NTNU

TFE4520: Digital System Design, Specialization Project

Project Report

Evaluation of Cache Architectures for a Low-Power Microcontroller System

Author:

Vinicius Almeida Carlos

January 28, 2013

Abstract

Energy consumption is one of the main concerns when designing battery powered embedded systems. On microcontroller systems, the memory systems is responsible for a great part of the total energy consumption. Flash memories, as the ones usually employed in this kind of system are very power hungry elements. Therefore this project aimed at investigating cache architectures that could be applied to a specific microcontroller system composed of a microcontroller and a Flash memory in order to improve their energy efficiency. Later on, simulations on the target environment were performed to determine the best solution for the given system. The results of these experiments are then presented at the end of the text, where a critical analysis of these findings is also provided.

Contents

Chapter 1

Introduction

It is a well-known fact that energy consumption is of great concern on battery powered embedded systems. Even though improvements on battery capacity have been made in the past years, they are unable to keep up with the increase of energy consumption of such systems. In general terms, many techniques can be devised to tackle this problem, as described for instance in [23]. Nevertheless, the focus here is on microcontroller based systems, in which memory systems are responsible for a great part of the system total energy consumption. In particular, Flash memories, as the one used in the target system of this project, are power hungry elements.

The baseline architecture of the target system of this project, displaying the connection between the microprocessor and the Flash memory, can be seen on figure 1.1.

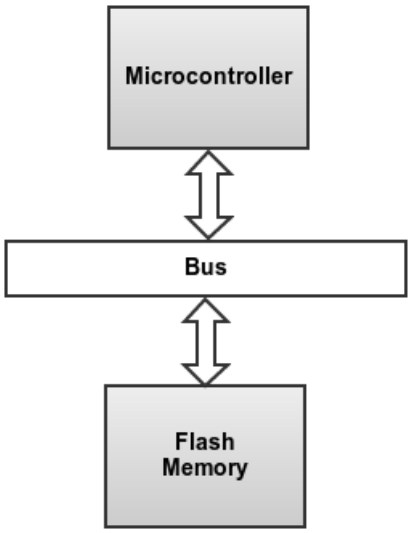

Figure 1.1: Baseline Architecture.

The aim of this project is to extend this architecture by adding a small Static Random-Access Memory (SRAM) to the system, in order to avoid, as much as possible, accesses to the main memory (the Flash memory). The extended version is depicted on figure 1.2.

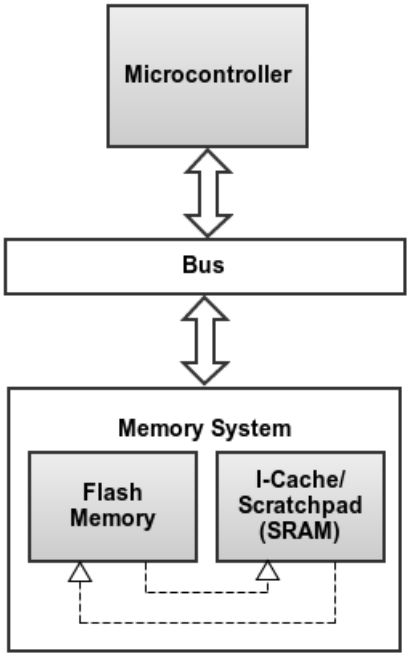

Figure 1.2: Extended Architecture.

As can be seen in the figure 1.2, the SRAM added to the system will be used to store instructions only. This small SRAM can then either be an instruction cache (I-cache) or a scratchpad memory (SPM). Either way, the purpose remains the same. More importantly, they still represent a great part of the total power consumption of the system, therefore careful design of these elements is paramount to reduce overall energy consumption.

Therefore, it is the goal of this project to assess memory system architectures, in order to find the best suitable options for the target embedded system.

The specific tasks pertaining to this project are listed in the following section.

1.1 Tasks

Task 1: Realize a broad literature research on the field of cache architectures for energy consumption reduction.

The intention behind this task is to obtain a wide overview of the possibilities available in terms of cache memories when it comes to reducing energy consumption, independent on whether they are suitable or not for the simulations to be performed in the scope of this project. The solutions that cannot be directly applied to this project can serve as ideas for further exploration in future projects.

Task 2: Select two or more memory system architectures to be later evaluated through simulation.

This task comprises not only critically choosing which solutions should be evaluated, but also implementing any required algorithm or tool for the given architectures before the simulation phase starts.

Task 3: Perform high-level simulations with the selected architectures and an actual application.

In this task, the selected architectures must be put to test, enabling analysis to be performed on the obtained results.

Chapter 2

Cache Architectures for Low Power

(This text assumes the reader is familiar with basics on cache. For an excellent introduction, please read Chapter 7 of [20]).

Caches are usually employed to hide the ever existing latency between the processor and the main memory. However, due to cost constraints, there is a limit to the size of these memories.

Caches are on the critical path of systems, hence contributing to much of a microprocessor system's power and energy consumption. For example, in [32], it is reported that caches may consume approximately 50% of a microprocessor's power. And, as shown in [11], instruction caches consume more energy than data caches, therefore this literature research focused on solutions that were more relevant to instruction caches.

Although the main interest of this project is on architectures for caches that can reduce energy consumption, circuit level techniques were also taken into account, as a possibility for a future work, and thus are also briefly presented in the next section.

2.1 Circuit Techniques

2.1.1 Way-Decay Cache

The idea, as presented in [14], is that after some period of time the cache will start to suffer more misses (spacial and temporal locality decrease overtime), that is when there is opportunity to shut down some of the cache ways to reduce energy consumption. Thus, additional structures (basically counters) are added to the cache architecture to monitor the hits and misses and make a decision on when it is time to shut down some ways or turn them on again. The shutting down mechanism is performed with a circuit technique called gated-GND. An extra transistor is added to the ground path or supply voltage of the SRAM

cell, as shown in 2.1, taken from [14]. The rationale is that, by turning off parts of the circuit, static power can be saved (no leakage). For a 8-way-64K bytes instruction cache, under the SPECint95 benchmark, an average of 7.39% of energy saving is reported.

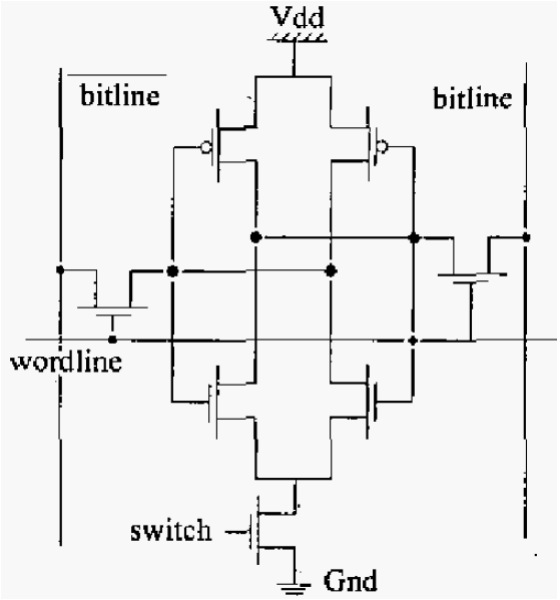

Figure 2.1: Gate-GND Technique

2.1.2 Drowsy Caches

This technique also focus on reducing current leakage, hence reducing static power consumption. As described in [10][3][9], the idea is to implement Dynamic Voltage Scaling (DVS) by changing the basic SRAM cell by introducing 2 transistors to control the voltage (as can be seen in 2.2, taken from [10]), enabling the memory to go to a state-preserving, low power drowsy mode. However, this solution requires that a prediction policy is implemented in order to determine when to put the cache lines in drowsy mode. Furthermore, there is a penalty of waking-up those lines when needed, increasing (slightly) the time to access the data. For a 2-way-32K bytes instruction cache, under SPEC2000, an average of 25% leakage reduction is reported.

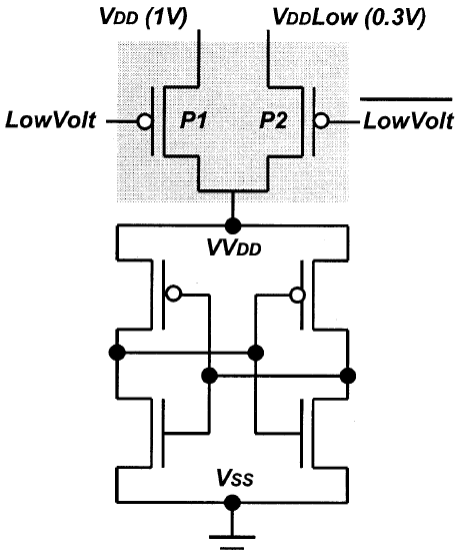

Figure 2.2: Drowsy Memory Cell

2.2 Architectures

2.2.1 Tiny Caches / Loop Caches

The starting point of this approach is the fact that the power per access of a memory begins to increase steeply at a size around 128 or 256 instructions ([6]). Therefore it is interesting to keep the size of the memory to a minimum.

A loop cache is a tagless cache, which stores the instructions of different loops and uses a controller to determine when the instructions should be fetched from the loop cache or from the next level on the memory hierarchy. The four variations available in the literature are presented in the following sections.

2.2.1.1 Dynamic Loop Cache (DLC)

In this type of loop cache, the loop detection is done dynamically based on the behavior of the program counter (PC). In the original proposal, as described in [6], the loop cache controller (LCC) detects that a loop is happening by observing the PC. When a branch with a negative offset happens, the LCC starts caching the instructions being fetched after this branch instruction. When the same branch instruction is executed again, the LCC knows that a loop is being executed, therefore the instructions are then fetched from the loop cache, hence not from the main memory. The drawback of this implementation is that it only works for loops with straight-line execution, rendering this solution quite limited.

2.2.1.2 Preloaded Loop Cache (PLC)

As the name suggests, the idea (proposed in [6]) is to preload into the cache the loops which represent the best candidates in terms of energy saving. This is performed based on profiling of the application. This solution achieves better energy savings than the DLC, however the obvious downside is the need for profiling, which means that it requires support of a set of external tools. Moreover, it is completely application-dependent.

2.2.1.3 Hybrid Loop Cache (HLC)

Also proposed in [6], it basically combines DLC and PLC into one cache. It offers even better results than the PLC.

Varying the loop cache sizes from 32 to 512 instructions, around 60% to 70% of reduction of fetches from L1 cache is reported, under the Mibench and Powerstone benchmarks.

2.2.1.4 Adaptive Loop Cache (ALC)

Proposed in [24], this solution tries to mimic the results of HLC, but without the need of profiling the application beforehand. It accomplishes that by implementing an enhanced version of the LCC, which takes care of determining during runtime the loops that should be cached. It produces better results than HLC when the cache size is smaller than 128 instructions, otherwise an increase of 5% on average is reported.

2.2.2 Scratchpad

Scratchpad memory (SPM) is a small, fast SRAM memory that can be used to store parts of code or data of an application. It takes up part of the full memory address space, providing fast and low power access to its contents. It is an interesting solution in comparison to caches because it does not contain any of the extra structures a cache needs (tags, muxes and comparators). The challenge then becomes determining which sections of code (or data) should be mapped into the scratchpad, in order to maximize energy savings.

The two basic variants of this approach are regarding whether to perform this allocation dynamically or statically. The former offers more flexibility and explore more opportunities during the lifetime of the application, however, as pointed out in [1], the overhead of moving chunks of memory between the main memory and the SPM might lead to performance degradation in terms of speed and energy in comparison to a static solution.

2.2.2.1 Dynamically allocated

In this type of solution, the set of instructions/data loaded into the SPM changes over time. This change happens with help from the compiler, which places special instructions telling when and what to copy from main memory (or L1 cache, in other words, the next level of memory in the hierarchy).

There are then algorithms focusing on making this dynamic placement as efficient as possible. In [8], basic blocks (BB) are the smallest unit of code that can be transferred to the SPM. Then, based on temporal relationship between the BBs (a metric called concomitance), instructions to move those BBs to SPM are inserted in the code. In [17], "the length of transfer blocks can be adjusted in increments of one instruction". According to [17], their algorithm "achieves 31% instruction delivery energy reduction over even an ideal coarse-grain algorithm".

2.2.2.2 Statically allocated:

There are two ways to perform static placement of code/data into SPM: compiler-aware allocation or post-compiling allocation. They both rely on the same principle, which is identifying the most suitable pieces of code/data to be moved permanently to the SPM. The main difference between these two methods is regarding the input for this identification process. For a compiler-aware allocation, the input is the source code, while for the post-compiler, it is a binary file. While the former is more flexible, the latter is more general, since it can be applied to any scenario, without knowledge or customization of the compiling tools.

Post-compiler allocation: In this approach, the following actions are necessary:

1) Identify, in the executable file, all units of code (or data) that are candidates to be placed in the SPM.

2) Given the set of candidates, solve a basic Knapsack ([15]) optimization problem, that is, find the best suitable set of blocks of code that can be placed in the SPM, in order to minimize energy consumption.

3) Patch the binary file, based on the solution found in item 2.

Both [1] and [29] developed their solutions based on these steps. The most important difference between them is the granularity considered when selecting a candidate to be placed in the SPM. In [1], they work with a fine-grain block boundary, with minimum size equal to one single instruction, whereas in [29] only BB and procedures are considered as possible candidates. Unfortunately, these two works cannot be directly compared because they use

different benchmarks, however both report a significant energy reduction in comparison to a system without SPM.

2.2.3 Associative Memory / Content-Addressable Memory

An associative memory (or Content-Addressable Memory (CAM)) is a type of memory in which one can search an input data against all values present at the memory in parallel, that is, it compares the input data with all stored values at the same time. That is usually used for devices that need very fast search times. On the other hand, it is a very power hungry type of memory. However, if kept at smalls sizes (namely 32 to 64 entries), it can be used for low power devices as well. The general idea of using CAM for caches is to store the tags in a CAM and the cache blocks in a normal SRAM. That way the tag search can be sped up, giving a hit rate of a 32-way associative memory, while keeping the power consumption to lower levels.

In figure 2.3, taken from [33], a basic structure of a CAM used to store the tag part of a cache entry is shown. As can be seen in the figure, the tag is fed into the CAM and all lines from the CAM will have the result of the search at the same time. If there is a match in any of the entries, its output will be driven to '1' (high/VCC) and that can be used to drive the cache line that contains the desired data.

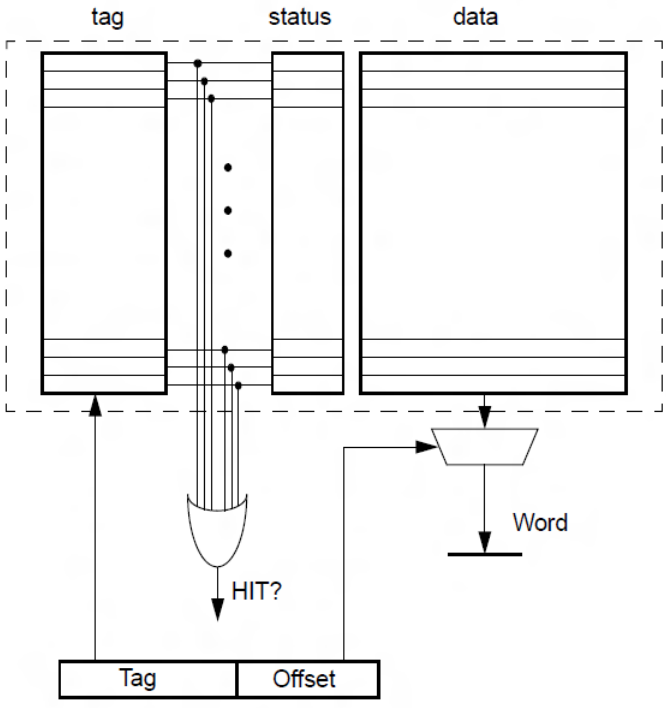

Figure 2.3: CAM Tag.

Although using CAM seems like an interesting solution, besides the fact that size must be kept to a limit, it is also important to mention that CAM memory cells are different from SRAM memory cells at the transistor level, therefore needing special design, making it not possible to use standard memory libraries during the design of the circuit.

In order to cope with the CAM size as the limiting factor, the author on [31] proposes to split the tag memory into two different memories: a CAM and a normal SRAM fully associative memory. A different direction was proposed on [2], in which the idea is to change the CAM cell circuitry to perform serial bit comparisons in order to save energy. It performs the serial bit comparison only for the 4 least signficant bits (LSBs), since most of the hits can be determined by these bits. Then, if it matches, the rest of the comparison is performed in parallel. Problem with this approach: 25% slower than normal CAM.

In [33], the author employs sub-banking, that is, split the memory in smaller banks, in which only one bank is active at a time, based on the address being requested. Each sub-bank contains a CAM memory to store the tags and drive the cache lines, when a hit occurs.

2.2.4 Way-Prediction

In set-associative caches, both the tag arrays and the data arrays are accessed in parallel (as can be seen in figure 2.4, taken from [20]). Thus, on a n-way set associative cache, n ways are accessed in parallel and then, based on the tag comparison, the data from the correct way is selected to be the output. However, in every hit, only one way holds the correct data, therefore the other $n-1$ ways accesses are useless, resulting in wasted energy. The idea is then to apply mechanisms that allow to predict the way that should be accessed next, saving the energy of reading from the other ways.

This kind of solution relies on adding extra structures to help with the prediction, such as extra bits to the Branch Target Buffer (BTB). The big disadvantage of this type of solution is the extra latency added to the access time when a miss prediction happens.

An alternative to way-prediction is to perform serial access to tags and ways, guaranteeing that only the correct way is accessed at all times. The obvious problem is the increase of as much as 60% on access time, as depicted in [22].

A similar idea, but with a slightly different take, is way-halting. Given the same assumption as above, the intention now is, instead of predicting the way, to avoid accessing the wrong ways. This method was proposed on [32].

In order to do that, a small fully associative memory is used to determine the ways that should be accessed. It works in the following way:

The tag bits are split into 2, where the four LSBs are used as address to the fully associative memory. The rationale behind using the four LSBs is that with those bits it is possible

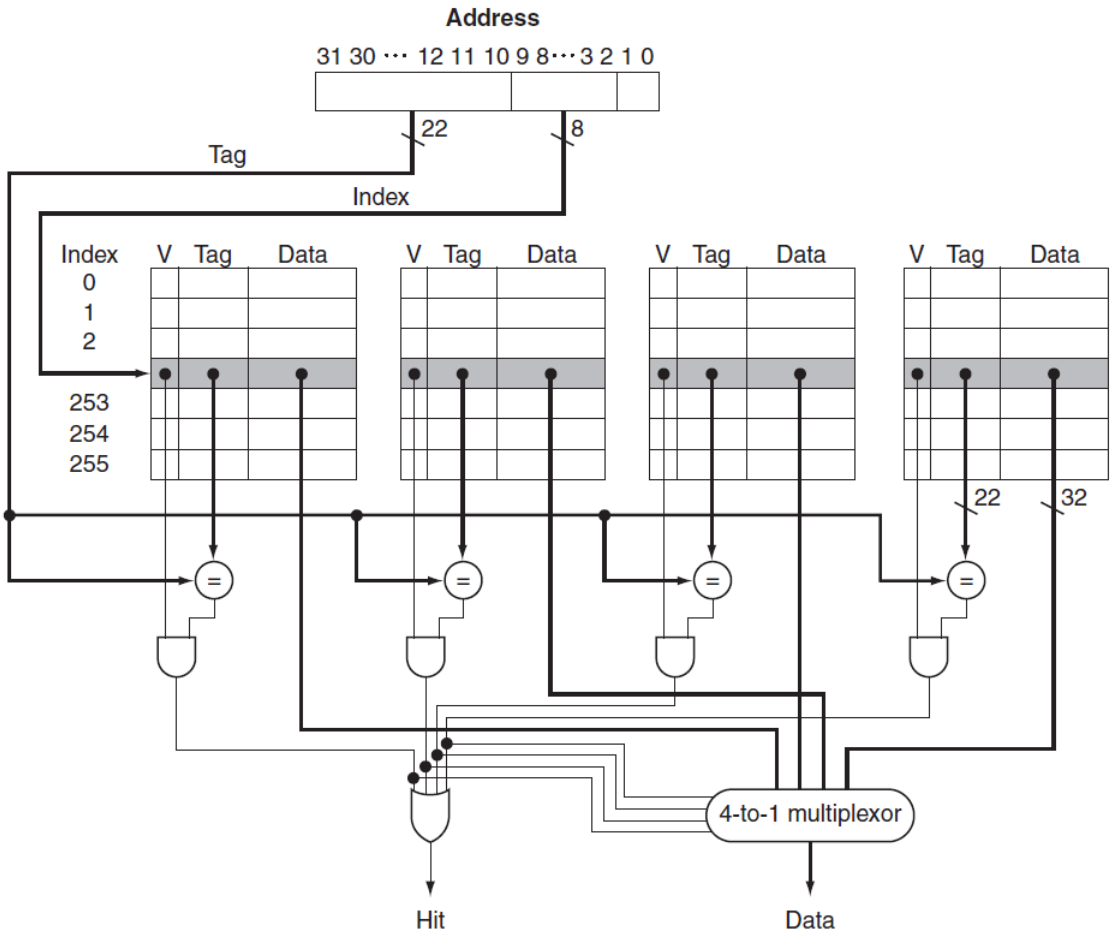

Figure 2.4: Example of 4-way Associative Cache.

to determine, most of the times, whether a hit happens or not. Then the fully associative memory is accessed in parallel with the index decoder. Thus, only those ways where there was a hit on the fully associative memory will actually be accessed, saving the dynamic power that would be used to read from the other ways.

