

# Catapult C Lab 4

## Loop Unrolling and Loop Pipelining

©Copyright Mentor Graphics Corporation 1995-2003. All rights reserved.

This document contains information that is proprietary to Mentor Graphics® Corporation. The original recipient of this document may duplicate this document in whole or in part for internal business purposes only, provided that this entire notice appears in all copies. In duplicating any part of this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use and distribution of the proprietary information.

Trademarks that appear in Mentor Graphics product publications that are not owned by Mentor Graphics are trademarks of their respective owners.

## Introduction

Within Catapult C, using loops is one of the major ways to optimize a design. You can control loops using.

- **Loop iterations:** Loop iterations are the number of times the loop runs before it exits. Loop iterations are a constraint to tell Catapult C that its estimate for the number of iterations in a loop is incorrect.
- **Loop unrolling:** Loop unrolling is the number of times to copy the loop body. After the loop iterations have been determined, the loop is unrolled based on the unrolling constraints. This will change the number of iterations of the loop, which is reflected in the reporting, but it won't change the iterations directive.
- **Loop merging:** Loop merging is a technique used to reduce latency and area consumption in a design by allowing parallel execution, where possible, of loops that would normally execute in series. Loop merging is applied only after loop iterations have been established and loop unrolling and memory mapping are done.
- **Loop pipelining:** Loop pipelining is how often to start the next iteration of the loop. After loop unrolling and several other transformations are complete, the scheduler uses the pipelining constraints to build a pipelined loop. This happens much later than the other two transformations, and pipelining will not occur if the loop has been optimized away because of the first two constraints.

This lab covers some of the basic optimization and analysis steps using loops.

## Loop Unrolling

### Set the working directory

- ✓ Launch Catapult C and set the working directory to the lab4 directory

### Add the input file

- ✓ Add the loop\_unrolling.cpp file to the Catapult project. This design adds two 8-element arrays together. The code is shown below:

```
void lab4(int a[nums], int b[nums], int c[nums]){  
  
    ACC:for ( int i = 0; i < nums; i++)  
        c[i] = a[i] + b[i];  
}
```

## Setup Design

- ✓ Click on the setup design icon in the design bar
- ✓ set the technology to Xilinx VIRTEX-E(Default part and speed grade).
- ✓ Set the clock frequency to 50 MHz.
- ✓ Include the Singleport Block RAM library.

## Loop Iterations

Iteration count is very important for optimization and for the latency estimates provided by Catapult C. If Catapult C can determine an iteration count, it is pre-filled in the Iteration Count field on the Architectural Constraints screen.

- ✓ Bring up the architectural constraints window and click on the ACC loop.
- ✓ Verify that the number of iterations is reported in the iteration count box (Figure 1).
- ✓ Double click on the ACC loop to cross-probe back to the source code and verify that the number of iterations reported matches the number of iterations in the “for” loop. Notice that the arrays on the design interface have been automatically mapped to singleport RAM interfaces. This is done automatically for large arrays exceeding a threshold specified under tools->set options-> Architectural.

