Static Power Analysis In RedHawk-SC

Table of Contents

1. Introduction

In this Appnote we will be taking about Different ways of coming up with power for static analysis in RedHawk-SC. We would be mainly talking about `SwitchingActivityView` which primarily does the activity computation hold toggle rate, 0/1 Probability of each instance input and output pins and `PowerView` which either computes the average power based on activity from input `SwitchingActivityView` or reads in the power number directly from IPF files.

2. SwitchingActivityView

The `SwitchingActivityView` assigns user-defined toggle rates and duty cycles and, optionally, uses activity propagation (based on the standard algorithm proposed by *Farid N Najm, Toronto University*) to propagate activity through the combinational logic between register outputs (see Figure 1). User inputs to control activities can be imported through *SAIF* (gate-level only) or *FSDB/VCD* (direct or from a `ValueChangeView` or by passing vcd_files as well as global design and block specific default settings.

The primary use for `SwitchingActivityView` is to feed into Static IR drop and EM analysis. However, the toggle rates can also feed into a vector-less simulation, providing individual flop toggle rates as a probability of switching rather than just the global/block level activity settings of `create_scenario_view`

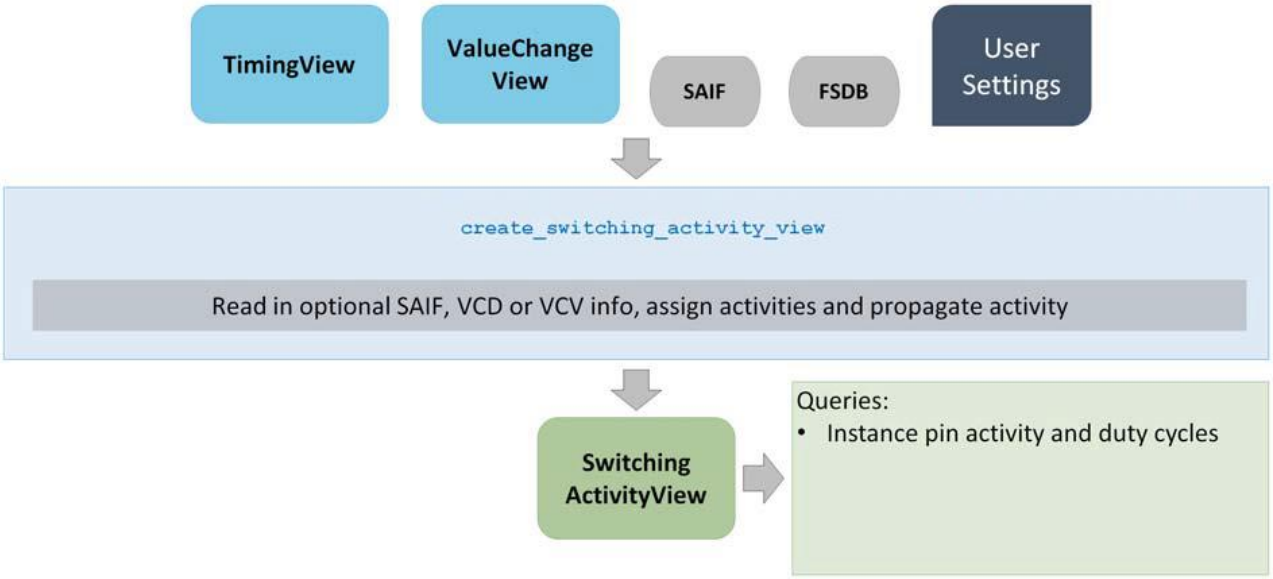

Figure 1. `SwitchingActivityView`

2.1 Syntax

`SwitchingActivityView` can be created using command `<SeaScapeDB>.create_switching_activity_view`. Listing down the arguments:

Argument	Description
timing_view	TimingView object (type=TimingView, required=True)
saif_files	List of SAIF files (type=object, default_value=None, constraint="list")
vcd_files	List of VCD files (type=list, default_value=None, constraint="VCD files")
value_change_data	List of dicts specifying ValueChangeView slices (type=list, default_value=None)
propagation_style	Type of propagation for instance pins without data (type=str, default_value='propagate_activity', constraint="One of ['propagate_activity', 'assign_default_activity', 'no_propagation']")
disable_clocks	List of clocks to disable (type=list, default_value=None, constraint="list of clock names or patterns to disable")

clock_source_toggle_rates	Dict of {clock_name : toggle_rate} to override default toggle rate of 2.0 on specific clocks (type=dict, default_value=None, constraint="dict")
use_sta_clock_info	Use clock info from STA file if available, otherwise use default clock info (type=bool, default_value=False)
settings	Activity settings (type=dict, default_value=None, constraint="dict of settings")
object_settings	Dict of settings for various scopes (type=dict, default_value={}, constraint="dict of design, cell and/or instance values")
time_interval_length	Length of each time interval of switching activity data (type=float, default_behavior='use entire duration')
tag	Name of the view (type=str, default_value='swa')
options	View creation options, typically from get_default_options() (type=object, default_value=None)
infer_icg_enable_signals_from_fanin_prop	Infer ICG behavior from signals propagated to enable pin (type=bool, default_value=False)
vcd_x_logic_state	available options for X handling (type=object, default_value=None, constraint="one of [None, -1, 0, 1, 2, 3, 'ignore_x', 'use_x_as_fall_toggle', 'use_x_as_rise_toggle', 'use_x_as_toggle']")

2.2 saif_files

saif_files argument takes in list of dict . Each of these dict are for one SAIF file . The keys inside each dict being:

Key	**Value**
file_name	Path to the IPF file
file_name	Path to the SAIF file
cell_name	Name of the DEF cell for which to apply the file (optional)
instances	Hierarchical instance object for which to apply the file (optional)

Example : `saif_files = [{'file_name' : 'example.saif',`

```
'instances' : [Instance('')],
'preamble'  : 'sys/block1'}]
```

2.3 vcd_files

`SwitchingActivityView` can directly take vcd/fsdb files as input. Here only rise/fall toggle counts are read from vcd along states of constant signal, hence its faster than `ValueChangeView` which stores complete information from vcd/fsdb files. vcd_files argument takes in list of dict . Each of these dict are for one VCD file . The keys inside each dict being:

Key	Value
file_name	Path to the IPF file
file_name	Path to the VCD file
instance_name	Hierarchical instance name for which to apply the file (optional)
preamble	Name prefix to remove from identifiers in the file (optional)
start_time	Starting time value to consider in VCD (optional)
stop_time	Ending time value to consider in VCD (optional)
start_tick	Starting tick value to consider in VCD (optional)
stop_tick	Ending tick value to consider in VCD (optional)
rtl_map	Type of name mapping for RTL VCD; must be one of ['', 'simple', 'mixed'] (optional)
top_only	Only read signals at top level of hierarchy (optional)
time_delay	Shift events/waveform by this amount (optional)
fuzzy_chars	DEF to VCD/FSDB name matching may treat these characters as equivalent if exact match cannot be found

The format and keys are very similar to what used in `ValueChangeView`. We do not have time_slices in SWA. A simple example is given below. For More details kindly refer to VCV AppNote in RedHawk-SC

Example :
```
vcd_files = [
   {'file_name' : '../design_data/FSDB/GLITCHFREE/top.fsdb',
   'instance_name': '',
   'preamble': 'dut/chip/chip_core_inst/top_inst/',
   'rtl_map':  'mixed',
   'start_time': 496000000e-12,
```

```
    'stop_time': 496010000e-12,
}, ]
```

2.4 value_change_data

If `ValueChangeView` is already available, then user can pass this information to `SwitchingActivityView` using value_change_data argument which takes in list of dict. The keys inside each dict being :

Key	Value
file_name	Path to the IPF file
view	ValueChangeView tag name, whose data we want to apply
slice_name	slice of the specified ValueChangeView
time_delay	amount of time to shift events/waveforms. Optional.
time_scale_factor	all time values for signal changes will be multiplied by this factor. Optional.

