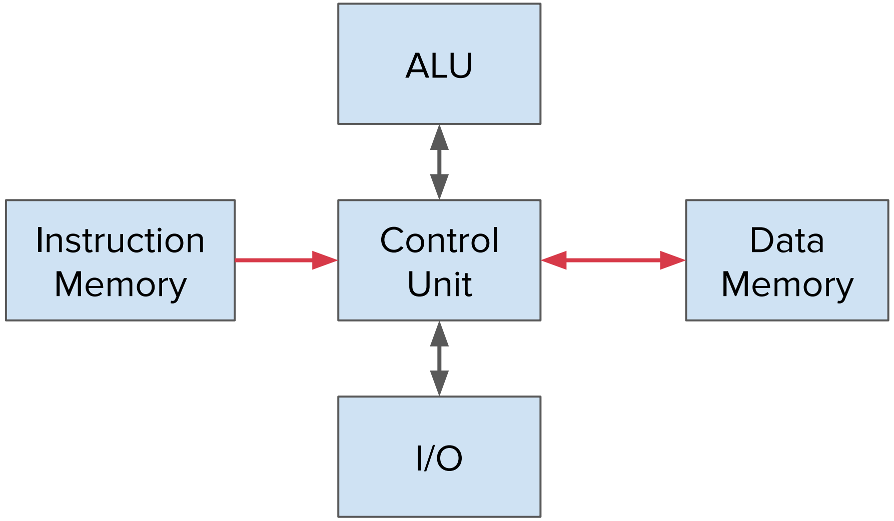
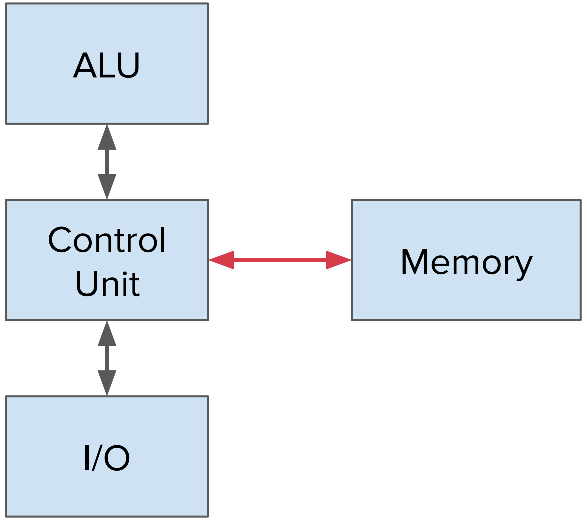
RAM. Память данных. Стек

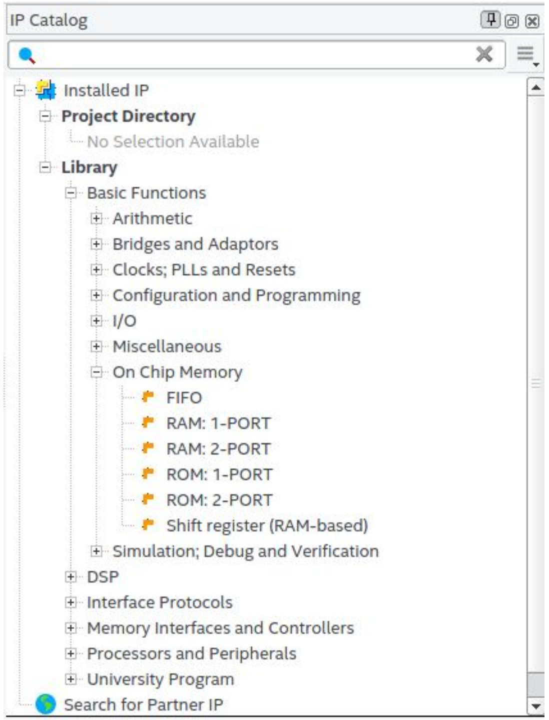
Память данных

- Архитектура фон Неймана — данные и команды хранятся совместно
- Гарвардская архитектура — данные и команды хранятся раздельно, каналы данных и инструкций тоже разделены



