Campus | B5 7mm×30行 40页

Digital System Synthesis – ELEC 6233

lk1n16@stm.ac.uk

Campus®

B5 点线本 | 7mm×30行 | 40页

KOKUYO

ELEC 6216

No.

Date

Summary of topics

High-level synthesis for system-on-chip design
- Data structure, modelling, algorithms, mathematical background
- Synthesis and design space, optimisation
- Scheduling and binding (调度)
- Testing in Soc design

Synthesis for low power
- clock gating techniques
- Dynamic power, leakage power, power estimation
- Power-gating

SystemC - a high-level hardware modelling language

Performance and energy efficiency
- Many-core processor architectures
- Power dissipation and performance monitoring

No.

Date

High-level synthesis – Overview

Mathematical Background

Certain concepts from Boolean algebra { Binary decision diagrams (BDDs)
Boolean satisfiability and cover

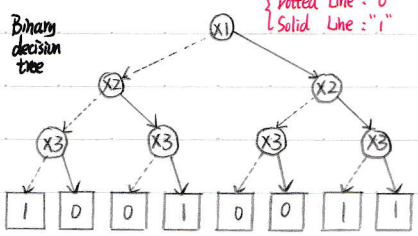
[BDDs]

A boolean function can be represented as a rooted, directed, acyclic graph, which consists of several decision nodes and terminal nodes. There are two types of terminal nodes called 0-terminal and 1-terminal. Each decision node N is labelled by Boolean variable V_N and has two child nodes called low child and high child.

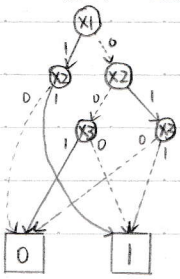
X1	X2	X3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Binary decision tree

{ Dotted Line : "0"
Solid Line : "1"



Binary decision diagram



[Boolean satisfiability] { a and not b , a=1, b=0 ⇒ TRUE | satisfiable
a and not a X | unsatisfiable

Boolean satisfiability problem is the problem of determining if there exists an interpretation that satisfies a given Boolean formula. In other words, it asks whether the variables of a given Boolean formula can be consistently replaced by the values TRUE or False in such a way that the formula evaluates to TRUE. If this is the case, the formula is called satisfiable

Boolean satisfiability (SAT)

Many high-level synthesis task can be solved by determining boolean satisfiability of logic functions

SAT:
Find a assignment to the ~~variables~~ variables $x_1, \cdots, x_n$ such that the boolean function $f(x_1, \cdots, x_n) = 1$, or prove that no such assignment exists.

Ordered Binary Decision Diagrams (OBDDs)

Different problems that can be modeled by OBDDs { Digital circuit design and verification
Sensitivity and probabilistic analysis on of digital circuits
Finite-state system analysis

Many Uses of OBDDs and SAT

SAT is a core computational engin for major applications

EDA { Testing
Logic synthesis
FPGA routing
Path delay analysis
...

AI { Knowledge based deduction
Automatic theorem proving

Also core engine for many other problems:
Integer Linear Programming
Theorem provers (HOL, Isabelle...)
{ HOL – Higher-Order Logic Proof Assistant
Isabelle – successor of HOL

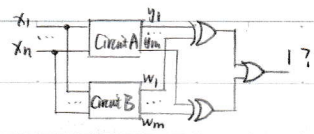
No.

Date

sa-0/sa-1 defects can be simplified by SAT solver

[Example] SAT can be used to solve the logic equivalence problem

{ Combinational circuit C_A, with n inputs and m outputs
{ Combinational circuit C_B, with n inputs and m outputs



Are the two circuits equivalent ?

{ A complicated problem formulation :
  Are the output equivalent for all input values ?
{ A simplier alternative :
  Are there (at least one) input values that distinguish outputs of the two circuits ?

abcd 0111 → 1
0

Digital integrated circuits can exhibit defects
Physical defects are modelled as logic faults
Most oft often used fault model : single stuck-at fault model

{ sa-0 : fixed at 0
{ sa-1 : fixed at 1

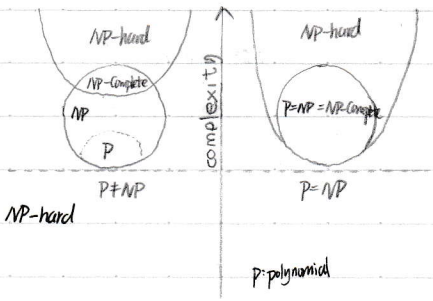
上图 Circuit A vs Circuit B
↳ Good Circuit vs Faulty circuit

for i=1 to n
  for j=i to n
    for k=j to n

$n^2+(n-1)^2+\cdots+2^2+1^2=\frac{n(n+1)(n+2)}{6}$ ✓循环次数，必定有 $\frac{n(n+1)(n+2)}{6} \le Cn^3$
时间复杂度为 $O(n^3)$

Is Boolean satisfiability (SAT) trackable ?

An algorithmic problem with n variables is said to be trackable if the algorithm necessary to solve it is of polynomial time
An algorithm is said to be of polynomial time (of P complexity) if its run time is upper bounded by a polynomial expression of the order of the input for the algorithm, i.e., $T(n) = O(n^k)$, k is a constant.
SAT is the first established NP problem (Non-deterministic polynomial complexity)
△ Cook's theorem : A problem is in NP if and only if it can be reduced to SAT.



P, NP, NP-complete complexity

If there is a polynomial algorithm for an NP problem, then P=NP

If there is an NP problem cannot be solved in polynomial time (P≠NP), it is NP-hard

An NP-complete problem is a problem that is both NP and NP-hard

Sizes of inputs increase, the complexity increase as exponential

No.

Date

$(\bar{a}b+b\bar{c})(a+c) = a\bar{a}b+\bar{a}b\,c+ab\bar{c}+b\bar{c}c$
$= \bar{a}bc+ab\bar{c}$

Conjunctive normal form (CNF) 合取范式

For SAT, the Boolean formula must be presented in CNF
A boolean formula is in CNF if it is a conjunction of clauses, where a clause is a disjunction of literals

$F = \bar{a}b\,c+ab\bar{c} = b(\bar{a}+\bar{c})(a+c)$

literal: $\bar{a}$, b, $\bar{c}$, a, c — clause: $(\bar{a}+\bar{c})$, $(a+c)$

CNF is an AND of ORs

$\{$ $(x+\bar{y})(\bar{x}+y)$ ✓
$(x\bar{y})+(\bar{x}y)$ X

[Algorithm to convert any Boolean formula to CNF (Tseitin transformation)

Recursive algorithm: convert(F):

① if F is a literal, return F
② if F has the form $F=pq$,
Convert(p), which must have the form $p=p_1p_2\dots p_n$
Convert(q), which must have the form $q=q_1q_2\dots q_m$
return $p_1p_2\dots p_nq_1q_2\dots q_m$
③ if F has the form $F=p+q$
Convert(p), which must have the form $p=p_1p_2\dots p_n$,
Convert(q), which must have the form $q=q_1q_2\dots q_m$
return $(p_1+q_1)(p_1+q_2)\dots(p_1+q_m)(p_2+q_1)\dots(p_2+q_m)\dots(p_n+q_m)$

④ if F has the form $\overline{a+b}$
return convert$(\bar{a}\bar{b})$
⑤ if F has the form $\overline{ab}$
return convert$(\bar{a}+\bar{b})$

$p_1p_2\dots p_n+q_1q_2\dots q_m = (p_1+q_1)(p_2+q_2)\dots(p_1+q_m)(p_2+q_1)\dots(p_2+q_m)\dots(p_n+q_m)$

$\{$ if $p_1=0 \Rightarrow q_1=\dots=q_m=1$
if $p_2=0 \Rightarrow q_1=\dots=q_m=1$ ⇐ The above expression is satisfiable if and only if

Literal and Clauses

e.g. p, $\neg p$, $p\vee q$, $p\vee\neg r\vee\neg q$, $\neg s\vee t\vee p$

A literal is either a propositional atom p or its negation $\neg p$.
A clause is a disjunction $l_1\vee l_2\vee\dots\vee l_n$ of one or more literals l_i

Conjunctive Normal Form (CNF)

A formula is in conjunctive normal form if it is a conjunction of one or more clauses
$\neg p$, $p\vee\neg q$, $(\neg p\vee q)\wedge(r\vee\neg t\vee\neg p)$

Testing validity of a formula in CNF

※Theorem
① A clause $l_1\vee l_2\vee\dots\vee l_n$ is valid iff there exists $i,j \Rightarrow l_i=\neg l_j$
② A CNF formula $C_1\wedge C_2\wedge\dots\wedge C_n$ is valid if each of its clauses C_i is valid

$\neg p\vee q\vee p\vee r$ is valid
$(\neg p\vee q\vee p)\wedge(r\vee\neg r)$ is valid
$(\neg p\vee q\vee p)\wedge(r\vee s)$ is NOT valid

Convert $F=\bar{a}b\,c+ab\bar{c}$

$F=\overline{\bar{a}b\bar{c}+\bar{a}bc+a\bar{b}c+ab\bar{c}+\bar{a}\bar{b}\bar{c}}$

$=(a+b+c)(a+\bar{b}+c)(a+\bar{b}+\bar{c})(\bar{a}+b+\bar{c})(\bar{a}+\bar{b}+c)(\bar{a}+\bar{b}+\bar{c})$

$(\bar{a}+\bar{a})(\bar{a}+\bar{b})(\bar{a}+\bar{c})(\bar{b}+a)(\bar{b}+\bar{b})(\bar{b}+\bar{c})(a+c)((+\bar{b})(\bar{c}+\bar{c})$

$(\bar{a}+\bar{b})(a+\bar{b}) = a\bar{a}+\bar{a}\bar{b}+a\bar{b}+\bar{b} = \bar{b}(a+\bar{a})+\bar{b}=\bar{b}$

$=\bar{b}(\bar{a}+\bar{b})(\bar{a}+\bar{c})(a+\bar{b})(b+\bar{c})(a+c)(\bar{b}+c)$
$=\bar{b}(\bar{a}+\bar{c})(a+c)$

KOKUYO

No.

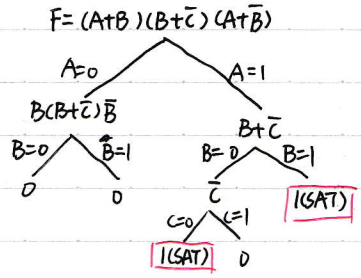
Date

Davis-Putnam SAT algorithm

Given a logic network, is there a sequence of inputs such that the network returns true as output?
Let the logical network be represented by the equation:
Davis-Putnam: Recursive Algorithm that creates a binary search tree by making assignments to the remaining variables at each stage of the algorithm $F = (A+B)(B+\bar{C})(A+\bar{B})$
Performs better than on average than an exhaustive search
- Effectively prunes failed branches from the search

Davis-Putnam (DP) Algorithm

```
produce split (F) {
    if F has an empty clause, then return
    if F has no clause, then exit with current partial assignment
    select next unassigned variable, xi in F
    split (F(xi=0))   // 分岐
    split (F(xi=1))
}
```



DA Algorithm complexity

Unfortunately in the worst case the Davis-Putnam Algorithm is still non-polynomial wrt number of variables (exponential, i.e. NP-complete)

Currently there is no known solution that provides a worst case performance better than exponential.

DLL Algorithm – an improvement over DP (Davis, Logemann and Loveland)

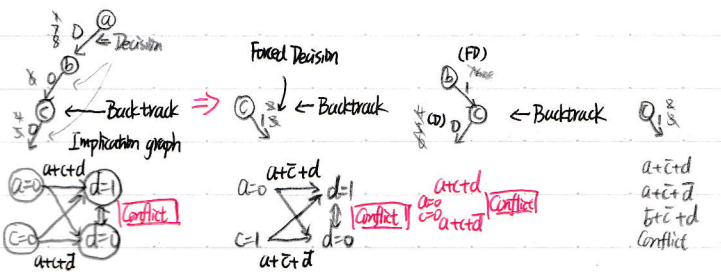
Also known as DPLL (Davis-Putnam-Logemann-Loveland) for historical reasons.
Basic framework for many modern SAT solvers

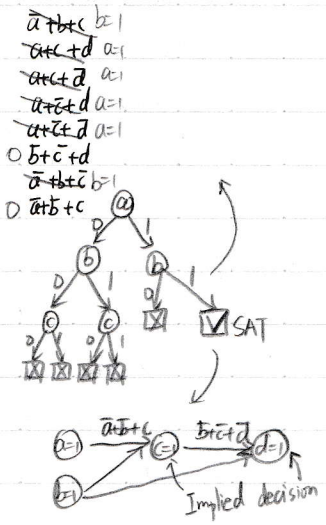
Basic DPLL Procedure - Depth First Search (DFS)

{FD: Forced decision
D: Decision

Example: set of clauses in a CNF logic formula function

① $\bar{a}+b+c$
② $a+c+d$
③ $a+c+\bar{d}$
④ $a+\bar{c}+d$
⑤ $a+\bar{c}+\bar{d}$
⑥ $\bar{b}+\bar{c}+\bar{d}$
⑦ $\bar{a}+b+\bar{c}$
⑧ $\bar{a}+\bar{b}+c$





Advantages off DLL over DP

Eliminates the exponential memory requirements
Exponential time is still a problem
Limited practical applicability - largest use seen in automatic theorem proving
Limited size of problems are allowed
- Problem size limited by total size of clauses (1300 clauses reported)

No.

