

UNIVERSIDADE FEDERAL DO SUL E SUDESTE DO PARÁ
INSTITUTO DE GEOCICENCIAS E ENGENHARIAS
FACULDADE DE COMPUTAÇÃO E ENGENHARIA ELÉTRICA
BACHAREL EM ENGENHARIA DA COMPUTAÇÃO

SISTEMAS EMBARCADOS

“NANO CONTROLADOR”

WARLEY RABELO GALVÃO - 201840601031
IAGO COSTA DAS FLORES – 201840601017
LEYRISVAN DA COSTA NASCIMENTO - 201740601025
GABRIEL OLIVEIRA MACHADO - 201540601037

MARABÁ/PA
2021

WARLEY RABELO GALVÃO
LEYRISVAN DA COSTA NASCIMENTO
IAGO COSTA DAS FLORES
GABRIEL OLIVEIRA MACHADO

SISTEMAS EMBARCADOS

“NANO CONTROLADOR”

Relatório do Projeto Final de um Nano
Controlador pra obtenção parcial de nota
da disciplina SISTEMAS
EMBARCADOS. Ministrado pelo
Professor: **Dr. José Carlos da Silva.**

MARABÁ/PA
2021

Sumário

1. INTRODUÇÃO	4
2. DESENVOLVIMENTO	4
2.1. BIBLIOTECAS E ENTIDADES.....	4
2.2. ARQUITETURA E ULA.....	5
2.3. ARQUITETURA E REGISTRADORES.....	6
2.4. SIMULAÇÃO	9
3. CONCLUSÃO.....	11
4. REFERÊNCIAS BIBLIOGRÁFICAS	11

1. INTRODUÇÃO

Objetivo desse relatório é apresentar os resultados obtidos na implementação e simulação de um “Nano Controlador” em linguagem VHDL, no qual possui entradas e saída de dados e um registrador para escolha da operação.

O Controlador deve receber dois valores de entrada e retornar uma saída com o resultado da operação utilizada. A operação utilizada na execução vai respeitar o comando de um registrador específico para essa função. As operações seriam as portas lógicas básicas: not, and, or, nor, nand e as 4 operações matemáticas: inversor subtração, inversor soma, subtração e soma.

A partir do valor armazenado no registrador de comando uma das operações matemática ou lógica será executada na simulação do “Nano Controlador”.

2. DESENVOLVIMENTO

Nosso algoritmo foi desenvolvido conforme as instruções do trabalho final, utilizando o software quartus II web edition versão gratuita do quartus II, software desenvolvido pela intel para criação de modelos de hardware.

2.1. BIBLIOTECAS E ENTIDADES

Para nosso exemplo temos implementados em toda a descrição VHDL a declaração de bibliotecas como a “library IEEE” e a “IEEE.Std_Logic_1164.all” a declaração destas indicam que o circuito está usando um tipo de modelagem lógica além de um e zero. Quando falamos de std_logic temos 9 níveis.

No início do teste, temos um novo componente, “IEEE.Std_Logic_unsigned.all” que funciona como um operador aritmético, entre vetores, isso na prática tem mais no interior do pacote.

Além disso, toda descrição do VHDL é um conjunto, ou um par de entidade-arquitetura, a entidade declara a pinagem externa do circuito, e a arquitetura detalha a implementação, portanto temos os sinais de entradas detalhados. Tendo um barramento de entrada de 3 bits que destacam a operação de leitura, e 5 sinais de saída, todos conectados posteriormente ao CE dos registradores.

2.2. ARQUITETURA E ULA

Na figura 01 temos o código de implementação da unidade de lógica e aritmética do nosso “NanoControlador”. Com isso pode-se observar na linha 40 a configuração de um processo que recebe três registradores: “ula_op” responsável por indicar qual a operação deve ser realizada e “A_int” e “B_int” correspondente aos dois valores de entrada.

