System-Level Low-Power Techniques

Naehyuck Chang
Dept. of EECS/CSE
Seoul National University
naehyuck@snu.ac.kr

Seoul National University

Contents

Definition

- Systems and components are
  - Designed to deliver peak performance
  - Not needing the peak performance most of the time
- Slack and idle time exist
- Dynamic power management (DPM)
  - In wide-sense definition, DPM includes DVS
  - Shut-down idle components
- Dynamic voltage scaling (DVS)
  - Slow-down components, by scaling down frequency and voltage
  - DFS and DVFS

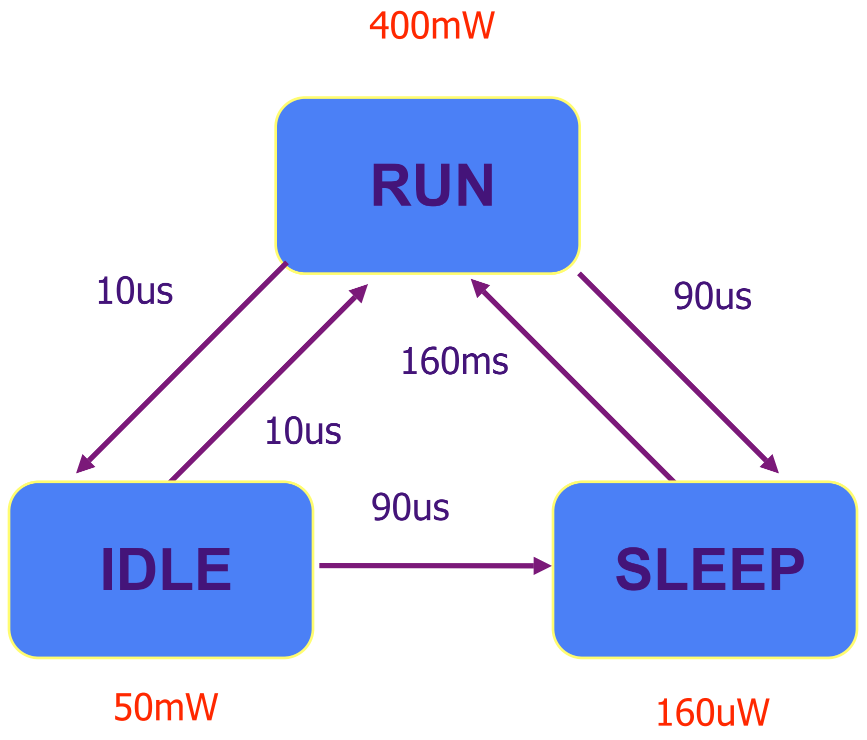
Power manageable components

- Components with several internal states
  - Corresponding to power and service levels
- Abstracted as power state machines
  - State diagram with:
    - Power and service annotation on states
    - Power and delay annotation on edges

ELPL Embedded Low-Power Laboratory

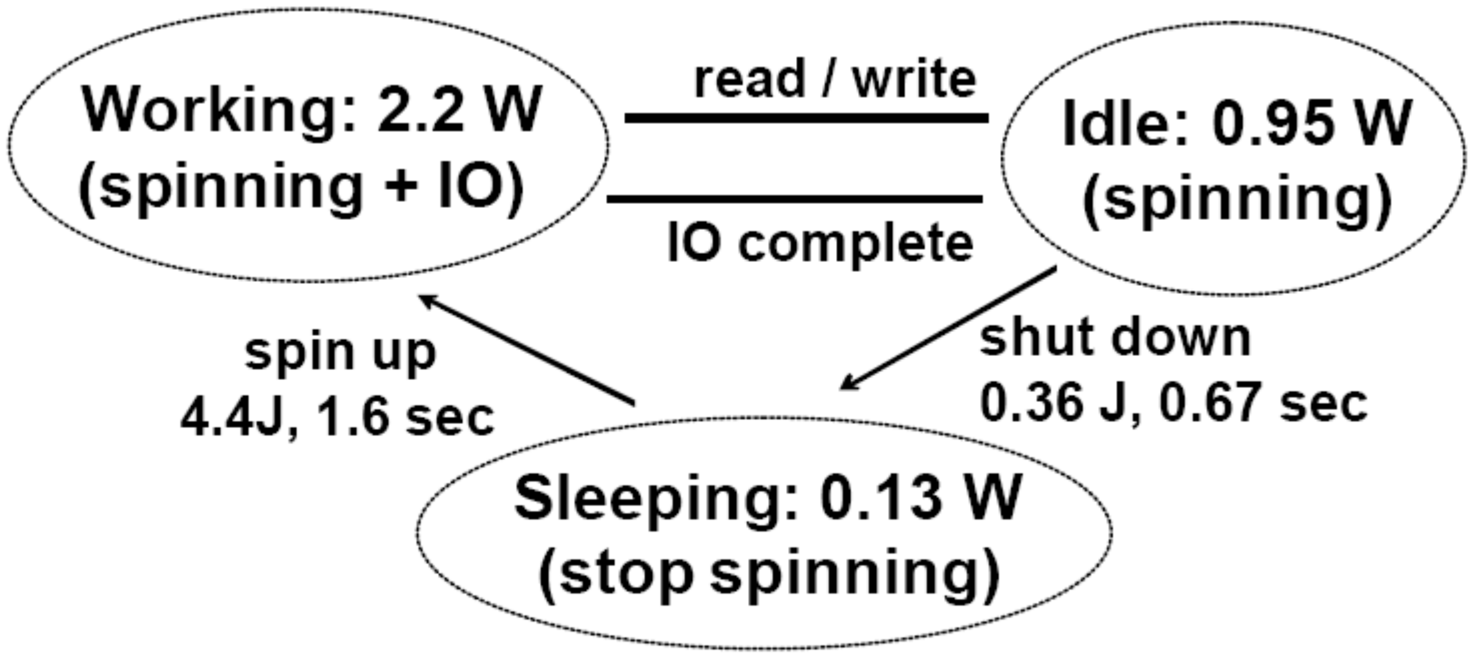
Example: SA1100

- RUN
  - Operational state
- IDLE
  - A software routine may stop the CPU when not in use, while monitoring interrupts
- SLEEP
  - Shutdown of on-chip activity



Another example: hard disk drive

- Model: (Fujitsu MHF 2043 AT)

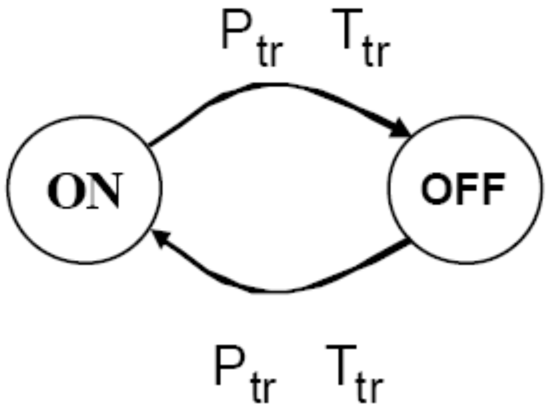


Structure of power-manageable systems

- System consists of several components:
  - E.g., Laptop: processor, memory, disk, display, and so on
  - E.g., SoC: CPU, DSP, FPU, RF unit, and so on
- Components may
  - Self-manage state transitions
  - Be controlled externally
- Power manager (PM)
  - Abstraction of power control unit
  - Implemented typically in software
  - Energy consumption of PM is negligible

The applicability of DPM

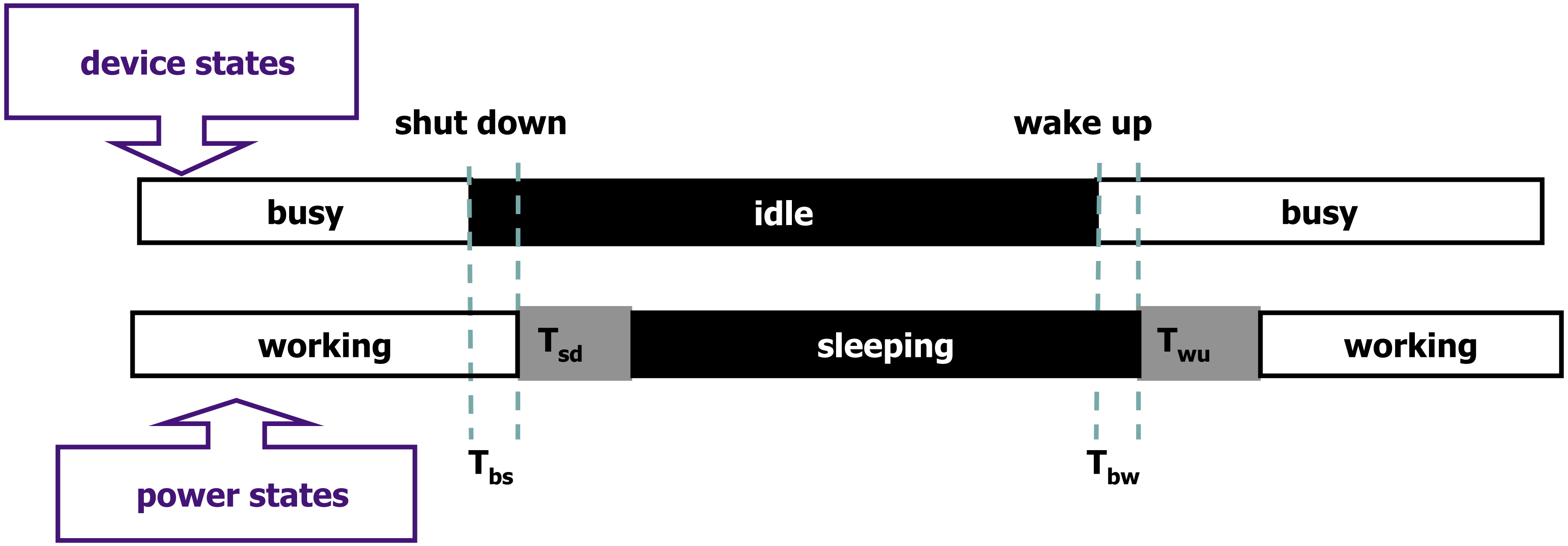
- State transition power (P_{tr}) and delay (T_{tr})
- If $T_{tr} = 0$, $P_{tr} = 0$ the policy is trivial
  - Stop a component when it is not needed
- If $T_{tr} \mathrel{!}= 0$ or $P_{tr} \mathrel{!}= 0$
  - Shutdown only when idleness is long enough to ar
  - What if T and P fluctuate?



ELPL Embedded Low-Power Laboratory

The opportunity

- Reduce power according to workloads
- Shutdown only during long idle time



T_{sd}: shutdown delay

T_{bs}: time before shutdown

T_{wu}: wakeup delay

T_{bw}: time before wakeup

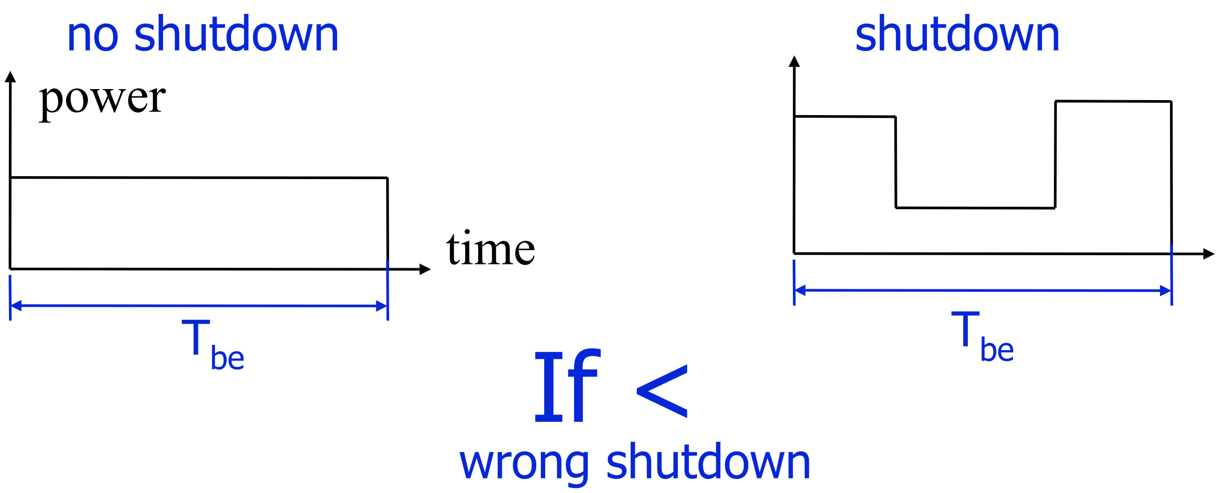
The challenge

Is an idle period long enough for shutdown (T_{be})?

