# Low Power Branch Predictor with Trace Reuse Cache for Embedded Processors

Tanjore Sukumar, Jaikrishna    Krishnamurthy, Sailesh Bharathwaaj    Mehrotra, Shreya    Rajora, Shiv

***Abstract*—Instruction cache accesses devour a significant amount of the total consumed processor power. Numerous methods have been proposed to improve the efficiency of instruction delivery. One significant method involves using a Trace Reuse (TR) Cache as an alternative source for instruction delivery [1]. This method employs an efficient scheme to reuse retired instructions from the pipeline back-end of the processor using a History Trace Buffer and provides significant energy conservation. However, it only uses a simple not taken branch predictor that compromises on energy efficiency due to a relatively high mis-prediction rate. The accuracy of branch predictors greatly determines the efficiency of embedded microprocessors and the prediction algorithm employed, significantly affects the amount of energy consumed. In this project, we propose a model where we employ a low power branch prediction algorithm [2] in the aforementioned method involving the Trace Reuse Cache to improve the overall energy efficiency of the processor. To evaluate the performance of our solution we will also incorporate a perfect branch predictor and a not taken branch predictor with the TR Cache method and then perform a comparison between the different combinations of all these methods.**

## I. Introduction

Embedded systems are growing in complexity and the performance of embedded processors needs to increase. But at the same time, embedded systems are constrained by energy requirements since many of them are battery powered. Hence, there is a growing need to realize processors with higher energy efficiency. Of all the power hungry processes, Instruction cache accesses devour a significant amount of the total consumed processor power. Also, The accuracy of branch predictors greatly determines the efficiency of embedded microprocessors and the prediction algorithm employed, significantly affects the amount of energy consumed. Hence, in this project, we propose a model where we employ a low power branch prediction algorithm [2] that uses a Taken Branch Identification Table (TBIT) to improve the prediction of branches, along with an algorithm that utilizes a Trace Reuse Cache to reuse retired instructions from the pipeline back-end of the processor using a History Trace Buffer[1]. We have evaluated the performance of our solution by incorporating a perfect branch predictor and a not taken branch predictor with the TR Cache method and then perform a comparison between the different combinations of all these methods. The MIBench benchmark was used for evaluating our proposed algorithm and has showed quite promising results.

## II. Motivation

Designing power efficient machines has been a growing need in the recent days. On closer look at the power consuming processes, it is evident that the cache accesses and the effectiveness of the branch prediction algorithm greatly determine a bigger part of the processor power consumption. Instruction caches use up to 27% of the processor power [3]. Also, cache misses increase energy consumption and reduce performance. Hence we have incorporated a technique that involves the addition of a trace reuse cache which increases cache hit-rate thus reducing the loss in energy due to cache misses. It also improves Instructions per Count (IPC) without penalizing energy efficiency. A branch predictor consumes 10% of a processor power in[2] and considering the market expansion in embedded mobile devices, reduction of the power consumed by the branch predictor is becoming an important research area. Unnecessary predictor accesses consume a large amount of power. Of the various types of branch prediction algorithms, the low power branch predictor which utilizes the TBIT table [2] is one of the power efficient algorithms and hence we have incorporated it. Our project is based on the fact that having power efficient algorithms for cache accesses and branch prediction will definitely show a drastic decrease in the total power consumption of the processor. Also, we have also implemented the Trace Reuse Cache technique with other branch predictors like perfect and not-taken to provide a comprehensive result on its capabilities. All the aforementioned techniques have been evaluated using the MIBench benchmark and results have been tabulated.

