భారతీయ సాంకేతిక విజ్ఞాన సంస్థ హైదరాబాద్
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Department of Electrical Engineering

Indian Institute of Technology Hyderabad

RTL Synthesis - Execution and Reporting

EE2510 : Productivity Tools for IC Design & Technology

by

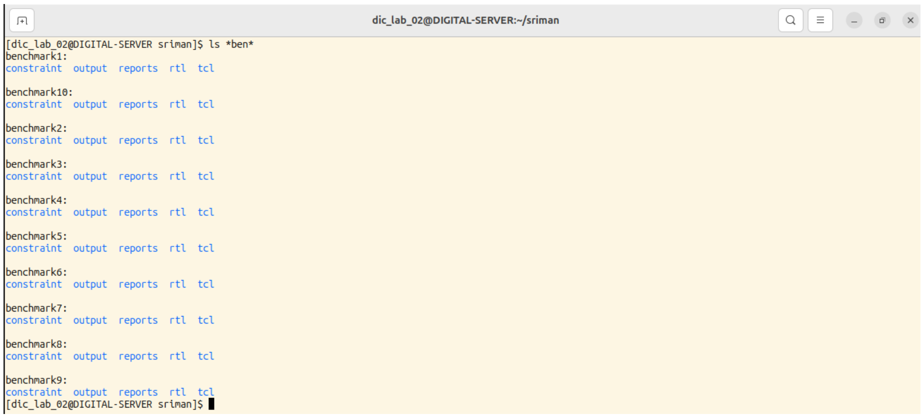
EE23BTECH11215 - Penmetsa Srikar Varma
EE23BTECH11208 - Manohar K

Approach for Automation :

We first created our own directory named **sriman** with login password sree@man

We wrote a bash script which creates folders as mentioned above in our directory

```bash
#!/bin/bash

benchmarkslist=()
for i in {1..10}; do
    benchmarkslist+=("benchmark$i")
done

subdirectories=("rtl" "tcl" "output" "reports" "constraint")

for benchmark in "${benchmarks[@]}";
 do
   mkdir -p "$benchmark"
  for sub_dir in "${sub_dirs[@]}";
    do
       mkdir -p "$benchmark/$sub_dir"
    done
done
```



```
[dic_lab_02@DIGITAL-SERVER sriman]$ ls *ben*
benchmark1:
constraint  output  reports  rtl  tcl

benchmark10:
constraint  output  reports  rtl  tcl

benchmark2:
constraint  output  reports  rtl  tcl

benchmark3:
constraint  output  reports  rtl  tcl

benchmark4:
constraint  output  reports  rtl  tcl

benchmark5:
constraint  output  reports  rtl  tcl

benchmark6:
constraint  output  reports  rtl  tcl

benchmark7:
constraint  output  reports  rtl  tcl

benchmark8:
constraint  output  reports  rtl  tcl

benchmark9:
constraint  output  reports  rtl  tcl
[dic_lab_02@DIGITAL-SERVER sriman]$ █
```

Script Explained :

Above bashscript creates 10 benchmarks folders with names benchmark1, benchmark2..so on to benchmark10

Creates sub folders ”rtl”, ”tcl”, ”reports”, ”output”, ”constraint” in each of benchmark

We then wrote a bash script which modifies constraint and tcl files from provided demo directory, creating new constraint, tcl files for each benchmark directory

```bash
#!/bin/bash

benchmarks=("benchmark7" "benchmark8" "benchmark3" "benchmark4" "benchmark5"
 "benchmark6" "benchmark1" "benchmark2" "benchmark9" "benchmark10")
top_modules=("FADD.v" "FADD_Dual_Main.v" "IIR_filter.v" "FIR_filter.v" "c6288.v"
        "b14.vhd" "b15.vhd" "UART.v" "FADD.v" "FADD_Dual_Main.v" "UART_RX.v" "UART_TX.v")

hometoaddr="/DIG_DESIGN/INTERNS/dic_lab_02/sriman"
freqs=("100MHtz" "400MHtz")
modes=("slow" "fast" "fast_hvt" "slow_hvt" "fast_lvt" "slow_lvt")

for i in "${!benchmarks[@]}";
do
rtlpath="$hometoaddr/$benchmark/rtl"
tclpath="$hometoaddr/$benchmark/tcl"
benchmark="${benchmarks[i]}"
top_module="${top_modules[i]}"
constraintpath="$hometoaddr/$benchmark/constraint"

if [[ "$top_module" == *.v ]];
then
top_module_name=$(basename "$top_module" .v)
elif [[ "$top_module" == *.vhd ]];
then
top_module_name=$(basename "$top_module" .vhd)
fi
for freq in "${freqs[@]}";
do
if [[ $freq == "100MHtz" ]];
then
clk_period="10"
clk_waveform="{0 5}"
fi
if [[ $freq == "400MHtz" ]];
then
clk_period="2.5"
clk_waveform="{0 1.25}"
fi
done

sdc_file="$constraintpath/${top_module_name}_${freq}.sdc"
{
```

```
echo "# Clock constraints"
clk_variables=$(grep -iP "input\s+(\w+\s+)?\S*(clk|clock)\S*" "$top_file" |
sed 's/;//' | tr -s ',' '\n' | tr -s '[:space:]' '\n' | grep -iP "(clk|clock)")
if [[ "$top_file" == *.vhd ]];
then
clk_variables=$(grep -iP "port\s*\(.*\s*in\s+.*(clk|clock)\S*.*\)" "$top_file" |
sed 's/port\s*//g' | tr -s ',' '\n' | tr -s '[:space:]' '\n' | grep -iP "(clk|clock)")
fi

if [[ -z "$clk_variables" ]]; then
clk_variables="clk"
fi
for clk_variable in $clk_variables;
do
echo "create_clock -name $clk_variable -period $clk_period -waveform $clk_waveform
[get_ports \"$clk_variable\"]"
echo "set_clock_transition -rise 0.1 [get_clocks \"$clk_variable\"]"
echo "set_clock_transition -fall 0.1 [get_clocks \"$clk_variable\"]"
echo "set_clock_uncertainty 0.01 [get_clocks \"$clk_variable\"]"
done

reset_variables=$(grep -iP "input\s+(\w+\s+)?\S*(rst|reset)\S*" "$top_file" |
sed 's/;//' | tr -s ',' '\n' | tr -s '[:space:]' '\n' | grep -iP "(rst|reset)")

if [[ "$top_file" == *.vhd ]];
then
reset_variables=$(grep -iP "port\s*\(.*\s*in\s+.*(rst|reset)\S*.*\)" "$top_file" | sed 's/port\s*//g'
fi

if [[ -z "$reset_variables" ]];
then
reset_variables="rst"
fi
for reset_variable in $reset_variables; do
echo "set_input_delay -max 1.0 [get_ports \"$reset_variable\"] -clock [get_clocks \"$clk_variable\"]"
done

grep -E "input\s+[^;]+;" "$top_file" | sed -E 's/input\s+|\s*;//g' | tr -s ',' '\n' | while read -r in
if [[ "$input_port" =~ \[.*\] ]]; then
echo "set_input_delay -max 1.0 [get_ports \"$input_port\"] -clock [get_clocks \"$clk_variable\"]"
else
echo "set_input_delay -max 1.0 [get_ports \"$input_port\"] -clock [get_clocks \"$clk_variable\"]"
fi
done

grep -E "output\s+[^;]+;" "$top_file" | sed -E 's/output\s+|\s*;//g' | tr -s ',' '\n' | while read -r
if [[ "$output_port" =~ \[.*\] ]]; then
echo "set_output_delay -max 1.0 [get_ports \"$output_port\"] -clock [get_clocks \"$clk_variable\"]"
else
echo "set_output_delay -max 1.0 [get_ports \"$output_port\"] -clock [get_clocks \"$clk_variable\"]"
fi
done
}        > "$sdc_file"

#TCL
for mode in "${modes[@]}"; do
    if [[ $mode == *"_lvt" ]]; then
        lib_name="${mode%%_*}_vdd1v0_basicCells_lvt.lib"
    elif [[ $mode == *"_hvt" ]]; then
        lib_name="${mode%%_*}_vdd1v0_basicCells_hvt.lib"
    else
        lib_name="${mode}_vdd1v0_basicCells.lib"
    fi
done

   tcl_path="$tclpath"
   tcl_file="$tcl_path/${top_module_name}_${freq}_${mode}.tcl"
                if [[ "$top_file" == *.v ]];
                then
                    hdl_files=$(ls "$rtlpath"/*.v 2>/dev/null | tr '\n' ' ')
                    read_hdl_cmd="read_hdl $hdl_files"
                elif [[ "$top_file" == *.vhd ]];
                 then
                    hdl_files=$(ls "$rtlpath"/*.vhd 2>/dev/null | tr '\n' ' ')
                    read_hdl_cmd="read_hdl -vhdl $hdl_files"
                fi

                cat <<EOT > "$tcl_file"
###################technology file setup

set_attribute init_lib_search_path /DIG_DESIGN/INTERNS/PDK_DIC
set_attribute init_hdl_search_path $rtlpath
set_attribute library $lib_name

$read_hdl_cmd

elaborate

read_sdc $sdc_file

set_attribute syn_generic_effort medium
set_attribute syn_map_effort medium
set_attribute syn_opt_effort medium

syn_generic
syn_map
syn_opt

######### output
write_hdl -mapped > $hometoaddr/$benchmark/output/${top_module_name}_${freq}_${mode}.v
write_sdc > $hometoaddr/$benchmark/output/${top_module_name}_${freq}_${mode}.sdc

# Write results in Report folder
report_timing > $hometoaddr/$benchmark/reports/${top_module_name}_${freq}_${mode}_timing.rpt
report_power  > $hometoaddr/$benchmark/reports/${top_module_name}_${freq}_${mode}_power.rpt
report_area   > $hometoaddr/$benchmark/reports/${top_module_name}_${freq}_${mode}_area.rpt

quit
EOT
    echo "Generated TCL script for $benchmark, freq=$freq, mode=$mode"
    done
      done
    else
 echo "RTL directory not found for running benchmark: $benchmark"
    fi
done
```

