PAPER

Architecture and Implementation of a Reduced EPIC Processor

Jun GAO[†a)], Minxuan ZHANG[†], Zuocheng XING[†], *Nonmembers, and* Chaochao FENG[†], *Student Member*

SUMMARY This paper proposes a Reduced Explicitly Parallel Instruction Computing Processor (REPICP) which is an independently designed, 64-bit, general-purpose microprocessor. The REPICP based on EPIC architecture overcomes the disadvantages of hardware-based superscalar and software-based Very Long Instruction Word (VLIW) and utilizes the cooperation of compiler and hardware to enhance Instruction-Level Parallelism (ILP). In REPICP, we propose the Optimized Lock-Step execution Model (OLSM) and instruction control pipeline method. We also propose reduced innovative methods to optimize the design. The REPICP is fabricated in Artisan 0.13 μm Nominal 1P8M process with 57 M transistors. The die size of the REPICP is 100 mm^2 (10 × 10), and consumes only 12 W power when running at 300 MHz.

key words: *ILP, EPIC, IA-64, processor architecture, hardware implementation*

1. Introduction

Over the past two decades, designers have focused on increasing frequencies and exploiting parallelism to improve the performance of processors. The development of semiconductor technology has led to the rapid increase in processor frequency. However, with the increasing complexity of integrated circuits, the frequency enhanced by using advanced technology is limited. To enhance the performance of processors further, it must be depended on improving the processor architecture [1].

ILP has made great contribution to improve the performance of the processor. There are two traditional ILP approaches: superscalar and VLIW [2]. Superscalar is a hardware-based approach and shown to have diminishing profit returns because of the increasing design complexity, which imposes a great challenge on integrated circuits industry. In contrast, VLIW is a software-based static compiler technique, which can analyze program with a large virtual instruction window globally. It can exploit more ILP than superscalar, and hardware design is much simpler. However, VLIW has its own drawbacks in handling numerous dynamic indeterminate factors in programs, such as the dynamic behaviors of branch, the indeterminacy of access time, etc. It also has serious problems of wasted code space and poor code compatibility. Consequently, an ILP-oriented technique, Explicitly Parallel Instruction Computing (EPIC), has emerged.

As a design philosophy, EPIC takes advantage of both static and dynamic scheduling sufficiently, utilizing the cooperation of compiler and hardware to enhance ILP. It can not only develop as much ILP as VLIW but also has fixed-length instructions and supports register-register operations similar to those in the superscalar RISC processor. EPIC avoids the hardware complexity of superscalar and the code space waste, poor code compatibility of VLIW, and introduces a communication mechanism between compiler and processor. Processor can expose hardware characteristics to compiler. Using this, compiler performs instruction scheduling and optimization, and then sends the information of optimization to hardware to guide the pipeline execution. Furthermore, EPIC emphasizes adaptive adjustment according to the dynamic changes of hardware resources [3].

This paper proposes a Reduced Explicitly Parallel Instruction Computing Processor (REPICP) which is a 64-bit general-purpose microprocessor. REPICP is based on EPIC design philosophy and IA-64 architecture. Compared with IA-64 architecture, REPICP is a reduced design, which removes the IA-32 engine, reduces instruction dispatch, simplifies memory hierarchy, decreases memory capacity and omits the hardware implemented Virtual Hash Page Table (VHPT). In REPICP, we propose the Optimized Lock-Step execution Model (OLSM) and instruction control pipeline method to improve the performance. The REPICP is fabricated in Artisan 0.13 μm Nominal 1P8M process with 57 M transistors. The die size of the REPICP is 100 mm^2 (10 × 10), and consumes only 12 W power when running at 300 MHz.

The rest of the paper is organized as follows. Section 2 overviews the related work. Section 3 describes REPICP architecture. Section 4 discusses details of REPICP microarchitecture. Section 5 describes the physical implementation of REPICP. In Sect. 6, the experimental results are presented and analyzed, followed by the conclusion and future work in Sect. 7.

2. Related Work

Some principles of EPIC originate from VLIW architecture, such as the long statically bundled instruction format and Non-Unit Assumed Latency (NUAL) execution model. The contemporary VLIW processors are primarily successful as embedded media processors for consumer electronic devices, including the TriMedia media processor [4] by NXP, the SHARC DSP [5] by Analog Devices,

Manuscript received April 12, 2012.
Manuscript revised August 15, 2012.
[†]The authors are with School of Computer, National University of Defense Technology, China.
a) E-mail: gjun78@sina.com
DOI: 10.1587/transinf.E96.D.9

and the STMicroelectronics ST200 family [6] based on the Lx architecture.

As a descendant of VLIW, EPIC is strongly supported by HP-Intel alliance as general-purpose high performance processors due to its simplicity and power-efficient. Itanium [7], [8] is the first generation implementation of Intel IA-64 architecture which is based on EPIC design philosophy. Tukwila [9], a four-core, eight-thread next generation Itanium, came to market in the first quarter of 2010. Intel Itanium series appear to be the widest used EPIC architecture.

Using Application Specific Integrated Circuit (ASIC) design methodology to design a high performance processor is not very practical. EPIC architecture can achieve relative higher performance with relative simpler hardware, thus is a more suitable architecture for ASIC implementation due to the limitation of ASIC. With the purpose of getting essentials of EPIC and designing a locally used low power processor, we have implemented a 64-bit general-purpose REPICP.

3. REPICP Architecture

This section presents instruction set architecture, system architecture, predication and speculation technique of REPICP.

3.1 Instruction Set Architecture

Implemented based on IA-64 architecture, the instruction set architecture (ISA) of REPICP resembles that of IA-64 [10]. Similar to VLIW, the instructions of the REPICP ISA are encoded in bundles. As can be shown in Fig. 1, each bundle is 128 bits, contains three 41-bit instruction slots and a 5-bit template field.

