Design of a Reconfigurable Embedded Data Cache

Ruchi Rastogi Bani[1], Saraju P. Mohanty[2], Elias Kougianos[3], and Garima Thakral[4]
NanoSystem Design Laboratory (NSDL, http://nsdl.cse.unt.edu)
University of North Texas, Denton, TX 76203, USA.
E-mail ID: ruchi.bani@gmail.com[1], saraju.mohanty@unt.edu[2], eliask@unt.edu[3], and gt0024@unt.edu[4]

***Abstract*—Performance and power consumption are very important aspects of embedded systems design. Several studies have shown that cache memory consumes as much as 50% of the total power in such systems. Thus, the architecture of the cache governs both performance and power usage of the embedded system. In this paper a new Reconfigurable Embedded Data (RED) cache is proposed especially targeted towards embedded systems. This paper further explores the issues and considerations involved in designing such a reconfigurable cache. The novelty of the RED cache architecture lies in the fact that it can be configured as direct-mapped, two-way, or four-way set associative using a mode selector function. Thus, one cache design can be used for different applications. The module has been designed, simulated and synthesized in Xilinx ISE 9.1i and ModelSim SE 6.3e using the Verilog hardware description language.**

***Keywords*-Reconfigurable Architecture, Data Cache, Cache Associativity, Embedded Systems**

I. Introduction and Motivation

The need for mobile systems, portable devices, and many other appliances used in our modern life results in a growing demand for embedded computing systems. As this growth occurs at tremendous rate, it reduces the window for time-to-market, which in turn is motivating us to design new core-based system-on-chip (SoC) architectures. In the design of such systems, embedded processors (microprocessor cores) are playing a vital role [1]. Several programming languages and Electronic Design Automation (EDA) tools are currently available for embedded processors, which make programming straightforward and offer different solutions to overcome design challenges and simultaneously optimize design metrics. Designers have to maintain a balance of these metrics and compromise between power, cost, performance and time-to-market. Cache memory, a crucial part of embedded systems, is responsible for consuming approximately half of the total power consumption by these systems [2]. A common approach is the use of separate data and instruction caches as a way of improving performance. Proper cache architecture can bring down the time overhead of accessing data and instructions from off-chip main memory, thereby reducing power consumption. High gains in performance have been achieved by tuning appropriate cache architectures to the application set of embedded systems. Cache sizes, the degree of associativity, block replacement algorithms, write policies, and block size (cache line) are the core parameters for optimizing the cache architecture. Suitable selection of these design parameters can enhance cache performance in terms of power consumption, hit ratio, access time and hardware requirements.

Embedded systems have always been cost sensitive. Cache occupies approximately 50% of the total processor area and also accounts for approximately 50% of a processor's total power in embedded systems, including both static and dynamic components [3]. Thus, cache governs the performance and cost of application specific embedded systems. The direct-mapped (DM) cache architecture is very popular in embedded systems because of its simplicity, faster access time and low power consumption. A DM cache is more energy efficient and uses less power than the same sized two-way or four-way set associative cache since it accesses only one location of tag and data arrays per access [4]. Moreover, a DM cache has faster access time as it does not require a multiplexer to select the requested data from multiple accessed data items in different sets. Although DM cache has the advantage of consuming less area and power, it suffers from relatively poor performance. One way to improve the performance of such systems is to use set associative cache at the expense of larger area and higher power consumption compared to direct-mapped cache.

The associativity requirement of data caches for almost all embedded systems is one way, two-way or four-way. In order to match the cost, performance, and power goals with targeted time-to-market, a new reconfigurable embedded data cache is proposed. The proposed design can be configured as direct-mapped, two-way or four-way set associative according to the system's requirement.

The rest of the paper is organized as follows: Section II presents our contributions and summarizes related prior research. III introduces the fundamental design elements used in the RED cache and its detailed architectures. In Section IV the prototype of the RED cache along with our design flow and experimental results are presented. Section V concludes the work and gives suggestions for future research.

