Power Consumption

Overview

In this lesson we will

- ✓ Identify the different sources of power consumption in embedded systems.
- ✓ Look at ways to measure power consumption.
- ✓ Study several different methods for managing power consumption.

Introduction

Today more and more embedded applications
Targeted towards
  Small hand held or other types of portable devices
Common thread through all such applications
  Need to have long battery life
    Translates to low power consumption
Power consumption can be attacked in several ways
  Certainly hardware solution
    Low power devices
    Turning portions of system off
      ACPI – Advanced Configuration and Power Interface
  Surprisingly have software contribution as well

Let’s look at each and begin with a view into the software

Software

There are a number of places that we can attack
  From software point of view
Initial places to look

- The algorithms that we use
- Location of code
  Memory accesses can have significant impact on power
- Using software to control subsystems

As we have been stating
  To analyze then control particular aspect of performance
    Must be able to measure that aspect
      Both before and after modification

Measuring Power Consumption

For the moment
    Will assume goal is to reduce power consumed by processor
        Processor can be stand alone or softcore on FPGA

To such an end
    Measuring power consumption is two step process

1. Identify the portion of code to be analyzed
    Typically this will be a loop
        Doesn’t need to be
   Measure the current consumed by the processor
    While the code is being exercised

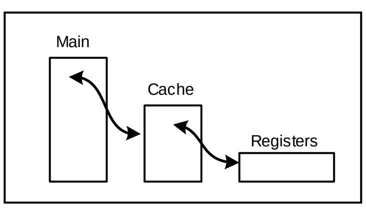
2. Modify the loop such that the code comprising the loop is disabled
   Ensure that the compiler hasn’t optimized loop out
   Measure the current consumed by the processor

Once we have identified power consumed
    Next step is to try to reduce if appropriate

Studies have identified several software factors
    That contribute to processor power consumption
        Among the contributors we find

- The kind of instruction
- The collection or sequence of instructions executed
- The locations of the instructions and their operands

Memory system and transfers in and out
    Have been shown to be most expensive operation
        Performed by processor
    Here memory is referring to main memory not cache
        This is the DRAM in our system
    Hierarchy will typically look like



Cache is high speed memory and is closer to the processor

Using simple addition operation as reference we find

Operation	*Relative Power Consumption*
16 Bit Add	1
16 Bit Multiply	3.6
8x128x16 SRAM Read - *Cache*	4.4
8x128x16 SRAM Write - *Cache*	9
I/O Access	10
16 Bit DRAM Memory Transfer - *Main*	33

Evident from table

- Using cache can have significant affect on power consumption
  - Assumes a cache hit
- Cache miss requires main memory access
- SRAM generally consumes more power than DRAM
  - On per cell basis
  - Cache is generally SRAM
- Want to optimize size of cache\
  - Want smallest that provides desired performance
  - Almost becomes empirical trade-off

Other optimizations

1. Power aware compilers
   - Take an instruction level view of problem
     - Modify schedule of bus activity
   - Bus drivers consume a lot of power

2. Use registers efficiently
   - Bring value into register and leave there for reuse
     - Don't move in and out
       - Read / write operations

3. Look for cache conflicts and eliminate if possible
     - Conflicts are data or instructions
       - That must be moved in or out of cache
     - For instruction conflicts
       - Rewrite if possible
       - May have to move code

     - Scalar data conflicts

Move data to different locations
Arrayed data
Move to alternate location
Change access pattern

4. Unroll loops
   Must be careful that unrolled loop doesn't result in cache misses

5. Eliminate recursive procedures or repeated function calls where possible
   Eliminates overhead of function call

Hardware

Another technique for managing the power consumption
In embedded applications
Draws from familiar schemes used at home
Turn off the portions of system not being used
Such a scheme has been used for years
In space program
Satellites
Orbital and interplanetary
Shuttle
Earlier Mercury, Gemini, Apollo
Therein hardware
Battery powered
Must be recharged
Done via solar panels of one form or another

Today can fly from Seattle to Japan in 14 hours
Laptop computer or other such tools
Typical battery life 3-5 hour
Yep a laptop is still an embedded application
We are in continuous race between
Battery technology
Demand for more and more powerful features
All such features require power

Power Management Schemes

To begin to address problem
- As part of design
  - Formulate power management strategy

✓ On one extreme
- Turn power off
  - In such state
    - Power consumption limited to leakage
- As with other metrics
  - ➢ Sets a lower bound on consumption

