# Logic Design (Part 3)

## Combinational Logic Devices (Chapter 3 + Notes)





## Digital Logic Circuits

- we can build the basic logic gates using transistors
- Can build any boolean function using these gates
  - Theory underlying design of Boolean functions ..Boolean Algebra
    - Optimize circuit using Karnaugh maps
- Power of abstraction….To build boolean functions, you can work with basic gates – no need to go down to the transistor level !!
- Use these gates as building blocks to build more complex combinational circuits
  - Combinational Logic Devices: Adder, Multiplier, Multiplexer, Decoder, …..
  - …any boolean function

## Definitions: Combinational and Sequential Logic Circuits

- A circuit is a collection of devices that are physically connected by wires
  - Combinational circuit
  - Sequential circuit
- In Combinational circuit the input determines output
- In sequential circuit, the input and the previous 'state' (previous values) determine output and next 'state'
  - Need to 'remember' previous value – need memory device
  - Need circuit to implement concept of storage
  - This is out next topic…..

3

3

## Recall our Goal….

- Design a machine that translates from natural language to electrons running around to solve the problem
  - We now have devices (transistor) that can get electronics to run around based on input signals….and built logic gates using transistors
- Next: we want to build a computer
  - First step: Design a collection of logic devices that implement important functions that will be needed to build our computer
- S/W Analogy: When you write your software, you are using a collection of concepts, tools, IDEs and libraries
  - Each has been built, and tested, for you
  - All you have to do is combine them!

4

4

## Combinational Logic Devices

- We saw how we can build the simple logic gates using transistors and build any boolean function using these gates
- Use these gates as building blocks to build more complex combinational circuits
  - Decoder: based on value of n-bit input control signal, select one of $2^N$ outputs
  - Multiplexer: based on value of N-bit input control signal, select one of $2^N$ inputs.
  - Adder: add two binary numbers
  - …any boolean function
- SW Analogy: We are building a library of functions
  - To design your solution, you can use any device in the library!

5

5

## Three Devices we focus on…

- N-bit Adder
  - Can build Subtract using Adder
- Decoder
  - Decode a bit string
- Multiplexer
  - A channel selector
- Other useful combinational logic devices
  - Multipliers
  - Shifters (but may need storage)
  - Comparators (to compare two numbers)
  - …

6

6

## 1. N-bit Adder

- Add two N-bit numbers, represented in 2's complement
- Algorithm (for now): add corresponding bit positions, starting with least significant position, and propagate the carry bit leftward.
  - In practice: there are faster algorithms
  - Big-Oh Analysis:
    - To add N bit numbers how 'far' will the carry propagate ?

7

7

## Binary Addition

- Binary addition – just like base 10 (decimal) !
  - Add from right to left, propagating carry
  - Example using unsigned integers

carry

```
  10010 (18)      10010 (18)      01111 (15)
+ 01001 (9)     + 01011 (11)    + 00001 (1)
  11011 (27)      11101 (29)      10000 (16)
```

Key Observation: We add one bit at a time

therefore, building block is a 1-bit adder

*Use 1-bit adder to build N-bit adder!*

8

8

## 1-bit Adder

- Two inputs A, B and Two outputs: S (sum) and Carry out (C)
- Truth table:

| A | B | S | C |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

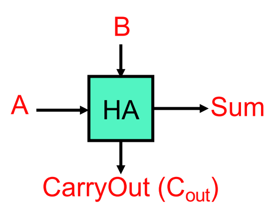
- Problem?
- This works only for bit 0 where there is no Carry-in
  - Called a half adder
- In general, we can have a carry-in input, so 3 inputs are A,B,$C_{in}$ (carry-in) and 2 outputs S, $C_{out}$ (carry out)



## Binary Arithmetic: Half Adder

- Logical Function: Half Adder, implement Carry Out:

| A | B | Sum | $C_{out}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |




Half Adder's Logic Function:

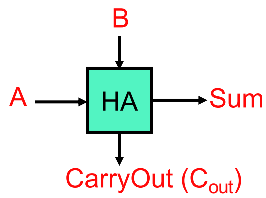
SUM=((A' AND B') OR (A AND B))'

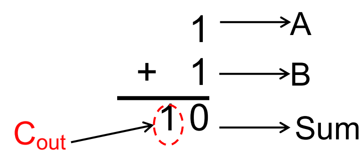
C=( (A' AND B') OR (A' AND B) OR (A AND B') )'

*Realize though: C= (A AND B), this isn't always best way!*

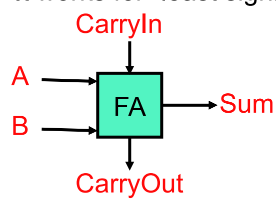
## Addition: Full Adders

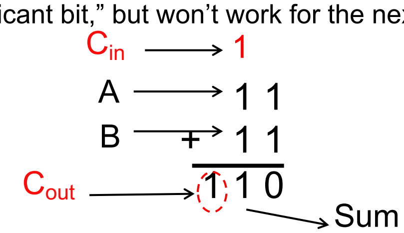
- There is a limit with the half adder
  - It can't implement multiple-bit addition




  - It works for "least significant bit," but won't work for the next



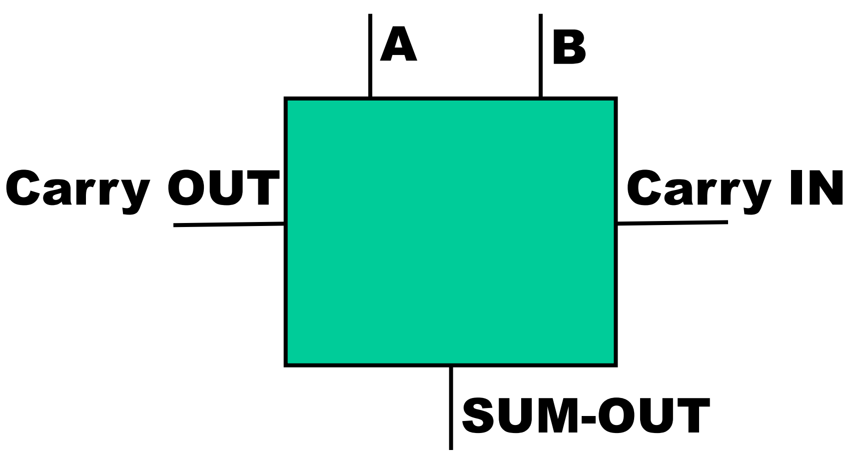

  - We need an adder that has 3 inputs and 2 outputs

3-11

11

11

## Full Adder




12

12

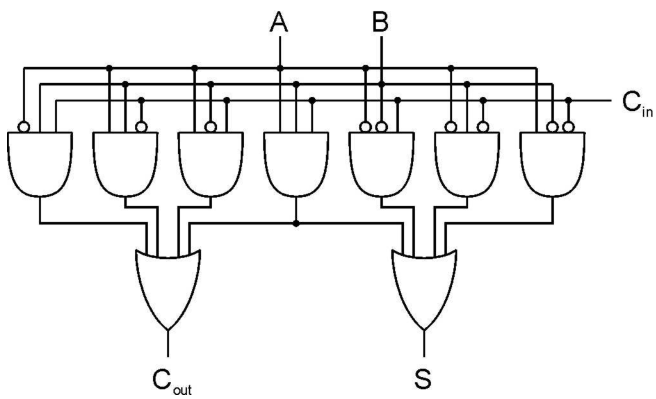
## Truth Table for Full-Adder

| A | B | Carry In | Out | Carry Out |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |



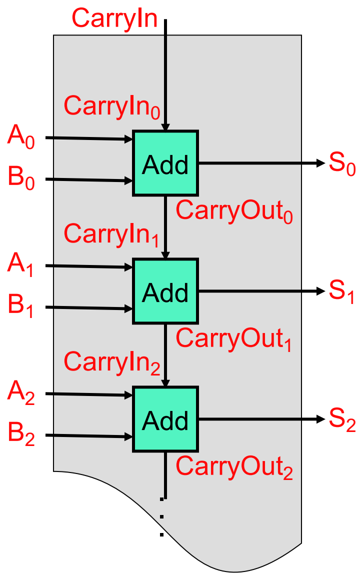
## 1-bit Full Adder

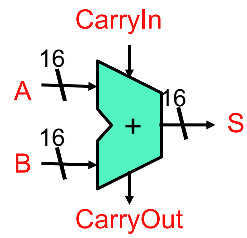
- Add two bits and carry-in, produce one-bit sum and carry-out.




| A | B | $C_{in}$ | S | $C_{out}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

## N-bit Adder




CarryOut: useful for detecting overflow

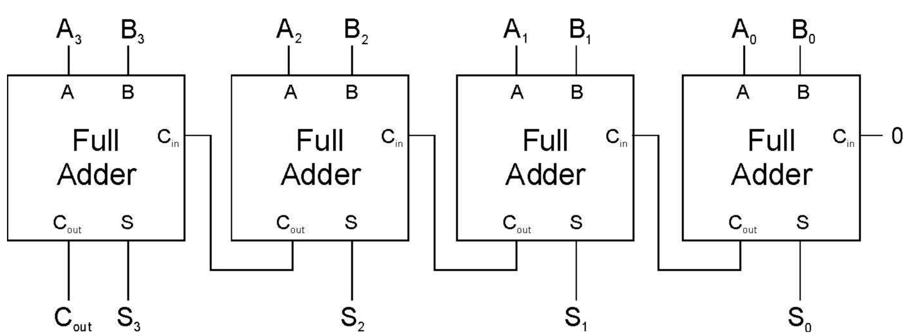
CarryIn: assumed to be zero if not present

3-15

15

15

## Four-bit Adder




16

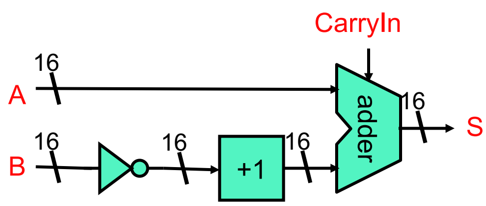
16

## How about a "subtractor?"

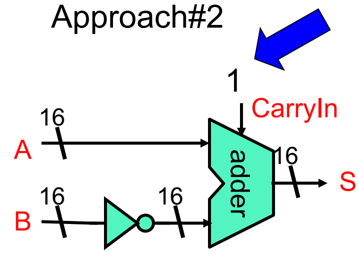
- Build a subtracter from out multi-bit adder
  - Calculate A – B = A + –B
  - Negate B
  - Recall –B = NOT(B) + 1

*We "carry in" a 1*
*(no longer need incrementer)*

Approach#1




Approach#2




17

17

## The Decoder

- Useful for recognizing a particular bit pattern of 0's and 1's
- Connection to Computer Organization:
  - Program consists of instructions -- coded in binary (0's and 1's)
  - We want to look at a bit string for the instruction and determine what the instruction is
    - Is it an ADD or a MULT or a GOTO or…..
    - each instruction is given a unique encoding & decoder looks at the encoding and determines which ONE of the instructions the code corresponds to (i.e, which instruction has to be executed)
- In S/W, a "case"/switch statement:
  - One of the cases will be evaluated depending on value of 'input'

