POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Energy Management for IoT

Lab 1 Report

Gabriel GANZER
271961

A.Y. 2020/21

Table of Contents

1. Workload Generation

The goal of this laboratory session was to understand the basics of a Dynamic Power Management (DPM) by simulating and modifying a Power State Machine (PSM) written in C language. This chapter discuss the workload generation of those profiles used in the PSM simulation. The following sections detail how the active and idle periods were obtained. Moreover, the generated vectors were combined into a "*.txt*" file that serves as the input requirement imposed by the simulator.

1.1. Active Periods

The length of each active period was generated in accordance to the parameters provided using a MATLAB script to build the probability density function. The piece of code in evidence below details how the random values were extracted from the function. Only integer numbers were considered, for simplicity. Figure 1.1 depicts the histogram obtained after retrieving the 5000 samples within the range $min = 1\mu s$ and $min = 500\mu s$.

```
samples = 5000;
pdf1 = makedist('Uniform', 'Lower', 1, 'Upper', 500);
active = zeros(1, samples);
for i = 1:samples
    active(i) = floor(random(pdf1));
end
```

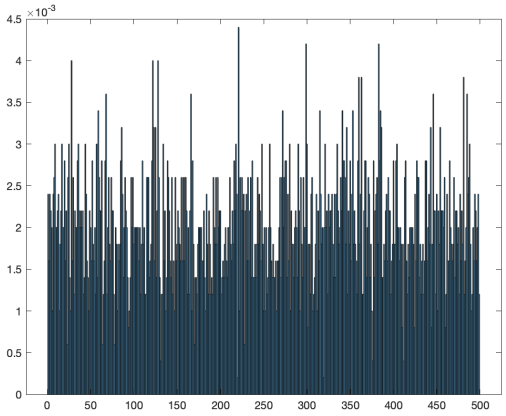


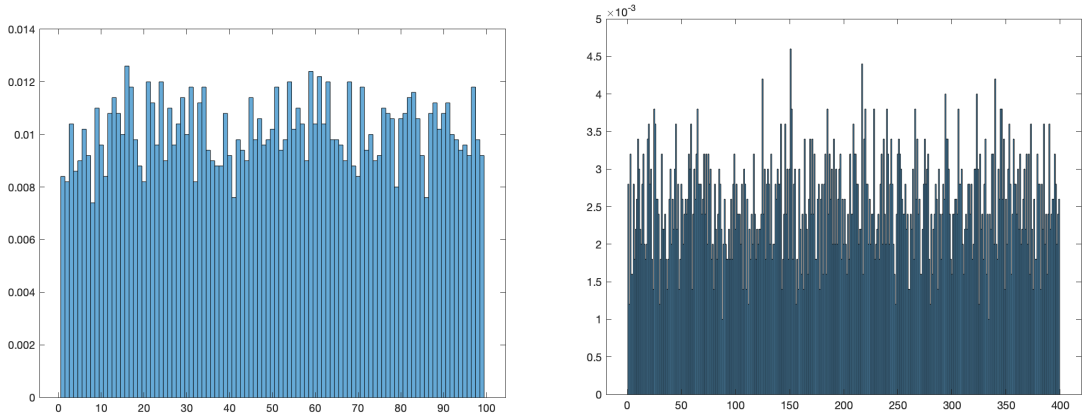
Figure 1.1: Uniform distribution of active periods.

1.2. Uniform Distribution

The high utilization profile has longer active periods, i.e., the idle periods must be relatively shorter in length. The same procedure as mentioned in the previous section was used, with the parameters $min = 1\mu s$ and $min = 100\mu s$. For the low utilization profile the parameters were $min = 1\mu s$ and $min = 400\mu s$. Figure 1.2 shows the histogram obtained for both profiles.

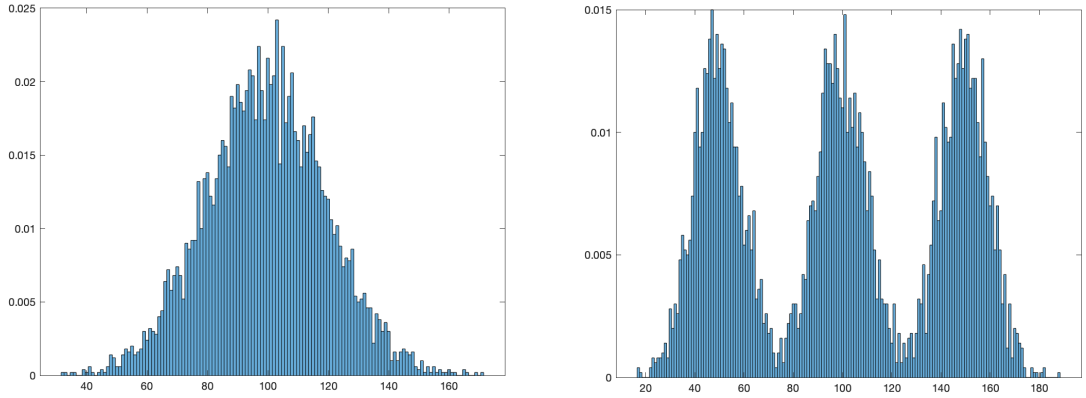
1.3. Gaussian Distribution

The probability function was modified to represent a Gaussian distribution in this case, with a mean of $\mu = 100\mu s$ and a standard deviation of $\sigma = 20\mu s$ for the Normal case. Three different means: $\mu = 50\mu s$, $\mu = 100\mu s$, and $\mu = 150\mu s$ were used in the Tri-modal case. Three vectors were extracted from different functions, each with a standard deviation of $\sigma = 20\mu s$. Furthermore, these vectors were sorted throughout the whole 5000 samples. The resultant histogram for both cases is depicted in Fig. 1.3.



