INTRODUCTION TO CATAPULT C

Vijay Madisetti, Mohanned Sinnokrot
Georgia Institute of Technology
School of Electrical and Computer Engineering

with adaptations and updates by:

Dongwook Lee, Andreas Gerstlauer
University of Texas at Austin
Department of Electrical and Computer Engineering

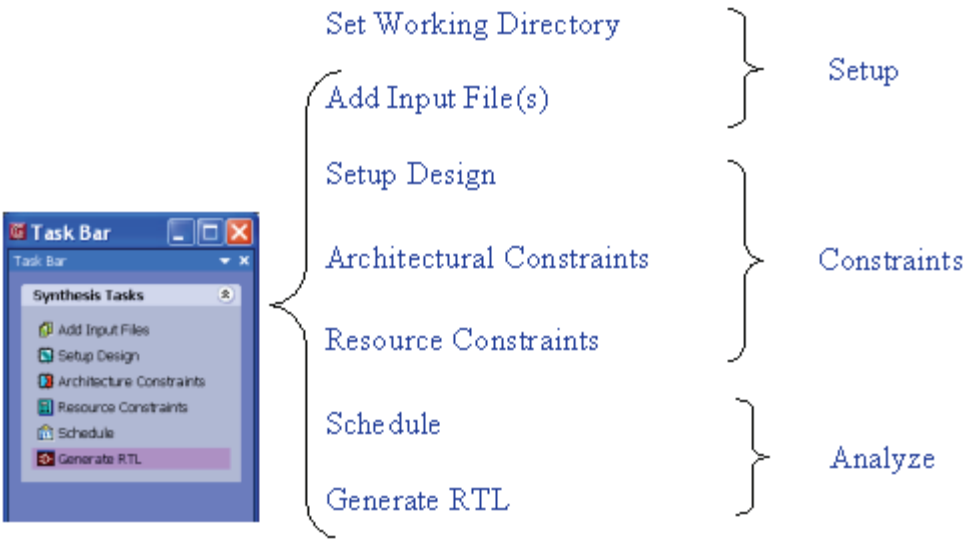
1 Introduction

The purpose of this tutorial is to introduce Catapult C, the design and synthesis tool that will be used throughout the course. We will walk you through an example that shows how to setup a design, impose architectural and resource constraints, generate synthesizable register transfer language (RTL) and analyze the generated design.

Note: On the UT Austin campus, the Catapult software by Mentor Graphics is installed on the ECE LRC servers. We will use Catapult together with the logic synthesis tool Precision (also by Mentor Graphics). To access the software, first setup the environment by loading the corresponding modules and then start the Catapult University Version software:

```
% module load mentor/precision
% module load mentor/catapult
% catapult
```

2 The Catapult C Workflow



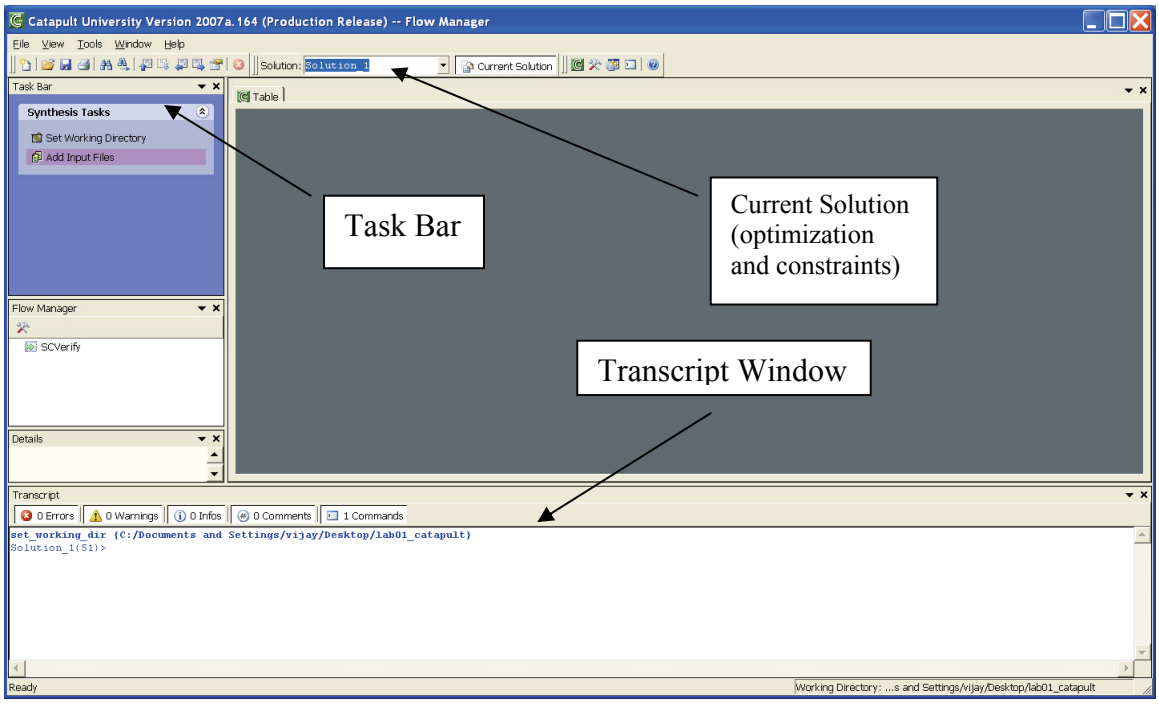
The Catapult C workflow consists of three steps:

- Setup: involves setting up the working directory or project directory and adding the project files.

- Constraints: involves imposing hardware constraints, architectural constraints and resource constraints
- Analysis: involves analyzing the design for timing, area and throughput to ensure that the implementation meets the design specifications

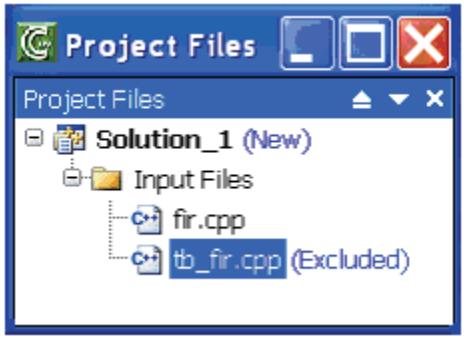
We will walk through a simple example to show how we can setup a design, constraint it and analyze it.

2.1 Design Setup



- Invoke Catapult C University Version
- Click on "`Set Working Directory`" on the "`Task Bar`" and choose the "`fixed`" set of files that come with this tutorial
  - You should see a message that confirms that the working directory has been set in the "`Transcript Window`"
- Click on "`Add Input Files`" on the "`Task Bar`" and select the file "`fir.cpp`"
  - You should see a message that confirms that one file has been added and you should see that a new command "`Setup Design`" appears on the "`Task Bar`"
- Click on "`Add Input Files`" on the "`Task Bar`" and select the file "`tb_fir.cpp`".
- Right click on the "`tb_fir.cpp`" file and select the option "Exclude from Compilation". This is a test bench file that will be used to verify the functional flow of the synthesized RTL. Therefore, the file "`tb_fir.cpp`" is not part of

the design and is simply a test bench file. We will cover the verification flow in a separate section.



Setting up a design in Catapult C is very similar to project setup in IDEs such as Microsoft Visual Studio, where a project is created and then files are added to the project. We next discuss the design under consideration (`fir.cpp`), which is an implementation of a finite impulse response filter (FIR). The FIR filter contains the basic building blocks of any DSP system: adders, multipliers and delay lines. Although we will not discuss the implementation in detail, it is important to understand the design under consideration in rather general terms.

```
 1: //
 2: // Catapult Design top (FIR filter)
 3: //
 4: #include "fir_inc.h"
 5:
 6: #pragma hls_design top
 7: void  fir(fxc_t coeffs[TAPS]
 8:      ,fx_t *in1
 9:      ,fx_t *out1
10:      )
11: {
12:     // Register and Accumulator Declaration
13:     static fx_t regs[TAPS];
14:     ac_fixed<G+2*W,G+1,true,AC_TRN,AC_WRAP> acc=0.0;
15:
16:     // Shift Register or Tap-Delay Line Implementation
17:     SHIFT:
18:     for(int i = TAPS-1; i>0; i--)
19:         {
20:         regs[i] = regs[i-1] ;
21:         }
22:     regs[0] = *in1 ;
23:
24:     // Multiply and Accumulate
25:     MAC:
26:     for(int i = 0 ; i < TAPS ; i++)
27:         {
28:         acc += regs[i] * coeffs[i];
29:         }
30:
31:     *out1 = acc;
32: }
```

- The filter coefficients are `coeffs`, the input data to be filtered is `in1` and the filtered output data is `out1`. Mathematically, the output of the filter is given by:
  $out1[n] = \sum_{k=0}^{TAPS-1} coeffs[k] \times in1[n-k]$
- The `SHIFT` loop (lines 17 through 22) is an implementation of the tap delay line or equivalently, $in1[n-k]$
- The `MAC` loop (lines 25 through 29) is an implementation of the multiply and accumulate unit, or equivalently $\sum_{k=0}^{TAPS-1} coeffs[k] \times in1[n-k]$

We will not concern ourselves with the input and output data types, bit widths or fixed-point arithmetic issues. The code already uses existing fixed-point arithmetic and data type libraries provided as part of Catapult C. Our main goal is to recognize two loops in the design: the `SHIFT` loop, which is an implementation of a tap delay line and the `MAC` loop, which is an implementation of multiplication and addition. We next discuss how to impose design constraints.

