Universidade Federal de Santa Catarina
Centro Tecnológico – CTC
Departamento de Engenharia Elétrica

"EEL7020 – Sistemas Digitais"

Prof. Eduardo Augusto Bezerra

Eduardo.Bezerra@eel.ufsc.br

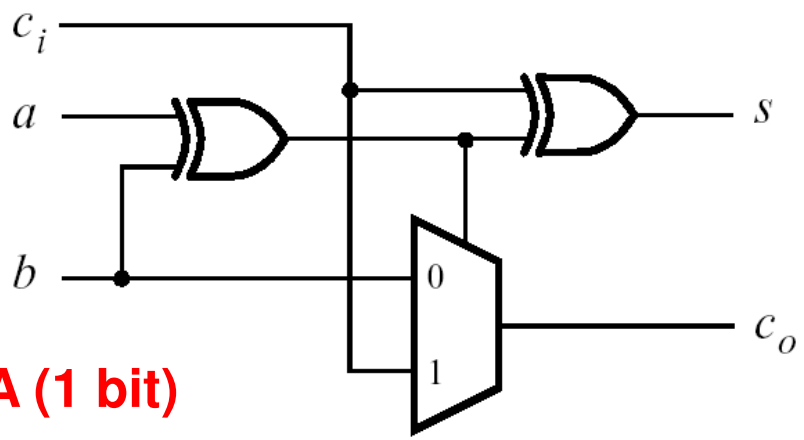
Florianópolis, agosto de 2011.

Sistemas Digitais

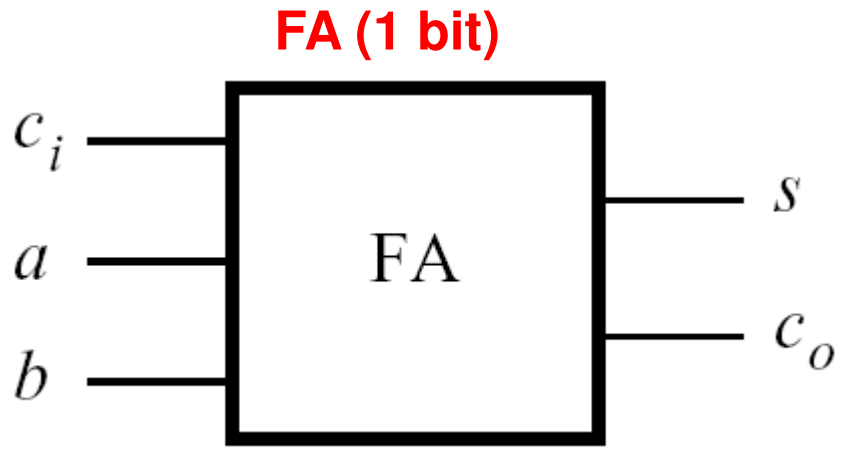
Projeto de somador de 4 bits
"construindo uma calculadora"

