
**PODER EXECUTIVO**
**MINISTÉRIO DA EDUCAÇÃO**
**UNIVERSIDADE FEDERAL DE RORAIMA**
**DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO**

**ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES**

# RELATÓRIO DO PROJETO: PROCESSADOR SONIC – 8 BITS

**ALUNOS:**

**Jorge Siqueira Serrão – 2018022235**
**William Thiago - 2018005029**

**Novembro de 2019**
**Boa Vista/Roraima**


**PODER EXECUTIVO**
**MINISTÉRIO DA EDUCAÇÃO**
**UNIVERSIDADE FEDERAL DE RORAIMA**
**DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO**

**ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES**

# RELATÓRIO DO PROJETO: PROCESSADOR SONIC – 8 BITS

**Outubro de 2019**
**Boa Vista/Roraima**

## Resumo

Este trabalho aborda a elaboração, desenvolvimento e implementação de um processador de 8 bits baseado na arquitetura MIPS usando a linguagem de descrição de hardware VHDL, tendo como base o software FPGA Ciclone V, que foi instalado no Quartus 2, um software disponibilizado pela Intel para implementação de hardwares, nesse projeto não implementamos o processador fisicamente, nos limitando apenas à elaboração do software.

# Conteúdo

# Lista de Figuras

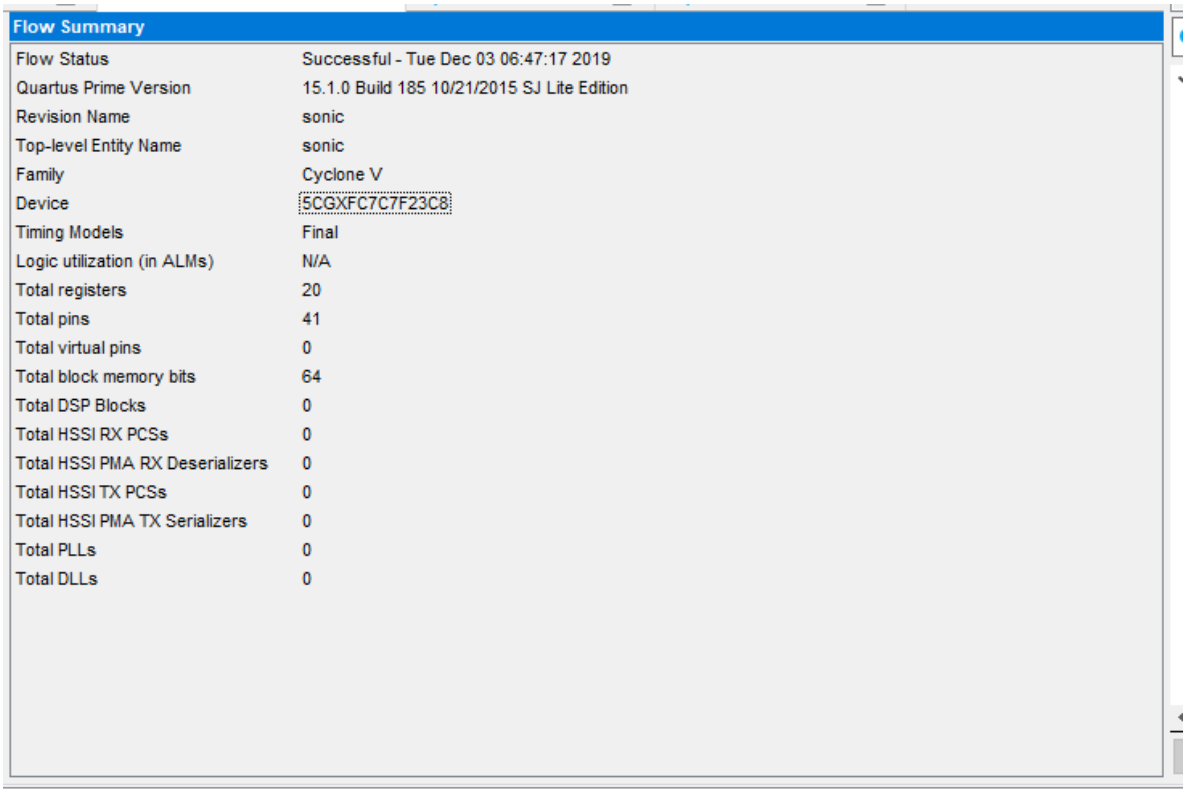
# Lista de Tabelas

# 1 Especificação

Nesta seção é apresentado o conjunto de itens para o desenvolvimento do processador SONIC, bem como a descrição detalhada de cada etapa da construção do processador, desde o datapath até o código, waveform e rtl dos componentes.

## 1.1 Plataforma de desenvolvimento

Para a implementação do processador SONIC de 8 bits foi utilizado a IDE: Quartus 2 v. Prime Version 15.1.0 Build 185 10/21/2015 SJ Lite Edition.



| Flow Summary | |
|---|---|
| Flow Status | Successful - Tue Dec 03 06:47:17 2019 |
| Quartus Prime Version | 15.1.0 Build 185 10/21/2015 SJ Lite Edition |
| Revision Name | sonic |
| Top-level Entity Name | sonic |
| Family | Cyclone V |
| Device | 5CGXFC7C7F23C8 |
| Timing Models | Final |
| Logic utilization (in ALMs) | N/A |
| Total registers | 20 |
| Total pins | 41 |
| Total virtual pins | 0 |
| Total block memory bits | 64 |
| Total DSP Blocks | 0 |
| Total HSSI RX PCSs | 0 |
| Total HSSI PMA RX Deserializers | 0 |
| Total HSSI TX PCSs | 0 |
| Total HSSI PMA TX Serializers | 0 |
| Total PLLs | 0 |
| Total DLLs | 0 |

