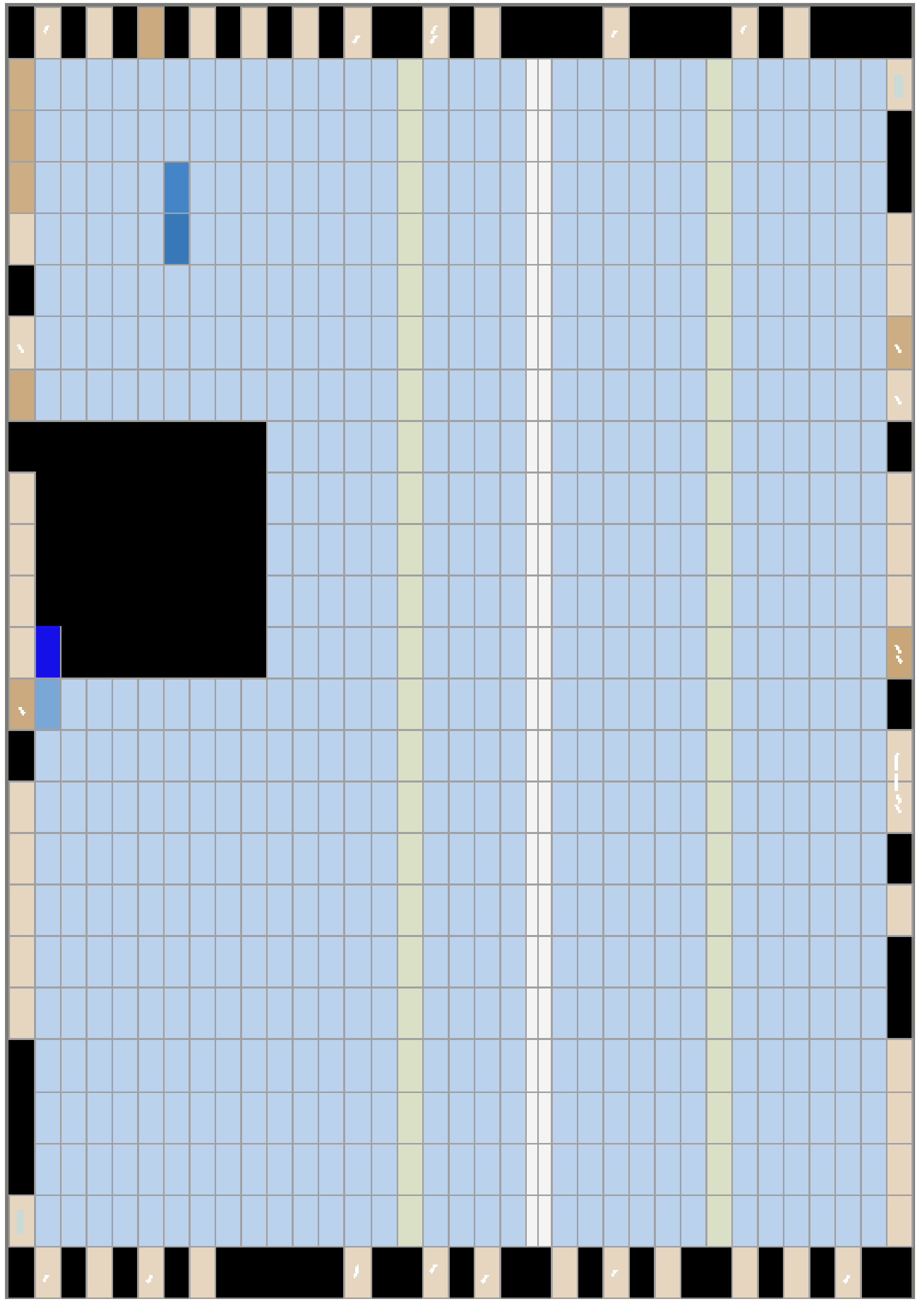
Память

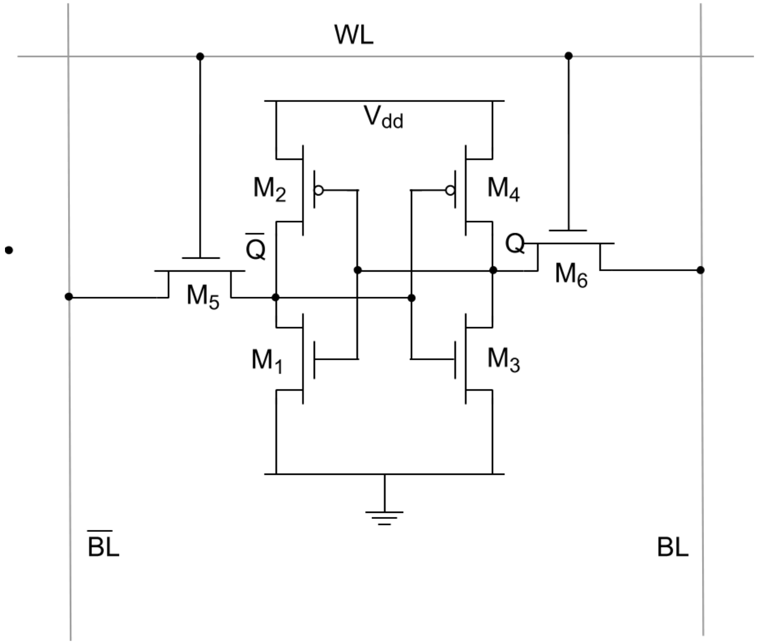
Что будем разбирать

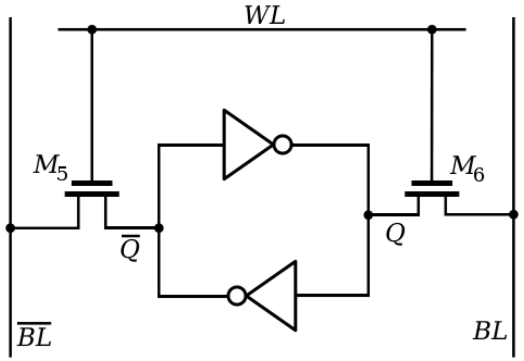
- Flash для прошивки
- M9K (режимы работы ROM, RAM, SR, FIFO)
- Внешняя RAM

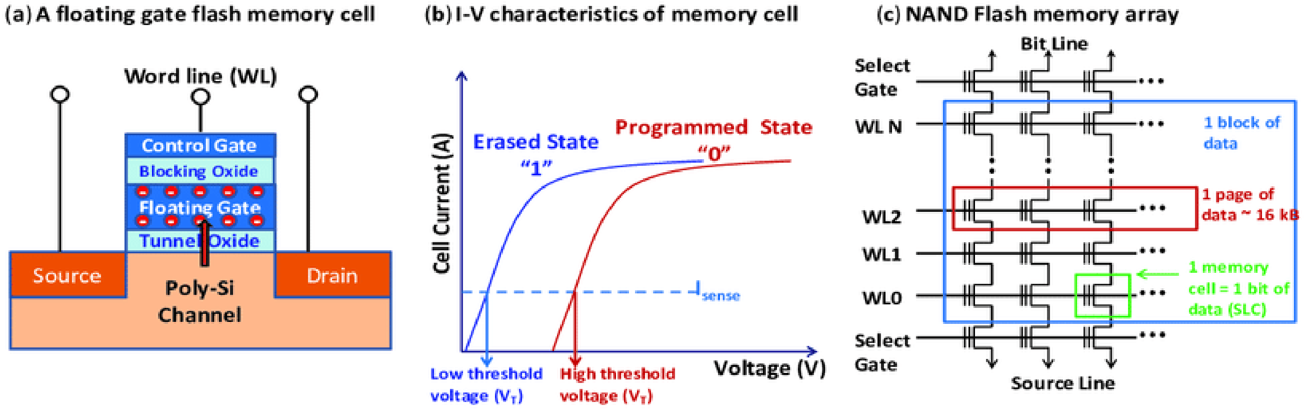


Прошивка ПЛИС

- SOF file – конфигурирует SRAM (полупроводниковая память, энергозависимая, регенерация не нужна)
- JIC file – конфигурирует flash (транзистор с плавающим затвором, энергонезависимая, регенерация не нужна)

