International Journal of Engineering & Technology, 7 (1.5) (2018) 285-290

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

L2 cache performance analysis for MPSoC by tag – comparision

Jenitha A [1*], Elumalai R [2]

[1]VTU Research Scholar, Bangalore, India
[2]Dept. of Electrical and Electronics, NHCE, Bangalore, India
**Corresponding author E-mail: 28jenitha@gmail.com*

Abstract

Memory systems in many applications are becoming increasingly large, contributing to many challenges in the memory management that has led to many method to manage memory. The tag comparison consumes large amount of cache energy. Current methods provide tag comparison cache or failure of the expected cache. Here is proposed an idea based on new call Comparing Tag stages, filter bloom is presented to improve the efficiency of the cache to predict failure and partial tag comparison for the cold line of verification and full comparison check for direct labels. Moreover, the administration of the cache that is filled with cache lines occurs when there is a cache miss. Today's embedded applications use MPSoC. The MPSoC consists of the following ie more than one processors, shared memory among the processors available and a global off-chip memory. Planning of the activities of an integrated application processor and memory partition between processors are two main critical problem. Here, for an integrated application, both task scheduling and partitioning the integrated available L2 cache to reduce the runtime approach is used.

Keyword: *Bloom filter (BF), Cache, power consumption, tag comparison, Multiprocessor system on chip (MPSoC)*

1. Introduction

Cache is defined as a relatively rapid and relatively small amount of memory face resides between a processor and a relatively larger amount of slow and economic memory, namely the main memory part. Cache is a small fast memory, capable of storing the data used most often. When the processor needs data from main memory, it first checks whether the data requested by the processor is present in the cache. If the data required by the processor in the cache, the cache simply returns the data to the processor. This type of operation allows the processor to prevent access to main memory. Since access to the cache is faster than accessing the main memory, the speed at which the data are processed by the processor increases. Multilevel caches generally operate by controlling the first-level cache smaller than 1 (L1), if there is a success, the processor will proceed by taking the required data from high speed L1 cache. If an error occurs in the smaller cache, checking the next level 2 cache (L2) larger cache, this process will continue even

Then the issue of multi-processor involvement they require Cache coherence i.e. a software or hardware management for look after missing of data or corruption of data, this also leads in increase of power consumption in the L2 Cache energy. Many studies were carried out in reduction of problem associated with the L2 Cache. These studies can be classified into basis of methods for minimizing the tag comparison Cache hit and Cache miss prediction Two step of tag comparison is done for predicting the Cache hit. Firstly Cache ways are compared in the tag for the likely present in the Cache if the prediction is positive consume power for the tag comparison otherwise if prediction fails skip the remaining portion of the tag comparison if not there will be a power consumption is happened even the case of Cache miss which leads to the additional latency cycle along with wastage of L2 Cache power.

Cache is a random access memory which is placed in the between CPU or application and main memory. Cache generally defined as a relatively small amount of fast memory, it is expensive because of its bits per storage value were main memory is inexpensive. Cache tries to store the data which is needed by the processor, if processor request for the data to Cache it simply gives up the data to the processor thus avoids the processor to check with main memory for the data, increased in rate of data processing by the processor increased because Cache provides the data faster than the main memory.

The model system in the multiprocessor chip uses the memory hierarchy with fastest L2 cache on the chip, and an inactive memory hierarchy as slow memory allows the proper allocation of variables between L2 cache than the multi-step labels on the chip and there by reducing access to off-chip memory. Therefore, the execution time of any program by the processor depends on the amount of L2 cache assigned to that processor. Methods of successful prediction commonly used comparing two steps from the cache tag. In the first phase, the labels are compared with cache forms that can control a cache hit problem. If it was a successful prediction, you can save energy required for comparison with other labels.

However, if there is a failure prediction, comparisons are made to label the remaining labels in next cycle, thus leading to a further latency cycle. Because the tag comparison is also performed if the cache is lost there is high energy consumption. The prediction methods attempts to predict the errors of cache misses cache, according to the Bloom filter. If the prediction is correct then the

comparison labels are skipped thus saving energy. Improve the accuracy of forecasting further reduces the amount of energy consumed in the label comparison. However, as stated in the cache miss prediction requires more energy consumption in the head and the chip area.

