

# Trabalho Final

raiz quadrada por busca binária

# Algoritmo

```
def sqrt_binary_search(x):

    start = 0
    end = x
    cont = 0

    while start <= end:

        mid = (start+end)//2
        cont = cont + 1

        if x == mid * mid:
            break
        elif x < mid * mid:
            end = mid - 1
        elif x > mid * mid:
            start = mid + 1

    print("cont: " + str(cont))
    return end

while 1:
    print("resultado: " + str(sqrt_binary_search(int(input()))))
```

Num = 64, start = 0, end = 64

Mid = 32, como  $32 * 32 > 64$  end = 31

Mid = 15,  $15 * 15 > 64$  end = 14

Mid = 7,  $7 * 7 < 64$  start = 8 end = 14

Mid =  $(8+14)/2 = 11$ ,  $11 * 11 > 64$  end = 10

Mid = 9,  $9 * 9 > 64$ , end = 8

Como  $8 * 8 == 64$ , saímos do while retornando o valor de end, que no momento está em 8, finalizando assim a operação.

# Algoritmo

Num = 10, start = 0, end = 10

Mid = 5, como  $5 * 5 > 10$  end = 4

Mid =  $2,2 * 2 < 10$  start = 3

Mid = 3,  $3 * 3 < 10$  start = 4 end = 3

Como start > end, devemos sair do programa e retornar end que é 3 e seria o valor aproximado para raiz de 10.

Mostrando desempenho do algoritmo em casos que possuem raízes não exatas.

```
def sqrt_binary_search(x):

    start = 0
    end = x
    cont = 0

    while start <= end:

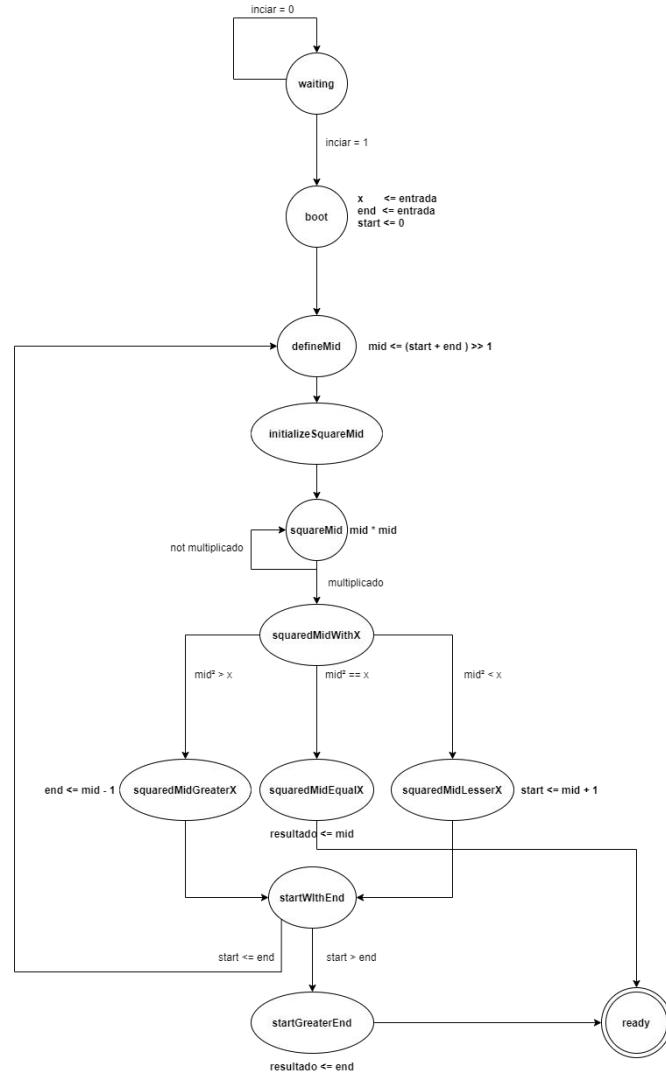
        mid = (start+end)//2
        cont = cont + 1

        if x == mid * mid:
            break
        elif x < mid * mid:
            end = mid - 1
        elif x > mid * mid:
            start = mid + 1

    print("cont: " + str(cont))
    return end

while 1:
    print("resultado: " + str(sqrt_binary_search(int(input())))))
```