The interesting advantage over the way-prediction solution is the fact that there is no extra latency added to the cache access, since the fully associative tag comparison happens in parallel with index decoding, as depicted on figure 2.5, taken from [32].

Instead of using CAM, the author implements the fully associative memory using normal SRAM memory cells.

For a 4-way-8K bytes cache, under the Mediabench, Powerstone and Spec2k benchmarks, reductions on power consumption ranging from 45% to 60% are reported.

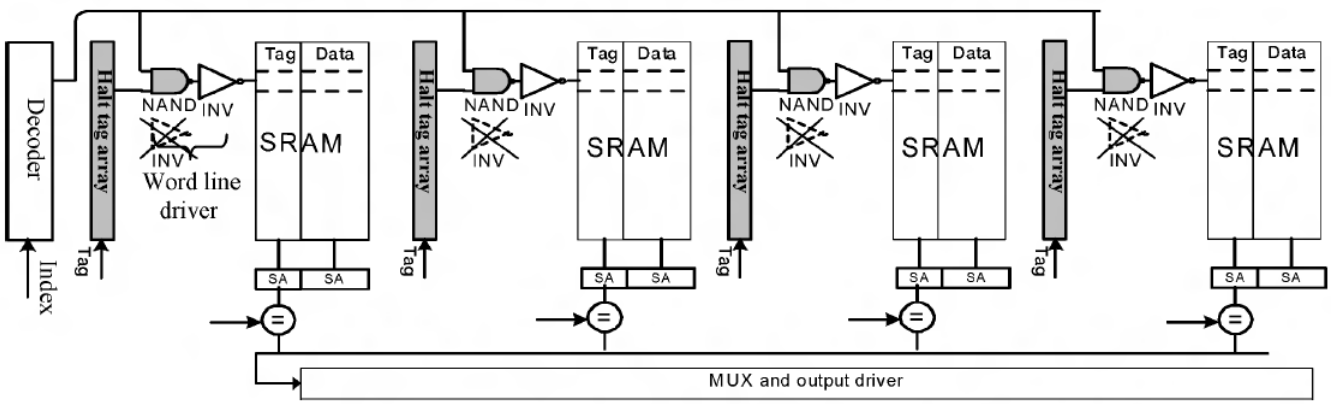

Figure 2.5: Baseline Architecture for Way-Halting Cache

2.2.5 Indexing / Hashing

As described in [30], a direct-mapped (DM) cache has many advantages in comparison to n-way associative caches. Some of them are listed below:

- Less power consumption per access.
- Less area (only one array of data and tags, no multiplexers).
- Faster access times.
- Easier to implement.

However, the biggest problem with DM caches is the higher miss rate, when compared to n-way associative caches. This elevated number of misses comes from the fact that many addresses end up being mapped to the same location in the cache, resulting in what is called conflict misses. These facts are behind the motivation for the solution presented in this section.

It is important then to understand that the mapping from the main memory address to the cache address is basically a hashing function. The standard way to realize this translation between addresses is to take n LSBs from the full address and use it as the address of the cache line, as can be seen on figure 2.6, taken from [20].

Although very easy to implement, this form of mapping is far from ideal, rendering many conflicts in the cache. The intention of indexing/hashing is then to change the access pattern of the memory address by using some hashing functions with the set of indexes bits. This is the general idea of all hashing/indexing approaches. What may varies is how the implementation is done and rather or not the hashing function is dynamically reconfigurable.

On [30] for example, reconfigurable decoders are used. Through prior profiling of the application, the configuration of the decoders are determined. It is reported an improvement

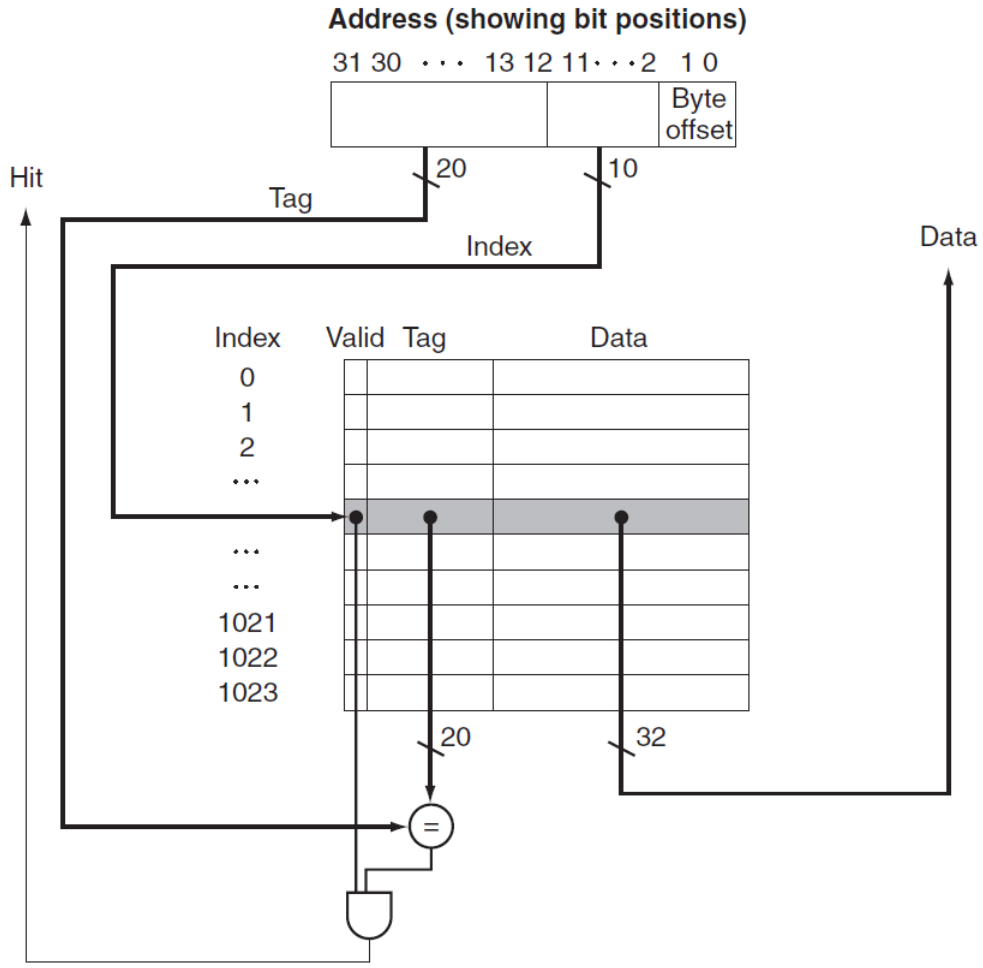

Figure 2.6: Basic Direct-Mapped Cache.

in the number of conflicts, achieving similar numbers of a 2-way cache. On [5], a zero-overhead scheme is presented, where the hashing function is nothing more than a bit-selection function, that is, based also on application profiling, a heuristic algorithm is used to determine the set of address bits that should be used to address the cache. In this specific work, it is reported more improvements for data than to instruction caches. The bit-selection function is called a zero-overhead solution because no extra hardware is needed to perform the hashing function, differently from the XOR-based functions, in which the hashing function relies on XOR gates placed between the main memory address and the cache address. A similar solution to [5], but with an optimal algorithm, is described on [18]. On [27], the algorithm presented on [18] is adapted to XOR-based functions.

2.2.6 Tag Omission

Because caches are much smaller memories than the main memory, it cannot hold all data (or instructions) at all time. That is why every cache line is also composed by the tag, a portion of the address of the original location of the data in the main memory, to allow a checking to be performed every time an access to the cache happens, making sure that the right data is being fetched from the cache. Nevertheless, tag checking during every cache access contributes to the total amount of power consumed by caches. This is the motivation behind the solutions presented in this section. Note that this is somehow intertwined with the idea of loop caches, which are tagless caches, and scratchpads. The key difference is that loop caches and scratchpads do not even have tag arrays in their architecture, while the aim of the solutions presented in this section is to minimize the access to the existent tag arrays of the caches.

The first idea, as presented in [16], is based on the concepts of intrablock and interblock flows. Assume that two instructions are fetched from the cache. If both instructions fall into the same cache line (or intrablock), then for the second instruction it is not necessary to perform tag comparison because it is certain that it will be a hit. Therefore only for interblock flows the tag check must be performed.

In [7] this notion was extended to be able to handle a larger window of memory accesses without the need to check tags. The basic idea behind this approach is the fact that the state of caches changes only when there is a miss, which means that new instructions must be fetched from the main memory. Thus, between two misses, the state of the cache remains stable. Therefore if an instruction is accessed repeatedly during this stable-time (as called by the author of the paper), only at the first reference a tag check has to be performed. To be able to detect the conditions for not performing unnecessary tag checks, the cache proposed in the paper, called History-Based Tag-Comparison (HBTC) cache, records execution footprints in an extended Branch Target Buffer (BTB) [7]. It is important to note that the HBTC cache works only with direct-mapped instruction caches.

For a DM cache with 16KB - 32 bytes on the cache line a reduction of about 90% on tag comparisons was reported, leading to about 15% reduction on energy consumption.

The drawbacks of this solution are the need of a BTB in the microprocessor and the fact that extra cycles are added to the cache access when an invalid footprint is found.

Chapter 3

Method

.

In this chapter, the preparations and the methodology employed to perform the desired experiments are outlined. Firstly, the cache architectures chosen for evaluation are presented, along with the rationale supporting such decisions. Later, the simulation environment, in which the actual evaluations take place, is described. And finally, the experiments realized in the scope of this project are listed, along with an explanation on how the energy consumption is calculated for each scenario.

3.1 Architectures Selected

The basic criteria used to select which architectures should be evaluated was obviously energy savings capability. However, because this work focus on a specific architecture (depicted previously on figure 1.2), the following requirements and constraints were also taken into consideration:

- The internal signals of the microprocessor are not visible to the memory system, all that is available are the addresses and data (instructions) that come through the bus.
- The whole system should work under the same system clock.
- Preferably, no processor stalling should happen.
- Area and current leakage should be kept to a minimum.
- The microprocessor does not have a Branch Target Buffer (BTB).
- Given the fact that a proprietary compiler is being used, no compiler techniques can be employed.

Time-to-market is usually also an important constraint when searching for new solutions in embedded systems. The same constraint was applied to this project, in which only solutions that were rendered feasible in the time frame available were examined.

The selected architectures are presented below.

3.1.1 Hashing / Indexing

As explained in section 2.2.5, DM caches are faster, smaller and consume less energy per access than its n-associative counterparts. Moreover, the results reported on [4] demonstrate that, for a system with a very similar configuration to the one in this project, DM cache outperforms 2-way and fully associative caches in terms of energy consumption.

The main problem with DM, as mentioned before, is the high number of conflicts. Therefore, a solution that improves DM in its weakest point is definitely worth looking at.

The solution presented in [19] was selected for implementation. It contains an optimal algorithm, based on Binary Decision Diagrams (BDDs) and Algebraic Decision Diagrams (ADDs), for finding the set of addresses bits that should be used to perform the mapping between the main memory and the cache. In [28], an implementation with XOR-mapping function is proposed, with slightly better results than [19]. However, giving the limited time available, [19] was chosen, leaving room for future work to be done with [28].

Before jumping into specifics about the algorithm, it is also important to mention that the DM caches used in this project have a line size of only one word (32 bits). That is the case because 32 bits is also the width of the data bus between the microcontroller and the Flash memory. If a bigger line size was chosen, for example 64 bits, it would be impossible to fill the cache line during one single clock cycle, which would then result in processor stalling.

3.1.1.1 Bit-selection function explained

It is not the intention of this section to fully describe how the bit-selection algorithm works, for that the reader is referred to [19], but rather briefly explain the idea behind it.

(The following example was taken from [19], to which a slight modification was made and more details were added for ease of understanding).

Assume that the main memory of a system is composed of 8 entries, hence from $\log_2 8 = 3$, it can be conclude that 3 bits (let's call them $a_2a_1a_0$) are needed to address this memory. Now assume that a DM cache is present as well, with 4 entries and therefore 2 bits to address it, resulting in the configuration shown in 3.1.

Given this basic configuration, the following options are available for bit-selection (that is, which bits should be used to address the DM cache):

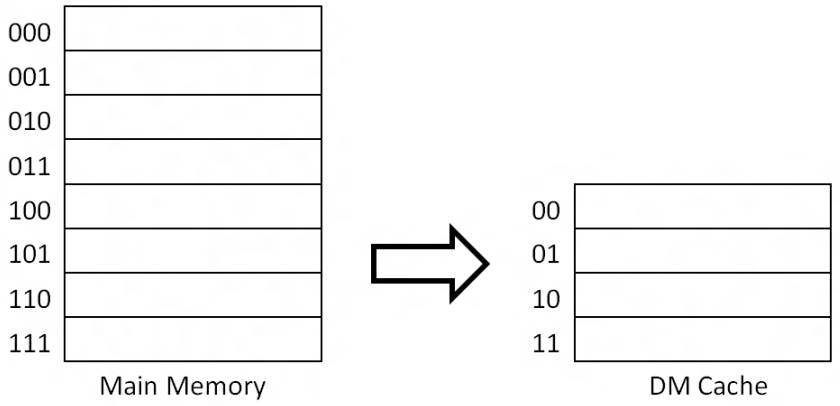

Figure 3.1: Basic Main Memory & DM cache Configuration

1. a_1 and a_0, which is the same as the standard bit-selection of DM caches (the least significant bits).

2. a_2 and a_0.

3. a_2 and a_1.

Now assume the following trace (sequence of addresses) is executed: $0, 1, 5, 3, 1, 5, 6, 7, 6$. What the algorithm calculates, in a symbolic way, which is translated to BDDs and ADDs, is the number of conflict misses that would occur if a given bit-selection was employed.

For example, to calculate the number of conflict misses that happens between the first access to address 1 (001_2, in binary) and its next occurrence, the algorithm checks in which situations contents of address 1 would be evicted from the cache.

Therefore, for the sequence $001_2, 101_2, 011_2, 001_2$ (in binary), it is easy to see that:

1. if a_1 and a_0 are used, accessing address 101_2 would evict 001_2 from the cache, resulting in 1 conflict miss when the next access to 001_2 happens.

2. if a_2 and a_0 are used, accessing address 011_2 would also evict 001_2 from the cache, resulting in 1 conflict miss when the next access to 001_2 happens.

3. if a_2 and a_1 are used, no conflict happens, therefore 001_2 would still be present on the next access to 001_2.

The algorithm performs this calculation for the entire program trace and outputs the mapping with less number of conflict misses.

For more details about the implementation, please check appendix A.

3.1.2 Scratchpad

Scratchpads match perfectly the requirements of the target system: low-power, small and fast. Because there are no tags involved, there also no conflicts, which means no processor stalling. Furthermore, the target system already has a SRAM memory that can be used as a SPM.

The main decision to be made was regarding how to map instructions to the SPM. Given that no compiler techniques could be employed, the available solution was the post-compiling approach.

As explained in section 2.2.2, the two post-compiling solutions differ basically on the granularity of the code considered as a candidate to be allocated in the SPM. For this project, only procedures are considered as candidates. This granularity was chosen for the following reasons:

- Procedures do not need any extra branch instructions to be inserted in the original binary file, making it easier to perform patching of the binary file later.
- Given the fact that the amount of SRAM available to be used as SPM in the target system is not an issue, as reported in [29], using procedure granularity results in the same amount of energy saving as basic block granularity for SPM of the size of 4K bytes.

As previously described on 2.2.2.2, 3 steps are needed in order to fully perform the mapping of code objects to the SPM. Step 1, namely, identifying the procedures in the binary file, was developed in previous works. Step 3, or patching the binary file, is not needed in order to perform the desired simulations. Step 2 is described below.

3.1.2.1 Optimization problem

As explained previously, determining which procedures must be place in the SPM is an optimization problem, for which many solutions are available in the literature. For this particular project, MinKnap [21] was adopted.

The basic problem to be solved is stated in the following equations [21]:

$$\begin{gathered}\text{Maximize } \sum_{j=1}^{n} p_j x_j \\ \text{Subject to } \sum_{j=1}^{n} w_j x_j \leq c \\ x_j \in \{0,1\}, j = 1, \ldots, n\end{gathered}$$

where:

- n represents the total number of procedures found in the binary file.

- p_j represents the profit of moving a procedure j to the SPM. How this profit is calculated is explained later on this section.
- w_j stands for the cost of moving a procedure j to the SPM. It is basically its size in bytes.
- x is the solution vector. When x of a given j is equal to 1, it means that the procedure j must be placed in the SPM.
- c is the constraint, that is, the total size of the SPM.

Calculating the profit vector: As will be explained in the section 3.3, two different solutions comprising SPM will be considered. One is using Flash and SPM only and the other is using Flash, Cache and SPM. Therefore, the profit of moving a procedure to the SPM depends on which solution is being employed.

Flash and SPM only: In this case, the profit of moving a single procedure from the Flash memory to the SPM is calculated as:

$$p_j = a_j \times (E_{flash} - E_{spm})$$

where p_j represents profit of moving procedure j, a_j is the number of total fetches, during the execution of an application, that fall within the limits of procedure j, E_{flash} is the energy spent when reading instruction from the Flash memory and E_{spm} is the energy spent when reading instruction from the SPM.

Flash, Cache and SPM: Now the calculation is not that straightforward anymore because the total energy consumption of a procedure depends on how many hits and misses happened in the cache while trying to fetch instructions that fall within its limits. Therefore, the profit must be calculated as follows:

$$p_j = (h_j \times E_{cache} + m_j \times (2 \times E_{cache} + E_{flash})) - a_j \times E_{spm}$$

where h_j is the number of hits that happen when accessing procedure j, m_j is the number of misses and E_{cache} is the energy spent accessing the cache.

3.2 Simulation environment

For clarity, the system being simulated is shown again on figure 3.2.

To perform the desired evaluations, a high-level simulation environment, that mimics the accesses to the memory system, was developed in Python [26]. This environment was used

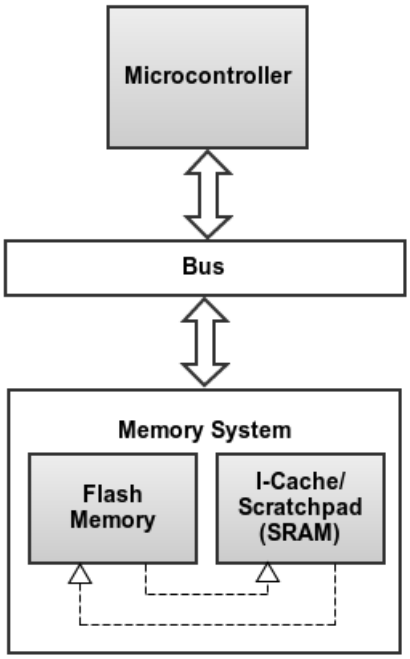

Figure 3.2: Simulated system.

basically to calculate the total energy consumption for an execution of a given program trace. A basic overview of the environment can be seen in the class diagram provided in figure 3.3.

