Using Single Port RAM on a Cyclone V

Liam Ramsey

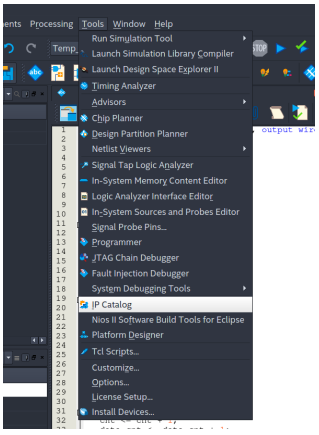
June 6, 2020

1 Adding the RAM to Your Project

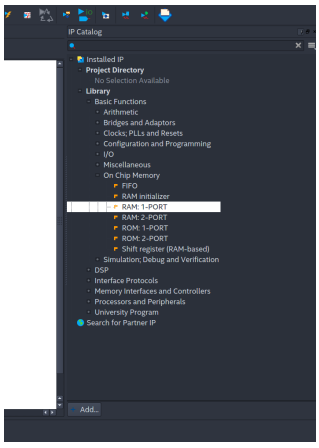
There are two ways you can add the RAM to your Quartus project, either with the IP Catalog or by writing it in by hand.

1.1 Using the IP Catalog

If you do not already have the IP Catalog open in Quartus, you can get to it by going to Tools→IP Catalog. From the IP Catalog you can go to Library→Basic Functions→On Chip Memory and double click RAM: 1-PORT. This opens the megafunction wizard for the module.



(a) IP Catalog



(b) RAM: 1-PORT

Figure 1

With the megafunction wizard open, you can specify the size in bits stored at each memory address by setting the width of the output bus "q". Setting how many words of memory specifies how many addresses we have to use, so total memory capacity will be number of words times how many bits per word. Leaving the rest to "Auto" and "Single clock" are fine, unless you know you need otherwise.

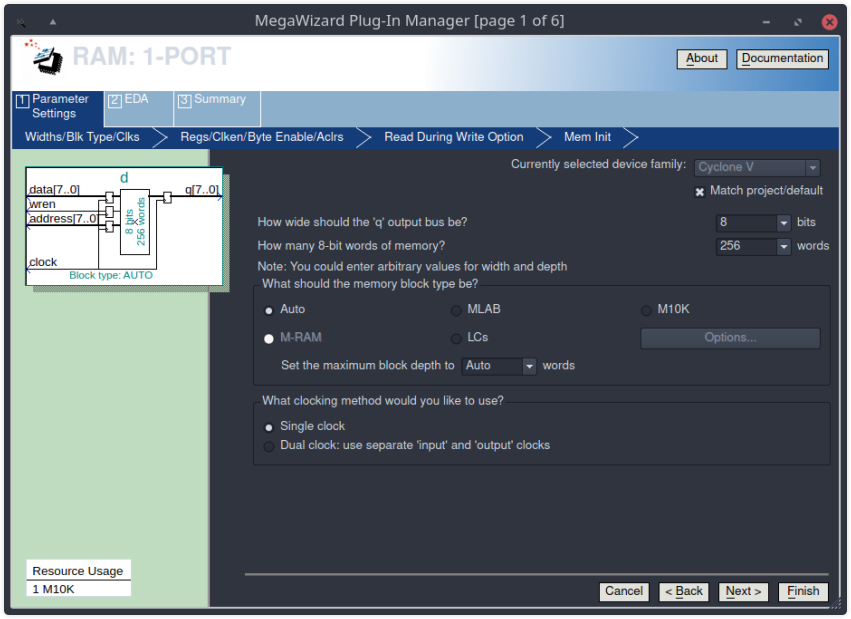


Figure 2: MegaWizard page 1

For pages 2-5 options can be left at default for a simple implementation. Things to note are the addition of an all clear signal and a read enable signal. On page 4 you can also specify initial contents of the memory. A .mif file can be generated with the mif python package.

