Politecnico di Torino

III Facoltà di Ingegneria

Energy management of IoT application
Report laboratory session 1

Master degree in Computer Engineering

Authors: group 14

Kristina Rogacheva, Giuseppe Carrubba

January 11, 2022

Contents

CHAPTER 1

Introduction

The goal of the first laboratory is to understand the basics of Dynamic Power Management by simulating a Power State Machine with a simulator. The ultimate intent is to study the different power management policies and the analysis of their results.

The activity is divided in three main parts described in this report:

- Default timeout policy
- Extension of the timeout policy with 2 low-power states
- Predictive policy

CHAPTER 2

Timeout policy

During the first part of this experience we used a simplified version of the Power State Machine (PSM) that can do transitions just from `Run` state to `Idle` and vice-versa. In the second part we repeated the experience for the other idle state `Sleep`, and for the last one we completed the PSM by using both low-power states `Sleep` and `Idle`.
In both cases, we analyzed the results that emerged from the two timeout parameters.

2.1 PART 1: Timeout policy with just one idle state in PSM

As said before, for the first part of the laboratory we worked with a simplified version of the PSM that has just two states: `Run` state and `Idle` state (see Figure 2.1). The third state (Sleep), is disabled by the DPM simulator which doesn't allow the transitions to and from this state. All the possible transitions are defined in the following table:

Transition	Energy costs	Time
`Run`→`Idle`	10 μJ	400 μs
`Idle` → `Run`	10 μJ	400 μs

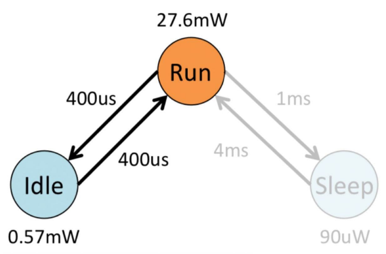


Figure 2.1: Power State Machine with two states

In this case, with just one active and one idle state, the system will work in `Run` state and evaluate if it should switch to an `Idle` state. To accomplish that, the system has to wait for a certain amount of time after the end of its work. This period is called `Timeout time`. If the idle period is bigger than the timeout time, the system can change its state from `Run` to `Idle`.

Taking into account the transition costs and the non-ideal power consumption, the timeout time should be carefully estimated: the risk is to consume more energy than if the system remained in the active state. To calculate the best timeout time we should find the `Breakeven time`:

$$Tbe = \begin{cases} Ttr & \text{when } P_{on} \geq P_{tr} \\ Ttr + Ttr\frac{Ptr-Pon}{Pon-Poff} & \text{when } P_{on} < P_{tr} \end{cases} \tag{2.1}$$

P_{on} is known, so we have to compute the P_{tr}:

$$P_{tr} = \frac{E_{tr}}{T_{tr}} \tag{2.2}$$

$$T_{tr} = T_{on-off} + T_{off-on} = 400\ \mu s + 400\ \mu s = 800\ \mu s \tag{2.3}$$

$$E_{tr} = E_{on-off} + E_{off-on} = 10\ \mu J + 10\ \mu J = 20\ \mu J \tag{2.4}$$

$$P_{tr} = \frac{E_{tr}}{T_{tr}} = \frac{20\ \mu J}{800\ \mu s} = 25\ mW \tag{2.5}$$

Since $P_{on} = 27,6\ mW > P_{tr} = 25\ mW$, $T_{be} = T_{tr} = 800\ \mu s$.

This Breakeven time acts as a threshold, every idle time equal or longer than T_{be} allows reducing the power consumption without encountering any performance penalties.

2.1.1 Python scripts

To make our analysis automatic and systematic, we developed different scripts written in Python. The script is called `script_part1_wN.py` and it systematically runs the simulator several times, with the respective workload, and produces useful data and graphs to understand the Policy applied to the Power State Machine (PSM). In summary, the script runs the simulator **N** times depending on the number in range specified in the first for loop. In our examples, we run the simulator 500 times with 500 different timeout values (from zero to 500) for the workload given in the example folder. For the remaining workloads, to obtain exhaustive results, the simulator is launched with more than 100.000 different timeout values.

The script is also capable to collect useful information and save them in different files with the following organizations:

- **results_workloadname.txt** that contains results printed directly by the simulator for every timeout value tested;
- **dpm_energy_workloadname.txt** contains list of pairs Timeout and Power consumed by the system with that specific timeout value;
- **best_dpm_workloadname.txt** shows the best results obtained after the numerous runs.

The last part of the script is in charge of producing figures to show the behavior of the PSM based on the results obtained by launching several policies. In particular, it analyzes the values collected in the file `dpm_energy_workloadname.txt`.

It produces two figures: the first one explains graphically how the power consumption behaves based on the value assigned to the timeout. The second one shows the Energy saved expressed in percentage following again the value of the timeout.

The script has been launched with the three given workloads and the results have also been collected inside folders to organize the files more precisely. The script contains all the comments necessary to make it understandable.

