Sistemas Digitais

Unidade Lógica e Aritmética - ULA

Referência Bibliográfica:

Contemporary Logic Design – Katz & Borriello
Logic and Computer Design Fundamentals – Mano & Kime
Embedded System Design – Vahid & Givargis
Sistemas Digitais – Tocci e Widmer
Adaptações de josé artur quilici-gonzalez

Sumário

- Introdução
- Parte *(Bit Slice)* de uma ALU Genérica
- Abordagem Alternativa para Projetar ALUs
- Descrição VHDL de uma ALU
- ALU 74x181

Introdução

Uma **Unidade Lógica e Aritmética**, ou **ALU**, ou **ULA**, é uma rede combinacional, que implementa uma função de suas entradas com base em **operações lógicas ou aritméticas**

ALUs constituem o elemento central dos computadores e da maioria dos sistemas digitais

Nesta aula, vamos aprender como projetar partes desse sistema

Parte (*Bit Slice*) de uma ALU Genérica

Operações Lógicas e Aritméticas

M = 0, Logical Bitwise Operations

S1	S0	Function	Comment
0	0	Fi = Ai	Input Ai transferred to output
0	1	Fi = not Ai	Complement of Ai transferred to output
1	0	Fi = Ai xor Bi	Compute XOR of Ai, Bi
1	1	Fi = Ai xnor Bi	Compute XNOR of Ai, Bi

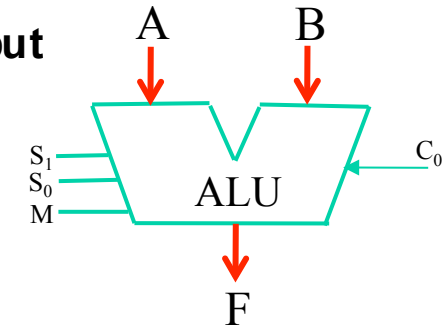
M = 1, C0 = 0, Arithmetic Operations

0	0	F = A	Input A passed to output
0	1	F = not A	Complement of A passed to output
1	0	F = A plus B	Sum of A and B
1	1	F = (not A) plus B	Sum of B and complement of A

M = 1, C0 = 1, Arithmetic Operations

0	0	F = A plus 1	Increment A
0	1	F = (not A) plus 1	Twos complement of A
1	0	F = A plus B plus 1	Increment sum of A and B
1	1	F = (not A) plus B plus 1	B minus A

M = 0 – Modo Lógico
M = 1 – Modo Aritmético



Nem todas operações parecem úteis, mas resultado da lógica interna

Minimização Tradicional em Dois Níveis

Na abordagem tradicional, a **Tabela da Verdade** da ALU é colocada num programa chamado SIS e minimizada com o comando *espresso*

```
.i 6  (6 entradas)
.o 2  (2 saídas)
.ilb m s1 s0 ci ai bi
.ob fi co
.p 23
111101 10
110111 10
1-0100 10
1-1110 10
10010- 10
10111- 10
-10001 10
010-01 10
-11011 10
011-11 10
--1000 10
0-1-00 10
--0010 10
0-0-10 10
-0100- 10
001-0- 10
-0001- 10
000-1- 10
-1-1-1 01
--1-01 01
--0-11 01
--110- 01
--011- 01
.e
```

O resultado para este exemplo foi 23 SOP (Soma de Produtos) (de dois níveis)

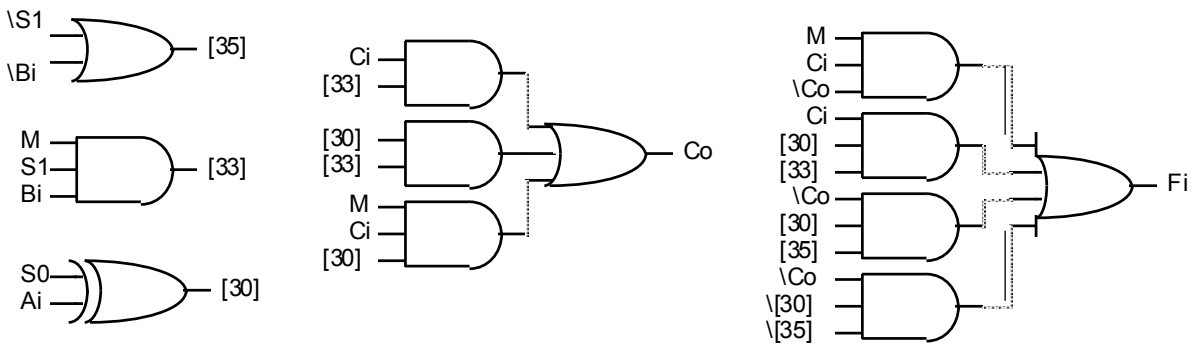
M	S1	S0	Ci	Ai	Bi	Fi	Ci+1
0	0	0	X	0	X	0	X
			X	1	X	1	X
	0	1	X	0	X	1	X
			X	1	X	0	X
	1	0	X	0	0	0	X
			X	0	1	1	X
			X	1	0	1	X
			X	1	1	0	X
	1	1	X	0	0	1	X
			X	0	1	0	X
			X	1	0	0	X
			X	1	1	1	X
1	0	0	0	0	X	0	X
			0	1	X	1	X
	0	1	0	0	X	1	X
			0	1	X	0	X
	1	0	0	0	0	0	0
			0	0	1	1	0
			0	1	0	1	0
			0	1	1	0	1
	1	1	0	0	0	1	0
			0	0	1	0	1
			0	1	0	0	0
			0	1	1	1	0
1	0	0	1	0	X	1	0
			1	1	X	0	1
	0	1	1	0	X	0	1
			1	1	X	1	0
	1	0	1	0	0	1	0
			1	0	1	0	1
			1	1	0	0	1
			1	1	1	1	1
	1	1	1	0	0	0	1
			1	0	1	1	1
			1	1	0	1	0
			1	1	1	0	1