**Figura 1 - Especificações no Quartus- modelo**

## 1.2 Conjunto de instruções

O processador Sonic possui 8 registradores: $S0,$s1,$s2 e $s3. Assim como 3 formatos de instruções de 8 bits cada, instruções do **tipo R, Instruções do tipo I e Instruções do tipo 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;

**Tipo de Instruções:**

- **Formato do tipo R:** Este formatado aborda instruções de Load, Store e instruções baseadas em operações aritméticas como add e sub.
- **Formato do tipo I**: aborda instruções baseadas em operações com valores imediatos, desvios condicionais ou operações relacionadas à memória: Add Imediato, Sub Imediato, Load e Store são alguns exemplos desse tipo de instrução;
- **Formato do tipo J:** Este formato aborda instruções do tipo jump condicional e incondicional, esta instrução permite saltar entre endereços de memória, muito utilizada em operações que exigem muito registradores.

Formato para escrita de código na linguagem SONIC:

| Operações do tipo R | | |
|---|---|---|
| Op | Reg 1 | Reg 2 |
| 4 bits | 2 bits | 2 bits |
| 7-4 | 3-2 | 1-0 |

| Operações do tipo I | | |
|---|---|---|
| Op | Reg 1 | Reg 2 |
| 4 bits | 2 bits | 2 bits |
| 7-4 | 3-2 | 1-0 |

| Operações do tipo J | |
|---|---|
| Op | Endereço |
| 4 bits | 4 bits |
| 7-4 | 3-0 |

**Tabela 1 - Conjunto de instruções**

**Visão geral das instruções do Processador SONIC:**

O número de bits do campo **Opcode** das instruções é igual a quatro, sendo assim obtemos um total $(Bit(0e1)^{NumeroTodaldeBitsdoOpcode} \therefore 2^X = X)$ de 8 **Opcodes (0-7)** que são distribuídos entre as instruções, assim como é apresentado na Tabela 1, dessa forma podemos ter 8 operações distintas sendo executadas no processador SONIC.

| Opcode | Nome | Formato | Breve Descrição | Exemplo |
|---|---|---|---|---|
| 0000 | add | R | Soma | add $s0, $s1 |
| 0001 | addi | I | Soma Imediata | addi $s0 3 |
| 0010 | sub | R | Subtração | sub Ss0, $s1 |
| 0011 | subi | I | Subtração Imediata | subi $s0 3 |
| 0100 | lw | I | Load Word | lw $s0 memória(00) |
| 0101 | sw | I | Store Word | sw $s0 memória(00) |
| 0110 | move | R | Mover | move $s0 $s1 |
| 0111 | li | I | Load Imediato | li $s0 1 |
| 1000 | beq | J | Desvio Condicional | beq endereço |
| 1001 | bne | J | Desvio Condicional | bne endereço |
| 1010 | If beq e bne | R | Condição para Desvio | if $s0 $s1 |
| 1011 | mul | R | Multiplicação | mul $s0 $s1 |
| 1111 | j | J | Desvio Incondicional | J endereço(0000) |

**Tabela 2 – Tabela que mostra a lista de Opcodes utilizadas pelo processador SONIC.**

## 1.3 Descrição do Hardware

Nesta seção são descritos os componentes do hardware que compõem o processador SONIC, incluindo uma descrição de suas funcionalidades, valores de entrada e saída e waveforms.

### 1.3.1 ALU ou ULA

O componente ula (Q Unidade Lógica Aritmética) tem como principal objetivo efetuar as principais operações aritméticas, dentre elas: soma, subtração, divisão (considerando apenas resultados inteiros) e multiplicação. Adicionalmente o QALU efetua operações de comparação de valor como maior ou igual, menor ou igual, somente maior, menor ou igual, beq, lw, li, jump, move e subi . O componente ula recebe como entrada três valores: **A** – dado de 8bits vindo diretamente do banco de registradores **B** - dado de 8 bits para operação, que pode vir ou do multiplexador ou do banco de registradores dependendo da operação e **OP** – identificador da operação que será realizada de 4bits, que vem diretamente da unidade de controle. A ula também possui três saídas: **zero** – identificador de resultado (2bit) para comparações (1 se verdade e 0 caso contrário); **overflow** – identificador de overflow caso a operação exceda os 8bits, infelizemente nesse projeto de processador, overflow não é tratado devidamente; e **result** – saída com o resultado das operações aritméticas

