# 3-Tier Reconfiguration Model For FPGAs Using Hardwired Network on Chip

Muhammad Aqeel Wahlah[1] and Kees Goossens[1,2]

[1] *Computer Engineering, Delft University of Technology, Delft, The Netherlands*
[1] aqeel@ce.et.tudelft.nl

[2] *Research, NXP Semiconductors, Eindhoven, The Netherlands*
[2] kees.goossens@nxp.com

***Abstract*—We envision that future Field-Programmable Gate Arrays (FPGAs) will use a Hardwired Network on Chip (HWNoC) as a unified interconnect for functional communications (data and control) as well as configuration (bitstream for soft IPs). In this paper we present a 3-tier reconfiguration model that uses the HWNoC as the underlying platform to realize dynamic loading, starting, and stopping of applications. The model ensures that applications are guaranteed their required resources (LUTs, communication, memory). Resource allocation is performed globally at design time. Applications are started and stopped dynamically at run time, yet are composable, i.e. do not affect each other when they do so. Our model comprises three layers: system manager, application manager, and application. The system manager instantiates (configures) and enforces the resource allocation (LUTs, NoC connections, memories) at run time. Each application is independent, and is accompanied by an application manager that programs (starts and stops) the application, within its allocated resources (a virtual platform). We model our system in cycle-accurate transaction-level SystemC which includes bitstream loading, HWNoC and IP programming, clocking, reset, computation.**

## I. Introduction

Today's FPGAs [1], [2] are used to implement multi-processor system on chips (MPSoCs) that can run complex use-cases (sets of applications). The applications often have real-time requirements (e.g. audio, video, wireless standards), and can be started and stopped independently (e.g. on user command). As applications are often developed by different companies, it is desirable that they can be designed and tested independently. Composability, or the absence of interference, is required for this [3]. An application is implemented by multiple IPs, such as processors, memories, and accelerators. Real-time behaviour of an application can be ensured by allocating and enforcing sufficient resources [4]. Usually not all applications run at the same time, nor do they all fit on the FPGA simultaneously. Hence dynamic loading of applications is required. In this paper we address this need by defining a reconfiguration model for FPGAs that contain hardwired NoCs, as described in [5], such that applications can be safely loaded at run time, without affecting already running applications.

Before continuing we define our terminology. A *soft* IP is mapped on FPGA reconfigurable computational blocks (LUTs) whereas an IP is hardwired or *hard* when it is directly implemented in silicon, e.g. Power PC. We define *(re)configuration* as the loading of new soft IPs in the FPGA by sending a bitstream to a reconfiguration region. An IP is *programmed* after it is configured, if necessary, which entails changing the state of its registers when it is in functional mode. A *use-case* is defined as a set of concurrently executing applications.

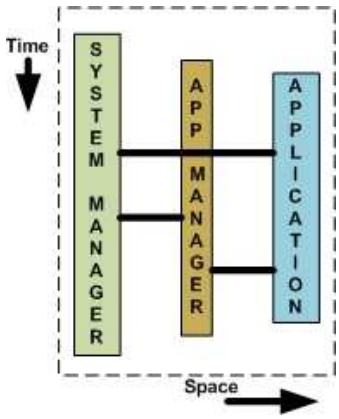


Fig. 1. Three Tiers Of Our Reconfiguration Model.

Several works offer dynamic reconfiguration without disturbing the existing applications [6], [7], [8], [9]. However, these use centralized resource managers, which may incur scalability problems. In contrast our reconfiguration model uses three tiers: system manager, application manager, and application. At run time, the *system manager* checks, configures, and enforces the resources globally allocated (at design time) to an application. Resources include the LUTs and BRAMs used to implement the soft IPs of the applications, connections on the HWNoC that are used for communication between the soft IPs, and the memories used by the application. While there is only one persistent system manager, each application is accompanied by its own *application manager* that exist only for the duration of the application. It programs, the application, and manages the application's IO and storage. The application itself is unaware of these aspects and focuses on computation, with IPs consuming and producing data on FIFOs. However, application IO and large FIFOs are implemented by RAMs with DMAs and address generators that are programmed by

the application manager.