Arquivo: lab2_VHDL.pdf
parte III

Exemplo: somador de 4 bits

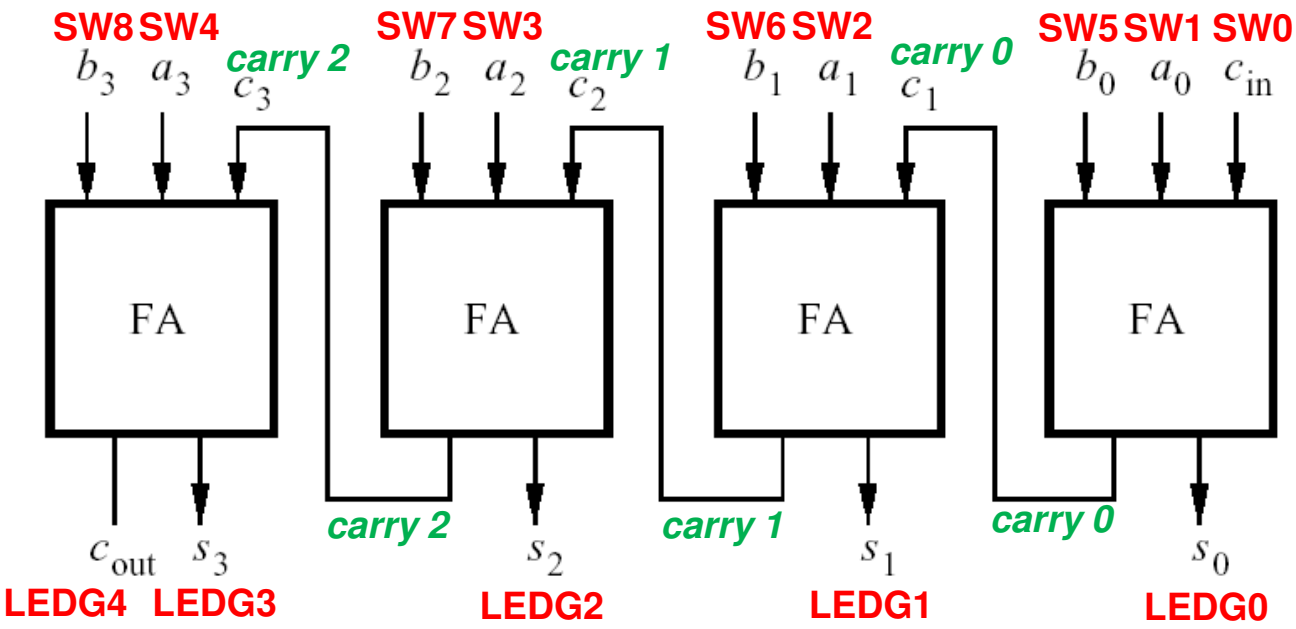


FA (1 bit)



b	a	c_i	c_o	s
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

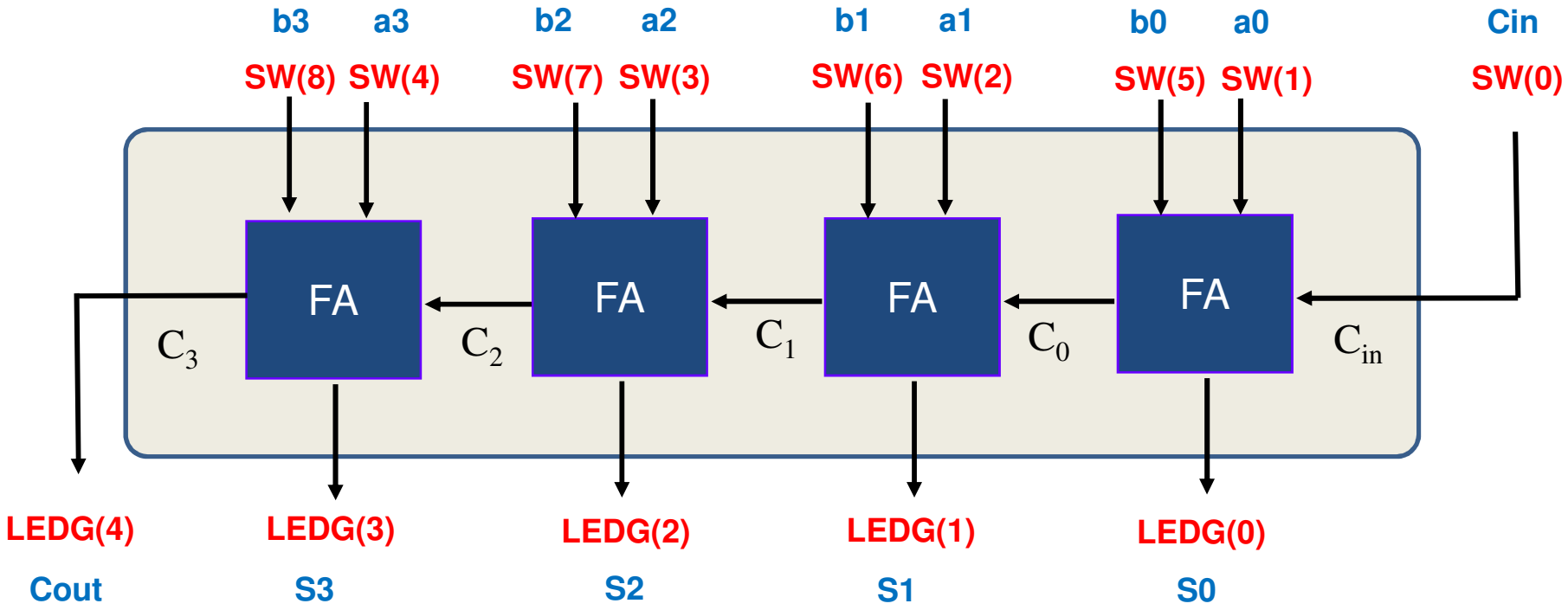
Ripple-carry adder de 4 bits, construído com 4 FAs



