Analysis on Low Power Cache Design

CSE 661 Term Paper

Lichen Liang

Abstract

Over the last few decades, engineers have been majorly focusing on the improvement of performance in computer systems. This has been very successful as the size of transistors has decreased to 7nm so far, the number of transistors on a die is increasing just like the Moore's law has estimated, and the speed of execution are improving from generation to generation. Along with the decrease in size of transistors, also comes with the decrease in cost of manufacture. However, the energy and power required for computer systems have also been increasing so does the cost. This may not be significant on personal computers as a person may be paying a little extra money every month because of a high performance computer he/she uses. However, in industrial level, such as warehouse scaled computers (WSC) or cloud servers, a little change in the design of a computer system can hugely improve the energy efficiency of a system. In other words, it can save a company millions of dollars every month on operational expenses. Cache, as a part of a computer system, is used at all times. A cache's power consumption simply cannot be ignored even it is a very little part of the whole computer system. This paper focuses on the analysis of imposed improvements on cache design in order to reduce power consumption.

I. Introduction

Caches have played an important role in the history of computers. Its design significantly affects the performance of a computer. Today, caches are not only used in computers but also other digital devices such as cameras, printers, smart watches, and mobile phones. There has been a lot of studies on cache such as size, hierarchy, set associativity, data and instruction cache, etc. In other words, there are many combinations we can use to design a cache. At the same time, we also have to take into consideration of what each cache is used for and what are its advantages. So far, there are some common sense on caches. For example, caches are used to store data or instructions such that it prevents the extra time spent on accessing to the main memory or disk. This improves the speed factor and instructions can be completed in a much shorter time. If there has been a cache miss, then there is the need to access the memory, in addition to the penalty of the cache miss. L1 caches are designed to be small and to be direct mapped. A smaller cache usually has a lower latency and a direct mapped cache reduces the hit time. However, this increases the miss rate of a cache. On the other hand, L2 caches are designed to be larger than L1 and to be set associative. Set associativity decreases the miss rate, but a larger cache results in a higher hit time.

Even though, it is still better than accessing the main memory as the access time is huge compared to caches.

Since caches are used constantly, its power consumption in the long run is significant factor. In this paper, there has been a few proposals or designs that focused on reducing the cache's power consumption. [1] focused on the optimization between performance and energy. It also included different components on a cache and how each component plays a part in optimizing energy consumption. Papers [2], [3], [4], and [5] used a similar approach but different methods. That is, using a prediction method to limit the part of the cache accessed just like using a buffer for a cache access. In [4], instead of predicting the one that is likely to be used, it predicts the ones that are most likely not going to be used. The concept is the same here, but with different logic. In [6], the design is taken to the cell level, where they introduced a 7 Transistor SRAM cell rather than conventional 6T SRAM cell. In [7], they introduced a Dynamic Zero-Sensitivity (DZS) scheme to reduce cache's power consumption by preventing bit lines from discharging when reading a 0.

All the papers that are referenced have achieved an energy reduction of 40-50%. This shows that it all comes down to pick the better design if the energy reduction is similar. This paper consists of five sections. Section 2 provides the basic information and types of cache, such as components and partitioning. Section 3 provides the analysis on each paper and related experimental results. Section 4 concludes the paper and mention any future improvements that can be done. Section 5 provides the references and Section 6 provides the appendix which include summary power point slides that can be used to introduce this topic to others.

II. Cache Component, Model and Designs.

Caches are made up of three different components: cell array, I/O path, and address decoding path [1]. Each component has sub-components that use the energy. Therefore, any sub-component that is dominant than other sub-components in energy consumption is dominant in the energy consumption of the whole cache.

The cell array consists of data array, tag array, and read or write circuitry. Tag and data arrays are dominant terms. Moreover, this can be further implemented in two ways: dynamic logic and static logic. Dynamic logic pre-charges the bit lines and the static does not pre-charge. Dynamic logic also consumes more energy than static. The tag and data arrays' energy consumption depend on the number of bit switches in the bit lines in the I/O path. The I/O path consists of the buses and I/O pads with I/O pads being dominant. The energy consumption in I/O path depends on the number of bit switches in the I/O pad, which also affect the cell arrays. The greater number of switches, the higher energy consumptions. Address decoding path consists of address decoding logic and address bus. The latter dominates. Finally, the total energy is calculated by the sum of energy consumption of the three components.

There are two ways to partition a cache: vertically and horizontally. Vertical partitioning cache focus on creating several levels of cache i.e. L1, L2, L3 and therefore decreasing the size of each cache. The smaller the size, the less power consumption because it has a lower load capacitance. This allows the introduction of block buffers, which papers [2], [3], and [4] had similar implementations.

A block buffer is similar to a cache, but for the cache. It is closer to the processor. Therefore, the processor checks the block buffer first before checking L1 cache. This way the energy consumption by the actual caches are reduced because block buffers take advantage by reducing load capacitance. However, the size of a block buffer also matters. If it is too small, then it contains limited number of blocks that can be hit, and not effective. On the other hand, if it is too big, then it may have too much unused blocks which consumes energy. Block buffers performs the best when there is a spatial locality like a loop or sequence of instructions.

The horizontal partitioning of a cache divides cache into banks called sub-bank. Accessing a sub-bank that contains the data has an advantage over accessing the whole cache. This is similar to accessing a single set/way of cache. This way, there is a reduction in energy consumption by not accessing other sub-banks. This method is also used in [3], [4], and [5].

III. Analysis and Results Discussion

[1] used four benchmark and twelve cache models: direct mapped, 2-way, 4-way, each with 2, 4, and 8 words. It tested caches in both data and instruction separately in terms: hit rate with size and associativity difference, access time and latency with different size and associativity, energy consumption in both data/instruction of dynamic/static logic, and cache partitioning. The result as follows.

