Algorítmo:

Raiz quadrada por busca binária:

Pensando da forma mais trivial possível, seria fácil descobrir a raiz, percorrermos todos os números menores que N(podendo ser reduzido pela metade de N), e verificar se a multiplicação de N1 * N1, acarreta na igualdade a n. Como exemplo, um algoritmo em python.

```
n = int(input())
res = 0
for (int i = 0; i <= n; i++):
    if i * i == n:
        res = i
        break
```

Basta pensar um pouco que já conseguimos imaginar os diversos problemas desse algoritmo trivial, além de ser lento pensando em possibilidades enormes de números ele também é falho quando lidamos com números que não possuem raiz exata como 33, nunca chegaríamos em uma possibilidade próxima da resposta já que não teríamos valor que suprisse a igualdade.

Por isso, decidimos usar o algoritmo de **busca binária** baseado em um conceito que é será apresentado de maneira aprofundado em estrutura de dados ao estudar Árvores de Busca Binária, este algoritmo elegante é muito útil quando vamos lidar com um vetor ordenado de números no nosso caso, por exemplo para um número 64, vetor = [1,2,3,4,5,6,7,8...64].

Assim como na árvore binária em nosso algoritmo, a cada passo que percorremos iremos eliminar de cara metade das possibilidades, ou seja, enquanto o algoritmo padrão tem complexidade N(com erros em vários valores), o nosso irá apresentar complexidade log 2 N.

Como exemplo, em um caso mais simples(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

Porém, como já mencionamos antes existe o caso de números não que a raiz é fracionária como o numero 10 por exemplo, para solucionar isso trabalhos com um conceito de piso e teto, para controlar quando devemos parar de verificar e retornar dessa forma um valor aproximado para o usuário, trecho do algoritmo em python.

```python
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
```

Fazendo passo a passo:

[0,1,2,3,4,5,6,7,8,9,10] n = 10, onde 1 é o piso, 10 é o teto

10 // 2 → 5(mid)

5 * 5 > 10, logo teremos que nosso teto passará a ser

5 - 1 = 4.

[0,1,2,3,4], continuando:

4 // 2 → 2 (mid)

2 * 2 < 10, logo teremos que nosso teto irá continuar em 4, e nosso piso passa a ser mid + 1, logo 3.

[3,4], temos que;

(3+4) // 2 → 3, seguindo sabemos que 3 * 3 < 10, então nosso piso passa a ser 4.

