UFSC - UNIVERSIDADE FEDERAL DE SANTA CATARINA
CTC - CENTRO TECNOLÓGICO
INE - DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES

Relatório de trabalho prático 1 do Grupo 7: Algoritmo de Booth

Alunos: Estevão Felzke da Rosa
Gabriela de Moura V. Pereira
Jonas Batista

Professor: Mateus Grellert

Florianópolis, 2022

Sumário

1. Introdução

O escopo da atividade procura entregar um multiplicador de dois números com sinal. A multiplicação de números binários é uma das operações mais básicas e fundamentais da computação. A partir dela se originam funções mais complexas, a versão mais básica de todas começa pela multiplicação de dois números naturais e não envolve muita complexidade. É uma sequência de somas, onde cada bit do multiplicador define se o valor da respectiva linha vai ser nulo ou igual ao multiplicando. A cada iteração do multiplicador, a cada dígito, a linha inferior se posiciona uma casa à direita, como uma multiplicação decimal. Existem muitos outros algoritmos para essa operação, todos derivados desse primeiro.

Para essa atividade o grupo optou pelo algoritmo de booth.

a. Algoritmo de Booth;

Começa-se armazenando o número de bits que o multiplicador/multiplicando possui na variável “N”, esse número de bits será a quantidade de deslocamentos que a operação irá realizar. Ainda usaremos um número “A” (nulo por default) com essa mesma quantidade de bits para contabilizar as operações, bem como efetuar os deslocamentos. O multiplicador ainda receberá um bit a mais do lado direito (parcela menos significativa) que recebe o nome de “Q-1” e recebe 0 por default.

O algoritmo inicia-se comparando os dois bits menos significativos do multiplicador, se “Q0” e “Q-1” forem iguais, sejam 00 ou 11 então só ocorrerão os deslocamentos. No caso da variável “A” e “Q” são deslocados um bit para a direita, sem nenhuma operação e decremento da variável N. Se “Q0” e “Q-1” forem iguais a 01 então antes do decremento e dos deslocamentos a variável A tem subtraído o valor do multiplicando, caso sejam iguais a 10 então A tem somado o valor do multiplicando. O loop se encerra quando a variável N é verificada como sendo igual a um no final do laço.

O algoritmo possui um grau de complexidade maior pois o mesmo gera produtos parciais e aplica adição e subtração ao longo do processo de multiplicação.

.

b. Proposta do trabalho;

A questão pesquisa que esse trabalho visa responder é: como circuitos de multiplicação comparam-se entre si em termos de área, frequência e taxa de processamento? Seguindo essa lógica tentaremos implementar o circuito da maneira mais otimizada possível. Tanto em matéria de tempo, quanto de área, consumo de energia, etc.

2. Conceitos fundamentais

a. ULAs;

A Unidade Lógica Aritmética é um circuito digital que realiza operações de adição e operações booleanas AND. Seu funcionamento varia bastante. Há ULAs que realizam adição, subtração, divisão, determinam se um número é positivo, negativo ou nulo. Bem como ULAs só para operações booleanas. Sendo parte fundamental da unidade central de processamento surgiu ainda na época dos relés, posteriormente adotou as válvulas para ser

implementada e mais recentemente os transistores, sendo um dos componentes mais antigos da computação.

b. FSMs;

Uma Finite State Machine (Máquina de Estados Finita) é uma ferramenta para modelagem de sistemas. É concebida uma máquina abstrata que possui um estado por vez, este chamado estado atual. Ela muda de estado, faz uma transição se, e somente se, uma condição for atendida. O estado atual contém informações a respeito dos estados anteriores, que juntos formam o caminho de estados que a execução do sistema percorreu até o estado atual. Essa ferramenta é muito simples, mas muito prática para projetar sistemas.

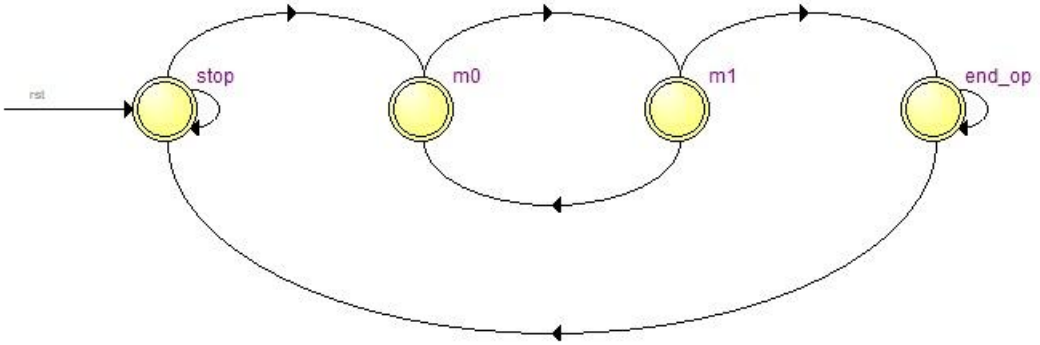


Figura 1 - FSM gerada pelo Quartus II do algoritmo empregado pelo grupo

3. Arquitetura proposta

A seguir está anexado o código main da estrutura implementada pelo grupo.