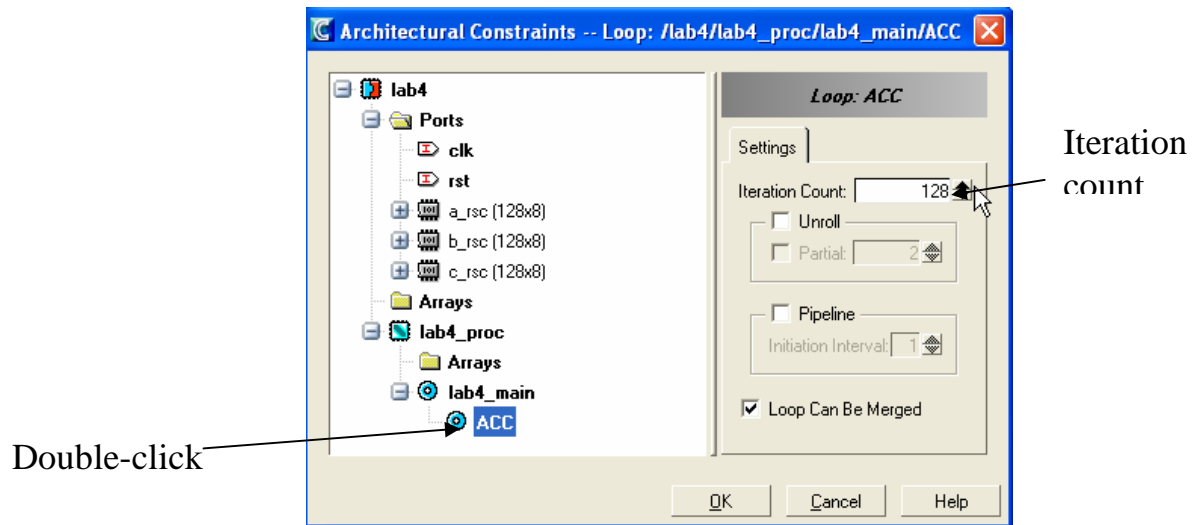


Figure 1 Checking Iteration Count

- ✓ Click OK and then generate RTL.
- ✓ Open the data path schematic and push down into the hierarchy by double-clicking on the lab4\_lab4\_proc block.
- ✓ Verify that only two adders are used in the un-optimized design (One is for the loop increment). E.g. all loops are left rolled (Figure 2) . Note that there is also an “increment” operation that is used to count the loop iteration.

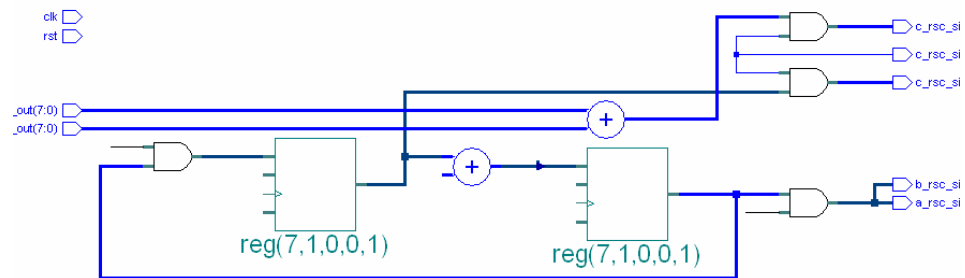


Figure 2 Datapath Schematic with Loop Left Rolled

- ✓ Double-click on the RTL Report in the Reports folder
- ✓ expand the Bill of Material section. Verify that the number of adders assigned is two, one for the ACC loop and one for the increment operation (Figure 3).

Bill Of Materials (Datapath)					
Component Name	Area Score	Area(Function_Generators)	Post Alloc	Post Assign	
[Lib: mgc_Xilinx-VIRTEX-E-6_beh_psr]					
mgc_add(7,0,2,1,8)	6.890	6.890	1	1	
mgc_add(8,0,8,0,8)	8.322	8.322	1	1	
mgc_and(1,2)	0.385	0.385	0	1	
mgc_and(7,2)	2.692	2.692	0	3	

Figure 3 RTL Report

- ✓ Click on Schedule in the design bar to open the Gantt chart
- ✓ Expand the LOOP ACC loop to see the schedule (Figure 4).