Exemplo: somador de 4 bits

$$
\begin{array}{cccccc}
 & C_{out} & & & & \\
 & \downarrow & & & & \\
 & C_3 & C_2 & C_1 & C_0 & C_{in} \\
 & & A_3 & A_2 & A_1 & A_0 \\
+ & & B_3 & B_2 & B_1 & B_0 \\
\hline
 & & S_3 & S_2 & S_1 & S_0
\end{array}
$$

Ripple Carry Addder (RCA)



Exemplo: somador de 4 bits

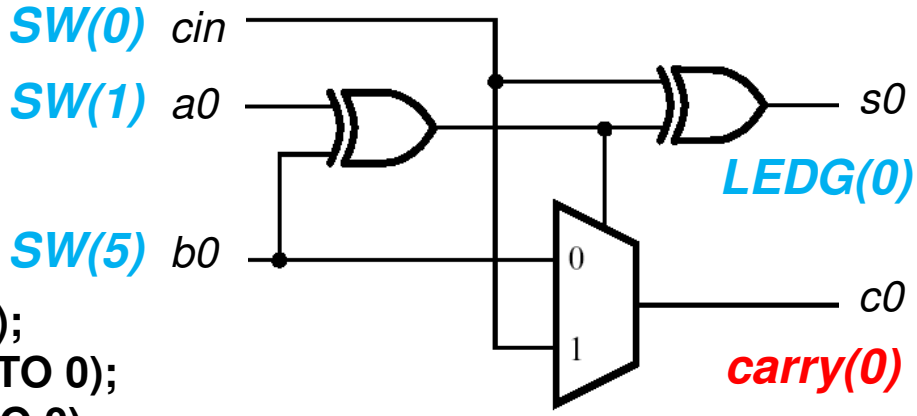
A seguir são apresentadas três implementações em VHDL para um somador de 4 bits para utilização no kit DE2 da Altera.

Algumas observações:

- **A "Solução I" é bastante semelhante às implementações em VHDL estrutural e comportamental do multiplexador e do decodificador desenvolvidas nas últimas aulas.**
- **Na "Solução I", notar a utilização do *signal carry* na *architecture*.**
- **Na "Solução II" foi criado um componente que implementa um somador completo (*Full-Adder* ou FA) na entity FA, e esse componente é utilizado na *architeture* RCA_stru da *entity* RCA.**
- **Nessa "Solução II", notar a existência de duas declarações *entity* e duas declarações *architecture* no mesmo código VHDL.**
- **Na "Solução III", foi utilizado o operador + para geração do somador de 4 bits.**
- **Para utilizar esse operador + é necessário incluir o**

```
use IEEE.std_logic_unsigned.all;
```

Solução I




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY Somador4 IS
  PORT (
    SW : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
    LEDR : OUT STD_LOGIC_VECTOR(17 DOWNTO 0);
    LEDG : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
END Somador4;
ARCHITECTURE lab4_stru OF Somador4 IS
  signal carry: std_logic_vector(3 downto 0);
BEGIN
  LEDG(0) <= ((SW(1) xor SW(5)) xor SW(0));
  carry(0) <= SW(5) when ((SW(1) xor SW(5)) = '0') else SW(0);
  LEDG(1) <= ((SW(2) xor SW(6)) xor carry(0));
  carry(1) <= SW(6) when ((SW(2) xor SW(6)) = '0') else carry(0);
  LEDG(2) <= ((SW(3) xor SW(7)) xor carry(1));
  carry(2) <= SW(7) when ((SW(3) xor SW(7)) = '0') else carry(1);
  LEDG(3) <= ((SW(4) xor SW(8)) xor carry(2));
  carry(3) <= SW(8) when ((SW(4) xor SW(8)) = '0') else carry(2);
  LEDG(4) <= carry(3);
  LEDR <= SW;
END lab4_stru;
```

```
-- SW(0) = Cin
-- SW(1) = A0
-- SW(2) = A1
-- SW(3) = A2
-- SW(4) = A3
-- SW(5) = B0
-- SW(6) = B1
-- SW(7) = B2
-- SW(8) = B3
-- LEDG(0) = S0
-- LEDG(1) = S1
-- LEDG(2) = S2
-- LEDG(3) = S3
-- LEDG(4) = Cout
```

