Catapult C Lab 3

Using Memories in Catapult C

Introduction

Memories can automatically be used once they are made available in the technology window of Catapult C. One of the most important concepts in Catapult C is the idea of a Resource. A resource represents the hardware component and the variable represents the data moving through the hardware component. Catapult C automatically creates a resource for every variable that can have its hardware mapping controlled. Within Catapult C, multiple memory arrays can be mapped to the same hardware component, in this case a RAM.

An array is a pointer to an area of data. Arrays are important because they must be used for inferring RAMs. Any array in the design, including arrays on the ports, local variables and global variables can be mapped to memory.

This lab shows how arrays can be mapped to RAMs, how to combine variables on the design interface, and how to control the addressing into a RAM. It also illustrates some of the built-in memory optimizations,

Mapping Arrays to RAMs

Set the working directory

- ✓ Launch Catapult C and set the working directory to the lab3 directory

Add the input file

- ✓ Add the lab3.cpp file to the Catapult project. This design adds a four element array "a" to the previous values of a four element array "b" stored in an internal four element array "tmp". The code is shown below:

```
void lab3(int8 a[nums], int8 b[nums], int9 c[nums]){
  static int8 tmp[nums];
    c[0] = a[0] + tmp[0];
    c[1] = a[1] + tmp[1];
    c[2] = a[2] + tmp[2];
```

```
    c[3] = a[3] + tmp[3];
    tmp[0] = b[0];
    tmp[1] = b[1];
    tmp[2] = b[2];
    tmp[3] = b[3];
}
```

Setup Design

- ✓ Click on “Setup Design” in the design bar and set the technology to Altera Stratix II (Default part and speed grade).
- ✓ Set the clock frequency to **100MHz.**
- ✓ Include the Singleport Block RAM IO library by checking the box in the compatible libraries pane of the setup design dialog box (Figure 1).

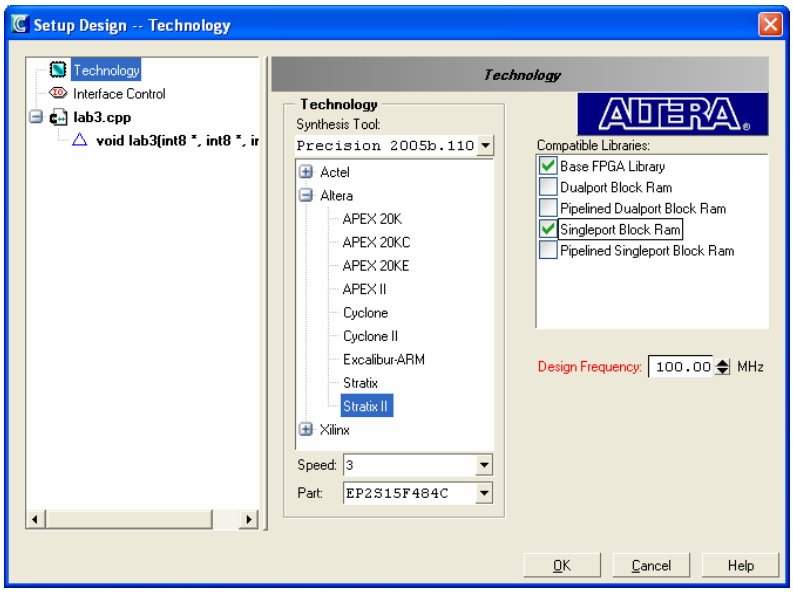


Figure 1 Adding Singleport RAMs

Architectural Constraints

- ✓ Click on the “Architectural Constraints” icon in the design bar. This will bring up a dialog, which will show you the ports and loops in your design.
- ✓ Expand the ports folder by clicking on the plus sign in the left pane of the dialog box.
- ✓ Also expand the internal arrays folder.

✓ Click on each resource and map them to the singleport RAM component (Figure 2). Note: you can select all resources at once by holding down the shift key while clicking. The select the singleport RAM.