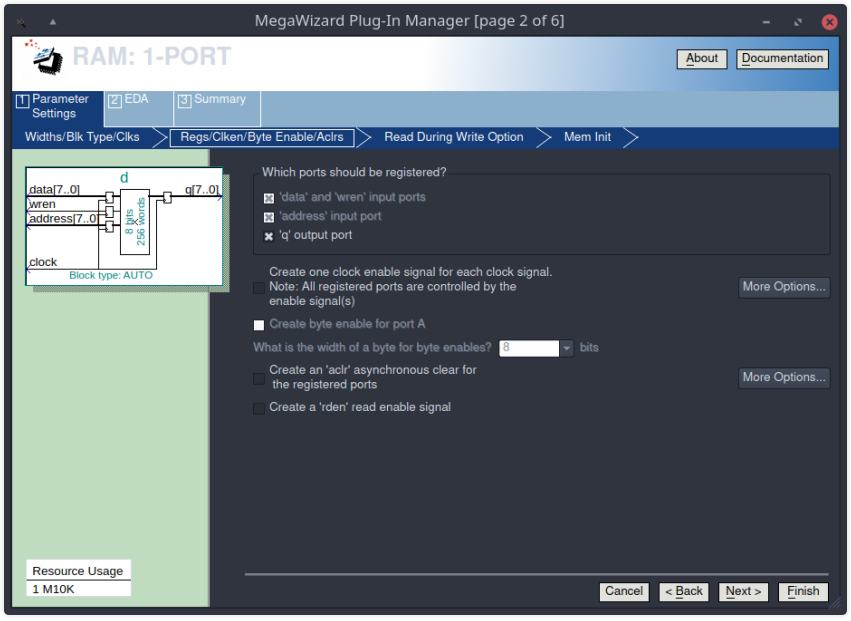


Figure 3: MegaWizard page 2

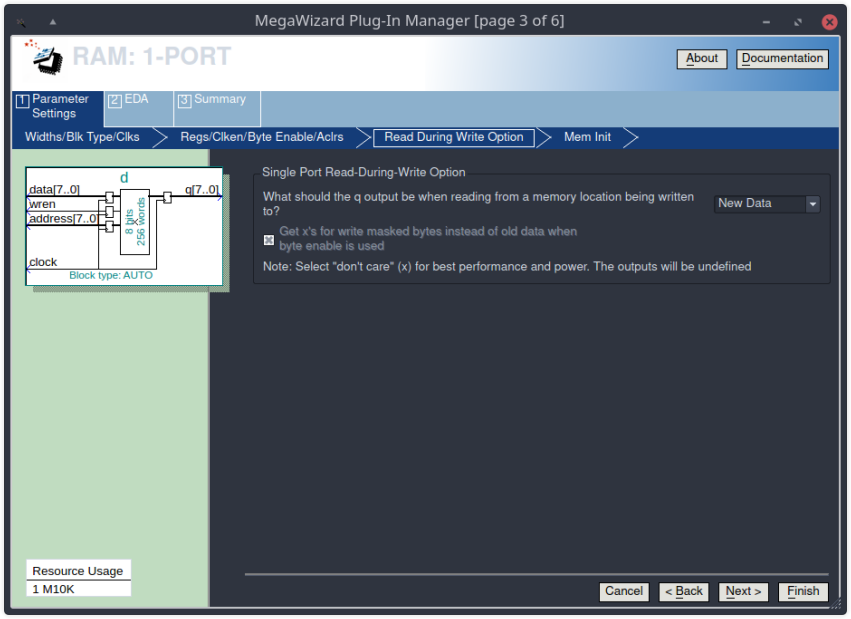


Figure 4: MegaWizard page 1

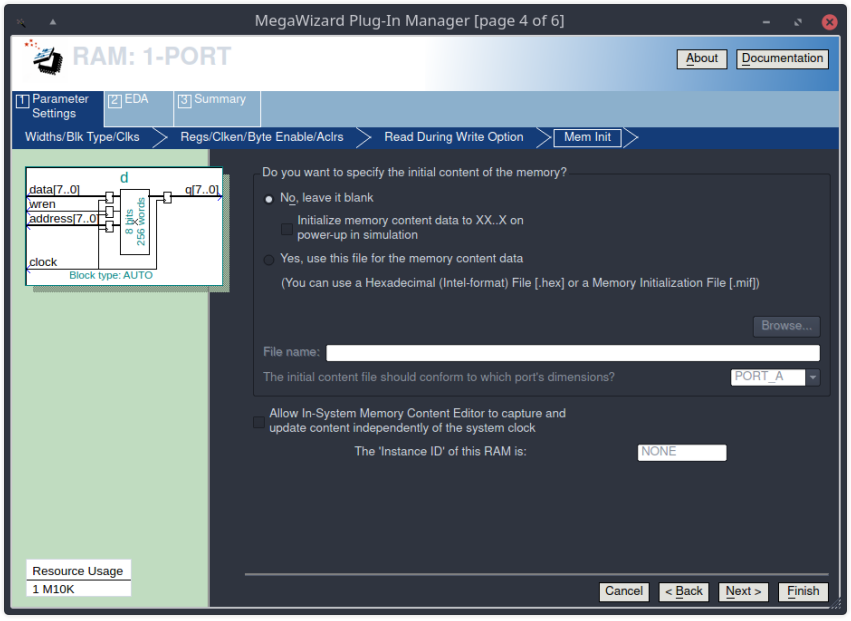


Figure 5: MegaWizard page 1

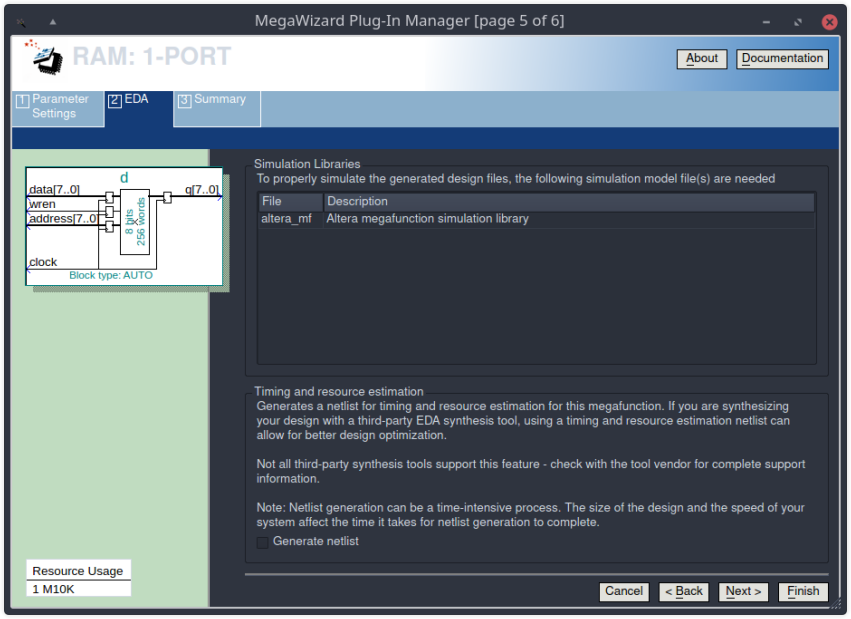


Figure 6: MegaWizard page 1

Here you can define which files Quartus will generate for you. The main <name>.v and <name>_bb.v files are checked by default. Additionally Quartus can generate <name>_inst.v, which contains an example if instantiating the module, you