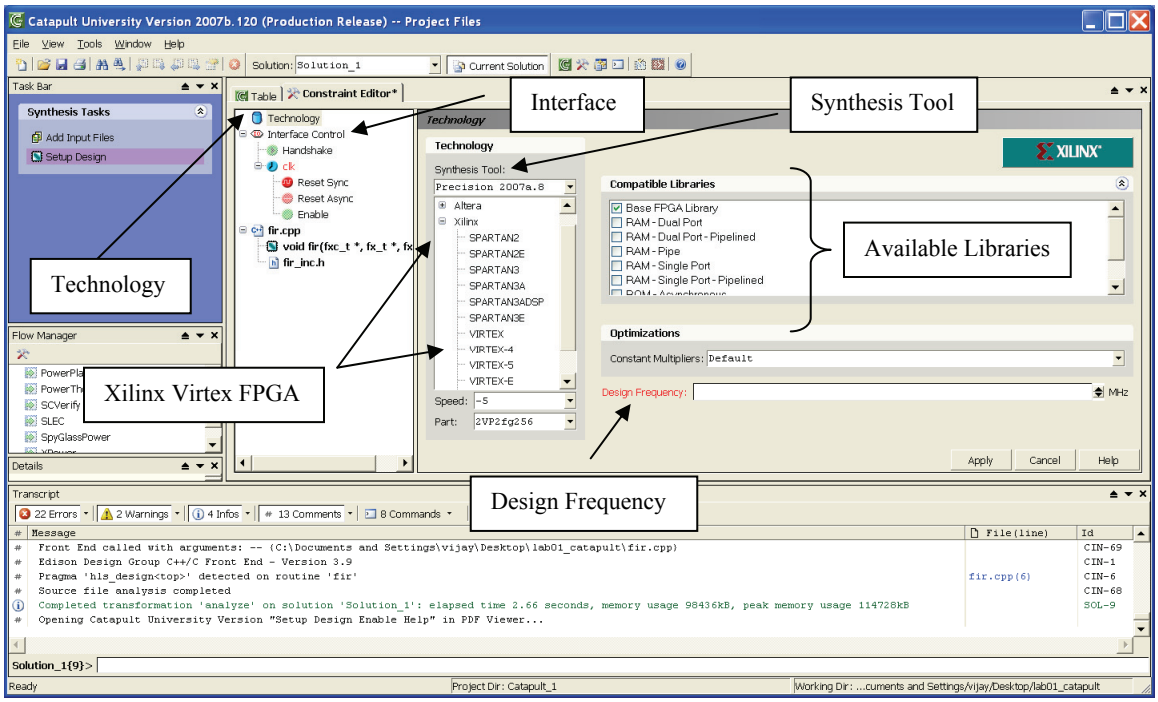
2.2 Imposing Constraints

Catapult C allows us to impose different constraints on the design that include hardware constraints, architectural constraints and resource constraints. We begin with the hardware constraints.

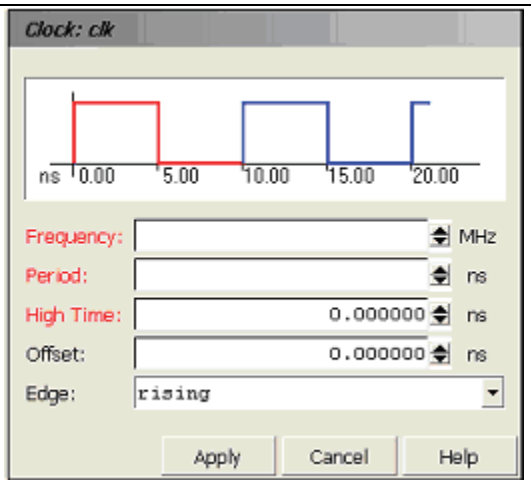
2.2.1 Hardware Constraints

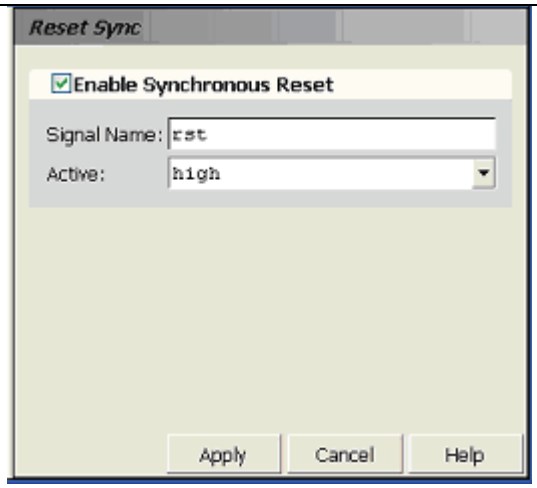
Hardware constraints allow us to control the technology or basic libraries to be used in the design, the interface of the design and the clock speed. The technology is made up of the basic library and custom IP blocks that we can make available to the tool.

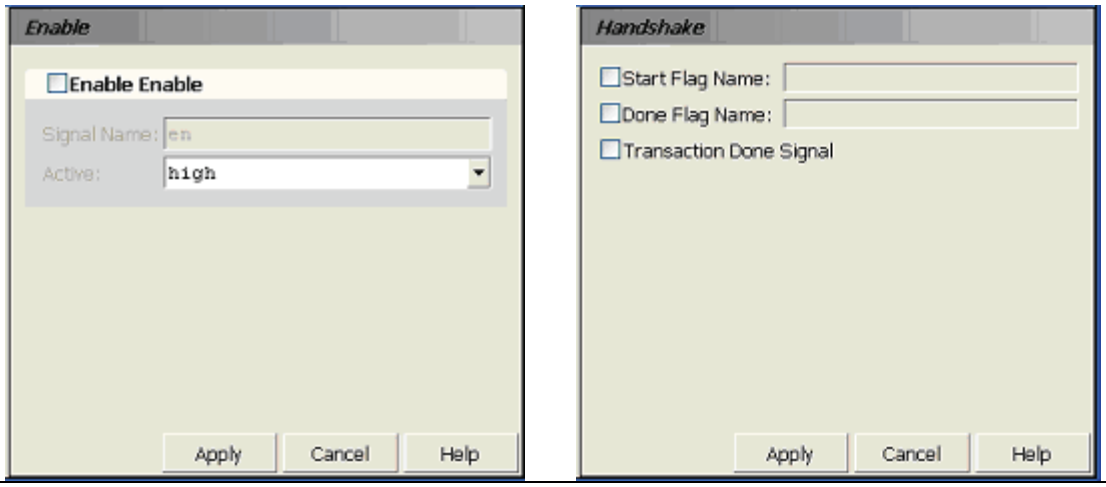
- Click on "`Setup Design`" on the "`Task Bar`". A "`constraint window`" should appear that gives you access to different libraries that are available in the design
- Choose "`Precision 2009a_up`" in the "`Synthesis Tool`" drop down menu. This gives us access to different FPGA libraries.
- Set the technology to Xilinx Virtex II-Pro (Speed grade: -5)
- Set to the clock frequency to 100 MHz
- Click the "`Apply`" button



Using this setup, we are basically targeting a Xilinx Virtex II-Pro FPGA board with a clock frequency of 100 MHz for synthesis. The next aspect of hardware constraints is the "`Interface Control`". Although we will not change the default settings of the "`Interface Control`", we will briefly examine the interface options.



