Aluno: Miller Raycell Monteiro Correia

Matricula: 2017009560

Disciplina: Arquitetura e Organização de Computadores

DESCRIÇÃO DOS COMPONENTES

COMPONENTE 1: REGISTRADOR D E JK

I) Registradores são conhecidos como as ferramentas para armazenar os valores que são trabalhos em baixo nível, a sua função se relaciona com o hardware guardando os valores que o programa irá usar, o registrador do tipo D, trabalha com uma entrada e o clock para armazenar dados, já o JK é um melhoramento que permite duas entradas e o clock.

No registrador do tipo D é utilizada uma entrada (pin) e o clock que são standard logical, e um pino de saída pout, e será armazenado um valor toda vez que o clock for 1.

Código:

```
Library ieee;
use ieee.std_logic_1164.all;

ENTITY registerd is
PORT(
        clk, pin : IN STD_LOGIC;
        pout : OUT STD_LOGIC
);
END registerd;
```

```
ARCHITECTURE behavior OF registerd IS
BEGIN
        pout <= pin WHEN clk = '1';
END behavior;
```

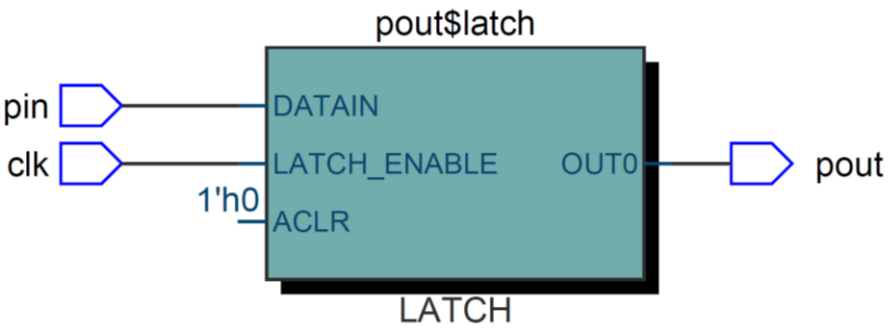
Já o registrador JK usa como entrada os valores J, K e o clock que são standard logical, e tem como saída Q e ~Q, que serão os valores que serão processados dependendo do clock.

Código:

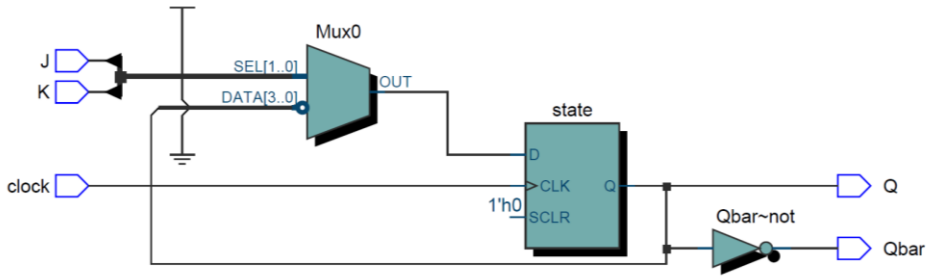
```
library ieee;
use ieee.std_logic_1164.all;

entity registerjk is
        port ( clock: in std_logic;
        J, K: in std_logic;
        Q, Qbar: out std_logic );
end registerjk;

architecture behavior of registerjk is

  signal state: std_logic;
  signal input: std_logic_vector(1 downto 0);
```

```
begin

  input <= J & K;

  p: process(clock) is
  begin
      if (rising_edge(clock)) then
          case (input) is
              when "11" => state <= not (state);
              when "10" => state <= '1';
              when "01" => state <= '0';
              when others => null;
          end case;
      end if;

  end process;
  Q <= state;
  Qbar <= not state;

end behavior;
```

II) RTL View Registrador D

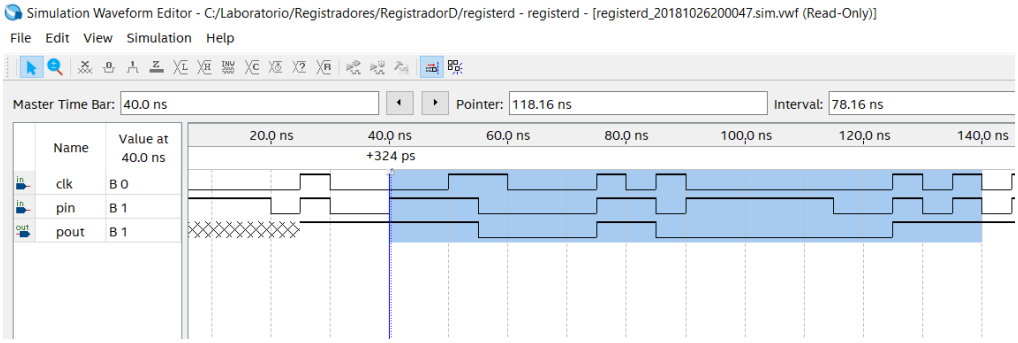


RTL View Registrador JK



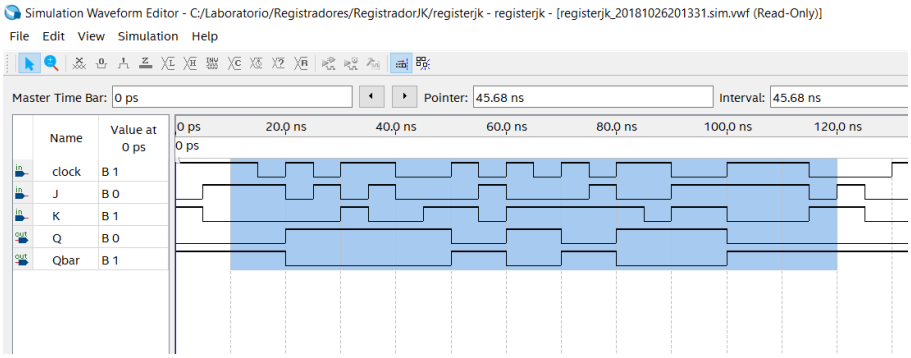
III) Waveform do registrador do tipo D

Quando o clock e o pin estiverem em high ele salva a informação e gera um pout.



Waveform do registrador do tipo JK

Quando j,k e o clock estiverem em high, logo será gerado uma saida Q e uma ~Q.



IV)

Registrador tipo D

Teste1: pin = 0, clock = 0, pout = 0;

Teste2: pin = 1, clock = 0, pout = 0;

Teste3: pin = 0, clock = 1, pout = 0;

Teste4: pin = 1, clock = 1, pout = 1;

Registrador tipo JK

Teste1: j = 0, k =0, clock = 0, q = 0, ~q = 1;

Teste2: j = 1, k =0, clock = 1, q = 0, ~q = 1;

Teste3: j = 1, k =0, clock = 1, q = 1, ~q = 0;

Teste4: j = 0, k =1, clock = 01 q = 0, ~q = 1;

COMPONENTE 2: MULTIPLEXADOR

I) Multiplexador é um componente que irá auxiliar a filtrar entradas, quando se tem várias entradas e só se precisa de uma especifica se usa o multiplexador, o multiplexador criado é de 4 entradas e uma saida.