RAM icons

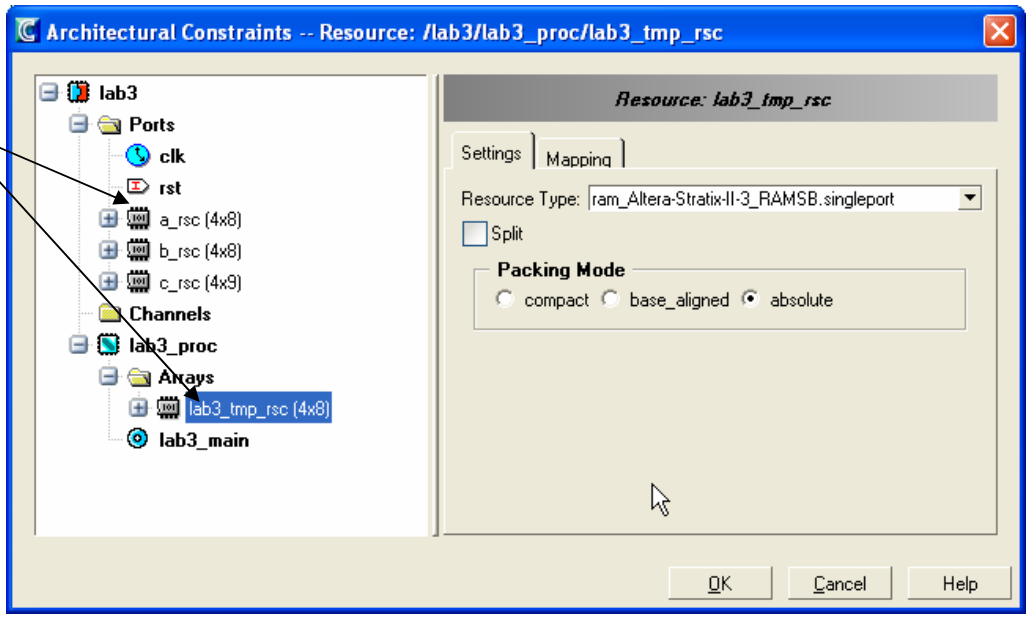


Figure 2 Mapping Resources to RAMs

Note that the resource icon changes when it is mapped to a RAM.

Generate RTL

✓ Click on the "Generate RTL" icon in the design bar. Double click on the Datapath Schematic icon (Figure 3).

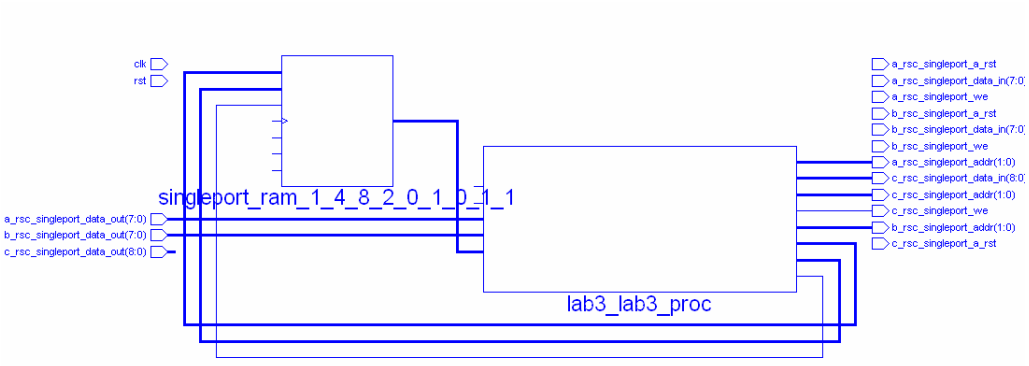


Figure 3 Datapath Schematic

Note that the arrays on the function interface have been mapped to singleport RAM interfaces. In addition to the 8-bit data port, address and control ports have been created for each RAM resource. The unused RAM ports are show tied together and are unused in the top-level RTL. The internal array "tmp" is mapped to a singleport RAM as a separate block of hierarchy.