Script Explained :

Above bashscript creates the whole lot of input files for synthesis within a single run

We defined each benchmark and their corresponding top module (where constraints are applied) are defined in different arrays

We also defined frequency and mode array (where constraints and libraries are defined in generated tcl file)

It loops over each benchmark, further loops frequency and then loop libraries creating loop variables which can be used in writing constraints (.sdc) and .tcl file generation

It generates constraint (.sdc) file according to frequency, checks for .v or .vhd file and checks for input and output ports to assign maximum delay of 1ns (for each port)

Saves them in constraint folder of same benchmark

Referring to generated .sdc file, script goes to every rtl folder of benchmarks, reads *.v or *.vhd, generates .tcl file and save them in tcl folder of same benchmark

This way above bashscript works really fine for generating whole lot of input constraints and tcl files within a single run

We then used below bash script for uploading downloaded files to remote server

```bash
#!/bin/bash

SERVER="dic_lab_02@192.168.88.31"
PASSWORD="sree@man"
REMOTE_BASE_DIR="sriman"

declare -A files_to_upload=(
    ["Downloads/IIR_filter.v"]="benchmark1/rtl"
    ["Downloads/FIR_filter.v"]="benchmark2/rtl"
    ["Downloads/c6288.v"]="benchmark3/rtl"
    ["Downloads/b14.vhd"]="benchmark4/rtl"
    ["Downloads/b15.vhd"]="benchmark5/rtl"
    ["Downloads/UART.v"]="benchmark6/rtl"
    ["Downloads/align_mantisa.v"]="benchmark7/rtl"
    ["Downloads/Extract.v"]="benchmark7/rtl"
    ["Downloads/FADD.v"]="benchmark7/rtl"
    ["Downloads/Normalization.v"]="benchmark7/rtl"
    ["Downloads/Operation.v"]="benchmark7/rtl"
    ["Downloads/Result_and_exception.v"]="benchmark7/rtl"
    ["Downloads/Alignment.v"]="benchmark8/rtl"
    ["Downloads/Extraction.v"]="benchmark8/rtl"
    ["Downloads/FADD_Dual_Main.v"]="benchmark8/rtl"
    ["Downloads/Normalization_1.v"]="benchmark8/rtl"
    ["Downloads/Operation_1.v"]="benchmark8/rtl"
    ["Downloads/Pipeline_Reg.v"]="benchmark8/rtl"
    ["Downloads/Result.v"]="benchmark8/rtl"
    ["Downloads/data_sampling.v"]="benchmark9/rtl"
    ["Downloads/deserializer.v"]="benchmark9/rtl"
    ["Downloads/edge_bit_counter.v"]="benchmark9/rtl"
    ["Downloads/par_chk.v"]="benchmark9/rtl"
    ["Downloads/stp_chk.v"]="benchmark9/rtl"
    ["Downloads/strt_chk.v"]="benchmark9/rtl"
    ["Downloads/uart_rx_fsm.v"]="benchmark9/rtl"
    ["Downloads/UART_RX.v"]="benchmark9/rtl"
    ["Downloads/mux.v"]="benchmark10/rtl"
    ["Downloads/mux_1.v"]="benchmark10/rtl"
    ["Downloads/parity_calc.v"]="benchmark10/rtl"
    ["Downloads/Serializer.v"]="benchmark10/rtl"
    ["Downloads/uart_tx_fsm.v"]="benchmark10/rtl"
    ["Downloads/UART_TX.v"]="benchmark10/rtl"
)

for file in "${!files_to_upload[@]}"; do
```

```
    destination="${files_to_upload[$file]}"
    echo "Uploading $file to $SERVER:$REMOTE_BASE_DIR/$destination"
    sshpass -p "$PASSWORD" scp "$file" "$SERVER:$REMOTE_BASE_DIR/$destination"
done

echo "All files uploaded successfully!"
```

Script Explained :

Above bashscript directly uploads downloaded rtl files from local directory to remote server within single run without giving password each time we upload

We then used following bashscript to execute, generated tcl files for each benchmark

```
#!/bin/bash
benchmarks=("benchmark1" "benchmark2" "benchmark3" "benchmark4"  "benchmark5"
"benchmark6" "benchmark7" "benchmark8" "benchmark9" "benchmark10")
homedirectory="/DIG_DESIGN/INTERNS/dic_lab_02/sriman"
for benchmark in "${benchmarks[@]}"; do
    tcl_dir="$homedirectory/$benchmark/tcl"
        tcl_files=$(find "$tcl_dir" -type f -name "*.tcl")
     for tcl_file in $tcl_files; do
         echo "Running TCL script: $tcl_file"
         genus -legacy_ui -f "$tcl_file"
     done
done
```

Script Explained :

Above bashscipt will execute generated tcl files across all benchmarks within single run and we can also adjust which benchmark will be running first also

We then used following command for extracting Timing, Area and Power reports data and set output files as timing.txt, area.txt and power.txt

```
grep -r "Timing slack" > timing.txt #timing
grep -r "Subtotal" > power.txt #power
grep -r "<none>" > area.txt #area
```

Commands Explained :

By executing above commands, all lines having ”text” along with their origin of file name will be stored in a .txt file

Several operations on those generated text files would give us .csv files which are ready to plot, analysis the generated reports