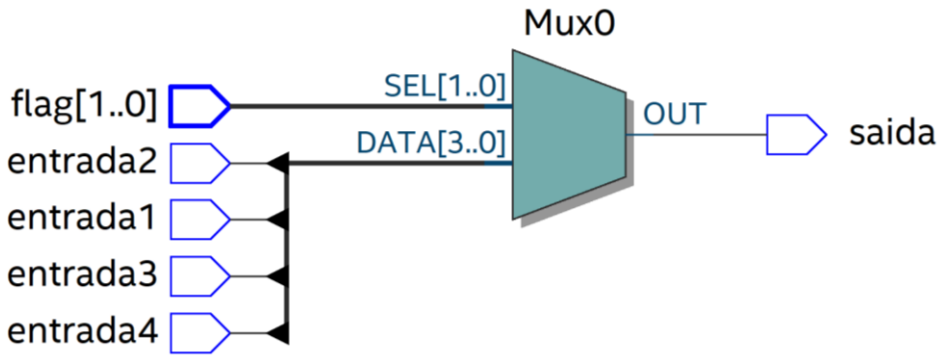
Código:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;


ENTITY multiplex IS
PORT
	( entrada1,entrada2,entrada3,entrada4 : in STD_LOGIC;
		flag : in BIT_VECTOR (1 DOWNTO 0);
		saida : out STD_LOGIC);
END multiplex;


ARCHITECTURE behavior OF multiplex IS
BEGIN
abc: PROCESS (entrada1,entrada2,entrada3,entrada4,flag)
begin
	CASE flag IS
		WHEN "00" =>  saida <= entrada1;
		WHEN "01" =>  saida <= entrada2;
		WHEN "10" =>  saida <= entrada3;
		WHEN "11" =>  saida <= entrada4;
	END CASE;
end PROCESS abc;
END behavior;
```

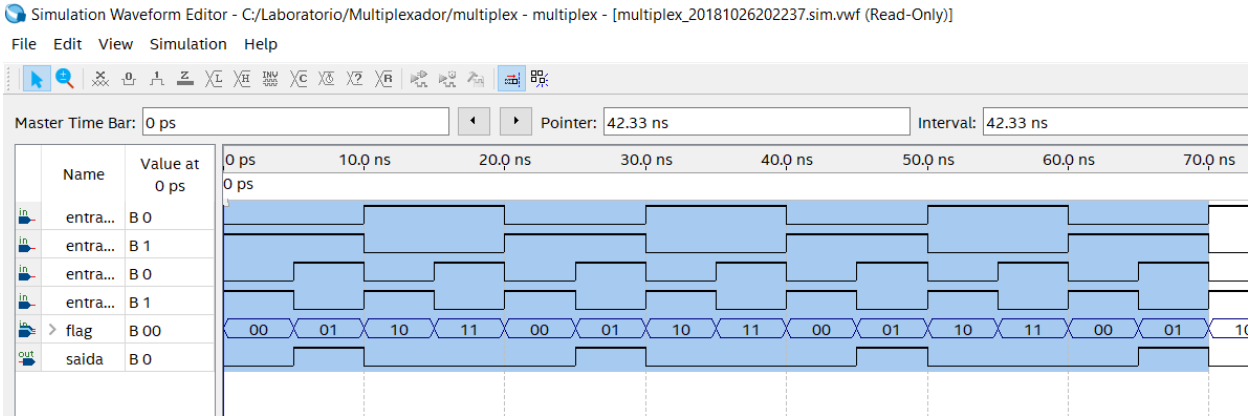
O código recebe 4 entradas e de acordo com a flag ele irá indicar a saida, se a flag = 00 a saida será a primeira entrada, flag = 01 a saida será a segunda

entrada, flag = 10 a saida será a terceira entrada, flag = 11 a saida será a quarta entrada. Para isso se usa o case when.

II) RTL View



III) Waveform



IV)

Teste 1: entrada1 = 1, entrada2 = 0, entrada3 = 0, entrada4 = 0, flag = 00 e saida = entrada1 = 0;

Teste 2: entrada1 = 1, entrada2 = 0, entrada3 = 0, entrada4 = 0, flag = 01 e saida = entrada2 = 0;

Teste 3: entrada1 = 1, entrada2 = 0, entrada3 = 1, entrada4 = 0, flag = 10 e saida = entrada3 = 1;

Teste 4: entrada1 = 1, entrada2 = 0, entrada3 = 1, entrada4 = 0, flag = 10 e saida = entrada4 = 0;

COMPONENTE 3: PORTA XOR

I) A porta lógica xor é uma porta para calculos lógicos, ela é constituida de duas portas not, duas and, uma or, que se traduzindo ficaria que a xor b é equivalente a ((not a).b) + ((not b).a), porém foi implementado com port map.

Código:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY QAnd is
port
	( a,b : in STD_LOGIC;
		s : out STD_LOGIC);
end QAnd;

ARCHITECTURE behavior of QAnd is
begin

	s <= a and b;
end behavior;
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY QNot is
port
```

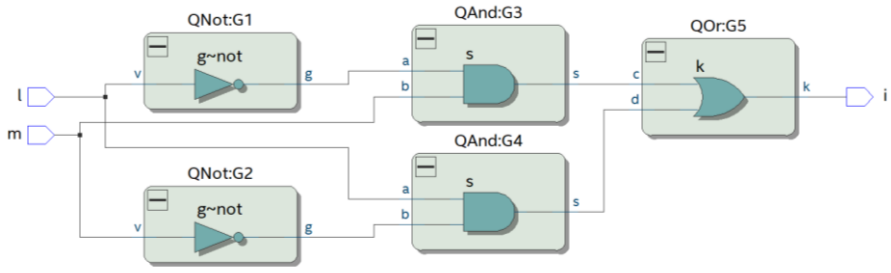
```
        ( v : in STD_LOGIC;
                g : out STD_LOGIC);
END QNot;

ARCHITECTURE behavior of QNot is
BEGIN
        g <= not v;

end behavior;
LIBRARY ieee;
use ieee.std_logic_1164.all;

ENTITY QOr is
port
        ( c,d : in STD_LOGIC;
                k : out STD_LOGIC);
END QOr;

ARCHITECTURE behavior OF QOr IS
BEGIN

        k <= c or d;
END behavior;
LIBRARY ieee;
```

```
USE ieee.std_logic_1164.all;

Entity QXor is
port
        (l,m : in STD_LOGIC;
                i : out STD_LOGIC);
end QXor;

ARCHITECTURE behavior of QXor is

component QAnd is
port
        ( a,b : in STD_LOGIC;
                s : out STD_LOGIC);
end component;

component QOr is
port
        ( c,d : in STD_LOGIC;
                k : out STD_LOGIC);
end component;

component QNot is
port
```