## III. Related Work

To boost the processor performance it is necessary to improve the efficiency of the instruction delivery. For better energy efficiency a filter cache system has been proposed [4]. Essentially, a filter cache is a small level 0 cache that resides closest to the CPU and it holds the most frequently used instructions to improve energy efficiency. There is however a trade-off between energy efficiency and performance. The previous works have focused on either increasing the performance or the energy efficiency of the instruction delivery. This proposed architecture uses a Trace Reuse (TR) cache that combines a history trace buffer (HTB) along with a Trace Entry Table (TET) at the back-end [1]. The HTB is a FIFO buffer where the retired instructions from the pipeline back-end are captured. The TET is a hardware module that enables easy and fast access to the HTB for fetching instructions. The memory hierarchy of the TR cache is same as that of the conventional instruction cache so cache latency is unaffected. The TR cache uses far less energy than the instruction cache due to reduced size and complexity. In order to predict the correct trace, this paper employs a simple not taken branch predictor to keep the complexity low. Branch Prediction reduces the control hazard in pipelining of processors. This research proposes a low power branch predictor for em-bedded processors [2]. The proposed branch predictor consists of a small cache which is the Branch target buffer (BTB) and a direction predictor (DP. The previous branch target addresses are contained in the BTB. A taken-branch identification table (TBIT) has been used and the well behaved taken branches (WBTB) have been exploited to reduce power. In a branch predictor data entry and table lookup are the main operations within the BTB and the DP which occur at every branch instruction. TBIT maintains the branch table and its entry consists of tag, taken counter(TC), Always taken(AT) and target address fields.

## IV. Approach

Our approach focuses on solving the power consumption problem by incorporating a power efficient cache access technique and a low power branch prediction algorithm together so as to produce maximum power efficiency.

(i) Low Power Branch Predictor:

A low power branch predictor exploits the well-behaved branches. Well-behaved branches are loops which show continuous Taken branch results. Our method is based on the Taken Branch Identification Table (TBIT). It maintains well-behaved taken-branch data that are updated by a 2-bit saturated counter learning algorithm. Through the TBIT, we eliminate unnecessary lookups and updates of a predictor when a well-behaved taken-branch instruction is executed repeatedly such as in a loop[2].

Most branch predictors consist of the branch target buffer(BTB) for target address prediction. Power consumption of a branch predictor can be reduced by using smaller BTB structure or filtering out unnecessary accesses to these predictor components. Adding extra circuitry may introduce extra power consumption into the processor and it may decrease predictor accuracy. Therefore, when implementing the predictor access control method, we should consider those two factors not to overshadow the advantages of the control circuit.[2]

This method focuses on one important point. Simply identifying non-branch instructions during fetch cycle and allowing only branch instructions to access the predictor can greatly save energy consumption[1]in[2]. A table-based predictor filter named Taken-Branch Identification Table(TBIT) is implemented for the branch predictor access control. We exploit the well-behaved taken-branches(WBTB) such as in loops for the power reduction. Those taken-branches can be accessed from the smaller table instead of the whole BTB structure[2].

Taken Branch Identification Table:

This method primarily relies on reducing the number of lookups and updates in branch prediction. For this, a new data

TABLE I
Entry of a Taken Branch Identification Table(TBIT)

| **TAG** | **TC** | **AT** | **Target Address** |
|---|---|---|---|
| 8 bit | 2 bit | 2 bit | 32 bit |

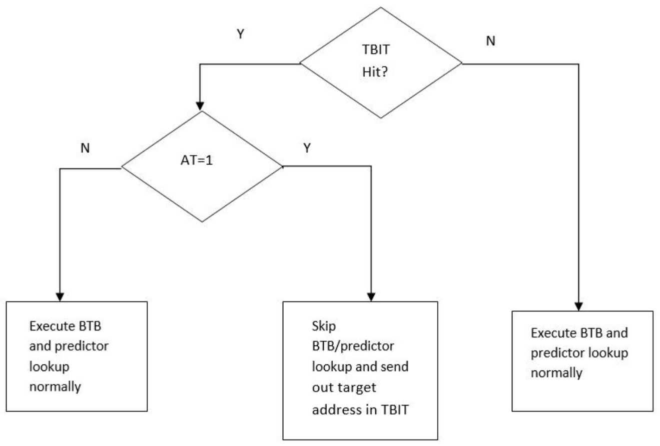
TBIT Hit?
Y
N
AT=1
N
Y
Execute BTB and predictor lookup normally
Skip BTB/predictor lookup and send out target address in TBIT
Execute BTB and predictor lookup normally

Fig. 1. Lookup access reduction with AT field

structure called TBIT is proposed. One TBIT entry consists of Tag, Taken Counter(TC), Always Taken(AT), and Target Address fields as shown in Table 1.

