Documentação do Trabalho Prático de OC1

Prof.: Antônio Otávio Fernandes
Monitor: Fernando Carvalho da Silva Coelho

Alunos:
Ronald Davi Rodrigues Pereira - 2015004437
Luiz Otávio Resende Vasconcelos - 2015042142
Richard Nikolas Chaves - 2015042185

➢ 1 - Introdução

O objetivo deste trabalho é a implementação de um processador multi ciclo de 16 bits com instruções de tamanho fixo, que utilizará 8 registradores e será capaz de realizar operações aritméticas simples. A linguagem utilizada para implementar este processador foi o Verilog. O processador implementado sempre executa uma instrução em um intervalo de 4 ciclos de clock, dessa forma, todas as instruções devem estar prontas na entrada de dados, exatamente ao final dos 4 ciclos de clocks necessários para a execução de uma instrução.

➢ 2 - Implementação

- **ALU**

O ALU (Arithmetic Logic Unit) é o circuito digital que realiza operações lógicas e aritméticas. Neste caso, o ALU tem dois inputs de 16 bits: **a** e **b,** o input **OpSelect** de 2 bits (este selecionará a operação a ser feita) e o output **reg** de também 16 bits. A função sempre será executada, pois always(*) sempre é verdadeiro, o case(OpSelect) funciona como um

“switch”. Se OpSelect = 0, é realizada a soma; se OpSelect = 1, é realizada a subtração; se OpSelect = 2, funciona como uma porta NAND; se caso OpSelect não for nenhum desses valores, a função retorna 0.

- **Complement**

Módulo responsável por realizar o complemento de dois de alguns registradores. Ele utiliza de uma simplificação da ALU, de modo a negar um dos registradores, para realizar a operação de SUB a partir de apenas um ADD.

Essa negação do registrador acontece a partir de um sinal enable, que dirá ao módulo se ele deve calcular o complemento de dois do registrador ou somente manter o valor dele ao atribuí-lo a saída.

- **Contador**

Unidade responsável por determinar o estágio do processador.

Recebe o clock, o clear e um *wire* de saída.

A cada pulso de clock, verifica se o clear está ativo, se sim, zera o contador, se não, incrementa a saída (out).

- **Extensor**

Separa o valor do Imediato da instrução antes do Multiplexador.

Recebe a entrada de instruções (DIN) de 16 bits.

Declaramos então dois *outputs regs:*

output reg [15:0] **out**;
output reg [9:0] **imediate**

Pegamos as 10 primeiros valores do DIN (**iin** imediate) e colocamos nas 10 primeiras posições do **out**, completamos com 0’s e retornamos.

- **Multiplexador**

Recebe os sinais de select dos registradores e o **iin** (DIN) após ser passado pelo Extensor de Sinal já com os valores de operações zerados (formato 00000 XXXXXXXXXXX).

Passando por uma estrutura condicional, retorna o valor de entrada no **bus**:

output reg [15:0] **bus**;

Que posteriormente será enviado para o **bus**, para a **ULA** e para os **registradores**.

- **Banco de Registradores**

Valida duas condições para que o valor possa ser escrito no registrador (**out**): se o clock estiver na borda de subida (**posedge**) e se o bit do reg está ativado (**enable**). Se essas duas condições forem verdadeiras, o registrador recebe o valor da entrada de dados.

- **Decoder**

O Decoder tem um input de 2 bits (**register**) e um output **reg** de 8 bits. Sempre que houver sinal do input (always@(register)), a função será iniciada. Aqui temos outro case(register), que funciona como um switch, e o decodificador serve para marcar quais registradores estão ativados naquele ciclo.

Por exemplo, se ele receber o sinal “3’b000”, atribui o valor 1 à posição out[0], logo o registrador out[0] está ativado/”enable”.

- **Unidade de Controle**

A unidade de controle realiza a comunicação com a maioria dos componentes. Ela também determina, a partir do valor recebido do contador, qual estágio de ciclo o processador está.

