Atividade 1 - Dispositivos Lógicos Programáveis II

Daniel Tatsch, Nelson Alves, Schaiana Sonaglio
Engenharia de Telecomunicações
IFSC - Instituto Federal de Santa Catarina, São José, SC
Abril de 2018

1 Introdução

Este documento apresentará o desenvolvimento, os resultados e a análise de dois exercícios propostos na disciplina de DLP2 (Dispositivos Lógicos Programáveis II), feitos no software Quartus. No primeiro exercício, foram feitas manipulações, utilizando os recursos da linguagem VHDL, para a criação de uma situação de falso caminho crítico; no segundo exercício, foram analisados os tempos de propagação (*delays*) e a área ocupada por um circuito somador, quando utilizado como entrada dois operadores variáveis ou um operador variável e diferentes valores constantes.

2 Implementação

Utilizando o software Quartus, foram implementados códigos VHDL que viabilizassem o proposto em cada exercício. O dispositivo utilizado foi o EP4CE115F29C7 e em todas as execuções foi utilizado logic lock.

2.1 Exercício 1: criação de uma situação de falso caminho crítico

Criar um falso caminho crítico, utilizando recursos da linguagem VHDL.

Neste exercício, foram utilizadas duas implementações de somadores desenvolvidos em aulas anteriores da disciplina.

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity somador_serie is
    Generic (N : INTEGER := 32);
    port(
        a : IN std_logic_vector(N-1 downto 0);
        b : IN std_logic_vector(N-1 downto 0);
        c : IN std_logic_vector(N-1 downto 0);
        d : IN std_logic_vector(N-1 downto 0);
        e : IN std_logic_vector(N-1 downto 0);
        f : IN std_logic_vector(N-1 downto 0);
        g : IN std_logic_vector(N-1 downto 0);
        h : IN std_logic_vector(N-1 downto 0);
        y: OUT std_logic_vector(N-1 downto 0)
    );
end entity;

architecture somador_serie of somador_serie is
BEGIN
    y <= std_logic_vector(((((((unsigned(a) + unsigned(b))+unsigned(c))+unsigned(d))
    + unsigned(e))+unsigned(f))+unsigned(g))+unsigned(h));
end architecture;
```

Figura 1 - Código somador série. Fonte: elaboração própria

```
    Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity somador_paralelo is
    Generic (N : INTEGER := 32);
    port(
        a : IN std_logic_vector(N-1 downto 0);
        b : IN std_logic_vector(N-1 downto 0);
        c : IN std_logic_vector(N-1 downto 0);
        d : IN std_logic_vector(N-1 downto 0);
        e : IN std_logic_vector(N-1 downto 0);
        f : IN std_logic_vector(N-1 downto 0);
        g : IN std_logic_vector(N-1 downto 0);
        h : IN std_logic_vector(N-1 downto 0);
        y: OUT std_logic_vector(N-1 downto 0)
    );
end entity;

architecture somador_paralelo of somador_paralelo is
BEGIN
    y <= std_logic_vector(((unsigned(a) + unsigned(b))+(unsigned(c)+unsigned(d)))
    + ((unsigned(e) + unsigned(f))+(unsigned(g)+unsigned(h))));
end architecture;
```

Figura 2 - Código somador paralelo. Fonte: elaboração própria

A única diferença entre as duas implementações é a ordem em que as somas são realizadas entre os operandos (alterando com o uso dos parênteses). Com essa simples modificação, pode-se observar uma melhoria significativa no *delay* do caminho crítico (*delay* que despreza os caminhos que ligam o bloco com a extremidade do FPGA) do circuito.

A partir desses somadores, foi desenvolvido um circuito que força a definição errada de um caminho crítico. Para a interligação dos somadores implementados, foram utilizados dois MUX com o *enable* em comum. Com isso, garantimos que o caminho crítico real do sistema fosse definido quando ambos os MUX estão em ’0’ ou ’1’, podendo compará-lo com o falso caminho gerado pelo compilador. O código VHDL do circuito é apresentado na Figura 3, na Figura 4 e na Figura 5.

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity falso_caminho is
    Generic (N : INTEGER := 32);
    port(
        sel : IN std_logic;
        a1 : IN std_logic_vector(N-1 downto 0);
        a2: IN std_logic_vector(N-1 downto 0);
        a3: IN std_logic_vector(N-1 downto 0);
        a4: IN std_logic_vector(N-1 downto 0);
        a5: IN std_logic_vector(N-1 downto 0);
        a6: IN std_logic_vector(N-1 downto 0);
        a7: IN std_logic_vector(N-1 downto 0);
        a8: IN std_logic_vector(N-1 downto 0);
        y: OUT std_logic_vector(N-1 downto 0)
    );
end entity;
```

