



EE542

Internet and Cloud Computing

Lecture 21 – Estimating Power with ML

Young Cho
Department of Electrical Engineering
University of Southern California

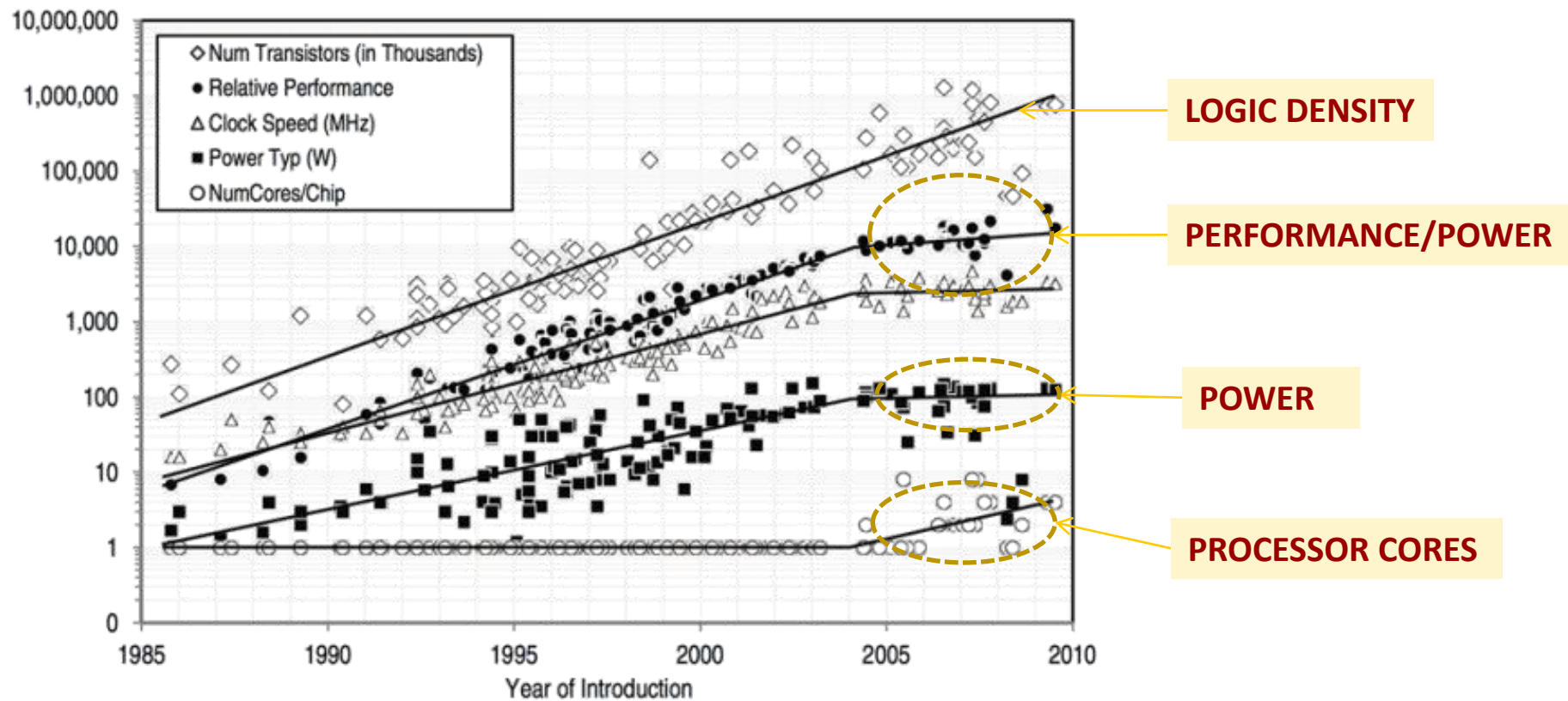
Final Project Progress Demos

- Due Every Mondays 11:59 PM in Nov
- Must Demonstrate Working System
- Incremental Functions Added with Corresponding Results
- System Must Work Every Demo
- Every Team Member Must Contribute in Describing the Demo

Final Project Progress Presentations

- November 18
 - Submit Practice Presentation Slides
- November 20
 - Go over the Practice Presentations
- November 25
 - Submit Final Project Progress Presentation Slides
- November 27, 29
 - Final Project Progress Presentation
 - 7 teams on 27th and 6 teams on 29th
 - Mandatory Attendance
- December 13
 - Final Project Demo Presentation Video

The History



Source: ITRS

The Problem

Voltage scaling as an effective weapon against power reaching its limits !

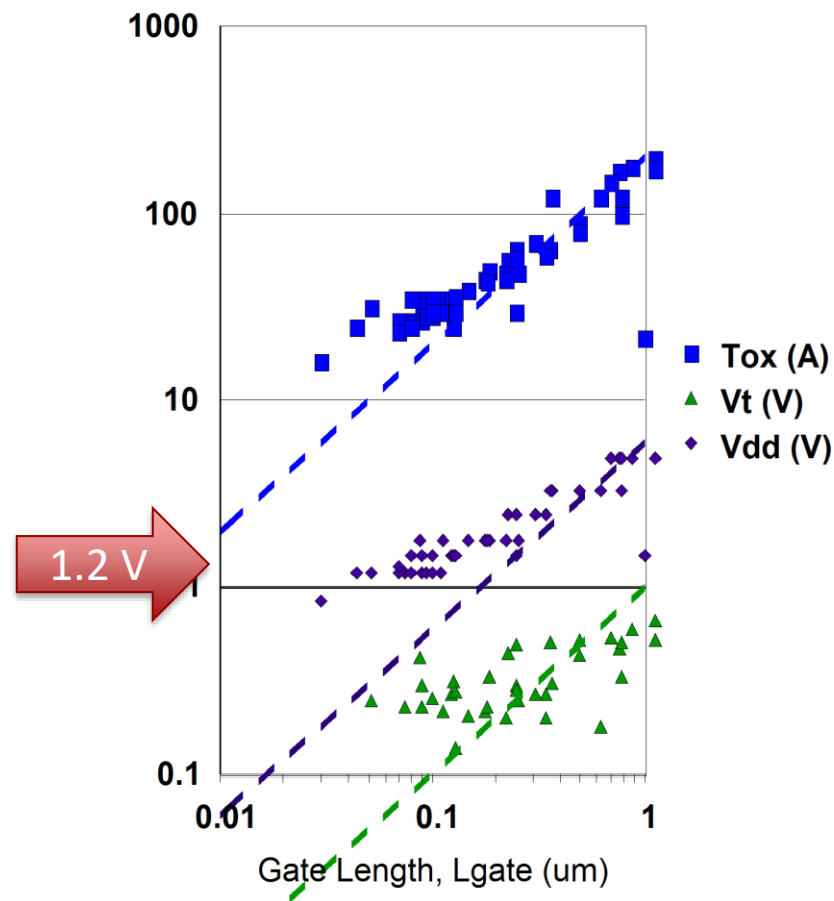
- CMOS at 45 & 22 nm nodes run at 1.2V.

Increased power per unit area

- Thermal Hot spots
- Inefficient Cooling systems

Other Alternative is, **Supply voltage gating**

- Gains diminished by inaccuracies in measurements

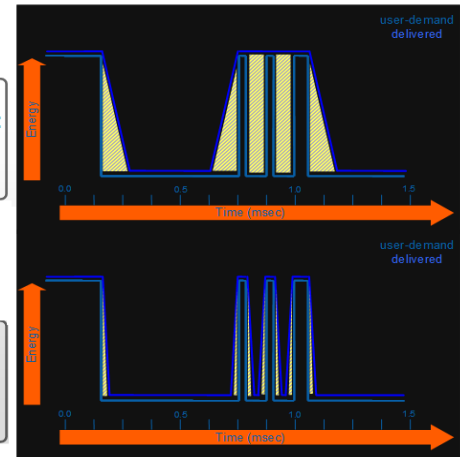
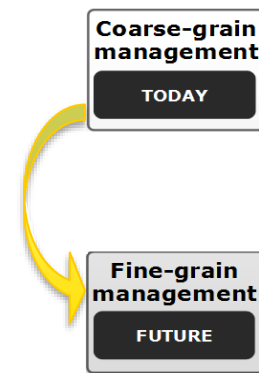


Source: Intel & IBM

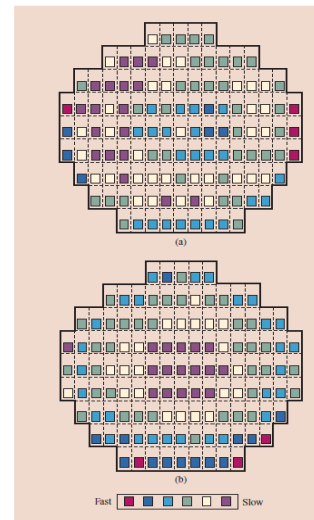
Current Solutions

Current in-situ methods not efficient, cost-effective or scalable

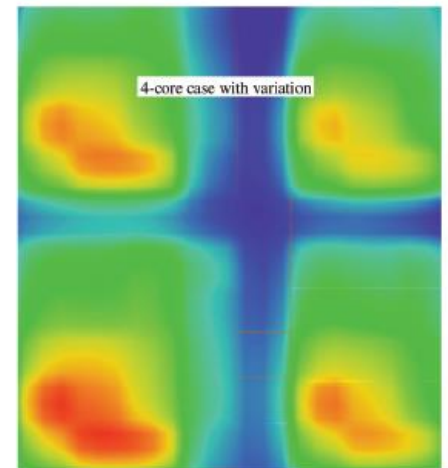
- Mostly analog or mixed signal (high power, area, cost, and poor scalability)
- MIT's on-chip method yields up to 60% power savings, but comes at a huge silicon area cost
- No or Minimal Accounting of PVT Variations
- Result: Less accurate and worse with increasing PVT variations



