Processador BIP 2

Alunos: Lucas Diniz Santos Henrique, Ronald Soares Lopes, Sara da Cunha Monteiro de Souza

Engenharia da Computação,
Disciplina de Sistemas Embarcados
Profº David Vasconcelos

SUMÁRIO

INSTITUTO FEDERAL
Fluminense

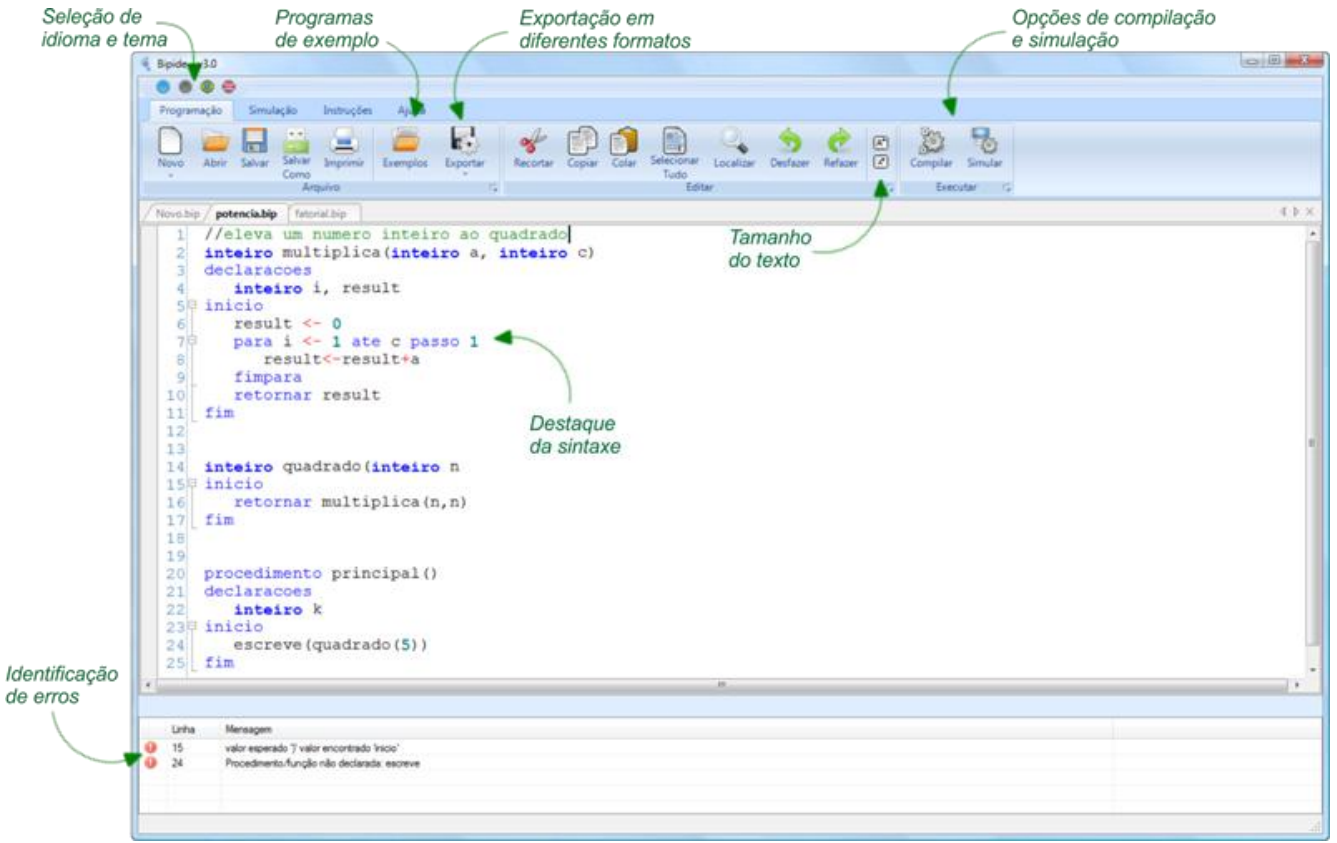
PROCESSADORES BIP

- Desenvolvida por pesquisadores do Grupo de Sistemas Embarcados e Distribuídos da Universidade do Vale do Itajaí (UNIVALI);
- Tem por objetivo contribuir para o ensino de conceitos sobre programação e funcionamento de sistemas computacionais;
- Utiliza-se de um modelo simplificado de processador especificado segundo uma abordagem multidisciplinar e focado na facilitação do aprendizado.

