

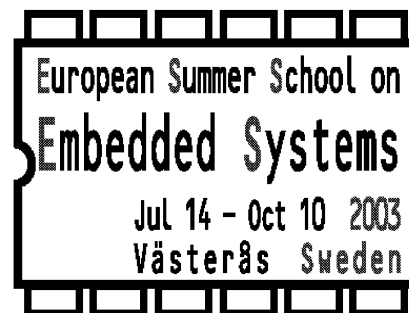
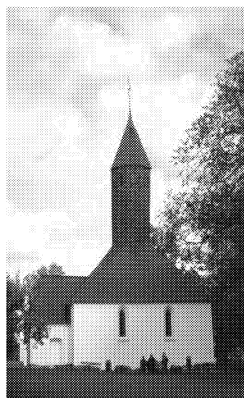
MÄLARDALENS HÖGSKOLA

# ESSES 2003

## European Summer School on Embedded Systems

### Lecture Notes Part VI

### Low Power Systems: Low-power System Modeling and I/O Systems



Editors: Ylva Boivie, Hans Hansson, Jane Kim, Sang Lyul Min

Västerås, July 28-August 1, 2003

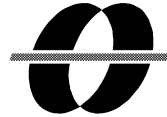
ISSN 1404-3041

ISRN MDH-MRTC-105/2003-1-SE

# MRTC

MÄLARDALEN REAL-TIME  
RESEARCH CENTRE

[www.mrtc.mdh.se](http://www.mrtc.mdh.se)



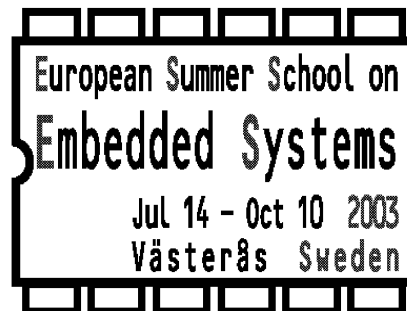
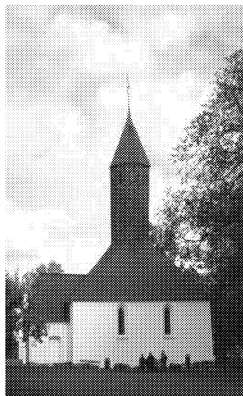
MÄLARDALENS HÖGSKOLA

# ESSES 2003

## European Summer School on Embedded Systems

### Lecture Notes Part VIII

### Low Power Systems: Low-power System Modeling and I/O Systems



Editors: Ylva Boivie, Hans Hansson, Jane Kim, Sang Lyul Min

Västerås, July 28-August 1, 2003

ISSN 1404-3041

ISRN MDH-MRTC-105/2003-1-SE

# MRTC

MÄLARDALEN REAL-TIME  
RESEARCH CENTRE

[www.mrtc.mdh.se](http://www.mrtc.mdh.se)



# **Floating-point to Integer Conversion for Low-power Implementation of Digital Signal Processing Programs**

Wonyong Sung

School of Electrical Engineering  
Seoul National University, Korea

# Floating-point to Integer Conversion for Low-power Implementation of Digital Signal Processing Programs

Wonyong Sung  
School of Electrical Engineering  
Seoul National University

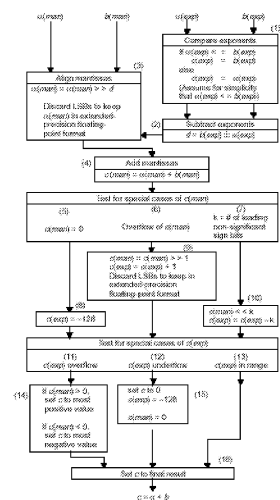
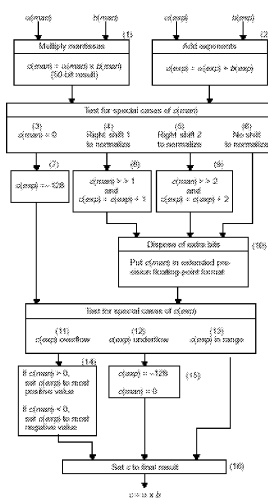
Version 2003. 7. 18

## What is fixed-point arithmetic?

- Floating-point arithmetic
  - For ex.)  $0.9 * 0.55 = 0.495$ ,  $0.9 * 5500 = 4950$   
 $0.9 * 0.5555555 = 0.49999995$
- Fixed-point arithmetic
  - For ex.)  $0.9 * 0.55 = 0.495$  (seems OK)  
 $0.9 * 5500 = ?$  (overflow)  
 $0.9 * 0.5555555 = 0.499$  (quantization)
  - Fixed-point arithmetic
    - Range is limited (scaling needed)
    - Precision limited

# Why fixed-point implementation?

- Fixed-point arithmetic (or integer arithmetic) requires less chip area, less delay, and less bus width (for memory).
  - Leads to 1/2 to 1/10 chip area reduction
  - Leads to x2 speed increase.
- Some embedded processors do not equip floating-point arithmetic unit
  - A floating-point arithmetic using library requires many (at least a few 10's) cycles.
  - Leads to x10 speed increase or energy reduction.



# Why is fixed-point implementation important in DSP

- Digital signal processing
  - Requires a large number of arithmetic operations (multiply, add, memory access)
    - 10M ~ 100M operations / sec
  - Do not require exact results, in terms of SQNR 40dB~100dB
- Embedded systems
  - Many do not equip floating-point arithmetic units
  - Require different word-length according to the applications
    - Audio: around 20 ~ 24 bits
    - Speech: 12~20 bits
    - Video: 8~16 bits
    - Graphics: high precision

# Issues in fixed-point optimization

- Performance estimation
  - By analytical methods (theory)
    - Usually limited to linear systems
  - By simulation
    - Requires simulation using a large number of input data (fast simulation required)
    - Easy simulation program generation required
- Automatic scaling
  - Scales the input and output data ( $\times 2^{-n}$ )
  - $0.9 \times 5500 = ? \rightarrow 0.9 \times 0.55 = 0.495 (*10^4)$
- Word-length optimization
  - Smaller word-length degrades the system performance
  - Larger word-length requires more chip area and power consumption.

## Overview of this talk

1. Fixed-point arithmetic and system design
2. Fixed-point simulation method
3. Autoscaler (floating-point to integer)
4. Fixed-point optimizer (wordlength opt.)

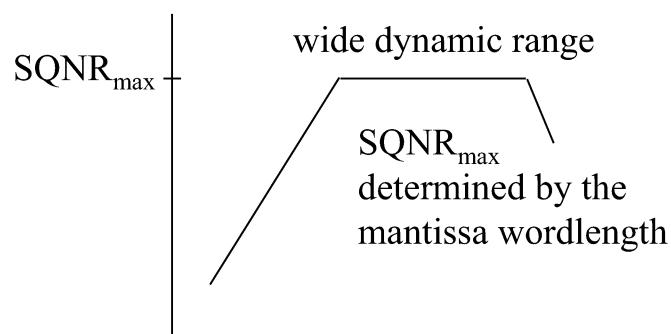
## Fixed-point Arithmetic and System Design

# 1. Number Representation

- Floating-point format
  - $x = M(x) 2^{E(x)}$
  - wide dynamic range <- no explicit scaling needed
  - good for algorithm develop, higher hardware cost
- Fixed-point format
  - only  $M(x)$  is used with constant  $E(x)$
  - minimizing the wordlength is important for economic hardware implementation

## Floating-point format

- SQNR



- 32bit IEEE standard format is most widely used (24bit mantissa, 8bit exponent)

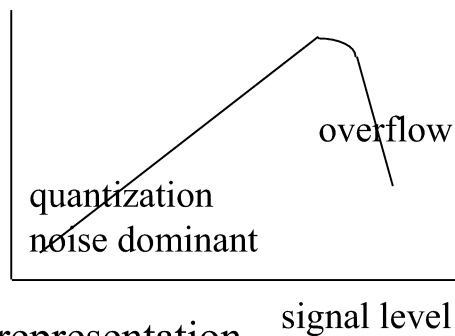
## Fixed-point format

- SQNR

- Scaling is needed

- Specification

- Negative number representation
- Overflow handling
- Word-length reduction
- Conversion to or from real value



## Negative number representation

- Unsigned

0000: 0, 0001:1, 0111:7, 1111:15

- Signed

- Two's complement ( $-x = \bar{x} + 1$ )

0000:0, 0001:1, 0111:7

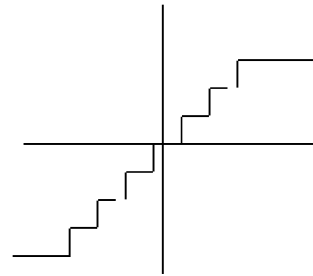
1111: -1, 1000: -8

- One's complement ( $-x = \bar{x}$ )

## Overflow handling

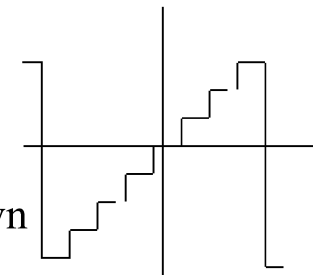
- Saturation

- no sign loss for overflow
- magnitude becomes maximum
- preferred in signal processing



- Overflow

- simple implementation
- good when signal range is known



## Quantization (Wordlength reduction)

- Methods

- Rounding
- Truncation

- Source of Quantization

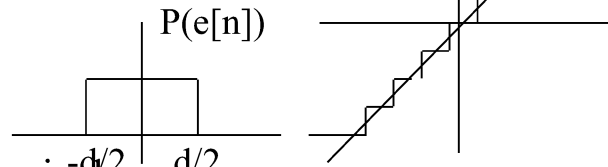
- Signal quantization
- Coefficient quantization



## Quantization methods

- Rounding

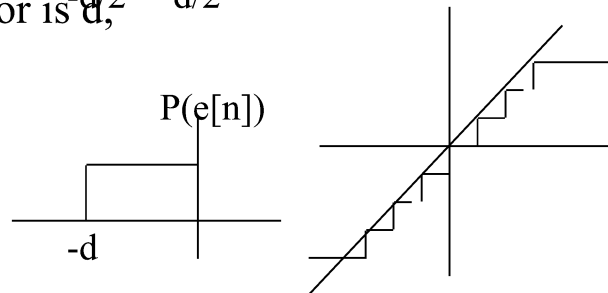
- max. quant. error is  $d/2$



- Truncation

- max. quant. error is  $d$ ,

- D.C. bias  
(fatal when accumulated)



## 2. Fixed-point Data Format

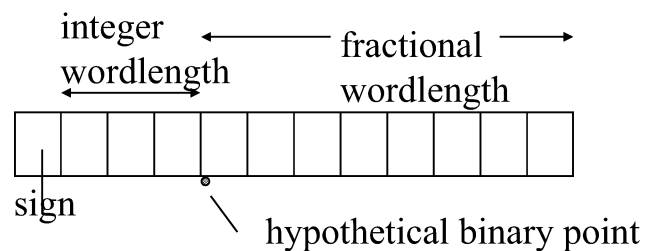
- Integer format

- all data is interpreted as an integer
- the quantization level is large (1)

- Fractional format

- all data is between -1 to 1
- the quantization level is  $2^{-(B-1)}$

- Generalized fixed-point format
  - allow both integer and fractional wordlengths
  - integer wordlength determines the maximum signal range, and the fractional wordlength determines the quantization level.



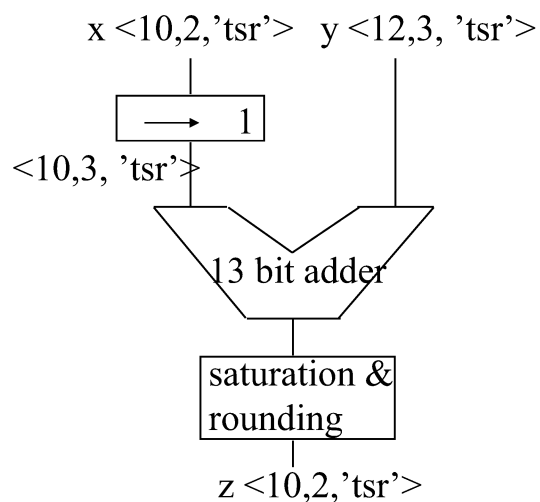
## Generalized fixed-point format

- SPW format
  - $\langle \text{wordlength, integer wordlength, sign} \rangle \langle 12, 3, \text{t} \rangle$
  - signal range:  $-2^3 \sim +2^3$ , quantization level:  $2^{-8}$
- Silage (DFL) format
  - fix  $\langle \text{wordlength, fractional wld} \rangle$ ; always two's compl.
  - fix  $\langle 12, 8 \rangle$
- SNU Fixed-point Simulator format
  - $\langle \text{wordlength, integer wordlength, sign\_overflow\_quantization mode} \rangle$
  - $\langle 12, 3, \text{'tsr'} \rangle$

## Generalized fixed-point format

- Arithmetic shift left by n bit
  - decreases the IWL by n
- Arithmetic shift right by n bit
  - increases the IWL by n
- Addition
  - the IWL of both operands should be the same
- Multiplication ( $z = x*y$ )
  - $IWL(z) = IWL(x) + IWL(y) - 1$ ; in 2's compl.

## Generalized fixed-point format



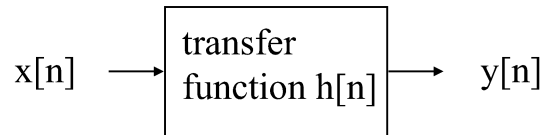
## 3. Scaling Method

- Purpose of Scaling
  - prevent overflows or waste of bits by proper shifting (arithmetic shift)
- Scaling implementation
  - overflow prevention: shift right (IWL increase)
  - save extra-bits: shift left (IWL decrease)
- Minimum integer wordlength
  - $IWL_{\min}(x) = \lceil \log_2 |R(x)| \rceil$
  - $R(x)$  is the range of a signal

### Range estimation method

- Analytical method
  - L1 norm based is most widely used
  - L1 norm based: very conservative estimate
  - applicable to linear or simple systems
- Simulation based
  - requires computing power
  - optimum estimate, but input signal dependent
  - applicable to non-linear and time-varying systems

## L1 norm based scaling



$$|y[n]| \leq x_{\max} \sum |h[n]|$$

This means that the output can be higher than the input level by L1 norm times

L1 norm computation

- by using analytical method
- by computing the unit pulse response, and then summing-up the absolute value of the response

## L2 norm based scaling

$$\text{L2 norm} = \sum |h[n]|^2$$

- L2 norm is an upperbound for the signal energy
- it is less pessimistic than the L1 norm
- it is optimum estimation when the input is a white random input

## Linf norm based scaling

- $L_{\infty} = \text{Max } |H(j\omega)|$ 
  - is an upperbound when the input is sinusoid
  - good when the input signal is very highly correlated
  - least pessimistic norm

## Simulation based range estimation

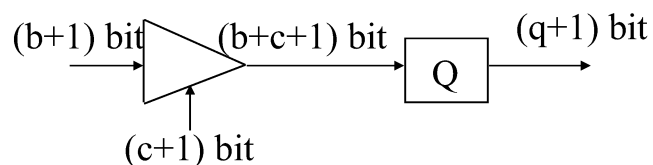
- collect information of sum, squared\_sum, absolute\_max, and the number of update during the floating-point simulation
- derive mean and standard deviation for a signal after the simulation
- $R(x) = \max \{ |m(x)| + n \sigma(x), A_{\text{Max}} \}$ ,  
where  $n$  is between 4 to 16

## Simulation based range estimation

- Comparison for a 2nd order IIR filter for speech application (WL = 16 bit)
  - IWL determined by the simulation based method is 4 bit smaller than that of the L1 norm based method
  - SQNR difference of 24 dB
- ADPCM implementation
  - needs 4 different speech files for obtaining a reliable range data

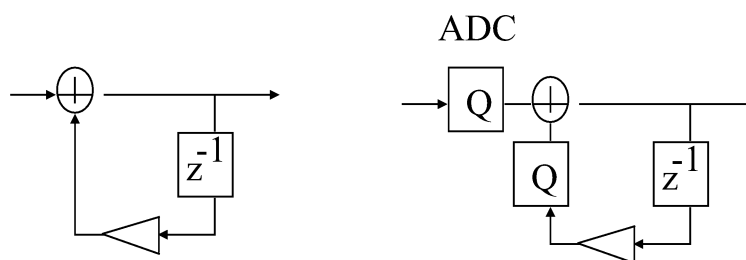
## 4. Wordlength Determination

- Wordlength reduction
  - reduce the wordlength, at an ADC or multiplier output, to minimize the hardware cost while keeping the signal accuracy acceptable

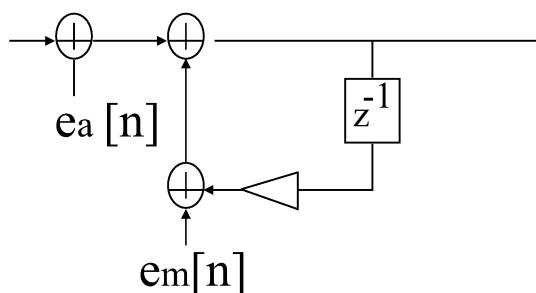


## Modeling of quantization

- Ideal system - for floating-point simulation
- Non-linear model  
- for fixed-point simulation
- Linearized model - for analytical method



## Modeling of quantization - continue

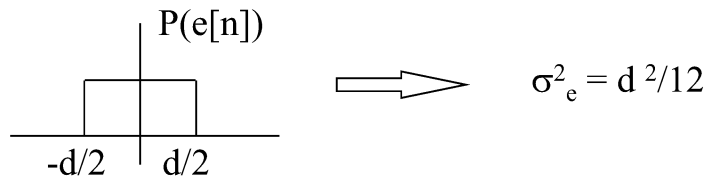


- the magnitude of  $e_a[n]$  and  $e_m[n]$  is dependent on the fractional word-length



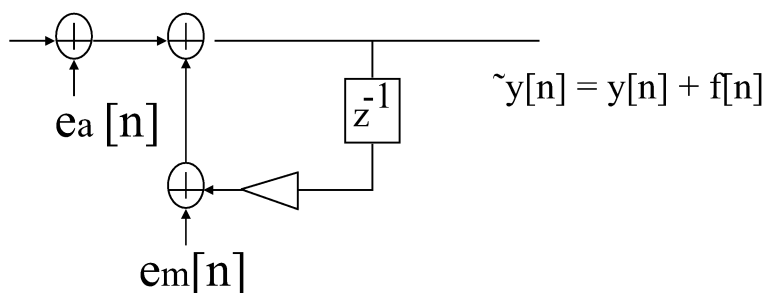
## Linearized modeling of signal quantization

- Add statistically equivalent quantization error signal instead of the quantizer
- Quantization noise
  - wide-sense stationary white noise
  - uniform distribution between  $-d/2$  to  $d/2$
  - different noise sources are uncorrelated



## Modeling of signal quantization

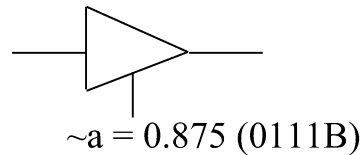
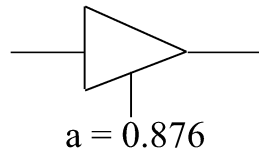
- Computation of output noise power



$$\begin{aligned} \sigma_f^2 &= (\sigma_a^2 + \sigma_m^2) (1/2\pi) \int |H(e^{j\omega})|^2 d\omega \\ &= (\sigma_a^2 + \sigma_m^2) \sum |h[n]|^2 \end{aligned}$$

## Coefficient quantization

- Word-length reduction in filter coefficients or (usually) constant system parameters



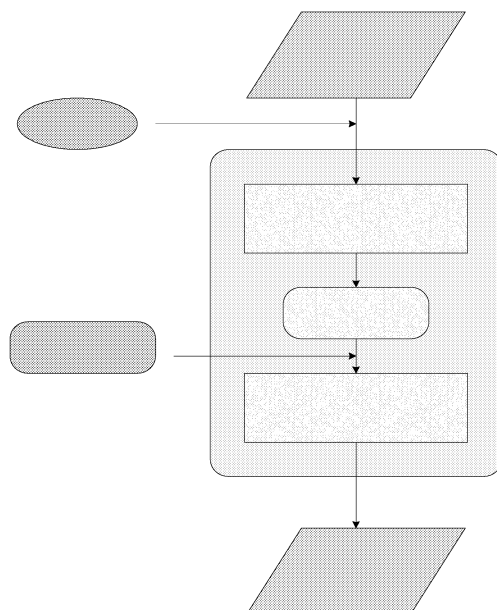
- Effects of coefficients for FIR and IIR digital filters
  - deterministic
  - easily known by obtaining the frequency response from the quantized coefficients
  - optimization program for minimum hardware cost

## 5. Fixed-point Performance

- Effects of quantization noise in digital signal processing systems
  - Linear digital filters:
    - additive quantization noise at the output
  - Constant filter coefficients:
    - transfer function is modified
  - Adaptive digital filter:
    - quantization noise changes the adaptation performance or the mean squared error (mse) after the convergence <- distortion

- Performance measure
  - Linear digital filters
    - SQNR
  - Adaptive digital filters
    - mean squared error after some time-off period
  - Speech coder
    - the SQNR between the original speech and the reconstructed speech after compensating the filtering or time-delay
  - Communication system(Tx-Rx system)
    - the bit-error rate is the best measure, but requires much simulation time
    - the peak error at the eye diagram can be a measure

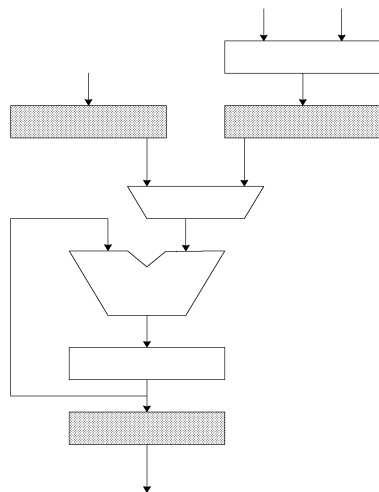
## Autoscaler for 'C2x/5x



## Hypothetical Floating-Point Instructions

Floating-point instructions		Corresponding fixed-point instructions
Instruction	Operation	
ADDF	Add to accumulator	ADD
APACF	Add P register to accumulator	APAC
INF	Input a floating-point data	IN
LACF	Load Accumulator	LAC
LTF	Load T register	LT
LTAF	Load T register, add P reg. to accumulator	LTA
LTDF	Load T register, add P reg. ..., move data	LTD
MACF	Multiply and accumulate	MAC
MACDF	Multiply, accumulate, and data move	MACD
MPYF	Multiply	MPY
SACF	Store accumulator	SACH
SACFV	Store with overflow check	macro
SQRTF	Square root	macro
DIVF	Divide	macro

## Simplified Data Path of TMS320C2x/5x



- Note three barrel shifters

- Shifter a
- Shifter o
- Shifter m

$$W_1(t) = \max W_1(x, P, ACC, z)$$

$$sm = 4, 1, 0, \text{ or } -6$$

$$W_1(ACC) = W_1(w) + W_1(y) + 1 - sm \geq W_1(t)$$

## Overview of this talk

1. Fixed-point arithmetic and system design
2. Fixed-point simulation method
3. Autoscaler (floating-point to integer)
4. Fixed-point optimizer (wordlength opt.)

## Fixed-point Simulation

- Introduction
  - Fixed-point format
  - C++ fixed-point class based method
  - Code conversion method
  - Assembly code simulator for C25 DSP
- 
- Reference: Seehyun Kim, Ki-II Kum, and Wonyong Sung, "Fixed-Point Optimization Utility for C and C++ Based Digital Signal Processing Programs," IEEE Tr. Circuits and Systems II, Vol. 45, No. 11, November, 1998.

# Introduction

- Fixed-point simulation is very needed for
  - Accurate performance estimation
  - But takes much time especially when the wordlength or quantization format is different
- Seamless conversion from floating-point to fixed-point code is needed
  - Do not want take much time. Just easy conversion
- Range estimation code is needed, usually range is obtained from the floating-point simulation

- Issues
  - Word-length determination: high-level design or low-level design?
- High-level (system or algorithm level) design
  - Fast simulation and wordlength determination
  - May not be bit-accurate
- Low-level (assembly, RTL level) design
  - Slow simulation and word-length determination

## 2. Fixed-point format

- Hardware oriented language supports good fixed-point format
  - VHDL
  - Silage
  - SPW
- C language does not. Only integer (short, long) and float.

## C++ fixed-point class

- C++ class – overloading
- Each arithmetic operation in C code is translated to the overloaded operations according to the type declaration of the variables or constants
- fSig: class for range estimation
  - Conduct floating-point arithmetic and update the mean, variation, and max of the variable
- gFix: class for fixed-point arithmetic according to the SNU fixed-point format
  - gFix a(12, 0, “tsr”)
- pFix: a pseudo fixed-point library using floating-point ALU: 53bit for mantissa): may not be bit-accurate for high precision data.

## gFix class

- gFix operator \* (const gFix& x, const gFix& y)
- // assume that
- // result.wl = x.wl + y.wl - 1
- // result.iwl = x.iwl + y.iwl
- {
- short     iwlen, wlen;
- Integer   I;
- 
- wlen = x.wl + y.wl - 1;
- if (wlen > MAXWL) wlen = MAXWL;
- iwlen = x.iwl + y.iwl;
- I = x.M \* y.M;
- 
- return gFix(I,wlen,iwlen);
- }
- 

### 3. C++ class based development procedure

- C or C++ programming using floating-point arithmetic -> algorithm verification
- Insert range estimation directives (fSig) to variable declaration statements
- Range estimator -> scaling
- Insert fixed-point simulation directive (gFix or pFix) s to variable declaration statements
- Fixed-point simulator -> wordlength determination



## Code example (1<sup>st</sup> order IIR filter)

```
void
iir1 (short argc, char *argv[])
{
    float Xin;
    float Yout; // fSig()
    float Ydly; // fSig()
    float Coeff;

    Coeff = 0.9;
    Ydly = 0.;
    for (i=0; i< 1000; i++) {
        infile >> Xin;
        Yout = Coeff * Ydly + Xin;
        Ydly = Yout;
        outfile << Yout << '\n';
    }
}
```

(a) Original C++ program

## Range estimation & fixed-point simulation code

```
void
iir1 (short argc, char *argv[])
{
    float Xin;
    static fSig Yout("iir1/Yout");
    static fSig Ydly("iir1/Ydly");
    float Coeff;

    Coeff = 0.9;
    Ydly = 0.;
    for (i=0; i< 1000; i++) {
        infile >> Xin;
        Yout = Coeff * Ydly + Xin;
        Ydly = Yout;
        outfile << Yout << '\n';
    }
}
```

(b) Range estimation code

```
void
iir1 (short argc, char *argv[])
{
    gFix Xin(12,0);
    gFix Yout(16,3);
    gFix Ydly(16,3);
    gFix Coeff(10,0);

    Coeff = 0.9;
    Ydly = 0.;
    for (i=0; i< 1000; i++) {
        infile >> Xin;
        Yout = Coeff * Ydly + Xin;
        Ydly = Yout;
        outfile << Yout << '\n';
    }
}
```

(c) Fixed-point simulation code

## Comparison of simulation time

- A Fourth order IIR filter
  - Floating: 2.19 sec
  - fSig: about twice of the floating-p simulation time
  - gFix: 340 sec
  - pFix: 16.3 sec
  - VHDL: 181 sec
  - SPW: 60 sec

## Architecture considerations

- Fixed-point performance is architecture dependent

```
gFix a(12, 0, "tsr")
gFix b(12, 0, "tsr")
gFix c(12, 0, "tsr")
gFix d(10, 1, "tsr")
```

```
d = a + b + c;
```

```
gFix a(12, 0, "tsr")
gFix b(12, 0, "tsr")
gFix tmp(10, 1, "tsr")
gFix c(12, 0, "tsr")
gFix d(10, 1, "tsr")
```

```
tmp = a + b;
d = tmp + c;
```

## 4. Scaling

- Can easily be determined from the range information
  - $IWL \geq |R(x)| \text{ ceiling}$

## Wordlength determination

- For uniform wordlength determination
  - Needs only one iteration of search using fixed-point simulation (16bit –ok-> 12bit –no-> 14bit-ok->13bit...)
- For different wordlength for each variable
  - It is needed to group the variables that can have the same wordlength
  - Needs to determine the minimum wordlength for each group
  - Then try to find out the combined wordlength satisfying the performance

## Overall procedure

- C or C++ programming using floating-point arithmetic
- Integer wordlength determination using fSig class
- Minimum wordlength determination (gFix)
- Optimum wordlength determination (gFix)

## Conclusion

- Fixed-point simulation for existing C program can be conducted fairly easily using C++ fixed-point class.
- C program level fixed-point simulation takes much less time than the assembly or RTL level fixed-point simulation, but should be careful for bit-accurate results.

## Overview of this talk

1. Fixed-point arithmetic and system design
2. Fixed-point simulation method
3. Autoscaler (floating-point to integer)
4. Fixed-point optimizer (wordlength opt.)

## Floating-Point to Integer C Program Converter

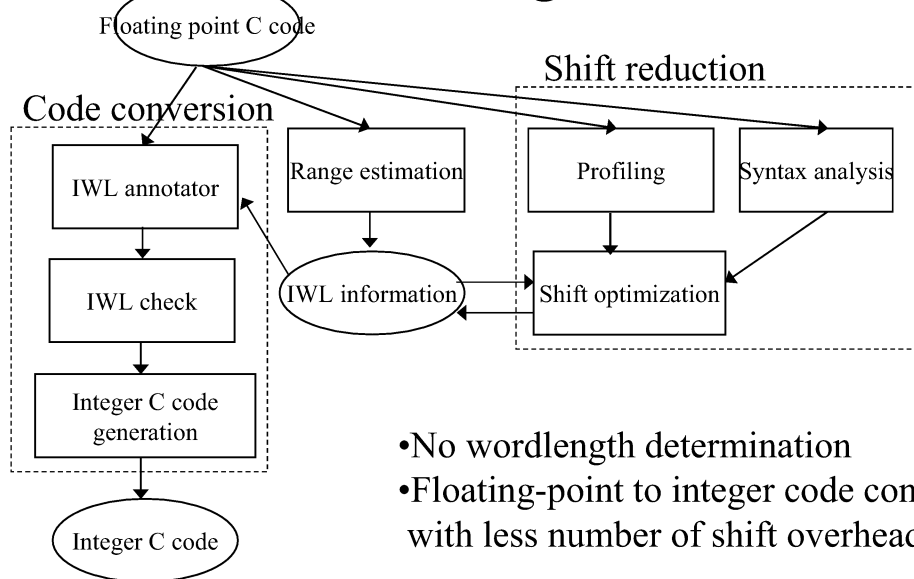
# Contents

- Introduction
  - Simulation based integer word-length determination
  - Code conversion
  - Shift optimization
  - Implementation examples
  - Conclusion
- Reference: Ki-Il Kum, Jiyang Kang and Wonyong Sung, "AUTOSCALER for C: An Optimizing Floating-point to Integer C Program Converter for Fixed-point Digital Signal Processors," IEEE Tr. Circuits and Systems II, Vol. 47, No. 9, September 2000, pp. 840-848.

# Introduction

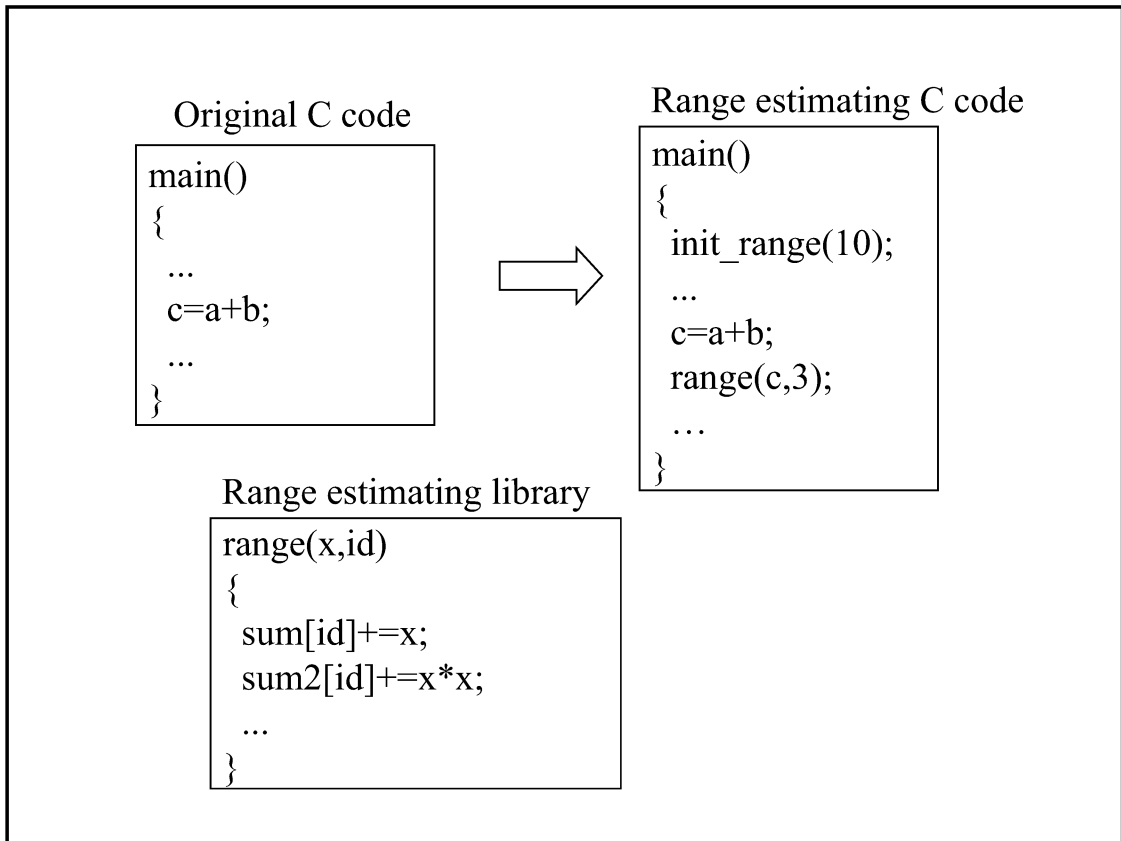
- Motivation
  - high level language for DSP
    - reduce development time, portability
  - fixed-point DSP
    - scaling is important for avoiding overflows and for accuracy
- Previous research
  - autoscaling assembler
  - C++ based fixed-point optimization utility
- Employed development tool
  - SUIF compiler system
    - parser, C code generator
    - all processes are performed with the SUIF data structure

# Overall design flow



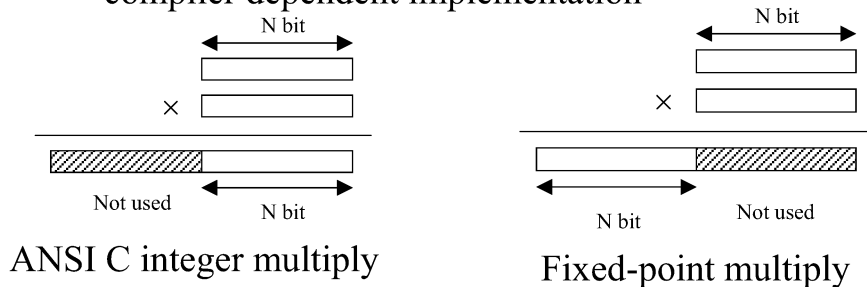
## Functional call based fixed-point simulation

- C++ class based method
  - applicable to static variables
- function call based method
  - applicable to static or automatic variables
    - recursion is allowed
  - automatically generate range estimating code
  - all floating variable in all symbol tables are given its identification number
  - statistic data are collected in a function call inserted for each assignment
  - 2.7 times faster than the C++ class based method
  - Implemented using SUIF compiler front end



### Code conversion

- Float to integer with known integer wordlength
- Fixed-point arithmetic rules
  - add/sub/assign
    - align binary point
  - multiply
    - high word of the multiplied result is used
    - compiler dependent implementation





Fixed-point arithmetic rules.

	Floating -point	fixed-point			result IWL
		Ix>Iy,Iz	Iy>Ix,Iz	Iz>Ix,Iy	
Assignment	x = y	x = y>>(Ix-Iy)	x = y<<(Iy-Ix)	-	Ix
Addition/ Subtraction	z = x+y	x+(y>>(Ix-Iy))	(x>>(Iy-Ix))+y	(x>>(Iz-Ix))+ (y>>(Iz-Iy))	max(Ix,Iy,Iz)
Multiplication	x*y	mulh(x,y)			Ix+Iy+1 or Ix+Iy

z: variable storing the result

Implementation of fixed-point multiplication.

Target processor	Implementation
TMS320C50	#define mulh(x,y) ((x)*(y)>>16)
TMS320C60	#define mulh(x,y) mpyh(x,y)
Motorola 56000	<pre> __inline int mulh(int x, int y) {     int z;     __asm("mpy %1,%2,%0":"=D"(z):"R"(x),"R"(y));     return z; } </pre>

## Code conversion (2)

- IWL constraints for array and pointer variables
  - each array element has the same IWL
  - pointer operation with different IWL is prohibited
    - assignment
    - function parameter

## Code conversion (3)

- Implementation using SUIF
  - IWL annotation
    - read IWL information file and modify symbol table
  - IWL check
    - floating-point variables without IWL info are checked
    - IWL constraints are checked
  - integer C code generation
    - type conversion of variables in symbol tables
    - expression tree conversion by scaling shift insertion

## Shift Optimization

- Scaling shifts are reduced by equalizing the relevant IWL's
- Architecture dependent (barrel shifter or not (DSP56000))
- Expression conversion
  - expressions are converted to simple sum of products form

$$x_i = \sum_{j,k} x_j * x_k + \sum_l x_l$$
$$x_i = \left\{ \sum_{j,k} ((x_j * x_k) \gg s_{j,k}) + \sum_l (x_l \gg s_l) \right\} \ll s_i$$

- shift amount calculation

$$I_{rhs} = \max_{j,k,l} (I_{x_j} + I_{x_k} + 1, I_{x_l}, I_{x_i})$$

$$s_{j,k} = I_{rhs} - (I_{x_j} + I_{x_k} + 1)$$

$$s_l = I_{rhs} - I_{x_l}$$

$$s_i = I_{rhs} - I_{x_i}$$

## Shift optimization (2)

- Shift overhead is calculated with the IWL of variables  $c_i = \sum_i (d_i + \sum_j e_{i,j})n_i$

- Shift reduction without a barrel shifter

– minimize  $x_{k_0} \leq x_k \leq x_{k_0} + b$

$$x_k = x_l$$

– constraints  $d_i \geq 0$

$$e_{i,j} \geq 0$$

$$e_{i,j} = d_i + x_k - (x_l + x_m + 1)$$

$$e_{i,j} = d_i + x_k - x_l$$

– solve using ILP

## Shift optimization (3)

- Shift reduction using a barrel shifter

– minimize

$$c_i = \sum_i (f_B(d_i) + \sum_j f_B(e_{i,j}))n_i, \quad f_B(x) = \begin{cases} 1, & x \neq 0 \\ 0, & x = 0 \end{cases}$$

– solve using simulated annealing

## Shift optimization (4)

- Implementation using SUIF
  - profiling
    - insert a function call which collect profiling information after each floating-point expression
  - syntax analysis
    - simplified parse tree is generated
  - shift optimization
    - generate a constraint file for ILP solver
    - generate a simulated annealing program
  - read initial IWL information determined by range estimation, write back optimized IWL information

## Implementation examples

- Fourth order IIR filter

### Floatig-point C code

```
float a1[3] = { 1, 0.355407, 1.0 };
float a2[3] = { 1, -1.091855, 1.0 };
float b1[2] = { 1.66664, -0.75504 };
float b2[2] = { 1.58007, -0.92288 };
float d1[2], d2[2];
void iir4(float *x, float *y)
{
    float x1, y1, t1, t2;
    x1 = 0.01 * *x;
    t1 = x1 + b1[0]*d1[0] + b1[1]*d1[1];
    y1 = a1[0]*t1 + a1[1]*d1[0] + a1[2]*d1[1];
    d1[1] = d1[0];
    d1[0] = t1;
    t2 = y1 + b2[0]*d2[0] + b2[1]*d2[1];
    *y = a2[0]*t2 + a2[1]*d2[0] + a2[2]*d2[1];
    d2[1] = d2[0];
    d2[0] = t2;
}
```

### Integer C code

```
#define sll(x,y) ((x)<<(y))
int a1[3] = { 1073741824, 381615360, 1073741824 };
int a2[3] = { 1073741824, -1172370379, 1073741824 };
int b1[2] = { 1789541073, -810718027 };
int b2[2] = { 1696587243, -990934855 };
int d1[2];
int d2[2];
extern void iir4(int *x, int *y)
{
    int x1;
    int y1;
    int t1;
    int t2;
    x1 = sll(mulh(1374389534, *x), 2);
    t1 = sll((x1 >> 4) + mulh(*b1, *d1) + mulh(b1[1], d1[1]), 2);
    y1 = mulh(*a1, t1) + mulh(a1[1], *d1) + mulh(a1[2], d1[1]);
    d1[1] = *d1;
    *d1 = t1;
    t2 = sll((y1 >> 4) + mulh(*b2, *d2) + mulh(b2[1], d2[1]), 2);
    *y = sll(mulh(*a2, t2) + mulh(a2[1], *d2) + mulh(a2[2], d2[1]), 3);
    d2[1] = *d2;
    *d2 = t2;
}
```

## Implementation examples (2)

Performance comparison for the fourth order IIR filter.

	# of cycles			SQNR	
	floating-p.	integer	speed-up	floating-p.	integer
'C50	2,980	100	29.8	-	49.3dB
'C60	3,659	9	406.6	-	57.9dB
56000	26,282	921	28.5	-	78.5dB

## Implementation examples (2)

The shift reduction results of the fourth order IIR filter.

IWL increment upper bound		0 (no shift reduction)	3	Infinite
# of shifts in C codes		7	4	2
'C50	# of cycles	100	96	94
	speedup	-	4%	6%
	SQNR	49.3dB	51.2dB	54.1dB
'C60	# of cycles	9	6	8
	speedup	-	33%	11%
	SQNR	57.9dB	57.1dB	54.2dB
# of shifts in C codes		5	3	2
56000	# of cycles	921	675	577
	speedup	-	27%	37%
	SQNR	78.5dB	78.5dB	78.5dB

## Implementation examples (3)

- QCELP
  - 16 source and 4 header files, total of 3648 lines
  - 381 floating-point variables
  - floating-point code: SQNR 17.9 dB
  - converted integer code: 17.36 dB
  - speedup: 24.6 times faster
  - shift optimization
    - shift cost is reduced by 88%
    - 4.5 % speedup
    - 0.1 dB performance degradation

## Conclusion

- Read and generate ANSI C compliant programs
- IWL information is kept in separate file.
- Target specific fixed-point multiplication and scaling shift optimization
- Converted integer C codes are 5 - 400 times faster than the floating-point C codes
- Fast simulation due to high level language simulation

$$I_{rhs} = \max_{j,k,l} (I_{x_j} + I_{x_k} + 1, I_{x_l}, I_{x_i})$$

$$s_{j,k} = I_{rhs} - (I_{x_j} + I_{x_k} + 1)$$

$$s_l = I_{rhs} - I_{x_l}$$

$$s_i = I_{rhs} - I_{x_i}$$

## Overview of this talk

1. Fixed-point arithmetic and system design
2. Fixed-point simulation method
3. Autoscaler (floating-point to integer)
4. Fixed-point optimizer (wordlength opt.)

# Word-Length Optimization for High Level Synthesis of Digital Signal Processing Systems

## Contents

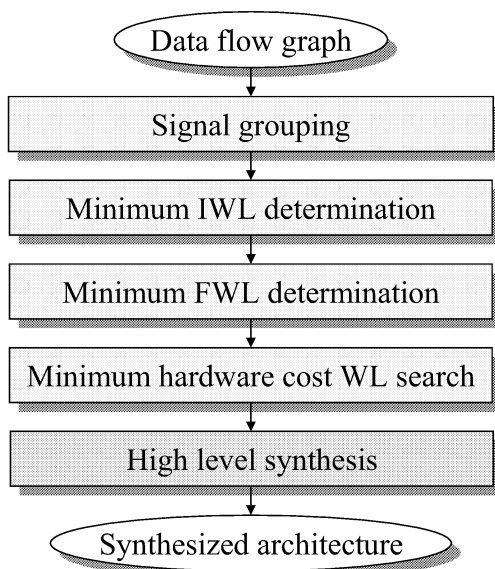
- Introduction
- Word-length optimization
  - DFG partitioning
  - Multiplication conversion
  - Minimum IWL, FWL determination
  - Scheduling with WL information
  - FU WL optimization
  - Shifter minimization
- Implementation examples
- Concluding remarks
- Reference: Ki-Il Kum, Wonyong Sung , "Combined word-length optimization and high-level synthesis of digital signal processing systems", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems , vol. 20, no. 8, pp. 921-930, August 2001.



# Introduction

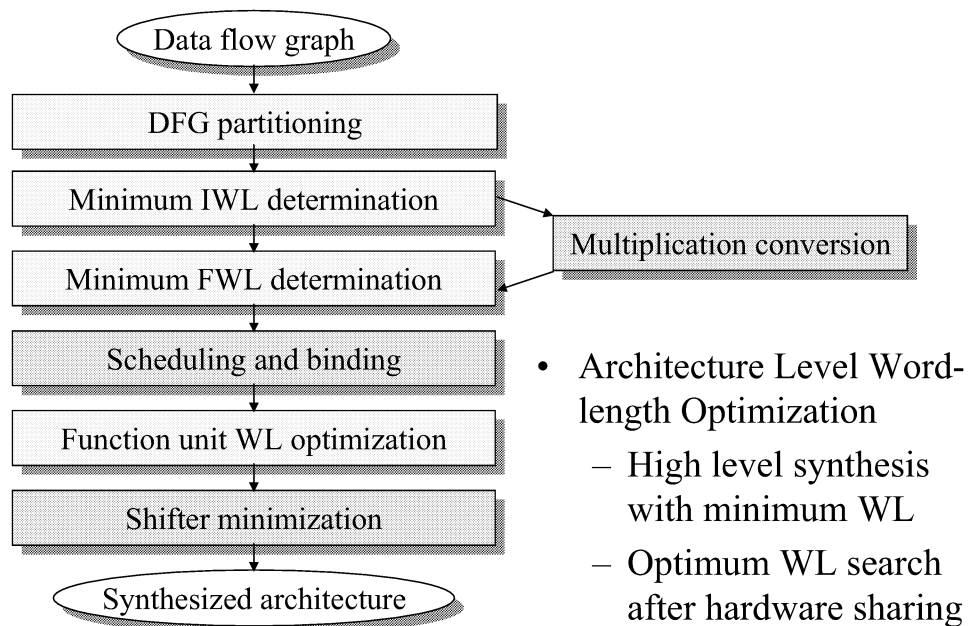
- Word-length determination is important for VLSI implementation (not integers).
- Objectives
  - To satisfy DSP performance requirements
  - To minimize hardware cost or power consumption
  - To meet time constraint (scheduling problem)

# Previous approach



- High level synthesis after algorithm level WL optimization
  - Hardware sharing is not considered in algorithm level WL optimization.
  - Too much simulation time for minimum hardware cost WL search
  - WL information is not considered in conventional high level synthesis.
  - WL can be increased during the high level synthesis.

# Proposed approach

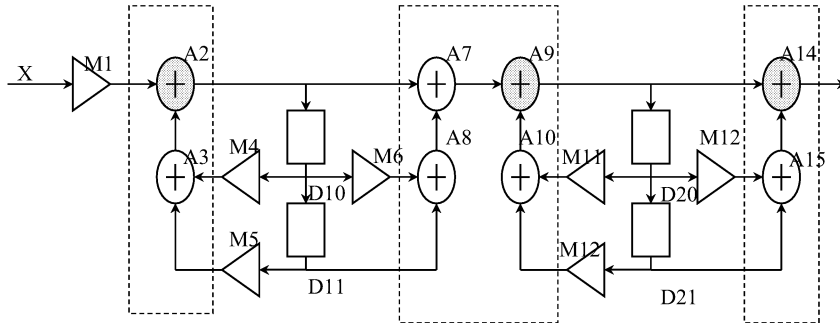


## Partitioning of data flow graph (1)

- Purpose
  - To reduce simulation time for the determination of the FWL.
  - The FWLs in a partition are determined together.
- Algorithm
  - Select an adder whose output is used for a multiplier input or a system output.
  - Select an adder whose output is used for more than two operators.
  - Search adders from the selected adders to the input direction until the other selected adder or multiplier is encountered.

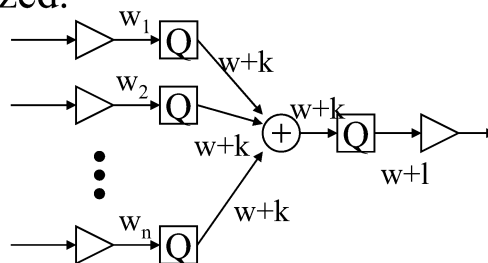
## Partitioning of data flow graph (2)

- A fourth order IIR filter example



## Partitioning of data flow graph (3)

- Quantization model
  - Both input and output of multipliers are quantized.



$$E_s = \frac{nE}{4^k} + \frac{E}{4^l} \quad E = \frac{(2^{-FWL})^2}{12}$$

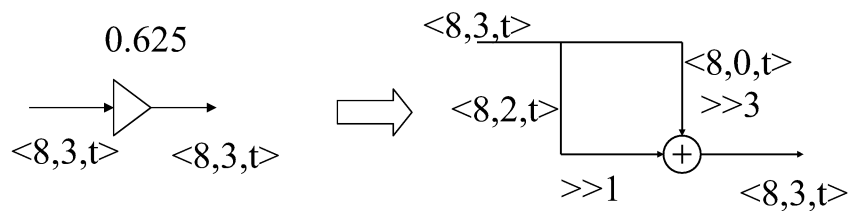
## Determination of minimum IWL

- Minimum FWL of a signal is determined with the range of the signal which is estimated using simulation results.
- Minimum IWL for the fourth order IIR filter

signals	min IWL
A9, D20, D21, M11, M12, M13	16
X, A10, A14	15
A7, A15	14
M4, A8	13
A2, D10, D11, A3, M5	12
M6	11
M1	10

## Constant multiplication conversion

- Implementation with shifters and adders
- Scaling shifters can be used for the shifts
  - concept of shifted value reading from registers



## Determination of minimum fractional word-length (1)

- Min FWL of a signal satisfies the fixed-point performance of a system, when the other signals have enough WL.
- Lower bound of optimal FWL
- FWL of output signal for an adder cluster is determined by simulation.
- FWL of other signals are determined using noise model equation.