(a) High Utilization **(b)** Low Utilization

Figure 1.2: Uniform distribution of idle periods.



(a) Normal **(b)** Tri-modal

Figure 1.3: Gaussian distribution of idle periods.

1.4. Exponential Distribution

The probability function was modeled as an Exponential distribution this time, with a mean $\mu = 50\mu s$. The resultant histogram is shown in Fig. 1.4. Notice the concentration of values on the lower bins, which indicates a high utilization profile with dominance of short idle periods.

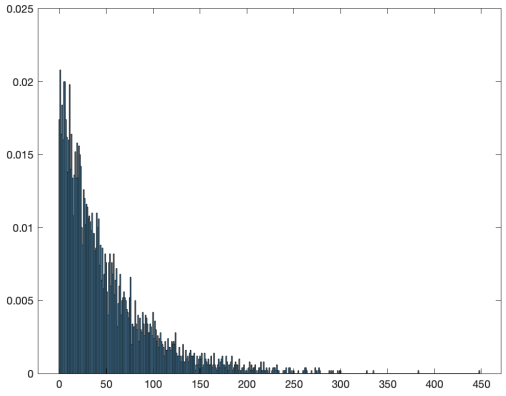


Figure 1.4: Exponential distribution of idle periods.

2. Timeout Policy

This section is dedicated to the Timeout Policy simulation using the workload profiles previously described. This policy keeps the system in a high power consuming *Run* state even after activity has ceased, transitioning to a low power state only after a certain threshold is reached, i.e., the *timeout* value.

The following code demonstrates how this policy is implemented. Two are the key parameters: the time to idle transition T_{Idle}; and the time to sleep transition T_{Sleep}. Such timeout values must be tuned to best fit the workload profile as well as to avoid unnecessary transitions that might increase the overall power consumption, as transitioning imply some energy draining and time delays. The code simply compares the current simulation time *curr_time* with both timeout parameters *tparams* and the start of the idle period, assigning the correspondent state.

```
1    if(curr_time > idle_period.start + tparams.timeout[0]) {
2        *next_state = PSM_STATE_IDLE;
3        if ((tparams.timeout[1] > tparams.timeout[0]) &&
4        (curr_time > idle_period.start + tparams.timeout[1]))
5            *next_state = PSM_STATE_SLEEP;
6    } else {
7        *next_state = PSM_STATE_ACTIVE;
8    }
```

Moreover, the simulator was modified to save the results in a text file, parsing the parameter *"-res fileName"* before execution. The results are saved in the CSV format, as it follows: first column contains the time to Idle; second column the time to Sleep; total time in Run state; total time in Idle; total time in Sleep; and finally the overall savings.

2.1. 2-State PSM

The first set of experiments were conducted over the PSM depicted in Fig. 2.1, that involves the *Run* and *Idle* states only. Each transition introduces a $10\mu s$ of delay and require $10\mu J$ of energy. The power to transition from *Run* to *Idle* and vice-versa is computed using equation 2.1.

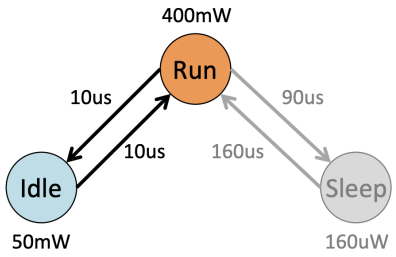


Figure 2.1: PSM implementing transitions from Run to Idle states only.

The *Break-Even Time* can be extracted from the PSM using equation 2.2. This value indicates the length of idle periods that allow a power reduction in the overall system, i.e., it gives a lower bound for worst-case energy consumption.

$$P_{tr} = \frac{E_{tr}}{T_{tr}} = \frac{E_{tr_{(Run \to Idle)}} + E_{tr_{(Idle \to Run)}}}{T_{tr_{(Run \to Idle)}} + T_{tr_{(Idle \to Run)}}} = \frac{10\mu s + 10\mu s}{10\mu J + 10\mu J} = \frac{20\mu s}{20\mu J} = 1W \tag{2.1}$$

$$T_{be} = \begin{cases} T_{tr} + T_{tr}\frac{P_{tr} - P_{on}}{P_{on} - P_{off}}, & \text{when } P_{tr} > P_{on} \\ T_{tr}, & \text{when } P_{tr} \leq P_{on} \end{cases} \tag{2.2}$$

