PROJECT 3
Energy Optimization

Michael Meli

ECE 561

Optimizations

Initial (Debugger Connected)

This was the initial run time. To begin, the SDA was connected.

Run Time: 6 seconds

	Active			Stop/Off			
Device	**Voltage (V)**	**Current (mA)**	**Power (mW)**	**Current (mA)**	**Power (mW)**	**Active Duty Cycle**	**Average Power (mW)**
MCU	3	9.7	29.1	0	0	1	29.1
Debug Adapter	3.1	16.6	51.46	0	0	1	51.46
Accelerometer	3.1	0.165	0.5115	0	0	1	0.5115
Red LED	3.1	1.6045	4.9741	0	0	0.02	0.0995

*Note: Accelerometer value is approximated from MMA8451Q datasheet

Optimization 1 – Disconnect SDA (Debug Adapter)

The most obvious first optimization step was to disconnect the SDA by removing the shorting jumpers on J3 and J14. This showed about double the improvement, but energy use was still high because of the professor.

Run Time: 12 seconds

	Active			Stop/Off			
Device	**Voltage (V)**	**Current (mA)**	**Power (mW)**	**Current (mA)**	**Power (mW)**	**Active Duty Cycle**	**Average Power (mW)**
MCU	3	9.7	29.1	0	0	1	29.1
Debug Adapter	3.1	16.6	51.46	0	0	0	0
Accelerometer	3.1	0.165	0.5115	0	0	1	0.5115
Red LED	3.1	1.6045	4.9741	0	0	0.02	0.0995

Optimization 2 – Enable Low Leakage Stop Mode

Since the processor was still using a lot of power, I enabled sleep modes. When the processor was not doing any work, I put it into low leakage stop mode, which was very small current consumption compared to normal run mode. This significantly improved runtime.

Run Time: 6 minutes, 11 seconds

	Active			Stop/Off			
Device	**Voltage (V)**	**Current (mA)**	**Power (mW)**	**Current (mA)**	**Power (mW)**	**Active Duty Cycle**	**Average Power (mW)**
MCU	3	9.7	29.1	0	0	0.033	0.9603
Accelerometer	3.1	0.165	0.5115	0	0	1	0.5115
Red LED	3.1	1.6045	4.9741	0	0	0.02	0.0995

*Note: Duty cycles are calculated using debug signals and an oscilloscope

Optimization 3 – Increase I^2C Speed

Clearly, reducing the amount of time the processor was in normal run mode was a fast way of getting big runtime gains. With my focus on reducing this time, I realized I could increase the speed of the I2C link with the accelerometer to reduce the amount of time it takes to read data, allowing me to go to sleep faster.

Run Time: 7 minutes, 5 seconds

	Active			Stop/Off			
Device	**Voltage (V)**	**Current (mA)**	**Power (mW)**	**Current (mA)**	**Power (mW)**	**Active Duty Cycle**	**Average Power (mW)**
MCU	3	9.7	29.1	0	0	0.017	0.4947
Accelerometer	3.1	0.165	0.5115	0	0	1	0.5115
Red LED	3.1	1.6045	4.9741	0	0	0.02	0.0995

Optimization 4 – Replace Unnecessary Angle Calculations with Pre-Computed Comparisons

Instead of computing the roll and pitch in terms of degrees, an optimization would be to compute the value that would go into the atan2f function and then compare that ratio to known and precomputed cut-off values for 15 and 30 degrees. This optimization reduces the number of computations that the processor needs to do, reducing active time.

Run Time: 7 minutes, 30 seconds

	Active			Stop/Off			
Device	**Voltage (V)**	**Current (mA)**	**Power (mW)**	**Current (mA)**	**Power (mW)**	**Active Duty Cycle**	**Average Power (mW)**
MCU	3	9.7	29.1	0	0	0.012	0.3492
Accelerometer	3.1	0.165	0.5115	0	0	1	0.5115
Red LED	3.1	1.6045	4.9741	0	0	0.02	0.0995

Optimization 5 – Reduce Sampling Rate of Accelerometer and Use Low Power Mode

The power models above show that the accelerometer uses a lot of power as it is constantly sampling, even when it is not being read. A few optimizations can be done in this area. First, I reduced the sample rate to a frequency of 12.5 Hz. This was the lowest frequency that ensured new data would be available every 100 ms. Next, auto-sleep mode was enabled on the accelerometer. Finally, the accelerometer's low power mode was enabled, which slightly reduces accuracy of the samples, but dramatically decreases power consumption. The result was quite significant, as the values in the power model, pulled from the datasheet, demonstrate.

Run Time: 14 minutes, 41 seconds

	Active			Stop/Off			
Device	**Voltage (V)**	**Current (mA)**	**Power (mW)**	**Current (mA)**	**Power (mW)**	**Active Duty Cycle**	**Average Power (mW)**
MCU	3	9.7	29.1	0	0	0.012	0.3492
Accelerometer	3.1	0.006	0.0186	0	0	1	0.0186
Red LED	3.1	1.6045	4.9741	0	0	0.02	0.0995

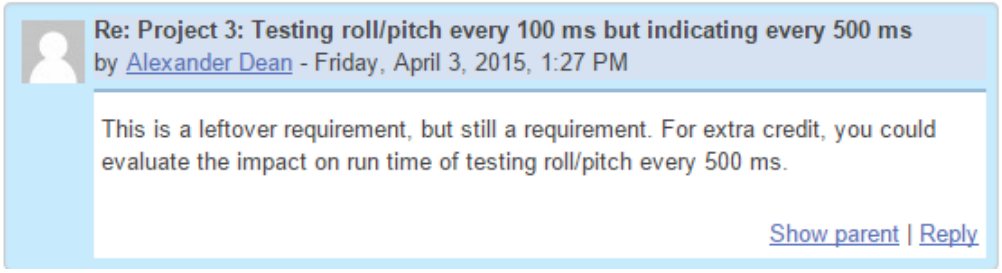
*Note: The active duty cycle cannot be determined as it depends on when the accelerometer goes into sleep, so this calculation assumes it is always in active mode.

(Extra Credit?) Further Analysis

Dr. Dean mentioned that we could receive extra credit for evaluating the impact on run time of testing roll/pitch every 500 ms (when the LED is updated) instead of every 100 ms.

See post titled “Project 3: Testing roll/pitch every 100 ms but indicating every 500 ms” on message board.

Link: https://moodle1415-courses.wolfware.ncsu.edu/mod/forum/discuss.php?d=241753



Using debug signals, we can see that the processor spends about 1.2% of every ~500 ms period awake. Most of that time awake is reading from the accelerometer, even though that data is not used but once every period.

The above logic analyzer display shows 4 periods of activity that are not necessary. If those were to be removed, the processor would only spend 0.28% of the time awake. This new value can be inserted into the power model from the last optimization step:

	Active			Stop/Off			
Device	**Voltage (V)**	**Current (mA)**	**Power (mW)**	**Current (mA)**	**Power (mW)**	**Active Duty Cycle**	**Average Power (mW)**
MCU	3	9.7	29.1	0	0	0.0028	0.08148
Accelerometer	3.1	0.006	0.0186	0	0	1	0.0186
Red LED	3.1	1.6045	4.9741	0	0	0.02	0.0995

As this demonstrates, the average power of use by the MCU is dramatically decreased. This would dramatically increase runtime.

To test the new runtime, I performed these optimizations and experimentally determined what the new runtime would be. The result was that the board lasted for 29 minutes, 5 seconds.