2. Related work

Koji et al presented a method to predict the cache Predicts the ultimate way in which to choose one to be correct, it means that the cache successfully completed access. If the login fails, the cache will seek other forms.
Powell et al used a method for prediction selectively direct path mapping to reduce the dynamic energy of L1 cache, but the performance keeping.
Z. Zhu et al presented the most recently used multiple method which provides for the most recently used partial label. On the basis of the result of the cache and the cache hit forecasting they are used to design the cache memory with the minimum of energy consumption.
Dai and Wang proposed a method to reduce the L2 cache tag to access the writing process through the L1 cache. With this method, the write cache offers better performance and a good tolerance to soft errors in cache memory chips. When there is a high cache of bankruptcy prediction methods suffer from high struck penalty problem cache. As compensation, the failure prediction method overcomes this limitation cache.
Zhang and others propose a cache architecture called the way to stop the architecture of the cache can reduce the energy without cache memory performance over the head. So cache to stop a 4-way associative cache which stores the lowest four bits of every tags in a fully associative cache memory data, called tag array shutdown is used. Energy can be saved to know mismatch labels and detention.
M Ghosh et al. Implemented a Bloom filter to find a "hit / miss predictor" to identify the juvenile faults cache pipelining.
Keramidas et al. Proposed a filter which combines the decomposition filter Bloom and the concept Cache Decay. The cache predicts cold disintegration of the cache line of the cache memory, that has a effect almost doubled. First, it reduces the number of addresses that use the Bloom filter is necessary to maintain and improve accuracy of prediction of failure based on cache. Bloom filter. If the number of addresses in the Bloom filter is lower, then the accuracy of the estimate is greater.

3. Preliminaries and motivation

3.1 L2 cache implementation

The proposed method is implemented and is illustrated in Figure 1. Which includes Tag Comparison Control (TC), the Bloom Filter (BF), the Tag, the Data array, the Miss State Holding Register (MSHR) and the Write Buffer [5]. The proposed method of multistep tag comparison combines both cache hit and miss prediction. The control logic for tag comparison called tag comparison control takes into account the input address from the L1 cache and determines how many steps it takes to make the comparison tags. It also performs the countdown timeout, ex. updating of the countdown based on the results of the comparison tags. Then tag counter is updated based on the sampled accesses (ie earlier for individually 128 retrieve in our experiments) which is used to minimize the energy value by accessing the local TO counters.
Figure 1 bottom most shows the Miss State Registration Register (MSHR), which plays the proper role in implementation. From this it is consider that MSHR performs the actual functions, e.g. replacing of the message in cache, communication a cache request to main memory and formerly setting the local TO counter of the cache. In presentation to the actual functionalities in the proposed means, the Bloom MSHR filter is updated for both cases of frame loading (ie to restore the exact counters) and further exposure (ie to minimize the counter). With a demand of L2 cache, the desired line arrives at L2 cache, it is then alternately given to L1 cache and also written to the L2 cache. MSHR updates the bloom filter by all of subsequent access to the BF. In these cases, the MSHR request takes the priority everywhere the next one and delays the next one by all of a clock cycle.