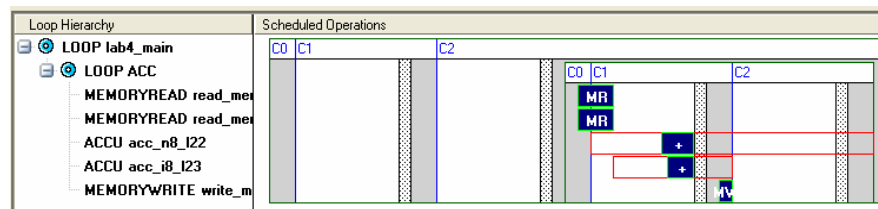


Figure 4 Schedule with Loops Left Rolled

- ✓ Click on the “Design Center” tab and then on the “Table” view. Verify the latency and throughput of the design, they should both be 258 (Figure 5)

Project Files

XY Plot

Bar Chart

Table

Solution	Status	Total Area	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time
Solution_1	Passed extract	41	258	5160.00	258	5160.00

Figure 5 Table View

- ✓ Double-click on the Cycle Report icon in the Reports folder
- ✓ Expand the “Loops” section. The overall latency as well as individual latency is reported here (Figure 6).

Multi-cycle (combinational) component usage							
Loops							
Process	Loop	Iterations	C-Steps	Latency	Duration	Unrol.	
/lab4/lab4_proc	lab4_main	Infinite	2	258	5.16 us		
/lab4/lab4_proc	ACC	128	2	256	5.12 us		

Figure 6 Cycle Report

## Loop Unrolling

Loop unrolling is a technique that allows loop iterations to execute in parallel whenever possible. After the loop iterations have been determined, the loop is unrolled based on the unrolling constraints. This will change the number of iterations of the loop, which is reflected in the reporting, but it won't change the iterations directive. During loop

unrolling, the body of the loop is copied several times. The result is that the loop is either completely dissolved, or a new loop is created that iterates fewer times. Adding more parallelism into a design decreases the latency and throughput while increasing the area.

- ✓ Click on the Architectural Constraints icon in the design bar and then on the ACC loop.
- ✓ Partially unroll the loop two times by checking the “Unroll” and “Partial” check boxes and set the partial unroll count to two (Figure 7). Note the change in the loop icon.

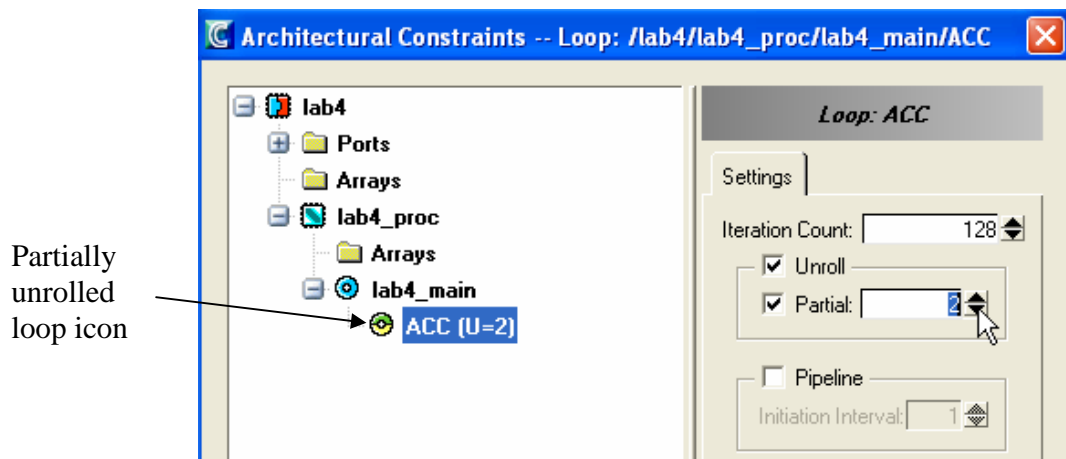


Figure 7 Partially Unrolling a Loop

- ✓ Schedule the design and generate RTL.
- ✓ Check the latency. Note that the design did not get much faster but is much larger.
- ✓ Open the Gantt chart and right-click and select “View sorted -> Component Utilization view”. Notice that there are two reads on each of the “a” and “b” RAM interfaces (Figure 8). Remember from the previous lab on memories that widening the word width is typically required to merge memory accesses.