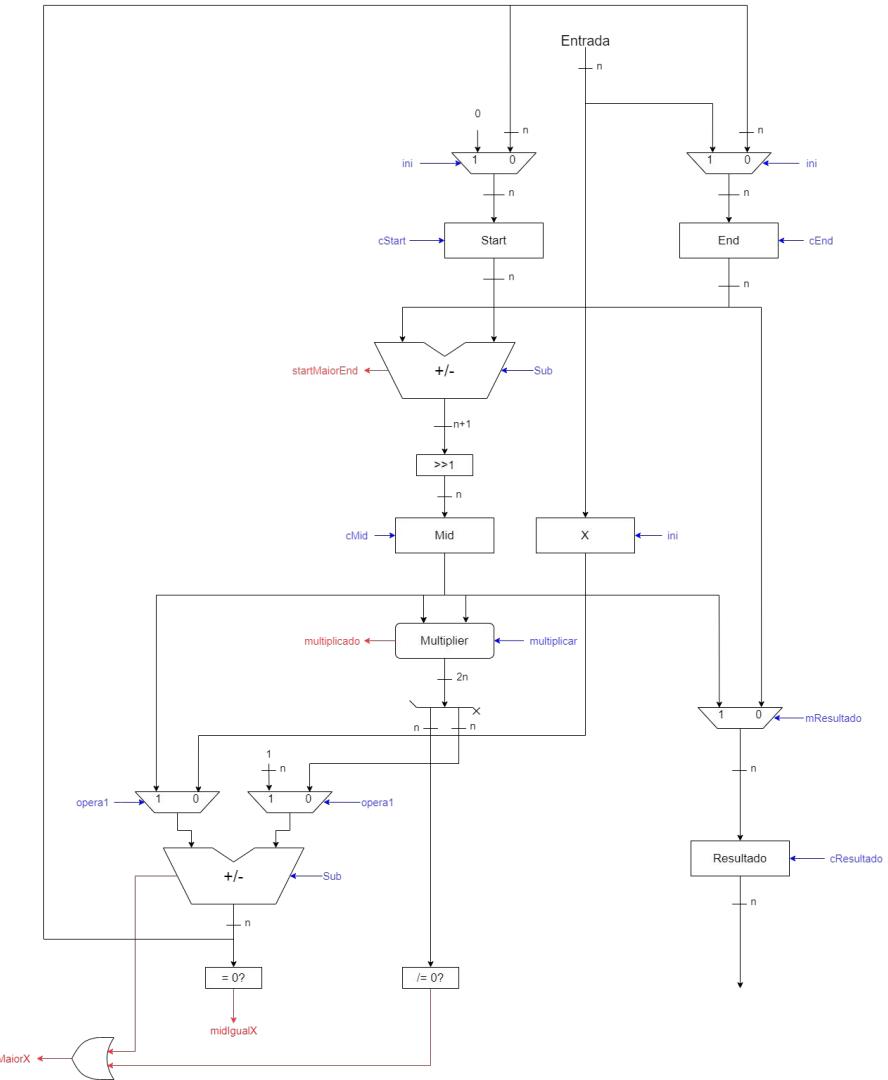
# Máquina de Estados



# Máquina de Estados

# Arquiteturas (diagramas de blocos do BO)

Transformando o algoritmo em arquitetura chegamos ao seguinte:



# VHDS - BC

```

when squaredMidWithX =>
    if midMaiorX = '1' then
        nextState <= squaredMidGreaterX;
    elsif mid igualX = '1' then
        nextState <= squaredMidEqualX;
    else
        nextState <= squaredMidLesserX;
    end if;
when squaredMidGreaterX =>
    nextState <= startWithEnd;
when squaredMidEqualX =>
    nextState <= ready;
when squaredMidLesserX =>
    nextState <= startWithEnd;
when startWithEnd =>
    if startMaiorEnd = '1' then
        nextState <= startGreaterEnd;
    else
        nextState <= defineMid;
    end if;
when startGreaterEnd =>
    nextState <= ready;
when others =>
    nextState <= ready;
end case;
end process;
OutputSignals : process (state)
begin
    case state is
        when waiting =>
            ini      <= '0';
            cStart   <= '1';
            cEnd     <= '1';
            cMid     <= '1';
            cResultado <= '1';
            sub      <= '1';
            multiplicar <= '0';
            mResultado <= '1';
            opera1   <= '1';
            pronto   <= '0';
        when boot =>
            ini      <= '1';
            cStart   <= '1';
            cEnd     <= '1';
            cMid     <= '1';
            cResultado <= '1';
            sub      <= '1';
            multiplicar <= '0';
            mResultado <= '1';
            opera1   <= '1';
            pronto   <= '0';
        when defineMid =>
            ini      <= '0';
            cStart   <= '0';
            cEnd     <= '0';
            cMid     <= '1';
            cResultado <= '1';
            sub      <= '0';
            multiplicar <= '0';
            mResultado <= '1';
            opera1   <= '1';
            pronto   <= '0';
    end case;
end process;

