

Approaches for reducing the power consumption in embedded systems

Lubomir Valeriev Bogdanov and Racho Marinov Ivanov

Abstract – The following paper describes some methods for the reduction of power consumption in embedded systems. They will be used as a basis for the development of a tool that will optimize the power consumption automatically.

Keywords – power optimization, power reduction, embedded systems, energy efficiency, Powot tool.

I. INTRODUCTION

Nowadays embedded systems are used in every aspect of our lives – from personal entertainment to healthcare and manufacturing. No matter where used, almost in any case the issue with the power consumption is one of the most daunting tasks to be solved. Portable devices in particular must use optimized software for maximum throughput, as well as minimum consumption. A tradeoff between these two must be found which should extend the discharge period of the battery.

Most of the microcontrollers today have ways to reduce their power consumption. For this purpose they use software and hardware methods. It is the software developer’s responsibility to take into account the power consumption while writing the program code. Thus the final results may vary. On the other hand the hardware is fixed and the power reduction is more determined. The overall optimization is dictated by the application.

A tool for automatic code optimization can be developed, so that the final program is more energy-efficient. This way the developer can write a program without observing any rules for power savings and then use this tool to automatically modify the code.

II. SOFTWARE METHODS FOR POWER REDUCTION

The firmware of the microcontroller has a certain set of options for reducing power. Among them are the dynamic change of the processor frequency, selective peripheral clocking, sleep and deep sleep modes of operation in Cortex-M cores, and interrupt handler operation.

A. Dynamic change of the processor frequency

The first and most widely-used method for power reduction is the dynamic change of the processor frequency during the execution of the firmware. If the application tolerates

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

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

such a behavior, then a low frequency could be used during the idle periods and a high frequency – during the intensive calculations. In Fig. 1 a graph of the current consumption is shown. The used part is the Texas Instruments’ Stellaris LM3S6965. An idle while(1) { } loop is used for the measurements.

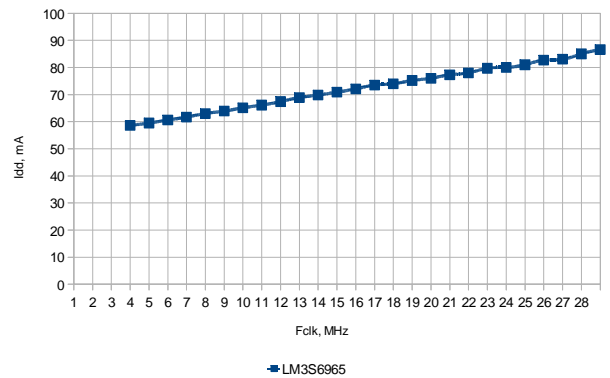


FIGURE 1. CURRENT CONSUMPTION VS MICROPROCESSOR FREQUENCY

It is obvious that the graph is linear and therefore the more throughput is required, the bigger the power consumption is.

B. Selective peripheral clocking

The trend in microcontrollers is to have individual control over the clocking of each peripheral [1]. This helps to decrease significantly the dynamic power of the whole chip.

Some measurements of the LM3S6965 peripherals are shown in Table 1. Their maximum data transfer rate is used. The number 0x55 is the sent data in all the cases to force maximum switch rate of the pins, which in turn simulates the worst case scenario of power consumption during the data transmission. If we look at each module apart from the others the consumption is not that high but

TABLE 1. PERIPHERAL CONSUMPTION OF LM3S6965

Peripheral type	Current consumption, mA
GPIO	2.91
SSI	12.64
I2C	5.52
UART	0.88
Ethernet	4.99
ADC	4.57
PWM	5.63
COMP	1.42

in most of the cases the application would require more than one module to operate. Combined together, the overall consumption increases significantly. Therefore the best way to write the program is to enable and disable each module on demand. This method has one drawback – the start-up time of the peripherals might slow down the execution of the program.

C. Sleep and Deep Sleep modes of operation in Cortex-M cores

The Cortex-M cores implement two low-power modes: sleep and deep sleep [2]. Since these cores are provided as an IP library to many manufacturers, a lot of the microcontrollers today also have these two modes of operation. In *sleep mode* the clocking to the microprocessor is stopped [3]. The peripheral modules such as UART, SPI, Ethernet and so on continue to operate independently and only when data processing is required the core is awakened from this mode. This is done with the help of interrupts.

