DYNAMIC MEMORY MANAGEMENT FOR FPGA-BASED RECONFIGURABLE ARCHITECTURES

Miss. Bhagyashri Sayajirao Patil[1], Miss. ShitalArjun Shivdas[2], Prof. (Mrs.) M. M. Raste[3]

[1,2] VLSI & Embedded System, ADCET Ashta, (India)
[3]Assistant Professor, VLSI & Embedded System, ADCET Ashta, (India)

ABSTRACT

In this paper, an adaptive architecture for dynamic management and allocation of on-chip FPGA Block Random Access Memory (BRAM) resources is presented. This facilitates the dynamic sharing of valuable and scarce on-chip memory among several processing elements (PEs), according to their dynamic run-time memory requirements. The proposed scalable BRAM memory management architecture adaptively manages these dynamic memory requirements and balances the buffer memory over several PEs to reduce the total memory required, compared to the worst-case memory footprint for all PEs. The runtime adaptive system allocates BRAM to each PE sufficiently fast enough as required and utilized. The proposed system suited for the dynamic memory footprints of FPGA-based reconfigurable architectures.

Keywords- Block Random Access Memory (BRAM), Dynamic On-chip Memory Management Unit (DOMMU), Dynamic Partial Reconfiguration (DPR), Processing Elements (PEs).

1. INTRODUCTION

With the increasing complexity and performance requirements of real-time embedded systems and the advances in FPGA technology, came the advent of multi-processor architectures and, more recently, of reconfigurable computing. Reconfigurable computing exploits the reconfiguration capabilities of FPGA devices to reconfigure the resources on the FPGA to modify and adapt the functionality of these resources to a specific application or computation that needs to be performed. More recently, dynamic partial reconfiguration (DPR) of FPGAs provided the possibility to specify and constrain certain partitions on an FPGA such that they can execute different tasks at different points in time without consuming additional area.

One main challenge of dynamic reconfigurable computing is the efficient assignment of resources to different partitions, such as the scarce and valuable block random access memory (BRAM), which is often a limiting factor in the design of complex embedded systems. Modules designed to occupy the same physical partition on FPGA can only utilize the on-chip BRAM resources within this partition, which are often not sufficient for memory-intense applications. However, this imposes many physical design constraints on the FPGA-based implementation, and reduces its potential for flexibility and reconfigurability. Moreover, local on-chip memory is almost always the preferred memory choice for real-time applications, since it is the lowest latency (one clock cycle), fastest, and highest bandwidth memory solution available. Hence, it becomes necessary to design the system using maximum worst-case memory footprint estimates, but such static memory allocation is inefficient and would impose excessive area and power consumption overheads. Dynamic memory management is needed to enhance the gains of reconfigurable computing by meeting the dynamic context dependent memory requirements of embedded reconfigurable applications and to avoid costly static memory allocations at design-time.

In this proposed paper, a Dynamic On-chip Memory Management Unit (DOMMU) which is customized to target the run-time dynamic management of on-chip BRAM to parallel FPGA-based PEs, according to their dynamic runtime memory footprints. DOMMU is designed with flexible user-configurability and scalability. It supports automated BRAM (de)allocation, which ensures that memory management remains transparent to the PEs. Support for sharing BRAM between PEs is also integrated, and can be extended to support additional BRAM configuration types.

II DESIGN GOALS AND FUNCTIONALITY OF DOMMU

For DOMMU to dynamically manage on-chip memory allocation of PEs in reconfigurable computing, it has to meet the following requirements:

2.1 Dynamic Memory (De) AlIocation

Static memory allocation architectures often force PEs to reserve enough BRAM to cover worst-case requirements and to resort to off-chip memory for more. In typical cases, significantly less than worst-case memory is required, and the worst-case buffer can be provided for other PEs while unused. This dynamic sharing and allocation of memory can reduce the total memory required at run-time and improve BRAM utilization. However, dynamic allocation should be guaranteed to occur faster than the first access of the PE to this BRAM to ensure that memory requirements are served with quality.

2.2Transparency

An important design goal is to decouple the internal functionality of DOMMU from the PEs using it. Therefore, DOMMU's interface as well as its behavior and timing performance has to be identical to that of traditional BRAM access. This is achieved by BRAM virtual address mapping which is transparent to the PEs, and maintaining a single clock cycle latency for BRAM access. Moreover, it is necessary to provide all the PEs with access to their allocated BRAM simultaneously via independent dedicated channels without any bandwidth sharing. To provide a PE transparently with memory when it is needed, automated dynamic BRAM (de)allocation is realized which should be enabled or disabled for different PEs independently at run-time, according to the application requirements.

