

Implementação M++ em FPGA

Aluno(a): André L. Bieging

Orientador: Miguel A. Wisintainer

Roteiro

- Introdução, objetivos;
- Fundamentação Teórica;
- Requisitos, especificação;
- Implementação;
- Operacionalidade;
- Resultados, conclusão.

Introdução

- Avanço dos semicondutores;
- Diferentes tecnologias;
- Possibilidade de um teste real.

Objetivos

- Portar a M++ para um FPGA
 - Carregar programas;
 - Ler entradas e produzir saídas;
 - Possibilidade de extensão da arquitetura;
 - Material de ensino.

Fundamentação Teórica

- Verilog;
- Quartus;
- ModelSIM.

Verilog - Módulos

```
module <module_name> (<module_terminal_list>);  
  
    ...  
    <module internals>  
    ...  
    ...  
endmodule
```

Verilog – Entradas e Saídas

```
module ram_decode  
(  
    en,  
    addr,  
    data  
);  
  
input en;  
input [3:0] addr;  
  
output [7:0] data;
```

Verilog - wire

```
module and_gate
(
    in1,
    in2,
    out1
);
input in1, in2;
output out1;
wire and_temp;
assign and_temp = in1 & in2;
assign out1 = and_temp;
endmodule
```


Verilog - reg

```
module and_gate
(
    in1,
    in2,
    out1
);

input in1, in2;

output out1;

reg out1;

always @ (in1 or in2)
begin
    out1 = in1 & in2;
end

endmodule
```

Verilog - Delay

```
always  
begin  
    #10 clk = !clk;  
end
```

Verilog - initial

```
initial  
begin  
    data = 8'b00000000;  
end
```

Verilog - always

```
always
begin
    clk = ~clk;
end

always @ (var1)
begin
    out = var1;
end

always @ (posedge var1)
begin
    out = var1;
end

always @ (var1 or var2)
begin
    out = var1 & var2;
end
```

Verilog – Sintaxe Variável

```
<tipo> [<tamanho>] <nome>;  
reg [7:0] var1;
```

Verilog – Sintaxe Array

```
<tipo> [<tamanho_vetor>] <nome> [<tamanho_array>];  
reg [7:0] var1 [0:15];
```

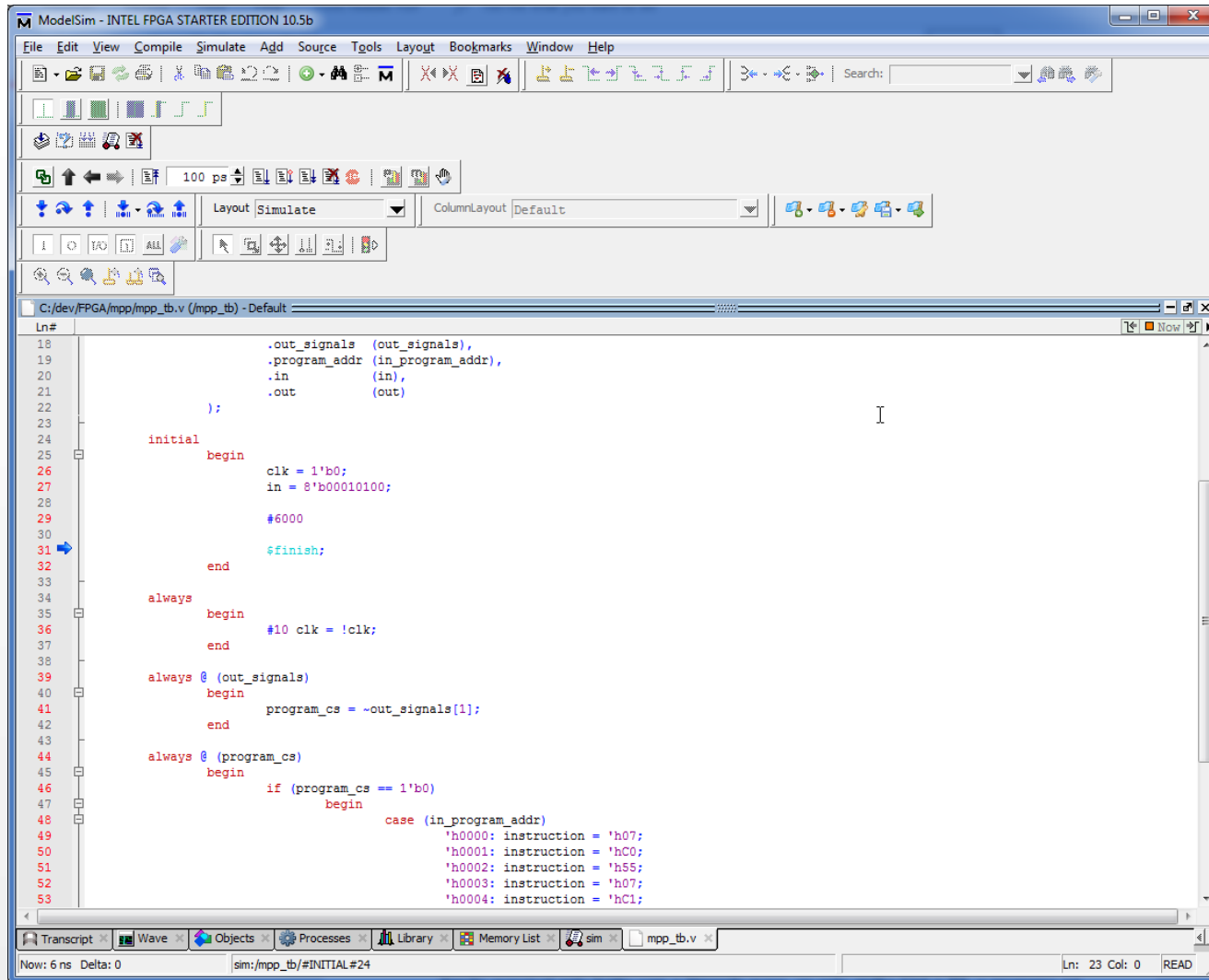
Quartus

The image shows the Quartus Prime Lite Edition software interface. The main window displays a Verilog HDL file named `mpp_tb.v`. The interface is annotated with large black numbers 1 through 6:

- 1**: Points to the Project Navigator on the left, specifically the Hierarchy view showing the project structure.
- 2**: Points to the Tasks window on the left, showing the compilation process.
- 3**: Points to the Messages window at the bottom, which is currently empty.
- 4**: Points to the top toolbar, specifically the Run button.
- 5**: Points to the main code editor area where the Verilog code is written.
- 6**: Points to the IP Catalog on the right side of the interface.

```
1 module mpp_tb ();
2   reg clk;
3   reg [7:0] in;
4
5   wire [7:0] out;
6   wire [4:0] out_signals;
7   wire [15:0] in_program_addr;
8
9   reg [7:0] instruction;
10  reg program_cs;
11
12  mpp m1
13  (
14    .clk          (clk),
15    .instruction  (instruction),
16    .out_signals  (out_signals),
17    .program_addr (in_program_addr),
18    .in           (in),
19    .out          (out)
20  );
21
22  initial
23  begin
24    clk = 1'b0;
25    in  = 8'b00010100;
26    #6000
27    $finish;
28  end
29
30  always
31  begin
32    #10 clk = !clk;
33  end
34
35  always @ (out_signals)
36  begin
37    program_cs = ~out_signals[1];
38  end
39
40  always @ (program_cs)
41  begin
42    if (program_cs == 1'b0)
43    begin
44      case (in_program_addr)
45
```

