# Power Management in Embedded Systems: A Static Power Approach

Lubomir Valeriev Bogdanov and Racho Marinov Ivanov

*Abstract* – **The following paper focuses on an energy reduction technique in embedded systems called Static Power Management (SPM). The SPM is applied at compile time and aims at powering and/or gating off unused and inactive peripheral modules in the system. By investigating a particular example we try to gain data for future optimization algorithms.**

*Keywords* – **static power management, SPM, embedded systems, energy reduction.**

## I. Introduction

Embedded systems executing bare-metal and single-threaded firmware (i.e. no OS) expose a good level of compile-time predictability. One could take advantage of this feature and use it for energy reduction. This is where the Static Power Management, or SPM, could be used [1]. Unused peripheral modules in a microcontroller or in an embedded system can be gated off to reduce dynamic power consumption. Some implementations allow for a complete power down of a module. The latter reduces static and leakage consumption.

Theoretically an SPM algorithm can be implemented easily – whenever there is no hardware access to a peripheral, it can be switched off. But practically this is a daunting task. The time of the powering on and off, as well as the initialization of the module, could seriously harm the energy optimization. The purpose of this paper is to assess an application before and after using an SPM.

## II. Test Setup

### A. Basic explanation

For a test system we have chosen a typical closed-loop embedded system for object control. The block diagram is shown in Figure 1. Here we have a source of data that could be analog and/or digital. Usually the input data is fed into a converter that transforms it to a set of digital values. Those values are then processed and sent to an output converter. The signals from this converter are used by an actuator that controls the object of interest.

Having in mind this theoretical setup, we have used an ARM Cortex-M4 based microcontroller (Texas Instruments' LM4F232H5QD) to develop such a system. It is shown in Figure 2. As it can be seen from the figure there is no object to be controlled. This is because it is irrelevant from software point of view. Furthermore this would allow us to modify the processing pseudo-algorithm for our needs.

R. Ivanov is with the Department of Electronics, Faculty of Electronic Engineering and Technologies, Technical University - Sofia, 8 Kliment Ohridski blvd., 1000 Sofia, Bulgaria, e-mail: r.ivanov@tu-sofia.bg
L. Bogdanov is with the Department of Electronics, Faculty of Electronic Engineering and Technologies, Technical University - Sofia, 8 Kliment Ohridski blvd., 1000 Sofia, Bulgaria, e-mail: lbogdanov@tu-sofia.bg

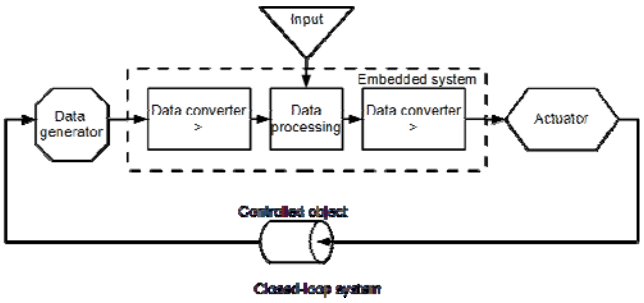

Fig. 1. Closed-loop embedded system.

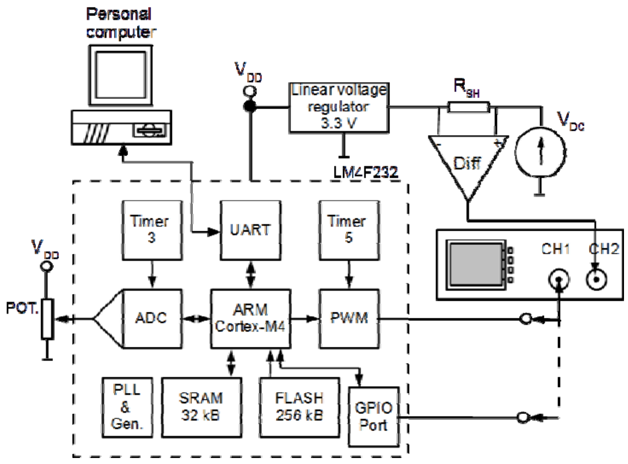

Fig. 2. System under test.