SIS – Ferramenta para Síntese Lógica (U. de Berkeley)
http://embedded.eecs.berkeley.edu/Alumni/rajeev/cs252/report/main/node11.html

Contemporary Logic Design – Katz e Borriello

Minimização Multinível

```
.model alu.espresso
.inputs m s1 s0 ci ai bi
.outputs fi co
.names m ci co [30] [33] [35] fi
110--- 1
-1-11- 1
--01-1 1
--00-0 1
.names m ci [30] [33] co
-1-1 1
--11 1
111- 1
.names s0 ai [30]
01 1
10 1
.names m s1 bi [33]
111 1
.names s1 bi [35]
0- 1
-0 1
.end
```

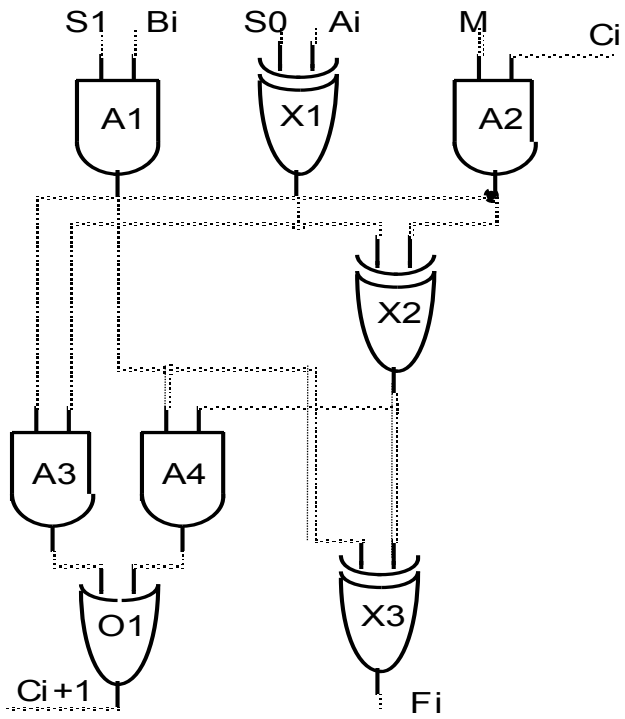


12 Portas Lógicas

A implementação multinível produz uma considerável redução no número de portas lógicas utilizadas

Minimização Multinível

Implementação multinível minuciosamente reelaborada para máxima simplificação



8 Portas Lógicas (porém 3 XOR)

S1 = 0 bloqueia Bi
Isto ocorre para operações que envolvem apenas Ai

O mesmo se aplica para Ci quando M = 0

A Adição (Ai xor Bi) ocorre quando M = 1

Ci, Bi passam para as XORs X2, X3

S0 = 0, X1 passa A

S0 = 1, X1 passa A´

Modo Aritmético (M=1):