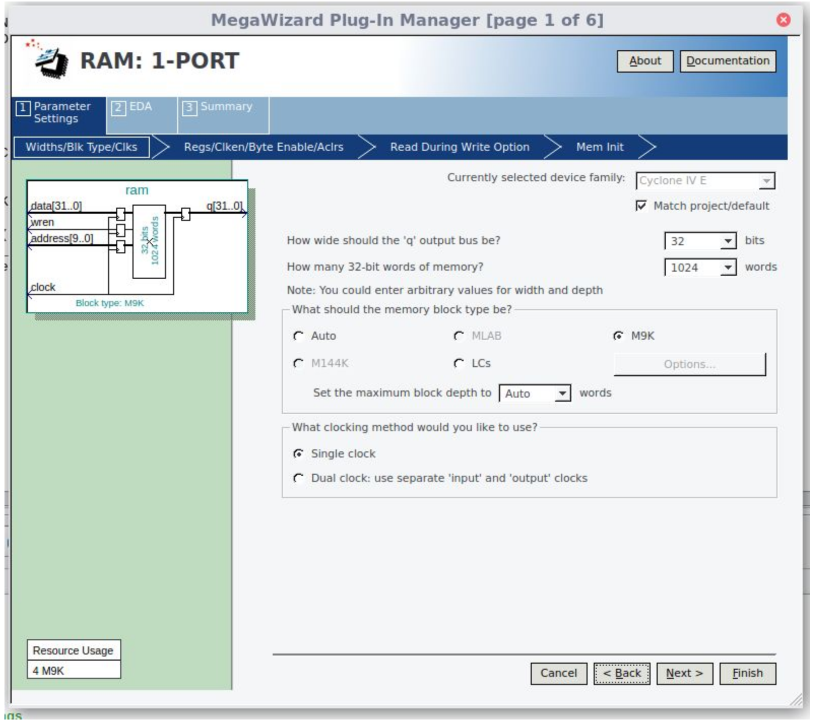


RAM: 1-PORT



IP Catalog

- Installed IP
  - Project Directory
    - No Selection Available
  - Library
    - Basic Functions
      - Arithmetic
      - Bridges and Adaptors
      - Clocks; PLLs and Resets
      - Configuration and Programming
      - I/O
      - Miscellaneous
      - On Chip Memory
        - FIFO
        - RAM: 1-PORT
        - RAM: 2-PORT
        - ROM: 1-PORT
        - ROM: 2-PORT
        - Shift register (RAM-based)
      - Simulation; Debug and Verification
    - DSP
    - Interface Protocols
    - Memory Interfaces and Controllers
    - Processors and Peripherals
    - University Program
- Search for Partner IP



MegaWizard Plug-In Manager [page 1 of 6]

RAM: 1-PORT

About | Documentation

1 Parameter Settings | 2 EDA | 3 Summary

Widths/Blk Type/Clks > Regs/Clken/Byte Enable/Aclrs > Read During Write Option > Mem Init >

Currently selected device family: Cyclone IV E

[✓] Match project/default

How wide should the 'q' output bus be? 32 bits

How many 32-bit words of memory? 1024 words

Note: You could enter arbitrary values for width and depth

What should the memory block type be?

Auto | MLAB | [✓ M9K]

M144K | LCs | Options...

Set the maximum block depth to Auto words

What clocking method would you like to use?

[✓ Single clock]

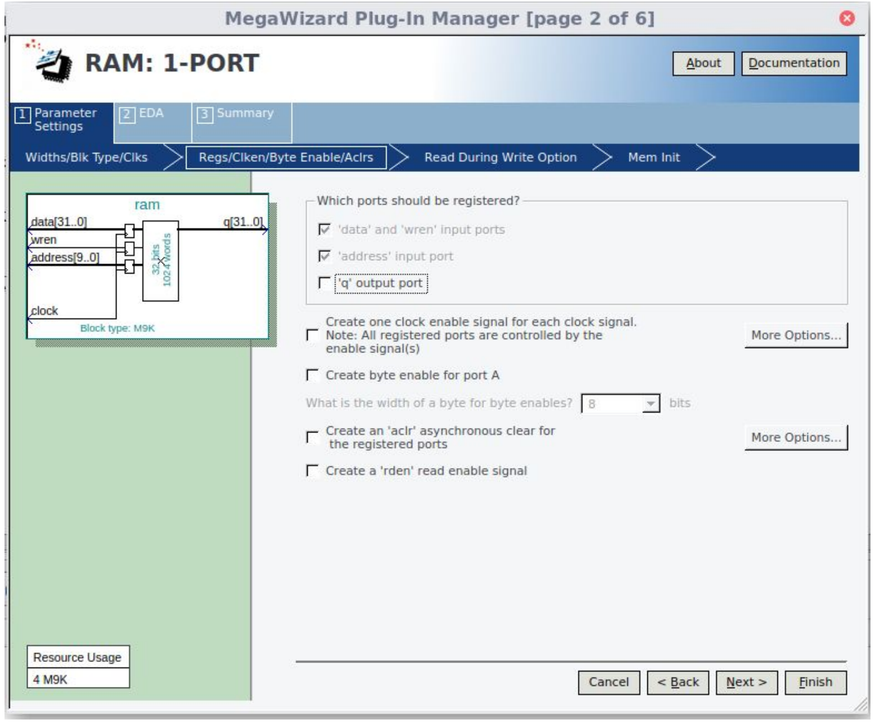
Dual clock: use separate 'input' and 'output' clocks