Figure 1 illustrates the time-space relation as well as abstracted interaction among the three tiers of our reconfiguration model. It shows how the system manager deals with global resource configuration (LUTs, NoC connections, memories), essentially instantiating a virtual platform for each application. Within their virtual platform, application managers deal with the starting (programming) and stopping of their applications, and also with DMAs, address generators, and IOs. The application focuses on the steady-state computation.

The remainder of this paper is structured as follows. Section II positions our HWNoC reconfiguration model with respect to related work. Section III describes our HWNoC reconfiguration model for realizing dynamic application swapping. Section IV shows a worked example, and we conclude in Section V.

## II. Related Work

Since, our reconfiguration model provides the run-time application swapping on an FPGA. We therefore discuss the researches which include methods and systems to deal with the dynamics of application swapping on an FPGA.

In [10], [11] the authors use two methods for dynamic placement of modules. One is based on 1-D placement of modules in vertical slots, which are connected using a reconfigurable multiple bus (RMB). For the second method the authors perform 2-D placement of modules with a NoC as underlying functional architecture where the dynamically inserted modules overwrite the areas occupied by network elements i.e. routers. [12] provides a way to place hardware modules of predetermined size and positions, above each other and use bus macros for connecting the signals to static region.

In [9] the authors achieve dynamic on-demand reconfiguration by making use of a run-time system software on MicroBlaze for controlling reconfiguration and message handling whereas the work in [8] presents a hardware framework for run-time reconfiguration with tightly coupled general purpose CPU. [13] at run time allocates FPGA resources by using a centralized resource manager which can preload FPGA configurations by utilizing its knowledge of application flow graph. [6] uses a reconfigurable system based on square-shaped and arbitrary-sized *swappable logic units* (SLUs) which are arranged in mesh and communicate with each other through a small communication buffer. A dynamic instruction set architecture based approach is used in [14] where authors make use of dynamically rotating instructions for runtime swapping of reconfigurable modules.

In contrast to our layered approach these approaches are either central resource manager based [10], [6], [13], [9], [14] or consider a single application in the system [14], [12] in addition to not taking into account its Quality of Service (QoS) demands. Additionally, the above approaches face certain limitations which are not an issue with our model e.g.: a) possibility of execution time being dominated by the reconfiguration time due to the small communication buffer [6], is avoided by an application manager which ensures the memory allocation for its client application till the required execution stage; b) high area ratio of processing to network element [10], is avoided by hard-wiring the underlying functional architecture; c) the possibility of run-time reconfiguration of the functional architecture [7] which in turn could disrupt the executing applications, is avoided because our functional architecture only needs to be programmed with each adding IP/application; d) no QoS guarantees for the dynamically adding application [14], [13], [9], is avoided by providing a virtual platform for each (sub)application and reserving the required resources across that platform.

Notably, the proposed 3-tier reconfiguration model can be realized with ICAP instead of HWNoC for bitstream loading and in the presence of a soft functional architecture e.g. soft NoC, which requires to be programmed instead of (re)configured with the addition/removal of IPs/applications.

## III. HWNoC Reconfiguration Model

Our work is based on the HWNoC introduced in [5], which we briefly recapitulate here. By hard-wiring an interconnect, such as a NoC, on an FPGA [5], [15], [16], the performance:cost improvement can be as high as a factor 150, at the cost of some loss of flexibility (fewer LUTs) [5]. The connections for functional data between IPs and control data of the IPs have a higher bandwidth and lower latency. By replacing the ICAP with the HWNoC too, i.e. sending bitstreams over the same HWNoC, "normal functional data" and bitstreams can be easily converted to each other. This is useful for bitstream en/decryption etc. We use the Æthereal NoC [17] as our HWNoC, to be able to offer different bandwidth and latency guarantees to different connections. It also offers composability, i.e. the communications of different applications do not affect each other. This holds even applications they (that) are started or stopped, i.e. when the system is being (partially) reconfigured.

