Laboratório 02 - OAC

Rodrigo Mamédio Arrelaro, 190095164
Thiago Masera Tokarski, 190096063
Eduardo Dantas Xavier, 190086530
Grupo 3

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CIC0231 - Laboratório de Circuitos Lógicos

190095164@aluno.unb.br, 190096063@aluno.unb.br, 190086530@aluno.unb.br

Abstract. *In the laboratory in question, studies on the languages of hardware description (LDH) will be elaborated, with emphasis on the language "object-oriented"* ***SystemVerilog****, together with the development software* ***QUARTUS****. Using such tools, the studies were supported by the implementations and optimizations (taking into account the physical and temporal requirements) of two types of* ***"ULA"*** *(Arithmetic logic unit), one for* ***ISA*** *"RV32imf ", focusing on integers and the second, from the same ISA, focusing on floating point numbers.*

Resumo. *No laboratório em questão será elaborado estudos sobre as linguagens de descrição de hardware (LDH), com ênfase na linguagem "orientada a objetos"[Wikipedia 2016]SystemVerilog, junto do software de desenvolvimento [Quartus 2021] QUARTUS. Utilizando tais ferramentas, os estudos se apoiaram nas implementações e otimizações (levando em consideração os requisitos físicos e temporais) de dois tipos de [Wikipedia 2021c]"ULA"(Arithmetic logic unit), sendo uma para a [Wikipedia 2021b]ISA "RV32imf", com enfoque em inteiros e a segunda, de mesma ISA, com enfoque em números de ponto flutuante.*

1. Objetivos

O laboratório em questão visa:

- Desenvolver a capacidade de codificação, síntese e desenvolvimento de circuitos digitais utilizando o SystemVerilog (LHD);
- Desenvolver a capacidade de análise de desempenho de entidades físicas e temporais em "FPGA"(simulada), utilizando o QUARTUS da Intel.

2. Desenvolvimento

2.1. ULA para inteiros

Utilizando os arquivos "ALU.QAR"(previamente disponibilizados pelo professor), foi realizado o processo de "restauração"do projeto da ULA. Posteriormente foi realizado uma análise do código em SystemVerilog e realizado uma tabela com seus comportamentos.

```
1   /*
2    * ALU
3    *
4    */
5   //`define RV32I
6   //`define RV32IM
7   //`define RV32IMF
8
9   `ifndef PARAM
10     `include "Parametros.v"
11  `endif
12
13
14
15  module ALU (
16     //input      [4:0]  iControl,
17     input signed [31:0] iA,
18     input signed [31:0] iB,
19     output logic [31:0] oResult
20     );
21
22     wire [4:0] iControl=OPSUB;    // Usado para as analises
23
24
25
26  `ifndef RV32I
27  wire [63:0] mul, mulu, mulsu;
28  assign mul  = iA * iB;
29  assign mulu = $unsigned(iA) * $unsigned(iB);
30  assign mulsu= $signed(iA) * $unsigned(iB);
31  `endif
32
33  /*
34  MUX_ADD m21(
35     .iAA (iA),
36     .iBB (iB),
37     .iCControl (iControl),
38     //.oRResult (SaidaMUX)
39     //.SaidaMux(oRResult)
40     );
41     wire SaidaMUX[31:0];
42     */
43
44
```