```
library  ieee;
use  ieee.std_logic_1164.all;
-- somador completo (FA)
entity FA is
   port (a, b, c: in std_logic;
         soma, carry: out std_logic);
   end FA;
architecture FA_beh of FA is
begin
   soma <= (a xor b) xor c;
   carry <= b when ((a xor b) = '0')
                else c;
end FA_beh;
```

```
-- SW(0) = Cin
-- SW(1) = A0
-- SW(2) = A1
-- SW(3) = A2
-- SW(4) = A3
-- SW(5) = B0
-- SW(6) = B1
-- SW(7) = B2
-- SW(8) = B3
-- LEDG(0) = S0
-- LEDG(1) = S1
-- LEDG(2) = S2
-- LEDG(3) = S3
-- LEDG(6) = Cout
-- LEDG(7) = Overf
```

Solução II

```
library  ieee;
use  ieee.std_logic_1164.all;
entity RCA is
   port (SW : IN STD_LOGIC_VECTOR(17 downto 0);
         LEDG : OUT STD_LOGIC_VECTOR(7 downto 0)
       );
end RCA;

architecture RCA_stru of RCA is
   signal carry: std_logic_vector (3 downto 0);
component FA
   port (a, b, c: in std_logic;
         soma, carry: out std_logic);
   end component;
begin
FA0:
  FA port map (sw(1), sw(5), sw(0), LEDG(0), carry(0));
FA1:
  FA port map (sw(2), sw(6), carry(0), LEDG(1), carry(1));
FA2:
  FA port map (sw(3), sw(7), carry(1), LEDG(2), carry(2));
FA3:
  FA port map (sw(4), sw(8), carry(2), LEDG(3), carry(3));
LEDG(6) <= carry(3);                    -- carry out
LEDG(7) <= carry(2) xor carry(3);   -- overflow

end RCA_stru;
```

Solução III

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity RCA is
   port (SW : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
         LEDG : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
        );
end RCA;

architecture RCA_stru of RCA is
begin
   LEDG(3 downto 0) <= SW(4 downto 1) + SW(8 downto 5) + ("000" & SW(0));
end RCA_stru;
```

```
-- SW(0) = Cin
-- SW(1) = A0
-- SW(2) = A1
-- SW(3) = A2
-- SW(4) = A3
```

```
-- SW(5) = B0
-- SW(6) = B1
-- SW(7) = B2
-- SW(8) = B3
```

```
-- LEDG(0) = S0
-- LEDG(1) = S1
-- LEDG(2) = S2
-- LEDG(3) = S3
```

Resultado da síntese (*compile* no Quartus II)