Predicting the future!

Shutdown criteria

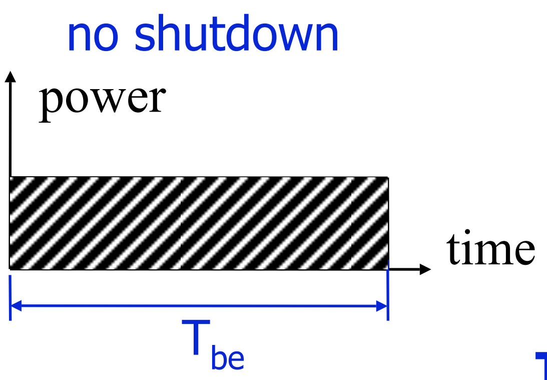
- Break even time: T_{be}
- Shortest idle period for energy saving

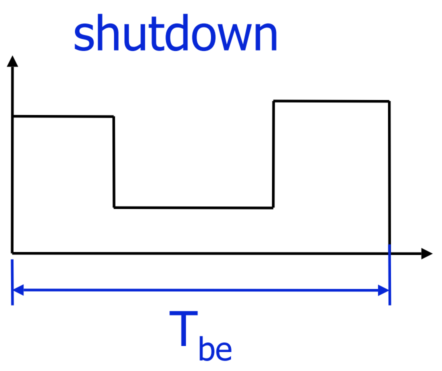


Idle period shorter than T_{be} is useless for energy saving

Shutdown criteria

- Break even time: T_{be}
- Shortest idle period for energy saving



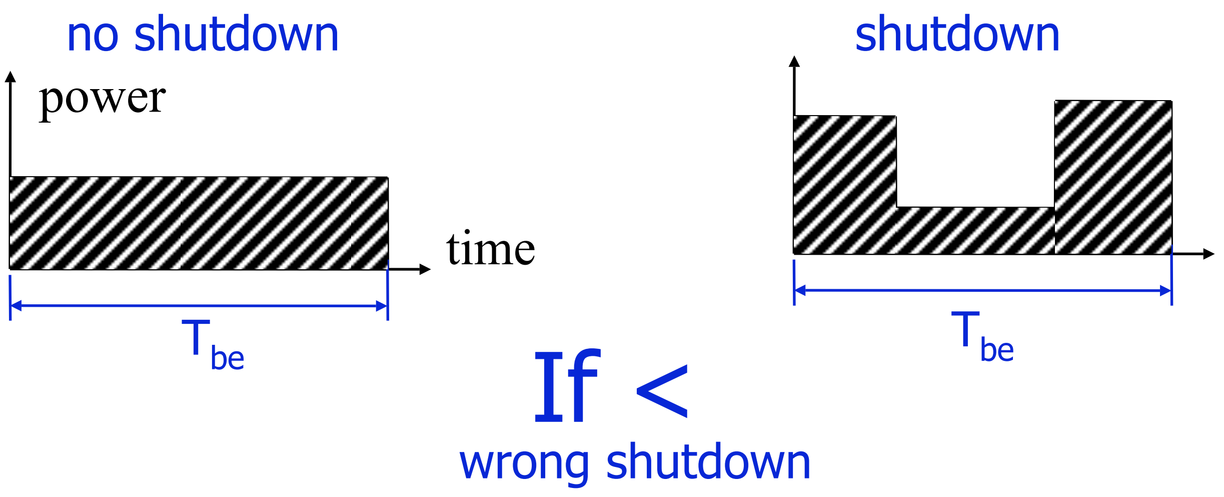


If <
wrong shutdown

Idle period shorter than T_{be} is useless for energy saving

Shutdown criteria

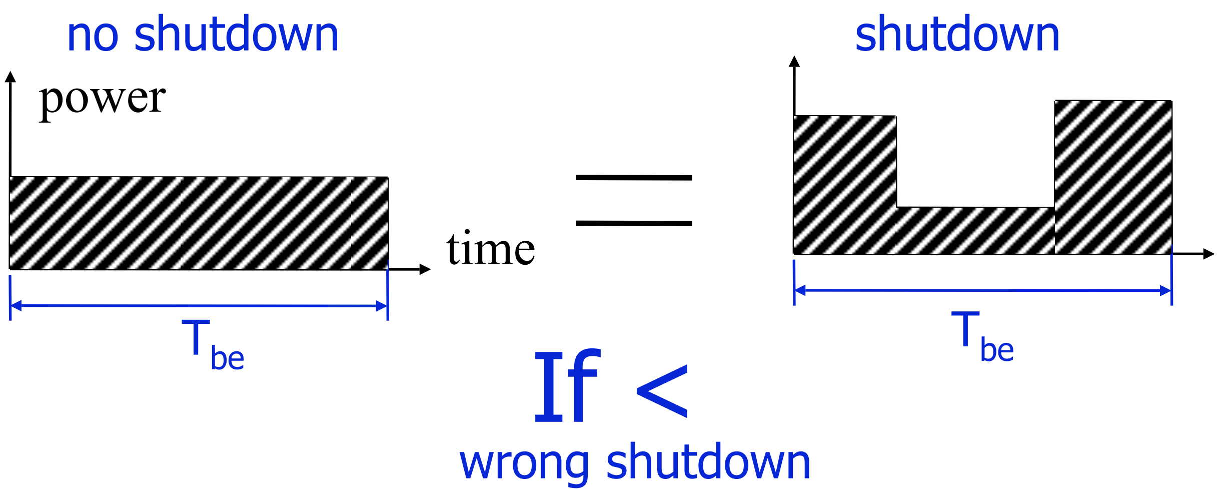
- Break even time: T_{be}
- Shortest idle period for energy saving



Idle period shorter than T_{be} is useless for energy saving

Shutdown criteria

- Break even time: T_{be}
- Shortest idle period for energy saving



Idle period shorter than T_{be} is useless for energy saving

System break-even time: T_{BE}

- Minimum idle time for amortizing the cost of component shutdown

$$T_{BE} = T_{tr} + T_{tr}\frac{P_{tr} - P_{on}}{P_{on} - P_{off}}$$

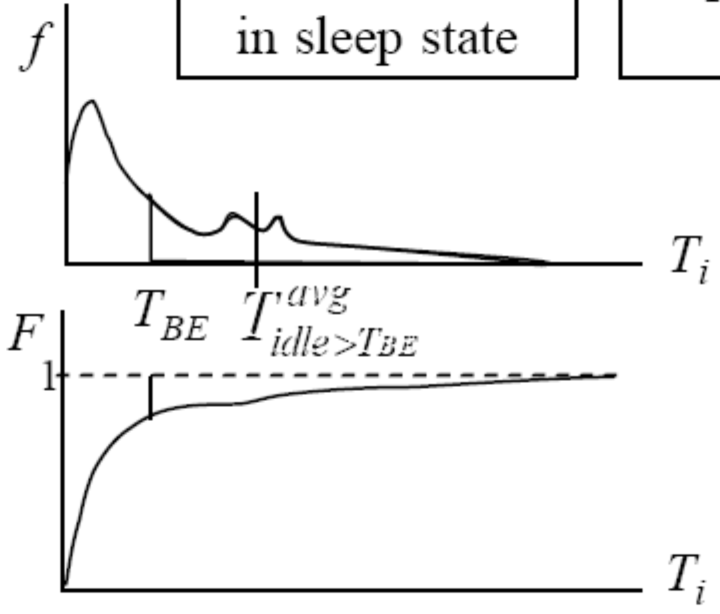
Transition delay (T_{tr})

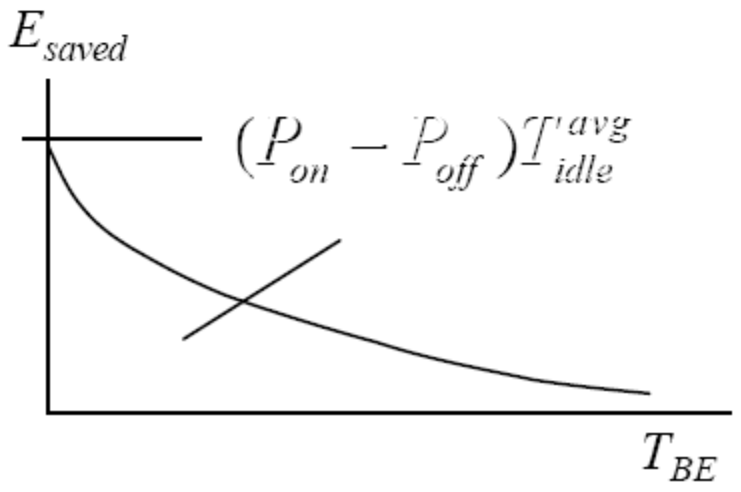
Transition power (P_{tr})

Sleep power (P_{off})

Effect of T_{BE} and $F(T_{idle})$ on power savings

$$E_{saved} = \underbrace{(P_{on} - P_{off})}_{\text{Power reduction in sleep state}} \cdot \underbrace{(T^{avg}_{idle>T_{BE}} - T_{BE})}_{\text{Expected useful idle time}} \cdot \underbrace{(1 - F(T_{BE}))}_{\text{Probability of going to sleep}}$$

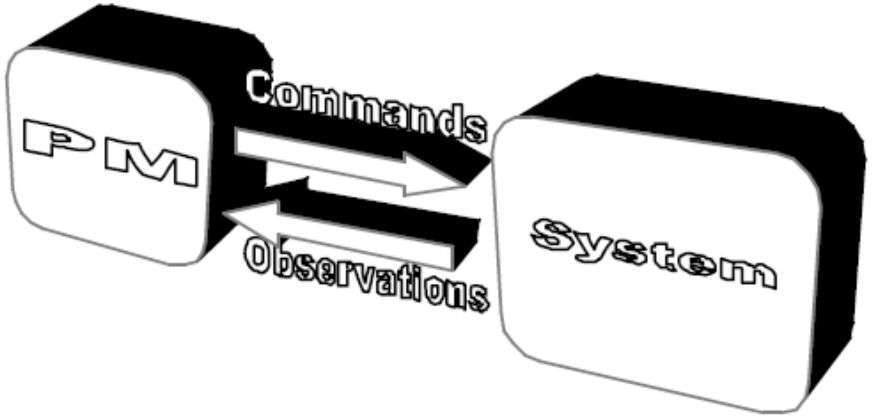




When to use power management

- When $T_{BE} < T^{avg}_{idle}$
  - Average idle periods are long enough
  - Transition delay is short enough
  - Transition power is low enough
  - Sleep power is low enough
- When designing system for a known workload
  - Criteria for component specification and design

Controlling PM systems



- DPM is a control problem: a policy is the control law
  - Collect observations
  - Issue commands
- Optimal control
  - Synthesize the *"best"* controller (PM)

Categories of DPM techniques

- Timeout : [Karlin94, Douglis95, Li94, Krishnan99]
  - Shutdown the system when timeout expires
  - Fixed vs. adaptive
- Predictive : [Chung99, Golding95, Hwang00, Srivastava96]
  - Shutdown the system if prediction is longer than Tbe
- Stochastic : [Chung99, Benini99, Qiu99, Simunic01]
  - Model the system stochastically (Markov chain)
  - Policy optimization with constraints
  - Trade off between energy saving and performance
  - Non-deterministic decision
  - Discrete time model/continuous time model
  - Superior to predictive and timeout

ELPL Embedded Low-Power Laboratory

Contents

Time-out method (I)

