UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIA E TECNOLOGIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DCC301 - ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

PAULO FERREIRA DA SILVA JUNIOR
FERNANDO SOUZA RODRIGUES

RELATÓRIO DO PROJETO ONE PIECE: COMPONENTES

BOA VISTA, RR
2022

UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIA E TECNOLOGIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
DCC301 - ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

RELATÓRIO DO PROJETO ONE PIECE: COMPONENTES

Relatório apresentado como para obtenção de nota na disciplina de Arquitetura e Organização de computadores, ofertada pelo curso de Ciência da Computação da Universidade Federal de Roraima. Prof. Dr. Herbert Oliveira Rocha.

BOA VISTA, RR
2022

RESUMO

O seguinte trabalho aborda o projeto que visa a implementação de componentes em MIPS utilizando uma arquitetura alternativa voltada para 8 bits, em consonância com a sua versão mais utilizada de 32 bits. Ao longo do trabalho serão descritos todos os estágios de funcionamento e desenvolvimento dos componentes, assim como as waveforms e resultados de testes implementados em cada componente. Ao todo, são 13 instruções divididas em três formatos de instruções, que foram construídas utilizando a linguagem VHSIC Hardware Description Language, conhecida como VHDL.

Palavras-chave: MIPS, 8bits, processador, VHDL.

SUMÁRIO

LISTA DE IMAGENS

1. ESPECIFICAÇÃO

Nesta seção é apresentado o conjunto de itens para o desenvolvimento dos componentes, bem como a descrição detalhada de cada etapa da construção. Em cada tópico, será abordado de maneira aprofundada os componentes e o que os compõem, visando simplificar e facilitar o entendimento da construção de um processador MIPS.

2. PLATAFORMA DE DESENVOLVIMENTO

Para a implementação de cada componente foi utilizado a IDE Intel Quartus Prime Lite Edition, na versão 20.1.

Quanto ao simulador utilizado, optou-se pelo uso do simulador ModelSim Altera, utilizando a linguagem VHDL para a construção dos códigos.

2.1. CONJUNTO DE INSTRUÇÕES

Foram utilizados 3 formatos de instruções de 8 bits cada, Instruções do tipo R, I, J, seguem algumas considerações sobre as estruturas contidas nas instruções:

- Opcode: a operação básica a ser executada pelo processador, tradicionalmente chamado de código de operação;
- Reg1: o registrador contendo o primeiro operando fonte e adicionalmente para alguns tipos de instruções (ex. instruções do tipo R) é o registrador de destino;
- Reg2: o registrador contendo o segundo operando fonte e que também é utilizado para realizar operações do tipo I, voltada para uso de operações imediatas.

2.2 TIPOS DE INSTRUÇÕES

Dentro da arquitetura MIPS, temos os seguintes tipos de instruções:

TIPO R: este formato é voltado para instruções que não requerem um endereço de destino, valor imediato ou mesmo deslocamento de bits. Com ele, podemos realizar operações aritméticas, como soma, subtração e multiplicação, além de operações lógicas. Originalmente, temos as divisões em seis campos (isso para o formato de 32 bits). Contudo, em uma adaptação para 8 bits, temos:

4 BITS	2 BITS	2BITS
7-4	3-2	1-0
OPCODE	REG1	REG2

IMAGEM 1 - FORMATAÇÃO DE INSTRUÇÕES DO TIPO R

TIPO I: este formato permite operações imediatas, além de um operações de memória e manipulação de desvios condicionais. Nela, podemos realizar soma e subtração de forma imediata, não sendo necessário armazenar o valor dentro de um registrador e sim passar o número desejado diretamente, por exemplo. Temos então sua forma de escrita:

4 BITS	2 BITS	2BITS
7-4	3-2	1-0
OPCODE	REG1	REG2

IMAGEM 2 - FORMATAÇÃO DE INSTRUÇÕES DO TIPO I

TIPO J: Realiza apenas instruções de salto que ocorrem sempre que a instrução que o especificou é executada, também conhecidas como desvios incondicionais. Como exemplo, temos a instrução Jump. Sua forma de escrita fica assim:

4 BITS	4 BITS
7-4	3-0
OPCODE	ENDEREÇO

IMAGEM 3 - FORMATAÇÃO DE INSTRUÇÕES DO TIPO J

3. DESCRIÇÃO DOS COMPONENTES

Passaremos a descrever cada um dos componentes. Cada tópico tem como objetivo descrever o funcionamento dos trilhos e sua importância para o funcionamento geral do projeto, seguindo uma certa ordem de complexidade.

3.1 ULA

A ULA significa Unidade Lógica e Aritmética (ou em inglês, Arithmetic Logic Unit, por isso ALU) e é a parte do computador responsável por realizar operações lógicas e aritméticas sobre os dados. É a principal responsável por gerenciar partes fundamentais do computador, desde uma pequena conta de soma de bits até mesmo operações que envolvam um conjunto mais complexo de operações matemáticas. Os demais componentes trabalham em função de fornecer dados para que a ULA processe e tome prosseguimento do que deve acontecer em seguida. Dentro dela, podemos realizar operações de soma de bits, subtração, multiplicação e operações booleanas AND, OR, XOR e NOT por exemplo. É o núcleo do computador. Ela recebe instruções de entrada com os dados provindos de outros componentes

e gera os resultados a serem aplicados no resto do processador. A visão geral de nossa ULA possui este formato:

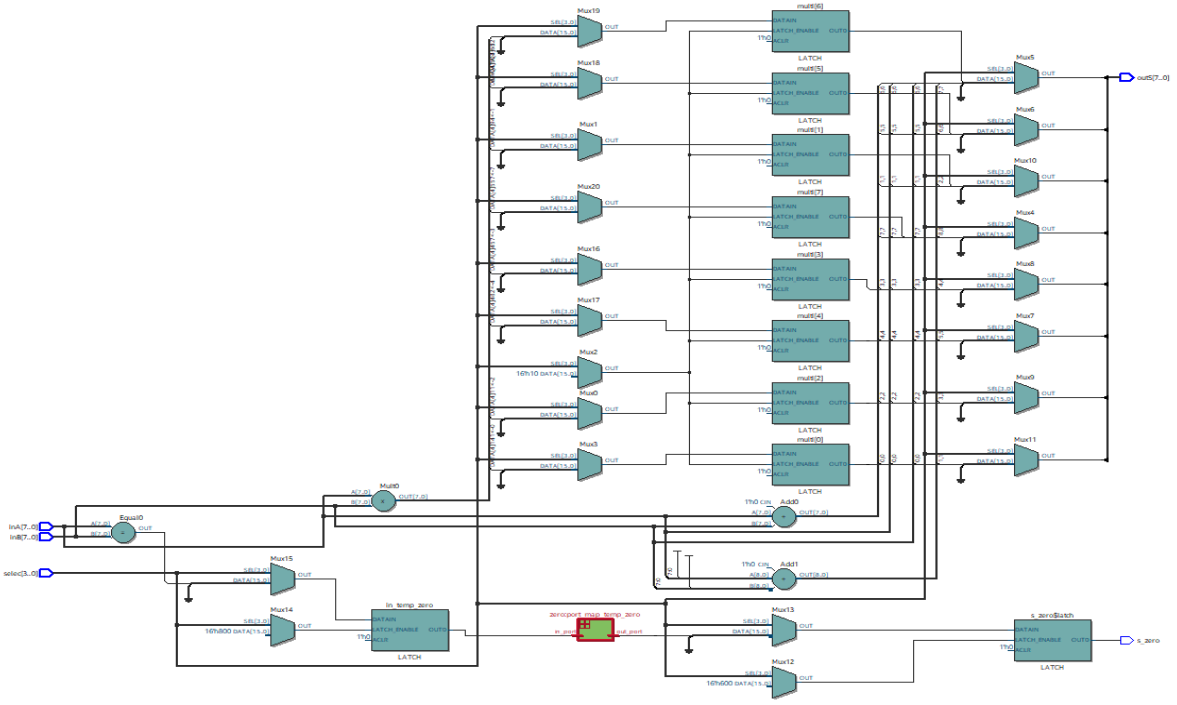
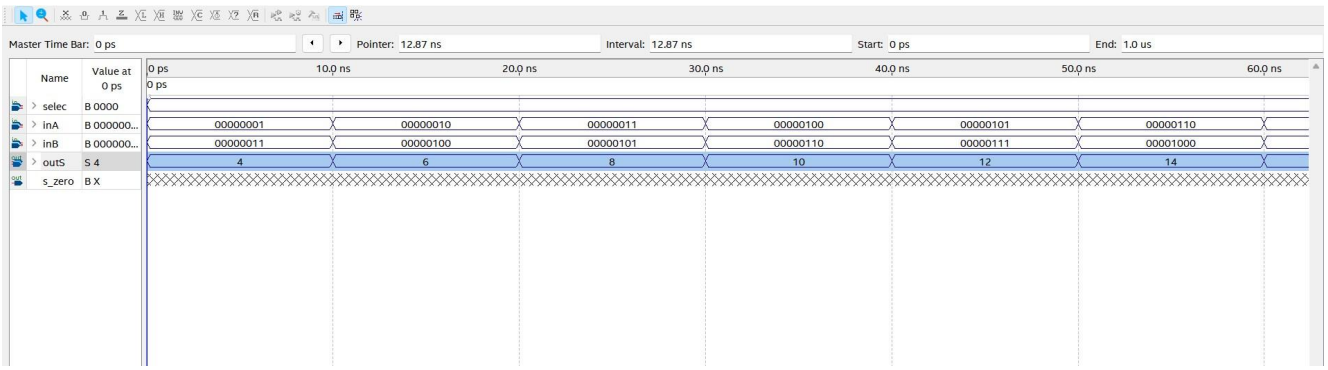


IMAGEM 4- RTL Viewer da ULA

Em nossa ULA, recebemos valores provindos do banco de registradores que possuem um tamanho de 8 BITS e que servirão como variáveis X e Y para a realização das operações. Além disso, há uma entrada voltada para a entrada da flag ALUop que é responsável por comunicar qual Opcode está sendo inserido, indicando quais operações serão realizadas. Possuímos ainda o componente denominado zero (explicado mais abaixo) também está inserido dentro do componente da ULA. Os resultados de saída gerados pela ULA podem ser armazenados em dois lugares: Memória RAM ou no banco de registradores.

TESTE:



Neste teste aplicamos a porta **selec** o valor 0000 para selecionar a operação de adição entre as portas **inA** com valor de 00000001 e **inB** com valor de 00000011, 8 bits cada, cujo o resultado vai para a porta **outS** sendo 4.

3.2 BANCO DE REGISTRADORES

Os registradores precisam ser alocados em algum local. O banco de registradores se responsabiliza por isso. Seu funcionamento é simples: ele recebe duas entradas de registradores provindas da divisão de instrução, cada uma possuindo 2 bits de espaço. Essas instruções podem ser repassadas para a ULA, logo, gera duas saídas de Bits que são repassadas para ULA. Como dito anteriormente, os resultados da ULA podem ser gravados novamente no banco de registradores, como em um caso de operação de soma, onde, por conta da arquitetura do MIPS, podemos escrever os dados diretamente da ULA para o banco de registradores.

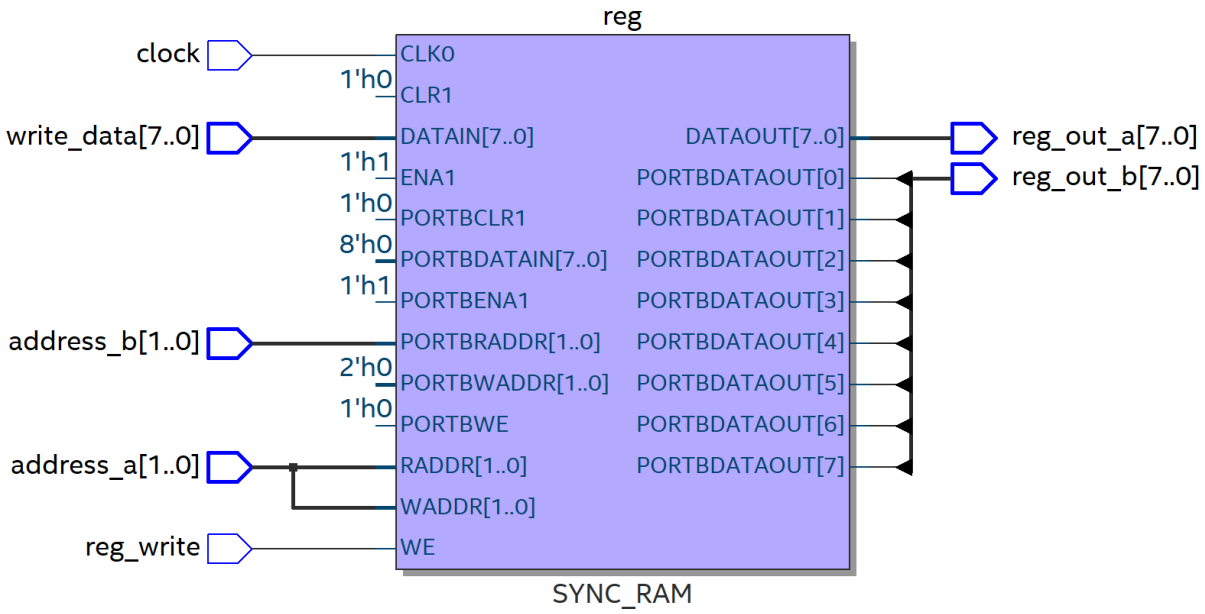
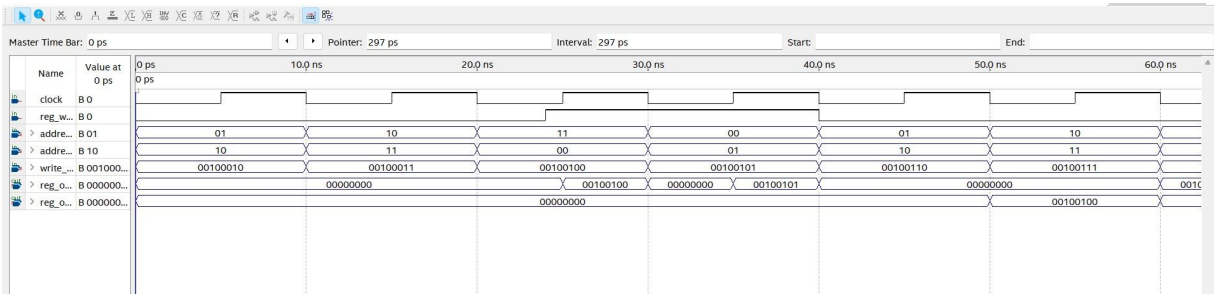