2.5 X Logic Handling

There would be X states i.e. don't care logic states in the vcd file. These cannot be used as such for further logic processing and have to be converted to either low (0) or high (1) state. There are different options available as to how this has to be done. The argument `vcd_x_logic_state` is used. The options being:

- `0 / 'use_x_as_fall_toggle'` : Converts all X to 0/low state.
- `1 / 'use_x_as_rise_toggle'` : Converts all X to 1/high state.
- `2 / 'use_x_as_toggle'` : Both X to logic and logic to X are considered as toggle.
- `3 / 'ignore_x'` : X logics are ignored. This is the default behavior.
- -1 : It considers X to logic as toggle and keeps logic to X as X itself. As such, this cannot be used for dynamic analysis.

We can consider an example as the following vcd file waveform:
x -> 1 -> x -> 0 -> x -> 0
`vcd_x_logic_state = 0` will give 0 -> 1 -> 0 -> 0 -> 0 -> 0
`vcd_x_logic_state = 3` will give 1 -> -> 0 -> 0

2.6 Toggle Rate Input (propagation_style / settings)

`settings` and `object_settings` are used to control toggle rate and duty cycle. For toggle rate, the frequency or clock period associated with an instance is also important and this depends on TimingView input.

`propagation_style` is used to set top level propagation style for the design. The default value is `propagate_activity` which propagates toggle rates. It can be set to `assign_default_activity` which will turn off this propagation.

`settings` has the following keys and default values:

- `'default_clock_period'`: clock information comes from TimingView. For the ones missing/untraceable, this default clock period is used. Default value is 1e-09
- `'default_primary_input_port_toggle_rate'`: toggle rate of primary input. Default value of 0.15
- `'minimum_toggle_rate_override_case_analysis'`: minimum toggle rate can be set for a scope using `object_settings`. This key is to determine whether this minimum toggle rate needs to override sta settings of constant set for a pin/net. Default is `False`
- name_mapping_settings : settings when RTL fsdb file is provided in vcd_files , this dict has following keys :
    - rtl_name_mapping_files: List of dict for RTL to Gate mapping file
    - rtl_name_mapping_rules : List of dict of RTL to Gate mapping rules

Kindly refer to VCV Appnote for details on rtl_name_mapping_files and rtl_name_mapping_rules , a simple example is given below :

```
rtl_name_mapping_files = [{'file_name'      :'./namemap.txt',
                           'rtl_preamble'   : 'RTLP1.RTLP2',
                           'gate_preamble'  : 'GATEP1.GATEP2',
                           'delimiter'      : '.',
                           'map_symbol'     : '=>',
                           'instances'      : ['']}]

rtl_name_mapping_rules = [
                 {'type':'rtl_to_gate', 'from':"_abc", 'to':""}]
```

`object_settings` is a dict of dict with many keys in multiple levels.

Key	**Value**
design_values	dict of default values for the entire design
block_values	list of dict of values that apply to objects within block matching the pattern
cell_values	list of dict of values that apply to objects within instances of cells matching the pattern
leaf_instance_values	list of dict of values that apply to specified leaf instances.
leaf_instance_pin_values	list of dict of values that apply to all the combinations of instance pins.
port_values	list of dict of values that apply to all the top-level ports.

The dict of values for a given scope (design, block, etc) can have these optional keys and values:

Key	Value
clock_pin_toggle_rate	float, must be >= 0.0. Default = 2.0.
sequential_output_pin_toggle_rate	float, must be >= 0.0. Default = 0.15.
macro_output_pin_toggle_rate	float, must be >= 0.0. Default = 0.15.
icg_output_pin_toggle_rate	float, must be >= 0.0. Default = 2.0.
combinational_pin_toggle_rate	float, must be >= 0.0. Default = 0.15.
propagation_style	one of ['propagate_activity', 'assign_default_activity', 'no_propagation']. Default = 'propagate_activity'.
minimum_toggle_rate	float, must be >= 0.0. Default = 0.0.
toggle_rate	float, valid for leaf_instance_values, cell_values and port values only, must be >= 0.0. No default.
sequential_clock_input_pin_toggle_rate	float, must be >= 0.0, No default.
macro_clock_input_pin_toggle_rate	float, must be >= 0.0, No default.
clock_pin_one_probability	float, must be >= 0.0. Default = 0.5.
sequential_output_pin_one_probability	float, must be >= 0.0. Default = 0.5.
macro_output_pin_one_probability	float, must be >= 0.0. Default = 0.5.
icg_output_pin_one_probability	float, must be >= 0.0. Default = 0.5.
icg_enable_pin_one_probability	float, must be >= 0.0. Default = 1.0.
icg_test_enable_pin_one_probability	float, must be >= 0.0. Default = 0.0.
combinational_pin_one_probability	float, must be >= 0.0. Default = 0.5.
one_probability	float, valid for leaf_instance_values and port values only, must be >= 0.0. No default.

The dict for block_values or cell_values must include this key :

'pattern' : String or CMRegex pattern to match. If a glob-style pattern is specified, '?' or '*' will not match '/'.

The dict for leaf_instance_values must include this key:

'instances' : List of Instance objects or a single Instance

The dict for leaf_instance_pin_values must include these keys:

'instances' : List of Instance objects or a single Instance

'pins' : List of Pin objects or a single Pin

The dict for port_values must include this key:

pins' : List of Pin objects or a single Pin

Example :

```
settings = {
      'default_clock_period': 2.3e-09,
      'default_primary_input_port_toggle_rate': 0.1
      }

object_settings = {
      'design_values': {
            'clock_pin_toggle_rate': 2.0,
            'combinational_pin_toggle_rate': 0.2,
            'macro_output_pin_toggle_rate': 0.25,
            'sequential_output_pin_toggle_rate' : 0.25,
            'icg_output_pin_toggle_rate': 2.0,
            'propagation_style': 'assign_default_activity'},
      'block_values': [
            {'pattern': 'cpu3',
            'clock_pin_toggle_rate': 1.5,
            'sequential_output_pin_toggle_rate': 0.2,
            'icg_output_pin_toggle_rate': 1.5,
            'propagation_style': 'propagate_activity'},

            {'pattern': 'cpu2',
            'propagation_style': 'propagate_activity'},

            {'pattern': 'cpu4',
            'clock_pin_toggle_rate': 1,
            'sequential_output_pin_toggle_rate': 0.3,
            'icg_output_pin_toggle_rate': 1,
            'propagation_style': 'assign_default_activity'},
            ]
}
```

3.PowerView

The `PowerView` stores the power for each instance in the design. It can be thought of as a lite version of a ScenarioView that only considers instance power (see Figure 2). There are several methods for creating a PowerView depending on the source of the data.

- It can be used to import and store a user-defined instance power file (.ipf formatted file).
- It can be used to calculate the power of instances using the toggle rates and duty cycles coming from a `SwitchingActivityView`

- It can be used to convert the currents from an existing scenario view.