Figure 2 shows the format of instruction held in slot. Instruction slots in the same bundle have the degrading priorities from slot0 to slot2.

Generated by complier, the template specifies what types of execution units each instruction in the bundle requires. The template also holds the stop information associated with instruction dependence in the bundle. The compiler encodes a stop when dependence exists, and the hardware issues the instructions separately according to the stop information. Besides, the hardware can dynamically collect resource information and specify an implicit stop when resource is adjusted, which is called adaptive scalability.

3.2 System Architecture

The system architecture of REPICP includes instruction fetch unit, branch prediction unit, instruction queue, instruction dispatch unit, execution unit, instruction control unit, register files and system bus, as shown in Fig. 3.

The program is loaded by instruction fetch unit, which consists of PC (Program Counter), L1ICache, and L1ITLB. PC is up to 64-bit wide and supports both virtual and real

127 87	86 46	45 5	4 0
Slot2	Slot1	Slot0	Template
41	41	41	5

Fig. 1 Bundle format of REPICP.

40 27	26 20	19 13	12 6	5 0
Opecode	Destination register	Source register2	Source register1	Predication register

Fig. 2 Instruction format of REPICP.

addresses. Instructions of REPICP are aligned on 128-bit boundary. L1ICache is four-way set associative, 16 Kbytes, with 32-byte blocks. L1ICache supports simultaneous write and read, and single-cycle non-blocking accesses. L1ITLB, which is full associative with 32 entries and a 4-Kbyte-page size, handles the instruction address translated from virtual to real.

The branch prediction unit (BP) of REPICP supports both hardware and software branch prediction. In software branch prediction, compiler generates prediction instructions, and hardware utilizes the software prediction information to guide the hardware branch prediction unit. In hardware branch prediction, multi-level branch prediction schemes, including IP-based single cycle prediction, adaptive multi-path prediction, return address prediction and prediction adjustment based on fetched instructions, improves the prediction accuracy and reduces pipeline flush caused by branch mis-prediction.

Instruction queue holds instructions that has been fetched from L1ICache and but not yet dispatched. It acts as a separate buffer, decouples the front end from the back end, and thus makes them independent with each other. In other words, instruction fetch in the front end still works when the back end is blocked. Similarly, as long as the instruction queue is not empty, the back end continues to issue instructions even if the front end is blocked.

Instruction dispatch unit delivers instruction bundles to the execution units of the pipeline. There are 11 instruction issue ports in REPICP, corresponding to four memory units (M), two integer units (I), two floating point units (F) and three branch units (B) respectively. The dispatch window of REPICP can hold two instruction bundles, three instructions each, and thus can issues up to six instructions per cycle.

There are abundant hardware resources in REPICP, including 11 execution units: four memory units, two integer units, two floating point units and three branch units. Memory unit, consisting of L1DCache, L1DTLB, L2Cache, and Advance Load Address Table (ALAT), fulfils data access, memory management and integer ALU operations. L1DCache is 16 K bytes, with 64-byte blocks and is write through and no-write allocate on a write miss. It supports four simultaneous write and read, and single cycle non-blocking access. L1DTLB which is full associative with 32 entries and page size ranging from 4-Kbyte to 4-Gbyte,

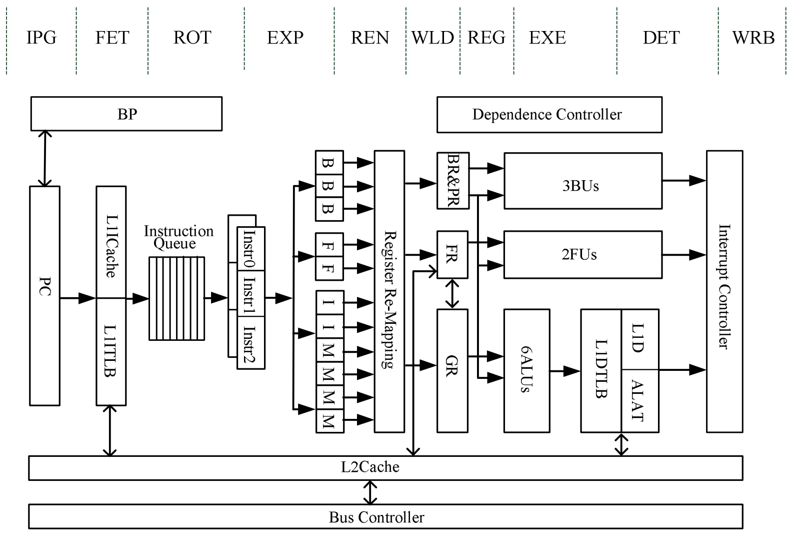


Fig. 3 System architecture of REPICP.

handles virtual to real address translation. L2Cache is a data/instruction mix cache with 128-byte blocks, and supports four requests each cycle through the multi-bank technique. L2Cache employed the four-state Modified, Exclusive, Shared, Invalid (MESI) protocol and write back policy to maintain cache coherency between multiple processors. L2Cache can handle data requests and instruction requests simultaneously. ALAT, which is an address conflicts detect table with 32 fully associative entries, is used to detect the address conflicts between speculative load and previous store, and acts as an important control mechanism for load data speculation. If a conflict is detected, then the speculative load fails. Integer unit deals with integer ALU and multimedia operations. Floating point unit supports both single and double precision floating point arithmetic and complies with IEEE754 standard. In addition, it supports 82-bit extended double precision floating point operations to meet the demand of scientific computation applications. Floating point unit also implements Single Instruction Multi Data (SIMD) operations for single precision floating point data. Branch unit is responsible for the transfer of control, and supports jump within 64-bit address space. Three branch units can work together, executing multiple parallel branch operations based on the compiler scheduling.