IMAGEM 5- RTL Viewer do BANCO DE REGISTRADORES

Explicando mais detalhadamente o funcionamento, temos, address_a e address_b que são as entradas recebidas, cada uma com 2 bits. O componente ainda possui uma Flag ligada a ele, que é ativada pelo clock. A RegWrite é responsável por indicar o uso do banco de registradores, sendo usada na maioria das operações existentes. Quando possui valor 1 indica que o banco será utilizado para armazenar ou repassar as informações para ULA ou mesmo

realizar ações de Load Word. Por fim, as saídas reg_out_a e reg_out_b, de 8 bits cada, recebem os valores armazenados nos dois registradores de 2 bits e se encaminham a ULA.3



Teste:

Ao banco de registradores aplica um ciclo de **clock** a cada 10 ns (nanosegundos); a flag **reg_write** , que indica o uso do banco de dados, tem clock de 15 ns; **addess_a** recebe 01 e **address_b** recebe 10; **write_data** recebe 00100010, a saída **reg_ou_a**, quando o **reg_write** é ativado, recebe o valor de write_data que estava escrita até aos 10 ns sendo 00100010.

3.3 CLOCK

Este componente gera uma taxa constante usada para quando os eventos ocorrem no hardware. Utilizamos o Simulator Waveform Editor, onde denominamos um valor para executar as operações do processador.

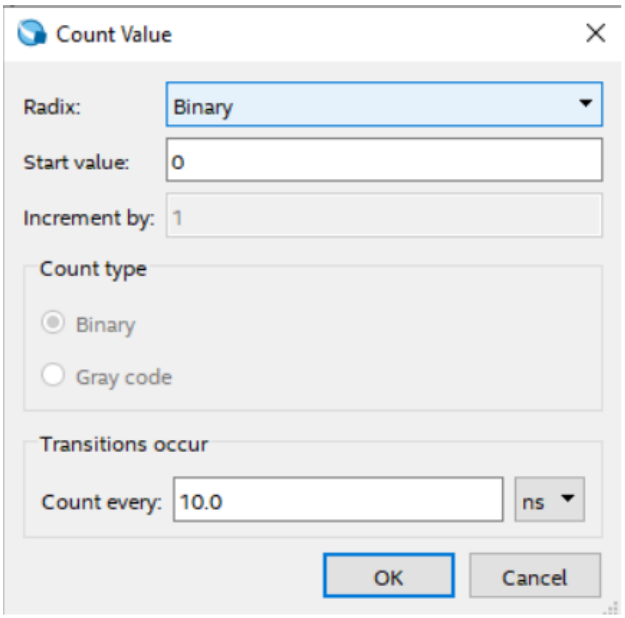
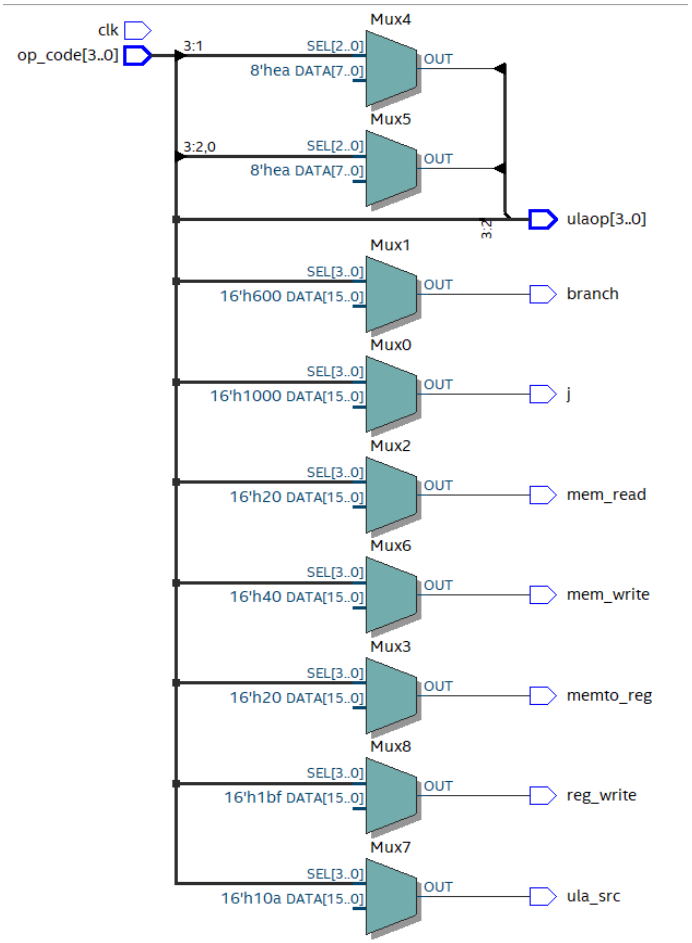


