FPGA BASED 128-BIT CUSTOMISED VLIW PROCESSOR FOR EXECUTING DUAL SCALAR/VECTOR INSTRUCTIONS

Rekha Halkatti[1], **Veeresh Pujari**[2]

[1]*Dept of MTECH VLSI and EMBEDDED SYSTEM, VTU Regional centre, Gulbarga*
[2]*Dept of MTECH VLSI and EMBEDDED SYSTEM, VTU Regional centre, Gulbarga*

Abstract

This paper proposes new processor architecture for accelerating data-parallel applications based on the combination of VLIW and vector processing paradigms. It uses VLIW architecture for processing multiple independent scalar instructions concurrently on parallel execution units. Data parallelism is expressed by vector ISA and processed on the same parallel execution units of the VLIW architecture. The proposed processor, which is called VLIW, has unified register file of 64x32-bit registers in the decode stage for storing scalar/vector data. VLIW can issue up to four scalar/vector operations in each cycle for parallel processing a set of operands and producing up to four results. However, it cannot issue more than one memory operation at a time, which loads/stores 128-bit scalar/vector data from/to data cache.. The complete design of our proposed VLIW processor is implemented using Verilog. our proposed VLIW processor is implemented using Verilog targeting the Xilinx FPGA Virtex-5, XC5VLX110T-3FF1136 device. The required numbers of slice registers and LUTs are 20292 and 24214 out of 28800 respectively.

Keywords*—VLIW architecture; vector processing; data-level parallelism; FPGA/Verilog implementation*

--***--

1. INTRODUCTION

Embedded systems have become common place nowadays and they are being utilized for many different applications such as image processing, computer vision, networking, wireless communication, etc. Because these applications offer a good amount of functional and data level parallelism, they can achieve better performance when run on multiprocessor systems rather than on uni-processor systems.

Further, to exploit instruction level parallelism, very long instruction word (VLIW) processors can be utilized to increase the performance beyond the single issue or reduced instruction set computer (RISC) architectures [1].

While RISC architectures only take advantage of temporal parallelism (by using pipelining), VLIW architectures can additionally take advantage of the spatial parallelism by using multiple functional units (FUs) to execute several operations simultaneously. VLIW multiprocessor systems (where each processor is a VLIW processor) can support both instruction level as well as data level parallelism.

RISC architectures are simpler, cheaper and achieve high-performance than CISCs; VLIW architectures require compiler support which performs most of the operations which was doing by hardware part in RISC. This reduction in hardware makes the VLIW simpler and cheaper than RISCs [3].

Superscalar means the ability to fetch, issue to execution units, and complete more than one instruction at a time [4, 5].. Superscalar implementations are required when architectural compatibility must be preserved.

Two types of processors have become core for the processing platforms. The first type is RISC processors that have been used. They are flexible in the sense that they can be easily reprogrammed to support different applications, but they have several disadvantages.

1) There is a lot of control overhead to correctly sequence the code execution on these processors leading to wasted power consumption.
2) To increase performance complex instruction fetch and decode mechanism are needed and in turn adding more to the power consumption.
3) In order to make them more power efficient, new instructions are commonly introduced, but this requires a large amount of effort in adapting the existing tools and compilers to take full advantage of these instructions.

The second type are VLIW processors that have gained a grip in embedded systems as they depend on compilers to schedule instruction execution and thereby overcoming the first and second disadvantages of RISC processors, as a result there is much more power efficient designs.

Field-programmable gate arrays (FPGAs) have become a widely used tool, providing software like flexibility and hardware-like performance. For an application to take advantage of performance improvement from FPGA implementations, it must possess inherent parallelism. Applications in different domains such as multimedia, bioinformatics, wireless communication, numerical analysis, etc. contain a high level of instruction level parallelism (ILP), as they have many independent repetitive calculations.

By issuing multiple operations in one instruction, a VLIW processor is able to accelerate an application many times compared to a RISC system [1][2]. This is further enhanced by the fact that VLIW processors are simpler in design when compared to their more complex (out-of-order) RISC counterparts.