```

```

when initializeSquareMid =>
    ini    <= '0';
    cStart <= '0';
    cEnd   <= '0';
    cMid   <= '0';
    cResultado <= '1';
    sub   <= '1';
    multiplicar <= '1';
    mResultado <= '1';
    opera1  <= '1';
    pronto  <= '0';
when squareMid =>
    ini    <= '0';
    cStart <= '0';
    cEnd   <= '0';
    cMid   <= '0';
    cResultado <= '1';
    sub   <= '1';
    multiplicar <= '0';
    mResultado <= '1';
    opera1  <= '1';
    pronto  <= '0';
when squaredMidWithX =>
    ini    <= '0';
    cStart <= '0';
    cEnd   <= '0';
    cMid   <= '0';
    cResultado <= '1';
    sub   <= '1';
    multiplicar <= '0';
    mResultado <= '1';
    opera1  <= '1';
    pronto  <= '0';
when squaredMidGreaterX =>
    ini    <= '0';
    cStart <= '0';
    cEnd   <= '1';
    cMid   <= '0';
    cResultado <= '1';
    sub   <= '1';
    multiplicar <= '0';
    mResultado <= '1';
    opera1  <= '0';
    pronto  <= '0';
when squaredMidEqualX =>
    ini    <= '1';
    cStart <= '1';
    cEnd   <= '1';
    cMid   <= '1';
    cResultado <= '1';
    sub   <= '1';
    multiplicar <= '1';
    mResultado <= '1';
    opera1  <= '1';
    pronto  <= '0';

```

```

when squaredMidLesserX =>
    ini    <= '0';
    cStart <= '1';
    cEnd   <= '0';
    cMid   <= '0';
    cResultado <= '1';
    sub   <= '0';
    multiplicar <= '0';
    mResultado <= '1';
    opera1  <= '1';
    pronto  <= '0';
when startWithEnd =>
    ini    <= '0';
    cStart <= '0';
    cEnd   <= '0';
    cMid   <= '0';
    cResultado <= '1';
    sub   <= '1';
    multiplicar <= '0';
    mResultado <= '1';
    opera1  <= '1';
    pronto  <= '0';
when startGreaterEnd =>
    ini    <= '1';
    cStart <= '1';
    cEnd   <= '1';
    cMid   <= '1';
    cResultado <= '1';
    sub   <= '1';
    multiplicar <= '1';
    mResultado <= '0';
    opera1  <= '1';
    pronto  <= '0';
when others =>
    ini    <= '1';
    cStart <= '1';
    cEnd   <= '1';
    cMid   <= '1';
    cResultado <= '0';
    sub   <= '1';
    multiplicar <= '1';
    mResultado <= '1';
    opera1  <= '1';
    pronto  <= '1';
end case;
end process;
end arch;

```

# BO

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity sqrt_bo is
    generic(n : natural);
    port(clk, reset,
        ini, cStart, cEnd, cMid, cResultado, sub, multiplicar, mResultado, opera1 : in std_logic;
        entrada : in std_logic_vector(n-1 downto 0);
        multiplicado, startMaiorEnd, midMaiorX, midEqualX : out std_logic;
        saida : out std_logic_vector(n-1 downto 0));
end sqrt_bo;

architecture estrutura of sqrt_bo is

COMPONENT registrador IS
    generic (n : natural);
    PORT (clk, carga : IN STD_LOGIC;
        d : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
        q : OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0));
END COMPONENT;

