Dynamic Partial Reconfigurable FPGA Framework For Agent Systems

Edward Chen[1] ,Victor Gusev[1], Dorian Sabaz[2], Lesley Shannon[1], and William A. Gruver[1,2]

[1] School of Engineering Science, Simon Fraser University
Burnaby, BC, Canada
{ekchen, vga9, lshannon, gruver}@sfu.ca

[2] Intelligent Robotics Corporation
North Vancouver, BC, Canada
{dorian, gruver}@iroboticscorp.com

Abstract. Dynamic Partial Reconfigurable (DPR) FPGAs enable software such as threads and agents to be executed directly in hardware. However, they were utilized as hardware extensions of software to execute individual threads or threads encapsulated in an agent. Thus, it was necessary for these systems to be administered by a CPU and did not take full advantage of the concurrency features that FPGAs provided. This paper presents a hardware framework in which the agent concept and the benefits that arise from an agent system are designed into the FPGA. This enables not only the hardware modules to be viewed as agents, but also provides a means to selectively design and componentize the communications network for the hardware agents. The proposed framework enables hardware agents to be implemented to run concurrently and allows them to communicate with each other without requiring a CPU.

Keywords: DPR, FPGA, Hardware Abstraction, Agent Systems

1 Introduction

A Field Programmable Gate Array (FPGA) provides designers with a programmable fabric that can be configured at runtime to execute the designed circuit, and allows designers to reconfigure the entire FPGA for a new application when the current application has been completed. FPGAs have been widely used in the fields of wired and wireless communication, [1] image and signal processing [2-3], medical equipment [4], robotics [5], automotive [6] and embedded control systems [7].

Dynamic Partial Reconfigurable (DPR) FPGAs allow multiple Hardware Modules (HMs) to spatially share a pre-defined portion of the programmable fabric while the remainder of the fabric stays active. Despite its benefits [9-10], the hardware implementation process for DPR continues to be complex and time consuming, and often requires hardware developers to have a thorough understanding of the underlying device and design methodology [8,12]. Also, since hardware design is inherently more complex than software, challenges such as concurrency and synchronicity between competing requests and the standardization of communication infrastructures must be addressed before hardware agents can be realized in an agent system.

This paper presents a DPR FPGA framework that utilizes a hardware representation of the agent system paradigm. The framework leverages software implementation benefits such as loose couplings between code modules, a flexible system framework, and the ability to facilitate the development of future system architectures. In addition, by implementing hardware agents in the partially reconfigurable regions of the FPGA, benefits such as higher throughput [11], robustness [7], and concurrent processing [10] can be achieved. With DPR, hardware agents may be moved within or between devices without the need to completely reconfigure the underlying application. Mission critical applications are able to stay active while pre-defined portions of the programmable fabric are partially reconfigured to accommodate the entries and exits of hardware agents.

2 Background

2.1 Field Programmable Gate Arrays

FPGAs are programmable semiconductor devices were first developed in the 1980s. They are based on a matrix of Configurable Logic Blocks (CLBs) connected via programmable interconnects [11]. Unlike Application Specific Integrated Circuits (ASICs) where the device is custom built for the specific design, FPGAs with static random access memory can be completely configured between applications to implement different circuits. Figure 1 shows the major components of a modern FPGA.