II. Contribution of this paper and Prior Research

The major contributions of this paper can be categorized under two different aspects:

1) The first aspect is the architecture of the RED-cache which is capable of operating in three configurations: (i) one-way associative (ii) two-way associative, and (iii) four-way associative. A mode selector module was designed within the data cache which allows it to operate in any of these three configurations. The same processor with RED-cache can be used for various applications requiring different associativity to achieve the desired

performance. This RED-cache has been designed to target embedded systems, as most of the popular embedded systems use direct-mapped, two-way or four-way data cache.

2) Another aspect is the use of a Hardware Description Language (HDL) for its implementation. To the best of our knowledge, this is the first time that an HDL has been used for the implementation of a reconfigurable cache architecture. This RED-cache architecture has been prototyped and synthesized using the Verilog HDL in Xilinx ISE 9.1i and simulated through the ModelSim SE 6.3e simulator. Our design has achieved the highest operating frequency among the reviewed designs at the expense of minimal static power overhead.

Several cache designs have been proposed by researchers either to improve the performance of direct-mapped cache or to reduce the access time and power consumption of set associative cache. Classification of related prior research based on design approach is given in Fig. 1 [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [4], [16].

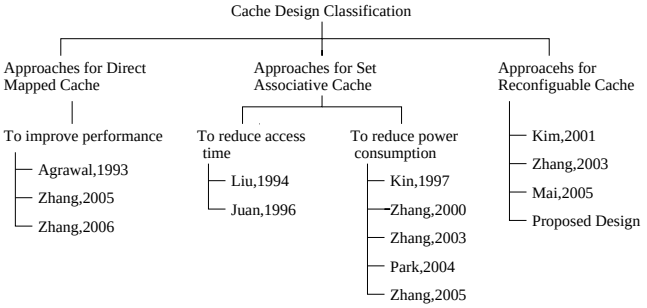

Fig. 1. Classification of related prior research in cache design.

III. The Proposed Data Cache

A. Overview of the RED-Cache

A data cache memory lies between the processor and the main memory [17]. Figure 2 shows the high level block diagram of the RED-cache along with the signals that it needs to communicate with the processor and main memory interface. The processor interface consists of the address bus (PAddress), data bus (PData), and three control signals ($PR\overline{W}$, PStrobe, and PReady). The processor starts a bus transaction when PStrobe is high and a requested address is placed on the address bus. The cache sends a PReady signal to the processor when the bus transaction is completed. The $PR\overline{W}$signal is low for write operation and high for read operation. The main memory interface consists of address bus (MAddress), data bus (MData), and three control signals ($MR\overline{W}$, MStrobe, and MReady).

In order to access the main memory, the requested address is first placed on the MAddress bus along with the MStrobe and $MR\overline{W}$control signals. The $MR\overline{W}$signal is low for write operation and high for read operation. MReady is used to signal the cache memory that the bus transaction is completed

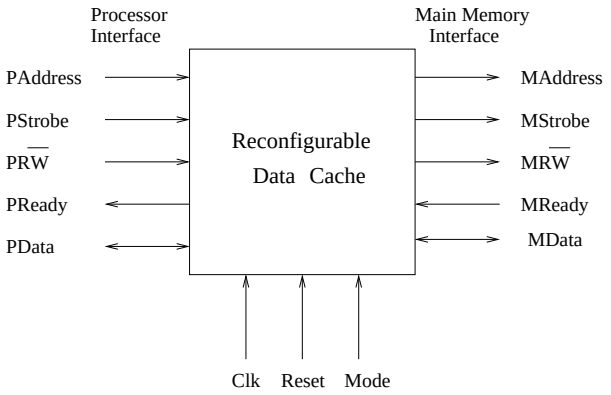

Fig. 2. High-level view of RED-cache.

by the main memory. Two global signals, Clk and Reset are used to synchronize the cache operation with the processor. There is an additional signal called mode, which makes the cache reconfigurable. Depending upon the value of the mode signal, the proposed design of reconfigurable data cache can work in any of three configurations: direct-mapped cache, two-way set associative cache, and four-way set associative cache.