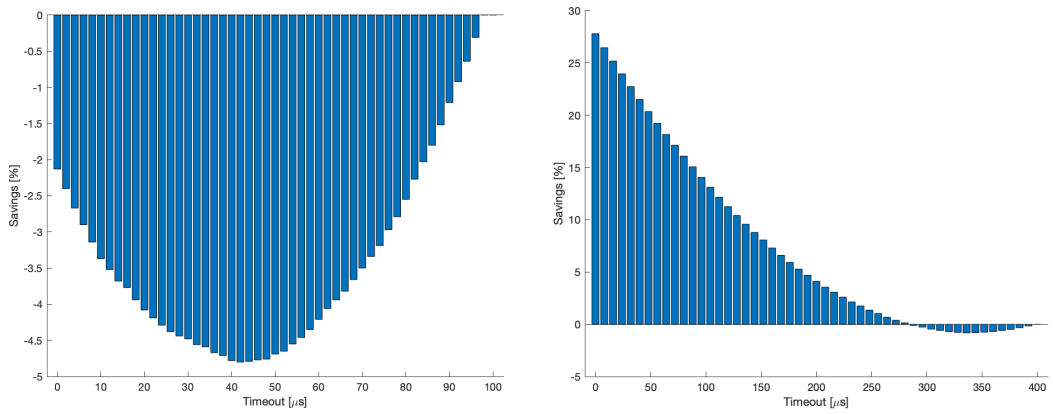
The power computed in 2.1 indicates that the system model with this PSM must actuate over some component when transitioning. Considering that $P_{on} = P_{Run} = 400mW$ and $P_{off} = P_{Idle} = 50mW$, the first case of equation 2.2 is used, as $P_{tr} > P_{on}$, yielding the T_{be} expressed by equation 2.3.

$$T_{be} = T_{tr}(1 + \frac{P_{tr} - P_{on}}{P_{on} - P_{off}}) = 20\mu s(1 + \frac{1W - 0.4W}{0.4W - 0.05W}) = 54.3\mu s \tag{2.3}$$

Last but not least, a simple bash script named *timeout_script.sh* was written to support the simulation flow. It simply increments the timeout parameter from 0 up to the maximum idle value to that specific workload, achieving 50 samples in total.

2.1.1 Results

First, the uniform distributions were evaluated. For the High Utilization profile the timeout parameter is incremented by a step of $2\mu s$ up to $100\mu s$, while the Low Utilization one is incremented by a step of $8\mu s$ up to $400\mu s$. Fig. 2.2 shows the overall savings for both workloads.



(a) High Utilization **(b)** Low Utilization

Figure 2.2: Obtained results with the Uniform Distribution of idle periods.

The discrepancy seen with these two profiles can be explained by the fact that this policy keeps the system running at a power-greedy state until the threshold is reached. On many occasions the next active period starts before the timeout threshold is met in the high utilization profile, due to its shorter idle time, discarding any chance of savings.

Likewise, the histogram depicted in Fig. 2.3 demonstrates that profiles with prevalence of shorter idle periods don't benefit from this policy. The performance here was inversely proportional to the workload profile itself, whose idle periods are concentrated towards the left edge of the histogram.

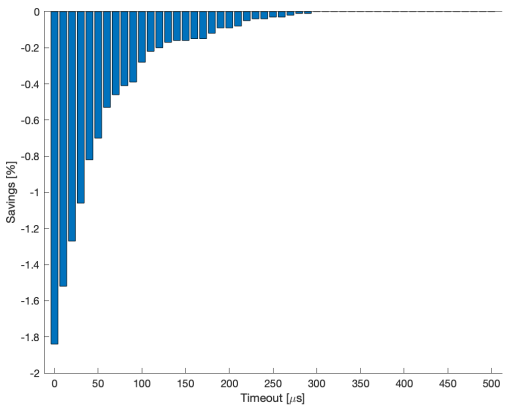
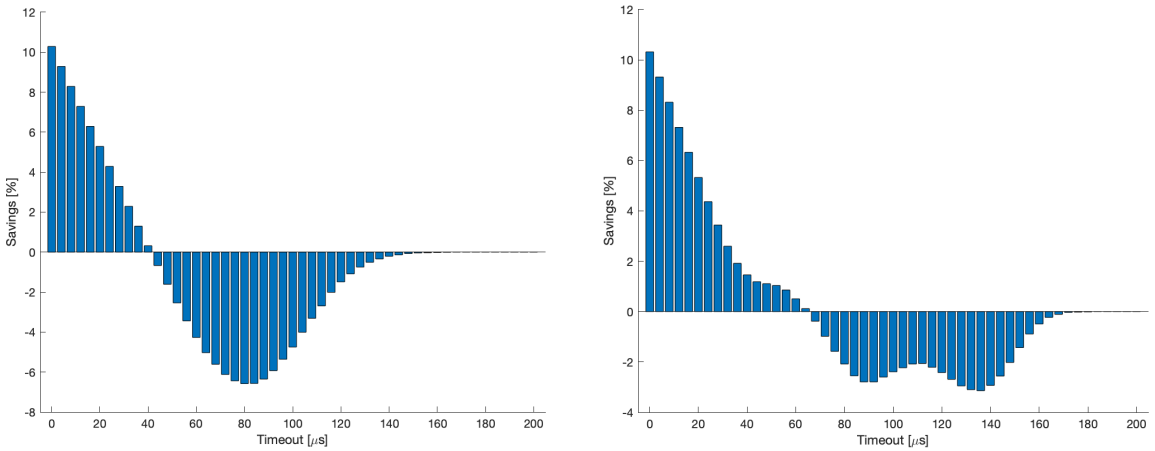


Figure 2.3: Overall savings resultant from the Exponential Distribution.

