CDF

Characterization Description Format

v20.1

Preface

This document is intended for users and developers of Guna

About This Manual

Audience

This manual is targeted for experienced library developers of digital integrated circuits standard cells and custom cells. Such designers are expected to have working knowledge of spice simulation and Tcl/Tk programming on UNIX operating system.

Conventions

`text`	indicates text must be typed as is in guna session.
`:<handle>`	set of arbitrary characters representing in-memory guna objects.
`[..]`	denotes optional arguments
[|]	indicates optional alternative, from which user must chose one.
{ | }	indicates a required arguments, from which user must chose one.

CDF: Characterization Description Format

CDF Interface Commands

cdfi active_inputs

Description: It returns switching or transitioning inputs in the stimulus.

Argument: table:<handle> stimulus:<handle>

See Also:

- cdfi tables - to get table:<handle>
- cdfi stimulus - to get stimulus:<handle>

Example:

```
guna> cdfi active_inputs $table $delay_stim
A
```

cdfi active_outputs

Description: It returns switching or transitioning outputs in the stimulus.

Argument: table:<handle> stimulus:<handle>

See Also:

- cdfi tables - to get table:<handle>
- cdfi stimulus - to get stimulus:<handle>

Example:

```
guna> cdfi active_inputs $table $delay_stim
Y
```

cdfi add_arc

Description:

It adds a model arc (typically) with simulation results. This command is for advanced application. Use with caution.

Argument: stimulus:<handle> sequence:<handle> type:<string> sim_deck:<file> result:<file> value:<float> [value:<float> …]

See Also:

- Stimulus type section later in this chapter.
- cdfi sequence later in this section.

Example:

cdfi add_waveform

Description:

This command is similar to "*cdfi add_arc*" and adds a model arc with simulation results as waveform. This command is for advanced applications. Use with caution.

Argument: stimulus:<handle> sequence:<handle> type:<string> sim_deck:<file> result:<file> arc<:string> slope<:float> time_vec<:list_float> load<:float> y_vector<:list_float>

See Also:

- Stimulus type section later in this chapter.
- cdfi sequence later in this section.

Example:

cdfi ccb

Description:

It returns ccb (channel connected block) definition of the table

Argument: table:<handle>

See Also:

- cdfi tables - to get table:<handle>

Example:

```
guna> cdfi ccb $table
D DFFX1_ccb_D.sp
```

cdfi cell_type

Description:

It returns type of the cell - i.e. comb, ff, sequential etc.

Argument: table:<handle>

See Also:

- cdfi tables - to get table:<handle>

Example:

```
guna> cdfi cell_type $table
comb
```

cdfi clocks

Description:

It returns clock inputs of the cell. It returns null for cell_type combinational.

Argument: table:<handle>

See Also:

- cdfi tables - to get table:<handle>
- cdfi inputs

Example:

```
guna> cdfi clocks $table
MCLK SCLK
```

cdfi data

Description:

It returns data inputs of the cell. In other words, it returns all inputs except clocks.

Argument: table:<handle>

See Also:

- cdfi tables - to get table:<handle>
- cdfi inputs

Example:

```
guna> cdfi inputs $table
A B
```

cdfi function

Description:

It returns data inputs of the cell. In other words, it returns all inputs except clocks.

Argument: table:<handle>

See Also:

- cdfi outputs

Example:

```
guna> cdfi function $table Y
and(A,B)
```

cdfi in_out_arcs

Description:

It returns representing input transition to output transition.

Argument:

table<:handle> stimulus<:handle>

See Also:

- cdfi transitions

Example:

```
guna> cdfi in_out_arcs $table $stimulus
```

cdfi inputs

Description:

It returns all inputs of the cell.

Argument: table:<handle>

See Also:

- cdfi tables - to get table:<handle>
- cdfi outputs

Example:

```
guna> cdfi inputs $table
A B
```

cdfi list_jobs

Description:

It returns list of triplets consisting of input file, command, and output file.

Argument: stimulus:<handle>

See Also:

- cdfi queue_job

Example:

```
guna> cdfi list_jobs $stimulus
```

```
{./file.sp "hspice -i ./file.sp" file.tr0}
```

cdfi noise

Description:

This command returns list of width and height of noise triangles. This input is used to compute input signal triangle waveform for ccs noise propagation model.