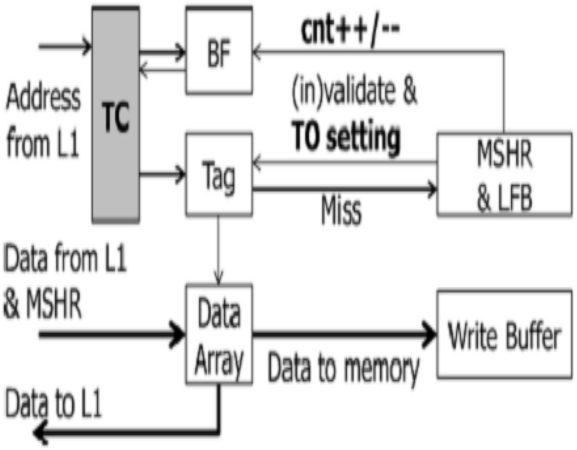

Fig. 1: Implementation of the method

3.2 Multistep method of tag comparison

The proposed method of multistep tag comparison consolidates both cache hit and miss expectation. Its dynamic expression progressively modifies the influence to maximize the flexibility of the cache hit and miss expectations. The evaluate of hot hit prediction to derive out which strategy cut back be applied first.. Fig. 2 shows two feasible courses of action used as a bit of the considered dynamic multistep tag comparison [5]. At low or medium hot hit rates, we can inhere as Fig. 2(a), to what place a gradually labelled Bloom filter (pBF) is alternately associated in break of the article that the amount of aid gets the time to be filtered all Bloom filter (= total access × cache miss rate × cache miss prediction accuracy) increments as the hot hit rate diminishes (i.e., cache miss rate increases). This declines the general label hit or miss which for the approximately pattern executed to give reserve misses as the results interval consuming energy. At valuable hot hit rates, as in Fig. 2(b) shows up, the hot line search is performed earlier the inadequately tagged BF check is finished. The hot line search will closely likely give reserved hits, which permits both resulting Bloom filter checks and tag comparison and reduces the energy consumption.
Threshold of hot hit proportion is performed as a part of the proposed technique (acquired over the study) and is previously compared with the contemporary hot hit ratio (figured over runtime) and before the threshold is utilized to complete on a runtime choice with recognize to which of the couple configurations in Fig. 2 is connected. The best performing threshold changes the programs as the cache access behaviour varies during the programs.

3.3 Partially tag enhanced BF

On account of a singleton, the first filter won't demonstrate whether the practically location is accessible in the cache. A partial tag for each BF participant is proposed to look the correspondence of roughly address on account of singleton entry. Fig. 3 shows a inadequately tagged bloom filter [5].
A BF entry has a tuple , to what place C is the counter, Z is the zero flag, S is the singleton flag, and P is the partial tag. The amount of the partial tag is low, 3 bits

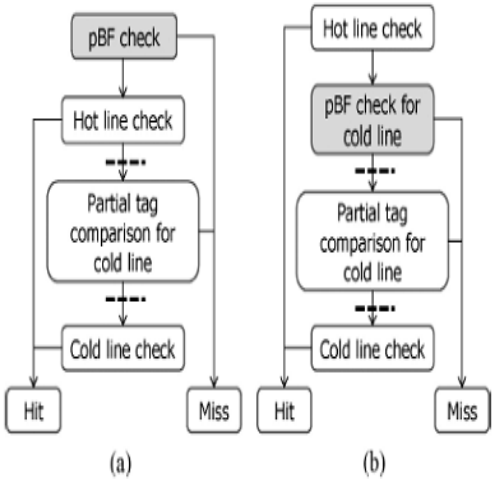

Fig. 2: Hot Hit Ratio Based Multistep Tag Comparisons (a) Medium/low Hot Hit Ratio. (b) High Hot Hit Ratio

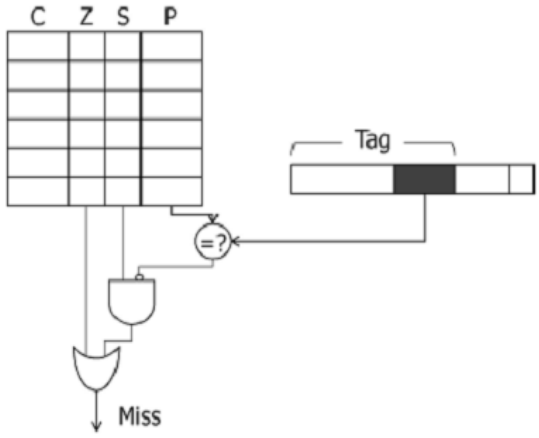

Fig. 3: Bloom filter with a partial tag

Fig. 3 shows a somewhat tagged counting bloom filter. A BF participant has a tuple , to what place C represents the counter, Z represents the zero flag, S represents the singleton flag, and P represents the partial tag. The measure of the partial tag is 3 bits which is small. Thus, compared to the original Bloom filter, the partial tag-enhanced.
Bloom filter has an overhead of 4 bits (including the S flag) per entry. On each entry/exit (program/de- program) of study to/from the
Bloom filter, the corresponding partial tag is expected in bitwise XOR operations as follows:

P Tag new = P Tag old XOR P Tag in //BF participant (program)