Starting from a general point that does not need much explanation. The hit rate increases with the increase in size and associativity. Therefore, a set associative cache has a higher hit rate than direct mapped cache with the same cache size. The access time of a direct mapped is lower than access time in associative cache. This is a very important point as in [3], [4], and [5] uses this as an advantage to change a set associative cache into a direct mapped by prediction. In an instruction cache, direct mapped has a shorter latency than set associative. However, in a data cache, it is the opposite. A set associative cache has a shorter latency than direct mapped.

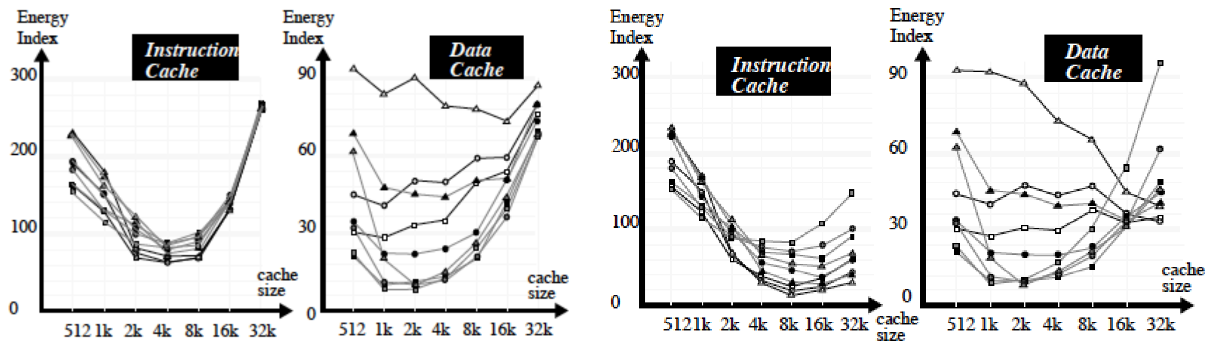


Figure 1. (left) Energy consumption in dynamic logic, (right) Energy consumption in static logic [1]

In figure 1, the four graphs show the energy consumption in cache with two different logics. The squares and rectangles on the line represent it is direct mapped, all others are set associative. From figure 1 can conclude that in both instruction and data cache, dynamic logic direct mapped caches uses less energy than set associative. The optimal cache size is around 4kb. There are some cases where a set associative is less than the direct mapped. This may be because of the size since smaller cache uses less energy. On the other hand, both instruction and data caches with static logic, set associative caches use less energy than direct mapped. Many computers now use dynamic logic and set associativity. Logic cannot be changed but set associativity can be changed to direct mapped. This point will be further elaborated in [2].

Paper [2] introduced a design by using a location cache. The location cache is similar to a block buffer in [1] but with minor difference. It is accessed as a direct mapped cache, but it is indexed virtually and processed along with TLB translation. When there is a miss in L1 cache, this missed information will be added to location cache, and is passed L2 cache so that L2 cache can be accessed like a direct mapped cache. If there is still a miss, then L2 cache is accessed in its original way. With the advantage of execution in parallel with TLB translation, even if it misses, it will not cause any delay or penalty. Instead, it holds the chance of saving energy and latency. This is a very smart and easy implementation especially because it does not add any latency if it does not work at all, i.e. no penalty. However, if L1's hit rate is extremely high, then the location cache is idling and consuming power.

In the test, they used 16Kb of both data and instruction cache. L1 cache is 4-way set associative with 64 byte cache line size. Set associative cache in L1 may cause the location cache to be in idle mode as mentioned earlier. L2 cache is 8-way 512Kb with 128 byte cache line size. They tested this in 26 benchmarks. The overall energy consumption of the L2 has decreased by 47% and overall latency decreased by 25%. The downside of this implementation is that it ignores the structure and execution of L1 cache completely. Therefore, it is best to be used with other implementations.

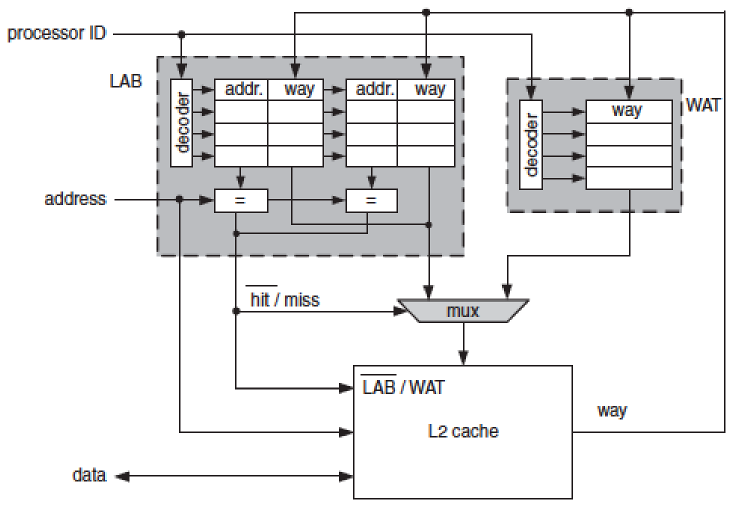


Figure 2. Design of WP-L2 [3]

In [3], they used look ahead buffer (LAB) and way affinity table (WAT) to construct the way predicting L2 cache (WP-L2) as shown in figure 2. This cache works best with sequential instructions. When there is a access of L2 that resulted in a hit, this information is passed to look ahead buffer. The look ahead buffer then looks for next instruction from the sequence and the way affinity table looks for the way of the next instruction.

