Use of Cadence RC and SoC Encounter to Implement a Configurable-Width SIMD FPU

Adam Rushby
Wolfson School of Mechanical,
Electrical and Manufacturing Engineering
Loughborough University
Loughbrough, Leicstershire, UK
Email: a.rushby-14@student.lboro.ac.uk

***Abstract*—This paper presents a method of automation for the design synthesis and implementation process of an IEEE 754-compliant, single instruction, multiple data floating point unit. The synthesis process is carried out using Cadence RC, and the implementation process is carried out using Cadence's SoC Encounter. The design files of the SIMD FPU are modified to introduce an increasing number of FPU lanes that the design is mapped against. The RC script used to execute Cadence RC is then modified to adjust the timing constraints on the design and the level of effort for design optimisation. The SIMD FPU design is compiled and synthesised of using the RC script and then implemented using an Encounter script. Automation is carried out by a BASH script which executes multiple iterations of the process while changing design parameters periodically.**

I. Introduction

The objective of this paper is to explain and detail the creation of an automation script used to automate both front-end synthesis, and back-end implementation Place & Route processes for a given set of Very High Speed Integrated Circuit (VHSIC), Hardware Description Language(VHDL) .vhd design files. The front-end process makes use of the Cadence Register Transfer Level (RTL) Compiler (RC) and the back-end process makes use of Cadence's System on Chip (SoC) Encounter. The .vhd files contain the design for a Single Instruction, Multiple Data (SIMD) Floating point Unit (FPU).

The first step in the work flow is to compile the SIMD FPU design files using Cadence RC. This is achieved using an .rc script. This tool creates an .rc.sdf file required for the place and route step. Once the compiler is finished, the next step is to implement the design using SoC Encounter. This is done using an .enc script. The .enc scipt import the FPU design using the .rc.sdf file and maps the design to a grid. Both steps create log files which contain the timing results and the design area. This data is collected and stored to a .csv file along with the corresponding number of lanes, clock timings and the compiler effort for the design.

II. SIMD processing

The Configurable nature of the SIMD FPU design exploits data-level parallelism through the use of parallel SIMD lanes [1]. Each SIMD lane is a clone of the of FPU design. Each FPU within the compiled design, executes the same instruction, for its given inputs, as every other lane. This design architecture is important for applications where large amounts of data need to be operated on using the same data operation. Instead of having one lane handle each and every operation, the data is shared amongst multiple lanes and each operation is applied to the data in parallel. Because each lanes execute the same operation, only one instruction call is required to saturate the lanes with data, and only one instruction is need to operate on the data stored in the lanes. This effectively saves processing time where originally multiple copies of the same set of instructions would called for one lane to operate on the data.

III. Design Environment

The design environment used is a remote Linux server running the Bourne Again Shell (BASH). The server is accessed using the Xming X Window System Server [2] and the PuTTY Secure Socket Shell (SHH) client [3]. Once connected to the server, a GNOME terminal is opened using the `gnome-terminal` command, allowing access to a Graphical User Interface. The `cd` command is used to access the simd_fpu project directory containing the necessary design files and scripts needed to execute the front-end and back-end processes manually

Automation within the BASH environment is done using a BASH shell script stored within the scripts folder of the simd_fpu project directory.

IV. Synthesis Environment

The Cadence RC tool is used to compile and synthesise RTL design files. This compilation process is executed using an .rc script. This .rc script contains fourteen steps that the compiler goes through to produce the files required for the place and route process. Those fourteen steps are:

- Specify Search Paths
- Specify Synthesis Library Estimator Mode
- Specify Physical Layout
- Specify LEF Files
- Specify .cap File

- Import RTL
- Clock Gating Phase 1
- Elaborate Top Level
- Clock Gating Phase 2
- Retiming
- Constraints
- Verify Constraints
- Synthesis Proper
- Generating Reports
- Export Output to Downstream Flows

The Specify search paths step sets the directories for the library, script and RTL files.

```
set_attribute lib_search_path {. libs/ lef/
     ram/}
set_attribute script_search { scripts/}
set_attribute hdl_search_path {rtl/}
```