- Shut-down the system if the idle time is longer than the pre-defined threshold
  - widely used technique
    - PC, monitor, disk, …
- Rationale
  - When $\mathbf{T_{idle} > T_{TO}}$ it is likely that: $\mathbf{T_{idle} > T_{TO} + T_{BE}}$
- How to determine the $\mathbf{T_{TO}}$?
  - Choice of T_{TO} is critical
    - Large is safe, but it could be useless
    - Too small is highly undesirable

Time-out method (II)

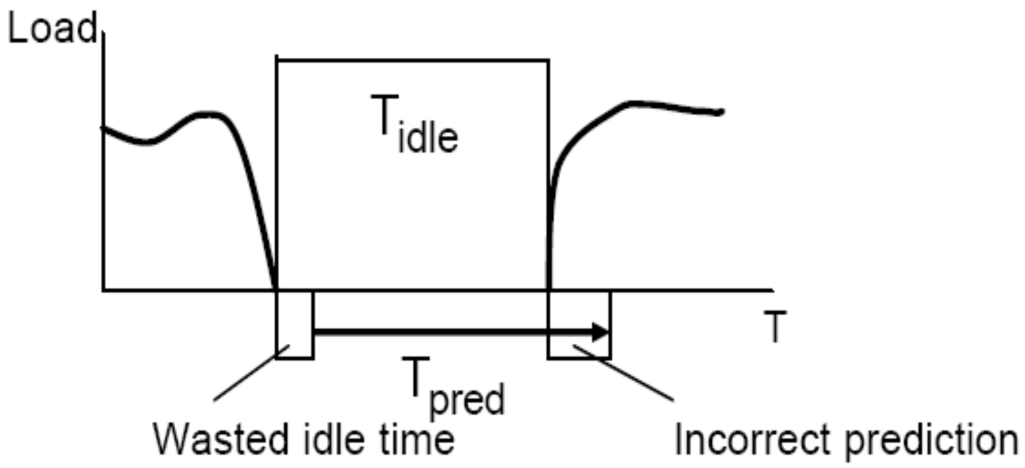
- **Two typical ways to control the time-out value**
  - Fixed time-out
    - independent to the workload
  - Adaptive time-out
    - Varies time-out value depending on the workload
- Limitations
  - Performance penalty for wake-up is paid after every shutdown
  - Power is wasted during T_{TO}
  - No way to handle them

Contents

- Dynamic Power Management
  - DPM introduction
  - Time-out method
  - Predictive method
  - Stochastic method
- Dynamic Voltage Scaling
  - DVS introduction
  - intra-task DVS
  - inter-task DVS

Predictive method

- Observe time-varying workload
  - Predict idle period $T_{pred} \sim T_{idle}$
  - Go to sleep state if T_{pred} is long enough to amortize state transition cost
- Main issue: prediction accuracy



When to use predictive methods?

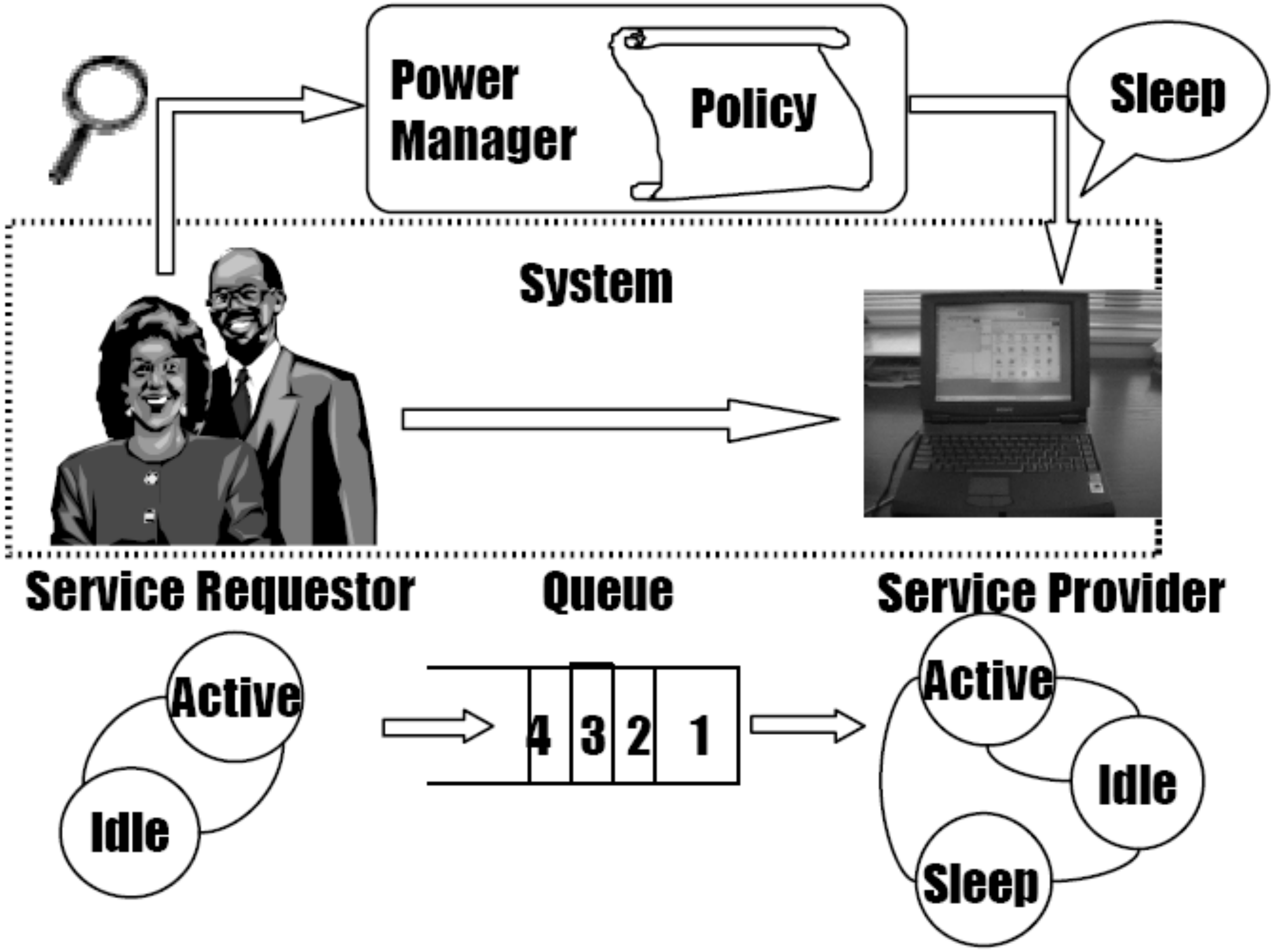
- When workload has memory
  - Implementing predictive schemes
    - Predictor families must be chosen based on workload types
    - Predictor parameters must be tuned to the instance-specific workload statistics
    - Low cost
    - When workload is non-stationary or unknown, on-line adaptation is required

Contents

Stochastic method

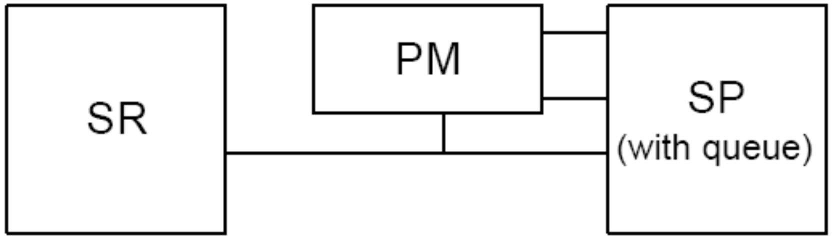
- Recognize inherent uncertainty
  - Exact prediction of future events is impossible
  - Abstraction of system model implies uncertainty
- Model components,system and workload as stochastic processes
- Expected values of cost metrics are optimized
  - Power
  - Latency

System modeling



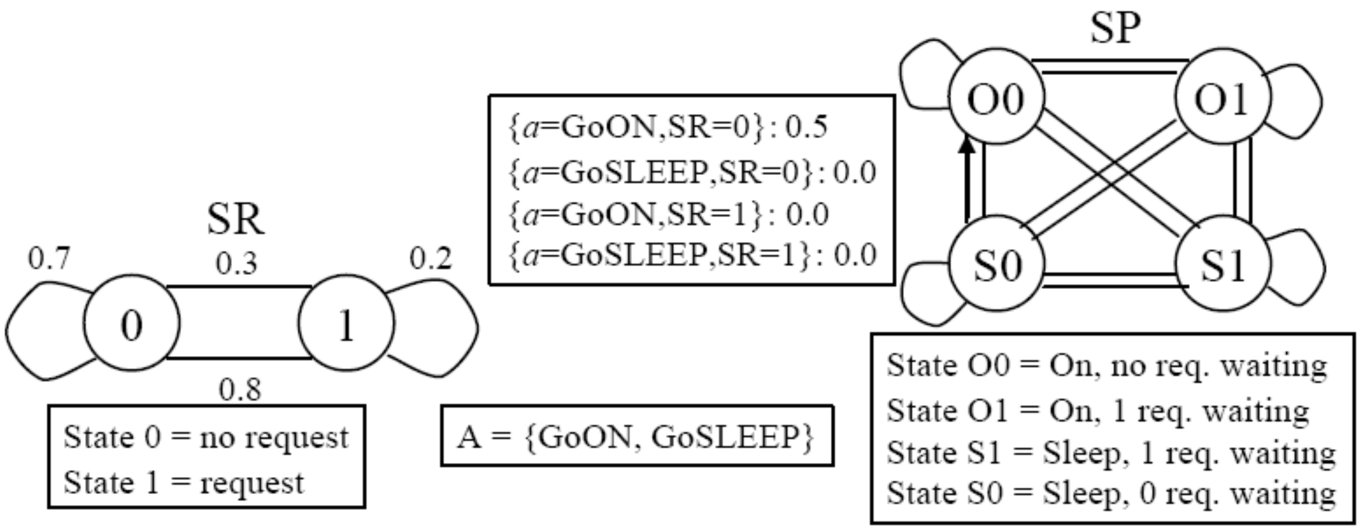
Controlled Markov Processes

- Component and workload modeled as Markov chains
  - Component is called service provider (**SP**)
  - Workload is called service requester (**SR**)
  - System (**S**) is the combination of SR and SP (with queue)
- SP is a controlled Markov chain:
  - State transition probabilities depends on commands
- The power manager (PM) observes the state of the system and issues commands to control evolution



Discrete-time, finite-state CMPs

- Discrete time t = 1, 2, …
  - System evaluated at periodic time points
- SR and SP are modeled by Markov chains
- PM can issue a finite number of commands *a* in *A*



State 0 = no request
State 1 = request

{a=GoON,SR=0}: 0.5
{a=GoSLEEP,SR=0}: 0.0
{a=GoON,SR=1}: 0.0
{a=GoSLEEP,SR=1}: 0.0

A = {GoON, GoSLEEP}

State O0 = On, no req. waiting
State O1 = On, 1 req. waiting
State S1 = Sleep, 1 req. waiting
State S0 = Sleep, 0 req. waiting