Instruction control unit consist of both dependence controller and interrupt controller. Dependence control in REPICP is handled by both hardware and software in a cooperative way. Software schedules the instructions to ensure there is no read-after-write (RAW) or write-after-write (WAW) dependence among instructions in the same instruction group, while hardware only concerns about the RAW and WAW dependence between different instruction groups. REPICP employs scoreboard mechanism and records the states of registers in scoreboard. If data dependence is detected, the scoreboard will stall the pipeline until the dependence is resolved. Interrupt controller handles the interrupt and detects all the exceptions from each execution unit before instruction commit, and then handles these exceptions according to the order of instructions to ensure accuracy.

According to the EPIC design philosophy, REPICP provides abundant register files, including 128 general purpose integer registers (GR), 128 floating point registers (FR), 64 predicate registers (PR), 8 branch registers (BR), 128 application registers and 82 control registers. This benefits compiler scheduling and parallelism exploiting. Moreover, REPICP implements register rotation for GR, FR and PR, facilitating software pipelining. For the call and return of function, register stack engine, which is implemented in hardware, provides software with a logically unlimited number of registers.

The system bus of REPICP is compatible with the Intel Itanium series, connecting the off-chip bridge processors. In order to improve the transfer rate, data bus is designed with double speed source synchronization signal latch protocol (2X transfer speed). Data is transmitted at the rising and falling edge of the bus clock. Selection signal (STBP#, STBN#) is transmitted at the 25% and 75% point of the bus clock. Receiver uses the selection signal to capture data. All other buses use synchronized common clock latch protocol (1X transfer speed).

3.3 Predication Technique

As an effective technique [11], predication employs a bit, called predicate, for each predication instruction. Instructions after the branch are predicated by specifying a predicate register, which indicates the legitimacy of the instruction. Therefore, control dependence is converted to data dependence, eliminating branch instructions and avoiding the extra cost of branch mis-prediction. Besides, it is more convenient for compiler to schedule instructions across branch.

REPICP defines a set of one-bit predicate registers to hold the results of branch comparing. Each predication instruction uses a predicate register to specify the execute condition. If the register value is 'true', the instruction can be executed and the states can be updated, otherwise the instruction would be treated as a NOP instruction.

3.4 Speculation Technique

In order to alleviate the long memory access delay, speculation is adopted in REPICP, including data speculation, which moves loads across stores, and control speculation, which moves loads across branches. If the speculation succeeds, access delay is concealed and performance is improved. Otherwise, recovery mechanisms are needed to ensure the correctness of programs.

REPICP provides both data speculation and control speculation. For data speculation, the compiler places a check instruction at the location of the load instruction, and then the load instruction is moved across the store instruction. Executing this load instruction will create an entry in the ALAT, which records address of the load. When the store is executed, an associative lookup against the ALAT entries is performed. If there is an entry matched, a conflict is found and an entry matching is cleared. When check instruction is executed, an associative lookup against the ALAT entries is also performed. If an entry matching exists, the data speculation is marked as success. Otherwise, the load speculation fails, and non-speculative load or recovery codes are needed.

Control speculation differs from data speculation slightly, with a series of speculative instructions executed, which may not need to be executed in fact. If the instructions need not to be executed, just discard the corresponding results. Similarly, if exception occurs when executing such instructions, the exception should be ignored. Deferring exception handle for control speculation instructions till the confirmation of speculative instructions for exception is considered to be effective [12]. Deferring exception is supported by adding an extra bit in registers, called Not a Thing (NaT) bit. To deal with control speculation, a check instruction is placed at the location of the load instruction. If an exception arises at the time of executing the speculative load instruction, then the extra bit NaT of the destination register is set and the exception is deferred. If the load instruction needs not to be executed, the exception is ignored. If the load instruction needs to be executed, then the check instruction determines whether the NaT bit is set. If the NaT bit is not set, the control speculation succeeds. Otherwise, recovery jumped from the check instruction is needed.

4. REPICP Microarchitecture

REPICP is a reduced design based on EPIC architecture. This section presents the details of the REPICP microarchitecture, including pipeline, execution mode, logic reduction and optimization.

4.1 Pipeline

Figure 3 shows the ten-stage pipeline of REPICP, including IPG (Instruction Pointer Generate), FET (Instruction Fetch), ROT (Instruction Rotation), EXP (Instruction Expand), REN (Register Rename), WLD (Word Line Decode), REG (Register File), EXE (Instruction Execution), DET (Deferred Exception Detected) and WRB (Write Back). The function of each stage is introduced as follow:

In the IPG stage, an IP-select multiplexer determines the proper IP for ICache access. Two instruction bundles can be fetched each cycle. Odd bundle address should be aligned to even address, therefore only the useful bundle is fetched, and others are abandoned.

In the FET stage, ICache is accessed simultaneously with ITLB access and tag comparison. On an ICache miss, the instruction bundles are filled from L2 Cache through the bypass logic.

After the FET stage, instructions and the pre-decode bits are inserted into the instruction buffer. Instruction buffer can hold up to eight bundles. If the instruction buffer is empty, the instructions and the pre-decode bits are sent directly to the instruction window, which has a capacity of two instruction bundles, including six instructions. If the instruction buffer is full, a signal is generated to indicate the overflow and instruction fetch stops. Instructions alignment will be performed in the ROT stage according to the EXP instruction window information collected per cycle. This is a guarantee of issuing two bundles per cycle.

