

A 0.18 μ m CMOS Implementation of an Area Efficient Precise Exception Handling Unit for Processing-In-Memory Systems

Sumit Mediratta, Craig Steele, Ravinder Singh, Jeff Sondeen, Jeffrey Draper

USC Information Sciences Institute
Marina del Rey, California, USA-90292

{sumitm@ISI.EDU, steele@ISI.EDU, ravinder@ISI.EDU, sondeen@ISI.EDU, draper@ISI.EDU}

Abstract- This paper describes the implementation of the exception handling mechanism in the second prototype version of the Data-Intensive Architecture (DIVA) processing-in-memory (PIM) chip. This implementation features architectural simplicity, low area (54289 μm^2), delay (2.643 nanosecond) and power consumption (7.6 milliwatts), and effective hardware support for complex cases of exception handling. This work provides a description of handling memory-access, execution and communication-related exceptions in an area- and power-efficient manner, which are key design specifications for DIVA. The current implementation has been tested by verifying various exceptions on DIVA-II PIM chips running at 140MHz in the memory system of a HP Itanium2-based Long's Peak server. The generic nature of the DIVA exceptions and their classification makes the current implementation suitable and easy for use in diverse microarchitectures with little modification.

I. INTRODUCTION

Exception handling has long been of great significance to a broad segment of the computer engineering community, from software designers [1, 2], synchronous and asynchronous microarchitecture designers [3, 4, 5] and hardware implementers [6]. Although exception handling has been identified as one of the most complex issues in advanced microarchitectures, but most of the solutions are proprietary and little attention has been paid to solve this issue in a generic, effective (trade-off between the hardware and software responsibility for complex cases), performance- (both normal and exception case) and implementation-efficient manner. Exception handling has also become crucial for PIM systems, such as DIVA^T [7], which aim to mitigate the processor-memory speed gap. The work presented in this paper was done as an integral part of the design of the DIVA (fig. 1), but it can be used as a standalone component and integrated in other similar microarchitectures with area and power efficiency as key design requirements.

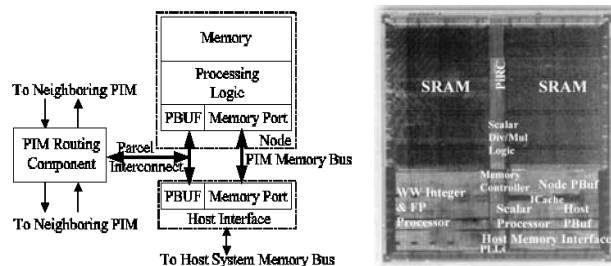


Figure 1. DIVA chip architecture and microphotograph

PIM architectures combine processing logic with memory in smart PIM chips to address the well-known performance gap between processor and memory speeds. Many previous architectural solutions to this *memory wall* problem, such as multithreading, prefetching, and speculation, seek to reduce or tolerate memory latency, at the expense of increased memory bandwidth requirements [8]. In contrast, PIM internal processors can be directly connected to the memory banks, so the memory bandwidth is tremendously increased. Latency of on-chip memory accesses is also reduced as compared to off-chip memory access latencies of conventional memory systems. One other approach is the use of large cache on processor chips, but this is of less help in cases where

memory accesses exhibit irregular patterns and there is a loss of locality in the distribution of the required data in the main memory. DIVA PIMs represent the first smart-memory coprocessors designed to support general memory-to-memory communication between independent threads of control. The support for in-memory virtual addressing, with the help of an address translation mechanism embedded in DIVA, also gives a leading edge to DIVA PIMs. The target applications for the DIVA PIM system are applications that need to operate on large data sets (image processing and multimedia applications requiring streaming of data) and programs where irregular memory accesses are abundant (programs performing sparse-matrix and pointer-based computations). DIVA PIMs support standard memory accesses as they can be configured in either a commercial DDR-DRAM or SDRAM interface mode. The second prototype of DIVA has been fabricated in TSMC 0.18 μ m technology (fig. 1) and is operating at 140MHz.

