INE5406-02208/B -Sistemas Digitais

Relatório Final

Cálculo do n-ésimo termo da série de Fibonacci

11

Aluno(a):
Tatianne Ramos Dal Ross

Professor:
Rafael Luiz Cancian

Novembro, 2020

Lista de figuras

Sumário

1 Introdução

Como projeto para a disciplina de Sistemas Digitais (INE5406) proponho o desenvolvimento de um circuito em VHDl que calcula o n-ésimo termo da série de Fibonacci utilizando o dispositivo Altera Cyclone IV EP4CE55F29C6.

A sequência de Fibonacci é composta por uma sucessão de números descrita pelo famoso matemático italiano Leonardo de Pisa (1170-1250), mais conhecido como Fibonacci, no final do século 12, ela é infinita e começa com 0 e 1. Os números seguintes são sempre a soma dos dois números anteriores. Portanto, depois de 0 e 1, vêm 1, 2, 3, 5, 8, 13, 21, 34...

Matematicamente, podemos definir o n-ésimo termo de Fibonacci como a soma do (n-1) -ésimo e (n-2) -ésimo.

$$F(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{if } n > 1 \end{cases}$$

Figure 1: Fórmula para sequência

2 Projeto do Sistema

O circuito em VHDL que calcula n-ésimo termo da série de fibonacci tera 4 entradas: go , n, clock, reset e 2 saídas: done e result.

O circuito deve inicialmente esperar até que go seja declarado (ativo alto), e então ler na entrada n , controlar o caminho de dados conforme necessário para gerar o n-ésimo termo da série de Fibonacci e então armazenar o resultado no registrador de result que está conectado ao resultado de saída . Depois que um resultado foi gerado, o done deve ser ativado até ir de volta para 0 (ou seja, até que o circuito começe novamente). O circuito não deve começar a calcular outro número até que go seja redefinido como 0 e em seguida, definido como 1.

2.1 Identificação das Entradas e Saídas

A figura 2 representa as 4 entradas: go , n, clock reset e as 2 saídas: done e result, propostas pelo projeto. Descrição e Captura do Comportamento O comportamento foi capturado em 7

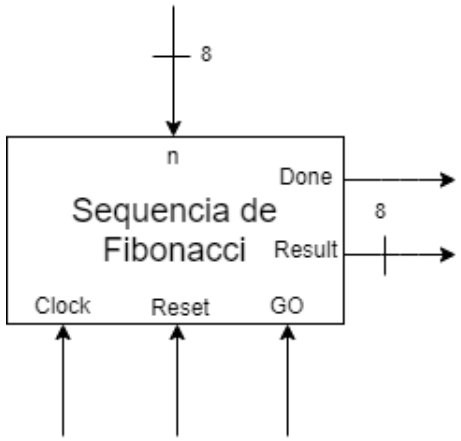


Figure 2: Interface do Circuito

estados, com as suas descrições a seguir:

- S_WAIT: Este estado espera verificar se Go=1 para ir para o próximo que é o estado S_INIT.
- S_RESTART: Este estado verifica sinal de reset e retona ao estado S_INIT .
- S_INIT: É o estado que reseta os registradores e faz a configuração inicial.
- S_LOOP_COND: É o estado que avalia se a posição calculada é igual a posição desejada enquanto for menor vai para para o estado S_ELSE e quando for igual vai pro estado S_DONE.
- S_ELSE: É o estado que faz o próximo calculo.
- S_DONE: É o estado que o calculo está pronto.
- S_WHILE_GO_EQ_1: Enquanto Go=1 permanece nesse estado e quando Go=0 vaipara o estado S_RESTART para começar um novo calculo.

2.2 Algorítimo

```
#include "stdio.h"
void main()
{
  // Declaração de variáveis.
int x, y, temp, i, n;

i = 3;
x = 1;
y = 1;

printf("Digite n: ");
scanf("%d", &n);

while (i <= n){
 temp = x+y;
  x = y;
  y = temp;
  i ++;
};
result = y;
}
```

Entrada: n, especifica o número de Fibonacci a ser calculado, requerido pelo usuário. ída: result, representa o resultado requerido pelo usuário e conclusão do calculo. Temos que x e y são os estados anteriores e estados atuais respectivamente, que armazenam os valores da sequencia. i: é a variável de controle do laço que vai sendo incrementada a cada soma. Com base neste algoritmo, podemos determinar os elementos que farão parte do sistema digital:

- Somadores: Implementa as somas sucessivas.
- Comparadores: Para comparar o numero requeridos da sequência de Fibonacci.
- Multiplexadores: Para para selecionar os sinais de entrada nos registradores.
- Registradores: Para poder armazenar as variáveis necessárias.

2.3 Projeto do Bloco Operativo

A seguir será apresentado a FSMD figura 3 proposta para o projeto, a partir dos requisitos encontrados anteriormente.

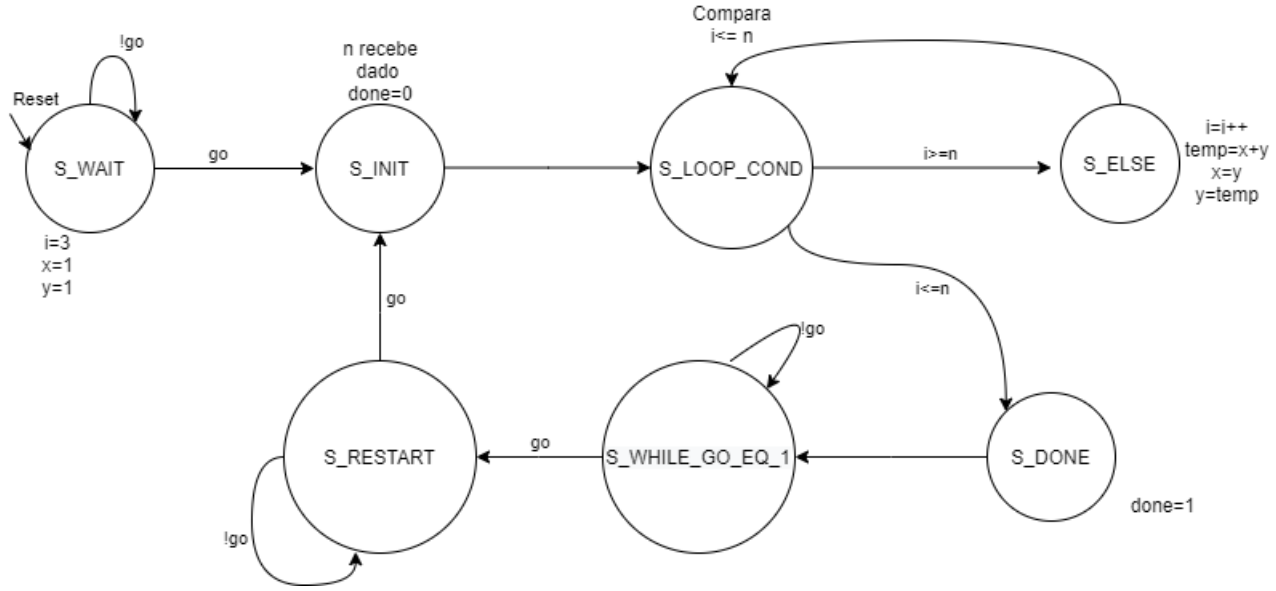


Figure 3: Diagrama da FSMD

Temos a seguir o desenvolvimento de cada bloco do sistema para converter a FSMD de alto nível para a FSM de baixo nível que será implementada no software *QUARTUS*.
Registrador N, atribuição de Dado para variável N, representado na Figura 4.

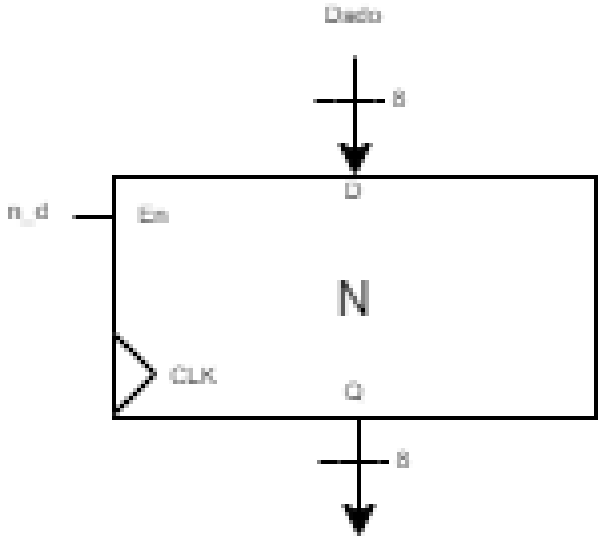


Figure 4: Registrador N

Somador I, faz somas sucessivas do controle de laço, representado na Figura 5.

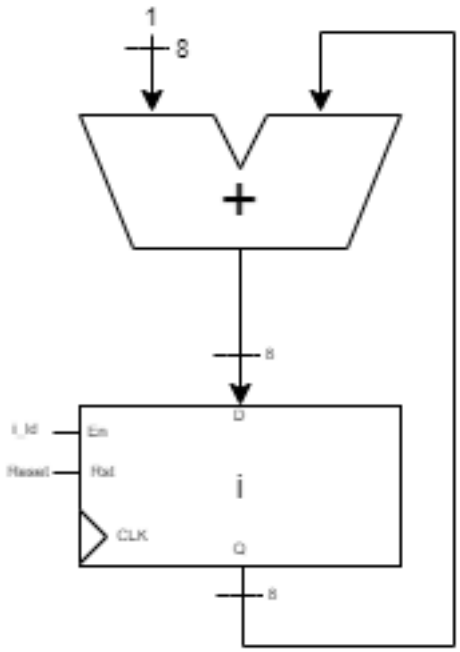


Figure 5: Somador I

Comparador, compara o valor atingido das somas de i com a variável n, representado na Figura 6.