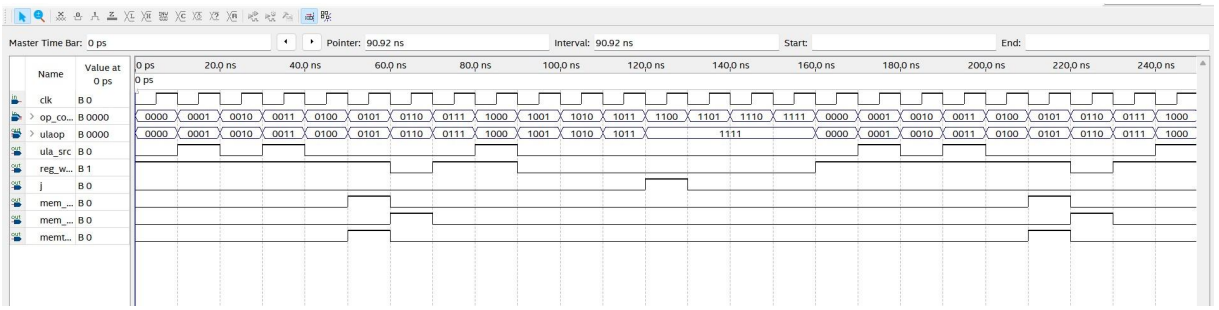
IMAGEM 6- Clock simulado no Quartus

3.4 UNIDADE DE CONTROLE

Para a Unidade de Controle temos a utilização dos 4 bits usados no Opcode, onde a depender de cada valor passado, temos diferentes valores para as flags. São por meio delas que temos as instruções:

- Jump – Usado exclusivamente para operações de Jump
- Branch – Utilizado em estruturas de decisão como beq e bne
- MemRead – Flag usada para leitura de memória. Usada no Load Store
- MentoReg – Usado em operações envolvendo leitura de memória
- AluOP – Opcodes passados na instrução de memória e que definem que instrução está sendo executada
- MemWrite – Flag usada quando se trabalha com a memória RAM. Ativada, pode-se escrever o dado armazenado.
- ALUsrc - Decide se o valor a ser usado provém do extensor de sinal ou do banco de registradores.
- RegWrite - Utilizado para escrita de dados na memória. É nela que o banco de registradores escreve o dado provindo da ULA.





Teste:

A unidade de controle aplica um ciclo de **clock** a cada 10 ns (nanosegundos); a **op_code** aplicamos o valor que 0000 que a cada 10 ns soma-se 1 sendo assim temo uma nova instrução; fazemos o mesmo para o a flag **ulaop**; os resultados das flags **ula_src**, **reg_write**, **j**, **men_read**, **men_write** e **mentoreg** seguem o resultado da **ulaop**.

3.5 MEMÓRIA ROM

A função deste componente funciona com o objetivo de armazenar instruções que venhamos a testar futuramente. Temos um endereço de 8 bits denominado address e uma saída denominada dout também de 8 bits. Dentro dele, temos um array que pega de 0 a 255 posições de 8 bits, possibilitando assim a construção de sequências de instruções bem definidas.

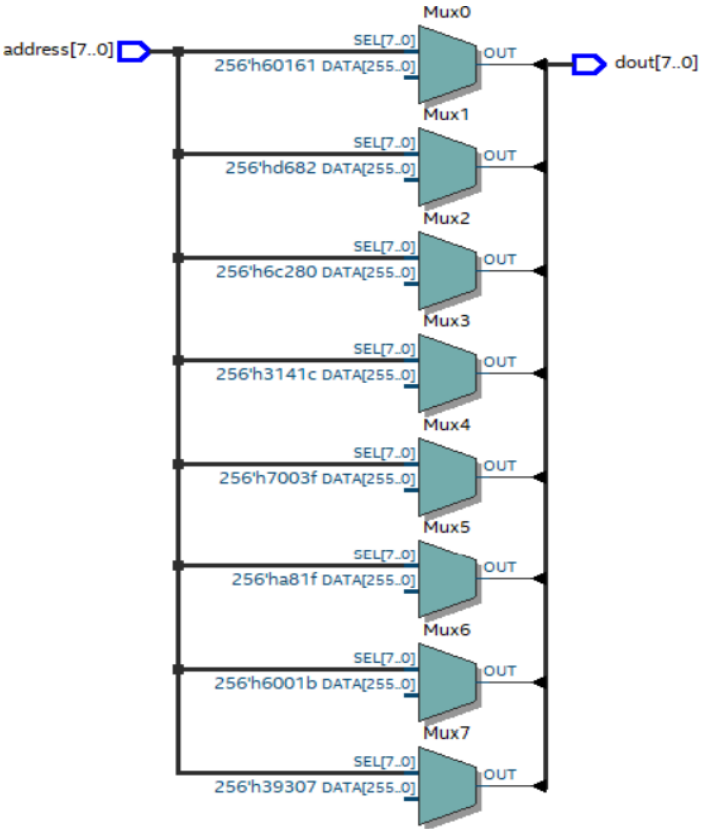
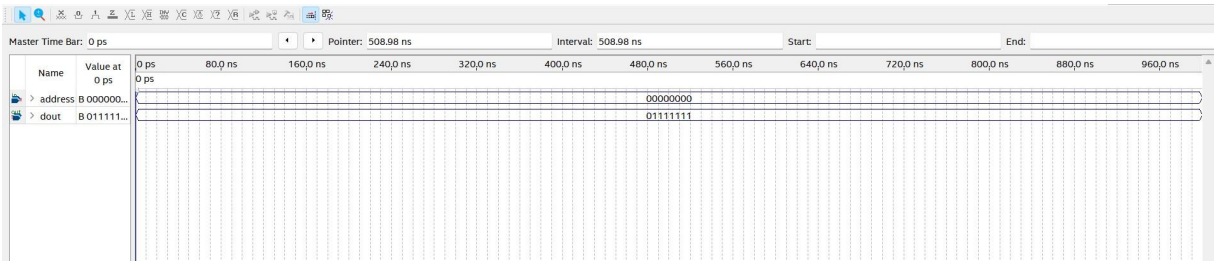


IMAGEM 8- RTL Viewer da Memória ROM

Teste:



Aqui temos duas portas, uma de entrada (**address**) que recebe 00000000 e outra de saída (**dout**) que recebe a saída padrão, neste caso 01111111.