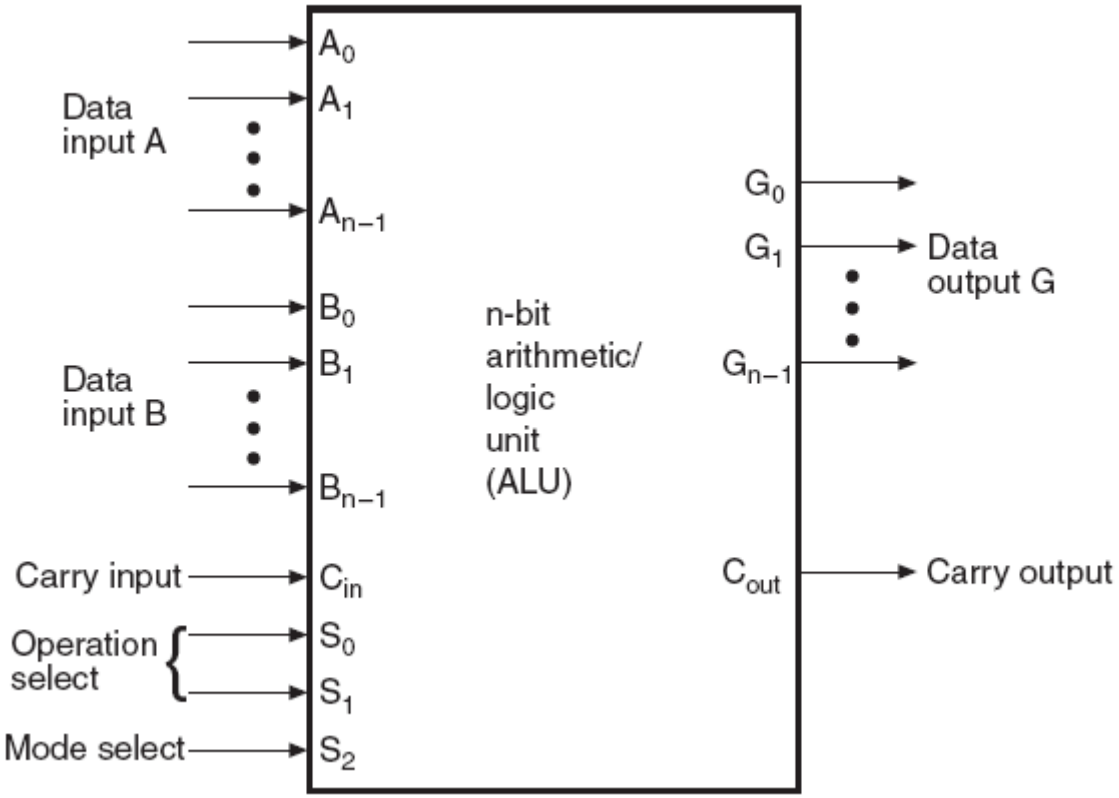
O Vai-Um na porta OR (O1) é Ai Ci + Bi (Ai xor Ci)

Modo Lógico (M=0):

XORs em cascata formam a saída a partir de Ai e Bi

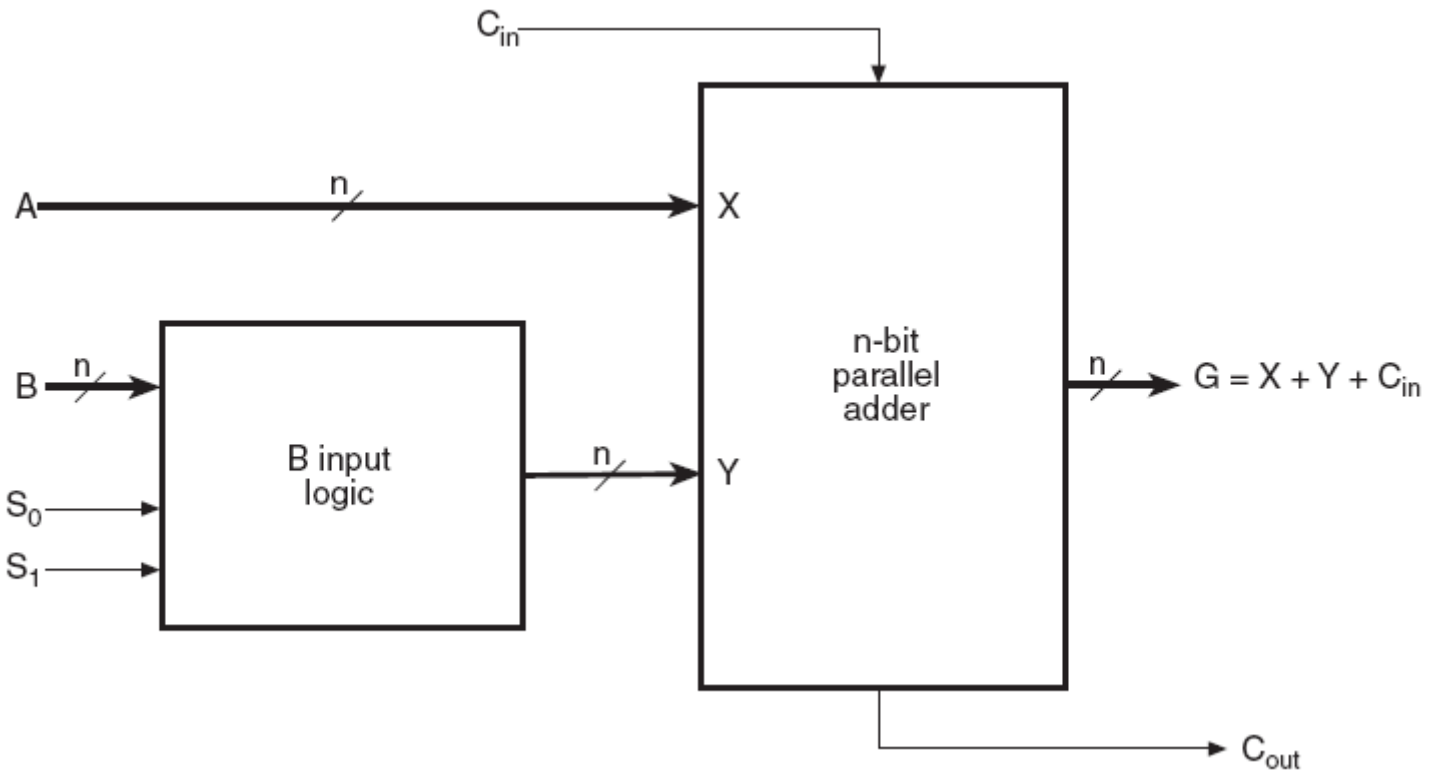
Contemporary Logic Design – Katz e Borriello

