

# The Address Translation Unit of the Data-Intensive Architecture (DIVA) System

Herming Chiueh, Jeffrey Draper, Sumit Mediratta and Jeff Sondeen  
USC Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292  
{chiueh,draper,sumitm,sondeen}@isi.edu

## Abstract

The Data-Intensive Architecture (DIVA) system incorporates Processing-In-Memory (PIM) chips as smart-memory coprocessors to a microprocessor. This architecture exploits inherent memory bandwidth both on chip and across the system. Thus, performances of pointer-based and sparse-matrix computations as well as multimedia applications are significantly enhanced.

A key feature of the DIVA architecture is the address translation mechanism, which supports virtual addressing of application code and data. Instead of prohibitive conventional page tables, DIVA provides a simplified mechanism using segments. In this paper, the design of the address translation unit is presented, and trade-offs in VLSI design including performance, area, and design modulation are also discussed.

## 1. Introduction

The Data-Intensive Architecture (DIVA) project is building a workstation-class system using embedded memory technology to replace the memory system of a conventional workstation with “smart memories” capable of very large amounts of processing. The goal of the project is to significantly reduce the ever-increasing processor-memory bandwidth bottleneck in conventional systems. System bandwidth limitations are thus overcome in three ways, as illustrated in Figure 1: (1) tight coupling of a single processing-in-memory (PIM) processor with an on-chip memory bank; (2) distributing multiple processor-memory nodes per PIM chip; and (3) utilizing a separate chip-to-chip interconnect, for direct communication between nodes on different chips that bypasses the host system bus.

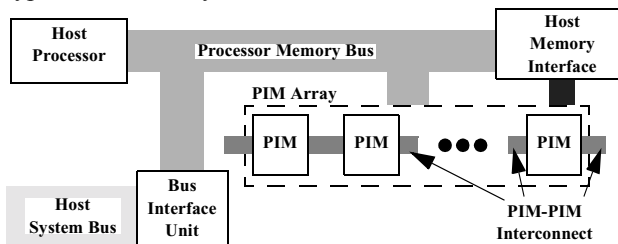


Figure 1. DIVA system architecture

This paper describes the design of an address translation unit, which is the key component that implements memory management in DIVA PIM chips. Previous literature [1] distinguished two aspects of memory management requirements from that of other PIM-based architecture.

- The PIM serves as the only memory for a standard host microprocessor, assuming the dual role of “smart memories” and conventional memory.
- DIVA targets applications that are most severely impacted by the processor-memory bottlenecks in conventional systems: sparse-matrix and pointer-based applications with irregular memory access patterns, and image and video applications with large working sets.

As compared to system-on-chip solutions [2-3], and multiprocessors made up solely of PIM chips [4-5], DIVA’s support for conventional memory access from an external host requires a dual view of memory, from host’s and the PIM’s perspective. A much broader range of programming paradigms are provided when compared with other PIM architectures. As a result, DIVA requires an efficient address translation mechanism and independent threads of control as the features in its memory models. A previous paper presented an overview of the DIVA project and described a memory model [6] and memory management [1] to support these requirements. This paper is focused on the design of the address translation unit and its circuit implementation. The remainder of the paper is organized as follows. Section 2 describes the mechanism of address translation in DIVA. Section 3 presents the detailed hardware design of the address translation unit (ATU). Section 4 presents a VLSI implementation and results, and Section 5 concludes the paper.

## 2. Address translation mechanism

The virtual address space of the host processor in the DIVA architecture can be categorized into three classifications:

- *Global memory* is composed of contiguous segments distributed across nodes, visible to applications running on the host and PIM nodes.
- *Dumb memory* is a region of a node’s memory allocated as conventional pages in a host

application's virtual space and untouched by PIM node processing.

- *Local memory* is a region of a node's memory used exclusively by node routines. This rule is excepted during initialisation when the host system boot process loads node software.

A node must be able to rapidly determine if an address is located in its own memory, and if so, find the physical address. Segments are used to condense translation information. Each segment is defined by segment registers containing a base address and size.

The local memory region is partitioned into eight segments in the DIVA architecture. Like pages in a conventional system, the segment descriptors are generic in nature. It is only through system programming that the segments serve a specific purpose [1].

Remote addresses are translated via the concept of a home node, which is guaranteed to have the translation information. In addition to the local segments, a node maintains translation information for its resident portion of the global memory, as well as for any remote data for which it is the home node.

The primary functions of the node address translation unit are to translate virtual addresses to physical addresses for those accesses that are locally resident and to provide access protection. The types of accesses generated by a DIVA PIM processor that require translation include instruction fetches and data accesses to memory or memory-mapped devices such as parcel buffers, generated by load or store instructions.