The significance of this work is in its exploration of exception handling mechanism implementation issues pertaining to requirements of processing-in-memory architectures. Despite prior research in the exception-handling field for the cases of multiple-issue and out-of-order execution deep pipeline processors [3, 4], this work follows the principles of the DIVA PIM philosophy of area and power efficiency where single-issue, in-order and five-stage pipeline processing logic has been implemented [9, 10, 11]. This exception-handling scheme has a modest hardware requirement (an important resource for processing logic on basically a memory chip), exporting much of the complexity to flexible software while providing hardware support for traditional stack-based handlers. It provides an integrated mechanism for handling hardware and software exception sources. It also features a flexible priority assignment scheme, which minimizes the time during which exception recognition is disabled. These properties make this design more attractive, as compared to mechanisms with multiple stack registers, extensive hardware-supported vectoring and priority-level controls of enabling exceptions, because of their more error-prone (priority inversion and stack management errors in handler) nature and difficulty in correcting errors (like priority assignment) once they are cast in hardware.

Section 2 gives a background on the types of exceptions used in DIVA and categorizes them, and section 3 presents the hardware support provided for exception detection and handling. Section 4 gives the implementation results of the current design in TSMC 0.18 μ m technology, analyzes its performance and characterizes its area and power consumption with respect to its operating speed. Finally, section 5 concludes this paper.

II. TYPES OF EXCEPTIONS

The DIVA exceptions are implemented as *precise* exceptions (the taxonomy provided in [12] is used). The instructions are executed in an in-order and single-issue manner to save hardware required for otherwise complex control logic, and this also prevents the exception handling mechanism from becoming *imprecise*. Most of the exceptions are generated in either the fetch, decode, execute or memory stage of the processor pipeline in a *synchronous* manner. But, they are recognized only in the memory stage to allow the successful execution of instructions issued before the

^T This research was supported by DARPA contract F33615-03-C-4105.

exception causing instruction. Only the network interface can generate communication related interrupts in an *asynchronous* manner.

A. Hardware-vectored exceptions

The PIM node processor has three hardware-vectored exceptions (table 1). All other exceptions are dispatched by software with some hardware assistance. All the vector addresses, pointing to the exception handler routines, are located in the vicinity of the start address of the node processor, the primary exception handler table must therefore be initialized and functional for any operation beyond Reset.

TABLE I

HARDWARE-VECTORED EXCEPTIONS

Exceptions	Handler Address
Reset	0x08000000
Undefined Instruction	0x08000100
Software-vectored exceptions	0x08000200

The undefined instruction handler serves all undefined instruction exceptions and also serves as the primary exception handler for breakpoint instructions. The software-vectored exception handler provides initial exception handling for all other exceptions and interrupts in the system, and performs a software-vectored dispatch to the appropriate exception handler.

B. Classification of software-vectored exceptions

The software-vectored exceptions constitute most of the exception sources in DIVA. They can be initiated by either hardware (HW) or software (SW). As the exception-handling address is the same for all the software-vectored exceptions, an Exception Source Word (ESW) register is provided to determine the exception cause and take appropriate action to handle it. Making use of the location of the exceptions in the ESW, the priority of exceptions can be flexibly defined in the exception handler routine at address 0x08000200. For example, one way of defining priority is from MSB to LSB of ESW.

The exception handler can support nested exceptions if it saves essential state information before enabling exceptions again. The exception handling procedure can be split into primary and secondary parts. The primary exception handlers are non-interruptible except for the *Reset* and *Undefined Instruction*. The secondary exception handlers can be interrupted by other exceptions and further occurrences of the same exception type (if it is not masked upon first occurrence). The software-vectored exceptions can be classified into three major categories, namely memory-access, execution and communication.

The memory-access exceptions (table 2) result from either an instruction or data access that is not within segment boundaries, or an unauthorized instruction or data access due to a violation of access privileges enforced by the address translation unit [13]. The secondary exception handler signals that it has corrected an exception by setting the corresponding *Fix-up* exception, which is noted by the primary exception handler.