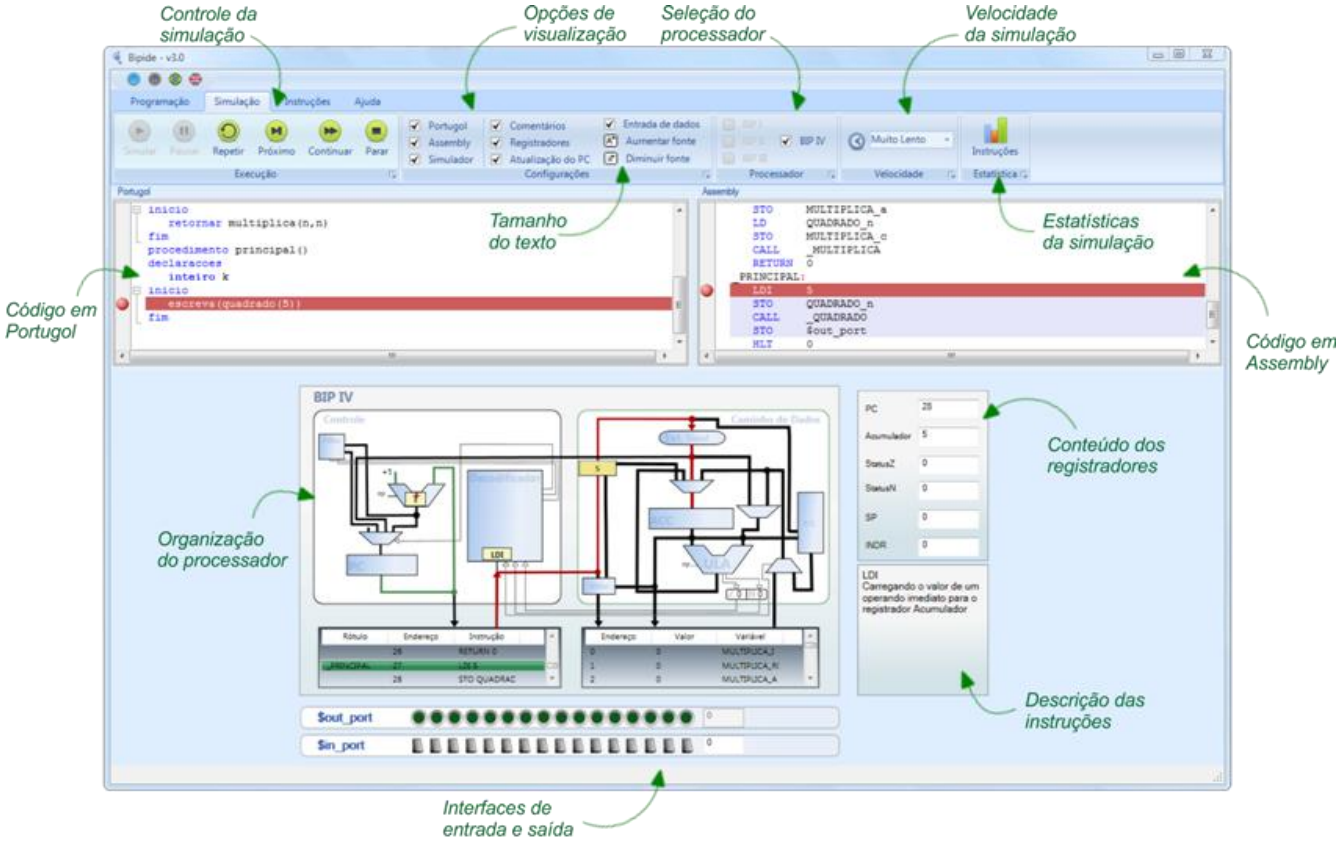
BIPIDE 

- IDE para a família de processadores BIP;
- Possibilita a criação de algoritmos em Portugol e sua execução passo a passo ou de forma contínua;
- Apresenta de forma visual o código correspondente ao programa em linguagem assembly e o estado dos componentes da organização do processador através de animações que ilustram o funcionamento interno do mesmo.

BIPIDE


Seleção de idioma e tema
Programas de exemplo
Exportação em diferentes formatos
Opções de compilação e simulação
Tamanho do texto
Destaque da sintaxe
Identificação de erros
Programação
Simulação
Instruções
Novo
Abrir
Salvar
Salvar Como
Imprimir
Exemplos
Exportar
Recortar
Copiar
Colar
Selecionar Tudo
Localizar
Desfazer
Refazer
Compilar
Simular
Arquivo
Editar
Executar
Novo.bip
potencia.bip
fatorial.bip
1 //eleva um numero inteiro ao quadrado
2 inteiro multiplica(inteiro a, inteiro c)
3 declaracoes
4 inteiro i, result
5 inicio
6 result <- 0
7 para i <- 1 ate c passo 1
8 result<-result+a
9 fimpara
10 retornar result
11 fim
12
13
14 inteiro quadrado(inteiro n
15 inicio
16 retornar multiplica(n,n)
17 fim
18
19
20 procedimento principal()
21 declaracoes
22 inteiro k
23 inicio
24 escreve(quadrado(5))
25 fim
Linha
Mensagem
15
valor esperado ')' valor encontrado 'inicio'
24
Procedimento,função não declarada: escreve