A partir do valor contido em ula_op será realizado uma operação ou lógica ou aritmética com os dados entradas recebidos.

```

40  ULA: PROCESS(ula_op, A_int, B_int)                -- início do processo da ula
41  BEGIN
42      CASE ula_op IS                                -- opcoes de configuracao da ula
43          WHEN "0000" => C_int <= A_int + B_int;    -- soma
44          WHEN "0001" => C_int <= A_int - B_int;    -- subtrai
45          WHEN "0010" => C_int <= NOT A_int + B_int; -- inversor soma
46          WHEN "0011" => C_int <= NOT A_int - B_int; -- inversor subtracao
47
48          WHEN "0100" => C_int <= not A_int;        -- not A
49          WHEN "0101" => C_int <= not B_int;        -- not B
50          WHEN "0110" => C_int <= A_int or B_int;   -- ou lógico
51          WHEN "0111" => C_int <= A_int and B_int;  -- e lógico
52          WHEN "1000" => C_int <= A_int;            -- passa A
53          WHEN "1001" => C_int <= B_int;            -- passa B
54          WHEN "1010" => C_int <= A_int nor B_int;  -- not or lógico
55          WHEN "1011" => C_int <= A_int nand B_int; -- not e lógico
56          WHEN OTHERS => C_int <= A_int;            -- default: passa A;
57      END CASE;
58  END PROCESS ULA;

