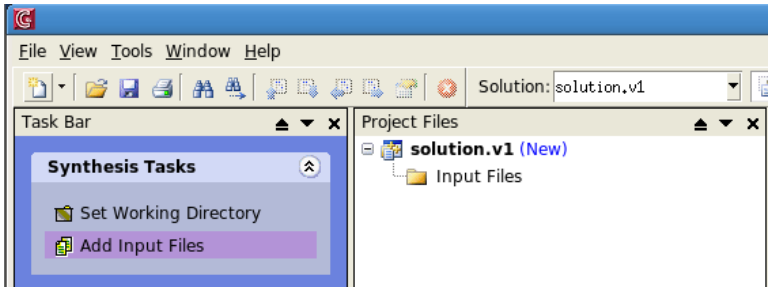
Catapult Basic Training 2011a

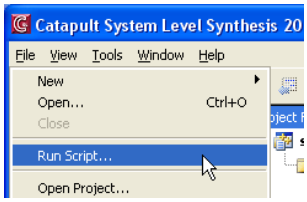
Lab 5 – Catapult Flow Walkthrough

This lab is intended to quickly take the user through the Catapult design flow so that they become familiar with the various stages of high level synthesis. Discussion of high-level synthesis constraints, writing synthesizable C++, and verification will be covered in the next lab. ***Please note that some of the options may be in different steps or tabs in the most recent version of the tool.***

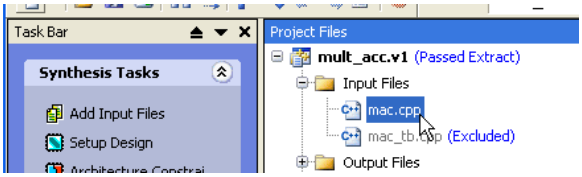
- ✓ Set the working directory to the Lab5 directory
- ✓ Launch Catapult from command line: catapult



- ✓ Select File > Run Script and source the directives.tcl file. This will add C++ input files and testbench and synthesize the design. We will then step through each phase of the synthesis flow.

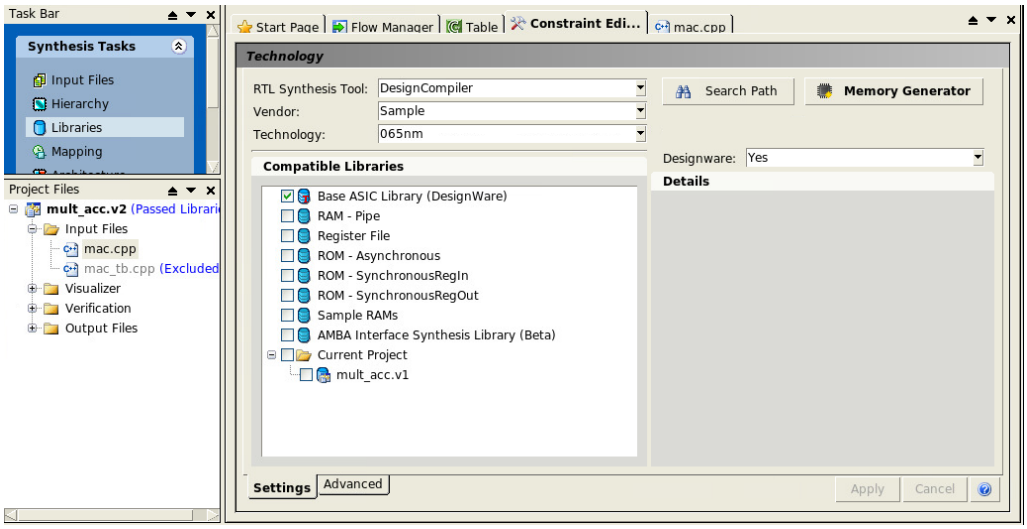


- ✓ Double-click on the mac.cpp file in the Input Files folder under Project Files. This will open the design that was just synthesized.

