EA773 - Laboratório de Circuitos Lógicos
Relatório Roteiro 5

Autor: Mateus Henrique Silva Araújo

RA: 184940

Data: 03/11/2022

0. Introdução:

Esta atividade busca trabalhar os conceitos relacionados a dispositivos de memória, permitindo o estudo e a implementação tanto de unidades de memória não volátil de leitura (ROM) quanto de unidades de memória volátil de escrita e leitura (RAM), por meio do uso não só da lógica *tri-state*, mas também de elementos armazenadores de memória (*flip-flops*).

O objetivo final desta atividade é o desenvolvimento de cinco unidades de memória distintas: quatro memória ROM 64x8 distintas e uma memória RAM 16x4. A implementação de ambas está baseada no circuito genérico apresentado na figura 1.

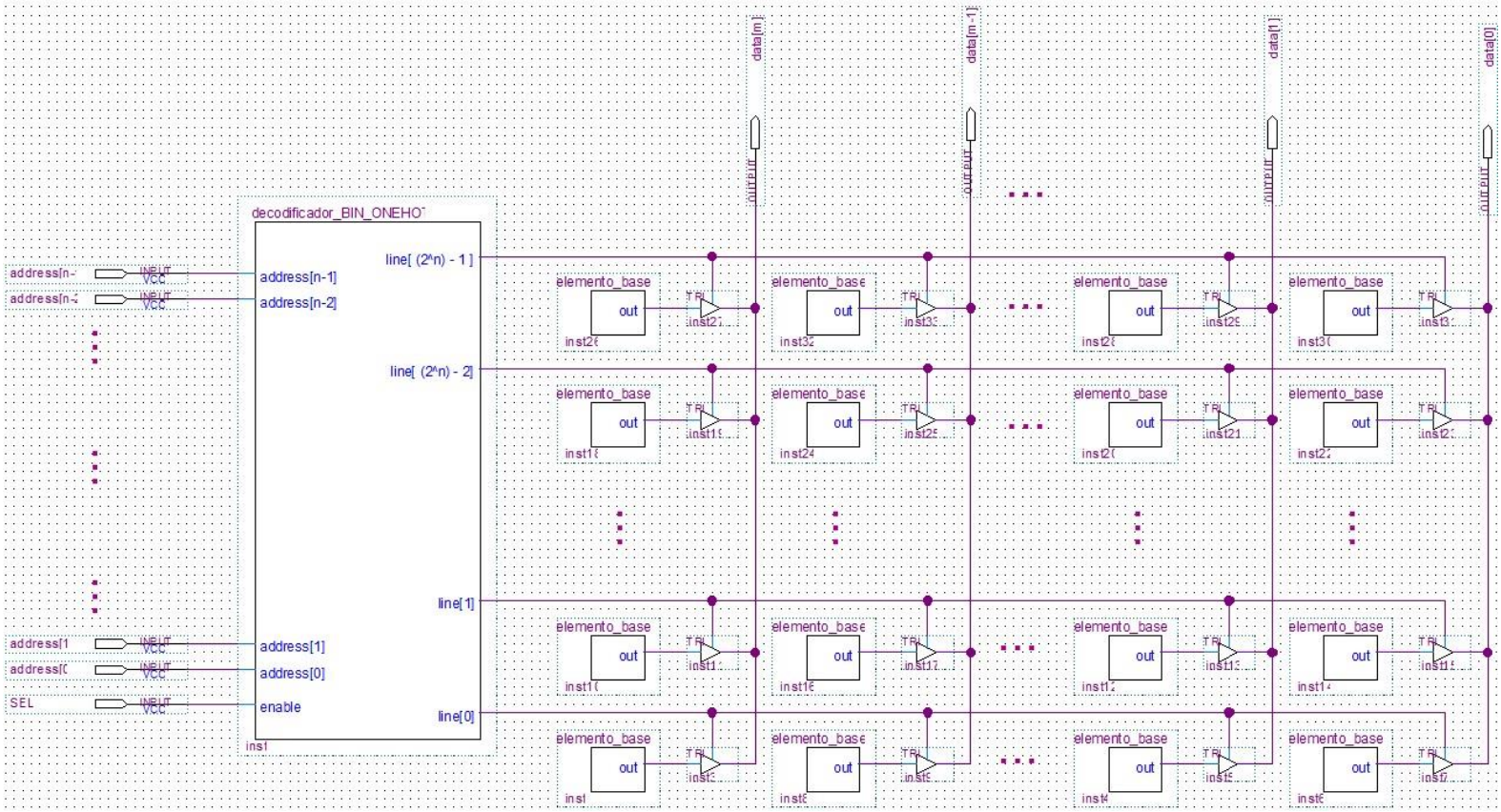


Figura 1: Circuito genérico base para a implementação dos dispositivos de memória desta atividade. O circuito se baseia em um decodificador binário para one-hot, o qual converte uma entrada binária para uma saída em representação one-hot (na qual somente uma das saídas tem valor igual a um), e em diversas linhas de células armazenadoras de dados, as quais são ligadas a um barramento controlado por buffers tri-state.

O circuito apresentado na figura acima identifica uma memória com **n** palavras de **m** bits, cujo modo de armazenamento de dados é definido pelo elemento base utilizado na sua implementação. No caso de uma memória ROM não volátil, uma vez que o dado presente em cada bit de cada palavra de memória se mantém constante mesmo mediante ao desligamento da

alimentação, o elemento base empregue deve ser ou *GND* ou *VCC* (dependendo se o bit em cada palavra será 0 ou 1, respectivamente). Já para uma memória RAM volátil com a funcionalidade de leitura e escrita, o elemento base empregue devem ser *flip-flops* (os quais armazenam os dados carregados em si enquanto a alimentação do circuito for mantida), sendo necessário também a adição de uma lógica de carga síncrona no circuito.

Nessa memória genérica, uma entrada de endereço ($address[(n-1)\,..\,0]$) seleciona qual palavra deve liberar sua informação para o barramento de saída ($data[m\,..\,0]$). Esta seleção só é possível por meio do decodificador empregue, o qual converte o valor binário do endereço desejado para sua representação em *one-hot* (na qual somente um dos bits apresenta valor 1 e os demais são zero). Dessa forma, como todos os circuitos desta atividade se sustentarão no comportamento desse decodificador, ele foi implementado em um módulo à parte para que, assim, os diferentes projetos desenvolvidos pudessem utilizá-lo simultaneamente. O procedimento de projeto e implementação de tal módulo (nomeado como "*decodificadorBIN_ONEH*") está apresentado a seguir:

I. Escopo:

Projeto de um circuito combinacional, o qual implemente um decodificador de binário para *one-hot* com 4 bits de entrada (na atividade, circuitos com maiores números de entradas de endereço serão construídos usando de módulos de menores, sendo a base de todos eles módulos com 4 bits de endereço).

II. Especificação de alto nível:

Entradas:
$\underline{x} = (x_3, x_2, x_1, x_0)$ com $x_i \in \{0,1\}$ e $i = 0, \dots, 3$;
$enable \in \{0,1\}$.

Saída:
$\underline{y} = (y_{15}, y_{14}, \dots, y_1, y_0)$ com $y_i \in \{0,1\}$ e $i = 0, 1, \dots, 15$.

Função de saída:

$$y_i = \begin{cases} 1, & para\ i = \sum_{k=0}^{n-1} x_j \cdot 2^j\ \ se\ enable = 1 \\ 0, & se\ enable = 0 \end{cases}$$

III. Especificação binária:

Entradas:
$\underline{x} = (x_3, x_2, x_1, x_0)$ com $x_i \in \{0,1\}$ e $i = 0, \dots, 3$;
$enable \in \{0,1\}$.

Saída:
$\underline{y} = (y_{15}, y_{14}, \dots, y_1, y_0)$ com $y_i \in \{0,1\}$ e $i = 0, 1, \dots, 15$.

Função de saída:
Como a tabela verdade completa para esse circuito teria 32 linhas (pois ele recebe 5 bits de entradas) e uma das entradas do circuito é um *enable*, podemos simplificá-la avaliando as saídas somente em função das entradas $\underline{x}$ e depois condenarmos elas à entrada $enable$. Dessa forma, temos a seguinte tabela verdade:

x_3	x_2	x_1	x_0	y_{15}	y_{14}	y_{13}	y_{12}	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IV. Minimizações:

Avaliando a tabela verdade apresentada acima, é possível perceber que cada saída só é alta para uma única combinação dos valores da entrada $\underline{x}$, o que é uma propriedade da conversão para a representação *one-hot*. Isso faz com que não seja necessária a construção de mapas de Karnaugh para obtenção da expressão mínima de cada saída, pois ela é dada diretamente pela única combinação das entradas que geram o valor 1. Combinando as expressões mínimas obtidas pela tabela acima com a lógica da entrada *enable* (ou seja, a saída dever ser 0 se $enable = 0$), obteremos as seguintes funções:

$$y_0 = enable \cdot x_3' x_2' x_1' x_0' \quad ; \quad y_8 = enable \cdot x_3 x_2' x_1' x_0'$$

$$y_1 = enable \cdot x_3' x_2' x_1' x_0 \quad ; \quad y_9 = enable \cdot x_3 x_2' x_1' x_0$$

$$y_2 = enable \cdot x_3' x_2' x_1 x_0' \quad ; \quad y_{10} = enable \cdot x_3 x_2' x_1 x_0'$$

$$y_3 = enable \cdot x_3' x_2' x_1 x_0 \quad ; \quad y_{11} = enable \cdot x_3 x_2' x_1 x_0$$

$$y_4 = enable \cdot x_3' x_2 x_1' x_0' \quad ; \quad y_{12} = enable \cdot x_3 x_2 x_1' x_0'$$

$$y_5 = enable \cdot x_3' x_2 x_1' x_0 \quad ; \quad y_{13} = enable \cdot x_3 x_2 x_1' x_0$$

$$y_6 = enable \cdot x_3' x_2 x_1 x_0' \quad ; \quad y_{14} = enable \cdot x_3 x_2 x_1 x_0'$$

$$y_7 = enable \cdot x_3' x_2 x_1 x_0 \quad ; \quad y_{15} = enable \cdot x_3 x_2 x_1 x_0$$

V. Esquemático do Circuito:

As equações de minimização apresentadas acima foram utilizadas para a construção do circuito do decodificador. O diagrama esquemático do circuito resultante está disposto na figura 2.

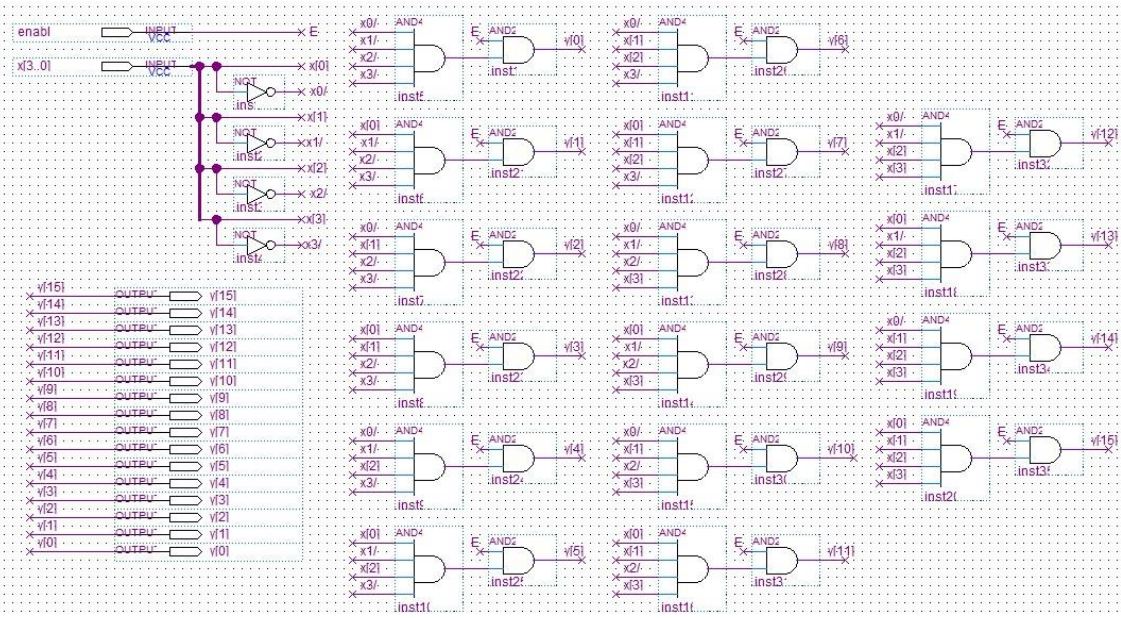
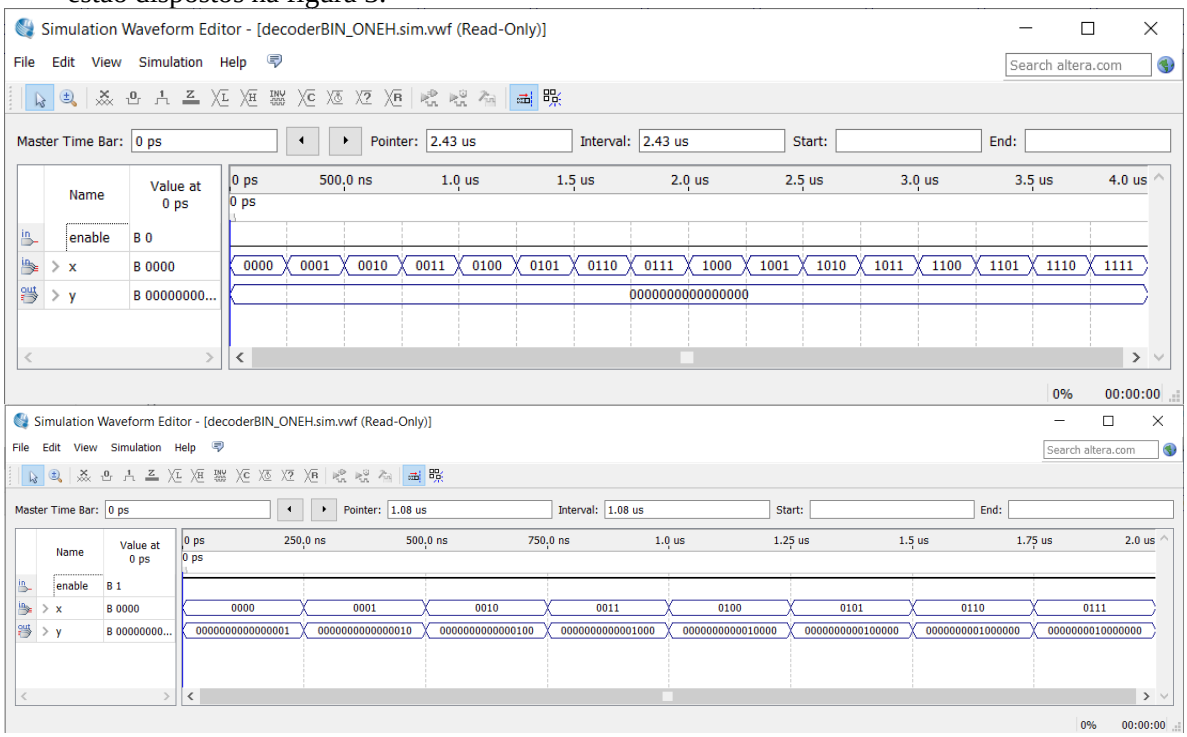


Figura 2: diagrama esquemático do módulo "decoderBIN_ONEH", o qual será utilizado em todas as unidades básicas de memória desenvolvidas nessa atividade e, portanto, estará indiretamente em todos os circuitos programados.

VI. Simulações:

O circuito do decodificador passou por simulações funcionais as quais testaram todas as combinações possíveis de suas entradas e comprovaram que seu funcionamento está de acordo com o esperado. Os resultados das simulações feitas estão dispostos na figura 3.



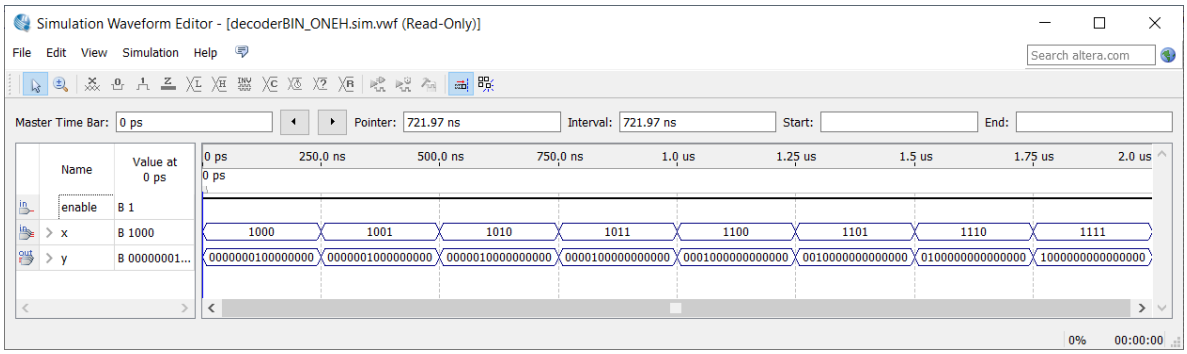


Figura 3: simulações funcionais do circuito do módulo "decoderBIN_ONEH". Nelas podemos notar que o comportamento do circuito está de acordo com o especificado.

Agora que o funcionamento do decodificador principal já está claro, é necessário ressaltar a lógica desejada para a implementação das memórias ROM 64x8 dessa atividade. Com o intuito de facilitar o trabalho de programação e construção do circuito final, cada memória ROM 64x8 será implementada a partir de duas memórias ROM 32x8, que, por sua vez, serão geradas a partir de duas memórias ROM 16x8, sendo essas compostas de duas memórias ROM 16x4. Além disso, foi requisitado que cada módulo apresentasse conteúdo específico, dado de acordo com as tabelas abaixo (sendo que, para endereços acima de 16, os dados deveriam repetir ciclicamente as 16 primeiras configurações):

Módulo de memória 0	
Endereço (Hexadecimal)	Conteúdo (Binário)
0	00000000
1	11111111
2	00000000
3	11111111
4	00000000
5	11111111
6	00000000
7	11111111
8	00000000
9	11111111
A	00000000
B	11111111
C	00000000
D	11111111
E	00000000
F	11111111

Módulo de memória 1	
Endereço (Hexadecimal)	Conteúdo (Binário)
0	10101010
1	01010101
2	10101010
3	01010101
4	10101010
5	01010101
6	10101010
7	01010101
8	10101010
9	01010101
A	10101010
B	01010101
C	10101010
D	01010101
E	10101010
F	01010101

Módulo de memória 2	
Endereço (Hexadecimal)	Conteúdo (Binário)
0	00000000
1	00000001
2	00000010
3	00000100
4	00001000
5	00010000
6	00100000
7	01000000
8	10000000
9	01000000
A	00100000
B	00010000
C	00001000
D	00000100
E	00000010
F	00000001

Módulo de memória 3	
Endereço (Hexadecimal)	Conteúdo (Binário)
0	11111111
1	11111110
2	11111101
3	11111011
4	11110111
5	11101111
6	11011111
7	10111111
8	01111111
9	10111111
A	11011111
B	11101111
C	11110111
D	11111011
E	11111101
F	11111110

Essas diferentes configurações de conteúdo provocaram a necessidade do desenvolvimento de diversos módulos ROM 16x4, 16x8 e 32x8. Portanto, com intuito de organizar este relatório e explicitar os diferentes circuitos que serão desenvolvidos em cada projeto, estão dispostas, nas figuras 4 a 7, as organizações hierárquicas das ROM 64x8 construídas, contendo cada módulo empregue na sua implementação.

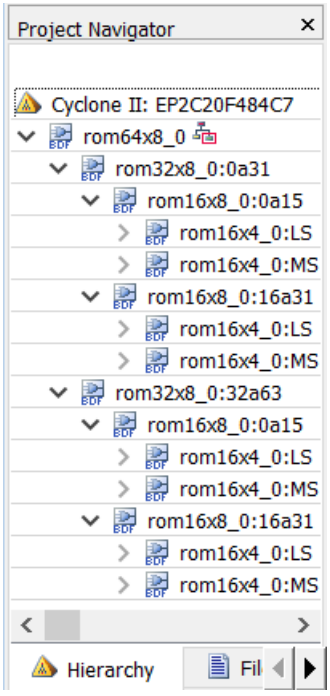


Figura 4: Hierarquia do circuito da ROM do módulo 0 ("rom64x8_0"). Para implementá-la, foram usados os módulos "rom32x8_0", "rom16x8_0" e "rom16x4_0".

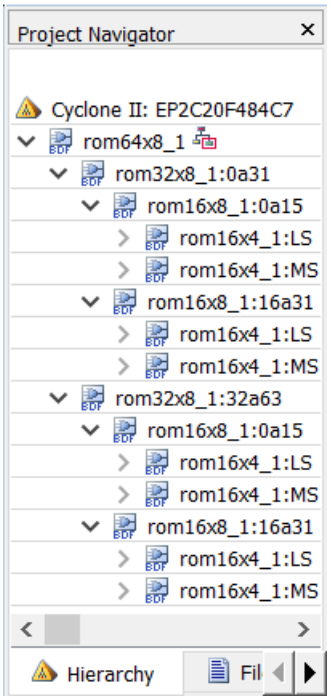


Figura 5: Hierarquia do circuito da ROM do módulo 1 ("rom64x8_1"). Para implementá-la, foram usados os módulos "rom32x8_1", "rom16x8_1" e "rom16x4_1".

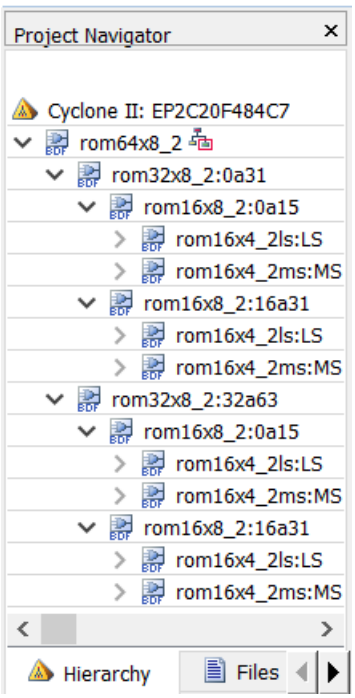


Figura 6: Hierarquia do circuito da ROM do módulo 2 ("rom64x8_2"). Para implementá-la, foram usados os módulos "rom32x8_2", "rom16x8_2", "rom16x4_2ls" e "rom16x4_2ms".

