

## Automated Partial Reconfiguration Design for Adaptive Systems with CoPR for Zynq

Kizheppatt Vipin, Suhaib A. Fahmy  
 School of Computer Engineering  
 Nanyang Technological University, Singapore  
 {vipin2,sfahmy}@ntu.edu.sg

**Abstract**—Dynamically adaptive systems (DAS) respond to environmental conditions, by modifying their processing at runtime and selecting alternative configurations of computation. Field programmable gate arrays, with their support for partial reconfiguration (PR) represent an ideal platform for implementing such systems. Designing partially reconfigurable systems has traditionally been a difficult task requiring FPGA expertise. This paper presents a fully automated framework for implementing PR based adaptive systems. The designer specifies a set of valid configurations containing instances of modules from a standard library. The tool automates partitioning of modules into regions, floorplanning regions on the FPGA fabric, and generation of bitstreams. A runtime system manages the loading of bitstreams automatically through API calls.

### I. INTRODUCTION

Dynamically adaptive systems (DAS) are systems that continuously monitor their environment and adapt their behaviour in response to changes in environmental conditions [1]. A DAS can be considered as a collection of different system operating modes, called *configurations*, of which only one is active at a given point in time. At runtime, changes in the operating environment can cause a DAS to switch its *configuration* to adapt. This switching operation, called *reconfiguration*, is controlled and managed by a *configuration manager* (CM) that monitors parameters and applies *adaptation algorithms*.

In this context, FPGA partial reconfiguration (PR) is a promising enabler. PR is the process by which a part of the FPGA configuration memory can be overwritten at runtime, thus modifying the behaviour of parts of the chip, while the remainder continues to function. This capability, formerly only offered for high-end FPGAs, is now supported across most FPGAs from Xilinx [2], and recently Altera has also begun offering it [3].

Building a partially reconfigurable system entails a number of design decisions. The first is a definition of regions on the device called partially reconfigurable regions (PRRs), for each of which, we must generate a set of valid bitstreams to be loaded at runtime. The second is the mechanism by which such bitstreams are stored and loaded at runtime. The final aspect is some control that manages the *reconfiguration* at runtime to meet overall application requirements.

Much PR design research has targeted the use of PR for time multiplexing a fixed set of tasks within a constrained reconfigurable area. In such cases, the management of the system at runtime is simple, and the sequence of reconfiguration can be determined in advance. In DASs, the adaptation mechanisms can be complex, and respond to unpredictable environmental conditions, and are part of what the application designer would develop. Although models have been proposed for mapping adaptive system descriptions to FPGAs, actual implementation of the resulting systems remains challenging [4], [5], [6], [7]. A fully automated flow that allows adaptive systems designers with minimal hardware expertise to map applications to a PR design has so far failed to materialise. We believe this is an essential step in PR achieving more widespread adoption, with applications that move beyond the small examples typically encountered in PR literature.

New hybrid configurable platforms such as the Xilinx Zynq offer new opportunities for DAS implementation as they tightly integrate processor cores with a reconfigurable fabric. The compute intensive data processing *configurations* can be implemented on the reconfigurable fabric while complex adaptation algorithms can be implemented in software, making them easily programmable. CoPR is the first fully automated flow for mapping high-level descriptions of adaptive systems to hybrid FPGAs. This framework can equally serve as the implementation basis for other techniques that require PR functionality.

### II. MAPPING DYNAMICALLY ADAPTIVE SYSTEMS

We now discuss the different aspects of a DAS and how they are mapped onto a hybrid-FPGA platform in the CoPR framework.

#### A. System Decomposition

The system level architecture for DAS we have chosen is depicted in Figure 1. The overall system is divided into two logical planes, namely the *control plane* and the *data plane*. The *configurations*, that complete data processing, reside within the data plane while the control plane monitors and regulates system state, managing *reconfiguration*. The data plane can support intensive computation by mapping

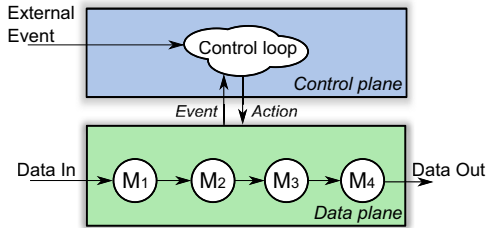


Figure 1. The control and data planes of a DAS.

it to the hardware fabric. Meanwhile, the control plane typically functions at a much lower data rate, but might use complex sequential algorithms, and is hence more suitable for software implementation.

