Cache Controller for 4-way Set-Associative Cache Memory

Article *in* International Journal of Computer Applications · November 2015

DOI: 10.5120/ijca2015906787

3 authors, including:

Gurmohan Singh
Centre for Development of Advanced Computing
50 PUBLICATIONS **312** CITATIONS

SEE PROFILE

Cache Controller for 4-way Set-Associative Cache Memory

Praveena Chauan
Student M.Tech VLSI Design
ACS Division, Centre for Development of Advanced Computing (C-DAC), Mohali, 160071, India

Gagandeep Singh
Lecturer
Electronics Communication & Engineering Department, Thapar University, Patiala (India)

Gurmohan Singh
Senior Engineer
DEC Division, Centre for Development of Advanced Computing (C-DAC), Mohali, 160071, India

ABSTRACT
This paper presents design of a cache controller for 4-way set associative cache memory and analyzing the performance in terms of cache hit verses miss rates. An FSM based cache controller has been designed for a 4-way set-associative cache memory of 1K byte with block size of 16 bytes. Main memory of 4K byte has been considered. The synthesis has been performed using Xilinx Synthesis Tool (XST) with Virtex-6 FPGA device *XC6VLX240T*. ISim simulator is used for functional verification of the designed code.The maximum output required time i.e. hold-time after clock is 0.777ns and minimum input time before clock arrival i.e. setup-time for designed module is 1.66ns. The maximum clock frequency is 257.202MHz.The design has also been synthesized in Cadence RTL Compiler tool. Finally, ASIC implementation of the designed cache controller has been done in Cadence Encounter Digital Implementation tool and the GDSII file has been generated.The designed cache controller consumes 5.53mW of total power.

Keywords
Cache Memory, Main Memory, Set-Associative Cache Design, Cache Controller.

1. INTRODUCTION
The data processing capabilities of a processor (also at times termed as processor speed) have been increased at a much faster rate when compared to the memoryperformance or speed in recent years [1]. This trend can be analyzed by looking at the time toperformance graphshown as Moore's law effect in figure 1. In this new generation, the processors are having very large main memory access latency and it has been predicted that it will be increased further [2].

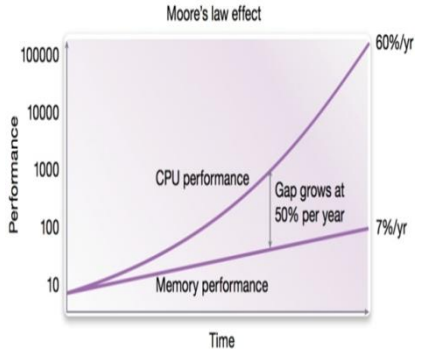

Fig.1: Memory speed lags behind CPU speed [1]

2. CACHE MEMORY
Cache memory is closest to the processor and fastest of all other memories. It is used to store the instructions or data which are frequently required by the processor. It has the highest level in the memory hierarchy. Whenever there is any request from the processor to read or write, cache memory is referenced first. While designing the cache, a lot of attention has to be paid to implement the best configuration as it affects the processor performance. The cache has to be designed such that the data required by the processor should be readily available in it. The time required to access cache should be minimum as well. While designing the cache, few steps have to be followed as shown in figure 2.

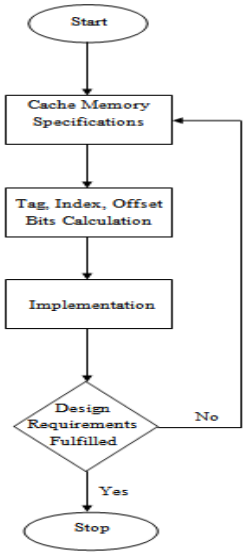

Fig.2: Cache design steps

2.1 Architecture of Cache Memory
The architecture of the cache memory designed is shown in figure 3. The cache memory of 1Kbyte i.e. 1024 bytes have been designed. It is a four-way set associative cache with block size of 16 bytes. There are 16 sets for each way. The address line coming from the processor is of 12 bits. Out of these 12 bits, bits 0 to 3 are called block offset bits as these are required to identify one byte out of 16 bytes of the cache block. The bits 4 to 7 are called index bits and these are used to select one set out of the 16 sets. The remaining four bits are known as tag bits which are used to identify the particular location in the cache which is to be accessed by the processor.

