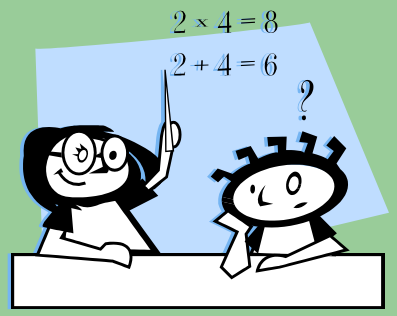
Lecture 4

Modeling, Analysis and Simulation in Logic Design



Dr. Yinong Chen

The Engineering Design Process

CAD

Define Problem and requirement

Research

Define Alternative solutions

Modeling

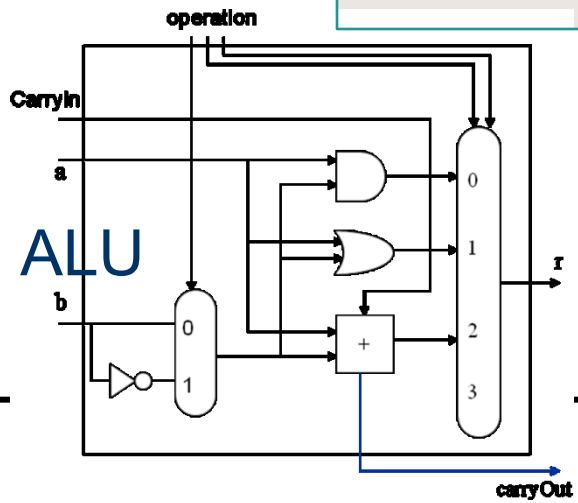
Analysis

Simulation

Prototyping

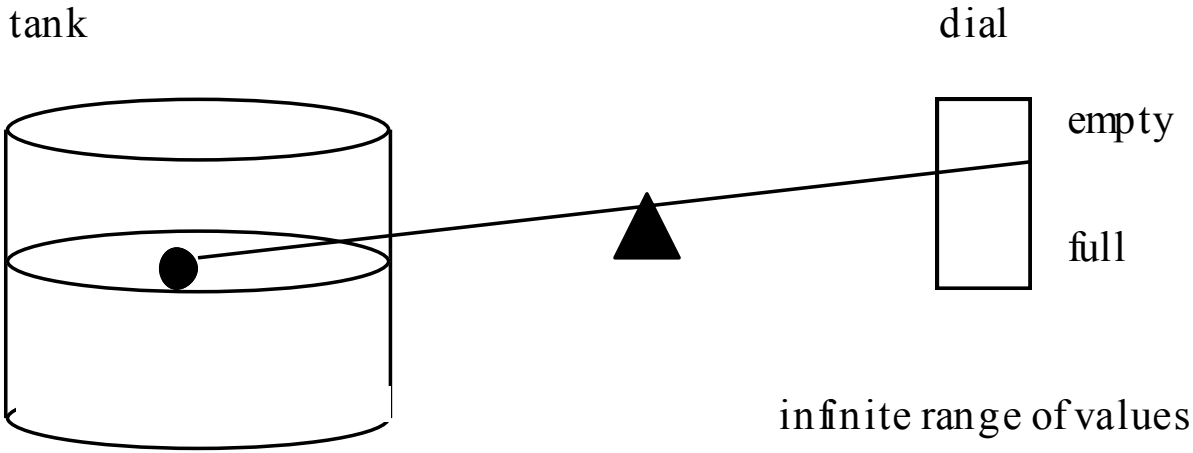
Final Selection

Implementation and testing

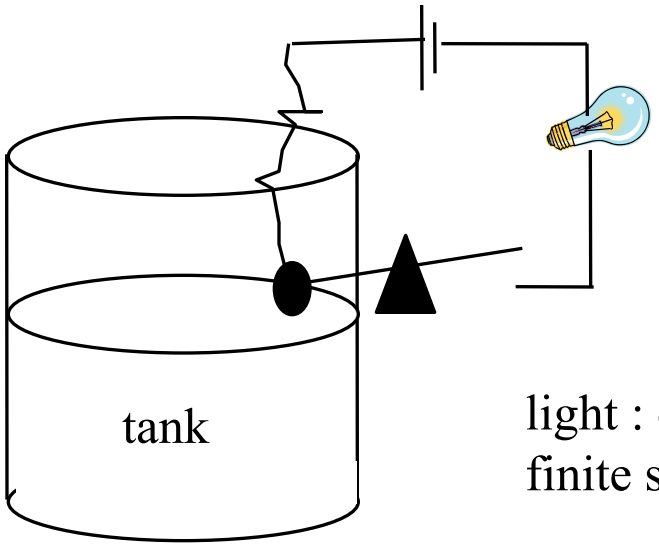


Contents

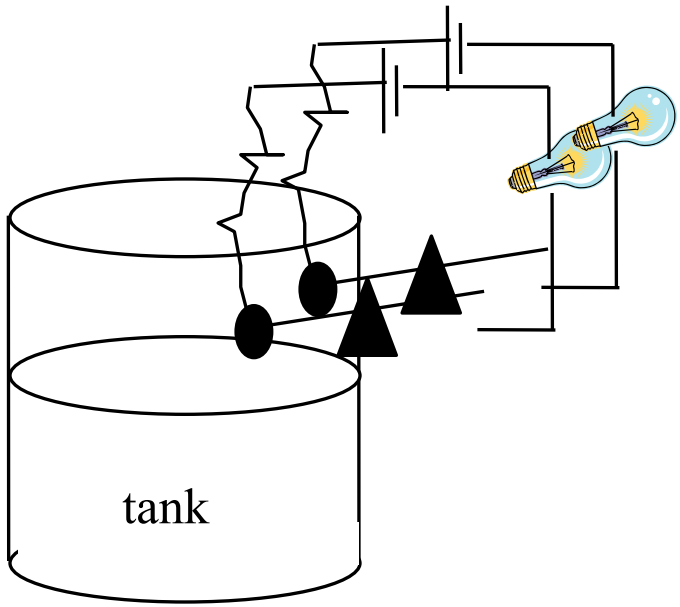
Logic Design: Analogue versus Digital



infinite range of values



light : on / off
finite set of values



Propositional Logic and its Elements

- Propositional logic is a language for modeling and specification
- Proposition: A statement that can either be true or false:
  - One plus two is three
  - There are two Nobel prize winners at Arizona State University
  - The sky is blue
  - How old are you?
  - You must ride a bike to school!
- Logic connectives
  - AND ($\wedge$), OR ($\vee$), NOT ($\neg$), IMPLIES ($\rightarrow$)
  - light is on $\rightarrow$ tank is full
  - (Light is off) $\wedge$ (bulb is not broken) $\rightarrow$ tank is empty
- Truth and falsity values – Truth Table

Truth Table is a Logic Specification/Model for Circuits

NOT

propositional variable	*statement*
P	¬ P
false	*true*
true	*false*

a	$\bar{a}$
0	1
1	0

AND

P	Q	P ∧ Q
false	*false*	*false*
false	*true*	*false*
true	*false*	*false*
true	*true*	*true*

a	b	a∧b
0	0	0
0	1	0
1	0	0
1	1	1

Truth Table (Contd.)