2.3Scalability

DOMMU has to be designed with user-configured parameters to make it reusable and scalable in terms of the number of memory ports, number of BRAMs managed, their types and configurations. Moreover, the required hardware resources have to scale well with increasing numbers of memory ports and managed BRAMs. Additionally, the design has to provide integrated support for shared BRAM for communication between PEs through dual-port BRAM access, and should be extensible to integrate application-specific BRAM type templates.

2.4 Conservation of an optimal point in design space

Since DOMMU replaces static allocation of BRAMs, the design space exploration for the architecture using DOMMU has to consider bandwidth, latency and hardware resources. Independent dedicated channels between PEs and their associated BRAMs assure a latency of one clock cycle for memory accesses. In order not to outweigh the gains of DOMMU, the hardware resources have to be kept minimal. This preserves the point in the design space of the original architecture, while enabling efficient utilization of BRAM resources by dynamic management.

III PROPOSED DESIGN

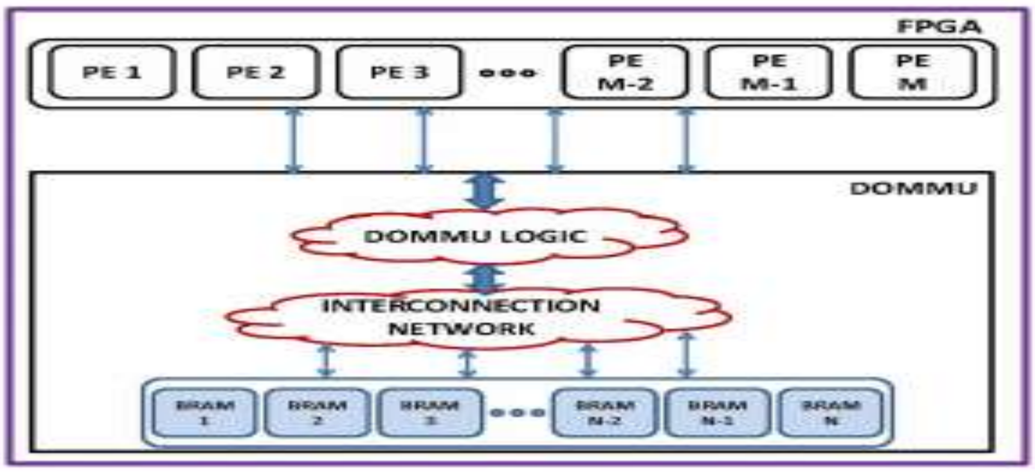

Fig. 1: Illustration of the general system overview of DOMMU

In Fig. 1, each PE is assigned one or more memory ports, by the user at design-time. These memory ports interface with DOMMU for BRAM (de)allocation and access. Memory ports share access to N BRAM elements via an interconnection network as shown in Fig. 1.

To manage this dynamic sharing while keeping the BRAM management transparent to the PEs, it must keep track of the BRAM configurations (width and depth) available "in stock", the BRAM assigned to each PE, the configuration details of this BRAM, how often the BRAM is accessed, and how much more or less BRAM is required by each PE at any point in time. To keep the BRAM management transparent to the PEs, an address mapping scheme ensures correct PE-BRAM association.