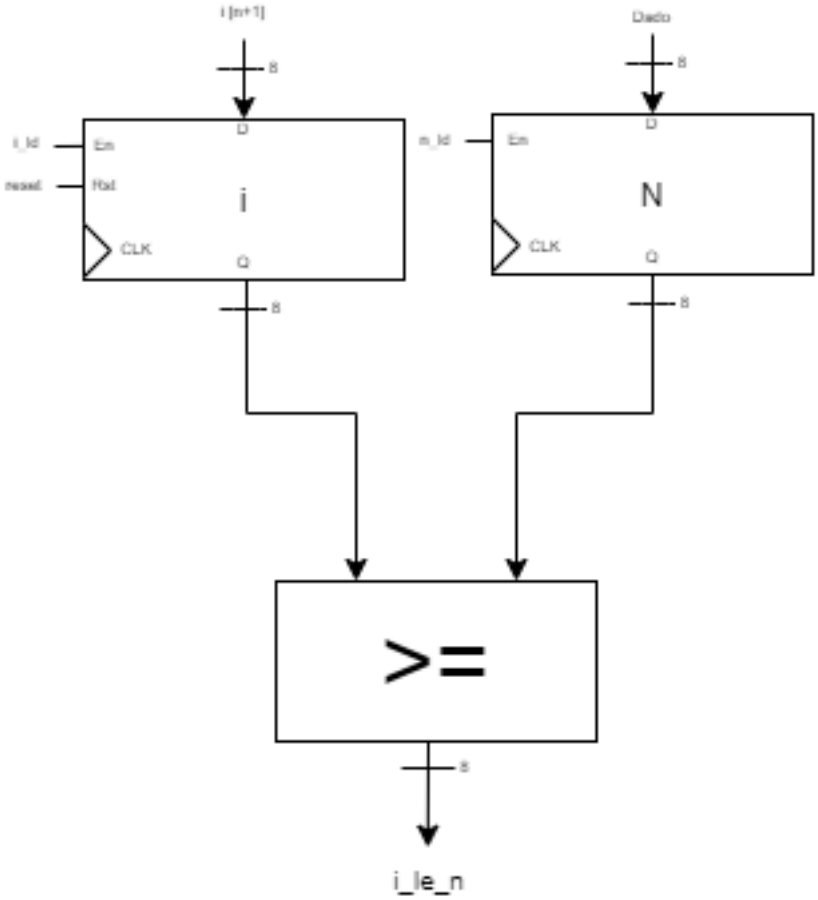


Figure 6: Comparador

Somas X e Y, soma as variáveis seguindo a lógica do algoritmo, representado na Figura 7.

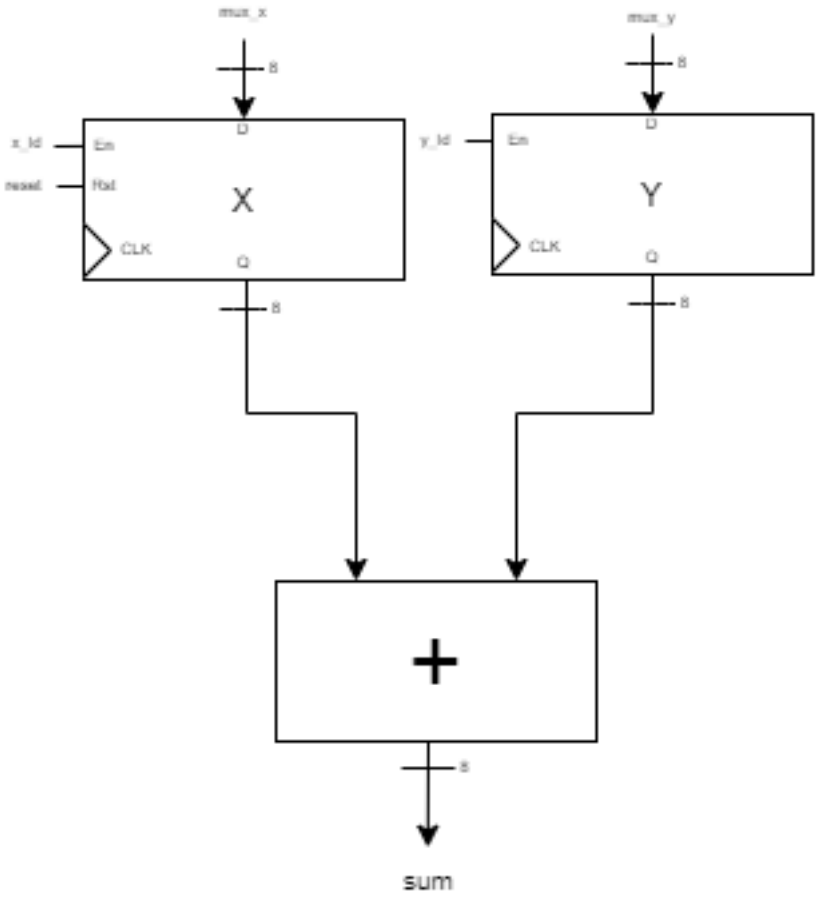


Figure 7: Soma

Multiplexador X, controla a entrada no registrador X, representado na Figura 8.

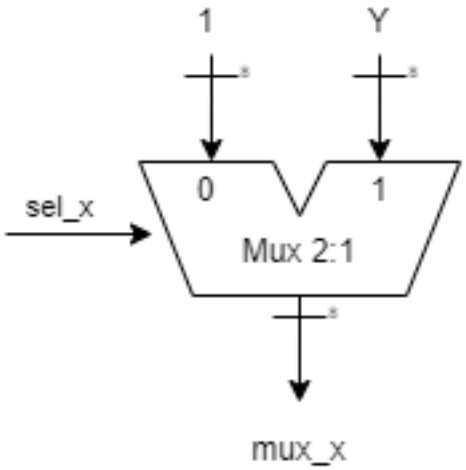


Figure 8: Multiplexador x

Multiplexador Y, controla a entrada no registrador Y, representado na Figura 9.

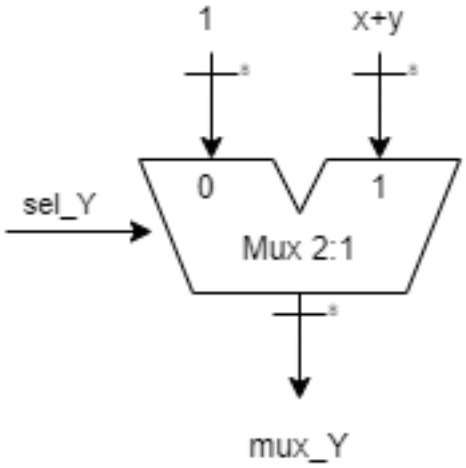


Figure 9: Multiplexador y

Multiplexador I, controla a entrada no registrador I, representado na Figura **??**.

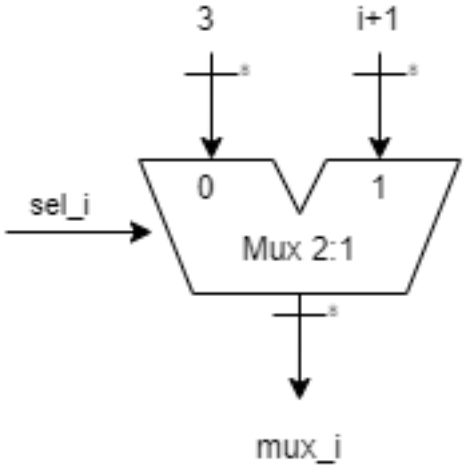


Figure 10: Multiplexador i

Registrador result, atribuição de Y para variável result, representado na Figura 11.

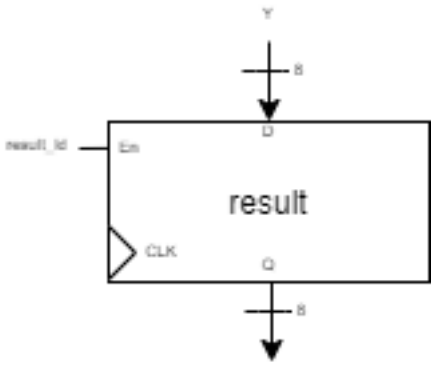


Figure 11: Registrador Result

2.4 Projeto do Bloco de Controle

A partir da FSMD Figura 3 que captura o comportamento do sistema em alto nível, projeta-se a FSM Figura 12 de baixo nível considerando os nomes dos sinais que foram estabelecidos, para representar o bloco de controle.