Conforme cada ciclo vai sendo executado, ele faz a chamada sequencial dos demais componentes do datapath, de modo a integrar e controlar todos os outros módulos do projeto. Além disso, a unidade de controle também é responsável por zerar o contador a partir do sinal de **resetn**.

➢ 3 - Testes

Cada pasta, exceto a da unidade de controle e do multiplexador, possuem 3 arquivos, um código fonte do módulo em Verilog, um testbench para o testar o módulo e um Makefile para compilação e execução rápida dos testes.

Para isso, basta digitar

$ make run

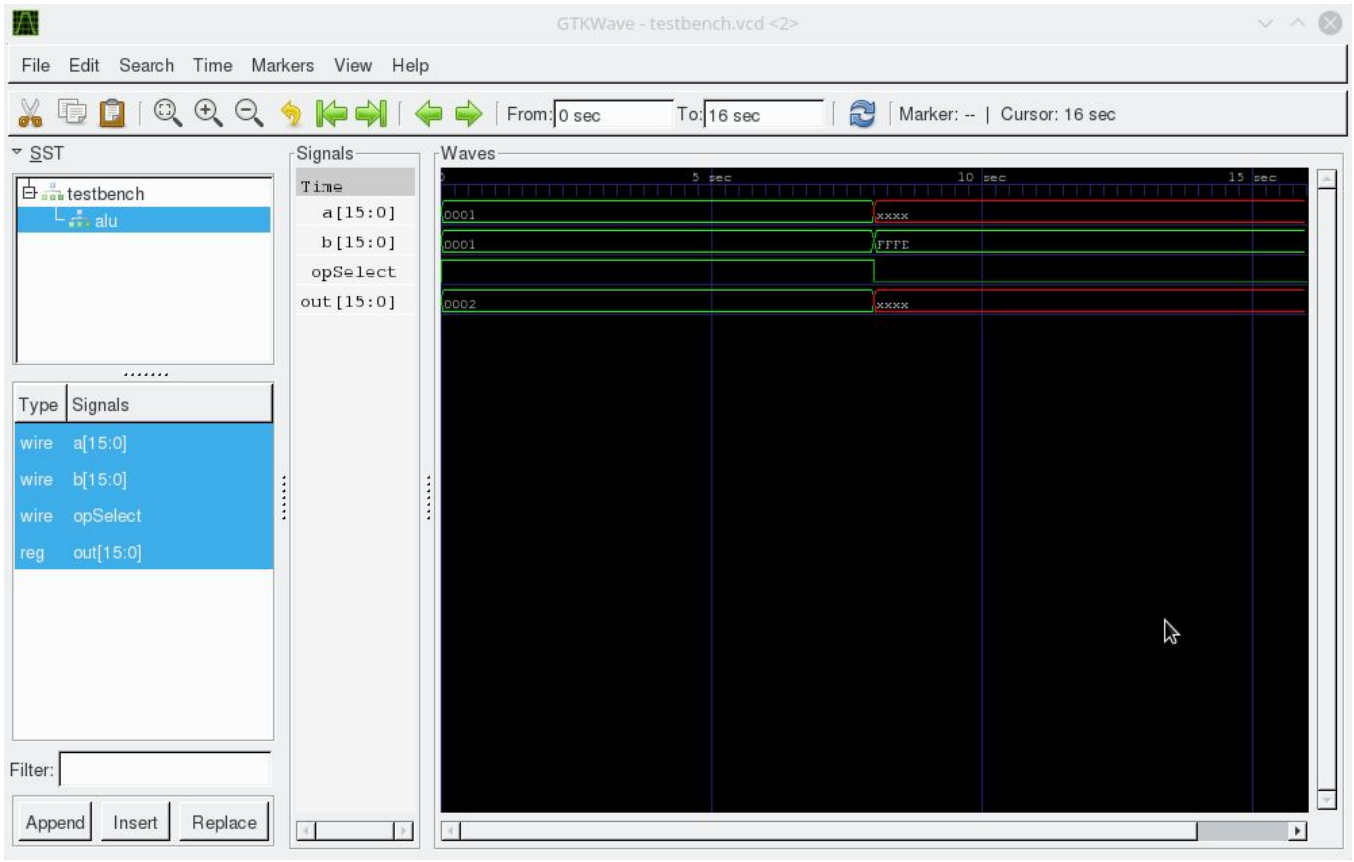
no terminal que será feita a compilação dos códigos fonte e aberto uma janela do GTKWave contendo o teste da pasta onde o Makefile se localiza.