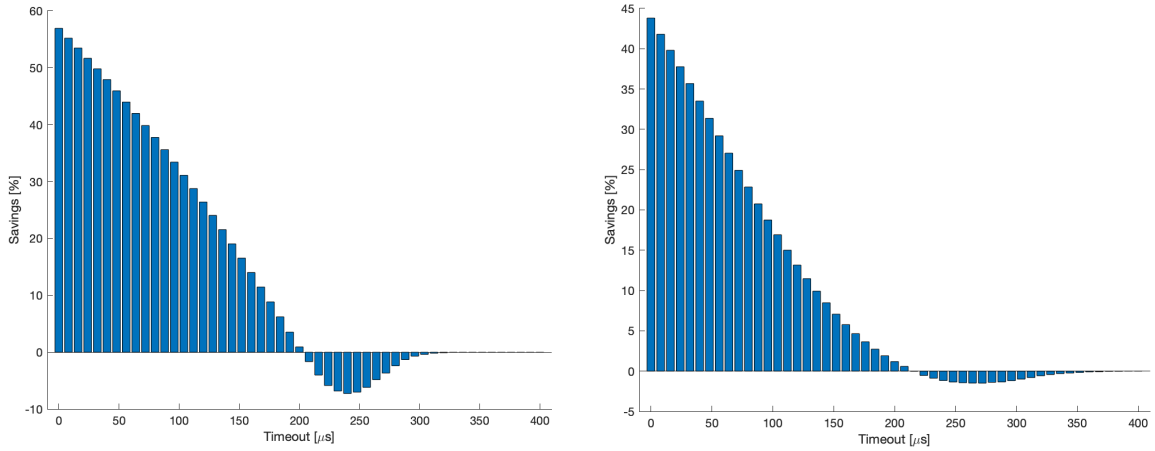
Moreover, large timeout values can be non-beneficial also. In this case, when the system finally transitions to a low-power state the amount of savings achieved is not sufficient to compensate the power overhead, as demonstrated by the results obtained with the normal and tri-modal distributions depicted in Fig. 2.4



(a) Normal **(b)** Tri-modal

Figure 2.4: Savings achieved with different Normal Distributions.

Finally, the assignment required evaluating two given profiles whose distribution used when generating the active and idle periods is unknown. At each iteration the timeout variable was incremented by a step of $8\mu s$ up to $400\mu s$. The results were similar to those observed in the uniform distribution with low utilization, characterizing longer idle periods, hence, better performance in terms of savings.



(a) Custom Profile 1 **(b)** Custom Profile 2

Figure 2.5: Histograms achieved with the provided workload profiles.

2.2. 3-State PSM

The second set of simulations included the *Sleep* state. For simplicity, the DPM simulator performs a transition from Idle directly to Sleep, as shown in Fig. 2.6. The delay in this case is the sum of the contributions of passing through Run before reaching Sleep. In fact, the code presented at the beginning of this section already implemented these transitions.

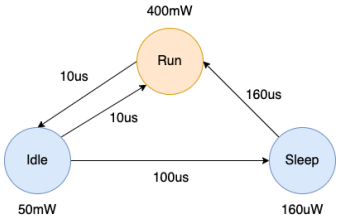
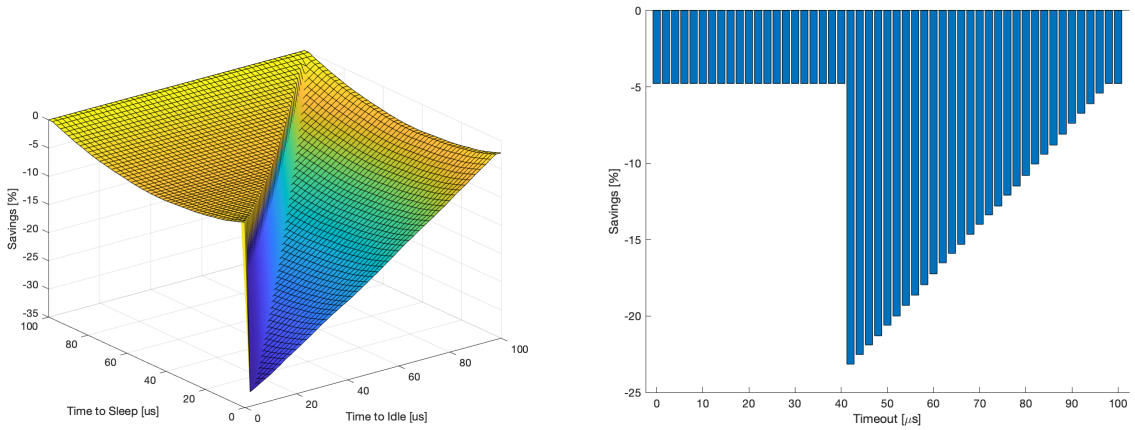


Figure 2.6: PSM as it is seen by the DPM simulator.