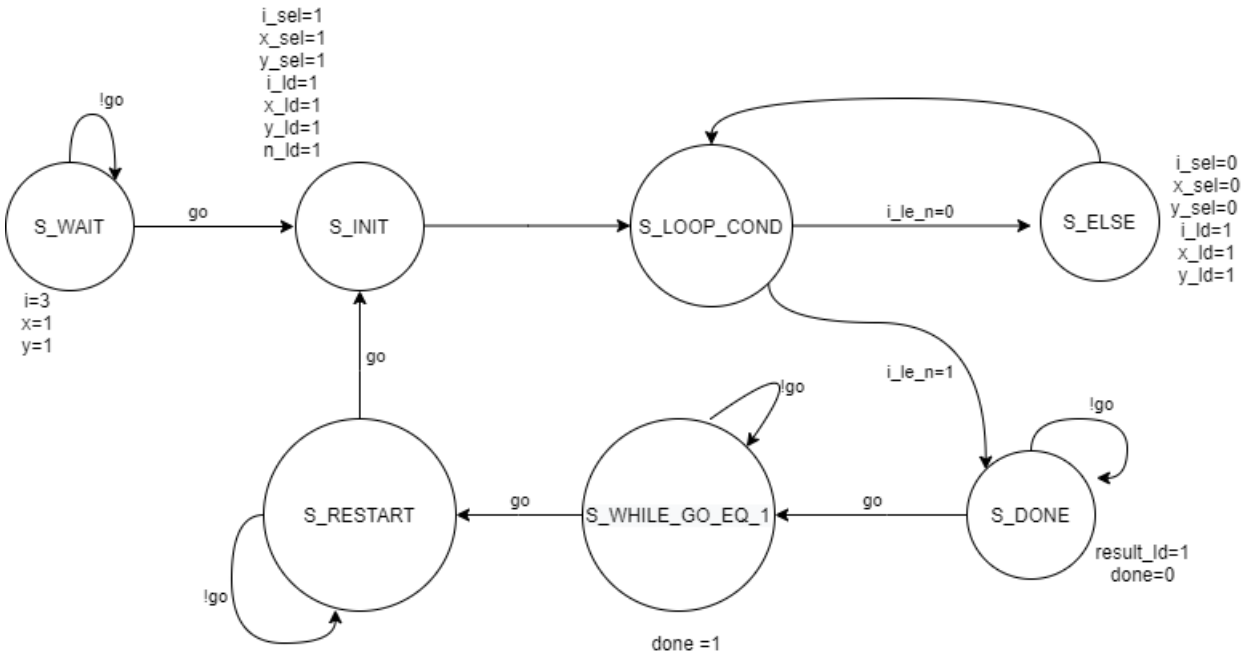


Figure 12: Diagrama da FSM

2.5 Projeto Unidade Aritmética - Cálculo do n-ésimo termo da série de Fibonacci

Com o bloco operativo e o bloco de controle projetados e prontos, pode-se projetar a topologia do sistema completo, ou seja, a Unidade aritmética . O sistema digital será composto de um único bloco de controle (Controller) e um único bloco operativo (Datapath), de modo que o projeto completo implica na integração de ambos os projetos anteriores Figura 13.

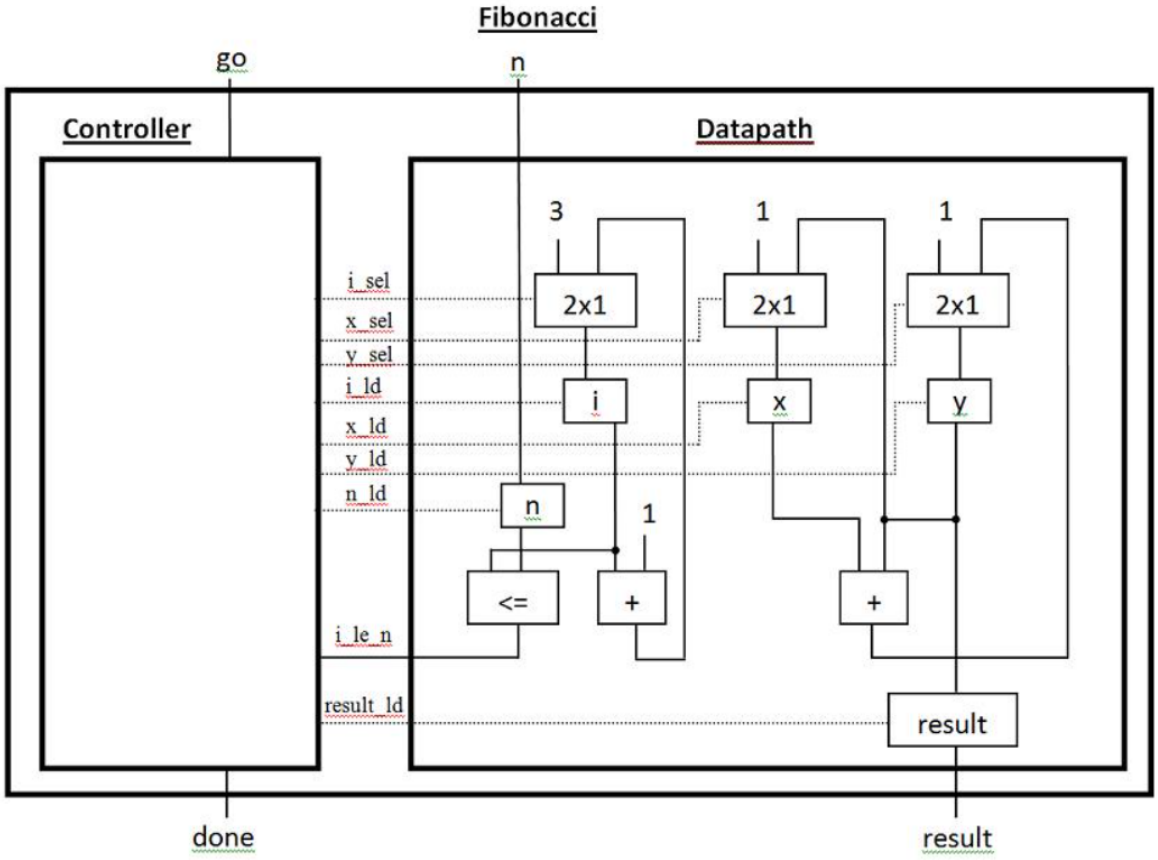


Figure 13: Diagrama de blocos do circuito

3 Desenvolvimento

3.1 Desenvolvimento do Bloco Operativo

A seguir será apresentado os arquivos VHDL dos componentes do bloco operativo e da unidade aritmética.

3.1.1 Registradores (N, result)

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

entity reg is
    generic ( width : positive := 8);

    port
     (clk    : in  std_logic;
    rst    : in  std_logic;
    en     : in  std_logic;
    input  : in  std_logic_vector(width-1 downto 0);
    output : out std_logic_vector(width-1 downto 0));
end reg;

architecture reg1 of reg is
subtype State is std_logic_vector(width-1 downto 0);
---- never change (or delete or add) the internal states ----

    signal nextState, currentState, resetState: State;

begin

resetState <= (others=>'0');


nextState <=  input when EN = '1' else
currentState;

    output <= currentState;


-- memory element (never change it)
    ME: process(clk, rst) is
begin
    if rst='1' then
    currentState <= resetState;
    elsif rising_edge(clk) then
    currentState <= nextState;
end if;
```

```
end process;
end reg1;
```

Flow Status	Successful - Thu Nov 19 08:09:40 2020
Quartus Prime Version	19.1.0 Build 670 09/22/2019 SJ Lite Edition
Revision Name	REG
Top-level Entity Name	reg
Family	Cyclone IV E
Total logic elements	8 / 6,272 (< 1 %)
Total registers	8
Total pins	19 / 92 (21 %)
Total virtual pins	0
Total memory bits	0 / 276,480 (0 %)
Embedded Multiplier 9-bit elements	0 / 30 (0 %)
Total PLLs	0 / 2 (0 %)
Device	EP4CE6E22C6
Timing Models	Final

Figure 14: Análise de área no FPGA do bloco registrador

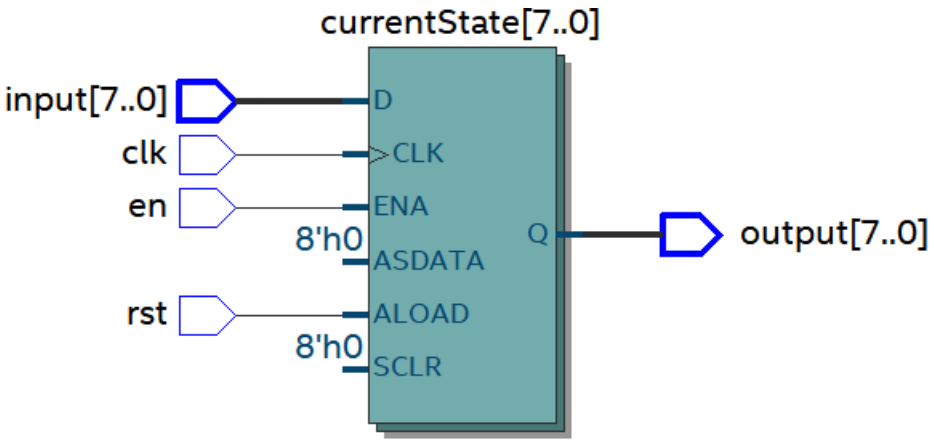


Figure 15: RTL view do blogo registrador

3.1.2 Somador

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

entity add is
  generic
  ( width : positive := 8);

  port
  (
    input1   : in  std_logic_vector(width-1 downto 0);
    input2   : in  std_logic_vector(width-1 downto 0);
    output   : out std_logic_vector(width-1 downto 0)
    );
end add;

architecture add1 of add is
begin
output <= std_logic_vector(unsigned(input1)+ unsigned(input2));
end add1;
```