```

Figura 1: Temos a implementação da ULA

2.3. ARQUITETURA E REGISTRADORES

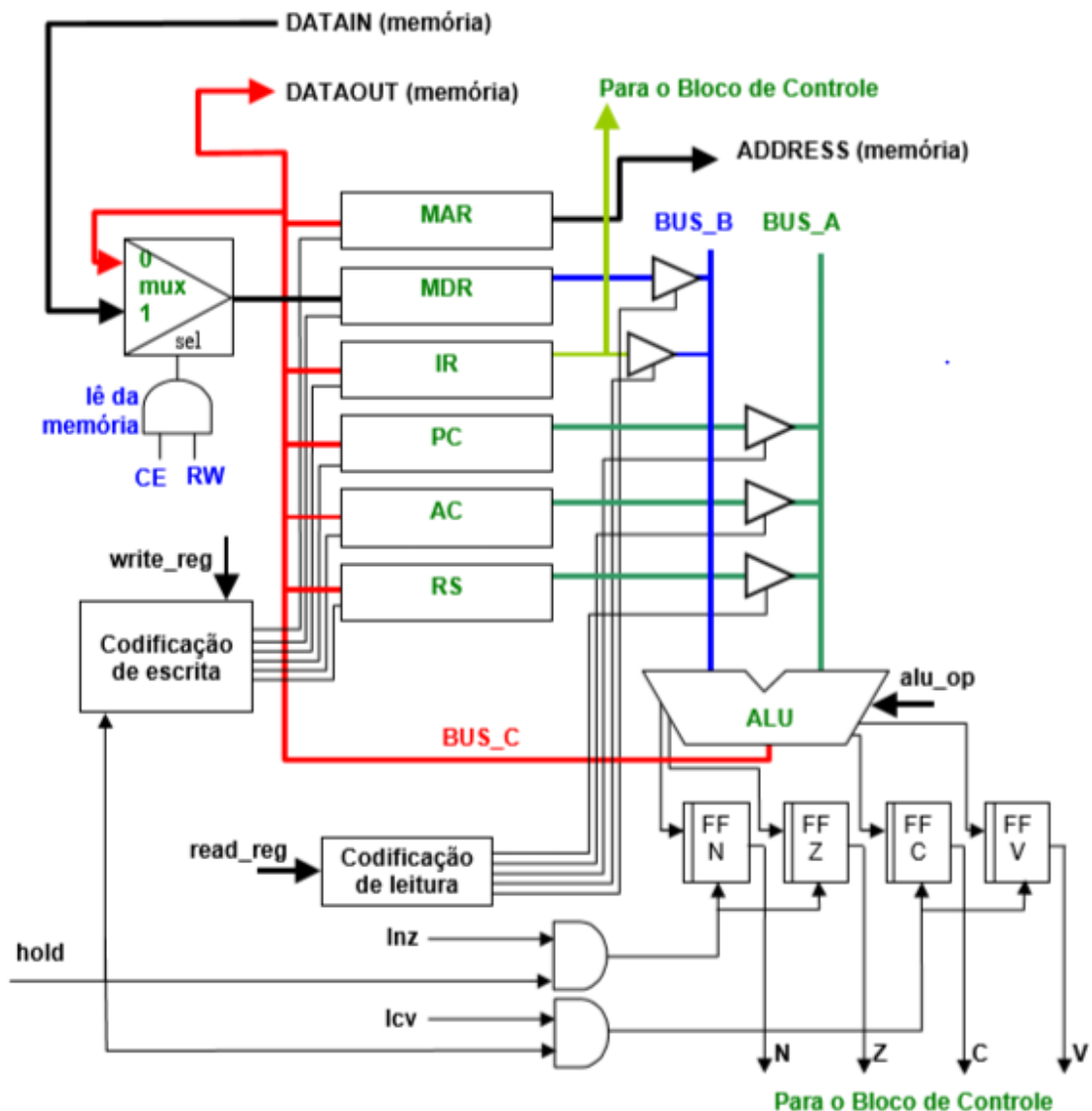


Figura 01: Registradores MAR, MDR, IR, PC, AC, RS

A imagem acima foi colocada pra melhor representar os registradores e a finalidade de suas funções dentro do projeto. Afinal é muito importante fazer o projeto passo a passo pra que se obtenha os resultados satisfatório com êxito.

Nesse diagrama temos 6 registradores que tem a estrutura definida como um bloco de dados que através de uma codificação em binário será possível fazer a leitura e a escrita em que o projeto segue os padrões pra uma possível simulação usando a linguagem Assembly.

Olhando o desenho do bloco de dados temos que a micro-instrução proveniente do bloco de controle é formada pelo conjunto:

micro-instrução = { write_reg, read_reg, ALU_op, ce, rw, lnz, lcv }

A semântica destes sinais é:

Write_reg: indica qual do(s) registradore(s) será(ão) escrito(s).

Denominação	Codificação	Descrição
MAR	000	Escreve no registrador MAR.
MDR	001	Escreve no registrador MDR.
IR	010	Escreve no registrador IR.
PC	011	Escreve no registrador PC.
AC	100	Escreve no registrador AC.
RS	101	Escreve no registrador RS.
PC_MDR	110	Escreve simultaneamente nos registradores MDR e PC.
NULL	111	Não escreve em nenhum registrador.

Read_reg: indica qual do(s) registradore(s) será(ão) lido(s).

Denominação	Codificação	Descrição
NULL	000	Não coloca nada em BUS_A nem em BUS_B.
MDR	001	Lê o registrador MDR para BUS_B. NADA em BUS_A
IR	010	Lê o registrador IR para BUS_B. NADA em BUS_A
PC	011	Lê o registrador PC para BUS_A. NADA em BUS_B
AC	100	Lê o registrador AC para BUS_A. NADA em BUS_B
RS	101	Lê o registrador RS para BUS_A. NADA em BUS_B
AC_MDR	110	Lê o registrador AC para BUS_A. Lê o registrador MDR para BUS_B.
PC_MDR	111	Lê o registrador PC para BUS_A. Lê o registrador MDR para BUS_B.

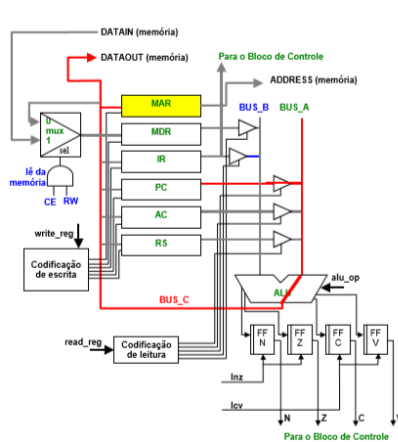
ALU_op: operação da Unidade Lógica Aritmética

Operação	Codificação	Descrição
Soma	000	$BUS_C \leftarrow BUS_A + BUS_B$ Gera N, Z, C e V.
Inc	001	$BUS_C \leftarrow BUS_A + 1$ Gera N, Z, C e V.
Não	010	$BUS_C \leftarrow \text{not } BUS_A$ Gera N, Z
PassaB	100	$BUS_C \leftarrow BUS_B$ Gera N, Z
Ou	101	$BUS_C \leftarrow BUS_A \text{ or } BUS_B$ Gera N, Z
E	110	$BUS_C \leftarrow BUS_A \text{ and } BUS_B$ Gera N, Z
PassaA	111	$BUS_C \leftarrow BUS_A$ Gera N, Z

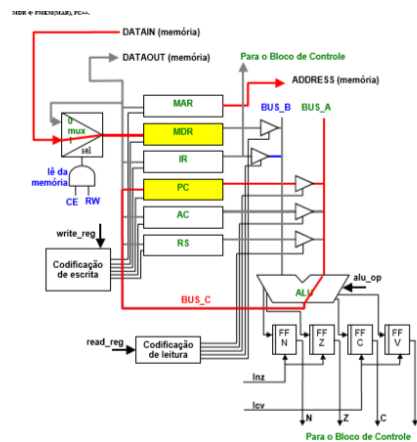
we, rw: controlam a operação com a memória.

lnz: comanda a carga dos flags N (negativo) e Z (zero). Em função da instrução assembly é feita esta carga. Por exemplo, ver o OR, que só carrega N/Z.