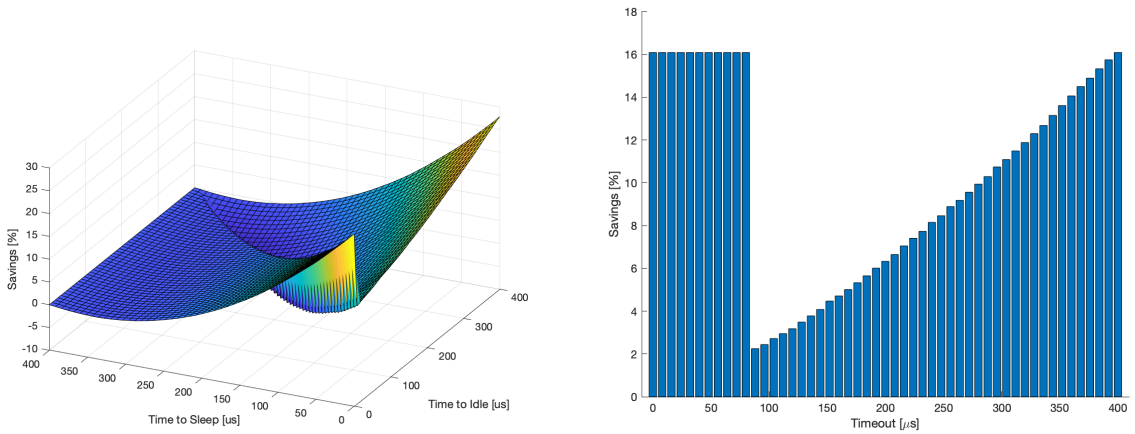
2.2.1 Results

The bash script was simply modified to run a second loop that increments the second timeout by a certain step as well, for a total of 250 samples. Figure 2.7 presents a surface plot associating the overall savings to both time to Idle and time to Sleep, as well as a cross-section of this plot highlighting the results at a fixed value.



(a) High Utilization **(b)** Time to Idle fixed at $40\mu s$

Figure 2.7: a) depicts the surface plot for the High Utilization profile; b) details the cross-section with time to Idle fixed at $40\mu s$ while increasing time to Sleep.

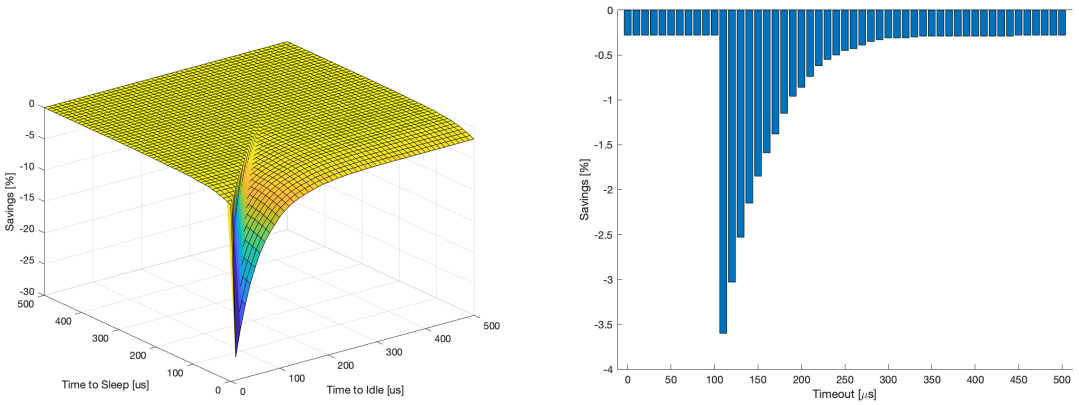


(a) Low Utilization **(b)** Time to Idle fixed at $80\mu s$

Figure 2.8: Savings obtained with the Low Utilization profile.

