

Hardware Design I Chap. 4 Representative combinational logic

Computing Architecture Lab.
Hajime Shimada
E-mail: shimada@is.naist.jp

Already optimized circuits

- There are many optimized circuits which are well used
 - You can reduce your design workload
 - You can use faster one than your design :-P
- Some of them has different optimization level
 - Optimized for logic gates reduction
 - Optimized for operating speed



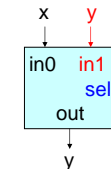
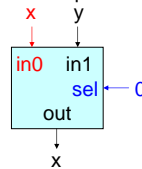
Outline

- Data path controlling circuits
 - Multiplexer/demultiplexer
 - Buffer/Three state buffer/Bi-directional buffer
 - Encoder/decoder
- Arithmetic circuits
 - Adder
 - Comparator/Majority vote
 - Shifter
 - Multiplier
 - Divider



Multiplexer (1/2)

- A circuit which outputs one of the inputs
 - Also called "Selector"
- e.g. 2-1 MUX (2-input 1-output multiplexer)
 - Output the value of "in0" if the input of "sel"=0
 - Output the value of "in1" if the input of "sel"=1



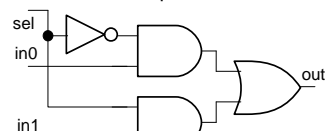
Relationship between inputs and output

sel	out
0	in0
1	in1



Multiplexer (2/2)

- Logical expression of 2-1 MUX:
 $out = (sel)'(in0) + (sel)(in1)$
- Assume that "sel" signal controls open/close of AND gate
- You can easily to extend logical expression to much more inputs with above design



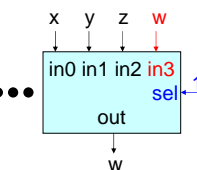
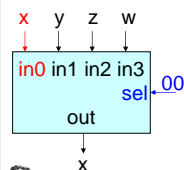
Truth table of 2-1 MUX

sel	in0	in1	out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



4-1 MUX (1/2)

- The input of "sel" becomes 2-bit width
 - I denote each bit of them as "sel₁" and "sel₀"
- The truth table becomes 6-value

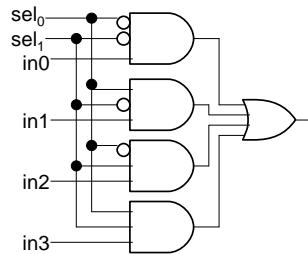


sel ₁	sel ₀	out
0	0	i0
0	1	i1
1	0	i2
1	1	i3



4-1 MUX (2/2)

- Assume that "sel" signal controls open/close of AND gate
 - "sel" = (0,0) opens "in0" gate
 - "sel" = (0,1) opens "in1" gate
 - "sel" = (1,0) opens "in2" gate
 - "sel" = (1,1) opens "in3" gate

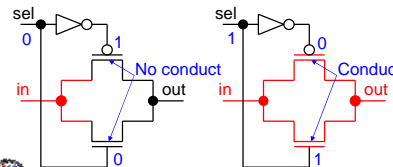


Multiplexer with transmission gate (1/2)

- Transmission gate
 - The circuit which can control conductivity
 - Input and output is conducted if "sel"=1
 - Warning: There's no current drive ability
- High impedance status (noted as Z)
 - The node is not connected either Vdd or Gnd

Operation	
sel	out
0	Z
1	in

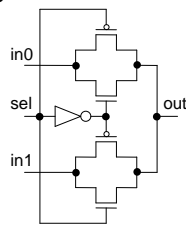
Truth table		
sel	in	out
0	0	Z
0	1	Z
1	0	0
1	1	1



Multiplexer with transmission gate (2/2)

