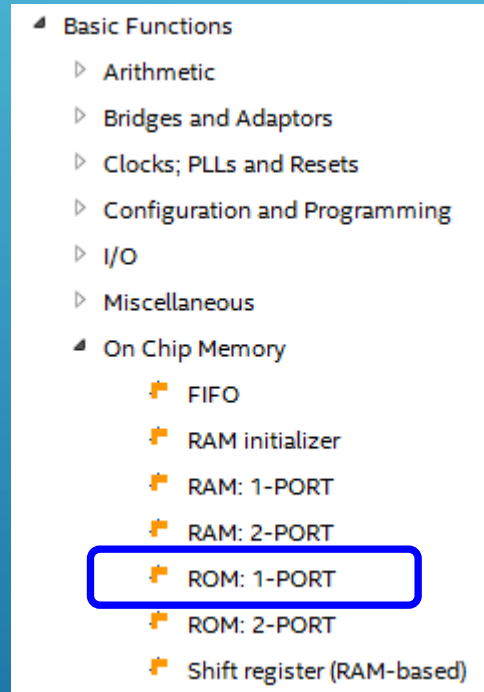


Creating an FPGA ROM

In Quartus, right panel: under Installed IP
Select **ROM: 1-PORT**



Creating an FPGA ROM

MegaWizard Plug-In Manager [page 1 of 5]

ROM: 1-PORT

About Documentation

1 Parameter Settings 2 EDA 3 Summary

General > Regs/Clock/Adrs > Mem Init >

Currently selected device family: Cyclone V

☒ Match project/default

How wide should the 'q' output bus be? 12 bits

How many 12-bit words of memory? 128 words

Note: You could enter arbitrary values for width and depth

What should the memory block type be?

☒ Auto ☐ MLAB ☐ M10K

☐ M-RAM ☐ LCs Options...

Set the maximum block depth to Auto words

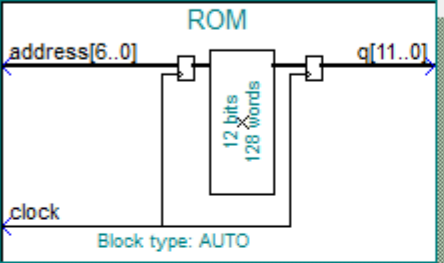
What clocking method would you like to use?

☒ Single clock ☐ Dual clock: use separate 'input' and 'output' clocks

Resource Usage

1 M10K

Cancel < Back Next > Finish



Select how large in bits the words in memory should be.

Here we have selected the word size to be 12 bits.

Creating an FPGA ROM

MegaWizard Plug-In Manager [page 1 of 5]

ROM: 1-PORT

About Documentation

1 Parameter Settings 2 EDA 3 Summary

General > Regs/Clock/Adrs > Mem Init >

Currently selected device family: Cyclone V

☒ Match project/default

How wide should the 'q' output bus be? 12 bits

How many 12-bit words of memory? 128 words

Note: You could enter arbitrary values for width and depth

What should the memory block type be?

☒ Auto ☐ MLAB ☐ M10K

☐ M-RAM ☐ LCs Options...

Set the maximum block depth to Auto words

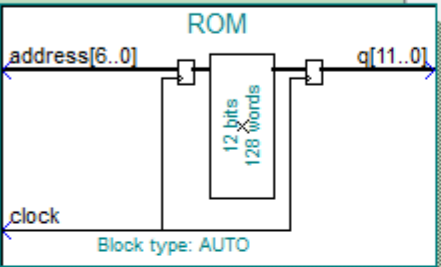
What clocking method would you like to use?

☒ Single clock ☐ Dual clock: use separate 'input' and 'output' clocks

Resource Usage

1 M10K

Cancel < Back Next > Finish



Select how many words in memory.

Here we select 128 12-bit words, which gives us a 7-bit address space.

Creating an FPGA ROM

Determine if the output q should be registered.

