GEBZE TEKNİK ÜNİVERSİTESİ

ELEKTRONİK MÜHENDİSLİĞİ

ELM235

LOJİK DEVRE TASARIM LABORATUVARI

LAB 0x5 Deney Raporu

Hafıza

Hazırlayanlar
1) 1801022035 – Ruveyda Dilara Günal
2) 200102002087 – Alican Bayındır

Bu labın amacı

- Hafıza oluşturup, veri okumak,
- Tasarlanan devreleri gerçekleyip test edebilmek.

Problem 1 - Hafıza oluşturma ve okuma

Bu problemde sizden istenen oluşturduğunuz küçük bir hafızadan verilen adres değerine göre, o adreste bulunan verileri çekmektir. Oluşturacağınız hafıza registerları tutacaktır. Toplam register sayısını 32 olarak alınız. Devrenin temel çalışma prensibi, rs1 ve rs2 değerlerini adres gibi alıp, hafızada içinde barındırdıkları 32-bit dataları rs1_data ve rs2_data portlarına aktarması istenmektedir. Okuma işlemini kombinasyonel tasarlayınız. Basit bir hafıza kodu örneği ekte verilmiştir.

Bu problem için sadece okumayı test eden bir testbench örneğini githubdaki repoda bulabilirsiniz.

```
module p1 (

input logic clk, reset,

// yazma portları

input logic we,

input logic [4:0] waddr,

input logic [31:0] wbdata,

// okuma portları

input logic [4:0] rs1,

input logic [4:0] rs2,

output logic [31:0] rs1_data,

output logic [31:0] rs1_data

);
```

İstenen hafızada herhangi bir clock, reset, yazma işlemi olmadığı için write enable, wbdata gibi değerler örnek koddan çıkarılmıştır. Yeni yazılan kod aşağıdaki gibidir;

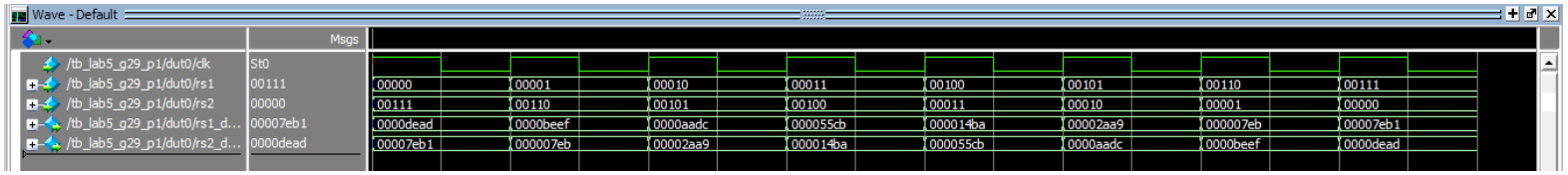
```
module lab5_g29_p1 (

	input logic clk, reset, wen,

	input logic [4:0] waddr,

	input logic [31:0] wbdata,

	input logic [4:0] rs1, rs2,

	output logic [31:0] rs1_data, rs2_data

);

	logic [31:0] mem [0:31];

	initial

		$readmemh ("reg_image.txt" , mem);

	integer i;

	always_ff @(posedge clk)

	begin

		if (reset)

			for (i=0; i<32; i=i+1)

				mem[i] <= 32'b0;

		else if (wen)

			mem[waddr] <= wbdata;

	end

	assign rs1_data = mem[rs1];

	assign rs2_data = mem[rs2];


endmodule
```