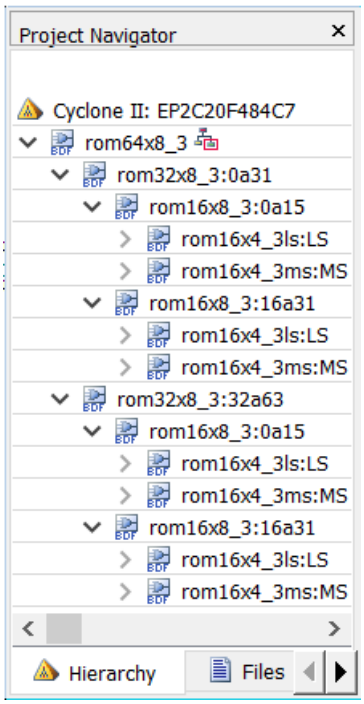


Figura 7: Hierarquia do circuito da ROM do módulo 3 ("rom64x8_3"). Para implementá-la, foram usados os módulos "rom32x8_3", "rom16x8_3", "rom16x4_3ls" e "rom16x4_3ms".

1. Projeto 1: ROMs 16x4

1.1. Escopo:

Projeto de circuitos combinacionais que implementem dispositivos de memória não volátil de leitura exclusiva com 16 palavras de memória de 4 bits. Os dispositivos devem ser implementados de modo a conter o conteúdo adequado para a implementação de cada módulo da atividade.

1.2. Especificação de alto-nível:

Entradas:
$\underline{addr} = (addr_3, addr_2, addr_1, addr_0)$, com $addr_i \in \{0,1\}$ e $i = 0, \dots, 3$;
$SEL \in \{0,1\}$.

Saída:
$\underline{d} = (d_3, d_2, d_1, d_0)$, com $d_i \in \{0,1\}$ e $i = 0, \dots, 3$.

Função de Saída:

$$\underline{d} = \begin{cases} [\underline{addr}], & se\ SEL = 1 \\ ZZZZ, & se\ SEL = 0 \end{cases},$$

onde $[\underline{addr}]$ significa o conteúdo da memória no endereço indicado pela entrada $\underline{addr}$.

1.3. Especificação binária:

Entradas:
$\underline{addr} = (addr_3, addr_2, addr_1, addr_0)$, com $addr_i \in \{0,1\}$ e $i = 0, \ldots, 3$;
$SEL \in \{0,1\}$.

Saída:
$\underline{d} = (d_3, d_2, d_1, d_0)$, com $d_i \in \{0,1\}$ e $i = 0, \ldots, 3$.

Função de saída:

Cada módulo do tipo ROM 16x4 apresenta uma função de saída distinta, a qual depende do conteúdo contido nele. Dessa forma, nas tabelas abaixo estão apresentadas as relações de conteúdo em função dos endereços utilizada para implementar cada uma das ROM 16x4 empregues nessa atividade.

Módulo "*rom16x4_0*"	
$\underline{addr}$	$\underline{d}$
0000	0000
0001	1111
0010	0000
0011	1111
0100	0000
0101	1111
0110	0000
0111	1111
1000	0000
1001	1111
1010	0000
1011	1111
1100	0000
1101	1111
1110	0000
1111	1111

Módulo "*rom16x4_1*"	
$\underline{addr}$	$\underline{d}$
0000	1010
0001	0101
0010	1010
0011	0101
0100	1010
0101	0101
0110	1010
0111	0101
1000	1010
1001	0101
1010	1010
1011	0101
1100	1010
1101	0101
1110	1010
1111	0101

Módulo "*rom16x4_2ms*"	
$\underline{addr}$	$\underline{d}$
0000	0000
0001	0000
0010	0000
0011	0000
0100	0000
0101	0001
0110	0010
0111	0100
1000	1000
1001	0100
1010	0010
1011	0001
1100	0000
1101	0000
1110	0000
1111	0000

Módulo "*rom16x4_2ls*"	
$\underline{addr}$	$\underline{d}$
0000	0000
0001	0001
0010	0010
0011	0100
0100	1000
0101	0000
0110	0000
0111	0000
1000	0000
1001	0000
1010	0000
1011	0000
1100	1000
1101	0100
1110	0010
1111	0001

Módulo "*rom16x4_3ms*"	
$\underline{addr}$	$\underline{d}$
0000	1111
0001	1111
0010	1111
0011	1111
0100	1111
0101	1110
0110	1101
0111	1011
1000	0111
1001	1011
1010	1101
1011	1110
1100	1111
1101	1111
1110	1111
1111	1111

Módulo "*rom16x4_3ls*"	
$\underline{addr}$	$\underline{d}$
0000	1111
0001	1110
0010	1101
0011	1011
0100	0111
0101	1111
0110	1111
0111	1111
1000	1111
1001	1111
1010	1111
1011	1111
1100	0111
1101	1011
1110	1101
1111	1110

Vale lembrar que a saída $\underline{d}$ só assume os valores dados na tabela acima se $SEL = 1$, ficando em estado de alta impedância caso contrário. Além disso, destaca-se que, como as palavras de memória dos módulos 0 e 1 apresentam os 4 bits mais significativos iguais aos 4 bits menos significativos para todos os endereços, faz- se necessário a implementação de somente um tipo de memória ROM 16x4 para tais módulos. Já os módulos 2 e 3 precisam de dois tipos, ao passo que a parte mais significativa da palavra de memória é diferente da parte menos significativa neles.

1.4. Minimizações:

Para a implementação das ROMs 16x4, o modelo genérico de dispositivo de memória apresentado na introdução deste relatório será utilizado, não sendo necessário fazer qualquer minimização. Será empregue como decodificador o módulo "*decodificadorBIN_ONEH"*, o qual também já foi apresentado, e como elemento base os pinos *GND* (para bits com valor zero) e *VCC* (para bits com valor 1).

1.5. Esquemático do circuito:

Os módulos descritos foram construídos seguindo o modelo genérico e os diagramas esquemáticos de seus circuitos resultantes estão apresentados nas figuras 8 a 13.

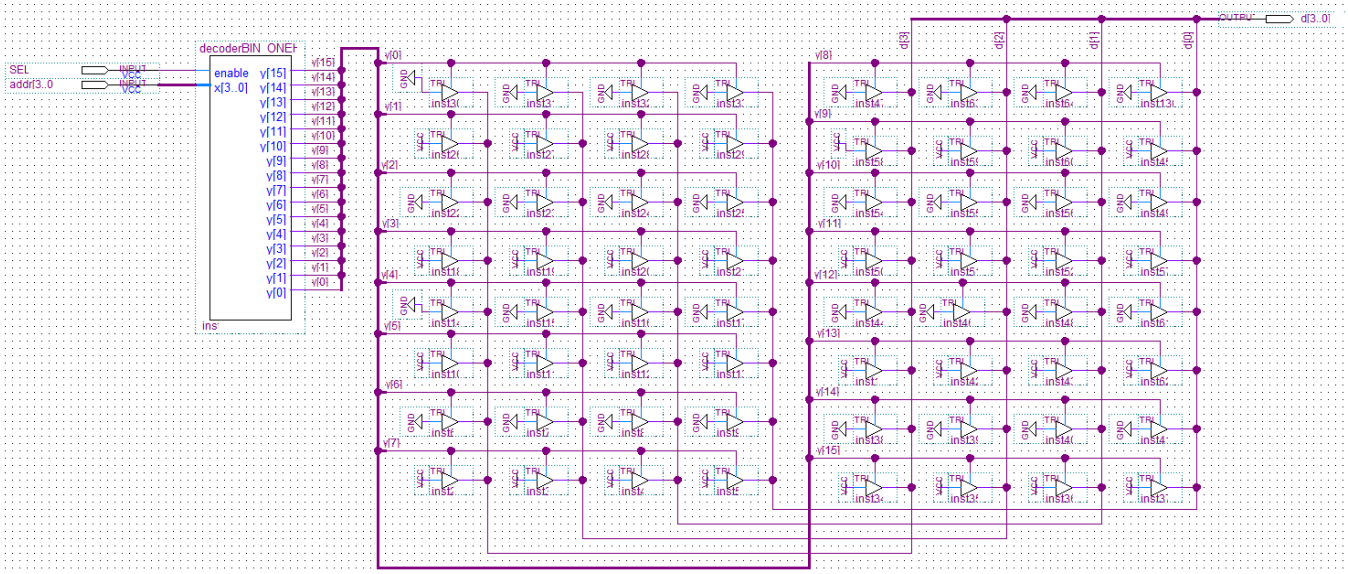


Figura 8: Diagrama esquemático do circuito do módulo "rom16x4_0" o qual implementa uma unidade de memória com 16 palavras de 4 bits e constituirá a ROM 64x8 do módulo 0. Como o conteúdo das palavras de memória da rom do módulo 0 apresentam os quatro bits mais significativos iguais aos 4 bits menos significativos, é necessário a implementação de somente um tipo de ROM 16x4 para esse módulo.

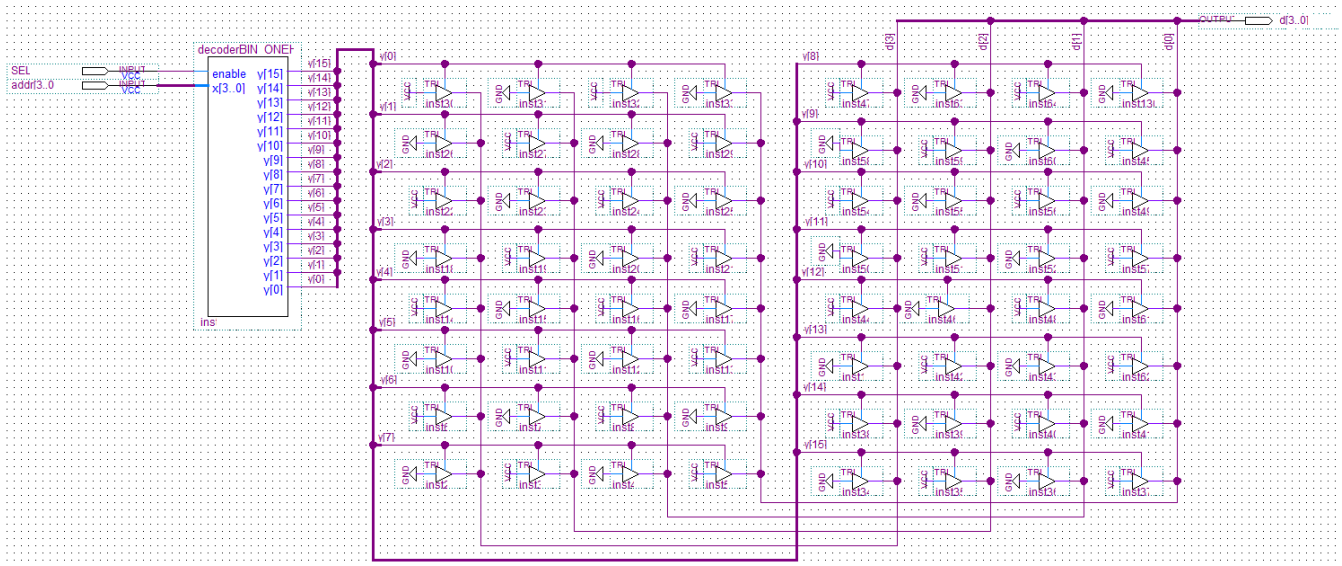


Figura 9: Diagrama esquemático do circuito do módulo "rom16x4_1" o qual implementa uma unidade de memória com 16 palavras de 4 bits e constituirá a ROM 64x8 do módulo 1. Como o conteúdo das palavras de memória da rom do módulo 1 apresentam os quatro bits mais significativos iguais aos 4 bits menos significativos, é necessário a implementação de somente um tipo de ROM 16x4 para esse módulo.

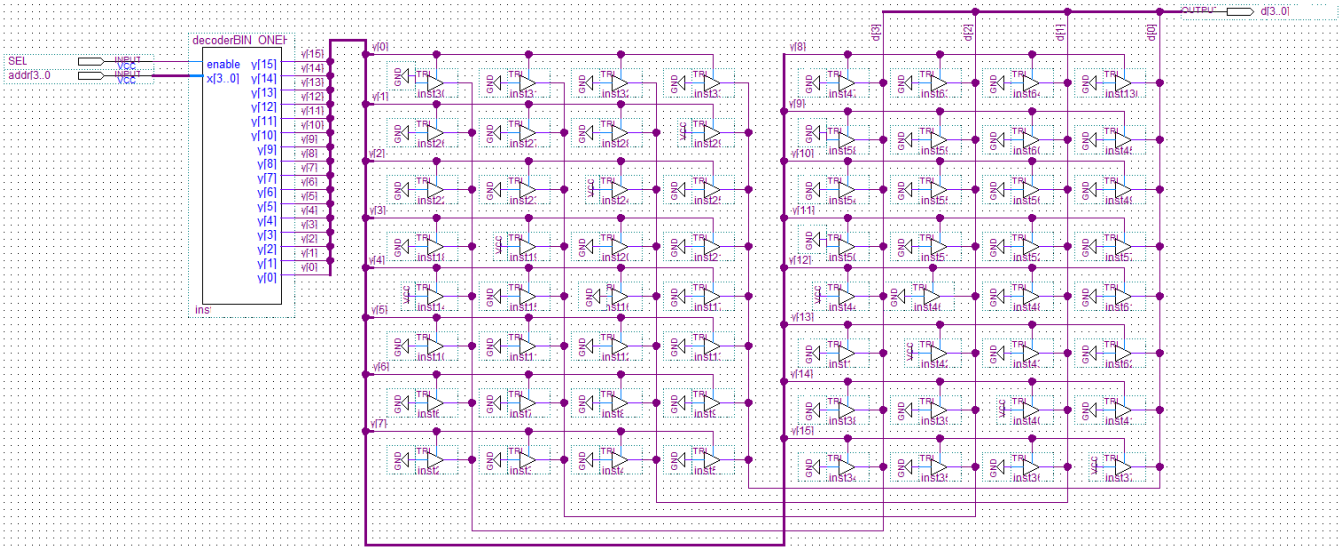


Figura 10: Diagrama esquemático do circuito do módulo "rom16x4_2ls" o qual implementa uma unidade de memória com 16 palavras de 4 bits e constituirá a ROM 64x8 do módulo 2. Como o conteúdo das palavras de memória da rom do módulo 2 apresentam os quatro bits mais significativos diferentes dos 4 bits menos significativos, é necessário a implementação de dois tipos de ROM 16x4 para esse módulo.

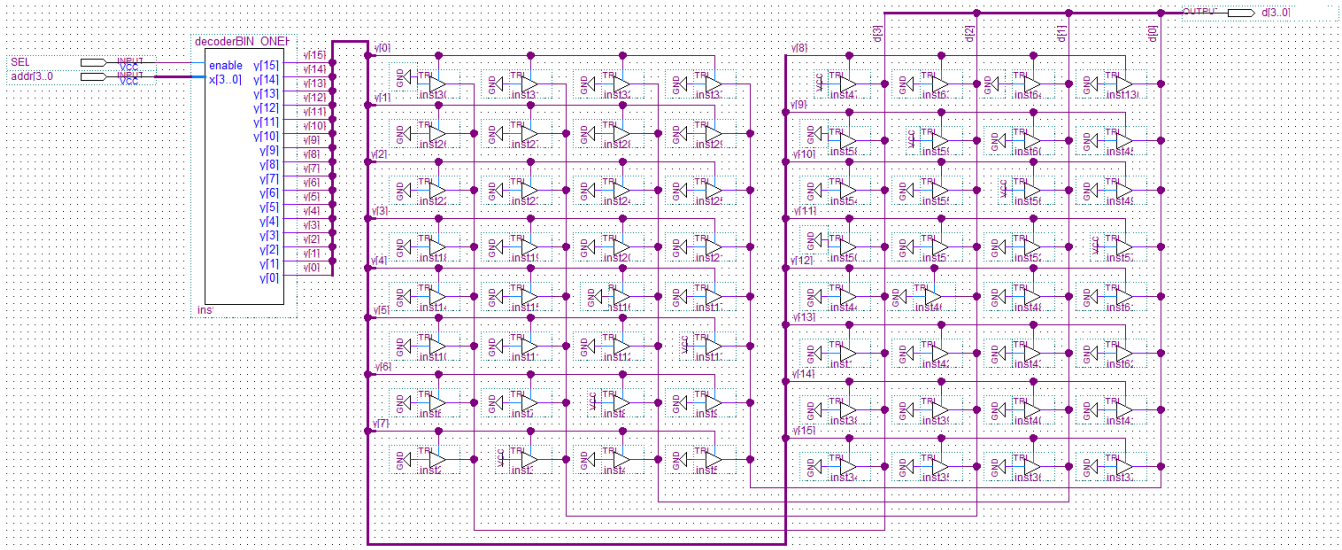


Figura 11: Diagrama esquemático do circuito do módulo "rom16x4_2ms" o qual implementa uma unidade de memória com 16 palavras de 4 bits e constituirá a ROM 64x8 do módulo 2. Como o conteúdo das palavras de memória da rom do módulo 2 apresentam os quatro bits mais significativos diferentes dos 4 bits menos significativos, é necessário a implementação de dois tipos de ROM 16x4 para esse módulo.

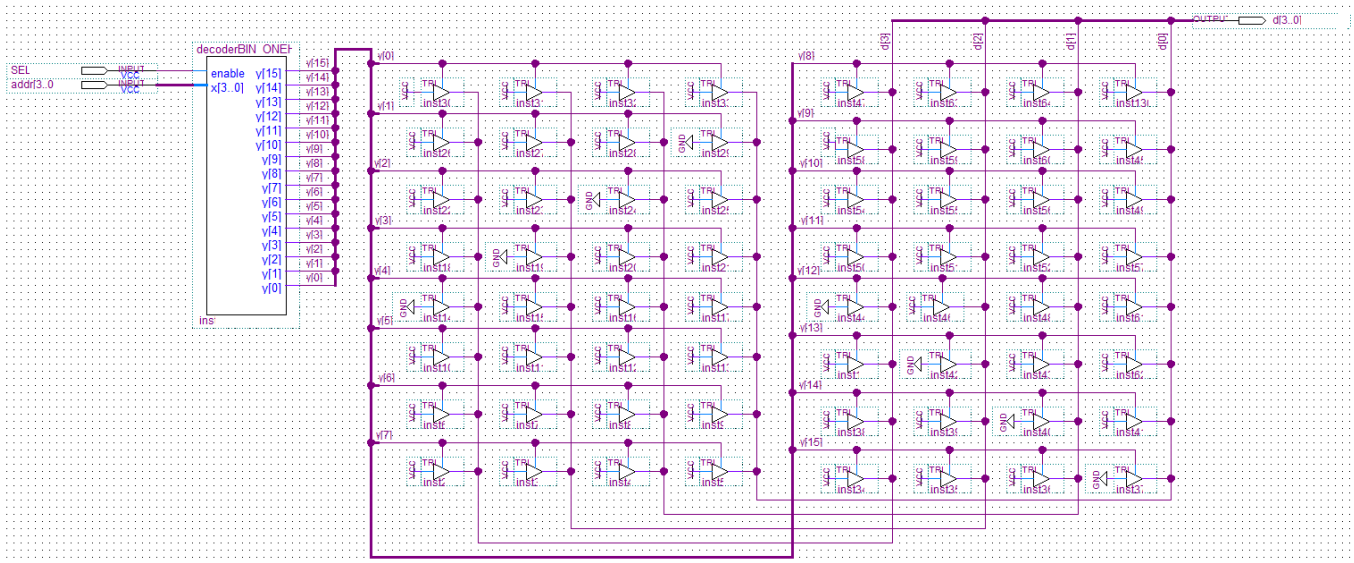


Figura 12: Diagrama esquemático do circuito do módulo "rom16x4_3ls" o qual implementa uma unidade de memória com 16 palavras de 4 bits e constituirá a ROM 64x8 do módulo 3. Como o conteúdo das palavras de memória da rom do módulo 3 apresentam os quatro bits mais significativos diferentes dos 4 bits menos significativos, é necessário a implementação de dois tipos de ROM 16x4 para esse módulo.

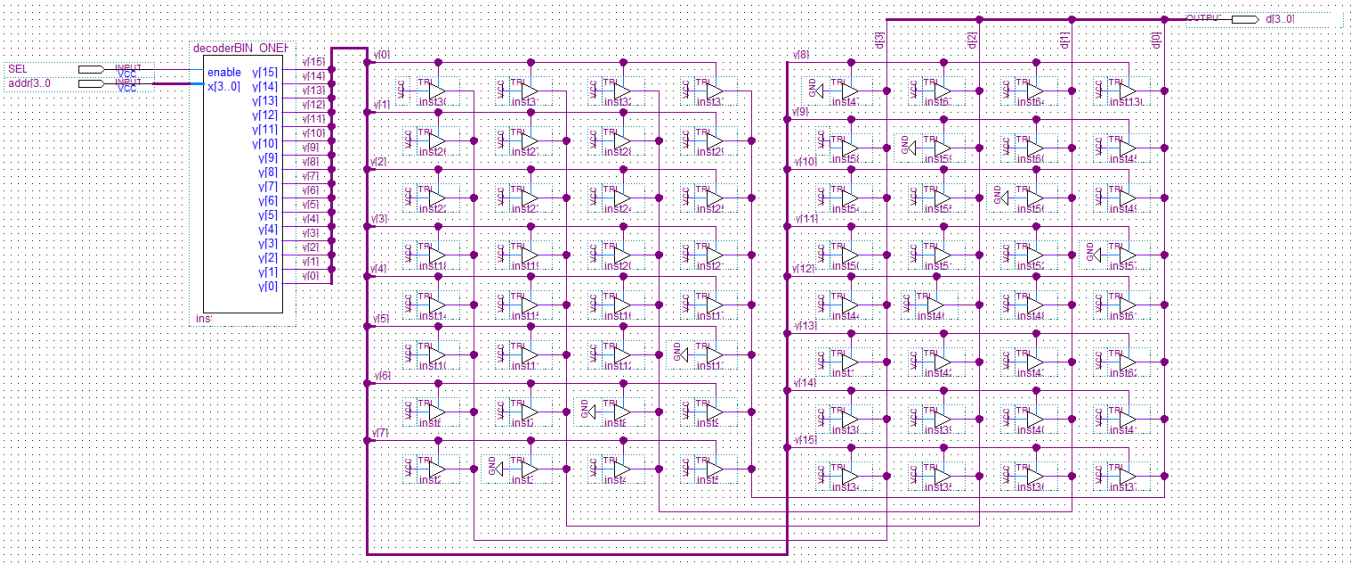


Figura 13: Diagrama esquemático do circuito do módulo "rom16x4_3ms" o qual implementa uma unidade de memória com 16 palavras de 4 bits e constituirá a ROM 64x8 do módulo 3. Como o conteúdo das palavras de memória da rom do módulo 3 apresentam os quatro bits mais significativos diferentes dos 4 bits menos significativos, é necessário a implementação de dois tipos de ROM 16x4 para esse módulo.

