel to another IP [10]. Moreover, NOCs with guaranteed communication services also decouple the temporal behaviours of communicating IP. This removes the possibility to obtain secret information either from the timing of communication (events), or from a malfunctioning system after injecting spurious data.

4. Results

We now compare various implementations of Æthereal NOC instances in terms of area, functional speed, footprint size, configuration throughput, programming speed, and general flexibility. The automated Æthereal design flow generates application-specific NOCs, based on the specified communication requirements of multi-mode applications [13]. The topology, TDMA table size, and FIFO sizes are all tailored to the application. The flow output includes the technology-independent RTL VHDL of the NOC, test bench, traffic generators, embedded C code to program the NOC, scripts for gate-level synthesis, and scripts for scan-chain insertion. It is also possible to specify a topology and generate the NOC programming code at a later stage, which is required for our purposes. We specified a simple system, consisting of two masters and three slaves, where each master can communicate to all slaves simultaneously. One master is the boot module. The NOC contains one router and five NIs, where the NI kernels/shells are shown in Figures 2 and 3. Each IP has its own NI and the single router has five bidirectional links. The NOC components use optimised hardware FIFOs [35] for high speed and small area. NI and router FIFOs contain 16, resp. 24 words.

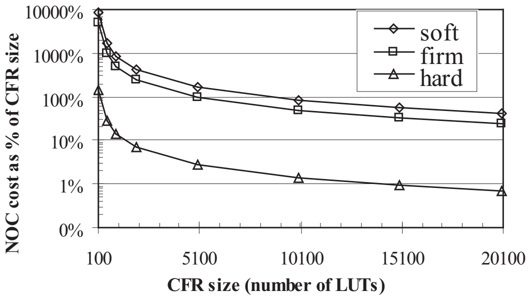
4.1. Area

Several router and NI instances have been implemented to timing back-annotated layout [12]. Based on these layout instances, our design flow accurately estimates the area of any generated NOC instance in $130nm$ CMOS technology. The area results for the router and NI instances are shown in "hard" rows of Table 1. We also synthesised, placed, and routed (to the pads) the NOC onto a Virtex-4 XC4VLX200ff1513-11, for which we used Xilinx ISE 8.2. The Virtex-4 contains 178176 LUTs in total. The Virtex-4 is fabricated in CMOS technology with a $90nm$ Copper CMOS process [36]. We used two versions of the NOC: one uses LUTs for all FIFOs and the other uses BRAMs. These are referred to as "soft" and "firm" in Table 1. We mapped a whole NOC and extracted the number of LUTs for a single router-NI pair, to take into account routing overhead, etc.

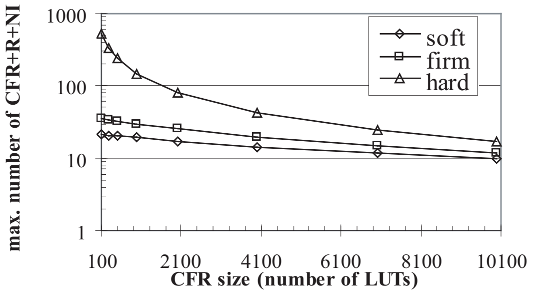
Table 1. Area of NOC Components. Note the area of firm components does not include BRAMs.

		# LUTs	mm^2 $130nm$	mm^2 $90nm$	mm^2 $90nm$
hard	router	-	**0.13**	0.065	-
	NI	-	**0.33**	0.167	-
soft	router	**2658**	-	2.28	*5.48*
	NI kernel	**3470**	-	3.58	*7.16*
	NI shell	**2171**	-	2.24	*4.48*
firm	router	**1988**	-	1.70	*4.10*
	NI kernel	**1524**	-	1.57	*3.15*
	NI shell	**1501**	-	1.55	*3.09*

In Table 1, the bold numbers are used to derive the remaining data in the following manner. We derived the $90nm$-equivalent area of the $130nm$ hard NOC components by dividing by two. For the fifth column, the area of soft router was computed by multiplying the area of the hard router by a factor 35, reported to be the cost ratio between ASIC and FPGA in [21]. The hard NI area is similarly scaled and divided over the soft NI kernel and shell in the ratio of their number of LUTs. The same holds for the firm router and NI, except that the BRAM area (used for the FIFOs) is omitted. Moreover, the RTL has not been optimised to minimise the LUT area. Hence we underestimate the area of the firm components. We ensured that all calculations, here and below, are conservative, i.e. underestimate cost of the soft and firm NOCs, and overestimate cost of hard NOCs. As an example, an alternative calculation of soft / firm router and NI area, shown in italics in the right-most column of Table 1, uses the area per LUT. The area is computed by dividing the number of required LUTs by the die size of the Virtex-4 device ($735mm^2$, estimated from `www.fpga-faq.org`). We divide the die size by 2, considering that the device contains embedded blocks (such as I/O pins, DSP, memories). In this case, the area ratio of soft:hard is above 70. Our analyses remain within the same order of magnitude. Note that the area for the soft/firm NOC does not include the area of the configuration interconnect, which the hard NOC also implements. Our estimates are therefore conservative.