To test the script and obtain results, it is needed to type in a python terminal the name of the script. It is mandatory to use a version of python equal to or greater than version 3.2.

To obtain readable graphs and useful readable results, different scripts have been developed, one for each given workload.

2.1.2 Results obtained

In this section, we will go through all of our results obtained by launching the scripts several times. We will proceed by showing also figures for each workload tested with the simulator.

First of all, the `Tbe` for the PSM has been calculated as stated above in 2.1. Once obtained the `Tbe`, it was immediately noticed that the simulator limits its discretization in time equal to 1ms making it impossible for the study values of the `Timeout` lower than 1ms.

Having a `Tbe` lower than 1ms, we tried to figure out how to test values lower than 1ms, but achieving this would have meant substantially changing the simulator. Analyzing values below the millisecond would have given us the possibility of obtaining more precise results and exploit more the timeout policy.

We decided to spend more effort in considering cases in which the Timeout used is exactly zero and the other values larger than the `Tbe`.

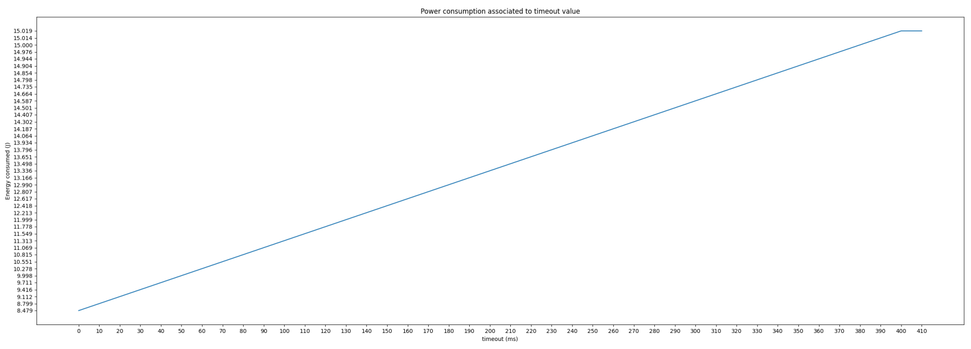


Figure 2.2: Power consumed by the PSM for workload0.

In the figure 2.2 the workload inside the `Example` folder has been analyzed and we have denominated it as `workload0`. It's easy to notice that the power consumed by the system **increases linearly** as the value assigned to the timeout is larger.

For this workload, graphs were produced by increasing the Timeout by 1 ms at a time. Considering how the PSM is characterized, the results are pretty obvious for this part. The cost of the transition is pretty low and the time for performing the transition is also limited. In summary, it is always convenient to shut down, no matter how much lasts the idle interval - the transition costs are negligible; this is because we have obtained a very small value for the `Tbe`. For every second we wait for shutdown, we pay that extra seconds staying in the `RUN` state before going to the `IDLE` state and so the power consumption increases accordingly. For this workload, a total number of 420 cases has been simulated to reach a power consumption equal to the one obtainable without using any policy.

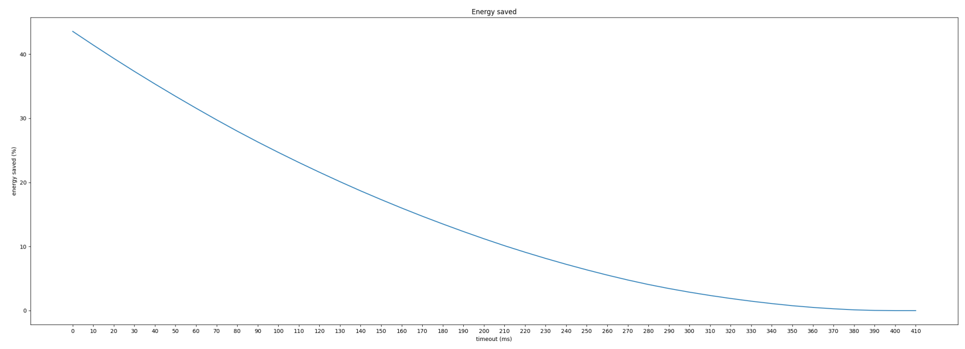


Figure 2.3: Energy saved in % with the DPM for workload0.

In 2.3 is showed how much energy (in percentage) is saved depending on which value of timeout is used for the policy. Ideally, the best choice is to use a timeout equal to zero, and this has been explained above. But, `Timeout=0` is considered only if any constraints in terms of performances have been set.

In conclusion, the best result is obtained by using a timeout equal to zero, ignoring the performance penalties, with almost 43% of savings.

Very similar results have been obtained for the other two given workloads. For them, a larger number of simulations have been launched. The reason why this has been done is that, due to the Idle interval given for the two workloads, higher values for the timeout were usable and so also testable before reaching the maximum power consumption.

Below are shown figures obtained by running several times the simulator for the workload1.