The microprocessor exits the normal mode of operation (also called thread mode) and enters the sleep mode when the WFI instruction is executed. WFI stands for Wait For Interrupt. One more instruction could be used for this same purpose – the WFE, or Wait For Event, instruction. The difference is that by using WFE the sleep mode is entered if the value in the Event Register is 0. If the value in the Event Register is 1, then it is cleared and the execution of the program continues without entering sleep mode. This mechanism is used when an external event signal is triggered or when another microprocessor in a multi-processor system has executed an SEV instruction. On wake up from sleep mode some system restore tasks must be executed first.

For further power savings a *deep sleep mode* is implemented. In this mode the system clock is stopped along with the PLL and the flash memory. The microprocessor enters one of the two modes depending on the second bit in the SCR (System Control Register) register. If this bit is 0, sleep mode is entered. If it is 1, deep sleep mode is entered after executing the WFE or WFI instruction.

To test the low power functionality of a LM3S6965 chip two measurements were made:

- One measurement without sleep mode
- One measurement with sleep mode.

In the first case the firmware implemented a simple UART echo with interrupts, the PLL was on, the processor frequency was 50 MHz, and an idle loop was used:

```
int main(void)
{
    //Initializations omitted for clarity
    .
    .
    while(1)
    {
        }
    }
}
```

This example yielded a power consumption of 350.58 mW. After using the sleep mode, which is implemented with the

SysCtlSleep() function in Stellaris Peripheral Driver Library [4], the program was modified to:

```
int main(void)
{
    //Initializations omitted for clarity
    .
    .
    while(1)
    {
        SysCtlSleep( );
    }
}
```

Now in the second example a power consumption of 272 mW was measured. The total reduction of power is 78 mW.

D. Interrupt handler operation

In some cases it is possible that the microcontroller's firmware operates entirely in interrupt handlers. That's why the ARM Cortex-M cores have this special feature – a Sleep-on-exit bit in the SCR register that allows the microcontroller to enter sleep or deep sleep mode automatically on return from an interrupt routine. This way an empty main() function could be used. If the Sleep-on-exit bit is clear the microprocessor would enter thread mode after returning from interrupt mode.

III. HARDWARE METHODS FOR POWER REDUCTION

Along with the software there are also hardware methods for power reduction. While developing the hardware certain rules must be followed in order to have a low-power system. The most common way to do this is to control the voltage regulator connected to the microcontroller. The power control over the external peripherals is another option. And last, but not least, is the ability to control the voltage of the core's DC/DC converter.

A. Main regulator control

In the common embedded system the voltage regulators of the microcontroller and the external peripherals are separated. This reduces the risk of a hardware reset of the controller caused by noisy periphery.

In some cases the application requires that the system works for a certain period of time, then goes to a low-power state, then the system is awakened by the user or an external device. In these cases the voltage regulator, supplying the controller, can have an Enable pin. With the help of it the microcontroller could turn itself off automatically and this way reduce the power consumed during the idle periods. A simple block diagram of such a system is shown in **Fig. 2**. Nowadays most voltage regulators have an Enable pin.

The limitations in this setup come from the turn on method. Once turned off, only an external event can turn the microcontroller back on. This can either be the operator of the device or another chip in the system. Therefore this method is not applicable for autonomous devices.

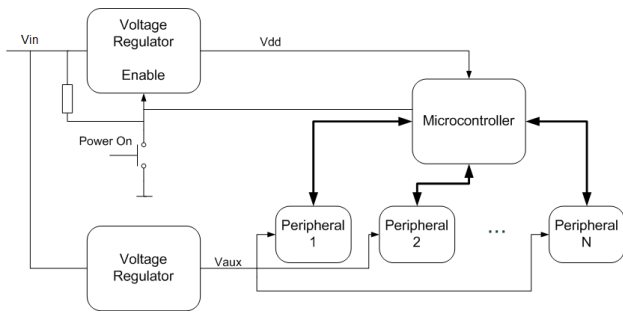


FIGURE 2. CONTROLLING THE MAIN REGULATOR

B. External peripheral control

Most of the applications require more than one external peripherals. No matter if they are used or not at a certain time, the static power consumption of all devices can decrease the energy efficiency of the system. Therefore if a certain peripheral is not needed at a certain time, it can be switched off by the processor. To do this, the whole system should have electronic power switches for every external device. Such a system is shown in **Fig. 3**.