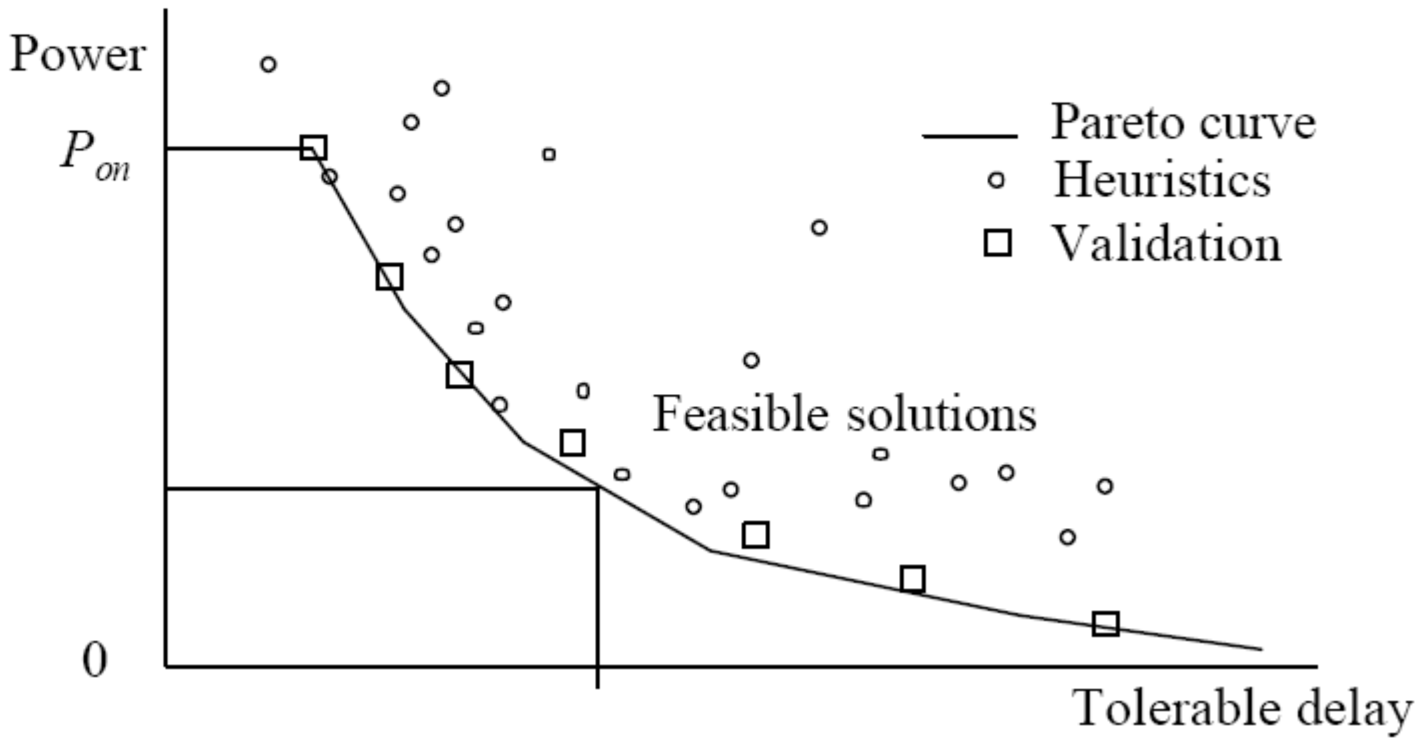
Power management policies

- PM observes system state and issues a command
- A policy is a sequence of commands
- A Markovian policy yields commands as function of system state (and not previous history)
- A deterministic policy
    - For each state s in *S*, policy specifies command *a* in *A*
- A randomized policy
    - For each state *s* in *S*, policy specifies the probability of issuing command *a*
- A stationary policy
    - The policy does not change with time

PM policy optimization

- Solve a stochastic optimal control problem:
  - Find a policy that
- Minimizes power cost function
- Satisfies performance constraints
- Dual formulation
- Key result for CMPs:
  - **Optimum policy is stationary, Markovian and randomized**
  - **Policy optimization can be reduced to a LP and solved exactly and efficiently**

Power-performance trade-off



CMP advantages

- Constrained optimization:
  - Energy/performance (latency) trade-off
- Global view of the system:
  - Workload and component models
- Optimum policy is captured by commands:
  - Control policy is a table
  - Policy implementation is easy
- Policy computation can be cast as a linear program and solved exactly and efficiently

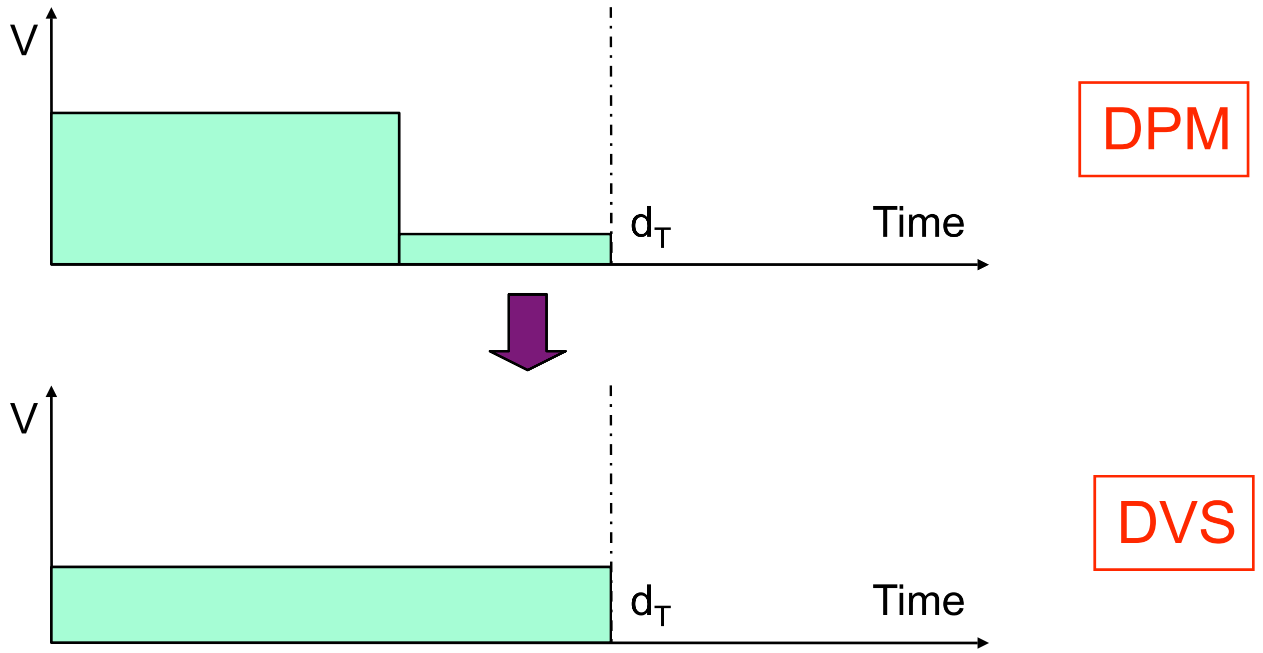
CMP limitations

- Discrete-time models require periodic evaluation
  - Use continuous-time Markov models
- Event-driven paradigm
- Stochastic distributions:
  - Geometric and exponential distribution of events may not fit component and workload
  - Use (time-indexed) semi-Markov models
- Non-stationary workloads
  - Use adaptive schemes

Contents

Dynamic Voltage Scaling definitions

- For a given task T and its deadline d_T
  - Reduce the voltage and frequency to finish task T as close as to its deadline d_T(but, not over the d_T)



Alpha-Power Model

- Simple hand calculation model that empirically fits the real data

Measured data

$$I_{DS} = \underset{\substack{\uparrow \\ \text{Measured data}}}{K_S} \frac{W}{L} (V_{GS} - V_T)^{\alpha}$$

  - α is close to 1 than 2, which is approximately 1.25, and continue to approach to 1 as technology scales

$$I_{ON} = I_0 (S\alpha)^{-\alpha} (V_{GS} - V_T)^{\alpha}$$

$$I_{sub} = I_0 e^{-\alpha} e^{\frac{V_{GS} - V_T}{S}}$$

$$\text{Delay} \propto \frac{V_{DD}}{(V_{DD} - V_T)^{\alpha}}$$

DVS effect

- Exploits under-utilized resources by reducing f and V
  - Power is proportional to frequency and voltage^2
  - Energy is proportional to power and time
    - Frequency scaling does not have an impact on energy
- Overhead: typically tenths of microseconds
  - Wait until voltage is stabilized
  - Wait until frequency is stabilized
- Order of change
  - When f is going up: change voltage first
  - When f is going down: change frequency first

DVS supporting HW block diagram

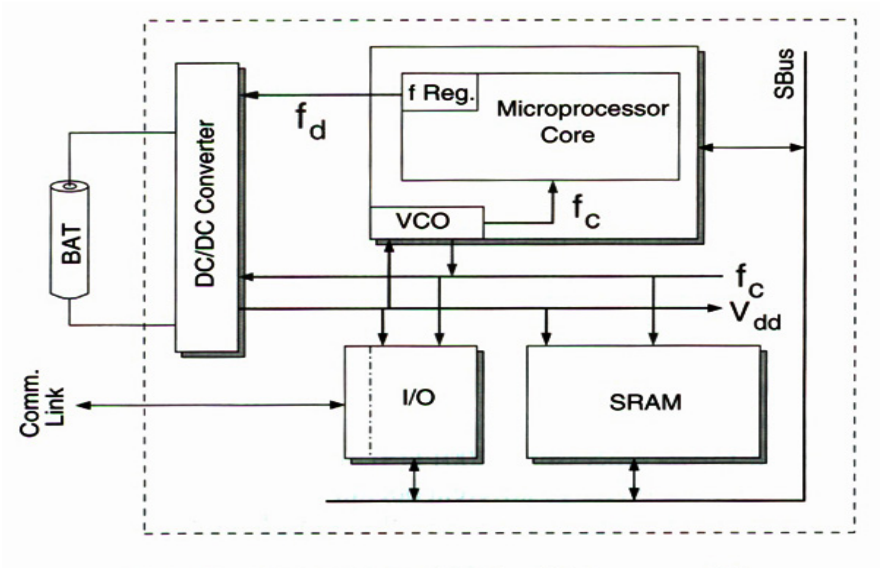
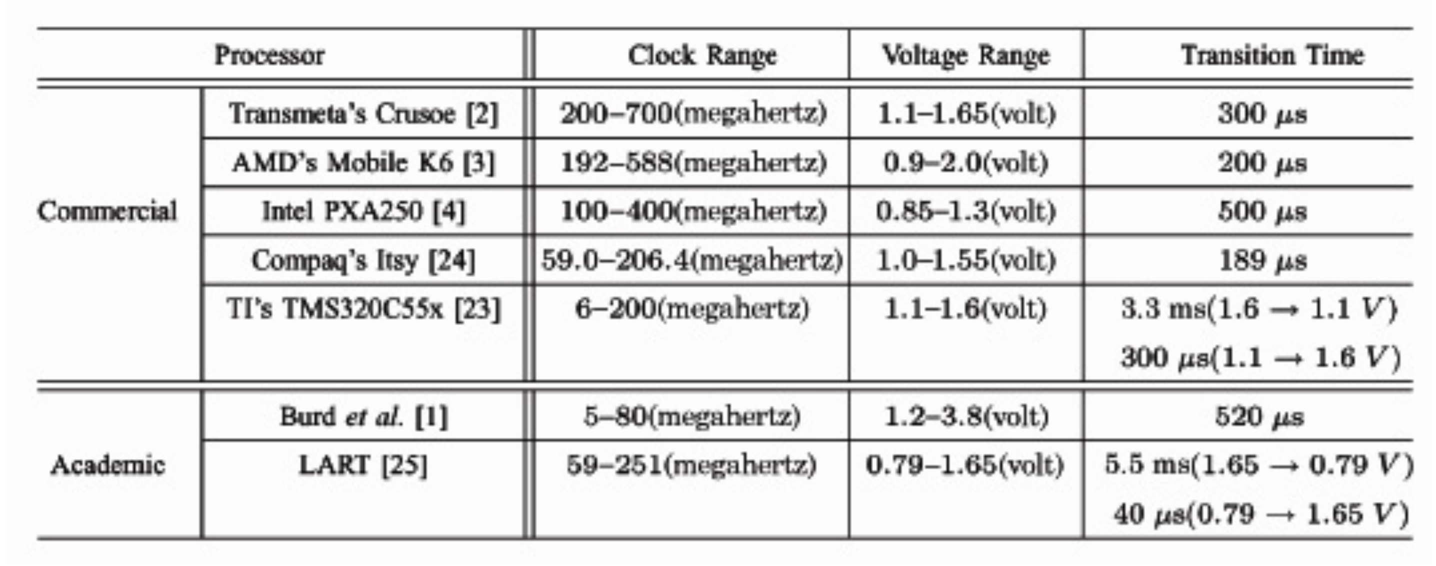


Figure 2.4. Block diagram of DVS-enabled processor [36]

- Procedure (when f_d is larger)
  - Processor writes the desired frequency to frequency register (f_d)
  - DC/DC converter compares f_d with f_c (current frequency)
  - DC/DC converter changes VDD to a certain value paired with f_d
  - VCO adapts the system clock