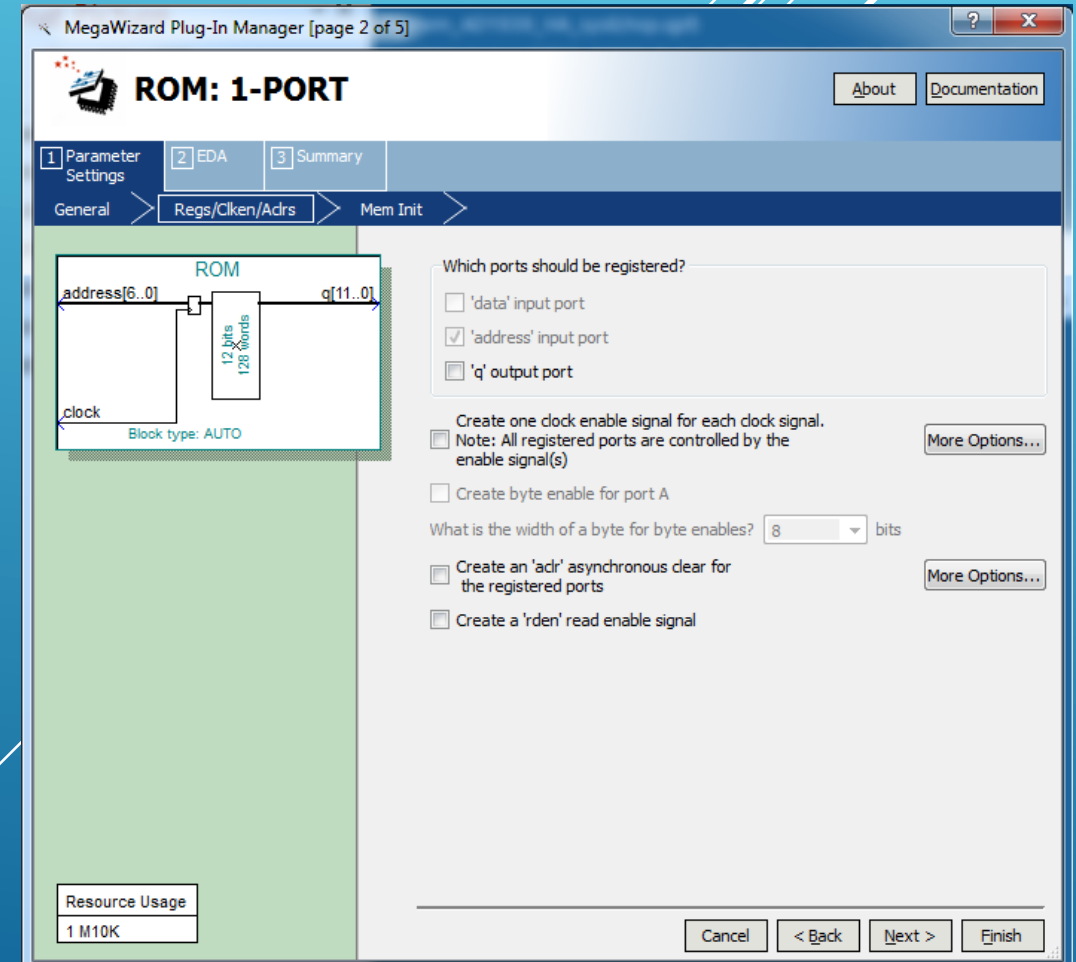
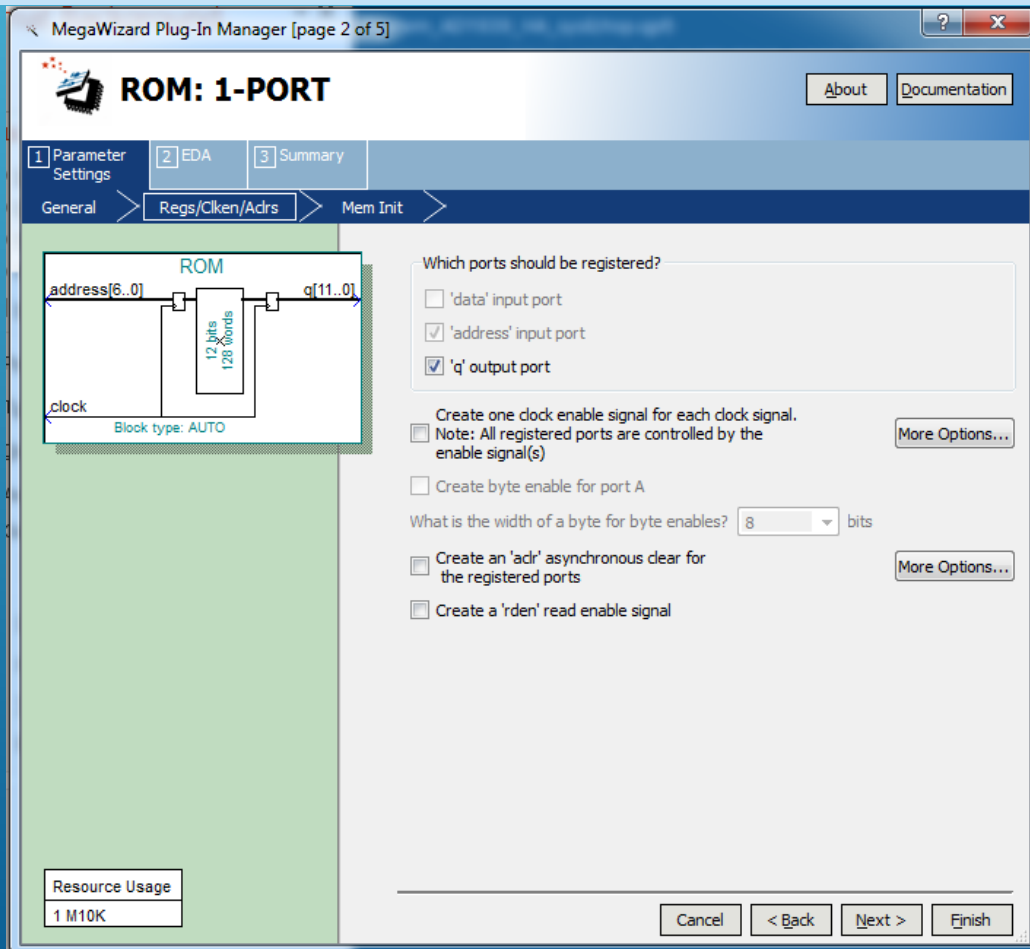
The output is registered.