They ran the WP-L2 and MRU heuristic to compare the difference. It turns out that WP-L2 performed much better that MRU heuristic. WP-L2 reduced 22% of the latency and 44% of the energy consumption. However, the look ahead buffer and way prediction table have penalty. Moreover, it is still only applied to L2 cache.

In [4], they introduce two filters for L1 and L2. The filter for L1 is the block buffer mentioned in [1]. However, block buffers highly depend on spatial locality. For L2 cache, they used a term called *sentry bit.* Since L2 cache are set associative, the sentry bit is used to filter out ways that are most likely not going to be accessed. The sentry bit is a part of the tag bit that is stored in the sentry-tag storage. Usually the lower tag bits are used to be a sentry bit. For example, if a 4 way cache has tag bits ending in 0, 0, 0, and 1, and the sentry bit of the current process is also 1, then the first three ways can be disabled. A similar scenario is demonstrated in figure 3.

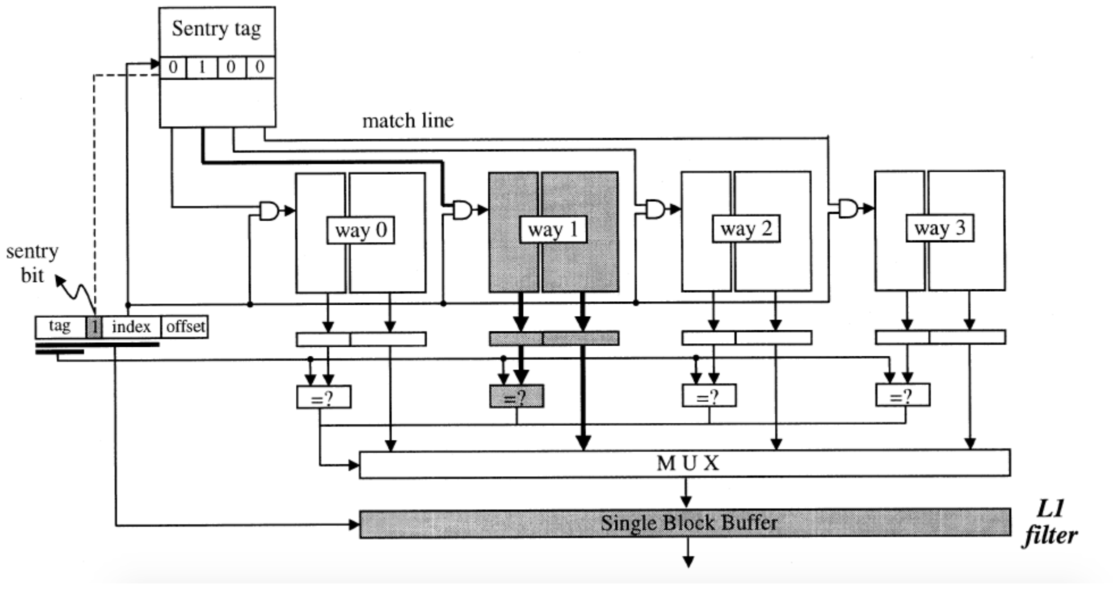


Figure 3. Two level filter with 4 way set associativity. [4]

To test the design, they fed address into L1 and L2 to check whether the address is in the block buffer and the sentry bit exist or not, respectively. Both processes are done concurrently. If it hits the L1 cache, no access to L2 is needed. If it misses from L1, and miss the sentry bit in L2, a regular L2 access is required. If the sentry bit hits in L2, then compare the rest of the tag. There exists a possibility that the tags do not match, which will result in a penalty. Therefore, the number of sentry bits used is very important as it narrows the search. For example, if the sentry bit is only one bit, there may be a lot of hits in the sentry bit storage. Then we have to access many cache ways and compare the tags. If the sentry bit is eight bits, then it may just match to one in the sentry bit storage and all other cache ways' access will be disabled. They tested two different caches a 2 way associativity and a 4 way associativity. The energy consumption reduction was 30% and 46% respectively.

In [5], they implemented an access mode prediction. That is, predict whether it is a cache hit or miss. If cache hit, use way prediction to find and use that cache way. If cache miss, use phase to access all tags, compare, and use the matched cache way. Figure 4 shows a flow chart of this method.

