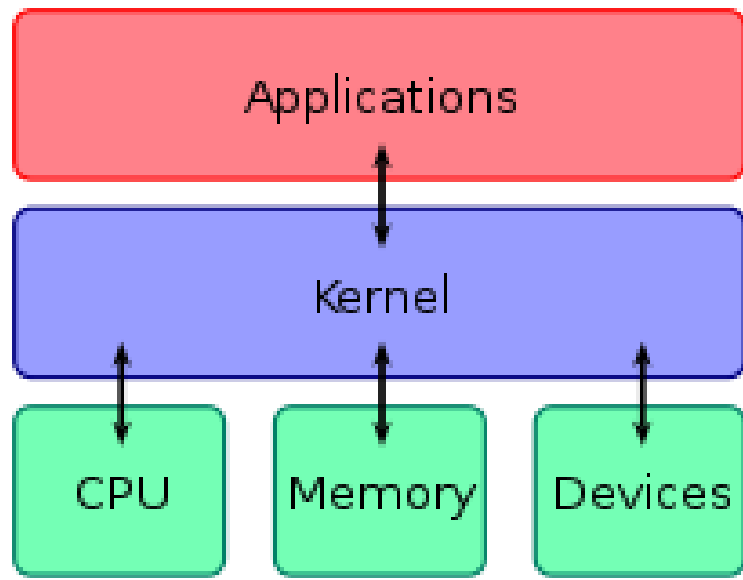
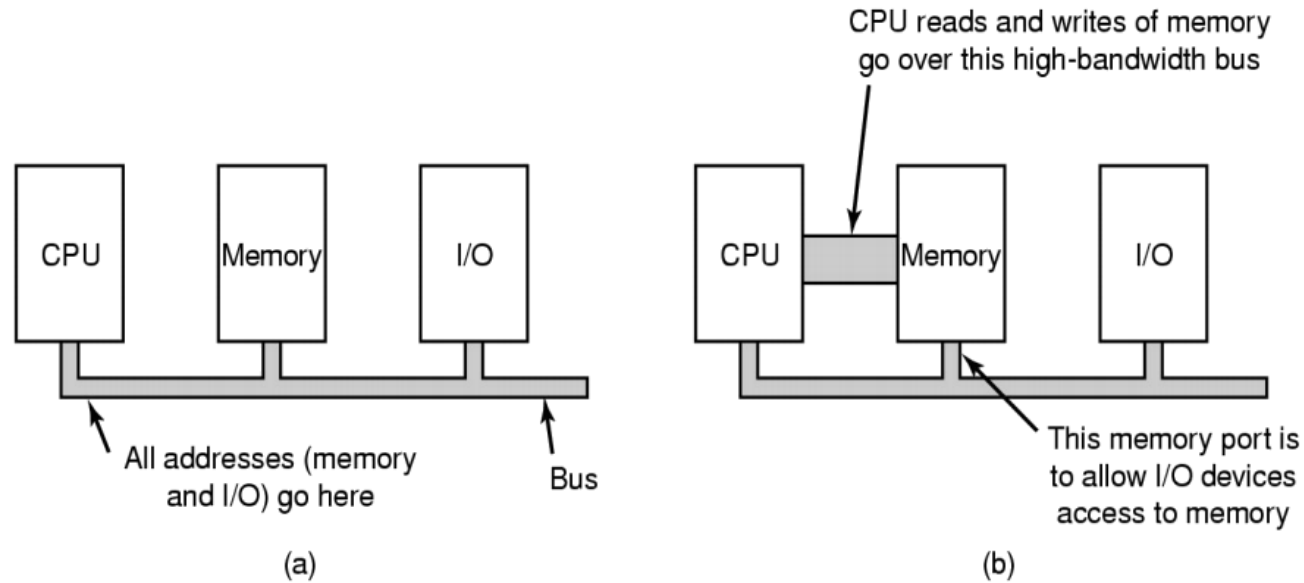


Tahmini Ders İçeriği

(Tentative Course Schedule – Syllabus)

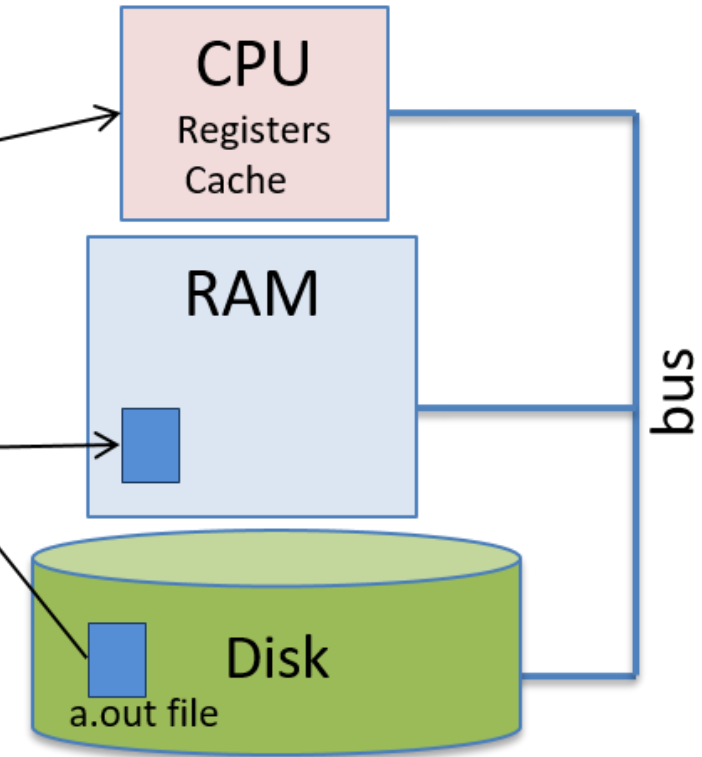
- 1. Hafta:** Sayısal Sinyaller/Sistemler, İkili Tabanda Sayılar, Taban Aritmetiği, İşaretili/Eksi Sayıların Gösterimi, Sayısal Tasarım Tarihçesi
- 2. Hafta:** İkili Mantık Aritmetiği ve Kapıları, Bool Cebiri Teorisi ve Tanımları, Bool Fonksiyonları, Kapı-Seviyesinde Yalınlaştırma, Karnough Haritası, Önemsenmeyen Durumlar, NAND, NOR, XOR
- 3-4. Hafta:** FPGA, Birleşik (Combinational) Devreler, Aritmetik Modüller, Decoder, Encoder, Mux, Verilog HDL
- 5. Hafta:** Ardışık (Sequential) Devreler, Mandal (Latch), Flip-Flop, Yazmaçlar (Registers)
- Lab Sınavı (265/264L)
- 6. Hafta:** Durum Makinaları, Örnek Tasarımlar, Sayaçlar (Counters)
- 7. Hafta:** FSM Örnekleri
- 8. Hafta:** RTL (Register Transfer Level) ASMD (Algorithmic State Machine and Datapath) Tasarımları
- 9. Hafta:** Durağan Zaman Analizi (Static Timing Analysis)
- Ara Sınav (265/264)
- 10. Hafta:** (14-18 Kasım) Bellekler, FPGA'da RAM, OpenRAM
- 11-12. Hafta:** (21-25 Kasım, 28 Kasım – 2 Aralık) Boru hattı, FPGA ve ASIC Tasarım Akışları
- Final (Aralık) – Proje Teslimleri (18 Aralık)

BELLEK (MEMORY)