Schedule

- ✓ Click on Schedule in the design bar to bring up the Gantt chart.
- ✓ Expand the lab3_main loop by clicking on the plus sign next to it in the loop hierarchy pane (Figure 4).

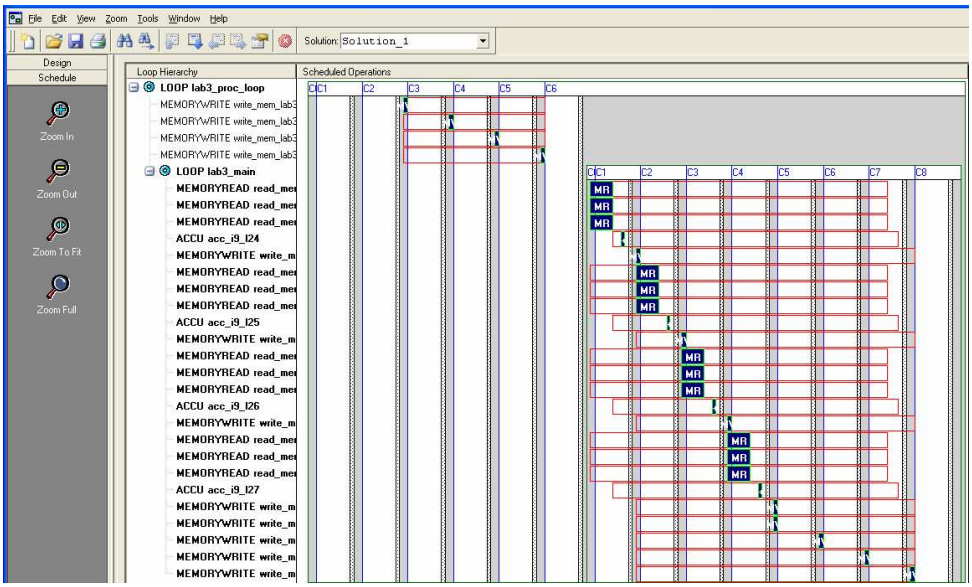


Figure 4 Schedule when mapping to RAMS

You will see two loops, the first loop initializes the internal ram, and the second loop implements the function body.

Note that the lab3_main loop takes 8 C-steps (clock cycles) to execute.

- ✓ To better understand why this is, right-click in the schedule window and select "view sorted -> by component utilization" (Figure 5).

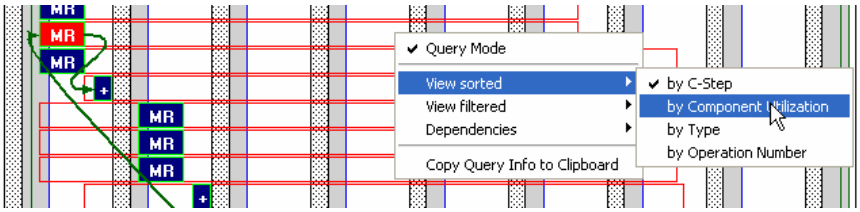


Figure 5 Setting Component Utilization View

The component utilization view shows how often a component is used in any give loop iteration. Each iteration of the main loop requires four reads on all input memory interfaces, and four writes on all output memory interfaces (Figure 6). This is because the memory interfaces are only 8-bits wide.