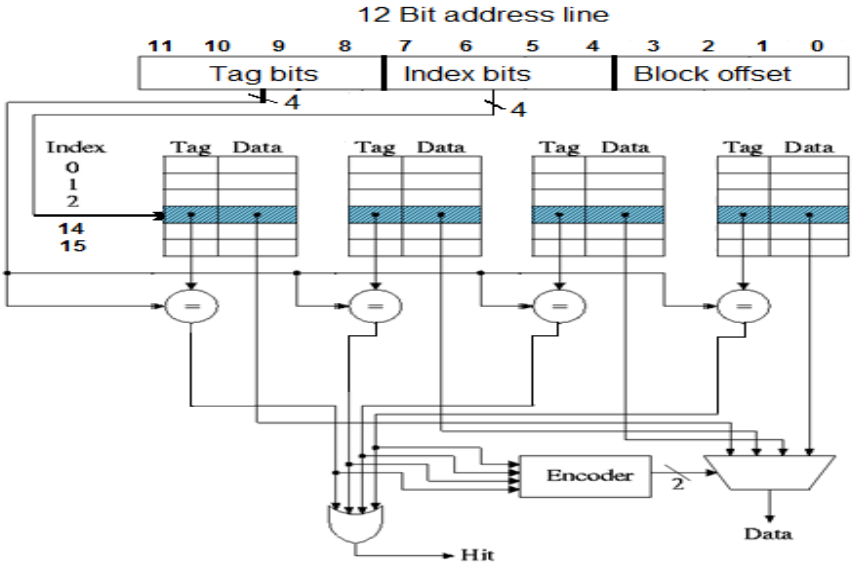

Fig. 3: Architecture of 4-way set-associative cache memory

It has two inputs, one from the processor and other from the main memory. Input from the processor consists of 8 bit data while the input from the main memory consists of 128 bit data. There is one 8-bit output data line which provides data to the processor as well as the main memory.

2.2 Write Operation of Cache Memory

During write operation, if there is a hit from any way, then the set corresponding to that particular way will be selected with the help of index bits e.g. if the index bits of the 12 bit address line is 0000, set number 0 will be selected. If it is a miss, the way will be selected randomly by using a counter as the replacement method. Also as the line size of cache is 16 bytes, where to write in the cache is obtained with the help of block offset bits e.g. if the block offset is 0001, the data would be written to the location of second byte from the right.

2.3 Read Operation of Cache Memory

While reading the cache all the four ways will be enabled. The set, from where the data has to be read, is selected, with the help of index bits extracted from the address bits coming from the processor. The byte which has to be read is selected with the help of block offset bits. Tags present in each way will be compared with the tag bits of the address coming from the processor. Comparator is used for comparing the tag. If the same tag is present in any way of the cache, it is ahit. The data will be fetched from the location corresponding to that particular tag and will be provided to the processor. If the tag is not presentin the cache, it is a miss and the data has to be brought from the main memory.

3. MAIN MEMORY

It is the memory where the data or instruction is actually present. It has lowest level in the memory hierarchy. The data has to be brought into the cache from the main memory as and when required by the processor.

3.1 Architecture of Main Memory

The main memory of 4K byte i.e. 4096 bytes have been designed to check the functionality of cache controller. Each location of main memory consists of 1byte i.e. 8 bits. Hence, there are total 4096 locations in the main memory in each location of memory is accessed by a 12 bit address line. There are two separate data lines for input and output. The input data line is of 8 bits where as the output data line is of 128 bits. The requirement of 128 bit data line for output will be described in the operation. The data has been arranged in the main memory as shown in figure 4.

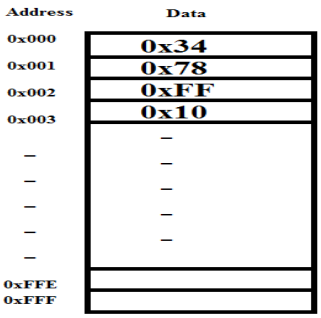

Fig.4: Main memory architecture

3.2 Operation of main memory

The architecture of main memory is simple as compared to cache memory. Whenever there is a miss in the cache during read operation the address coming from the processor will be forwarded to the main memory with the help of cache controller. After the address arrives, the whole block of data consisting of 128 bits will be forwarded to the cache. This 128 bits data not only consists of the required data by the processor, but the neighboring data is also provided to the cache keeping in view the property of locality of the cache. In case of write operation, if there is a hit in cache, the data will be written to the cache as well as main memory by write through policy. If there is a miss, the data will be written directly to the main memory.

4. CACHE CONTROLLER

Cache controller is a device which is used to control the transfer of data between the processor, main memory and the cache memory. When the processor wants to access any location in the main memory by sending the address, cache controller will check for that location in the cache. If the

location is there in the cache, the data will be provided to the processor. If the location is not present in the cache, the cache controller will fetch the data from the main memory and update the cache and provide the data to the processor.

4.1 Architecture of Cache Controller

The cache controller designed here consists of four operations i.e. fetching address from the processor, read cache and main memory, write main memory and cache and providing the required data to the processor. All these operations are implemented using a Finite State Machine. Figure 5 shows the state diagram for the designed cache controller.