Reference : V.De , "Fine grain Power management", Forum talk, ISSCC 2013

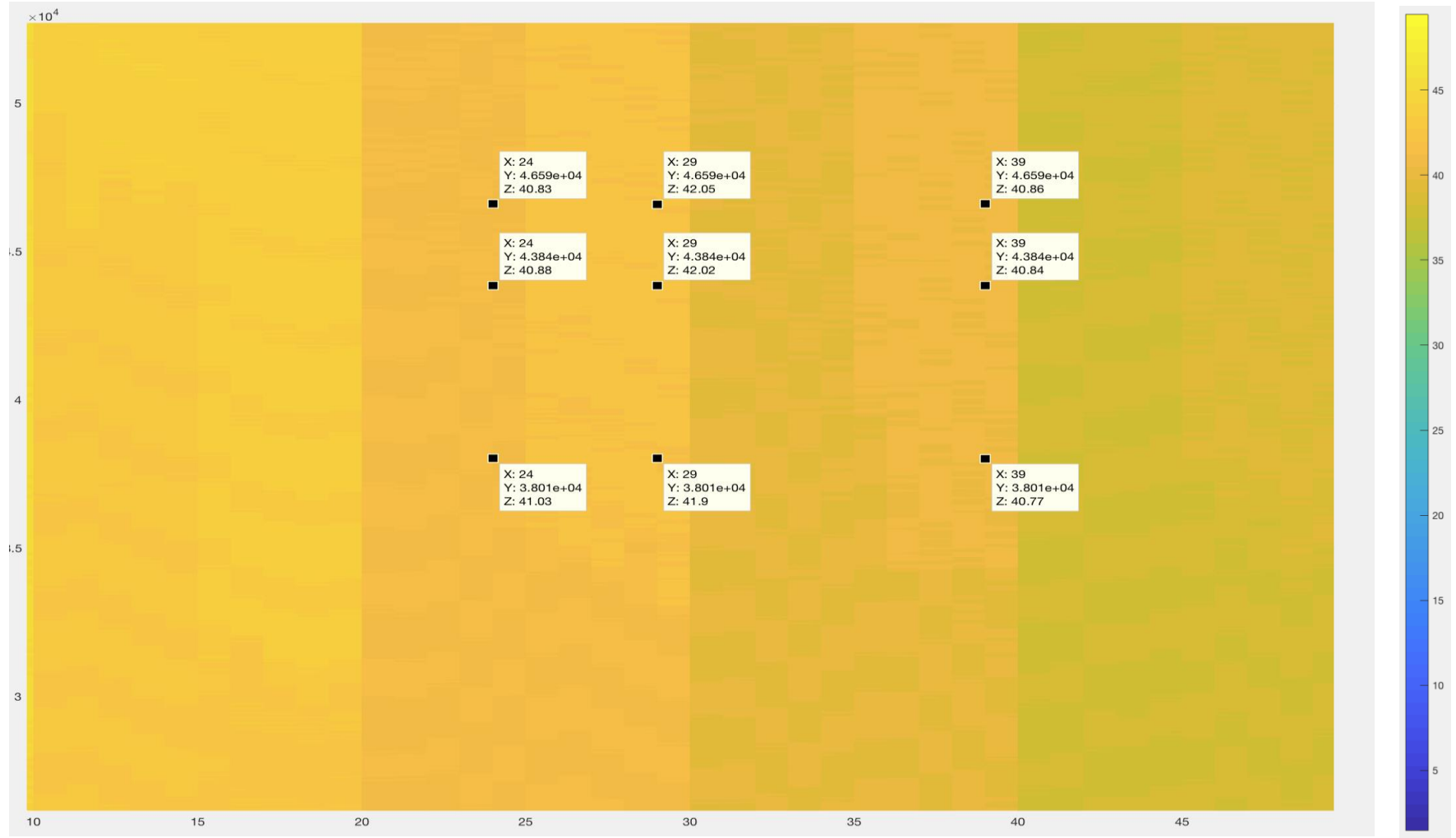


PVT variations



Temperature variations due to packaging

GTX 1070 – 15 Core GPU



Enable 12
best cores

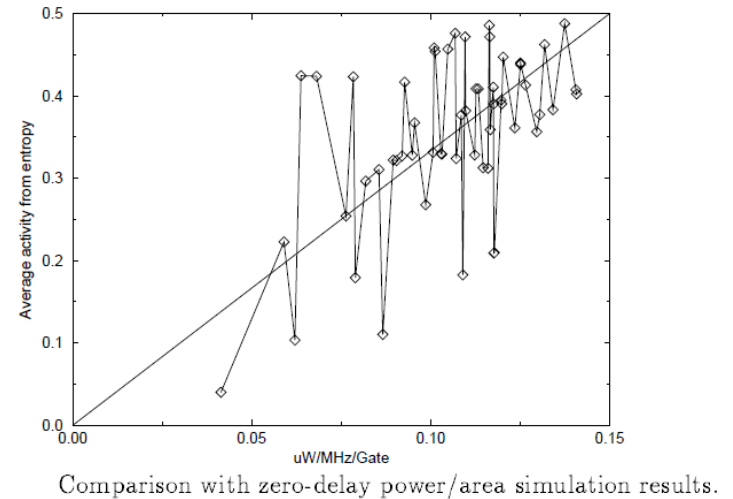
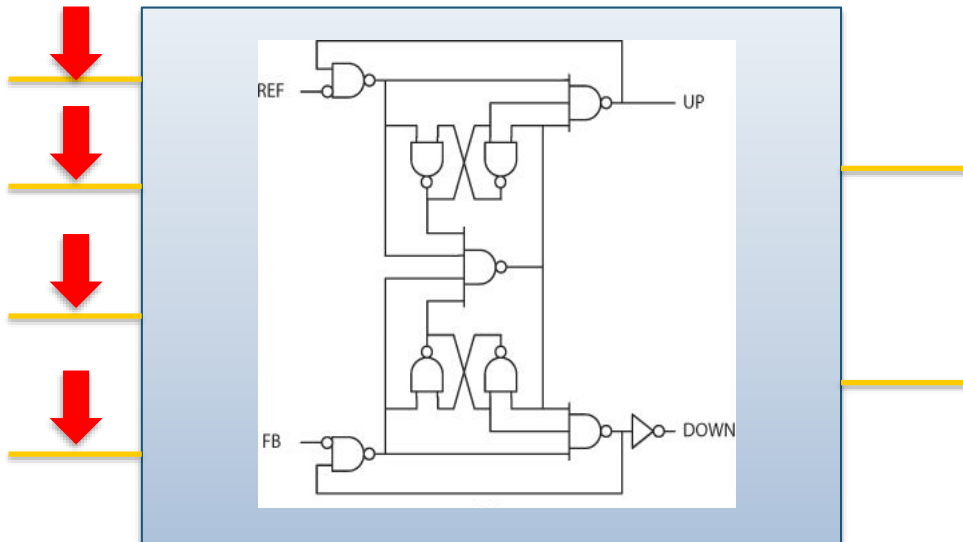
Enable 12
worst cores

Enable 11
worst cores

Our Work

- Combine Direct measurements and fundamental theory of estimation
- Online linear solver to partition power
- Regressively extract the best model for the device by accurate model collapsing per chip basis.

Background



$$P_{dyn} = \alpha \cdot Act$$

References

- M. Nemani and F. Najm, "Toward a high-level power estimation capability," IEEE Trans. Computer-Aided Design, vol. 15, no. 6, pp. 588–598, 1996.
- P. E. Landman, J. M. Rabaey, "Architectural Power Analysis: The Dual Bit Type Method," IEEE Transactions on VLSI, Vol 4 pp. 173-187, June 1995.

Theory

Component C consisting of n wires . Each wire driving a capacitive load of C_i

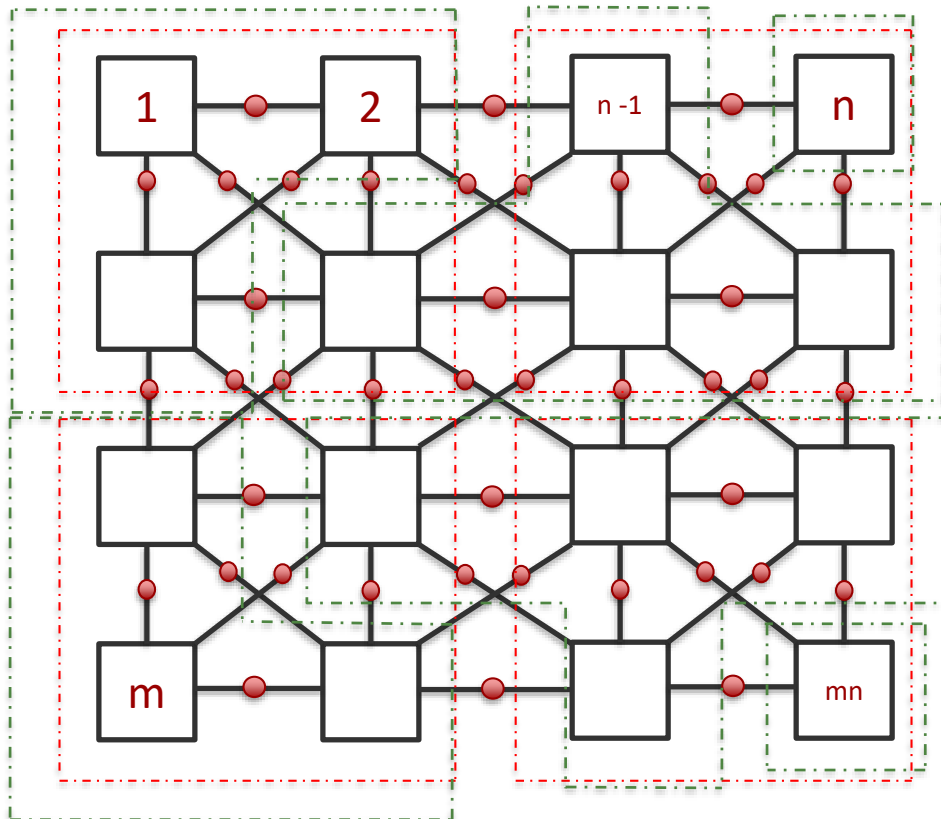
Signal Transitions at each line is represented as tr_i and Supply voltage as V_{DD}

$$P_{nom} = \sum_{i=1}^n (tr_i \cdot c_i \cdot V_{DD}^2) = \sum_{i=1}^n (tr_i \cdot W_i)$$

$$|P_{nom} - P_{est}| = \epsilon$$

$\epsilon = 0$, Ideal Conditions

$\epsilon \neq 0$, Under PVT variations
and measurement noise



$$P_{est,1} = \sum_{l=1}^m (tr_l \cdot W_l)$$

$$P_{est,2} = \sum_{q=1}^p (tr_q \cdot W_q)$$

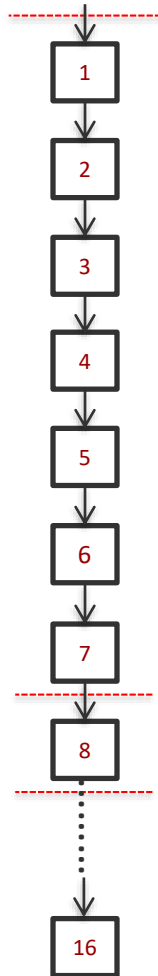
where, $n > m > p$

$$|P_{nom} - P_{est,1}| = \epsilon_1$$

$$|P_{nom} - P_{est,2}| = \epsilon_2$$

$$\epsilon_2 \geq \epsilon_1$$

Evaluating logic cuts to extract desired instrumentation point



$$P_{nom} = \sum_{i=1}^n (tr_i \cdot W_i) , \text{ Where } n = \text{all lines in } C$$

$$P_{est,1} = \sum_{p=m} (tr_p \cdot W_p), \text{ Where } m = \text{inputs of } C.$$