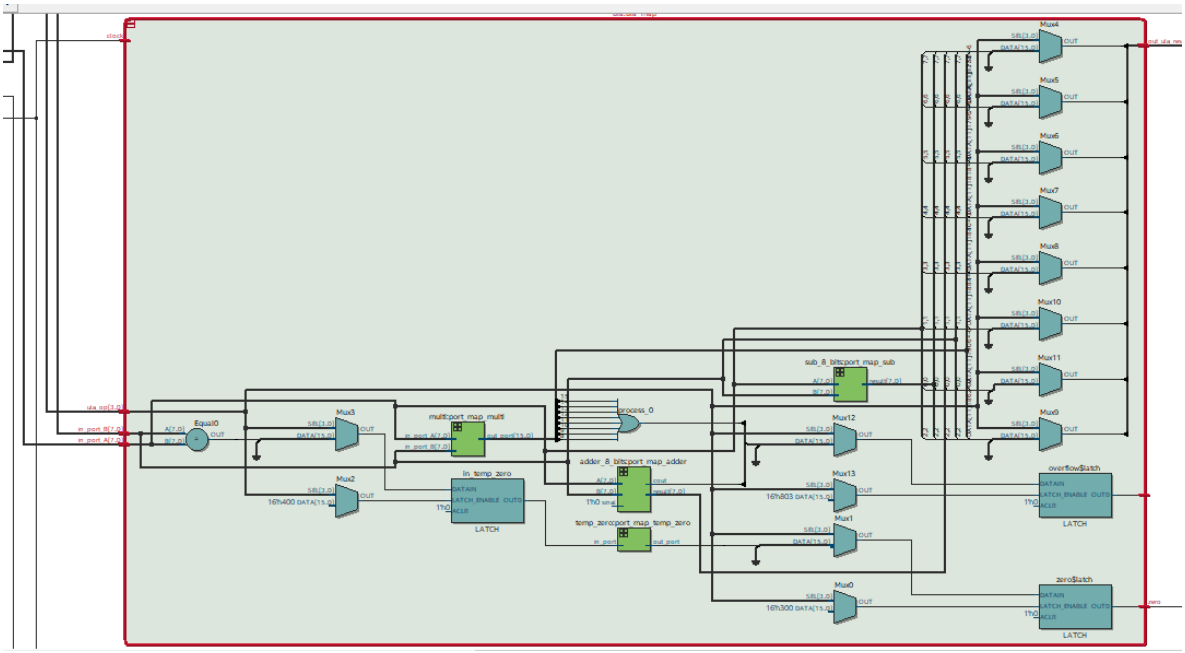


**Figura 2 - Bloco simbólico do componente ALU gerado pelo Quartus**

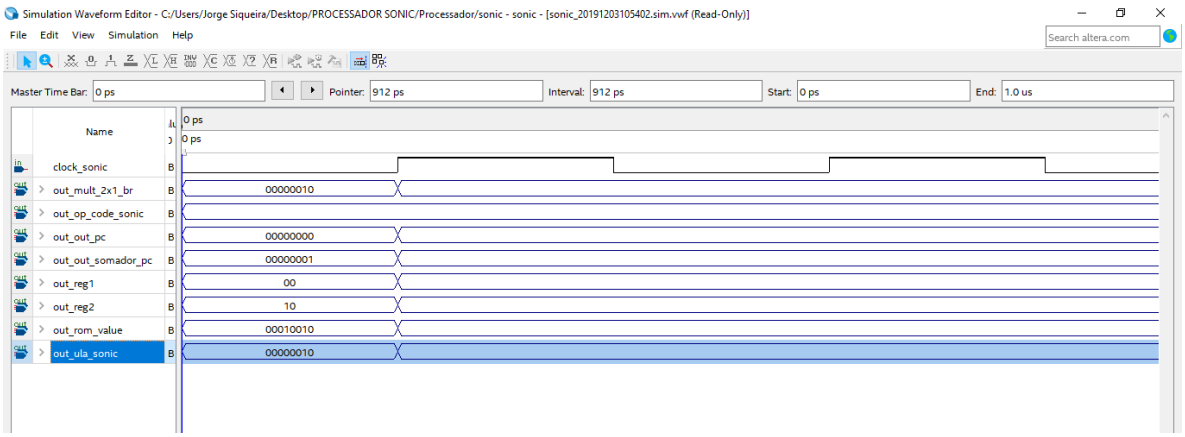



**Figura 3 – Waveform da ula para uma instrução do tipo addi em 1 ns**

### 1.3.2 Banco de registradores

O componente BancoRegistradores tem como papel armazenar e enviar informações a serem utilizadas ao decorrer da execução do programa, cada registrador corresponde a um vetor de 8 posições, que está indexado dentro de um vetor de 4 posições, dessa forma fica mais fácil de acessar e gravar conteúdo nos registradores.

