LPD

0. Introduction

Why LPD? 1. Batery life 2. Decrease Aging
3. Increase reliability 4. Cooling & cost

Aging phenomenon reduces lifetime of the chip
Introduced by technology scaling
Thermal issues increase aging, degrade performance,
increase leakage power, necessiate expensive cooling
55% of all electronic failure is caused by temperature issues

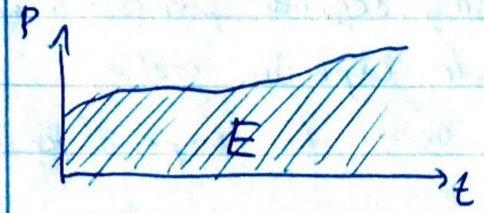
Design of LPD serves for
1. → Portable Systems, reduce energy drain
to increase battery-life & decrease weight
batteries do not scale with power demand
2. → Thermal considerations, more transistors & higher CLK
increase problem rates
+10°C ~ 2x failures, cooling requirements, packaging cost
3. → Environmental Concerns, Green PC
4. → Reliability issues, direct relation to power draw
Electro migration, IR drops, inductive effects

Power vs Energy
Minimization of power is used for: design of power supply
" " VRM
" " interconnect
" " short term cooling
" " Enegy " " ": maximize computations with
given amont of available energy
cooling, long term low temperetures

Power = $P_{switching\_capecitance} + P_{Leakage} + P_{short\_circuit} + P_{static}$

$P_{switching\_capacitance} = 1/2 \cdot C_L \cdot Vdd^2 \cdot N \cdot f$

Relationship between Power & Energy:



$E = \int P \, dt$

Power eficient systems may not be energy efficient

Jevon's Paradox:
efficiency ⇒ more total use
Switching capacitance eficiency improved by factor > 1000
⇒ more power hungry applications emerge

Dennard Scaling
power density no longer constant $P = 1/2 \, C \cdot V^2 \, N \cdot f$
no longer scaling of Vdd^2 possible
still need for more ~~computation~~ computation power: use more core
Thermal gradients: spatial / temporal : goal: find balance

Energy vs programmability
ASIC is very eficient, not programm able
... FPGA ... ASIP ... GPCPU

Dark Silicon: used cores heat up unused ones →
they can't be used at the same time

Landauer principle:

1 bit operation must at least consume 0.69 kT of energy

= 0.0172 eV @ 20°C ≈ 1.602 × 10^{-21} Joule

Levels of power optimization:

Application → Compiler → OS → Micro-Arch. → Circuit → Transistor

1. Energy sources

Batteries don't scale with power demanding components

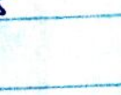
Human power sources:

Breathing, Body heat (both inefficient)

Blood pressure, vibrations from motion

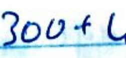
used in watches

moving/walking human costs 300+ W



Piezoelectric: like pressing two capacities onto each other

Capacitive: Change in capacitance causes either voltage or charge increase



Mass
Magnet

Inductive: Coil moves through magnetic field causing current in wire

Other sources: Li-Ion Battery, Zinc-Air, Solar cells, Vibration, Sound, Fuel cell

Swanson's Law:

Solar cell price drop 20% / doubling of cumulative cells shipped

15 mW/cm^2 @ direct sun BUT: current source with optimum point for maximum power extraction

→ battery needed to store harvested energy, cannot directly supply ES or IC

Fuel cells

efficiency ~ 40% - 60% , clean (H_2O out)

Alkaline fuel cell ~ 70% efficiency

Super Capacitors

Capacitor w/ high ~~dens~~ capacitance (~ 100x)

+ high power density (W / kg)

+ long life + high cycle efficiency

+ ~~much~~ many cycles / life

+ no danger of overcharge

- low energy density (Wh / kg)

- expensive

- high self-discharge

- high leakage

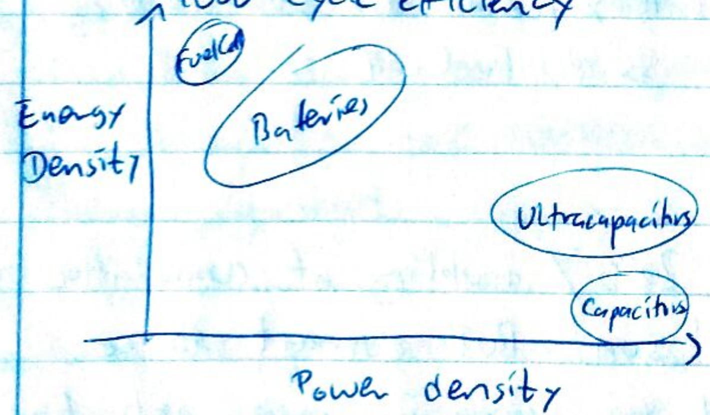
Batteries

+ Energy density $\frac{V \cdot I \cdot h}{m}$

- Power density $\frac{V \cdot I}{m}$

- lifecycles / recharge count

- low cycle efficiency



Hybrid - Electric - Storage - System

charge - transfer - interconnect combines multiple batteries / capacitors / ...