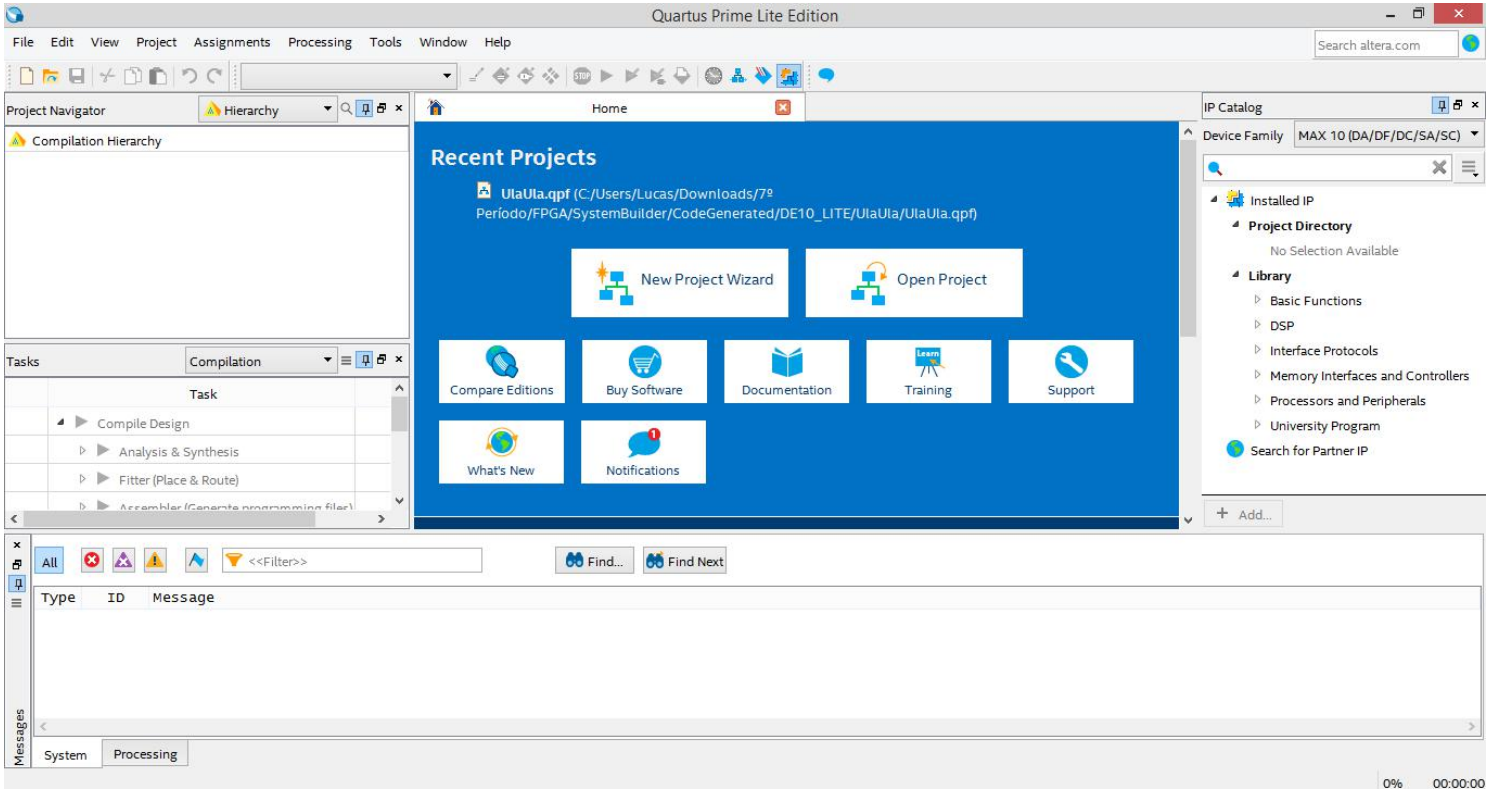
BIPIDE


Controle da simulação
Opções de visualização
Seleção do processador
Velocidade da simulação
Tamanho do texto
Estatísticas da simulação
Código em Portugol
Código em Assembly
Conteúdo dos registradores
Organização do processador
Descrição das instruções
Interfaces de entrada e saída
BIP IV
$out_port
$in_port

QUARTUS

- Software de design de dispositivos lógicos programáveis produzido pela Altera;
- Permite a análise e a síntese de projetos de **linguagens de descrição de hardware** (HDL);
- O usuário pode compilar seus projetos, executar análises de tempo, examinar diagramas RTL, simular a reação de um projeto a diferentes estímulos e configurar o dispositivo de destino com o programador.

QUARTUS

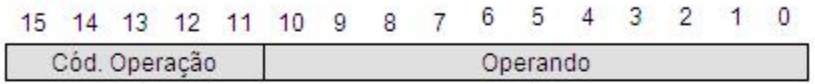

Quartus Prime Lite Edition
File Edit View Project Assignments Processing Tools Window Help
Search altera.com
Project Navigator
Hierarchy
Compilation Hierarchy
Home
Recent Projects
UlaUla.qpf (C:/Users/Lucas/Downloads/7º Período/FPGA/SystemBuilder/CodeGenerated/DE10_LITE/UlaUla/UlaUla.qpf)
New Project Wizard
Open Project
Compare Editions
Buy Software
Documentation
Training
Support
What's New
Notifications
IP Catalog
Device Family
MAX 10 (DA/DF/DC/SA/SC)
Installed IP
Project Directory
No Selection Available
Library
Basic Functions
DSP
Interface Protocols
Memory Interfaces and Controllers
Processors and Peripherals
University Program
Search for Partner IP
Add...
Tasks
Compilation
Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
All
<<Filter>>
Find...
Find Next
Type ID Message
Messages
System
Processing
0%
00:00:00

PROCESSADOR BIP II

O BIP II utiliza uma arquitetura baseada na arquitetura RISC do microcontrolador PIC (Programmable Intelligent Computer). Esta versão do BIP possui de três a quatro registradores:

- PC (Program Counter) - O contador de programa aponta para o endereço da próxima instrução a ser executada
- IR (IR – Instruction Register) - O registrador de instrução armazena a instrução que está em execução
- ACC (Accumulator) - O acumulador é utilizado para armazenamento de dados durante uma operação.
- STATUS - Registrador de estado com dois flags, que são utilizados em instruções de desvio.

PROCESSADOR BIP II

Tamanho da palavra de dados	16 bits
Tipo de dados	Inteiro de 16 bits com sinal: –32768 a +32767
Tamanho da palavra de instrução	16 bits
Formato de instrução	
Modos de endereçamento	Direto: o operando é um endereço da memória Imediato: o operando é uma constante
Registradores	ACC: acumulador IR: registrador de instrução PC: contador de programa STATUS: registrador de estado com dois *flags* (Z e N)
Classes de instrução	Transferência (acesso à memória): STO, LD, LDI Aritmética: ADD, ADDI, SUB e SUBI Controle: HLT Desvio: BEQ, BNE, BGT, BGE, BLT, BLE e JMP

Fonte: Adaptado de Morandi, Raabe e Zeferino (2006).

PROCESSADOR BIP II

Código da operação	Instrução	Operação e atualização do PC	
00000	HLT	Paralisa a execução	PC ← PC
00001	STO operand	Memory[operand]← ACC	PC ← PC + 1
00010	LD operand	ACC ← Memory[operand]	PC ← PC + 1
00011	LDI operand	ACC ← operand	PC ← PC + 1
00100	ADD operand	ACC ← ACC + Memory[operand]	PC ← PC + 1
00101	ADDI operand	ACC ← ACC + operand	PC ← PC + 1
00110	SUB operand	ACC ← ACC - Memory[operand]	PC ← PC + 1
00111	SUBI operand	ACC ← ACC - operand	PC ← PC + 1
01000	BEQ operand		Se (STATUS.Z=1) então PC ← endereço Senão PC ← PC + 1
01001	BNE operand		Se (STATUS.Z=0) então PC ← endereço Senão PC ← PC + 1
01010	BGT operand		Se (STATUS.Z=0) e (STATUS.N=0) então PC ← endereço Senão PC ← PC + 1

PROCESSADOR BIP II

01011	BGE operand		Se (STATUS.N=0) então PC ← endereço Senão PC ← PC + 1
01100	BLT operand		Se (STATUS.N=1) então PC ← endereço Senão PC ← PC + 1
01101	BLE operand		Se (STATUS.Z=1) ou (STATUS.N=1) então PC ← endereço Senão PC ← PC + 1
01110	JMP operand		PC ← endereço
01111-11111	Reservado para as futuras gerações		

Fonte: Adaptado de Zeferino (2007).