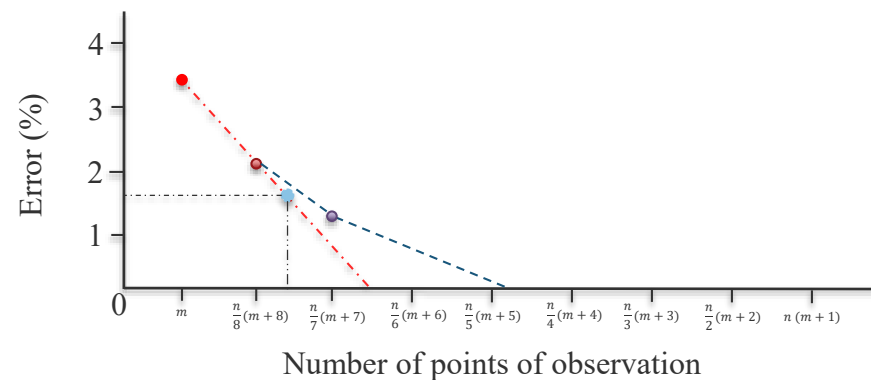
$$|P_{nom} - P_{est,1}| = \varepsilon_1$$

$$\text{if } \max(\varepsilon_1) > E ,$$

$$f(x) = \cup_{k=1}^j V_k, \text{ Such that } f(V_k) = f(V_{k+1}) \quad \forall k + 1 \leq j$$

$$P_{est,2} = \sum_{p=a} (tr_p \cdot W_p), \text{ Where } a = \text{inputs of cuts.}$$

$$|P_{nom} - P_{est,2}| = \varepsilon_2$$



Using Online Solver for Weights

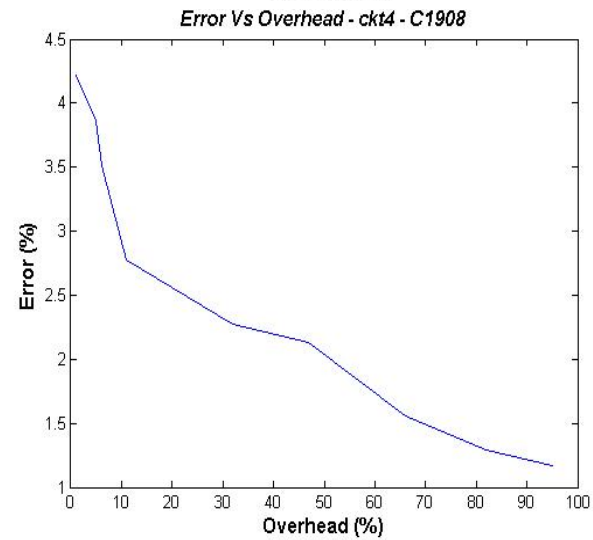
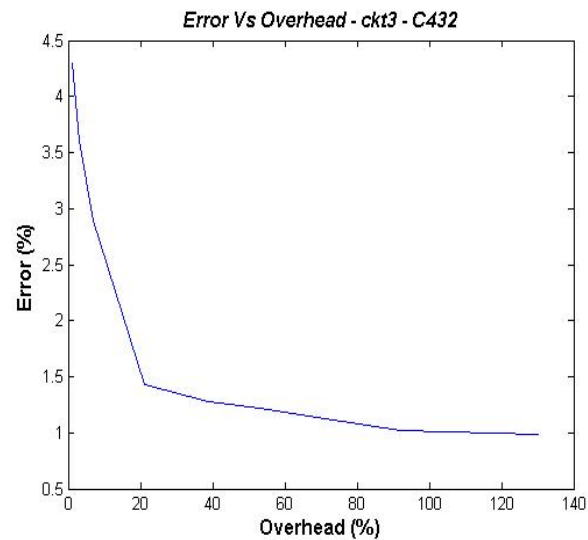
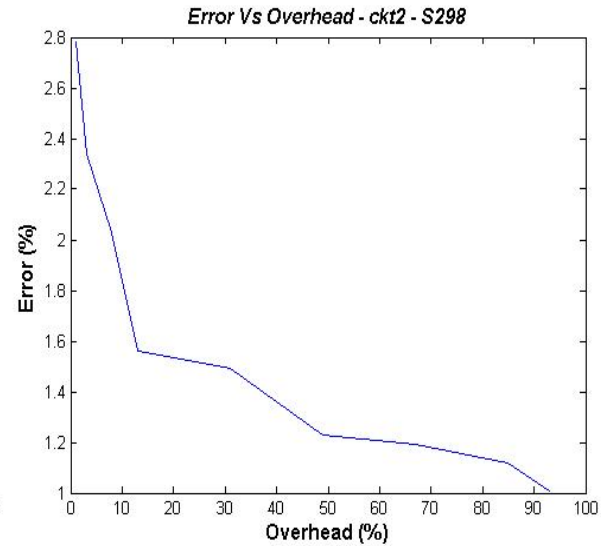
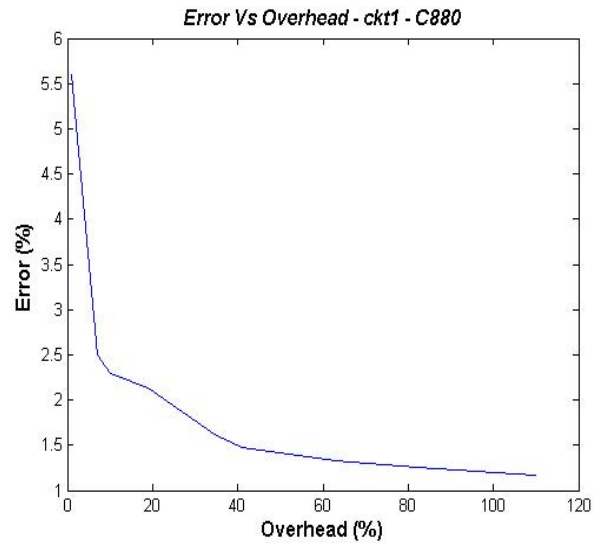
Counter values	Power Values	Residue	W1	W2	W3	W4	W5	W6	W7	WS
28 1 0 3 1 1 2 26 1	0.3063	0	0.000463369	0.000203485	0.000208883	3.94E-05	0.000166	0.000171	5.54E-05	0.000883
3 0 1 3 1 0 2 3 1	0.0717	0	0.000924628	0.000317377	0.00023828	0.000078	0.00023	0.00028	7.93E-05	0.001344
3 0 0 2 0 0 2 3 1	0.0078	0	0.000917426	0.00031614	0.000237881	7.74E-05	0.00023	0.00028	7.91E-05	0.00135
5 1 0 5 1 1 2 5 1	0.1025	0	0.000921373	0.000317757	0.00023748	7.78E-05	0.00023	0.000279	7.94E-05	0.001346
5 0 1 5 1 0 2 5 1	0.0933	0	0.000957923	0.000330416	0.000240154	8.08E-05	0.000234	0.000291	8.16E-05	0.001396
5 0 0 4 0 0 2 5 1	0.0039	0	0.000957177	0.000330682	0.000240756	8.07E-05	0.000234	0.000291	8.15E-05	0.001397
7 1 0 7 1 1 2 7 1	0.1265	0	0.000964848	0.00033605	0.000241628	8.14E-05	0.000237	0.000295	8.24E-05	0.001415
7 0 1 7 1 1 2 7 1	0.1429	0	0.000963136	0.000335876	0.000242144	8.13E-05	0.000237	0.000295	8.22E-05	0.001417
7 0 0 6 0 0 2 7 1	0.0973	8.05E-07	0.001044859	0.00037248	2.51E-04	8.83E-05	0.000253	3.24E-04	8.69E-05	0.001543
9 1 0 9 1 1 2 9 1	0.1409	9.01E-07	0.001045028	0.000372572	2.51E-04	8.83E-05	0.000253	3.24E-04	8.69E-05	0.001543
9 0 1 9 1 0 2 9 1	0.1453	9.02E-07	0.001044964	0.000372552	2.51E-04	8.83E-05	0.000253	3.24E-04	8.69E-05	0.001543
9 0 0 8 0 0 2 9 1	0.0179	9.02E-07	0.001045086	0.000372471	2.51E-04	8.83E-05	0.000253	3.24E-04	8.69E-05	0.001543
11 1 0 11 1 1 2 11 1	0.1747	9.16E-07	0.001045088	0.000372474	2.51E-04	8.83E-05	0.000253	3.24E-04	8.69E-05	0.001543
10 0 1 10 1 0 2 10 1	0.0012	9.18E-07								
11 0 0 11 0 0 2 11 1	0.1695	9.23E-07								
13 1 0 13 1 1 2 13 1	0.2305	9.24E-07								
13 0 1 13 1 0 2 13 1	0.2165	9.78E-07								
12 0 0 12 0 0 2 12 1	0.0039	9.88E-07								
15 1 0 15 1 1 2 15 1	0.2233	9.08E-07								
15 0 0 15 0 1 2 15 1	0.2683	9.18E-07								
16 0 0 17 1 0 2 16 1	0.2347	9.8E-07								

