FARADAY TESTER RULER CHECKER PRODUCT

User Guide
Rev.: 200801.1.1v1.0
Issue Date: January 2008

FARADAY
TECHNOLOGY CORPORATION

REVISION HISTORY

FTRC Product User Guide

Date	Rev.	From	To
May 2007	200701.1.1v1.0	-	Original
Jan. 2008	200801.1.1v1.0	-	2008 major release

Printed in Taiwan 2008

Faraday Technology Corporation
No. 5, Li-Hsin Road III, Hsinchu Science Park, Hsinchu City, Taiwan 300, R.O.C.

Faraday's home page can be found at:
http://www.faraday-tech.com

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

The **FTRC** product user guide, including the **tpre**, **ftrc**, **ftl2ver** tools, is mainly to smooth the generation of *ftl* patterns. This document focuses on how to use the Faraday design kit to generate the *ftl* patterns (**ftrc**) and translate the *ftl* pattern to the test bench (**ftl2ver**).

The functions of FTRC product include:

- Translate the *fsdb/vcd/tabular/wgl* files to the *ftl* pattern, and detect the violations against the rules of the testers (**ftrc**)
- Convert FTL (Faraday Tester interface Language) to Verilog HDL test bench (**ftl2ver**)

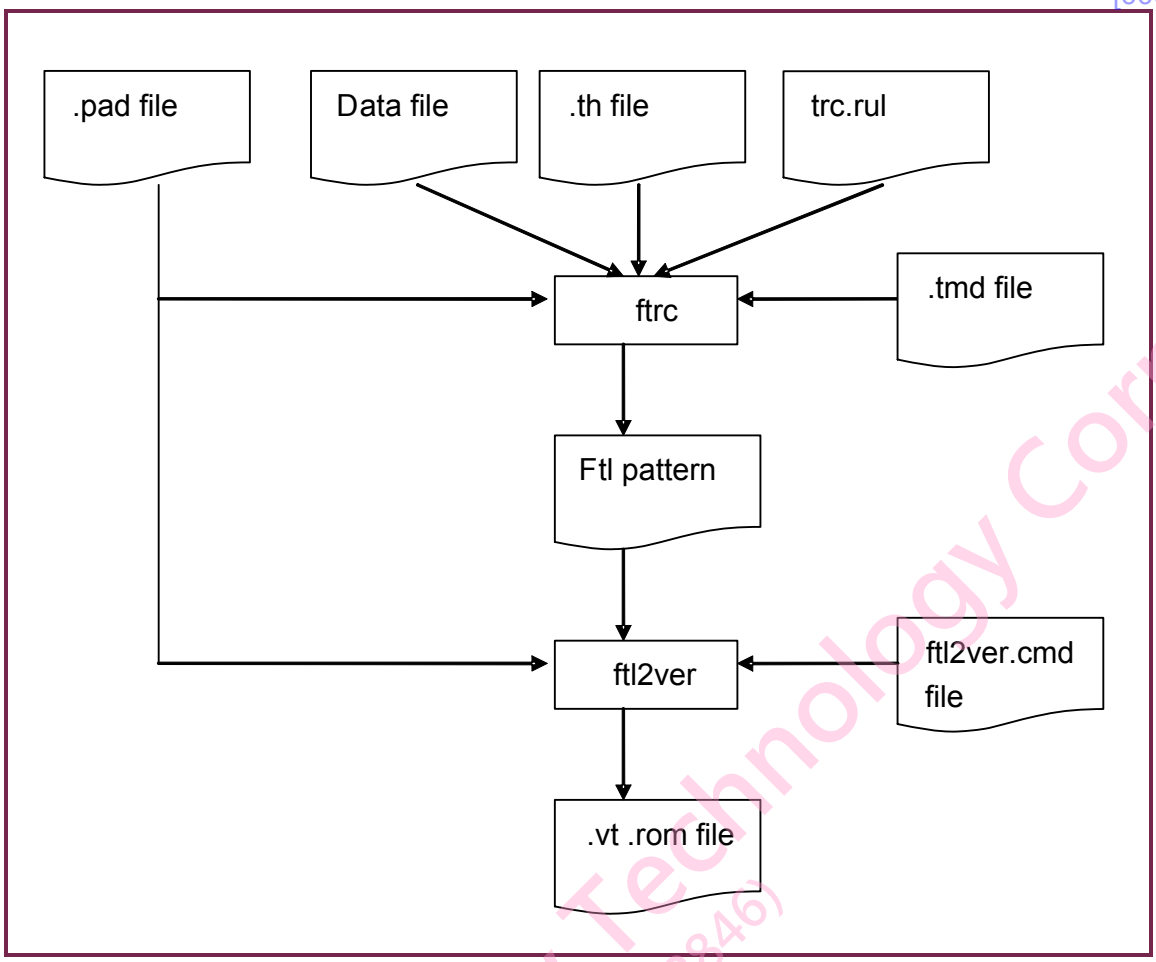


Figure 1-1. Operation Flow of FTRC Product

Chapter 2

tpre Command Syntax

The **tpre** tool includes the following commands:

```
-HEADER      header_file
-INPUT       pad_file
[-LOG        log_file]
```

-HEADER: Specify the name of the output header file

-INPUT: Specify the pad file generated by users

-LOG: Specify the output log file

Chapter 3

ftrc Command Syntax

The **ftrc** tool includes the following commands:

-HEADER header_file
-INPUT data_file
[-LOG log_file]
[-REPORT report_file]
[-OUTPUT output_file]
[-FORMAT FSDB|VCD|TABULAR|WGL]
[-TESTER DEFAULT|FREE]
[-MAXERROR max_errors]
[-MAXWARNING max_warning]
[-MAXREPEATMSG max_repeat_message]
[-CASETOUPPER ON|OFF]
[-CASESENSITIVITY ON|OFF]
[-NOWARNING]
[-FLOAT 0|1]
[-V Show the current **ftrc** version]
[-H Show this message]

-HEADER	Specify the header file name generated by **tpre**
-INPUT	Specify the name of the input data file, the file format could be *fsdb*, *vcd*, *tabular*, or *wgl*
-LOG	Specify the name of the output log file
-REPORT	Specify the name of the output report file
-OUTPUT	Specify the name of the output *ftl* file
-FORMAT	Specify the input data file format, the file format could be *fsdb* or *vcd*
-TESTER	Specify the target tester rule
-MAXERROR	Specify the maximum acceptable number of errors. If the number of error message is more than MAXERROR, the program will stop immediate.
-MAXWARNING	Specify the maximum acceptable number of warnings. If the number of warning messages is more than MAXERROR, the program won't stop, but the output log file will ignore the warning messages afterwards.
-MAXREPEATMSG	Specify the maximum acceptable number of the same messages. If the number of the same message is more than MAXREPEATMSG, the program won't stop, but the output log file will ignore the messages afterwards.
-CASETOUPPER	Convert the letter case of all the signal names of the input files to the upper case
-CASESENSITIVITY	Perform the comparison of all the signal names listed in the input files, it will be performed in a case-sensitive manner.
-NOWARNING	Ignore the warning messages
-FLOAT	Set the input of don't care ("X") or high impedance ("Z") as Boolean 0 or Boolean 1, and report the warning messages
-V	Show the current version of FTRC product
-H	Show the help messages

Chapter 4

ftl2ver Command Syntax

The **ftl2ver** tool includes the following commands:

```
-i              <input_ftl_file>
-f              [VERILOG]
-t              <top_module>
-ControlPad     <pad_file>
-tester         [DEFAULT|FREE]
-ruleCheck
-casesensitivity [ON|OFF]
-casetoupper    [ON|OFF]
```

where

-i: Specify the name of the input *ftl* file

-f: Specify the simulator

-t: Indicate the name of the top module

-ControlPad: Specify the pad file

-tester: Specify the tester rule

-ruleCheck: Enable the rule check

-CASETOUPPER: Convert the letter case of all the signal name of the input files to the upper case

-CASESENSITIVITY: Perform the comparison of all signal names listed in the input files, the comparison will be performed in a case-sensitive manner.

Chapter 5

tpre Input/Output Files

This chapter contains the following sections:

5.1 Input Files

Input File Name	Description
<DESIGN>.pad	Pad cell information of the design
tpre.cmd	This is the command file. This file holds the options and command arguments of the **ftrc** program. For more details, please refer to Chapter 2.

5.2 Output File

Input File Name	Description
<DESIGN>.th	This is the header file. This file contains the signal and timing definitions necessary for the conversion of the test data.

Chapter 6

ftrc Input/Output Files

This chapter contains the following sections:

6.1 Input Files

Input File Name	Description
<id>.dmp	This is the *vcd* dump file of the simulation result. The extension *.dmp* stands for the "vcd format." This file is required only when TRC_FORMAT = vcd.
<id>. fsdb	This is the *fsdb* dump file of the simulation result. The extension *.fsdb* stands for the "Debussy fsdb format." This file is required only when TRC_FORMAT = fsdb.
<id>.th	This is the header file. It contains the signal and timing definitions necessary for the conversion of the test data. This file is required when TRC_FORMAT = vcd or fsdb.
<id>.pad	This is the pad file. It contains the information of the I/O pad and the control pin of the bi-directional pad.
<id>.tmd	This is the command file. This file holds the options and command arguments of the **ftrc** tool. For more details, please refer to Chapter 3
<id>.tab	Tabular data of the simulation results The extension .tab stands for the "tabular format." It is a "print on change" output of the simulation. This file is required only when TRC_FORMAT = tabular.
<id>. wgl	This is the *wgl* test pattern file generated by ATPG tool. This file is required only when TRC_FORMAT = wgl.

6.2 Output Files

Input File Name	Description
<id>.ftl	This is the generated *ftl* test pattern.
<id>.trp	This is the signal and waveform analysis report file.
<id>.tlg	This is the log file that records the warning and error messages occurred during the **ftrc** execution.

Chapter 7

ftl2ver Input/Output Files

This chapter contains the following sections:

7.1 Input Files

Input File Name	Description
<id>.ftl	This is the generated *ftl* test pattern.
<id>.pad	This is the pad file. It contains the information of the I/O pad and the control pin of the bi-directional pad.
ftl2cae.cmd	This is the command file. This file holds the options and command arguments of the **ftrc** tool. For more details, please refer to Chapter 4

7.2 Output Files

Input File Name	Description
<id>.vt	This is the Verilog stimulus file written in the HDL format. This .vt file handles the waveform formats of all the signals and performs the comparison between the simulation results and the expected values of the output signals. A ROM table file (*.rom*) as described below is required when running the simulations.
<id>.rom	This is the ROM table file. It describes the input values and the expected output values that come from the *ftl* file in the cycle-based representations.
<id>.flg	This is the execution log file.

7.3 Test Data of ftl Format

The examples of the **ftl2ver** input/output files, such as the *ftl* file, Verilog stimulus, and the ROM file, are illustrated below. Users can easily understand the relationships among these files through the provided examples.