Vale ressaltar também que existe um comando

$ make clean

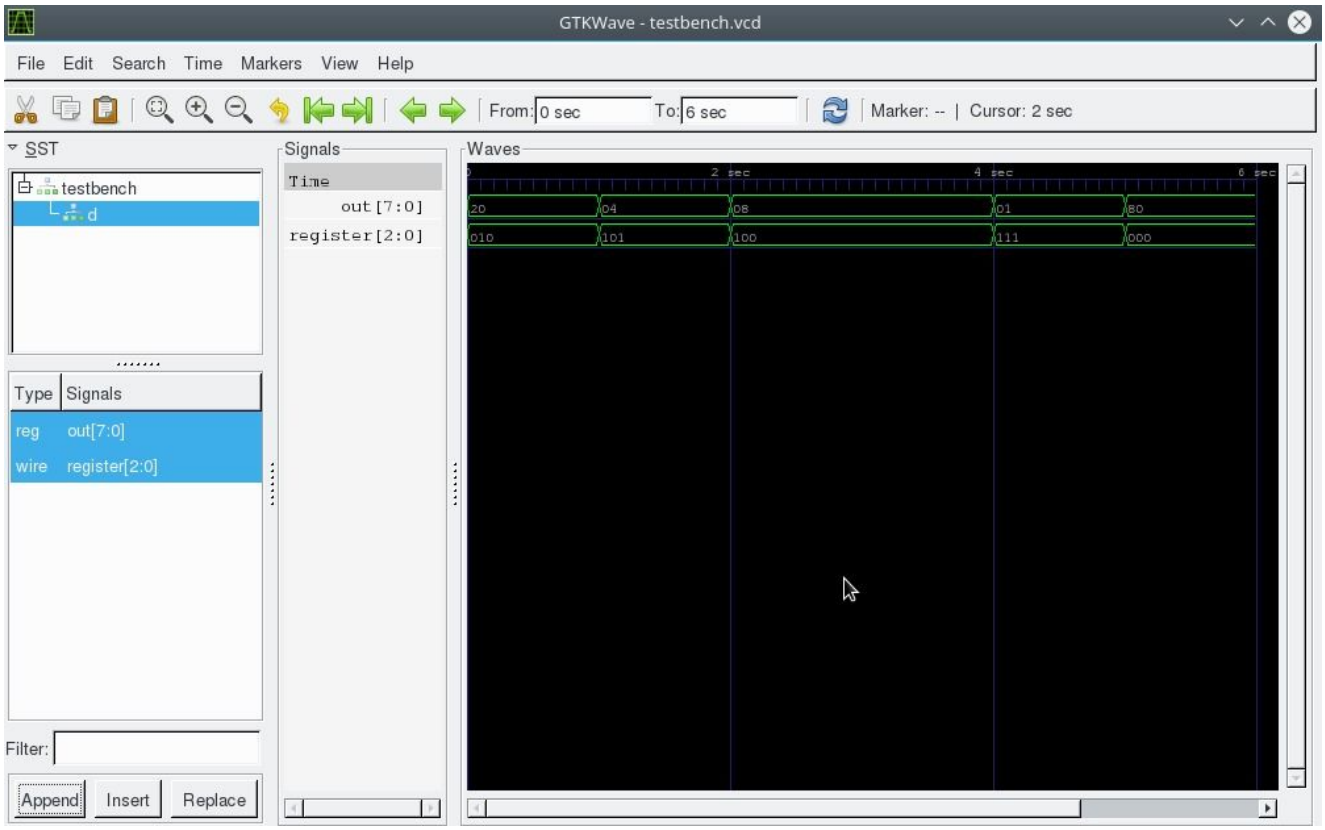
que remove todos os dumpfiles gerados (a.out e testbench.vcd) para a execução do teste.

