CALYPTO®

Design · Optimize · Verify

Catapult Design Flow Overview

Catapult Basic Training Module 1

Mike Fingeroff & Rich Toone

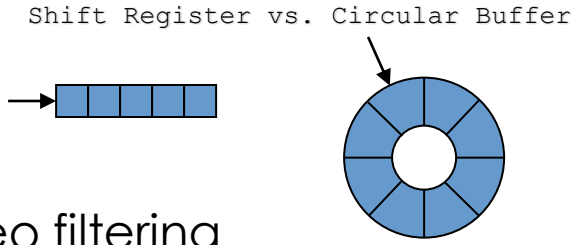
Agenda

- What is High-Level Synthesis?
- Catapult C++ Development Flow Overview
- Setting up the design
- Architectural Constraints
- Resource Constraints
- Scheduling
- RTL and Report Generation
- Automated Verification
- Lab1

Basic Definitions & Concepts

- Algorithm
  - Described by the C++/SystemC source code
  - Examples: FIR, FFT, DCT, Median Filter

- Architecture
  - The general memory access and computation structure of the algorithm
  - Determined by the coding style of the C++/SystemC source code
  - Examples:
    - Shift register vs. circular buffer FIR
    - In-place vs. systolic array FFT
    - Frame-based vs. window-based video filtering



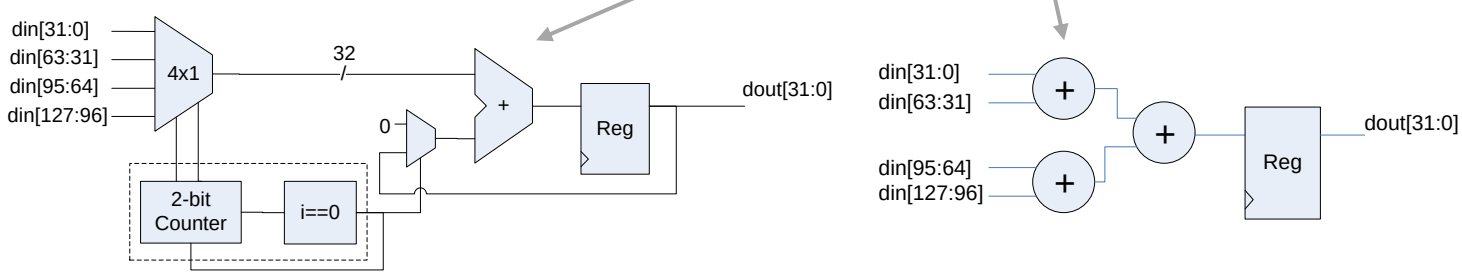
Basic Definitions & Concepts

- Micro-Architecture
  - Implementation variants of a given architecture, e.g.
    - Performing 4 multiplications using 1,2, or 4 multipliers
    - Changing the width of a RAM to increase bandwidth
    - Implementing an array as a RAM or as a register file
    - Pipelining sequential operations to increase throughput
  - This can represent major area and performance changes due to parallelism and I/O bandwidth

```
void accumulate4(int din[4], int &dout){
   int acc=0;
   ACCUM:for(int i=0;i<4;i++){
      acc += din[i];
   }
   dout = acc;
}
```

Algorithm/Architecture

Micro-architectures



CALYPTO®

The Catapult C Flow

Project Files display design files and output results

Solution Picks

Start Page, Flow Manager & Additional Tabs

Task Bar Drives Flow

Transcript History

Tcl Command line

File View Tools Window Help

Solution: solution.v1 All Solutions

Task Bar

Synthesis Tasks

Set Working Directory

Add Input Files

Setup Design

Project Files

solution.v1 (New)

Input Files

FIR_Fixed.cpp

FIR_Compare_Testbench.cp

FIR_Reference.cpp (Exclude

Details - If

Start Page Flow Manager Table

Catapult® SL
System Level Synthesis

Recent Projects

Catapult

Catapult

New Project

Open Project

Getting Started

Toolkits

Videos

AppNotes

Search Support

Manual Bookcase

SupportNet

ESL Communities

ESL Blogs

ESL User Forum

Catapult can't compile C/C++ source code with g++ on 64-bit linux machine

How about Handel-C?

What's the latency cycle with/without SCVerify & Transaction done ?

How to split the memory?

Splitting memory using interleave constraint

HOWTO: Add a register output to mgc_mul?

ac_matrix?

Does Catapult 2008.a support a dualport-RAM which can read and write at the same time?

Floating point representation

Verfication Problem

Verilog, System Verilog and SystemC

How should we treat HIER-4 ?

Matrix conversion support

Hello Example, Vista

Catapult's DAC Suite Presentation Video now Available

Transcript

0 Errors 0 Warnings 12 Infos # 6 Comments 3 Commands Location

#	Message	File(line)	Id
#	/INPUTFILES/3		

solution{4}>

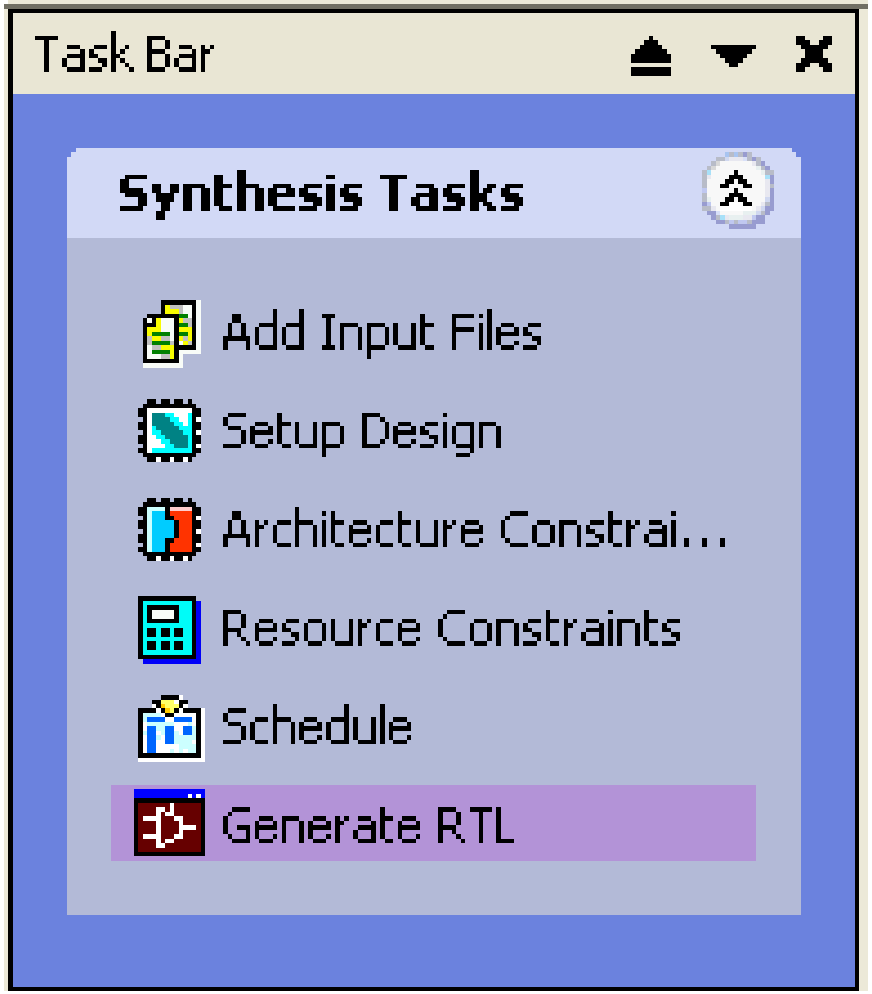
Ready

Working Dir: ...lt/Documents/Catapult_Training/One_day_basic/Lab1

CALYPTO®

Catapult C Full Task Bar Flow

- Setup
  - Add Input Files
  - Setup Design
- Constraints
  - Architectural
  - Resource
- Schedule
- Generate RTL