In order for a simulation to happen, the following elements should be present:

- Simulation object.
- Program trace.
- Memory System.

The simulation object is in control of the execution. It sets up the system with the desired parameters and runs the simulation. The program trace acts as the processor, providing the addresses of instructions that should be fetched from memory. The memory system is the core of the environment, since it is its behavior that the simulation is trying to reproduce. Its basic functionality is implemented in the `Read` method.

A memory system can be composed of many memory elements, thus being able to represent different hardware configurations. For example, to simulate a system composed of a Flash memory and DM cache, the objects `DMCache` and `FlashMemory` can be combined to form a memory system of the type `CachedMemorySystem`.

It is very important to mention that, although a high-level simulation environment, such as the one used here, increases dramatically the time spent on the evaluation phase (not only

because it is faster to run, but also because it is quicker to implement than a more detailed version written in Verilog, for example), it hides essential issues that must not be overlooked when it comes to actual hardware implementation.

For example, for the `CachedMemorySystem`, the `Read` method is implemented as presented below.

```
def Read(self, address):

    hit = self.dm_cache.Read(address)
    self.cache_energy += self.dm_cache.energy_per_read
    if hit == 1:
        return
    else:
        self.miss_count += 1
        self.cache_energy += self.dm_cache.energy_per_write
        self.flash_energy += self.flash.energy_per_read
```

What this method is modeling is the following:

1. The DM cache checks whether the contents of `address` is in the cache. If it is, it adds the energy spent reading from cache and no access to the Flash memory is made.
2. If there is a miss, then data must be read from the Flash memory and written into the cache. The energy consumption of both accesses is computed.

What is implicitly assumed in this scenario is the fact that the entire reading operation fits the requirements described in section 3.1, in particular those regarding time, which are reproduced below:

- The whole system should work under the same system clock.
- Preferably, no processor stalling should happen.

In order for this assumption to be valid, an analysis of which hardware implementation could support such requirements was performed, resulting in the design shown on figure 3.4.

This setup works because the access to cache and flash combined takes less time than the length of the clock cycle (62.5 ns).

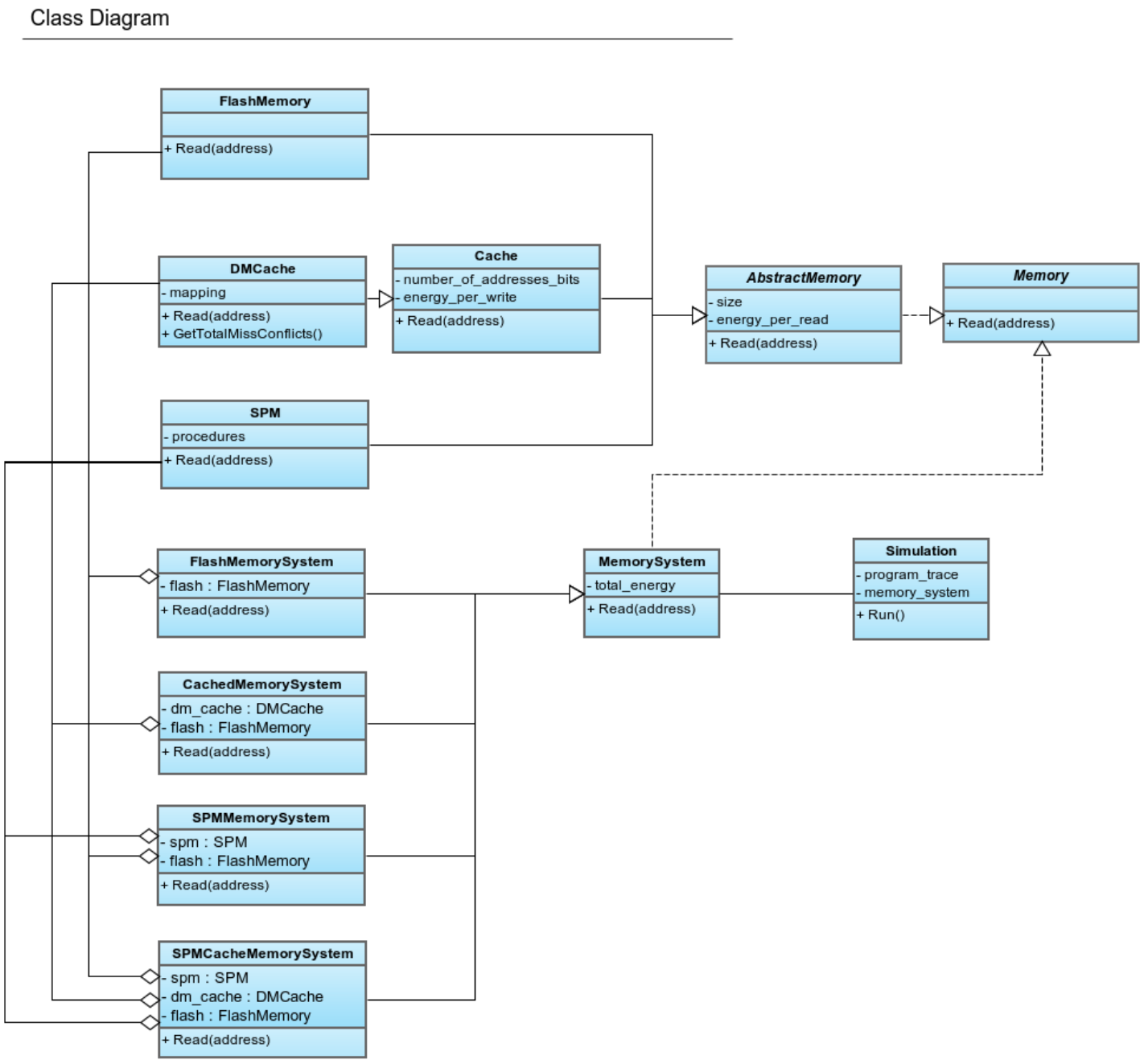

Figure 3.3: Simulation Environment Class Diagram

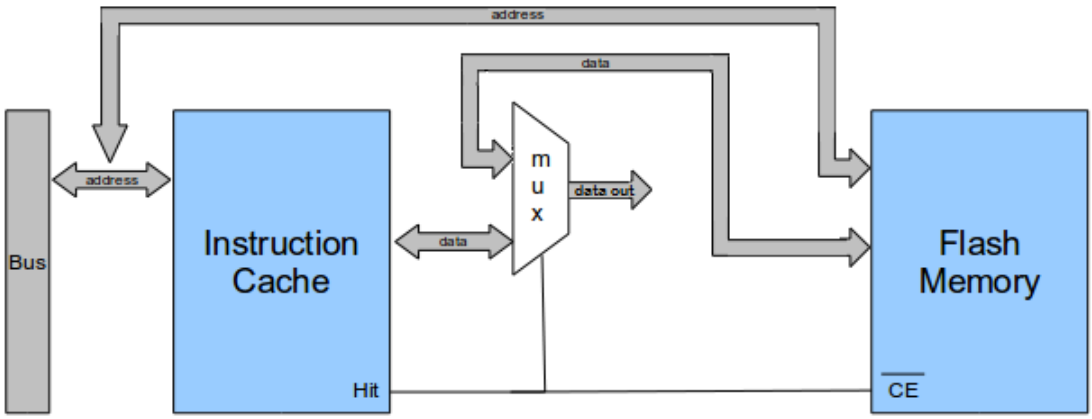

Figure 3.4: Flash memory + DM cache hardware implementation

3.3 Experiments

For this project, the memory systems modeled and evaluated on the simulation environment described previously are listed below. Between brackets is the name of the class used to model each scenario.

- Flash memory only. Used for comparison against all other scenarios. [`FlashMemorySystem`]
- Flash memory + DM cache (with sizes 128, 256, 512, 1K, 2K, 8K and 16K bytes). [`CachedMemorySystem`].
- Flash memory + DM cache with bit-selection (with sizes 128, 256, 512, 1K, 2K, 8K and 16K bytes). [`CachedMemorySystem`].
- Flash memory + Scratchpad (with sizes 512, 1K, 2K and 4K bytes). [`SPMMemorySystem`]
- Flash memory + DM cache with bit-selection (with sizes 128, 256, 512, 1K, 2K, 8K and 16K bytes) + Scratchpad (with sizes 512, 1K, 2K and 4K). [`SPMCacheMemorysystem`]

All the relevant parameters used throughout the simulations are listed on tables 3.1, 3.2 and 3.3. The parameters related to the Flash memory were provided by Nordic Semiconductor, while the DM cache and SPM parameters were obtained as explained in the following section.

Parameter	Value
Clock	16 MHz
Flash size	256 kB
Flash energy per read	0.500 nJ
Flash leakage power	0 W
Flash access time	30 ns
Cache size	128 - 16384 B
Cache line size	4 B
Scratchpad size	512 - 4096 B

Table 3.1: Main parameters used during simulations.

The

3.3.1 Energy Models

The first option when it comes to energy models for caches is the CACTI [12] tool. It models the dynamic power, access time, area and leakage power of caches. Although widespread in its use, this tool was not employed for this project because it is targeted on newer technologies

Size (bytes)	Energy per read (nJ)	Energy per write (nJ)	Access time (ns)
176	0.0174	0.0186	1.053
344	0.0264	0.0276	1.309
672	0.0276	0.0294	1.360
1312	0.03	0.0324	1.456
2560	0.0342	0.0378	1.642
4998	0.0402	0.0468	2.006
9728	0.0433	0.0495	2.053 (*)
18944	0.0473	0.0545	2.309 (*)

Table 3.2: Cache parameters. (*) Obtained through interpolation.

Size (bytes)	Energy per read (nJ)
512	0.0186
1024	0.0222
2048	0.024
4096	0.0312

Table 3.3: SPM parameters.

only. The technology node of the cache memories to be evaluated in this project is larger than the minimum available on the CACTI tool, which is 90 nm.

The Artisan tool [13] was used instead. This tool models a SRAM of a given size, targeting a specific hardware implementation. When modeling the cache, the word size of the memories being modeled with this tool reflected the size of tag array plus the size of the data array of a cache line. Obviously, the tool does not model the comparator used to check for hits in the tag array, nor the AND gate used to assert a hit. These components however do not have a big impact in the energy consumption of a direct-mapped cache.

The Artisan tool available models only memories up to around 4K bytes, therefore the energy per values for caches of size 9728 bytes and 18944 bytes were obtained via interpolation.

An issue detected with the values provided by the Artisan tool was that they were too conservative. It did not match exactly the values obtained through power characterization performed by Nordic Semiconductor on a memory of same size. Therefore the values provided by the Artisan were divided by 2 to try to approximate them to more plausible ones.

The final values for energy per access can be seen on tables 3.2 and 3.3.

3.3.2 Calculating Energy Consumption

The energy consumption calculation depends on the memory system being evaluated. For the memory systems with cache, it is necessary to know the number of hits and misses. For every miss in the cache, data must be retrieved from the Flash memory and then written in the cache. For the memory systems with SPM it is simpler, since it is necessary to compute only the number of times the SPM is accessed plus the number of times the Flash memory is accessed. For the scenario which combines DM cache and SPM, both contributions are taken into account.

The equations used in these different setups are:

DM cache only:

$$FlashEnergyConsumption = NumberOfMisses * FlashEnergyPerRead$$
$$CacheEnergyConsumption = NumberOfMisses * (CacheEnergyPerRead + CacheEnergyPerWrite) + NumberOfHits * CacheEnergyPerRead$$
$$TotalEnergyConsumption = FlashEnergyConsumption + CacheEnergyConsumption$$

Scratchpad only:

$$FlashEnergyConsumption = NumberOfAccessesToFlash * FlashEnergyPerRead$$
$$SPMEnergyConsumption = NumberOfAccessesToSPM * SPMEnergyPerRead$$
$$TotalEnergyConsumption = FlashEnergyConsumption + SPMEnergyConsumption$$

DM cache and SPM combined:

$$FlashEnergyConsumption = NumberOfAccessesToFlash * FlashEnergyPerRead$$
$$SPMEnergyConsumption = NumberOfAccessesToSPM * SPMEnergyPerRead$$
$$CacheEnergyConsumption = NumberOfMisses * (CacheEnergyPerRead + CacheEnergyPerWrite) + NumberOfHits * CacheEnergyPerRead$$
$$TotalEnergyConsumption = FlashEnergyConsumption + SPMEnergyConsumption + CacheEnergyConsumption$$

It is important to notice that there is no factor for static power consumption in the equations above. That is the case because the values obtained for energy per read and write for the memories evaluated already include this component.

3.3.3 Benchmark

For this project, only one program was available for evaluation. The software being executed by the microcontroller system is a Bluetooth Low Energy application, running a heart-rate

monitor profile. It is an example application used in the Bluetooth Software Development Kits available at Nordic Semiconductor.

Chapter 4

Results

In this chapter the results from the different scenarios simulated are presented, along with comparisons among them.

4.1 Direct-Mapped Cache with Optimal Indexing

For each of the cache sizes evaluated, the optimal indexing algorithm described in section 3.1.1.1 was run for the application being analyzed, resulting in the indexing presented in the table 4.1, in which a_n means that the bit n of the address bus is used to compose the set of bits used to address the DM cache. Thus, $a_{13}a_5a_4a_3a_2$ means that bits 13, 5, 4, 3 and 2, in this order, are used to address the cache line in the DM cache.

For ease of comparison, the indexing using in standard DM caches is also presented in this table. The DM cache configuration with the standard mapping is called the baseline implementation, because it is the one used as a base for comparison with the results obtained with the optimal indexing mapping.

Size (bytes)	Baseline Indexing	Optimal Indexing
176	$a_6a_5a_4a_3a_2$	$a_{13}a_5a_4a_3a_2$
344	$a_7a_6a_5a_4a_3a_2$	$a_{10}a_7a_5a_4a_3a_2$
672	$a_8a_7a_6a_5a_4a_3a_2$	$a_{11}a_7a_6a_5a_4a_3a_2$
1312	$a_9a_8a_7a_6a_5a_4a_3a_2$	$a_{11}a_8a_7a_6a_5a_4a_3a_2$
2560	$a_{10}a_9a_8a_7a_6a_5a_4a_3a_2$	$a_{14}a_{11}a_{10}a_7a_6a_5a_4a_3a_2$
4998	$a_{11}a_{10}a_9a_8a_7a_6a_5a_4a_3a_2$	$a_{15}a_{10}a_9a_8a_7a_6a_5a_4a_3a_2$
9728	$a_{12}a_{11}a_{10}a_9a_8a_7a_6a_5a_4a_3a_2$	$a_{15}a_{12}a_{10}a_9a_8a_7a_6a_5a_4a_3a_2$
18944	$a_{13}a_{12}a_{11}a_{10}a_9a_8a_7a_6a_5a_4a_3a_2$	$a_{14}a_{12}a_{11}a_{10}a_9a_8a_7a_6a_5a_4a_3a_2$

Table 4.1: Baseline and Optimal Indexing.

4.1.1 Hits and Misses Rates

As explained before, the main motivation behind finding an optimal indexing for DM caches is to increase the hit rate, that is, the number of times data is found directly in the cache, without the need to go to the main memory to fetch it. That, obviously, reduces the miss rate, which is the number of times the data was not found in the cache and therefore an access to the main memory was needed, resulting in a penalty from the point of view of energy consumption.

In the figure 4.1 a comparison between the hit rates of the baseline and the optimal indexing implementations is presented. The indexing version presents a better hit rate for all cache sizes evaluated. It is interesting to note that the amount of improvement from one size to another is not constant in any way. For instance, for cache size equal 176 bytes, the improvement in the hit rate is better than for cache size equal 2560 bytes. This is due to the fact that the optimal indexing is completely application-dependent. That means that an optimal indexing can vary greatly from one application to the other.

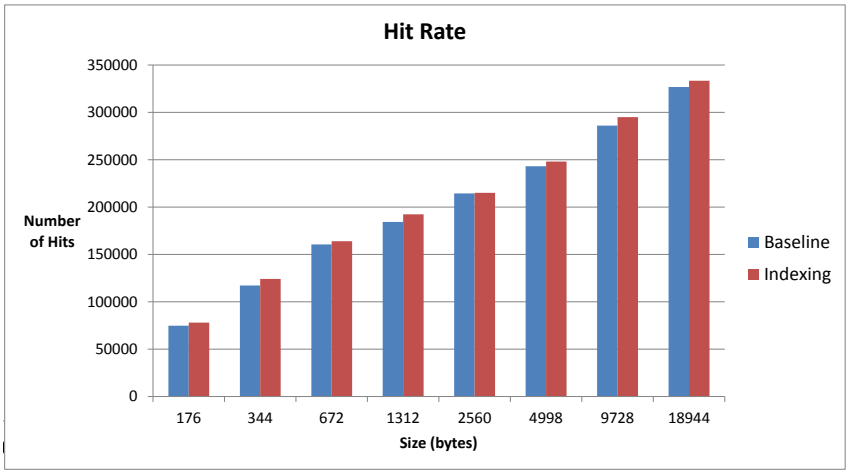

Figure 4.1: Hit rates for baseline and indexing implementations.

Showing the same result, but with a different metric, the figure 4.2 presents the miss rates for each cache size evaluated.

An interesting result is presented on figure 4.3, in which it is possible to see how much improvement was achieved by using the DM cache with indexing in comparison with the baseline implementation. Note again, that for the cache size equal 2560 bytes the improvement was almost 0%.

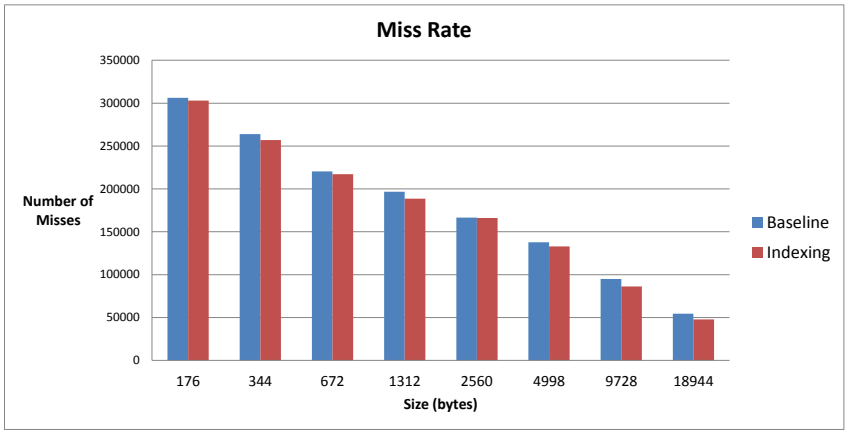

Figure 4.2: Miss rates for baseline and indexing implementations.

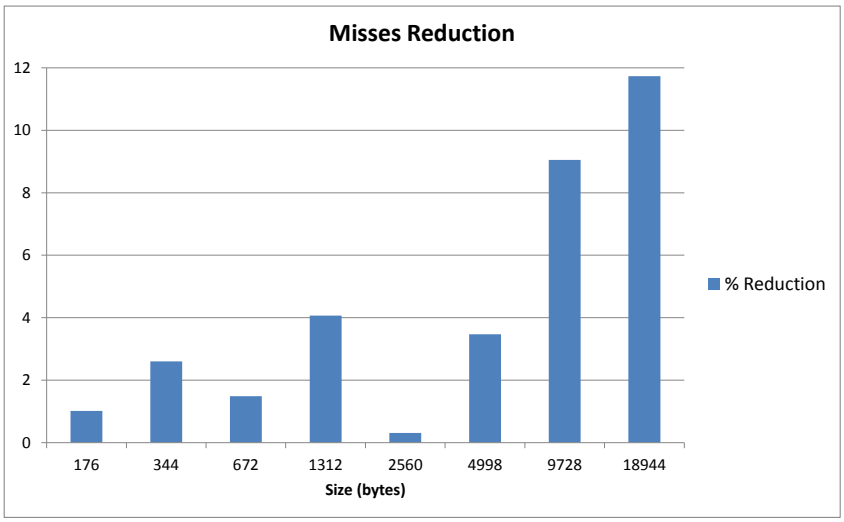

Figure 4.3: Miss reduction from baseline to indexing, in percentage.