3.6 MEMÓRIA RAM

Após os processamentos realizados na ULA, podemos aplicar os valores diretamente no banco de registradores ou passar para a memória RAM. Quando trabalhamos com as instruções LW (Load Word) e SW (Store Word), fazemos a manipulação de dados entre os registradores e a memória RAM. As flags MEM_WRITE quando ativadas indicam que a instrução está sendo escrita na memória RAM (é o que acontece com a instrução Store Word). Já quando usamos a flag MEM_TO_REG indicamos que desejamos usar tal dado no banco de registradores. Há ainda o uso da flag MEM_READ que também é usada. Nesse último caso, tanto a flag MEM_TO_REG quanto a MEM_READ são usadas na instrução de LW. Em nosso processador temos a seguinte construção do RTL VIEWER.

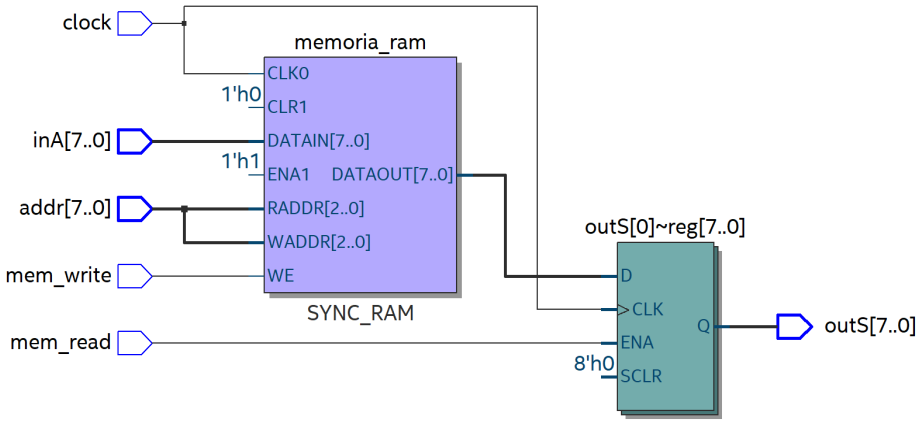
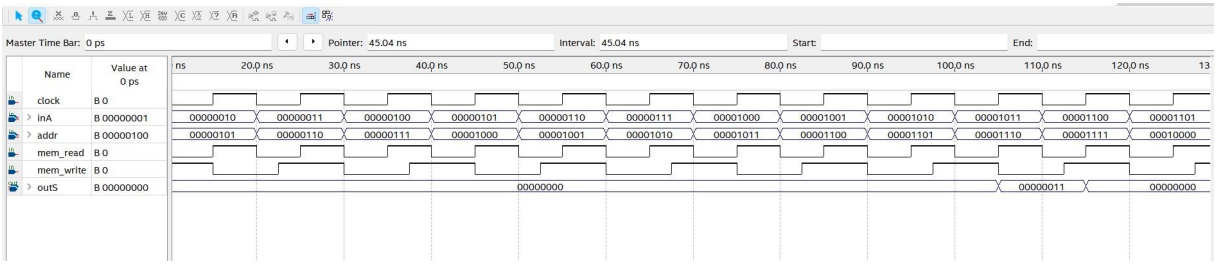


IMAGEM 9- RTL Viewer da Memória RAM

Recebemos 5 variáveis de entrada, onde uma se refere ao Clock, utilizada para manter a consistência de relacionamento entre as execuções, além das duas flags mem_write e

mem_read, que dependendo de estar igual a 1, realizará as instruções de escrita de dados ou a leitura. Se mem_read for igual a 1, a variável gera a saída de 8 bits que fica registrada em outS.

Teste:



A memória RAM possui um ciclo de **clock** a cada 10 ns (nanosegundos); aplicamos a entrada **inA** o valor 00000001 que, a cada 10 nanosegundos, é acrescentado mais 1; a entrada **addr** recebe o valor de 00000100, a cada 10 nanosegundos, é acrescentado mais 1; aplicamos na entrada **mem_read** um clock de 10 ns; aplicamos a entrada **mem_write** um clock de 15 ns para variar tanto a escrita quanto a leitura de dados; a saída **outS** recebe a partir da ativação da flag **mem_write** o valor de **inA** pois este ativa a escrita na memória.

3.7 SOMADOR 8 BITS

Componente responsável por realizar a soma entre dois dados de entrada. Nele, recebemos duas entradas de valores correspondentes à 8 bits que serão somados bit a bit por meio do uso de variáveis auxiliares.

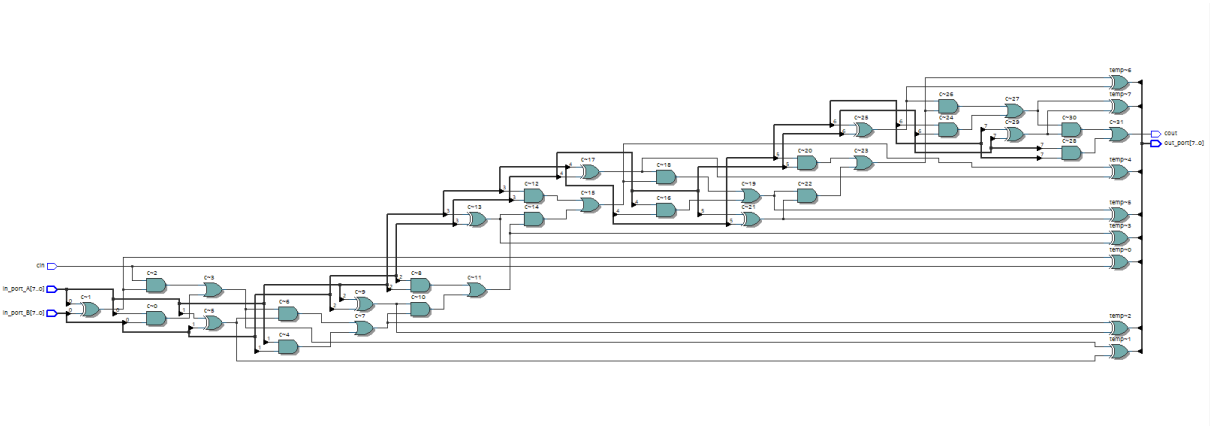
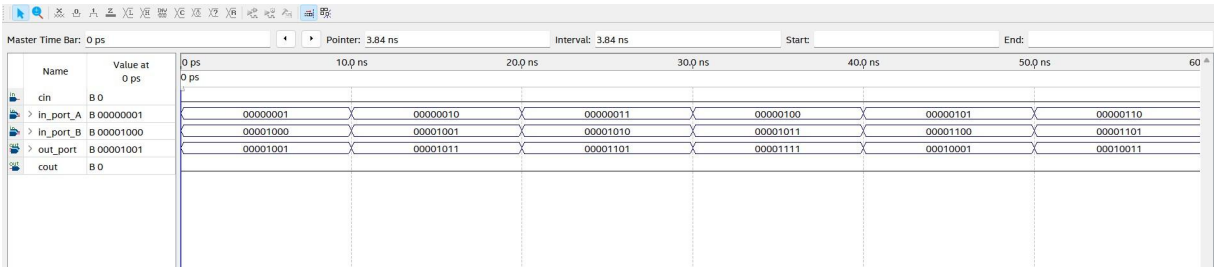


IMAGEM 10- RTL Viewer do Somador 8 BITS

Teste:



O somador possui uma entrada **cin** que está em borda baixa, logo não possui nenhum "vem um"; a entrada **in_port_A** recebe de entrada o valor 00000001 que acrescenta 1 bit a cada 10 ns (nanosegundos); a entrada **in_port_B** recebe da entrada o valor 00001000 que acrescenta 1 bit a cada 10 ns; a saída out port recebe a soma **in_port_B** e **in_port_A.** Como não há estouro de bits a saída **cout** não é ativada continuando em borda baixa.