1.6. Simulações:

Para testar as memórias construídas, simulações funcionais acessando cada endereço disponível, bem como avaliando seu comportamento mediante a alteração da entrada SEL, de cada módulo implementado foram realizadas. Os resultados obtidos a partir delas estão dispostos nas figuras 14 a 19.

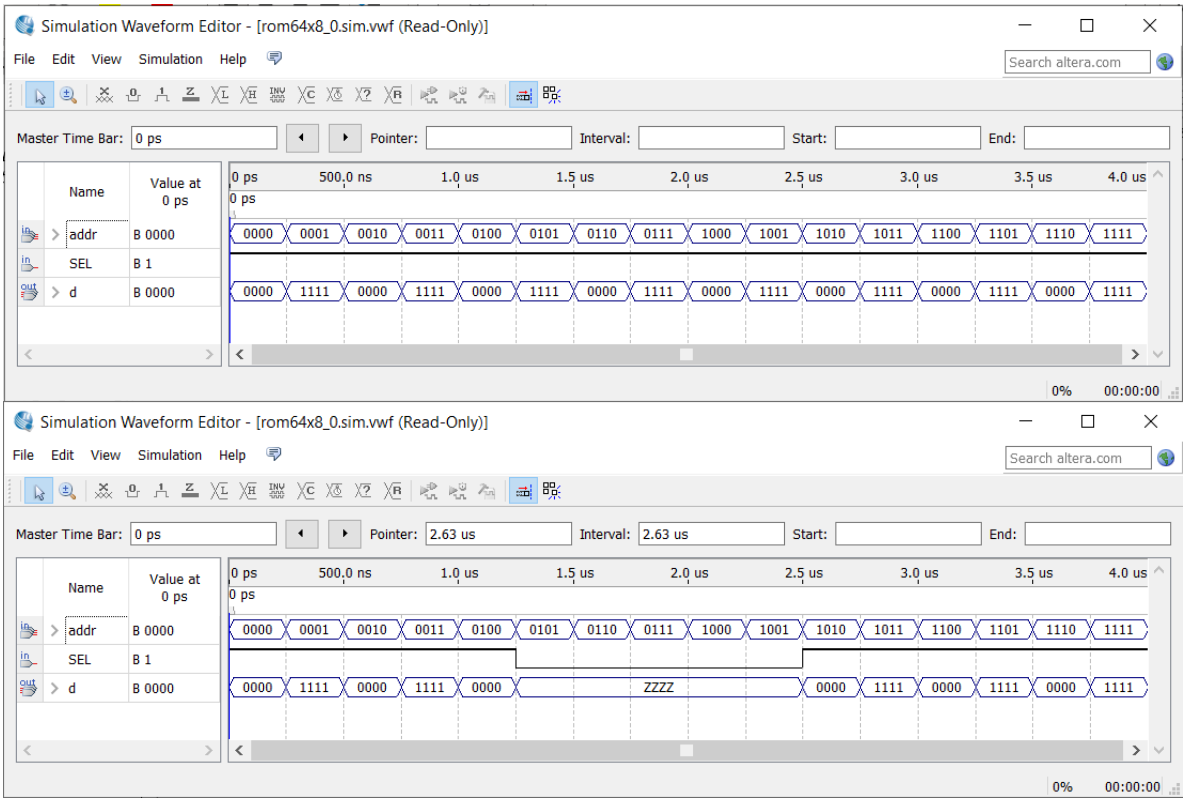


Figura 14: Simulações funcionais do circuito do módulo "rom16x4_0". Nela é possível perceber que o sistema se comporta de acordo com o especificado.

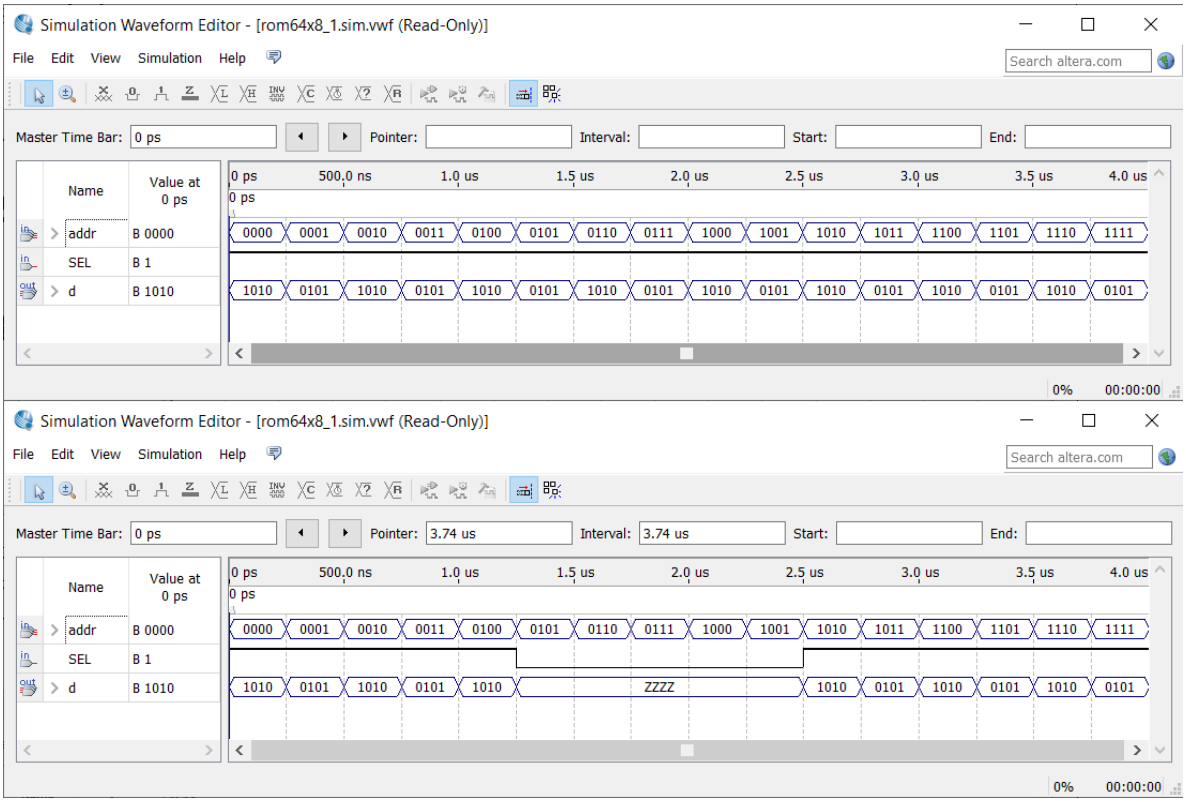


Figura 15: Simulações funcionais do circuito do módulo "rom16x4_1". Nela é possível perceber que o sistema se comporta de acordo com o especificado.

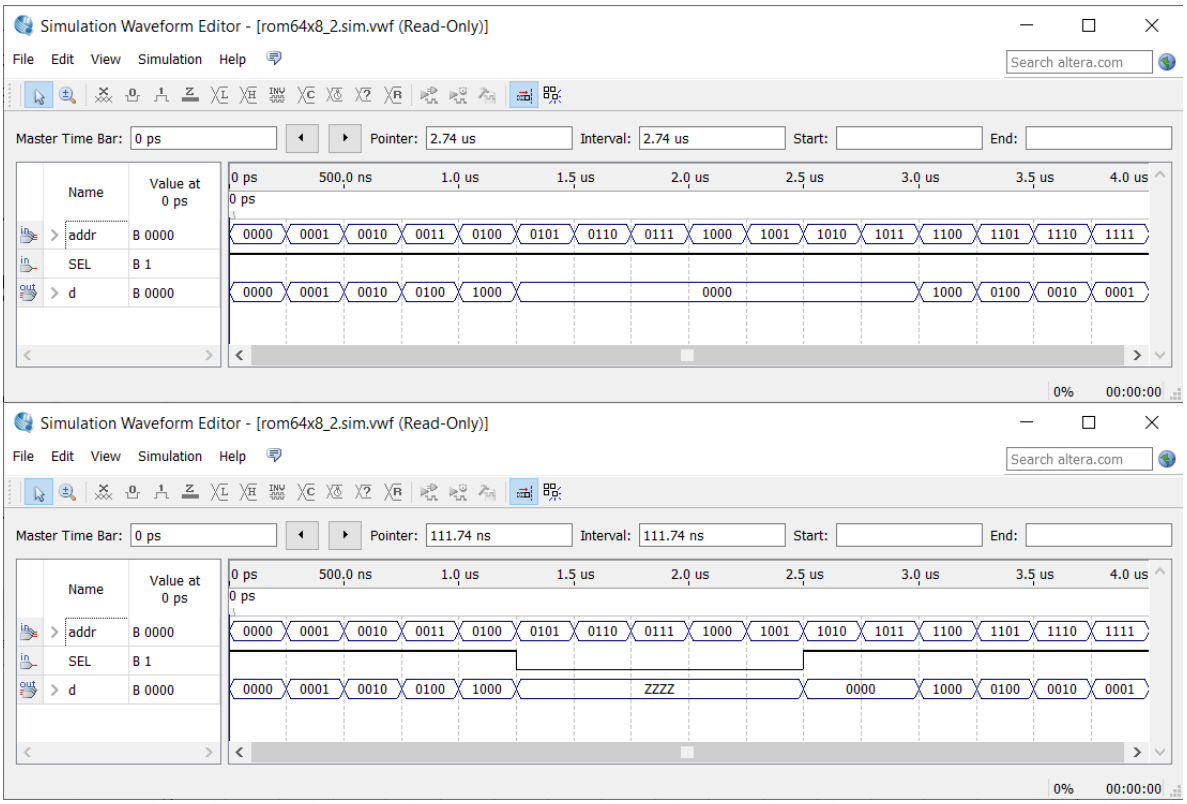


Figura 16: Simulações funcionais do circuito do módulo "rom16x4_2ls". Nela é possível perceber que o sistema se comporta de acordo com o especificado.

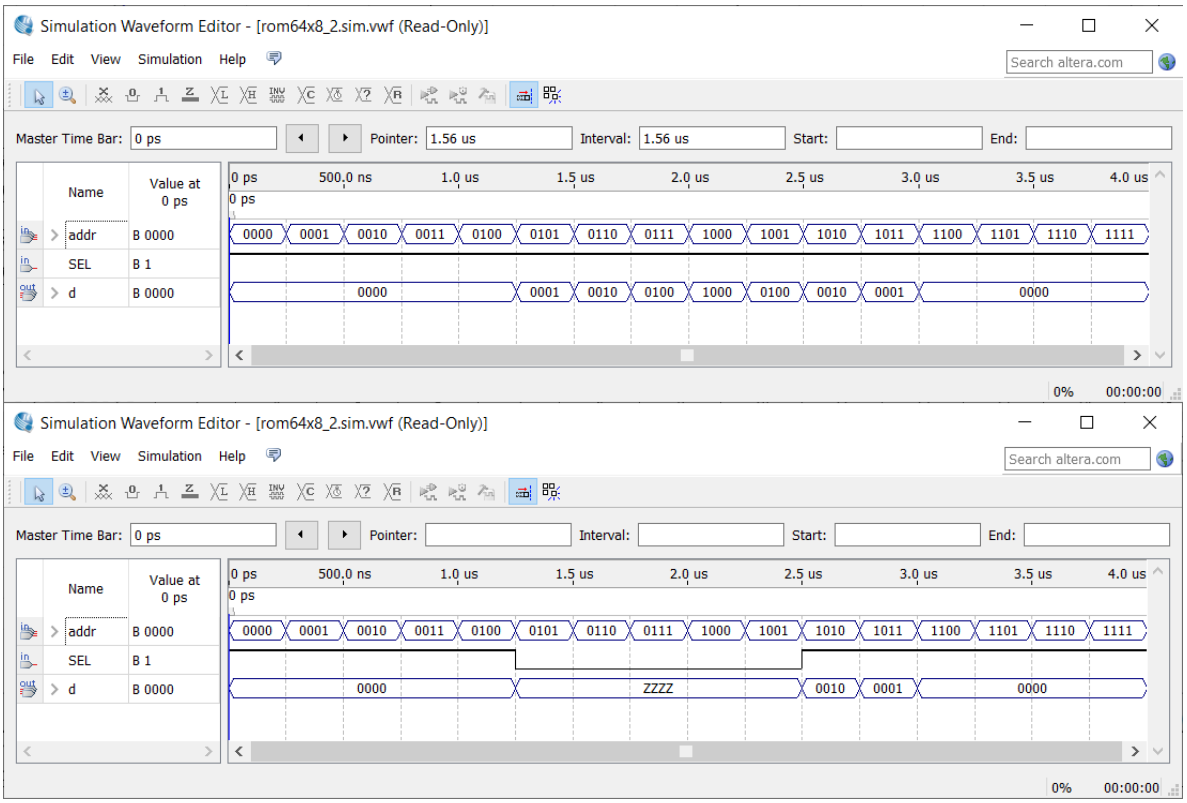


Figura 17: Simulações funcionais do circuito do módulo "rom16x4_2ms". Nela é possível perceber que o sistema se comporta de acordo com o especificado.

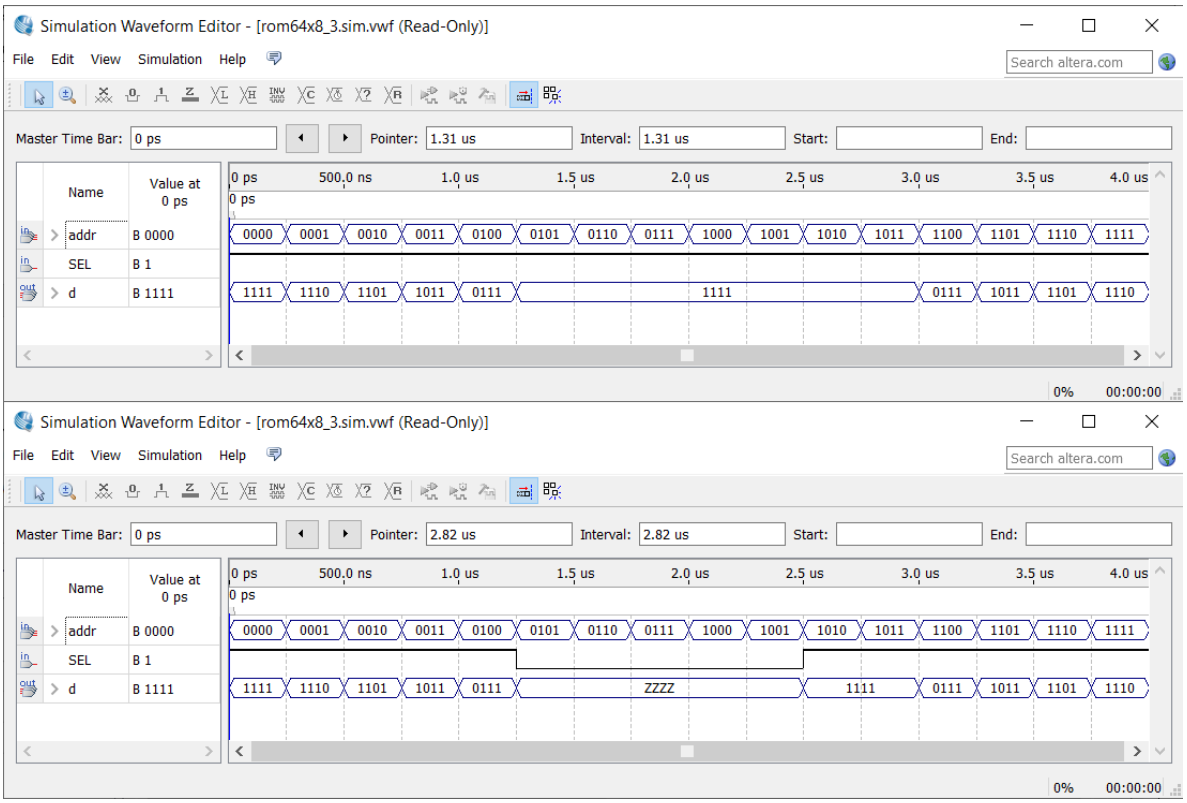


Figura 18: Simulações funcionais do circuito do módulo "rom16x4_3ls". Nela é possível perceber que o sistema se comporta de acordo com o especificado.

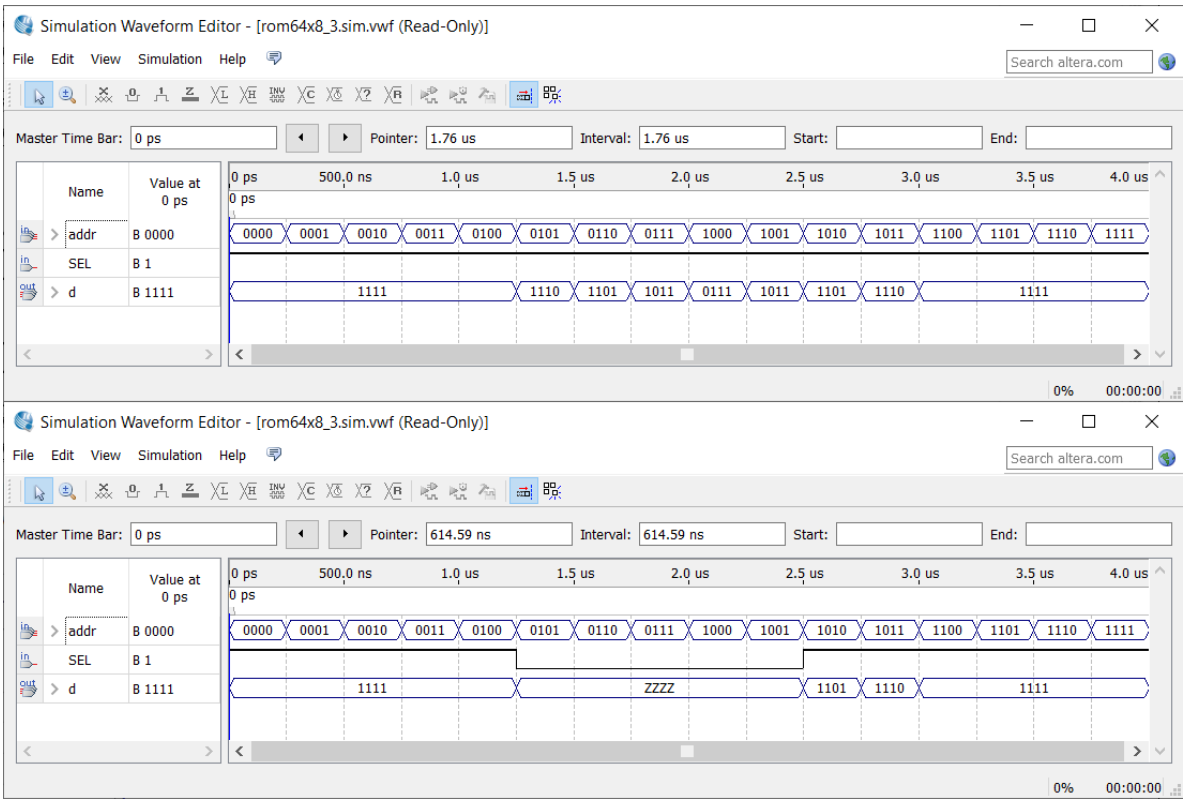


Figura 19: Simulações funcionais do circuito do módulo "rom16x4_3ms". Nela é possível perceber que o sistema se comporta de acordo com o especificado.

2. Projeto 2: ROMs 16x8

2.1. Escopo:

Projeto de circuitos combinacionais que implementem dispositivos de memória não volátil de leitura exclusiva com 16 palavras de memória de 8 bits. Os dispositivos devem ser implementados de modo a conter o conteúdo adequado para a implementação de cada módulo da atividade e devem utilizar dois módulos do tipo ROM 16x4.

2.2. Especificação de alto-nível:

Entradas:
$\underline{addr} = (addr_3, addr_2, addr_1, addr_0)$, com $addr_i \in \{0,1\}$ e $i = 0, \dots, 3$;
$SEL \in \{0,1\}$.

Saída:
$\underline{d} = (d_7, d_6, d_5, d_4, d_3, d_2, d_1, d_0)$, com $d_i \in \{0,1\}$ e $i = 0, \dots, 7$.

Função de Saída:

$$\underline{d} = \begin{cases} [\underline{addr}], & se\ SEL = 1 \\ ZZZZZZZZ, & se\ SEL = 0 \end{cases},$$

onde $[\underline{addr}]$ significa o conteúdo da memória no endereço indicado pela entrada $\underline{addr}$.

2.3. Especificação binária:

Entradas:
$\underline{addr} = (addr_3, addr_2, addr_1, addr_0)$, com $addr_i \in \{0,1\}$ e $i = 0, \dots, 3$;
$SEL \in \{0,1\}$.

Saída:
$\underline{d} = (d_7, d_6, d_5, d_4, d_3, d_2, d_1, d_0)$, com $d_i \in \{0,1\}$ e $i = 0, \dots, 7$.