IV BRAM ORGANIZATION AND ADDRESS TRANSLATION SCHEME

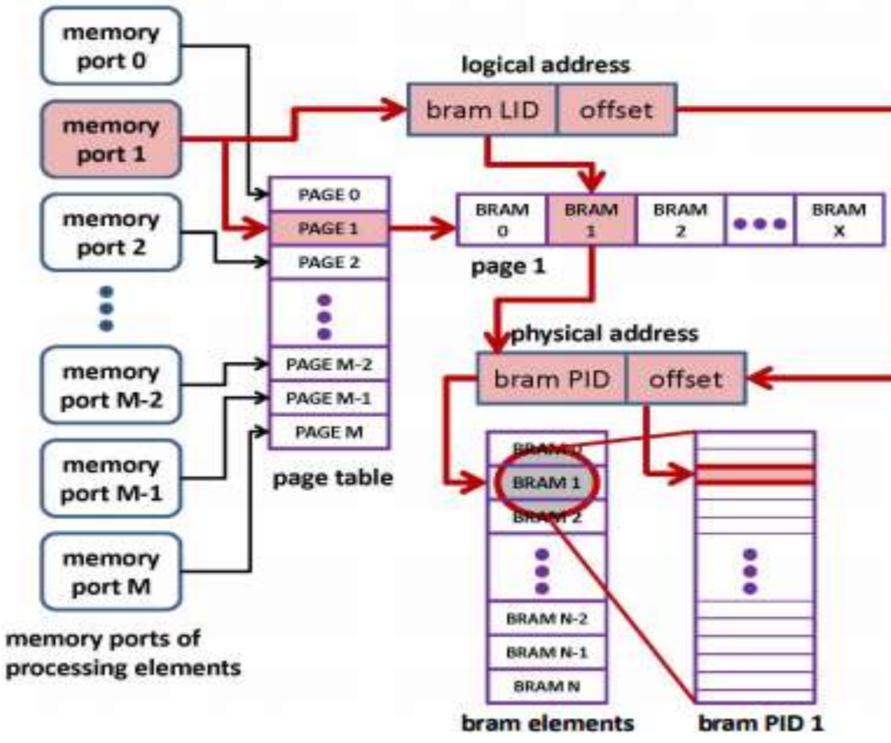

Fig. 2: Logical to physical address translation scheme of DOMMU

The set of BRAM elements shown in Fig. 2 and their physical configurations is the BRAM physical address space, which is realized by initializing a subset of the available BRAM resources on the device in different configurations (width X depth) depending on the design requirements.

To provide transparency to the PEs, the BRAM elements are also arranged in a logical address space, in the form of logical pages. Concepts of logical addressing and paging are borrowed from software memory management of operating systems, and employed similarly in the design of DOMMU. Each memory port is assigned a logical page which can be assigned up to X BRAM elements as shown in Figure 2. The BRAM elements are assigned a Logical Identification (LID) according to their order of assignment within the logical page. These LIDs are assigned at run-time independently of the Physical ID (PIDs) of the BRAM elements managed by DOMMU. Each memory port should "know" its logical page, its word width and depth. Each PE accesses its allocated BRAM by communicating the logical addresses via its memory port(s) to the DOMMU. The logical address is mapped to the physical address (BRAM PID and offset within the BRAM element) to access the correct data word. DOMMU interfaces with the PEs via the memory ports shown in Fig. 2, which introduces a degree of freedom to assign more than one memory port for each PE at design-time.

V ARCHITECTURE OF DOMMU

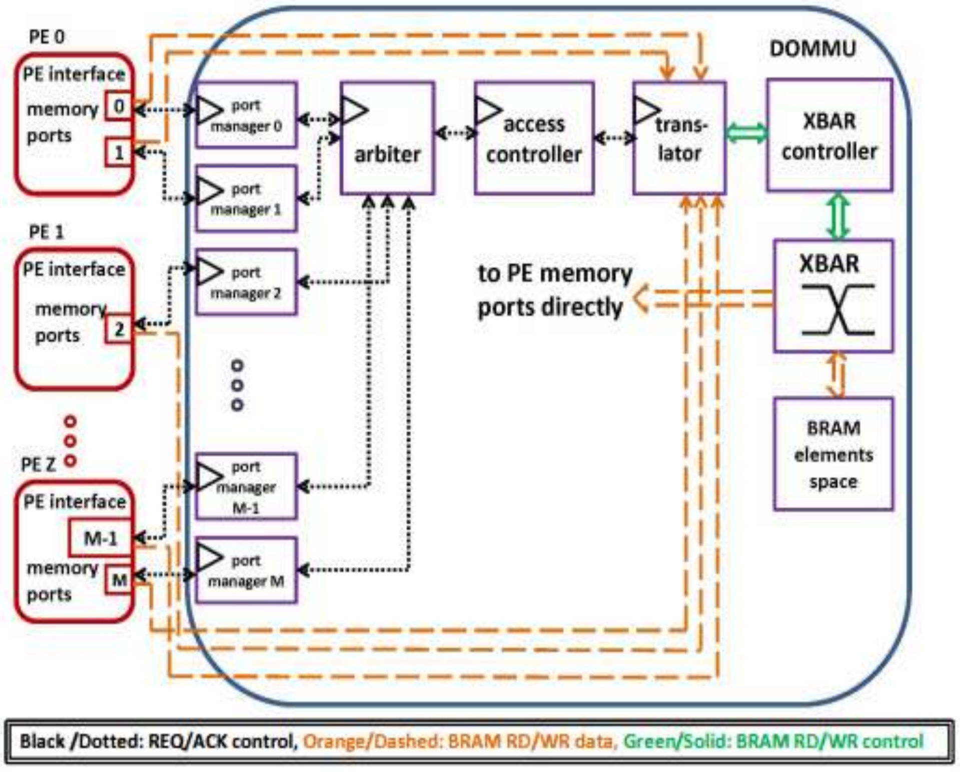

Fig. 3: Block diagram of DOMMU architecture

A detailed block diagram of DOMMU and its components is illustrated in Fig. 3.

