Laboratory 1 Report

Dynamic Power Management

Michael Elias, Juan José Restrepo

Energy management for IoT

2024

Contents

1 Introduction

The main focus of this lab was to understand the basics of Dynamic Power Management. Thus, it is based on evaluating multiple policies to control the transitions of the power unit.

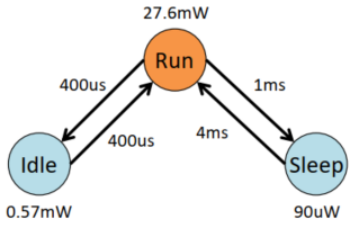

Figure 1: Power State Machine

Figure 1 shows the power state machine of the lab. Each state corresponds to an operation mode and each one has an associated value of power consumption and the delay caused by doing a transition from one state to another one.

In particular, we have three states: *Run*, *Idle* and *Sleep*. Moreover, the idea of Dynamic Power Management is to decrease energy consumption by putting the devices in a low-power state when peak performance is not required. Therefore, as it is shown in Figure 1, we have four possible transitions: *Run* - > *Idle*, *Run* - > *Sleep*, *Idle* - > *Run* and *Sleep* - > *Run*. One can notice that there is no transition between *Idle* and *Sleep*.

The main purpose of each state could be described as follows:

a) *Run*: Device is operational at different f and V

b) *Idle*: The CPU is stopped when it is not executing anything; however, it monitors interrupts

c) *Sleep*: Shutdown of on-chip activity

The goal of Dynamic Power Management is to determine the optimal allocation of power states over time given a PSM and a workload such that power is minimized. With this purpose in mind, it is of paramount importance to make a transition from one state to another only if it is convenient(when idleness is going to be long enough). Therefore, predicting idle times is essential because wrong predictions cause energy waste.

There are two types of mispredictions:

a) Overprediction: the predicted idle value is longer than the actual one.

b) Underprediction: the predicted idle value is shorter than the actual one.

In the next sections, two policies, whose purpose is to minimize the number of mispredictions, are analyzed: *Timeout* policy and *History* policy.

2 Timeout policy (Part I)

2.1 Description

The technique is based on setting a fixed time such that after this time the device makes the transition from ON state to OFF state. In other words, the idea is to put the device into OFF state *Tto* time units after it has entered the inactive state.

Then, it is important to introduce a term: *System break-even time(Tbe)*. The *Tbe* corresponds to the minimum idle time for amortizing the cost of shutdown; thus, if the period of idle is smaller than *Tbe*, there is not enough time to enter the low-power state.

In particular, the goal of this policy is to choose the proper value of *Tto* based on the principle that when $Tidle > To$ it is likely that $Tidle > To + Tbe$ so, $Prob(Tidle > To + Tbe|Tidle > To) = 1$. This affirms that if a system has been idle for *Tto*, it is mostly probable that it will continue to be idle for $Tidle > To + Tbe$

There are two important transitions to analyze: *Run - > Idle*, *Run - > Sleep*. The purpose of this section is to analyze the first case (*Run - > Sleep* will be analyzed in the next section).

2.2 Bash script

In order to run the simulator in a systematic way, a script called "timeout.sh" was created. This script asks for 5 parameters:

a) Workload: Choose the workload to analyze (In our example we have 2 options).

b) PSM: Choose the psm (In our example there is only 1).

c) Transition: Select the kind of transition: 0(Run - > IDLE) or 1(Run - > Sleep)

d) Initial TO value: This is the first TO used to run the dpm.

e) Final TO value: Starting from TO = initial value given previously, TO is incremented by 1 and then used for the dpm. This is done up until TO is equal to the final TO value given in the parameter.

f) Final output: Select the place where to store/save the output of the script: 0(Terminal) or 1(Files)

An example of running the script with timeout policy is given in Figure 2 (running history policy is shown later on).