Date

Binary Decision Diagrams - efficient Boolean space search

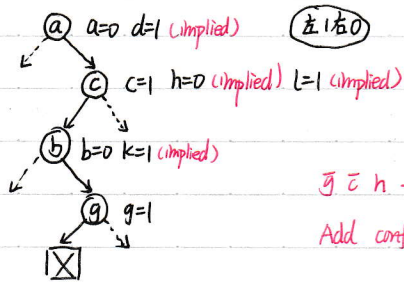
Store the Boolean function in a Directed Acyclic Graph (DAG) representation. Compacted form of the binary decison tree

Reduction rules to manipulate the graph.

[Conflict Driven Learning and Non-Chronological Backtracking]

$a+d$
$\bar{a}+\bar{e}+\bar{h}$
$a+h+l$
$b+k$
$\bar{g}+\bar{e}+i$
$\bar{g}+h+\bar{j}$
$g+h+j$
$g+j+\bar{k}$

Conflict



Backtrack to the decision level of variable c with g=0 (implied)

$\bar{g}\ \bar{c}\ h \rightarrow$ conflict

Add conflict clause : $\bar{g}+\bar{c}+h$

c=1 h=0 (implied) l=1(i) ⇒ g=0(i)

Conflict-Driven Learning - advantages

Learned clause is useful forever

Useful in generating future conflict clauses

Can restart, i.e. abandon the current search tree and re construct a new one

- Adds to robustness in the solver
- The clauses learned before the restart are still in the Boolean function after the restart and can help pruning the search space

Boolean Constraint Propagation

Boolean Constraint Propagation (BCP) == Unit Propagation (UP) == One literal Rule (OLR)

BCP is based on unit clauses, i.e. clauses that are composed of a single literal.

If a set of clauses contains the unit clause U, the other clauses are simplified by the iterative application of the following 2 ru

- Every clause (other than clause U itself) containing U is removed.
- In every clause that contains the negation of U: $\bar{U}$, the literal $\bar{U}$ is removed.

BCP example

$F = a(a+b)(\bar{a}+c)(\bar{c}+d)$

① Since $(a+b)$ contains a, this clause can be removed

② Since $(\bar{a}+c)$ contains the negation of a, $\bar{a}$ can be removed from the clause Hence : $F = ac(\bar{c}+d)$

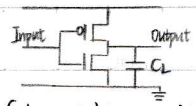
③ Since $(\bar{c}+d)$ contains the negation of c, $\bar{c}$ can be removed from the clause Hence : $F = acd$

No.

Date

Dynamic Power Reduction

Revision

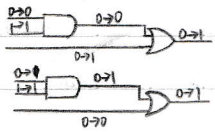


Power of Digital Circuit : $P = P_{dynamic} + P_{leakage}$

Gate switching activity (0-1 toggles)

Pdynamic caused by charging and discharging of capacitors $P_d = V_{dd}^2 * F_{clk} * C_L * E_{sw}$

Switching activity depends on input patterns



OR gates consume power if input 2 of OR gate changes from 0 to 1

AND gates consume power if input 1 of AND gate changes from 0 to 1

Dynamic power reduction happens at various design levels

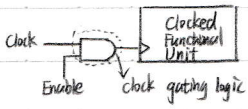
- Physical layout level = shorter interconnect → smaller C_L
- Transistor level = transistor with smaller W → smaller C_L
- Architectural level { Switching activity reduction : Clock gating; Vdd reduction : Multiple Vdd

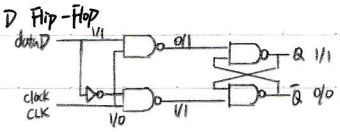
reason why we want 3-dimension chip → reduce the connection length

The longer the length, the bigger the capacitance

Clock gating : architectural level dynamic power reduction

When clocked modules (memory, register files, ...) are inactive (stored values not changing), use clock gating to disable the clock feeding module.





FF consumes power due to switching activity even when input is not changing (clock is changing)

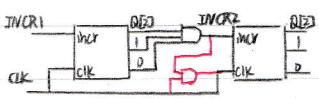
Clock gating reduces switching activity/dynamic power, up to 50% reduction

Example 1 : Clock gating two, 3-bit counters

Occur only on synchronous circuit

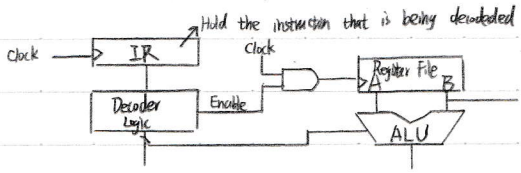
Each counter increments on every clock cycle when INCR1 and INCR2 high

INCR1 always high, COUNTER 1 increment every clock ; COUNTER 2 increment only once every 8 clocks, but consume power during each clock cycle.



- Clock gating logic (2AND) cause counters to be clocked by only when their increment signals high
- COUNTER 2 not clocked when INSR2 is low

Example 2 : Clocking gating processor register file

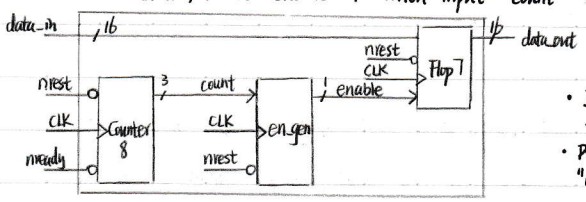


Only clock register file when instruction decoding require the use of register file

美国新思科技公司

RTL Clock Gating Using Synopsys Design Compiler

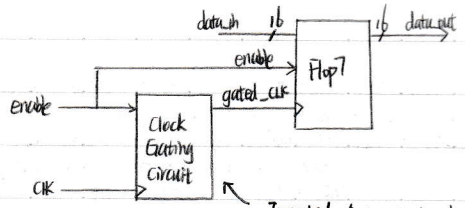
Example: Clock 16-bit "data_in" into flip-flops "Flop7" when Flop7 enable is "1", "enable" generated by "en_gen" block, where enable "1" when input "count" is "111" produced by "counter8" block



- Invoke "RTL clocking gating" by the command
  -set_clock_gating_style -sequential_cell latch -minimum_bitwidth 3
- RTL clocking gating works by identifying groups of flip-flops that share common "enable" indicating new values need to be clocked.

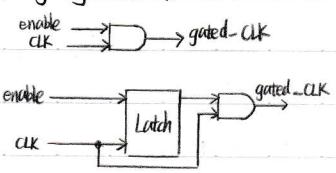
flop7.vhd

```
entity flop7 is port (clk: in std_logic;
                      nreset: in std_logic;
                      enable: in std_logic;
                      data_in: in std_logic_vector (15 downto 0);
                      data_out: out std_logic_vector (15 downto 0));
end entity flop7;

architecture flop7_arch of flop7 is
begin
    process (nreset, clk) is
    begin
        if (nreset = '0') then
            data_out <= (others => '0');
        elsif (rising_edge (clk)) then
            if (enable = '1') then
                data_out <= data_in;
            end if;
        end if;
    end process;
end flop7_arch;
```



Inserted Automatically by Power Compiler into the clock network

Clock gating circuits



Latch captures the state of the enable signal and hold it until next clock cycle.

No.

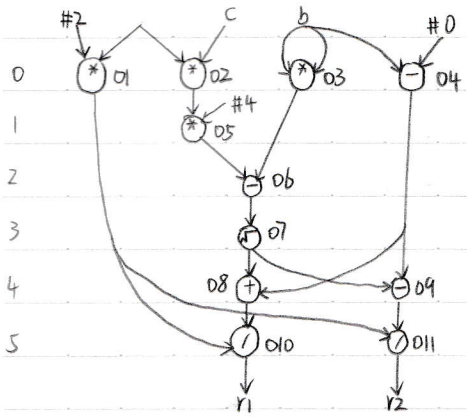
Date

$Delay = Kd * Vdd / (Vdd - Vt)^2$

Multiple Vdd

Architectural level dynamic power reduction.

- Modules on critical paths use highest Vdd (to meet required timing constraints)
- Modules on non-critical paths use lower Vdd (reducing power)



Assume Vdd = 2.2V, Vt = 0.5V, FUs delay = 10ns

Critial path operation: 02, 05, 06, 07, 08, 09, 010, 011

Operation: 01, 03, 04 are candidates for multi-cycled FUs

Critial path length = 10 + 10 + 10 + 10 + 10 + 10 = 60ns

Delay of multi-cycled 01 = 50ns, 03 = 20ns, 04 = 40ns (delay 冗余)

Delay of digital circuit, $D = Kd * Vdd / (Vdd - Vt)^2$

Where D functional unit delay, Vdd {supply voltage}, Vt {threshold voltage} of the functional unit, and Kd a constant

Consider multi-cycled FU 01 $\begin{cases} 50 = (Kd * Vdd) / (Vdd - 0.5)^2 \quad \dots (1) \\ 10 = (Kd * 2.2) / (2.2 - 0.5)^2 \quad \dots (2) \end{cases}$

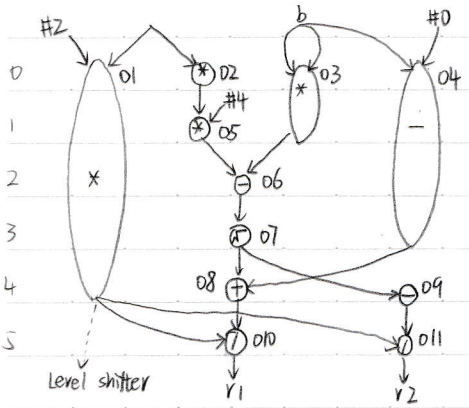
From Eq.(2), Kd = 13.1. Substitute Kd in Eq(1) and solve for Vdd gives Vdd = 1V

The multi-cycled * (01) will operate with Vdd = 1V.

Similarly, multi-cycled FU 03, Vdd = 1.48V

将 01 的 Vdd 调低，门延迟增至与 level 4 相同

Similarly, multi-cycled FU 03, Vdd = 1.48V



Multi-Vdd Design Requirements

- Use of multiple supply voltages
- Level shifts needed after reduced voltage module
- Multi-Vdd design flow (page 4, part3 notes)

For 03 $\begin{cases} 20 = Kd * Vdd / (Vdd - 0.5)^2 \\ 10 = Kd * 2.2 / (2.2 - 0.5)^2 \end{cases}$

$\frac{20}{10} = \frac{Vdd}{(Vdd-0.5)^2} \times \frac{(2.2-0.5)^2}{2.2} = 4.4 = \frac{2.89Vdd}{(Vdd-0.5)^2}$

$2.89Vdd = 4.4(Vdd^2 - Vdd + 0.25)$

$2.89Vdd = 4.4Vdd^2 - 4.4Vdd + 1.1$

$4.4Vdd^2 - 7.29Vdd + 1.1 = 0 \Rightarrow Vdd^2 \approx 1.48$

For 04 $\begin{cases} 40 = Kd * Vdd / (Vdd - 0.5)^2 \\ 10 = Kd * 2.2 / (2.2 - 0.5)^2 \end{cases}$

$4 = \frac{Vdd}{(Vdd-0.5)^2} \quad \frac{(2.2-0.5)^2}{2.2}$

$4(Vdd^2 - Vdd + 0.25) = 1.314Vdd$

$4Vdd^2 - 5.314Vdd + 1 = 0$

1	1.1
-0.314	-3.4×10^{-3}

$Vdd = 1.1V$

↑ In order to use Multiple VDD at RTL level UBF was invented.

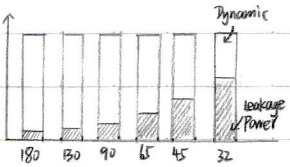
KOKUYO

No.