Figura 3 - Código caminho falso parte 1. Fonte: elaboração própria

```
COMPONENT somador_serie is
    port(
        a : IN std_logic_vector(N-1 downto 0);
        b : IN std_logic_vector(N-1 downto 0);
        c : IN std_logic_vector(N-1 downto 0);
        d : IN std_logic_vector(N-1 downto 0);
        e : IN std_logic_vector(N-1 downto 0);
        f : IN std_logic_vector(N-1 downto 0);
        g : IN std_logic_vector(N-1 downto 0);
        h : IN std_logic_vector(N-1 downto 0);
        y: OUT std_logic_vector(N-1 downto 0)
    );
end COMPONENT;

COMPONENT somador_paralelo is
    port(
        a : IN std_logic_vector(N-1 downto 0);
        b : IN std_logic_vector(N-1 downto 0);
        c : IN std_logic_vector(N-1 downto 0);
        d : IN std_logic_vector(N-1 downto 0);
        e : IN std_logic_vector(N-1 downto 0);
        f : IN std_logic_vector(N-1 downto 0);
        g : IN std_logic_vector(N-1 downto 0);
        h : IN std_logic_vector(N-1 downto 0);
        y: OUT std_logic_vector(N-1 downto 0)
    );
end COMPONENT;
```

Figura 4 - Código caminho falso parte 2. Fonte: elaboração própria

```
SIGNAL saida_s1, saida_p1 : std_logic_vector(N-1 downto 0);
SIGNAL saida_s2, saida_p2 : std_logic_vector(N-1 downto 0);
SIGNAL y1 : std_logic_vector(N-1 downto 0);
BEGIN
   componente_somadorS1 : somador_serie
   PORT MAP (a => a1, b => a2, c => a3, d => a4, e => a5, f => a6, g => a7, h => a8, y => saida_s1);

   componente_somadorP1 : somador_paralelo
   PORT MAP (a => a1, b => a2, c => a3, d => a4, e => a5, f => a6, g => a7, h => a8, y => saida_p1);

   componente_somadorS2 : somador_serie
   PORT MAP (a => y1, b => a2, c => a3, d => a4, e => a5, f => a6, g => a7, h => a8, y => saida_s2);

   componente_somadorP2 : somador_paralelo
   PORT MAP (a => y1, b => a2, c => a3, d => a4, e => a5, f => a6, g => a7, h => a8, y => saida_p2);

   WITH sel SELECT
   y1 <= saida_p1 WHEN '0',
         saida_s1 WHEN '1';

   WITH sel SELECT
   y <= saida_s2 WHEN '0',
        saida_p2 WHEN '1';
end ARCHITECTURE;
```

Figura 5 - Código caminho falso parte 3. Fonte: elaboração própria

2.2 Exercício 2: utilização de um somador com diferentes operandos

6.5 Assume that a and b are 16-bit inputs interpreted as unsigned numbers. Write five VHDL programs for the following operations. Synthesize the programs using an ASIC device. Compare their area and propagation delay and discuss the impact of a constant operand.

- a + b
- a + "0000000000000001"
- a + "0000000010000000"
- a + "1000000000000000"
- a + "1010101010101010"

6.6 Repeat Problem 6.5, but use an FPGA device.

Figura 6 - Exercício 6.6. Fonte: Chu (2006).

O solicitado no exercício proposto na Figura 6 foi implementado na Figura 7 - primeira condição, na Figura 8 - segunda condição, na Figura 9 - terceira condição, na Figura 10 - quarta condição e, finalmente, na Figura 11 - quinta condição.

```
----------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.numeric_std.all;
USE ieee.std_logic_1164.all;
----------------------------------------------------------------------------
ENTITY somador IS
GENERIC (N: integer := 16);
PORT (a, b: IN unsigned(N-1 DOWNTO 0);
y: OUT unsigned(N-1 DOWNTO 0));

END ENTITY;
----------------------------------------------------------------------------
ARCHITECTURE somador OF somador IS


BEGIN

y <= a+b;

END ARCHITECTURE;
```

