Reconfigurable cache Implementation on FPGA

K.A.Naveen Kumar, M.Bharathi, S.A.Hariprasad

Abstract-Cache memory is a common structure in computer system and has an important role in microprocessor performance. The design of a cache is an optimization problem that is mainly related with the maximization of the hit ratio and the minimization of the access time. Some aspects related with the cache performance are the cache size, associativity, number of words per block and latency. In this paper, we propose a reconfigurable cache design with two cache organizations direct mapped and 2-way Set Associative each with four modes each. The designed instruction cache is of size 64 lines and each line can store a word of 18 bit wide. The designed cache is integrated with an 8 bit Pico Blaze Processor. Hit ratio analysis is done for all the modes with different algorithms. Reconfiguration can be done at any point in the assembly program by just changing the cache configuration port. The hit ratio analysis for the algorithms is reported.

Index terms: - Block-RAM, Cache memory, Direct mapped, FPGA, Miss Ratio, Reconfiguration, Set associative.

1. INTRODUCTION

Cache memory is an important part in computer system and has a major role in microprocessor performance. There are three types of cache organizations they are: Direct mapped cache, fully associative cache and set associative cache. In a Direct mapped cache a block of main memory page has to be mapped to a particular location in the cache. This block is not allowed to be kept elsewhere in the cache. In fully associative cache a block of main memory can be kept anywhere in the cache. Set associative lies in between the two extremes. In set associative a block of main memory page has freedom to keep in any of the sets but within a set it should be mapped to particular location. A lot of research has been done on cache memories. The main research lines focus on cache memories architecture simulation oriented to performance analysis; low power cache systems, implementation of fixed and reconfigurable architecture on FPGA's for testing theoretical designs and analysis of cache implemented for high-end or for embedded processors.

In this paper a Reconfigurable cache with 8 cache modes is designed. The designed cache has been integrated with an 8-bit Picoblaze processor. Different algorithms are analysed on all configurations. The hit ratio is analysed and given a report mentioning which organization gives best hit ratio for the algorithm. 8 Block RAM's of FPGA are utilised in the design to make cache with 64 lines of storage capacity.

The paper is organized as follows: Section 2 reviews related work in reconfigurable cache; Section 3 describes the proposed design, implementation and the testing platform; Section 4 presents some results related to basic testing algorithms. Finally, Section 5 presents the conclusions and suggested future work.

- *K.A.Naveen Kumar Currently Pursuing M.tech in VLSI design and Embedded systems at RVCE, Bangalore, India. Ph +919738846497, Naveen.k.86@gmail.com.*
- *M.Bharthi, Associate Professor at RVCE, Bangalore, India, Ph +919880423214, bharathim@rvce.edu.in.*
- *S.A.Hariprasad, Associate Professor at RVCE, Bangalore, India, hariprasad@rvce.edu.in*

2. PREVIOUS WORK

Ranganathan et al.[1] proposes a reconfigurable memory organization that allows the on-chip SRAM to be dynamically divided into different partitions that can be assigned to cache and others conventional processor activities. This organization can benefit applications that doesn't utilise the storage allocated to large conventional caches.

In this paper authors have implemented an associative partitioning method based on N-way associative caches. During reconfiguration, Cache is divided into partitions, with a granularity corresponding to the number of ways of the conventional cache, exploiting the conceptual division into ways already present in a conventional cache. The design has the ability to enable all of the cache ways when required to achieve high performance, but to enable only a subset of ways when cache demands are least. Thus it is mainly applicable for low power embedded system applications. Associativity based partitioning and overlapped wide-tag partitioning are the two techniques used for variable sized partitioning, and were able to address these partitions efficiently. The addressing scheme must efficiently adapt to dynamic resizing of the partition resizes.