Flow Status	Successful - Thu Nov 19 07:28:00 2020
Quartus Prime Version	19.1.0 Build 670 09/22/2019 SJ Lite Edition
Revision Name	REG
Top-level Entity Name	add
Family	Cyclone IV E
Total logic elements	8 / 6,272 (< 1 %)
Total registers	0
Total pins	24 / 92 (26 %)
Total virtual pins	0
Total memory bits	0 / 276,480 (0 %)
Embedded Multiplier 9-bit elements	0 / 30 (0 %)
Total PLLs	0 / 2 (0 %)
Device	EP4CE6E22C6
Timing Models	Final

Figure 16: Análise de área no FPGA do bloco Somador

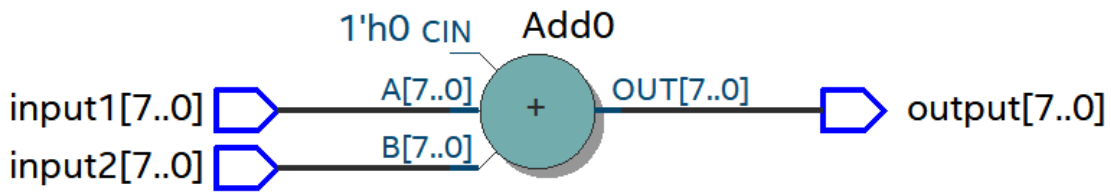


Figure 17: RTL view do blogo Somador

3.1.3 Comparador

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

entity comp is
    generic
     ( width : positive := 8);
    port
     ( input1 : in  std_logic_vector(width-1 downto 0);
     input2 : in  std_logic_vector(width-1 downto 0);
     ne : out std_logic );
end comp;


architecture comp1 of comp is

signal less: std_logic;
begin

 less <= '0' when (unsigned(input1) <= unsigned(input2)) else '1';
 ne  <= less;



end comp1;
```

Flow Status	Successful - Thu Nov 19 07:42:03 2020
Quartus Prime Version	19.1.0 Build 670 09/22/2019 SJ Lite Edition
Revision Name	REG
Top-level Entity Name	comp
Family	Cyclone IV E
Total logic elements	8 / 6,272 (< 1 %)
Total registers	0
Total pins	17 / 92 (18 %)
Total virtual pins	0
Total memory bits	0 / 276,480 (0 %)
Embedded Multiplier 9-bit elements	0 / 30 (0 %)
Total PLLs	0 / 2 (0 %)
Device	EP4CE6E22C6
Timing Models	Final

Figure 18: Análise de área no FPGA do bloco Comparado

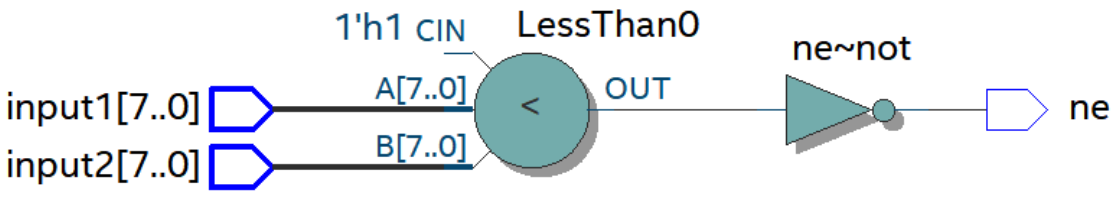


Figure 19: RTL view do blogo Comparador

3.1.4 Multiplexadores (X, Y, I)

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

entity mux_2x1 is
generic ( width :positive :=8);
port
(input0 :in std_logic_vector(width-1 downto 0);
input1 :in std_logic_vector(width-1 downto 0);
sel :in std_logic;
output :out std_logic_vector(width-1 downto 0));
end entity;

architecture mux_2x11 of mux_2x1 is
begin

output <=  input0 when sel='0' else
input1;

end mux_2x11;
```

Flow Status	Successful - Thu Nov 19 10:38:09 2020
Quartus Prime Version	19.1.0 Build 670 09/22/2019 SJ Lite Edition
Revision Name	REG
Top-level Entity Name	mux_2x1
Family	Cyclone IV E
Total logic elements	8 / 6,272 (< 1 %)
Total registers	0
Total pins	25 / 92 (27 %)
Total virtual pins	0
Total memory bits	0 / 276,480 (0 %)
Embedded Multiplier 9-bit elements	0 / 30 (0 %)
Total PLLs	0 / 2 (0 %)
Device	EP4CE6E22C6
Timing Models	Final

Figure 20: Análise de área no FPGA do bloco Multiplexador

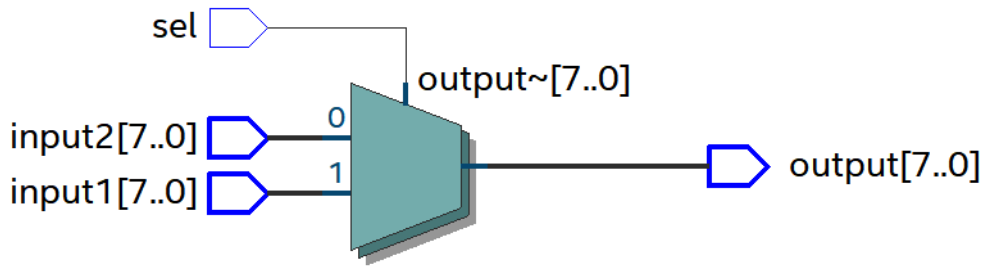


Figure 21: RTL view do blogo Multiplexador

3.2 Desenvolvimento do Bloco de Controle

```
 library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all; -- biblioteca que premite trabalhar com operações aritimeticas

entity fib_control is
port
(clk :in std_logic;
rst :in std_logic;
go :in std_logic;
done:out std_logic;
i_le_n :in std_logic;
i_sel :out std_logic;
x_sel :out std_logic;
y_sel :out std_logic;
i_ld :out std_logic;
x_ld :out std_logic;
y_ld :out std_logic;
n_ld :out std_logic;
result_ld :out std_logic);
end fib_control;


architecture FSM of fib_control is

type STATE_TYPE is (S_WAIT, S_RESTART, S_INIT, S_LOOP_COND, S_ELSE, S_DONE, S_WHILE_GO_E
signal state, st : STATE_TYPE; --state é o valor armazenado no reg e next_state é o prox
--sinais que armazenam o estados
begin
process(st,go,i_le_n )
begin

state <= st; --state =  proximo estado
case st is

when S_WAIT =>
--estado espera verifica se go=1 para ir para o proximo estado
```

```
if(go='1') then
state <= S_INIT;
end if;

when s_RESTART =>

if(go = '1') then

state <= S_INIT;
end if;

when S_INIT =>
state <= S_LOOP_COND;


when S_LOOP_COND =>
if(i_le_n='1') then

state <= S_DONE;
else
state <= S_ELSE;
end if;


when S_ELSE =>
state <= S_LOOP_COND;


when S_DONE =>
if(go='0') then
state <= S_WHILE_GO_EQ_1;
end if;


when S_WHILE_GO_EQ_1 =>

if(go='0') then
state <= S_RESTART;
--volata para o 1 estado
   end if;
when others => null;

   end case;
end process;

-- memory element (sequential)
process(clk, rst) is
begin
```

```
if(rst = '1') then --rst assicrono não depende do clk
st <= S_WAIT;
elsif(clk'event and clk = '1') then -- se não e tiver evento de clk de borda de subida
       st <= state;

end if;
end process;

--começa logica de saida
i_sel <= '1'       when st = S_INIT else '0';
x_sel <= '1'       when st = S_INIT else '0';
y_sel <= '1'       when st = S_INIT else '0';
i_ld <= '1'       when st = S_INIT or st = S_ELSE else '0';
x_ld <= '1'       when st = S_INIT or st = S_ELSE else '0';
y_ld <= '1'       when st = S_INIT or st = S_ELSE else '0';
n_ld <= '1'       when st = S_INIT  else '0';
result_ld <= '1' when st = S_DONE else '0';
done <= '1'       when st = S_DONE or st = S_WHILE_GO_EQ_1 else '0';


end FSM;
```

Flow Status	Successful - Thu Nov 19 11:31:21 2020
Quartus Prime Version	19.1.0 Build 670 09/22/2019 SJ Lite Edition
Revision Name	REG
Top-level Entity Name	fib_control
Family	Cyclone IV E
Total logic elements	14 / 6,272 (< 1 %)
Total registers	7
Total pins	13 / 92 (14 %)
Total virtual pins	0
Total memory bits	0 / 276,480 (0 %)
Embedded Multiplier 9-bit elements	0 / 30 (0 %)
Total PLLs	0 / 2 (0 %)
Device	EP4CE6E22C6
Timing Models	Final