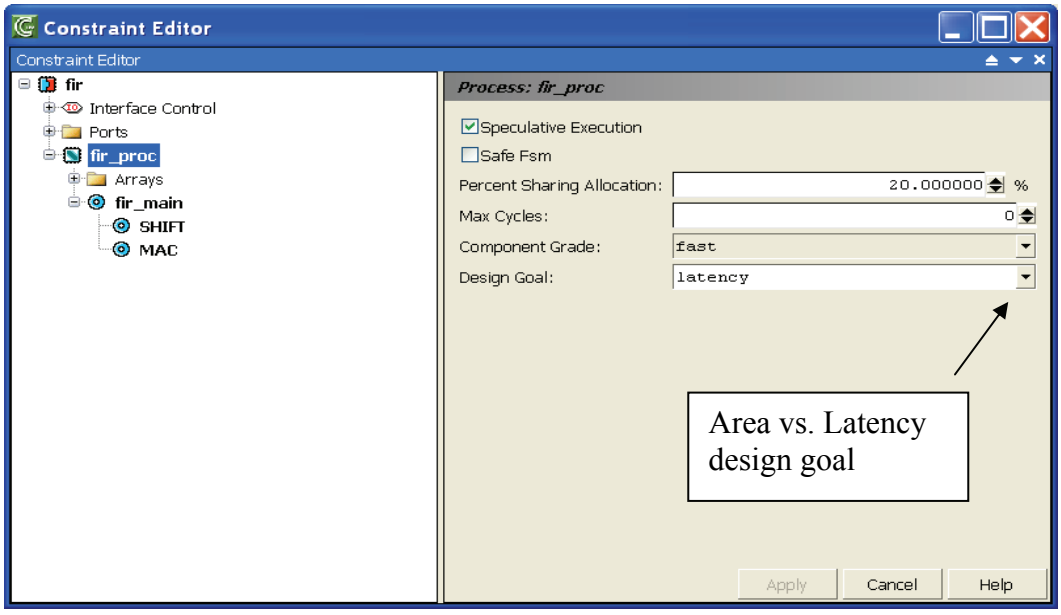




- *Clock*: the signal name (default `clk`) and the active edge for the internal registers can also be changed (default `rising` edge)
- *Reset*: you can enable synchronous, asynchronous or both types of reset. By default, the design has synchronous reset and reset is active high.
- *Enable*: you can add an enable signal to the design that allows the design to hold its current state, which is a useful option that allows us to put the hardware in a low power mode. By default, the design does not have an enable signal.
- *Handshake*: a design with a start handshake waits for the start signal to be asserted before reading all the non-pointer inputs. A design with a done handshake sets the done signal to high in the same cycle that the output value is written. By default, handshaking is not enabled.

2.2.2 Architectural Constraints

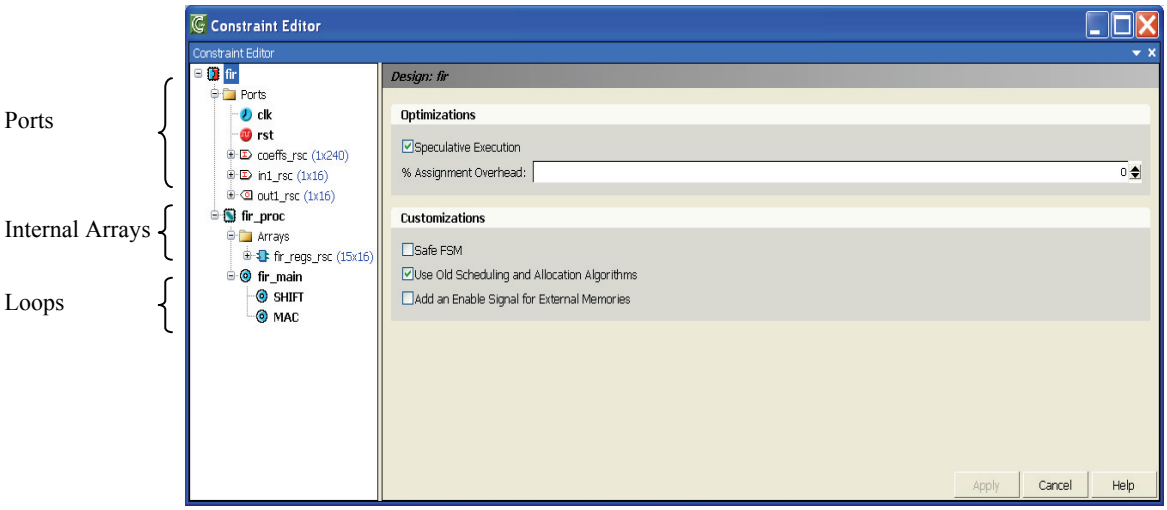
The architectural constraints refer to imposing constraints or optimizations on the ports, arrays and loops in the design. Also, the overall design goal of small area or low latency is specified in the architectural constraints editor.



The design goal of area versus latency optimization can be specified as follows:

- Click on the "`Architecture Constraints`" on the "`Task Bar`".
- Select "Area" from the pull-down menu. We will return to the "Latency" optimization later in the tutorial.

Since some of the ports in the design are not specified in the C++ code, they can be viewed in the architectural constraint window.



- Click on the "`Architecture Constraints`" on the "`Task Bar`".
- Observe that in addition to the two input ports (`coeffs_rsc` and `in1_rsc`) and one output port (`out1_rsc`), the `clk` and `rst` ports have been added automatically in the design. To see the correspondence between the ports and the variables defined in the design, double click on a port name to cross-probe, which will open up the `fir.cpp` file and show the corresponding variable.
- Also observe the internal array declaration `fir_regs`, which corresponds to the `regs` declaration in the `fir.cpp` file.
- The last part of the architecture constraint window allows us to perform loop optimizations. The two loop optimizations that can be performed are loop unrolling and loop pipelining.
- Click on the `SHIFT` loop. Note that the SHIFT loop is not unrolled nor pipelined. The same is true for the `MAC` loop. We will return to these two optimizations when we compare different architectures.

We will not cover the more advanced resource constraints, which are used to fine tune the hardware architecture. The hardware and architecture constraints allow us to obtain efficient designs without requiring advanced fine tuning techniques for the design.

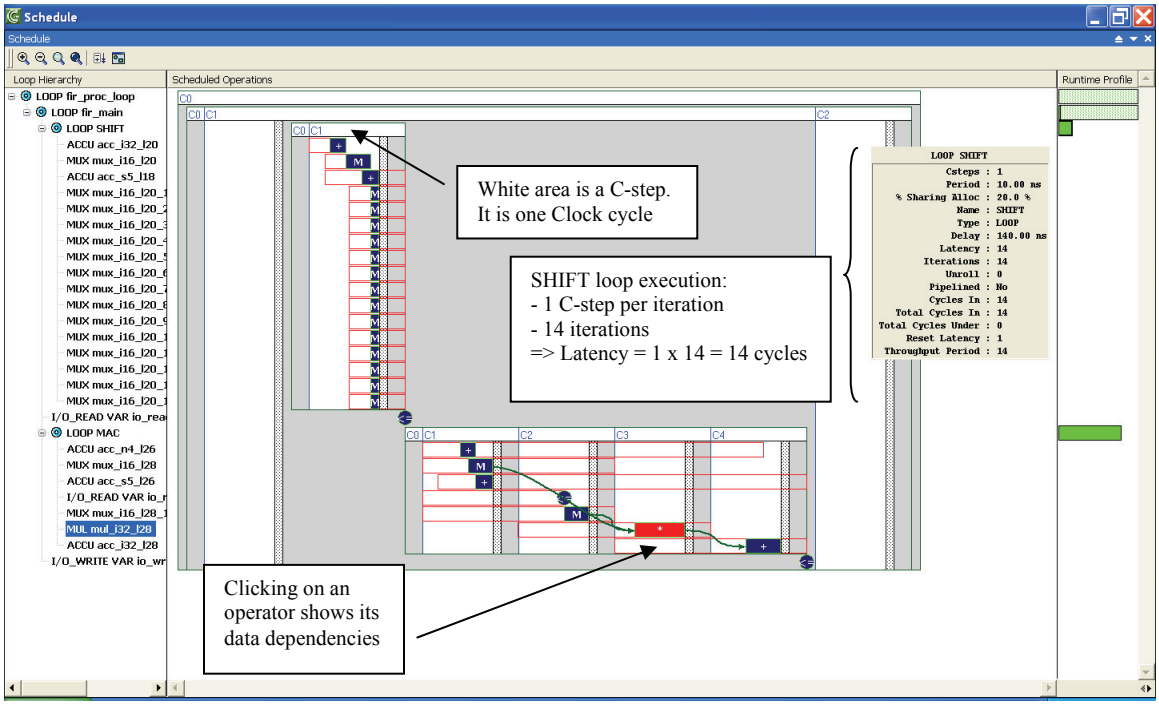
2.3 Design Analysis

Design analysis refers to design scheduling and RTL generation, which will be discussed next.