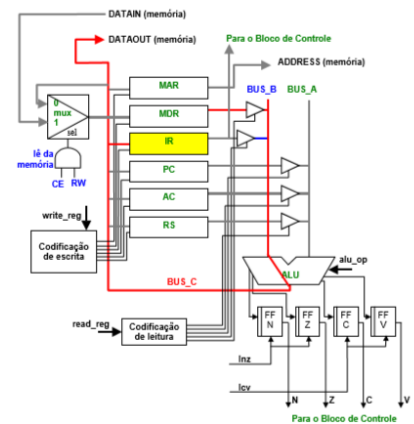
lev: comanda a carga dos flags C (carry) e V (overflow). Em função da instrução assembly é feita esta carga. Por exemplo, ver a soma, que só carrega os quatro flags.



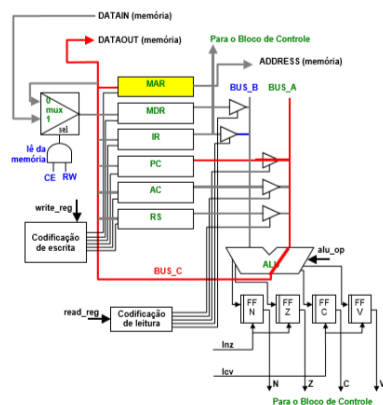
CICLO 1



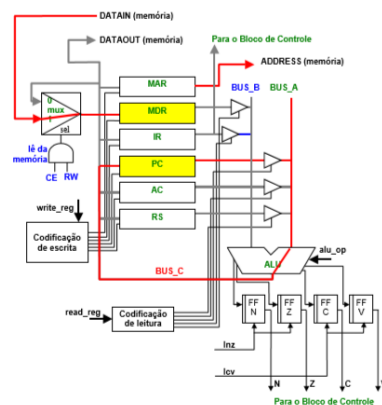
CICLO 2



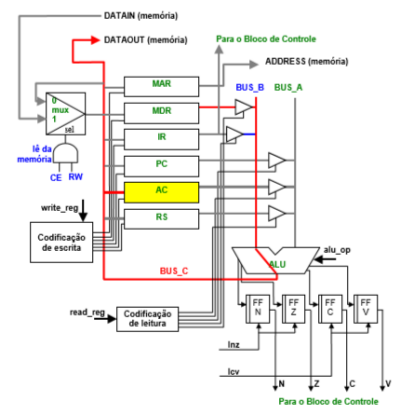
CICLO 3



CICLO 4



CICLO 5



CICLO 6

- **Ciclo 01: MAR ← PC.** Carrega o conteúdo do PC para o registrador MAR. [Lê do PC], [ALU opera em PASSA_A], e [grava no MAR]
- **Ciclo 02: MDR ← PMEM(MAR), PC++.** Uma operação em que tem dois caminhos paralelos, e que não há problema de ler e escrever o PC no mesmo ciclo (flip-flops mestre /escravo).
- **Ciclo 3: IR ← MDR.** Terminou o ciclo de busca. O IR está carregado. Agora o bloco de controle decodifica e dispara as instruções de execução.
- **Ciclo 4: MAR ← PC.** Temos o primeiro ciclo de execução
- **Ciclo 5: MDR ← PMEM(MAR), PC++.** Busca do operando na memória.

- **Ciclo 6: $AC \leftarrow MDR$.** Terminou a execução do LDA modo imediato de endereçamento.

2.4. SIMULAÇÃO

Para que fosse realizada a simulação do código fonte em VHDL primeiramente foi definido algumas configurações iniciais que são essenciais para o devido funcionamento das operações aritméticas e lógicas. As entradas A e B foram inicializadas com o valor 12 e 24 respectivamente para que seja possível efetuar uma operação matemática utilizando os registradores.

Na variável C é inicializada com zero, pois é onde vai ser mostrado a nossa saída assim como foi definido no código fonte em VHDL. Na variável address encontra-se os endereços em que serão salvos os resultados na memória. Vale destacar que a variável enable é ativada somente quando ocorrer um pulso de clock de nível baixo para alto, ou seja, de 0 para 1 e por fim na variável ula_op é onde ocorre a definição de qual operação matemática será realizada de acordo com o que foi definido no código fonte em VHDL.