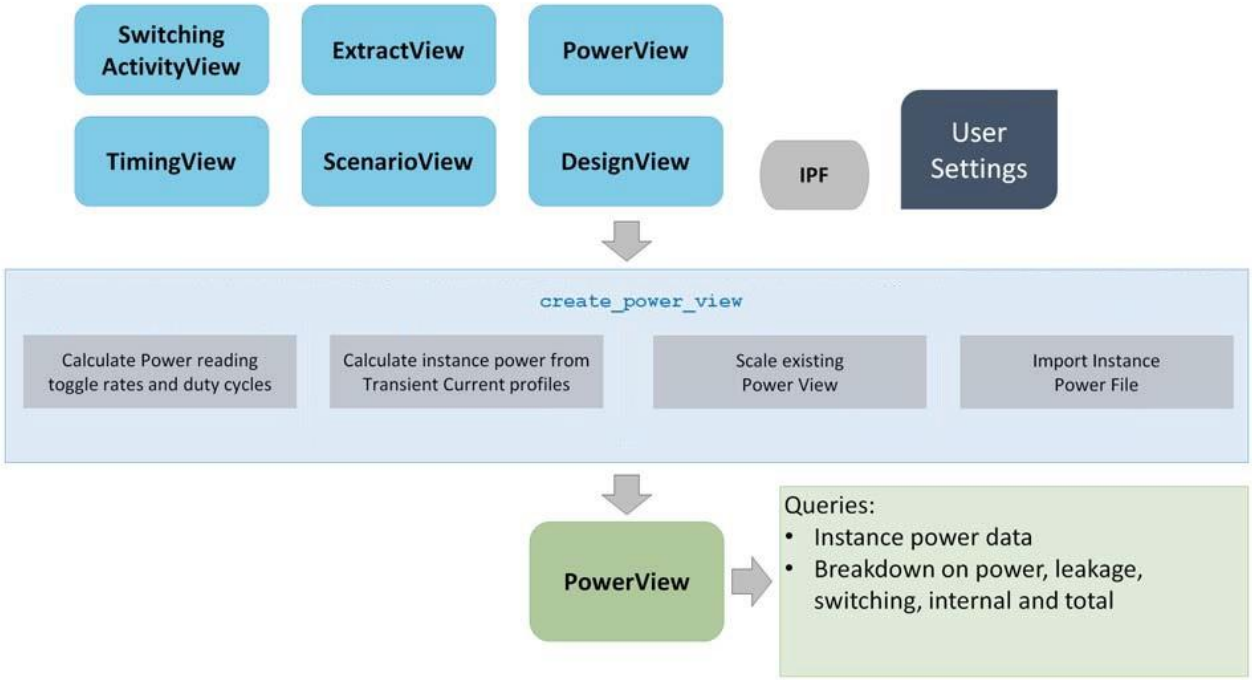

Figure 2. `PowerView`

3.1 Syntax

`PowerView` can be created using command `<SeaScapeDB>.create_power_view`. Listing down the arguments:

Argument	Description
timing_view	TimingView object. This is mandatory
design_view	DesignView object (type=DesignView, default_behavior='Obtain DesignView from the other specified views')
power_files	List of IPF file names (type=object, default_value=None, constraint="list")
switching_activity_view	SwitchingActivityView object (type=SwitchingActivityView, default_value=None)

extract_view	ExtractView object from extraction (type=ExtractView, default_value=None)
external_parasitics	ExtractView object from files (type=ExtractView, default_value=None)
timing_view	TimingView object (type=TimingView, default_value=None)
process_corner	liberty process corner (type=str, default_value='default_process')
voltage_levels	Dict of voltage levels (type=object, default_value=None, constraint="dict of net name pattern to float")
virtual_supplies	Virtual Supplies (type=list, default_value=None, constraint="List of virtual supplies")
supply_connection_rules	supply connection rules (type=list, default_value=None, constraint="List of supply connection rules")
temperature	Temperature to use for power calculation (type=float, default_value=25.0)
object_settings	Dict of settings for various scopes (type=dict, default_value={}, constraint="dict of design, cell and/or instance values")
tag	Name of the view (type=str, default_value='pwr')
options	View creation options, typically from get_default_options() (type=object, default_value=None)
limit_net_cap_to_pin_max_cap	limit net cap to its driver pin max_capacitance given in liberty (type=bool, default_value=False)
current_data_precedence	List of data sources to consider for current/power calculation (type=list, default_value=['NLPM', 'APL', 'CCSPower'])
calculation_settings	Controls how different components of power are calculated (type=dict, default_value=None, constraint="dict of power component string to dict of values")

3.2 power_files

power_files argument takes in list of dict . Each of these dict are for one IPF file . The keys inside each dict being:

Key	**Value**

file_name	Path to the IPF file
instance_name	Data will be applied to this hierarchical instance (optional)
cell_name	Data will be applied to all instances of this hierarchical cell (optional)
scaling_factors	A dict of type {<DomainNet#> scaling_factor} (optional). <DomainNet#> Specify domain net to be scaled by the scaling_factor. Use '*' to specify all the domains not specified.
override	boolean, True means data in this file will be with higher precedence (optional). False by default. There can only be one power file with 'override' : TRUE
type	Type of data in this power file (optional). Possible values ['instance','cell']. The default is 'instance', which means the first column in power file is instance name, 'cell' means it is cell name. if 'cell' is used, 'scaling_factors' setting will be ignored, also 'instance_name' or 'cell_name' can not be defined.

Two formats of IPF (Instance Power File) are supported : 9-column and 3-column

3.2.1 9-column ipf file format

```
# <instance_name> <power_pin_name> <voltage> <toggle_rate> <frequency>
<total_power> <switching_power> <internal_power> <leakage_power>
inst_129902 VDD 1.20000004768 0.20000000298 781249984.0
0.00499999988824 0.0 0.00499999988824 0.0
inst_129995 VDD 1.20000004768 0.583509027958 390624992.0
3.36912999046e-05 1.08022995846e-05
2.28824992519e-05 6.4573502101e-09
inst_130106 VDD 1.20000004768 0.0583508983254 390624992.0
1.66340998931e-06 1.44139005442e-06
2.20369997805e-07 1.65312996625e-09
```

3.2.2 3-column ipf file format

```
# <instance_name> <pin_power_W> <Vdd_pin_name>
# <instance_name> <pin_current_A> <Vss_pin_name>
U1/U1 8e-8 VDD
U1/U2 1e-8 VDD
```

A simple example for power_files is given below.

```
power_files = [
   {'file_name': '../design_data/top.ipf',
   'instance_name': '',
},]
```

3.3 voltage_levels

voltage_levels is a dict. Keys may be of type str (glob pattern) or CMRegex object for power/ground net name matching. Values are float (voltage in V).

For example: {'VDD1' : 0.9, CMRegex('VDDB.*') : 0.75, 'VSS' : 0.0}

Important Note : ground net names do not match based on glob or regex; they have to be exact (string) match to apply.

3.4 current_data_precedence

current_data_precedence is a list of strings such as ['APL', 'CCSPower', 'NLPM'] describing the precedence for calculating current or power. In this example, APL data would be used if it exists for a cell, otherwise CCS Power, and finally NLPM. The default precedence for this command is ['NLPM', 'APL', 'CCSPower'].

3.5 calculation_settings

calculation_settings is a dict describing how to calculate various power components. Valid keys are ['leakage', 'internal']. For each key, the value is a dict with settings for that power component. The component power dicts may contain the following keys and values.

Key	Value
cell_types_with_independent_nlpm	For the 'internal' power component, list of cell types that have independent nlpm. Valid cell types are ['flip_flop', 'latch', 'memory', 'combinational', 'pad', 'macro', 'unknown'].
nlpm_extrapolation	For the 'internal' power component, bool, if True extrapolation beyond the table limits will be enabled.
default_input_pin_transition_time	For the 'internal' power component, float, transition time for instance pins with no data from STA.
assign_cell_leakage_to_primary_power	For the 'leakage' power component, bool, if True for cells with only cell_leakage_power put entire value on the primary power pin, if False divide equally among all power pins.