18

18

## Switch (case) statement in C

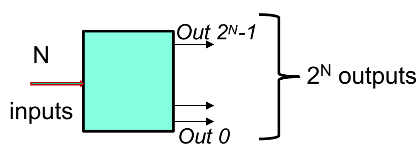
```
int x;
….
switch(x) {
        case 0:        /* if x=0 call func Kevin */
                Kevin(); /* ex: Kevin does add */
                break;
        case 1:        /* if x=1 call func Graham */
                Graham();
                break;
        case 2:        /* if x=2 call func Sarah */
                Sarah(); /* ex: Sarah does AND */
                break;
        case 3:        /* if x=3 call func Linnea */
                Linnea();
                break;
        default: printf(”invalid value of x”\n);
                break;
}
..
```

19

19

## N-$2^N$ Decoder

- N inputs – these represent the binary encoding of the $2^N$ Outputs
  - Ex: if N=2, then 4 outputs 0,1,2,3, encoded to be ‘switched on’ when inputs are one of 00, 01, 10, 11 respectively
- Schematic:




Notational Convenience: Typically we label the outputs from 0 to $2^N$-1: $x_0, x_1, \ldots, x_{N-1}$
Output $x_i$ =1 if decimal value of input = i

20

20

## Designing a Decoder: Truth table
## 4 output lines x0,x1,x2,x3 & 2 inputs a1,a0

From truth table, design circuit:

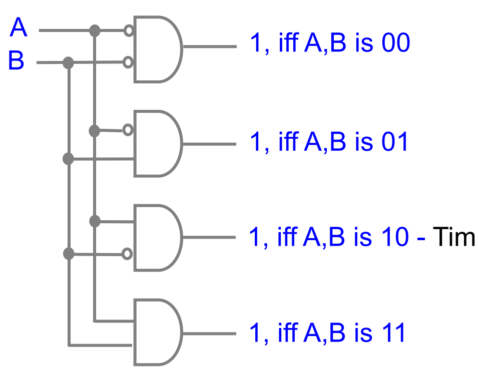
$x_0 = a_1'.a_0'$ *(i.e., (NOT $a_1$) AND (NOT $a_0$))*

$x_1 = a_1'.a_0$ $x_2 = a_1.a_0'$ $x_3 = a_1.a_0$

| $a_1$ | $a_0$ | $x_0$ | $x_1$ | $x_2$ | $x_3$ |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

21

## Decoder




**2-bit decoder**
**(4 input decoder)**

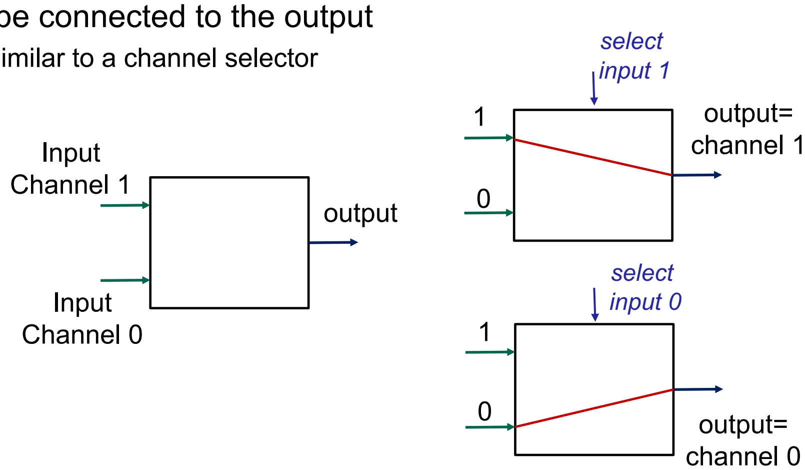
- An n input decoder has $2^n$ outputs.
- $\text{Output}_i$ is 1 iff the binary value of the n-bit input is i.
- At any time, exactly one output is 1, all others are 0.