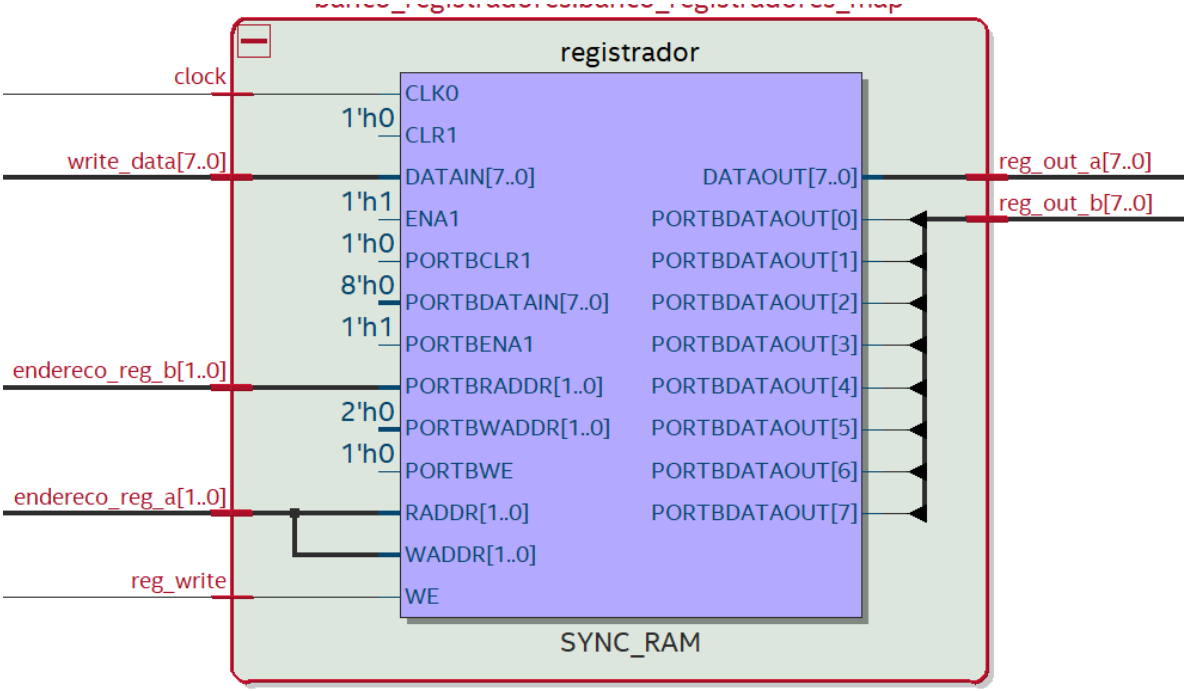



Figura 4 - RTL view do componente Banco de Registradores gerado pelo Quartus.

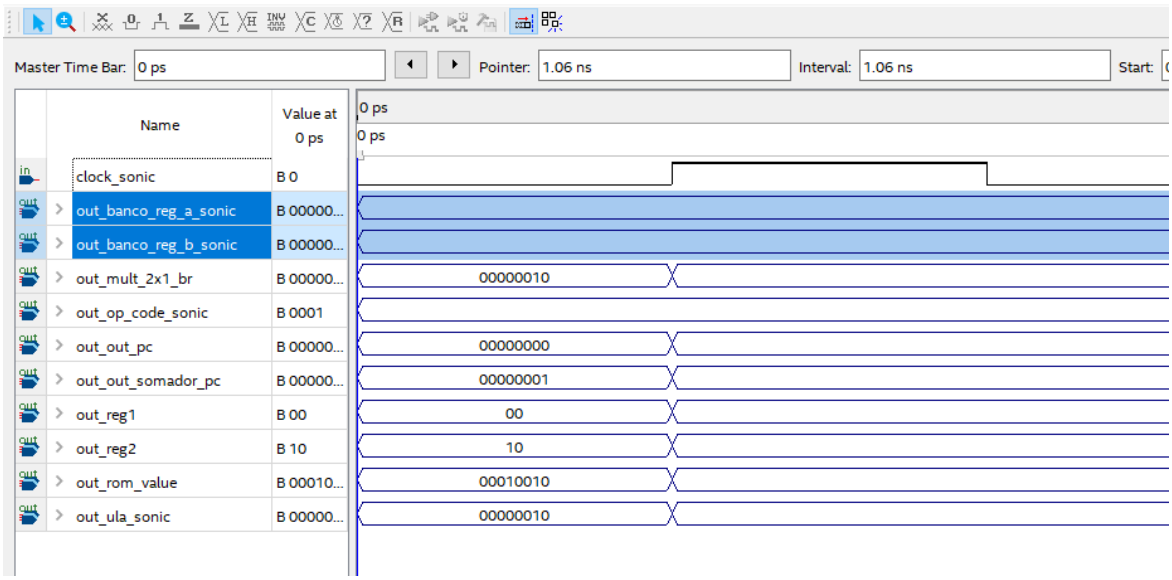



Figura 5 – Wave form do registrador para uma instrução do tipo addi em 1ns

### 1.3.3 Clock

Tem como objetivo evitar que os processador fique limitado à frequência, é o componente do computador que mais influencia no desempenho.

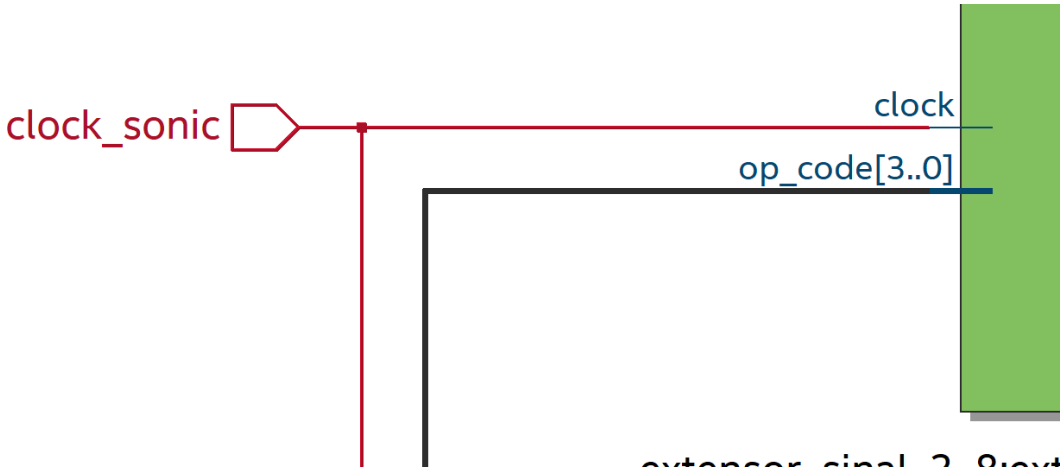



Figura 6 - RTL view do componente Clock gerado pelo Quartus.