The example below shows the original *ftl* file.

```
TESTTYPE FUNC;

INPUT(1)      A0,A1;
INPUT(3)      DISTRN,DOSTRN;
INPUT(4)      RCLK;
OUTPUT(2)     BAUDOT,DDIS ;
INOUT(1,2)    D0,D1,D2,D3,D4,D5,D6,D7 ;

TIMEUNIT      1 NS;
CYCLE         100;
TIMEGEN(1)    DNRZ,0;
TIMEGEN(3)    DNRZ,30;
TIMEGEN(4)    RZ,25,50;
TIMEGEN(2)    STROBE,50,10;
SEQUENCE      A0,A1,DISTRN,DOSTRN,,RCLK,,
              BAUDOT,DDIS,,D0,D1,D2,D3,D4,D5,D6,D7;
BEGIN
0000_1_XX_XXXXXXXX;      // 1 : 0
0000_0_HH_11110000;      // 2 : 1000
1100_0_HL_XXXXXXXX;      // 3 : 2000
1100_1_HL_LLLLHHHH;      // 4 : 3000
0011_0_HH_XXXXXXXX;      // 5 : 4000
0011_0_LL_00001111;      // 6 : 5000
0000_0_LL_XXXXXXXX;      // 7 : 6000
0000_0_LL_HHHHLLLL;      // 8 : 7000
END
```