Figura 7 - Código da resolução da primeira condição do exercício 6.6. Fonte: elaboração própria

```
----------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.numeric_std.all;
USE ieee.std_logic_1164.all;
----------------------------------------------------------------------------
ENTITY somador1 IS
GENERIC (N: INTEGER := 16);
PORT (a: IN unsigned(N-1 DOWNTO 0);
y: OUT unsigned(N-1 DOWNTO 0));

END ENTITY;
----------------------------------------------------------------------------
ARCHITECTURE somador1 OF somador1 IS


BEGIN

y <= a+"0000000000000001";

END ARCHITECTURE;
```

Figura 8 - Código da resolução da segunda condição do exercício 6.6. Fonte: elaboração própria

Conforme ilustrado na Figura 8, foi realizada a soma de um operando variável e de um valor de constante (0000000000000001), para análise do seu tempo de propagação e da sua área.

```
----------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.numeric_std.all;
USE ieee.std_logic_1164.all;
----------------------------------------------------------------------------
ENTITY somador2 IS
GENERIC (N: INTEGER := 16);
PORT (a: IN unsigned(N-1 DOWNTO 0);
y: OUT unsigned(N-1 DOWNTO 0));

END ENTITY;
----------------------------------------------------------------------------
ARCHITECTURE somador2 OF somador2 IS


BEGIN

y <= a+"0000000010000000";

END ARCHITECTURE;
```

Figura 9 - Código da resolução da terceira condição do exercício 6.6. Fonte: elaboração própria

Conforme ilustrado na Figura 9, foi realizada a soma de um operando variável e de um valor de constante (0000000010000000), para análise do seu tempo de propagação e da sua área.

Conforme ilustrado na Figura 10, foi realizada a soma de um operando variável e de um valor de constante (1000000000000000), para análise do seu tempo de propagação e da sua área.

```
----------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.numeric_std.all;
USE ieee.std_logic_1164.all;
----------------------------------------------------------------------------
ENTITY somador3 IS
GENERIC (N: INTEGER := 16);
PORT (a: IN unsigned(N-1 DOWNTO 0);
y: OUT unsigned(N-1 DOWNTO 0));

END ENTITY;
----------------------------------------------------------------------------
ARCHITECTURE somador3 OF somador3 IS


BEGIN

y <= a+"1000000000000000";

END ARCHITECTURE;
```

Figura 10 - Código da resolução da quarta condição do exercício 6.6. Fonte: elaboração própria

```
----------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.numeric_std.all;
USE ieee.std_logic_1164.all;
----------------------------------------------------------------------------
ENTITY somador4 IS
GENERIC (N: INTEGER := 16);
PORT (a: IN unsigned(N-1 DOWNTO 0);
y: OUT unsigned(N-1 DOWNTO 0));

END ENTITY;
----------------------------------------------------------------------------
ARCHITECTURE somador4 OF somador4 IS


BEGIN

y <= a+"1010101010101010";

END ARCHITECTURE;
```

Figura 11 - Código da resolução da quinta condição do exercício 6.6. Fonte: elaboração própria

Conforme ilustrado na Figura 11, foi realizada a soma de um operando variável e de um valor de constante (1010101010101010), para análise do seu tempo de propagação e da sua área.

3 Apresentação e Análise dos Resultados

3.1 Exercício 1: RTL do falso caminho crítico

A figura a seguir apresenta o RTL do *hardware* descrito nas Figuras 3,6 e 5.
A Figura 12 e a Figura 13 detalham a arquitetura de cada um desses somadores.