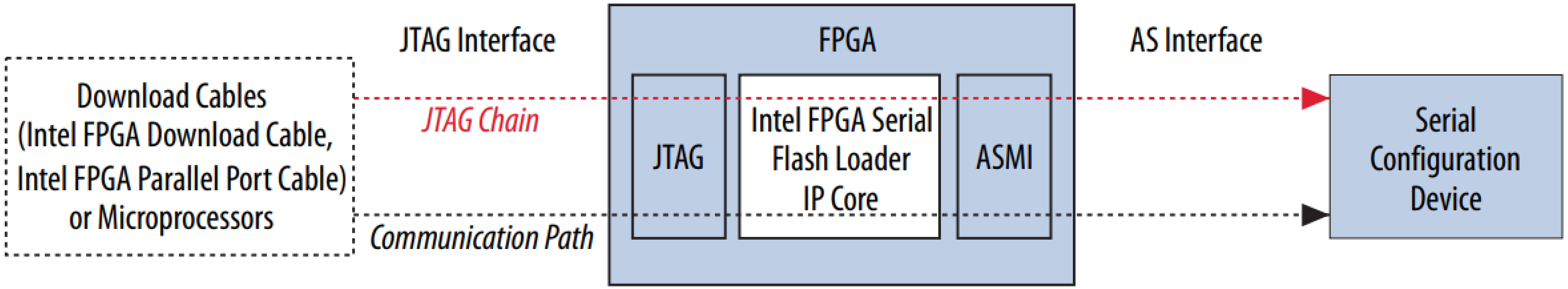






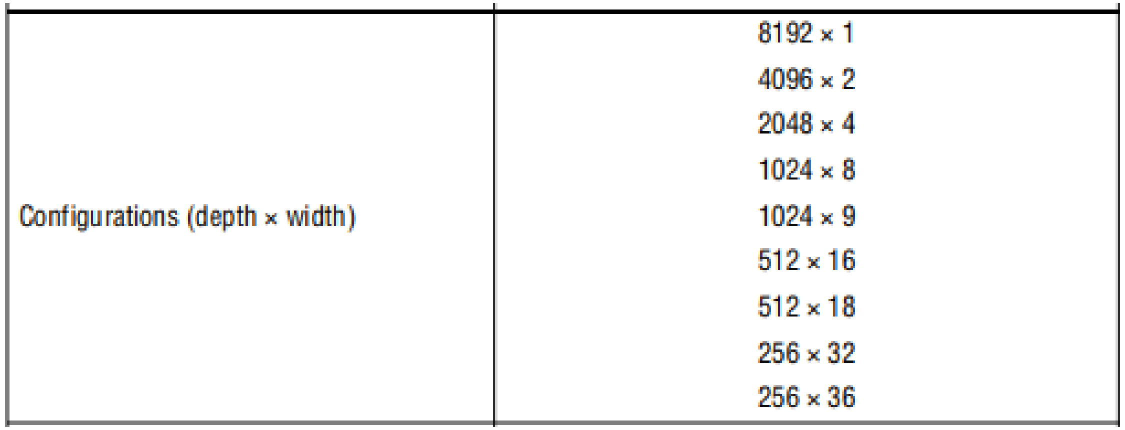
Конфигурация flash

- Implement altera serial flash loader ip
- Convert configuration file to .jic format
- Configure JTAG chain
- Configure memory



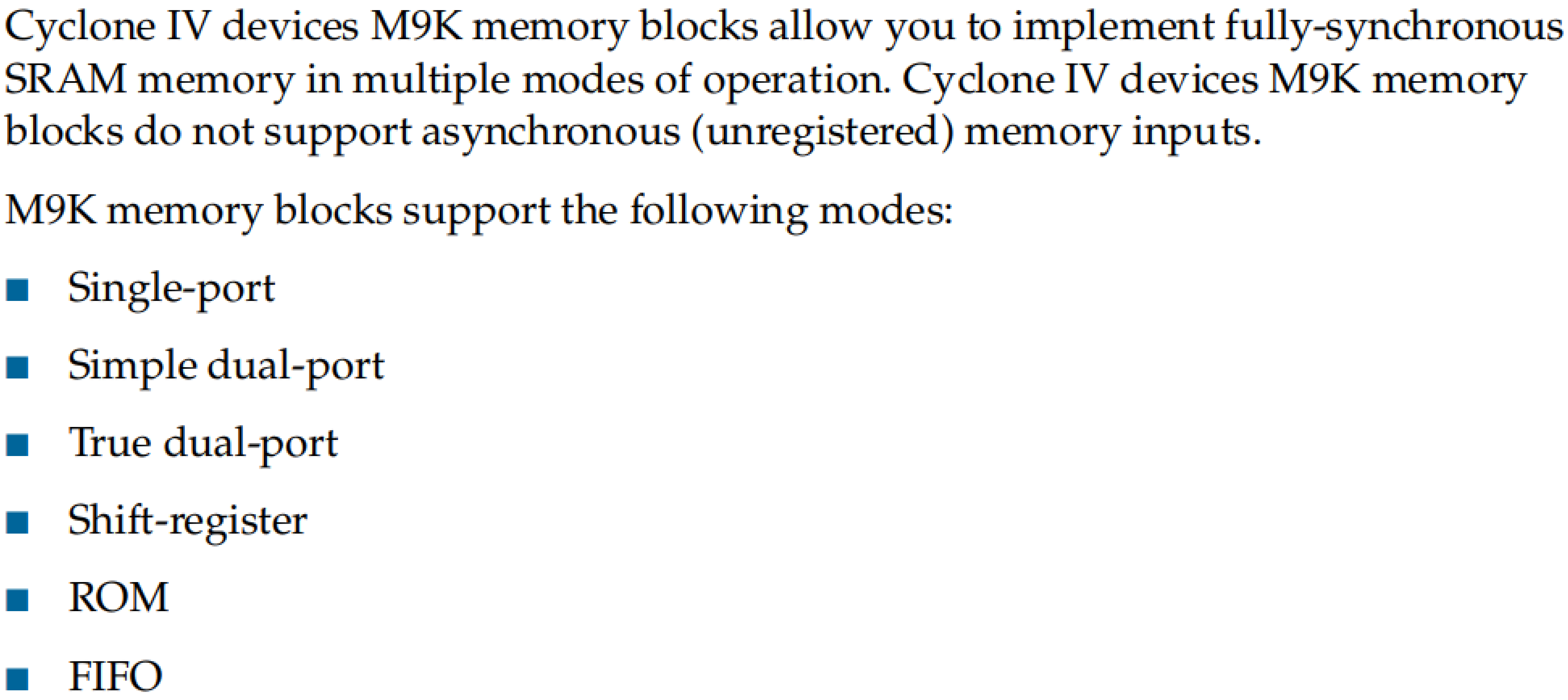
M9K

- SOF file – конфигурирует SRAM (полупроводниковая память, энергозависимая, регенерация не нужна)
- JIC file – конфигурирует flash (транзистор с плавающим затвором, энергонезависимая, регенерация не нужна)



Configurations (depth × width)	8192 × 1 4096 × 2 2048 × 4 1024 × 8 1024 × 9 512 × 16 512 × 18 256 × 32 256 × 36

M9K modes



Cyclone IV devices M9K memory blocks allow you to implement fully-synchronous SRAM memory in multiple modes of operation. Cyclone IV devices M9K memory blocks do not support asynchronous (unregistered) memory inputs.

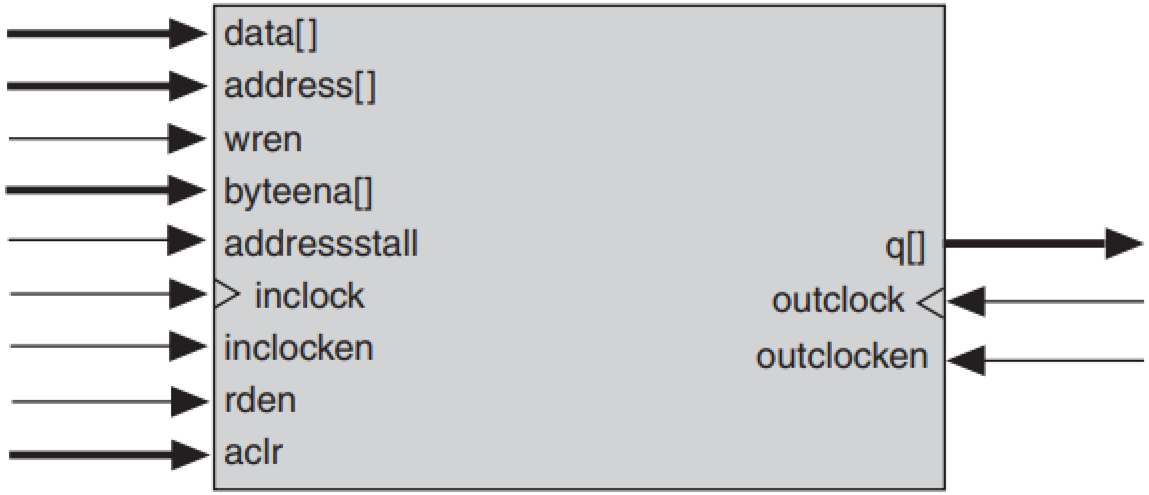
M9K memory blocks support the following modes:

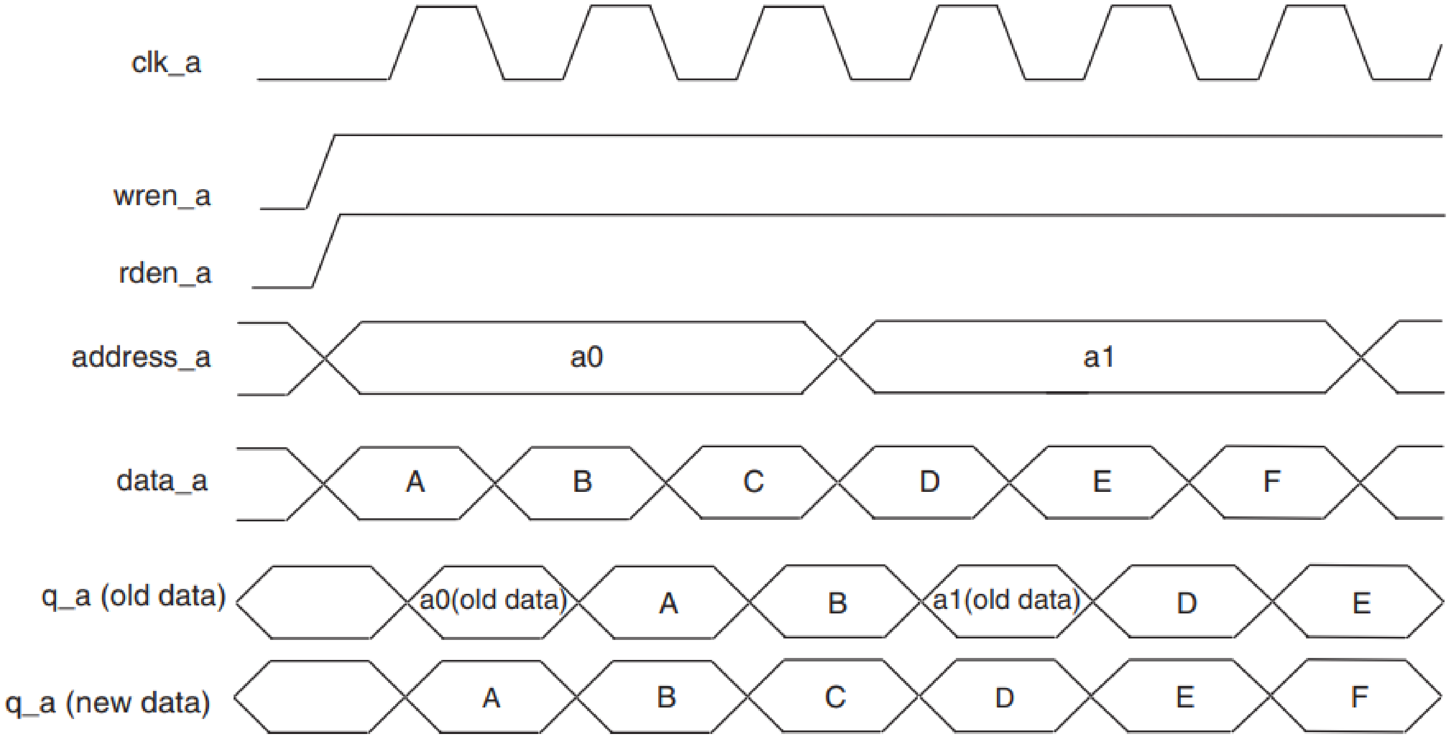
- Single-port
- Simple dual-port
- True dual-port
- Shift-register
- ROM
- FIFO

M9K implementatin

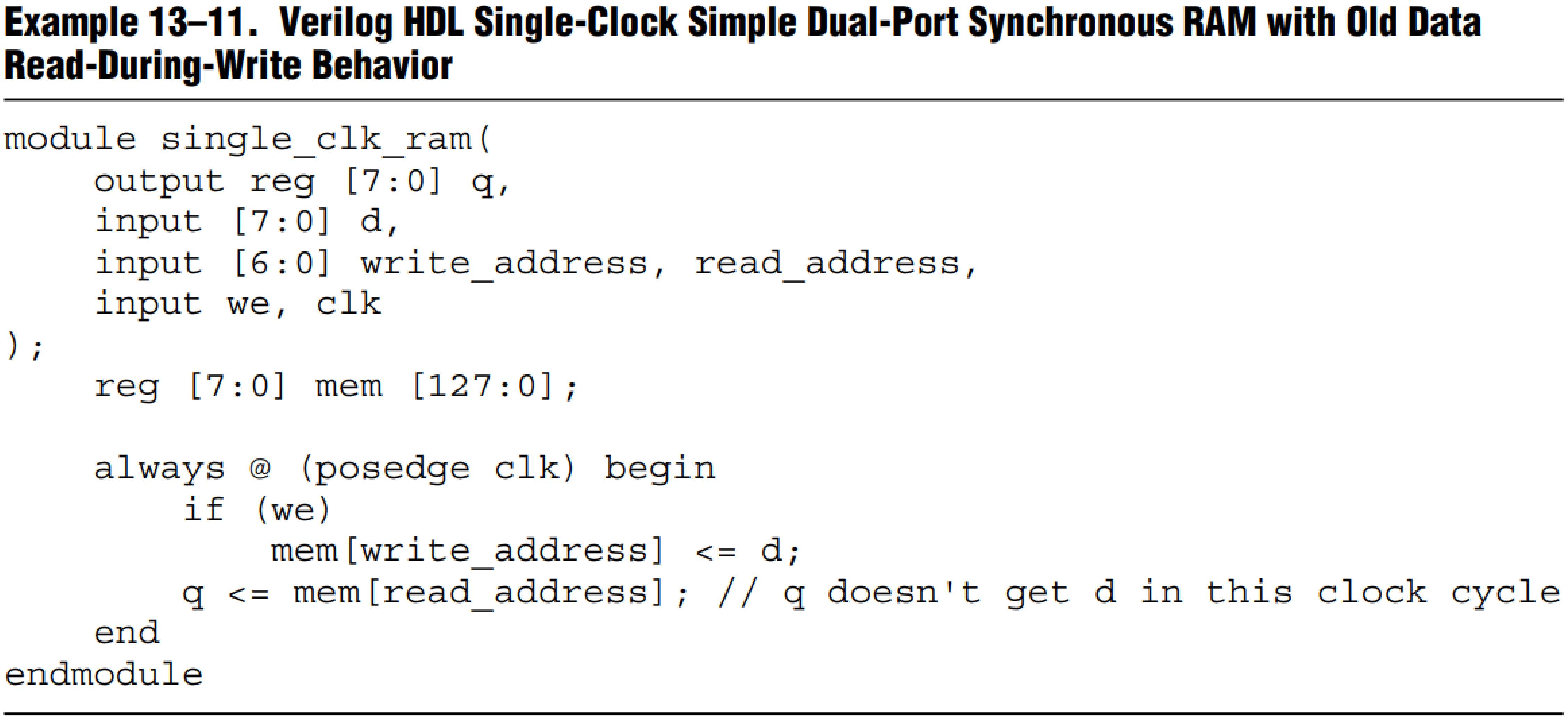
- При помощи IP каталога
- Через HDL код