P Tag new = P Tag old XOR P Tag out //BF quit (de-program)

where P Tag old and P Tag new represents the old and newly calculated partial tags, respectively. P Tag in and P Tag out characterize the partial tags of the incoming (i.e., newly fetched) and coming (i.e., evicted) cache lines, respectively. Such a partial tag manipulation gives the partial tag of the currently existing address in the situation of a singleton entry.
A pseudo code of the inadequately tagged bloom filter operation when k=1 is shown in the Fig.4.

```
1  BF_query() {            // for each BF query
2    If Z=1, return 'miss'
3    Else
4      If S=1 & partial tag mismatch, return 'miss'
5      Else, return 'likely hit'
6  }
7  BF_entry/exit() {       // for each BF entry/exit
8    Counter++ for entry (Counter-- for exit)
9    If Counter = 0, Z=1
10   If Counter = 1, S=1
11   Partial tag calculation
12 }
```

Fig. 4: Partial tag enhanced bloom filter operation

4. Proposed algorithm

4.1 Architecture overview

Fig. 5 demonstrates the study of MPSoC which consist of of the accompanying components forthcoming specific disparate processors, a commonplace L2 reserve partitioned bounded by the untold processors and a overall off-chip memory that can be used by these processors [4]. This procedure can likewise be utilized for architecture where all processor has a isolated L2 cache with the end goal that every processor can earn to the L2 cache of different processors.

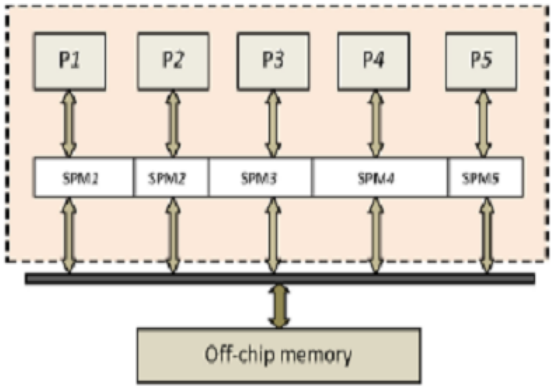

Fig. 5: Architectural model of five processors

Each embedded investigation can be abandoned into an situation of tasks where the accessible processors respond at far one individualistic tasks in parallel. This is amazingly valuable in MPSoC and leads to overcome the execution time.

4.2 Task dependence sketch (TDG)

A design is a nodes and edges is that point the nodes together. A TDG is a on the way to acyclic graph where each of the vertex is expected as a task in entire of the embedded application and with weighted edges. Ti and Tj are two different tasks in the TDG. Fig. 6 shows a task dependence graph [4]. An edge from Ti to Tj always represents a scheduling order where Tj is executed only when necessary data is obtained from Ti after its execution.
The execution of task Tj cannot be started unless all the necessary data communication is carried out.
Fig. 6 shows a TDG

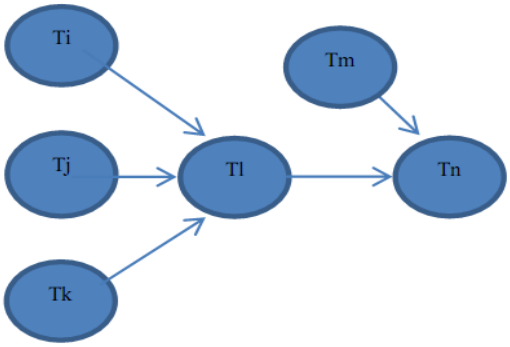

Fig. 6: Task dependence graph

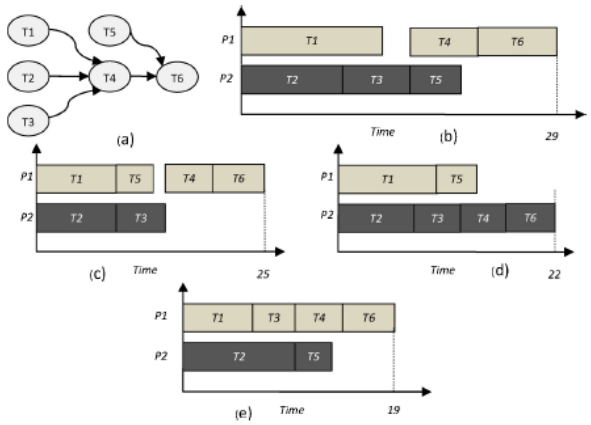

Fig. 7: (a) TDG. Schedule based on (b) no L2 cache, (c) equal partitioned L2 cache, (d) non-equal partitioned L2 cache,
(e) integrated approach