B. Fundamental Elements used in the Cache Design

The overall performance of a cache is determined by the basic elements of its design, such as size, mapping function, replacement policy, write policy, and block size. Table I gives the design features of the proposed architecture. A 256 byte cache has been chosen for implementation due to limitations of FPGA chips. In order to satisfy the associativity requirement of embedded applications, our design can use one of the following mapping functions: direct-mapped, two-way set-associative, and four-way set-associative. Although this design is extendable up to N-way associativity with slight modifications in the mode selector unit, we have implemented only up to 4-way associativity. Least Recently Used (LRU) replacement policy has been used in order to obtain high hit ratio. Write through policy has been chosen for write operation. Easy implementation and consistency among main memory and cache are two major benefits of this policy. The block size of one word has been taken for simplicity.

TABLE I
ELEMENTS AVAILABLE FUSED BY THE RED CACHE ARCHITECTURE [18]

Elements	Existing	Proposed Architecture
Cache Size	Few bytes to several Kbytes	256 bytes
Mapping Function	Direct Fully Associative N-way Set Associative	Direct Mapped Two-way Set-associative Four-way Set-associative
Replacement Policy	Least Recently Used (LRU) First-in-first-out (FIFO) Least Frequently Used (LFU) Random	Least Recently Used (LRU)
Write Policy	Write through Write Buffer Write Back	Write through
Block Size	Multiple Words	1 Word

C. Architecture of the RED-Cache

The top-level design module shown is further divided into sub modules to build the main module as shown in figure 3. The proposed reconfigurable cache consists of six sub modules: Tag RAMs, Valid RAMs, Data RAMs, Line Replacement unit, Mode Selector Unit, Cache Controller.

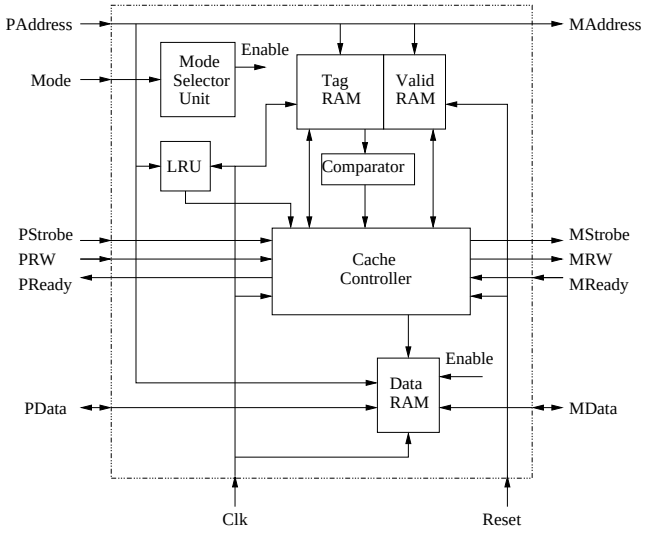

Fig. 3. Sub-modules of the proposed design.

1) Tag RAM: The tag RAM is an array of SRAM cells used to hold the tag fields of the physical addresses which are currently stored in the data ram. When the processor makes an access call for any particular memory location, first it checks to see whether the requested memory address is in the cache memory or not by looking for that address in the tag RAM. If its there, it gets the data from the cache, otherwise from the main memory.

2) Valid RAM: Contains a valid bit associated with each memory location that is currently mapped to cache. This valid bit indicates whether the cache entry is valid or not. Initially, all entries are set to invalid. As the data contents are moved from main memory to cache, the valid bit corresponding to them is set to valid.

3) Data RAM: The data RAM is also an array of SRAM cells; they store the data contents of physical addresses which are currently mapped to cache.