Figura 1. Inicio do código LDH da ULA

Suprimindo os conteúdos escritos como ”comentários”, o código começa a partir da linha de número nove, a mesma realiza uma simples verificação para a existência do arquivo ”Parametros.v”, sendo este realmente presente nos arquivos do projeto. Tal arquivo indica ao projeto da ”ULA”qual ISA será usada, qual tipo de processador será implementado (no caso deste laboratório, nenhum processador foi utilizado), o comportamento das operações que serão realizadas pela ULA (como exemplo ”AND”, ”OR”, ”XOR”e etc) e outras funcionalidades não relevantes no momento.
Posteriormente o módulo ”ALU”é necessariamente declarado, recebendo como ”input”os valores ”iA”e ”iB”(ambos com 32 bits de extensão), o sinal ”iControl”indicando qual operação será realizada e como ”output”o sinal ”lógicooResult”(também com extensão de 32 bits). É perceptível a presença de outro barramento com 5 fios, mas este é usado apenas para as análises individuais de cada operação da ULA, tão logo que na linha ”22”o próprio ”iControl”recebe o valor para a operação a ser analisada.
Seguido da declaração do módulo ”ALU”, o código aloca a quantidade de bits que serão necessários para as operações de multiplicação (visto que uma multiplicação entre dois números de 32 bits, resulta em um número de 64 bits) e declara o comportamento das operações propriamente ditas (”Mul”, ”Mulu”e ”Mulsu”).
A partir deste momento o código apresenta um circuito combinacional (visto a declaração do comando ”**always @(*)**”) acompanhado de um switch case para as operações. Desta forma, para uma fácil interpretação e análise dos dados foi montado a tabela com cada instrução e seus requisitos físicos.

Operação	nº ALM	nº Registradores	nº bits em memória	nº DSP
AND	65	0	0	0
OR	65	0	0	0
XOR	65	0	0	0
ADD	65	0	0	0
SUB	65	0	0	0
SLT	79	0	0	0
SLTU	79	0	0	0
SLL	120	0	0	0
SRL	121	0	0	0
SRA	118	0	0	0
LUI	49	0	0	0
MUL	56	0	0	2
MULH	72	0	0	3
MULHU	72	0	0	3
MULHSU	88	0	0	6
DIV	704	0	0	0
DIVU	633	0	0	0
REM	731	0	0	0
REMU	652	0	0	0
ULA Completa	3029	0	0	12

Uma observação de extrema importância a ser feita com esta tabela é o fato da quantidade de ”ALM”e ”DPS”não ”bater”com a quantidade por cada operação, contudo tal evento se deve ao fato que o ”QUARTUS”realiza (automaticamente, no processo de compilação) uma otimização do circuito, desta forma o valor ”teórico”de 3899 acaba sendo reduzido para os 3029 ALMs existentes no circuito, assim como os ”teóricos”14 DSPs que são reduzidos para 12.
Seguindo com as análises propostas, foi montada outra tabela com os requisitos temporais para as operações e para a ULA completa.

Operação	Caminho de maior atraso	Maior TDP
AND	iB[18] ->oResult[18]	2421
OR	iB[18] ->oResult[18]	2421
XOR	iB[18] ->oResult[18]	2421
ADD	iB[10] ->oResult[27]	3127
SUB	iA[6] ->oResult[18] e iA[6] ->oResult[24]	3282
SLT	iA[18] ->oResult[0]	4490
SLTU	iA[18] ->oResult[0]	4490
SLL	iA[19] ->oResult[24]	4150
SRL	iA[27] ->oResult[18]	4412
SRA	iA[20] ->oResult[11]	4639
LUI	iB[29] ->oResult[29]	0.516
MUL	iB[9] ->oResult[28]	7734
MULH	iB[1] ->oResult[22]	8156
MULHU	iA[6] ->oResult[31]	8684
MULHSU	iB[28] ->oResult[31]	10803
DIV	iB[31] ->oResult[31]	94892
DIVU	iB[27] ->oResult[0]	83135
REM	iB[31] ->oResult[24]	90438
REMU	iB[26] ->oResult[9]	88790
ULA Completa	iB[9] ->oResult[10]	106096

Ao final do processo de montagem das tabelas, também foi realizado estudos apoiados nas formas de onda do circuito em questão. Desta forma é possível se analisar o comportamento da ULA (principalmente para quesitos como Overflow, underflow e etc).

