Reconfigurable Instruction Decoding for a Wide-Control-Word Processor

Alen Bardizbanyan, Magnus Själander, and Per Larsson-Edefors
VLSI Research Group, Dept. of Computer Science and Engineering,
Chalmers University of Technology, SE-412 96 Gothenburg, Sweden
{alenb,hms,perla}@chalmers.se

***Abstract*—Fine-grained control through the use of a wide control word can lead to high instruction-level parallelism, but unless compressed the words require a large memory footprint. A reconfigurable fixed-length decoding scheme can be created by taking advantage of the fact that an application only uses a subset of the datapath for its execution. We present the first complete implementation of the FlexCore processor, integrating a wide-control-word datapath with a run-time reconfigurable instruction decompressor. Our evaluation, using three different EEMBC benchmarks, shows that it is possible to reach up to 35% speedup compared to a five-stage pipelined MIPS processor, assuming the same datapath units. In addition, our VLSI implementations show that this FlexCore processor offers up to 24% higher energy efficiency than the MIPS reference processor.**

I. Introduction

Different software applications have different computing demands on the hardware. In order to obtain high performance and energy efficiency, reconfigurable architectures are important as they allow for post-fabrication alterations. FPGA technology is the natural choice for reconfigurable systems as it intrinsically supports straightforward run-time reconfiguration. However, reconfiguration comes with a power dissipation overhead [1].

The FlexSoC scheme [2] represents an attempt to introduce aspects of reconfiguration to a processor-centric ASIC platform. The FlexCore processor—the key component of this scheme—has a wide control word to enable the implementation of a datapath with many different units and communication paths. A wide control word, however, presents challenges to the design of an instruction decoding scheme, since wide instructions would require high memory bandwidth and footprint. The FlexCore processor is based on an *instruction decompressor* that can expand the compressed instructions from memory. Since a compressed instruction lacks the expressiveness of the wide control word, it can only control a reduced set of the datapath. Thus, a FlexCore instruction decompressor must be run-time reconfigurable to track the need of the application currently being executed.

Conceptually, the instructions that are fetched from memory are decompressed into a wide control word which is applied to the FlexCore datapath [2]. An algorithm for compressing wide control words has already been proposed [3], but a decompressor for this algorithm has never been implemented. This paper presents the first implementation of a complete baseline FlexCore processor—including instruction decompressor and datapath—and an evaluation of its performance and energy efficiency.

II. The FlexCore Processor

The baseline FlexCore processor [4] has the same datapath components as a five-stage pipelined general-purpose processor (GPP), such as the MIPS R2000 or the DLX [5], with the addition of a 32-bit multiplier. There is one distinct feature that sets FlexCore apart from other, conventional GPPs: FlexCore does not have a traditional instruction decoder. Instead, all of the control signals for the datapath units and interconnect are directly exposed to the compiler as a wide control word, called the native-ISA (N-ISA) instructions. This feature gives the compiler fine-grained control of the datapath units, and results in efficient utilization of the datapath resources [4]. The expressiveness at instruction level makes it possible to customize the datapath interconnect, which is made up of a crossbar structure with switchbox-based multiplexers. When implementing a FlexCore, the interconnect is customized to the application domain in order to obtain high energy efficiency. Thanks to the expressive instructions and the customizable interconnect, the FlexCore architecture becomes moldable and new datapath units can be added simply by adding the required ports to the interconnect and concatenating the new control signals to the existing N-ISA.

A. Instruction Decompressor and Compression Algorithm

In the FlexSoC scheme [2], the instruction memory contains compressed instructions—the application-specific ISA (AS-ISA)—which are expanded on the fly to N-ISA instructions in a reconfigurable instruction decompressor.