The design will be able to run with a faster clock.

The result will take an extra clock cycle to show up

The output is **NOT** registered.

Easier to design into system



Creating an FPGA ROM

MegaWizard Plug-In Manager [page 3 of 5]

ROM: 1-PORT

About Documentation

1 Parameter Settings 2 EDA 3 Summary

General > Regs/Clen/Adrs > Mem Init >

ROM

address[6..0] q[11..0]

12 bits
128 words

clock

Block type: AUTO

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: rsqrt_table.mif

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 ROM is: NONE

Resource Usage

1 M10K

Cancel < Back Next > Finish

Specify .mif file that initializes memory with custom values.

Note: The .mif file must match the memory size in terms of word size and number of memory words.

Creating an FPGA ROM

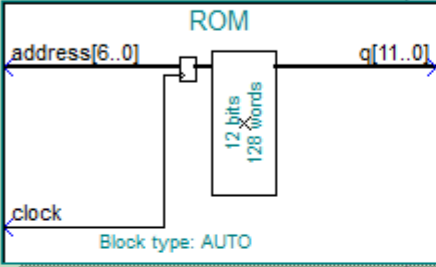
Select the component declaration file and the instantiation template file.

MegaWizard Plug-In Manager [page 5 of 5]

ROM: 1-PORT

About Documentation

1 Parameter Settings 2 EDA 3 Summary



Turn on the files you wish to generate. A gray checkmark indicates a file that is automatically generated, and a green checkmark indicates an optional file. Click Finish to generate the selected files. The state of each checkbox is maintained in subsequent MegaWizard Plug-In Manager sessions.