NOTE 1: In case you get a "-bash: ./timeout.sh: bin/bash: bad interpreter: No such file or directory" error when running the script, try this approach "bash ./script.sh".
NOTE 2: In case you get an error along the lines of "$'\r': command not found" run the following command: dos2unix <name of script >.sh.

```
Select the workload to analyze: workload_1
workload_1.txt will be used for simulation
Select the file that describes the PSM of the resource: psm
psm
Select the kind of transition: 0(Run -> IDLE), 1(Run -> Sleep): 0
0
Enter the initial TO for the simulation(unit is in ms): 0
0
Enter the final TO for the simulation(unit is in ms): 50
50
psm.txt will be used for simulation
```

Figure 2: Script example for running timeout policy

The energy consumption for each *Tto* and the *Tto* values are saved in a directory "results/using_TO".

2.3 MATLAB

Using the file workload_1.txt, workload_2.txt and the values generated by the bash script, we were able to write a MATLAB script that plots Figure 3, Figure 4 and Figure 5 (a similar graph to Figure 3 was plotted for workload_2 but it is not shown in this report).

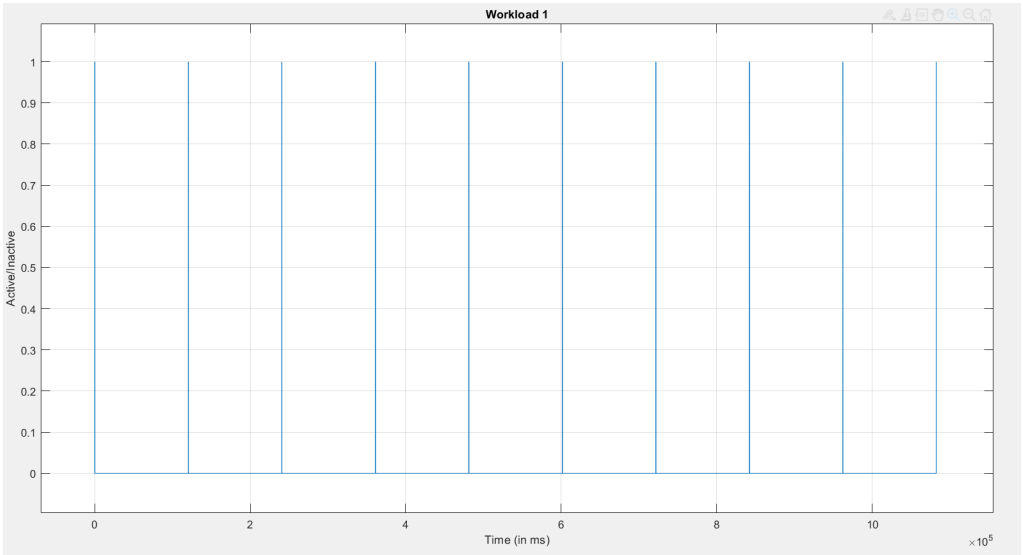

Figure 3: Workload 1

Figure 3 simply shows workload 1, although not much can be understood when it's zoomed out. A closeup image can be seen in Figure 4 where we can observe the workload moving back and forth from active to inactive.
Figure 5 is what really interests us here, as it shows the variation of the energy dissipated for a range of *Tto* values (from 0 to 50 in this example). The lowest energy consumption can be seen for $Tto = 0$ ms, and then it increases as *Tto* increases. Therefore it is obvious that the best choice for *Tto* for this workload is 0.