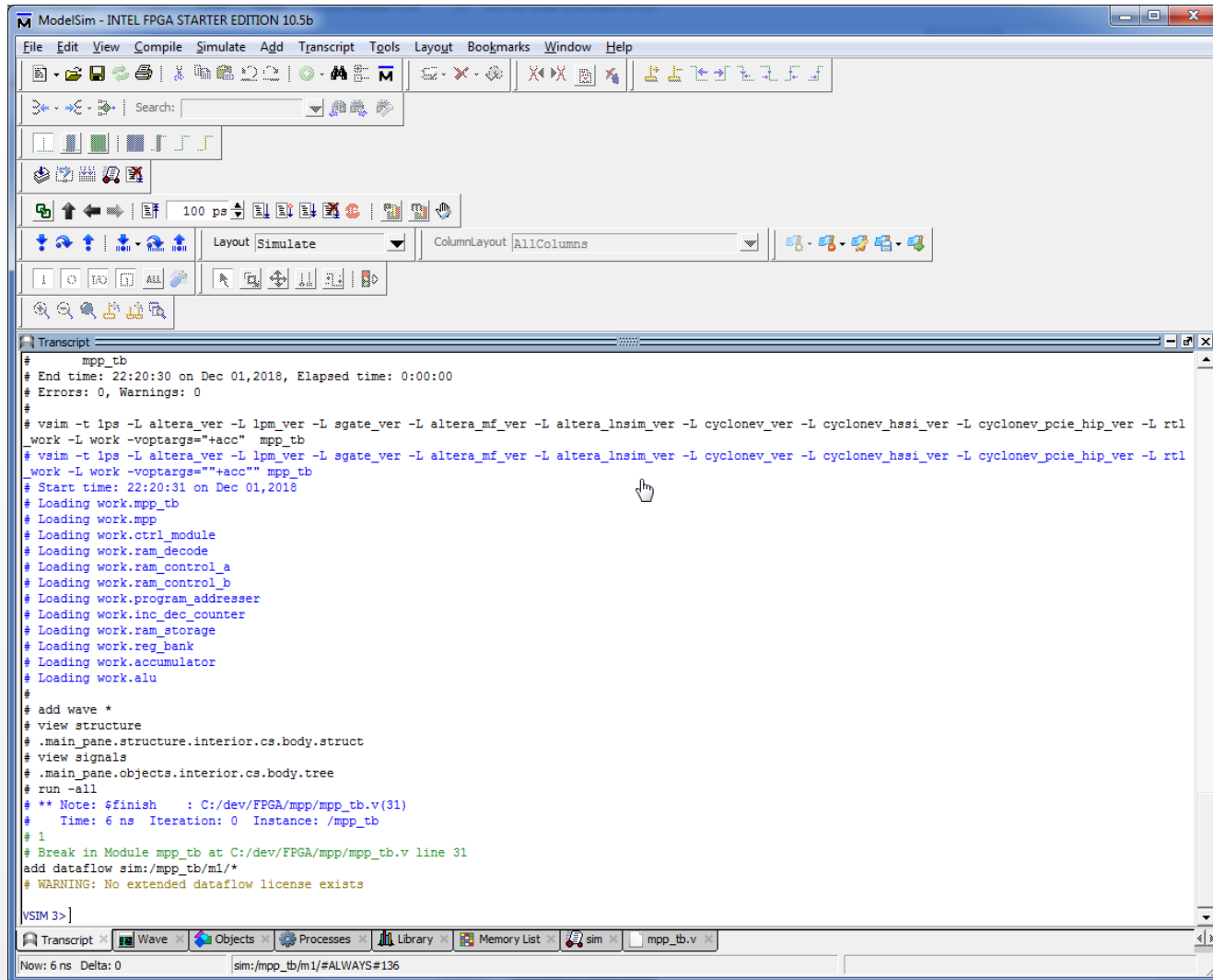
ModelSIM



```
ModelSim - INTEL FPGA STARTER EDITION 10.5b
File Edit View Compile Simulate Add Source Tools Layout Bookmarks Window Help
C:/dev/FPGA/mpp/mpp_tb.v (mpp_tb) - Default
Ln#
18         .out_signals (out_signals),
19         .program_addr (in_program_addr),
20         .in           (in),
21         .out          (out)
22     );
23
24     initial
25     begin
26         clk = 1'b0;
27         in = 8'b00010100;
28
29         #6000
30
31         $finish;
32     end
33
34     always
35     begin
36         #10 clk = !clk;
37     end
38
39     always @ (out_signals)
40     begin
41         program_cs = ~out_signals[1];
42     end
43
44     always @ (program_cs)
45     begin
46         if (program_cs == 1'b0)
47             begin
48                 case (in_program_addr)
49                     'h0000: instruction = 'h07;
50                     'h0001: instruction = 'hC0;
51                     'h0002: instruction = 'h55;
52                     'h0003: instruction = 'h07;
53                     'h0004: instruction = 'hC1;
```