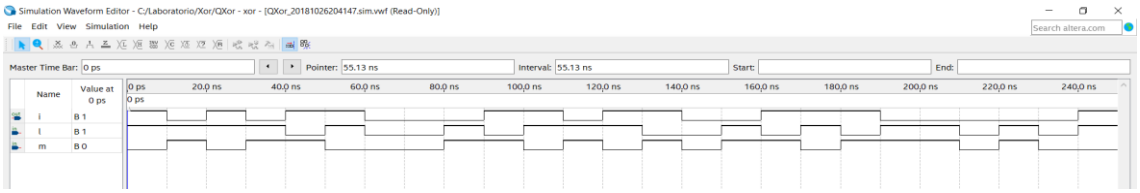
```
        ( v : in STD_LOGIC;
                g : out STD_LOGIC);
end component;

signal x1,x2,x3,x4 : STD_LOGIC;

begin

        G1: QNot port map(l,x1);
        G2: QNot port map(m,x2);
        G3: QAnd port map (x1,m,x3);
        G4: QAnd port map (l,x2,x4);
        G5: QOr port map (x3,x4,i);
end behavior;
```

O código seguira da seguinte forma primeiro se fará as negações, os ands com os negados e as portas, depois o or com os restantes e dessa forma se terá o equivalente lógico do xor, sendo l e m as entradas e i a saida.

II) RTL View



III) Waveform



IV)

Teste 1: l = 0, m = 0, I = 0;

Teste 2: l = 1, m = 0, I = 1;

Teste 3: l = 0, m = 1, I = 1;

Teste 4: l = 1, m = 1, I = 0;

COMPONENTE 4: SOMADOR MAIS 4

I) Um somador é um componente que irá pegar um valor de 16 bits, e somará mais 4, esse somador clona a ideia do pc do processador, que irá buscar as próximas instruções a serem realizadas.

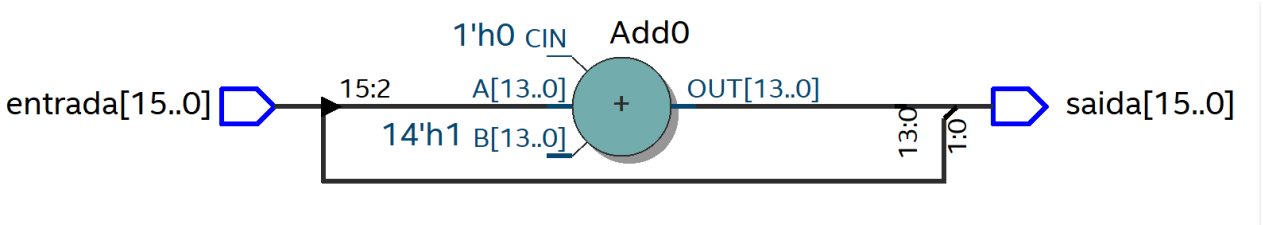
Código:

```
LIBRARY ieee;

USE ieee.std_logic_1164.all;

use ieee.std_logic_signed.all;

use ieee.std_logic_arith.all;

ENTITY somador IS

PORT

        (entrada : in STD_LOGIC_VECTOR (15 DOWNTO 0);

        saida: out STD_LOGIC_VECTOR (15 DOWNTO 0));
```

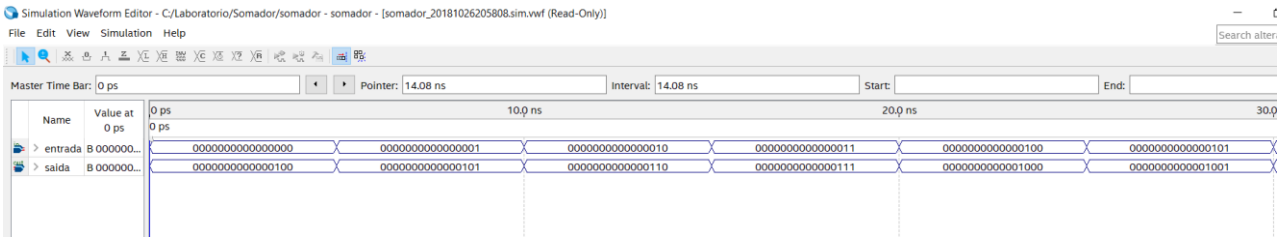
```
END somador;


ARCHITECTURE behavior OF somador IS
BEGIN
        saida <= entrada + "0000000000000100";
END behavior;
```

O código recebe uma entrada e soma 4 em binário com 16 bits, resultando em uma saida com 16 bits, usa a biblioteca artih, que permite fazer somas.

II) RTL View



III) Waveform



IV)

Teste 1: entrada: 0000000000000000; saida: 0000000000000100;

Teste 2: entrada: 0000000000000001; saida: 0000000000000101;

Teste 3: entrada: 0000010000000000; saida: 0000010000000100;

Teste 4: entrada: 0000000000010000; saida: 0000000000010100;

Teste 5: entrada: 0000000000000111; saida: 0000000000001011;

Teste 6: entrada: 0000000000010000; saida: 0000000000010100;

COMPONENTE 6: MEMORIA RAM

I) A memória Ram é um hardware de armazenamento assim como a memória ROM, porém a sua utilidade se espande para leitura e escrita de dados, e ela é volátil, quando tem energia ela armazena, uma vez que a energia cessa, ela elimina os processos.

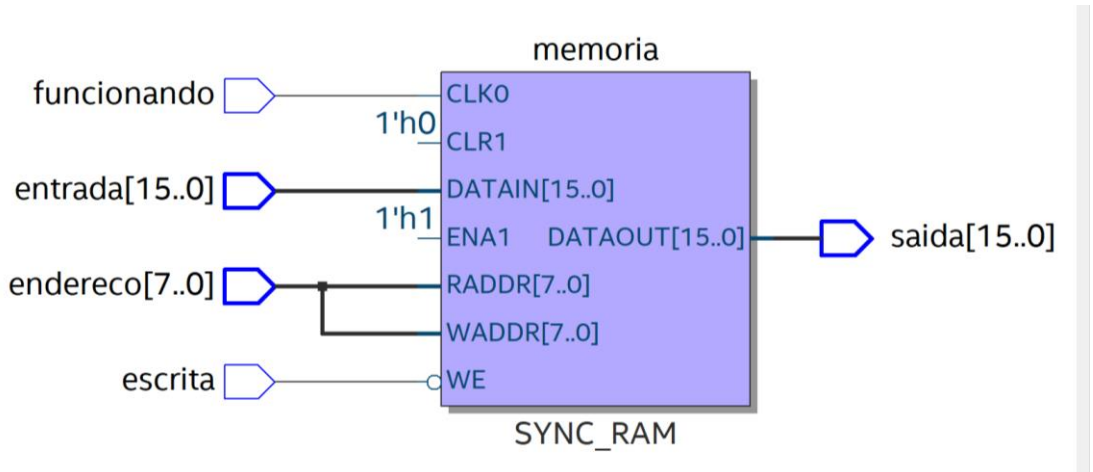
Código:

```
library ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY memram IS
PORT
(
        entrada : in UNSIGNED (15 DOWNTO 0);
        saida : out UNSIGNED (15 DOWNTO 0);
        endereco : in UNSIGNED (7 DOWNTO 0);
        escrita,funcionando : in std_logic);
end memram;

ARCHITECTURE behavior OF memram IS
```