If one or more keys are not specified, the following defaults will be used: {'internal': {'default_input_pin_transition_time': 1e-11, 'cell_types_with_independent_nlpm': [], 'nlpm_extrapolation': False}, 'leakage': {'assign_cell_leakage_to_primary_power': False}}

3.6 object_settings argument input

object_settings is a dict with these optional keys and values:

Key	Value
design_values	dict of default values for the entire design
block_values	list of dict of values that apply to objects within block matching the pattern
cell_values	list of dict of values that apply to objects within instances of cells matching the pattern
leaf_instance_values	list of dict of values that apply to specified leaf instances.

The dict for block_values or cell_values must include this key:

'pattern' : String or CMRegex pattern to match. If a glob-style pattern is specified, '?' or '*' will not match '/'.

The dict for leaf_instance_values must include this key:

'instances' : List of Instance objects or a single Instance

The dict of values for a given scope (design, block, etc) can have these optional keys and values:

'mode_control' : dict to specify the switching states on macros. The keys defined for the mode_control dict are as below.

Key	Value
mode_probabilities	A dict of mode name or dict of tuple of mode names to probability value
mode_sequence	A list of mode names or list of tuple of mode names to be applied. This always repeats.

Only one of mode_sequence, or mode_probabilities can be given. There are implicit modes available too: _low_energy_mode_, _median_energy_mode_, _high_energy_mode_, _leakage_only_mode_, _off_mode_. Energy modes are categorized by the energy value, _leakage_only_mode_ is for the case where only leakage current is required. _off_mode_ is for turning off the instance, hence cannot be mixed with other modes.

Example :

```
'mode_control' : {'mode_sequence' : ['READ', 'WRITE', 'READ',
'STANDBY'], }
```

```
'mode_control' : {'mode_probabilities' : {'READ':0.3, 'WRITE':0.5,
'STANDBY':0.2},}
```

The precedence of an individual setting for a given leaf instance is as follows:

1. A setting directly on the leaf instance pins or ports
2. A setting directly on the leaf instance
3. A setting on the cell of the leaf instance
4. Setting on the lowest enclosing block that has settings, then the corresponding cell of that block
5. Setting from the design_values
6. Tool default values

4. Different Power Calculation Flows

RedHawk-SC provides various ways of coming up with average power using above two views for Static Analysis , there are described below

4.1 IPF file based Static Flow

Figure 3 describes how IPF file based Static flow can be performed. The IPF file should cover all the instances in the design: otherwise, any missed instances will not get power assignments. If IPF coverage is not 100% , then it is recommended to have SWA ->PV flow as described in Section 4.2

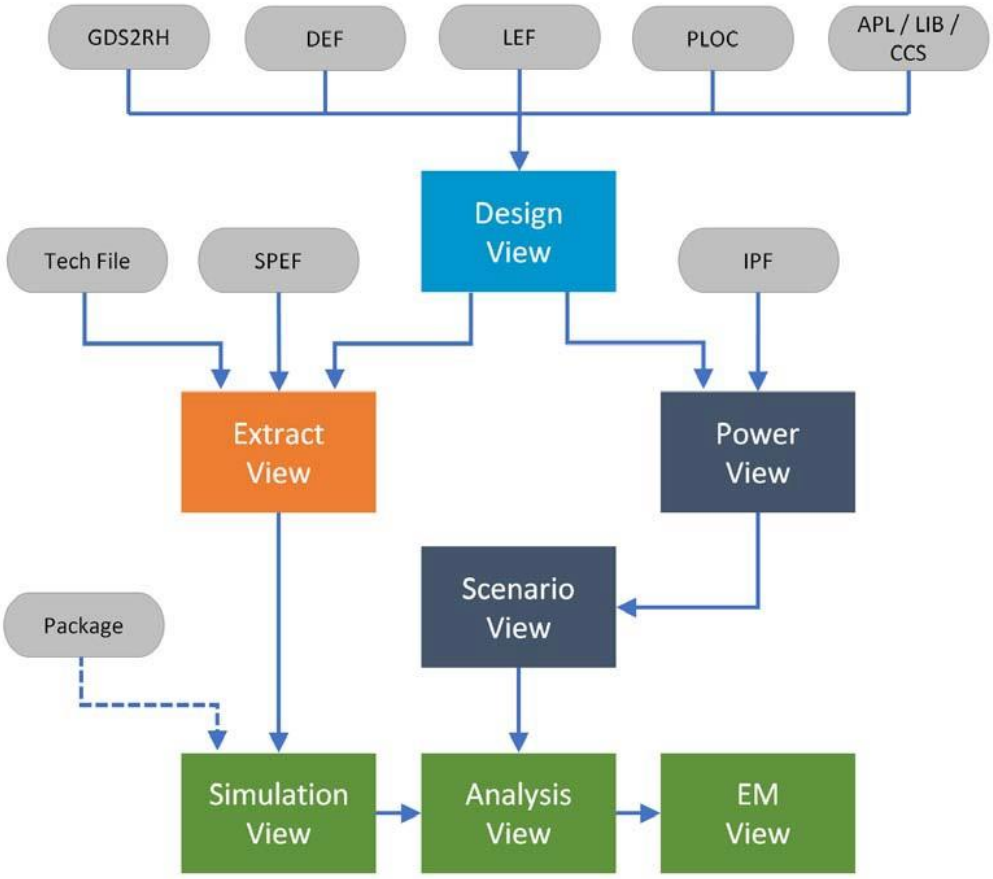

Figure 3. IPF file based Static Flow

Flow setup

The following command lines are the typical steps to use in an IPF static flow.

```
dv0 =db.create_design_view(def_files=def_files,lef_files=
lef_files,lib_views=lv,options=options,package=None,tag= 'dv0',
top_cell_name = 'GENERIC',cmm_views=[model])
dv = db.create_modified_design_view(dv0,eco_commands=
pkg.parse_ploc_func(th_ploc),tag='dv')
tv = db.create_timing_view(dv, timing_window_files=tw_files,
options=options)
nv =db.create_tech_view(tech_file_name='../design_data/tech/
GENERIC.tech',options=options)
ev
=db.create_extract_view(dv,tech_view=nv,temperature=110,options=option
s)
evx =db.create_extract_view_from_files(dv,input_files =spef_files,
tag='evx',options=options)
pwr = db.create_power_view(dv=dv, power_file_names='power.ipf.gz',
tag='pwr')
```

```
scn = db.create_scenario_view(power_view=pwr,scenario_type='Static',
options=options, voltage_levels=voltage_levels,tag='scn')
sv = db.create_simulation_view(ev, enable_reduction=False,
options=options, tag = 'sv')
av = db.create_analysis_view(sv, scn, tag='av', options=options)
```

4.2 User controlled toggle rate driven power for static analysis

You can specify the following control variables as an input in the *dict* data type as part of the user control setting. You can also choose to set the propagation to more of a plain activity in creating a switching activity view.

settings is a dict with these keys and default values:

```
{
'default_clock_period' : 1e-09,
'default_clock_pin_toggle_rate' : 2.0,
'default_combinational_pin_toggle_rate' : 0.15,
'default_macro_output_pin_toggle_rate' : 0.15,
'default_sequential_output_pin_toggle_rate' : 0.15,
}
```

Figure 4 then describes how toggle rate driven Static flow can be performed.