PROCESSADOR BIP II

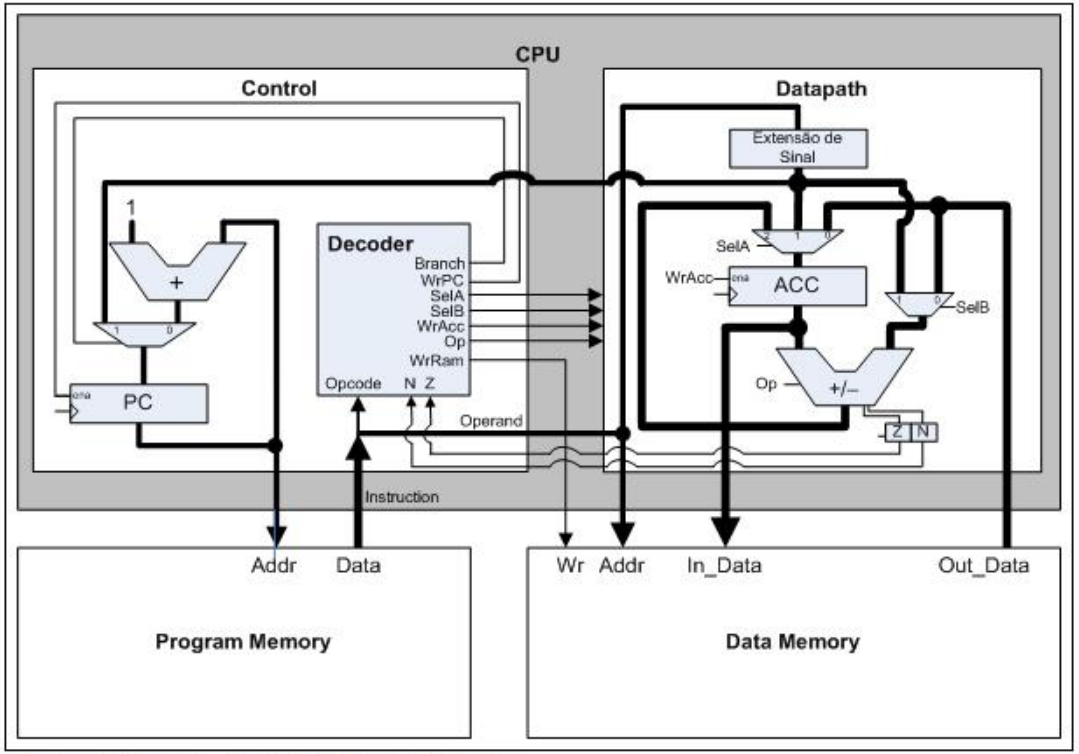
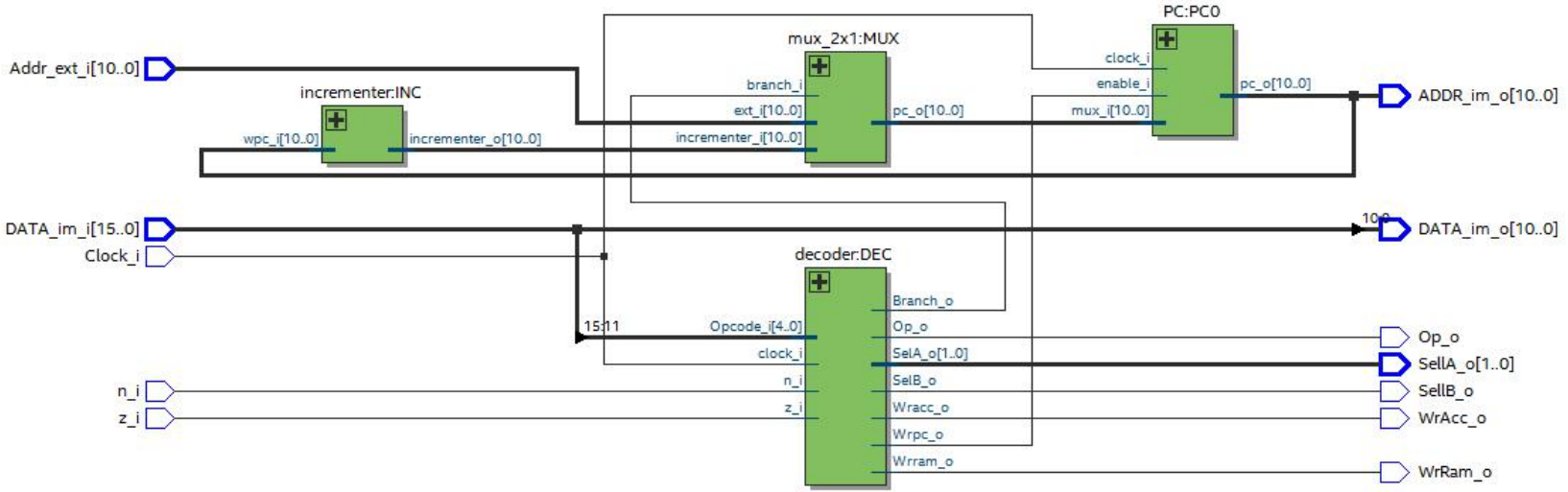


Figura 4. Organização do Processador BIP II

Fonte: Pereira (2008)