do not need to select this, as I will provide instantiation code here. After clicking finish, Quartus will prompt you to add the IP to your project.

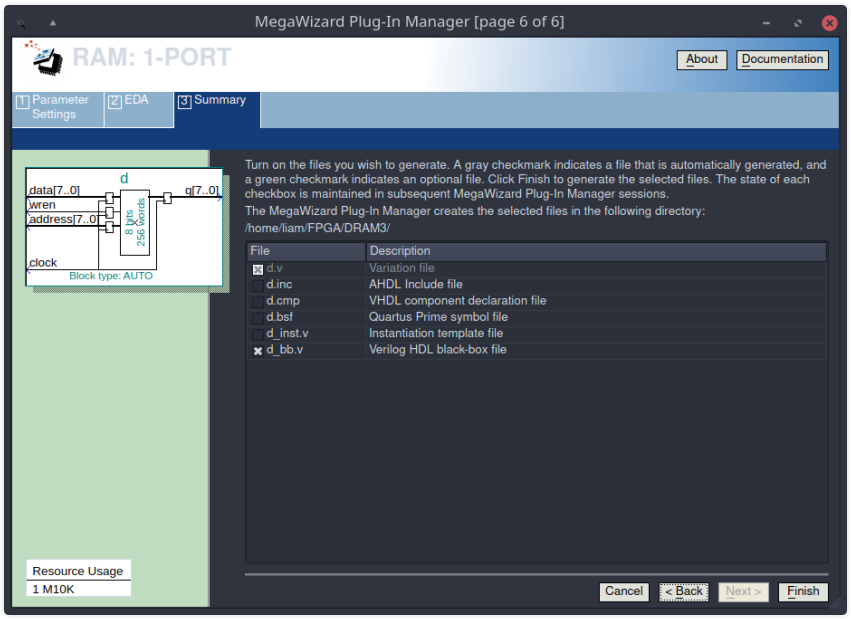


Figure 7: MegaWizard page 1

2 Using the Memory

2.1 Instantiating the module and writing to it

The memory can be instantiated with the following:

```
1 <name> ram_inst (
2     .address ( address_sig ),
3     .clock ( clock_sig ),
4     .data ( data_sig ),
5     .wren ( wren_sig ),
```

```
    .q ( q_sig )
);
```