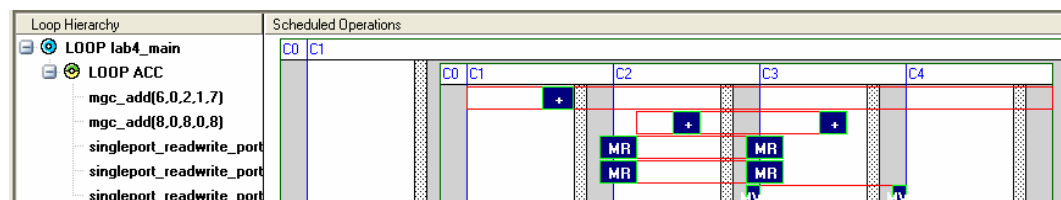


Figure 8 Partially Unrolling Without Changing Word Width

- ✓ Go back to the architectural constraints view and set the word width of a, b, and c to 16 (Figure 9).

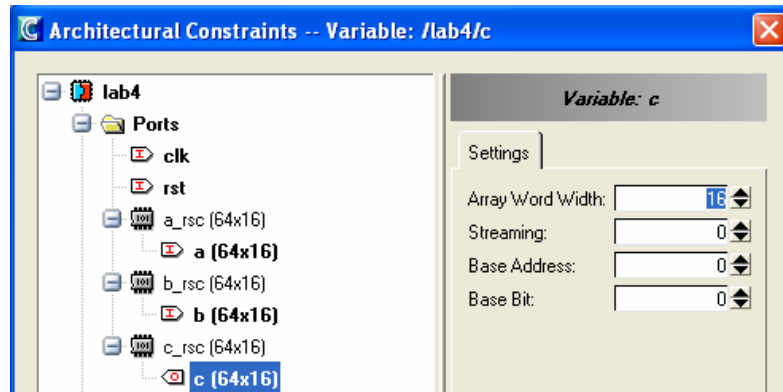
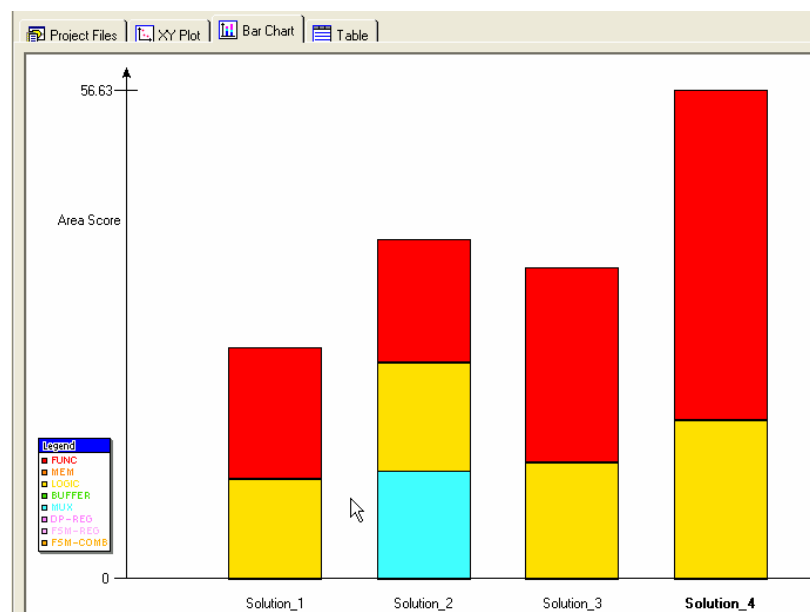


Figure 9 Setting Word Width to 16

- ✓ Generate the RTL and check the latency and area. How many adders are used?
- ✓ Re-check the schedule.
- ✓ Go back to architectural constraints and set the word width to 32 on a,b, and C.
- ✓ Then set the partial unroll count to 4.
- ✓ Regenerate the RTL and reanalyze the design.
- ✓ Look at the table, bar chart, and XY plot views in the design center (Figure 10).