ALU Genérica de *n-bits*



O componente básico de um circuito aritmético é o somador paralelo, que é construído por vários somadores completos de 1 *bit* ligados em cascata

Diagrama de Blocos do Circuito Aritmético



Nesta configuração, as *n* entradas *B* do somador paralelo são controladas pelas linhas de seleção S_1 e S_0

Desta forma, a entrada *Y* do somador pode receber além das entradas *B*, o complemento destas entradas, um conjunto de *0s* ou *1s etc.*

A entrada C_{in} (Vem_Um) é colocada na posição *Least-Significant-Bit*, enquanto que a saída C_{out} (Vai_Um), na posição *Most-Significant-Bit*

Tabela de Funções do Circuito Aritmético

Select		Input	$G = A + Y + C_{in}$	
S_1	S_0	Y	$C_{in} = 0$	$C_{in} = 1$
0	0	all 0's	$G = A$ (transfer)	$G = A + 1$ (increment)
0	1	B	$G = A + B$ (add)	$G = A + B + 1$
1	0	$\overline{B}$	$G = A + \overline{B}$	$G = A + \overline{B} + 1$ (subtract)
1	1	all 1's	$G = A - 1$ (decrement)	$G = A$ (transfer)

A tabela acima mostra as possíveis operações aritméticas obtidas com a ajuda das linhas de seleção S_1 e S_0

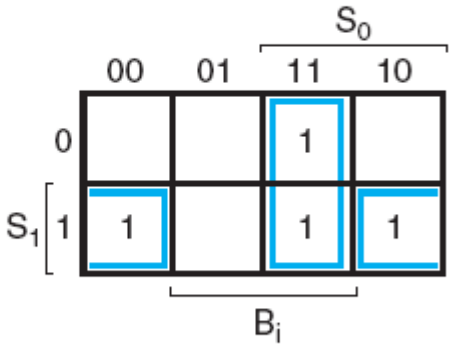
Se as entradas provenientes de B forem ignoradas, a entrada Y recebe *0s* e a saída G simplesmente reproduz o valor das entradas A, desde que $C_{in} = 0$ $(G = A + 0 + C_{in})$

Se o complemento de B for aplicado em Y, e $C_{in} = 1$, obtém-se a subtração aritmética $G = A - B$, ou seja, A é somado com o complemento de *2* de B $(G = A + B' + 1)$

Tabela da Verdade do Circuito Aritmético

Inputs			Output	
S_1	S_0	B_i	Y_i	
0	0	0	0	$Y_i = 0$
0	0	1	0	
0	1	0	0	$Y_i = B_i$
0	1	1	1	
1	0	0	1	$Y_i = \overline{B}_i$
1	0	1	0	
1	1	0	1	$Y_i = 1$
1	1	1	1	