Huesung Kim et al. [2] proposes an algorithm for reconfiguration. The algorithm is named as ABC meaning Adaptive Balanced Computing. This does dynamic resource configuration on demand from application between memory and computing resources. Whenever cache demand is less, then the cache can be utilised for other computing applications.

The paper proposed by the author David H.Albonesi [6] provides selective cache ways which has the ability to disable a subset of the ways in a set associative cache during the periods of modest cache activity, while the full cache way remains operational for more cache-intensive periods.

In the paper authored by Santana Gil et.al [7] proposes a reconfigurable cache with fixed size. The cache can work as direct mapped (D.M.) cache or as 2 way set associative (S.A.) cache. For each mode, we can select 1, 2, 4 or 8 words per block.

3. RECONFIGURABLE CACHE

We propose a reconfigurable cache with fixed size. The cache can work as direct mapped (D.M) cache or as 2 way set associative (S.A) cache. The different cache modes are tabulated in Table I. The cache implements "write-through" as the write policy and LRU(least recently used) as replacement policy, in 2 way associative modes.

TABLE 1. CACHE SUPPORTED MODES.

Mode	b2 b1 b0	Configuration
0	0 0 0	Directed Mapped, 1 Word X Block
1	0 0 1	D.M., 2 W X B
2	0 1 0	D.M., 4 W X B
3	0 1 1	D.M., 8 W X B
4	1 0 0	2 Way Set Associative, 1 W X B
5	1 0 1	2 Way S.A., 2 W X B
6	1 1 0	2 Way S.A., 4 W X B
7	1 1 1	2 Way S.A., 8 W X B

A block diagram of the reconfigurable cache structure is presented in Figure 3.1. The data, tags, valid bits and LRU bits are stored in Block RAM of the FPGA. The cache contains data bus, address input bus, output bus, a mode selection write port and a status read only port. The cache module is interfaced with the KCPSM3 processor code which is available from XILINX [13].

3.1 PROPOSED WORK

Direct mapped and 2-Way set associative 1-Word X-Block, 2-Word X-Block, 4-Word X Block and 8- Word X-Block are designed. The basic blocks Mentioned in the block diagram has been designed for the cache functionality. The Cache configuration controller has the option to choose one of the 8 cache modes. Tag address Controller block will separate Tag address and Index address fields of instruction address and then directs the outputs to Tag Equivalence checker.

Tag Equivalence checker will compare the tag address in the memory with the present tag address on the address bus, if both are equal then Cache hit occurs and hit counter is incremented. Then the Index address is decoded from the remaining address field, this will point to the memory location where the instruction is present in cache instruction memory. The Cache instruction memory will output the corresponding 18-bit wide instruction to the processor. If there is an occurrence of cache miss, then the instruction has to be fetched from main memory via Cache Interface controller and fetches the data.

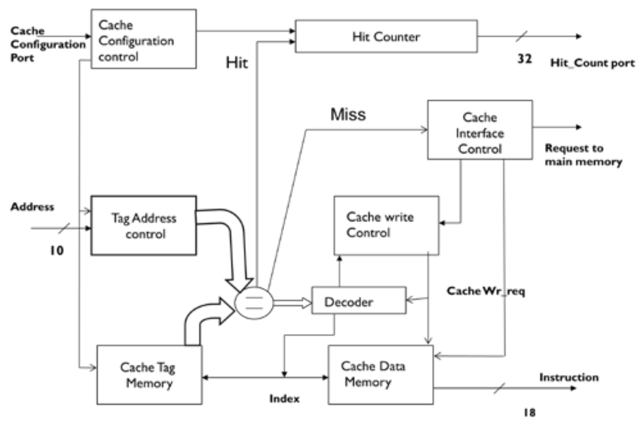


Fig 3.1 Reconfigurable Cache design Block diagram.

The cache fetches 8 consecutive instructions from main memory; only required number of instructions will be copied into cache according to the cache mode. For example in the case of 1-word X-block only the first instruction is copied remaining 7 instructions are discarded, this data will be written to Cache memory through cache write controller. The total number of memory access is considered for hit and miss ratio calculations.