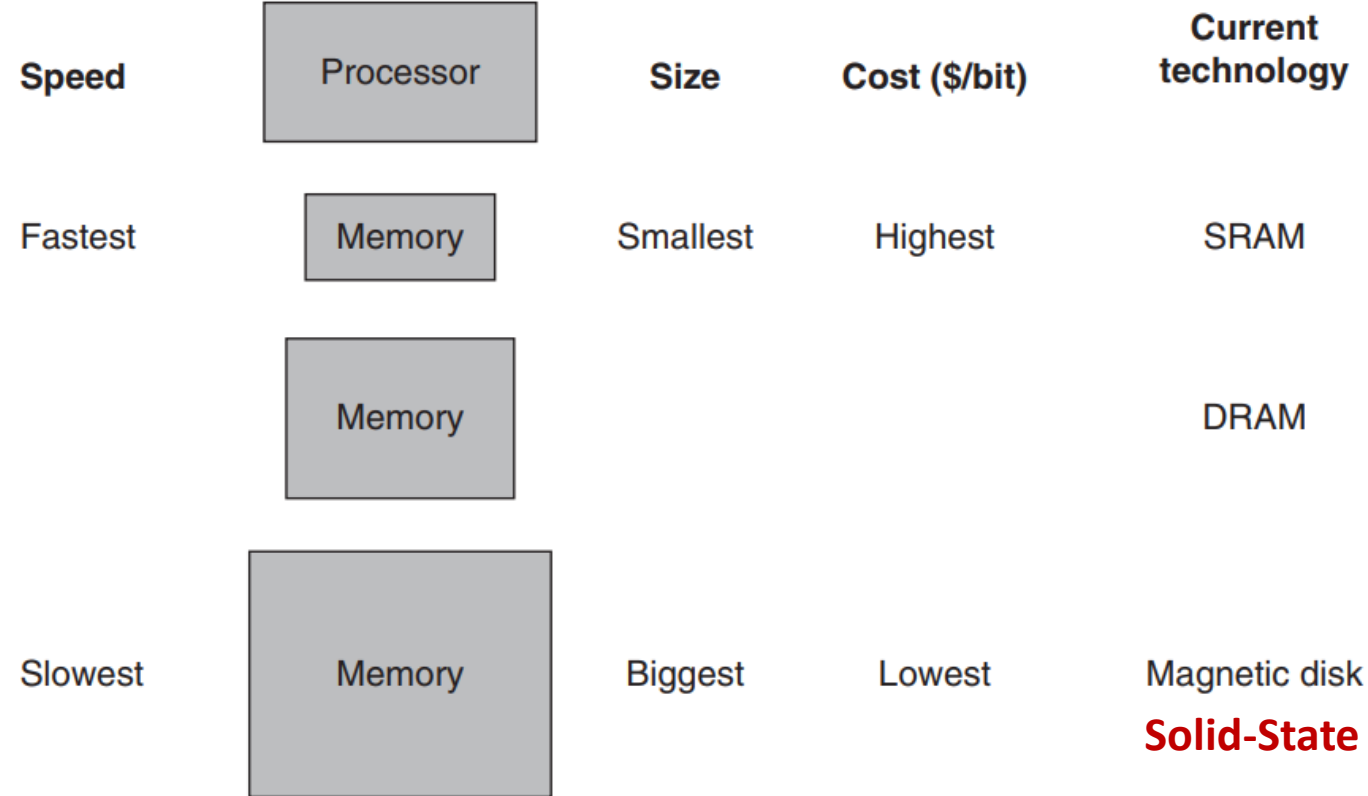


Starting a Program Running on System

1. Load binary from disk into RAM
2. Create & init new process
3. Init CPU state to run process



BELLEK MİMARİSİ



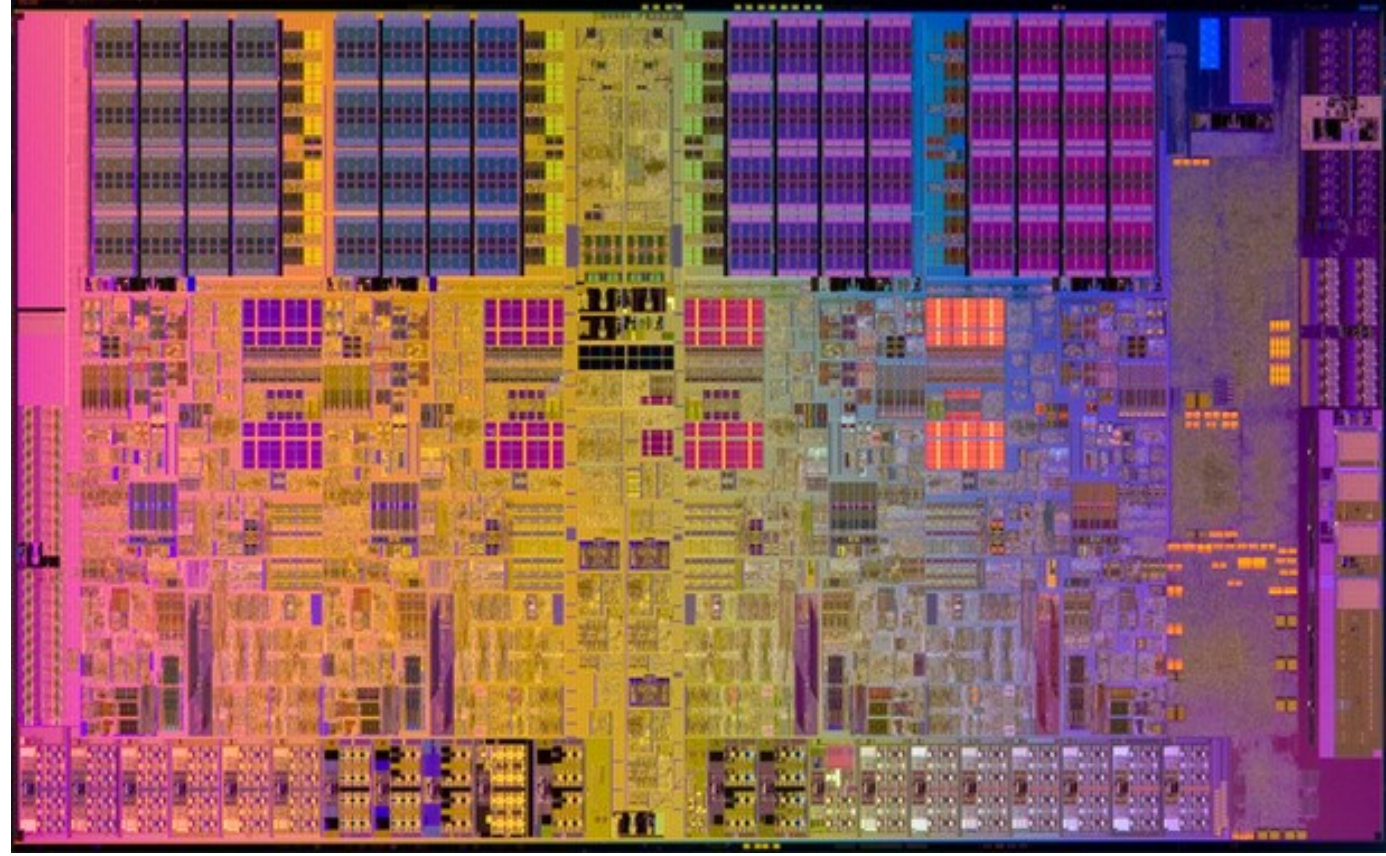
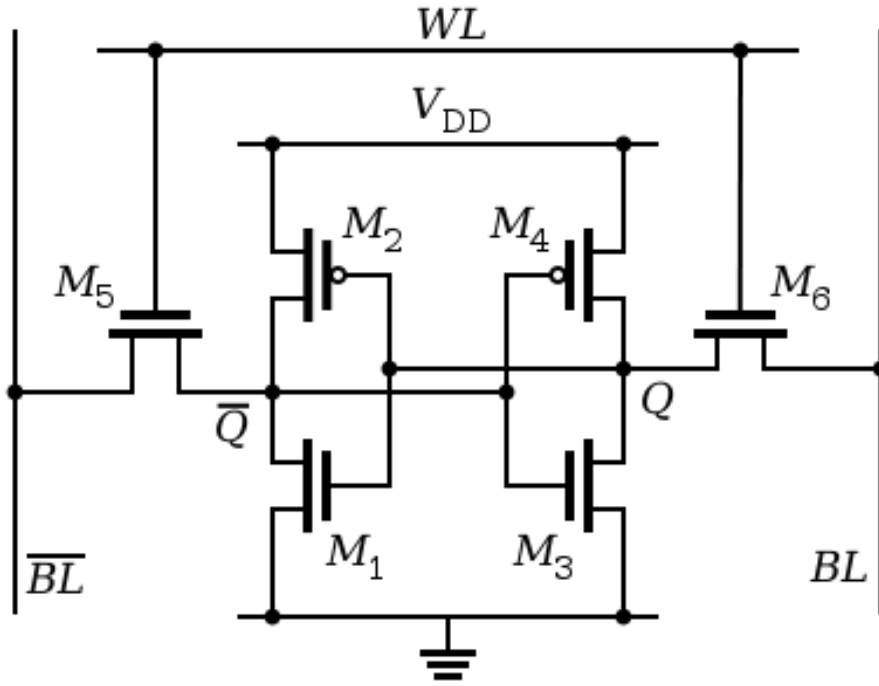
Memory technology	Typical access time	\$ per GiB in 2012
SRAM semiconductor memory	0.5–2.5 ns	\$500–\$1000
DRAM semiconductor memory	50–70 ns	\$10–\$20
Flash semiconductor memory	5,000–50,000 ns	\$0.75–\$1.00
Magnetic disk	5,000,000–20,000,000 ns	\$0.05–\$0.10

VOLATILE vs NON-VOLATILE

Volatile (Uçucu, geçici) : Güç kesildiğinde içerdiği bilgiler yok olan bellekler
Non-Volatile : Güç kesildiğinde dahi içerdiği bilgileri saklayan bellekler

Volatile Bellekler : SRAM, DRAM
Non-Volatile Bellekler : Flash (USB, SSD, QSPI), Magnetic Tape, CD/DVD, FRAM

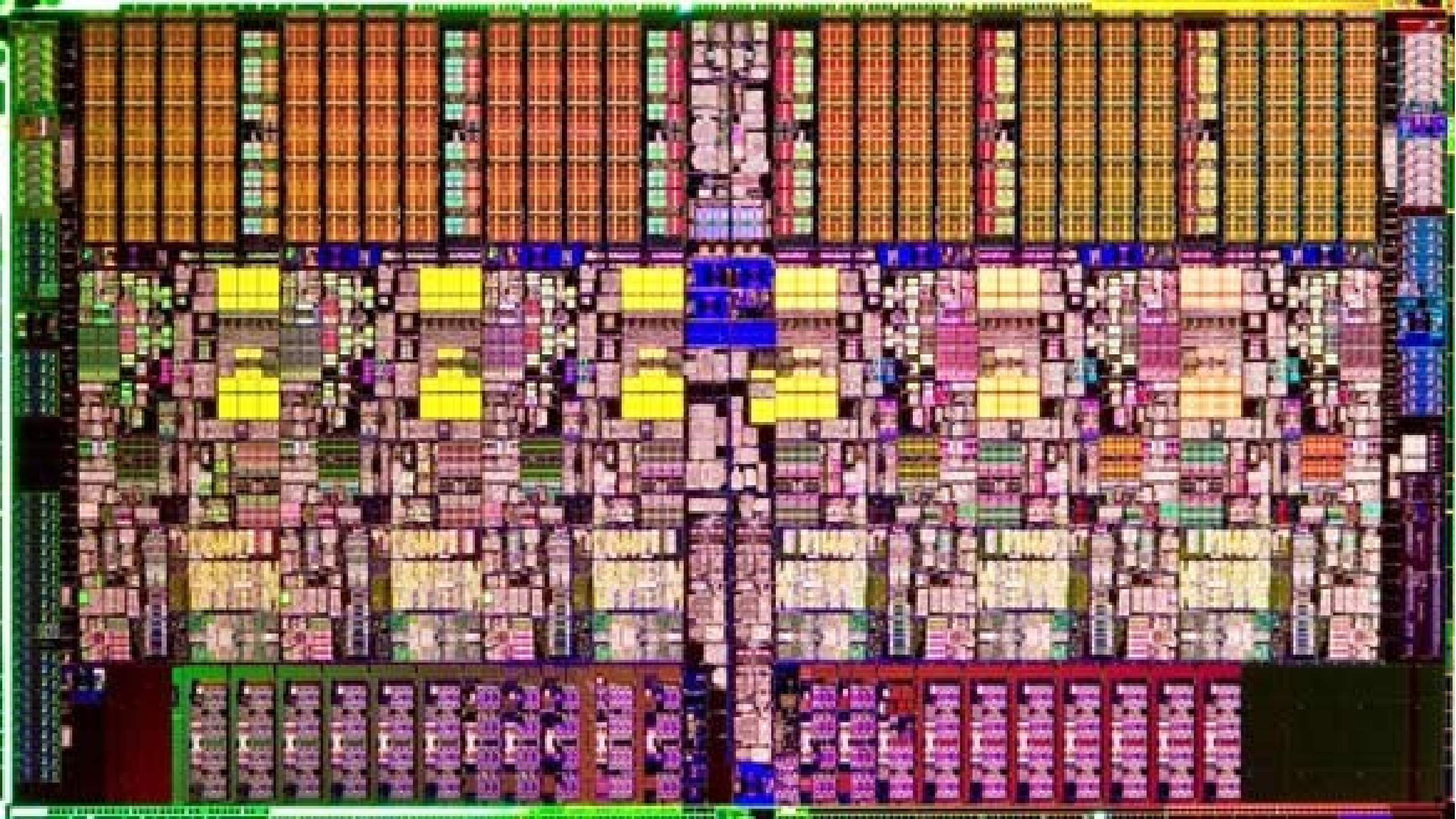
SRAM – Static Random Access Memory



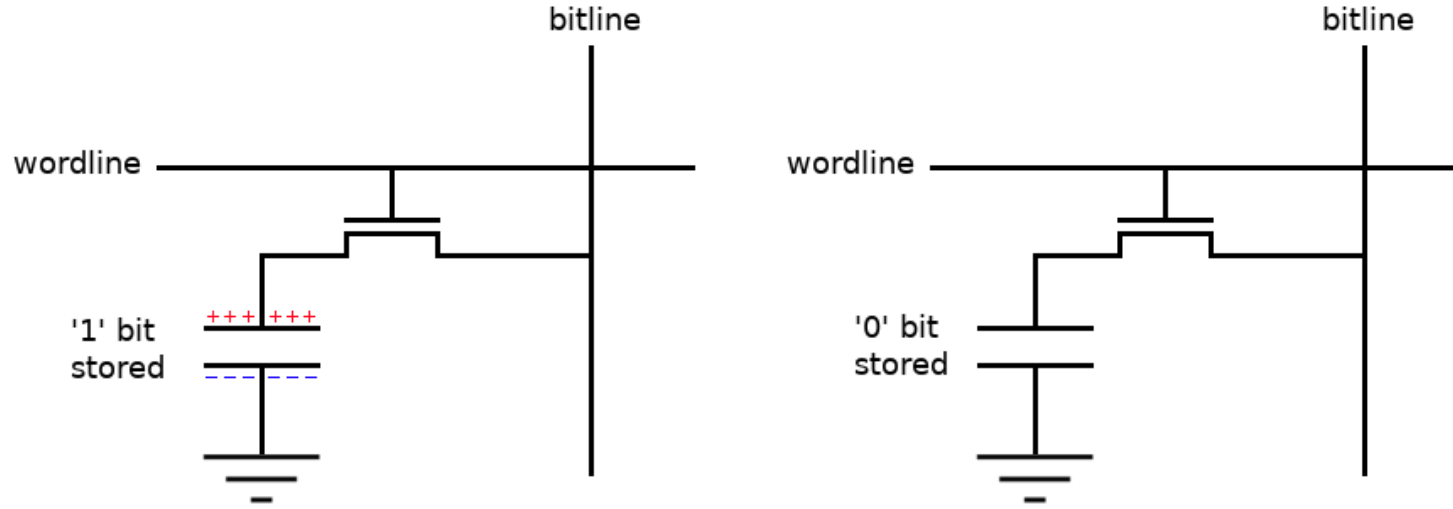
SRAM hızlı fakat pahalı ve yüksek alan tüketimine sahip