The tasks are eventual by all of approximate divided L2 cache surrounded by the two processors that are accessible as appeared in Fig. 7(c). In this group of scheduling the L2 cache is separated similarly in the two processors P1 and P2 after all of the tasks that are mapped to the processors. Here tasks T1 is mapped to processor P1 and undertaking T2 to processor P2. After the execution of task T2, task T3 is subject to to processor P2. Since task T4 is executed practically after the finishing of the tasks T1, T2 and T3, T5 is mapped to P1 processor after the execution of taskT1. At last task T4 and task T6 are given to processor P1 [4].
Fig. 7(d) demonstrates the scheduling of tasks which gave a pink slip be utilized to additionally abate the applications computational time by partitioning the discernible L2 cache in complete proportion between the evident processors [4]. In the TDG tasks are eventual with the end goal that task T4 begins its execution simply subsequent to finishing the execution of task T3 by processor P2. So there is a dead time between task T1 and task T4.
The principle issue is to abate the precisely time by forcing the calculation time of task T2 and task T3. The solution utilized that to diminish the straight time is to allocate more L2 cache to processor P2 dependent in mind the end goal to decrease the calculation time of task T2 and task T3. On the off chance that generally told the L2 savings is situated P2 before the calculation time of T1 will increment and accordingly the base am a native of time of T4 will increment. To shuffle this sprinkling L2 something for a rainy day is allocated to P1 to adjudicate the execution time.
Here the tasks are planned for a well known a by the number, to the relate that T1 is supposing to P1 and T2 to P2. Since the execution time of T2 is not as for all practical purposes as that of T1, T3 is mapped to P2. After the execution. Initially the review or processor require the reference to L2 Cache. At that involve tag stylistic device is satisfied with Bloom filter which gives miss or incompletely hit at that point tag comparision for partially has a look see is done earlier the fractional search for blah line is done completely and the tag comparision happens in the last. Bloom filter is the eventuality checker which tells whether the fit is characterize the everything or not. This gives the show has a "true" or "false", true for the data is probably present in the total , false for the data completely not a part of the group.
Here the use of Bloom filter for checking the tag-comparison. the incoming address were loaded to the Bloom filter in the form of hash functions Bloom filter checks or compares with the incoming request tag address if it matches it will say Cache hit partially, if does not matches it says Cache miss that by avoiding the remaining tag line comparison of the data in next cycle and Bloom filter check reduces additional latency time and the energy consumption of the L2 Cache.