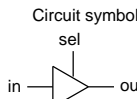
- Much simpler than MUX with logic gates
- Warning: There's no current drive ability (= output drive ability)
 - Current drive ability is depends on the logic gate before transmission gate
 - You have to increase drive ability of prior gate depending on outputs of transmission gate

sel	out
0	in0
1	in1



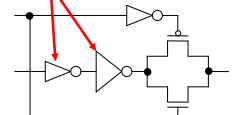
Three state buffer (tri-state buffer)

- A buffer which can output disconnected status
 - Buffer: a circuit which amplifies signal strength
- Assuming two not gates which drives output current before transmission gate
 - Strictly speaking, the buffer and transmission gate is unified
- Also called tri-state buffer



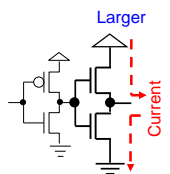
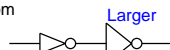
Truth table		
sel	in	out
0	0	Z
0	1	Z
1	0	0
1	1	1

Amplifies signal
(source of current)



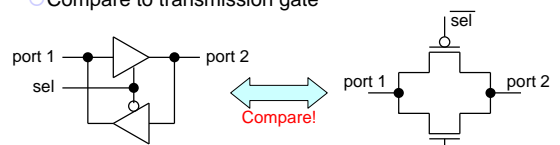
Buffer

- A circuit which amplifies signal strength
 - The current of amplified signal is comes from internal of the buffer
- Usually, we utilize larger (wide gate width) FET to drive much current
- Usage
 - Emphasize signal to drive much gates in output side
 - Emphasize signal to drive long signal line
- Variations
 - Implement NOT gates separately
 - Utilize negated output



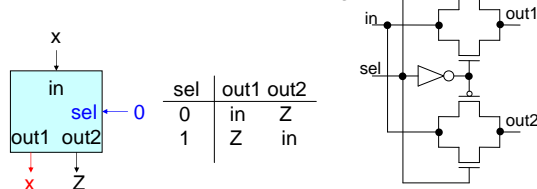
Bi-directional buffer

- A buffer which can control signal flow
 - The signal flows port 2 to port 1 if sel=0
 - The signal flows port 1 to port 2 if sel=1
- Note that the port 1 and port 2 is separated in electrical viewpoint
 - Compare to transmission gate



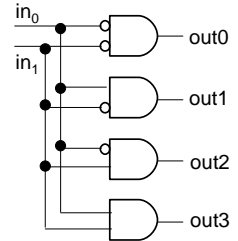
Demultiplexer

- The opposite operation to multiplexer
- The output which has not elected becomes high impedance status
- Constructed with transmission gate



Decoder

- The circuit which output 1 signal to corresponding output from input value
 - Assume that a multiplexer with logic gate which has no input
- The output is also called "1-hot code"



Encoder

- A circuit which outputs the number with binary notation which is corresponding to inputs
 - Opposite function to decoder
 - The output value under multiple input is undefined

in0	in1	in2	in3	out ₁	out ₀
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

Priority encoder

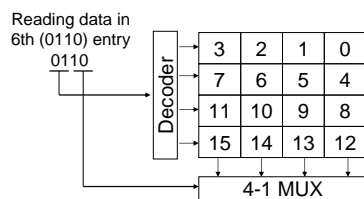
- The encoder which gives priority to specified order
 - It can tolerate multiple inputs
 - e.g. The priority encoder which has priority to smaller inputs

Priority encoder which has priority to smaller inputs

in0	in1	in2	in3	out ₁	out ₀
1	0	0	0	0	0
0	1	0	0	0	1
1	1	0	0	0	0
0	0	1	0	1	0
1	0	1	0	0	0
0	1	1	0	0	1
1	1	1	0	0	0
0	0	0	1	1	1
1	0	0	1	0	0
0	1	0	1	0	1
1	1	0	1	0	0
0	0	1	1	1	0
1	0	1	1	0	0
0	1	1	1	0	1
1	1	1	1	0	0