2. Battery Modelling

Full charge capacity :=
Remaining capacity of a fully charged battery at beginning of a discharge cycle

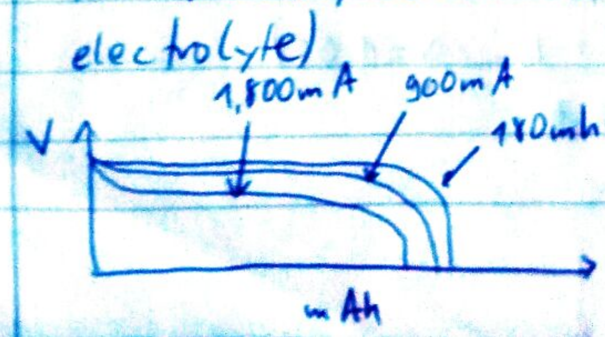
Full Design capacity :=
Capacity of a newly fabricated batery

Theoretical Capacity :=
Maximum amount of charge that can be extracted from a battery based on the amount of active materials (chemicals) contained

Standard Capacity :=
Amount of charge that can be extracted from a battery when discharged under standard load and temp. conditions

Actual capacity :=
Amount of charge the battery delivers under applied load and given temperature.

Rate Dependent Battery Capacity
current @ electrolyte, if too high, diffusion of active species in the battery can't keep pace.
concentration gradient builds up along the electolyte
cut-off, remaining active species can't be accessed by the consumer
lower discharge rates reduce this efect, more room for recovery (diffusion of available species to the electrolyte)



Temperature effects

too low: chemical activity bad, internal resistance high, lower full-charge capacity, faster discharge

too high: begin of self-discharge

Fading of battery Capacity

every cycle reduces full-charge-capacity

reason: side-effect of chemical reaction

→ Electrolyte decomposition

→ Active material dissolution

→ passive film formation

Irreversible

Reduced capacity in short-term, failure in long-term

How to reduce: shut-off before critical-low-level

Battery models @ design time allows:

maximation of energy draw, longer runtime before recharge

optimization of battery life time

correct dimension of battery for a given system

What accuracy is needed?

Trade-off: accuracy vs. computation time

#parameters →

Peukert's Law (example for battery model)

normalized capacity (1A) = $t_{run} * I^{\alpha}$

α = discharge rate (Li-Ion: $\alpha = 1.05$)

temperature dependent

Battery ~~Em~~ Emulation

Goal: full vepro ducibility

emulation of battery with other hardwere

Observed voltage, $V_b = V_{oc} - I R_1$

@ discharge: V_{oc} decreases and R_1 increases
depending on temp & battery state

Discrete time battery Models

using VHDL to bring electrical level into high-level-sim.

Applications: Battery aware scheduling
" " supply design
Load-profile shaping for multi-battery Systems
- static switching after time for recovery
- dynamic: monitor status of batteries

Rechargeable batteries have non-ideal effects like
recovery
temperature dependence
capacity depending on discharge rate

modelling possible, if factors known

tradeoff between accuracy & sim. time

models can be deployed & simulated for estimations & optimizations

overall-goal: increase system's runtime and reduce recharge rates

Self-discharge due to chemical side reactions, internal short-circuits

~ charge   Li-Ion ~ 2%/month

~ temperature   NiMH ~ 15% /month

Stochastic battery model

idea: Battery Life-time estimation of HW/SW ES

Exploration of design space fastly done w/o accuracy loss

Definitions

charge unit := smallest amount of charge that can be discharged

T := # max. available charge units

N := nominal capacity of charge units

N, T vary with battery technology, discharge current, ...

State of charge is tracked via discrete time transient stochastic prozess

using a probabilistic FSM

allows modelling of ~~stat~~ idle states → recovery

Battery aware Scheduling

Goal: extend battery Lifespan

Means: schedule transformations

$$P^{act} = \int dI \, \frac{V \cdot I}{c(I)} \cdot \hat{P}(I)$$

allows:

t1(5)	t3(1)	t4(1)	t5(5)
		e1(1)	
t6(5)	t7(1)	t2(1)	t8(5)
10	2	2	10

15% more efficient

t3(1)	t1(5)	t4(1)	t5(5)
			e1(1)
t6(5)	t7(1)	t8(5)	t2(1)
6	6	6	6

Variable Voltage Scheduling

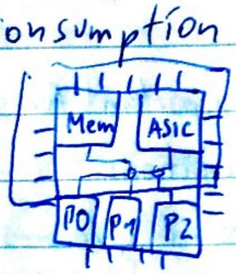
lower voltage if op has time until deadline left

Same computation, more evenly energy drain

swapping ops might bring more benefit, if deadlines can still be met.