Given the simplicity of the address translation scheme discussed above, very little hardware support is needed to effect translation. A segment base address register and limit register is needed for each of the eight local segments. Also, one virtual base, limit, and physical base register are needed for each resident global segment. The DIVA architecture provides four sets of global segment registers. The address translation unit contains no direct support for home node translation, although the preferred system programming is such that the global segments resident on a node form the portion of global memory for which that node is the home node. If this is not the case, address faults invoke system software that performs the home node translation.

### 3. Design of ATU

The DIVA PIM processor provides 4 Gbytes of virtual address space accessible to kernel and user applications via segments that are a power of 2 in size. Segment sizes can range from 256 bytes to the maximum amount of physical memory available to a node. The maximum segment size in the initial DIVA system design is 16 Mbytes. Each virtual address generated by the PIM processor is 32 bits, and the resulting physical address generated by the address translation unit is also 32 bits.

The PIM processor address translation unit supports three main types of address translation: direct address

translation, local address translation, and global address translation

Figure 2 shows the three main address translation mechanisms provided. When the address translation unit is disabled, direct address translation occurs, and the address translation unit will not generate any exceptions. In this case, the resulting physical address is identical to the virtual address. If address translation is enabled, then the scope field of the virtual address must be inspected to determine what type of translation should be used.

The scope field of the virtual address is the most significant five bits of the virtual address VA. If this 5-bit value is zero, then local translation is used. If the scope field equals binary value 00001, i.e., the virtual address falls in the range of 0x08000000 to 0x0FFFFFFF, direct translation is used to generate the physical address; however, unlike the mode where address translation is disabled, an exception can be generated in this case if access privileges are violated. By definition, the address region 0x08000000 to 0x0FFFFFFF is a supervisor-level region. Therefore, any user-level attempt to access this region while address translation is enabled will trigger an exception. Lastly, if any of the four most significant bits of the virtual address are non-zero, i.e.,  $va[0:3] \neq 0$ , then global translation is used.

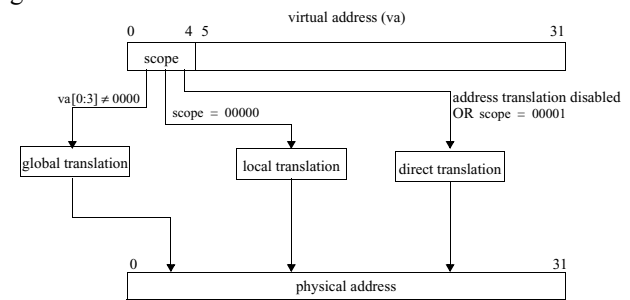


Figure 2. Address translation types

Figure 3 shows the steps involved in local address translation. The 3-bit index field of the virtual address is used to select a set of local segment registers for the translation. The segment base is simply bitwise-ORed with the zero-padded offset of the virtual address to form the physical address. The specified segment limit register is also accessed and manipulated in conjunction with the offset to determine if the virtual address is valid.

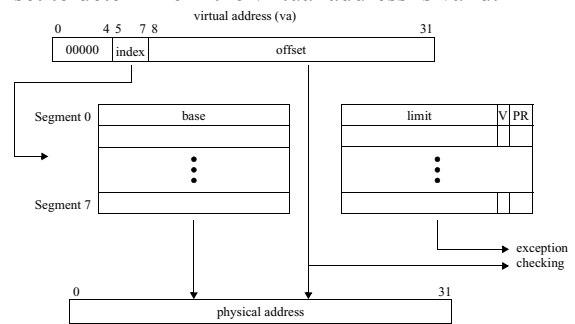


Figure 3. Local address translation

Figure 4 shows the steps involved in global address translation, which is a reverse address translation style. In this case, the address is checked to see if it is mapped

locally by simply ensuring that the address is within the range specified by a valid set of the global segment base address and limit registers. The hardware does not protect against overlapping global segments. The multiple sets of global segment registers are checked concurrently to see if any one of them should be used for the translation, similar to a fully associative cache. If there is a match, the virtual address is simply translated into a physical address by a bitwise-OR of an offset with the global segment physical base register of the matching global segment. The offset is formed by using the limit register of the matching segment to mask off the appropriate part of the virtual address.

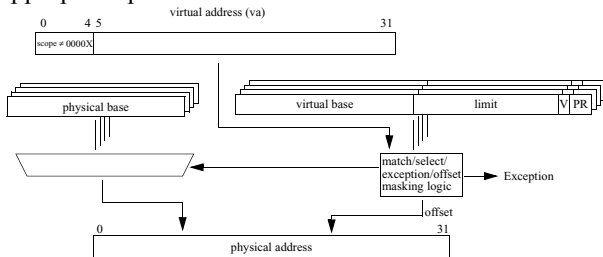


Figure 4. Global address translation

In addition to the translation of virtual addresses to physical addresses, the address translation unit provides access protection and bounds checking to ensure that the offset portion of an address is not outside the range of the segment. The 2 PR bits of a segment limit register specify the access protection mode for that segment. Table 1 shows the possible access modes and their corresponding encodings.