4.2 Working of Cache Controller

The cache controller designed here consists of 8 states as shown in figure 5. Description of each state is as follows:

1. **Reset:** It is the initial state of the cache controller. In this state, all the enable inputs to cache as well as main memory are set to zero. All the controlling output from the cache controller will be reset. No operation is performed in this state.
2. **Request from Processor:** Whenever there is any request from the processor to access the memory, cache controller will check whether it is a read request or write request. Depending upon the request, the controller will jump to the desired state.
3. **Read Cache:** If there is a read request, the cache controller will jump to this state. In this state the cache controller will check whether the data is present in cache or not. If data is present in the cache, then the data will be provided to the processor. In case the data is not present in the cache, the cache controller will jump to the next state i.e. read main memory.
4. Read Main Memory: If there is a miss during read operation, the cache controller will jump to this state. In this state, the cache controller will fetch data from the main memory and jump to the next state i.e. bring data from main memory to cache.
5. **Provide data to the processor:** In this state, the cache controller will provide the required data to the processor from the cache memory.
6. **Write Cache:** The cache controller will jump to this state whenever there is a write request from the memory and the required location is available in cache i.e. write hit. In this state, the data will be written in the cache. After writing data in cache, Cache controller will jump to next state i.e. write to main memory.
7. **Write to Main Memory:** Cache controller will jump to this state in two cases i.e. when there is write hit or when there is write miss in cache. In case of write hit, the controller will jump to this state after writing the data into cache as the main memory have to be updated using write through policy. In cache of write miss, the data will be directly written to the main memory without affecting the cache.

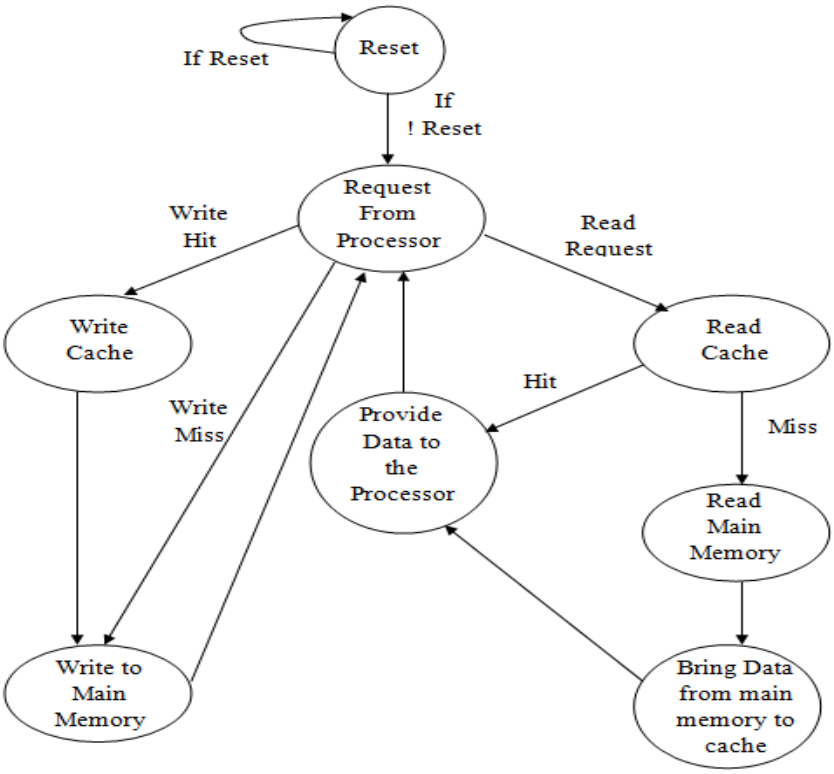

Fig. 5: State Diagram for the cache controller

5. RESULTS AND DISCUSSIONS

All the modules designed for this paper have been simulated to check whether they are performing as required. After simulation, the whole design has been synthesized to obtain its timing summary, device utilization summary, area report and power report. This section presents the tools used, the simulation results and wave forms of all modules designed to show the working of cache controller. Design has been simulated and synthesized using ISE Design Suite. The design has also been synthesized using Cadence RTL Compiler. ASIC implementation of the design has been done in Cadence Encounter Digital Implementation tool.

5.1 Testing the Main Memory

The main memory is tested to check whether it is performing as desired. It is tested for read and write operation.

5.1.1 Simulation waveform of main memory for read operation

The main memory has been enabled to read data from it as *mm_we* is *0* and *mm_re* is *1*as shown in figure 6 and 7 respectively. Also *en_BRAM* has to be kept *1* in order to perform any read or write operation in main memory. The main memory will be referred for read only when there is a miss in the cache. So, not only the data required by the processor will be read but the whole block will be sent to the cache keeping in view the property of spatial locality. The required data will be then sent to the processor by the cache controller. In figure 6, the requested address is *0x001*, the whole block corresponding to addresses *0x000* to *0x00f* will be output of the cache and the required data i.e. *0x29*will be sent to the processor..