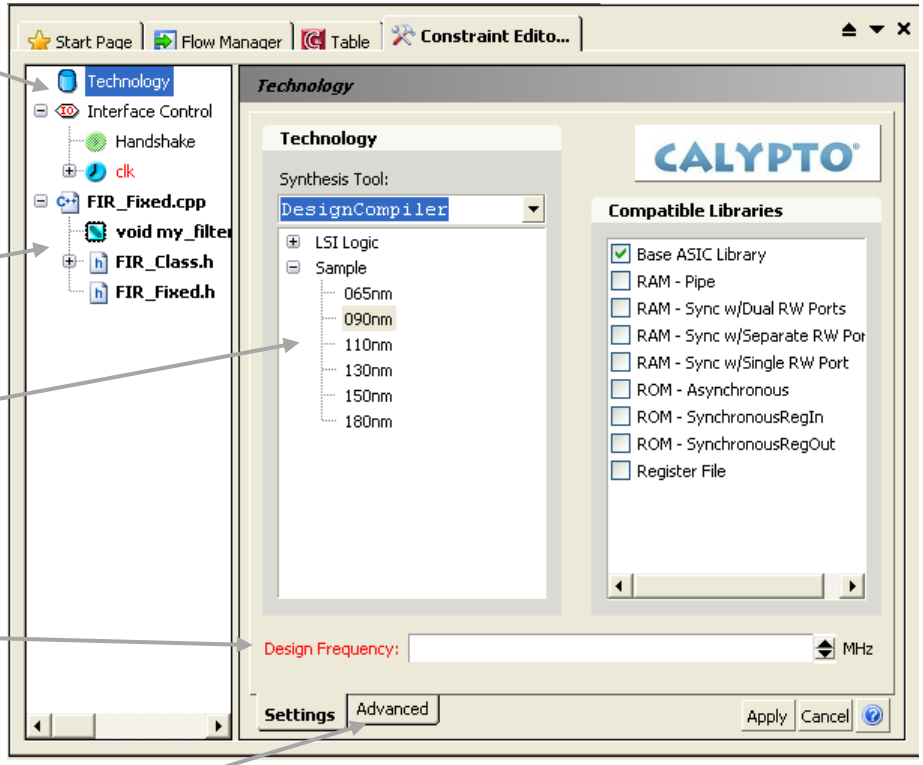


Adding Input Files

- Add files needed for design
- Files may be excluded if not intended for Synthesis
  - Testbench files
  - Helper files used as part of test infrastructure
- Do not add header files that are #include in code
- Options may be set on files (Right Hand Mouse)
  - Exclude from build
  - Options => Individual compiler flags
- Menu: Tools => Set Options => Input
  - Compiler flags
  - Search paths
  - Compiler Home directory

Setup Design

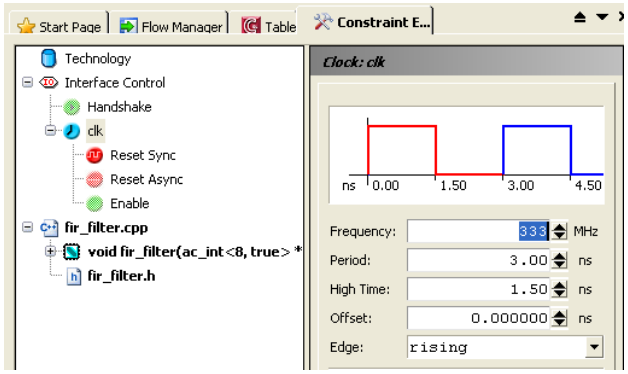
- Select Interface Control
  - Transaction Done
  - Start/Done
  - Reset behavior
- Select Function Hierarchy
  - Block, In-line, Top
- Select Synthesis tool & Library
  - Plus Compatible libraries
    - RAM's & ROM's & custom operators
- Set a clock frequency
  - Output RTL is created to meet this frequency
  - You can use one clock per hierarchical block
- Advanced settings
  - CSA for ASIC
  - Constant MPY extraction

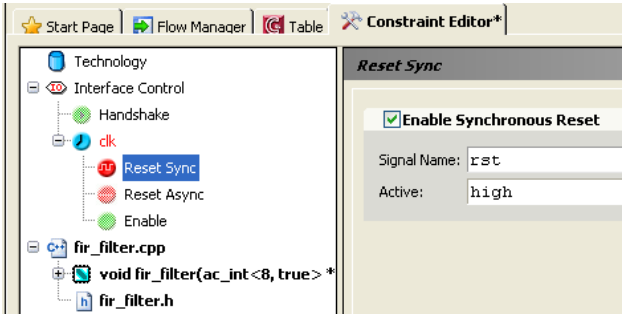


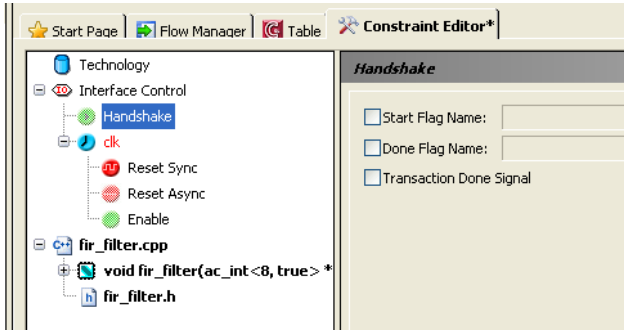
CALYPTO

Setup Design - Interface Control

- Clock
  - One clock created and assigned to all hierarchical blocks (default: clk)
  - Separate hierarchical blocks can have separate clocks
- Reset (per clock)
  - Synchronous reset (default: rst)
  - Asynchronous reset (default: arst_n)
  - Supports both being used
- Enable (per clock)
  - Enable (default: en)
  - Optional
- Transaction Done (global)
  - Most often used for system integration
  - Also used as part of Verification flow
  - Generates a single one-bit flag for each Resource



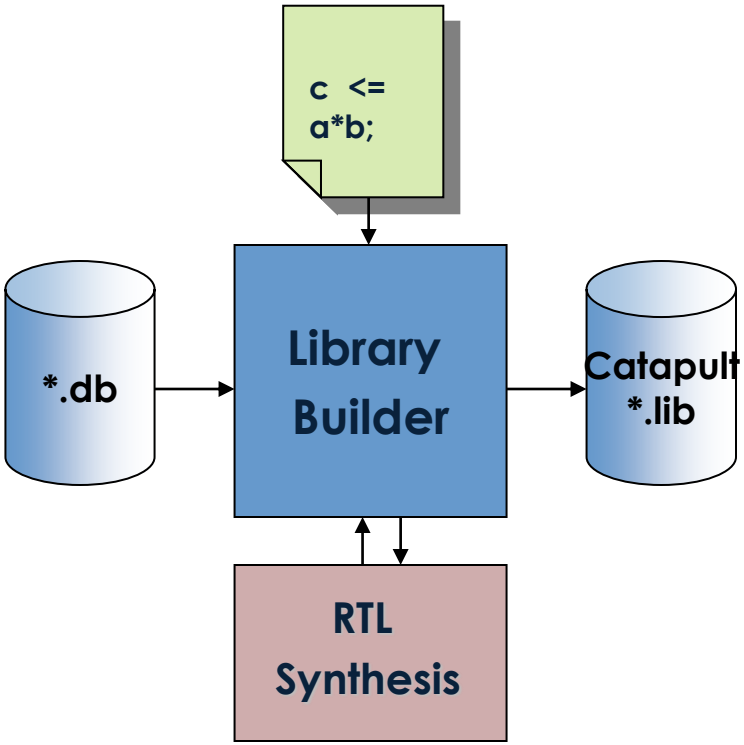




CALYPTO®

Setup Design – Synthesis Tool & Library

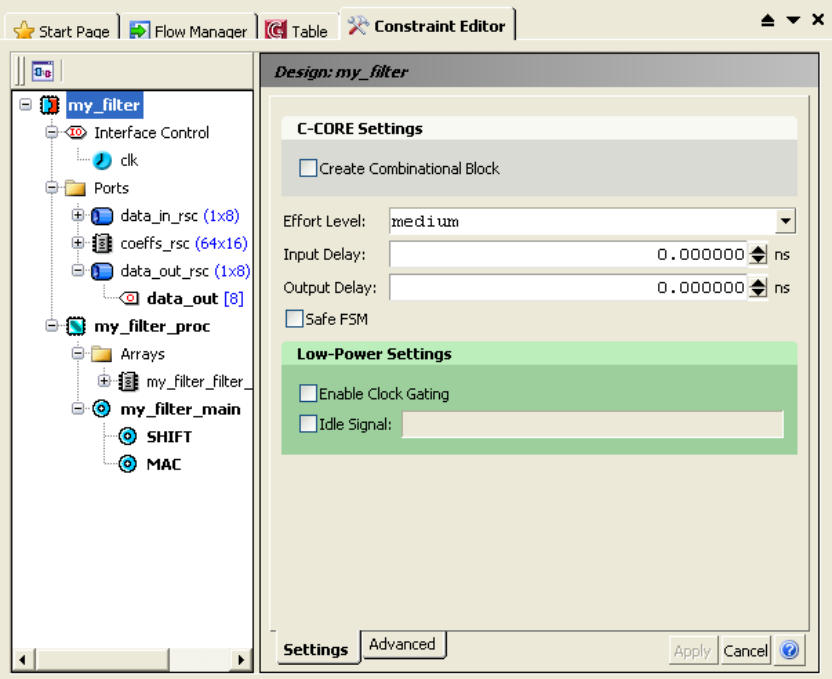
- Technology defines the building blocks in your design
  - Basic Blocks
    - Adders, Subtractors, Bitwise operators
  - IP Blocks
    - RAMs, ROMs, Custom Components (Not covered in this training)
- Libraries are generated using Catapult C Library Builder
  - System is designed to support any RTL synthesis tool
    - Precision RTL, DC, and Magma are currently supported by the GUI
  - FPGA technologies are generated at the factory
  - ASIC Library is generated by user using your RTL synthesis tool
  - IP can be targeted using Library Builder (FPGA or ASIC)
- Best "Quality of Results", Area, and Fmax correlation will be seen when you use a correctly characterized technology library