Figure 7-1. ftl Test Vector: tpl.ftl

The example below shows the generated *vt* file.

```
// Date: Mon Aug 18 14:33:48 2003
// Creator: FTC
// Top Module: TOP


`ifdef fsim_typical_corner
    `define  result_file  "n_tp1.out"
    `define  tog_file  "n_tp1.tog"
    `define  cfl_file  "n_tp1.cfl"
    `define  flt_file  "n_tp1.flt"
    `define  dmp_file  "n_tp1.dmp"
    `define  fsdb_file "n_tp1.dmp.fsdb"
`endif

`ifdef fsim_best_corner
    `define  result_file  "b_tp1.out"
    `define  tog_file  "b_tp1.tog"
    `define  cfl_file  "b_tp1.cfl"
    `define  flt_file  "b_tp1.flt"
    `define  dmp_file  "b_tp1.dmp"
    `define  fsdb_file "b_tp1.dmp.fsdb"
`endif

`ifdef fsim_worst_corner
    `define  result_file  "w_tp1.out"
    `define  tog_file  "w_tp1.tog"
    `define  cfl_file  "w_tp1.cfl"
    `define  flt_file  "w_tp1.flt"
    `define  dmp_file  "w_tp1.dmp"
    `define  fsdb_file "w_tp1.dmp.fsdb"
`endif

`timescale 10ps/1ps
module TOP_sim;
parameter
          cycle              = 10000,
          input_pattern_file = "tp1.rom",
          pattern_width      = 23,
          expect_width       = 10,
          input_vectors      = 8;
integer index,m,n,outfile,i;
```

```
wire [0:pattern_width] ERR;
reg [pattern_width-1:0] rom[0:input_vectors-1];


// Simulation non-bus I/O port
wire     A0, A1, DISTRN, DOSTRN, RCLK
         , BAUDOT, DDIS, D0, D1, D2
  , D3, D4, D5, D6, D7;

// ROM non-bus I/O port
wire     A0_i, A1_i, DISTRN_i, DOSTRN_i, RCLK_i
         , BAUDOT_e_i, DDIS_e_i, D0_i, D0_e_i, D1_i, D1_e_i, D2_i, D2_e_i, D3_i,
D3_e_i
         , D4_i, D4_e_i, D5_i, D5_e_i, D6_i, D6_e_i, D7_i, D7_e_i;

// Input assignment
assign   {A0_i, A1_i, DISTRN_i, DOSTRN_i, RCLK_i
         , BAUDOT_e_i, DDIS_e_i, D0_i, D0_e_i, D1_i
         , D1_e_i, D2_i, D2_e_i, D3_i, D3_e_i
         , D4_i, D4_e_i, D5_i, D5_e_i, D6_i
         , D6_e_i, D7_i, D7_e_i } = rom[m];


// Add input offset
DNRZ #(10000,0)          _I_A0(A0,A0_i);
DNRZ #(10000,0)          _I_A1(A1,A1_i);
DNRZ #(10000,3000)       _I_DISTRN(DISTRN,DISTRN_i);
DNRZ #(10000,3000)       _I_DOSTRN(DOSTRN,DOSTRN_i);
RZ   #(10000,2500,5000)   _I_RCLK(RCLK,RCLK_i,m);
DNRZ #(10000,0)          _I_D0(D0,D0_i);
DNRZ #(10000,0)          _I_D1(D1,D1_i);
DNRZ #(10000,0)          _I_D2(D2,D2_i);
DNRZ #(10000,0)          _I_D3(D3,D3_i);
DNRZ #(10000,0)          _I_D4(D4,D4_i);
DNRZ #(10000,0)          _I_D5(D5,D5_i);
DNRZ #(10000,0)          _I_D6(D6,D6_i);
DNRZ #(10000,0)          _I_D7(D7,D7_i);


// Add output offset
STROBE  #(10000,4500,2000)   _O_BAUDOT(ERR[5],BAUDOT,BAUDOT_e_i);
STROBE  #(10000,4500,2000)   _O_DDIS(ERR[6],DDIS,DDIS_e_i);
STROBE  #(10000,4500,2000)   _O_D0(ERR[7],D0,D0_e_i);
STROBE  #(10000,4500,2000)   _O_D1(ERR[8],D1,D1_e_i);
```

```
STROBE  #(10000,4500,2000)    _O_D2(ERR[9],D2,D2_e_i);
STROBE  #(10000,4500,2000)    _O_D3(ERR[10],D3,D3_e_i);
STROBE  #(10000,4500,2000)    _O_D4(ERR[11],D4,D4_e_i);
STROBE  #(10000,4500,2000)    _O_D5(ERR[12],D5,D5_e_i);
STROBE  #(10000,4500,2000)    _O_D6(ERR[13],D6,D6_e_i);
STROBE  #(10000,4500,2000)    _O_D7(ERR[14],D7,D7_e_i);


// Main body of stimulus file
// initial $dcalc_path(top,"","");
// initial $sdf_annotate("TOP.sdf",top,,,,,"FROM_MTM");

// Dump file
`ifdef fsim_dump
initial
  begin
    $dumpfile(`dmp_file);
    $dumpvars;
  end

`endif
// Debussy waveform display
`ifdef fsim_fsdb
initial
  begin
     $fsdbDumpfile(`fsdb_file);
     $fsdbDumpvars;
  end

`endif

// Input assignment
initial
   begin
     // Optional PLI control
     `ifdef toggle
      $toggle_rate_begin(0,`tog_file,TOP_sim);
     `endif
     `ifdef content
       $bus_content_begin(0,`cfl_file,TOP_sim);
     `endif
     `ifdef float
       $bus_float_begin(10000,1,`flt_file,TOP_sim,"TOP.fbs");
```

```
        `endif
        $readmemb(input_pattern_file,rom);
        for(m=0;m<input_vectors;m=m+1)
        begin
            #cycle;
        end
        `ifdef fsim_dump
          $dumpflush;
        `endif
        // Optional PLI control
        `ifdef toggle
          $toggle_rate_end;
        `endif
        `ifdef content
          $bus_content_end;
        `endif
        `ifdef float
           $bus_float_end;
        `endif
        $fdisplay(outfile,"====================================");
        $fdisplay(outfile,"Normal Completion");
        $finish;
    end


initial
    begin
       outfile = $fopen(`result_file);
       $fdisplay(outfile,"O E SignalName      Cycle_no");
       $fdisplay(outfile,"====================================");
    end

always @(ERR[5])
if($time!=0) $fdisplay(outfile,"%b %b BAUDOT     (%d)",BAUDOT,BAUDOT_e_i,
$time/cycle+1);
always @(ERR[6])
if($time!=0)  $fdisplay(outfile,"%b %b DDIS
(%d)",DDIS,DDIS_e_i,$time/cycle+1);
always @(ERR[7])
if($time!=0)  $fdisplay(outfile,"%b %b D0       (%d)",D0,D0_e_i,$time/cycle+1);
always @(ERR[8])
if($time!=0)  $fdisplay(outfile,"%b %b D1       (%d)",D1,D1_e_i,$time/cycle+1);
always @(ERR[9])
```

```
if($time!=0)  $fdisplay(outfile,"%b %b D2       (%d)",D2,D2_e_i,$time/cycle+1);
always @(ERR[10])
if($time!=0)  $fdisplay(outfile,"%b %b D3       (%d)",D3,D3_e_i,$time/cycle+1);
always @(ERR[11])
if($time!=0)  $fdisplay(outfile,"%b %b D4       (%d)",D4,D4_e_i,$time/cycle+1);
always @(ERR[12])
if($time!=0)  $fdisplay(outfile,"%b %b D5       (%d)",D5,D5_e_i,$time/cycle+1);
always @(ERR[13])
if($time!=0)  $fdisplay(outfile,"%b %b D6       (%d)",D6,D6_e_i,$time/cycle+1);
always @(ERR[14])
if($time!=0)  $fdisplay(outfile,"%b %b D7       (%d)",D7,D7_e_i,$time/cycle+1);
// Top module instance

TOP top(.A0(A0), .A1(A1), .DISTRN(DISTRN), .DOSTRN(DOSTRN)
          , .RCLK(RCLK), .BAUDOT(BAUDOT), .DDIS(DDIS), .D0(D0), .D1(D1)
          , .D2(D2), .D3(D3), .D4(D4), .D5(D5), .D6(D6)
          , .D7(D7));
endmodule

module DNRZ(out, in);
input in;
output out;
parameter cycle=10, Td = 10;
  assign  #Td out = in;
endmodule


module STROBE(error, sim, exp);
input sim,exp;
output error;
parameter cycle = 10, Tsd=80, Tsw=10;
reg enable, error;
initial
begin
  error = 0;
  enable = 0;
  while (1)
    begin
      #Tsd enable = ~enable;
      #Tsw enable = ~enable;
      #(cycle - Tsd - Tsw);
    end
 end
```

```
always @(sim or enable)
  if ((enable == 1'b1) & (exp !== 1'bx) & (sim !== exp))
     error = ~error;
endmodule

module RZ(out, in, cycle_begin);
input in, cycle_begin;
output out;
reg out;
parameter cycle = 10, Td = 10, Tp = 20;
always @(cycle_begin)
begin
      case (in)
             1'b0: out = 0;
             1'b1: begin
                    out = 0;
                    #Td out = ~out;
                    #Tp out = ~out;
                    end
             1'bz: out = 1'bz;
          default : out = 1'bx;
      endcase
end
endmodule
```

Figure 7-2. Generated vt File

```
00001ZZZZZZZZ_XXXXXXXXXX
0000011110000_1111110000
11000ZZZZZZZZ_10XXXXXXXX
11001ZZZZZZZZ_1000001111
00110ZZZZZZZZ_11XXXXXXXX
0011000001111_0000001111
00000ZZZZZZZZ_00XXXXXXXX
00000ZZZZZZZZ_0011110000
```

Figure 7-3. Generated ROM File

Chapter 8

Generate Header with tpre

This chapter contains the following sections:

8.1 Prepare *.pad* File

To generate the header file template and to use the design kits afterwards, users must prepare the pad file before using the **FTRC** product.