(a) Truth table

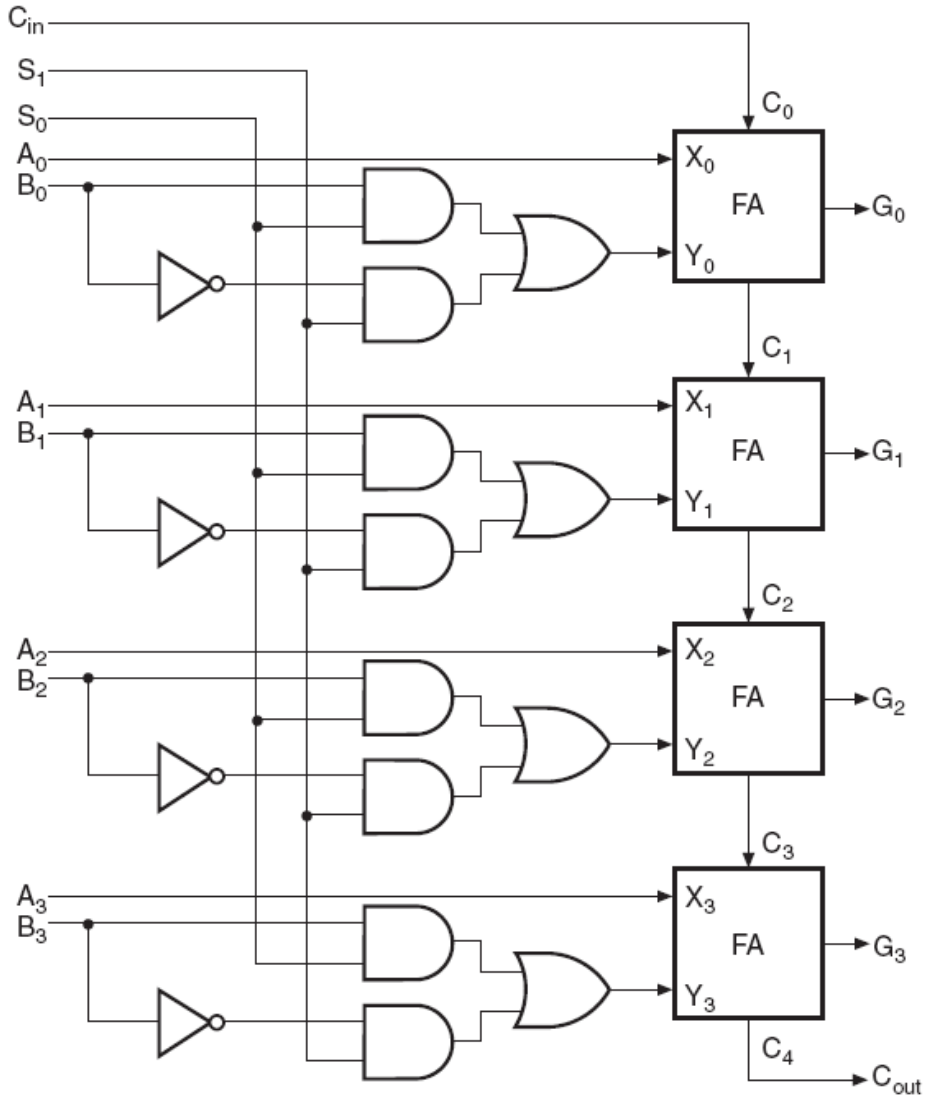


(b) Map Simplification:
$Y_i = B_i S_0 + \overline{B}_i S_1$

A lógica de entrada de *B* (*B input logic*) pode ser obtida com *n MUXes 4x1* (*00, 01, 10, 11*) ou, com menos portas, através de uma simplificação usando Mapas de Karnaugh

Colocando-se numa Tabela da Verdade as entradas S_1, S_0 e B_i (um *bit* de *B* apenas), tendo como saída Y_i, a equação booleana simplificada de Y_i pode ser facilmente obtida como uma *SOP* (Soma de Produtos) de suas entradas

Diagrama Lógico do Circuito Aritmético

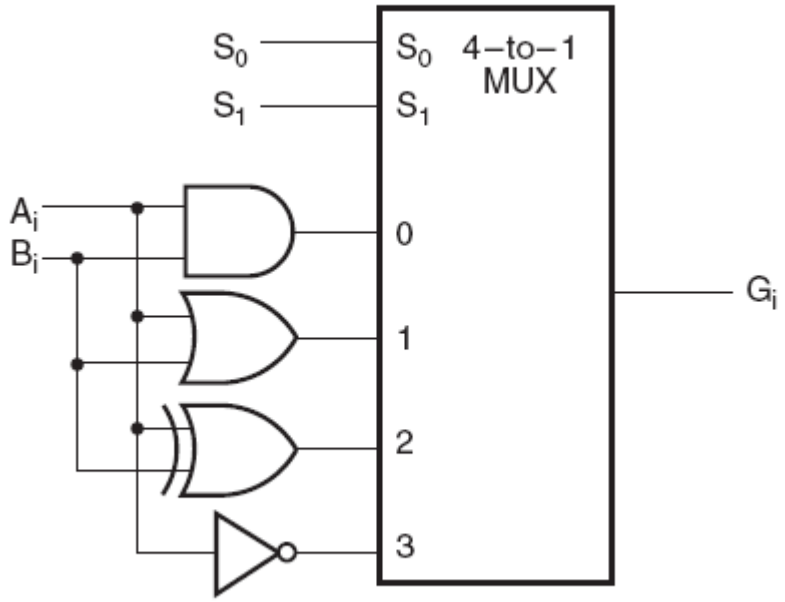


A figura mostra o diagrama lógico de um circuito aritmético para n = 4

Os quatro somadores completos (*FA – Full-Adder*) constituem o somador paralelo

Logic and Computer Design Fundamentals – Mano e Kime

Um Estágio do Circuito Lógico



(a) Logic Diagram

S_1	S_0	Output	Operation
0	0	$G = A \wedge B$	AND
0	1	$G = A \vee B$	OR
1	0	$G = A \oplus B$	XOR
1	1	$G = \overline{A}$	NOT

