**Universidade Federal da Paraíba**

Drayton Corrêa Filho

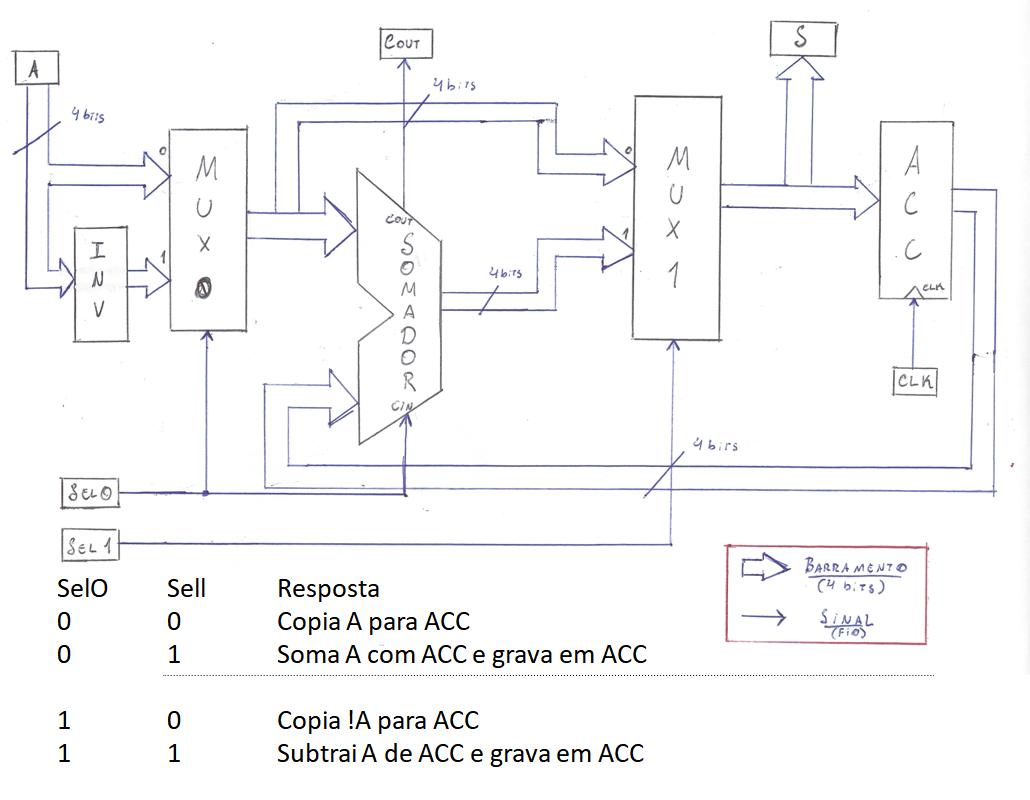
**Relatório da 4ª Avaliação**

**Projeto no Quartus II**

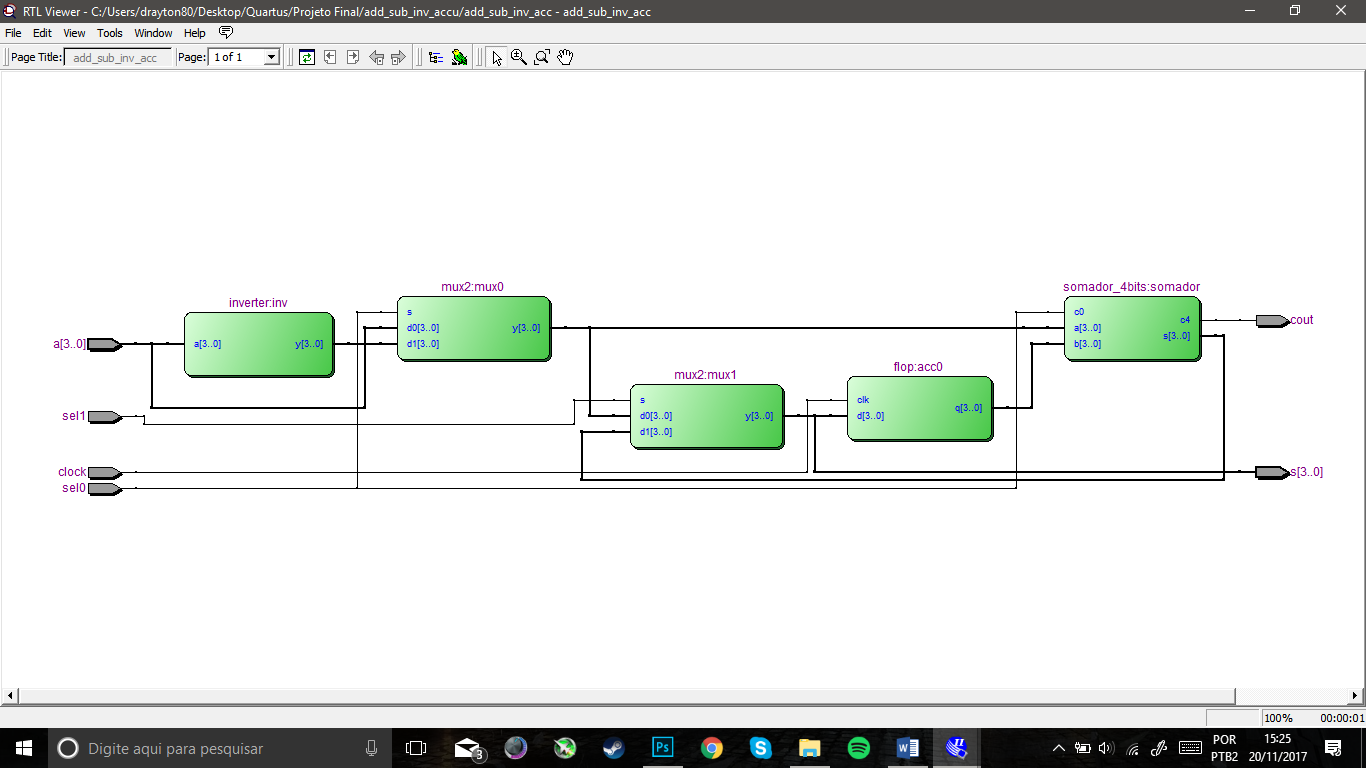
João Pessoa - Paraíba

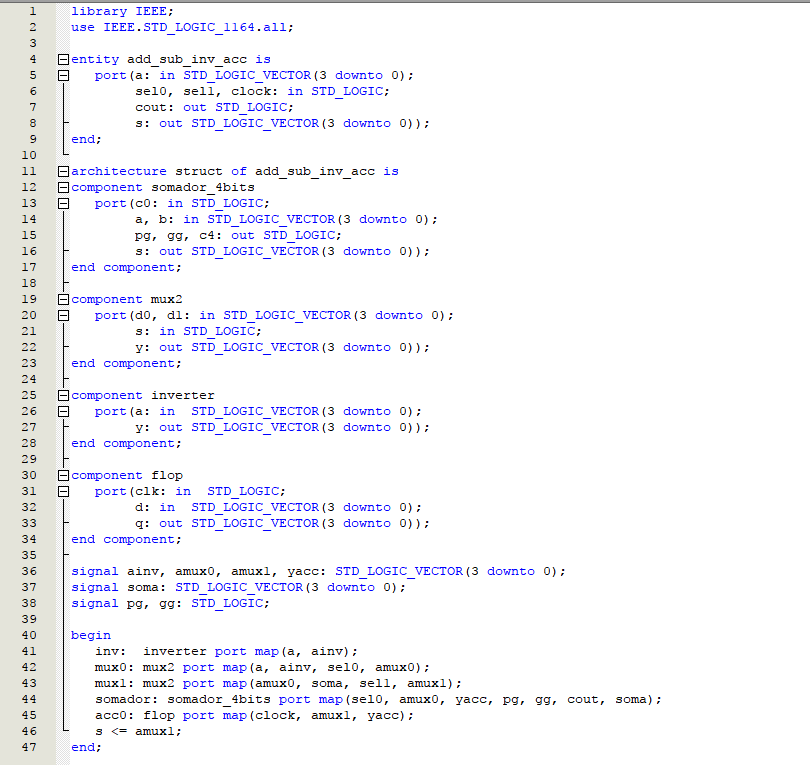
2017

Como último projeto o professor requisitou que fosse feito um operador matemático que poderia somar, subtrair, inverter e acumular um determinado valor. Tal conjunto consistia de dois *multiplexers,* um inversor, um somador de 4 bits e um *flip-flop* que serviria como acumulador. A disposição que deveria ser organizada tais componentes foi fornecida através da imagem a seguir.