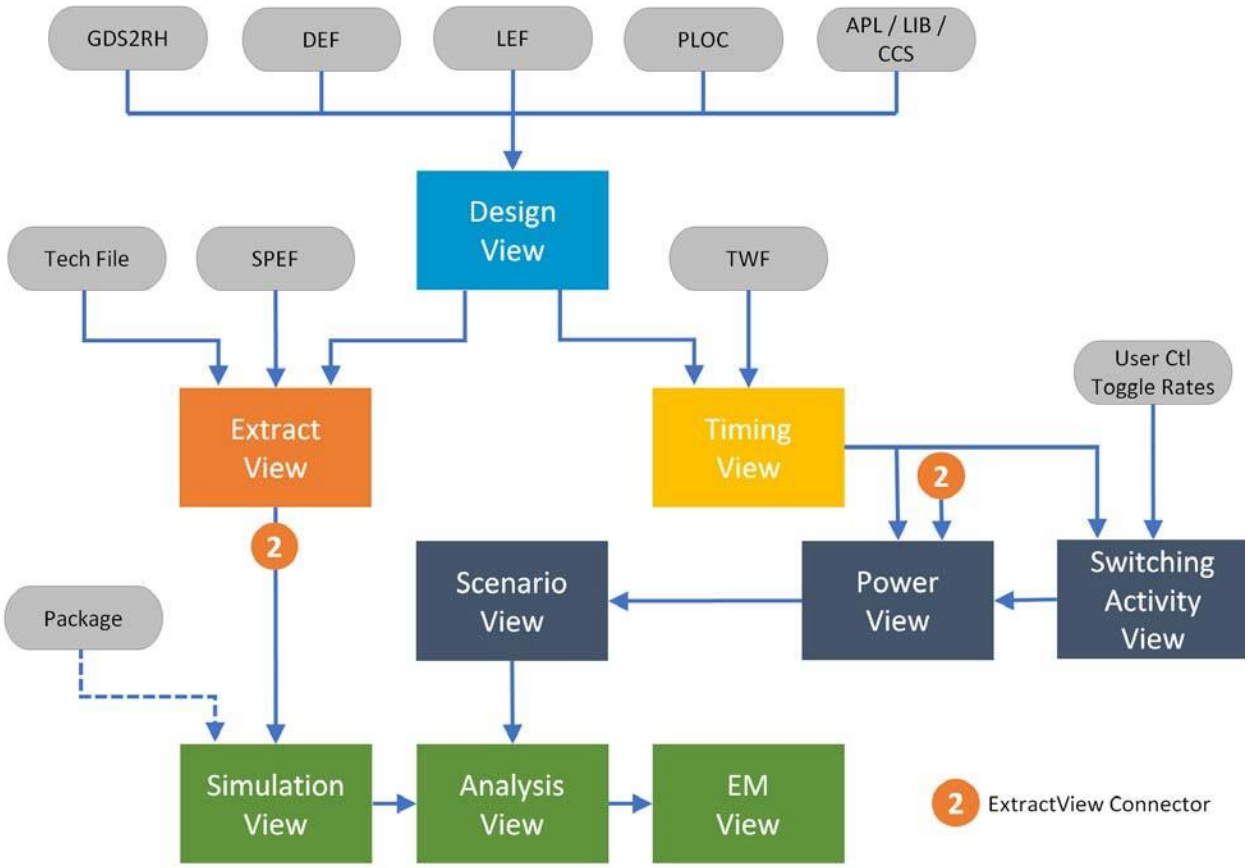

Figure 4. Toggle Rate driven Static Analysis

Flow setup

The following command lines are the typical steps to use in a global toggle rate-based flow.

```
swa_settings = {'default_sequential_output_pin_toggle_rate' : 0.5,
'default_clock_period' : 2.564102564102564e-09,
'default_clock_pin_toggle_rate' : 2.0,
'default_combinational_pin_toggle_rate' : 0.50,
'default_icg_output_pin_toggle_rate' : 1.3,
'default_macro_output_pin_toggle_rate' : 0.50,
'default_primary_input_port_toggle_rate' : 0.15
}

swa_prop = db.create_switching_activity_view(tv,settings =
swa_settings,propagation_style = 'assign_default_activity',tag =
'swa_prop', options = options)

pwr_avg =
db.create_power_view_from_switching_activity(swa_prop,extract_view =
ev,external_parasitics = evx,timing_view = tv,temperature =
25.0,voltage_levels = voltage_levels,tag = 'pwr_avg', options =
options)
scn_static = db.create_scenario_view(power_view =
pwr_avg,scenario_type = 'static',voltage_levels = voltage_levels,tag =
'scn_static', options = options)
```

4.3 SAIF file based static analysis

Figure 5 describes how SAIF file based Static flow can be performed. The SAIF file can be used to generate switching activity view; it needs to be a Gate-level SAIF file. User control settings are used where there is no SAIF coverage (see Section 4.2, “User controlled toggle rate driven static analysis”).