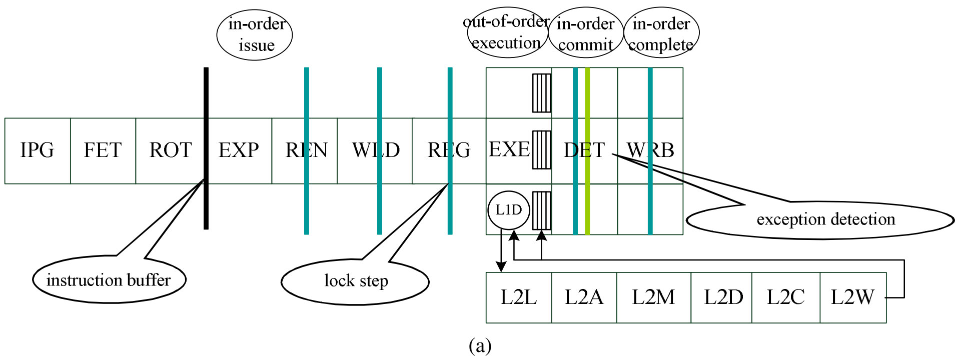
The two bundles templates in the current instruction window are decoded in the EXP stage. Instructions are dispatched to the corresponding ports according to the templates field and dispatch rules, as long as there is no stall or not short of resources. Compiler sets the template field, which indicates the execution units and the corresponding stall bits. Information of hardware resources was detected dynamically by hardware.

In the REN stage, re-mapping of logical register to physical register is performed. In REPICP, only physical register indicates the actual register file that can be accessed. Furthermore, register renaming is implemented for both register stack and rotation in REPICP.

The word line decoding of register files is implemented in the WLD stage. The REG stage performs register file access, source operands read, data dependence check and scoreboard update.

In the EXE stage, instructions are executed in functional units, which are fully pipelined. Therefore, there is always a new instruction ready to be executed every cycle without stall.

The DET stage is responsible for exception detection and handle. When an exception happens, the pipeline would be flushed. In addition, the DET stage is the second execution stage of the integer pipeline for multimedia instructions, and is also a stage of the memory pipeline, responsible for updating the ALAT table. Branch pipeline checks



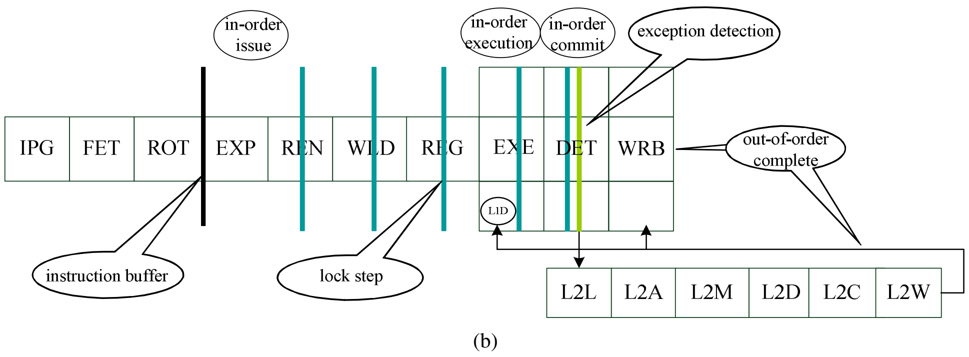


Fig. 4 (a) LSM of EPIC. (b) OLSM of REPICP.

the branch prediction in the DET stage. If the branch prediction is false, the pipeline would be flushed and the new instruction pointer would be sent to the IPG stage. In the WRB stage, instruction retirement and architecture state update are performed.

4.2 Optimized Lock-Step Execution Model

In EPIC, any instruction i_j can be issued if all previous instructions have been issued or issued simultaneously with i_j. Instructions that can be issued simultaneously are treated as an executing group, which is specified by the compiler. The issue mode of EPIC, in essence, is in-order issuing, which enhances the instruction-level in-order issue to group-level. In REPICP, each group-level issues up to six instructions per cycle. And such group-based in-order issuing mode achieves high instruction issue performance, in respect that 11 execution units in REPICP are fully pipelined.

The basic EPIC structure adopts Lock-Step Model (LSM), in which multiple instructions in the same issue group move forward with synchronization before entering into the EXE stage and instructions in different issue groups don't intersect before entering into the EXE stage and remain the strict order. However, after the issue group enters into the EXE stage, due to the uncertainty of the memory operation latency, the commit time of the instruction arriving at the DET stage in the same issue group is also uncertain. In order to maintain the correction of the program, each pipeline needs a delay buffer to write results. As soon as the results of out-of-order executing are available, it will write results into the delay buffer. Once all instructions in the same issue group execute completely, they can be committed simultaneously. The DET stage of all function pipeline need to adopt a delay buffer queue with the same depth to buffer the results. Figure 4 (a) shows the in-order issuing, out-of-order executing, in-order committing and in-order completing in LSM of EPIC. The memory operation is responsible for accessing the L1D Cache at the EXE stage. If the accessing is missing, it needs to access the L2 Cache or main memory, and then fill the L1D Cache and the delay buffer. After all data of the same issue group in the delay buffer are available, it will enter into the DET stage to commit instructions in order and support accurate interruption, and finally write back the results in order.

The depth of the delay buffer queue in the basic EPIC structure will affect the performance directly. The deeper the depth of the buffer queue is, the higher the potential Memory-Level Parallelism (MLP) is and the higher performance the system can achieve, but the required hardware cost is larger. In addition, because of the introduced buffer, each entry in the buffer may become the register of the results, which will introduce large cost of dependent control logic and data bypass logic and make the latency of the circuit uncontrollable.

In fact, most of operations in EPIC can finish the calculation at the EXE stage. Although a few memory operations cannot finish all calculations at the EXE stage, whether an exception occurring or not can be confirmed before entering into the DET stage. Based on this feature, we propose an Optimized Lock-Step execution Model (OLSM) [13]. In OLSM, the time point of the instruction commit and the data write-back is divided. When an instruction is committed in the DET stage, it only checks whether the instruction causes an exception or not and do not check whether the data have been written back or not. Figure 4 (b) shows the in-order issuing, in-order executing, in-order committing and out-of-order completing in OLSM of REPICP. The stages before DET work under in-order mode. The memory operation finishes accessing the L1D Cache at the EXE stage. If the access of the L1D Cache hits, the L1D Cache can provide data directly. If the access of the L1D Cache misses, the L1D Cache does not access L2 Cache directly. After the instructions in the same issue group enter into the DET stage to commit in order, the control pipeline will control whether to access the L2 Cache or not. If exceptions occur, it doesn't need to access the L2 Cache, otherwise the L2 Cache will be taken as the data provider to write back data and fill the L1D Cache.