$$P_{\text{total},t} = \sum_{i=1}^k (a_i \cdot P_{\text{dyn}}(I_i)) + P_{\text{St}}(\sum_{i=1}^k I_i) + \text{Residue}$$

$$P_{\text{Total}} = (\sum_{j=1}^n W_j a_j) + (1 * W_S) + \text{Residue}$$

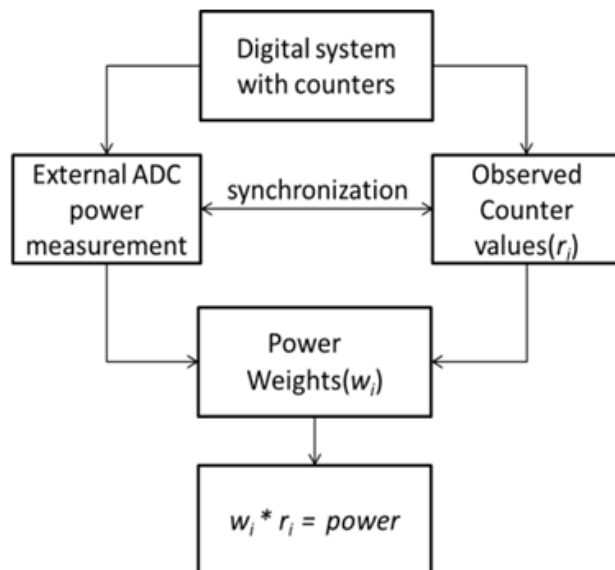
Generalized Recursive information form of estimation,
 $Z(k + 1) = H(k + 1).W + V(k + 1)$

Initial Results

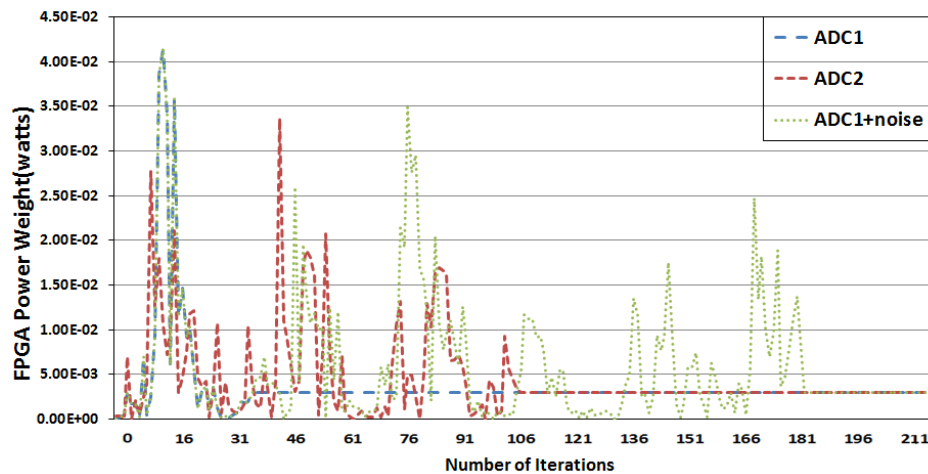
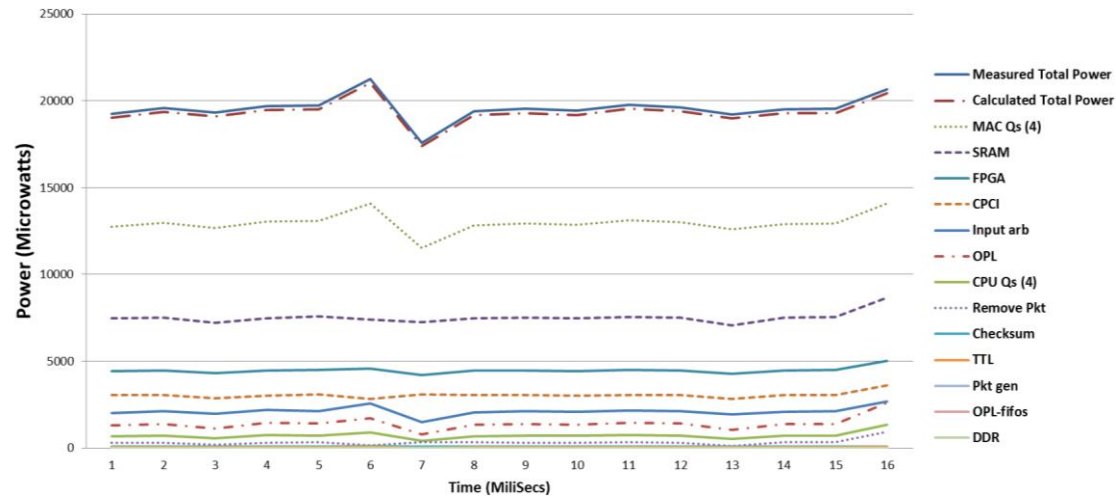


Reduction of methodology to practice

- NetFPGA : Open source hardware with all the networking appliances built in.
- The goal was to identify the power used by different subcomponents of the board. Eg: Eth Phy , SRAM, DDR



Initial Hardware Results



Maximum deviation less than 3% from ADC

Reduce Overhead without losing accuracy

Problem : How do I reduce the instrumentation overhead and yet maintain accuracy ?

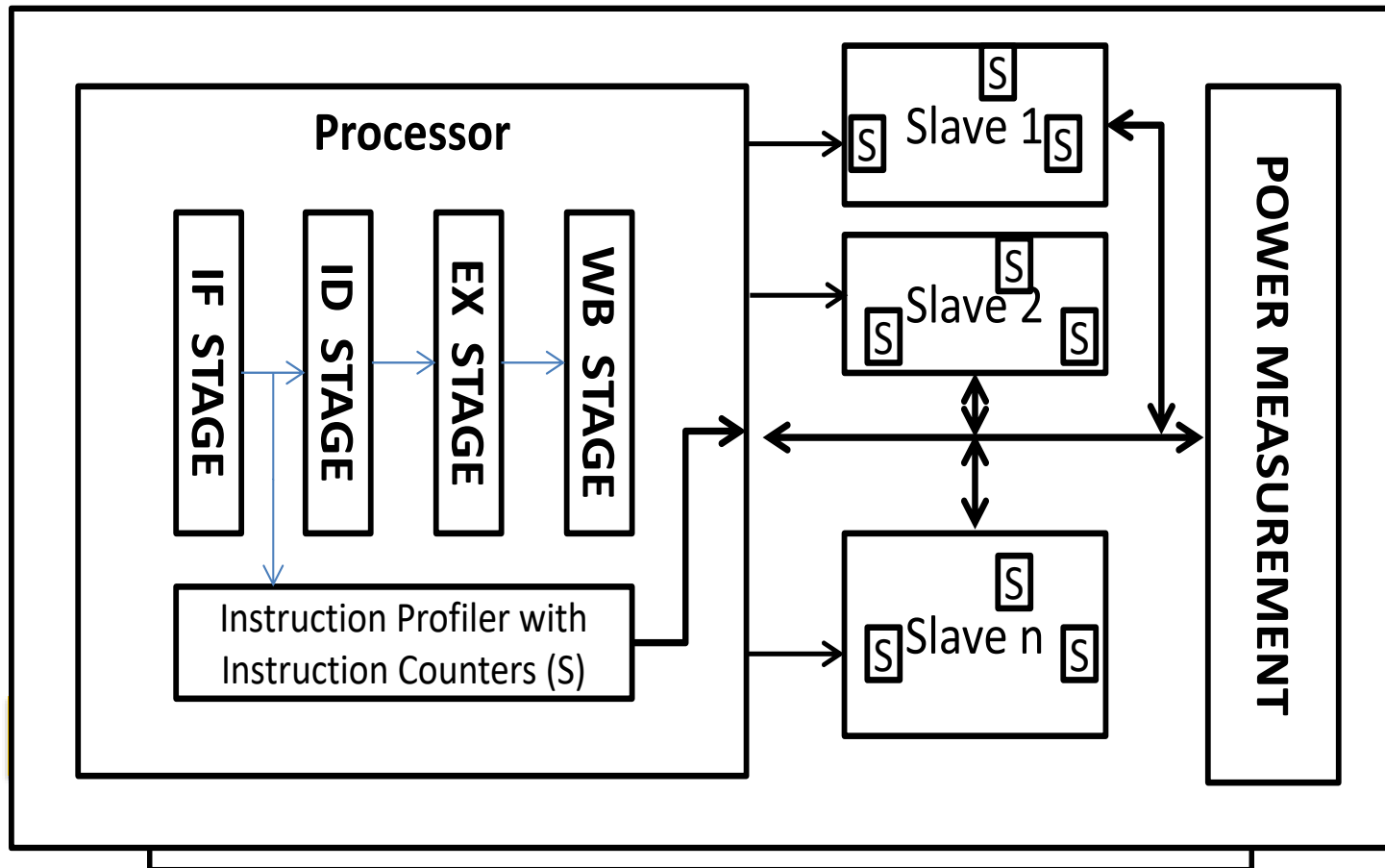
Solution :

- What is the proxy for activity within CPU ?

Answer : Instructions

- Can I use these instructions to measure power accurately ?