The direct mapped cache is designed by taking 8 Block RAM's each with 1024 memory locations of FPGA and dividing each into equal parts thus 16 memory Blocks are obtained. Since Block RAM is of dual ported each port is allocated to upper and lower half memory for read and write purposes. From Each memory block 4 lines are used as cache lines and thus 8*(4 upper + 4 lower) =64 lines of cache are designed. The main memory of 1024 memory locations is grouped into 16 pages, thus each page contains 64 lines (16*64 =1024). In this direct mapped memory page-0 of main memory can be mapped to cache memory block-0 and so on respectively until page-15.

In case of direct mapped 2-Word X-Block, two words from main memory are brought into cache memory. They are mapped into block-0 and block-8 respectively. If those two are filled then the next two instructions are mapped to block-1 and block-9 correspondingly. Similarly for 4-Word X-Block, four words are mapped to block-0, block-1, block-8 and block-9 respectively. Similarly for 8-Word X Block, eight consecutive words are mapped to block-0, block-1, block-2, block-3, block-8, block-9, block-10 and block-11 respectively. For the 2-way set associative cache the entire 64 lines of cache are divided into two sets way-0 and way-1. In this mapping technique for 1-Word X Block page-0 of main memory can be mapped to block-0 of either way-0 or way-1. Immediately if page-64 is referred in the main memory it will be mapped to the unoccupied block-0 of either way-0 or way-1. But in the case of direct mapped 1-Word X Block the block-0 content of cache is overwritten which may lead to more number of misses. In the case of set associative within a set direct mapped technique is implemented, so this makes a compromise between the direct mapped and fully associative mapping technique. Similarly 2-Word X Block, 4-Word X Block and 8-Word X Block are designed for the 2-Way set associative cache design. In the case of 2-Word X Block set associative cache two consecutive words will be brought to the cache memory and stored. Similarly 4 words and 8 words will be brought from the main memory for the 4-Word X Block and 8-Word X Block set associative cache respectively. Suppose both the ways in the 2-ways set associative are filled and a new instruction is fetched which is having same index address then it replaces the data which is least recently used.

The main memory code is available in the Xilinx ISE simulator. Assembly programs are written with the assembly instructions available for the KCPSM3 processor. These files are saved with .psm extension and executed using windows DOS command prompt. Once the file is executed, KCPSM3 Assembler will generate several files in the working directory. The required files from working directory are <filename>.vhd, <filename>.v, <filename>.coe, <filename>.m, <filename>.hex, <filename> .log and <filename> .dec. The contents of registers, address and instructions can be monitored in the output waveforms. Whenever instruction is present in the cache, hit flag is set and hit counter is incremented.

The designed Cache has been interfaced with Xilinx Picoblaze KCPSM38-bit processor. Assembly program for the same processor has been Written, Compiled, Executed and Results are simulated in the XILINX ISE simulator. The cache successfully fetches the instructions from the main memory and the hit counter displays the number of hits for each cache configuration. Reconfiguration is introduced in such a way that whenever user want to change the cache mode, the cache configuration port can be loaded with the value from 0 to 7 corresponding to the cache modes. This will reconfigure the cache into the new mode and all the data present in the cache will be flushed out but the cache hit counter will be maintained as it is and continues with new mode.

4. RESULTS

The following algorithms have been implemented in the assembly language by instruction set provided in the KCPSM3 processor. The miss ratio is analyzed for bubble sort, merge sort, selection sort and binary search programs. The Fig 4.1, 4.2, 4.3, 4.4 and 4.5 shows the miss ratio plots for bubble sort, merge sort, selection sort, binary search and alarm programs respectively.