The MegaWizard Plug-In Manager creates the selected files in the following directory:
C:\Users\Ross\Desktop\temp\

File	Description
<input checked="" type="checkbox"/> ROM.vhd	Variation file
<input type="checkbox"/> ROM.inc	AHDL Include file
<input checked="" type="checkbox"/> ROM.cmp	VHDL component declaration file
<input type="checkbox"/> ROM.bsf	Quartus Prime symbol file
<input checked="" type="checkbox"/> ROM_inst.vhd	Instantiation template file

Resource Usage
1 M10K

Cancel < Back Next > Finish

```
component ROM
  PORT
  (
    address      : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
    clock        : IN STD_LOGIC := '1';
    q            : OUT STD_LOGIC_VECTOR (11 DOWNTO 0)
  );
end component;
```

```
ROM_inst : ROM PORT MAP (
  address => address_sig,
  clock   => clock_sig,
  q       => q_sig
);
```

```
Open Files
rsqrt_table.mif x
0 10 20 30 40 50 60 T
1 DEPTH = 256;
2 WIDTH = 16;
3 ADDRESS_RADIX = BIN;
4 DATA_RADIX = BIN;
5 CONTENT
6 BEGIN
7 00000000 : 1000000000000000; -- 1 : (1)^(-3/2)=1
8 00000001 : 0111111101000001; -- 2 : (1.0039)^(-3/2)=0.99417
9 00000010 : 0111111101000100; -- 3 : (1.0078)^(-3/2)=0.9884
10 00000011 : 0111111101100100; -- 4 : (1.0117)^(-3/2)=0.98267
11 00000100 : 0111111101000111; -- 5 : (1.0156)^(-3/2)=0.97702
12 00000101 : 0111111101010111; -- 6 : (1.0195)^(-3/2)=0.97141
13 00000110 : 0111111101100001; -- 7 : (1.0234)^(-3/2)=0.96585
14 00000111 : 0111111101101101; -- 8 : (1.0273)^(-3/2)=0.96036
15 00001000 : 0111111101101101; -- 9 : (1.0313)^(-3/2)=0.9549
16 00001001 : 0111111101100101; -- 10 : (1.0352)^(-3/2)=0.94949
17 00001010 : 0111111101101101; -- 11 : (1.0391)^(-3/2)=0.94415
18 00001011 : 0111111101100110; -- 12 : (1.043)^(-3/2)=0.93884
19 00001100 : 0111111101100000; -- 13 : (1.0469)^(-3/2)=0.93359
20 00001101 : 0111111101101110; -- 14 : (1.0508)^(-3/2)=0.92841
21 00001110 : 0111111101101101; -- 15 : (1.0547)^(-3/2)=0.92325
22 00001111 : 0111111101100101; -- 16 : (1.0586)^(-3/2)=0.91812
23 00010000 : 0111111101100000; -- 17 : (1.0625)^(-3/2)=0.91309
24 00010001 : 0111111101100111; -- 18 : (1.0664)^(-3/2)=0.90805
25 00010010 : 0111111101101101; -- 19 : (1.0703)^(-3/2)=0.90311
26 00010011 : 0111111101101111; -- 20 : (1.0742)^(-3/2)=0.89816
27 00010100 : 0111111101100110; -- 21 : (1.0781)^(-3/2)=0.89331
28 00010101 : 0111111101101101; -- 22 : (1.082)^(-3/2)=0.88846
29 00010110 : 0111111101100110; -- 23 : (1.0859)^(-3/2)=0.88367
30 00010111 : 0111111101100001; -- 24 : (1.0898)^(-3/2)=0.87894
31 00011000 : 0111111101100111; -- 25 : (1.0938)^(-3/2)=0.87424
32 00011001 : 0111111101100110; -- 26 : (1.0977)^(-3/2)=0.86957
33 00011010 : 0111111101101110; -- 27 : (1.1016)^(-3/2)=0.86493
34 00011011 : 0111111101100000; -- 28 : (1.1055)^(-3/2)=0.86035
35 00011100 : 0111111101100110; -- 29 : (1.1094)^(-3/2)=0.85583
36 00011101 : 0111111101101100; -- 30 : (1.1133)^(-3/2)=0.85132
37 00011110 : 0111111101100110; -- 31 : (1.1172)^(-3/2)=0.84686
38 00011111 : 0111111101101101; -- 32 : (1.1211)^(-3/2)=0.84244
39 00100000 : 0111111101100101; -- 33 : (1.125)^(-3/2)=0.83804
40 00100001 : 0111111101101111; -- 34 : (1.1289)^(-3/2)=0.83371
41 00100010 : 0111111101100101; -- 35 : (1.1328)^(-3/2)=0.82941
42 00100011 : 0111111101101110; -- 36 : (1.1367)^(-3/2)=0.82513
43 00100100 : 0111111101100101; -- 37 : (1.1406)^(-3/2)=0.82089
44 00100101 : 0111111101100001; -- 38 : (1.1445)^(-3/2)=0.81668
45 00100110 : 0111111101100000; -- 39 : (1.1484)^(-3/2)=0.81253
46 00100111 : 0111111101101101; -- 40 : (1.1523)^(-3/2)=0.80841
```