The in-order committing and out-of-order completing of OLSM will utilize a great number of computing and memory resources to hide memory latency and improve MLP significantly. The OLSM model removes the mechanism in basic EPIC model that developing MLP is depending on the buffer to improve performance. Removing the buffer can reduce the hardware cost and make the control simple. OLSM of REPICP is proved to be effective, not only achieving higher performance but also making hardware design simpler [13].

4.3 Instruction Control Pipeline

There are 11 pipeline stages in REPICP, and each instruction in the pipeline needs to know its own bundle IP to support precise interrupt or execute data transfer from bundle IP to register files. If each pipeline stage contains the 64-bit bundle IP, there are a large number of pipeline registers. Because the instruction dispatch window contains two bundles and the dispatch logic always splits the bundle when encountering branch instructions, it can be inferred that the simultaneous issue instructions have continuous IP. Therefore, we propose an instruction control pipeline method [14] based on OLSM, which carries the common information of the 11 pipeline stages.

The instruction control pipeline is an individual pipeline including IREN, IWLD, IREG, IEXE and IDET, which can share a front pipeline with 11 execute pipelines. The stage in the instruction control pipeline is correspond with the stage in the execute pipeline and moves forward in lock step with other pipelines. The instruction dispatch module dispatches instruction to the execute pipeline at the meanwhile dispatches the common information, such as IP address, to instruction control pipeline. Figure 5 shows the structure of the instruction control pipeline. The real line and dashed line represent the data path and control path between the instruction control pipeline and the execute pipeline respectively. The data path implements the interaction of common information and the control path implements the lock step execution. Comparing with the traditional method that each pipeline takes the common in-

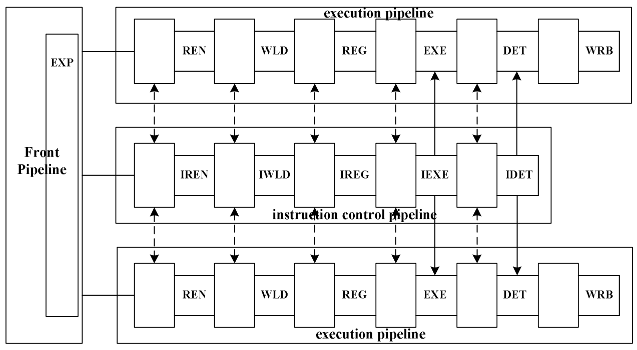


Fig. 5 Instruction control pipeline.

formation independently, using the instruction pipeline will reduce the utilization of the stage registers significantly. In REPICP, the required stage registers in the instruction pipeline are only 1/11 of the traditional method. For the IP address, the reduced number of flip-flop cells is $5 \times 10 \times 64$, where 5 is the number of pipeline stages, 10 is the number of execute pipeline stages minus 1 and 64 is the bit-width of the IP address. Except the reduced stage registers, using the instruction pipeline can still increase the signal locality, which will reduce the interaction between the instruction control system and the execution pipeline.

4.4 Logic Reduction and Optimization

REPICP is designed with Verilog language. We propose many innovative methods to reduce and optimize the design. REPICP is a reduced design based on IA-64 architecture, which removes the IA-32 engine, reduces instruction dispatch, simplifies memory hierarchy from IA-64 three-level to REPICP two-level, decreases memory capacity from 256 KB L2Cache to 128 KB L2Cache and omits the hardware implemented Virtual Hash Page Table (VHPT).

In REPICP, there are 11 instruction dispatch ports and 6 issue slots with 41-bit long instructions. Considering the routing and timing of the large cross network caused by the arbitrary instruction issue, some restrictions are employed by IA-64 compiler to simplify the dispatch network, allocating no more than 3 instruction types for each dispatch port through only supporting 24 kinds of limited instruction combinations which are defined by IA-64 instruction templates. Besides it, the asymmetry design strategy of multi same type execution units is adopted in REPICP. In multi execution units of the same type, one executes the entire instructions and others execute the simplified common instructions. In order not to affect the performance, the dispatch units are designed to use pre-decoded main op-codes, and prefer to dispatch the more districted instructions of the same type firstly. Performance degradation due to asymmetry design strategy is trivial, but chip timing and area are improved.

There are four memory units in REPICP, using single-port implementing multi-ports data cache to support four simultaneous memory requests. REPICP adopts the

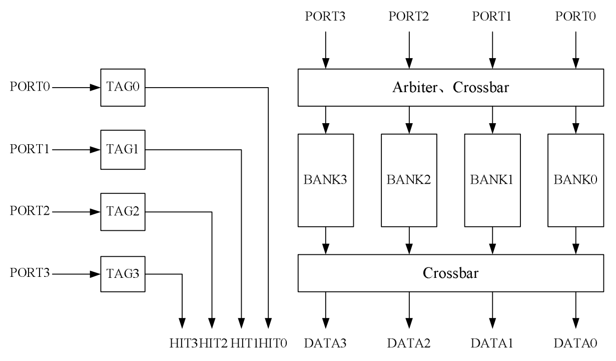


Fig. 6 DTD multi-ports data cache.

Different Tag Data (DTD) design [15] for 16 K L1DCache with 64-byte blocks. As depicted by Fig. 6, cache is made up of data array and tag array. DTD design duplicates resources, and produces four tag array copies, with each copy corresponding to an access port. Therefore, the four pipelines don't stall for tag array access port conflict. DTD adopts the cross multi-bank design for data array. Each bank holds a 16 B of 64 B-sized block of L1DCache and the two low bits of block address are crossed. Since only the different access requests to the same bank in a cycle raise conflict, REPICP compiler can schedule instructions to avoid the conflict. Conflicts that cannot be statically determined, is detected at runtime by the hardware-implemented logic. If bank conflict is detected, it is handled in the order of priority. The hardware implementation of static priority without starvation is simple.