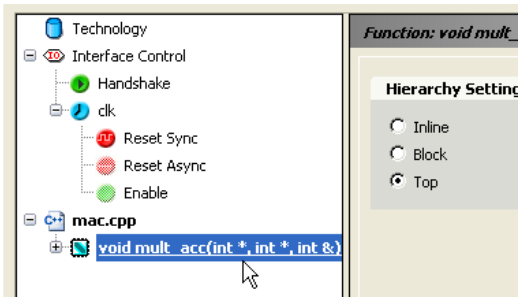


```
1  #pragma design top
2  void mult_acc(int a[4], int b[4], int &dout){
3    int acc=0;
4    MAC:for(int i=0;i<4;i++){
5      acc += a[i]*b[i];
6    }
7    dout = acc;
```

- ✓ Look at the C++ design and note that this is a simple multiply-accumulate algorithm that multiplies and adds two arrays of integers, a[4] and b[4]. You can see that the native C++ data types are used for this design. Also note that the "for" loop has been labeled and is called "MAC". We will see later that labeling loops is very useful for analyzing the design.
- ✓ Click on the Libraries icon in the Synthesis Tasks window pane. This is where we select the RTL synthesis tool, target technology, memory, and libraries, in the "Constraint Editor" tab.

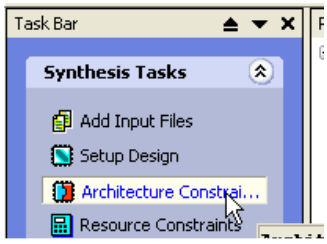


- ✓ Note that this design is setup for the DesignCompiler tool, 65nm technology.
- ✓ Click on Hierarchy under Synthesis Tasks. You will now see the C++ file list, and in particular the mac.cpp file.
- ✓ Click on mult_acc icon and note that this function has been set to be the top-level design. Double-click on the mult_acc or its icon to cross probe back to the C++ code.

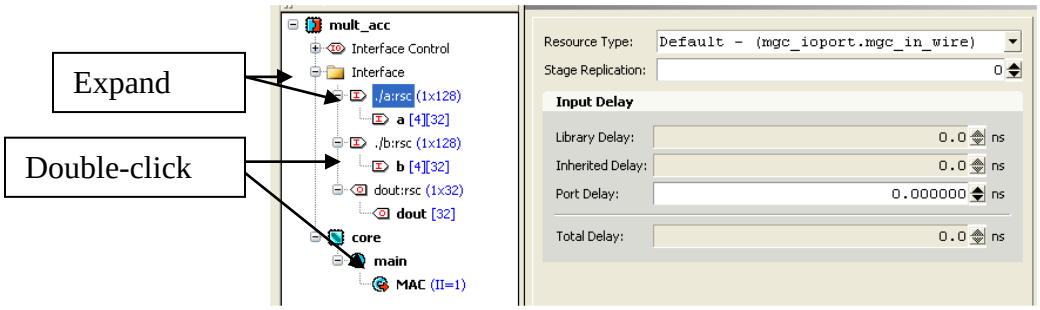


- ✓ Click on Mapping under Synthesis Tasks to open the Mapping Editor.
- ✓ Note that "Transaction Done Signal" is enabled. Enabling this signal will allow automated verification of the Catapult generated RTL against the original C++.

- ✓ This is where you can set the clock frequency. In the Module pane in the Constraint Editor tab, click “core (clk)”. Select 200 MHz and select Apply.
- ✓ Click on Architecture in the Synthesis tasks window pane (there is no Resource Constraints icon in the University version of the tool).



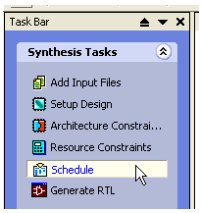
- ✓ In the Architecture view, Module pane, expand the Interface folder and expand the interface resources under the folder to see all the variables that have been mapped to a resource. Note that the resources icons indicate the port direction.
- ✓ Click on the a:rsc resource and note that the default interface is wire interfaces (No hand shake). Note that the “a” variable under the resource shows both the number of array elements from the C++, array elements == “[4]”, and the bit-width of the data type, int == “[32]”. Double-click on the “a” variable under the resource to cross probe back to the C++ and note that “a” is a four element array of type integer, “int a[4]”.
- ✓