TABLE II

MEMORY-ACCESS RELATED EXCEPTIONS

Exception Name	Initiator	ESW Bit
Unmapped Instruction Access	HW	1
Invalid Instruction Access	HW	2
Unmapped Data Access	HW	3
Invalid Data Access	HW	4
Address Fault Fix-up	SW	10

The DIVA execution related exceptions (table 3) are very commonly used and many of them are self-explanatory by their names. The processing logic consists of the scalar processor (32-bit integer data path) [9] and the wide-word processor (256-bit data path consisting of eight integer and eight single-precision floating point units) [10, 11]. The *wide-word (WW) not available* and *floating point (FP) not*

available exceptions are generated, when a WW or FP instruction is attempted when the corresponding enable bit in the Program Status Word (PSW) is not set. Since only the supervisor can write the PSW, this provides a mechanism of granting particular capabilities to users. Similarly, if a user tries to execute an unauthorized supervisor level instruction then the *privileged instruction violation* exception occurs and invokes the kernel. The FP and WW integer exceptions represent the aggregated version of the exceptions in one or more of the eight 32-bit units.

TABLE III
EXECUTION RELATED EXCEPTIONS

Exception Name	Initiator	ESW Bit
Interval Timer	HW	7
WW Not Available	HW	8
FP Not Available	HW	9
FP DZ (Divide By Zero)	HW	15
FP Overflow or Underflow	HW	17
Context swapper	SW	18
System Call	HW	19
Privileged Instruction Violation	HW	20
Scalar ALU Overflow or DZ	HW	21
WW Integer ALU Overflow	HW	22
FP IEEE 754 Inexact/Invalid	HW	23
Integer ALU Fix-up	SW	24
WW ALU Fix-up	SW	25
FP Fix-up	SW	26
Lock Buzzer	SW	28
Thread Rescheduler	SW	29
Thread Dispatcher	SW	30
Return to user mode	SW	31

The communication related exceptions are specific to the DIVA communication mechanism. The network interface, known as the parcel buffer (pbuf), generates *Receive Interrupt* upon receiving a parcel (or packet) with interrupt bit set, or if reception of a new parcel is blocked for 1024 cycles, or if the *eid* of the received parcel mismatches with that of the reading process. The *Send Interrupt* is activated upon the inability to generate route by hardware. The *Received Packet Processing* and *Send Packet Processing* are used by the secondary exception handler to signal successful handling of exceptions to the primary exception handler.

TABLE IV

COMMUNICATION RELATED EXCEPTIONS

Exception Name	Initiator	ESW Bit
PBuf Receive Interrupt	HW	5
PBuf Send Interrupt	HW	6
Received Packet Processing	SW	11
Send Error Processing	SW	12

The ESW bits not defined in the given tables are implemented as software-initiated exceptions and reserved for future enhancements in the exception mechanism. The definition of software exceptions is extremely liberal here, as the runtime kernel will define both the priority and meaning of software-initiated exceptions.

III. EXCEPTION HANDLING SUPPORT AND MECHANISM

A. Hardware Support for Exception Detection and Handling

Several protected (accessible in supervisor mode only) registers are provided for exception capturing, efficient state saving and recovery purpose. Hardware Initiated exceptions (HI_Except) can be captured directly in the ESW (fig. 2(a)), and Software Initiated exceptions (SI_Except) can be written by executing a protected register write to the Exception Set Register (ESR) with the corresponding data bit set (fig. 2(b)). Any exception bit can be reset by another protected register write to the Exception Reset Register (ERR) with the corresponding data bit set.

For recognizing the exception (fig. 3(a)), the ESW's input (ESW) will be updated at the next positive edge of the clock) is bit wise ORED with its output (useful for nested exceptions where the address of an exception enabling instruction is

captured), the result is subjected to a bit wise AND operation with the output of a protected register used for individual exception enabling called the Exception Mask Register (EMR). This 32-bit result, along with the undefined instruction exception bit, is searched for any set bit. If any bit is set and the exception enable bit in PSW is set (serving as a global exception enabling function), the exception is detected. The exceptions can be detected only in a particular state (fig. 3(b)), are disabled when an exception handler is running and can be enabled again when a Return From Exception (RFE) instruction is executed.