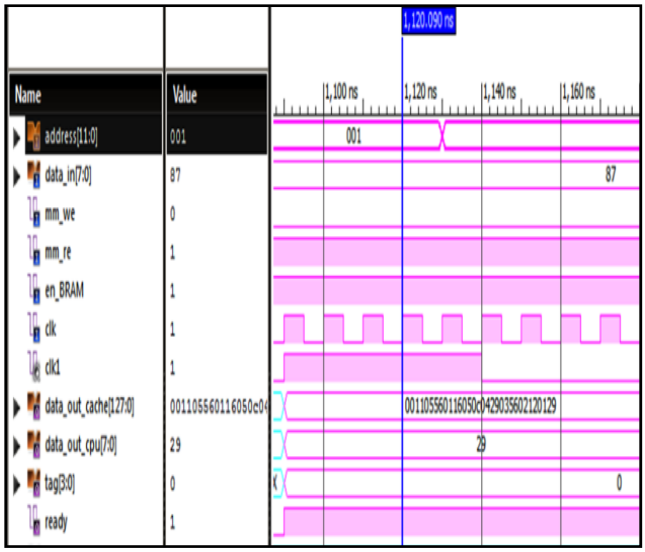

Fig. 6: Simulation waveform for reading from main memory location 0x001

In figure 7, the requested address is *0x020*, so the whole block corresponding to the addresses *0x020* to *0x02f* will be transferred to the cache. Also the ready signal will be set to *1*, when the whole block is transferred during read operation from the main memory. This ready signal is to inform the cache that the data has arrived from the main memory. Along with the data, the tag corresponding to the particular location will also be sent to be updated in the cache memory.

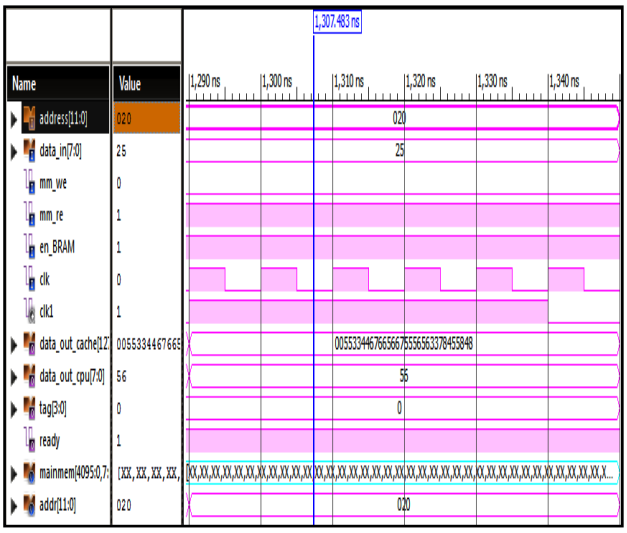

Fig. 7: Simulation waveform for reading from main memory location 0x020

5.1.2 Simulation waveform of main memory for write operation

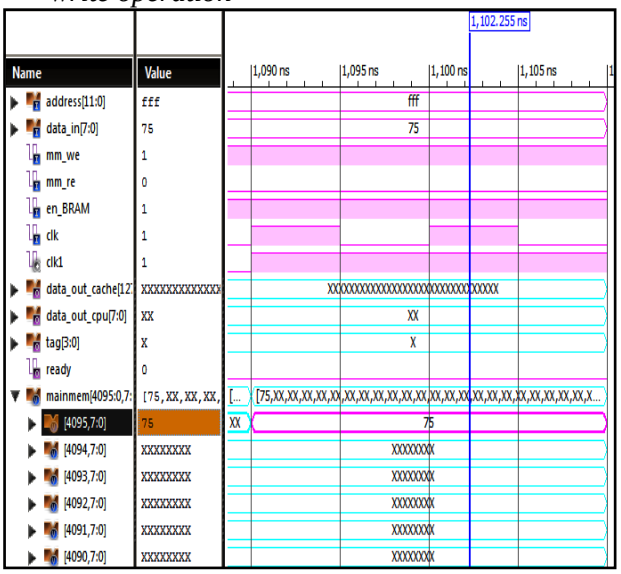

Fig.8: Simulation waveform for writing to main memory location 0xfff

In figure 8, it is a write request as *mm_re* is *0* and *mm_we* is *1*. The data have to be written in the main memory locationcorresponding to address *0xfff*. Here, the data to be written is *0x75*.

5.2 Testing the Cache Memory

The cache memory is tested for its read and write operationduring hit or miss.

5.2.1 Simulation waveform of cache data memory for write hit

During write operation, the cache will be checked for hit or miss by comparing the tags present in the cache with the tag bits of the incoming address. If the tag is present, it is a hit and the data coming from the processor is written to the cache.

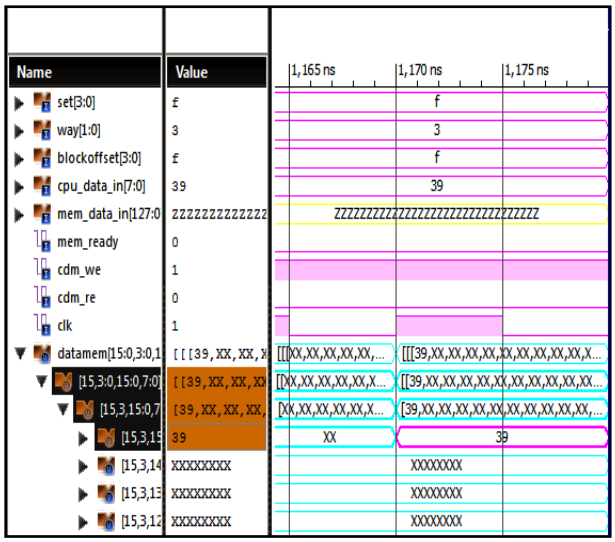

Fig. 9: Simulation waveform for writing to cache memory for write hit

Figure 9 shows the simulation results when it is a write hit. The address from the processor after being divided into three fields i.e. tag, index and block offset by the cachewill be 0. These are the control signals provided by the cache controller.If it is a write request and the requested address where the processor wants to write is not present in the cache, then it is a write miss. In case of write miss, the data will be directly written to the main memory without affecting the cache.