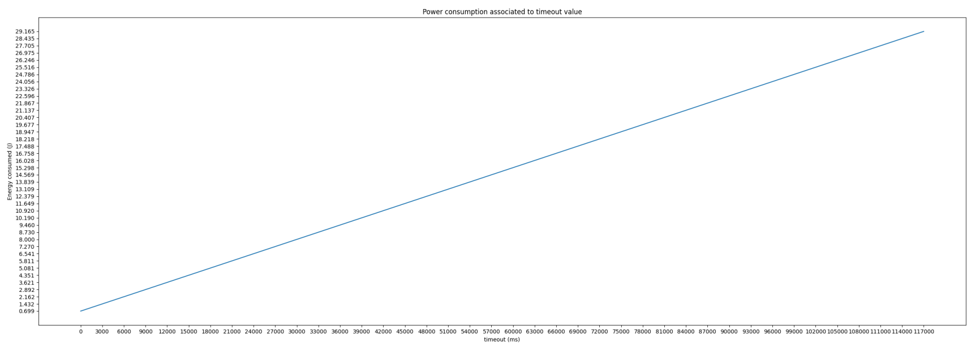


Figure 2.4: Power consumed by the PSM for workload1.

For the two extra workloads, a total number of `120.000` cases have been simulated with values of Timeout in the range from 0 to `119.999`. To run the script in a reasonable amount of time, the sample we analyzed has been reduced by a factor of 50.

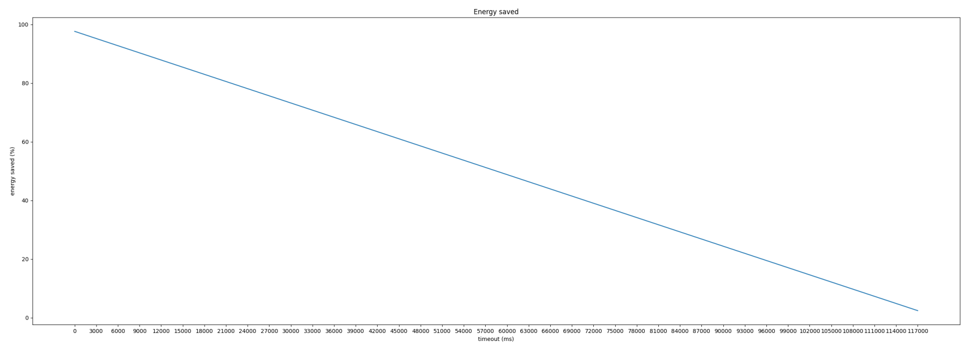


Figure 2.5: Energy saved in % with the DPM for workload1.

Below are showed the figures obtained by running several times the simulator for the workload2.

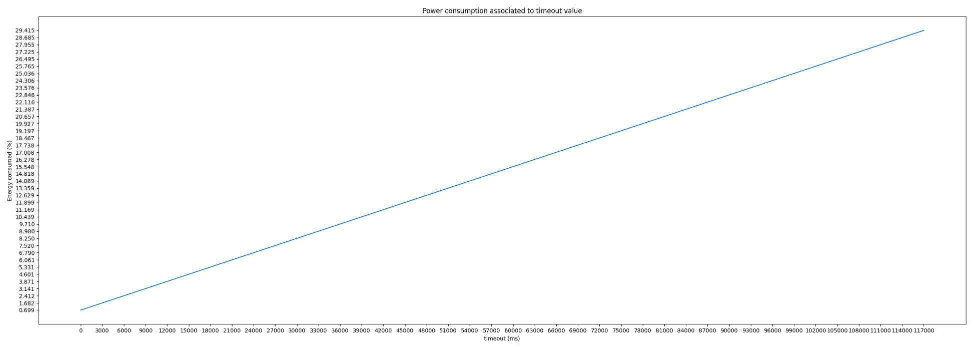


Figure 2.6: Power consumed by the PSM for workload2.

With respect to the graph obtained for `workload0` about the energy saved, there is no appreciably visible curve. This is due to the higher amount of cases tested and also because for a certain range of values (like from 2.000 to 15.000) of Timeout, the energy consumption decreases linearly for the reason explained above about the linear increase of power consumption.

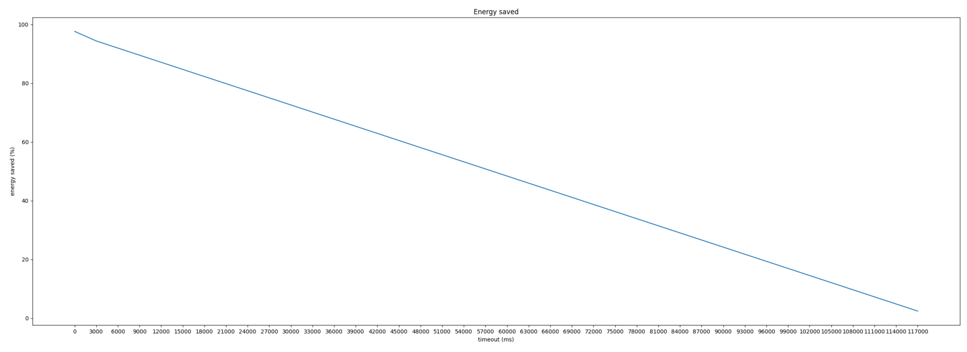


Figure 2.7: Energy saved in % with the DPM for workload2.