Project Files   XY Plot   Bar Chart   Table						
Solution	Status	Total Area Score	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time
Solution_1	Passed extract	26.75	258	5160.0	258	5160.0
Solution_2	Passed extract	39.29	257	5140.0	257	5140.0
Solution_3	Passed extract	35.99	130	2600.0	130	2600.0
Solution_4	Passed extract	56.63	66	1320.0	66	1320.0



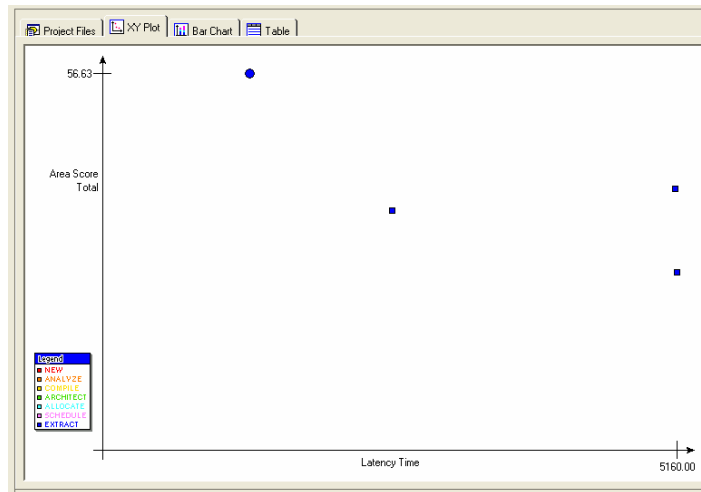


Figure 10 Table, Bar Chart, and XY View

- ✓ Close all windows and select File->new project

## Loop Merging

Catapult will automatically merge loops, when possible, to allow them to run in parallel. The loop merging constraint is on by default, however it can be turned off.

## Create a New Project

- ✓ Goto the File menu and select “New Project”.

## Add the input file

- ✓ Add the loopmerging.cpp file to the Catapult project. This design has two loops that add two arrays together. The loops are independent of one another. The code is shown below:

```
loop_merge0(int a[10], int b[10], int c[10],int x[10],int y[10],int z[10]){
    LOOP0:for ( int i = 0; i < 10; i++)
        a[i] = b[i] + c[i];
    LOOP1:for ( int i = 0; i < 10; i++)
        x[i] = y[i] + z[i];
}
```

## Setup Design

- ✓ Click on the setup design icon in the design bar
- ✓ Set the technology to Xilinx VIRTEX-E(Default part and speed grade).
- ✓ Set the clock frequency to 100 MHz.

## Architectural Constraints

- ✓ Bring up the architectural constraints dialog
- ✓ Disable loop merging on both loops in the design. Notice that the icon changes for loops that can't be merged (Figure 11).