5.2.2 Simulation waveform of cache data memory for read hit

When there is a read hit i.e. the address requested by the processor is present in cache, the required data corresponding to the requested address will be provided at the output as shown in figure 10. Here the requested data is *0xd0*. Also, we see that during read hit *cdm_re* will be *1*, *cdm_we* will be *0* and *mem_ready* will be *0*.

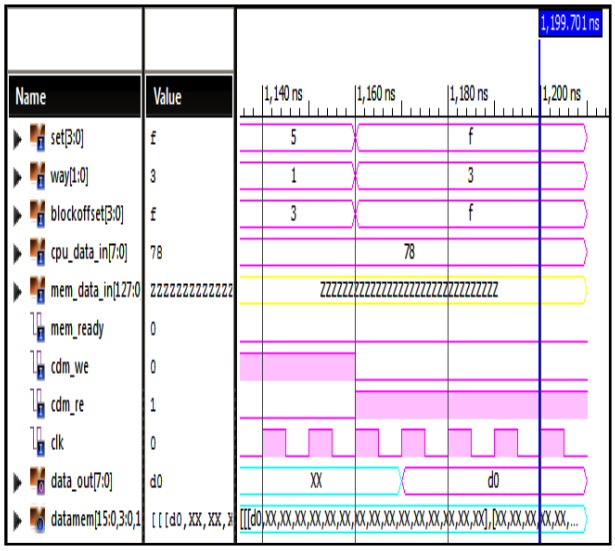

Fig.10: Simulation waveform of cache memory for read hit

5.2.3 Simulation waveform of cache data memory for read miss

If it is a read miss, first the data will be brought from the main memory to the cache and then it will be sent to the processor. While bringing the data from the main memory *cdm_we* will be *1*, *cdm_re* will be *0* and *mem_ready* will be *1* and while sending the data to the processor *cdm_we* will be *0*, *cdm_re* will be *1* and *mem_ready* will be *0*. The whole block which consists of 128 bits will be copied to the cache as shown in figure 11. Here, the requested data is *0x58*.

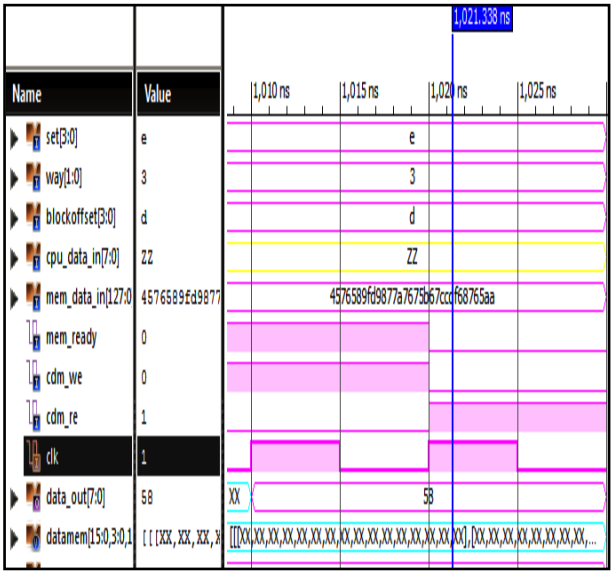

Fig.11: Simulation waveform of cache memory for read miss

5.3 Testing the Cache Controller

The cache controller has been tested for its function of transferring data among processor, cache memory and main memory during hit or miss in cache.

5.3.1 Simulation waveform of cache controller for read miss

Figure 12 shows the simulation waveform for the cache controller designed using a finite state machine. As *Readbar_write* is 0, it is a read operation. The incoming address i.e. *cpu_Address* is divided into three fields i.e. *calc_tag*, *index* and *block_offset*. As the requested address is not present in the cache, *tag_hit* is *0*. So, the controller goes through different states as shown in figure 12 and the data is fetched from the main memory and provided to the processor.During cache miss, the whole block of data is copied from main memory to cache. It takes 9 clock cycles to perform read operation whenever there is a miss. Here, the requested data is *0x87*.