## Determination of minimum fractional word-length (2)

- Minimum FWL for the fourth order IIR filter

cluster	adders	output FWL	adder FWL
1	A2,A3	-2	-1
2	A9,A10,A7,A8	-6	-4
3	A14,A15	-7	-6

signals	min FWL
M1,A3,M4,M5	-1
A2,D10,D11	-2
M6,A7,A8,A10,M11,M12	-4
A9,D20,D21	-6
M13,A15	-6
A14	-7

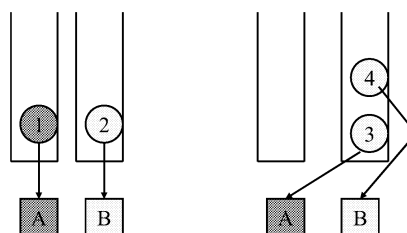
## Scheduling with word-length information (1)

- Modified list scheduling
  - Ready lists are generated for each WL of FU.
  - First, large WL OP is scheduled to large WL FU.
  - Small WL OP can be scheduled to large WL FU, when the large WL FU is not used.

```
insert_ready_operations;
cstep = 0;
while (not all lists are empty) {
  cstep = cstep + 1;
  for (all list from largest WL
       to smallest WL) {
    for (all resource from largest WL
         to smallest WL) {
      if (the resource is free and
          can do the first node
          of the list) {
        schedule_operation;
        delete first node from the list;
      }
    }
  }
  insert_ready_operations;
}
```

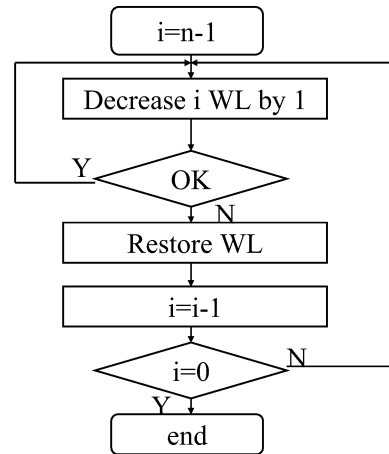
## Scheduling with word-length information

- Scheduling algorithm
  - Insert ready operations for each FU having different WL
  - First, for the ready list of largest WL, the resource of largest WL is assigned.
  - When there is no ready operation in the list, a ready operation in the other list having smaller WL is assigned to the resource.
  - If all the ready operations are scheduled with free resources, control step is increased and ready lists are generated again.



## Scheduling with word-length information (2)

- Number of FU and WL determination
  - List scheduling is resource constrained algorithm.
  - Our problem is control step constrained algorithm.
  - Iterative scheduling with modifying the # of FU and WL until the control step constraints are satisfied.
    - Find minimum # of FU with enough WL
    - Reduce WL step by step



n : # of FU

## Scheduling with word-length information (3)

- Scheduling result for the fourth order IIR filter
  - four 9b, three 11b multiplications : 7 muls (16b coeff)
  - three 13b, one 12b, five 11b, one 10b, two 9b ALUs : 12 ALUs
  - hardware cost :  $8n$  for  $n$ -bit multipliers,  $n$  for  $n$ -bit adders

cstep	mul	ALU	total cost
7	11,11	13,11,9	209
8	11,9	13,13	186
9	11,9	13,11	184
10	11	13,11	112

# Register binding with word-length Information

- Modified clique partitioning
  - Select a node with largest WL first for finding a maximum clique.
  - Large WL register is allocated first.

```
clique_partition(G(V,E)) {
  while (G(V,E) is not empty) {
    C = max_clique(G(V,E));
    delete C from G(V,E);
  }
}

max_clique(G(V,E)) {
  C = node with largest WL;
  repeat {
    repeat {
      U = set of nodes not included in C
        and adjacent to all nodes of C;
      if (U is empty) {
        return C;
      }
      else {
        select a node with largest WL in U;
        add the node to C;
      }
    }
  }
}
```

# Register binding with word-length information (2)

- Algorithm
  - while  $G(V,E)$  is not empty, extract the largest clique  $C$  from the graph  $G$
  - largest clique finding algorithm
    - conventional
      - select a initial node with largest degree
    - proposed
      - select a initial node with largest WL
      - select a node with largest WL from the nodes adjacent to all nodes included in the clique



## Word-length optimization of functional units

- System performance is better than desired value because of hardware sharing
  - Decrease WL of selected FUs
- System performance is worse than desired value because minimum WLs are used
  - Increase WL of selected FUs
- Select FU for minimizing the total hardware cost

## Shifter minimization

- Shifters are used for aligning adder inputs and scaling multiplier outputs.



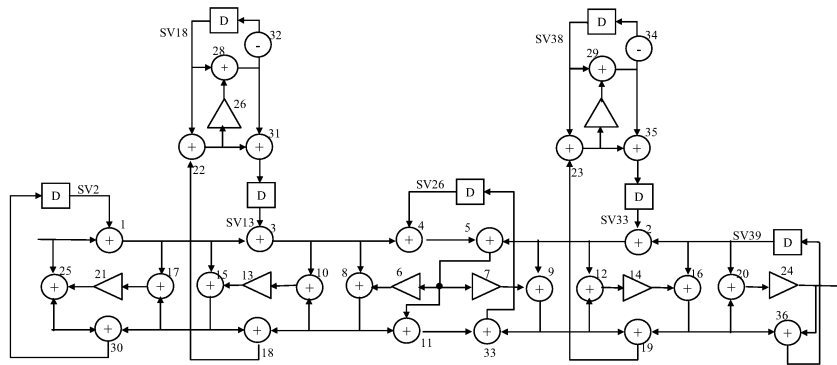
- Shifters are implemented with multiplexers.
- Shifter hardware can be reduced by exchanging the two inputs of FU.

– ex)

cstep	shift bits	shif bits		cstep	shift bits	shif bits
1	1	2	⇒	1	2	1
2	2	1		2	2	1

## Implementation examples (1)

- Fifth order elliptic filter



## Implementation examples (2)

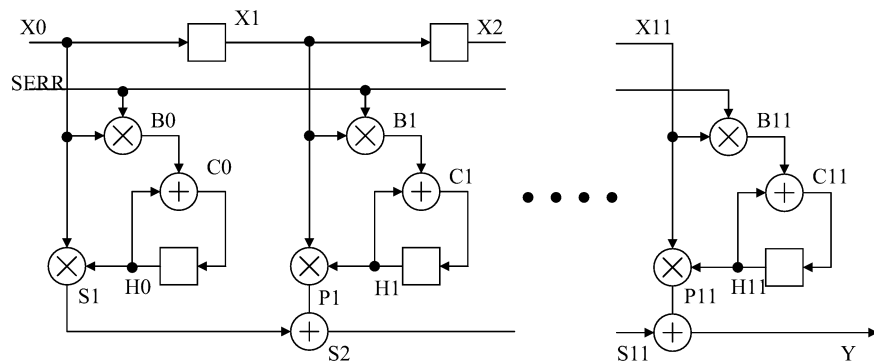
- Design spec.
  - CSD optimized 6 bit coefficients
  - White noise with the range of  $(-2^{15}, 2^{15})$  is used for input.
  - SQNR 40 dB is used for the desired fixed-point performance.
- DFG partitioning
  - grouped into 17 adder clusters
- min IWL
  - 15 bit to 19 bit
- min FWL
  - -7 bit to -10 bit

## Implementation examples (3)

- A 6b by 11b multiplier, a 12b ALU, and a 11b ALU are used for 18 control steps.
- 42.86 dB SQNR
- Further optimization of FU is not possible.
- Shift minimization
  - # of multiplexer inputs are reduced from 3, 4, 3, and 3 to 3, 3, 2, and 3.
- Constant multiplication conversion
  - 8 mul  $\rightarrow$  10 ALU operations
  - A 14b ALU, and a 12b ALU are used for 21 control steps.

## Implementation examples (4)

- Adaptive LMS Filter



## Implementation examples (5)

- Design spec.
  - 12th order adaptive LMS filter for a channel identification
  - The average power of the error signal after some time off period is used for the fixed-point performance.
- DFG partitioning
  - grouped into 13 adder clusters
- WL of the signals

signals	IWL	FWL	WL
X0,X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11	15	-5	11
S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,Y P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11	15	-1	15
H0,H1,H2,H3,H4,H5,H6,H7,H8,H9,H10,H11 C0,C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11 B0,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11	0	12	13
SERR	-18	27	10

## Implementation examples (6)

- An 11b by 13b multiplier, an 11b by 10b multiplier, a 15b ALU, and a 13b ALU are used for 13 control steps.
- Since the fixed-point performance after hardware sharing is not satisfied, WL optimization of FU's is performed.
  - An 11b by 10b multiplier and 15b ALU are replaced by an 11b by 11b multiplier and a 16b ALU.

## Concluding remarks

- Architecture level word-length optimization
  - WL optimization and high level synthesis are combined.
- High level synthesis techniques using word-length information
  - Modified list scheduling
  - clique partitioning based register allocation
- Signal grouping using hardware sharing information
  - Optimum WL search time is reduced.

## Fixed-point optimization tools

- Fixed-point optimization utility for C based DSP programs
  - range estimator, fixed-point simulator
- Fixed-Point Optimizer for SPW
  - search based wordlength optimization method for reducing the hardware cost
- Autoscaler
  - automatic determination of the number of shifts for the TMS 320C25 and C50 implementations

# **Analysis and Optimization of Power Consumption for an ARM7-based Multimedia Handheld Device**

Wonyong Sung

School of Electrical Engineering  
Seoul National University, Korea

# Analysis and Optimization of Power Consumption for An ARM7-based Multimedia Handheld Device

Wonyong Sung  
School of Electrical Engineering  
Seoul National University

July 18, 2003

VLSI Signal Processing Lab.



Seoul National University

## DSP implementation using RISC CPU's

- Digital signal processing
  - Essential for multimedia (MP3, CELP internet telephone, MPEG video, H.263 video communication,..)
  - Requires a lot of arithmetic (mul, add) and memory access operations -> large power consumption
- Traditional approaches: programmable digital signal processors (TI's C54x,...)
  - Efficient for multiplication and many-operands algorithms
  - Low-power for DSP only systems
  - \*Not good for compiler based developments
  - \*Not good for control intensive, large scale applications
  - \*Prog DSP's *were* fast when compared to conventional controllers (68000,..), but no more.

VLSI Signal Processing Lab.



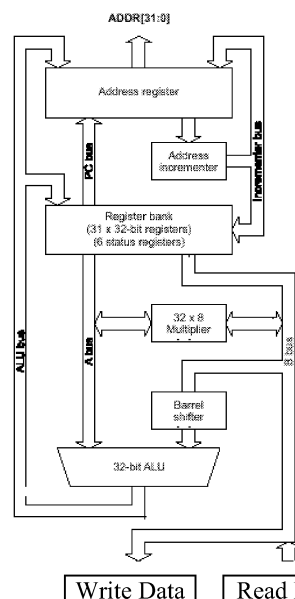
Seoul National University

## Architectural requirements for efficient DSP processing

Feature	Architectural components	Usage
High bandwidth memory I/O	Harvard architecture (separate program and data bus)	Simultaneous access of instr and data
Specialized addressing modes	Dedicated address generation units	Indirect addressing with post-inc, bit reversal and circular addressing
Fast multiply-accumulate	Single cycle MAC, parallel execution	Most DSP algorithms
Efficient looping	Hardware looping	Iterative DSP algorithms (FIR, FFT)



## Block Diagram of ARM 7TDMI CPU

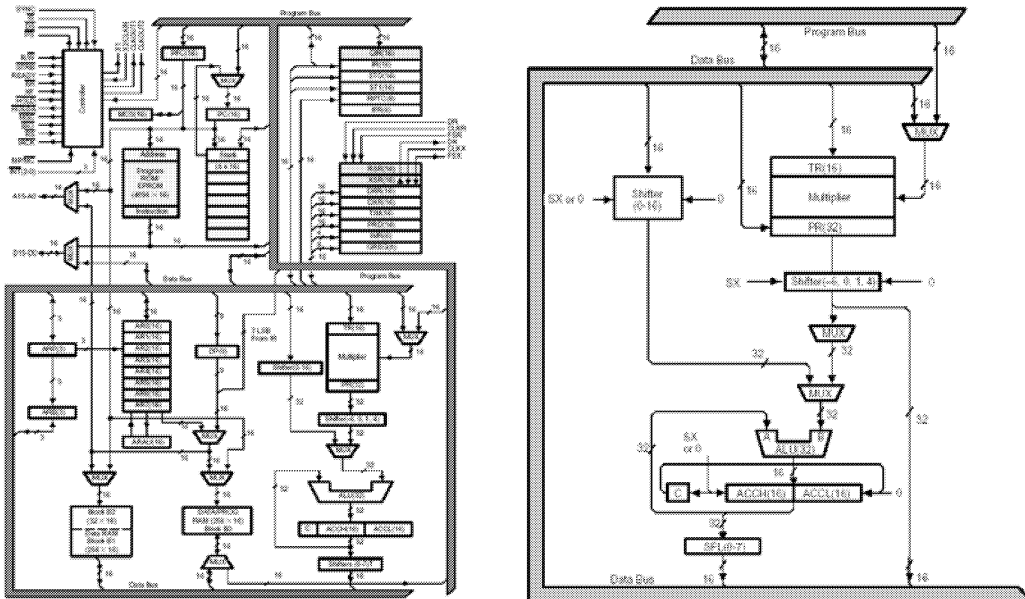


- Von Neumann Architecture (Unified Instruction/Data BUS)
- Multiplier (32\*8)
- MAC (Multiply and Add : MLA)
- Barrel Shift
- 32bit ARM/16bit Thumb Instruction
- 31 32-bit general purpose register





## Block Diagram of TI TMS320C25



VLSI Signal Processing Lab.



Seoul National University

## Comparison of RISC CPU's and programmable DSP's

### A. RISC CPU

- Small # of instructions. (less than 128)
- Simple instruction format (l.t. 4)
- Simple addressing modes (l.t. 4)
- Single cycle execution of most instructions
- >
- Simple control unit (hardwired controller)
- Good compiler support
- Larger number of instructions

### B. DSP CPU

- Many special purpose instructions  
ex) FFT (bit reversed addressing.  
Viterbi (CSSU)
- Many instruction formats (better code density)
- Zero-loop overhead  
ex) RPT, RPB
- Address generation units  
ex) circular addressing
- Single Cycle MAC (multiply-accumulation)

VLSI Signal Processing Lab.



Seoul National University

## Changing needs

- Current embedded systems require a large size of applications
  - Requires OS (such as Linux or WinCE)
  - Requires good compiler support
  - Requires DRAM
  - Requires cache memory systems to supplement DRAM memory
- Intermediate Solution: Combining RISC CPU and DSP (most wireless telephone solutions today).
  - TI's OMAP, Qualcomm CDMA chipset.
  - Can use existing DSP libraries
  - Ideally it can be power-efficient
  - Data communication overheads (usually shared memory)
  - Requires different program development environments
  - More IP, complex hardware



## RISC CPU based DSP systems

- RISC CPU based systems
  - More flexible, can handle large applications
  - But requires a more number of clock cycles for running a same application.
- How can we narrow the gap?
  - Increased clock speed(>70MHz) with less current per MHz.
  - Increase cache size and bandwidth (I, D separate caches)
  - Equip full precision multipliers
  - Employ efficient programming styles
    - Loop optimization (because it has no repeat controller)
    - Memory access optimization
    - Strength reduction (precision reduction)



## Speaking Partner

- An ARM7 based multimedia handheld device for foreign language learning
  - Cheap ARM7 CPU (from Samsung, LCD controller, DRAM controller, 8KB unified cache) based (around \$5).
  - Implemented MP3 decoding (for audio), CELP decoding and ADPCM (for speech), GIF animation, and speech recognition
  - Can use about 15 hrs with 2 AA size batteries
    - We will analyze the power consumption at each hardware components.



## Overview of My Talk

- Efficient Programming of DSP Algorithms on RISC CPU's
- Speaking Partner and Power Consumption Analysis



# Efficient Programming of DSP Algorithms on RISC CPU's

Wonyong Sung



## Contents

1. Introduction
2. Optimization of DSP Algorithms on a RISC CPU Architecture
  - Profiling for speed and code optimization
  - Loop optimization
  - Memory optimization
3. ARM7-based optimizations
  - Loop fusion
  - Loop Unrolling
  - Merging arrays
  - Circular Addressing
  - ETC
  - CELP Decoder Implementation and Comparison
4. Comparison of DSP and RISC Performances
5. Conclusion

Reference: Jiyang Kang, Jongbok Lee, Wonyong Sung , "A Compiler-Friendly RISC-Based Digital Signal Processor Synthesis and Performance Evaluation", JOURNAL OF VLSI SIGNAL PROCESSING , vol. 27, pp. 297-312, March 2001.



# 1. Introduction

- RISC based implementation of DSP applications
  - Speech: MP3 decoder, MPEG video, H.263,...
  - Graphics: Laser printer
  - Large size, multi-tasking applications
  - Good compiler support, supports large system memory (DRAM)
- However, RISC processors may lack
  - Full precision multiplier
  - Large bandwidth memory (Harvard architecture - multiple on-chip memories)
  - Repeat unit (decrease the loop-overhead)
  - Dedicated address generation units
- Need to increase the performance by employing different programming techniques



# 2. Optimization of DSP algorithms on a RISC CPU

- Embedded DSP applications are extremely **computationally demanding**, but often require **low power, low memory use**.
- Optimization yields a competitive advantage
  - **Execution speed**
    - Can use a slower and less expensive processor
    - Allows upgraded functionality using the same processor
    - More functionality, e.g., more channels
  - **Memory usage**
    - Memory usage contributes to system cost
  - **Power consumption**
    - Improved battery life or reduced power supply size



## Optimization objectives

- **Speed-optimized code**
  - May consume less energy
  - But often sacrifices memory (loop unrolling, function inlining)
- **Memory-optimized code**
  - For less memory size
  - May reduce memory accesses and hence energy consumption
  - But generally slows down code (thumb mode in ARM)
- **Power-optimized code**
  - Avoid instructions requiring large energy consumption
  - Reduce external memory accesses
  - Usually, speed can mean less-energy.
- **Performance gain vs. Development time**



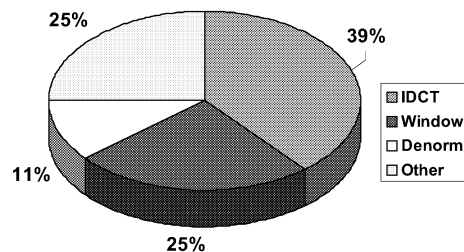
## Where to optimize?

- **The 80/20 rule**
  - 20% of the software uses 80% of the processing time
  - and 80% of the software uses 20% of the processing time
- **DSP algorithm software spends most of its time in loops**
  - This is where optimization for speed and power is most valuable
  - So loop optimization is important (for speed and power)
- **Control software takes up the bulk of the program memory, but only a fraction of the processing time**
  - Control software is the best candidate for memory optimization (instead of speed or power)



## Application profiling

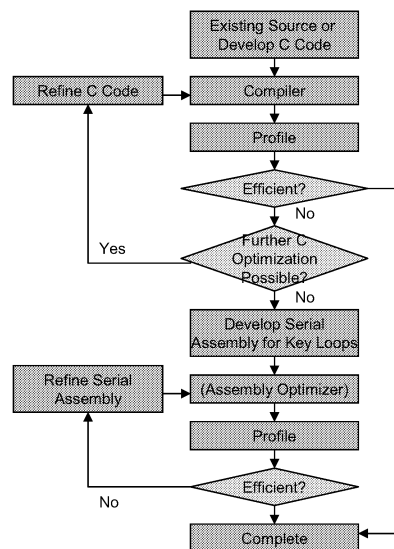
- Example: AC-3 execution time profile
  - Which modules take the longest time?



- Which modules take the most memory?
- Which modules are the most power-consuming?

## Code generation flow

- Iterate these steps:
  1. **select algorithm**
  2. **develop C code**
    - source debugger, profiler
  3. **refine C code**
    - code transformations, compiler options, intrinsics, statements
  4. **develop and refine assembly**
    - hand optimize critical functions



## Compiler shortcomings

- **Compilers typically don't generate sufficiently optimized DSP software**
  - Some DSP features aren't supported in the most common high-level languages
  - Memory layout is important, but compilers don't know that
  - Specialized addressing modes aren't supported
  - The best optimizer is still the human mind!
- **The programmer often must hand-optimize software**
  - But then there's the development time..
- ***Help the compiler!***



## Loop optimizations

- **DSP algorithm software spends most of its time in loops**
  - This is where optimization for speed and power is most valuable
  - So loop optimization is important (for speed and power)
- Loop Unrolling
- Loop Interchange
- Loop Termination





## Loop unrolling

- **Small loops can be unrolled for higher performance**
  - but with disadvantage of **increased code size**
  - the loop counter updated less often
  - fewer branches executed (less overhead)

```
int countbit1(uint n)
{ int bits=0;
  while (n!=0)
  {
    if (n&1) bits++;
    n >>= 1;
  }
}
```

unrolled  
4 times  
→

```
int countbit2(uint n)
{ int bits=0;
  while (n!=0)
  {
    if (n&1) bits++;
    if (n&2) bits++;
    if (n&4) bits++;
    if (n&8) bits++;
    n >>= 4;
  }
}
```



## Loop interchange

- **make a loop having stride 1 reference to locate in the innermost loop**
- improve spatial locality

```
do i = 1,n
  do j = 1,n
    A[i] = A[i]+B[i,j]*B[j,i]
```

→

```
do j = 1,n
  do i = 1,n
    A[i] = A[i]+B[i,j]*B[j,i]
```



## Loop termination

- Loops are not equal:
  - **Count-down-to-zero loops *may* be faster**
  - **ex. ARM:**

```
int fact1(int n)
{ int i, fact=1;
  for(i=1;i<=n;i++)
    fact*=i;
  return (fact);
}
```

```
|L1|
MUL a3,a2,a3
ADD a2,a2,#1
CMP a2,a2
BLE |L1|
```

```
int fact2(int n)
{ int i, fact=1;
  for(i=n;i!=0;i--)
    fact*=i;
  return (fact);
}
```

```
|L1|
MUL a3,a2,a3
SUBS a2,a2,#1
BLE |L1|
```



## Memory optimizations

- So many issues here:
  - On-chip memory vs. Cache
  - On-chip vs. off-chip
  - Multiple memory banks
  - Synchronous DRAM (row access means less energy and less delay)
- Note:
  - **If your application runs too slow, check *memory* first!**



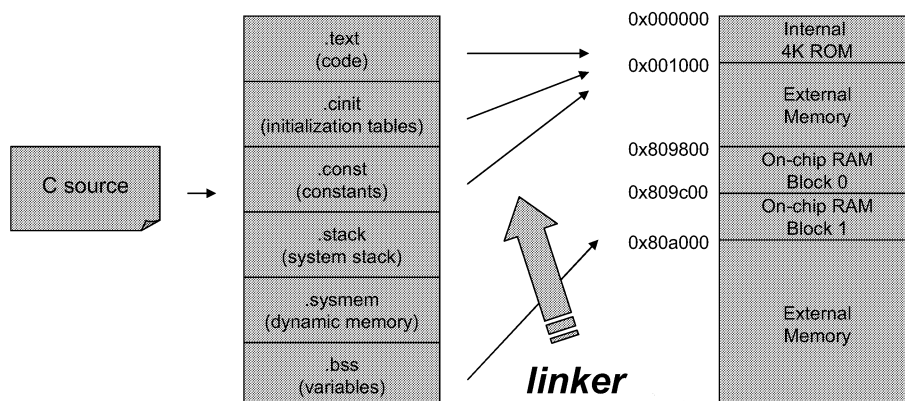
## On-chip memory vs. cache

- They are different:  
What's on your target processor?

	On-chip memory	Cache
Access time	deterministic	unpredictable
Data management	manual by programmer	automatic by hardware
Address space	separate fixed address space	'mirror' of dynamically determined address space
Traditionally employed in..	DSP	MCU, GPP
Note:	-	hybrid 'locking' features

## Where are my data: On-chip or Off-chip?

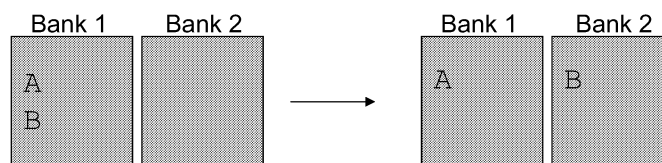
- Assembled or compiled
- Usually specified in your **linker command file (.cmd)**



## Arrange data in memory banks

- Virtually all DSP processors use **multiple memory banks**
  - Statements that require multiple data accesses per arithmetic operation can take advantage of multiple banks
  - Some GPPs also have multiple memory banks

sum += A[i] \* B[i]



## Inline functions

- **Most compilers support inline functions with a keyword (ARM: `__inline`) or a compiler option**
- Advantages:
  - no function call overhead
  - lower argument evaluation overhead
  - more optimizations possible
    - ex. MUL + ADD
- Disadvantage:
  - code size increase
- **Use only for small but critical functions**

```
__inline int square(int x)
{
    return x*x;
}
```

### 3. ARM7-based optimizations

1. Transform the DSP architecture to software optimization
  - reduce the loop overhead
  - circular addressing
  - indirect addressing
  - MLA (MAC)
2. Minimize memory access
  - reduce cache misses
  - reduce power consumption
  - reduce cycles (speedup)

=> maximize register usage



### 3.1 Loop fusion

- Reducing #of Loop overhead
- Reducing cache miss by temporal locality

```
for ( j = 0 ; j < LpcOrder ; j ++ )
  PrevLsp[j] = sub(PrevLsp[j], LspDcTable[j] ) ;
for ( j = 0 ; j < LpcOrder ; j ++ ) {
  Tmp = mult_r( PrevLsp[j], Lprd ) ;
  Lsp[j] = add( Lsp[j], Tmp ) ;
for ( j = 0 ; j < LpcOrder ; j ++ ) {
  PrevLsp[j]=add(PrevLsp[j],LspDcTable[j] ) ;
  Lsp[j] = add( Lsp[j], LspDcTable[j] ) ;}

→

for ( j = 0 ; j < LpcOrder ; j ++ ) {
  z=LspDcTable[j];
  x = sub(PrevLsp[j], z ) ;
  Tmp = mult_r( x, Lprd ) ;
  y = add( Lsp[j], Tmp ) ;
  PrevLsp[j]= add( x, z ) ;
  Lsp[j] = add( y, z ) ; }
```

PrevLsp : load (3->1)  
          store (2->1)  
LspDcTable: load (3->1)  
Lsp : load (2->1)  
      store(2->1)



## 3.2 Loop unrolling

-Reducing LDR and STR and #of Loop

-According to Bus width

32bit bus-width

16bit bus-width

```

for ( i = 0 ; i < SubFrLen ; i ++ )
    TmpVect[i] = Tv[i] ;
    →
for ( i = 0 ; i < SubFrLen/4 ; i ++ ){
    TmpVect[I*4] = Tv[i] ;
    TmpVect[I*4+1] = Tv[I*4+1] ;
    TmpVect[I*4+2] = Tv[I*4+2] ;
    TmpVect[I*4+3] = Tv[I*4+3] ;
}
    
```

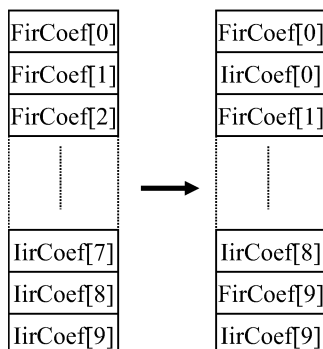
Tv : short (2bytes)  
int(4bytes)



## 3.3 Merging arrays

- Reducing Cache miss by spatial locality

FirCoef[10],IirCoef[10] -> Fir\_IirCoef[20]



```

for ( i = 0 ; i < LpcOrder ; i ++ ) {
    FirCoef[i] = mult_r( Lpc[i], PostfirFiltTable[i] ) ;
    IirCoef[i] = mult_r( Lpc[i], PostiirFiltTable[i] ) ;
}
    
```

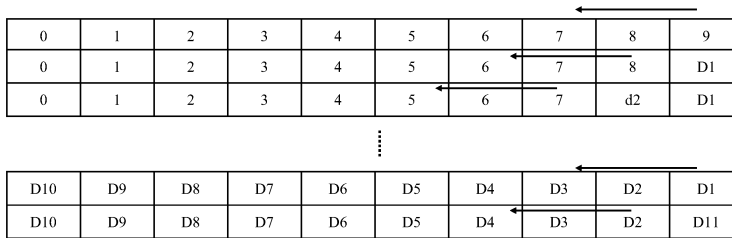


```

for ( i = 0 ; i < LpcOrder ; i ++ ) {
    Fir_IirCoef[i*2] = mult_r( Lpc[i], PostFiltTable[i*2] ) ;
    Fir_IirCoef[i*2+1] = mult_r( Lpc[i], PostFiltTable[i*2+1] ) ;
}
    
```



## 3.4 Circular addressing method



```

for ( j = 0 ; j < LpcOrder ; j ++ )
    Acc0 = L_mac( Acc0, Lpc[j], SyntfirDI[j] ) ;
for ( j = LpcOrder-1 ; j > 0 ; j -- )
    SyntfirDI[j] = SyntfirDI[j-1] ;
    SyntfirDI[0] = round( Acc0 ) ;
    ↓
for ( j =LpcOrder-1 ; j >= 0 ; j-- )
    Acc0 = L_mac( Acc0, Lpc[j],SyntfirDI[j] ) ;
    SyntfirDI[0] = round( Acc0 ) ;
    
```

## 3.5 ETC

### A. Multiplication by a constant

- Multiply by 6

ADD Ra,Ra,Ra,LSL #1 ;Ra\*3

MOV Ra,Ra,LSL #1 ;Ra\*2

-Multiply by 10 and Add(Rc)

ADD Ra,Ra,Ra,LSL #2 ;Ra\*5

ADD Ra,Rc,Ra,LSL #1 ;Ra\*2+Rc

## B. Calculation of Correlation

- Reduce # of Load

```
X=TmpVect[0]
for ( i = 1 ; i < SubFrLen ; i ++ ) {
    Acc0 = L_mac( Acc0, TmpVect[i], TmpVect[i-1] );
    Acc1 = L_mac( Acc1, TmpVect[i], TmpVect[i] );
    X=TmpVect[i];
}
```

## C. Indirect Addressing and Loop count

- Remove loop count
- Reduce Code size of Loop



## 3.6 Combined methods

(Loop fusion, circular addressing, loop unrolling, merging array)

```
/* FIR part */
for ( j = 0 ; j < LpcOrder ; j ++ )
    Acc0 = L_msu( Acc0, FirCoef[j], PostFirDI[j] );
for ( j = LpcOrder-1 ; j > 0 ; j -- )
    PostFirDI[j] = PostFirDI[j-1];
/* IIR part */
for ( j = 0 ; j < LpcOrder ; j ++ )
    Acc0 = L_mac( Acc0, IirCoef[j], PostIirDI[j] );
for ( j = LpcOrder-1 ; j > 0 ; j -- )
    PostIirDI[j] = PostIirDI[j-1];

↓

for ( j = LpcOrder-1 ; j >= 0 ; j -- ) {
    Acc0=L_msu( Acc0, Fir_IirCoef[j*2], PostFir_IirDI[j*2] );
    Acc0= L_mac( Acc0, Fir_IirCoef[j*2+1], PostFir_IirDI[j*2+1] );
}
```





## 3.7 Performance after each optimization

		Instruction	Cycles	
			SRAM	DRAM
Loop fusion	Before opt.	487	538	1576
	After opt.	150	167	436
Loop unrolling	Opt1	360	382	1076
	Opt2	90	109	446
Array merging	Opt1	150	181	536
	Opt2	140	162	446
Circular addressing	Opt1	4735	5935	21586
	Opt2	4377	4718	14268
Correlation	Opt1	354	491	1646
	Opt2	357	445	1341
Combined Method	Opt1	10197	12389	42717
	Opt2	9473	7920	22645

- reduce the loop overhead

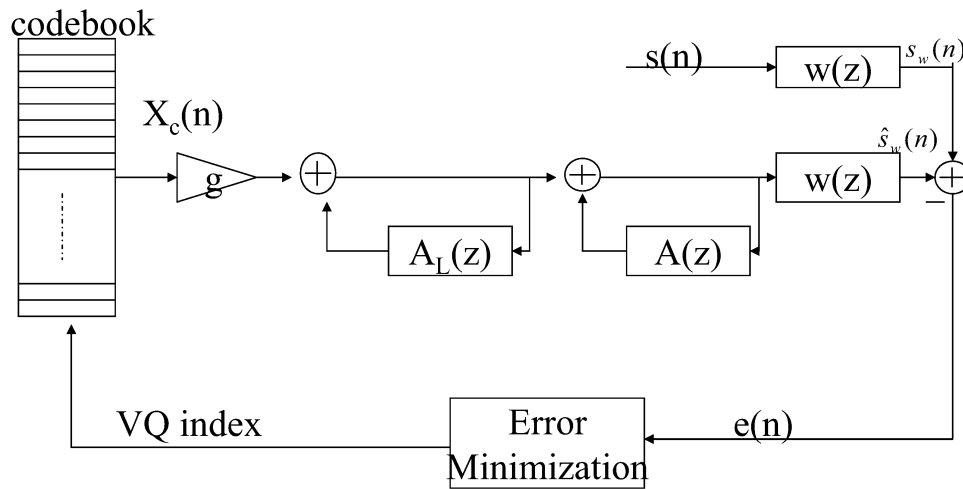


## 3.8 CELP decoder implementations

1. Code Excited Linear Predictive Coding
  - Combination of vector quantization and linear predictive coding
2. Used for internet and wireless telephony (G. 723, G.729, QCELP)
3. Quite good voice quality at the bit rate of 5.6 kbps to 16 kbps
4. Encoder is very demanding for the best code search, but decoder is less.



# Celp

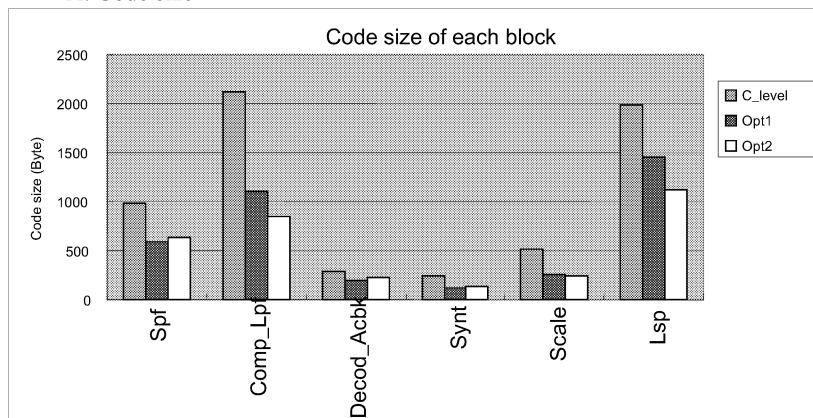


\* Federal standard 1016 Celp,  
 Vecor Sum Excited Linear prediction(VSELP)  
 Low-delay Celp

# CELP optimization results

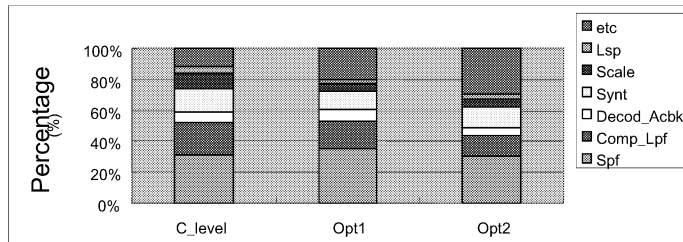
- Opt1 : loop fusion, Indirect addressing,
- Opt2 : loop unrolling, circular addressing, merging arrays

A. Code size

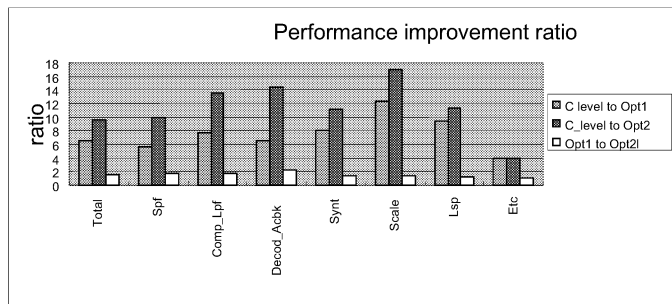


C\_level: 12.5kbytes  
 Opt1 : 8.3kbytes  
 Opt2 : 7.5kbytes

## B. Percentages for the processing of each block



## C. Performance improvement ratio



## 4. Comparison of DSP and RISC performances

- Performance comparison after the optimization
- Should be dependent on the algorithms
  - Data intensive and regular algorithms -> good for DSP
  - Control intensive algorithms -> good for RISC
- Should be dependent on the development methods
  - Assembly programming -> good for DSP (CISC, specific hw)
  - C compiler based -> good for RISC

## Example 1: FIR-IIR filtering(C) – a regular & multiply-intensive algorithm

- Formant postfilter (ARMA)

```

for(i=0;i < SubFrLen;i++)    {
    sum = In_data[i];
/* Fir Part */
    for(j=0;j<LpcOrder;j++)
        sum -= FirCoeff[j]*PostFir[j];
    for(j=LpcOrder-1;j>0;j--)
        PostFir[j]=PostFir[j-1];
    PostFir[0] = In_data[i];
/* Iir part */
    for(j=0;j<LpcOrder;j++)
        sum += IirCoeff[j]*PostIir[j];
    for(j=LpcOrder-1;j>0;j--)
        PostIir[j]=PostIir[j-1];
    PostIir[0] = sum;
}
    
```

-Loop Fusion,  
-Merging Array,  
-Loop Unrolling  
-Circular Addressing



## Example 1 : FIR -IIR filtering(ASM)

° ARM7TDMI

```

Spf5
LDRSH a2,[v8]
Spf6
LDR    a3,[v6],#4
MOV    a4,a3,ASR #16
MOV    a3,a3,LSL #16
MOV    a3,a3,ASR #16
LDR    v1,[v3],#4
MOV    v2,v1,ASR #16
MOV    v1,v1,LSL #16
MOV    v1,v1,ASR #16
MUL    v1,a3,v1
SUB    a2,a2,v1
MLA    a2,v2,a4,a2
CMP    v4,v6
ADDEQ  v6,v6,#0x28
LDR    a3,[v6],#4
MOV    a4,a3,ASR #16
MOV    a3,a3,LSL #16
MOV    a3,a3,ASR #16
LDR    v1,[v3],#4
MOV    v2,v1,ASR #16
MOV    v1,v1,LSL #16
MOV    v1,v1,ASR #16
MUL    v1,a3,v1
SUB    a2,a2,v1
MLA    a2,v2,a4,a2
CMP    v4,v6
ADDEQ  v6,v6,#0x28
CMP    lr,v3
BLT    Spf6
ADD    v3,lr,#0x28
;***** Load DecStat.PostIir[1]
MOV    v1,v6
ADD    v2,v4,#0x28
CMP    v2,v1
SUBEQ  v2,v2,#0x22
ADDNE  v2,v6,#0x6
LDRSH  a4,[v2]
MLA    a3,a4,v5,a3
MOV    a3,a3,LSL #1
;***** Store PostFir[0].PostIir[0]
MOV    a2,a2,LSL #16
BIC    ip,ip,#0xFF000000
BIC    ip,ip,#0x00FF0000
ORR    a2,a2,ip
STR    a2,[v6],#4
CMP    v4,v6
ADDEQ  v6,v6,#0x28
STRH   a4,[v8],#2
CMP    v8,a1
BLT    Spf5
    
```

-Loop Fusion,  
-Merging Array,  
-Loop Unrolling (x2)  
-Circular Addressing



° TMS320C54x

;AR3=&iir\_coef,AR4=&Fir\_coef,AR5=&Post\_Fir,AR6=&Post\_iir

	STM	#_DecStat+190,AR5	}	RPT,RPTB, circular addressing
	STM	#_DecStat+200,AR6		
	STM	#{LpcOrder-1},BK		
	STM	#SubFrLen,BRC		
	RPTB	L25-1		
	RPT	#LpcOrder-1		
	MAS	*AR4+%, *AR5+%,A		
	RPT	#LpcOrder-1		
	MAC	*AR3+%, *AR6+%,A		
L25:	STH	B,*(DecStat+181)		
	STM	#_DecStat+182,AR3		
	MVMM	SP,AR4		
	MAR	*+AR4(#12)		
	STH	A,*AR2+		



## Example : ADPCM

(Quantizer Scale factor Adaptation Part) - a control intensive algorithm

```
if(ap > (1 << 13)) al = (1 << 14);
else al = (ap << 1);
y = (al * yu + ((1 << 14) - al) * yl) >> 14;
```

° ARM7TDMI

```
;a2=ap, a3=a1,v1=yu,v2=yl
MOV a1,#0x2000 ;1 <<13
CMP a2,a1 ;if(ap>(1<<13)) ??
MOVGT a3,#0x4000 ;if(ap>al) al=1<<14
MOVLE a3,a2,LSL #1 ;else al=ap <<1
SUB a2,v1,v2 ;a2=yu-yl
MUL a1,a2,a1 ;a1=al*(yu-yl)
ADD a1,a1,v2,LSL #14 ;a1=a1+yl*(1<<14)
MOV a1,a1,ASR #14 ;a1>>14
```

° TMS320C54x

```
;AR1=ap,AR2=al,AR3=yu,AR4=yl
LD #0x2000,A
SUB *AR1,A
BC L1,AGEQ
ST #0x4000,*AR5
B L2
L1:
ADD *AR1,1,B
ST BL,*AR5
L2:
LD *AR3,A
MPY *AR2,*AR3,A
MPY *AR2,*AR4,B
ADD B,A
SUB #0x4000,A
ADD A,14,B
```



## Comparison of implementations (ARM CPU v.s. TI DSP)

		TI DSP(Tms320c54x)		ARM(7TDMI)	
		C Code	Assembly Code	C Code	Assembly Code
ADPCM	Code size(b)	3.2k	3.1k	5k	2k
	Cycle/ Sample	12000	3200	4369	799
FIR	Code size*	22	16	60	60
	Cycle	5.06	1.11	6.09	5.13
FFT	Code size	7.1k	2.5k	11.8k	4.6k
	Cycle**	226134	15485	126883	45683
Viterbi	Code size	9.2k	5.4k	9.0k	14k
	Cycle/frame	338727	11845	220526	94298
DCT	Code size	1.5k	0.7k	1.3k	1k
	Cycle***	10656	4513	6607	4615
Synthesis (IIR)	Code size	136	64	248	136
	Cycle	420	55	520	140

\* kernel code size  
 \*\* 256 point complex FFT  
 \*\*\* 8\*8 matrix

ADPCM: very control intensive



## 5. Conclusion

1. DSP processors show better implementation performance compared to RISC processors, however the gap is becoming narrowed down due to the architectural improvements of RISC cpu's (dual cache, wide bandwidth data bus, hardware multiplier).

2. Many optimized DSP algorithms usually contains less number of multiplications, in this case, the RISC CPU based implementation can show better performances.

3. When the application program is very complex, the compiler based development is critical. In this case, the use of RISC based architectures can be more advantageous.

4. By changing the programming style, which may be the opposite of the DSP based methods, the performance of the RISC CPU can also be improved much.

For example: CELP decoder

- code size : 13K -> 7.5kbytes

- speedup : app. 10 times.



## Overview of My Talk

- Efficient Programming of DSP Algorithms on RISC CPU's
- Speaking Partner and Power Consumption Analysis



## SPEAKING PARTNER: AN ARM7-BASED MULTIMEDIA HANDHELD DEVICE

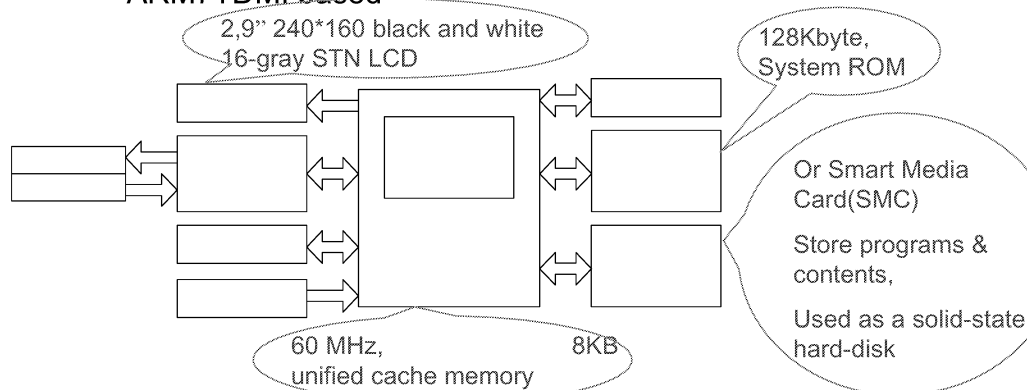


# Contents

- Introduction
- System overview
- Application implementations
  - MP3 audio decoder
  - Speech recognition
  - Image decompression(LZW)
- Power consumption reduction using DFS
- Conclusion
  
- Reference: 1. Hoseok Chang, Wonchul Lee and Wonyong Sung, "Optimization of Power Consumption for an ARM7-Based Multimedia Handheld Device," ACCEPTED at IEEE International Symposium on Circuits and Systems, Bangkok 2003.
- 2. Wonong Sung, Hoseok Chang, Wonchul Lee and Subong Ryu, "Speaking partner: an arm7-based multimedia handheld device," IEEE Workshop on Signal Processing Systems, pp. 218-221, October 2002.

# Introduction

- Speaking Partner
  - A handheld educational device for kids
  - MP3 audio decoding, speech recognition, Animation
  - ARM7TDMI based





## System overview

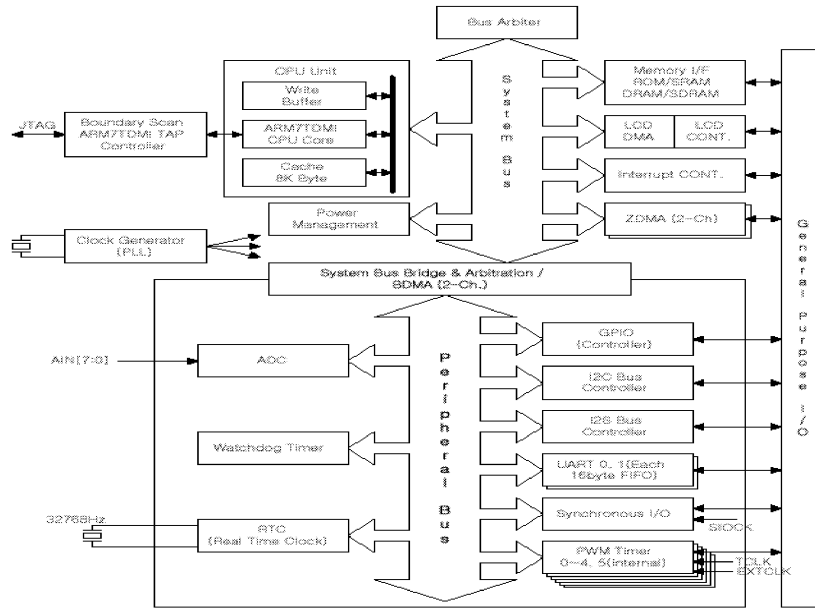
- CPU : S3C44B0X
  - ARM7TDMI Without MMU
  - 60MHz
  - 8 kByte unified cache
  - LCD controller, Timer, IIC, IIS, ADC ...
- Memory : 2 Mbyte DRAM, NOR Flash, NAND Flash
- Audio CODEC, USB controller, LCD, Key



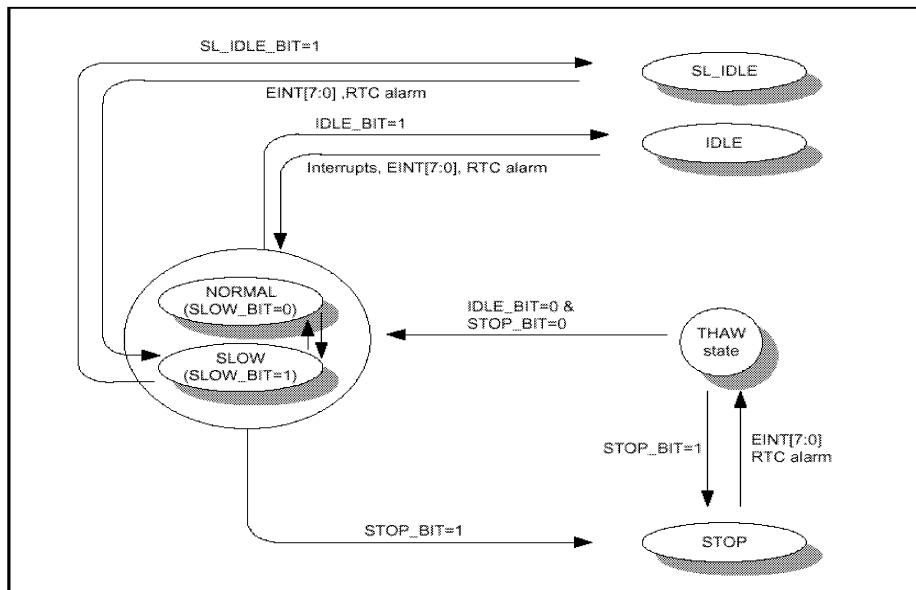
## S3C44B0X

- **Feature**
  - ASSP with ARM7TDMI by Samsung
  - 0.25um CMOS standard cells, and memory compiler
  - Bus architecture - SAMBAII ( Samsung ARM CPU embedded Microcontroller Bus Architecture.
  - System & Peripheral
    - SRAM(8K), Memory controller(FP/EDO/SDRAM/SRAM)
    - LCD, DMA(Memory, Peripheral), UART with IrDA
    - SIO, Timer with PWM, External DMA, Watch Dog Timer
    - Power control : Normal, Holding, Slow ,Idle, Thaw, and STOP
    - RTC, PLL

# Block Diagram



# Power Mode



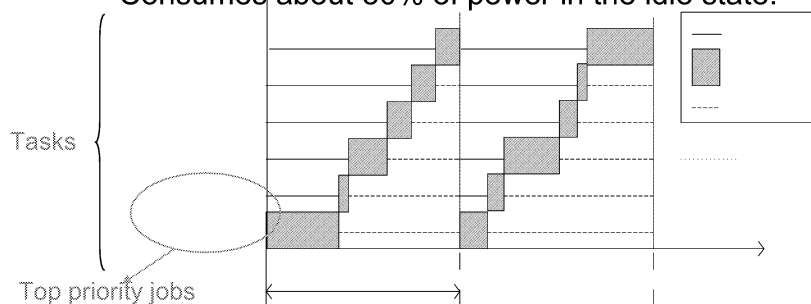
## Speaking partner OS

- Multi-tasking kernel
  - For hard real-time, task switching
- Audio output
  - CELP, MIDI, PCM/MP3
- GUI
  - 240X160, 16 gray-level, 2-layer color-keying
  - key input
- File system
  - Internal NAND flash, Smart media
  - SSFDC, long file name support
  - PC interface
  - USB



## Multi-tasking implementations

- Using a simple real-time operating system
  - Frame rate of 240 ms (can be changed in SW).
  - Conduct audio, graphic and system control tasks.
  - Be in the idle state when all the tasks for each frame are finished. -> power saving.
  - Consumes about 30% of power in the idle state.