22

22

## The Multiplexer - selector

- Multiplexer (MUX) is a device that selects one of the inputs to be connected to the output
  - Similar to a channel selector

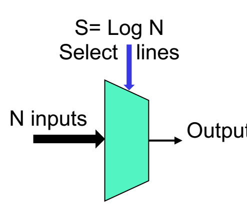



This is a 2-1 Multiplexer: Selects one of 2 inputs as the output 23

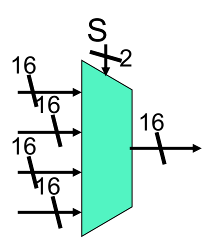
23

## N-1 Multiplexer

- Multiplexer selects one of the N inputs as the output
  - It needs $\log_2 N$ 'select lines' to determine which of the N inputs is selected to appear at the output
  - Schematic of a MUX:




- Multi-bit muxes
  - Can switch an entire "bus" or group of signals
  - Switch n-bits with *n* muxes with the same select bits
  - Built using 16 1-bit 4-1 MUXes and has same S

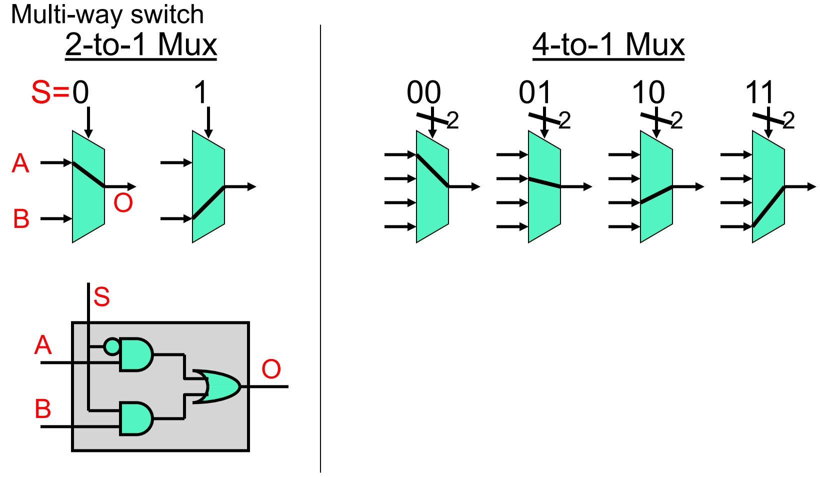



24

24

## The Multiplexer (MUX) – 2-1 and 4-1 MUX

- Selector/Chooser of signals *– Imagine Switching Railroad Tracks*
  - Multi-way switch



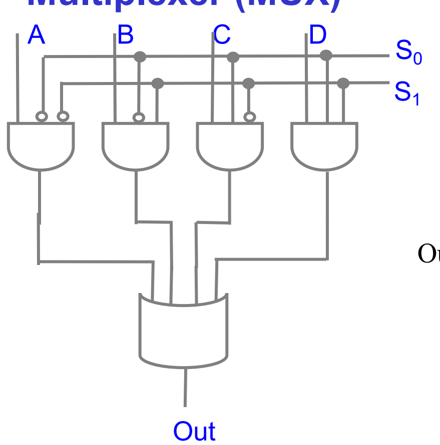

Input "S" *selects* A or B to attach to "O" *output*
Acts like an "IF/ELSE" statement

3-25 25



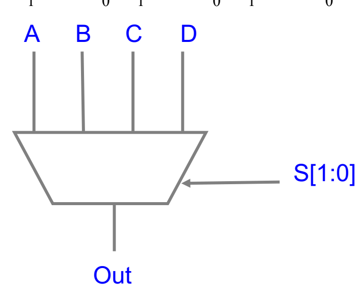
## Designing 4-1 MUX Using Logic Gates

### Multiplexer (MUX)




- In general, a MUX has
  - $2^n$ data inputs
  - n select (or control) lines
  - and 1 output.
- It behaves like a channel selector.