Statiktirler çünkü veriyi saklamak için DRAM gibi sürekli refresh edilmesi gerekmez

SRAM kullanım alanları: CPU içerisinde bulunan yazmaçlar, önbellekler, FPGA içerisinde bulunan Block RAM yapıları



DRAM – Dynamic Random Access Memory



DRAM ucuz ve düşük alan tüketimine sahip fakat yavaş

Dinmaiktirler çünkü veriyi saklamak için sürekli refresh edilmeleri gerekir

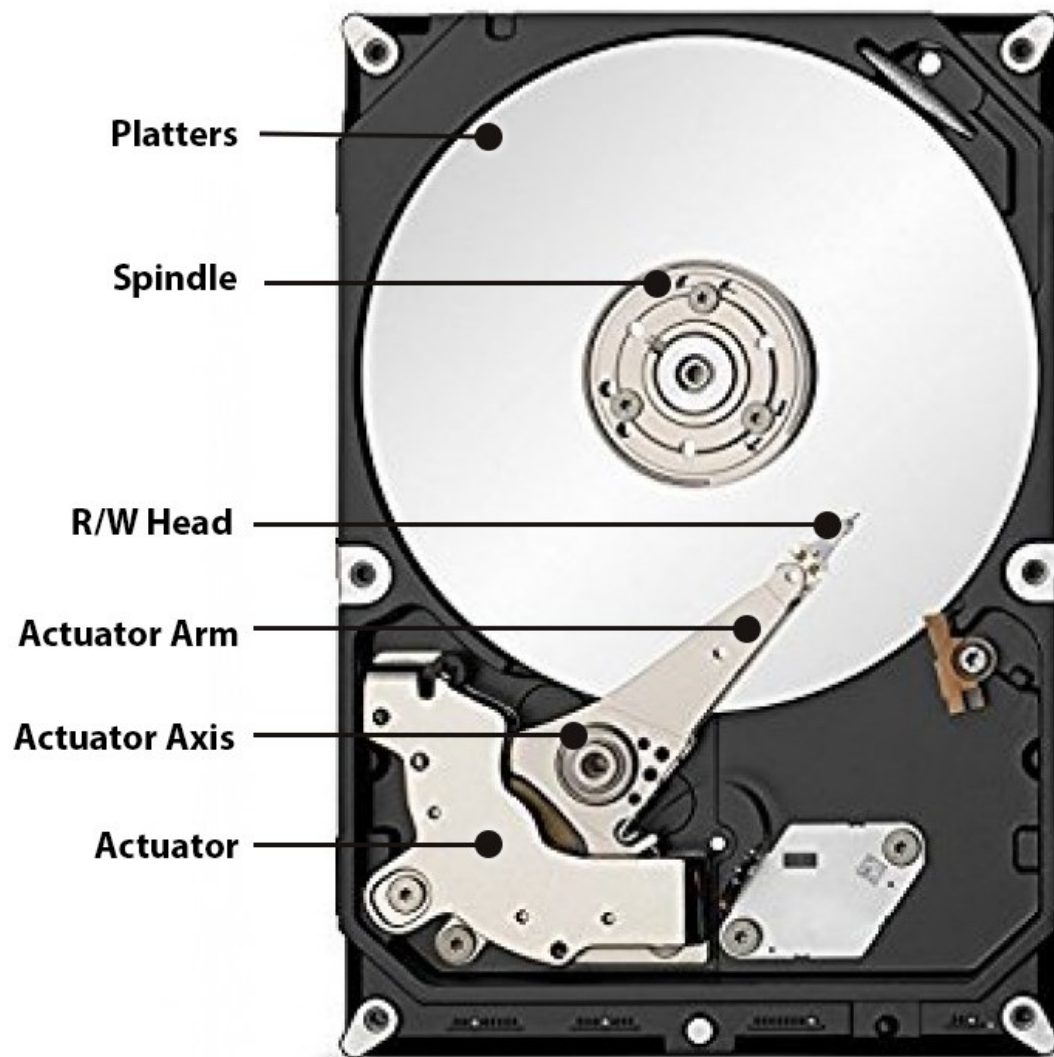
DRAM kullanım alanları: Bilgisayarda ana bellek, diğer işlemci birimleri için harici bellek

Flash Memory



HDD

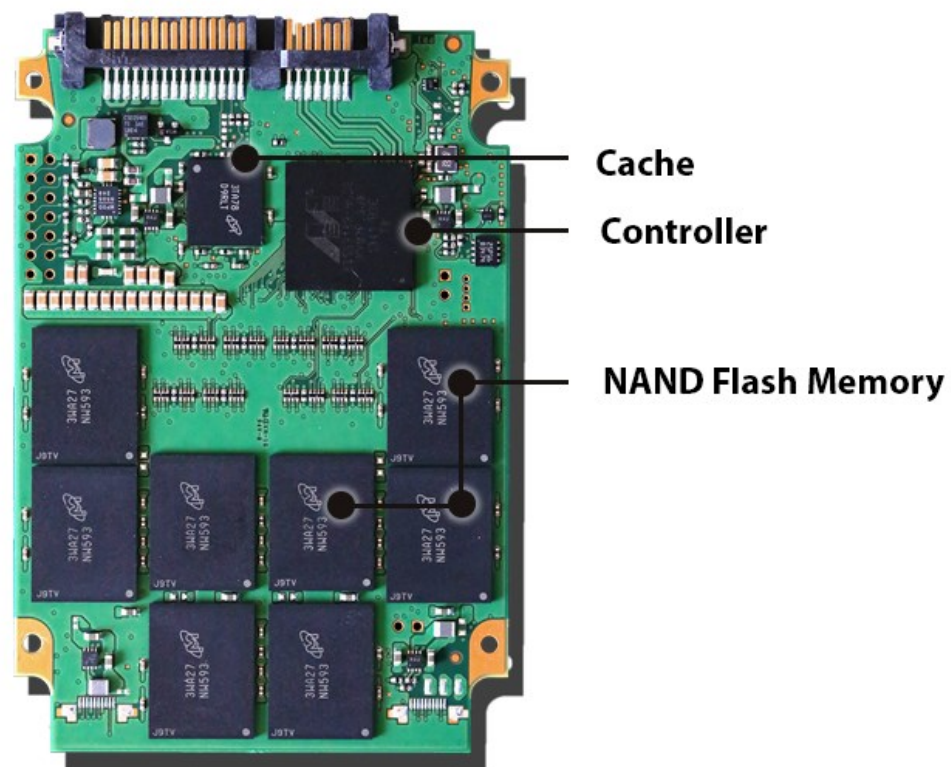
3.5"



Shock resistant up to 55g (operating)
Shock resistant up to 350g (non-operating)

SSD

2.5"



Shock resistant up to 1500g
(operating and non-operating)

FPGA'DA BELLEK

Flip-Flop

: En yaygın bulunan saklama birimi

Distributed RAM (LUTRAM)

: FPGA'daki LUT'ların RAM olarak kullanılması

Block RAM

: Özel olarak tasarlanmış saklama birimi

Artix-7 FPGA Feature Summary

Table 4: Artix-7 FPGA Feature Summary by Device

Device	Logic Cells	Configurable Logic Blocks (CLBs)		DSP48E1 Slices ⁽²⁾	Block RAM Blocks ⁽³⁾			CMTs ⁽⁴⁾	PCIe ⁽⁵⁾	GTPs	XADC Blocks	Total I/O Banks ⁽⁶⁾	Max User I/O ⁽⁷⁾
		Slices ⁽¹⁾	Max Distributed RAM (Kb)		18 Kb	36 Kb	Max (Kb)						
XC7A12T	12,800	2,000	171	40	40	20	720	3	1	2	1	3	150
XC7A15T	16,640	2,600	200	45	50	25	900	5	1	4	1	5	250
XC7A25T	23,360	3,650	313	80	90	45	1,620	3	1	4	1	3	150
XC7A35T	33,280	5,200	400	90	100	50	1,800	5	1	4	1	5	250
XC7A50T	52,160	8,150	600	120	150	75	2,700	5	1	4	1	5	250
XC7A75T	75,520	11,800	892	180	210	105	3,780	6	1	8	1	6	300
XC7A100T	101,440	15,850	1,188	240	270	135	4,860	6	1	8	1	6	300
XC7A200T	215,360	33,650	2,888	740	730	365	13,140	10	1	16	1	10	500

Notes:

- Each 7 series FPGA slice contains four LUTs and eight flip-flops; only some slices can use their LUTs as distributed RAM or SRLs.
- Each DSP slice contains a pre-adder, a 25 x 18 multiplier, an adder, and an accumulator.
- Block RAMs are fundamentally 36 Kb in size; each block can also be used as two independent 18 Kb blocks.
- Each CMT contains one MMCM and one PLL.
- Artix-7 FPGA Interface Blocks for PCI Express support up to x4 Gen 2.
- Does not include configuration Bank 0.
- This number does not include GTP transceivers.

Flip-Flop (FF)