The firmware that we are testing reflects the closed-loop concepts. The original (non-instrumented) program is shown below with pseudo-code. The main loop of the program is

```
init( );

while(1){

set_gpio_pin( );

adc_val = measure_adc( );
pwm_val = process_data(adc_val);
set_pwm(pwm_val);
printf_uart(msg);

clear_gpio_pin( );

delay_ms(1);
}
```

separated with a toggle of a GPIO pin and a small delay for oscilloscope synchronization. Along with the current measurements, we are able to estimate the energy consumption for one period of the firmware. The algorithm starts with ADC measurements that represent the input data conversion part. The analog values are kept constant during the measurements. They can be adjusted by the potentiometer POT in Figure 2. The converted samples are fed to a processing function. The calculations that we perform are dummy. Then the processed data is passed to a function that sets the duty cycle of a PWM module correspondingly. The PWM is the output converter. It is monitored by an oscilloscope to make sure that the firmware is alive during the measurements. Between the processing part and the PWM module we have inserted some asynchronous data communication (UART) because usually such systems are controlled remotely.

### *B. Used modules*

To make a conclusion of the above-mentioned peripherals we will list them. Those will be the modules that will be switched on/off during the experiment. They are shown in Table 1. Unfortunately the manufacturer does not provide information about the current consumption of the separate modules. This is normal since a microcontroller is a software controlled device. Some currents are provided under some specific frequencies and conditions (such as all peripherals on, all peripherals off, sleep modes, etc). Therefore we cannot anticipate the current consumption of a separate module. That’s why we have chosen to measure the current before and after the optimization, as many researchers in the same field do.

TABLE 1. USED MODULES

| **Peripheral** | **Type** | **Function** |
|---|---|---|
| ADC | Analog-to-digital converter | Converts analog values to digital values |
| ARM Cortex-M4 | Microprocessor | Executes instructions |
| PWM | Pulse-Width-Modulator | Generates voltage impulses with variable length |
| UART | Universal Asynchronous Receiver-Transmitter | Communication module for sending/receiving bytes on the RS232 interface |
| Timer | Timer | Asserts interrupts at given time periods. |
| FLASH | Memory | Program memory |
| SRAM | Memory | Data memory |
| GPIO | General-Purpose Input Output | Module for software control of the pin levels in a micro-controller |

## III. STATIC POWER MANAGEMENT

A typical SPM algorithm for optimizing energy consumption checks at compile time for parts in the firmware code where certain modules of the microcontroller are inactive or are not used at all. Depending on the power management features, the clock signal supplied to the module could be gated off or even its entire supply voltage could be switched off [2]. The first case allows the programmer to initialize the module only once and reduce its dynamic power consumption during the inactive periods. The second case requires multiple initialization and will lead to dynamic power reduction, as well as leakage power, during the inactive periods [3] [4].

In this case study we focus on 4 variations of the application. The first three investigate optimization dependence on inactive-active transition time of the module, while the fourth one concerns optimizations with changes in the execution model of the firmware.

There are three possible scenarios that could affect the SPM optimization:

- The execution time of the main processing $t_{EXEC}$ is less or equal to the execution time of the functions that make the active-inactive transitions $t_{A\text{-}IT}$
$$t_{EXEC} \leq t_{A-IT} \qquad (1)$$
- Time $t_{EXEC}$ is considerably greater than $t_{A\text{-}IT}$
$$t_{EXEC} >> t_{A-IT} \qquad (2)$$
- Time $t_{EXEC}$ is considerably greater than $t_{A\text{-}IT}$ and interrupts from timers are used
$$t^{*}_{EXEC} >> t^{*}_{A-IT} \qquad (3)$$

The instrumented program differs from the original one by the additional API calls for the state transitions. The code is shown below. Other high-level source code transformations exist and are revised in [5].

```
init( );

while(1){

        set_gpio_pin( );

        init_adc( );
        adc_val = measure_adc( );
        disable_adc( );

        pwm_val = process_data(adc_val);
        set_pwm(pwm_val);

        init_printf( );
        printf_uart(msg);
        disable_printf( );

        clear_gpio_pin( );

        delay_ms(1);
}
```

As it can be seen, the PWM module is not turned off as this would affect the control of the object. On the other hand the ADC could be turned off because no conversion takes place while the bodies of the other functions are being executed. The only negative effect in this case could be slower response time which should be assessed by the developer as whether is admissible or not.