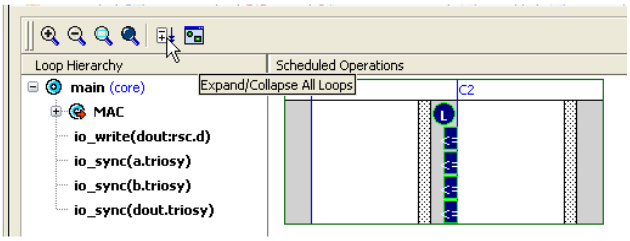
- ✓ Click on the MAC loop in the Architecture view. Note that the number of iterations for the MAC loop equals 4. You should also begin to see why labeling loops is useful. When there are many loops it is easier to know which is which when they are labeled. For now we will ignore the other loop settings and will revisit them in the next couple of labs.
- ✓ Double-click on the MAC loop to cross-probe back to the C++. Observe that the “for” loop in the C++ has 4 iterations

```
1   #pragma design top
2   void mult_acc(int a[4], int b[4], int &dout){
3     int acc=0;
4     MAC:for(int i=0;i<4;i++){
5       acc += a[i]*b[i];
6     }
7     dout = acc;
```

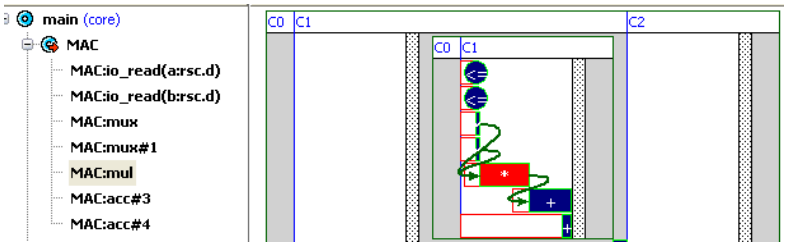
- ✓ Click on the Schedule in the Synthesis Tasks window pane.



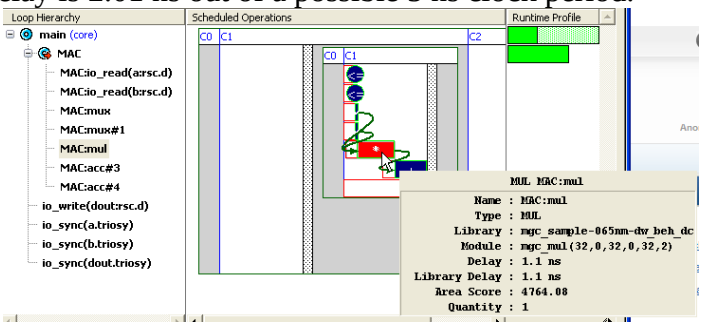
- ✓ Expand the loops in the Gantt chart by clicking on them (or by selecting the Expand all icon).



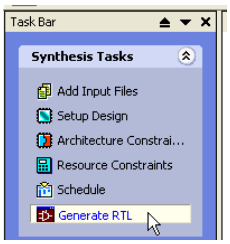
- ✓ Note the MAC loop shown on the left hand side. Labeling the loops in the C++ allows us to see where things come from in the Gantt chart.
- ✓ Double-click on the MAC loop to cross-probe back to the C++.
- ✓ Click on the multiplier in the Gantt chart. Note the green lines with arrows that show the data dependencies.



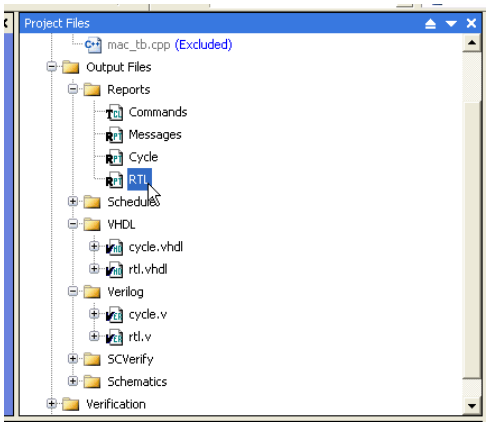
- ✓ Double-click on the multiplier to cross-probe back to the C++. This is the "*" used in the multiply-accumulate.
- ✓ Double-Click on the adder that is dependent on the multiplier. This is the add from the "+=" in the MAC loop.
- ✓ Mouse over the multiplier to see the component information. Note that the multiplier delay is 2.01 ns out of a possible 5 ns clock period.