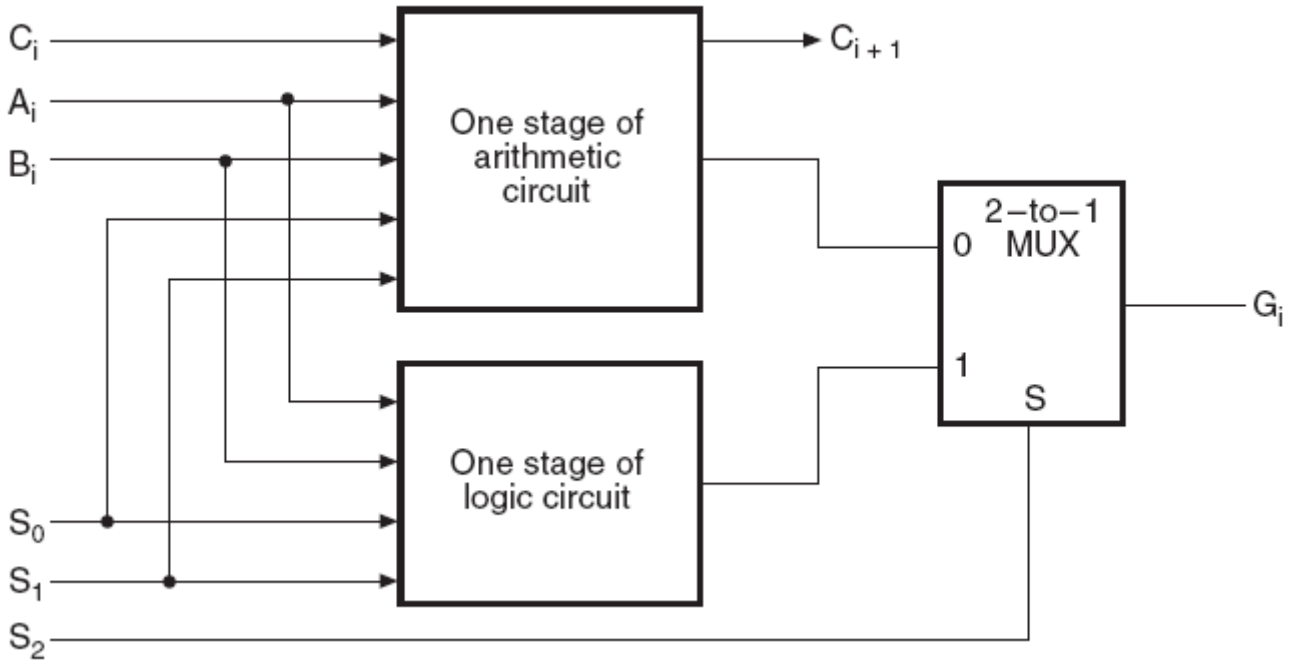
(b) Function Table

Microoperações lógicas manipulam os *bits* dos operandos considerando cada *bit* em um registrador como uma variável binária, desta forma realizando operações *bit* a *bit*

A partir das quatro operações lógicas disponíveis nas entradas do *MUX 4x1*, é possível obter outras operações lógicas

Para um circuito lógico de *n bits*, este diagrama precisa ser repetido *n* vezes

Um Estágio da ALU



A *ALU* é construída combinando-se o circuito aritmético com o lógico, sendo as entradas de seleção S_1 e S_0 comum aos dois circuitos, com S_2 decidindo sobre o modo de operação, aritmético ($S_2 = 0$) ou lógico ($S_2 = 1$)

Para uma *ALU* de *n bits*, este diagrama precisa ser repetido *n* vezes

Tabela de Funções da ALU

Operation Select					
S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	$G = A$	Transfer A
0	0	0	1	$G = A + 1$	Increment A
0	0	1	0	$G = A + B$	Addition
0	0	1	1	$G = A + \overline{B} + 1$	Add with carry input of 1
0	1	0	0	$G = A + \overline{B}$	A plus 1's complement of B
0	1	0	1	$G = A + B + 1$	Subtraction
0	1	1	0	$G = A - 1$	Decrement A
0	1	1	1	$G = A$	Transfer A
1	0	0	X	$G = A \wedge B$	AND
1	0	1	X	$G = A \vee B$	OR
1	1	0	X	$G = \overline{A} \oplus B$	XOR
1	1	1	X	$G = \overline{A}$	NOT (1's complement)

A ALU do exemplo oferece oito operações aritméticas e quatro lógicas, selecionadas através de S2, S1, S0 e Cin (durante as operações lógicas, Cin não tem nenhum efeito sobre o resultado, podendo seu valor ser 0 ou 1)