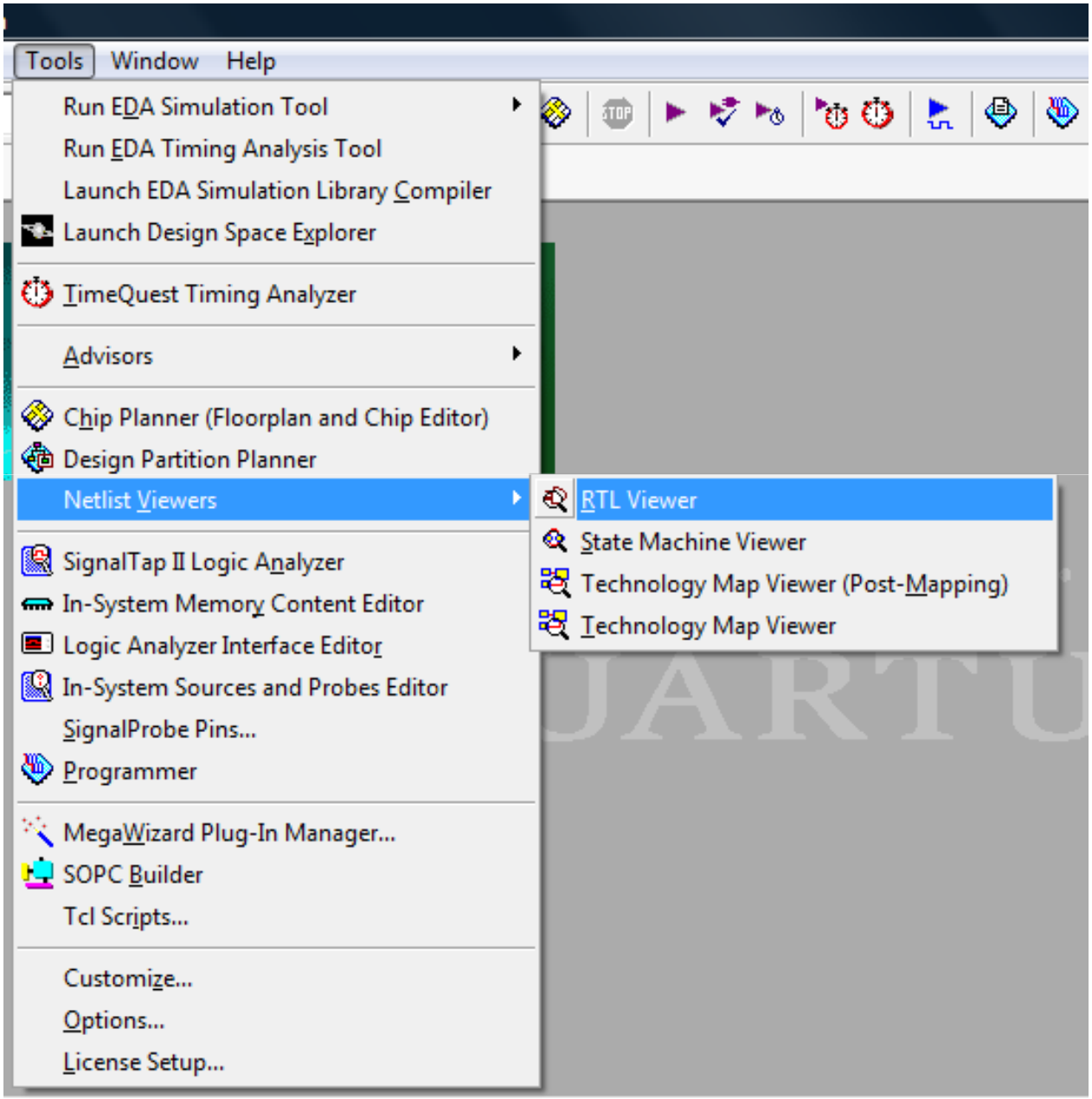
Solução II

Flow Status	Successful - Sat Oct 1
Quartus II Version	9.1 Build 350 03/24/2
Revision Name	rca
Top-level Entity Name	RCA
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	10 / 33,216 (< 1 %)
Total combinational functions	10 / 33,216 (< 1 %)
Dedicated logic registers	0 / 33,216 (0 %)
Total registers	0
Total pins	44 / 475 (9 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Solução III

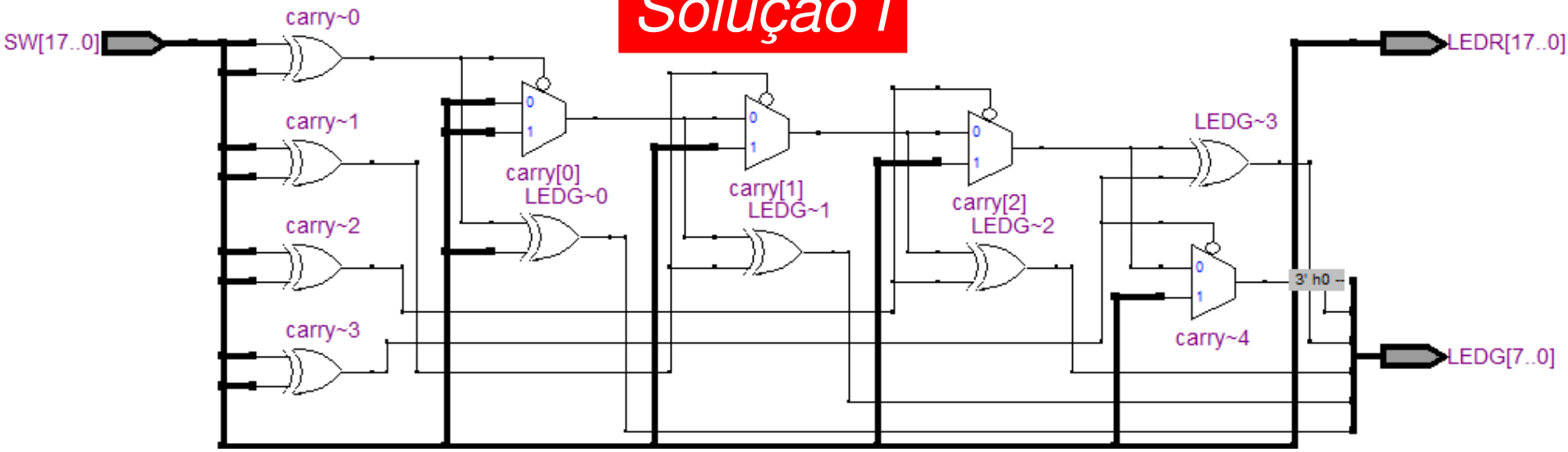
Flow Status	Successful - Sat Oct
Quartus II Version	9.1 Build 350 03/24/
Revision Name	rca
Top-level Entity Name	RCA
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	5 / 33,216 (< 1 %)
Total combinational functions	5 / 33,216 (< 1 %)
Dedicated logic registers	0 / 33,216 (0 %)
Total registers	0
Total pins	26 / 475 (5 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Utilizando o *RTL Viewer* do Quartus II