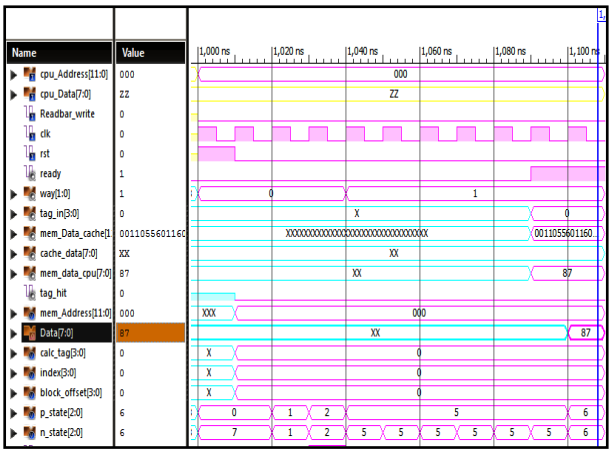

Fig.12: Simulation waveform of cache controller for read miss

5.3.2 Simulation waveform of cache controller for read hit

Figure 13 shows the simulation waveform of cache controller during read hit. The address requested by the processor is present in the cache as *tag_hit* is *1*. So, the data will be fetched from the cache memory by the cache controller andprovided to the processor. The controller go through different states as shown in figure 13 and it takes 4 clock cycles to perform read operation when there is a hit. Here, the requested data is *0x16*.

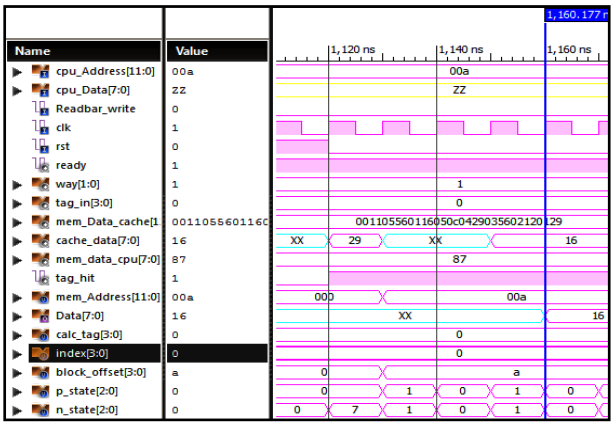

Fig.13: Simulation waveform of cache controller for read hit

5.3.3 Simulation waveform of cache controller for write miss

When there is a miss during write request, the data will be directly written to the main memorywithout affecting the cache. It will take 3 clock cycles to perform this operation and the states are as shown infigure 14. The data written to the main memory is *0x3f*.

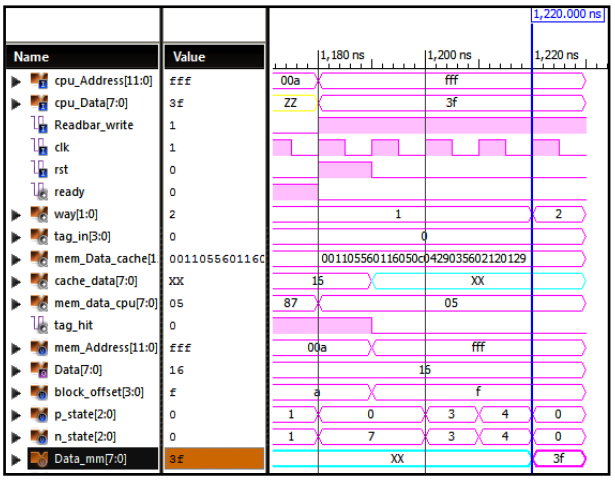

Fig.14: Simulation waveform of cache controller for write miss

5.3.4 Simulation waveform of cache controller for write hit

When there is a hit during write request, the data will be written to the cache at the requested address. It will take 2 clock cycles to perform this operation as shown in figure 15. The data will also be copied to the main memory in the next cycle using write through policy. The data written to the cache is *0xd4*.

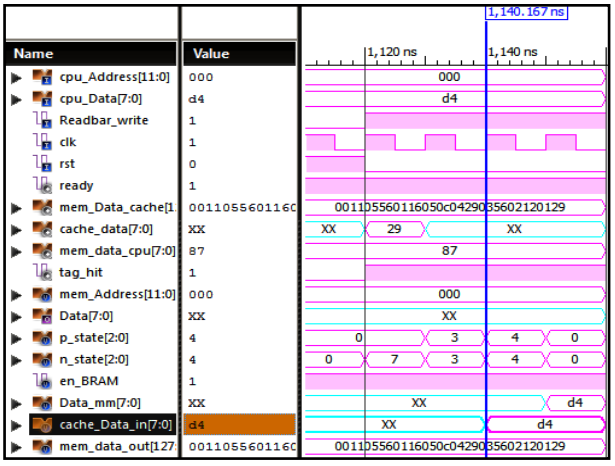

Fig.15: Simulation waveform of cache controller for write hit

5.3.5 Simulation waveform of hit verses miss count

Figure 16 and figure 17 shows the parts of simulation performed to get number of hit verses miss counts ofinstruction cache when a series of operations have been performed and the instructions are fetched from the cache memory. *mem_Address* is the address of the main memory from which data to be operated is fetched. *mem_Address1* is the address of the main memory from where instructions are fetched. 0x070 is the address of the first instruction in the main memory. The address is incremented or changed depending upon the instruction or opcode present in the memory until all the operations have been performed.