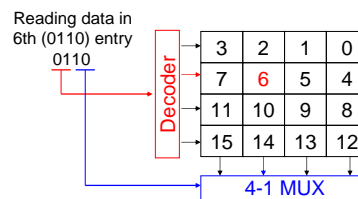
Reading table with decoder and multiplexer (1/2)

- We can read data in table organization by utilizing decoder and multiplexer
 - As shown in Chap. 5, we can minimize storage by utilizing table organization



Reading table with decoder and multiplexer (2/2)

- Operation
 - Select row by inputting higher side bits into decoder
 - Select column by inputting lower side bits into 4-1 MUX
- Widely used in RAM, flash memory, and so on



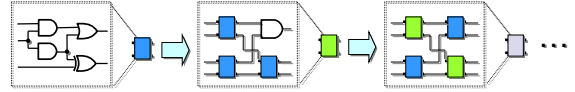
Outline

- Data path controlling circuits
 - Multiplexer/demultiplexer
 - Buffer/Three state buffer/Bi-directional buffer
 - Encoder/decoder
- Arithmetic circuits
 - Adder
 - Comparator/Majority vote
 - Shifter
 - Multiplier
 - Divider



How to design arithmetic circuits?

- From 1-bit arithmetic to multi bit arithmetic
 - Design and optimize 1-bit module
 - Under considering expansion to multi bit
 - Create multi bit circuit by utilizing 1-bit module
 - Similar to create program with function call



- Special technique for optimizing arithmetic circuits
 - Utilize characteristic of binary integer
 - Optimize under usual algebra



The notation of integer in binary

- We can represent 0 to 2^n-1 integer with n-bit binary notation (if we consider positive value)

$2^{n-1} 2^{n-2} \quad \quad \quad 2^3 \ 2^2 \ 2^1 \ 2^0$ Add weight to each digit
 14

0	0	...	0	1	1	1	0
---	---	-----	---	---	---	---	---

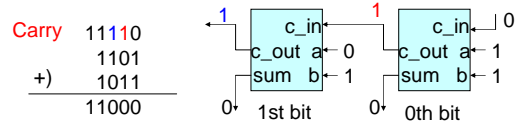
- We use two's complement to represent signed integer (detail: subtraction circuit)

- We can represent it -2^{n-1} to $+2^{n-1}-1$
- e.g. 8-bit signed integer with two's complement can represent from -128 to +127



Addition of binary integer

- Addition of 1-bit
 - $0+0=0, 0+1=1, 1+0=1, 1+1=10$ Carry
- By considering carry, an addition of one digit becomes addition of three 1-bit
 - Addition of augend (a), addend (b), and carry (c)



Addition of binary integer

- Generalized notation of n-bit binary integer
 - The result becomes (n+1)-bit binary integer
 - $c_0 = 0$
 - $c_n = s_n$

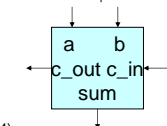
$$\begin{array}{r}
 c_n \ c_{n-1} \ c_{n-2} \ \dots \ c_1 \ c_0 \\
 +) \ a_{n-1} \ a_{n-2} \ \dots \ a_1 \ a_0 \\
 \quad b_{n-1} \ b_{n-2} \ \dots \ b_1 \ b_0 \\
 \hline
 s_n \ s_{n-1} \ s_{n-2} \ \dots \ s_1 \ s_0
 \end{array}$$



1-bit full adder

- Definition of the circuit
 - Inputs: two 1-bit binary and 1-bit carry input from lower digit
 - Operation: sum all of inputs
 - Outputs: sum and carry output
- Half adder
 - An adder which has no carry input

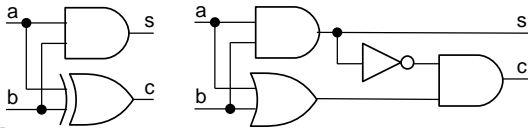
a	b	c _{in}	c _{out}	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Implementation of half adder