3. HW - Power Optimization & estimation

Power consumption in HW:



Interconnect / RAM / ASIC / CPU

Generic HW synthesis flow: System-Level Design → HLS → Logic Synthesis → Layout

W/ power optimizations: accompanied by System-level power analysis & Libraries for that →

Architecture level power analysis @ RTL & Libraries →

Logic level power analysis @ logic synthesis & Libraries →

Transistor power analysis @ Layout

$$P_{avg} = P_{switching\_capacitance} + P_{Leakage} + P_{short\_circuit} + P_{static}$$

$$= \tfrac{1}{2} C_L V_{dd}^2 N f + K(V_{dd} - 2V_T)^3 T N f + (I_{subthresh} + I_{oxide} + I_{diode}) \cdot V$$

frequency

cumulative parasitic capacitance

supply voltage

#expected transitions per cycle

constant (tech)

threshold Voltage

input rise/fall time

I_{diode}: diodes formed between diffusion & substrate

I_{oxide}: electrons tunneling through the gate oxide

$I_{subthreshold}$: $K W_{eff} \cdot e^{\frac{V_{in} - V_{T_n}}{S}}$

Operator scheduling

assignment of operations given in the behavioral description to a cycle

behavioral description → sequence of operations performed

⇒ allocation & possible resource sharing

exploiting of regularity of programs / the given schedule

exploit slack or mobility of operations

⇒ schedule them on slower, more power efficient hardware (module selection)

⇒ try to reduce peak power. Important for packaging, cooling & reliability considerations

Algo for finding optimal voltages for a given schedule:

```
1. init : all voltages to max
2. compute slack
3. find max, if 0, done.
4. DFS on all max found
    4.1 create dual graph by adding edge between all
        unconnected nodes in DFS-tree, if their order of
        traversion is inversed.
5. Weight assignment   $W = (V_{ck}^2 - V_{ck+1}^2)$
6. Longest weighted path
7. reassign voltages in longest weighted path : $V_{ck} \rightarrow V_{ck+1}$
8. goto step 2
```

Module Selection

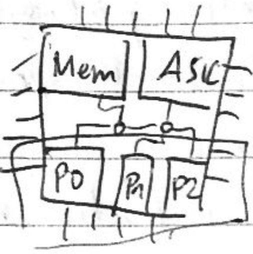
example: + done by ripple-carry / Carry-lookahead -..

trade-offs @selection of specific instances possible

not always are the fastest components necessary to meet the timing constraint. in that case : choose a slower one. if

the timing constraint is still met & it is more efficient
often the case in non-critical path of CDFG

4. SW-Power optimization & estimation



Applications, OS, Compiler

program power model:

$$\text{cost} = \sum_i (\text{Base}_i \cdot N_i) + \sum_{i,j} (\text{overhead}_{i,j} \cdot N_{i,j}) +$$
$$N_{CM} \cdot \text{penalty}_{CM} + N_{stall} \cdot \text{penalty}_{stall}$$

used for simulation of code power consumption to give an estimation

alternative: apply ampere-meter to CPU & put program into loop

Compiler for power optimization:

- register optimizations
- reduce execution time
- optimizations by hand

Instruction dependent power consumption of CPU
Cost sums up: internal busses carrying immediate values, register numbers, and the instruction address

Data-dependent power consumption of CPU
Cost sums up: n data accesses and their address, the data itself & their direction (read/write)

Instruction scheduling

use instruction-level energy costs to guide code generation

minimize memory accesses

reduce context saving

CPU-specific: dual memory loads / instruction packing

optimize instruction schedule such that activity in specific parts of the system is reduced

Like instruction bus, processor-memory bus, instruction register & register decoder

Traditional compilers try to increase performance by reducing # of pipeline-stalls

For low-power instruction scheduling:

use switching activity as metric

simulate RTL model of processor and measure switching activity on busses when running 1 (or n) command(s).

Use these profiled values as a cost metric

Assumption: there is leeway that allows reordering

Reordering may cost performance, needs to be taken into account.

Cold Scheduling

$S(I_j, I_{j+1})$ Switching activity, if I_{j+1} follows I_j

$BS = \sum S(I_j, I_{j+1}), j = 0, 1, \dots n-1$ Block switching activity

$Cost = 1/k (W_1 \cdot BS_1 + \dots + W_k \cdot BS_k)$ cost function, k = # of BB

problem: difficult to obtain bit switching activity from symbolic representation

jump/branch targets may not be known before scheduling and register allocation

sizes of BB may change during scheduling & reg. allocation

binary representation of indexes to symbol table may not be available

⇒ Phase problem of instruction scheduling and assembly
- If scheduling precedes assembly: may reduce potential of reducing bit switches
- If assembly precedes scheduling: flexibility of schedule limited

One solution: need to estimate binary representation of an instruction

use cases showed, that there might be no correlation between energy / power - savings and performance loss

Space for instruction schedules: w/o precedence constraints:

$(N-1)! / 2$ N = # instructions

efficient approach to the instruction scheduling ~~algorithm~~ problem

given: table with power consumption, if an instruction is followed by another

given control dependency graph

generate Weighted Strongly connected Graph by assigning values to edges. Add weighted edges, if precedence of two instructions isn't given SCG (strongly connected graph)

generate Minimum spanning tree (MST)

use simulated annealing to find Hamiltonian path (exact solution NP-hard)

⇒ energy eficient instruction schedule

other ideas on power optimization:

eliminate dead code / combine common sub-expressions / memory hierarchy optimizations: loop tiling / keeping data on chip

doesn't alway affect performance, for example: optimization not on the critical path

Compiler-driven DVS

DVS $\hat{=}$ dynamic voltage scaling

most eficient due to quadratic impact in power formular

apply DVS when it has no impact on performance

find the best scaling points in code to adjust voltage

Overhead: switching to an from new voltage settings cost time & power ⇒ may reduce or eliminate potential savings

~ 100 s of μs, even longe than a ~~c~~hache-miss, can't be used for woltage swap

intra-task DVS: scaling points may be in the middle of task execution (opposite: inter-task DVS)

- sub categories:
  1. interval-based DVS: fixed length time intervals rely solely on state of the system and trace history. Scaling points determined online or offline
  2. checkpoint-based DVS: scaling points are determined offline, scaling factors are determined online. Scaling points are placed at selected branches to exploit the slacks due to runtime variations.