M9K Single-Port





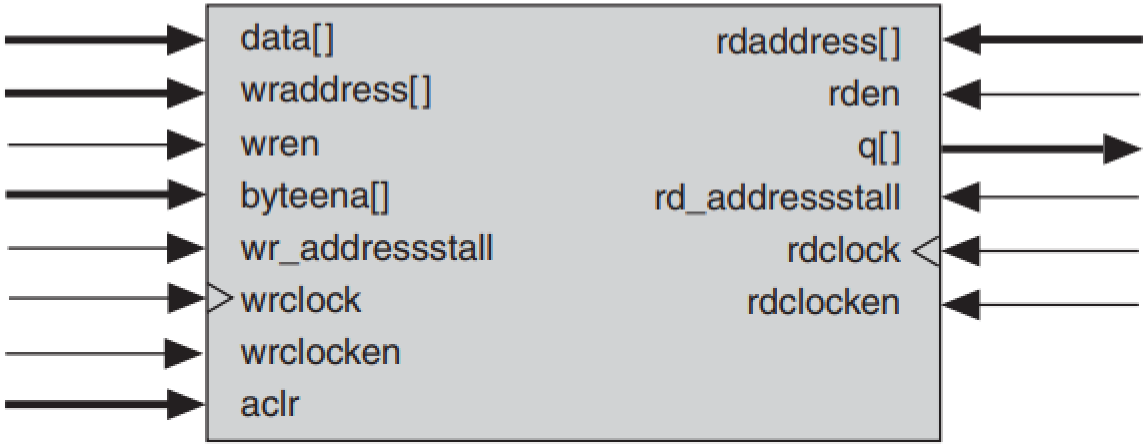
M9K Single-Port

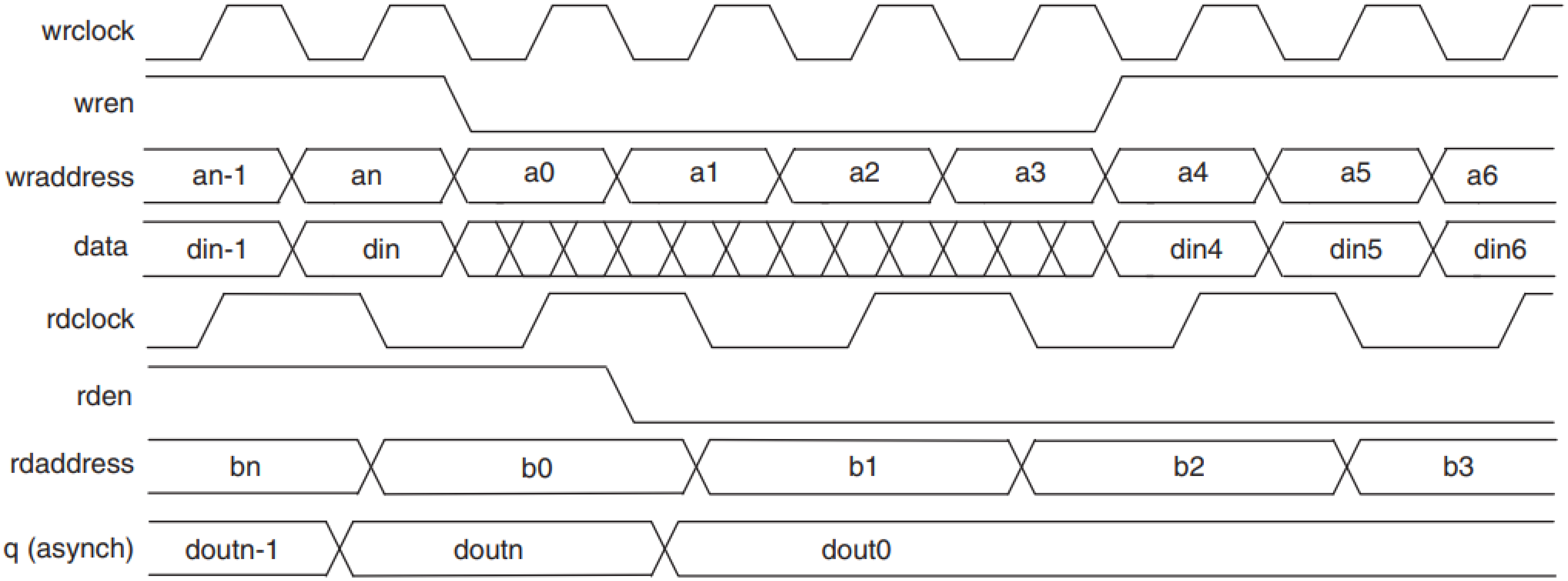


Example 13–11. Verilog HDL Single-Clock Simple Dual-Port Synchronous RAM with Old Data Read-During-Write Behavior

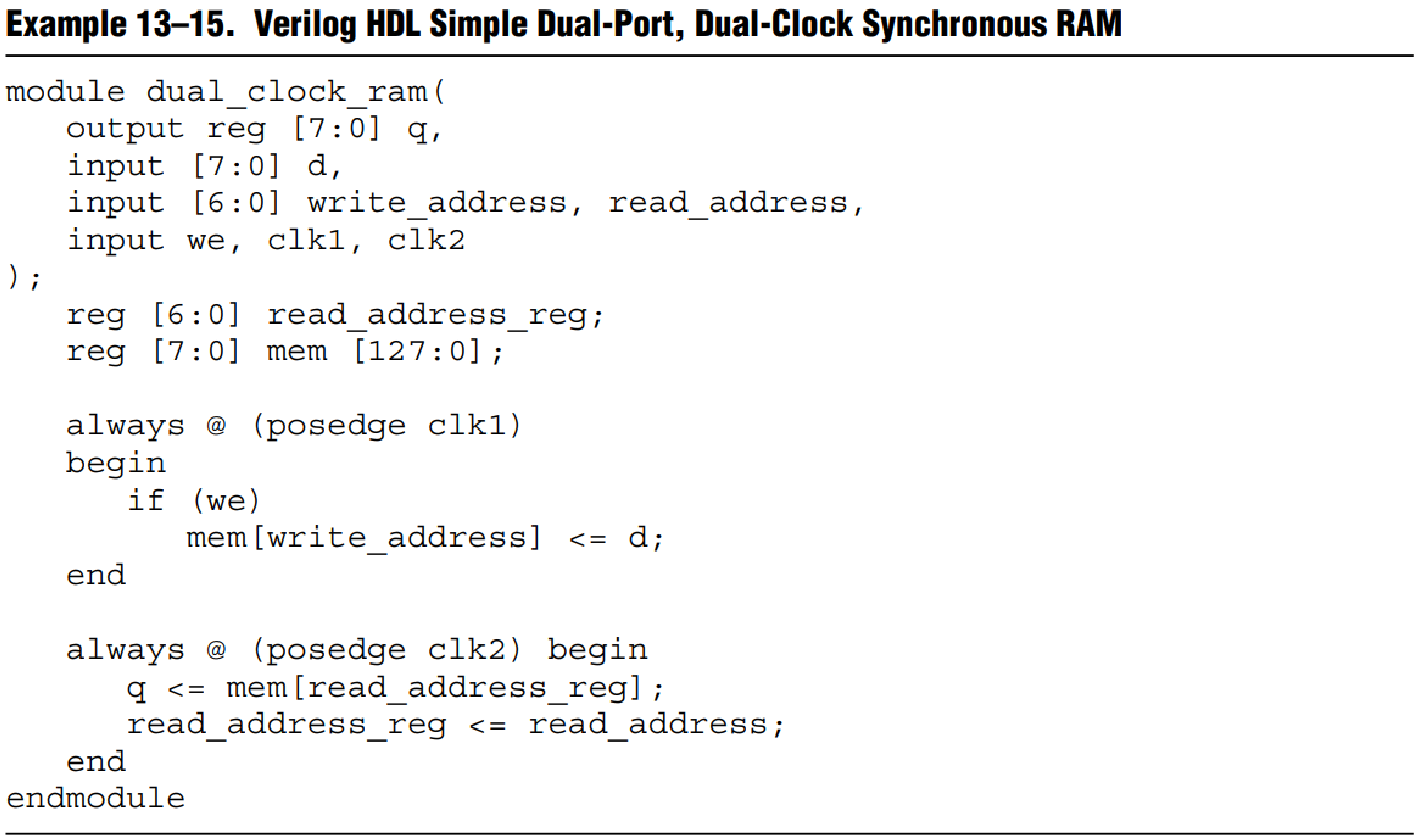
```
module single_clk_ram(
    output reg [7:0] q,
    input [7:0] d,
    input [6:0] write_address, read_address,
    input we, clk
);
    reg [7:0] mem [127:0];

    always @ (posedge clk) begin
        if (we)
            mem[write_address] <= d;
        q <= mem[read_address]; // q doesn't get d in this clock cycle
    end
endmodule
```