Below is an example of the pad list file:

```
Instance Name          Cell Name   Connected Net   Port Name    Cell Type    Control Net
-------------------------------------------------------------------------------------
VMDLY_IN_0_INPAD       XFAB         VMDLY_IN_0     VMDLY_IN_0   IP           --
VLDLY_IN_3_IOPAD       ZFA2GSB      VLDLY_IN_3     VLDLY_IN_3    BP          n35
VLDLY_IN_2_IOPAD       ZFA2GSB      VLDLY_IN_2     VLDLY_IN_2   BP           n40
```

Each column represents:

- Instance Name: The name of the I/O instances
- Cell Name: The name of the I/O cells
- Connected Net: The corresponding net connected from the I/O cell to the corresponding pin of a chip
- Control Net: The net that connects to the E or EB pin of I/O cells

8.2 Generate Header File Template

Edit the command file *tpre.cmd* and generate a template header file with the following command:

tpre.lnx tpre.cmd

tpre will generate a header with respect to the pins defined in the pad file. An example of the header generated by **tpre** is shown below:

```
// Header File generated by TPRE
// Creation Date: May 3 18:42:03 2007
// Pad File: FSA0AC078A.pad
// Please modify this file to make it complete!

TESTTYPE FUNC;

// ------------------------------------
// Please fill timegen number for each pin
// ------------------------------------
INPUT( )  X_RESETB;
INPUT( )  X_TESTEN;
INPUT( )  X_TIN1;
INPUT( )  X_TIN0;
INPUT( )  X_RD;
INPUT( )  X_DI1;
```

```
OUTPUT( ) X_READY;
OUTPUT( ) X_CSPRB;
OUTPUT( ) X_CSPDA;
OUTPUT( ) X_CSPCL;
OUTPUT( ) X_DO0;
OUTPUT( ) X_DO1;
OUTPUT( ) X_OSCCLK;
INOUT( , )     VBG_TA;
OUTPUT( ) VBG_VREF;

// ------------------------------------------------
// Please dispatch INOUT pins to correct OUTIF0/OUTIF1
// ------------------------------------------------
   OUTIF1 VBG_TA;

// ----------------------------------------
// Please make your own waveform generated by
// the reference of 4 TIMEGEN examples
// ----------------------------------------
// TIMEUNIT    1 NS;
// CYCLE  1000;
// TIMEGEN( )  DNRZ, 0;
// TIMEGEN( )  RZ, 0, 10;
// TIMEGEN( )  RO, 0, 10;
// TIMEGEN( )  STROBE, 0, 10;

SEQUENCE  X_RESETB,X_TESTEN,X_TIN1,X_TIN0,X_RD,X_DI1,X_READY,X_CSPRB,
          X_CSPDA,X_CSPCL,X_DO0,X_DO1,X_OSCCLK,VBG_TA,VBG_VREF;

// ------------------------------------------------
// Please fill correct setting for your tabular file
// ------------------------------------------------
TABULAR
 // TAB_TIMEUNIT   10 PS;
 START_LINE    1;
 END_LINE $;
 // FORMAT     "$a_time $timeunit $states";
END_TABULAR
```

Chapter 9

ftl Generation with ftrc

This chapter contains the following sections:

The *ftl* generation flows of different input data formats and each step in these flows will be described later in this section.

- Tabular to generate *ftl*
  1. Generate the simulation dump data
  2. Prepare the header file
  3. Prepare the command file
  4. Execute the **trc.lnx <id>.tmd**
  5. Examine the output data

- vcd to generate *ftl*
  1. Generate the simulation dump data
  2. Prepare the header file
  3. Prepare the pad file
  5. Prepare the command file
  6. Execute the **trc.lnx <id>.tmd**
  7. Examine the output data

- fsdb to generate *ftl*
  1. Generate the simulation dump data
  2. Prepare the header file
  3. Prepare the pad file
  4. Prepare the command file
  5. Execute the **trc.lnx <id>.tmd**
  6. Examine the output data

- wgl to generate *ftl*
  1. Generate the ATPG *wgl* pattern
  2. Prepare the command file
  3. Execute the **trc.lnx <id>.tmd**
  4. Examine the output data

For more clear understanding on the *ftl* generation flows, the following example is presented to explain each step in a translating flow.

Assume we have a Verilog test pattern as follows:

```
module TEST_PATTERN1;
…
   MY_DESIGN TOP (ACKN, ADI, BIS, CK18M, PSIORN, PSIOWN, ADO, BDIRO, CHRDY, CK18MO, D0, D1,
   D2, D3, D4, D5, D6, D7);
…
endmodule
```

Figure 9-1 is the top view of a design in the test bench. The instance name of this design in the test bench is TOP. There are six input pins, four output pins, and eight bi-directional pins in the TOP.

Figure 9-1. Design Example of the ftrc Flow

9.1 Generation of the Simulation Dump Data

ftrc reads the simulation dump data, in the tabular, vcd, or fsdb format to create the *ftl* test pattern. To generate the simulation dump data, users should add the statements in the Verilog test bench.