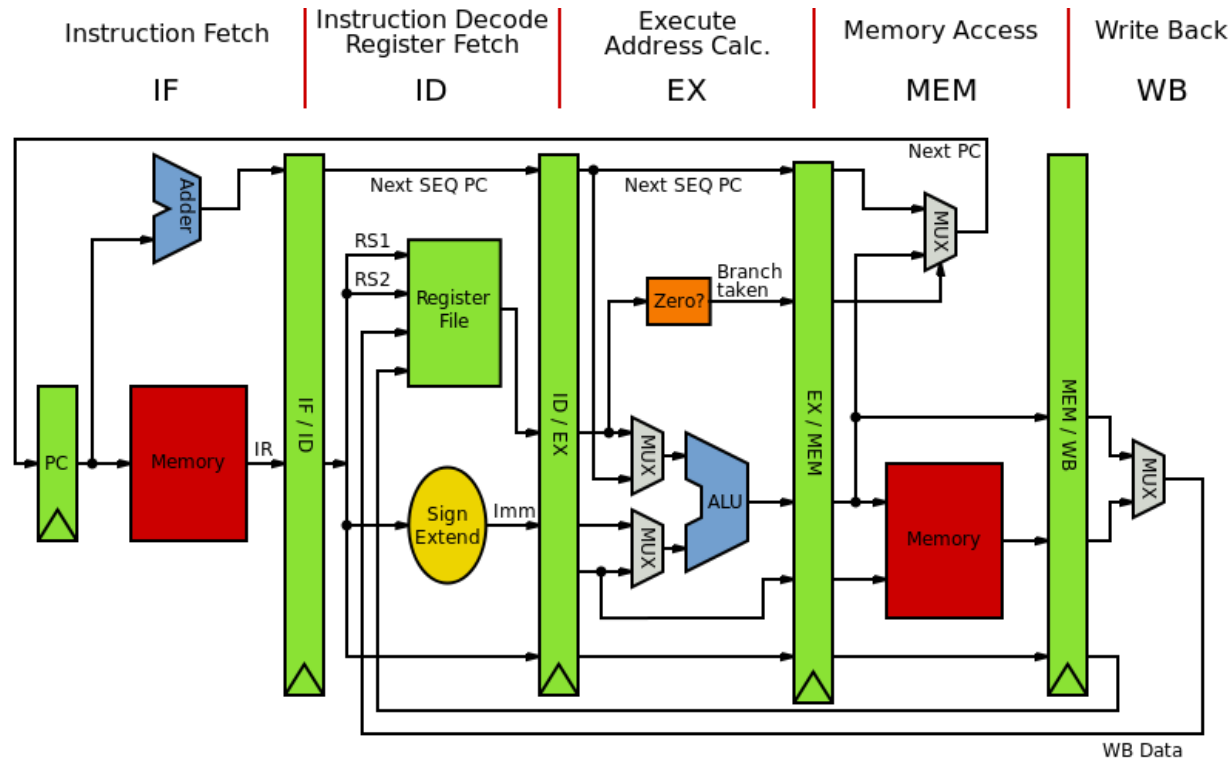
Accounting Activity of SoC



Summarizing the Activity as temporal and spatial activity
Summarizing The spatial activity

Online Estimation of Energy Per Instruction of a processor

- Weights represent contribution of instructions to power.



Integrating Peripheral Sensors of an SoC into Instruction profiler

```
C:\Users\student\Downloads\OpenRISC\orpsocv2\sw\drivers\ethmac\include\ethmac.h - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
ethmac.h int.h orpsoc_top.v

128 OETH_RX_BD_INVSIMB | \
129 OETH_RX_BD_DRIBBLE | \
130 OETH_RX_BD_TOOLONG | \
131 OETH_RX_BD_SHORT | \
132 OETH_RX_BD_CRCERR | \
133 OETH_RX_BD_LATECOL)
134
135 /* MODER Register */
136 #define OETH_MODER_RXEN 0x00000001 /* Receive Enable */
137 #define OETH_MODER_TXEN 0x00000002 /* Transmit Enable */
138 #define OETH_MODER_NOPRE 0x00000004 /* No Preamble */
139 #define OETH_MODER_BRO 0x00000008 /* Reject Broadcast */
140 #define OETH_MODER_IAM 0x00000010 /* Use Individual Hash */
141 #define OETH_MODER_PRO 0x00000020 /* Promiscuous (receive all) */
142 #define OETH_MODER_IFG 0x00000040 /* Min. IFG not required */
143 #define OETH_MODER_LOOPBCK 0x00000080 /* Loop Back */
144 #define OETH_MODER_NOBCKOF 0x00000100 /* No Backoff */
145 #define OETH_MODER_EXDFREN 0x00000200 /* Excess Defer */
146 #define OETH_MODER_FULLLD 0x00000400 /* Full Duplex */
147 #define OETH_MODER_RST 0x00000800 /* Reset MAC */
148 #define OETH_MODER_DLYCRCEN 0x00001000 /* Delayed CRC Enable */
149 #define OETH_MODER_CRCEN 0x00002000 /* CRC Enable */
150 #define OETH_MODER_HUGEN 0x00004000 /* Huge Enable */
151 #define OETH_MODER_PAD 0x00008000 /* Pad Enable */
152 #define OETH_MODER_RECSCALL 0x00010000 /* Receive Small */
153
154 /* Interrupt Source Register */
155 #define OETH_INT_TXB 0x00000001 /* Transmit Buffer IRQ */
156 #define OETH_INT_TXE 0x00000002 /* Transmit Error IRQ */
157 #define OETH_INT_RXF 0x00000004 /* Receive Frame IRQ */
158 #define OETH_INT_RXE 0x00000008 /* Receive Error IRQ */
159 #define OETH_INT_BUSY 0x00000010 /* Busy IRQ */
160 #define OETH_INT_TXC 0x00000020 /* Transmit Control Frame IRQ */
161 #define OETH_INT_RXC 0x00000040 /* Received Control Frame IRQ */
162
163 /* Interrupt Mask Register */
164 #define OETH_INT_MASK_TXB 0x00000001 /* Transmit Buffer IRQ Mask */
165 #define OETH_INT_MASK_TXE 0x00000002 /* Transmit Error IRQ Mask */
166 #define OETH_INT_MASK_RXF 0x00000004 /* Receive Frame IRQ Mask */
167 #define OETH_INT_MASK_RXE 0x00000008 /* Receive Error IRQ Mask */
168 #define OETH_INT_MASK_BUSY 0x00000010 /* Busy IRQ Mask */
169 #define OETH_INT_MASK_TXC 0x00000020 /* Transmit Control Frame IRQ Mask */
170 #define OETH_INT_MASK_RXC 0x00000040 /* Received Control Frame IRQ Mask */
```

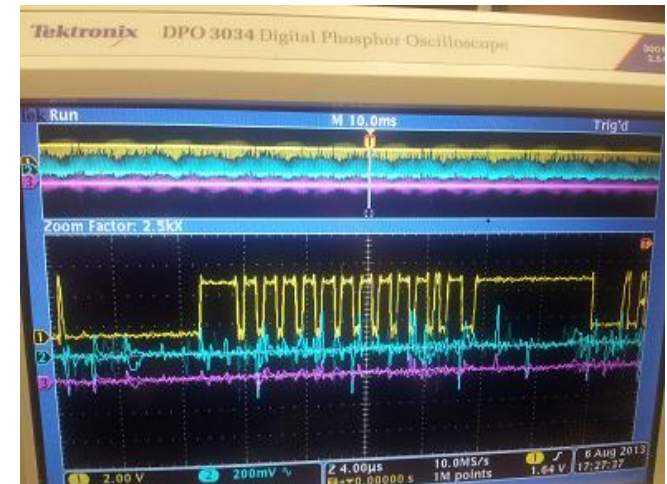
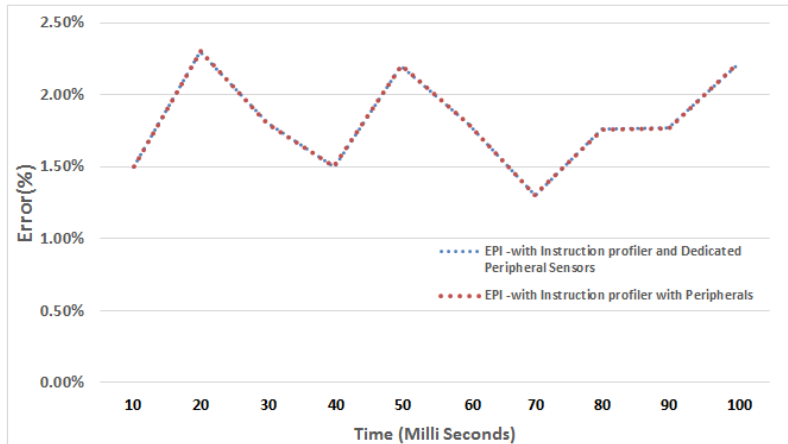
```
74 /* Ethernet buffer descriptor */
75 typedef struct _oeth_bd {
76 #if 0
77     ushort len; /* Buffer length */
78     ushort status; /* Buffer status */
79 #else
80     uint len_status;
81 #endif
82     uint addr; /* Buffer address */
83 } oeth_bd;
84
85 // from board.h
86 #define ETH_BASE_ADDR ETH0_BASE
87
88 #define OETH_REG_BASE ETH_BASE_ADDR
89 #define OETH_BD_BASE (ETH_BASE_ADDR + 0x400)
90 #define OETH_TOTAL_BD 128
91 #define OETH_MAXBUF_LEN 0x600
92
93 /* Tx BD */
```

```
C:\Users\student\Downloads\OpenRISC\orpsocv2\board\winvalley\sw\board\include\board.h - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
ethmac.h int.h orpsoc_top.v board.h

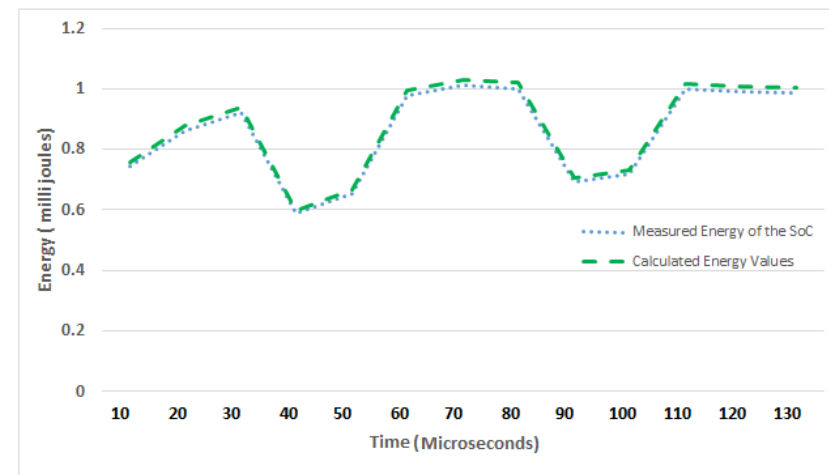
22 // Offset 0x100000
23 #define BOOTROM_ADDR_BYTE2 0x1c
24 #define BOOTROM_ADDR_BYTE1 0x00
25 #define BOOTROM_ADDR_BYTE0 0x00
26 // Causes SPI bootloader to loop if SPI didn't give correct size of image
27 #define SPI_RETRY_IF_INSAME_SIZEWORD
28
29 //
30 // Defines for each core (memory map base, OR1200 interrupt line number, etc.)
31 //
32 #define SDRAM_BASE 0x0
33
34 #define GPIO_0_BASE 0x91000000
35
36 #define UART0_BASE 0x90000000
37 #define UART0_IRQ 2
38 #define UART0_BAUD_RATE 115200
39
40
41 #define SPI0_BASE 0xb0000000
42 #define SPI0_IRQ 6
43
44 #define I2C_0_BASE 0xa0000000
45 #define I2C_0_IRQ 10
46
47 #define I2C_1_BASE 0xa1000000
48 #define I2C_1_IRQ 11
49
50 #define ETH0_BASE 0x92000000
51 #define ETH0_IRQ 4
52
53 #define ETH_MACADDR0 0x00
54 #define ETH_MACADDR1 0x12
55 #define ETH_MACADDR2 0x34
56 #define ETH_MACADDR3 0x56
57 #define ETH_MACADDR4 0x78
58 #define ETH_MACADDR5 0x9a
59
60 //
61 // OR1200 tick timer period define
62 //
63 #define TICKS_PER_SEC 100
```