The Specify synthesis library step sets the desired library that the compiler will use. For the SIMD FPU design, slow.lib is called which targets the worst case library.

```
set_attribute library {slow.lib
    ra2sh_256W_32B_8MX_offWRMSK_8WRGRAN
    _slow_syn.lib}
```

The Specify Physical Layout Estimator (PLE) Mode step enables the physical optimisation mode for the design target. This is used as it is the best mode for communicating between Library Exchange Format (.lef) files and the synthesis tools.

```
set_attribute interconnect_mode ple
```

The Specify LEF Files step sets the file location for any necessary .lef files required for design synthesis.

```
set_attribute lef library {tsmc13_hs_8lm.
    lef tsmc13_hs_8lm_antenna.lef
    ra2sh_256W_32B_8MX_offWRMSK_8WRGRAN.
    vclef
    ra2sh_256W_32B_8MX_offWRMSK_8WRGRAN.ant
    .lef}
```

The Specify .cap Files step is used to set a capacitance file for design synthesis. For the SIMD FPU design the is no .cap file. The Import RTL step reads the RTL of the design using the vhdl parameter.

```
read_hdl -vhdl {simd_fpu_pkg.vhd dbgif.vhd
    lane.vhd simd_fpu.vhd}
read_hdl -vhdl datapath.vhd
```

Phase 1 of the Clock Gating step sets the clock-gating and operand isolation attributes as true which will replace the muxe's in the 'enable' flops with clock-gating cells and prevents data paths from being driven when inactive.

```
set_attribute lp_insert_clock_gating true
set_attribute lp_insert_operand_isolation
    true
```

This reduction in toggling helps to reduce the dynamic power of the design. The Elaborate Top Level step builds a internal representation of the design's structure used for mapping generic and technology gates.

```
elaborate simd_fpu
```

Phase 2 of the Clock Gating step specifies the minimum and maximum number of flops used for inserting clock-gating cells.

```
set_attribute lp_clock_gating_max_flops 16
    simd_fpu
set_attribute lp_clock_gating_min_flops 8
    simd_fpu
```

The Retiming step allows the tool to rearrange the placement of registers to achieve the targeted cycle time for a minimal area.

```
set_attribute retime true simd_fpu
```

The constraints step defines the timings for the clock and the input/output delays.

```
define_clock -name clk -p 10000 clk
external_delay -input 5000.0 -clock clk
    addr[*]
external_delay -input 5000.0 -clock clk
    wrdata[*]
external_delay -input 5000.0 -clock clk rd
external_delay -input 5000.0 -clock clk wr
external_delay -output 500.0 -clock clk
    rddata[*]
```

```
set_attribute max_fanout 10 simd_fpu
```

The verify Constraints step generates reports for the timing constraints applied in the previous step.

```
report timing -endpoint
report timing -lint
report clocks
report ple
```

The Synthesis Proper step sets the effort level for the synthesis tool to one of three options: low, medium and high. The default option is medium effort.

```
synthesize -to_mapped -effort low
```

The Generating Report step sets the type of information reports to be generated by the tool.

```
report timing
report design_rules
report gates
report nets
report nets
report summary
report power
```

Lastly the Export Output to Downstream Flows step write the design and constraints to files that are needed for the place and route process.

```
write_design -encounter
write_hdl -mapped > simd_fpu.rc.v
write_sdc > simd_fpu.rc.sdc
write_sdf -timescale ps -design simd_fpu >
    simd_fpu.rc.sdf
```

V. Place & Route

The Cadence SoC Encounter place & route tool is used to implement the design. The place & route process is executed using an .enc script. This script contains sixteen steps that the tool goes through to produce the final design file. These steps are:

- Design Import
- Initialize Floor Plan
- Power Planning
- Complete Power Planning with Placed Cells
- Standard Cell Placement
- Clock Tree Synthesis
- Route Design
- Timing Analysis
- Area Report
- Add Filler Cells
- Add Metal Fill and Connect to VDD
- verification: Metal Density
- verification: Geometry
- verification: DRC
- Verification: Connectivity
- Verification: Antenna Effects

The first step is to import the design, tech, library, LEF and constraint files into the tool [6]. The Initialize Floor plan step sets the size of the floor plan and utilisation of the floor. The Power Planning step creates a power ring followed by power stripes to create a power distribution network for the cells in the design. The complete power planning and standard cell placement steps place the design cells onto the floor and connect them to the power stripes. The clock tree synthesis step specifies a file containing the clock specifications needed to specify clock signal names, maximum required delay, minimum required delay, maximum clock skew etc.