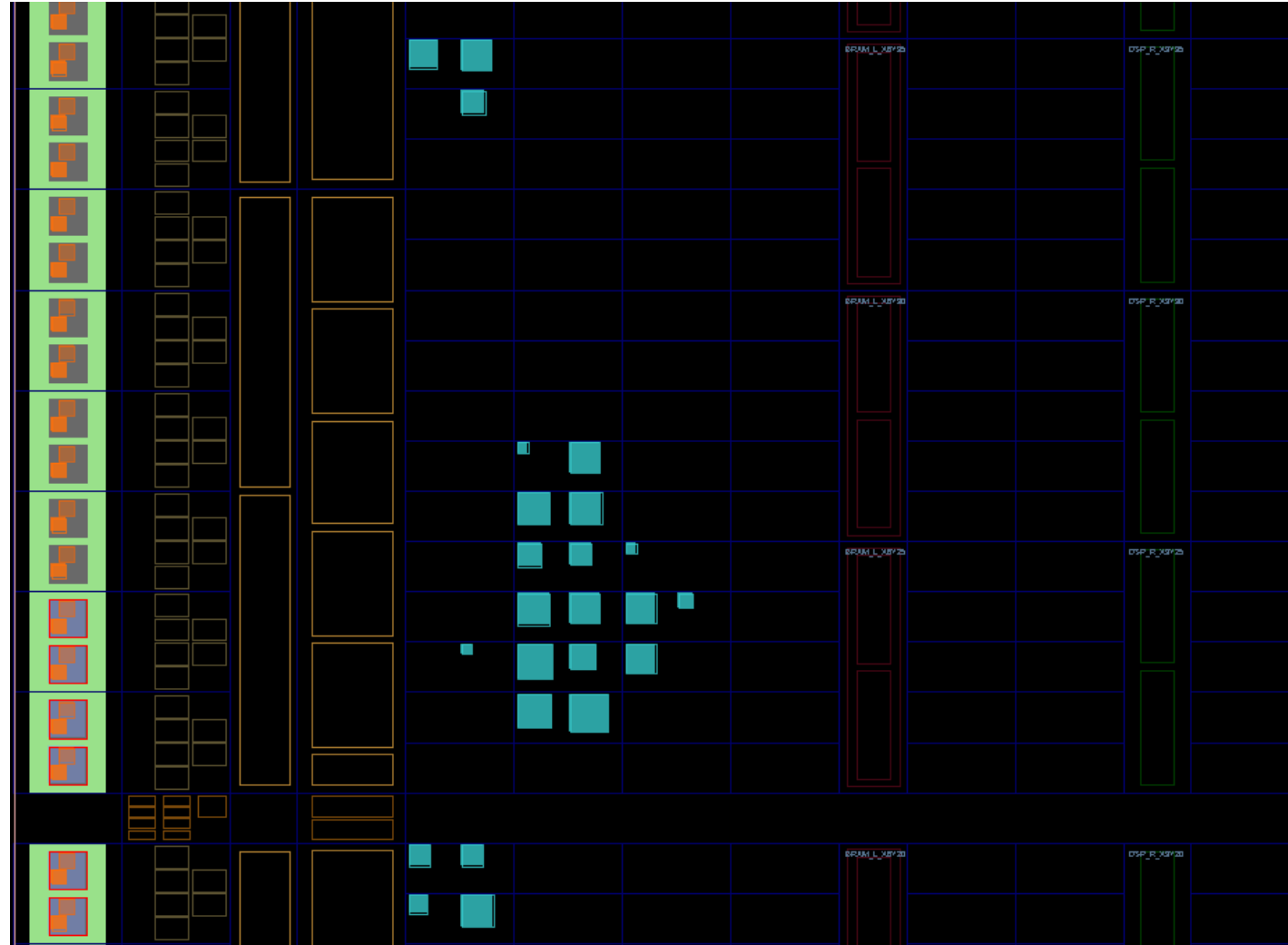


```
always @(posedge clk, negedge rst_n) begin
    if (rst_n == 1'b0) begin
        state                <= ASSISTANCE;
        gaz                  <= 1'b0;
        fren                 <= 1'b0;
        kirmizi_isik         <= 1'b0;
        yaya_gecidi          <= 1'b0;
        takip_mesafe         <= 1'b0;
        takip_mesafe_kaydedilen <= 50;
        hiz_kaydedilen        <= 100;
        for (i = 0; i < 3 ; i=i+1) begin
            olcum_buffer[i]    <= {8{1'b0}};
        end
    end
    else begin

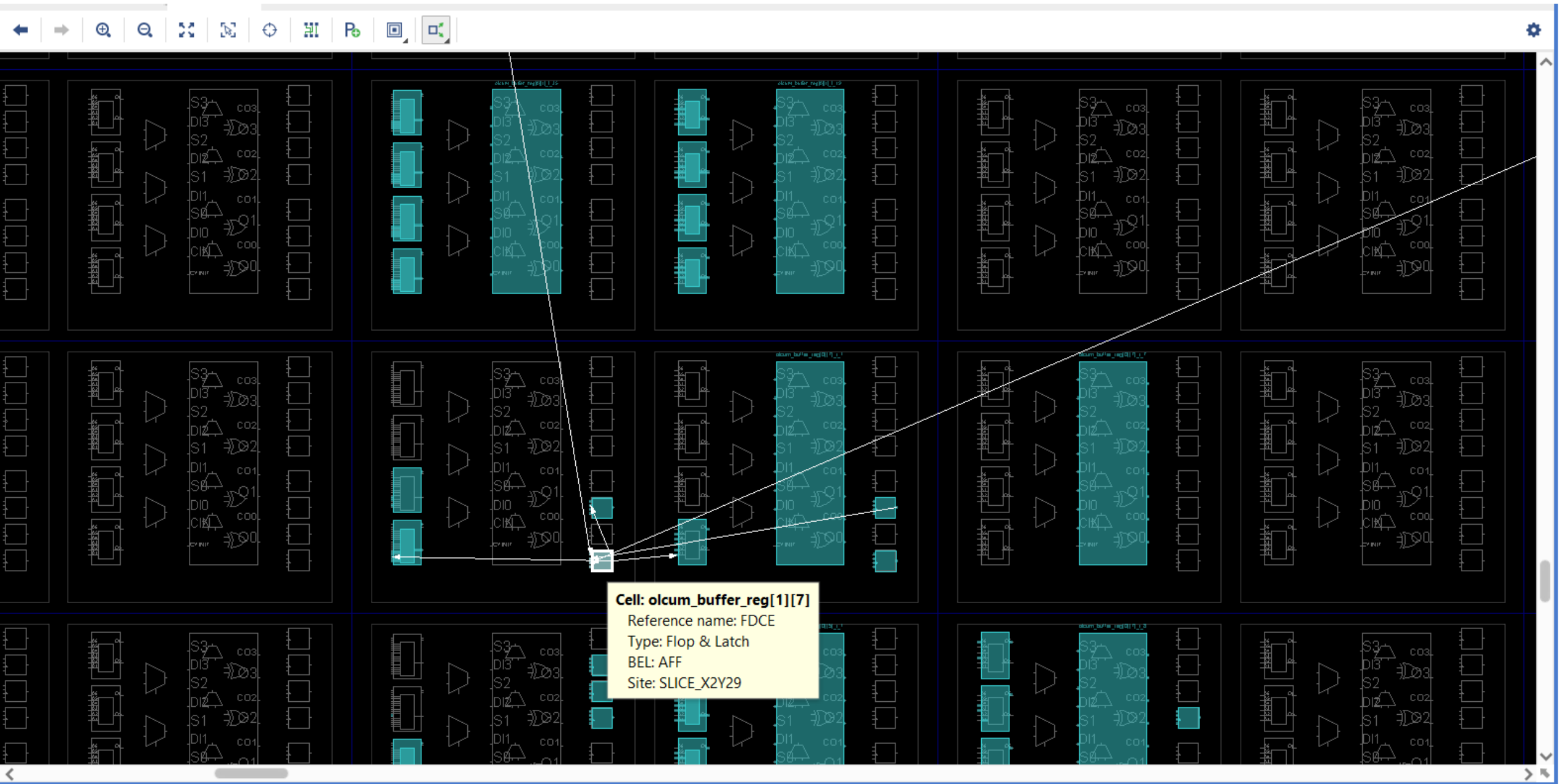
        olcum_buffer[2] <= olcum_buffer[1];
        olcum_buffer[1] <= olcum_buffer[0];
        olcum_buffer[0] <= olcum_ortalama;

        case (state)
            ASSISTANCE : begin
                if (timer_tick_i == 1'b1) begin

                    if (kirmizi_isik_var == 1'b1) begin
                        kirmizi_isik <= 1'b1;
                    end
                end
            end
        endcase
    end
end
```



Flip-Flop (FF)



Flip-Flop (FF)

Utilization

Post-Synthesis

| **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	64	20800	0.31
FF	47	41600	0.11
IO	53	106	50.00
BUFG	1	32	3.13

Distributed RAM (LUTRAM)

RAM_STYLE

RAM_STYLE instructs the Vivado synthesis tool on how to infer memory. Accepted values are:

- **block** : Instructs the tool to infer RAMB type components.
- **distributed** : Instructs the tool to infer the LUT RAMs.
- **registers** : Instructs the tool to infer registers instead of RAMs.
- **ultra** : Instructs the tool to use the UltraScale+ TM URAM primitives.
- **mixed** : Instructs the tool to infer a combination of RAM types designed to minimize the amount of space that is unused.
- **auto** : Lets the synthesis tool decide how to implement the RAM. This is the same as the default behavior. The main usage of this value is by XPMs that must choose a value for RAM_STYLE.

Distributed RAM (LUTRAM)

Table 2-3: Distributed RAM Configuration

RAM	Description	Primitive	Number of LUTs
32 x 1S	Single port	RAM32X1S	1
32 x 1D	Dual port	RAM32X1D	2
32 x 2Q	Quad port	RAM32M	4
32 x 6SDP	Simple dual port	RAM32M	4
64 x 1S	Single port	RAM64X1S	1
64 x 1D	Dual port	RAM64X1D	2
64 x 1Q	Quad port	RAM64M	4
64 x 3SDP	Simple dual port	RAM64M	4
128 x 1S	Single port	RAM128X1S	2
128 x 1D	Dual port	RAM128X1D	4
256 x 1S	Single port	RAM256X1S	4

Distributed RAM (LUTRAM)



```
module ram
#(
parameter RAM_WIDTH = 8,           // Specify RAM data width
parameter RAM_DEPTH = 256         // Specify RAM depth (number of entries)
)
(
input clk,                         // Clock
input we_i,                       // Write enable
input [clogb2(RAM_DEPTH)-1:0] addr_i, // Address bus, width determined from RAM_DEPTH
input [RAM_WIDTH-1:0] din_i,      // RAM input data
output [RAM_WIDTH-1:0] dout_o     // RAM output data
);
```

```
(* ram_style="distributed" *)
reg [RAM_WIDTH-1:0] BRAM [RAM_DEPTH-1:0];
reg [RAM_WIDTH-1:0] ram_data = {RAM_WIDTH{1'b0}};
```

```
always @(posedge clk) begin
// write
if (we_i == 1'b1) begin
BRAM[addr_i] <= din_i;
end
// read
ram_data <= BRAM[addr_i];
end
```

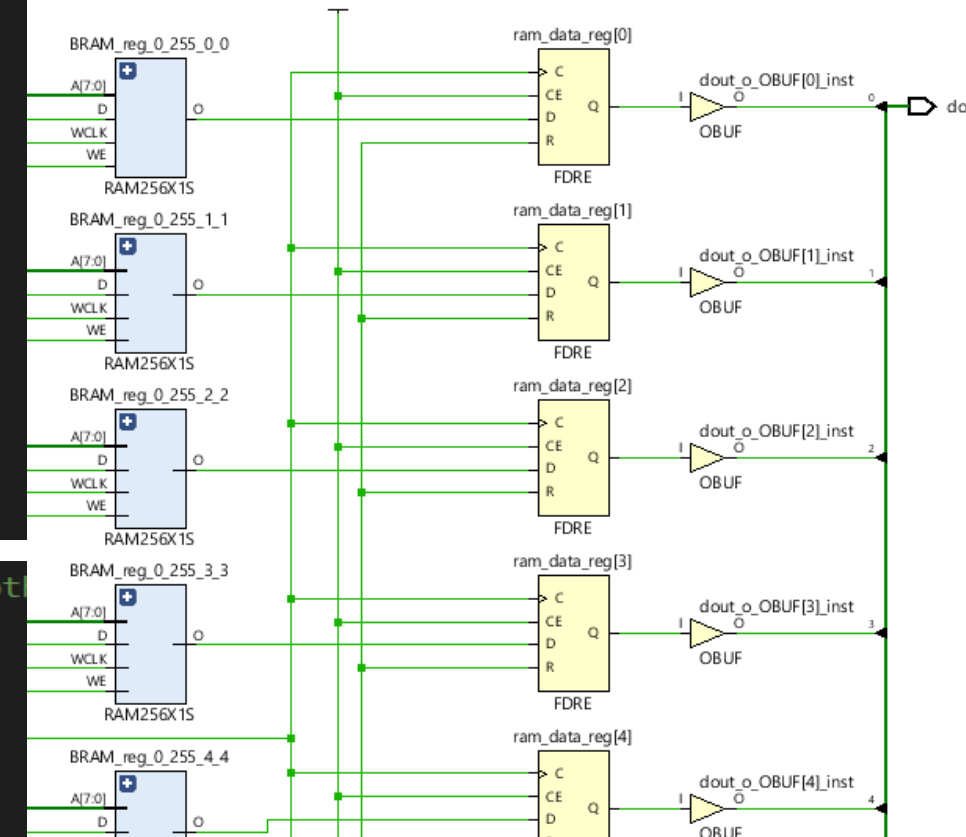
```
assign dout_o = ram_data;
```

```
// The following function calculates the address width based on specified RAM depth
```