4.1.2 Energy Consumption

While hit rates are an interesting metric, only by analyzing the energy consumption of each solution that any conclusions can be drawn. However, firstly, it is interesting to confirm one of the main assumptions of this project, which is that adding a small SRAM to a memory system can greatly improve its energy efficiency. This is exactly what the figure 4.4 shows. A memory system composed of a Flash memory plus a DM cache consumes from 14% to 76% less energy than a system composed of only a Flash memory. For a cache as small as 176 bytes, savings of 14% can be achieved.

Regarding the comparison on energy savings between the baseline and the indexing implementations, the figure 4.5 summarizes the results obtained. As expected, as the hit rate increases, the total energy consumption decreases. Following the same trend of the hit

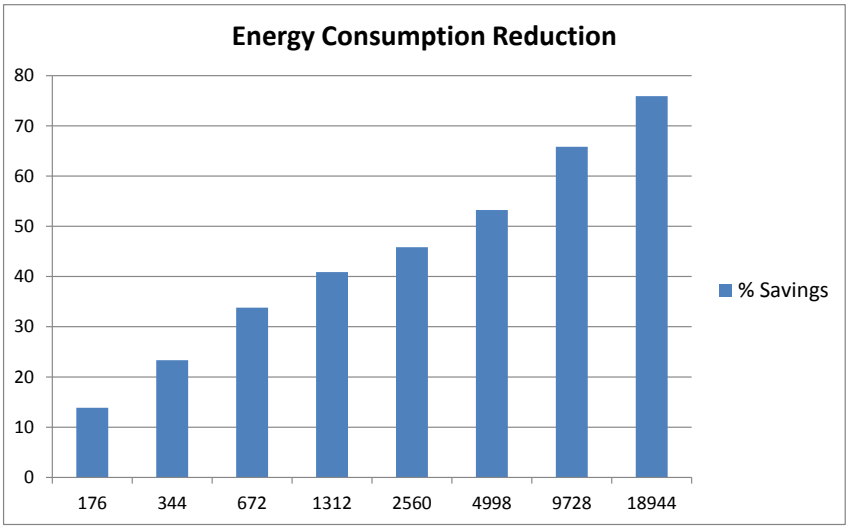

Figure 4.4: Energy savings by adding a DM cache to the memory system.

rates, the indexing version is slightly better than the baseline version.

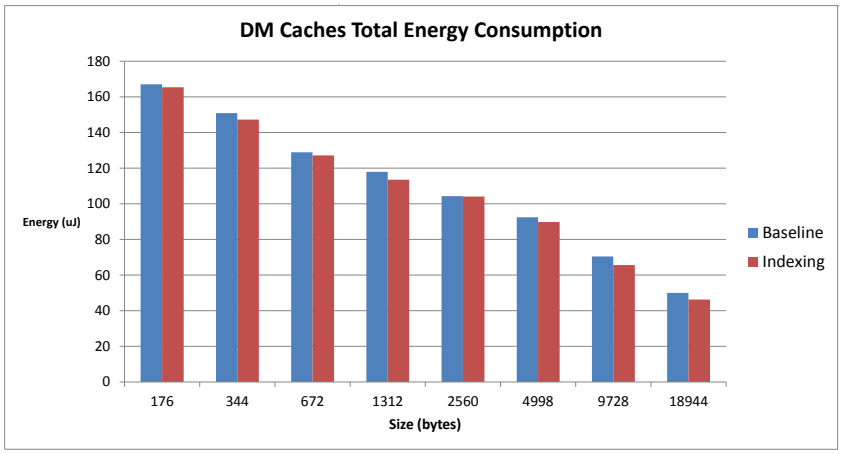

Figure 4.5: Energy consumption of baseline and indexing implementations comparison.

On figure 4.6 it is presented the improvement on energy savings achieved for the indexing implementation in comparison to the baseline version. Not surprisingly, it presents exactly the same pattern on the improvement on miss reductions seen on figure 4.3.

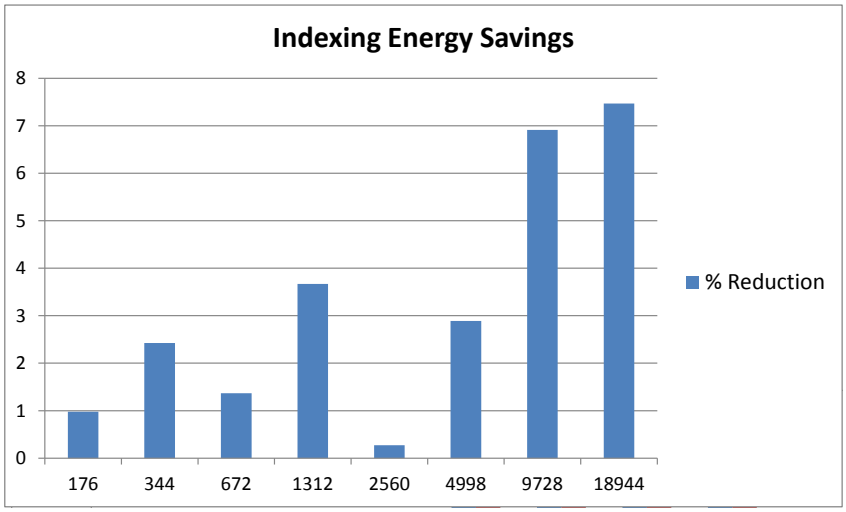

Figure 4.6: Energy reduction from baseline to indexing, in percentage.

4.2 Scratchpad

For each of the SPM sizes evaluated, the optimization problem described in section 3.1.2.1 was solved. As explained before, that meant finding the set of procedures to be allocate into the SPM.

Since SPMs do not actually have the concept of hit rates, only the values regarding energy consumption are presented in the following section.

4.2.1 Energy Consumption

The impact of having a SPM memory added to the memory system is shown on figure 4.7. As for the DM caches, it is clear that adding a SPM to the memory system greatly improves its energy efficiency.

It is important to note that two different scenarios were evaluated in terms of SPM. The first one assumes that an already existent SRAM is present in the system and up to 4K bytes are allowed to be used as SPM (while it certainly would be beneficial to have a bigger SPM, more than 4K bytes would take up too much of the total amount of SRAM available). This is the actual setup of the microcontroller system that this project has as target. However, for evaluation purposes and to allow for more comparisons to be made, a different scenario was envisioned in which SPMs of different sizes were added to the Flash memory to compose the memory system.

These two setups differ basically in the energy per read of the SPM. For the first scenario, a SRAM memory of fixed size was used, which implies that the energy per read for all different

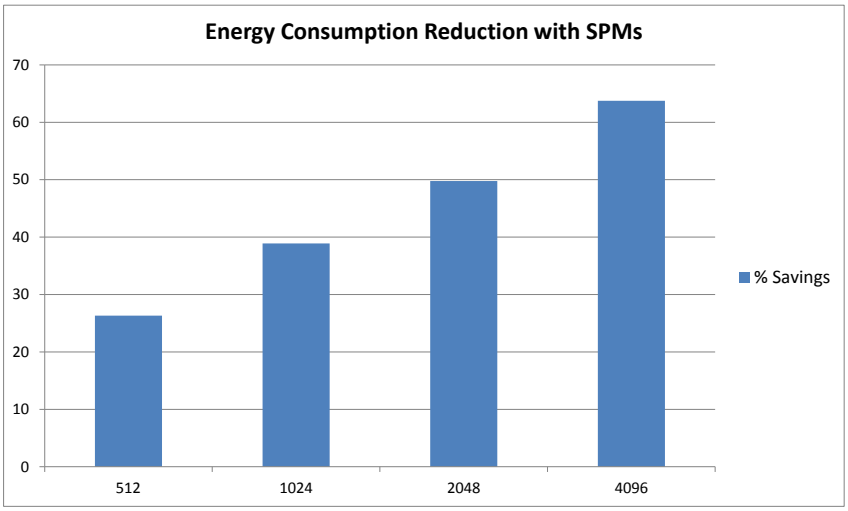

Figure 4.7: Energy savings by adding a SPM to the memory system.

sizes (from 512 bytes to 4K bytes) is the same, because the memory is the same, which is a 4K bytes SRAM. On the other hand, for the second scenario, different values for energy per read were used for each SPM size. These values were already presented in table 3.3.

Results from both simulations are presented in the figure 4.8, in which SPM-Fixed refers to the 4K SRAM memory while SPM-Variable assumes that the size of SPM added to the memory system was not constant. As can be seen in this figure, the difference in energy savings between these two scenarios is rather small.

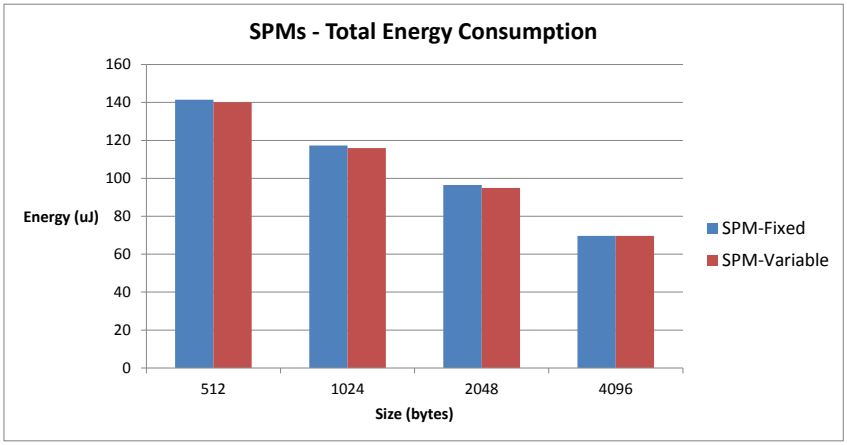

Figure 4.8: Energy consumption of memory systems with SPMs.

4.3 DM cache plus SPM

For this type of memory system, composed of a Flash memory, DM cache and SPM, all 64 possible combinations were simulated. The results from these simulations are summarized on figures 4.9 and 4.10. (Please note that a *(I)* added to the label of a curve means that it is using the Indexing solution, as explained in the previous chapters).

In figure 4.9 all possible combinations were compared to the DM cache solution, named DM cache-only. It is clear that the solutions combining DM cache and SPM are much better than the DM cache-only ones. In particular for small cache sizes, the contribution of having a SPM added to the system is paramount, improving the energy efficiency in about 32% for a cache size of 128 bytes, for example.

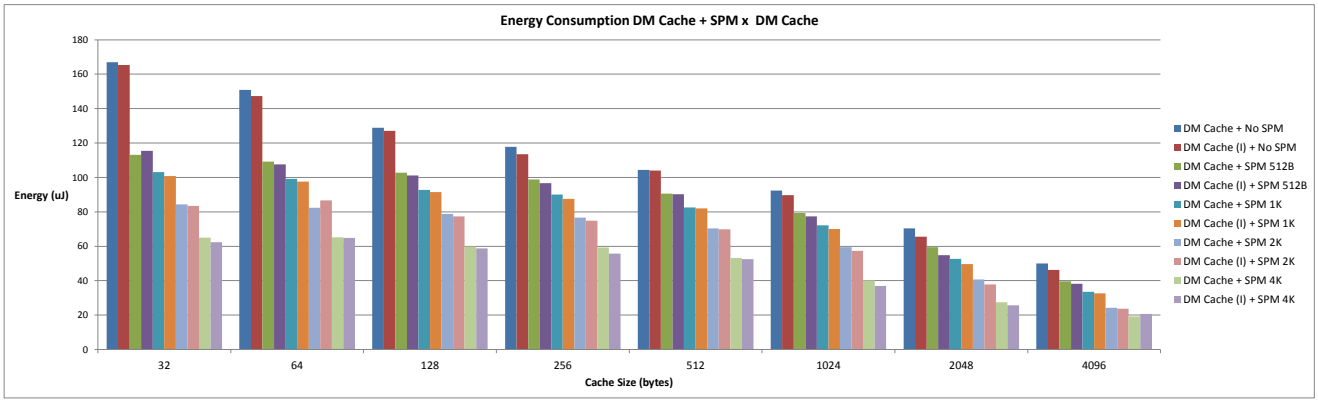

Figure 4.9: Energy consumption of DM cache-only versus DM cache + SPM solutions.

An interesting observation can be made by comparing the solutions for the DM caches with standard mapping and those with optimal indexing. Again for the cache size of 128 bytes, the baseline (or the one with standard mapping) implementation offers better results than the indexing version. This is a completely normal outcome because the optimal indexing was calculated based on the fact that all accesses to the memory were cacheable, that is, the algorithm used as input the entire program trace. However after the optimization problem was executed, and some portions of the program trace were re-mapped to the SPM, the set of addresses that the cache sees has changed, therefore changing the balance of mapping of addresses to the cache. A possible improvement could be made by running the Indexing algorithm again to calculate the optimal mapping for the new set of addresses. While it seems as an interesting idea, based on the results presented in the previous sections, it is unlikely that this would ensue in big changes compared to the values already obtained and displayed in figure 4.9.

In figure 4.10 all possible combinations were compared to the memory system composed of a SPM but no cache, named SPM-only solution. As before, the combined solutions are better than the SPM-only ones. It is worth noting the big impact that adding such a small cache

(of size 128 bytes) had on the overall energy consumption of the system with a 512 bytes SPM, as can be seen in the beginning of the plot of figure 4.9, in which a reduction on energy consumption of around 20% was achieved in comparison to the version without cache.

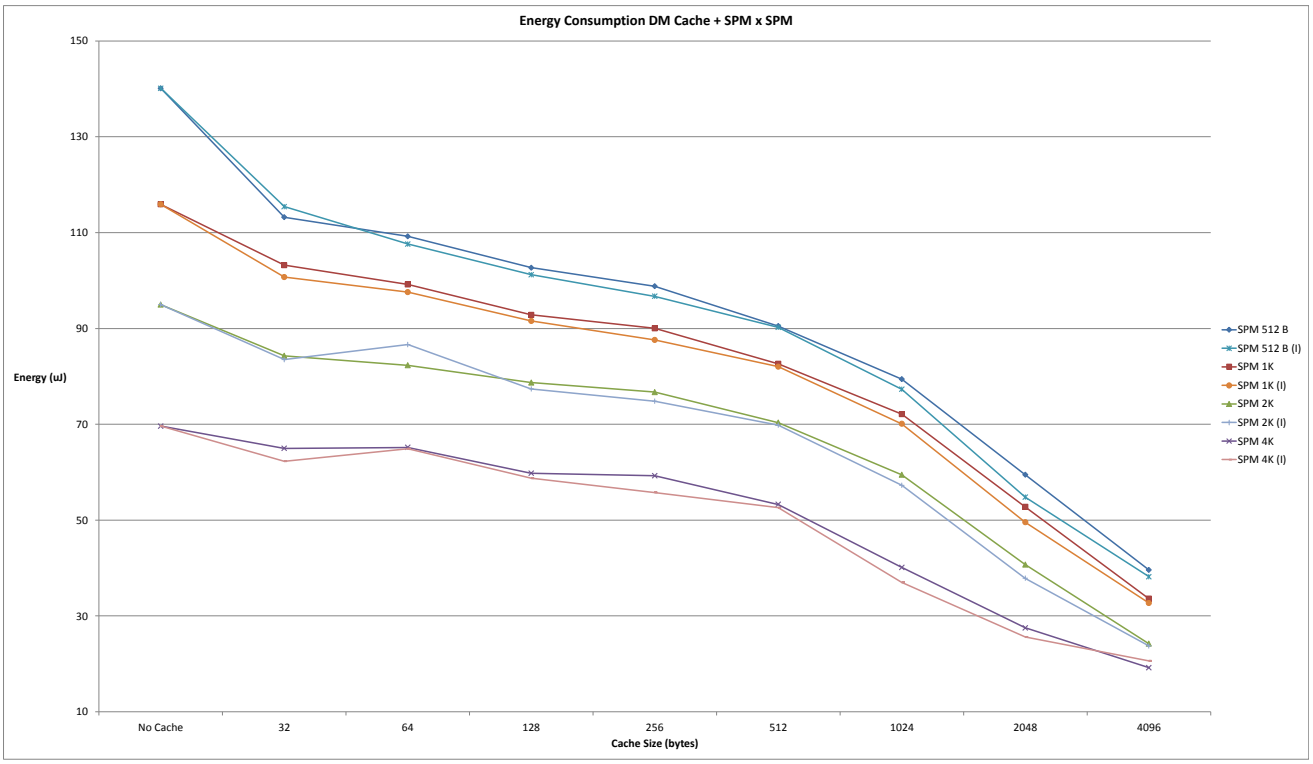

Figure 4.10: Energy consumption of SPM-only versus DM cache + SPM solutions.

4.4 DM cache-only versus SPM-only

The comparison among these two solutions can be summarized on figure 4.11. On this figure, SPM-fixed means the SPM which used the SRAM already present in the system, hence with a fixed energy per read value across the different sizes, while SPM-variable means a different energy per read for each size. It is easy to see that for DM caches smaller than 2K bytes, the energy consumption is smaller than that of the SPMs. However that is inverted for SPMs of sizes 2K and 4K bytes. That is an expected result, since as the SPM grows, more functions can be fit into the it, hence the number of accesses to the main memory decreases.

A very interesting conclusion can be drawn from the analysis of figure 4.12, which presents the hit rates comparison between SPM and DM caches (even though SPM do not have the concept of hit rates, here they were treated as the number of accesses made to the SPM memory). For SPM of size 2K bytes, although it presents lower hit rate in comparison to the

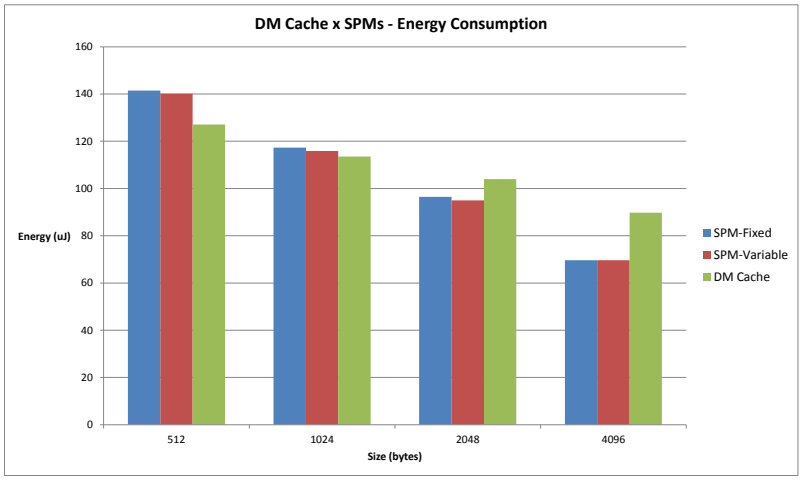

Figure 4.11: Energy consumption of DM caches and SPMs comparison.

DM cache, it still results in less energy consumption. That is due the fact that caches have misses, while SPM do not. Moreover, as mentioned before, as the size of the SPM increases, more functions (and bigger functions) fit into the SPM, boosting also the hit rate.

It is also interesting to note that there are no differences between the hit rates of SPM-fixed and SPM-variable, hence only SPM-fixed, simply named SPM, is present in figure 4.12. The small difference in the energy per read values of SPMs does not change the result of the optimization problem, that is, the set of functions mapped to the SPM.