2. BLOCK DIAGRAM OF GENERIC VLIW PROCESSOR

VLIW architectures offer high performance at a much lower cost than dynamic out-of-order superscalar processors. By allowing the compiler to directly schedule machine resource usage, the need for expensive instruction issue logic is obviated. Furthermore, while the enormous complexity of superscalar issue logic limits the number of instructions that can be issued simultaneously, VLIW machines can be built with a large number of functional units allowing a much higher degree of instruction-level parallelism (ILP). VLIW instructions indicate several independent operations. Instead of using hardware for parallelism, VLIW processors use compiler that generates the VLIW code to clearly specify parallelism.

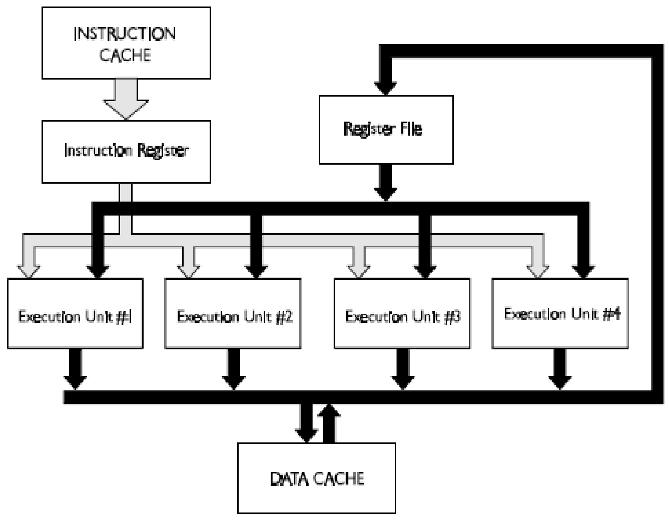

Fig 1 Block diagram of generic VLIW implementation

In VLIW complexity of hardware is moved to software. This trade-off has a benefit: only once the complexity is paid when the compiler is written instead of every time a chip is fabricated. Smaller chip, which leads to increased profits for the microprocessor vendor and/or cheaper prices for the customers. It's easier to deal Complexity with in a software design than in a hardware design. Thus, the chip may cost less to design, be quicker to design, and may require less debugging, all of which are factors that can make the design cheaper. Improvements to the compiler can be made after chips have been fabricated; improvements to superscalar dispatch hardware require changes to the microprocessor, which naturally incurs all the expenses of turning a chip design.

VLIW instruction format encodes an operation for every execution unit. This shows that every instruction will always have something useful for every execution unit. Unfortunately it's not possible to pack every instruction with work for all execution units. Also, in a VLIW machine that has both integer and floating-point execution units, the best compiler would not be able to keep the floating point units busy during the execution of an integer-only application.

The problem with some VLIW instructions is that they do not make full use of all execution units which results in waste of precious processor resources like waste of instruction memory space, instruction cache space, and bus bandwidth.

There are two solutions to reduce the waste of resources.

1) Instructions can be compressed with a more highly-encoded representation. Different techniques, such as Huffman encoding can be employed to allocate the fewest bits to the most frequently used operations.
2) To define an instruction word that encodes fewer operations than the number of available execution units.

2.1 Generic Architecture of VLIW Processor

Each 128-bit VLIW instruction word consists of two operations. The architecture is built such that two operations cane be executed in parallel to maximize the performance ability. Each operation uses a register file. Register file consist a set of sixteen internal registers each are of 64-bit.