Argument: table:<handle> high_voltage<:float> low_voltage<:float>

See Also:

- config ccsn_prop_width_height

Example:

```
guna> cdfi noise $table 0.95 0.0
{{0.462478 0.565067 0.653260} {0.091502 0.045994 0.031775
0.109212 0.054895 0.037925}} {{0.471257 0.360431 0.273122}
{0.013313 0.010966 0.010580 0.011725 0.009659 0.009318}}
```

cdfi outputs

Description:

It returns all inputs of the cell.

Argument: table:<handle>

See Also:

- cdfi tables - to get table:<handle>
- cdfi inputs

Example:

```
guna> cdfi outputs $table
Y
```

cdfi queue_job

Description:

It returns 0 on a successful submission of job to farm queue.

Argument: stimulus:<handle> input:<file> command:<string> output:<file>

See Also:

- cdfi list_jobs
- cdfi remove_job

Example:

guna> cdfi queue_job $stim ./file.sp “hspice -i ./file.sp” file.tr0

cdfi remove_job

Description:

It returns a 0 on a successful removal of job from farm queue.

Argument: stimulus:<handle>

See Also:

- cdfi list_jobs
- cdfi queue_job

Example:

guna> cdfi queue_job $stim ./file.sp “hspice -i ./file.sp” file.tr0

cdfi sequence

Description: It returns a list of input state and output transitions.

Argument: stimulus:<handle>

See Also:

- cdfi stimulus

Example:

guna> cdfi sequence $delay_stim 0
{01,11} r

cdfi simfeed

Description: It returns or sets simulation feed in a stimulus.

Argument: stimulus:<handle> type:<string> [node_value_list:<string>]

See Also:

- cdfi types

Example:

guna> cdfi simfeed $delay_stim delay "v(A)=0.0 v(B)=0.8 v(Y)=0.0"
0
guna> cdfi simfeed $delay_stim delay
v(A)=0.0 v(B)=0.8 v(Y)=0.0

cdfi stimulus

Description:

It returns stimulus scoped in the table. It optionally takes <stimulus> handle to return stimulus sub-scoped to stimulus type. If no stimulus is given, it return all stimulus defined in table scope.

Argument: table:<handle> [<stimulus>:string]

See Also:

- cdfi tables - to get table:<handle>
- cdfi inputs

Example:

```
guna> cdfi stimulus $table
_a0b6450200000000_p_cdf__stimulus
guna> cdfi stimulus $table incap
_a0b6450200000000_p_cdf__stimulus
```

cdfi tables

Description: It returns table:<handle>s. These handles are passed as argument to other cdfi commands.

Argument: cdf:<handle>

See Also:

- read_cdf in guna commands chapter.

Example:

```
guna> set cdf [read_cdf AND2X1.cdf]
guna> set tables [cdfi tables $cdf]
```

cdfi transitions

Description: It returns transitions

Argument: table:<handle> stimulus:<handle> port:<string>

See Also:

Example:

```
guna> cdfi transitions $table $delay_stim A
01
guna> cdfi transitions $table $delay_stim B
11
guna> cdfi transitions $table $delay_stim Y
r
```

cdfi types

Description: It returns types of the stimulus.

Argument: stimulus:<handle>

See Also:

Example:

```
guna> cdfi types $delay_stim 0
delay dpower
```

CDF format

CDF stands for Characterization Description Format. This format

1. captures function, stimulus and other details of circuits.
2. acts as database for cdf interface language called cdfi.

CDF is statically scoped format. Syntactical description of CDF format is below:

```
cdf <string>
      supplies0 <comma-separated-supply-string>;
      supplies1 <comma-separated-supply-string>;
      table <string>
      inputs [<bus-range>] <comma-separated-port-string>;
      clocks [<bus-range>] <comma-separated-port-string>;
      outputs [<bus-range>] <comma-separated-port-string>;
      function
            <function-definition>
      end_function
      ccb <port> <ccb-file>
      stimulus_type_enum
            sim_feed <string>
            model <model_type_enum>
```

```
            stimulus vector
        end_table
end_cdf
```

Where:

<comma-separated-supply-string>

Supply string consists of supply names delimited by '**,**' character.

<bus-range> :

CDF bus notation is similar to verilog, meaning bus-open literal is '**[**', bus-close literal is '**]**' and bus delimiter is '**:**'. Buses are optional in port declaration. These literals are fixed and not configurable.

<comma-separated-port-string> :

Port strings consists of port names and port relationships delimited by '**,**' character. Port relationships are delimited by '**=**' character.

Stimulus Type Enum

CDF supports following enumerated stimulus types.