- Peripherals are accessed by load or stores to a pre-defined memory address.
- The hardware Profiler matches these instructions and addresses to account for the peripherals.

Initial Results on EPI



- Using the partitioning tool , we needed 150% additional hardware. We now have reduced it to less than 1% of the total SoC



Drawbacks of EPI

- Instruction is conceptual and are not mapped into physical component.
 - Hard to use it to assist thermal management

Accurate Sub-circuit Physical level power measurement in Processors/SoC

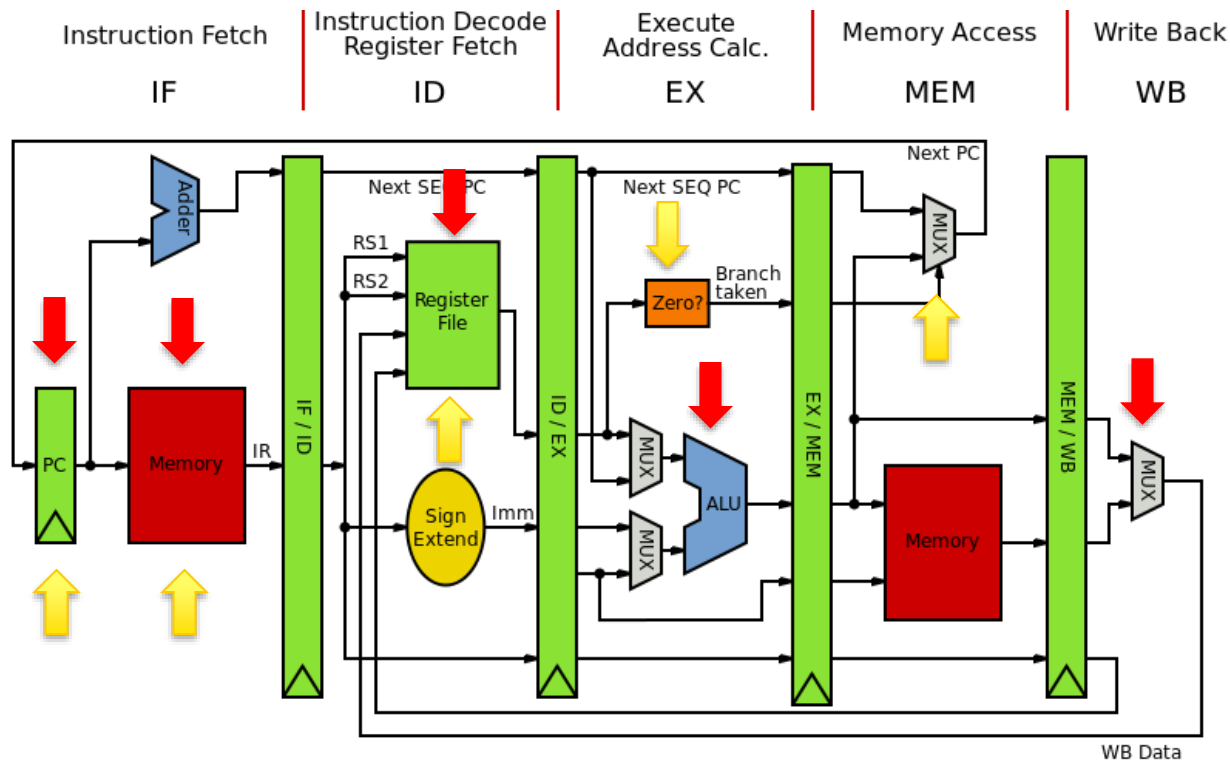
Problem : How do I measure physical level sub-component power accurately without instrumenting additional counters?

Solution :

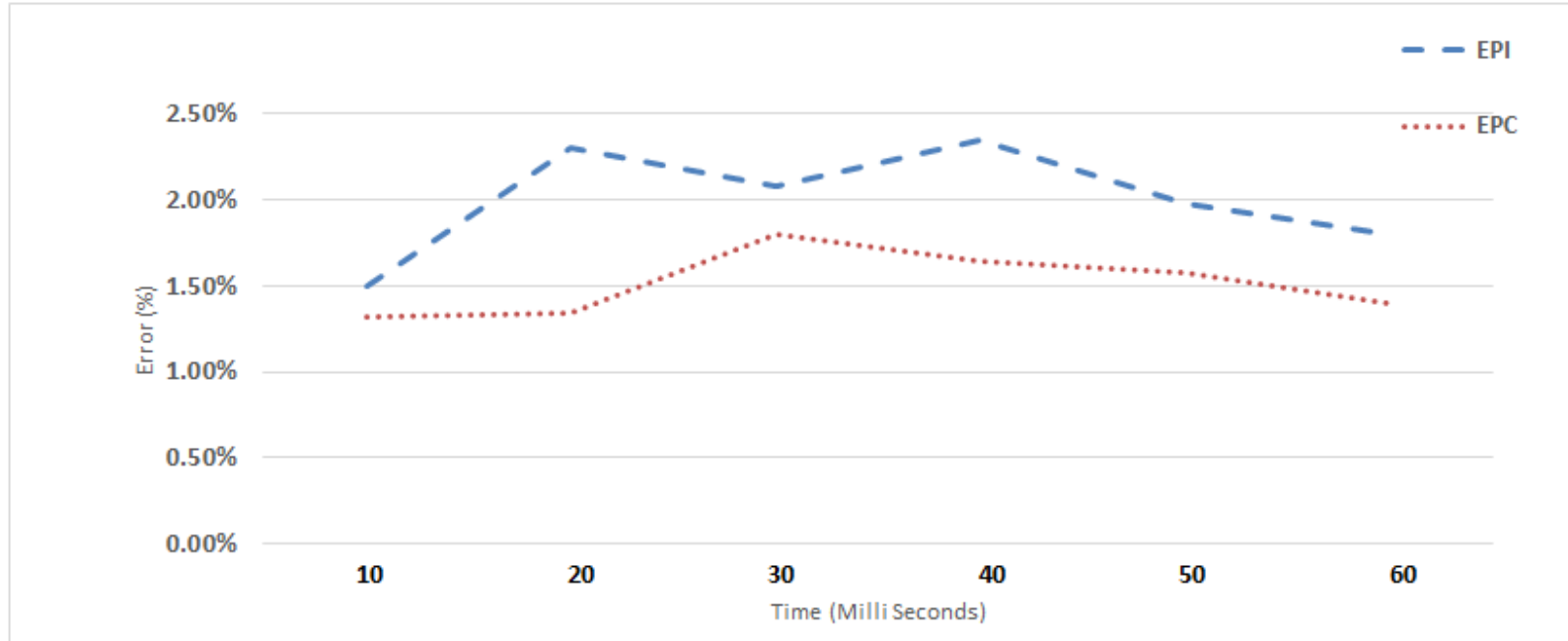
- We explored the possibility by looking into processor itself
- By doing three levels of manual mapping and we were able to map 80 instructions to 55 components.

Energy Per Component of a Processor

- Perform analysis on processor instruction execution paths.
 - Find the number of components at the lowest granularity that gives same accuracy.