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4
5   entity multiplicador_binario is
6      port(
7         -- sinais de entrada de um bit
8         clk,in_multiplicando, rst,
9         init, in_multiplicador: in std_logic;
10        -- entrada do valor de multiplicador/ multiplicado
11        mult_in: in std_logic_vector(3 downto 0);
12        -- Saida do resultado
13        mult_out: out std_logic_vector(7 downto 0);
14        -- flag de pronto
15        finished: out std_logic );
16  end multiplicador_binario;
17
18  architecture comportamento of multiplicador_binario is
19     -- Declaração dos estados
20     type tipo_estado is (m0, m1, stop, end_op);
21     signal estado, prox_estado : tipo_estado;
22     -- Declaração dos registradores
23     signal A, B, Q, M: std_logic_vector(3 downto 0);
24     --sinal para o contador
25     signal P: std_logic_vector(1 downto 0);
26     -- sinais de flag
27     signal C, zero: std_logic;
```

Figura 2 - Primeira parte do código;

```
28  begin
29     zero <= P(1) NOR P(0);
30     -- Process de ckock/reset com alteração de estados
31     registra_estado: process (clk, rst)
32     begin
33        if (rst = '1') then
34           estado <= stop;
35        elsif (clk'event and clk = '1') then
36           estado <= prox_estado;
37        end if;
38     end process registra_estado;
39     -- Logica de estados
40     prox_estado_f: process (init, zero, estado)
41     begin
42        case estado is
43           when stop =>
44              if init = '1' then
45                 prox_estado <= m0;
46              else
47                 prox_estado <= stop;
48              end if;
49           when m0 =>
50              prox_estado <= m1;
51           when m1 =>
52              if zero = '1' then
53                 prox_estado <= end_op;
54              else
```

Figura 3 - Segunda parte do Código;

```
55                prox_estado <= m0;
56             end if;
57          when end_op =>
58             if init = '1' then
59                prox_estado <= end_op;
60             else
61                prox_estado <= stop;
62             end if;
63       end case;
64    end process prox_estado_f;
65    -- Logica de fluxo de dados
66    fluxo_dados_f: process (clk)
67    variable reg_ca: std_logic_vector(4 downto 0);
68    begin
69       if (clk'event and clk = '1') then
70          if in_multiplicando = '1' then
71             B <= mult_in;
72          end if;
73          if in_multiplicador = '1' then
74             M <= mult_in;
75          end if;
76          case estado is
77             when stop =>
78                finished <= '0';
79                if init = '1' then
80                   C <='0';
81                   A <= "0000";
```

Figura 4 - Terceira parte do código;