The route design step performs the nanorouting task which executes several iterations of routing with fast RTLC extraction and timing analysis to improve the timing slack. The Area report step generates a summary report file containing the area used by individual cells, IO information, floorplan/placement information, wire length distribution, and area of power net distribution.

VI. Campaign Automation

Process automation is handled using a BASH shell script stored in the scripts directory named auto.sh. The work flow used for the automation script is shown in Fig.1. The script contains commands to edit the simd_fpu_pkg.vhd RTL design file and also edit the simd_fpu.rc script used to execute the Cadence RC process. the .rc and .enc scripts used for the synthesis and implementation tools, read data from reports generated by the tools, store the data into the res.csv file. The files used to gather the results are deleted to allow for new reports to be generated under the exact same file name and extension.

The command to execute the auto.sh script requires an input variable which defines the number of loop iterations that the script will execute. Each iteration covers the 10ns, 5ns, 2.5ns timing requirements, and low and high effort levels for each set of lane values. The script starts at 2 lanes and doubles with each iteration passed. For the input:

```
sh -f auto.sh 4
```

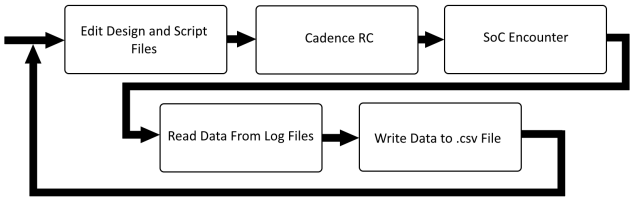

Fig. 1. Overview of Automation Script Workflow.

The script will run for 4 loop iterations, covering lane values of 2, 4, 8, and 16.

The script firstly initialises any variables needed for the automation process and creates a res.csv file with the necessary headings at the top of the file. If the .csv file already exists then the file is overwritten. A while loop is entered containing the main body of the script. the total number of loops is controlled by an incrementing counter, which exists when the value is equal to the input variable.

The design and script files are edited using the `sed` command which searches for a term within the file and replaces it with a new term [4]. The number of lanes is defined by editing the `simd_fpu_package.vhd` package file. the `sed` command searches for the line containing the definition for the number of lanes and edits only the text following after, as shown below.

```
sed -i -e "s/\(LANES : integer := \).*/\1
    $init_lanes;/" rtl/simd_fpu_pkg.vhd
```

The synthesis effort levels are modified by searching for the part of the command which sets the effort level within the `simd_fpu.rc` script file and replacing the original value with either high or low, as shown below.

```
sed -i "s/-effort low/-effort high/g"
    scripts/simd_fpu.rc
```

The timing constraints for the synthesis of the FPU design is modified in two steps: the first step is to set the clock definition. This value is set to either 10ns, 5ns, or 2.5ns, depending on the stage of the script, and is stored in pico seconds. The second step is to set the external input delays which are equal to the clock time divided by 2.

```
clock_time=$(echo "$clock_time *
    divmul_const" | bc)
prev_time=$input_time
input_time=$(echo "clock_time /
    divmul_const" | bc)

sed -i -e "s/\(define_clock -name clk -p \)
    .*/\1$clock_time clk/" scripts/simd_fpu
    .rc
sed -i "s/$prev_time.0/$input_time/g"
```

The automation script first synthesises the FPU design for a clock of 2.5ns and increases this value by multiplying the clock by 2, up to 10ns. Once synthesis and implementation is complete for 10ns, the clock value is replaced by 2.5ns and the cycle repeats. For each set of timing constraints, the synthesis script runs for high level effort, and then low level effort before changing the timing constraints to a new value. Overall, for each lane value, the number of synthesis/implementation sequences ran before the while loop iterates is 6.

