SETIT 2007
4th International Conference: **S**ciences of **E**lectronic,
Technologies of **I**nformation and **T**elecommunications
March 25-29, 2007 – TUNISIA

Embedded Software Power Optimization Techniques in Real Time Operating System.

Mr. N. Abid Ali Khan* and Prof. K. Ushadevi **

*R&D Engineer-Embedded Systems & Lecturer,
TIFAC-CORE,
SASTRA University,
Thanjavur-613402,
Tamil Nadu, India.
abid_ms@rediffmail.com

** Dean-School of Computing & Coordinator-TIFAC-CORE,
TIFAC-CORE,
SASTRA University,
Thanjavur-613402,
Tamil Nadu, India.
ushadevi@hotmail.com

Abstract: The timing analysis of an Embedded Processor in multitasking Real Time Operating System helps the Embedded Software designer to know the amount of CPU spent while executing a task. This paper describes the firmware approach that can be used to judge the system software design versus a maximum processor load. Once the timing analysis of the processor is done, it is the designer's best interest to minimize the power consumption of an embedded CPU. We will look at four software methods of intelligent waiting, event reduction, performance control and intelligent shutdown to minimize the processor power consumption. These software approaches are particularly for embedded applications where the processor has to react to external events such as detecting a key by the user through handheld terminals, PDA(s) or sensing the touch screens.
Key words: Embedded processor, power, task, event, interrupt, Real Time-Operating System (RTOS), scheduler.

INTRODUCTION

Selecting a processor is one of the most critical decisions when we are designing an embedded system. This selection is based on the features required to satisfy the control functionality of the final product and the raw computing power needed to fulfill those system requirements. Once the processor is finalized, it is the embedded software designer's role to identify the number of tasks that the processor has to perform to satisfy the system requirements. This paper provides the way to evaluate the processor utilization and methods to optimize the processor power consumption through software in multitasking Real Time Operating System (RTOS).

The first half of the paper deals with the techniques of evaluating the processor utilization by performing the timing analysis of the processor. The idea of calculating the time spent by the processor in ideal task helps the designer to know how well the kernel is scheduling the tasks and how much time the processor is idle. To perform this analysis, we will look into the Logic State Analyzer (LSA) method.

Knowing the idle time of the processor, it is the system developer's best interest to know how to minimize the power consumption of the processor. We will look in to the four of such software methods of intelligent waiting, event reduction, performance control and intelligent shutdown to minimize the power. Writing power aware system software for applications where the processor has to trigger to external events such as detecting a key in hand held terminals, PDAs or sensing the touch screens helps to improve the lifetime of the processor as well as to improve overall reliability of the system, satisfying the requirements of the customer.

1. Timing Analysis-Logic State Analyzer (LSA) Method

The CPU utilization 'U' is defined as the total amount of time spent in the idle task as Equation-1.

U = 100% - (% of time spent in the idle task). ----- (1)

The idle task is the task with the absolute lowest priority in a multitasking system. This task is sometimes called as the background task or background loop. Normally this task is an infinite loop that spins the CPU waiting for an indication that critical work need to be done as shown in Listing-1.

The Logic State Analyzer (LSA) watches the address, data buses and captures the data, which we can interpret at any time. Here we should configure the Logical State Analyzer to trigger on an instruction fetch from a specific address and measure time between each occurrence of an observation of this specific address.

The address to watch can be any address within the while (1) loop in Listing-1. The task of identifying the address is tricky but not difficult to find out. We can use the map file output by the linker to get close to the good address. The beginning of the While (1) loop is very easy to identify with a little iteration and some guessing work to get an address close to the desired one. Another way to measure the time is, if we observe Listing-1, the Check CRC () function is called every time through the background loop. If we could ensure that this is the only place where this function is called, we could use the entry to this function as the marker for taking the timing measurements.

```
int main (void)
{
   Setup Interrupts ();
   Initialize Modules ();
   Enable Interrupts ();
   while (1)
     {
        Check CRC ();
        Monitor Stack ();
        // Some non-time critical logic here.
     }
}
```

Listing-1: Method-1 for triggering LSA.