In our set-up a boot processor manages the HWNoC, to ensure system integrity (the HWNoC is a critical shared resource). It opens connections from external memory to load bitstreams in the configuration functional regions (CFRs) In previous work [18], the boot manager also managed the programming of IPs and DMAs, and address generators. In this paper, the various functions previously all performed by the boot processor have been more cleanly separated in the application manager (system resource management and configuration) and the application manager (application programming and memory management). We describe how applications are started and stopped in more detail below.

### A. Three-Tiered Structure Overview

Figure 2(A) shows the interaction among the 3-tiers where the system manager initially loads application IPs and programs the data connections. Using memory-mapped IO, the application manager is then programmed with application parameters, such as address ranges for the direct memory

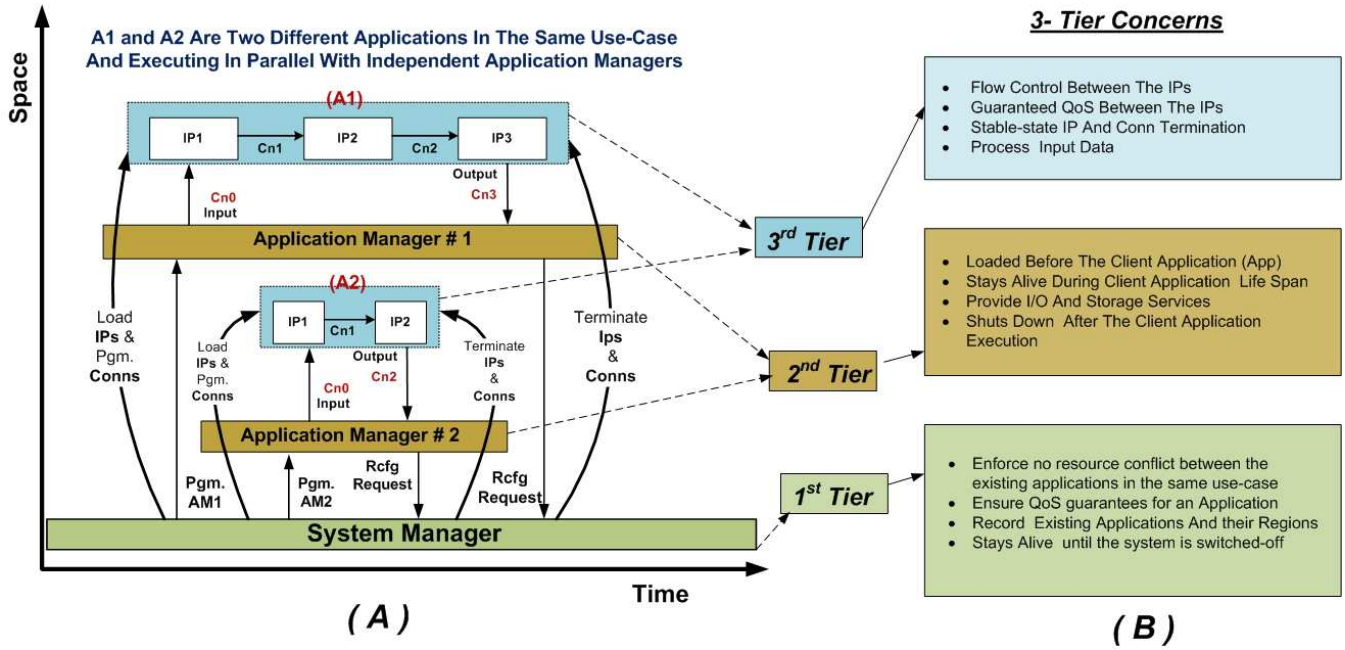


Fig. 2. 3-Tier HWNoC Reconfiguration Model Showing (A): *Interaction* (B): *Concerns* i.e. *Issues* And *Responsibilities* Across The 3-Tiers

access (DMA) engine and address generation unit (AGU). The application manager then programs the IPs with appropriate parameters, again using memory-mapped IO. The application receives its streaming data from the DMA engine, and executes, producing data that is placed by the DMA on the locations pointed by the AGU, as illustrated in Figure 3. The application manager observes the progress of the application. When all input data has been consumed, and all output data has been produced (which be substantially later, due to application pipelining), the application manager stops sending/receiving data to/from the application and signals reconfiguration request to the system manager.