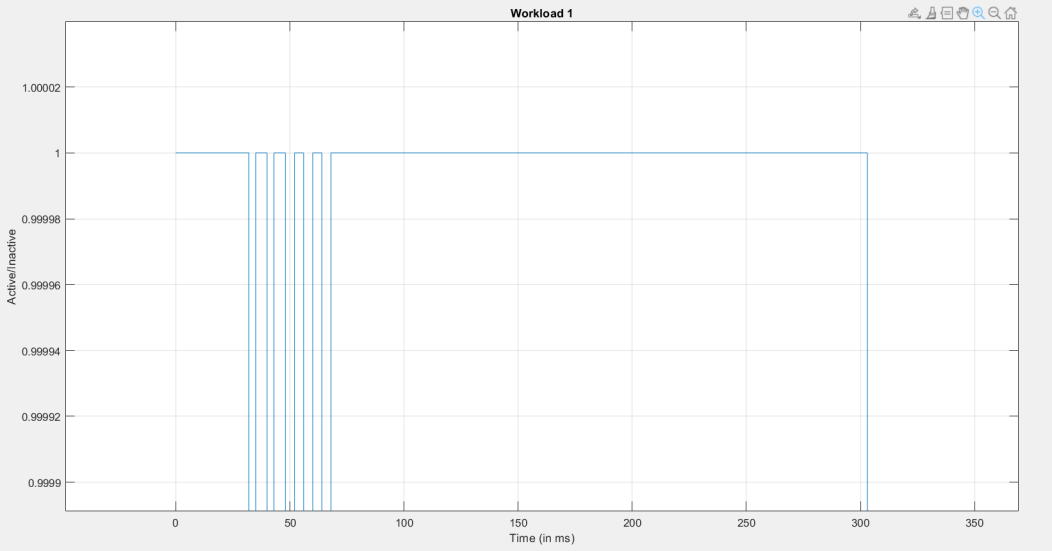

Figure 4: Workload 1 closeup

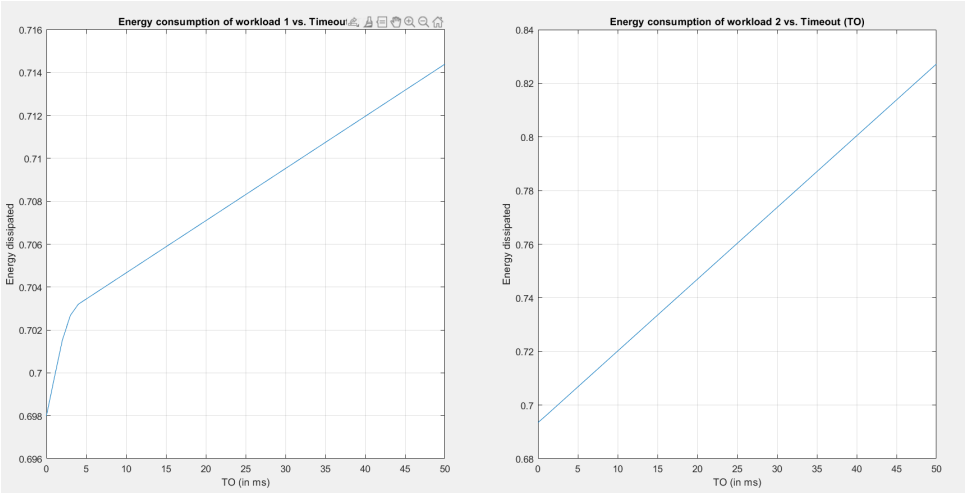

Figure 5: Energy dissipated vs Tto when transition: Run - > Idle

2.4 Analysis

Workload 1: At the beginning, the transition *Run - > Idle* was considered. One can easily compute T_{be} using the formula $T_{be} = T_{tr} + T_{tr}*(P_{tr} - P_{on})/(P_{on} - T_{off})$, by plugging in the numbers we get that T_{be} is equal to 0.723 ms. Consequently, it is possible to conclude that the best thing to do is to have *Tto* equal to 0 to immediately make the transition to *Idle* and save more energy as it can be seen in Figure 6. This could be corroborated by considering both the *Tbe*(equal to 0.723 ms) and that the minimum inactive time is 3ms in the sensing period. Thus, the inactive time is guaranteed to be higher than the *Tbe*.

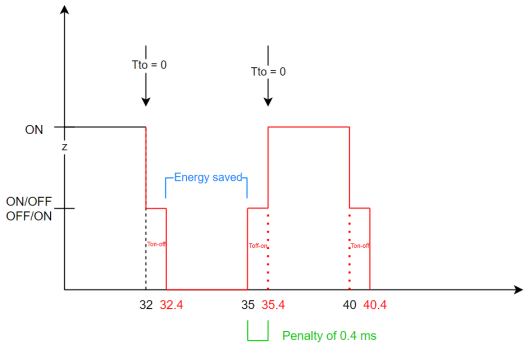

Figure 6: Analysis gap from 32ms to 35ms when *Tto* is 0

Workload 2: In workload 2, the inactive time during the sensing period is even more than in workload 1 (around 10 ms). Therefore, the fact that the best choice for *Tto* is 0 ms can be explained using the same reasoning.