1. delay
2. tristate
3. incap
    a. it is nldm style input capacitance. It does not model slope, load or mid-transition sensitivity.
4. rcap
    a. This keyword is reserved for more accurate input capacitance characterization. It models following sensitivities.
        i. input slope
        ii. output load
        iii. mid-transition sensitivity. It breaks transition into 2 parts to model in cap.
5. ipower
6. dpower
7. apower
8. lpower
9. maxcap
10. setup
11. hold
12. recovery
13. removal
14. mpw
15. ccsn_vivo
16. ccsn_prop
17. ccsn_volt
18. ccsn_mcap

Model Type Enum

1. delay
2. ccs_driver
3. ccs_receiver
4. tristate
5. incap
6. ipower
7. dpower
8. pg_dcurrent
9. pg_icurrent
10. lcurrent
11. apower
12. lpower
13. maxcap
14. setup
15. hold
16. recovery
17. removal
18. mpw
19. vivo
20. prop
21. volt
22. mcap

CDF Function

CDF function statement is like liberty format for most cells. CDF function is extracted automatically from transistor netlist for digital cells. Occasionally, it fails for custom digital and analog cells. In those cases, stimulus is provided by user. Next section provides CDF function examples for common type of cells.

Examples

Simple Inverter

```
cdf INVX1
        supply0 VSS ;
        supply1 VDD ;
        table INVX1
                inputs A ;
                outputs Y ;
                function
                        Y = (!A)
```

```
            end_function
        end_table
    end_cdf
```

Simple Nand

```
cdf AND2X1
    table AND2X1
        inputs A , B ;
        outputs Y ;
        function
            Y = (!A & !B)
        end_function
    end_table
end_cdf
```

Exclusive OR

```
cdf XOR2X1
    table XOR2X1
        inputs A , B ;
        outputs Y ;
        function
            Y = (A ^ B)
        end_function
    end_table
end_cdf
```

Tri-state Buffer

```
cdf TBUFX1
    table TBUFX1
        inputs A , OE ;
        outputs Y ;
        function
            Y = A
            Y Z = !OE
```

```
        end_function
    end_table
end_cdf
```

Latch

```
cdf TLATX1
    table TLATX1
        inputs D ;
        clocks G ;
        outputs Q , QN ;
        function
            latch(IQ, IQN) {
                enable : "G"
                data_in : "D"
            }
            Q = IQ
            QN = IQN
        end_function
    end_table
end_cdf
```

D Register

```
cdf DFFX1
    table DFFX1
        inputs D ;
        clocks CK ;
        outputs Q , QN ;
        function
            ff(IQ, IQN) {
                clocked_on : "CK"
                next_state : "D"
            }
            Q = IQ
            QN = IQN
        end_function
    end_table
end_cdf
```

D Register Noise Model

```
cdf DFF_X1_QN
    supply1 VDD;
    supply0 VSS, 0;
    table …
    end_table
    table DFF_X1_QN
        inputs N_9_M8_d;
        outputs QN;
        function
            QN = (!N_9_M8_d)
        end_function
        ccb QN DFFX1_QN.cir
        ccsn_vivo ccsn_prop ccsn_volt ccsn_mcap
            state 0-
            model vivo
            model prop
            model volt
            model mcap
            0 , 1 = f ;
        ccsn_volt
            …
            …
    end_table
end_cdf
```

Memory

```
cdf SRAM
    supply1 VDD! ;
    supply0 GND! ;
    table SRAM
        clocks  CLK, CLK_;
        inputs  R_W;
        inputs  [0:4]  A wenable=R_W;
        inputs  [0:4] _A wenable=R_W;
        inputs  [0:7]  D address=A clock=CLK ;
```

```
            outputs [0:7]  W address=A  clock=CLK renable=!R_W ;
            outputs [0:7] _W address=_A clock=CLK_ renable=!R_W;
            function
                memory() {
                    type : ram;
                    address_width : 7;
                    word_width : 16;
                }
            end_function
        end_table
end_cdf
```

CDF Stimulus

Stimulus Vector

Stimulus vector is generated automatically from CDF function statement. Stimulus vector consists of multiple inputs signal states and single outputs transition.

Input Signal States

Signal values are specified by logical values zero (‘0’) and one (‘1’). They are specified in order of their inputs in the beginning of ‘table’. Example of one such input signal values is below:

```
10 , 11 , 00 , 01
```

Output Transitions

Output transitions are specified by following 7 characters.