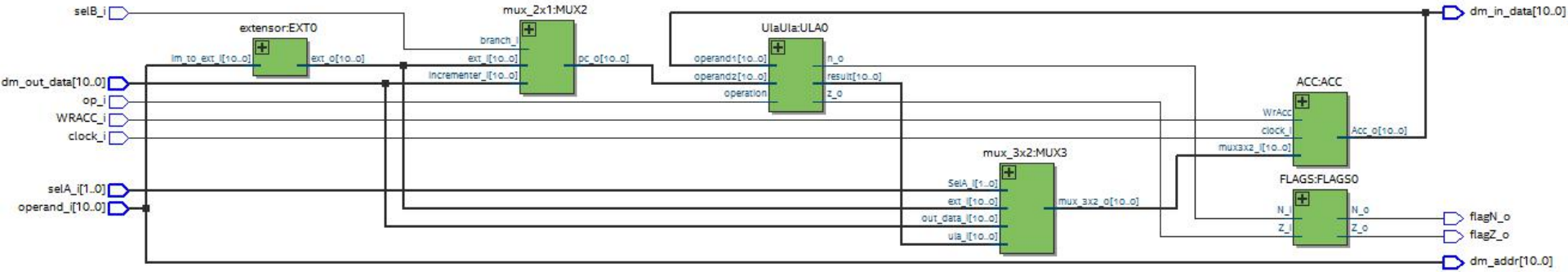
PROCESSADOR BIP II

➔ Control



PROCESSADOR BIP II

➔ Datapath



PROCESSADOR BIP II

➔ Integração de módulos: processor_bip2.v

```
module processor_bip2
#(
    parameter MSB_DATA = 16,
    parameter MSB_ROM = 11,
    parameter MSB_OPERAND = 11,
    parameter MSB_SELA = 2,
    parameter LSB = 0
)
(
// INTERFACE SISTEMA
    CLOCK_i,    //Clock do sistema
// INTERFACE MEMÓRIA DE INSTRUÇÃO (Memória ROM)
    ADDR_im_o, //Endereço
    DATA_im_i, //Valor
//INTERFACE MEMÓRIA DE DADOS (Memória RAM)
    WRRAM_o,
    ADDR_dm_o,
    IN_DATA_o,
    OUT_DATA_i
);
```

```
input wire CLOCK_i;
input wire [(MSB_DATA-1):LSB] DATA_im_i;
input wire [(MSB_OPERAND-1):LSB] OUT_DATA_i;

output wire [(MSB_ROM-1):LSB]ADDR_im_o;
output wire WRRAM_o;
output wire [(MSB_ROM-1):LSB] ADDR_dm_o;
output wire [(MSB_OPERAND-1):LSB]IN_DATA_o;

wire [(MSB_OPERAND-1):LSB] wExt_Mux2x1;
wire wN;
wire wZ;
wire wOp;
wire [(MSB_SELA - 1):LSB] wSelA;
wire wSelB;
wire wWrAcc;
wire [(MSB_OPERAND-1):LSB] wOperand;
```

PROCESSADOR BIP II

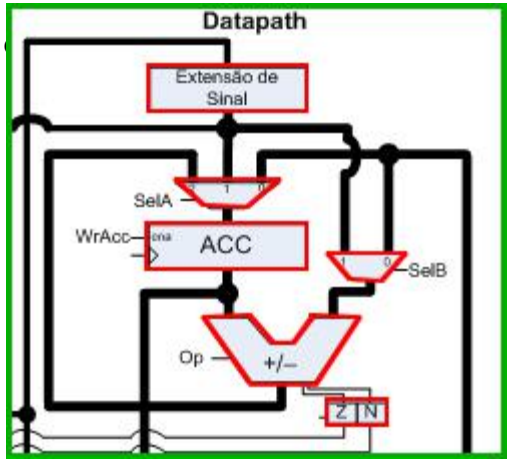
➔ Integração de módulos: processor_bip2.v

```
controle
Controle0
(
    .Addr_ext_i(wExt_Mux2x1),
    .z_i(wZ),
    .n_i(wN),
    .SelIA_o(wSelA),
    .SelIB_o(wSelB),
    .WrAcc_o(wWrAcc),
    .Op_o(wOp),
    .WrRam_o(WRRAM_o),
    .Clock_i(CLOCK_i),
    .DATA_im_i(DATA_im_i),
    .DATA_im_o(wOperand),
    .ADDR_im_o(ADDR_im_o)
);
```

```
datapath
Datapath0
(
    .clock_i(CLOCK_i),
    .selA_i(wSelA),
    .selB_i(wSelB),
    .WRACC_i(wWrAcc),
    .operand_i(wOperand),
    .op_i(wOp),
    .dm_out_data(OUT_DATA_i),
    .dm_in_data(IN_DATA_o),
    .dm_addr(ADDR_dm_o),
    .flagZ_o(wZ),
    .flagN_o(wN),
    .ext_o(wExt_Mux2x1)
);

endmodule
```