The Cadence RC synthesis script is executed using the following command:

```
rc -f scripts/simd_fpu.rc
```

The encounter script for Cadence's SoC Encounter is executed using the following command.

```
encounter -64 -init scripts/simd_fpu.enc
```

Once the Synthesis and implementation process is complete, multiple lines of data is fetched from log files generated by the tools using the `grep` command [5]. Data collected is stored in an intermediate res.txt file and then the `awk` command is used to fetch specific pieces of data from the lines stored in the res.txt file. data collected using the `awk` command is then stored the res.csv file. An image of the contents of the res.csv file from a complete automation campaign is displayed in Fig.2.

```
grep "Total area of chip" summaryReport.rpt
     > res.txt
var1=`awk '{print $5}' res.txt`
```

The last action the script takes is to delete the log files generated by the tools. This is done due to restrictions imposed on the automation script preventing the use of `ls -t` command to do a directory wide search for a files with the extension .log*. The `-t` parameter filters the results based on the time they were created which, when combined with the

	A	B	C	D	E	F	G	H	I	J	K	L
1	Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8	Column9	Column10	Column11	Column12
2	Lanes				RC Effort low					RC Effort high		
3		Treq	Freq	T(RC)	Area(RC)	T(Enc)	Area(Enc)	Freq	T(RC)	Area(RC)	T(Enc)	Area(Enc)
4	2	2.5	400	+0ps	884403	0.014	1202062.003	400	+0ps	869151	0.016	1187595.445
5	2	5.0	200		881503	0.034	1192056.823	200		852090	0.017	1163219.098
6	2	10.0	100		855507	0.024	1164884.717	100		841668	0.031	1153309.838
7	4	2.5	400	+3ps	1758301	0.026	2337004.008	400	+0ps	1725631	0.017	2308010.610
8	4	5.0	200		1758760	0.076	2325966.670	200		1694186	0.037	2262059.528
9	4	10.0	100		1702128	0.033	2265965.394	100		1674194	0.046	2242507.046
10	8	2.5	400	+0ps	3505140	0.126	4583964.701	400	+0ps	3448013	0.064	4534599.292
11	8	5.0	200		3485128	0.182	4532412.024	200		3383387	0.030	4441386.877
12	8	10.0	100		3396544	0.204	4446837.268	100		3338524	0.026	4399471.670
13	16	2.5	400		7051710		8923842.096	400		6862493	0.084	8923842.096
14	16	5.0	200		6983309	0.125	8982284.036	200		6756968	0.222	8766227.412
15	16	10.0	100		6780078	0.036	8774380.068	100		6667277	0.067	8683342.722

Fig. 2. Contents of the res.csv file showing the intended layout

trailing `| head -1` parameter, would result in the newest file created with the .log* returned as the result. However, because this action cannot be performed from the script, instead of searching for the newest created file amongst many files, the `rm` command is used to make sure that any new files created are the only files present with the matching file extension.

```
rm rc.log
rm encounter.log
rm rc.cmd
rm encounter.logv
rm encounter.cmd
```

This way the script can guarantee that any new files created have a particular file extension, which while not ideal, works for this application of automation as key data points from those files are being stored to a separate file.

VII. Results

Overall, twenty four different configurations of the SIMD FPU were synthesised and implemented: twelve high effort and twelve low effort configurations. Contained within both sets are four different SIMD lane configurations: three sets for two SIMD lanes, three sets for four SIMD lanes, three sets for eight SIMD lanes, and three sets for sixteen SIMD lanes. These configurations are the different combinations of clock timings: 10ns, 5ns and 2.5ns. The recorded area and timing results for high level compiler effort are displayed in table I. Results for low level compiler effort are shown in table II.

Figs. 3 and 4 shows a comparison between the RC effort levels, the number of lanes and clock timings, and the impact it has on the design area. As the number of lanes present increase by 100%, the area of the design also increases by roughly 100%. This shows a positive relation between the number of lanes and the area of the design. For different levels effort, the charts show that when target a low effort level, compiler produces area values that are marginally higher than when a high level of compiler effort is targeted. As the clock timings for the design is increased from 2.5ns to 10ns, it can be seen that the area of the design decreases be a slight amount. As the timing constraints on the design are relaxed, the design can optimised and validated for a smaller design area.