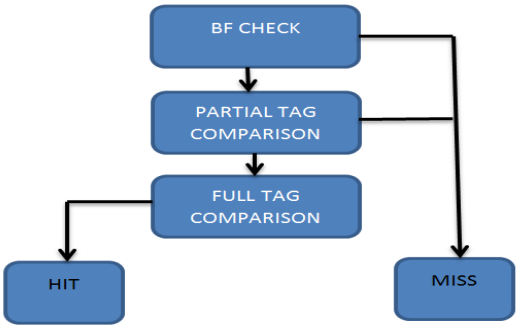

Fig. 8: Block Diagram of Tag Comparision

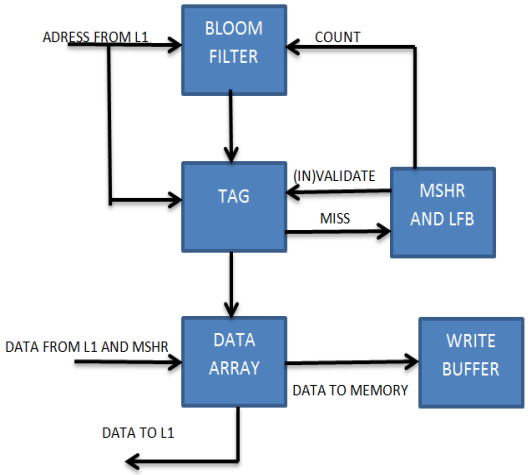

Fig. 9: Block diagram of RAM

5. Implementation

The proposed multistep method combines cache label comparison of prediction and cache miss prediction methods (Bloom filter with greater label). This approach increases the failure prediction cache. Bloom filter tester is likely that indicates whether the set belongs to the group or not. This is if the result is true or false, if the condition is true means that data are probably in the group, if the condition is false, the data are not fully part of the group. Here the use of the filter is used to control flowering comparison between the labels. The input address is loaded into the bloom filter in the form of hash functions, then the filter control or flowering compared with the tag address of the incoming call if it matches partially cache hit, if the results do not match the cached avoiding miss comparing the remaining data tag line in the next cycle and the bloom filter control further reduces the latency time and L2 cache consumption.

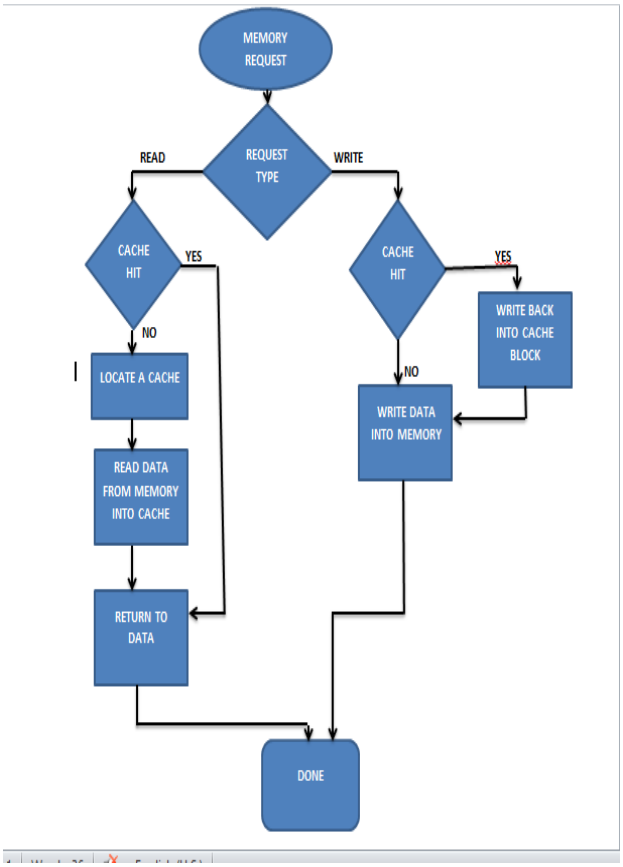

Fig. 10: Flow diagram of L2 Cache System

6. Experimental results

XILINX ISE 14.1 is used for the simulation of L2 cache architecture for MPSoC

Task Scheduling and Memory Partitioning

1. Divide the application into tasks T_i.
2. Perform dependence analysis between tasks.
3. Construct the *TDG* based on dependence analysis and communication costs.
4. Divide the SPM memory equally between the processors.
5. **For** each task T_i and processor P_j, extract the following:
6. (i) Minimum computation time on P_j, Min_{ij}.
7. (ii) Maximum computation time on P_j, Max_{ij}.
8. (iii) Average computation time on P_j, Avg_{ij}.
9. Find ASAP for all the tasks based on *Avg* values.
10. L_1 = List of tasks in increasing order of ASAP.
11. **While** (L_1 not empty) **do**:
12. Get the first task T_f from L_1.
13. **For** each processor P_k:
14. Calculate the *elasticity* and *PEC* of P_k if T_f is mapped to P_k.
15. Find the minimum start time of T_f on P_k.
16. Find $END\_time(P_k)$ if T_f is mapped to P_k.
17. **if** $((END\_time(P_k) < min \;\&\&\; PEC(P_j) \geq (1 - \delta\%)PEC(P_k)) \|$ $(END\_time(P_k) > min \;\&\&\; PEC(P_k) \leq (1 - \delta\%)PEC(P_j)))$
18. $min = END\_time(P_k)$
19. **else if** $(END\_time(P_k) = min)$
20. min = END_time of processor with the higher *elasticity*.
21. **End For**
22. Assign T_f to P_j corresponding to *min*.
23. Delete T_f from L_1.
24. Call Balance().
25. **End While**
26. **For** i = 1 to *t* **do**:
27. Call Balance().