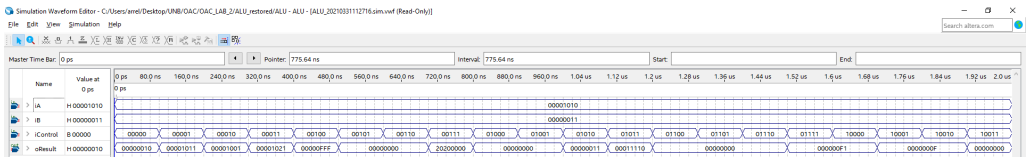

Figura 2. Forma de onda resultante entre os valores 10 e 3

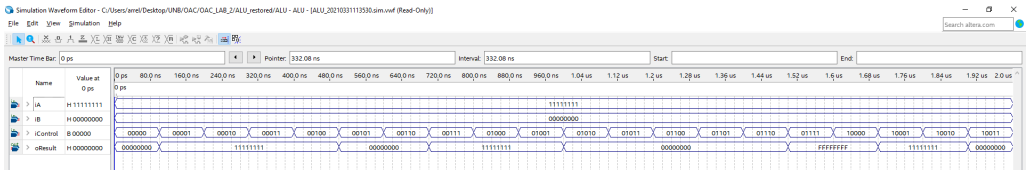

Figura 3. Forma de onda resultante entre os valores 255 e 0

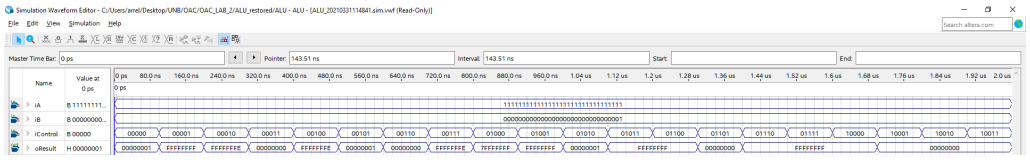

Figura 4. Forma de onda resultante entre os valores 255 e 1

Principalmente nas operações de código ”00011”é perceptível os problemas de ”Overflow”.
Desta forma a ULA de inteiros acaba se apresentando por:

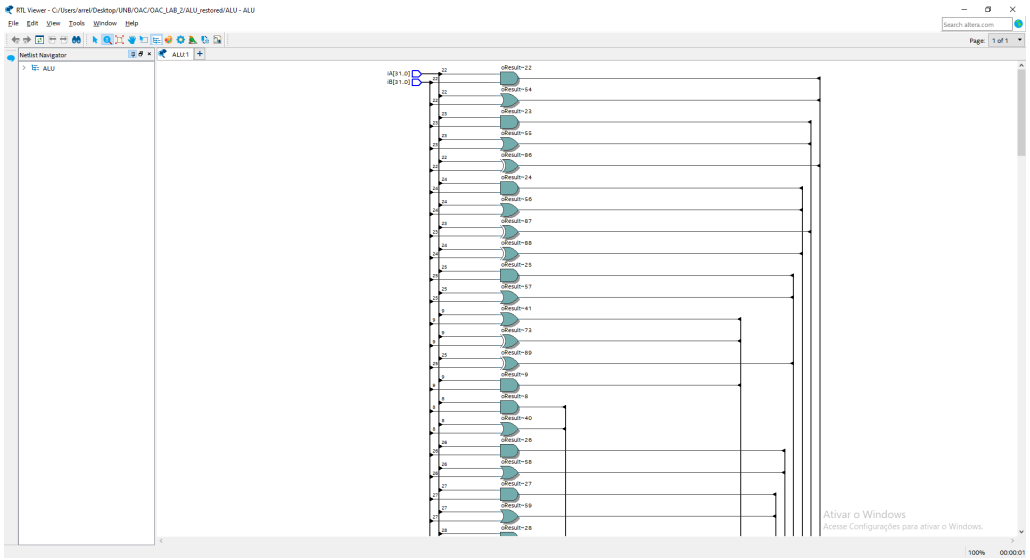

Figura 5. Resultado do processo de compilação do QUARTUS para a ALU

2.2. Plano para otimização dos requisitos temporais e/ou físicos da ULA para inteiros

Ulteriormente as análises realizadas, foi levantado duas hipóteses para a otimização do circuito.

- A adição de um multiplexador antes de determinadas operações (teoricamente diminuindo a quantidade de entidades dentro da ULA);
- A subtração do ”default”case (por mais que seja contra as ”boas práticas”).

Partindo para a análise do primeiro tópico, a ideia levantada nasceu ao ser realizada uma observação (utilizando o ”RTL Viewer”) do circuito da ULA.