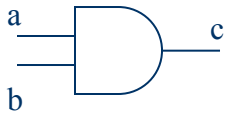
OR

P	Q	$P \vee Q$
false	*false*	*false*
false	*true*	*true*
true	*false*	*true*
true	*true*	*true*

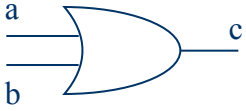
a	b	a∨b
0	0	0
0	1	1
1	0	1
1	1	1

Building Blocks of Digital Circuits

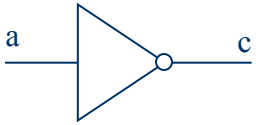
Building Blocks



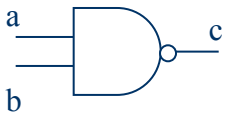
AND gate
$c = a \wedge b$



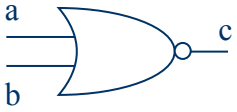
OR gate
$c = a \vee b$



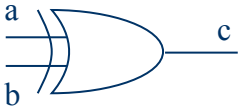
NOT
$c = \bar{a}$



NAND gate
$c = \overline{a \wedge b}$



NOR gate
$c = \overline{a \vee b}$



XOR gate
$c = \bar{a} \wedge b \vee a \wedge \bar{b}$

Truth Tables

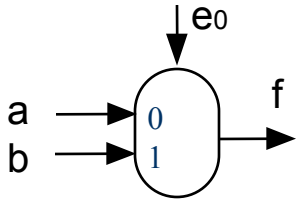
a	b	a∧b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a∨b
0	0	0
0	1	1
1	0	1
1	1	1

a	$\bar{a}$
0	1
1	0

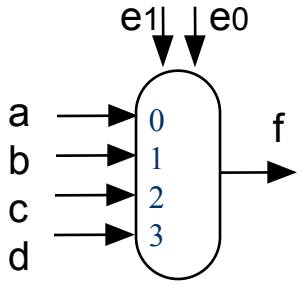
a	b	$\overline{a \wedge b}$	$\overline{a \vee b}$	$\bar{a} \wedge b \vee a \wedge \bar{b}$
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	0	0	0

2-1 Multiplexor



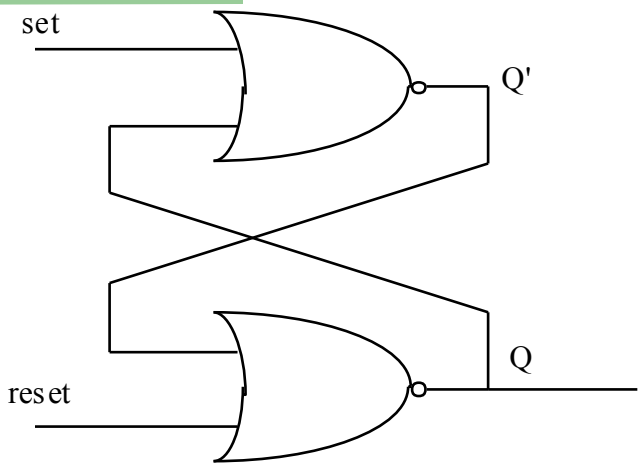
If (e_0=0) (f=a) else (f=b);

4-1 Multiplexor

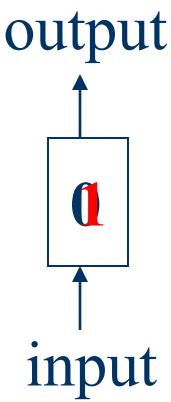


e_1	e_0	f
0	0	a
0	1	b
1	0	c
1	1	d

Memory Design: bit and Byte

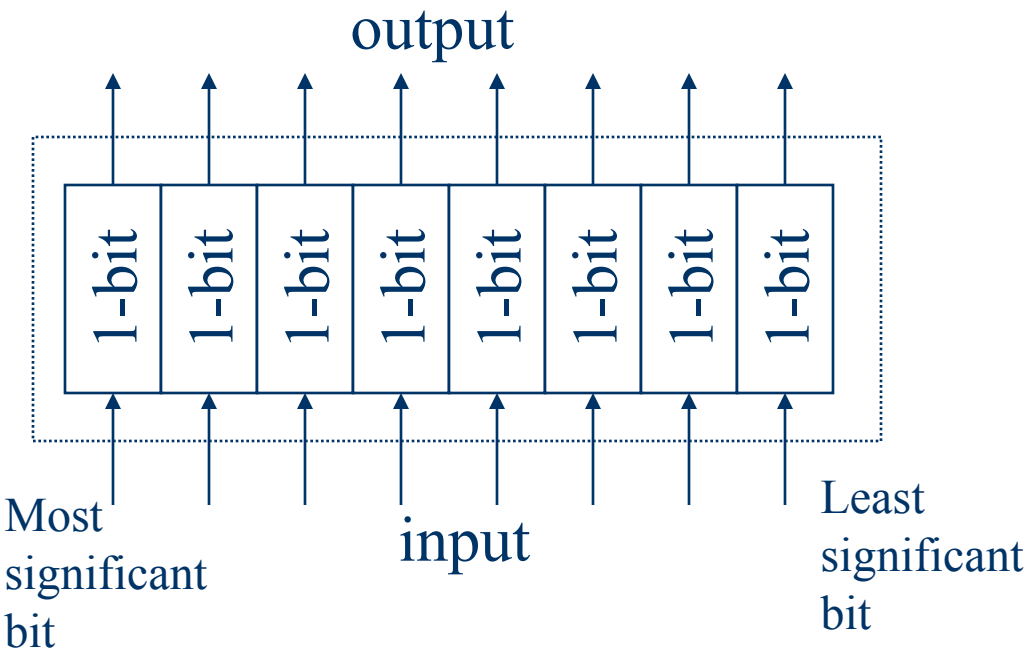


One-bit memory design



One bit can store a "1" or a "0"