Compiler directed DVS:

Find region R in program P and frequency f ~~[illegible]~~ such that P \ R is executed at peak_frequency f_max, the total execution time + switching overhead (T_trans · 2 · N(R)) is nomore increased than a factor r, while the total energy usage is reduced

SW power estimation is possible & faster than estimating power consumption on circuit level. Compiler may include optimizations: Instruction scedulin & Intra-procedural DVS, these are distinct tasks to performance optimization.

5. Thermal Aware Design

Circuit must fulfill constraints

- +50°C painful for users (especially @ mobiles)
- fire hazards / short-ciruit
- melting insulation
- mechanical stress

Reliability & performance degradations

Short-term effects:

- Leakage-current increases
- Reduction of driving current

Long-term effects:

- Aging / Degradation of transistor

$$I_{Drain} \approx \frac{1}{2} \mu C_{ox} W/L (V_{gs} - V_{th})^2$$

Lower @ high T

μ: mobility of electrons V_{gs}: gate-source voltage

C_{ox}: Oxide capacitance V_{th}: Threshold "

$I_{off} \widehat{=}$ leakage current   Loop: more heat → less Vth → more leakage →

$I_{on} \widehat{=}$ driving current   Delay ~ $1/I_{on}$

Delay also depends on input slew & Load capacitance

Long-Term effects of Temperature

$$\text{reactionRate} = A \cdot e^{-\frac{E_A}{k_B \cdot T}}$$

A = some factor

E_A = activation Energy in Joule

k_B = Boltzmann constant

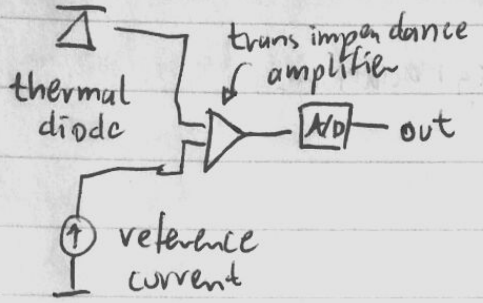
T = Temperature (K)

⇒ (high T ⇒ increased reaction)

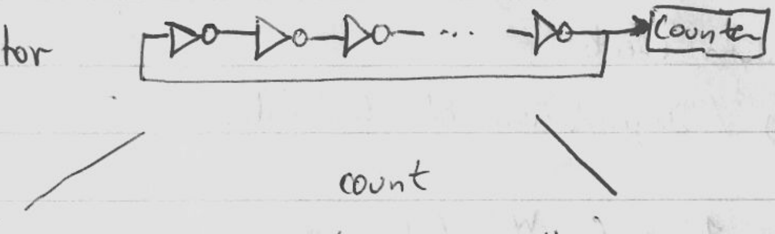
Temperature Sensors:

Thermal diode: forward voltage shifts => current through diode shifts w/ T

measure current through diode, transimpendance amp a comparator -> A/D converter -> digital sensor output



Ring oscillator



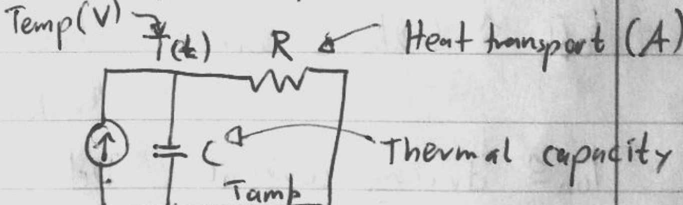
reading counter periodically and compare with calibrated LUT => temp.

Thermal simulation

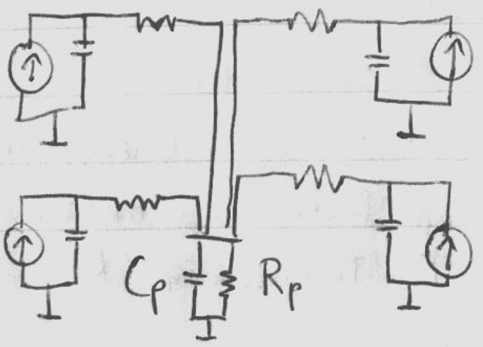
Resistor - Capacity equivalent to single component w/ heat transport.