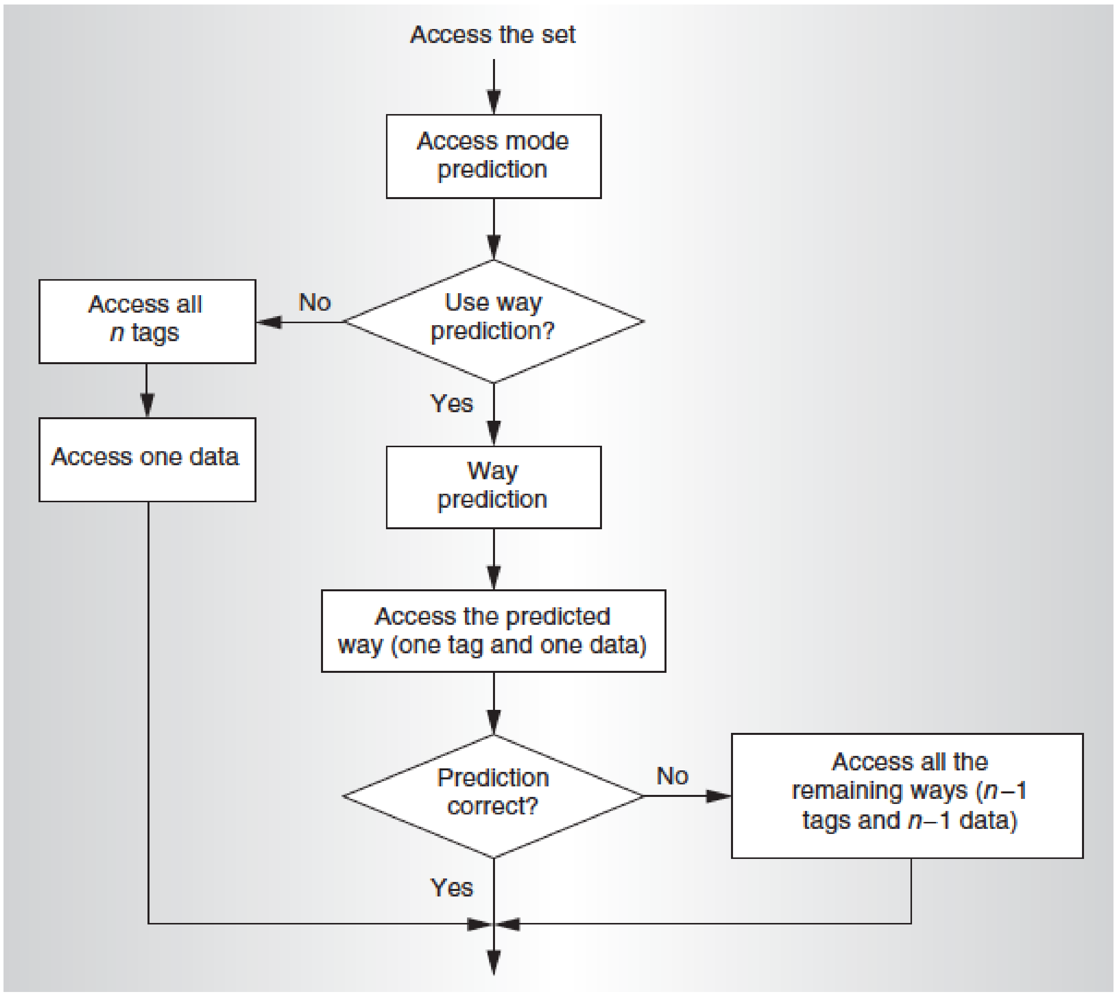


Figure 4. Flow chart for [5]

In order to predict way, they first need to predict a cache hit or miss. A global share predictor is implemented. This predictor, unlike others, does not need the next cache's reference address to predict. Therefore, it can return a prediction immediately just based on access history. The down side of this implementation is that if the global share predictor is wrong, then the way predictor uses more energy. They tested this method with 26 benchmarks, and the energy consumption reduction is averaged at 8.6% with highest reduction on 46.2%

[6] used a different approach. It introduced a brand new cell design of 7 transistor SRAM instead of conventional 6 transistors. The diagram of the design is shown in figure 5.

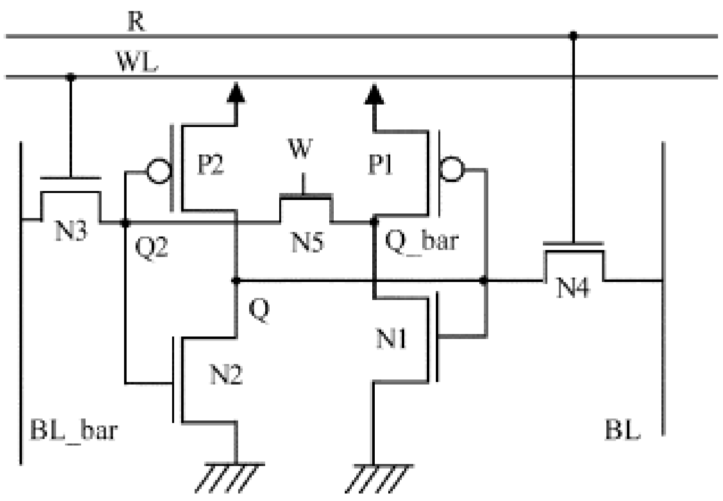


Figure 5. 7T SRAM design [6]

Traditional 6T SRAM cells has two back to back inverters for reading and writing. These two inverters are constantly transmitting feedback from one to another for write operations. 7T SRAM cuts off this feedback channel. Instead, it uses N5 and BL_bar as shown in figure 5 to perform a write operation. The idea of this implementation is to reduce the number of bit switches. However, this design will increase the area used on a die by 12.25% of a 6T SRAM. The energy reduction of this design is 49%.

In [7], the approach is very interesting. For a SRAM with two bit line design, one of the bit lines must discharge in order to read a 0 or 1. The same amount of energy is used in both reading. However, the bit distribution of 0 is much greater than 1. Therefore, the idea is let only one bit line discharge when the stored value is 1, or prevent both bit line from discharging when the value is 0. This is called Dynamic Zero Sensitivity (DZS) implementation. This way, the amount of bit line switches is reduced, so is the energy. The changes are also made in the cell level by adding two extra transistors. The overall energy reduction is 44.2% for instruction cache and 53.6% for data cache.

IV. Conclusion and Future Improvement

This paper has analyzed some of the methods proposed to reduce the power consumption of cache usage. Many papers have used a very similar idea but with different implementation. From that, we can conclude the following:

- Caches are composed of three components, and some of their sub components dominates the energy consumption.
- Dynamic logic pre-charges the bit lines, whereas static logic do not.
- Caches can be vertically partitioned into hierarchy and implement a block buffer. It can also be horizontally partitioned into sub-banks where each sub-bank is accessed individually like a direct mapped.
- A larger blocked and set associative cache can increase the hit rate.
- The access time of a direct mapped cache is lower than set associative cache.
- In an instruction cache, direct mapped has a shorter latency than set associative.
- In an data cache, set associative mapped has a shorter latency than direct mapped.
- Using dynamic logic, direct mapped cache uses less energy in both instruction and data cache.
- Using static logic, set associative cache uses less energy in both instruction and data cache.
- Cache size from 4K to 16K are the most energy efficient.
- The location cache of [2] saves the miss from L1 and pass it to L2 so that L2 can be accessed as direct mapped.
- The look ahead buffer and way affinity table in [3] works best with sequential data. However, it has penalty and the design is more complex but limited to sequential data only. It may not seem to be a good trade off.
- The sentry bit in [4] predicts the L2 ways that are unlikely to be accessed to . It also uses a block buffer for L1.
- The access mode prediction in [5] also uses way prediction to access L2 in a direct mapped fashion. However, it also need a prediction for whether it is cache hit or miss. This makes access mode prediction dependent on other methods.
- 7T SRAM cell in [6] reduces the energy consumption, but increases the area usage on a die.
- The dynamic zero sensitivity also uses a new design that keeps bit lines from discharging when reading a 0, which is the majority of reads.

Many of the implementations focuses on accessing the L2 cache in a precise way. This prevents the access of useless data and reduce energy usage. Since the energy reduction rate between the methods are in the approximate range of 40%-50%, then the simplest implementation is preferred. That is the implementation in [4] with two filters. There can be a slight change to this implementation. Instead of predicting the way that is likely not going to be used, make it predict the way that is likely to be used. Therefore, by using the block buffer from [4] and the location cache from [2] will create a great combination. This combination also does not have much penalty or latency increase.

With future improvements, there can be more cases to be studied and analyzed. However, keeping the design simple is very important because the goal is to decrease energy consumption and not affecting any other factors negatively. There can also be more tables and figures in the paper to demonstrate a detailed and precise comparison.

V. References

[1]Su, C.-L., & Despain, A. M. (1995). Cache design trade-offs for power and performance optimization. *Proceedings of the 1995 International Symposium on Low Power Design - ISLPED 95*. doi: 10.1145/224081.224093

[2]Min, R., Jone, W.-B., & Hu, Y. (2004). Location cache. *Proceedings of the 2004 International Symposium on Low Power Electronics and Design - ISLPED 04*. doi: 10.1145/1013235.1013271

[3]Chung, C.-M., & Kim, J. (2010). Low-power L2 cache design for multi-core processors. *Electronics Letters*, *46*(9), 618. doi: 10.1049/el.2010.0642

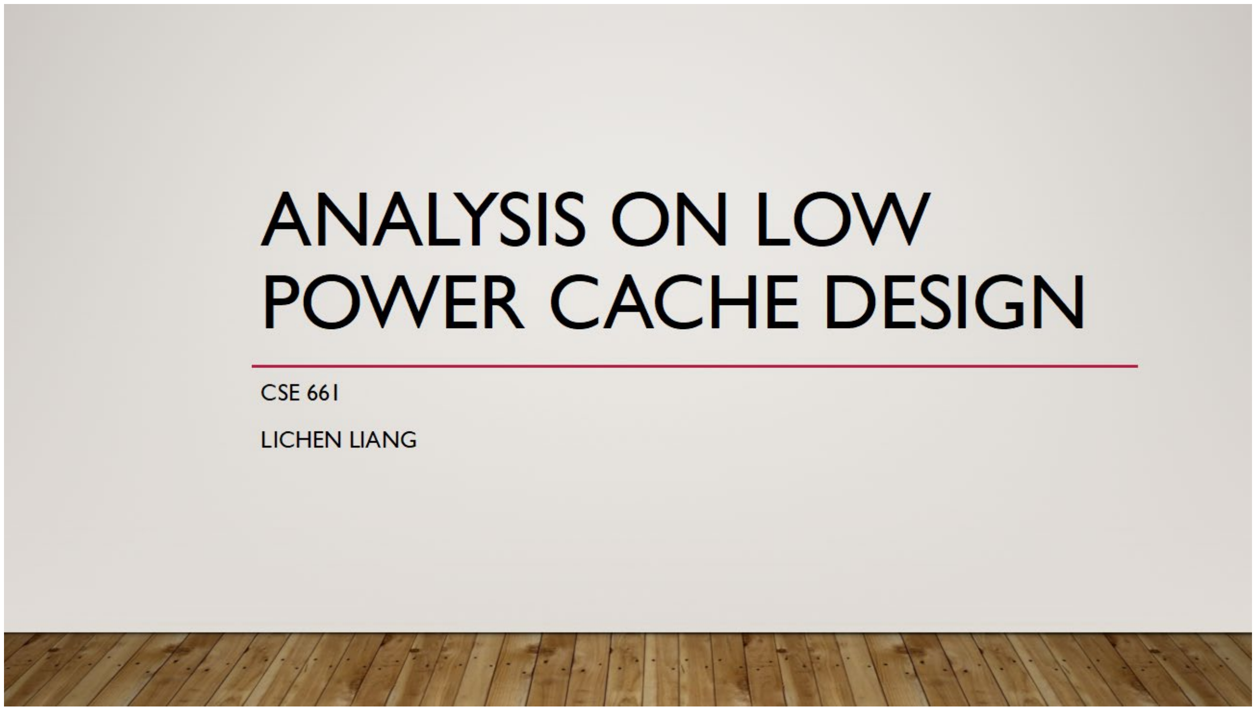
[4]Chang, Y.-J., Ruan, S.-J., & Lai, F. (2003). Design and analysis of low-power cache using two-level filter scheme. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *11*(4), 568–580. doi: 10.1109/tvlsi.2003.812292

[5]Zhu, Z., & Zhang, X. (2002). Access-mode predictions for low-power cache design. *IEEE Micro*, *22*(2), 58–71. doi: 10.1109/mm.2002.997880