Now: 6 ns Delta: 0 sim:/mpp_tb/#INITIAL#24 Ln: 23 Col: 0 READ

ModelSIM



The screenshot displays the ModelSim software interface. The main window title is "ModelSim - INTEL FPGA STARTER EDITION 10.5b". The menu bar includes File, Edit, View, Compile, Simulate, Add, Transcript, Tools, Layout, Bookmarks, Window, and Help. The toolbar contains various icons for file operations, simulation control, and layout management. The Transcript window is open, showing the following text:

```
# mpp_tb
# End time: 22:20:30 on Dec 01,2018, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
#
# vsim -t lps -L altera_ver -L lpm_ver -L sgate_ver -L altera_mf_ver -L altera_insim_ver -L cyclonev_ver -L cyclonev_hssi_ver -L cyclonev_pcie_hip_ver -L rtl
_work -L work -voptargs="+acc" mpp_tb
# vsim -t lps -L altera_ver -L lpm_ver -L sgate_ver -L altera_mf_ver -L altera_insim_ver -L cyclonev_ver -L cyclonev_hssi_ver -L cyclonev_pcie_hip_ver -L rtl
_work -L work -voptargs="+acc" mpp_tb
# Start time: 22:20:31 on Dec 01,2018
# Loading work.mpp_tb
# Loading work.mpp
# Loading work.ctrl_module
# Loading work.ram_decode
# Loading work.ram_control_a
# Loading work.ram_control_b
# Loading work.program_addresser
# Loading work.inc_dec_counter
# Loading work.ram_storage
# Loading work.reg_bank
# Loading work.accumulator
# Loading work.alu
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Note: $finish      : C:/dev/FPGA/mpp/mpp_tb.v(31)
#      Time: 6 ns Iteration: 0 Instance: /mpp_tb
# 1
# Break in Module mpp_tb at C:/dev/FPGA/mpp/mpp_tb.v line 31
add dataflow sim:/mpp_tb/m1/*
# WARNING: No extended dataflow license exists

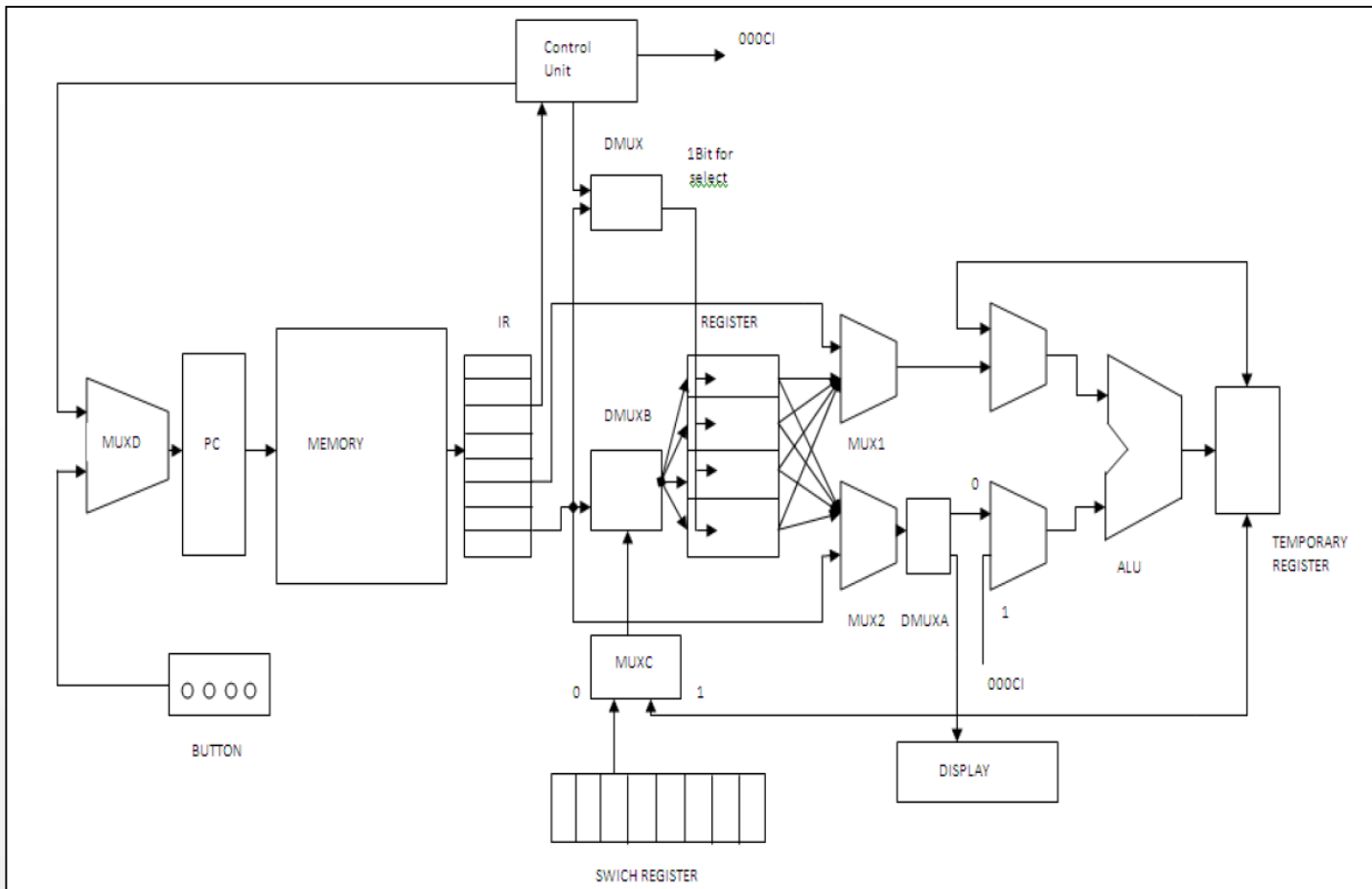
VSIM 3>
```

The status bar at the bottom shows "Now: 6 ns Delta: 0" and "sim:/mpp_tb/m1/#ALWAYS#136". The taskbar at the bottom of the window lists several open tabs: Transcript, Wave, Objects, Processes, Library, Memory List, sim, and mpp_tb.v.

Trabalhos Correlatos

- FPGA IMPLEMENTATION OF AN 8-BIT SIMPLE PROCESSOR;
- A VERY SIMPLE 8-BIT RISC PROCESSOR FOR FPGA;
- DESIGN OF A GENERAL PURPOSE 8-BIT RISC PROCESSOR FOR COMPUTER ARCHITECTURE LEARNING.

FPGA IMPLEMENTATION OF AN 8-BIT SIMPLE PROCESSOR



Fonte: Ayeh et al., (2008).

A VERY SIMPLE 8-BIT RISC PROCESSOR FOR FPGA

The screenshot displays the IDEuva1 software interface, which is used for simulating the RISC processor. The interface is divided into several sections:

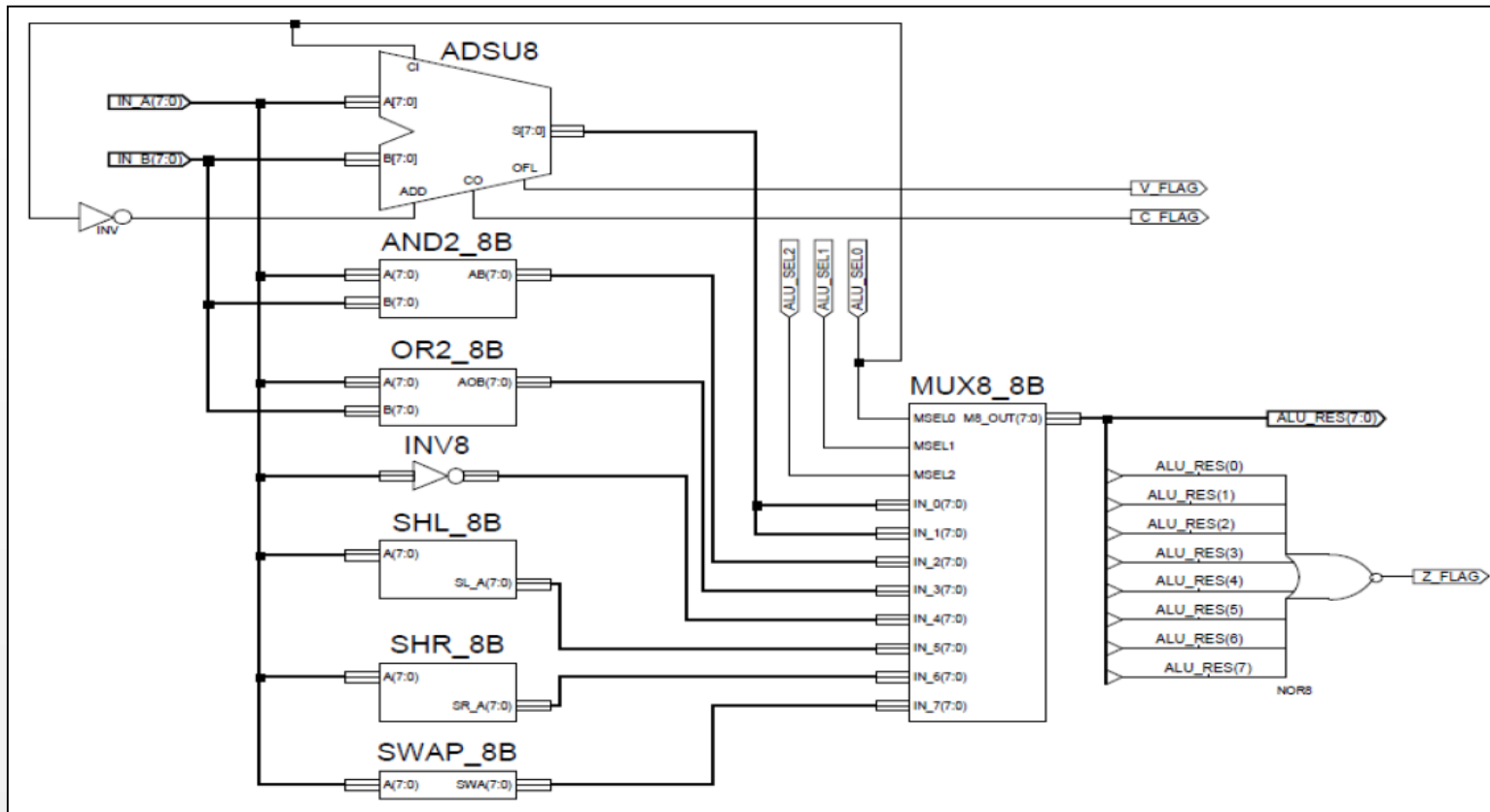
- Assembly Code (1):** A list of assembly instructions with their addresses. The instructions are:

```
0x0002: inc r14
0x0003: adc r15, r0
0x0004: mov [224], r14
0x0005: mov {++}, r15
0x0006: pop r2
0x0007: reti
0x0008: mov r0, 0 <Main>
0x0009: mov r1, 1
0x000A: mov r14, 0
0x000B: mov r15, 0
0x000C: ei
0x000D: mov r3, 0
0x000E: mov r4, 16
0x000F: mov r2, {r3} <StrCpy>
0x0010: mov {r4}, r2
```
- Internal Registers (2):** A table showing the state of 16 internal registers (R0 to R15). Each register is represented by a gauge and contains the value 255 - 0xFF. The registers are labeled R8 through R15 on the right side of the table. Below the table are buttons for 'A/B' and 'D/H'.
- Message Log (3):** A log window showing the execution progress. The messages are:

```
Opening file 'H:/FPGAworld2006/RISCuva1/Demo_FPGAworld2006.asm'.
Trying to connect with the emulator ...
Connected and synchronized.
Processing 'H:/FPGAworld2006/RISCuva1/Demo_FPGAworld2006.asm' ...
RISCuva1 Macro-Assembler v3.01
Assembler finished ok (Program = 24 words; Data = 0 bytes; Ports = 226 bytes).
Physical processor on Reset state.
```
- Processor State:** A section on the right side of the message log containing status indicators for V, S, C, Z, PC (0x00000), SP (0), and CLK.