CALYPTO®

Architectural Constraints

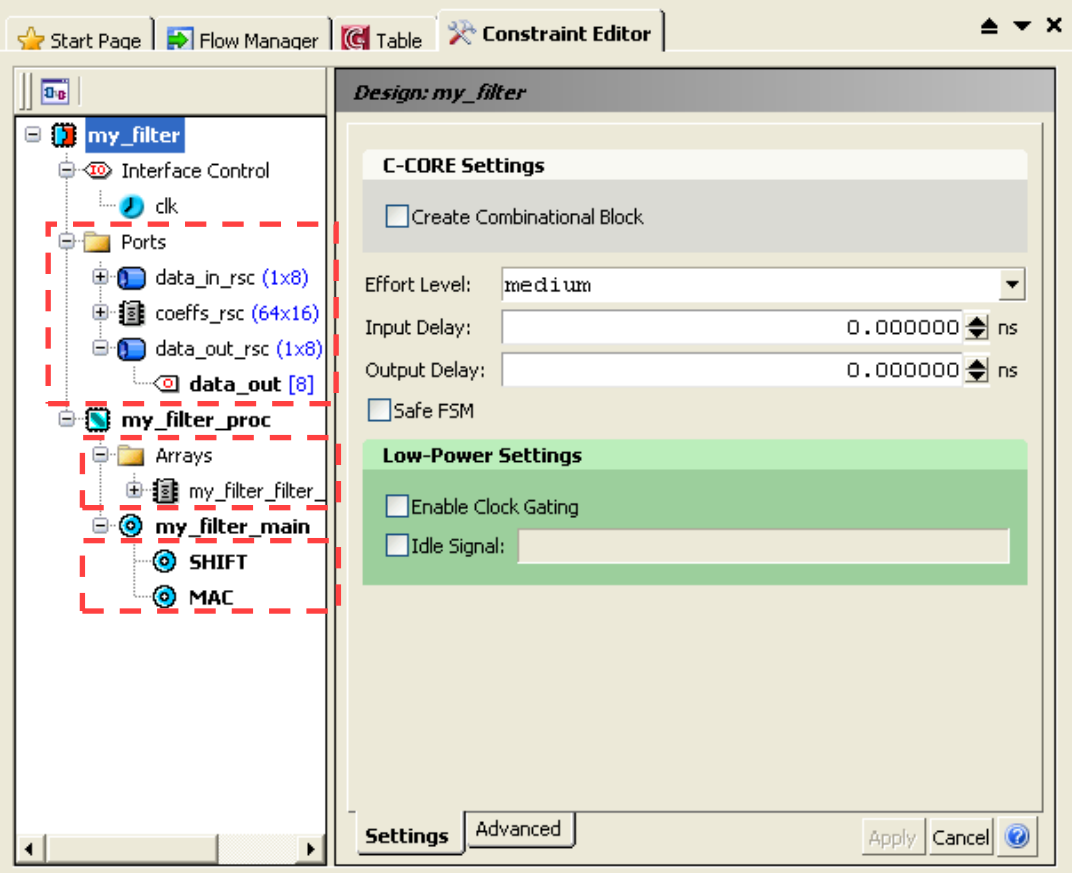
- Drive the RTL micro-architecture
  - I/O interface mapping
  - Variable-to-resource mapping
  - Resource and channel mapping (RAM's/Registers)
  - Loop optimizations
    - Unrolling
    - Pipelining
- Set scheduling constraints
  - Effort & Latency/Area goal
  - Percent Sharing Allocation
  - Power Optimizations



CALYPTO®

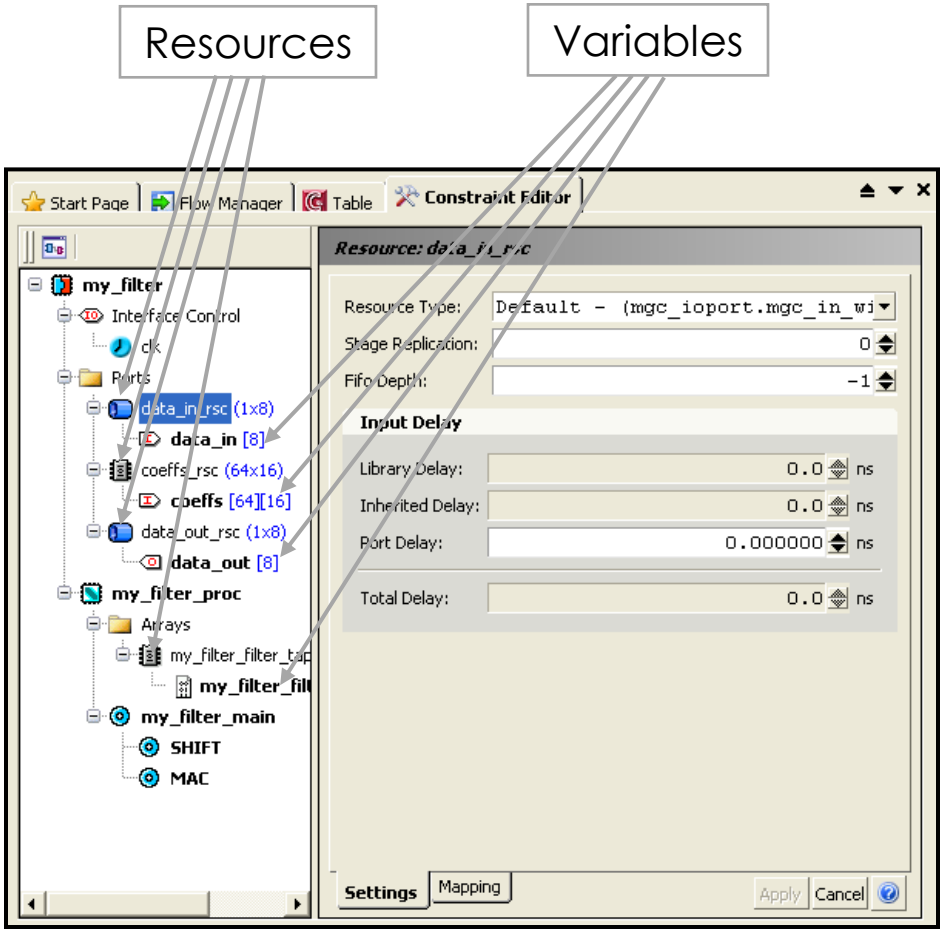
Architectural Constraints

- Ports/Resources
  - Variables beneath
- Internal Arrays
- Constant arrays
- Loops
- Effort Level
  - Medium or High
- Global Clock Gating



Resources & Variables

- A resource is automatically created for every variable
- A resource represents the hardware component interface
- The variable represents the data moving through, or stored in the hardware component
- Variables can be moved onto different Resources (but not down hierarchy)



Interface Constraints

- Interface protocol set by selecting a resource and applying an interface synthesis constraint

Start Page | Flow Manager | Table | Constraint Editor

my_filter
- Interface Control
- Interface
  - data_in:rsc (1x8)
  - coeffs:rsc (1x1024)
  - data_out:rsc (1x8)
- core
  - Arrays
  - main

Resource: data_in:rsc

Resource Type: Default - (mgc_ioport.mgc_in_wire_en)

Stage Replication:

Fifo Depth:

Input Delay

Library Delay:

Inherited Delay:

Port Delay: 0.000000 ns

Default - (mgc_ioport.mgc_in_wire_en)
mgc_ioport.mgc_in_wire
mgc_ioport.mgc_in_wire_en
mgc_ioport.mgc_in_wire_wait
mgc_ioport.mgc_inout_buf_wait
mgc_ioport.mgc_inout_fifo_wait
mgc_ioport.mgc_inout_stdreg_en
mgc_ioport.mgc_inout_stdreg_wait