Resource Usage

4 M9K

Cancel | < Back | Next > | Finish

RAM: 1-PORT



MegaWizard Plug-In Manager [page 2 of 6]

RAM: 1-PORT

About | Documentation

1 Parameter Settings | 2 EDA | 3 Summary

Widths/Blk Type/Clks > Regs/Clken/Byte Enable/Aclrs > Read During Write Option > Mem Init >

Which ports should be registered?

- [✓] 'data' and 'wren' input ports
- [✓] 'address' input port
- [ ] 'q' output port

- [ ] Create one clock enable signal for each clock signal. Note: All registered ports are controlled by the enable signal(s) — More Options...
- [ ] Create byte enable for port A

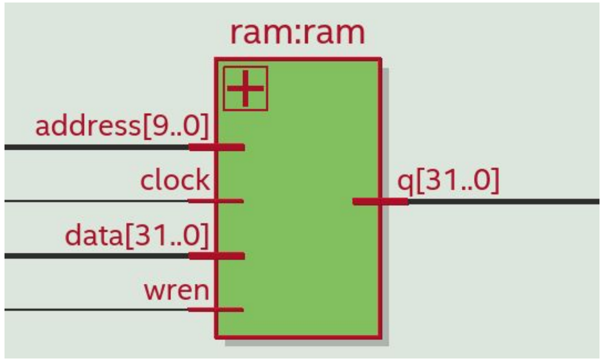
What is the width of a byte for byte enables? 8 bits

- [ ] Create an 'aclr' asynchronous clear for the registered ports — More Options...
- [ ] Create a 'rden' read enable signal

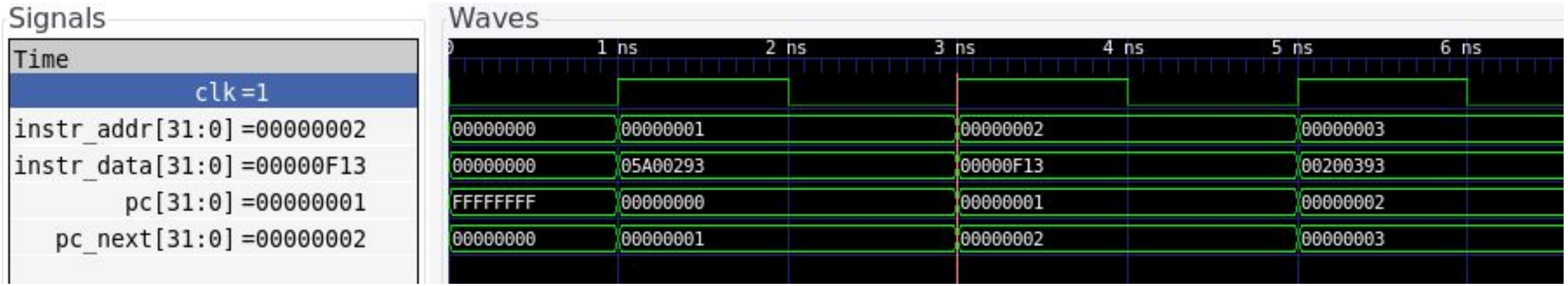
Resource Usage
4 M9K

Cancel | < Back | Next > | Finish

```
module ram(
        input       [9:0]address,
        input       clock,
        input       [31:0]data,
        input       wren,
        output      [31:0]q
);
```