Thuresson *et al.* proposed an algorithm, based on partitioned look-up tables (LUTs), to compress an N-ISA stream into an AS-ISA stream [3]. Here, an N-ISA instruction is divided into sub-groups that consist of highly correlated bits of the N-ISA, for example, the N-ISA bits representing the opcode of the ALU. From the set of all sub-groups all possible sub-sets are generated. Each sub-set represents a LUT candidate that possibly could be used for the final design. The LUT candidates are then grouped together into all possible solutions such that, for one solution, the candidates together contain all sub-groups of the N-ISA. To determine the number of entries of each LUT, for one solution, a set of representative applications can be used. The number of entries is given by the number of unique bit patterns for the sub-groups of the LUT in all N-ISA instructions of an application. The LUTs can also be dynamically reloaded during execution in order to avoid excessively large LUTs. The reload overhead of an application can be reduced by identifying suitable places to insert reload instructions in the AS-ISA code [3].

For a particular FlexCore design and set of applications, a cost function can be used to select the most appropriate solution from the set of all possible solutions. Such a cost function could consider the total width of the AS-ISA, and/or estimates of power, area, and timing of the combined set of LUTs for a solution. The AS-ISA is subsequently created by concatenating the address bits of the LUTs. An AS-ISA instruction is decompressed into an N-ISA instruction by addressing the LUT entries that contain the bit-pattern to be used in the final N-ISA instruction, as illustrated in Fig. 1.

An instruction decompressor for this compression algorithm has never been implemented. The implementation and evaluation of a complete FlexCore is the key contribution of this paper.

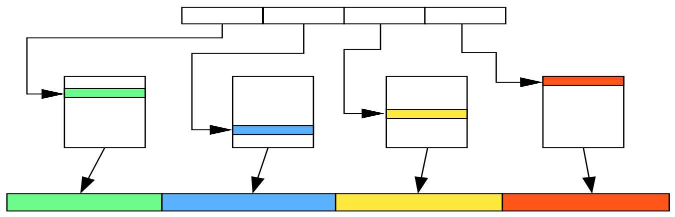

Fig. 1. Decompression scheme from AS-ISA into N-ISA instruction.

III. Instruction Decompressor Implementation

The decompressor implementation is based on look-up tables (LUTs) whose internal structure is shown in Fig. 2. The total LUT count and the configuration of the LUTs are determined off-line during the design phase.

Each LUT entry corresponds to a register composed of a number of flip-flop cells. The appropriate entry is selected using a multiplexer. We integrate clock-gating cells to reduce the flip-flop size; clock-gated flip-flops do not need an enable input or a multiplexer to keep the state.

There is a pipeline stage between the decompressor and the datapath. This is important for reasons that relate to timing; a pipeline stage is needed in order not to create a critical path in the decompressor path. Without the pipeline stage, most timing paths will have an additional delay from the LUT multiplexers, which adversely would affect area and power.

A. AS-ISA Instructions

Two types of AS-ISA instructions are used with the decompressor: Load instructions (Fig. 3) and normal instructions (Fig. 4). The LSB of the AS-ISA word determines the type: Logic 1 implies a load instruction, logic 0 implies a normal instruction.

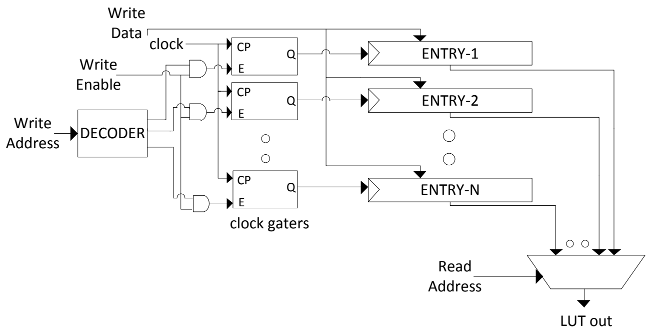

Fig. 2. Detailed LUT block diagram.

1) Configuration of the load instructions: As shown in Fig. 3, the LUT enable (LE) signals determine which LUT will be loaded—there are as many LE signals, as there are LUTs. Next to the LUT enable field, we have the write address: The size of this address depends on the address size of the corresponding LUT which is enabled for write operation. The third bit field is the write data, whose size is determined by the width of the corresponding LUT.

2) Configuration of the normal instructions: As shown in Fig. 4, the least significant 16 bits of the immediate are located next to the LSB of the AS-ISA. Next to the immediate field, the read addresses of the LUTs are located.