Yukarıdaki kodu test etmek için yazılan testbench dosyası aşağıdadır;

```
`timescale 1ps/1ps

module tb_lab5_g29_p1 ();

logic clk,   reset, wen;

logic [4:0]  waddr;

logic [31:0] wbdata;

logic [4:0]  rs1, rs2;

logic [31:0] rs1_data, rs2_data;


lab5_g29_p1 dut0(clk, reset, wen, waddr, wbdata, rs1, rs2, rs1_data, rs2_data);

always

begin

        clk = 1;

        forever #5 clk = ~clk;

end


initial

begin

        wen = 0; reset = 0; wbdata = 16'b0;

        rs1 = 5'b00000; # 10;

        rs1 = 5'b00001; # 10;

        rs1 = 5'b00010; # 10;

        rs1 = 5'b00011; # 10;

        rs1 = 5'b00100; # 10;

        rs1 = 5'b00101; # 10;

        rs1 = 5'b00110; # 10;
```

```
        rs1 = 5'b00111; # 10;

end



initial

begin

        wen = 0; reset = 0; wbdata = 16'b0;

        rs2 = 5'b00111; # 10;

        rs2 = 5'b00110; # 10;

        rs2 = 5'b00101; # 10;

        rs2 = 5'b00100; # 10;

        rs2 = 5'b00011; # 10;

        rs2 = 5'b00010; # 10;

        rs2 = 5'b00001; # 10;

        rs2 = 5'b00000; # 10;

        $stop;

end



endmodule
```

Testbench dosyasının düzgün çalışması için aşağıdaki reg_image.txt dosyası ile projenin aynı dosyada olması gerekmektedir. Yukarıdaki testbench kodları föy ile birlikte verilen reg_image.txt dosyası ile birlikte çalıştırılınca aşağıdaki dalga formu elde edilmiştir. Rs2_data nın geriye doğru çıkması register adresleri üzerinde geriye doğru sayarak ilerlemesidir.



Şekil 1 Modelsim Dalga formu

Reg_image.txt dosyasının içindeki veriler aşağıdaki şekilde değiştirildiği zaman;