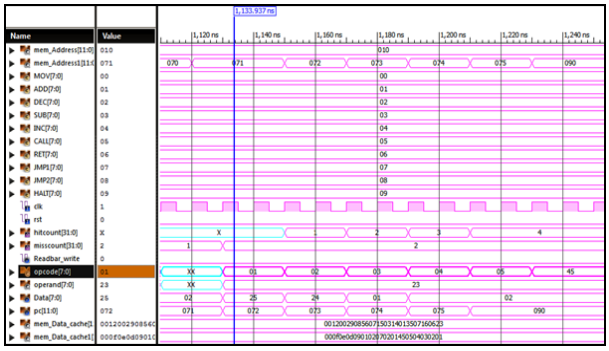

Fig.16: Waveform showing different instructions used to check hit verses miss count

This address is first checked in cache. If it is present than the instruction present in that particular location will be fetched and will be executed. If the address is not available in cache, then it will be fetched from main memory and also copied tothe cache memory. The data present in location *0x010* is copied to a register named Data and the operations are performed on it.

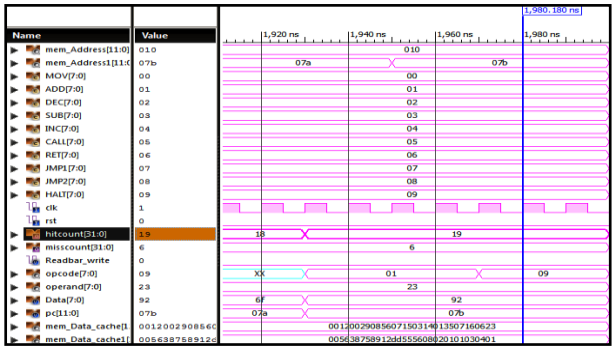

Fig. 17: Simulation waveform showing number of hits and misses in instruction cache

Figure 17 shows the simulation waveform, when all the operations have been performed. It shows that the number of hits and misses in cache after completing the operations are 19 and 6 respectively. From the simulation results it has been noticed that most of the miss were due to jump and call instructions.

5.4 Synthesizing and Implementing the Design

The design has been synthesized using Xilinx Synthesis Tool (XST). Table 1 shows the device utilization summary of cache controller designed. The development board that has been targeted is Virtex-6 ML605 evaluation board.

Table 1: Device utilization summary of the designed cache controller

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	8480	301440	2%
Number of Slice LUTs	21453	150720	14%
Number used as Logic	19917	150720	13%
Number used as Memory	1536	58400	2%
Number of bonded IOBs	31	600	5%
Number of Block RAM	14	416	3%

It shows number of logics available which includes number of Slice Registers, number of slice LUTs, number used as logic, number used as memory, number of bonded IOBs and number of Block RAM. It has been shown that the logic utilization is 2%, 14%, 13%, 2%, 5%, and 3% respectively.

Table 2 shows the timing summary of the cache controller designed.

Table 2:Timing summary of the designed cache controller

Speed Grade	-1
Minimum period	3.888ns
Maximum Frequency	257.202MHz
Minimum input arrival time before clock	1.660ns
Maximum output required time after clock	0.777ns

The design has also been synthesized in Cadence RTL Compiler. Area and power report have been shown in table 3 and 4 respectively.

Table 3:Area report of the complete design

Module	Cells	Cell Area	Net Area	Total Area	Wire Load
Cache Controller	37088	178313	0	178313	None
Cache Data Memory	36283	172342	0	172342	None
Cache Tag Memory	412	3681	0	3681	None
Way Encoder	37	123	0	123	None
Comparator1	9	21	0	21	None
Comparator 2	9	21	0	21	None
Comparator 3	9	21	0	21	None
Comparator 4	9	21	0	21	None
4-Input OR Gate	2	7	0	7	None

Table 4:Power report of the complete design

Module	Cells	Leakage Power (nW)	Dynamic Power (nW)	Total Power (nW)
Cache Controller	37088	2580.462	5530180.321	5532760.783
Cache Data Memory	36283	2474.583	3611353.587	3613828.170
Cache Tag Memory	412	62.983	63849.371	63912.353
Way Encoder	37	1.429	11072.135	11073.564
Comparator1	9	0.321	1286.035	1286.356
Comparator2	9	0.321	1330.024	1330.345
Comparator3	9	0.321	1180.913	1181.235
Comparator4	9	0.321	1030.035	1030.356
4-Input OR Gate	2	0.037	473.082	473.119

6. CONCLUSION

A Cache Controller has been designed for a 4-way set-associative cache memory using Xilinx ISE 14.6 Design Suite in Verilog HDL and it has been found to work successfully with given inputs. In this design, the main memory of 4K byte has been considered. Along with this, a 4-way set-associative cache memory of 1K byte has been designed to check the functionality of the designed cache controller. A test module has also been designed which consists of some instructions to be fetched from the instruction cache. It shows that there were total 19 hits and 6 misses to do a series of operations. It could be seen that most of the misses were due to call and jump instructions.The design has been implemented on Virtex-6 ML605 evaluation board. From the synthesis report, it could be seen that the maximum output required time i.e. hold-time after clock is 0.777ns and minimum input time before clock

arrival i.e. setup-time for designed module is 1.66ns. The maximum clock frequency is 257.202MHz.The device utilization summary showed that minimum resources were consumed.Finally, the ASIC implementation of Cache Controller along with other modules has been done on Cadence Encounter Digital Implementation tool and the GDSII file has been generated. The designed cache controller consumes 5.53mW of total power.