Figure 22: Análise de área no FPGA do bloco de Controle

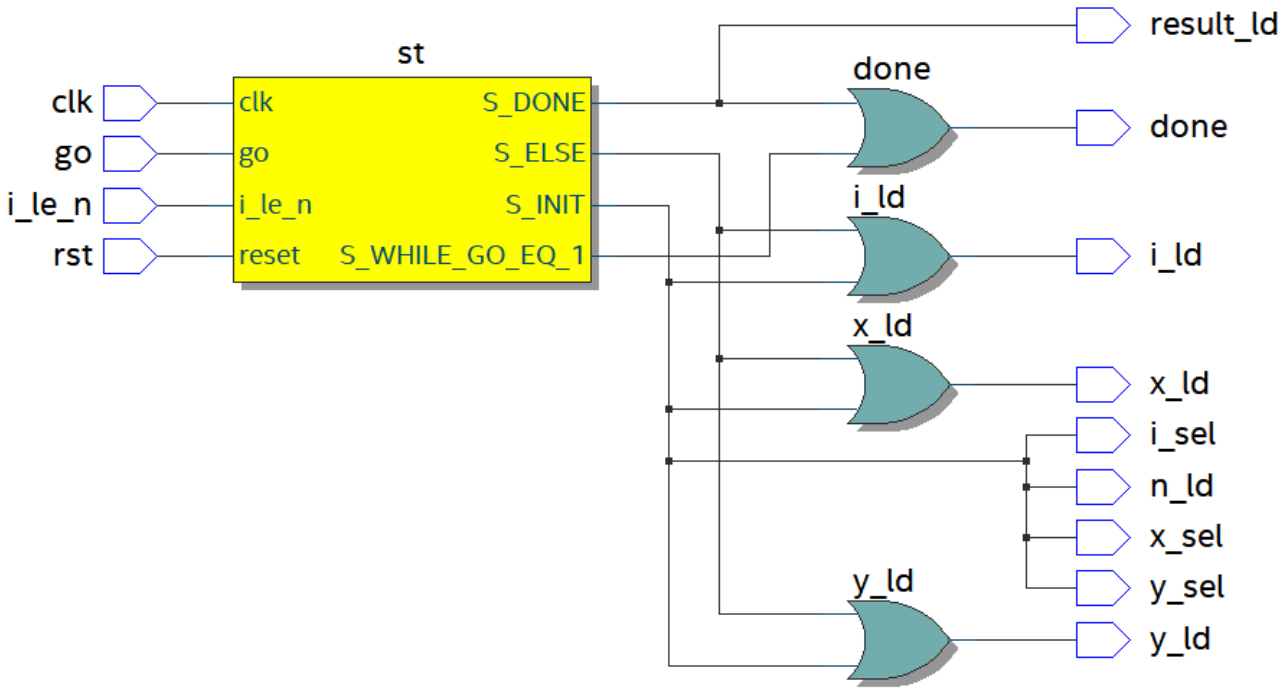


Figure 23: RTL view do blogo De controle

3.3 Desenvolvimento do Bloco Operativo

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

entity fib_datapath is
     generic
     ( width :positive := 8);
port
         (clk :in std_logic;
        rst :in std_logic;
        i_sel :in std_logic;
        x_sel :in std_logic;
        y_sel :in std_logic;
        i_ld :in std_logic;
        x_ld :in std_logic;
        y_ld :in std_logic;
        n_ld :in std_logic;
        result_ld :in std_logic;
        i_le_n :out std_logic;
        n :in std_logic_vector(width-1 downto 0);
        output  :out std_logic_vector(width-1 downto 0));
end fib_datapath;


architecture structure of fib_datapath is

COMPONENT mux_2x1 is

         generic ( width :positive :=8);

         port
         (input0 :in std_logic_vector(width-1 downto 0);
         input1 :in std_logic_vector(width-1 downto 0);
         sel :in std_logic;
         output :out std_logic_vector(width-1 downto 0));
END COMPONENT;

COMPONENT reg is

            generic ( width : positive := 8);

            port
            (clk    : in  std_logic;
            rst    : in  std_logic;
            en     : in  std_logic;
            input  : in  std_logic_vector(width-1 downto 0);
            output : out std_logic_vector(width-1 downto 0));
```

```
END COMPONENT;


COMPONENT add is

          generic ( width : positive := 8);

          port
          (
            input1   : in  std_logic_vector(width-1 downto 0);
            input2   : in  std_logic_vector(width-1 downto 0);
            output   : out std_logic_vector(width-1 downto 0)
            );
END COMPONENT;

COMPONENT comp is

            generic ( width : positive := 8);

            port
         ( input1 : in  std_logic_vector(width-1 downto 0);
             input2 : in  std_logic_vector(width-1 downto 0);
             ne : out std_logic );

END COMPONENT;


signal i_mux_out, x_mux_out, y_mux_out :std_logic_vector(width-1 downto 0);
signal i_reg_out, x_reg_out, y_reg_out, n_reg_out :std_logic_vector(width-1 downto 0);
signal add_out_1, add_out_2  :std_logic_vector(width-1 downto 0);
begin

l0 :entity work.mux_2x1 generic map ( width => width)
         port map
         (input0  => add_out_1,
         input1  => std_logic_vector(to_unsigned(3, width)),
         sel  => i_sel,
         output  => i_mux_out);

l1 :entity work.mux_2x1 generic map ( width => width)
         port map
         (input0  => y_reg_out,
         input1  => std_logic_vector(to_unsigned(1, width)),
         sel  => x_sel,
         output  => x_mux_out);

l2 :entity work.mux_2x1 generic map ( width => width)
         port map
         (input0  => add_out_2,
```

```
        input1  => std_logic_vector(to_unsigned(1, width)),
        sel  => y_sel,
        output  => y_mux_out);

l3 : entity work.reg generic map( width => width)
               port map
        (  clk    => clk,
        rst    => rst,
        en     => i_ld,
        input  => i_mux_out,
        output => i_reg_out);

l4 : entity work.reg generic map( width => width)
               port map
        (clk    => clk,
        rst    => rst,
        en     => x_ld,
        input  => x_mux_out,
        output => x_reg_out);

l5 : entity work.reg generic map( width => width)
               port map
        (clk    => clk,
        rst    => rst,
        en     => y_ld,
        input  => y_mux_out,
        output => y_reg_out);

l6 : entity work.reg generic map( width => width)
               port map
        (clk    => clk,
        rst    => rst,
        en     => n_ld,
        input  => n,
        output => n_reg_out);

l7: entity work.comp generic map ( width => width)
               port map
        ( input1 => i_reg_out,
                input2 => n_reg_out,
        ne  => i_le_n);

l8 : entity work.add  generic map ( width => width)
               port map
        ( input1 => i_reg_out,
        input2  => std_logic_vector(to_unsigned(1, width)),
        output  => add_out_1);

l9 : entity work.add generic map ( width => width)
```

```
                port map
          ( input1 => x_reg_out,
          input2  => y_reg_out,
          output  => add_out_2);

l10: entity work.reg generic map( width => width)
                port map
          ( clk    => clk,
          rst    => rst,
          en     => result_ld,
          input  => y_reg_out,
          output => output);

end structure;
```

Flow Status	Successful - Thu Nov 19 12:11:43 2020
Quartus Prime Version	19.1.0 Build 670 09/22/2019 SJ Lite Edition
Revision Name	REG
Top-level Entity Name	fib_datapath
Family	Cyclone IV E
Total logic elements	41 / 6,272 (< 1 %)
Total registers	40
Total pins	27 / 92 (29 %)
Total virtual pins	0
Total memory bits	0 / 276,480 (0 %)
Embedded Multiplier 9-bit elements	0 / 30 (0 %)
Total PLLs	0 / 2 (0 %)
Device	EP4CE6E22C6
Timing Models	Final

Figure 24: Análise de área no FPGA do bloco de Controle

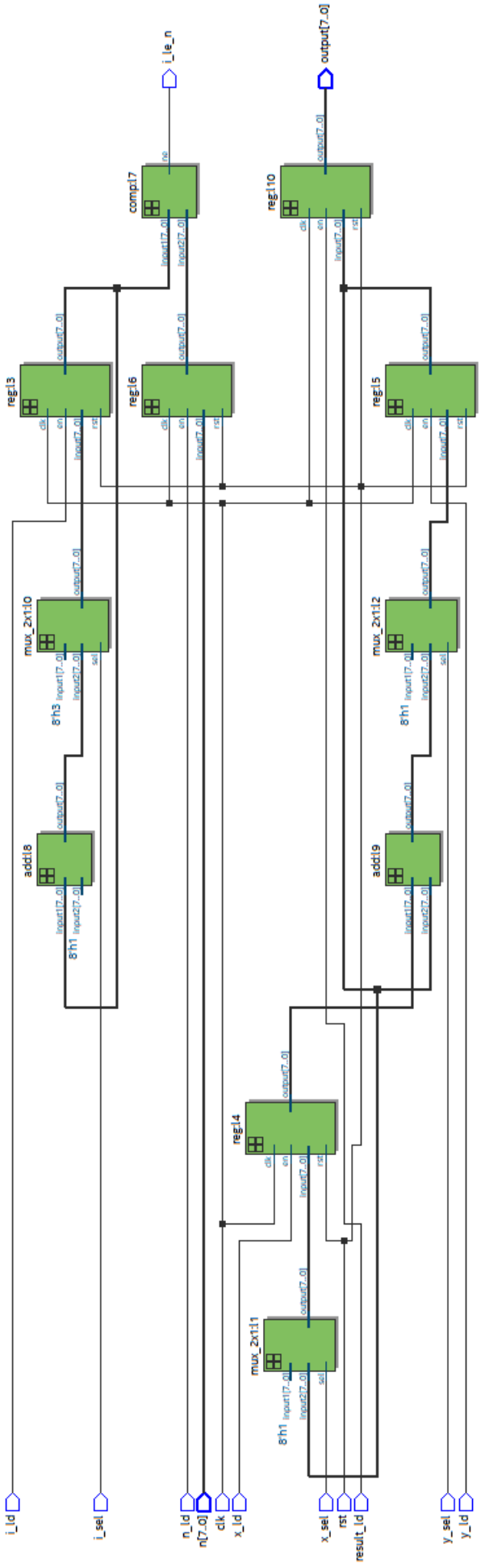


Figure 25: RTL view do bloco Operativo

3.4 Desenvolvimento da Unidade Aritmética

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all; --permite usar bibliotecas do tipo tipo UNSIGNED

entity fib_top is
generic
( width: positive:= 8);
port
(  rst, clk : in std_logic;
n : in std_logic_vector(width-1 downto 0); --numero da sequencia
go : in std_logic;
done : out std_logic; --acabou de executar
output : out std_logic_vector(width-1 downto 0) );--valor do numero na sequencia

end entity;




architecture top_lv of fib_top is

COMPONENT fib_control is
port
         (clk :in std_logic;
         rst :in std_logic;
         go :in std_logic;
         done:out std_logic;
         i_le_n :in std_logic;
         i_sel :out std_logic;
         x_sel :out std_logic;
         y_sel :out std_logic;
         i_ld :out std_logic;
         x_ld :out std_logic;
         y_ld :out std_logic;
         n_ld :out std_logic;
         result_ld :out std_logic);
END COMPONENT;

COMPONENT fib_datapath is

         generic ( width :positive := 8);

         port
         (clk :in std_logic;
         rst :in std_logic;
         i_sel :in std_logic;--sina de seleção do mux
```

```
        x_sel :in std_logic;--sina de seleção do mux
        y_sel :in std_logic;--sina de seleção do mux
        i_ld :in std_logic;
        x_ld :in std_logic;
        y_ld :in std_logic;
        n_ld :in std_logic;
        result_ld :in std_logic;
        i_le_n :out std_logic;
        n :in std_logic_vector(width-1 downto 0); --n termo
        output  :out std_logic_vector(width-1 downto 0));

END COMPONENT;

signal i_sel  :std_logic;
signal x_sel  :std_logic;
signal y_sel  :std_logic;
signal i_ld  :std_logic;
signal x_ld  :std_logic;
signal y_ld  :std_logic;
signal n_ld  :std_logic;
signal result_ld :std_logic;
signal i_le_n  :std_logic;


begin

controle : entity work.fib_control port map
        (  clk       => clk,
        rst       => rst,
        go        => go,
        done      => done,
        i_sel   => i_sel,
        x_sel     => x_sel,
        y_sel     => y_sel,
        i_ld   => i_ld,
        x_ld      => x_ld,
        y_ld      => y_ld,
        n_ld   => n_ld,
        result_ld => result_ld,
        i_le_n    => i_le_n);

    data: entity work.fib_datapath generic map (width => width) port map
            (  clk       => clk,
                    rst       => rst,
                    i_sel   => i_sel,
        x_sel     => x_sel,
        y_sel     => y_sel,
        i_ld   => i_ld,
        x_ld      => x_ld,
```

```
        y_ld      => y_ld,
        n_ld   => n_ld,
        result_ld => result_ld,
        i_le_n    => i_le_n,
                  n         => n,
                output    => output);
end top_lv;
```