Where you define the inputs and outputs before this with the names in the parentheses. Their sizes must match the values specified when the module was created. For example a RAM module with 8-bit words and a 8-bit address space would be:

```
reg address_sig[7:0];
reg data_sig[7:0];
wire wren_sig;
wire clk_sig;
reg q_sig[7:0];

assign clk_sig = clk;
```

Where clk is the input clock to your main module, and <name>is the name given to the RAM module. To write a value to memory, first the wren_sig wire must be set high (you could assign a register to it), then the desired values need to be put into the address_sig and data_sig registers. Some thought needs to be given to the TODO: Document undefined behavior in single clock cycles.

2.2 Interfacing With the Memory Using the In-System Memory Content Editor

The simplest way to read and write to the memory on the FPGA is to use Quartus's In-System Memory Content Editor. However, more functionality can be achieved by using the Python packaged described in the next section.

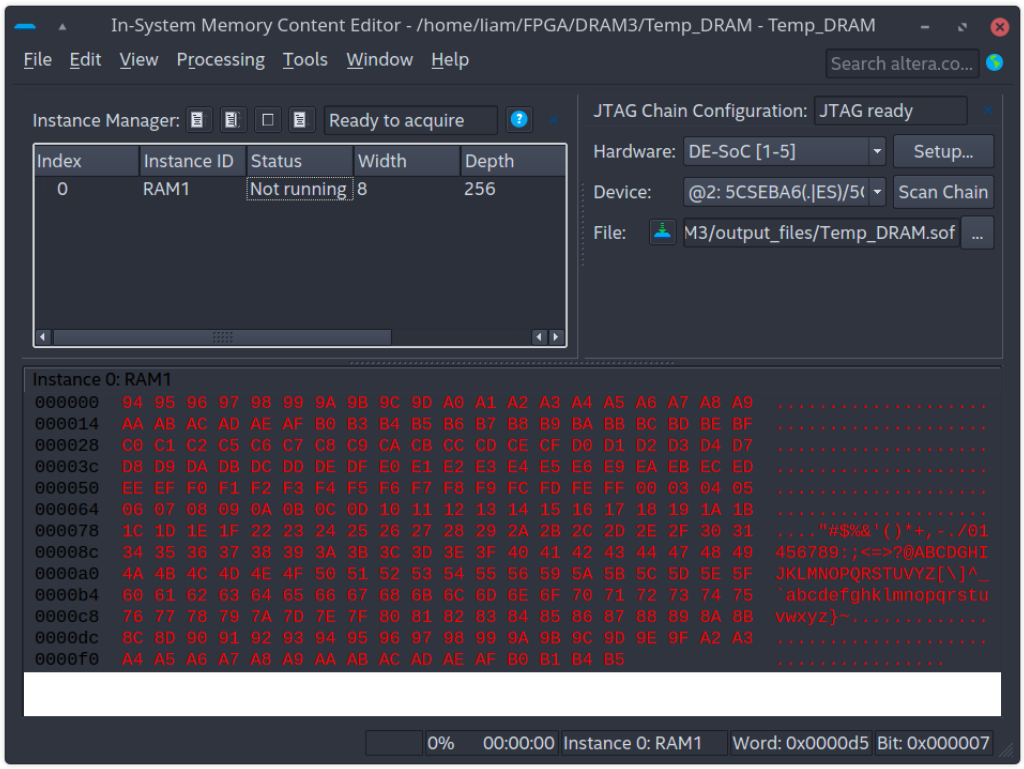


Figure 8: In-System Memory Content Editor

In the upper right section of the window, we have the basic functionality of the device programmer. Clicking the button directly to the right of "File:" will program the FPGA with the specified .sof file. The top left section lists the memory instances Quartus has found on the board. Right clicking an instance provides different options to read and write to the memory. The bottom section contains the memory currently selected in hex values on the left, and ASCII decoded on the right.