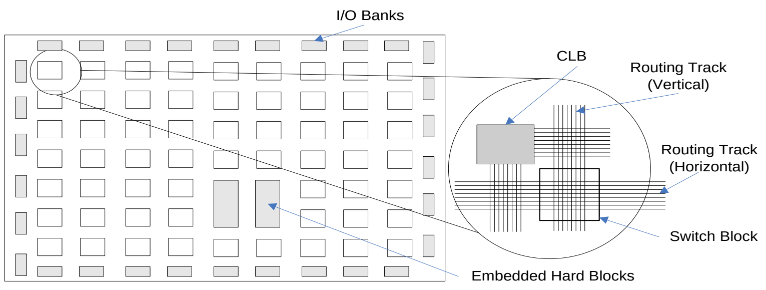

Fig. 1. Major Components of a Modern FPGA

The CLB is the basic logic unit in an FPGA. The exact numbers and features can vary between devices, but every CLB consists of a Look-Up Table (LUT) with multiple inputs, some selection circuitry and flip flops. The CLBs can be configured to implement combinatorial logic, shift registers, or RAM. While the CLBs provide logical implementation, flexible interconnect routing is used to route the signals between CLBs, and to/from I/Os. Different length routing wires are available to efficiently route signals to adjacent CLBs or across the FPGA. Routing decisions are typically based on Computer Aided Design (CAD) tools unless specified by the designer. Embedded blocks such as Block RAM, Multipliers, and DSPs are available to the designers to provide specialized functionalities. Such hard IPs are efficient alternatives to implementing resource-intensive modules using generic CLBs.

2.2 Dynamic Partial Reconfiguration

Whereas FPGAs are traditionally reconfigured between applications, Xilinx® FPGAs [8], and more recently those from Altera® [12], enable DPR of the programmable fabric. DPR allows multiple HMs to time-share a pre-defined portion of the programmable fabric while the remainder of the fabric stays active. Mission-critical applications are able to stay active while parts of the fabric are being updated

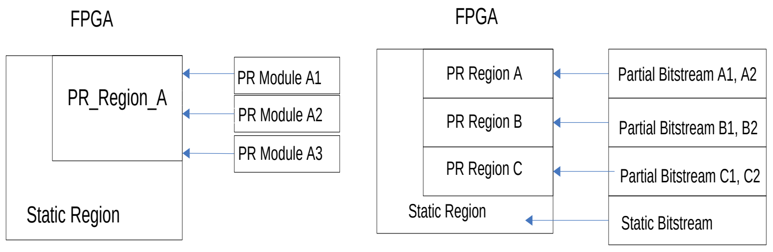

Fig. 2. DPR Concept and Terminologies

Figure 2 highlights the key DPR terminologies in this paper. A *Partially Reconfigurable* Region (PR Region) refers to a physical area of an FPGA that can be dynamically reconfigured to implement different tasks, and a Partially Reconfigurable Module (PR Module) is one of the possible tasks that can be implemented in a PR Region. Therefore, there exists an *n-to-1* mapping in time between PR Modules to PR Regions. Also shown in Figure 2, each PR Module has its own bitstream file, or *partial bitstream*, as does the static portion of the design (the *static bitstream*). A *full bitstream* for the FPGA comprises the static bitstream plus default partial bitstreams for each of the PR Regions.

2.3 Agent System

Agents in an agent system have the following properties [13]:

- Behavioral Autonomy – The ability for agents to determine autonomously how and whether to respond to requests.
- Localized Goal – No agent should have the entire view of the system, or the system is too complex for an agent to make practical use of such knowledge
- Loose Coupling – Agents, unlike programmed objects, are not tightly bounded and are allowed to determine how to best interconnect among themselves.
- Distributed Services – System services are distributed among the agents, where the agents work together without a central controlling authority.

An agent system is often used to solve complex problems that are difficult or impossible for an individual agent or a monolithic system to solve. Individual agents may be limited by their resources, but collectively they can be grouped to achieve a common objective that precedes individual goals. Agent systems have been widely used in practical applications such as automotive [14], manufacturing [15], automation [16], and pattern recognition [17]. They have also been widely advocated for use in networking and mobile technologies to achieve dynamic load balancing, high scalability, and self-healing networks [13].

Typically, agents are implemented in software to leverage code-reuse, portability, scalability and high levels of abstraction normally associated with software implementations. With DPR of FPGAs, it is now possible implement frameworks that accommodate hardware agents and leverage the benefits of a modern FPGA such as embedded processors, concurrency, and partially hardware updatability.

