Lehrstuhl für Rechnerarchitektur und Parallele Systeme
TUM School of Computation, Information and Technology
Technical University of Munich

Power Tuning

Group 8

Lars Leimbach ✉ **and Mehriniso Mangliyeva**
TUM School of Computation, Information and Technology, Technical University of Munich
✉ lars.leimbach@tum.de

July 16, 2023

1 Measurements

All five to ten measurements for each task can be found in the appendix.

1.1 Startup Phase

160MHz + ALS	80MHz	160MHz	240MHz	DFS(80,240)	DFS(80,240) + ALS
972,2	897,3	1051,7	1144	1084,7	902,8

Table 1 The average mAs for the Startup phase.

`startMeasurment()` is called after the initialization of the pins and `stopMeasurment()` is called when every internet functionality is initialized. To restart the measurement, we disconnected the power supply of the ESP. The normal connecting time to Wifi was 5 seconds. In the university, we would measure different mAs because it takes longer to connect. DFS(80,240) showed the best performance (see table 1) because it might have faster calculation than 80MHz due to dynamic frequency scaling.

1.2 Wifi Savings

To indicate Internet functionalities were initialized correctly, we give a semaphore. After the semaphore can be taken, the ESP starts the measurement and waits 10 seconds in a `vTaskDelay()`. We can see that the bigger the beacon interval is the smaller the power consumption gets (see table 2). However, I did not test the drawbacks of a higher beacon interval. In addition, here we implemented an automatic restart with `esp_restart()` after the measurements.

No improvments	Interval 2	Interval 4	Interval 16	Interval 128	Interval 256
758,2	751,7	301,5	159,7	106,6	100,1

Table 2 The average mAs in Wifi savings for a corresponding beacon interval of 1, 2, 4, 16, 128, and 256.

1.3 Calculation Phase

Internet- and display functionalities were switched off by a `#define`. In addition, we had to insert a `vTaskDelay(10)` in the calculation `for`-loop. Because the variance of the temporal execution and power consumption was not high, we only performed five measurements instead of ten. However, we see that again DFS(80,240) + ALS performs best. Even if 240MHz was a bit faster, it had a higher energy consumption. We point out, that even if the slower frequency had a slower execution, the power consumption of all frequencies are similar(as we see in table 3).

80MHz	Time 80MHz	160MHz	Time 160MHz	240MHz	Time 240MHz	160MHz + ALS	Time 160MHz + ALS	DFS(80,240) + ALS	Time DFS(80,240) + ALS
1520	56	1153,2	37	1039,4	31	1158,4	37	918,2	33

Table 3 The average mAs and seconds during the calculation phase. Seconds are marked with the column name "Time".

1.4 Parallelization of Calculation Phase

One of the most effective improvements of the calculation code is to split equally the most outer loop to the two cores. Such that core 1 executes the t from 0 to 7 and core 2 executes t from 8 to 15. That reduces the execution time by a factor of 0.5 (see table 4). Of course, there are no huge energy savings because we are using two cores at the same time.

80 MHz	Time 80 MHz	160 MHz	Time 160 MHz	240 MHz	Time 240 MHz
986,4	28	816,4	19	758,2	16

Table 4 The average mAs and seconds during the paralleled calculation phase. Seconds are marked with the column name "Time".

1.5 Light and Deep Sleep

We use DFS(80,240) + ALS with Wifi savings and a beacon interval of 128. So, before entering for 10 seconds each sleep mode, internet will be activated. We see that there is a higher energy consumption with activated display and deep sleep consumed way less energy than light sleep, see table 5. Deep sleep consumed 0.1 mAs for the 10 seconds without the display and 1 mAs with the display. There is that inaccuracy in the table 5 because the measurement is stopped after booting the ESP. So, it reveals how much energy is used to wake up form deep sleep too.

Light no display	Light with display	Deep no display	Deep with display
30,2	51,6	3	5

Table 5 The average mAs for the sleep modes.