### 1.3.4 Controle

O componente unidade de controle tem como objetivo realizar o controle de todos os componentes do processador de acordo com o opcode. Esse controle é feito através das flags de saída abaixo:

- **DvC**: XXXX.
- **en_data**: XXXX.
- **EscMem**: sinal enviado ao componente MemoriaRAM que definirá se os dados serão
- ou não armazenados na memória RAM.
- **MemParaReg**: sinal enviado para um multiplexador para decidir qual dado será
- enviado ao banco de registradores.
- **UlaOp**: XXXX.
- **LwSwOp**: XXXX.
- **EscReg:** sinal que decide se o dado de escrita de volta será armazenado ou não..
- **Wrt_LRT**: XXXX.
- **FlagPC**: XXXX.

Abaixo segue a tabela, onde é feita a associação entre os opcodes e as flags de controle:

**Tabela 3 - Detalhes das flags de controle do processador.**

| Comando | Jump | Branch | M_read | M_to_reg | UlaOp | M_write | ULA_src | Reg_Write |
|---|---|---|---|---|---|---|---|---|
| add | 0 | 0 | 0 | 0 | 0000 | 0 | 0 | 1 |
| addi | 0 | 0 | 0 | 0 | 0001 | 0 | 1 | 1 |
| sub | 0 | 0 | 0 | 0 | 0010 | 0 | 0 | 1 |
| subi | 0 | 0 | 0 | 0 | 0011 | 0 | 1 | 1 |
| lw | 0 | 0 | 1 | 1 | 0100 | 0 | 0 | 1 |
| sw | 0 | 0 | 0 | 0 | 0101 | 1 | 0 | 0 |
| move | 0 | 0 | 0 | 0 | 0110 | 0 | 0 | 1 |
| li | 0 | 0 | 0 | 0 | 0111 | 0 | 1 | 1 |
| beq | 0 | 1 | 0 | 0 | 1000 | 0 | 0 | 0 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **bne** | 0 | 1 | 0 | 0 | 1001 | 0 | 0 | 0 |
| **if beq e bne** | 0 | 0 | 0 | 0 | 1010 | 0 | 0 | 0 |
| **mul** | 0 | 0 | 0 | 0 | 1011 | 0 | 0 | 1 |
| **jump** | 0 | 0 | 0 | 0 | 1111 | 0 | 0 | 0 |

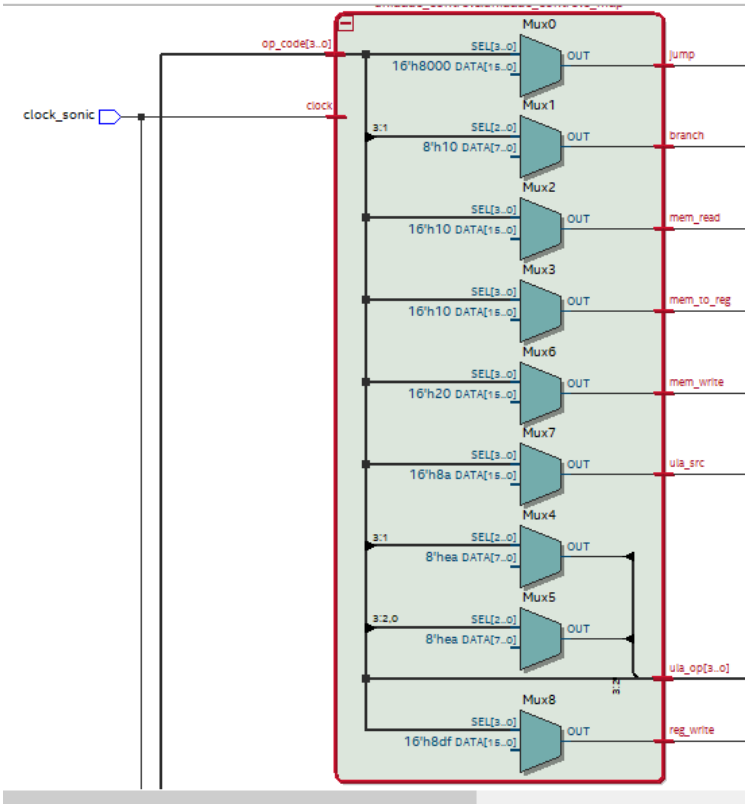



Figura 7 - RTL view do componente unidade de controle gerado pelo Quartus.

### 1.3.5 Memória de dados-RAM

A memória RAM vai permitir que o seu computador tenha acesso imediato aos dados que ele deseja, contribuindo para uma maior rapidez e capacidade de resposta, a memória rom no VHDL corresponde a um vetor de 8 posições de 8 bits cada posição.

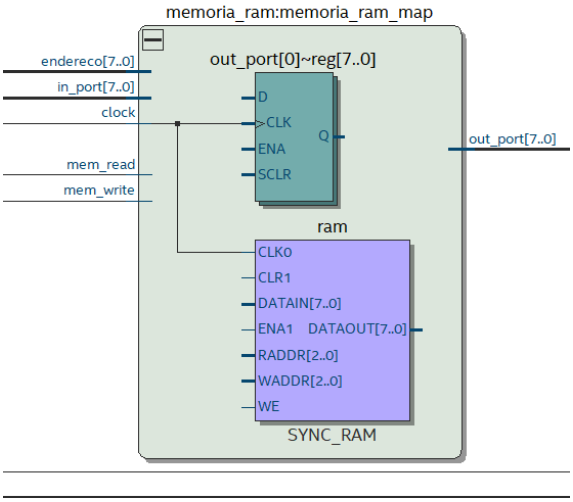



Figura 8 - RTL view do componente Memória de dados gerado pelo Quartus.