9.1.1 Generate the Tabular Data

The statement of generating the "print on change" tabular file of TOP is as follows:

```
$fmonitor(“tt.tab”, $time, , ACKN, ADI, BIS, CK18M, PSIORN, PSIOWN, ADO, BDIRO, CHRDY, CK18MO,
D0, top.D0EN, D1, top.D1EN, D2, top.D2EN, D3, top.D3EN, D4, top.D4EN, D5, top.D5EN, D6,
top.D6EN, D7, top.D7EN);
```

In the statement, "*tt.tab*," the file name of the output tabular file and all the input/output/bi-directional pin names are listed in this file. Please note that there are eight additional control signals listed in the statement (*top.D0EN, top.D1EN, top.D2EN, top.D3EN, top.D4EN, top.D5EN, top.D6EN,* and *top.D7EN*). There are restrictions to the tabular format file as described in the following:

For the bi-directional signal, both the signal itself and the related control node must be monitored. The corresponding control node must follow the bi-directional signal in the $fmonitor statement. (For example, the control node *top.D0E0* of the bi-directional signal *D0* follows *D0.*)

ftrc treats one line of the state values in the tabular file as one action of the value change. Therefore, it is not allowed to set any '\n' (New line) in the $fmonitor statement.

ftrc only recognizes the binary state of the signals. The "vectorized" signals (For example, D[7:0]) should be expanded into the scalar format in the $fmonitor statement.

The following table shows the results of the tabular file "*tt.tab*":

```
0          001011xxxxxxxxxxxxxxxxxxx
3          0010110101x1x1x1x1x1x1x1x1
25         0011110111z1z1z1z1z1z1z1z1
75         00101101010101010101010101
100        00101101101111111111111111
300        11001110110000000000000010
320        11001110011101110110111101
330        11000110010101010101010110
400        11100110010000000000000010
430        11101110110000000000000010
500        11101110110000000000000010
```

9.1.2 Generate the *fsdb* Dump Data

The statements of generating the *fsdb* dump file of TOP are as follows:

```
$fsdbDumpfile("tt.fsdb");
$fsdbDumpvars;
```

The first statement is used to specify which file will be dumped, while the second statement is used to tell the Verilog-XL simulator to dump all the signal transitions into the file specified by the first statement.

9.1.3 Generate the *vcd* Dump Data

The statements of generating the *vcd* dump file of the TOP are as follows:

```
$dumpfile("tt.vcd");
$dumpvars;
```

The first statement is used to specify which file will be dumped, while the second statement is used to tell the Verilog-XL simulator to dump all the signal transitions into the file specified by the first statement.

9.2 Prepare the Header File

In addition to the simulation dump data, **ftrc** also requires the header file for a complete execution. The detailed descriptions and definitions of the header file are illustrated in the Appendix A2.

Based on the example described at the beginning of this chapter, the corresponding header file is as follow:

```
TESTTYPE        FUNC;
INPUT(1)        ACKN, ADI, BIS;
INPUT(3)        PSIORN, PSIOWN;
INPUT(4)        CK18M;
OUTPUT(2)       ADO, BDIRO, CHRDY, CK18MO;
INOUT(1,2)      D0,D1,D2,D3,D4,D5,D6,D7;
OUTIF1          D0,D1,D2,D3,D4,D5,D6,D7;
TIMEUNIT        1 NS;
CYCLE           100;
TIMEGEN(1)      DNRZ,0;
TIMEGEN(3)      DNRZ,30;
TIMEGEN(4)      RZ,25,50;
```

```
TIMEGEN(2)        STROBE,80,10;

SEQUENCE          ACKN, ADI, BIS, CK18M, PSIORN, PSIOWN,
                  ,BDIRO, CHRDY, CK18MO,
                  D0, D1, D2, D3, D4, D5, D6, D7;
TABULAR
              TART_LINE         1;
              END_LINE          $;
              TAB_TIMEUNIT 1ns;
              FORMAT        "$a_time $states";
              END_TABULAR
```

The statements between the TABULAR and END_TABULAR commands are required only when the dump data is in the tabular format.

Since the dump data generated in the previous section is named as tt.xxx, the header file should be saved as "*tt.th.*"

9.3 Prepare the Pad File

When the input dump data is in the *vcd* or *fsdb* format, **ftrc** also requires the pad file of the design to complete the execution.

To generate the pad file, please refer to Chapter 8 for more details.

9.4 Specify the Parameters in the *setup.ftc*

First of all, change TRC_FORMAT to the format of the dump data. The valid options are the *tabular*, *fsbd*, *vcd*, and *wgl*.

If the format of the dump data is *vcd*, the TEST_TOPMODULE must be set accordingly. Based on the example described in the beginning of this chapter, the TEST_TOPMODULE should be set to `TEST_PATTERN1.TOP`.

9.5 Execute the ftrc

Once all the required data are prepared, user can execute the following command to generate the *ftl* pattern:

```
Trc.lnx <id>.tmd
```

9.6 Examine the Output Files

ftrc generates three output files, the execution log file, *ftl* pattern file, and waveform report file. The files listed below are based on the design and the header file illustrated in the previous sections. The corresponding output files are illustrated below.

9.6.1 Execution Log File (*.tlg*)

This file records the messages occurred in the process of **ftrc**. This file includes a summary of the tabular parameter and the error and warning messages while parsing and processing the input files (Header and tabular files).

```
Parsing rule...
Parsing template...
Parsing header...

// ================================================
// SUMMARY OF HEADER PARAMETERS
// ================================================
TABULAR
    TAB_TIMEUNIT 1000 PS;
    START_LINE        1;
                                  END_LINE          $;
    FORMAT            "$A_TIME $STATES";
    TERMINATOR        "";
END_TABULAR

Parsing data...
..wng_dat008 transition within strobe region of control pin of <D7> at cycle<4>
Total warning message : 1
Total error message   : 0
```

Figure 9-2. Execution Log File

9.7 ftl Test Pattern (*.ftl*)