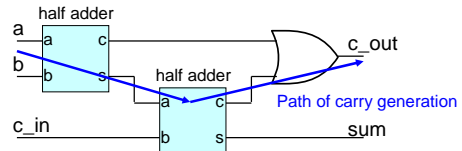
- Usually implemented with XOR gate
 - Much smaller gate number than AND-OR organization

a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



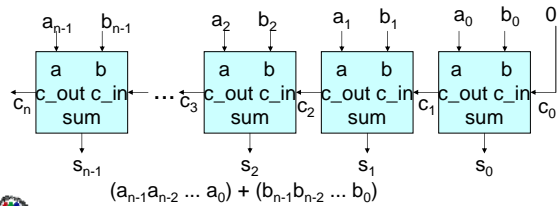
Implementation of full adder

- We can create full adder with half adder
- Usually, the path of carry generation becomes critical path
 - Critical path: the path has longest route



n-bit ripple carry adder

- An adder which layouts n of 1-bit full adder
 - Called ripple carry adder (RCA)
- The calculation time is **in proportion to n**



The RCA is slow

- Why RCA is slow?
 - c_5 will be defined after c_4 has defined
 - c_4 will be defined after c_3 has defined
 - ...
- c_5 is defined under sequential definition
- n-bit addition requires $O(n)$ time
 - Definition of $O(n)$:

	11110
	1101
+	1011
	11000

- Assuming function $f(n)$ and $G(n)$
- $f(n) = O[g(n)]$ if constant c and n_0 which satisfy $f(n) \leq c \cdot g(n), n \geq n_0$
- Note that $f(n) > 0, g(n) > 0$

Carry look-ahead adder

- The critical point is carry
 - Are there any way to speeding up carry generation?
 - Idea: separate carry to two category
 - Generation of carry: $g_i = a_i \cdot b_i$
 - The carry must occur in this digit
 - Propagation of carry: $p_i = a_i + b_i$
 - The carry will occur if carry from (i-1) has arrived
 - Note that the generation of g_i and p_i are easy
- Assuming $(a_{n-1}a_{n-2} \dots a_0) + (b_{n-1}b_{n-2} \dots b_0)$

Extracting carry with $g_i, p_i,$ and c_0

- C_n becomes $n+1$ sum of term of $n+1$ literal

- $c_1 = g_0 + p_0 c_0$
- $c_2 = g_1 + p_1 c_1 \rightarrow c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$ Calculated without c_1
- $c_3 = g_2 + p_2 c_2 \rightarrow c_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$
- \vdots
- $c_n = g_{n-1} + p_{n-1} c_{n-1} \rightarrow c_n = g_{n-1} + p_{n-1} g_{n-2} + p_{n-1} p_{n-2} g_{n-3} + \dots + p_{n-1} \dots p_k g_{k-1} + \dots + p_{n-1} \dots p_0 c_0$

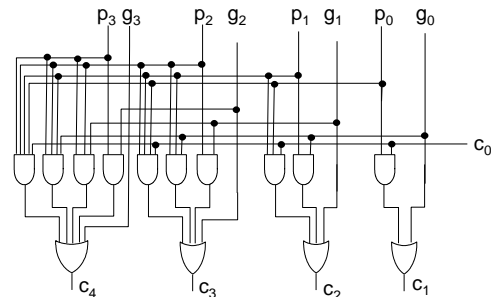
The condition which C_n becomes 1

- Sum of later itemize
 - $g_{n-1} = 1$
 - g_{n-2} was propagated after $n-1$ digit ($=p_{n-1}$)
 - g_{n-3} was propagated after $n-2$ digit ($=p_{n-1} \cdot p_{n-2}$)
 -
 - g_{k-1} was propagated after k digit
 -
 - c_0 was propagated through all digits

$$C_n = g_{n-1} + p_{n-1}g_{n-2} + p_{n-1}p_{n-2}g_{n-3} + \dots + p_{n-1} \dots p_k g_{k-1} + \dots + p_{n-1} \dots p_0 c_0$$