Creating the Memory Initialization File

Creating the Memory Initialization File rsqrt_lookup_table_mif_gen.m

```
%-----  
% Create lookup table for (x_beta)^(-3/2)  
%-----  
Nbits_address = 8; % How many fraction bits will be used as the address?  
Nbits_output_fraction = 7; % The number of fractional bits in result. The output word size will be Nbits_output + 1, since we need a bit  
Nwords = 2^Nbits_address  
for i=0:(Nwords-1) % Need to compute each memory entry (i.e. memory size)  
    x_beta_table{i+1}.address = i; % Memory Address  
    fa = fi(i,0,Nbits_address,0);  
    fa_bits = fa.bin; % Memory Address in binary  
    fb = fi(0, 0, Nbits_address+1, Nbits_address); % Set number of bits for result  
    fb.bin = ['1' fa_bits]; % set the value using the binary representation. The address is our input value 1 <= x_beta < 2 where the leading  
    x_beta_table{i+1}.input_value = double(fb); % convert this binary input to it's double representation  
    x_beta_table{i+1}.input_bits = fa.bin; % keep track of the input bits  
    fy = fi(double(fb)^(-3/2),0,Nbits_output_fraction+1,Nbits_output_fraction); % compute (x_beta)^(-3/2) and convert to fixed-point  
    x_beta_table{i+1}.output_value = double(fy); % store the result as a double  
    x_beta_table{i+1}.output_bits = fy.bin; % store the resulting binary representation  
end
```


Creating the Memory Initialization File rsqrt_lookup_table_mif_gen.m

```
%-----  
% Create the Altera .mif file for the lookup table  
%-----  
fid = fopen('rsqrt_table.mif','w');  
%-----  
% Write File Header  
%-----  
line = ['DEPTH = ' num2str(2^Nbits_address) ';'']; % The size of memory in words  
fprintf(fid,'%s\n',line);  
line = ['WIDTH = ' num2str(Nbits_output_fraction+1) ';'']; % The size of data in bits  
fprintf(fid,'%s\n',line);  
line = ['ADDRESS_RADIX = BIN;']; % The radix for address values  
fprintf(fid,'%s\n',line);  
line = ['DATA_RADIX = BIN;']; % The radix for data values  
fprintf(fid,'%s\n',line);  
line = ['CONTENT'];  
fprintf(fid,'%s\n',line);  
line = ['BEGIN'];  
fprintf(fid,'%s\n',line); % start of (address : data pairs)
```

Creating the Memory Initialization File rsqrt_lookup_table_mif_gen.m

```
%-----  
% Write Memory Data  
%-----  
for data_index = 1:Nwords  
    line = [x_beta_table{data_index}.input_bits ' : ' x_beta_table{data_index}.output_bits ';    -- '  
           num2str(data_index) ' : (' num2str(x_beta_table{data_index}.input_value) ')^(-3/2)='  
           num2str(x_beta_table{data_index}.output_value) ];  
    fprintf(fid,'%s\n',line);  
end  
%-----  
% Write File End  
%-----  
line = ['END;'];  
fprintf(fid,'%s\n',line);  
%-----  
% Close File  
%-----  
fclose(fid);
```