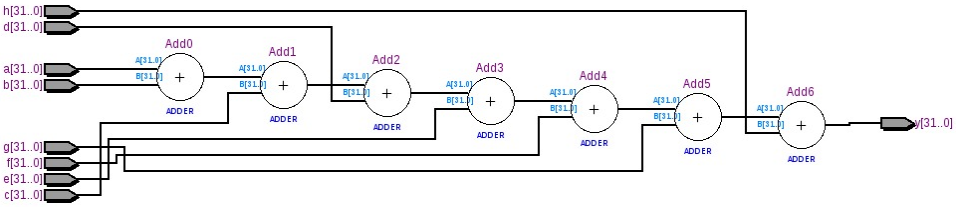

Figura 12 - RTL somador série. Fonte: elaboração própria

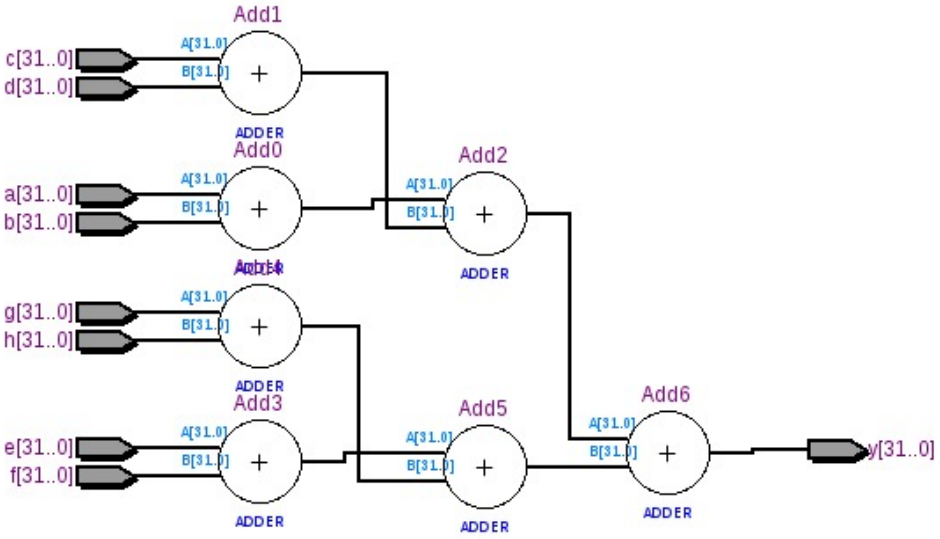

Figura 13 - RTL somador paralelo. Fonte: elaboração própria

3.2 Exercício 1: Análise do falso caminho crítico

O caminho crítico real do circuito ocorre na passagem de um somador em cascata com um somador em paralelo (*enable* dos MUX em '0' CONFERIR) ou de um somador em paralelo com um somador em cascata(*enable* dos MUX em '1'), com um tempo de propagação de 14.574 ns, sendo mais rápido que o caminho falso apresentado pelo compilador de 20.642 ns. Isso ocorre porque o compilador não verifica as interpretações de *hardware* durante a análise, considerando apenas os *delays* de cada trecho separadamente.

3.3 Exercício 1: comandos utilizados para obtenção do tempo de propagação do caminho crítico

Tempo de propagação total:

report_path -npaths 1 -panel_name Report Path Tempo de propagação do caminho crítico:

False path: report_path -from componente_somadorS* -to componente_somadorP* -npath 20 -panel_name Report Path

Somadores: report_path -from Add* -to Add* -npath 20 -panel_name Report Path

3.4 Exercício 2: RTL de cada condição do somador

Na Figura 14, são utilizados todos os bits dos operandos a e b para realizar a soma.