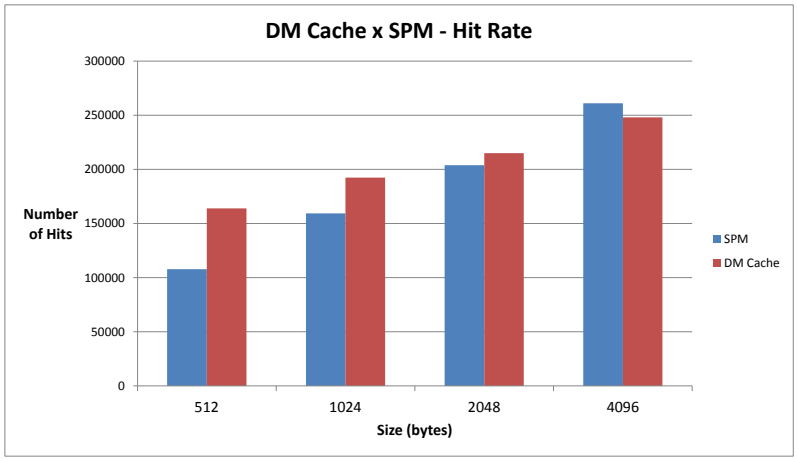

Figure 4.12: Hit rates of DM caches and SPM comparison.

Chapter 5

Discussion

In this chapter, a summary and analysis of the results from simulations are presented. Later, an evaluation of the tools and methods applied in this project is also provided.

5.1 Memory Systems

As became clear by the results from the previous section, adding either a DM cache or a SPM (or both) to a memory system composed of a Flash memory greatly increases its energy efficiency. This result corroborates with all previous works cited on section 2.

The indexing (or bit-selection) technique applied to the DM cache improved the energy efficiency for every cache size. Although this improvement was not particularly impressive, it still might be a good idea to employ it on an actual system. For example, for a cache size of 1312 bytes, the indexing version saves around 4% in comparison with the standard implementation. Furthermore, as previously stated, indexing is completely application-dependent. For example, in the experimental results provided by [18], the reduction of conflict misses for the Powerstone benchmark ranged from 0% to 100%, with an average of 19.2% for an instruction cache of size 1KB.

The SPM seems to be a very interesting solution. It sports a considerable amount of energy saving with basically zero overhead, given the fact the target microcontroller system already has a SRAM available to be used as scratchpad. Moreover, as expected, as the size of the SPM increases, more functions can be placed in the SPM, increasing the energy savings.

It is important to mention the fact that using a fixed size 4K SRAM as the SPM, instead of SRAMs of different sizes, did not affect expressively the results in energy savings.

The best solution regarding energy savings is certainly the combination of both DM cache and SPM in the same memory system.

When it comes to each individual solution, the decision about whether to add just a DM cache or use SPM (with fixed energy per read value) clearly depends on the size of these memories. For DM cache and SPM of size 512 bytes, the DM cache is around 11% more energy efficient than the SPM. However, adding a DM cache implies in area overhead in the chip, while SPM do not.

5.2 Tools and Benchmark

As explained before, the evaluations performed in this project targeted an actual microcontroller system. Not only the selection of the architectures to be evaluated, as described in section 3.1, were made with this system in mind, but also it shaped the way simulations were performed. For example, instead of using generic benchmarks to evaluate the solutions proposed, the software that actually runs in this system was used to perform the simulations. This was particularly interesting because both architectures evaluated are application-dependent, therefore the results of simulations are even more relevant.

The drawback is obvious: only one application was evaluated, thus the solutions were not thoroughly put to test. It is possible that, for a different application, the DM cache can hold better hit rates, becoming a more interesting solution at different sizes.

In terms of tool, the simulation tool written in Python proved to be very helpful to speed up the simulations, while providing a simple output in terms of results.

Chapter 6

Conclusions and future work

This project, as described in section 1.1, was decomposed in different tasks. An account of each of them is provided in the following paragraphs.

Task 1: Realize a broad literature research on the field of cache architectures for energy consumption reduction.

As can be seen on section 2, this task was fully accomplished. More than 20 papers were researched, resulting in a sound foundation for future work in different cache solutions for energy efficient memory systems. As predicted, not all ideas proposed in the papers could be applied to this project. Some of the solutions focused on circuit techniques, which demand a different evaluation approach. This is also the case with associative memories.

Tiny Caches seem to be an interesting approach, but it was discharged because it needs access to the internal signals of the processor to be fully implemented. Way-prediction requires extra micro-architecture structures which are not present in the microprocessor used in the target system.

Task 2: Select two or more memory system architectures for evaluation through simulation.

This task was also fully accomplished, as exposed in section 3.1. As a result of this task, an optimal bit-selection algorithm was implemented in C++. It makes used of a BDD and ADD package called CUDD, available on [25].

This task comprises not only critically choosing which solutions should be evaluated, but also implementing any needed algorithm or tool for the given architectures.

Task 3: Perform high-level simulations with the selected architectures and an actual application.

The accomplishment of this task is thoroughly described in chapters 3 and 4.

In more general terms, this project achieved its goal of evaluating different caches architectures for a given microcontroller system. The basic conclusion to be drawn is that it is highly advantageous to add a small SRAM to the memory system. There is a clear trade-off between area overhead versus energy efficiency when it comes to both solutions evaluated.

6.1 Future Work

This project gives room to many interesting possibilities. In one hand, as mentioned before, the literature research presented various ideas that can be used in different contexts. On the other hand, in the scope of this project, at least another architecture could be modeled in the high-level simulation environment for comparison with the already implemented ones. A good candidate would be the way-halting cache architecture, presented in section 2.2.4. The idea presented in [32] describes a 4-way halting cache, while for this project a possible solution could be a 2-way halting cache.

As explained before, for this project only one application was simulated. Therefore, a further improvement of this project would be to perform the simulations with different applications. That would improve the quality of the comparisons and assessments of the solutions modeled.

Yet another step for this project would be to have the solutions here presented modeled in a Hardware Description Language (HDL), such as Verilog, to validate the findings obtained.

Bibliography

[1] Federico Angiolini, Francesco Menichelli, Alberto Ferrero, Luca Benini, and Mauro Olivieri. A post-compiler approach to scratchpad mapping of code. In *Proceedings of the 2004 international conference on Compilers, architecture, and synthesis for embedded systems*, CASES '04, pages 259–267, New York, NY, USA, 2004. ACM.

[2] A. Efthymiou and J.D. Garside. A cam with mixed serial-parallel comparison for use in low energy caches. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 12(3):325 –329, march 2004.

[3] K. Flautner, Nam Sung Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. In *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*, pages 148 –157, 2002.

[4] Stian Fredrikstad. Saving energy in periodic embedded systems with memory system techniques. Master's thesis, Norwegian University of Science and Technology, June 2012.

[5] Tony Givargis. Zero cost indexing for improved processor cache performance. *ACM Trans. Des. Autom. Electron. Syst.*, 11(1):3–25, January 2006.

[6] Ann Gordon-Ross, Susan Cotterell, and Frank Vahid. Tiny instruction caches for low power embedded systems. *ACM Trans. Embed. Comput. Syst.*, 2(4):449–481, November 2003.

[7] Koji Inoue, Vasily Moshnyaga, and Kazuaki Murakami. Dynamic tag-check omission: A low power instruction cache architecture exploiting execution footprints. In Babak Falsafi and T.N. Vijaykumar, editors, *Power-Aware Computer Systems*, volume 2325 of *Lecture Notes in Computer Science*, pages 18–32. Springer Berlin Heidelberg, 2003.

[8] Andhi Janapsatya, Aleksandar Ignjatović, and Sri Parameswaran. A novel instruction scratchpad memory optimization method based on concomitance metric. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, ASP-DAC '06, pages 612–617, Piscataway, NJ, USA, 2006. IEEE Press.

[9] Nam Sung Kim, K. Flautner, D. Blaauw, and T. Mudge. Drowsy instruction caches. leakage power reduction using dynamic voltage scaling and cache sub-

bank prediction. In *Microarchitecture, 2002. (MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on*, pages 219 – 230, 2002.

[10] Nam Sung Kim, K. Flautner, D. Blaauw, and T. Mudge. Circuit and microarchitectural techniques for reducing cache leakage power. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 12(2):167 –184, feb. 2004.

[11] Soontae Kim, N. Vijaykrishnan, Mahmut Kandemir, Anand Sivasubramaniam, and Mary Jane Irwin. Partitioned instruction cache architecture for energy efficiency. *ACM Trans. Embed. Comput. Syst.*, 2(2):163–185, May 2003.

[12] HP Labs. CACTI. `http://www.hpl.hp.com/research/cacti/`, 2012. [Online; accessed 13-November-2012].

[13] ARM Ltd. ARM Artisan Physical IP Solutions. `http://www.arm.com/products/physical-ip/index.php`, 2012. [Online; accessed 13-November-2012].

[14] Xin Lu and Yuzhuo Fu. Reducing leakage power in instruction cache using wdc for embedded processors. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, ASP-DAC '05, pages 1292–1295, New York, NY, USA, 2005. ACM.

[15] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[16] Ramesh Panwar and David Rennels. Reducing the frequency of tag compares for low power i-cache design. In *Proceedings of the 1995 international symposium on Low power design*, ISLPED '95, pages 57–62, New York, NY, USA, 1995. ACM.

[17] Jongsoo Park, James Balfour, and William James Dally. Fine-grain dynamic instruction placement for l0 scratch-pad memory. In *Proceedings of the 2010 international conference on Compilers, architectures and synthesis for embedded systems*, CASES '10, pages 137–146, New York, NY, USA, 2010. ACM.

[18] K. Patel, E. Macii, L. Benini, and M. Poncino. Reducing cache misses by application-specific re-configurable indexing. In *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, pages 125 – 130, nov. 2004.

[19] K. Patel, E. Macii, L. Benini, and M. Poncino. Reducing cache misses by application-specific re-configurable indexing. In *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, pages 125 – 130, nov. 2004.

[20] David A. Patterson and John L. Hennessy. *Computer Organization and Design - The Hardware / Software Interface (Revised 4th Edition)*. The Morgan Kaufmann Series in Computer Architecture and Design. Academic Press, 2012.

[21] David Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45:758–767, 1994.

[22] Michael D. Powell, Amit Agarwal, T. N. Vijaykumar, Babak Falsafi, and Kaushik Roy. Reducing set-associative cache energy via way-prediction and selective direct-mapping. In *Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture*, MICRO 34, pages 54–65, Washington, DC, USA, 2001. IEEE Computer Society.

[23] Jan Rabaey. *Low Power Design Essentials*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[24] Marisha Rawlins and Ann Gordon-Ross. Lightweight runtime control flow analysis for adaptive loop caching. In *Proceedings of the 20th symposium on Great lakes symposium on VLSI*, GLSVLSI '10, pages 239–244, New York, NY, USA, 2010. ACM.

[25] Fabio Somenzi. CUDD package. `http://vlsi.colorado.edu/~fabio/CUDD/`, 2012. [Online; accessed 13-November-2012].

[26] Guido van Rossum. Python Official Website. `http://python.org/`, 2012. [Online; accessed 13-November-2012].

[27] H. Vandierendonck and K. De Bosschere. Xor-based hash functions. *Computers, IEEE Transactions on*, 54(7):800 – 812, july 2005.

[28] H. Vandierendonck, P. Manet, and J.-D. Legat. Application-specific reconfigurable xor-indexing to eliminate cache conflict misses. In *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, volume 1, pages 1 –6, march 2006.

[29] D.P. Volpato, A.K.I. Mendonca, L.C.V. dos Santos, and J.L. GuÌ andntzel. A post-compiling approach that exploits code granularity in scratchpads to improve energy efficiency. In *VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on*, pages 127 –132, july 2010.

[30] Chuanjun Zhang. An efficient direct mapped instruction cache for application-specific embedded systems. In *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '05, pages 45–50, New York, NY, USA, 2005. ACM.

[31] Chuanjun Zhang. A low power highly associative cache for embedded systems. In *Computer Design, 2006. ICCD 2006. International Conference on*, pages 31 –36, oct. 2006.

[32] Chuanjun Zhang, Frank Vahid, Jun Yang, and Walid Najjar. A way-halting cache for low-energy high-performance systems. *ACM Trans. Archit. Code Optim.*, 2(1):34–54, March 2005.

[33] M. Zhang and K. Asanovic. Highly-associative caches for low-power processors. In *Kool Chips Workshop, MICRO*, volume 33, 2000.

Appendix A

Bit-selection optimal algorithm

```
#include "cuddObj.hh"
#include <math.h>
#include <iostream>
#include <fstream>
#include <cassert>
#include <sstream>
#include <bitset>
#include <map>
#include <stdio.h>
#include <limits>


using namespace std;

int const NUMBER_OF_ADDRESS_BITS = 18;
int const NUMBER_OF_INDEX_BITS = 5;
int const MAX_NUMBER_OF_INDEX_BITS = 12;

// New type definition
typedef std::map<int, int> MemoryDict;
typedef std::map<std::pair<int, int>, int> PairDict;
typedef MemoryDict::const_iterator It;

/*---------------------------------------------------------------------------*/
/* Variable declarations                                                     */
/*---------------------------------------------------------------------------*/

#ifndef lint
static char rcsid[] UNUSED = "$Id: testobj.cc,v 1.7 2012/02/05 01:06:40 fabio Exp fabio $";
#endif

/*---------------------------------------------------------------------------*/
/* Static function prototypes                                                */
/*---------------------------------------------------------------------------*/

static void testAdd(Cudd& mgr, int verbosity);


/*---------------------------------------------------------------------------*/
/* Definition of exported functions                                          */
/*---------------------------------------------------------------------------*/
```

```
/**Function********************************************************************

  Synopsis    [Main program for testobj.]

  Description []

  SideEffects [None]

  SeeAlso     []

******************************************************************************/
int
main(int argc, char **argv)
{
    int verbosity = 0;

    if (argc == 2) {
        int retval = sscanf(argv[1], "%d", &verbosity);
        if (retval != 1)
            return 1;
    } else if (argc != 1) {
        return 1;
    }

    Cudd mgr(0,2);
    cout << "wataheu\n";
    testAdd(mgr,6);
    return 0;

} // main


int ConvertStrToInt(string data){
    int x;
    std::stringstream str(data);
    str >> x;
    return x;

}

MemoryDict GetLastMemoryAccessedCache(vector<int> program_trace){

    MemoryDict last_memory_access;

    int number_of_mem_accesses = program_trace.size();
    int address_i;

    for (int i = 0; i < number_of_mem_accesses; i++){
        address_i = program_trace[i];
        last_memory_access[address_i] = i;
    }

    return last_memory_access;

}


bitset<NUMBER_OF_ADDRESS_BITS> CalculateDCP(int address_i, int address_j){
//    cout << "\nxoring " << address_i << " , " << address_j << "\n";
    bitset<NUMBER_OF_ADDRESS_BITS> address_i_bits(address_i);
    bitset<NUMBER_OF_ADDRESS_BITS> address_j_bits(address_j);
    bitset<NUMBER_OF_ADDRESS_BITS> xor_addresses = address_i_bits ^ address_j_bits;
//    cout << xor_addresses;
```

```
    return xor_addresses;

}

void CalculateCP(bitset<NUMBER_OF_ADDRESS_BITS> DCP, vector<bitset<NUMBER_OF_ADDRESS_BITS> > &CP){

    // if the vector is empty, just add it the DCP
/*    cout << "\nCP ADDRESS " << &CP;
    cout << "\nSize " << CP.size();*/
    int size = CP.size();
    if (CP.empty()){
        CP.push_back(DCP);
        return;
    }
    bitset<NUMBER_OF_ADDRESS_BITS> temp;

    for (int i = 0; i < size; i++){
        // make a AND with all elements from CP, one at a time
//        cout << "\n\nDCP " << DCP.to_ulong() << " CP[" << i << "] " << CP[i].to_ulong() << "\n";
        temp = DCP & CP[i];
//        cout << "temp " << temp.to_ulong() << "\n";
        if (temp == DCP){
            CP[i] = temp;
            return;
        }
        else if (temp == CP[i])
            return;
        else{
            CP.push_back(DCP);
            return;
        }
    }

    return;

}


vector<bitset<NUMBER_OF_ADDRESS_BITS> > GeneratePatterns(int number_of_index_bits){

  cout << " GeneratePatterns \n";
  int number_of_addresses = pow(2, NUMBER_OF_ADDRESS_BITS);

  cout << number_of_addresses << "\n";

  vector<bitset<NUMBER_OF_ADDRESS_BITS> > patterns;

  int count_number_of_ones = 0;
  for (int i = 0; i < number_of_addresses; i++){
      bitset<NUMBER_OF_ADDRESS_BITS> temp(i);

      count_number_of_ones = 0;
      for (int j = 0; j < NUMBER_OF_ADDRESS_BITS; j++){
          if (temp[j] == 1)
              count_number_of_ones++;
      }

      if (count_number_of_ones == number_of_index_bits){
          patterns.push_back(temp);
      }
  }

  return patterns;
```

```
}


void Evaluate(vector<bitset<NUMBER_OF_ADDRESS_BITS> > &all_patterns, int number_of_index_bits, ADD &add_result){

    cout << "\nStarting evaluation of cache size with " << number_of_index_bits << " indexes \n";

    int minimum, temp_value;
    minimum = std::numeric_limits<int>::max();

    ADD eval_add;

    DdNode* n;
    DdNode min;
    int inputs[NUMBER_OF_ADDRESS_BITS];
    int bit;

    for (int p = 0; p < all_patterns.size(); p++){

        cout << "\nEvaluating: ";
        for (int s = 0; s < NUMBER_OF_ADDRESS_BITS; s++){
            bit =  all_patterns[p][s];
            cout << bit;
            inputs[s] = bit;
        }


        cout << "\n";
        eval_add = add_result.Eval(inputs);
        eval_add.print(2, 6);

        n = eval_add.getNode();
        temp_value = n->type.value;
        if (temp_value < minimum)
            minimum = temp_value;
            min = *n;

    }

    // finding the minimum
    cout << "\nMinimum value: ";
    cout <<  minimum;
    cout << "\n";
    cout << n;
    cout << "\n";

    return;

}


static void
testAdd(
  Cudd& mgr,
  int verbosity)
{

    // initializing the ADD with background constant as plusInfinity, so we can
    // actually see the 0
    ADD pInf = mgr.plusInfinity();
    mgr.SetBackground(pInf);

    // creating the initial ADD vars and result
```

```
ADD result = mgr.constant(0);

vector<BDD> y(NUMBER_OF_ADDRESS_BITS);

// initialize the ADD
int i;
for (i = 0; i < NUMBER_OF_ADDRESS_BITS; i++) {
    y[i] = mgr.bddVar(i);
}

bitset<NUMBER_OF_ADDRESS_BITS> address_i_bits;
bitset<NUMBER_OF_ADDRESS_BITS> address_j_bits;
bitset<NUMBER_OF_ADDRESS_BITS> DCP;

// read from file and put it on an list of integers
std::string line;
int number;
ifstream myfile ("ant_tx.log.out");

int myints[] = {0, 1, 5, 6, 1, 5, 6, 7, 6};
vector<int> program_trace; // (myints, myints + sizeof(myints) / sizeof(int) );

// read the trace file and puts in the variable program_trace
if (myfile.is_open())
{
  do {
    getline (myfile,line);
    number = ConvertStrToInt(line);
    program_trace.push_back(number);
  }
  while ( myfile.good() );
  myfile.close();
}


// cache the last index of each address in a dict like structure
// this is used to speed up the process of checking whether the current address
// being analysed is the last reference being made to it,
// if it is, then nothing should be done


MemoryDict last_memory_access;
last_memory_access = GetLastMemoryAccessedCache(program_trace);

// main body of the algorithm
// double loop to go over the addresses
bool index_found;
int number_of_mem_accesses = program_trace.size();
int address_i, address_j;
int last_access_of_address_i;

pair<int, int> pair_of_addresses;

BDD minterm, temp;

int pair_found;

for (int i = 0; i < number_of_mem_accesses; i++){

    PairDict edge_cache;
    vector<bitset<NUMBER_OF_ADDRESS_BITS> > CP;
    vector<BDD> CP_BDD;
```

```
        // used to cache pairs in the same edge (address_i -> address_j)

        address_i = program_trace[i];
        last_access_of_address_i = last_memory_access[address_i];

        index_found = last_access_of_address_i > i;

        // this means that there will be a reference to address_i in the future
        if (index_found){

            for (int j = i + 1; j < number_of_mem_accesses; j++){
                address_j = program_trace[j];

                // if the addresses match, we found an edge, that is, a list
                // of addresses for which the conflicts should be analysed
                if (address_i == address_j){

                    // add CP to ADD
                    temp = mgr.bddZero();

                    for (int each_cp = 0; each_cp < CP.size(); each_cp++){

                        minterm = mgr.bddOne();
                        for (int bit_index = 0; bit_index < NUMBER_OF_ADDRESS_BITS; bit_index++){
                            if (CP[each_cp][bit_index] == 1)
                                minterm *= ~y[bit_index];
                        }
                        temp += minterm;

                    }
                    result += temp.Add();

                    break;
                }


                // i will cache the expression (ADD) for each pair of addresses
                // because the same pair generates the same boolean expression
                // which when "ORed" with same pair, will be simplified
                // therefore there is no need to add it to the ADD
                if (address_i > address_j)
                    pair_of_addresses = make_pair(address_j, address_i);
                else
                    pair_of_addresses = make_pair(address_i, address_j);

                pair_found = edge_cache.count(pair_of_addresses);

                // if pair not found, then add expression to the ADD and add it to
                // the cache, otherwise, do nothing =)
                if (!pair_found){
                    edge_cache[pair_of_addresses] = 1;

                    DCP = CalculateDCP(address_i, address_j);
                    CalculateCP(DCP, CP);
                }
            }
        }
    }

    cout << "\nresult", result.print(2,verbosity);
    cout << "\n\n";

    for (int n = NUMBER_OF_INDEX_BITS; n <= MAX_NUMBER_OF_INDEX_BITS; n++){
```

```
vector<bitset<NUMBER_OF_ADDRESS_BITS> > patterns = GeneratePatterns(n);
Evaluate(patterns, n, result);

    }


//    vector<bitset<NUMBER_OF_ADDRESS_BITS> > all_patterns_2k = GeneratePatterns(NUMBER_OF_INDEX_BITS + 1);
//    vector<bitset<NUMBER_OF_ADDRESS_BITS> > all_patterns_4k = GeneratePatterns(NUMBER_OF_INDEX_BITS + 2);
//
//    Evaluate(all_patterns_2k, NUMBER_OF_INDEX_BITS + 1, result);
//    Evaluate(all_patterns_4k, NUMBER_OF_INDEX_BITS + 2, result);

} // testAdd
```