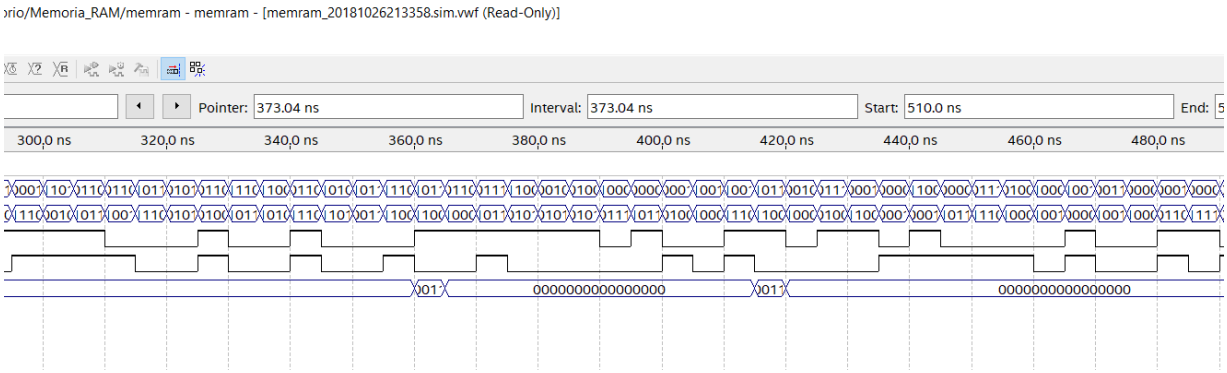
```
        TYPE arranjo is array (0 to 65535) OF UNSIGNED (15 DOWNTO 0);
        SIGNAL memoria:arranjo;
BEGIN
        process(funcionando,endereco)
        BEGIN
                IF rising_edge(funcionando) THEN
                        IF escrita = '0' THEN memoria(to_integer(endereco)) <=
entrada;
                                END IF;
                        END IF;
                END PROCESS;
        saida <= memoria(to_integer(endereco));
end behavior;
```

A memória RAM descrita possui 655356 espaços de memória por ser de 16 bits, quando ela for ativada para leitura e escrita a saida será alterada, pois quando se escrever não vai ter saida.

II) RTL View



III) Waveform



Como na waveform representa só vai ter saida quando se for fazer leitura de arquivos.

IV)

Teste1: entrada:0000000000000000, leitura = 1, escrita = 0, saida = 00000111000000000

COMPONENTE 7: BANCO DE REGISTRADORES

I) O banco de registradores recebe 3 endereços indicando o endereço dos 3 registradores que serão usado, e as portas para isso são INSTR2,INSTR3 E INSTRD que irão ser esses endereços. a WRITEFLAG será a flag ativada pela unidade de controle para dizer se é preciso escrever no registrador de destino ou não, WRITEBACKDATA será os dados que vem da memória de dados para escrever no registrador de destino e OUTREG1 E OUTREG2 serão os valores dos registradores que sairão para a unidade de controle que serão identificados pelo INSTR2 e INSTR3. Seguindo a ordem do código primeiro temos o BANK que será onde estará os registradores e o Signal BANK que será usado para acessar o BANK (uma ‘instaciação’) os valores OUTREG1 e OUTREG2 recebem o os valores respectivos do INSTR2 e INSTR3 acessando o vetor do BANKREG com esses endereços, depois é feito uma análise se a flag WRITEFLAG está ativada para caso esteja ativada o dado que veio no

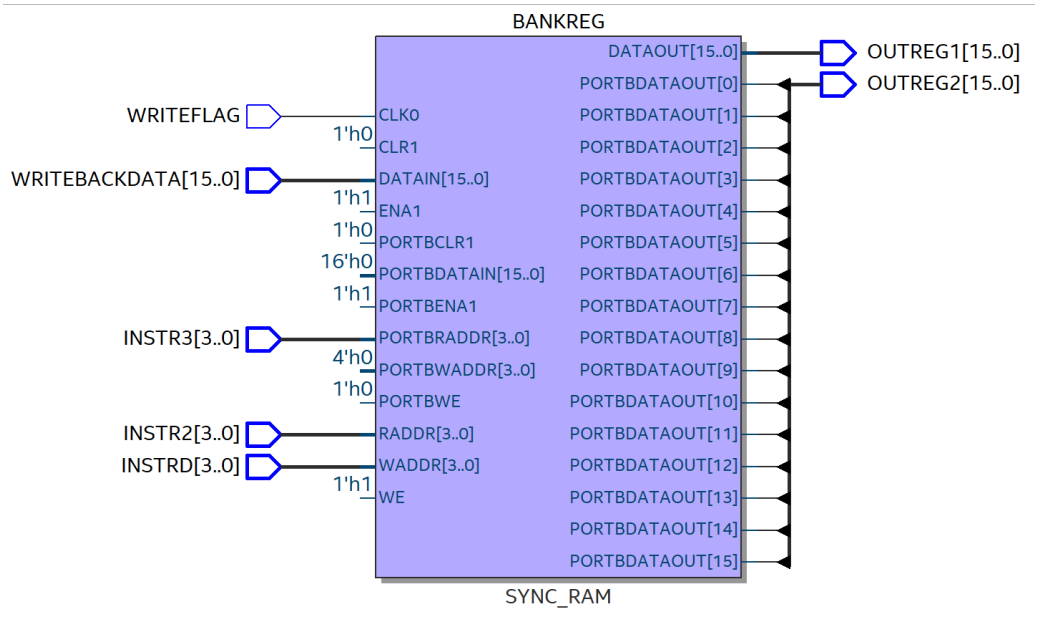
WRITEBACK é mandado para onde está o INSTRD no vetor BANK salvando assim o valor nele.

Código:

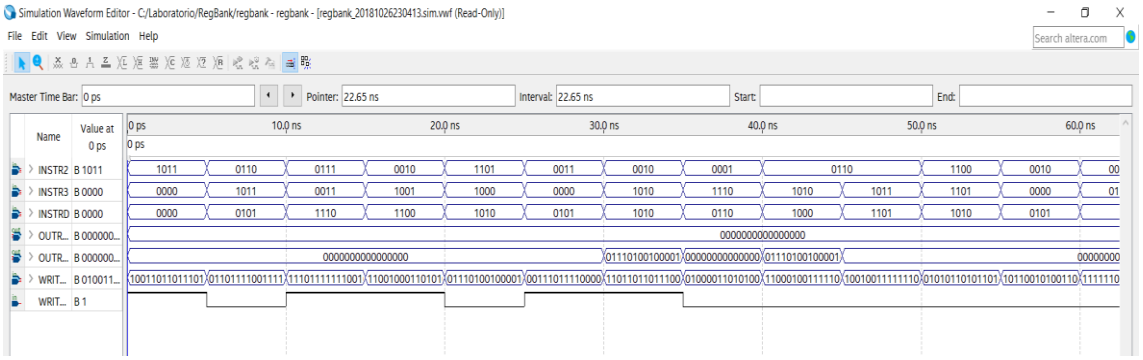
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