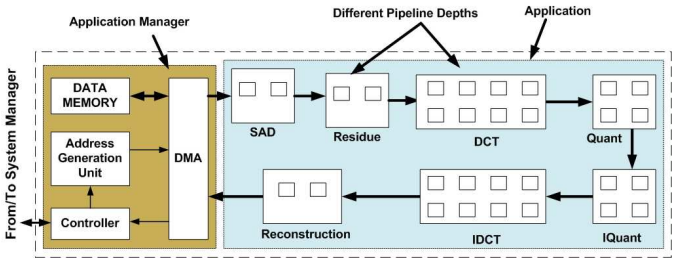


Fig. 3. Application And Its Manager Interaction

### B. Three-Tier Concerns

The interactions between the three tiers, and their respective responsibilities are illustrated in Figure 2(B) and are explained below.

*1) Application Concerns:* A real-time application which comprises soft IPs forms the third tier of our reconfiguration model. The application IPs operate i.e. consume and produce, address-less data which is pipelined and therefore is distributed in spatial and temporal domains. The streaming data at the start and end of the pipeline is read from and written to the FIFOs. However, large FIFOs and IO to the system are implemented by RAMs that require addressing. An IP communicating to its peers requires sufficient bandwidth pipe on the connecting channels (dedicated or shared) to meet its real-time throughput guarantees. Peer IPs as shown with Figure 3, could have different pipeline depths and clock frequencies therefore a flow-control mechanism is required to avoid loss of data. These soft IPs are configured before being initialized, programmed and executed. In addition, candidate IP cores could have different sizes, shapes, and number of functional ports.

*2) Application Manager Concerns:* An application manager per application is configured and programmed before loading the bitstream for its client application(s). The application manager performs resource management i.e. pumping/storing RAM data from/to IPs etc, and it must be placed close to its client application. Since the system manager is unaware of the application execution status the application manager notifies it about the application termination. An application manager must be stopped if the application no longer exists. It can also re-programmed by the system manager whenever new application has been instantiated.

*3) System Manager Concerns:* The system manager is executed on the boot processor and executes until the system is switched off. Its responsibilities include keeping a record of information for all the system applications, such as: application identifier, application manager, configuration region, use-cases etc. The system manager also keeps traces of QoS information (obtained at compile time) for all the system applications which include: reservation of the path for application IPs to communicate, number and positioning

of time slots across the path, credit counter between the source and destination application IPs across the network. Importantly, the system manager ensures that there no conflicts between applications. Hence it takes control of application-specific actions, such as bitstream transportation for the application cores, their initialization, stoppage, allocation and deallocation of NoC resources.

We next explain the procedure to reconfigure an application using 3-tier reconfiguration model.

### C. Application Reconfiguration Procedure

In this section we describe the reconfiguration procedure in more detail. Initially, the system manager loops over all the applications (client plus managers) in the active use-case, and loads them into their destination configuration regions. To ensure the availability of application I/O and storage resources, it makes sure that the applications manager is started before the application on the reconfigurable fabric.

To configure an application the system manager uses the application information as detailed in section III-B3. The application configuration plus initialization is carried out by looping over all of the application configuration functional regions [5] as shown in Figure 4. 1) The system manager first establishes a guaranteed-latency connection to each CFR for loading the bitstream. 2) It fetches the IP bitstreams from the configuration memory to be placed in that specific configuration region. 3) The system manager closes the bitstream connection once the all the IPs in that configuration region are configured. 4) Afterwards on a separate connection it carries out the initialization per IP, which includes sending clock and reset information to memory-mapped clock and reset generators. 5) It is followed by programming the connections among the application IPs with QoS information obtained at compile time. 6) Once the application connections are in position the system manager programs the respective application manager with its client application's input/output base addresses, strides, data ranges and number of executions to perform.