Date

Differ leakage current becomes dominant depends on different scaling

Leakage Power Reduction



- Leakage Power consumption occurs as long as the circuit is powered on
- Sub-threshold current between source and drain in MOS transistor occurs when gate voltage (V_{gs}) is below transistor threshold voltage V_t
- Sub-threshold current increases by 3x with each technology generation due to scaling of the transistor threshold voltage V_t
- At 45nm, total power (60% dynamic and 40% leakage)
- Vdd reduces with technology scaling (15% lower operating voltage, 30% smaller transistor), V_t must scale to deliver transistor performance

exam

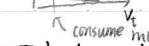
$V_t \neq V_t$ ← 阈值电压

$P_{leakage} \propto V_{dd} \cdot I_{leakage}$

$I_{leakage} \propto I_0 e^{\frac{V_{gs}-V_t}{nV_T}}$, $I_0 = \mu_0 C_{ox} \frac{W}{L} (n-1) \cdot V_T^2$ ← 热力电压

technology dependent

↓ Low V_t → More power



↑ consume more power but perform faster

⑤ Sleep Transistor (Power Gating)

Leakage Power Reduction ① Technology Level ② Stacking Effect ③ Body Bias ④ Adaptive Body Bias

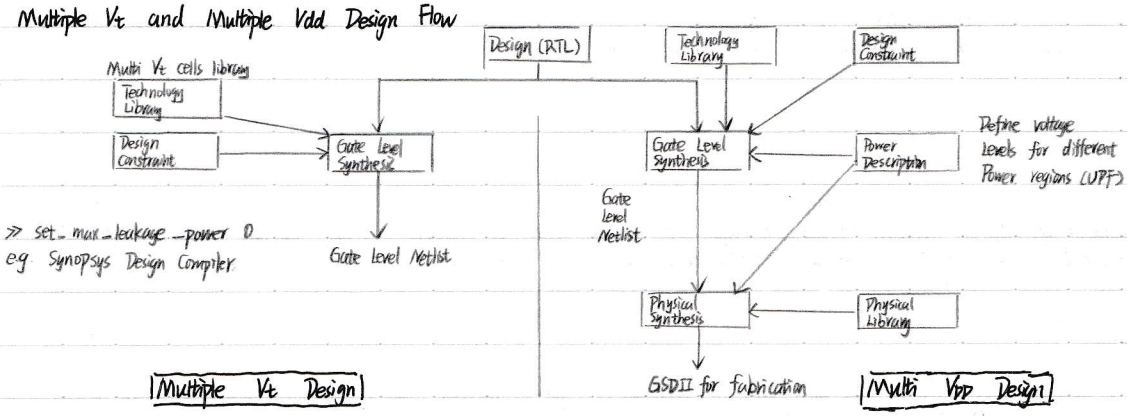
[Technology Level] Many libraries are required, hard to use multiple libraries

Multiple V_t technology where fabrication process provides both high and low threshold transistor V_t (low: fast and leaky and high: slow and less leaky) for N and P transistors. Default design uses high V_t and careful replacing high V_t with low V_t, one can get low V_t performance and yet significantly lower leakage power.

Synopsys Galaxy Design Platform provides an automated way of embedding "multi-V_t" to reduce leakage power.

△ Example: critical path delay determines performance of the design, all other paths have lower delay. Use low V_t transistors in modules on the critial path, and high V_t transistors in modules on noncritical paths.

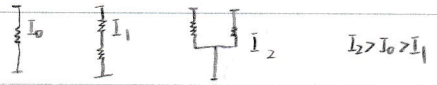
Multiple V_t and Multiple Vdd Design Flow



↑ Vocabulary in the slide

No.

Date

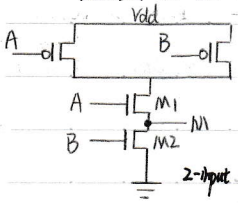


疊加效應

[Stacking Effect]

Exam

Sub-threshold leakage current flowing through stacks of series connected transistors reduces when more than one transistor in the stack is turned off. This effect is known as Stacking Effect.



Principle:

When M1 and M2 are off, voltage at node N is positive due to small drain current.
Due to $V_N > 0$, Body-to-Source Potential (V_{SB}) of M1 becomes negative resulting in increase in threshold voltage (larger body effect) of M1 and hence reducing the sub-threshold leakage

$$V_t = V_{to} + \gamma\left(\sqrt{(V_{SB} + 2\phi_F)} - \sqrt{2\phi_F}\right)$$ ← Not Stacking Effect

Σ for body bias

$V_t = V_{to}$ (when $V_{SB} = 0$)

{ V_{to} = Zero Substrate Bias
γ = Body Effect Parameter

△ Example: 2-input NAND gate, leakage current (IS) A=B=0, PMOS conducting, NMOS non-conducting, leakage current through 2 stacked transistors 0.01 pA

A=1, B=0 or A=0, B=1, one of the NMOS in weak region, leakage current through 1 transistor 0.46pA
A=1, B=1, both PMOS in weak region, leakage current = 2IS = 0.90pA. Fow how much values were obtained

△ Key observations

Exam

1. If number of stacked transistors > 3 leakage current very small
2. Leakage current for transistors in parallel is the sum of the individual currents. For 2-input NAND highest leakage current occurs for input 11, similarly for NOR, highest leakage current occurs for input 00.

△ Note:

It is possible to find an input vector that puts the logic in a minimal leakage state. This input vector can hard-wired into the logic and activated when the logic is not in use (standby mode)

Cost too much

[Body Bias]

Exam

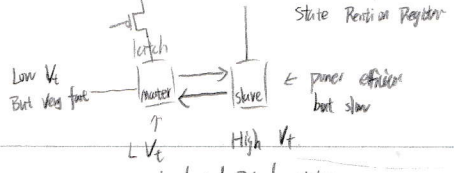
Applying reverse body bias to transistor reduces leakage and can be used to reduce standby leakage power. Alternatively apply forward body bias, which reduces V_t and improves circuit performance but increases leakage.

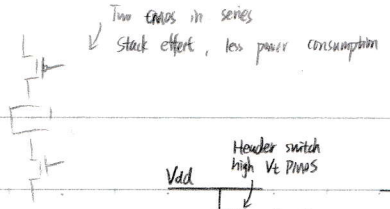
[Adaptive Body Bias]

Combination of reverse and forward body bias can be applied to reduce leakage power with reduced performance or improve performance with higher leakage power.

△ Case study

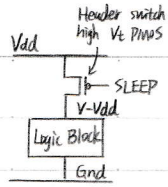
- Test chip is divided into 23 smaller designs, each design has circuitry to apply forward and reverse body. P transistors in each design can be biased individually, but all N transistors of the test chip have the same body bias.
- Slower dies are applied with forward bias to improve performance and faster leaky dies are applied with reverse body bias to reduce leakage
- 100% yield achieved, significant boost to high frequency binning.

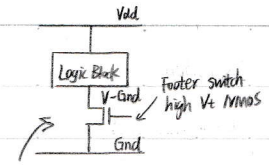




No.

Date





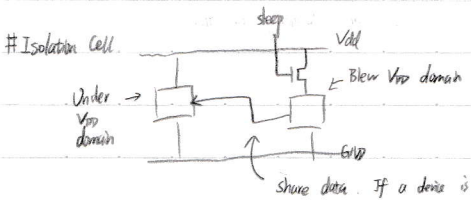
More transistors → Stacking Effect → High resistance
Low current

[Power Gating]

Exam Principle

- Sleep transistors
  - Insert high Vt MOSFET between power rail and logic blocks
  - Header transistor off lead to logic block floating to near zero
  - Footer transistor off lead to logic block floating to near 1

- Header switches preferred if multiple power rails since gated block logics float to zero irrespective of supply voltages



Follow a protocol in order to do the power gating
- Store states (avoid losing data)
- output isolation

Exam

UPF ✓ Pow. Gating design synthesis Flow. (Asking draw a flow) Code { FIFO, Power gating

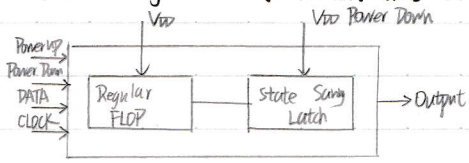
Gate leakage X ← Not a big problem, high grain material has been invented
notably about this

isolation cell → tracing the problem

State Retention Register

Retention registers are used to hold data/register states, before a power down. These states can be restored from these registers on power up. Hence retention registers are always on. Which means that even during power down mode, these will be on and will be consuming power.

A retention register consists of
① A latch or regular Flip-Flop - LVT (Low Threshold Voltage) cell for high performance. This will be given the normal power supply.
② State saving latch - High Threshold Voltage cells with different power supply. This power supply should be active during power done mode too.



In normal mode the state-saving latch will not be active.
Just before power down, the value in the regular flip flop, will be transferred to the state-saving register, and the power to the regular register will be shut down.
The power to the state-saving latch, should be maintained and thus it keeps the register state.

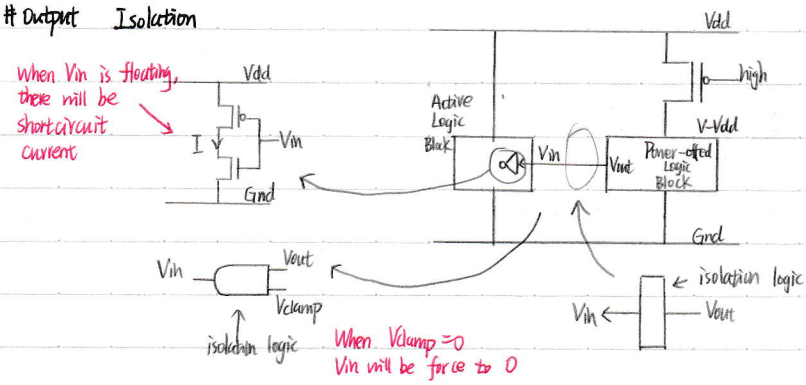
[PPT]

Sleep mode: state stored in balloon (latch) which is powered by Vdd
Active mode: balloon inactive; do not interfere with normal operation
B1 and B2 pins control the saving and restoring of state. In practice, retention registers may have only 1 control pin.

No.

Date

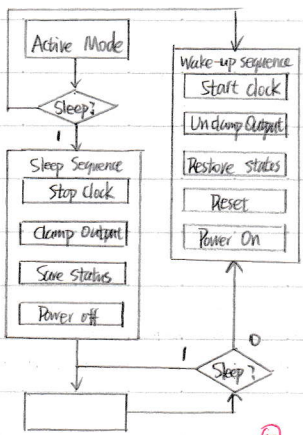
Output Isolation



Without output isolation, floating output causes short-circuit current in active logics

Isolation logic clamping the floating output prevents the short-circuit current in active logics

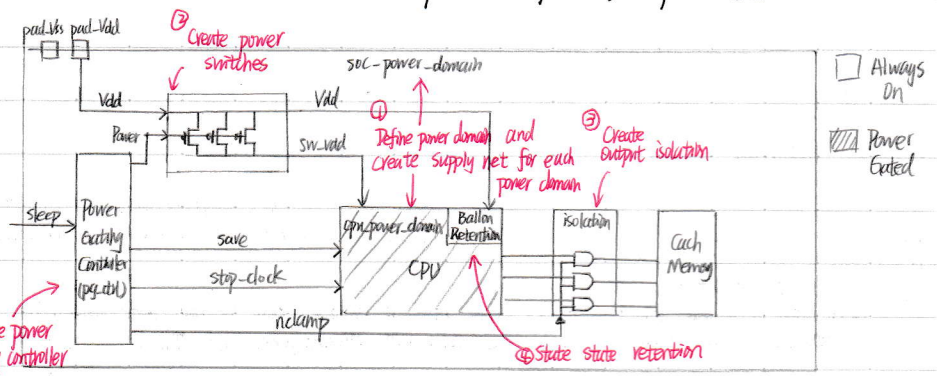
Power Gating



Require a controller to:

Clamp the power gated block output and save states before power off

Restore the states and unclamp the output after power on.



UPF (Unified Power Format)

① Define power domain and create supply net for each power domain

```
create_power_domain soc_power_domain
create_power_domain cpu_power_domain -elements {cpu}

set_domain_supply_net soc_power_domain \
                      -primary_power_net Vdd \
                      -primary_ground_net Vss
set_domain_supply_net cpu_power_domain \
                      -primary_power_net sw_vdd \
                      -primary_ground_net Vss
```

② Create power switches

```
create_power_switch pg_switch -domain cpu_power_domain \
                    -input_supply_port {Vdd Vdd} \
                    -output_supply_port {sw_vdd sw_vdd} \
                    -control_port {power pg_ctrl !power} \
                    -on_state {on_state Vdd {power}} \
                    -off_state {off_state {!power}}
```

No.