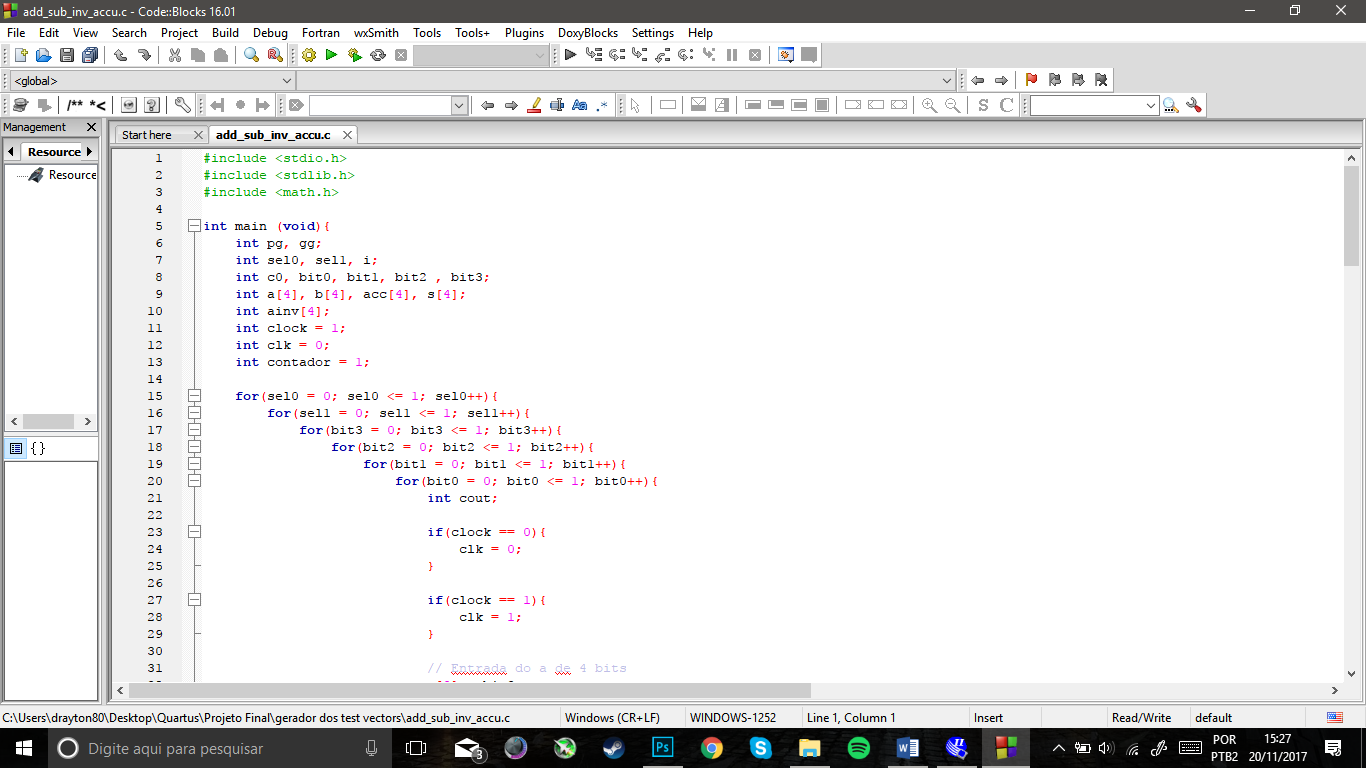
**

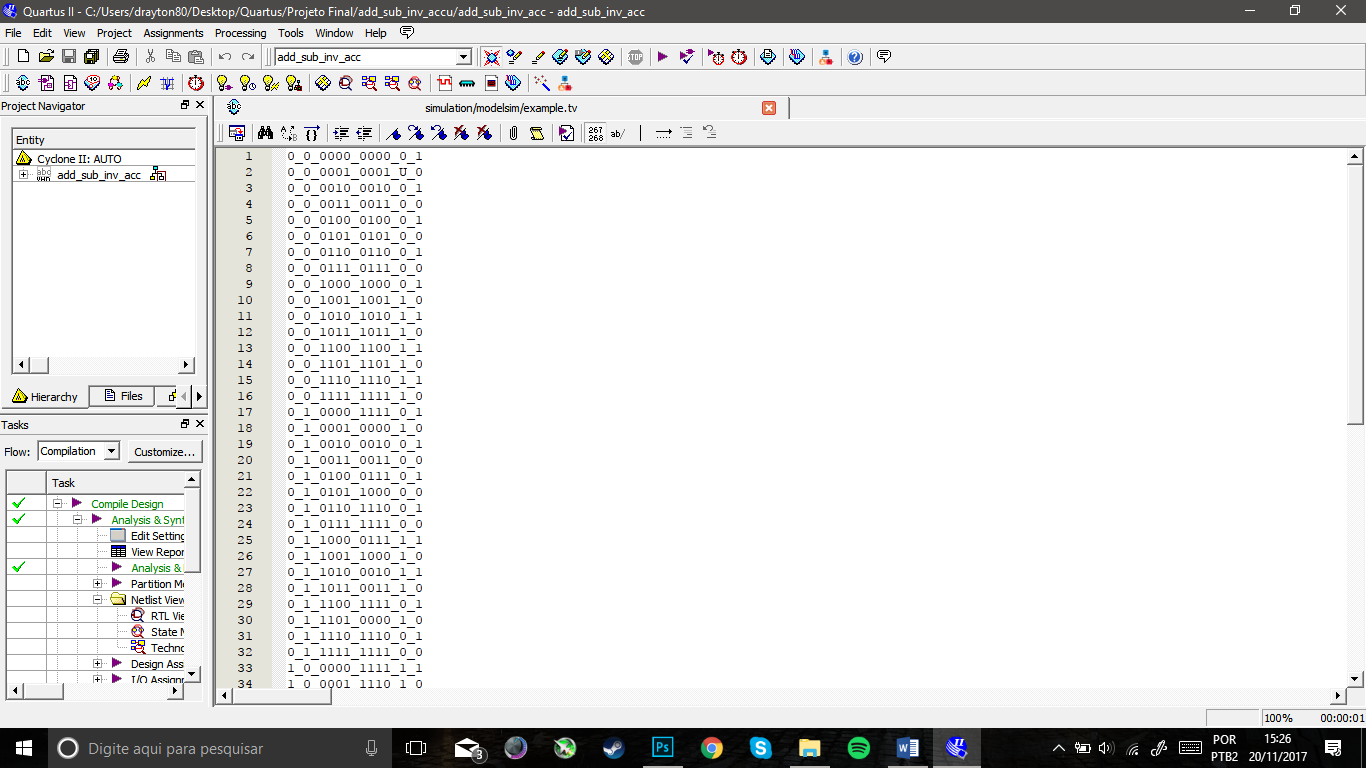
Para executar isso no *Quartus* foi necessário primeiramente buscar os arquivos *VHDL* de antigos exemplos os quais serviriam como *components* dessa aplicação e coloca-los no diretório em que seria posto o *VHDL* de mais alto níveldesse projeto.