Fig. 11: Integrated scheduling heuristic

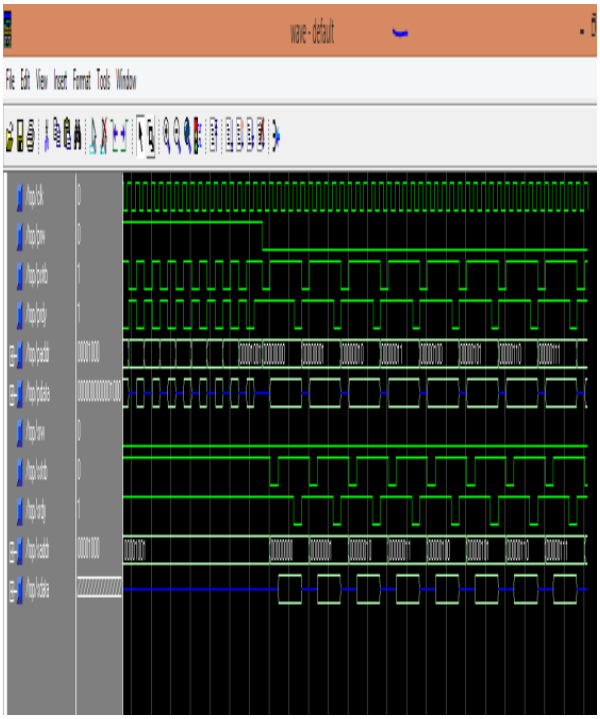

Fig. 12: Simulation result

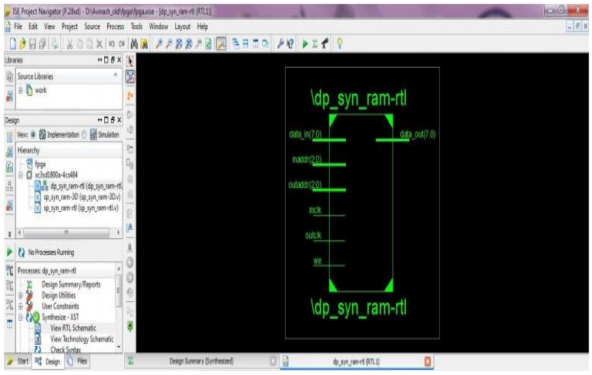

Fig. 13. Schematic diagram

```
run
#          5060: Read hit to set 3
#          5100: Reading from addr=03
run
#          5140: Read hit to set 3
#          5180: Reading from addr=04
#          5220: Read hit to set 3
run
#          5260: Reading from addr=05
#          5300: Read hit to set 3
run
#          5340: Reading from addr=06
#          5380: Read hit to set 3
#          5420: Reading from addr=07
run
#          5460: Read hit to set 3
#          5500: Reading from addr=08
run
#          5540: Read hit to set 3
#          5580: Reading from addr=09
#          5620: Read hit to set 3
run
# Read/Write test done
# ** Note: $stop    : C:/Modeltech_5.7f/examples/proc.v(87)
#    Time: 5660 ns  Iteration: 0  Instance: /top/p
# Break at C:/Modeltech_5.7f/examples/proc.v line 87
run
#          5700: Starting Read/Write test
#          5700: Writing data=0000 to addr=00
#          5740: Write hit to set 3
run
run
#          5900: Writing data=0001 to addr=01
#          5940: Write hit to set 3
run
run
#          6100: Writing data=0002 to addr=02
#          6140: Write hit to set 3
run
run
#          6300: Writing data=0003 to addr=03
#          6340: Write hit to set 3
run
run
#          6500: Writing data=0004 to addr=04
#          6540: Write hit to set 3
```

7. Conclusion

Bloom filter tester is likely that indicates whether the set belongs to the group or not. This is if the result is true or false, if the condition is true means that data are probably in the group, if the condition is false, the data are not fully part of the group. Here the use of the filter is used to control flowering comparison between the labels. The input address is loaded into the bloom filter in the form of hash functions, then the filter control or flowering compared with the tag address of the incoming call if it matches partially cache hit, if the results do not match the cached avoiding miss comparing the remaining data tag line in the next cycle and the bloom filter control further reduces the latency time and L2 cache consumption.