Exceptions are detected and arbitrated in the DET stage. Branch exceptions can only be determined until the branch predicate has been accessed in the DET stage, thus the combinational path is rather long. Therefore, branch stall mechanism is used in the DET stage of REPICP, which means that the pipeline is halted one cycle for branch so as to access the predicate first, and then performs exceptions arbitration in the next cycle. As a result, the process of branch exceptions is divided into 2 cycles, reducing the logic each cycle and improving the chip frequency.

Multi-ports register files are implemented in REPICP in order to support parallel execution of multi-function units. For example, 128 65-bit general-purpose registers support up to 12 reads and 8 writes per cycle. However, large multi-ports register files cause the problem of long access delay and restrict frequency. To overcome it, WLD stage is added in REPICP pipeline, decoding the registers and dividing the register access into two stages.

5. Physical Implementation

REPICP is designed with the ASIC design flow. We use Design Compile of Synopsys Company for synthesis, SoC Encounter of Cadence Company for floorplan, place and route, Celtic of Cadence Company for crosstalk analysis, Fireice of Cadence Company for RC extraction, PrimeTime of Synopsys Company for static timing analysis, and Calibre of Mentor Company for physical verification. Figure 7 shows the layouts of REPICP.

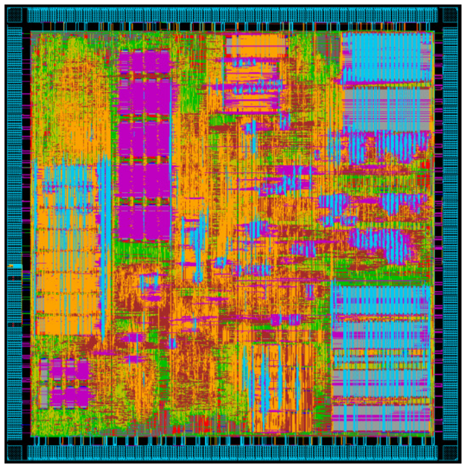


Fig. 7 Layout for REPICP.

Table 1 Technical parameters of REPICP.

Word Length	64 bits
Issue Width	6 instruction/cycle
Primary Cache	16KB L1Icache+16KB L1Dcache
Secondary Cache	128KB L2Cache
Frequency	300MHz@1.2V
Power	12W@1.2V
Process Technology	0.13μm CMOS
Nominal Voltage	1.2V
Number of Metal Layers	8
Die Size	100mm^2（10×10）
Transistor Count	57Mega
Package	HPBGA
Number of Pins	696

Despite inserting WLD stage in logic design, large multi-ports register files of REPICP for parallel execution of multi function units, which needs 12 reads and 8 writes, are still in the critical path. Therefore, in addition to the standard ASIC design flow, harden design of large multi-ports register files is implemented to shorten the critical path more. By manual half-customized placement and routing, the access delay of large multi-ports register files is reduced more than 1/3 and the area of register files is also decreased.

REPICP is implemented with a standard-cell in Artisan 0.13 μm Nominal 1P8M process, and die size is 100 mm^2 (10 × 10), its power is only 12 W when running at frequency of 300 MHz. The chip integrates 57 Mega transistors and is packaged into HPBGA with 696 pins. Table 1 lists the main technical parameters of REPICP.

6. Performance Analysis

6.1 Reduced Structure Performance Analysis

The design of instruction dispatch and memory system in

REPICP is reduced in consideration of chip timing and chip area. We simulate the performance of reduced strategy to ensure its rationality. The simulation is based on the Palladium hardware accelerator of Cadence Company, running both reduced and unreduced RTL design to test the execution time of programs. We simulated three specint2000 and three specfp2000 benchmarks. The emulation frequency of Palladium is 300 KHz.

Figure 8 shows the performance comparison of Non-Restricted Instruction Dispatch and Restricted Instruction Dispatch. The Non-Restricted Instruction Dispatch adopts symmetric strategy of execution units which execution units of the same type are designed the same to execute the entire instructions. The Restricted Instruction Dispatch adopts asymmetric strategy of execution units. One execution unit with the same type implements a full version and others implement a simple version. The full version supports all instructions for the unit and the simple version only implements several common instructions. For the Restricted Instruction Dispatch, in order to avoid the negative effect of the asymmetric strategy, we add a priority dispatch rule, in which the main operation code of the instruction is pre-decoded, the simultaneous issued instruction is adjusted dynamically and the limited instruction is dispatched with a higher priority. Through the extra dispatch restricted rules introduced by Restricted Instruction Dispatch, the programs execution time differs little. The area of the execution unit can be reduced and the performance of the processor can be improved. In REPICP, one of two I units and two of three B units adopt the simplified design strategy. The areas of one simplified I unit and B unit are 1/3 and 1/4 of the full version I unit and B unit respectively.

Figure 9 shows the performance comparison between real four-ports and virtual four-port L1DCache, which also differ little in programs execution time. The virtual four-ports L1DCache is implemented with the memory bank technique of DTD, and each bank holds 16 B of the 64 B-sized block of L1DCache. The real four-ports L1DCache is implemented with the technique of duplicate data arrays, and four data arrays support four simultaneous memory requests. Compared with real four-ports L1DCache, DTD designed virtual four-ports L1DCache exists bank conflict. However, this bank conflict can be avoided effectively by instruction scheduling. Thus, Compared with real four-ports L1DCache, DTD designed with virtual four-port L1DCache decreases 3 data array area, which is 3 × 16 KB, with almost the same performance of the real four-ports L1DCache.