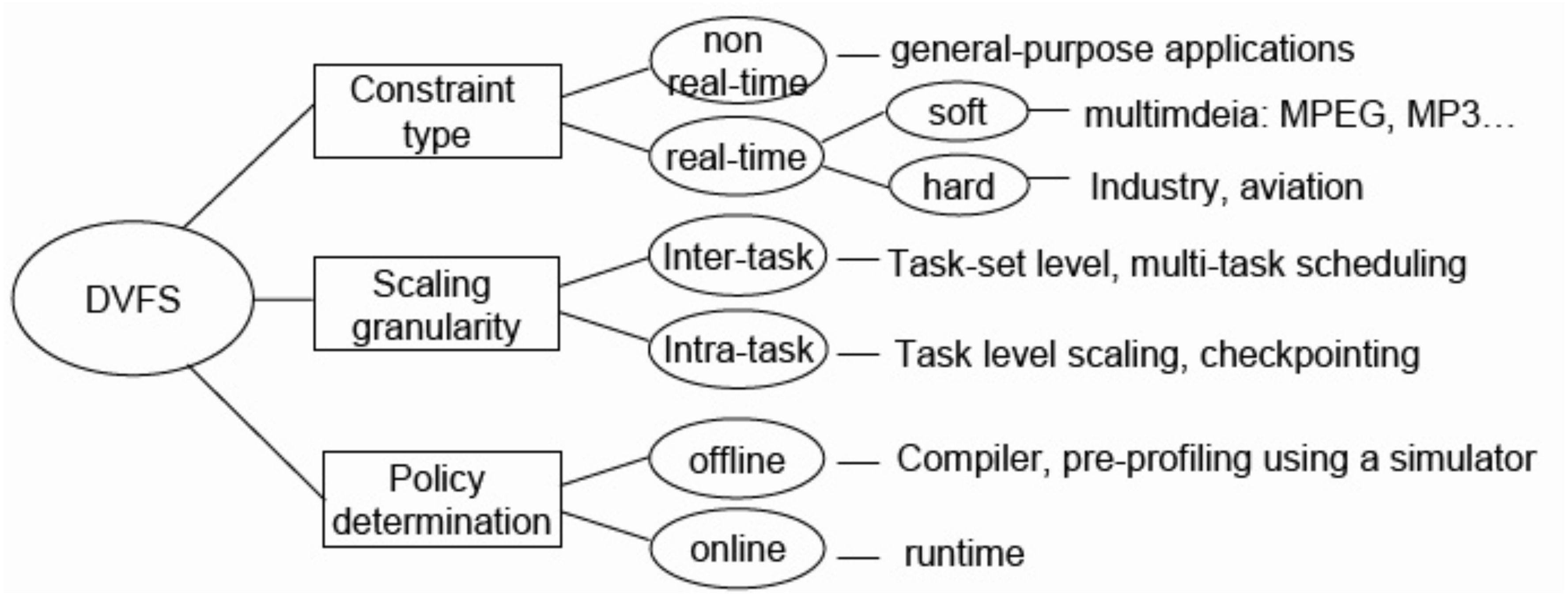
Processors supporting DVS

- Recent processors such Xscale and ARM11 series also support DVS
  - IEM: Intelligent Energy Manager from ARM



Processor		Clock Range	Voltage Range	Transition Time
Commercial	Transmeta's Crusoe [2]	200–700(megahertz)	1.1–1.65(volt)	300 μs
	AMD's Mobile K6 [3]	192–588(megahertz)	0.9–2.0(volt)	200 μs
	Intel PXA250 [4]	100–400(megahertz)	0.85–1.3(volt)	500 μs
	Compaq's Itsy [24]	59.0–206.4(megahertz)	1.0–1.55(volt)	189 μs
	TI's TMS320C55x [23]	6–200(megahertz)	1.1–1.6(volt)	3.3 ms($1.6 \rightarrow 1.1\ V$)
				300 μs($1.1 \rightarrow 1.6\ V$)
Academic	Burd *et al.* [1]	5–80(megahertz)	1.2–3.8(volt)	520 μs
	LART [25]	59–251(megahertz)	0.79–1.65(volt)	5.5 ms($1.65 \rightarrow 0.79\ V$)
				40 μs($0.79 \rightarrow 1.65\ V$)

DVS classification



Inter-task vs. Intra-task DVS (I)

- Classification is based on the scaling granularity
- Inter-task DVFS
  - Scaling occurs at the start of a task
    - It is unchanged until the task is completed
  - Use worst-case slack time (= $\text{Deadline}_{\text{task}} - \text{WCET}_{\text{task}}$)
  - Usually used in multi-task scheduling scenario at OS level
- Intra-task DVFS
  - Scaling occurs at the sub-task level
    - Different frequency is set for each sub-task
  - Use workload-variation slack time

Inter-task vs. Intra-task DVS (II)

- Average-Case Execution Time (ACET) rather than Worst-Case Execution Time (WCET)
  - Much finer granularity than inter-task
  - Fully exploits the slack time arising from task execution time variation
  - Requires off-line profiling and source code modification
  - Can achieve higher energy saving compared to inter-task
  - Energy and delay overheads of voltage switching must be carefully considered

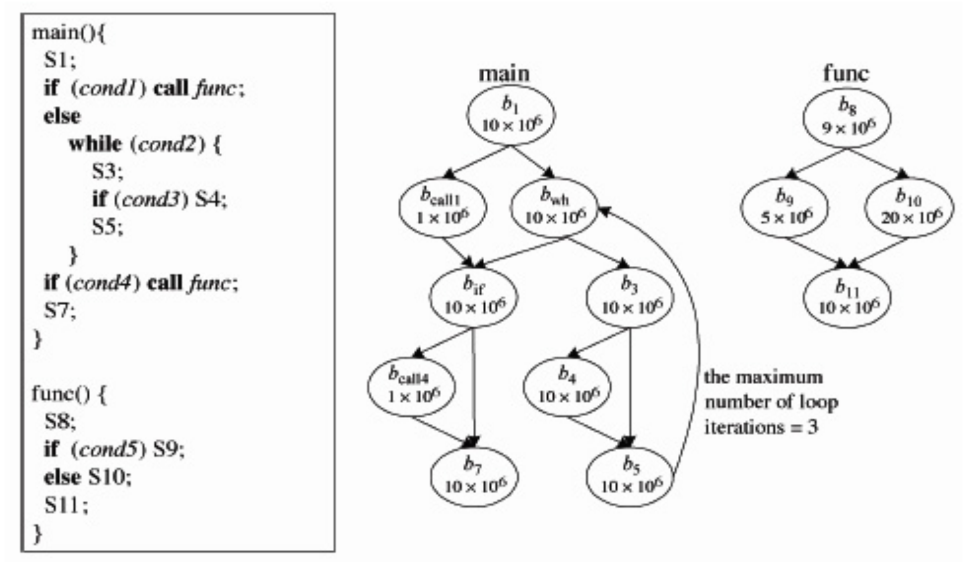
Contents

Intra-task DVS

- By Shin and Kim (SNU)
- For the given hard-real time constrained code
  - Extract CFG
- Each execution path has different execution time
- WCET method
  - Loss of too much slack
- ACET method
  - Hard to predict which path will be executed

Intra-task DVS

- Example



```
main(){
  S1;
  if (cond1) call func;
  else
     while (cond2) {
        S3;
        if (cond3) S4;
        S5;
     }
  if (cond4) call func;
  S7;
}

func() {
  S8;
  if (cond5) S9;
  else S10;
  S11;
}
```

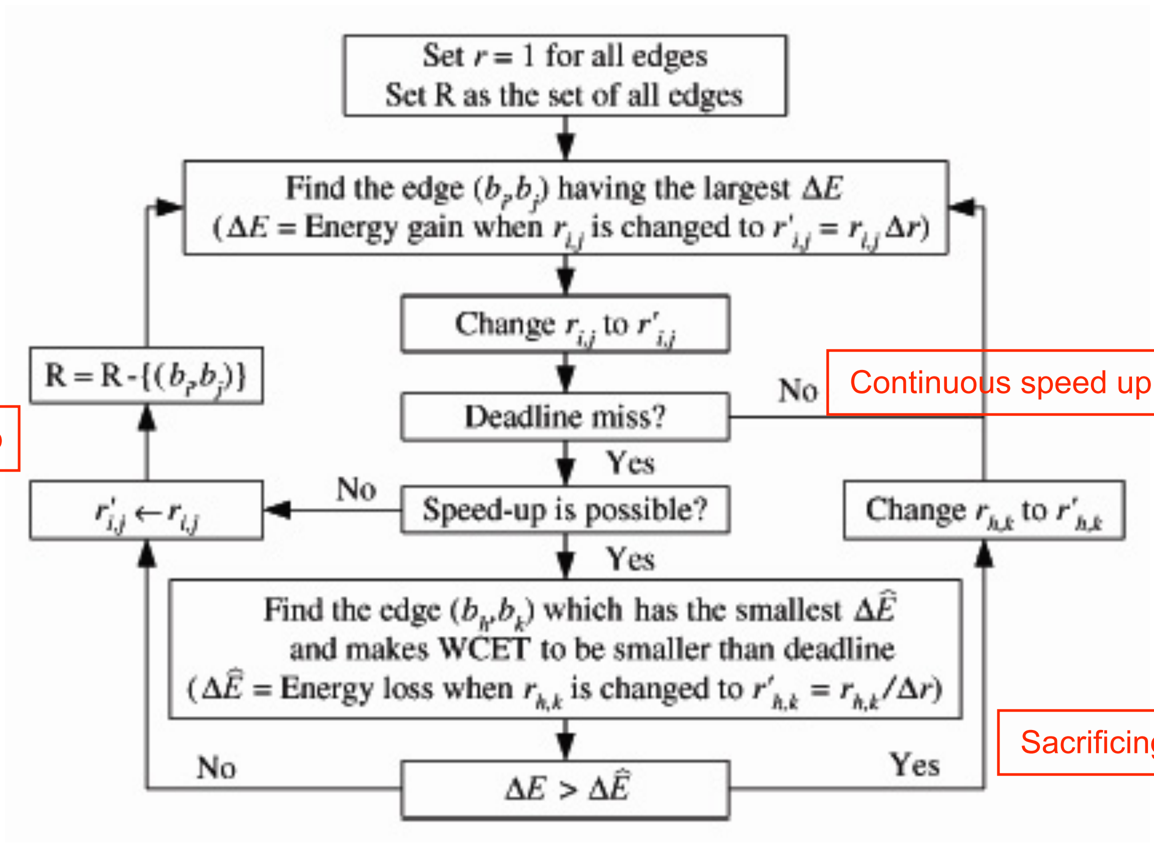
main
b_1 10×10^6
b_{call1} 1×10^6
b_{wh} 10×10^6
b_{if} 10×10^6
b_3 10×10^6
b_{call4} 1×10^6
b_4 10×10^6
b_7 10×10^6
b_5 10×10^6
the maximum number of loop iterations = 3
func
b_8 9×10^6
b_9 5×10^6
b_{10} 20×10^6
b_{11} 10×10^6

- 51 paths exist
  - Worst path : 200 * 10^6 cycles
  - 12 out of 51 paths: under 100 * 10^6 cycles

Intra-task DVS

- Problem definition
  - Find an optimal voltage / frequency pair at each edge (bi, bj) in a given CFG
- Pre-processing
  - Estimate the # of clock cycles required for each basic block
  - Estimate the visiting probability for each path
- Indirect energy consumption metrics
  - Clock speed representation
    - normalized to initial clock speed
  - Speed Update ratio
    - clock speed ratio between two edges
- Energy can be easily estimated from the information above