Feito isso, criei o *top-level* nomeando-o de *add\_sub\_inv­\_acc* edeclarei no interior de sua *architecture* os *components* para logo após chamá-los de tal forma a que fosse construído uma composição parecida à do desenho acima, utilizando, é claro, de sinais auxiliares que serviram de forma similar a fios ligando os blocos, o código do *top-level* é mostrado na próxima página e a sua visualização *Register-Transfer Level* (feita indo em *Tools, Netlist Viewers* e *RTL Viewer*) é exibida abaixo.

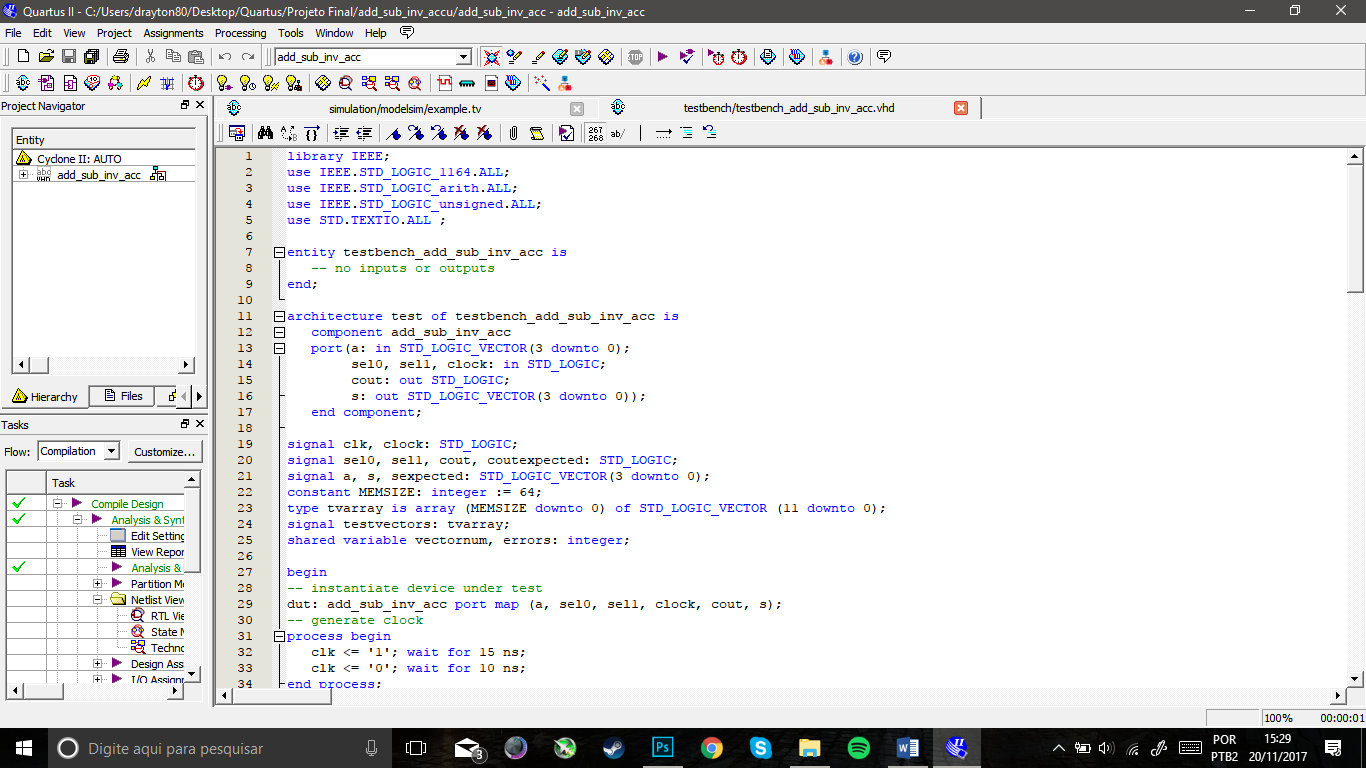


Em seguida, foi o momento de criar os vetores que seriam usados para testar o código acima e, para isso, eu confeccionei um programa em C que basicamente executava da mesma forma que o *add\_sub\_inv\_acc*, mas com uma sequência de entrada preestabelecida.