Flow Status	Successful - Thu Nov 19 12:34:00 2020
Quartus Prime Version	19.1.0 Build 670 09/22/2019 SJ Lite Edition
Revision Name	fib_v3
Top-level Entity Name	fib_top
Family	Cyclone IV E
Device	EP4CE55F29C6
Timing Models	Final
Total logic elements	54 / 55,856 (< 1 %)
Total registers	47
Total pins	20 / 375 (5 %)
Total virtual pins	0
Total memory bits	0 / 2,396,160 (0 %)
Embedded Multiplier 9-bit elements	0 / 308 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 26: Análise de área no FPGA da Unidade Aritmética

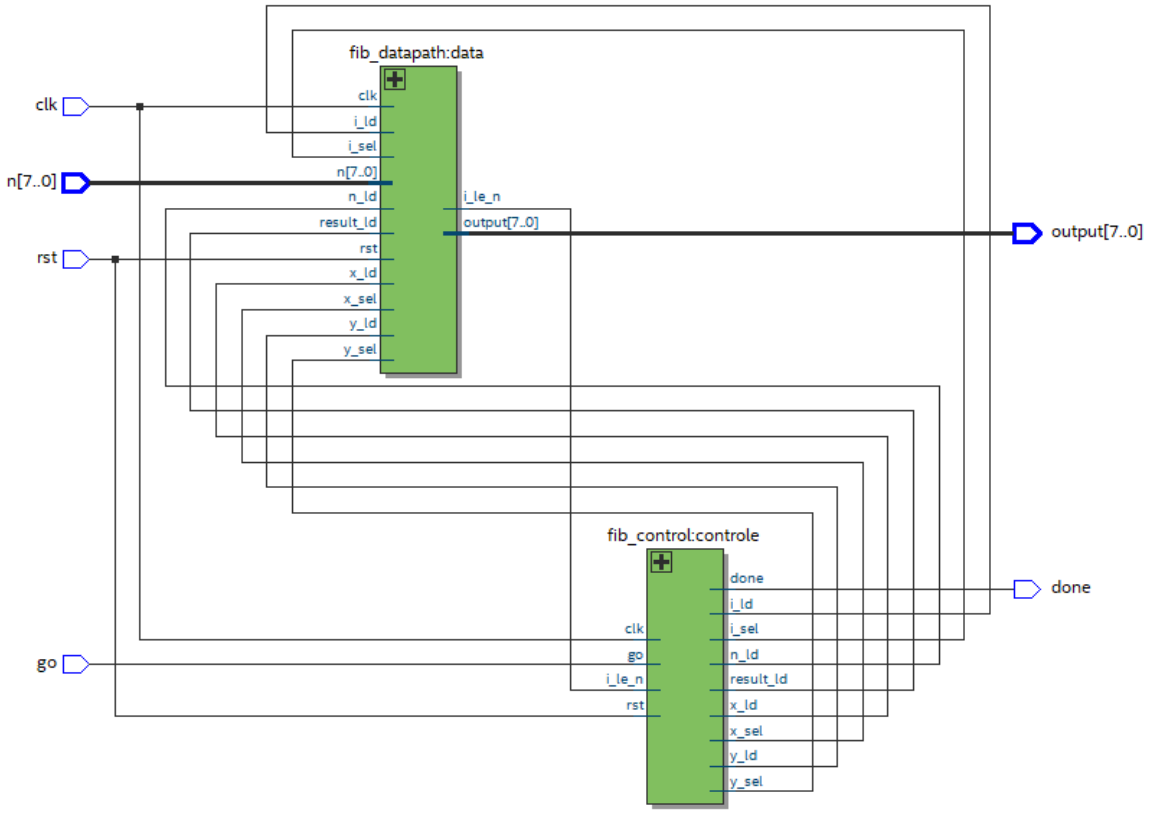


Figure 27: RTL view da Unidade Aritmética

4 Testes e Validação

4.1 Validação do Bloco Operativo

Será apresentado a seguir as simulações dos componentes do Bloco Operativo da unidade aritmética, com a finalidade de validar o seu funcionamento.

