TDT4258: Energy Efficient Computer Design
Exercise #1

Due on Monday, February 10, 2014

Per Øyvind Kanstrøm
Tor-Håkon Bonsaksen
Pia Lindkjølen

Abstract

This assignment is about solving a fairly simple task and then experiment with different energy saving measurements based on software design methodology and a good understanding of how to alter the hardware setup to get better energy consumption. First, the group had to get familiar with the tools and the assignment. The process started with going through the test code delivered with the assignment, before writing our own code. When our program functioned the way the group had decided, that one specific LED is to light up when a corresponding button is being pushed, the process of examine the energy modes began. Noting how much energy was saved, or gained, was noted whenever something was changed. The start point of the board was 3.65 mA, that was when polling was used to check if a button was being pushed. Interrupts was implemented, without any sleep and the energy usage went up to 6.12 mA. Deep sleep was enabled and the energy consumption went down to 1.70 μA. The RAM blocks 1 - 3 was disabled and the energy consumption went further down to around 855 nA. The last action taken to lower the power consumption was to reduce the clock frequency and this got it down to around 845 nA, all of these measures are when no buttons are pushed.

Contents

About the group and the subject

The group consists of Tor-Håkon Bonsaksen, Per Øyvind Kanestrøm and Pia Lindkjølen, we are all on our last semester of bachelor in informatics. The decision to take this course was based on good recommendations from previous students.

Introduction

Why is this topic of interest?

Energy efficiency is an interesting field due to the massive, and still growing market of portable devices. Portable devices are dependent on batteries, and low power consumption and long battery time is a big selling point for many industries. The topic of reducing the power usage is of great importance and a field where massive improvements can be done.

What was attempted in the present effort?

The first thing that the group did was to write code that used polling to read from the GPIO. Then, this code was changed, to instead handle interrupts on the GPIO pins. Further improvements was made by using the Cortex M3 energy modes and set the microcontroller to sleep when it was waiting for interrupts. Lastly, some redundant SRAM blocks was disabled.

What will be presented in this report?

In this report the assignment will be presented and our approach to it step by step. Each partial assignment will be presented as it gets solved and the improvements as the gets implemented. The results section will describe in detail our power saving changes and their effect on the system. The source code will be attached and referenced.

Description and Methodology

Preparation and planning

The group started this assignment by getting to know the tools that are ment to be used, and getting to know the controller. This involved actually creating a project and going through all of the steps described for us in the compendium. It involved starting with exercise 0: building and running a testing program that was handed out and see if it worked. Exercise 0 went relatively fast and it was time to create our own project. This is our plan on how to proceed with this excercise:

- First, make our own program that does something simple; like lighting all the LED's on reset.
- Then, get our program to do what was decided to be our goal for this assignment, using polling. Our goal is to give each button a corresponding LED that is to light up when its button is being pushed.
- Third, our program should do the same, but with interrupts instead of polling.
- Last, make our program more energy efficient.

Polling

As explained, it was decided to make our program light the corresponding LED given that a button is being pushed. This is being done with a loop. The code[1] checks if a button is pushed or not by reading the buttons state, then updating the gpio register for the LED's with the state. Afterwards be branch back to repeat the process.

Interrupts

Now it was over to the third step which involved changing the logic to interrupts instead of the loop. The group followed the instructions for enabling interrupts in the compendium and started testing. At first nothing happened. Our interrupt handler did not run. It seemed like our code[**?**] was stuck in an infinite loop of some kind. After a bit of debugging it was figured out that the dummy handler branching out to nothing, was actually what was causing the code to get stuck. This was corrected by branching back to the return address in the link register.

Energy saving

Now the group have a working, interrupted driven program. It's time to look at how to tweak the hardware to be more energy efficient. Without any form of improvement the program was drawing around 5.8 mA on idle which meant there was room for major improvements. This section explains what was done. The results of the changes can be found under Results and Tests . All the changes are included in the file energysavings.s [3]. The efm32gg board is classified in five different energy modes. Where EM0 is normal operation, EM1 to EM4 are classifications of different energy optimizations. These techniques alter peripheral settings, high and low frequency clocks and clock frequencies, sleep mode, and other modifications.