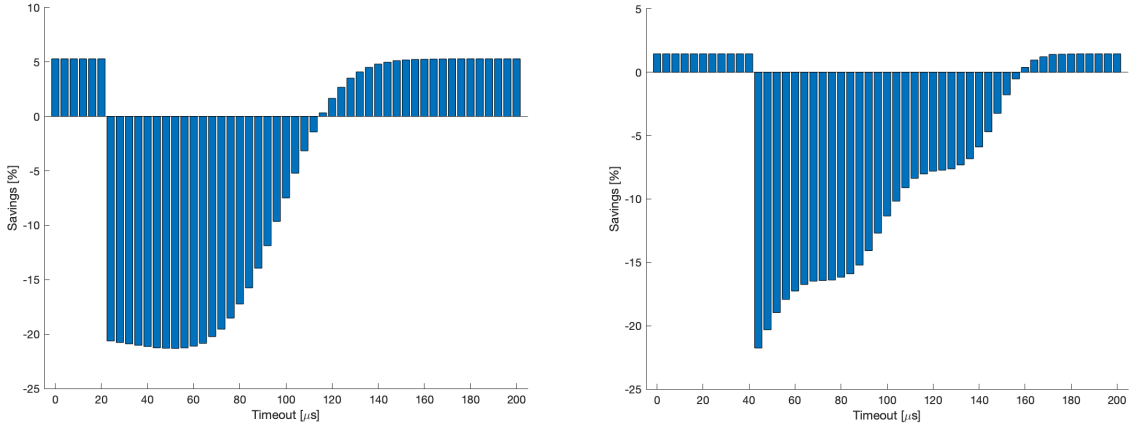
Notice how the savings worsen when the time to Idle is surpassed by the time to Sleep, since the latter would be never reached in this way. Again, the Low Utilization profile had a better performance as in this case the Sleep state can be reached without penalties, as shown in 2.8.

The transition overhead is best exemplified by the results obtained with the exponential distribution, shown in Fig. 2.9. As the timeout increases the system makes less transitions, thus, the power consumption penalty decreases. Figure 2.10 illustrates the results with both Gaussian distributions. Similarly, a third low-power state did not improve the overall energy efficiency.



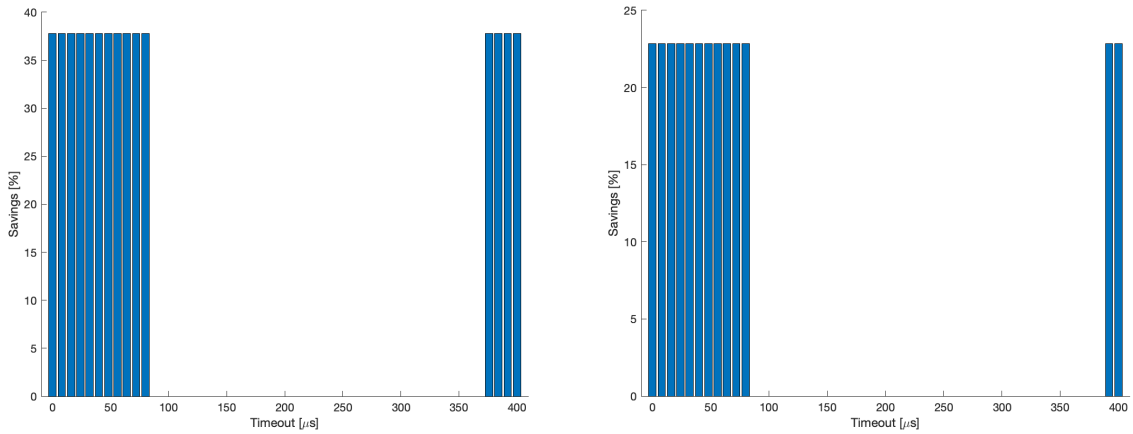
(a) Exponential **(b)** Time to Idle fixed at $100\mu s$

Figure 2.9: Surface plot and cross-section regarding the exponential distribution.



(a) Normal **(b)** Tri-modal

Figure 2.10: Gaussian distributions. Time to Idle fixed at $20\mu s$ in a) and $40\mu s$ in b).



(a) Custom Profile 1 **(b)** Custom Profile 2

Figure 2.11: Results obtained with time to Idle fixed at $80\mu s$ and time to Sleep increasing.

Finally, the results obtained with the provided custom workloads are shown in Fig. 2.11. Note how the achieved savings dramatically worsen with this PSM, in contrast with those results presented by the previous experiments. Table 2.1 gives the average time spent at each state. Notice that these profiles spent less time at the *Run* state, while *Sleep* is almost never reached, explaining their poor performance.

Distribution	Run [μs]	Idle [μs]	Sleep [μs]
High Utilization	1.42386	0.06264	0.01912
Low Utilization	1.92502	0.25779	0.08019
Normal	1.62243	0.10674	0.02132
Exponential	1.47611	0.02379	0.00221
Tri-modal	1.60592	0.11578	0.02956
Custom 1	1.06797	0.18238	0.00002
Custom 2	1.14409	0.11029	0.00146

Table 2.1: Average time spent at each state.