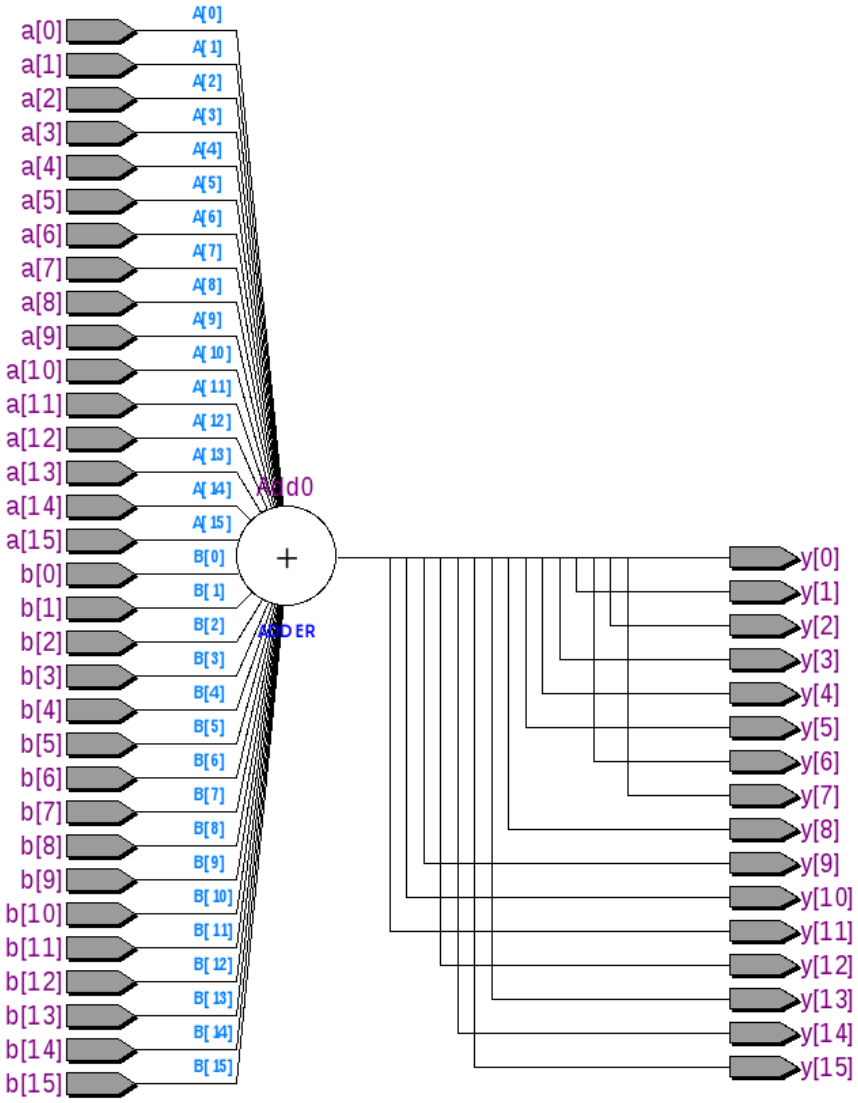

Figura 14 - RTL: y = a + b. Fonte: elaboração própria.

Na Figura 15, são utilizados todos os bits do operando a somados com todos os dezesseis bits da constante.

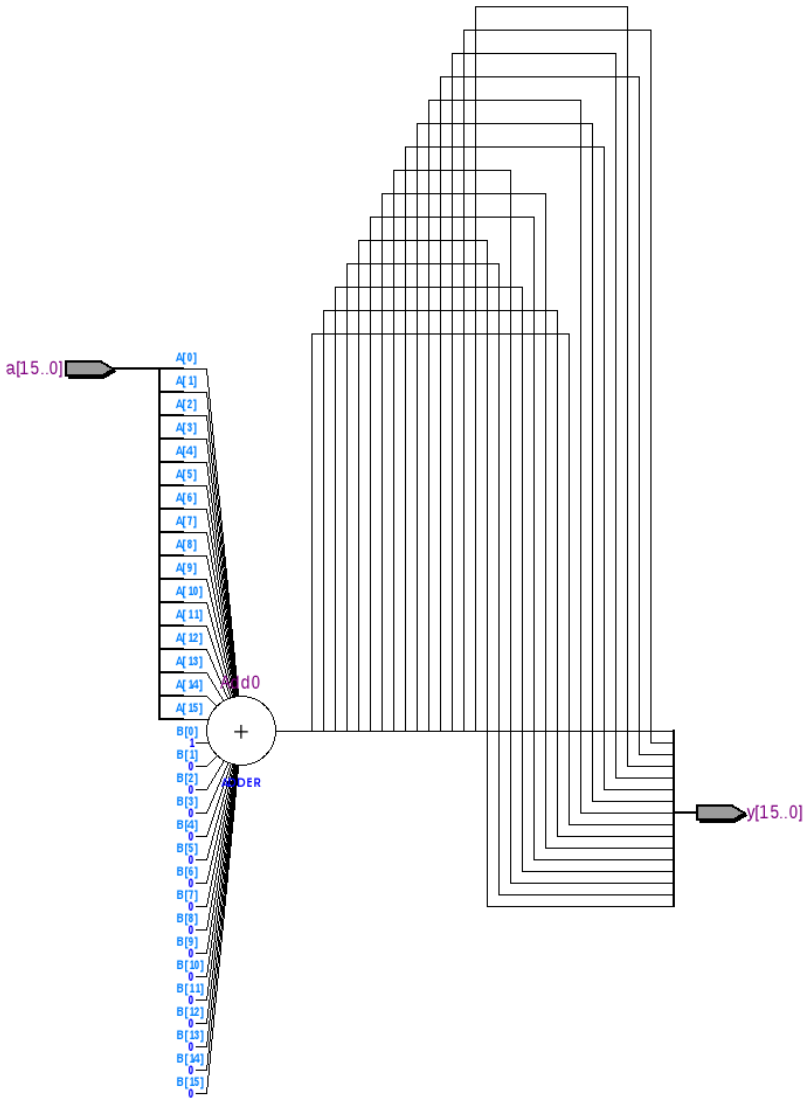

Figura 15 - RTL: y = a + "0000000000000001". Fonte: elaboração própria.