Date

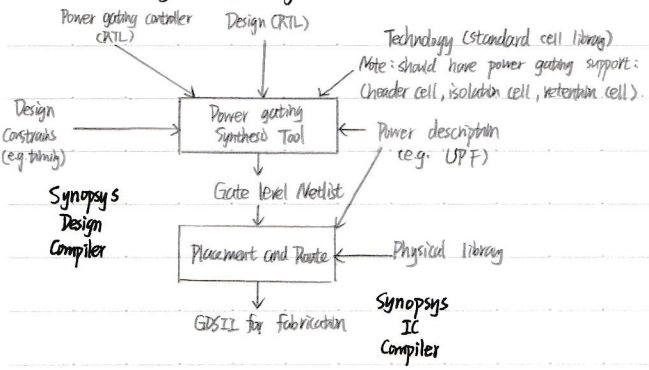
③ Create output isolation

```
set_isolation iso1 -domain cpu_power_domain \
                   -isolation_power_net Vdd \
                   -isolation_ground_net Vss \
                   -clamp_value 0 \
                   -applies_to outputs

set_isolation_control iso1 -domain cpu_power_domain \
                           -isolation_signal pg_ctrl/nclamp \
                           -isolation_sense low \
                           -location parent
```

④ Create state retention

```
set_retention ret1 -domain cpu_power_domain \
                   -retention_power_net Vdd \
                   -retention_ground_net Vss

set_retention_control ret1 -domain cpu_power_domain \
                           -save_signal {pg_ctrl / retain high} \
                           -restore_signal {pg_ctrl / retain low}
```

Power Gating Design Synthesis Flow



Power gating design synthesis requires:

① Power description (upf file)
② Power gating controller model (RTL)
③ Cell library with power gating support
④ Synthesis Tool with power gating capability

```
module fifo (power, retain, nclamp, clk, n_rst, wr_en,
             rd_en, d_in, d_out, full, empty);
parameter DATA_WIDTH=8;
parameter ADDR_WIDTH =3;
parameter DEPTH =8;
input power, retain, nclamp, clk, n_rst, wr_en, rd_en;
input [DATA_WIDTH-1:0] d_in;
output full, empty;
output [DATA_WIDTH-1:0] d_out;
reg [ADDR_WIDTH:0] distance;
reg [DATA_WIDTH-1:0] buffer [DEPTH-1:0];
reg [ADDR_WIDTH-1:0] wr_p, rd_p;
integer i;

always @ (posedge clk) begin
    if (~n_rst) wr_p <= {(ADDR_WIDTH){1'b0}};
    else if (wr_en && !full) wr_p <= wr_p+1;
end

always @ (posedge clk) begin
    if (~n_rst) rd_p <= {(ADDR_WIDTH){1'b0}};
    else if (rd_en && !empty) rd_p <= rd_p+1;
end
```

FIFO (A design to be power gated)

```
always @ (posedge clk) begin
    if (~nrest) distance <= {(ADDR_WIDTH+1){1'b0}};
    // if read only, decrease distance
    else if (rd_en && !empty && (!wr_en || full)) distance <= distance -1;
    // if write only, increase distance
    else if (wr_en && !full && (!rd_en||empty)) distance <= distance +1;
    // otherwise keep distance the same
end

always @ (posedge clk) begin
    if (~nreset) begin
        for (i=0; i<DEPTH; i=i+1) buffer[i] <= {(DATA_WIDTH){1'b0}};
    end
    else if (wr_en && !full) begin
        buffer[wr_p] <= d_in;
    end
end

assign full = (distance == DEPTH) ? 1'b1 : 1'b0;
assign empty = (distance == 0) ? 1'b1 : 1'b0;
assign d_out = buffer[rd_p];

endmodule
```

No.

Date

Gate Leakage

This is ~~not~~ another type of leakage current (beside subthreshold current) and happens because descrease in transistor oxide. ~~thinka~~ thickness which takes place with every technology node.

Gate ~~tunning~~ tunneling current increases by 25x for every 1A° decrease in oxide thickness, resulting in nearly 30x increas in gate tunneling current per technology generation

Gate leakage power needs to be considered, in particular in 45nm and beyond.

No.

Date

Energy Management

Energy vs Power

Power is energy consumption per unit time

Heat generation depends on power consumption. Fast chips run hot, and controlling power consumption is important for increasing reliability.

By reducing the speed at which the chip operates, we can reduce its power consumption (although not the total energy required for the operation)

dynamic power

$P_{dyn} = V_{dd}^2 * f * C_g * E_g(sw)$ ↓ Energy = P * t ↑

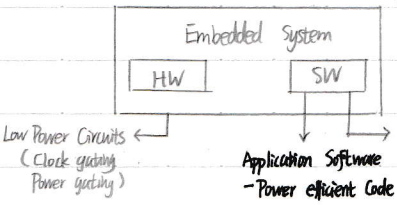
降低运行速度，功率损耗降低，但并不意味着总能量降低

因为速度下降，需要更多的工作时间.

Battery life, depends on energy consumption which can be lowered by power consumption.

For example reducing power supply, 1.2V to 0.9V causes the power consumption to drop by $1.2^2/0.9^2 = 1.78$

Energy Efficient Embedded Systems



Embedded OS

- Active power management OS provides the appropriate V/F values to run the application at the desired performance
- Idle power management OS puts part of the system into sleep and weak up based on requirement

Force on Software ↓ remove the leakage power → dis: shut down, require time to recover

Energy Management : system level

DPM　　DVFS

Dynamic power management (DPM), and dynamic voltage frequency scaling (DVFS)

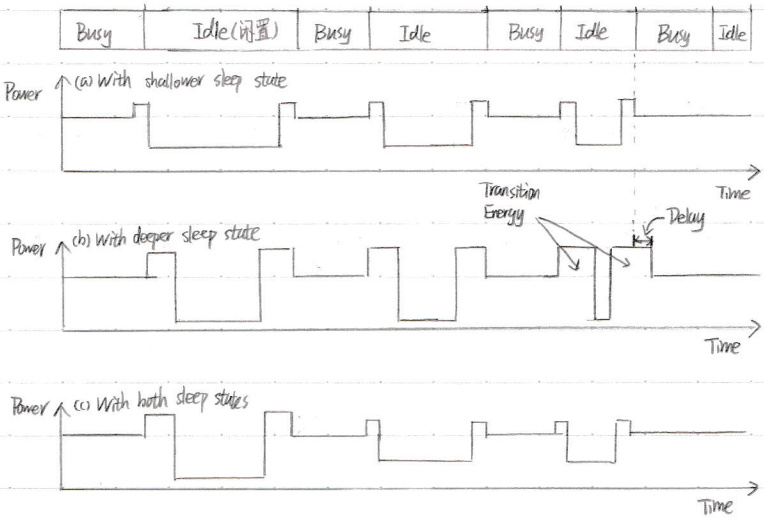
DPM: shut downs processing elements (PE) when inactive.

Different algorithms

- Greedy: Go to sleep as soon as processing is finished
- Timeout: Stay on expecting a new request. After time t of idleness go to sleep

Adaptive Timeout: Vary timeout t depending on activity.

Shallow sleep states are entered and exited quickly (free clock cycles), deep sleep states cost more in time and energy (tens of clock cycles)



← DPM

Take a lot of time and energy from sleep if it is in deep sleep state.

If the system wakes up every second, it should not go to a deep sleep state.

KOKUYO

No.

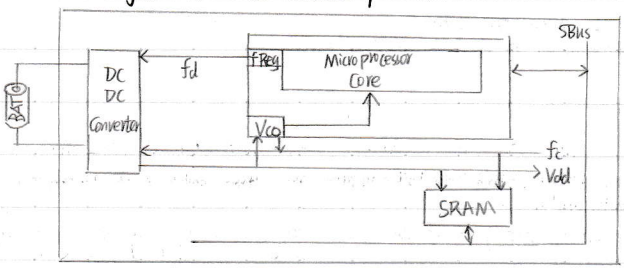
Date

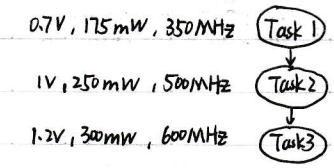
DVFS

#Energy Management : System level

DVFS : adapts PE performance to requirement by dynamically changing PE clock frequency and Vdd.
PE has non-uniform workload , executing MPEG requires 1 order of magnitude > processing performance than MP3
非均匀负载

Block-diagram of DVFS-enable processor





Turning off or reducing Vdd and hence power optimisation is possible due to the presence of idle time and slack time (碎片时间) in systems schedule.

In DVFS, power changes cubicly with frequency-voltage scaling, and overall performance is roughly linear with clock frequency.

Supply voltage	Frequency	Power saving	Perf-Loss (Performance-Loss)
100%	100%	0	0
95%	95%	14%	5%
90%	90%	27%	10%
85%	85%	39%	15%

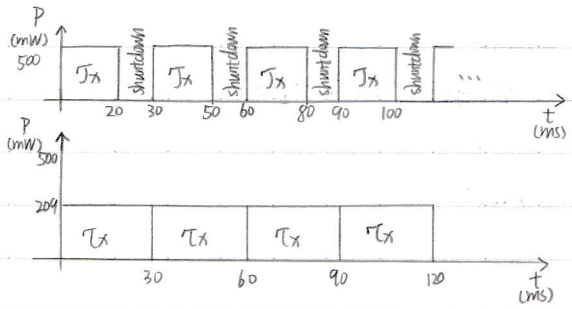
[Wiki] Dynamic voltage scaling is a power management technique in computer architecture, where the voltage used in a component is increased or decreased, depending upon circumstance.

[Techtarget] DVFS is the adjustment of power and speed settings on a computing device's various processors, controller chips and peripheral devices to optimize resources allotment (分配) for tasks and maximize power saving when those resources are not needed.
DVFS allows devices to perform needed tasks with the minimum amount of required power. The technology is used in allmost all modern computer hardware to maximize power savings, battery life and longevity (寿命) of devices. while still maintaining ready compute performance availability

Idle time : periods where PEs do not experience any workload
Slack time : amount of time that a PE is idle during the execution of a function. Result from tasks finishing execution before their deadline
slack = deadline - finish time
Why Idle and slack times exist in systems?
- not all PEs utilised constantly during run-time
- performance of PEs can not be perfectly matched to application needs : i.e. over-design commonly employed, with benfit of re-use of hardware over several product generations.

Energy management examples

PE performs a task T_x in 20ms and dissipates power of 500mW when running at 33MHz at $V_{dd}=2.5V$, and $V_t=0.8V$. To meet the application performance requirements, the task need to be repeated every 30ms.



← Figure shows a shutdown during idle time of 10ms.
Energy consumption (DPM) of executing task T_x : 500mW * 20ms = 10mJ

← Figure shows DVS exploiting of slack time, instead of shutting down the PE during the 10ms of idleness, Vdd can be reduced from 2.5V to 2.06V. To ensure correct operation frequency is reduced from 33MHz to 22MHz = (20*33/30). Tx in 20ms → Tx in 30ms

$500 \times \frac{22}{33} \times \frac{2.06^2}{2.5^2}$ → 226.3mW

With this voltage and frequency values, the PE dissipates power of
Therefore, energy consumption of executing task :
$T_v = 226.3mV * 30ms = 6.79mJ$.

Reduction of 32.1% in energy consumption

Energy Management Application

Approach 1 (design-time): explictly hand-code the shut down or changing clock frequency as part of the processor application software. This requires understanding of the application and processors power states.

Approach 2 (run-time): automatically by the OS once the processor has identified to be inactive or the required performance of the processor to execute a particular task has been established. This requires energy-aware OS/middle ware, capable of identifying processor inactive states, and determing the working frequency without violating deadline.
- characterise the application running tasks at runtime and accordingly make the V-f setting selection making use of runtime statistics (CPU utilisation, cache hit/miss ratio, ...) obtained from the hardware performance counters.

Approach 1 is mature, approach 2 still developing

$$V_{dd} = V_t + \frac{V_0}{2d^r} + \sqrt{(V_t + \frac{V_0}{2d^r})^2 - V_t^2} \quad (1)$$

$$V_0 = \frac{(V_{max}-V_t)^2}{V_{max}} = \frac{(2.5-0.8)^2}{2.5} = 1.156 \quad (2)$$

$$d^r = \frac{d(V_{dd})}{d(V_{max})} = \frac{30}{20} = 1.5$$

$$V_{dd} = 0.8 + \frac{1.156}{2*1.5} + \sqrt{(0.8+\frac{1.156}{2*1.5})^2 - 0.8^2} = 2.06 \quad (3)$$

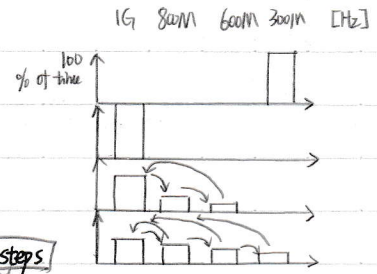
$$\frac{P_{Vdd}}{P_{Vmax}} = \frac{1}{d^r} \cdot \frac{V_{dd}^2}{V_{max}^2}$$

$$P_{Vdd} = \frac{1}{1.5} \cdot \frac{2.06^2}{2.5^2} * 500\,mW = 226.3mW$$

No.

Date

Linux Power Governors 功率调节器



Powersave - Set to the lowest frequency

Performance - Set to the highest frequency

Ondemand - { Set the freq. based on the current CPU usage
{ Max freq. when CPU usage is high ; decrease gradually

Conservative - Similar to ondemand ; increase / decrease frequency in steps

Userspace - User defined frequency setting

[shell command] sudo cpufreq-set --cup core_no --governor name
cpufreq is a Linux power management tool ; "sudo apt-get install cpufrequtils" to install
example: sudo cpufreq-set --cpu0 --governor ondemand

Interplay of DPM and DVFS

Processors support DPM and DVFS, important to understand the interplay between DPM and DVFS
Energy saved with DVFS come at the cost of increased execution time, implying shortened idle period for applying DPM, and greater leakage energy consumption.
Ideal DVFS policy must understand this interplay and hence perform V-f setting selection with the objective of reducing the overall system energy consumption.
This depends on the processor workloads characterisation must identify idle period
- DPM, the characterisation is in terms of the distributions of idle period durations. ← Save power (leakage power)
- DVFS, it is in terms of CPU / memory intensiveness of the executing tasks.
DPM and DVFS outperform each other under different workloads, no single PM policy fits perfectly all operating conditions, hence hybird power management is needed.