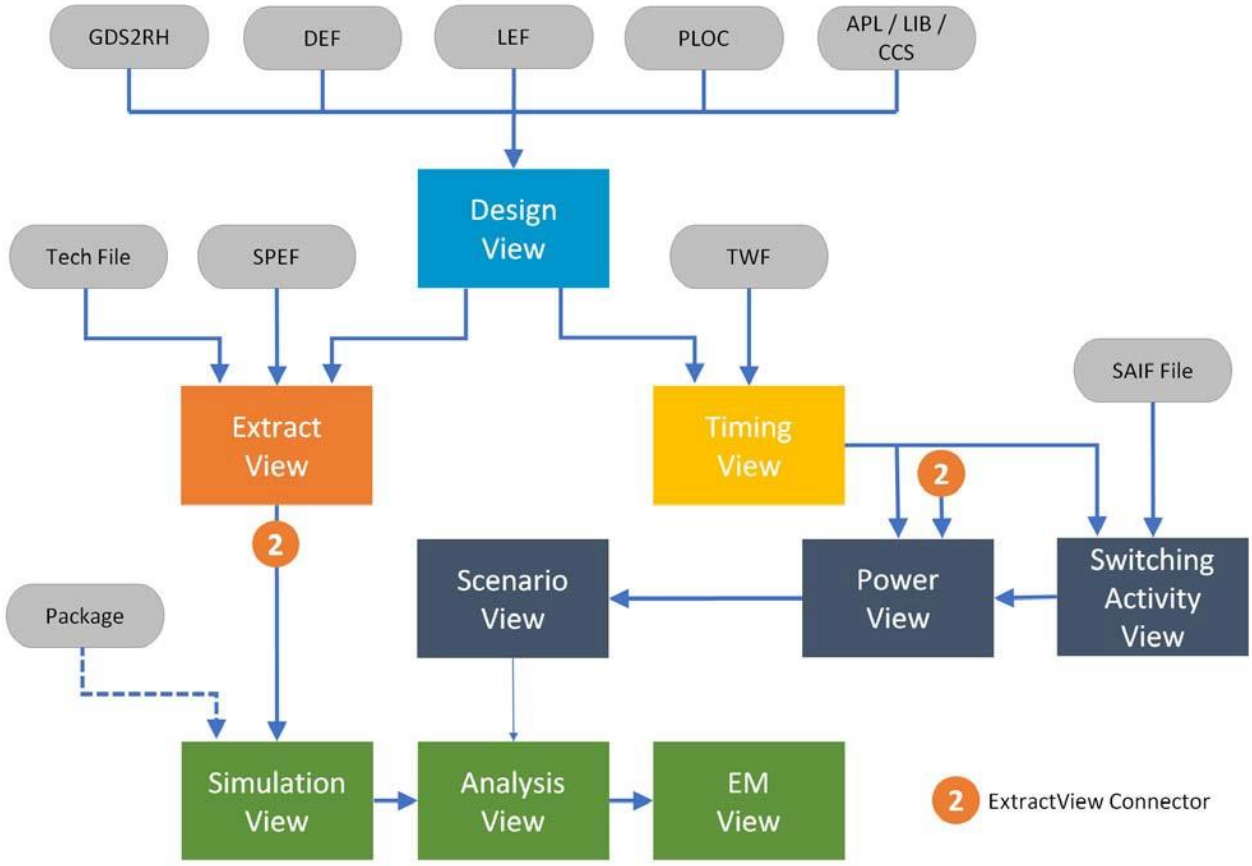

Figure 5. SAIF file based Static Flow

Flow setup

The following command lines are the typical steps to use in a SAIF file flow.

```
swa_settings = {'default_sequential_output_pin_toggle_rate' : 0.5,
'default_clock_period' : 2.564102564102564e-09,
'default_clock_pin_toggle_rate' : 2.0,
'default_combinational_pin_toggle_rate' : 0.50,
'default_icg_output_pin_toggle_rate' : 1.3,
'default_macro_output_pin_toggle_rate' : 0.50,
'default_primary_input_port_toggle_rate' : 0.15
}
saif_files = [{'file_name': 'Input.saif', 'preamble' : 'TESTBENCH'}]

swa_prop = db.create_switching_activity_view(tv,saif_files =
saif_files,settings = swa_settings,propagation_style =
'assign_default_activity',tag = 'swa_prop', options = options)

pwr_avg =
db.create_power_view_from_switching_activity(swa_prop,extract_view =
```

```
ev,external_parasitics = evx,timing_view = tv,temperature =
25.0,voltage_levels = voltage_levels,tag = 'pwr_avg', options =
options)

scn_static = db.create_scenario_view(power_view =
pwr_avg,scenario_type = 'static',voltage_levels = voltage_levels,tag =
'scn_static', options = options)
```

4.4 VCD/FSDB mode static analysis

For VCD/FSDB, the **value_change_view** needs to be created from the VCD/FSDB files in the flow, as shown in Figure 6. For the blocks where there are no VCD or FSDB files, the user-controlled toggle rate is used to drive the toggle rate. SAIF and VCD/FSDB can't be mixed together. RTL or GATE level VCD/FSDB can be mixed in different blocks and Tick based or Timing based can be mixed in different blocks as well. If IPF is defined to create the power view, none of those settings, such as user-controlled toggle rate, SAIF, VCD, or FSDB can be use with the IPF file to drive the power view.

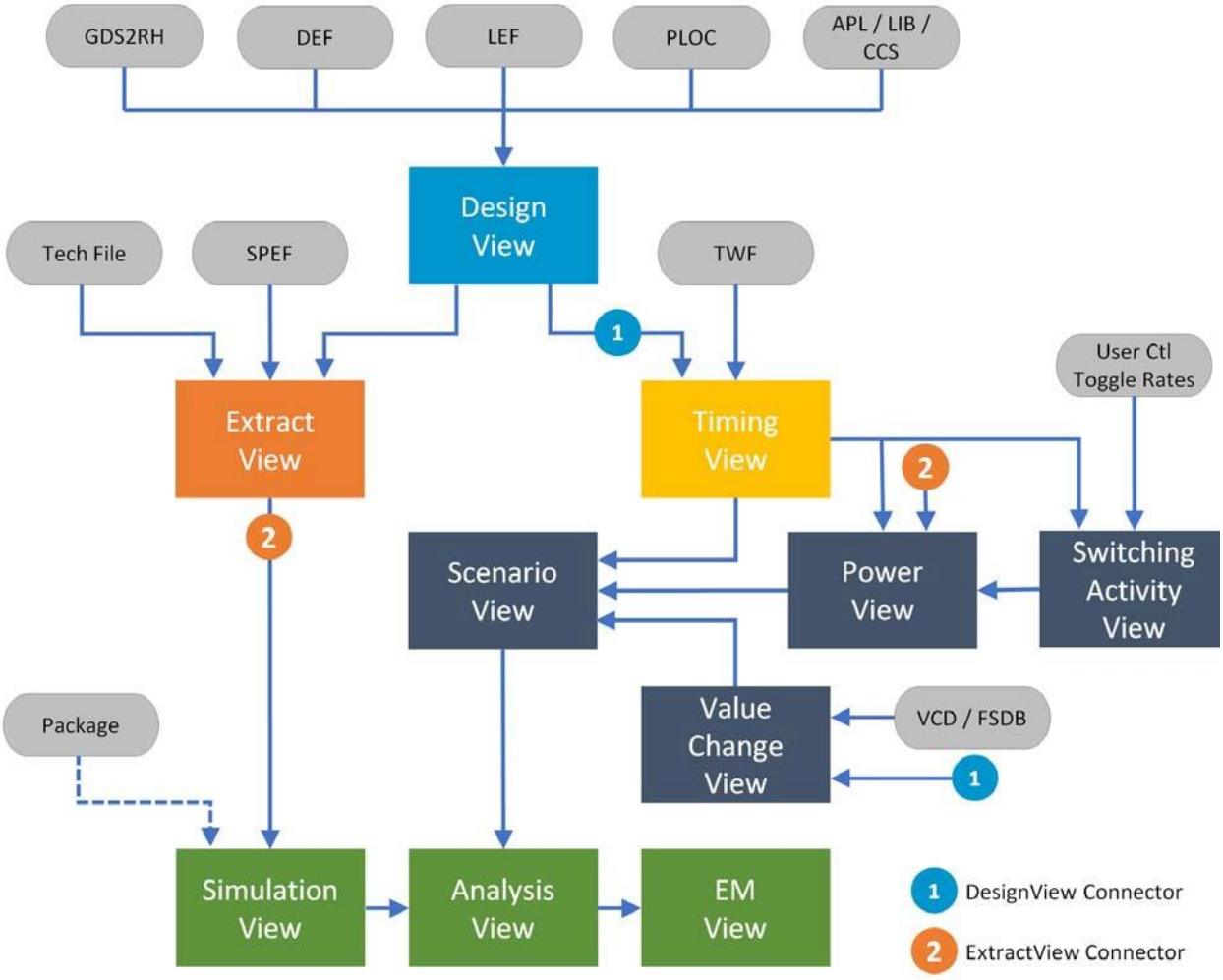

Figure 6. VCD/FSDB Mode Static Analysis Flow

Flow setup

The following command lines are the typical steps to use VCD/FSDB files with user-controlled toggle rate for static
analysis.

```
vcd_files = [ {'file_name' : '../design_data/vcd/GENERIC.vcd',
'preamble' : 'test_bench1/GENERIC/',
'instances' : '',
'time_slices' : [{'slice_name' : 'all', 'start_time' : 1e-12 ,
'stop_time' : 1e-6}]} ]

vcv = db.create_value_change_view(dv,vcd_files = vcd_files,tag = 'vcv', options = options)

vcv_data = [{'view' : vcv, 'slice_name' : 'all', 'instance_prefix' : ''}]
voltage_levels = {'VSS' : 0, 'VDD' : 1.2}

swa_settings =
{'default_sequential_output_pin_toggle_rate' : 0.5,
'default_clock_period' : 2.564102564102564e-09,
'default_clock_pin_toggle_rate' : 2.0,
'default_combinational_pin_toggle_rate' : 0.50,
'default_icg_output_pin_toggle_rate' : 1.3,
'default_macro_output_pin_toggle_rate' : 0.50,
'default_primary_input_port_toggle_rate' : 0.15
}

swa_vcv = db.create_switching_activity_view_from_sim(tv,
value_change_data=vcv_data, tag='swa_vcv',settings = swa_settings)

pwr_avg = db.create_power_view_from_switching_activity(swa_vcv,extract_view =
ev,external_parasitics = evx,timing_view = tv,temperature = 25.0,voltage_levels = voltage_levels,tag =
'pwr_avg', options = options)

scn_static = db.create_scenario_view(power_view = pwr_avg,scenario_type = 'static',voltage_levels =
voltage_levels,tag = 'scn_static', options = options)

sv = db.create_simulation_view(ev,tag = 'sv', options = options)

av_static = db.create_analysis_view(sv,scn_static,tag = 'av_static', options = options)
```