Задержка

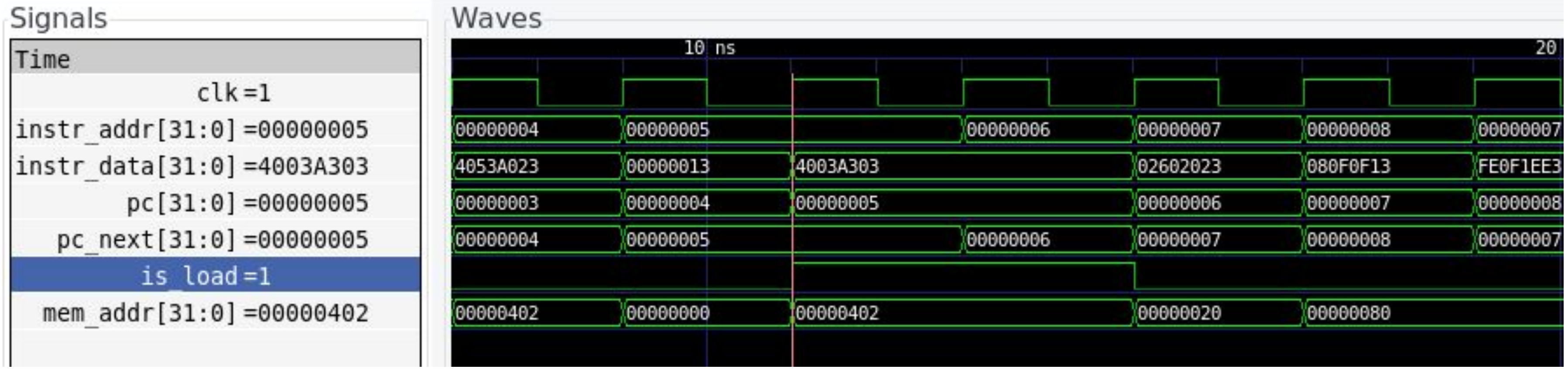


Signals

```
Time
clk=1
instr_addr[31:0]=00000002
instr_data[31:0]=00000F13
pc[31:0]=00000001
pc_next[31:0]=00000002
```

Waves

```
0         1 ns      2 ns      3 ns      4 ns      5 ns      6 ns
00000000  00000001            00000002            00000003
00000000  05A00293            00000F13            00200393
FFFFFFFF  00000000            00000001            00000002
00000000  00000001            00000002            00000003
```



Signals

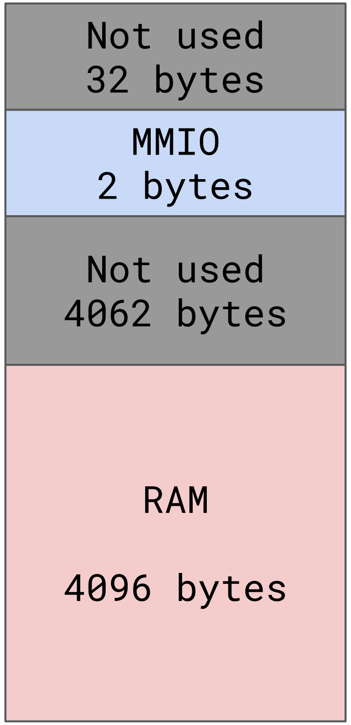
```
Time
clk=1
instr_addr[31:0]=00000005
instr_data[31:0]=4003A303
pc[31:0]=00000005
pc_next[31:0]=00000005
is_load=1
mem_addr[31:0]=00000402
```

Waves

```
                    10 ns                                                20
00000004  00000005            00000006  00000007  00000008  00000007
4053A023  00000013  4003A303            02602023  080F0F13  FE0F1EE3
00000003  00000004  00000005            00000006  00000007  00000008
00000004  00000005            00000006  00000007  00000008  00000007
00000402  00000000  00000402            00000020  00000080
```