- **ALU**



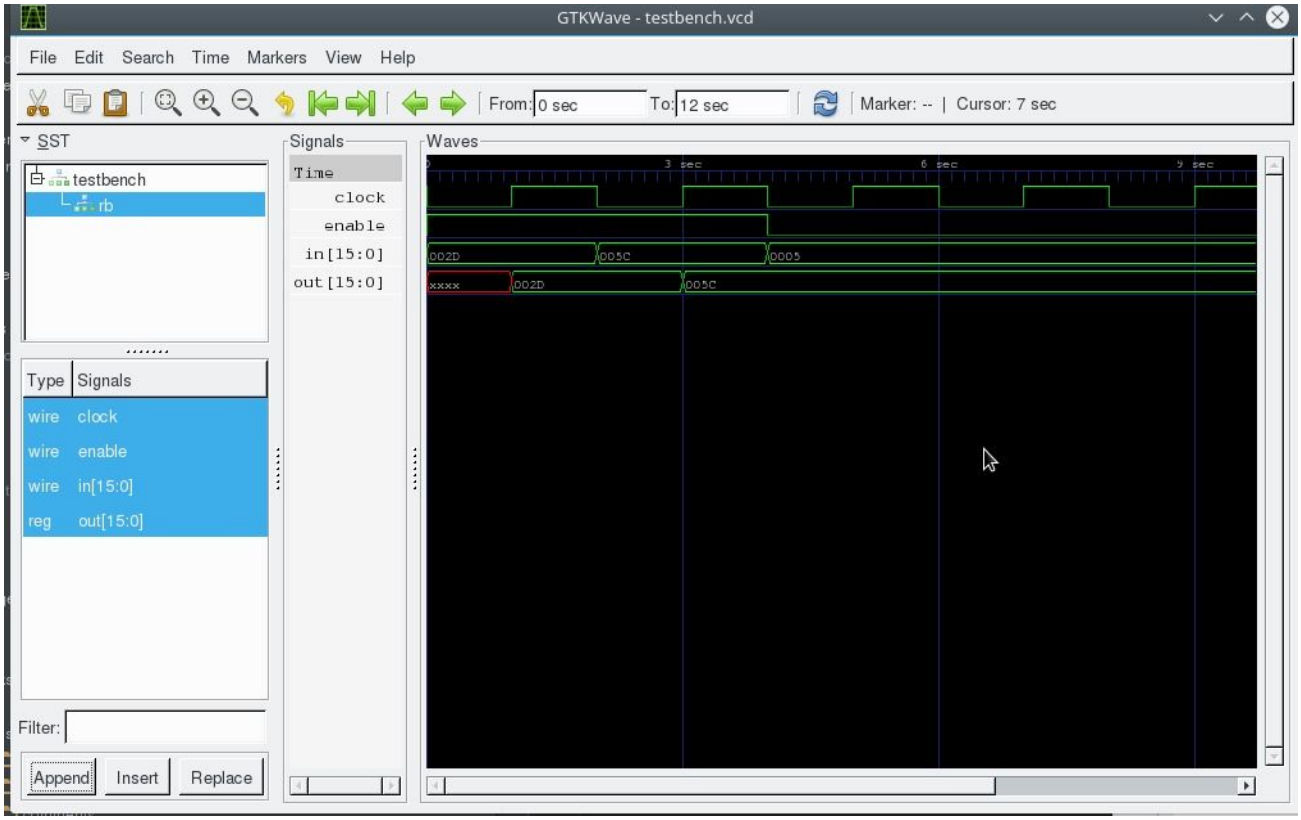
No primeiro teste *setamos* **a** e **b** = 1 e **opSelect** = **1'b1** que é a operação de soma.
No **out** temos então a soma de **a** e **b** = **2.**