5. Analyzing outputs of SwitchingActivityView and Power View

5.1 SwitchingActivityView VCD / FSDB Annotation Reporting

The API <SwitchingActivityView>.get_coverage_summary() provides vcd coverage summary for SWA. There are no other arguments/options available and the output is as:

```
{Instance(''): {
      'covered_instance_count': 59275,
      'total_instance_count': 59278}}
```

5.2 SwitchingActivityView get_attributes API

The API <SwitchingActivityView>.get_attributes(instance) provides SwitchingActivityView attributes of an Instance

For each input/output pin it provides following attributes :

- clock_period
- one_probability
- source from where the activity was annotated , possible values are in order of increasing priority:
    - **Propagation** : If propagation is turned on, otherwise default activities will be taken. The source can be ConstFromPropagation if, after propagation from inputs, the output gets constant.
    - **DefaultActivity** : If nothing specified for the pin, default activity will be taken
    - **InferIcg :** When infer_icg_enable_signals_from_fanin_prop is True in create_switching_activity_view, some instances will have this source.
    - **ConstFromPropagation** : It is const but from propagation. For example, if a buffer's input is const 0, then we know the output is const 0 from propagation.
    - **ConstFromLso** : const from LSO (Logic Signal Override)
    - **ConstFromSca** : if pin defined as set_case_analysis 0/1 [get_ports pin] in SDC file
    - **ConstFromTiming** : defined as constant in timing file
    - **SaifOrVcd :** If VCD and Saif both are given together, VCD gets the priority
    - **ConstFromLib** : Function of the output pin is defined as 0 or 1 in .lib file
    - **ConstFromPGNet** : Pin is tied to PG net in the design
    - **PinLevelSettings** : Settings from leaf_instance_pin_values
- toggles_per_second
- zero_probability

Example : swa.get_attributes(Instance('inst'))

```
{Pin('A1'): {'clock_period': 7.999999773744548e-09,
             'one_probability': 0.5,
             'source': 'default',
             'toggles_per_second': 18750002.0,
             'zero_probability': 0.5},
```

```
Pin('A2'): {'clock_period': 7.999999773744548e-09,
            'one_probability': 0.5,
            'source': 'default',
            'toggles_per_second': 18750002.0,
            'zero_probability': 0.5},
Pin('ZN'): {'clock_name': 'dco_clk',
            'clock_period': 7.999999773744548e-09,
            'one_probability': 0.5,
            'source': 'default',
            'toggles_per_second': 18750002.0,
            'zero_probability': 0.5}}
```

5.3 SwitchingActivityView get_clock_attributes API

The API <`SwitchingActivityView`>.get_clock_attributes(instance) provides `SwitchingActivityView` clock attributes of an Instance . It provides below information for an instance :

- clock_instance : True / False
- clock_levle
- clocks_reached
- frequency
- logic_level
- observed_at

Example : `swa.get_clock_attributes(Instance('inst'))`

```
{'clock_instance': False,
'clock_level': None,
'clocks_reached': ['clk'],
'frequency': 125000000.0,
'logic_level': None,
'observed_at': (Instance('_43127_'), Pin('ZN'))}
```

5.4 PowerView get_attributes API

The API <`PowerView`>.get_attributes(instance) provides `PowerView` attributes of an Instance

For each Power pin it provides following attributes :

- clock_pin_power
- frequency
- toggle_rate
- internal_power
- leakage_power
- switching_power
- total_power
- voltage
- source from where the power was calculated , possible values are :

- **SwichingActivity** : If power is computed from SWA activity
- **IPF** : If Power source is user IPF
- **ModeControl** : if power is computed from user defined modes via mode_control in object_settings

Example : `pwr.get_attributes(Instance('inst'))`

```
{Pin('VDD'): {'clock_pin_power': 0.0,
              'frequency': 125000000.0,
              'internal_power': 7.325807160896147e-08,
              'leakage_power': 3.6209556242283725e-08,
              'source': 'SwitchingActivity',
              'switching_power': 8.812949090497568e-07,
              'toggle_rate': 0.15000000596046448,
              'total_power': 9.907624871630105e-07,
              'voltage': 1.100000023841858}}
```

5.5 Generating power reports from PowerView

Power reports can be generated using **emir_reports**.

emir_reports.write_instance_power_report_and_summary writes a formatted power report

```
emir_reports.write_instance_power_report_and_summary
```

Arguments:

Argument	Description
`view`	The PowerView or ScenarioView from which to write the report (required)
`file_name_detailed_report`	The file_name for the detailed instance power report, if opted to report (optional. Default value is '`./power.rpt`')
`file_name_summary_report`	The file_name for the summary power report (optional. Default value is '`./power_summary.rpt`')
`columns`	A list of columns to be included in the report (optional. Detailed information on the allowed column names is provided below)
`sort`	Boolean option to determine if the detailed instance power report file, if opted to report needs to be sorted (optional. Default value is True)
`sort_order`	Set to either 'descending' or 'ascending' to determine the order of the sort, if 'sort' option is turned on (optional. Default value is 'descending')

`sort_columns`	A list of columns that are used to sort the detailed instance power report, if 'sort' option is turned on (optional. Values can be any or all of the allowed column names from 'columns' argument. Default value is `['total_power']`)
`header_instance_report`	A string that will be used as a header for the detailed instance power report, if opted to report (optional. Default value is the list of columns reported by default)
`header_summary_report`	A string that will be used as the header in the power summary report (optional. By default, no header is reported)
`format_instance_report`	A Python format[1] string that is used to format each line of the detailed instance power report, if opted to report (optional. See examples below for default formatting)
`format_summary_report`	A Python format[1] string that is used to format each line of the power summary report (optional. See examples below for default formatting)
`max_lines_detailed_report`	The maximum number of lines to include in the detailed instance power report, if opted to report (optional. Default value is 5000)
`write_instance_file`	Boolean option to determine if the detailed instance power report needs to be written out. Please note that this is a **computationally expensive** operation (optional. Default value is `False`)

Table 1 - List of Arguments for `emir_reports.write_instance_power_report_and_summary`

The argument columns accept a list of allowed column names.