A data plane configuration is composed of several functional units such as  $M_1, M_2, M_3$  and  $M_4$  interfaced with each other, as shown in Figure 1. We define the atomic functional unit as a *module*. Each module may have a set of parameters which determine its operating characteristics. These parameters can be modified at runtime to control the modules and thus the data plane behaviour.

The control plane implements the DAS *configuration manager* (CM). The CM monitors and regulates system state by implementing the *observe, decide, act* loop (control loop) [8]. The loop constantly monitors the system environment to detect changes in operating conditions called *events*. These events are analysed to decide whether changes in system state are required and how to reach the intended state through *actions*.

### B. Modelling Adaptation

A set of modules in the data plane which implements a mode of functionality is called a *configuration*. For example,  $\{M_1, M_2, M_3, M_4\}$  in Figure 1 comprise a system configuration. When the system adapts to a new operating state, the configuration changes by replacing one or more modules with new ones. This is called a *structural reconfiguration*.

In another scenario, modifications to the system operating characteristics are achieved by modifying one or more parameters of the modules without physically replacing them. This is a *parametric reconfiguration*. Ideally a system designer should be able to model both these types of reconfiguration in a way that suits the applications without worrying about how they are actually implemented.

### C. Zynq Architecture

New hybrid reconfigurable devices, such as the Xilinx Zynq are an ideal choice for implementing such systems as they include a powerful processor, standard communication infrastructure, and integrated reconfigurable fabric [9]. The Zynq tightly couples a dual-core ARM Cortex A9 processor

with a reconfigurable fabric. The ARM processor communicates with on-chip memory, memory controllers, and peripherals through AXI interconnect. Together, these hardened blocks constitute the Processor System (PS). The on-chip PS is attached to the Programmable Logic (PL) through multiple AXI ports, offering high bandwidth between the two key components of the architecture. The processor configuration access port (PCAP) in the PS supports full and partial configuration of the PL.

Optimising PR based designs requires a solid understanding of the underlying configurable fabric architecture. Like other Xilinx FPGAs, the Zynq configurable fabric (PL) is divided into rows and columns. A *tile* is one row high and one column wide, and contains a single type of resource such as CLBs or Block RAMs. The basic unit for defining a partially reconfigurable region (PRR) is a tile and a tile should not be shared between multiple regions. Present partial reconfiguration tool flows require the designer to know these low level architecture details for system implementation and optimisation. In the CoPR flow, these details are abstracted away from the designer’s point of view but are used internally by the tool to optimise implementation.

### D. Architecture Mapping

The DAS data plane is implemented on the Zynq PL and structural reconfiguration is achieved by configuring PRRs with appropriate *partial bitstreams*. The control plane is implemented as two logically separate software components called the *adaptation manager* and the *configuration manager* running on the ARM processor. The *adaptation manager* is software written by the system designer that implements the *control loop* discussed in Section II-A in an implementation-independent format. It communicates with the *configuration manager* through APIs provided by the CoPR framework. The *configuration manager* performs the implementation dependent structural and parametric reconfigurations by loading partial bitstreams or varying module register contents.

An important factor in data plane implementation is inter-module communication. We adopt an *AXI4-Stream* based interface for high throughput inter-module communication. IP cores from Xilinx as well as modules generated using high-level synthesis languages such as Vivado HLS support this interface.

For communication between the control and data planes, a lightweight interface is required for parametric reconfiguration by modifying module registers. *AXI4-Lite* is ideal for this and is supported between the Zynq PS and PL.

## III. TOOL FLOW

Figure 2 shows the CoPR tool flow. The flow includes both software and hardware, accepts user specifications, applies optimisation algorithms, and interfaces with vendor

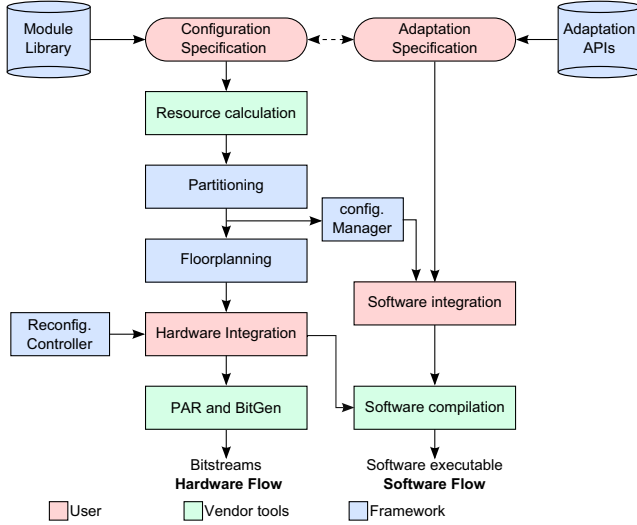


Figure 2. CoPR tool flow for PR based adaptive systems design, showing steps performed by the user, vendor tools, and framework.

tools through a set of custom scripts. The following sections describe each step in detail.