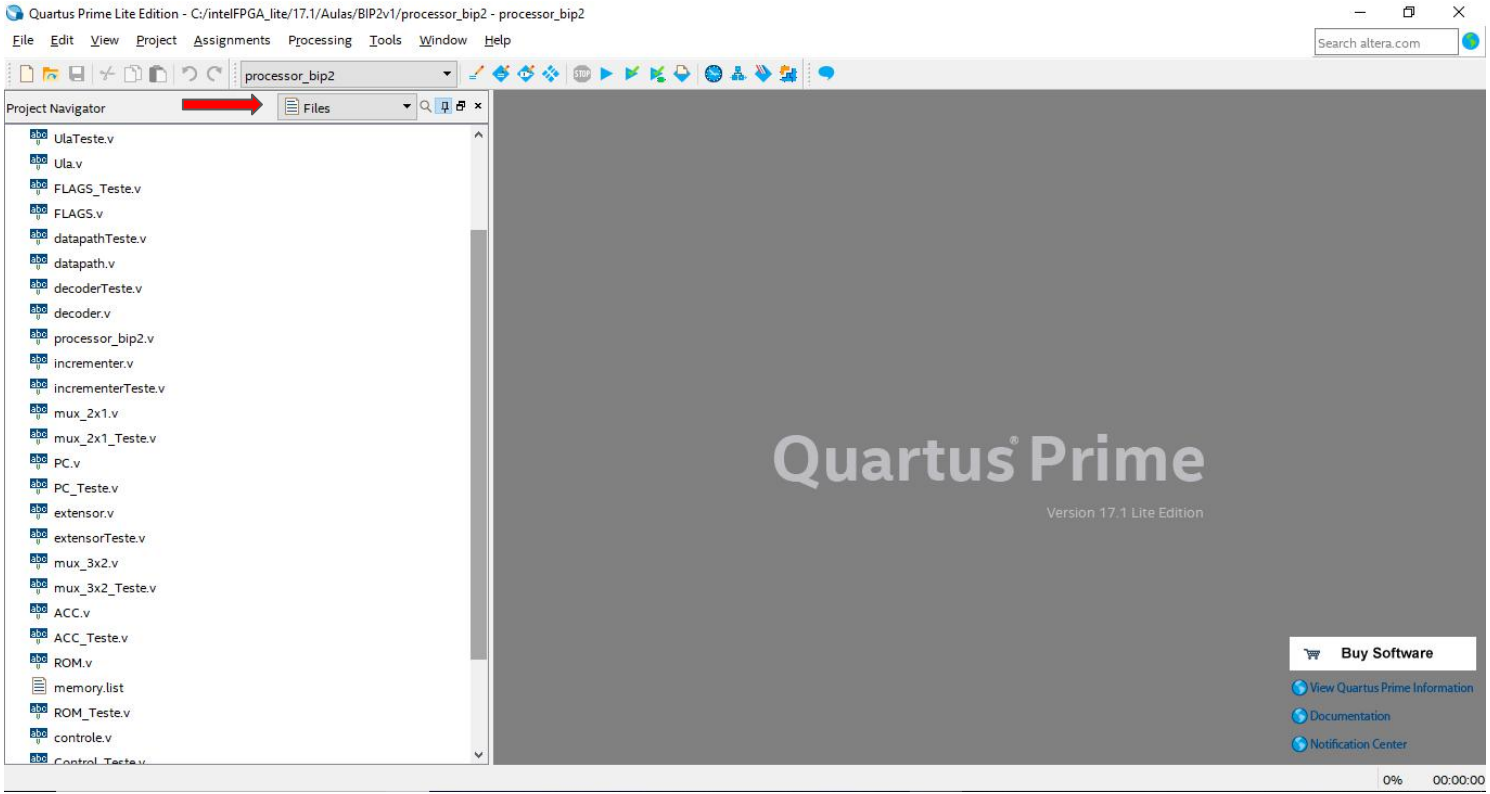
METODOLOGIA DE TESTES

- ➔ Separar a estrutura do processador em módulos;
- ➔ Testar as funcionalidades desses módulos separadamente;
- ➔ Agregar os módulos de forma a criarem uma nova entidade hierárquica;
- ➔ Testar a funcionalidade dessa entidade maior, que caracteriza o teste do funcion dos módulos.



- Módulos
- Entidade Hierárquica Maior

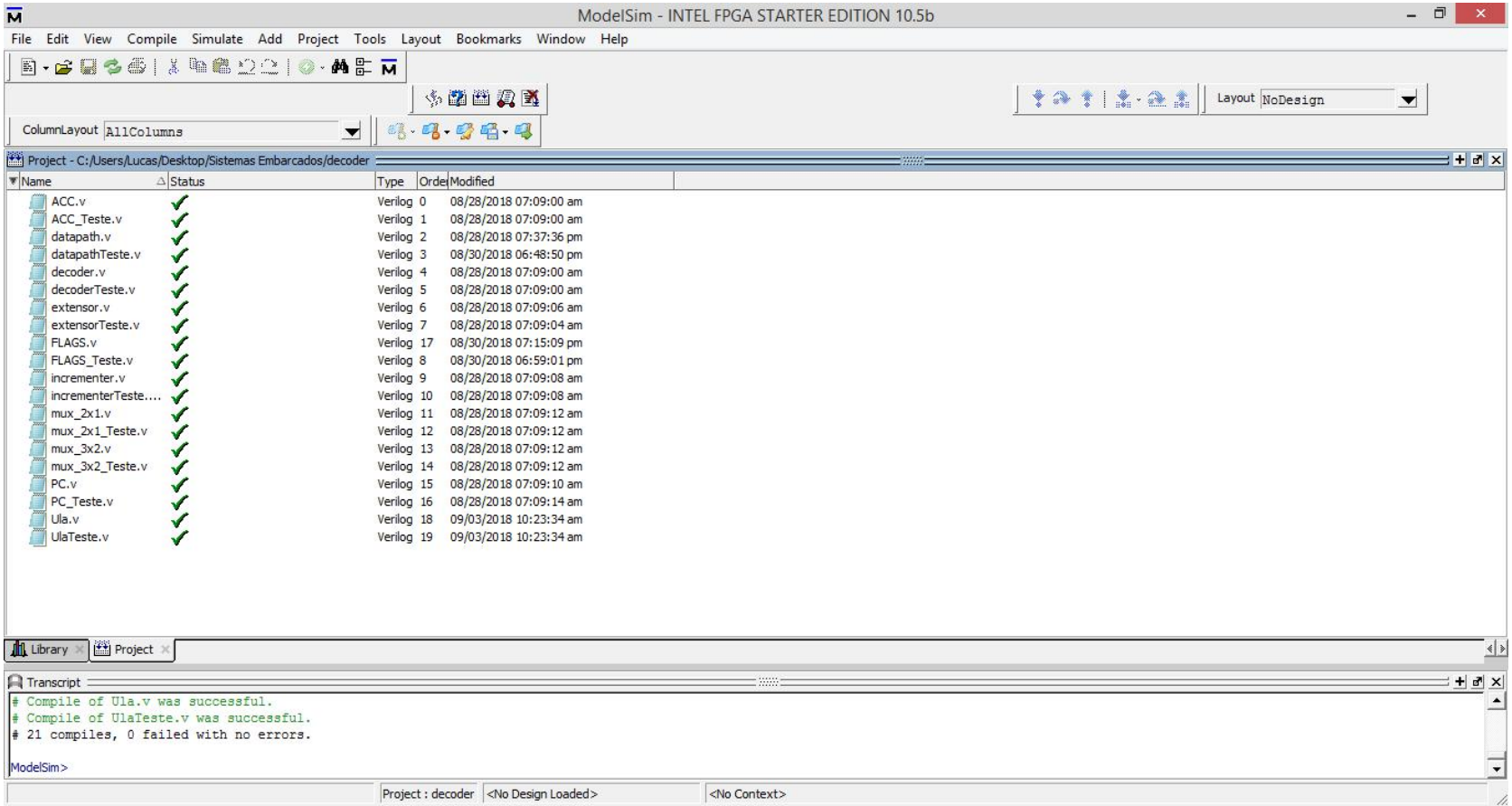
METODOLOGIA DE TESTES



METODOLOGIA DE TESTES

- Teste realizados com ModelSim ;
- Trata-se de um ambiente de simulação HDL multi-linguagem da Mentor Graphics;
- Usado para simulação de linguagens de descrição de hardware;
- Suporta VHDL, Verilog e SystemC, e inclui um depurador C embutido.