If it is difficult to capture the address within while (1) loop, another method is to set some dummy variable with some value every time through the background loop. The LSA could trigger on writing to this special variable as in Listing-2.

```
extern INT8U ping;
int main (void)
{
   Setup Interrupts ();
   Initialize Modules ();
   Enable Interrupts ();
   while (1)
     {
        ping = 0x9B;
        Check CRC ();
        Monitor Stack ();
        // Some non-time critical logic here.
     }
}
```

Listing-2: Method-2 for triggering LSA.

However the code change in Listing-2 is so minor that it should have a negligible effect in over all system.

Once the LSA is successfully triggered with any of the methods specified, the next step is to collect the time measured from instance to instance. To measure the CPU utilization accurately, the average time to execute the background task must be as accurate as possible. To get an accurate measurement of the background task using the LSA method, we must ensure that the background task gets interrupted as little as possible.

No interrupts at all is an ideal task. Essentially two classes of interrupts can disrupt the background loop. These are event-based triggers and time based triggers. Event based triggers are usually instigated by devices, modules and signals external to the microprocessor. When measuring the average background time, we should take all possible steps to remove the chance that these items can cause an interrupt that would artificially elongate the time attributed to the background task. It may be possible to disable the timing interrupt using the configuration options. If it's possible, the background measurement should be extremely accurate and the load test can proceed further. However, if it's impossible to disable the time-based interrupts, then we have to conduct a statistical analysis of the timing data.

Once we know the average background task execution time, we can measure the CPU utilization while the system is under various states of loads by using equation (1). As there is no other way to measure the CPU utilization directly, we have to derive the CPU utilization from measured changes in the background loop.

2. Power Consumption Techniques:

Once the timing analysis of the application software in multi tasking RTOS platform is done by the methods specified earlier, it is the designer's best interest to know the software methods to reduce the power consumption of the processor while the processor has nothing to do.

2.1. Intelligent Waiting

Many of the latest embedded processors include run-time power modes that can be used to scale power consumption. The most common of these is idle mode of operation, in which the instruction-executing portion of the processor core shuts down while all peripherals and interrupts remain powered and active. Idle mode consumes substantially less power than the processor is actively executing instructions. A key aspect of idle mode is that it requires little overhead to enter and exit, usually allowing it to be applied many times every millisecond. Any time the operating system detects that all threads are blocked-waiting on an interrupt, event, or timeout-it should place the processor into idle mode to conserve power. Since any interrupt can wake the processor from idle mode, use of this mode enables software to intelligently wait for events in the system. For maximum power efficiency, however, this tool requires that we design our software carefully.

Operating the processor in idle mode is much simpler during the cases where the application software is waiting for an external event. For example, the processor has to wait for an event from the user of the hand held terminal or PDA until the user presses a key or sense the input form the user of a touch screen to process the request. During the time when the processor doesn't get any external interrupt, switching the processor in idle mode is best choice to reduce the power.

Idle mode can even be used in cases where the event cannot be directly tied to an external interrupt. In these situations, using a system timer to periodically wake the processor is still preferable to polling. For example, if we are waiting for an event and know that the event can process it quickly enough as long as we check its status every millisecond, enabling a 1ms timer and placing the processor into idle mode will be the best implementation to minimize the power. We have to check for an event every time the interrupt fires; if the event doesn't get, we have to return to idle mode immediately.

2.2. Event Reduction

Another technique to consider is event reduction. Whereas intelligent waiting enables the processor to enter its idle mode as often as possible, event reduction attempts to keep the processor in idle mode as long as possible. It is implemented by analyzing the application software code and system requirements to determine the effect of alteration of processing the interrupts. For example, if we are working with a multitasking RTOS that uses time slicing to schedule threads, the RTOS will typically set a timer interrupt to occur at the slice interval, which is often as small as 1ms. Assuming that the code makes good use of intelligent waiting techniques, the RTOS will frequently find the opportunities to place the processor into idle mode, where it stays until it is awakened by an interrupt.