Click to select resource

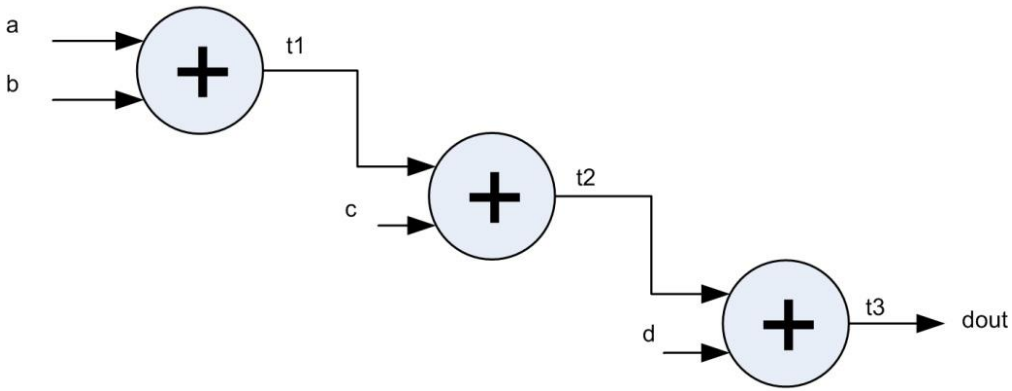
Select interface protocol

CALYPTO®

Basic Definitions & Concepts

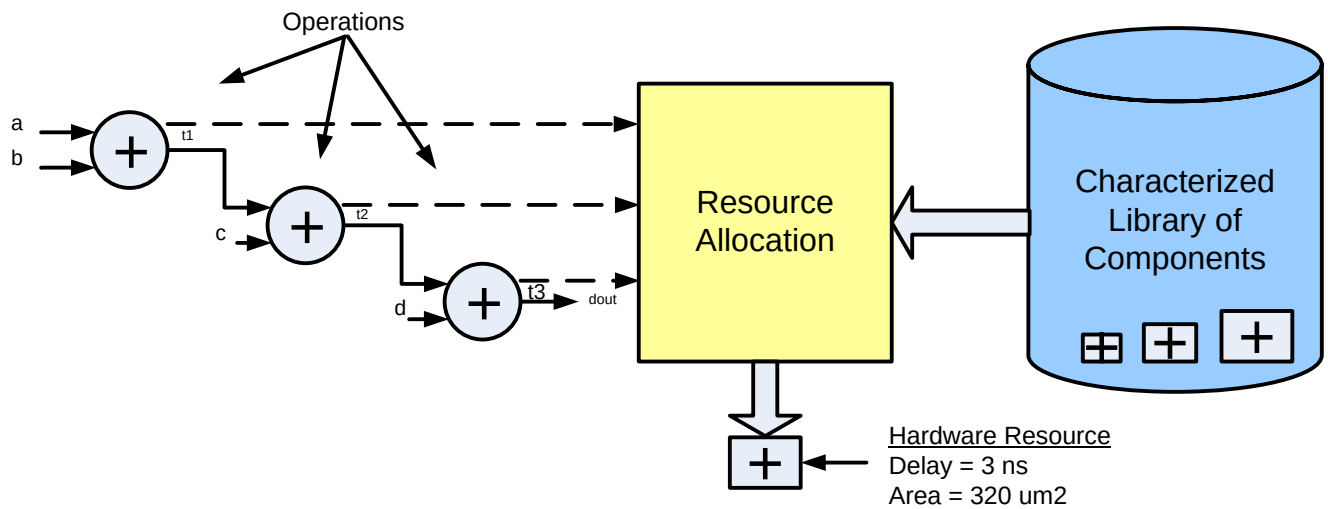
- Data Flow Graph Analysis
  - High-level synthesis analyzes the data dependencies between the various steps in the algorithm
    - Analysis leads to a Data Flow Graph (DFG) description
    - Each node of the DFG represents an operation defined in the C++ code
      - for this example all operations use the "add" operator
    - Connections between nodes represents data dependencies and indicates the order of operations

```
void accumulate(int a, int b,
                int c, int d,
                int &dout){
   int t1,t2;
   t1   = a  + b;
   t2   = t1 + c;
   dout = t2 + d;
}
```



Basic Definitions & Concepts

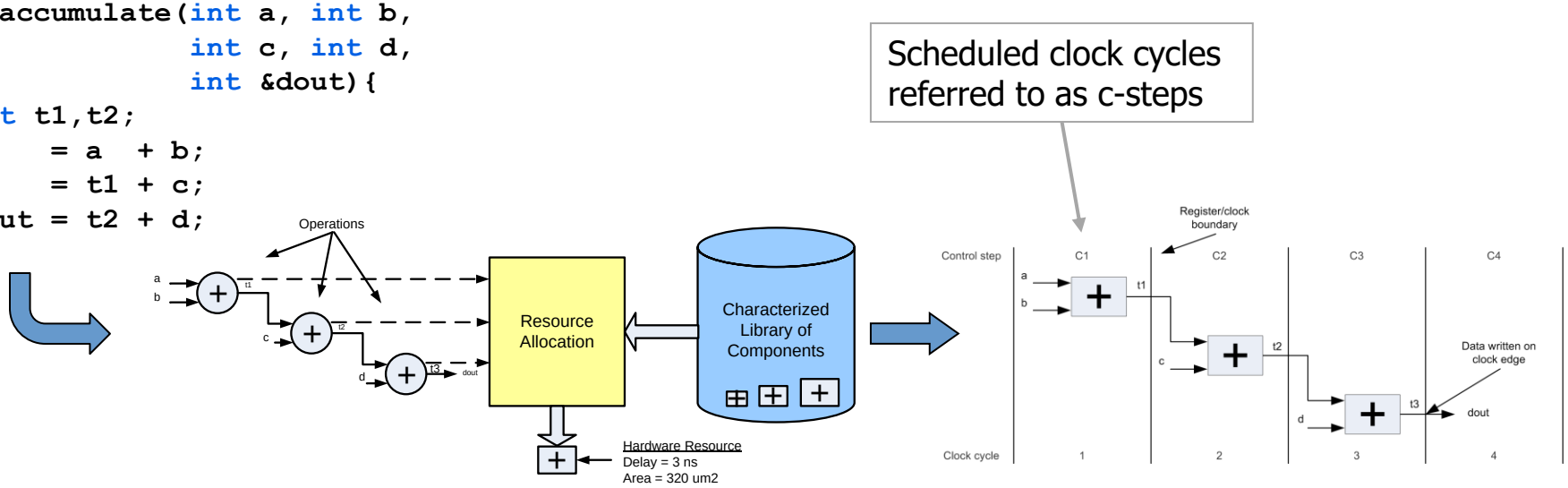
- Resource Allocation
  - After DFG analysis each operation is mapped onto a hardware resource which is then used during scheduling.
  - Resources correspond to a physical implementation of the operator hardware
    - Implementation is annotated with both timing and area information which is used during scheduling
    - Operators may have multiple hardware resource implementations that each have different area/delay/latency trade-offs
  - Resources are selected from a technology specific library



Basic Definitions & Concepts

- Scheduling
  - High-level synthesis adds "time" to the design during the process known as "scheduling"
  - Scheduling automatically shares resources
  - Scheduling takes the operations described in the DFG and decides when (in which clock cycle) they are performed
    - Has the effect of adding registers when needed to meet timing
    - Similar to what RTL designers would call pipelining, by which they mean inserting registers to reduce combinational delays

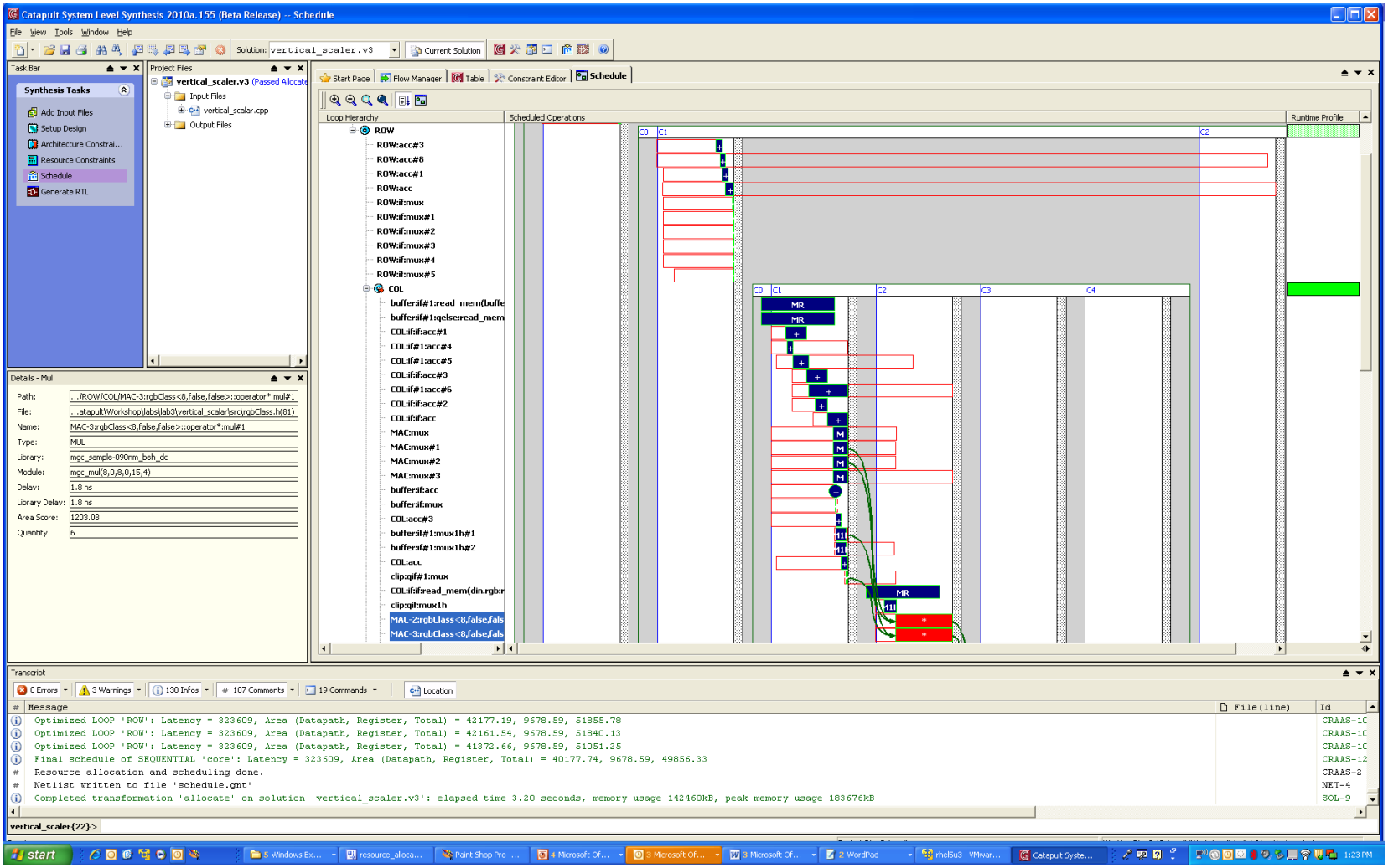
```
void accumulate(int a, int b,
                int c, int d,
                int &dout){
   int t1,t2;
   t1   = a  + b;
   t2   = t1 + c;
   dout = t2 + d;
}
```