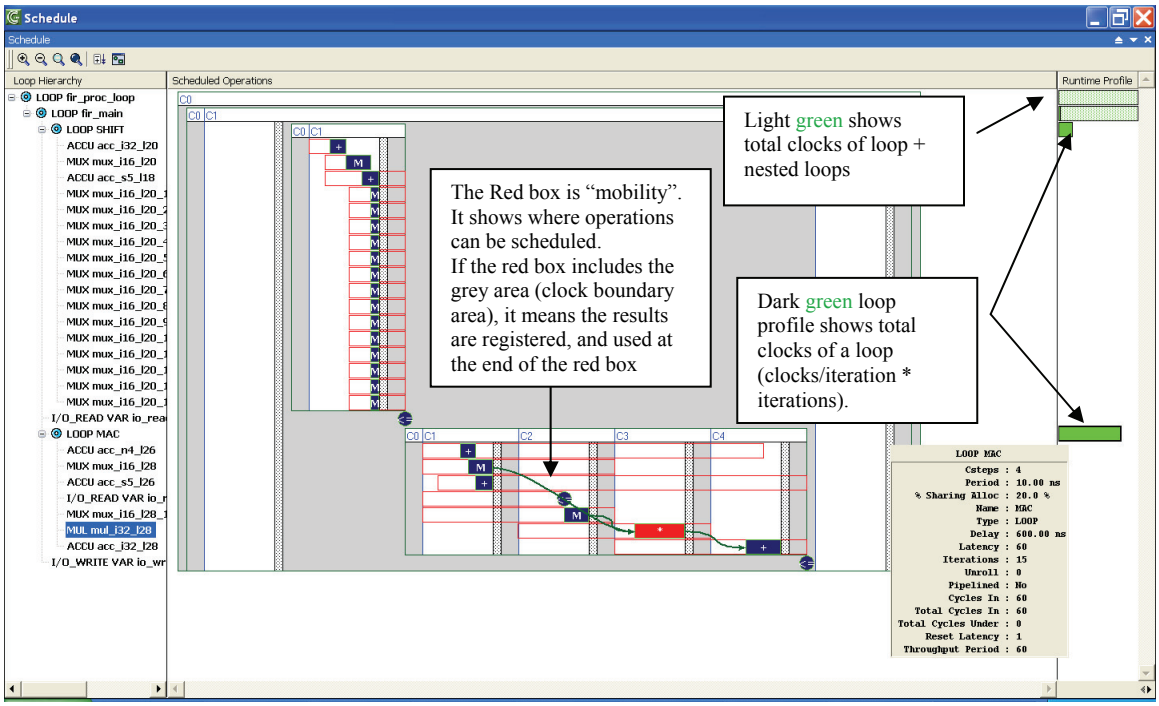
2.3.1 Design Scheduling

Design scheduling refers to the placement of hardware resources (multipliers, adders, etc.) into clock cycles, which determines the latency and throughput of the design. Catapult C gives a graphical description of the design in the form of a Gantt chart.

- Click on the "`Schedule`" on the "`Task Bar`".



- Note that the execution of the each iteration of the `SHIFT` loop occurs in 1 clock cycle (1 C-step). Since there are a total of 14 iterations, the latency for the `SHIFT` loop is 1×14 = 14 cycles.
- You can view data dependencies and cross-probe an operator in the Gantt chart.

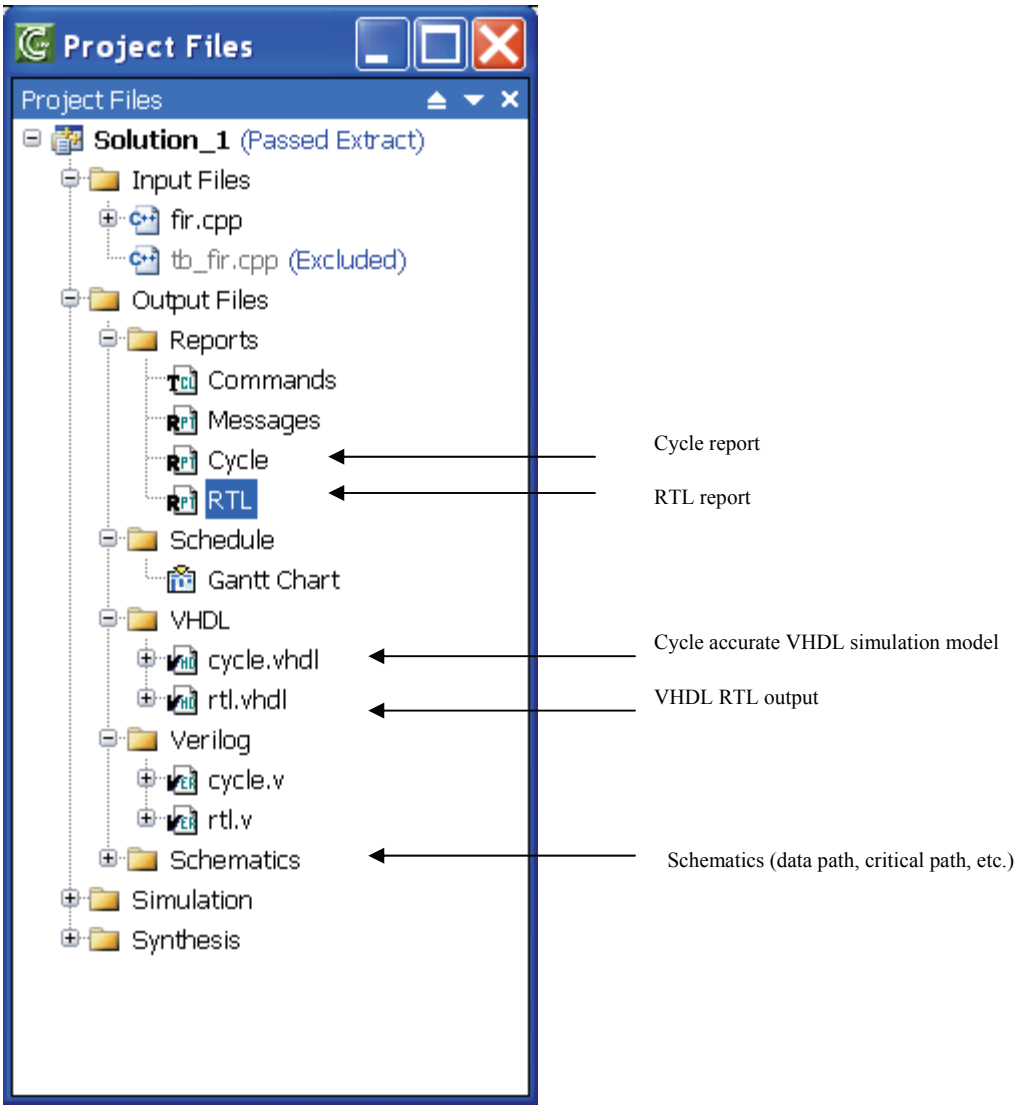


- The execution profile of the SHIFT and MAC loop is shown on the right in dark green. Move the mouse pointer over the dark green area and it will show the execution profile. For the MAC loop, executing one iteration takes 3 C-steps, and the total latency is 45 cycles since there are 15 iterations.
- The current constraints and optimization is "`Solution_1`". In this solution, we are not performing loop unrolling or loop pipelining. We will compare different solutions that have different optimizations later in the tutorial.

2.4 RTL Generation

The last step is the RTL generation, which generates synthesizable RTL in Verilog, VHDL and SystemC along with other reports, schematics and charts. Area estimates that are generated are fairly accurate.