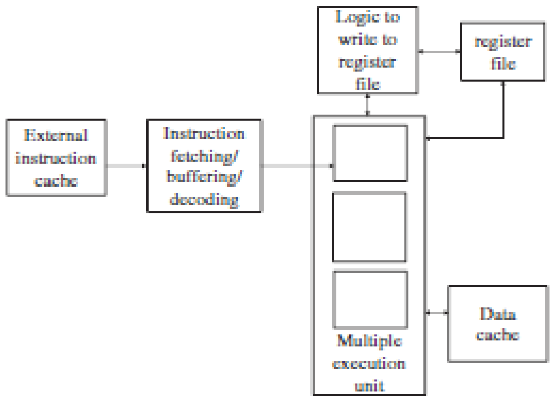
Logic to write to register file
register file
External instruction cache
Instruction fetching/ buffering/ decoding
Multiple execution unit
Data cache

Fig 2 Block diagram of VLIW architecture

On multiple execution units, this paper proposes new processor architecture for accelerating data-parallel applications by the combination of VLIW and vector processing paradigms. It is based on VLIW architecture for processing multiple scalar instructions concurrently. Moreover, data-level parallelism (DLP) is expressed efficiently using vector instructions and processed on the same parallel execution units of the VLIW architecture. Thus, the proposed processor, which is called, exploits ILP using VLIW instructions and DLP using vector instructions. The use of vector instruction set architecture (ISA) lead to expressing programs in a more concise and efficient way (high semantic), encoding parallelism explicitly in each vector instruction, and using simple design techniques (heavy pipelining and functional unit replication) that achieve high performance at low cost . Thus, vector processors remain the most effective way to exploit data-parallel applications

Each operation is divided into four stages:

1) Fetch stage
2) Decode stage
3) Execute stage and
4) Write back stage.

Fetch stage: The next instruction is fetched from the memory address that is currently stored in the program counter (PC), and stored in the instruction register (IR). At the end of the fetch operation, the PC points to the next instruction that will be read at the next cycle.

Decode stage: interprets the instruction. During this cycle the instruction inside the IR (instruction register) gets decoded.

Execute stage: The control unit of the CPU passes the decoded information as a sequence of control signals to the relevant function units of the CPU to perform the actions required by the instruction such as reading values from registers, passing them to the ALU to perform mathematical or logic functions on them, and writing the result back to a register. If the ALU is involved, it sends a condition signal back to the CU.

Write back stage: The result generated by the operation is stored in the main memory, or sent to an output device. Based on the condition of any feedback from the ALU, Program Counter may be updated to a different address from which the next instruction will be fetched

During the decode stage, data are read from the register file and during write back stage, data are written into the register file. Based on these requirements, the VLIW microprocessor is implemented. The incoming instructions and data from external systems to the VLIW microprocessor are fetched by the fetch unit.

After the instruction and data have been fetched, it is given to the decode stage. The 128-bit instruction consists of two operations. Each operation is given to the corresponding decode stage. Each operation is also passed from the fetch stage to the register file to allow the data to be read from the register file for each corresponding operation.

In the decode stage, the operations are decoded and passed onto the execute stage. The execute stage, as its name implies, will execute the corresponding decoded operation. The execute stage has access to the shared register file for reading of data during execution. Upon completion of execution of an operation, the final stage (write back stage) will write the results of the operation into the register file, or read data to the output of the VLIW microprocessor for read operation.

2.2 Top Level Architecture

Instructions and data are fetched using an external instruction memory that has its own instruction cache. The defined VLIW microprocessor loads instructions and data directly from the external instruction memory through the 6-bit bus interface word and the128-bit bus interface data. The output interface signal jump from the VLIW microprocessor is feedback as an input to the external instruction memory as an indicator that a branch has been taken and the instruction memory needs to pass another portion of instructions and data to the VLIW microprocessor. The top level architecture of VLIW processor is shown in the below figure.