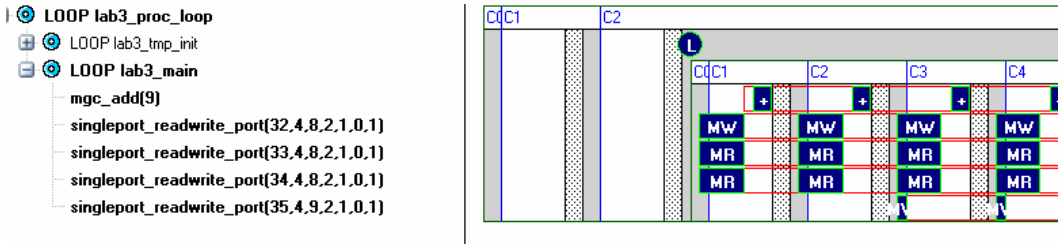


Figure 6 Component Utilization View

Setting Array Word Width

When mapping an array to memory, the mapping can be controlled using the word width constraint. The word width constraint allows the array to be re-mapped with a different number of bits per memory word. The most common and useful use of word width is to map more than one array word to one memory word to improve memory bandwidth. This is usually done in combination with loop unrolling which is covered in a later lab. Catapult will attempt to merge memory accesses if possible when the word width is made wider. The requirement for merging to take place is that the memory accesses must be sequential and must also start on an even word boundary.

- ✓ Click on architectural constraints and double the word width of all internal and external arrays (e.g a=16,b=16,tmp=16,c=18). This is done by expanding each resource and clicking on the variable associated with that resource (Figure 7).

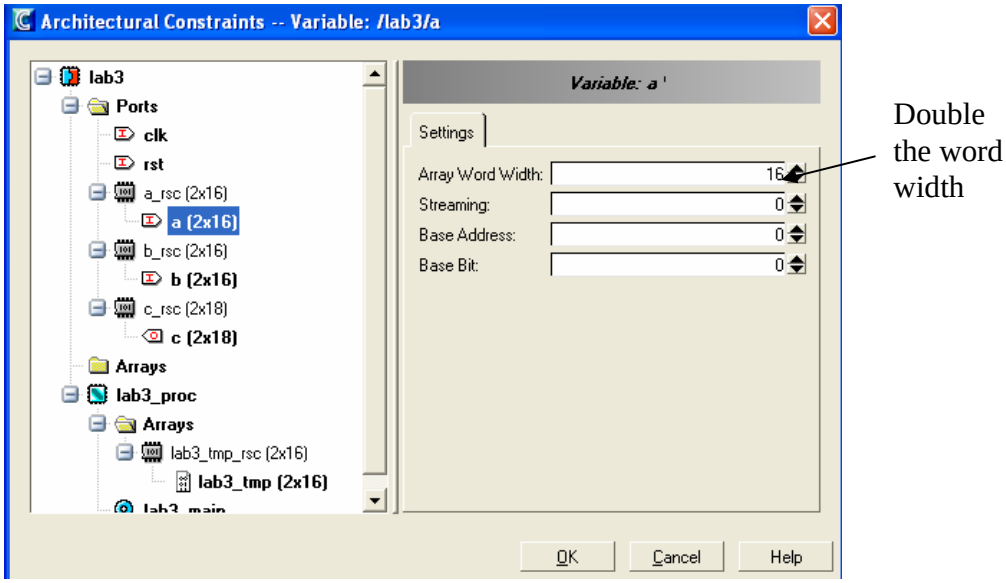


Figure 7 Setting the Word Width

- ✓ Re-schedule the design and examine the Gantt chart. Notice that the latency has been reduced by half. Doubling the word width has allowed Catapult to merge the reads and writes.

Combining RAM Interfaces

✓ Re-activate Solution_1 by double clicking on it in the design center window.

Catapult allows more than one variable to be associated with a resource. The previous example used a separate RAM for each interface variable. All the interface variables can be mapped to a single RAM resource.

✓ Open the architectural constrains window and expand all the resources in the ports folder. Click and drag the “b” and “c” variables onto the “a_rsc” resource (figure 8).