8 bits = 1 Byte
Can store
a "character"
Or a short *integer*



Memory: Word, Long Word

In a 32-bit computer:
4 Bytes = 1 word and 8 bytes = 1 double word

word

Can store:
int and *float*

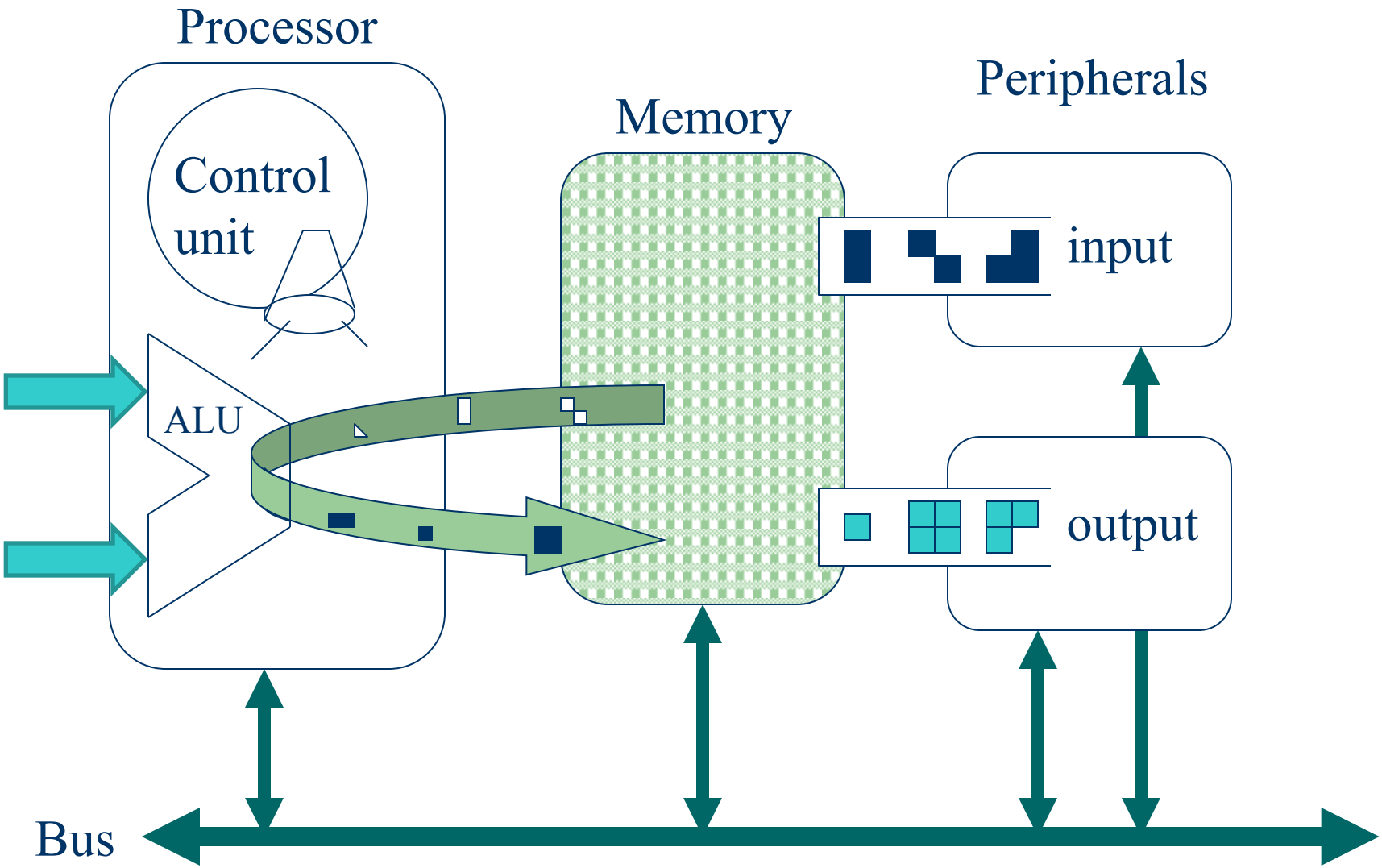
double word

Can store: a *long int* and a *double float*

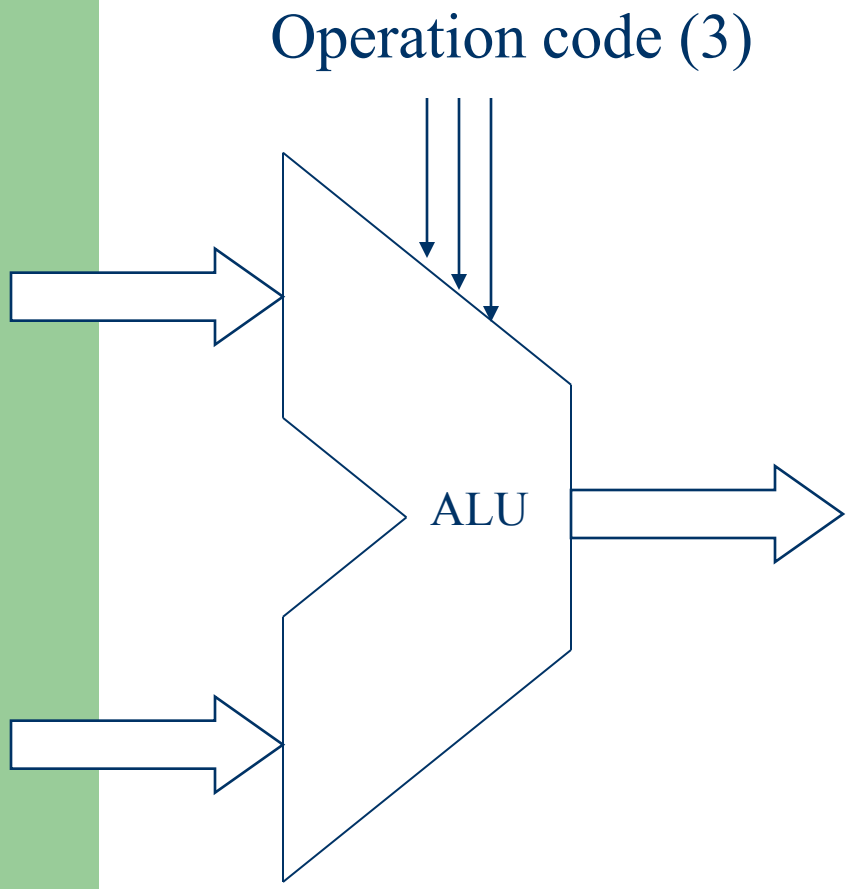
The Entire Memory, with 32-bit address space and byte-addressable

Hex address	31 30 29			2 1 0
00000000h	1 byte	1 byte	1 byte	1 byte
00000004h	1 byte	1 byte	1 byte	1 byte
00000008h	1 byte	1 byte	1 byte	1 byte
0000000Ch	1 byte	1 byte	1 byte	1 byte
00000010h	1 byte	1 byte	1 byte	1 byte
00000014h	1 byte	1 byte	1 byte	1 byte
00000018h	1 byte	1 byte	1 byte	1 byte
0000001Ch	1 byte	1 byte	1 byte	1 byte
00000020h	1 byte	1 byte	1 byte	1 byte
00000024h	1 byte	1 byte	1 byte	1 byte
00000028h	1 byte	1 byte	1 byte	1 byte
0000002Ch	1 byte	1 byte	1 byte	1 byte
00000030h	1 byte	1 byte	1 byte	1 byte
. . .	. . .	. . .	. . .	. . .
FFFFFFF0h	1 byte	1 byte	1 byte	1 byte
FFFFFFF4h	1 byte	1 byte	1 byte	1 byte
FFFFFFF8h	1 byte	1 byte	1 byte	1 byte
FFFFFFFCh	1 byte	1 byte	1 byte	1 byte