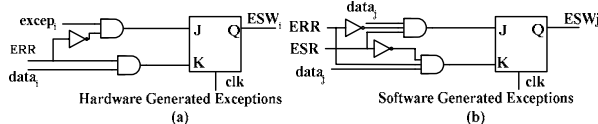


Figure 2. Exception capturing in ESW

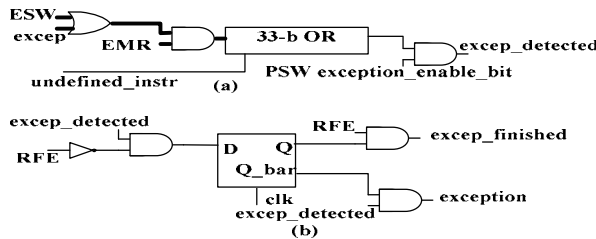


Figure 3. Exception detection and completion

Other important protected registers are the faulting instruction address register (FADR), next to faulting instruction address register (NFADR), saved program status word (SSW) and faulting data memory address register (MADR). All these supervisor-level registers can be written either directly or when the corresponding exceptional conditions occur (fig. 4 (a)). The value of the instruction addresses in the memory and execute stages are stored in FADR and NFADR, respectively. The mode is changed to supervisor (active low means supervisor) and exception recognition is disabled (enable is active high) when an exception is taken (fig. 4 (b)). PSW is restored back to its previous value as saved in SSW, upon the execution of RFE.

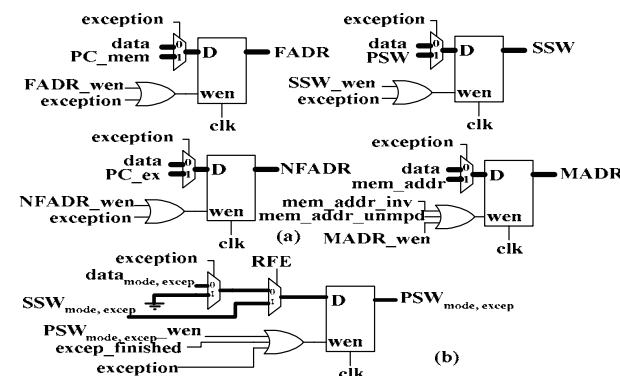


Figure 4. Protected registers writing

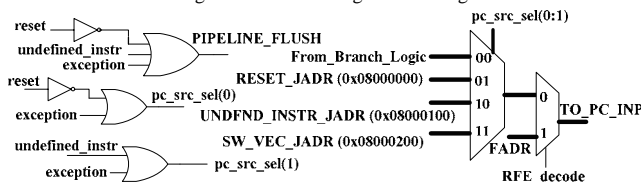


Figure 5. Handler and return address generation

The fetch, decode and execute stages (containing instructions after the exception causing instruction) are flushed upon recognition of any of the three hardware vectored exceptions (fig. 5). The input to the Program

Counter (PC) is calculated using the normal case instruction address (critical path comes from the branch logic) and the type of hardware-vectored exception (each with a separate jump address (JADR) for the exception handler routine). Further, the RFE instruction in the decode stage will copy the FADR to the PC to restore the program execution at the faulting instruction address.

B. Special Design Considerations

Some special cases were faced while implementing this exception handling mechanism. One of the most important issues was support for exceptions caused by a branch delay slot instruction (branch is one delay slot in DIVA). If the delay slot instruction causes an exception and the branch was taken, without more information than just the faulting address the kernel cannot restore execution in the right order because there is not any way of knowing what instruction was issued after the delay slot instruction. To prevent unsupported exceptions in this case, the NFADR was added to communicate the address of the instruction following the delay slot to the exception handler. The exception handler, upon finding a discontinuity between the FADR and NFADR values (differ by more than 0x00000004), can restore execution correctly at the instruction address provided by NFADR instead of FADR.