Fonte: PABLO et al., (2016).

DESIGN OF A GENERAL PURPOSE 8-BIT RISC PROCESSOR FOR COMPUTER ARCHITECTURE LEARNING



Fonte: Zalava et al., (2015).

Arquitetura Atual

- M++;
- CPU hipotética;

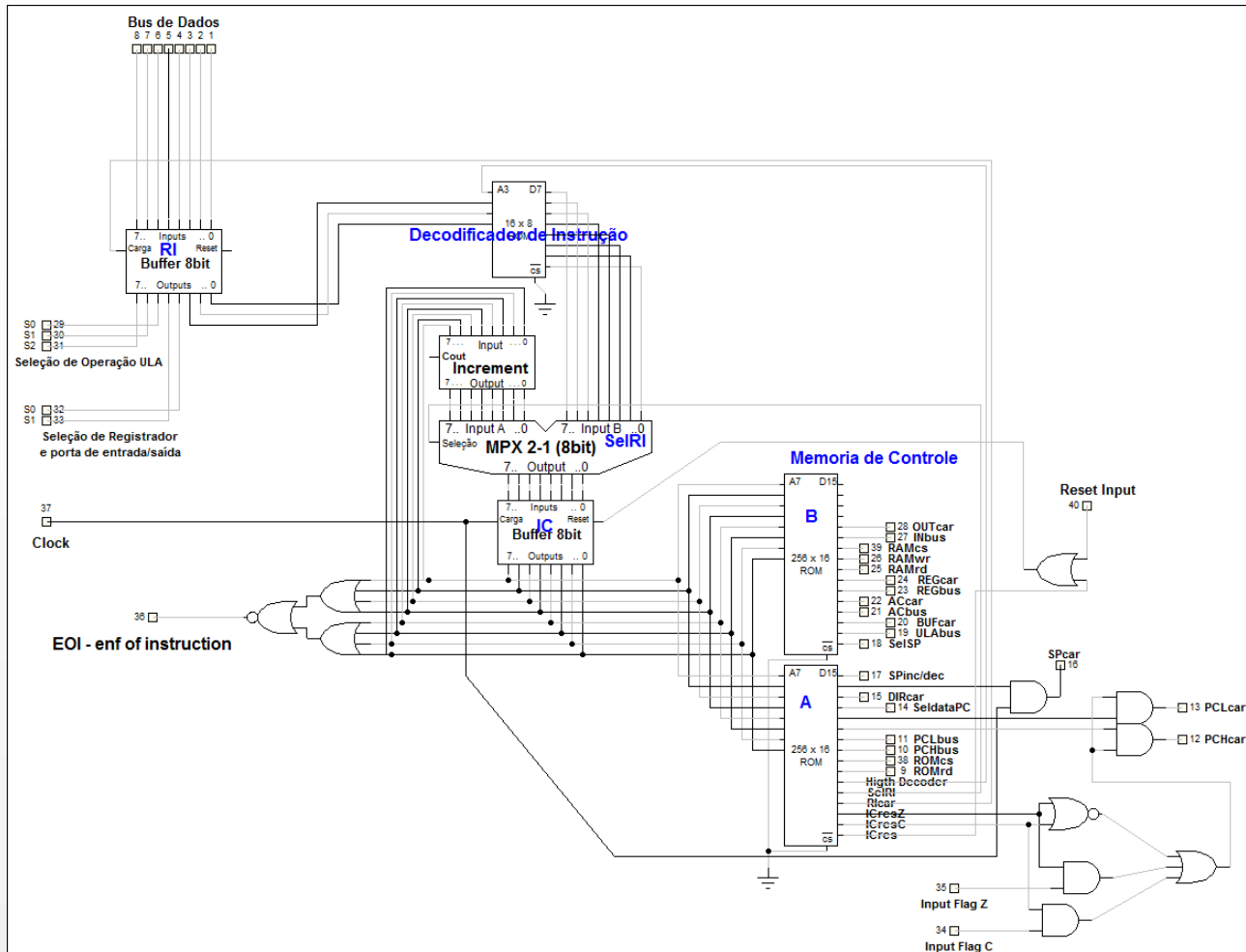
Requisitos Funcionais

- RF1 - armazenar programa em memória interna;
- RF2 - ler programa armazenado em memória interna;
- RF3 - ler entradas acionadas pelo usuário;
- RF4 - acionar saídas conforme processamento do programa.