Five Component Model of a Computer -- A Conceptual Model



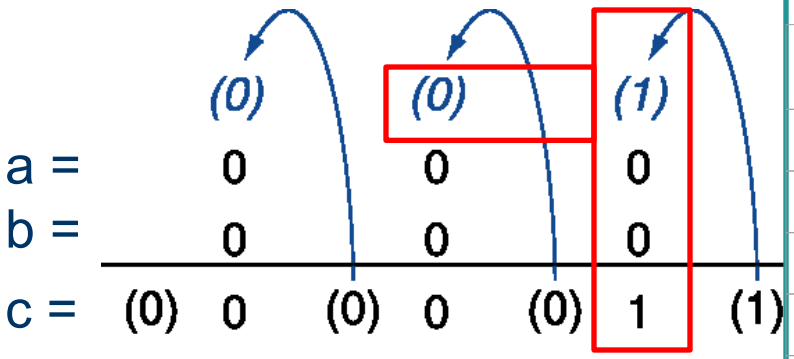
Arithmetic/Logic Unit (ALU)

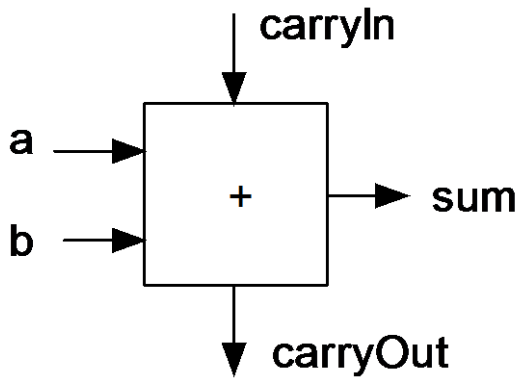


Operation code	function
0 0 0	AND
0 0 1	OR
0 1 0	ADD
1 1 0	SUB

Adder

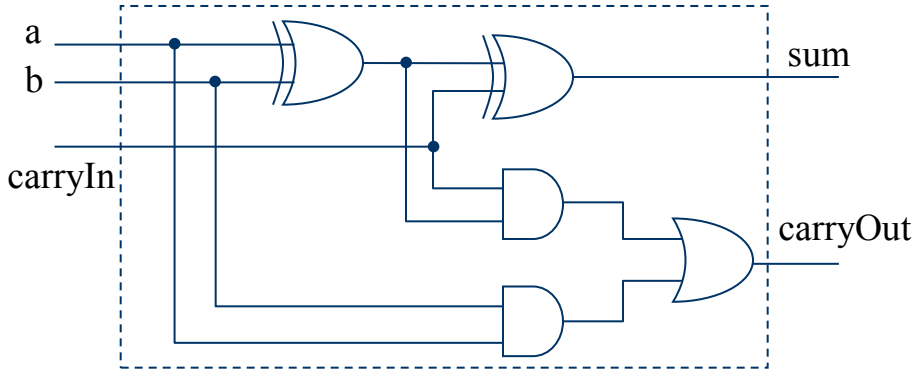
Truth Table and Design of a One-Bit Adder





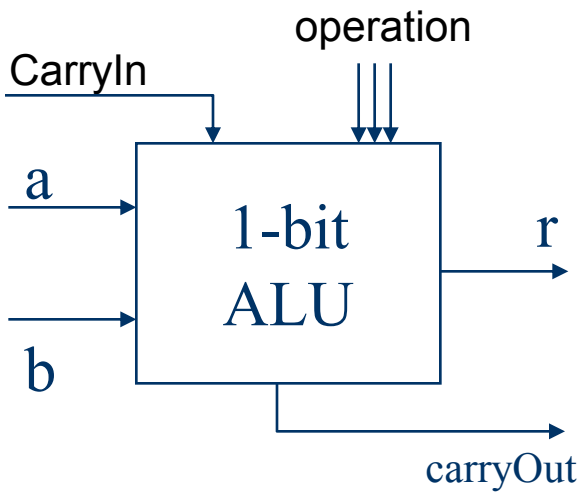
Truth Table

input			output	
a	b	carryIn	carryOut	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