Energy Efficient Software

Individual instructions of software is the fundamental unit of software, similar to gates in hardware.
Software organization (coding) affects the energy cost of a program since this affects how processors / DSP consumes energy
Energy cost (static, base cost) = power * execution time = (I*V)*(n*T)
I= current drawn by processor during execution of program, V=Vdd, n= No. of cycles needed to execute program. T=clk period.

Instruction	Current (mA)	Cycles
NOP	276	1
ADD	314	1
JUMP	373	3
MOV	428	1

Energy cost (dynamic) accounts for the previous processor state and depends on sequence of events (e.g. cache misses)
Energy cost (program) = $\sum(B_i, N_i) + \sum(O_{ij}, N_{ij})$
B_i = base cost of instruction I, N_i = no. of occurance of instruction i
O_{ij} = dynamic cost of sequence, N_{ij} = no of occurance of sequence

No.

Date

Low-power system-level designs : General approaches

System Definition

An interconnection of possibly heterogeneous resources (electronic, electro-mechanical, optical, etc) which are often separately designed,
An interconnection of multiple cores within a chip or across several chips / boards

Power / Energy Optimization Opportunities

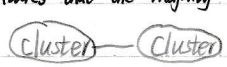
Hardware components { Optimise communications / Programmability

Software and algorithmic considerations { Operating system / Applications, compilation techniques and algorithms

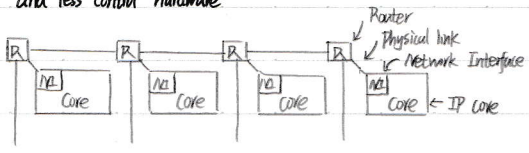
Optimising communication

Observation has already been made that energy in real-life systems is to a large extend dissipated in communication channels, sometimes even more than in the computational elements
Experiments have demonstrated that in designs, about 10%-40% of the total power may be dissipated in buses, multiplexers and drivers
This amount can be increase dramatically for systems with multiple chips.
Power consumption of the communication channels is highly dependent on algorithm and architecture-level design decisions
Properties of algorithms and architectures are important for reducing the energy consumption due to the communication channels:
- locality 地域性, regularity 正則性

[Locality] • relates to the degree to which a system or a algorithm has natural isolated clusters of operation or storage with few interconnections between them
• Partitioning the system or algorithm into spatially local clusters ensures that the majority of the data transfers take place within the clusters(局所的) and relatively few between clusters



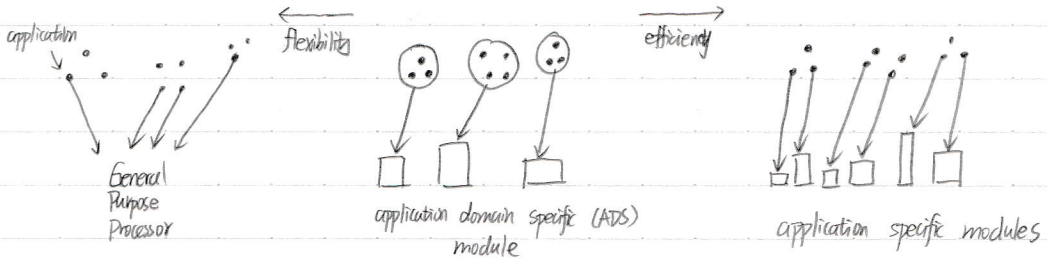
[Regularity] • In an algorithm, it refers to the repeated occurence of computational patterns.
• Common patterns enable the design of less complex architecture and therefore simplier interconnect structure (buses, multiplexers, buffers) and less control hardware



Programmability

It allows one system to be used for many applications
More important for mobile systems because they operate in a dynamically changing environment and must be able to adapt to the new Envir
- A mobile computer should be able to switch to a different network, without changing the application
- It should have the flexibility to handle a variety of multimedia services and standards
- It should have adaptability to accommodate the nomadic (遊牧民的) environment, required level of security, and available resources.

[Spectrum of applications and hardware implementations]



No.

Date

Software and algorithmic considerations

Operating system based power/energy management

Dynamic power management

Power management exploits periods of idleness caused by system under-utilisation
- Especially in mobile systems, the utilization is not constant

Designers naturally focus on worst-case conditions, peak performance requirements and peak utilization.
As a result, systems are also designed to operate under high utilisation, but they are actually fully exploited during a relatively small fraction of their lifetime

Dynamic power management { Binary power management
                           Dynamic power management of adaptive modules.

[Binary power management]

Deactivate functional units when no computation is ~~done~~ required
The main cost problem is ~~to~~ the cost of restarting a powered down module or component
Restarting induces an increase in ~~latency~~ latency (e.g. time to restore a saved CPU state, spin-up of a disk), and possibly also an increase in energy consumption (e.g. due to higher start-up current in disks)
The two main questions involved are then: 1) when to shut-down, and 2) when to wake-up
↳ a predictive approach, where the system initiates a wakeup in advance of the predicted end of an idle interval, works better.

[Dynamic power management of adaptive modules] this equivalent to DVFS

Computer subsystems may be designed for multiple levels of reduced energy consumption at the expense of some other system performance measure
Key to the approach is a high degree of adaptability of the modules.
It is needed to adapt to the changing operating conditions dynamically in the most (energy) efficient way
Using this model, the inherent trade-offs between e.g. performance and energy consumption can be evaluated and a proper adaption to the whole system can be made.

High-level Synthesis Steps - Scheduling

Typical high-level synthesis steps

High-level spec in C++/ SystemC or other

↓

Intermedia representation e.g CDFG (Control Data-Flow Graph)

↓

Optimise CDFG

↓

Perform Scheduling and determine resources (order in which operations execute)

↓

Perform binding (绑定,捆绑)

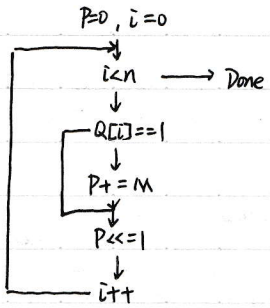
↓

Generate RTL model

Example - high-level synthesis from C code by hand

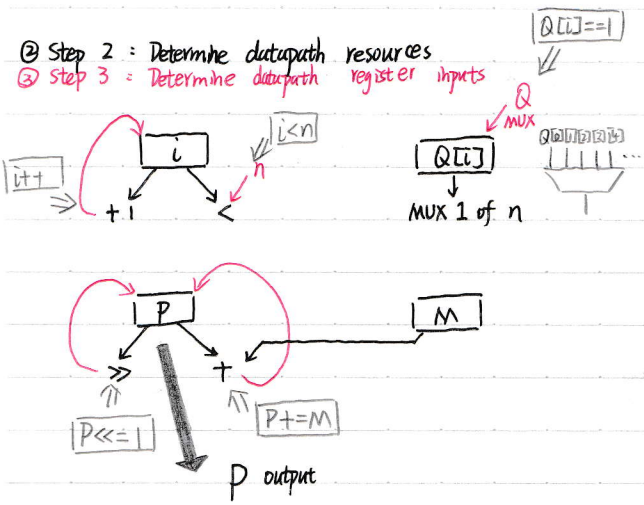
```
//Shift and add multiplication
//P = M* Q , n-bits
P=0;
for (i=0 ; i<n ; i++)
{
    if (Q[i] =1)
        P= P+ M;
    P= P <<1 ;
}
```

① Step 1: Build CFG for controller (an FSM)



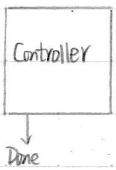
② Step 2 : Determine datapath resources

③ Step 3 : Determine datapath register inputs



④ Step 4 : Add control signals

⑤ Step5 : Combine control and datapath.



Automated high-level synthesis

Determine when to perform each operation
- Scheduling
- Consider alternatives and optimise

Allocate resources for each operation
- Resource allocation
- Consider alternatives and optimise, for performance

Map operations onto resources
- Binding

Optimisations

After creating CDFG, optimise the graph
Data flow optimisations
Control flow optimisations
Cost functions
- Area (IC/S' size)
- Performance (speed)
- Latency
- Power / energy

No.

Date

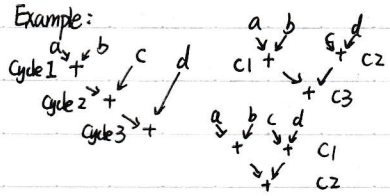
Scheduling

Data flow graph

Scheduling assigns a start time to each operation in DFG
- Start times must not violate dependencies in DFG
- Start times must meet performance constraints (Also, resource constraints)

Performed on the DFG at each CFG node (each state)
- Difficult to process multiple CFG nodes in parallel.

Example:



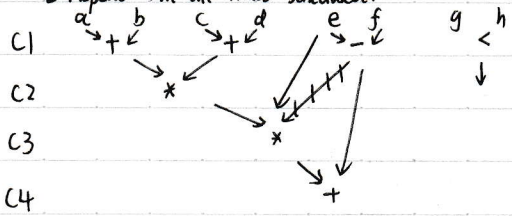
Typical problems in scheduling

Several types of scheduling issues
- Usually some trade-off of performance and resource constraints

Issues:
- Unconstrained scheduling
  - Not very useful, every schedule is valid
- Minimum latency scheduling - can be found through optimisation
- Latency constrained scheduling
- Resource constrained scheduling
- Minimum-latency, resource constrained scheduling
  - Find the schedule with the shortest latency, that uses less than a specified # of resources
  - NP - Complete problem (因多式项算法而使问题无法解决)
- Minimum-resource, latency constrained scheduling
  - Find the schedule that meets a given latency constraint, and uses the minimum # of resources
  - NP - Complete !

Minimum Latency Scheduling

ASAP (as soon as possible) algorithm
① Find a candidate node in DFG
  - Candidate is a node whose predecessors have been scheduled (or no)
② Schedule node one cycle later than max cycle of predecessor
③ Repeat until all nodes scheduled.

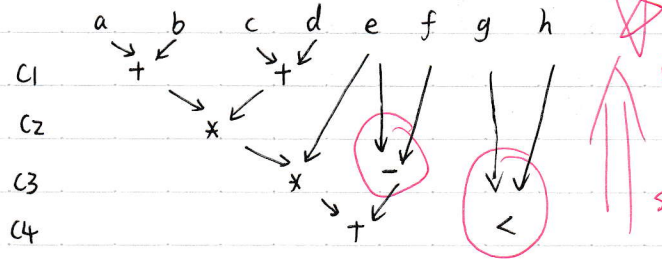


ALAP (as late as possible) algorithm
- Run ASAP, get minimum Latency L
- Find a candidate node in DFG
  - Candidate is node whose successors are scheduled (or there are none)
- Schedule node one cycle before start cycle of successor
  - Nodes with no successors scheduled to last cycle L
- Repeat until all nodes scheduled.

Latency-Constrained Scheduling

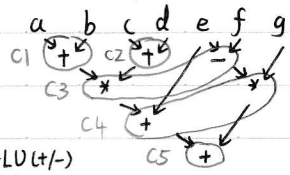
Instead of finding the minimum latency, find latency at most that of a given constraint L

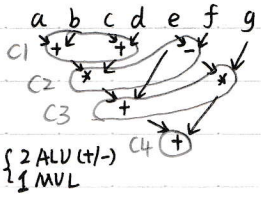
- Solutions:
  ① Use ASAP, verify that minimum latency is at most L
  ② Use ALAP starting with cycle L instead of minimum lancy (no need to run ASAP first, as L is given)



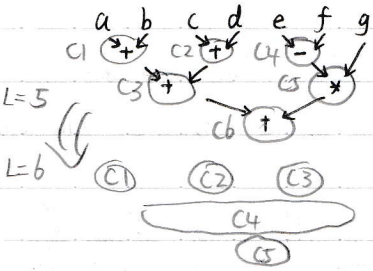
Scheduling with Resource Constraints

Scheduling must use less than specified number of resources.





Minimum-Latency, Resource-Constrained Scheduling



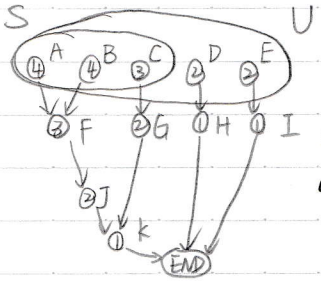
Different schedules may use same resources, but have different latencies

Hu's algorithm - example

Basic idea (number of resources)

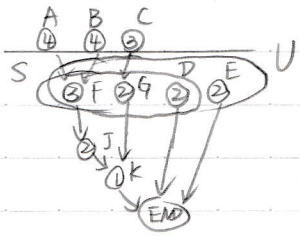
- Input: Data Flow Graph G, resource constraint r
- 1. Label each node in G by the longest path passing through it.
- 2. Initialise cycle count: L=1
- 3. Repeat
  - 3.1. Determine the subset U of unscheduled nodes in G whose predecessors have been scheduled or there are none
  - 3.2 Select subset S of U such that |s| <= r and labels in S are largest
  - 3.3 Schedule the S nodes at cycle L
  - 3.4 L++

Until all nodes in G are scheduled.