Função de Saída:

Cada módulo do tipo ROM 16x8 apresenta uma função de saída distinta, a qual depende do conteúdo contido nele. Dessa forma, na tabela abaixo estão apresentadas as relações de conteúdo em função dos endereços utilizada para implementar cada uma das ROM 16x8 empregues nessa atividade.

Módulo "*rom16x8_0*"	
$\underline{addr}$	$\underline{d}$
0000	00000000
0001	11111111
0010	00000000
0011	11111111
0100	00000000
0101	11111111
0110	00000000
0111	11111111
1000	00000000
1001	11111111
1010	00000000

Módulo "*rom16x8_1*"	
$\underline{addr}$	$\underline{d}$
0000	10101010
0001	01010101
0010	10101010
0011	01010101
0100	10101010
0101	01010101
0110	10101010
0111	01010101
1000	10101010
1001	01010101
1010	10101010

Módulo "*rom16x8_2*"	
$\underline{addr}$	$\underline{d}$
0000	00000000
0001	00000001
0010	00000010
0011	00000100
0100	00001000
0101	00010000
0110	00100000
0111	01000000
1000	10000000
1001	01000000
1010	00100000

Módulo "*rom16x8_3*"	
$\underline{addr}$	$\underline{d}$
0000	11111111
0001	11111110
0010	11111101
0011	11111011
0100	11110111
0101	11101111
0110	11011111
0111	10111111
1000	01111111
1001	10111111
1010	11011111

1011	11111111
1100	00000000
1101	11111111
1110	00000000
1111	11111111

1011	01010101
1100	10101010
1101	01010101
1110	10101010
1111	01010101

1011	00010000
1100	00001000
1101	00000100
1110	00000010
1111	00000001

1011	11101111
1100	11110111
1101	11111011
1110	11111101
1111	11111110

Vale lembrar que a saída $\underline{d}$ só assume os valores dados na tabela acima se $SEL = 1$, ficando em estado de alta impedância caso contrário.

2.4. Minimizações:

Como o circuito das unidades de memória ROM 16x8 devem ser implementadas a partir de dois módulos de memória ROM 16x4 já desenvolvidos, não se fazem necessária qualquer minimização. Basta, para cada um dos módulos apresentados acima, escolher um módulo de memória 16x4 para representar os 4 bits mais significativos e um para representar os 4 bits menos significativos da palavra de 8 bits. Uma vez feita tal escolha, basta ligar os sinais de $\underline{addr}$ e SEL nas entradas respectivas de ambos os módulos e compor a saída $\underline{d}$ a partir das saídas destes, sendo (d_3, d_2, d_1, d_0) dados pela saída do módulo tido como parte menos significativo e (d_7, d_6, d_5, d_4) dados pela saída do outro módulo. Dessa forma, os módulos de memória ROM 16x8 foram implementados de acordo com o esquema apresentado na tabela abaixo:

Módulo de memória 16x8	Módulo de memória 16x4 mais significativo (MS)	Módulo de memória 16x4 menos significativo (LS)
"rom16x8_0"	*"rom16x4_0"*	*"rom16x4_0"*
"rom16x8_1"	*"rom16x4_1"*	*"rom16x4_1"*
"rom16x8_2"	*"rom16x4_2ms"*	*"rom16x4_2ls"*
"rom16x8_3"	*"rom16x4_3ms"*	*"rom16x4_3ls"*

2.5. Esquemático do circuito:

Seguindo o raciocínio apresentado acima, os circuitos dos 4 módulos de memória 16x8 da atividade foram implementados e seus diagramas esquemáticos resultantes estão dispostos nas figuras 20 a 23.

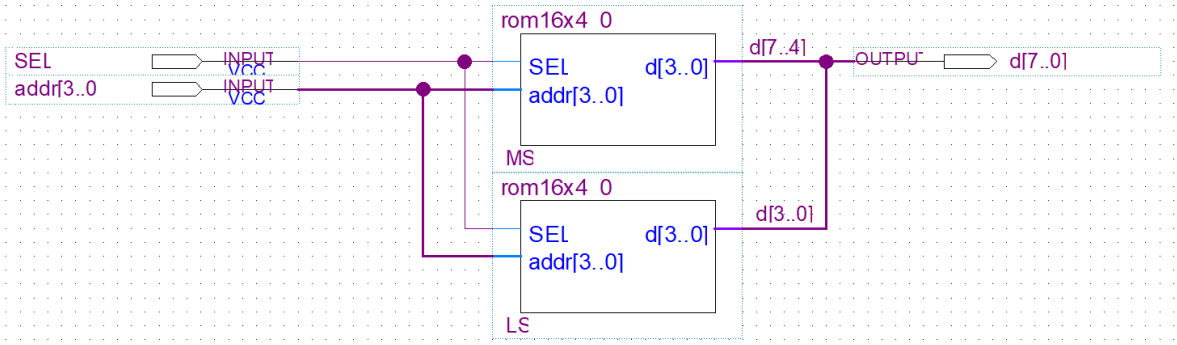


Figura 20: diagrama esquemático do circuito do módulo "rom16x8_0" o qual implementa uma unidade de memória com 16 palavras de 8 bits e constituirá a ROM 64x8 do módulo 0. Este circuito emprega dois símbolos do módulo "rom16x4_0" já apresentado anteriormente para compor tanto a parte mais significativa quanto a parte menos significativa de suas palavras de memória. Isto é possível pois o conteúdo de ambas as partes (MS e LS) para esse módulo é igual em todos os endereços.

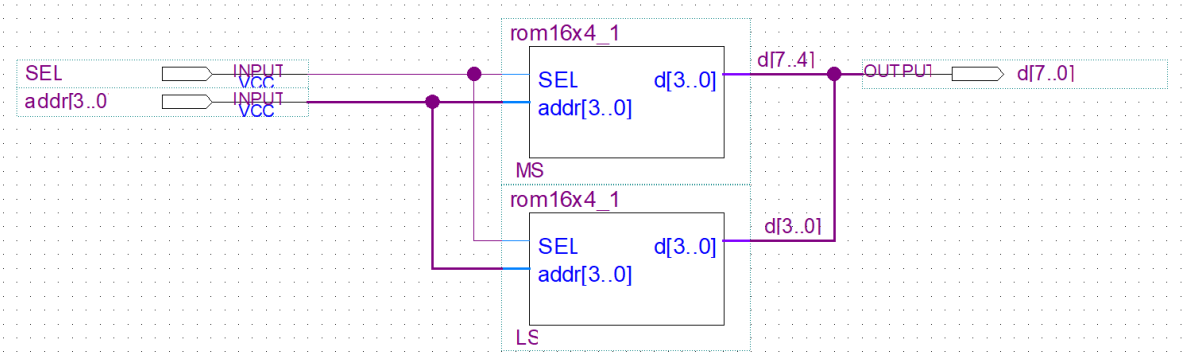


Figura 21: diagrama esquemático do circuito do módulo "rom16x8_1" o qual implementa uma unidade de memória com 16 palavras de 8 bits e constituirá a ROM 64x8 do módulo 1. Este circuito emprega dois símbolos do módulo "rom16x4_1" já apresentado anteriormente para compor tanto a parte mais significativa quanto a parte menos significativa de suas palavras de memória. Isto é possível pois o conteúdo de ambas as partes (MS e LS) para esse módulo é igual em todos os endereços.

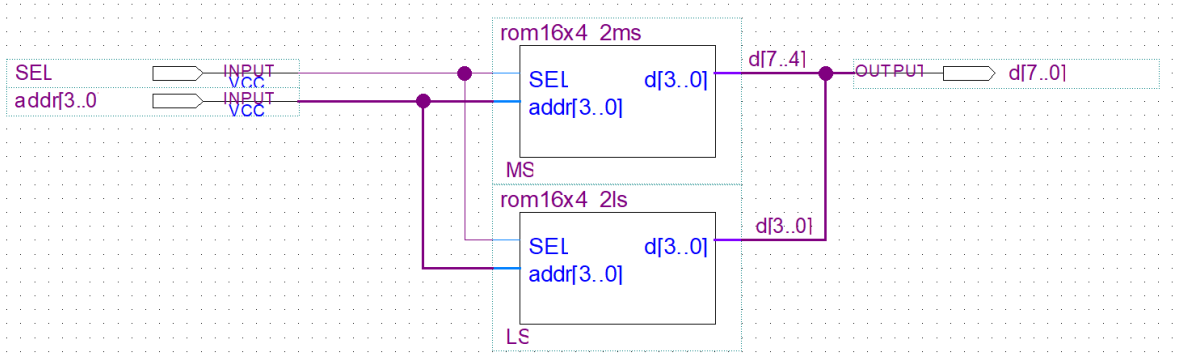


Figura 22: diagrama esquemático do circuito do módulo "rom16x8_2" o qual implementa uma unidade de memória com 16 palavras de 8 bits e constituirá a ROM 64x8 do módulo 2. Este circuito emprega o módulo "rom16x4_2ls" para compor a parte menos significativa de sua palavra de memória e o módulo "rom16x4_2ms" para compor a parte mais significativa de sua palavra de memória.

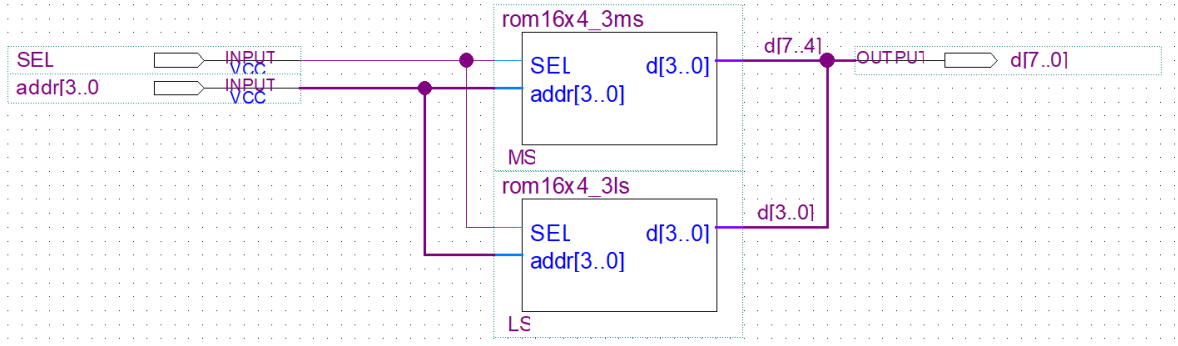


Figura 23: diagrama esquemático do circuito do módulo "rom16x8_3" o qual implementa uma unidade de memória com 16 palavras de 8 bits e constituirá a ROM 64x8 do módulo 3. Este circuito emprega o módulo "rom16x4_3ls" para compor a parte menos significativa de sua palavra de memória e o módulo "rom16x4_3ms" para compor a parte mais significativa de sua palavra de memória.

2.6. Simulações:

Para testar as memórias construídas, simulações funcionais acessando cada endereço disponível, bem como avaliando seu comportamento mediante a alteração da entrada SEL, de cada módulo implementado foram realizadas. Os resultados obtidos a partir delas estão dispostos nas figuras 24 a 27.

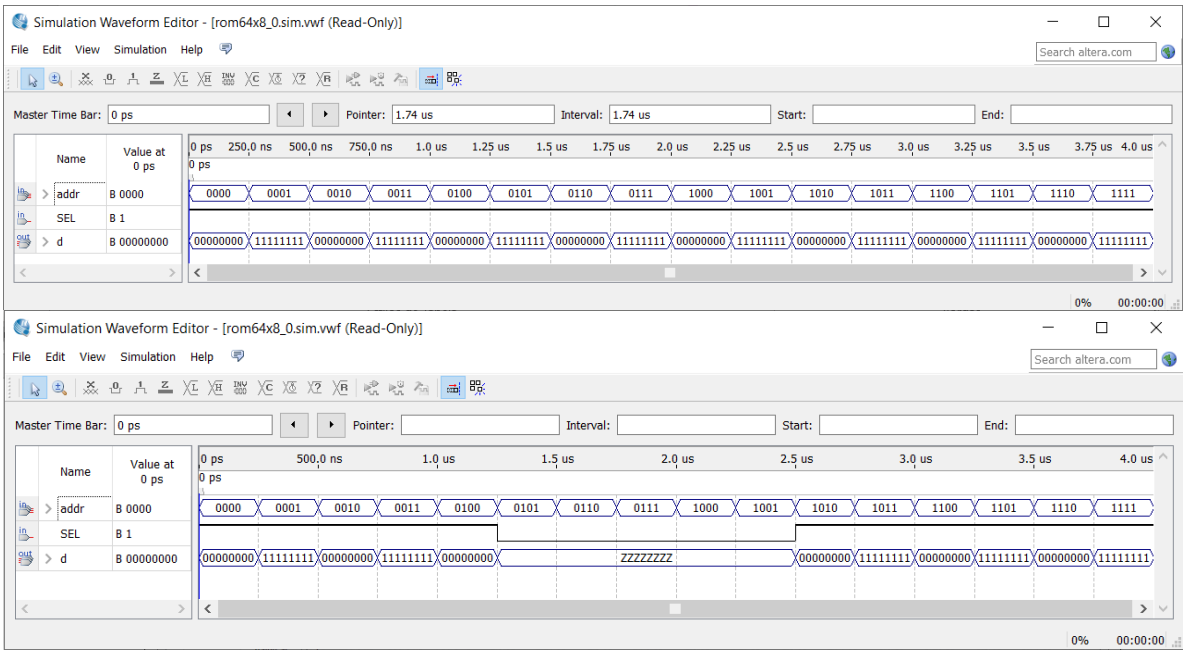


Figura 24: simulação funcional do módulo "rom16x8_0". Nela, nota-se que o sistema se comporta de acordo com o especificado.

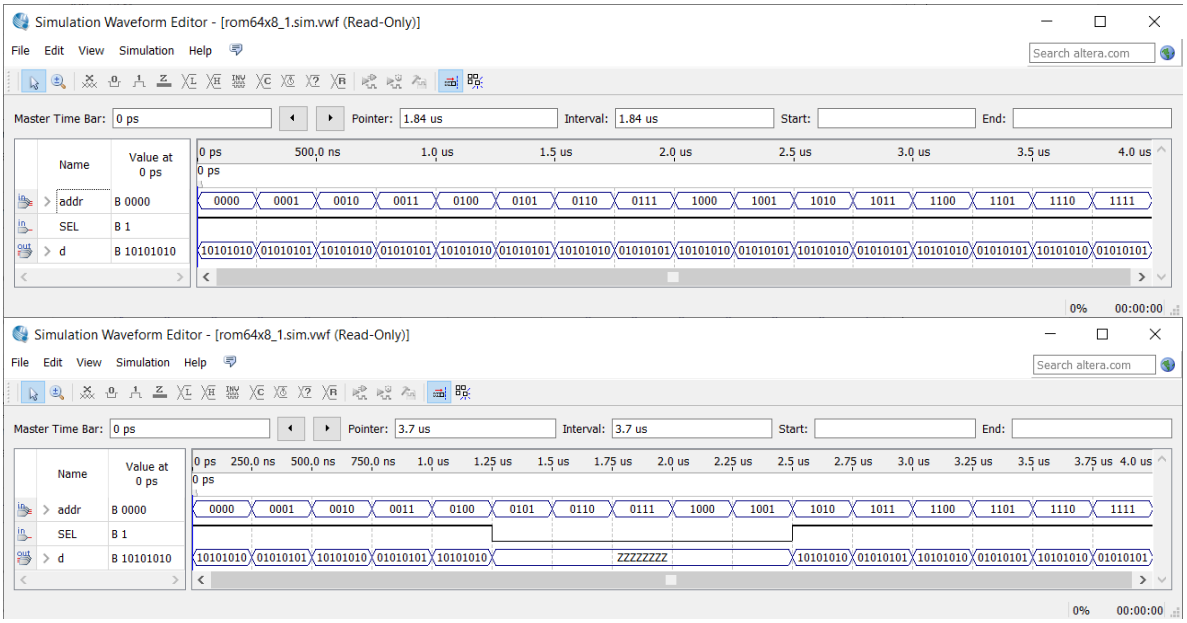


Figura 25: simulação funcional do módulo "rom16x8_1". Nela, nota-se que o sistema se comporta de acordo com o especificado.

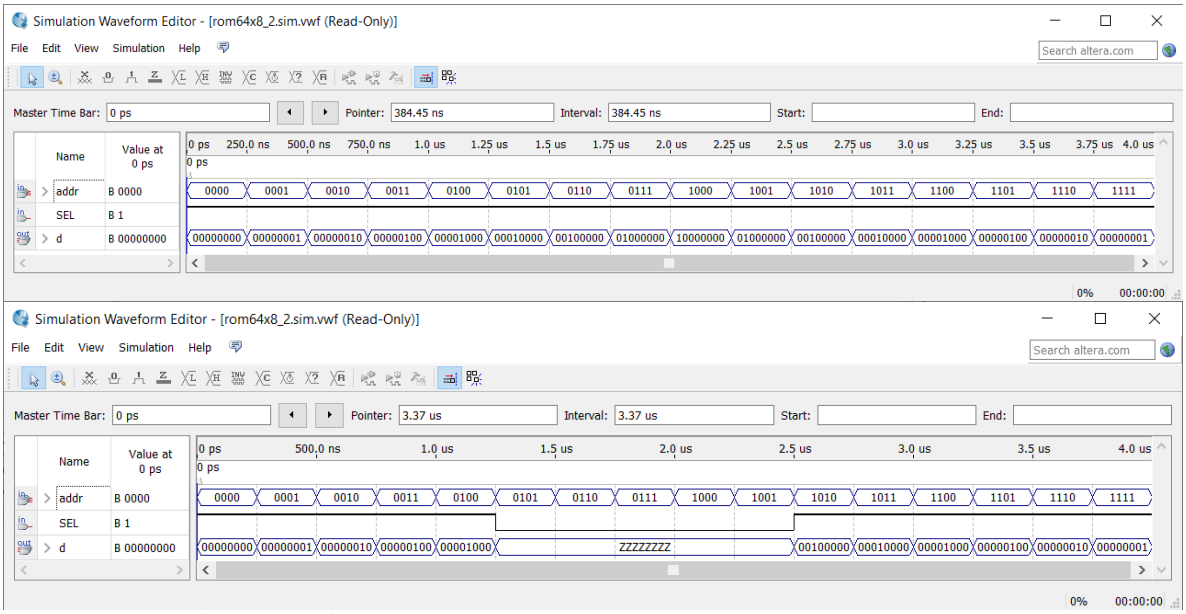


Figura 26: simulação funcional do módulo "rom16x8_2". Nela, nota-se que o sistema se comporta de acordo com o especificado.

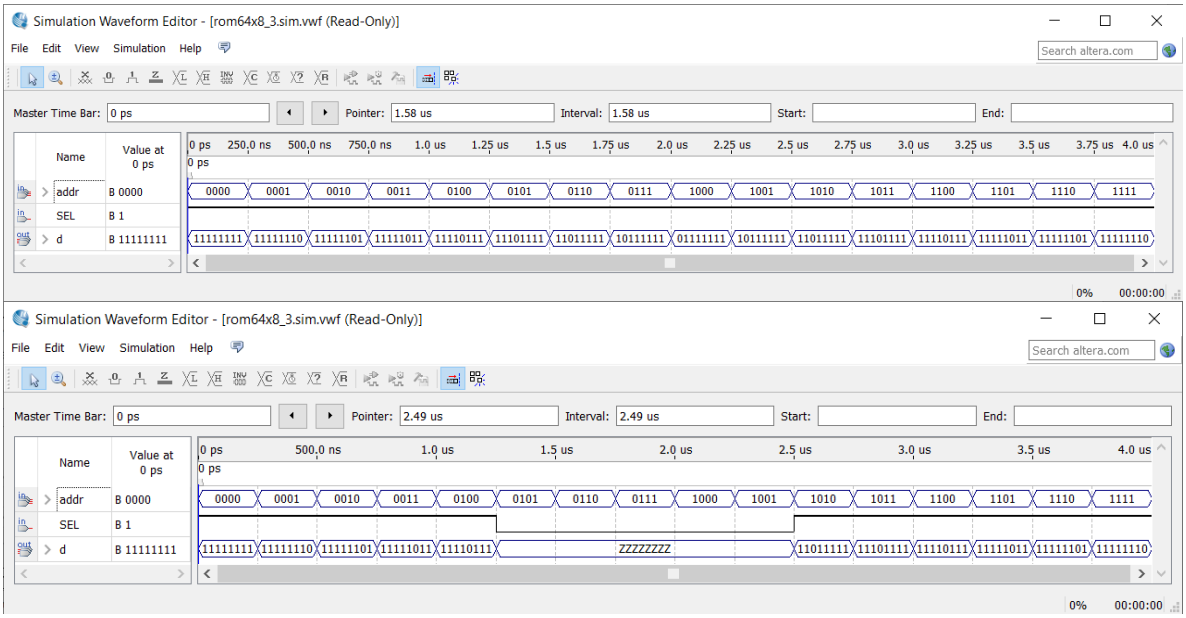


Figura 27: simulação funcional do módulo "rom16x8_3". Nela, nota-se que o sistema se comporta de acordo com o especificado.

3. Projeto 3: ROMs 32x8

3.1. Escopo:

Projeto de circuitos combinacionais que implementem dispositivos de memória não volátil de leitura exclusiva com 32 palavras de memória de 8 bits. Os dispositivos devem ser implementados de modo a conter o conteúdo adequado para a implementação de cada módulo da atividade e devem utilizar dois módulos do tipo ROM 16x8.

3.2. Especificação de alto-nível:

Entradas:
$\underline{addr} = (addr_4, addr_3, addr_2, addr_1, addr_0)$, com $addr_i \in \{0,1\}$ e $i = 0, \dots, 4$;
$SEL \in \{0,1\}$.

Saída:
$\underline{d} = (d_7, d_6, d_5, d_4, d_3, d_2, d_1, d_0)$, com $d_i \in \{0,1\}$ e $i = 0, \dots, 7$.

Função de Saída:

$$\underline{d} = \begin{cases} [\underline{addr}], & se\ SEL = 1 \\ ZZZZZZZZ, & se\ SEL = 0 \end{cases}$$

onde $[\underline{addr}]$ significa o conteúdo da memória no endereço indicado pela entrada $\underline{addr}$.