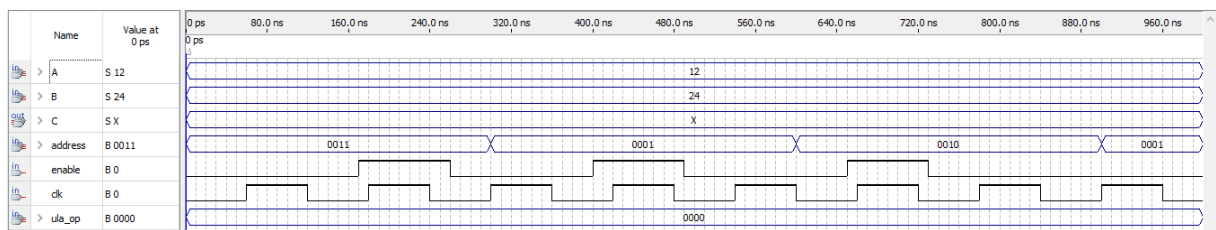


Figura 02: Representação de entradas, saídas, endereços, clock e ula na simulação

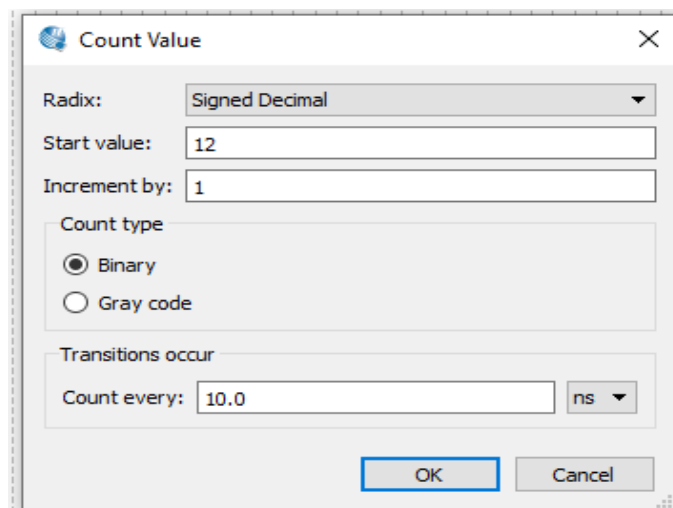
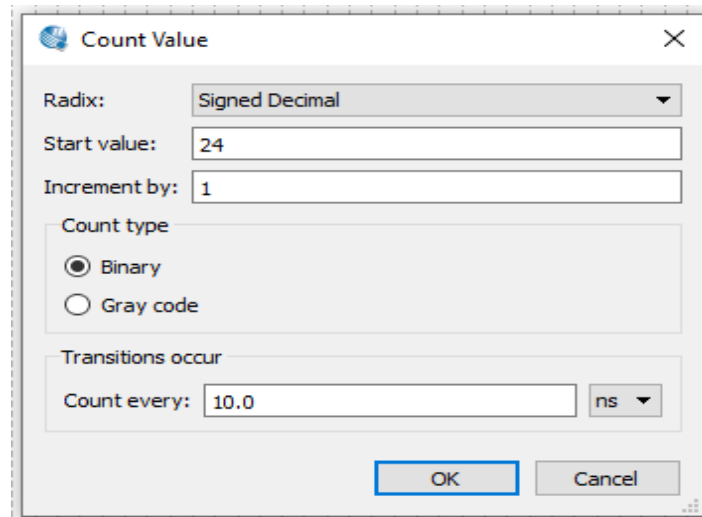