- ✓ Look at the Loop Execution Runtime Profile in the top right of the Gantt chart. You can see that the MAC loop is where the algorithm is spending most of its time, indicated by the solid green bar.
- ✓ Click on RTL in the Synthesis Tasks window pane.



✓ In Project Files pane, open the Output Files folder, then Reports. Double-click to open the RTL.rpt file.



✓ Go to the BOM section of the report. Note that the final design has been built using one multiplier.

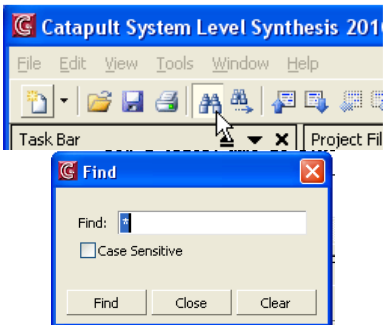
```
Design Total:                      11       4          6            0  0

Bill Of Materials (Datapath)
  Component Name                 Area Score Delay Post Alloc Post Assign
  ------------------------------ ---------- ----- ---------- -----------
  [Lib: mgc_ioport]
  mgc_in_wire(1,128)                  0.000 0.000          1           1
  mgc_in_wire(2,128)                  0.000 0.000          1           1
  mgc_io_sync()                       0.000 0.000          3           3
  mgc_out_stdreg(3,32)                0.000 0.000          1           1
  [Lib: mgc_sample-065nm-dw_beh_dc]
  mgc_add(2,0,2,1,3,4)               18.755 0.199          1           1
  mgc_add(32,0,32,0,32,3)           440.458 0.915          1           1
  mgc_and(1,2,4)                      2.400 0.030          0           1
  mgc_and(2,2,4)                      4.800 0.030          0           1
  mgc_and(32,2,4)                    76.800 0.030          0           1
  mgc_mul(32,0,32,0,32,2)          4764.082 1.059          1           0
  mgc_mul(32,0,32,0,32,3)          4710.423 1.419          0           1
  mgc_mux(32,1,2,3)                 127.903 0.081          0           1
```

✓ Open the VHDL (rtl.vhdl) or Verilog (rtl.v) output. Scroll to the bottom of the file and find the module or entity declaration.

```
260   module mult_acc (
261     a_rsc_z, b_rsc_z, dout_rsc_z, a_triosy_lz, b_trios
262   );
263     input [127:0] a_rsc_z;
264     input [127:0] b_rsc_z;
265     output [31:0] dout_rsc_z;
266     output a_triosy_lz;
267     output b_triosy_lz;
268     output dout_triosy_lz;
269     input clk;
270     input rst;
```

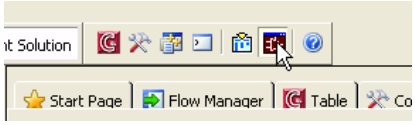
- ✓ Note that the module/entity has the same name as the C++ function. Double-click on the module name to cross-probe to C++.
- ✓ Note that the ports for "a" and "b" are flattened into wire interfaces: 32 bits per integer * 4 array elements = 128 wires
- ✓ Click on the Find Icon in the tool bar and search for a multiplier operation "*".