ENTITY regbank IS
	PORT(
		INSTR2,INSTR3, INSTRD : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
		WRITEFLAG: IN STD_LOGIC;
		WRITEBACKDATA: IN STD_LOGIC_VECTOR(15 DOWNTO 0);
		OUTREG1,OUTREG2 : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
	);
END regbank;

ARCHITECTURE behavior OF regbank IS

	TYPE BANK IS ARRAY(0 TO 15) OF STD_LOGIC_VECTOR(15 DOWNTO 0);
	SIGNAL BANKREG : BANK;
```

```
BEGIN

        OUTREG1 <= BANKREG(
TO_INTEGER(UNSIGNED(INSTR2)));

        OUTREG2 <= BANKREG(
TO_INTEGER(UNSIGNED(INSTR3)));

        PROCESS(WRITEFLAG)

            BEGIN

                IF WRITEFLAG = '1' THEN

                    BANKREG(TO_INTEGER(
UNSIGNED(INSTRD))) <= WRITEBACKDATA;

                END IF;

        END PROCESS;


END behavior;
```

II) RTL View



III) Waveform



IV) Como podemos ver na WaveForm o retorno dos dois registradores é 000000000 isso acontece porque ainda não possuem valores dentro dos registradores (o que é certo pois ainda não foi setado nenhum valor para eles e o retorno tem que ser realmente ‘vazio’)**.**

COMPONENTE 8: SOMADOR DE 16 BITS

I) Este é um componente é muito importante, pois é uma das operações básicas que se espera que um computador faça, somar dois valores é fundamental par ao funcionamento de qualquer computador. Este componente faz parte da ULA.

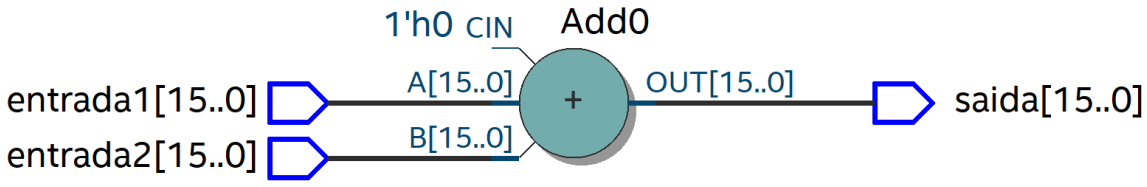
Código:

```
LIBRARY ieee;

USE ieee.std_logic_1164.all;

use ieee.std_logic_signed.all;

use ieee.std_logic_arith.all;


ENTITY sum IS

PORT
```

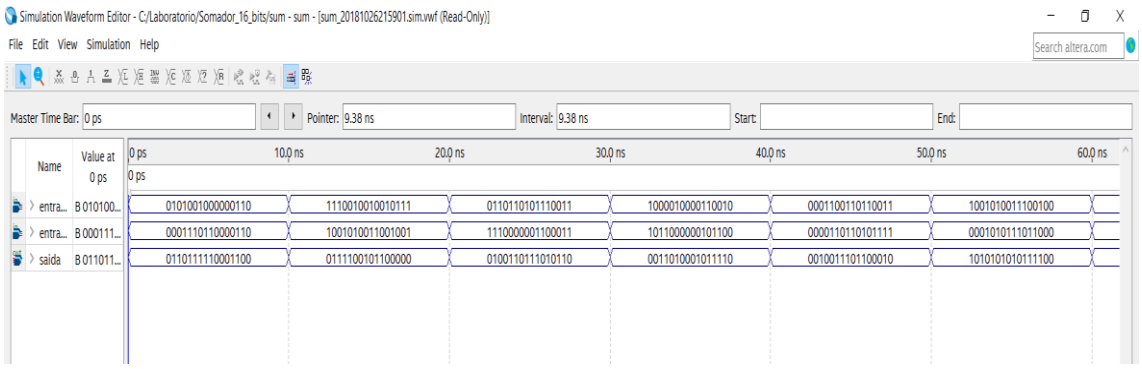
```
(
        entrada1,entrada2: in STD_LOGIC_VECTOR(15 DOWNTO 0);
        saida : out STD_LOGIC_VECTOR(15 DOWNTO 0));
END sum;

ARCHITECTURE behavior OF sum IS
BEGIN
        saida <= entrada1 + entrada2;
END behavior;
```

O código vai pegar dois valores de 16 bits e vai realizar a soma dos dois.

II) RTL View



III) Waveform



IV)

Teste 1: entrada1: 0101001000000110, entrada2: 0001110110000110, saida: 0110111110001100.

COMPONENTE 9: UNIDADE DE CONTROLE

I) A unidade de controle é o cerebro do computador, é o componente que vai guiar os outros componentes, ela funciona como um gestor das atividadaes, para cada tipo de operação.

Código:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY uc IS
PORT
(
        entrada : in std_logic_vector (4 DOWNTO 0);
        regdest : out std_logic;
        origalu : out std_logic;
        mempararegs : out std_logic;
        escrevereg : out std_logic;
        lemem : out std_logic;
        escrevemem : out std_logic;
        branch : out std_logic;
```