TABLE I
Timing and Area Results for High Effort Compilation

Lanes	T Req (ns)	Freq (MHz)	RC High Effort			
			T RC (ps)	Area RC	T Enc (ns)	Area Enc
2	2.5	400	0	869151	0.016	1187595.445
	5.0	200	-	852090	0.017	1163219.098
	10.0	100	-	841669	0.031	1153309.838
4	2.5	400	0	1725631	0.017	2308010.610
	5.0	200	-	1694186	0.037	2262059.528
	10.0	100	-	1674194	0.046	2242507.046
8	2.5	400	0	3448013	0.064	4534599.292
	5.0	200	-	3383387	0.030	4441386.877
	10.0	100	-	3338524	0.026	4399471.670
16	2.5	400	-	6862493	0.084	8923842.096
	5.0	200	-	6756968	0.222	8766227.412
	10.0	100	-	6667277	0.067	8683342.722

TABLE II
Timing and Area Results for Low Effort Compilation

Lanes	T Req (ns)	Freq (MHz)	RC Low Effort			
			T RC (ps)	Area RC	T Enc (ns)	Area Enc
2	2.5	400	0	884403	0.014	1202062.003
	5.0	200	-	881503	0.034	1192056.823
	10.0	100	-	855507	0.024	1164884.717
4	2.5	400	3	1758301	0.026	2337004.008
	5.0	200	-	1758760	0.076	2325966.670
	10.0	100	-	1702128	0.033	2265965.394
8	2.5	400	0	3505140	0.126	4565965.394
	5.0	200	-	3485128	0.182	4532412.024
	10.0	100	-	3396544	0.204	4446837.268
16	2.5	400	-	7051710	-	8923842.096
	5.0	200	-	6983309	0.125	8982284.0.36
	10.0	100	-	6780078	0.036	8774380.068

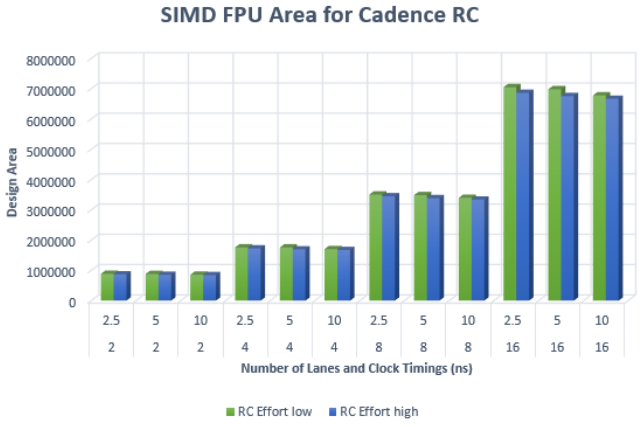

Fig. 3. Bar Chart Comparison of RC Area Results

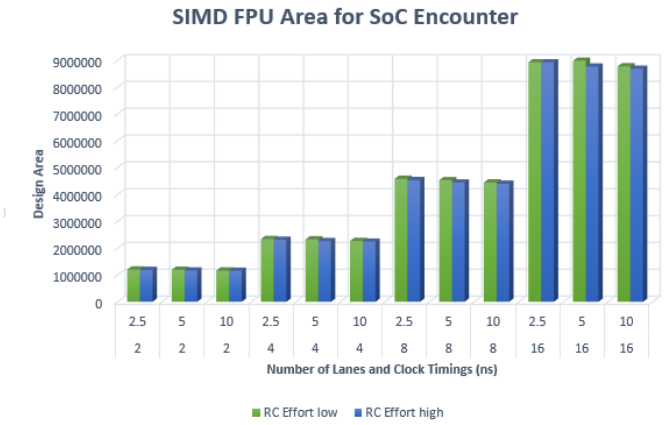

Fig. 4. Bar Chart Comparison of Enc Area Results

Out of all 24 data sets, the automation script only managed to fetch RC timing values for 6 configurations, each of which were configured for a clock speed of 2.5ns. This result could be due to the method used to extract data from the files produced by the RC tool and requires further investigation.

VIII. Conclusion

Due to the large number of configurations, the automation script to upwards of around sixteen hours to complete. This factor alone demonstrates the importance of process automation for large scale repetitive tasks executed in a UNIX/Linux environment.

For Future work, changes need to be made to the automation script to improve the consistency when reading data from log and report files generated by the Cadence tools.