4) Tag Comparators: When a data access request is initiated by the processor, then all tag comparators simultaneously compare the content of the tag array indicated by the index field with the requested address's tag field. If any of the tag arrays hold the requested address, the corresponding tag comparator generates the active high match signal.

5) Line Replacement Unit: When a cache miss occurs, the line replacement unit determines which line should be removed from the cache. According to the mode selection signals, this unit will replace the least recently used (LRU) line from the requested address in case of cache misses. The LRU unit has been implemented using the counter method, which provides good performance for low associativity (two or four) [19]. An n-bit register is associated with each cache line in a set, which is used to store the LRU record. The register value shows the order in which the cache lines have been accessed. A register with lower value indicates the least recently used line with in a set. The LRU implementation using the counter method for a set is shown in figure 4. All registers are set to zero initially. When a particular cache line within the set is being accessed, the corresponding register value is compared with other register values and set to the largest value. Finally, the priority encoder chooses the line with highest register value as the LRU line.

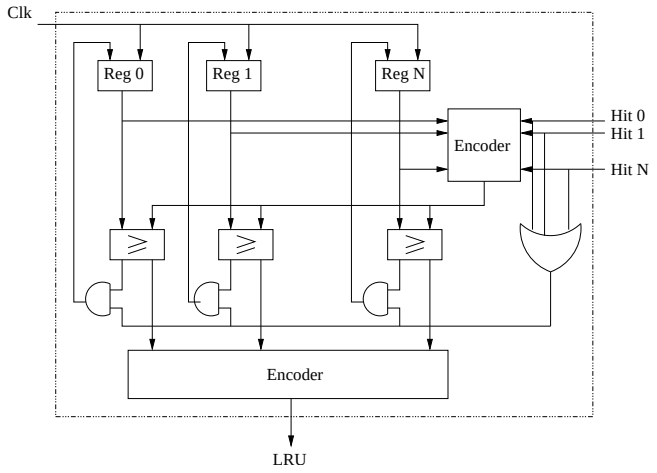

Fig. 4. LRU Implementation using the counter method.

6) Mode Selector: The data cache can be configured as one, two and four way set associative according to the input value at a two bit mode selection terminal. In order to generate the enable control signal for each of the four sets, these two mode bits are combined with the A7 and A6 bits of the physical address.

7) Cache Controller: Controls all operations within the cache and is implemented using a finite state machine as shown in figure 5. Control signals asserted during each state are summarized in Table II.

1) Idle State: No memory access and the processor is idle in this state. Controller remains in this state until some read or write operation is requested by the processor. If a read request is initiated by processor, control transfers to the read state. If a write request is initiated, control transfers to the write state.
2) Read State: In this state, the cache is checked for availability of the requested address, as the processor initiates a read operation. If the requested address is currently in cache, a cache hit occurs and control returns to the idle state during the next active clock. Otherwise, a cache miss occurs and control transfers to readmiss state to initiate main memory access.
3) ReadMiss State: Main memory read access is initiated by the cache controller and control handed over to readmemory state.

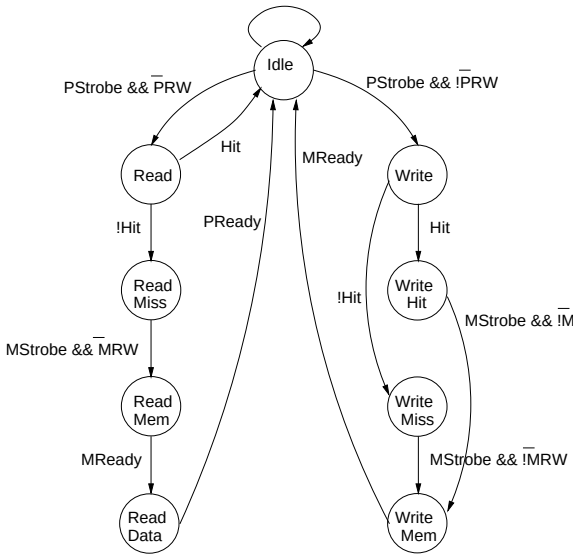

Fig. 5. Cache controller state machine.