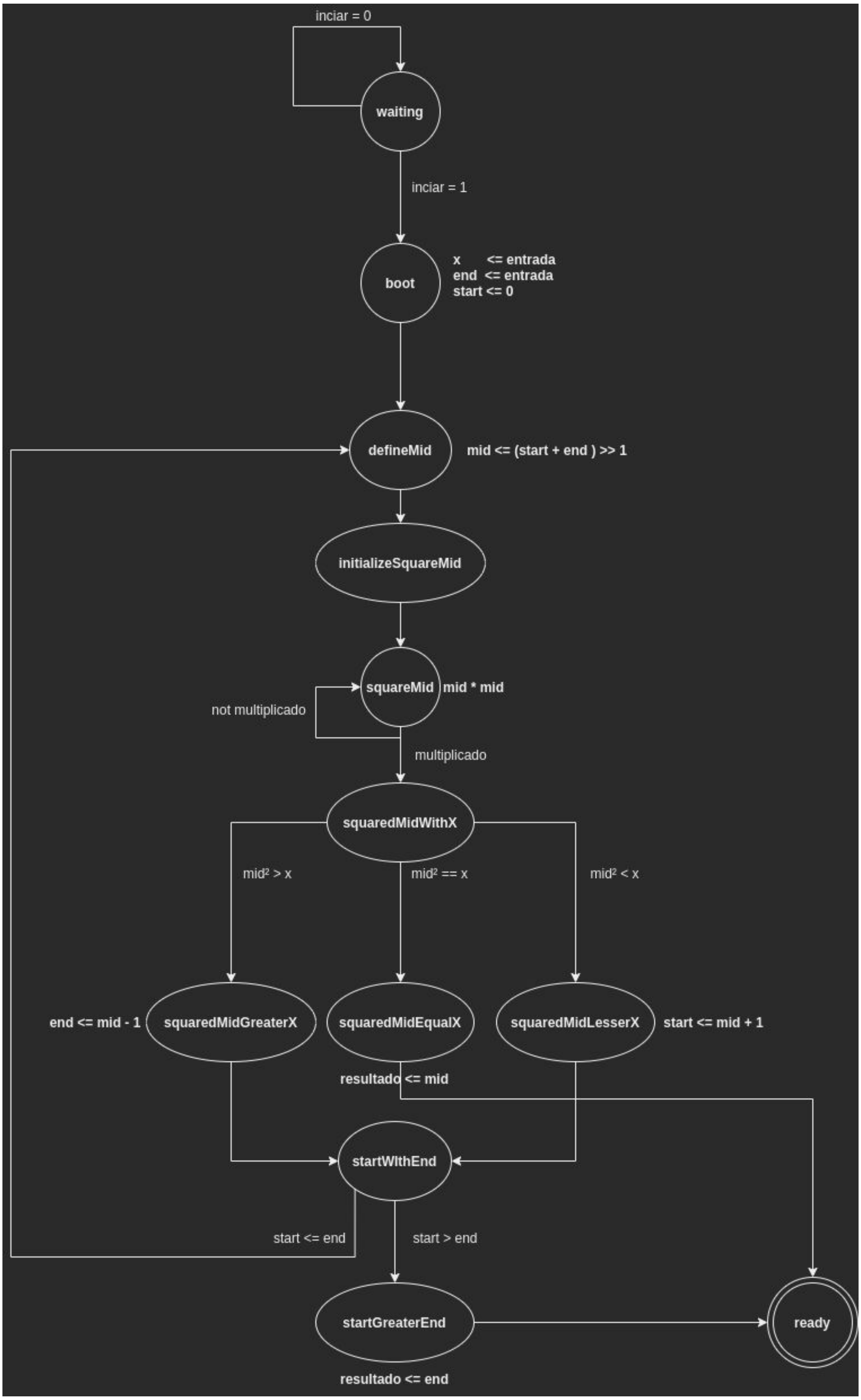
[4,4], continuando:

(8)//2 → 4(mid)

4 * 4 > 10, portanto nosso teto passará a ser 3, e sairemos do loop pois o piso está maior que o teto.

Ficando assim com a resposta final aproximada armazenada no teto, que é o 3(valor arredondado para raiz de 10).

Máquina de Estados



Transformando o algoritmo em máquina de estados chegamos ao seguinte resultado.

Começa no Estado inicial com até que seja iniciado.

Depois vai para o estado de boot, onde as variáveis recebem seus respectivos valores. após as variáveis terem recebido seus valores, chega-se ao estado defineMid, onde o mid recebe a média entre as variáveis start e end (start + end e desloca para direita, ou seja, divide por 2).

