

[EPC-43]

**How Software and
Hardware Can Cooperate To
Manage Power Consumption**

John Carbone, Express Logic, Inc.
Bob Boys, ARM UK, Ltd.

September 17, 2008

Outline

- **Power Consumption In Today's Embedded Systems**
 - CPU, memory, other logic all consume power
 - How processor clock speed relates to power
- **Hardware Technologies for Power Management**
 - Sleep, Stop, Standby Modes
 - Clock speed adjustment
- **Software Technologies for Power Management**
 - Small memory
 - Efficient RTOS operations
 - Clock management
- **Conclusions**

Power Consumption In Today's Embedded Systems

- **Sources of Power Consumption**
 - All logic in general
 - Particularly, memory and processor
- **Effect of memory size on power**
 - More memory = more power consumption
- **Effect of processor clock rate on power**

Mode	MHz	MIPS	Power	Ratio	Savings
Normal	5000	5000	1000mW	100.0%	-
Slow Clock	1000	1000	210mW	21.0%	79.0%
Sleep	1	1	14mW	1.4%	98.6%
Power Down	0	0	~0mW	~ 0.0%	~100.0%

Source: ARM/National Semiconductors, 2002
http://www.arm.com/pdfs/NS3003_v1a.pdf

Hardware Technologies for Power Management

- **Power saving technology in physical elements**
 - Gate level power optimization
 - Multiple voltage domains in design
 - Multi-threshold logic
 - Power gating
 - Silicon on insulator
- **Power saving technology in processor**
 - “Run” Mode
 - “Slow Clock” Mode
 - “Sleep” Mode
 - “Power Down” Mode

Processor “Low Power” Modes

- **“Run” Mode**
 - Processor continues to fetch and execute instructions
 - Clock to some of the peripherals can be turned off
- **“Slow Clock” Mode**
 - Processor clock can be slowed down (e.g. multiple clock sources, or using clock divider)
 - Clock to some peripherals can be turned off
 - Performance and power both reduced
 - No latency

Sleep Mode

- **“Sleep” Mode**

- Processor stopped, but timer and interrupts continue to run, or some of the clock signals to processor, system bus and peripherals could be stopped
 - Processor clock can still be running. Since the signal toggling is reduced, power consumption is also reduced.
 - Some parts of the microcontroller could be turned off (e.g. PLL, flash memory)
 - Some sleep modes might require additional processing during wakeup
- Minimal impact on software
- Re-awaken on interrupt or other event
- Low wakeup latency

Power Down Mode

- **“Power Down” Mode**
 - Most logic powered off
 - Greatest power saving
 - Largest latency to re-start
 - Processor state may not be retained, complicating software recovery

Example: ARM Cortex-M3

- **Cortex-M3 Wake-Up Interrupt (WIC) Controller**
 - Allows almost instantaneous (12-18 cycles) return to fully active mode from “Sleep Mode”
 - Wake-up latency is the response time for the circuit to power-up.
 - Cortex-M3 provides sleep and deep-sleep modes. (Interrupt latency of 12 cycles, 18 cycles if powered up instantly).
 - Using Wake-Up Interrupt Controller (WIC) with deep sleep allow further power reduction.
 - Based on power gating implementation flow - processor states stored in state retention logic cell, while majority of the circuit is powered down.
 - The 18 cycle latency consists of the 12 clock cycle interrupt latency, a number of cycle for the power up sequence, which include the "wake up latency".
 - In the 18 cycle figure we assume it takes just 1 cycle for the power to be ready.
 - In some cases the wakeup latency can take several cycles, but normally less than 10 cycles in the latest semiconductor process.

Hardware Summary

- **ARM's Cortex-M3 is an example of how a processor can be designed for low power applications:**
 - Sleep modes can be architecturally defined with sleep instructions and sleep interface.
 - Sleep and Wake can be implemented with low overhead
 - Clock domain separation allows most of the logic to be stopped during sleep
 - Semiconductor technology and EDA tool enhancements also play an important part in lowering power
- **Hardware technology can be controlled by the RTOS**
 - Transparent to the application code
 - Easier for the programmer
 - “Sleep Mode” offers good balance of benefit/transparency