M9K Simple Dual-Port





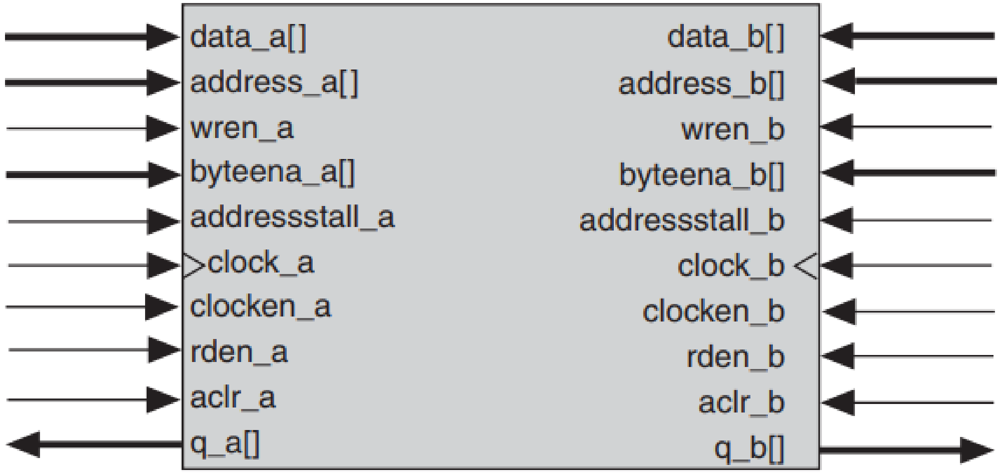
M9K Simple Dual-Port

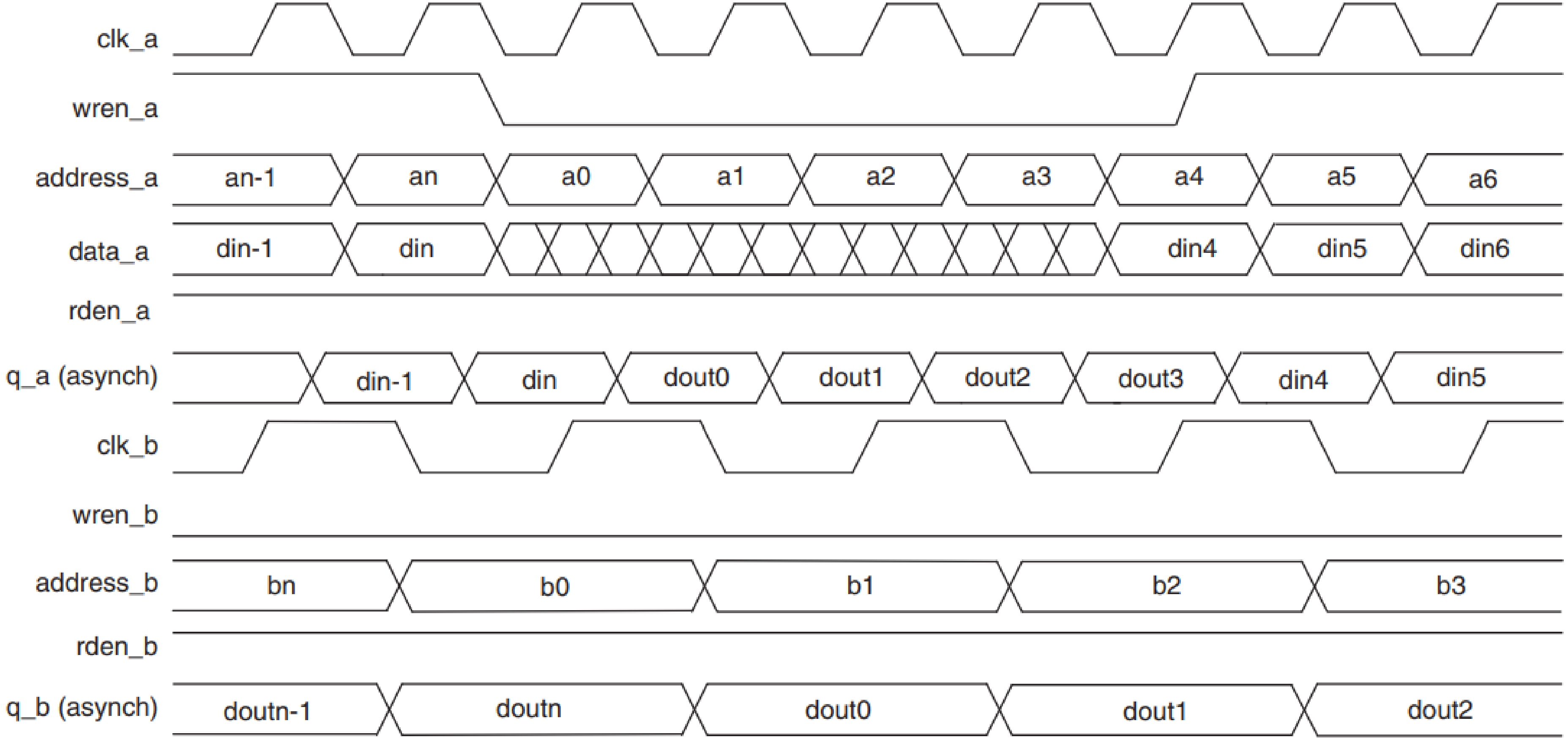


Example 13–15. Verilog HDL Simple Dual-Port, Dual-Clock Synchronous RAM

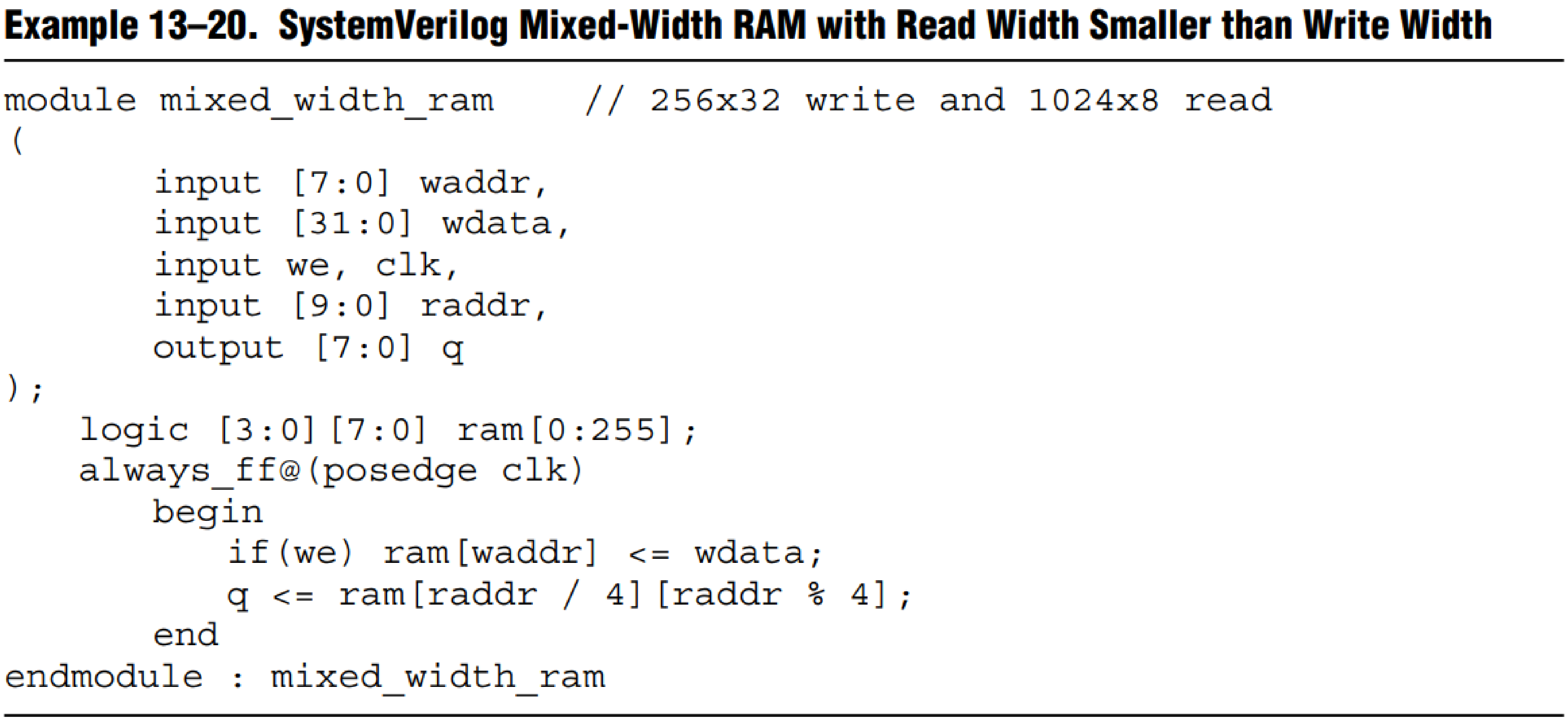
```
module dual_clock_ram(
   output reg [7:0] q,
   input [7:0] d,
   input [6:0] write_address, read_address,
   input we, clk1, clk2
);
   reg [6:0] read_address_reg;
   reg [7:0] mem [127:0];

   always @ (posedge clk1)
   begin
      if (we)
         mem[write_address] <= d;
   end

   always @ (posedge clk2) begin
      q <= mem[read_address_reg];
      read_address_reg <= read_address;
   end
endmodule
```