(a) Percentage overhead for varying CFR sizes.



(b) Number of IPs on a Virtex-4 for varying CFR sizes.

Figure 6. NOC cost for varying CFR sizes.

In Figure 6(a), the cost of a router and NI is plotted against the size of the CFR. As an example, for soft, firm, and hard NOCs, the overhead reaches 830%, 500%, and 14% for a CFR size of 1000 LUTs, respectively. This means that the area overhead of hard NOC is acceptable for typical IP size, while the area overhead of soft/firm NOC is unacceptably large. Figure 6(b) depicts the number of (equal-sized) IPs that can be mapped on a Virtex-4. As an example, for soft, firm, and hard NOCs, the maximum number of IPs are 19, 30, and 144 for a CFR size of 1000 LUTs, respectively. This means that sufficiently many IPs can be mapped when the hard NOC is accommodated, while only small number of IPs can be mapped when the soft/firm NOC is utilized. Hence a soft NOC has less than 10% area cost only for a small number of large ($\approx 80,000$ LUTs) CFR, for which a NOC is not attractive to use. A hard NOC is already attractive for small CFRs (≥ 1400 LUTs), allowing over a hundred of CFRs on a Virtex-4. Firm NOCs score in between, but it is difficult to state when they are attractive, given the current data.

4.2 Functional Performance

In this section, we compare the speed of the NOCs. The hard implementation of Æthereal operates at (worst-case) $500MHz$ in $130nm$ [12]. The soft NOC operates at $118MHz$ (the NI) and $124MHz$ (the router) in the $90nm$ Virtex-4 device. Although the speed of the soft NOC can undoubtedly be improved, a speed improvement of a factor 5, which ignores the speed increase from ASIC $130nm$ to FPGA $90nm$, is a conservative estimate. For a fair benchmark of soft and hard NOCs, the performance:cost ratio ($bit/sec/mm^2$) should be compared. With equal topologies and architectures, we need to only compare the raw link bandwidths (32 bits × operating frequency) per area:

$$\begin{array}{lll} \text{soft} & 32bits \times 118MHz/8.10mm^2 = 466 & \mathbf{1} \\ \text{firm} & 32bits \times 122MHz/4.83mm^2 = 808 & \mathbf{1.7} \\ \text{hard} & 32bits \times 500MHz/0.23mm^2 = 69565 & \mathbf{149} \end{array}$$

Thus, the performance of firm NOC is 1.7 times better than a soft NOC and a hard NOC is 149 times better.

4.3. Configuration and Programming

In this section, we quantitatively compare the number of bits and the required time for configuration & programming.

1) Conventional Configuration: The configuration unit of a Virtex-4 device is a frame, containing 41 32-bit configuration words [36]. SelectMAP provides configuration at a rate of $1.9Gb/s$ (32-bit interface at $60MHz$ [36]). Therefore, a frame can ideally be configured in $41word \times 32bit/1.9Gbit/sec = 0.7\mu s$. A CLB column is the smallest coherent reconfiguration unit, which, containing 22 frames, takes $0.7 \times 22 = 15\mu s$ to configure.

2) NOC Programming: A NOC, whether soft or hard, must be programmed as described in Section 3.3. Each connection (a pair of request-response channels) in Æthereal has between 832 and 2096 state bits [14]. This requires $2.5\mu s$ to program in a hard NOC [14]. An ARM processor is used as the boot module, and it uses optimised MMIO read and write transactions to configure the NIs. (Routers are not programmed.) The boot time of the ARM processor is not taken into account here.

3) Soft NOC: We consider a conventional configuration interconnect with a soft NOC. The soft NOC requires 8300 LUTs per router-NI pair (Table 1). Therefore, at least $\frac{\text{8300 LUTs} \times \text{22 frames per column}}{\text{8 LUTs per CLB} \times \text{16 CLBs per column}} = 1427$ frames are required to configure a single router-NI pair. This is equivalent to a bitstream of 1427 frames × 41 words × $32b = 1.8Mb$. It takes at least 1427 frames × $0.7\mu s = 998\mu s$ configure the router-NI pair. However, the soft NOC is distributed over the FPGA and is likely to occupy more frames. Moreover, the soft NOC frames must be disjoint from CFR frames, otherwise they cannot be reconfigured independently. Both increase the NOC configuration time. Therefore, both the configuration time and bitstream size are optimistic estimates. The time to program a connection in the soft NOC for the functional data communication can be derived by $2.5\mu s \times 500/118 = 10.6\mu s$ (converting the hard NOC frequency to the slower soft NOC speed).