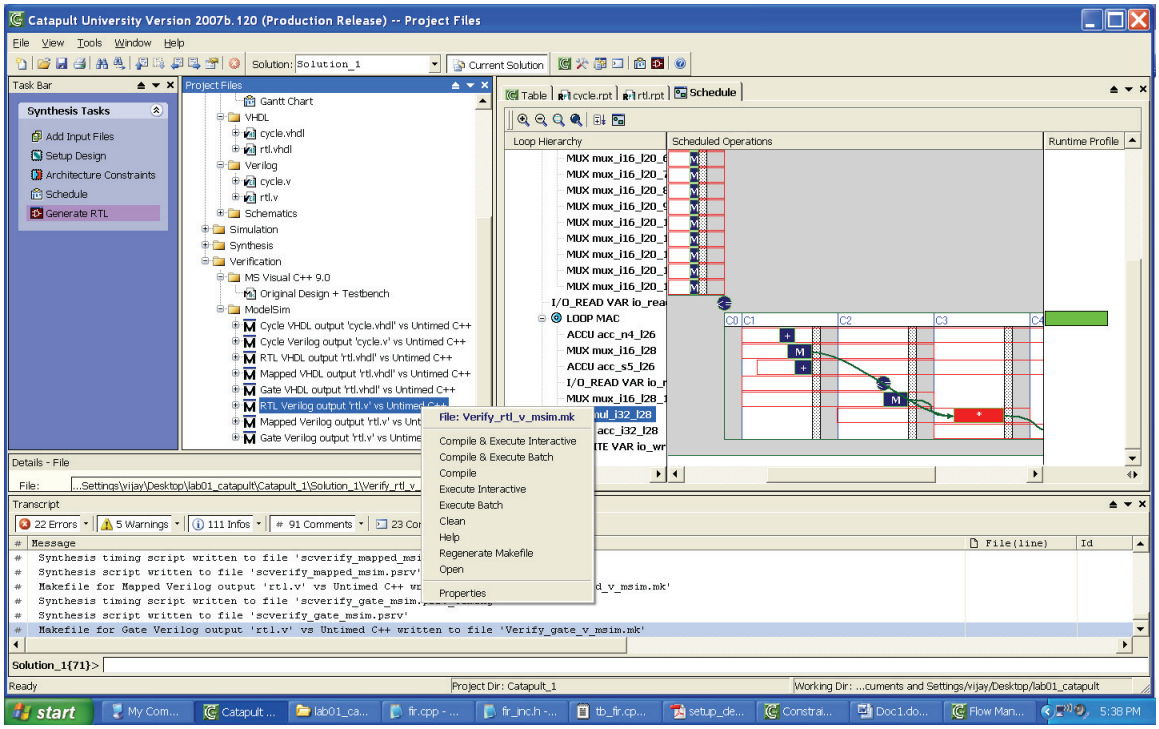
- Click on the "`Generate RTL`" on the "`Task Bar`".
- For every solution, Catapult C will generate two reports automatically, which are the Cycle report and RTL report
    - The Cycle report (`cycle.rpt`) is a high-level algorithm report that includes information on loop iterations, latency and throughput. Open the `cycle.rpt` file by double clicking on it and view the Loop Execution Profile. Note that the `SHIFT` and `MAC` loop latencies are 14 and 45, respectively, as expected from the Gantt chart analysis. The total latency is 61 cycles with two additional read and write cycles.
    - The RTL report (`rtl.rpt`) gives an estimate of the allowable slack in the clock period and area. Area estimates are usually within 20% of the area reported by RTL synthesis. Open the `rtl.rpt` and view the Timing Report. Note that the maximum delay is 7.46 ns while the slack is 2.54 ns, indicating that we have met the timing requirements of 100 MHz clock speed (or 10 ns clock period). Notice that the area score in the RTL report is 2061.50 LUTs (Look-up Tables).



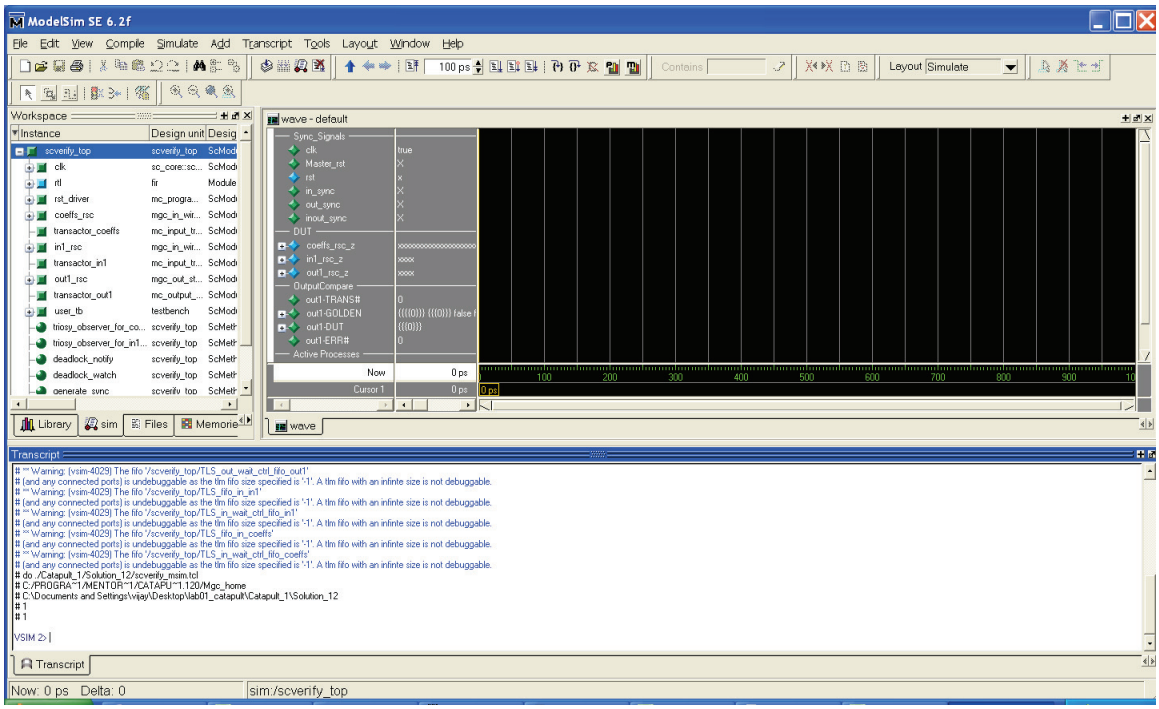
2.5 RTL Verification

The RTL verification flow tests that the generated RTL matches the original C model, based upon the C++ test bench that runs the C++ model. Also, RTL verification will verify the Catapult reported throughput and latency. At this stage, we can that initial testing and verification has been performed at the C-level and this verification flow verifies that the generated RTL behaves as the C-model. At this stage, we will not verify the functional behavior of the block, rather, we will verify that the design under consideration gives the expected latency and throughput. Because the input is the unit impulse, the output is simply the filter coefficients. Therefore, we will also verify that the output is the same as the filter coefficients. Right click on `SCVerify` in the "`Flow Manager`" and select Enable

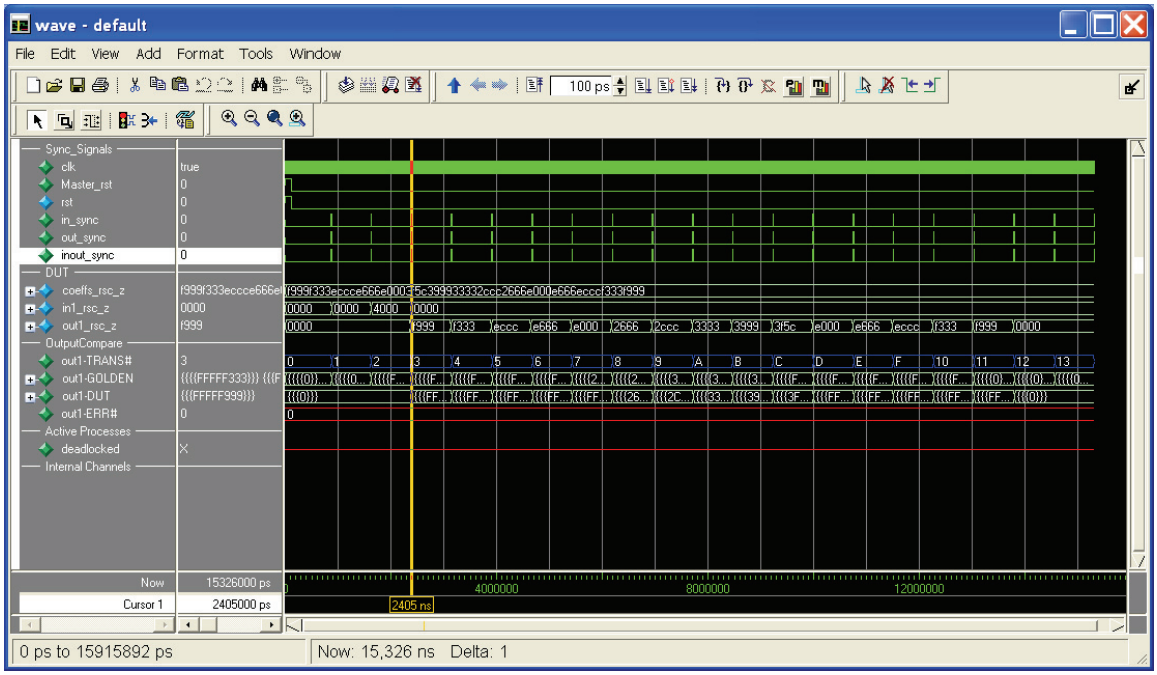
- A new directory "`Verification`" appears under "`Project Files`". Expand the `Verification` directory, then the `ModelSim` directory