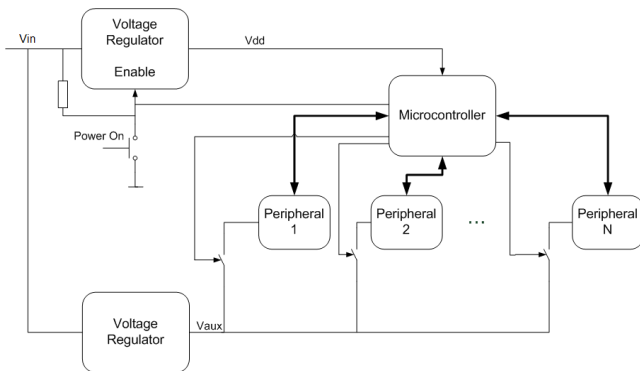


FIGURE 3. CONTROLLING THE EXTERNAL PERIPHERALS

Of course the corresponding data buses must be put to a high-impedance state or to a logic zero before the actual device turn off. Otherwise the risk of powering on the periphery, through one of the transient suppression diodes, exists.

Once again – the start up time of the peripheral is critical as with the internal peripherals. It might make this method pointless if the time is longer than the working period. Some devices also require processor interaction for the initialization which might waste more energy than save it.

C. Core voltage control

Some microcontrollers provide a separate pin/pins for the microprocessor power supply. This allows us to use a digitally programmable voltage converter to scale its output voltage to reflect the power needs of the situation. The higher the voltage, the better performance and vice versa. This method is shown in **Fig. 4**. It is obvious that if the quiescent current of the converter is significant, then this type of setup is pointless.

Chips that don't have extra pins for the core supply can't optimize power using this method. Some microcontrollers however have a built in programmable LDO which exclu-

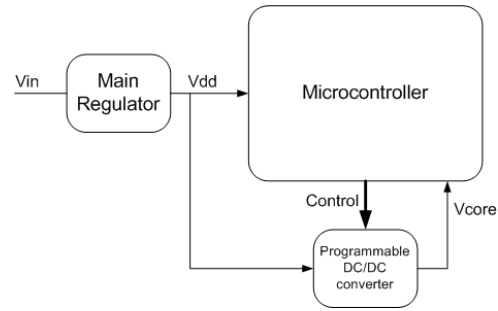


FIGURE 4. CONTROLLING THE CORE VOLTAGE

des the need of an external one. The LM3S6965 is such an example. In this case the power reduction is indirect because the low voltage will reflect the maximum operable processor frequency. The LDO doesn't reduce the power itself since the pass element operates in linear mode.

IV. POWER OPTIMIZATION TOOL

By combining the software and hardware methods described in this paper a tool that automatically optimizes the program code can be developed. Let's call this tool Powot (**Power Optimization Tool**). The idea is to maintain system operability while reducing the power. This is a daunting task since some demanding criteria must be met. The tool has to be:

- microcontroller independent
- application independent
- hardware power reduction aware
- able to update itself
- command line tool for better IDE integration.

The most hard to implement of these features is the second one – the application independence. Sometimes the application might not allow the program to be optimized because the optimization algorithm relies on sleep modes, frequency reduction and switching off power to the microcontroller.

The *first step* of the tool genesis would be the creation of a benchmark program that will test the tool's work. It has to be a program that will be first compiled without the optimization and loaded into the chip, then compiled with the optimization and loaded again. The power measurements should show power decrease without slowing down the program. The latter could be verified if the benchmark sets a pin every few hundred calculations as a flag that can be monitored with an oscilloscope. A small decrease in performance is negotiable if the power reduction is significant. It is also a good idea to use a common peripheral that can be found in most microcontrollers, such as UART, otherwise only the microprocessor would be used. The benchmark will be chip-dependent, unlike the Powot tool, but the algorithm of the program will be the same. A simplified algorithm is shown in **Fig. 5**.

In the *second step* an IDE for the creation of the tool should be chosen. The priority is to use open source tools and the Powot itself will be open source. The Eclipse IDE is suitable for this purpose [5]. The GCC compiler and Eclipse both have versions for Windows as well as Linux.

Other open source IDE is the Qt Creator [6]. Qt includes a framework that could reduce the effort for development with high-level built-in APIs. The only disadvantage of this approach – the Powot tool will be dependent on Qt libraries. Therefore if Qt Creator is selected as the IDE, then the first thing that has to be done is to rebuild it with the ‘static’ option. This way the executable will be Qt independent but might grow in size.