Again, for both of them, the best result is obtained by using a timeout equal to zero and achieving almost the 97% of savings for workload1 and a 98% for workload2.

Looking at both workloads, there is a specific interval that is huge. Shut down in that interval gives the heaviest contribution in calculating the average of energy saved.

2.2 PART 2: Timeout policy with one Sleep state in PSM

For the second part of the laboratory, we worked with another simplified version of the PSM, but this time instead of the `Idle` state we had the `Sleep` state. The idle state is disabled by the DPM simulator which doesn't allow the transitions to and from this state. All the possible transitions are defined in the following table:

Transition	Energy costs	Time
`Run`→`Sleep`	0.02 mJ	1 ms
`Sleep` → `Run`	2 mJ	4 ms

In this case, still, with just one active and one idle state, the system will work in `Run` state and evaluate if it should switch to an `Sleep` state. As before, the system has to wait for Timeout time after the end of its work, and check if it can change its state from `Run` to `Sleep`.

Taking into account the transition costs and the non-ideal power consumptions, the timeout time should be carefully estimated: the risk is to consume more energy than if the system remained in the active state. In this case, the transition costs are much bigger: we can evaluate the change by calculating the `Breakeven time`:

$$Tbe = \begin{cases} Ttr & \text{when } P_{on} \geq P_{tr} \\ Ttr + Ttr\frac{Ptr-Pon}{Pon-Poff} & \text{when } P_{on} < P_{tr} \end{cases} \tag{2.6}$$

P_{on} is known, so we have to compute the P_{tr}:

$$P_{tr} = \frac{E_{tr}}{T_{tr}} \tag{2.7}$$

$$T_{tr} = T_{on-off} + T_{off-on} = 1\ ms + 4\ ms = 5\ ms \tag{2.8}$$

$$E_{tr} = E_{on-off} + E_{off-on} = 0.02\ mJ + 2\ mJ = 2.02\ mJ \tag{2.9}$$

$$P_{tr} = \frac{E_{tr}}{T_{tr}} = \frac{2.02\ mJ}{5\ ms} = 404\ mW \tag{2.10}$$

Since $P_{on} = 27,6\ mW < P_{tr} = 404\ mW$ we have to calculate the Tbe using the second equation of (2.6):

$$Tbe = Ttr + Ttr\frac{Ptr - Pon}{Pon - Poff} = 5 + 5\frac{404\ mW - 27.6\ mW}{27.6\ mW - 0.09\ mW} = 73.41\ ms \tag{2.11}$$

Now we know that $T_{be} = 73.41\ ms$.

This Breakeven time acts as a threshold, every idle time equal or longer than T_{be} allows reducing the power consumption without encountering any performance penalties.

2.2.1 Simulator modification

To run the new `OFF-STATE` in the PSM, the simulator has been modified as suggested in the Lab guide provided. In the function `dpm_decide_state` in `src/dpm_policies.c` instead of the transition to the `Idle` state, we perform a transition to the `Sleep` state when the Timeout runs out.

2.2.2 Python scripts

Again for this part, several python scripts have been developed to automatically and systematically make our analysis. The scripts are called `script_part2_wN.py` where N defines which workload is under analysis. Like for part 1, the scripts run the simulator N times depending on the number in range specified in the first for loop, but now the PSM has only the sleep state as `OFF-STATE`. The scripts produce the same files as described above for part 1.

In this part, 2 different `.txt` files have been produced for each given workload in the `Energy_w_Dpm` folder. One contains a certain number value of Timeouts, and so several more simulations (420 for w0 and 120000 for w1 and w2), in order to explore the space of solutions of the policy until reaching the maximum energy consumption equal to the energy consumed without any policy. The second file named as `" Tbe"` in the end, are files where the region near the Tbe has been zoomed in. To obtain this, fewer simulations have been launched. In summary, to zoom in, only the first 140 possible values of the Timeout have been simulated.

Also, new scripts in this part have been added to highlight the differences between results obtained in part 1 and part 2. These scripts are simply called `script_differences_wN.py` where N defines which workload is under analysis. The script takes the results collected in the files `dpm_energy_wN.txt` in the folder `Energy_w_dpm`. Figures produced by the new scripts can be found in the folder `printed_graphs`.

Taking the pair **Timeout** and **Energy_saved**, the script is capable to plot a figure that shows both curves and so makes more understandable the differences between the two policies.

2.2.3 Results and differences between part1

Now that the PSM is not equal to the one in part 1, a different Tbe has to be considered and it has been calculated in 2.11. This means that we should test the Timeout value equal to or greater than 73ms to have a policy that allows us to save energy without any performance penalties. In our script, we tested the PSM trying to explore the space solution as much as possible and focusing on also simulating the value of timeout lower and grater the TBE and see what happens to zoom near it. This has been done in order to answer all the questions given in the laboratory PDF guide.