VHDL Comportamental de uma ALU

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ALU is
    port ( A : in  STD_LOGIC_VECTOR (1 downto 0);
           B : in  STD_LOGIC_VECTOR (1 downto 0);
           S : in  STD_LOGIC_VECTOR (2 downto 0);
           F : out  STD_LOGIC_VECTOR (1 downto 0));
end ALU;

architecture Behavioral of ALU is
begin
process(A, B, S)
begin
    case S is
        when "000" => F <= "00";
        when "001" => F <= (NOT A) + B + "01";
        when "010" => F <= (NOT B) + A + "01";
        when "011" => F <= A + B;
        when "100" => F <= A XOR B;
        when "101" => F <= A OR B;
        when "110" => F <= A AND B;
        when "111" => F <= "01";
        when others => null;
     end case;
end process;
end Behavioral;
```

S2	S1	S0	F
0	0	0	"00"
0	0	1	(NOT A)+B+"01"
0	1	0	(NOT B)+A+"01"
0	1	1	A+B
1	0	0	A XOR B
1	0	1	A OR B
1	1	0	A AND B
1	1	1	"01"

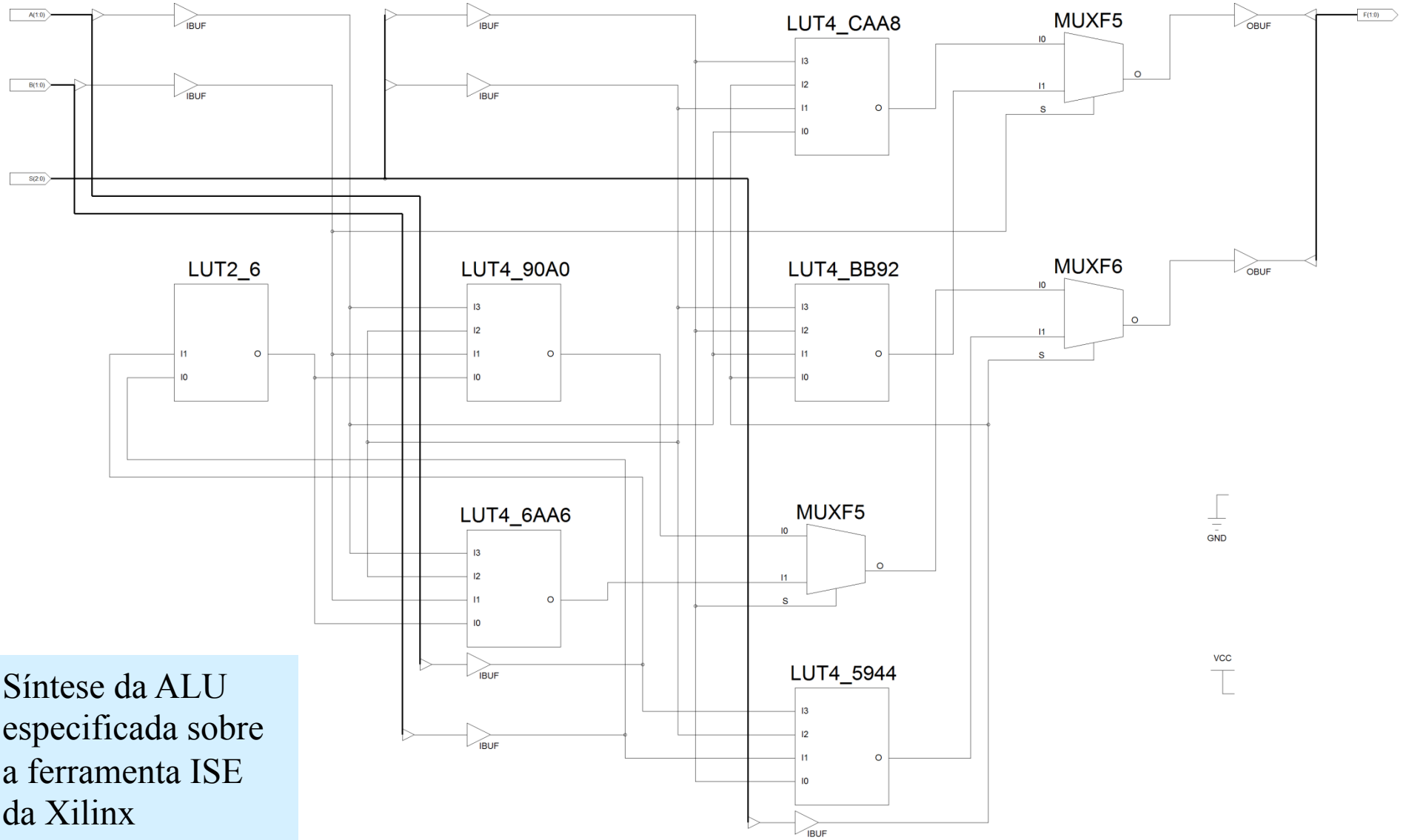
Uma ALU pode facilmente ser especificada no nível comportamental com a cláusula

case expressão **is**
    when opções => comando
end case

ou

with expressão **select**
nome <= comando **when** opções

Diagrama Esquemático Xilinx da ALU



Síntese da ALU especificada sobre a ferramenta ISE da Xilinx

Exemplo Prático: ALU 74HC181

4-bit arithmetic logic unit 74HC/HCT181

FUNCTION TABLES

MODE SELECT INPUTS				ACTIVE HIGH INPUTS AND OUTPUTS	
S_3	S_2	S_1	S_0	LOGIC (M=H)	ARITHMETIC(2) (M=L; C_n=H)
L	L	L	L	$\overline{A}$	A
L	L	L	H	$\overline{A+B}$	A + B
L	L	H	L	$\overline{A}B$	A + $\overline{B}$
L	L	H	H	logical 0	minus 1
L	H	L	L	$\overline{AB}$	A plus A$\overline{B}$
L	H	L	H	$\overline{B}$	(A + B) plus A$\overline{B}$
L	H	H	L	A ⊕ B	A minus B minus 1
L	H	H	H	A$\overline{B}$	A$\overline{B}$ minus 1
H	L	L	L	$\overline{A}$ + B	A plus AB
H	L	L	H	$\overline{A \oplus B}$	A plus B
H	L	H	L	B	(A + $\overline{B}$) plus AB
H	L	H	H	AB	AB minus 1
H	H	L	L	logical 1	A plus A(1)
H	H	L	H	A + $\overline{B}$	(A + B) plus A
H	H	H	L	A + B	(A + $\overline{B}$) plus A
H	H	H	H	A	A minus 1