3.3. Especificação binária:

Entradas:
$\underline{addr} = (addr_4, addr_3, addr_2, addr_1, addr_0)$, com $addr_i \in \{0,1\}$ e $i = 0, \dots, 4$;
$SEL \in \{0,1\}$.

Saída:
$\underline{d} = (d_7, d_6, d_5, d_4, d_3, d_2, d_1, d_0)$, com $d_i \in \{0,1\}$ e $i = 0, \dots, 7$.

Função de Saída:

Cada módulo do tipo ROM 32x8 apresenta uma função de saída distinta, a qual depende do conteúdo contido nele. Dessa forma, serão implementados os módulos *"rom32x8_0"*, *"rom32x8_1"*, *"rom32x8_2"* e *"rom32x8_3"*, os quais apresentam, para os primeiros 16 endereços de memória, o mesmo conteúdo que as unidades de memória ROM 16x8 com nome correspondente e, para os demais endereços, repetições cíclicas deste conteúdo.

3.4. Minimizações:

Como o circuito das unidades de memória ROM 32x8 devem ser implementadas a partir de dois módulos de memória ROM 16x8 já desenvolvidos, será necessária criar uma lógica para que uma das memórias menores sirva como os primeiros 16 endereços da memória maior (endereços 0 a 15) e a outra sirva como os últimos 16 endereços (endereços 16 a 31). Tal lógica pode ser implementada a partir dos sinais de entrada SEL e $addr_4$ acionando o sinal de seleção dos módulos internos (SEL_{0a15} para a memória com a primeira metade dos endereços e SEL_{16a31} para a outra), de acordo com a tabela verdade abaixo:

SEL	$addr_4$	SEL_{0a15}	SEL_{16a31}
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

$$SEL_{0a15} = SEL \cdot addr_4{}'$$
e
$$SEL_{16a31} = SEL \cdot addr_4$$

É válido ressaltar que é possível minimizar as expressões acima a partir da própria tabela verdade, pois somente uma das combinações da entrada gera um valor alto em cada uma das duas saídas. Assim, basta ligar os bits restantes do sinal $\underline{addr}$, isto é,

($addr_3, addr_2, addr_1, addr_0$) nas entradas respectivas de ambos os módulos e ligar a saída destes ao sinal de saída $\underline{d}$.

3.5. Esquemático do circuito:

Seguindo o raciocínio anterior, os circuitos dos 4 módulos de memória 32x8 da atividade foram implementados e seus diagramas esquemáticos resultantes estão dispostos nas figuras 28 a 31.

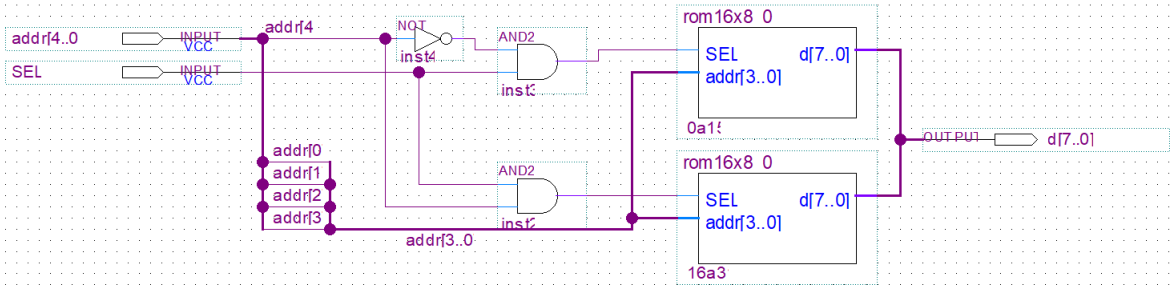


Figura 28: diagrama esquemático do circuito do módulo "rom32x8_0" o qual implementa uma unidade de memória com 32 palavras de 8 bits e constituirá a ROM 64x8 do módulo 0. Este circuito emprega dois símbolos do módulo "rom16x8_0" já apresentado anteriormente para compor as duas metades dos 32 endereços necessários.

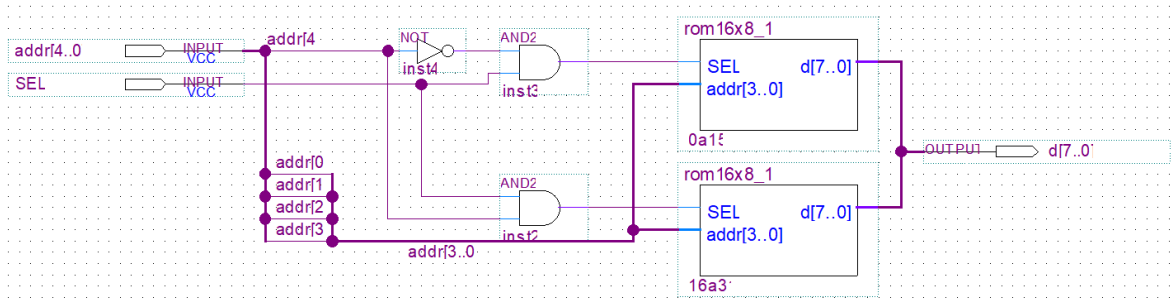


Figura 29: diagrama esquemático do circuito do módulo "rom32x8_1" o qual implementa uma unidade de memória com 32 palavras de 8 bits e constituirá a ROM 64x8 do módulo 1. Este circuito emprega dois símbolos do módulo "rom16x8_1" já apresentado anteriormente para compor as duas metades dos 32 endereços necessários.

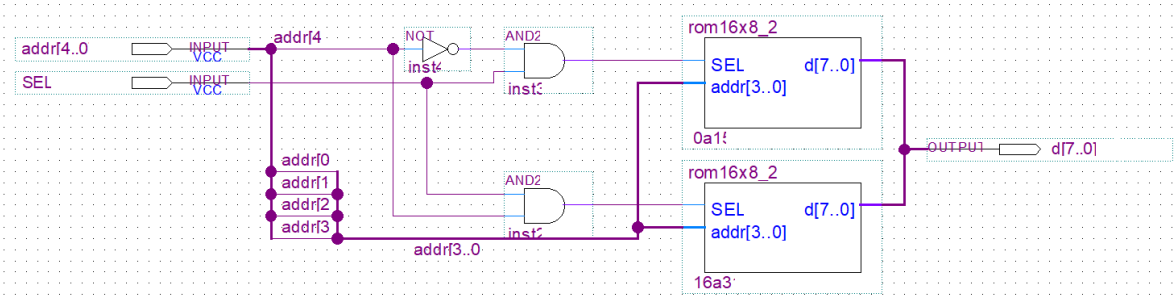


Figura 30: diagrama esquemático do circuito do módulo "rom32x8_2" o qual implementa uma unidade de memória com 32 palavras de 8 bits e constituirá a ROM 64x8 do módulo 2. Este circuito emprega dois símbolos do módulo "rom16x8_2" já apresentado anteriormente para compor as duas metades dos 32 endereços necessários.

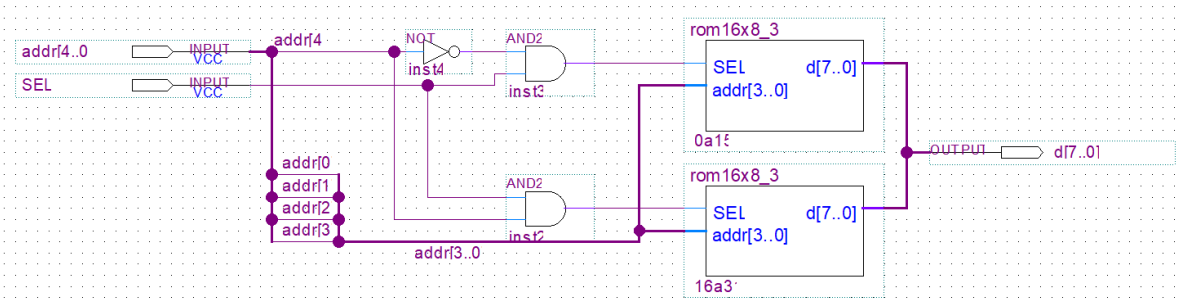


Figura 31: diagrama esquemático do circuito do módulo "rom32x8_3" o qual implementa uma unidade de memória com 32 palavras de 8 bits e constituirá a ROM 64x8 do módulo 3. Este circuito emprega dois símbolos do módulo "rom16x8_3" já apresentado anteriormente para compor as duas metades dos 32 endereços necessários.

3.6. Simulações:

Para testar as memórias construídas, simulações funcionais acessando cada endereço disponível, bem como avaliando seu comportamento mediante a alteração da entrada *SEL*, de cada módulo implementado foram realizadas. Os resultados obtidos a partir delas estão dispostos nas figuras 32 a 35.

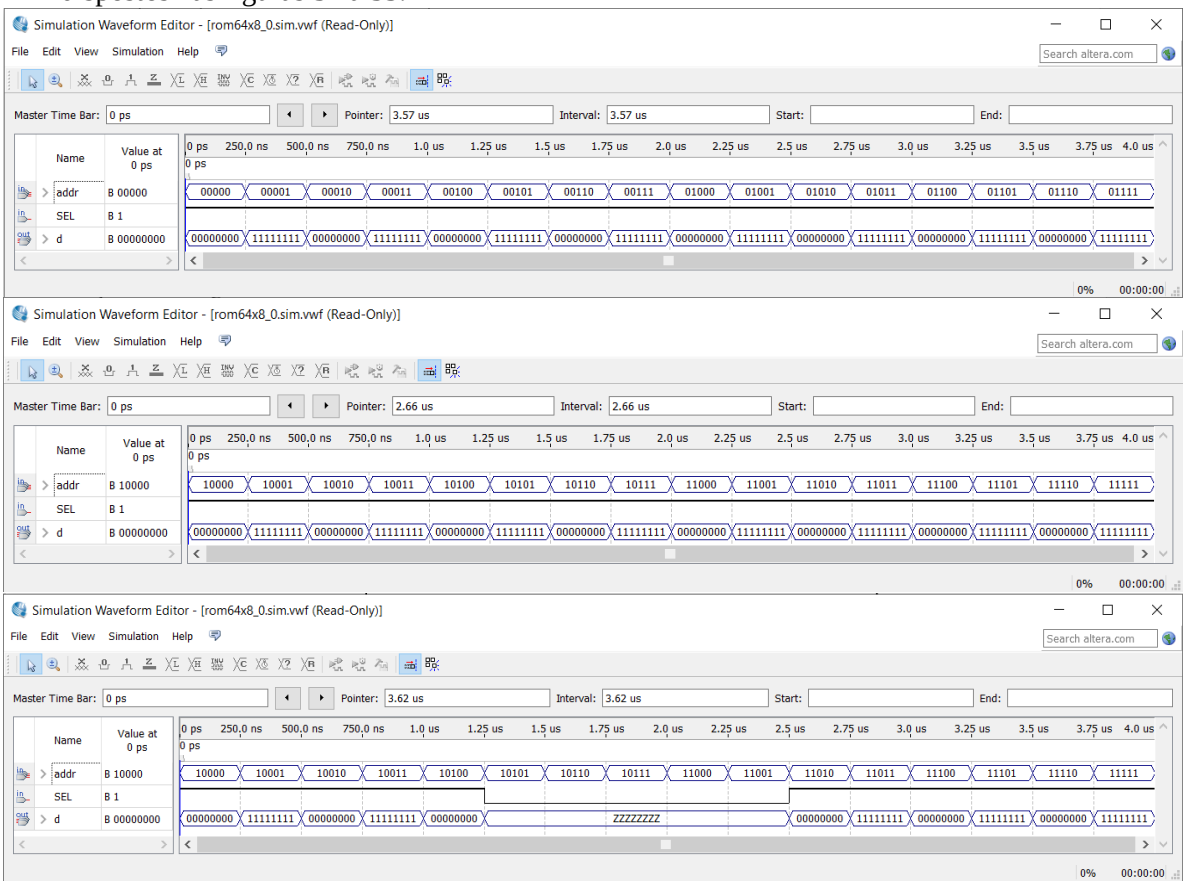


Figura 32: simulação funcional do módulo "rom32x8_0". Nela, nota-se que o sistema se comporta de acordo com o especificado.

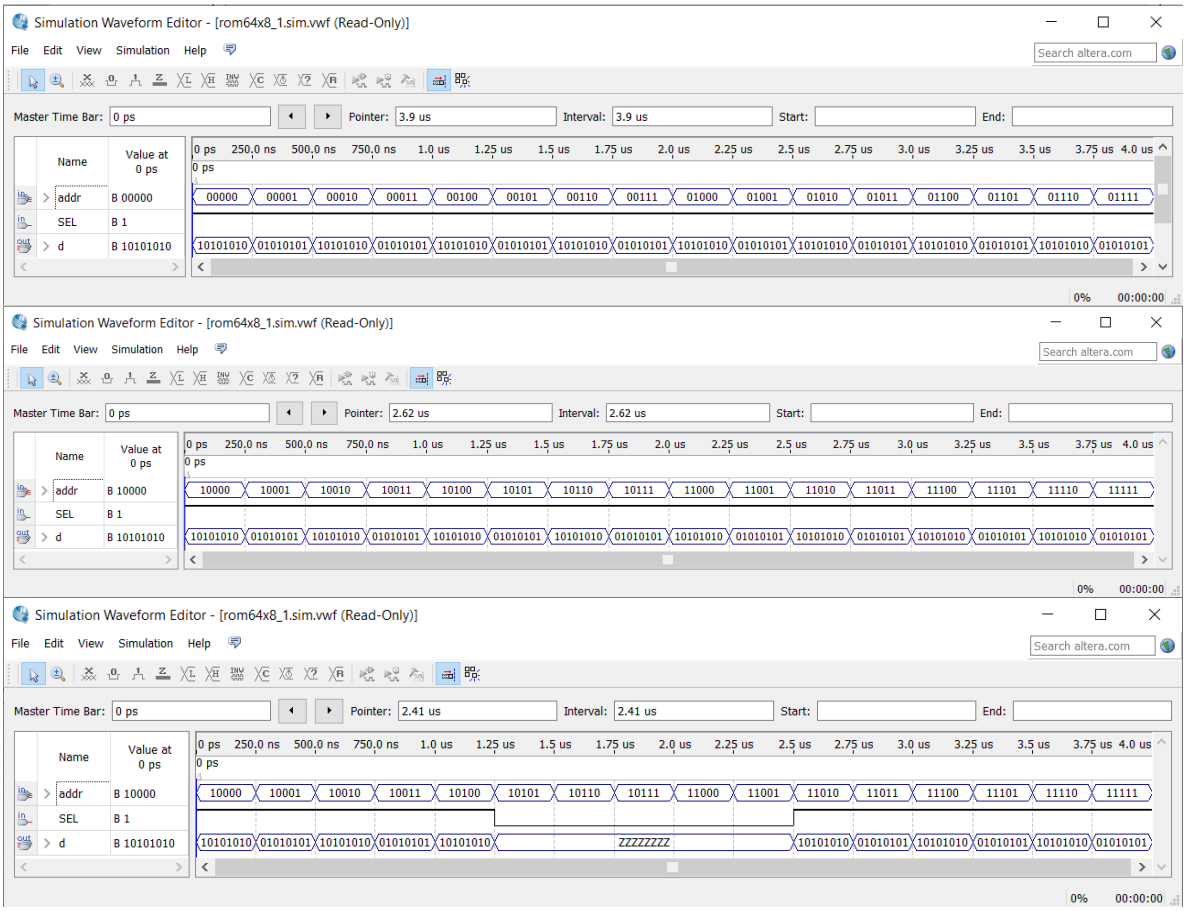
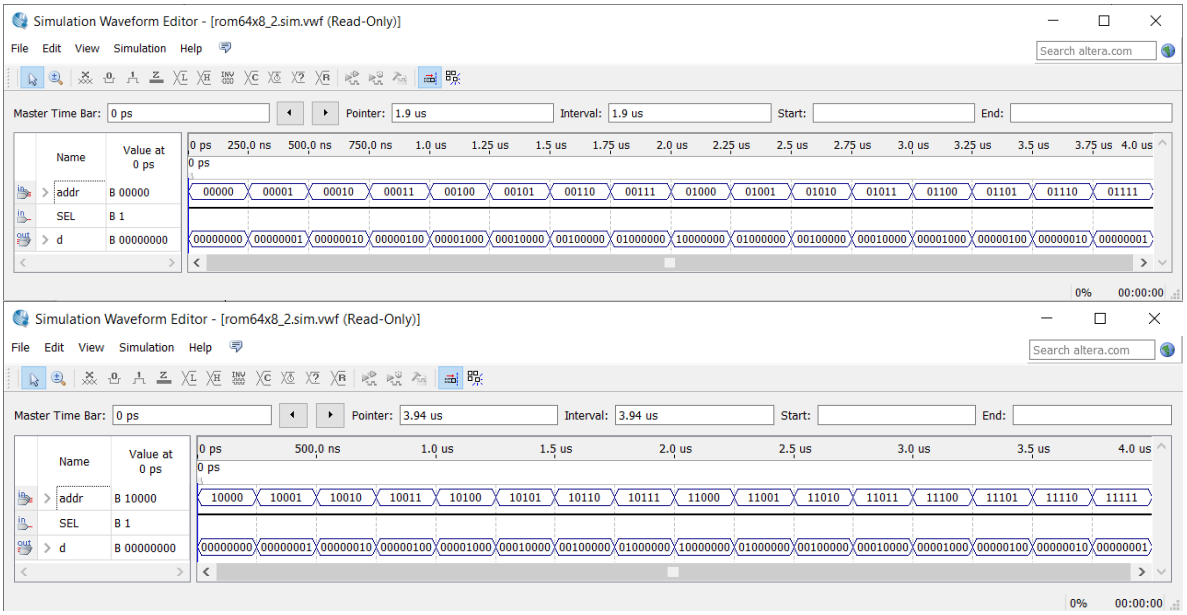


Figura 33: simulação funcional do módulo "rom32x8_1". Nela, nota-se que o sistema se comporta de acordo com o especificado.



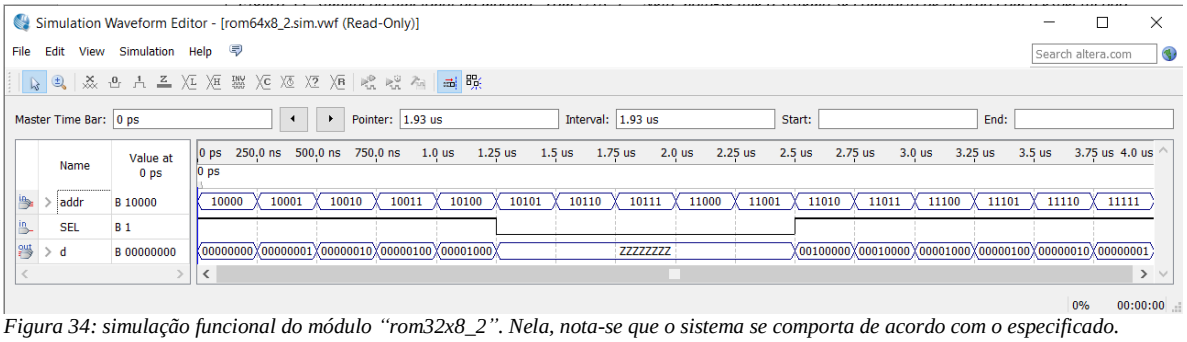


Figura 34: simulação funcional do módulo "rom32x8_2". Nela, nota-se que o sistema se comporta de acordo com o especificado.

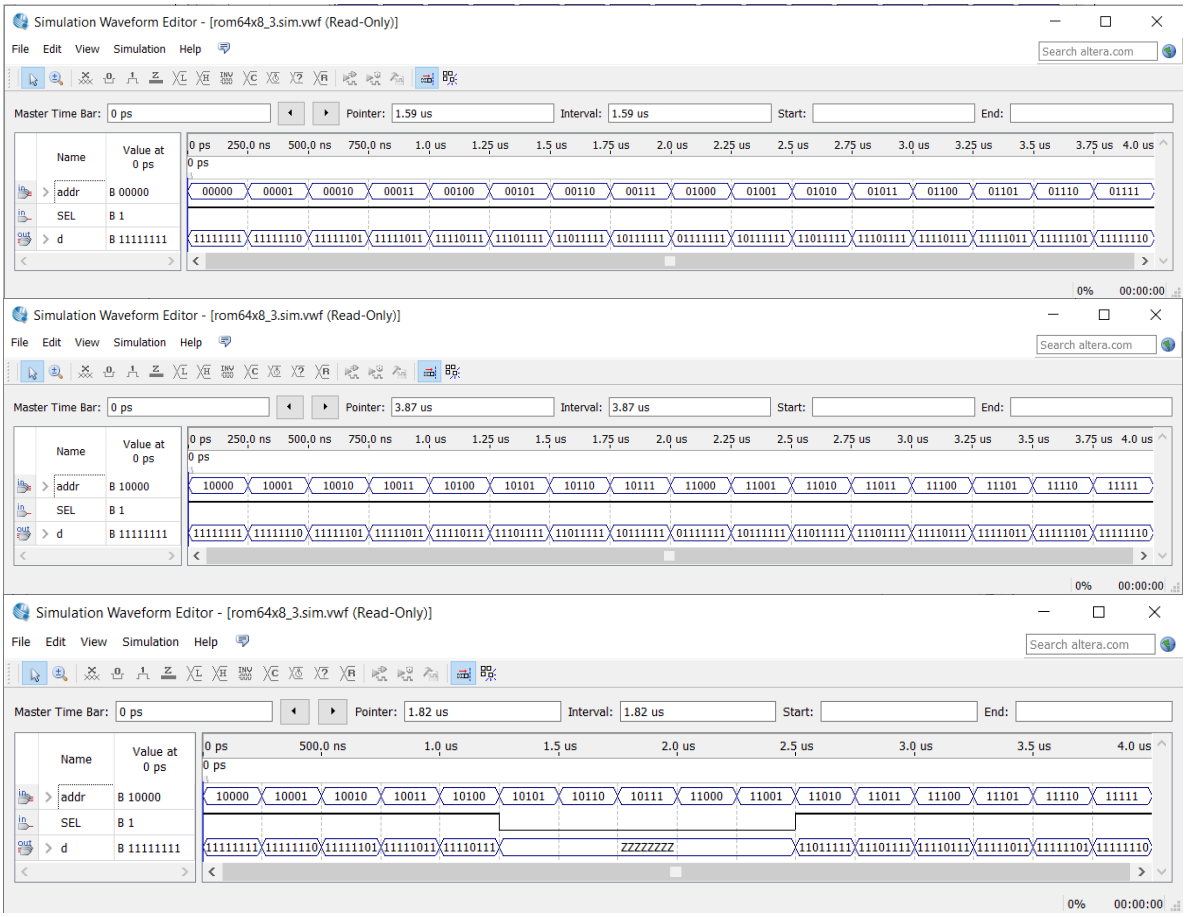


Figura 35: simulação funcional do módulo "rom32x8_3". Nela, nota-se que o sistema se comporta de acordo com o especificado.