COMPONENT mux2para1 IS
    generic (n : natural);
    PORT (a, b : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
        sel: IN STD_LOGIC;
        y : OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0));
END COMPONENT;

COMPONENT somadorsubtrator IS
    generic (n : natural);
    PORT (a, b : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
        op: IN STD_LOGIC;
        Cout : OUT STD_LOGIC;
        s : OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0));
END COMPONENT;

COMPONENT igualazero IS
    generic (n : natural);
    PORT ( a : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
        igual : OUT STD_LOGIC);
END COMPONENT;

COMPONENT diferentezero IS
generic (n:natural);
PORT ( a : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
        diferente : OUT STD_LOGIC);
END COMPONENT;

COMPONENT multiplier2 IS
generic (n;natural);
port(entA, entB : in std_logic_vector(n-1 downto 0);
        iniciar, Reset, ck : in std_logic;
        pronto : out std_logic;
        mult : out std_logic_vector((2*n)-1 downto 0));
END COMPONENT;

signal saidaSTART, saidaMID, saidaEND, saidaX, saidaSOMASUB_StartEnd, saidaSOMASUB_MidX, saidaSOMASUB_StartEndShifted,
saídaMuxSTART, saídaMuxEND, saídaMuxRESULTADO, saídaMuxMidX, saídaMuxMidOne,
zero : std_logic_vector(n-1 downto 0);
signal saidaMULT : std_logic_vector(2*n-1 downto 0);
signal coutStartEnd, coutMidX, midMaiorXPorBits : std_logic;

begin
    X: registrador
        generic map (n => n)
        port map(clk, ini, entrada, saidaX);

    Start : registrador
        generic map (n => n)
        port map(clk, cStart, saidaMuxSTART, saidaSTART);

    Mid : registrador
        generic map (n => n)
        port map(clk, cMid, saidaSOMASUB_StartEndShifted, saidaMID);

    End_r : registrador
        generic map (n => n)
        port map(clk, cEND, saidaMuxEND, saidaEND);

    Resultado : registrador
        generic map (n => n)
        port map(clk, cResultado, saidaMuxRESULTADO, saida);

    muxStart : mux2para1
        generic map (n => n)
        port map(saidaSOMASUB_MidX, (others => '0'), ini, saidaMuxSTART);

    muxEnd : mux2para1
        generic map (n => n)
        port map(saidaSOMASUB_MidX, entrada, ini, saidaMuxEND);

    muxMidX : mux2para1
        generic map (n => n)
        port map(saidaX, saidaMID, opera1, saidaMuxMidX);

    muxMidOne : mux2para1
        generic map (n => n)
        port map(saidaMULT(n-1 downto 0), ((0) => '1', others => '0'), opera1, saidaMuxMidOne);

```

```

muxResultado : mux2para1
generic map (n => n)
port map(saidaEND, saidaMID, mResultado, saidaMuxRESULTADO);

SomaSub_StartEnd : somadorsubrator
generic map (n => n)
port map(saidaEND, saidaSTART, sub, coutStartEnd, saidaSOMASUB_StartEnd);
saidaSOMASUB_StartEndShifted <= coutStartEnd & saidaSOMASUB_StartEnd(n-1 downto 1);
startMaiorEnd <= coutStartEnd;

SomaSub_MidX : somadorsubrator
generic map (n => n)
port map(saidaMuxMidX, saidaMuxMidOne, sub, coutMidX, saidaSOMASUB_MidX);

StartIgualX : igualzero
generic map (n => n)
port map(saidaSOMASUB_MidX, midIgualX);

Multiplier : multiplier2
generic map (n => n)
port map(saidaMID, saidaMID, multiplicar, reset, clk, multiplicado, saidaMULT);

MidGreaterThanNBits : diferenzerzero
generic map (n => n)
port map(saidaMULT(2*n-1 downto n), midMaiorXPorBits);