Intra-task DVS

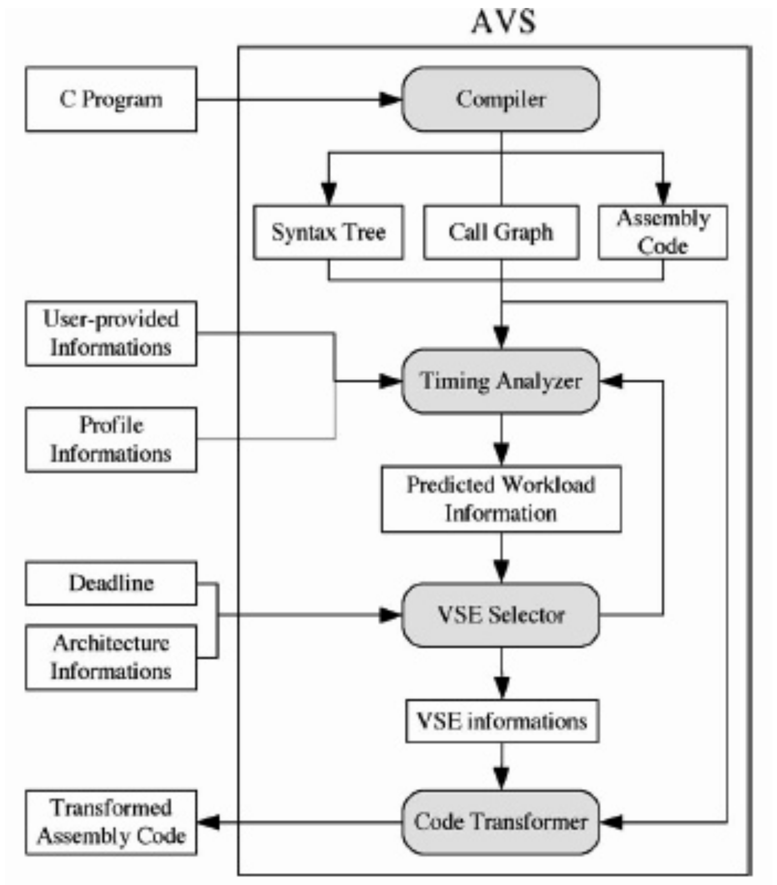

Set $r = 1$ for all edges
Set R as the set of all edges
Find the edge (b_i, b_j) having the largest ΔE
$(\Delta E$ = Energy gain when $r_{i,j}$ is changed to $r'_{i,j} = r_{i,j}\Delta r)$
Change $r_{i,j}$ to $r'_{i,j}$
Deadline miss?
No
Yes
Speed-up is possible?
No
Yes
$r'_{i,j} \leftarrow r_{i,j}$
$R = R - \{(b_i, b_j)\}$
Find the edge (b_h, b_k) which has the smallest $\Delta\hat{E}$
and makes WCET to be smaller than deadline
$(\Delta\hat{E}$ = Energy loss when $r_{h,k}$ is changed to $r'_{h,k} = r_{h,k}/\Delta r)$
$\Delta E > \Delta\hat{E}$
No
Yes
Change $r_{h,k}$ to $r'_{h,k}$
No more speed up
Continuous speed up
Sacrificing other edges

Intra-task DVS

- Too many decision points
  - Increase voltage / frequency changing overhead
- To deal with this issue, predict the path!
  - RWEP: Predict the worst case execution path
  - RAEP: Predict the average case execution path
- To cope with the mis-prediction
  - Voltage scaling edges (VSE) are selected
  - Based on static timing analysis for the given code
- VSE can change the speed
  - RWEP: monotonically decrease
  - RAEP: either decrease or increase

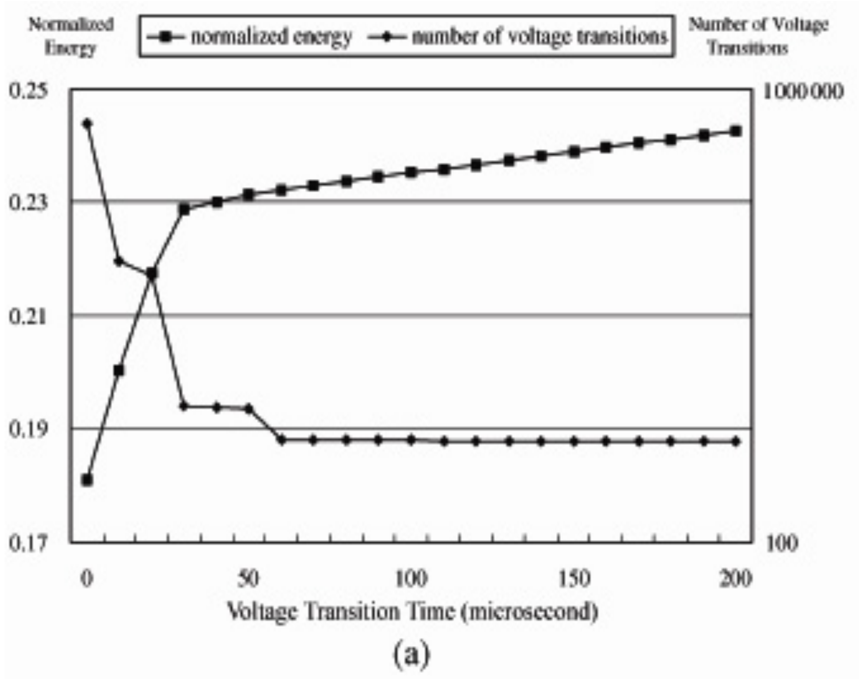
Intra-task DVS

- Automated tool flow for this method

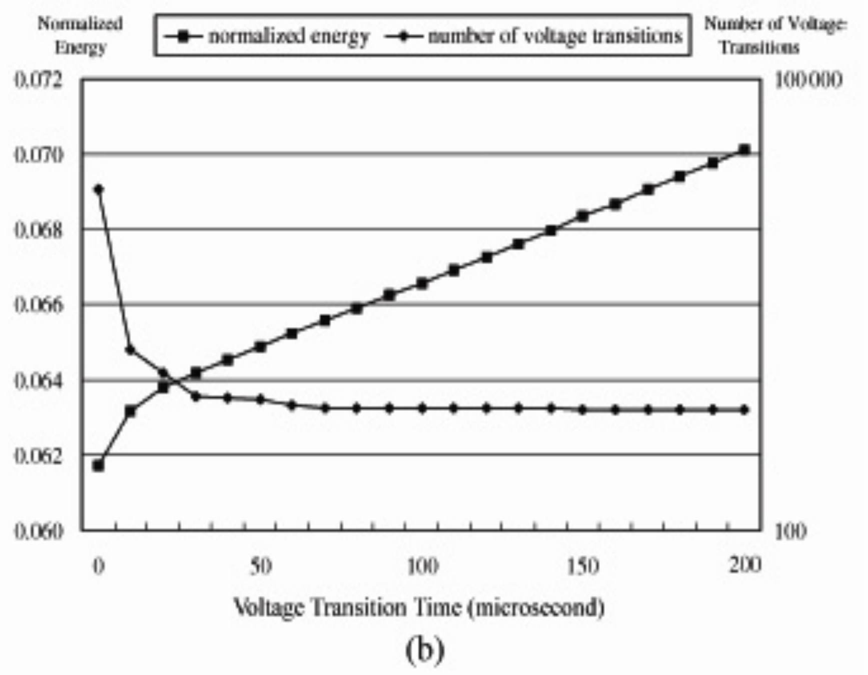


Intra-task DVS

- Result for RWEP method



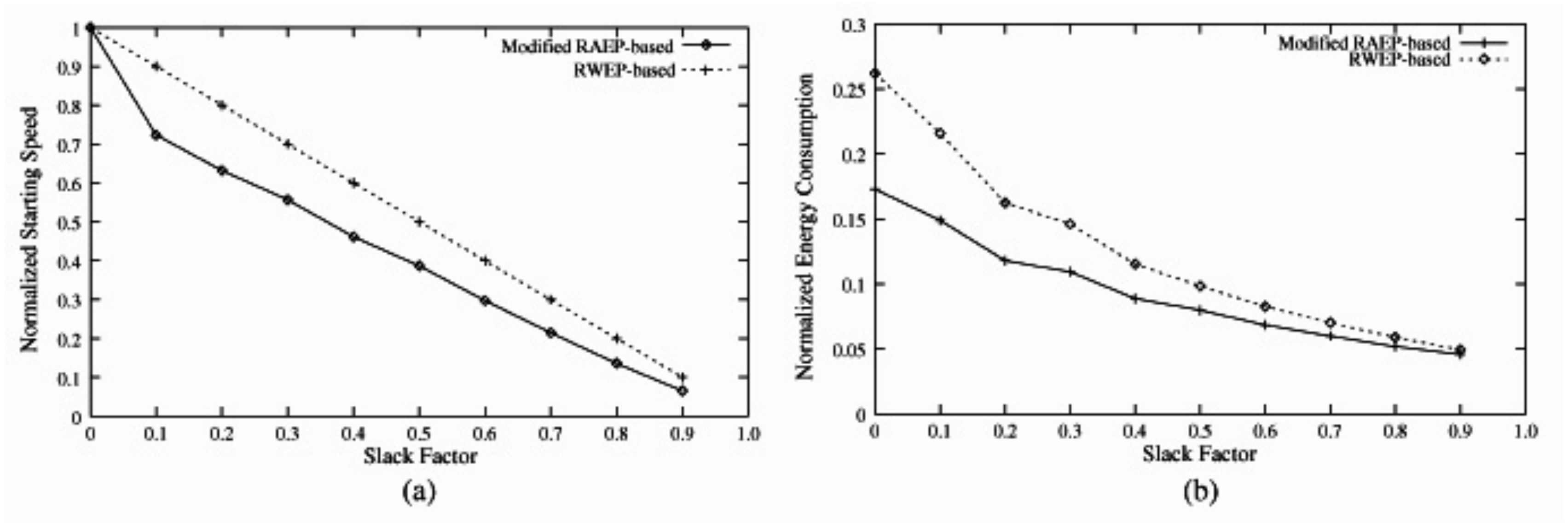
(a)



(b)

Intra-task DVS

- Comparison of RWEP and RAEP



-

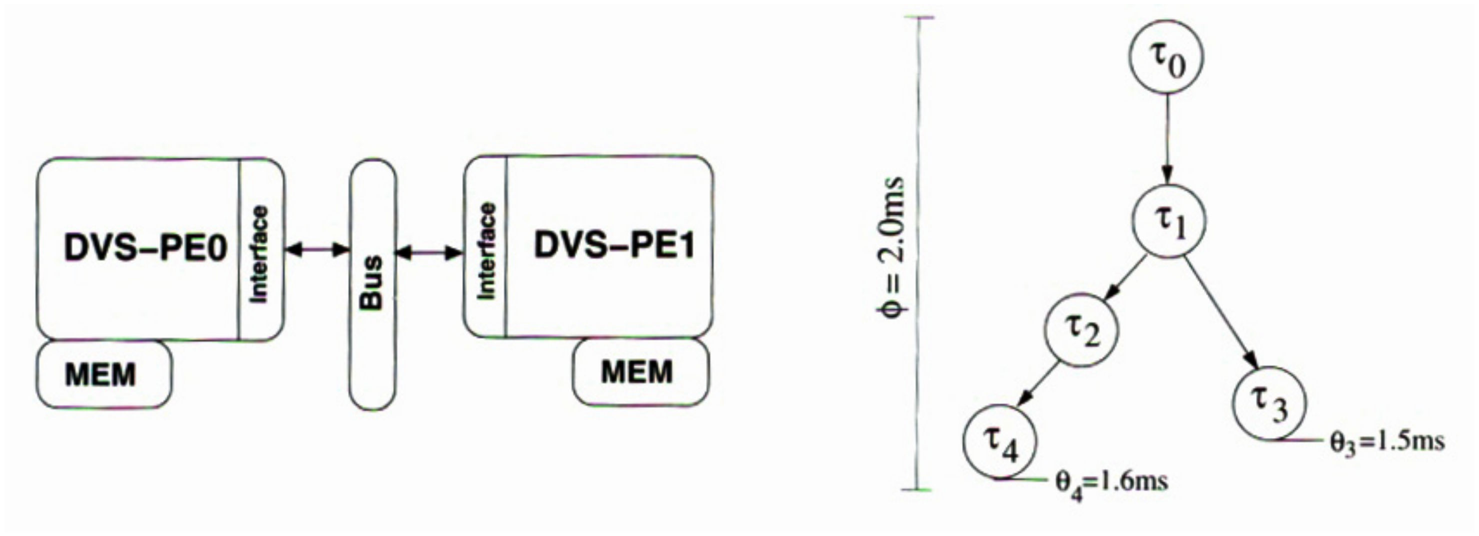
Contents

Inter-task voltage scaling technique

- Single processor environment
  - Similar to the conventional task scheduling method
  - Additional work is to exploits slacks maximally
- Multi processor environment
  - In conjunction with task assignment problem
  - Need to consider the communication overhead
- We will see the multi processor environment with the consideration of energy gradient
  - $\Delta E_{T} = E_{T}(t_{exe}) - E_{T}(t_{exe} + \Delta t)$