2.3 Using the mif and quartustcl Python Packages

In order to interface with the memory using these packages we will be using the QuartusMemory class. Code for the class is provided in Appendix A. Here is some example code using the class:

```
from QuartusMemory import QuartusMemory

q = QuartusMemory()

# Find index of instance named RAM1
inst = q.find_instance('RAM1')

# Read memory from device
arr = q.read_mem(inst,True)

# Print contents of address 0x04
print('Arr1:')
for k in arr[4]:
    print(str(k) + ' ',end='')
print()

# Copy the array and change one of the bits in 0x04
arr2 = arr
arr2[4][1] = 1

# Print the new array
print('Arr2:')
for k in arr2[4]:
    print(str(k) + ' ',end='')
print()

# Write the memory to the device
q.write_mem(inst,arr2,True)

# Read memory into a new array
arr3 = q.read_mem(inst,True)

# Print to confirm that memory was changed
print('Arr3:')
for k in arr3[4]:
    print(str(k) + ' ',end='')
```

```
print()
```

The code here simply reads the memory from the device into an array, changes a bit in it, writes it back to the device, then lastly reads to confirm the write executed correctly. The boolean argument at the end of each function call is whether to delete the .mif file generated. Both read and write default to not delete it. It is worth noting that running this script multiple times will overwrite and delete previously generated .mif files. If saving them is desired they should be renamed. The array format is of a two dimesional numpy array with type uint8 (unsigned 8-bit integer). This code can be found at https://github.com/Noeloikeau/TDC/tree/Liam in the Python folder.

A QuartusMemory Python Class

Code for the QuartusMemory class. Also available at https://github.com/Noeloikeau/TDC/tree/Li in the Python folder.

```
import os
import math
import mif
import quartustcl
import copy
import numpy as np



class QuartusMemory():

    def __init__(self,chip_number=0,fpga_number=1):

        self.quartus = quartustcl.QuartusTcl()
        self.hwnames = self.quartus.parse(self.quartus.get_hardware_names())
        self.hwname = self.hwnames[chip_number] # Picking first chip
```

```
17          self.devnames = self.quartus.parse(self.quartus.ge
            ↪  t_device_names(hardware_name=self.hwname))
18          self.devname = self.devnames[fpga_number] # Skip
            ↪  SOC chip, which is index 0
19          self.path=''
20          self.name='mem{0}.mif'
21
22
23
24      #Below finds instance index given a name (string)
25
26          def find_instance(self,inst_name,N_levels=2):
27
28          self.memories_raw = self.quartus.get_editable_mem_
            ↪  instances(hardware_name=self.hwname,\
29              device_name=self.devname)
30          self.memories =
            ↪  self.quartus.parse(self.memories_raw,
            ↪  levels=N_levels)
31          found_memid = None
32
33          for memid, depth, width, rw, type, name in
            ↪  self.memories:
34              if name == inst_name:
35                  found_memid = memid
36
37          if found_memid is None:
38              raise RuntimeError('Could not find memory
                ↪  '+inst_name)
39          return int(found_memid)
40
41
42
43      #Below reads memory from instance and returns as an
        ↪  array
44      #Generates intermediary MIF file which is then
        ↪  optionally deleted
```

```
45
46      def read_mem(self,inst,delete_mif=False):
47
48          fname=self.path+'r'+self.name.format(inst)
49
50          self.quartus.begin_memory_edit(hardware_name=self.
            ↪  hwname,\
51
52              device_name=self.devname)
53
54          self.quartus.save_content_from_memory_to_file(
55
56              instance_index=inst,
57
58              mem_file_path=fname,
59
60              mem_file_type='mif'
61
62          )
63
64          with open(fname, 'r') as f:
65              data = mif.load(f)
66              f.close()
67          self.quartus.end_memory_edit()
68
69          if delete_mif:
70              os.remove(fname)
71
72          return data
73
74
75      # Below writes memory to an instance from an array by
        ↪  writing data to mif file, then to instance
76      # Optionally will not delete temporary .mif
77
78      def write_mem(self,inst,data,delete_mif=False):
79
```

```
80          fname=self.path+'w'+self.name.format(inst)
81
82          self.quartus.begin_memory_edit(hardware_name=self.
            ↪  hwname,
            ↪  device_name=self.devname)
83
84          try:
85              with open(fname, 'w') as f:
86                  mif.dump(data, f)
87                  f.close()
88              self.quartus.update_content_to_memory_from_fil
                ↪  e(
89                  instance_index=inst,
90                  mem_file_path=fname,
91                  mem_file_type='mif',
92              )
93              self.quartus.end_memory_edit()
94          except:
95              self.quartus.end_memory_edit()
96          if delete_mif:
97              os.remove(fname)
```