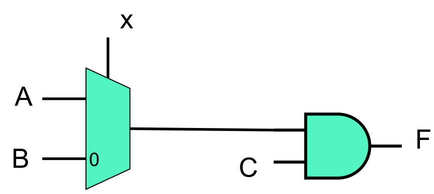
$$\text{Out} = A.\overline{S}_0.\overline{S}_1 + B.\overline{S}_0.S_1 + C.S_0.\overline{S}_1 + D.S_0.S_1$$




**A 4-to-1 MUX:**
**Out takes the value of A,B, C or D depending on the value of S (00, 01, 10, 11)**
Out = A if S=00 Out = B if S = 01
Out = C if S=10 Out=D if S=11

## Example: MUX in a circuit

- Inputs A,,B,C and x (select signal); Output F
- Devices/Gates: 2-1 MUX, AND gate




- If x=0, output of MUX = B and F= B.C
- If x=1, output of MUX = A and F = A.C
- Can write F = xAC + x'BC

28

28

## Combinational vs. Sequential

- Combinational Circuit
  - always gives the same output for a given set of inputs
    - ex: adder always generates sum and carry, regardless of previous inputs
- Sequential Circuit
  - stores information
  - output depends on stored information (state) plus input
    - so a given input might produce different outputs, depending on the stored information
  - useful for building “memory” elements and “state machines”

29

29

## Next . . Circuits with "memory"

- First we need to build a device that can store a bit
  - Using our current 'library' of gates
  - Building memory follows
- How to model sequential circuits/machines
  - Methodology for designing these machines: **Finite state machine**
  - Model as a directed graph
- How to we "synchronize" and "coordinate" the different pieces in the circuit….enter the CLOCK
- can we use a sequential circuit to "control" how computations take place in a processor ?
- Is a sequential circuit = Computer ?
  - Limitations of sequential machines..more in Foundations course

30

30

## Appendix

31

31

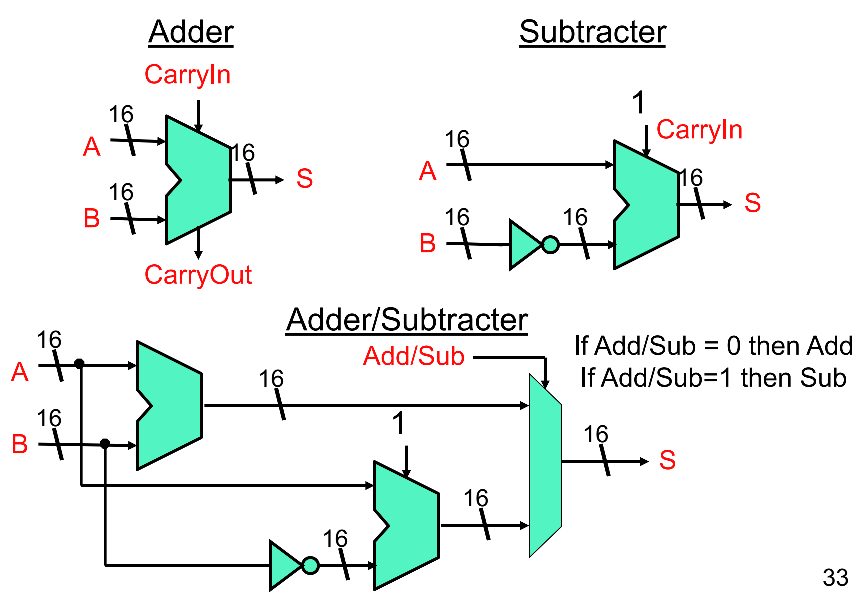
## A multi function Arithmetic Unit

- In a CPU, we'd like to do BOTH addition and subtraction
  - Can we give the CPU the ability to choose between two pieces of hardware?
  - Yes!
- Using a MUX to build a multifunction ALU

32

32

## Building an ALU using MUX
## Adder/Subtracter - Approach #1




33

33

## Adder/Subtracter - Approach #2 (Optimize HW)