4-bit CLA



The characteristic of CLA

- It can calculate c_i in parallel
- Much complicated than RCA
- Calculation time becomes $O(\log n)$
 - c_n becomes sum of $n+1$ term
 - Each term is consist of $n+1$ literals
 - > If we implement it with balance tree, the height becomes $\log n$



Quiz

- How long does CLA requires to calculate 64-bit value with NAND2 gate delay?
 - Around 8 NAND2 gate delay
 - Around 12 NAND2 gate delay
 - Around 16 NAND2 gate delay
 - Around 20 NAND2 gate delay
- 64-bit RCA requires around 129 NAND2 gate delay



Answer

- 2. Around 12 NAND2 gate delay
 - 1 NAND2 delay for prepare p_i and g_i
 - About 8 NAND2 delay for prepare c_i from p_i and g_i
 - 3 NAND2 delay for calculate s_i
- Result of practical implementation
 - Alpha 21264 processor utilizes 12 FO4 delay for each pipeline stage pipeline stage -> Chap. 11
 - It execute 64-bit arithmetic in 1 pipeline stage



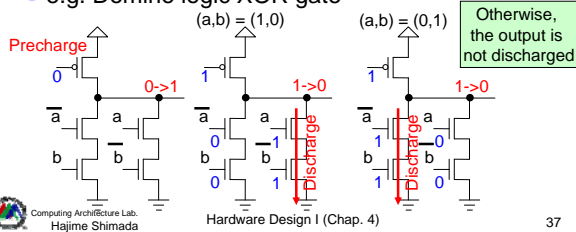
Explore of faster adder

- Adder is one of the important circuit so that there's many implementations
- Examples
 - Carry select adder
 - Conditional sum adder
 - Carry skip adder
 - Carry bypass adder
 - Carry complete adder
 - Domino logic adder for Pentium 4



Outline of domino logic

- Operate with precharge and evaluation (=discharge)
 - If input satisfies condition, output is discharged
- e.g. Domino logic XOR gate



Twos complement (1/2)

- A method which gives negative weight for most significant bit

$$-2^{n-1} 2^{n-2} \dots 2^3 2^2 2^1 2^0$$

0	0	...	0	1	1	1	0
---	---	-----	---	---	---	---	---

- e.g. Twos complement with 8-bit width

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	=	-2^7	=	-128
1	0	0	0	0	0	0	0				
1	0	0	0	0	0	0	1	=	$-2^7 + 2^0$	=	-127
0	0	1	0	0	0	0	1	=	$2^6 + 2^0$	=	65
0	1	1	1	1	1	1	1	=	$2^6 + 2^5 + \dots + 2^0$	=	127

Twos complement (2/2)

- We can represent -2^{n-1} to $+2^{n-1}-1$
 - e.g. Twos complement with 8-bit width
- Why we do not use independent sign bit?
 - It creates "positive 0" and "negative 0"
 - > redundant!!!

Example of 8-bit width

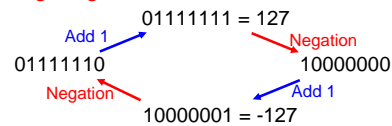
01111111	"+127"
...	...
00000001	"+1"
00000000	"0"
11111111	"-1"
...	...
10000000	"-128"

Independent sign bit

0 00000000	"positive 0"
1 00000000	"negative 0"

How to create twos complement?