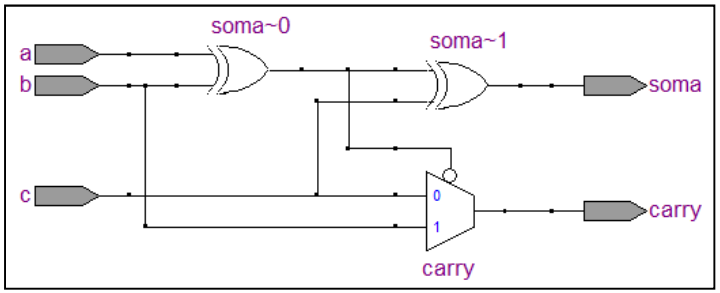
Utilizando o *RTL Viewer* do Quartus II

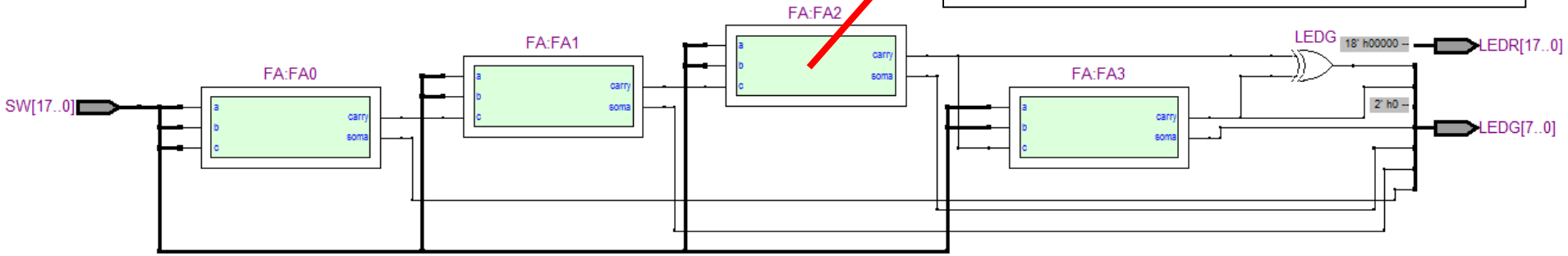
Solução I



Solução II

Entrando no projeto hierárquico





Flags

- **Na Solução II foram realizadas as seguintes atribuições:**

```
LEDG(6) <= carry(3);
LEDG(7) <= carry(2) xor carry(3);
```