3. History Policy

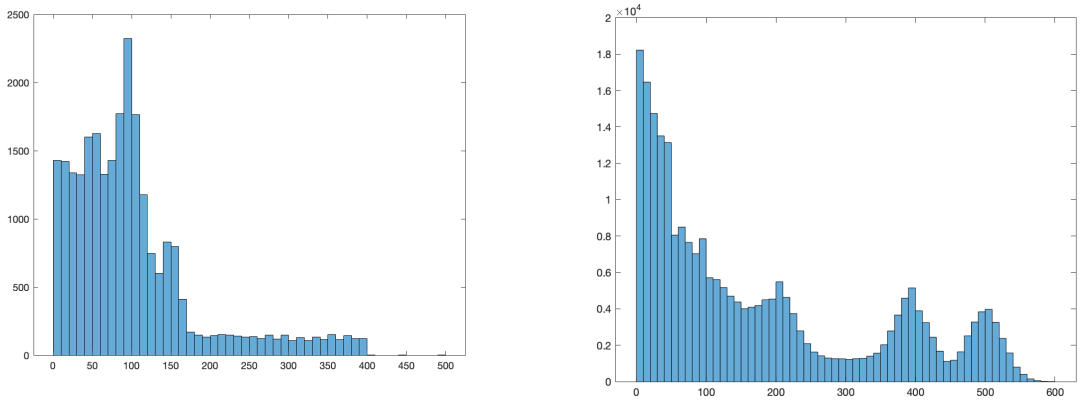
The History Policy is a predictive policy, i.e., it estimates the time to transition based on the previous workload pattern. The computed value T_{pred} is compared to the time to Idle and Sleep thresholds. After a decision is taken the system is able to reach any one of the low-power states independently.

```
1        if(curr_time < idle_period.start) {
2            *next_state = PSM_STATE_ACTIVE;
3        } else {
4            *next_state = PSM_STATE_ACTIVE;
5            value_prediction = hparams.alpha[0] * pow(history[2], 2) + hparams.
     alpha[1] * history[1] + hparams.alpha[2];
6            if (value_prediction >= (double)hparams.threshold[0])
7                *next_state = PSM_STATE_IDLE;
8            if ((value_prediction >= (double)hparams.threshold[0]) && (
     value_prediction >= (double)hparams.threshold[1]) && (hparams.threshold[1] >
     hparams.threshold[0]))
9                *next_state = PSM_STATE_SLEEP;
10       }
11
```

The assignment required implementing the logic behind this policy as well. The piece of code above demonstrates how T_{pred} is computed and the decision-making that assigns the next state. The prediction might not represent the next actual idle period, imposing a penalty. Otherwise, transition occurs immediately and no power is wasted by waiting for a certain timeout to expire.

$$T_{pred}(x) = K_1 T_{idle}[i-1]^2 + K_2 T_{idle}[i-2] + K_3 \tag{3.1}$$

The double degree polynomial expressed in equation 3.1 was used to predict the length of the idle periods. Polynomials of different degrees were considered, however, the data-set used in the regression had a great impact in the final performance than the polynomial degree itself. In Fig. 3.1 two training sets are depicted. The one in a) was based on the profiles described in Section 1, after randomly sorting them out. In b) the samples were arbitrary, somewhat complying with the overall behavior of the workload.



(a) Training data-set 1 **(b)** Training data-set 2

Figure 3.1: a) the data-set based on the generated workloads; b) illustrates the randomly generated data-set.

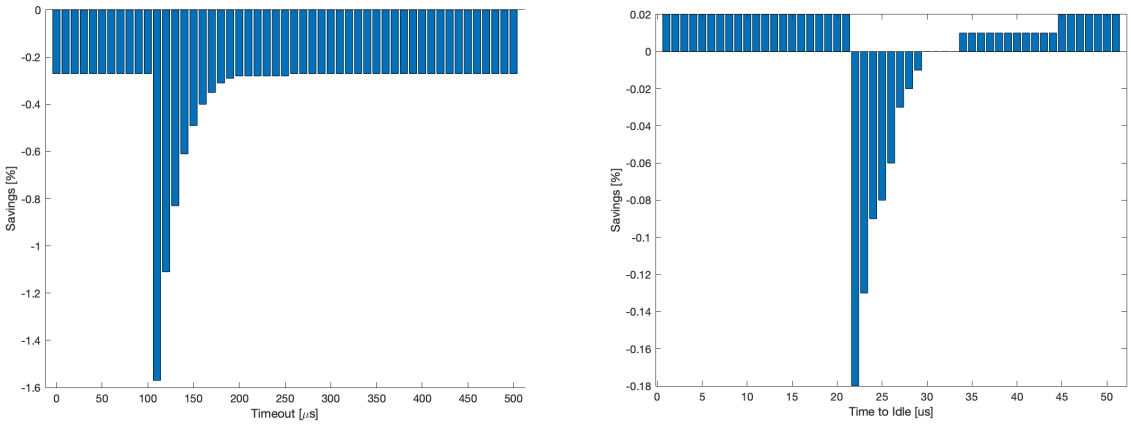
Table 3.1 presents the coefficients obtained with each data-set. The error margin acts as an indicator whether the polynomial function best fits the data or not. Although the first training set was closer to the idle periods within the system, the second randomly generated set presented a better performance.

Data-set	K1	K2	K2	Error
Training 1	-4.42185232119070e-05	0.726771518733501	27.9968345531671	74.55
Training 2	4.21975932546161e-05	0.962808289733700	4.13829772112614	29.75

Table 3.1: Average time spent at each state.

3.1. Results

The History Policy also favors workloads with longer idleness. However, it must be highlighted that all profiles were submitted to the same prediction scheme. Perhaps a different training set would best suit those workloads with a higher utilization rate.