Even if the scheduler determines that all threads are blocked and quickly returns the processor to idle mode, this frequent operation can waste considerable power. In these situations, the time-slice interrupt should be disabled when idle mode is entered, waking only when another interrupt occur. Ideally, RTOS should be able to set variable timeouts for its scheduler. The operating system knows whether each thread is waiting indefinitely for an external or internal event or is scheduled to run again at a specific time. The operating system can then calculate when the first thread is scheduled to run and set the timer to fire accordingly before placing the processor in idle mode. Variable timeouts do not impose a significant burden on the scheduler and can save both power and processing time.

But variable scheduling timeouts are just one means of reducing events. Direct memory access (DMA) allows the processor to remain in idle mode for significant periods even while data is being sent to or received from peripherals. DMA should be used in peripheral drivers whenever possible. The savings can be quite impressive. For example, the receive FIFO on a serial port of a Rabbit processor generates an interrupt for approximately every byte that is received. At 19200 kbps, a 1KB burst of data sent to this serial port would cause the processor core to be interrupted and possibly awakened the processor from idle-almost 2400 times per second. Therefore during the situations where the processor is more prone to face the external interrupts, the DMA approach will be the solution.

2.3. Performance Control

Dynamic clock and voltage adjustments represent the cutting edge of power reduction capabilities in embedded processors. This advance in power management is based on the following observation: the energy consumed by a processor is directly proportional to the clock frequency driving it and to the square of the voltage applied to its core.

Processors that allow dynamic reductions in clock speed provide a first step towards power saving. Cutting the clock speed in half, drops the power consumption proportionately. Effective strategies using this technique alone can be implemented since the code being executed may take twice as long to complete, in that case, no power may be saved.

Dynamic voltage reduction is another approach. An increasing number of processors allow voltage to be dropped in concert with a drop in processor clock speed, resulting in a power savings even in cases when a clock-speed reduction alone offers no advantage. In fact, as long as the processor does not saturate, the frequency and voltage can be continually reduced. In this way, work is still being completed, but the energy consumed is lower overall. The challenge is identification of when to decrease clock frequency and when to decrease voltage without noticeably affecting performance. These two techniques become even more challenging tasks for software developer in multitasking environments.

2.4. Intelligent Shutdown

All the above three methods are for the power consumption when the device is running. This method of intelligent shutdown is useful when the device is turned off. If we consider most of the applications involving the initiation by hand held devices or PDAs or even touch screens, the customer will be satisfied if the device runs from the step where it has been stopped previously. This is accomplished with an intelligent shutdown procedure.

When the user turns the device off by pressing the power down button, an interrupt signals the RTOS to begin a graceful shutdown that includes saving of the context of lowest-level registers in the system. The operating system does not actually shuts programs down, but leaves their contents (code, stack, heap and static data) in memory. It then puts the processor into sleep mode, which turns off the processor core and peripherals but continues to power important internal peripherals, such as the real-time clock. In addition, battery-backed DRAM is kept in a self-refresh state during sleep mode, allowing its contents to remain intact. When the power button is pressed again, an interrupt signals the processor to wakeup. The wakeup ISR uses a checksum procedure to verify that the contents of DRAM are still intact before restoring the internal state of the processor. Since DRAM should contain the same data as when power down occurred, the operating system can return directly to the thread that was running when the device was powered off. As for as the running application is concerned it never ever knew that something happened. When we think about the number of times that we turn off and turn on the battery operated devices, an intelligent shutdown makes a lot of sense, both for reducing power consumption and improving usability.