References

[1] V.A. Chouliaras, V.M. Dwyer, S. Agha, J.L. Nunez-Yanez, D. Reisis, K. Nakos and K. Manolopoulos, “Customization of an embedded RISC CPU with SIMD extensions for video encoding: A case study”, Elsevier Press Integration: The VLSI Journal

[2] Xming X Server, http://www.straightrunning.com/XmingNotes/

[3] PuTTY FAQ, https://www.chiark.greenend.org.uk/ sgtatham/putty/faq.html#faq-what

[4] Advanced Bash-Scripting Guide: Appendix C. A Sed and Awk Micro-Primer, https://www.tldp.org/LDP/abs/html/x23170.html

[5] Advanced Bash-Scripting Guide: Chapter 16. External Filters, Programs and Commands, https://www.tldp.org/LDP/abs/html/textproc.html#GREPREF

[6] Place and Route with SoC Encounter, Vlsicad.ucsd.edu, https://vlsicad.ucsd.edu/courses/ece260b-w04/Lab2/

Appendix A
Automation Script

```
#!/bin/bash
#
#############################
declare -i init_time
declare -i init_lanes
declare -i divmul_const
declare -i clock_time
declare -i input_time
declare -i new_lanes
declare -i prev_time
declare -i counter
#############################
iterations=$1
init_lanes=2 ## variable input
init_time=2500 ##time in pico
    seconds
divmul_const=2
varhigh=high
counter=0
#############################
echo "Lanes                    RC
    Effort low
                RC Effort high" >
    res.csv
echo " Treq Freq  T(RC) Area(RC)
        T(Enc) Area(Enc)  Freq  T(
    RC) Area(RC)    T(Enc) Area(
    Enc)" >> res.csv
#############################
rm rc.log
rm encounter.log
rm rc.cmd
rm encounter.logv
rm encounter.cmd
#############################
# loop content
while [ $counter -lt $1 ]; do

clock_time=$init_time
input_time=$(echo "$clock_time /
    $divmul_const" | bc)

echo $init_lanes
echo $clock_time
echo $input_time
echo $prev_time
#########################
# edit files for high effort,
sed -i "s/-effort low/-effort high
    /g" scripts/simd_fpu.rc
sed -i -e "s/\(LANES : integer :=
    \).*/\1$init_lanes;/" rtl/
    simd_fpu_pkg.vhd
sed -i -e "s/\(define_clock -name
    clk -p \).*/\1$clock_time clk
    /" scripts/simd_fpu.rc
sed -i "s/5000.0/$input_time.0/g"
    scripts/simd_fpu.rc
#########################
# execute synthesis and
    implementation
rc -f scripts/simd_fpu.rc
encounter -64 -init scripts/
    simd_fpu.enc
#########################
# get values for high effort
grep "Total area of Chip"
    summaryReport.rpt > res.txt
var1=`awk '{print $5}' res.txt`
echo "$var1"

grep "WNS (ns):" encounter.log |
    tail -2 > res.txt
var4=`awk '{print $10}' res.txt`
echo "$var4" > res.txt
```

```
var6=$( tail -2 res.txt | head -1)
var7=$( tail -1 res.txt | head -1)
echo "$var6"
echo "$var7"

grep -A2 "Instance Cells Cell Area
     Net Area Total Area " rc.log
    > res.txt
var5=`awk '{print $5}' res.txt`
echo "$var5" > res.txt
var8=$( tail -1 res.txt | head -1)
echo "$var8"

grep -A2 " Slack

    Endpoint

    Cost Group" rc.log > res.txt
var9=`awk '{print $1}' res.txt`
echo "$var9" > res.txt
var10=$( tail -1 res.txt | head
    -1)
echo "$var10"
#########################
#delete files created from process
rm rc.log
rm encounter.log
rm rc.cmd
rm encounter.logv
rm encounter.cmd
#########################
#edit file for low effort
sed -i "s/-effort high/-effort low
    /g" scripts/simd_fpu.rc
#########################
#repeat synthesis implementation
    process
rc -f scripts/simd_fpu.rc
encounter -64 -init scripts/
    simd_fpu.enc
#########################