Analyzing the workload1, it is noticeable that for a very low value of timeout, some overhead is appreciable due to the high number of transitions combined with the time needed by the PSM to make that transition to the sleep state.

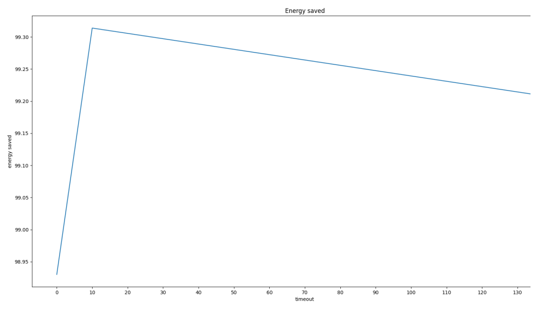
Indeed, looking at the initial values, we can see that the energy saved is not as much as it is obtainable for larger values. By simulation, it has been obtained that the best **Tto** is 1 with a power-saving of 99.32%. Considering the constraint of the performances and so the Tbe, we can achieve the best policy using a **Tto** = 73ms achieving 98.40% of savings.

For workload2, we can see that the best **Tto** is zero. This could be true if the implementation of the policy aims at having a system that is more energy-efficient with negligible performance penalties.

Otherwise, in presence of performance constraints, a value of timeout greater than the Tbe should be taken.

The best **Tto** for the workload2 considering the Tbe constraint is a Tto = 117ms with a savings of 99.40%.

In general, this very high percentage of savings is possible because, looking at the workload files, there are one or more idle intervals that last for a very long time.



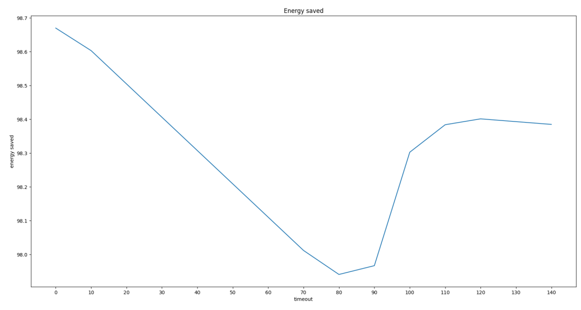


Figure 2.8: Energy saved behaviour of workload1. Figure 2.9: Energy saved behaviour of workload2.

Those intervals are crucial for the policies. Having such a big Idle interval makes the contribution of energy consumption extremely heavy in the calculation of the average consumption.
Now let's highlight the difference between the two policies, where the first one use only the `Idle` state and the other with only the `Sleep` state available as `OFF-STATE`.

The Workload given as an example (also called by us "workload0") is the first that has been analyzed and compared with the two policies.

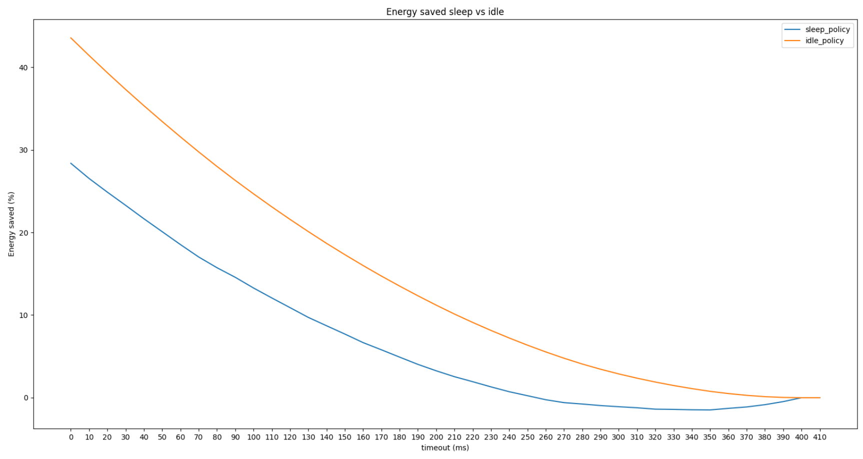


Figure 2.10: Energy saved comparison between different policies for workload0.

In figure 2.10 we can see that the `Idle` policy is more exploitable because we can use timeout values below 73ms, due to different Tbe, and save more energy without sacrifice performances. But in general, the behavior is that, for equal values of Timeout, the idle policy allows to save more energy.

For workload1 the situation is the opposite. In this scenario, the `Sleep` policy allows us to save more energy for any Timeout value 2.11.