Notes to the function tables

1. Each bit is shifted to the next more significant position.
2. Arithmetic operations expressed in 2s complement notation.

H = HIGH voltage level
L = LOW voltage level

MODE SELECT INPUTS				ACTIVE LOW INPUTS AND OUTPUTS	
S_3	S_2	S_1	S_0	LOGIC (M=H)	ARITHMETIC(2) (M=L; C_n=L)
L	L	L	L	$\overline{A}$	A minus 1
L	L	L	H	$\overline{AB}$	AB minus 1
L	L	H	L	$\overline{A}$ + B	A$\overline{B}$ minus 1
L	L	H	H	logical 1	minus 1
L	H	L	L	$\overline{A+B}$	A plus (A + $\overline{B}$)
L	H	L	H	$\overline{B}$	AB plus (A + $\overline{B}$)
L	H	H	L	$\overline{A \oplus B}$	A minus B minus 1
L	H	H	H	A + $\overline{B}$	A + $\overline{B}$
H	L	L	L	$\overline{AB}$	A plus (A + B)
H	L	L	H	A ⊕ B	A plus B
H	L	H	L	B	A$\overline{B}$ plus (A + B)
H	L	H	H	A + B	A + B
H	H	L	L	logical 0	A plus A(1)
H	H	L	H	A$\overline{B}$	AB plus A
H	H	H	L	AB	A$\overline{B}$ plus A
H	H	H	H	A	A

Notes to the function tables

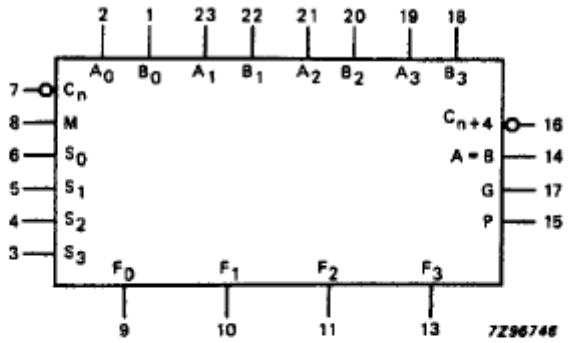
1. Each bit is shifted to the next more significant position.
2. Arithmetic operations expressed in 2s complement notation.

H = HIGH voltage level
L = LOW voltage level

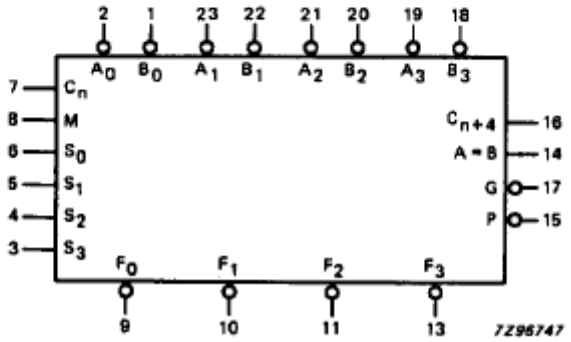
As entradas e saídas da 74x181 são normalmente ativas em *LOW*, mas esta ALU pode ser utilizada com suas entradas e saídas ativas em *HIGH* (fazendo operações complementares)

Operandos ativos em HIGH e LOW na 74HC181

Active HIGH operands



Active LOW operands



A pinagem da 74x181 tanto para o operação com suas entradas e saídas ativas em *HIGH* ou *LOW* é a mesma, mas os conjuntos de operações lógicas e aritméticas são distintos

As entradas (A_3-A_0 e B_3-B_0) e as saídas (F_3-F_0) são para quatro *bits*.

Quando $M=1$, são selecionadas através de S_3-S_0 as operações lógicas, cujos resultados dependem apenas das entradas A_i e B_i, sendo C_n (*Vem_Um*) e C_{n+4} (*Vai_Um*) ignorados

Quando $M=0$, são selecionadas as operações aritméticas e o *Vem_Um* se propaga entre os estágios

Diagrama Lógico da ALU 74HC181