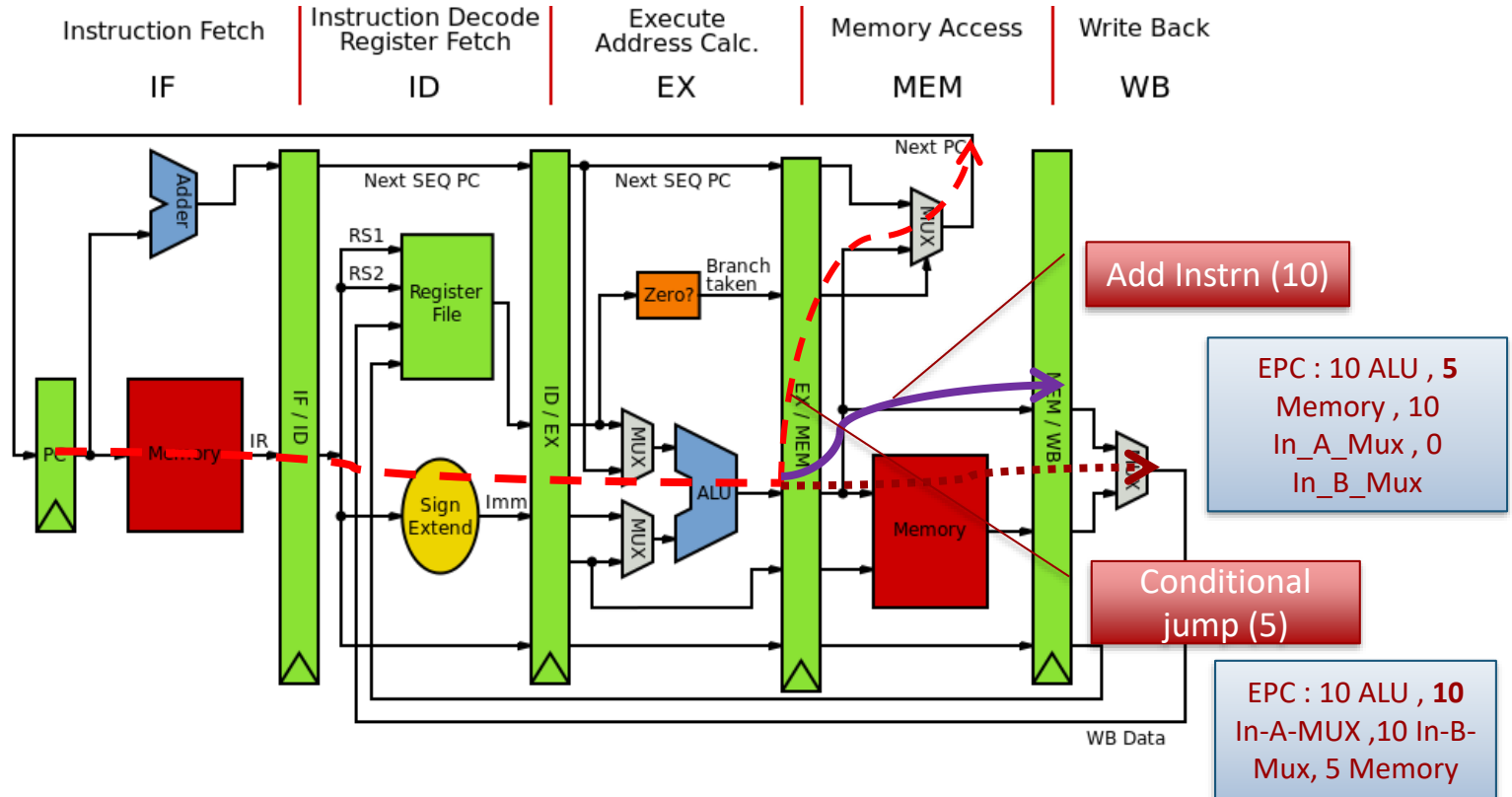


EPC results



- Reduced 80 instruction counters to 55 Component counters
- Replaced 70 dedicated peripheral counters with 12 counters in the instruction profiler

Comparison of EPI and EPC



- EPI Produces an average weights associated with switching at the mux and memory.
- EPC accounts for these switching – hence more accurate

Drawbacks of EPC

- It is we difficult to extract different layers of components for EPC for a very complex design.
- Instruction overlaps not considered.

Improving Accuracy of estimation without increasing overhead

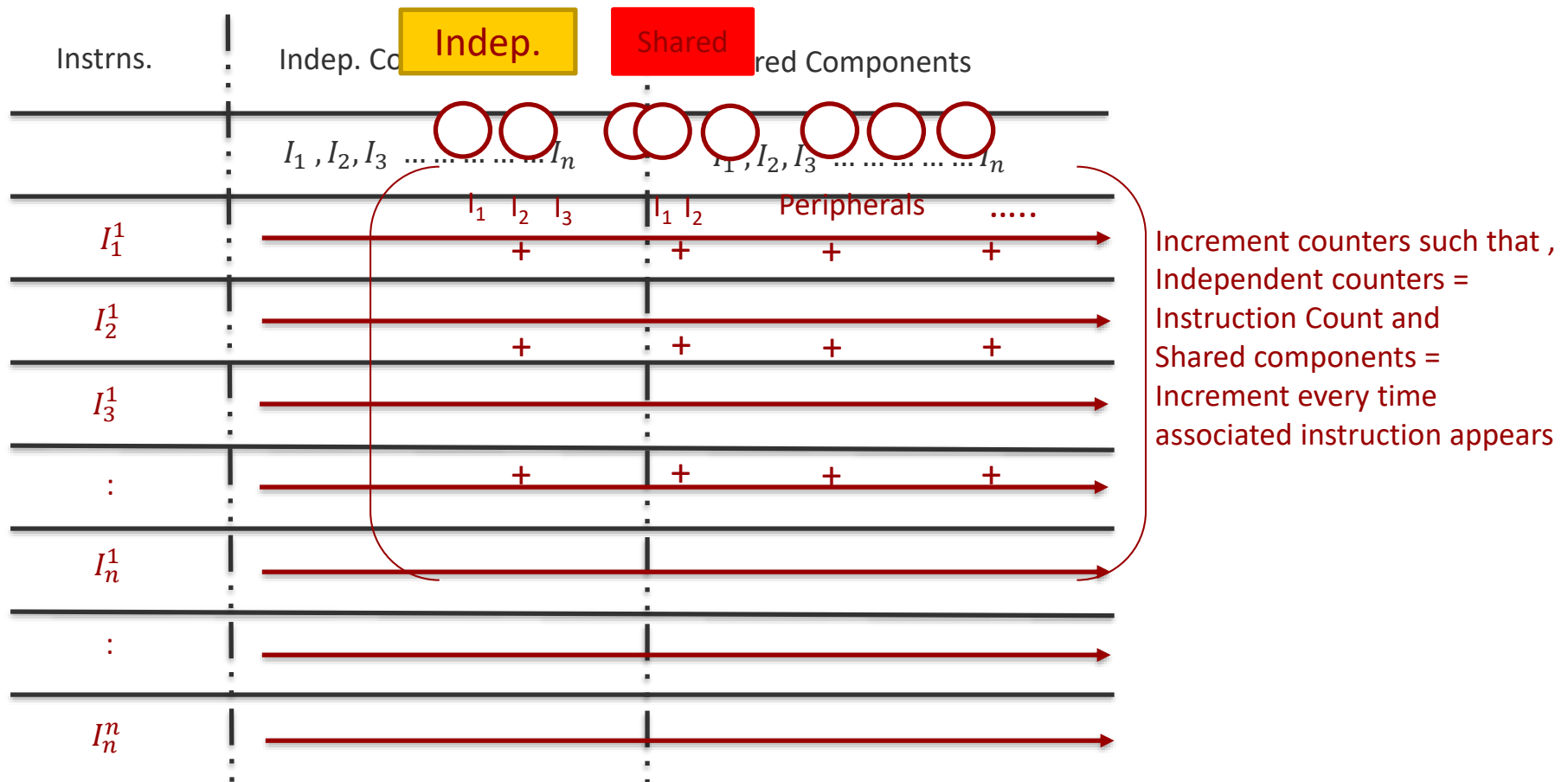
Problem : How do I improve the accuracy with the same overhead?

- How do I account for various execution overlaps ?

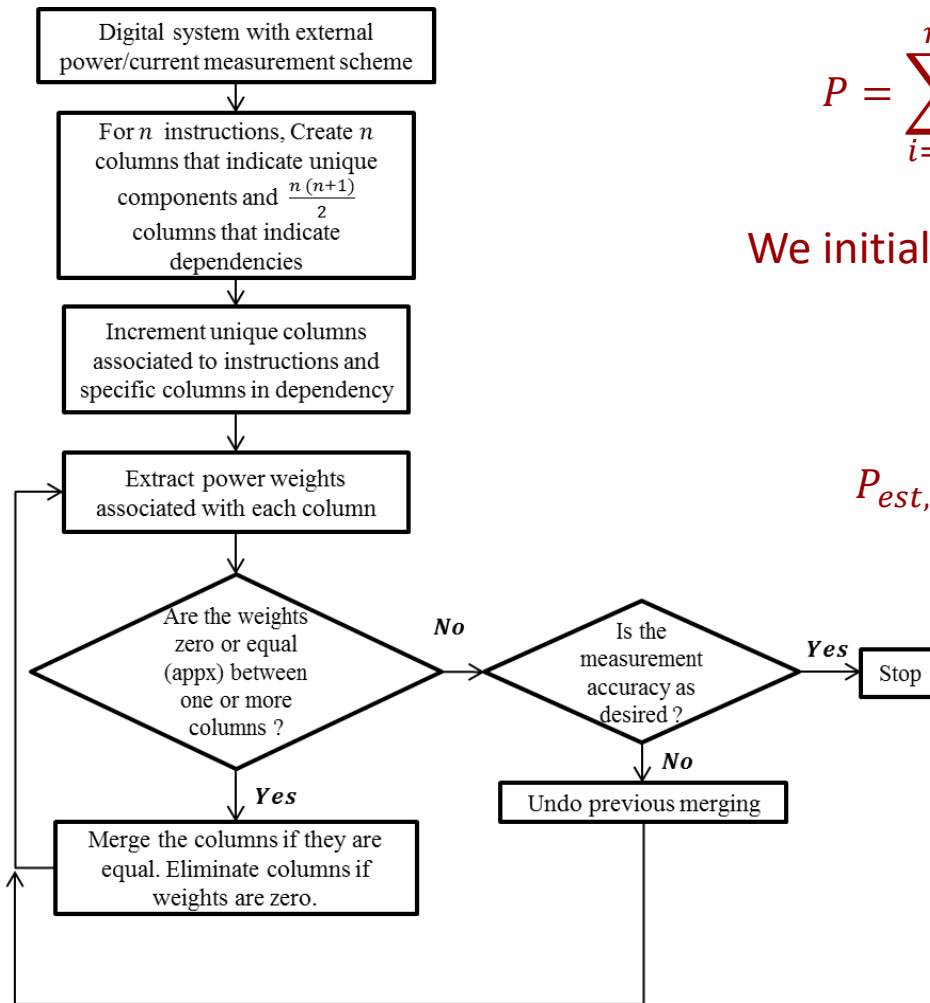
Ans : Implicit enumeration of shared components

- Can I still use instruction counts to measure power accurately ?