4) ReadMemory State: Controller has to wait until main memory finds the requested address and reads the data from that location. Once main memory loads the data contents on data bus, it asserts a ready signal to controller and control transfers to readdata state.
5) ReadData State: The requested data is available on the data bus. Write this data to least recently used cache line and simultaneously load it to the processor's data bus to complete the read request. After the completion of processor's initiated read operation control goes back to idle state.
6) Write State: In this state, the cache is checked for availability of the requested address, as the processor initiates a write operation. If the requested address is currently in cache, a cache hit occurs and control returns to writehit state during the next active clock. Otherwise, a cache miss occurs and control transfers to writemiss state. Whether it is a write hit or miss, data must be written to cache and simultaneously updated to main memory also.
7) WriteHit State: On a cache hit for write operation, the controller stimulates the write control signal of data cache to write the data contents sent by the processor and also initiates write through for main memory. Control transfers to writedata state on the next active clock.
8) WriteMiss State: On a cache miss for write operation, data contents sent by the processor are written to least recently used cache line and associated tag and valid rams are also updated. Controller initiates write through policy for main memory and transfers control to writedata state on the next active clock.
9) WriteData State: Controller has to wait until main memory completes the write operation and sends back a ready signal to the controller. After completing the requested write operation, controller will come back to idle state.

TABLE II
ACTIVE HIGH CONTROL SIGNALS DURING EACH STATE

State	Active High Control Signals
Idle	None
Read	PStrobe, PRW , PDataOE, PReadyEnable, Hit (for read hit)
ReadMiss	PRW , PDataOE, Miss, Mstrobe
ReadMemory	PRW , PDataOE, MRW
ReadData	PRW , MRW , PDataOE, PDataSelect, CacheDataSelect, Ready
Write	PStrobe
WriteHit	Hit, MStrobe, MDataOE
WriteMiss	Miss, MStrobe, MDataOE
WriteMemory	MStrobe, Ready

D. Overall Architecture

The detailed architecture of the proposed design is shown in figure 6, which consists of sub modules along with connecting logic such as an Encoder, Tag Comparators, and several Multiplexors. For simplicity we have chosen the following:

1) 16-bit address bus, which gives us a total address space of 64K as main memory.
2) 256 bytes of cache, which means that only the 8 least significant bits are required to address the cache.
3) Data bus of 8 bits.