- Right click on "`RTL Verilog output 'rtl.v' vs Untimed C++`" and choose "`Compile & Execute Interactive`"
- Compilation might take a few minutes (about 10 minutes). The `ModelSim` program will be invoked, which is used to verify the functional behavior of the generated Verilog RTL



- From the menu, select `Simulate` => `Run` => `Run All`
- If you right click on the waveform viewer, different zooming options appear. Choose "`zoom full`", then you can zoom into different places of the waveform



- Verify that the output of the block is simply the filter coefficients
- Verify that the latency between each two output samples is 61 cycles

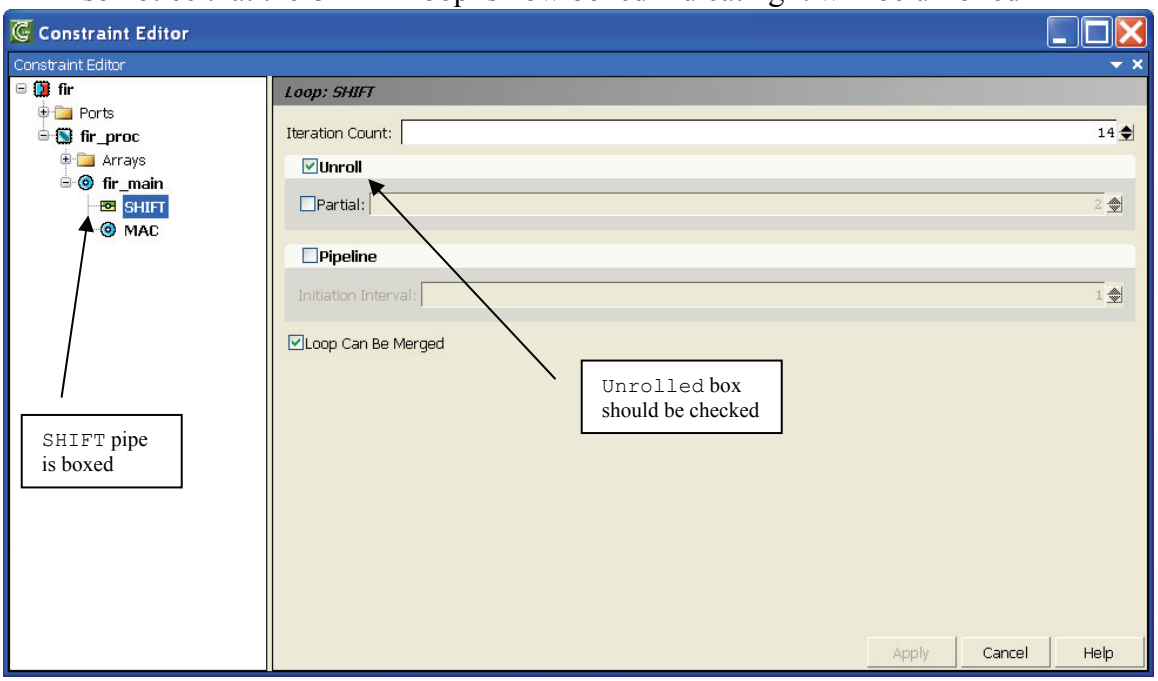
3 Warm-up

In this section, we will perform different optimizations and compare the different solutions in terms of area, latency, throughput and maximum delay. The two optimizations that will be performed are loop unrolling and loop pipelining.

3.1 Loop Unrolling

When a loop is unrolled, all the iterations of the loop will occur in parallel. Loop unrolling not only decreases latency, but in certain cases, it also reduces hardware size since control logic for the loop iterations is eliminated. We will next examine the effect of loop unrolling on the design.

- Click on "`Architecture Constraints`" on the "`Task Bar`" and Select the "`Constraint Editor`" tab.
- Click on the "`SHIFT`" loop under "`fir_main`"
- Check the "`Unroll`" box. Notice that Catapult C generates a new solution (`Solution_2`), which gives us the ability to compare with any previous solutions. In this case, there is one previous solution, which is `Solution_1`. Also notice that the `SHIFT` loop is now boxed indicating it will be unrolled

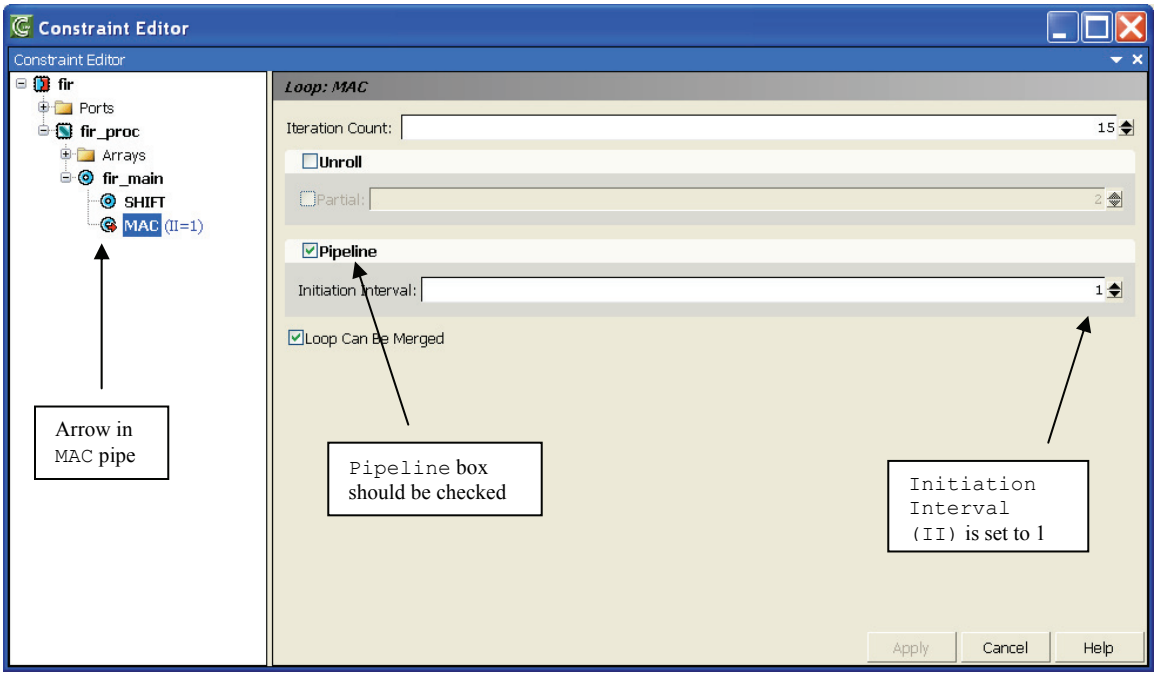


- Click the "`Apply`" button.
- Click on "`Schedule`" and view the Gantt chart of `Solution_2` and notice the `SHIFT` loop has been optimized out since all the iterations are now done in parallel in one cycle.
- Click on "`Generate RTL`" and view the Loop Execution Profile in `cycle.rpt`. Notice that the `SHIFT` loop latency has been eliminated and the total latency is 45 + 2 = 47. View the Timing Report in `rtl.rpt` and notice that the maximum delay has decreased to 7.46 ns from 5.48 ns. Note that the area has also decreased from 2061.50 LUTs to 1177.65 LUTs.