3.8 AND

Por meio dele podemos realizar a operação lógica necessária para instruções voltadas a desvio condicional, por exemplo. Ele receberá duas entradas de bits (in_port_A e in_port_B) e verifica se as duas condições geral um valor verdadeiro que será posteriormente armazenado em out_port. Caso os dois valores sejam iguais a 1, retorna-se o valor 1. Nos demais casos, retorna 0.

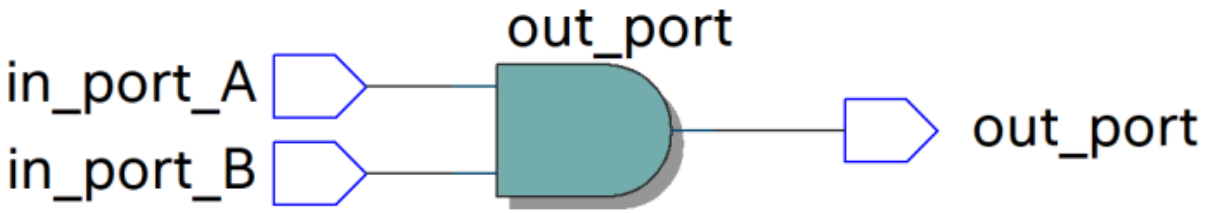
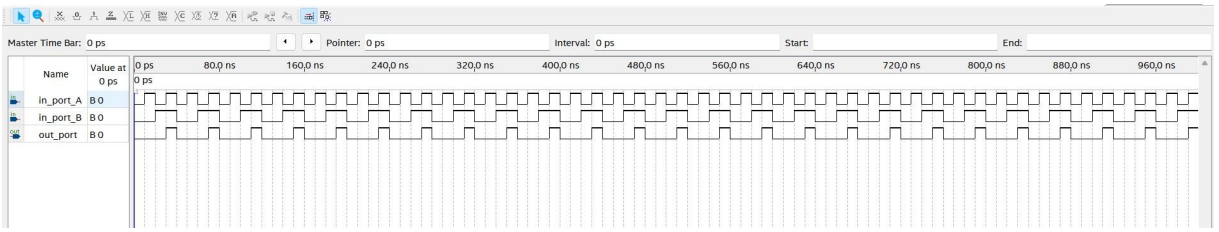


IMAGEM 11- RTL Viewer do AND

Teste:



A porta **and** possui duas entradas a **in_port_A**, com ciclo que clock de 10 ns (nanosegundos), e a **in_port_B** que possui um clock de 20 ns; a saída **out_port** varia a borda

alta do seu clock de acordo com o clock das entradas **in_port_A** e **in_port_B** tiverem também a borda alta de seus respectivos clock como deve ser em uma porta and.

3.9 EXTENSOR DE 2 PARA 8 BITS

O uso do extensor de sinal se dá pela necessidade de realizar operações condicionais, por exemplo. Seu uso é simples: uma variável denominada in_port entra com um valor de 2 bits e agrega-se a quantidade faltante. Após a concatenação de bits faltantes, o valor de saída fica gravado em out_port.

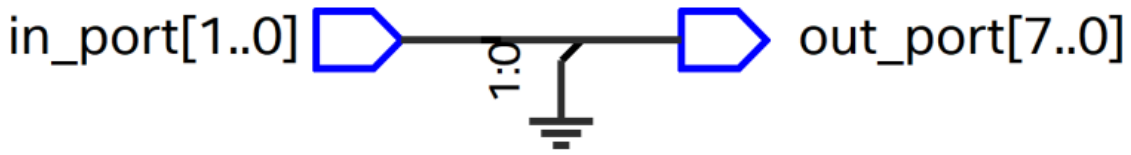
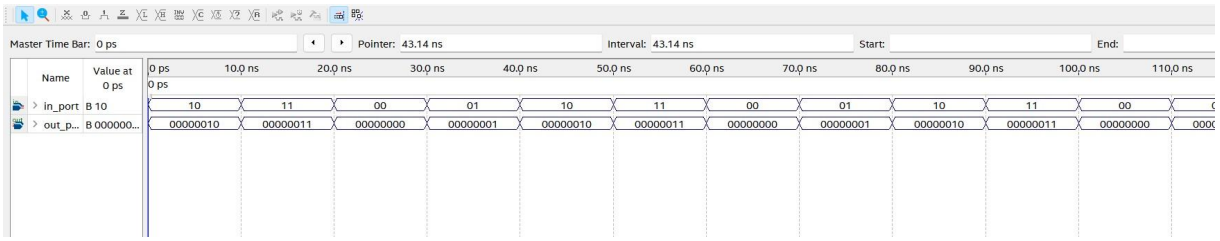


IMAGEM 12- RTL Viewer do Extensor de sinal de 2 para 8 BITS

Teste:



Aqui o componente extensor de sinal 2 para 8 bits possui uma entrada **in_port** que recebe o valor 10 que soma 1 bit a cada 10 ns (nanosegundos). Já a saída **out_port** recebe o valor que corresponde ao acrescimo de 6 0s (zeros) a esquerda do valor de entrada, assim alinhado ele aos 8 bits da saída do componente.

3.10 EXTENSOR DE 4 PARA 8 BITS

Seu funcionamento é semelhante ao extensor de 2 para 8 bits, sendo que a única diferença é que agora temos como valor de entrada um dado de 4 bits.