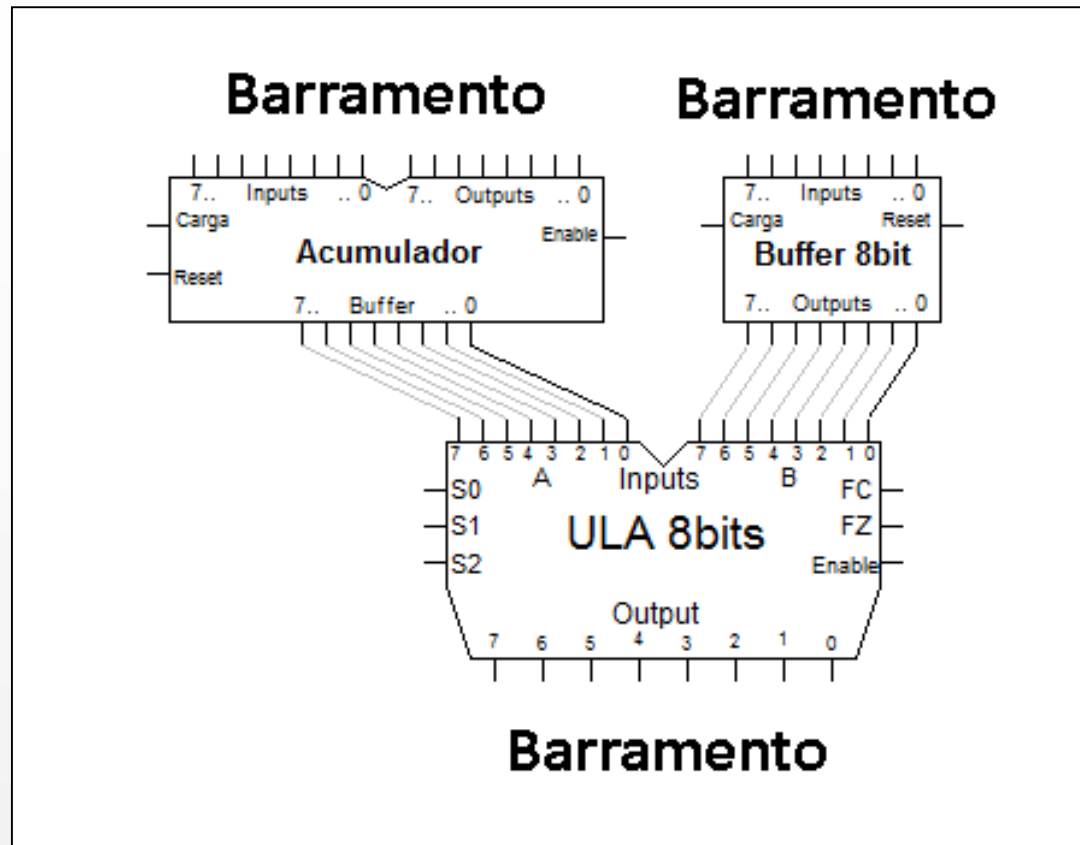
Requisitos Não Funcionais

- RNF1 - ser programado na linguagem de descrição de hardware Verilog no ambiente de programação Intel Quartus Prime;
- RNF2 - ter a funcionalidade testada e comprovada com o software de simulação ModelSim*-Intel® FPGA edition;
- RNF3 - ser implementado em um kit de desenvolvimento da Altera, com um FPGA da família Cyclone V;
- RNF4 - funcionar em um clock superior à 10KHz.