METODOLOGIA DE TESTES

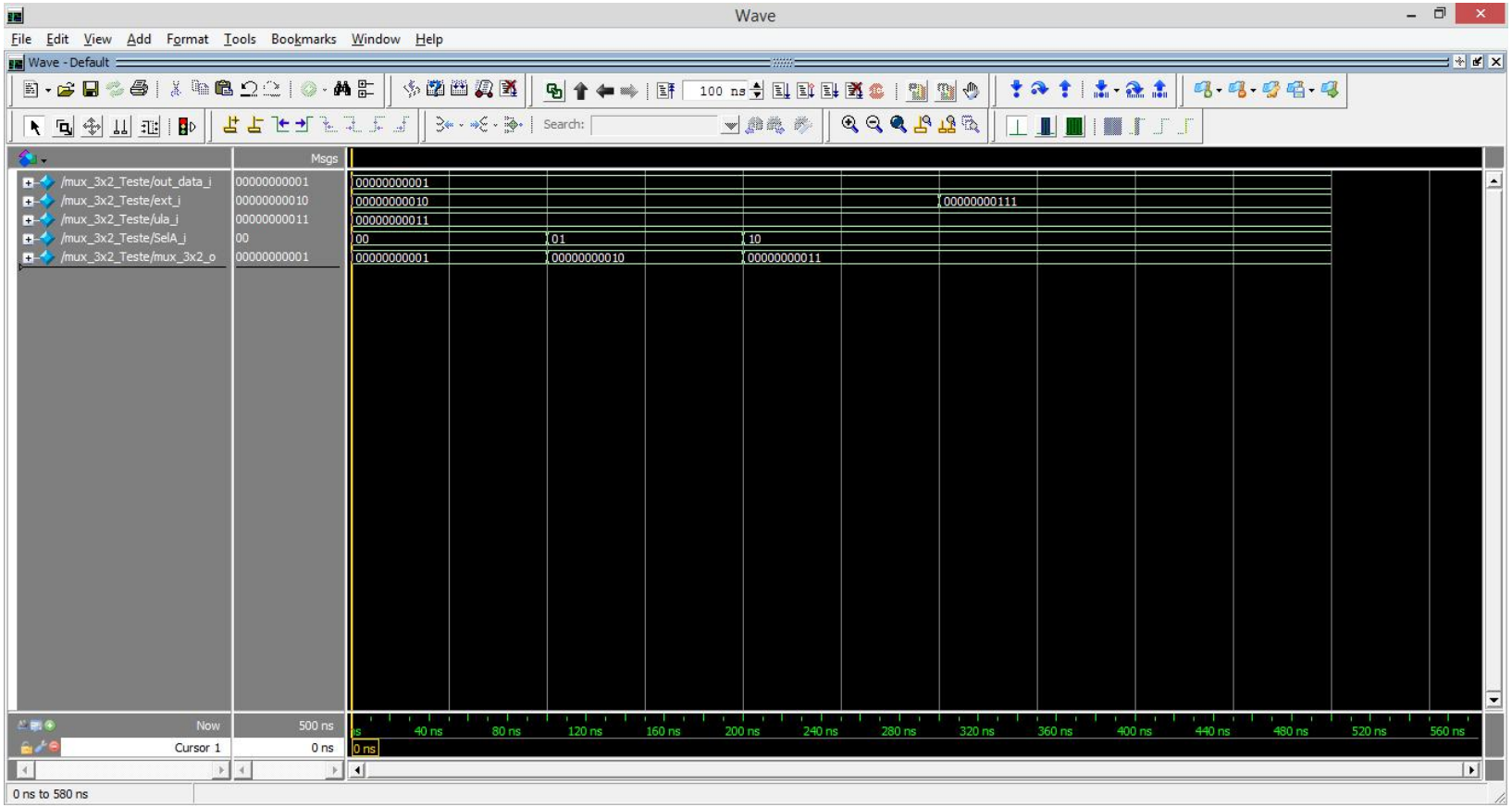


METODOLOGIA DE TESTES

```
module mux_3x2
#(
    parameter MSB_ROM = 11,
    parameter SELA = 2,
    parameter LSB = 0
)
(
    out_data_i, //Saída do incrementador
    ext_i, //Selecionador
    ula_i,
    SelA_i,

    mux_3x2_o//Saída do mux
);
  input wire [(MSB_ROM-1):LSB] out_data_i;
  input wire [(MSB_ROM-1):LSB] ext_i;
  input wire [(MSB_ROM-1):LSB] ula_i;
  input wire [(SELA-1):LSB] SelA_i;

  output reg [(MSB_ROM-1):LSB] mux_3x2_o;

  always @(out_data_i or ext_i or ula_i or SelA_i)
begin
    case(SelA_i)
    2'b0: mux_3x2_o = out_data_i;
    2'b1: mux_3x2_o = ext_i;
    2'b10: mux_3x2_o = ula_i;
    endcase
end
endmodule
```

```
`timescale 10 ns / 10 ns

module mux_3x2_Teste;

//Inputs
reg [10:0] out_data_i;
reg [10:0] ext_i;
reg [10:0] ula_i;
reg [1:0] SelA_i;

//Outputs
wire [10:0] mux_3x2_o;//Saída do mux

mux_3x2 DUT(out_data_i,ext_i, ula_i, SelA_i, mux_3x2_o);
initial
   begin
      out_data_i = 11'b1;
      ext_i = 11'b10;
      ula_i = 11'b11;
      SelA_i = 2'b0;

      #10 SelA_i = 2'b1;

      #10 SelA_i = 2'b10;

      #10 ext_i = 11'b111;

      #10 ext_i = 11'b111;

   end
endmodule
```