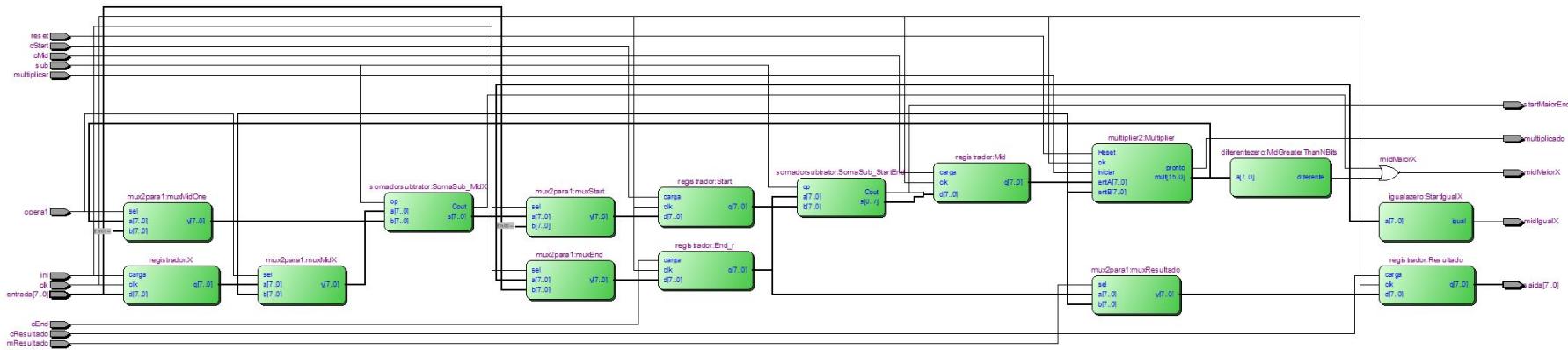
midMaiorX <= coutMidX or midMaiorXPorBits;