Due to the physical implementation restriction, REPICP employs the reduced two-level memory hierarchy, decreasing the capacity of L2Cache from 256 KB to 128 KB. Figure 10 shows the performance of reduced and non-reduced memory system, with large distinctions in programs execution time. Gzip, swim and equake access memory a lot. Swim and equake equal to almost two to four times of that of crafty and mesa, and gzip equals to even four to eight times memory access. All these programs are typical memory-bound programs, whose performances are seriously affected by the reduction of memory system. Twolf and mesa don't have much memory access. The reduction of memory system has little impact on the execution time. Compared with twolf, mesa accesses memory more irregularly. Therefore, as L2Cache decreases from 256 KB to 128 KB, replace conflict increases, resulting in longer program execution time, which is unconspicuous. Memory access of crafty is relatively low and regular, however, it is seriously influenced by the capacity of memory. Therefore, crafty is the most affected program by the reduction of the memory system. When L2Cache decreases from 256 KB to 128 KB, crafty has two to three more times of access miss, consequently its execution time increases much more than other program. The design of REPICP memory system is limited by chip physical implementation. Though reduced memory system design increases the programs execution time and affects the performance, it indeed saves the chip area and improves chip timing.

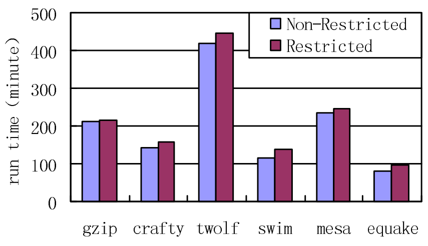


Fig. 8 Instruction Dispatch comparison.

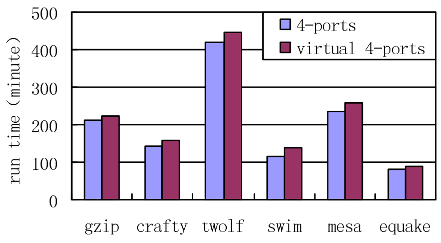


Fig. 9 L1DCache comparison.

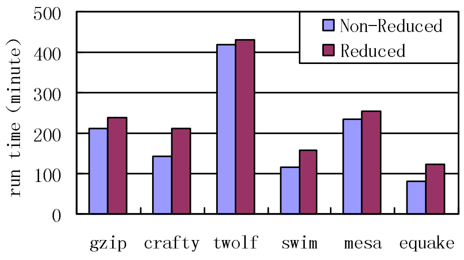


Fig. 10 Memory System comparison.

6.2 Chip Performance Analysis

A special test board is designed for REPICP to verify chip function and performance. The test system is made up of two REPICPs, a north bridge implemented in FPGA, a spe-

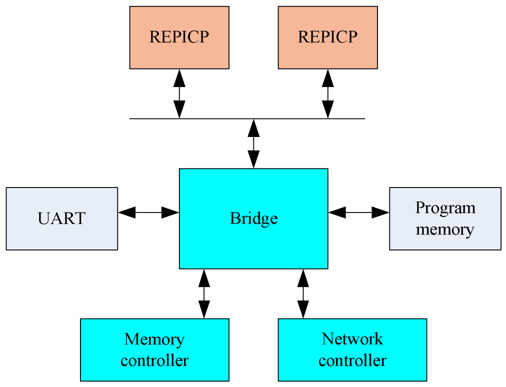


Fig. 11 Structure of the test system.

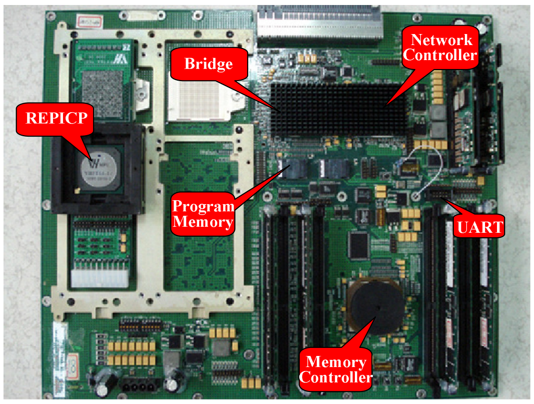


Fig. 12 REPICP-based test board.

cial network controller, a memory controller, a program memory and a uart interface. Figure 11 shows the structure of test system. Figure 12 shows REPICP-based test board. On the test board, Reduced Linux OS and various applications can successfully run on REPICP.

Based on the test board shown by Fig. 12, we test the REPICP chip performance by using spec2000 benchmarks. Figure 13 is the experimental results of specint2000. Figure 14 is the experimental results of specfp2000. An 800 MHz Itanium server (Dell WorkStation 730) is used as the baseline for comparison [16]. The Itanium processor has a 16 KB L1ICache, a 16 KB L1DCache and a 96 KB L2Cache, and uses the Intel C++ and Fortran Compiler 5.0.1. Our REPICP has the same configuration for L1ICache and but larger L2Cache (128 KB) and uses the Intel C/C++ and Fortran Compiler 9.0.18. Figure 13 and Fig. 14 show the execution time difference between REPICP, denoted by REPICP curve, and the baseline Itanium, denoted by Itanium curve. The X axis represents different benchmarks used. The Y axis stands for the execution time. As shown by the Fig. 13 and Fig. 14, the execution time of REPICP is proportional to that of the baseline Itanium at most points, indicating that though reduced, REPICP is able to maintain performance feature of Itanium while running some workloads. But compared with baseline Itanium, the execution time of REPICP is quite long.

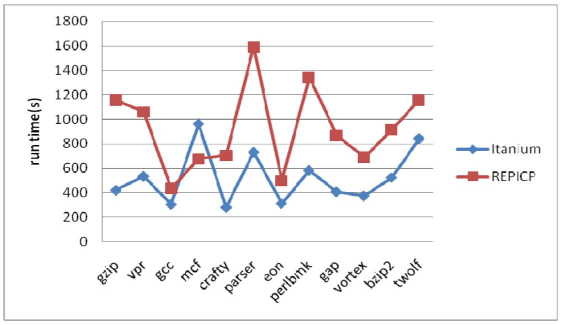


Fig. 13 SPECint2000 performance comparison.