3) Cost function for the compression algorithm: The cost function currently used for determining which solution to select from all possible solutions given by the compression algorithm focuses on reducing the AS-ISA width (Eq. 1) and the total size of the decompressor (Eq. 2) as shown in Eq. 3.

$$AS\text{-}ISA_{width} = (\sum \lceil \lg(LUT_{entries}) \rceil) + 17 \tag{1}$$

$$Size = \sum (LUT_{entries} \times LUT_{width}) \tag{2}$$

$$F_{cost} = AS\text{-}ISA_{width} =< 64 \; \& \; \min(Size) \tag{3}$$

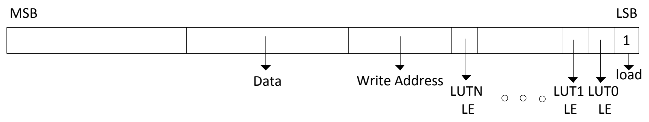

Fig. 3. The content of the AS-ISA word for load operations.

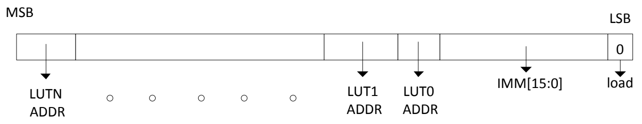

Fig. 4. AS-ISA word during normal execution of the program.

B. Decompressor Clock Network

Although the integrated clock-gating cells reduce the clock pin power of the LUT flip-flops during normal program execution, the accumulated capacitance of these cells still gives rise to high clock network power. In order to reduce the clock power, the clock signal to the clock-gating cells can also be gated, unless a LUT instruction load is performed.

An additional clock-gating cell is inserted between the main clock and the LUT clock inputs. The enable signal for the added clock-gating cell is the LSB of the AS-ISA word, since this is a logic 1 when LUT load is performed.

IV. Framework for Evaluation

Three benchmarks from the Embedded Microprocessor Benchmark Consortium (EEMBC) Telecom benchmark suite were used for the evaluation; Autocorrelation (Autcor), Fast Fourier Transform (FFT) and Convolutional Encoder (Conven). The evaluation is based on the kernel of the benchmarks.

We compare the FlexCore processor implementation with a five-stage in-order MIPS R2000 implementation. Both processors have the same datapath units. Since the MIPS compiler generates the assembly code for the MIPS R2000 processor with one branch delay slot, the branch logic is built in the instruction decode stage of the MIPS R2000 reference processor. This somewhat affects the timing constraint on the register file read multiplexers. In order to avoid the negative impact of branch logic on the timing constraint of the register file, the timing paths that go through the register file and end in the PC register are defined as false paths during synthesis for the MIPS reference.

A. FlexCore Interconnect Configuration

In this evaluation, we adopt the 60-link interconnect configuration that was identified in an earlier study [6][1]. This configuration was shown to yield the lowest energy dissipation across nine benchmarks from the EEMBC benchmark suite, however, this energy estimate does not include the decompressor.

B. Tools

Both processor designs are defined using VHDL. An existing processor design toolchain called FlexTools is used to generate the VHDL code for a specific FlexCore configuration automatically [6]. The current FlexSoC compiler generates statically scheduled FlexCore assembly instructions—called Register Transfer Notation (RTN) instructions—from MIPS assembly code [6] and each basic block of the MIPS assembly code is scheduled independently. Synopsys Design Compiler is used for synthesis and Synopsys PrimeTime is used to analyze the power dissipation of the switching activity files generated by Cadence NCSIM, while simulating the benchmarks on processor netlists. The clock network power estimation feature of PrimeTime is also used.

[1]This configuration includes all communication paths of a MIPS R2000 processor. However, in comparison, the MIPS R2000 has only roughly half of these links.

C. Technology

A commercial 65-nm low-power standard cell library is used for synthesis. The worst case library corner at 125°C and 1.1 V is used for synthesis and timing analysis. The nominal library corner at 25°C and 1.2 V is used for power estimation.

V. Results

A. Generated LUT Configuration

The kernels of the benchmarks are given as input to the compression algorithm. The resulting LUT configuration when applying the cost function described in Sec. III-A3 is shown in Table I.