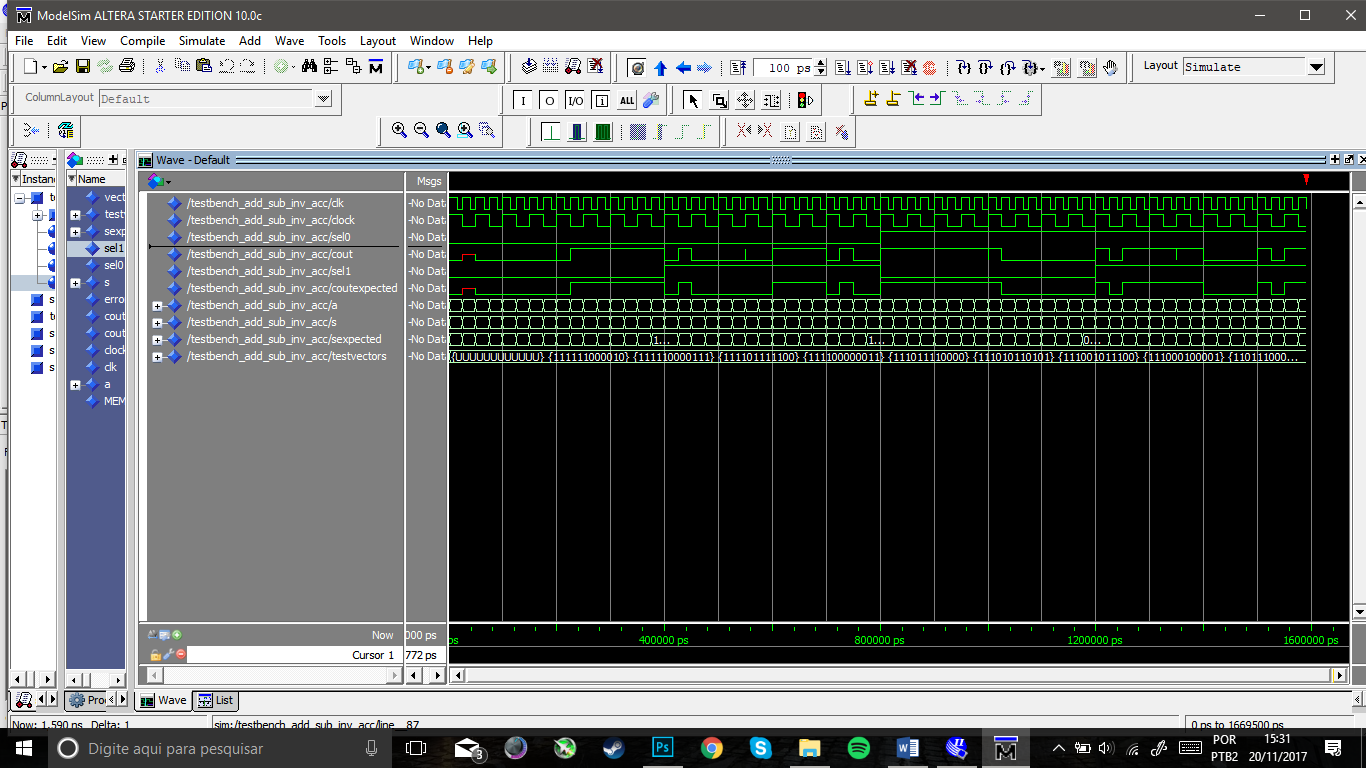


Feito isso, executei-o e copiei os vetores gerados da *prompt* para um arquivo *example.tv* que foi posto no diretório *simulation/modelsim* dentro da pasta do projeto. Um pequeno fragmento desses vetores é exibido ao lado.

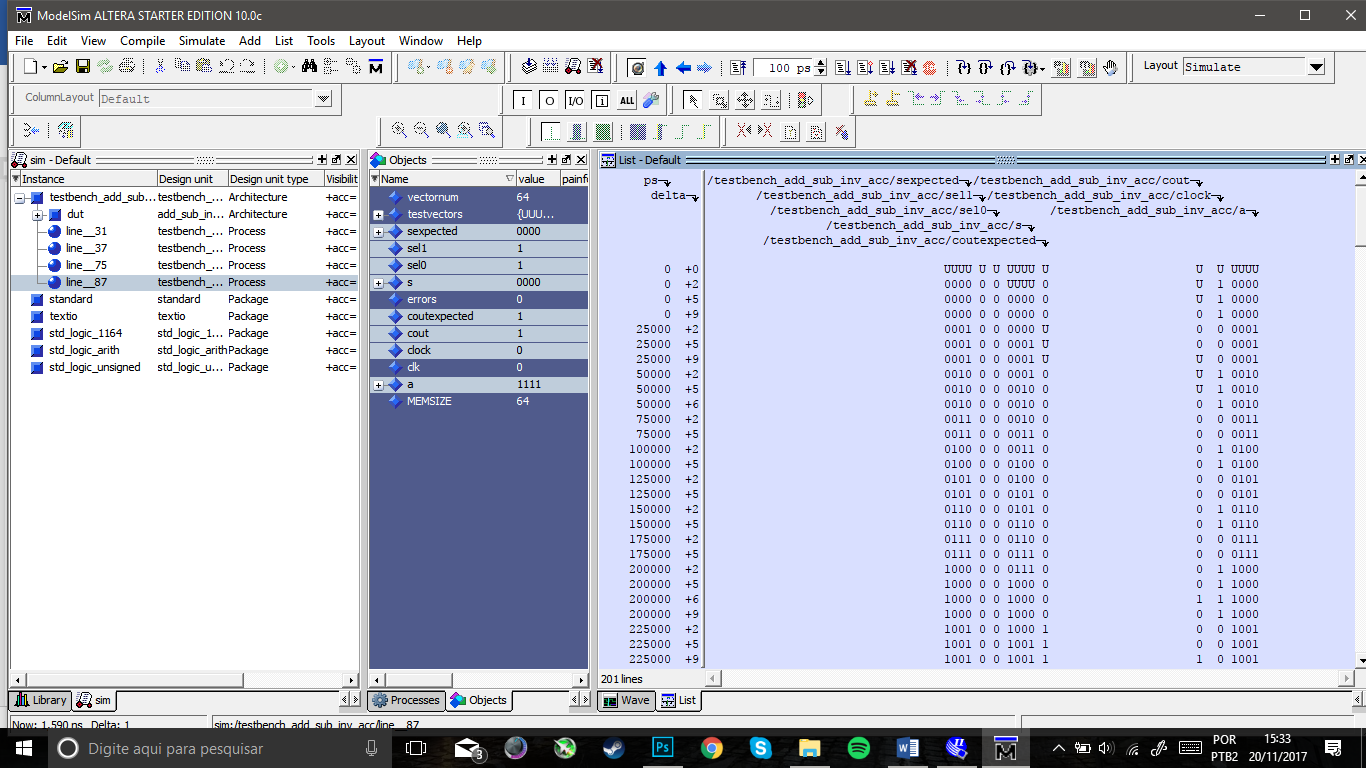
Em seguida, foi o momento de criar o *test bench* que aceitaria esses vetores e, para isso, eu copiei um de exemplos passados e o adaptei, um detalhe importante é que, assim como o *flop* e o *flopenr,* o *add\_sub\_inv\_acc* precisou de um *clock* auxiliar dentro do *testbench* pois em seu interior foi usado um *flip-flop*, já que esse circuito tem uma memória e os valores posteriores de cada linha de teste dependem dos anteriores guardados no acumulador.