Figura 6. Circuito para as operações ”SUM”e ”SUB”.

A metodologia para simplificação seria a seguinte:

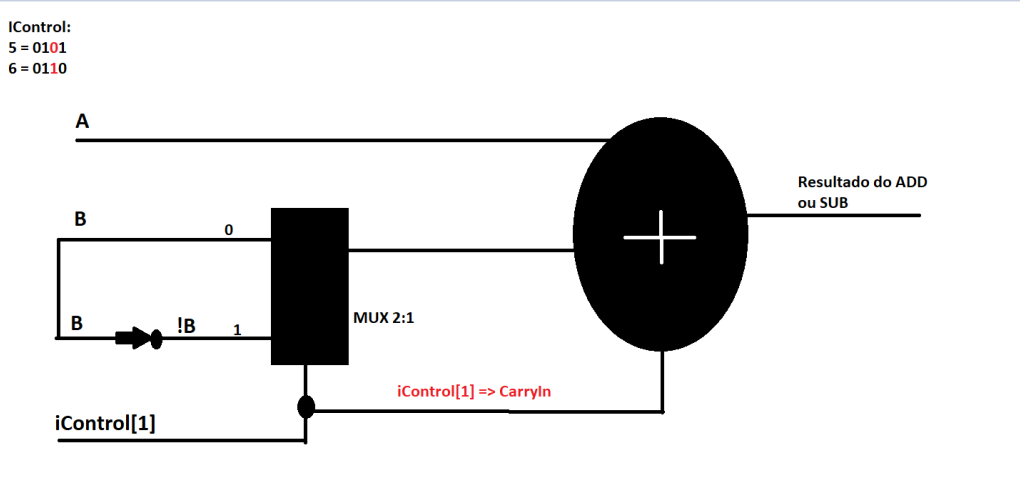

Figura 7. ”Planta”para o processo de simplificação das operações ”Sum”e ”Sub”.

O código para a operação (vindo do iControl) ”Sum”é ”0101”e para a operação ”Sub”é ”0110”, desta forma o circuito consistiria em pegar o segundo bit do ”iControl”e conectá-lo como o controlador do ”MUX 2:1”, não apenas isso mas o mesmo ”segundo bit”seria o **”Carry in”** do somador.
Exemplificando, caso a operação fosse uma subtração, o segundo bit seria ”1”, assim o valor que iria para a saída do ”MUX”seria ”!B”e o primeiro ”Carry in”do somador seria ”1”, desta forma a subtração seria realizada da forma ”correta”de acordo com o algoritmo (uma soma entre um número positivo e o complemento de 2 do outro).
Contudo, ao tentar ”sintetizar”esta solução no ”QUARTUS”o mesmo (por motivos de otimização do próprio compilador), acabou entregando o circuito de outra forma:

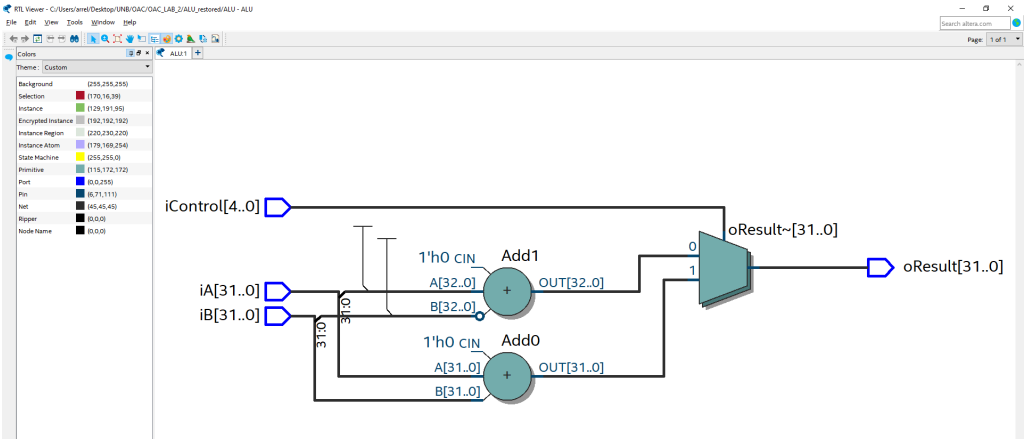

Figura 8. Resultado do processo de compilação do QUARTUS para a implementação do MUX 2:1

Tal resultado foi obtido devido o busca da otimização ”temporal”do QUARTUS, o compilador busca a melhor implementação para os tempos necessários serem os menores.