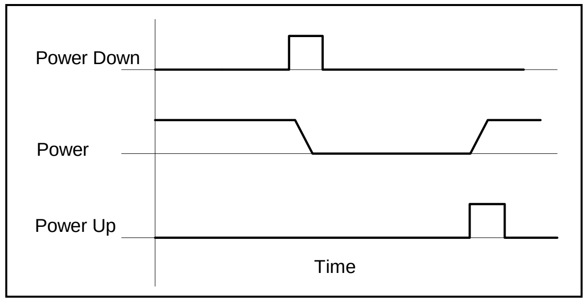
✓ Opposite extreme
- Power to all parts of system on and all parts operating
  - In such state
    - Power consumption approaches maximum
- Such condition sets upper bound
  - Softer than lower bound
    - See earlier discussion on software affects
  - Goal is somewhere in middle
    - Governed once again by requirements specification
    - Based upon such a goal
      - We segregate system components into two categories
        - Those that must remain powered up
          - Referred to as static components
        - Those that may be powered down
        - Referred to as dynamic components

Such a scheme sounds simple …and is at the high level

Like everything else we are doing
- We have certain trade-offs

We must
1. Decide which portions of the system to power down
   These may be
   - All dynamic components
   - Subgroups based up need or not need
2. Recognize that components cannot be shut down instantly
3. Recognize that components cannot be powered up instantly

These factors can be expressed with a simple first cut graphically as



Let’s consider a topographic mapping satellite application

As satellite is circling the earth collecting data

Data is sent to ground station at known points in orbit

When over appropriate station

No reason to keep transmitter powered up

When not in position to transmit

Further timing of orbit known with sufficient resolution

Know in advance when will need to transmit

After passing ground station

Shut down transmitter

Re-enable shortly before reaching next download point

Locations of each ground station known in advance

1. Such fixed schedule scheme is among simplest

   Can be very effective

   Observe similar to

   Round robin schedule with no preemption

2. Next level of sophistication

   Recognize that schedule may not be fixed

   Problem now moves from deterministic to probabilistic

   Use knowledge of

   Current history

   Understanding of problem

   To anticipate when to shut dynamic portions of system down

   Denoted predictive shutdown

   Observe that such a scheme commonly used in

   Branch prediction logic in instruction prefetch pipeline

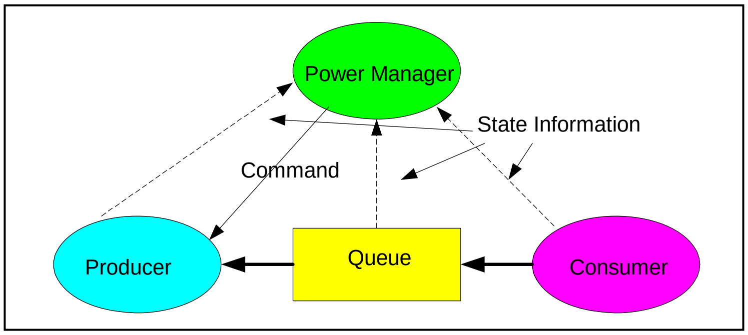
Managing if-else constructs
Using such a scheme
Subject to shutdown or restart too early

3. Related idea
Rather than set schedule
Control algorithm with associated timer
Monitors activities of devices to be dynamically controlled
If timer expires
Device is powered down
Device reactivated on demand
We’ve already used such a scheme in a watchdog timer

4. Next level of sophistication
Draws from basic queuing theory
Under such a scheme we have
A resource or producer
Service provided by system whose power is being controlled
A consumer
Portion of the system that needs the service
A queue of service requests
A power manager
Monitors behaviour of system
Producer
Consumer
Queue
Can build schedule using Markov modeling
Statistical technique
Forecast future behaviour of variable or system
Who's current state of behaviour
Does not depend upon past
Random like flipping an honest coin

Approach maximizes
System computational performance
While satisfying specified power budget

Simple data / control flow diagram appears as



Let's look at an example of simple power management

- ✓ Queue
  - Queue of requests for power
- ✓ Consumer
  - Issues requests for power

Operating system responsible for dynamically controlling power
  In simple I/O subsystem
Dynamically controlled portion
  Supports two modes
    Off and On
Dynamic subcomponents
  Terminology
    joule = 1 watt sec
    power = joule / sec

  Consume 10 watts when on and 0 watts when off
    10 watts when on then in 25 sec since always on
      10 watts • 25 sec = 250 joules