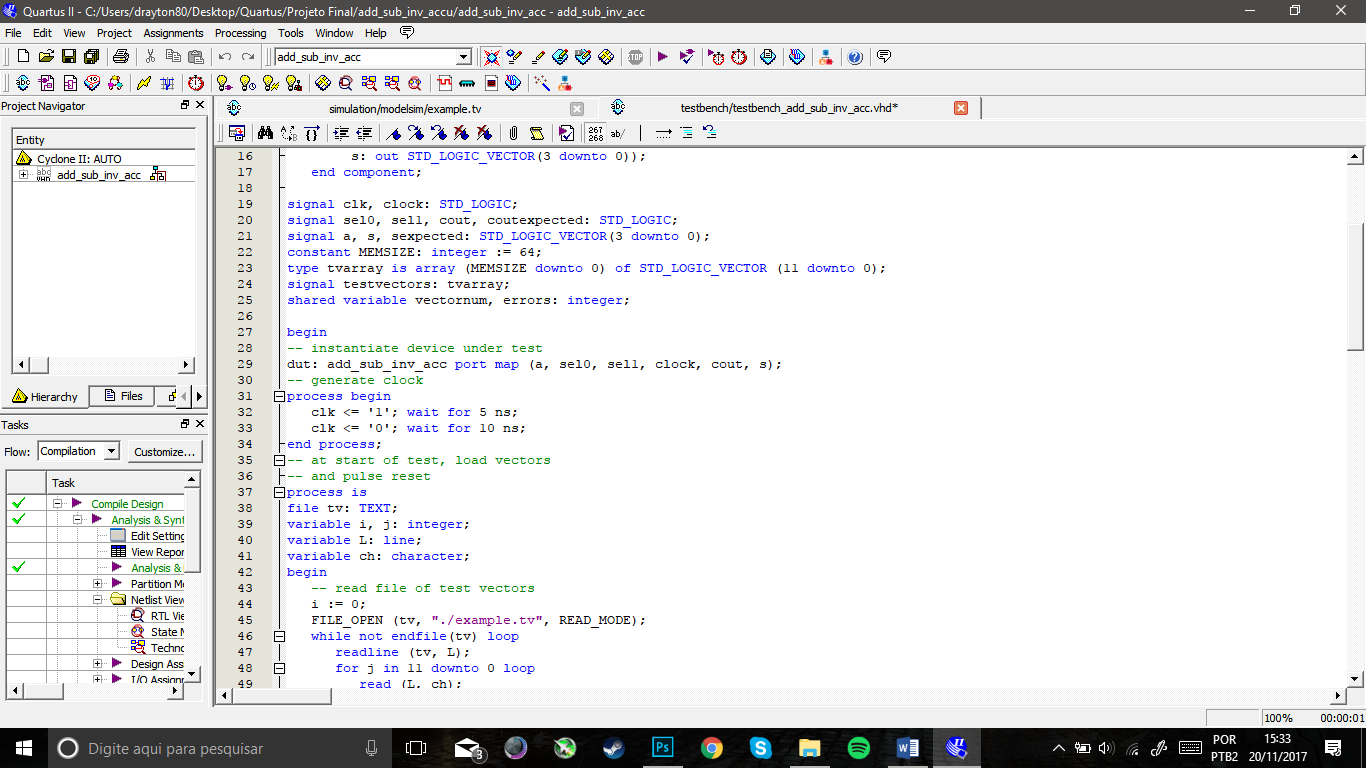


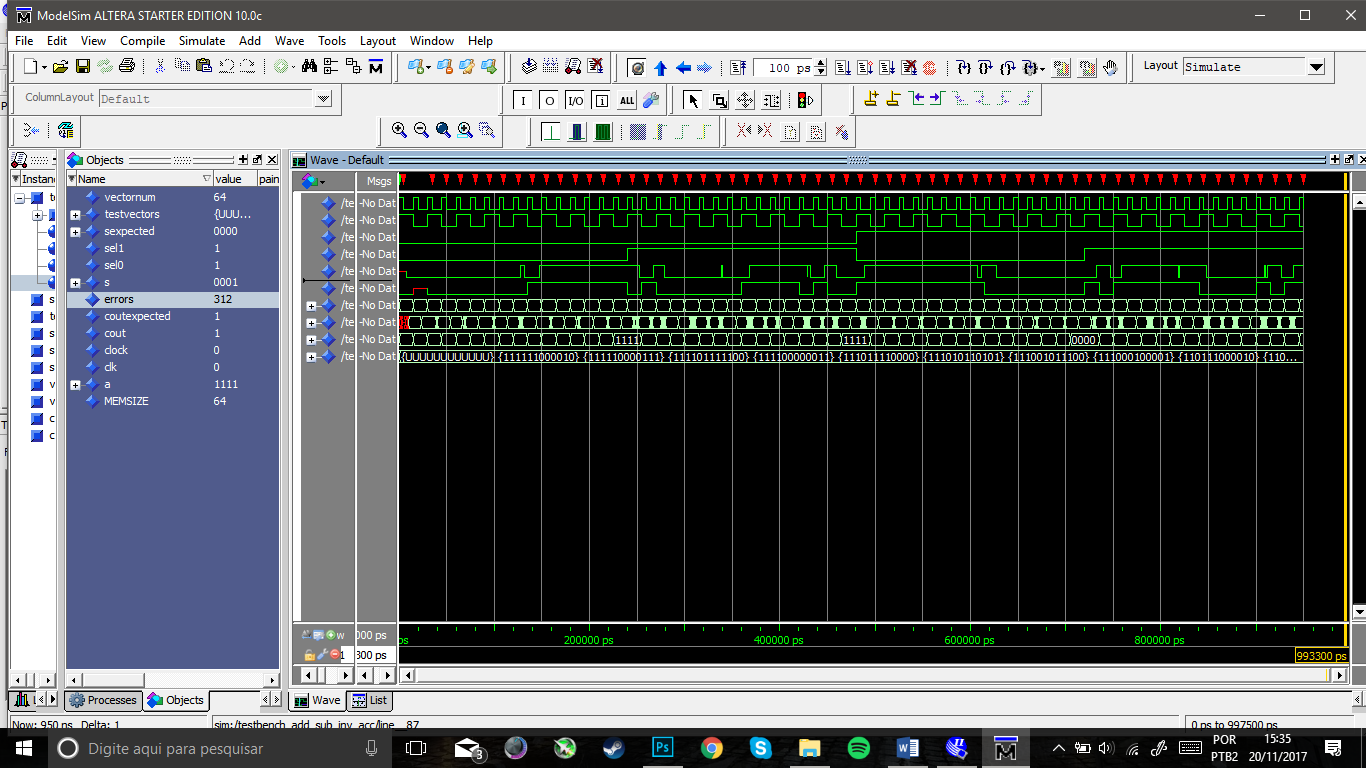
A próxima etapa consistiu de efetivamente testar o projeto rodando a simulação *RTL* indo em *Tools, Run EDA Simulation Tool* e *EDA RTL Simulation*.