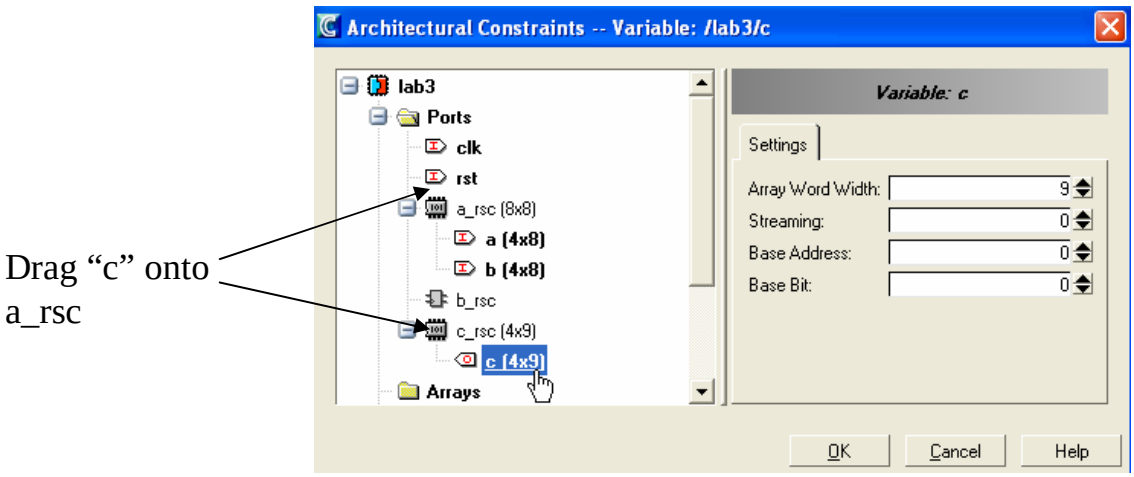


Figure 8 Combining Resources

✓ To see how the different variables have been mapped into a single RAM, click on the “a_rsc” resource and then click the mapping tab (Figure 9).

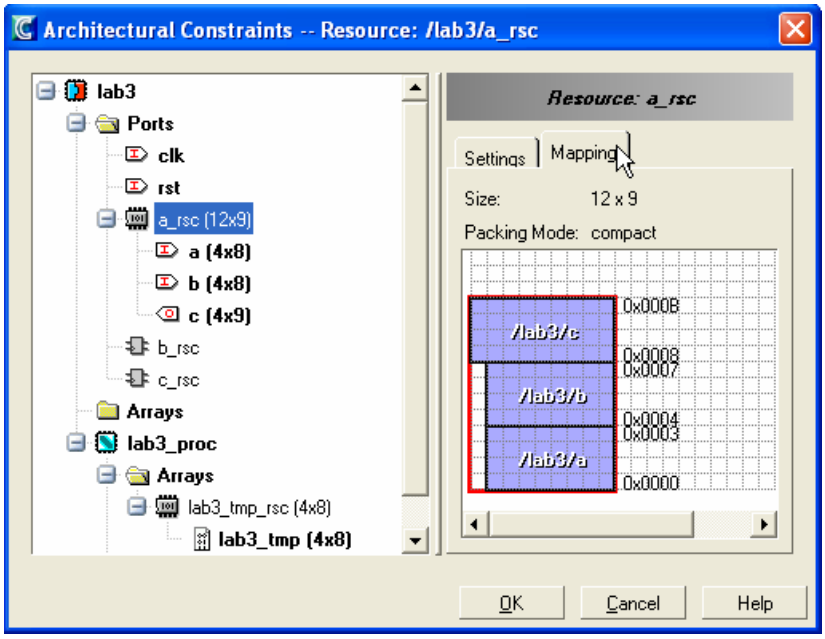


Figure 9 Combined RAM Address Mapping

The base address for each RAM default to the order in which they are defined in the C++ code. However Catapult allows you to re-organize your memories using the base address setting under the resource settings tab.

- ✓ Click on the “c” variable under the “a_rsc” resource and set the base address to 12 under the settings tab. Then click on the “a_rsc” mapping tab(Figure 10).