If AT value is 1, BTB and DP are not accessed. Simply TBITs target address field is checked. Thus our branch predictor accesses a much smaller table.

This technique exploits the temporal locality of memory locations requested by the process. Since branch instructions show strong temporal locality[2] this technique theoretically reduces the number of BTB accesses by a considerable amount. Figure 1 provides a brief depiction of how a branch in this technique is handled and how it reduces the lookup access in BTB, thereby saving power. The AT field is made 1 after two hits on the same address. Whenever there is a TBIT hit and the AT field is 1, the BTB lookup is skipped and target address is sent from the TBIT table. BTB normal lookup is executed for all other cases.

Figure 2 depicts a clear picture of how this technique treats a branch. There are four operational cases. Case 1 is the learning phase of the branch where the TBIT table is updated. Case 2 is the removal of unnecessary updates where the actual power reduction is achieved. Case 3 is the addition of TBIT entries where new entries are created in the table. Case 4 is the deletion of TBIT entries where obsolete entries are removed.

(ii) Trace Reuse Cache

The Trace Reuse Cache which is a source for delivery of instructions has been proposed. It improves energy efficiency of instructions in embedded processors. It consists of a History trace buffer (HTB) and a Trace entry table (TET). The HTB gets instructions from the pipeline back-end of the

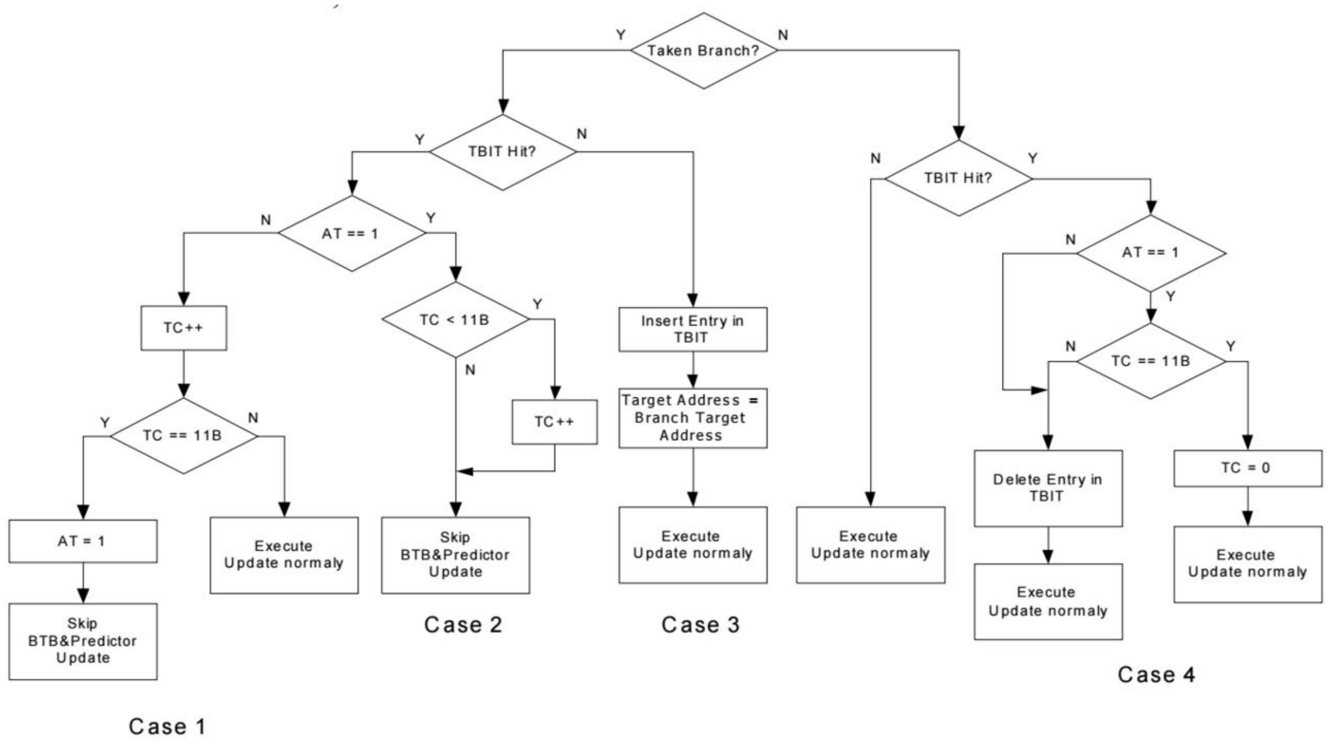

Fig. 2. TBIT maintenance and update access reduction

processor[1] and the TET provides fast access to trace buffer. There is no added cache latency because the proposed TR cache is present in the same level of memory hierarchy as the conventional cache Fig. 3 shows the flow of instruction delivery; The processor is in cache mode by default, and for each cache-mode cycle the TET is searched in parallel with the cache. It is assumed that the TET search can be completed in a single cycle. The processor remains in cache mode until there is a hit by TET. On a hit the processor switches to TRC mode in the next cycle.

## V. Evaluation Methodology

In our method the evaluation tools mentioned in the Experimental Setup will be used to compare processor energy efficiency and performance for the following scenarios:

1) No TR Cache or Low Power Branch Predictor (Only Not taken branch predictor)
2) Only Low power branch predictor
3) TR Cache and theoretical perfect branch predictor
4) TR Cache and bimod branch predictor
5) TR Cache and not taken branch predictor
6) TR Cache and Low Power Branch Predictor

Energy saving is computed by calculating the power consumption in each of the aforementioned methods and comparing. Power is calculated by modifying the power.c code in the WATTCH project [8] (a modification of simplescalar) for each of our combination of techniques. The power was modelled using CACTI [6]. Also, the total cache hit and miss rates and the branch predictor hit and miss rates are monitored to determine a an efficient combination among the above mentioned techniques.

## VI. Experimental Setup

Our main motive is to incorporate power saving cache access and branch prediction techniques together to achieve maximum power efficiency.

The project was primarily modelled using Simplescalar/PISA [5]. The PISA architecture was used as it is a simple MIPS based architecture and it closely resembles the low power architectures used in embedded systems.

We used the MIBench Benchmark for our analysis. MIBench is a free, commercially representative embedded benchmark suite. It also exhibits a wider behavioral range suitable for embedded applications. We have cross-compiled the benchmark for the PISA architecture using a gcc cross-compiler. There are different program groups in MIBench. The WATTCH software extension of simplescalar sim-outorder was used to model and measure power consumption[8]. We used the automotive subset of MIBench as it has a set of benchmarks that encompass various computational scenarios that low power high performance embedded systems face. The results were averaged across the various benchmarks in this benchmark suite.

To perform out experiments we had to model the TR Cache and the Low Power Branch Predictor and their energy profiles in simplescalar. We heavily modified the sim-outorder.c, power.c, cache.c and bpred.c source files to apply

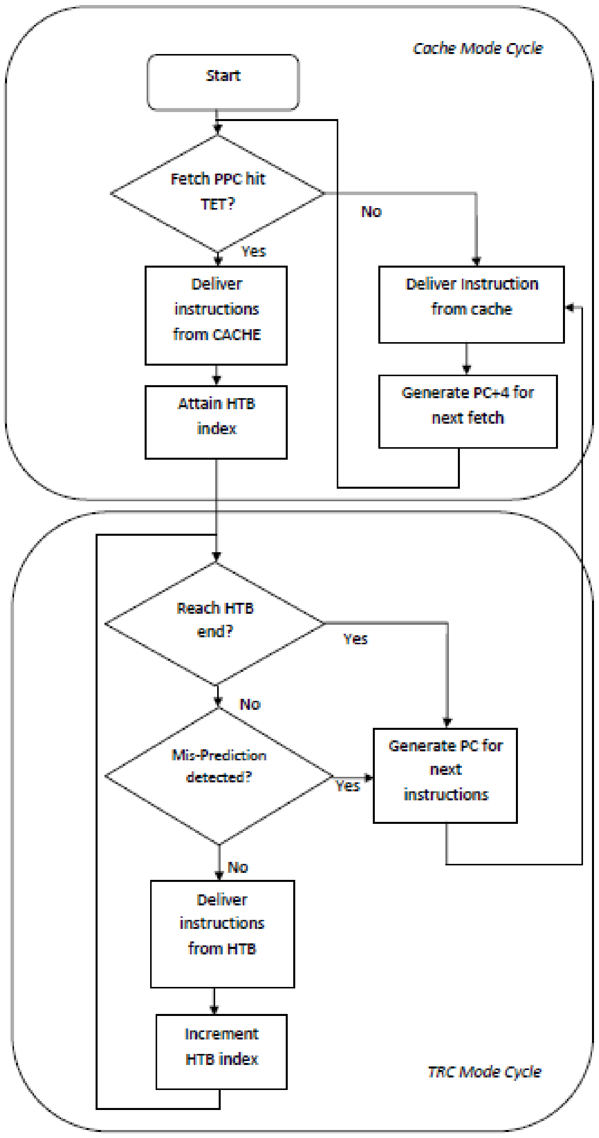
Cache Mode Cycle
Start
Fetch PPC hit TET?
No
Yes
Deliver instructions from CACHE
Deliver Instruction from cache
Attain HTB index
Generate PC+4 for next fetch
Reach HTB end?
Yes
No
Mis-Prediction detected?
Yes
Generate PC for next instructions
No
Deliver instructions from HTB
Increment HTB index
TRC Mode Cycle