The origin of the instructions and data of the microprocessor could also affect the energy consumption. For this reason we have setup another experiment and have investigated execution from RAM and ROM. Modifications to the start-up code and linker script are required. To make a more general conclusion we have used several microcontrollers (LM4F232, LPC1114, MSP430FR5739). The experiment is described later in this section.

The compiler used is a GCC cross compiler. The target architecture is ARM Cortex-M and the cross compiler prefix is ‘arm-none-eabi-‘. The LM4F232 tests were performed with the following command line parameters:

```
-mthumb -mfloat-abi=softfp -Os -ffunction-sections -
fdata-sections   -MD   -std=c99   -Wall   -pedantic   -
DPART_LM4F232H5QD -c
```

The LPC1114 and MSP430FR5739 tests were performed with:

```
-D__REDLIB__ -D__USE_CMSIS -DDEBUG -D__CODE_RED -O0 -
g3   -Wall   -c   -fmessage-length=0   -fno-builtin   -
ffunction-sections  -fdata-sections  -mcpu=cortex-m0  -
mthumb -MMD -MP
```

As it can be seen from the above, in the first case we use optimizations for size of the code and in the second one – no optimizations at all. Those parameters were taken from the respective manufacturer’s integrated environments and were not changed. The effects of the optimization levels are irrelevant to this experiment.

The linker used is a GCC’s LD. Again, the prefix is ‘arm-none-eabi-‘. The linker script changes include:
- relocation of the .text section to SRAM, right next to the .data and .bss sections;
- relocation of the vector table in SRAM;
- changes in the start-up code to copy the .text section and vector table to SRAM.

The address range of the SRAM was provided in the respective microcontroller datasheets.

## *A. Measurement equipment and statistical analysis.*

The measurement equipment used in this experiment is a shunt-resistor and a differential amplifier based one. More information about it is published in [6]. The error of the Idd current readings is 0,33 % in the range 0 ÷ 300 mA.

The number of measurements performed was once per each test case. As mentioned before, when we talk about optimizations, seldom is someone doing absolute measurements. What we need here is one measurement before and one measurement after the optimization.

## *B. Execution time $t_{EXEC}$ commensurable with transition time $t_{A\text{-}IT}$*

The original program was measured to consume 112 µJ per one period (one while(1) loop). The instrumented program in the first case had the process_data( ) function simplified down to one line of code with a dummy calculation. As expected, the resulting SPM optimization yielded 144 µJ per period (increase of 29 %). The time period was increased as well (52 %). Therefore we can conclude that the optimization was unsuccessful.

## *C. Execution time $t_{EXEC}$ greater than transition time $t_{A\text{-}IT}$*

In the second example we increased the time of the process_data( ) function by adding more computations. Thusly we simulate negligible time for the transitions. The original program now consumes 15 555 µJ per cycle. With the help of SPM this was reduced to 14 256 µJ (8,4 %). The time was increased with 0,6 %.

## *D. Execution time $t_{EXEC}$ greater than transition time $t_{A\text{-}IT}$ using timers.*

In the third case we used timers that trigger wake events every 0.5 s. During the rest of the time the system is sleeping. It appears that the wake-up of the system costs additional 5066 µJ which increased the consumption of the original program to 20 621 µJ per cycle. Despite this, the SPM was successful – the energy was reduced to 18343 µJ (11 %) per cycle. The increase in time was 1,3 %.

Figure 3 shows graphical view of the results. With SPM the execution time of one period is inevitably increased. The API functions that are inserted for clock gating and power switching introduce this overhead.

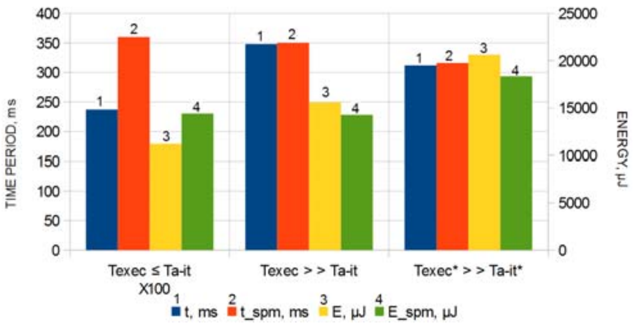

Fig. 3. Time period and energy consumption change before and after applying SPM.