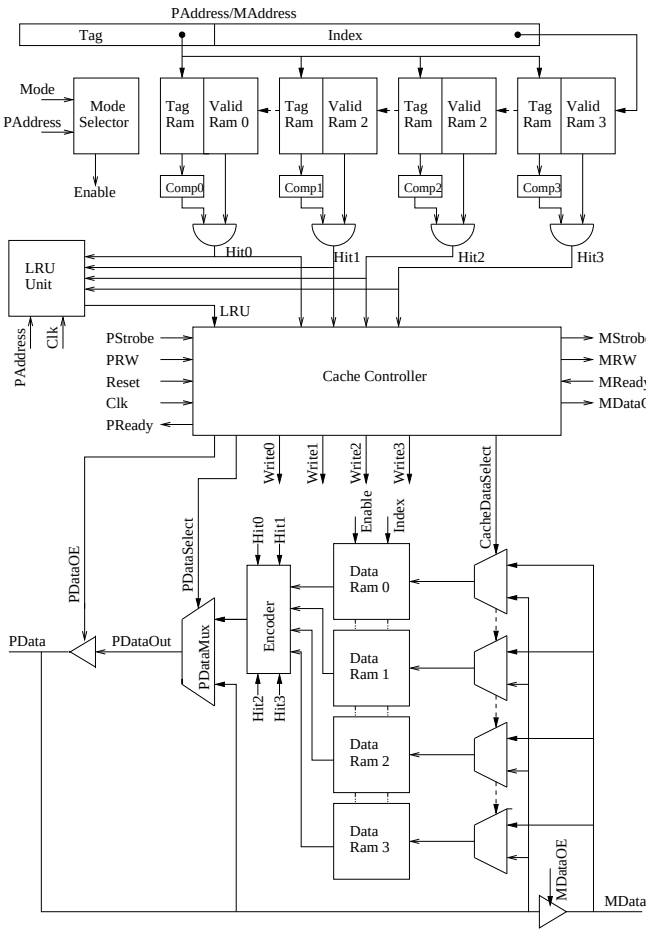

Fig. 6. Detailed architecture of the reconfigurable data cache.

The main memory is divided up into 256 blocks of 256 bytes each, where each block is mapped to the cache. Because only 8 address bits are needed to identify the address in cache, the 16 bit physical address is divided into an 8 bit tag field and an 8 bit index field. Due to the reconfigurable architecture, the total cache of 256 bytes is divided into four sets of 64 locations each. Thus, we have used four sets of Tag, Valid, and Data RAMs in this design.

IV. Prototyping and Simulation Results

A. Implementation Flow

We followed a top-down flow for the implementation and simulation of our design. First, we have logically and structurally divided the main module into various submodules,as we did in the architectural design flow. Each submodule has been designed, synthesized, and simulated individually. Once all the submodules were ready and tested for functional verification, we integrated them to obtain the main module.

B. FPGA Prototyping

The architecture of the reconfigurable data cache was modeled using Verilog and the functional simulation was carried out using Modelsim SE 6.3 e. The code is written in a hierarchical fashion and is a combination of structural and synthesizable behavioral coding. This Verilog code was compiled and synthesized using Xilinx ISE 9.1i. The implementations are targeted for Xilinx's Virtex-5 family of FPGAs. The Virtex-5 family is built in a 65-nm copper CMOS technology. The Virtex-5 Configurable Logic Blocks (CLBs) are based on 6-input look-up tables and a flip-flop. Each CLB contains a pair of bit slices; each bit slice further consists of four 6-input look-up tables and four flip-flops, for a total of eight 6-input look-up tables and eight flip-flops per CLB [20]. Complete synthesis of all Verilog modules is performed along with their mapping, placement, and routing. For functional verification of the designed cache module in all three operational modes, a trace file of 20 test cases has been applied through a driver.

Complete results obtained from the trace file in the three different modes are summarized in Table III. The memory write time for all three configurations (1-way, 2-way, and 4-way) is the same because we have used a write-through policy for write operations. In this policy, the main memory is simultaneously updated with data cache for every write access.

Various design metrics of the proposed design in direct mapped, two-way and four-way set-associative modes are summarized in Table IV. The timing path from a clock to any other clock in the design indicates the minimum period. The proposed design possesses the smallest minimum period and also the highest maximum operating frequency with respect to 1-way, 2-way, and 4-way set-associative caches. The maximum path delay is an indicator of the maximum path from inputs to outputs. This delay is smallest for the proposed design. The proposed design has obtained performance improvement in terms of minimum period, maximum operating frequency and maximum path delay due to distribution of clock in target device (Virtex-5). In virtex-5 family of FPGAs, there are total 32 global clocks and each device is divided into regions for distribution of clock. The design of the proposed module is mapped into the target device in a very compact manner as compared to the other three configurations, thus producing a minimum timing path from one synchronous element to another. Since the proposed design is mapped compactly, maximum delay from any one node to any other node is also small compared to other designs. The cell usage, expressed in BELS, reports the count of all the logical cells that are basic elements of the Virtex technology, for example, LUTs, MUXCY, MUXF5, MUXF6, MUXF7, MUXF8. Flip-flops or slice register counts indicates the total number of latches and flip-flops used by the design. The reconfigurable cache design occupies a larger number of cells, slice registers and LUTs in order to provide reconfigurability. Table V summarizes design metrics of the reconfigurable data cache for three different FPGA technologies. The power consumption has been calculated under ambient temperature of $25°C$. Comparison of the proposed design with other reconfigurable memory is given in Table VI.