Both Eclipse and Qt Creator can build console applications which satisfies the fifth requirement for Powot.

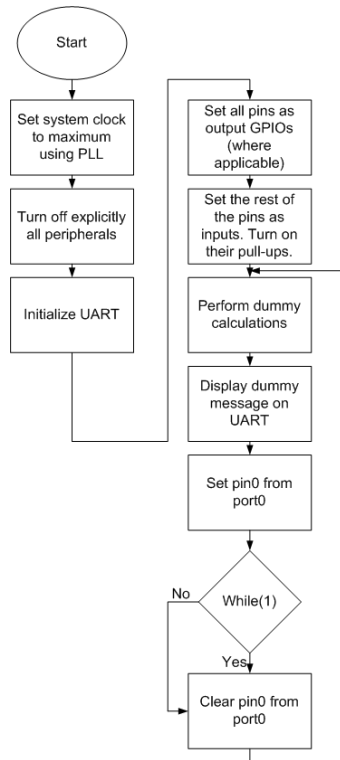


FIGURE 5. BENCHMARK ALGORITHM FOR POWOT TESTS

Next, in the *third step*, a version control must be chosen. The fourth requirement for update capability can be accomplished with the help of CVS, Svn, Git, Bazaar, Mercurial, etc. All of the mentioned version control tools are integrated in Qt Creator.

Another important preparation for the development is the use of the Doxygen tool. It is a tool that creates documentation from the comments in the source code. Every project should have this option since it might grow in size and it is convenient to have some kind of documentation.

The *fourth step* is the development of the Powot tool when everything else is set up. Now the first three requirements should be taken into account.

The microcontroller independence can be overcome with the help of auxiliary files. They will contain rules for every company and every family of microcontrollers within this company. The syntax for these files should be kept as simple as possible in order to help spread the tool among the different companies. This syntax will describe microcontroller-specific structures for sleep modes, clock control and so on, that will be inserted in the code later on. The ‘hardware power reduction aware’ requirement is actually that part of the Powot tool that will be responsible

for the optimization of the code using the hardware methods described earlier in this paper. This means that the user will have to edit configuration files for each hardware system. The user must tell the Powot tool what kind of peripherals there are and what kind of power reduction methods can be applied to them. Because there is a user interaction, a simple syntax of those files is obligatory, otherwise the prolife of Powot might be limited.

Finally, there’s no recipe for the second requirement – the application independence. A certain set of algorithms must be implemented and tested. The code that would give best results would be included in the final release.

The code analysis can be implemented in two ways:

- top-level source file analysis
- entire project analysis.

The first method includes analysis only of the top-level source file in the project. This is the file where the main() function is. The other source files and headers (.h, .c, .cpp) are not checked and therefore the functions implemented in them will not be power optimized. The Pngen tool uses the same approach for making a sequential code into a parallel one [7].

The second method includes analysis of all the project files. This way every function could be optimized for power and the overall power consumption may be reduced more than the one in the previous case.

V. CONCLUSION

The need to reduce the power consumption in embedded systems is becoming a serious issue in the recent years. Many microcontrollers have ways to reduce power but the software developer must be aware of them and apply them carefully in each system. This increases the development time since in many cases the power optimization is trial-and-error based. Therefore a tool that does the power optimizations automatically can be of a real help during the firmware development. It will set apart the programmer from all the details of the power management system. The development time will be reduced and the final product will be more energy-efficient. That’s why the future Powot tool will find its place in the firmware IDE and hopefully catch on in more than one companies.

ACKNOWLEDGEMENT.

The present research is supported by the Technical University – Sofia under contract № 121ПД0005-03.

REFERENCES

- [1] Stellaris LM3S6965 Microcontroller Data Sheet, Texas Instruments Incorporated, 2007 – 2011.
- [2] Cortex-M3 Technical Reference Manual, r2p1, ARM.
- [3] Cortex-M3 Devices Generic User Guide, ARM, 2010.
- [4] Stellaris Peripheral Driver Library User’s Guide, Texas Instruments Incorporated, 2006 – 2011.
- [5] <http://www.eclipse.org>
- [6] <http://qt.nokia.com>
- [7] Nikolov H., System-Level Design Methodology for Streaming Multi-Processor Embedded Systems, ISBN 978-90-9024163-0, 2009.