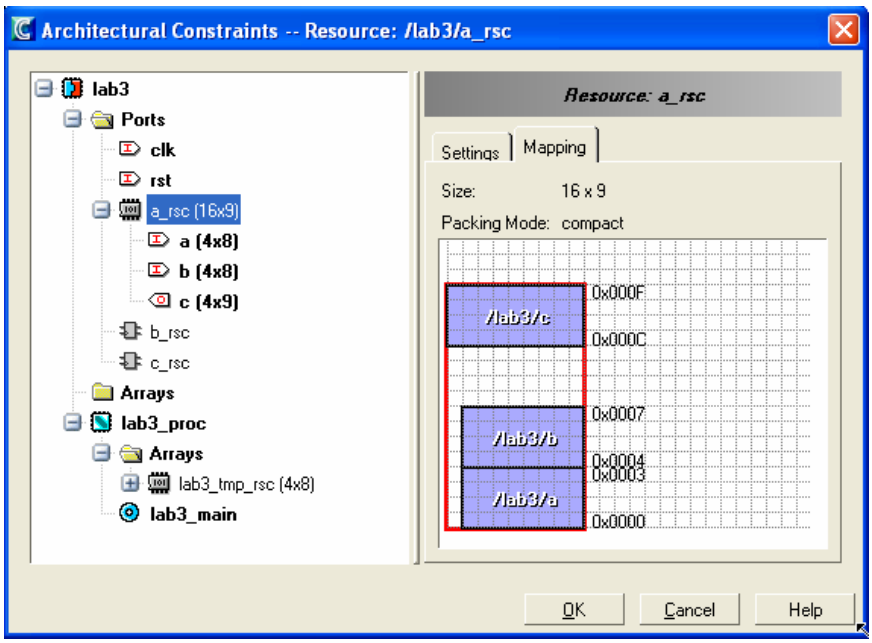
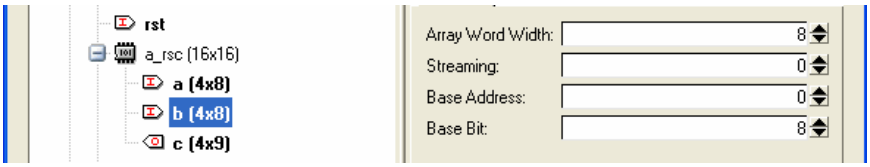


Figure 10 Setting the Base Address

Catapult also allows you to reorganize your memories by specifying where a variable should be mapped using the “base bit” setting. This allows you to place variables side-by-side in memory.

- ✓ Map “a” and “b” side by side by setting the base bit of b to 8 (Figure 11).