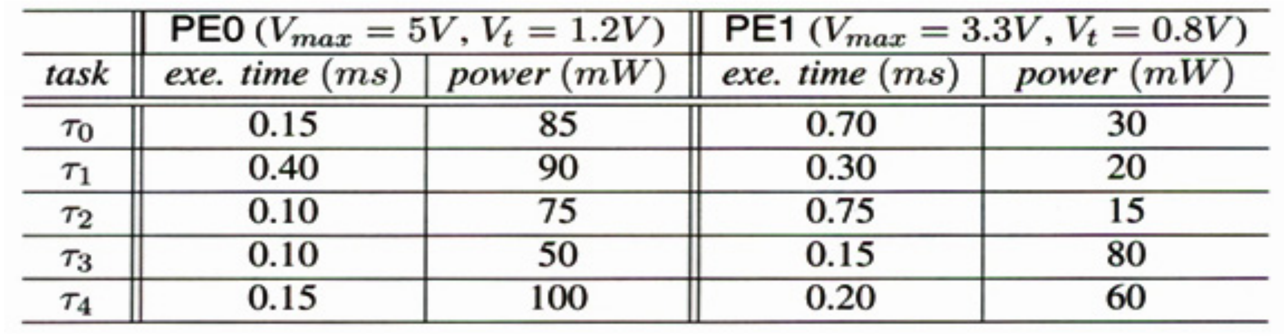
Target architecture and task graph

- Two heterogeneous processors
  - Transmeta Crusoe and StrongARM with Xscale technology
  - connected by a single bus
- Each processors has its dedicated memory
- Task graph (system specification)
  - Already scheduled (five tasks)



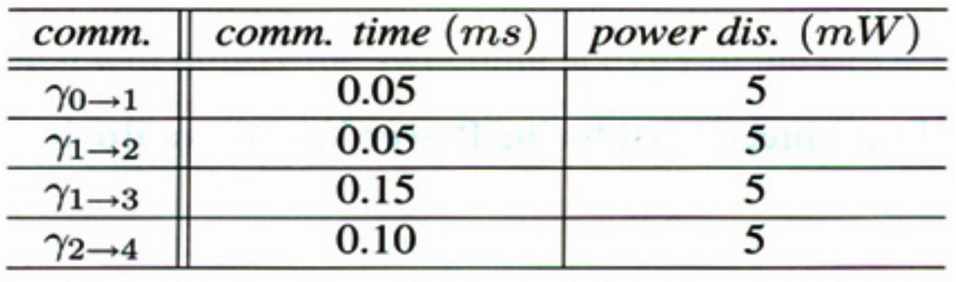
Task and inter-task information

- Nominal task execution time / power dissipation



	PE0 ($V_{max} = 5V, V_t = 1.2V$)		PE1 ($V_{max} = 3.3V, V_t = 0.8V$)	
task	*exe. time* (ms)	*power* (mW)	*exe. time* (ms)	*power* (mW)
τ_0	0.15	85	0.70	30
τ_1	0.40	90	0.30	20
τ_2	0.10	75	0.75	15
τ_3	0.10	50	0.15	80
τ_4	0.15	100	0.20	60

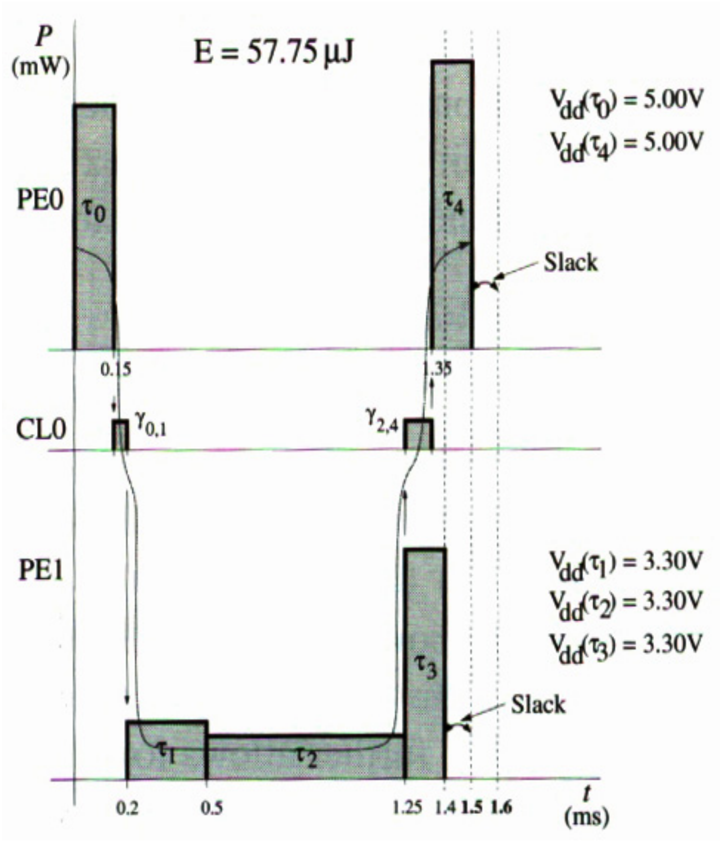
- Communication time / power dissipation



comm.	*comm. time* (ms)	*power dis.* (mW)
$\gamma_{0\to1}$	0.05	5
$\gamma_{1\to2}$	0.05	5
$\gamma_{1\to3}$	0.15	5
$\gamma_{2\to4}$	0.10	5

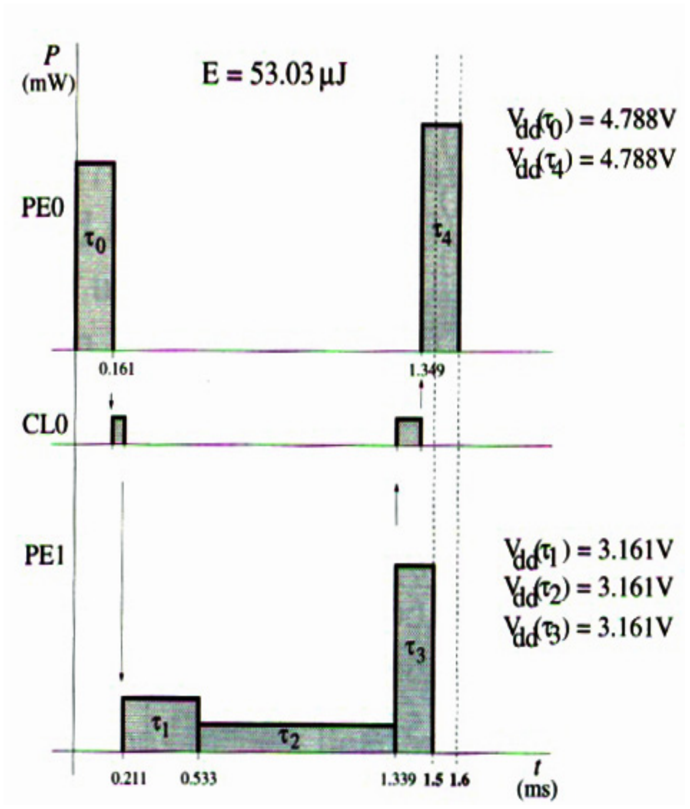
One possible mapping scenario

- Task mapping
  - P0: T0, T4
  - P1: T1, T2, T3
- Simply computes the power of each processor (at nominal)
- Slack exist for T3 and T4



With non energy-gradient model

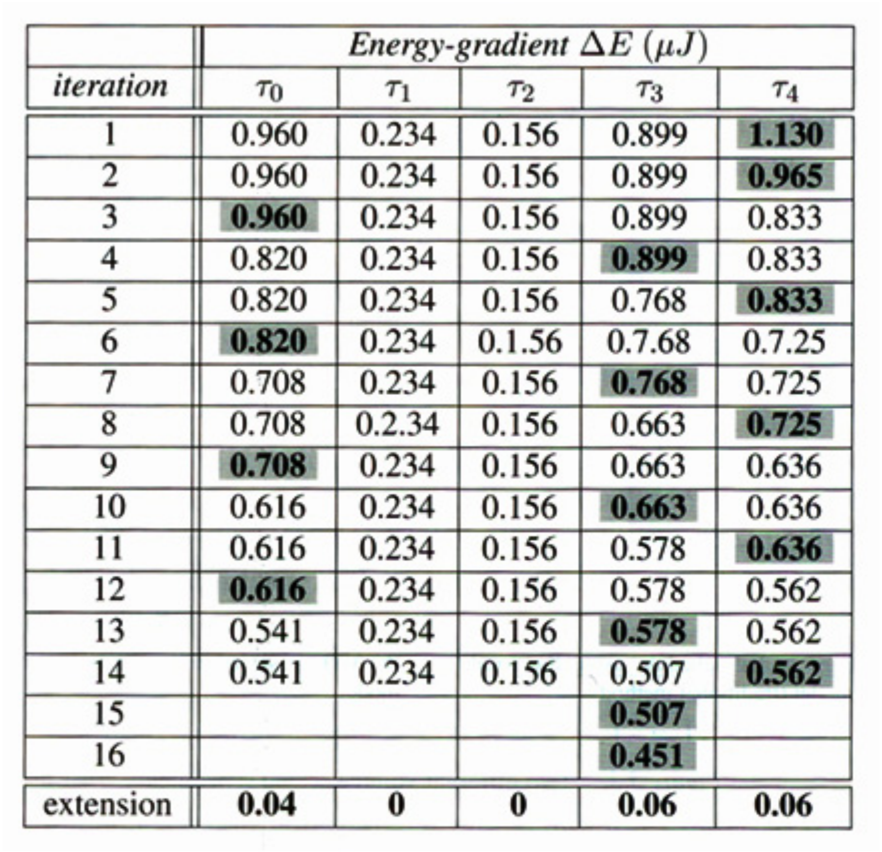
- Evenly distribute the slack to all the tasks
- Extension factor
  - $e = ((\Sigma t_{nom}(\top)) + ts) / \Sigma t_{nom}(\top)$ for all $\top$
- Delay
  - $\alpha \sim 1/f \sim k_d V_{dd} / (v_{dd} - V_t)^2$
- Voltage
  - $V_{dd} = V_t + V_0/2d^* + ((V_t + V_0/2d^*)^2 - V_t^2)^{1/2}$
- Energy reduction: 8.2%



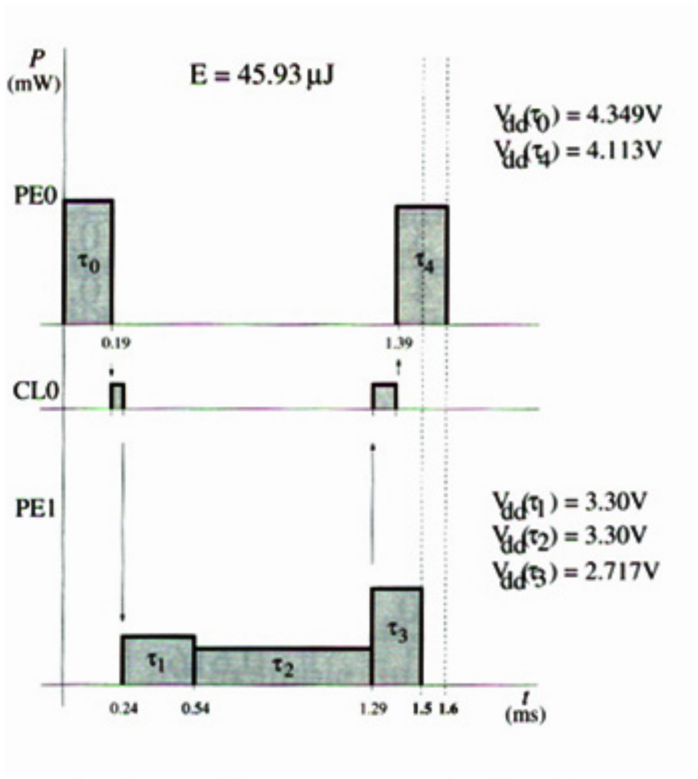
With energy-gradient model

- Suppose that $\Delta t = 0.01$ms
  - Ten times smaller than the slack
- Compute energy gradient for all tasks
  - Using $\Delta E_{\top} = E_{\top}(t_{exe}) - E_{\top}(t_{exe} + \Delta t)$
  - $E_{\top}(t_{exe})$ is given from the table
  - $E_{\top}(t_{exe} + \Delta t)$ is computed by using the previous method for entire slack range
  - The task which has the largest gradient is the winner
    - The largest energy saver
- Iteratively perform the energy gradient computation until slack is reached