1. `r:` rising output
2. `f:` falling output
3. `t:` one to hiZ
4. `T:` zero to hiZ
5. `z:` hiZ to zero
6. `Z:` hiZ to one
7. `-:` Don’t care

Example of one such output signal value is below:

```
rf
```

Input signal states and output transitions are combined to create an stimulus. Example of a simple stimulus is given below:

```
10 , 11 , 00 , 01 = rf ;
```

Writing Stimulus Vector

Writing stimulus vector requires knowledge of cell's functionality and switching/non-switching profile of the cell. Typically last two input states combined with output transitions provide arc modeling information.

Delay Vector

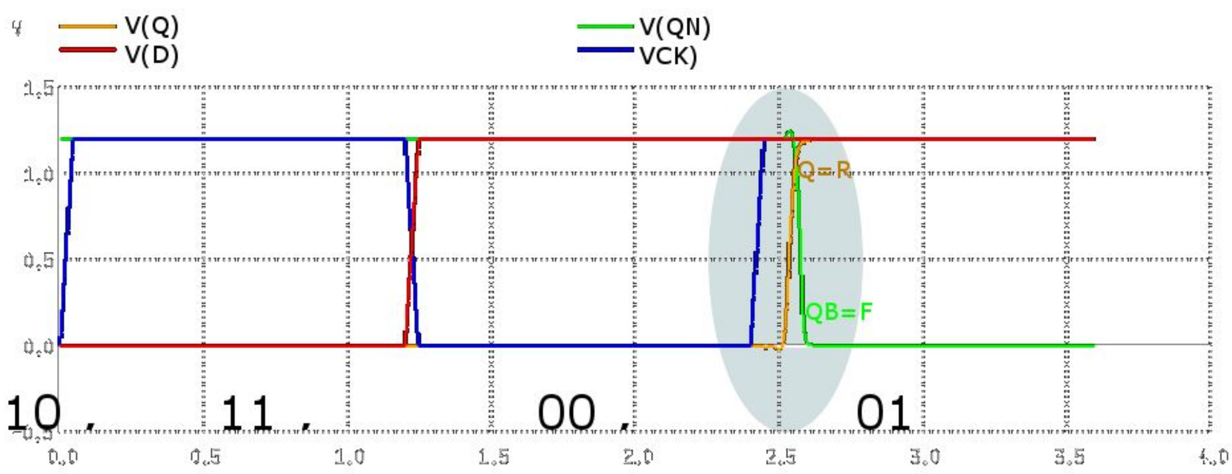
Delay of an arc is defined as time a signal takes to propagate from input to output. Delay captures two quantities:

1. Time difference between signal (typically 50%) thresholds of input to output called cell_rise/cell_fall.
2. Time difference between transition thresholds (typically 10% and 90%) of the output signal called rise_transition/fall_transition.

To understand this, pay attention to a delay stimulus vector for a simple DFF with two inputs (D and CLK) and two outputs (Q and QN) in order:

```
delay
      10 , 11 , 00 , 01 = rf ;
```

Simulation waveform of this vector is shown in below:



In the last two input states (`00 , 01`), input D is non-switching and input CLK is switching. It provides half of the timing arc, i.e. "CLK rising". Output transitions rf, provides the other half, i.e. "Q rising" and "QN falling". Combining these two provides following two delay arcs:

```
model delay CLK r Q  r
model delay CLK r QN f
```

This semantic is shown with shaded area in the picture above. `model` statement is not required. It is used only in cases where automatic inference of timing arc is incorrect for cells

like differential outputs, current mirrors etc.
In summary, delay vector is nothing but a sequence of input states required to switch output(s).