## Software optimization

- ARM7TDMI
  - A fairly large number of registers(31)
    - Reduce the memory access, good compiler performance
  - Conditional execution
    - Reduce control overhead
  - 32bit barrel shifter
    - Execute shift/rotation with ALU operations
  - Block load/store transfer instructions
- Need to convert the floating-point version to fixed-point one
  - Using automatic scaling method
- Optimizations
  - Algorithm optimization
  - Loop transformation
  - Using ARM CPU specific features



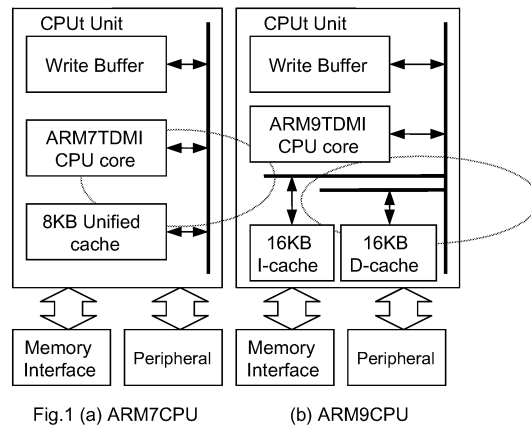
## Software optimization-continued

- ARM RISC Processors
  - No DSP specific architectural features
    - No full precision multiplier (only 32\*8 bits) – is disadvantageous for implementing computation intensive DSP algorithms
    - No block repeat, No address calculation units – need software optimization methods using RISC specific features
    - Has only small internal memory size (8KB cache for ARM7, 32 KB cache for ARM9) – need to reduce memory access and cache misses.



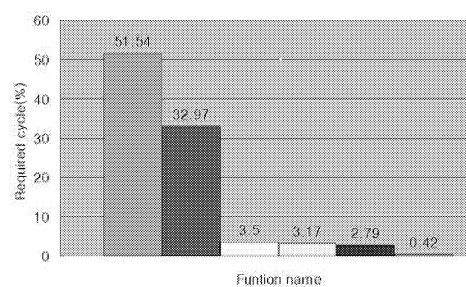
## ARM family architectures

- ARM7: unified cache 8KB
- ARM920: separate I, D cache 16 KB each



## Implementation of MP3 decoder

- MPEG 1/2 Layer-III(MP3) algorithm
  - Subband synthesis, IMDCT, Requantization
    - => 85% of processing time
  - Control intensive parts
    - Huffman decoding
    - Unpacking scale-factor
    - Good for conditional execution



## IMDCT and subband synthesis optimization

- Employed Britanak and Rao IMDCT algorithm
  - Small number of multiplications for N=36, 12
    - Good for ARM CPU since it doesn't have a full precision multiplier
- MPEG1/2 Audio standard
  - N=36 for long blocks, N=12 for short blocks

	N=36		N=12	
	Mul	Add	Mul	Add
ISO reference	648	630	72	66
Britanak & Rao	47	165	13	39

< Iso reference vs. Britanak and Rao's algorithm >



## Use of 32\*16 and 32\*32 multiplications

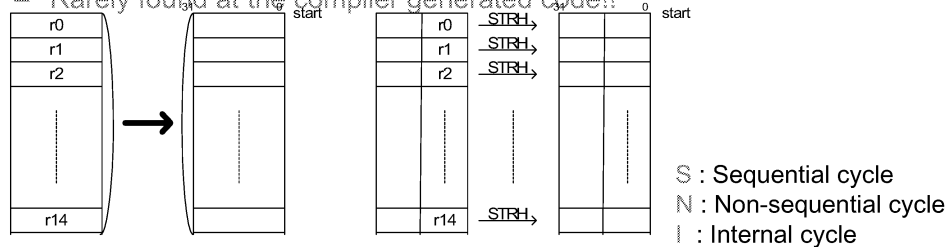
- 32\*32 multiplications – 6 cycles
- 32\*16 multiplications – 4 cycles
  
- Overall: 32\*16 multiplications is 9.4% faster
- But the quality: 32\*32 mul: 91 dB  
32\*16 mul: 84 dB  
(acceptable, but not desired)



## Assembly language optimization

- Using block transfer instructions, LDM, STM

– Rarely found at the compiler generated code!!



14S + 2N cycles (Store)

15S + 1N + 1I cycles (Load)

2N \* 15 cycles (Store)

(1S + 1N + 1I) \* 15 cycles (Load)

- Accessing the internal memory and cache
  - N, S, I = 1 cycle
- Accessing for external DRAM access
  - N >> 1 cycle



## Instructions vs. Clock cycles

- Optimization?
  - Using block transfer instructions, LDM, STM

	IMDCT		Subband	
	Ins.	Cycles	Ins.	Cycles
Before Opt.	134144	275840	345580	790986
After Opt.	97024	217472	227328	515484

- IMDCT
  - Instructions : 28 % decreased
  - Clock cycles : 21 % decreased
- Subband
  - Instructions : 34 % decreased
  - Clock cycles : 35 % decreased



## Memory access results

- Cache miss effects using block transfer instructions (LDM, STM)
- Cache environments
  - ARM7 CPU core, No write allocation
  - 8KB unified cache, 64 way set-associative
- Instruction vs. Data cache miss

	I demand	I miss	D demand	D miss
Before Opt.	6021044	145704	2135815	289573
After Opt.	4766283	84082	1972899	266959

1.76% miss ratio

13.5% miss ratio

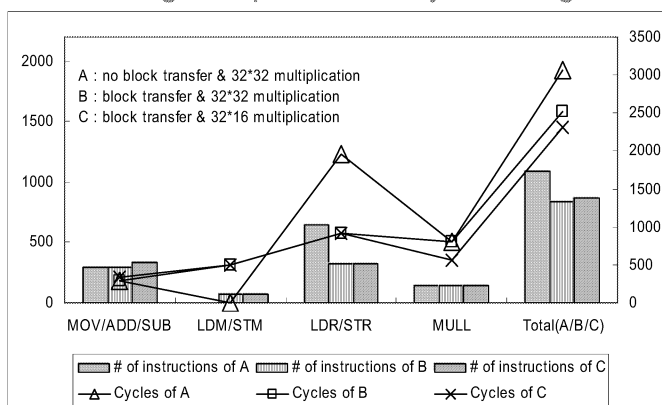
- Required off-chip memory access for a second

	Demand Fetches (From Mem)	Demand Writes (To Mem)	Total R/W Memory
Before Opt.	8.9MB	3.50MB	12.48MB
After Opt.	6.5MB	3.41MB	9.92MB



## IMDCT

- Reducing multiplier accuracy vs. using block transfer



SNR

32\*32 bits : 91.61dB

32\*16 bits : 82.45dB

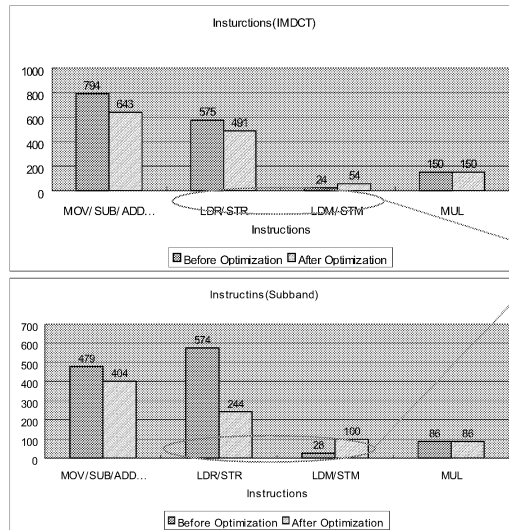
- A -> B : 17.8% The effect of reducing the data memory
- B -> C : 8.1% > the effect of reducing the precision for multiply





## Instruction types

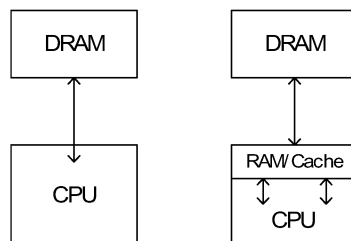
- Subband part is more efficient using block transfer instructions



Load/Store reduction !!  
 → Reducing Memory access operand !!

## ARM7 vs ARM9

- ARM7 architecture based implementation
  - Unified Cache
- ARM9 architecture based implementation
  - Separated Cache
  - Cache simulator
    - DinerolV



## ARM7 architecture-based implementation

- ARM7
  - 8KByte Unified Cache
  - Performance degradation due to cache miss can be significant
  - No support write allocation

	I demand	I miss	D demand	D miss
Before Opt.	6021044	145704	2135815	289573
After Opt.	4766283	84082	1972899	266959

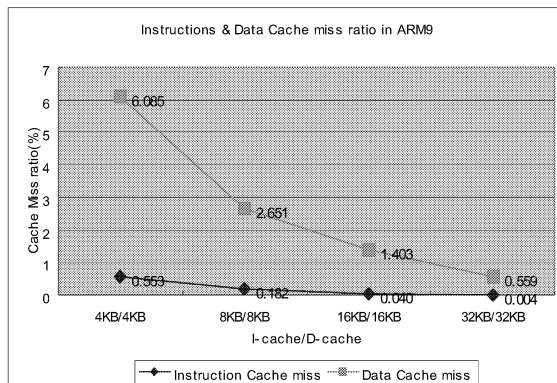
- Improve the spatial locality of data which reduces the miss ratio with block transfer instructions
- => reduce the # of accesses to/from the external DRAM

	Demand Fetches (From Mem)	Demand Writes (To Mem)	Total R/W Memory
Before Opt.	8.95MB	3.50MB	12.48MB
After Opt.	6.5MB	3.41MB	9.92MB



## ARM9 architecture-based implementation

- ARM9
  - Separate 16KB Instruction and 16KB Data cache
  - Support write allocation



ARM7(Unified 8KB)  
 Instruction cache miss : 13.5%  
 Data cache miss : 1.7%



## Miss penalty

- Miss penalty between the main processor and the external SDRAM
- SDRAM model
  - 8 clock cycles of latency for the first word read
  - 5 clock cycles for the first word write
  - 1 clock cycles for successive memory read and write
  - Assume
    - SDRAM clock freq.  $\approx$  CPU clock freq.
  - ARM7 with 8KB: 40% more clock cycles with 16 bit SDRAM  
30% more clock cycles with 32 bit SDRAM
  - ARM9 with 16KBI, 16KBD, 4% more clock cycles with 32 bit SDRAM



## Performance

- Tested using the ISO reference standard, and proved by using 10 popular pop-songs
  - With ARM7TDMI@60MHz
  - Mono, stereo, joint stereo
  - Sampling freq. : 44.1kHz, 22.05kHz
  - Bit rate : 32kbps ~ 192kbps
  - Average of 94.24dB SNR
  - Average of 16.5 MIPS



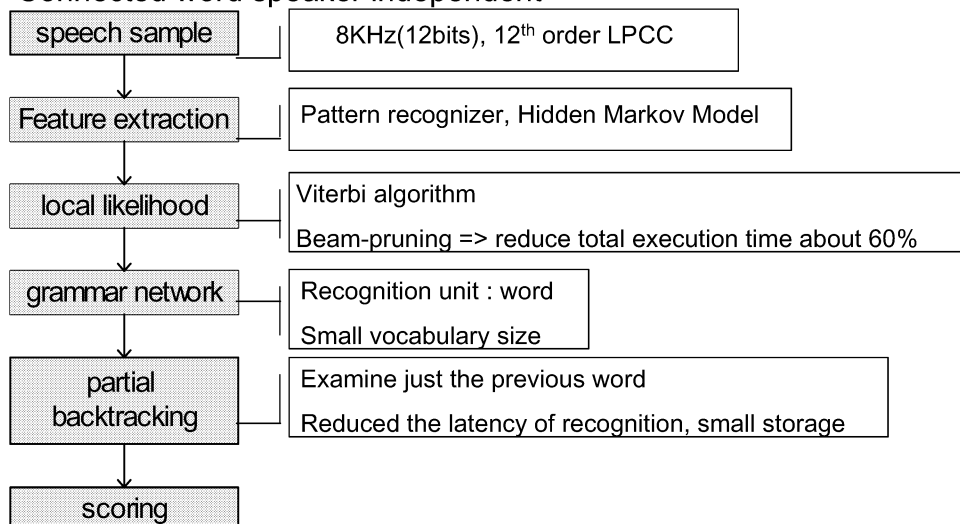
## Performance of MP3 implementations

- Implementation of MPEG1/2 Layer-III decoding algorithm using ARM7 and ARM9 based systems
- Optimization Summary
  - Multiply resolution reduction (32\*16): 9.4% faster
  - Block load & store instructions: 17.8% faster
  - Cache miss effects: ARM7: upto 40% slower  
ARM9: upto 4 % slower
- The overhead of data-transfer should be considered very seriously for real-time and low-power implementation



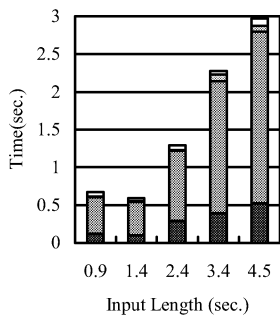
## Implementation of speech recognition

- Connected word speaker independent

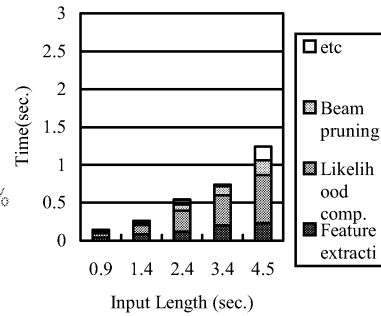


## Profiling results

- Using software optimization methods
  - Loop transformations
  - Post increment/decrement conversion
  - 16 bit multiplications



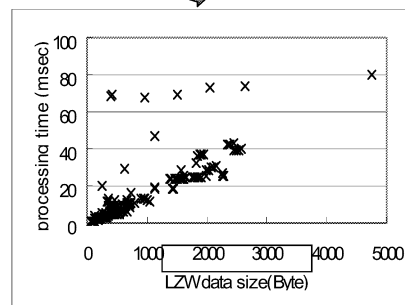
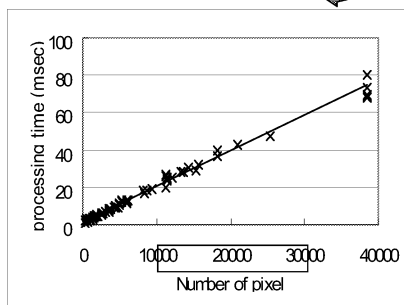
Execution time  
=> Reduced about 30 ~ 45%



- Test
  - 215 word-set
  - Over 90% recognition rate

## Implementation of animation

- Using Lempel-Ziv-Welch(LZW) image decompression algorithm
- Decompression time
  - Proportional to the image size, not the compressed data size.



## Power consumption

- The system is operating using two AA-size 1.5V batteries
  - Most digital & analog circuits : 3.3V
  - CPU core : 2.5V, LCD : 21V, Audio amp : 150mW

Activity	CPU Activity	LCD	Audio amp.	Current
Menu display	About 10%	On	Off	90mA
Song with animation	About 90%	On	On	250mA
Speech recognition	About 100%	On	Off	160mA
Sleep	About 0%	Off	Off	20mA

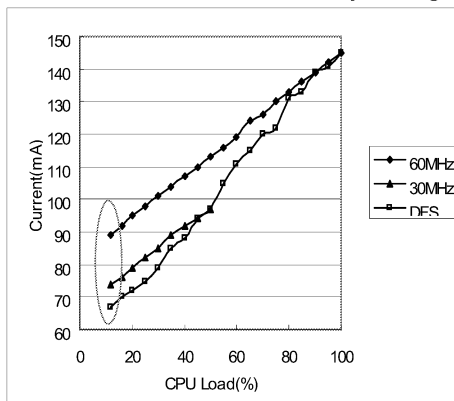
Current consumption according to the activity

H/W block	Current	Comment
CPU	70mA	100% running
DRAM	60mA	Running MP3 decoder
LCD display	15mA	Includes frame buffer access
Audio	90mA	Peak Value
Others	20mA	LCD bias current 20mA

Current consumption at each H/W block

## Power optimization using DFS

- DFS: reduce the clock frequency to minimize the idle period.
- Current measured
  - With LZW compression
    - Except for speaker drive and back light
  - CPU load is controlled by changing the size and the # of images to



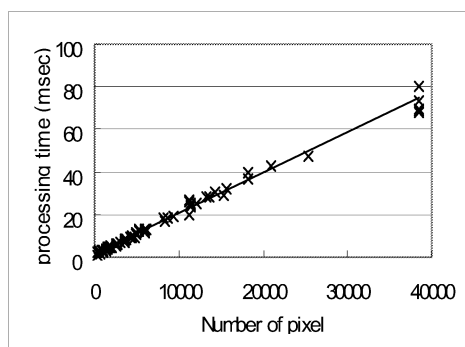
H/W block	Current	
	Constant freq.	DFS
CPU	34mA	11mA(-67%)
DRAM	29mA	29mA
LCD display	15mA	15mA
Others	11mA	11mA
Total	89mA	66mA (-26%)

CPU load : 10%

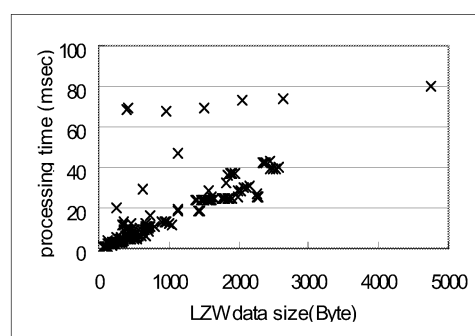
## CPU load estimation for DFS

- CPU load for executing is need for determining the optimum clock frequency.
  - LZW, MP3, CELP based speech decoding, speech recognition
- The load for MP3
  - Dependent on the bit rate and sampling clock frequency
  - 60MHz clock
    - 10% for 56kbps, 22.05 KHz
    - 9.6% for 32kbps, 22.05 KHz
    - 7% for 32 kbps, 16kHz
- The load for CELP decoding
  - Almost constant and is 18% of the 60MHz CPU load
- The load for LZW
  - Varying in each frame

## LZW Image



Processing time of LZW according to the number of pixels



Processing time of LZW according to the compressed data size

LZW decompression time is proportional to image size, not the compressed data size

## Execution time prediction for DFS

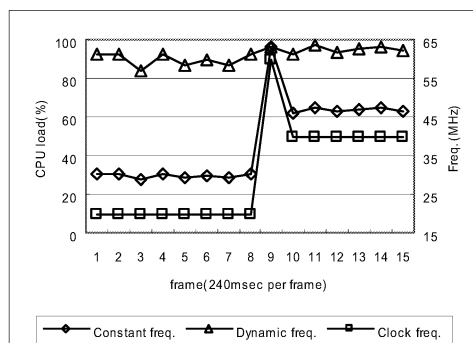
- LZW decompression
  - 15 msec of overhead is needed for updating each frame of image
- ADPCM encoding
  - Includes the CPU load for drawing speech waveforms on the LCD screen
- Speech recognition

S/W component	Execution time at 60MHz(ms)
LZW decompression	$\frac{\text{number of pixel}}{800} \times 1.55 + 15$
MP3 decoding	27.5
G.729 decoding	42.5
ADPCM encoding	56.3
ADPCM decoding	2.5
Margin	10



## CPU Load

- Constant frequency and dynamic frequency
  - CPU displays animation while playing MP3 sound.



Activity	CPU load	Original current	Optimized current
Menu Display	11 %	92 mA	68 mA
Song with animation	65 %	175 mA	160 mA
Speech Recognition	100 %	150 mA	150 mA
MP3 play	10 %	125 mA	100 mA

3.1 v : Most circuits  
 2.5 v : CPU core,  
 2.1 v : LCD

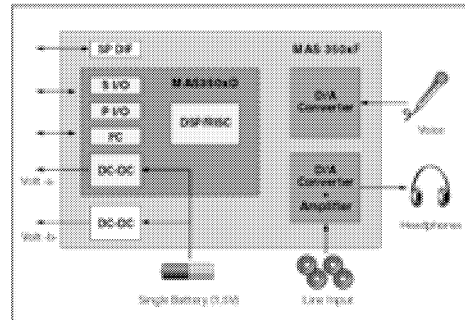
– The average current consumption in the system level is reduced by 30%





## Design review (1)

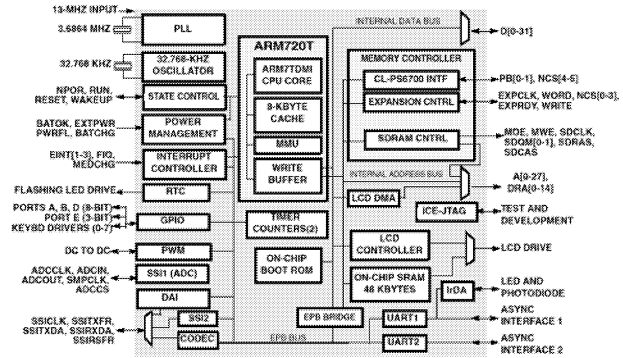
- First version of the SpeakingPartner employed Micronas MP3 chip: ARM7+DSP architecture
  - MAS3509 contains DSP for MP3 decoding/encoding, CELP and so on. -> we can expect lower power consumption.
  - MAS3509 contains ADC and DAC codec
  - MAS3509 contains two DC-DC converters
- Design results
  - We need to use a separate DC-DC converter due to the short capacity of MAS3509 DC-DC converters (total 3 DC-DC converters besides one (20volt) for LCD (large PCB area))
  - The power consumption reduction was not remarkable (almost same) because we applied 2.5 volt to MAS3509 (for CELP).
  - We still use the RISC CPU for speech recognition, which is the most intensive DSP job. (lack of programmability for MAS3509).
  - Encountered many software problems, which were eventually solved but had to ask Micronas frequently. (e.g. missed very first part of audio). There are still infrequent communication missing between the two CPU's. (IIC)



	SP(Micronas)	SP(no Micronas)	Bat terminal에 3.0V
Menu 상태	86	95	
노래	160	180	At the beginning of the song
STOP	39	18	
Idle	87	63	
LCD off	70	74	
30MHz Idle	65	60	
PLL off	58	51	
CPU 100%	146	140	

## Design review (2)

- Internal memory (Y/N)
- Samsung's ARM7 chip does not have internal memory (but 8KB caches), but Cirrus EP7312 contains 48KB of internal memory
  - Good for power reduction
  - But seems too large size unless for DRAM less systems



## Design review (3)

- The idle system current consumption is important, but very hard to reduce
  - About 10 mA even after optimization
  - Power consumption in the CPU stop mode seems about 8 mA
  - DC-DC converter inefficiency, ...
  - With 1400mAH batteries -> 140hr -> less than one week
    - There is an on-off switch, but sometimes it is left with switch-on.



## Design review (4)

- Can expect lower power consumption with ARM9?  
(similar data-path, higher clock frequency with lower voltage 2.5 v-> 1.8 v, improved cache – 16KB I,D, write allocate)
  - Much less power consumption in DRAM due to cache misses
    - But how about bigger applications (video)?
  - Much reduced CPU power consumption
    - But how about demanding applications (video and graphics)?
  - Much improved stop mode current consumption  
(longer hold time in stop mode)
  - But, same analog current consumption (can be improved by employing D-class amp)
  - Same back light power consumption (50mA, maybe more for color LCD)
  - LCD bias (20mA for STN)
  - LCD frame buffer (13mA for 240\*160, 4bit -> 4 times for 320\*240\*8bit)
    - this would be solved by using the LCD containing the frame buffer inside.



## Conclusion

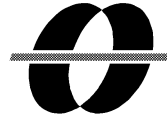
- MP3 decoding, speech recognition, animation
  - Implemented in software using ARM7TDMI
- Lacks of features in programmable digital signal processors
  - Need software optimization techniques
- The effect of reducing the data transfer  
> the effect of reducing the precision for multiply
- Power analysis
  - Current consumed at the DRAM
  - ≈ Current consumed at the CPU core

=> reducing cache miss is most important for lowering power consumption (this would be possible by the next generation cpu.)
- Current consumption in frame buffer would be significant for large size color display, which can't be reduced by employing a better CPU.
- CPU clock frequency scaling
  - Load estimated by the real-time OS according to the profiling results
- Current can be further reduced by voltage scaling
  - With dynamic voltage scaling such as Intel's Xscale



Sponsored by:





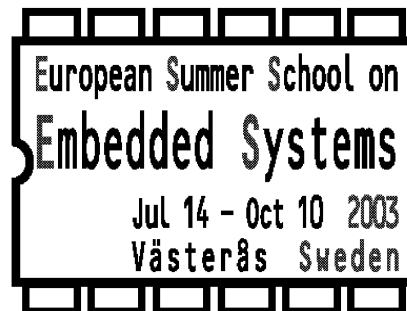
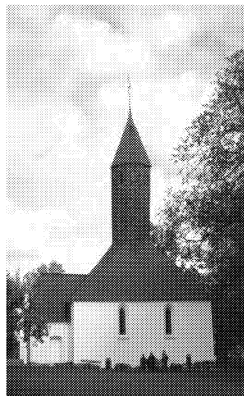
MÄLARDALENS HÖGSKOLA

# ESSES 2003

## European Summer School on Embedded Systems

### Lecture Notes Part VII

### Low Power Systems: Low-power System Modeling and I/O Systems



Editors: Ylva Boivie, Hans Hansson, Jane Kim, Sang Lyul Min

Västerås, July 28-August 1, 2003

ISSN 1404-3041

ISRN MDH-MRTC-105/2003-1-SE

# MRTC

MÄLARDALEN REAL-TIME  
RESEARCH CENTRE

[www.mrtc.mdh.se](http://www.mrtc.mdh.se)

# **Designing Energy Aware Systems**

**Vijaykrishnan Narayanan**

Microsystems Design Lab  
Pennsylvania State University, USA

*ESSES 2003*

**Designing Energy Aware Systems**


Vijaykrishnan Narayanan  
Microsystems Design Lab  
Pennsylvania State University

Acknowledgments: Profs. Irwin and Kandemir;  
MDL Graduate Students

<http://www.cse.psu.edu/~mdl>




*Part 1: Energy Estimation  
Background and Tools*



*Outline - Part 1*

- ◆ Power Metrics
- ◆ Energy Estimation Tools


3 ESSES 2003 Vijaykrishnan Narayanan

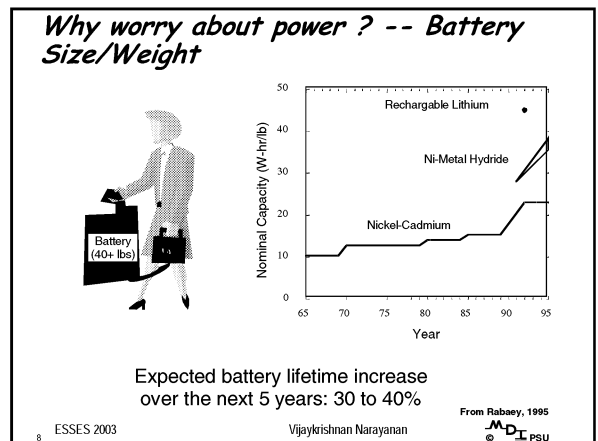
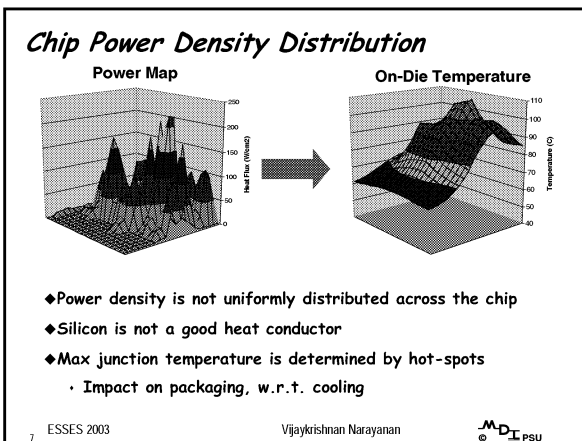
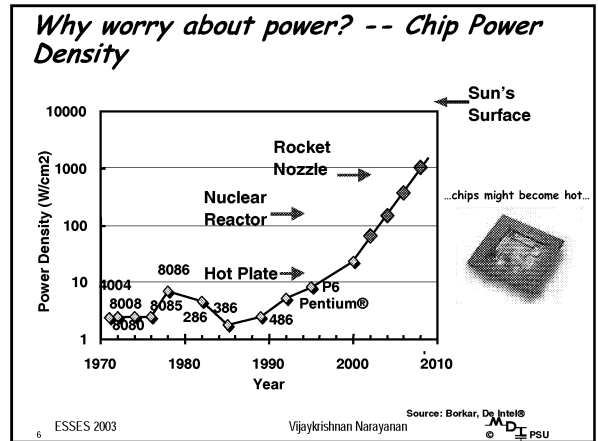
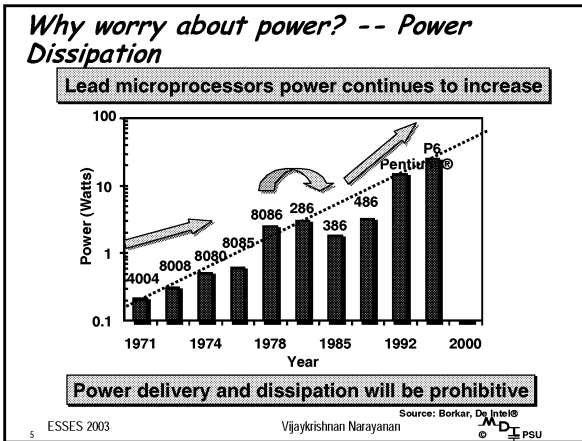


*Why Power Matters?*

- ◆ Packaging costs
- ◆ Power supply rail design
- ◆ Chip and system cooling costs
- ◆ Noise immunity and system reliability
- ◆ Battery life (in portable systems)
- ◆ Environmental concerns
  - Office equipment accounted for 5% of total US commercial energy usage in 1993
  - *Energy Star* compliant systems

4 ESSES 2003 Vijaykrishnan Narayanan







### Why worry about power? -- Standby Power

Year	2002	2005	2008	2011	2014
Power supply $V_{dd}$ (V)	1.5	1.2	0.9	0.7	0.6
Threshold $V_T$ (V)	0.4	0.4	0.35	0.3	0.25

□ Drain leakage will increase as  $V_T$  decreases to maintain noise margins and meet frequency demands, leading to excessive battery draining standby power consumption.

Source: Borkar, De Intel®  
Vijaykrishnan Narayanan

### Power and Energy Figures of Merit

- ◆ Power consumption in Watts
  - determines battery life in hours
- ◆ Peak power
  - determines power ground wiring designs
  - sets packaging limits
  - impacts signal noise margin and reliability analysis
- ◆ Energy efficiency in Joules
  - rate at which power is consumed over time
- ◆ Energy = power \* delay
  - Joules = Watts \* seconds
  - lower energy number means less power to perform a computation at the same frequency

ESSES 2003 Vijaykrishnan Narayanan

### Power versus Energy

Power is height of curve  
Lower power design could simply be slower

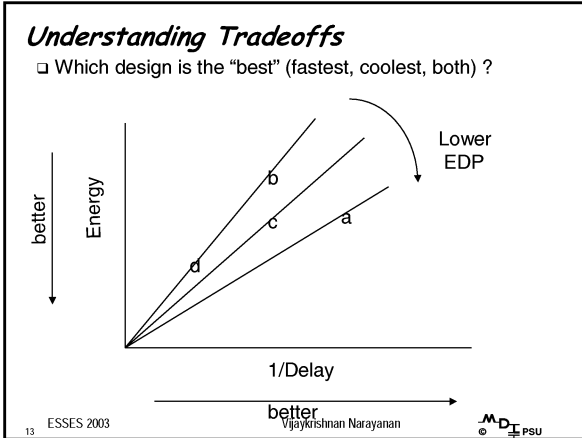
Energy is area under curve  
Two approaches require the same energy

ESSES 2003 Vijaykrishnan Narayanan

### PDP and EDP

- ◆ Power-delay product (PDP) =  $P_{av} * t_p = (C_L V_{DD}^2)/2$ 
  - PDP is the average energy consumed per switching event (Watts \* sec = Joule)
  - lower power design could simply be a slower design
- Energy-delay product (EDP) =  $PDP * t_p = P_{av} * t_p^2$ 
  - EDP is the average energy consumed multiplied by the computation time required
  - takes into account that one can trade increased delay for lower energy/operation (e.g., via supply voltage scaling that increases delay, but decreases energy consumption)

ESSES 2003 Vijaykrishnan Narayanan



### CMOS Energy & Power Equations

$$E = C_L V_{DD}^2 P_{0 \rightarrow 1} + t_{sc} V_{DD} I_{peak} P_{0 \rightarrow 1} + V_{DD} I_{leakage}$$

$$f_{0 \rightarrow 1} = P_{0 \rightarrow 1} * f_{clock}$$

$$P = C_L V_{DD}^2 f_{0 \rightarrow 1} + t_{sc} V_{DD} I_{peak} f_{0 \rightarrow 1} + V_{DD} I_{leakage}$$

Dynamic power
Short-circuit power
Leakage power

ESSES 2003  
Vijaykrishnan Narayanan  
M.D. PSU

### Dynamic Power Consumption

Energy/transition =  $C_L * V_{DD}^2 * P_{0 \rightarrow 1}$

$P_{dyn} = \text{Energy/transition} * f = C_L * V_{DD}^2 * P_{0 \rightarrow 1} * f$

$P_{dyn} = C_{EFF} * V_{DD}^2 * f$  where  $C_{EFF} = P_{0 \rightarrow 1} * C_L$

Data dependent - a function of switching activity!

ESSES 2003  
Vijaykrishnan Narayanan  
M.D. PSU

### Lowering Dynamic Power

Capacitance: Function of fan-out, wire length, transistor sizes

Supply Voltage: Has been dropping with successive generations

Activity factor: How often, on average, do wires switch?

Clock frequency: Increasing...

$$P_{dyn} = C_L V_{DD}^2 P_{0 \rightarrow 1} f$$

ESSES 2003  
Vijaykrishnan Narayanan  
M.D. PSU

### Short Circuit Power Consumption

Finite slope of the input signal causes a direct current path between  $V_{DD}$  and GND for a short period of time during switching when both the NMOS and PMOS transistors are conducting.

17 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### Short Circuit Currents Determinates

$$E_{sc} = t_{sc} V_{DD} I_{peak} P_{0 \rightarrow 1}$$

$$P_{sc} = t_{sc} V_{DD} I_{peak} f_{0 \rightarrow 1}$$

- ◆ Duration and slope of the input signal,  $t_{sc}$
- ◆  $I_{peak}$  determined by
  - the saturation current of the P and N transistors which depend on their sizes, process technology, temperature, etc.
  - strong function of the ratio between input and output slopes
    - ◆ a function of  $C_L$

18 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### Impact of $C_L$ on $P_{sc}$

Large capacitive load: Output fall time significantly larger than input rise time.

Small capacitive load: Output fall time substantially smaller than the input rise time.

19 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

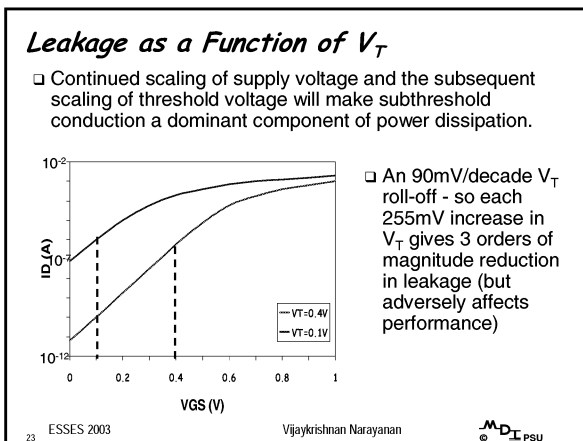
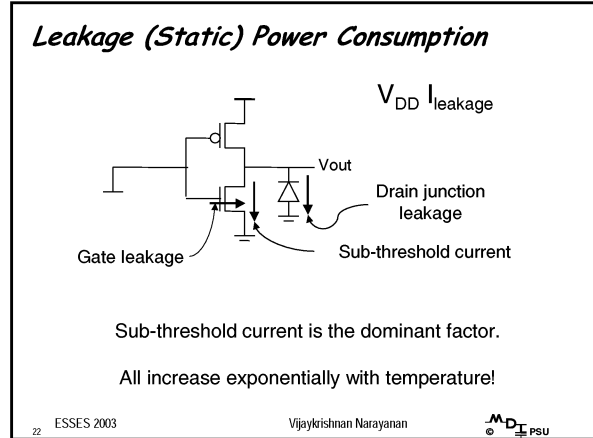
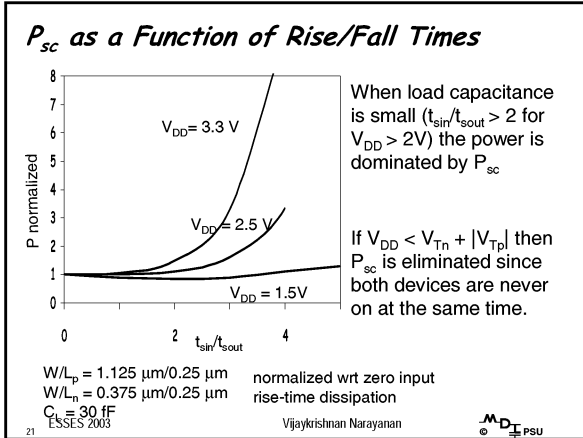
### $I_{peak}$ as a Function of $C_L$

When load capacitance is small,  $I_{peak}$  is large.

Short circuit dissipation is minimized by matching the rise/fall times of the input and output signals - slope engineering.

500 psec input slope

20 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

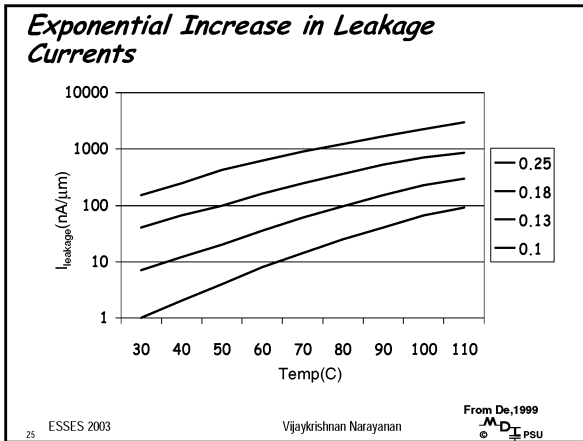


### TSMC Processes Leakage and $V_T$

	CL018 G	CL018 LP	CL018 ULP	CL018 HS	CL015 HS	CL013 HS
$V_{dd}$	1.8 V	1.8 V	1.8 V	2 V	1.5 V	1.2 V
$T_{ox}$ (effective)	42 Å	42 Å	42 Å	42 Å	29 Å	24 Å
$L_{gate}$	0.16 $\mu\text{m}$	0.16 $\mu\text{m}$	0.18 $\mu\text{m}$	0.13 $\mu\text{m}$	0.11 $\mu\text{m}$	0.08 $\mu\text{m}$
$I_{dsat}$ (n/p)	600/260	500/180	320/130	780/360	860/370	920/400
$I_{off}$ (leakage) ( $\mu\text{A}/\mu\text{m}$ )	20	1.60	0.15	300	1,800	13,000
$V_{Tn}$	0.42 V	0.63 V	0.73 V	0.40 V	0.29 V	0.25 V
FET Perf. (GHz)	30	22	14	43	52	80

From MPR, 2000

ESSES 2003 Vijaykrishnan Narayanan © M D I PSU



### Review: Energy & Power Equations

$$E = C_L V_{DD}^2 P_{0 \rightarrow 1} + t_{sc} V_{DD} I_{peak} P_{0 \rightarrow 1} + V_{DD} I_{leakage}$$

$$P = C_L V_{DD}^2 f_{0 \rightarrow 1} + t_{sc} V_{DD} I_{peak} f_{0 \rightarrow 1} + V_{DD} I_{leakage}$$

$f_{0 \rightarrow 1} = P_{0 \rightarrow 1} * f_{clock}$

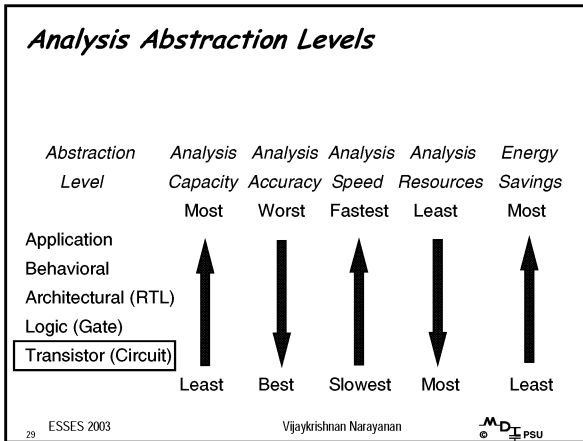
Dynamic power (~90% today and decreasing relatively)	Short-circuit power (~8% today and decreasing absolutely)	Leakage power (~2% today and increasing)
---	--	---

ESSES 2003 Vijaykrishnan Narayanan

## Energy Estimation Tools

M D E PSU

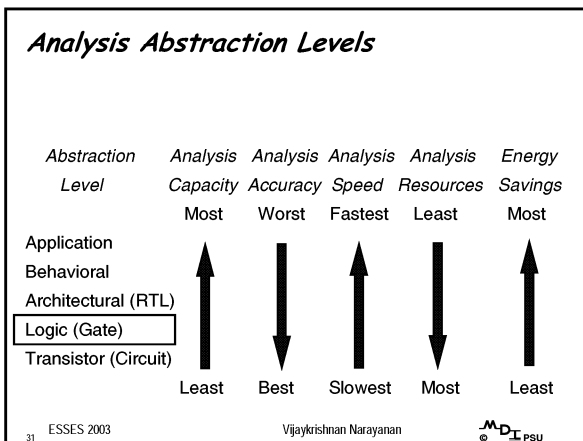
- ### Energy Analysis Techniques
- ◆ Simulation based techniques
    - characterization - using lower level analysis results to construct higher level models
    - very computational intensive; cycle accurate
  - ◆ Probabilistic based techniques
    - signals viewed as random zero-one processes with certain statistical characteristics
    - computationally efficient; average values for sequences of cycles
    - large errors are possible if statistical assumptions are incorrect
- ESSES 2003 Vijaykrishnan Narayanan



### Circuit Level Simulation

- ◆ Extract circuit netlist description from layout
  - captures internal (diffusion) and external (wiring and gate fanout) capacitances
- ◆ Run an analog simulation
  - characterization of device models (nfets, pfets)
  - solution of large system of equations so very computationally intensive (< few thousand transistors)
  - can accurately estimate (within a few %) dynamic and leakage power dissipation
- ◆ HSPICE (Avant!), spectre (Cadence), PowerMill (Synopsys)

ESSES 2003 Vijaykrishnan Narayanan © M.D. PSU



### Medium-level models: RTL

- ◆ Logic simulation obtains switching events for every signal
- ◆ Structural VHDL or verilog with zero or unit-delay timing models
- ◆ Capacitance estimates performed
  - Device Capacitance
    - ◆ Gate sizing estimates performed, similar to synthesis
  - Wiring Capacitance
    - ◆ Wire load estimates performed, similar to placement and routing
- ◆ Switching event and capacitance estimates provide dynamic power estimates

ESSES 2003 Vijaykrishnan Narayanan © M.D. PSU

### Gate Level Simulation

- ◆ Perform logic simulation to obtain the switching events for each net (signal)
  - logic description in structural VHDL or verilog
  - zero-delay or unit-delay timing models
- ◆ Determine frequency of each net  $f_y = t_y / (2T)$ , where  $t_y$  is the number of logic switches of net  $y$  and  $T$  is the simulation time, to compute *dynamic* power

$$P_{dyn} = \sum C_y V_{DD}^2 f_y$$

- ◆ Pre-layout so must estimate  $C_y$

ESSES 2003 Vijaykrishnan Narayanan © M.D.T. PSU

### Capacitance Estimation

- ◆ Device (diffusion and gate) capacitance
  - depends on width/length of driving gate's source/drain diffusion and fanout gates
  - part of characterization of cell based designs
- ◆ Wiring capacitance
  - depends on placement and routing
  - wire load - predict wire length of a net from the number of pins incident to the net
    - ◆ mapping table can be constructed from historical data of existing designs

ESSES 2003 Vijaykrishnan Narayanan © M.D.T. PSU

### Analysis Abstraction Levels

Abstraction Level	Analysis Capacity	Analysis Accuracy	Analysis Speed	Analysis Resources	Energy Savings
Application	Most	Worst	Fastest	Least	Most
Behavioral	↑	↓	↑	↓	↑
Architectural (RTL)	↑	↓	↑	↓	↑
Logic (Gate)	↑	↓	↑	↓	↑
Transistor (Circuit)	Least	Best	Slowest	Most	Least

ESSES 2003 Vijaykrishnan Narayanan © M.D.T. PSU


### Power Modeling at Architecture Level

- ◆ Previously most academic research has either:
  - Calculated power within individual units: ie cache
  - Calculated abstract metrics instead of power
    - ◆ eg "needless speculative work saved per pipe stage"
- ◆ What is needed now?
  - A single common power metric for comparing different techniques
  - Reasonable accuracy
  - Flexible/modular enough to explore a design space
  - Fast enough to simulate real benchmarks
  - Facilitate early experiments: before HDL or circuits...

ESSES 2003 Vijaykrishnan Narayanan © M.D.T. PSU


**Architecture level models**

- ◆ **Examples:**
  - SimplePower, Tempest, Wattch, PowerTimer, Softwatt...
- ◆ **Components of a "good" Arch. Level power model**
  - Capacitance model
  - Circuit design styles
  - Clock gating styles & Unit usage statistics
  - Signal transition statistics

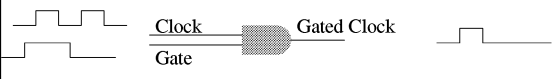
37 ESSES 2003 Vijaykrishnan Narayanan 

**Accounting for Different Circuit Design Styles**


- ◆ RTL and Architectural level power estimation requires the tool/user to perform circuit design style assumptions
  - Static vs. Dynamic logic
  - Single vs. Double-ended bitlines in register files/caches
  - Sense Amp designs
  - Transistor and buffer sizings
- ◆ Generic solutions are difficult because many styles are popular
- ◆ Within individual companies, circuit design styles may be fixed

38 ESSES 2003 Vijaykrishnan Narayanan 

**Clock Gating: What, why, when?**




- ◆ Dynamic Power is dissipated on clock transitions
- ◆ Gating off clock lines when they are unneeded reduces activity factor
- ◆ But putting extra gate delays into clock lines increases clock skew
- ◆ End results:
  - Clock gating complicates design analysis but saves power. Used in cases where power is crucial.

39 ESSES 2003 Vijaykrishnan Narayanan 

**Signal Transition Statistics**

- ◆ Dynamic power is proportional to switching
- ◆ How to collect signal transition statistics in architectural-level simulation?
  - Many signals are available, but do we want to use all of them?
  - One solution (register file):
    - ◆ Collect statistics on the important ones (bitlines)
    - ◆ Infer where possible (wordlines)
    - ◆ Assign probabilities for less important ones (decoders)
  - Use Controllability and Observability notions from testing community?