Sleep mode Sleep mode was achieved by writing the value 6 to the M3's System Control Register. This activates deep sleep mode. Entering deep sleep can be done with a wfi (wait for interrupt) or wfe (wait for event) instruction. Sleep mode indicates a transition to EM2.

```
ldr r4, scr_addr
mov r5, #6
str r5, [r4]
wfi
```

Disabling sram To disable unused ram the value 7 was written to EMU_MEMCTRL. By default 128KB of ram is enabled. The linker script must be informed of this change. This is done by changing the length of the ram from 128KB (four blocks) to 32KB (one block) in the linker script[4].

```
mov r5, #7
str r5, [r4, #EMU_MEMCTRL]
```

Reducing clock frequency Slowing down the clock frequency on the high and low oscillator was done by setting the CMU_LFRCOCTRL and CMU_HFRCOCTRL to 0. This reduced the HFRCO to 1 MHz and lowered the LFRCO. This is EM3. The HFRCO is disabled in sleep mode but we lower it because it gets enabled during run mode after an interrupt.

```
mov r5, #0
str r5, [r1, #CMU_LFRCOCTRL]
mov r5, #0
str r5, [r1, #CMU_HFRCOCTRL]
```

Results and Tests

The results were measured using the eAPower software to get the average power consumption. Due to the spikes in energy usage at low energy levels we included the energy space (The approximate lowest and highest level of energy usage), to get the space the energy monitor on the board was used.
The list below contains the different versions created and the power consumption data that was pulled. All results exclude the led power consumption.

Polling The first program[1] was very basic with a loop that did all the work. All the logic runs whether or not a button is pressed this causes the power usage to be high at all times with little to no changes when a button was pressed.

Average: 3.65 mA
Space: 3.6-3.7 mA
Button press: 3.7 mA

Interrupts In our second version[2] interrupts were implemented without any kind of sleep. Due to the lack of sleep and the increased amount of logic that needs to run the power consumption was increased by 2.47 mA from the previous program. The power spiked a little during interrupts from 6.12 mA to 6.2 mA.

Average: 6.12 mA
Space: 6.1 mA
Interrupt: 6.2 mA

Deep sleep By enabling deep sleep major improvements to the power consumption was made. In EM2 the CPU is sleeping and the high frequency oscillator is turned off until woken up by an interrupt. The interrupt sets the chip to EM0 mode for the duration of the interrupt code and back to EM2 when the code finishes. This wakeup takes about 2μs. To get accurate power usage measurements in EM2 mode disconnecting the debugger was necessary, as it doesn't disable the high frequency oscillator otherwise. The EM2 mode uses the DMA for certain tasks instead of the CPU to save power as well.

Average: 1.70 μA
Space: 1.5 - 1.9 μA
Interrupt: 82.82 μA

Disabling ram The Cortex M3 consists of four 32KB blocks of SRAM for data space. Unused blocks of ram draws around 170 nA each. If the data space is bloated and unneeded, blocks one to three can be disabled. This will give a quite significant decrease in power consumption. In our case the remaining 32KB of SRAM is more than enough.

Average: around 855 nA
Space: 700-1000 nA
Interrupt: 81.95 μA

Reducing clock frequency The high frequency oscillator is disabled in EM2 and higher, but it re-enables after an interrupt so the group reduced it as well. The exact power usage reduction is hard to work out as the power usage measurements tools are not completely accurate at nano level.

Average: around 845 nA

Space: 700-1000 nA
Interrupt: 81.95 μA

Evaluation of Assignment

The assignment went fairly well but going from higher level programing to assembly programing was a challenge. All our previous assembly experience is from computer fundamentals course. Usually when encountering a problem programming in for instance Java it can easily be found information about the problem on the internet. During this assignment learning how to read microcontroller reference manuals and work out the solution ourselves was an important part and a new experience. This way the group learned to properly read and process advanced technical manuals. This assignment was very educational, fun and challenging!

Conclusion

With a few fairly simple steps the board went from consuming 5.8 mA to around 800 nA while idle. This staggering result came from good software methodology design and from learning how to tweak the mcu to be more energy efficient. Knowing these tricks are essential to succeed in a project using mcu's, especially in todays focus on energy effiency and in the forthcoming rise of IoT (Internet of Things) devices.
It has also been a good learning experience for using the GNU toolchain and giving a better understanding of the lower levels of using mcu's with gpio control and general assembly coding.

References

[1] Polling source file:
polling.s

[2] Interrupt driven source file:
interrupt.s

[3] Energy efficient source file:
energysavings.s

[4] Linker script file:
lfm32gg.ld