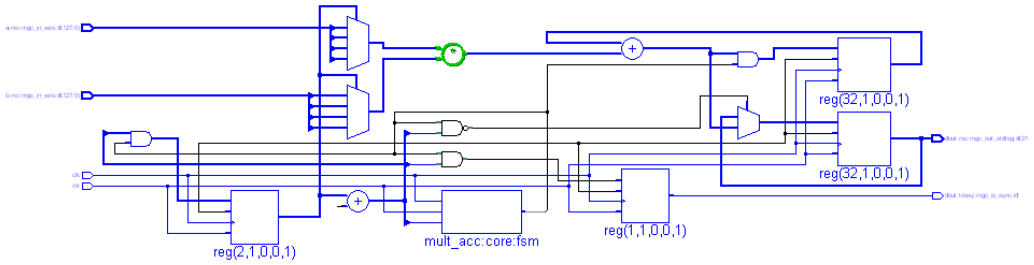
- ✓ Double-click on the multiplier in the RTL and cross-probe back to the C++.

```
105       .rst(rst),
106       .fsm_output(fsm_output),
107       .st_MAC_tr0((MAC_acc_4_tmp[2]))
108     );
109   assign dout_triosy_mgc_io_sync_ld = reg_a_triosy_mgc_io_sync_ld_cse;
110   assign MAC_mux_nl = MUX_v_32_4_1(((a_rsc_mgc_in_wire_d[31:0]) , (a_rsc_mgc_in_wire_d[63:32])
111       , (a_rsc_mgc_in_wire_d[95:64]) , (a_rsc_mgc_in_wire_d[127:96])), MAC_i_1_sva_2);
112   assign MAC_mux_1_nl = MUX_v_32_4_1(((b_rsc_mgc_in_wire_d[31:0]) , (b_rsc_mgc_in_wire_d[63:32])
113       , (b_rsc_mgc_in_wire_d[95:64]) , (b_rsc_mgc_in_wire_d[127:96])), MAC_i_1_sva_2);
114   assign acc_sva_1 = acc_sva + conv_s2s_64_32((MAC_mux_nl) , (MAC_mux_1_nl));
115   assign MAC_acc_4_tmp = conv_u2u_2_3(MAC_i_1_sva_2) + 3'b1;
```

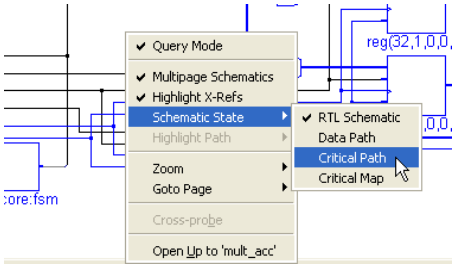
- ✓ Open the RTL schematic by clicking on the schematic icon in the tool bar.



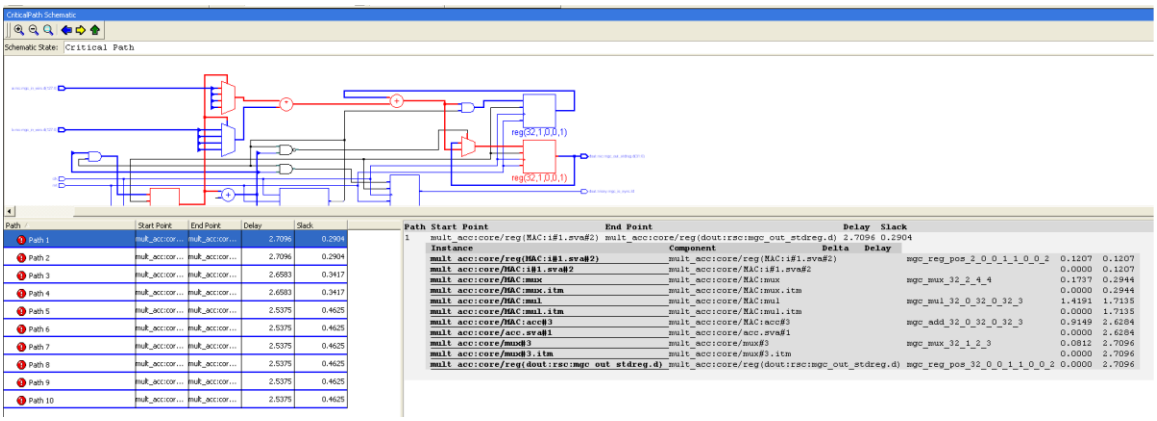
- ✓ Double-click on the block to push down into its schematic



- ✓ Note that the design is built with a single multiplier that is shared (MUXes on inputs) to implement the multiply-accumulate of "a" and "b". Double-click on it to go back to the C++.
- ✓ Right-click in the schematic and set the schematic state to Critical Path