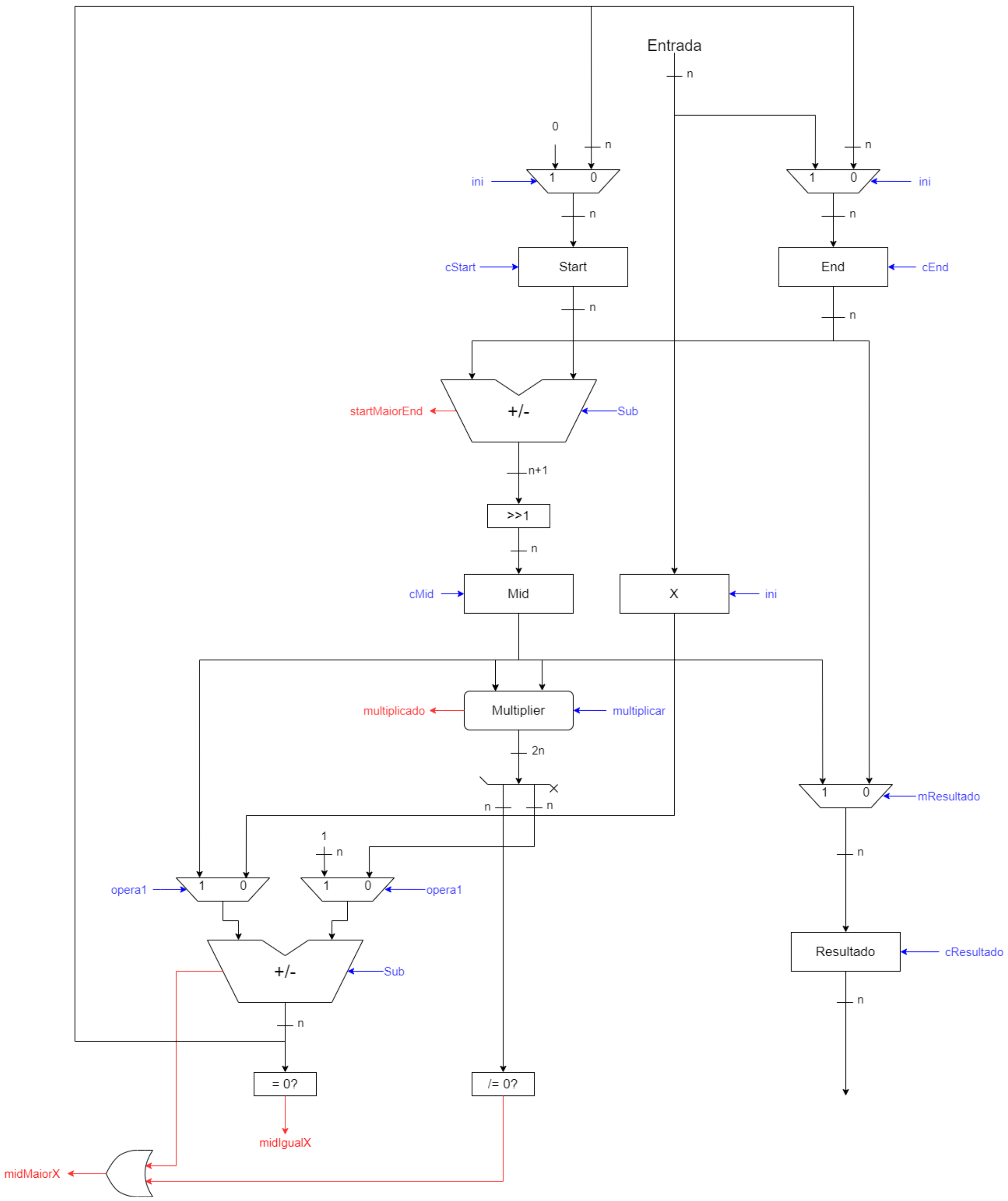
Com o mid setado, chegamos ao estado initializeSquareMid, onde os multiplicadores são inicializados.

Após isso, o Estado squareMid eleva o mid ao quadrado, jogando-o ao estado squaredMidWithX. A partir daí tem-se 3 caminhos diferentes que podem ser seguidos. Caso mid**2 seja maior que x (número que queremos calcular a raiz), segue-se para o estado squaredMidGreaterX, onde o end recebe o valor de mid - 1, caso mid**2 seja igual a x, segue-se para squaredMidEqualX, onde o resultado recebe o valor de mid e a máquina vai para o estado pronto (estado final), caso mid**2 < x, segue-se para o estado squaredMidLesserX, onde start recebe o valor de mid + 1.

A partir dos dois estados, squaredMidGreaterX e squaredMidLessserX, a máquina vai para o estado startWithEnd, onde checka se start > end (o que é um absurdo e indica que o valor da raiz aproximada já foi achado), a partir daí, caso start seja menor ou igual ao end a máquina volta ao estado defineMid e repete o processo até chegar ao resultado.

Arquiteturas (diagramas de blocos do BO)