4) Hard NOC: The second scenario uses a hard NOC to configure the FPGA. First, the hard NOC must be programmed. For each NI, two connections must be programmed (one to program the NI, and one to configure the CFR), or $2 \times 2.5 = 5\mu s$ per NI. Second, the bitstreams must be loaded in the CFRs. The NOC transports the configuration data at, say, $8Gb/s$ (conservative conversion from $16Gb/s$ raw bandwidth to nett bandwidth), and is then a factor four faster than the conventional FPGA at $1.9Gb/s$.

5) Overall: Recall Figures 4 and 5 where the phases of booting a system were depicted. For each of the soft and hard NOC Table 2 the time spent on the following phases: programming the configuration interconnect, configuring the functional interconnect, programming the functional interconnect, and configuring the CFRs. A hard NOC requires programming per NI, whereas the soft NOC requires configuration per router-NI. Since programming requires fewer bits and is faster too, a system with a hard NOC is ready for functional operation $998\mu s/5\mu s = 200$ times faster. The gain of $(10.6 - 2.5)\mu s$ to program each functional connection does not significantly improve this number. The configuration footprint of the hard NOC is also smaller. The footprint size for a single soft router-NI pair is $1.8Mb$, whereas only $8.4Kb$ is required for the hard NI with 4 connections (routers are not programmed) [14].

Table 2. Configuration and Programming

phase	soft NOC	hard NOC	Fig.4	Fig.5
configuration mode				
conf. NOC	$998\mu s$/NI	-	1	-
prog. conf. interc.	-	$5\mu s$/NI	-	2
conf. CFRs	$1.9Gb/s$	$8Gb/s$	1	3
functional mode				
prog. NOC & CFR	$10.6\mu s$/conn.	$2.5\mu s$/conn.	2-3	4

For configuration regions equal to those of a current Virtex-4 device (16 CLB columns of 128 LUTs each, i.e. 2048 LUTs), a hard NOC has a 7% area overhead versus 405% for a soft NOC. Note that our analyses are independent from the particular NOC because they essentially depend on three factors: the ASIC:FPGA area ratio, the ASIC:FPGA operating frequency ratio, and the configuration:programming footprint (bitstream versus number of MMIO bits) ratio. For example, using a GS-only instead of BE+GS router [12] reduces the area cost of both soft and hard NOC by a factor four and increases their speed by a factor two, but the ASIC:FPGA area and speed ratios remain the same.

5. Conclusions

We proposed to replace the current FPGA configuration interconnect by a hard network on chip (NOC) and to use it also as the functional interconnect. The configuration and functional regions (CFR) are independent, as in current interconnects, but both connect to the hard NOC. The proposed architecture reduces timing closure problems in the functional interconnect because it is hardwired and preverified. A hard NOC is disjoint from soft IPs, eliminating an interference during dynamic partial reconfiguration. For example, this reduces the placement and layout of restrictions of soft IPs, which otherwise have to be e.g. in separate CLBs from a soft NOC.

A number of novel applications are possible in the new architecture. First, the hard NOC can be extended with soft network components (routers, NIs) implemented in the functional regions, by adding a simple bridge. This allows dynamic addition of soft routers and NIs. Second, bitstreams can be generated and/or modified at run time by IPs. For example, they can be optimised at run time, encrypted, etc. Bitstreams can also be moved from one configuration region to another, allowing, for example, dynamic relocation of IPs and embedded built-in self test.

We compared a conventional FPGA and soft NOC with our proposed architecture on area, functional speed, configuration footprint and speed, and general flexibility. 10% area cost is achieved by a soft NOC only for large CFRs ($80,000$ LUTs), but already for small CFRs ($\geq$ 1400 LUTs), allowing a maximum of 109 such CFRs on a Virtex-4. The hard NOC has a bandwidth:area performance advantage of a factor 150 or more over a soft NOC. Because a hard NOC operates at a higher speed, and is programmed rather than configured like a soft NOC, the configuration of the FPGA is 200 times faster when using a hard NOC, and its configuration footprint of the hard NOC is also smaller than that of a soft NOC. This suggests that our unified hard NOC outperforms the conventional configuration interconnect and soft NOC, at acceptable cost.

References

[1] A. M. Amory, et al. Wrapper design for the reuse of a bus, network-on-chip, or ther functional interconnect as test access mechanism. *IET Comp. & Dig. Tech.*, 1(3), 2007.

[2] A. M. Amory, et al. Software-based test for non-programmable cores in bus-based system-on-chip architectures. In *VLSI-SoC*, 2003.

[3] ARM. *AMBA AXI Protocol Specification*, June 2003.