4d4543

49540a

484f43

414d0a

4c5554

46454e

0a5955

4b5345

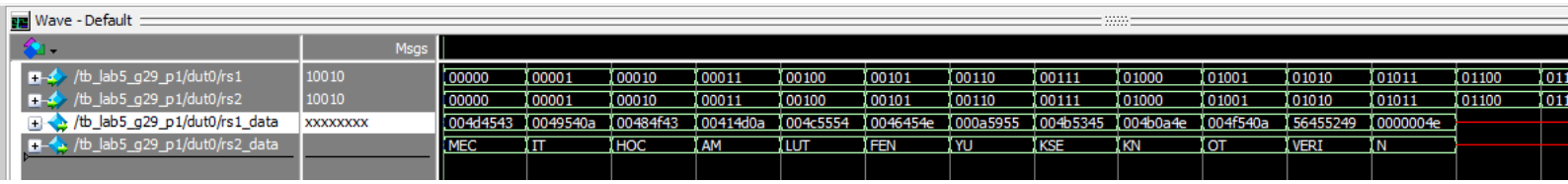
4b0a4e

4f540a

56455249

4e

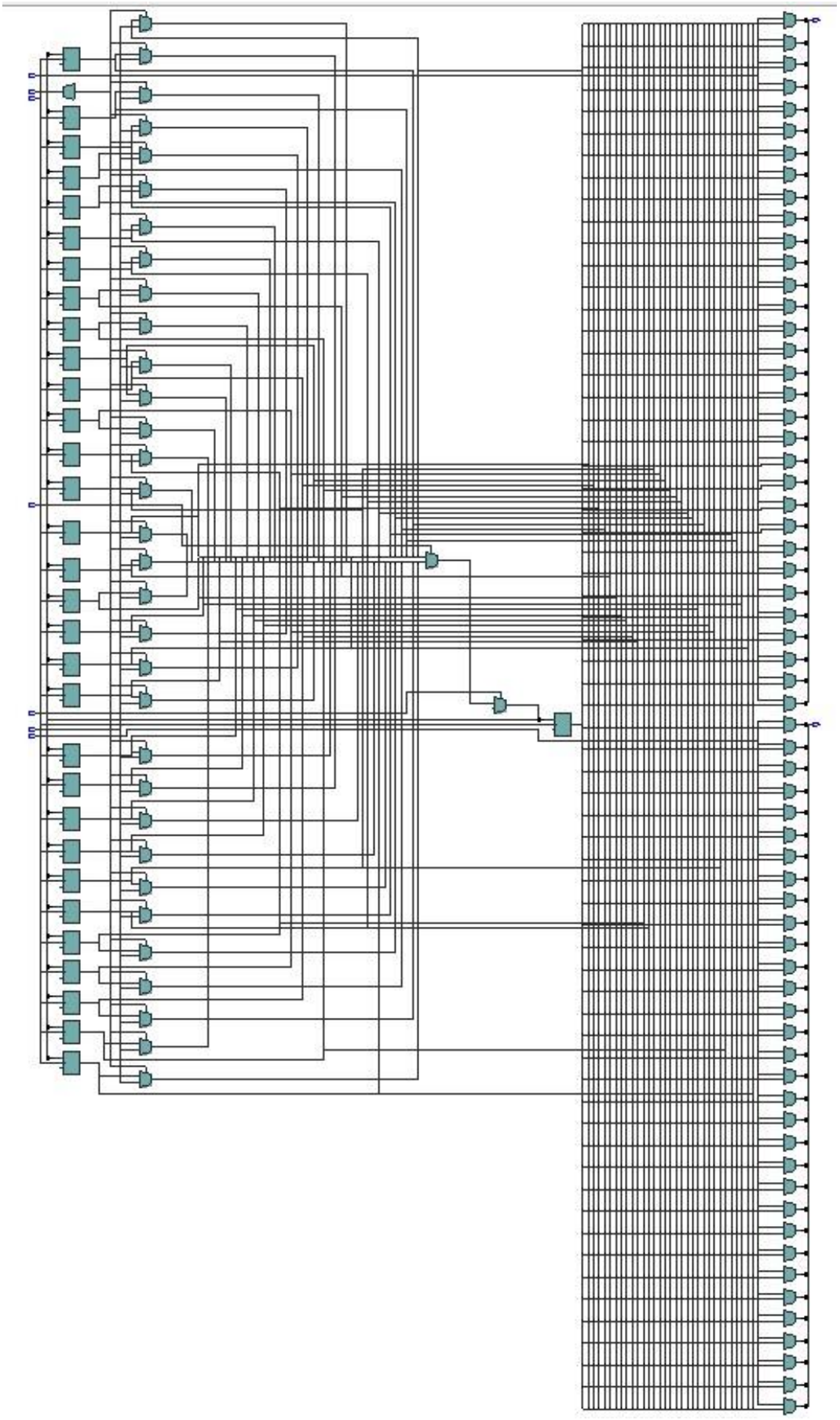
Yukarıdaki txt file'ı tasarlanan hafıza bloğunun içerisinden okunmaya çalışıldığı zaman aşağıdaki sinyal elde edilmiştir;



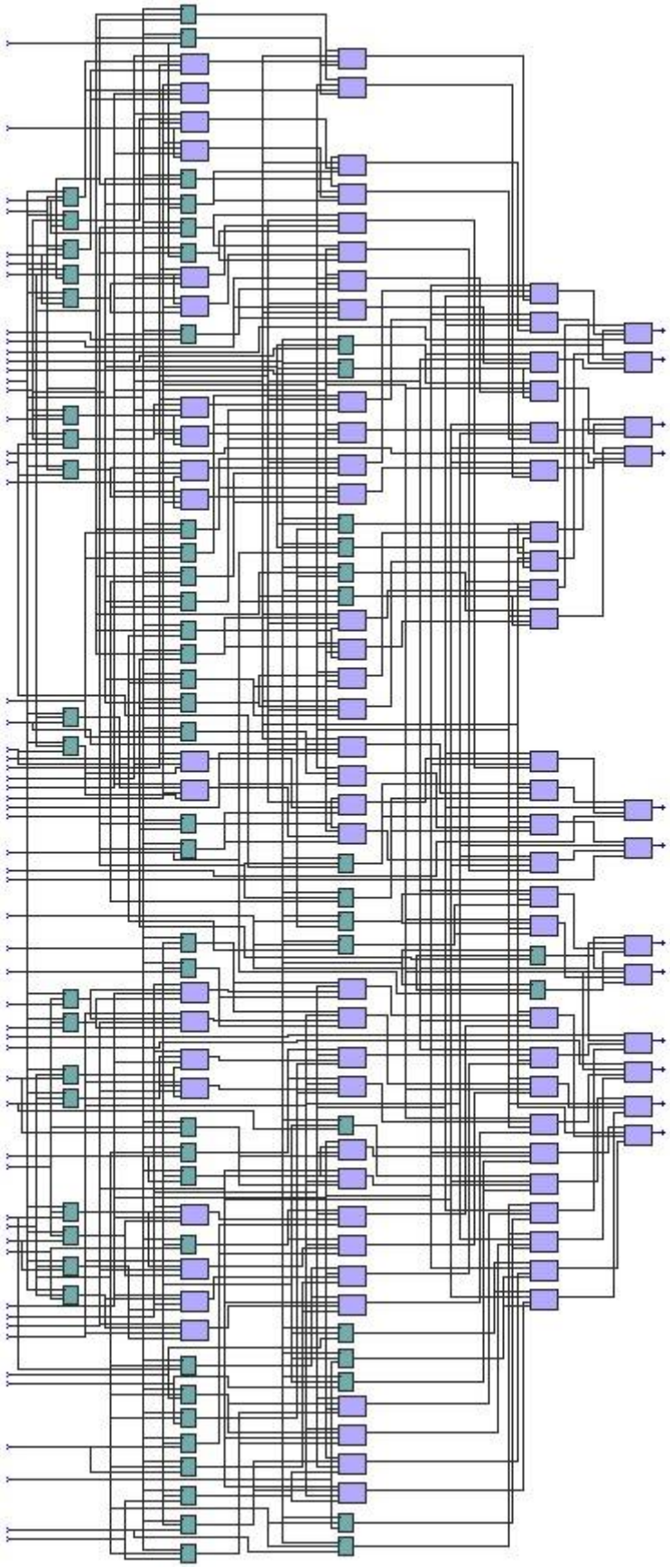
Şekil 2 Devrenin Modelsim üzerindeki Dalga Formları.