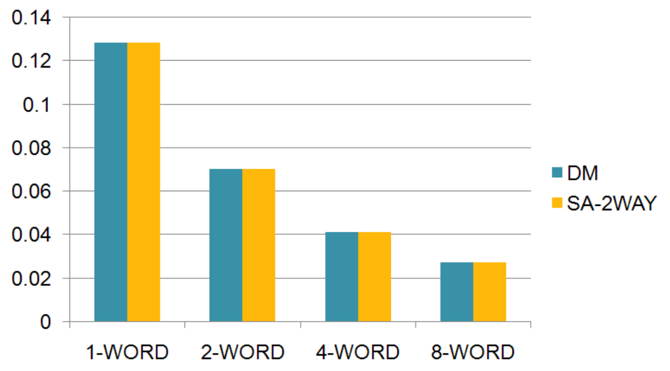


Fig.4.1 Bubble sort

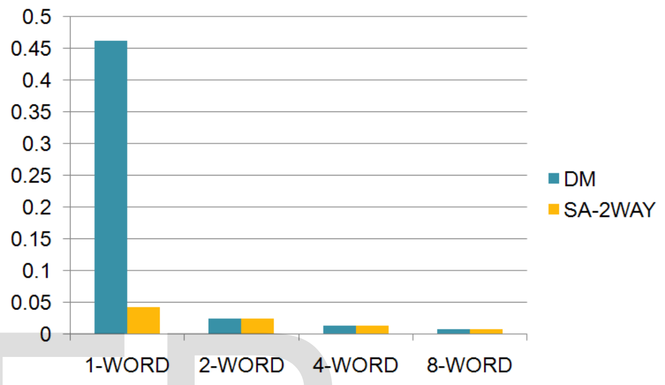


Fig.4.2 Merge sort

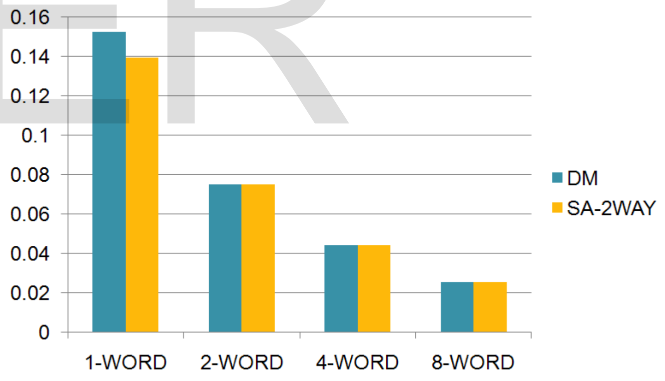


Fig.4.3 Selection sort

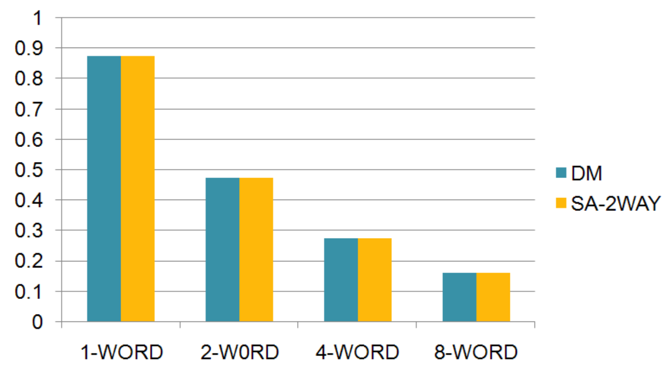


Fig.4.4 Binary search

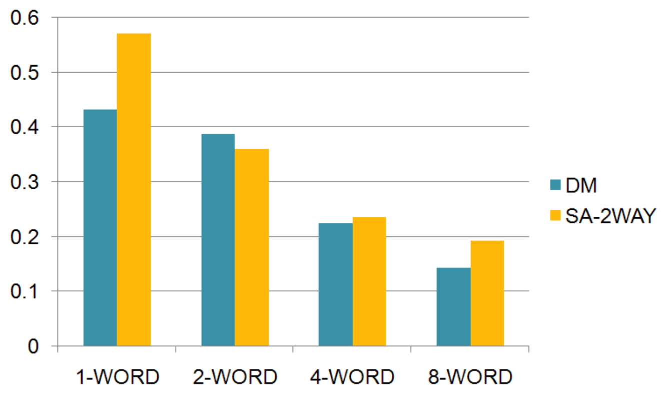
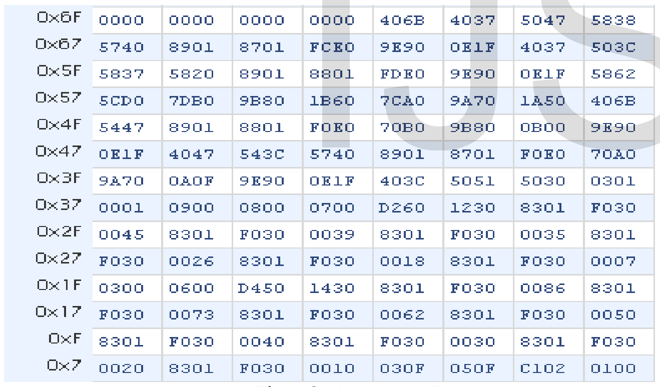


Fig.4.5 Alarm program

From the miss ratio plots it can be inferred, for the entire algorithms miss ratio is worst for 1-Word X Block. The lowest miss rate is observed for modes with 8-Word X Block. No significant difference is appreciated between direct mapped and set associative modes for bubble sort, selection sort, and binary search programs, but there is difference in miss ratio for merge sort 1-Word X Block. The alarm program in fig 4.5 shows significant difference for direct mapped and set associative. The fig 4.6 shows the instruction opcode stored in FPGA main memory BlockRAM generated for merge sort program.



0x6F	0000	0000	0000	0000	406B	4037	5047	5838
0x67	5740	8901	8701	FCE0	9E90	0E1F	4037	503C
0x5F	5837	5820	8901	8801	FDE0	9E90	0E1F	5862
0x57	5CD0	7DB0	9B80	1B60	7CA0	9A70	1A50	406B
0x4F	5447	8901	8801	F0E0	70B0	9B80	0B00	9E90
0x47	0E1F	4047	543C	5740	8901	8701	F0E0	70A0
0x3F	9A70	0A0F	9E90	0E1F	403C	5051	5030	0301
0x37	0001	0900	0800	0700	D260	1230	8301	F030
0x2F	0045	8301	F030	0039	8301	F030	0035	8301
0x27	F030	0026	8301	F030	0018	8301	F030	0007
0x1F	0300	0600	D450	1430	8301	F030	0086	8301
0x17	F030	0073	8301	F030	0062	8301	F030	0050
0xF	8301	F030	0040	8301	F030	0030	8301	F030
0x7	0020	8301	F030	0010	030F	050F	C102	0100

Fig.4.6 Merge sort RAM

The Fig 4.7 shows the waveform containing hit-counter, total memory access, address, instruction opcode and some registers of the processor for bubble sort program.