Fig. 3. Flowchart for Instruction delivery

our modifications. Simulations were run for various scenarios and it was automated using shell scripting.

During the calculation of energy, it is assumed that the normal branch predictor and I-cache are clock gated when the low power branch predictor and the TR Cache return a hit.

## VII. Results

Figures 4-8 represent the results of our experiments.

Fig 4 represents the total power consumption of the processor. It is clear from the graph that using either TR Cache or Low Power Branch Predictor shows a significant reduction in power consumption. The maximum power reduction is seen when the Lowe Power Branch Predictor is combined with the TR Cache.

Fig 5 depicts the branch predictor power consumption. We can observe from the graph that the Low Power Branch

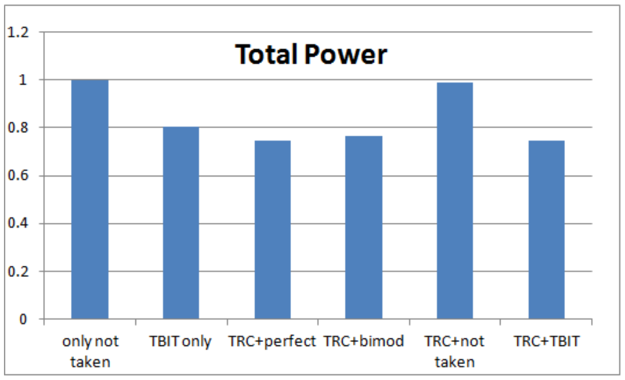

Fig. 4. Total Power Consumption normalized against base value (no TR cache or Low Power Branch Predictor, Only not taken branch predictor)

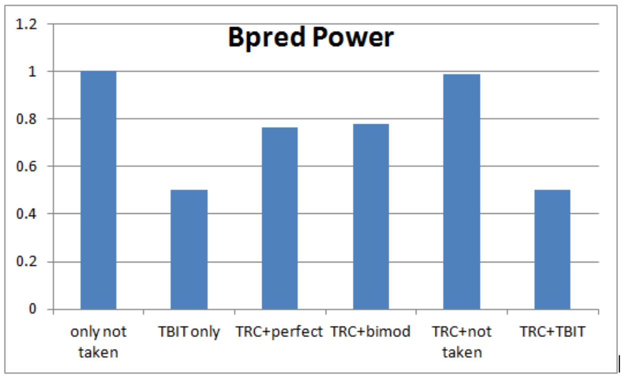

Fig. 5. Branch Predictor Power Consumption normalized against base value

Predictor works well in reducing power consumption. There is a reduction of 35% in branch predictor power consumption from TRC+bimod to TRC+TBIT.

Fig 6 represents the total instruction fetch power. It is a combination of the TR Cache power and the I-Cache power consumption. We can see from this graph that TR Cache performs well in reducing the power consumption. There is a reduction of about 18% in power consumption.