## *E. SPM with change in the execution model.*

Changing the instructions’ address of execution may lead to energy reduction. This is dictated by the fact that volatile and non-volatile memories have different static and dynamic power consumption, as well as access times. The most common scenario in an ARM Cortex-M based microcontroller is to execute parts of the program in read-only and random access memory. The other concept is to store the program in ROM and at start-up relocate it to RAM. We used two more microcontrollers for the experiment. Their architectures are listed in Table 2.

Every instruction has a load memory address (LMA) and a virtual memory address (VMA) [7]. The LMA is the address at which the instruction is stored. In our case this is the read-only memory. The VMA is the address of the instruction when it is executed.

TABLE 2. TARGET MICROCONTROLLERS

| Microcontroller | Microprocessor | Architecture | ROM |
|---|---|---|---|
| LM4F232 | ARM Cortex-M4 | Harvard | Flash |
| LPC1114 | ARM Cortex-M0 | Von Neumann | Flash |
| MSP430FR5739 | MSP430 | Von Neumann | FRAM |

Here we investigate execution from ROM and RAM. The RAM case is accomplished by modifying the start-up code of the firmware to relocate the vector table. The linker script is modified to have the appropriate LMA and VMA for each microcontroller and its respective address map. Once set-up, the addresses of the running code were verified with a debug adapter.

The tests were conducted with 3 different applications:

- Computationally intensive (processData)
- Memory access intensive (copyBuffer)
- External communication intensive (sendOnUART)

One might think that executing code from non-volatile memory consumes more energy than the case with the volatile one. The test, however, yielded different results (shown in Table 3). It can be seen that the technology of the non-volatile memory could make read/write accesses faster and more energy efficient. A proof for this is the communication application run on an MSP430FR5739 where the execution from RAM consumes more energy compared to the ROM execution. We also have 4 cases where there is little or no difference for the memory used.

TABLE 3. ENERGY CONSUMPTION OF CODE EXECUTED FROM ROM AND RAM

| Arch Test | Harvard (SRAM + Flash) | Von Neumann (SRAM + Flash) | Von Neumann (SRAM + FRAM) |
|---|---|---|---|
| processData | $E_{FLASH} \approx E_{SRAM}$ | $E_{FLASH} > E_{SRAM}$ | $E_{FRAM} = E_{SRAM}$ |
| copyBuffer | $E_{FLASH} > E_{SRAM}$ | $E_{FLASH} > E_{SRAM}$ | $E_{FRAM} = E_{SRAM}$ |
| sendOnUART | $E_{FLASH} > E_{SRAM}$ | $E_{FLASH} = E_{SRAM}$ | $E_{FRAM} < E_{SRAM}$ |

## IV. CONCLUSION

The work presented in this paper could help in the making of an SPM optimization algorithm. The important information gathered here is that the energy reduction by the means of static power management comes always at a prize and that would be the execution time of the application. However we still consider this as a successful method because for a small time degradation we achieve decent energy reduction (values close to 10 %). Furthermore the SPM algorithm should provide the programmer with means of excluding modules from the optimization as this might interfere with the system's behavior (in our case this was the PWM module).

The SPM should also be supplied with information about the execution times of the application of interest and the active-inactive transition times. This would allow the prediction of the optimization outcome. In the cases where the A-IT times are greater or equal to the main application's duration the optimization will fail. Otherwise the SPM will succeed.

## REFERENCES

[1] W. Chedid, C. Yu. Survey on Power Management Techniques for Energy Efficient Computer Systems, Cleveland State University, 2002.
[2] ARM926EJ-S Technical Reference Manual, ARM Limited, 2008.
[3] F. Shearer. Power Management in Mobile Devices, Elsevier Inc., 2008.
[4] T. Gloekler, H. Meyr. Design of Energy-Efficient Application-Specific Instruction Set Processors (ASIPs), Kluwer Academic Publishers, 2004
[5] C. Hsu. Compiler-directed dynamic voltage and frequency scaling for CPU power and energy reduction, PhD thesis, New Jersey, 2003.
[6] L. Bogdanov. Embedded System For Consumption Investigation Of Function Calls With The Optimization Program Powot, Techsys Conference, Plovdiv, 2013, Book1, p.149.
[7] S. Chamberlain, I. Taylor. The GNU linker, Free Software Foundation Inc., 2013.