### A. Specification

The primary designer inputs to CoPR are the *configuration* and *adaptation* specifications. The configuration specification details the different valid DAS configurations and the corresponding library modules present in each configuration, and is described in *XML* format as shown in Figure 3. Each configuration has an associated *name* and the modules are listed in the order they appear in the processing chain.

The configuration specification also lists the possible parameter values for each module. Module parameters can be changed at runtime, and this can lead to a PR reconfiguration or the setting of a register value. The tool automatically analyses the parameter definitions and elaborates the configuration specification to include additional configurations resulting from parametric reconfiguration of the system.

The *adaptation specification* contains the software code for the *adaptation manager* implementation. This is written by the designer using the functions offered by the CoPR API, without reference to the implementation details, instead referencing the labels in the configuration specification.

CoPR first uses the vendor synthesis tool (XST) to synthesise all modules to determine resource requirements. Scripts extract the resource requirements from the synthesis reports and these numbers are later used for design partitioning and floorplanning.

### B. Partitioning and Interface Generation

Partitioning determines the number of reconfigurable regions (PRRs) and allocates modules to them. The way the system is partitioned greatly influences the resource

```

1 <configurations>
2   <config name="tx_chain">
3     <module name="encoder">
4       <parameter standard="enc1"/>
5     </module>
6     <module name="modulator">
7       <parameter standard="mod1"/>
8     </module>
9   </config>
10  <config name="rx_chain">
11    <module name="demodulator">
12      <parameter standard="dmod1"/>
13    </module>
14    <module name="decoder">
15      <parameter standard="dcod1"/>
16    </module>
17  </config>
18  ...
19 </configurations>

```

Figure 3. Configuration specification in *XML* format. Each configuration is specified by its name and the list of modules.

requirements and reconfiguration time. Our algorithm [10] partitions the design in such a way that the total reconfigurable area is minimised. System configurations play an important role here, as they greatly reduce the number of module combinations that must be considered to arrive at an efficient allocation.

The partitioning algorithm initially clusters modules based on how often they occur together in a configuration. Clusters of co-occurring modules are identified, and then compatible sets are found, where the modules in compatible clusters are not required in a single configuration. These compatible clusters can be swapped in a single region, ensuring that all configurations are covered by the compatible clusters assigned to all regions. The final output of partitioning is a list of PRRs and the corresponding partition allocation to them.

After partitioning, wrappers that instantiate the required modules are generated for each PRR, ensuring a unified interface across different configurations. A *pr\_system\_top* wrapper is also generated which instantiates and connects all the PRRs as *black boxes*. The generated wrapper module is in IP core format, which can be directly imported into Xilinx’s XPS tool flow.

### C. Floorplanning

Floorplanning involves determining the physical locations of the PRRs on the PL fabric. Regions must be rectangular in shape and not overlap. As discussed in Section II-C, the basic unit for floorplanning is a *tile*. Since each tile is one device row high, the height of PRRs is an integer multiple of device rows.

The resource requirements for each PRR are converted to the corresponding number of tiles with an added 10% margin to avoid routing congestion during implementation. Our approach, adapted from [11], uses resource *templates* that encode the horizontal arrangements of tiles on the target architecture, as shown in Figure 4. To satisfy the resource

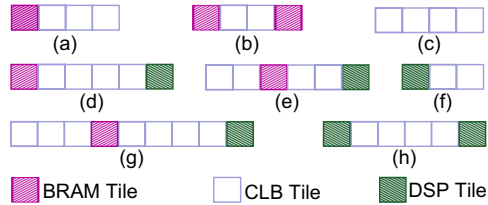


Figure 4. Resource templates, which are tiled vertically to satisfy PRR resource requirement during floorplanning.

requirements of a PRR, templates containing all the required types of tiles can be stretched vertically until the required size is achieved. The final output of floorplanning is a *user constraints file* (ucf) specifying the coordinates of the PRRs.