Table 1. Segment access modes and corresponding PR bit encodings

Encoding of PR Bits	Supervisor Privilege	User Privilege
00	RW(read-write)	RW
01	RW	RO(read only)
10	RW	None
11	RO	None

Each local segment limit register consists of a limit value, a valid bit, and the two PR bits. The first level of protection for local addresses is provided by ensuring that a valid set of segment registers is used. If the V bit of the selected local segment is not asserted, an unmapped access exception occurs. The second level of protection is provided by the PR bits. If the PIM processor mode and access type are not allowed by the PR bit setting of the selected segment, an invalid access exception occurs. The final level of protection for local addresses is provided with bounds checking. The limit value of the specified segment is used to inspect bits in the virtual address offset to ensure that the offset has not exceeded the segment size. If the segment size is exceeded, an unmapped access exception occurs. Equation (1) specifies the exception condition E for local translations.

$$E = (\overline{va_8} \wedge \overline{limit[index]_8}) \vee (\overline{va_9} \wedge \overline{limit[index]_9}) \vee \dots \vee (\overline{va_{23}} \wedge \overline{limit[index]_{23}}) \dots \dots \dots (1)$$

Although the conditions for address translation exceptions for global virtual addresses are similar to that of local addresses, the mechanism is quite different due to the fully associative nature of the global segment hardware. Basically, if one of the four sets of global segment registers does not *match* an attempted global address access, an exception occurs. A successful *match* occurs when a set of segment registers is valid, the PR bit setting allows the access type being attempted, and the address range specified by the global virtual base and limit encompasses the global address of the operation. Equation (2) specifies the range match condition RM, where va is the virtual address and base is the contents of the global virtual base register.

$$RM = (\overline{limit_0} \wedge (va_0 \oplus base_0)) \vee (\overline{limit_1} \wedge (va_1 \oplus base_1)) \vee \dots \vee (\overline{limit_{23}} \wedge (va_{23} \oplus base_{23})) \dots \dots \dots (2)$$

An unmapped access exception is triggered if there is no valid set of registers that satisfies the range match test. If there is a valid set of registers that satisfies the range match test, but the PR bits for that segment do not allow the attempted access, an invalid access exception occurs.

#### 4. Implementation and results

Based on the design presented in Section 3, the schematic for the ATU design is presented in Figure 5. Four major components were designed and implemented with Synopsys tools: virtual to physical translation(V2P) module, controller, special purpose register files, and probe circuit.

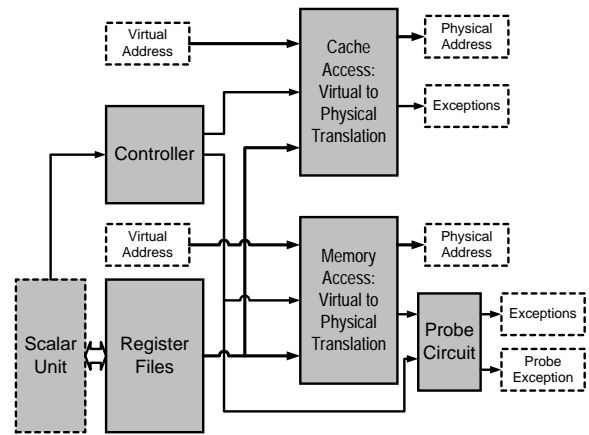


Figure 5. Schematic of ATU implementation

Design issues for these blocks are as follows:

- V2P module:** This is the core circuit to implement the translation scheme specified in Section 3. The key design issue for this circuit is the translation speed. To achieve the defined specification of 5ns, several techniques in VHDL coding for synthesis were required. The result was a pure combinational logic circuit that is able to implement the translation mechanism with minimum overhead in speed and circuit area.