One-bit adder

ALU Design: One-Bit ALU



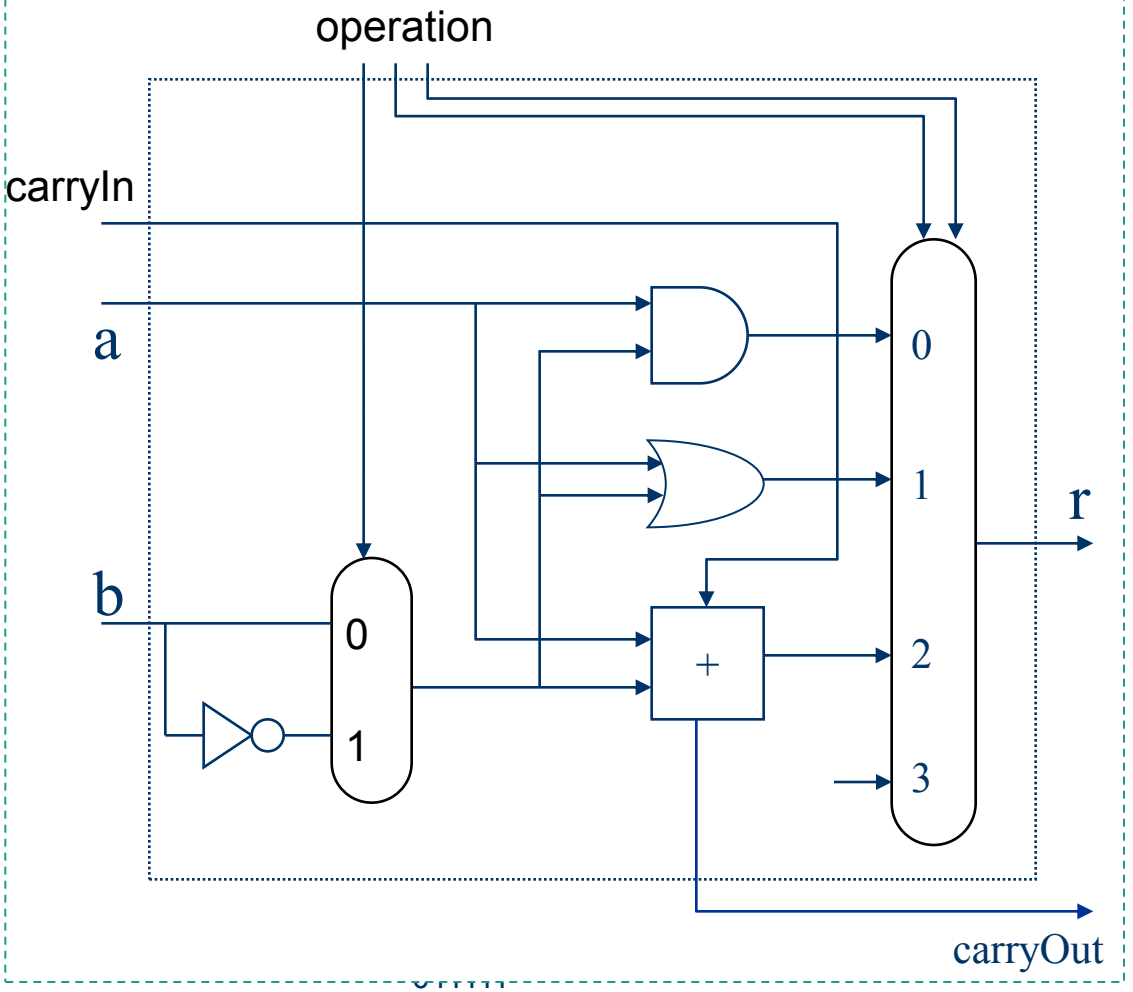
Six inputs and two outputs

Partial Truth Table

operation	function
0 0 0	AND
0 0 1	OR
0 1 0	ADD
1 1 0	SUB

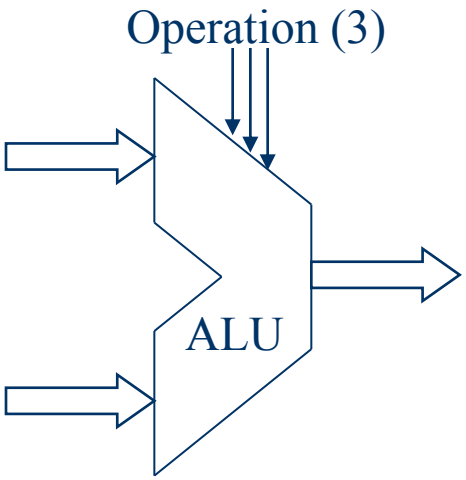
Inputs	carryOut	r
000001	d	0
000010	d	0
100000		