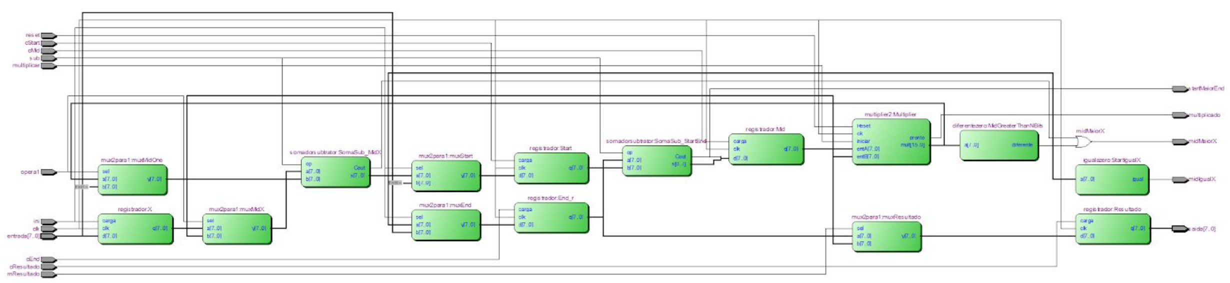
Transformando a máquina de estados em arquitetura chegamos ao seguinte Bloco de Operativo:



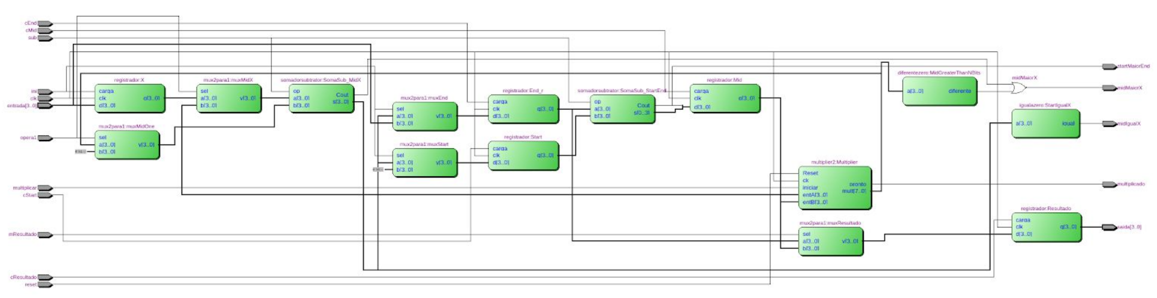
ULAs são usadas para obter o valor do mid ((start + end) / 2) e para diminuir x do mid e comparar quem é maior. Enquanto isso o multiplicador foi reaproveitado do último trabalho em equipe para fazer a multiplicação do mid*mid.

Netlists

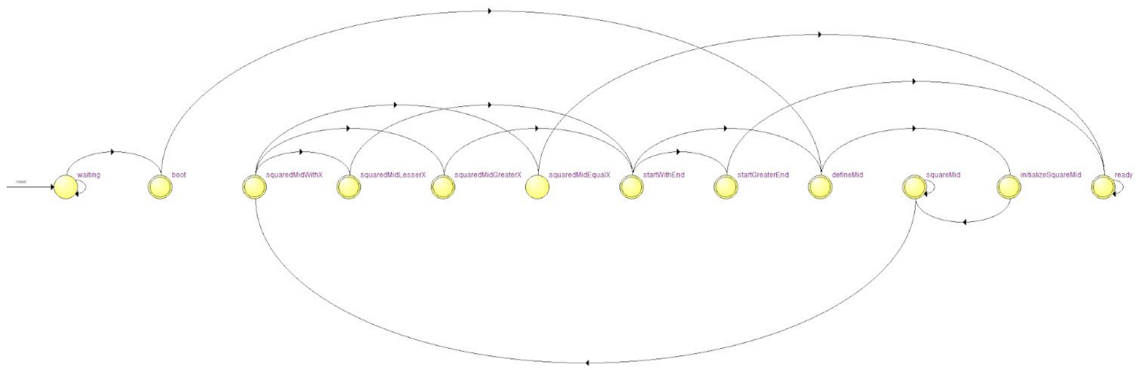
bloco operativo 8bits:



bloco operativo 4bits:



bloco de controle:



Resultados das netlists do bloco operativo de 4 e de 8 bits, e do 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

n = 8

Slow Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	222.27 MHz	222.27 MHz	clk	