3 Timeout policy (Part II)

3.1 Description

This section considers the same timeout policy. However, in this case, the transition *Run - > Sleep* is analyzed.

3.2 Bash script

The same bash script is used as in the previous section. One has to choose the *Run - > Sleep* in the terminal.

3.3 MATLAB

Figure 7 shows the results obtained for both workloads. On the one hand, it is possible to see that the best option for *Tto* is 4 ms in the case of the first workload. On the other hand, *Tto* is 0 is the best choice for the second workload. The reasons for obtaining these values are well explained in the following section.

Finally, one can calculate the T_{be} with the same formula shown before. In this case, the T_{be} is equal to 73.41 ms.

3.4 Analysis

Workload 1: *Tbe* for *Run - > Sleep* is 73.41 ms, and we already know that the inactive time during the sensing period is around 3 ms. Therefore, to get the best power savings, the system should not go to sleep during the sensing period as *Tbe* > inactive time. This explains why the best value for *Tto* is 4 ms as it guarantees that the system never goes to sleep during the sensing period (*Tto* > 3 ms).

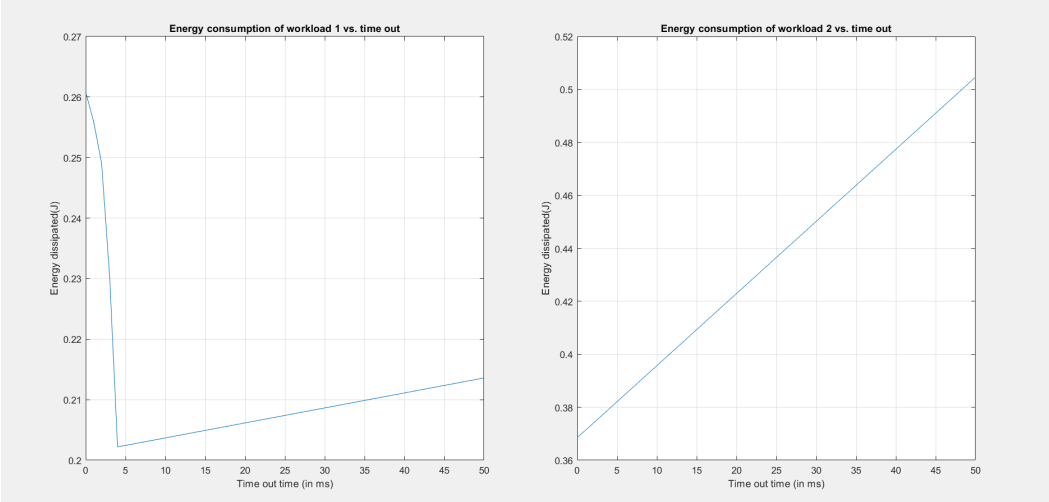

Figure 7: Energy dissipated vs Tto when transition: Run - > Sleep

Workload 2: This is a different case with respect to the first workload. In particular, the inactive time is around 85ms which means it is greater than the *Tbe*; hence, going to the sleep state is the proper solution. Like, in the *Run - > Idle* case, the sooner the system goes to sleep state, the better. Then, it is reasonable to use $Tto = 0$.