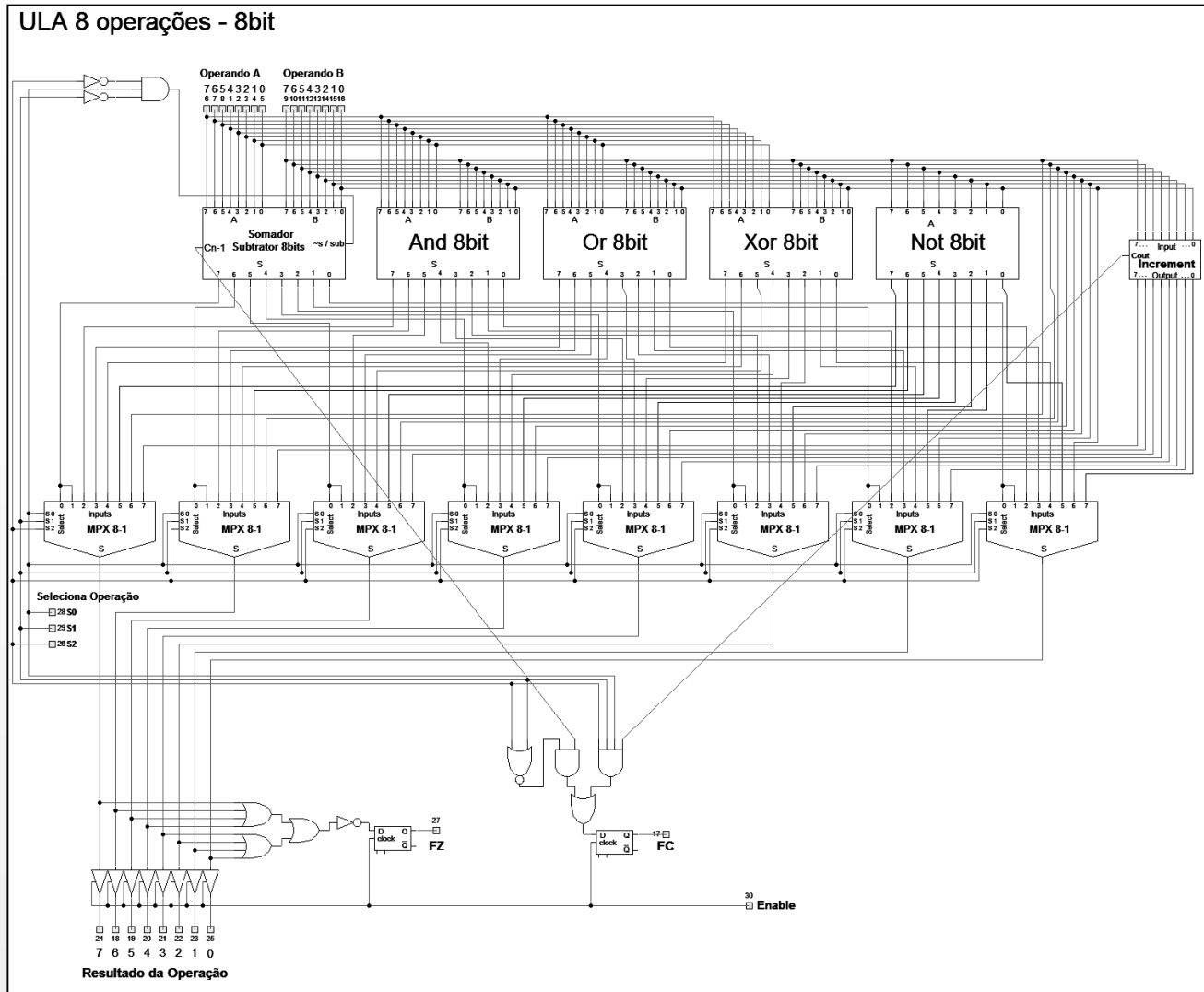
Módulo de Controle



ULA

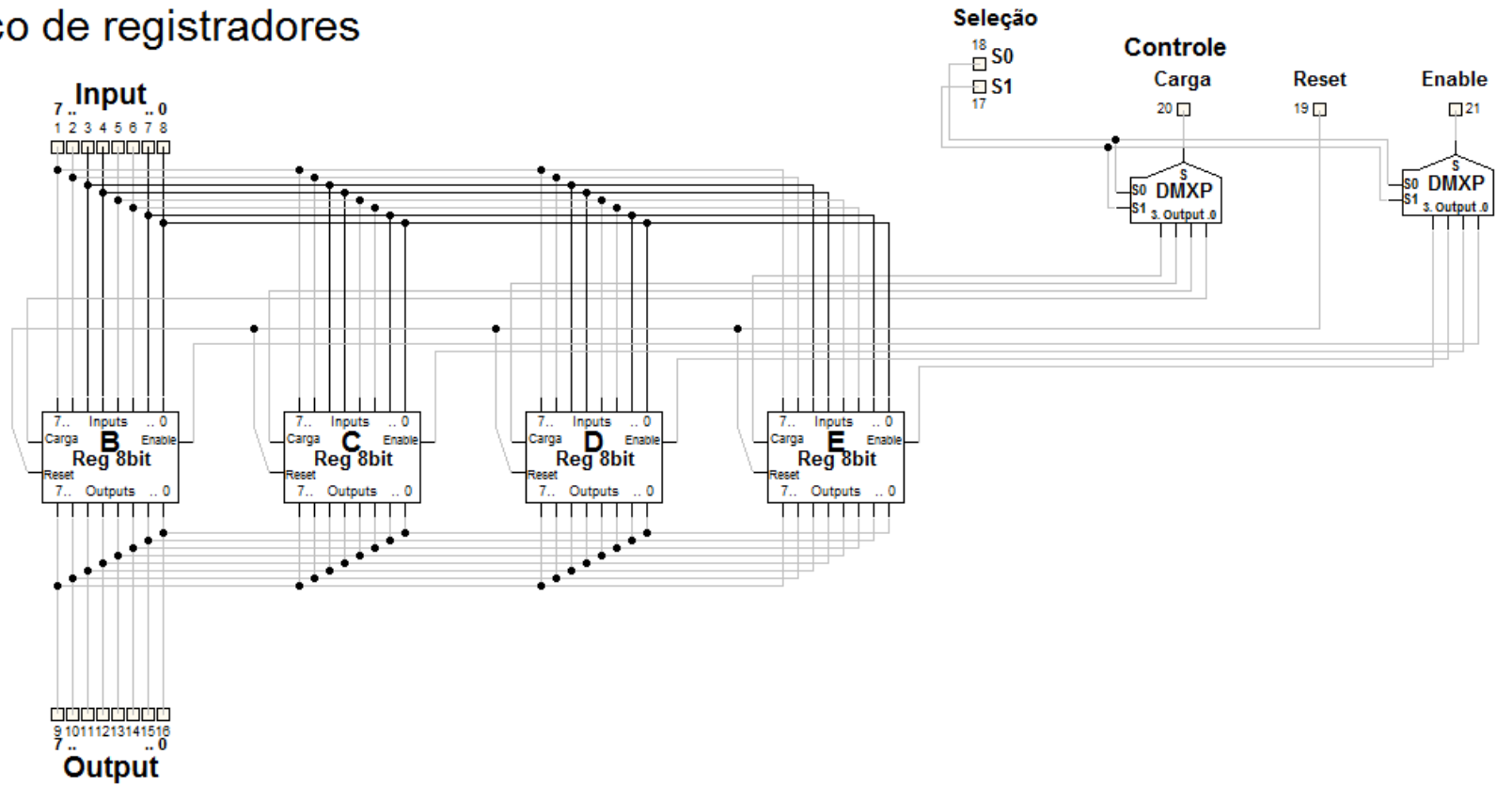


ULA - Interno

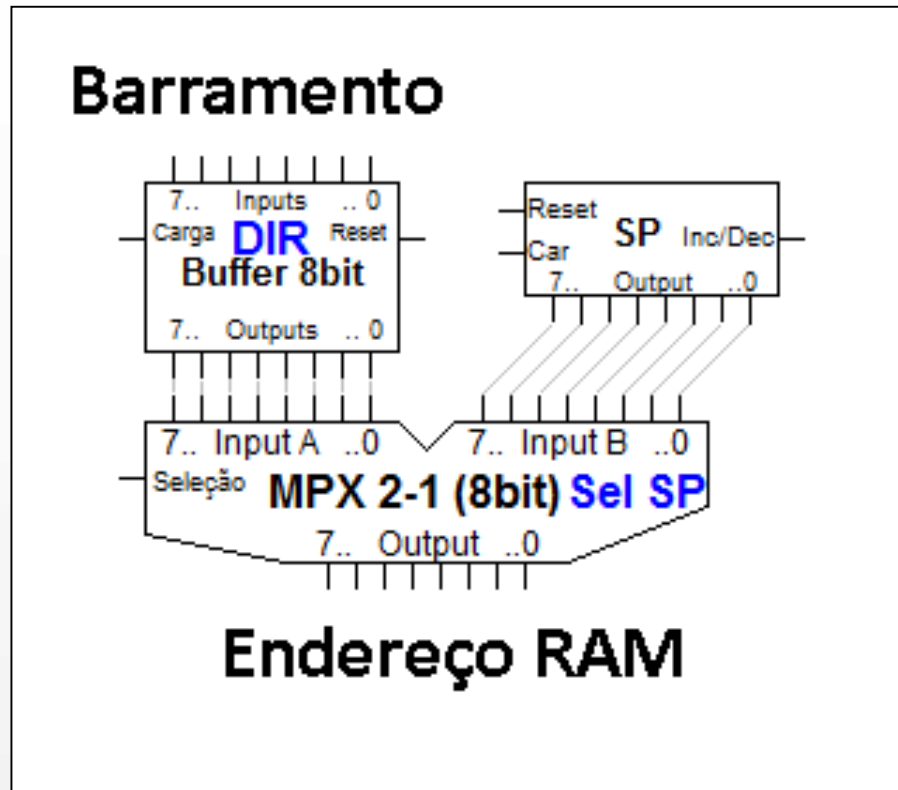


Banco de Registradores

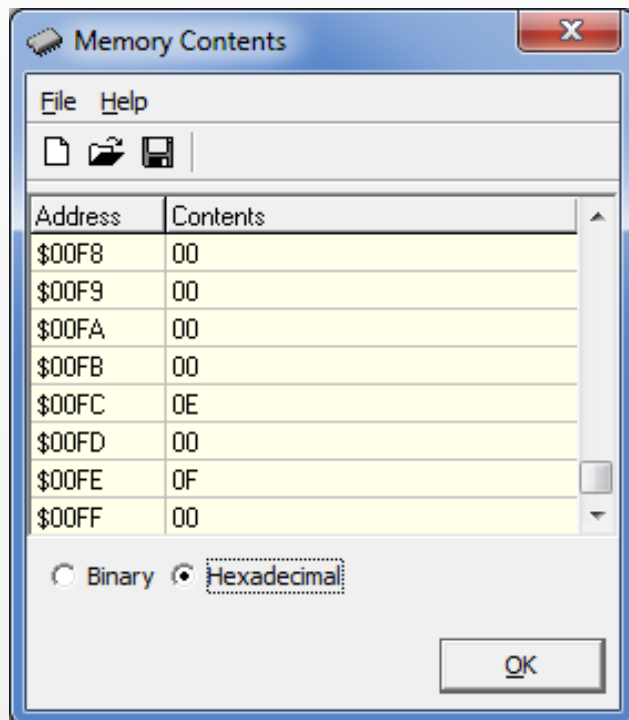
Banco de registradores



Endereçador da Memória RAM



Call Stack



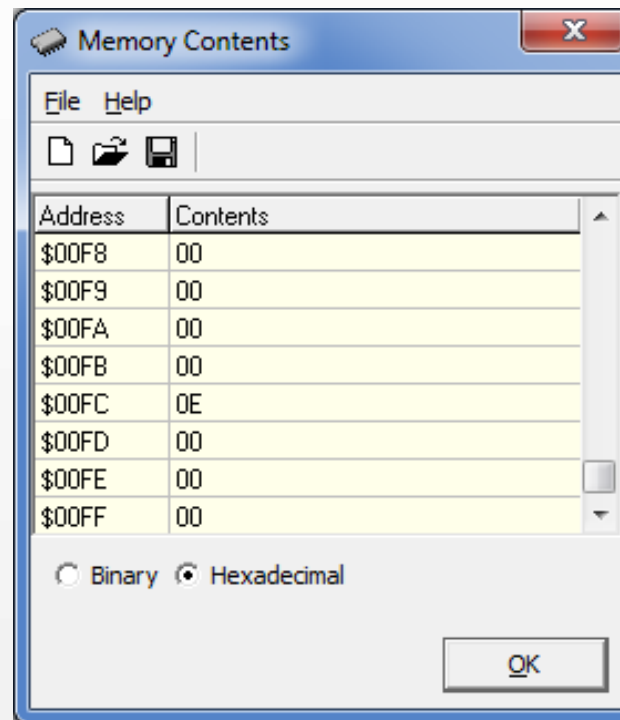
Memory Contents

File Help

| Address | Contents |
|---------|----------|
| \$00F8 | 00 |
| \$00F9 | 00 |
| \$00FA | 00 |
| \$00FB | 00 |
| \$00FC | 0E |
| \$00FD | 00 |
| \$00FE | 0F |
| \$00FF | 00 |

Binary Hexadecimal

OK



Memory Contents

File Help

| Address | Contents |
|---------|----------|
| \$00F8 | 00 |
| \$00F9 | 00 |
| \$00FA | 00 |
| \$00FB | 00 |
| \$00FC | 0E |
| \$00FD | 00 |
| \$00FE | 00 |
| \$00FF | 00 |