2.4 Hardware Implementation Challenges

FPGA-based systems use dedicated hardware for processing logic. Unlike software-based solutions that use context-switching to service multiple threads, FPGAs offer true concurrent and spatial processing so that different processing operations do not compete for the same resources. Other advantages include consistent and reliable performance at reduced clock rates, power consumption, and device count [8-10].

Despite its benefits, hardware designs remain complex, extremely time consuming, and require in-depth knowledge of the underlying device technology. Hardware design methodology has not been able to keep pace with the increased design complexity predicted by Moore's Law. These challenges include:

- *System Control Complexity (SCC)* - Hardware design offers true concurrency and therefore lacks a dedicated central controller that arbitrates competing requests.
- *Degree of Modifiability (DoM)* - An incremental change in hardware logic may require new spatial planning of the previously placed HMs and rerouting of their connections. This could lead to new spatial and latency

issues that are not present before the change, thus further complicating the design.

- *Universal Communication Abstraction (UCA)* - In addition to being physically connected, HMs must share complementing infrastructures to enable their communication. Unlike software in which there are commonly used universal communication abstractions, hardware developers are often required to implement unique interfaces to facilitate communication between modules.

There exists a need for a generic framework and accompanying communication infrastructure that is completely customizable to address these challenges and satisfy the dynamic nature of the HMs in a DPR application. The infrastructure must allow changes to how the HM is used at runtime, while keeping the communication infrastructure lightweight to efficiently utilize system resources within a PR Region.

3 Implemented Framework

3.1 Framework

An architectural example of the generic framework is shown in Figure 3. This framework is implemented using the Xilinx dynamic partial reconfigurable FPGA (Virtex5-LC50). Table 1 outlines the functionalities of the components in the framework.

The FPGA is divided into two partitions: Static and Partially Reconfigurable (PR). The Static Partition contains the embedded soft-core processor, MicroBlaze (μB), Hardware Administrator (HA), and a number of hardware peripherals. The Static Partition provides high-level administrative control and interfaces between the modules in the PR partition and the embedded processor. The PR Partition contains a predetermined number of PR Regions to host multiple HMs that time-share each PR Region. All HMs are implemented with an identical hardware interface, and are directly connected to the HA.

An extensive list of pre-verified HMs implemented as PR Modules can be made available to the embedded system developers. This architecture can easily be scaled to accommodate more complex applications. Multiple HMs and HAs can be added when there are sufficient resources available on the target device. Alternatively, multiple HMs can be connected directly in a simple mesh design without the use of an HA. Other Network-on-Chip (NoC) configurations such as star or bus structures can also be realized.