CALYPTO®

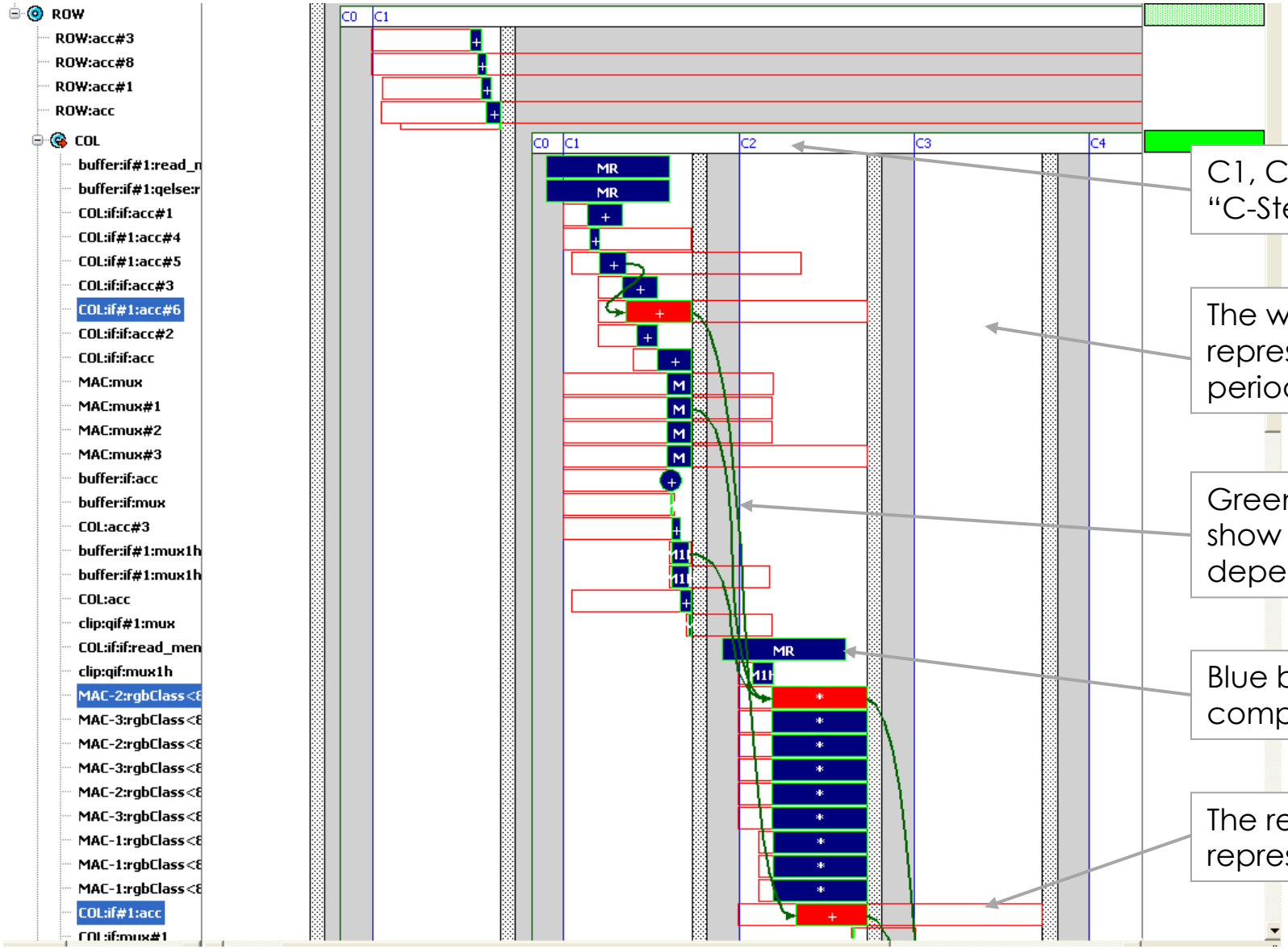
Viewing Scheduling Using the Gantt Chart

- Graphical view of the Algorithm/Architecture



CALYPTO®

The Gantt Chart C-Step View



C1, C2 etc. are "C-Steps"

The white area represents 1 clock period

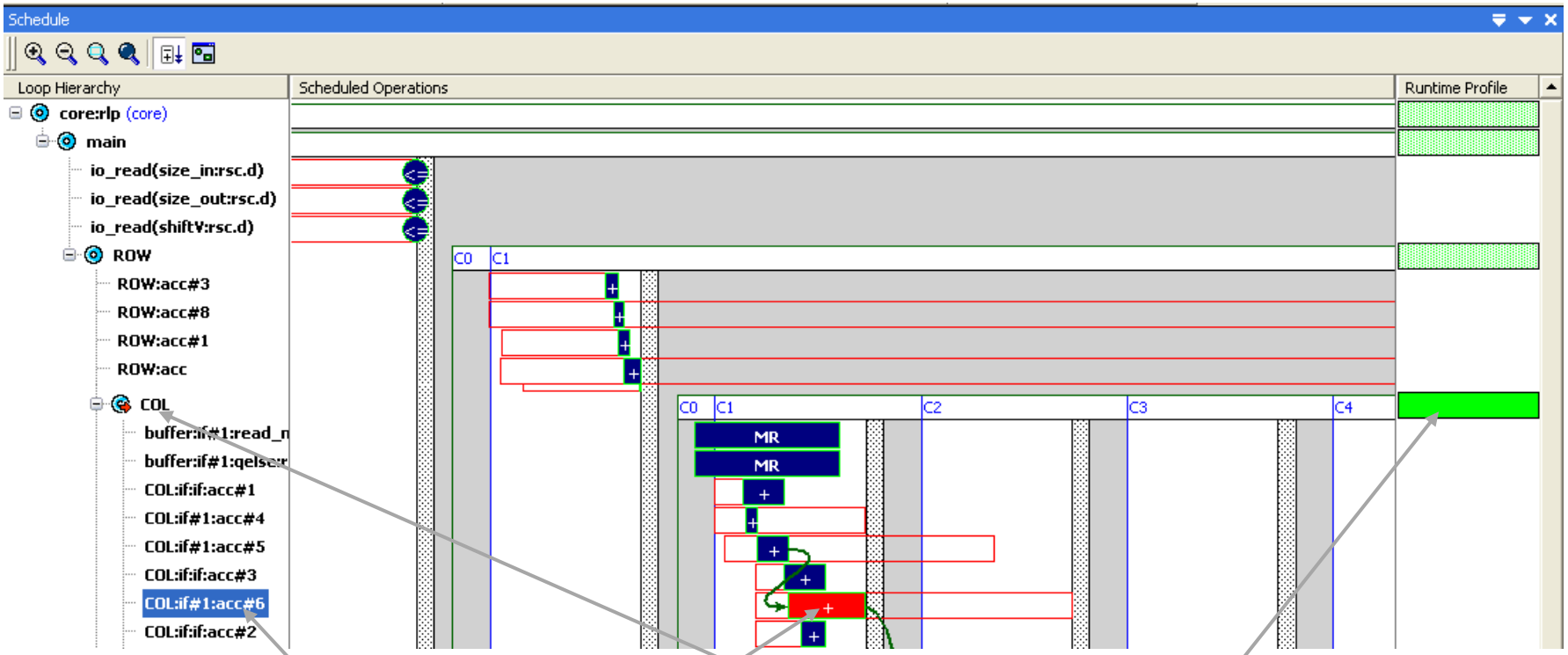
Green arrows show operator dependencies

Blue bars show component delay

The red outline represents "slack"

CALYPTO®

The Gantt Chart C-Step View



Double-click to cross probe from operation name to C source code

The loop profile shows which loops are used the most.

CALYPTO®

The Gantt Chart Component Utilization View

Loop Hierarchy

core:rlp (core)

main

ROW

COL

Query Mode

View sorted

View filtered

Dependencies

Compact Mode

by C-Step

by Component Utilization

by Type

mgc_mul(8,0,8,0,15,3)

mgc_mul(8,0,9,1,15,1)

mgc_mul(8,0,9,1,15,1)

mgc_mul(8,0,9,1,15,1)

mgc_mul(8,0,9,1,15,2)

mgc_mul(8,0,9,1,15,4)

Scheduled Operations

C0 C1 C1 C2 C1 C2 C3 C4 C5

MR

Runtime Profile

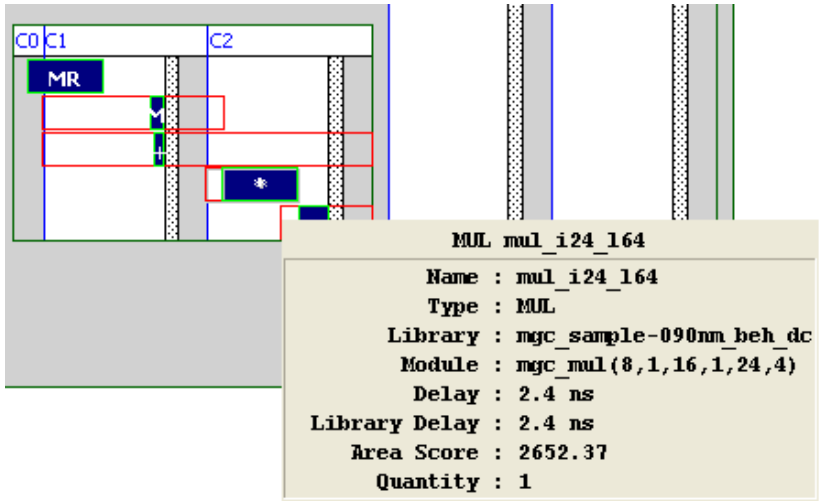
Bring up right mouse click menu & select View sorted by Component Utilization

This multiplier is scheduled in two different clock cycles and can be shared

CALYPTO®

Component Delay

- Catapult Scheduler estimates area and delay for all operators in the data path
  - Using values from library characterization
- Makes decisions based on area/latency goals and constraints as to what size/speed components to use
  - ASIC typically has 4 data points
  - FPGA's typically only have one implementation
- Components will be "scheduled" into the clock period specified by the clock frequency
  - Less "Percent Sharing Allocation"
- Area & delay data can be observed by hovering the mouse pointer over the component

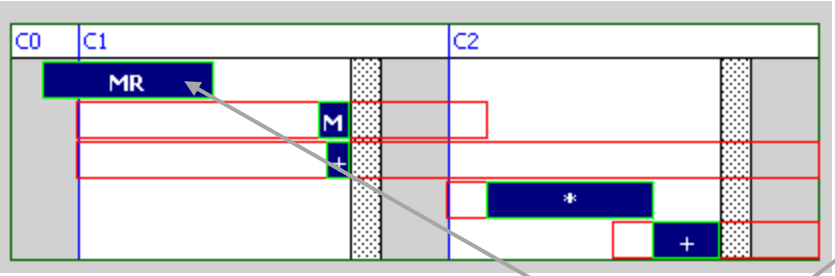


CALYPTO®

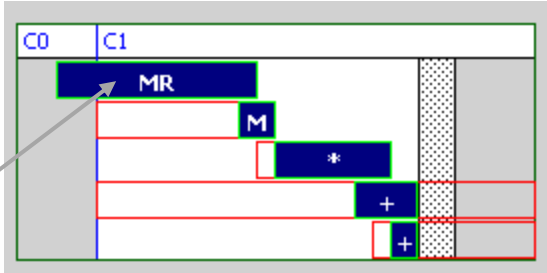
Area vs Latency Scheduling Results

- Area goal
  - Uses slower components to stretch out data path at the expense of latency – trade register area vs. component area
- Latency goal
  - Uses fast components to reduce latency as much as possible
- Area goal + maximum latency constraint
  - Achieve latency and then work on reducing area
- Latency goal + maximum area constraint
  - Deliver lowest latency not exceeding area target

Area Goal:



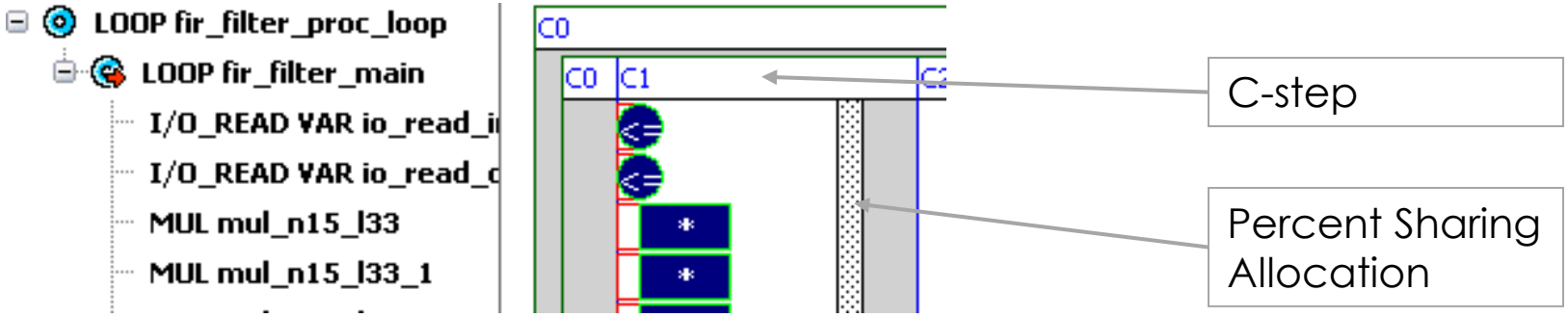
Latency Goal:



Consistent RAM clk2Q

The C-Step

- A "State" in a Finite State Machine
  - More C-Steps in a schedule = More states in RTL FSM
  - More states = More, and larger one-hot muxing
  - Larger one-hot muxes = longer FSM delays
- "Percent Sharing Allocation"
  - Portion of the clock reserved for later FSM control muxing and logic
  - Default is 20%
    - Increase to improve RTL slack across whole design
    - Reduce to decrease latency of schedule
  - FPGA's may need as high as 40% due to routing
  - Too high a value and designs may not schedule due to feedback or minimum delay issues



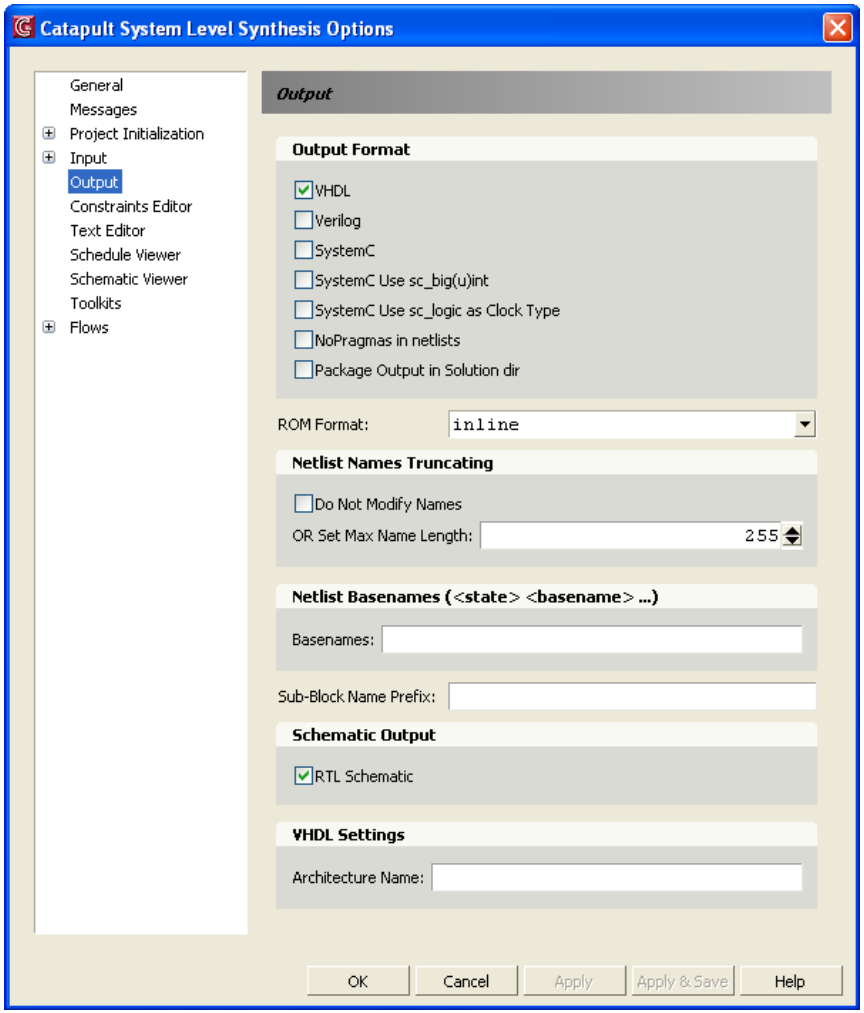
CALYPTO®

Generating RTL

- Last stage of Synthesis
  - Typically takes 50% of run-time, so only done when schedule is satisfactory
- Constructs final RTL implementation
  - FSM generation
  - Timing analysis and sharing/replication
  - Reporting – Area, Bill of Materials
  - Schematics
  - Downstream tool script generation

Output Options

- Tools => Set Options => Output
- Output languages
  - VHDL
  - Verilog
  - SystemC RTL
- Remainder will probably remain as defaults
  - unless there are specific downstream requirements



Comparing Solutions

- Each different set of constraints will result in a new "solution"
- Catapult places each solution in a different directory on disk
- Multiple solutions are listed in the Table View

Start Page | Flow Manager | Table | Constraint Editor

Report: General

Solution	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Total Area	Slack
my_filter.v1 (extract)	65	650.00	67	670.00	30266.60	2.51
my_filter.v2 (extract)	65	650.00	64	640.00	20823.69	1.41
my_filter.v3 (extract)	34	340.00	32	320.00	24844.10	2.60
my_filter.v4 (extract)	16	160.00	16	160.00	34850.11	1.95
my_filter.v5 (extract)	2	20.00	1	10.00	244390.65	2.91

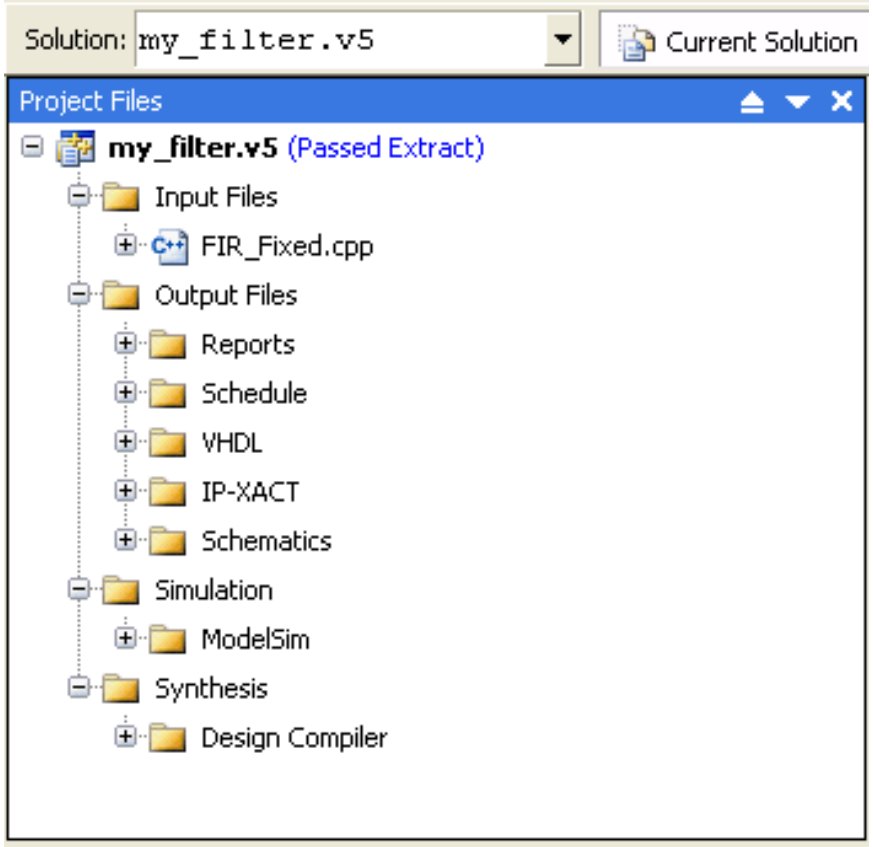
Bar chart & X-Y plot views also available

Timing only after Generate RTL

CALYPTO®

Output Files

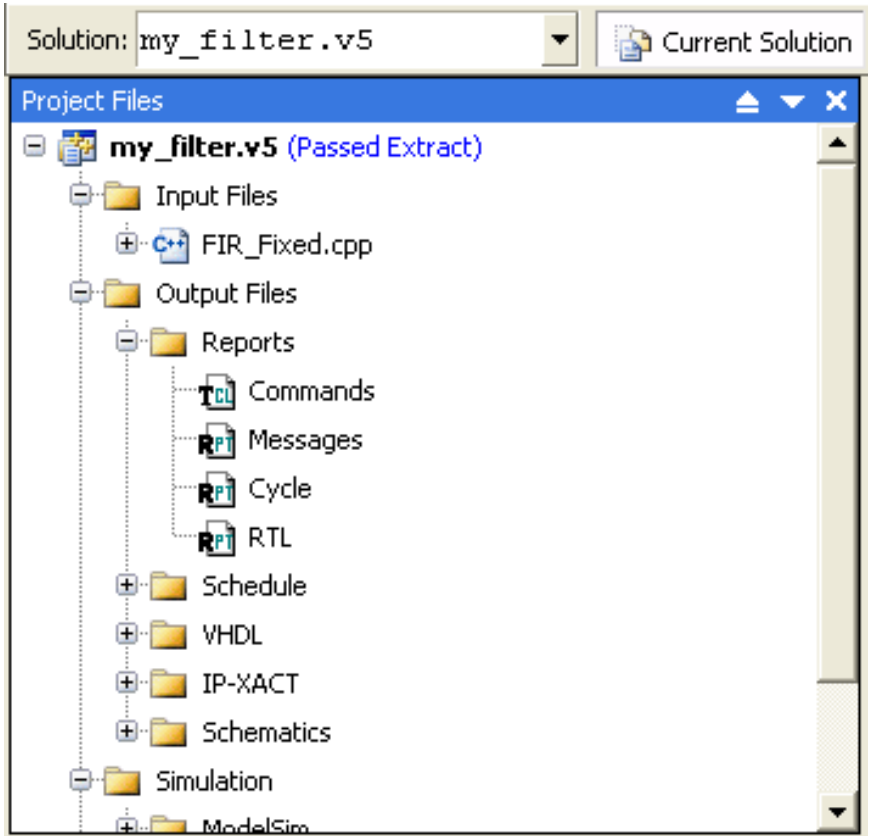
- Consolidated for each solution directory
  - Reports
  - Schedule
  - RTL
  - Schematics
- Simulation & Verification script generation
- RTL Synthesis and downstream tool scripts
- Flow Manager setup



CALYPTO®

Reports

- Commands
  - TCL script file of commands required to reproduce the solution ("directives.tcl")
- Messages from Catapult for Solution
- Cycle data report
- RTL report
  - Most used
  - Includes bill of materials & critical timing report



RTL report – Bill of Materials

- Components
  - Bit widths
  - Area
  - Delay
  - Quantity
- Alloc
  - Post Scheduling
- Assign
  - Post RTL generation

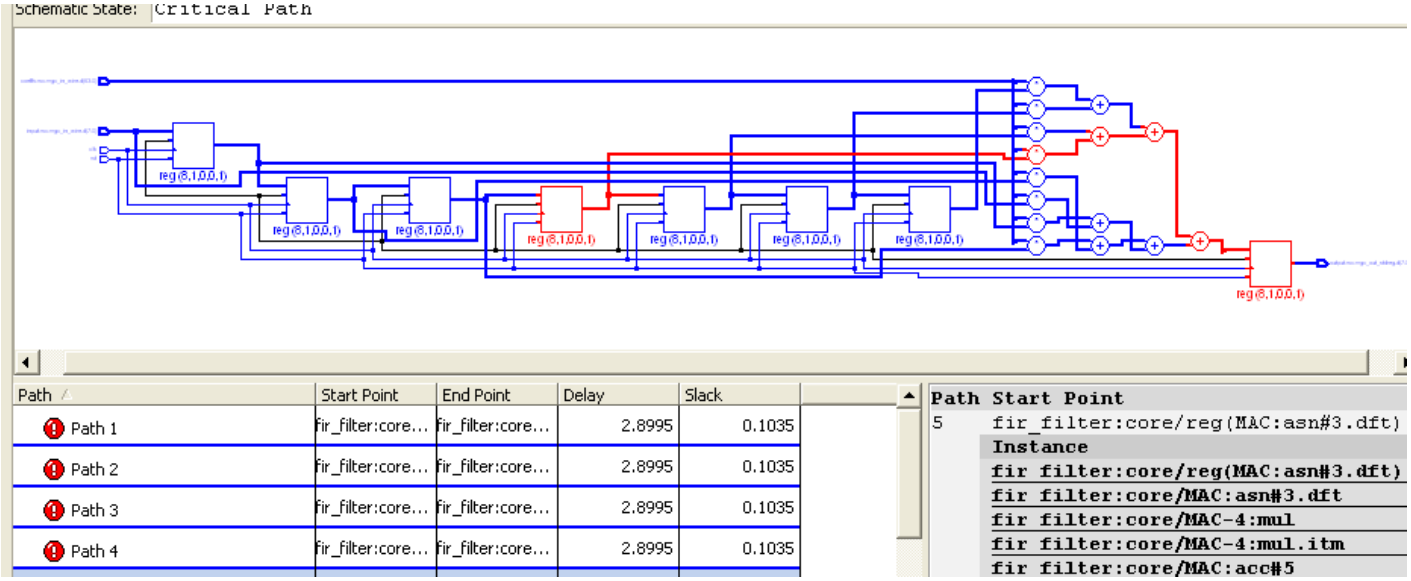
Start Page | Flow Manager | Table | Constraint Editor | Schedule | rtl.rpt

```
Solution Settings: my_filter.v2
  Current state: extract
  Project: Catapult

  Design Input Files Specified

  Processes/Blocks in Design

  Bill Of Materials (Datapath)
    Component Name                                  Area Score  Delay Post Alloc Post Assign
    ---------------------------------------------- ---------- ------ ---------- -----------
    [Lib: mgc_ioport]
    mgc_in_wire_en(2,8)                                   0.000  0.000          1           1
    mgc_out_stdreg_en(4,8)                                0.000  0.000          1           1
    [Lib: mgc_sample-090nm_beh_dc]
    mgc_add(15,1,1,0,16,3)                              170.718  0.594          1           0
    mgc_add(15,1,1,0,16,4)                              141.969  1.079          0           1
    mgc_add(30,0,24,1,30,4)                             522.215  2.778          1           1
    mgc_add(6,0,1,1,7,4)                                102.578  0.766          1           1
    mgc_and(1,2,4)                                        3.131  0.031          0           1
    mgc_and(1,3,4)                                        4.715  0.054          0           1
    mgc_and(30,2,4)                                      93.937  0.031          1           0
    mgc_and(30,3,4)                                     141.439  0.054          0           1
    mgc_and(6,2,4)                                       18.787  0.031          0           1
    mgc_and(8,2,4)                                       25.050  0.031          0          63
    mgc_mul(8,1,16,1,24,4)                             2652.370  2.435          1           1
    mgc_mux(1,1,2,4)                                      5.716  0.133          0           1
    mgc_mux(6,1,2,4)                                     34.295  0.133          0           1
    mgc_mux(7,1,2,4)                                     40.011  0.133          1           0
    mgc_mux(8,1,2,4)                                     45.727  0.133         64          65
    mgc_mux(8,6,64,4)                                  1835.141  0.453          1           1
```