```
82                   Q <= M;
83                   P <= "11";
84                end if;
85             when m0 =>
86                if Q(0) = '1' then
87                   reg_ca := ('0' & A) + ('0' & B);
88                else
89                   reg_ca := C & A;
90                end if;
91                C <= reg_ca(4);
92                A <= reg_ca(3 downto 0);
93             when m1 =>
94                C <= '0';
95                A <= C & A(3 downto 1);
96                Q <= A(0) & Q(3 downto 1);
97                P <= P - "01";
98             when end_op =>
99                finished <= '1';
100         end case;
101      end if;
102   end process fluxo_dados_f;
103   mult_out <= A & Q;
104end comportamento;
```

Figura 5 - Quarta parte do código;

O código main do projeto;

A seguir a NetList da arquitetura implementada pelo grupo.

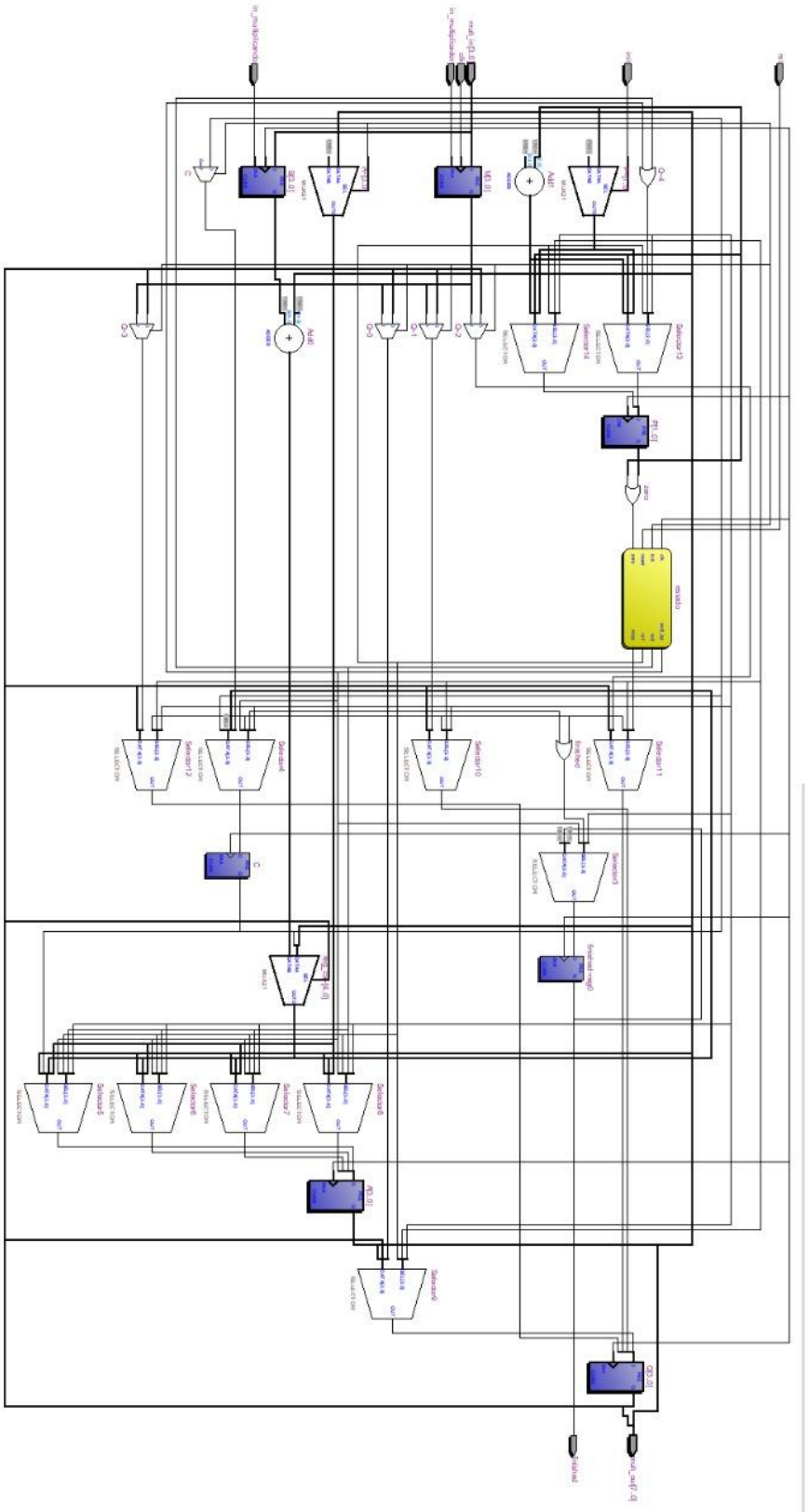


Figura 6 - NetList da arquitetura implementada

4. Resultados

Utilizando de ferramentas dispostas pelo próprio Quartus, sintetiza-se algumas plataformas de dados que ajudam a compreender e analisar o circuito sintetizado.

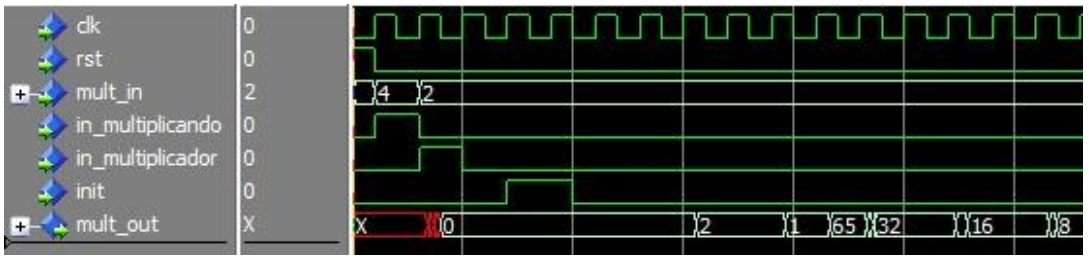


Figura 7 - Simulação RTL da arquitetura empregada

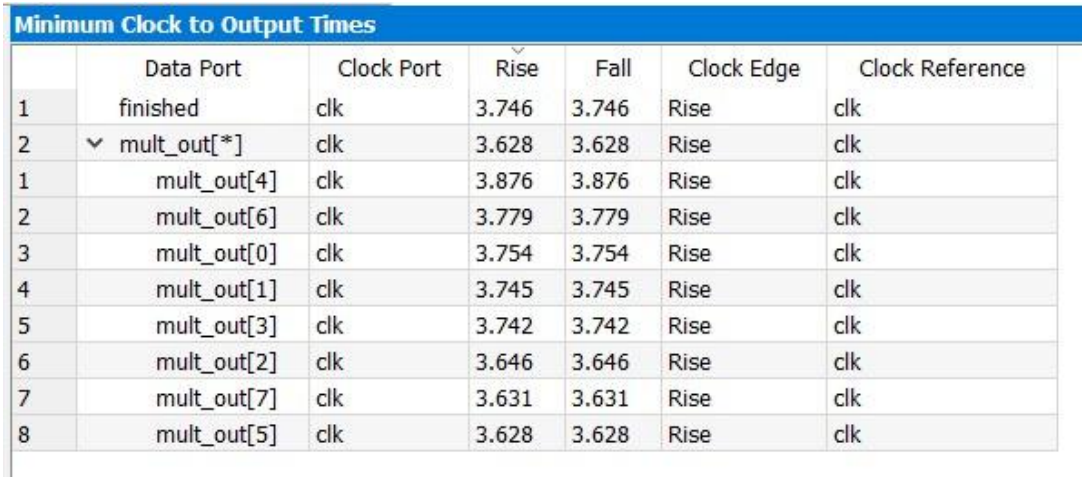
A simulação RTL mostra como os sinais de entrada e saída dos componentes se comportam conforme o sistema executa suas funções.

Flow Summary

Flow Status	Successful - Fri Jul 22 20:45:38 2022
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	multiplicador_binario
Top-level Entity Name	multiplicador_binario
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	30 / 33,216 (< 1 %)
Total combinational functions	25 / 33,216 (< 1 %)
Dedicated logic registers	24 / 33,216 (< 1 %)
Total registers	24
Total pins	18 / 475 (4 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Figura 8 - Relatório de Área

O relatório de área nos ajuda a compreender o custo de sintetização da arquitetura do multiplicador de Booth, bem como, compreender sua complexidade, dando informações como número de registradores, quantidade de memória utilizada, elementos lógicos, etc.