4. Projeto 4: ROMs 64x8

4.1. Escopo:

Projeto de circuitos combinacionais que implementem dispositivos de memória não volátil de leitura exclusiva com 64 palavras de memória de 8 bits. Os dispositivos devem ser implementados de modo a conter o conteúdo adequado para a implementação de cada módulo da atividade e devem utilizar dois módulos do tipo ROM 32x8.

4.2. Especificação de alto-nível:

Entradas:
$\underline{addr} = (addr_5, addr_4, addr_3, addr_2, addr_1, addr_0)$, com $addr_i \in \{0,1\}$ e $i = 0, \dots, 5$;
$SEL \in \{0,1\}$.

Saída:
$\underline{d} = (d_7, d_6, d_5, d_4, d_3, d_2, d_1, d_0)$, com $d_i \in \{0,1\}$ e $i = 0, \dots, 7$.

Função de Saída:
$$\underline{d} = \begin{cases} [\underline{addr}], & se\ SEL = 1 \\ ZZZZZZZZ, & se\ SEL = 0 \end{cases}$$
onde $[\underline{addr}]$ significa o conteúdo da memória no endereço indicado pela entrada $\underline{addr}$.

4.3. Especificação binária:

Entradas:
$\underline{addr} = (addr_4, addr_3, addr_2, addr_1, addr_0)$, com $addr_i \in \{0,1\}$ e $i = 0, \dots, 4$;
$SEL \in \{0,1\}$.

Saída:
$\underline{d} = (d_7, d_6, d_5, d_4, d_3, d_2, d_1, d_0)$, com $d_i \in \{0,1\}$ e $i = 0, \dots, 7$.

Função de Saída:

Cada módulo do tipo ROM 64x8 apresenta uma função de saída distinta, a qual depende do conteúdo contido nele. Dessa forma, serão implementados os módulos *"rom64x8_0"*, *"rom64x8_1"*, *"rom64x8_2"* e *"rom64x8_3"*, os quais apresentam, para os primeiros 32 endereços de memória, o mesmo conteúdo que as unidades de memória ROM 32x8 com nome correspondente e, para os demais endereços, repetições cíclicas deste conteúdo.

4.4. Minimizações:

Como o circuito das unidades de memória ROM 64x8 devem ser implementadas a partir de dois módulos de memória ROM 32x8 já desenvolvidos, será necessária criar uma lógica para que uma das memórias menores sirva como os primeiros 32 endereços da memória maior (endereços 0 a 31) e a outra sirva como os últimos 32 endereços (endereços 31 a 63). Tal lógica pode ser implementada a partir dos sinais de entrada SEL e $addr_5$ acionando o sinal de seleção dos módulos internos (SEL_{0a31} para a memória com a primeira metade dos endereços e SEL_{32a63} para a outra), de acordo com a tabela verdade abaixo:

SEL	$addr_5$	SEL_{0a31}	SEL_{32a63}
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

$$SEL_{0a31} = SEL \cdot addr_5'$$
e
$$SEL_{31a63} = SEL \cdot addr_5$$

Assim, basta ligar os bits restantes do sinal $\underline{addr}$, isto é, $(addr_4, addr_3, addr_2, addr_1, addr_0)$ nas entradas respectivas de ambos os módulos e ligar a saída destes ao sinal de saída $\underline{d}$.

4.5. Esquemático do circuito:

Seguindo a lógica apresentada anteriormente, os circuitos dos 4 módulos de memória 64x8 da atividade foram implementados e seus diagramas esquemáticos resultantes estão dispostos nas figuras 36 a 39.

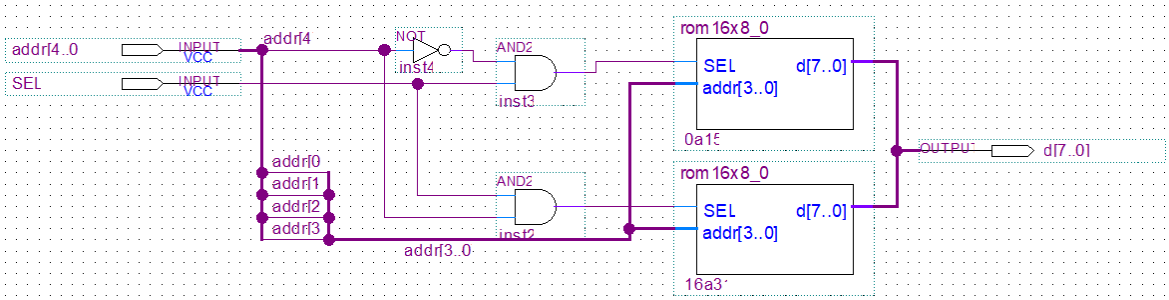


Figura 36: diagrama esquemático do circuito do módulo "rom64x8_0" o qual implementa uma unidade de memória com 64 palavras de 8 bits. Esta é o dispositivo de memória final do módulo 0, o qual constituirá uma parte do circuito do projeto "rom_4blocks". Este circuito emprega dois símbolos do módulo "rom32x8_0" já apresentado anteriormente para compor as duas metades dos 64 endereços necessários.

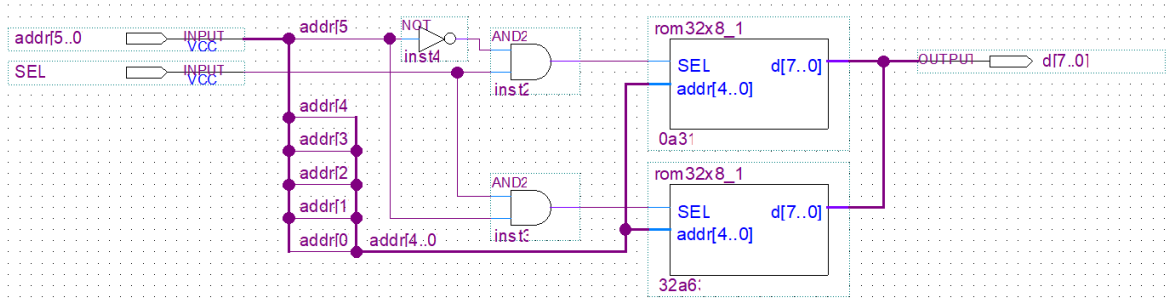


Figura 37: diagrama esquemático do circuito do módulo "rom64x8_1" o qual implementa uma unidade de memória com 64 palavras de 8 bits. Esta é o dispositivo de memória final do módulo 1, o qual constituirá uma parte do circuito do projeto "rom_4blocks". Este circuito emprega dois símbolos do módulo "rom32x8_1" já apresentado anteriormente para compor as duas metades dos 64 endereços necessários.

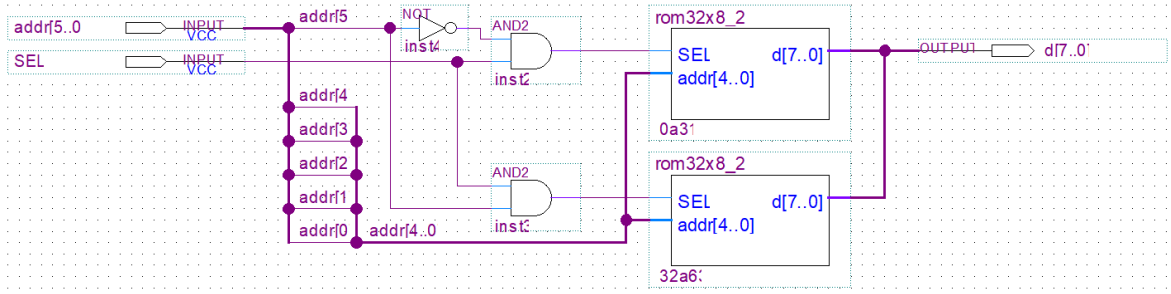


Figura 38: diagrama esquemático do circuito do módulo "rom64x8_2" o qual implementa uma unidade de memória com 64 palavras de 8 bits. Esta é o dispositivo de memória final do módulo 2, o qual constituirá uma parte do circuito do projeto "rom_4blocks". Este circuito emprega dois símbolos do módulo "rom32x8_2" já apresentado anteriormente para compor as duas metades dos 64 endereços necessários.

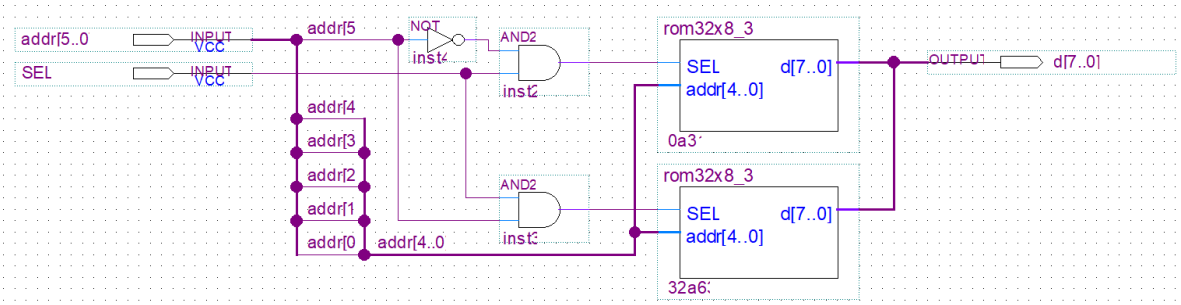
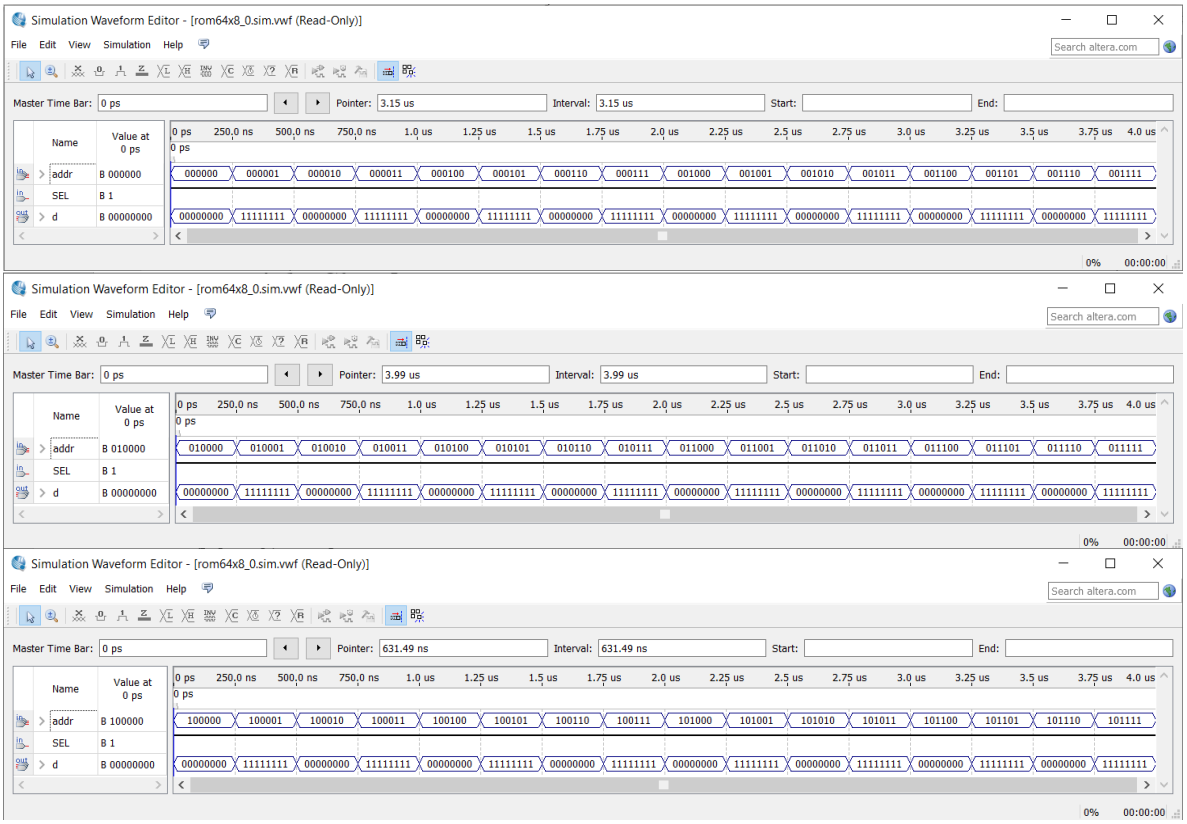


Figura 39: diagrama esquemático do circuito do módulo "rom64x8_3" o qual implementa uma unidade de memória com 64 palavras de 8 bits. Esta é o dispositivo de memória final do módulo 3, o qual constituirá uma parte do circuito do projeto "rom_4blocks". Este circuito emprega dois símbolos do módulo "rom32x8_3" já apresentado anteriormente para compor as duas metades dos 64 endereços necessários.

4.6. Simulações:

Para testar as memórias construídas, simulações funcionais acessando cada endereço disponível, bem como avaliando seu comportamento mediante a alteração da entrada *SEL*, de cada módulo implementado foram realizadas. Os resultados obtidos a partir delas estão dispostos nas figuras 40 a 43.



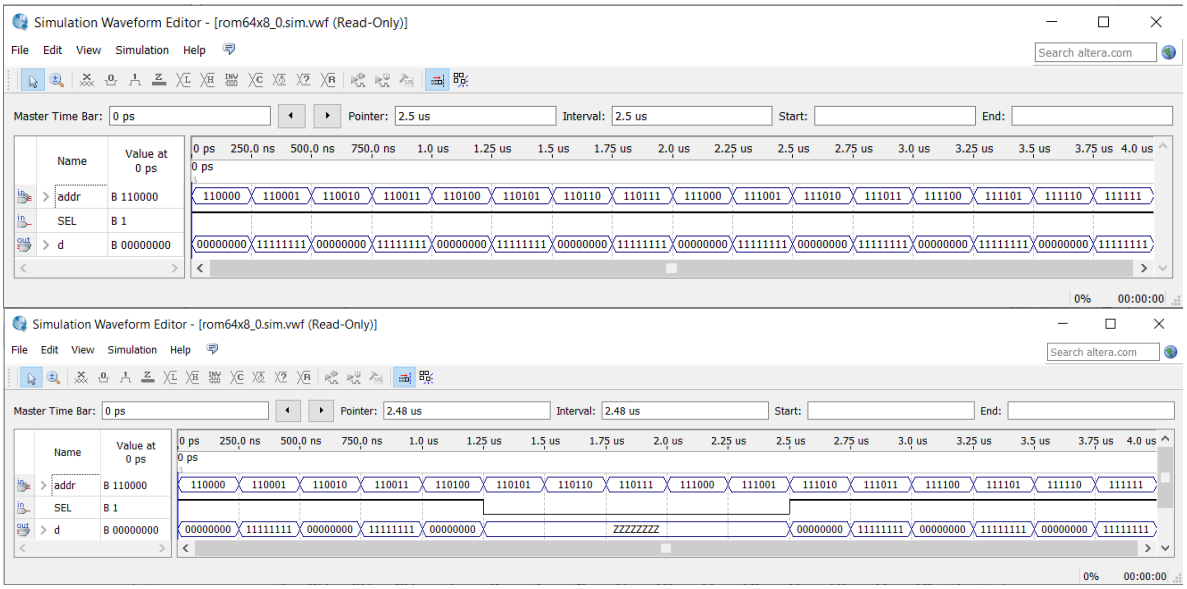
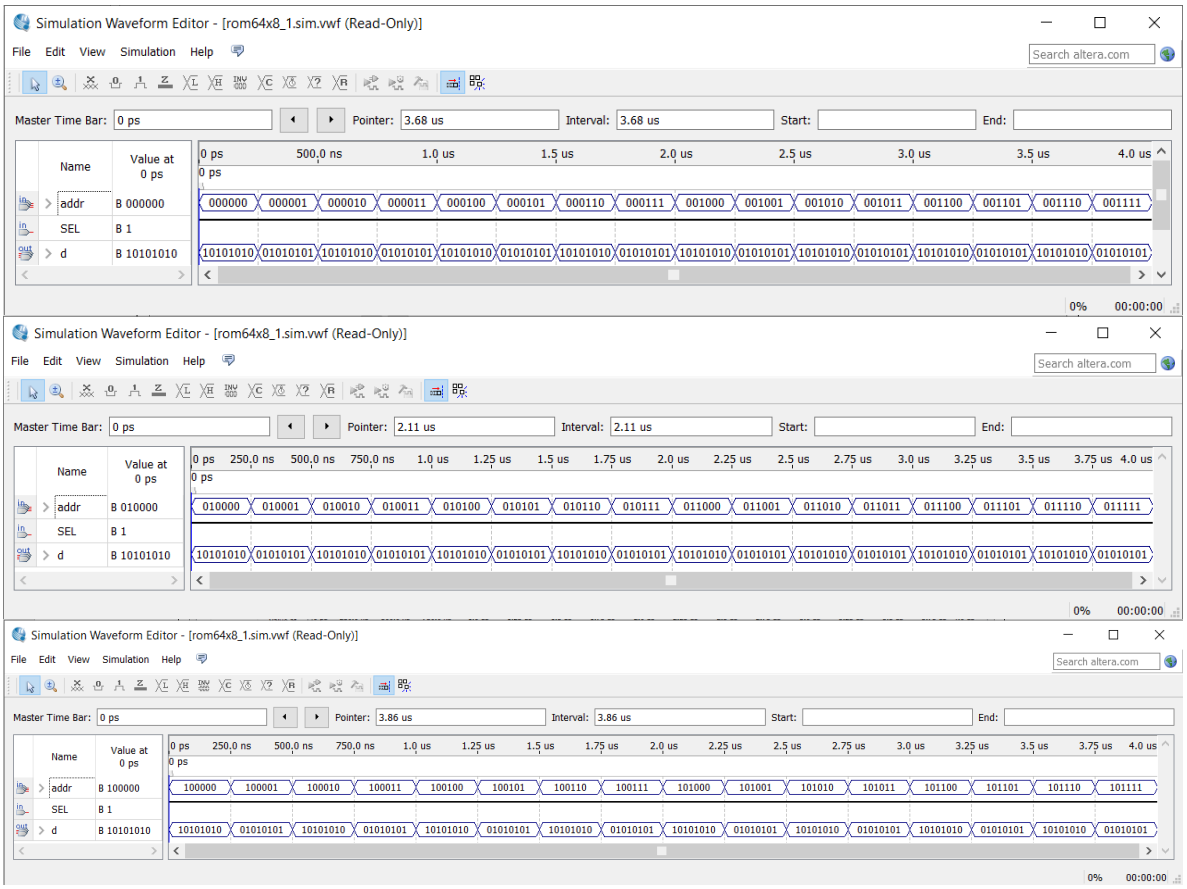


Figura 40: simulação funcional do módulo "rom64x8_0". Nela, nota-se que o sistema se comporta de acordo com o especificado.