Fig 7 represents Cache miss rate for the TR Cache. As

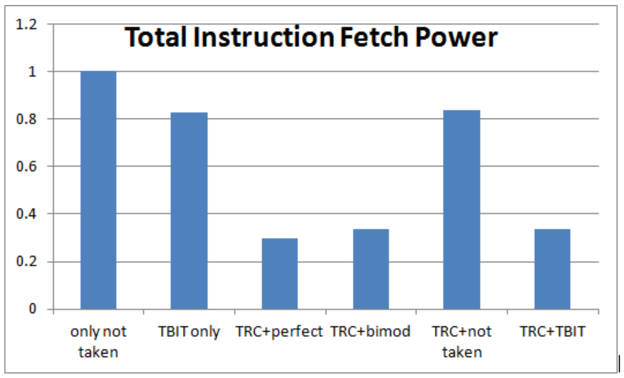

Fig. 6. Instruction Fetch Power Consumption (I-Cahce + TR Cache power) normalized against base value

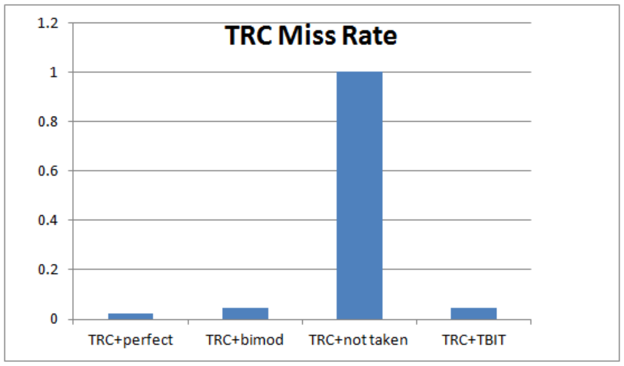

Fig. 7. TR Cache miss rate normalized against base value

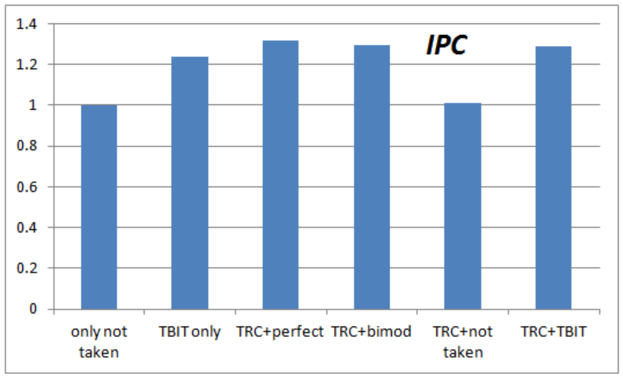

Fig. 8. IPC normalized against base value

expected, when the predictor works well, the miss rate is low.

Fig 8 represents the average IPC of the programs simulated.

## VIII. Analysis

Our first figure, Fig. 4, depicts the total power consumed by the benchmark program when executed using the various aforementioned combinations of algorithms. The values depicted in the figure are normalised against the base value. It is evident that the not taken branch predictor with no additional cache algorithm has the highest power consumption.Also, using the Trace Reuse cache reduces the power consumed by a relatively meagre amount. But the low power branch prediction algorithm reduces the total power consumption by 20% almost by itself. This makes it evident that the TRC+TBIT implementation reduces the power consumption by almost 25%, a larger amount, which is proven from the figure. Interestingly, TRC implementation with a perfect branch also provides an approximately same level of power savings compared to TRC+TBIT.

Fig 5, depicts the branch predictor power alone. the figure illustrates that the cache access technique employed also considerably affects the branch predictor power. As stated theoretically, the low power branch predictor produces the lowest power consumption both by itself and along with the TRC cache implementation, providing almost 50% power savings.

Fig 6, depicts the cache power for different cache access techniques employed. Here again as stated theoretically, the TRC methods provided considerable power savings when paired with the TBIT branch predictor method. But another fact to be noted is that, the TRC technique is seen to do well with the perfect branch prediction technique also, which is evident from all the comparison figures plotted.