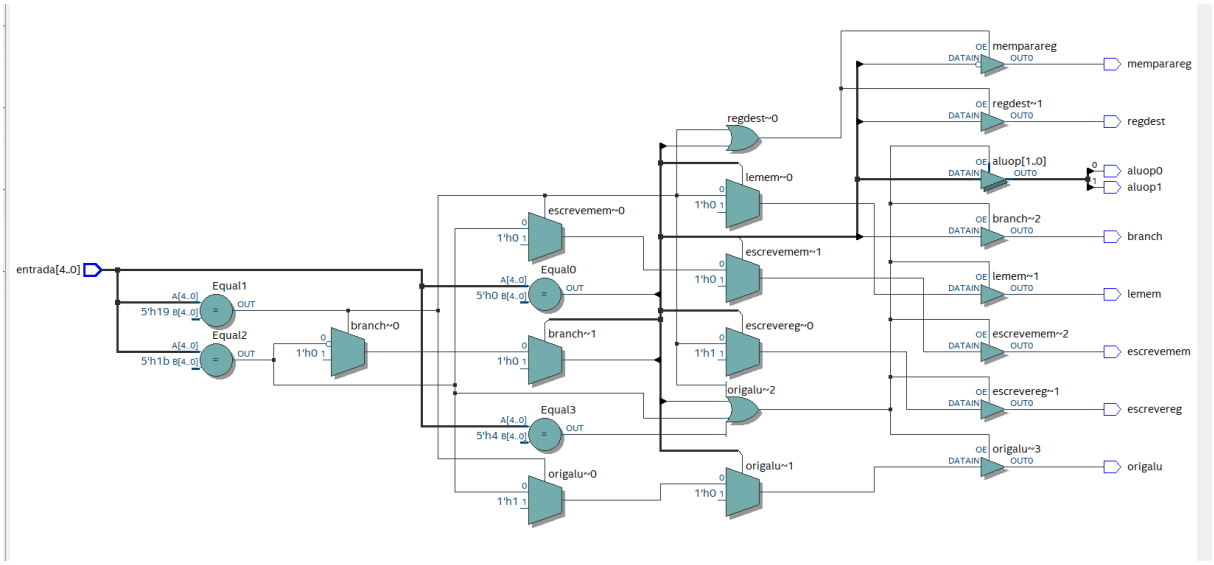
```
        aluop1 : out std_logic;
        aluop0 : out std_logic);
END uc;

ARCHITECTURE behavior OF uc IS
BEGIN
abc: PROCESS (entrada)
        BEGIN
                IF entrada = "00000" then --instrução tipo r
                        regdest                 <= '1';
                        origalu                 <= '0';
                        memparareg  <= '0';
                        escrevereg  <= '1';
                        lemem       <= '0';
                        escrevemem  <= '0';
                        branch      <= '0';
                        aluop1      <= '1';
                        aluop0      <= '0';

                ELSIF entrada = "11001" then --instrução load
                        regdest     <= '0';
                        origalu     <= '1';
                        memparareg  <= '1';
                        escrevereg  <= '1';
```

```
        lemem       <= '1';
        escrevemem  <= '0';
        branch      <= '0';
        aluop1      <= '0';
        aluop0      <= '0';

ELSIF entrada = "11011" then --instrução store
        regdest     <= 'Z';
        origalu     <= '1';
        memparareg  <= 'Z';
        escrevereg  <= '0';
        lemem       <= '0';
        escrevemem  <= '1';
        branch      <= '0';
        aluop1      <= '0';
        aluop0      <= '0';

ELSIF entrada = "00100" then --instrução branch
        regdest     <= 'Z';
        origalu     <= '0';
        memparareg  <= 'Z';
        escrevereg  <= '0';
        lemem       <= '0';
        escrevemem  <= '0';
```

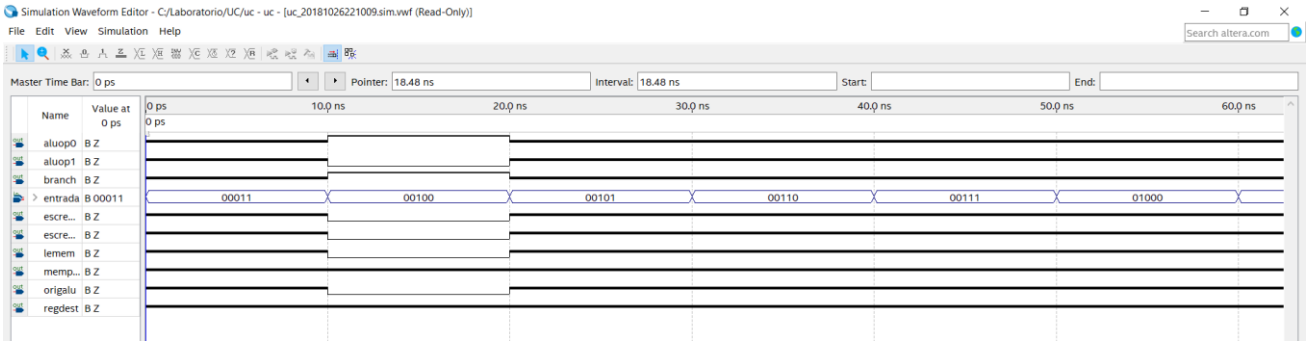
```
                branch      <= '1';
                aluop1      <= '0';
                aluop0      <= '1';
            ELSE
                regdest     <= 'Z';
                origalu     <= 'Z';
                memparareg  <= 'Z';
                escrevereg  <= 'Z';
                lemem       <= 'Z';
                escrevemem  <= 'Z';
                branch      <= 'Z';
                aluop1      <= 'Z';
                aluop0      <= 'Z';
            END if;
        END PROCESS abc;
END behavior;
```

Segundo as entradas vão ser ativadas as flags, segue o padrçao da intrução do tipo r, load, store e branch do mips de 16 bits.

II) RTL View



III) Waveform



IV)

Teste1: entrada:00000, regdest = '1'; origalu <= '0'; mempareg <= '0'; escrevereg <= '1'; lemem <= '0';escrevemem <= '0';branch <= '0';aluop1 <= '1'; aluop0 <= '0';

COMPONENTE 10: ULA

I) A ula ou unidade lógica artmetrica é o componente que realiza as operações do processador, a descrita no código realiza AND, OR, NOT, NOR, NAND, XOR, SHIFT de 2 bits à esquerda, SHIFT de bits à direita, soma e subtração, para

melhor visualização apenas o código principal será mostrado que ele faz é receber duas entradas de 16 bits e uma flag que é oriunda da unidade que controle que indica que operação será realizada.

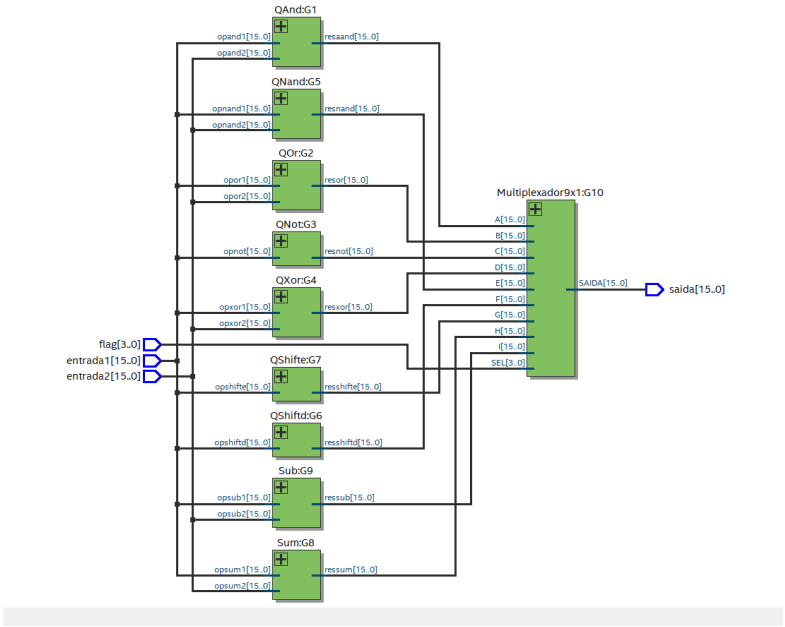
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;

ENTITY ula is
PORT
(
	flag : in std_logic_vector (3 DOWNTO 0);
	entrada1 : in std_logic_vector (15 DOWNTO 0);
	entrada2 : in std_logic_vector (15 DOWNTO 0);
	saida : out std_logic_vector (15 DOWNTO 0));
end ula;

ARCHITECTURE behavior of ula IS
COMPONENT QAnd
PORT
(
	opand1,opand2 : in std_logic_vector(15 DOWNTO 0);
	resaand : out std_logic_vector(15 DOWNTO 0));
END COMPONENT;