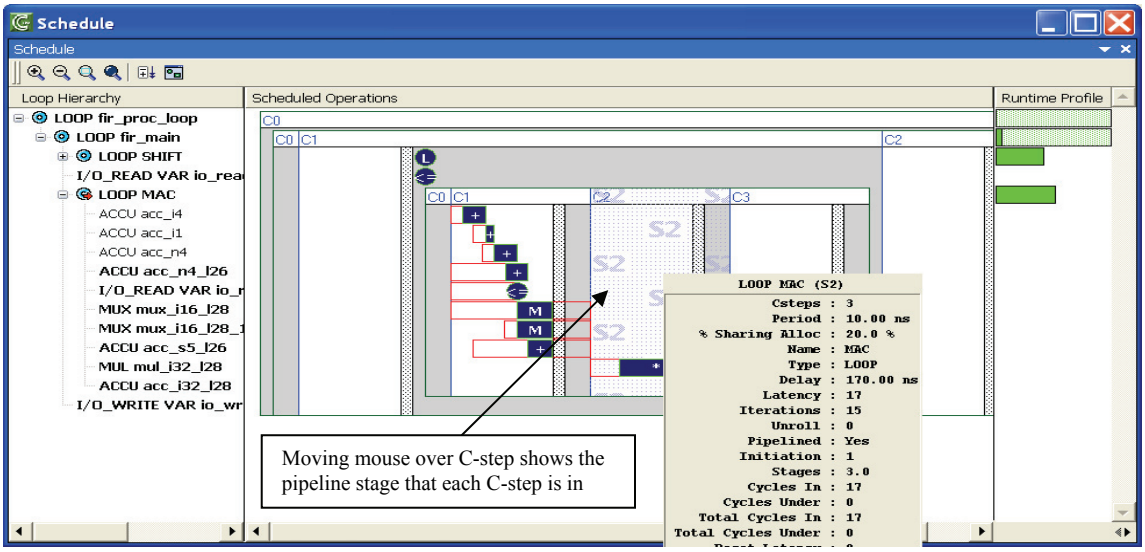
3.2 Loop Pipelining

When a loop is pipelined, the second iteration of the loop executes *x* cycles after the first iteration and so on. In Catapult C, the value x is known as the `Initiation Interval (II)`. For example, if `II = 1`, this means that the second iteration will execute one clock cycle after the first iteration, second iteration after the third and so on. Since we want to examine the effect of loop pipelining on the design and not loop unrolling, we will disable loop unrolling for the `SHIFT` loop and enable loop pipelining for the `MAC` loop.

- Click on "`Architecture Constraints`" on the "`Task Bar`" and Select the "`Constraint Editor`" tab.
- Click on the "`SHIFT`" loop under "`fir_main`" and make sure that the "`Unroll`" box is unchecked. The SHIFT loop should return to a circle or pipe when you uncheck the "`Unroll`" box
- Click on the "`MAC`" loop under "`fir_main`" and check the "`Pipeline`" box Notice that Catapult C generates a new solution (`Solution_3`), which again gives us the ability to compare with the previous two solutions. Also notice that the `MAC` loop has an arrow in the pipe indicating that the loop will be pipelined



- Click the "`Apply`" button
- Click on "`Schedule`" and view the Gantt chart of `Solution_3` and notice that when you move the mouse pointer over a C-step, it shows the pipeline stage that each C-step is in



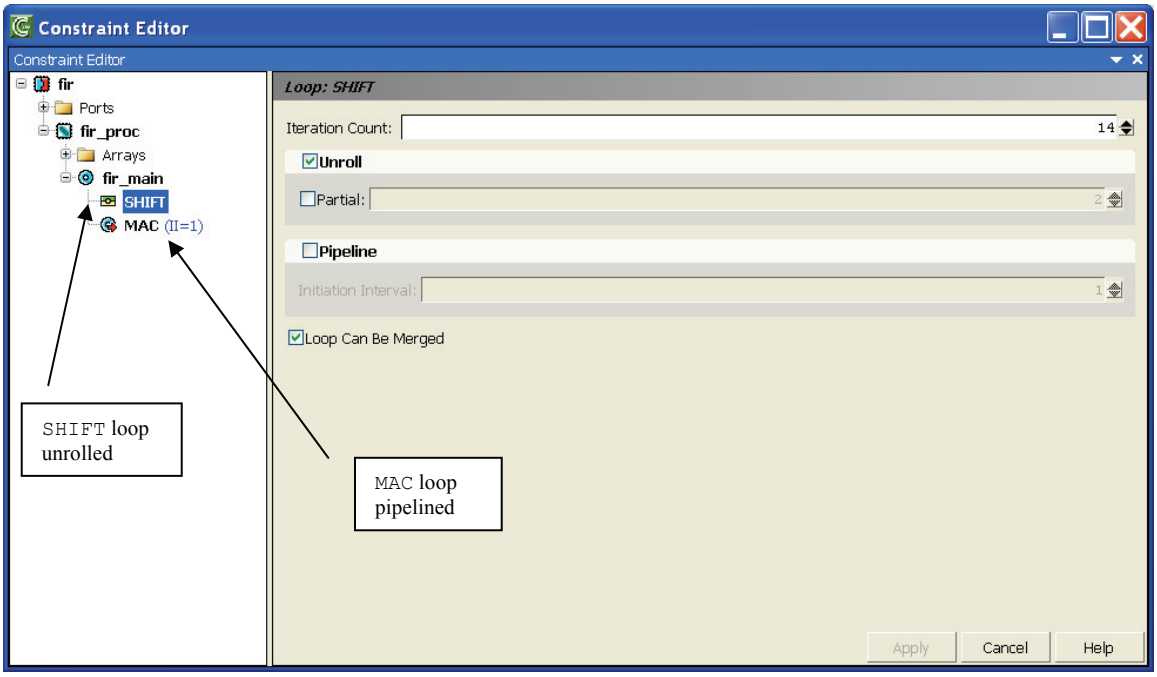
- Click on "`Generate RTL`" and view the Loop Execution Profile in `cycle.rpt`. Notice that the `MAC` loop latency has decreased from 45 to 17. Since there are 15 iterations, each iteration is now executing in one cycle and there is a two-cycle latency or overhead in the loop execution. View the Timing Report in `rtl.rpt` and notice that the maximum delay has is 7.46 ns, which indicates that the critical path is in the `SHIFT` loop since it is the same maximum delay as `Solution_1`, but greater than `Solution_2`. Finally, there is a marginal decrease in area from 2061.60 LUTs (`Solution_1`) to 2060.58 LUTs.

4 Exercises

In this section, we will examine a few more solutions and compare them in terms of area, latency, throughput and maximum delay. We will simply compare the different solutions as shown in the Table I at the end of the tutorial.

In `Solution_4`, we want to combine both loop unrolling and loop pipelining.

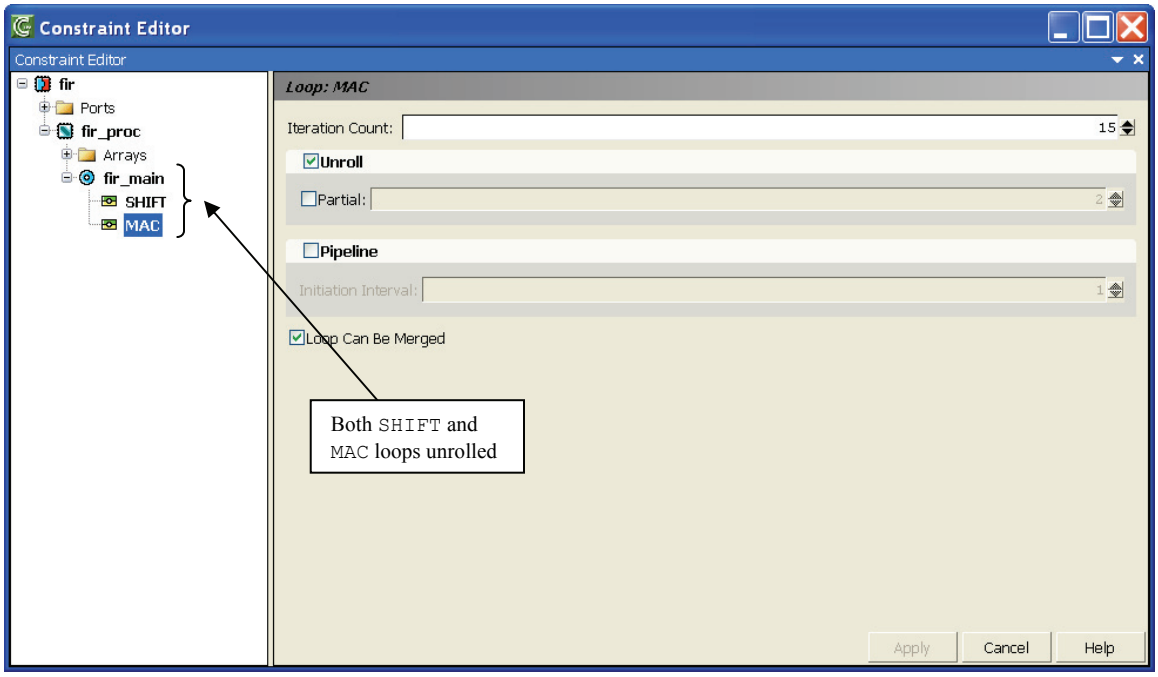
- Click on "`Architecture Constraints`" on the "`Task Bar`" and Select the "`Constraint Editor`" tab.
- If starting from `Solution_1`, then click on the "`SHIFT`" loop under "`fir_main`" and check the "`Unroll`" box. Click on the "`MAC`" loop under "`fir_main`" and check the "`Pipeline`" box. Notice that Catapult C generates a new solution (`Solution_4`). If starting from `Solution_2`, you only have to pipeline the `MAC` loop and if starting from `Solution_3`, you only have to unroll the `SHIFT` loop. Notice that the `SHIFT` pipe is now boxed and the `MAC` pipe has an arrow



- Click on "Schedule" and view the Gantt chart of Solution_4
- Click on "Generate RTL" and view the generated cycle.rpt and rtl.rpt reports and determine the area score, latency and maximum delay

In Solution_5, we want to unroll both the SHIFT and MAC loops.