M9K True Dual-Port





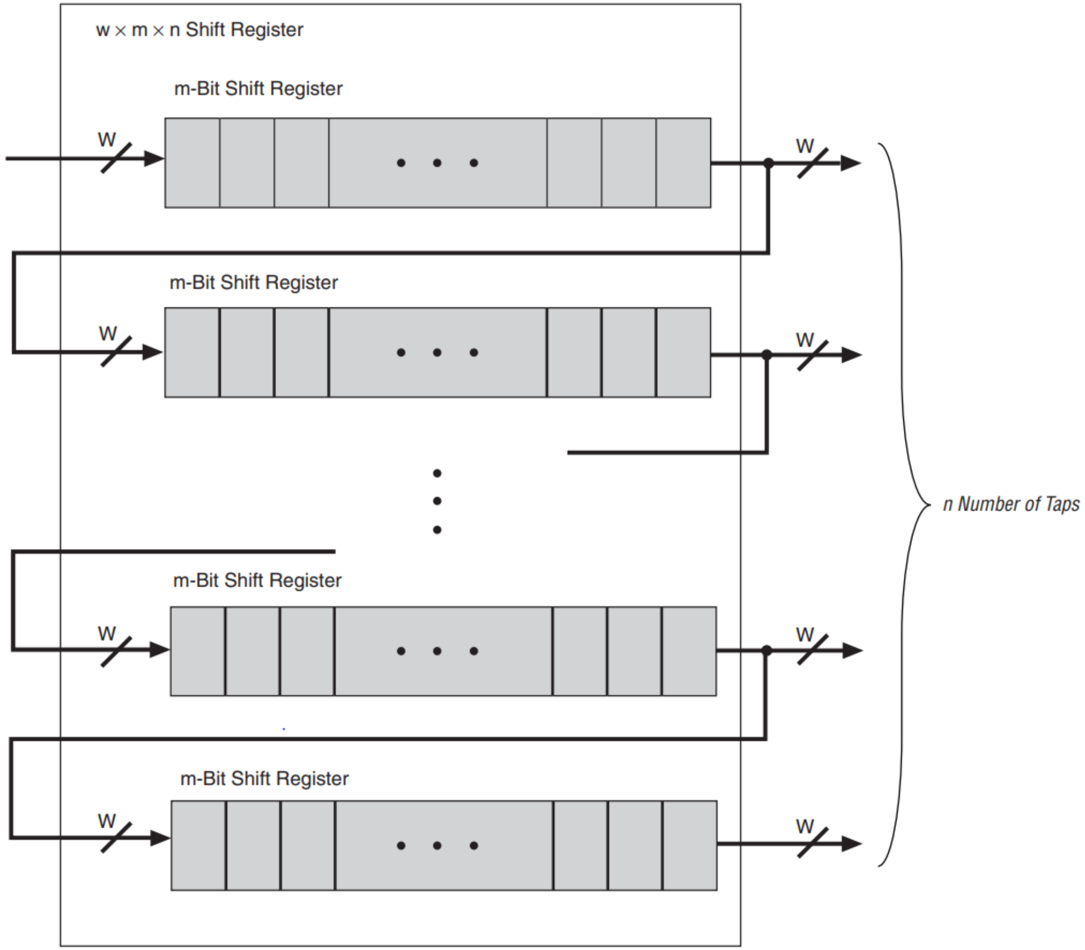
M9K True Dual-Port



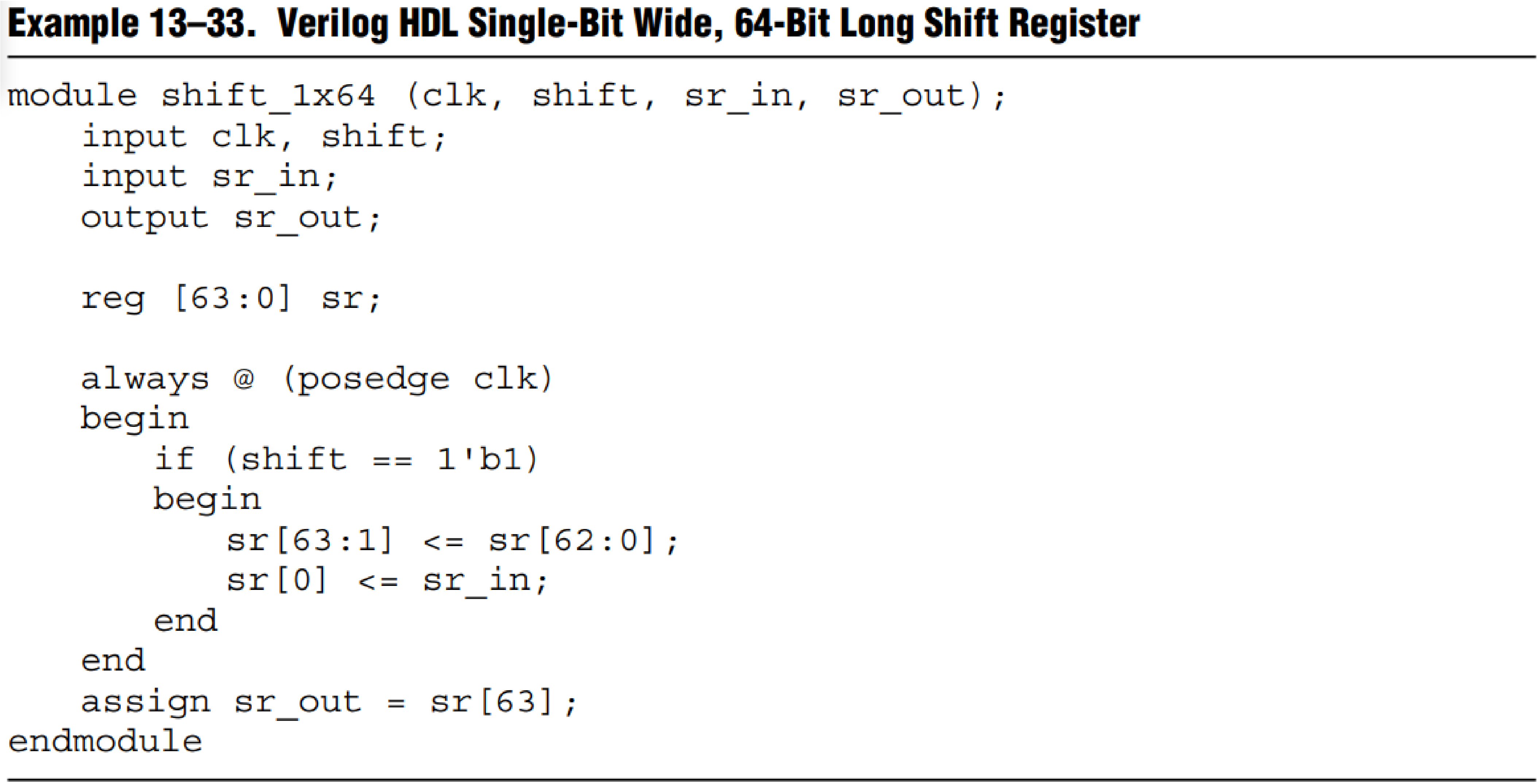
Example 13–20. SystemVerilog Mixed-Width RAM with Read Width Smaller than Write Width

```
module mixed_width_ram    // 256x32 write and 1024x8 read
(
       input [7:0] waddr,
       input [31:0] wdata,
       input we, clk,
       input [9:0] raddr,
       output [7:0] q
);
   logic [3:0][7:0] ram[0:255];
   always_ff@(posedge clk)
       begin
          if(we) ram[waddr] <= wdata;
          q <= ram[raddr / 4][raddr % 4];
       end
endmodule : mixed_width_ram
```

M9K Shift Register



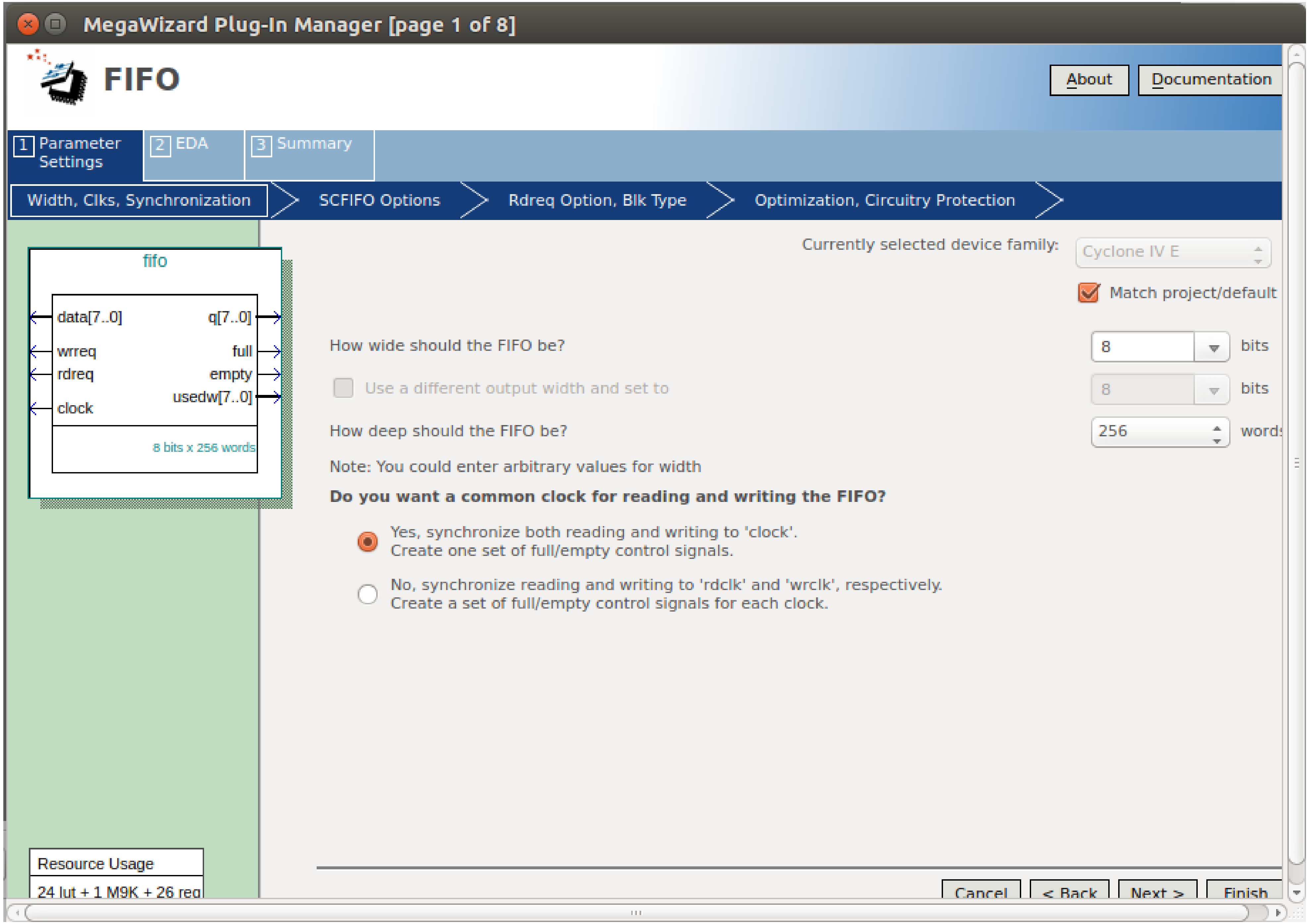
M9K Shift Register



Example 13–33. Verilog HDL Single-Bit Wide, 64-Bit Long Shift Register

```
module shift_1x64 (clk, shift, sr_in, sr_out);
    input clk, shift;
    input sr_in;
    output sr_out;

    reg [63:0] sr;

    always @ (posedge clk)
    begin
        if (shift == 1'b1)
        begin
            sr[63:1] <= sr[62:0];
            sr[0] <= sr_in;
        end
    end
    assign sr_out = sr[63];
endmodule
```

M9K FIFO



MegaWizard Plug-In Manager [page 1 of 8]

FIFO

Currently selected device family: Cyclone IV E

☑ Match project/default

How wide should the FIFO be? 8 bits

□ Use a different output width and set to 8 bits

How deep should the FIFO be? 256 words

Note: You could enter arbitrary values for width

Do you want a common clock for reading and writing the FIFO?