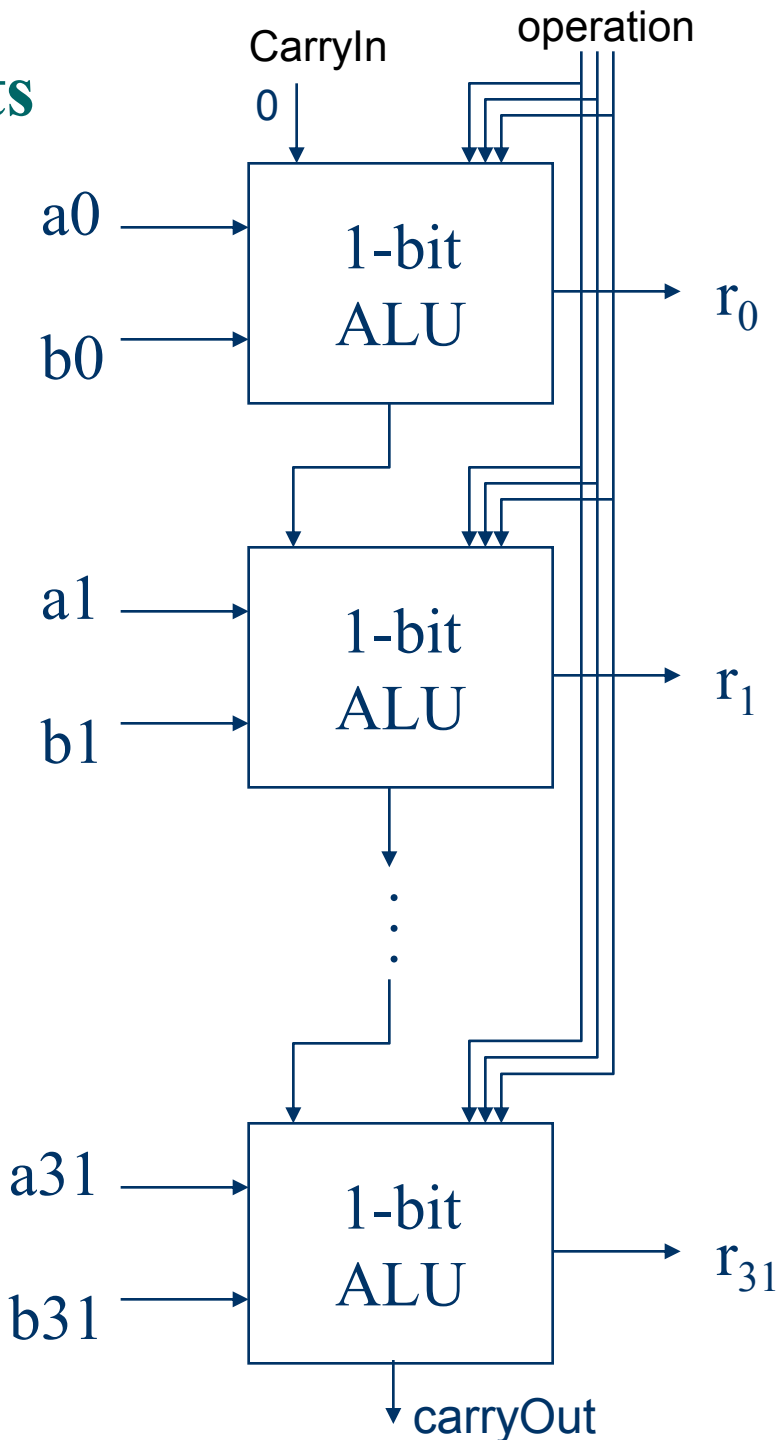
Component-based design



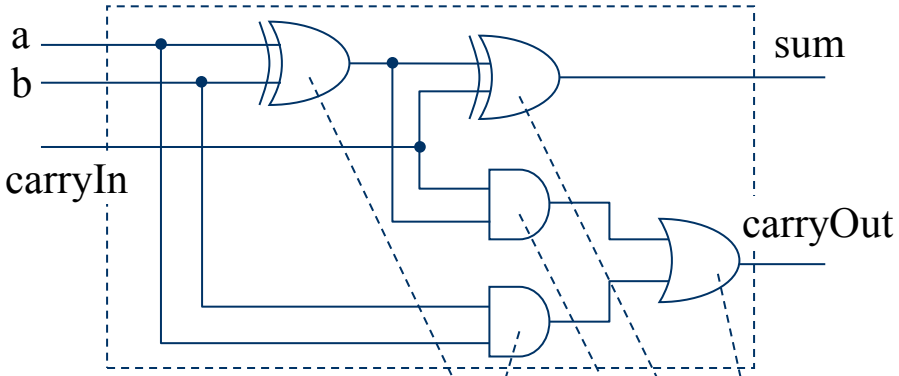
32-Bit ALU with 96 Inputs

operation	function
0 0 0	AND
0 0 1	OR
0 1 0	ADD
1 1 0	SUB





Simulation of 1-bit Adder



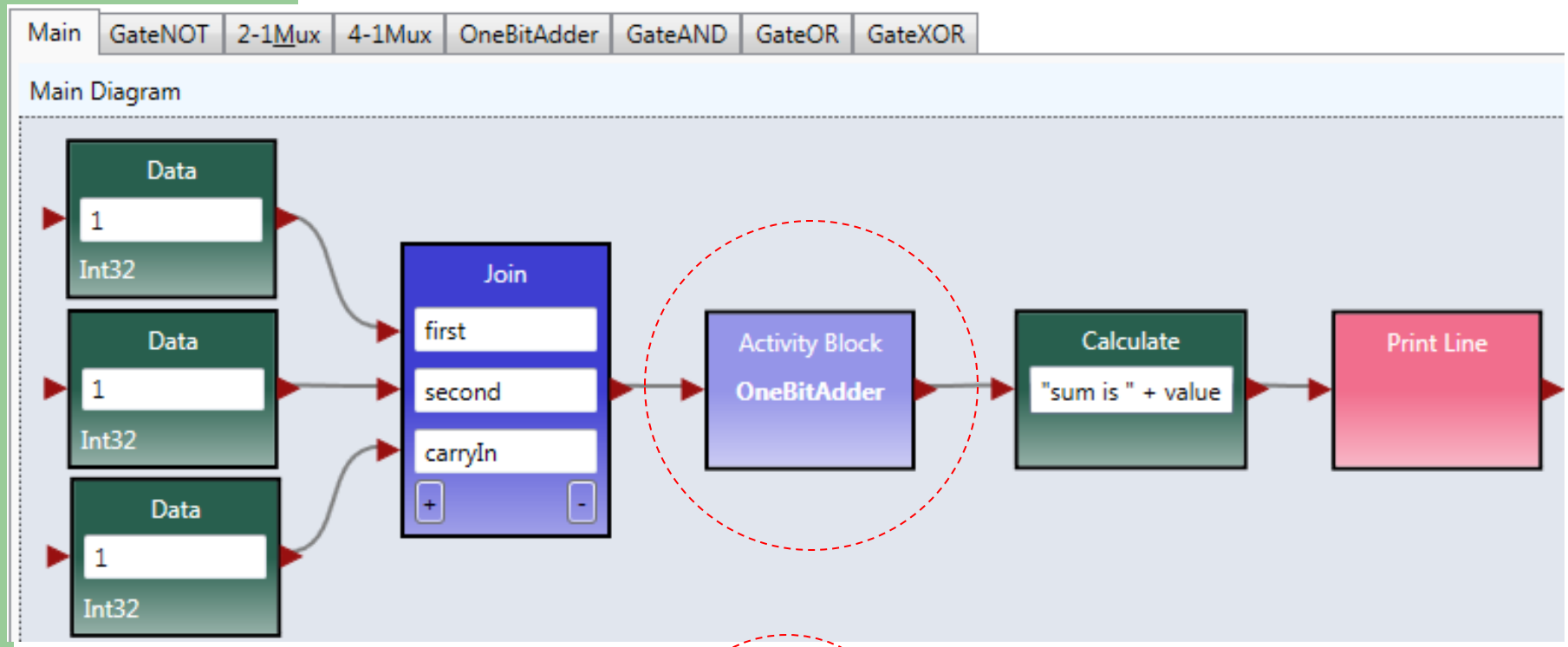
Main | GateNOT | 2-1Mux | 4-1Mux | OneBitAdder | GateAND | GateOR | GateXOR

Modify Inputs:

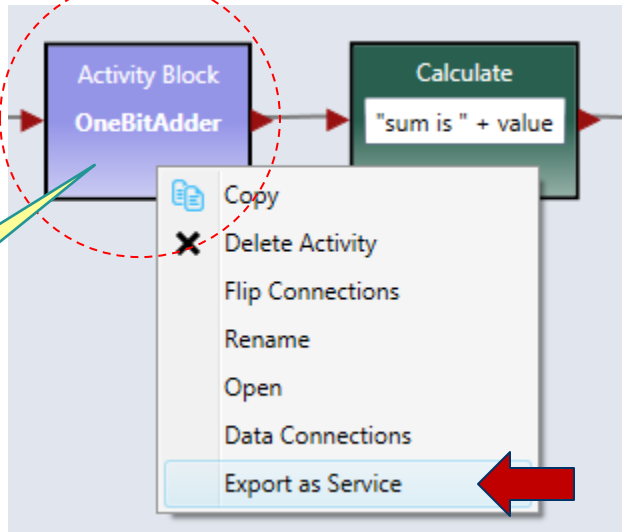
Calculate instance.a
Calculate instance.b
Calculate instance.c_in
Join first second
Join firstXORsecond carryIn
Join first second
Join output firstANDsecond
Activity Block GateXOR
Activity Block GateXOR
Activity Block GateAND
Activity Block GateAND
Activity Block GateOR
output
output
output
output
output
Calculate output "Carry out: " + value
Print Line

Define Before Using: Define the input whenever you see a warning sign

Testing the One-Bit Adder



Convert Activity to Service



Right click

Converting Decimal to Binary Pattern for Automated Test Case Generation

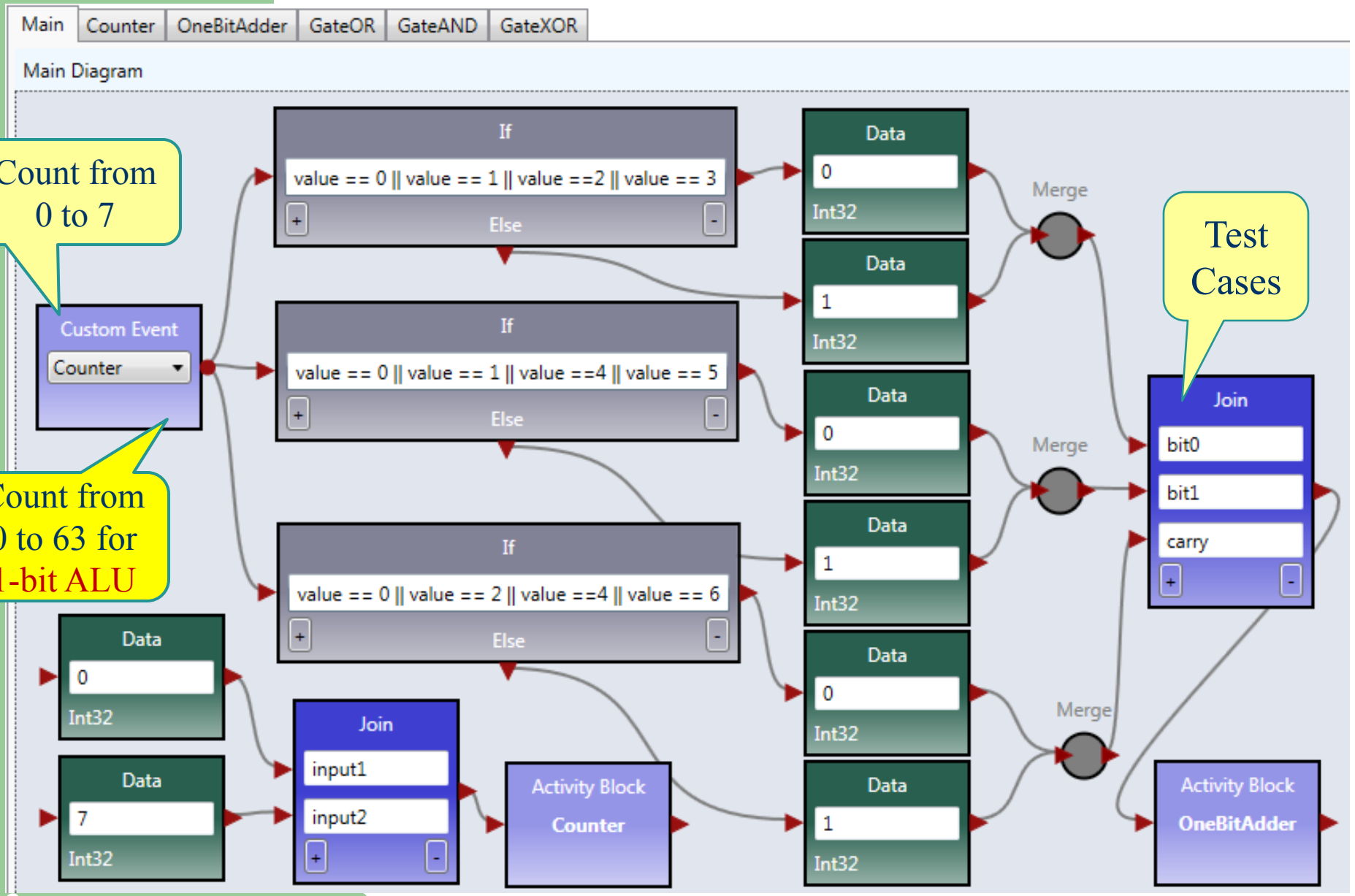
CountTo7	a	b	carryIn
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