Software's Role In Power Management

- **General**
 - Software can affect memory size
 - Software can help reduce need for high clock rate
- **Software's Role In Enabling Reduction of Processor Clock Rate Dynamically**
 - Run-time clock management
 - Impact on the application
 - How the RTOS can help

Reduce Memory Size

- **Memory size influences power consumption**
 - For small applications, RTOS size can be significant
 - Small memory RTOS can help reduce power consumption
 - Application size can be bigger factor
- **Reduce RTOS Size**
 - Start with a small-memory RTOS
 - Use compiler that generates smaller code
- **Reduce Application Size**
 - Use compiler that generates smaller code
 - Efficient use of registers can reduce memory accesses, thus reduce power consumption in the memory system
 - Good layout of instruction and data can reduce cache misses
 - Use architecture that provides denser code (ie Thumb-2)
 - Reductions here can be very significant

Reduce Processor Clock Rate

- **Using A Lower Processor Clock Rate Means Lower Power (and lower cost)**
 - RTOS
 - A streamlined RTOS can operate on slower processor than a large complex RTOS
 - Fast RTOS services can help reduce power consumption
 - Application
 - Fast application code can help reduce power consumption
 - Compiler
 - Efficient compiler generates faster code to help both RTOS and application use lower MHz processor and increase opportunity to enter “power-saving mode”

Dynamic Management of Power Consumption

- **What if application demands high-performance?**
 - Can't reduce memory or change CPU dynamically, but we can "change the CPU" by varying its clock rate and hence its power consumption.
 - MHz = Performance
 - MHz = Power
- **Many Applications Have Bursts of Activity**
 - Streaming video/audio, rendering, network access
 - Between bursts of high-demand may be periods of inactivity
- **Application can power-down or power-off CPU when not needed**
 - Enter power-saving mode when idle
 - Exit power saving mode upon interrupt or event
 - Keep application coherent (oblivious to slowdown)

Idle Time

- **Lowest priority thread, or OS component**
 - Runs only when nothing else is going on in the system

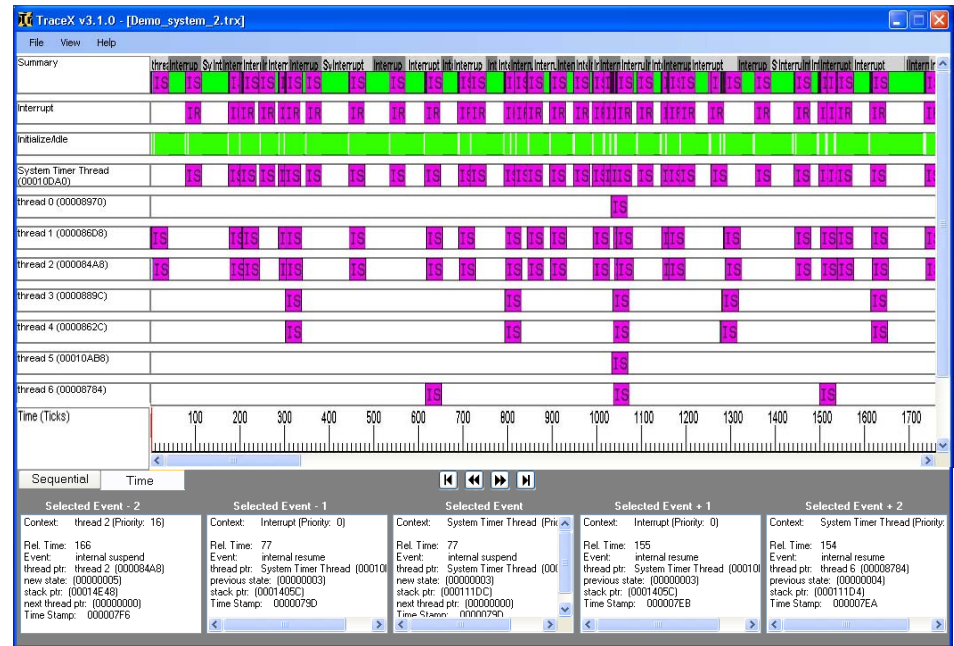
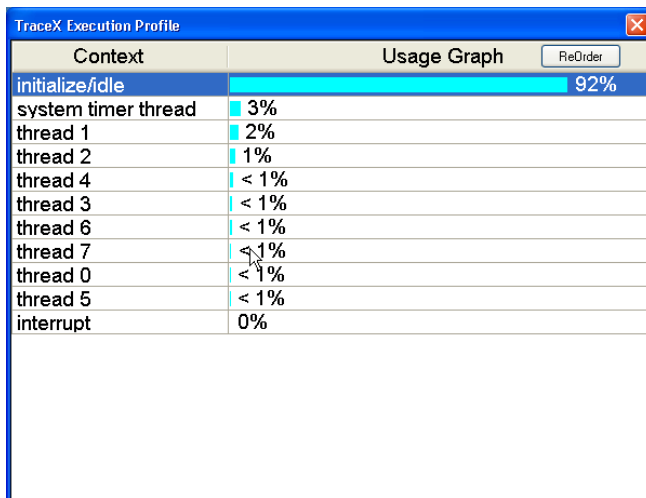
- **Always enter “sleep mode” when idle?**
 - Might not be a good idea
 - What is overhead and frequency of timer interrupt?
 - If “frequent,” reawakening and processing at each timer interrupt might outweigh benefits of power saving. Determine whether gain (power saving) outweighs loss (overhead added to sleep/wake)
 - When is next scheduled event?
 - If “soon” relative to time to sleep/wake, might be little gain.