1.6 Code explanations

The overall idea is to stay in deep sleep as long as possible and add there events to the buffer. We never tested how long the ESP might stay in deep sleep, such that it wakes up after `WAKEUP_AFTER` seconds. Currently, it is limited to 20 minutes. Before sleeping 20 minutes after the first boot, it initializes internet parallel to activity receiving new events. This is necessary, first, because the last implementation rebooted if after 10 tries of established Wifi connection, that lead to a deletion of the buffer. This version tries infinitely to connect to Wifi. Second, the Wifi connection might take some minutes. During that time we would not detect incoming new events if the internet initialization is not parallel. So, waking up if the buffer is full is not an option because then this functionality is broken. After this, it will wait for the semaphore `xInternetActive` before analyzing the buffer. The analyzing algorithm is the same as in the last report; however, this time we do not use the NVS. If the algorithm detected an increase or decrease in the count, it stores that in a `cJSON` object which is hold in the RAM. After emptying the buffer, the `cJSON` object is converted to a string, and it is sent via MQTT to the database. The last step before entering deep sleep is to check whether it is 23 o'clock. Then, we reset the count and enter the deep sleep for 7 hours. If that is not the case, we enter deep sleep for `WAKEUP_AFTER` seconds.
There is the possibility to refresh the ESP's display with a button because it can occur that the display's values are not up-to-date. Under the roof, this button forces the ESP to wake up from deep sleep and performs then the above described procedure.
We renamed the `wakeup_stub()` to `wakeup_routine()`.

1.7 Battery lifetime

We are estimating the battery lifetime for our implementation that includes analyzing the buffer. Because the ESP uses about 900 mAs to boot from deep sleep, then analyzes the buffer which takes about 3 seconds we came to the conclusion that the ESP would live for one deep sleep cycle of one hour.

2 Appendix

160MHz + ALS	80MHz	160MHz	240MHz	DFS(80,240)	DFS(80,240) + ALS
1109	942	1288	994	1134	1025
1006	1001	1073	1212	1023	1068
1082	755	1054	1143	1138	947
1108	901	1071	1290	981	779
933	912	906	1158	1137	932
887	789	974	1222	1243	960
880	904	1013	1152	1121	802
940	864	1053	1119	979	785
888	924	1074	1019	1104	940
889	981	1011	1131	987	790

Table 6 All measurements in mAs for the Startup phase.

No improvements	Interval 2	Interval 4	Interval 16	Interval 128	Interval 256
760	728	325	145	107	100
772	758	253	173	97	113
770	766	334	146	99	83
740	758	267	165	109	85
746	748	286	162	101	125
745	759	244	173	114	76
767	752	309	173	107	105
763	740	315	149	122	103
762	748	362	139	110	118
757	760	320	172	100	93

Table 7 All measurements in mAs for a corresponding beacon interval of 1,2,4,16,128, and 256.

80MHz	Time 80MHz	160MHz	Time 160MHz	240MHz	Time 240MHz	160MHz + ALS	Time 160MHz + ALS	DFS(80,240) + ALS	Time DFS(80,240) + ALS
1513	56	1150	37	1040	31	1148	37	896	33
1519	56	1157	37	1057	31	1165	37	911	33
1521	56	1146	37	1038	31	1164	37	939	33
1519	56	1160	37	1052	31	1155	37	927	33
1528	56	1153	37	1010	31	1160	37	918	33

Table 8 All measurements in mAs and seconds during the calculation phase. Seconds are marked with the column name "Time".

80 MHz	Time 80 MHz	160 MHz	Time 160 MHz	240 MHz	Time 240 MHz
982	28	805	19	770	16
986	28	842	19	756	16
993	28	808	19	750	16
985	28	808	19	760	16
986	28	819	19	755	16

Table 9 All measurements in mAs and seconds during the paralleled calculation phase. Seconds are marked with the column name "Time".

Light no display	Light with display	Deep no display	Deep with display
31	51	3	5
31	50	3	5
30	55	3	5
30	51	3	5
29	51	3	5

Table 10 All measurements in mAs for the sleep modes.