Appendix B

Simulation Environment

```
#============================================================================
# General Simulation Parameters
#============================================================================
MAIN_MEMORY_SIZE = 262144 #bytes
FLASH_ENERGY_PER_READ = 0.1593
SPM_ENERGY_PER_READ = 0.0218


#============================================================================
# Specific Simulation Parameters
#============================================================================
class DMCache_32Words:
    energy_per_read = 0.3
    energy_per_write = 0.3
    cache_size = 176 #bytes
    mapping = "000010000000111100"
    trace_file = "trace.log"


class DMCache_64Words:
    energy_per_read = 0.3
    energy_per_write = 0.3
    cache_size = 344 #bytes
    mapping = "000000010010111100"
    trace_file = "trace.log"


class DMCache_128Words:
    energy_per_read = 0.3
    energy_per_write = 0.3
    cache_size = 672 #bytes
    mapping = "000000100011111100"
    trace_file = "trace.log"


class DMCache_256Words:
    energy_per_read = 0.3
    energy_per_write = 0.3
    cache_size = 1312 #bytes
    mapping = "000000100111111100"
    trace_file = "trace.log"
```

```
class DMCache_512Words:
    energy_per_read = 0.3
    energy_per_write = 0.3
    cache_size = 2560 #bytes
    mapping = "000100110011111100"
    trace_file = "trace.log"


class DMCache_1024Words:
    energy_per_read = 0.3
    energy_per_write = 0.3
    cache_size = 4998 #bytes
    mapping = "001000011111111100"
    trace_file = "trace.log"


class DMCache_2048Words:
    energy_per_read = 0.3
    energy_per_write = 0.3
    cache_size = 9728 #bytes
    mapping = "001001011111111100"
    trace_file = "trace.log"


class DMCache_4096Words:
    energy_per_read = 0.3
    energy_per_write = 0.3
    cache_size = 9728 #bytes
    mapping = "000101111111111100"
    trace_file = "trace.log"
```

```
'''
Created on Nov 15, 2012

@author: vcarlos
'''


import inspect
import scenarios

for name, obj in inspect.getmembers(scenarios):
    if inspect.isclass(obj):
        if name.startswith("Factory"):
            if obj.build:
                print "----------------------", name, "-------------------------"
#                print name
                sim = obj.Build()
                sim.Run()
                print sim
                print


if __name__ == '__main__':
    pass
```

```
'''
Created on Nov 3, 2012

@author: vcarlos
'''

class Simulation(object):
    '''
    Runs the simulation of the system
    '''


    def __init__(self, memory_system, trace_file):
        '''
        Constructor
        '''
        self.memory_system = memory_system
        self.trace_file_name = trace_file
        self.program_trace = self._ReadProgramTrace()


    def _ReadProgramTrace(self, ):
        f = open(self.trace_file_name, "r")
        program_trace = []
        line  = f.readline()
        while line != "":
            l = int(line, 16)
            program_trace.append(l)
            line = f.readline()

        f.close()

        return program_trace


    def Run(self):

        for each_address in self.program_trace:
            self.memory_system.Read(each_address)


    def __str__(self, *args, **kwargs):
        result = "------------------------ Simulation Results -------------------------"
        result += "\nProgram size: " + str(len(self.program_trace))
        result += str(self.memory_system)
        result += "\n"
        return result
```

```
'''
Created on Nov 3, 2012

@author: vcarlos
'''
from sim import Simulation
from mem.flash import FlashMemory
from mem.memory_system import FlashMemorySystem, SPMMemorySystem, \
    CachedMemorySystem, SPMCacheMemorySystem
from mem.cache import DMCache, XORDMCache
from mem.spm_builder import SPMBuilderWithCache, SPMBuilder
from param import sim1, sim2, sim3

#==============================================================================
# General Simulation Parameters
#==============================================================================
MAIN_MEMORY_SIZE = 262144  # bytes
FLASH_ENERGY_PER_READ = 0.500
SPM_ENERGY_PER_READ = 0.0312

#==============================================================================
# Current Simulation Parameters
#==============================================================================
current_sim = sim1
trace_file = current_sim.trace_file
analysis_file = current_sim.analysis_file


# LIST_OF_FUNCTIONS_WORK_SHEET = "/home/vcarlos/workspace/simulation/input/list_of_functions.xls"

#==============================================================================
# Cache Factories
#==============================================================================

#==============================================================================
# Baselines
#==============================================================================
class CacheFactory32WordsBaseline():

    @staticmethod
    def Build():
        energy_per_read = 0.0174
        energy_per_write = 0.0186
        cache_size = 176  # bytes
        mapping = "00000000001111100"

        return DMCache(MAIN_MEMORY_SIZE, cache_size, mapping, energy_per_read, energy_per_write)


class CacheFactory64WordsBaseline():

    @staticmethod
    def Build():
        energy_per_read = 0.0264
        energy_per_write = 0.0276
        cache_size = 344  # bytes
        mapping = "00000000011111100"

        return DMCache(MAIN_MEMORY_SIZE, cache_size, mapping, energy_per_read, energy_per_write)


class CacheFactory128WordsBaseline():
```

```
    @staticmethod
    def Build():
        energy_per_read = 0.0276
        energy_per_write = 0.0294
        cache_size = 672  # bytes
        mapping = "000000000111111100"

        return DMCache(MAIN_MEMORY_SIZE, cache_size, mapping, energy_per_read, energy_per_write)


class CacheFactory256WordsBaseline():

    @staticmethod
    def Build():
        energy_per_read = 0.03
        energy_per_write = 0.0324
        cache_size = 1312  # bytes
        mapping = "000000001111111100"

        return DMCache(MAIN_MEMORY_SIZE, cache_size, mapping, energy_per_read, energy_per_write)


class CacheFactory512WordsBaseline():

    @staticmethod
    def Build():
        energy_per_read = 0.0342
        energy_per_write = 0.0378
        cache_size = 2560  # bytes
        mapping = "000000011111111100"

        return DMCache(MAIN_MEMORY_SIZE, cache_size, mapping, energy_per_read, energy_per_write)


class CacheFactory1024WordsBaseline():

    @staticmethod
    def Build():
        energy_per_read = 0.0402
        energy_per_write = 0.0468
        cache_size = 4998  # bytes
        mapping = "000000111111111100"

        return DMCache(MAIN_MEMORY_SIZE, cache_size, mapping, energy_per_read, energy_per_write)


class CacheFactory2048WordsBaseline():

    @staticmethod
    def Build():
        energy_per_read = 0.0433
        energy_per_write = 0.0496
        cache_size = 9728  # bytes
        mapping = "000001111111111100"

        return DMCache(MAIN_MEMORY_SIZE, cache_size, mapping, energy_per_read, energy_per_write)


class CacheFactory4096WordsBaseline():

    @staticmethod
    def Build():
        energy_per_read = 0.0473
```

```
        energy_per_write = 0.0545
        cache_size = 18944  # bytes
        mapping = "000011111111111100"

        return DMCache(MAIN_MEMORY_SIZE, cache_size, mapping, energy_per_read, energy_per_write)


#==============================================================================
# Indexing
#==============================================================================
class CacheFactory32Words():

    @staticmethod
    def Build():
        energy_per_read = 0.0174
        energy_per_write = 0.0186
        cache_size = 176  # bytes
        mapping = current_sim.mapping["CacheFactory32Words"]

        return DMCache(MAIN_MEMORY_SIZE, cache_size, mapping, energy_per_read, energy_per_write)



class CacheFactory64Words():

    @staticmethod
    def Build():
        energy_per_read = 0.0264
        energy_per_write = 0.0276
        cache_size = 344  # bytes
        mapping = current_sim.mapping["CacheFactory64Words"]

        return DMCache(MAIN_MEMORY_SIZE, cache_size, mapping, energy_per_read, energy_per_write)



class CacheFactory128Words():

    @staticmethod
    def Build():
        energy_per_read = 0.0276
        energy_per_write = 0.0294
        cache_size = 672  # bytes
        mapping = current_sim.mapping["CacheFactory128Words"]

        return DMCache(MAIN_MEMORY_SIZE, cache_size, mapping, energy_per_read, energy_per_write)



class CacheFactory256Words():

    @staticmethod
    def Build():
        energy_per_read = 0.03
        energy_per_write = 0.0324
        cache_size = 1312  # bytes
        mapping = current_sim.mapping["CacheFactory256Words"]

        return DMCache(MAIN_MEMORY_SIZE, cache_size, mapping, energy_per_read, energy_per_write)



class CacheFactory512Words():
```

```
    @staticmethod
    def Build():
        energy_per_read = 0.0342
        energy_per_write = 0.0378
        cache_size = 2560  # bytes
        mapping = current_sim.mapping["CacheFactory512Words"]

        return DMCache(MAIN_MEMORY_SIZE, cache_size, mapping, energy_per_read, energy_per_write)


class CacheFactory1024Words():

    @staticmethod
    def Build():
        energy_per_read = 0.0402
        energy_per_write = 0.0468
        cache_size = 4998  # bytes
        mapping = current_sim.mapping["CacheFactory1024Words"]

        return DMCache(MAIN_MEMORY_SIZE, cache_size, mapping, energy_per_read, energy_per_write)


class CacheFactory2048Words():

    @staticmethod
    def Build():
        energy_per_read = 0.0433
        energy_per_write = 0.0496
        cache_size = 9728  # bytes
        mapping = current_sim.mapping["CacheFactory2048Words"]

        return DMCache(MAIN_MEMORY_SIZE, cache_size, mapping, energy_per_read, energy_per_write)


class CacheFactory4096Words():

    @staticmethod
    def Build():
        energy_per_read = 0.0473
        energy_per_write = 0.0545
        cache_size = 18944  # bytes
        mapping = current_sim.mapping["CacheFactory4096Words"]

        return DMCache(MAIN_MEMORY_SIZE, cache_size, mapping, energy_per_read, energy_per_write)


#==============================================================================
# XORing
#==============================================================================

class CacheFactory256WordsXOR():

    @staticmethod
    def Build():
        energy_per_read = 0.0174
        energy_per_write = 0.0174
        cache_size = 1312  # bytes

        return XORDMCache(MAIN_MEMORY_SIZE, cache_size, "", energy_per_read, energy_per_write)
```

```
#==============================================================================
# Memory System Factories
#==============================================================================
class FactoryFlashMemoryBaselineMemorySystem():

    build = current_sim.run['FactoryFlashMemoryBaselineMemorySystem']

    @staticmethod
    def Build():

        trace_file = current_sim.trace_file
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        memory_system = FlashMemorySystem(flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


#==============================================================================
# SPM 512B Scenarios
#==============================================================================
class FactorySPM512MemorySystem():

    build = current_sim.run['FactorySPM512MemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 512  # bytes

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        spm = SPMBuilder(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size)

        memory_system = SPMMemorySystem(spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM512WithDMCache256WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM512WithDMCache256WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 512  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache256WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory256WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim
```

```
class FactorySPM512WithDMCache256WordsMemorySystem():

    build = current_sim.run['FactorySPM512WithDMCache256WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 512  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache256WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory256Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM512WithDMCache128WordsMemorySystem():

    build = current_sim.run['FactorySPM512WithDMCache128WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 512  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache128WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory128Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM512WithDMCache128WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM512WithDMCache128WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 512  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache128WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)
```

```
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory128WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM512WithDMCache32WordsMemorySystem():

    build = current_sim.run['FactorySPM512WithDMCache32WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 512  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache32WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory32Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM512WithDMCache32WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM512WithDMCache32WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 512  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache32WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory32WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM512WithDMCache64WordsMemorySystem():

    build = current_sim.run['FactorySPM512WithDMCache64WordsMemorySystem']

    @staticmethod
    def Build():
```

```
        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 512  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache64WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory64Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM512WithDMCache64WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM512WithDMCache64WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 512  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache64WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory64WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM512WithDMCache512WordsMemorySystem():

    build = current_sim.run['FactorySPM512WithDMCache512WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 512  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache512WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory512Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim
```

```
class FactorySPM512WithDMCache512WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM512WithDMCache512WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 512  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache512WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory512WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM512WithDMCache1024WordsMemorySystem():

    build = current_sim.run['FactorySPM512WithDMCache1024WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 512  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache1024WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory1024Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM512WithDMCache1024WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM512WithDMCache1024WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 512  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache1024WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)
```

```
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory1024WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM512WithDMCache2048WordsMemorySystem():

    build = current_sim.run['FactorySPM512WithDMCache2048WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 512  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache2048WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory2048Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM512WithDMCache2048WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM512WithDMCache2048WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 512  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache2048WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory2048WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM512WithDMCache4096WordsMemorySystem():

    build = current_sim.run['FactorySPM512WithDMCache4096WordsMemorySystem']

    @staticmethod
    def Build():
```

```
        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 512  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache4096WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory4096Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM512WithDMCache4096WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM512WithDMCache4096WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 512  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache4096WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory4096WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



#==============================================================================
# SPM 1K Scenarios
#==============================================================================
class FactorySPM1kMemorySystem():

    build = current_sim.run['FactorySPM1kMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 1024  # bytes

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        spm = SPMBuilder(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size)

        memory_system = SPMMemorySystem(spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim
```

```
class FactorySPM1kWithDMCache256WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM1kWithDMCache256WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 1024  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache256WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory256WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM1kWithDMCache256WordsMemorySystem():

    build = current_sim.run['FactorySPM1kWithDMCache256WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 1024  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache256WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory256Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM1kWithDMCache128WordsMemorySystem():

    build = current_sim.run['FactorySPM1kWithDMCache128WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 1024  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache128WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)
```

```
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory128Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM1kWithDMCache128WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM1kWithDMCache128WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 1024  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache128WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory128WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM1kWithDMCache32WordsMemorySystem():

    build = current_sim.run['FactorySPM1kWithDMCache32WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 1024  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache32WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory32Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM1kWithDMCache32WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM1kWithDMCache32WordsBaselineMemorySystem']

    @staticmethod
    def Build():
```

```
        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 1024  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache32WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory32WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM1kWithDMCache64WordsMemorySystem():

    build = current_sim.run['FactorySPM1kWithDMCache64WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 1024  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache64WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory64Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM1kWithDMCache64WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM1kWithDMCache64WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 1024  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache64WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory64WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim
```

```
class FactorySPM1kWithDMCache512WordsMemorySystem():

    build = current_sim.run['FactorySPM1kWithDMCache512WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 1024  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache512WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory512Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM1kWithDMCache512WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM1kWithDMCache512WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 1024  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache512WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory512WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM1kWithDMCache1024WordsMemorySystem():

    build = current_sim.run['FactorySPM1kWithDMCache1024WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 1024  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache1024WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)
```

```
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory1024Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM1kWithDMCache1024WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM1kWithDMCache1024WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 1024  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache1024WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory1024WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM1kWithDMCache2048WordsMemorySystem():

    build = current_sim.run['FactorySPM1kWithDMCache2048WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 1024  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache2048WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory2048Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM1kWithDMCache2048WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM1kWithDMCache2048WordsBaselineMemorySystem']

    @staticmethod
    def Build():
```

```
        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 1024  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache2048WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory2048WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM1kWithDMCache4096WordsMemorySystem():

    build = current_sim.run['FactorySPM1kWithDMCache4096WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 1024  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache4096WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory4096Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM1kWithDMCache4096WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM1kWithDMCache4096WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 1024  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache4096WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory4096WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim
```

```
#==============================================================================
# SPM 2k Scenarios
#==============================================================================
class FactorySPM2kMemorySystem():

    build = current_sim.run['FactorySPM2kMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 2048  # bytes

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        spm = SPMBuilder(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size)

        memory_system = SPMMemorySystem(spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM2kWithDMCache256WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM2kWithDMCache256WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 2048  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache256WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory256WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM2kWithDMCache256WordsMemorySystem():

    build = current_sim.run['FactorySPM2kWithDMCache256WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 2048  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache256WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
```

```
        dm_cache = CacheFactory256Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM2kWithDMCache128WordsMemorySystem():