3.5 Comparison between *Run - > Idle* and *Run - > Sleep*

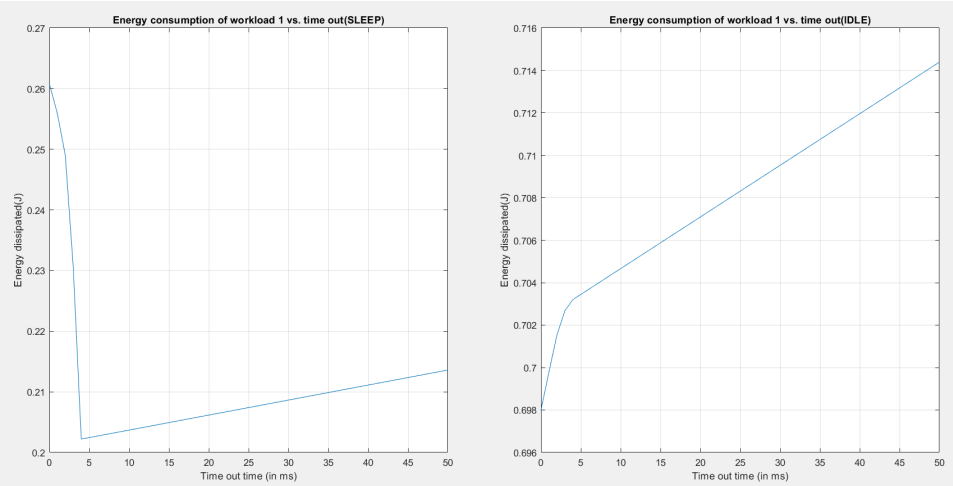

Figure 8: Comparison between Run - > Sleep and Run - > Idle for workload 1

Figure 8 shows the comparison between the two transitions for workload 1. It is obvious that we get better energy savings when the system makes the transition from Run state to Sleep state, with a minimum of 0.202 J, compared to the transition from Run to Idle, with a minimum value of 0.697 J. Even though the system does not go the sleep during the sensing period, missing out on some savings compared to idle transition, we compensate for this loss in the "waiting" period because we save more energy in the sleep state compared to the idle

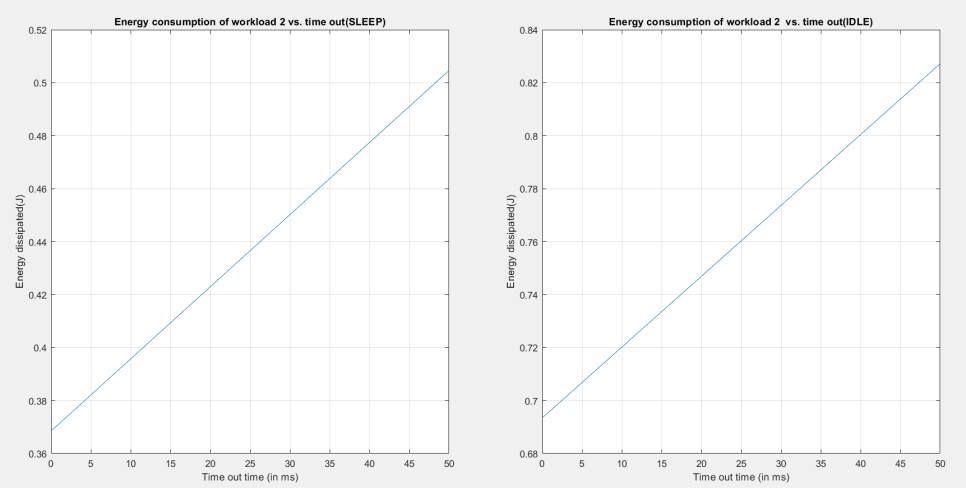

Figure 9: Comparison between Run - > Sleep and Run - > Idle for workload 2

state.

Figure 9 shows the comparison between the two transitions for workload 2. As before, the system gets better power savings when it makes the transition from Run state to Sleep state, with a minimum of 0.368 J, compared to the transition from Run to Idle, with a minimum value of 0.693 J. Once more, it is demonstrated that it is better to make the transition from Run - > Sleep in the "waiting" period.

4 History policy (Part III)

4.1 Bash script

Another bash script was created to run the history-based policy, an example can be seen in Figure 10. There are 4 inputs to this script, workload and psm work the same way as explained earlier. The difference here is that there are 2 thresholds in the history policy, therefore the script requires 2 inputs (each input is 2 integer values separated by a space) to set the ranges for these thresholds.
Keep in mind that due to having 2 threshold values for the history policy, 2 nested loops are used here, 1 loop to iterate over threshold 1 and the second to iterate over threshold 2. Additionally, in each iteration, the simulator is called 3 times (for plotting purposes). Therefore the ranges for the thresholds should be chosen carefully otherwise the simulation will take a long time.