5.1 Crossbar (XBAR) switch

The PE←> BRAM interconnection network required in DOMMU must allow all PEs to be physically able to access all the configured BRAM elements. Bi-directional communication is required to support both read and write access, as well as non-blocking switching to ensure that multiple simultaneous PE ←> BRAM interconnections can always be established. The crossbar switch satisfies these requirements.
The original idea was to dynamically reconfigure the FPGA routing resources to implement the crossbar switch, or implementing the crossbar multiplexers using LUTs and reconfiguring their configuration contents by bit stream manipulation via internal dynamic partial reconfiguration of the corresponding FPGA configuration frames, in order to control the multiplexed output. However, for ease of initial implementation and proof-of-concept, the crossbar is implemented in this work using regular multiplexers. It is realized using two crossbars: a unidirectional (PE → BRAM) crossbar for writing to BRAM, and a bi-directional (PE ←> BRAM) crossbar for reading from BRAM.

5.2 Address translator (BRAT)

The PEs communicate with the BRAMs by logical addresses. Hence, each memory port is assigned a BRAM Address Translator (BRAT), which performs the functionality described in Fig. 2.

When a read or write access request is received through a memory port (orange/dashed path in Fig. 3), BRAT maps this memory port to the associated logical page by queuing an array which maps each port to its corresponding logical page and the allowed access credentials (RD, WR, or RDIWR) of this memory port to this page. If the address is out-of-bounds or involves illegal access, the incoming address is rejected, and the PE is flagged for requesting an illegal access. This feature enforces implicit memory access rights to ensure that each PE can only access its assigned memory.

BRAT also receives incoming control requests from a controller to update its stored arrays for new (de)allocations. ACKINACK message reporting the status and details of each request is returned to the controller. Errors such as a full logical page that cannot be allocated more BRAM or an empty page that cannot be (de)allocated from are handled by returning the corresponding NACK message back to the controller. In general, all incoming control requests are acknowledged with ACKINACK response messages communicated to the controller which indicate the details of status of the request. BRA T also keeps track of the logical page associated with each memory port, and the details of each logical page, such as its access credentials, word width, allowed maximum and actual depth. All details about the BRAM elements assigned to each page are also stored to ensure correct PE-BRAM association, correct logical-to-physical address mapping, and detection of illegal accesses.

5.3 Arbiter- Arbitration using adaptive, dynamic and user configurable priorities was implemented, since this was most suited for dynamic reconfigurable systems. The scheduling priorities associated with these PEs are dynamic and can change at run-time. Every memory port is assigned a priority, which is one of the three levels: low, medium or high, and this priority level is assigned as static or dynamic either at design-time or run-time. A static priority maintains its default value throughout operation, unless it gets re-assigned explicitly by the PE. At every clock cycle, all incoming requests from all memory ports are read and arbitration selects the request to serve. If the waiting time of a request exceeds a userconfigured threshold and if the priority of that port is dynamic, then the corresponding priority is upgraded to the next level. The dynamic priority of a memory port also gets downgraded if its pending request gets served, and its request waiting time is smaller than a user-configured threshold. Configurable and dynamic priority arbitration is suitable for real-time embedded systems in which some running applications are more time critical than others, and scheduling priorities can be adjusted accordingly. Hierarchical arbitration is implemented in which higher priority is always reserved for all allocation requests followed by a lower priority for all (de)allocation requestsbecause allocation requests are more critical to the PE. Within every level of hierarchy, the assigned priorities are examined to schedule the highest priority request to be first served. Latency overhead due to arbitration is unavoidable yet critical, since it is a significant factor in the latency incurred in serving memory allocation requests, which is crucial for scheduling memory requests associated with real-time applications. Arbitration latency has a deterministic maximum which is a function of the number of PE memory ports configured and the maximum number of BRAM elements that can be requested at one time by any memory port. This latency should be considered when scheduling BRAM allocation requests, and is guaranteed, when dynamic automated BRAM allocation is enabled, to remain below the first access of the PE to the requested BRAM. This overhead can be reduced by minimizing the arbiter logic and dynamism. Moreover, more aggressive pipelining can be attempted in order to serve multiple BRAM requests at one time, although in the current architecture design this would result in inconsistencies in shared data arrays.