40 ESSES 2003 Vijaykrishnan Narayanan 



**SimplePower:  
Architectural Level Simulation**

- ◆ Perform RTL simulation to obtain the input activity for each major functional unit
  - architectural description in behavioral VHDL or verilog or C, C++
- ◆ Energy characterization of functional units
  - analytical energy models
    - ◆ caches, DRAMs
  - macro energy models
    - ◆ system buses
    - ◆ ALUs, register file, pipeline registers

41 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

**Analytical Energy Model Example**

- ◆ On-chip cache

Energy = Ebus + Ecell + Epad + Emain

...

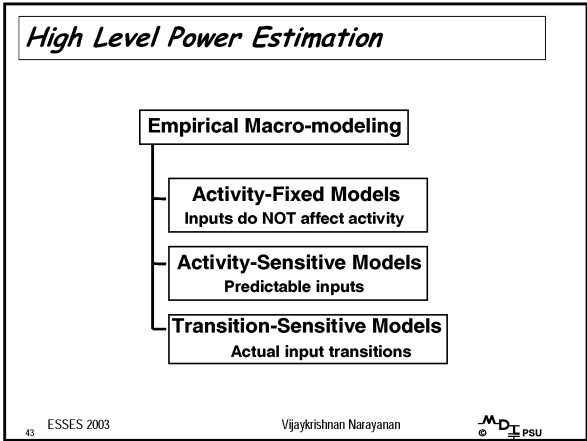
$$E_{cell} = \beta * (w_{l\_length}) * (b_{l\_length} + 4.8) * (N_{hit} + 2 * N_{miss})$$

$$w_{l\_length} = m * (T + 8L + St)$$

$$b_{l\_length} = C / (m * L)$$

Nhit = number of hits; Nmiss = number of misses; C = cache size; L = cache line size in bytes; m = set associativity; T = tag size in bits; St = # of status bits per line;  $\beta = 1.44e-14$  (technology parameter)

42 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU



**Transition Sensitive Energy Model**

- ◆ Must first design and layout a functional unit and then simulate it to capture switch capacitances
  - bit independent - bus lines, pipeline registers
    - ◆ one bit switching does not affect other bit slices' operations
  - bit dependent - ALU, MAC, decoders
    - ◆ one bit switching does affect other bit slices' operations
- ◆ Once constructed, the models can be reused in simulations of other architectures built with the same technology

44 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### Switch Capacitance Table

Previous Input Vector	Current Input Vector	Switch Capacitance
0 . . . 0 0	0 . . . 0 0	cap <sub>0→0</sub>
0 . . . 0 0	0 . . . 0 1	cap <sub>0→1</sub>
...	...	...
1 . . . 1 1	1 . . . 1 1	cap <sub>2<sup>n</sup>-1→2<sup>n</sup>-1</sub>

45 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

### Table Compression

◆ **Problem**

- Results in large uncompressed table (e.g., 16-bit adder → 2<sup>32</sup> rows)
- Excessive simulation (e.g., 2<sup>32</sup> !)

◆ **Solution**

- Clustering Algorithm  
Reference: Huzefa Mehta, et al "Module Energy Characterization using Clustering", DAC'96
- For 16-bit adder, to keep 12% average error → 1000 simulation points, 97 rows

46 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

### 2:1 Multiplexer Table

Uncompressed (64 rows)		Compressed (32 rows)		Reduced (11 rows)	
000 000	0.00	000 0xx	0.00	000 0xx	0.00
000 001	0.00	000 100	0.04	000 1xx	0.045
000 010	0.00	000 101	0.05	001 0xx	0.00
000 011	0.00	000 110	0.04	...	...
000 100	0.04	000 111	0.05	...	...
000 101	0.05	001 0xx	0.00	...	...
000 110	0.04	...	...	...	...
000 111	0.05	...	...	...	...
001 000	0.00	...	...	...	...
001 001	0.00	...	...	...	...
...	...	...	...	...	...

47 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

### Register File Example

Bit Dependent Cells

Write Decoder	5:32
Write Decoder	5:32
Read Decoder	5:32
Read Decoder	5:32
Read Decoder	5:32
Read Decoder	5:32

Bit Independent Cells

Write Data Drivers
32 x 32 Cell Array
Read Sense Amps

Word Line Drivers

48 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

### Decoder Characterization

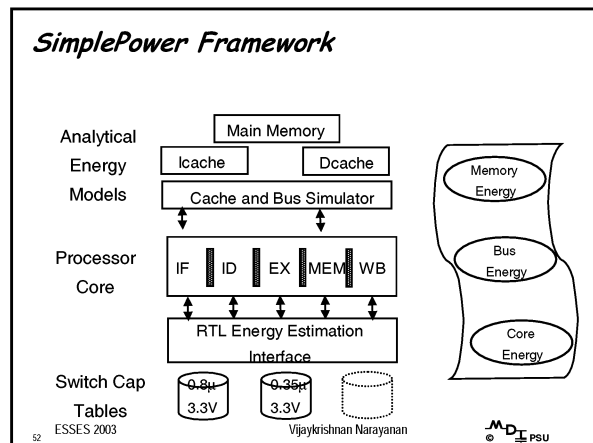
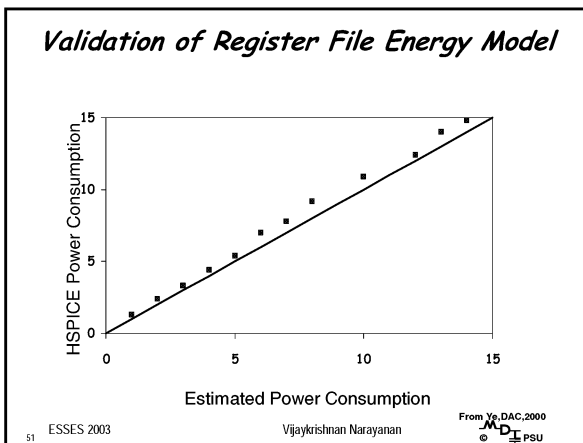
- ◆ A 5:32 decoder →  $2^{10}$  row table!
- ◆ Build 5:32 decoder out of smaller decoders

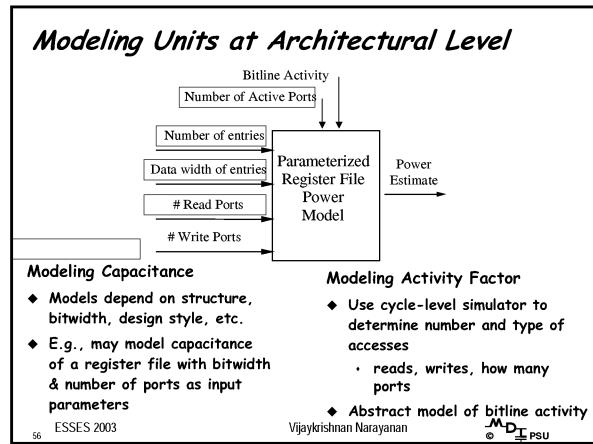
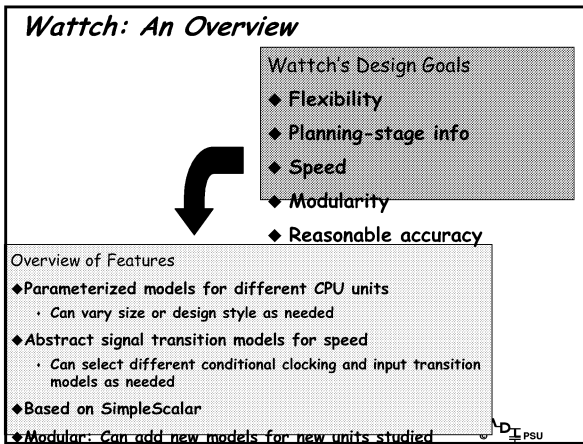
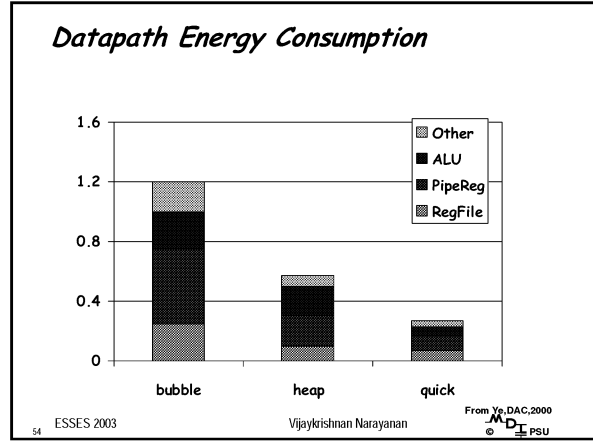
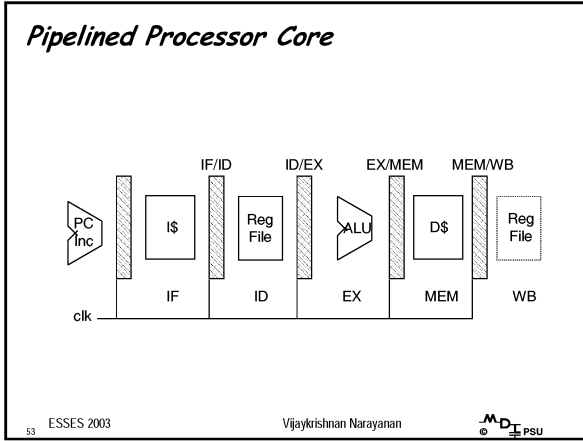
- A 2:4 decoder (with enable) →  $2^6$  row table

49 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### 6-T SRAM Cell

50 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU





### One Cycle in Wattch

	Fetch	Dispatch	Issue/Execute	Writeback/Commit
<b>Power (Units Accessed)</b>	<ul style="list-style-type: none"> <li>I-cache</li> <li>Bpred</li> </ul>	<ul style="list-style-type: none"> <li>Rename Table</li> <li>Inst. Window</li> <li>Reg. File</li> </ul>	<ul style="list-style-type: none"> <li>Inst. Window</li> <li>Reg File</li> <li>ALU</li> <li>D-Cache</li> <li>Load/St Q</li> </ul>	<ul style="list-style-type: none"> <li>Result Bus</li> <li>Reg File</li> <li>Bpred</li> </ul>
<b>Performance</b>	<ul style="list-style-type: none"> <li>Cache Hit?</li> <li>Bpred Lookup?</li> </ul>	<ul style="list-style-type: none"> <li>Inst. Window Full?</li> </ul>	<ul style="list-style-type: none"> <li>Dependencies Satisfied?</li> <li>Resources?</li> </ul>	<ul style="list-style-type: none"> <li>Commit Bandwidth?</li> </ul>

◆ On each cycle:

- determine which units are accessed
- model execution time issues
- model per-unit energy/power based on which units used and how many ports.

ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### Units Modeled by Wattch

- ⌘ **Array Structures**
  - ⌘ Caches, Reg Files, Map/Bpred tables
- ⌘ **Content-Addressable Memories (CAMs)**
  - ⌘ TLBs, Issue Queue, Reorder Buffer
- ⌘ **Complex combinational blocks**
  - ⌘ ALUs, Dependency Check
- ⌘ **Clocking network**
  - ⌘ Global Clock Drivers, Local Buffers

ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### Wattch Simulation Speed

- ◆ Roughly 80K instructions per second (PII-450 host)
- ◆ ~30% overhead compared to performance simulation alone
  - Could be decreased if power estimates are not computed every cycle
- ◆ Many orders of magnitude faster than lower-level approaches
  - For example, PowerMill takes ~1hour to simulate 100 test vectors on a 64-bit adder

ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### Architectural Level Analysis Considerations


- ◆ Very computationally efficient
  - requires predefined analytical and transition-sensitive energy characterization models
  - requires design only to RTL (with some idea as to the kind of functional units planned)
  - coarse grain - use of gated clocks implicit
- ◆ Simulation based so can be used to support architectural, compiler, operating system, and application level experimentation
- ◆ WattWatcher (Sente), DesignPower and PowerCompiler (Synopsys), prototype academic tools (Wattch - Princeton, SimplePower - PSU)

ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

*Tools for low-power IC design*


**Abstraction: architecture**  
 WattWatcher / Architect - Sente, Inc. Register transfer level analysis

**Abstraction: gate**  
 WattWatcher / Gate - Sente Full chip gate-level analysis  
 POET Viewlogic, Inc. -Cycle-by-cycle gate-level analysis  
 Power Compiler - Synopsys, Inc. Gate-level power optimization  
 Design Power - Synopsys Gate-level probabilistic analysis  
 QuickPower - Mentor Graphics Corp. Cycle-by-cycle gate level analysis  
 Power Gate - Epic Design Automation Cycle-by-cycle gate level analysis

61 ESSES 2003 Vijaykrishnan Narayanan 

*Tools for low-power IC design*


**Abstraction: transistor**  
 PowerMill - Epic Transistor-level analysis  
 LSimPower - Mentor Transistor-level analysis  
 AMPS - Epic Transistor-level optimization  
 RailMill - Epic Transistor-level electromigration and IR voltage drop analysis  
 PowerArc - Epic Circuit characterization  
 Thunder & Lightning - Simplex Solutions Inc.  
 Transistor-level electromigration and IR voltage drop analysis  
 Star-Sim - Avant! Corp. High-capacity SPICE compatible transistor- level analysis  
 Star-HSPICE - Avant! Circuit-characterization  
 Star-Power - Avant! Transistor-level electromigration and IR voltage drop analysis

62 ESSES 2003 Vijaykrishnan Narayanan 

*CASTLE: Measuring Power Data from Event Counts*

Basic idea:

- ◆ Most (all?) high-end CPUs have a bank of hardware performance counters
- ◆ Performance counters track a variety of program events
  - Cache misses
  - Branch mispredicts...
- ◆ If these events are also the main power dissipators, then we can hope these counters can also help power measurement
- ◆ Estimate power by weighting and superimposing event counts appropriately


63 ESSES 2003 Vijaykrishnan Narayanan 

*CASTLE: Details & Methodology*

Counter Values

$$\begin{matrix} \text{Appls} \\ \left[ \begin{array}{ccc} C_{11} & \dots & C_{1M} \\ \vdots & & \vdots \\ C_{IN} & \dots & C_{NM} \end{array} \right] \begin{matrix} \left[ \begin{array}{c} W_1 \\ \vdots \\ W_M \end{array} \right] \end{matrix} = \begin{matrix} \left[ \begin{array}{c} P_1 \\ \vdots \\ P_M \end{array} \right] \end{matrix}
 \end{matrix}$$

- ◆ Gather M counts for N training applications
- ◆ Compute weights using least-squares error
- ◆ Use these weights ( $W_1-W_M$ ) to estimate power on other apps
- ◆ Consider accuracy of power estimates compared to other power measurements

64 ESSES 2003 Vijaykrishnan Narayanan 

**Example & Results**

Benchmark	Estimation Error (%)
go	2.36
m88ksim	-2.31
gcc	1.49
compress	4.49
li	1.04
jpeg	4.03
perl	-7.94
vortex	-6.36

- ◆ For each of M benchmarks in suite:
  - use counters from M-1 other benchmarks
  - determine weights using least-squares estimation
  - Then apply weights to this benchmark
  - Compare calculated power to that given by a Wattch simulation
  - A benchmark is never used in the calculation of its own weights

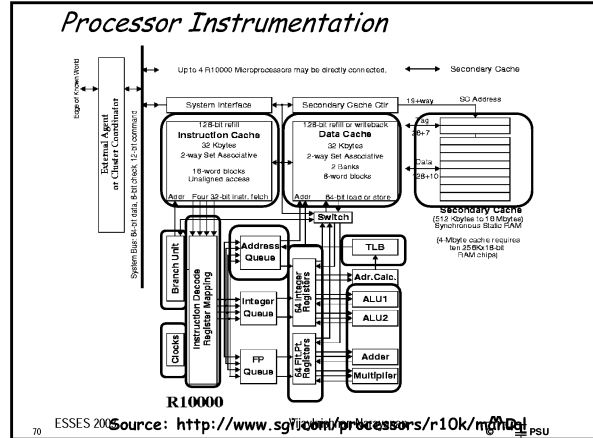
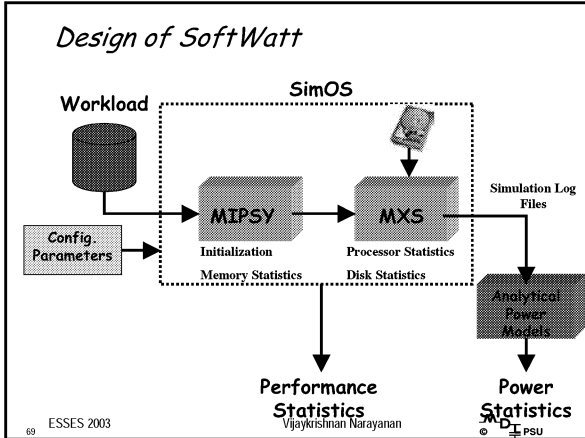
**Complete System Simulation**

**Complete Machine Simulation**

- ◆ Software: Application + OS + Runtime System + Compiler
- ◆ Architecture: Processor + Memory Subsystem + Peripherals
- ◆ Complete Machine Simulation Allows Studying Design Issues in an Integrated Manner.

**Softwatt**

- ◆ SimOS simulates Hardware in Sufficient Detail to run the entire IRIX 5.3 Operating System.
- ◆ Supports CPU Models: *Embra, Mipsy, MXS*.
- ◆ SoftWatt - Complete System Power Simulator built on top of SimOS.



### Power Modeling

- ◆ Analytical Power Models Based On:
  - Array Structures - Kamble et. al. - ISLPED '97,
  - Brooks et. al. - ISCA '00
  - Clock - Duarte et. al. - VLSI '01
  - CAM Structures - Palacharla et. al. - ISCA '97,
  - Brooks et. al. - ISCA '00

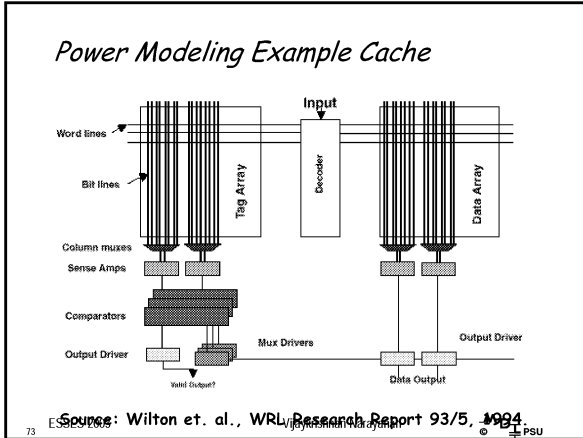
ESSES 2003  
Vijaykrishnan Narayanan  
PSU

### Power Model Validation

- ◆ R10000 Maximum Power Dissipation:
  - R10000 Datasheet - 30 W
  - SoftWatt - 25.3 W
- ◆ Estimation error due to non-availability of detailed circuit-level information.

ESSES 2003  
Vijaykrishnan Narayanan  
PSU





73

ESSES 2003 Vijaykrishnan Narayanan © PSU

### Power Modeling Example Cache

$$P_{\text{Cache}} = P_{\text{Data-Array}} + P_{\text{Tag-Array}}$$

$$P_{\text{Data-Array}} = P_{\text{Decoder}} + P_{\text{Wordline}} + P_{\text{Bitline}} + P_{\text{Senseamp}}$$

Therefore, Dynamic Energy Consumption of the Cache is:

$$E_{\text{cache}} = \# \text{ of Cache Accesses} * P_{\text{cache}}$$

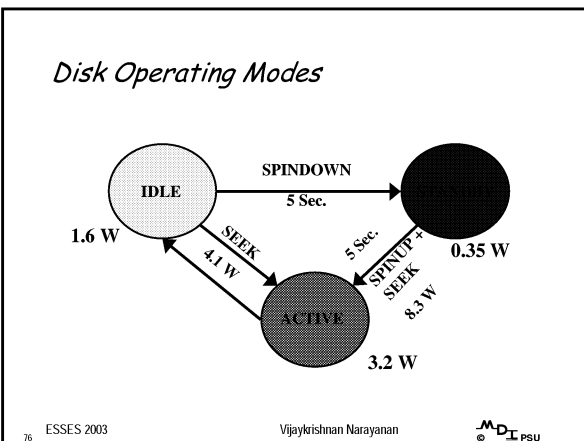
74

ESSES 2003 Vijaykrishnan Narayanan © PSU

- ### Components Not Modeled
- ◆ Display
  - ◆ Interconnects
  - ◆ Network Interface
  - ◆ Cooling System
  - ◆ Leakage Energy

75

ESSES 2003 Vijaykrishnan Narayanan © PSU




76

ESSES 2003 Vijaykrishnan Narayanan © PSU


### Processor Configuration

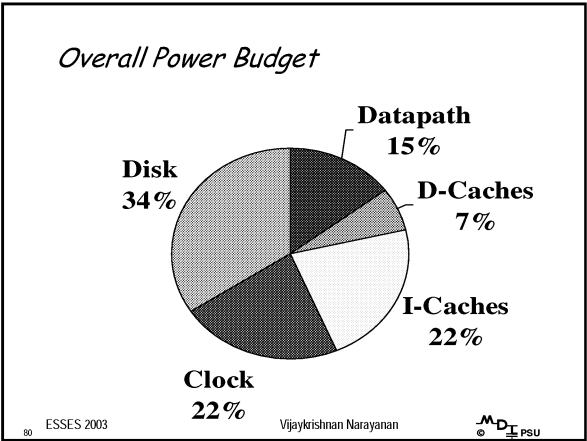
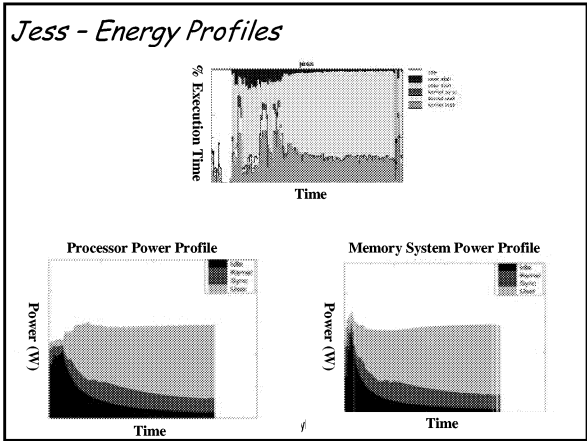
- ◆ 4-Issue Superscalar Processor
- ◆ 32 Entry Load/Store Queue
- ◆ 2 IALU, 2 FPALU
- ◆ 1024-Entry Branch-History Table

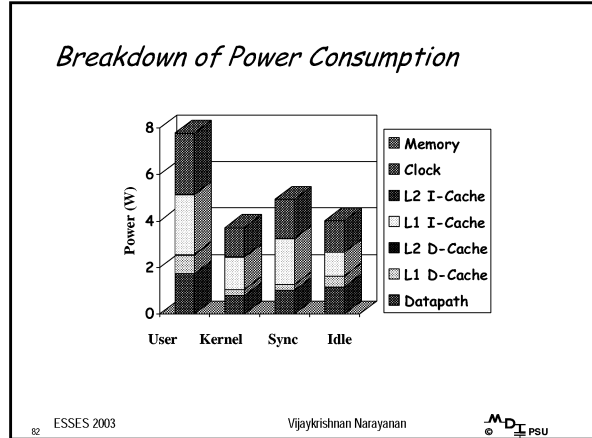
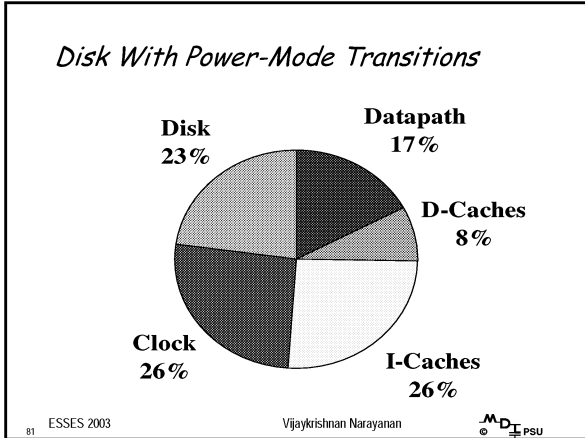
77 ESSES 2003 Vijaykrishnan Narayanan 

### Memory-System Configuration

- ◆ 2-Level Cache Hierarchy
- ◆ 32KB 2-Way Set-Associative L1 I- and D-Caches
- ◆ 1MB 2-Way Set-Associative L2-Cache
- ◆ 128 MB Memory

78 ESSES 2003 Vijaykrishnan Narayanan 



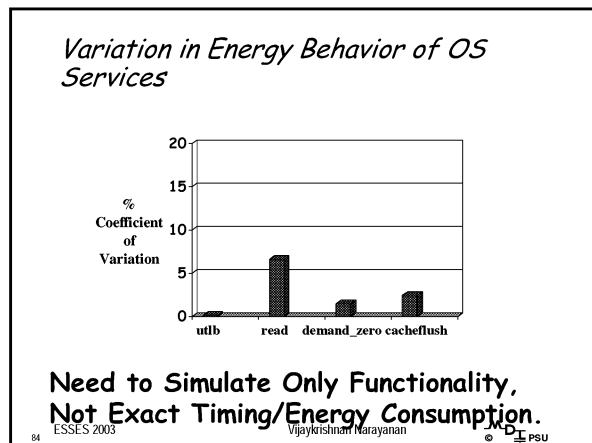


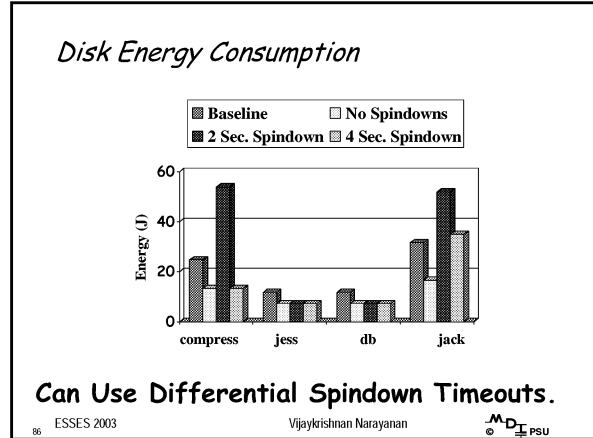
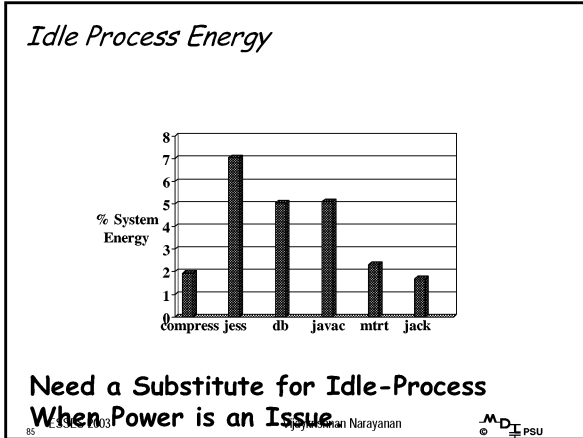
### OS Energy

◆ The OS Consumes Over 15% of the Overall System Energy.

Service	% Energy
utlb	65.43
read	13.4
demand_zero	4.35
cacheflush	1.19

83 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

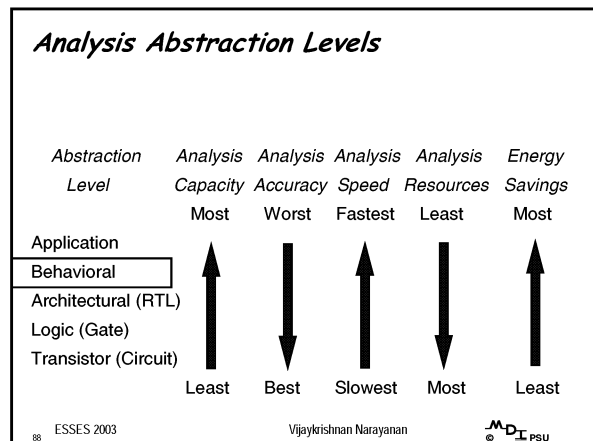




### Summary

- ◆ Disk Without Any Low-Power Modes is Single Largest Power Consumer.
- ◆ With Disk Power-Mode Transitions, Processor and Memory Components Dominate.
- ◆ OS Can Be a Significant Consumer of Power in the System.
- ◆ Need Power-Efficient Substitute for Idle-Process.

87 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU



### *Energy Estimation During Compilation*

- ◆ compilers have an important influence on system power consumption
  - determining types, order (to some extent), and number of instructions
- ◆ most compiler optimizations focus on performance and code size
- ◆ energy consumption is a very important metric for embedded systems

89 ESSES 2003

Vijaykrishnan Narayanan



### *Assembly vs. High Level Estimation*

- ◆ estimation of energy consumption at assembly level
  - restricting to a small number of alternative optimizations
  - very slow
  - difficult to integrate high-level energy optimization strategies
- ◆ estimating energy at high level is very important
  - allowing energy/performance tradeoffs involving high-level code and data optimizations.

90 ESSES 2003

Vijaykrishnan Narayanan



### *Energy-Aware Compilation (EAC) Framework*

- ◆ can estimate energy consumption of a high-level (source) code
  - using the architectural and technology parameters, energy models and energy/performance constraints
- ◆ determines the impact of optimizations on the energy consumption of different components of a system (on-chip or off-chip)
- ◆ be able to apply high level code and data transformations (loop-level and procedure-level) to optimize energy

91 ESSES 2003

Vijaykrishnan Narayanan



### *EAC Continue*

- ◆ can generate a high-level code
  - satisfying a given specific set of operating constraints (energy and performance) and a set of high level optimizations
- ◆ fast
  - key to the exploration of a large optimization alternatives
- ◆ moderately accurate
  - with %6 error margin (compared to a cycle-accurate architectural level energy simulator)
- ◆ currently supporting only array and nested-loop based applications

92 ESSES 2003

Vijaykrishnan Narayanan



**System Architecture**

- ◆ a single-issue, five stage pipe line
  - instruction fetch (IF), instruction decode/operand fetch (ID), execution (EXE), memory access (MEM), and write-back (WB)
- ◆ system components
  - datapath, cache, main memory, buses, and clock network.

93 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

**Major Parameters for Energy Estimation of System Components**

Component	Compiler-Supplied Parameters	Architecture/Circuit Parameters
datapath	number of activations for each component	structure, bit-width, and switched capacitance
cache	number of hits/misses number of reads/writes	bitline, wordline and decoder capacitance, tag size, voltage swings
main memory	number of accesses, intervals between accesses	off-chip capacitance, refresh rate, number and types of low-power modes
buses	number of transactions	wire capacitance, bus width
clock network	Number of execution cycles, number of memory stall cycles	Clock generation circuitry, size and capacitance of distribution buffers, load capacitance of clocked components

94 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

**Applications Supported by EAC**

- ◆ array dominated nested-loop based
- ◆ constructs
  - nested loops
  - assignment statements
  - multi-dimensional array references
  - scalar variable references

95 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

**Datapath Energy Estimation**

- ◆ need to estimate the number of instructions of each type associated with the high-level constructs
  - an assembly equivalent of each high-level construct is obtained using a back end compiler (a variant of gcc) with O2 level optimization
  - the number and type of each instruction are extracted from the assembly code corresponding to the high-level construct
- ◆ instructions with similar functionality and energy consumption are grouped together (ex. bnc and bcq)
- ◆ assumption of the clock gating to reduce stall energy in datapath due to branches or cache misses

96 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

**Cache Energy Estimation**

- ◆ high-level code optimizations have more impact on data cache than instruction cache
  - Considering only data cache energy consumption
- ◆ miss estimation technique proposed by McKinley
  - group the array references according to potential group reuse
  - for each representative reference, calculate a reference cost (misses during a complete execution of the inner most loop)
    - ◆ 1, temporal reuse
    - ◆ trip/(cls/stride), spatial reuse
    - ◆ trip, otherwise
  - sum of the contribution of each representative reference gives the total number of misses
- ◆ EAC differentiates between read and write misses

97 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

**Clock & Bus Energy Estimation**

- ◆ clock energy: need to estimate the number of execution cycles
  - counting the number of instruction (our system is a single-issue architecture)
  - counting the number of stall cycles due to misses
- ◆ bus energy: need to estimate the number of bus transactions
  - proportional to the number of cache and memory accesses

98 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

**Validation**

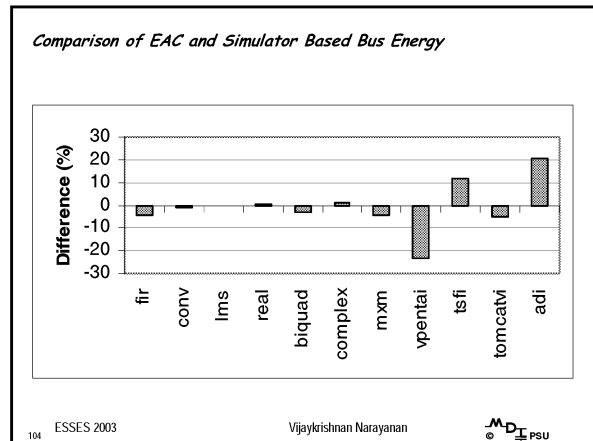
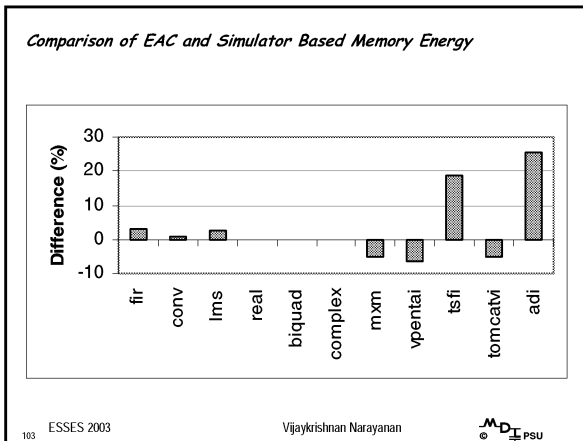
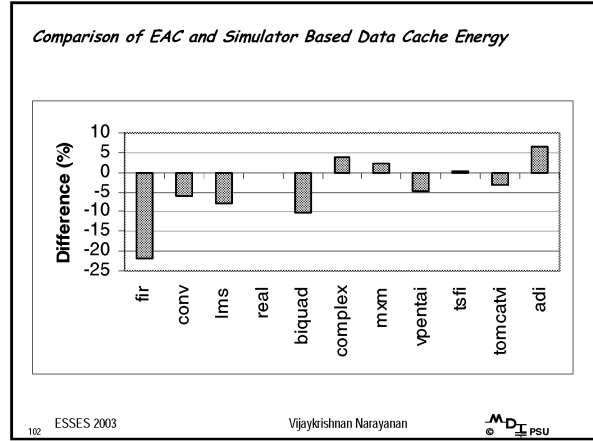
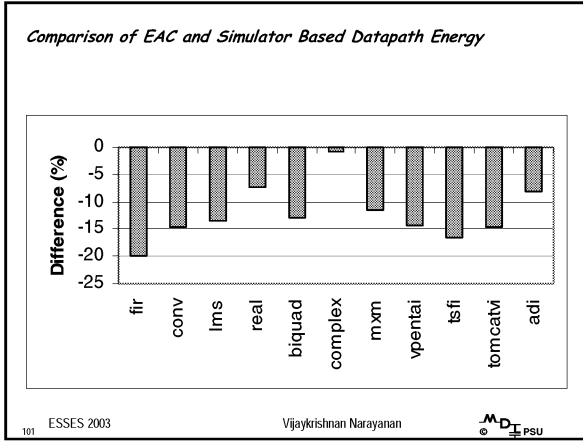
- ◆ transition-sensitive energy models for the data path (e.g, SimplePower)
  - based on the current and previous data inputs to a circuit (bit switching)
  - very accurate
- ◆ analytical energy models for other components (caches, main memory, and clock network, and buses)
  - based on activity-based approaches
  - assume a fixed (component-specific) energy consumption when the component is accessed
  - independent of the specific data input values

99 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

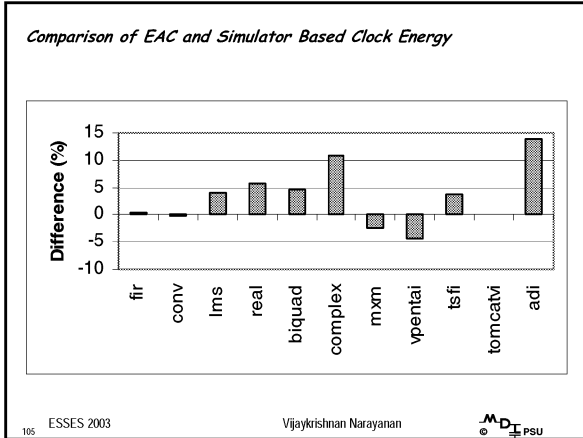
**Base Configuration Parameters**

Supply Voltage	3.3 V
Data Cache Configuration	4KB, 2-way, 32byte line size
Data Cache Hit Latency	1 cycle
Memory Access Latency	100 cycles
Per Access Read Energy for Cache	0.20 nJ
Per Access Write Energy for Cache	0.21 nJ
Per Access Energy for Memory	4.95 nJ
On-Chip Bus Transaction Energy	0.04 nJ
Off-Chip Bus Transaction Energy	3.48 nJ
Per Cycle Clock Energy	0.18 nJ
Technology	0.35 micron

100 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU







*Comparison of Estimation Times (in seconds) for EAC and the Simulator*

fir	0.31	434.44
conv	0.27	260.51
lms	0.28	598.89
real	0.34	563.24
biquad	0.35	1,366.60
complex	0.41	2,762.11
mxm (100)	0.28	36,602.23
vpentai	0.53	2,265.45
tsfi	0.30	547.12
tomcatvi	0.46	5,614.99
adi	0.33	2,652.31

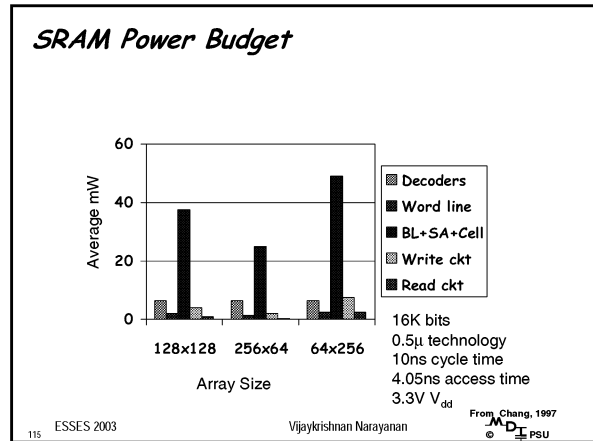
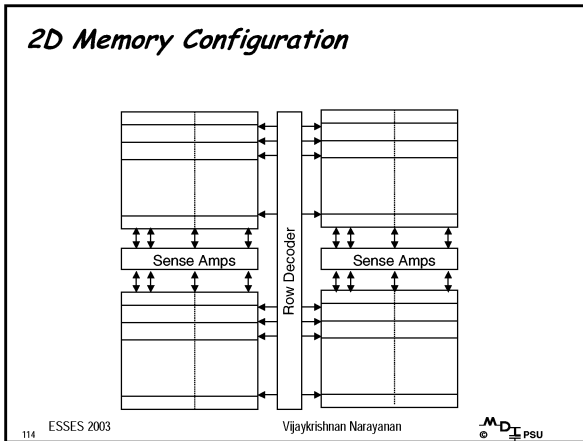
106 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

- Causes of Errors*
- ◆ datapath
    - estimating the number of instructions (5% contribution)
    - the impact of input transitions (7.2% contribution)
  - ◆ cache & main memory
    - inaccurate estimation of the number of cache hits and misses
      - ◆ not taking into account of conflict misses
      - ◆ not modelling data locality across separate nested loops
  - ◆ clock network
    - inaccurate estimation of the number of instructions
    - inaccurate prediction of the number of cache misses
    - not accounting for pipeline stalls (due to data or control hazards)
- 107 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

*Part II: Energy Efficient Memory Systems*

© M D T PSU





- ### Low Power SRAM Techniques
- ◆ Operating power reduction
    - memory bank partitioning
    - SRAM cell design
    - reduced bit line swing (pulsed word line and bit line isolation)
    - divided word line
    - bit line segmentation
  - ◆ Standby power reduction
  - ◆ Can use the above in combination!
- 116 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

- ### Memory Bank Partitioning
- ◆ Partition the memory array into smaller banks so that only the addressed bank is activated
    - improves speed and lowers power
    - word line capacitance reduced
    - number of bit cells activated reduced
  - ◆ At some point the delay and power overhead associated with the bank decoding circuit dominates (2 to 8 banks typical)
- 117 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### Partitioned Memory Structure

Advantages:

1. Shorter word and/or bit lines
2. Block addr activates only 1 block saving power

118 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### SRAM Cell

- ◆ 6-T SRAMs cell reduces static current (leakage) but takes more area
- ◆ Reduction of  $V_{th}$  in very low  $V_{dd}$  SRAMs suffer from large leakage currents
  - use multiple threshold devices (memory cells with higher  $V_{th}$  to reduce leakage while peripheral circuits use low  $V_{th}$  to improve speed)

119 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### Switched Power Supply with Level Holding

- ◆ Multi  $V_t$  device by changing Well voltages;  $V_t$  high during standby & low otherwise

120 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### Reduced Bit Line Swing

- ◆ Limit voltage swing on bit lines to improve both speed and power
  - need sense amp for each column to sense/restore signal
  - isolate memory cells from the bit lines after sensing (to prevent the cells from changing the bit line voltage further) - pulsed word line
  - isolate sense amps from bit lines after sensing (to prevent bit lines from having large voltage swings) - bit line isolation

121 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### Pulsed Word Line

- ◆ Generation of word line pulses very critical
  - too short - sense amp operation may fail
  - too long - power efficiency degraded (because bit line swing size depends on duration of the word line pulse)
- ◆ Word line pulse generation
  - delay lines (susceptible to process, temp, etc.)
  - use feedback from bit lines

122 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### Pulsed Word Line Structure

- ◆ Dummy column
  - height set to 10% of a regular column and its cells are tied to a fixed value
  - capacitance is only 10% of a regular column

123 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### Pulsed Word Line Timing

- ◆ Dummy bit lines have reached full swing and trigger pulse shut off when regular bit lines reach 10% swing

124 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

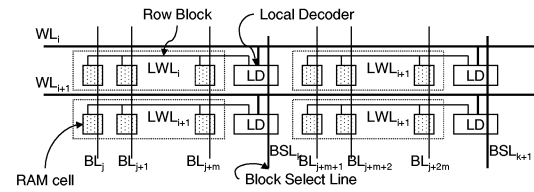
### Bit Line Isolation

125 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

**Divided Word Line**

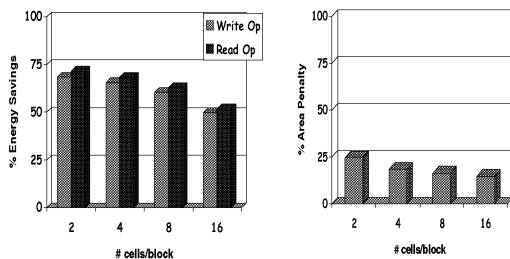
- ◆ RAM cells in each row are organized into blocks, memory cells in each block are accessed by a local decoder
- ◆ Only the memory cells in the activated block have their bit line pairs driven
  - improves speed (by decreasing word line delay)
  - lowers power dissipation (by decreasing the number of BL pairs activated)

**Divided Word Line Structure**



- ◆ Load capacitance on word line determined by number/size of local decoder
  - faster word line (since smaller capacitance)
  - now have to wait for local decoder delay
  - BSL produced by column decoder

**DWL Energy Area Tradeoffs**



**Bit Line Segmentation**

- ◆ RAM cells in each column are organized into blocks selected by word lines
- ◆ Only the memory cells in the activated block present a load on the bit line
  - lowers power dissipation (by decreasing bit line capacitance)
  - can use smaller sense amps

### Bit Line Segmented Structure

- ◆ Address decoder identifies the segment targeted by the row address and isolates all but the targeted segment from the common bit line
- ◆ Has minimal effect on performance

130 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### Cache Power

- ◆ On-chip I\$ and D\$ (high speed SRAM)
  - DEC 21164a (2.0V<sub>dd</sub>, 0.35μ, 400MHz, 30W max)
    - ◆ I/D/L2 of 8/8/96KB and 1/1/3 associativity
    - ◆ caches dissipate 25% of the total chip power
  - DEC SA-110 (2.0V<sub>dd</sub>, 0.35μ, 233MHz, 1W typ)
    - ◆ I/D of 16/16KB and 32/32 associativity (no L2 on-chip)
    - ◆ I\$ (D\$) dissipate 27% (16%) of the total chip power
- ◆ Improving the power efficiency of caches is critical to the overall system power

131 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### Cache Energy Consumption

- ◆ Energy Dissipated by Bitlines: precharge, read and write cycles
- ◆ Energy Dissipated by Wordlines: when a particular row is being read or written
- ◆ Energy Dissipated by Address Decoders
- ◆ Energy Dissipated by Peripheral Circuit - comparators, cache control logic etc.
- ◆ Off-Chip Main Memory Energy is based on per-access cost

132 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### Analytical Energy Model Example

- ◆ On-chip cache

$$\text{Energy} = E_{\text{bus}} + E_{\text{cell}} + E_{\text{pad}} + E_{\text{main}}$$

...