COMPONENT QOr
PORT
(
	opor1,opor2 : in std_logic_vector(15 DOWNTO 0);
	resor : out std_logic_vector(15 DOWNTO 0));
END COMPONENT;

COMPONENT QNot
```

```
PORT
(
        opnot: in std_logic_vector(15 DOWNTO 0);
        resnot : out std_logic_vector(15 DOWNTO 0));
END COMPONENT;

COMPONENT QNor
PORT
(
        opnor1,opnor2 : in std_logic_vector(15 DOWNTO 0);
        resnor : out std_logic_vector(15 DOWNTO 0));
END COMPONENT;

COMPONENT QNand IS
PORT
(
        opnand1,opnand2 : in std_logic_vector(15 DOWNTO 0);
        resnand : out std_logic_vector(15 DOWNTO 0));
END COMPONENT;

COMPONENT QXor
PORT
(
        opxor1,opxor2 : in std_logic_vector(15 DOWNTO 0);
        resxor : out std_logic_vector(15 DOWNTO 0));
END COMPONENT;

COMPONENT Sum
PORT
(
        opsum1,opsum2 : in std_logic_vector(15 DOWNTO 0);
        ressum : out std_logic_vector(15 DOWNTO 0));
END COMPONENT;
```

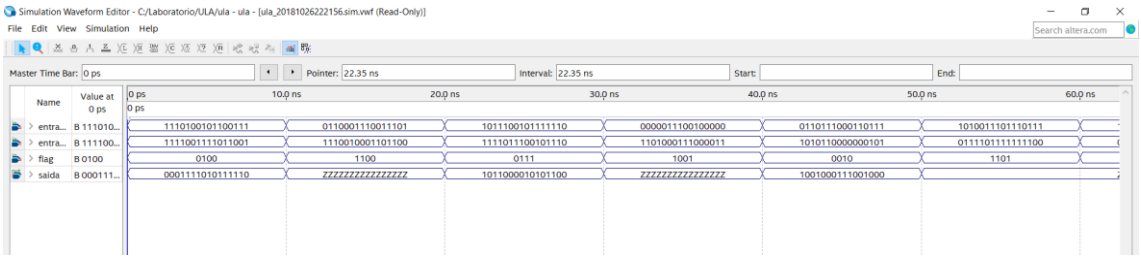
```
COMPONENT Sub
PORT
(
        opsub1,opsub2 : in std_logic_vector(15 DOWNTO 0);
        ressub : out std_logic_vector(15 DOWNTO 0));
END COMPONENT;

COMPONENT QShiftd
PORT
(
        opshiftd : in std_logic_vector(15 downto 0);
        resshiftd : out std_logic_vector(15 downto 0));
END COMPONENT;

COMPONENT QShifte
PORT
(
        opshifte : in std_logic_vector(15 downto 0);
        resshifte : out std_logic_vector(15 downto 0));
END COMPONENT;

COMPONENT  Multiplexador9x1
        PORT(
        SIGNAL A,B,C,D,E,F,G,H,I: IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        SIGNAL SEL : IN STD_LOGIC_VECTOR(3 DOWNTO 0);

        SIGNAL SAIDA: OUT STD_LOGIC_VECTOR(15 DOWNTO 0)) ;

END COMPONENT;

signal auxand : std_logic_vector(15 DOWNTO 0);
signal auxor : std_logic_vector(15 DOWNTO 0);
signal auxnot : std_logic_vector(15 DOWNTO 0);
signal auxnor : std_logic_vector(15 DOWNTO 0);
```

```
signal auxnand : std_logic_vector(15 DOWNTO 0);
signal auxxor : std_logic_vector(15 DOWNTO 0);
signal auxsum : std_logic_vector(15 DOWNTO 0);
signal auxsub : std_logic_vector(15 DOWNTO 0);
signal auxshiftd : std_logic_vector(15 DOWNTO 0);
signal auxshifte : std_logic_vector(15 DOWNTO 0);

BEGIN

        G1: QAnd PORT MAP(entrada1,entrada2, auxand);
        G2: QOr PORT MAP(entrada1,entrada2,auxor);
        G3: QNot PORT MAP(entrada1,auxnot);
        G4: QXor PORT MAP(entrada1,entrada2,auxxor);
        G5: QNand PORT MAP(entrada1,entrada2,auxnand);
        G6: QShiftd PORT MAP(entrada1,auxshiftd);
        G7: QShifte PORT MAP(entrada1,auxshifte);
        G8: Sum PORT MAP(entrada1,entrada2,auxsum);
        G9: Sub PORT MAP (entrada1,entrada2,auxsub);
        G10: Multiplexador9x1 PORT MAP(auxand, auxor, auxnot,
auxxor,auxnand, auxshiftd, auxshifte ,auxsum, auxsub,flag,saida);



END behavior;
```

II) RTL View



III) Waveform



IV)

Teste 1: entrada1 = 10111001010110, entrada2 = 0111010000011010, flag:1000 (subtração), saída = 0100010100111110

Teste2:entrada1: 1101010011110011, entrada2 = 1101010001100100, flag = 0000 (and), saída = 1101010001100000.

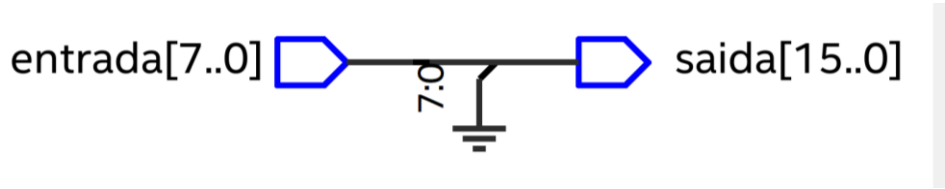
COMPONENTE 11: EXTENSOR DE SINAL

I) O extensor de sinal é um componente fundamental, principalmente na execução da instrução, pois algumas vezes se recebe menos bits do que se precisa para que se possa ser realizada a operação e com o extensor de sinal não há perigo de ter problemas por causa de tamanho incompatível de bits, o extensor implementado é um extensor de 8 para 16 bits, a ideia do código é ele pegar uma entrada de 8 bits e concatenar com um vetor de 8 bits com zeros, é importante que o vetor concatenado fique a frente do vetor original de 16, para que não tenha mudança do valor do vetor.

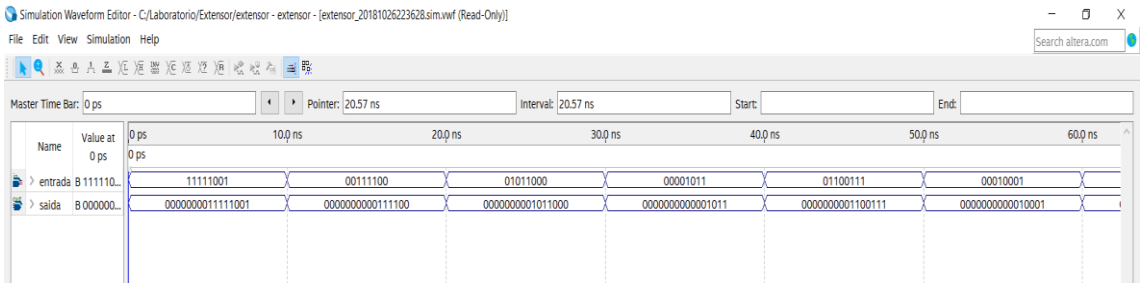
Código:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY extensor IS
PORT
(
        entrada : in std_logic_vector(7 DOWNTO 0);
        saida : out std_logic_vector(15 DOWNTO 0));
END extensor;

ARCHITECTURE behavior of extensor IS
BEGIN
        saida<= "00000000" & entrada;
END behavior;
```