Binary Hexadecimal

OK

```
Loop:  
CALL SUBROT;  
JMP LOOP;  
  
SUBROT:  
RET;
```

Implementação - Módulo M++

```
ctrl_module cm
(
    .clk          (clk),
    .instruction  (instruction),
    .ctrl_signals (ctrl_signals),
    .sp_car      (sp_car)
);

program_addresser pa
(
    .ctrl_signals (program_signals),
    .reset        (reset),
    .in           (bus),
    .out_low      (program_low),
    .out_high     (program_high)
);

reg_bank rb
(
    .sel         (reg_bank_sel),
    .en          (ctrl_signals[22]),
    .load        (ctrl_signals[21]),
    .reset       (reset),
    .in          (bus),
    .out         (reg_bank_out)
);

accumulator acc
(
    .en          (ctrl_signals[24]),
    .load        (ctrl_signals[23]),
    .reset       (reset),
    .in          (bus),
    .out         (acc_out),
    .buffer      (acc_buffer)
);

alu al
(
    .sel         (alu_sel),
    .en          (ctrl_signals[26]),
    .in_a        (acc_buffer),
    .in_b        (alu_buffer),
    .flag_c      (alu_fc),
    .flag_z      (alu_fz),
    .out         (alu_out)
);
```

Módulo M++

```
always @ (*)
begin
  if (out_signals[0] == 1'b1)
    begin
      bus = instruction;
    end
  else if (storage_cs == 1'b1)
    begin
      bus = storage_data_out;
    end
  else if (ctrl_signals[2] == 1'b1)
    begin
      bus = program_high;
    end
  else if (ctrl_signals[3] == 1'b1)
    begin
      bus = program_low;
    end
  else if (ctrl_signals[22] == 1'b1)
    begin
      #5
      bus = reg_bank_out;
    end
  else if (ctrl_signals[24] == 1'b1)
    begin
      #5
      bus = acc_out;
    end
  else if (ctrl_signals[26] == 1'b1)
    begin
      #5
      bus = alu_out;
    end
  else if (ctrl_signals[17] == 1'b1)
    begin
      bus = in;
    end
end
```

Módulo de Controle

```
always @ (posedge ctrl_a_data[3])
begin
    decode_address[0] = instruction[0];
    decode_address[1] = instruction[1];
    decode_address[2] = instruction[2];

    ctrl_signals[14] = instruction[3]; // REG0
    ctrl_signals[15] = instruction[4]; // REG1

    ctrl_signals[11] = instruction[5]; // ULA0
    ctrl_signals[12] = instruction[6]; // ULA1
    ctrl_signals[13] = instruction[7]; // ULA2
end
```

Módulo de Controle

```
always @ (posedge clk)
begin
    // Instruction buffer reset.
    if (ctrl_a_data[0] == 1'b1)
        begin
            ctrl_addr = 8'b00000000;
        end
    else
        begin
            ctrl_addr <= decode_reg;
        end
    end
end
```

Módulo de Controle

```
always @ (decode_data or ctrl_addr)
  begin
    #5
    if (ctrl_a_data[4] == 1'b0)
      begin
        decode_reg = incr_result;
      end
    else
      begin
        decode_reg = decode_data;
      end
    end
  end
```

Módulo de Controle

```
nor u1 (pchl_condition, ctrl_a_data[1], ctrl_a_data[2]);
always @ (ctrl_a_data or ctrl_b_data)
begin
    // 4th address decoder signal.
    decode_address[3] = ctrl_a_data[5];

    // Control signals.
    ctrl_signals[0] = ctrl_a_data[6]; // ROMrd
    ctrl_signals[1] = ctrl_a_data[7]; // ROMcs
    ctrl_signals[2] = ctrl_a_data[8]; // PCHbus
    ctrl_signals[3] = ctrl_a_data[9]; // PCLbus

    // PCHcar
    if (ctrl_a_data[10] == 1'b1 & pchl_condition == 1'b1)
        begin
            ctrl_signals[4] = 1'b1;
        end
    else
        begin
            ctrl_signals[4] = 1'b0;
        end

    // PCLcar
    if (ctrl_a_data[11] == 1'b1 & pchl_condition == 1'b1)
        begin
            ctrl_signals[5] = 1'b1;
        end
    else
        begin
            ctrl_signals[5] = 1'b0;
        end

    // Reception of all other signals hidden.
    #5 // signals that have to wait to avoid race condition.
    ctrl_signals[6] = ctrl_a_data[12]; // selDataPC
end
```


ULA

```
always @ (*)
begin
  if (en == 1'b1)
  begin
    case (sel)
      3'b000: out = in_a + in_b;
      3'b001: out = in_a - in_b;
      3'b010: out = in_a & in_b;
      3'b011: out = in_a | in_b;

      3'b100: out = in_a ^ in_b;
      3'b101: out = !in_b;
      3'b110: out = in_b;
      3'b111: out = in_b + 1;
    endcase
  end
else
begin
  out = 8'bzzzzzzzz;
end
end
```

Endereçador da Memória do Programa

```
// Assigns.
assign incr_result = incr_input + 1;

// Always.
always @ (ctrl_signals)
begin
    // Selection (SelDataPC).
    if (ctrl_signals[0] == 1'b0)
        begin
            mid_low[0] = incr_result[0];
            mid_low[1] = incr_result[1];
            mid_low[2] = incr_result[2];
            mid_low[3] = incr_result[3];

            mid_low[4] = incr_result[4];
            mid_low[5] = incr_result[5];
            mid_low[6] = incr_result[6];
            mid_low[7] = incr_result[7];

            mid_high[0] = incr_result[8];
            mid_high[1] = incr_result[9];
            mid_high[2] = incr_result[10];
            mid_high[3] = incr_result[11];

            mid_high[4] = incr_result[12];
            mid_high[5] = incr_result[13];
            mid_high[6] = incr_result[14];
            mid_high[7] = incr_result[15];
        end
    else
        begin
            mid_low[0] = in[0];
            mid_low[1] = in[1];
            mid_low[2] = in[2];
            mid_low[3] = in[3];

            mid_low[4] = in[4];
            mid_low[5] = in[5];
            mid_low[6] = in[6];
            mid_low[7] = in[7];

            mid_high[0] = in[0];
            mid_high[1] = in[1];
            mid_high[2] = in[2];
            mid_high[3] = in[3];

            mid_high[4] = in[4];
            mid_high[5] = in[5];
            mid_high[6] = in[6];
            mid_high[7] = in[7];
        end
end
end
```

Endereçador da Memória do Programa

```
// Always.
always @ (ctrl_signals)
begin
    // clock L (PCLcar).
    if (ctrl_signals[2] == 1'b1 & prev_pcl_state == 1'b0)
        begin
            out_low = mid_low;

            incr_input[0] = out_low[0];
            incr_input[1] = out_low[1];
            incr_input[2] = out_low[2];
            incr_input[3] = out_low[3];

            incr_input[4] = out_low[4];
            incr_input[5] = out_low[5];
            incr_input[6] = out_low[6];
            incr_input[7] = out_low[7];
        end

    prev_pcl_state = ctrl_signals[2];

    // clock H (PCHcar).
    if (ctrl_signals[1] == 1'b1 & prev_pch_state == 1'b0)
        begin
            out_low = mid_low;

            incr_input[8] = out_high[0];
            incr_input[9] = out_high[1];
            incr_input[10] = out_high[2];
            incr_input[11] = out_high[3];

            incr_input[12] = out_high[4];
            incr_input[13] = out_high[5];
            incr_input[14] = out_high[6];
            incr_input[15] = out_high[7];
        end

    prev_pch_state = ctrl_signals[1];
end
```