METODOLOGIA DE TESTES



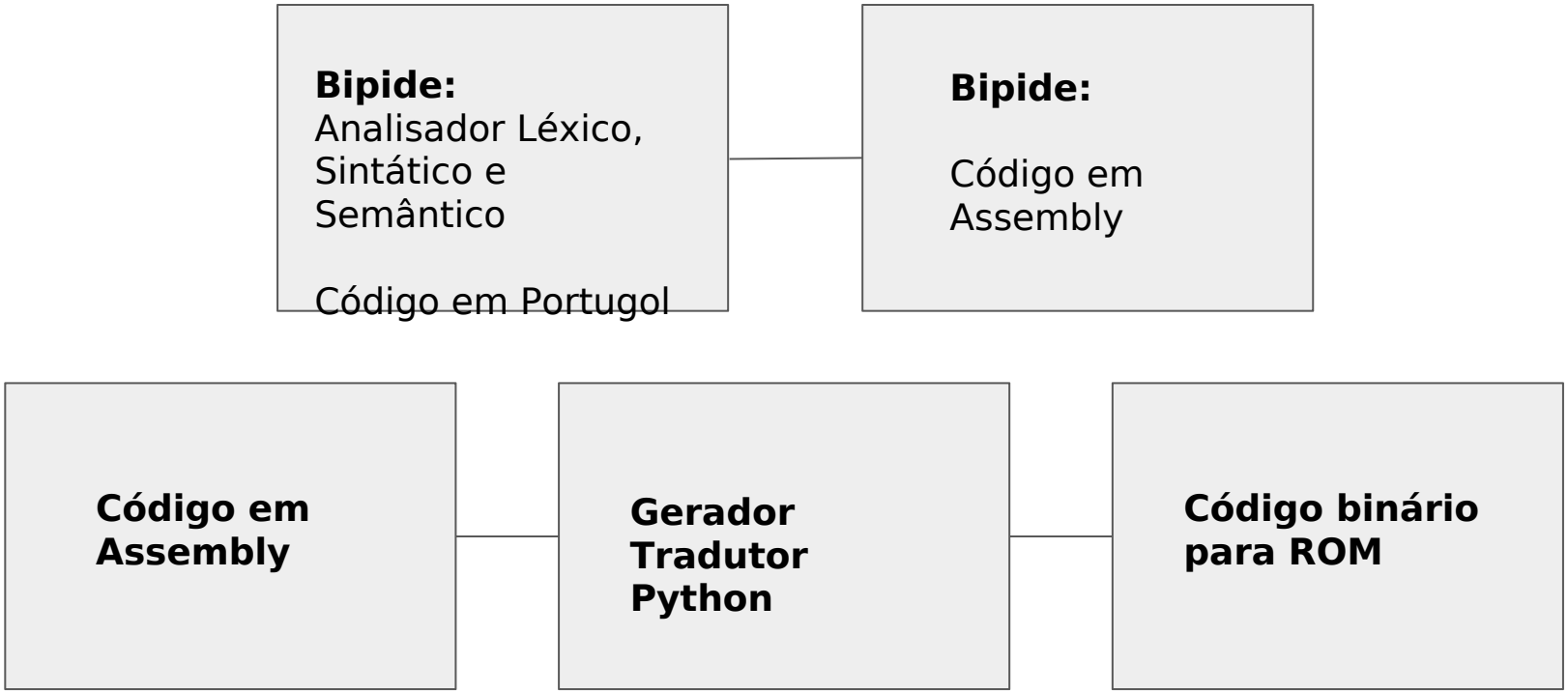
MEMÓRIAS - ROM E RAM

```
 2  module RAM
 3 #(
 4     parameter MEM_SIZE = 1024,
 5     //parameter MEM_SIZE = 4,
 6     parameter WORD_WIDTH = 11,
 7     parameter MSB_RAM = 11,
 8     parameter LSB = 0,
 9     parameter MSB = 0
10  )
11 (
12     WR_i,
13     ADDR_dm_i, //Endereço
14     IN_DATA_i,
15     OUT_DATA_o
16  );
17  input wire WR_i;
18  input wire [(MSB_RAM-1):LSB] ADDR_dm_i;
19  input wire [(WORD_WIDTH-1):LSB] IN_DATA_i;
20
21  output reg [(WORD_WIDTH-1):LSB] OUT_DATA_o;
22
23  reg [(WORD_WIDTH)-1:LSB] RAM_mem[MSB:(MEM_SIZE-1)];
24
```

```
25  initial
26 begin
27     $readmemb("C:/intelFPGA_lite/17.1/Aulas/BIP2v1/dataMemory.list", RAM_mem);
28  end
29  //BLOCO DE ESCRITA = escreve na memória em uma borda de subida de WR
30  always @(posedge WR_i)
31 begin
32        RAM_mem[ADDR_dm_i] = IN_DATA_i;
33  end
34  //BLOCO DE LEITURA = atualiza a saída sempre que um endereço muda
35  always @(ADDR_dm_i)
36 begin
37   OUT_DATA_o = RAM_mem[ADDR_dm_i];
38  end
39  endmodule
40
```

```
22  initial
23 begin
24     $readmemb("C:/intelFPGA_lite/17.1/Aulas/BIP2v1/programMemory.list", ROM_mem);
25  end
26  always @(ADDR_im_i)
27 begin
28     DATA_im_o = ROM_mem[ADDR_im_i];
29  end
30  endmodule
31
```

Gerador de código binário



REFERÊNCIAS

[1] http://bipide.com.br/bipide/

[2] https://www.altera.com/products/design-software/fpga-design/quartus-prime/overview.html

[3] https://www.embarcados.com.br/processador-em-verilog-3/

INSTITUTO FEDERAL
Fluminense

INSTITUTO FEDERAL
Fluminense

ENGENHARIA DA COMPUTAÇÃO