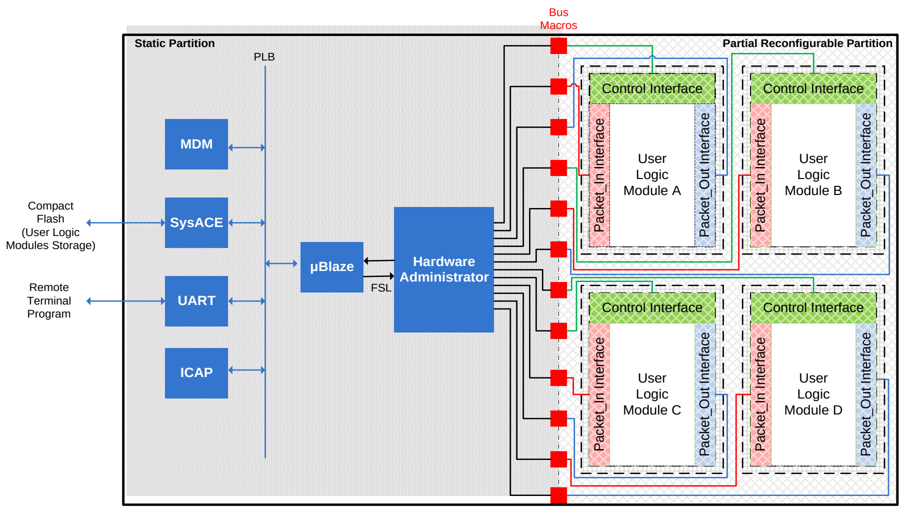

Fig. 3. Proposed FPGA-based Framework

Table 1. Functionalities of the Components in the Framework

Peripheral Local Bus (PLB) Xilinx 129-bit bus infrastructure for connecting an optional number of PLB masters and slaves into an overall PLB System **Block RAM (BRAM)** Memory used by the embedded processor **Hardware Internal Configuration Access Port (HWICAP)** Enables an embedded microprocessor such as the MicroBlaze to modify the current circuit structure and functionality during its operation **System ACE Controller** Interface between the PLB and the MicroBlaze to read and write to the System ACE™ Compact Flash. **Bus Macro (BM)** Xilinx-provided hard cores that are placed at fixed locations to facilitate unidirectional point-to-point communication between the Static and PR partitions. **Hardware Administrator** Functions as the administrator of the PR Partition. Routes data and control packets between the MicroBlaze and a specific PR Region. Provides real-time updates to the MicroBlaze with the availability and status of each PR Region. The MicroBlaze uses this information and makes intelligent scheduling and placement decisions of the PR Modules to the PR Regions. Provides flow and congestion control between the MicroBlaze and the PR Regions.

3.2 Hardware Modules (Hardware Agents)

As shown in Figure 3, each HM is encapsulated with a standardized communication infrastructure that is both light-weight and fully customizable. It abstracts low-level communication details between the HMs and allows developers to focus their efforts on high-level functionalities of the HMs, rather than low-level design intricacies.

Each HM has three interfaces: Control, Packet_Out (PO), and Packet_In (PI). The User Logic (UL) contains the actual functionality of the HM. The interfaces are shown in Figure 4.

The Packet-In (PI) interface receives customizable data and control information from other HMs or a central controller. The PI interface also provides input flow-control functionality to the User Logic. The Packet-Out (PO) interface functions identically to the PI interface, except that data, control, and flow-control information is sent out to other HMs or a central controller. The Control Interface provides supervisory control over the User Logic, and can inform other HMs or central controllers the current status or state of the User Logic. The functionality of the User Logic is application-specific and is implemented by the developers. The developers must conform to the standardized interfaces when designing the User Logic. All three interfaces (PI, PO, and Control) can be easily updated to provide customized solutions for different applications.

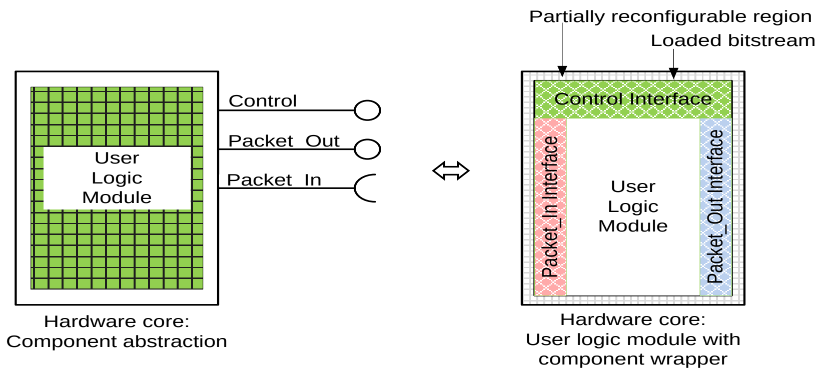

Fig. 4. HM Communication Infrastructure Abstraction

There are two important outcomes resulting from the framework: implementation of a hardware API wrapper that encapsulates the User Logic, and separation of a communication infrastructure from the User Logic. As previously stated in section 2.3, agents require the ability to be separated from the communication infrastructure which in turn allows for functional transparency. This loose coupling between communication and function is accomplished via the hardware wrapper. User Logic implemented as part of a PR Module can be dynamically loaded and unloaded without impacting the remainder of the programmable fabric. Also, the entire communication framework can be made into a dynamic module that allows different communication topologies to be swapped.