Each destination configuration functional region in our FPGA [5] has an appropriate circuitry i.e. address decoder and registers, to parse the incoming bitstream from the system manager and to place it at the correct locations [18]. The CFR also comprises a single clock generator, which restricts all the candidate IPs to placed in a configuration region to run at same frequency. In addition there is a reset generator from which 32 1-bit reset links comes out which allows a maximum of 32 IPs to be placed in a single configuration functional region.

An application manager comprises a controller, DMA engine, AGU and embedded on-chip memory as shown in Figure 3 and which provides storage/IO services to the third tier of our reconfiguration model as explained in Figure 4. On an application manager part the application manager *controller* after receiving the client application parameters from the system manager, calculates the required input/output addresses by using the embedded address generation unit. The DMA engine afterwards fetches the data from those RAM addresses and forwards it to the required IP on an appropriate data connection, as previously established by the system manager. The DMA interacts with the executing application both in the forward (producing application input) and the backward directions (consuming application output) and each time after pumping/storing the RAM data it contacts the *application manager controller* which on the basis of the received information from the system manager decides to continue or terminate the application execution.

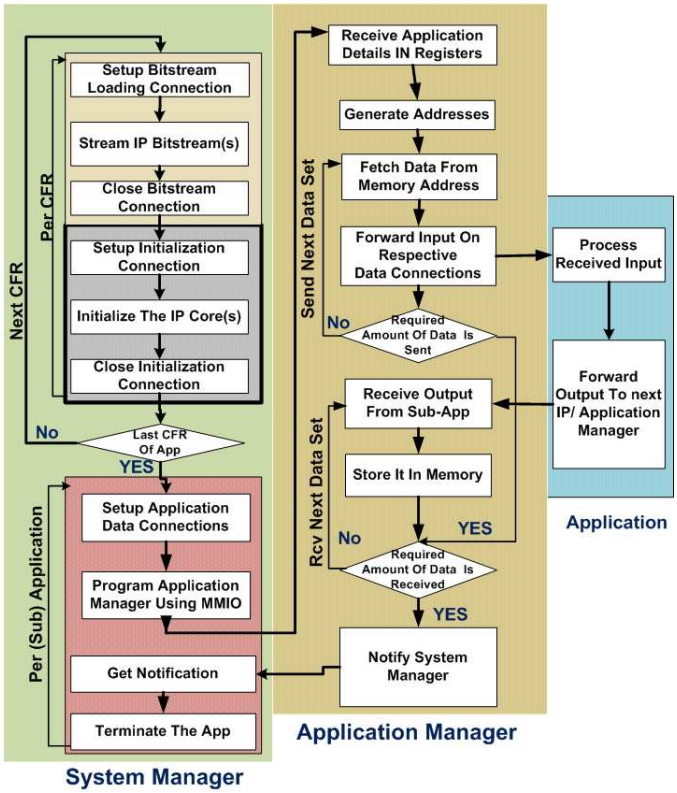


Fig. 4. Application Reconfiguration

## IV. Experiments And Results

We modeled the HWNoC reconfiguration model in SystemC using the design flow of [19]. We use a simplified H.264 application with behavioral models of the three IPs as shown in Figure 5 to encode Quarter Common Intermediate Format (QCIF) resolution video frames. Synthesis of the VHDL implementations of the Residue and DCT IPs on a Virtex-4 XC4VLX200 chip using Xilinx ISE 8.2 provided their frequencies which were used in SystemC, table I.

The size of their bitstreams was estimated from their $k$LUT areas using the equation *(IP LUTs * frames per column) / (LUTs per CLB * CLB per column)*. For Virtex-4 [20] a single CLB contains 8 LUTs, and a column contains 22 frames and 16 CLBs. The Quantizer area and frequency were estimated to be between the DCT and the Residue respective values. The NoC contains 4 routers and network interface kernels with respective FIFO sizes of 24 and 41x words.