#### D. Hardware Integration

At this stage the designer can add the outputs of partitioning (pr\_system\_top wrapper) and floorplanning (ucf file) to a Zynq embedded project using the Xilinx XPS software. Although this step can be automated, user intervention offers flexibility to choose additional system peripherals.

The designer can choose to use either the PCAP or our higher performance ZyCAP controller [12] that triples reconfiguration speed at the cost of modest resource usage.

#### E. Place and Route and Bitstream Generation

The designer now runs the automation scripts which direct the vendor placement and routing tools, then the bitstream generation tool to generate all the necessary partial bitstreams and the full system bitstreams. The most resource intensive configuration is implemented first since the quality of static region routing depends upon the first configuration implemented. The partial bitstreams must then be copied to an SD card which is inserted in the board.

#### F. Software Implementation

As described in Section III-A, the adaptation specification is programmed by the user, referring to the configurations defined in the configuration specification. Presently it is written in C compatible with the Zynq ARM compiler. The framework automatically generates the configuration manager (CM) based on the *configuration specification* and the output of the partitioning step.

APIs are provided for determining the present configuration (*get\_config()*) and for changing configuration (*set\_config(configuration\_name)*). The designer does not need to know which partial bitstream corresponds to which configuration or where they are stored. The *configuration\_name* used in the API is the same as the one specified by the user in the configuration specification file. APIs are also provided for accessing hardware module parameters (*get\_param(module,parameter)*) and modifying them (*set\_param(module,parameter,value)*). Changing parameters

sometimes leads to the reconfiguration of modules, which is automatically handled by the CM.

#### IV. CONCLUSION AND FUTURE WORK

We have introduced the CoPR framework for the design of dynamically adaptive systems using partial reconfiguration. It accepts a high-level description of an adaptive system, automatically partitions the design into reconfigurable regions, determines a floorplan and generates bitstreams. Runtime adaptation control is also automatically generated, isolating the designer from the low-level aspects of the implementation.

Presently, the reconfiguration manager is implemented for the Stalalone operating system, and we are working to add support for Linux. The automation scripts are also being modified to work with the new Xilinx Vivado tool flow.

#### REFERENCES

- [1] B. H. Cheng, P. Sawyer, N. Bencomo, and J. Whittle, "A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty," in *Model Driven Engineering Languages and Systems*. Springer Berlin Heidelberg, 2009, vol. 5795, pp. 468–483.
- [2] *UG702: Partial Reconfiguration User Guide*, Xilinx, 2010.
- [3] M. Bourgeault, "Altera's partial reconfiguration flow," Altera, Tech. Rep., 2011.
- [4] T. Bapty, S. Neema, J. Scott, J. Sztipanovits, and S. Asaad, "Model-integrated tools for the design of dynamically reconfigurable systems," *VLSI Design*, vol. 10, no. 3, pp. 281–306, 2000.
- [5] M. Santambrogio, V. Rana, and D. Sciuto, "Operating system support for online partial dynamic reconfiguration management," in *Proc. of International Conf. on Field Programmable Logic and Applications (FPL)*, 2008, pp. 455–458.
- [6] J. Lotze, S. Fahmy, J. Noguera, and L. Doyle, "A model-based approach to cognitive radio design," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 29, no. 2, pp. 455–468, Feb. 2011.
- [7] J. Vidal, F. De Lamotte, G. Gogniat, J.-P. Diguët, and S. Guillet, "Dynamic applications on reconfigurable systems: from UML model design to FPGAs implementation," in *Proceedings of Design, Automation and Test in Europe (DATE)*, 2011.
- [8] J. Mitola and G. Q. Maguire, "Cognitive radio: making software radios more personal," *IEEE Personal Communications*, vol. 6, no. 4, pp. 13–18, 1999.
- [9] *UG585: Zynq-7000 All Programmable SoC Technical Reference Manual*, Xilinx, Mar. 2013.
- [10] K. Vipin and S. A. Fahmy, "Automated partitioning for partial reconfiguration design of adaptive systems," in *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW)*, Boston, MA, 2013, pp. 172–181.
- [11] K. Vipin and S. A. Fahmy, "Architecture-aware reconfiguration-centric floorplanning for partial reconfiguration," in *Proceedings of the International Symposium on Applied Reconfigurable Computing (ARC)*, 2012, pp. 13–25.
- [12] K. Vipin and S. A. Fahmy, "ZyCAP: Efficient partial reconfiguration management on the Xilinx Zynq," *IEEE Embedded Systems Letters (to appear)*, vol. 6, 2014.