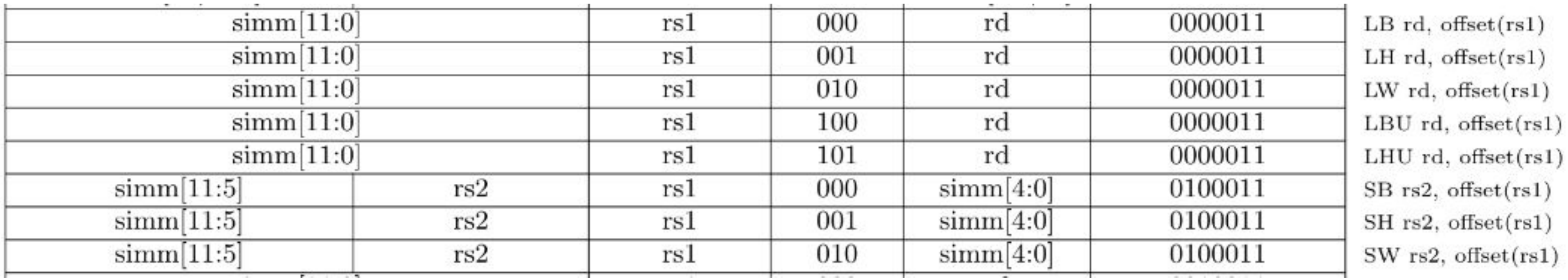
mem_ctrl.v

```
always @(*) begin
    q = 32'b0;
    mmio_data = 16'b0;
    mmio_we = 1'b0;
    ram_data = 32'b0;
    ram_addr = 10'b0;
    ram_we = 1'b0;

    casez (addr)
        32'h20: begin // MMIO
            mmio_data = data[15:0];
            mmio_we = we;
        end
        32'b1_????_????_????: begin // RAM
            ram_addr = addr[11:2];
            ram_data = data;
            ram_we = we;
            q = ram_q;
        end
    endcase
end
```



control.v

```
7'b0000011: begin // LW
     imm12 = instr[31:20];
     rf_we = (funct3 == 3'b010);
     has_imm = 1'b1;
     is_load = 1'b1;
 end
 7'b0100011: begin // SW
     imm12 = {instr[31:25], instr[11:7]};
     has_imm = 1'b1;
     mem_we = (funct3 == 3'b010);
 end
```