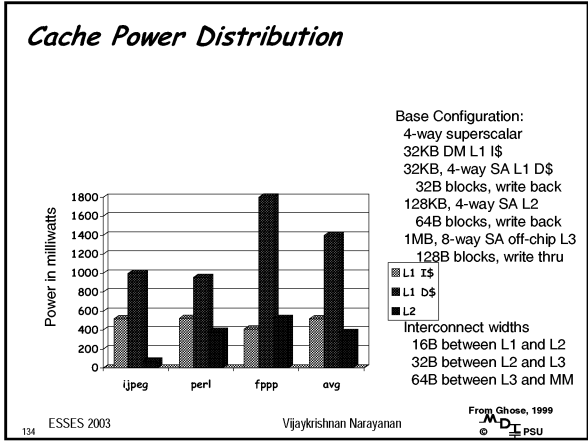
$$E_{\text{cell}} = \beta * (\text{wl\_length}) * (\text{bl\_length} + 4.8) * (\text{Nhit} + 2 * \text{Nmiss})$$

$$\text{wl\_length} = m * (T + 8L + St)$$

$$\text{bl\_length} = C / (m * L)$$

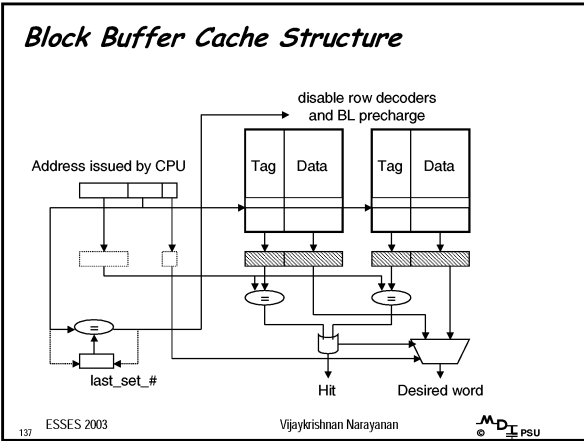
Nhit = number of hits; Nmiss = number of misses; C = cache size; L = cache line size in bytes; m = set associativity; T = tag size in bits; St = # of status bits per line; β = 1.44e-14 (technology parameter)

133 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

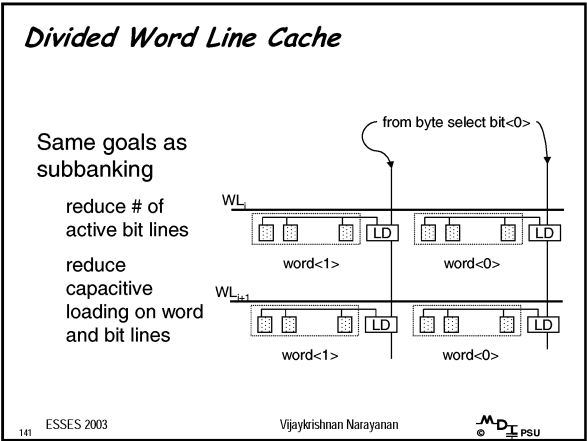
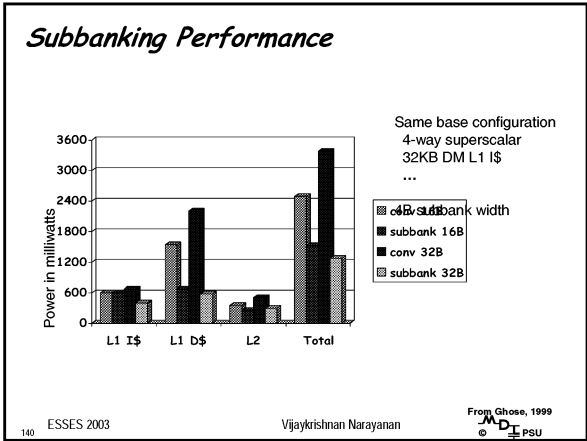
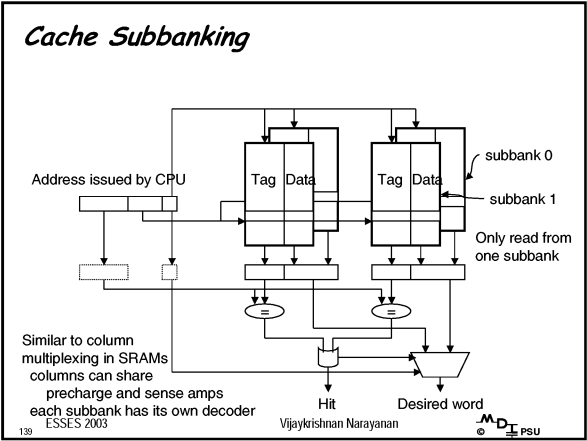
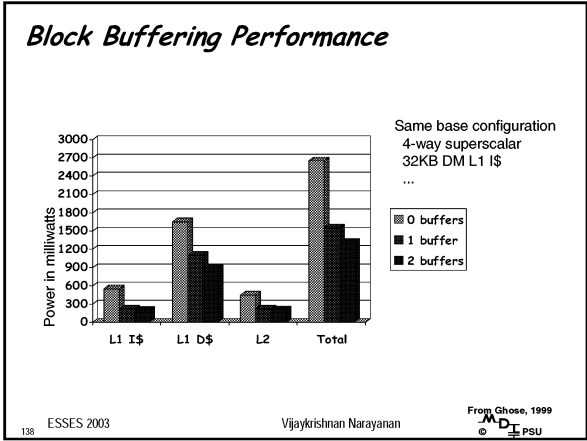


- ### Low Power Cache Techniques
- ◆ SRAM power reduction
  - ◆ Cache block buffering
  - ◆ Cache subbanking
  - ◆ Divided word line
  - ◆ Multidivided module (MDM)
  - ◆ MRU Cache
  - ◆ Modifications to CAM cell (for FA cache and FA TLB)
- ESSES 2003  
 Vijaykrishnan Narayanan  
 © M.D. PSU

- ### Cache Block Buffering
- ◆ Check to see if data desired is in the data output latch from the last cache access (i.e., in the same cache block)
  - ◆ Saves energy since not accessing tag and data arrays
    - minimal overhead hardware
  - ◆ Can maintain performance of normal set associative cache
- ESSES 2003  
 Vijaykrishnan Narayanan  
 © M.D. PSU







### Multidivided Module Cache

Address issued by CPU

With M modules and only one module activated per cycle, load capacitance is reduced by a factor of M (reduces both latency and power)

Can combine multidivided module, buffering, and subbanking or divided word line to get the savings of all three

142 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

### MRU Cache

- ◆ Access only the Most Recently Used way in a multi-way cache
- ◆ If it misses, access all the other ways
- ◆ If prediction is successful, can decrease the energy cost per access for other ways
- ◆ But has performance penalty and could increase system energy due to this penalty

143 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

### Most Recently Used Cache

Way 0 Way 1 Way 2 Way 3

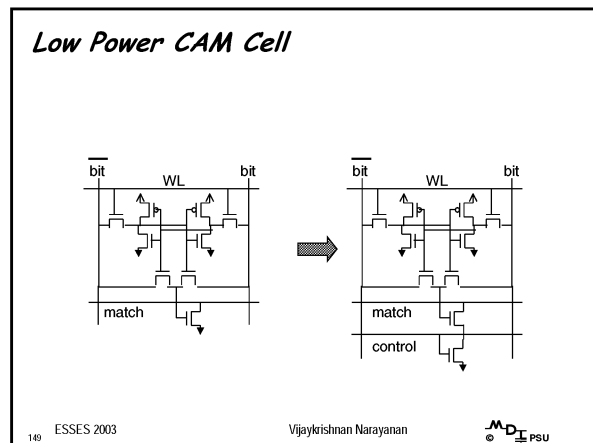
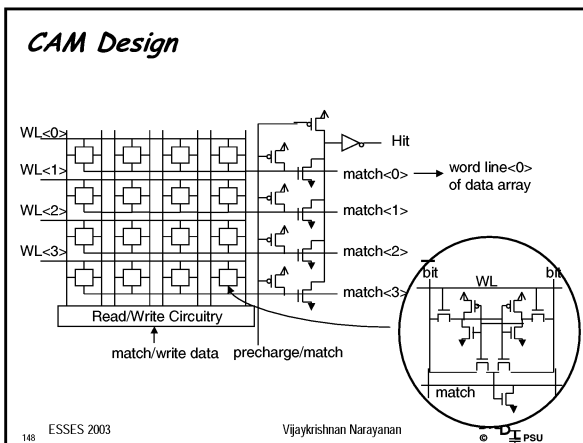
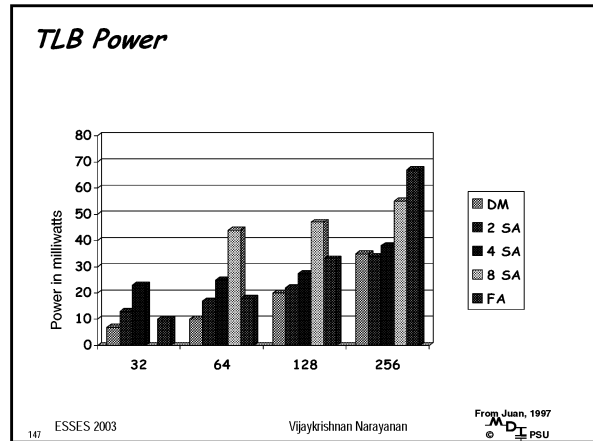
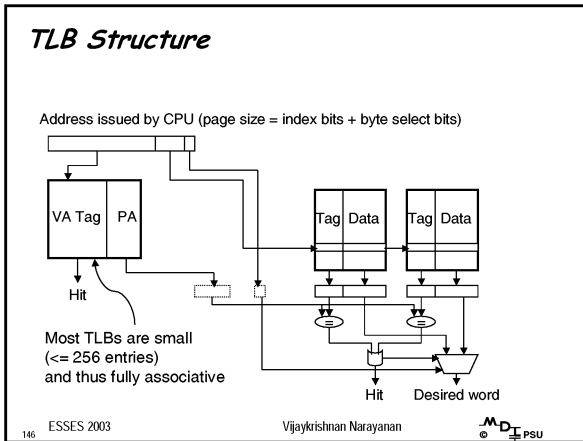
Data Select Logic

144 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU


### Translation Lookaside Buffers

- ◆ Small caches to speed up address translation in processors with virtual memory
- ◆ All addresses have to be translated before cache access
  - DEC SA-110 ( $2.0V_{dd}$ ,  $0.35\mu$ , 233MHz, 1W typ)
  - ◆ I\$ (D\$) dissipate 27% (16%) of the total chip power
  - ◆ TLB 17% of total chip power
- ◆ I\$ can be virtually indexed/virtually tagged

145 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU




*Quick Overview of Other Techniques*



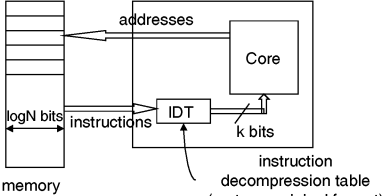
*Application-Specific Memories*


- ◆ **Data and Code Compression**
  - Custom instruction sets: ARM thumb code: interleaving of 32-bit and 16-bit thumb codes
  - Reduces memory size
  - Reduces width of off-chip buses
  - location of compression unit is important
  - Compress only selective blocks

151 ESSES 2003 Vijaykrishnan Narayanan 

*Hardware Code Compression*


- ◆ Assuming only a subset of instr's used, replace them with a shorter encoding to reduce memory bandwidth



152 ESSES 2003 Vijaykrishnan Narayanan 

*Other Techniques*

- ◆ **Customizing Memory Hierarchy**
  - Close vs. far memory accesses
  - Close - faster, less energy consuming, smaller caches
  - Energy per access increases monotonically with memory size
  - Automatic memory partitioning

153 ESSES 2003 Vijaykrishnan Narayanan 

**Memory Partitioning**

- ◆ A memory partition is a set of memory banks that can be independently selected
- ◆ Any address is stored into one and only one bank
- ◆ The total energy consumed by a partitioned is the sum of the energy consumed by all its banks
- ◆ Partitions increasing selection logic cost

Macii, 2000

**Scratch Pad Memory**

- ◆ Use of Scratch Pad Memory instead of Caches for locality
  - Memory accesses of embedded software are usually very localized
  - Map most frequent accessed locations onto small on-chip memory
  - Caches have tag overhead - eliminate by application specific decode logic
  - Map small set of most frequently accessed addresses to consecutive locations in small memory

Benini 2000

**Local Memory**

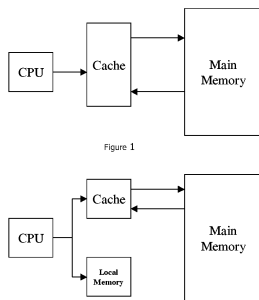


Figure 1

**Key References, Caches**

Ghose, Reducing Power in SuperScalar Processor Caches Using Subbanking, Multiple Line Buffers and Bit-Line Segmentation, *Proc. of ISLPED*, pp. 70-75, 1999.

Juan, Reducing TLB Power Requirements, *Proc. of ISLPED*, pp. 196-201, Aug 1997.

Kin, The Filter Cache: An Energy-Efficient Memory Structure, *Proc. of MICRO*, pp. 184-193, Dec. 1997.

Ko, Energy Optimization of Multilevel Cache Architectures, *IEEE Trans. On VLSI Systems*, 6(2):299-308, June 1998.

Panwar, Reducing the Frequency of Tag Compares for Low Power I\$ Designs, *Proc. of ISLPED*, pp. 57-62, 1995.

Shimazaki, An Automatic Power-Save Cache Memory, *Proc. of SLPE*, pp. 58-59, 1995.

Su, Cache Design Tradeoffs for Power and Performance Optimization, *Proc. of ISLPED*, pp. 63-68, 1995.



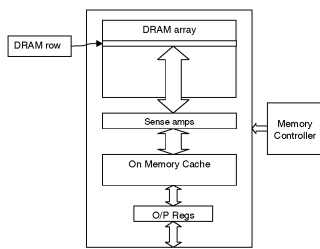
**Optimizing Refresh**

- ◆ **Selective refresh architecture (SRA)**
  - add a valid bit to each memory row and only refresh rows with valid bit set
  - reduces refresh 5% to 80%
- ◆ **Variable refresh architecture (VRA)**
  - data retention time of each cell is different
  - add a refresh period table and refresh counter to each row and refresh with the appropriate period to each row
  - reduces refresh about 75%

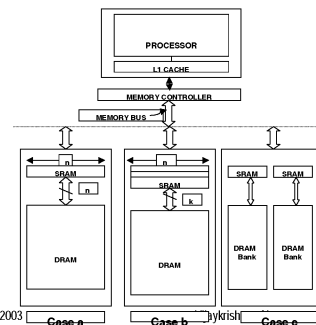
**Cached DRAM**

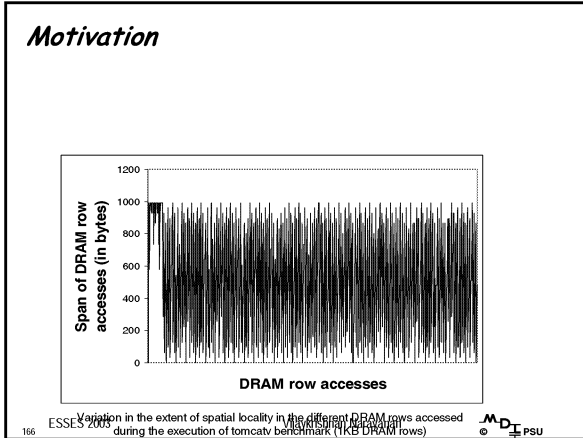
- ◆ Integrates a cache on a DRAM chip that optimizes cost/performance/energy.
- ◆ Relies on the fact that SRAM accesses are faster than DRAM accesses
- ◆ Different from traditional on-processor caches because of the width of transfer

**Cached DRAM**



**Different Cached DRAMs**





**Motivation**

- ◆ Scalar variables and arrays have different spatial locality.
- ◆ Each array has a different size and this can influence the extent of locality
- ◆ Data reuse might occur in different loop levels

```

for j= 1 to 32
  for i = 1 to 1024
    a[i]
    b[1024*i]
    c[j]
  endfor
endfor
    
```

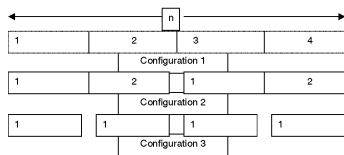
- Motivation**
- ◆ DRAM access takes around 30-90% of energy
  - ◆ DRAM access is also time consuming
  - ◆ Not loading the unnecessary data from a DRAM row can save energy.
  - ◆ It would also increase the hit rate in the on-memory cache.

- Variable Line Sized Cached DRAM (VLCDRAM)**
- ◆ Operates similar to the CDRAM except that it does not buffer a fixed portion of the DRAM row.
  - ◆ The size of the cache line is smaller than the DRAM row size.
  - ◆ Based on the extent of spatial locality in a given DRAM row access, multiple adjacent cache lines are used to buffer the selected width of the DRAM row.
  - ◆ The replacement policy identifies the least recently used cache line and also uses the lines adjacent to it if the buffering size is larger than a cache line.



**VLCDRAM**

◆ In order to enable transfer of multiple cache lines in the same cycle, an alignment needs to be enforced for the variable size blocks



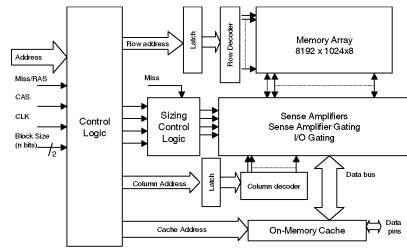
170

ESSES 2003

Vijaykrishnan Narayanan



**VL-CDRAM Architecture**



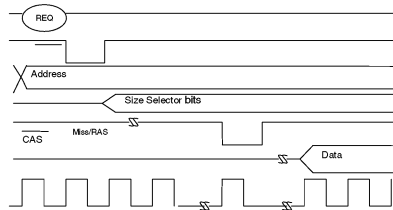
171

ESSES 2003

Vijaykrishnan Narayanan



**Timing Diagram for a Miss**



172

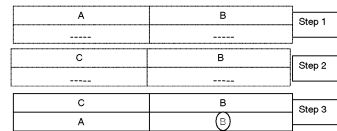
ESSES 2003

Vijaykrishnan Narayanan



**Constraints**

- ◆ First, if multiple cache lines chosen for replacement are dirty, the delay incurred is proportional to the number of cache lines that must be written back.
- ◆ Second, due to the caching of multiple cache lines of variable length, it needs to be ensured that there is no duplication of the data in the on-memory cache.



173

ESSES 2003

Vijaykrishnan Narayanan



### The Algorithm

To generate the log n bits to be associated with each load/store instruction

1. All the applications were profiled to identify the extent of spatial locality of load/stores.
2. All the load/stores that incur an on-memory cache miss were tracked as they initiate the DRAM row buffering.
3. To associate a single value with the static load/store instructions in the program, the extents observed at all dynamic instances of that static instruction were averaged.
4. This associated information indicates the number of cache lines that needs to be buffered when the DRAM row access was initiated by that static instruction.
5. This technique is called as the *Average Block Size (ABS)* technique.

174 ESSES 2003 Vijaykrishnan Narayanan © JMDT PSU

### The Algorithm

175 ESSES 2003 Average span for the blocks for each static load/store instance and their standard deviation for the m3m benchmark. © JMDT PSU

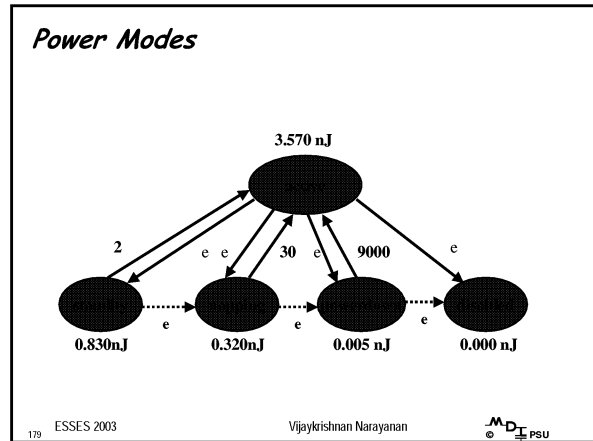
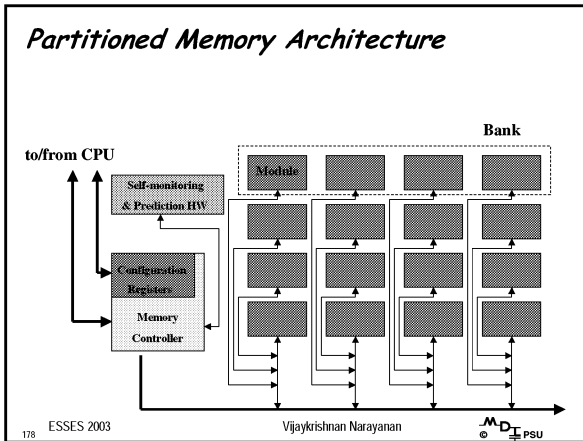
### Results

◆ Spatial Locality

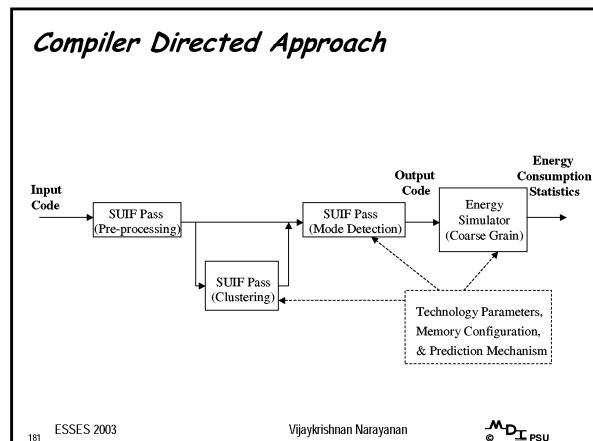
176 ESSES 2003 Varying spatial locality by benchmark. © JMDT PSU

### Comparison

177 ESSES 2003 Vijaykrishnan Narayanan © JMDT PSU



- ### Alternatives
- ◆ All memory modules are ON all the time
  - ◆ Mode Control Only
    - If not used, reduce power
    - No data/access pattern modifications
  - ◆ Mode Control + Optimizations
    - Data Transformations (e.g., Clustering, Interleaving)
    - Loop Optimizations (e.g., Loop Splitting)
- The bottom right corner contains the text 'ESSES 2003', 'Vijaykrishnan Narayanan', and the PSU logo.



### Array Clustering Heuristics

- ◆ Profile-Based
- ◆ Static Analysis-Based
  - Constructive Algorithm (Graph-Based)
  - Iterative Algorithm

**Module/Bank Configuration is Important!**

182 ESSES 2003 Vijaykrishnan Narayanan © M.D.T. PSU

### Array Access Profile (vpenta)

Phase Number	Array Variables							
	U1	U2	U3	U4	U5	U6	U7	U8
1	X			X	X	X	X	
2		X		X	X		X	X
3		X		X	X		X	X
4	X	X	X	X	X	X	X	X
5	X	X	X			X	X	X
6	X	X	X				X	X
7		X	X					X
8		X	X					X

183 ESSES 2003 Vijaykrishnan Narayanan © M.D.T. PSU

### Iterative Algorithm (vpenta)

184 ESSES 2003 Vijaykrishnan Narayanan © M.D.T. PSU

### Bank Access Profile (vpenta) (unoptimized)

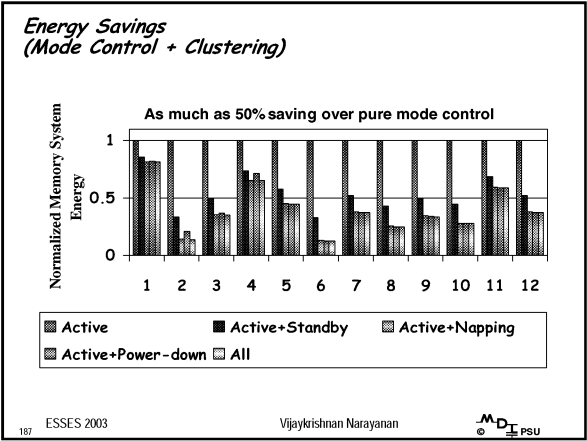
Phase Number	Memory Banks							
	B0	B1	B2	B3	B4	B5	B6	B7
1	X			X	X			
2	X			X	X	X		
3	X			X	X	X		
4	X	X	X	X	X	X		
5	X	X	X	X	X	X		
6	X	X	X	X	X	X		
7	X	X	X	X		X		
8	X	X	X			X		

185 ESSES 2003 Vijaykrishnan Narayanan © M.D.T. PSU

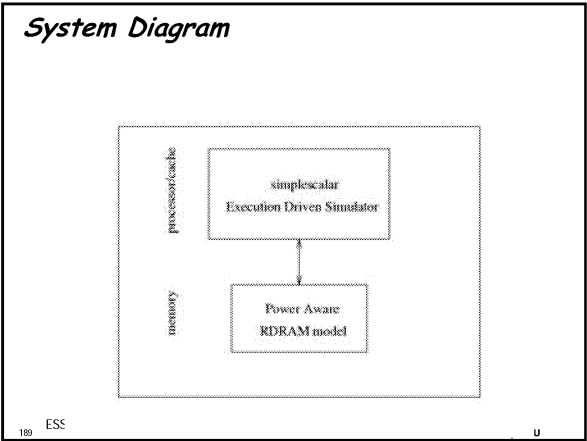
**Bank Access Profile (vpenta) (optimized)**

Phase Number	Memory Banks							
	B0	B1	B2	B3	B4	B5	B6	B7
1	X	X						
2	X		X	X				
3	X		X	X				
4	X	X	X	X				
5		X	X					
6		X	X					
7			X	X	X	X		
8			X	X	X	X		

ESSES 2003 Vijaykrishnan Narayanan © M D I PSU



- Hardware Optimizations for Partitioned Memory**
- ◆ Simple transitioning logic in current implementations (Intel BX, 820 chipsets) to only one level degradation
  - ◆ [Lebeck et al., 2000] explore runtime transitioning to other levels
  - ◆ [Delaluz et al., 2001] compiler-directed and runtime mechanism.
  - ◆ [Fan et. al. 2001] show memory access pattern is memory-less
  - ◆ [Nandagopal 2002] show history and memory is useful
- ESSES 2003 Vijaykrishnan Narayanan © M D I PSU



### Processor Parameters

Parameter	Value
Processor Core	
Clock Speed	400MHz
RUU size	256 instructions
LSQ size	128 instructions
Fetch Queue Size	8 instructions
Fetch width	8 instructions/cycle
Decode width	8 instructions/cycle
Issue width	8 instructions/cycle
Commit width	8 instructions/cycle
Cache/Memory	
L1	Unified 32KB, 4-way, 32byte blocks
L2	Unified 256KB, direct mapped, 64byte blocks
Memory	8x 256Mbit RDRAM 256MB

190

PSU

- ### Memory Module Idleness
- ◆ Transition module to appropriate low power mode when it is idle
    - When to transition to the lower power mode?
    - Which mode to transition to?
    - How much additional time penalty can we incur?
    - When to transition back to active state?

191

ESSES 2003

Vijaykrishnan Narayanan

PSU

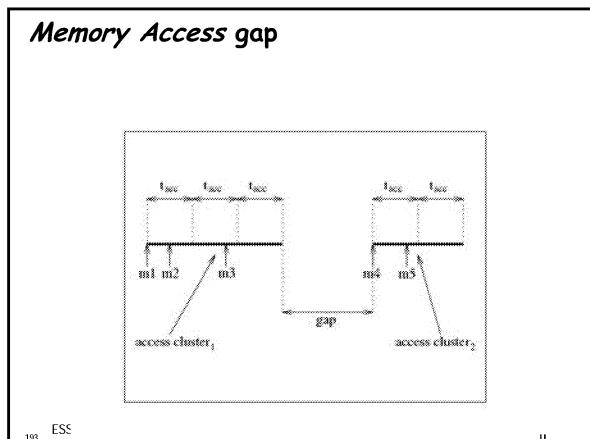
- ### Definitions
- ◆ *Clustered accesses* - sequence of accesses arrive before completion of previous access.
  - ◆ *Gap* - interval between two successive access clusters
  - ◆ Predictability of *gap* duration is important for power-mode control

192

ESSES 2003

Vijaykrishnan Narayanan

PSU



193

ESSES

U

### Optimal Policies

- ◆ **OPT\_time**
  - Minimum energy when gap duration needs to remain the same as the base case
- ◆ **OPT\_energy**
  - Minimum energy at the expense of larger execution time
- ◆ **OPT\_mode**
  - Minimum energy-delay given that transition to active state happens only when there is a access (incurs resynchronization latency)

194 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### OPT\_time Timing Diagram

195 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### OPT\_energy Policy

- ◆ **OPT\_energy** concentrates on minimizing energy first
- ◆ Range of gap durations exist where incurring a small additional latency results in a lower power mode, lower energy

196 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

### OPT\_energy Results

197 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

**OPT\_mode Policy**

- ◆ Useful for comparing with practical policies that do not try to predict *gap* duration.
- ◆ Incur resynchronization delay on first access in next cluster

198 ESSES 2003 Vijaykrishnan Narayanan © M.D. PSU

**OPT\_mode Timing Diagram**

199 ESS U

**OPT\_mode Results**

200 ESS U

**OPT\_mode Policy**

- ◆ Observation: 3-11% within *opt\_time*
- ◆ Heuristic: Predicting mode may be sufficient instead of exact *gap* duration

201 ESSES 2003 Vijaykrishnan Narayanan © M.D. PSU



**Behavior of OPT\_time**

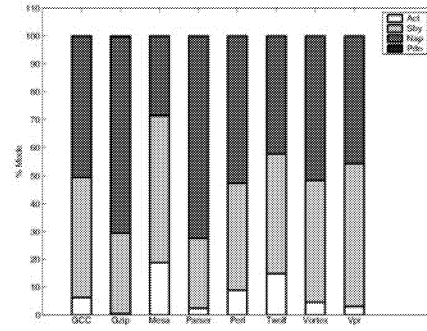
- ◆ Number of times a particular mode (active, standby, nap, powerdown) was selected
- ◆ Number of cycles spent in each mode ( and in transitions)
- ◆ Amount of energy consumed in each mode (and in mode transitions).

202 ESSES 2003

Vijaykrishnan Narayanan



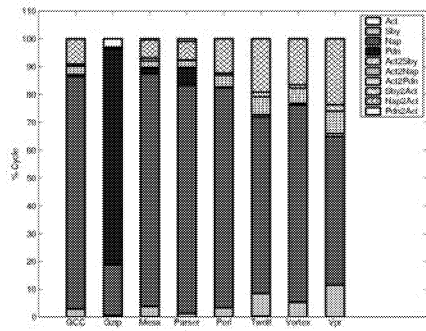
**Opt\_time Mode Distribution**



203 ESS

U

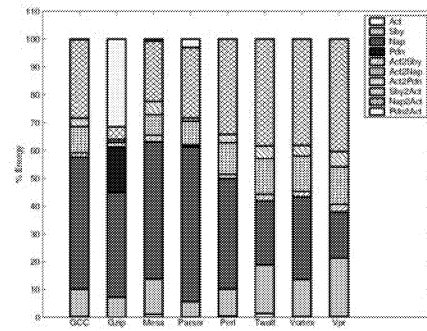
**Opt\_time Cycle Distribution**



204 ESS

U

**Opt\_time Energy Distribution**



205 ESS

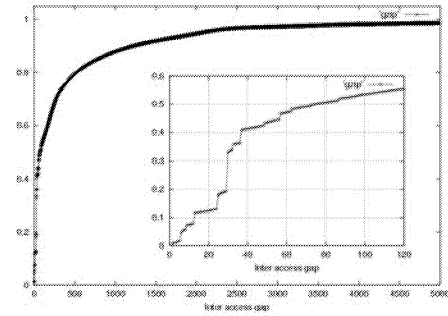
U

*Cumulative distribution functions*

- ◆ The % of *gap* durations less than a given time interval is plotted against the time interval
- ◆ [Fang01] claim exponential distribution, indicating memory-less behavior of *gap* distributions.
- ◆ Execution-driven simulations show jumps in accumulated % in the left-hand portion of the graph.
- ◆ Jumps correspond to *gap* durations that occur more frequently than others

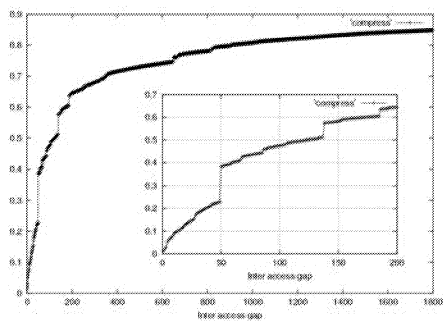
ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

*Gzip CDF*



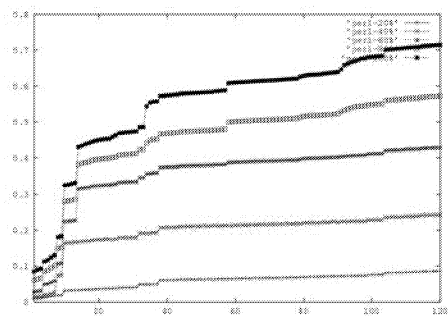
ESS U

*Compress CDF*



ESS U

*Perl CDF : Execution Phases*



ESS U

**Power Mode Control**

- ◆ Transition module immediately to a selected mode when idleness is detected
  - Static-standby
  - Static-Nap [Fang01]
- ◆ Use predictions to select the mode and the time at which the transitions must be effected
  - Prediction may be for power mode or gap interval
  - Constant Threshold Policies [Intel, Lebeck00]
  - Adaptive Threshold Policies [Delaluz01]
  - History Based Predictor [Delaluz01]

210 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

**Constant Threshold Predictor**

- ◆ Rationale: If a module has not been accessed for a while, it will not be needed for a while.
- ◆ Approach:

Active  $\xrightarrow{10 \text{ cycle Idleness}}$  Standby  $\xrightarrow{100 \text{ cycle Idleness}}$  Nap  $\xrightarrow{1M \text{ cycle Idleness}}$  Power down

211 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

**Adaptive Threshold Predictor**

- ◆ Rationale: Maybe different thresholds at different times, and adapt to the threshold dynamically
- ◆ Approach: If the threshold was conservative the previous time, double threshold, else if we transitioned wrongly reset threshold to CTP values.

212 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

**History Based Predictor**

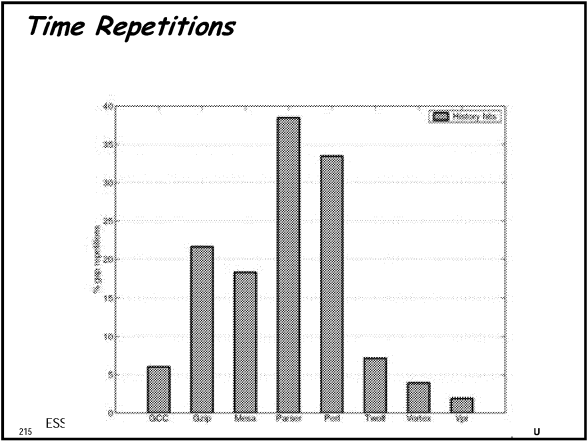
- ◆ Rationale: Previous gap duration indicative of future gap interval.
- ◆ Approach: Next gap duration is predicted to be the same as previous gap duration. Transition to appropriate mode immediately, and bring it back to active mode (hopefully before next access).

213 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

### Modified History Predictor

- ◆ Rationale: If next gap is much larger than previous gap, prediction is overly conservative
- ◆ Approach: Transition to low power modes after a preset threshold beyond previous gap interval.

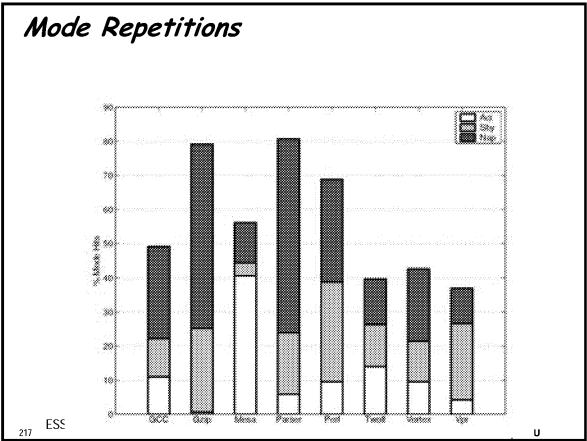
214 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

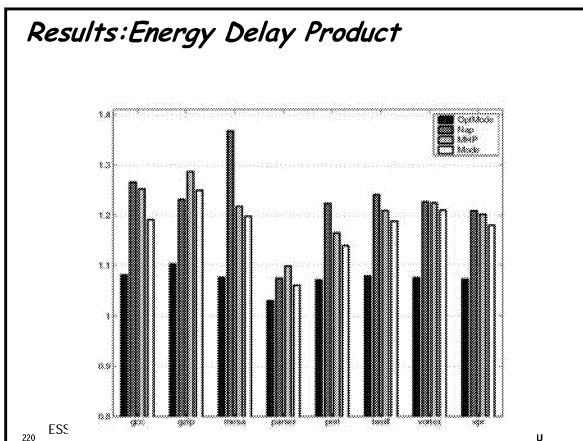
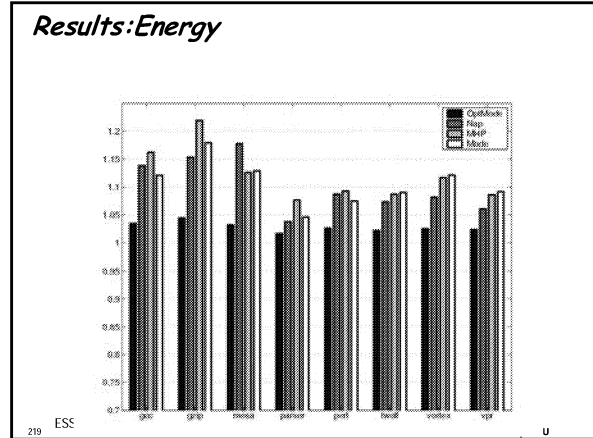
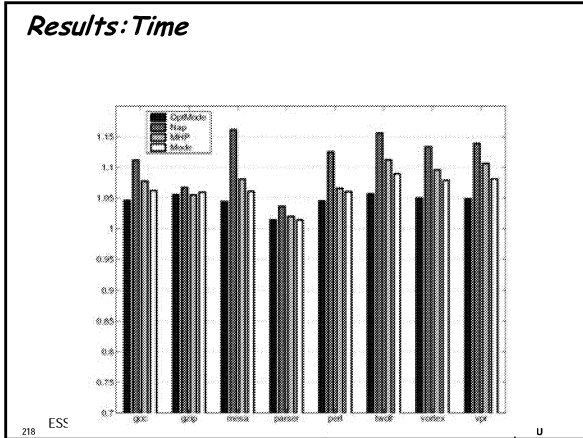


### History-based Mode Prediction

- ◆ Rationale: Gap durations don't repeat themselves too often, but consecutive gap durations have strong correlation
- ◆ Approach:
  - If you detect idleness, transition to the low power mode which would have been ideal for the preceding gap.

216 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU





**Summary of Results**

- ◆ History-based mode predictor is generally best scheme (EDP)
- ◆ Performs within 8%-25% of OPT\_mode EDP
- ◆ Gzip is the only benchmark where it loses out (marginally) to StaticNap
- ◆ When only execution time is considered, the gains are even more impressive.

221 ESSES 2003 Vijaykrishnan Narayanan

### Conclusions

- ◆ Application characteristics are key to mode control policies
- ◆ Memory access pattern is not purely memory-less
- ◆ History based policies are useful, especially for simple predictors
  - mode prediction policies
- ◆ Complex history predictors not be very beneficial
  - increase overhead without significant gains

222 ESSES 2003 Vijaykrishnan Narayanan © M.D. PSU

### Compiler Optimizations for MultiBanked Memories

223 ESSES 2003 Vijaykrishnan Narayanan © M.D. PSU

### Array Allocation Module

- ◆ Places arrays with similar access patterns into the same bank (set of banks)
- ◆ Two-step graph-based approach
- ◆ Builds an Array Relation Graph (ARG)
- ◆ Runs an algorithm that generates a maximum weight cover

224 ESSES 2003 Vijaykrishnan Narayanan © M.D. PSU

### Example

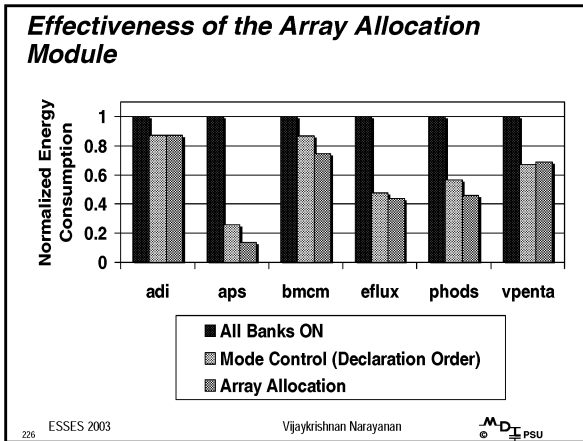
```

for (i = 0; i < N; i++)
    { A[ i ], B[ i ] }

for (i = 0; i < N; i++)
    { C[ i ], D[ i ], E[ i ] }

for (i = 0; i < N; i++)
    { E[ i ], F[ i ] }
    
```

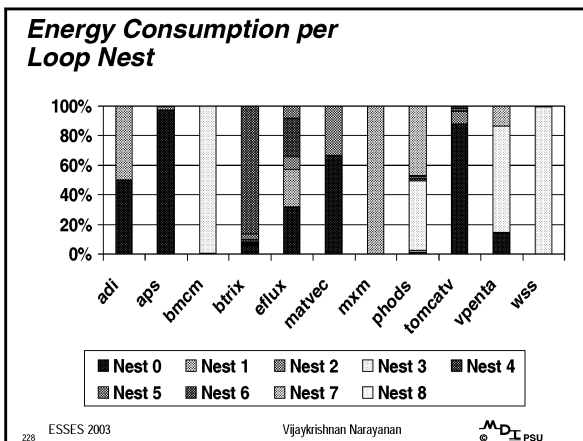
225 ESSES 2003 Vijaykrishnan Narayanan © M.D. PSU



### Benchmarks

Benchmark	Data size (MB)	Bank Configuration
adi	48.0	8 x 8 MB
aps	41.5	8 x 8 MB
bmcm	3.0	8 x 0.5 MB
btrix	47.3	8 x 8 MB
eflux	33.6	16 x 4 MB
matvec	16.0	8 x 8 MB
mxm	48.0	8 x 0.5 MB
phods	33.0	8 x 8 MB
tomcatv	56.0	8 x 8 MB
vpenta	60.0	32 x 2 MB
wss	30.0	8 x 0.5 MB

227 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU



### Optimization Module: Loop Fission

- ◆ Takes a nested loop and creates multiple nested loops each with a subset of the original statements
- ◆ Generates K+1 alternatives and selects the one with the minimum energy consumption

229 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

**Optimization Module:  
Loop Fission**

```

for (....){
  S1
  .
  Sk-1
  Sk
}

for (...){
  S1
  for (...){
    S2
    .
    Sk-1
    Sk
  }
}

for (.){
  S1
  .
  Sk-1
  for (...){
    Sk
  }
}

for (...){
  { S1 }
  for (...){
    { S2 }
    .
    for (...){
      { Sk-1 }
    }
  }
  for (...){
    { Sk }
  }
}
    
```

230 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

**Percentage Energy Improvements:  
Loop Fission**

Bchmrk	Alternative Fission Strategy								
	#1	#2	#3	#4	#5	#6	#7	#8	#9
adi	47.0	47.0	61.2						
aps	48.5	48.5	48.5	48.5					
efflux	47.0	45.2	43.3	66.9					
matvec	49.8	33.2	16.6	58.2					
tomcatv	49.5								
vpenta	8.2	23.5	16.6	24.9	18.0	16.6	20.7	8.2	48.4

231 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

**Optimization Module:  
Loop Splitting**

- ◆ Divides index set of a nested loop into two disjoint parts
- ◆ Tries to modify the nested loop structure by creating several nests
- ◆ Always legal
- ◆ Works from the innermost to the outermost index

232 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

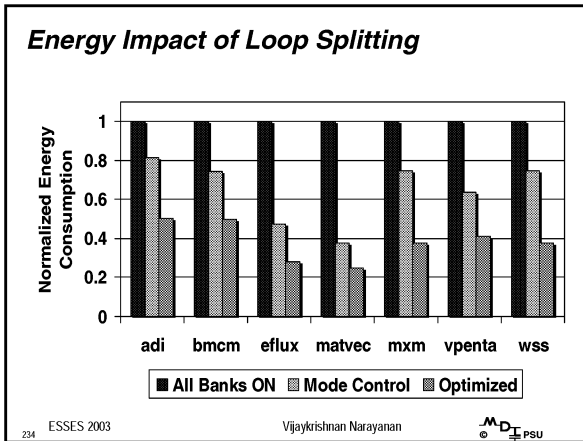
**Optimization Module:  
Loop Splitting**

```

for (i = 0; i < N; i++)
{ a[i], b[i]; }
    →
for (i = 0; i < N/2; i++)
{ a[i], b[i]; }
for (i = N/2+1; i < N; i++)
{ a[i], b[i]; }
    
```

233 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU





### Optimization Module: Array Renaming

- ◆ Use of live array variable analysis
- ◆ Reuse the same memory space for storing arrays with *disjoint* lifetimes

```

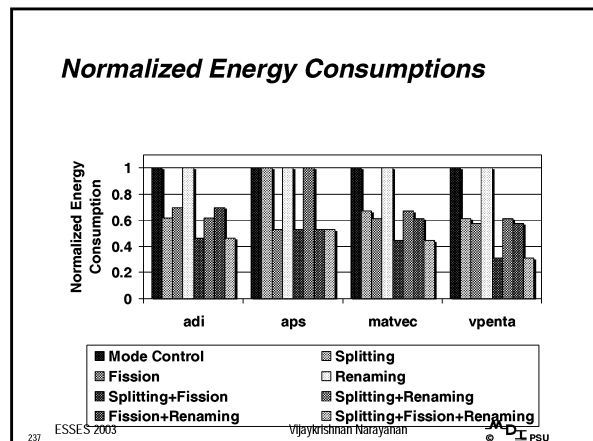
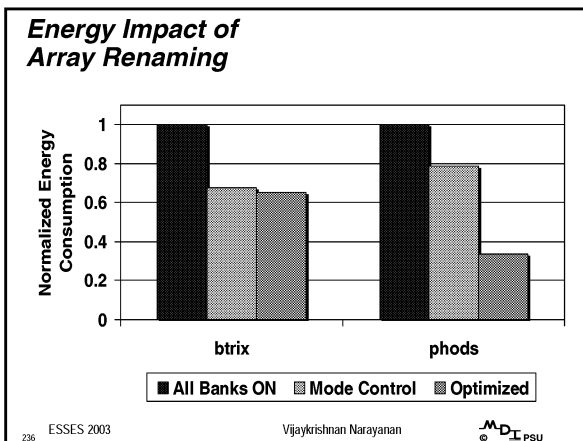
for (i = 0; i < N; i++) {a[i], c[i]}
.....
for (i = 0; i < N; i++) {b[i], c[i]}
    
```

➔

```

for (i = 0; i < N; i++) {a[i], c[i]}
.....
for (i = 0; i < N; i++) {a[i], c[i]}
    
```

ESSES 2003 Vijaykrishnan Narayanan © MDI PSU



**Array Interleaving**

```

for I = 1, N
  b += U1[I] + U2[I];
    
```

➔

```

for I = 1, N
  b += X[2I-1] + X[2I];
    
```

**Mappings:**

$U_1[I] \rightarrow X[2I-1] \quad \& \quad U_2[I] \rightarrow X[2I]$

239 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

**Array Interleaving**

```

for I = 1, N
  for J = 1, N
    b += U1[I][J] + U2[I][J];
    
```

➔

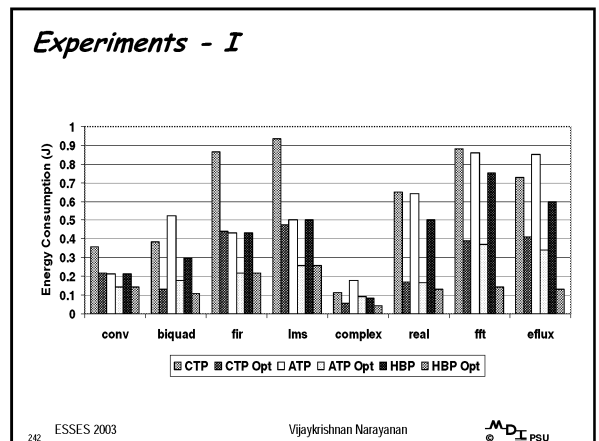
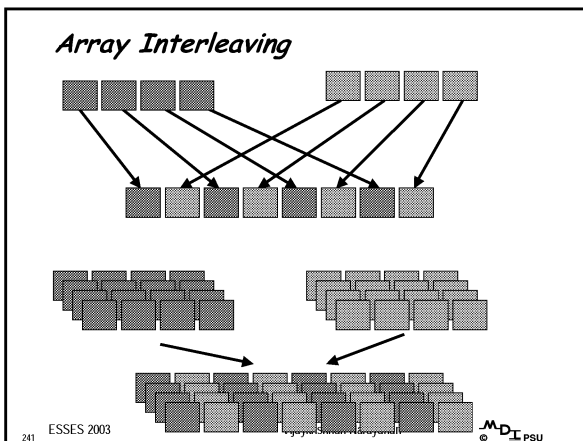
```

for I = 1, N
  for J = 1, N
    b += X[I][2J-1] + X[I][2J];
    
```

**Mappings:**

$U_1[I][J] \rightarrow X[I][2J-1] \quad \& \quad U_2[I][J] \rightarrow X[I][2J]$

240 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU



**OS-based Power Mode Control**

- ◆ OS has global view of the system.
- ◆ Information about actual physical frame allocation.
- ◆ The OS can determine points during execution of an application where banks would remain idle, so they can be transitioned to low power modes.

243 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

**Scheduler-based Approach**

- ◆ Transitions the banks at context switch time.
- ◆ We expect the same banks will be used in the next quantum.
- ◆ We need to keep track of the banks that are used by the application per time quantum.
- ◆ In order to do that, we propose the creation of a Bank Usage Table (BUT).

244 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

**Bank Usage Table (BUT)**

Process	Bank Number				
	B1	B2	B3	...	Bn
P1	X		X	...	
P2	X	X		...	X
...	...	...	...	...	
Pm	X			X	

- ◆ One row per application/task.
- ◆ An X means a bank was used in the previous quantum.

245 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

**Bank Usage Table (2)**

How do we update the BUT?

- ◆ Page Permissions. Wipe out access permissions on the page table for those virtual pages mapped to physical memory.
- ◆ Page Reference Bit. Reset reference bits on the page table.

246 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

**Bank Usage Table (3)**

- ◆ **TLB Miss Handler.** Modify the TLB miss handler. This is a hand-optimized, very efficient code. Not a good idea to modify it.
- ◆ **Dump the TLB contents.** Is expected that the TLB will do a good job capturing the program's behavior.

We use the first approach in this work.

247 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

**Bank Usage Table (4)**

248 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

**Integral Approach**

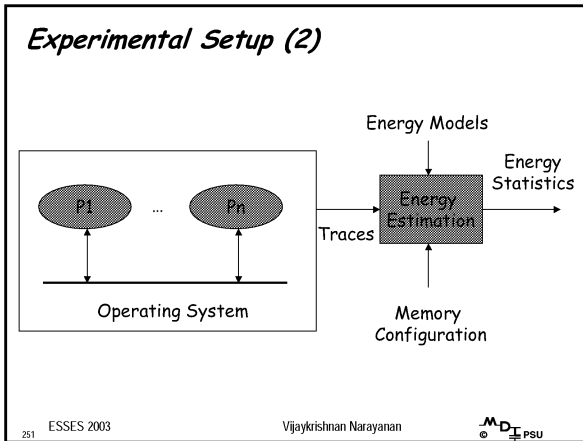
- ◆ Nothing prevents us from combining our scheduler-based technique with the hw-based ones (CTP, HBP).

249 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU

**Experimental Setup**

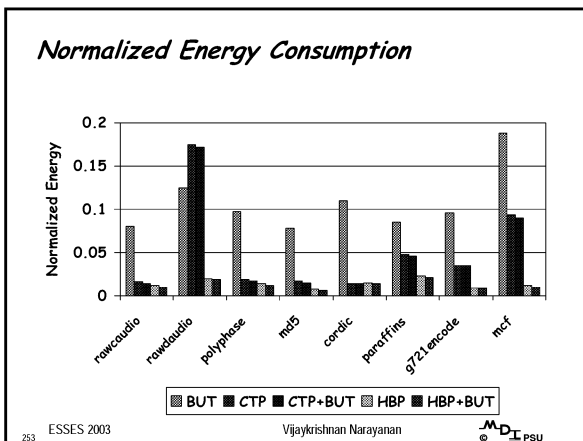
- ◆ SUN UltraSparc 5, Linux RH 6.2.
- ◆ Kernel 2.2.14.
- ◆ 128 MB RAM, 16 banks.
- ◆ BUT rows implemented as part of the task structure in the kernel.
- ◆ Scheduler and page fault handler were modified.
- ◆ Complete, full-fledged operating system used for evaluation.

250 ESSES 2003 Vijaykrishnan Narayanan © M D T PSU



### Benchmarks


Benchmark	Source
rawaudio	MediaBench
rawdabio	MediaBench
polyphase	MediaBench
md5	MediaBench
cordic	MediaBench
paraffins	Trimaran
g721encode	MediaBench
mcf	Spec2000



### Conclusions

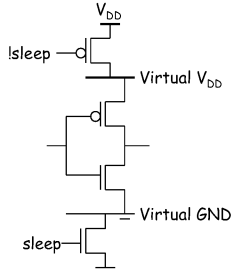
- ◆ We presented an scheduler-based technique to transition memory banks to low power modes.
- ◆ We show that important savings can be achieved (up to 92%) with little HW support.
- ◆ Further improvement can be achieved by applying our technique along with existing hardware-based techniques.