Rs1_data hexadecimal rs2_data ise ASCII gösterim formatına alındığı zaman sinyal daha net görülür.

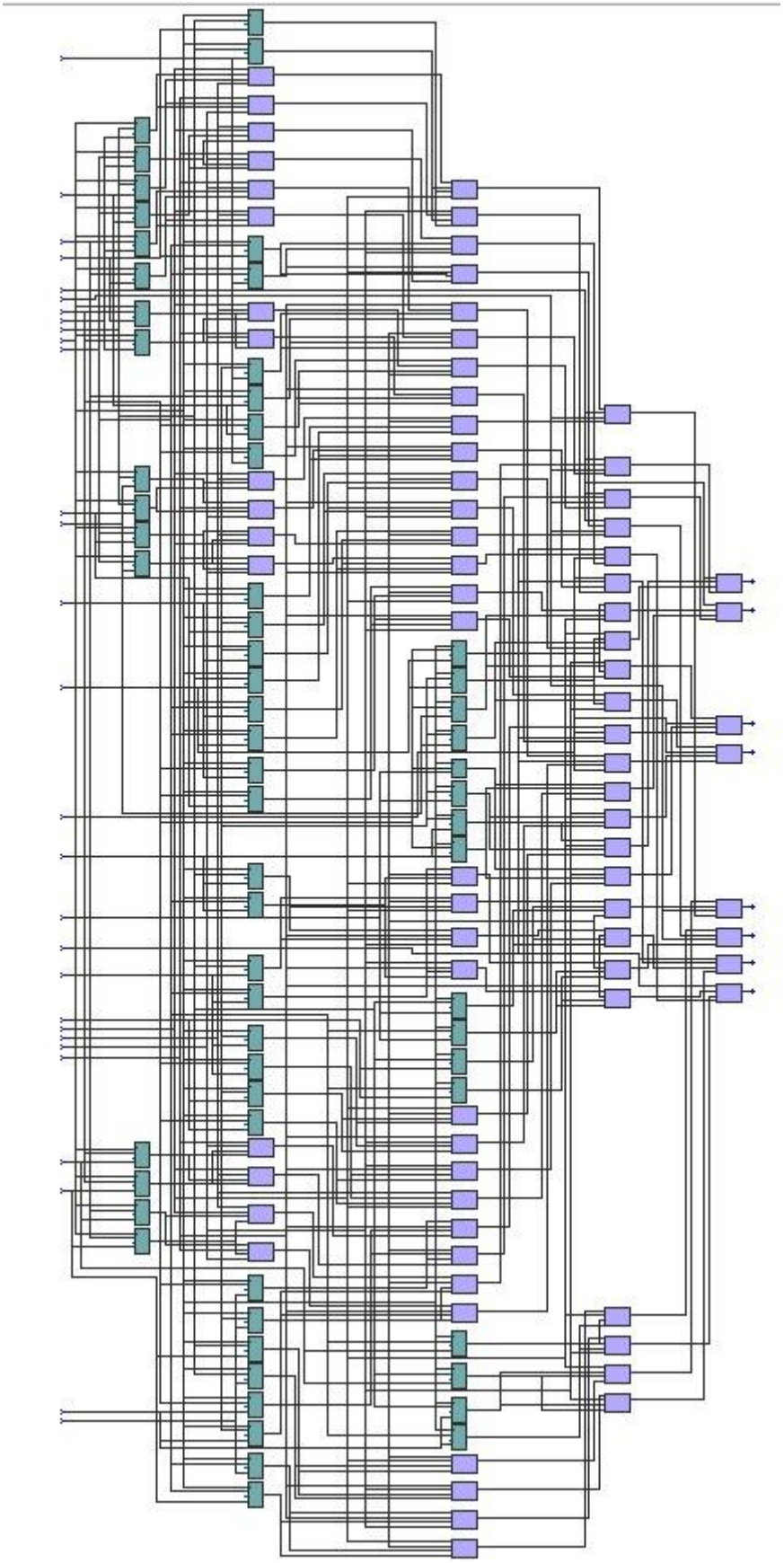
Tasarlanan devrenin Quartus uygulamasında çalıştırıldıktan sonra alınan çıktılar ise aşağıdaki gibidir;