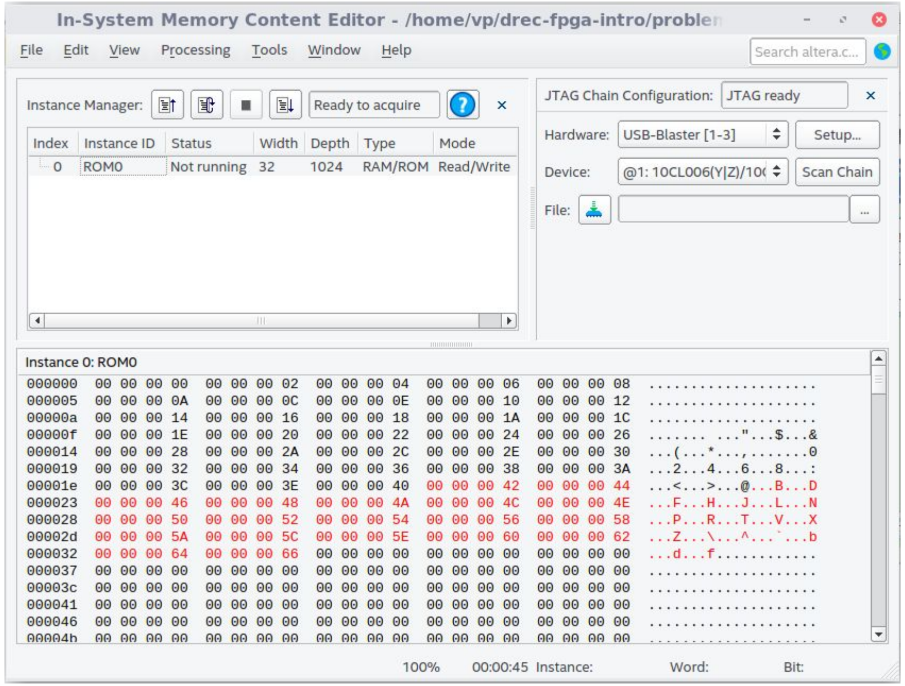


simm[11:0]		rs1	000	rd	0000011	LB rd, offset(rs1)
simm[11:0]		rs1	001	rd	0000011	LH rd, offset(rs1)
simm[11:0]		rs1	010	rd	0000011	LW rd, offset(rs1)
simm[11:0]		rs1	100	rd	0000011	LBU rd, offset(rs1)
simm[11:0]		rs1	101	rd	0000011	LHU rd, offset(rs1)
simm[11:5]	rs2	rs1	000	simm[4:0]	0100011	SB rs2, offset(rs1)
simm[11:5]	rs2	rs1	001	simm[4:0]	0100011	SH rs2, offset(rs1)
simm[11:5]	rs2	rs1	010	simm[4:0]	0100011	SW rs2, offset(rs1)

core.v

```
wire [31:0]pc_next = ((pc == LAST_PC) || (is_load && ~load_delay)) ? pc : pc_target;

reg load_delay = 1'b0;

always @(posedge clk) begin
    pc <= pc_next;
    load_delay <= is_load && ~load_delay;
end

....................

wire [31:0]rf_wdata = is_load ? mem_q : alu_result;

....................

assign mem_addr = alu_result;
assign mem_data = rf_rdata1;
```

In-System Memory Content Editor


In-System Memory Content Editor - /home/vp/drec-fpga-intro/problem

File Edit View Processing Tools Window Help

Instance Manager: Ready to acquire

Index	Instance ID	Status	Width	Depth	Type	Mode
0	ROM0	Not running	32	1024	RAM/ROM	Read/Write

JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [1-3]

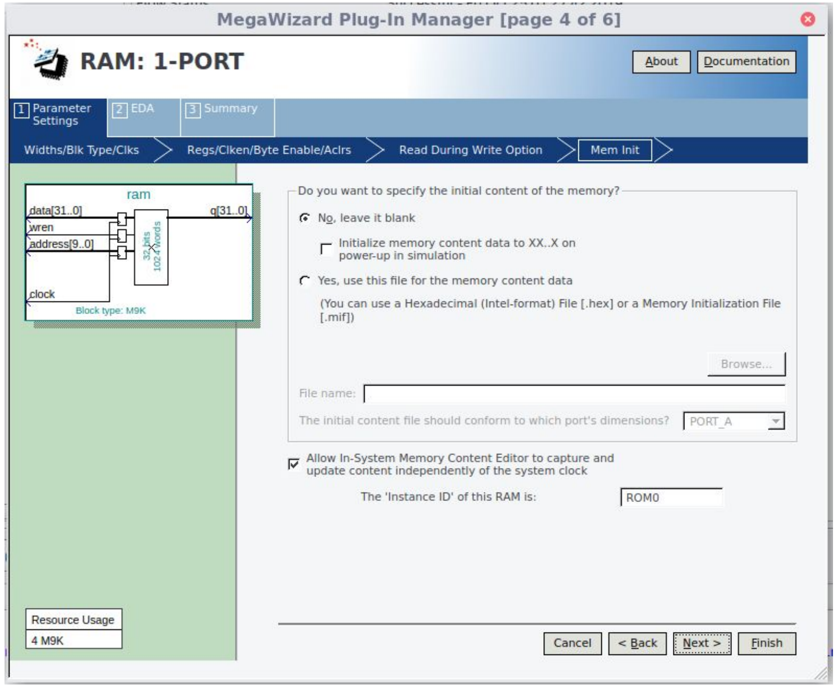
Device: @1: 10CL006(Y|Z)/10(

File:

Instance 0: ROM0

```
000000  00 00 00 00  00 00 00 02  00 00 00 04  00 00 00 06  00 00 00 08  ....................
000005  00 00 00 0A  00 00 00 0C  00 00 00 0E  00 00 00 10  00 00 00 12  ....................
00000a  00 00 00 14  00 00 00 16  00 00 00 18  00 00 00 1A  00 00 00 1C  ....................
00000f  00 00 00 1E  00 00 00 20  00 00 00 22  00 00 00 24  00 00 00 26  ....... ..."...$...&
000014  00 00 00 28  00 00 00 2A  00 00 00 2C  00 00 00 2E  00 00 00 30  ...(...*...,.......0
000019  00 00 00 32  00 00 00 34  00 00 00 36  00 00 00 38  00 00 00 3A  ...2...4...6...8...:
00001e  00 00 00 3C  00 00 00 3E  00 00 00 40  00 00 00 42  00 00 00 44  ...<...>...@...B...D
000023  00 00 00 46  00 00 00 48  00 00 00 4A  00 00 00 4C  00 00 00 4E  ...F...H...J...L...N
000028  00 00 00 50  00 00 00 52  00 00 00 54  00 00 00 56  00 00 00 58  ...P...R...T...V...X
00002d  00 00 00 5A  00 00 00 5C  00 00 00 5E  00 00 00 60  00 00 00 62  ...Z...\...^...`...b
000032  00 00 00 64  00 00 00 66  00 00 00 00  00 00 00 00  00 00 00 00  ...d...f............
000037  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ....................
00003c  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ....................
000041  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ....................
000046  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ....................
00004b  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ....................
```

100% 00:00:45 Instance: Word: Bit:


MegaWizard Plug-In Manager [page 4 of 6]

RAM: 1-PORT

Do you want to specify the initial content of the memory?

[✓ No, leave it blank]

☐ Initialize memory content data to XX..X on power-up in simulation

Yes, use this file for the memory content data

(You can use a Hexadecimal (Intel-format) File [.hex] or a Memory Initialization File [.mif])

Browse...

File name:

The initial content file should conform to which port's dimensions? PORT_A

[✓] Allow In-System Memory Content Editor to capture and update content independently of the system clock

The 'Instance ID' of this RAM is: ROM0

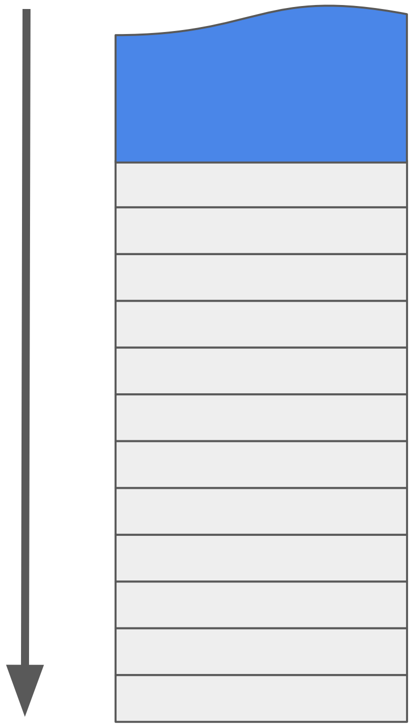
Пример функции

```c
typedef unsigned int uint32_t;
typedef unsigned char uint8_t;

uint32_t mean4(uint32_t x0, uint32_t x1, uint32_t x2, uint32_t x3) {
    uint32_t mean;

    mean = (x0 + x1 + x2 + x3) >> 2;

    return mean;
}
```

```
$ riscv64-linux-gnu-gcc -c mean4.c -march=rv32i -O0 -mabi=ilp32 -nostdlib -o mean4.o
$ riscv64-linux-gnu-objdump -d mean4.o
```

Стековый фрейм

```
00000000 <mean4>:
   0: fd010113          addi   sp,sp,-48
   4: 02812623          sw     s0,44(sp)
   8: 03010413          addi   s0,sp,48
   c: fca42e23          sw     a0,-36(s0)
  10: fcb42c23          sw     a1,-40(s0)
  14: fcc42a23          sw     a2,-44(s0)
  18: fcd42823          sw     a3,-48(s0)
  1c: fdc42703          lw     a4,-36(s0)
  20: fd842783          lw     a5,-40(s0)
  24: 00f70733          add    a4,a4,a5
  28: fd442783          lw     a5,-44(s0)
  2c: 00f70733          add    a4,a4,a5
  30: fd042783          lw     a5,-48(s0)
  34: 00f707b3          add    a5,a4,a5
  38: 0027d793          srli   a5,a5,0x2
  3c: fef42623          sw     a5,-20(s0)
  40: fec42783          lw     a5,-20(s0)
  44: 00078513          mv     a0,a5
  48: 02c12403          lw     s0,44(sp)
  4c: 03010113          addi   sp,sp,48
  50: 00008067          ret
```



GitHub

github.com/viktor-prutyanov/drec-fpga-intro