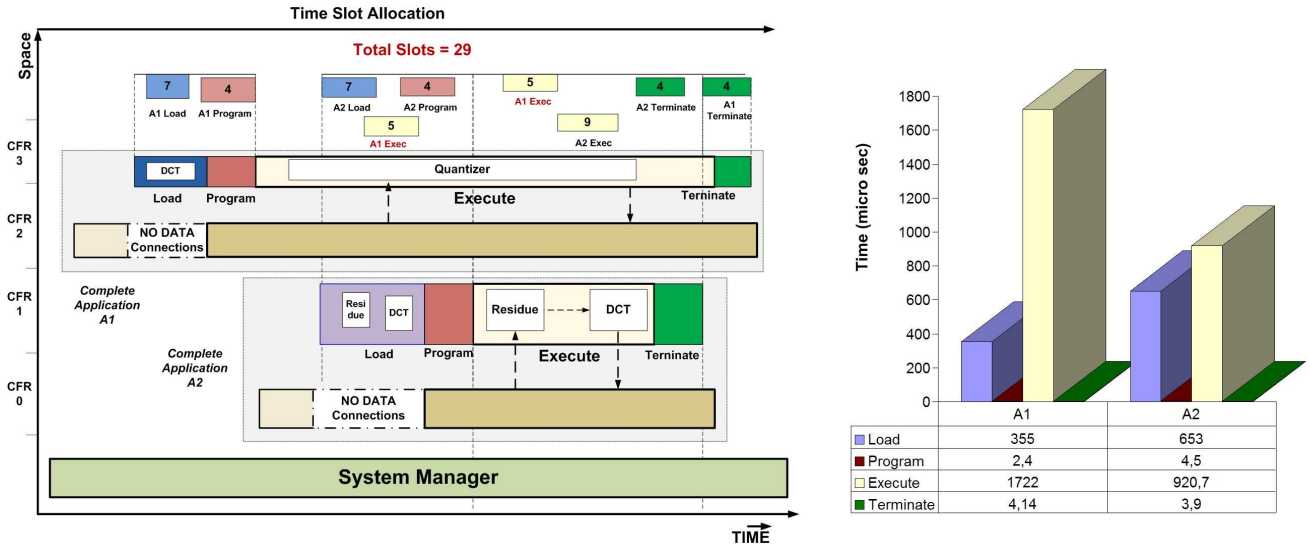


Fig. 5. Application *Temporal Analysis* With *System Resource Reservation* At Different Stages

TABLE I
APPLICATION IP SYNTHESIZED AREA, FREQUENCY AND RECONFIGURATION TIME

| IP | Area (*kLUTs*) | Frequency (*MHz*) | Bitstream (*Frames*) | (Re)config Time ($\mu s$) |
|---|---|---|---|---|
| RESIDUE | **1.68** | **100** | **285** | **273.6** |
| DCT | **2.36** | **66** | **396** | **380.16** |
| Quantizer | **2.21** | **75** | **370** | **355.2** |

We provide the analysis and proof of the concept of virtual platforms (i.e one applications does not disrupt the execution of the other) for each application by using the HWNoC and carrying out their structural and functional mapping onto the required FPGA regions.

### A. Application Timing Analysis

We considered two applications A1 (comprising Quantizer) and A2 (comprising Residue + DCT) with independent application managers. These applications belong to the same use-case and and execute in parallel without affecting each other, Figure 5. The Figure 5 shows the timings required to load, program, execute and terminate both the applications along with the corresponding allocated NoC resources during the each phase. It also illustrates the different CFRs occupied by the applications and their managers. In the following discussion we will expound temporal details for the A2 during the different phases.

Each application phase is preceded by programming the NoC, as illustrated in [21], in $0.18\,\mu s$ to $0.24\,\mu s$, so that the data can reach the required location. This time has been included in the preceding discussions while mentioning individual phase delay. Notably, the source and the destination in our network at the maximum can be three nodes apart and with each extra router an additional delay of $0.006\,\mu s$ is encountered.

Loading of A2 starts with streaming the bitstream by the system manager on a fixed and low latency connection. It takes $653\,\mu s$ to load the 681 frames of the A2 in the destination CFR.