4.2 History policy

Looking at workloads 1 and 2, it can be seen that the best history policy is measuring the length of the previous active phase, then simply choosing T_{pred} to be equal to this length, in other words: T_{pred} = length of last active phase.
Now T_{pred} can be used to choose the next state, the policy used goes like this (pseudo-code):

```
michael@MSI:/mnt/c/Users/Eden/OneDrive/Desktop/IoT/Labs/Lab1$ bash ./history.sh
Select the workload to analyze: workload_1
workload_1.txt will be used for simulation
Select the file that describes the PSM of the resource: psm
psm.txt will be used for simulation
Enter the initial and final value for the first threshold (unit is in ms): 0 300
Threshold 1 starts at 0 and ends at 300, with an increment of 10
Enter the initial and final value for the second threshold (unit is in ms): 0 300
Threshold 2 starts at 0 and ends at 300, with an increment of 10
```

Figure 10: Script example for running history policy

```
$T_p_r_e_d$ = length of last active phase;
if ($T_p_r_e_d$ <= Threshold 1)
{
    *next_state = PSM_STATE_RUN;
}
else if ($T_p_r_e_d$ <= Threshold 2)
{
    *next_state = PSM_STATE_IDLE;
}
else
{
    *next_state = PSM_STATE_SLEEP;
}
```

This code can be explained as such: If the predicted inactive time is too short then stay in run state, if it is a bit longer then go to idle, if it's predicted to be too long, go to sleep.

4.3 MATLAB

The previous MATLAB script was modified to also plot the results generated by the history policy. The new plots are shown in Figure 11, Figure 12, Figure 13.

Figure 11 shows the energy consumption when both Threshold 1 and Threshold 2 are varying, X represents Threshold 1, Y represents Threshold 2 and Z represents energy consumption. But this figure is hard to read as it is plotted in 3D. Therefore we had to plot the energy consumption when one of the thresholds is fixed and the other varies, which is what we did in Figure 12.

Based on Figure 12 we could find the ranges for the thresholds that produce the best power savings for workload 1: Threshold 1 should be 0 or 1 and Threshold 2 belongs to [40:180]. When the thresholds satisfy these conditions, we get the best power savings, where consumption is 0.1973 Joules.

For workload 2, we can look at Figure 13. Doing the same analysis we can find the thresholds as follows: Threshold 1 is 0 and Threshold 2 belongs to [10:220]. For this values power consumption is 0.2197 Joules.