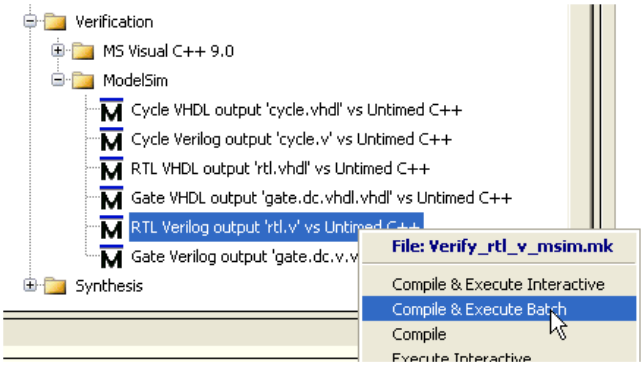


Click through some of the paths shown. Note how the path is highlighted in the schematic with the timing paths shown on the right.

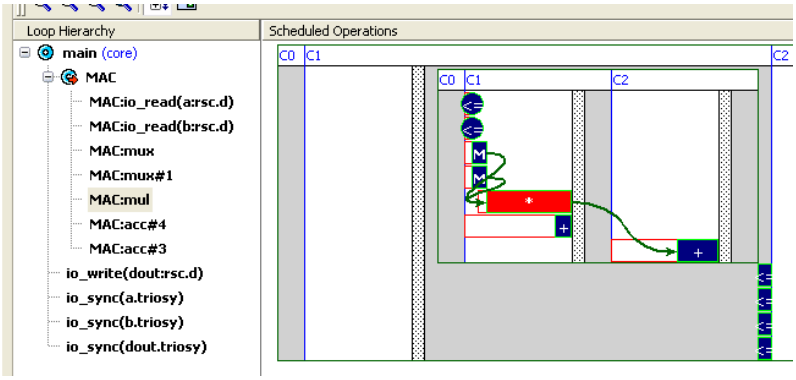


- ✓ Select the Flow Manager tab in the right window, then click on the SCVerify item and check that it is Enabled. If it was not (for any reason) you will now see that the tool generates the SystemC files and Modelsim scripts that are needed for C++/RTL co-simulation.
- ✓ Open the Verification > Modelsim folder in the Project Files tab and right click on either the VHDL or Verilog RTL verification flow. Select Compile and Execute Batch. This will automatically verify the C++ against the RTL. You should see a "PASSED" message in the transcript. This means that the RTL behaves exactly like the C++ code (i.e. it is as "correct" as the C++ code).

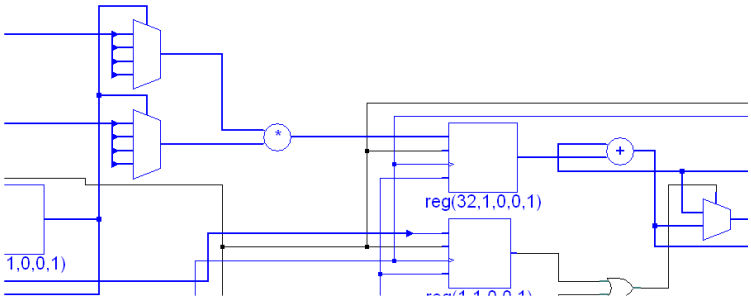
- ✓ If you want to look at the waveforms, select Compile and Execute Interactive, and then the “Run -All” button. Of course, you should know how to use Questa Sim (not covered in this course) if you want to analyze the waveforms.



- ✓ As a final step go back to Mapping in the Synthesis Tasks panel, click “core (clk)” under the Module pane, set the clock frequency to 800MHz and click Apply.
- ✓ Click on Schedule in the Synthesis Tasks window pane and open the Gantt chart. You should now see the multiplier and adder of the MAC scheduled in different c-csteps because of the increase in clock frequency. This means that a register will be inserted between them since the data dependency is closing a clock boundary.



- ✓ Generate RTL and open the RTL schematic. Find the multiplier and adder from the MAC loop. You should see a register between them now. In other words high-level synthesis automatically added more pipeline registers based on the clock frequency and target technology in order for the design to meet timing. Although this example is simplistic it illustrates one of the most powerful aspects of high-level synthesis in that the algorithm becomes somewhat independent of the target technology and design constraints.


*
+
reg(32,1,0,0,1)
1,0,0,1)