Flow Summary

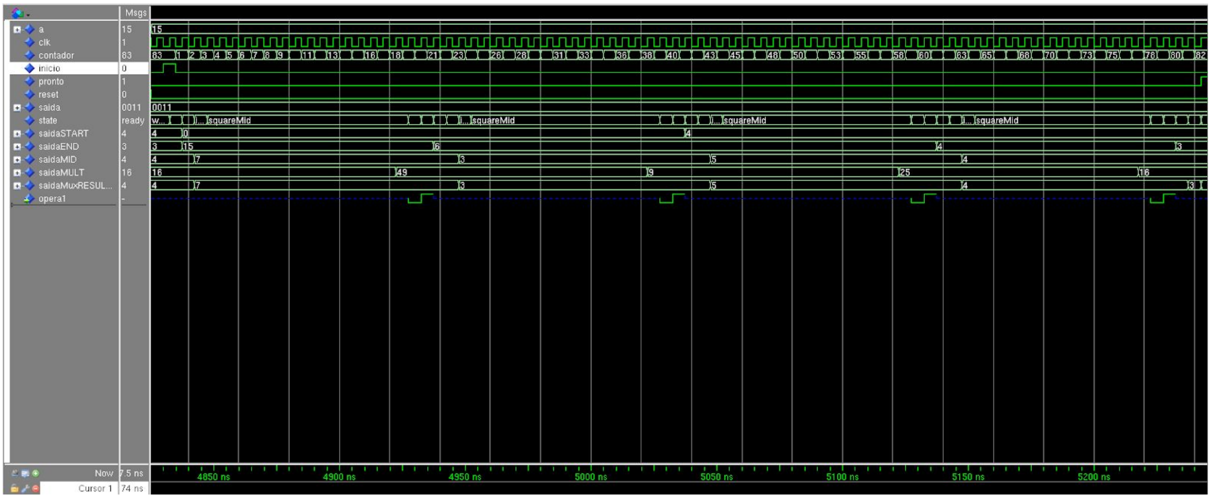
Flow Status	Successful - Wed Dec 09 19:08:19 2020
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	sqrt
Top-level Entity Name	sqrt
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	151 / 33,216 (< 1 %)
Total combinational functions	143 /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

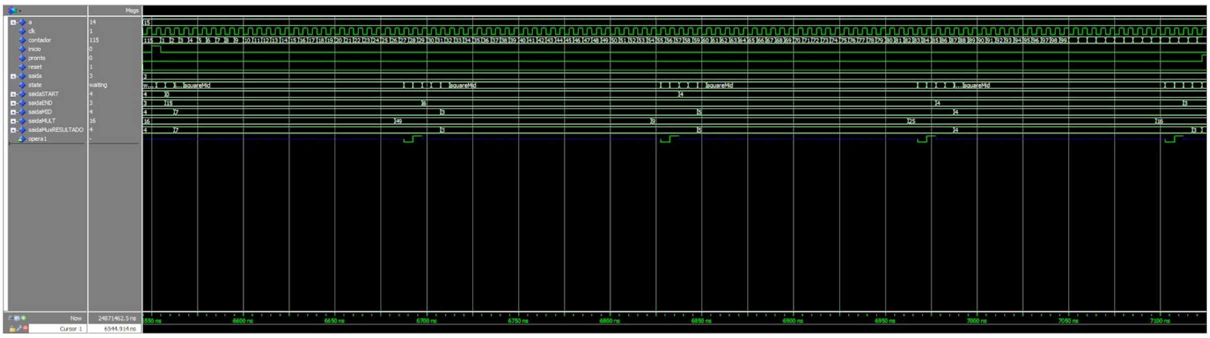
Analysis & Synthesis Resource Usage Summary		
	Resource	Usage
1	Estimated Total logic elements	177
2		
3	Total combinational functions	143
4	Logic element usage by number of LUT inputs	
1	-- 4 input functions	33
2	-- 3 input functions	78
3	-- <=2 input functions	32
5		
6	Logic elements by mode	
1	-- 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

Geral

	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

Resultados de Validação





Análise Crítica

Na solução adotada em nosso projeto, como já dito anteriormente, foi levado em conta 2 fatores principais: cálculo do resultado de maneira eficiente, e lidar com valores mais complexos como raiz de número não exato retornando uma aproximação de resposta.

Na projeção deste trabalho, reaproveitamos o que já tinha sido desenvolvido do multiplicador 2 para base das operações, levando em consideração que dentre os multiplicadores ele era o que apresentou melhor desempenho(custo/benefício).

Ademais bastou fazer, as operações lógicas e cálculos para poder realizar o controle de condições e seleção de teto e piso, para saber quando trata-se do resultado final.