Tais informações foram confirmadas dentro do próprio suporte da **”ALTERA”**. Exemplificando outras adversidades que o próprio ”QUARTUS”apresenta é o fato da plataforma não aceitar a troca dos valores do ”carry in”para os somadores (visto que os mesmos são P.I da INTEL).

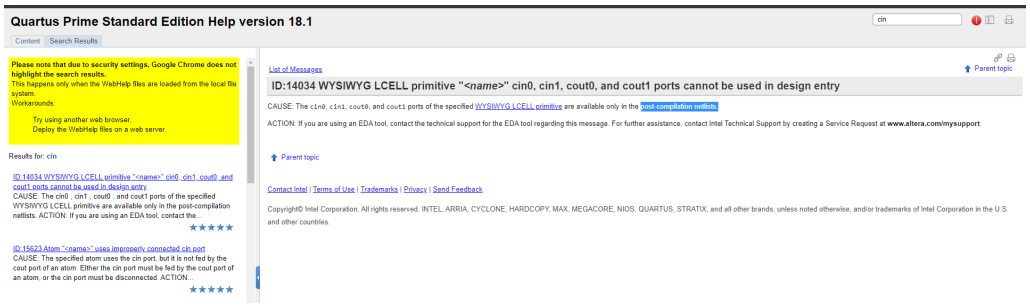

Figura 9. Página explicando a limitação de ”não alteração dos carry in”.

Seguindo para a análise do segundo tópico levantado para uma ”otimização”do circuito foi debatido a ideia da retirada case ”Default”do circuito, visto que o mesmo acaba por gerar um ”ALM”para cada saída possível para ”oResult”(por mais que seja correto declarar um caso ”default”em códigos para descrição de hardware).

2.3. ULA para ponto flutuante

Utilizando o arquivo ”FPALU.qar”(previamente disponibilizado pelo professor), foi realizado também o processo de ”restauração”do projeto da ULA. Para assim, serem feitas as análises do código em SystemVerilog e a montagem das tabelas.

Assim como observado na ULA para inteiros, o QUARTUS (em seu processo de compilação) acaba ”otimizando”o circuito para ter o menor valor de atraso possível, assim também otimizando a parte física do circuito. Sabendo disso, o valor teórico de 3029 ALM’s acaba caindo para 2662. Contudo, ao ser realizado as análises na quantidade de blocos ”DSP”, o circuito final acabou utilizando mais unidades para a implementação da ”ULA”, sendo o valor teórico de 8 e o real implementado de 12.
Seguindo com as análises propostas, foi montada a tabela com os requisitos temporais para as operações da ULA para ponto flutuante.

Operação	nº ALM	nº Registradores	nº bits em memória	nº DSP
ADD	386	284	0	0
SUB	383	289	0	0
MUL	116	77	0	1
DIV	207	290	31744	3
SQRT	126	129	15872	2
ABS	49	2	0	0
CEQ	79	24	0	0
CLT	87	31	0	0
CLE	87	30	0	0
CVTSW	186	157	0	0
CVTWS	202	77	0	0
CVTSWU	160	110	0	0
CVTWUS	188	43	0	0
MV	49	2	0	0
SGNJ	49	2	0	0
SGNJN	49	2	0	0
SGNJX	50	2	0	0
MAX	95	2	0	0
MIN	95	2	0	0
NULL	49	2	0	0
ULA completa	1325	1100	47616	6

Outra observação de vital importância para o experimento em questão é a otimização temporal que o QUARTUS realiza, visto que para a ULA de números de ponto flutuante existe a necessidade do uso de um sinal de ”clock”, contudo devido as otimizações (por incrível que pareça) os requerimentos temporais para a ULA ”funcionar”acabaram sendo os mesmos para um clock de frequência 50Mhz e para uma frequência de 500Mhz.
Seguindo a mesma metodologia usada para as análises da ULA para inteiros, foi realizado estudos apoiados nas formas de onda da ULA em questão. Desta forma é possível se analisar os comportamentos práticos e/ou erros ”singulares”como ”Overflow”, ”Underflow”e etc.