Programming of the A2 starts afterwards by the system manager. The system manager first initializes its IPs in $0.63\,\mu s$ by programming memory mapped reset and clock generator in the destination configuration region. It is followed by setting-up of three data connections for those IPs in approximately $2.9\,\mu s$ which corresponds to write path, credits and slots information to the required network interface kernels which provide/schedule the communication of the attached IP blocks. As the last step before the A2 execution the respective application manager is programmed by the system manager in $0.97\,\mu s$ with 48-words application parameters which comprise full application's I/O addresses and ranges.

Execution of A2 is carried out afterwards where the single execution of its IPs process one 4x4 pixel-block, and 16 such pixel blocks constitute single Macro Block (MB). It takes $920.7\,\mu s$ to process 2QCIF (198MBs) video frame in peer to peer streaming communication fashion.

Termination of the Application A2 is triggered once the desired execution is achieved. It initiates with an application manager notification to the system manager in $0.24\,\mu s$ on an already established connection. Afterwards the system manager shuts down the three A2 connections in $2.9\,\mu s$ time which accounts for blocking the source shell from emitting new transactions, emptying input/output queues, and clearing the respective slot table entries. Disabling the application IPs comes next which starts with opening a reset connection to network interface kernel(s) associated with application IPs and afterwards sending a 32-bit reset signal to disable IPs from processing further. It takes $2.7\,\mu s$ to shut down the A2 IPs which are placed in the same CFR.

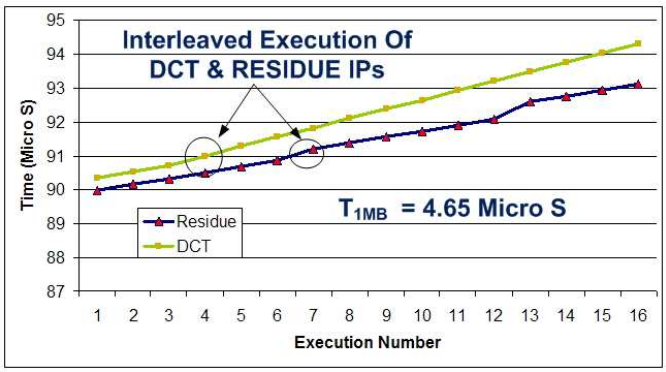


Fig. 6. Application A2 IPs In Pipelined Execution Mode

### B. Application Resource Allocation

The Figure 5 shows that each application phase is allocated with the specific quota of resources i.e. time slots. The computation of the time slots is performed at compile time according to the design time specification and their allocation is ensured at run time by the system manager. Allocating dedicated system resources for each phase, as illustrated in Figure 5, allows both the applications to coexist and that as well in different phases. It is evident from the Figure 5 that $24.15\%$ of the total system resources are reserved for $A1$ and $A2$ during the load phase whereas $13.8\%$ each for their programming and termination phases. However, both the $A1$ and $A2$ have different QoS requirements during the execution phase and which respectively corresponds to $17.2\%$ and $31\%$ of the total system resources.

In the example, the NoC has 29 time slots. This illustrates that time slots are re-used between different phases (use cases) because the total number of slots of the *load, program, execute and terminate* phases is larger (43).

## V. Conclusion

In this paper, we presented a three-tier reconfiguration model of HWNoC which provides the dynamic run-time reconfiguration of an application. We presented the mechanism to achieve above objectives and modeled the application dynamic behavior in cycle-accurate transaction-level SystemC. The two-tiers (the system manage and an application manager) of our reconfiguration model were responsible for managing services across the system. The system manager at runt time, instantiated and enforced the resource allocation both in communication (NoC) and logic (FPGA) planes. This resource allocation corresponds to NoC connections, LUTs and memories. On the other hand an application manager programmed (started and stopped) the client application. Additionally, the application manager by making use of its embedded AGU and DMA engine provided computational data and memory allocation for the client application.

## References


[1] Xilinx Inc., "Virtex-4 Data Sheets," 2005. [Online]. Available: http://www.xilinx.com

[2] ——, "Virtex-5 Data Sheets," 2006. [Online]. Available: http://www.xilinx.com

[3] J. Kramer and J. Magee, "The evolving philosophers problem: Dynamic change management," *IEEE Transactions on Software Engineering*, vol. 16, 1990.