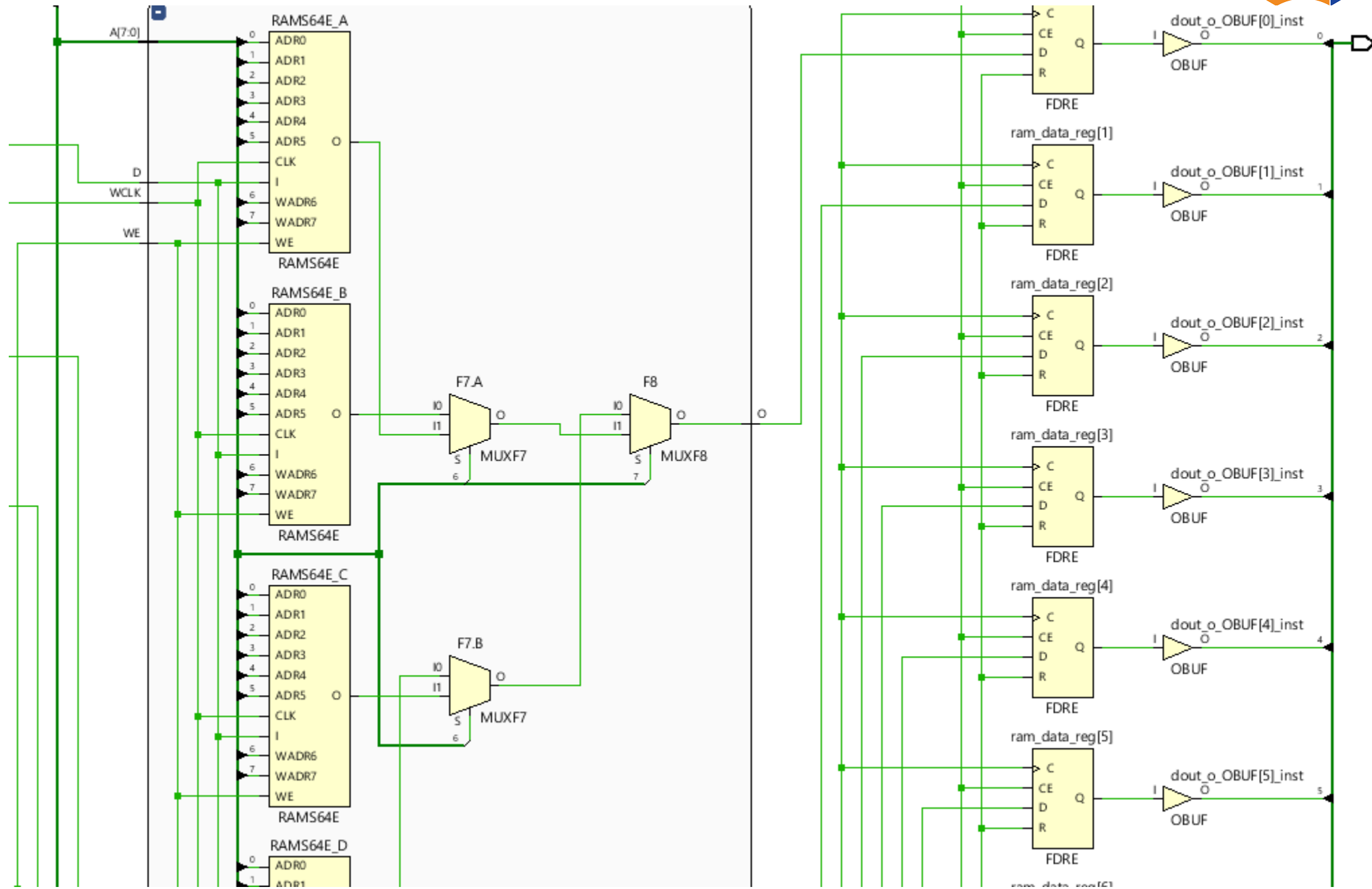
```
function integer clogb2;
input integer depth;
for (clogb2=0; depth>0; clogb2=clogb2+1)
depth = depth >> 1;
endfunction
```

```
endmodule
```

Resource	Estimation	Available	Utilization %
LUT	32	20800	0.15
LUTRAM	32	9600	0.33
FF	8	41600	0.02
IO	26	106	24.53
BUFG	1	32	3.13



Distributed RAM (LUTRAM)



Distributed RAM (LUTRAM)

Table 2-3: Distributed RAM Configuration

RAM	Description	Primitive	Number of LUTs
32 x 1S	Single port	RAM32X1S	1
32 x 1D	Dual port	RAM32X1D	2
32 x 2Q	Quad port	RAM32M	4
32 x 6SDP	Simple dual port	RAM32M	4
64 x 1S	Single port	RAM64X1S	1
64 x 1D	Dual port	RAM64X1D	2
64 x 1Q	Quad port	RAM64M	4
64 x 3SDP	Simple dual port	RAM64M	4
128 x 1S	Single port	RAM128X1S	2
128 x 1D	Dual port	RAM128X1D	4
256 x 1S	Single port	RAM256X1S	4

```
module ram
#(
parameter RAM_WIDTH = 8,
parameter RAM_DEPTH = 4
)
```

Resource	Estimation	Available	Utilization %
LUT	8	20800	0.04
LUTRAM	8	9600	0.08
FF	8	41600	0.02
IO	20	106	18.87
BUFG	1	32	3.13

Block RAM

7 Series FPGAs Memory Resources User Guide (UG473)

The block RAM in Xilinx® 7 series FPGAs stores up to 36 Kbits of data and can be configured as either two independent 18 Kb RAMs, or one 36 Kb RAM. Each 36 Kb block RAM can be configured as a 64K x 1 (when cascaded with an adjacent 36 Kb block RAM), 32K x 1, 16K x 2, 8K x 4, 4K x 9, 2K x 18, 1K x 36, or 512 x 72 in simple dual-port mode. Each 18 Kb block RAM can be configured as a 16K x 1, 8K x 2, 4K x 4, 2K x 9, 1K x 18 or 512 x 36 in simple dual-port mode.

Read Operation

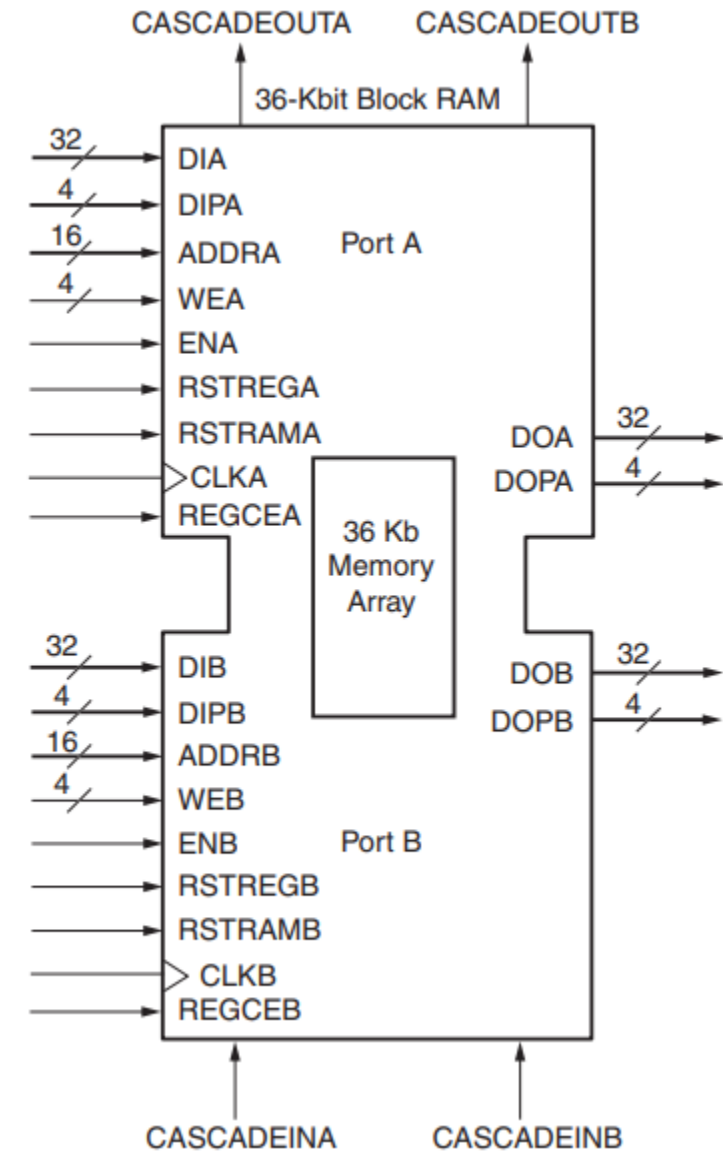
In latch mode, the read operation uses one clock edge. The read address is registered on the read port, and the stored data is loaded into the output latches after the RAM access time. When using the output register, the read operation takes one extra latency cycle.

Write Operation

A write operation is a single clock-edge operation. The write address is registered on the write port, and the data input is stored in memory.

Write Modes

Three settings of the write mode determines the behavior of the data available on the output latches after a write clock edge: WRITE_FIRST, READ_FIRST, and NO_CHANGE. Write mode selection is set by configuration. The Write mode attribute can be individually selected for each port. The default mode is WRITE_FIRST. WRITE_FIRST outputs the newly written data onto the output bus. READ_FIRST outputs the previously stored data while new data is being written. NO_CHANGE maintains the output previously generated by a read operation.