Automated Energy per component – The concept



Automated Energy per component



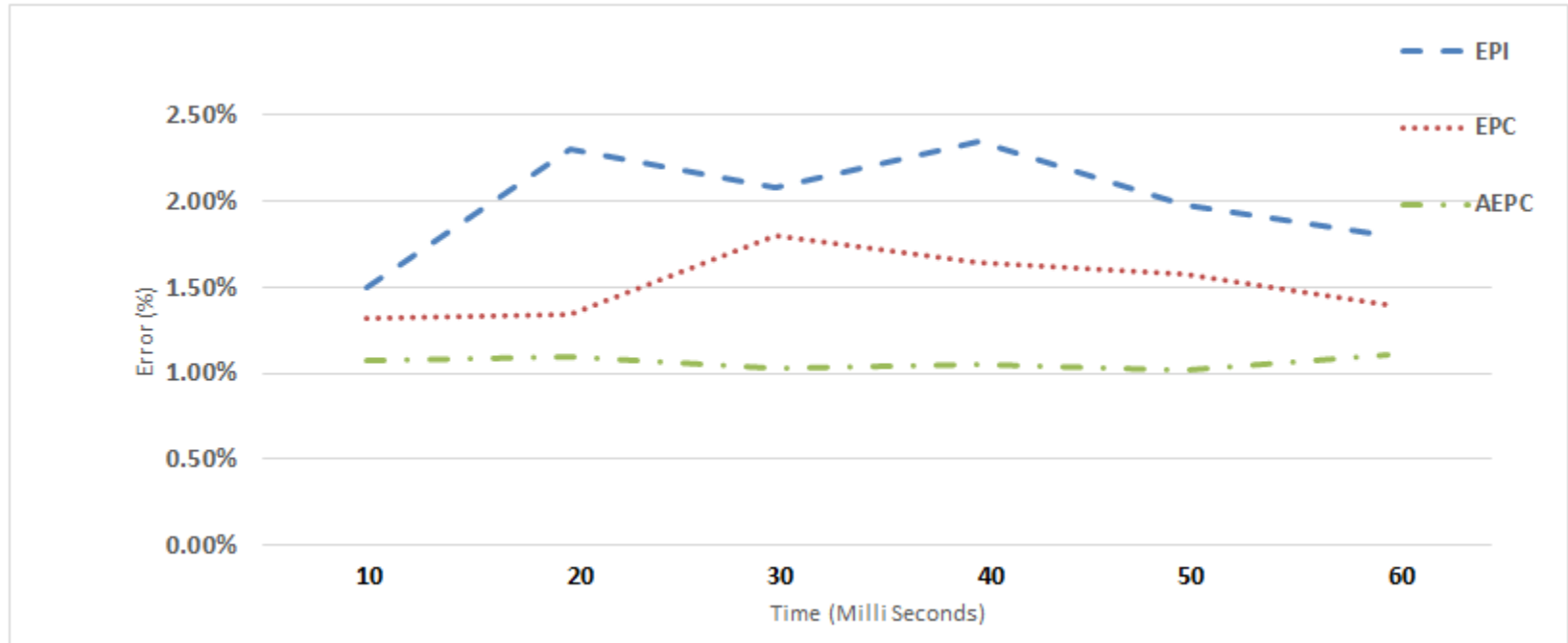
$$P = \sum_{i=1}^n a_i w_i + \sum_{j=1}^{\frac{n(n+1)}{2}} a_j w_j + \sum_{k=1}^m a_k w_k + P_{static} + R$$

We initially assume that, $\exists C_z$, such that, $C_z \in I_x \cap I_y$

$$P_{est,C1} = \sum_{p=u} (tr_p \cdot W_p) \left\{ \begin{array}{l} = 0, C_z \text{ does not exist } (W = 0) \\ = P_{est,Cv}, \text{ if } C_z \text{ is equally shared between multiple instructions} \\ \text{Unique} \end{array} \right.$$

The weights are means to perform implicit enumeration of component sharing.

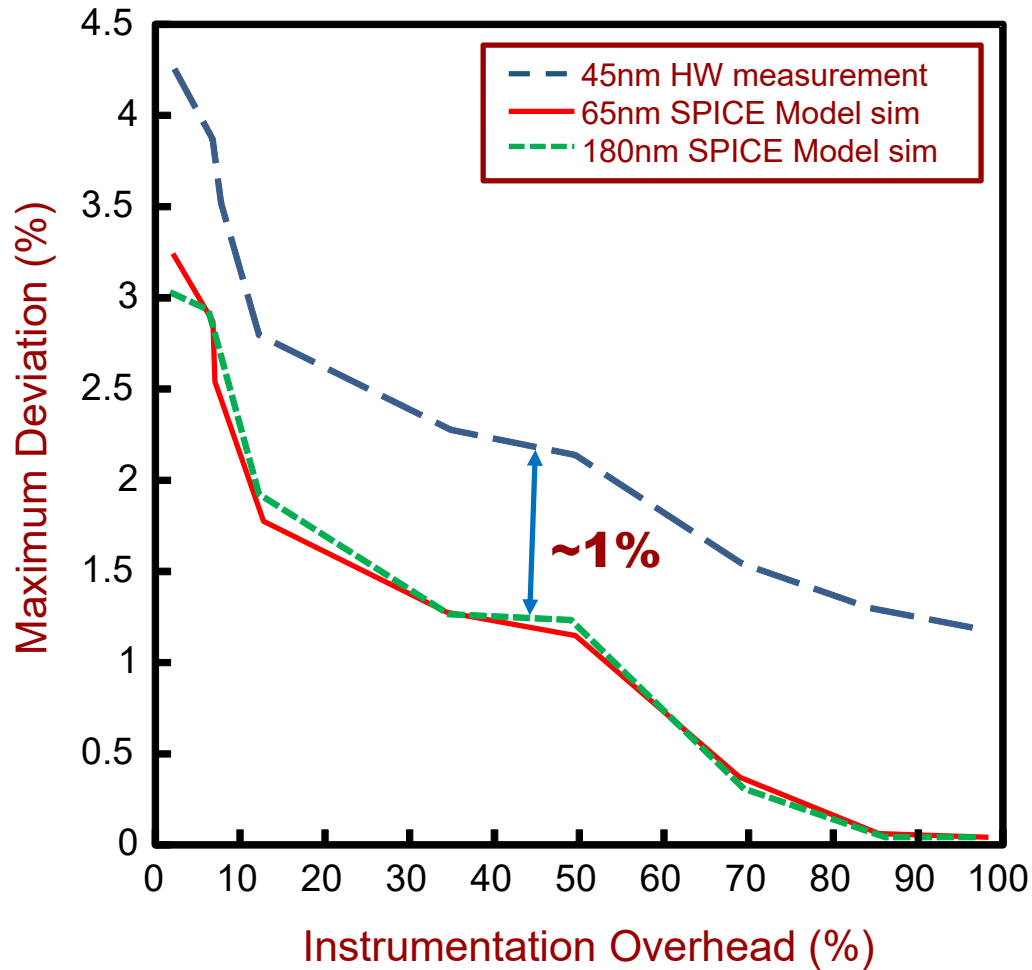
AEPC Results



We compare the weights of components from EPC than those of AEPC.
(based on the sharing information in AEPC).

For ALU and register file , the difference between power estimated by AEPC and EPC $< 0.5\%$

Comparison of simulator and hardware results



Accelerated Accurate Simulator



Circuit	Number of Gates	Comm. SPICE (ref)		Comm. Accelerated. SPICE			Our method		
		Simulation Time (s)	Memory (MB)	Simulation Time (s)	Memory (MB)	Accuracy (%)	Simulation Time (s)	Memory (MB)	Accuracy (%)
C432	163	20412.69	927.24	2886.72	274.93	99.971	24.59	7.90	99.999
S298	240	23790.45	1082.26	3364.45	320.42	99.971	30.07	8.20	99.999
C880	383	14760.83	670.90	2087.52	199.81	99.971	11.82	6.07	99.999
C1908	2092	172808.22	7904.09	24439.01	2326.92	98.150	286.46	32.00	99.999
Microctr.	50000	310320.00	14000.00	43887.00	4000.00	98.003	9092.40	49.00	99.999

Results on Simulation of various benchmark circuits with Comm. SPICE, Accl. SPICE and our method

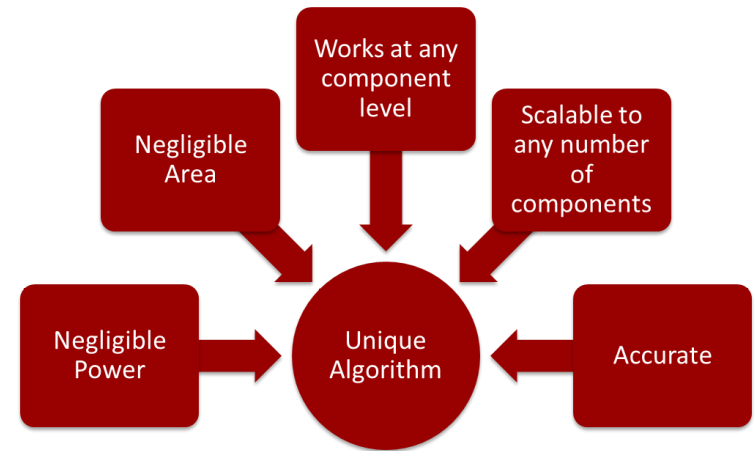
277 x over reference for the same accuracy

Accuracy (%)	Raw Convergence				Convergence with seed value				Speedup in conv. using Accl. Sim for training		Comparison of Weights from Comm. SPICE and Accl. SPICE
	Comm. SPICE		Accl. Sim		Comm. SPICE		Accl. Sim		Raw	Seeded	Same Values
	Samples	Time (mins)	Samples	Time (mins)	Samples	Time (mins)	Samples	Time (mins)			
99.999	19055	953.50	19070	121.19	66	3.12	102	0.59	7.88x	5.29x	
99.7	4120	210.00	4157	30.35	63	3.10	89	0.53	6.92x	5.85x	Same Values
98	2230	122.00	2283	18.01	55	3.02	71	0.47	6.77x	6.43x	Same Values
97	1775	69.00	1810	14.51	43	1.56	57	0.38	4.76x	4.10x	Same Values

Results on rate of convergence with Comm. SPICE, Accl. SPICE and our method

Summary

Goal : To develop an accurate method to measure the power consumption of components on-chip which is fine-grained, and low overhead.



Method : By observing a summary of internal circuit activity along with our online solver and extract the best model for the device by accurate model collapsing

Result : A Power measurement system that is :

- Accurate
- Fine granularity
- Low overhead