Result of energy-gradient model



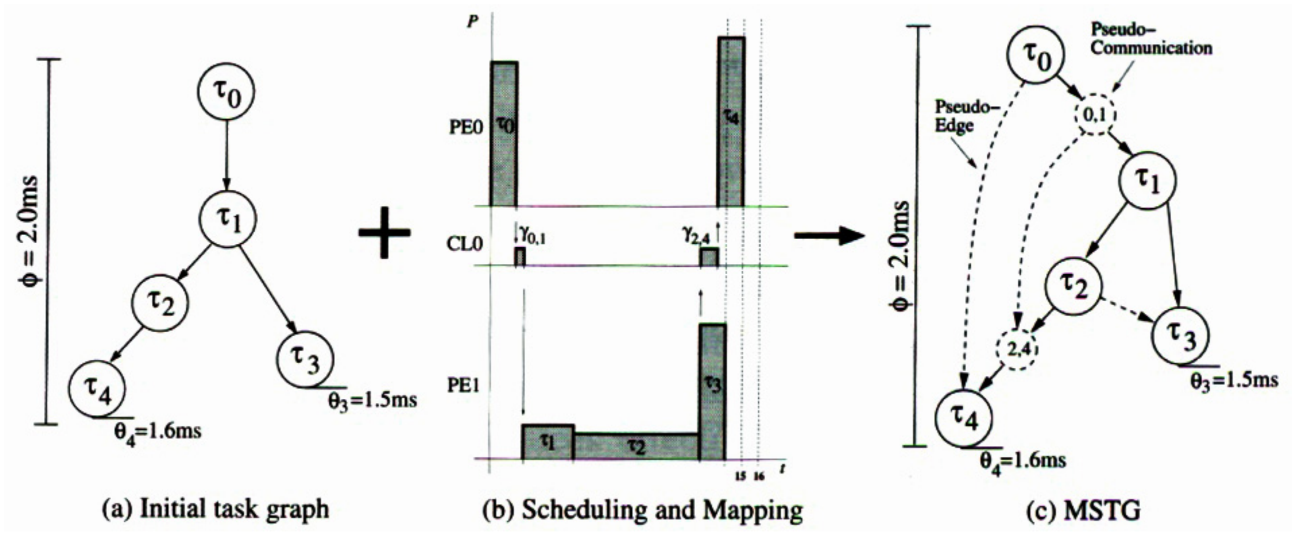
iteration	Energy-gradient ΔE (μJ) τ_0	τ_1	τ_2	τ_3	τ_4
1	0.960	0.234	0.156	0.899	**1.130**
2	0.960	0.234	0.156	0.899	**0.965**
3	**0.960**	0.234	0.156	0.899	0.833
4	0.820	0.234	0.156	**0.899**	0.833
5	0.820	0.234	0.156	0.768	**0.833**
6	**0.820**	0.234	0.1.56	0.7.68	0.7.25
7	0.708	0.234	0.156	**0.768**	0.725
8	0.708	0.2.34	0.156	0.663	**0.725**
9	**0.708**	0.234	0.156	0.663	0.636
10	0.616	0.234	0.156	**0.663**	0.636
11	0.616	0.234	0.156	0.578	**0.636**
12	**0.616**	0.234	0.156	0.578	0.562
13	0.541	0.234	0.156	**0.578**	0.562
14	0.541	0.234	0.156	0.507	**0.562**
15				**0.507**	
16				**0.451**	
extension	**0.04**	**0**	**0**	**0.06**	**0.06**



▇ winner at the corresponding iteration

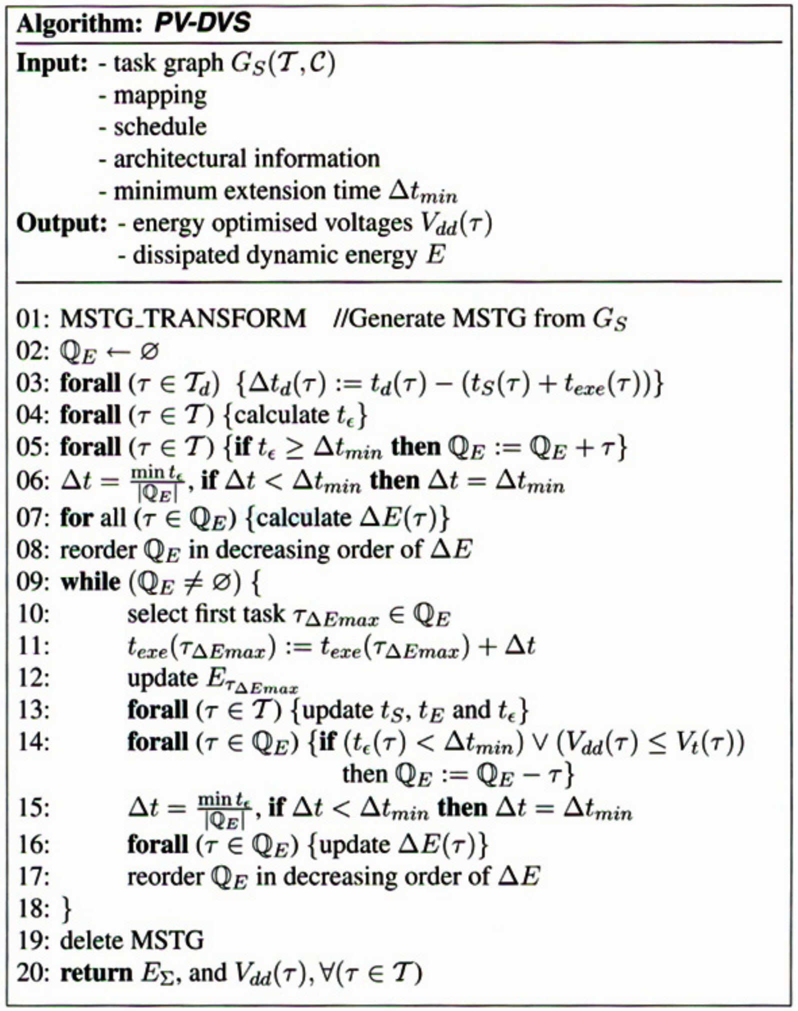
Formal approach (I)

- MSTG generation
  - Mapped-Scheduled Task Graph
  - Insert communication edges
  - Each comm. edge is represented as a pseudo node
  - Insert pseudo edge
    - Dependency of tasks mapped to the same resource



Formal approach (II)

- Perform the schedule as qualitatively mentioned earlier
  - Q_E: priority queue
  - T_d: a set of tasks who have deadlines
  - $T_s(\top)$: task start time



Algorithm: ***PV-DVS***

```
Input:  - task graph $G_S(\mathcal{T},\mathcal{C})$
        - mapping
        - schedule
        - architectural information
        - minimum extension time $\Delta t_{min}$
Output: - energy optimised voltages $V_{dd}(\tau)$
        - dissipated dynamic energy $E$

01: MSTG_TRANSFORM   //Generate MSTG from $G_S$
02: $\mathbb{Q}_E \leftarrow \varnothing$
03: forall $(\tau \in \mathcal{T}_d)$ $\{\Delta t_d(\tau) := t_d(\tau) - (t_S(\tau) + t_{exe}(\tau))\}$
04: forall $(\tau \in \mathcal{T})$ {calculate $t_\epsilon$}
05: forall $(\tau \in \mathcal{T})$ {if $t_\epsilon \geq \Delta t_{min}$ then $\mathbb{Q}_E := \mathbb{Q}_E + \tau$}
06: $\Delta t = \frac{\min t_\epsilon}{|\mathbb{Q}_E|}$, if $\Delta t < \Delta t_{min}$ then $\Delta t = \Delta t_{min}$
07: for all $(\tau \in \mathbb{Q}_E)$ {calculate $\Delta E(\tau)$}
08: reorder $\mathbb{Q}_E$ in decreasing order of $\Delta E$
09: while $(\mathbb{Q}_E \neq \varnothing)$ {
10:         select first task $\tau_{\Delta Emax} \in \mathbb{Q}_E$
11:         $t_{exe}(\tau_{\Delta Emax}) := t_{exe}(\tau_{\Delta Emax}) + \Delta t$
12:         update $E_{\tau_{\Delta Emax}}$
13:         forall $(\tau \in \mathcal{T})$ {update $t_S$, $t_E$ and $t_\epsilon$}
14:         forall $(\tau \in \mathbb{Q}_E)$ {if $(t_\epsilon(\tau) < \Delta t_{min}) \vee (V_{dd}(\tau) \leq V_t(\tau))$
                    then $\mathbb{Q}_E := \mathbb{Q}_E - \tau$}
15:         $\Delta t = \frac{\min t_\epsilon}{|\mathbb{Q}_E|}$, if $\Delta t < \Delta t_{min}$ then $\Delta t = \Delta t_{min}$
16:         forall $(\tau \in \mathbb{Q}_E)$ {update $\Delta E(\tau)$}
17:         reorder $\mathbb{Q}_E$ in decreasing order of $\Delta E$
18: }
19: delete MSTG
20: return $E_\Sigma$, and $V_{dd}(\tau), \forall(\tau \in \mathcal{T})$
```

ELPL Embedded Low-Power Laboratory

Voltage change is practically discrete (I)

- For a task T
  - We have t_{exe} with V_{dd} in continuous domain
- In discrete domain
  - two nearest voltage (one is lower, the other is higher) can be utilized
  - e.g. $V_{d1} < V_{dd} < V_{d2}$
- How?
  - $t_{exe} = t_{d1} + t_{d2}$
  - $t_{d1} = t_{exe} (V_{d1}(v_{dd}-V_t)^2 / (v_{d1}-V_t)^2 V_{d1}) \times \{(V_{dd}/(V_{dd}-V_t)^2 - V_{d2}/(V_{d2}-V_t)^2) / (V_{d1}/(V_{d1}-V_t)^2 - V_{d2}/(V_{d2}-V_t)^2)\}$

ELPL Embedded Low-Power Laboratory

Voltage change is practically discrete (I)

- Make t_{d1} and t_{d2} as integers
  - Since they should be represented as # of clock cycles
- t_i = NCi * f_i
  - NC_i: # of clock cycles executed at frequency f_i
- NC_{d1} = base(t_{d1} * f_{d1})
- NC_{d2} = NC_{tot} – NC_{d1}
- td_1 = NC_{d1} / f_{d1}
- t_{d1} = NC_{d1} / f_{d1}

Seoul National University

Summary

- Two system-level power management techniques
  - DPM by shutdown
  - DVS by extending the execution time
- DPM
  - Time-out / Predictive / Stochastic
  - Prediction accuracy is critical
- DVS
  - Intra-task / Inter-task
- Common feature of DVS and DPM
  - Exploiting idleness

References

- Luca Benini, Giovanni De Micheli, “Dynamic Power Management: Design Techniques and CAD Tools”, Springer, 2005
- Marcus T. Schmitz, Bashir M. Al-Hashimi, and Petru Eles, “System-Level Design Techniques for Energy-Efficient Embedded Systems”, Kluwer Academic Publishers, 2003
- D. Shin, J. Kim and S. Lee, "Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications," IEEE Design and Test of Computers, Vol.18 No.2, pp. 20-30, March 2001
- E. Macci, M. Pedram, “Leakage Power Modelling and Minimization”, Tutorial at ICCAD, 2004