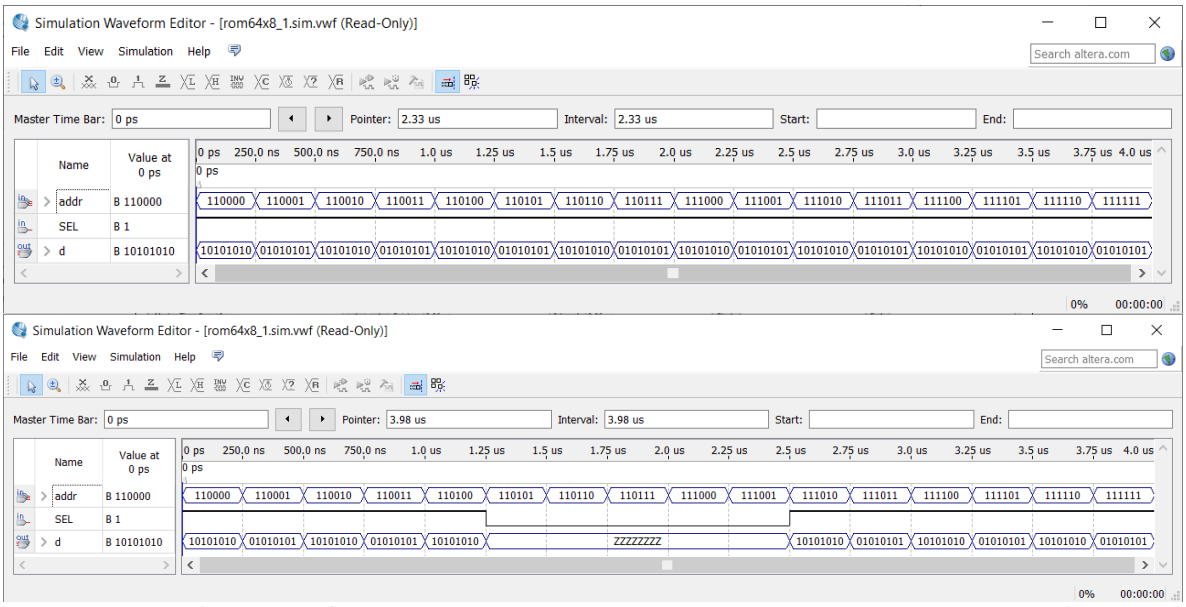
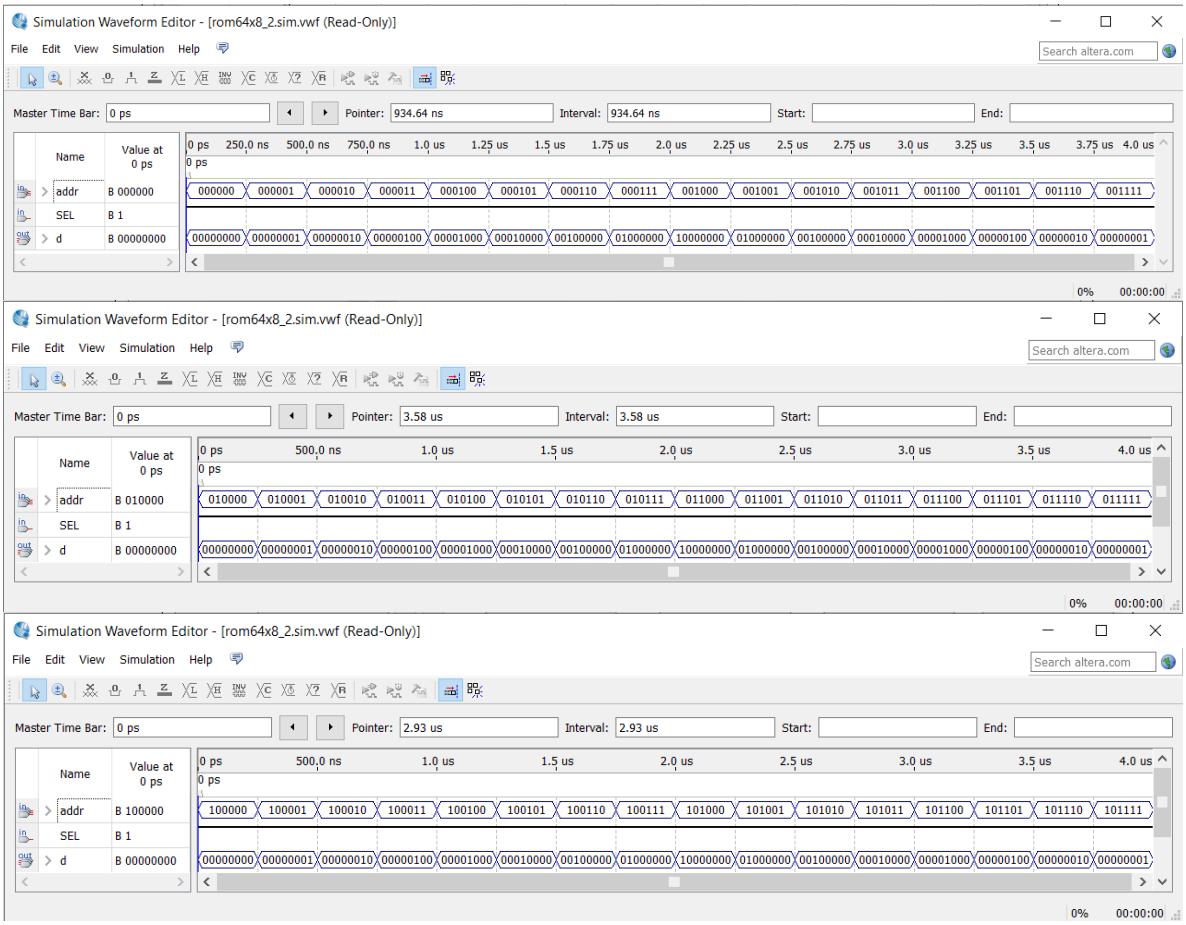


Figura 41: simulação funcional do módulo "rom64x8_1". Nela, nota-se que o sistema se comporta de acordo com o especificado.



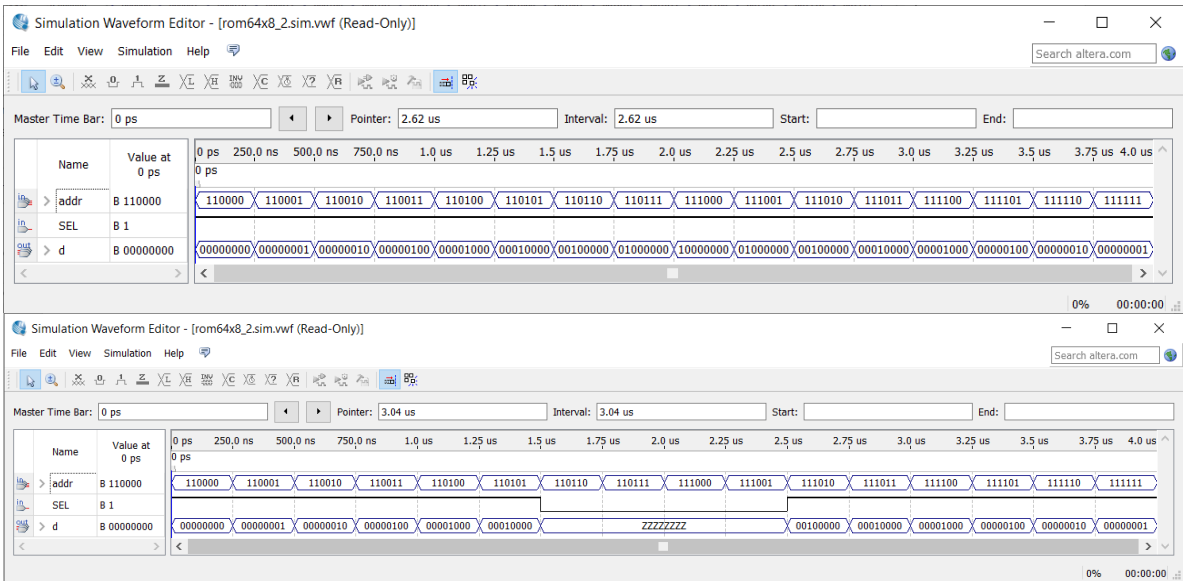
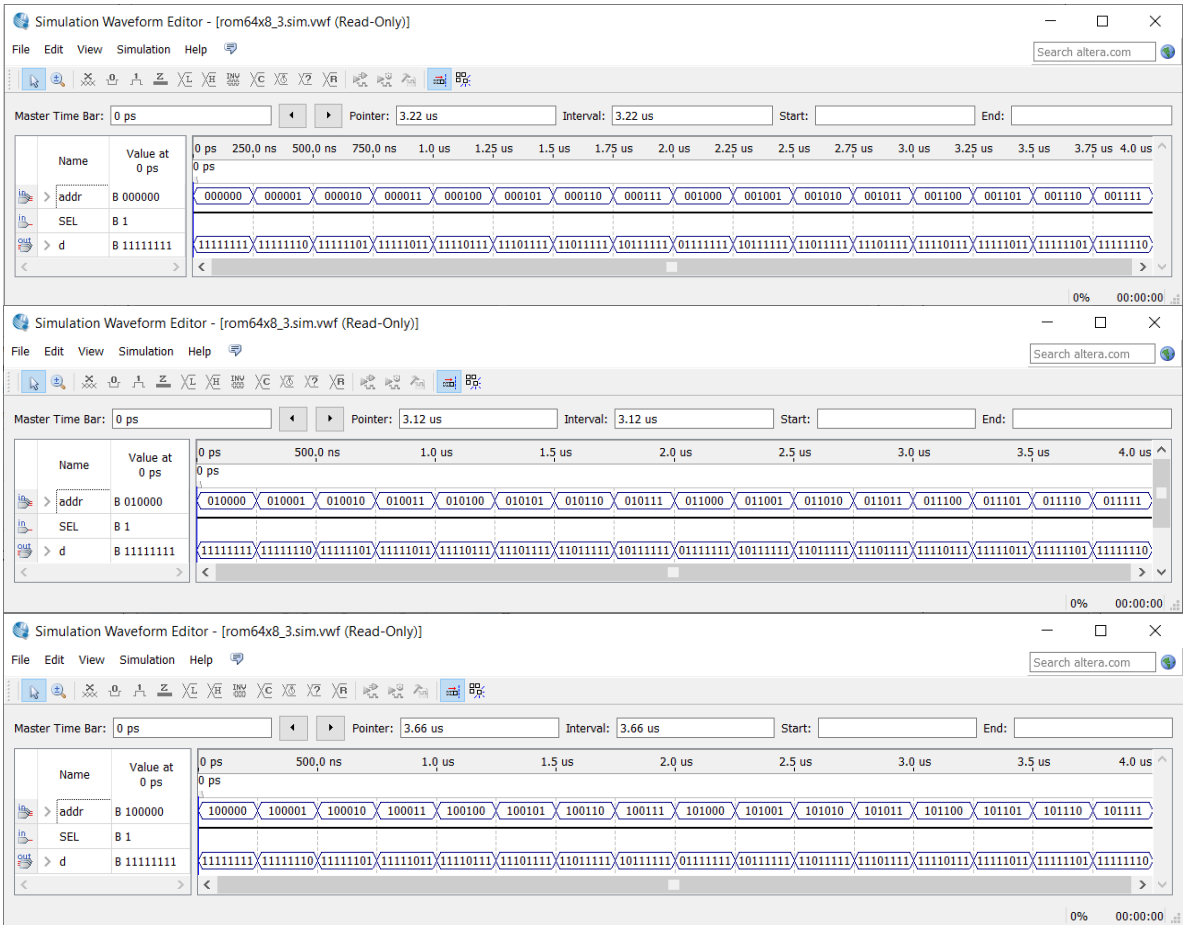


Figura 42: simulação funcional do módulo "rom64x8_2". Nela, nota-se que o sistema se comporta de acordo com o especificado.



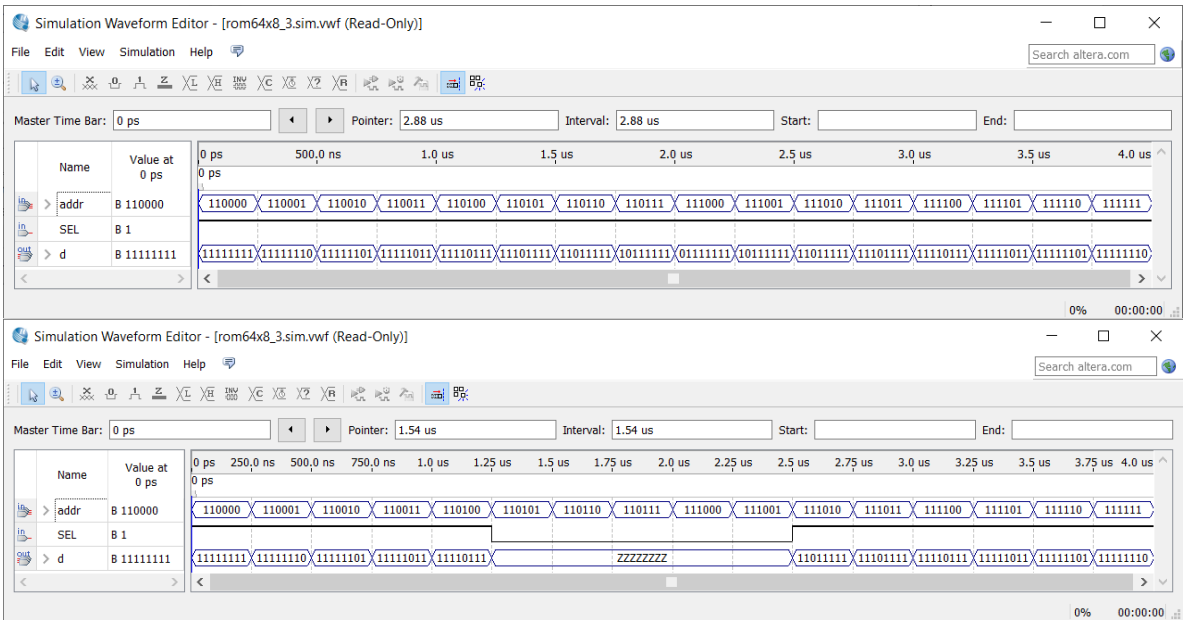


Figura 43: simulação funcional do módulo "rom64x8_3". Nela, nota-se que o sistema se comporta de acordo com o especificado.

5. Projeto 5: Quatro módulos de ROM 64x8 selecionáveis individualmente

5.1. Escopo:

Projeto de um circuito combinacional que reúna os quatro módulos de memória ROM 64x8 desenvolvidos até agora em um único sistema, tornando-os acessíveis a partir de uma entrada seletora que especifica qual bloco deve fornecer seu conteúdo à saída. A seleção da palavra de memória a ser utilizada deve continuar sendo dada a partir de uma entrada de endereços.

5.2. Especificação de alto-nível:

Entradas:

$\underline{addr} = (addr_5, addr_4, addr_3, addr_2, addr_1, addr_0)$, com $addr_i \in \{0,1\}$ e $i = 0, \dots, 5$;

$\underline{BLCK} = (BLCK_1, BLCK_0)$, com $BLCK_i \in \{0,1\}$ e $i = 0, \dots, 1$;

$SEL \in \{0,1\}$.

Saída:

$\underline{d} = (d_7, d_6, d_5, d_4, d_3, d_2, d_1, d_0)$, com $d_i \in \{0,1\}$ e $i = 0, \dots, 7$.

Função de Saída:

$$\underline{d} = \begin{cases} [\underline{addr}]_0, & se\ SEL = 1\ e\ \underline{BLCK} = 00 \\ [\underline{addr}]_1, & se\ SEL = 1\ e\ \underline{BLCK} = 01 \\ [\underline{addr}]_2, & se\ SEL = 1\ e\ \underline{BLCK} = 10, \\ [\underline{addr}]_3, & se\ SEL = 1\ e\ \underline{BLCK} = 11 \\ ZZZZZZZZ, & se\ SEL = 0 \end{cases}$$

onde $[\underline{addr}]_k$ significa o conteúdo do módulo k de memória no endereço indicado pela entrada $\underline{addr}$.

5.3. Especificação binária:

Entradas:

$\underline{addr} = (addr_5, addr_4, addr_3, addr_2, addr_1, addr_0)$, com $addr_i \in \{0,1\}$ e $i = 0, \dots, 5$;

$\underline{BLCK} = (BLCK_1, BLCK_0)$, com $BLCK_i \in \{0,1\}$ e $i = 0, \dots, 1$;

$SEL \in \{0,1\}$.

Saída:

$\underline{d} = (d_7, d_6, d_5, d_4, d_3, d_2, d_1, d_0)$, com $d_i \in \{0,1\}$ e $i = 0, \dots, 7$.

Função de Saída:

Como o circuito deste projeto deve ser implementado usando dos 4 módulos do tipo ROM 64x8 desenvolvidos até então, a função de saída dele será igual as funções de saída dos módulos já implementados. A única diferença é que a saída dependerá dos sinais de entrada $\underline{BLCK}$, o qual atuará como multiplexador, e SEL, que atuará como *enable* do multiplexador .

5.4. Minimizações:

A partir do raciocínio apresentado anteriormente, sendo os sinais de entrada de ativação dos módulos *"rom64x8_0", "rom64x8_1", "rom64x8_2"* e *"rom64x8_3"* respectivamente iguais a SEL_0, SEL_1, SEL_2 e SEL_3, teremos a seguinte tabela verdade:

SEL	$BLCK_1$	$BLCK_0$	SEL_3	SEL_2	SEL_1	SEL_0
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

$SEL_0 = SEL \cdot BLCK_1' \cdot BLCK_0'$;

$SEL_1 = SEL \cdot BLCK_1' \cdot BLCK_0$;

$SEL_2 = SEL \cdot BLCK_1 \cdot BLCK_0'$;

$SEL_3 = SEL \cdot BLCK_1 \cdot BLCK_0$;

Novamente, não se faz necessário a construção de mapas de Karnaugh, pois cada saída só é alta para uma única combinação da entrada. Dessa forma, para completar o circuito basta ligar o sinal entrada $\underline{addr}$ na entrada de endereço de cada módulo e conectar as saídas deles em um único barramento que constitui a saída $\underline{d}$.

5.5. Esquemático do circuito:

De acordo com o apresentado acima, o circuito do módulo *"rom_4blocks"* foi implementado. Seu diagrama esquemático está disposto na figura 44.

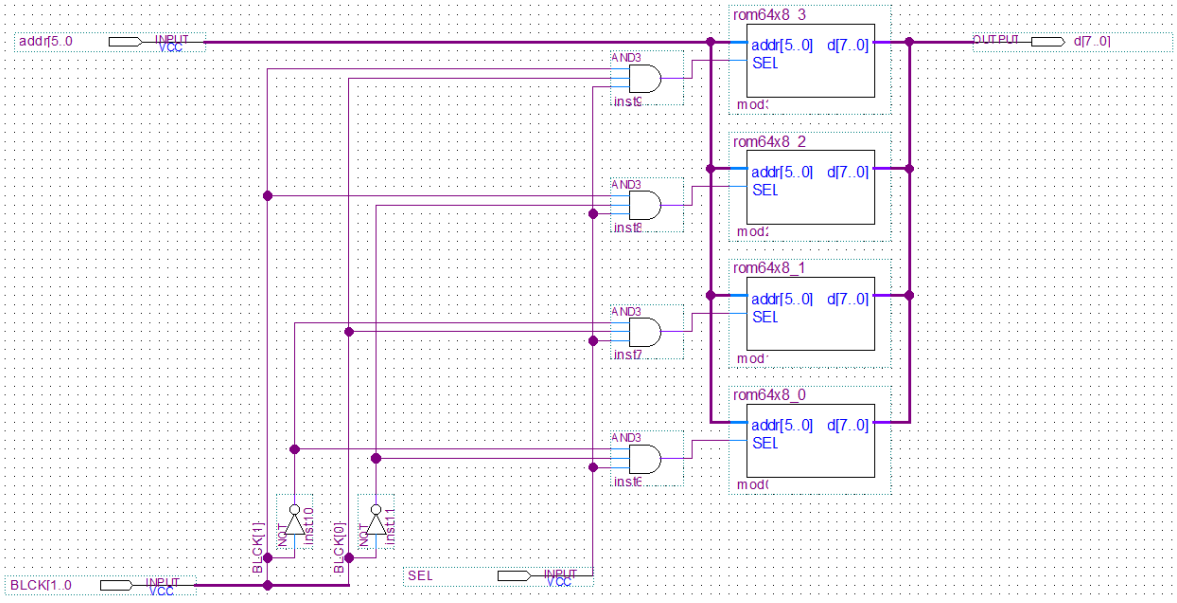
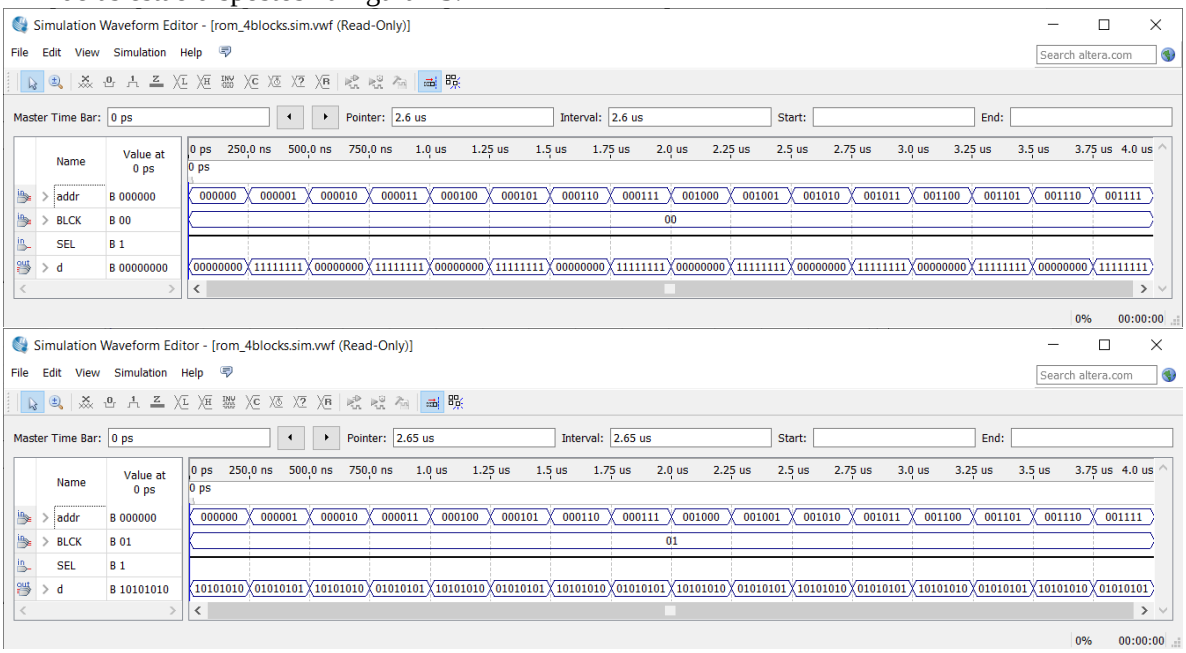


Figura 44: diagrama esquemático do circuito do módulo "rom_4blocks" o qual implementa uma unidade com 4 memórias De 64 palavras de 8 bits selecionáveis e independentes. Este é o circuito final da primeira parte do projeto, o qual utiliza os 4 módulos de memória ROM 64x8 desenvolvidos até então. Este circuito emprega símbolos dos módulos "rom64x8_0", "rom64x8_1", "rom64x8_2" e "rom64x8_3".

5.6. Simulações:

Como todos os módulos usados no circuito já foram testados separadamente e foi comprovado que seu comportamento segue o especificado, o circuito implementado do módulo *"rom_4blocks"* passou por simulações funcionais somente para comprovar que a multiplexação do sinal seletor estava se dando de forma correta. Os resultados obtidos a partir delas estão dispostos na figura 45.