Allowed column names for 'columns' argument	**Explanation for the column**
`instance`	The name of the instance in the design
`cell_name`	The Cell Name of the instance in the design
`loc_x`	The Lower Left x-coordinate for the instance bounding box
`loc_y`	The Lower Left y-coordinate for the instance bounding box
`pin`	The Pin name of the Instance for which power is being calculated
`domain`	The domain name for the instance pin

clock_pin_power	The clock pin power for the instance (this is included in the calculated internal_power)
frequency	Frequency of the instance
internal_power	Internal Power for the instance
leakage_power	Leakage Power for the instance
source	Power Source for the pin. Possible sources are SwitchingActivity, IPF (Instance Power File), Scenario and DynamicPower
switching_power	The switching power of the instance
toggle_rate	The toggle rate of the instance
total_power	The total power consumed by the instance
voltage	The ideal voltage level for the instance pin

Table 2: Allowed column names for 'columns' argument of `emir_reports.write_instance_power_report_and_summary`

The order of the column names specified in the list passed to columns argument defines the order of columns in the generated output. The default columns argument is a list of the following column names in the following order.

pin
domain
frequency
toggle_rate
clock_pin_power
internal_power
leakage_power
switching_power
total_power
voltage

cell_name
instance

Table 3: Default order and columns used for `emir_reports.write_instance_power_report_and_summary`

Usage (with default arguments)

```
>>> emir_reports.write_instance_power_report_and_summary(scn)
```

By default, only the Power Summary Report is generated. The default report name is '`./power_summary.rpt`'

```
**** RedHawk-SC Power Summary Report ****

Created: Thu Mar 19 04:51:16 2020

A total of 1185384 instances were summarized for this report while 0 were omitted due to missing power data (100.00% coverage).

A total of 0 pins were omitted because they were not attached to a power domain.

***

grouping            clock_pin_power(W)  internal_power(W)  leakage_power(W)  switching_power(W)  total_power(W)  percent_power(%)  pin_count  instance_count

*** Power Domains:
VDD                      0.024397           0.043581           0.01125           0.071504           0.12634           82.26         912472       909926
core3/VDD_INT            0.001946          0.0080933          0.0033901          0.015767          0.027251           17.74         275458       275458
Total                    0.026343           0.051674           0.01464           0.087272           0.15359          100.00        1187930      1185384

*** Frequency Domains:
1.25e+08                 0.026343           0.051674          0.013597           0.087272           0.15254           99.32         377445       377445
0                           0              6.5505e-08         0.0010437             0              0.0010437          0.68         810485       807939
Total                    0.026343           0.051674           0.01464           0.087272           0.15359          100.00        1187930      1185384

*** User Defined Groups:
combinational logic         0               0.013521          0.0088797          0.080138           0.10254           66.76         310766       308220
sequential logic         0.025916           0.037725          0.0057253          0.007129           0.05058           32.93         71800        71800
memory                 0.00042705         0.00042705          3.544e-05          4.275e-06         0.00046677          0.30           8            8
decap/filler                0                  0                  0                  0                  0              0.00         805356       805356
Total                    0.026343           0.051674           0.01464           0.087272           0.15359          100.00        1187930      1185384

**** End Report ****
```

More Usage Examples

a) Report the detailed instance power file

```
>>> emir_reports.write_instance_power_report_and_summary(scn,
write_instance_file=True, max_lines_detailed_report=None)
```

Reports detailed power information for all instances in the design

```
# clock_pin_power is included in internal_power
# pin     domain  frequency  toggle_rate  clock_pin_power  internal_power  leakage_power  switching_power  total_power  voltage  cell_name              instance
#                   (Hz)                      (W)              (W)             (W)              (W)            (W)        (V)
  VDD     VDD    1.25e+08     2.00           0           3.427e-06      4.865e-07       8.23e-05      8.622e-05    1.10    INV_X32                cts_inv_590761458
  VDD     VDD    1.25e+08     2.00           0           3.028e-06      4.865e-07      8.228e-05      8.579e-05    1.10    INV_X32                cts_inv_528960840
  VDD     VDD    1.25e+08     2.00           0           1.701e-06      2.433e-07      8.182e-05      8.377e-05    1.10    INV_X16                cts_inv_526660817
  VDD     VDD    1.25e+08     2.00           0           1.028e-06       2.16e-07      8.136e-05      8.261e-05    1.10    INV_X16                cts_inv_526160812
  VDD     VDD    1.25e+08     2.00           0           7.824e-06      4.865e-07      7.411e-05      8.242e-05    1.10    INV_X32                cts_inv_589961450
  VDD     VDD    1.25e+08     2.00           0           3.371e-06      4.865e-07      7.797e-05      8.183e-05    1.10    INV_X32                cts_inv_530560856
  VDD     VDD    1.25e+08     2.00           0           5.009e-06      4.321e-07      7.616e-05       8.16e-05    1.10    INV_X32                cts_inv_527960830
  VDD     VDD    1.25e+08     2.00           0           4.208e-06      4.865e-07      7.684e-05      8.154e-05    1.10    INV_X32                cts_inv_557661127
  VDD     VDD    1.25e+08     2.00           0           4.672e-06      4.321e-07      7.616e-05      8.126e-05    1.10    INV_X32                cts_inv_526960820
  VDD     VDD    1.25e+08     2.00           0           1.651e-06       2.16e-07      7.933e-05       8.12e-05    1.10    INV_X16                cts_inv_528460835
  VDD     VDD    1.25e+08     2.00           0           2.663e-06      4.321e-07      7.811e-05       8.12e-05    1.10    INV_X32                cts_inv_529760848
  VDD     VDD    1.25e+08     2.00           0           3.645e-06      4.321e-07      7.623e-05      8.031e-05    1.10    INV_X32                cts_inv_528760838
  VDD     VDD    1.25e+08     2.00           0            3.9e-06       4.865e-07      7.578e-05      8.017e-05    1.10    INV_X32                cts_inv_588961440
  VDD     VDD    1.25e+08     2.00           0           4.171e-06      4.865e-07      7.499e-05      7.965e-05    1.10    INV_X32                cts_inv_558261133

.............
```

b) Customize the columns in the detailed power report file

```
>>> columns = ['pin', 'domain', 'total_power', 'instance']
>>> format_instance_report = '{0:10} {1:10} {2:10.4f} {3}'
>>> header_instance_report = '{0:10} {1:10} {2:10} {3}'.format(*columns)
>>> sort_columns = ['total_power']
>>> emir_reports.write_instance_power_report_and_summary(scn, write_instance_file=True,
sort=True, columns=columns, format_instance_report=format_instance_report,
header_instance_report=header_instance_report, sort_columns=sort_columns)
```

The sorted, customized report will be of the format

```
pin        domain     total_power        instance
VDD        VDD            0.0001         inst0.clock_module_0.clock_gate_dbg_clk.CLKGATETST_X1_1
VDD        VDD            0.0001         inst1.clock_module_0.clock_gate_dbg_clk.CLKGATETST_X1_1
VDD        VDD            0.0001         inst0.clock_module_0.clock_gate_dbg_clk.CLKGATETST_X1_1
VDD        VDD            0.0001         cts_buf_656062111
VDD        VDD            0.0001         core0.openMSP430_inst1.clock_module_0.clock_gate_mclk.CLKGATETST_X1_1
VDD        VDD            0.0001         cts_inv_526160812
VDD        VDD            0.0001         cts_inv_526660817
VDD        VDD            0.0001         cts_inv_527960830
VDD        VDD            0.0001         cts_inv_526960820
VDD        VDD            0.0001         cts_inv_528060831
```

5.6 Exporting IPF file from PowerView

`PowerView.write_instance_power_file()` API can be used to dump IPF file from PowerView

`PowerView.write_instance_power_file(file_name, comment=None)`

ARGUMENTS

Allowed column names for 'columns' argument	**Explanation for the column**
file_name	Output file name (type=str, required=True)
comment	Comment string; will be printed as first line in output file (type=str, default_value=None)

6. Conclusion

This application note covers different ways of power calculation in RedHawk-SC, presents the details of `SwitchingAvtivityView` and `PowerView` , and presents methods to generate power reports in RedHawk-SC and analyzing `SwitchingAvtivityView` results .