4.1.1 Registrador (N, Result)

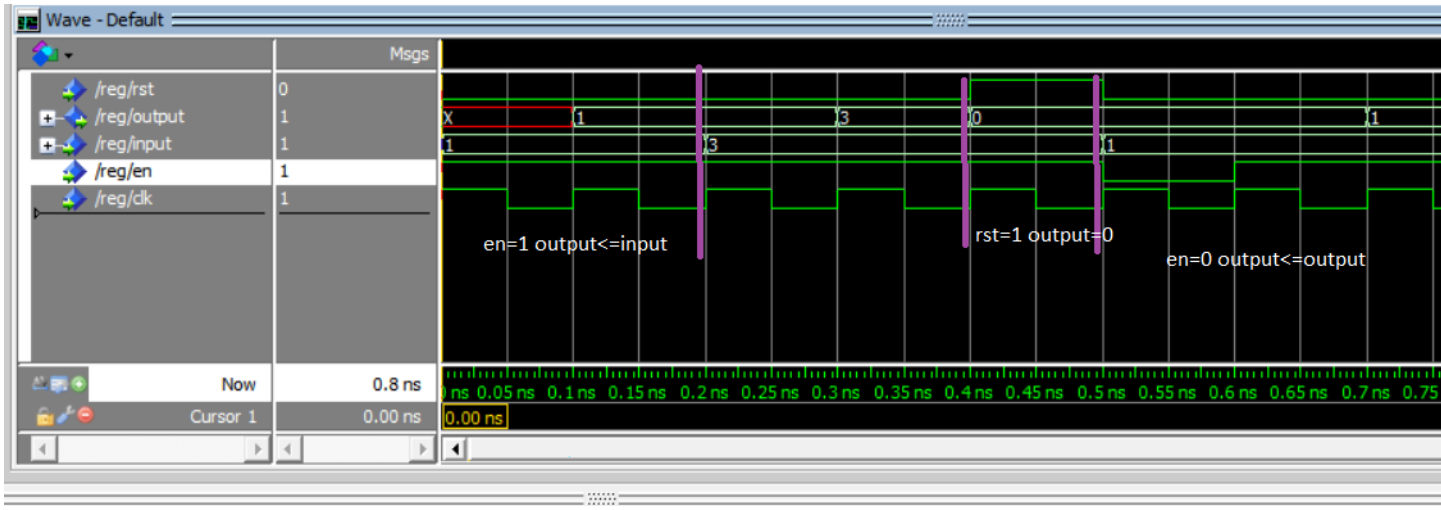


Figure 28: Teste e Validação do Registrador

4.1.2 Somador I

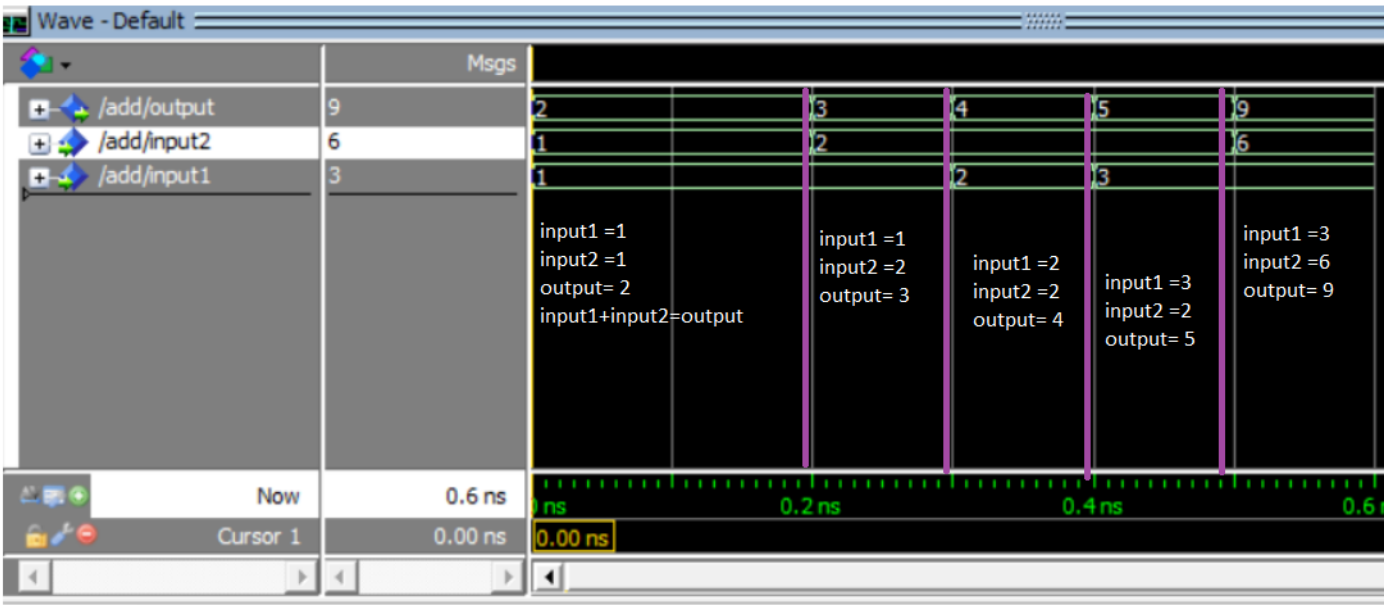


Figure 29: Teste e Validação do Somador

4.1.3 Comparador

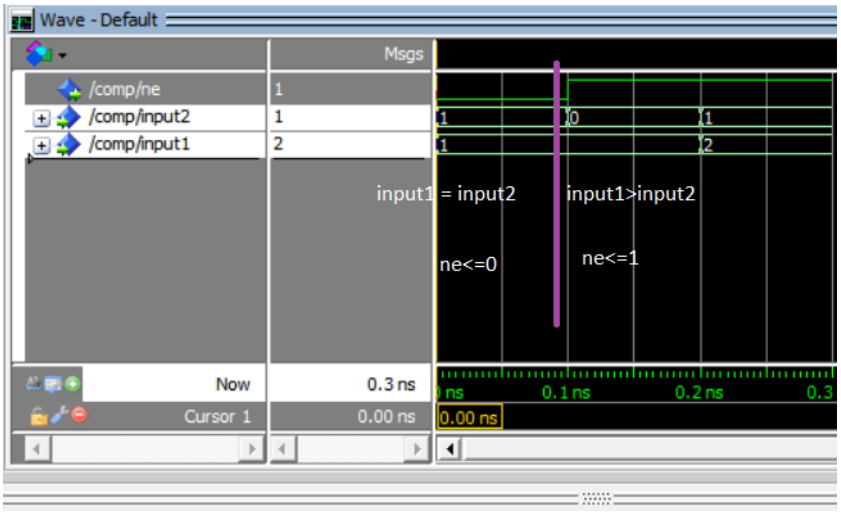


Figure 30: Teste e Validação do Comparador

4.1.4 Multiplicadores (X, Y, I)

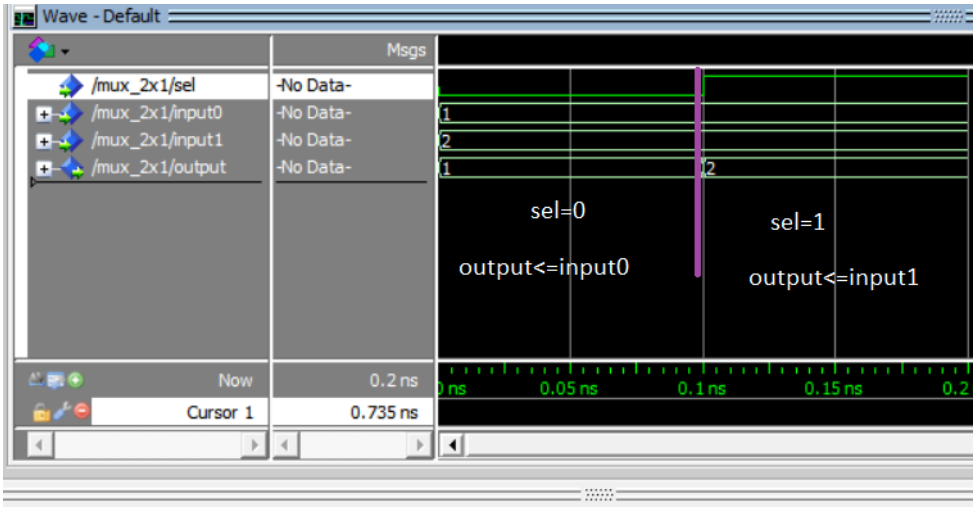


Figure 31: Teste e Validação do Multiplexador

5 Validação do Bloco de Controle

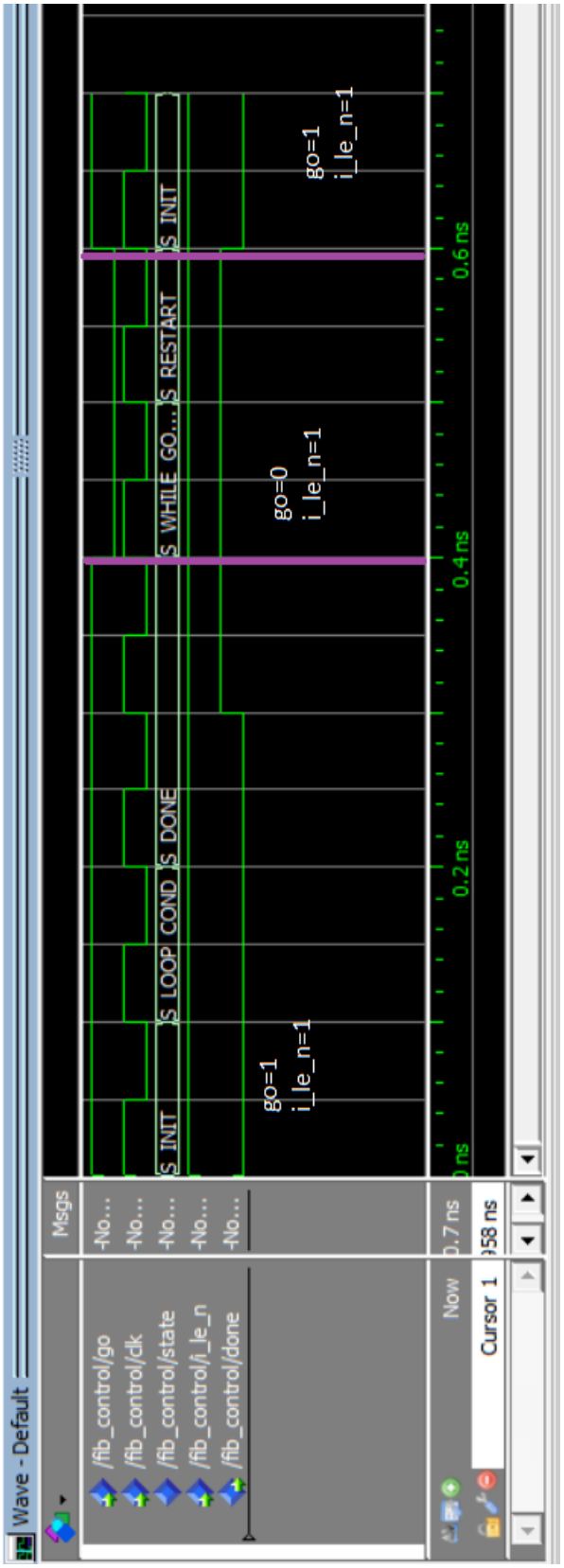
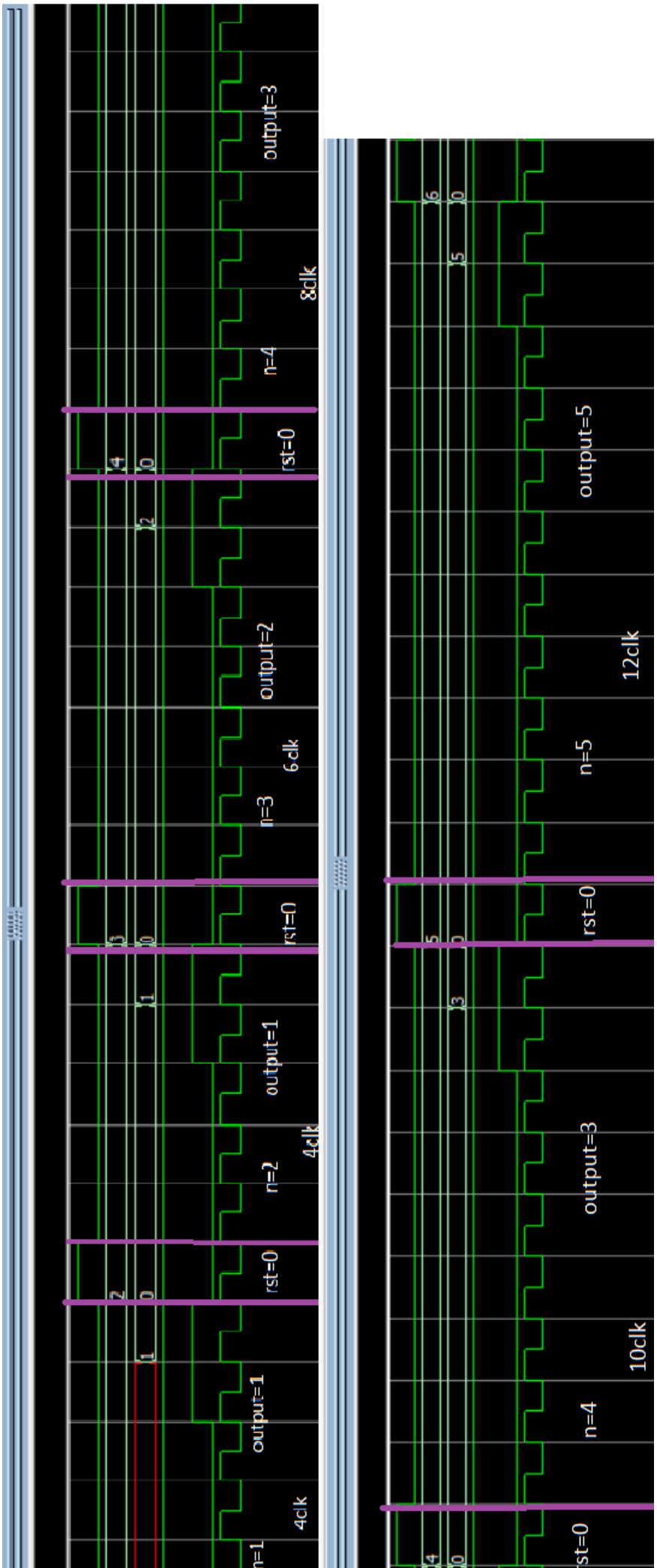