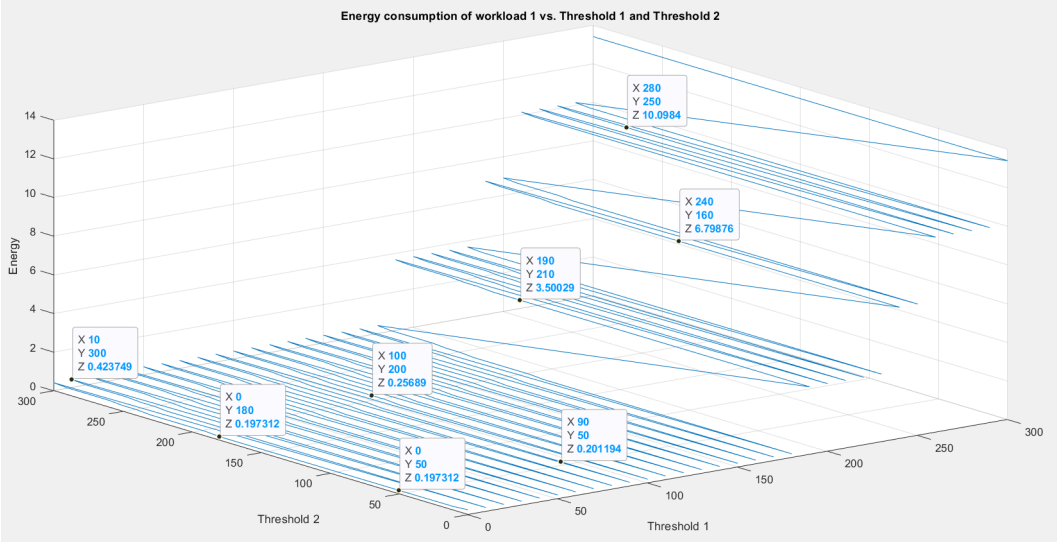

Figure 11: Energy consumption of workload 1 vs. Threshold 1 and 2

4.4 Analysis

Workload 1: We already know that T_{be} for idle is equal to 0.723 ms and T_{be} for sleep is 73.41 ms. So it is obvious that the best solution is to go to idle state during the "sensing" period and to go to sleep during the "wait" period.
In order to go idle during the sensing period, T_{pred} should be strictly greater than Threshold 1. Knowing that T_{pred} = last active, we see that T_{pred} is almost always greater than 1. Therefore if Threshold 1 is 0 or 1, the condition that T_{pred} > Threshold 1 is always satisfied. This explains why to get the best saving Threshold 1 should be 0 or 1, any other value will lead to staying in run state thus more energy consumption.
Now for the "wait" period, we should go to sleep state, therefore T_{pred} should be strictly greater than Threshold 2. We know that T_{pred} = last active, but the active time right before the "wait" period is simply the "compute" period. Here is the values for all "compute" periods in workload 1: 235, 255, 181, 328, 318, 293, 319, 325, 327 (all in ms), these values can also be seen as T_{pred}. As a result, to satisfy the condition T_{pred} > Threshold2, Threshold 2 should have a value at a maximum equal to the minimum value in the previous list, which is 181 ms. This explains the upper bound for Threshold 2 found earlier.
The lower bound for Threshold 2 can be explained as follows: During the "sensing" period, we should never transition to "sleep" state as the inactive time is way shorter than $T_{BEsleep}$. So we should choose Threshold 2 in such a way to guarantee that T_{pred} <= Threshold 2 during the sensing period. The largest active time during "sensing" period was found to be 32 ms, therefore the largest value T_{pred} can have is 32 and Threshold 2 has to be greater than that. That is where the lower bound of 40 ns for Threshold 2 comes from.

Workload 2: Workload 2 goes through the same periods as we have seen for workload 1, "sensing", "compute" and "wait". Therefore the same analysis can be applied here but we will get different ranges for Threshold 1 and Threshold 2 as this workload has different

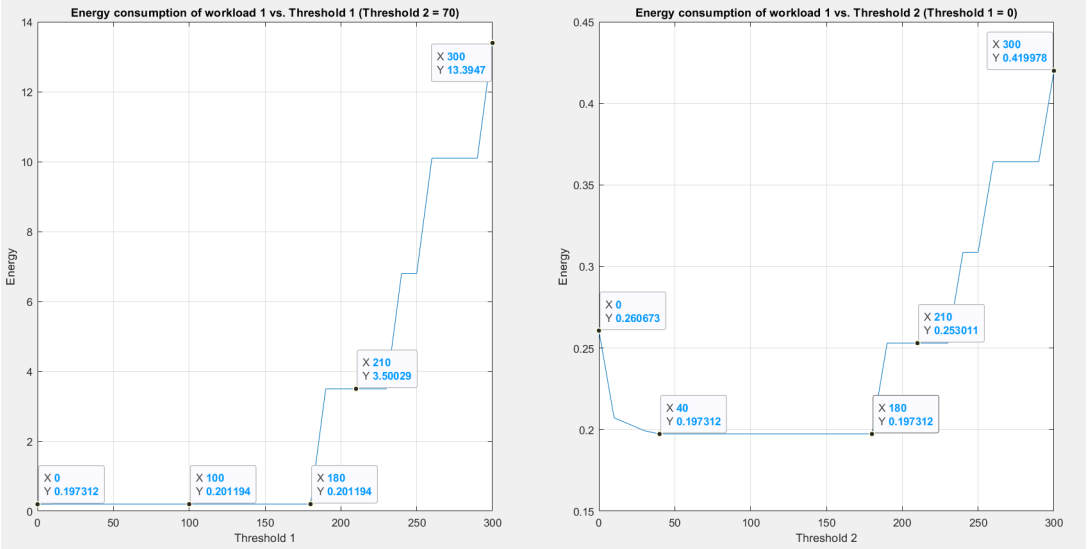

Figure 12: Energy consumption of workload 1 vs. Threshold 1 and workload 1 vs. Threshold 2

active and inactive time lengths.