II) RTL View



III) Waveform



IV)

Teste 1: entrada = 00000000, saída = 0000000000000000;
Teste 2: entrada = 10101110, saída = 0000000010101110;
Teste 3: entrada = 10111011, saída = 0000000010111011;
Teste 4: entrada = 11111111, saída = 0000000011111111;

COMPONENTE 13: CONTADOR SINCRONO

I) Contador Síncrono é um componente que baseia seu comportamento em um auxiliar para realizar a contagem dos tempos, o contador síncrono desenvolvido é de 3 bits, o contador síncrono terá um comportamento de saída : 000, 001, 010, 011, 100, 101, 110, 111 e recomeça com 000, como ele se comporta com 3 bits, seu comportamento pode ser definido por uma máquina de estados, só irá ser realizada a contagem quando o j e k de cada registrador que o compõe for high, também é utilizada uma porta and para diminuir o atraso, que é um problema do contador assíncrono que tem um clock para cada registrador, diferente do contador síncrono.

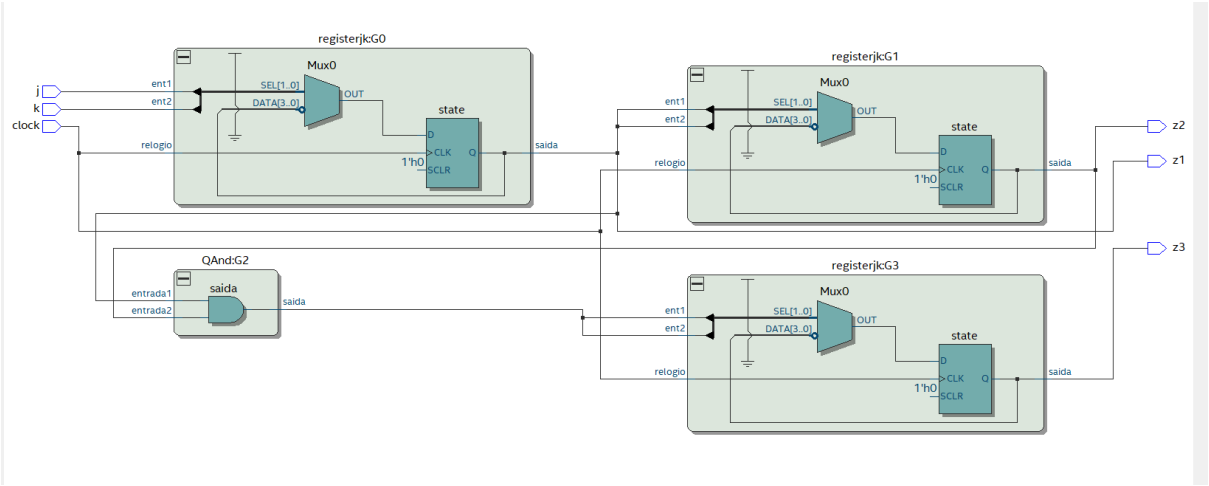
Código:

```
library ieee;
use ieee.std_logic_1164.all;

entity registerjk is
	port (
	ent1, ent2,relogio: in std_logic;
	saida: out std_logic );
end registerjk;

architecture behavior of registerjk is

  signal state: std_logic;
  signal input: std_logic_vector(1 downto 0);

begin

  input <= ent1 & ent2;

  p: process(relogio) is
  begin
	if (rising_edge(relogio)) then
		case (input) is
			when "11" => state <= not (state);
			when "10" => state <= '1';
			when "01" => state <= '0';
			when others => null;
		end case;
	end if;

  end process;
  saida <= state;
```

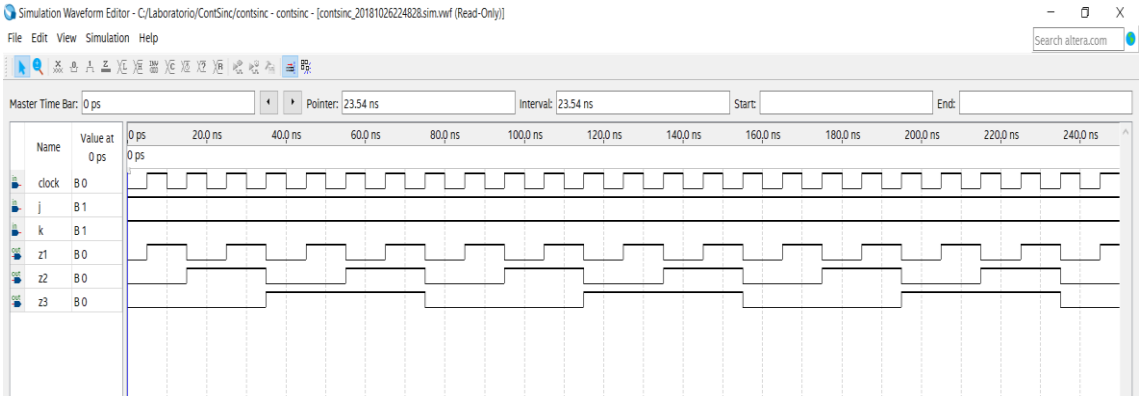
```
end behavior;
-------------------------------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY QAnd IS
PORT
(
        entrada1,entrada2 : in std_logic;
        saida : out std_logic);
end QAnd;

ARCHITECTURE behavior OF QAnd IS
BEGIN

        saida <= entrada1 and entrada2;

END behavior;

-------------------------------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

ENTITY contsinc IS
PORT
(
                j,k,clock : in std_logic;
                z1,z2,z3 : out std_logic);
END contsinc;

ARCHITECTURE behavior OF contsinc IS

component registerjk
```

```
        port (
        ent1, ent2,relogio: in std_logic;
        saida: out std_logic );
end component;

COMPONENT QAnd
PORT
(
        entrada1,entrada2 : in std_logic;
        saida : out std_logic);
END COMPONENT;

SIGNAL a1,a2,a3,a4 : std_logic;
BEGIN
        G0: registerjk PORT MAP (j,k,clock,a1);
        z1 <= a1;
        G1: registerjk PORT MAP (a1,a1,clock,a2);
        z2 <= a2;
        G2: QAnd PORT MAP (a1,a2,a3);
        G3: registerjk PORT MAP (a3,a3,clock,a4);
        z3 <= a4;
END behavior;
```

II) RTL View



III) Waveform



IV) Os testes que podem ser realizados estão descritos na waveform acima, com a borda alta de j e k, seguindo o clock se terá 000, 001, 010, 011, 100, 101, 110, 111.