With the aid of the Hardware Administrator, hardware agents can easily discover and provide services for each other. This alleviates potential strain of the embedded processor that is already burdened with system administrative duties such as loading and unloading of the PR Modules.

4 Operating System Support

Figure 5 provides an illustration of how the framework could be implemented with embedded operating system (OS) design. The system contains two busses (PLB0 and PLB1) to minimize bandwidth bottlenecks during the hardware configuration process and other bus transactions resulting from HA and HM activities. MicroBlaze_0 hosts the OS (e.g., Petalinux) and Microblaze_1 is responsible for the management of the PR Regions including loading and unloading of the partial bitstreams through the internal configuration port (ICAP). The communication between the processors is done via a message passing mechanism. Hardware threads are able to be executed independently without the corresponding threads. It is also important to recognize that the burden of thread management can be off-loaded to the MicroBlaze_1 and the HA. Consequently, OS support can be focused on running intelligent system software for agent systems with DPR.

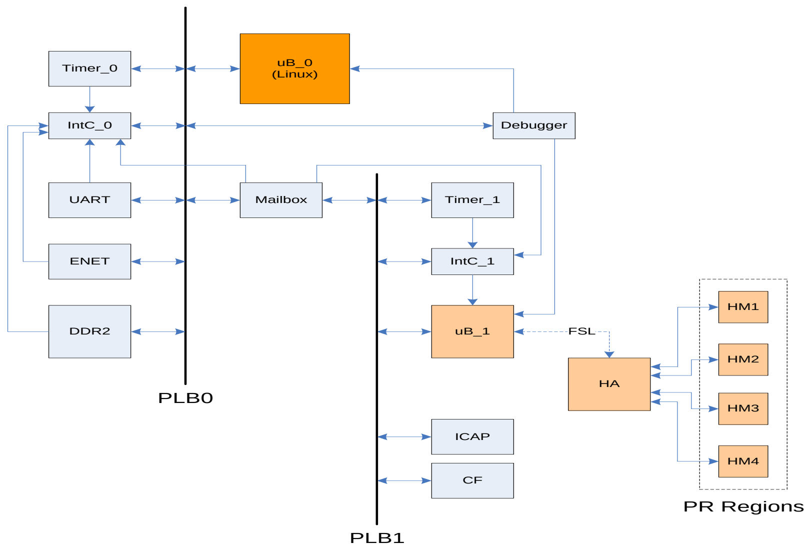

Fig. 5. DPR System with Linux

5 Conclusions and Comments

FPGAs using SRAM technology provide designers with a programmable fabric that can be configured at runtime to implement a specific hardware circuit. This technology allows designers to reconfigure the entire FPGA for a new application when the current application has been completed. DPR of FPGAs extends this technology by allowing multiple hardware modules to time-share a pre-defined portion of the programmable fabric while the remainder of the fabric stays active. Its advantages include partial updateability of the programmable fabric, reduced footprint, lower cost, reduced device count, and low-power dissipation [8-10].

This paper presented a FPGA-based framework with DPR that can be used to implement agents in an agent system. By using an agent paradigm, it is possible to separate the device logic structure from the communication infrastructure to leverage the benefits of software implementation such as component based designs. With DPR, communication infrastructure topologies can be dynamically replaced, thus raising the abstraction level of hardware design. Designers can focus on the functionality of system, rather than low-level communication and device architecture complexity.

The implementation of agents with partially reconfigurable modules bridges the software and hardware domains. This approach enables software and hardware systems to be more closely integrated during the design and development phases. The functionalities of the modules, whether described in software code or hardware logic, can be abstracted and CAD tools can be used to satisfy the design constraints of low-level implementations.