7. REFERENCES

[1] John L. Hennessy and David A. Patterson, Computer Architecture: A Quantitative Approach, 4th ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[2] Maurice V. Wilkes, "The memory gap and the future of high performance memories", *SIGARCH Comput. Archit. News, 29(1),* 2–7 March, 2001.

[3] A. J. Smith, "Cache memories", *ACM Computing Surveys*, vol. 14, no. 3, pp. 473-530, September 1982.

[4] Bani R.R, Mohanty, S.P, Kougianos E, and Thakral G, "Design of a Reconfigurable Embedded Data Cache", *inProc. Int. Symp. Electronic System Design*, pp.163-168, December 2010.

[5] Chuanjun Zhang, Vahid F, and Lysecky R, "A self-tuning cache architecture for embedded systems", *inProc. Europe Conf. and Exhibition on Design, Automation and Test*, vol.1, pp.142,147, 16-20 February 2004.

[6] Frank Vahid and Tony Givargis, Embedded System Design: A Unified Hardware / Software Approach. *John Wiley & Sons*, 2006.

[7] Chenxu Wang, Jiamin Zheng, and Mingyan Yu, "Cache Performance Research for Embedded Processors" *inProc. SciVerse Science Direct Int. Conf. Solid State Devices and Materials Science,* pp.1322-1328, March 2012.

[8] Hassan S.L.M, Johari M.N, Saparon A, Halim I.S.A, and Ab Rahim A.A, "Multi-sized Output Cache Controllers", *in Proc. Int. Conf. Technology, Informatics, Management, Engineering & Environment (TIME-E 2013), Bandung, Indonesia*, pp.186-191, June 2013.

[9] Chuanjun Zhang, "An efficient direct mapped instruction cache for application-specific embedded systems", *inProc. Third IEEE/ACM/IFIP Int. Conf. Hardware/Software Co design and System Synthesis*, pp.45-50, September 2005.

[10] Crisu D, "An architectural survey and modelling of data cache memories in Verilog HDL", *inProc. Int. Semiconductor Conf. CAS '99*, vol.1, pp.139, 142, 1999.

[11] Chuanjun Zhang, Vahid F, and Najjar W, "A highly configurable cache architecture for embedded systems", *inProc. 30th Annual Int. Symp. Computer Architecture*, pp.136-146, June 2003.

[12] Bhure V.S and Padole D, "Design of Cache Controller for Multi-core Systems using Multilevel Scheduling Method", *inProc. Fifth Int. Conf. Emerging Trends in Engineering and Technology (ICETET)*, pp.167-173, November 2012.

[13] Sparsh Mittal, "A survey of architectural techniques for improving cache power efficiency", *Sustainable Computing: Informatics and Systems*vol.4, Issue 1, pp. 33-43,March 2014.

[14] Givargis T, "Improved indexing for cache miss reduction in embedded systems", *inProc. Conf. Design Automation*, pp.875,880, 2-6 June 2003.

[15] Malik A, Moyer B, and Cermak D, "A low power unified cache architecture providing power and performance flexibility", *inProc. Int. Symp. Low Power Electronics and Design (ISLPED)*, pp. 241,243, 2000.

[16] Agarwal A and Pudar S.D, "Column-associative Caches: A Technique For Reducing The Miss Rate Of Direct-mapped Caches", *inProc. 20th Annual Int. Symp. Computer Architecture*, pp.179,190, 16-19 May 1993.

[17] James K.Peckol and Embedded Systems: A Contemporary Design Tool. *John Wiley & Sons*, 2008.

[18] Olukotun K, Mudge T.N, and Brown R.B, "Multilevel optimization of pipelined caches", *IEEE Trans. Computers*, vol.46, no.10, pp.1093-1102, Oct 1997.

[19] Dash A and Petrov P, "Energy-Efficient Cache Coherence for Embedded Multi-Processor Systems through Application-Driven Snoop Filtering", *9th EUROMICRO Conf. Digital System Design: Architectures, Methods and Tools*, pp.79,82, 2006.

[20] Balwant Raj, Anita Suman, Gurmohan Singh, "Analysis of Power Dissipation in DRAM Cells Design for Nanoscale Memories", *International Journal of Information Technology & Knowledge Management*, July-December 2009, Volume-2,No. 2,pp. 371-374.

[21] Karishma Bajaj, Manjit Kaur, Gurmohan Singh," Design and Analysis of Hybrid CMOS SRAM Sense Amplifier, *International Journal of Electronics and Computer Science Engineering*, Volume-1, Number-2, pp. 718-726, 2012.