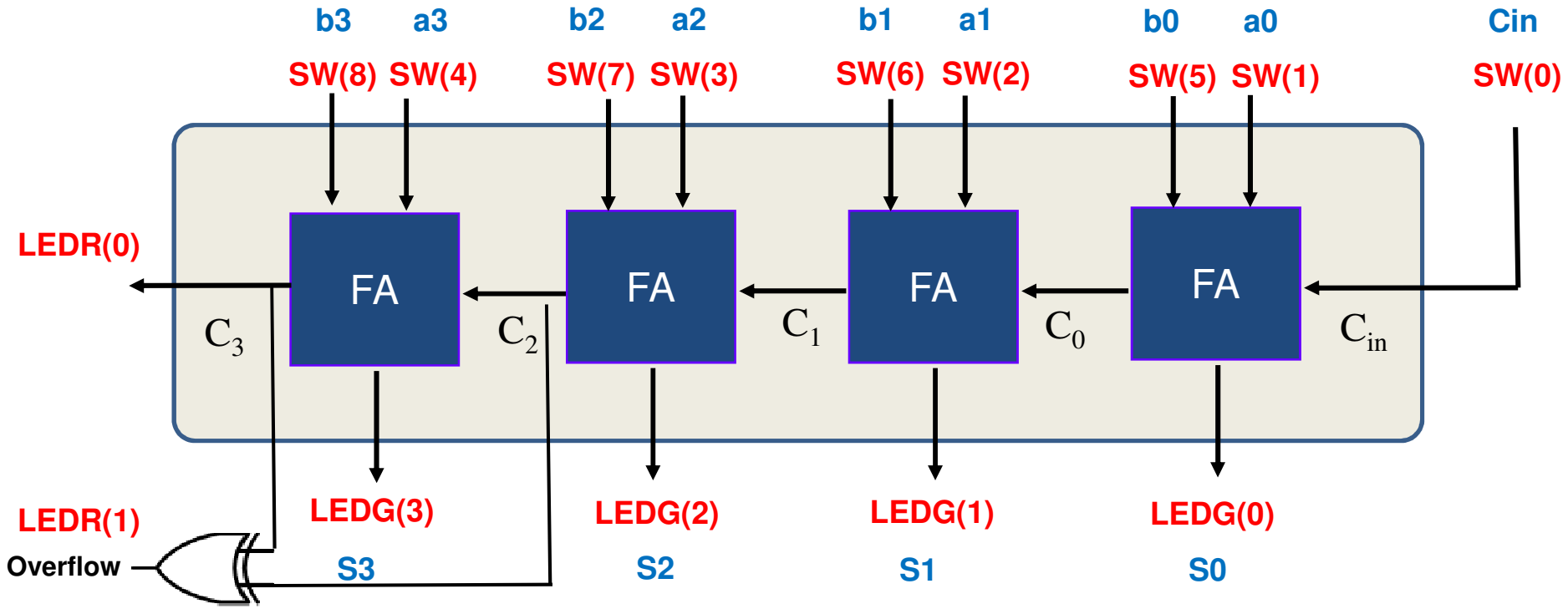
- **A princípio, essas sinalizações (*flags*) não são necessárias, uma vez que o resultado da soma já está sendo indicado em LEDG(3), LEDG(2), LEDG(1), LEDG(0).**

- **Porém, flags são bastante úteis para sinalizar situações, tais como:**
  - **Resultado está incorreto devido a *overflow* (LEDG(7))**
  - **Somador gerou um carry out (LEDG(6))**
  - **Resultado da soma foi um valor negativo**
  - **Resultado da soma foi ZERO**

Tarefa a ser realizada:
Projeto de somador de 8 bits com flags
"construindo uma calculadora"

Projeto de somador paralelo RCA

- O diagrama a seguir apresenta um RCA de 4 bits
- Utiliza um XOR entre o C_2 e C_3 para indicar ocorrência de *overflow*.
- Sinaliza em LEDR(0) a ocorrência de carry out



Tarefa

A partir da Solução II apresentada anteriormente, implementar um somador de 8 bits com as seguintes entradas e saídas:

Entrada Cin e A	*Entrada B*	*Saída Soma*	*Saída Flags*
SW(0) = Cin	SW(9) = B0	LEDG(0) = S0	LEDR(0) = carry out
SW(1) = A0	SW(10) = B1	LEDG(1) = S1	LEDR(1) = overflow
SW(2) = A1	SW(11) = B2	LEDG(2) = S2	LEDR(2) = negativo
SW(3) = A2	SW(12) = B3	LEDG(3) = S3	LEDR(3) = zero
SW(4) = A3	SW(13) = B4	LEDG(4) = S4	
SW(5) = A4	SW(14) = B5	LEDG(5) = S5	
SW(6) = A5	SW(15) = B6	LEDG(6) = S6	
SW(7) = A6	SW(16) = B7	LEDG(7) = S7	
SW(8) = A7			

$$
\begin{array}{rcccccccc}
 & C_8 & C_7 & C_6 & C_5 & C_4 & C_3 & C_2 & C_1 \\
 & & A_7 & A_6 & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 \\
+ & & B_7 & B_6 & B_5 & B_4 & B_3 & B_2 & B_1 & B_0 \\
\hline
 & & S_7 & S_6 & S_5 & S_4 & S_3 & S_2 & S_1 & S_0
\end{array}
$$