Voltage ≙ Temperature

Current ≙ Heat Transport

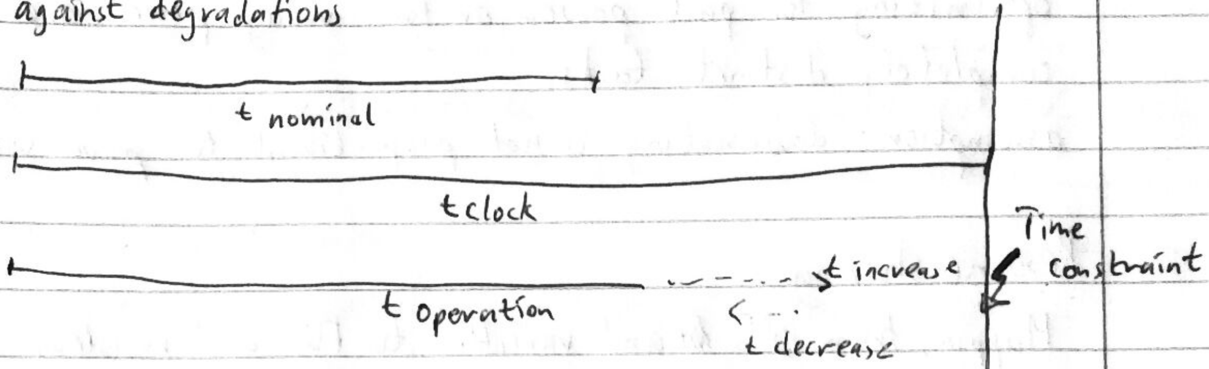


can be expanded to multiple components



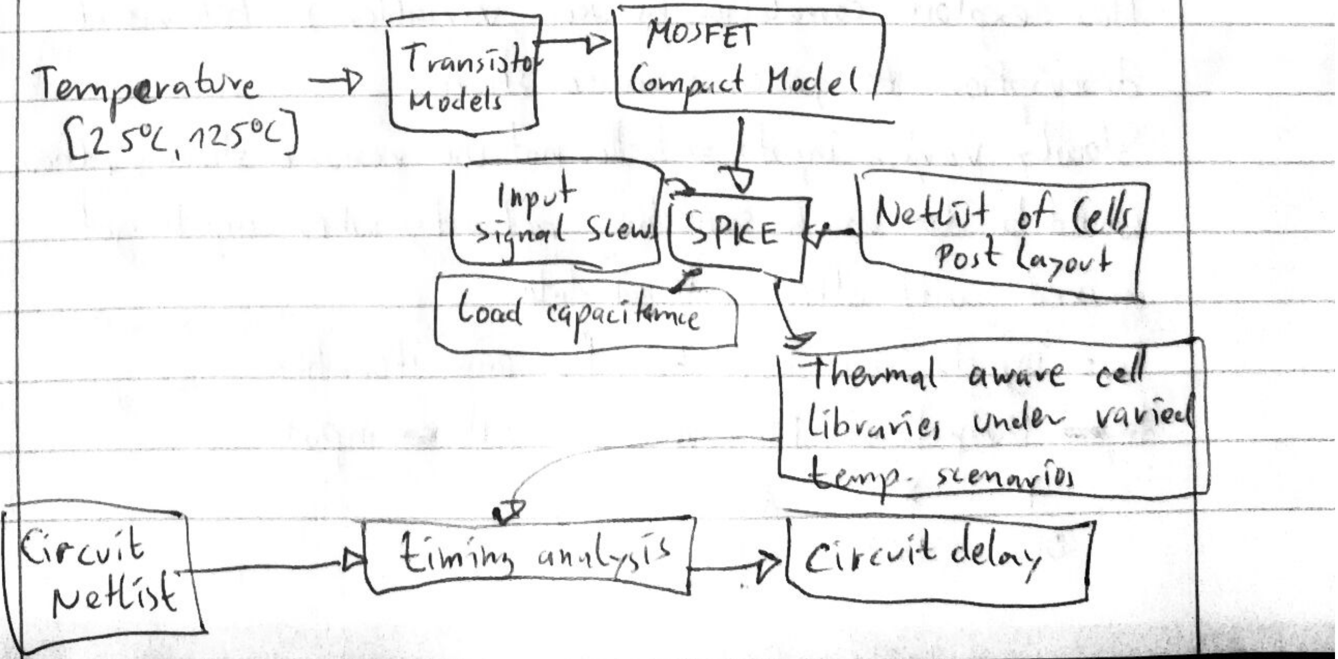
Temperature introduces degradations
lower I_{on} → Higher t_{delay} → Slower f_{clk}
higher I_{off} → worse leakage loop

Guardband is over-designing the circuit on purpose to protect against degradations



Ring Oscillator not suitable for guardband estimation because it is too local on one place of the chip, can't be used for ~~[illegible]~~ a holistic information of temp. of chip.

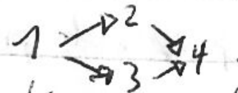
Characterize factured standard cells for different T and improve library with this information to enable ~~[illegible]~~ better design @ EDA
Set guardband based on $t_{delay}(125°C) - t_{delay}(25°C)$
set clock to $f_{clk} = 1/t_{delay}(reference) + t_{delay}(125°C) - t_{delay}(25°C) + ..$



6. More Low Power Concepts

Reduce peak power

by sequencing parallel instructions and use faster hardware. 1 → 2 → 3 → 4 has better avg. power than



. Energy might be the same.

optimizing for peak power or for average power are completely distinct tasks.

assumption: downscaling is not proportional to power saving

Resource sharing

Mapping from OPs and variables to FUs and registers

Definition of interconnects & MUXs builds RTL

Function ↪ Structure

Directly impacts power consumption by determining switching activity at various signals, buses, wires, macro blocks, ...

Observation:

resource sharing of variable's values → time muxed registers

if ~~OP~~ FUs are shared, their input switching activity is determined by the hamming distance of the variables that are passed to it. This might even reach over iterations.