Minimizing sleep mode power consumption requires analyzing the hardware and determining how to set it to the lowest possible power state. Most battery-operated systems continue to power their general-purpose I/O pins during sleep mode. As inputs, these I/O pins can be used as interrupts to wake up the device; as outputs, they can be used to configure to external peripheral. Careful configuration of how these pins are configured can have large effect on sleep mode power consumption.

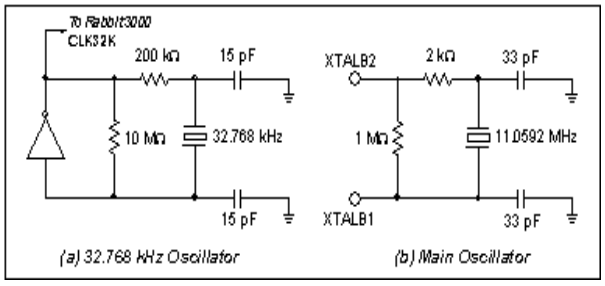

Figure 1: Processor Clock Options.

The embedded processors that are available today are tolerable to industry specific voltages of 3.3V or 5V. These processors are even operating at more than one clock frequency. For example, the embedded Rabbit microprocessor can operate at two clock frequencies, one is the system main clock and other is external clock of 32.768kHz, used for the sleepy mode of operation as shown in Figure 1. As the processor power consumption is directly proportional to the square of the operating voltage, we can save up to 10.9/25 or 43% of power by operating the processor at 3.3V instead of 5V. Moreover the operating current is substantially reduced in proportion to the operating supply voltage. The mathematical calculations of power analysis of the processor with respect to the supply voltage are straightforward. But the designer has to be intelligent enough in identifying the usage of the sleepy mode clock to reduce the processor power consumption, when the task switches in idle mode of operation.

3. Conclusion

The logic analyzer method helps us to identify the amount of CPU spent in the ideal task. This method is so generic that it can be applied to any processor working in any real time operating system platform and performs the accurate timing analysis of the designed application. The precise measurement of the timing behavior of tasks created by the application helps the designer for proper understanding of scheduling strategy. Operating the processor in idle mode when the processor has nothing to do is good choice to reduce the power. But, reducing the number of events by DMA to the external events whose occurrence avoids the processor to wakeup regularly is an attractive solution for huge power saving. Finally, by operating the processor with variable clock speeds and intelligently shutting down the processor also provides reduction in power to the applications that involves processing the interrupts like key detection in a hand held terminals or PDA(s) or to sense from touch screens. However, researchers are working on compilers that can optimize code to reduce the power consumption. In future, designers working with the embedded compilers can expect better software tools that takes care some of the issues of power management techniques of the created application automatically.

ACKNOWLEDGMENT

This work was funded by M.O.U b/n Telecom Division of Electronics Corporation of India Limited (ECIL-TCD/IT&TG), Hyderabad and TIFAC-CORE, SASTRA Deemed University. The authors would like to thank Mr. M. R. K. Naidu, DGM, ECIL and Dr. R. Sethuraman, Vice-Chancellor, SASTRA Deemed University, Thanjavur for their support.

REFERENCES

– Allvin, Kristian and Christer Eriksson, “Constructive Feedback Turns Failure into Success for Pre-Run-Time Scheduled Systems”, Journal of Association of

Computer Machinery, 2001.

- Flavius Gruian, “Hard Real-Time Scheduling for Low-Energy using Stochastic Data and DVS Processors”, Journal of Association of Computer Machinery, 2001.
- Labrosse, Jean J., MicroC/OS-II: The Real Time Kernel, CMP Books, 2002.
- Michael T. Trader, “How to calculate CPU utilization”, Embedded Systems Programming magazine, 2004.
- Richard A. Quinnell, “Designing Embedded Software for Lower Power”, TechOnLine publication, December-2002.
- TCP/IP Kits (Dynamic C SE), Rabbit 3000 microprocessor User’s Manual. www.rabbitsemiconductor.com