Figura 3: Entrada A inicializada com valor 12



Count Value

Radix: Signed Decimal

Start value: 24

Increment by: 1

Count type

☒ Binary

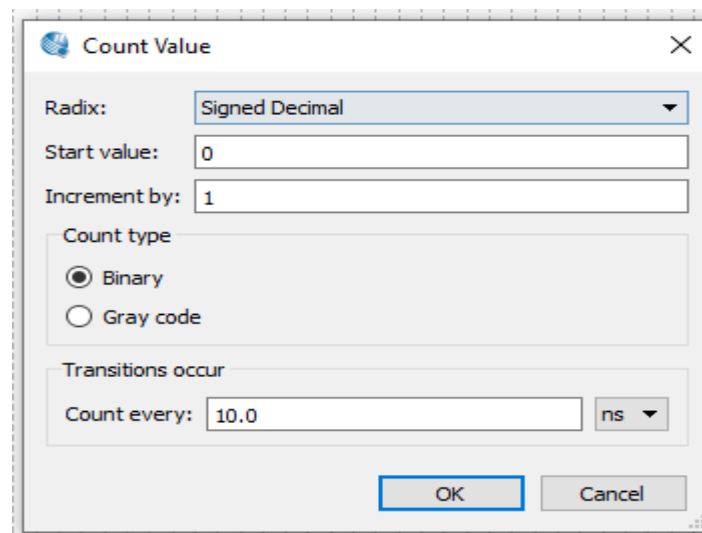
☐ Gray code

Transitions occur

Count every: 10.0 ns

OK Cancel

Figura 4: Entrada B inicializada com o valor 24



Count Value

Radix: Signed Decimal

Start value: 0

Increment by: 1

Count type

☒ Binary

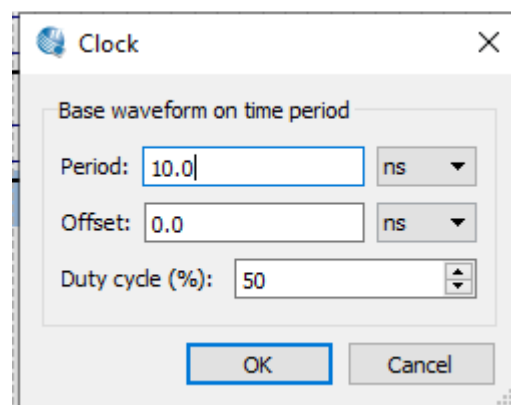
☐ Gray code

Transitions occur

Count every: 10.0 ns

OK Cancel

Figura 5: Saída C com valor nulo



Clock

Base waveform on time period

Period: 10.0 ns

Offset: 0.0 ns

Duty cycle (%): 50

OK Cancel

Figura 6: Porta Enable

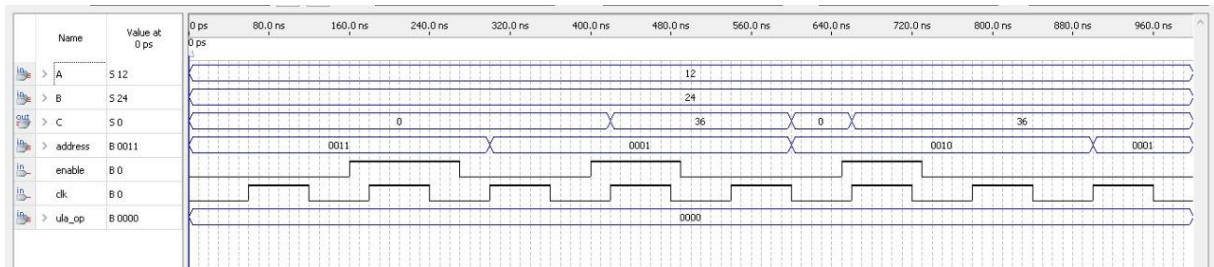


Figura 7: Resultado final após simulação

Após a execução da simulação verifica-se na figura 7 que foi utilizada o código “0000” na ula_op, dessa forma definindo que será efetuada a operação matemática de soma e logo após a simulação fica evidente que foi possível alcançar o resultado esperado, pois a soma está sendo efetuada de forma correta $12 + 24 = 36$ sendo apresentado esse resultado na saída C e salvando o resultado em memória nos seus respectivos endereços.

3. CONCLUSÃO

Devido as implementações feitas em conjunto de todos os membros da equipe, a projeção de um programa responsável pela demonstração de blocos de dados em VHDL para a criação de um nano controlador, com base em todos os resultados obtidos e apresentados, os estudos praticados pelos discentes foi possível compreender melhor o funcionamento e criação de trabalho mais complexos que são capazes de exercer operações aritméticas.

Portanto analisando os casos apresentados foi possível validar a eficiência, levando em consideração o sucesso em implementar e executar a tarefa proposta pelos discentes, tanto como projeto com os dados obtidos e apresentados, resultando em incrementação do projeto e expansão da compreensão do conteúdo previamente apresentado. Portanto é nítido a eficiência do nano controlador desenvolvido.

4. REFERÊNCIAS BIBLIOGRÁFICAS

MARWEDEL, Peter. **Embedded Systems**. Boston: Kluwer Academic Publishers, 2006