Idea: exploit correlation between variables in behavioral description to guide resource sharing

Slowly varyin inputs ⇒ better not use resource sharing, there would be too much switching activity when inputs get muxed with other internal data.

Two inputs are corelated for one iteration

~~Input~~ Output of FU coveelates with input

+	*	AND	OR
0,5	0,617	0,75	0,75

Exploiting Signal Regularity

i.e. repeated occurence of computational pattern in algo

idea: exploit regularity to reduce interconnect power

Detect instances of repetitive patterns in the computation and resource shared by

reusing same interconnect structure for as many instances of computation as possible

Given DFG, regularities can be exploited and muxes/ bus wires / capacitance / driver logic can be saved in

Idea: define E-instances

E-instance is classified by type of in- and output

coverage: #E-instance / #edges

take this into account when building RTL for resource sharing

may come at cost of additional hardware (needs space)

Create CDG (connection distribution graph) to indicate what instruction needs to preceed another

Create FDG (final distribution graph) for mapping operations to RUs in a constructive way

Combine them both in order to minimize HW-need & maximize regularity exploiting

Switching activity of arithmetical units can be calculated:

$sw\_act(X \cdot Y) = 0$, if $X == 0 \lor Y == 0$

$sw\_act(X \cdot Y) = X$, if $Y == 1$

$sw\_act(X \cdot Y) = Y$, if $X == 1$

$sw\_act(X + Y) = \max(sw\_act(X), sw\_act(Y))$

=> for addition: place low-switching activities together to have another wire / bus w/ low switching activity

Glitch Power reduction

power, caused by switching activity caused by glitches

glitches occur due to temporal difference of transitions of input signals. Can't be seen in truth table

Glitches propagate through circuit. Needed to be stopped as soon as possible (where they are generated)

insert prime implicant and OR it to the output

needs additional HW + needs to be inserted in cell libraries if desired to be automatically added

Clock gating

disable clk by (clk ∧ 0) to specific parts of the chip that is not needed at a given ~~point in~~ point in time

no unnessecary propagation of clk, no storing of values if registers are connected, no calculation whose result is discarded

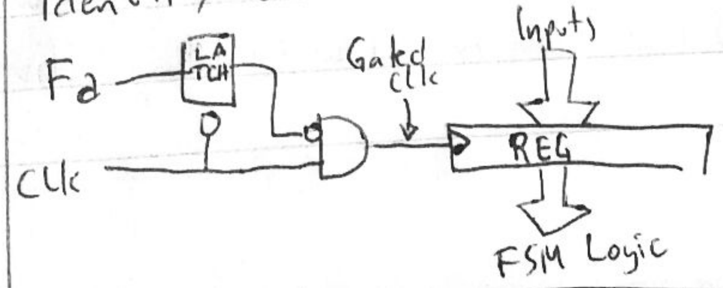
If a known off-condition can be determined, calculate it and AND it with clk. clk => '0'

If a known off-condition should freeze the clock @ '1', invert it and OR it to the clk (clk ∨ 1) clk = '1'

If an off-condition can be derived from previous clock signal, attach a latch. Example Decode-stage sees NOP → gate clock for next few cycles of EX, WB, MEM

However, additional circuity needed => possible new glitches, more HW space needed, more complex Synthesis and analysis and, if attached to clock, higher clock delay & higher clock skew

Identify off-conditions that are easy to calculate



Latch ensures that no glitch of F_2 can propagate

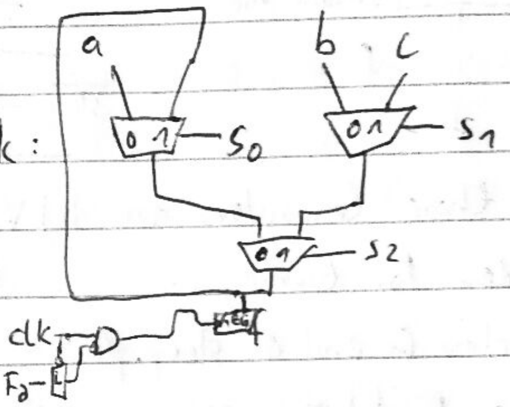
In case of FSM: if transition to next state is into the same state again, clock to registers can be suppressed

$F_a = \sum_{i=0,\dots,|S|-1} Self_{s_i} \cdot x_i$ a function that captures a set of inputs under which the FSM does a self-loop

Activation Function might be complex!

Clock gating for data paths when inputs are fed back:

$F_a = S_0 \wedge \overline{S_2}$



Fa needs to stabilize before clk does '1' -> '0'

When slower clk would be necessary, reduce Fa complexity

Clock gating @ clock-tree-level

=> + shuts off thousands of transistors at once

+ with only a single Fa

- Fa is rarely active

Clock-tree construction has impact on rate of Fa -> '1'

put registers w/ similar Fa into same clock-subtree

Multiple gated clocks

clock gating can also be control flow dependant: calculation that don't use an register every n % 2 cycles

clock gating only saves P_{switch} in clock tree @ this method

If DFG allows, split it into 2 clock intervals. If $C_1 + C_2 < C$, power savings are possible. Additional HW might be needed