5.4 Memory Port Manager

Each PE memory port that interfaces with DOMMU consists of a dedicated BRAM access port and a control port for (de)allocation control requests. The BRAM access port constitutes of two independent ports. Data can be read from or written to one or both of them simultaneously which enables access of single-port BRAM as well as dual-port BRAM for double the bandwidth, and supports inter-PE communication, which is often required in real-time image processing applications. If a PE requires more BRAM bandwidth, additional memory ports can be configured for it. The control port is assigned a memory port manager which matches the requested BRAM type, word width, and number of words to the closest BRAM configuration (width X depth) available. This ensures that the internal BRAM management and configuration details remain transparent to the PE. Since this mapping is embedded and time-critical, and has to occur with minimal impact on timing performance and area overhead, this limits the maximum complexity of the methodology and logic implemented. There is no optimal resolution to this mapping problem due to the different optimization factors that can be considered such as speed, power or area utilization. Themethodology implemented in this work selects the match that minimizes in the number of BRAM elements assigned.

Automated dynamic (de)allocation of BRAM is one of the distinguishing features of DOMMU. This allows additional BRAM to be requested for allocation automatically when the assigned BRAM for the memory port is close to running out. This is indicated when the number of BRAM addresses that get written to, increase beyond a user-configured threshold. If the assigned BRAM remains idle. Dynamic (de)allocation can be enabled or disabled by each port manager at run-time according to the application requirements. This feature is based upon several simplifying assumptions that every incoming read/write access is a valid one, that every incoming write access is associated with a new BRAM address, and that when the number of idle cycles exceeds a certain threshold, that this BRAM is not required by the memory port anymore, and should be (de)allocated.

The currently supported control requests a PE can issue via a memory port to its memory port manager are allocating a new single-port or shared BRAM logical page, (de)allocating a BRAM logical page, or a requested number of words from a logical page, or assigning a new priority to the concerned memory port. The parameters required for each request depend on the request code issued, and each request is acknowledged by a response message which indicates the details of the granted/denied request.

VI CONCLUSION

In this paper, a Dynamic On-chip Memory Management Unit (DOMMU) is proposed to support dynamic BRAM sharing among several processing elements in FPGA-based dynamic reconfigurable architectures, such that the BRAM allocation and utilization adapt to the variable run-time memory footprints of the PEs. A dynamic fine-grain control of BRAM (de)allocation, as opposed to previous static traditional approaches is introduced, as well as a virtual BRAM addressing scheme, and an automated dynamic memory (de)allocation algorithm, thus making DOMMU superior to previous architectures in terms of scalability, flexibility and its usability for reconfigurable computing in particular.

REFERENCES

[1] M. Majer, J. Teich, A.Ahmadinia, and C. Bobda, "The Erlangen slot machine: A dynamically reconfigurable FPGA-based computer," *The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology, vol. 47, no.I, pp. 15-31,2007.*

[2] I. Koutras, A.Bartzas, and D. Soudris, "Adaptive dynamic memory allocators by estimating application workloads," in 2012 *International Conference on Embedded Computer Systems (SAMOS)*. IEEE, 2012, pp.252-259.

[3] D. Goehringer, L. Meder, M. Hubner, and J. Becker, "Adaptive multi-client network-on-chip memory," in 2011 *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, 2011, pp. 7-12.

[4] l. Anagnostopoulos, S. Xydis,A. Bartzas, Z. Lu, D. Soudris, and A. Jantsch, "Custom microcoded dynamic memory management for distributed on-chip memory organizations," IEEE *Embedded Systems Letters*, vol. 3, no. 2, pp. 66-69, 2011.

[5] M. ShalanandY. 1. Mooney, "A dynamic memory management unit for embedded real-time system-on-a-chip," *in International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 2000, vol. 17, no. 19, 2000, pp. 180-186.

[6] C. H. Hoo and A. Kumar, "An area-efficient partially reconfigurable crossbar switch with low reconfiguration delay," in 22nd 2012 *International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2012, pp. 400-406.