### Optimizing Leakage Power



### Reducing Power in Standby (Sleep) Mode

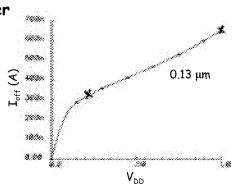
- ◆ For idle components, all power dissipation is due to leakage (subthreshold+gate)
- ◆ Can reduce leakage by gating the supply rails when the module is idle
  - In normal mode, sleep = 1 and the sleep transistors must present as small a resistance as possible (via sizing)
  - In sleep mode, sleep = 0, the transistor stack effect reduces leakage by orders of magnitude
  - Granularity has an area/performance cost
- ◆ Can eliminate leakage by switching off the power supply to the module




256 ESSES 2003 Vijaykrishnan Narayanan 

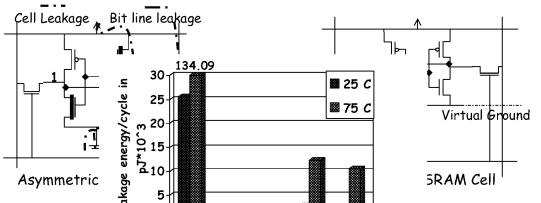
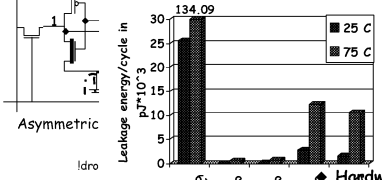
### Reducing Power in On-Chip Memory

- ◆ On-chip caches consume 15%-20% of the total power in today's uniprocessors
- ◆ Leakage in caches is becoming a major issue
  - leakage increase from 0.18μm to 0.13 μm is a factor of almost 7
- ◆ Can put cache lines not in the current working set into a low power mode ("drowsy") by lowering the supply voltage (e.g., 1V → 190mV)
  - Reduces standby power by 8x
  - Cell state is preserved
  - Incurs a performance penalty to "wake-up" line prior to access (<10ns)



Source: Jan Rabaey  
257 ESSES 2003 Vijaykrishnan Narayanan 


### Leakage Controlled SRAM Cells

Leakage energy/cycle in  $10^{-3}$  J

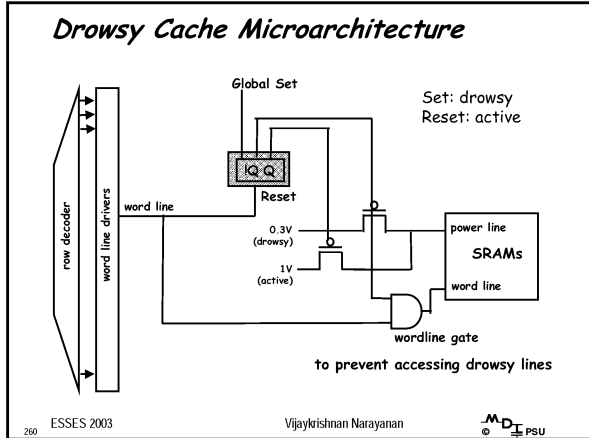
256 bits, 70 nm, 1 ns cycle

- ◆ Hardware versus software control of the sleep state
  - Only a small fraction of the cache lines are accessed in a time period

258 ESSES 2003 Vijaykrishnan Narayanan 

*Shutting down unused cache lines*

© M D PSU



### Global Counter Controlled Drowsy Cache

- ◆ Periodically put all cache lines into drowsy mode independent of their access history
  - ◆ Periodic global counter (~4000 cycles has good E-D trade-off) asserts Global Set
  - ◆ Wake-up cache line on access (incurs 1 cycle delay)
- ◆ Cache energy savings
  - standby energy by 75%
  - total energy by 55%
- ◆ Run time increases
  - <1% for D\$
  - >5% for some I\$ access patterns

Regular D\$      Drowsy D\$

Flautner, Kim, Martin, Blaauw, Mudge, ISCA'02

Kim, Flautner, Blaauw, Mudge, Micro'02

ESSES 2003      Vijaykrishnan Narayanan      © M D PSU

### Compiler Controlled Drowsy ICache

- ◆ Use an optimizing compiler to insert explicit drowse instructions at the loop granularity level

Normalized Energy

Loop (Compiler)    Drowsy-2K    Drowsy-4K

i230.comp    mpeg2dec    rawdatafp    vperm

Zhang et. al.

Micro'02

- Involves sophisticated program analysis
- Requires modification of the ISA
- Requires access to source code

◆ Drowsy works well for loops with long execution times

ESSES 2003      Vijaykrishnan Narayanan      © M D PSU

### Impact of Compiler Optimizations

- ◆ Many compiler optimizations can modify the instruction execution order leading to significantly different energy picture;
- ◆ Awareness of this impact helps develop better (e.g., energy-aware) compilation strategies;
- ◆ Four frequently-used loop transformation strategies are studied:
  - Loop fission (distribution)
  - Loop fusion
  - Loop tiling
  - Loop unrolling

263 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

### Loop Distribution

Fig. 9 (a) A code fragment with a loop. (b) The distributed version of (a). (c) The instruction cache layout for (a). (d) The instruction cache layout for (d).

264 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

### Impact of Compiler Optimizations

All values are normalized w.r.t to the case without power management. Strategies 1-3 are diff. Compiler controlled techniques

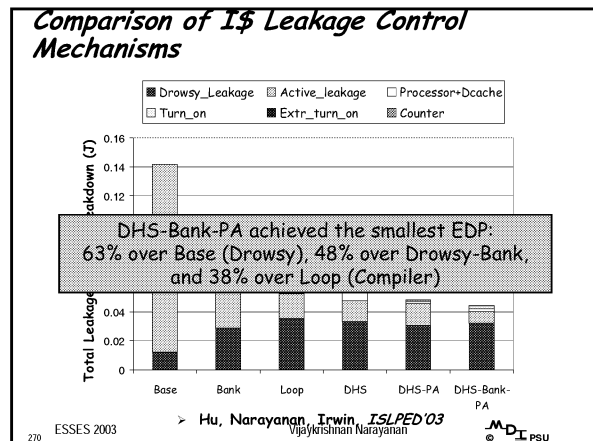
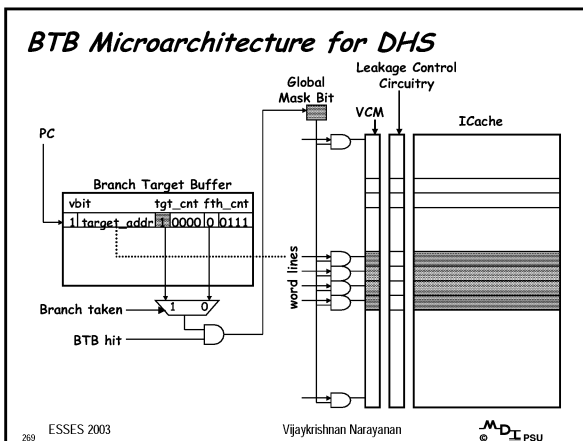
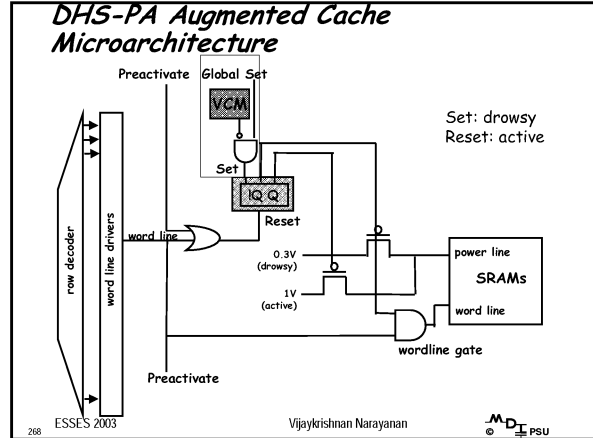
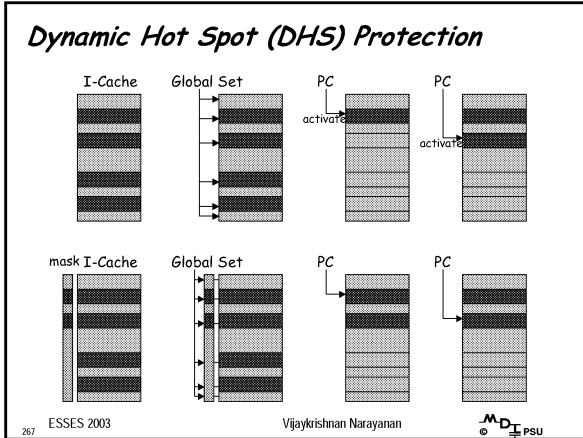
265 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

### Augmented BTB Controlled Drowsy ICache

- ◆ Program phases produce hot-spots in the cache
  - Instructions in a phase may not be localized in the address space (or the ICache)
  - Can track program hot-spots through the branch predictor (BTB)
- ◆ Add hardware (DHS) to BTB and cache to
  - Keep hot-spot cache lines from being turned off by the Global Set counter
  - Allow early turn-off of cache lines not in a newly detected hot-spot
- ◆ Take advantage of sequential access patterns to do just-in-time activation (PA) of the next cache line when the current line is accessed

266 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU





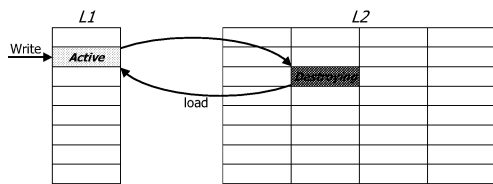
*Avoiding duplication in hierarchy*

- ◆ We implement five leakage reduction strategies that exploit data duplication in the cache hierarchy.
- ◆ We employ state-destroying or state-preserving mechanisms to L2 subblocks when their data also exist in L1.

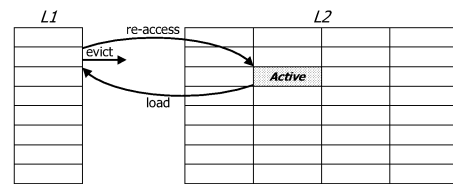
*Leakage Optimization Strategies (II)*

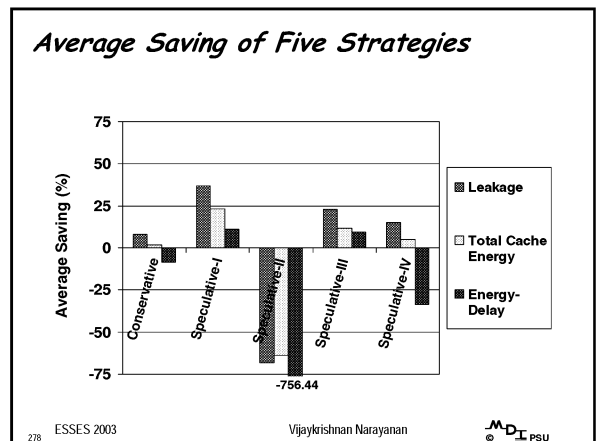
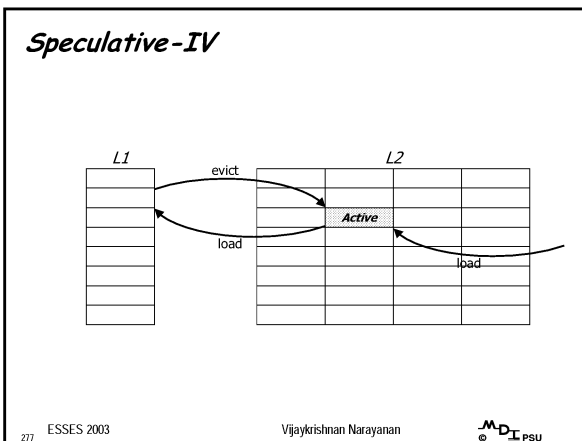
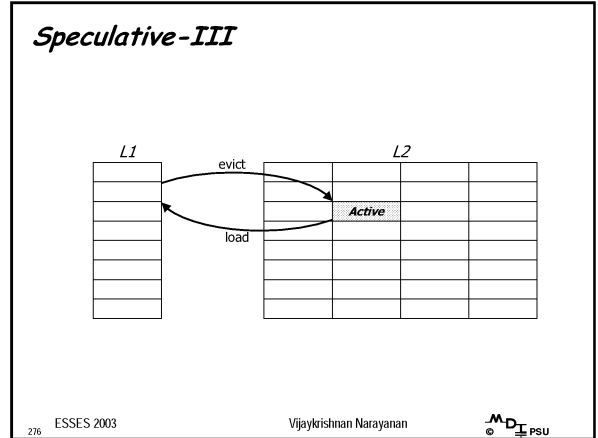
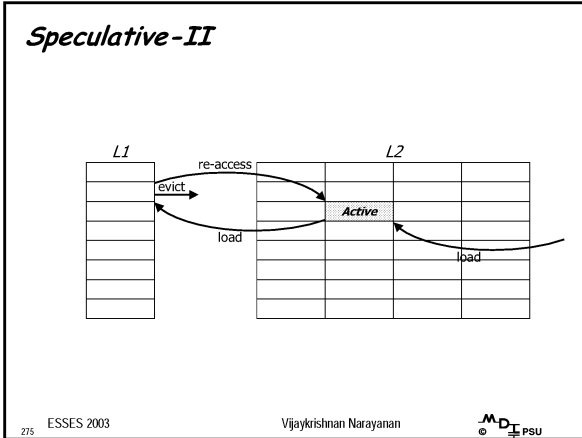
Strategy	When is L2 subblock turned off?	Mechanism in L2	When is L2 subblock reactivated?
Conservative	when L1 block becomes dirty	state-destroying	when accessed
Speculative-I	when L2 subblock is moved to L1	state-preserving	when accessed
Speculative-II	when L2 subblock is moved to L1	state-destroying	when accessed
Speculative-III	when L2 subblock is moved to L1	state-preserving	when L1 block is evicted
Speculative-IV	when L2 subblock is moved to L1	state-destroying	when L1 block is evicted

*Conservative*



*Speculative-I*

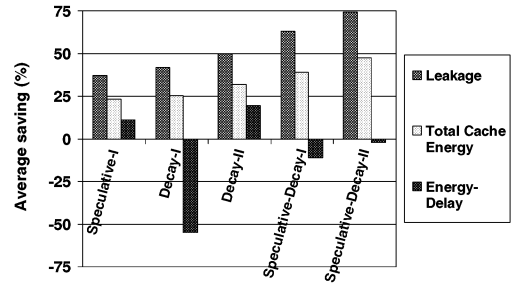




**Integration With Other Strategies**

- ◆ Cache decay, presented by Kaxiras et al., is a leakage energy reduction technique.
- ◆ Four strategies
  - Decay-I, state-destroying on both L1 and L2.
  - Decay-II, state-destroying on L1 and state-preserving on L2.
  - Speculative-Decay-I, applying speculative-I to L2 and cache decay to L1.
  - Speculative-Decay-II, applying speculative-I to L2 and cache decay to both L1 and L2.

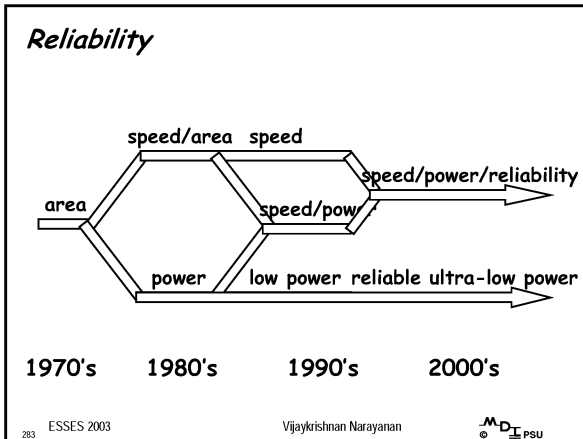
**Average Savings**



**Conclusion**

- ◆ Duplication of data at different levels of memory hierarchy is costly from the leakage energy perspective.
- ◆ Applying state-preserving leakage control strategy to L2 cache can reduce energy consumption significantly.
- ◆ Our strategies can be combined with other techniques to provide additional energy gains.

**Influence of Leakage Control on Reliability**



### Soft Errors

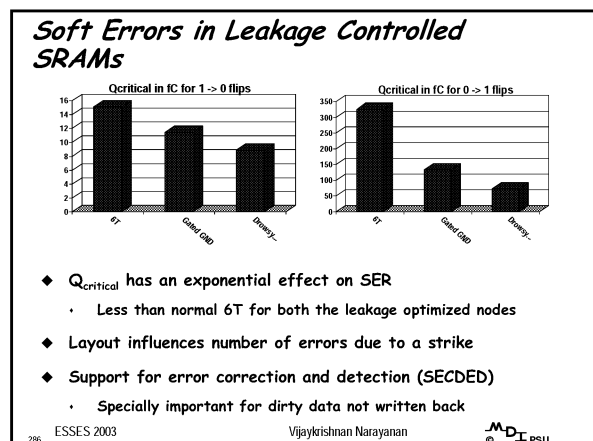
- ◆ Soft errors or transient errors are circuit errors caused due to excess charge carriers induced primarily by external radiations
- ◆ These errors cause an upset event but the circuit it self is not damaged.

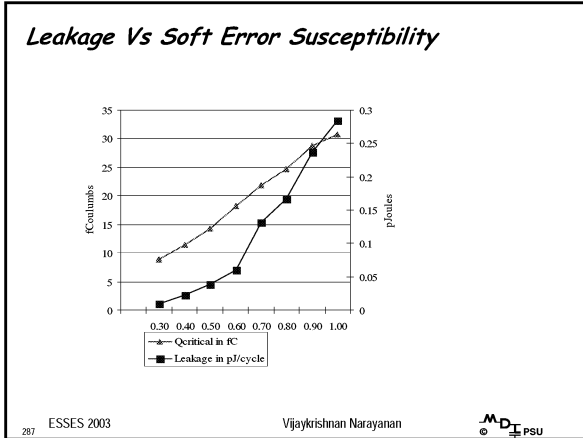
284 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

### Soft Errors

- ◆ For a soft error to occur at a specific node in a circuit, the collected charge  $Q$  at that particular node should be more then  $Q_{critical}$
- ◆ As CMOS device sizes decrease, the charge stored at each node decreases (due to lower nodal capacitance and lower supply voltages).
- ◆ This potentially leads to a much higher rate of soft errors

285 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

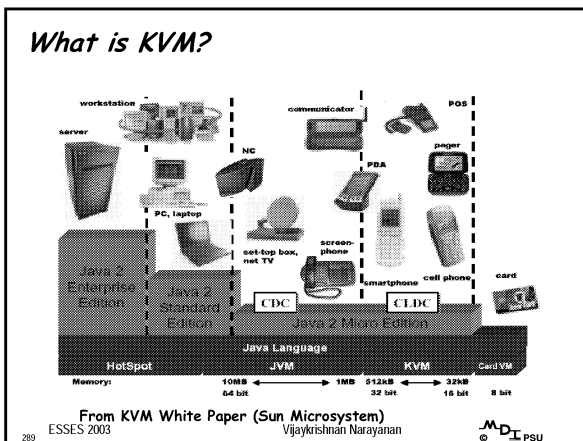




### Using System Software for Leakage Reduction

#### Case Study- Garbage Collection in Embedded Java Virtual Machines

ESSES 2003  
Vijaykrishnan Narayanan  
© M.D. PSU



- ### Resource Constraints on KVM
- ◆ Small memory budget
    - Typically 512 KB. "K" stands for "kilo"
    - Lots of research has been carried out
  - ◆ Limited energy supply
    - Powered by batteries
    - Not yet well exploited
  - ◆ Requires: redesign of all the components of virtual machine. Our work is focused on Automatic Memory Management Subsystem (Garbage Collector)
- ESSES 2003  
Vijaykrishnan Narayanan  
© M.D. PSU

### Garbage Collection in KVM

- ◆ **Why?**
  - To reclaim memory occupied by objects no longer needed by the application
- ◆ **When?**
  - Not enough free space in the heap for new object (conventionally)
- ◆ **How? - 3 Phases:**
  - **Mark** - find out garbage
  - **Sweep** - free the memory occupied by garbage
  - **Compact** - combine fragments (optional)

291 ESSES 2003 Vijaykrishnan Narayanan © MDI PSU

### Mark Phase - detection of garbage

- **Roots:** internally defined by the KVM implementation
- **Live Objects:** reachable from the roots
- **Garbage (Dead Objects):** not reachable from the roots, not accessible to the application

```

    graph TD
      root((root)) --> A((A))
      root --> B((B))
      A --> C((C))
      A --> D((D))
      B --> E((E))
      E --> G((G))
      F((F))
  
```

292 ESSES 2003 Vijaykrishnan Narayanan © MDI PSU

### Compaction Phase

- ◆ **Why?**
  - Heap fragmentation wastes heap memory, increases allocation cost due to longer free list scanning
- ◆ **How?**
  1. Slide live objects to one end of the heap and free blocks to the other end.
  2. Combine free blocks into one contiguous free area
  3. References to the moved objects are updated

293 ESSES 2003 Vijaykrishnan Narayanan © MDI PSU

### Mark / Sweep / Compaction

Before GC: Memory contains objects A-G and free space. Arrows show root pointing to A and B, and A pointing to C and D.

After Mark: Objects A, B, C, D, E, and G are shaded (Live). Object F is white (Unknown). Object G is grey (Garbage).


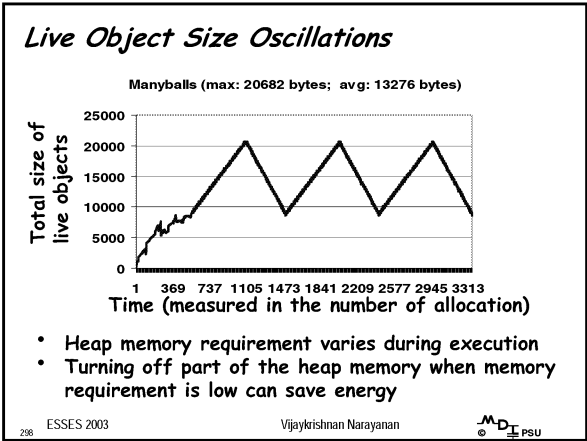
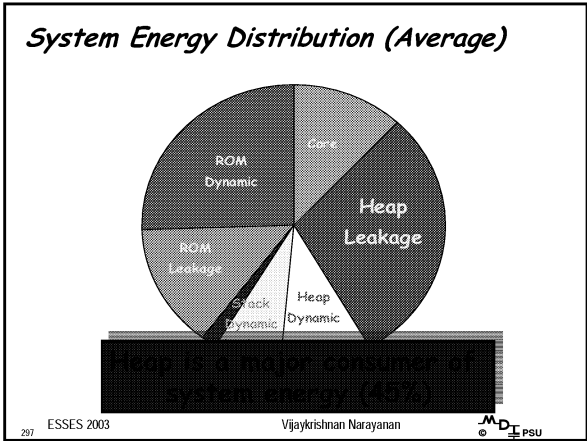
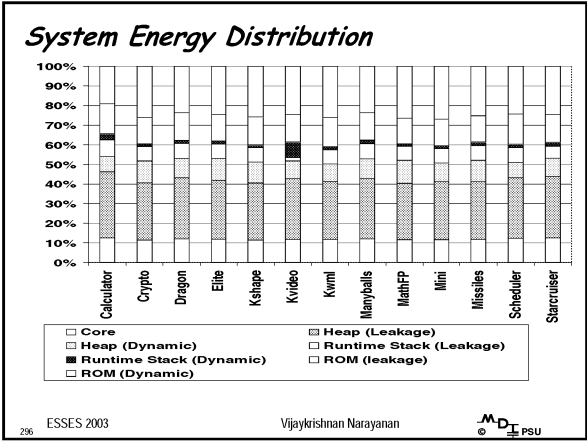
After Sweep: Objects A, B, C, D, E, and G are shaded (Live). Object F is white (Unknown). Object G is grey (Garbage).

After Compact: Live objects A-G are moved to the top of the heap. Free space is now a single contiguous block at the bottom.

■ Live ■ Garbage □ Unknown ■ Free

294 ESSES 2003 Vijaykrishnan Narayanan © MDI PSU

**Banked Memory Architecture  
&  
GC Controlled Bank Turn off**



### Banked Memory

1. Each bank can be turned on/off by software independently  
 2. When a bank is turned off, it consumes very little energy  
 3. If a bank is turned off, data in it is lost

299 ESSES 2003 Vijaykrishnan Narayanan © M.D. PSU

### Energy Characteristics of Memory Banks

Mode		Dynamic	Leakage	Data Lost
ON	Read	yes	yes	Not lost
	Write	yes	yes	Not lost
	No access	no	yes	Not lost
OFF		no	no	Lost

Note: turning on a currently off bank incurs 350 cycles' penalty (resynchronization cost)

300 ESSES 2003 Vijaykrishnan Narayanan © M.D. PSU

### Embedded System Configuration

301 ESSES 2003 Vijaykrishnan Narayanan © M.D. PSU

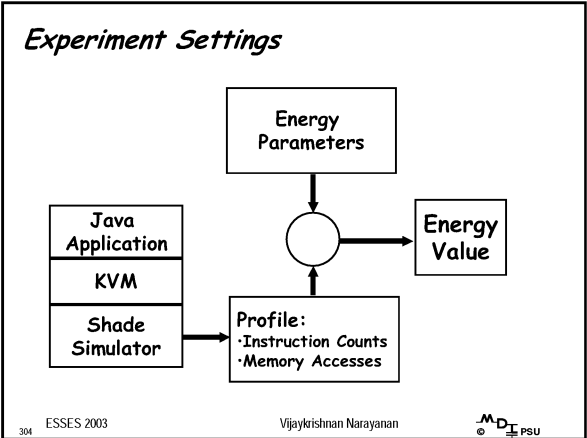
### GC Controlled Bank Turn Off

	BANK1	BANK2	BANK3	BANK4	
T=0	OFF	OFF	OFF	OFF	
T=1	A	OFF	OFF	OFF	
T=50	A B	C D	E F	G	
T=100	A B	C D	E F	G	
T=200	A	C D	OFF	G	GC
T=500	A	C D	OFF	G	
T=800	A C G	OFF	OFF	G	GC

Legend:   
 [Light Gray] Live   
 [Dark Gray] Garbage

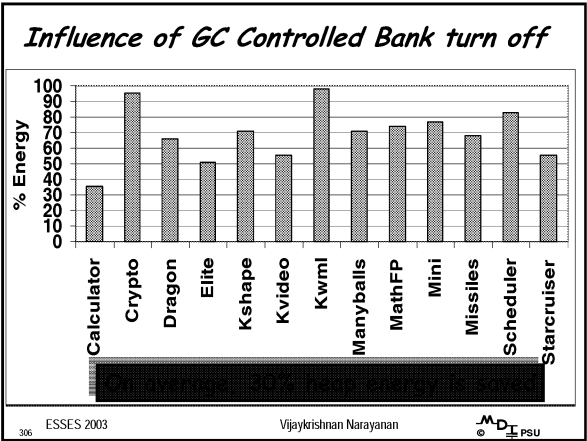
302 ESSES 2003 Vijaykrishnan Narayanan © M.D. PSU

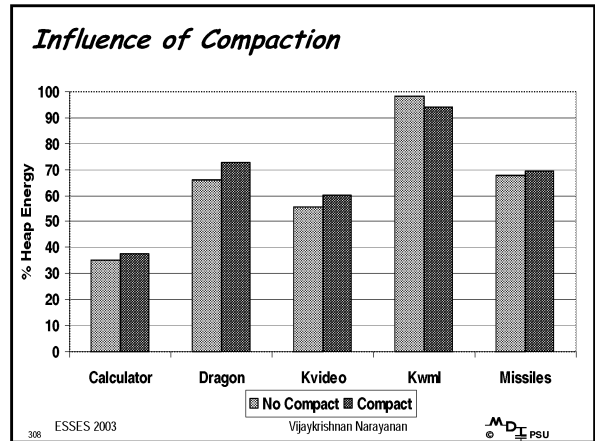
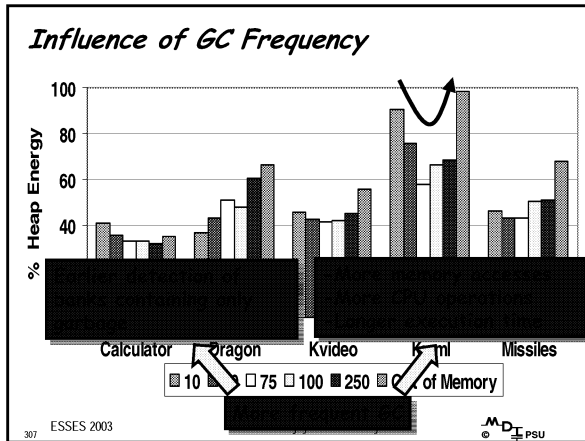
## Tuning KVM's Garbage Collection



### Benchmarks

Application	Description	Application	Description
Crypto	Cryptograph Lib	MathFP	Fixed-point math lib
Calculator	Arithmetic calculator	Kwml	WML browser
Kshape	E-map on KVM	Scheduler	Personal scheduler
Elite	3D rendering	Kvideo	KPG (MPEG for KVM) decoder
ManyBalls	Game	Missiles	Game
Dragon	Game	StarCruiser	Game



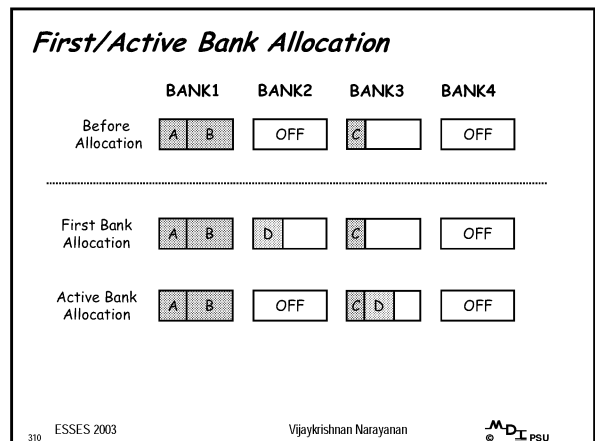


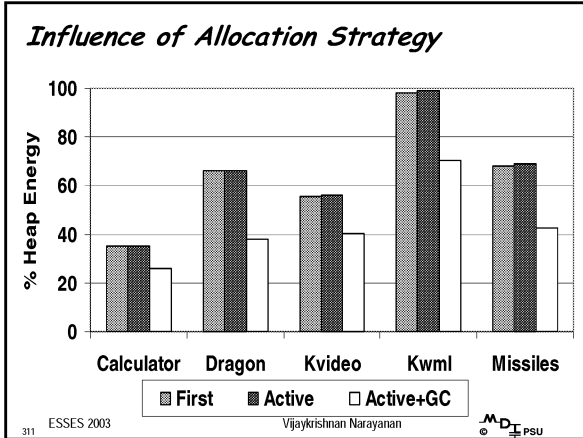
### Influence of Allocation Strategy

Allocation policy of objects also influences energy consumption

- **First Bank:** Allocate objects from the first bank having enough space
- **Active Bank:** Try to allocate objects from currently active banks. If fails, turn on a new bank
- **Active+GC:** In addition to *Active*, perform GC to find space in active banks before turning on new bank

309 ESSES 2003 Vijaykrishnan Narayanan © M.D. PSU





### Conclusions

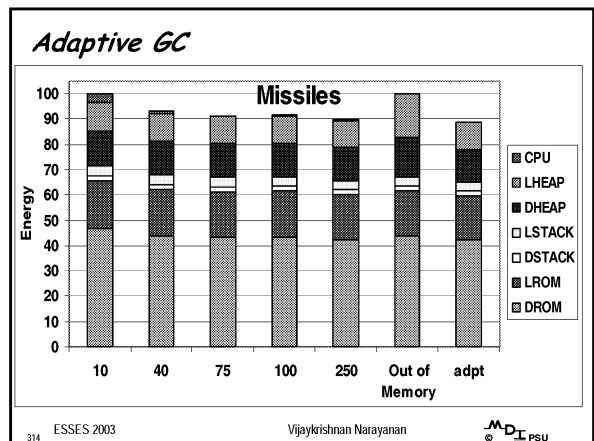
- ◆ Banked memory with GC controlled turn off reduces heap energy (30% on average)
- ◆ GC invocation frequency critically impacts system energy (optimal frequency depends on the application)
- ◆ Object allocation and compaction schemes affect the potential of memory bank turn-off
- ◆ Strategies can be applied to design energy aware memory management software

312 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU

### OtherWork

- ◆ Adaptive GC
  - Automatically find out the optimal GC frequency
- ◆ Object Collocation
  - Put objects having same life time in a same bank
- ◆ More memory bank states
  - Active, Stand-by, Sleep, Power-off

313 ESSES 2003 Vijaykrishnan Narayanan © M D I PSU



*Conclusion*

- ◆ Energy optimization needs a holistic perspective to optimization
  - Include impact on other components
  - Needs to span from circuit to software
  - Tools are critical in evaluating and enabling optimizations

*Thank you!*

Sponsored by:



# **System-level Energy Optimization**

**Naehyuck Chang**

School of Computer Science & Engineering  
Seoul National University, Korea

Sweden, July 14 - October 10,  
2003

European Summer School on Embedded Systems

# System-level Energy Optimization

Naehyuck Chang

Assistant Professor

School of Computer Science and Engineering

Seoul National University

[naehyuck@snu.ac.kr](mailto:naehyuck@snu.ac.kr)

SEOUL NATIONAL UNIVERSITY

## Lecture Outline

- Day 1 (July 28)
  - High-fidelity system-wide energy estimation of embedded systems
- Day 2 (July 29)
  - Low-power LCD systems
  - SEE Web and experiments



# High-fidelity system-wide energy estimation of embedded systems

Naehyuck Chang  
Assistant Professor  
School of Computer Science and Engineering  
Seoul National University  
[naehyuck@snu.ac.kr](mailto:naehyuck@snu.ac.kr)

SEOUL NATIONAL UNIVERSITY

## Agenda

- Introduction
- Research perspective in SNU
- In-house energy measurement tools
- In-house integrated energy estimation tool
- Applications
  - Energy characterization of ARM7TDMI
  - SDRAM memory system optimization
  - Where the power goes?
- On-going project
- Conclusions

# Introduction

- Goal of engineering
  - Design and implementation
- Invention versus innovation
  - No more breakthrough invention
- Innovation is better optimization
  - Traditional metrics
    - Performance
    - Design
    - Volume and weight
    - Cost
    - You name it
  - Modern performance indices
    - Energy
    - Power (heat)

# Introduction

- Characteristics of general purpose systems
  - Applications are not defined at the time of design
  - Layered optimization
    - Each layer is not supposed to be optimized adjacent layers
    - Isolated design layers
- Characteristics of embedded systems
  - Dedicated applications
  - Layer-through optimization
    - Software  $\leftrightarrow$  system hardware  $\leftrightarrow$  devices
    - Codesign

# Introduction

- Design of modern embedded systems
  - Component-based (IP-based) design
- Performance and energy optimization
  - Low-level approach
  - System-level approach
  - Application software-level approach
- Topics of interest
  - System-level energy optimization

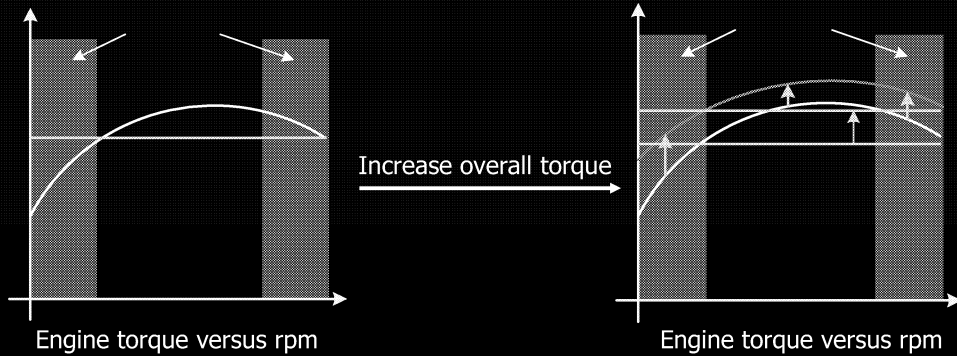
# Introduction

- City bus service example
  - Goal → achieve more gas mileages
  - Low-level approach
    - Application independent approach
      - swap the engine with more gas-mileage efficient one
  - System-level approach
    - Application-dependent approach
      - tune the engine and the transmission for desired route
  - Application-level approach
    - Application-oriented approach
      - develop a gas-mileage efficient driving method

# Introduction

## City bus service example: low-level approach

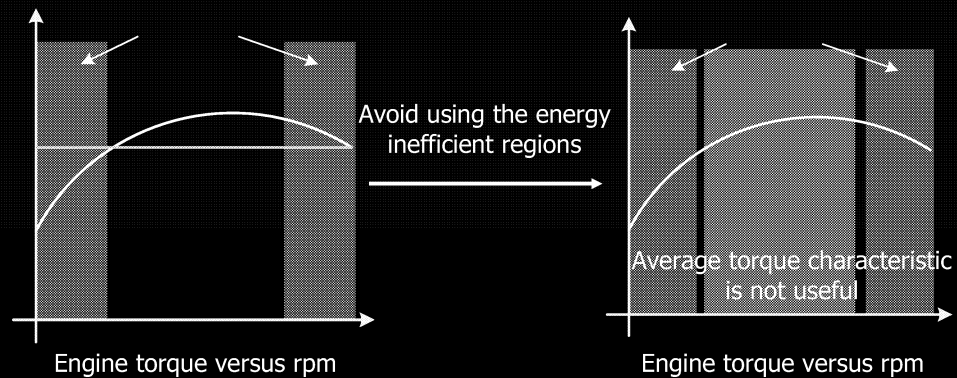
- No information of the route and the schedule
- Upgrade the engine



# Introduction

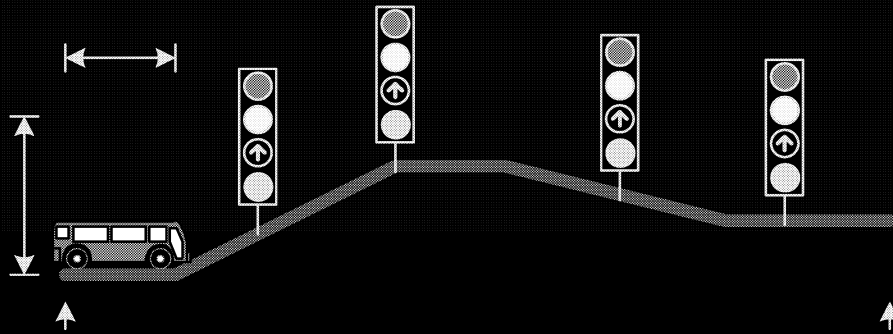
## City-bus service example: system-level approach

- Keep the engine but avoid using the energy efficient regions
- Tune the transmission shift position and/or the final gear ratio based on the route of the buses



# Introduction

- City bus service example: application-level approach
  - No hardware change
  - Develop the optimal scheduling of the speed profile



# Introduction

- City bus service example
  - Low-level approach
    - Engine upgrade
    - Enhancement of efficiency of the hardware components
      - CPU or components upgrade
  - System-level approach
    - Transmission shift policy optimization
    - Control logic (policy) optimization
      - avoiding bad usage of the components
    - Operating system and middleware optimization
  - Application-level approach
    - Efficient driving schedule
    - Task scheduling, power management, algorithm optimization, etc.

# Introduction

- High-level (system-level) power optimization
  - Given IPs
  - Given COTS chips
  - System configuration, architecture and policy, device control, system software and application software
- Example
  - Within a given silicon space, order priority of the IPs for the lowest power consumption for a task
    - Size of I\$, and D\$, MMU, bus structures, and so on
  - With a given application, find the optimal memory clock frequency for the lowest power consumption.
    - Given CPU clock, caches and the bus structure

# Introduction

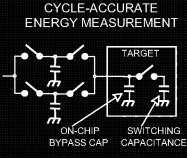


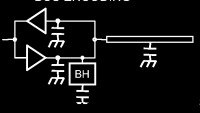

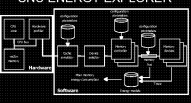
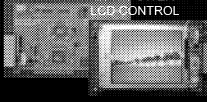

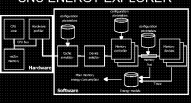
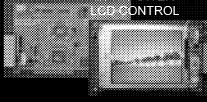


- Power versus energy
  - Power
    - Energy consumption per unit time
    - Instantaneous values
  - Energy
    - Integration of power over time
  - Power is often useful when it does not fluctuate
  - Energy is more precise representation if
    - Time unit is enough short
    - Power does fluctuate

# Introduction

- Low-power versus low-energy
  - The total energy consumption to complete a given task is meaningful
  - Low power is easily achievable as time is longer
  - Low power is also important to reduce
    - Device rating
    - Heat
    - Volume
  - Low power often mean low energy
- We go by low energy!

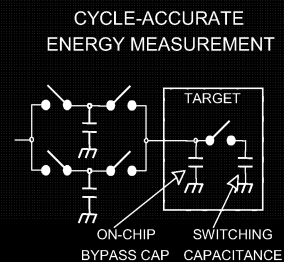
# Questions?

# Research perspective of SNU

1999	2000	2001	2002	2003
<p>CYCLE-ACCURATE ENERGY MEASUREMENT</p>  <p>ON-CHIP BYPASS CAP    SWITCHING CAPACITANCE</p> <p>IEE ELECTRONICS LETTERS 2000</p>	<p>SNU ENERGY SCANNER (CYCLE-ACCURATE ENERGY CHARACTERIZER)</p>  <p>DAC UBOTH 2000</p> <p>ARM7 ENERGY ANALYSIS</p>  <p>ISLPED 2000 IEEE TVLSI 2002</p> <p>BUS MODELS &amp; WEIGHT-BASED BUS ENCODING</p>  <p>DAC 2000</p>	<p>SNU ENERGY SCANNER FOR ARM7TDMI</p>  <p>IEEE DESIGN &amp; TEST OF COMPUTERS 2002</p> <p>DAC UBOTH 2001</p> <p>DATE UBOTH 2001</p> <p>SNU ENERGY EXPLORER</p>  <p>DAC UBOTH 2002</p> <p>LOW-POWER LCD CONTROL</p>  <p>ISLPED 2002</p> <p>SNU ENERGY SCANNER FOR FPGAS</p>  <p>DAC UBOTH 2002</p> <p>ISQED 2003</p>	<p>IEEE DESIGN &amp; TEST OF COMPUTERS 2002</p> <p>DAC UBOTH 2001</p> <p>DATE UBOTH 2001</p> <p>SNU ENERGY EXPLORER</p>  <p>DAC UBOTH 2002</p> <p>LOW-POWER LCD CONTROL</p>  <p>ISLPED 2002</p> <p>SNU ENERGY SCANNER FOR FPGAS</p>  <p>DAC UBOTH 2002</p> <p>ISQED 2003</p>	<p>LOW-POWER NON-VOLATILE MEMORY SYSTEMS</p> <p>ISLPED 2003</p> <p>MEMORY SYSTEM AWARE DVS</p> <p>NAT SOC PERFORMANCE ANALYSIS &amp; ENHANCEMENT</p> <p>DBLS (DYNAMIC BACKLIGHT LUMINANCE SCALING)</p>  <p>ISLPED DESIGN CONTEST 2002 ENTRY #1</p> <p>LPBP (LOW-POWER J2ME)</p> <p>ISLPED 2003</p> <p>COMPRESSED FRAME BUFFER</p> <p>POWER OPTIMIZATION OF FPGA DESIGN</p>
<p>Low-power System Modeling and I/O Systems P17</p>	<p>SEOUL NATIONAL UNIVERSITY</p>			<p>Naehyuck Chang naehyuck@snu.ac.kr</p>

# In-house energy measurement tools

- Cycle-accurate energy characterization (IEE Electronics Letters and IEEE TVLSI)
  - Ultimate characterization for synchronous state machines
  - Based on cycle-accurate energy estimation or measurement
  - Technical barrier in cycle-accurate energy measurement
    - Difficult to measure by conventional equipment
- Applications of cycle-accurate energy measurement
  - Data-driven energy characterization
  - Control-driven energy characterization
  - Event-driven asynchronous device characterization

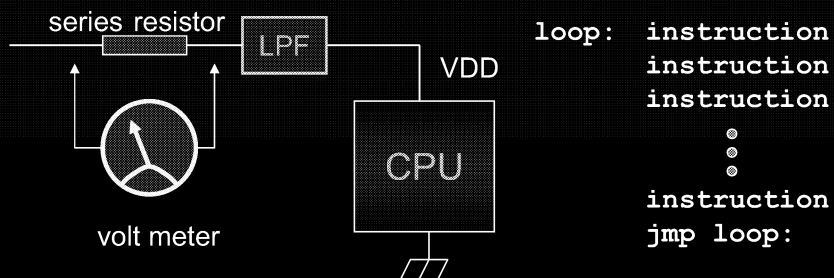




# In-house energy measurement tools

## ■ Use of a digital multimeter

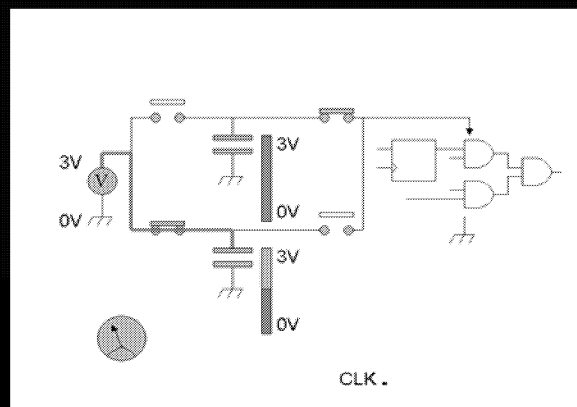
- Designed for DC or sinusoidal current and voltage.
- True RMS version for periodic-waveform current and voltage that have limited bandwidth: tens KHz order.