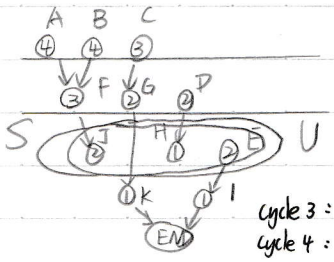


Assume r=3

Longest path, node scheduled at cycle 1 : A, B, C



cycle 2 : F, G, D



cycle 3 : J, H, E
cycle 4 : K, I

Summary

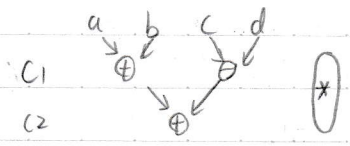
Scheduling assigns each operation in a DFG a start time
- Done for each DFG in the CDFG

Types of scheduling algorithms
- Minimum latency
  - ASAP, ALAP
- Latency constrained
  - ASAP, ALAP
- Minimum-latency, resource-constrained
  - Hu's algorithm
  - List scheduling (extension of Hu's algorithm)
- Minimum-resource, latency-constrained.
  - List scheduling

Next

Extension to Hu's Scheduling algorithm

Common Extensions:
- Multiple resource types
- Multi-cycle operation, e.g., multiplication may take 2 cycles



Minimum latency, resource-constrained scheduling

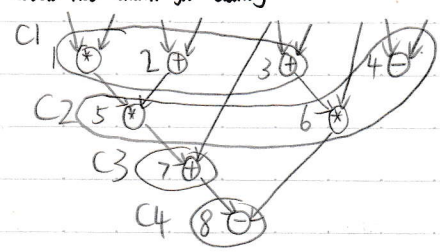
List-scheduling
- Extension for multiple resource types (resource)
- Basic idea - run Hu's algorithm for each type
  1. Input: graph, set of constraints R for each resource type t
  2. Label nodes based on max distance to output
  3. For each resource type t
     - 3.1. Determine candidate nodes, C (no predecessors or scheduled predecessors)
     - 3.2 Schedule up to R_t operations from C based on priority, to current cycle.
       - Where R_t is the constraint on resource type t.
  4. Increment cycle, repeat from 3, until all nodes scheduled.

Step @:

{ 2 ALU
{ 2 MUL

Data input

Labels not shown for clarity



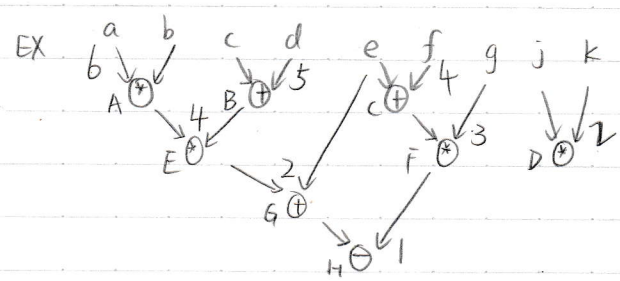
Candidates

Candidate, but not scheduled due to Low distance to output

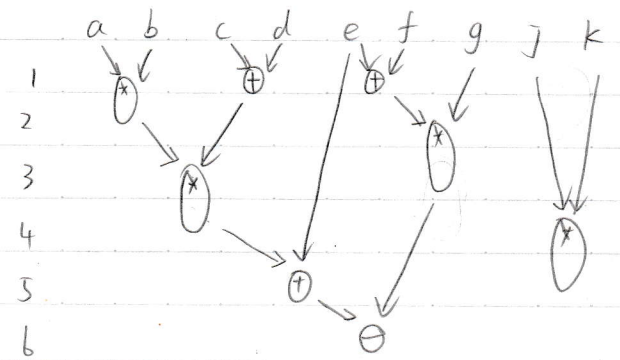
	Mult	ALU	Cycle
Step 3	1	2,3,4	1
Step 4 C+1			2
Step 3	5,6	4	
		7	3
		8	4

EX

{ 2 ALU
{ 2 MUL



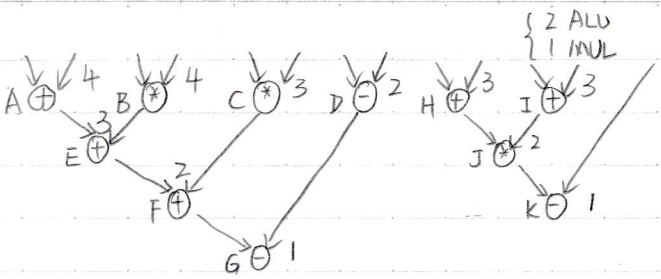
MUL	ALU	Cycle
A, D	B, C	1
F	-	2
E D	-	3
D	-	4
-	G	5
-	H	6



No.

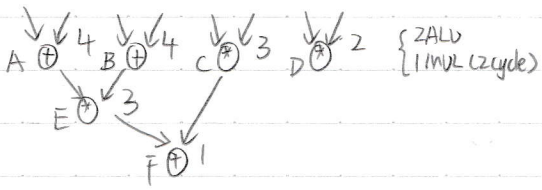
Date

Ex 1

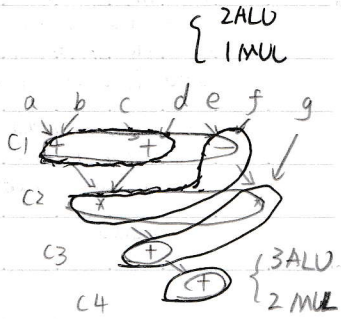


ALU	MUL	Cyc
A, D, H, I	B, C	1
E, D, I	C	2
F, D	J	3
G, K	-	4

Ex 2



ALU	MUL	Cyc
A, B	C, D	1
-	-	2
-	E	3
-	-	4
F	D	5
-	-	6

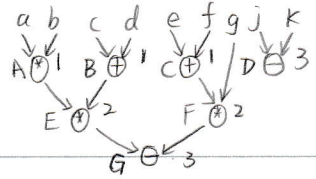


Scheduling for minimum resources with latency constraints

If no resource constraints are given, schedule should determine numbers of required resources.
- Max # of each resource type used in a single cycle.

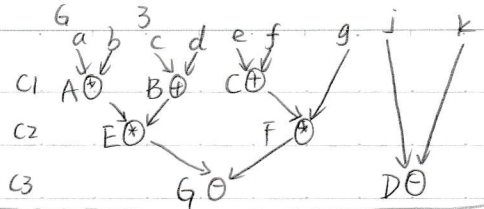
Minimum - Resource Latency - Constrained Scheduling
- Optimise: for all schedules that have latency less than the constraint, find the one that uses the fewest resources

List scheduling for minimum resources
Basic ideas:
- 1. Compute lastest start times for each node using ALAP with specific latency constraint
  - Latest start times must take into account ~~multiple~~ multicycle operations
- 2. For each cycle and for each resource type
  - 2.1 Determine candidate nodes
  - 2.2 Compute slack for each candiate Slack = current cycle - latest possible cycle
  - 2.3 Schedule the nodes with 0 slack
    Update required number of resources (start with 1 of each at the beginning)
  - 2.4 Schedule the nodes that require no extra resources
- 3. Repeat loop 2. until all nodes scheduled.



Last Possible Cycle

Node	LPC
A	1
B	1
C	1
D	3
E	2
F	2
G	3



Example : minimum resources with latency constraints

① Find ALAP schedule

Latency Constraint = 3 cycles

② For cycle and for each resource type
- 2.1 Determine candidate nodes C
- 2.2 Compute slack for each candidate
  - Slack = current cycle - last possible cycle

Candidate {A, B, C, D}

Initial Resource = 1 MUL, 1 ALU

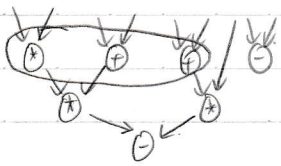
- 2.3 Schedule candidate nodes with 0 slack
  Update required number of resources
- 2.4 Schedule candidate nodes that require no extra resources.

Original

Node	LPC	Slack	Cycle
A	1	1-1 = 0	1
B	1	1-1 = 0	1
C	1	1-1 = 0	1
D	3	3-1 = 2	X
E	2		
F	2		
G	3		

Node D requires 1 more ALU
Not scheduled.

Candidates = {A, B, C, D}



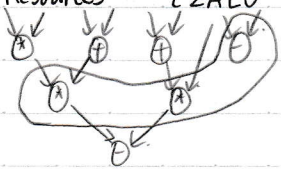
{ 2 ALU
{ 1 MUL

② For cycle and for each resource type ...

2.3 ...
2.4 ...

Candidate = {D, E, F}

Resources = { 2 MUL
{ 2 ALU



Node	LPC	Slack	Cycle
A	1		1
B	1		1
C	1		1
D	3	3-2 = 1	(2)
E	2	2-2 = 0	2
F	2	2-2 = 0	2
G	3		

require no extra resource thus being scheduled

② Candidate = {G}

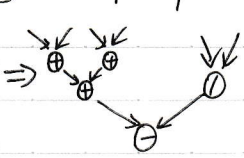
Resource = { 2 ALU
{ 2 MUL

G	3	3-3 = 0	3

Other extensions of Hu's algorithm

Chaining - Multiple operations in a single cycle

Multiple adds may be faster than 1 divide - perform adds in one cycle



Pipeline

Input : DFG, required data delivery rate

For fully pipelined circuit, one resource is needed per operation

Power constrained scheduling : based on scheduling for minimum resources with latency constraints

Each resource has an additional attribute : power consumption
Modification of Hu's algorithm for minimum power consumption
- 1. Start with only one type of each resource
- 2. Complete latest start times for each node using ALAP with specified latency constraint
  - Latest start times must take into account multicycle operations.
- 3. For each cycle and for each resource type
  - 3.1 Determine candidate nodes
  - 3.2 Compute slack for each candidate Slack = current cycle - last possible value
  - 3.3 If there are nodes with 0 slack, schedule all of them - ask for extra resources if necessary
  - 3.4 If there are no nodes with 0 slack, schedule only one node from those with the lowest slack arbitrary, but first try a node with the highest power consumption.
- 4. Repeat loop 3. until all nodes scheduled.

1. Power : ALU - 1mW MUL - 5mW
2. Find ALAP (Latency Constraint = 3 cycles)
3. For cycle and for each resource type (3.1 + 3.2)

Node	LPC	Slack	Cycle
A	1	0	1
B	1	0	1
C	1	0	1
D	3	2 1 0 3	
E	2	0	2
F	2	0	2
G	3	0 3	

Candidate = {1, 2, 3, 4} (A B C D)

Initial Resources = { 1 ALU, 1 MUL

3.3 → Resources { 2 ALU, 1 MUL Power = 7mW

D E F
{4, 5, 6}

→ Resource { 2 ALU, 2 MUL Power = 10mW

There are spare ALUs BUT do not schedule !!!

{4, 7} (D G)

→ Resource { 2 ALU, 2 MUL Power = 2mW

No.