This is an intermediate file in the flow of Faraday's test pattern generation. The *ftl* file describes the signal attributes, timing/waveform, pattern sequence, and cycle-based patterns. The detailed descriptions of *ftl* are illustrated in Appendix A1.

```
// FTC_tRC 1.0
// Date: Tue Jun 27 17:23:17 1995
// Header File: tt.th
// Input File: tt.tab
// Rule File: /ASICAE/FTC/ftclib/ETC/TRC/trc.rul
// Template File: /ASICAE/FTC/ftclib/ETC/TRC/trc.tmp
// Total_Pin_No.: 0018
// Input: 0006 Output: 0004 Bidirection: 0008

TESTTYPE FUNC;

INPUT(1)  ACKN, ADI, BIS ;
INPUT(3)  PSIORN, PSIOWN;
INPUT(4)  CK18M;
OUTPUT(2)  ADO, BDIRO, CHRDY, CK18MO ;
INOUT(1,2)  D0,D1,D2,D3,D4,D5,D6,D7 ;

TIMEUNIT 1 NS;
CYCLE 100;
TIMEGEN(1)  DNRZ,0;
TIMEGEN(3)  DNRZ,30;
TIMEGEN(4)  RZ,25,50;
TIMEGEN(2)  STROBE,80,10;
SEQUENCE     ACKN, ADI, BIS, CK18M, PSIORN, PSIOWN,
             ADO, BDIRO, CHRDY, CK18MO,
             D0, D1, D2, D3, D4, D5, D6, D7 ;

BEGIN
001111LHLHLLLLLLLL; // 1 : 0
001011LHHLHHHHHHHH; // 2 : 100
001011LHHLHHHHHHHH; // 3 : 200
110001HLLHLLLLLLLX; // 4 : 300
111011HLHH00000001; // 5 : 400
END
```

Figure 9-3. *ftl* Test Pattern

9.8 Report File (*.trp*)

This file includes four major sections, the descriptions of the timing generation format, to address the problems of the input signal, output signal, bi-directional signals, and the stable region analysis of the output/bi-directional signals. The corresponding report file and the descriptions of these fields are illustrated as follows.

9.8.1 Input Signal Report

```
================================================================
                          Input Signal Report
----------------------------------------------------------------
NO.          Name     Format            Action   Cycle
----------------------------------------------------------------
1            ACKN     DNRZ, 0.0
2            ADI           DNRZ, 0.0
3            BIS           DNRZ, 0.0
4            CK18M         RZ,  25.0, 50.0
5            PSIORN        DNRZ, 30.0
6            PSIOWN        DNRZ, 30.0

================================================================
```

Figure 9-4. Input Signal Report

The contents of the fourth column (Action) in Figure 9-4 can be either:

- Error: It indicates the waveform recorded in the tabular file does not match the specified timing generation format.
- Adjust: It indicates the waveform recorded in the tabular file does not exactly match the specified timing generation format, but can be adjusted (Or aligned) to the specified timing generation format. Please also refer to the Ttol setting in the following section for more information.

The "Cycle" column lists the occurrence cycles of the corresponding "Action." It covers the following formats:

- 10, 15: Occurred at the 10^{th} and 15^{th} cycles.
- 10 - 15: Occurred from the 10^{th} to the 15^{th} cycles (10, 11, 12, 13, 14, and 15).

9.8.2 Output Signal Report

```
================================================================
                        Output Signal Report
----------------------------------------------------------------
NO.          Name     Format           Action   Cycle
----------------------------------------------------------------
7            ADO          [ 80.0, 10.0]
8            BDIRO        [ 80.0, 10.0]
9            CHRDY        [ 80.0, 10.0]unstable  1
10           CK18MO       [ 80.0, 10.0]

================================================================
```

Figure 9-5. Output Signal Report

The contents of the "Action" column in Figure 9-5 can be either:

- Mask X: The signal value is "X" during the strobe window. It will be labeled as "X" in FTL.
- Mask Tsr-: The transition occurs within the Tsr- region. It will be labeled as "X" in FTL.
- Mask Tsw: The transition occurs within the Tsw region. It will be labeled as "X" in FTL.
- Mask Tsr+: The transition occurs within the Tsr+ region. It will be labeled as "X" in FTL.
- Unstable: More than three transitions occurred in a cycle, but do not know which one is stable within the strobe window. This is only a warning message.