← "vai-um" (*carry*)
← 1º operando
← 2º operando
← soma

Tarefa (continuação)

- **Os LEDs verdes (LEDG) deverão apresentar o resultado da soma dos operandos A e B (fornecidos com SW).**
- **O LED vermelho 0 deverá acender, sempre que o resultado de uma soma gerar um *carry out*, caso contrário deverá permanecer apagado.**
- **O LED vermelho 1 deverá acender, sempre que o resultado de uma soma resultar em *overflow*, caso contrário deverá permanecer apagado.**
- **O LED vermelho 2 deverá acender, sempre que o resultado de uma soma resultar em um valor negativo (bit 7 = '1'), caso contrário deverá permanecer apagado.**
- **O LED vermelho 3 deverá acender, sempre que o resultado de uma soma resultar em zero, caso contrário deverá permanecer apagado.**

LEDR(0) = carry out
LEDR(1) = overflow
LEDR(2) = negativo
LEDR(3) = zero

Tarefa (continuação)

• Realizar a **simulação temporal** (e não funcional das aulas anteriores), de forma a verificar os valores intermediários da soma, até o circuito combinacional do somador fornecer o resultado correto na saída.

Tarefa adicional!!

Para aqueles que conseguiram terminar a tarefa prevista para o laboratório:

• **Projetar um decodificador de 7-segmentos**, semelhante ao da aula anterior, porém com capacidade para decodificação de números de 0 a F, para apresentação dos operandos e do resultado da soma nos displays de 7-segmentos.

Simulação com *ModelSim*

Simulação do projeto com ModelSim

1. Criar uma nova pasta dentro da pasta do projeto.
2. Copiar os *scripts* de simulação disponíveis na página da disciplina para dentro da nova pasta.
3. Entrar na nova pasta, editar o arquivo "*compila.do*", e alterar *rca.vhd* para o nome do seu arquivo VHDL a ser simulado.
4. Copiar APENAS o seu arquivo VHDL (somador de 8 bits) a ser simulado para a nova pasta, que já deve possuir os *scripts* de simulação copiados da página da disciplina.
5. Executar o *ModelSim-Altera*, que se encontra no menu *Iniciar* do Windows, pasta "*Altera*".
6. No menu "*File*" do *ModelSim*, definir a pasta do projeto (opção "*Change Directory*"), selecionando a nova pasta.

Simulação do projeto com ModelSim (cont.)

7. **Execução da simulação (arquivo *compila.do*)**
   a) **No menu "Tools" do ModelSim, selecionar "*Tcl*" -> "*Execute Macro*".**
   b) **Selecionar o arquivo "*compila.do*", e "*Open*".**

8. **O *ModelSim* irá compilar os arquivos VHDL e iniciar a simulação.**
9. **A janela com as formas de onda irá abrir, apresentando o resultado da simulação.**

***Obs.* Se desejar, editar o arquivo tb.vhd para alterar a simulação a ser realizada, e repetir o passo 7.**

Simulação do projeto com ModelSim (cont.)

Obs. **Se o resultado da simulação não estiver de acordo com o esperado, alterar o seu VHDL, salvar, e executar novamente a simulação (arquivo *compila.do*).**

Obs. **A simulação só irá funcionar se o seu projeto possuir EXATAMENTE a seguinte *entity*:**

```
entity rca IS
    PORT (SW   : IN   STD_LOGIC_VECTOR(17 DOWNTO 0);
          LEDG: OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
          LEDR: OUT STD_LOGIC_VECTOR(17 DOWNTO 0)
    );
end rca;
```

Simulação do projeto com ModelSim (cont.)

Obs. O arquivo "*compila.do*" contém os comandos do *ModelSim* necessários para realizar a simulação, incluindo:

a) Criação da biblioteca de trabalho - comando *vlib.*

b) Compilação dos arquivos VHDL para a biblioteca de trabalho - comando *vcom*.

c) Inicialização do simulador com o arquivo *testbench* - comando *vsim*.

d) Execução da janela de formas de onda (*waveform*) - comando *wave*.

e) Adição dos sinais na janela de formas de onda - comando *wave*.

f) Execução da simulação - comando *run*.