5 Comparing History and Timeout policies

The summary for the results obtained in this lab can be seen in Table 1. Based on this table, the policies can be ranked as follows (best to worst): History, Timeout (with sleep) and Timeout (with idle). The comparison between the two Timeout policies was done in section 3.5. Then, here we are only comparing history policy and timeout policy. The fact that the history-based policy resulted in better energy consumption as opposed to timeout timeout-based policy can be explained by 2 main reasons:

a) Time penalty: For timeout policy, when workload goes to inactive, the system has to wait for a time equal to *Tto* before deciding whether to change power state or not. This "wait" is the wasted time that the system could have been used to switch to a low-power state and save energy. This time penalty doesn't exist in the history policy as the decision is taken immediately when the workload goes inactive. This applies when *Tto* is different from zero, in case $Tto = 0$ ms this penalty is absent for both policies.

b) History policy is "smarter": As the history policy is based on T_{pred} which depends on the active and inactive time of the workload, we can say that the history policy adapts to the workload and is more dynamic compared to the timeout policy where *Tto* is fixed for the entire duration of the workload. Another reason would be that the history policy includes three states: *run*, *idle* and *sleep*; hence, the system can go to idle during the sensing period and sleep during the waiting period, which maximizes the energy savings. However, for the timeout policy, the low-power state is either idle or sleep, never both at the same time.

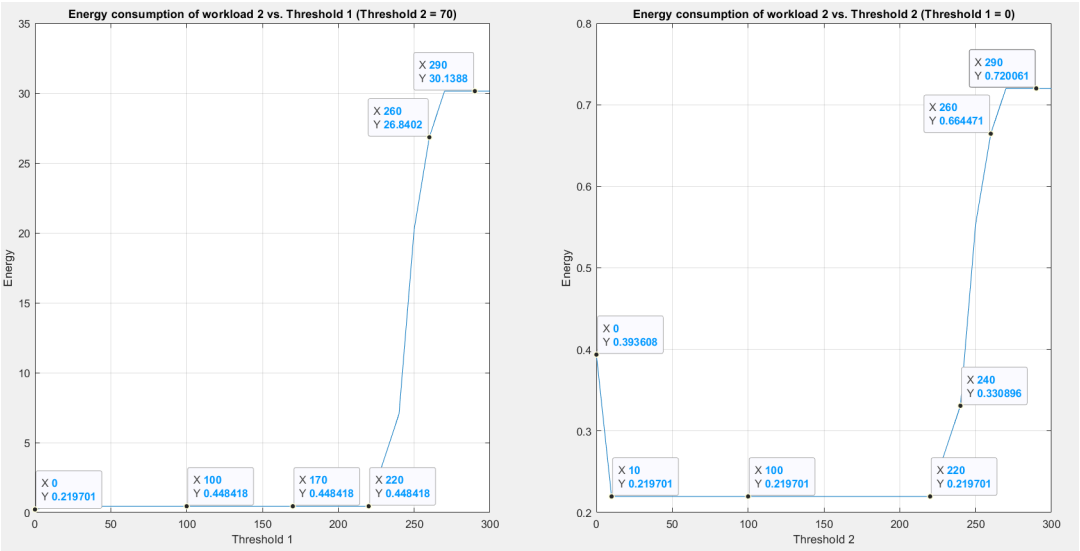

Figure 13: Energy consumption of workload 2 vs. Threshold 1 and workload 1 vs. Threshold 2

Workload	**Timeout (with idle)**	**Timeout (with sleep)**	**History**
Workload 1	0.697 J	0.202 J	0.197 J
Workload 2	0.693 J	0.368 J	0.219 J

Table 1: Summary of the power consumption