TABLE III
Summary of reconfigurable cache operation in three modes.

Design Metrics	Direct Mapped	2-Way	4-Way
Mode	00	01/10	11
No. of Access	20	20	20
Read Access	11	11	11
Write Access	9	9	9
Read Hits	2	7	7
Read Miss	9	4	4
Write Hits	2	4	4
Write Miss	7	5	5
Total Hits	4	11	11
Total Miss	16	9	9
Memory Read Time	1030	730	730
Memory Write Time	945	945	945

TABLE IV
Comparison of various design metrics of proposed design with direct-mapped, 2-way, and 4-way set associative caches.

Design Metrics	Direct Mapped	2 Way	4 Way	Reconfi. Cache
Maximum Frequency (MHz)	154.036	131.683	184.706	212.513
Minimum Period (ns)	6.492	7.594	5.41	4.706
Maximum Combinational Path Delay (ns)	4.901	4.897	3.338	3.342
Cell Usage (BELS)	399	782	1468	1888
FlipFlops/Slice Reg	267	530	946	1228
Slice LUTs	395	776	1446	1881
IO Utilization	56	56	56	58
No of Bit Slices	63	1006	1666	1991
Power (mW)*	-	-	1033	1362
Gate Count	135,952	271,795	280,781	288,986

V. Conclusions and Future Research

This paper presented the architecture and design of a new N-way reconfigurable data cache for embedded systems. The proposed data cache can be configured as direct-mapped, two-way and four-way set-associative to fulfill the systems'

TABLE VI
COMPARISON OF PROPOSED CACHE WITH EXISTING RECONFIGURABLE CACHE.

Name	Multi-function Computing Cache [15]	Way Concatenation Cache [4]	Reconfigurable Memory [16]	Proposed Design
Year	2001	2003	2005	2009
Performance Improvement	Up to factor of 50-60 for computational applications	Due to way concatenation and way shutdown circuitry		In terms of Maximum Frequency, Combinational delay
Hardware and area overhead	10-20% of base cache memory	Negligible	15%	Due to Mode Selector Unit
Access Time	Increased by 1.6%	As 4-way	Increased by 10%	
Associativity or Configuration		1,2,4-way	Cache, FIFO, Scratchpad	1,2,4-way
Power		Dynamic Power savings up to 40% compared to conventional 4-way	Power Overhead 10%	Static Power Overhead 20%
Simulation approach		Simple Scalar		ModelSim

TABLE V
COMPARISON OF DESIGN METRICS OF RECONFIGURABLE DATA CACHE FOR VARIOUS FPGA TECHNOLOGIES.

Design Metrics	QPro Virtex Hi-Rel XQV1000	Automotive Spartan 3E	Virtex 5
Maximum Frequency (MHz)	37.111	76.429	212.513
Minimum Period (ns)	26.946	13.084	4.706
Maximum Combinational Path Delay (ns)	21.074	6.280	3.342
Cell Usage (BELS)	7596	4726	1888
FlipFlops/Slice Reg	1311	1230	1228
Slice LUTs	6361	4014	1881
IO Utilization	58	58	58
No of Bit Slices	3335	2090	1991
Power (mW)	-	102	1362

requirements. We have achieved this reconfigurability with the help of a mode selector while utilizing the full capacity of cache. The FPGA implementations of the reconfigurable cache along with direct-mapped, two-way and four-way set-associative caches have been demonstrated. The performance of the proposed design is compared with the direct-mapped, two-way and four-way cache architectures. The proposed design can be further optimized in terms of speed, area and power consumption.

REFERENCES

[1] T. Givargis, "Improved indexing for cache miss reduction in embedded systems," in *Proc. Design Automation Conference*, Jun. 2–6, 2003, pp. 875–880.

[2] J. L. Hennessy and D. A. Patterson, *Computer architecture Quantitative Approach*. Morgan-Kaufmann Publishing Co., 1996.

[3] A. Malik, B. Moyer, and D. Cermak, "A low power unified cache architecture providing power and performance flexibility," in *Proc. International Symposium on Low Power Electronics and Design ISLPED '00*, 2000, pp. 241–243.

[4] C. Zhang, F. Vahid, and W. Najjar, "A highly configurable cache architecture for embedded systems," in *Proc. 30th Annual International Symposium on Computer Architecture*, Jun. 9–11, 2003, pp. 136–146.

[5] A. Agarwal and S. Pudar, "Column-associative caches: A technique for reducing the miss rate of direct-mapped caches," in *Proceedings of the International Symposium on Computer Architecture*, 1993, pp. 179–180.

[6] C. Zhang, "An efficient direct mapped instruction cache for application-specific embedded systems," in *CODES+ISSS '05: Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. New York, NY, USA: ACM, 2005, pp. 45–50.

[7] ——, "Balanced cache: Reducing conflict misses of direct-mapped caches," in *ISCA '06: Proceedings of the 33rd annual international symposium on Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 155–166.

[8] L. Liu, "Cache designs with partial address matching," in *Proc. 27th Annual International Symposium on MICRO-27 Microarchitecture*, Nov. 30–Dec. 2, 1994, pp. 128–136.

[9] T. Juan, T. Lang, and J. Navarro, "The difference-bit cache," in *Proceedings of the 27th International Symposium on Computer Architecture*, 1996.

[10] J. Kin, M. Gupta, and W. H. Mangione-Smith, "The filter cache: an energy efficient memory structure," in *Proc. Thirtieth Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 1–3, 1997, pp. 184–193.

[11] M. Zhang and K. Asanovic, "Highlyassociative caches for low-power processors," 2000. [Online]. Available: citeseer.ist.psu.edu/zhang00highlyassociative.html

[12] C. Zhang, F. Vahid, and W. Najjar, "Energy benefits of a configurable line size cache for embedded systems," in *Proc. IEEE Computer Society Annual Symposium on VLSI*, Feb. 20–21, 2003, pp. 87–91.

[13] J. Park, C. Kim, J. Lee, and S. Kim, "An energy efficient cache memory architecture for embedded systems," in *Proceedings of the ACM symposium on Applied Computing*, 2004, pp. 884–890.

[14] C. Zhang, F. Vahid, J. Yang, and W. Najjar, "A way-halting cache for low-energy high-performance systems," *ACM Transactions on Architecture and Code Optimization*, vol. 2, no. 1, p. 34 54, March 2005.

[15] H. Kim, A. Somani, and A. Tyagi, "A reconfigurable multifunction computing cache architecture," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, pp. 509 – 523, Aug 2001.

[16] K. Mai, R. Ho, E. Alon, D. Liu, Y. Kim, D. Patil, and M. Horowitz, "Architecture and circuit techniques for a 1.1-GHz 16-kb reconfigurable memory in $0.18 - \mu m$ CMOS," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 1, pp. 261– 275, Jan 2005.

[17] D. Crisu, "An architectural survey and modeling of data cache memories in verilog hdl," in *Proc. International Semiconductor Conference CAS '99*, vol. 1, Oct. 5–9, 1999, pp. 139–142.

[18] "Cache memory implementation and design techniques," http://www.faculty.iu-bremen.de/birk/lectures/PC101-2003/07cache/.

[19] T. Sudarshan, R. A. Mir, and S.Vijayalakshmi, "Highly Efficient LRU Implementations for High Associativity Cache Memorys," in *Proceedings of 12th IEEE International Conference on Advanced Computing and Communications*, Dec 2004, pp. 24–35.

[20] http://www.xilinx.com.