Para manter uma ”integridade”na forma de medição e análise dos comportamentos, foram realizados os testes com valores ”similares”aos com a ULA de inteiros.

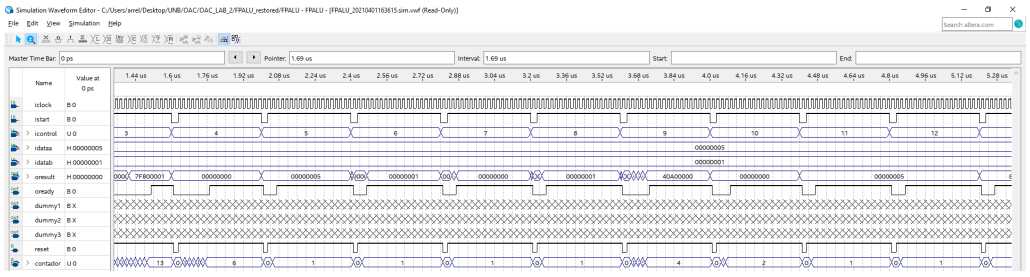

Figura 10. Forma de onda resultante para os valores 5 e 1.

Operação	Caminho de maior atraso	Maior TDP
ADD	idatab[26]	5423
SUB	idataa[3]	3296
MUL	istart	0.966
DIV	istart	2351
SQRT	istart	0.560
ABS	istart	-0.615
CEQ	istart	0.131
CLT	idatab	0.220
CLE	istart	0.796
CVTSW	idataa[4]	4535
CVTWS	idataa[26]	2657
CVTSWU	istart	3209
CVTWUS	idataa[24]	4296
MV	istart	-0.456
SGNJ	istart	-0.456
SGNJN	istart	-0.456
SGNJX	istart	-0.498
MAX	istart	-0.434
MIN	istart	-0.434
NULL	istart	-0.446
ULA completa	idata[24]	7438

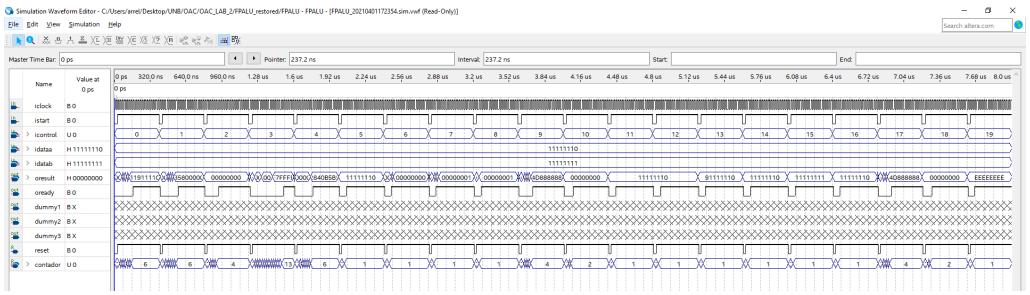

Figura 11. Forma de onda resultante para os valores 254 e 255.

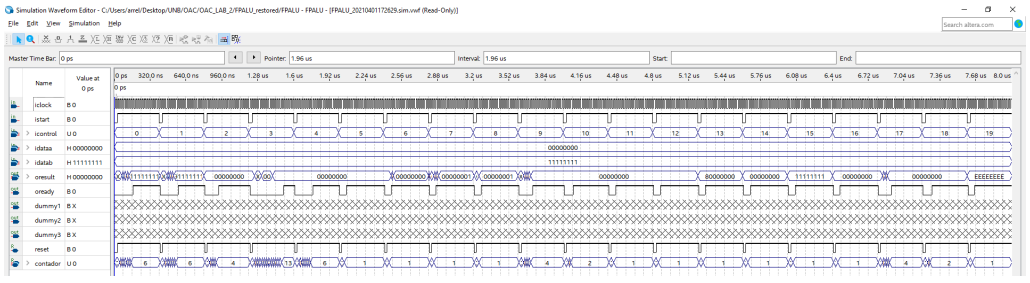

Figura 12. Forma de onda resultante entre os valores 0 e 255.