Na Figura 16, os sete primeiros bits do operando a são conectados diretamente aos sete primeiros bits da saída y, pois qualquer valor somado ao zero da constante é o próprio valor. Os outros nove bits da saída y é o resultado da soma entre os nove últimos bits do operando a com a constante "000000001".

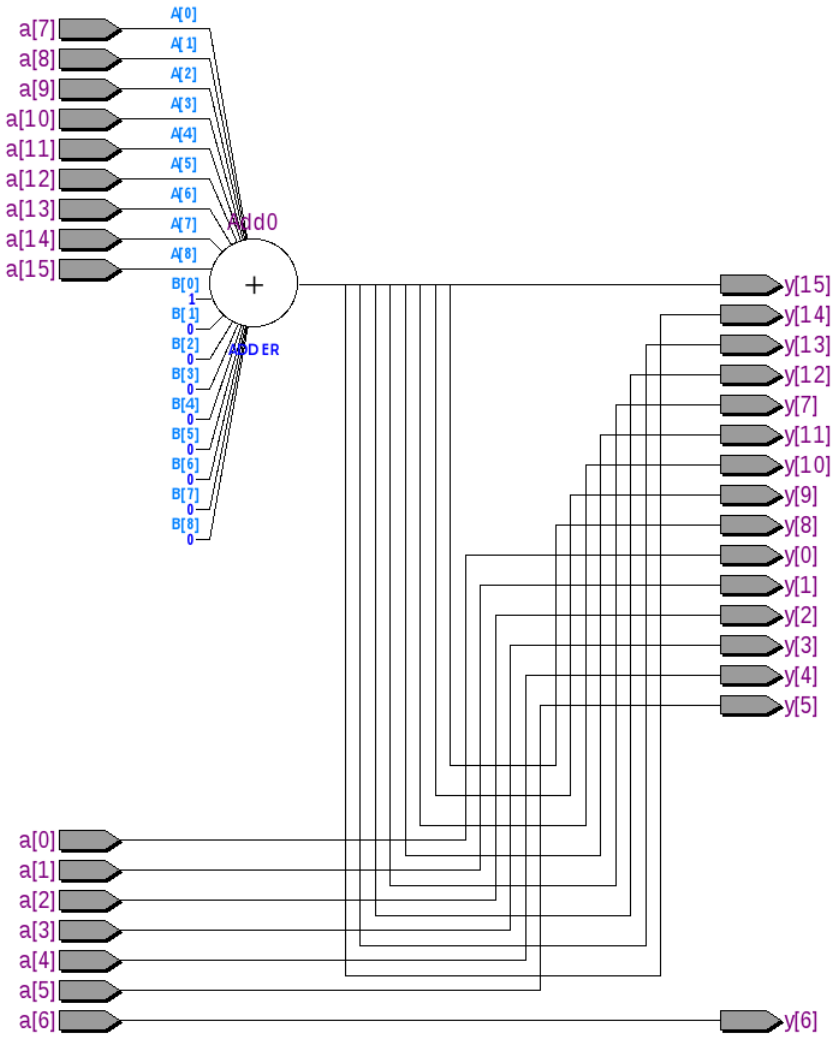

Figura 16 - RTL: y = a + "0000000010000000". Fonte: elaboração própria.

Na Figura 17, os quinze primeiros bits do operando a são conectados diretamente aos quinze primeiros bits da saída y. O último bit do operando a é somado à constante "1", podendo ocorrer overflow.