References

[1] Qualcomm, Inc. (2004). Snapdragon Dual Core CPU Processor [Online]. Available: http://www.Qualcomm.Com/Snapdragon

[2] AnandTech. (2011, Mar. 19). The Apple iPad2 Review [Online]. Available: http://www.anandtech.com/show/4225/the-ipad-2-review/4

[3] ARM Ltd. (2007). PL310 Cache ControllerTechnical Reference Manual [Online]. Available: http://infocenter.arm.com [4] ARM Ltd. (2011). CoreLink CCI-400 Cache Coherent Interconnect (CCI) [Online]. Available: http://www.arm.com

[4] K. Aisopos, C. Chou, and L. Peh, "Extending open core protocol to support system-level Cache coherence," in Proc. CODES+ISSS, 2008, pp. 167–172.

[5] K. Inoue, T. Ishihara, and K. Murakami, "Way-predicting set-associative Cache for high performance and low energy consumption," in Proc. ISLPED, 1999, pp. 273–275.

[6] M. D. Powell, A. Agarwal, T. N. Vijaykumar, M. Falsafi, and K. Roy, "Reducing set-associative Cache energy via way-prediction and selective direct-mapping," in Proc. Int. Symp. Microarchitecture, 2001, pp. 54–65.

[7] Z. Zhu and X. Zhang, "Access-mode predictions for low-power cache design," IEEE Micro, vol. 22, no. 2, pp. 58–71, Mar.–Apr. 2002.

[8] C. Zhang, F. Vahid, J. Yang, and W. Najjar, "A way-halting cache for low-energy high-performance systems," in Proc. ISLPED, 2004, pp. 126–131.

[9] J.-K. Peir, S.-C. Lai, S.-L. Lu, J. Stark, and K. Lai, "Bloom filtering cache misses for accurate data speculation and prefetching," in Proc. Supercomputing, 2002, pp. 189–198.

[10] M. Ghosh, [4] Z. Zhu and X. Zhang, "Access-mode predictions for low-power cache design," IEEE Micro, vol. 22, no. 2, pp. 58–71, Mar.–Apr. 2002.

[11] G. Keramidas, P. Xekalakis, and S. Kaxiras, "Applying Decay to reduce dynamic power in set-associative caches," in Proc. Int. Conf. High- Performance Embedded Architectures Compilers, 2007, pp. 38–53.

[12] L. Benini, D. Bertozzi, A. Guerri, and M. Milano, "Allocation and scheduling for MPSOC via decomposition and no-good generation," in Proc. IJCAI, 2005, pp. 107–121.

[13] P. Panda, N. Dutt, and A. Nicolau, MemoryIssues in Embedded Systemson- Chip: Optimization and Exploration. Dordrecht, The Netherlands: Kluwer, 1999.

[14] P. Panda, N. D. Dutt, and A. Nicolau, "On-chip vs. off-chip memory: The data partitioning problem in embedded processor-based systems," ACM Trans. Des. Automat. Electron. Syst., vol. 5, no. 3, pp. 682–704, Jul. 2000.

[15] Rajesh, M., and J. M. Gnanasekar. & quot; Constructing Well-Organized Wireless Sensor Networks with Low-Level Identification." World Engineering & Applied Sciences Journal 7.1 (2016).

[16] S.V.Manikanthan and T.Padmapriya "Recent Trends In M2m Communications In 4g Networks And Evolution Towards 5g", International Journal of Pure and Applied Mathematics, ISSN NO:1314-3395, Vol-115, Issue -8, Sep 2017.

[17] S.V.Manikanthan and K.srividhya "An Android based secure access control using ARM and cloud computing", Published in: Electronics and Communication Systems (ICECS), 2015 2nd International Conference on 26-27 Feb. 2015,Publisher: IEEE,DOI: 10.1109/ECS.2015.7124833.

[18] T. Padmapriya and V. Saminadan, "Improving Throughput for Downlink Multi user MIMO-LTE Advanced Networks using SINR approximation and Hierarchical CSI feedback", International Journal of Mobile Design Network and Innovation-Inderscience Publisher, ISSN : 1744-2850 vol. 6, no.1, pp. 14-23, May 2015.