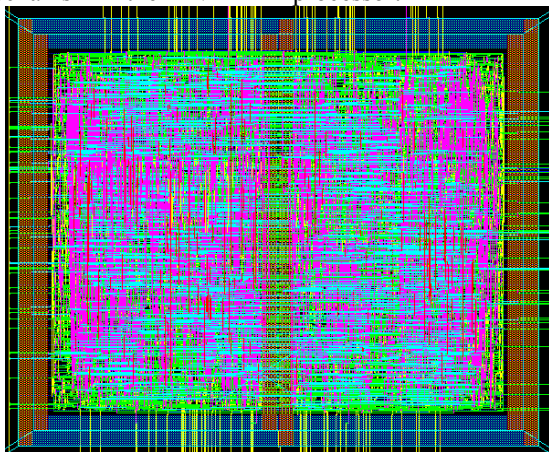
- Special purpose register file: This module is an interface for the PIM processor to set up the translation table. Since simultaneous translation and table set-up will never occur with proper system software, the core issue of this module's design is to reduce the circuitry area while providing a wide data bus with low propagation speed, which provides a complete static table look up for V2P module to speed up the translation.
- Controller: This module translates bus signals and memory access signals to two V2P modules, one for processor memory requests and the other for instruction cache memory requests.
- Probe Circuit: This module is used when a specific DIVA instruction is used to probe the address translation unit, allowing a user-level process to interrogate the status of a virtual address without incurring an exception if the address is not mapped.

Each module was designed and synthesized using Synopsys tools; the timing and circuit size of each module was optimised using the constraints mentioned above. The circuitry of the whole address translation unit was generated using Cadence Silicon Ensemble. The design is based on TSMC 0.18 $\mu$ m technology with Artisan standard cells. Table 2 summarizes the results. Varying the use of different optimisations, a great difference in circuitry area is observed. After several iterations in both synthesis and layout generation, a 30% reduction in circuitry area is achieved while maintaining the same fast translation speed by ignoring the constraint of the special register file's data path, which does not affect the translation speed.

**Table 2. Circuit Summary**

Gate counts	7967
Power	Core: 18.93mW Total: 41.78mW
Area	500 x 450 $\mu$ m
Percentage of modules' area	V2P Memory: 19.95%, V2P Cache: 19.58% Controllers: 0.03%, Probe circuitry: 0.2% Special purpose register file: 60.24%.

In Figure 6, a layout of the ATU is presented. The purpose of this layout is to form an initial estimate of the overhead of implementing an address translation mechanism in the DIVA PIM processor.



**Figure 6. Layout of ATU**

This circuitry occupies 500 x 450  $\mu$ m, which is 2.2% of the DIVA PIM processor area [7-8]. Considering most of the circuitry (60% of standard cells is register files) is not switching during the translation, there is only a 2.9% power increase for the DIVA PIM processor (28.8mW/580mW) to support the address translation mechanism. The overall delay for the ATU is 4.76ns, which is sufficiently fast to integrate it into the DIVA PIM processor without any extra delay.

## 5. Conclusion

This paper has presented the design and implementation of the Address Translation Unit to be used in the DIVA PIM processor. An implementation of this design, based on TSMC 0.18 $\mu$ m technology, has proven to be easily integrated into the current DIVA PIM prototype. The ATU is a key component to enable DIVA's memory management design, which is essential for a user-friendly programming paradigm for PIM systems like DIVA.

## Acknowledgments

This research was supported by DARPA contract F30602-98-2-0180.

## References

- [1] M. Hall and C. Steele, "Memory Management in PIM-Based Systems," in Proc of the Workshop on Intelligent Memory Systems, held in conjunction with Architectural Support for Programming Languages and Operating Systems, Boston, MA, 2000.
- [2] D. Patterson, et al., "A case for Intelligent DRAM: IRAM," IEEE Micro, 1997.
- [3] Mitsubishi, "M32R/D Series: 32-bit RISC Processor, On-chip DRAM," www.mitsubishi-chips.com/data/datasheets/mcus/m32rdgrp.html, May 6, 1999.
- [4] A. Saulsbury, T. Wilkinson, J. Carter, and A. Landin, "An Argument for Simple COMA," In Proc. of the Symposium on High-Performance Computer Architecture, 1995.
- [5] P. Kogge, "The EXECUBE Approach to Massively Parallel Processing," 1994 International Conference on Parallel Processing, Chicago, IL, 1994.
- [6] M. Hall, P. Kogge, J. Koller, P. Diniz, J. Chame, J. Draper, J. LaCoss, J. Granacki, J. Brockman, W. Athas, A. Srivastava, J. Shin, and J. Park, "Mapping Irregular Computations to DIVA, a Data-Intensive Architecture," in Proc. of SC '99, 1999.
- [7] J. Draper, I. Kim, J. Sondeen, and S. Mediratta, "Implementation of a 32-bit RISC Scalar Processor for the Data-Intensive Architecture (DIVA) Processing-In-Memory (PIM) Chip," Submitted to International Conference on Application-Specific Systems, Architectures, and Processors (ASAP), 2002.
- [8] J. Draper, C. W. Kang, and J. Sondeen, "Implementation of a 256-bit Wide Word Processor for the Data-Intensive Architecture (DIVA) Processing-In-Memory (PIM) Chip," Submitted to ESSCIRC, 2002.