end estrutura;

```

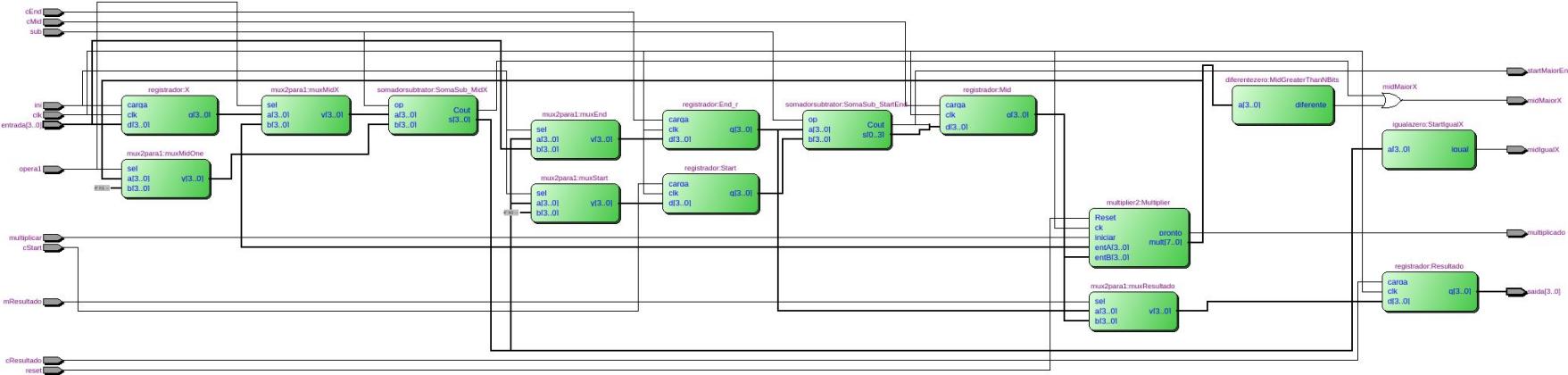
# Netlists

## Bloco operativo 8 bits



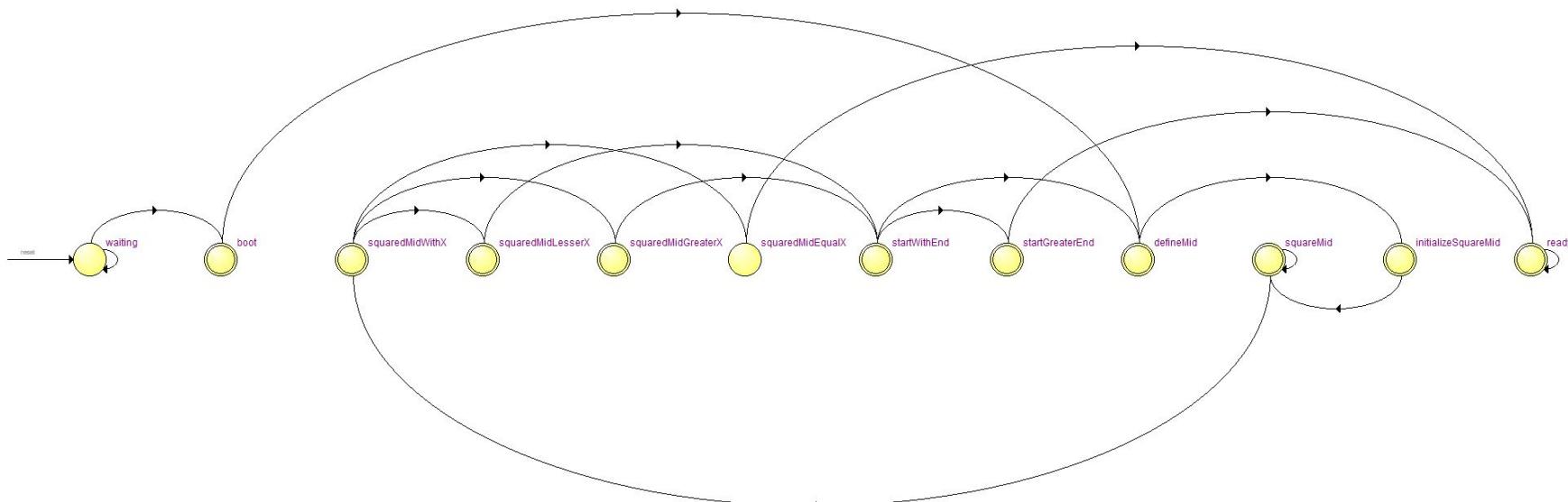
# Netlists

## Bloco operativo 4 bits



# Netlists

## Bloco de Controle



# Resultados de síntese

N = 4

Slow Model Fmax Summary				
	Fmax	Restricted Fmax	Clock Name	Note
1	237.59 MHz	237.59 MHz	clk	

Flow Summary		
Flow Status	Successful - Wed Dec 09 20:04:34 2020	
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition	
Revision Name	sqrt	
Top-level Entity Name	sqrt	
Family	Cyclone II	
Device	EP2C35F672C6	
Timing Models	Final	
Total logic elements	96 / 33,216 (< 1 %)	
Total combinational functions	89 / 33,216 (< 1 %)	
Dedicated logic registers	67 / 33,216 (< 1 %)	
Total registers	67	
Total pins	12 / 475 (3 %)	
Total virtual pins	0	
Total memory bits	0 / 483,840 (0 %)	
Embedded Multiplier 9-bit elements	0 / 70 (0 %)	
Total PLLs	0 / 4 (0 %)	

Analysis & Synthesis Resource Usage Summary		
	Resource	Usage
1	Estimated Total logic elements	107
2		
3	Total combinational functions	89
4	Logic element usage by number of LUT inputs	
1	-- 4 input functions	19
2	-- 3 input functions	45
3	-- <=2 input functions	25
5		
6	Logic elements by mode	
1	-- normal mode	72
2	-- arithmetic mode	17
7		
8	Total registers	67
1	-- Dedicated logic registers	67
2	-- I/O registers	0
9		
10	I/O pins	12
11	Embedded Multiplier 9-bit elements	0
12	Maximum fan-out node	clk
13	Maximum fan-out	67
14	Total fan-out	465
15	Average fan-out	2.77