[4] A. Hansson, K. Goossens, and A. Rădulescu, "A unified approach to constrained mapping and routing on network-on-chip architectures," in *Int'l Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Sep. 2005.

[5] K. Goossens, M. Bennebroek, J. Y. Hur, and M. A. Wahlah, "Hardwired networks on chip in FPGAs to unify data and configuration interconnects," in *Proc. Int'l Symposium on Networks on Chip (NOCS)*, Apr. 2008.

[6] G. Brebner, "The Swappable Logic Unit: a Paradigm for Virtual Hardware," in *Proc. Int'l Conference on Field-Programmable Custom Computing Machines (FCCM)*, Apr 1997.

[7] M. Huebner, C. Schuck, M. Kiihnle, and J. Becker, "New 2-dimensional partial dynamic reconfiguration techniques for real-time adaptive micro-electronic circuits," in *Emerging VLSI Technologies and Architectures*, Mar 2006.

[8] M. L. Silva and J. C. Ferreira, "Support for partial run-time reconfiguration of platform FPGAs," *Journal of Systems Architecture: the EUROMICRO Journal*, vol. 52, Dec. 2006.

[9] M. Ullmann, M. Huebner, B. Grimm, and J. Becker, "An FPGA run-time system for dynamical on-demand reconfiguration," in *Proc. Int'l Parallel and Distributed Processing Symposium (IPDPS)*, Apr 2004.

[10] C. Bobda and A. Ahmadinia, "Dynamic interconnection of reconfigurable modules on reconfigurable devices," in *Proc. Int'l Conference on Design And Test Of Computer (DATC)*, vol. 22, Sep 2005.

[11] C. Bobda, A. Majer, A. Ahmadinia, T. Haller, A. Linarth, and J. Teich, "The Erlangen Slot Machine: Increasing Flexibility in FPGA-based Reconfigurable Platforms," in *Proc. Int'l Conference on Field-Programmable Technology (FPT)*, Dec. 2005.

[12] P. Sedcole, B. Blodget, T. Becker, J. Anderson, and P. Lysaght, "Modular dynamic reconfiguration in virtex FPGAs," *IEE Proceedings Computers and Digital Techniques*, vol. 153, May 2006.

[13] J. Jean, K. Tomko, V. Yavagal, R. Cook, and J. Shah, "Dynamic reconfiguration to support concurrent applications," in *IEEE Transactions on Computers*, vol. 48, June 1999.

[14] L. Bauer, M. Shafique, and J. Henkel, "Efficient resource utilization for an extensible processor through dynamic instruction set adaptation," in *Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 16, Oct. 2008.

[15] R. Hecht, S. Kubisch, A. Herrholtz, and D. Timmermann, "Dynamic Reconfiguration with hardwired Networks-on-Chip on future FPGAs," in *Proc. Int'l Conference on Field Programmable Logic, Reconfigurable Computing, and Applications (FPL)*, Aug. 2005.

[16] R. Gindin, I. Cidon, and I. Keidar, "NoC-Based FPGA: Architecture and Routing," in *Proc. Int'l Symposium on Networks on Chip (NOCS)*, May 2007.

[17] K. Goossens, J. Dielissen, and A. Rădulescu, "The Æthereal network on chip: Concepts, architectures, and implementations," *IEEE Design and Test of Computers*, vol. 22, no. 5, Sept-Oct 2005.

[18] M. A. Wahlah and K. Goossens, "Modeling reconfiguration in a FPGA with a hardwired network on chip," in *Proc. Reconfigurable Architecture Workshop (RAW)*, May 2009.

[19] K. Goossens, J. Dielissen, O. P. Gangwal, S. González Pestana, A. Rădulescu, and E. Rijpkema, "A design flow for application-specific networks on chip with guaranteed performance to accelerate SOC design and verification," in *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, Mar. 2005.

[20] Xilinx Inc., "Virtex-4 Configuration Guide." [Online]. Available: http://www.xilinx.com

[21] A. Hansson and K. Goossens, "Trade-offs in the configuration of a network on chip for multiple use-cases," in *Proc. Int'l Symposium on Networks on Chip (NOCS)*, May 2007.