● Yes, synchronize both reading and writing to 'clock'.
Create one set of full/empty control signals.

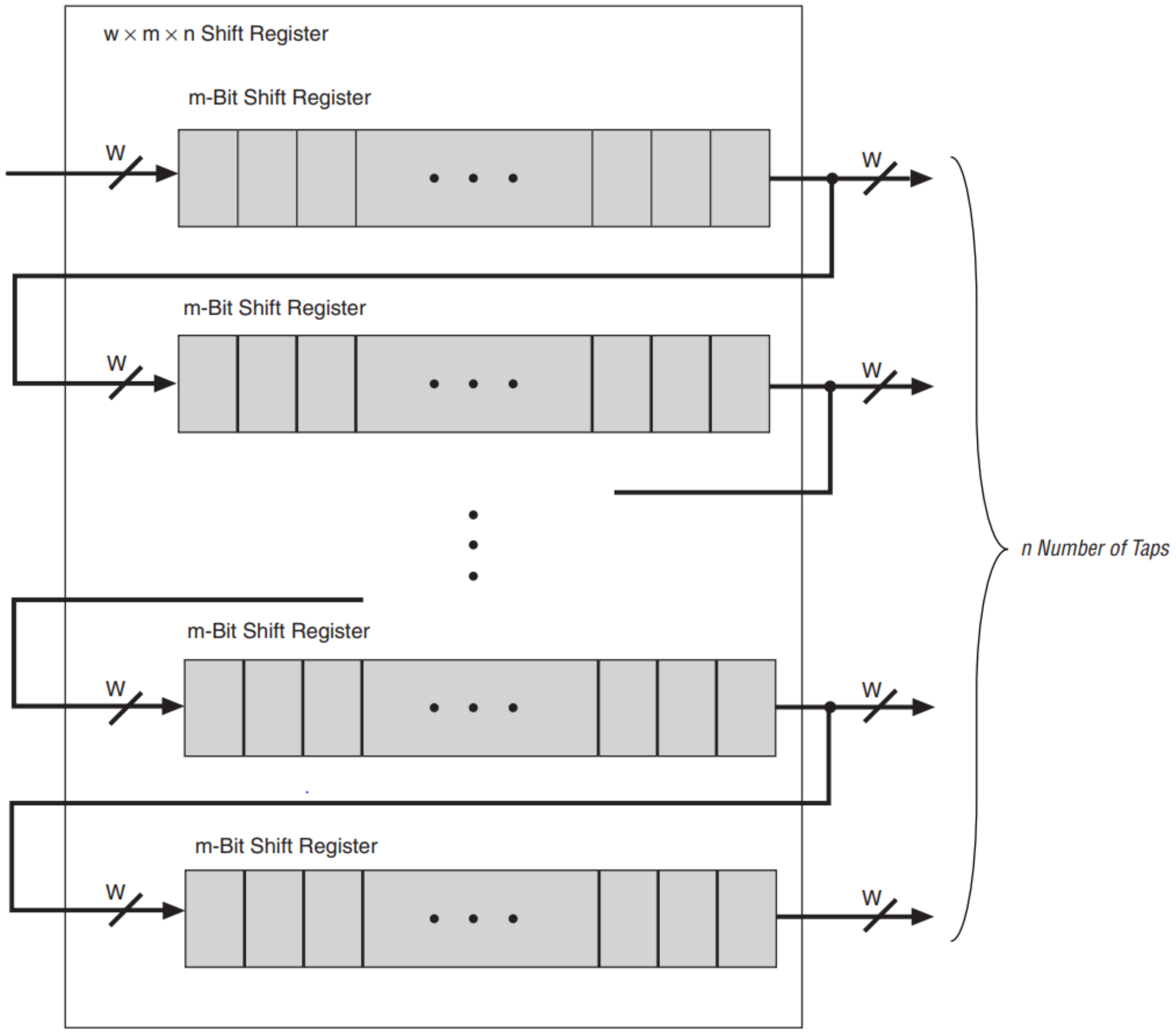
○ No, synchronize reading and writing to 'rdclk' and 'wrclk', respectively.
Create a set of full/empty control signals for each clock.