# In-house energy measurement tools

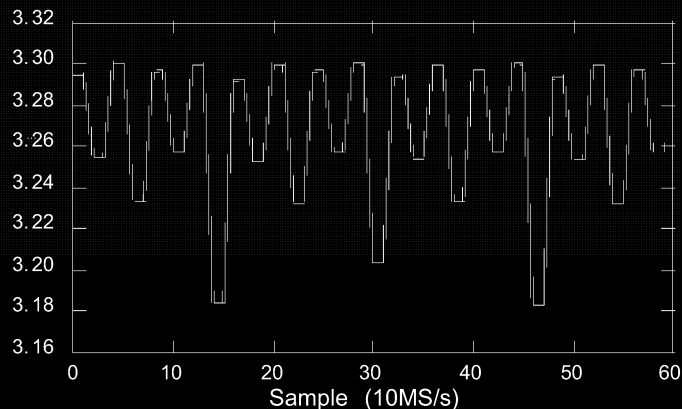
## ■ Cycle-accurate energy measurement (IEE Electronics Letters 2000)

- Principle of operation



## In-house energy measurement tools

- Cycle-accurate energy measurement
  - Example: a high-level energy characterization
    - 4-bit binary counter (74HC393)



## In-house energy measurement tools

- Cycle-accurate energy measurement
  - Example: a high-level energy characterization
  - 4-bit binary counter (74HC393)
  - Average power supply current with a multimeter
    - 1.68mW (@ 5MHz), 3.39mW (@10MHz)
  - Cycle-accurate energy consumption by the switched capacitors with  $C = 3.200\text{pF}$ .
    - 1-bit change (0000 to 0001, 0010 to 0011, etc.)  $\rightarrow 0.21\text{nJ}$
    - 2-bit change (0001 to 0010, 0101 to 0110, etc.)  $\rightarrow 0.35\text{nJ}$
    - 3-bit change (0011 to 0100, 1011 to 1100)  $\rightarrow 0.49\text{nJ}$
    - 4-bit change (0111 to 1000)  $\rightarrow 0.62\text{nJ}$
    - The equivalent average power consumption  $\rightarrow 1.64\text{mW}$  (@5MHz),  $3.26\text{mW}$  (@10MHz)

# In-house energy measurement tools

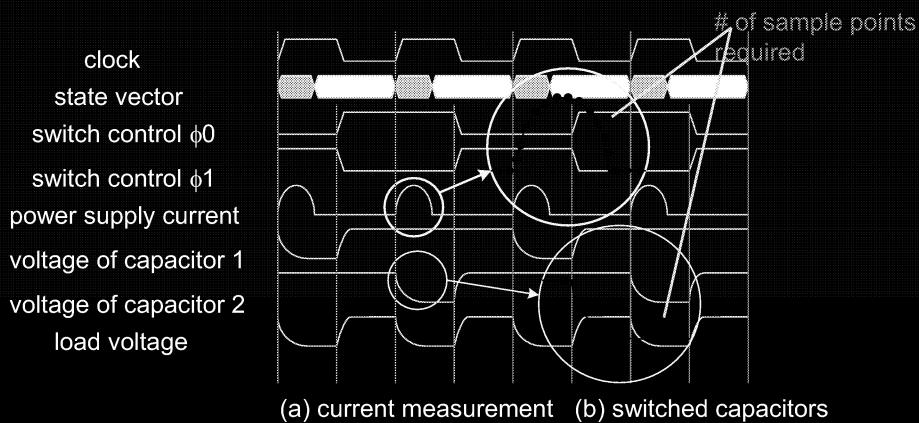
- Proper tools of high-level power/energy optimization
  - Commercial energy information

## DC AND OPERATING CHARACTERISTICS

Item	Symbol	Test Conditions	Min	Typ	Max	Unit	
Input leakage current	$I_{II}$	$V_{IN}=V_{SS}$ to $V_{CC}$	-1	-	1	$\mu A$	
Output leakage current	$I_{LO}$	$\overline{CS1}=V_{IH}$ or $CS2=V_{IL}$ or $\overline{WE}=V_{IL}$ , $V_{IO}=V_{SS}$ to $V_{CC}$	-1	-	1	$\mu A$	
Operating power supply current	$I_{CC}$	$I_{IO}=0mA$ , $\overline{CS1}=V_{IL}$ , $CS2=V_{IH}$ , $V_{IN}=V_{IH}$ or $V_{IL}$ , Read	-	5	10	mA	
Average operating current	$I_{CC1}$	Cycle time=1 $\mu s$ , 100% duty, $I_{IO}=0mA$ , $\overline{CS1s}=0.2V$ , $CS2s=V_{CC}-0.2V$ , $V_{INs}=0.2V$ or $V_{INs}=V_{CC}-0.2V$	Read	-	2	5	mA
			Write	-	20	35	
	$I_{CC2}$	Cycle time=Min, 100% duty, $I_{IO}=0mA$ , $\overline{CS1}=V_{IL}$ , $CS2=V_{IH}$ , $V_{IN}=V_{IL}$ or $V_{IH}$	-	45	60	mA	
Output low voltage	$V_{OL}$	$I_{OL}=2.1mA$	-	-	0.4	V	
Output high voltage	$V_{OH}$	$I_{OH}=1.0mA$	2.4	-	-	V	
Standby Current (TTL)	$I_{SB}$	$\overline{CS1}=V_{IH}$ , $CS2=V_{IL}$ , Other Input= $V_{IL}$ or $V_{IH}$	-	-	3	mA	
Standby Current (CMOS)	KM681000CL	$\overline{CS1s}=V_{CC}-0.2V$ , $CS2s=V_{CC}-0.2V$ or $CS2s=0.2V$ Other Input = 0~ $V_{CC}$	Low Power	-	1	50	$\mu A$
	KM681000CL-L		Low Low Power	-	0.3	10	
	KM681000CLI		Low power	-	1	50	
	KM681000CL-L		Low Low Power	-	0.3	15	

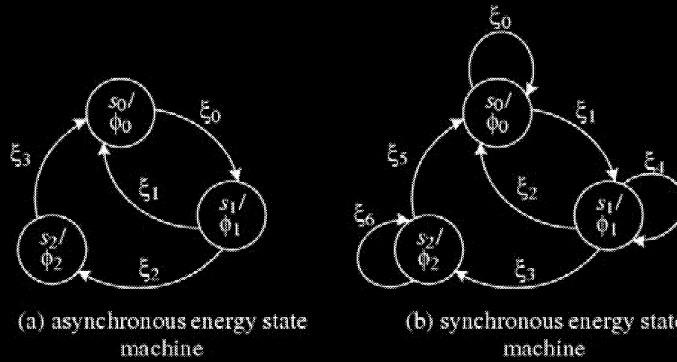
# In-house energy measurement tools

- Cycle-accurate energy measurement
  - Comparison with voltage and current measurement



# In-house energy measurement tools

- Energy consumption by state transitions (DAC2002, ACM TECS2003)

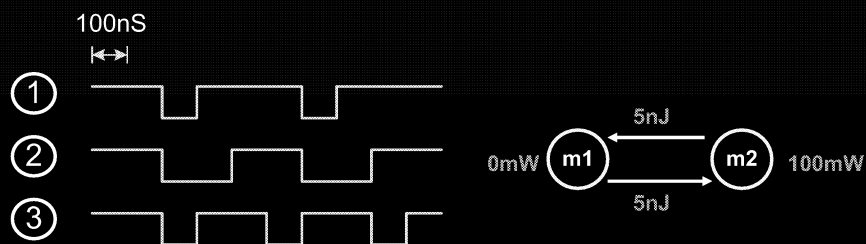


# In-house energy measurement tools

- Comparison with previous models

- Unit energy per access
- Power/mode
- Energy state machine

①	50mW	②	50mW	③	66mW
①	25mW	②	50mW	③	33mW
①	50mW	②	66mW	③	66mW

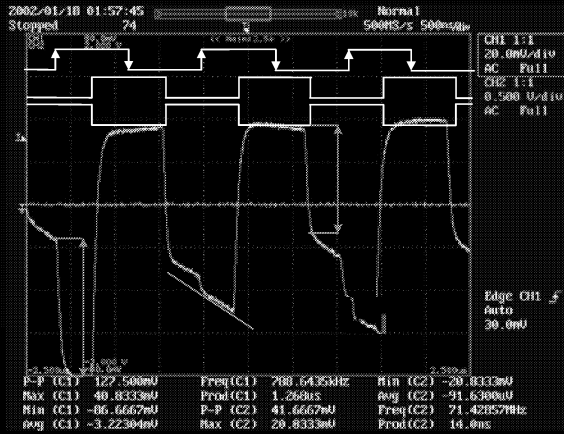
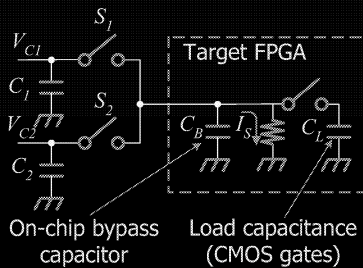


# Cycle-accurate energy measurement

- Principle of operation (TVLS2002, ISQED 2003)

- Energy stored in a capacitor
- Charge sharing

$$E_{C1} = \frac{1}{2} C_1 V^2, Q_{C1} = C_1 V_{C1}$$

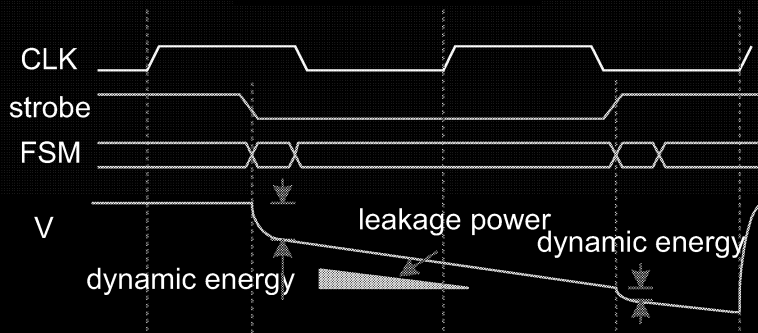


# In-house energy measurement tools

- Cycle-accurate energy measurement

- Leakage energy consumption is denoted by

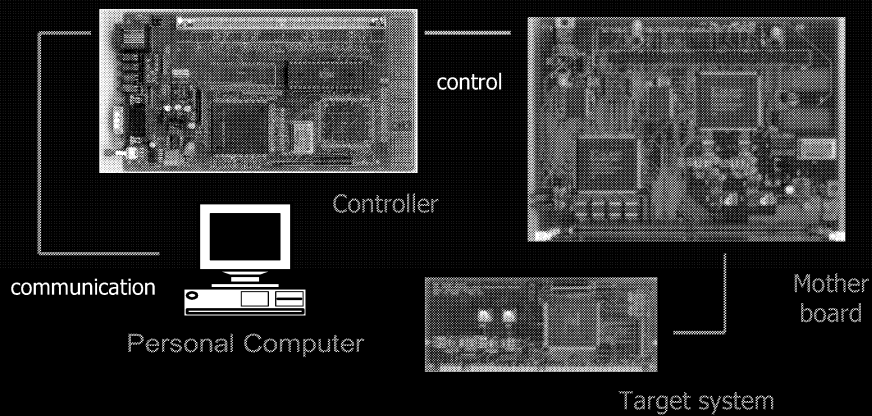
$$\Phi_j = \frac{C_x v_s^2(t + \Delta t) - v_s^2(t)}{\Delta t}$$



# Questions?

# In-house energy measurement tools

- SEC (SNU Energy Characterizer)  
(DAC Ubooth 2001, 2002)

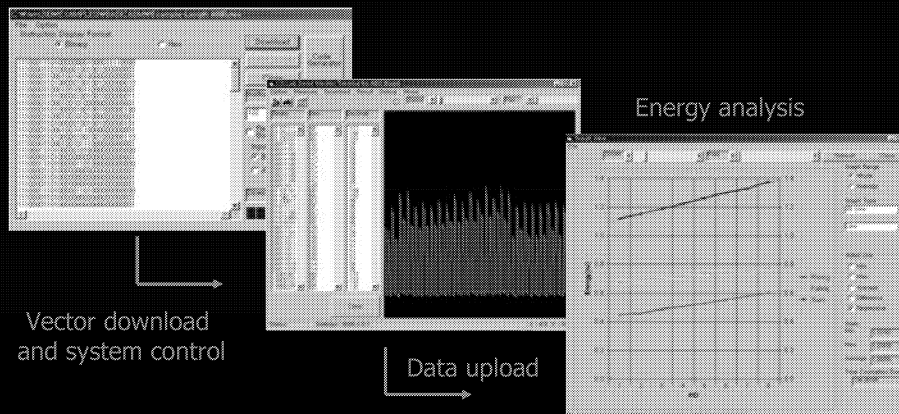


# In-house energy measurement tools

- SEC (SNU Energy Characterizer) V3.0
  - Tools for energy characterization of
    - Memory devices
    - Peripheral devices
  - Versions of SEC
    - Universal PCB version to verify the idea
      - 74HC393
      - Use of a DSO
    - SEC for the ARM7TDMI
      - Serial interface
      - 64 word vector length

# In-house energy measurement tools

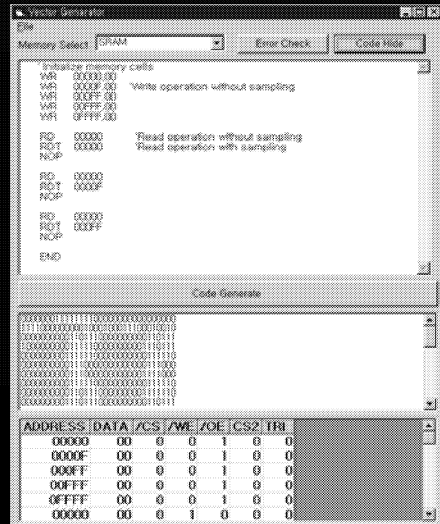
- SEC (SNU Energy Characterizer)
  - Software support for SEC tools



# In-house energy measurement tools

## SEC (SNU Energy Characterizer)

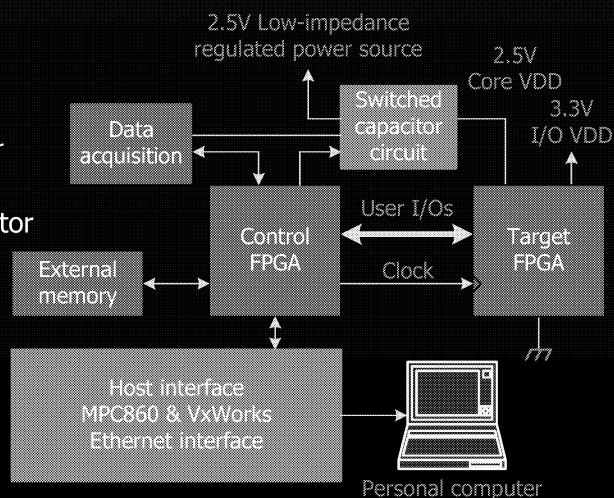
- Software support for SEC tools
  - Macro assembler for memory devices
  - Device dependent timing generator
    - SRAM
    - DRAM
    - SDRAM



# In-house measurement tool

## SECF (ISQED 2003)

- 10-bit A/D converter @50Ms/s
- Computer controlled
  - PC Windows-based measurement manager
  - Ethernet interface
- Hardware vector generator
  - 256KB vector memory
  - FPGA controlled
  - Programmed via network w/o a download cable

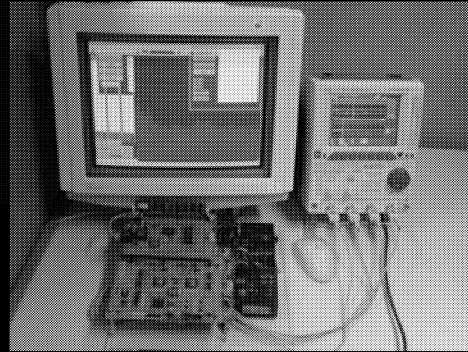
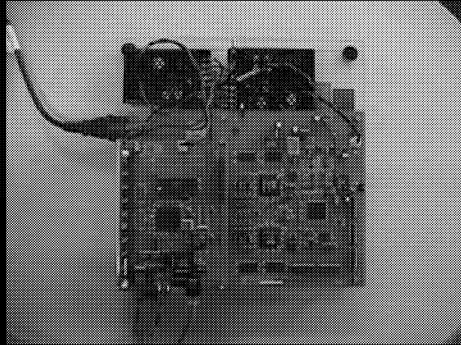




# In-house measurement tool

## ■ SECF 2.0

- Target FPGA: Xilinx Spartan II XC2S50TQ144
- Vector generator: Xilinx Spartan II XC2150FG456
- Data acquisition: Xilinx Spartan II XC2150FG456



# In-house energy measurement tools

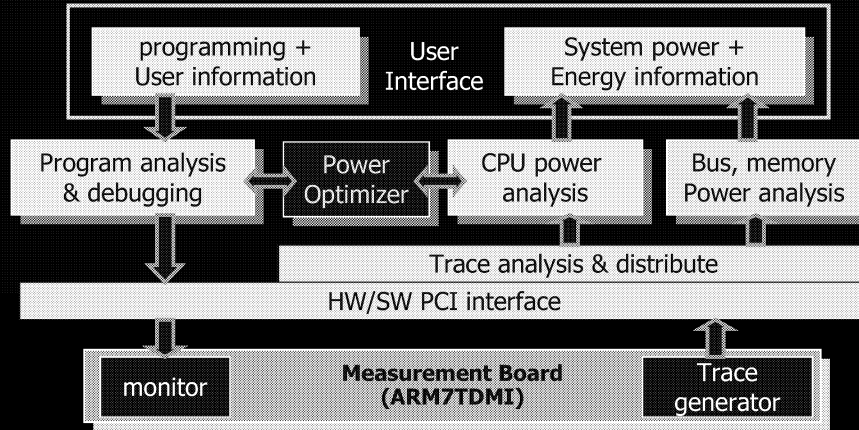
## ■ SES (SNU Energy Scanner) (IEEE Design & Test of Computers 2002)

- Motivation
  - Tool for high-level software energy optimization
- Dedicated tool for ARM7TDMI
- Large memory for real application programs
  - Up to 8M words
- Fast data communication
  - PCI local bus interface
  - Real-time acquisition for continuous execution
- Upgraded accuracy
  - 10-bit A/D converter
- Popular environment
  - Powered by desktop Linux

# In-house energy measurement tools

## SES (SNU Energy Scanner)

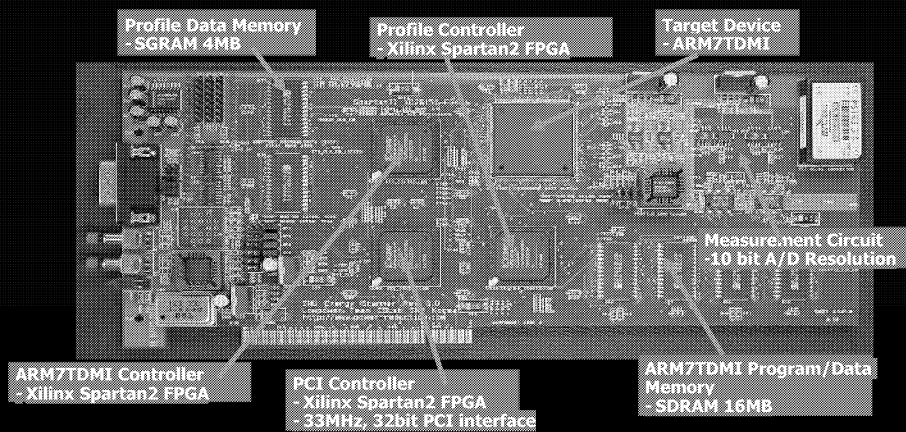
### - SES architecture



# In-house energy measurement tools

## SES (SNU Energy Scanner)

### - SES tool hardware



# In-house energy measurement tools

## ■ SES (SNU Energy Scanner)

### – SES software support

#### ▪ Profile data form

data bus    address bus    control signals    ADC value

#### ▪ Monitor program running on Linux over PCI local bus interconnection

#### ▪ Function call support

#### ▪ GUI interface

Energy consumed  
at falling edge of  
MCLK

Energy consumed  
at rising edge of  
MCLK

0x000000ec	0xe3a01000	0x781f9378	0x601fa31f	554.7660pJ	425.5536pJ
0x00000118	0xe3a01000	0x78200371	0x601fc2ef	534.4873pJ	353.4322pJ
0x0000011c	0xe3a00701	0xf81f734c	0xe01f42e8	494.6298pJ	355.0500pJ
0x00000120	0xe5801000	0xf81f8356	0xe01fa30a	507.5128pJ	395.3080pJ
0x00000124	0xe5801004	0x781fc35c	0xe01f830a	510.2559pJ	398.2391pJ
0x00000128	0xe5801008	0x781fb340	0xe01f42ef	471.5374pJ	365.1574pJ

# In-house energy estimation tool

## ■ SEE (SNU Energy Explore)

### – Motivation

#### ▪ Tool for component-based system-level low-power design

### – System components

#### ▪ CPU core

#### ▪ Cache

#### ▪ Main memory

#### ▪ Auxiliary memory

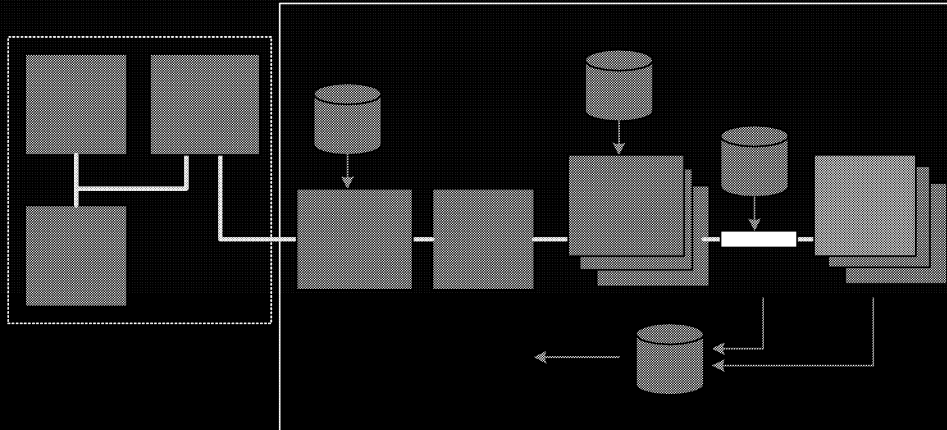
#### ▪ Communication controllers

#### ▪ LCD display

### – Based on SEC and SES hardware

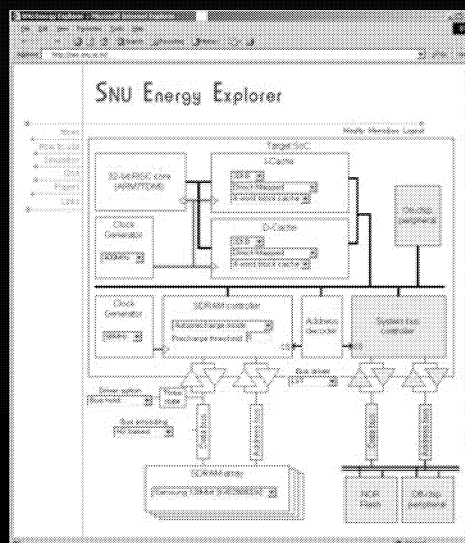
# In-house energy estimation tool

- SEE (SNU Energy Explore)
  - Fully integrated system-level energy estimation tool



# In-house energy estimation tool

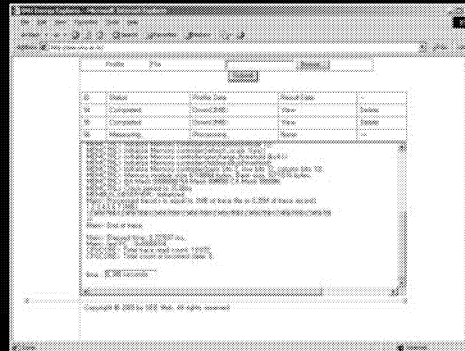
- SEE Web
  - SEE connected to Web interface
  - <http://see.snu.ac.kr>
  - Cache energy model based on XCACTI
  - GUI support
  - Simple Web environment for standard Web browsers
  - Now open to public!



# In-house energy estimation tool

## SEE Web

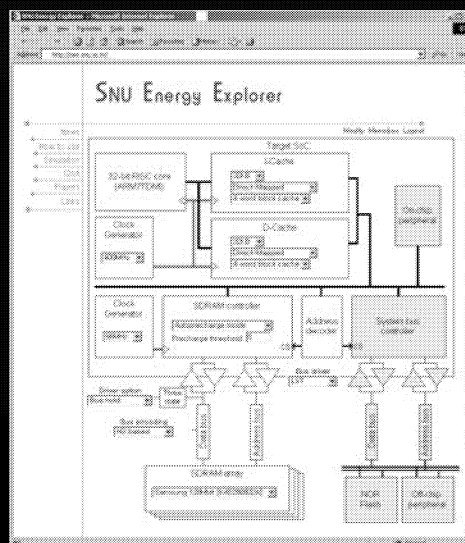
- Configuration of a target hardware
- Prepare a binary code
- Upload the file and execute SEE Web
- Verify the output



# In-house energy estimation tool

## Configuration of a target hardware

- Processor clock
- I-cache and D-cache
  - Capacity
  - Associativity
  - Line size
- SDRAM controller
  - Auto-precharge
  - Active-page
  - Delayed precharge
- SDRAM
  - Clock
- Bus interface
  - Bus Hold
  - Bus encoding





# Questions?

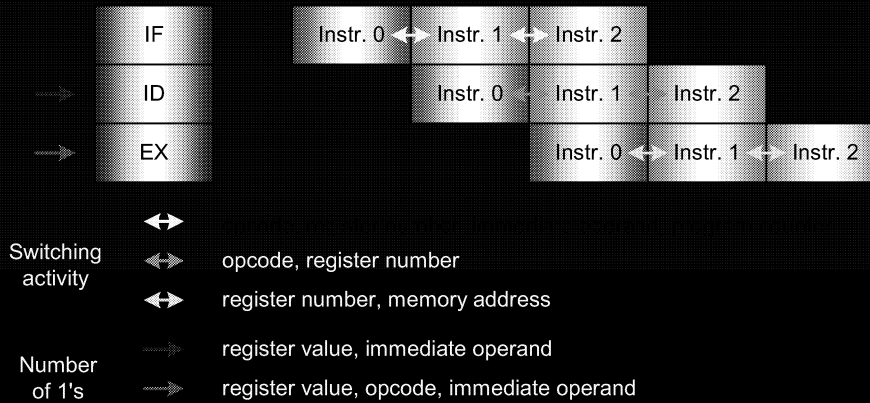
## Energy characterization of ARM7TDMI

- Energy characterization (IEEE TVLSI 2002, ISLPED 2000)
  - Goal
    - Energy optimization by software: instruction and data
    - Characterization of energy-sensitive factors
  - Energy components
    - Controllable
      - HDD (transition) and WDD (# of 1)
    - Uncontrollable
      - Leakage and CD (Common-mode dynamic, state transition)
    - CD energy
      - Default energy includes invisible parts from software: clock drivers, etc.
      - HDD = Max(dynamic) – Min(dynamic)
      - Common-mode energy = Min(dynamic)

$$\begin{array}{|c|} \hline \text{default} \\ \text{energy} \\ \hline \end{array} + \begin{array}{|c|} \hline \text{variable} \\ \text{energy} \\ \hline \end{array} = \begin{array}{|c|} \hline \text{min}(op_1, \dots, op_n) \\ + \text{relative energy} \\ \hline \end{array}$$

# Energy characterization of ARM7TDMI

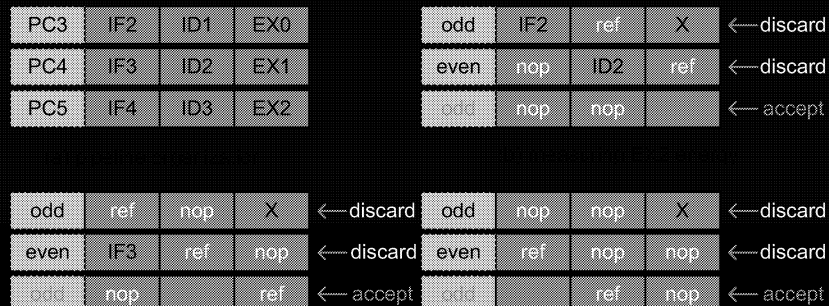
## Energy-sensitive factors of ARM7TDMI RISC core



# Energy characterization of ARM7TDMI

## Energy characterization of a pipelined microprocessor

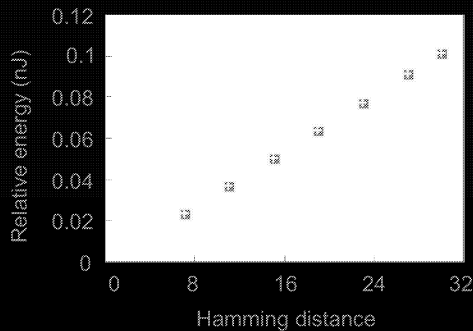
- Keep the same condition of other factors of no interest





# Energy characterization of ARM7TDMI

- Low-level (data-driven) characterization
  - PC stage energy behavior
    - Hamming distance between current and previous IF address



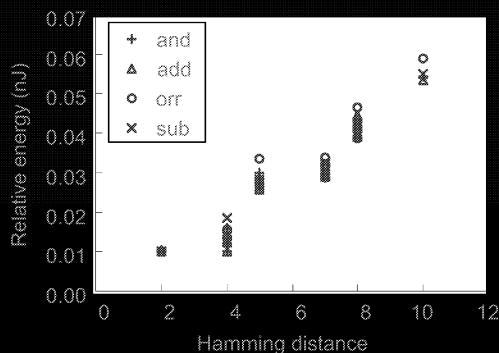
```
0ffffffc    add r0 r1 r2
10000000    mov r2 r3
```

```
10000000    add r0 r1 r2
10000004    mov r2 r3
```

24bit more address bit changes  
result in 0.082nJ more energy  
(7.2% of the total core energy)

# Energy characterization of ARM7TDMI

- Low-level (data-driven) characterization
  - EX stage energy behavior (1)
    - Hamming distance between current and previous register IDs.



```
10000000    add r0 r0 r1
10000004    sub r7 r7 r6
```

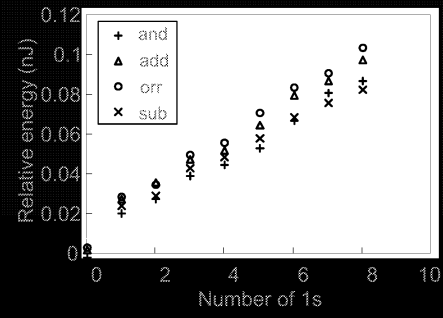
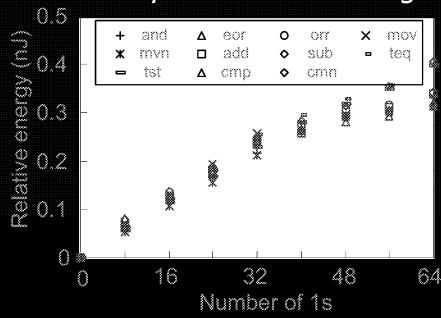
```
10000000    add r0 r0 r1
10000004    sub r6 r6 r2
```

10bit more register ID bit changes  
result in 0.06nJ more energy  
(5.2% of the total core energy)

# Energy characterization of ARM7TDMI

## Low-level (data-driven) characterization

- EX stage energy behavior (2)
  - Number of 1's in the current data value
  - Dynamic CMOS configuration



# Energy characterization of ARM7TDMI

## Previous two dimensional characterization

- Base cost + inter-instruction cost

	inst 1	inst 2	inst 3	inst 4	inst 5	inst 6
inst 1	base 1					
inst 2		base 2				
inst 3			base 3			
inst 4				base 4		
inst 5					base 5	
inst 6						base 6

# Energy characterization of ARM7TDMI

- Average variable cost between two instructions

- Fetch stage



- Decode stage



- Execution stage



- Suitable for average power estimation

# Energy characterization of ARM7TDMI

- Average variable cost between two instructions

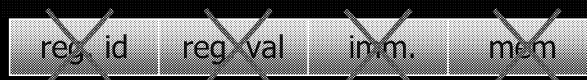
- Fetch stage



- Decode stage



- Execution stage



- Suitable for average power estimation

# Energy characterization of ARM7TDMI

## ■ Suggested parametric model

–  $H$ : Hamming distance,  $W$ : weight

Relative base cost ( nJ)

opcode	ID	EX
and	0.10	0.10
eor	0.22	0.10
sub	0.06	0.02
rsb	0.17	0.20
add	0.15	0.10
bic	0.00	0.00

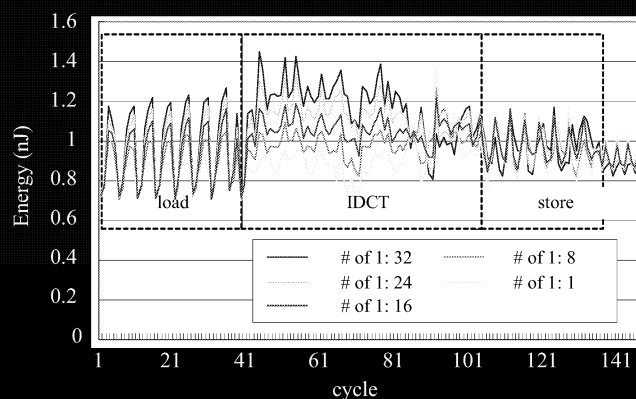
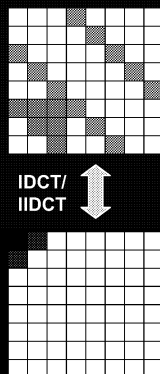
Variable cost ( pJ)

factor	IF	stage	ID	stage	EX	stage
	E	%	E	%	E	%
opcode	$4.5H$	$1.58$	-	$33.3$	-	$40.4$
reg #	$2.5H$	$2.63$	$7.5H$	$7.9$	$5.4H$	$5.7$
reg_val.	$0$	$0$	-	-	$6.4W$	$37.6$
pc	$3.4H$	$9.6$	$0$	$0$	$0$	$0$
mem.	$0$	$0$	-	-	$3.4H$	$9.6$
imm.	$6.2H$	$3.4$	$1.0H$	$7.0$	$1.13W$	$7.9$

# Energy characterization of ARM7TDMI

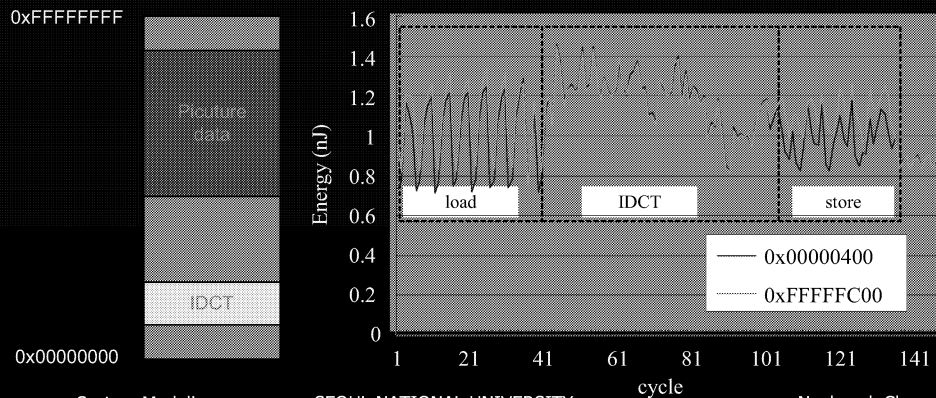
## ⦿ High-level (function-level) characterization

- ♦ IDCT routine function-level characterization
  - ♦ Energy versus the number of 1's of the input data



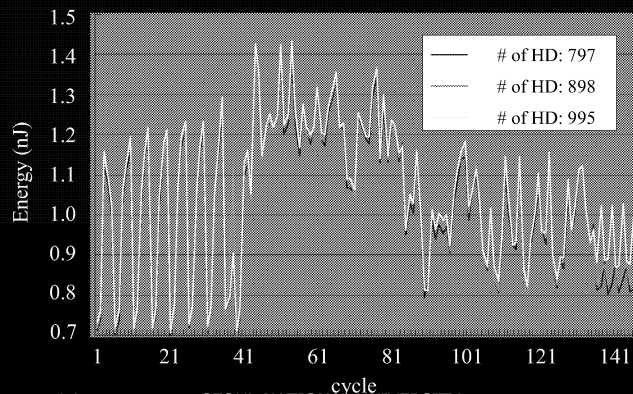
# Energy characterization of ARM7TDMI

- High-level (function-level) characterization
  - IDCT routine function-level characterization
    - Energy versus the Hamming distance between the code and the data segments



# Energy characterization of ARM7TDMI

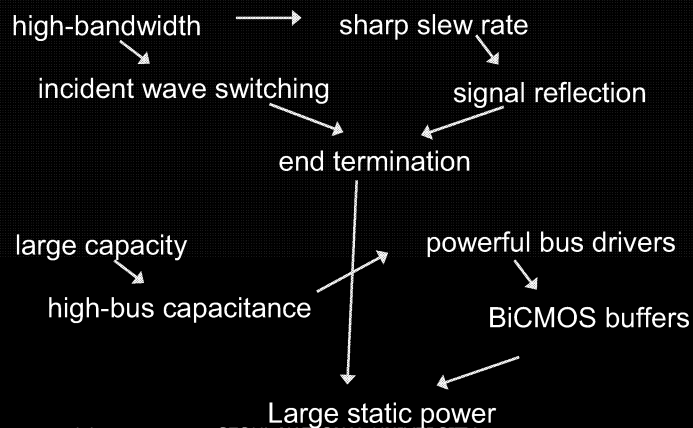
- High-level (function-level) characterization
  - IDCT routine function-level characterization
    - Energy reduction by register renaming



# Questions?

# Low-power bus encoding

- Motivation (DAC 2000)
  - High-performance, large-capacity memory systems

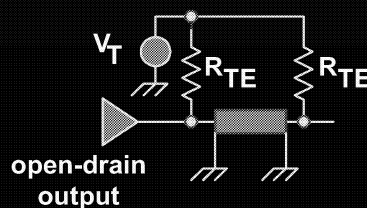


# Low-power bus encoding

- Target system: high-performance memory systems
  - High-transfer rate
    - high-performance bus specifications
  - Large-capacity
    - heavy loaded bus
    - transmission lines
  - Embedded L1 or L2 cache memories
    - Intel Pentium™ processor
  - SDRAM arrays
    - DDR SDRAM

# Low-power bus encoding

## ■ GTL+



- Processors to memory controllers
- Intel Pentium™ processor
- 3.3V supply, open-drain output end-terminated to 1.5V
- CMOS and BiCMOS versions
- Open-drain output

## bus encoding coding

### ■ LVT (ABT)

bipolar  
totem-pole  
output

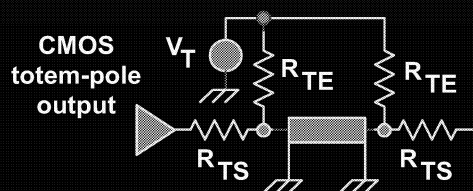


### ■ general purpose interconnection

- TTL (ABT) or low-voltage TTL (LVT) compatible
- BiCMOS technology
- bipolar totem-pole output

## bus encoding coding

### ■ SSTL\_2

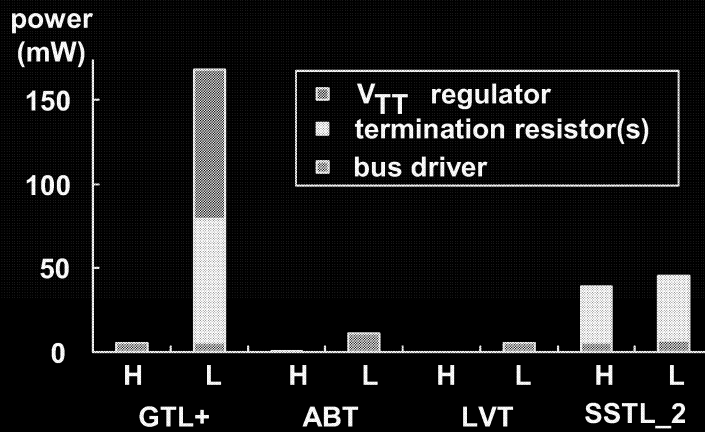


- Memory controllers to DDR SDRAM arrays
- 2.5V CMOS totem-pole terminated to 1.25V
- CMOS technology
- CMOS totem-pole



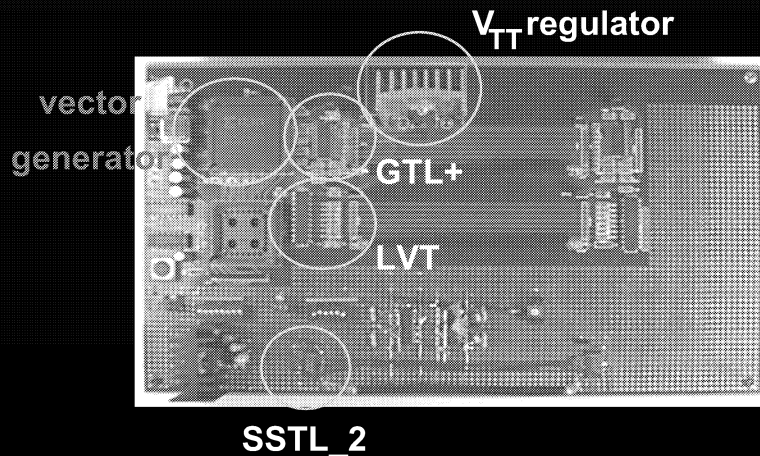
# bus encoding coding

## Static power consumption



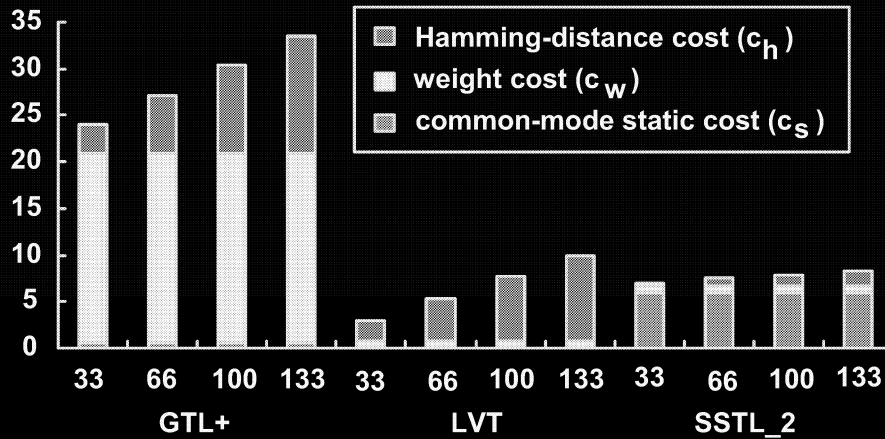
# bus encoding coding

## Prototyping for measurement



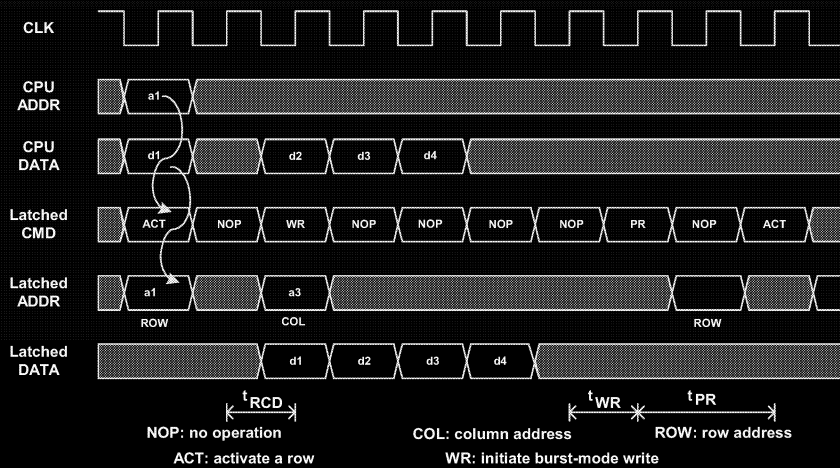
# bus encoding coding

## Power consumption coefficients



# bus encoding coding

## SDRAM burst-mode transfer



# bus encoding coding

## Original sequence

		burst sequence	d5	d1	d2	d3	d4	total
<b>data</b>		b0	1	0	1	1	0	1
		b1	1	1	0	1	1	1
		b2	1	0	1	0	0	1
		b3	1	0	0	1	0	1
		b4	1	0	0	1	0	1
		b5	1	0	1	0	1	1
		b6	1	0	1	0	1	1
		b7	1	1	1	0	0	1
<b>cost</b>		weight	6	3	4	5	5	18
		Hamming distance	3	5	7	5	5	20
		weight + Hamming distance						38

**actual energy cost**  
 18  
 28  
 46

6 →

# bus encoding coding

## Existing bus inversion decision

		burst sequence	d5	d0	d1	d2	d3	d4	d5	total
<b>data</b>		b0	0	1	0	0	1	1	1	
		b1	1	1	1	1	1	0	1	
		b2	0	1	0	0	0	1	1	
		b3	1	1	0	1	1	1	1	
		b4	1	1	0	1	1	1	1	
		b5	0	1	0	0	0	0	1	
		b6	0	1	0	0	0	0	1	
		b7	0	1	1	0	0	1	1	
<b>cost</b>		inversion (-)	1	1	1	0	1	0	1	
		weight	0	6	6	4	4	4	0	20
		Hamming distance	6	4	2	4	4	4		20
		weight + Hamming distance								40

**energy cost of original sequence**  
 18  
 28  
 46

# bus encoding coding

## Consideration of the transfer protocol

burst sequence		d5	d0	d1	d2	d3	d4	d5	total
data	b0	0	1	1	1	0	0	1	
	b1	1	1	0	0	0	1	1	
	b2	0	1	1	1	1	0	1	
	b3	1	1	1	0	0	0	1	
	b4	1	1	1	0	0	0	1	
	b5	0	1	1	1	1	1	1	
	b6	0	1	1	1	1	1	1	
	b7	0	1	0	1	1	0	1	
inversion (-)		1	1	0	1	0	1	1	
weight		0	3	3	5	5	0		16
cost	Hamming distance	3	4	2	4	5			18
	weight + Hamming distance								34
									26

energy cost of original sequence

16

18

34

26

28

46

# bus encoding coding

## Consideration of the static power

burst sequence		d5	d0	d1	d2	d3	d4	d5	total
data	b0	0	1	1	1	1	1	1	
	b1	1	1	0	0	1	0	1	
	b2	0	1	1	1	0	1	1	
	b3	1	1	1	0	1	1	1	
	b4	1	1	1	0	1	1	1	
	b5	0	1	1	1	0	0	1	
	b6	0	1	1	1	0	0	1	
	b7	0	1	0	1	0	1	1	
inversion (-)		1	1	0	1	1	0	1	
weight		0	3	3	4	4	0		14
cost	Hamming distance	3	4	7	4	4			22
	weight + Hamming distance								36
									18

energy cost of original sequence

14

22

36

18

28

44

# bus encoding coding

- Consideration of the exact power consumption model (heuristics)

burst sequence		d5	d0	d1	d2	d3	d4	d5	total
data	b0	0	1	1	1	0	1	1	
	b1	1	1	0	0	0	0	1	
	b2	0	1	1	1	1	1	1	
	b3	1	1	1	0	0	1	1	
	b4	1	1	1	0	0	1	1	
	b5	0	1	1	1	1	0	1	
	b6	0	1	1	1	1	0	1	
	b7	0	1	0	1	1	1	1	
inversion (-)		1	1	0	1	0	0	1	
weight		0	3	3	5	4	0		15
cost	Hamming distance	3	4	2	5	4			18
	weight + Hamming distance								33
									16

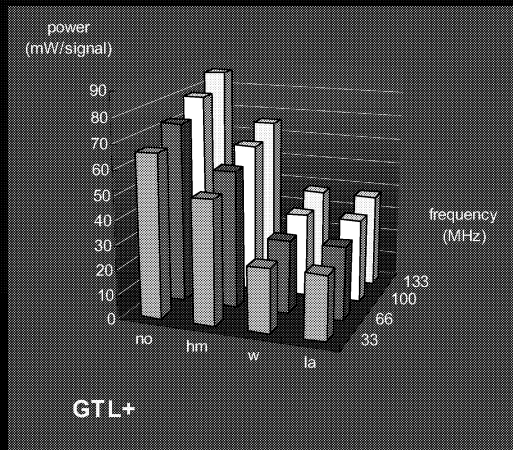
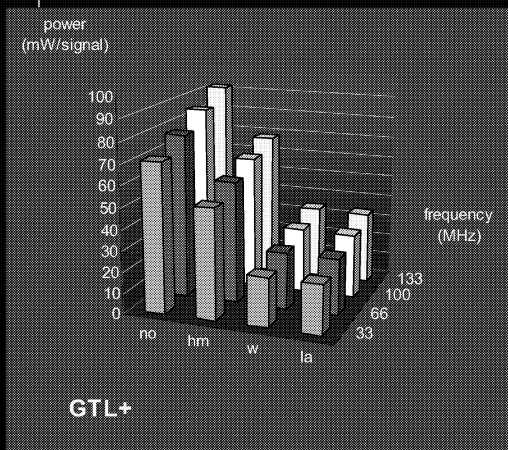
energy cost of protocol-based inversion

# bus encoding coding

- Trace-data-driven simulation

CRC

DCT

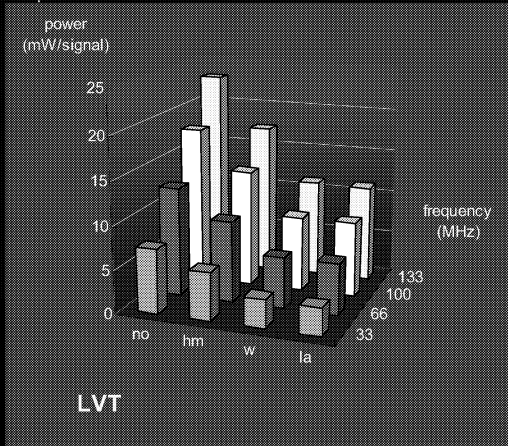


# bus encoding coding

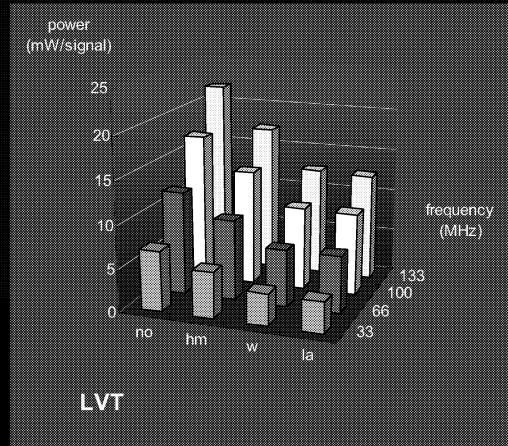
## Trace-data-driven simulation (contd.)

CRC

DCT



LVT



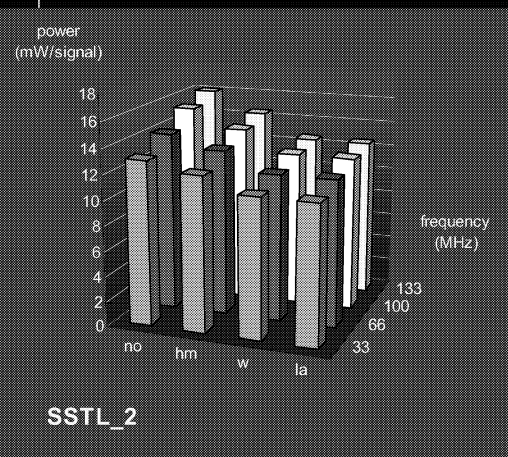
LVT

# bus encoding coding

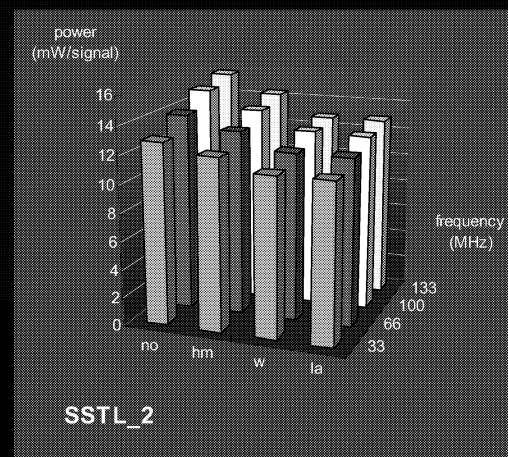
## Trace-data-driven simulation (contd.)