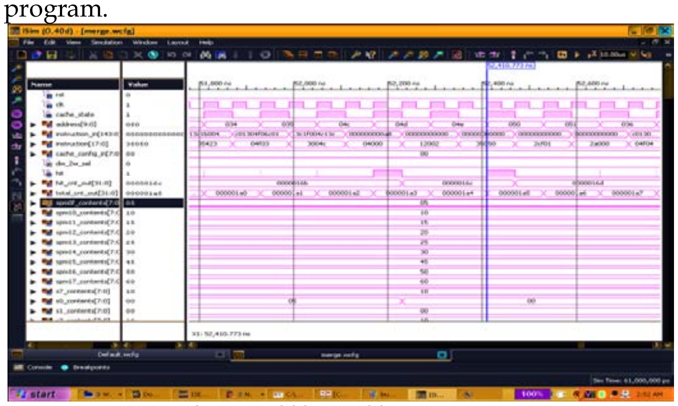


Fig.4.7.Bubble sort hit-counter

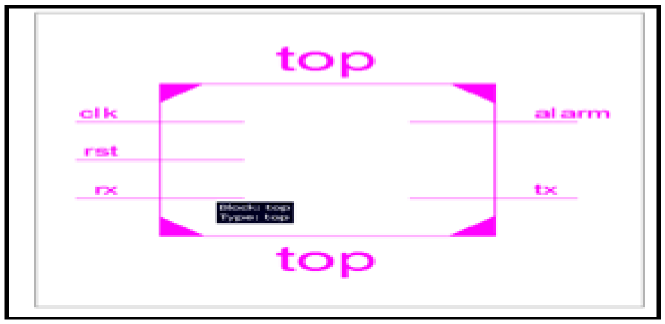


Fig.4.8 RTL schematic of top module block.

Fig 4.8 shows the RTL schematic of the top module. Fig 4.8 shows the expansion of the top module with internal blocks such as processor, cache configuration controller and reconfigurable cache etc.

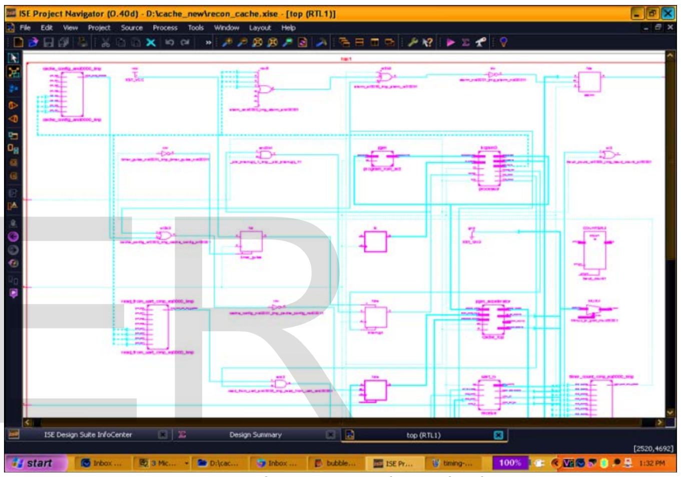
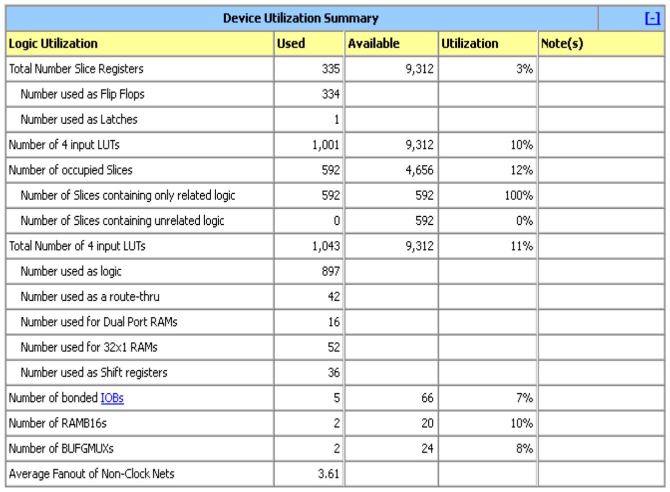


Fig.4.9 RTL schematic view.

Fig 4.10 shows the synthesis report containing no. of LUT blocks utilised for the design.



Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Total Number Slice Registers	335	9,312	3%	
Number used as Flip Flops	334			
Number used as Latches	1			
Number of 4 input LUTs	1,001	9,312	10%	
Number of occupied Slices	592	4,656	12%	
Number of Slices containing only related logic	592	592	100%	
Number of Slices containing unrelated logic	0	592	0%	
Total Number of 4 input LUTs	1,043	9,312	11%	
Number used as logic	897			
Number used as a route-thru	42			
Number used for Dual Port RAMs	16			
Number used for 32x1 RAMs	52			
Number used as Shift registers	36			
Number of bonded IOBs	5	66	7%	
Number of RAMB16s	2	20	10%	
Number of BUFGMUXs	2	24	8%	
Average Fanout of Non-Clock Nets	3.61			

Fig.4.10 Device utilization

5. CONCLUSIONS

The reconfigurable cache with 8 cache modes has been implemented on XILINX Spartan-3E board [14]. In addition a Picoblaze hardware/software platform for control and testing purposes is developed. Some simple testing algorithms have been programmed and the results are discussed in this paper. The dynamic reconfigurable cache can be achieved by taking the miss ratio as the parameter. Whenever miss ratio reaches certain threshold value it can be switched to some other configuration. In this way miss ratio can be frequently monitored and dynamic reconfiguration can be achieved.

6. REFERENCES

[1] Ranganathan, P., Adve, S. and Jouppi, N. P., "Reconfigurable caches and their application to media processing". ACM SIGARCH Computer Architecture News, 2000, 28(2), 214-224. doi: 10.1145/342001.339685,.

[2] H. Kim, A.K. Somani, and A. Tyagi, "A Reconfigurable Multifunction Computing Cache Architecture", IEEE Transactions on VLSI, Vol. 9, No. 4, pp. 509-523, Aug., 2001.

[3] Zhang, C., Vahid, F., & Najjar, W., A highly configurable cache architecture for embedded systems. Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA'03), June, 2003.

[4] Peng, M., Sun, J., & Wang, Y. A Phase-Based Self-Tuning Algorithm for Reconfigurable Cache. First International Conference on the Digital Society (ICDS'07), 27-27. IEEE. doi: 10.1109/ICDS.2007.2, 2007.

[5] Chen, L., Zou, X., Lei, J., & Liu, Z. (2007). Dynamically Reconfigurable Cache for Low-Power Embedded System. Third International Conference on NaturalComputation (ICNC 2007) Vol

[6] V, (Icnc), 180-184. Ieee. doi: 10.1109/ICNC.2007.346.

[7] Albonesi, D. H. "Selective cache ways: on demand cache resource allocation". Journal of Instruction Level Parallelism May. 2002.

[8] Santana Gil, A.D., Benavides, Hernandez, Herruzo.,"Reconfigurable Cache implemented on an FPGA" 2010 International Conference on Reconfigurable Computing.05695314.

[9] CACTI 4.0 Tarjan, David; Thoziyoor, Shyamkumar; Jouppi, Norman P. HPL-2006-86 20060606, 2006. 0

[10] Computation (ICNC 2007) Vol V, (Icnc), 180-184. IEEE. doi: 0.1109/ICNC.2007.346.

[11] Zhang, C., & Vahid, F. Self-tuning cache architecture for embedded systems. ACM Transactions on Embedded Computing System, Vol.3, May. 2004.

[12] Ting, Y., & Chen, B.. Combining selective cache line replacement and active management for data caching. Thesis. 2005.

[13] Balasubramonian, R., & Albonesi, D. Memory Hierarchy Reconfiguration for Energy and Performance in General Purpose Architecture. Proc of 33 rd Intl Sym on Microarchiterture, 245-257, Dec., 2000.

[14] Coutinho L. M., Mendes J. L., Martins C. A., Dynamically Reconfigurable Split Cache Architecture. 2008 International Conference on Reconfigurable Computing and FPGAs, 163-168. IEEE. doi: 10.1109/ReConFig. 2008.46.

[15] www.xilinx.com, 2013

[16] http://www.xilinx.com/support/documentation/ip_documentation/ug129.pdf, 2013