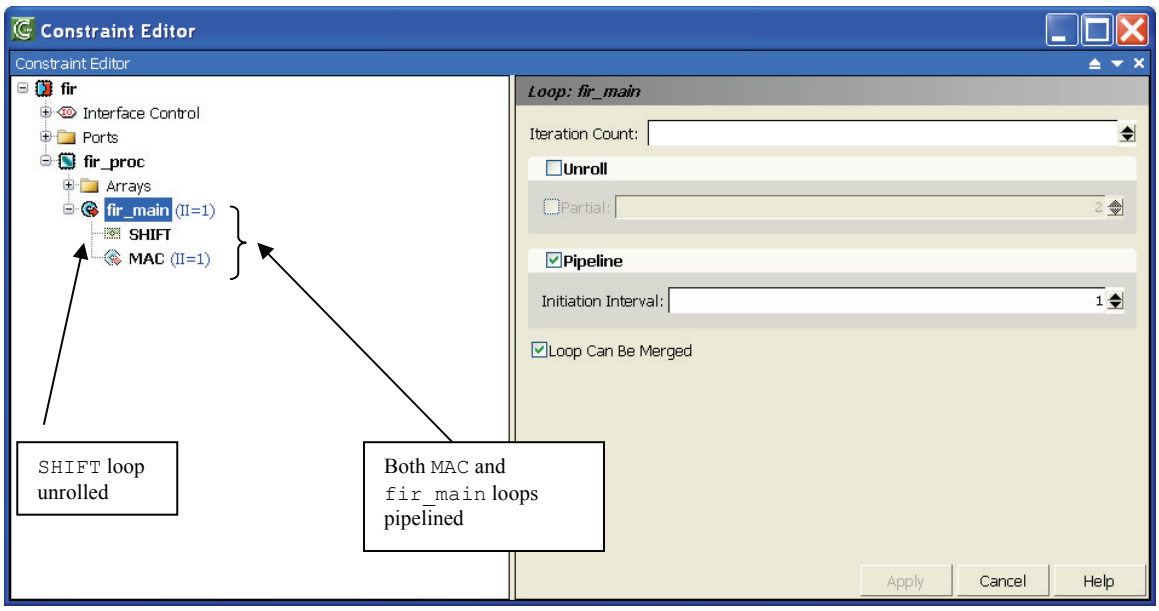
- Click on "Architecture Constraints" on the "Task Bar" and Select the "Constraint Editor" tab.
- If starting from Solution_1, then click on the "SHIFT" loop under "fir_main" and check the "Unroll" box. Click on the "MAC" loop under "fir_main" and check the "Unroll" box. Notice that Catapult C generates a new solution (Solution_5). If starting from any other solution you simply need to unroll both the MAC and SHIFT loops. Notice that the SHIFT and MAC pipes are now boxed



- Click on "Schedule" and view the Gantt chart of Solution_4
- Click on "Generate RTL" and view the generated cycle.rpt and rtl.rpt reports and determine the area score, latency and maximum delay

In Solution_6, we want to unroll the SHIFT loop and pipeline that MAC and fir_main loops.

- Click on "Architecture Constraints" on the "Task Bar" and Select the "Constraint Editor" tab.
- Starting from Solution_4, click on the "fir_main" loop and check the "Pipeline" box. Notice that Catapult C generates a new solution (Solution_6).



Up until now, we considered optimization for area rather than latency. Consider the design optimization for latency. Repeat the configurations of `Solution_1` through `Solution_6` that are optimized for latency, which would give `Solution_7` through `Solution_12`.

Finally, consider `Solution_13`, in which the design goal is latency, both the `MAC` and `SHIFT` loops are unrolled and the `fir_main` loop is pipelined with initiation interval equal to one (II = 1).

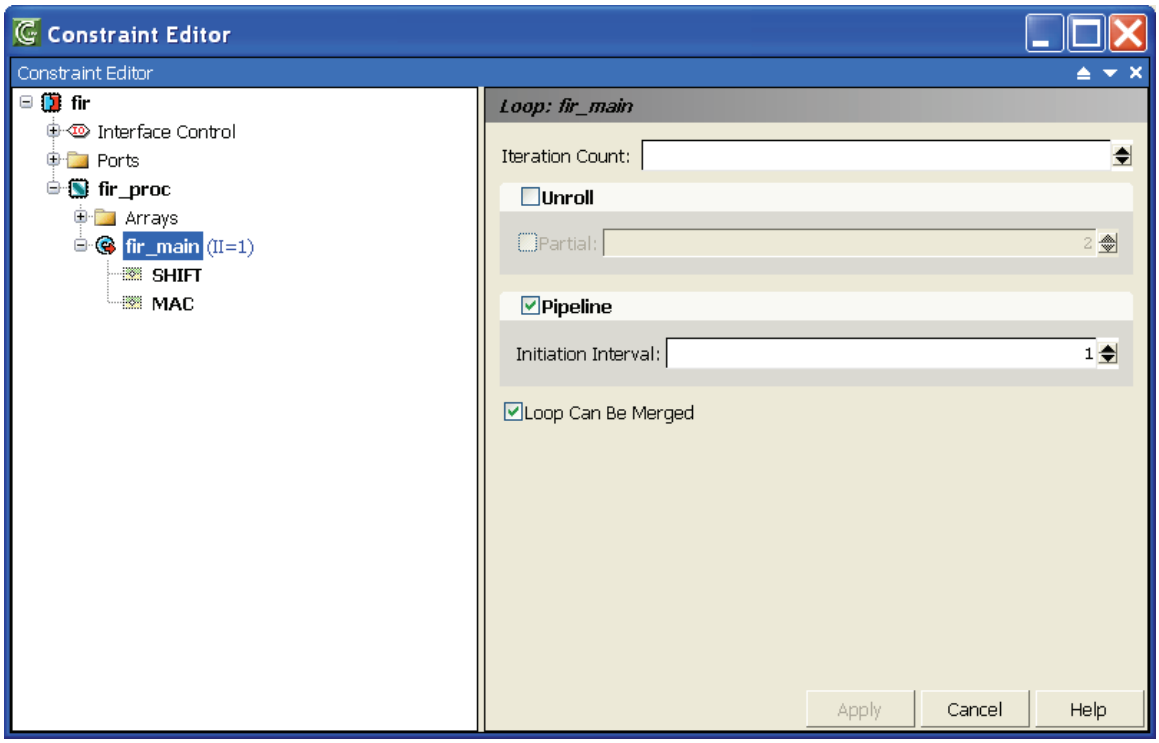


Table I. Comparison of different solutions based on loop unrolling, loop pipelining and area versus latency optimization.

Solution	Optimization (Design Goal)	Area Score	Latency Cycles / Throughput Cycles	Maximum Delay (ns)
`Solution_1`	Area	2061.50	61/61	7.46
`Solution_2`	Area	1177.65	47/47	5.47
`Solution_3`	Area	2060.58	33/33	7.46
`Solution_4`	Area			
`Solution_5`	Area			
`Solution_6`	Area			
`Solution_7`	Latency			
`Solution_8`	Latency			
`Solution_9`	Latency			
`Solution_10`	Latency			
`Solution_11`	Latency			
`Solution_12`	Latency			
`Solution_13`	Latency			

a. Based on Table I, which of the solutions has the best complexity-performance tradeoff? Explain briefly.
b. Which of the solutions has the lowest best throughput and lowest latency? Which has the smallest area?
c. Which solution has a throughput of one cycle (i.e., one output sample per cycle)?
d. Why does unrolling the `MAC` loop result in a significant area increase?
e. What are the advantages and disadvantages of loop pipelining vs. loop unrolling in the `MAC` loop?