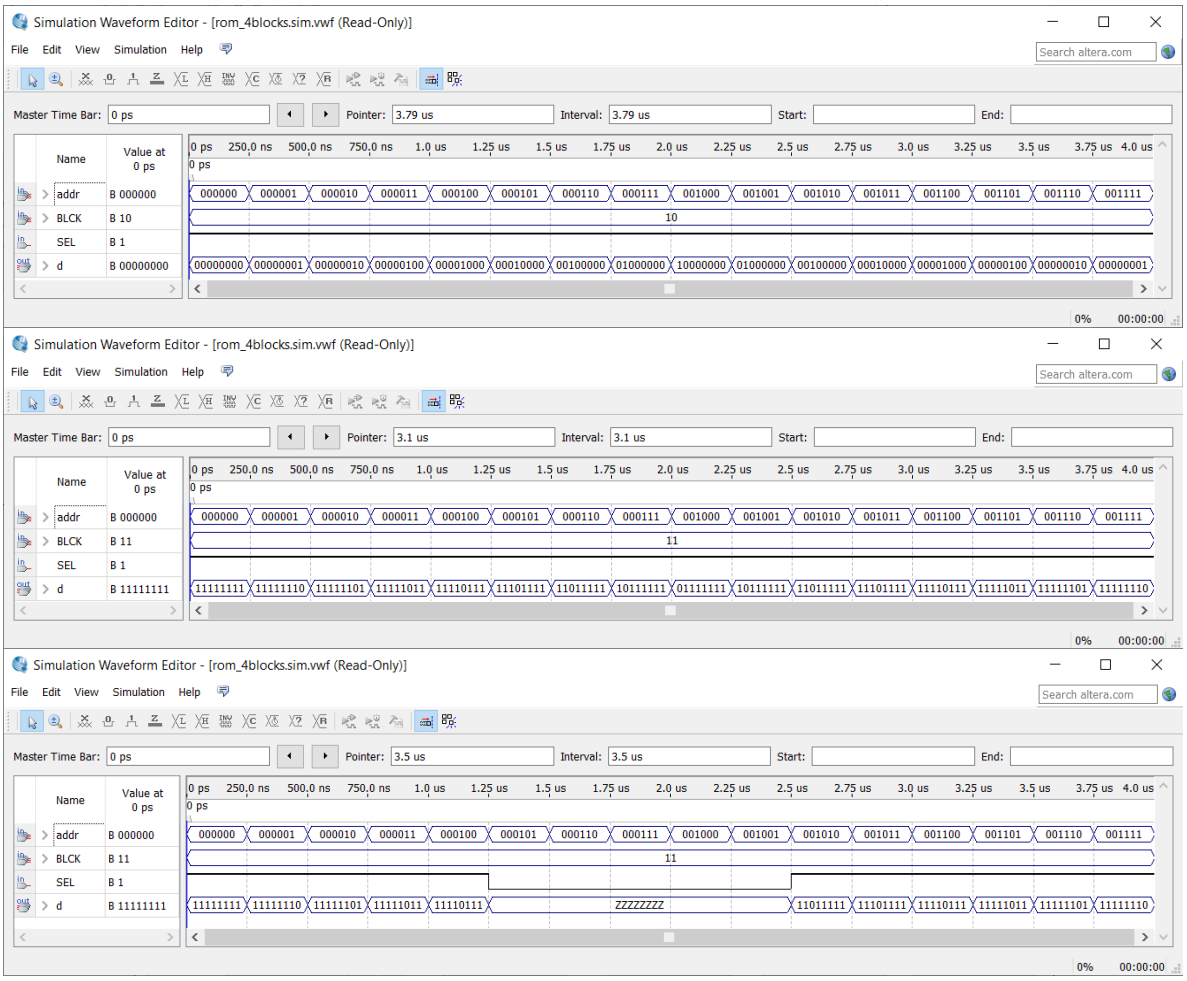


Figura 45: simulação funcional do módulo "rom_4blocks". Nela, nota-se que o módulo se comporta de acordo com o especificado, selecionando o conteúdo de qual módulo de memória ROM 64x8 deve ser transmitido para saída a partir da entrada seletora. Como os módulos de memória usados já tiveram todos os seus endereços acessados anteriormente, se torna desnecessário vasculhá-los novamente.

6. Projeto 6: Circuito de teste dos 4 módulos de memória ROM 64x8

Este projeto finaliza a primeira parte desta atividade, consistindo, basicamente, em encapsular o circuito anterior em um módulo (*"teste_ROM"*) para testá-lo na FPGA, usando da associação de entradas e pinos apresentada no roteiro. Como ele foi desenvolvido com a única intenção de testar o módulo *"rom_4blocks"*, é desnecessário definir seu escopo, especificações e simulações. Dessa forma, na figura 46 é apresentada o diagrama final do circuito de teste dos 4 módulos de memória ROM 64x8. Esse circuito foi testado da mesma forma que no vídeo disponibilizado no roteiro da atividade, apresentando comportamento idêntico.

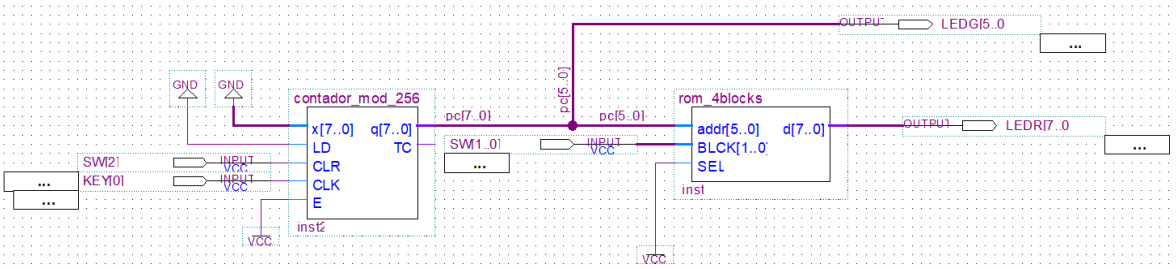


Figura 46: diagrama esquemático do circuito de teste do módulo "rom_4blocks". Como dito anteriormente, este módulo foi encapsulado no módulo denominado de "teste_ROM" e a associação de entradas e pinas para teste foi adotada de acordo com o apresentado no roteiro da atividade. O circuito foi testado na FPGA de modo análogo ao vídeo fornecido no roteiro da atividade e desenvolveu comportamento idêntico ao observado.

7. Projeto 1: Módulo de memória RAM 16x4

7.1. Escopo:

Projeto de um circuito sequencial que implementem dispositivos de memória volátil de leitura e escrita com 16 palavras de memória de 4 bits. O dispositivo deve ser implementado utilizando de *flip-flops* tipo D sensíveis a borda de subida, bem como ter sua funcionalidade de leitura e escrita condicionada a um sinal de controle ($R0W1$) e um de habilitação (SEL).

7.2. Especificação de alto-nível:

Entradas:
$\underline{addr} = (addr_3, addr_2, addr_1, addr_0)$, com $addr_k \in \{0,1\}$ e $k = 0, \dots, 3$ (entrada seletora de endereços);
$\underline{i} = (i_3, i_2, i_1, i_0)$, com $i_k \in \{0,1\}$ e $k = 0, \dots, 3$ (entrada de escrita);
$SEL \in \{0,1\}$;
$R0W1 \in \{0,1\}$.

Saída:
$\underline{o} = (o_3, o_2, o_1, o_0)$, com $o_k \in \{0,1\}$ e $k = 0, \dots, 3$.

Função de Saída:

SEL	$R0W1$	Operação	$\underline{o}$
1	0	Leitura ($\underline{o} = [\underline{addr}]$)	$[\underline{addr}]$
1	1	Escrita ($[\underline{addr}] = \underline{i}$)	$ZZZZ$
0	0	-	$ZZZZ$

onde $[\underline{addr}]$ significa o conteúdo de memória no endereço indicado pela entrada $\underline{addr}$. Sendo que a operação de escrita só executa mediante a ocorrência da borda ativa do clock.

7.3. Especificação binária:

Entradas:
$\underline{addr} = (addr_3, addr_2, addr_1, addr_0)$, com $addr_k \in \{0,1\}$ e $k = 0, \dots, 3$ (entrada seletora de endereços);
$\underline{i} = (i_3, i_2, i_1, i_0)$, com $i_k \in \{0,1\}$ e $k = 0, \dots, 3$ (entrada de escrita);
$SEL \in \{0,1\}$;
$R0W1 \in \{0,1\}$.

Saída:
$\underline{d} = (d_3, d_2, d_1, d_0)$, com $d_k \in \{0,1\}$ e $i = 0, \ldots, 3$.

Função de Saída:

Dado que é inviável construir uma tabela verdade de transição, excitação e saída com todas as possibilidades de entradas do sistema completo, pois cada uma das 16 palavras apresenta 4 *flip-flops* os quais precisariam ter seus estados analisados, além dos 12 bits de entrada, será produzido um módulo específico para atuar como uma memória RAM de 1 palavra com 4 bits (módulo *"ram1x4"*). Dessa forma, para construir a RAM 16x4 basta juntar 16 símbolos desse módulo de menor memória e fazer as conexões adequadas.

Para implementar o módulo *"ram1x4"*, primeiro é necessário notar não só que o comportamento de leitura e escrita desejado para a unidade RAM 16x4 deve ser repetido por cada componente de menor escala , mas também que essa funcionalidade se daria de forma idêntica à operação de carga síncrona em um registrador de 4 bits. De fato, a única diferença de um circuito do módulo *"ram1x4"* para um registrador do tipo citado seria que a saída da unidade de memória deve entrar em estado de alta impedância para o caso em que ela não foi habilitada pela entrada SEL ou que a operação realizada é de escrita ($SEL = 1$ e $R0W1 = 1$) . Portanto, é possível utilizar dos módulos *"registrador_4_bits"* (especificado, implementado e testado na atividade de laboratório 3) e *"bus_tristate"* (especificado, implementado e testado na atividade de laboratório 4) para tornar simples e direta a implementação dessa memória base, bastando conectar a entrada de carga da memória à entrada de carga do registrador, a saída do registrador à entrada dos *buffers* e configurar os sinais de controle restantes.

Assim, teremos que o módulo *"ram1x4"* deve apresentar entradas $\underline{i}$ (bits para carga síncrona), SEL (entrada de habilitação) e $R0W1$ (entrada de controle de operação) similares às entradas da unidade maior. Dessa forma, sendo LD e CLR os sinais de carga e limpeza síncrona do registrador, bem como e o sinal de habilitação do módulo de *buffers tri-state*, teremos que:

SEL	$R0W1$	LD	CLR	e	
0	1	0	0	0	$LD = SEL \cdot R0W1$
0	1	0	0	0	$CLR = 0$
1	0	0	0	1	$e = SEL \cdot R0W1'$
1	0	1	0	0	

Além disso, para manter a sincronia do circuito, também é necessário que o clock fornecido a memória completa seja igual ao clock fornecido a suas unidades menores o qual deve ser inserido na entrada CLK do registrador. Este raciocínio foi utilizado para implementar o circuito do módulo *"ram1x4"* cujo diagrama esquemático resultante está disposto na figura 47. O módulo passou por uma simulação funcional, disponível na figura 48, a qual comprovou que o circuito age de acordo com o especificado

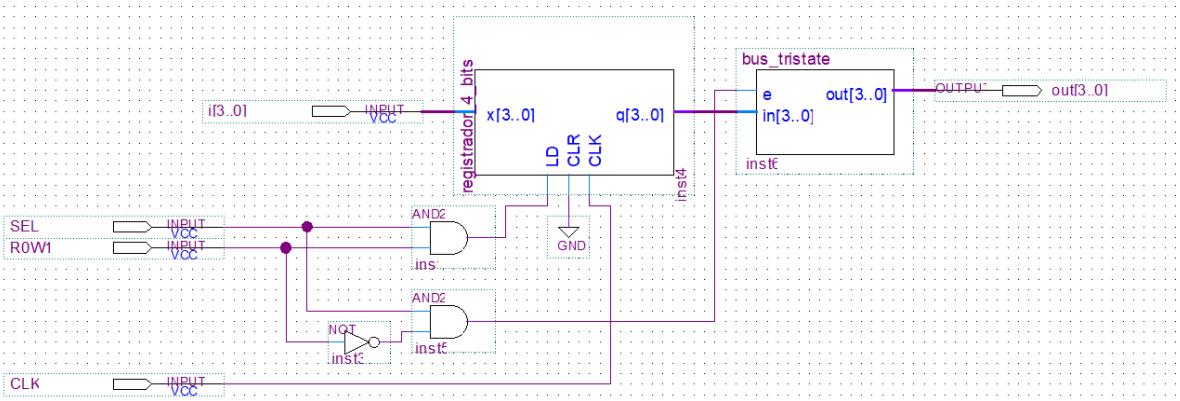


Figura 47: diagrama esquemático do circuito do módulo "ram1x4" o qual implementa uma unidade de memória RAM de 1 palavra de 4 bits. Este módulo foi desenvolvido utilizando dos módulos "registrador_4_bits" e "bus_tristate" implementados e testados nos roteiros 3 e 4. Tal método foi empregado de modo a facilitar a implementação e organizar o circuito da memória RAM 16x4, que é o objetivo final desta segunda parte da atividade.

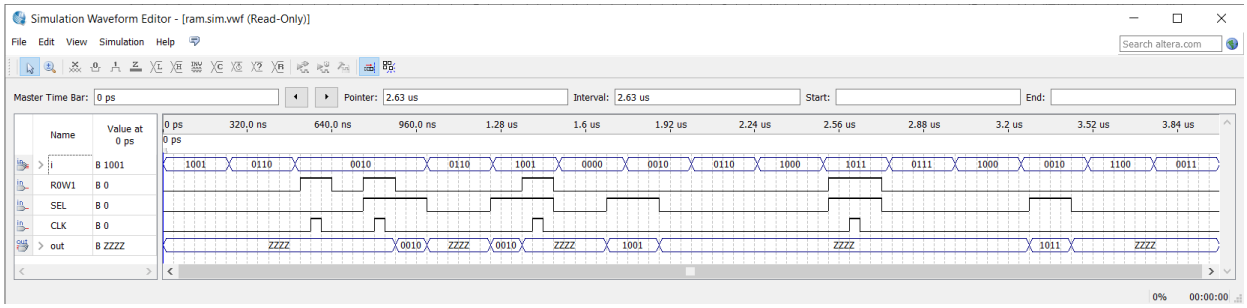


Figura 48: simulação temporal do módulo "ram1x4". Nela, nota-se que o comportamento descrito pelo circuito obedece ao especificado.

7.4. Minimizações:

Usando do módulo *"ram1x4"* especificado acima e do módulo *"decodificadorBIN_ONEH"* apresentado na introdução, torna-se trivial implementar o circuito da memória RAM 16x4 (o qual recebeu o nome de "ram", para se manter fiel ao roteiro da atividade), não sendo necessária qualquer outra minimização. Basta concatenar 16 símbolos do tipo *"ram1x4"*, fornecer a eles as mesmas entradas CLK, $ROW1$ e $\underline{i}$ do circuito da RAM final e, em cada um deles, conectar uma das saídas do decodificador (que, por sua vez, recebe as entradas SEL e $\underline{addr}$ do circuito externo). Dessa forma, a saída de dados $\underline{o}$ será composta por um barramento que interliga todas as saídas das múltiplas unidades de memória de menor porte.

7.5. Esquemático do circuito:

Seguindo as estratégias supracitadas, o circuito do módulo *"ram"* foi implementado e seu diagrama esquemático resultante está disposto na figura 49.

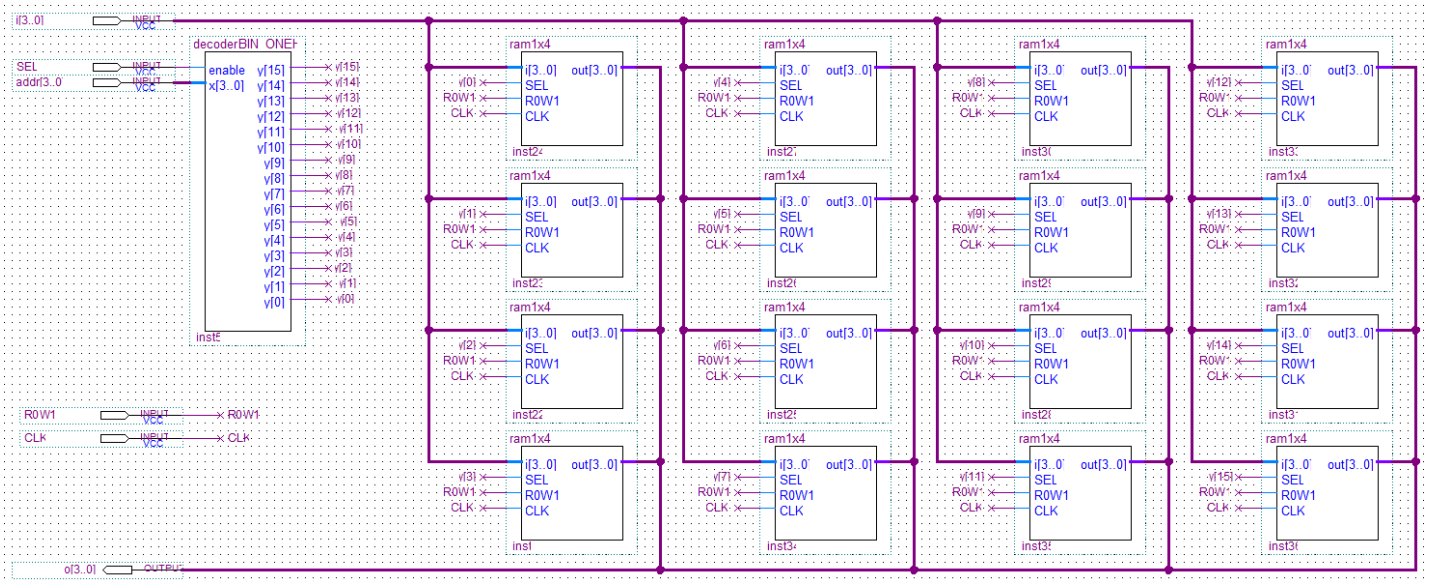


Figura 49: diagrama esquemático do circuito do módulo "ram" o qual implementa uma unidade de memória RAM de 16 palavras de 4 bits com funcionalidade de leitura e escrita síncrona. Este módulo foi desenvolvido utilizando dos módulos "ram1x4" e "decodificadorBIN_ONEH" implementado anteriormente. Este é o circuito final da segunda parte do projeto, o qual será encapsulado em um símbolo para ser testado na FPGA no projeto posterior.

7.6. Simulações:

Para testar a memórias construída, uma simulação funcional percorrendo cada endereço disponível duas vezes, a primeira escrevendo valores aleatórios neles e a segunda lendo o que foi escrito, foi realizada. Os resultados obtidos a partir dela está disposto na figura 50.

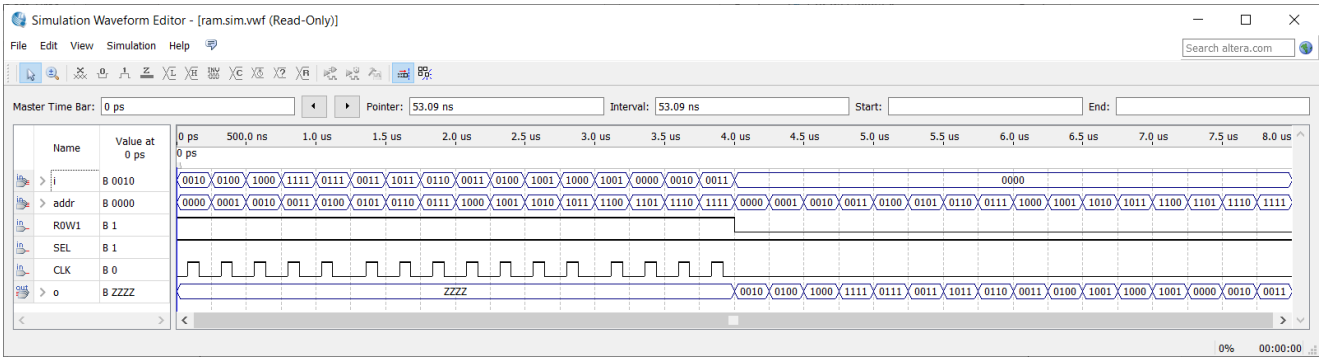


Figura 50: simulação funcional do circuito do módulo "ram". Nela, nota-se que o comportamento descrito pelo circuito é igual ao especificado para ele.

8. Projeto 8: Circuito de teste da memória RAM

Este projeto finaliza a segunda parte desta atividade, consistindo, basicamente, em encapsular o circuito anterior em um módulo (*"teste_RAM"*) para testá-lo na FPGA, usando da associação de entradas e pinos apresentada no roteiro. Como ele foi desenvolvido com a única intenção de testar o módulo *"ram"*, é desnecessário definir seu escopo, especificações e simulações. Dessa forma, na figura 51 é apresentada o diagrama final do circuito de teste da memória RAM. Esse circuito foi testado da mesma forma que no vídeo disponibilizado no roteiro da atividade, apresentando comportamento idêntico.

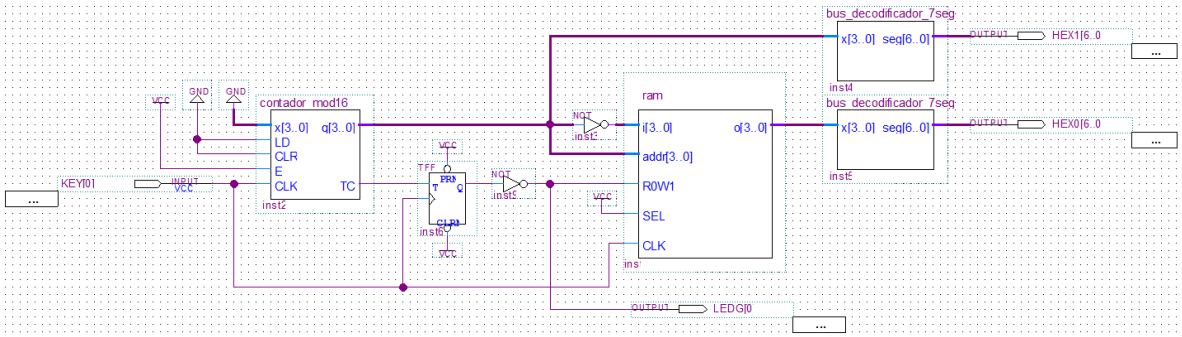


Figura 51: diagrama esquemático do circuito de teste do módulo "ram". Como dito anteriormente, este módulo foi encapsulado no módulo denominado de "teste_RAM" e a associação de entradas e pinas para teste foi adotada de acordo com o apresentado no roteiro da atividade. O circuito foi testado na FPGA de modo análogo ao vídeo fornecido no roteiro da atividade e desenvolveu comportamento idêntico ao observado.