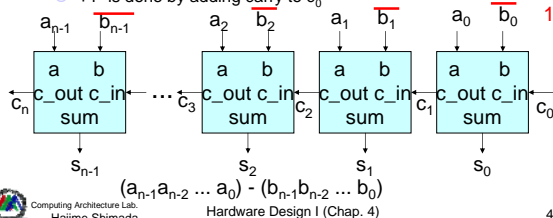
- You can gain inverse sign of twos complement by **negating all bits** and **add 1 to it**



- Why it becomes twos complement?
 - A negation of m becomes $-2^{n-1} + (2^{n-1}-1) - m$
 - By adding 1 to above one, we can gain $-m$
 - Note that $-m = -2^{n-1} + (2^{n-1}-1) - m + 1$

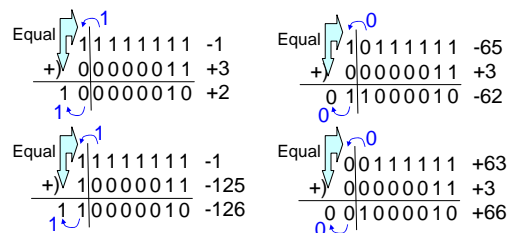
Subtraction with twos complement

- Create twos complement of subtrahend and add it with adder
- Organization of the circuit
 - Negate all bits before adder
 - "+1" is done by adding carry to c_0



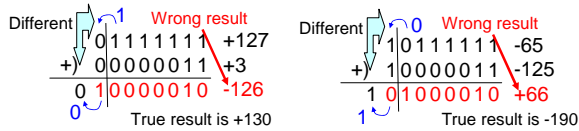
The sign bit after adding positive and negative values

- Assume signed 8-bit world
- If carry in and carry out of the sign bit are same, you only have to add them



The sign bit after adding positive and negative values

- Sign bit: equals to most left bit (MSB: most significant bit)
- If carry in and carry out of the sign bit are different, you have to treat it **overflow**
- The result exceeds range which can be represented with signed 8-bit
 - From -128 to +127



Short exercise

- Show arithmetic result under signed 8-bit world
 - Show both binary and decimal notation
 - Notate "overflow" if it occurs

$$\begin{array}{r} 10111101 \text{ -67} \\ +) 00100011 \text{ +35} \\ \hline \end{array} \quad \begin{array}{r} 00111111 \text{ +63} \\ +) 01111111 \text{ +127} \\ \hline \end{array}$$

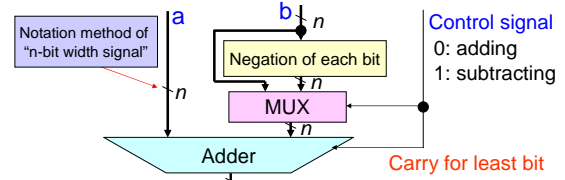
Answer

$$\begin{array}{r} 10111101 \text{ -67} \\ +) 00100011 \text{ +35} \\ \hline 01110000 \text{ -32} \end{array} \quad \begin{array}{r} 00111111 \text{ +63} \\ +) 01111111 \text{ +127} \\ \hline 01011110 \text{ -66} \end{array}$$

•Overflow!
•True result is +190

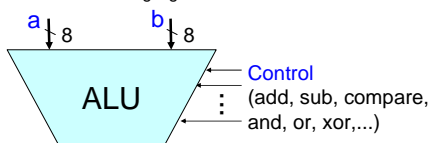
Unify adder and subtractor

- We can unify adder and subtractor
 - Control signal provide carry for least bit which is required to create two's complement



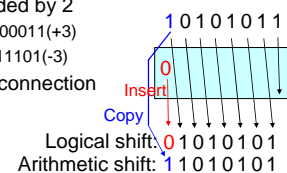
ALU (Arithmetic Logic Unit)

- Usually, we implement multiple arithmetic function to one circuit
- We can share logic gates between arithmetics
 - e.g. AND/XOR operation of a and b are partial result of half adder
 - We can save number of logic gates



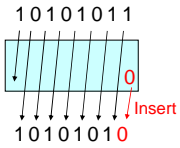
1-bit right shift

- Move 1 digit to right
 - Least significant bit (LSB) is banished
 - MSB differs between shift method
 - Logical shift: insert 0
 - Arithmetic shift: insert prior MSB
- The result becomes divided by 2
 - e.g. 00000111(+7) -> 00000011(+3)
 - e.g. 11111010(-6) -> 11111101(-3)
- Achieved with only wire connection