Resource Usage

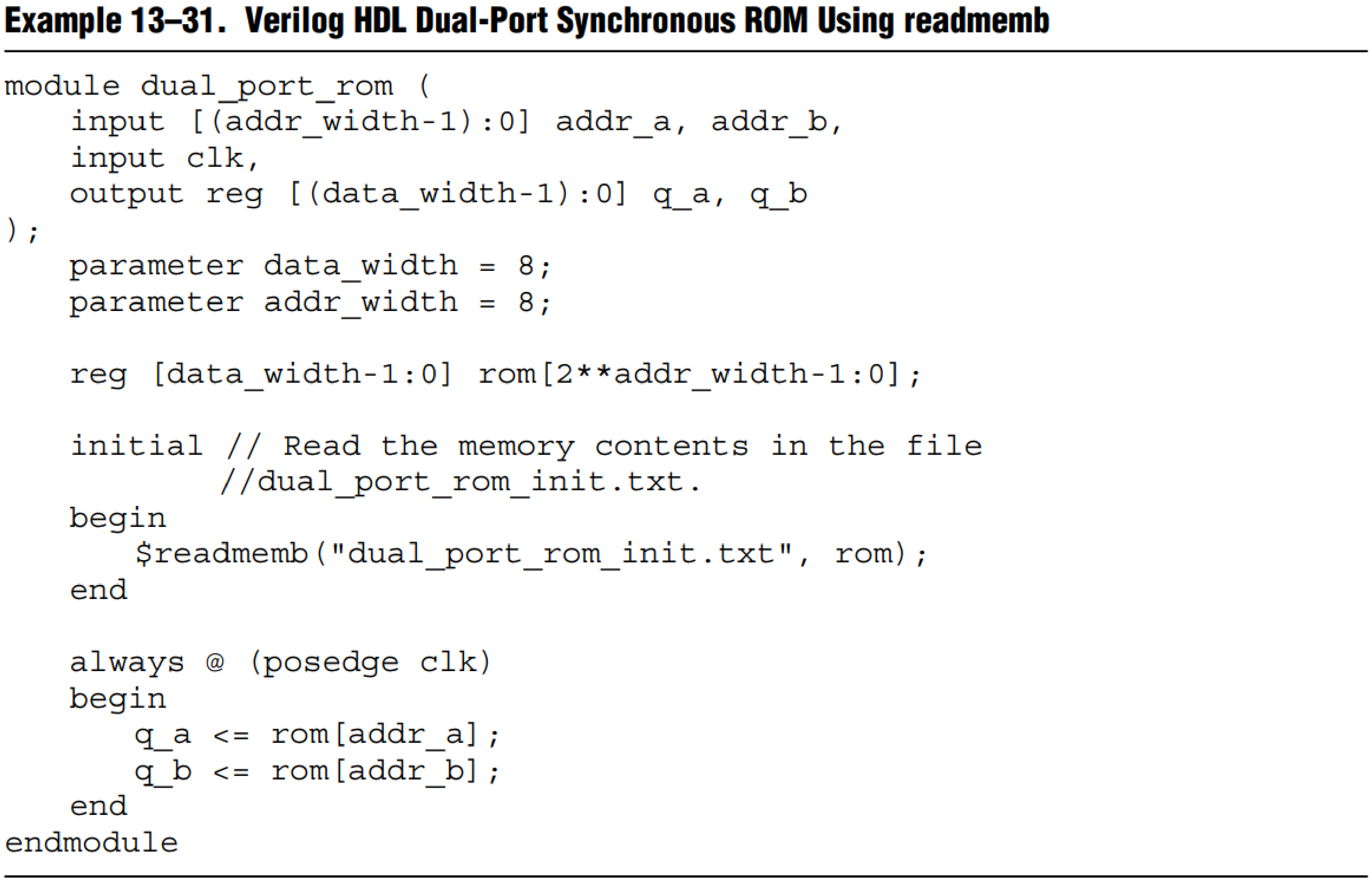
24 lut + 1 M9K + 26 reg

M9K ROM

- ALTSHIFT_TAPS MegaFunction



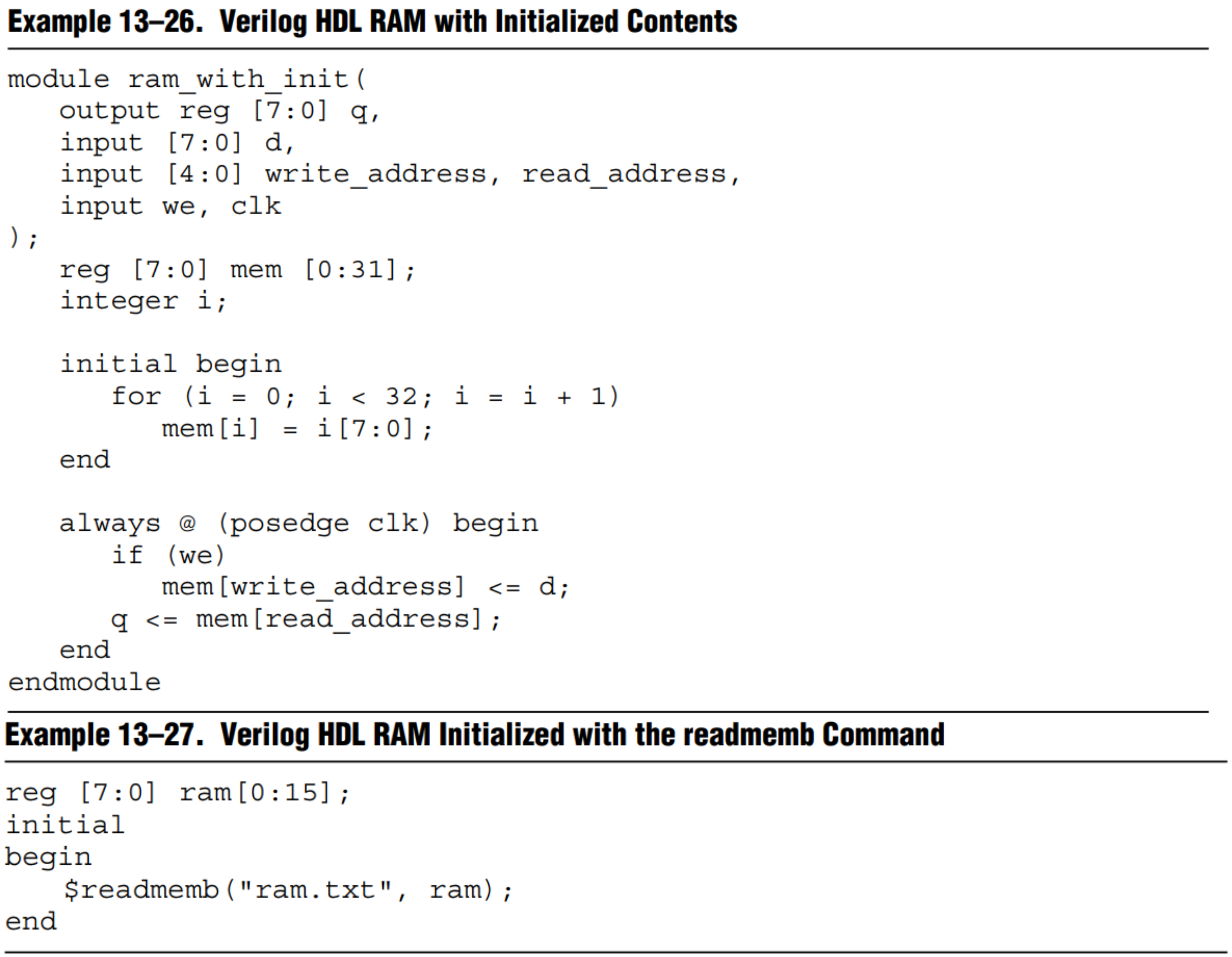
M9K ROM



Example 13–31. Verilog HDL Dual-Port Synchronous ROM Using readmemb

```
module dual_port_rom (
   input [(addr_width-1):0] addr_a, addr_b,
   input clk,
   output reg [(data_width-1):0] q_a, q_b
);
   parameter data_width = 8;
   parameter addr_width = 8;

   reg [data_width-1:0] rom[2**addr_width-1:0];

   initial // Read the memory contents in the file
           //dual_port_rom_init.txt.
   begin
      $readmemb("dual_port_rom_init.txt", rom);
   end

   always @ (posedge clk)
   begin
      q_a <= rom[addr_a];
      q_b <= rom[addr_b];
   end
endmodule
```

Memory Initialization



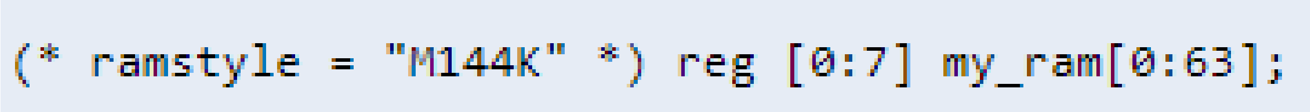
Example 13–26. Verilog HDL RAM with Initialized Contents

```
module ram_with_init(
   output reg [7:0] q,
   input [7:0] d,
   input [4:0] write_address, read_address,
   input we, clk
);
   reg [7:0] mem [0:31];
   integer i;

   initial begin
      for (i = 0; i < 32; i = i + 1)
         mem[i] = i[7:0];
   end

   always @ (posedge clk) begin
      if (we)
         mem[write_address] <= d;
      q <= mem[read_address];
   end
endmodule
```

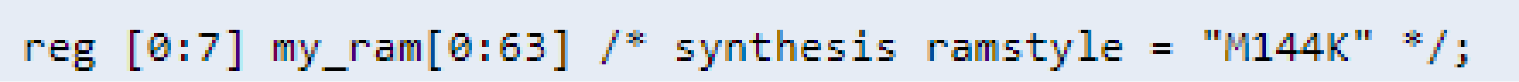
Example 13–27. Verilog HDL RAM Initialized with the readmemb Command

```
reg [7:0] ram[0:15];
initial
begin
   $readmemb("ram.txt", ram);
end
```

Атрибуты и параметры



```
(* ramstyle = "M144K" *) reg [0:7] my_ram[0:63];
```

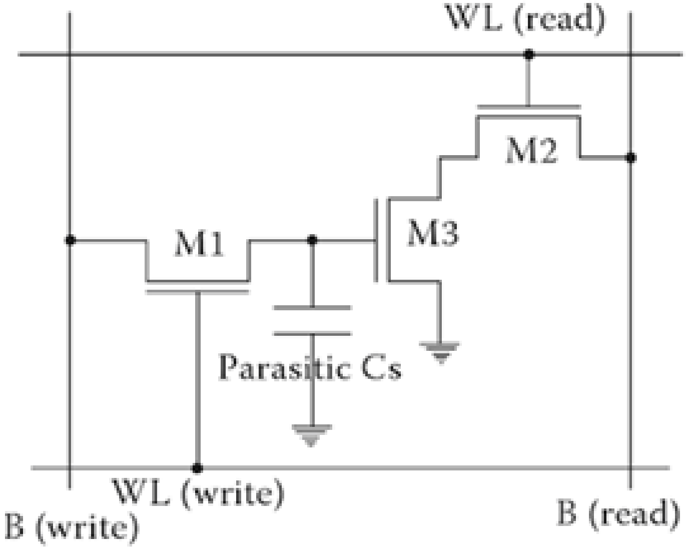


```
reg [0:7] my_ram[0:63] /* synthesis ramstyle = "M144K" */;
```

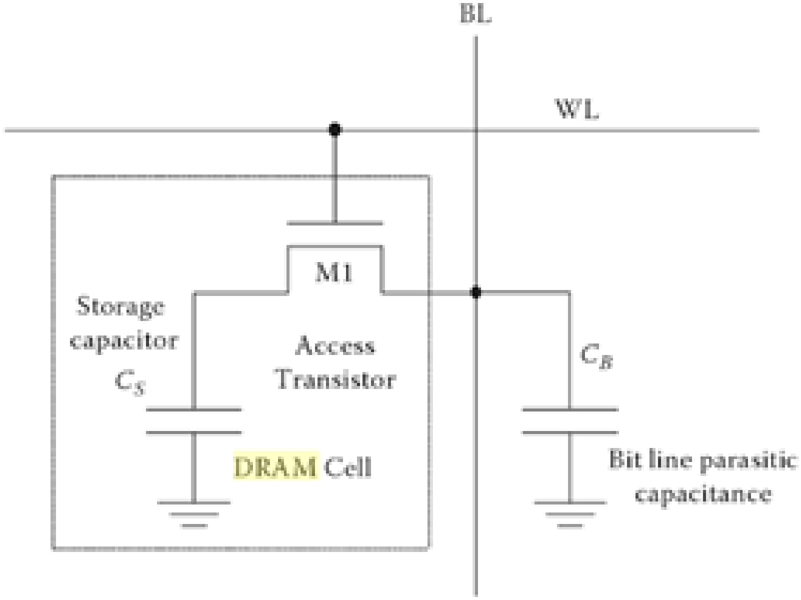
- no_rw_check
- M9K
- logic

Внешняя SDRAM

- 8Mb
- Энергозависимая
- Интерфейс доступа специфицирован
-



WL (read)
M2
M1
M3
Parasitic Cs
WL (write)
B (write)
B (read)



BL
WL
M1
Storage capacitor C_S
Access Transistor
DRAM Cell
C_B
Bit line parasitic capacitance