- **Decoder**



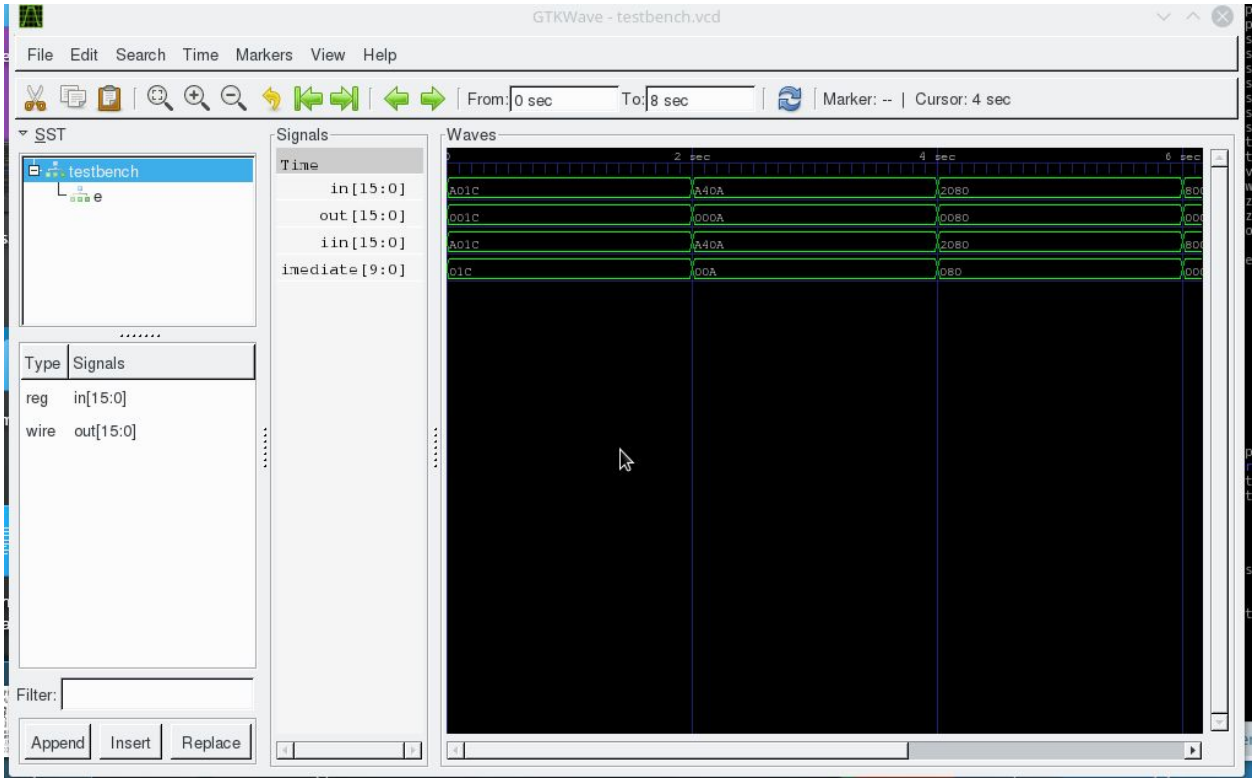
Neste teste, o decoder recebeu os seguintes sinais: 010, 101, 100, 111, 000. Logo, os registradores **out[2], out[5], out[4], out[7]** e **out[0]** estão ativados/enable.

- **Banco de Registradores**



Apenas em dois momentos são cumpridas as condições necessárias para escrever as informações nos registradores de saída (posedge e enable = 1), as saídas resultantes foram **002D** e **005C**, que eram os valores de entrada no momento de *setar* o valor.