We then used a bash script to extract final .csv data from .txt files and plotted graphs for pictorial representation

```bash
#!/bin/bash
input_dir="./input_files"
output_file="./extracted_lines.txt"
> "$output_file"
for file in "$input_dir"/*; do
    if [ -f "$file" ]; then  # Ensure it's a file
# here we extract the line 13(this line is repeated in every file) and append it to the output file
    sed -n '13p' "$file" >> "$output_file"
    fi
done
```

Script Explained :

Above script extracts 13th line of all generated Area reports which gives us total Area of each benchmark module and gets ready for .csv file generating

We then used following python code for extracting count for each cell type, extracting into .csv file

```python
import re
from collections import defaultdict

def count_cell_types_in_file(file_path):
    cell_counts = defaultdict(int)
    current_module = None
    with open(file_path, 'r') as file:
        for line in file:
            line = line.strip()
            module_match = re.match(r'^\s*module\s+(\w+)\s*', line)
            if module_match:
                current_module = module_match.group(1)
                continue

            if re.match(r'^\s*endmodule', line):
                current_module = None
                continue

        if current_module:
        cell_match = re.match(r"^\s*(\w+)\s+\\?[\w\[\]\d]+(?:\([\w\d,\s]+\))?\s*\(", line)
        if cell_match:
        cell_type = cell_match.group(1)
            cell_counts[cell_type] += 1

    return cell_counts

file_path = "zebra/sriman/benchmark10/output/UART_TX_100MHtz_fast.v"

total_cells = 0
for cell_type, count in sorted(count_cell_types_in_file(file_path).items()):
    print(f"  {cell_type},{count}")
    total_cells += count
```

Script Explained :

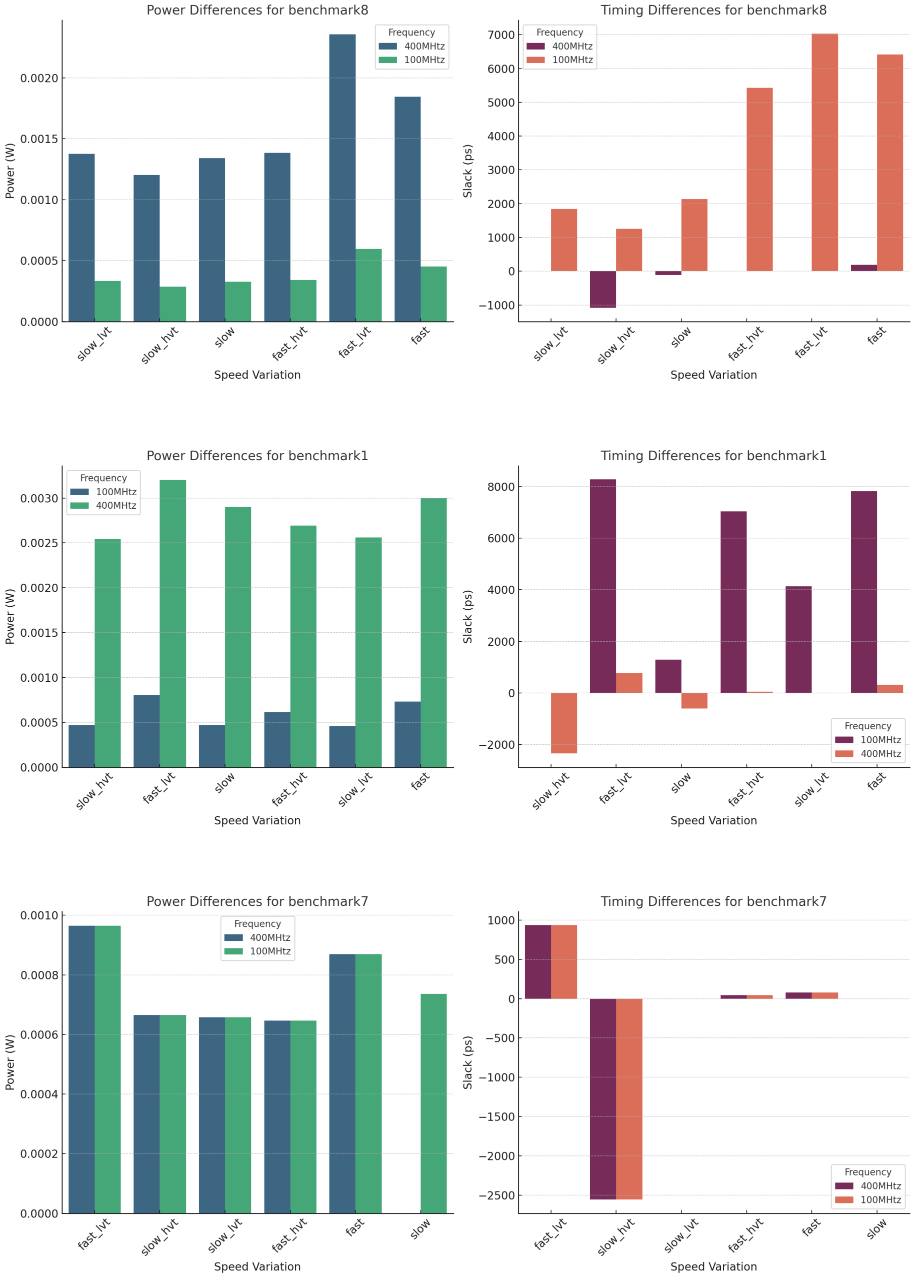
Above Python code generates csv of cell type and count, outputting to a .csv file directly generates .csv file, which is ready to plot

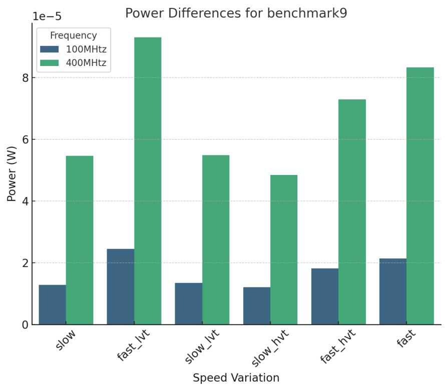
Observations on Automation :

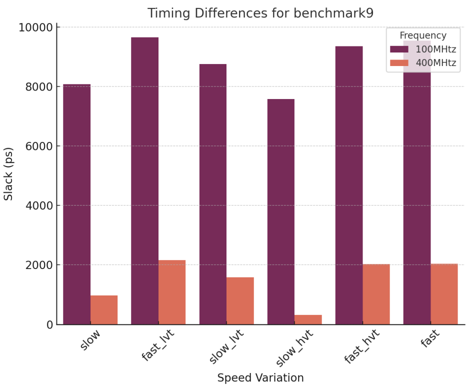
We used following python code for getting plot of Timing and Power reports

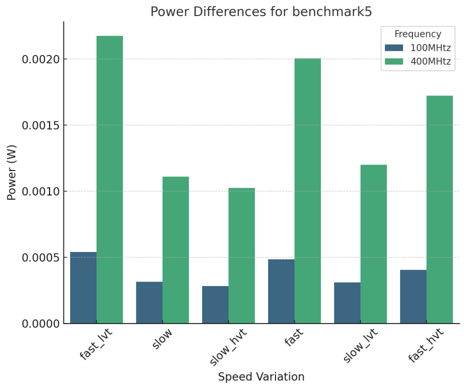
```
def plot_benchmark_differences(data):
    benchmarks = data['Benchmark'].unique()
    for benchmark in benchmarks:
        benchmark_data = data[data['Benchmark'] == benchmark]
        plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
sns.barplot(x="Speed_Variation", y="Power", hue="Frequency", data=benchmark_data,
palette="viridis")
plt.title(f"Power Differences for {benchmark}", fontsize=14)
plt.xlabel("Speed Variation")
plt.ylabel("Power (W)")
plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
sns.barplot(x="Speed_Variation", y="Slack", hue="Frequency", data=benchmark_data,
palette="rocket")
plt.title(f"Timing Differences for {benchmark}", fontsize=14)
plt.xlabel("Speed Variation")
plt.ylabel("Slack (ps)")
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```

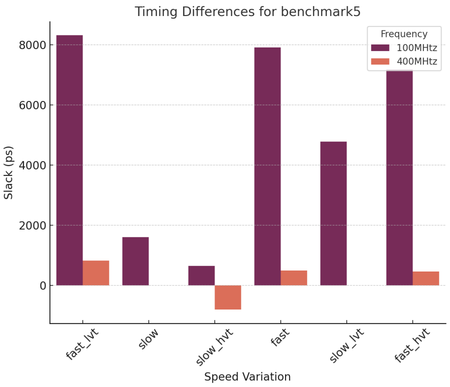
Timing and Power reports of six libraries of each benchmark as per frequency

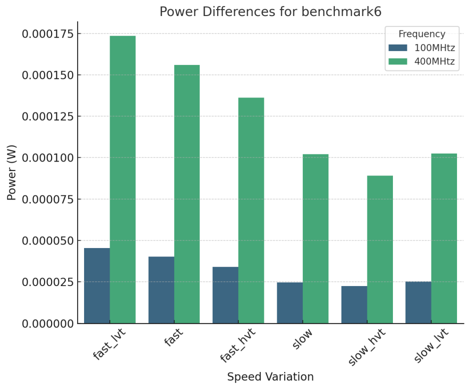


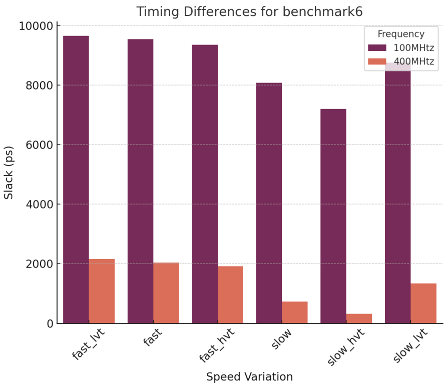

Power Differences for benchmark9
1e−5
Frequency
100MHtz
400MHtz
Power (W)
0
2
4
6
8
slow
fast_lvt
slow_lvt
slow_hvt
fast_hvt
fast
Speed Variation

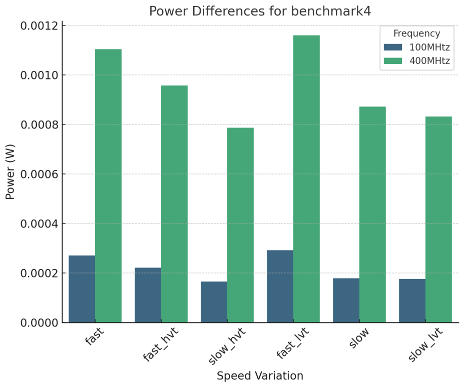

Timing Differences for benchmark9
Frequency
100MHtz
400MHtz
Slack (ps)
0
2000
4000
6000
8000
10000
slow
fast_lvt
slow_lvt
slow_hvt
fast_hvt
fast
Speed Variation

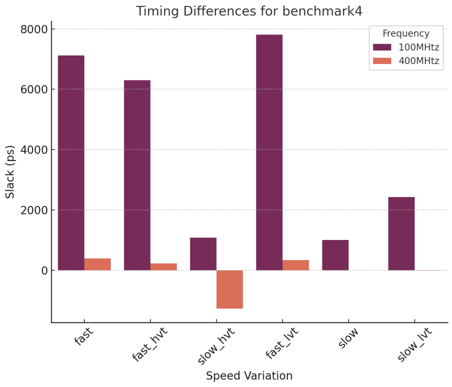

Power Differences for benchmark5
Frequency
100MHtz
400MHtz
Power (W)
0.0000
0.0005
0.0010
0.0015
0.0020
fast_lvt
slow
slow_hvt
fast
slow_lvt
fast_hvt
Speed Variation

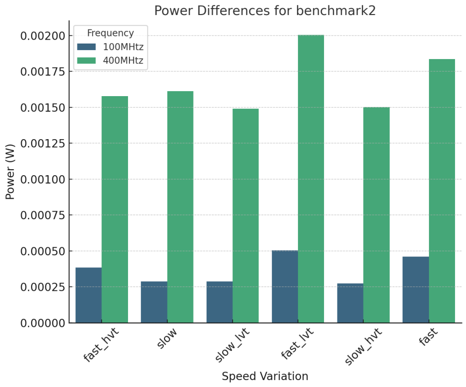

Timing Differences for benchmark5
Frequency
100MHtz
400MHtz
Slack (ps)
0
2000
4000
6000
8000
fast_lvt
slow
slow_hvt
fast
slow_lvt
fast_hvt
Speed Variation

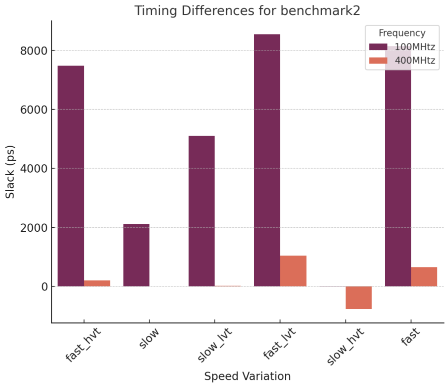

Power Differences for benchmark6
Frequency
100MHtz
400MHtz
Power (W)
0.000000
0.000025
0.000050
0.000075
0.000100
0.000125
0.000150
0.000175
fast_lvt
fast
fast_hvt
slow
slow_hvt
slow_lvt
Speed Variation

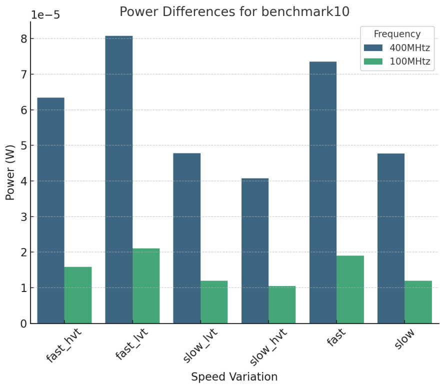

Timing Differences for benchmark6
Frequency
100MHtz
400MHtz
Slack (ps)
0
2000
4000
6000
8000
10000
fast_lvt
fast
fast_hvt
slow
slow_hvt
slow_lvt
Speed Variation

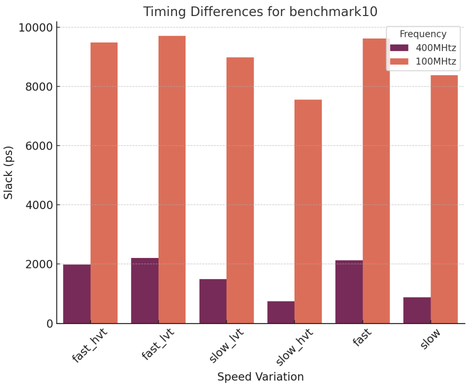

Power Differences for benchmark4
Frequency
100MHtz
400MHtz
Power (W)
0.0000
0.0002
0.0004
0.0006
0.0008
0.0010
0.0012
fast
fast_hvt
slow_hvt
fast_lvt
slow
slow_lvt
Speed Variation


Timing Differences for benchmark4
Frequency
100MHtz
400MHtz
Slack (ps)
0
2000
4000
6000
8000
fast
fast_hvt
slow_hvt
fast_lvt
slow
slow_lvt
Speed Variation


Power Differences for benchmark2
Frequency
100MHtz
400MHtz
Power (W)
0.00000
0.00025
0.00050
0.00075
0.00100
0.00125
0.00150
0.00175
0.00200
fast_hvt
slow
slow_lvt
fast_lvt
slow_hvt
fast
Speed Variation


Timing Differences for benchmark2
Frequency
100MHtz
400MHtz
Slack (ps)
0
2000
4000
6000
8000
fast_hvt
slow
slow_lvt
fast_lvt
slow_hvt
fast
Speed Variation

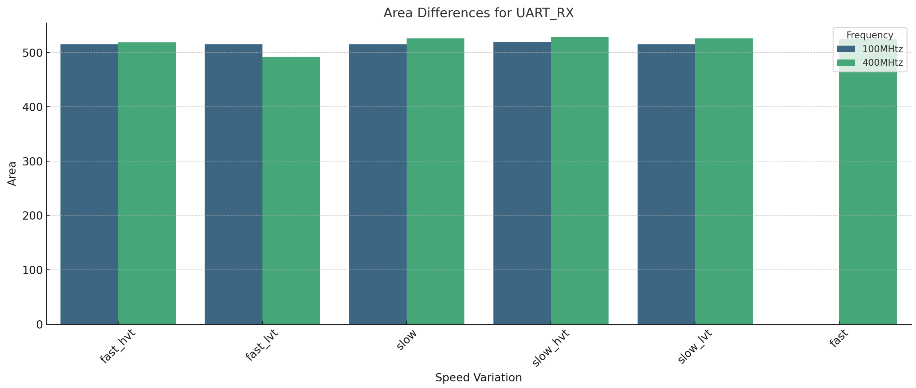

Power Differences for benchmark10
1e−5
Frequency
400MHtz
100MHtz
Power (W)
0
1
2
3
4
5
6
7
8
fast_hvt
fast_lvt
slow_lvt
slow_hvt
fast
slow
Speed Variation

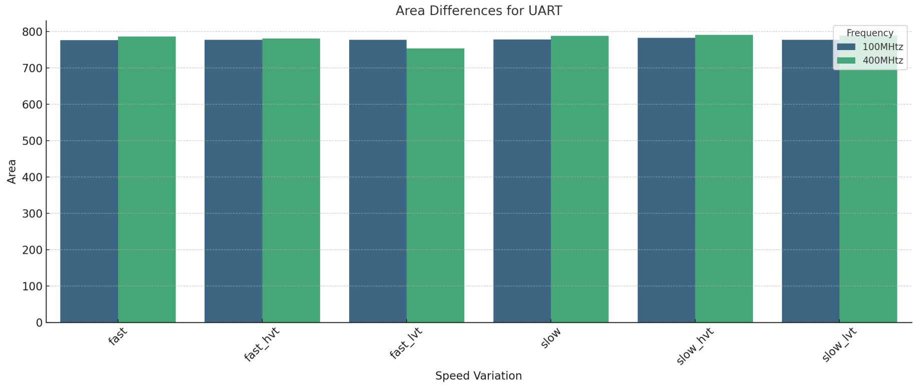

Timing Differences for benchmark10
Frequency
400MHtz
100MHtz
Slack (ps)
0
2000
4000
6000
8000
10000
fast_hvt
fast_lvt
slow_lvt
slow_hvt
fast
slow
Speed Variation

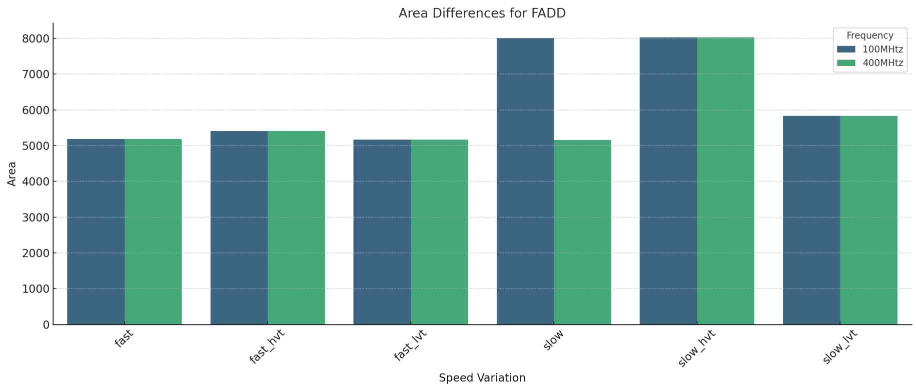
We wrote following python code for getting plot of Area reports

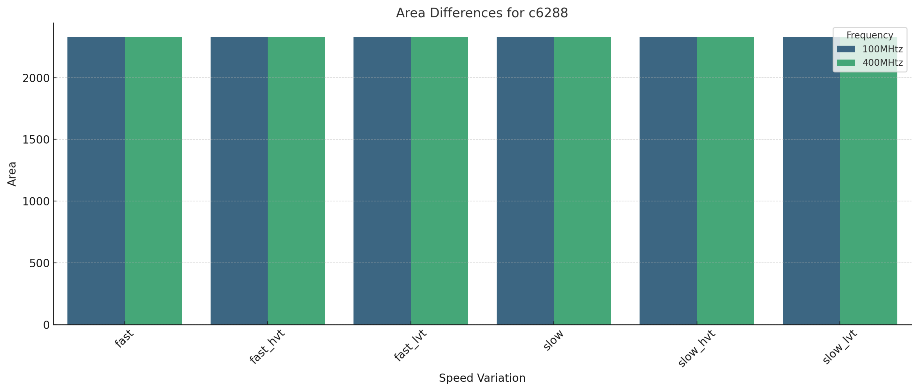
```
def plot_area_differences(data):
    modules = data['Module'].unique()
    for module in modules:
        module_data = data[data['Module'] == module]

plt.figure(figsize=(14, 6))

sns.barplot(x="Speed_Variation", y="Area", hue="Frequency", data=module_data,
palette="viridis")
plt.title(f"Area Differences for {module}", fontsize=14)
plt.xlabel("Speed Variation")
plt.ylabel("Area")
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```

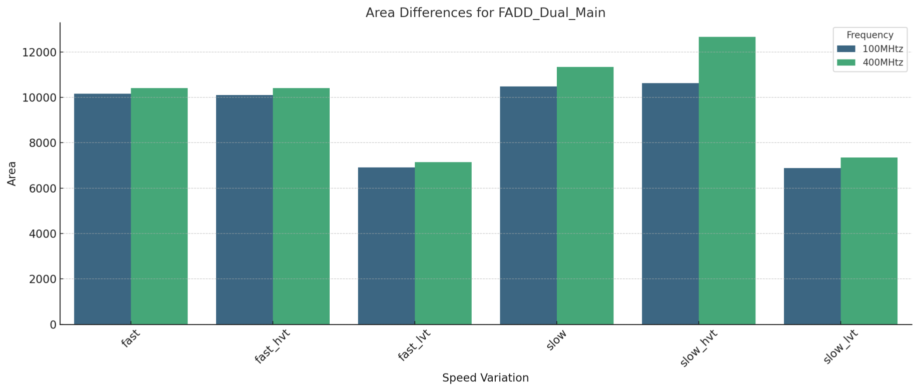
Area reports of six libraries of each benchmark as per frequency

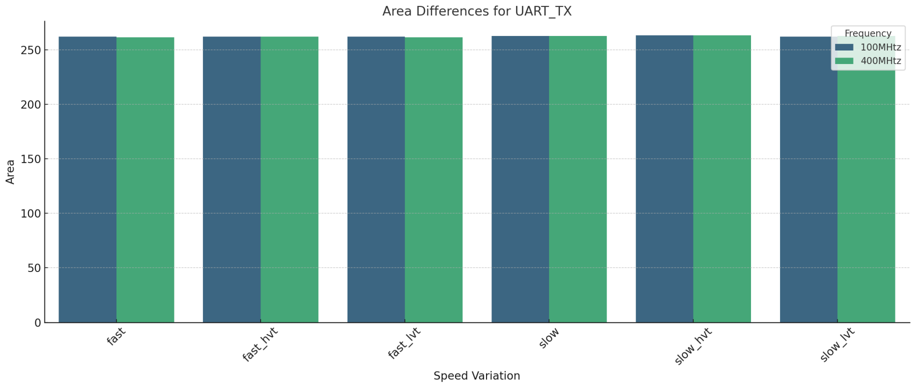


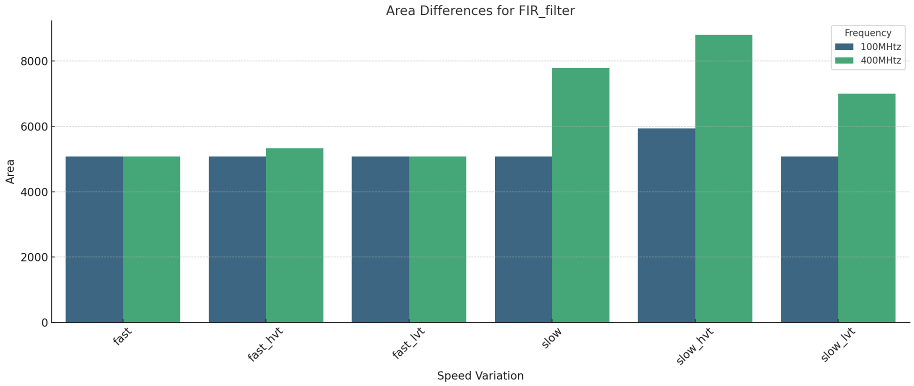


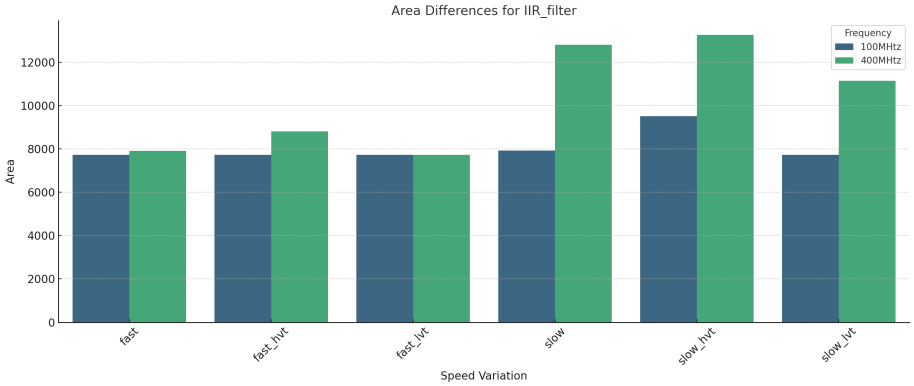

Area Differences for FADD
Area
0
1000
2000
3000
4000
5000
6000
7000
8000
Frequency
100MHtz
400MHtz
fast
fast_hvt
fast_lvt
slow
slow_hvt
slow_lvt
Speed Variation

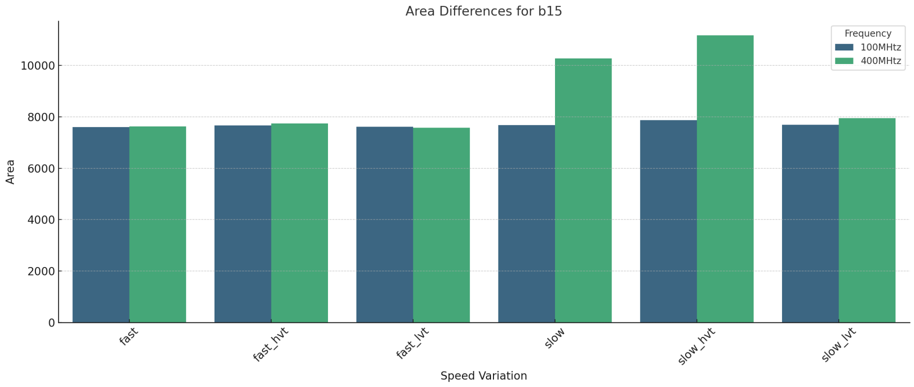

Area Differences for c6288
Area
0
500
1000
1500
2000
Frequency
100MHtz
400MHtz
fast
fast_hvt
fast_lvt
slow
slow_hvt
slow_lvt
Speed Variation

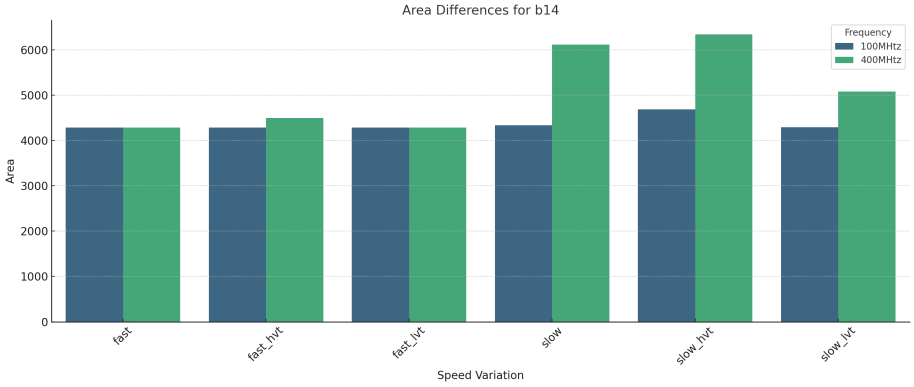

Area Differences for FADD_Dual_Main
Area
0
2000
4000
6000
8000
10000
12000
Frequency
100MHtz
400MHtz
fast
fast_hvt
fast_lvt
slow
slow_hvt
slow_lvt
Speed Variation


Area Differences for UART_TX
Area
0
50
100
150
200
250
fast
fast_hvt
fast_lvt
slow
slow_hvt
slow_lvt
Speed Variation
Frequency
100MHtz
400MHtz

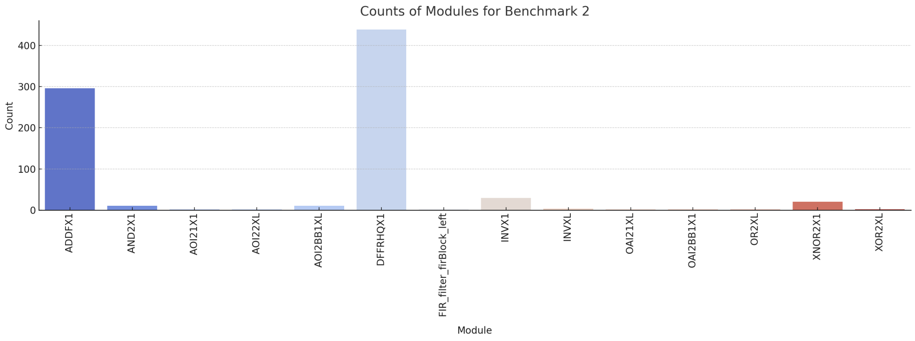

Area Differences for FIR_filter
Area
0
2000
4000
6000
8000
fast
fast_hvt
fast_lvt
slow
slow_hvt
slow_lvt
Speed Variation
Frequency
100MHtz
400MHtz

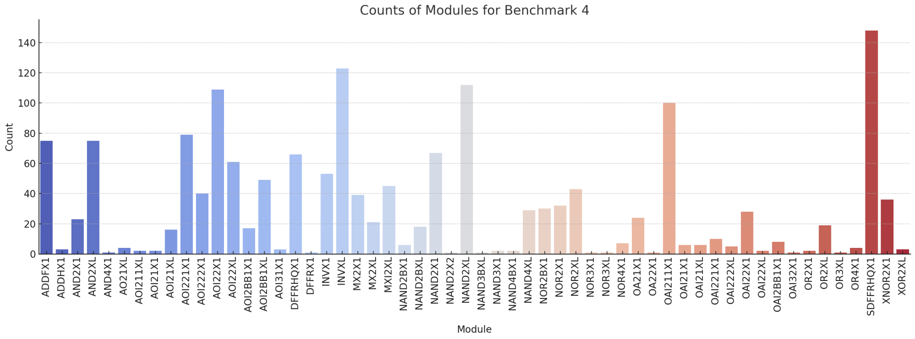

Area Differences for IIR_filter
Area
0
2000
4000
6000
8000
10000
12000
fast
fast_hvt
fast_lvt
slow
slow_hvt
slow_lvt
Speed Variation
Frequency
100MHtz
400MHtz

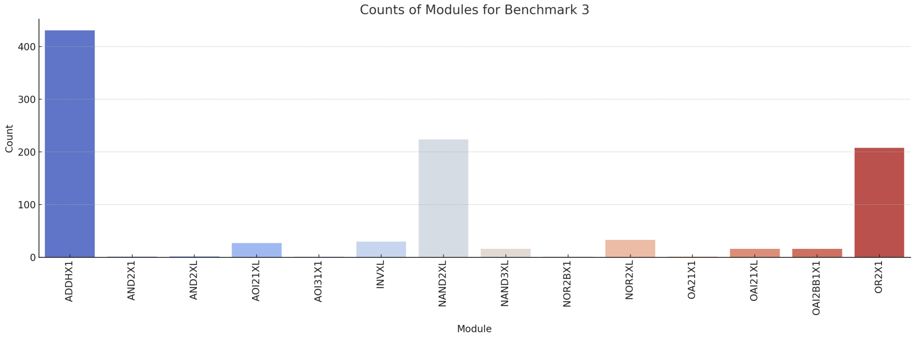


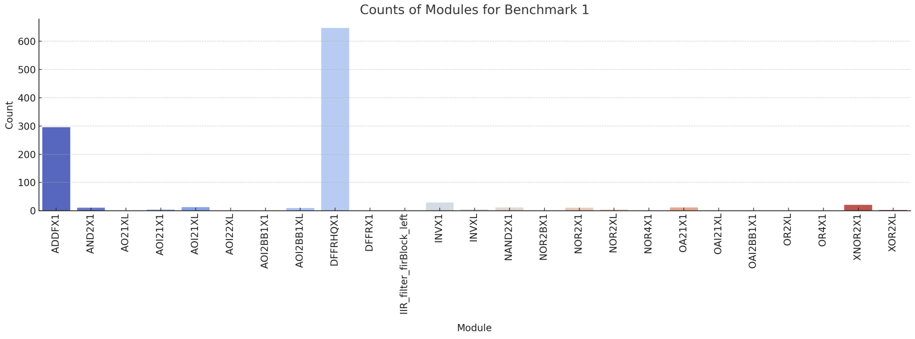


Count of each cell type of benchmarks (fast type with 100 Mega Hz)

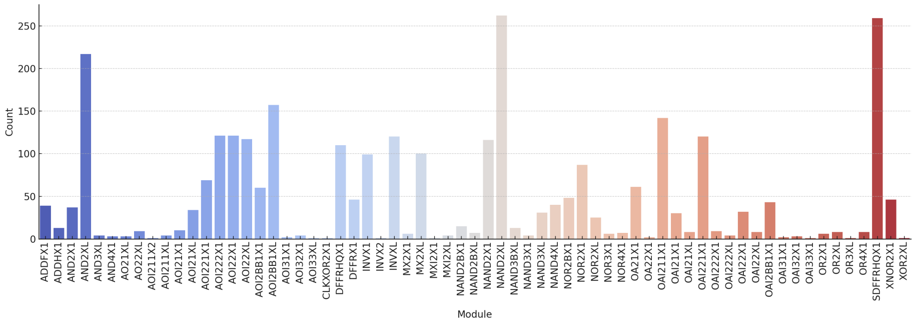



Counts of Modules for Benchmark 4
Count
0
20
40
60
80
100
120
140
Module
ADDFX1
ADDHX1
AND2X1
AND2XL
AND4X1
AO21XL
AOI211XL
AOI21X1
AOI21XL
AOI221X1
AOI222X1
AOI22X1
AOI22XL
AOI2BB1X1
AOI2BB1XL
AOI31X1
DFFRHQX1
DFFRX1
INVX1
INVXL
MX2X1
MX2XL
MXI2XL
NAND2BX1
NAND2BXL
NAND2X1
NAND2X2
NAND2XL
NAND3BXL
NAND3X1
NAND4BX1
NAND4XL
NOR2BX1
NOR2X1
NOR2XL
NOR3X1
NOR3XL
NOR4X1
OA21X1
OA22X1
OAI211X1
OAI21X1
OAI21XL
OAI221X1
OAI222XL
OAI22X1
OAI22XL
OAI2BB1X1
OAI32X1
OR2X1
OR2XL
OR3XL
OR4X1
SDFFRHQX1
XNOR2X1
XOR2XL

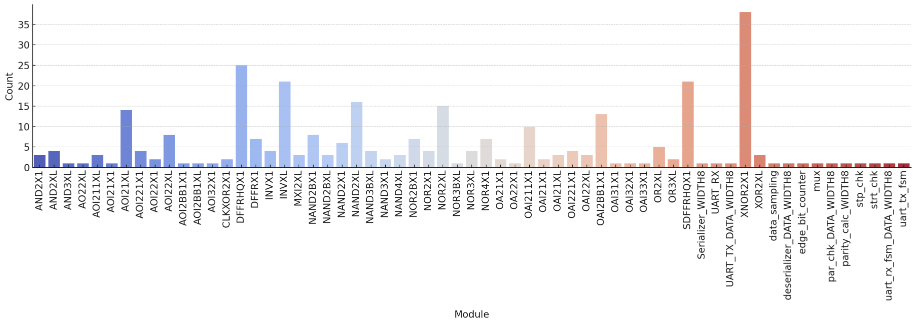

Counts of Modules for Benchmark 3
Count
0
100
200
300
400
Module
ADDHX1
AND2X1
AND2XL
AOI21XL
AOI31X1
INVXL
NAND2XL
NAND3XL
NOR2BX1
NOR2XL
OA21X1
OAI21XL
OAI2BB1X1
OR2X1


Counts of Modules for Benchmark 1
Count
0
100
200
300
400
500
600
Module
ADDFX1
AND2X1
AO21XL
AOI21X1
AOI21XL
AOI22XL
AOI2BB1X1
AOI2BB1XL
DFFRHQX1
DFFRX1
IIR_filter_firBlock_left
INVX1
INVXL
NAND2X1
NOR2BX1
NOR2X1
NOR2XL
NOR4X1
OA21X1
OAI21XL
OAI2BB1X1
OR2XL
OR4X1
XNOR2X1
XOR2XL

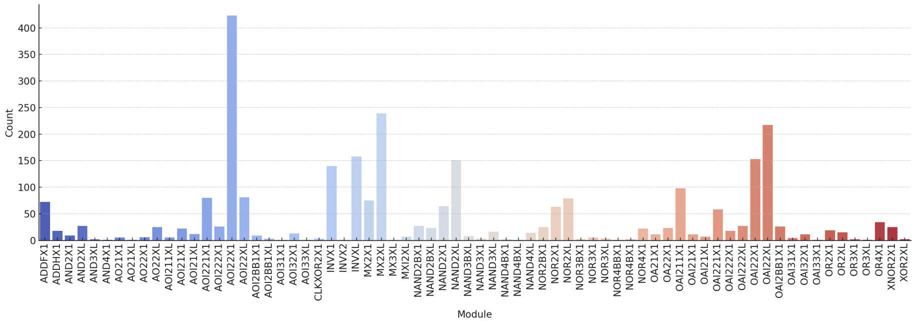
Counts of Modules for Benchmark 5

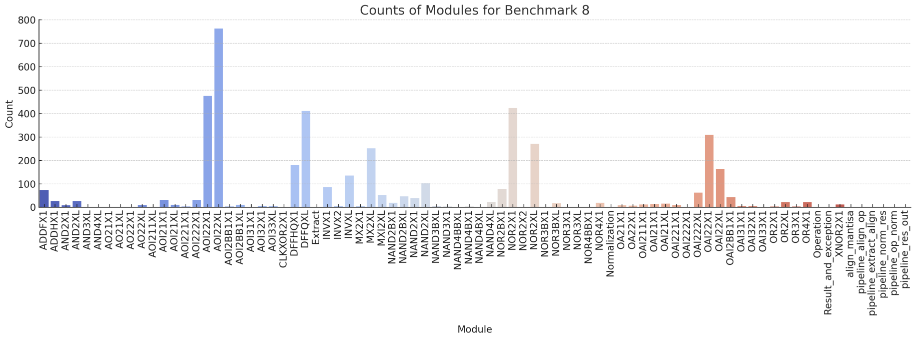


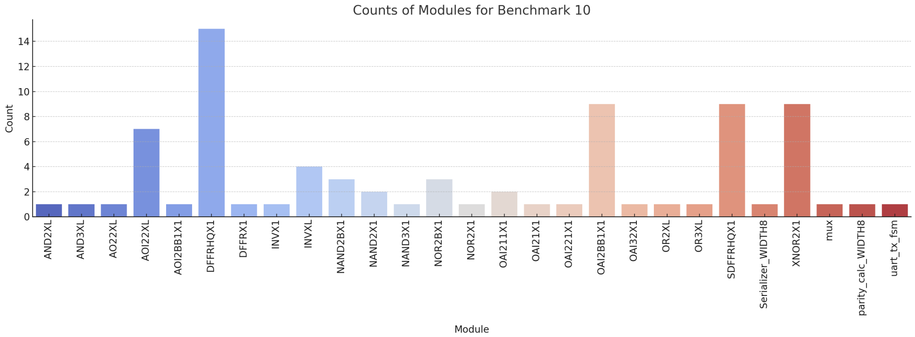
Counts of Modules for Benchmark 6

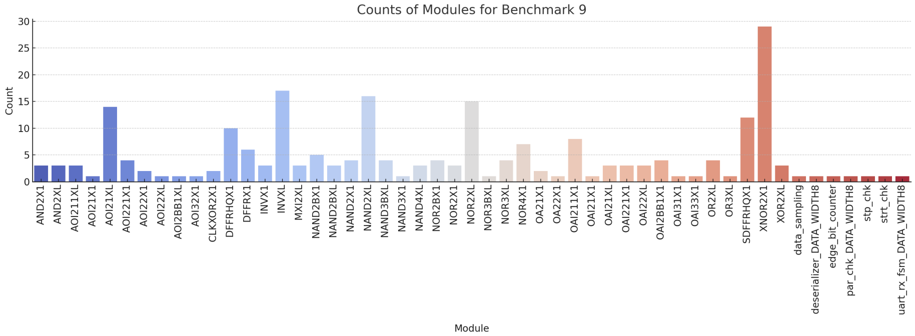


Counts of Modules for Benchmark 7




Counts of Modules for Benchmark 8
Count
0
100
200
300
400
500
600
700
800
Module
ADDFX1
ADDHX1
AND2X1
AND2XL
AND3XL
AND4XL
AO21X1
AO21XL
AO22X1
AO22XL
AOI211XL
AOI21X1
AOI21XL
AOI221X1
AOI222X1
AOI22X1
AOI22XL
AOI2BB1X1
AOI2BB1XL
AOI31X1
AOI32X1
AOI33XL
CLKXOR2X1
DFFHQX1
DFFQXL
Extract
INVX1
INVX2
INVXL
MX2X1
MX2XL
MXI2XL
NAND2BX1
NAND2BXL
NAND2X1
NAND2XL
NAND3BXL
NAND3X1
NAND4BBXL
NAND4BX1
NAND4BXL
NAND4XL
NOR2BX1
NOR2X1
NOR2X2
NOR2XL
NOR3BX1
NOR3BXL
NOR3X1
NOR3XL
NOR4BBX1
NOR4X1
Normalization
OA21X1
OA22X1
OAI211X1
OAI21X1
OAI21XL
OAI221X1
OAI222X1
OAI222XL
OAI22X1
OAI22XL
OAI2BB1X1
OAI31X1
OAI32X1
OAI33X1
OR2X1
OR2XL
OR3X1
OR4X1
Operation
Result_and_exception
XNOR2X1
align_mantisa
pipeline_align_op
pipeline_extract_align
pipeline_norm_res
pipeline_op_norm
pipeline_res_out


Counts of Modules for Benchmark 10
Count
0
2
4
6
8
10
12
14
Module
AND2XL
AND3XL
AO22XL
AOI22XL
AOI2BB1X1
DFFRHQX1
DFFRX1
INVX1
INVXL
NAND2BX1
NAND2X1
NAND3X1
NOR2BX1
NOR2X1
OAI211X1
OAI21X1
OAI221X1
OAI2BB1X1
OAI32X1
OR2XL
OR3XL
SDFFRHQX1
Serializer_WIDTH8
XNOR2X1
mux
parity_calc_WIDTH8
uart_tx_fsm


Counts of Modules for Benchmark 9
Count
0
5
10
15
20
25
30
Module
AND2X1
AND2XL
AOI211XL
AOI21X1
AOI21XL
AOI221X1
AOI22X1
AOI22XL
AOI2BB1XL
AOI32X1
CLKXOR2X1
DFFRHQX1
DFFRX1
INVX1
INVXL
MXI2XL
NAND2BX1
NAND2BXL
NAND2X1
NAND2XL
NAND3BXL
NAND3X1
NAND4XL
NOR2BX1
NOR2X1
NOR2XL
NOR3BXL
NOR3XL
NOR4X1
OA21X1
OA22X1
OAI211X1
OAI21X1
OAI21XL
OAI221X1
OAI22XL
OAI2BB1X1
OAI31X1
OAI33X1
OR2XL
OR3XL
SDFFRHQX1
XNOR2X1
XOR2XL
data_sampling
deserializer_DATA_WIDTH8
edge_bit_counter
par_chk_DATA_WIDTH8
stp_chk
strt_chk
uart_rx_fsm_DATA_WIDTH8

We have used following bashscript to re download, generated reports from server into local directory

```bash
#!/bin/bash
SERVER="192.168.88.31"
USER="dic_lab_02"
PASSWORD="sree@man"
REMOTE_DIR="/DIG_DESIGN/INTERNS/dic_lab_02"
LOCAL_DIR="$HOME/Downloads"
sshpass -p "$PASSWORD" scp -r "$USER@$SERVER:$REMOTE_DIR/sriman" "$LOCAL_DIR"
sshpass -p "$PASSWORD" scp "$USER@$SERVER:$REMOTE_DIR/runbash.sh" "$LOCAL_DIR"
echo "Download completed successfully."
```

Most Power consuming Sub blocks :

For FIR_filter 400 MHz slow, 400 MHz slow_hvt

For IIR_filter 400 MHz slow, 400 MHz slow_lvt, 400 MHz slow_hvt

For entire c6288 and FADD modules, except for 100 MHz, 400MHz of FADD_Dual_Main except fast_lvt

For b14 400 MHz slow, 400 MHz slow_lvt, 400 MHz slow_hvt have **Logic** as most power consuming sub block

For all other remaining **Registers** are being most power consuming sub block of modules

Most Area consuming Sub blocks :

For UART_TOP UART_RX, for UART_RX edge_bit_counter and for UART_TX Serializer_WIDTH8 are being most Area consuming modules

For remaining, benchmark modules themselves contribute largest Area in the design

THANK YOU