Another important issue was the support of a delay slot instruction for the RFE because it behaves like a branch. If there are nested exceptions waiting in the ESW, since exceptions are enabled by RFE, the pipeline will be flushed immediately and the delay slot instruction is not executed. This condition is avoided by actually treating the RFE delay slot instruction as an RFE instruction (fig. 6), and thus initiating exception handling completion actions upon the arrival of delay slot instruction in the memory stage (except for the calculation of PC that is initiated when RFE is in decode stage (fig. 5)). Again NFADR can be used in this case also to restore the program execution correctly.

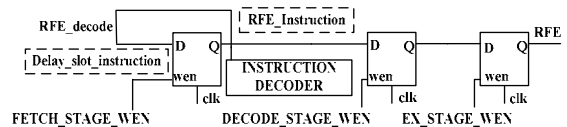


Figure 6. Support for RFE delay slot instruction

For efficient state saving and recovery, the exception handler can check the WW and FP enable bits provided in the PSW and avoid saving of the WW register file contents (if they were not enabled before exception), and thus save power consumption and gain in performance.

IV. IMPLEMENTATION RESULTS AND PERFORMANCE ANALYSIS

The current design is specified in RTL-level behavioral VHDL code. The code was synthesized using Synopsys Design Compiler targeting the Artisan standard cell library for TSMC 0.18μm technology. The resulting netlist was timing-driven placed and routed using Cadence First Encounter (FE) and timing of the placement was optimized using the In-Place Optimization (IPO) feature of FE. The clock tree displays an insertion delay of 0.5 nanosecond, slew rate of 0.2 nanosecond and skew of 0.1 nanosecond. The layout of the exception-handling unit is shown in fig. 7.

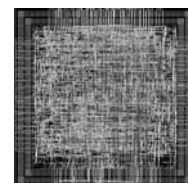


Figure 7. Layout of exception handling unit

Table 5 summarizes the performance metrics of the exception-handling unit, which was generated with the constraints representing the metrics of the actual implementation that is a tightly integrated part of the DIVA PIM chip (5 nanosecond clock cycle constraint). Its area constitutes 0.136% of the total processing logic (approximately 39.85 mm²) and 0.045% of the total chip area (10.5 mm X 11.5 mm). The layout has 80% cell density, and power measurements are made assuming a 20% duty cycle.

TABLE V
IMPLEMENTATION RESULTS

Transistors	17,002
Total area	233 μm X 233 μm
Delay (nanosecond)	2.643
Power (milliwatts)	7.6

PIM chips containing the exception-handling implementation have been assembled into DIMM boards (fig. 8) and have been integrated into a Hewlett-Packard (HP) Itanium2-based Long's Peak server. The exception handling mechanism has been tested in the above system, with the PIM processor running at 140 MHz, for *Reset*, *Undefined Instruction* and various *Software-Vectored* exceptions. The verified software-vector exceptions include *Unmapped Data Access* (chosen from the memory-access related category), *Interval Timer* (important for operating system functions), *FP IEEE 754 Inexact* (to verify ALU related exceptions), and *PBuf Receive Interrupt*. Proper pipeline flushing was observed for each of these tests, ensuring a precise exception handling mechanism. Instrumented measurement experiments are currently being conducted for several other benchmarks.

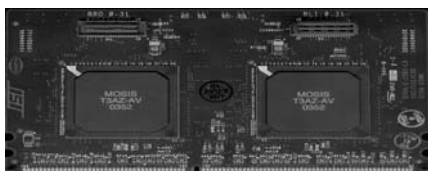


Figure 8. SODIMM containing DIVA chips

The area and power of the current design were characterized with respect to the variation of operating speed constraints (fig. 9). The sequential logic area (approximately 22,000 μm²) and wiring area (cell density of layout approximately 80%) didn't vary much with delay, and mostly the combinational logic increase dictated the shape of the total area curve. The increasing power trend of the design with the delay reduction displays the combined effects of increased area (hence more transient and static power) and higher frequency. While reducing the clock cycle from 2.643 nanosecond to 1.968 nanosecond can be accomplished with modest area increase, further reductions are not merited due to significant area and power increase.