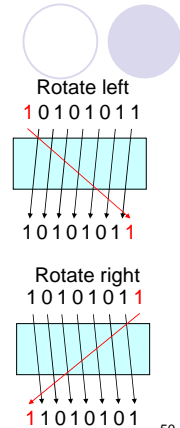
1-bit left shift

- Move 1 digit to right
 - MSB is banished
 - 0 is inserted into LSB
- The result becomes multiplied by 2
 - e.g. 00000111(+7) -> 00001110(+14)
 - e.g. 11111010(-6) -> 11110100(-12)
- Also achieved with only wire connection
- You have to consider overflow if you execute arithmetic shift
 - e.g. 10000000(-128) -> 00000000(0) **Overflow!**



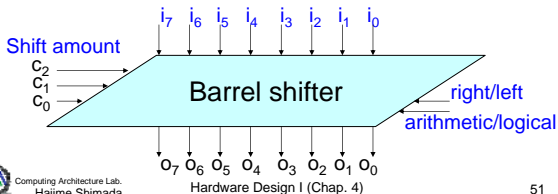
1-bit rotate left (or right)

- 1-bit rotate left
 - Move 1 digit to left
 - MSB is moved to LSB
- 1-bit rotate right
 - Move 1 digit to right
 - LSB is moved to MSB



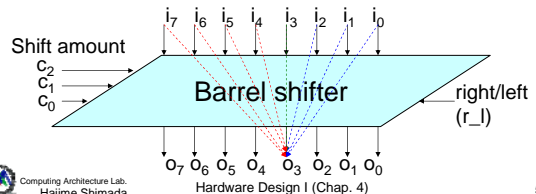
Barrel shifter

- A circuit which can achieve arbitrary shift
 - Usually, it permits several shift related operations
- n-bit shift gives result of multiplied by 2^n or divided by 2^n



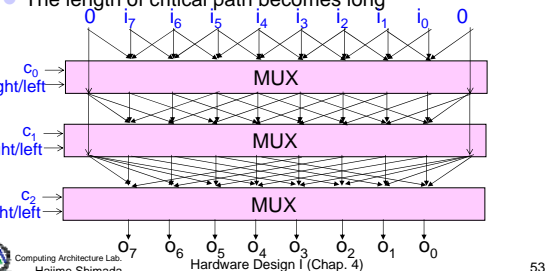
One implementation of barrel shifter

- Creating logical expression of each outputs and construct two level logic
- e.g. $O_3 = \overline{r_l}(i_0 \cdot C_1 \cdot C_0 + i_1 \cdot C_1 + i_2 \cdot C_0) + C_2 \cdot C_1 \cdot C_0 + r_l(i_4 \cdot C_0 + i_5 \cdot C_1 + i_6 \cdot C_1 \cdot C_0 + i_7 \cdot C_2)$
 - Assuming $r_l = 1$ under right shift



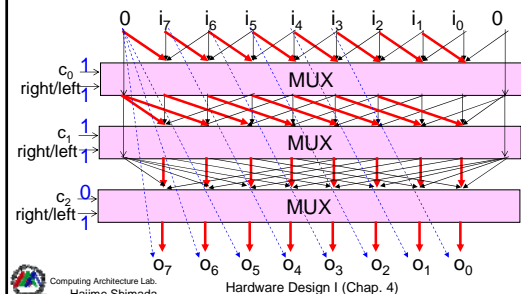
The other implementation of barrel shifter

- Cascaded MUX which selects 2^n bits right shift, 2^n bits left shift, or no shift
- The length of critical path becomes long