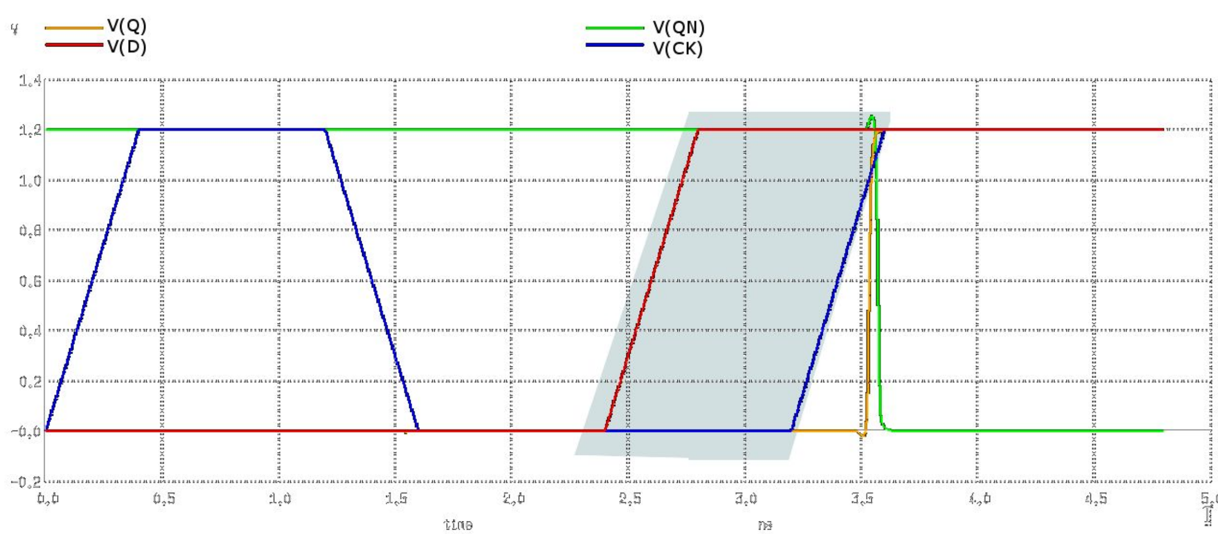
Setup/Hold Vectors

Setup time is the minimum amount of time the data signal should be held steady before the clock event so that the it is reliably sampled by the clock. And Hold time is the minimum amount of time the data signal should be held steady after the clock event so that the data are reliably sampled by the clock. These two parameters (setup and hold) are essentially time required to maintain relationship between data and clock.
To understand this, pay attention to a setup and hold stimulus vector for a simple DFF with two inputs (D and CLK) and two outputs (Q and QN) in order:

```
Setup hold
      00 , 01 , 00 , 11 = rf ;
```

Last two input states (`00 , 11`) show data and clock transition required to cause an event (typically a transition at the output), which would pass/fail depending on the temporal distance between these two input signals. Last but two states before that are used to bring sequential cell to bring in a state to make that event possible. In the example above, input D is referred as data to clock signal CLK. Combining these two provides following two setup arc:

```
model setup D r CK r
```



This semantic is shown with shaded area in the picture above. `model` statement is not required. It is used only in cases where automatic inference of timing arc is incorrect.

Dynamic Power (dpower) Vector

Dynamic power vectors are similar to delay vectors.

Internal Power (ipower) Vector

When input stimulus is applied at the input pins of the cell, capacitors internal to circuit are charged/discharged and resistors converts electrical energy into heat energy. Component of power dissipated inside circuit is called internal power.

Internal power vectors are sequence of input states that cause no output transition. To understand this, pay attention to internal power stimulus vector for a simple DFF with two inputs (D and CLK) and two outputs (Q and QN) in order:

```
ipower
    00 , 01 , 00 , 01 = -- ;
```

Note the transition of clock in last two input states (**00 , 01**). Internal power arc is referred as the input pin(s) switching in last two states.

Appendix 1: Interactive Command Editing

Control key modifiers

`Ctrl + A`	Jump to the beginning of the line
`Ctrl + B`	Move left one character
`Ctrl + C`	Generates SIGINT signal handler
`Ctrl + D`	Delete character ahead of cursor
`Ctrl + E`	Jump to the end of the line
`Ctrl + F`	Move forward one character
`Ctrl + H`	Delete character before cursor
`Ctrl + I`	Represents Tab key, used for command and file completion
`Ctrl + J`	Line feed (Enter/Return) key
`Ctrl + K`	Delete (Cut) characters between cursor and end of line
`Ctrl + L`	Clear screen
`Ctrl + M`	Line feed (Enter/Return) key
`Ctrl + N`	Print next line from history
`Ctrl + P`	Print previous line from history
`Ctrl + Q`	Resumes transmission of characters to guna
`Ctrl + R`	Search backwards in history
`Ctrl + S`	Suspends transmission of characters to guna
`Ctrl + T`	Swap last two characters
`Ctrl + V`	Insert key
`Ctrl + W`	Delete last word
`Ctrl + Y`	Paste (lines cut with CTRL+K)
`Ctrl + Z`	Suspends session.
`Ctrl + [`	Esc key

Escape Sequence

`Esc + B`	Backspace
`Esc + D`	Delete
`Esc + F`	Form feed
`Esc + R`	Kill line
`Esc + T`	Tab