Fig 7, depicts the cache miss rate for different cache algorithm implementations. We can clearly see that using an advanced branch predictor improves the TR Cache miss rate by a large percentage compared to using only not taken branch predictor. This is contributes to improving upon the original research, in which only the not taken branch predictor was used [1].

Fig 8 depicts the Instructions Per Cycle count for all the aforementioned algorithm combinations.Higher the IPC count, higher is the program efficiency. Similar to all other plots, the not taken branch predictor produces the worst result. The TRC method with both the Perfect and TBIT branch predictor provides the highest IPCs hence proving that our proposed algorithm outweighs all other combinations in almost all the parameters considered here.

The overall analysis of these results clearly conveys the fact that the TRC method along with the TBIT implementation is the best choice for embedded applications to bring about a drastic reduction in power consumption. Interestingly, the TRC method with the Perfect branch predictor also produces almost similar results which can be further researched in the future for a few more improvements in power consumption.

## IX. Conclusion and Future Work

Our method focuses on utilizing a low power branch predictor and a trace reuse cache for reducing the overall power consumption in the system. The low power branch predictor utilizes a TBIT table to reduce the BTB lookups for branches. The trace reuse cache serves to reuse the retired instructions from the pipeline backend of the processor using a History Trace Buffer(HTB) and a Trace Entry Table(TET). These two methods are power efficient methods by themselves and hence when integrated together guarantee a considerable improvement in power efficiency.

Energy saving is computed by calculating the power consumption in each of the aforementioned methods and comparing. Power is calculated by modifying the power.c code in the sim-watch of simplescalar for each of our combination of techniques. Also, the total cache hit and miss rates and the branch predictor hit and miss rates are monitored to determine an efficient combination among the above mentioned techniques.

There is a lot of scope for future work. We performed our simulations for a certain size of the TBIT, HTB and TET. We did leave the size variable in our modifications of simplescalar so that in future we can optimize their values for reducing power consumption or increasing performance. The sizes of the various structures can be optimized as per the application requirement.

Our experiments were performed for low power systems that assumed a simple single-issue architecture. We can incorporate our low power design to superscalar architectures to improve their consumption for a modest reduction in performance. The power profiles can be studied for superscalar architectures and optimizations can be performed.

## References

[1] Y. Tsai and C. Chen, *"Energy-Efficient Trace Reuse Cache for Embedded Processors"* IEEE Trans. on VLSI Systems, Sept. 2011.

[2] S. Kim, E. Jo and H. Kim, *"Low Power Branch Predictor for Embedded Processors"*. 2010 10th IEEE Int. Conf. on Comp. and Info. Tech. (CIT 2010).

[3] J. Montanaro, R. T. Witek, K. Anne, A. J. Black, E. M. Cooper, D. W.Dobberpuhl, P. M. Donahue, J. Eno, G. W. Hoeppner, D. Kruckemyer, T. H. Lee, C. M. Lin, L. Madden, D. Murray, M. H. Pearce, S. Santhanam, K. J. Snyder, R. Stephany, and S. C. Thierauf, *"A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor"* IEEE J. Solid-State Circuits, vol. 31, no. 11, pp. 17031714, Nov. 1996.

[4] J. Kin, M. Gupta, and W. H. Magione-Simth, *"Filter cache: An energy efficient memory structure"* in Proc. 30th Int. Symp. Microarch., Dec. 1997, pp. 184193.

[5] T. Austin, E. Larson, and D. Ernst, *"SimpleScalar, An infrastructure for computer system modeling"* IEEE Trans. Comput., vol. 35, no. 1, pp. 5967, Feb. 2002.

[6] HP Labs, Palo Alto, CA, *"CACTI: An integrated cache and memory access time, cycle time, area, leakage, and dynamic power model,"* 2009. [Online]. Available: http://www.hpl.hp.com/research/cacti/

[7] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, *"MiBench: A free, commercially representative embedded benchmark suite,"* in Proc. IEEE 4th Annu. Workshop Workload Characterization, Dec. 2001, pp. 314.

[8] David Brooks, Vivek Tiwari, Margaret Martonosi, *"WATTCH: A Framework for Architectural-Level Power Analysis and Optimizations,"* In Proceeding of ISCA 2000, ACM Press, 2000