An automated C-to-GDS flow using open-source EDA tools for medium-sized SOC design and implementation

Kunal Ghosh & Anagha Ghosh

VLSI System Design Corp. Pvt. Ltd.

contactvsd@vlsisystemdesign.com — (91) 9686428727

Abstract

VSDFLOW is an automated solution to programmers, hobbyists and small scale semiconductor technology entrepreneurs who can craft their ideas in C language, and convert the design to hardware using VSD (C-to-GDS) FLOW. VSDFLOW is completely build using OPHW tools, where the user gives input algorithm in C. From here on the VSDFLOW takes control, algorithm (in C-language) is converted to RTL (using AHIR compiler), RTL is synthesized (using Yosys). The synthesized netlist is given to PNR tool (Qflow) and finally Sign-off is done with STA tool (using Opentimer). The output of the flow is GDSII layout and performance & area metrics of your design. VSDFLOW also provide hooks at all stages for users working at different levels of design flow. It is tested for 30k instance count design like ARM Cortex-M0, and can be further tested for multi-million instance count using hierarchical or glue logic.

Introduction

In recent years, there has been a steep increase in OPHW EDA tool usage, specially, among students and educational institutes, looking for a solution to innovate and implement their ideas. OPHW EDA toolset generally consists of a C-to-RTL compiler, an RTL synthesizer, a place-and-route engine and static timing analysis engine. VSDFLOW provides a plug-n-play solution to use entire OPHW toolset under single umbrella, where user needs to provide design in C language (or RTL language as hook) and VSDFLOW will generate GDSII and performance metrics of design. This enables user to focus on improvising quality of design ideas, while leaving the implementation to VSDFLOW. As a prototype, currently VSDFLOW uses toolset shown in below figure no. 1

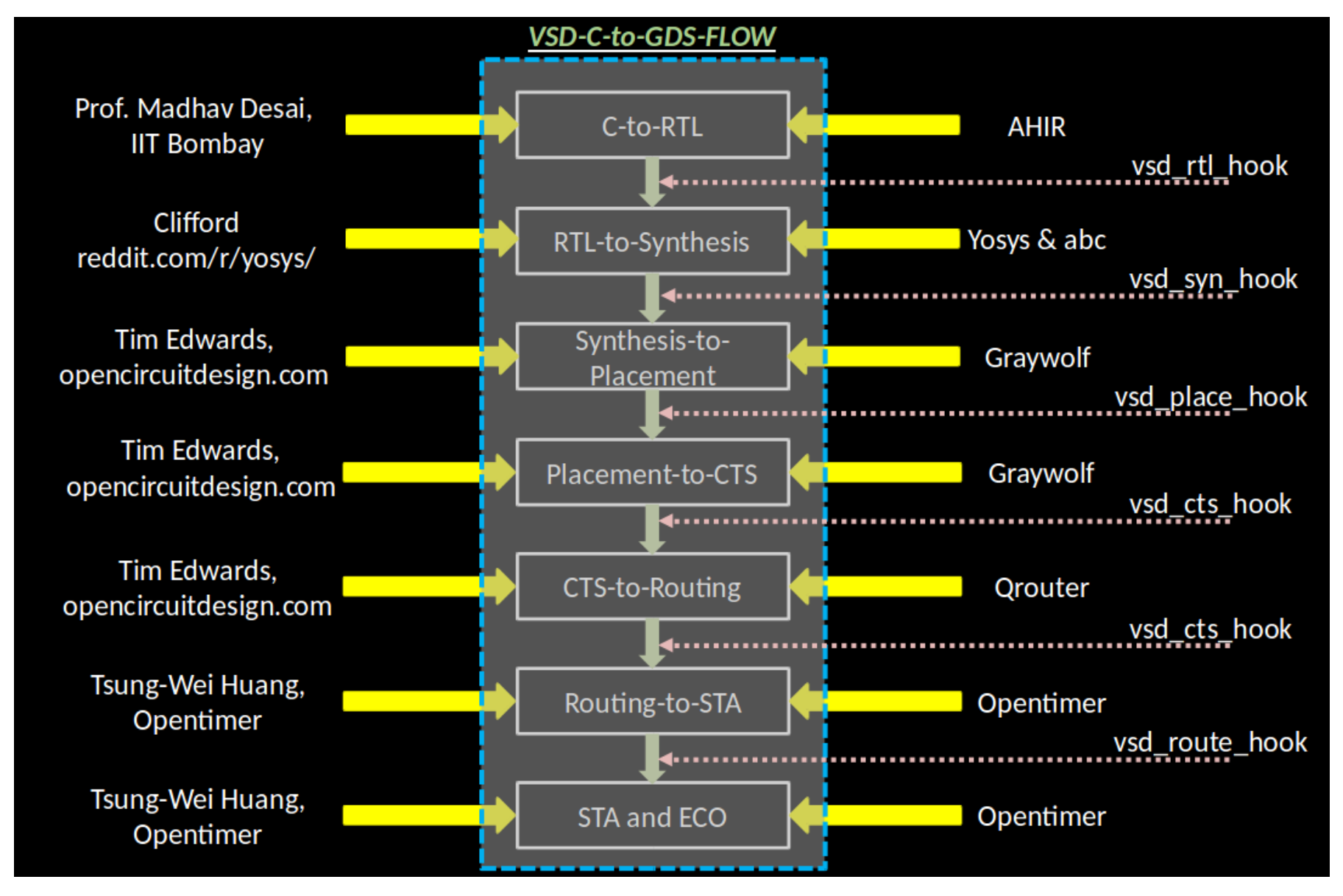
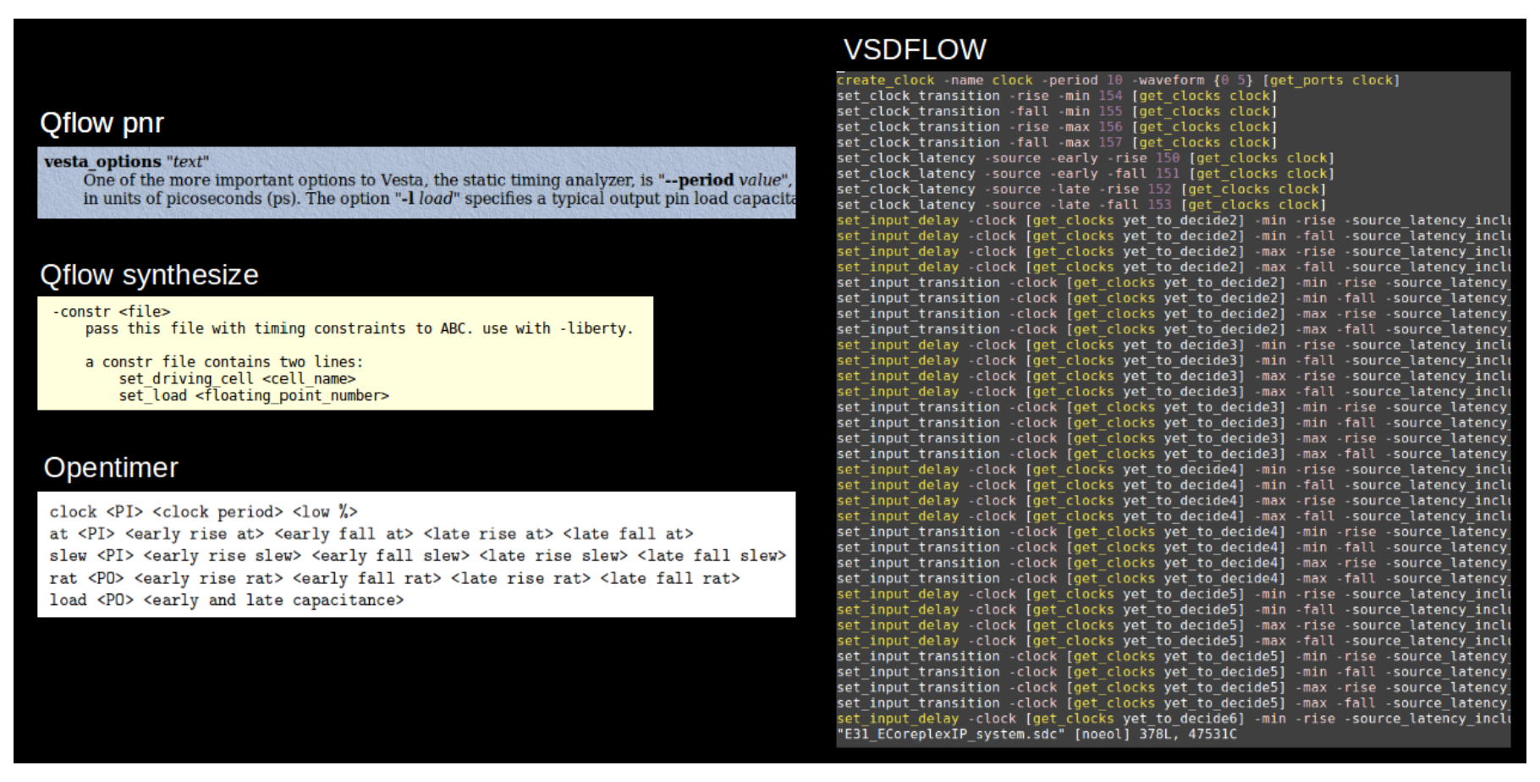


Figure 1: VSDFLOW framework

VSDFLOW is designed in such a flexible way, that user can pitch-in at any point of flow and some of them are enumerated below:

1. VSDFLOW hooks enables user to provide constraints in individual tool format or standard SDC format. Using this feature, user can switch to any tool within VSDFLOW framework. See figure 2



Qflow pnr

vesta_options "*text*"
One of the more important options to Vesta, the static timing analyzer, is "**--period** *value*", in units of picoseconds (ps). The option "**-l** *load*" specifies a typical output pin load capacita

Qflow synthesize

```
-constr <file>
    pass this file with timing constraints to ABC. use with -liberty.

    a constr file contains two lines:
        set_driving_cell <cell_name>
        set_load <floating_point_number>
```

Opentimer

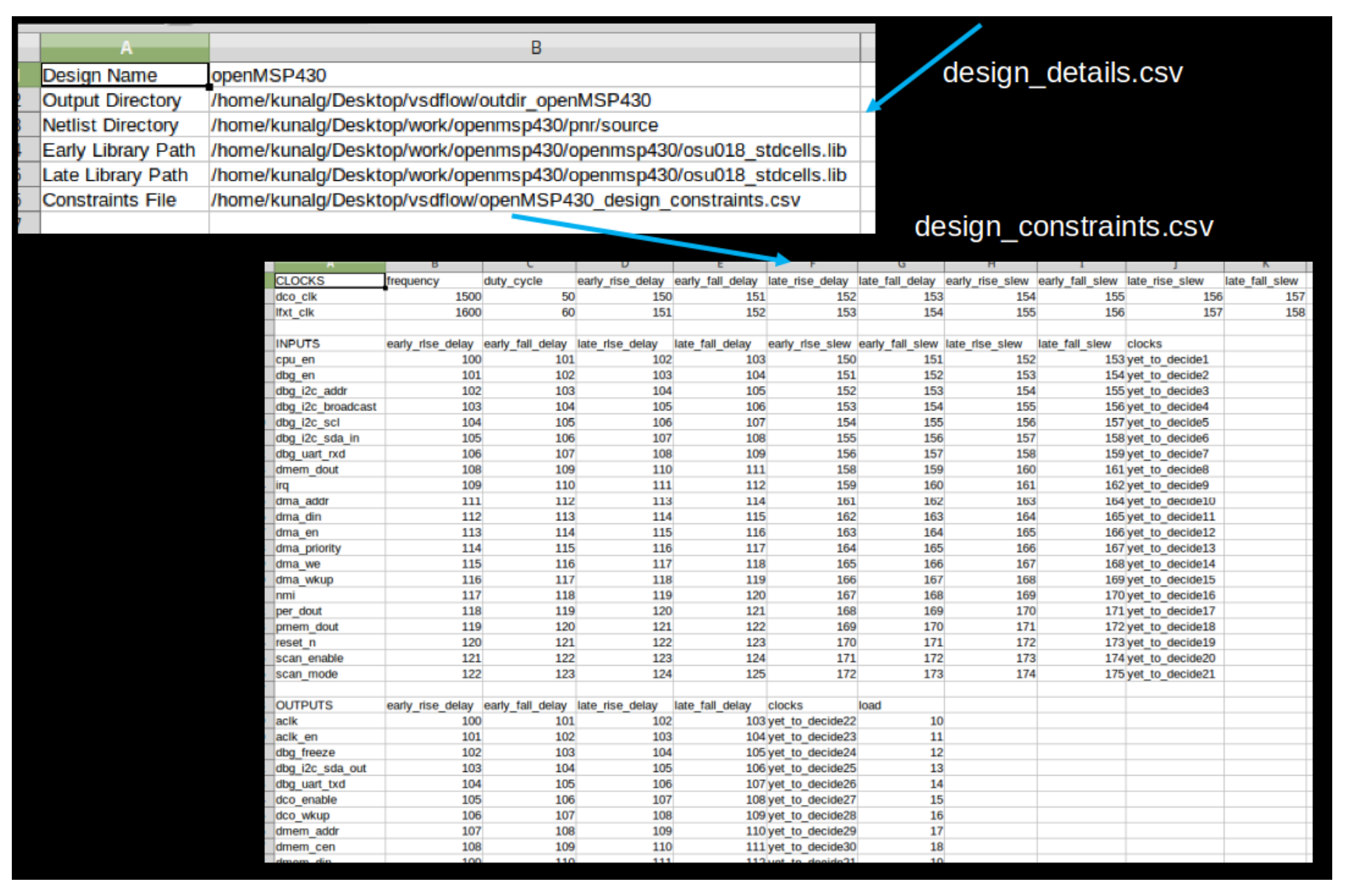
```
clock <PI> <clock period> <low %>
at <PI> <early rise at> <early fall at> <late rise at> <late fall at>
slew <PI> <early rise slew> <early fall slew> <late rise slew> <late fall slew>
rat <PO> <early rise rat> <early fall rat> <late rise rat> <late fall rat>
load <PO> <early and late capacitance>
```

VSDFLOW

```
create_clock -name clock -period 10 -waveform {0 5} [get_ports clock]
set_clock_transition -rise -min 154 [get_clocks clock]
set_clock_transition -fall -min 155 [get_clocks clock]
set_clock_transition -rise -max 156 [get_clocks clock]
set_clock_transition -fall -max 157 [get_clocks clock]
set_clock_latency -source -early -rise 150 [get_clocks clock]
set_clock_latency -source -early -fall 151 [get_clocks clock]
set_clock_latency -source -late -rise 152 [get_clocks clock]
set_clock_latency -source -late -fall 153 [get_clocks clock]
set_input_delay -clock [get_clocks yet_to_decide2] -min -rise -source_latency_inclu
set_input_delay -clock [get_clocks yet_to_decide2] -min -fall -source_latency_inclu
set_input_delay -clock [get_clocks yet_to_decide2] -max -rise -source_latency_inclu
set_input_delay -clock [get_clocks yet_to_decide2] -max -fall -source_latency_inclu
set_input_transition -clock [get_clocks yet_to_decide2] -min -rise -source_latency_
set_input_transition -clock [get_clocks yet_to_decide2] -min -fall -source_latency_
set_input_transition -clock [get_clocks yet_to_decide2] -max -rise -source_latency_
set_input_transition -clock [get_clocks yet_to_decide2] -max -fall -source_latency_
set_input_delay -clock [get_clocks yet_to_decide3] -min -rise -source_latency_inclu
set_input_delay -clock [get_clocks yet_to_decide3] -min -fall -source_latency_inclu
set_input_delay -clock [get_clocks yet_to_decide3] -max -rise -source_latency_inclu
set_input_delay -clock [get_clocks yet_to_decide3] -max -fall -source_latency_inclu
set_input_transition -clock [get_clocks yet_to_decide3] -min -rise -source_latency_
set_input_transition -clock [get_clocks yet_to_decide3] -min -fall -source_latency_
set_input_transition -clock [get_clocks yet_to_decide3] -max -rise -source_latency_
set_input_transition -clock [get_clocks yet_to_decide3] -max -fall -source_latency_
set_input_delay -clock [get_clocks yet_to_decide4] -min -rise -source_latency_inclu
set_input_delay -clock [get_clocks yet_to_decide4] -min -fall -source_latency_inclu
set_input_delay -clock [get_clocks yet_to_decide4] -max -rise -source_latency_inclu
set_input_delay -clock [get_clocks yet_to_decide4] -max -fall -source_latency_inclu
set_input_transition -clock [get_clocks yet_to_decide4] -min -rise -source_latency_
set_input_transition -clock [get_clocks yet_to_decide4] -min -fall -source_latency_
set_input_transition -clock [get_clocks yet_to_decide4] -max -rise -source_latency_
set_input_transition -clock [get_clocks yet_to_decide4] -max -fall -source_latency_
set_input_delay -clock [get_clocks yet_to_decide5] -min -rise -source_latency_inclu
set_input_delay -clock [get_clocks yet_to_decide5] -min -fall -source_latency_inclu
set_input_delay -clock [get_clocks yet_to_decide5] -max -rise -source_latency_inclu
set_input_delay -clock [get_clocks yet_to_decide5] -max -fall -source_latency_inclu
set_input_transition -clock [get_clocks yet_to_decide5] -min -rise -source_latency_
set_input_transition -clock [get_clocks yet_to_decide5] -min -fall -source_latency_
set_input_transition -clock [get_clocks yet_to_decide5] -max -rise -source_latency_
set_input_transition -clock [get_clocks yet_to_decide5] -max -fall -source_latency_
set_input_delay -clock [get_clocks yet_to_decide6] -min -rise -source_latency_inclu
"E31_ECoreplexIP_system.sdc" [noeol] 378L, 47531C
```

Figure 2: Different constraints format

2. VSDFLOW allows user to feed design details and constraints in generlaised format in csv file, which is a standard practise in industries, and achieve desired results, there-by enabling user to focus on crafting the best requirements for design. See figure 3

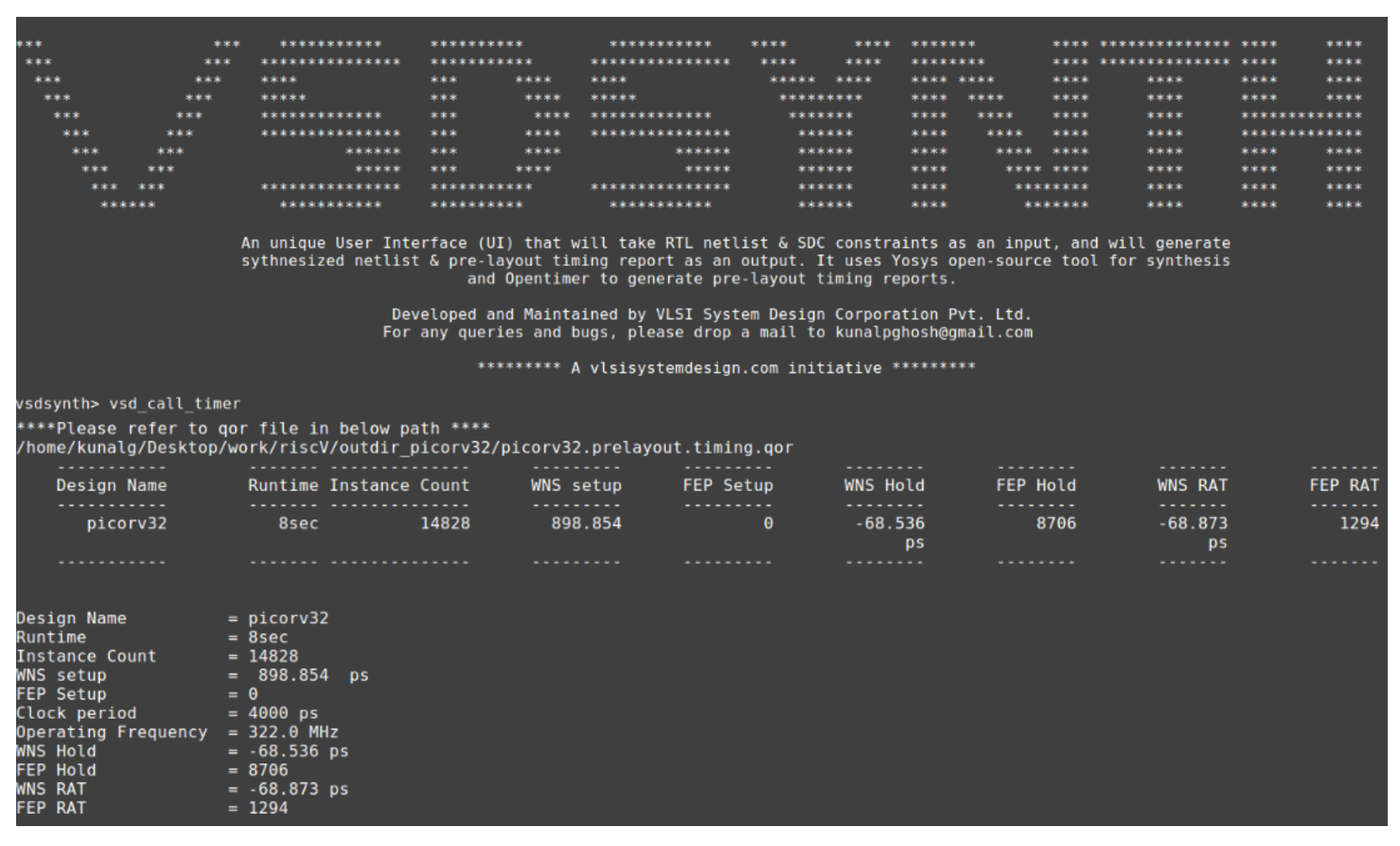


A	B
Design Name	openMSP430
Output Directory	/home/kunalg/Desktop/vsdflow/outdir_openMSP430
Netlist Directory	/home/kunalg/Desktop/work/openmsp430/pnr/source
Early Library Path	/home/kunalg/Desktop/work/openmsp430/openmsp430/osu018_stdcells.lib
Late Library Path	/home/kunalg/Desktop/work/openmsp430/openmsp430/osu018_stdcells.lib
Constraints File	/home/kunalg/Desktop/vsdflow/openMSP430_design_constraints.csv

CLOCKS	frequency	duty_cycle	early_rise_delay	early_fall_delay	late_rise_delay	late_fall_delay	early_rise_slew	early_fall_slew	late_rise_slew	late_fall_slew
dco_clk	1500	50	150	151	152	153	154	155	156	157
lfxt_clk	1600	60	151	152	153	154	155	156	157	158
INPUTS	early_rise_delay	early_fall_delay	late_rise_delay	late_fall_delay	early_rise_slew	early_fall_slew	late_rise_slew	late_fall_slew	clocks	
cpu_en	100	101	102	103	150	151	152	153	yet_to_decide1	
dbg_en	101	102	103	104	151	152	153	154	yet_to_decide2	
dbg_i2c_addr	102	103	104	105	152	153	154	155	yet_to_decide3	
dbg_i2c_broadcast	103	104	105	106	153	154	155	156	yet_to_decide4	
dbg_i2c_scl	104	105	106	107	154	155	156	157	yet_to_decide5	
dbg_i2c_sda_in	105	106	107	108	155	156	157	158	yet_to_decide6	
dbg_uart_rxd	106	107	108	109	156	157	158	159	yet_to_decide7	
dmem_dout	108	109	110	111	158	159	160	161	yet_to_decide8	
irq	109	110	111	112	159	160	161	162	yet_to_decide9	
dma_addr	111	112	113	114	161	162	163	164	yet_to_decide10	
dma_din	112	113	114	115	162	163	164	165	yet_to_decide11	
dma_en	113	114	115	116	163	164	165	166	yet_to_decide12	
dma_priority	114	115	116	117	164	165	166	167	yet_to_decide13	
dma_we	115	116	117	118	165	166	167	168	yet_to_decide14	
dma_wkup	116	117	118	119	166	167	168	169	yet_to_decide15	
nmi	117	118	119	120	167	168	169	170	yet_to_decide16	
per_dout	118	119	120	121	168	169	170	171	yet_to_decide17	
pmem_dout	119	120	121	122	169	170	171	172	yet_to_decide18	
reset_n	120	121	122	123	170	171	172	173	yet_to_decide19	
scan_enable	121	122	123	124	171	172	173	174	yet_to_decide20	
scan_mode	122	123	124	125	172	173	174	175	yet_to_decide21	
OUTPUTS	early_rise_delay	early_fall_delay	late_rise_delay	late_fall_delay	clocks	load				
aclk	100	101	102	103	yet_to_decide22	10				
aclk_en	101	102	103	104	yet_to_decide23	11				
dbg_freeze	102	103	104	105	yet_to_decide24	12				
dbg_i2c_sda_out	103	104	105	106	yet_to_decide25	13				
dbg_uart_txd	104	105	106	107	yet_to_decide26	14				
dco_enable	105	106	107	108	yet_to_decide27	15				
dco_wkup	106	107	108	109	yet_to_decide28	16				
dmem_addr	107	108	109	110	yet_to_decide29	17				
dmem_cen	108	109	110	111	yet_to_decide30	18				

Figure 3: Input to vsdflow in csv format

3. VSDFLOW has built-in proprietary-free commands, which can be used in a stand-alone fashion on VSDFLOW terminal. An example of proprietary-free command of VSDSYNTH (VSDSYNTH is a "synthesis+STA" section of VSDFLOW) is shown in figure 4



```
                An unique User Interface (UI) that will take RTL netlist & SDC constraints as an input, and will generate
                sythnesized netlist & pre-layout timing report as an output. It uses Yosys open-source tool for synthesis
                                   and Opentimer to generate pre-layout timing reports.

                          Developed and Maintained by VLSI System Design Corporation Pvt. Ltd.
                         For any queries and bugs, please drop a mail to kunalpghosh@gmail.com

                               ********* A vlsisystemdesign.com initiative *********

vsdsynth> vsd_call_timer
****Please refer to qor file in below path ****
/home/kunalg/Desktop/work/riscV/outdir_picorv32/picorv32.prelayout.timing.qor
    -----------        ------- --------------     ---------     ---------       --------       --------       -------       -------
    Design Name        Runtime Instance Count     WNS setup     FEP Setup       WNS Hold       FEP Hold       WNS RAT       FEP RAT
    -----------        ------- --------------     ---------     ---------       --------       --------       -------       -------
       picorv32           8sec          14828       898.854             0        -68.536           8706       -68.873          1294
                                                                                      ps                           ps
    -----------        ------- --------------     ---------     ---------       --------       --------       -------       -------

Design Name         = picorv32
Runtime             = 8sec
Instance Count      = 14828
WNS setup           =  898.854  ps
FEP Setup           = 0
Clock period        = 4000 ps
Operating Frequency = 322.0 MHz
WNS Hold            = -68.536 ps
FEP Hold            = 8706
WNS RAT             = -68.873 ps
FEP RAT             = 1294
```

Figure 4: Input to vsdflow in csv format

Results

We have tested VSDFLOW with OSU 180nm technology and below designs show the results in terms of performance and area

OPENMSP430

This is a synthesizable 16bit microcontroller core which is written in Verilog and available at opencores.org

Synthesis	
Runtime	1min
Operating Frequency	142MHz
Instance Count	9353
PNR	
Runtime	28min
Operating Frequency	187MHz
Area	965um x 697um
Instance Count	9353

Table 1: openMSP430 (by opencores.org)

PICORV32

PicoRV32 is a CPU core written by Clifford, that implements the RISC-V RV32IMC Instruction Set and available on github

Synthesis	
Runtime	1min
Operating Frequency	322MHz
Instance Count	14828
PNR	
Runtime	1hr 20min
Operating Frequency	387MHz
Area	1192um x 877um
Instance Count	14828

Table 2: picorv32 (by Clifford)

CORTEX-M0

An example system of MCU which contains a single cortex-m0 processor, internal program memory, SRAM data memory and boot-loader. Other peripherals include timers, GPIO, UART and watchdog timer. The evaluation RTL is available on ARM website

Synthesis	
Runtime	1min 9sec
Operating Frequency	238MHz
Instance Count	29479
PNR	
Runtime	3hr 47min
Operating Frequency	139MHz
Area	1705um x 1237um
Instance Count	29479

Table 3: Cortex-M0 (by ARM)

E31_ECoreplexIP_system

E31 Coreplex is a high performance implementation of the RISC-V RV32IMAC architecture. The evaluation RTL is available on SiFive website

Synthesis	
Runtime	1hr 55min
Operating Frequency	127MHz
Instance Count	1614028
PNR	
Runtime	WIP*
Operating Frequency	WIP*
Area	WIP*
Instance Count	WIP*

Table 4: E31_ECoreplexIP_system (by SiFive)

Conclusions

- VSDFLOW was shared with VSD community and 253 people sucessfully executed their designs with this flow
- A beginners guide in form of video course is developed to assist for OPHW tool installation and enhanced the usage of VSDFLOW, which in turn is bringing out new ideas and designs
- VSDFLOW has been tested and works seamlessly on designs upto 100k instance count, from opencores(for eg. openMSP430) and industry(ARM, SiFive)

Forthcoming Research

1. For design size with more than 100k to 1.6M instance count, a hierarchical approach is under development
2. VSDFLOW is working towards to include post-layout timing results with back-annotated parasitics included
3. VSDFLOW is enabling an automated timing ECO framework, using ECO engine from Opentimer. Using this feature, user will be able to provide list of violating endpoints and VSDFLOW-ECO will return list of ECO fixes

Acknowledgements

- Thanks to Tim Edwards, who has been continously helping in various aspects of qflow enablement within VSDFLOW. Tim's continous and timely support has been really helpful in VSDFLOW inception and making it available for community usage
- Thanks a lot to Tsung-Wei Huang for prompt response on any query relate to Opentimer, which has led to resolve many doubts and queries raised by community to VSD team. From community feedback, Tsung-Wei Huang plans to update Opentimer STA tool, which will support advanced timing models

**graywolf placement runtime reduction for designs with instance count 100k+ is under work*