Power Gating

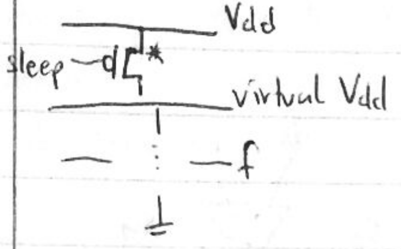
if circuit ~~[illegible]~~ idle: bring Vdd & Gnd to one level (potential = 0V)

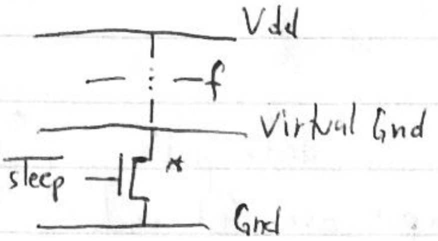
+ reduces greatly leakage power, more than P(saved) of clock gating

- slower than clock gating, needs time to drain voltage

- circuit state lost

- ⇒ needs retention registers





* those transistors are high V_{th} to minimize leakage currents

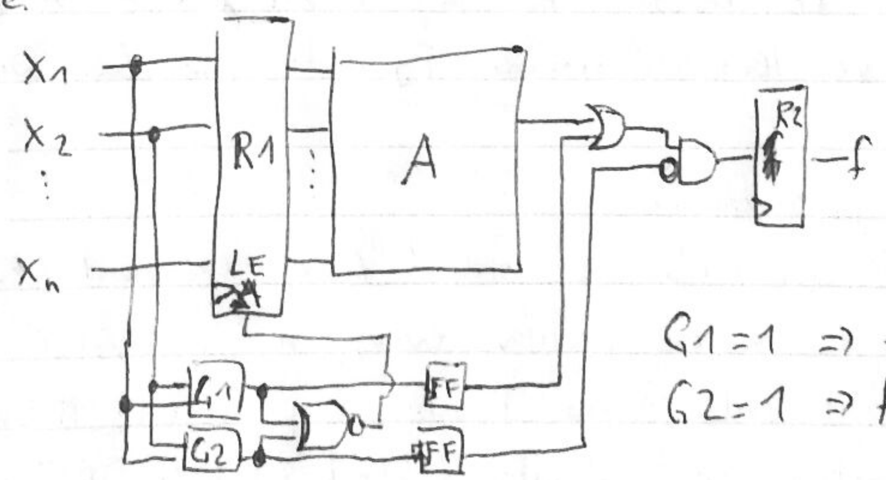
Header for turning on   Footer for switching off

delay @ end of sleep phase   delay @ begin of sleep phase

Also switching-power consumed, sleep interval needs to be long enough

Pre-Computation

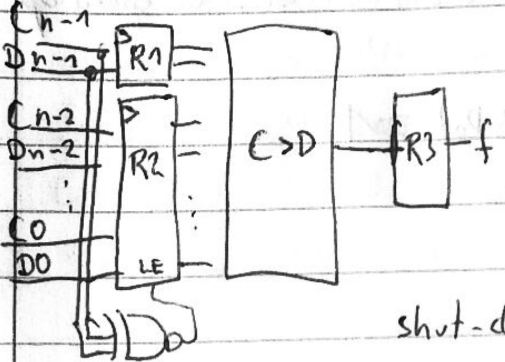
If the majority of input values' outputs can be determined, precompute them. For all outputs, complex logic is necessary. This can be shut ~~off~~ down in the first case.



G1 = 1 ⇒ f = 1

G2 = 1 ⇒ f = 0

For a circuit C>D, it is sufficient to look @ the MSBs of C & D, to have a 50% chance predicting '1' or '0' if uniform distribution of logic values to the inputs is assumed. Looking @ their second most significant bits, a 75% chance is achieved.



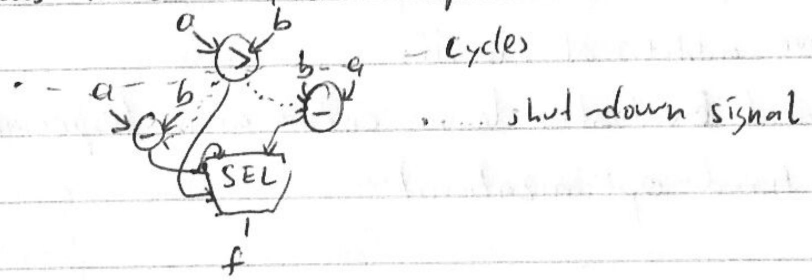
In case $C_{n-1} \Leftrightarrow D_{n-1}$, just keep the last values for $C_{n-2} \dots C_0$ and $D_{n-2} \dots D_0$ to reduce switching activity

shut-down criterium is logic-dependent

Managing power through scheduling

idea: calculate conditional statement and select FU to calculate result. Needs more HW, but is power efficient:

$f = |a-b|$



Operand isolation

if a block of logic can be shut down, but has no registers @ the inputs: insert transparent latches and disable their EN if $F_0 = 1$

Guarded evaluation

If input of chip is considered ODC to output of a combinational logic, it can be shut down using a latch enable $\overline{LE}$. Transparent latches need to be shut down early to ensure power saving @ switching logic

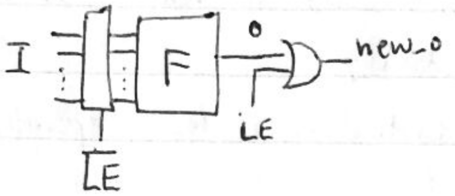
$t_1(LE)_{LE=1}$ latest time LE stabilizes at '1'