There are in total 10 LUTs and the two columns on the right show the total width and depth of these LUTs. The total width of the AS-ISA word is 60 bits, which is determined by the normal AS-ISA instruction. Some of the N-ISA fields, such as the contents of LUT4 and LUT5, are very effectively compressed in terms of bit count. On the other hand, some of the grouping only reduced the total bit count by one bit. This may be improved when software support for reloading of the instructions during program execution is implemented in the processor design toolchain. Currently, the LUT depth is generated in such a way that the benchmark with the largest combination[2] of N-ISA signals fits.

TABLE I
LUT Contents

LUTID	Contents	Field width	LUT width	LUT depth
LUT0	IMM[31:16]	16	16	2
LUT1	BUF1_INTR_ADDR	3	6	15
	BUF2_INTR_ADDR	3		
LUT2	LSWRITE_INTR_ADDR	3	6	21
	LSADDR_INTR_ADDR	3		
LUT3	ALUINA_INTR_ADDR	3	6	22
	ALUINB_INTR_ADDR	3		
LUT4	LSOPERATION	2	15	16
	LSSIZE	2		
	LSSTALL	1		
	PCIMMSEL	1		
	PCOP	3		
	PCSTALL	1		
	MULTSTALL	1		
	MULTWE	1		
	MULTA_INTR_ADDR	2		
	MULTB_INTR_ADDR	1		
LUT5	ALUOPCODE	4	8	14
	ALUSTALL	1		
	FBCONTROL_INTR_ADDR	3		
LUT6	BUF1WE	1	6	23
	BUF2WE	1		
	RFWRITE_INTR_ADDR	4		
LUT7	RFWRITEADDR	5	6	27
	RFWENABLE	1		
LUT8	RFREADADDR2	5	6	24
	RFSTALL2	1		
LUT9	RFREADADDR1	5	6	26
	RFSTALL1	1		

[2]The number of unique bit patterns in a LUT for an application.

B. Cycle Count

For all the three benchmarks, the cycle count of the FlexCore processor is less than that of the MIPS (Table II). The cycle counts shown for FlexCore include the LUT load cycles. For the Autcor application, FlexCore is 35% faster compared to the MIPS reference. For the Conven application, FlexCore is 18% faster since this application has short loops and, as a result, it is hard to exploit instruction-level parallelism (ILP).

TABLE II
TOTAL CYCLE COUNT FOR BENCHMARK KERNELS

	Autcor	FFT	Conven
MIPS	1,223 (100%)	58,036 (100%)	91,193 (100%)
FlexCore	801 (65%)	40,361 (69%)	74,971 (82%)

The load cycle counts for the FlexCore processor are 100, 190, and 135 for the applications Autcor, FFT, and Conven, respectively. In fact, the Autcor application requires 701 cycles without loading, which implies a 43% shorter execution time as compared to the MIPS reference. Since the kernel of the two other benchmarks takes significantly longer time to execute, their load cycle counts can almost be neglected.

The load cycle count of the FFT application is equal to the total depth of the LUTs. This means that this is the application which has the biggest combination of N-ISA signals.

TABLE III
MINIMUM CYCLE TIME AND CORRESPONDING AREA

	Cycle time (ns)	Area (μm^2)
MIPS	1.84 (100%)	56,264 (100%)
FlexCore	1.91 (104%)	73,610 (130%)

TABLE IV
EXECUTION TIME AND ENERGY DISSIPATION AT CLOCK PERIOD OF 2.2 NS

	Autcor		FFT		Conven	
	Time (μs)	Energy (nJ)	Time (μs)	Energy (μJ)	Time (μs)	Energy (μJ)
MIPS	2.69 (100%)	16.8 (100%)	127.6 (100%)	1.23 (100%)	200.6 (100%)	1.35 (100%)
FlexCore	1.76 (65%)	12.8 (76%)	88.80 (70%)	1.08 (88%)	164.9 (82%)	1.27 (94%)

C. Area, Performance and Energy Efficiency

The minimum cycle times of the processors, and the corresponding area requirements, are shown in Table III. The MIPS reference can operate at a 4% higher clock rate than FlexCore, the reason being that the MIPS multiplier unit has two links on its inputs, while the FlexCore multiplier input has three links [6]. The reason the FlexCore multiplier has one extra link is that it can exploit ILP. In general, in order to exploit available ILP for different applications sometimes more interconnect links are needed compared to the MIPS.