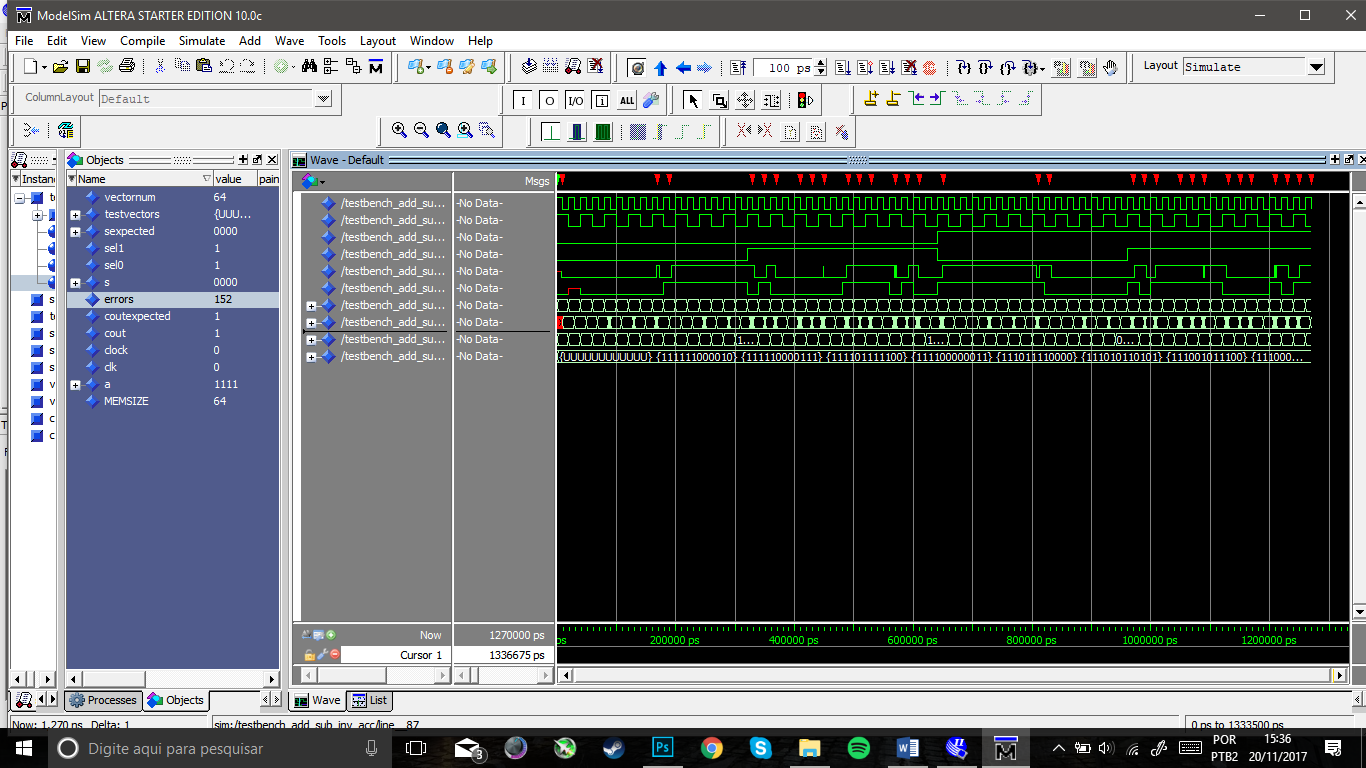
Em seguida, eu selecionei os *signals s, sexpected, sel0, sel1, cout, coutexpected, clock* e *a* e os exibi no formato de lista abaixo.