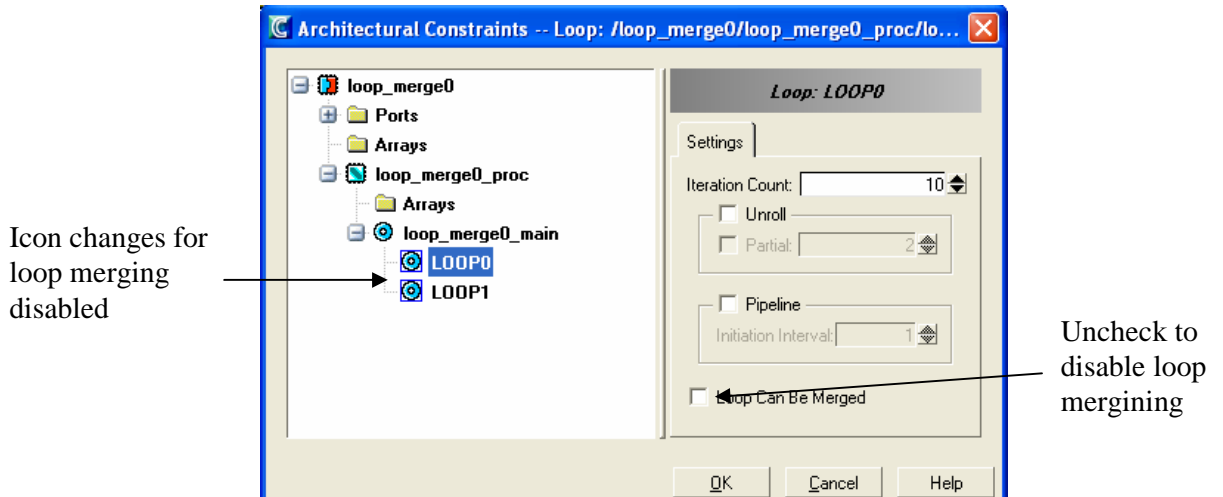


Figure 11 Disabling Loop Merging

- ✓ Click on the Schedule button to bring up the Gant chart (Figure 12). You can see both loops are scheduled
- ✓ Hover the mouse over the main loop to see the total latency in the design. Then hover over each individual loop. Note that the total latency is equal to the sum of the two loops plus 2 clock cycles latency for the main loop. These loops do not run in parallel.

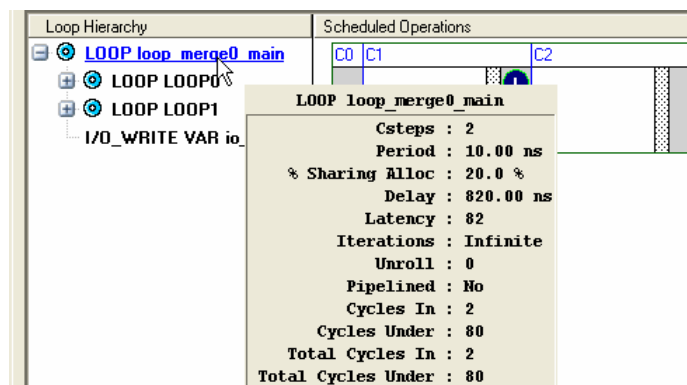


Figure 12 Disable Loop Merging

- ✓ Go back to architectural constraints and enable loop merging on LOOP0 and LOOP1.



- ✓ Reschedule the design and view the Gantt chart. How many loops are there? What is the latency? The loops have been merged and scheduled in parallel.
- ✓ Goto to the transcript window and scroll up to see what loops have been merged (Figure 13)

```
// tmp_loopmerge0.cpp(2): Loop '/loop_merge0/loop_merge0_proc/loop_
# indp_loopmerge.cpp(2): Loop '/loop_merge0/loop_merge0_proc/loop_
# Loop 'LOOP1' is merged and folded into Loop 'LOOP0' (LOOP-9)
# I/O-Port inferred - resource 'a_rsc' (from var: a) mapped to 'm
# I/O-Port inferred - resource 'b_rsc' (from var: b) mapped to 'm
..
```

Figure 13 Transcript Message

- ✓ Close all windows and select File->new project

## Loop Pipelining

Loop Pipelining is not the same pipelining that RTL designer's use. Loop pipelining is similar to the pipelining done in CPUs where the second iteration/operation is started before the first one finishes. Just like in loop unrolling, loop pipelining is limited by the looping dependency chains in your loop. Internal loops can be pipelined to reduce latency. The outermost loop can be pipelined to produce a throughput driven design.

## Create a New Project

- ✓ Goto the File menu and select “New Project”.