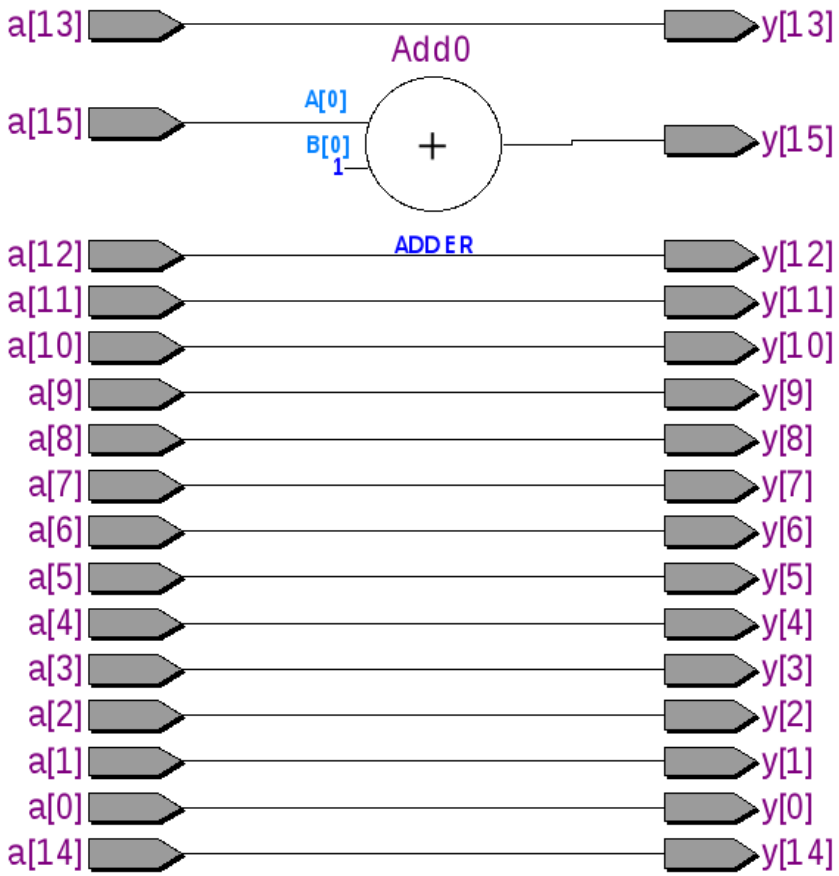

Figura 17 - RTL: y = a + "1000000000000000". Fonte: elaboração própria.

Na Figura 18, o primeiro bit do operando a é conectado diretamente ao primeiro bit da saída y, pois qualquer valor somado ao zero da constante é o próprio valor. Os demais bits do operando são somados à constante "101010101010101".