[4] T. Bjerregaard. *The MANGO clockless network-on-chip: Concepts and implementation*. PhD thesis, DTU, 2006.

[5] C. Bobda, et al. The Erlangen Slot Machine: Increasing Flexibility in FPGA-based Reconfigurable Platforms. In *FPT*, 2005.

[6] G. Brebner and D. Levi. Networking on chip with platform FPGAs. In *FPT*, 2003.

[7] C. Chang, et al. BEE2: a high-end reconfigurable computing system. *IEEE Design & Test of Computers*, 22(2), 2005.

[8] I. Cidon, et al. Network and transport layers in networks on chip. In G. De Micheli and L. Benini, editors, *Networks on Chips: Technology and Tools*. Morgan Kaufmann, 2006.

[9] M. Coenen, et al. A buffer-sizing algorithm for networks on chip using TDMA and credit-based end-to-end flow control. In *CODES+ISSS*, 2006.

[10] J.-P. Diguet, et al. NOC-centric security of reconfigurable SoC. In *NOCS*, 2007.

[11] R. Gindin, et al. NoC-Based FPGA: Architecture and Routing. In *NOCS*, 2007.

[12] K. Goossens, et al. The Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Design and Test of Computers*, 22(5):414–421, Sept-Oct 2005.

[13] A. Hansson, et al. Undisrupted quality-of-service during reconfiguration of multiple applications in networks on chip. In *DATE*, 2007.

[14] A. Hansson, et al. Trade-offs in the configuration of a network on chip for multiple use-cases. In *NOCS*, 2007.

[15] A. Hansson, et al. A unified approach to mapping and routing on a network on chip for both best-effort and guaranteed service traffic. *VLSI Design*, 2007. Hindawi Publishing Corp.

[16] R. Hecht, et al. Dynamic Reconfiguration with hardwired Networks-on-Chip on future FPGAs. In *FPL*, 2005.

[17] R. Hecht, et al. Network-on-chip basierende laufzeitsysteme für dynamisch rekonfigurierbare hardware. In *Dynamisch Rekonfigurierbare Systemen at Conf. on Arch. of Comp. Sys.*, Mar. 2004.

[18] M. Huebner, et al. Scalable Application-Dependent Network on Chip Adaptivity for Dynamical Reconfigurable Real-Time Systems. In *FPL*, 2004.

[19] J. Hur, et al. Partially reconfigurable point-to-point interconnects in Virtex-II Pro FPGAs. In *ARC*, 2007.

[20] N. Kapre, et al. Packet switched vs time multiplexed FPGA overlay networks. In *FCCM*, 2006.

[21] I. Kuon, et al. Measuring the gap between FPGAs and ASICs. *IEEE Trans. on CAD of Int. Circ. and Sys.*, 26(2):203–215, Feb. 2007.

[22] T. Marescaux, et al. Networks on chip as hardware components of an OS for reconfigurable systems. In *FPL*, 2003.

[23] M. Millberg, et al. Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In *DATE*, 2004.

[24] H. Nikolov, et al. Efficient automated synthesis, programming, and implementation of multi-processor platforms on FPGA chips. In *FPL*, 2006.

[25] OCP Int. Partnership. Open core protocol specification, 2001.

[26] A. Patel, et al. A scalable FPGA-based multiprocessor. In *FCCM*, 2006.

[27] Philips Semiconductors. *Device Transaction Level (DTL) Protocol Specification. Version 2.2*, July 2002.

[28] T. Pionteck, et al. Applying partial reconfiguration to networks-on-chips. In *FPL*, 2006.

[29] A. Rădulescu, et al. An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network programming. *IEEE Trans. on CAD of Int. Circ. and Sys.*, 24(1):4–17, Jan. 2005.

[30] P. Sedcole, et al. Modular dynamic reconfiguration in Virtex FPGAs. *IEE Proc. Comp. and Dig. Tech.*, 153(3), 2006.

[31] M. Sgroi, et al. Addressing the system-on-a-chip interconnect woes through communication-based design. In *DAC*, 2001.

[32] F. Steenhof, et al. Networks on chips for high-end consumer-electronics TV system architectures. In *DATE*, 2006.

[33] A. Thompson. Silicon evolution. In *Proc. of Genetic Programming*, pages 444–452. MIT Press, 1996.

[34] S. Wee, et al. A Practical FPGA based Framework for Novel CMP Research. In *FPGA*, 2007.

[35] P. Wielage, et al. Design and DFT of a high-speed area-efficient embedded asynchronous FIFO. In *DATE*, 2007.

[36] Xilinx. Virtex-4 Configuration Guide, UG071 (v1.8), 2007.

[37] P. S. Zuchowski, et al. A hybrid ASIC and FPGA architecture. In *ICCAD*, 2002.