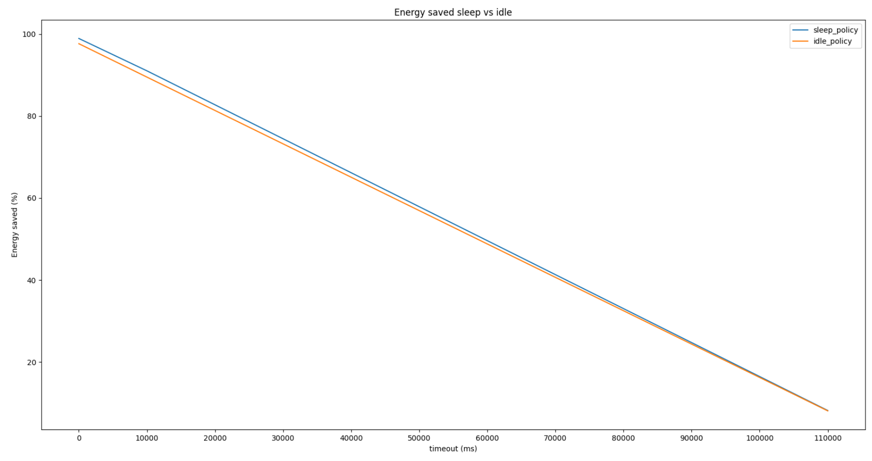


Figure 2.11: Energy saved comparison w1.

The results obtained by comparing the policies for `workload2` (Figure 2.12) are also very similar to the one obtained with `workload1`. The policy with the sleep state seems that give more energy savings but with less exploitability for the Timeout values due to the Tbe constraint.

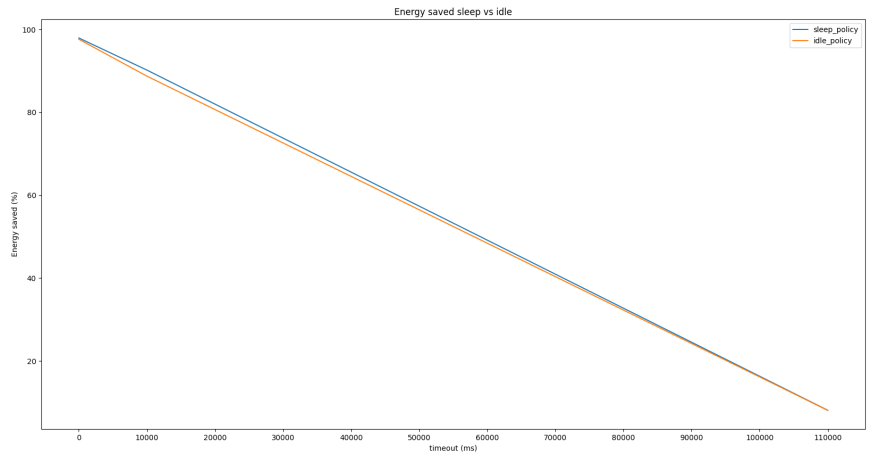


Figure 2.12: Energy saved comparison w2.

2.3 PART 2 EXTRA: Timeout policy with transition to both Idle and Sleep states in PSM

After the implementation of the timeout policy for both idle states separately, we tried to merge these solutions to create a timeout policy with both idle states - the new PSM can be seen in the Figure 3.5). All the possible transitions are defined in the following table:

Transition	Energy costs	Time
`Run`$\rightarrow$`Idle`	10 μJ	400 μs
`Idle` $\rightarrow$ `Run`	10 μJ	400 μs
`Run`$\rightarrow$`Sleep`	0.02 mJ	1 ms
`Sleep` $\rightarrow$ `Run`	2 mJ	4 ms

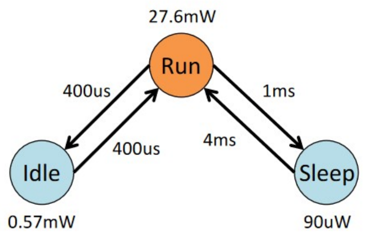


Figure 2.13: Power State Machine with all states

In this case, there are three possible states and no link between the two idle ones. To realize a policy that includes all phases there are different approaches. Instead of creating a link between all states, we decided to maintain the original PSM. We used two different timeouts to establish when and to what state we should proceed - the system checks the first timeout for the idle state and then the timeout for the sleep state. In case the timeout for the Sleep state is expired so we can transit to this state, but the current state is `Idle`, there is a quick transition to `Run` state and then to `Sleep`. Taking into account the transition costs and the non-ideal power consumption, the timeout time should be carefully estimated: the risk is to consume more energy than if the system remained in the active state. The `Breakeven time` for both states was calculated in the previous parts of this report.

2.3.1 Python scripts

For this part, a python script has been developed to simulate exhaustively our policy. This time, having both `OFF-STATE` available and according to the modifications we have done to the simulator, we launched several simulations. To obtain good results and to search in a semi-exhaustive way the space of possible solutions, we tested `N` values of the Timeout for the idle state with `M` value of the Timeout for the sleep state. Proceeding with this logic, we tested a quadratic number of cases to find the best Tto for both idle and sleep states.

Similarly, as we did for previous scripts, in the end, we searched for the best results and tried then to show obtained samples through a graph.

2.3.2 Results Obtained

To test several cases in a reasonable amount of time we limited the number of tests to a maximum of 50 tries for the Idle timeout and 100 tries for the sleep timeout. In this way to test in total 5000 cases to find the best Tto for the policy.