# Resultados de síntese

N = 8

Slow Model Fmax Summary				
	Fmax	Restricted Fmax	Clock Name	Note
1	222.27 MHz	222.27 MHz	clk	

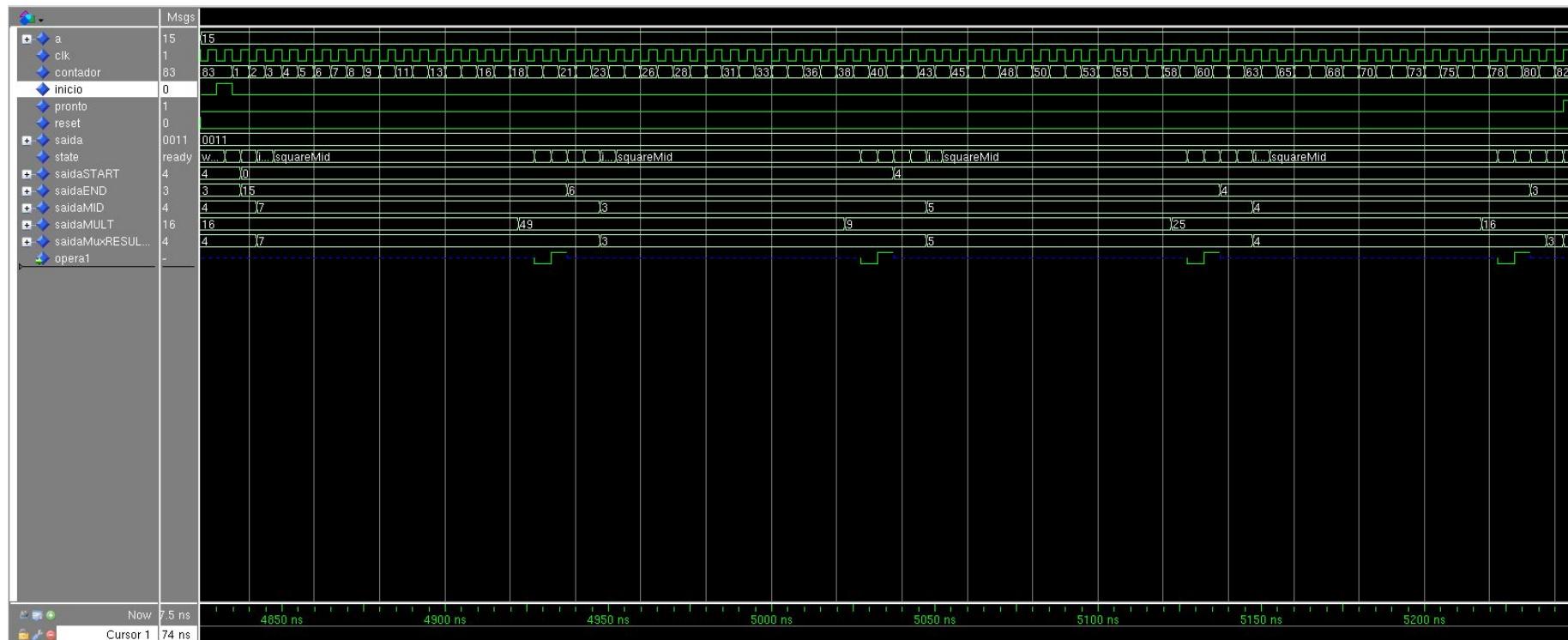
Flow Summary		
Flow Status:	Successful - Wed Dec 09 19:08:19 2020	
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition	
Revision Name	sqrt	
Top-level Entity Name	sqrt	
Family	Cyclone II	
Device	EP2C35F672C6	
Timing Models	Final	
Total logic elements	151 / 33,216 (< 1 %)	
Total combinational functions	143 / 33,216 (< 1 %)	
Dedicated logic registers	112 / 33,216 (< 1 %)	
Total registers	112	
Total pins	20 / 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 %)	

Analysis & Synthesis Resource Usage Summary		
	Resource	Usage
1	Estimated Total logic elements	177
2		
3	Total combinational functions	143
4	Logic element usage by number of LUT inputs	
1	-- 4 input functions	33
2	-- 3 input functions	78
3	-- <=2 input functions	32
5		
6	Logic elements by mode	
1	-- normal mode	110
2	-- arithmetic mode	33
7		
8	Total registers	112
1	-- Dedicated logic registers	112
2	-- I/O registers	0
9		
10	I/O pins	20
11	Embedded Multiplier 9-bit elements	0
12	Maximum fan-out node	clk
13	Maximum fan-out	112
14	Total fan-out	776
15	Average fan-out	2.82

# Resultados de síntese

	N = 4	N = 8
Clock	237.59 MHz	222.27 MHz
Logic elements (normal)	72	110
Logic elements (arithmetic)	17	33
Registradores	67	112
Atraso crítico	4.209 ns	4.500 ns
Pior caso em ciclos	83	243

# Resultados de Validação - 4 bits



# Resultados de Validação - 8 bits