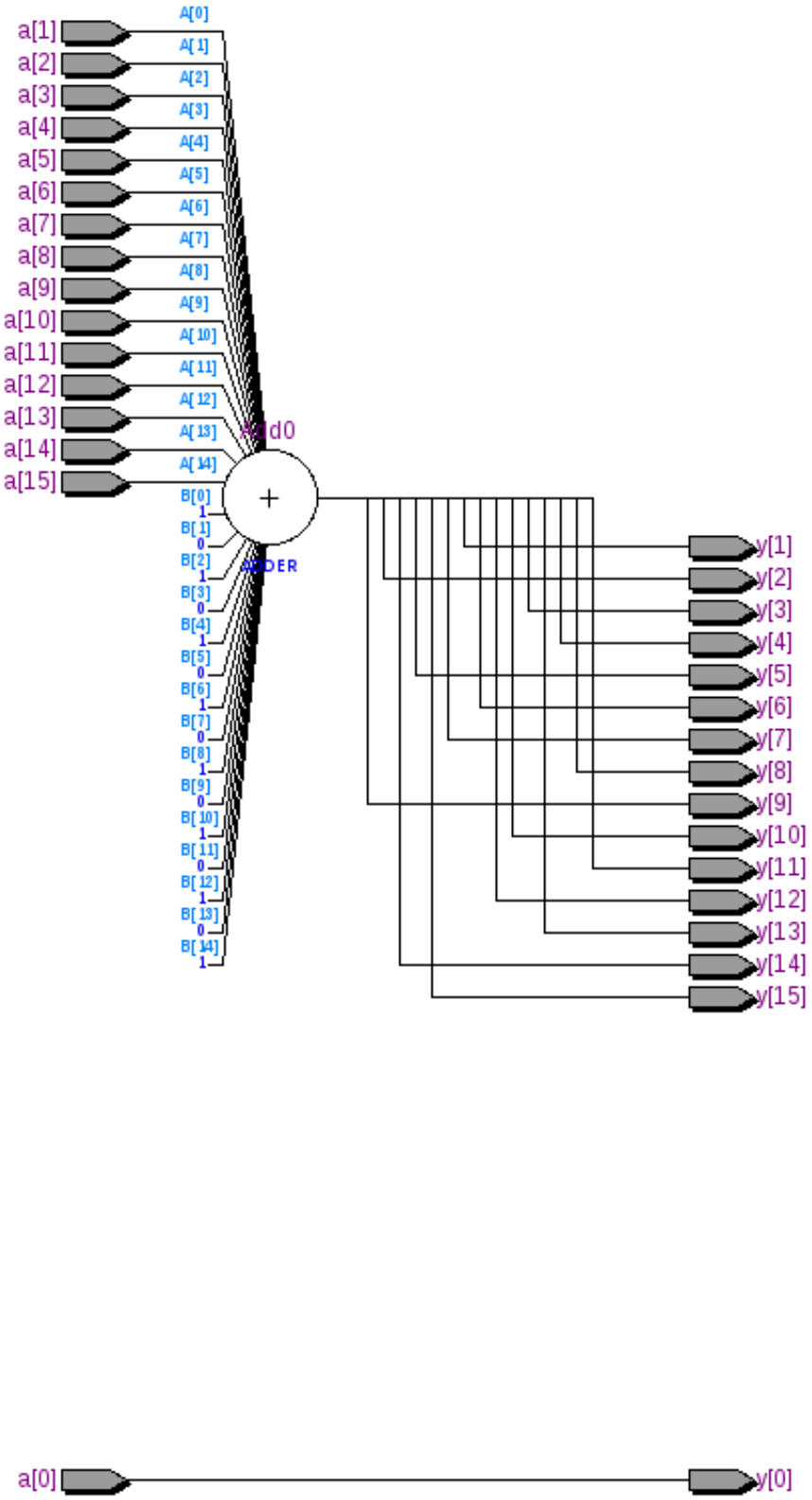

Figura 18 - RTL: y = a + "1010101010101010". Fonte: elaboração própria.

3.5 Exercício 2: Análise do somador com diferentes operandos

Os resultados obtidos nos diferentes somadores está ilustrado na Figura 19.

somador	y = a + b	y = a +"0000000000000001"	y = a + "0000000010000000"	y = a +"1000000000000000"	y = a +"1010101010101010"
Total de EL	16	15	8	0	14
Delay total (ns)	11.115	10.434	9.440	8.427	10.155
Delay do caminho crítico (ns)	0.924	0.858	0.396	0.000	0.792

Figura 19 - Resultados obtidos na resolução do exercício 6.6. Fonte: elaboração própria

De acordo com a Figura 19, quando uma constante é somada ao operando a o impacto - delay + área (número de elementos lógicos) - no circuito varia dependendo do seu valor. Por exemplo, na operação y = a + "0000000010000000", parte dos bits do operando a vão diretamente para a saída y, pois não precisam ser somados com os zeros menos significativos da constante, a outra parte é somada normalmente. Sabendo disso, a compreensão da Figura 19 se torna mais fácil: em casos onde há esta "simplificação", há ou uma diminuição de área ou de delay ou dos dois. No caso do y = a + "1000000000000000", o total de elementos lógicos foi zero, mesmo aparecendo um somador no RTL, isso provavelmente ocorreu por causa da ferramenta utilizada (Quartus), que otimizou utilizando um somador já existente na placa. Na operação y = a + b, como não é possível simplificar nenhuma parte do circuito, é necessário fazer a soma de todos os bits, aumentando a área e o delay.

3.6 Exercício 2: comandos utilizados para obtenção dos tempos de propagação dos caminhos críticos

Tempo de propagação total:
y = a+b: report_path -from a*b* -to y* -npaths 20 -panel_name Report Path
Demais: report_path -from a* -to y* -npaths 20 -panel_name Report Path
Tempo de propagação do caminho crítico:
Todas as condições: report_path -from Add* -to Add* -npath 20 -panel_name Report_Path

4 Considerações Finais

Após a resolução do exercício 1 foi possível ter um maior entendimento das funcionalidades da ferramenta utilizada, o Quartus, e o seu modo de operação. Mesmo que forcemos o circuito a passar por um caminho crítico que logicamente iria acontecer, o compilador do Quartus não interpreta o *hardaware* gerado e acaba apenas verificando os blocos de maior tempo de propagação para retornar o caminho crítico que, nesse caso, é falso. Para verificar esse problema, analisamos separadamente cada um dos componentes que compunham o projeto, estimando qual seria de fato o *delay* crítico do circuito. Já no exercício 2, constatou-se que, ao utilizar constantes no somador existe a possibilidade de simplificação do circuito, diminuindo sua área/delay.

5 Referências

CHU, Pong P.. RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability. Cleveland: Wiley-ieee Press, 2006. 694 p.