$t_e(I)_{LE=1}$ earliest time, @ which inputs may change when LE='1'

$$t_1(LE)_{LE=1} < t_e(I)_{LE=1}$$

Relaxation of LE criteria is once again a way to speed up LE stabilization

$$LE \Rightarrow (O + OCD_O) \Leftrightarrow \overline{LE} + (O + OCD_O) \equiv 1$$



Pre-Computation needs additional circuitry

Guarded Evaluation is derived from within the circuit

Pre-Computation may require re-synthesis to efficiently derive additional circuits

Guarded Evaluation leaves circuit as-is (especially impotant in hand-optimizations)

Given DFG & RTL

idle cycles of FU can be directly read from DFG

insert for each FU that has at least one idle cycle & at least one changing input in ~~Betwee~~ the idle cycles a transparent latch. LE is 0 in idle cycles.

Possible disadwantages:

additional circuity / delays / overhead / power consumption

delay constraints also apply to critical path, might not be acceptable in HPC.

Register sharing

Might assign some FUs w/ wrong values. Their result is discraided, but still calculated.
Idea: use more registers to prevent switching activity @FUs
A functional Unit w/o altering input does not perform a wrong operation, assign shared registers accordingly
Problems with loading inputs @ beginning of each iteration can be eliminated by preserving the old (from former iteration) register value

Managing power through the controller
re-design exiting logic in order to configure MUX-networks and FUs in the data path
low-cost, but might not eliminate all unnecessarary activity
best suited to control flow intensive designs
re-ordering states in FSM may lead to saving of power due to reduced switching activity
done by re-specifying control signals
Different incoming & outgoing transition in the idle state have different propabilities for execution
Values @ MUX themselves may change, only selecting the same one does not prevent switching activity to propagate.

7. Aging Aware Design

Aging weakens transistors ⟶ slower circuit
" leads to catastrophic failure of transistor.
Continuous degradation → "

→ timing violations

→ data corruption

Different phenomena:

Bias temperature instability (BTI)

→ Slower / weaker transistors

Hot carrier injection (HCI)

→ s.a.

Time dependent dielectric breakdown (TDDB)

→ catastrophic transistor failure

Electromigration (EM)

→ s.a.

Degradation in µs instead of years

Properties of aging constantly changing as well as technologies → not fully understood → conflicting evidence / multiple ~~theory~~ theories / new aspects uncovered every year

BTI is dominant phenomenon since 45 nm feature size, can be measured after µs

Defects in gate dielectric due to electric field

has a recovery effect

2 Theories:

1. Reaction - Diffusion

2. Trapping / Detrapping

2 Defect types: Interface traps:

Broken Si-H bonds

Oxide traps:

Oxide vacancies

Defect Generation during stress

Defect Healing during recovery

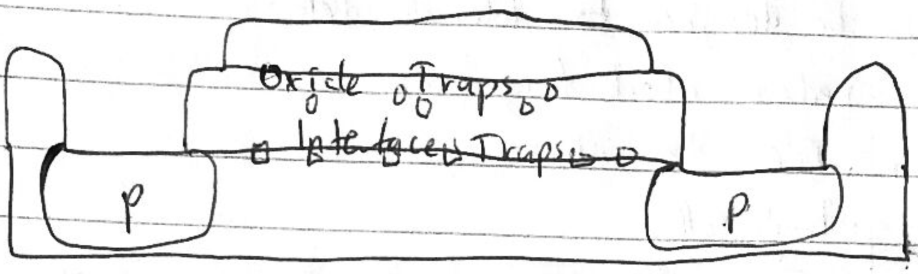
Defects are abstracted

agnostic to defect types

due to manufacturing, don't generate

& can't be healed

Reaction, Diffusion



Oxide traps are located in gate-dielectric

Oxide vacancies, positive charge, pre-existing due manufactoring & generation during operation

Interface traps in between dielectric & substrate

Broken Si-H bonds, positive charge, generated during operation

Trapping Detrapping

Electrical activation of pre-existing defects instead of defect generation

Activated by capturing a carrier from the channel

Deactivated " emitting " " back to " "

Aging occurs bottom-up, traverses through abstraction levels

Defect level → Transistor level → Cell level → M-Arch level → Chip level

Origin

The same aging degradation affects the delay of a cell differently - Parameters are clock slew & load capacitance

⇒ new critical paths emerge

Guardbands designed for former critical path
can also be designed to tolerate defects

- timing: reduce clock frequency "
- strength: wider transistors
- always a trade-off
- can be designed for a specific scenario or any

↓ ↓

mixed guardbands efficient for that inefficient, but safe

Guardbands can't protect against TDDB / EM
only against degradation

Redundancy needed at this point

How to reduce degradation? → decrease stimuli

→ Balance duty cycles (move job to other core)
→ reduce temperature (better cooling, add maxT and shutdown)
→ " Voltage (lower f clk → lower Vdd)

Low Power chips are manufactured today in newest, not completely optimized technology
→ high # of defects & degradation

· Reduce stimuli, accurate modelling → efficient guardbands

Aging has a direct impact on the power consumption
static power drops because worse conductivity ⇒ V_{th} increases
dynamic " " " " " ⇒ slower f clk