- **Extensor**



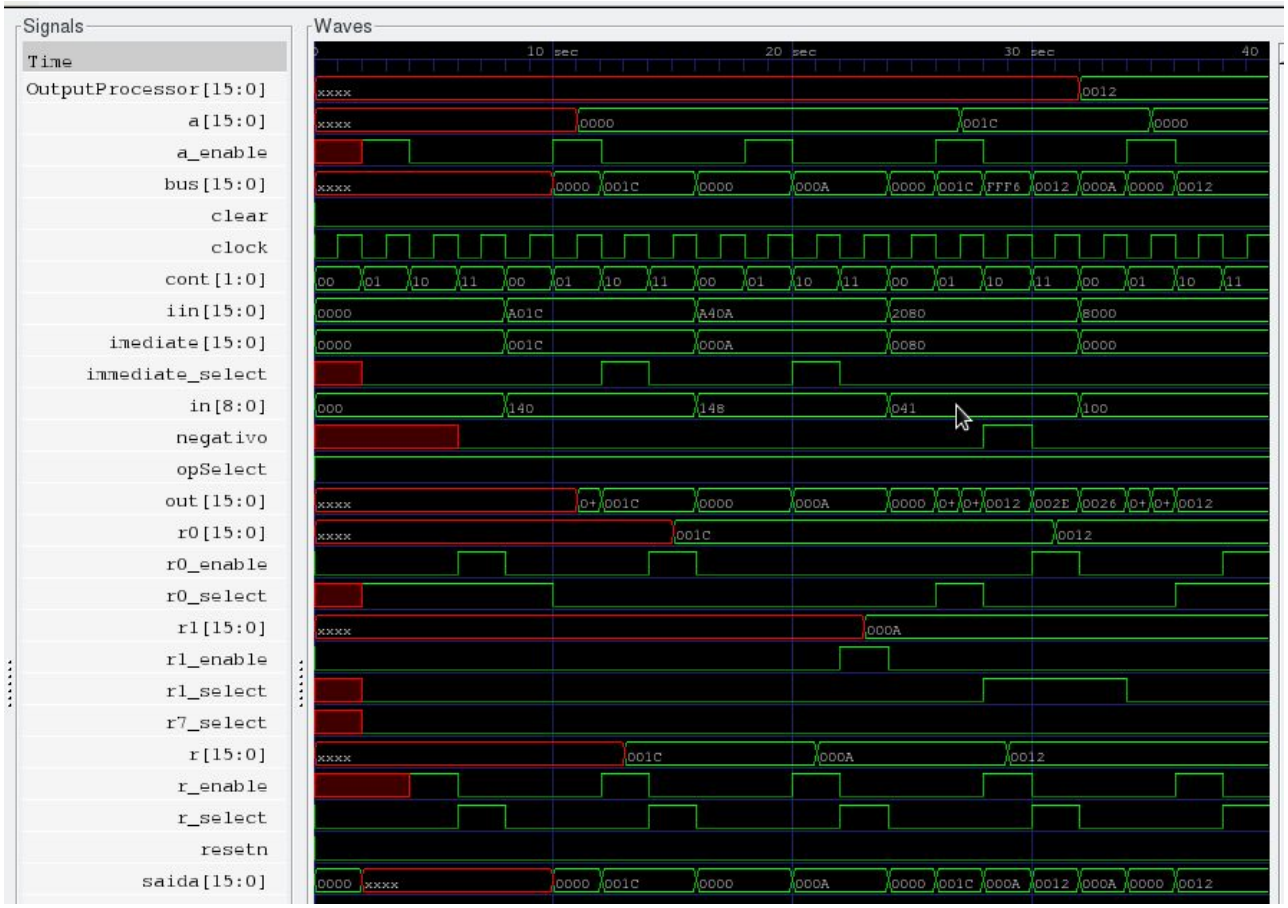
Recebe o **iin** (DIN) e remove os 6 primeiros bits (identificador de função => 15 até 10).

Para isso, ele pega os bits de 9 à 0 do **iin** e joga no **imediate**.

No primeiro valor de entrada, o **iin** é setado em 0xA01C, então, podemos ver o **imediate** sendo setado com 0x01C (sem o primeiro bit hexadecimal A do **iin**) nos primeiros 2 segundos.

Então, o **out** é *setado* com o **imediate** antecedido de 0 => 0x**0**01C, para então ser utilizado pelo Multiplexador.