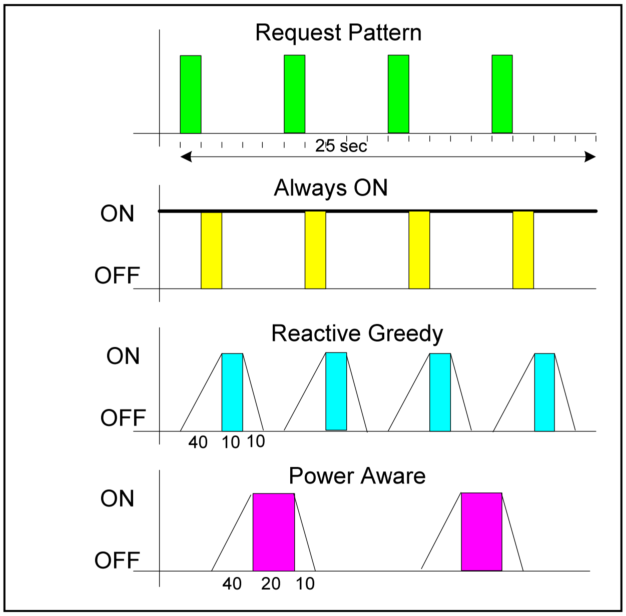
Switching

- 2 seconds and 40 joules
  - Switch from OFF state to ON state
- 1 second and 10 joules
  - Switch from ON to OFF
- Request has period of 25 seconds

Graphically we illustrate 3 alternative schemes

Observe that we have the same average throughput

- Substantially reduced power consumption



Policy	Energy Consumed in 25 sec	Average Latency Per Request
Always ON	250 J (10 x 25 sec) (10 when on x 25)	1 sec
Reactive Greedy	240 J (40+10+10)x2 (60x4)	3 sec (2+1)x4/4
Power Aware	140 J (70 x 2)	2.5 sec (2+1)/2 +1

5. Advanced Configuration and Power Interface – ACPI

Industry standard power management

- Initial application was to PC
  - More specifically windows
  - Currently targeted to wider variety of operating systems

Standard

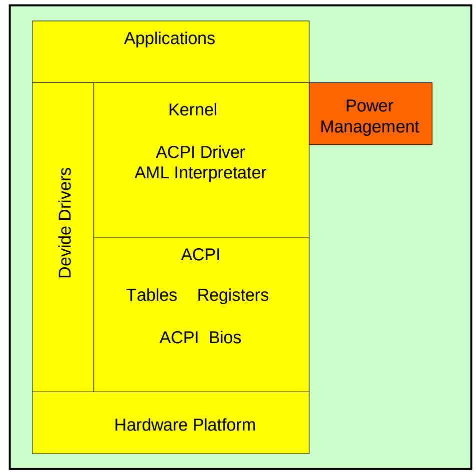
- Provides some basic power management facilities
- Provides interface to the hardware
- Software more specifically the operating system on a system
  - Provides power management module
  - It is the responsibility of the OS
    - Specifying the power management policy for the system

The operating system uses the ACPI module

To send required controls to the hardware

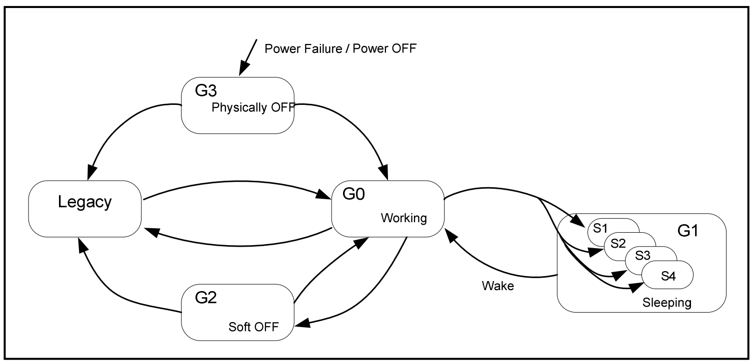
Monitor the state of the hardware

As input to the power manager



We express the scheme in the following

- Block diagram
- State diagram



Standard supports five global power states

1. G3 – hard off or full off
   Defined as physically off state
   System consumes no power
2. G2 – soft off requires full OS reboot to restore system to full operational condition
   Substates
   S1 – low wake-up latency
   Ensures no loss of system context
   S2 – low wake-up latency state
   Has loss of CPU and system cache state
   S3 – low wake-up latency state
   All system state except for main memory is lost
   S4 – lowest power sleeping state
   All devices are off
3. G1 – sleeping state
   System appears to be off
   Time required to return to working condition
   Inversely proportional to power consumption

4. G0 – working state in which system is fully usable
5. Legacy state
    a. System does not comply with ACPI

Trade-offs

Often times performance is optimization issue

Involves trading several contradictory requirements

- Speed
- Memory size
- Cost
- Weight
- Power

Time must be spent up front to thoroughly understand

- Application
- Constraints

Summary

In this lesson we

- ✓ Identify the different sources of power consumption in embedded systems.
- ✓ Look at ways to measure power consumption.
- ✓ Study several different methods for managing power consumption.