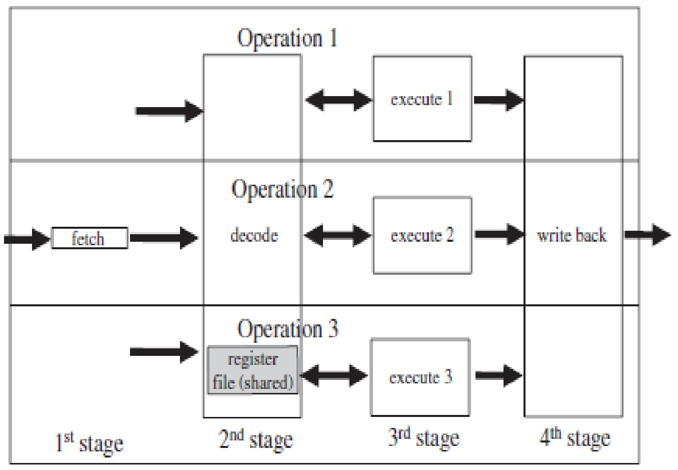

Fig 3 Top level architecture of VLIW

3. IMPLEMENTATION RESULTS

VLIW processor is implemented using Verilog targeting the Xilinx FPGA Virtex-5 device. A single Virtex-5 configurable logic blocks (CLB) comprises two slices, with each containing four 6-input LUTs and four flip-flops, for a total of eight 6-LUTs and eight flip-flops per CLB. Virtex-5 logic cell ratings reflect the increased logic capacity offered by the new 6-input

LUT architecture. The Virtex-5 family is the first FPGA platform to offer a real 6-input LUT with fully independent (not shared) inputs. By properly loading LUT, any 6-input combinational function can be implemented. Moreover, the LUT can also be Configured as a 64×1 or 32×2 distributed RAM. Besides, a single LUT supports a 32-bit shift register. See [6-9]

Table 1 Synthesis result

PARAMETER	**VIRTEX-5(XC5VLX50)**
No. of Slices	20292 out of 28800(70%)
No. of slice LUTs	24214 out of 2880 (84%)
No. of bonded IOBs	291 out of 440(66%)
Minimum Period	4.073ns
Maximum Frequency	245.531MHz

4. CONCLUSIONS

This paper proposes new processor architecture called VLIW for accelerating data-parallel applications. VLIW executes multi-scalar and vector instructions on the same parallel execution data path. VLIW has a modified five-stage pipeline for fetching 128-bit VLIW instruction (four individual instructions), decoding/reading operands of the four instructions packed in VLIW, executing four operations on parallel execution units, loading/storing 128- bit (4×32-bit scalar/vector) data from/to data memory, and writing back 4×32-bit scalar/vector results. Moreover, this paper presents the FPGA implementation of our proposed VLIW.

REFERENCES

[1] P. Faraboschi, G. Brown, J.A. Fisher, G. Desoli, and F. Homewood, "Lx: A Technology Platform for Customizable VLIW Embedded Processing", in Proceedings of the 27th Annual International Symposium of Computer Architecture (ISCA 00), pp. 203 - 213, 2000.

[2] S. Wong, T.V. As, and G. Brown, "ρ-VEX: A Reconfigurable and Extensible Soft-core VLIW Processor", in IEEE International Conference on Field-Programmable Technologies (ICFPT 08), pp. 369 - 372, 2008.

[3] J. Mike, Superscalar Microprocessor Design, Prentice Hall (Prentice Hall Series in Innovative Technology), 1991.

[4] J. Smith and G. Sohi, “The micro architecture of superscalar processors,” Proceedings of the IEEE, vol. 83, no. 12, pp. 1609-1624, December 1995.

[5] J. Fisher, P. Faraboschi, and C. Young, Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools, Morgan Kaufmann, 2004.

[6] Philips, Inc., An Introduction to Very-Long Instruction Word (VLIW) Computer Architecture, Philips Semiconductors, 1997.

[7] A. Cosoroaba and F. Rivoallon, “Achieving higher system performance with the Virtex-5 family of FPGAs,” White Paper: Virtex-5 Family of FPGAs, Xilinx WP245 (v1.1.1), July 2006.

[8] A. Percey, “Advantages of the Virtex-5 FPGA 6-Input LUT architecture,” White Paper: Virtex-5 FPGAs, Xilinx WP284 (v1.0), December 2007.

[9] Virtex-5 FPGA User Guide UG190 (v5.4), March 2012.