Block RAM



```
module ram
#(
parameter RAM_WIDTH = 8,           // Specify RAM data width
parameter RAM_DEPTH = 256         // Specify RAM depth (number of entries)
)
(
input clk,                          // Clock
input we_i,                        // Write enable
input [clogb2(RAM_DEPTH)-1:0] addr_i, // Address bus, width determined from RAM_DEPTH
input [RAM_WIDTH-1:0] din_i,       // RAM input data
output [RAM_WIDTH-1:0] dout_o      // RAM output data
);

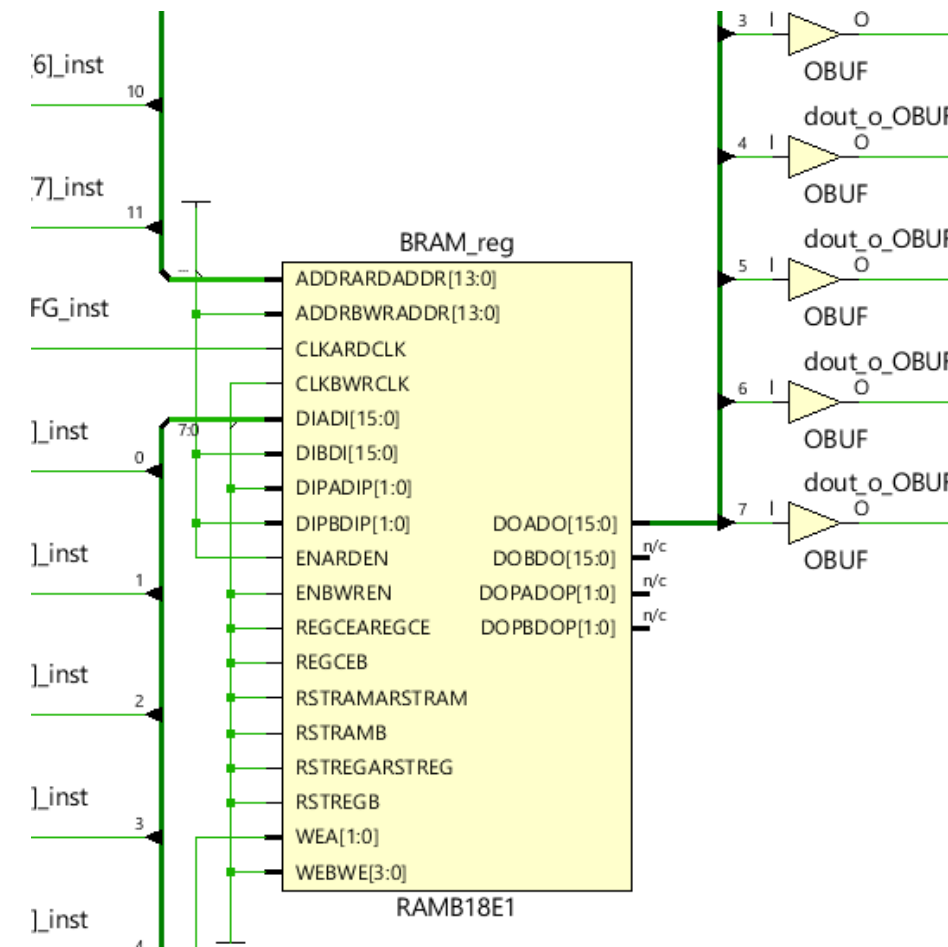
(* ram_style="block" *)
reg [RAM_WIDTH-1:0] BRAM [RAM_DEPTH-1:0];
reg [RAM_WIDTH-1:0] ram_data = {RAM_WIDTH{1'b0}};

always @(posedge clk) begin
    // write
    if (we_i == 1'b1) begin
        BRAM[addr_i] <= din_i;
    end
    // read
    ram_data <= BRAM[addr_i];
end

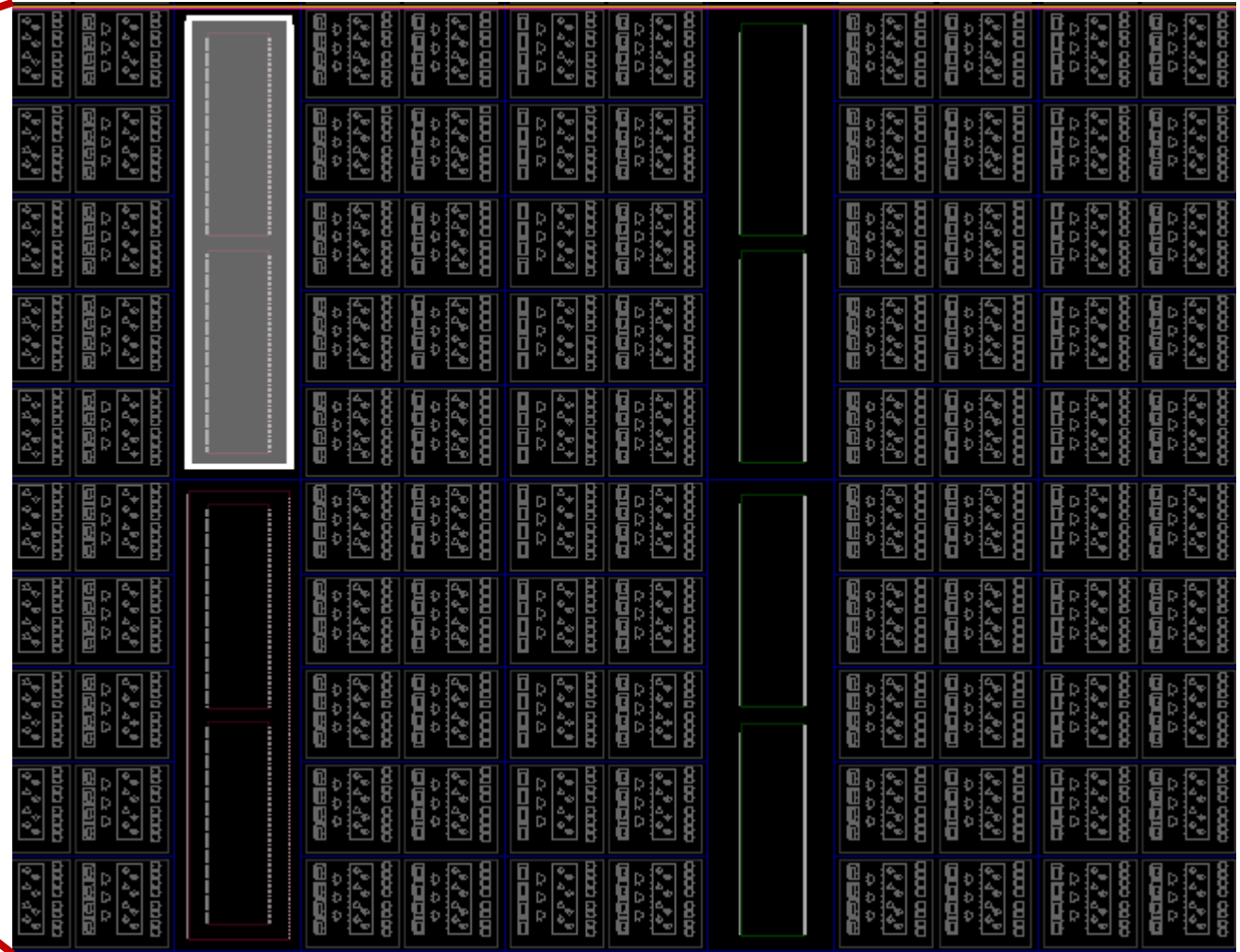
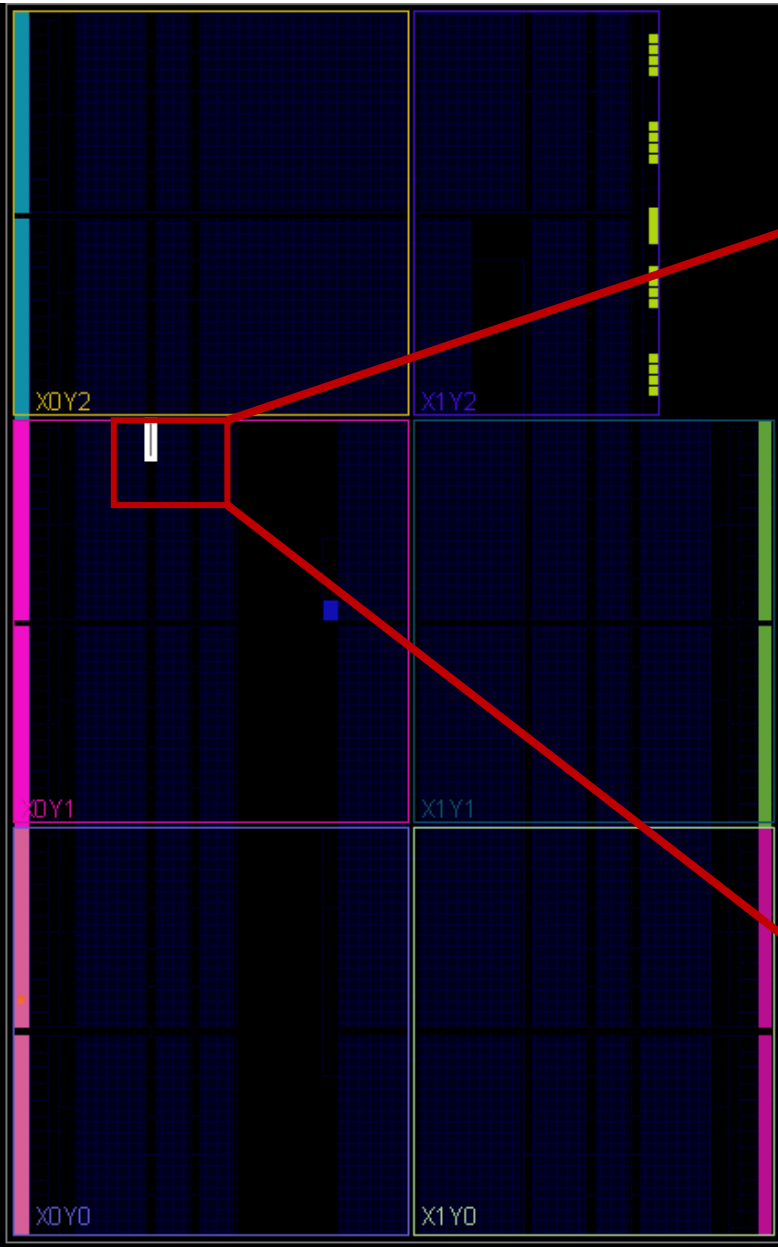
assign dout_o = ram_data;
endmodule
```

```
// The following function calculates the address width based on specified RAM depth
function integer clogb2;
input integer depth;
for (clogb2=0; depth>0; clogb2=clogb2+1)
    depth = depth >> 1;
endfunction
```

Resource	Estimation	Available	Utilization %
BRAM	0.50	50	1.00
IO	26	106	24.53
BUFG	1	32	3.13



Block RAM



FPGA'DA BELLEK ÖRNEĞİ

4 KB'lık bir bellek alanına sahip bir sistemde:

- İlk 1 KB private alan olarak tanımlı, yazma ve okuma yasak
- Sonraki 1 KB public read-only alan olarak tanımlı, sadece okuma yapılabililiyor
- Sonraki 2 KB public write-only alan olarak tanımlı, sadece yazma yapılabililiyor

0x000	PRIVATE
...	
0x3FF	
0x400	READ-ONLY
...	
0x7FF	
0x800	WRITE-ONLY
...	
0xFFFF	

```
module ram_ornek
#(
parameter RAM_WIDTH = 8,
parameter RAM_DEPTH = 4096
)
(
input clk,
input rst_n,
input yaz_komut_i,
input [RAM_WIDTH-1:0] yaz_veri_i,
input [clogb2(RAM_DEPTH)-1:0] yaz_adres_i,
input oku_komut_i,
output [RAM_WIDTH-1:0] oku_veri_o,
input [clogb2(RAM_DEPTH)-1:0] oku_adres_i,
output yaz_hata_o,
output oku_hata_o
);
```

```

reg we;
reg [clogb2(RAM_DEPTH)-1:0] addr;
reg [RAM_WIDTH-1:0] din;
wire [RAM_WIDTH-1:0] dout;

```

```

reg yaz_komut_ff;
reg oku_komut_ff;
reg yaz_hata;
reg oku_hata;

```

```

ram
#(
    .RAM_WIDTH(RAM_WIDTH),
    .RAM_DEPTH(RAM_DEPTH)
)
ram_i
(
    .clk      (clk),
    .we_i     (we),
    .addr_i   (addr),
    .din_i    (din),
    .dout_o   (dout)
);