## Add the input file

- ✓ Add the MAC.cpp file to the Catapult project. This design multiplies and accumulates two arrays.

```
void MAC(uint16 a[nums], uint16 b[nums], uint32 *c){
    int tmp=0;
    MAC:for ( int i = 0; i < nums; i++)
        tmp += a[i] * b[i];
    *c = tmp;
}
```

## Setup Design

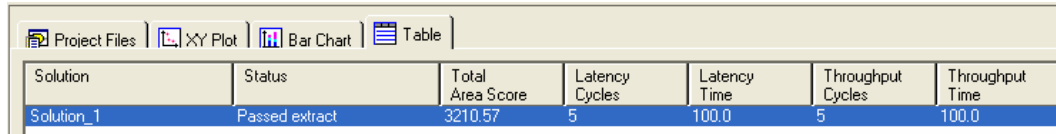
- ✓ Click on the setup design icon in the design bar
- ✓ Set the technology to Xilinx VIRTEX-E(Default part and speed grade).
- ✓ Set the clock frequency to 50 MHz.

## Architectural Constraints

- ✓ Click on the Architectural constraints icon and completely unroll the MAC loop by checking the “unroll” box..

## Schedule

- ✓ Schedule the design and select the table view in the design center. Note that even though all the loop iterations are scheduled in parallel by completely unrolling the MAC loop, the design still has a throughput of 5 clock cycles (Figure 14).



Solution	Status	Total Area Score	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time
Solution_1	Passed extract	3210.57	5	100.0	5	100.0

Figure 14 Throughput in Table View

- ✓ Click on the Schedule icon to look at the Gantt chart. Note that although all the multiplies and adds are scheduled in parallel, it takes 5 clock cycles (or C-steps) to produce one output for each iteration of the main loop. To decrease the throughput we want to be able to start a new iteration of the main loop before the current iteration has finished. We can do this by pipelining the main loop.
- ✓ Goto architectural constraints and click on the main loop.
- ✓ Check the pipelining box and set the “initiation interval” to 2.
- ✓ Reschedule the design and check the throughput. It should now be equal to 2.
- ✓ Goto architectural constraints and click on the main loop.
- ✓ Check the pipelining box and set the “initiation interval” to 1.
- ✓ Reschedule the design and check the throughput. It should now be equal to 1.

Note that in a typical design adding more pipelining will make the design larger.

- ✓ Close all windows and select File->new project

## Debugging Pipelining Dependency Chains

### Create a New Project

- ✓ Goto the File menu and select “New Project”.

### Add the input file

- ✓ Add the pipeline.cpp file to the Catapult project.

### Setup Design

- ✓ Click on the setup design icon in the design bar
- ✓ Set the technology to Xilinx VIRTEX-E(Default part and speed grade).

- ✓ Set the clock frequency to **33 MHz**.

## Schedule

Click on schedule and check the latency.

## Pipeline the Design

- ✓ Goto architectural constraints and pipeline the main loop with an initiation interval equal to one.
- ✓ Reschedule the design.

The design fails to schedule because it cannot meet the constraint.

- ✓ Look at the transcript to see the message. It indicates that there is a feedback data dependency(Figure 15).

Click here to see data dependency in code

Figure 16 Transcript Error Message

Click here to see description of error

- ✓ Trace to the source code by clicking on the error message (Figure 15). This shows that “a[n+1] is dependent on “a[n]”.

To view the data dependency in the Gantt chart:

- ✓ reactivate the first solution by double clicking on it in the design center.
- ✓ Click on schedule and then right-click and select “View filtered -> Show All Ops”(Figure 17).