### 1.3.6 Memória de Instruções-ROM

É um tipo de memória que permite apenas a leitura, ou seja, as suas informações são gravadas pelo fabricante uma única vez e após isso não podem ser alteradas ou apagadas, somente acessadas. São memórias cujo conteúdo é gravado permanentemente, no quartus a memória ROM corresponde a um vetor de 255 posições com 8 bits cada.

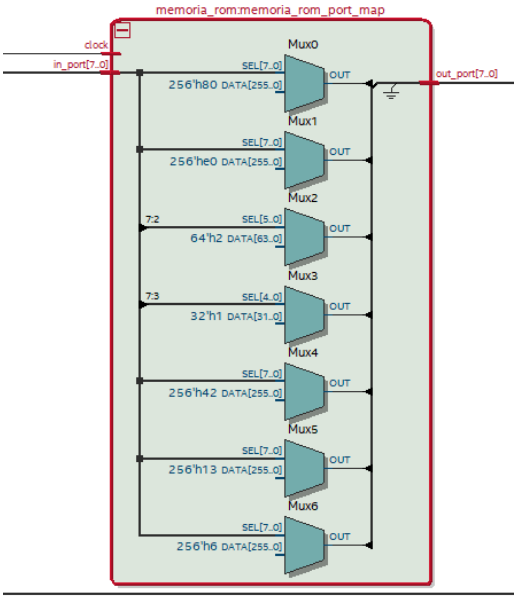



Figura 9 - RTL view do componente Memória de Instrução gerado pelo Quartus.

### 1.3.7 Somador

Somador com entradas e saídas usando a adaptação da representação, no quartus o somador usa diversas entidades e componentes para gerar o resultado esperado.

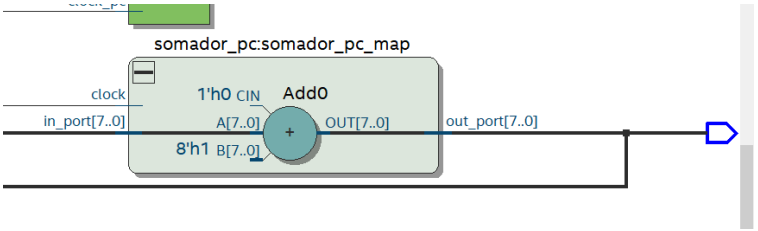



Figura 10 - RTL view do componente somador gerado pelo Quartus.

### 1.3.8 And

Usado para a verificação de ocorrência de branch. Dessa forma, tem o papel de decidir a flag que define se haverá um desvio condicional ou não.

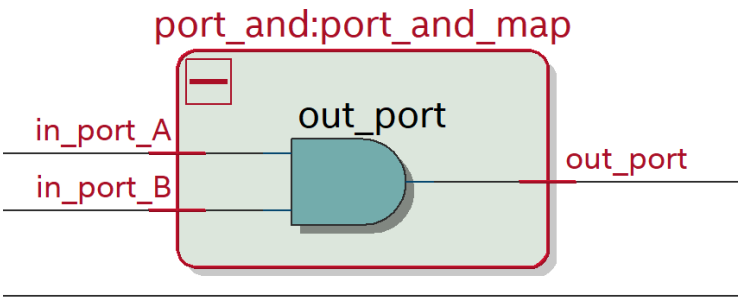



Figura 11 - RTL view do componete AND gerado pelo Quartus.

### 1.3.9 Multiplexador 2x1

Multiplex (MUX). Circuito combinacional composto por Porta Lógicas. Usado para envio de informações de várias fontes através de um único, e daí ele pode decidir que trilha poderá ser usada de acordo com o código de entrada.

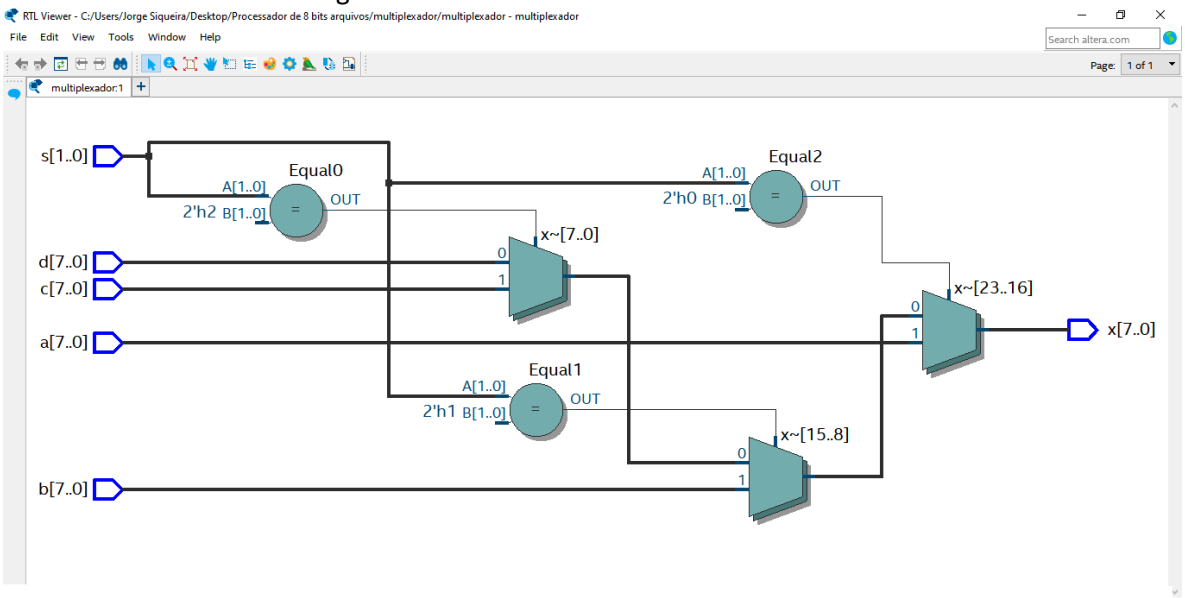



Figura 12 - RTL view do componete Multiplexador 2x1 gerado pelo Quartus.