```

```

always @(posedge clk, negedge rst_n) begin

```

```

    if (rst_n == 1'b0) begin
        we      <= 1'b0;
        addr    <= {clogb2(RAM_DEPTH-1){1'b0}};
        din     <= {RAM_WIDTH{1'b0}};
        yaz_komut_ff <= 1'b0;
        oku_komut_ff <= 1'b0;
        yaz_hata  <= 1'b0;
        oku_hata  <= 1'b0;
    end

```

```

    else begin

```

```

        yaz_hata  <= 1'b0;
        oku_hata  <= 1'b0;
        we        <= 1'b0;

```

```

        yaz_komut_ff <= yaz_komut_i;
        oku_komut_ff <= oku_komut_i;

```

```

        if (yaz_komut_ff == 1'b0 && yaz_komut_i == 1'b1) begin
            if (yaz_adres_i < 12'h800) begin
                yaz_hata  <= 1'b1;
            end
            else begin
                we      <= 1'b1;
                addr    <= yaz_adres_i;
                din     <= yaz_veri_i;
            end
        end

```

```

    end

```

```

        else if (oku_komut_ff == 1'b0 && oku_komut_i == 1'b1) begin
            if (oku_adres_i < 12'h400 || oku_adres_i > 12'h7FF) begin
                oku_hata  <= 1'b1;
            end
            else begin
                addr    <= oku_adres_i;
            end
        end

```

```

    end

```

```

end

```

```

assign oku_veri_o = dout;
assign yaz_hata_o = yaz_hata;
assign oku_hata_o = oku_hata;

```

// The following function calculates the address width based on speci

```

function integer clogb2;
input integer depth;
    for (clogb2=0; depth>0; clogb2=clogb2+1)
        depth = depth >> 1;
endfunction

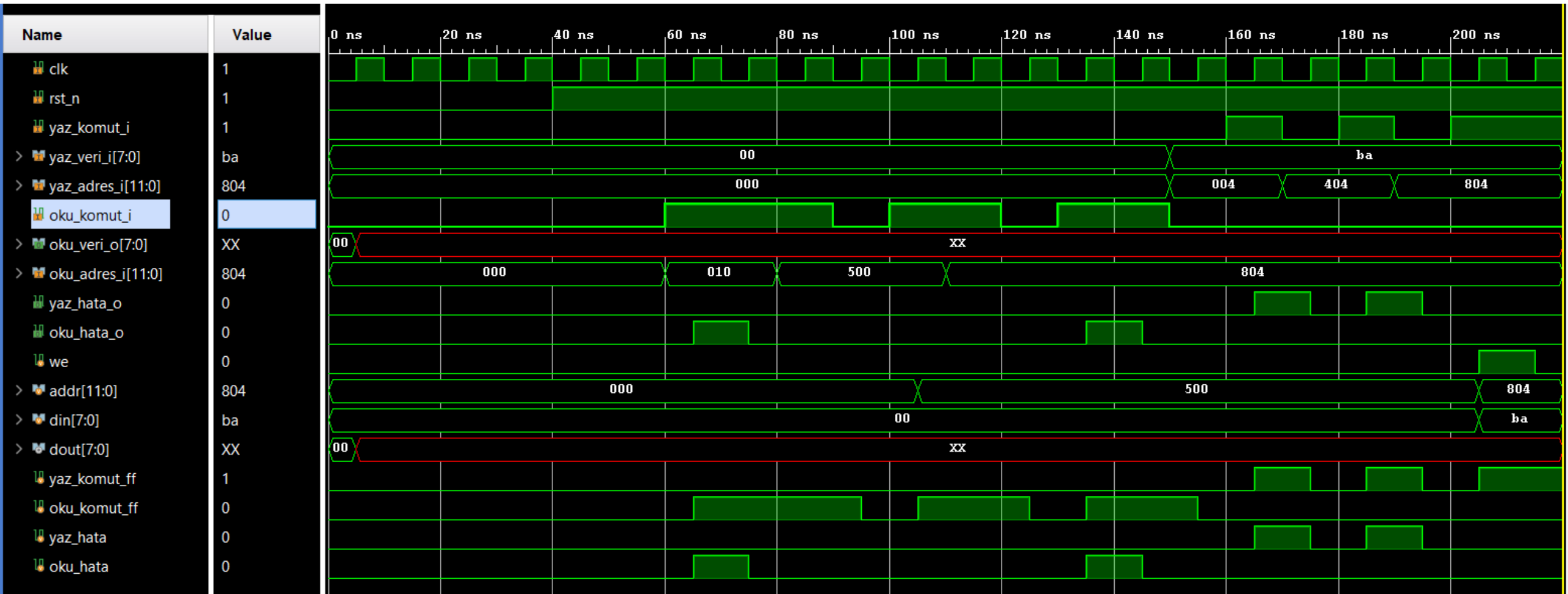
```

```

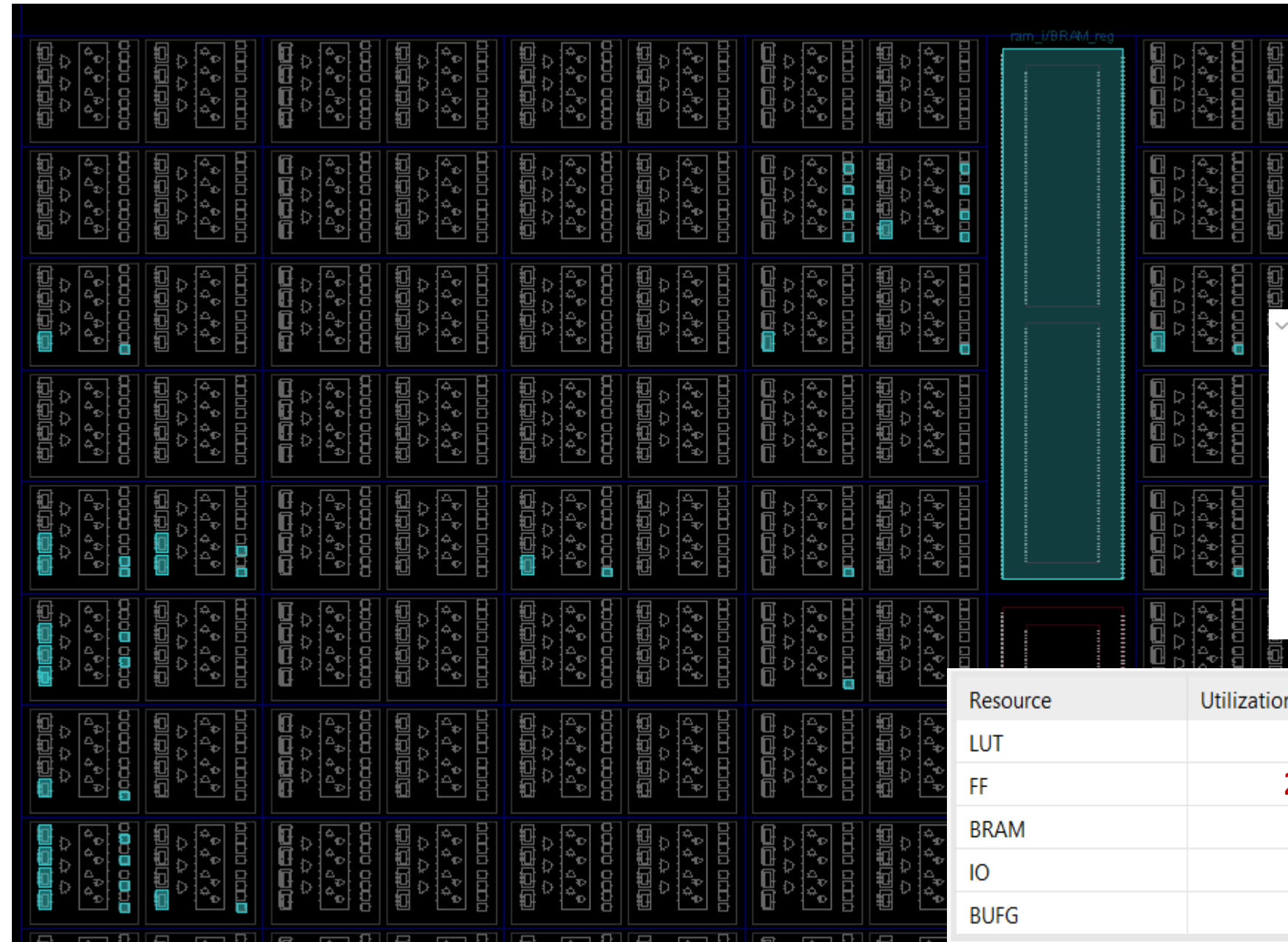
endmodule

```

FPGA'DA BELLEK ÖRNEĞİ - SİMÜLASYON



FPGA'DA BELLEK ÖRNEĞİ – P&R



- Leaf Cells (9)
- BRAM_reg (RAMB36E1) ?
 - BRAM_reg_ENARDEN_coolgate_en_gate_3_LOPT_REMAP (LUT4)
 - GND (GND)
 - GND_1 (GND)
 - GND_coolDelFlop (GND) ?
 - ram_i/BRAM_reg_coolgate_en_gate_1_coolDelFlop (FDCE)
 - ram_i/BRAM_reg_coolgate_en_gate_2_coolDelFlop (FDCE)
 - VCC (VCC)
 - VCC_coolDelFlop (VCC) ?


Resource	Utilization	Available	Utilization %
LUT	18	20800	0.09
FF	25 ?	41600	0.06
BRAM	1	50	2.00
IO	46	106	43.40
BUFG	1	32	3.13

OpenRAM (ASIC RAM MACRO)

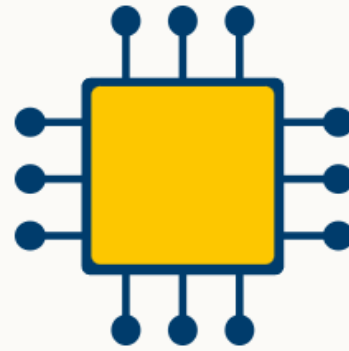


OpenRAM

An open-source static random access memory (SRAM) compiler.

 [View On GitHub](#)

This project is maintained by [VLSIDA](#)



OpenRAM

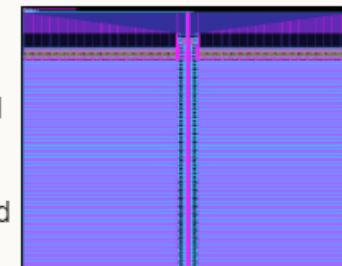
OpenRAM

Python 3.5 License BSD 3-Clause download stable download unstable

An open-source static random access memory (SRAM) compiler.

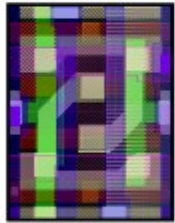
What is OpenRAM?

OpenRAM is an award winning open-source Python framework to create the layout, netlists, timing and power models, placement and routing models, and other views necessary to use SRAMs in ASIC design. OpenRAM supports integration in both commercial and open-source flows with both predictive and fabricable technologies.