We obtained interesting results.

For the first workload, we obtained that the policy has as best Timeout for idle equal to 0 and 100 for the sleep timeout, with the energy savings equal to 99,30%. Same values of the Timeout have been discovered for the workload 2 with a percentage of energy saved equal to 99%. Again, we have not modified the PSM so we tried to adapt the policy in such a way both `OFF-STATE` were exploitable. The negative aspect of this choice is that not having a direct connection between `Idle` and `Sleep` state causes a huge overhead of transition that in cascade causes a lot of energy waste. For this reason with this configuration, we have not obtained better results compared to the other two policies where only one `OFF-STATE` was allowed.

CHAPTER 3

History prediction policy

3.1 PART 3: Predictive History Policy based on the threshold

For the last part of the laboratory experience, we used the complete PSM (Figure 3.5) with a `Run` state and both idle states `Idle` and `Sleep`. This time we used the Predictive History Policy based on the threshold, but we had to modify it to ensure the best results possible. We analyzed the provided workloads and the previous active time before every idle time: we discovered that long active periods are followed by even longer idle periods. This is the reason why we evaluated T_{active} - if $T_{active} > T_{th}$ we can assume that $T_{idle} > T_{be}$ so we can shut down immediately without waiting for the timeout to expire. Although it is convenient being able to shutdown immediately, the threshold on T_{active} doesn't guarantee that T_{idle} will be $>$ or $<$ T_{be}.

3.1.1 Simulator changes

To implement this custom policy we modified the function `dpm_decide_state` in `src/dpm_policies.c` of the simulator. We added a new parameter to the function: `prev_active_time`, which is the history criterion on which the policy is based.
The previous T_{active} is evaluated: in case $T_{active} > T_{th}$ the system immediately shuts down to `Sleep` state, otherwise there is a transition to `Idle` state. From the study of the PSM for the previous policies, we know that it is always convenient to perform a transition to `Idle` state because of the reduced transition costs, but this state still consumes power. On the other hand, the `Sleep` state consumes way less power, but its transition costs are too expensive. These are the reasons why it is important to set the right T_{th}.

3.1.2 Python scripts

For this part, again, python scripts have been developed to perform our analysis. The scripts follow the same steps as the scripts written for the previous parts, but now they run the command that makes the simulator use the predictive policy. For each given workload, a script has been developed in such a way the number of simulations returns a reasonable amount of data, from each workload simulation, to visualize them in a more readable manner. Then once all the simulations have been launched, the script is capable to collect those data and express them through graphs that show the energy consumption and the energy saved based on the value of the threshold tested.

In addition, a script called `w_analysis.py` has been developed to study the workloads. We started this part trying to develop the best predictive policy and therefore, we analyzed offline our workload trying to identify an L-curve that would allow us to use a policy based on a threshold. Our policy, in

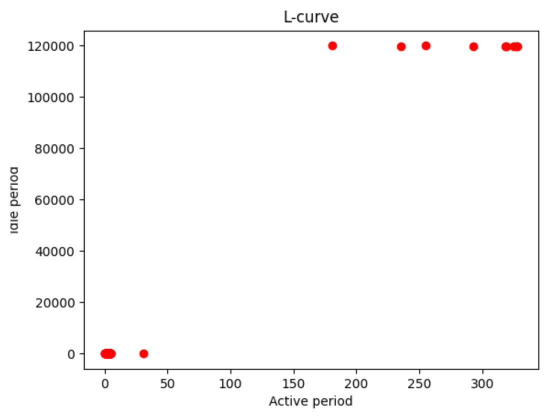


Figure 3.1: Relation curve between active(ms) and idle(ms) period W1.

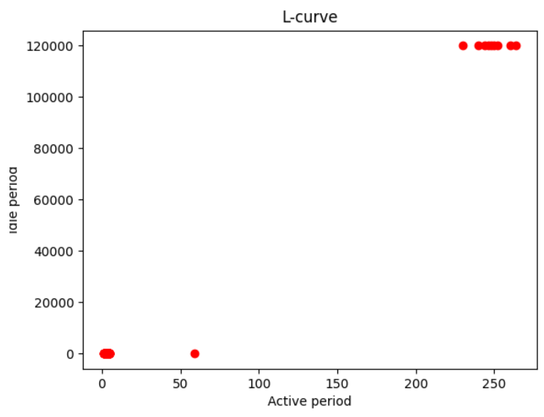


Figure 3.2: Relation curve between active(ms) and idle(ms) period W2.

the end, is the opposite of the one seen during lessons. In summary, that script takes all the **Active** and **Idle** intervals, collecting them in lists that are used to print a figure that shows the relation between the two.

3.1.3 Result obtained

As we said before we started this part by analyzing the given workload to develop the best policy. By combining active and idle intervals in a graph, we noticed that:

Very short active periods correspond to short idle periods, and long active periods correspond to **very long** idle periods. This behavior is even stronger and noticeable for workload2.