    build = current_sim.run['FactorySPM2kWithDMCache128WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 2048  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache128WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory128Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM2kWithDMCache128WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM2kWithDMCache128WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 2048  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache128WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory128WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM2kWithDMCache32WordsMemorySystem():

    build = current_sim.run['FactorySPM2kWithDMCache32WordsMemorySystem']

    @staticmethod
    def Build():
```

```
        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 2048  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache32WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory32Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM2kWithDMCache32WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM2kWithDMCache32WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 2048  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache32WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory32WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM2kWithDMCache64WordsMemorySystem():

    build = current_sim.run['FactorySPM2kWithDMCache64WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 2048  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache64WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory64Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim
```

```
class FactorySPM2kWithDMCache64WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM2kWithDMCache64WordsBaselineMemorySystem']

    @staticmethod
    def Build():

        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 2048  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache64WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory64WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM2kWithDMCache512WordsMemorySystem():

    build = current_sim.run['FactorySPM2kWithDMCache512WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 2048  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache512WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory512Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM2kWithDMCache512WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM2kWithDMCache512WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 2048  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache512WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
```

```
        dm_cache = CacheFactory512WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM2kWithDMCache1024WordsMemorySystem():

    build = current_sim.run['FactorySPM2kWithDMCache1024WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 2048  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache1024WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory1024Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM2kWithDMCache1024WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM2kWithDMCache1024WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 2048  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache1024WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory1024WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM2kWithDMCache2048WordsMemorySystem():

    build = current_sim.run['FactorySPM2kWithDMCache2048WordsMemorySystem']

    @staticmethod
    def Build():
```

```
        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 2048  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache2048WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory2048Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM2kWithDMCache2048WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM2kWithDMCache2048WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 2048  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache2048WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory2048WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim




class FactorySPM2kWithDMCache4096WordsMemorySystem():

    build = current_sim.run['FactorySPM2kWithDMCache4096WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 2048  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache4096WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory4096Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim
```

```
class FactorySPM2kWithDMCache4096WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM2kWithDMCache4096WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 2048  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache4096WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory4096WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



#============================================================================
# SPM 4k Scenarios
#============================================================================
class FactorySPM4kMemorySystem():

    build = current_sim.run['FactorySPM4kMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 4096  # bytes

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        spm = SPMBuilder(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size)

        memory_system = SPMMemorySystem(spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM4kWithDMCache256WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM4kWithDMCache256WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 4096  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache256WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)
```

```
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory256WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM4kWithDMCache256WordsMemorySystem():

    build = current_sim.run['FactorySPM4kWithDMCache256WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 4096  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache256WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory256Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM4kWithDMCache128WordsMemorySystem():

    build = current_sim.run['FactorySPM4kWithDMCache128WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 4096  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache128WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory128Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM4kWithDMCache128WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM4kWithDMCache128WordsBaselineMemorySystem']

    @staticmethod
    def Build():
```

```
        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 4096  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache128WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory128WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM4kWithDMCache32WordsMemorySystem():

    build = current_sim.run['FactorySPM4kWithDMCache32WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 4096  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache32WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory32Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM4kWithDMCache32WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM4kWithDMCache32WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 4096  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache32WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory32WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim
```

```
class FactorySPM4kWithDMCache64WordsMemorySystem():

    build = current_sim.run['FactorySPM4kWithDMCache64WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 4096  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache64WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory64Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM4kWithDMCache64WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM4kWithDMCache64WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 4096  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache64WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory64WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM4kWithDMCache512WordsMemorySystem():

    build = current_sim.run['FactorySPM4kWithDMCache512WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 4096  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache512WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory512WordsBaseline.Build()
```

```
        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM4kWithDMCache1024WordsMemorySystem():

    build = current_sim.run['FactorySPM4kWithDMCache1024WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 4096  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache1024WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory1024Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM4kWithDMCache1024WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM4kWithDMCache1024WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 4096  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache1024WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory1024WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM4kWithDMCache2048WordsMemorySystem():

    build = current_sim.run['FactorySPM4kWithDMCache2048WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 4096  # bytes
```

```
        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache2048WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory2048Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM4kWithDMCache2048WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM4kWithDMCache2048WordsBaselineMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 4096  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache2048WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory2048WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim



class FactorySPM4kWithDMCache4096WordsMemorySystem():

    build = current_sim.run['FactorySPM4kWithDMCache4096WordsMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 4096  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache4096WordsMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory4096Words.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


class FactorySPM4kWithDMCache4096WordsBaselineMemorySystem():

    build = current_sim.run['FactorySPM4kWithDMCache4096WordsBaselineMemorySystem']
```

```
    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = 4096  # bytes

        # this is needed for pre-computation of best SPM mapping
        sim = FactoryDMCache4096WordsBaselineMemorySystem.Build()
        spm = SPMBuilderWithCache(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size, sim)

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        dm_cache = CacheFactory4096WordsBaseline.Build()

        memory_system = SPMCacheMemorySystem(dm_cache, spm, flash_memory)
        sim = Simulation(memory_system, trace_file)
        return sim


#================================================================================
# Control Scenario
#================================================================================
class FactorySPMFullSizeMemorySystem():
    '''
    This scenario assumes a SPM of the size of the full memory.
    Used to compare with the results from analysis.txt
    '''

    build = current_sim.run['FactorySPMFullSizeMemorySystem']

    @staticmethod
    def Build():


        spm_energy_per_read = SPM_ENERGY_PER_READ
        spm_size = MAIN_MEMORY_SIZE  # bytes

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        spm = SPMBuilder(analysis_file, FLASH_ENERGY_PER_READ, spm_energy_per_read, spm_size)

        memory_system = SPMMemorySystem(spm, flash_memory)
        sim = Simulation(memory_system, trace_file)

    #    for f in spm.status:
    #        print spm.status[f], " , ", f
    #    print
    #    for f in spm.mapping:
    #        print f.number_fetches, ",", f.name, ",", hex(f.starting_address + 1)
        return sim


#================================================================================
# Cache-only Scenarios
#================================================================================

class FactoryDMCache32WordsBaselineMemorySystem():

    build = current_sim.run['FactoryDMCache32WordsBaselineMemorySystem']

    @staticmethod
    def Build():
```

```
        dm_cache = CacheFactory32WordsBaseline.Build()
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        memory_system = CachedMemorySystem(dm_cache, flash_memory)

        sim = Simulation(memory_system, trace_file)
        return sim


class FactoryDMCache64WordsBaselineMemorySystem():

    build = current_sim.run['FactoryDMCache64WordsBaselineMemorySystem']

    @staticmethod
    def Build():



        dm_cache = CacheFactory64WordsBaseline.Build()
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        memory_system = CachedMemorySystem(dm_cache, flash_memory)

        sim = Simulation(memory_system, trace_file)
        return sim


class FactoryDMCache128WordsBaselineMemorySystem():

    build = current_sim.run['FactoryDMCache128WordsBaselineMemorySystem']

    @staticmethod
    def Build():



        dm_cache = CacheFactory128WordsBaseline.Build()
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        memory_system = CachedMemorySystem(dm_cache, flash_memory)

        sim = Simulation(memory_system, trace_file)
        return sim


class FactoryDMCache256WordsBaselineMemorySystem():

    build = current_sim.run['FactoryDMCache256WordsBaselineMemorySystem']

    @staticmethod
    def Build():



        dm_cache = CacheFactory256WordsBaseline.Build()
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        memory_system = CachedMemorySystem(dm_cache, flash_memory)

        sim = Simulation(memory_system, trace_file)
        return sim


class FactoryDMCache512WordsBaselineMemorySystem():

    build = current_sim.run['FactoryDMCache512WordsBaselineMemorySystem']
```

```
    @staticmethod
    def Build():



        dm_cache = CacheFactory512WordsBaseline.Build()
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        memory_system = CachedMemorySystem(dm_cache, flash_memory)

        sim = Simulation(memory_system, trace_file)
        return sim


class FactoryDMCache1024WordsBaselineMemorySystem():

    build = current_sim.run['FactoryDMCache1024WordsBaselineMemorySystem']

    @staticmethod
    def Build():



        dm_cache = CacheFactory1024WordsBaseline.Build()
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        memory_system = CachedMemorySystem(dm_cache, flash_memory)

        sim = Simulation(memory_system, trace_file)
        return sim


class FactoryDMCache2048WordsBaselineMemorySystem():

    build = current_sim.run['FactoryDMCache2048WordsBaselineMemorySystem']

    @staticmethod
    def Build():



        dm_cache = CacheFactory2048WordsBaseline.Build()
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        memory_system = CachedMemorySystem(dm_cache, flash_memory)

        sim = Simulation(memory_system, trace_file)
        return sim

class FactoryDMCache4096WordsBaselineMemorySystem():

    build = current_sim.run['FactoryDMCache4096WordsBaselineMemorySystem']

    @staticmethod
    def Build():



        dm_cache = CacheFactory4096WordsBaseline.Build()
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        memory_system = CachedMemorySystem(dm_cache, flash_memory)

        sim = Simulation(memory_system, trace_file)
        return sim
```

```
class FactoryDMCache32WordsMemorySystem():

    build = current_sim.run['FactoryDMCache32WordsMemorySystem']

    @staticmethod
    def Build():



        dm_cache = CacheFactory32Words.Build()
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        memory_system = CachedMemorySystem(dm_cache, flash_memory)

        sim = Simulation(memory_system, trace_file)
        return sim



class FactoryDMCache64WordsMemorySystem():

    build = current_sim.run['FactoryDMCache64WordsMemorySystem']

    @staticmethod
    def Build():



        dm_cache = CacheFactory64Words.Build()
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        memory_system = CachedMemorySystem(dm_cache, flash_memory)

        sim = Simulation(memory_system, trace_file)
        return sim


class FactoryDMCache128WordsMemorySystem():

    build = current_sim.run['FactoryDMCache128WordsMemorySystem']

    @staticmethod
    def Build():



        dm_cache = CacheFactory128Words.Build()
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        memory_system = CachedMemorySystem(dm_cache, flash_memory)

        sim = Simulation(memory_system, trace_file)
        return sim



class FactoryDMCache256WordsMemorySystem():

    build = current_sim.run['FactoryDMCache256WordsMemorySystem']

    @staticmethod
    def Build():

        dm_cache = CacheFactory256Words.Build()
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
```

```
        memory_system = CachedMemorySystem(dm_cache, flash_memory)

        sim = Simulation(memory_system, trace_file)
        return sim


class FactoryDMCache512WordsMemorySystem():

    build = current_sim.run['FactoryDMCache512WordsMemorySystem']

    @staticmethod
    def Build():

        dm_cache = CacheFactory512Words.Build()
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        memory_system = CachedMemorySystem(dm_cache, flash_memory)

        sim = Simulation(memory_system, trace_file)
        return sim


class FactoryDMCache1024WordsMemorySystem():

    build = current_sim.run['FactoryDMCache1024WordsMemorySystem']

    @staticmethod
    def Build():

        dm_cache = CacheFactory1024Words.Build()

        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        memory_system = CachedMemorySystem(dm_cache, flash_memory)

        sim = Simulation(memory_system, trace_file)
        return sim



class FactoryDMCache2048WordsMemorySystem():

    build = current_sim.run['FactoryDMCache2048WordsMemorySystem']

    @staticmethod
    def Build():

        dm_cache = CacheFactory2048Words.Build()
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        memory_system = CachedMemorySystem(dm_cache, flash_memory)

        sim = Simulation(memory_system, trace_file)
        return sim


class FactoryDMCache4096WordsMemorySystem():

    build = current_sim.run['FactoryDMCache4096WordsMemorySystem']

    @staticmethod
    def Build():

        dm_cache = CacheFactory4096Words.Build()
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        memory_system = CachedMemorySystem(dm_cache, flash_memory)
```

```
        sim = Simulation(memory_system, trace_file)
        return sim


class FactoryDMCache256WordsXORMemorySystem():

    build = current_sim.run['FactoryDMCache256WordsXORMemorySystem']

    @staticmethod
    def Build():



        dm_cache = CacheFactory256WordsXOR.Build()
        flash_memory = FlashMemory(MAIN_MEMORY_SIZE, FLASH_ENERGY_PER_READ)
        memory_system = CachedMemorySystem(dm_cache, flash_memory)

        sim = Simulation(memory_system, trace_file)
        return sim


#
# def BuildDMCache2kBaseline(trace_file):
#
#    CACHE_SIZE = 2048
#    MAPPING = "000000011111111100"
#    CACHE_ENERGY_PER_READ = 5
#    CACHE_ENERGY_PER_WRITE = 8
#
#    dm_cache = DMCache(MAIN_MEMORY_SIZE, CACHE_SIZE, MAPPING, CACHE_ENERGY_PER_READ, CACHE_ENERGY_PER_WRITE)
#    flash_memory = FlashMemory(FLASH_ENERGY_PER_READ)
#    memory_system = CachedMemorySystem(dm_cache, flash_memory)
#
#    sim = Simulation(memory_system, trace_file)
#    sim.Run()
#
#    print sim
#
#
#
# def BuildDMCache2k(trace_file):
#
#    CACHE_SIZE = 2048
#    MAPPING = "001001011111111100"
#    CACHE_ENERGY_PER_READ = 5
#    CACHE_ENERGY_PER_WRITE = 8
#
#    dm_cache = DMCache(MAIN_MEMORY_SIZE, CACHE_SIZE, MAPPING, CACHE_ENERGY_PER_READ, CACHE_ENERGY_PER_WRITE)
#    flash_memory = FlashMemory(FLASH_ENERGY_PER_READ)
#    memory_system = CachedMemorySystem(dm_cache, flash_memory)
#
#    sim = Simulation(memory_system, trace_file)
#    sim.Run()
#
#    print sim
#
# def BuildDMCache4kBaseline(trace_file):
#
#    CACHE_SIZE = 4096
#    MAPPING = "000000111111111100"
#    CACHE_ENERGY_PER_READ = 5
```

```
#    CACHE_ENERGY_PER_WRITE = 8
#
#    dm_cache = DMCache(MAIN_MEMORY_SIZE, CACHE_SIZE, MAPPING, CACHE_ENERGY_PER_READ, CACHE_ENERGY_PER_WRITE)
#    flash_memory = FlashMemory(FLASH_ENERGY_PER_READ)
#    memory_system = CachedMemorySystem(dm_cache, flash_memory)
#
#    sim = Simulation(memory_system, trace_file)
#    sim.Run()
#
#    print sim
#
#
#
# def BuildDMCache4k(trace_file):
#
#    CACHE_SIZE = 4096
#    MAPPING = "000101111111111100"
#    CACHE_ENERGY_PER_READ = 5
#    CACHE_ENERGY_PER_WRITE = 8
#
#    dm_cache = DMCache(MAIN_MEMORY_SIZE, CACHE_SIZE, MAPPING, CACHE_ENERGY_PER_READ, CACHE_ENERGY_PER_WRITE)
#    flash_memory = FlashMemory(FLASH_ENERGY_PER_READ)
#    memory_system = CachedMemorySystem(dm_cache, flash_memory)
#
#    sim = Simulation(memory_system, trace_file)
#    sim.Run()
#
#    print sim
#
#
# trace_file = "trace.log"

# get parameters for each scenario on module parameters

# BuildFlashMemoryBaseline(trace_file)
# BuildSPM1k(trace_file)
# BuildSPM4k(trace_file)
# BuildSPMFullSize(trace_file)


# BuildDMCache1kBaseline(trace_file)
# BuildDMCache1k(trace_file)
#
# BuildDMCache2kBaseline("trace.log")
# BuildDMCache2k("trace.log")
#
# BuildDMCache4kBaseline("trace.log")
# BuildDMCache4k("trace.log")



if __name__ == '__main__':
    pass
```

```
import abc


class Memory(object):

    def __init__(self, size, energy_per_read):
        self.energy_per_read = energy_per_read
        self.size = size


    @abc.abstractmethod
    def Read(self, address):
        return
```

```
'''
Created on Nov 3, 2012

@author: vcarlos
'''
import abc


#===============================================================================
# MemorySystem
#===============================================================================
class MemorySystem(object):
    '''
    classdocs
    '''

    def __init__(self):
        '''
        Constructor
        '''
        self.total_energy = 0


    @abc.abstractmethod
    def Read(self, address):
        return


#===============================================================================
# FlashMemorySystem
#===============================================================================
class FlashMemorySystem(MemorySystem):

    def __init__(self, flash_memory):
        super(FlashMemorySystem, self).__init__()
        self.flash = flash_memory

        self.flash_energy = 0


    def Read(self, address):

        self.flash_energy += self.flash.energy_per_read
        return 1


    def __str__(self, *args, **kwargs):
        result = "\nMemory System composed of"
        result += "\n"
        result += "\nFlash Memory"
        result += "\n"
        result += "Total Energy: " + str(self.flash_energy)

        return result


#===============================================================================
# CachedMemorySystem
#===============================================================================
class CachedMemorySystem(MemorySystem):


    def __init__(self, dm_cache, flash_memory):
```

```
        super(CachedMemorySystem, self).__init__()
        self.dm_cache = dm_cache
        self.flash = flash_memory
        self.miss_count = 0
        self.hit_count = 0

        self.cache_energy = 0
        self.flash_energy = 0


    def Read(self, address):

        hit = self.dm_cache.Read(address)
        self.cache_energy += self.dm_cache.energy_per_read
        if hit == 1:
            self.hit_count += 1
            return 1
        else:
            self.miss_count += 1
            self.cache_energy += self.dm_cache.energy_per_write
            self.flash_energy += self.flash.energy_per_read
            return 0


    def __str__(self, *args, **kwargs):
        result = "\nMemory System composed of"
        result += "\nCache:"
        result += "\n\tDM: " + str(self.dm_cache.size) + " bytes"
        result += "\n\t\tMapping: " + self.dm_cache.mapping
        result += "\n\t\tTotal Misses: " + str(self.miss_count)
        result += "\n\t\tTotal Hits: " + str(self.hit_count)
        result += "\n\t\tConflict Misses: " + str(self.dm_cache.GetTotalMissConflicts())
        result += "\n\t\tEnergy: " + str(self.cache_energy)
        result += "\n"
        result += "\nFlash Memory"
        result += "\n\tEnergy: " + str(self.flash_energy)
        result += "\n"
        result += "Total Energy: " + str(self.flash_energy + self.cache_energy)