Due to the rapid area increase for strict synthesis timing constraints, we use a relaxed timing constraint for the ensuing energy dissipation evaluation: For a 2.2-ns timing constraint, the area of the MIPS and the FlexCore is 45,370 μm^2 and 65,642 μm^2, respectively. Since the FlexCore employs many LUTs in its instruction decompressor, the area of the MIPS is 31% smaller for the relaxed constraint.

Table IV presents total execution time and energy dissipation. For all three benchmarks, the energy dissipation of the FlexCore processor is always smaller than that of the MIPS reference. In the Autcor application, FlexCore offers a 24% energy reduction, with a 35% execution time reduction. In Conven, only an energy reduction of 6%, with an execution time reduction of 18%, is obtained. This means speedup is important for energy efficiency.

VI. RELATED WORK

One of the inherent advantages of the wide control word and the interconnect scheme used is that FlexCore is not a register-file centric processor. This means that references can be done to data outside the register file, such that data in the output registers of the datapath units can be forwarded to other datapath units. Sami *et al.* demonstrated that forwarding in VLIW embedded architectures, with deeper pipeline stages, can save energy in the register file [7].

The No Instruction Set Computer (NISC) [8] uses an exposed datapath that is close in concept to the FlexSoC scheme. NISC uses a dictionary-based code compression scheme for an FGPA environment [9]. They employ a multi-dictionary method, which is similar to the partitioning of the FlexCore LUTs, however, no possibility of reloading is mentioned.

VII. CONCLUSION

The implementation and integration of an instruction decompressor shows encouraging results in terms of reducing the instruction memory bandwidth of the FlexCore processor. In this first decompressor implementation, the 97-bit N-ISA instruction is reduced to 60 bits. Additionally, a complete FlexCore processor shows promising results in terms of cycle count and energy efficiency compared to the MIPS reference.

REFERENCES

[1] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," in *Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, 2006.

[2] J. Hughes, K. Jeppson, P. Larsson-Edefors, M. Sheeran, P. Stenstrom, and L. J. Svensson, "FlexSoC: Combining flexbility and efficiency in SoC designs," in *Proc. IEEE NorChip Conf.*, 2003.

[3] M. Thuresson, M. Själander, and P. Stenstrom, "A flexible code compression scheme using partitioned look-up tables," in *Proc. Int. Conf. on High Performance Embedded Architectures and Compilers*, 2009, pp. 95–109.

[4] M. Thuresson, M. Själander, M. Björk, L. Svensson, P. Larsson-Edefors, and P. Stenstrom, "Utilizing exposed datapath control for efficient computing," *J. Signal Processing Systems*, vol. 57, no. 1, pp. 5–19, 2009.

[5] D. A. Patterson and J. L. Hennessy, *Computer Organization & Design, The Hardware/Software Interface*, 2nd ed. Morgan Kaufman Publishers Inc., 1998.

[6] T. T. Hoang, U. Jälmbrant, E. der Hagopian, K. P. Subramaniyan, M. Själander, and P. Larsson-Edefors, "Design space exploration for an embedded processor with flexible datapath interconnect," in *Proc. IEEE Int. Conf. Application-specific Systems Architectures and Processors*, Jul. 2010, pp. 55–62.

[7] M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, and R. Zafalon, "Low-power data forwarding for VLIW embedded architectures," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 10, no. 5, Oct. 2002.

[8] M. Reshadi and D. Gajski, "A cycle-accurate compilation algorithm for custom pipelined datapaths," in *Proc. 3rd IEEE/ACM/IFIP Int. Conf. on Hardware/Software Codesign and System Synthesis*, 2005, pp. 21–26.

[9] B. Gorjiara, M. Reshadi, and D. Gajski, "Merged dictionary code compression for FPGA implementation of custom microcoded PEs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 1, pp. 11:1–11:21, Jun. 2008.