Timer Overhead

- **Every Nth Clock Tic Generates Interrupt**
 - Prescaler operates to interrupt on 1/n clock tics
 - Reduces interrupt frequency from full clock rate (ie: 25ns), which would be impossible to support, to something less frequent (ie: 10ms)
- **Interrupt processing involves overhead**
 - Update real-time clock
 - Update application timers
 - Check for scheduled event
- **Generally 1-10 μ s to service timer interrupt**

Idle Time

- Many periods of inactivity, opportunities for power saving
- Note 92% system idle time (in this example)



Two Approaches

- **Two approaches to power management through use of processor sleep mode.**
 - Always Sleep When Idle
 - Enter sleep mode whenever there is nothing to do, and awaken from sleep to service timer and external interrupts. Keeps system time intact
 - Managed Sleep
 - Eliminate Timer Interrupts During Sleep
 - Stop CPU and defer timer interrupts
 - Delay next timer interrupt until next scheduled event

Always Sleep When Idle

- **Enter sleep mode immediately upon reaching idle state**
 - Pros
 - Power consumption reduced during periods of inactivity
 - Easy to implement in RTOS or application
 - Cons
 - Incurs overhead re-awakening and servicing timer interrupt to update timers
 - Net gain/loss depends on frequency and relative magnitude of these overheads
 - Without extremely low overhead mechanisms, this approach might not work well
 - Fast RTOS timer and interrupt servicing helps make this approach viable, but hardware mechanism is key

Managed Sleep

- **Defer timer interrupts and then enter sleep mode**
 - Pros
 - Avoids overhead of servicing timer interrupts
 - Enables processor to sleep for longer continuous periods, rather than to re-awaken for each tic
 - Cons
 - Complicates activating next timed event
 - Introduces need to adjust timer upon wakeup
 - Requires 2 services from the RTOS
 - Net gain/loss depends on overhead of sleep enter/exit, which is weighed against idle time until next timed event.
 - Fast RTOS services and timer/interrupt handling help make this approach viable

RTOS Services To Support Managed Sleep

- **(1) Find Next Scheduled Event**
 - The first service tells the application the number of timer ticks until the next scheduled event.
 - The application would call this service prior to entering low power mode.
 - The timer would be re-programmed to generate its next tick interrupt just prior to the next scheduled event
 - This avoids missing a scheduled event even though timer interrupts are eliminated

Upon Reawakening

- **(2) How Many Tics Skipped**

- The second service, used upon re-awakening, informs the application how many ticks (at the original tic rate) were skipped
- The RTOS then updates system time to correct for the tics that were skipped, and resets the system timer to the previous tick rate
- This reestablishes all internal timers to the state at which they would have been had the system never gone into low power mode
- Application continues normally

Conclusion

- **Processor technology** addresses power management through efficient design and clock adjustment services
- **An efficient compiler** can reduce size and execution time, both helpful in reducing power consumption
- **An RTOS** can exploit advanced processor technology to enable power conservation with little, if any, application software impact