[6]Aly, R. E., & Bayoumi, M. A. (2007). Low-Power Cache Design Using 7T SRAM Cell. *IEEE Transactions on Circuits and Systems II: Express Briefs*, *54*(4), 318–322. doi: 10.1109/tcsii.2006.877276

[7]Chang, Y.-J., & Lai, F. (2005). Dynamic Zero-Sensitivity Scheme for Low-Power Cache Memories. *IEEE Micro*, *25*(4), 20–32. doi: 10.1109/mm.2005.64

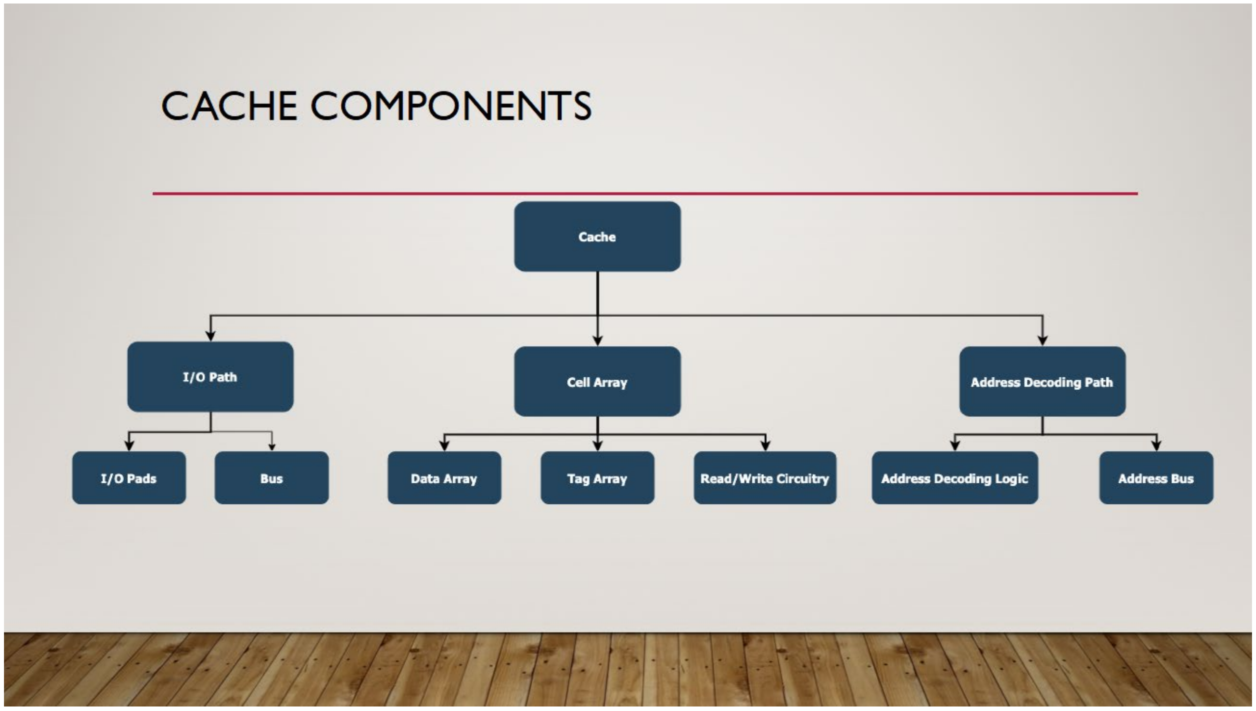
VI. Appendix



ANALYSIS ON LOW POWER CACHE DESIGN

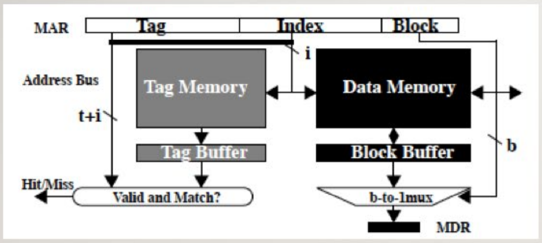
CSE 661

LICHEN LIANG

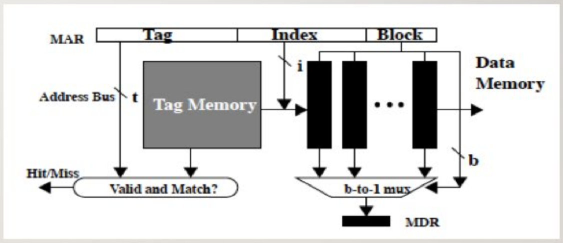


CACHE COMPONENTS

CACHE PARTITIONING

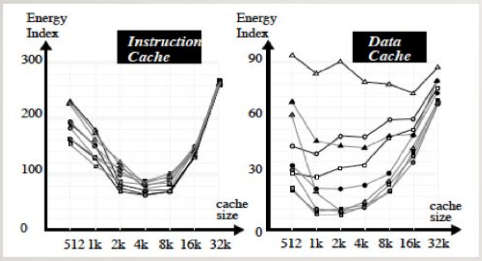


Vertical Cache w/ Block Buffer

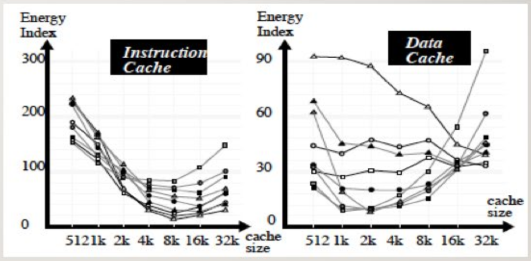


Horizontal Cache w/ Sub-Banks

ENERGY CONSUMPTION



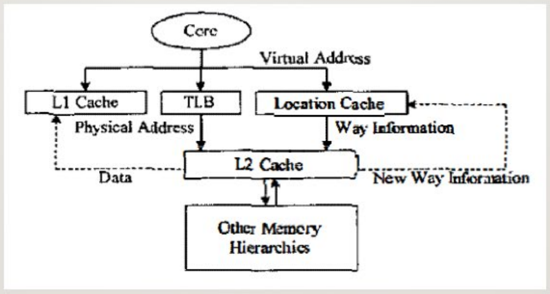
Energy Consumption in Dynamic Logic



Energy Consumption in Static Logic

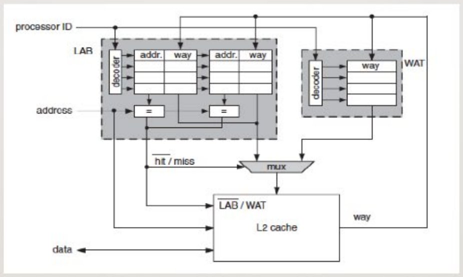