Para a ULA de ponto flutuante, também foi realizado outros testes, sendo eles a implementação de um sinal de [Wikipedia 2021a] ”clock”no circuito, assim

2.4. Plano para otimização dos requisitos temporais e/ou físicos da ULA para números de ponto flutuante

Assim como foi proposto para as otimizações da ULA de números inteiros, a mesma metodologia foi adotada para a ”FPALU”, desta forma, teoricamente obtendo-se os mesmos comportamentos observados e discutidos com a ULA de inteiros. Visto que as implementações levantadas acabam sendo ”barradas”pelo próprio compilador no processo de otimização temporal.
Desta forma a ULA de números flutuantes acaba se apresentando por:

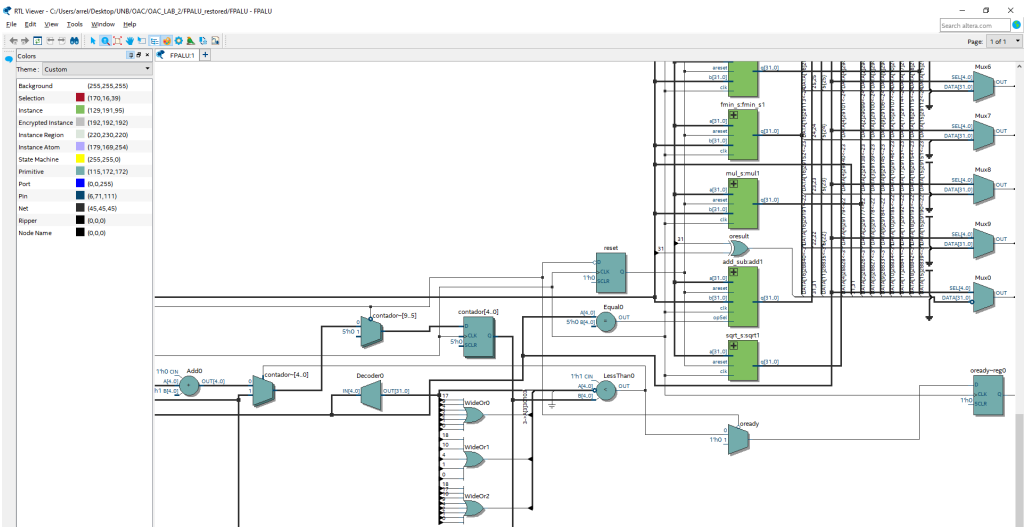

Figura 13. Resultado do processo de compilação do QUARTUS para a FPALU

Referências

[Quartus 2021] Quartus (2021). Quartus. https://www.intel.com/content/www/us/en/programmable/products/design-software/fpga-design/quartus-prime/user-guides.html. [Online; accessed 3-April-2021].

[Wikipedia 2016] Wikipedia (2016). Systemverilog — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=SystemVerilog&oldid=728456019. [Online; accessed 9-July-2016].

[Wikipedia 2021a] Wikipedia (2021a). Clock. https://pt.wikipedia.org/wiki/Sinal_de_rel%C3%B3gio#:~:text=Em%20eletr%C3%B4nica%20e%20especialmente%20em,dois%20ou%20mais%20circuitos%20eletr%C3%B4nicos. [Online; accessed 3-April-2021].

[Wikipedia 2021b] Wikipedia (2021b). Isa. https://en.wikipedia.org/wiki/Instruction_set_architecture. [Online; accessed 3-April-2021].

[Wikipedia 2021c] Wikipedia (2021c). Ula. https://pt.wikipedia.org/wiki/Unidade_l%C3%B3gica_e_aritm%C3%A9tica. [Online; accessed 3-April-2021].