### 1.3.10 PC

Responsável por controlar outros componentes mandando endereços de instruções, no nosso projeto o PC incrementa de 1 em 1, ou seja, cada instrução de memória de instrução é acessada sequencialmente.

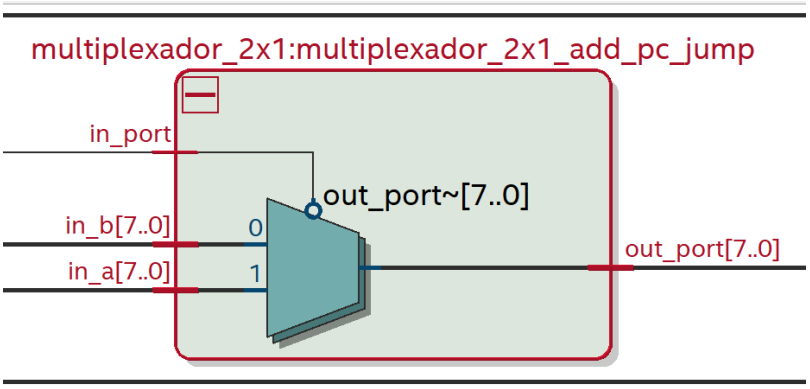



Figura 13 - RTL view do componete Multiplexa pc gerado pelo Quartus.

### 1.3.11 ZERO

Flag representando o resultado de comparações (1 caso for verdade, 0 caso contrário), no nosso processador esse componente foi integrado à ULA, e representa apenas uma saída lógica na mesma.

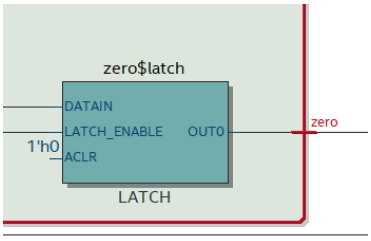



Figura 14 - RTL view do componete zero gerado pelo Quartus.

### 1.3.12 EXTENSORDESINAL_2X8

O extensor de sinal de 2 para 8 bits funciona da seguinte maneira: A entrada **input** com 2 bits será convertida para uma saída **output** com 8 bits.

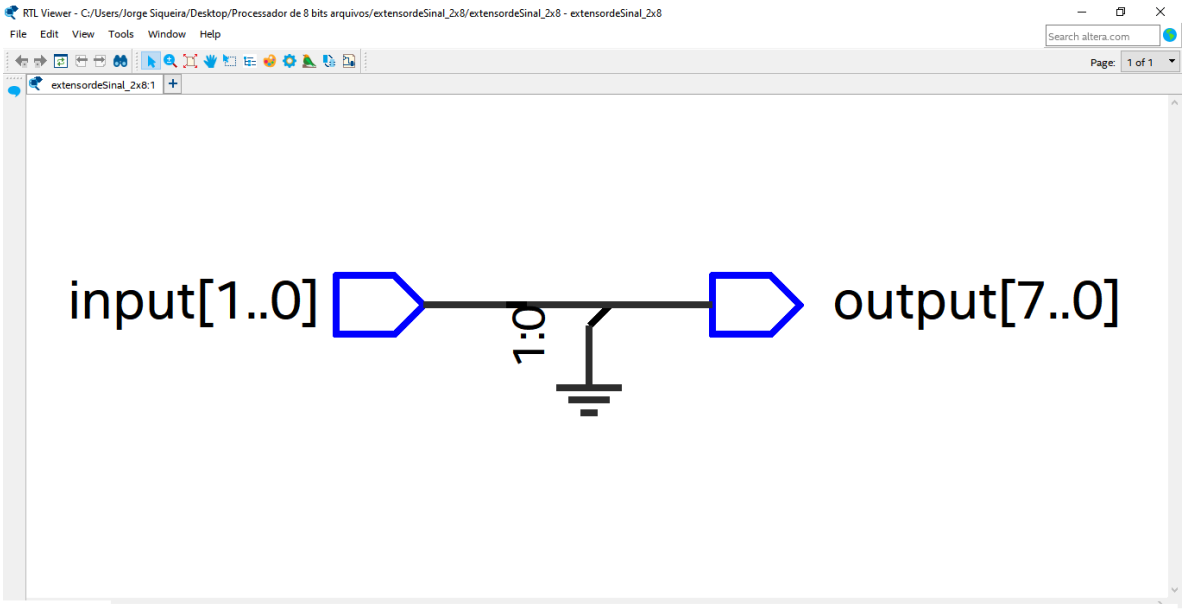



Figura 15 - RTL view do componete extensordesinal_2x8 gerado pelo Quartus.

### 1.3.13 EXTENSOR DE SINAL 4X8

É o componente que recebe como entrada uma trilha d 4 bits e transforma a mesma em uma trilha de 8 bits, é utilizado para realizar a conexão entre o pc_add (que incrementa a instrução do pc) com o mux, que verifica se deve seguir para a próxima instrução.