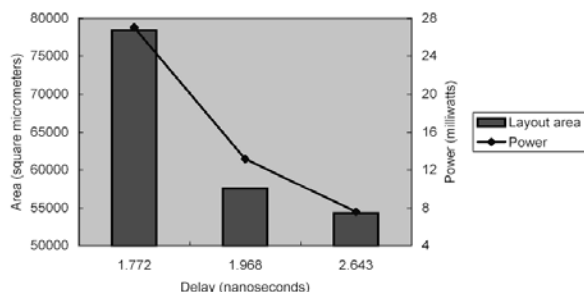


Figure 9. Characterization of area and power with respect to delay

V. CONCLUSION

The implementation details of an area and power efficient exception-handling unit, suited especially for PIM systems, are reported in this paper. The presented exception handling mechanism requires little specialized hardware support and supports preemption of lengthy low priority handlers. This design is thus ideal for integration in the DIVA system, with low area and power being a critical performance metric in the DIVA microarchitecture philosophy. The implementation results on the HP Itanium2-based Long's Peak server containing DIMMs that use DIVA-II PIM chips have been reported.

This paper also provided an overview and description of memory-access, execution and communication related exceptions implemented in DIVA. The generic nature of these exceptions and their classification, make this exception-handling unit attractive for assimilation into other systems with area and power efficiency as important performance metrics. Furthermore, the characterization of area and power of the current design with respect to variation in its delay constraints is provided to facilitate its integration into systems running at higher speed.

REFERENCES

- [1] J. DeVale, P. Koopman, "Performance evaluation of exception handling in I/O libraries," *International Conference on Dependable Systems and Networks*, pp. 519-524, July 2001
- [2] K. Ishizaki, T. Inagaki, H. Komatsu, T. Nakatani, "Eliminating exception constraints of Java programs for IA-64," *International Conference on Parallel Architectures and Compilation Techniques*, pp. 259-268, Sept. 2002
- [3] S.W. Kekckler, A. Chang, W.S.L.S Chatterjee, W.J. Dally, "Concurrent event handling through multithreading," *IEEE Transactions on Computers*, pp. 903-916, Sept. 1999
- [4] C. Kozyrakis, D. Patterson, "Overcoming the limitations of conventional vector processors," *30th Annual International Symposium on Computer Architecture*, pp. 399-409, June 2003
- [5] S.B. Furber, J.D. Garside, D.A. Gilbert, "AMULET3: A High-Performance Self-timed ARM Microprocessor," *International Conference on Computer Design: VLSI in Computers and Processors*, pp. 247-252, October 1998
- [6] R. Manohar, M. Nystrom, A.J. Martin, "Precise exceptions in asynchronous processors," *Conference on Advanced Research in VLSI*, pp. 16-28, March 2001
- [7] J. Draper, J. Chame, M. Hall, C. Steele, T. Barrett, J. LaCoss et al., "The Architecture of the DIVA Processing In Memory Chip," *International Conference on Supercomputing*, June 2002
- [8] D. Burger, J. Goodman, and A. Kagi. "Memory bandwidth limitations of future microprocessors," *23rd Annual International Symposium on Computer Architecture*, May 1996
- [9] J. Draper, J. Sondeen, S. Mediratta, I. Kim, "Implementation of a 32-bit RISC Processor for the Data-Intensive Architecture Processing-In-Memory Chip," *IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, July 2002
- [10] J. Draper, J. Sondeen, C. W. Kang, "Implementation of a 256-bit WideWord Processor for the Data-Intensive Architecture Processing-In-Memory Chip," *28th European Solid-State Circuit Conference*, September 2002
- [11] J.S. Moon, T.J. Kwon, J. Sondeen, J. Draper "An Area-Efficient Standard-Cell Floating-Point Unit Design for a Processing-In-Memory System," *29th European Solid-State Circuit Conference*, September 2003
- [12] J. Hennessy, D. Patterson, *Computer Architecture: A Quantitative Approach*, 2nd edition, Morgan Kaufmann Publishers, 1996
- [13] H. Chiueh, J. Draper, S. Mediratta, J. Sondeen, "The Address Translation Unit of the Data-Intensive Architecture System", *28th European Solid-State Circuit Conference*, September 2002