# get values for low effort
grep "Total area of Chip"
    summaryReport.rpt > res.txt
var2=`awk '{print $5}' res.txt`
echo "$var2"

grep "WNS (ns):" encounter.log |
    tail -2 > res.txt
var4=`awk '{print $10}' res.txt`
echo "$var4" > res.txt
var6=$( tail -2 res.txt | head -1)
var3=$( tail -1 res.txt | head -1)
echo "$var6"
echo "$var3"

grep -A2 "Instance Cells Cell Area
     Net Area Total Area " rc.log
    > res.txt
var5=`awk '{print $5}' res.txt`
echo "$var5" > res.txt
var11=$( tail -1 res.txt | head
    -1)
echo "$var11"

grep -A2 " Slack

    Endpoint

    Cost Group" rc.log > res.txt
var9=`awk '{print $1}' res.txt`
echo "$var9" > res.txt
var12=$( tail -1 res.txt | head
    -1)
echo "$var12"
#########################
#delete files
rm rc.log
rm encounter.log
rm rc.cmd
rm encounter.logv
rm encounter.cmd
#########################
#print values to .scv
echo "$init_lanes 2.5   400
    $var12 $var11 $var3 $var2 400
    $var10 $var8 $var7 $var1" >>
    res.csv
#########################
#########################
echo "########## Part 1/3 complete
     ##########"
#########################
clock_time=$(echo "$clock_time *
    $divmul_const" |bc)
prev_time=$input_time
input_time=$(echo "$clock_time /
    $divmul_const" | bc)
#
echo $init_lanes
echo $clock_time
echo $input_time
echo $prev_time
#########################
sed -i "s/-effort low/-effort high
    /g" scripts/simd_fpu.rc
sed -i -e "s/\(LANES : integer :=
    \).*/\1$init_lanes;/" rtl/
    simd_fpu_pkg.vhd
sed -i -e "s/\(define_clock -name
    clk -p \).*/\1$clock_time clk
    /" scripts/simd_fpu.rc
sed -i "s/$prev_time.0/$input_time
    .0/g" scripts/simd_fpu.rc
#########################
rc -f scripts/simd_fpu.rc
encounter -64 -init scripts/
    simd_fpu.enc
#########################
#get values for high effort
grep "Total area of Chip"
    summaryReport.rpt > res.txt
var1=`awk '{print $5}' res.txt`
echo "$var1"
#
grep "WNS (ns):" encounter.log |
    tail -2 > res.txt
var4=`awk '{print $10}' res.txt`
echo "$var4" > res.txt
var6=$( tail -2 res.txt | head -1)
var7=$( tail -1 res.txt | head -1)
echo "$var6"
echo "$var7"
#
```

```
grep -A2 "Instance Cells Cell Area
     Net Area Total Area " rc.log
    > res.txt
var5=`awk '{print $5}' res.txt`
echo "$var5" > res.txt
var8=$( tail -1 res.txt | head -1)
echo "$var8"
#
grep -A2 " Slack

    Endpoint

    Cost Group" rc.log > res.txt
var9=`awk '{print $1}' res.txt`
echo "$var9" > res.txt
var10=$( tail -1 res.txt | head
    -1)
echo "$var10"
#########################
rm rc.log
rm encounter.log
rm rc.cmd
rm encounter.logv
rm encounter.cmd
#########################
sed -i -e "s/-effort high/-effort
    low/g" scripts/simd_fpu.rc
#########################
rc -f scripts/simd_fpu.rc
encounter -64 -init scripts/
    simd_fpu.enc
#########################
# get value for low effort
grep "Total area of Chip"
    summaryReport.rpt > res.txt
var2=`awk '{print $5}' res.txt`
echo "$var2"
#
grep "WNS (ns):" encounter.log |
    tail -2 > res.txt
var4=`awk '{print $10}' res.txt`
echo "$var4" > res.txt
var6=$( tail -2 res.txt | head -1)
var3=$( tail -1 res.txt | head -1)
echo "$var6"
echo "$var3"
#
grep -A2 "Instance Cells Cell Area
     Net Area Total Area " rc.log
    > res.txt
var5=`awk '{print $5}' res.txt`
echo "$var5" > res.txt
var11=$( tail -1 res.txt | head
    -1)
echo "$var11"
#
grep -A2 " Slack

    Endpoint

    Cost Group" rc.log > res.txt