Then, starting from these results we have developed a threshold predictive policy explained in 3.1.

We also studied the workload for the given workload `example` getting very different results.

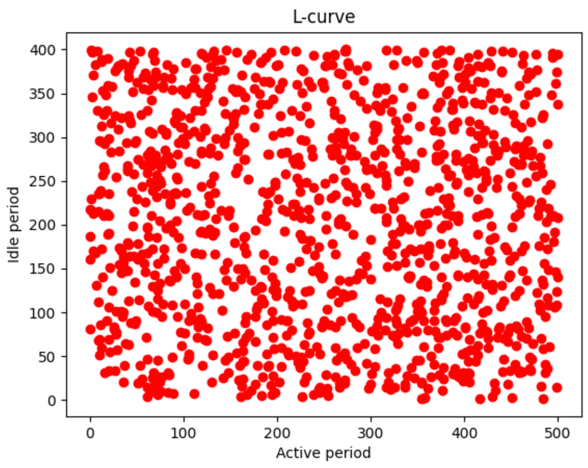


Figure 3.3: Relation curve between active(ms) and idle(ms) period W0.

Here it is noticeable that no curve can be distinguishable and so it is not sure that our policy can give good results. But, we have tested the policy also for this workload and we obtained almost the same Energy savings as we obtained for the first timeout policy in `Part1`.

Instead, testing our policy for `workload1` and `workload2`, we have obtained very excellent results compared to the `Timeout` policies.

For the first workload, we obtained very good results, surpassing the saved energy in the Timeout policy with the sleep state.

In figure 3.4 we can see that for a small threshold an overhead is experienced due to the very high amount of transition to the sleep state, according to the rules of our policy. Moving on to the higher value of threshold we can clearly notice that a huge amount of savings can be obtained because combining the switch prediction to the idle or sleep state in an efficient manner, is it possible to achieve very good savings using a Tbe greater than 73ms that allows us to exploit the energy savings without sacrifice any performances.

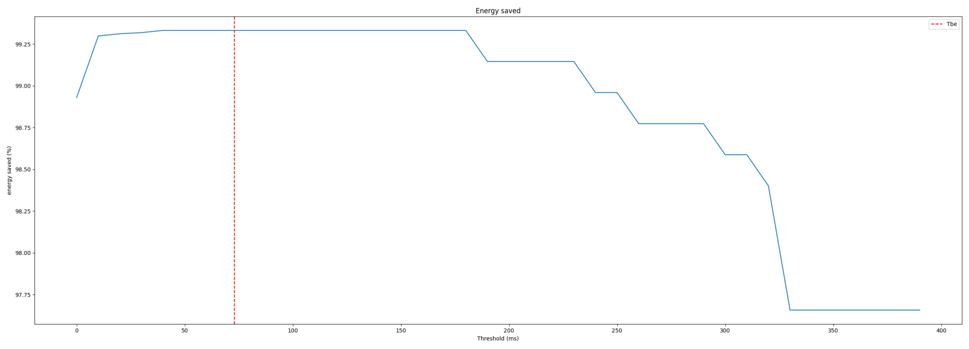


Figure 3.4: Saved energy with predictive policy on W1.

For the `workload 1` we have obtained savings equal to 99,33% with a value of the threshold equal to 34 but the same result is achievable using a range of values of the threshold from 34 to 181ms.

For the `workload2` we have experienced the same behavior as workload 1 for the energy saved. For a small threshold value, we obtain some higher consumption due to the overhead of transitions to the sleep state according to the rules of our policy.

then for a range of threshold values from 62ms to 230ms, we can achieve the best energy savings that are equal to 99,34%.

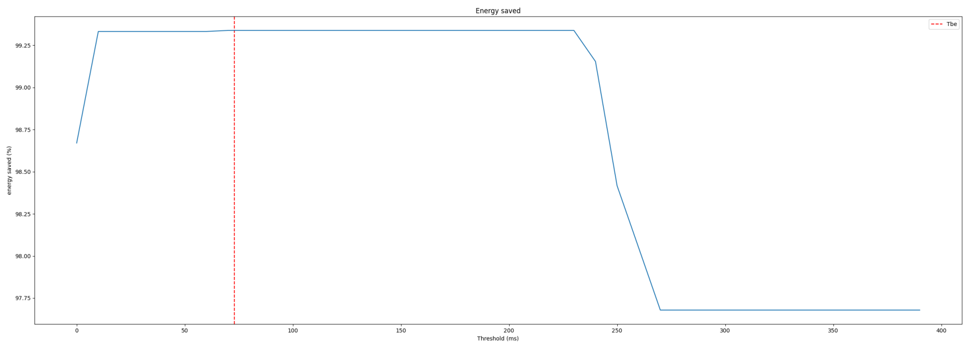


Figure 3.5: Saved energy with predictive policy on W2.

These results based on a Predictive policy, are better compared to the results obtained with a Timeout policy. Furthermore, if a predictive policy is developed in the right way can give you more ways to exploit idle intervals to save much energy.