Minimum Clock to Output Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	finished	clk	3.746	3.746	Rise	clk
2	mult_out[*]	clk	3.628	3.628	Rise	clk
1	mult_out[4]	clk	3.876	3.876	Rise	clk
2	mult_out[6]	clk	3.779	3.779	Rise	clk
3	mult_out[0]	clk	3.754	3.754	Rise	clk
4	mult_out[1]	clk	3.745	3.745	Rise	clk
5	mult_out[3]	clk	3.742	3.742	Rise	clk
6	mult_out[2]	clk	3.646	3.646	Rise	clk
7	mult_out[7]	clk	3.631	3.631	Rise	clk
8	mult_out[5]	clk	3.628	3.628	Rise	clk

Figura 9 - Relatório de atraso

Através dele conseguimos aferir a eficiência do algoritmo implementado.

Quanto a comparações, o nosso grupo esteve defasado na proposta de comparar com um algoritmo implementado em sala, justamente por não possuir todos os membros presentes. Isso inviabilizou a proposta original, visando superar essa adversidade nos propomos a comparar os resultados da nossa arquitetura implementada com a de outros colegas (grupo 4) e chegamos a resultados satisfatórios tanto em relação à eficiência, os valores de atraso foram quase metade dos valores apresentados pelo outro algoritmo, quanto a valores de custo, essa otimização se torna ainda mais evidente chegando a ter menos de 40% do valor apresentado pelo grupo 4 em alguns quesitos. Isso mostra como o algoritmo é muito eficiente, tanto em questões de custo quanto de performance. Sendo uma arquitetura muito otimizada e que cumpre a função do desafio proposto.

5. Conclusão

O projeto foi um desafio para o grupo. A princípio tivemos grandes problemas de cronograma pois tivemos poucos encontros pessoais, e dada a complexidade do algoritmo isso tornava o desenvolvimento muito complicado, muito ruído na comunicação. Mas superamos essa adversidade e conseguimos implementar uma arquitetura muito otimizada. Sinceramente os resultados foram bem surpreendentes. O algoritmo de Booth é muito rápido e barato, chegando a ter metade do custo e o dobro de eficiência que outros algoritmos analisados. O real custo de implementação está no recurso intelectual empregado, que é usado para superar dificuldade de implementação, isso exigiu muita paciência, comunicação, persistência e estudo. Dado isso fomos bem sucedidos no desafio proposto, que era fazer o algoritmo mais otimizado possível.