if CountTo7 = 0, 1, 2, 3, then a = 0, else a = 1;

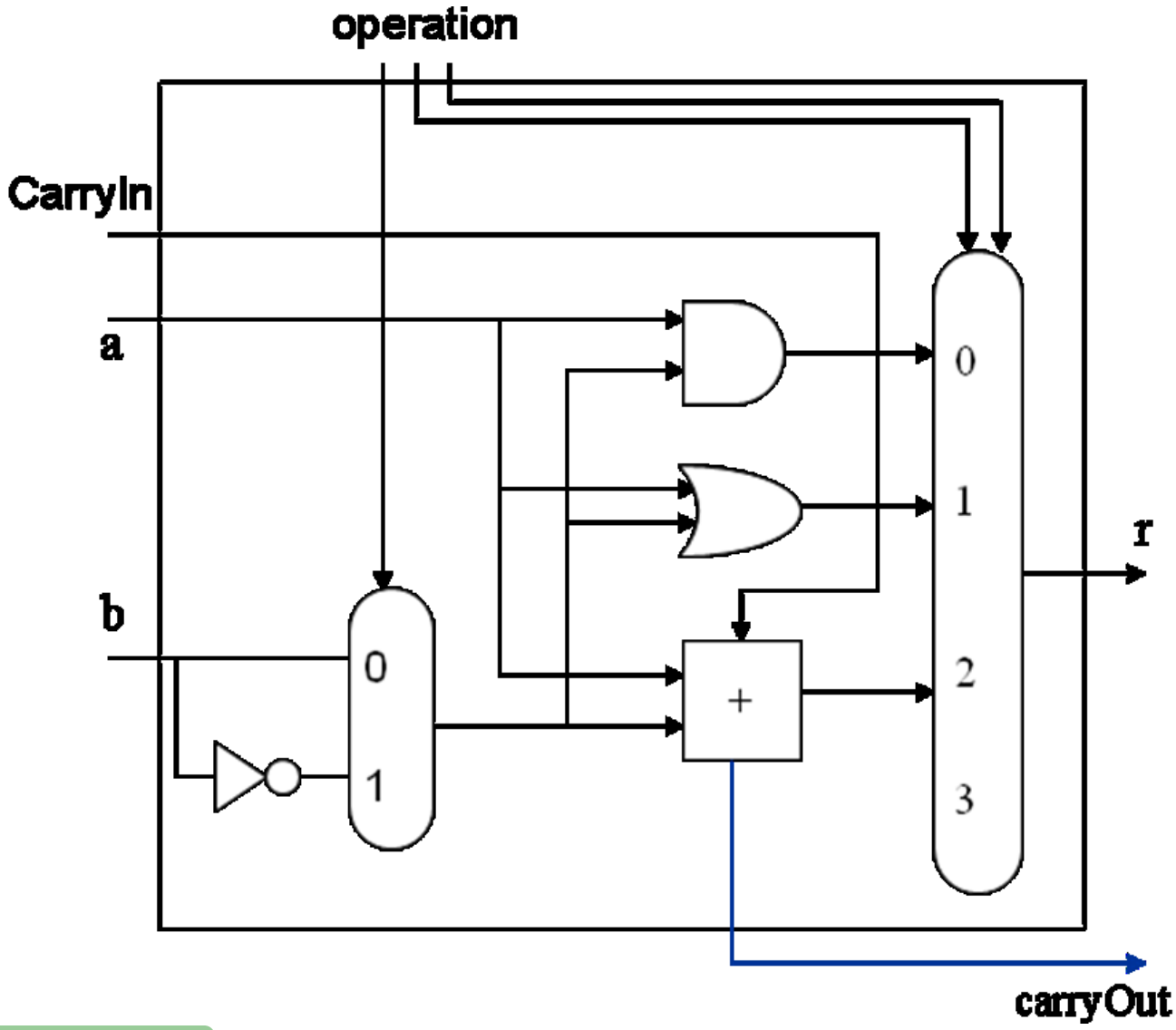
if CountTo7 = 0, 1, 4, 5, then b = 0, else b = 1;

if CountTo7 = 0, 2, 4, 6, then carryIn = 0, else carryIn = 1;