#        result += "\n\n number of sets: " + str(len(self.dm_cache.data))

        return result


#===============================================================================
# SPMMemorySystem
#===============================================================================
class SPMMemorySystem(MemorySystem):

    def __init__(self, spm, flash_memory):
        super(SPMMemorySystem, self).__init__()
        self.spm = spm
        self.flash = flash_memory

        self.spm_energy = 0
        self.flash_energy = 0
        self.cfa = 0


    def Read(self, address):

        hit = self.spm.Read(address)
        if hit:
```

```
            self.spm_energy += self.spm.energy_per_read
            return 1
        else:
#            print "address going to flash: ", hex(address)
            self.cfa += 1
            self.flash_energy += self.flash.energy_per_read
            return 0


    def __str__(self, *args, **kwargs):
        result = "\nMemory System composed of"
        result += "\nSPM:"  + str(self.spm.size) + " bytes"
        result += "\n\tEnergy: " + str(self.spm_energy)
        result += "\n"
        result += "\nFlash Memory"
        result += "\n\tEnergy: " + str(self.flash_energy)
        result += "\n"
        result += "\nTotal Energy: " + str(self.flash_energy + self.spm_energy)

        return result



#=============================================================================
# CachedWithFunctionsMemorySystem
#=============================================================================
class CachedWithFunctionsMemorySystem(CachedMemorySystem):

    def __init__(self, dm_cache, flash_memory, list_of_functions):
        super(CachedWithFunctionsMemorySystem, self).__init__(dm_cache, flash_memory)

        self.mapping = list_of_functions
        self.total = 0


    def Read(self, address):
        '''
            "Read" from memory and return 1 if found, 0 otherwise.
        '''
        hit = super(CachedWithFunctionsMemorySystem, self).Read(address)
        self.total+=1

        for function in self.mapping:
            if address >= function.starting_address and address <=function.final_address:
                if hit == 1:
                    function.hit_count += 1
                else:
                    function.miss_count += 1
                break

        return hit


#=============================================================================
# SPMCacheMemorySystem
#=============================================================================
class SPMCacheMemorySystem(MemorySystem):

    def __init__(self, dm_cache, spm, flash_memory):
        super(SPMCacheMemorySystem, self).__init__()

        self.cached_ms = CachedMemorySystem(dm_cache, flash_memory)
        self.spm_ms = SPMMemorySystem(spm, flash_memory)
```

```
def Read(self, address):

    hit = self.spm_ms.Read(address)
    if not hit:
        hit = self.cached_ms.Read(address)

    return hit

def __str__(self, *args, **kwargs):
    result = "\nMemory System composed of"
    result += "\nCache:"
    result += "\n\tDM: " + str(self.cached_ms.dm_cache.size) + " bytes"
    result += "\n\t\tMapping: " + self.cached_ms.dm_cache.mapping
    result += "\n\t\tTotal Misses: " + str(self.cached_ms.miss_count)
    result += "\n\t\tTotal Hits: " + str(self.cached_ms.hit_count)
    result += "\n\t\tConflict Misses: " + str(self.cached_ms.dm_cache.GetTotalMissConflicts())
    result += "\n\t\tEnergy: " + str(self.cached_ms.cache_energy)
    result += "\n"
    result += "\n\tSPM:"  + str(self.spm_ms.spm.size) + " bytes"
    result += "\n\t\tEnergy: " + str(self.spm_ms.spm_energy)
    result += "\n"
    result += "\nFlash Memory"
    result += "\n\tEnergy: " + str(self.spm_ms.flash_energy)
    result += "\n"
    result += "\nTotal Energy: " + str(self.spm_ms.flash_energy + self.spm_ms.spm_energy + self.cached_ms.cache_energy)

    return result
```

```
'''
Created on Nov 3, 2012

@author: vcarlos
'''
from mem.memory import Memory

class SPM(Memory):
    '''
    In this abstraction of scratchpad, we will have mapping of all functions that are
    located in the scratchpad. So, instead of a contiguous address space, lets say, from 0x0000 to 0xFFFF,
    we will actually just have a mapping of FunctionN (address of instruction calling the function) to: yes,
    it is on the scratchpad, or no, it is not in the scratchpad.

    '''


    def __init__(self, size, energy_per_read):
        '''
        Constructor
        '''
        super(SPM, self).__init__(size, energy_per_read)
        # contain the function objects that are mapped to scratchpad
        self.mapping = []

        # auxiliary data structure to return the actual number of "hits" for each function
        # to be compared with the data on analysis.txt
        self.status = {}

        self.size = size



    def Read(self, address):
        '''
            "Read" from memory and return 1 if found, 0 otherwise.
        '''
        for function in self.mapping:
            if address >= function.starting_address and address <= function.final_address:
                if self.status.has_key(function.name):
                    count = self.status[function.name]
                    self.status[function.name] = count + 1
                else:
                    self.status[function.name] = 1

                return 1
        return 0
```

```
'''
Created on Nov 14, 2012

@author: vcarlos
'''
from ctypes import *
import xlrd
from spm import SPM
from mem.memory_system import CachedWithFunctionsMemorySystem


class Function(object):

    def __init__(self, times_function_called, number_fetches, function_name, starting_address, size):
        self.times_function_called = times_function_called
        self.number_fetches = number_fetches
        self.name = function_name
        self.starting_address = starting_address
        self.size = size
        final_address = starting_address + size
        self.final_address = final_address
        # see later where to put them
        self.hit_count = 0
        self.miss_count = 0


    def __str__(self, *args, **kwargs):
        res = "---------------Function------------------"
        res += "\nName: " + self.name
        res += "\nNumber of Fetches: " + str(self.number_fetches)
        res += "\nstarting_address: " + str(self.starting_address)
        res += "\nfinal_address: " + str(self.final_address)
        res += "\nsize: " + str(self.size)
        res += "\nhit_count: " + str(self.hit_count)
        res += "\nmiss_count: " + str(self.miss_count)
        res += "\n"

        return res


def CreateListOfFunction(excel_worksheet):

    book = xlrd.open_workbook(excel_worksheet)
    sheet = book.sheet_by_index(0)

    list_of_functions = []

    for row in xrange(sheet.nrows-1):
        row += 1
        times_function_called = int(sheet.cell_value(row, 0))
        number_fetches = int(sheet.cell_value(row, 1))
        function_name = sheet.cell_value(row, 2)
        starting_address = int(sheet.cell_value(row, 3), 16) - 1
        size = int(sheet.cell_value(row, 4))
        f = Function(times_function_called, number_fetches, function_name, starting_address, size)
        list_of_functions.append(f)

    return list_of_functions


def LoadMinknapSharedLibrary():
    cdll.LoadLibrary("/home/vcarlos/workspace/test/src/minknap.so")
    libc = CDLL("/home/vcarlos/workspace/test/src/minknap.so")
```

```
    return libc



def RunMinknap(list_of_functions, flash_energy_per_read, sram_energy_per_read, spm_size):
    libc = LoadMinknapSharedLibrary()
    number_of_functions = len(list_of_functions)

    # create the input/output arrays for the minknap function
    n = number_of_functions
    # x has the solution array (0 or 1)
    x = (c_uint*n)()
    # p has the profit function. #fetches * (flash - sram)
    p = (c_uint*n)()
    # w is the cost, in number of bytes
    w = (c_uint*n)()
    c = spm_size

    base_profit = flash_energy_per_read - sram_energy_per_read

    # fill in the arrays with data from functions objects
    for i, function in enumerate(list_of_functions):
#        print "int(round(function.number_fetches * base_profit))", int(round(function.number_fetches * base_profit))
        p[i] = int(round(function.number_fetches * base_profit))
        w[i] = function.size

    res = libc.minknap(n, p, w, x, c)

    return res, x


def SPMBuilder(excel_worksheet, flash_energy_per_read, sram_energy_per_read, spm_size):
    '''
    Constructs a SPM based on the trace analysis excel sheet.

    It makes used of the shared library minknap.so to determine which functions should be placed on
    the SPM.

    The end results is an SPM object, with the functions that are allocated to it.

    A function is an object of class Function.
    '''

    # first create list of functions based on the excel work sheet
    list_of_functions = CreateListOfFunction(excel_worksheet)
    res, solution = RunMinknap(list_of_functions, flash_energy_per_read, sram_energy_per_read, spm_size)

    #now I can create the SPM
    spm = SPM(spm_size, sram_energy_per_read)
    for i, sol in enumerate(solution):
        if sol == 1:
            func = list_of_functions[i]
#            print func
            spm.mapping.append(func)

    return spm


def RunMinknap2(list_of_functions, flash_energy_per_read, sram_energy_per_read, spm_size, cache_energy_per_read, cache_energy_pe
    libc = LoadMinknapSharedLibrary()
    number_of_functions = len(list_of_functions)

    # create the input/output arrays for the minknap function
```

```
    n = number_of_functions
    # x has the solution array (0 or 1)
    x = (c_uint*n)()
    # p has the profit function.
    p = (c_uint*n)()
    # w is the cost, in number of bytes
    w = (c_uint*n)()
    c = spm_size

    # here, spm.mapping has all functions
    # now, I need to run a system with a cache, having the knowledge of those functions to determine
    # the number of misses that happened on each function, so I can come with the correct profit value
    # for putting a particular function into the scratchpad
    # remember that, without a cache, the profit is all fetches * main_memory_energy - all_fetches * spm_energy
    # now, because there is a cache, not all_fetches go to MM, some of them go to cache, so the calculation is
    # number_hits * cache_energy + number_miss * (cache_energy*2 + mm_energy) - all_fetches * spm_energy

    hit_energy = cache_energy_per_read
    miss_energy = cache_energy_per_read + cache_energy_per_write + flash_energy_per_read

    # fill in the arrays with data from functions objects
    for i, function in enumerate(list_of_functions):
        energy_mm  = function.hit_count * hit_energy + function.miss_count * miss_energy
        energy_spm = function.number_fetches * sram_energy_per_read
#        print "int(round(energy_mm - energy_spm))", int(round(energy_mm - energy_spm))
        p[i] = int(round(energy_mm - energy_spm))
        w[i] = function.size

    res = libc.minknap(n, p, w, x, c)

    return res, x


def SPMBuilderWithCache(excel_worksheet, flash_energy_per_read, sram_energy_per_read, spm_size, simulation):
    '''
    Constructs a SPM based on the trace analysis excel sheet.

    It makes used of the shared library minknap.so to determine which functions should be placed on
    the SPM.

    The end results is an SPM object, with the functions that are allocated to it.

    A function is an object of class Function.
    '''

    # first create list of functions based on the excel worksheet
    list_of_functions = CreateListOfFunction(excel_worksheet)

    dm_cache = simulation.memory_system.dm_cache
    flash_memory = simulation.memory_system.flash
    simulation.memory_system = CachedWithFunctionsMemorySystem(dm_cache, flash_memory, list_of_functions)

    simulation.Run()

    cache_energy_per_read = dm_cache.energy_per_read
    cache_energy_per_write = dm_cache.energy_per_write

    res, solution = RunMinknap2(list_of_functions, flash_energy_per_read, sram_energy_per_read, spm_size, cache_energy_per_read,

    total = 0

    #now I can create the SPM
    spm = SPM(spm_size, sram_energy_per_read)
```

```
    for i, sol in enumerate(solution):
        func = list_of_functions[i]
        total += func.hit_count + func.miss_count
        if sol == 1:
#            print func
            spm.mapping.append(func)

    print "Total", total
    assert total == 381661
    return spm


#def Run():
#    from scenarios import FactoryDMCache128Words
#    sim = FactoryDMCache128Words.Build()
#    SPMBuilderWithCache("/home/vcarlos/workspace/simulation/input/list_of_functions.xls", 0.1593, 0.0593, 4096, sim)
#
#
#Run()


#============================================================================
# main
#============================================================================
if __name__ == '__main__':
    pass
```

```
from mem.memory import Memory


class FlashMemory(Memory):

    def __init__(self, size, energy_per_read):
        super(FlashMemory, self).__init__(size, energy_per_read)
        self.energy_per_read = energy_per_read
        self.size = size


    def Read(self, address):
        '''
        Instruction is always in the given address.
        '''
        return 1
```

```
'''
Created on Nov 2, 2012

@author: vcarlos
'''
import math
from mem.memory import Memory


class Cache(Memory):
    '''
    Implements a cache behaviour
    '''


    def __init__(self, size, main_memory_size, energy_per_read, energy_per_write):
        '''
        Constructor
        '''
        super(Cache, self).__init__(size, energy_per_read)
        self.size = size
        self.main_memory_size = main_memory_size
        self.number_of_address_bits = int(math.log(main_memory_size, 2))
        self.energy_per_read = energy_per_read
        self.energy_per_write = energy_per_write

        self.data = {}


class DMCache(Cache):

    def __init__(self, main_memory_size, size, mapping, energy_per_read, energy_per_write):
        super(DMCache, self).__init__(size, main_memory_size, energy_per_read, energy_per_write)
        self.mapping = mapping

        self._str_format = '{0:0' + str(self.number_of_address_bits) + 'b}'



    def _MapAddress(self, address):
        # format the address  to binary like string
        address = self._str_format.format(address)

        # where we have ones, are those that form the address in the cache
        mapped_address = ""
        for i in xrange(self.number_of_address_bits):
            if self.mapping[i] == "1":
                mapped_address += address[i]

        # return the integer (address on cache)
#        print "mapped_address", mapped_address
        return int(mapped_address,2)


    def _ReadFromCache(self, full_address, cache_address):

        value = full_address
#        print full_address, cache_address
        if self.data.has_key(cache_address):
            value = self.data[cache_address]
            v = value[-1]
            if v != full_address:
#                print cache_address, v, full_address
```

```
#                print bin(cache_address), bin(v), bin(full_address)
                value.append(full_address)
                self.data[cache_address] = value
                return 0
            else:
                return 1
        else:
            # address not there yet, cold miss
            self.data[cache_address] = [value]
            return 0

        return 0

    def _ReadFromCache2(self, full_address, cache_address):

        value = full_address
        if self.data.has_key(cache_address):
            if self.data[cache_address] == full_address:
                return 1
            else:
                self.data[cache_address] = value
                return 0
        else:
            # address not there yet, cold miss
            self.data[cache_address] = value
            return 0


    def Read(self, address):
        '''
            "Read" from memory and return 1 if hit, 0 if miss.
        '''

        pos = self._MapAddress(address)
        hit = self._ReadFromCache(address, pos)
        return hit


    def GetTotalMissConflicts(self):
        total = 0
        for item in self.data.iteritems():
            # in each position there is a list with all addresses mapped to that position
            conflicting_addresses = item[1]
            conflicting_addresses.sort()

            j = 0
            number_of_addresses = len(conflicting_addresses)
            if number_of_addresses > 1:
        #        print conflicting_addresses
                while j < number_of_addresses:
                    address = conflicting_addresses[j]
                    c = conflicting_addresses.count(address)
                    total+= c - 1
                    j += c

        return total


class XORDMCache(DMCache):


    def __init__(self, main_memory_size, size, mapping, energy_per_read, energy_per_write):
        super(XORDMCache, self).__init__(main_memory_size, size, mapping, energy_per_read, energy_per_write)
```

```
    def _MapAddress(self, address):


        # format the address  to binary like string
        address = self._str_format.format(address)

        bits = ""
        for i in range(3, 9):
            bits += str(int(address[-1*i+1]) ^ int(address[-1*i -6]))

#-------------------------------------------------------------- reverse
        a = bits
        a = list(a)
        a.reverse()
        bits = ""
        for i in a:
            bits +=i

#        print bits
        pos = int(bits + address[-2:], 2)

#        print bits + address[-2:]
#        print pos
        return pos



#==============================================================================
# Main
#==============================================================================
if __name__ == '__main__':
    pass
```

```
trace_file = "trace.log"
analysis_file = "input/list_of_functions.xls"
mapping = {
           "CacheFactory32Words"   : "000010000000111100",
           "CacheFactory64Words"   : "000000010010111100",
           "CacheFactory128Words"  : "000000100011111100",
           "CacheFactory256Words"  : "000000100111111100",
           "CacheFactory512Words"  : "000100110011111100",
           "CacheFactory1024Words" : "001000011111111100",
           "CacheFactory2048Words" : "001001011111111100",
           "CacheFactory4096Words" : "000101111111111100"
}

run = {
        "FactoryDMCache1024WordsBaselineMemorySystem"            : True,
        "FactoryDMCache1024WordsMemorySystem"                    : True,
        "FactoryDMCache128WordsBaselineMemorySystem"             : True,
        "FactoryDMCache128WordsMemorySystem"                     : True,
        "FactoryDMCache2048WordsBaselineMemorySystem"            : True,
        "FactoryDMCache2048WordsMemorySystem"                    : True,
        "FactoryDMCache256WordsBaselineMemorySystem"             : True,
        "FactoryDMCache256WordsMemorySystem"                     : True,
        "FactoryDMCache256WordsXORMemorySystem"                  : True,
        "FactoryDMCache32WordsBaselineMemorySystem"              : True,
        "FactoryDMCache32WordsMemorySystem"                      : True,
        "FactoryDMCache4096WordsBaselineMemorySystem"            : True,
        "FactoryDMCache4096WordsMemorySystem"                    : True,
        "FactoryDMCache512WordsBaselineMemorySystem"             : True,
        "FactoryDMCache512WordsMemorySystem"                     : True,
        "FactoryDMCache64WordsBaselineMemorySystem"              : True,
        "FactoryDMCache64WordsMemorySystem"                      : True,
        "FactoryFlashMemoryBaselineMemorySystem"                 : True,
        "FactorySPM512MemorySystem"                              : True,
        "FactorySPM512WithDMCache1024WordsBaselineMemorySystem"  : True,
        "FactorySPM512WithDMCache1024WordsMemorySystem"          : True,
        "FactorySPM512WithDMCache128WordsBaselineMemorySystem"   : True,
        "FactorySPM512WithDMCache128WordsMemorySystem"           : True,
        "FactorySPM512WithDMCache2048WordsBaselineMemorySystem"  : True,
        "FactorySPM512WithDMCache2048WordsMemorySystem"          : True,
        "FactorySPM512WithDMCache256WordsBaselineMemorySystem"   : True,
        "FactorySPM512WithDMCache256WordsMemorySystem"           : True,
        "FactorySPM512WithDMCache32WordsBaselineMemorySystem"    : True,
        "FactorySPM512WithDMCache32WordsMemorySystem"            : True,
        "FactorySPM512WithDMCache4096WordsBaselineMemorySystem"  : True,
        "FactorySPM512WithDMCache4096WordsMemorySystem"          : True,
        "FactorySPM512WithDMCache512WordsBaselineMemorySystem"   : True,
        "FactorySPM512WithDMCache512WordsMemorySystem"           : True,
        "FactorySPM512WithDMCache64WordsBaselineMemorySystem"    : True,
        "FactorySPM512WithDMCache64WordsMemorySystem"            : True,
        "FactorySPM1kMemorySystem"                               : True,
        "FactorySPM1kWithDMCache1024WordsBaselineMemorySystem"   : True,
        "FactorySPM1kWithDMCache1024WordsMemorySystem"           : True,
        "FactorySPM1kWithDMCache128WordsBaselineMemorySystem"    : True,
        "FactorySPM1kWithDMCache128WordsMemorySystem"            : True,
        "FactorySPM1kWithDMCache2048WordsBaselineMemorySystem"   : True,
        "FactorySPM1kWithDMCache2048WordsMemorySystem"           : True,
        "FactorySPM1kWithDMCache256WordsBaselineMemorySystem"    : True,
        "FactorySPM1kWithDMCache256WordsMemorySystem"            : True,
        "FactorySPM1kWithDMCache32WordsBaselineMemorySystem"     : True,
        "FactorySPM1kWithDMCache32WordsMemorySystem"             : True,
        "FactorySPM1kWithDMCache4096WordsBaselineMemorySystem"   : True,
        "FactorySPM1kWithDMCache4096WordsMemorySystem"           : True,
        "FactorySPM1kWithDMCache512WordsBaselineMemorySystem"    : True,
```

```
        "FactorySPM1kWithDMCache512WordsMemorySystem"             : True,
        "FactorySPM1kWithDMCache64WordsBaselineMemorySystem"      : True,
        "FactorySPM1kWithDMCache64WordsMemorySystem"              : True,
        "FactorySPM2kMemorySystem"                                : True,
        "FactorySPM2kWithDMCache1024WordsBaselineMemorySystem"    : True,
        "FactorySPM2kWithDMCache1024WordsMemorySystem"            : True,
        "FactorySPM2kWithDMCache128WordsBaselineMemorySystem"     : True,
        "FactorySPM2kWithDMCache128WordsMemorySystem"             : True,
        "FactorySPM2kWithDMCache2048WordsBaselineMemorySystem"    : True,
        "FactorySPM2kWithDMCache2048WordsMemorySystem"            : True,
        "FactorySPM2kWithDMCache256WordsBaselineMemorySystem"     : True,
        "FactorySPM2kWithDMCache256WordsMemorySystem"             : True,
        "FactorySPM2kWithDMCache32WordsBaselineMemorySystem"      : True,
        "FactorySPM2kWithDMCache32WordsMemorySystem"              : True,
        "FactorySPM2kWithDMCache4096WordsBaselineMemorySystem"    : True,
        "FactorySPM2kWithDMCache4096WordsMemorySystem"            : True,
        "FactorySPM2kWithDMCache512WordsBaselineMemorySystem"     : True,
        "FactorySPM2kWithDMCache512WordsMemorySystem"             : True,
        "FactorySPM2kWithDMCache64WordsBaselineMemorySystem"      : True,
        "FactorySPM2kWithDMCache64WordsMemorySystem"              : True,
        "FactorySPM4kMemorySystem"                                : True,
        "FactorySPM4kWithDMCache1024WordsBaselineMemorySystem"    : True,
        "FactorySPM4kWithDMCache1024WordsMemorySystem"            : True,
        "FactorySPM4kWithDMCache128WordsBaselineMemorySystem"     : True,
        "FactorySPM4kWithDMCache128WordsMemorySystem"             : True,
        "FactorySPM4kWithDMCache2048WordsBaselineMemorySystem"    : True,
        "FactorySPM4kWithDMCache2048WordsMemorySystem"            : True,
        "FactorySPM4kWithDMCache256WordsBaselineMemorySystem"     : True,
        "FactorySPM4kWithDMCache256WordsMemorySystem"             : True,
        "FactorySPM4kWithDMCache32WordsBaselineMemorySystem"      : True,
        "FactorySPM4kWithDMCache32WordsMemorySystem"              : True,
        "FactorySPM4kWithDMCache4096WordsBaselineMemorySystem"    : True,
        "FactorySPM4kWithDMCache4096WordsMemorySystem"            : True,
        "FactorySPM4kWithDMCache512WordsMemorySystem"             : True,
        "FactorySPM4kWithDMCache64WordsBaselineMemorySystem"      : True,
        "FactorySPM4kWithDMCache64WordsMemorySystem"              : True,
        "FactorySPMFullSizeMemorySystem"                          : True
}
```