Date

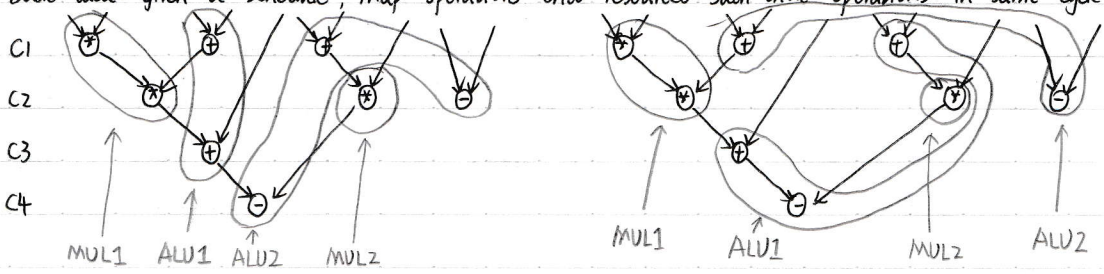
High-level Synthesis Steps - Binding and resource sharing

Binding 捆绑，装订

Scheduling determines
- When ops will execute
- How many resource are needed

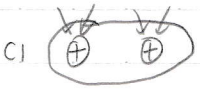
Binding determines which ops execute on which resources
- If multiple ops use the same resource
  - Resource sharing

Basic idea: given a schedule, map operations onto resources such that operations in same cycle do not use same resource



Many binding possibilities and many side effects
- Bund binding may increase resources, require huge steering logic, reduce clock rate

Can't bind same resource to more than op in a cycle
- 1 resource can't perform multiple ops simultaneously

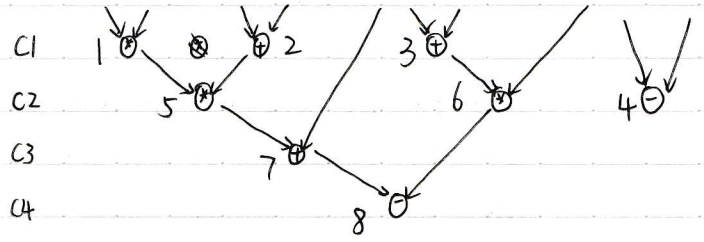


X 同一周期内不能共享一个资源

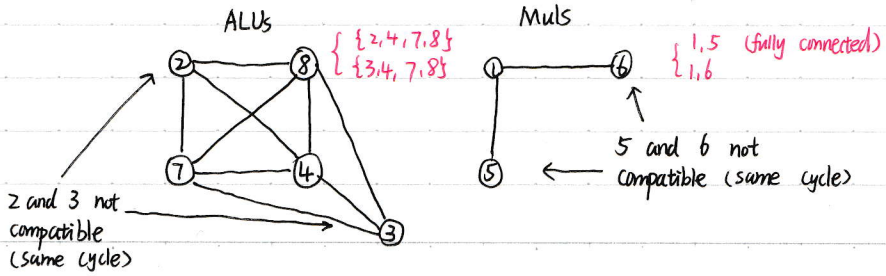
How to automate binding?
- More graph theory

Binding uses compatibility graphs

Compatibility graph, definition:
- Each node is an operation
- Edges represent compatible operations between a pair nodes
  - Compatible means that two ops can share a resource
    - Compatible ops use same type of resource (ALU, etc.) and are scheduled to different cycles.



Note: {Fully connected} subgraphs can share one resource (all involved nodes are compatible).
e.g. nodes {2, 4, 7, 8} in ALU subgraph can share a single ALU

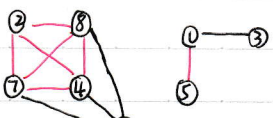


No.

Date

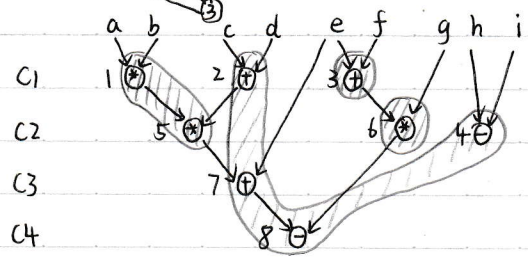
Compatibility Graph

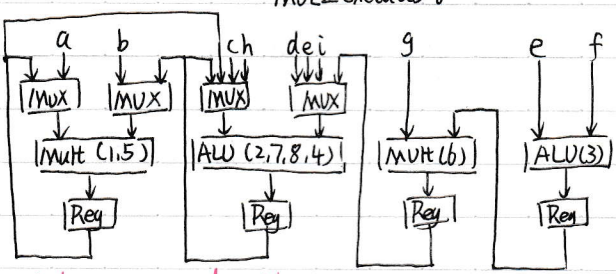
Binding goal: Find minimum number of fully connected subgraphs that cover entire graph.
- Well-known problem in graph theory: Clique partitioning (NP-complete) 已证明



Cliques = {{2,8,7,4}, {3}, {1,5}, {6}}

{ ALU1 executes 2,8,7,4 另一种略
ALU2 executes 3
MUL1 executes 1,5
MUL2 executes 6

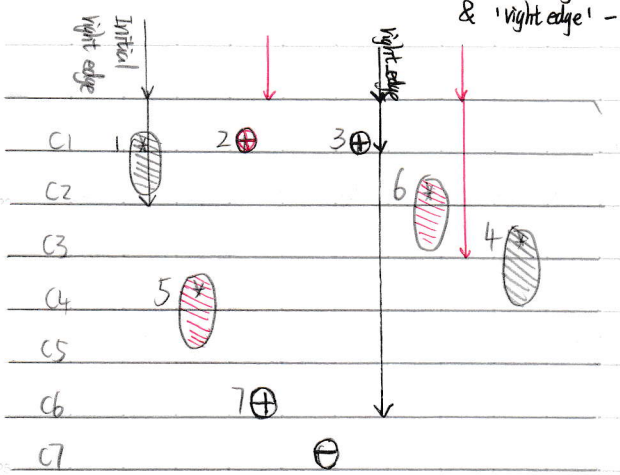


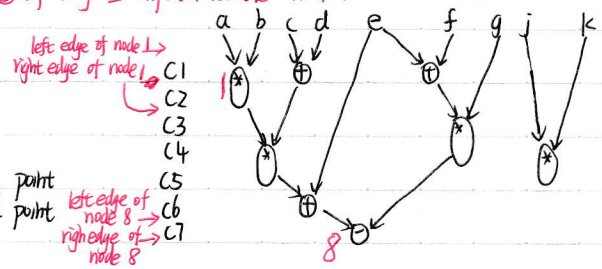


① Add resources and registers
② Add mux for each input
③ Add input to left mux for each left input in DFG
④ Do same for right ~~node~~ mux
⑤ If only 1 input, remove mux.

Left Edge Algorithm

Alternative to clique partitioning
- Name 'left edge' adopted from graph theory
- Each node in scheduled DFG has a 'left edge' - start point & 'right edge' - end point





1) Inialize right_edge to 0
2) Find a node N whose left edge is >= right_edge
3) Bind N to a particular resource
4) Update right_edge to the right edge of N
5) Repeat from 2) for nodes using the same resource type until right_edge processes all nodes
6) Repeat from 1) until all nodes bound.

Extensions

Algorithms presented here will find a valid binding
- But they do not consider amount of steering logic required
- Different bindings can require significantly different # of muxes

Possible solution
- Extend compatibility graph
  - Use weighted edges / nodes to introduce cost function representing steering logic
  - Perform clique partitioning, finding the best set of cliques that minimize weight

Binding Summary

Binding maps operations onto physical resources - Determine sharing among resources

Binding may greatly affect steering logic (控制逻辑)

Binding becomes trivial for fully pipelined circuits - 1 resource per operation

Translation from bound DFG to data path hardware is straightforward.

Summary of main HLS steps

HLS tool front-end (前端) (解析) parsing / compiling converts user code into intermediate representation - We considered CDFG

Scheduling assigns a start time for each operation in DFG
- CFG nodes start times are defined by control dependencies
- Resource allocation determined by schedule

Binding maps scheduled operations onto physical resources - Determines how resources are shared

Final result:

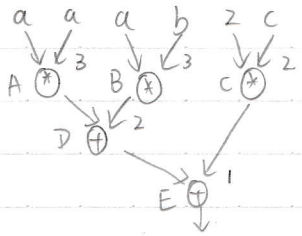
- Scheduling / Bound DFG can be translated into a data path
- CFG can be translated to a controller FSM
- This way High-level synthesis can create a custom circuit for any CDFG.

No.

Date

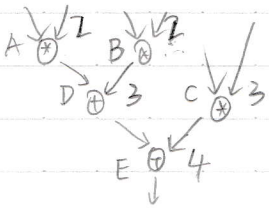
$a^2+a*b+2*c$ { 1 ALU, 2 MUL

ASAP



Cycle	ALU	MUL
1	-	A B C
2	D	C
3	E	

ALAP



Cycle 1: C = {A, B, C}

	LPC
A	1
B	1
C	2

arbitrary schedule A 5mW

Cycle 2: C = {B, C}

	LPC
B	0
C	1

⇒ Schedule B 5mW

Cycle 3: C = {C, D}

C	0
D	0

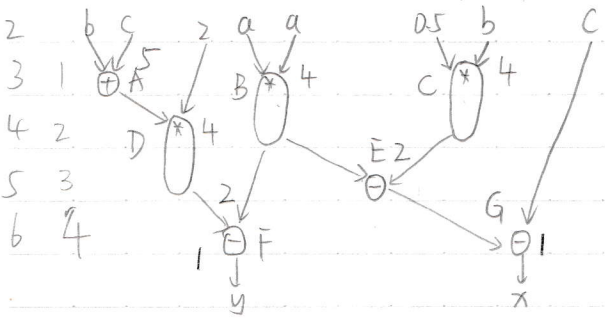
⇒ Schedule C, D 6mW

Cycle 4: C = {E}

E	0

⇒ Schedule E

1 { $X = a^2 - 0.5*b - c$, $y = a^2 - 2^*(b+c)$ { 1 ALU, 1 MUL



Cycle	MUL	ALU
1	B C	A
2	-	-
3	C D	-
4	-	-
5	D	E
6	-	G
7	-	F

① C = {A, B*, C*} R = {1 MUL, 1 ALU} ⇒ S = {A, B*}

② C = {D*, C*} R = {1 MUL, 1 ALU} ⇒ S = {}

③ C = {D*, C*} R = {1 MUL, 1 ALU} ⇒ S = {C*}

④ C = {D*} R* = {2 MUL, 1 ALU} ⇒ S = {D*}

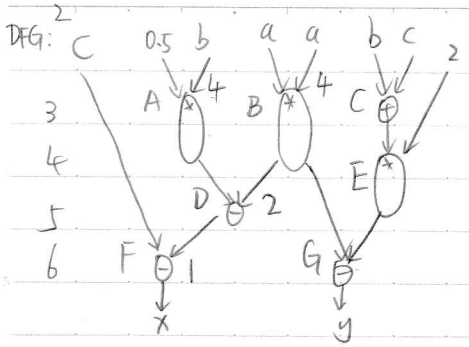
⑤ C = {E} R* = {2 MUL, 1 ALU} ⇒ S = {E}

⑥ C = {F, G} R* = {2 MUL, 2 ALU} ⇒ S = {F, G}

{ 2 MUL, 2 ALU

	LPS	Slack time
✓A	3	3-1 2
✓B	3	3-1 2
✓C	3	3-1 2, 3-2 = 1, 3-3 0
D	4	4-2 2, 4-3 1, 4-4 0
E	5	5-5 0
F	6	6-6 0
G	6	6-6 0

DFG:



Cycle	ALU	MUL
1	C	A, B
2	-	-
3	-	B, E
4	-	-
5	D	E
6	F	-
7	-	G

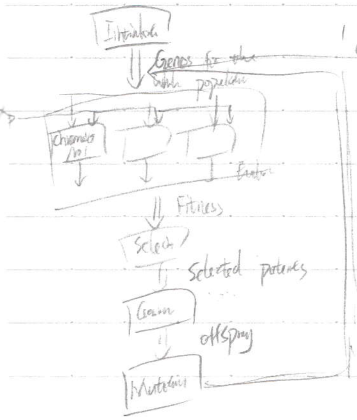
Step 1 : ALAP

						LPS	Slack	Cycle
r = {1ALU, 1MUL}	C = {A, B, C}	S = {A, C}	1		A	3		1
r = {1ALU, 1MUL}	C = {B, E}	S = {X}	2		B	3		
r = {1ALU, 1MUL}	C = {B, E}	S = {B}	3		C	3		1
r = {1ALU, 1MUL}	C = {E}	S = {E}	4	r* = {1ALU, 2MUL}	D	5		
r = {1ALU, 2MUL}	C = {D}	S = {D}	5		E	4		
r = {1ALU,		S = {F, G}	6		F	6		
					G	6		

find a assignment to variables $x_0, x_1, \dots, x_n$ such that the boolean function $f(x_0, x_1, \dots, x_n) = 1$ or prove that there are none

OBDDs provide an representation of boolean function in binary tree. ~~DB~~ In contrary to the truth table and algebric function, OBDDs provide an efficient means of recursive analysis that tackle the solution of more complex SAT problems

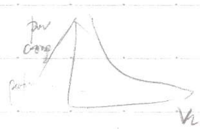
A temperature T the current design x'. Generate a random perturbation x' of. If the simulated energy, e.g. the cost function f(x') has decreased, move to this new design point. If the energy f(x') has increased, the new point is accepted ~~due~~ according to the probability of the boltzman distribution. $P = e^{\frac{f(x')-f(x)}{kT}}$ P,T → 0 K constant. When the system reaches the thermal equilibrium, reduce the temperature and repeat. ... guarantee to find a global optimum in the presence of many local optima provided it has long enough. Slow performance, highly dependent on the heuristic concerning the cooling schedule and probability of keeping a worse move.