var9=`awk '{print $1}' res.txt`
echo "$var9" > res.txt
var12=$( tail -1 res.txt | head
    -1)
echo "$var12"
#########################
rm rc.log
rm encounter.log
rm rc.cmd
rm encounter.logv
rm encounter.cmd
#########################
echo "$init_lanes 5.0   200
    $var12 $var11 $var3 $var2 200
    $var10 $var8 $var7 $var1" >>
    res.csv
#########################
echo "########## Part 2/3 complete
     ##########"
#########################
clock_time=$(echo "$clock_time *
    $divmul_const" | bc)
prev_time=$input_time
input_time=$(echo "$clock_time /
    $divmul_const" | bc)
#
echo $init_lanes
echo $clock_time
echo $input_time
echo $prev_time
#########################
sed -i "s/-effort low/-effort high
    /g" scripts/simd_fpu.rc
sed -i -e "s/\(LANES : integer :=
    \).*/\1$init_lanes;/" rtl/
    simd_fpu_pkg.vhd
sed -i -e "s/\(define_clock -name
    clk -p \).*/\1$clock_time clk
    /" scripts/simd_fpu.rc
sed -i "s/$prev_time.0/$input_time
    .0/g" scripts/simd_fpu.rc
#########################
rc -f scripts/simd_fpu.rc
encounter -64 -init scripts/
    simd_fpu.enc
#########################
# get values for high effort
grep "Total area of Chip"
    summaryReport.rpt > res.txt
var1=`awk '{print $5}' res.txt`
echo "$var1"
#
grep "WNS (ns):" encounter.log |
    tail -2 > res.txt
var4=`awk '{print $10}' res.txt`
echo "$var4" > res.txt
var6=$( tail -2 res.txt | head -1)
var7=$( tail -1 res.txt | head -1)
echo "$var6"
echo "$var7"
#
grep -A2 "Instance Cells Cell Area
     Net Area Total Area " rc.log
    > res.txt
var5=`awk '{print $5}' res.txt`
echo "$var5" > res.txt
var8=$( tail -1 res.txt | head -1)
echo "$var8"
#
grep -A2 " Slack

    Endpoint
```

```
    Cost Group" rc.log > res.txt
var9=`awk '{print $1}' res.txt`
echo "$var9" > res.txt
var10=$( tail -1 res.txt | head
    -1)
echo "$var10"
#########################
rm rc.log
rm encounter.log
rm rc.cmd
rm encounter.logv
rm encounter.cmd
#########################
sed -i "s/-effort high/-effort low
    /g" scripts/simd_fpu.rc
#########################
rc -f scripts/simd_fpu.rc
encounter -64 -init scripts/
    simd_fpu.enc
#########################
# get values for low effort
grep "Total area of Chip"
    summaryReport.rpt > res.txt
var2=`awk '{print $5}' res.txt`
echo "$var2"
#
grep "WNS (ns):" encounter.log |
    tail -2 > res.txt
var4=`awk '{print $10}' res.txt`
echo "$var4" > res.txt
var6=$( tail -2 res.txt | head -1)
var3=$( tail -1 res.txt | head -1)
echo "$var6"
echo "$var3"
#
grep -A2 "Instance Cells Cell Area
     Net Area Total Area " rc.log
    > res.txt
var5=`awk '{print $5}' res.txt`
echo "$var5" > res.txt
var11=$( tail -1 res.txt | head
    -1)
echo "$var11"
#
grep -A2 " Slack

    Endpoint

    Cost Group" rc.log > res.txt
var9=`awk '{print $1}' res.txt`
echo "$var9" > res.txt
var12=$( tail -1 res.txt | head
    -1)
echo "$var12"
#########################
rm rc.log
rm encounter.log
rm rc.cmd
rm encounter.logv
rm encounter.cmd
#########################
echo "$init_lanes 10.0  100
    $var12 $var11 $var3 $var2 100
    $var10 $var8 $var7 $var1" >>
    res.csv
#########################
echo "##########\nPart 3/3
    complete\n##########"
#########################
echo "Iteration $counter Complete"
init_lanes=$(echo "$init_lanes *
    $divmul_const" | bc)
let counter=counter+1
#########################
done
#end loop content
#############################
```

APPENDIX B
SIMD_FPU SCHEMATIC