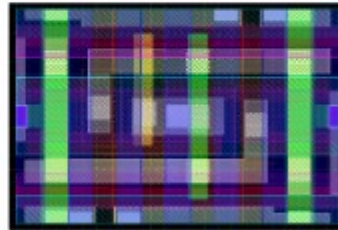


OpenRAM (ASIC RAM MACRO)

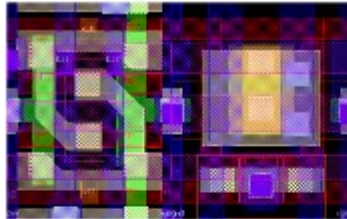
“Thin” SRAM Bitcells (130nm)



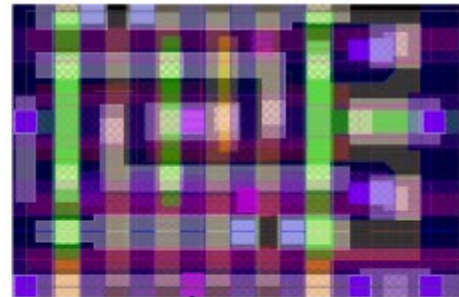
Single Port
1.2um x 1.58um



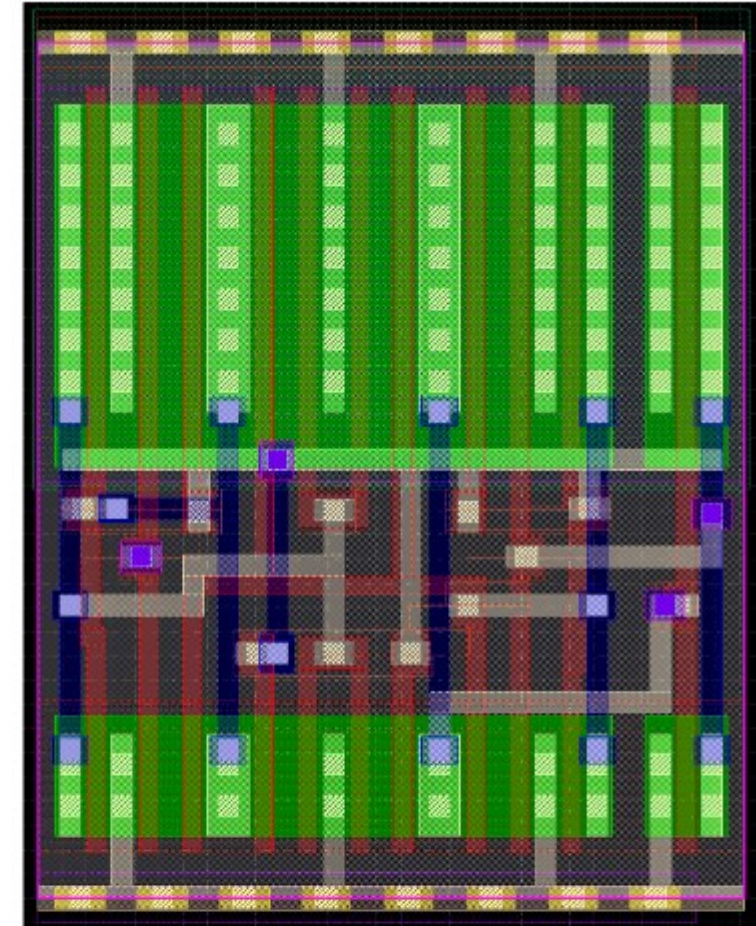
Dual Port
2.40um x 1.58um



Single Port
(w/ straps & taps)
2.49um x 1.58um



Dual Port
(w/ straps & taps)
3.12um x 1.97um



DFF (for reference)
5.83um x 7.07 um

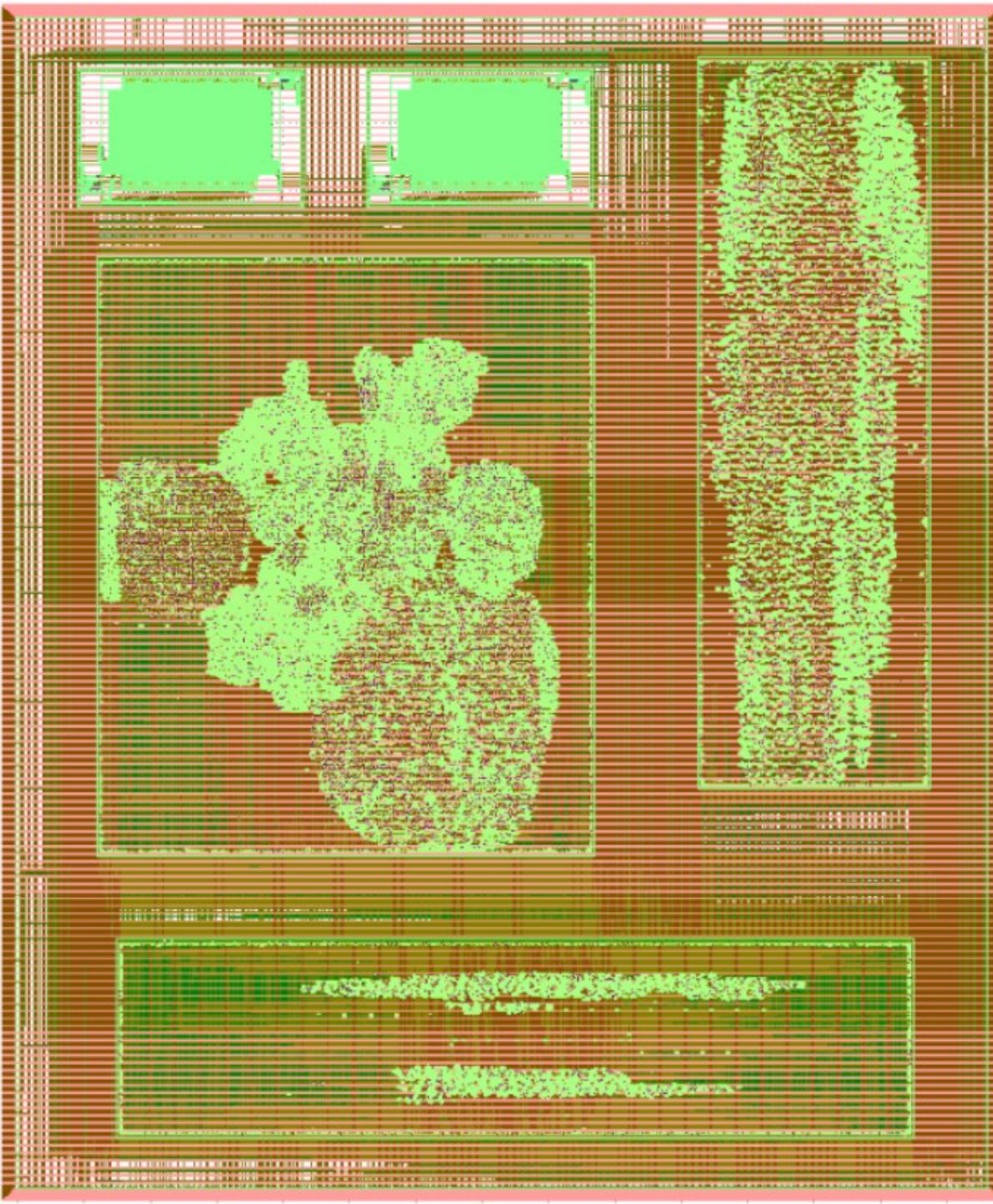
Actual cells from SKY130!

OpenRAM (ASIC RAM MACRO)

```
MACRO sky130_sram_2kbyte_1rw1r_32x512_8
```

```
CLASS BLOCK ;
```

```
SIZE 683.1 BY 416.54 ;
```



OpenRAM (ASIC RAM MACRO)



```
// OpenRAM SRAM model
// Words: 512
// Word size: 32
// Write size: 8

module sky130_sram_2kbyte_1rw1r_32x512_8(
`ifdef USE_POWER_PINS
    vccd1,
    vssd1,
`endif
    // Port 0: RW
    clk0, csb0, web0, wmask0, addr0, din0, dout0,
    // Port 1: R
    clk1, csb1, addr1, dout1
);

parameter NUM_WMASKS = 4 ;
parameter DATA_WIDTH = 32 ;
parameter ADDR_WIDTH = 9 ;
parameter RAM_DEPTH = 1 << ADDR_WIDTH;
// FIXME: This delay is arbitrary.
parameter DELAY = 3 ;
parameter VERBOSE = 1 ; //Set to 0 to only display warnings
parameter T_HOLD = 1 ; //Delay to hold dout value after posedge. Value is arbitrary
```

```
`ifdef USE_POWER_PINS
    inout vccd1;
    inout vssd1;
`endif

input  clk0; // clock
input  csb0; // active low chip select
input  web0; // active low write control
input [NUM_WMASKS-1:0] wmask0; // write mask
input [ADDR_WIDTH-1:0] addr0;
input [DATA_WIDTH-1:0] din0;
output [DATA_WIDTH-1:0] dout0;
input  clk1; // clock
input  csb1; // active low chip select
input [ADDR_WIDTH-1:0] addr1;
output [DATA_WIDTH-1:0] dout1;

reg  csb0_reg;
reg  web0_reg;
reg [NUM_WMASKS-1:0] wmask0_reg;
reg [ADDR_WIDTH-1:0] addr0_reg;
reg [DATA_WIDTH-1:0] din0_reg;
reg [DATA_WIDTH-1:0] dout0;
```

OpenRAM (ASIC RAM MACRO)



```
// All inputs are registers
always @(posedge clk0)
begin
    csb0_reg = csb0;
    web0_reg = web0;
    wmask0_reg = wmask0;
    addr0_reg = addr0;
    din0_reg = din0;
    #(T_HOLD) dout0 = 32'bx;
    if ( !csb0_reg && web0_reg && VERBOSE )
        $display($time," Reading %m addr0=%b dout0=%b",addr0_reg,mem[addr0_reg]);
    if ( !csb0_reg && !web0_reg && VERBOSE )
        $display($time," Writing %m addr0=%b din0=%b wmask0=%b",addr0_reg,din0_reg,wmask0_reg);
end

reg csb1_reg;
reg [ADDR_WIDTH-1:0] addr1_reg;
reg [DATA_WIDTH-1:0] dout1;

// All inputs are registers
always @(posedge clk1)
begin
    csb1_reg = csb1;
    addr1_reg = addr1;
    if (!csb0 && !web0 && !csb1 && (addr0 == addr1))
        $display($time," WARNING: Writing and reading addr0=%b and addr1=%b simultaneously!",addr0,addr1);
    #(T_HOLD) dout1 = 32'bx;
    if ( !csb1_reg && VERBOSE )
        $display($time," Reading %m addr1=%b dout1=%b",addr1_reg,mem[addr1_reg]);
end

reg [DATA_WIDTH-1:0] mem [0:RAM_DEPTH-1];
```


OpenRAM (ASIC RAM MACRO)



```
// Memory Write Block Port 0
// Write Operation : When web0 = 0, csb0 = 0
always @ (negedge clk0)
begin : MEM_WRITE0
    if ( !csb0_reg && !web0_reg ) begin
        if (wmask0_reg[0])
            mem[addr0_reg][7:0] = din0_reg[7:0];
        if (wmask0_reg[1])
            mem[addr0_reg][15:8] = din0_reg[15:8];
        if (wmask0_reg[2])
            mem[addr0_reg][23:16] = din0_reg[23:16];
        if (wmask0_reg[3])
            mem[addr0_reg][31:24] = din0_reg[31:24];
    end
end

// Memory Read Block Port 0
// Read Operation : When web0 = 1, csb0 = 0
always @ (negedge clk0)
begin : MEM_READ0
    if (!csb0_reg && web0_reg)
        dout0 <= #(DELAY) mem[addr0_reg];
end

// Memory Read Block Port 1
// Read Operation : When web1 = 1, csb1 = 0
always @ (negedge clk1)
begin : MEM_READ1
    if (!csb1_reg)
        dout1 <= #(DELAY) mem[addr1_reg];
end

endmodule
```