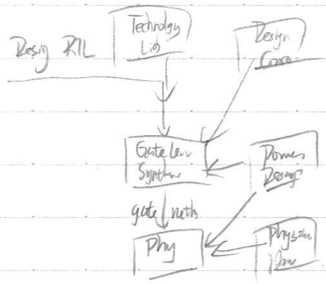


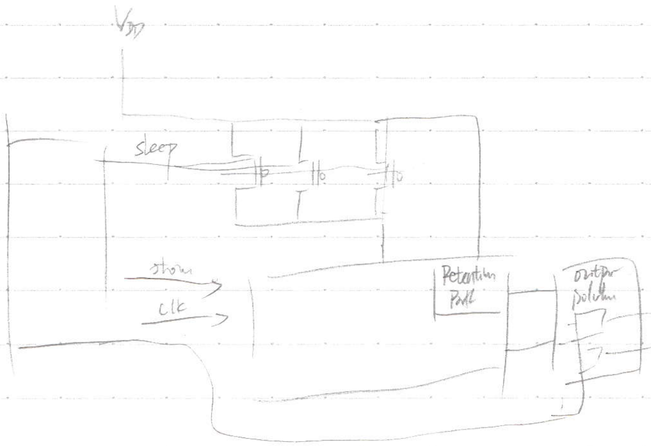
1. Can Easily used in multi-objective optimization
2. Is capable of finding a global minimum
3. Lend itself easy implementation on parallel platform
4. Easy to be synthesized both for static ...

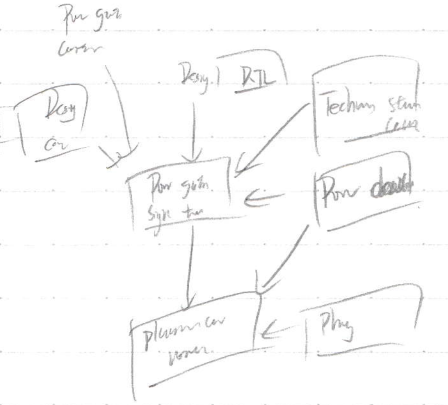
GA has slow p... compare to other stochastic methods

GA is ... the first ...









Stop clock
⇓
unclamp output
⇓
restore states
⇓
power off

SystemC module

Module is the basic building block in SystemC
SystemC has a macro for convenient module definition
- Macro name : SC_MODULE
- Macro implements C++ class inheritance

```
struct adder: sc_module {
    // ports, processes, internal data
};
```

⇕

```
SC_MODULE(adder) {
    // ports, processes, internal data
};
```

Module constructor

Module constructor is macro
- Macro name : SC_CTOR
- SC_CTOR must have module name as an argument

```
SC_MODULE(adder) {
    // ports, processes, internal data
    SC_CTOR(adder) {
        // body of constructor
        // process declaration, sensitivities
    }
};
```

No.

Date

SystemC

What is SystemC?

Superset of C++ - SystemC is a library of classes and macros designed to facilitate hardware modelling.
SystemC supports hardware and software co-design
SystemC has a rich set of data types to model hardware and software systems.
It allows multiple abstraction levels, from high level down to cycle-accurate RTL level.

SC_MODULE() ← is a macro that defines a new class for a SystemC module

Why SystemC?

Arbitrary levels of abstraction { From RTL level to system-level transaction modelling
Hardware/Software Co-design

Simulation performance is high { High-performance C++ compilers
Illusion of concurrency

High productivity in system development

SystemC design flow

Functional Specification
↓
SystemC Functional Model ← SystemC Validation Test Bench
↓
Synthesis
↓
Gate Level HDL
↓
Hardware

SystemC module hierarchy

Top-Level Module { C/C++ & SystemC data types
signals
Simulation control directives

Basic Module(s) ⟷ Signals ⟷ Testbench (Modules)
- Ports
- Data/signal

Testbench (Modules)
- Ports
- Data/signal

SystemC jargon 术语

[Module]
- Basic building block for structural partitioning
- Contains ports, processes, data
- Other Modules

[Process]
- Basic specification mechanism for functional description
- Three types of processes
  SC_method : sensitive to some ports/signals, no wait statements
  SC_thread : sensitive to some ports/signals, with wait statements
  SC_cthread : sensitive to only clock

[Interface]
- A set of operations for communication
- No specification about the actual implementation
- For example, a signal supports read and write operations

[Channel]
- An implementation of the interface operations
- Performs the actual data transfer
- Vary from basic signals to complex protocols

[Ports]
- Objects for inter-module communication
- Each port is a class and specifies communication interface.

space !!!!!

<SC_unit<8>> 我操!!

No.

Date

背景颜色 Tools → Edit Preferences → more windows

design.cpp

```
#include "systemc.h"

SC_MODULE(first_counter){
    sc_in_clk  clock;   // Clock input
    sc_in<bool> reset;  // Syn Reset
    sc_in<bool> enable; // Enable
    sc_out<sc_uint<4>> counter_out; // 4-bit vector output

    sc_unit<4> count;  // Local variable

    void incr_count(){
        if(reset.read()==1){
            count = 0;
            count_out.write(count);
        }else if(enable.read()==1){
            count = count+1;
            count_out.write(count);
            cout<<"@"<<sc_time_stamp()<<":: Incremented Counter"
                <<count_out.read()<<endl;
        }
    }

    SC_CTOR(first_count){
        cout<<"Executing new"<<endl;
        SC_METHOD(incr_count);
        sentitive<<reset;
        sentitive<<clock.pos(); // Positive edge trigged
    }
};
```

Module

Output & Input

Local Signal

Functionality

Constructor Block

Method

Sentitive List

SC_METHOD
SC_THREAD

sc_method will run more than once by design and methods cannot be suspended by a wait statement.

testbench.cpp

```
#include "systemc.h"
#include "design.cpp"

int sc_main(int argc, char* argv[]){
    sc_signal<bool> clock;
    sc_signal<bool> reset;
    sc_signal<bool> enable;
    sc_signal<sc_unit<4>> counter_out;

    int i=0;
    first_counter counter("COUNTER");
    counter.clock(clock);
    counter.reset(reset);
    counter.enable(enable);
    counter.counter_out(counter_out);

    sc_start(1,SC_NS);

    //Open VCD file
    sc_trace_file *wf = sc_create_vcd_trace_file("counter");
    //Dump the desired signals
    sc_trace(wf,clock,"clock");
    sc_trace(wf,reset,"reset");
    sc_trace(wf,enable,"enable");
    sc_trace(wf,counter_out,"count");

    //Initialize all variables
    reset=0;
    enable=0;
    for(i=0;i<5;i++){
        clock=0;
        sc_start(1,SC_NS);
        clock=1;
        sc_start(1,SC_NS);
    }
    ...
}
```

No.

Date

SystemC

[Work Example]

#add1.cpp

```
#include "systemc.h"

SC_MODULE(BIT_ADDER){
    sc_in <sc_logic> a,b,cin;          // Input
    sc_out <sc_logic> sum,cout;        // Output

    SC_CTOR(BIT_ADDER){                // Constructor Block
        SC_METHOD(process);            // Declare method
        sensitive << a << b << cin;    // Sensitivity list
    }

    void process(){                    // Define the functionality
        sc_logic aANDb, aXORb, cinANDaXORb;   // Declare variables of type "logic"
        aANDb = a.read() & b.read();
        aXORb = a.read() ^ b.read();
        cinANDaXORb = cin.read() & aXORb;

        sum = aXORb ^ cin.read();
        cout = aANDb | cinANDaXORb;
    }
};                                     // semicolon
```

#add1_tst.cpp

```
#include "system.h"                    // Write to the inputs and read the outputs

SC_MODULE(testbench){
    sc_out <sc_logic> A_p, B_p, CIN_p;
    sc_in  <sc_logic> SUM_p, COUT_p;

    SC_CTOR(testbench){                // CTOR: Construct Block
        SC_METHOD
        SC_THREAD(process);
    }

    void process(){
        A_p = SC_LOGIC_0;
        B_p = SC_LOGIC_0;
        CIN_p = SC_LOGIC_0;
        wait(10, SC_NS);
        print();
        sc_stop(); // End Simulation
    }

    void print(){
        cout << "At time " << sc_time_stamp() << " :: ";
        cout << "(a, b, carry_in): ";
        cout << A_p.read() << B_p.read() << CIN_p.read();
        cout << " (sum, carry_out) : " << SUM_p.read() << COUT_p.read() << endl;
    }
};
```

#add1_main.cpp

```
#include "add1.cpp"
#include "add1_tst.cpp"

int sc_main(int argc, char* argv[]){
    //Declare signals to be tied to the modules
    sc_signal <sc_logic> A_s, B_s, CIN_s, SUM_s, COUT_s;

    //Instantiate
    ///Must associate the inputs and outputs in EXACT ORDER
    BIT_Adder
    BIT_ADDER adder1("BitAdder1");
    adder1 << A_s << B_s << CIN_s << SUM_s << COUT_s;

    testbench test1("TestBench1");
    test1 << A_s << B_s << CIN_s << SUM_s << COUT_s;

    sc_start(2000, SC_NS);

    return 0;
}
```

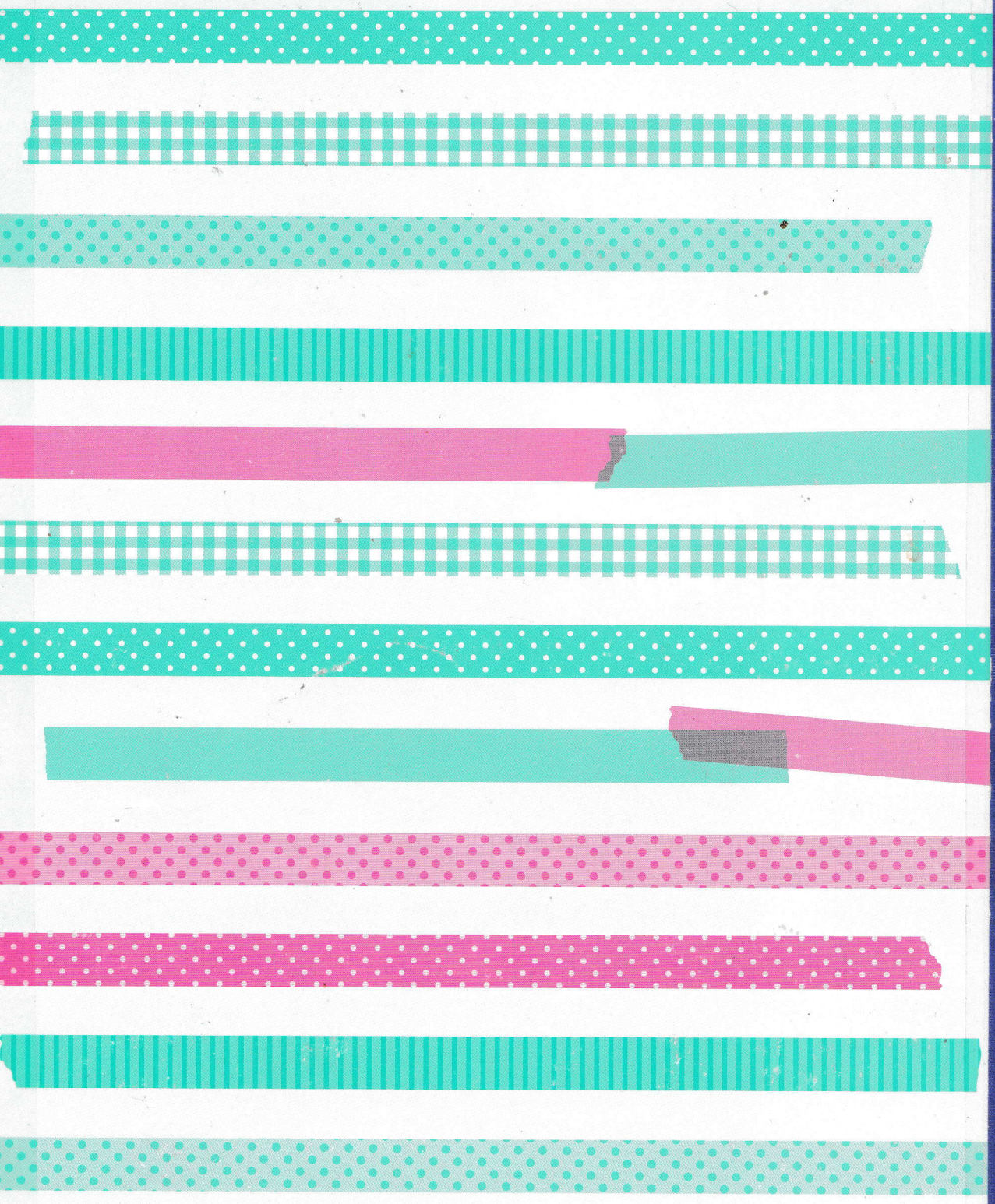
将Testbench模块输出接入Adder模块输入

PS: 主体分为三层

① Module层，加法器的代码

② Testbench层，生成输入逻辑，读取仿真结果

③ Top-level层，实例化模块，模块间接线



Campus 无线装订本
B5 40页
WCN-CNB1430

6 937748 309932
MADE IN CHINA

国誉商业(上海)有限公司
http://www.kokuyo.cn/st/
上海市奉贤区人杰路128号
TEL：400-820-0798　FAX：021-3255-8508
产地：上海市　QB/T1438-2007合格
B5　252×179mm

● 采用日本进口纸张，纸质细滑，牢固不掉页。