CRC

DCT



SSTL\_2

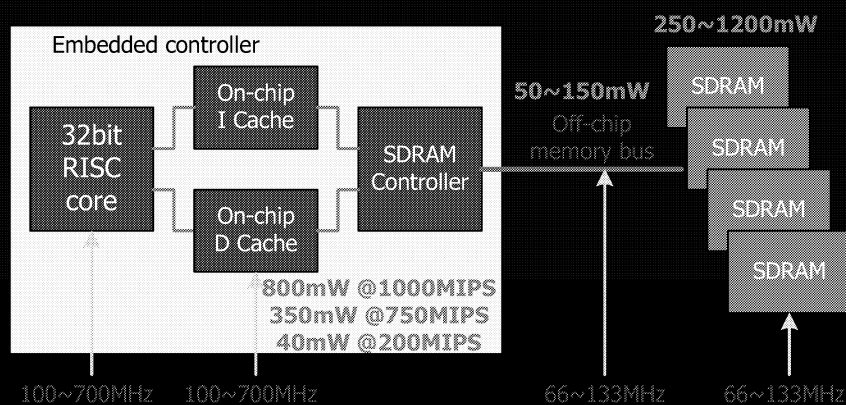


SSTL\_2

# Questions?

## SDRAM memory system optimization

- Energy portion of embedded systems (ACM TECS 2003)
  - Energy exploration and reduction of off-chip SDRAM memory systems



# SDRAM memory system optimization

- Why design space exploration?

- Globally optimal configuration

- Example 1) Small cache size?

1KB cache → 1200mW/310mJ consumed in SDRAMs

16KB cache → 300mW/40mJ consumed in SDRAMs

- Example 2) Slow down the CPU?

100MHz → 300mW/87mJ consumed in SDRAMs

1GHz → 1000mW/65mJ consumed in SDRAMs

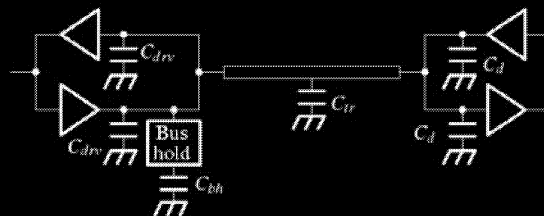
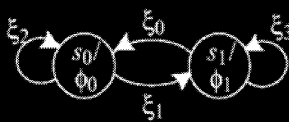
- With respect to

- System configurations
  - Applications and data sets
  - User behavior

# SDRAM memory system optimization

- Annotation of energy state machine

- Memory data bus



$$\xi_0 = \xi_1 = \frac{1}{2}(2C_{drv} + C_{bh} + C_{tr} + 2C_d)V_{dd}^2 + E_{drv}$$

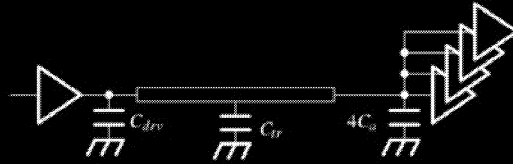
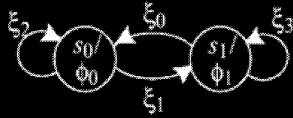
$$\Xi_{sB} = (0.55, 0.55, 0.00, 0.00), \Phi_{sB} = (0.0053\tau, 0.00) \quad (\text{unit: nJ/bit})$$



# SDRAM memory system optimization

## Annotation of energy state machine

– Memory address bus



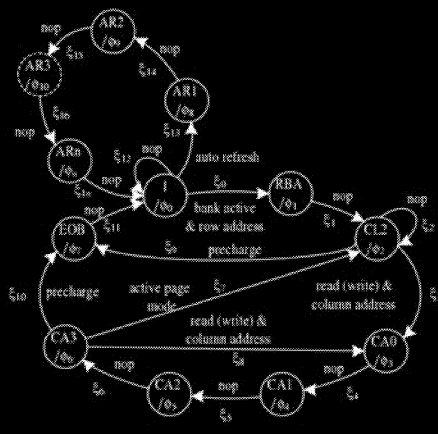
$$\xi_0 = \xi_1 = \frac{1}{2}(C_{drv} + C_{tr} + 4C_a)V_{dd}^2 + E_{drv}$$

$$\Xi_{sB} = (0.58, 0.58, 0.00, 0.00), \Phi_{sB} = (0.0053\tau, 0.00) \quad (\text{unit: nJ/bit})$$

# SDRAM memory system optimization

## Annotation of energy state machine

– SDRAM



$\xi$	Energy Cost
$\xi_0$	$\Theta_{ra} + c_{ra}f_1(A_r)$
$\xi_1 = \xi_2 = \xi_7 = \xi_{11} = \xi_{12}$	0
$\xi_3 = \xi_8$	read: $\Theta_{cur} + c_{do}f_1(D_0) + c_{cur}f_1(A_c)$ write: $\Theta_{caw} + c_{di}f_1(D_0) + c_{caw}f_1(A_c)$
$\xi_4 = \xi_5 = \xi_6$	read: $\Theta_{cur} + c_{do}f_1(D_i)$ write: $\Theta_{caw} + c_{di}f_1(D_i)$
$\xi_9 = \xi_{10}$	$\Theta_{pr}$
$\xi_{13} + \dots + \xi_{1n}$	$\Theta_{rf}$

(a) Dynamic energy = CD + WDD

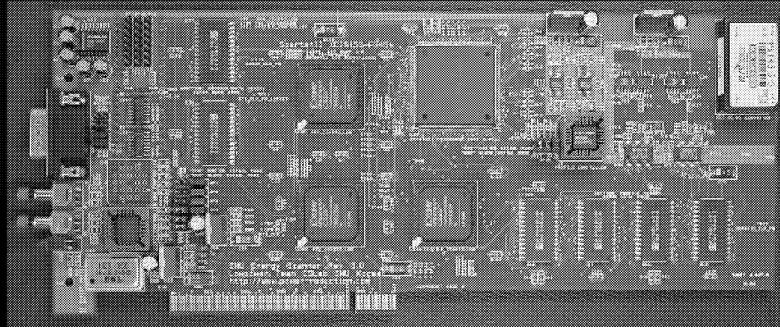
Symbol	Description	Samsung	Micron
$\Theta_{ra}$	row active	1.625	1.158
$\Theta_{cur}$	column active (read)	1.80	0.929
$\Theta_{caw}$	column active (write)	0.681	0.783
$\Theta_{pr}$	precharge	0.149	0.261
$\Theta_{rf}$	refresh	4.51	5.30

(b) CD constants

(unit: nJ/bit)

## SDRAM memory system optimization

- Input stimuli
  - Dedicated processor approach
  - SNU Energy Scanner for ARM7TDMI
    - SES: A highly integrated energy monitoring tool for low-power embedded programs, in IEEE Design and Test of Computers, 2002.



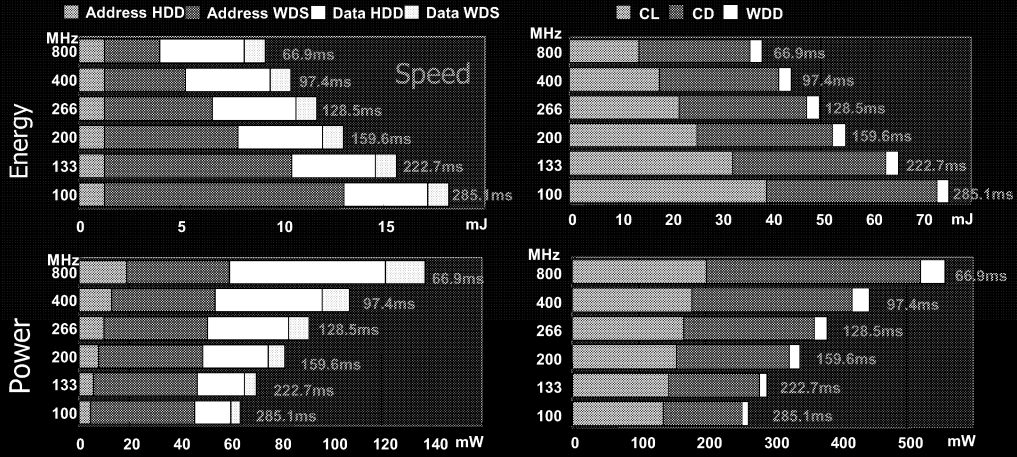
## SDRAM memory system optimization

- Application programs for the simulation
  - MP3 decoder
  - JPEG compressor and decompressor
    - $512 \times 512$  gray-scale Lenna image
  - MPEG4 decoder
    - $176 \times 144$  pixel size
- Default system configuration
  - 266MHz processor clock frequency
  - 66MHz memory clock frequency
  - 8KB/2way-set/4word block for Instruction cache and Data cache
  - Auto precharge policy of the SDRAM controller

# SDRAM memory system optimization

## Energy and speed exploration

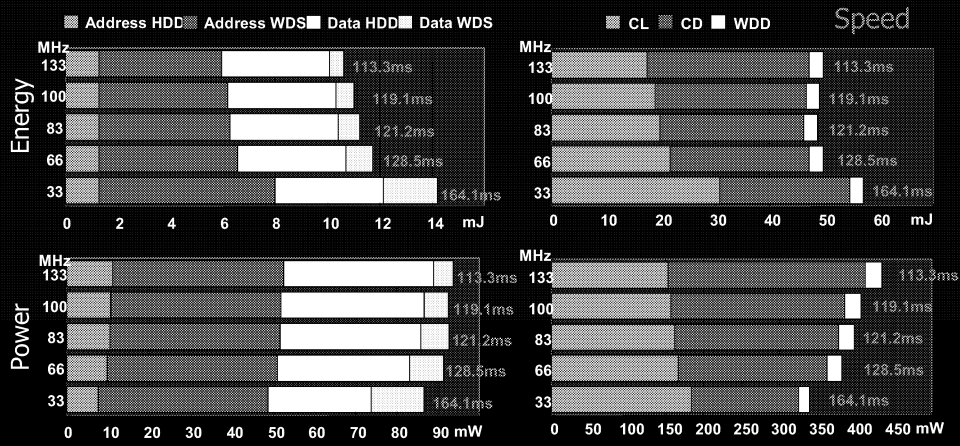
– By processor clock with JPEG compressor, 24M instructions



# SDRAM memory system optimization

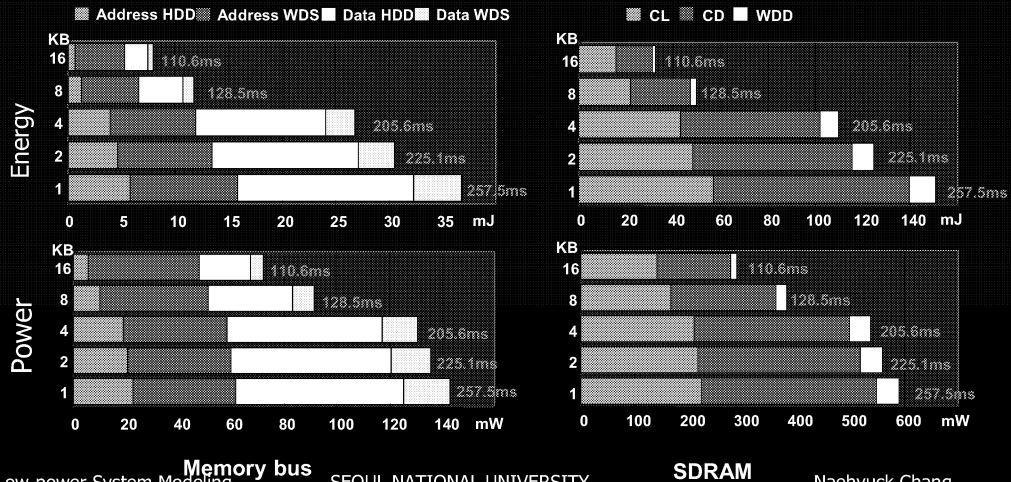
## Energy and speed exploration

– By memory clock JPEG compressor, 24M instructions



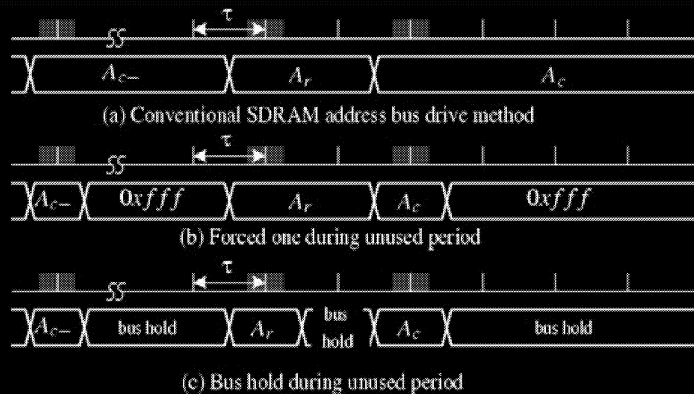
# SDRAM memory system optimization

- Energy and speed exploration
  - By cache size JPEG compressor, 24M instructions



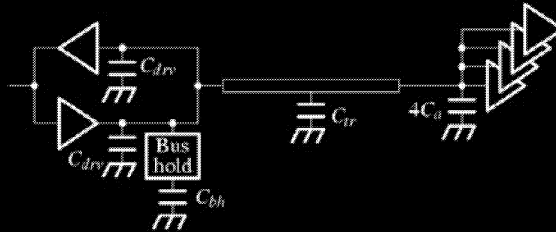
# SDRAM memory system optimization

- Energy reduction of address bus
  - Static energy reduction techniques
    - Forced 1 scheme
    - Bus-hold scheme



# SDRAM memory system optimization

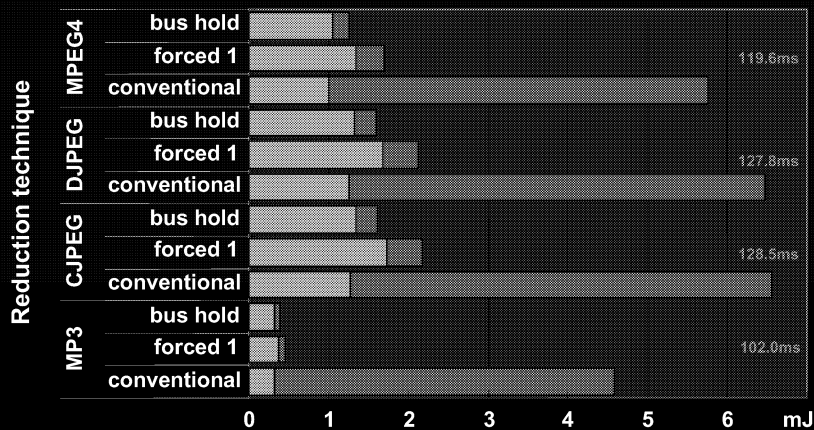
- Energy reduction of address bus
  - Static energy reduction techniques
    - Bus-hold scheme



Bus-hold scheme

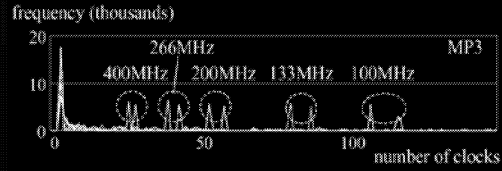
# SDRAM memory system optimization

- Energy reduction of address bus
  - Performance evaluation
    - HDD ■ WDS

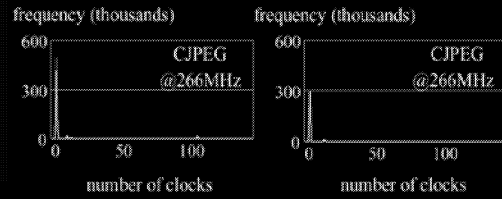


# SDRAM memory system optimization

- Energy reduction of memory device
  - Active  $\leftrightarrow$  idle mode control
    - Idle clocks between successive memory operations
  - Optimal configuration
    - COTS controller is in 256 clock steps
    - MP3 decoder: 3 clocks
    - JPEG compressor: 4 clocks
    - JPEG decompressor: 4 clocks
    - MPEG4 decoder: 10 clocks



(a) idle clock distribution

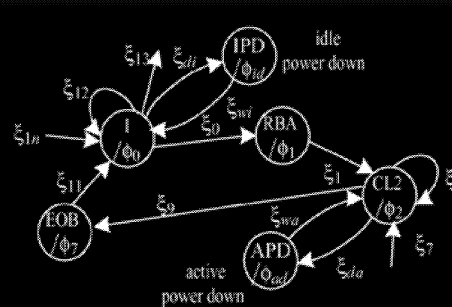


(b) total idle clocks

(c) idle clocks in row-active mode

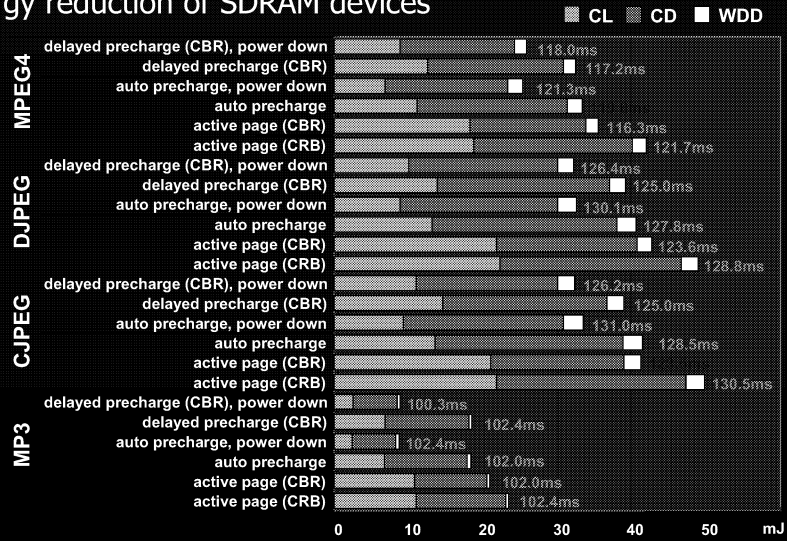
# SDRAM memory system optimization

- Energy reduction of memory device
  - Idle  $\leftrightarrow$  power down mode control
    - Idle time distribution
    - Dynamic cost and static cost in calculating mode control energy overhead
    - Cost for idle  $\rightarrow$  idle-mode power down  $\rightarrow$  idle



# SDRAM memory system optimization

## Energy reduction of SDRAM devices



## Questions?

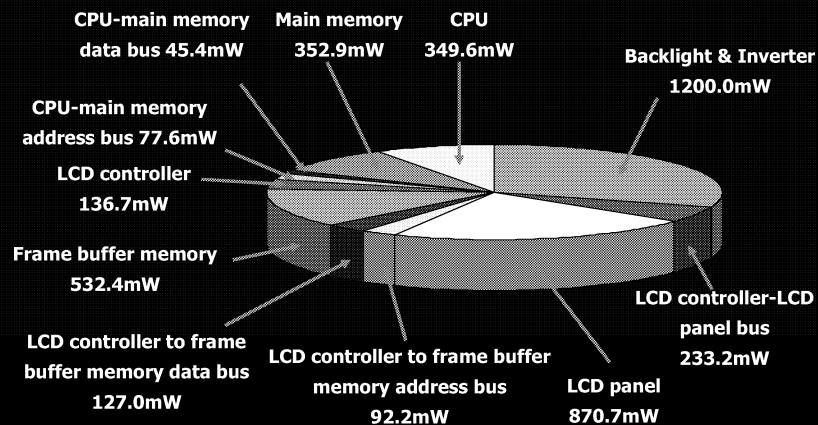
# Low-power LCD systems

Naehyuck Chang  
Assistant Professor  
School of Computer Science and Engineering  
Seoul National University  
[naehyuck@snu.ac.kr](mailto:naehyuck@snu.ac.kr)

SEOUL NATIONAL UNIVERSITY

## Where the power goes?

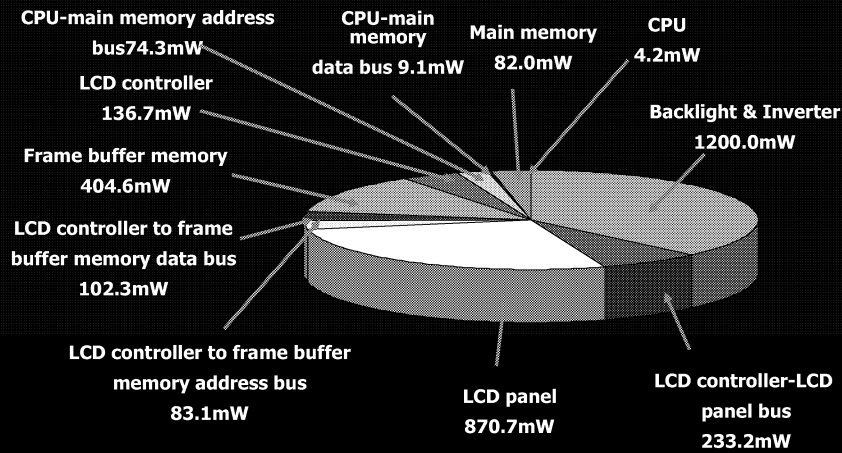
- ◆ System wide power consumption of an MPEG4 player





# Where the power goes?

## ■ System wide power consumption of a document viewer



# Typical embedded systems

## ◆ Embedded systems with a quality LCD system

- Processor
  - 32bit RISC CPU @206MHz
- Cache
  - 8KB 2-way D-cache
  - 8KB 2-way I-cache
- Main memory
  - 64MB SDRAM @66MHz
  - 2" LVT address bus
  - 2" 32bit LVT data bus

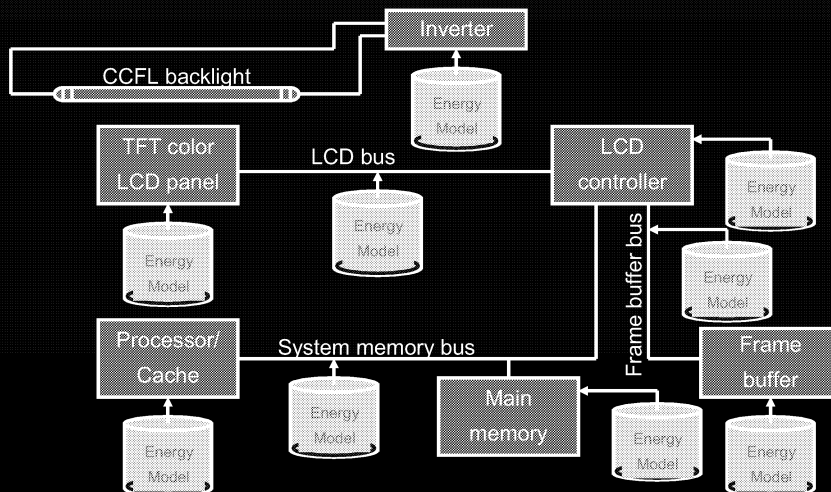
- LCD controller
  - Xilinx Spartan-II XC2S150-FG456 FPGA
- Frame buffer memory
  - 8MB SDRAM @66MHz
  - 2" LVT address bus and 32bit data bus
- LCD panel
  - LG-Philips LP064V1 transmissive color TFT LCD
  - 6" 640x480 @60Hz refresh
  - CCFL backlight
- LCD to LCDC bus
  - 4" control bus and 16bit data bus

# Power reduction of LCD systems

- Low-power techniques for CPU and memory systems
  - Power portion of the LCD systems become more significant
- Power reduction of LCD systems
  - No explicit slack time
    - Discontinuous service, Serious quality degradation
    - N/A power management scheme
- Major power consumers
  - Backlight system
  - Frame buffer memory
  - LCD bus
  - Frame buffer memory bus
- Power reduction w/o appreciable quality degradation
  - Use of nature of the LCD panel and backlight system

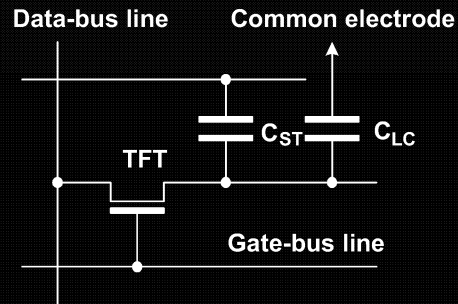
# Power reduction of LCD systems

## Block diagram of an LCD system



# Variable-duty-ratio refresh

- LCD and CRT display
  - Compatibility
- Keep the transparency between CPU and frame buffer
- LCD sub-pixel equivalent circuit
  - Equivalent liquid crystal capacitance  $C_{LC}$
  - Explicit storage capacitor  $C_{ST}$

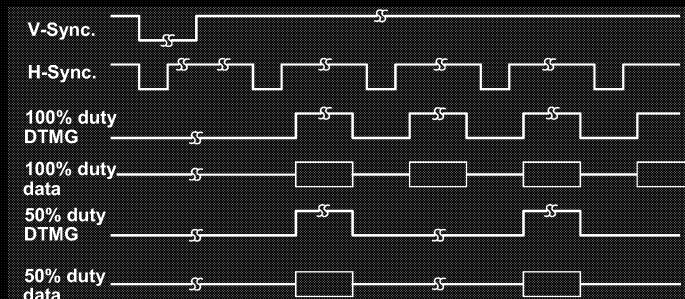


- Once the refresh rate is higher than the time constant of the CST, the image does not actually flick
- Variable-dot-clock frequency requires a special hardware support: non-interruptible PLL

# Variable-duty-ratio refresh

## ■ Implementation of the variable-duty-ratio refresh

- Energy saving in frame buffer memory, frame buffer memory bus and LCD bus



- 75% duty ratio refresh → 129.8mW energy saving w/o flickering
- 50% duty ratio refresh → 259.7mW energy saving w/o flickering
- 33% duty ratio refresh → 346.2mW energy saving w/ minor flickering
- 25% duty ratio refresh → 389.5mW energy saving w/ noticeable flickering

# Dynamic-color-depth control

## Conventional mapping

- Reduced 8bit/pixel color depth
  - Default energy consumption
  - Larger page size or more number of pages

## Suggested mapping

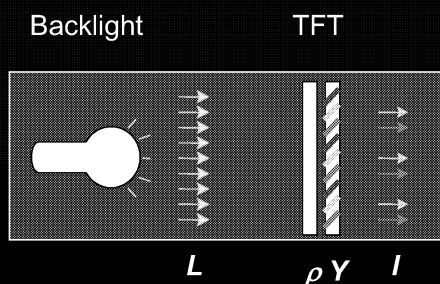
- Reduced 8bit/pixel color depth
  - Shut down the lower half of the frame buffer
  - Default page size or default number of pages
  - About 50% energy saving for frame buffer memory
  - Significant energy saving for the frame buffer bus and the LCD bus



# LCD backlight systems

## Brightness compensation

- Dim the backlight
  - Power consumption is proportional to the luminance
- How to maintain the display quality?



$L$  : luminance of backlight

$\rho$  : transmittance of image

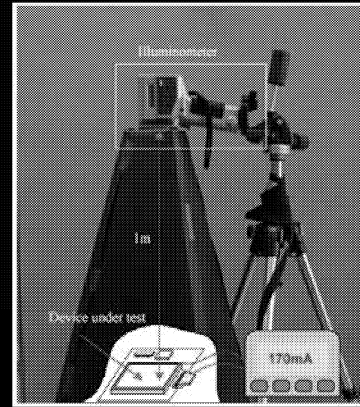
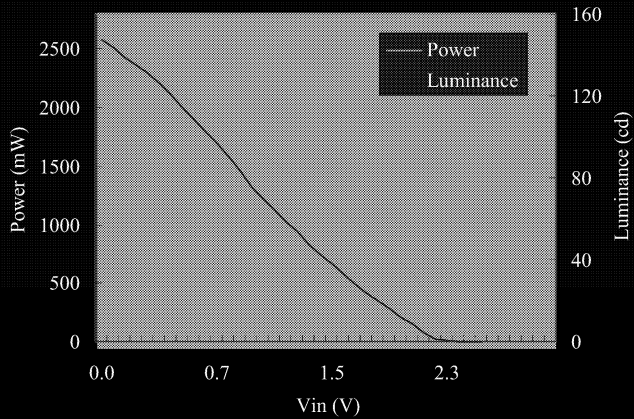
$Y$  : luminance of image

$I$  : perceived intensity

$$I = \rho LY$$

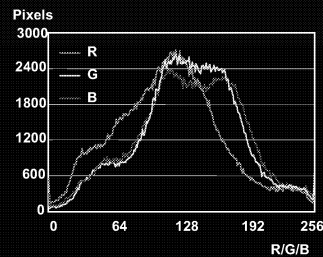
# LCD backlight systems

## Luminance versus power consumption

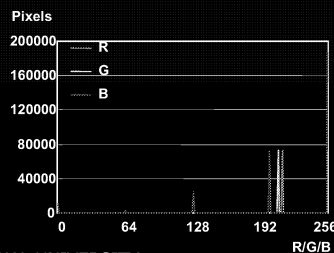
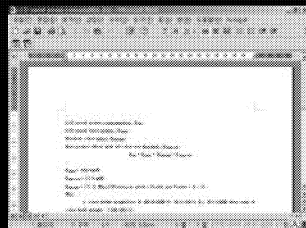


# Brightness compensation

## Image processing to recover the brightness or contrast

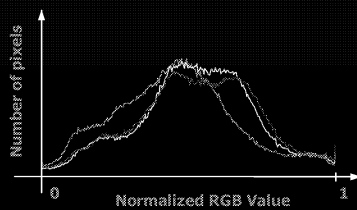
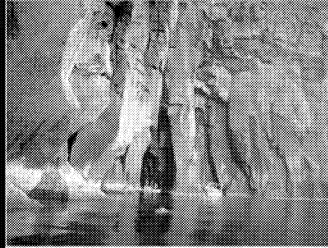


Brightness compensation is suitable

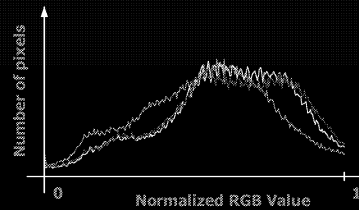


# Brightness compensation

■ Before



■ After



# Brightness compensation

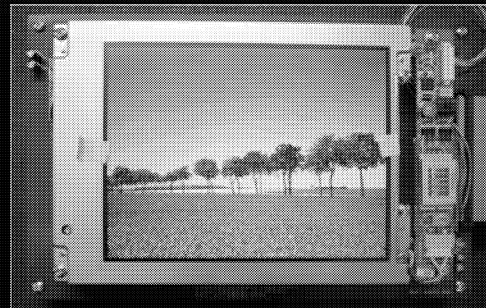
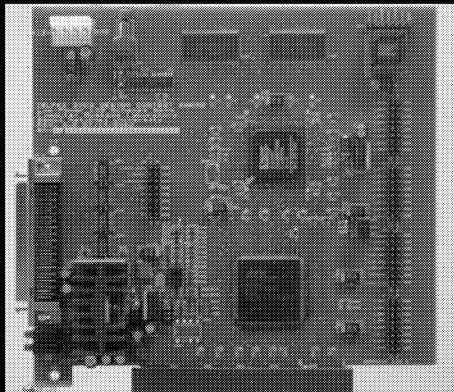
- SBLS (Static backlight luminance scaling)
- Maintain the original display color very closely
- Is suitable for graphic image
- Stretch the color spectrum to compensate the given amount of the backlight dimming
- Results in variable image distortion ratio

# Image enhancement

- Contrast enhancement
  - Histogram stretching and histogram equalization
  - Attempt aggressive power saving by aggressive dimming
  - Give up the original display color
  - Keep the similar contrast
  - Is suitable for text display

# Low-power LCD platform

- LCDC
- LCD panel & backlight system



# Low-power design practices

- MPEG4 player and Image viewer
  - Variable-duty-ratio
  - dynamic-color-depth control
  - Brightness shift
- MP3 player, Document viewer and Text editor
  - Variable-duty-ratio
  - dynamic-color-depth control
  - Contrast shift
- System-wide energy reduction

Application	Original (mW)	Reduction (mW)	%
MPEG4 player	4028	3358	16.6
MP3 player	3430	2545	25.7
Image viewer	4015	3408	15.1
Document viewer	3202	2315	27.6
Text editor	3197	2313	27.6

# Questions?



## DLS (Dynamic Backlight Luminance Scaling)

- DBLS (ISPLED design contest award 2002)

- The Saturation ratio ( $S_R$ ) is given
- Calculate  $Y^{TH}$

$$S_R = \frac{\int_{y^{TH}} y_{h,w} dy}{\int y_{h,w} dy}$$

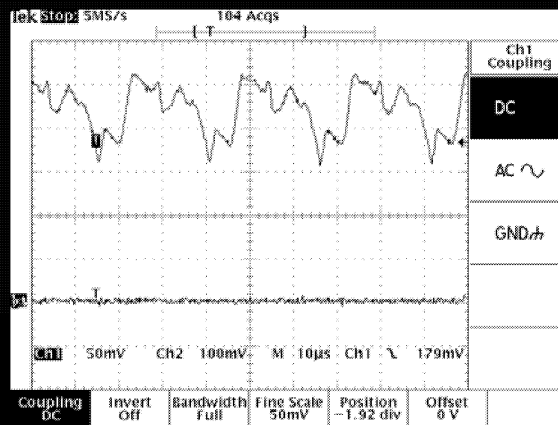
- Convert the image to have  $Y'$ 
  - Image conversion equation

$$Y' = \min\left(1, \frac{1}{Y^{TH}} Y\right)$$

- Dim the backlight by  $Y^{TH}$
- $Y^{TH}$  is variable for every frame

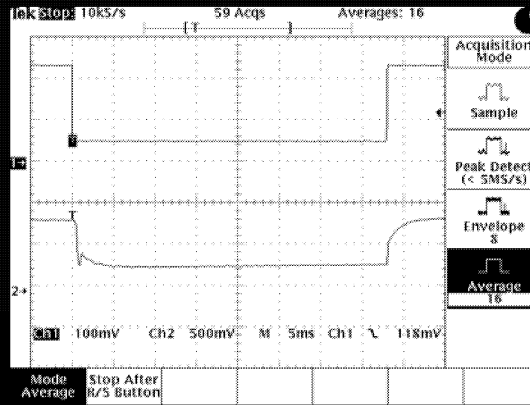
## Dynamic control of a CCFL

- Voltage across the CCFL



# Dynamic control of a CCFL

- Adding a photo diode

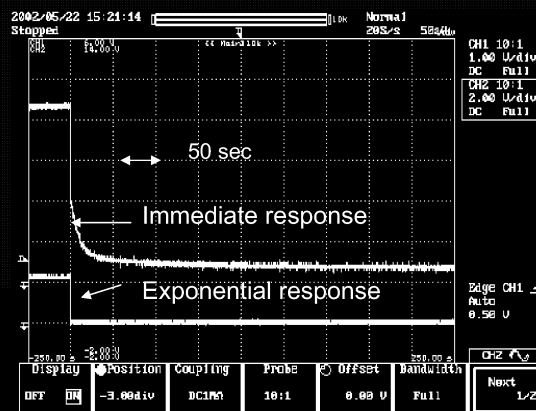


# Dynamic control of a CCFL

- Challenges
  - Extreme slow response of the CCFL

Response of  
the backlight

Command  
input

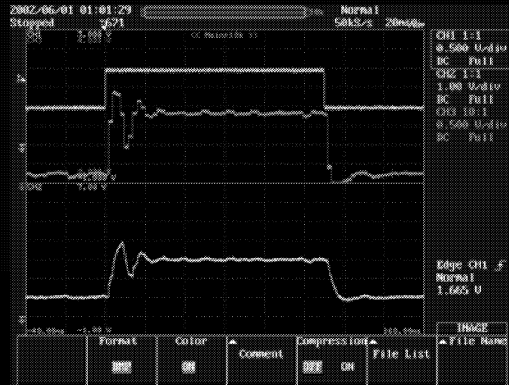


# Dynamic control of a CCFL

## ■ PID control

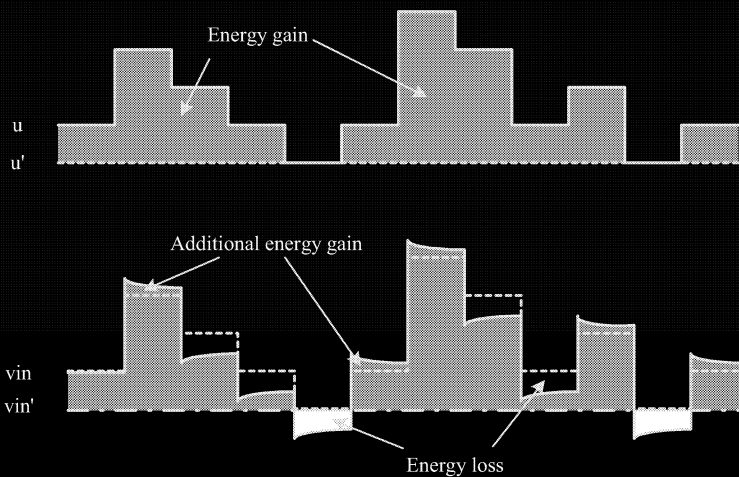
$$v_i = v_{i-1} + K_p e_i + K_i \frac{1}{T} \sum_{k=0}^i e_k + K_d T (e_i - e_{i-1}),$$

$$e_i = u_i - f_p(V_{Di})$$



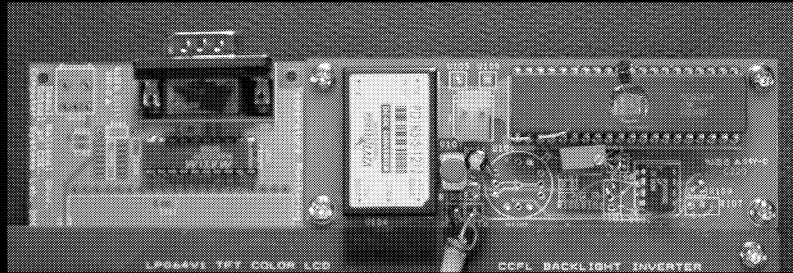
# Dynamic control of a CCFL

## ■ Energy gain and loss after feedback control



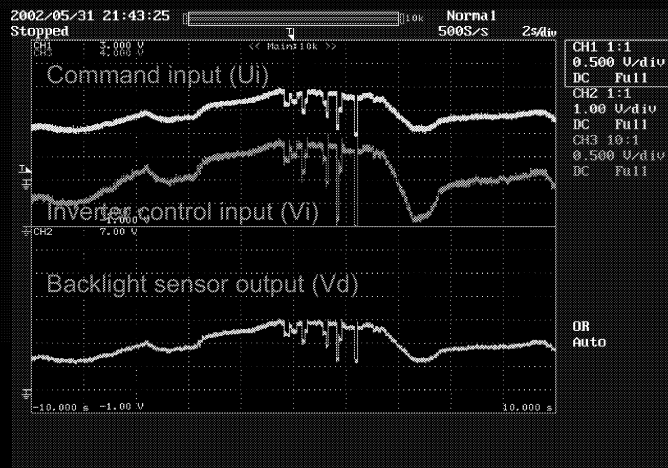
# Dynamic control of a CCFL

- Feedback controller
  - 1ms time tick
  - PIC microcontroller



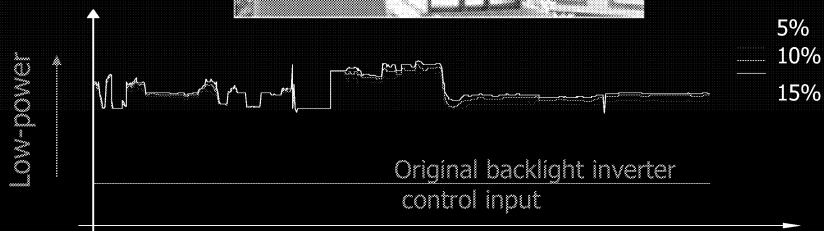
# Sequence of backlight luminance

- A movie example



# Sequence of backlight luminance

- Movie stream 1: Funny movie



# Sequence of backlight luminance

- Energy gain of the backlight system

Movie 1



Heo Joon

Movie 2

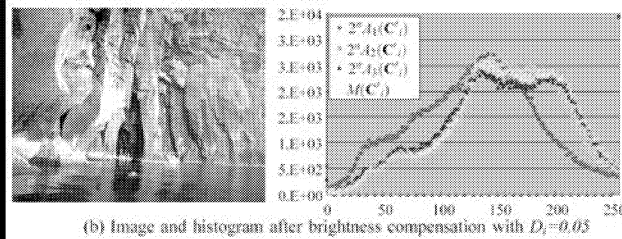
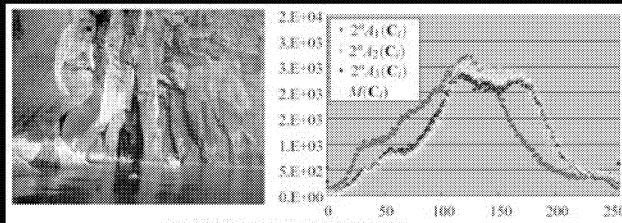


Funny movie

Saturated pixels (%)	Gain (%)	
	Movie1	Movie2
0	0	0
5	18.2	30.6
10	19.2	36.8
15	23.5	41.4

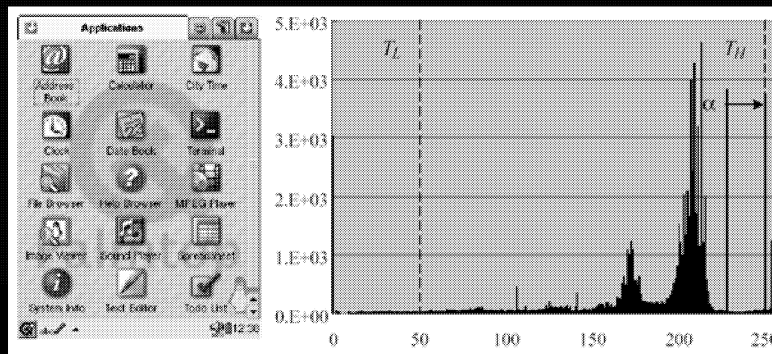
# Image processing for DLS

- Brightness compensation



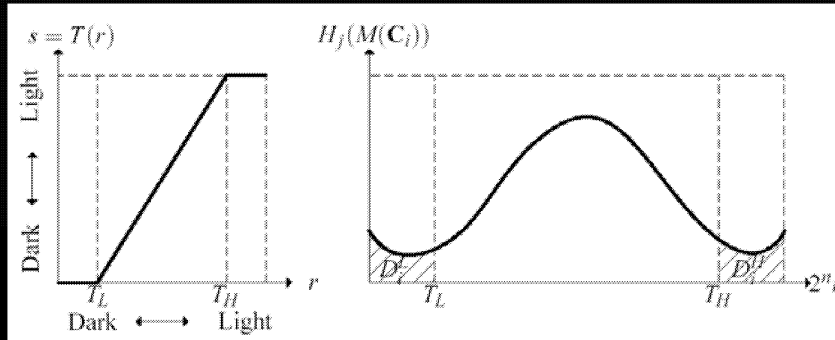
# Image processing for DLS

- Brightness compensation
  - Unstable with discrete spectrums



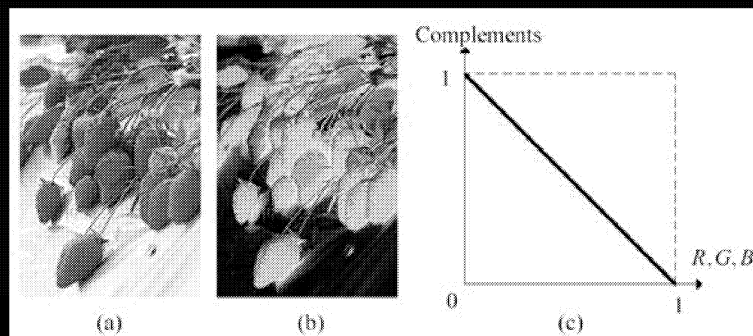
# Image processing for DLS

- Histogram stretching
  - Robust than the brightness compensation
  - Double computational complexity



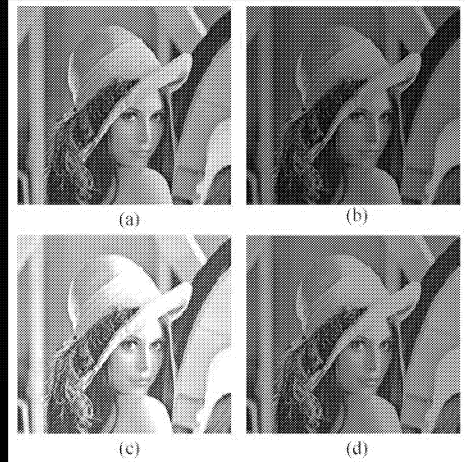
# Image processing for DLS

- Use of complementary colors



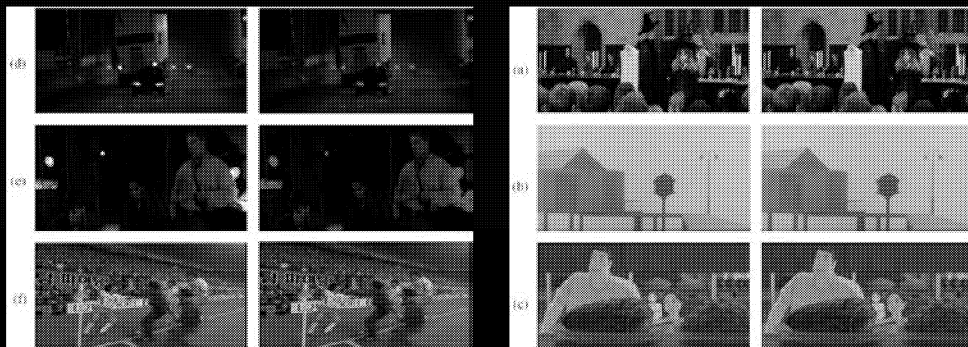
# Image processing for DLS

- Brightness compensation
  - (a) original image
  - (b) simulation of original image on LCD with the dimmed backlight (64%)
  - (c) compensated image,  $D_i = 0.7$
  - (d) simulation of compensated image on LCD with the dimmed backlight (64%).



# Image processing for DLS

- Brightness compensation
  - Movie examples with  $D_i=0.03$ ;
    - (a) 147mW/87.5%; (b) 147mW/87.5%; (c) 188mW/82.0%; (d) 703mW/42.1%; (e) 805mW/35.9%; and (f) 278mW/75.0%.

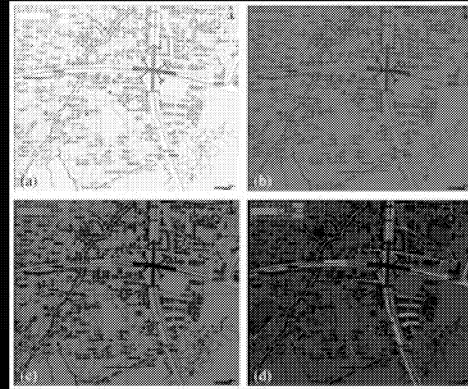




# Image processing for DLS

## ■ Image enhancement

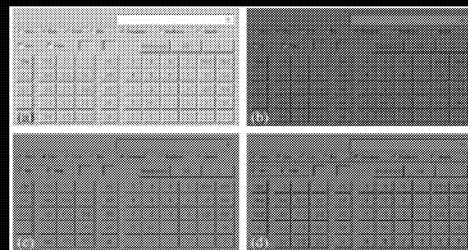
- (a) Original image
- (b) Simulation of the original image under dimmed backlight (50%)
- (c) Simulation of histogram stretching with 10% threshold under dimmed backlight (50%)
- (d) Simulation of histogram equalization under dimmed backlight (50%).



# Image processing for DLS

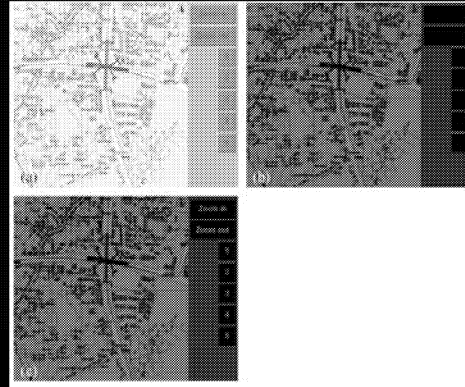
## ■ Image enhancement

- (a) Original image
- (b) Simulation of the original image under dimmed backlight (50%)
- (c) Simulation of histogram stretching with 10% threshold under dimmed backlight (50%)
- (d) Simulation of histogram equalization under dimmed backlight (50%).



# Image processing for DLS

- Context processing
  - (a) Original image
  - (b) Simulation of histogram stretching with 10% threshold
  - (c) Simulation of context processing after histogram stretching under dimmed backlight (50%).



Sweden, July 14 - October 10,  
2003

European Summer School on Embedded Systems

## SEE Web and Experiments

Naehyuck Chang  
Assistant Professor  
School of Computer Science and Engineering  
Seoul National University  
[naehyuck@snu.ac.kr](mailto:naehyuck@snu.ac.kr)

# Conclusions

- High-level, system-wide energy reduction
  - Promising and ultimate energy optimization
  - Need of an accurate, fine-grain energy estimator
- Cycle-accurate energy measurement tools
  - Switched-capacitor-based measurement circuit (IEE Electronics Letters 1999)
  - SNU Energy Characterizer (Ubooth 2000)
  - SNU Energy Scanner (Ubooth 2001, IEEE D&T 2002)
  - SNU Energy Characterizer for FPGAs (Ubooth 2002, ISQED 2003)
  - SNU Energy Explore (2001~)
  - Apollo testbed (USC and DARPA 2002~2003)

# Conclusions

- Energy reduction techniques
  - Bus encoding for high-performance buses (DAC 2000)
  - ARM7TDMI energy characterization (DAC 2000, IEEE TVLSI 2002)
  - Low-energy SDRAM memory systems (DAC 2002, ACM TECS 2003)
  - Low-power LCD controller (ISLPED 2002, Ubooth 2002)
  - Dynamic backlight luminance control for MPEG4 players (Ubooth 2002, ACM ISLPED design contest award 2002)
  - Dynamic backlight luminance control for J2ME PBP (ISLPED 2003)
  - Low-power non-volatile memory systems (ISLPED 2003)

# Conclusions

- Recent work
  - Compressed frame buffer  
(2003 Low-power design contest award 2003)
  - SEE Web (demonstrated at Ubooth 2003)

Sponsored by:

