

FPGA - Fabric, Design and Architecture

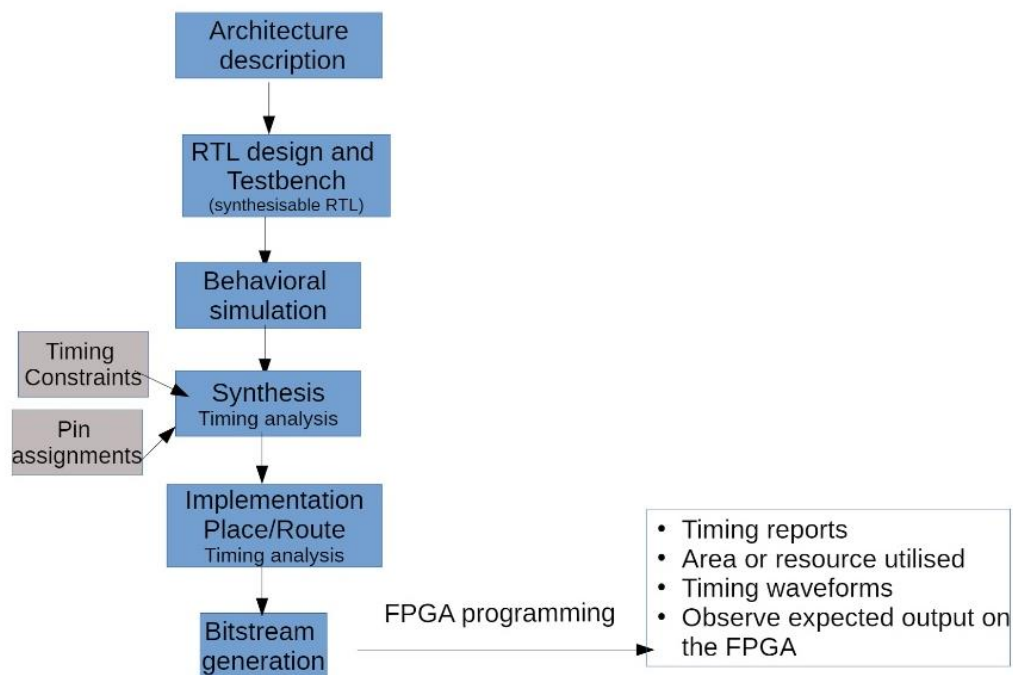
This repository contains all the information studied and created during the [FPGA - Fabric, Design and Architecture](#) workshop.

Vivado / Open FPGA

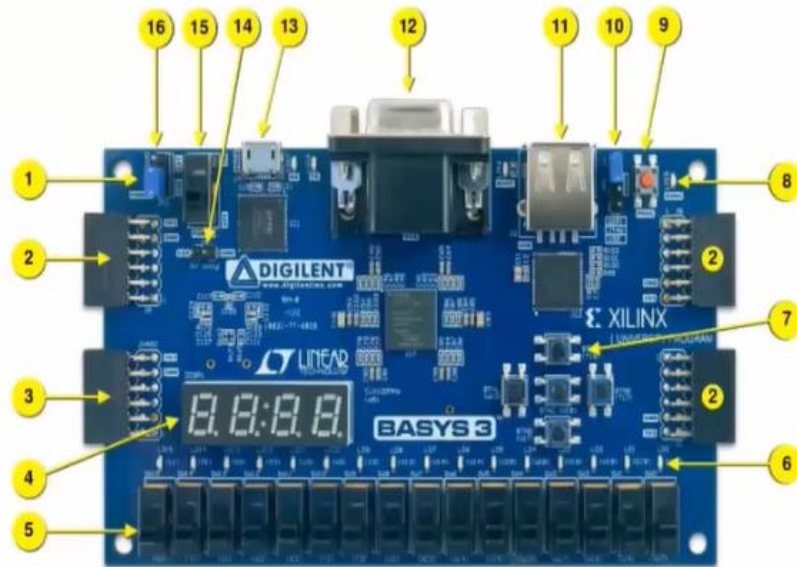
Day 1

FPGA Introduction

FPGA Design Flow



Design of a 4-bit counter through Vivado on Basys3 FPGA



- 15 Power Switch
- 1 Power good LED
- 4 Four digit 7-segment display
- 12 VGA connector
- 5 Slide switches
- 13 Shared UART/ JTAG USB port
- 6 LEDs (16)
- 14 External power connector
- 7 Pushbuttons (5)
- 15 Power Switch
- 8 FPGA programming done LED

20/10/21

18

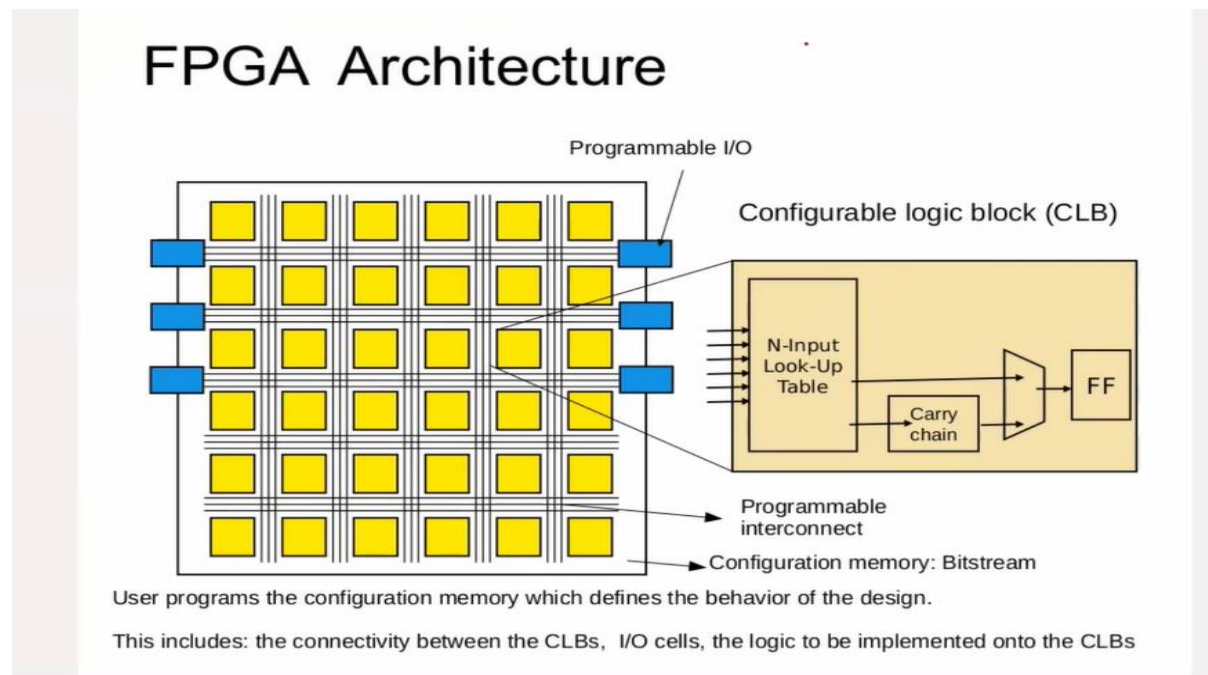
Basys 3 Board for Implementation of Design

Use of Virtual Input/Output (VIO)

Flow of VIO

Day 1 - Exploring FPGA Basics and Vivado

FPGA Architecture



Counter Example in Vivado

A 4-bit up counter is being used for exploring the Vivado tool and OpenFPGA. Below mentioned the RTL for the counter modules that is being used

```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////////////////
// Description: 4 bit counter with source clock (100MHz) division.
```

```
module counter_clk_div(clk,rst,counter_out);
```

```
    input clk,rst;
    reg div_clk;
    reg [25:0] delay_count;
    output reg [3:0] counter_out;
```

```
    //////////clock division block//////////
```

```

always @(posedge clk) begin

    if(rst) begin
        delay_count<=26'd0;
        div_clk <= 1'b0; //initialise div_clk
    end
    else
        if(delay_count==26'd212) begin
            delay_count<=26'd0; //reset upon reaching the max value
            div_clk <= ~div_clk; //generating a slow clock
        end
        else begin
            delay_count<=delay_count+1;
        end
    end
end

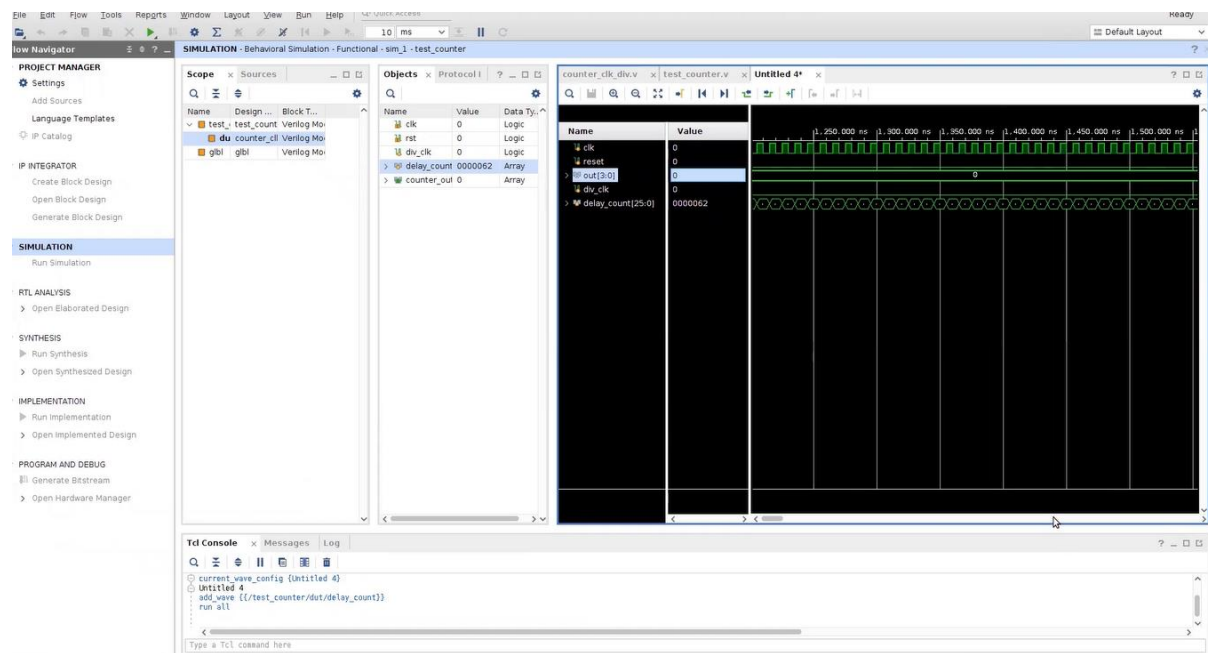
//////////4 bit counter block//////////
always @(posedge div_clk) begin

    if(rst) begin
        counter_out<=4'b0000;
    end
    else begin
        counter_out<= counter_out+1;
    end
end

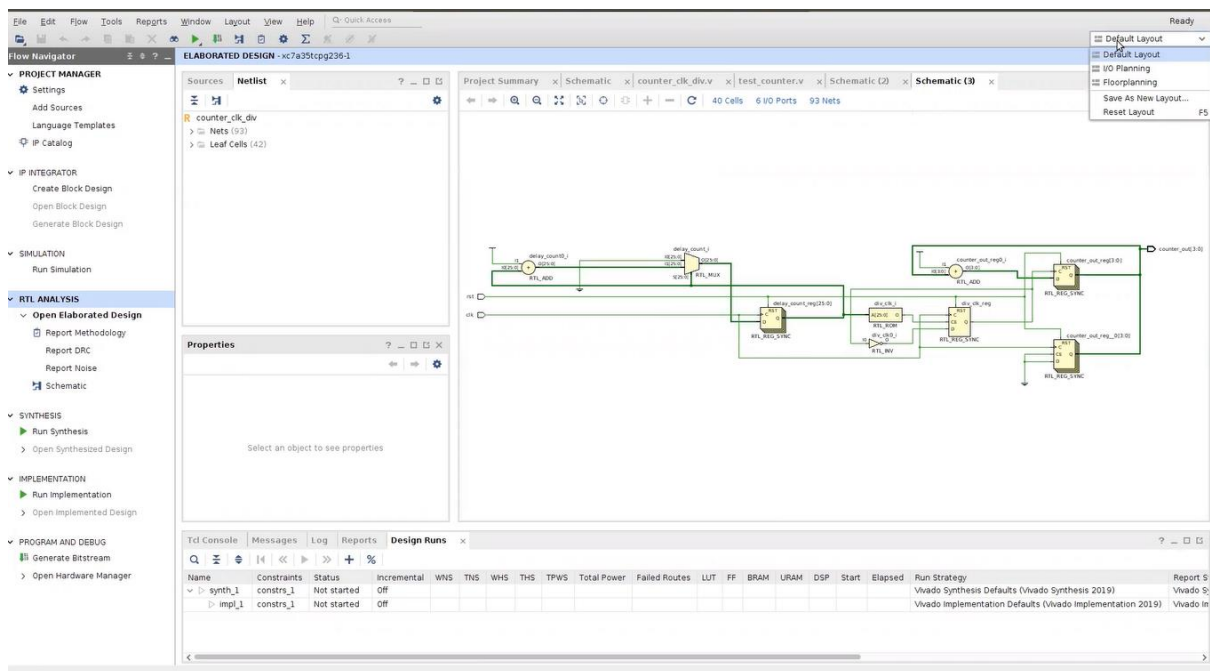
endmodule

```

Counter Simulation and Elaboration



The snippet below is the schematic of the counter design after elaboration.



Constraints

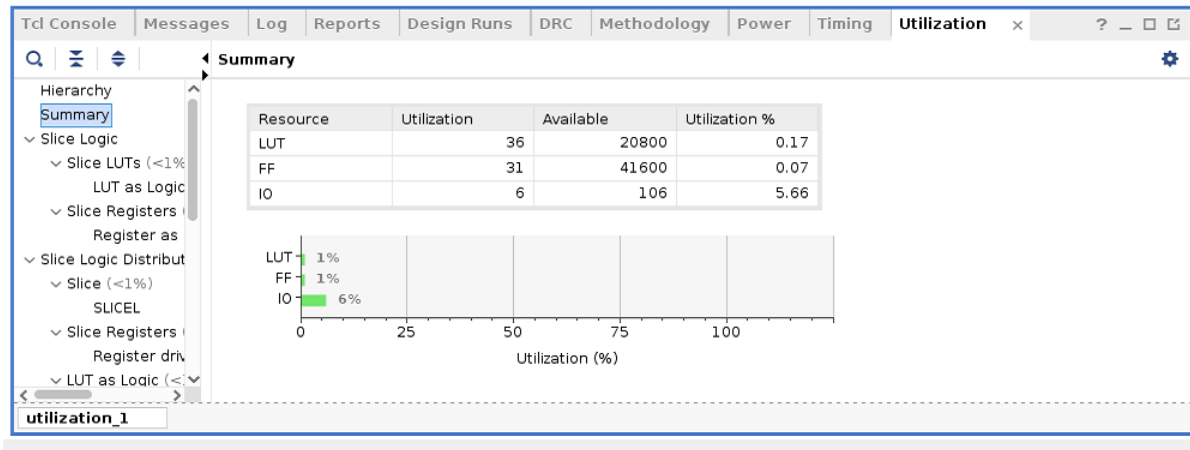
Constraints in simple are the specifications of your design like timing specifications, ports declaration, input/output delays, etc.

Bitstream

A bitstream is a binary sequence that comprises a sequence of bits. These are used in FPGA applications for programming purposes and to establish communication channels. FPGA bitstream is a file containing the programming data associated with your FPGA chip.

Counter Timing, Power and Area

Implementation of a design also gives details like the timing summary, device utilization, power analysis, etc. The below snippets show the brief timing sumary, implementation and power analysis of the up-counter design.

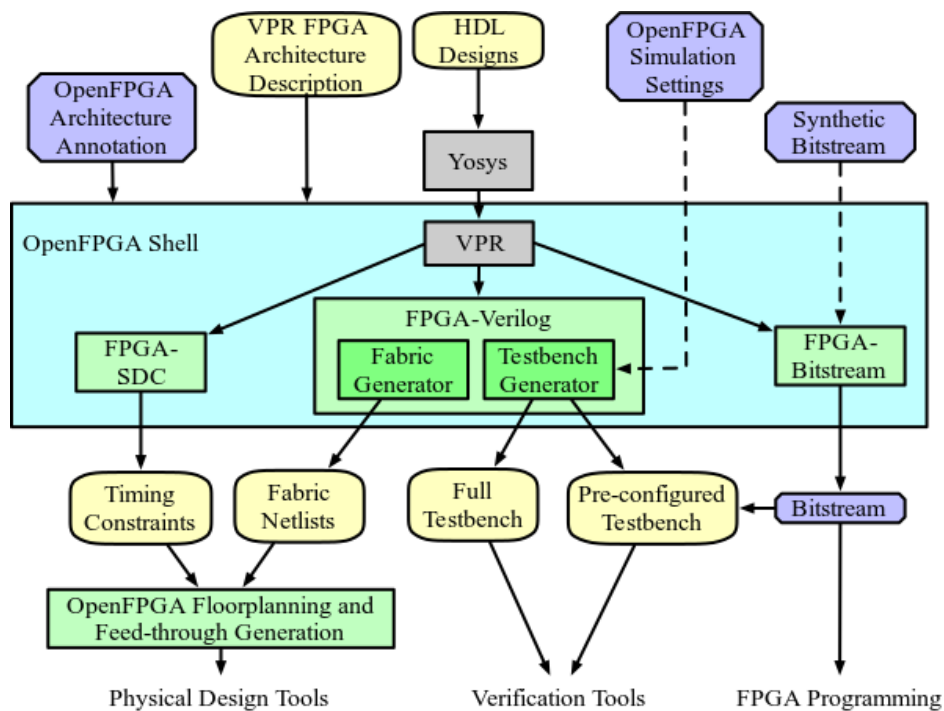


Day 2 - Exploring OpenFPGA, VPR and VTR

Introduction To OpenFPGA

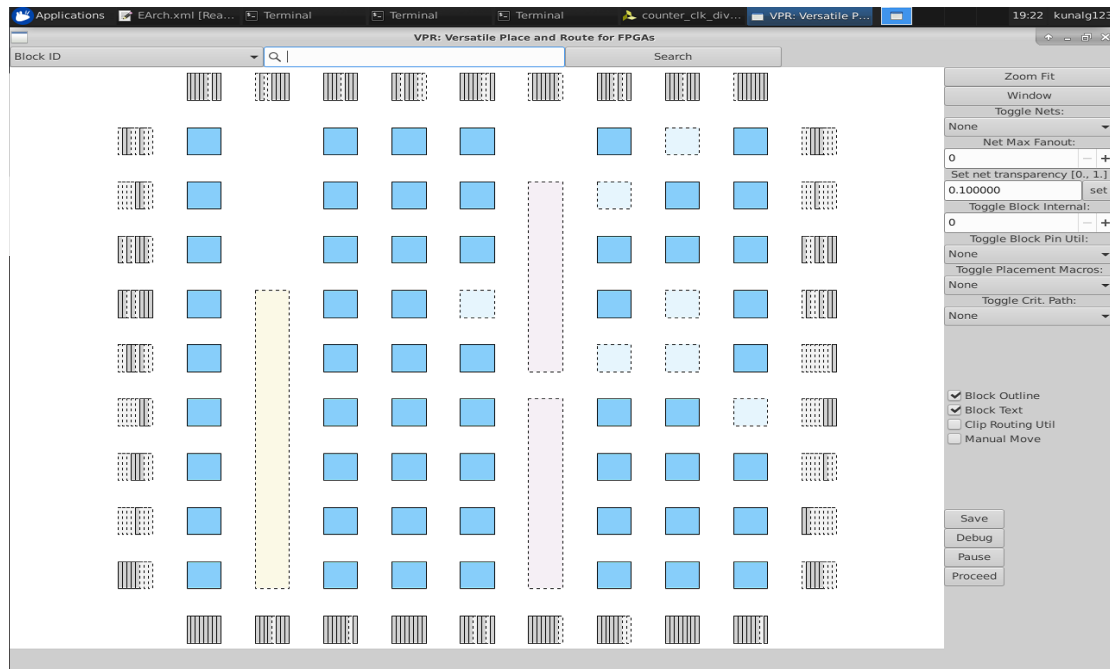
The OpenFPGA framework is the first open-source FPGA IP generator which supports highly-customizable homogeneous FPGA architectures. OpenFPGA provides a full set of EDA support for customized FPGAs, including Verilog-to-bitstream generation and self-testing verification. OpenFPGA targets to democratizing FPGA technology and EDA techniques, with agile prototyping approaches and constantly evolving EDA tools for chip designers and researchers.

Some key features of OpenFPGA are: - Use of Automation Techniques - Reduction of FPGA development cycle to few days - Provides open source design tools



VPR

VPR (Versatile Place and Route) is an open source academic CAD tool designed for the exploration of new FPGA architectures and CAD algorithms, at the packing, placement and routing phases of the CAD flow. As input, VPR takes a description of an FPGA architecture along with a technology-mapped user circuit. It then performs packing, placement, and routing to map the circuit onto the FPGA. The output of VPR includes the FPGA configuration needed to implement the circuit and statistics about the final mapped design (eg. critical path delay, area, etc).



o invoke VPR from terminal:

```
$VTR_ROOT/vpr/vpr \
$VTR_ROOT/vtr_flow/arch/timing/EArch.xml \
<blif-file-path \
--route_chan_width 100 \
--disp on
```

The basic VPR flow involves below mentioned steps:

- Packing - combines primitives into complex blocks
- Placement - places complex blocks within the FPGA grid
- Routing - determines interconnections between blocks
- Analysis - analyzes the implementation

VTR

The Verilog to Routing (VTR) project provides open-source CAD tools for FPGA architecture and CAD research. The VTR design flow takes as input a Verilog description of a digital circuit, and a description of the target FPGA architecture.

To invoke VTR from command-line:

```
***
```

```
$VTR_ROOT/vtr_flow/scripts/run_vtr_flow.py \  
$VTR_ROOT/doc/src/<verilog-file-path>  
$VTR_ROOT/vtr_flow/arch/timing/EArch.xml \  
-temp_dir . \  
--route_chan_width 100
```

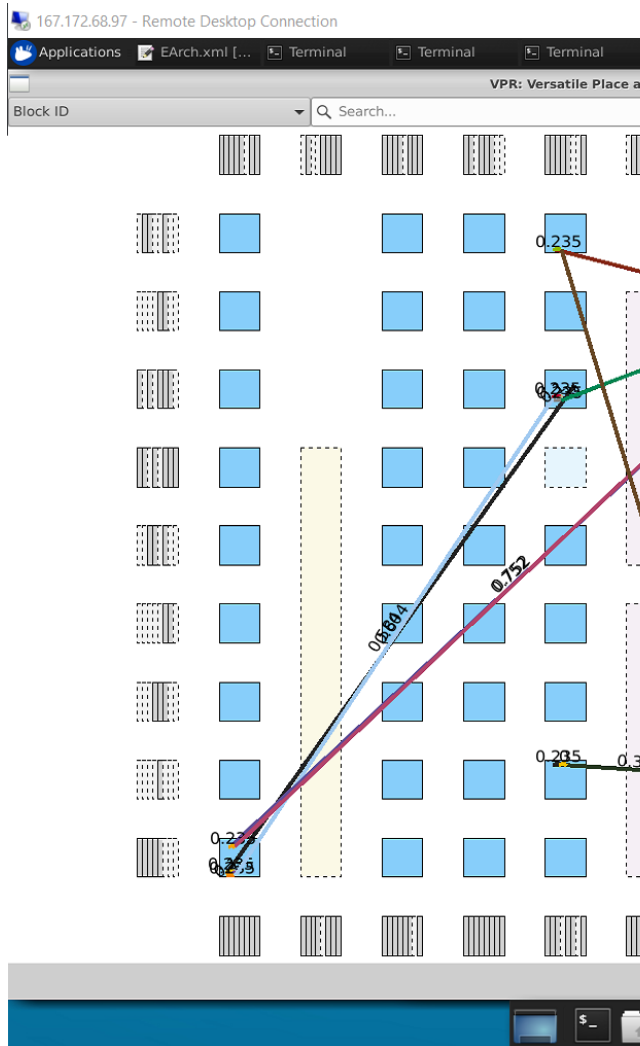
^^^

VTR Flow

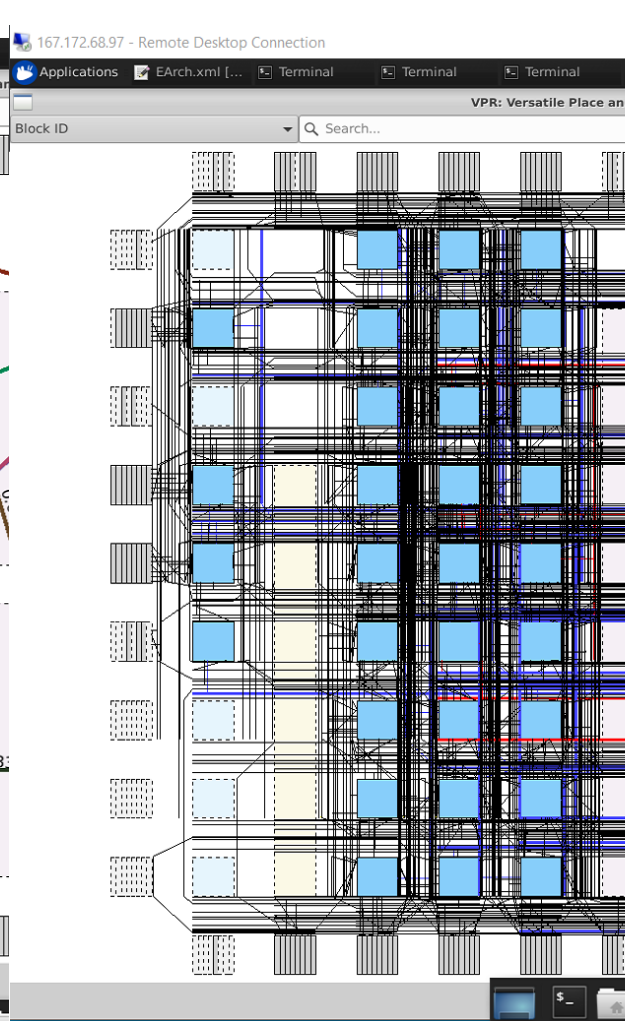
The basic VTR flow performs:

- Elaboration & Synthesis (ODIN II)
- Logic Optimization & Technology Mapping (ABC)
- Packing, Placement, Routing & Timing Analysis (VPR)

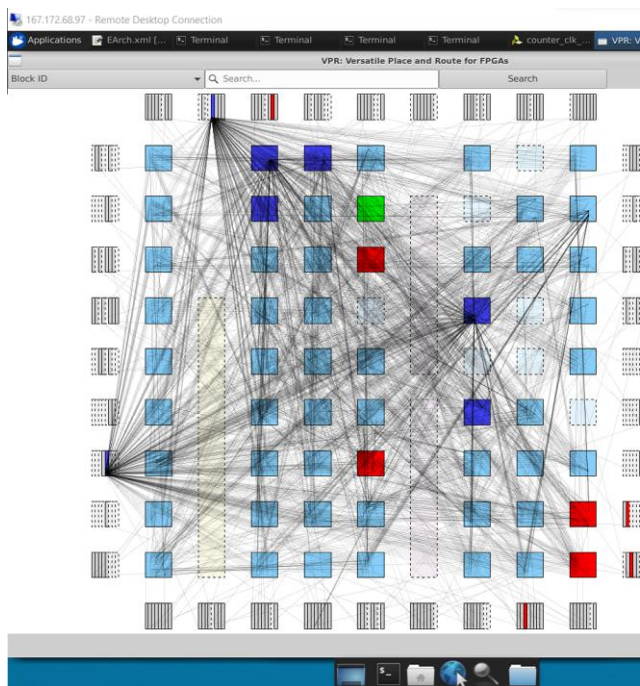
The snippets mentioned below show some of the stages from the VTR flow



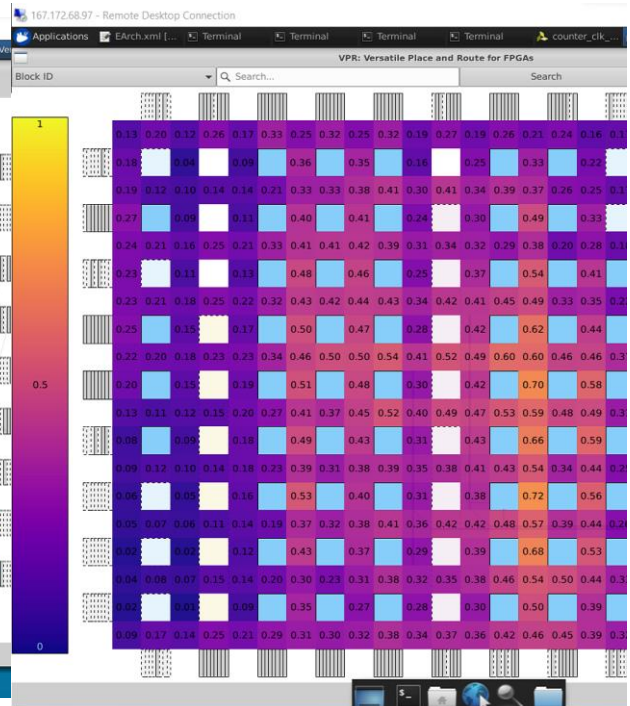
Critical Paths



Nets



Logical Connections



Routing Utilizations

Timing Analysis VTR Flow

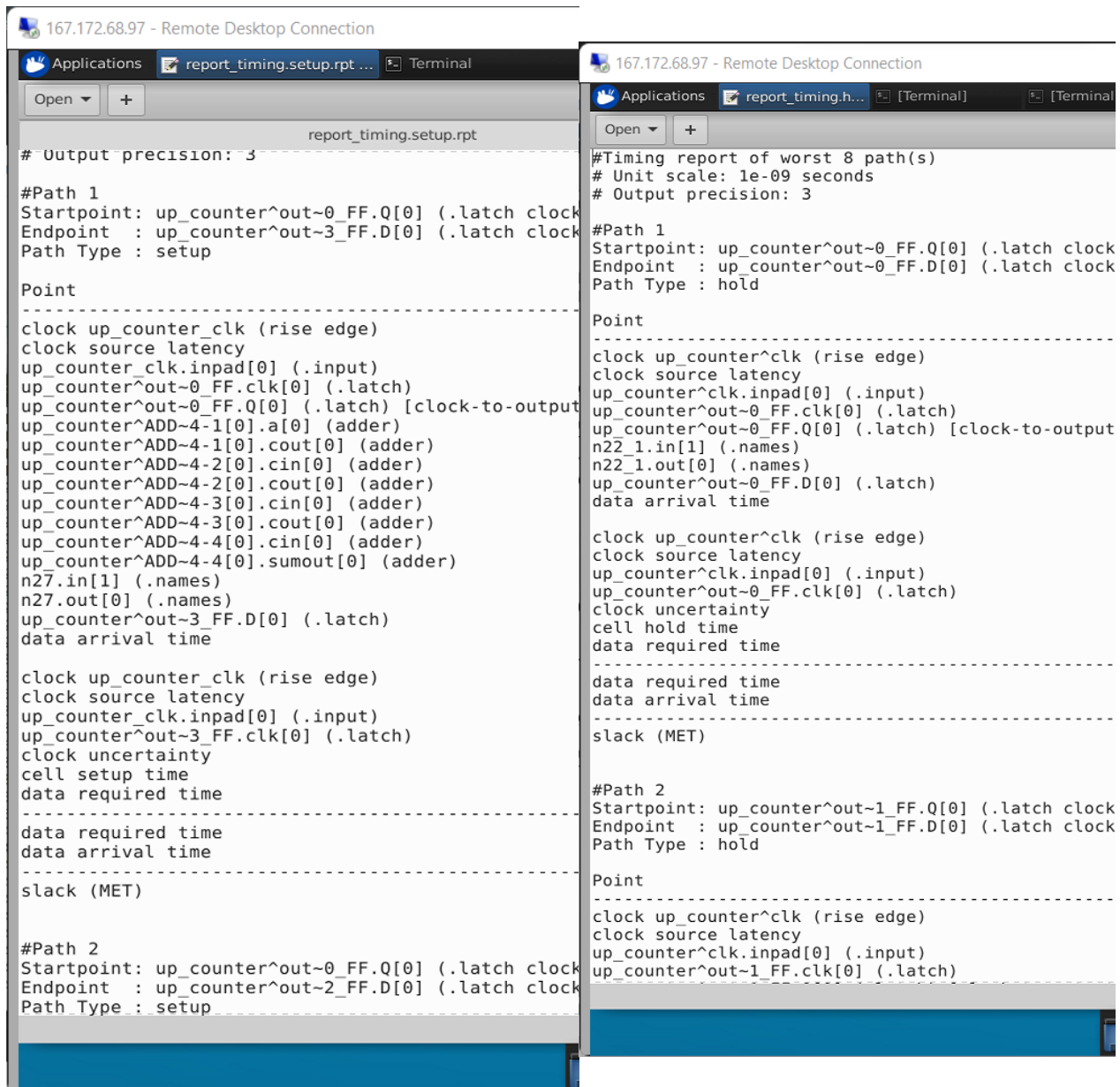
In order to perform timing analysis, a constraint file needs to be created. This constraint file is provided as an input to tool. To perform timing analysis from command-line, below mentioned switch should be enabled.

```

...
## .sdc constraint file is required
--sdc_file <sdc-file-path>
...

```

The snippets below show the setup and hold timing reports generated by the tool.



Post Synthesis Simulation

Post Synthesis simulation in VTR flow is same as Post Implementation simulations in general. To generate the post synthesis netlist for simulation, the below mentioned switch should be enabled in during the VPR stage in VTR flow.

```
## To generate post synthesis netlist
--gen_post_synthesis_netlist on
---
```

The image displays two side-by-side screenshots of a Verilog code editor, showing the implementation of a 4-bit up-counter in an FPGA. The code is organized into two main blocks: the counter logic and the routing segments.

Left Screenshot (up_counter_post_synthesis.v):

```

//Verilog generated by VPR 8.1.0-dev+220fa98c0 from post-place-and-route implementation
module up_counter (
    input \up_counter^enable ,
    input \up_counter^clk ,
    input \up_counter^reset ,
    output \up_counter^out-0 ,
    output \up_counter^out-1 ,
    output \up_counter^out-2 ,
    output \up_counter^out-3
);

//Wires
wire \up_counter^enable_output_0_0 ;
wire \up_counter^clk_output_0_0 ;
wire \up_counter^reset_output_0_0 ;
wire \lut_n22_output_0_0 ;
wire \lut_n27_output_0_0 ;
wire \lut_n32_output_0_0 ;
wire \lut_n37_output_0_0 ;
wire \lut_vcc_output_0_0 ;
wire \adder_up_counter^ADD-4-0[0]_output_0_0 ;
wire \adder_up_counter^ADD-4-1[0]_output_0_0 ;
wire \adder_up_counter^ADD-4-1[0]_output_1_0 ;
wire \lut_gnd_output_0_0 ;
wire \adder_up_counter^ADD-4-2[0]_output_0_0 ;
wire \adder_up_counter^ADD-4-2[0]_output_1_0 ;
wire \adder_up_counter^ADD-4-3[0]_output_0_0 ;
wire \adder_up_counter^ADD-4-3[0]_output_1_0 ;
wire \adder_up_counter^ADD-4-4[0]_output_1_0 ;
wire \latch_up_counter^out-0_FF_output_0_0 ;
wire \latch_up_counter^out-1_FF_output_0_0 ;
wire \latch_up_counter^out-2_FF_output_0_0 ;
wire \latch_up_counter^out-3_FF_output_0_0 ;
wire \lut_n22_input_0_2 ;
wire \lut_n27_input_0_4 ;
wire \lut_n37_input_0_4 ;
wire \lut_n32_input_0_4 ;
wire \latch_up_counter^out-0_FF_clock_0_0 ;
wire \latch_up_counter^out-1_FF_clock_0_0 ;
wire \latch_up_counter^out-2_FF_clock_0_0 ;
wire \lut_n22_input_0_1 ;
wire \lut_n27_input_0_2 ;
wire \lut_n37_input_0_2 ;
wire \lut_n32_input_0_4 ;

```

Right Screenshot (up_counter_post_synthesis.v):

```

fpga_interconnect \routing_segment_up_counter^clk_output_0_0_to_latch_up_counter^out-1
    .dataout(\latch_up_counter^out-3_FF_clock_0_0 )
);

fpga_interconnect \routing_segment_up_counter^clk_output_0_0_to_latch_up_counter^out-1
    .dataout(\latch_up_counter^out-1_FF_clock_0_0 )
);

fpga_interconnect \routing_segment_up_counter^clk_output_0_0_to_latch_up_counter^out-1
    .dataout(\latch_up_counter^out-2_FF_clock_0_0 )
);

fpga_interconnect \routing_segment_up_counter^reset_output_0_0_to_lut_n22_input_0_1
    .dataout(\lut_n22_input_0_1 )
);

fpga_interconnect \routing_segment_up_counter^reset_output_0_0_to_lut_n27_input_0_2
    .dataout(\lut_n27_input_0_2 )
);

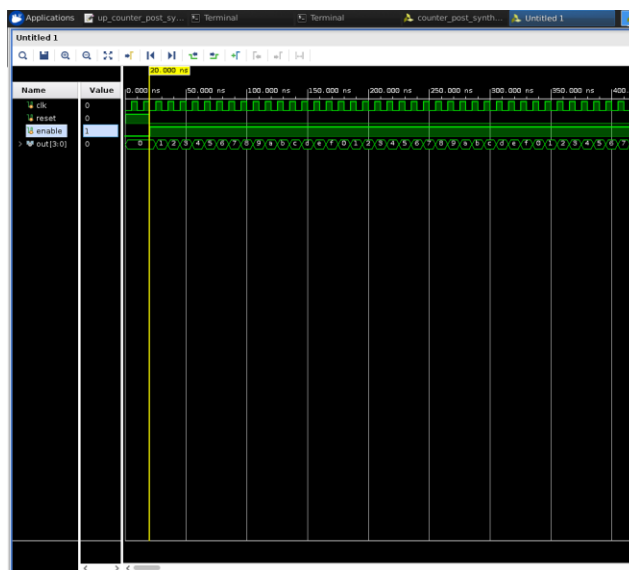
fpga_interconnect \routing_segment_up_counter^reset_output_0_0_to_lut_n37_input_0_2
    .dataout(\lut_n37_input_0_2 )
);

fpga_interconnect \routing_segment_up_counter^reset_output_0_0_to_lut_n32_input_0_4
    .dataout(\lut_n32_input_0_4 )
);

fpga_interconnect \routing_segment_lut_n22_output_0_0_to_latch_up_counter^out-0_FF_inj
    .dataout(\latch_up_counter^out-0_FF_input_0_0 )
);

fpga_interconnect \routing_segment_lut_n27_output_0_0_to_latch_up_counter^out-3_FF_inj
    .dataout(\lut_n27_output_0_0 )
);

```



VTR provides a option to perform power analysis over the design. To enable power analysis in command-line, the switch mentioned below should be used.

VTR provides a option to perform power analysis over the design. To enable power analysis in command-line, the switch mentioned below should be used.

```

```
For power analysis
```

```
-power -cmos_tech
```

```
$VTR_ROOT/vtr_flow/tech/PTM_45nm/45nm.xml
```

```

The snippet below shows a brief summary of the power analysis report generated by the VTR

flow.

```
Applications counter.power (~vtr_w... Terminal Terminal
Open + counter.power
~/vtr_work/quickstart/counter_e
45nm.xml x
----- Summary -----
Circuit: counter
Architecture: EArch.xml
Technology (nm): 45
Voltage: 0.90
Temperature: 85
Critical Path: 1.46e-09
Size of FPGA: 3 x 3
Channel Width: 100

----- Warnings -----
No transistor counter function for BLIF model: .subckt adder
No dynamic power defined for BLIF model: .subckt adder
No leakage power defined for BLIF model: .subckt adder

----- Power Breakdown -----
Component Power (W) %-Total %-Dynamic Me
Total 0.0003212 1 0.7689
Routing 0.0001066 0.3319 0.4223
Switch Box 9.697e-05 0.3019 0.3837
Connection Box 5.033e-06 0.01567 0.6368
Global Wires 4.617e-06 0.01437 1
PB Types 8.287e-05 0.258 0.8582
Primitives 6.158e-05 0.1917 0.9314
Interc Structures 7.248e-06 0.02256 0.6487
Buffers and Wires 1.404e-05 0.0437 0.6451
Other Estimation Methods 0 0 -nan
Clock 0.0001317 0.4101 0.9932

----- Power Breakdown by PB -----
This sections provides a detailed breakdown of power usage by PB (phys
block). For each PB, the power is listed, which is the sum power of al
instances of the block. It also indicates its percentage of total pow
FPGA), as well as the percentage of its power that is dynamic (vs. sta
also indicates the method used for power estimation.

The data includes:
Modes: When a pb contains multiple modes, each mode i
its power statistics.
Bufs/Wires: Power of all local buffers and local wire swit
(transistor-level estimation only).
Interc: Power of local interconnect multiplexers (tran
level estimation only)
```


Day 3 - RISC-V Core Programming Using Vivado

A 4-stage pipelined RISC-V core, named RVMYTH, is used in the repository. A complete RTL to Bitstream flow is implemented over the RVMYTH core. The Core is initially developed in High-level language named TL-Verilog and finally compiled to Verilog HDL.

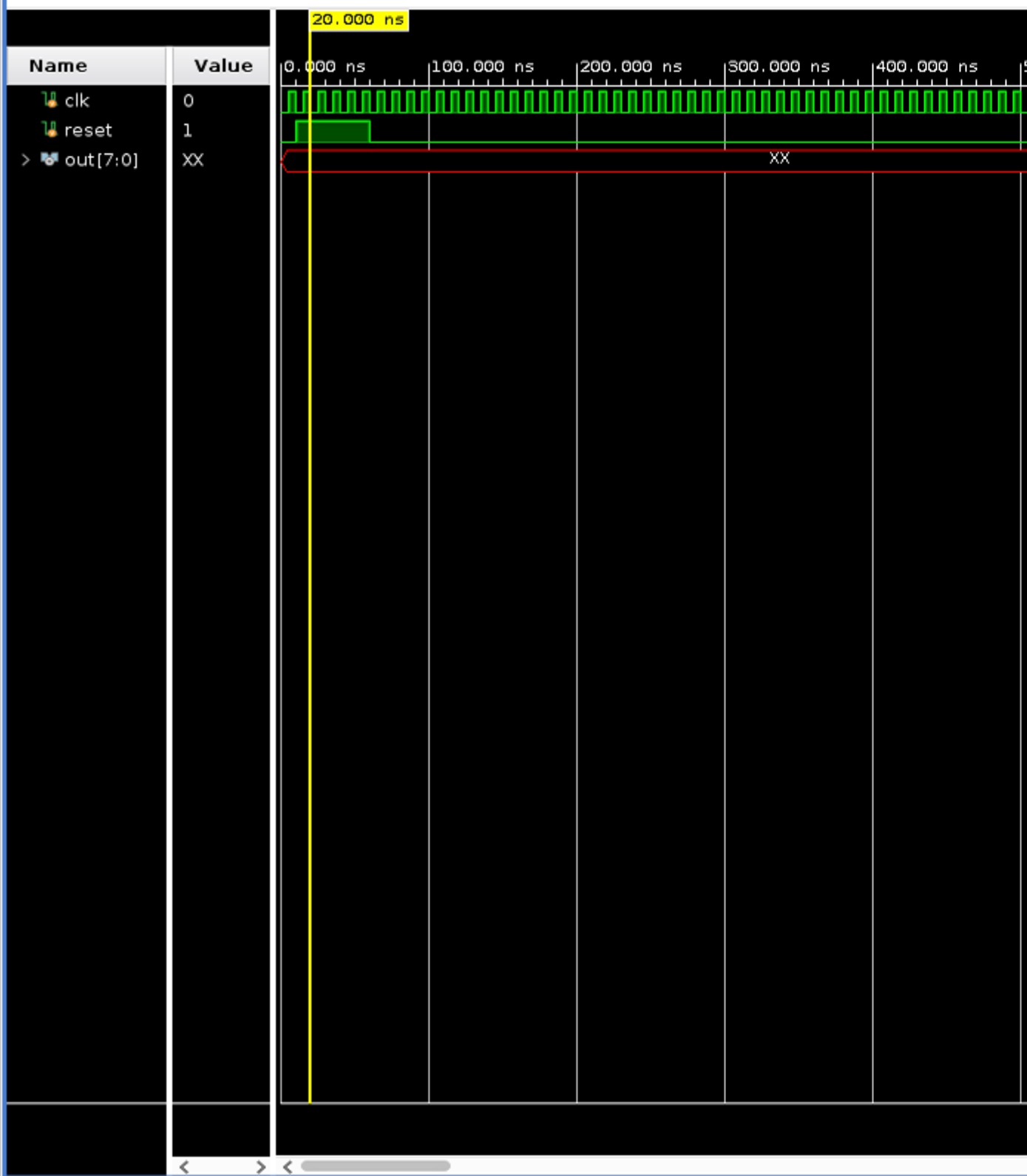
RTL To Synthesis

The RISC-V RTL consists of multiple blocks/modules. Some of them are:

- Instruction memory
- Data memory
- ALU
- I/O ports

The snippets below shows the behavioural simulation of the RVMYTH RISC-V core in vivado simulator. The instruction memory contains instructions for addition of integers from 1 to 9. The output signal in the waves display the final sum.

Untitled 1

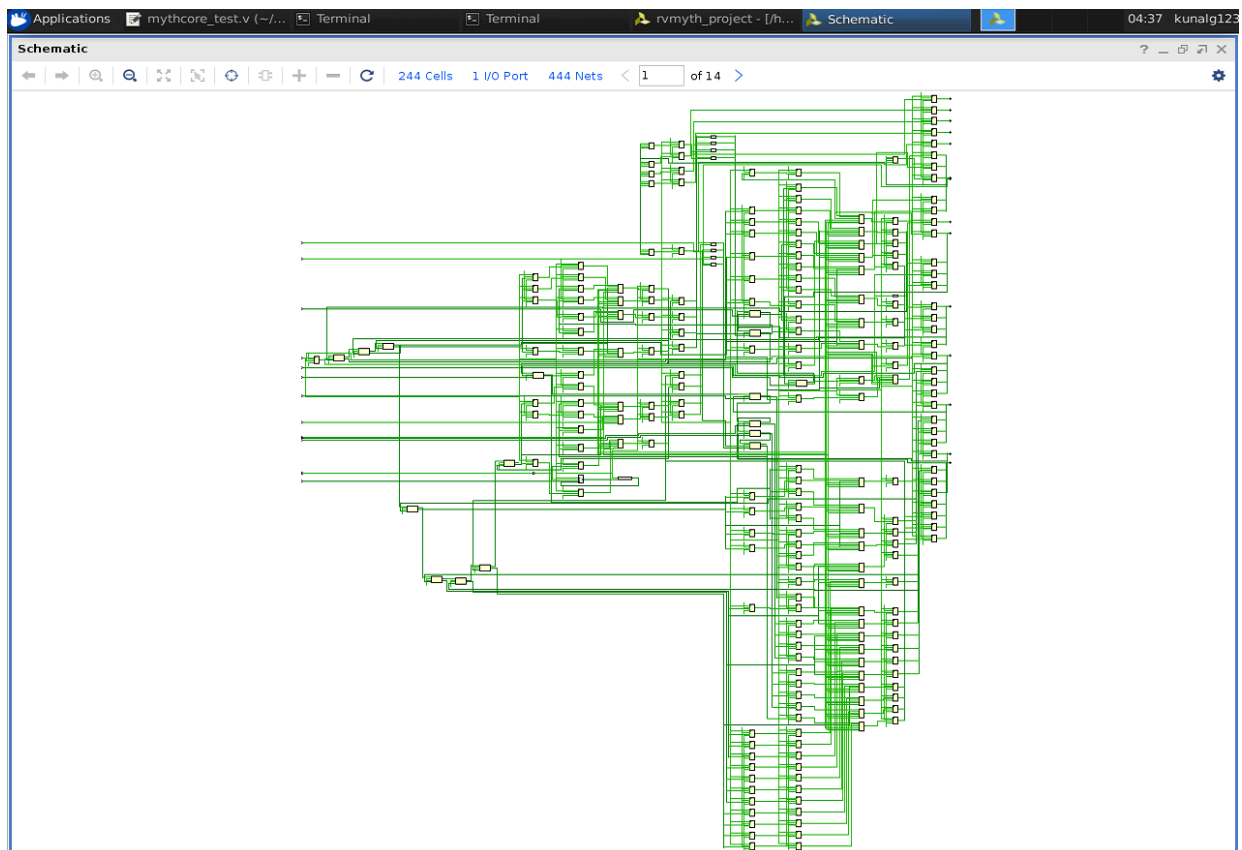


The snippet below shows the elaboration stage where the FPGA pins are mapped to the RTL input/output ports.

The screenshot shows the Vivado IDE interface during the elaboration stage. The top window, titled 'ELABORATED DESIGN - xc7a35tcpg236-1', displays the 'Device Constraints' tab. It shows a tree view of constraints, including 'Internal VREF' with voltage options (0.6V, 0.675V, 0.75V, 0.9V) and 'NONE (4)' with I/O Bank settings (I/O Bank 14, I/O Bank 16, I/O Bank 34, I/O Bank 35). A note at the bottom states: 'Drop I/O banks on voltages or the "NONE" folder to set/unset Internal VREF.' The right window shows a package pin map for 'mythcore_test_no_ILA.v' on a 19x19 grid. The bottom window, titled 'I/O Ports', displays a table mapping RTL ports to physical pins and standards.

Name	Direction	Package Pin	I/O Std	Vcco	Board Part Pin	Board Part Interface	Neg Diff Pair	Fix
▼ All ports (10)								
▼ out (8)	OUT		default (LVCMOS18)	1.800				
out[7]	OUT		default (LVCMOS18)	1.800				
out[6]	OUT		default (LVCMOS18)	1.800				
out[5]	OUT		default (LVCMOS18)	1.800				
out[4]	OUT		default (LVCMOS18)	1.800				
out[3]	OUT		default (LVCMOS18)	1.800				
out[2]	OUT		default (LVCMOS18)	1.800				
out[1]	OUT		default (LVCMOS18)	1.800				
out[0]	OUT		default (LVCMOS18)	1.800				
▼ Scalar ports (2)								
clk	IN	W5	LVCMOS33*	3.300				
reset	IN	R2	LVCMOS33*	3.300				

The design is synthesized in Vivado tool for Basys3 FPGA, along with some constraints. The below snippet show the schematic of design after synthesis and the constraints used for synthesis.



167.172.68.97 - Remote Desktop Connection

rvmyth_project - [home/taware.shon/fpga_workshop/vivado_projects/rvmyth_project/rvmyth_project.xpr] - Vivado 2019.2

File Edit Flow Tools Reports Window Layout View Help Quick Access

Synthesis Complete

Flow Navigator

- PROJECT MANAGER
 - Settings
 - Add Sources
 - Language Templates
 - IP Catalog
- IP INTEGRATOR
 - Create Block Design
 - Open Block Design
 - Generate Block Design
- SIMULATION
 - Run Simulation
- RTL ANALYSIS
 - Open Elaborated Design
 - Report Methodology
 - Report DRC
 - Report Noise
 - Schematic
- SYNTHESIS
 - Run Synthesis
 - Open Synthesized Design
 - Constraints Wizard
 - Edit Timing Constraints
 - Set Up Debug
 - Report Timing Summary
 - Report Clock Networks
 - Report Clock Interaction
 - Report Methodology

SYNTHESIZED DESIGN - synth_1 | xc7a35tcbg236-1

Sources

- Design Sources (1)
 - core (mythcore_test_with_ILA.v) (8)
- Constraints (1)
 - constrs_1 (1)
 - constraints.xdc
- Simulation Sources (1)
 - sim_1 (1)
 - test (test.v) (1)
 - utils_1

Project Summary

Device: test.v

constraints.xdc

```

1 |set_property PACKAGE_PIN WS [get_ports clk]
2 |set_property IOSTANDARD LVCMOS33 [get_ports clk]
3 |set_property IOSTANDARD LVCMOS33 [get_ports reset]
4 |set_property PACKAGE_PIN R2 [get_ports reset]
5
6
7 |create_clock -period 10.000 -name clk -waveform {0.000 5.000} [get_ports clk]
8 |set_property C_CLK_INPUT_FREQ_HZ 300000000 [get_debug_cores dbg_hub]
9 |set_property C_ENABLE_CLK_DIVIDER false [get_debug_cores dbg_hub]
10 |set_property C_USER_SCAN_CHAIN 1 [get_debug_cores dbg_hub]
11 |connect_debug_port dbg_hub/clk [get_nets clk_IBUF_BUF]
12
13

```

Tcl Console

```

Finished Parsing XDC File [home/taware.shon/fpga_workshop/vivado_projects/rvmyth_project/rvmyth_project.srscs/sources_1/ip/ila_0/ila_0.xdc]
Parsing XDC File [home/taware.shon/fpga_workshop/vivado_projects/rvmyth_project/rvmyth_project.srscs/constrs_1/imports/Day3/constrain
Finished Parsing XDC File [home/taware.shon/fpga_workshop/vivado_projects/rvmyth_project/rvmyth_project.srscs/constrs_1/imports/Day3/
INFO: (Opt 31-158) Pushed 0 inverter(s) to 0 load pin(s).
Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00 ; Memory (MB): peak = 8124.652 ; gain = 0.000 ; free physical
INFO: (Project 1-111) Unisim Transformation Summary:
A total of 36 instances were transformed.
CFGLUTS => CFGLUTS (SRL16E, SRLC32E): 36 instances

open_run: Time (s): cpu = 00:00:08 ; elapsed = 00:00:06 ; Memory (MB): peak = 8124.652 ; gain = 0.000 ; free physical = 10040 ; free

```

Terminal Emulator

Use the command line

Synthesis To Bitstream

During implementation, the synthesis design is translated to a logic design file and mapped to small blocks and sub-blocks that can fit in FPGA CLBs. These blocks are then placed and routed in an optimized way.

The snippet below shows a small part of the implemented design, the used LUTs, MUX and CLBs.



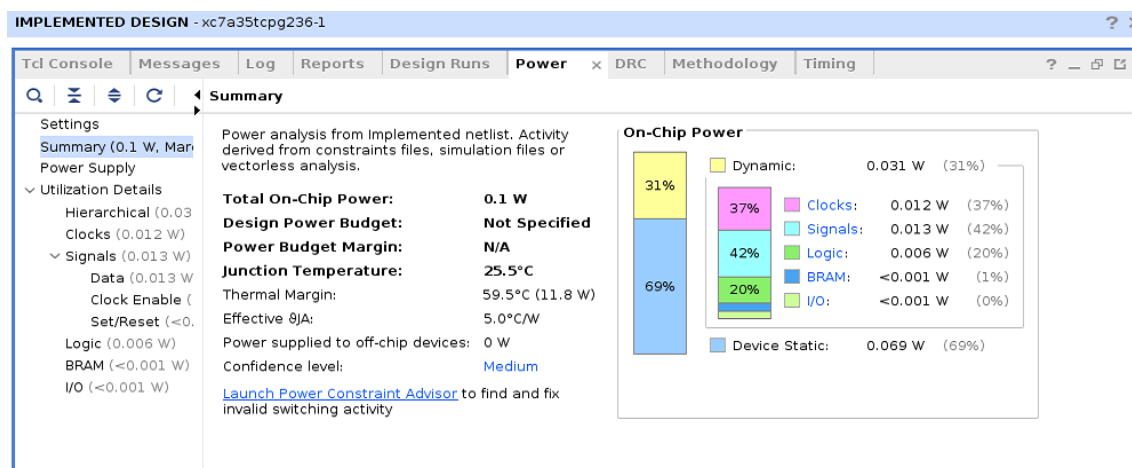
The snippet below shows the timing summary for the RVMYTH implemented core.

Tcl Console Messages Log Reports Design Runs Power DRC Methodology Timing x			
Design Timing Summary			
General Information			
Timer Settings			
Design Timing Summary			
Clock Summary (2)			
> Check Timing (1)			
> Intra-Clock Paths			
> Inter-Clock Paths			
> Other Path Groups			
User Ignored Paths			
Unconstrained Paths			
Setup			
Worst Negative Slack (WNS): 2.101 ns			
Total Negative Slack (TNS): 0.000 ns			
Number of Failing Endpoints: 0			
Total Number of Endpoints: 8525			
Hold			
Worst Hold Slack (WHS): 0.053 ns			
Total Hold Slack (THS): 0.000 ns			
Number of Failing Endpoints: 0			
Total Number of Endpoints: 8509			
Pulse Width			
Worst Pulse Width Slack (WPS): 0.000 ns			
Total Pulse Width Negative Slack (TPWS): 0.000 ns			
Number of Failing Endpoints: 0			
Total Number of Endpoints: 8509			
All user specified timing constraints are met.			

The snippet below shows the Device Utilization summary for the RVMYTH implemented core.

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Block RAM Tile (50)
core	1967	3767	260	96	1405	1872	95	0.5
dbg_hub (dbg_hub)	442	727	0	0	233	418	24	0
gen_clkP_CPU_dmem_rd_en_a5 (clk_gate)	1	1	0	0	1	1	0	0
gen_clkP_CPU_rd_valid_a2 (clk_gate_0)	2	1	0	0	3	2	0	0
gen_clkP_CPU_rd_valid_a3 (clk_gate_1)	1	1	0	0	1	1	0	0
gen_clkP_CPU_rd_valid_a4 (clk_gate_2)	1	1	0	0	1	1	0	0
gen_clkP_CPU_rd_valid_a5 (clk_gate_3)	1	1	0	0	2	1	0	0
gen_clkP_CPU_rs1_valid_a2 (clk_gate_4)	1	1	0	0	2	1	0	0
gen_clkP_CPU_rs2_valid_a2 (clk_gate_5)	2	1	0	0	2	2	0	0
your_instance_name (ila_0)	613	1080	4	0	339	543	70	0.5

The snippet below shows the power analysis for the RVMYTH implemented core.



Day 4 - Introduction To SOFA FPGA Fabric

SOFA (Skywater Opensource FPGAs) are a series of open-source FPGA IPs designed using open-source Skywater 130nm PDKs and OpenFPGA framework.

The FPGA IP design used in this repository is FPGA1212_QLSOFA_HD_PNR which has 50MHz of maximum operating speed, 1152 LUTs, 2304 Flip-flops, 1152 soft adders. The complete design is used over OpenFPGA framework and the various reports are generated.

SOFA Counter Area

The snippet below shows the utilization of the counter in SOFA FPGA Fabric.

```

Circuit Statistics:
  Blocks: 17
    .input :      3
    .latch :      4
    .output:      4
    4-LUT :      6
  Nets : 13
    Avg Fanout:    2.5
    Max Fanout:    5.0
    Min Fanout:    1.0
  Netlist Clocks: 1
# Build Timing Graph
  Timing Graph Nodes: 45
  Timing Graph Edges: 60
  Timing Graph Levels: 8
# Build Timing Graph took 0.00 seconds (max_rss 16.7 MiB, delta_rss +0.0 MiB)
Netlist contains 1 clocks
  Netlist Clock 'clk' Fanout: 4 pins (8.9%), 4 blocks (23.5%)

```

SOFA Counter Timing

The snippet below shows the setup and hold timing summary the counter in SOFA FPGA Fabric.

```

report_timing.setup.rpt... Terminal
report_timing.setup.rpt
~/fpga_workshop/clones/SOFA/FPGA1212_Q..._f_arch/up_counter/MIN_ROUTE_CHAN_WIDTH
generate_testbench.openfpga x report_timing.setup.rpt x
#Timing report of worst 8 path(s)
# Unit scale: 1e-09 seconds
# Output precision: 3

#Path 1
Startpoint: out[0].Q[0] (.latch clocked by clk)
Endpoint : out[3].D[0] (.latch clocked by clk)
Path Type : setup

Point                               Incr      Path
-----
clock clk (rise edge)                0.000      0.000
clock source latency                  0.000      0.000
clk.inpad[0] (.input)                 0.000      0.000
out[0].clk[0] (.latch)                0.110      0.110
out[0].Q[0] (.latch) [clock-to-output] 0.430      0.540
$abc$159$new_n13_in[3] (.names)       9.470     10.010
$abc$159$new_n13_out[0] (.names)      1.210     11.220
n28.in[2] (.names)                    4.280     15.500
n28.out[0] (.names)                   0.920     16.420
out[3].D[0] (.latch)                  0.000     16.420
data arrival time                     16.420

clock clk (rise edge)                20.000     20.000
clock source latency                  0.000     20.000
clk.inpad[0] (.input)                 0.000     20.000
out[3].clk[0] (.latch)                0.110     20.110
clock uncertainty                     0.000     20.110
cell setup time                       -0.390     19.720
data required time                    19.720

data required time                    19.720
data arrival time                     -16.420
-----
slack (MET)                           3.300

#Path 2
Startpoint: out[0].Q[0] (.latch clocked by clk)
Endpoint : out[2].D[0] (.latch clocked by clk)
Path Type : setup

Point                               Incr      Path
-----
clock clk (rise edge)                0.000      0.000

```

References

- VLSI System Design: <https://www.vlsisystemdesign.com/ip/>
- RISC-V based Microprocessor: <https://github.com/shivanishah269/risc-v-core>
- 4-stage RISC-V Core: https://github.com/ShonTaware/RISC-V_Core_4_Stage
- SOFA: <https://github.com/lnis-uofu/SOFA>
- OpenFPGA: <https://openfpga.readthedocs.io/en/master/>
- VPR: <https://docs.verilogtorouting.org/en/latest/vpr/>
- VTR: <https://docs.verilogtorouting.org/en/latest/>

Acknowledgement

- [Kunal Ghosh](#), Co-founder, VSD Corp. Pvt. Ltd.
- [Nanditha Rao](#)