Example of operation

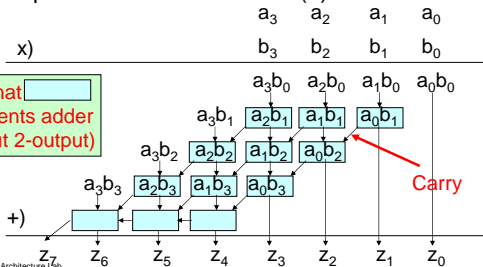
- 3-bit right shift



Array multiplier

- Align adder to array
- The operation time becomes $O(n)$

Note that  represents adder (3-input 2-output)

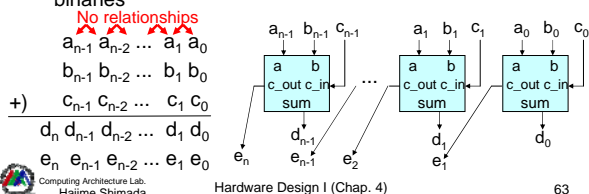


Outline of Wallace tree multiplier

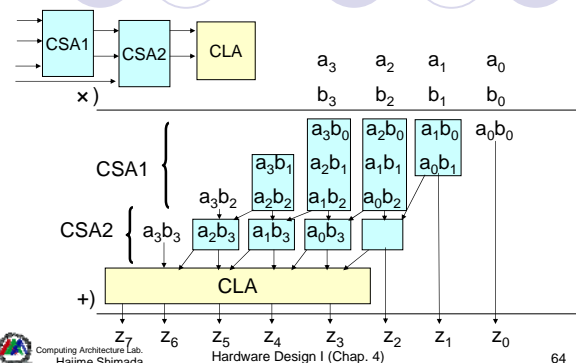
- One digit of n -bit multiply becomes **summation of n binaries**
- If we utilize **carry save adder**, we can construct 3-2 arithmetic tree
 - Group 3 binaries from summation of n and apply carry save addition
 - The result becomes summation of $(2/3 \times n)$
 - Back to 1. until the summation becomes summation of 2 (usual addition)
- It can operate multiply with $O(\log n)$

Carry save adder (CSA)

- An array of n full adders
- Output sum of 3 binary inputs (2 binary outputs)
 - There's no carry propagation
 - Operation time is constant (independent to number of inputs)
- It can quickly translate sum of 3 binaries to sum of 2 binaries



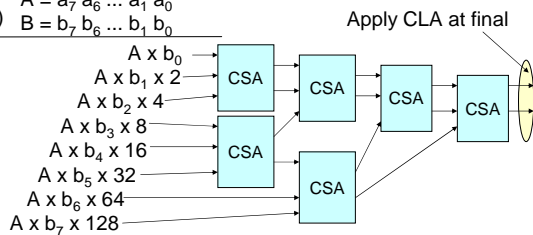
3-2 Wallace tree multiplier



Example of 8-bit multiply

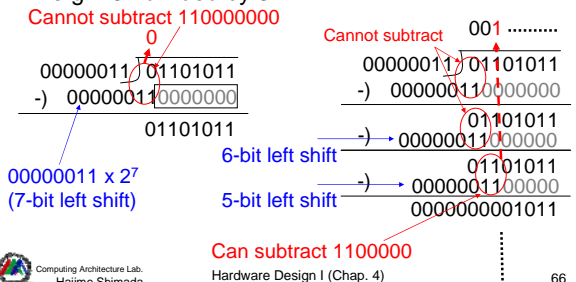
- Number of summation becomes $2/3$ under one CSA
- After $O(\log n)$ of CSAs, it becomes sum of 2 binaries

$$\begin{array}{r} A = a_7 a_6 \dots a_1 a_0 \\ \times B = b_7 b_6 \dots b_1 b_0 \end{array}$$



Division

- Implement computation on paper frankly
- e.g. 107 divided by 3



Quiz

- What is the correct organization of 8-1 MUX?



Answer

- Both 1 and 4 are correct answer
 - 1 is based on AND-OR logic gate based organization
 - 4 is based on transmission gate based organization