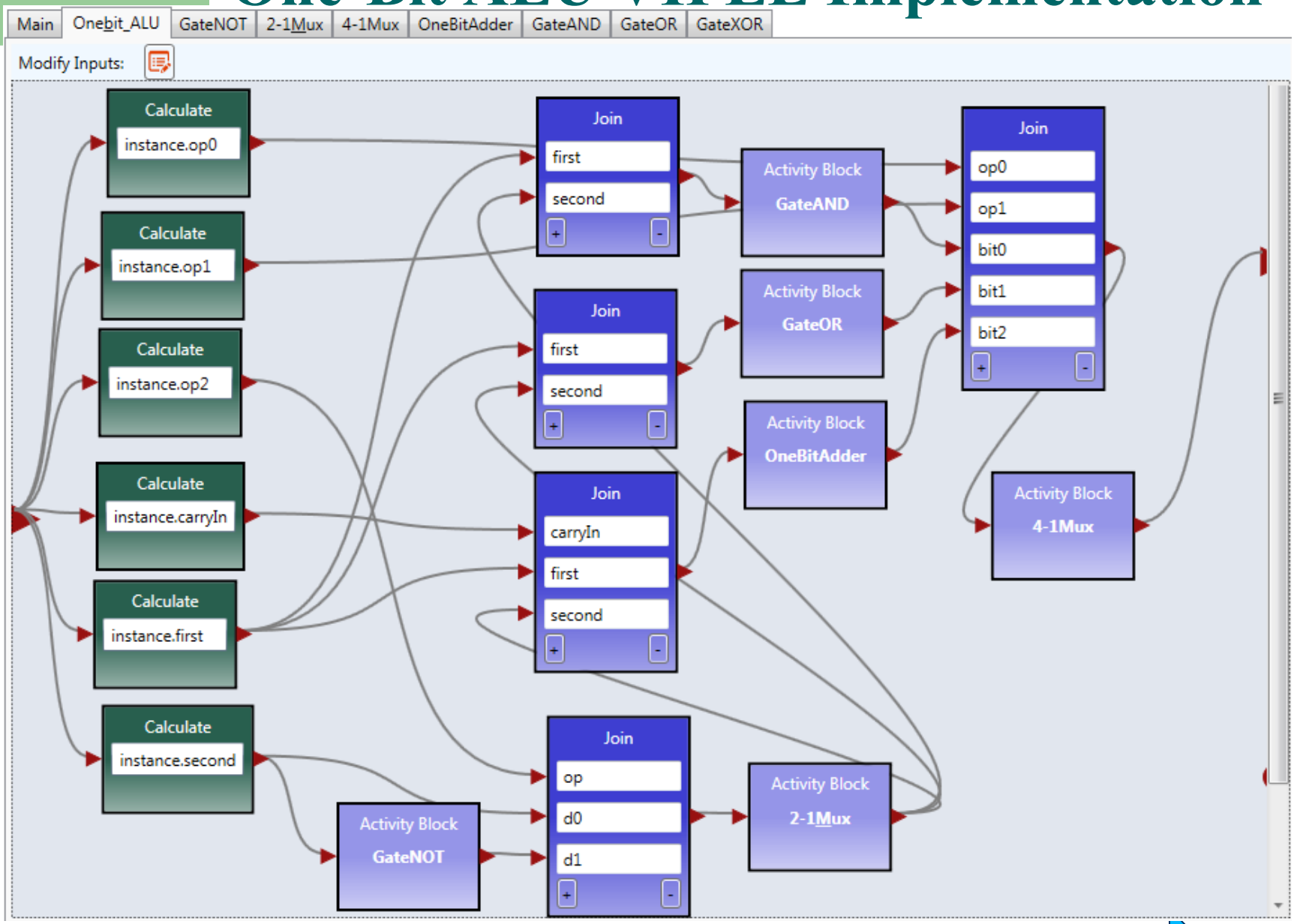
Automated Test Case Generation



One-Bit ALU

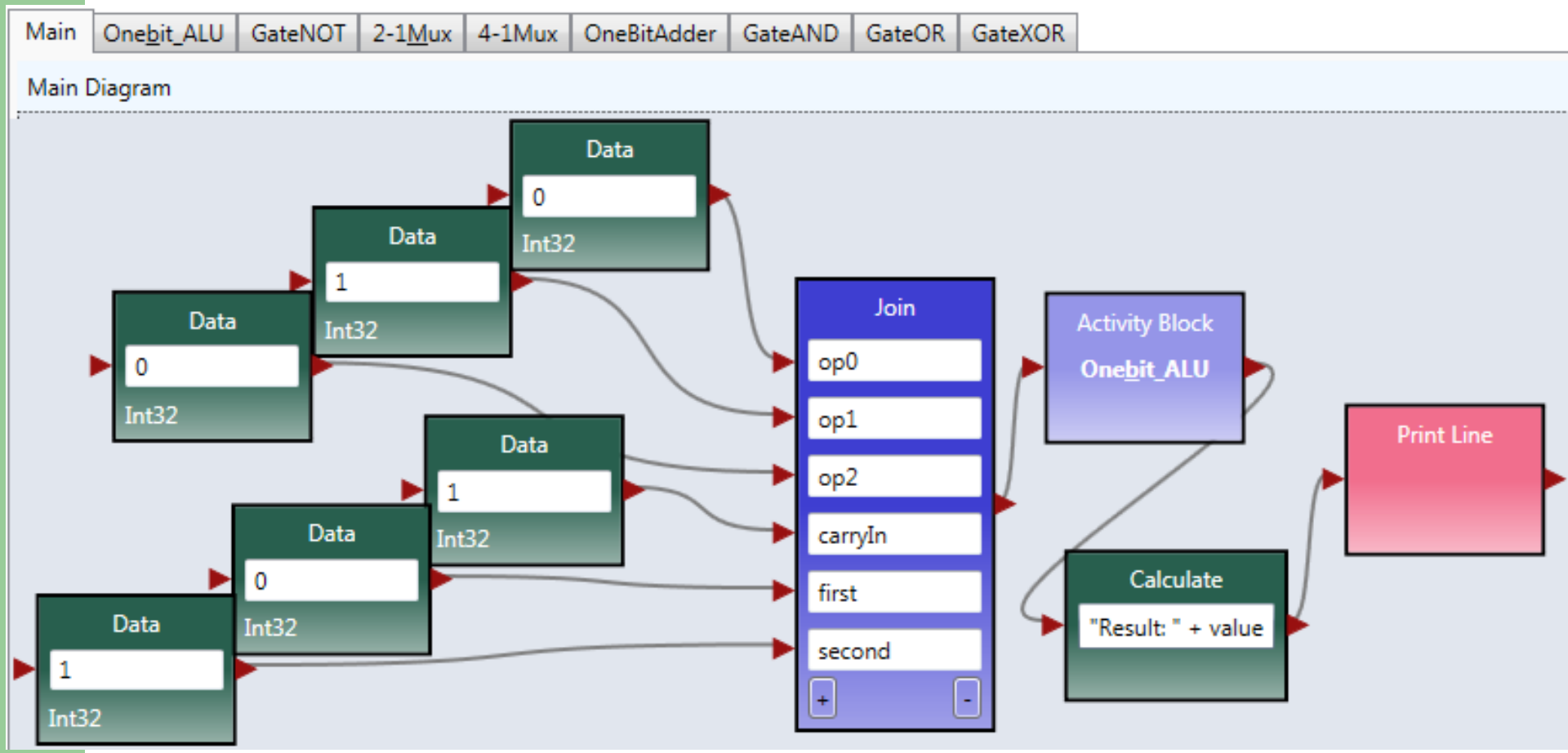


One-Bit ALU VIPLE Implementation



Define Before Using: Define the input whenever you see a warning sign

Testing the One-Bit ALU



Automated Test Case Generation