Şekil 3 Devrenin RTL şeması



Şekil 4 Devrenin Post mapping şeması



Şekil 5 Devrenin post fitting şeması

	Resource	Usage
1	Estimated Total logic elements	2,408
2		
3	Total combinational functions	1416
4	⌄ Logic element usage by number of LUT inputs	
1	-- 4 input functions	1320
2	-- 3 input functions	64
3	-- <=2 input functions	32
5		
6	⌄ Logic elements by mode	
1	-- normal mode	1416
2	-- arithmetic mode	0
7		
8	⌄ Total registers	1024
1	-- Dedicated logic registers	1024
2	-- I/O registers	0
9		
10	I/O pins	114
11		
12	Embedded Multiplier 9-bit elements	0
13		
14	Maximum fan-out node	clk~input
15	Maximum fan-out	1024
16	Total fan-out	8786
17	Average fan-out	3.29

Şekil 6 Devrenin resource usage summary'si

- Devrede 32x32 den dolayı 1024 adet register kullanılmıştır ve 114 adet pin kullanılmıştır. Toplamda ise 2408 adet lojik element kullanılmıştır.

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	UFM Blocks	DSP Elements
1	|lab5_g29_p1	1416 (1416)	1024 (1024)	0	0	0

Şekil 7 Devrenin resource utilization report'u.

DSP 9x9	DSP 18x18	Pins	Virtual Pins	ADC blocks	Full Hierarchy Name	Entity Name	Library Name
0	0	114	0	0	|lab5_g29_p1	lab5_g29_p1	work

Şekil 8 Resource Utility by Entity

Genel sonuç ve yorumlar;

Bu laboratuvar föyünde tasarlanması istenen hafıza birimi bütün isterleri başarılı bir biçimde yerine getirebilecek halde tasarlanmıştır. Tasarlanan hafıza birimi sorunsuz bir biçimde .txt uzantılı metin belgesindeki bütün verileri okuyabilmiş ve sinyal olarak dalga formunda gözlemlenmesini sağlamıştır. Bu deney sonucunda hafıza birimlerinin nasıl çalıştığı, nasıl tasarlandığı ve system verilog kullanılarak nasıl tasarlanılabileceği öğrenilmiştir.

Referanslar:

[1] https://github.com/fcayci/sv-digital-design

[2]https://www.intel.com/content/www/us/en/programmable/quartushelp/13.0/mergedProjects/hdl/vlog/vlog_pro_ram_inferred.htm

[3] https://www.youtube.com/watch?v=G9IPd9Bm_7

[4] Harris D.M., Harris S.L. - Digital Design and Computer Architecture (2016).pdf

[5] https://www.chipverify.com/verilog/verilog-single-port-ram