Banco de Registradores

```
reg [7:0] regs [0:3];
integer i;
initial
  begin
    out = 8'b00000000;
    for (i = 0; i < 4; i = i + 1)
      begin
        regs[i] = 8'b00000000;
      end
  end
```

Banco de Registradores

```
always @ (sel or en or load or reset)
begin
    if (en == 1'b1)
        begin
            out = regs[sel];
        end

    if (load == 1'b1)
        begin
            regs[sel] = in;
        end

    if (reset == 1'b1)
        begin
            for (i = 0; i < 4; i = i + 1)
                begin
                    regs[i] = 8'b00000000;
                end
        end
end
```

Endereçador da Memória RAM

```
initial
begin
    out = 8'b00000000;
end

always @ (posedge set)
begin
    if (reset == 1'b1)
        begin
            out = 8'b00000000;
        end
    else if (ctrl == 1'b0)
        begin
            out = out + 1;
        end
    else if (ctrl == 1'b1)
        begin
            out = out - 1;
        end
    end
end
```

Endereçador da Memória RAM

```
always @ (ctrl_signals)
begin
    storage_rw = ctrl_signals[20]; // RAMrd

    // DIR buffer.
    if (ctrl_signals[7] == 1'b1)
        begin
            dir_sp_buffer = bus;
        end

    // sel SP MUX.
    if (ctrl_signals[10] == 1'b0)
        begin
            storage_addr = dir_sp_buffer;
        end
    else
        begin
            storage_addr = storage_signals;
        end

    if (ctrl_signals[16] == 1'b1)
        begin
            out <= bus;
        end

    #5
    storage_cs = ctrl_signals[18]; // RAMcs
    #10
    storage_cs = 0; // RAMcs
end
```

Memórias ROM

```
module ram_decode
(
    en,
    addr,
    data
);
input en;
input [3:0] addr;
output [7:0] data;
reg [7:0] data;
initial
begin
    data = 8'b00000000;
end
always @ (addr)
begin
    if (en == 1'b0)
        begin
            case (addr)
                4'b0000: data = 8'b00000011;
                4'b0001: data = 8'b00001000;
                4'b0010: data = 8'b00001101;
                4'b0011: data = 8'b00010100;

                4'b0100: data = 8'b00011001;
                4'b0101: data = 8'b00011110;
                4'b0110: data = 8'b00100101;
                4'b0111: data = 8'b00101010;

                4'b1000: data = 8'b00101100;
                4'b1001: data = 8'b00110001;
                4'b1010: data = 8'b00110110;
                4'b1011: data = 8'b00111101;

                4'b1100: data = 8'b01000110;
                4'b1101: data = 8'b01010000;
                4'b1110: data = 8'b01011001;
                4'b1111: data = 8'b01101100;
            endcase
        end
    end
endmodule
```


Memórias ROM

```
always @ (program_cs)
begin
  if (program_cs == 1'b0)
  begin
    case (in_program_addr)
      'h0000: instruction = 'h07;
      'h0001: instruction = 'hc0;
      'h0002: instruction = 'h55;
      'h0003: instruction = 'h07;
      'h0004: instruction = 'hc1;
      'h0005: instruction = 'h66;
      'h0006: instruction = 'h07;
      'h0007: instruction = 'h06;
      'h0008: instruction = 'h00;
      'h0009: instruction = 'h0E;
      'h000A: instruction = 'h07;
      'h000B: instruction = 'h03;
      'h000C: instruction = 'h00;
      'h000D: instruction = 'h00;
      'h000E: instruction = 'h04;
      'h000F: instruction = 'h07;
      'h0010: instruction = 'h07;
    endcase
  end
end
```

Bloco de Estímulo

```
module mpp_tb ();  
  
    reg clk;  
    reg [7:0] in;  
  
    wire [7:0] out;  
    wire [4:0] out_signals;  
    wire [15:0] in_program_addr;  
  
    reg [7:0] instruction;  
    reg program_cs;  
  
    mpp m1  
    (  
        .clk          (clk),  
        .instruction  (instruction),  
        .out_signals  (out_signals),  
        .program_addr (in_program_addr),  
        .in           (in),  
        .out          (out)  
    );  
endmodule
```

Bloco de Estímulo

```
initial
begin
    clk = 1'b0;
    in = 8'b00010100;

    #6000

    $finish;
end
```

Bloco de Estímulo

```
always
begin
    #10 clk = !clk;
end

always @ (out_signals)
begin
    program_cs = ~out_signals[1];
end

always @ (program_cs)
begin
    if (program_cs == 1'b0)
        begin
            case (in_program_addr)
                'h0000: instruction = 'h07;
                'h0001: instruction = 'hc0;
                'h0002: instruction = 'h55;
                'h0003: instruction = 'h07;
                'h0004: instruction = 'hc1;
                'h0005: instruction = 'h66;
                'h0006: instruction = 'h07;
                'h0007: instruction = 'h06;
                'h0008: instruction = 'h00;
                'h0009: instruction = 'h0E;
                'h000A: instruction = 'h07;
                'h000B: instruction = 'h03;
                'h000C: instruction = 'h00;
                'h000D: instruction = 'h00;
                'h000E: instruction = 'h04;
                'h000F: instruction = 'h07;
                'h0010: instruction = 'h07;
            endcase
        end
    end
end
```

Operacionalidade da Implementação

```
Loop:  
MOV 55,A;  
MOV 66,B;  
ADD B,A;  
MOV A,B;  
MOV A,OUT1;  
CALL MOVE;  
JMP LOOP;  
  
MOVE:  
RET;
```


Comparação Correlatos

| correlatos características | AYEH et al., (2008) | PABLO et al., (2016) | ZALAVA et al., (2015) | M++ FPGA |
|-------------------------------|---------------------|----------------------|-----------------------|----------|
| fabricante | Xilinx | Xilinx | Xilinx | Altera |
| tecnologia | FPGA | FPGA | FPGA | FPGA |
| número de instruções | 4 | 29 | 29 | 14 |
| arquitetura | Própria | Harvard | Harvard | Harvard |
| set de instruções | Próprio | RISC | RISC | Próprio |
| frequência | ~95MHz | ~40MHz | Não inf. | 90MHz |

Resultados e Discussões

- 90MHz teóricos;
- 129 elementos lógicos (<1%);
- Arquitetura funciona corretamente.

Conclusões

- Objetivos atendidos;
- Importante para a área;
- Prova real da arquitetura;
- Limitações.

Extensões

- M+++;
- RISC;
- Otimização;
- Barramento de 16 *bits*;
- Melhorias no código;
- *Open Source*;
- Documentação;
- Interface externa.

Referências

- AYEH, Eric et al. **FPGA Implementation of an 8-bit Simple Processor**. Denton, [2008]. Disponível em: <<https://www.researchgate.net/publication/4342279>>. Acesso em: 4 abril 2018.
- BORGES, Jonathan M.. **CONSTRUÇÃO DE UMA UCP HIPOTÉTICA M++**. Blumenau, [2003]. Disponível em: <<http://www.inf.furb.br/~maw/mmaismais/artigos/artigo.pdf>>. Acesso em: 6 abril 2018.
- PABLO, S. de. et al. **A very simple 8-bit RISC processor for FPGA**. Valladolid, [2016]. Disponível em: <<https://www.researchgate.net/publication/267766934>>. Acesso em: 4 abril 2018.
- Zavala, Antonio H. et al. **Design of a General Purpose 8-bit RISC Processor for Computer Architecture Learning**. Querétaro, [2015]. Disponível em: <<http://www.scielo.org.mx/pdf/cys/v19n2/v19n2a13.pdf>>. Acesso em: 4 abril 2018.

Obrigado!