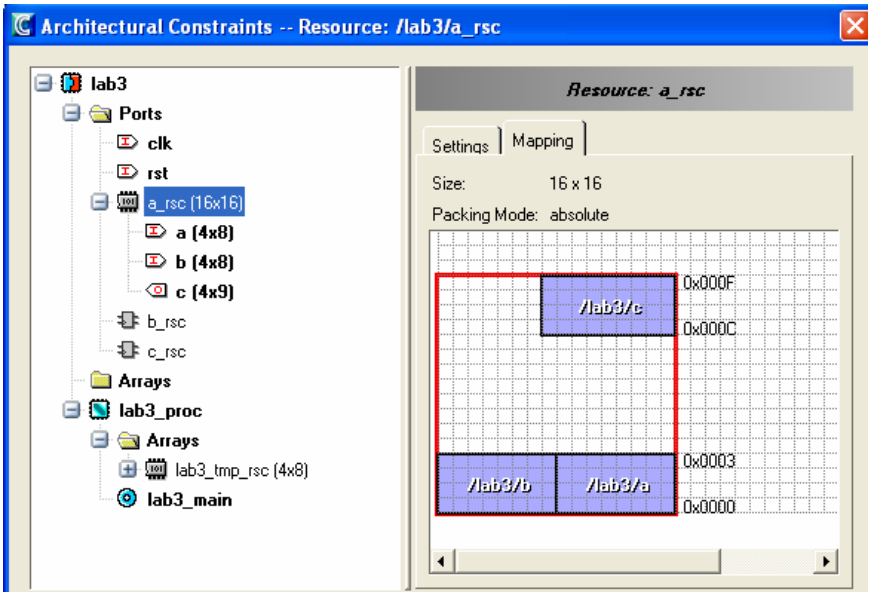


Figure 11 Setting the Base Bit

- ✓ Click Generate RTL and open the Cycle Report file. Expand the "memory resources" section. This will also give the address mapping for each memory resource (Figure 12).

```
32 I/O Data Ranges
42
43 Memory Resources
44   Resource Name: /lab3/a_rsc
45     Memory Component: singleport                    Size:          16 x 16
46     External:         true                          Packing Mode: absolute
47     Memory Map:
48     Variable Indices Phys Memory Address
49     -------- ------- -------------------------
50     /lab3/a      0:7 00000003-00000000 (3-0)
51     /lab3/b     8:15 00000003-00000000 (3-0)
52     /lab3/c      0:8 0000000f-0000000c (15-12)
53
54   Resource Name: /lab3/lab3_proc/lab3_tmp_rsc
61
```

Figure 12 Cycle.rpt

- ✓ View the Gantt chart and notice that the schedule is much longer than before. This is because all interface variables are now accessing the same 16-bit wide RAM.

Making Internal Arrays External

Catapult C allows internal resources to be mapped to the design interface by simply clicking and dragging that resource onto the ports folder. This is useful for mapping large internal arrays to external RAMs or when using ASIC RAM models. Dragging the resource to the interface will cause Catapult to create the necessary address, data, and control signals on the top-level RTL interface.

- ✓ Click on the architectural constraints icon in the design bar.

✓ Open the internal arrays folder and drag the “lab3_tmp_rsc” resource onto the ports folder, making it external (Figure 11).

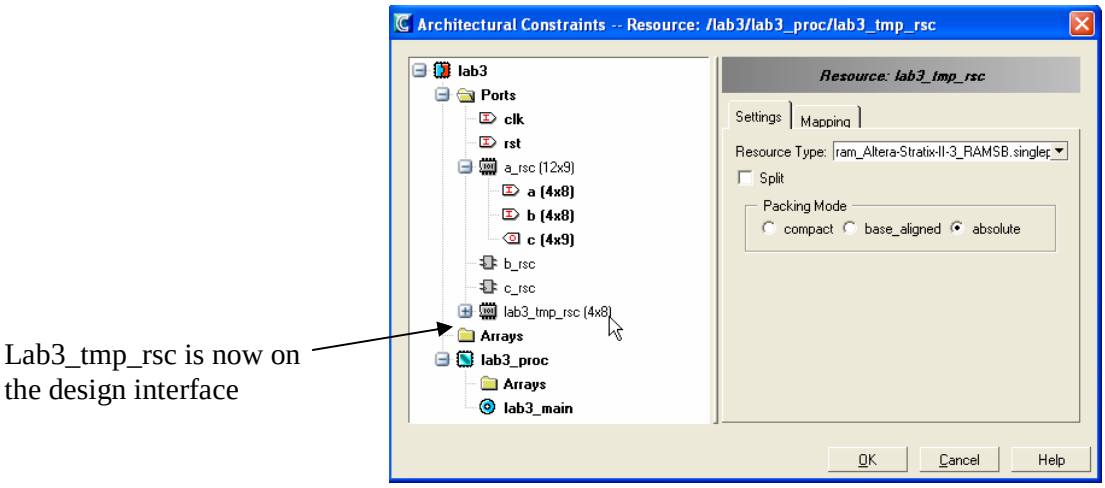


Figure 11 Making a Resource External

✓ Close all windows and select File->new project

END