The squares and rectangles in the diagram represent direct mapped cache, and all others represent set associative cache

LOCATION CACHE



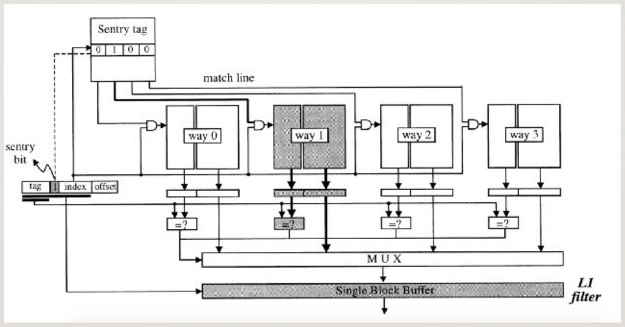
The location cache works concurrently with L1 and TLB.
When there is a L1 miss, the location cache allows L2 cache to be access like a direct mapped cache

LOOK AHEAD BUFFER AND WAY AFFINITY TABLE



The look ahead buffer then looks for next instruction from the sequence and the way affinity table looks for the way of the next instruction.

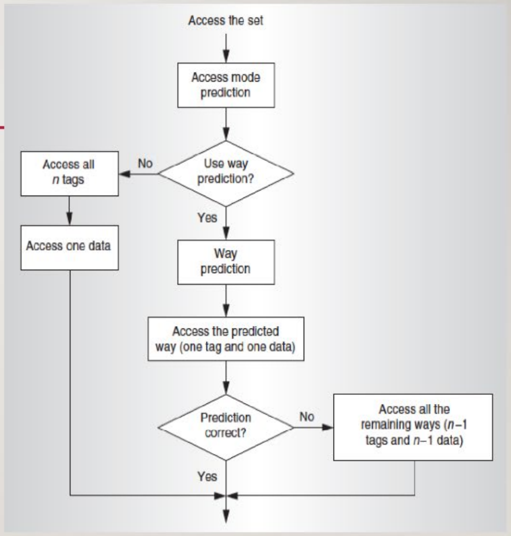
TWO LEVEL FILTER



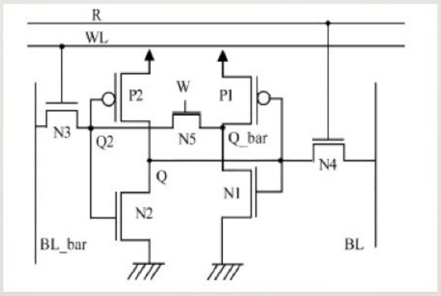
One block buffer and one sentry bit storage.
The sentry bit is part of the tag, which is compared to filter out ways that are likely not going to be accessed.

ACCESS MODE PREDICTION

First, predict whether a cache is a hit or miss, then use access mode prediction to predict which way of L2 is going to be accessed.