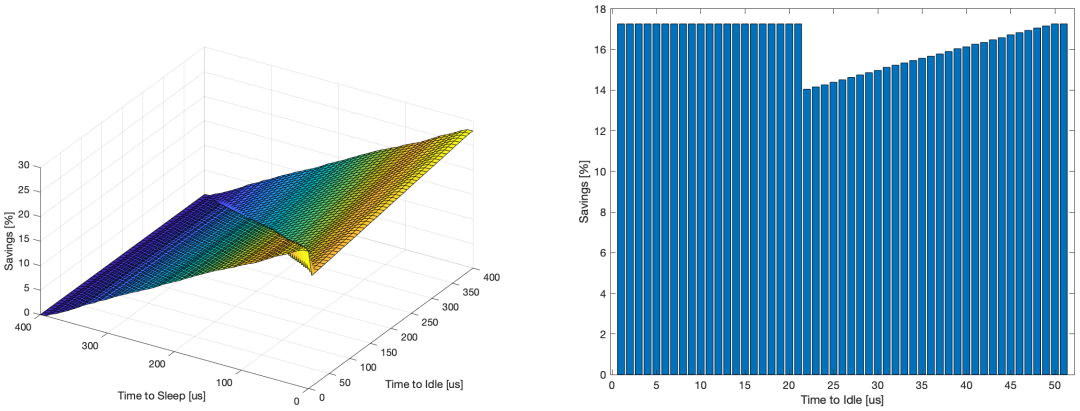
(a) Training data-set 1 **(b)** Training data-set 2

Figure 3.2: Exponential profile with first threshold fixed at $80\mu s$, increasing the second threshold by a $10\mu s$ step.

Figure 3.2 illustrates the differences in savings from one set to another. Notice that in b) some positive compensation was achieved for the exponential profile, even if rather small. In fact, all the other profiles had a better performance when using the coefficients from the second training set.

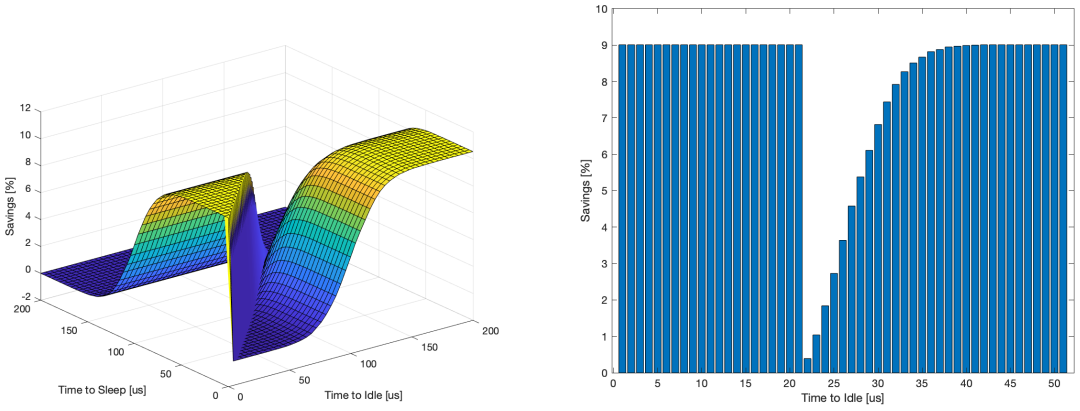
This demonstrates that a good predictor does not rely on the exact workload behavior, profiting also from uncorrelated samples.

Moreover, the savings achieved with this policy not only were stable throughout the execution, as it was superior to the previous experiments. This can be seen in Fig. 3.3 that depicts the same analysis as done before for the uniform low utilization profile, as well as Fig. 3.4 for the Normal profile. For simplicity, other distributions were not include in this report, but they can be found inside the folder under the name *"results"*.



(a) Low Utilization **(b)** First threshold fixed at $80\mu s$

Figure 3.3: Low Utilization profile under training set 2.



(a) Low Utilization **(b)** First threshold fixed at $20\mu s$

Figure 3.4: Normal distribution profile under training set 2.

4. Conclusion

The experiments have shown that the History Policy prevails over the Timeout Policy. Not only the results improved from one policy to another, as a certain regularity can be observed on the results. However, this kind of predictive policy requires additional computations that could grow in complexity. The simulator does not take into account this kind of overhead, such as timing and area dedicated to this task. For this reason, the application requirements must be considered when choosing which policy to implement. Finally, both policies involve some previous analysis regarding the timing thresholds, whose dependency on the workload profile has been demonstrated.

Bibliography

[1] A. Sinha and A. P. Chandrakasan. «Dynamic voltage scheduling using adaptive filtering of workload traces». In: *VLSI Design 2001. Fourteenth International Conference on VLSI Design*. 2001, pp. 221–226. DOI: 10.1109/ICVD.2001.902664.

[2] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. «Scheduling for reduced CPU energy». In: *Mobile Computing*. Springer, 1994, pp. 449–471.

[3] L. Benini, A. Bogliolo, and G. De Micheli. «A survey of design techniques for system-level dynamic power management». In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 8.3 (2000), pp. 299–316. DOI: 10.1109/92.845896.