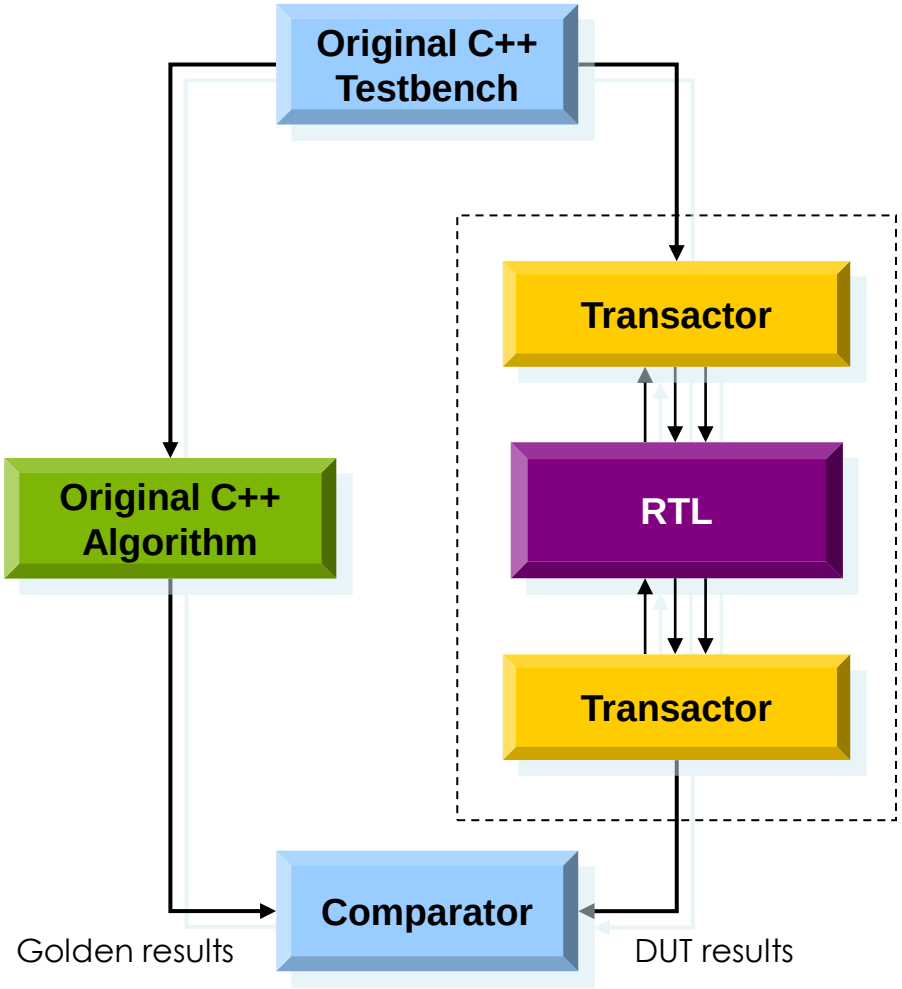
RTL Report – Critical Path Timing

- Critical path report
- Can be viewed in schematic
- Estimated timing
  - More accurate timing available after RTL synthesis

```
Register-to-Variable Mappings

Timing Report
  Critical Path
    Max Delay:  2.89947402
    Slack:      0.1035289800000001

    Path                                                    Startpoint
    ------------------------------------------------------  ----------------------
    1                                                       fir_filter:core/reg(MAC

      Instance                                              Component
      --------                                              ---------
      fir_filter:core/reg(MAC:asn.dft)                      mgc_reg_pos_8_0_0_1_1_0
      fir_filter:core/MAC:asn.dft
      fir_filter:core/MAC-1:mul                             mgc_mul_8_1_8_1_15_4
      fir_filter:core/MAC-1:mul.itm
```



SCVerify Catapult Verification Extension

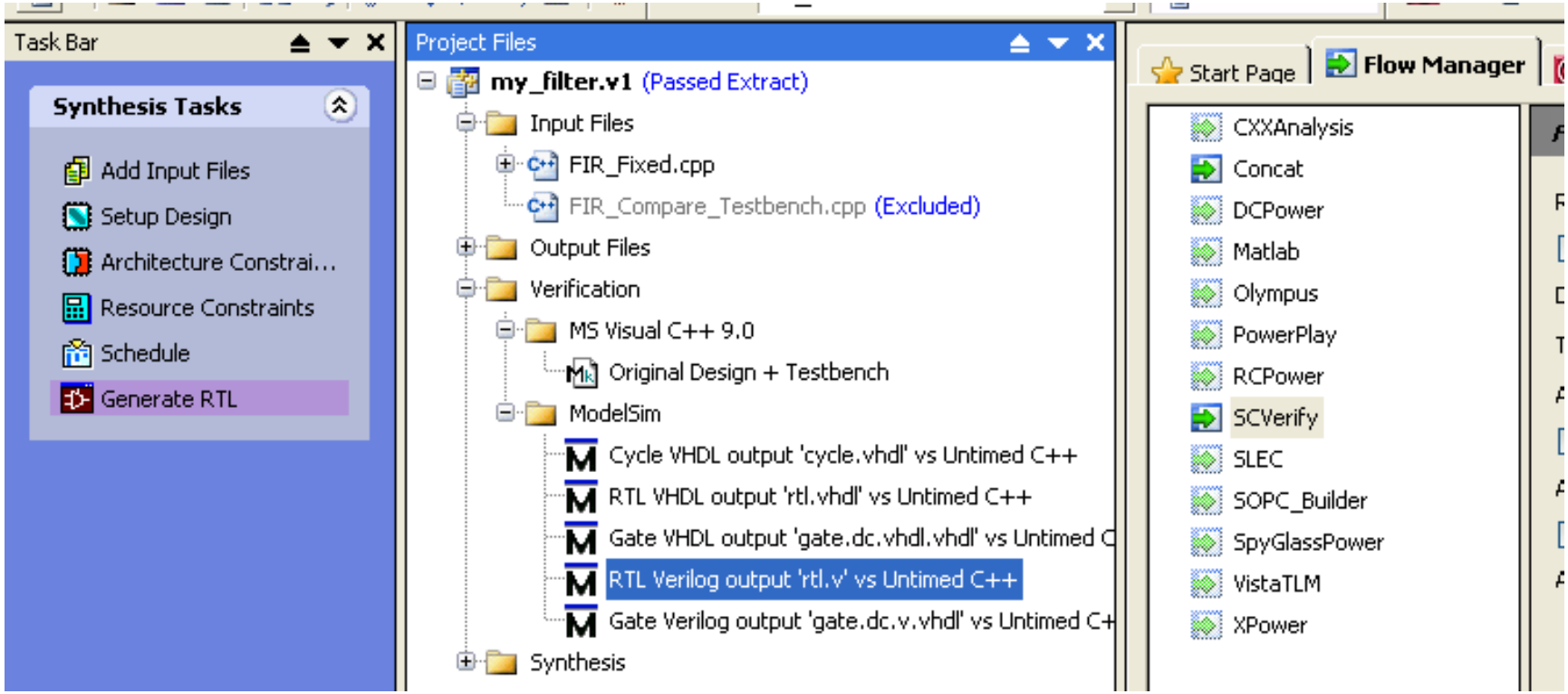


- Facilitates the Verification of the synthesized design
- The original C++ testbench can be reused (after a few changes) to verify the RTL
- Transactors convert function calls to pin-level signal activity
- Push button verification solution creates Makefiles and Simulation Scripts for ModelSim

CALYPTO®

Running SCVerify Automated Verification

- Verification scripts generated automatically
  - Push-button checking of C++ against generated RTL



CALYPTO®

Lab5 – Catapult Design and Analysis Flow

- In this lab exercise you will
  - Become acquainted with the Catapult GUI
  - Setup & synthesize a simple example design
  - Use the Catapult tools to explore the schedule and resulting schematics

- Unzip ../lab5.zip
- Go to the Lab5 directory
- Follow the instructions in Lab5.doc

CALYPTO®

Design • Optimize • Verify