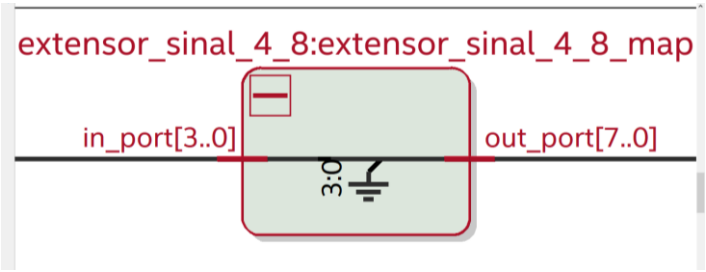



Figura 16 - RTL view do componente extensor_4x8 gerado pelo Quartus.

## 1.4 Datapath

É a conexão entre as unidades funcionais formando um único caminho de dados e acrescentando uma unidade de controle responsável pelo gerenciamento das ações que serão realizadas para diferentes classes de instruções. Para o processador SONIC foi decido colocar a memória de dados e a memória de instruções, pois ambas possibilitam um gerenciamento melhor dos testes facilitando no momento da execução dos algoritmos que foram codificados.

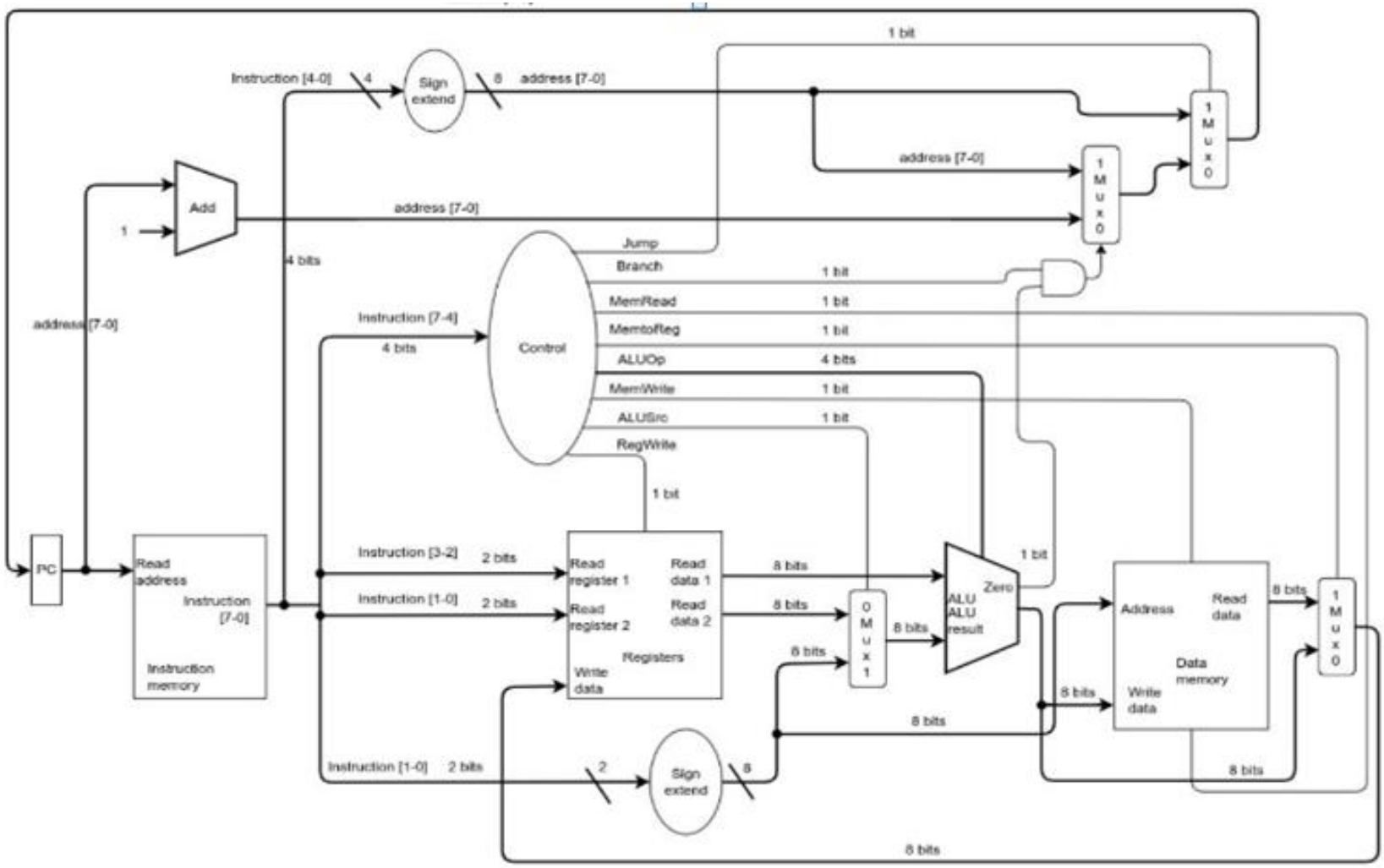



*Figura 17 – Datapath utilizado para montar o VHDL e os componentes n o Quartus*

# 2 Considerações finais

Este trabalho apresentou o projeto e implementação do processador de 8 bits denominado de SONIC que é a junção dos conhecimentos adquiridos durante o período de ensino da disciplina Arquitetura e Organização de Computadores. O nome escolhido faz referência simplesmente ao tempo corrido.