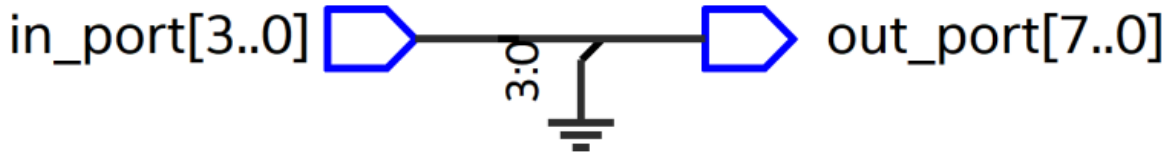
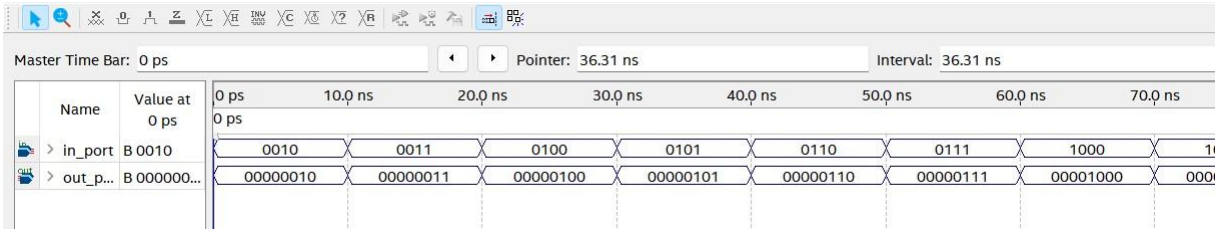


IMAGEM 13- RTL Viewer do Extensor de sinal de 4 para 8 BITS

Teste:



Aqui o componente extensor de sinal 4 para 8 bits possui uma entrada **in_port** que recebe o valor 0010 que soma 1 bit a cada 10 ns (nanosegundos). Já a saída **out_port** recebe o valor que corresponde ao acrescimo de 4 0s (zeros) a esquerda do valor de entrada, assim alinhado ele aos 8 bits da saída do componente.

3.11 SUBTRATOR

Operações de subtração são necessárias para que outras como multiplicação venham a ser geradas. Temos duas variáveis de entrada, denominadas A e B que são dados de entrada de 8 bits, que após serem subtraídas, geram a saída na variável SUB também de 8 bits.

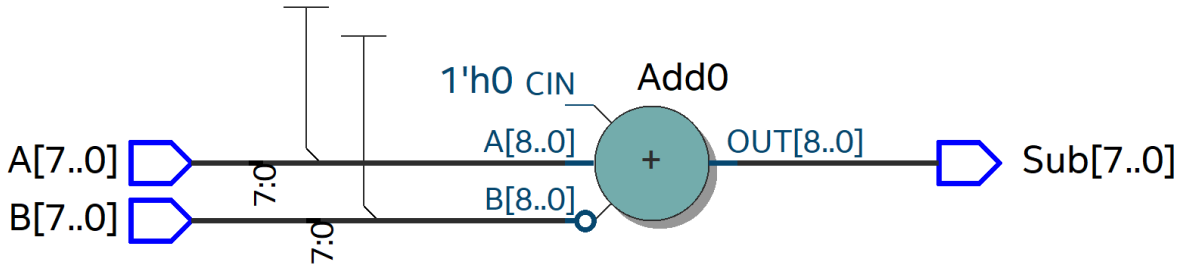
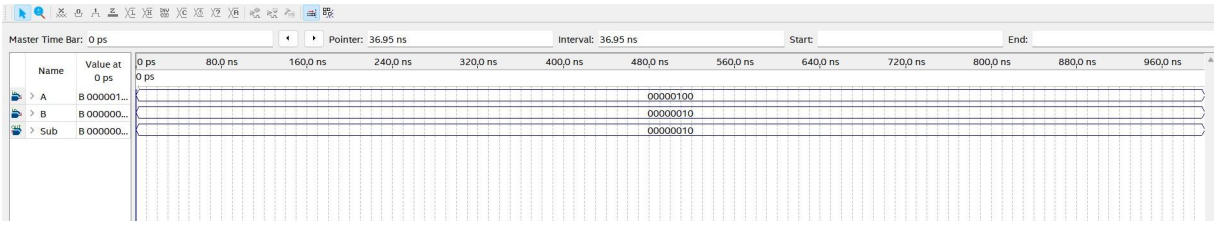


IMAGEM 14- RTL Viewer do Subtrator

Teste:



O componente subtrator possui duas portas de entrada a **A**, que recebe o valor de 00000100, e a **B**, que recebe o valor 00000010. A saída **Sub** o valor 00000010 que corresponde a subtração entra as entradas **A** e **B** respectivamente.

3.12 MULTIPLICADOR

O multiplicador inicia com duas variáveis denominadas in_port_A e in_port_B que entram com os valores de 8 bits. Após uma série de operações de manipulação de bits, uma saída é gerada e gravada na variável out_port que possui o espaço de 16 bits, que é o tamanho necessário para guardar os resultados das operações de multiplicação.

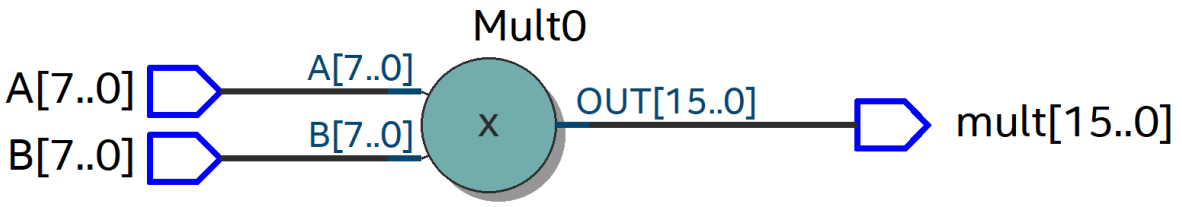


IMAGEM 16- RTL Viewer do Multiplicador

3.13 MULTIPLEXADOR

Tem como objetivo realizar a seleção de trilhas a serem manipuladas. Recebe-se em in_A e in_B, cujas entradas possuem 8 bits, respectivamente e aplica-se o inPort que servirá de seletor de trilhas. O resultado será atribuído a saída em outPort.