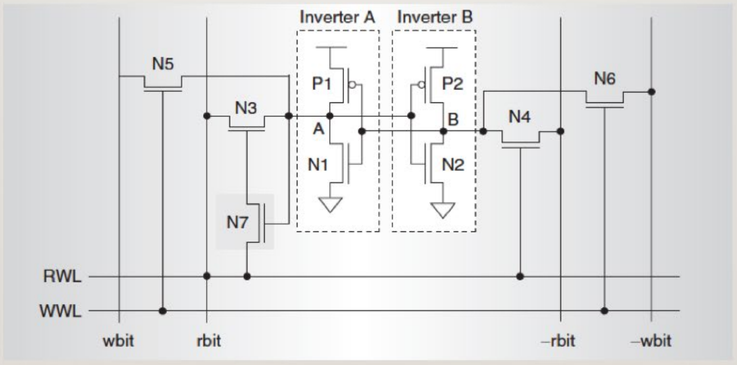


7T SRAM DESIGN

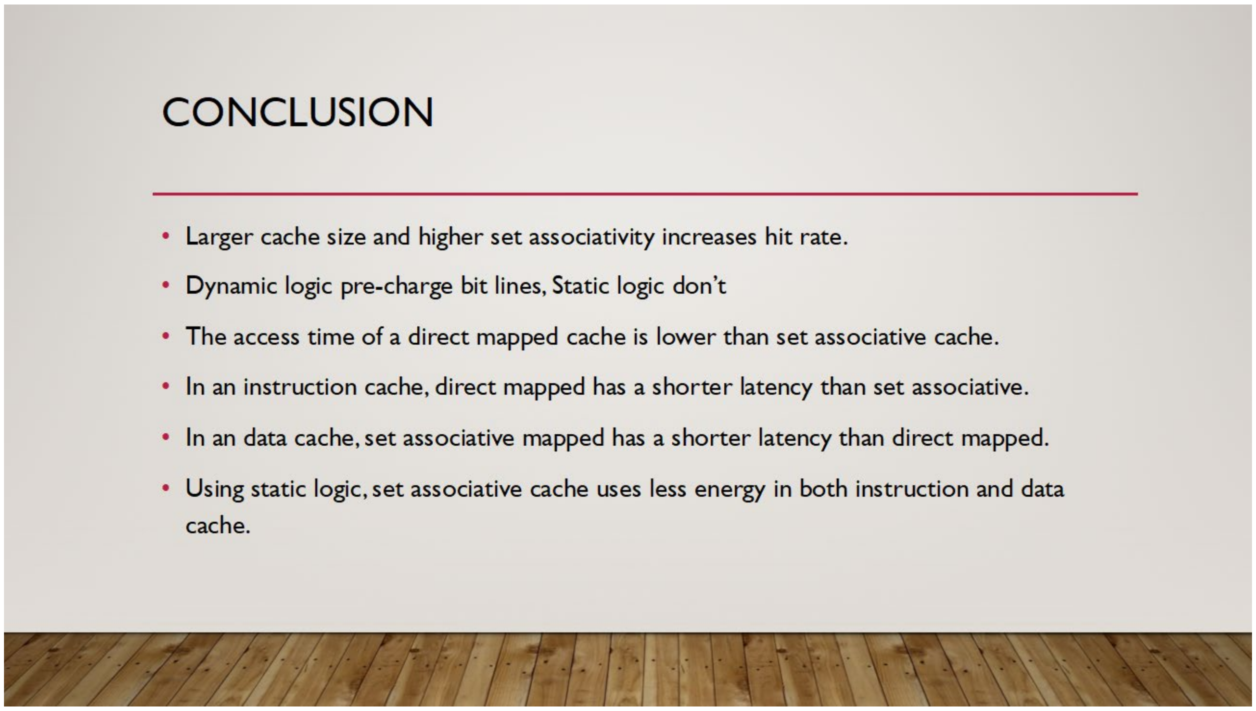


The idea of 7 transistor SRAM is to reduce the number of bit switches in the SRAM

DYNAMIC ZERO SENSITIVITY

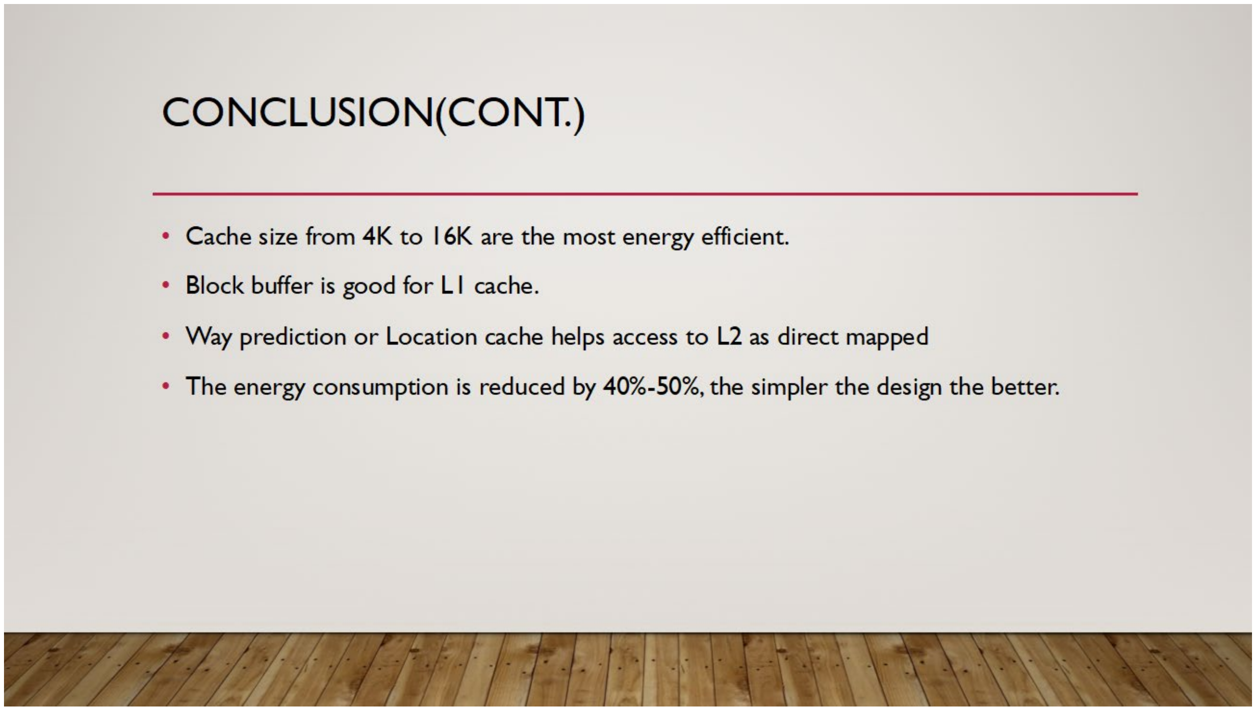


Prevent both bit lines from discharging when reading a 0, which 0 is much more frequently read than 1.



CONCLUSION

- Larger cache size and higher set associativity increases hit rate.
- Dynamic logic pre-charge bit lines, Static logic don't
- The access time of a direct mapped cache is lower than set associative cache.
- In an instruction cache, direct mapped has a shorter latency than set associative.
- In an data cache, set associative mapped has a shorter latency than direct mapped.
- Using static logic, set associative cache uses less energy in both instruction and data cache.



CONCLUSION(CONT.)

- Cache size from 4K to 16K are the most energy efficient.
- Block buffer is good for L1 cache.
- Way prediction or Location cache helps access to L2 as direct mapped
- The energy consumption is reduced by 40%-50%, the simpler the design the better.