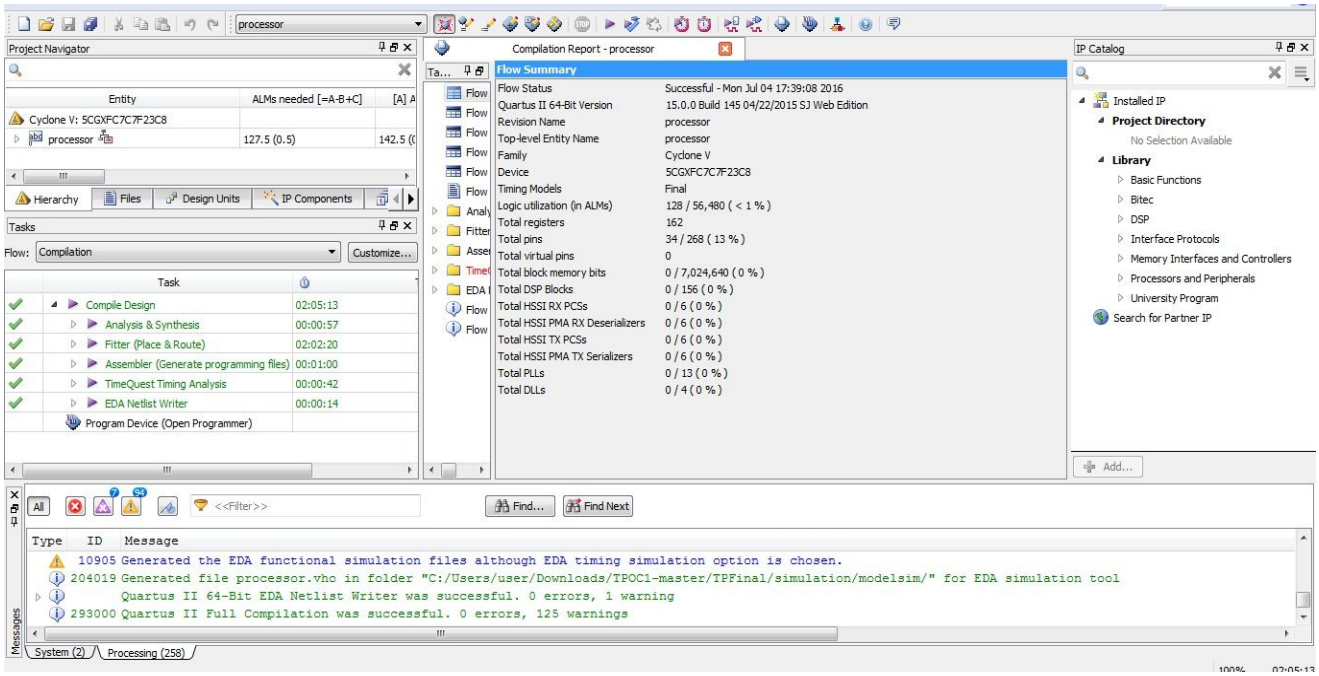
- **Processador**



Teste final do processador, usando todos os módulos e funções implementados. (Alguns registradores foram ocultados para reduzir a imagem, pois, não são utilizados nos testes)

Analisando os valores encontrados na saída dos registradores e a saída da forma de onda, tudo funcionou corretamente.

→ **Teste feito no Quartus II**



Para que fosse possível executar o código na placa FPGA Cyclone V, ocorreram algumas mudanças de sintaxe no código fonte do programa, pelo fato do Quartus II somente aceitar versões mais antigas da linguagem Verilog.

Com o teste feito acima, o código compila perfeitamente no Quartus II Web Edition, de modo que já estava configurada a placa utilizada no laboratório (Cyclone V: 5CGXFC7C7F23C8), atestando que o programa executaria perfeitamente em uma placa FPGA do modelo especificado acima.

A pasta com os arquivos gerados pela compilação está sendo enviado juntamente com o trabalho na pasta “TPFinal - Quartus II”.

➢ 4 - Conclusão

Neste trabalho colocamos em prática vários assuntos discutidos e aprendidos em curso. Pudemos praticar implementação em Verilog, modularização, análise de forma de onda e o funcionamento de um processador básico da arquitetura MIPS.

O maior desafio foi, após o desenvolvimento, integrar todos os módulos de modo a fazer o datapath ser semelhante à arquitetura MIPS. Porém, mesmo assim conseguimos concretizar o projeto e todas as saídas estão conforme o esperado pelos testes.