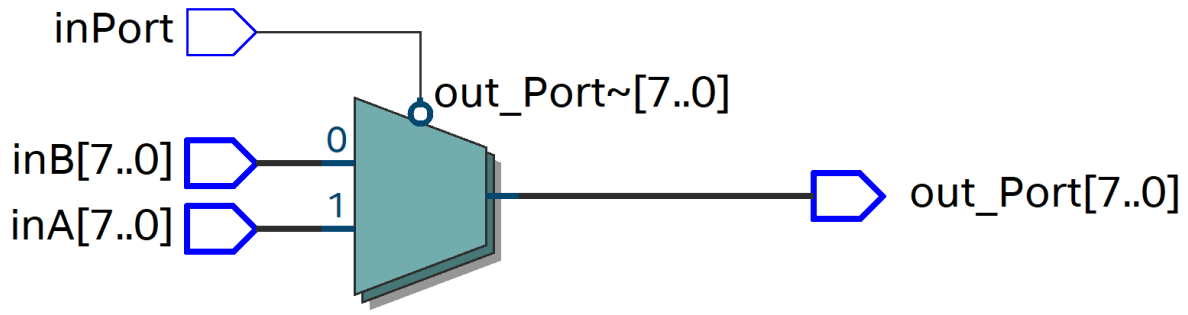
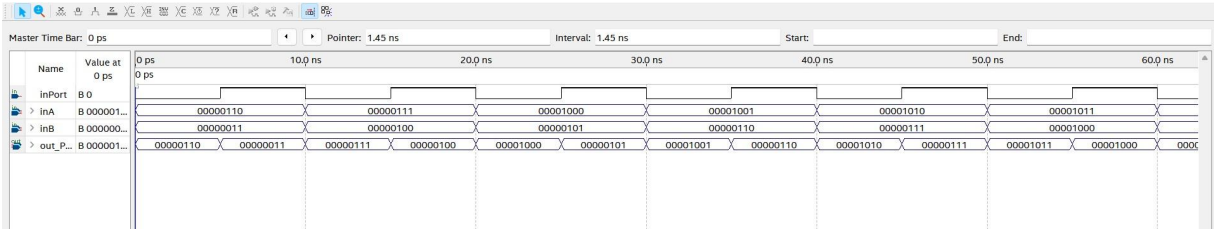


IMAGEM 16- RTL Viewer do Multiplexador

Teste:



O componente multiplexador possui a entrada **inPort,** que recebe um ciclo de clock a cada 10 ns (nanosegundos); a entrada **inA** recebe o valor 00000110, que adiciona 1 bit a cada 10 ns; a entrada **inB** recebe o valo 00000011, que adiciona 1 bit a cada 10 ns; a saída **out_Port** recebe o valor de **inA** quando a entrada **inPort** está em borda baixa e quando este está em borda alta, **out_Port** recebe o valor de **inB**.

3.14 PC

Também conhecido como program counter ou contador de programa, serve como registrador de endereço de instrução. É por meio dele que podemos apanhar o próximo par de instruções de memória. Ele recebe um endereço de 8 bits provindos de inPort além da entrada clock, que irá realizar a ativação do componente inPort gerando a saída out_port que dará continuidade a execução de código com estes bits.

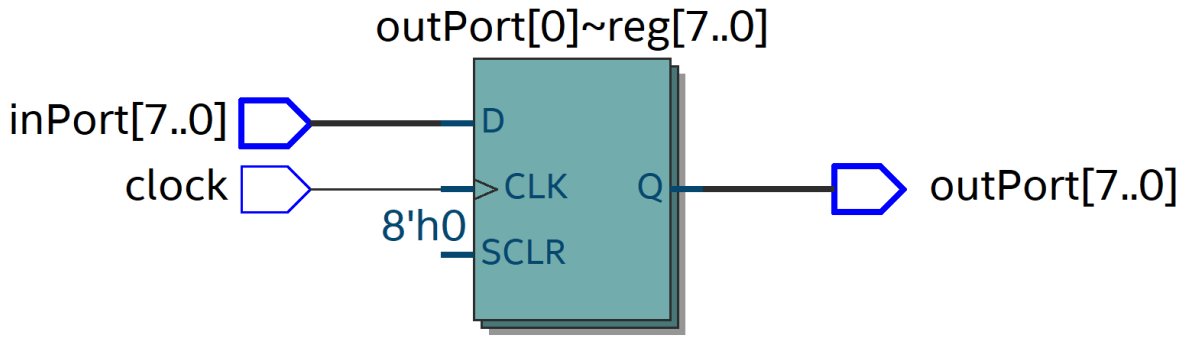
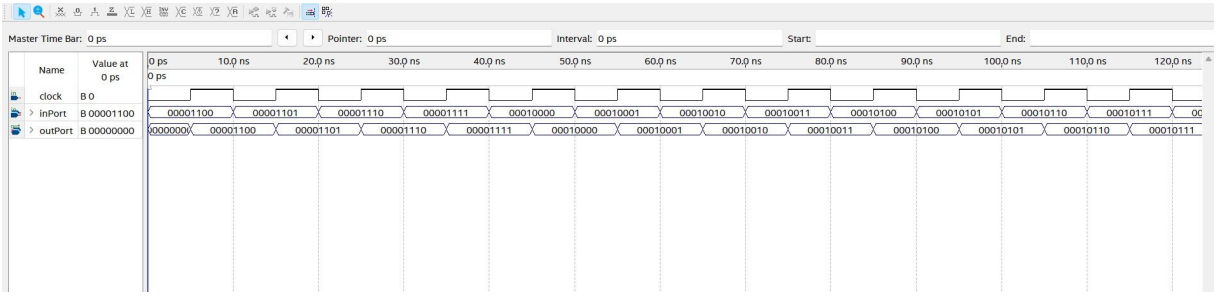


IMAGEM 17- RTL Viewer do PC

Teste:



O componente PC possui uma entrada **clock** que varia seu ciclo de clock a cada 10 ns (nanosegundos); a entrada **inPort** que recebe a entrada 00001100 e que adiciona 1 bit a cada 10 ns; a saída **outPort** que a cada ciclo de clock (borda alta e borda baixa) seleciona o valor da entrada **inPort**.

4. CONCLUSÃO

Este trabalho apresentou a implementação de componentes de 8 bits, um desafio complexo que foi feito com base em muito estudo e pesquisa. Cada componente foi realizado com base em análises, pesquisas e horas de estudo. Com maiores implementações, em um projeto de 16 bits, por exemplo, teríamos muito mais possibilidades, como a implementação das operações de divisão, left shift dentre outros.

Contudo, como um dia já fora no campo da computação, o uso de apenas 8 bits é algo bastante surpreendente, principalmente se levarmos em conta toda a questão de evolução computacional gerada nos últimos 70 anos. A criação desses componentes é um projeto oriundo de pesquisas de grandes conceitos que foram lapidados na ciência da computação pelos primeiros computadores, cuja capacidade de armazenamento não ultrapassava alguns Kilobytes e que hoje conseguem realizar até mesmo processamentos quânticos. Sua implementação buscou ser didática para aqueles que desejam compreender mais acerca do funcionamento de hardware e que ainda pode evoluir muito mais no futuro.