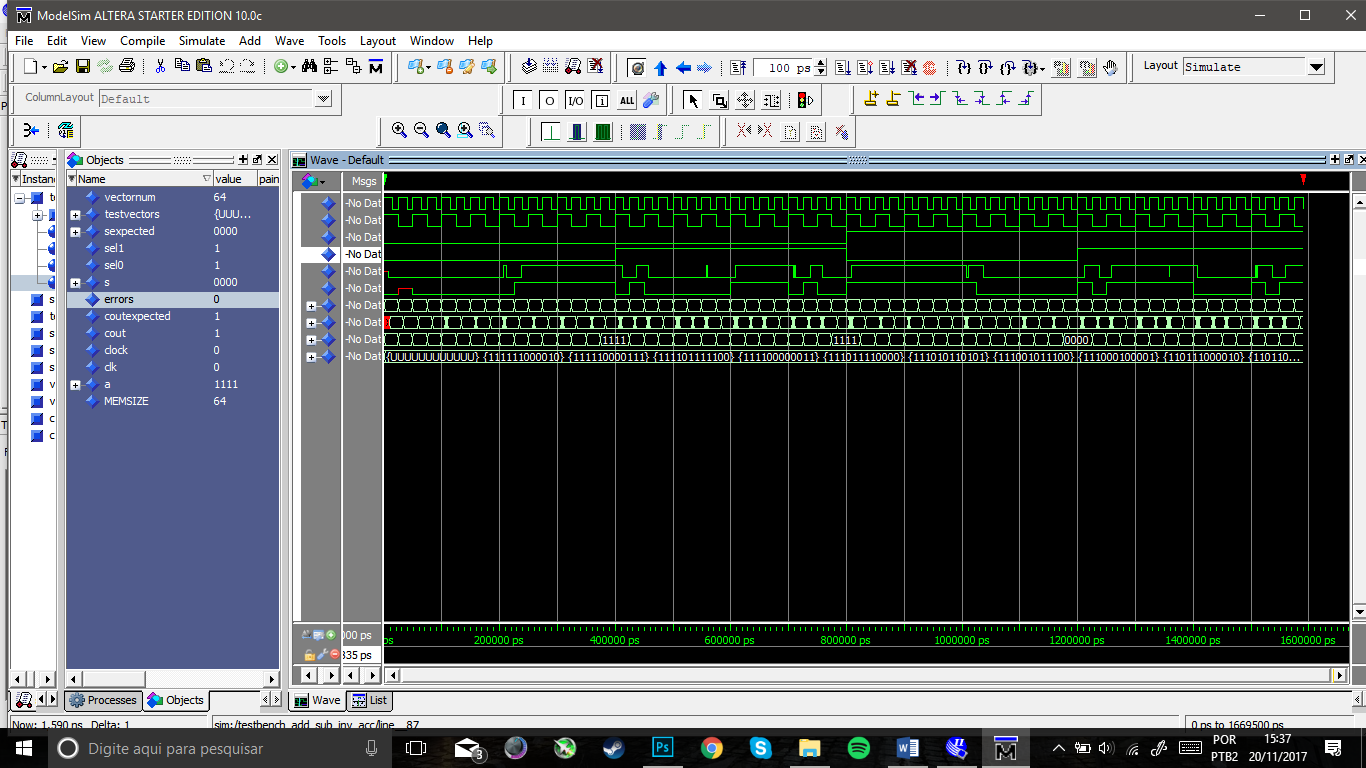
Como é possível observar, a simulação *RTL* não gerou quaisquer erros, então foi possível seguir para o próximo passo, que foi a simulação *gate level*. Mas antes, como sempre feito anteriormente, eu fui no *testbench* do projeto e mudei o tempo de espera de subida do *clk* de 15 ns para 5 ns. Após isso, rodei a simulação.



Com isso foram gerados 312 erros, logo eu subi o tempo de espera para 10 ns, o que produziu o seguinte resultado.



O número de erros decaiu para 152, mas não chegou a 0, então foi preciso subir o tempo de espera para 15 ns.



O que finalmente fez o número de erros chegar em 0, sendo possível concluir que 15 ns é o tempo de espera mínimo para que não sejam gerados erros.