Future work includes the application of the proposed framework and communication infrastructure to practical agent systems, development of architectures for targeted applications, and the inclusion of an embedded OS to leverage software code-reuse, portability, interfaces, and existing services.

References

1. Lee, D., Choi, A., Koo, J., Lee, J., Kim, B.: A Wideband DS-CDMA Modem for a Mobile Station. IEEE Transactions on Consumer Electronics, vol. 45, no. 4, pp. 1259–1269 (1999).
2. Pirsch, P., Demassieux, N., Gehrke, W.: VLSI Architectures for Video Compression - A Survey. Proceedings of IEEE, vol. 83, no. 2, pp. 220–246 (1995).
3. Ovaska, S., and Vainio, O.: Evolutionary-programming-based Optimization of Reduced-rank Adaptive Filters for Reference Generation in Active Power Filters. IEEE Transactions on Industrial Electronics, vol. 51, no. 4, pp. 910–916 (2004).
4. Chen, R., Chen, G., and Chen, L.: System Design Consideration for Digital Wheelchair Controller. IEEE Transactions on Industrial Electronics, vol. 47, no. 4, pp. 898–907 (2000).
5. Sridharan, K., and Priya, T.: The Design of a Hardware Accelerator for Real-time Complete Visibility Graph Construction and Efficient FPGA Implementation. IEEE Transactions on Industrial Electronics, vol. 52, no. 4, pp. 1185–1187 (2005).

6. Gabrick, M., Nicholson, R., Winters. F., Young, B., Patton, J.: FPGA Considerations for Automotive Applications. Proceedings of SAE World Congress and Exhibition (2006).
7. Wang, J., Katz, R., Sun, J., Cronquist, B., McCollum, J., Speers, T., Plants, W.: SRAM based Reprogrammable FPGA for Space Applications. IEEE Transactions on Nuclear Science, vol. 46, no. 6, pp. 1728–1735 (1999).
8. Lysaght, P., Blodget, B., Mason, J., Young, J., Bridgeford, B.: Enhanced Architecture, Design Methodologies and CAD tools for Dynamic Reconfiguration for Xilinx FPGAs. Proceedings of International Conference on Field Programmable Logic and Applications, Madrid, Spain, pp. 1-6 (2006).
9. Kao, C.: Benefits of Partial Reconfiguration. Xcell Journal, Fourth Quarter, Xilinx, Inc. pp. 65-68 (2005).
10 Monmasson, E., Cristea, M.: FPGA Design Methodology for Industrial Control Systems – A Review. IEEE Transactions on Industrial Electronics, vol. 54, no 4 pp 1824-1842 (2007).
11. Field Programmable Gate Arrays [Online]. Available: http://en.wikipedia.org/wiki/fpga
12. Altera Corporation: FPGA Run-Time Reconfiguration: Two Approaches. White Paper 01055, version 1.0 (2008).
13. Wooldridge, M., Jenning, N.: Intelligent Agents – Theories, architectures, and Languages. Lectures Notes in Artificial Intelligence (1995).
14. Parunak, H. V. D. Brueckner, S. A., Sauter, J.: Digital Pheromones for Coordination of Unmanned Vehicles. Lecture Notes in Computer Science, vol. 3374, pp. 232–263. Springer Verlag (2004).
15. Valckenaers, P., T. Holvoet: An essential abstraction formanaging complexity in MAS-based manufacturing. Lecture Notes in Computer Science, vol. 3830. Springer Verlag (2006)
16. Weyns., K., Holvoet, T.: Exploiting a Virtual Environment in a Real-world Application. Lecture Notes in Computer Science, vol. 3830. Springer Verlag (2006).
17. Brueckner and, S., Parunak, H.: Swarming Distributed Pattern Detection and Classification. Lecture Notes in Computer Science vol. 3374, pp 232-245. Springer Verlag (2005).