Figure 32: Teste e Validação do Bloco de Conrole

6 Validação do cálculo do n-esimo termo da série de Fibonacci


output=3
8clk
n=4
rst=0
output=2
6clk
n=3
rst=0
output=1
4clk
n=2
rst=0
output=1
4clk
output=5
12clk
n=5
rst=0
output=3
10clk
n=4
rst=0

7 Validação do cálculo do n-esimo termo da série de Fibonacci com testbench

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

entity fib_tb is
end fib_tb;

architecture TB of fib_tb is

constant TEST_WIDTH : positive :=8;

signal clk : std_logic := '0';
signal rst : std_logic := '0';
signal go : std_logic := '0';
signal n  : std_logic_vector(TEST_WIDTH-1 downto 0);
signal done : std_logic;
signal output : std_logic_vector(TEST_WIDTH-1 downto 0);

begin

UUT :entity work.fib_top
generic map(width => TEST_WIDTH)
port map (

clk  => clk,
rst => rst,
go   => go,
n  => n,
output  => output,
done  => done
);

clk <= not clk after 10 ns;

process
begin

rst  <= '1';
go   <= '0';
--done <= '0';
n <= (others => '0');

wait for 100 ns;

rst <= '0';
```

```
for i in 0 to 2 loop
wait until rising_edge(clk);
end loop;

--Checking for n = 1
n <= std_logic_vector(to_unsigned(1, TEST_WIDTH));
go <= '1';
wait until rising_edge(clk);
go <= '0';
wait until done = '1';
assert (output = std_logic_vector(to_unsigned(1, TEST_WIDTH)))
report "Value of series is not correct";


wait for 100ns;


--Checking for n = 2
n <= std_logic_vector(to_unsigned(2, TEST_WIDTH));
go <= '1';
wait until rising_edge(clk);
go <= '0';
wait until done = '1';
assert (output = std_logic_vector(to_unsigned(1, TEST_WIDTH)))
report "Value of series is not correct";

wait for 100ns;

--Checking for n = 3
n <= std_logic_vector(to_unsigned(3, TEST_WIDTH));
go <= '1';
wait until rising_edge(clk);
go <= '0';
wait until done = '1';
assert (output = std_logic_vector(to_unsigned(2, TEST_WIDTH)))
report "Value of series is not correct";

wait for 100ns;


--Checking for n = 4
n <= std_logic_vector(to_unsigned(4, TEST_WIDTH));
go <= '1';
wait until rising_edge(clk);
go <= '0';
wait until done = '1';
assert (output = std_logic_vector(to_unsigned(3, TEST_WIDTH)))
report "Value of series is not correct";
```

```
wait for 100ns;



--Checking for n = 5
n <= std_logic_vector(to_unsigned(5, TEST_WIDTH));
go <= '1';
wait until rising_edge(clk);
go <= '0';
wait until done = '1';
assert (output = std_logic_vector(to_unsigned(5, TEST_WIDTH)))
report "Value of series is not correct";

wait for 100ns;
--Checking for n = 8
n <= std_logic_vector(to_unsigned(8, TEST_WIDTH));
go <= '1';
wait until rising_edge(clk);
go <= '0';
wait until done = '1';
assert (output = std_logic_vector(to_unsigned(21, TEST_WIDTH)))
report "Value of series is not correct";


wait for 100ns;



wait;
end process;
end TB;
```

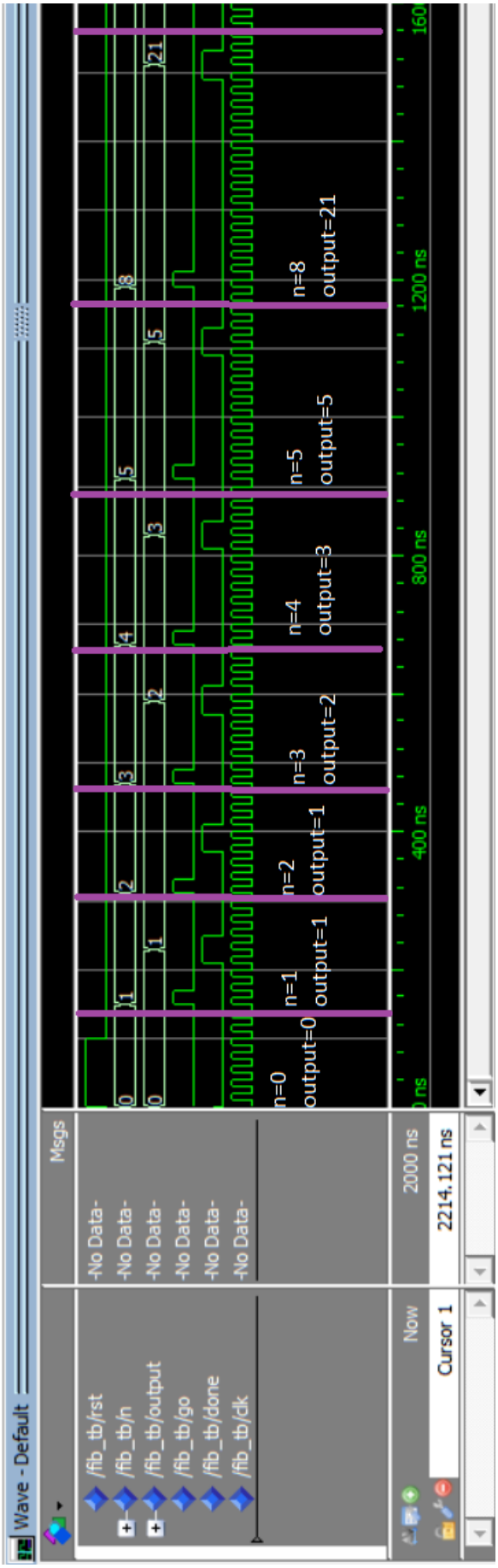


Figure 34: cálculo do n-esimo termo da série de Fibonacci com testbench

8 Conclusão

O projeto funciona e foi implementado comforme as orientações da disciplina de Sistemas Digitais INE 5406-02208B (20201)

9 Referências

- VAHID, Frank. Sistemas Digitais: projeto, otimização e HDLs. Porto Alegre: Bookman, 2008. ISBN 978-85-7780-190-9

- BROWN, Stephen; VRANESIC, Zvonko. Fundamentals of Digital Logic with VHDL Design. Third Edition. Boston, MA.: McGraw-Hill, 2009. ISBN 978–0–07–352953–0

- PEDRONI, Volnei. Circuit Design with VHDL. Third edition. Cambridge, MA: MIT Press, 2020. ISBN 978-0-262-04264-2

- ASHENDEN, Peter J. The Designer's Guide to VHDL.8232;Third Edition. Burlington, MA .: Morgan Kaufmann Publishers , 2008. ISBN: 978-0-12-088785-98232; Disponível em: https://www.sciencedirect.com/book/9780120887859/the-designers-guide-to-vhdl