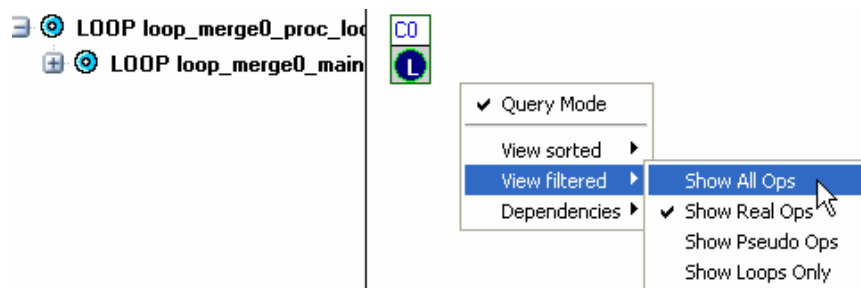


Figure 17

- ✓ Expand LOOP0

- ✓ Search in the schedule of the first solution for the operator listed in the error message, in this case “mux\_n32\_l28”. This is done using the “Find” icon in the design bar. You will see the feedback data dependency by tracing the assignments driving the mux (Figure 18).

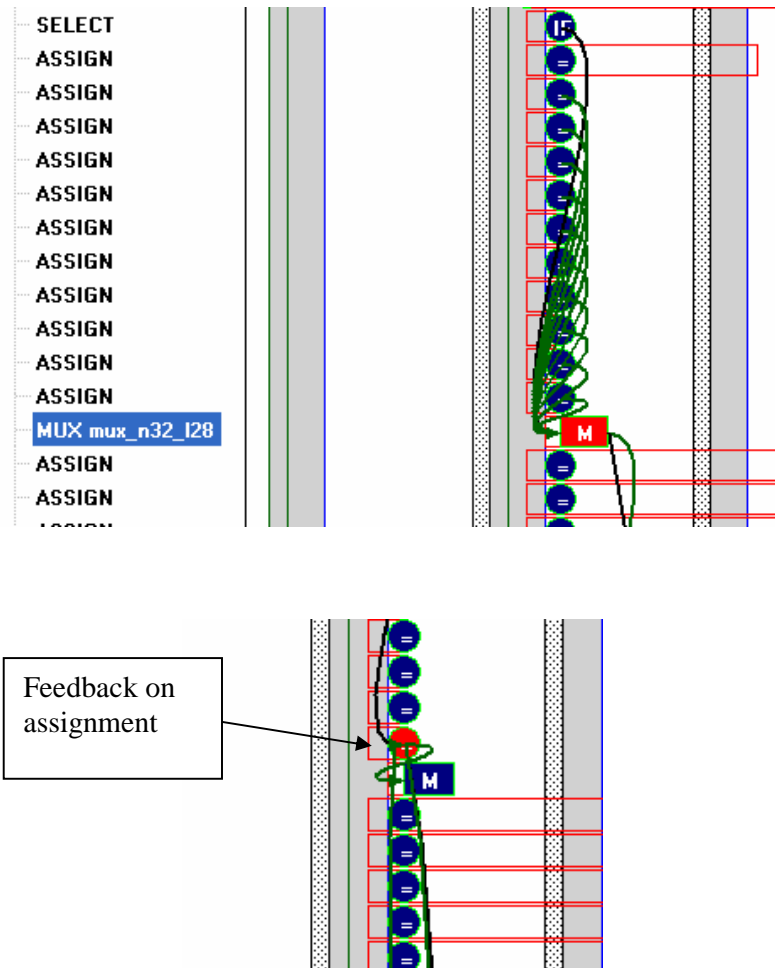


Figure 18 Feedback Dependency Chain

- ✓ Try cross probing from the dependency chain operators to the source code.

From the Gantt chart we can see that the assignment to  $a[n+1]$  from  $a[n]$  takes two clock cycles. Thus it is not possible to start a new iteration every clock cycle (Pipelining with  $II=1$ ). To pipeline this design you can try the following :

- ✓ Try Pipelining with initiation interval ( $II$ ) set equal to 2.
- ✓ Use a slower clock frequency so the dependency chain is scheduled in the same clock cycle and pipeline with  $II=1$
- ✓ Use a faster speed grade device so the dependency chain is scheduled in the same clock cycle.

END