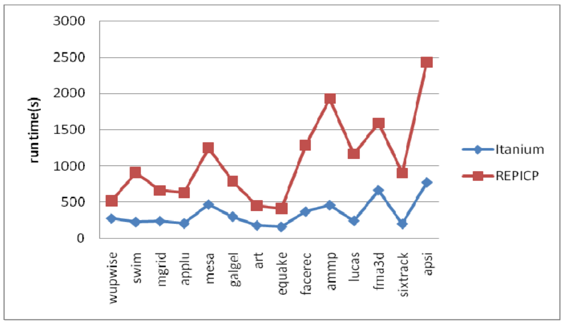


Fig. 14 SPECfp2000 performance comparison.

Three reasons may cause this drawback of REPICP. First of all, REPICP is implemented with ASIC, rather than full-custom, which limited its frequency to 300 MHz. Secondly, to simplify the test, we use a self-designed 100 MHz north bridge implemented with FPGA on the test board, with limited off-chip memory bandwidth. Thirdly, to save area, we cut hardware implemented VHPT in REPICP.

7. Conclusions

REPICP is an independently designed 64-bit general-purpose processor, supporting the main OS. The article describes architecture, implementation and performance of REPICP. REPICP is an IA-64 based processor that exploits EPIC philosophy. We have grasped IA-64 ISA and EPIC architecture design techniques in the implementation of REPICP. EPIC architecture explores ILP of programs. However, inherent ILP of programs is limited, Thread Level Parallelism (TLP) is explored at a higher level. In future, we will focus on the combination of EPIC, multi-thread, multi-core in the corresponding processor design.

Acknowledgments

The authors acknowledge the contributions from the entire REPICP research team. This work is partially supported by national 863 plans projects under Grant No. 2009AA01Z102 and by natural science foundation of china under Grant No. 60970036, 60873016, 61170083, 61170045.

References

[1] M. Zhang and Y. Wang, High-performance microprocessors: techniques and architectures (in Chinese). Changsha: NUDT press, 2004.

[2] J.L. Hennessy and D.A. Patterson, Computer Architecture: A Quantitative Approach, pp.182–315, 3rd Edition, Morgan Kaufmann, 2002.

[3] M.S. Schlansker and B.R. Rau, "EPIC: Explicitly parallel instruction computing," Computer, vol.33, no.2, pp.37–45, 2002.

[4] G. Slavenburg, S. Rathnam, and H. Dijkstra, "The TriMedia TM-1 PCI VLIW media processor," Hot Chips 8, Aug. 1996.

[5] http://www.analog.com, SHARC Processor Architectural Overview: Super Harvard Architecture.

[6] P. Faraboschi, G. Brown, J.A. Fisher, G. Desoli, and F. Homewood, Lx: a technology platform for customizable VLIW embedded processing, SIGARCH Comput. Archit. News 28, 2, pp.203–213, May 2000.

[7] H. Sharangpani and K. Arora, "Itanium processor microarchitecture," IEEE Micro, vol.20, no.5, pp.24–43, Sept.-Oct. 2000.

[8] S. Naffziger, "The implementation of the Itanium 2 microprocessor," IEEE J. Solid-State Circuits, vol.37, no.11, pp.1448–1460, Nov. 2002.

[9] B. Stackhouse, S. Bhimji, C. Bostak, D. Bradley, B. Cherkauer, J. Desai, E. Francom, M. Gowan, P. Gronowski, D. Krueger, D. Morganti, and S. Troyer, "A 65 nm 2-billion transistor quad-core itanium processor," IEEE J. Solid-State Circuits, vol.44, no.1, pp.18–31, Jan. 2009.

[10] Intel® IA-64 Architecture Software Developer's Manual vol.3: Instruction Set Reference, July 2000.

[11] D.I. August, Systematic Compilation for Predicated Execution, Ph.D. thesis, University of Illinois at Urbana-Champaign, 2000.

[12] Y. Wang and M. Zhang, "A compiler and architecture combined mechanism for speculative data load," J. Computer Research and Development, 39 (Suppl), pp.220–224, 2002. (in Chinese)

[13] R. Deng, H.-Y. Chen, Z.-C. Xing, L.-G. Xie, and X.-J. Zeng, "The research on memory-level parallelism execution model in EPIC architecture," Chinese Journal of Computers, vol.1, pp.74–80, 2007. (In Chinese)

[14] J. Jiang, "Research and implementation on instruction control pipeline in general-purpose EPIC microprocessor," J. Chinese Computer Systems, vol.9, pp.1661–1664, 2006. (in Chinese)

[15] Y. Wang and M. Zhang, "On low power multi-port data cache design for multi-issue processors," Proc. 13th National Conference on Information Storage Technology, Xian, pp.326–331, 2004. (in Chinese)

[16] http://www.spec.org

Jun Gao received his B.S. degree in computer science and technology from National University of Defense Technology, Changsha, China, in 2002. His main research interests include high-performance computer architecture, micro-processor design and VLSI design.

Minxuan Zhang received the M.Sc. degree in computer science and technology from National University of Defense Technology, Changsha, China, in 1987. He is currently a Professor at School of Computer, National University of Defense Technology. His research interests include high-performance computer architecture, micro-processor design and VLSI design.

Zuocheng Xing is professor at College of computer, National University of Defense Technology. He received his B.S. degree in computer science and technology from National University of Defense Technology in 1987. His main research interests include high performance computer system, micro-processor architecture and micro-electronics design.

Chaochao Feng received the B.Sc. and M.Sc. degrees in computer science and electronic science from National University of Defense Technology, Changsha, China in 2005 and 2007 respectively, where he is currently working toward the Ph.D. degree in electronics science and technology. His current research interests include Network-on-Chip, high-performance microprocessor design and VLSI design.