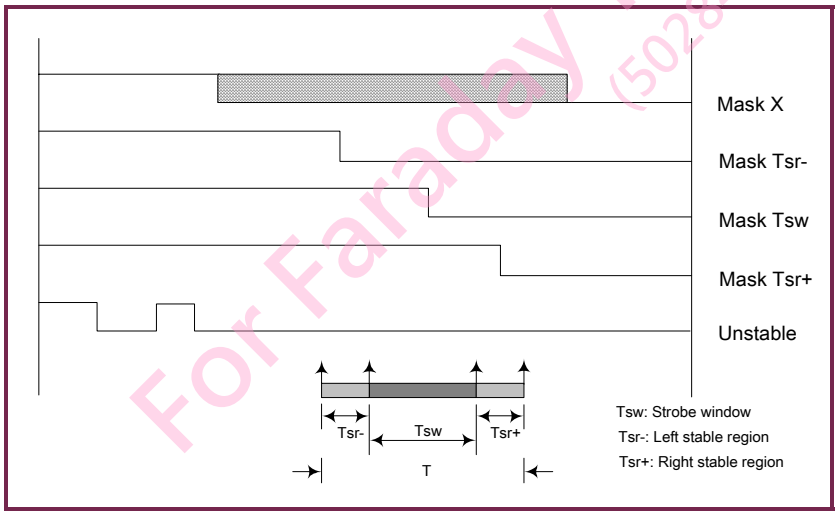


Figure 9-6. Actions Defined in the Output Signal Report

9.8.3 Bi-directional Signal Report

```
=================================================================
                          Inout Signal Report
-----------------------------------------------------------------
NO.          Name     Format           Action      Cycle
-----------------------------------------------------------------
11           D0           DNRZ, 0.0
                          [ 80.0, 10.0]unstable 1,4
12           D1           DNRZ, 0.0
                          [ 80.0, 10.0]unstable 1
13           D2           DNRZ, 0.0
                          [ 80.0, 10.0]unstable 1,4
14           D3           DNRZ, 0.0
                          [ 80.0, 10.0]unstable 1
15           D4           DNRZ, 0.0
                          [ 80.0, 10.0]unstable 1,4
16           D5           DNRZ, 0.0
                          [ 80.0, 10.0]unstable 1
17           D6           DNRZ, 0.0
                          [ 80.0, 10.0]unstable 1,4
18           D7           DNRZ, 0.0
                          [ 80.0, 10.0]control X    4
                                       unstable 1
```

Figure 9-7. Bi-directional Signal Report

The format of the bi-directional signal report is a combination of the "input signal report" and "output signal report." Please refer to the descriptions of the previous sections.

9.8.4 Stable Region Report

```
================================================================================
                          Stable Region Report (Tsr = 5.0 ns)
--------------------------------------------------------------------------------
NO.  Name    Stable Region "-" = 7.7 ns (Tsr-: Cycle) (Tsr+: Cycle) Format
--------------------------------------------------------------------------------
7    ADO     \---------       ( 77.0: 1)  ( 10.0: 3)  O[ 80.0, 10.0]
             ----[ ]-/
8    BDIRO   /---------       ( 77.0: 1)  ( 10.0: 3)  O[ 80.0, 10.0]
             ----[ ]-\
9    CHRDY   \[ ]-/           ( 5.0: 1)   ( 10.0: 1)  O[ 80.0, 10.0]
10   CK18MO  /---------       ( 77.0: 1)  ( 10.0: 1)  B[ 80.0, 10.0]
             ----[ ]-\
11           \[ ]-/           ( 5.0: 1)   ( 10.0: 1)  B[ 80.0, 10.0]
12           \[ ]-/           ( 5.0: 1)   ( 10.0: 1)  B[ 80.0, 10.0]
13           \[ ]-/           ( 5.0: 1)   ( 10.0: 1)  B[ 80.0, 10.0]
14           \[ ]-/           ( 5.0: 1)   ( 10.0: 1)  B[ 80.0, 10.0]
15           \[ ]-/           ( 5.0: 1)   ( 10.0: 1)  B[ 80.0, 10.0]
16           \[ ]-/           ( 5.0: 1)   ( 10.0: 1)  B[ 80.0, 10.0]
17           \[ ]-/           ( 5.0: 1)   ( 10.0: 1)  B[ 80.0, 10.0]
18           \[ ]-/           ( 5.0: 1)   ( 10.0: 1)  B[ 80.0, 10.0]
```

Figure 9-8. Stable Region Report

The stable region is a guard-band of the signal stability in a simulation to compensate for the signal skew in the tester. The section reports the timing margins of all the output and bi-directional signals. The third field in the report shows a representation of the minimal stable region, "[ ]" stands for the strobe window, "-" stands for the stable status, "\" stands for the transition from high to low, and "/" stands for the transition from low to high. The algorithm to evaluate the stable region is shown in the following diagram:

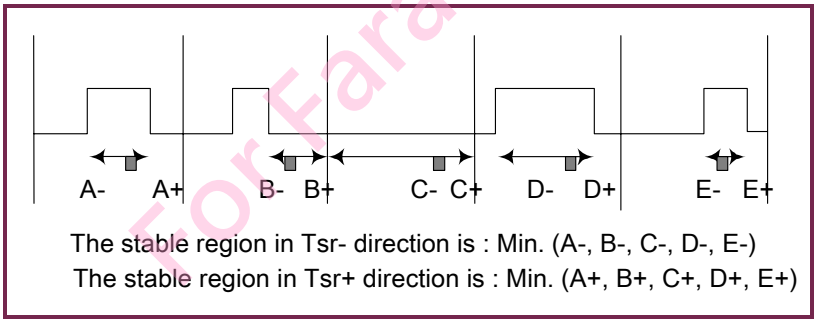


Figure 9-9. Stable Region Determination Methods

9.9 Bus Format Support

After the major release of 200209, the **ftrc** supports the bus notation used in the header file, *vcd* file, *fsdb* file, and *wgl* file.

Set forth below are the bus notations supported in this edition of TRC:

- *vcd*

```
$scope module FS70B552_SIM $end
$scope module top  $end
$var wire    1  !    data [3]   $end
$var wire    1  "    data [2]   $end
$var wire    1  #    data [1]   $end
$upscope $end
$upscope $end
$enddefinitions $end
$dumpvars
1!
0"
#100
0!
#200
1!
0"
#"300
1"
```

Figure 9-10. *vcd* File with Bus Notation

- wgl

```
waveform risc8

Signal
  "data[3]" : input;
  "data[2]" : input;
  "data[1]" : input;
End

timeplate tp0 period 100.0ns
   "data[3]" :=input[0.0ns:S];
   "data[2]" :=input[0.0ns:S];
   "data[1]" :=input[0.0ns:S];
end

pattern pat("data[3]", "data[2]", "data[1]")
vector(+,tp0) := [110]
end
end
```

Figure 9-11. *wgl* File with Bus Notation

- Header file

  ftrc supports the bus notation in the header files, and the bus can be defined as B[0], B[1], …, B[N], or B[0:N] in the signal declaration section or the sequence command.

  The followings are three examples of the header file with the bus notation:

```
TESTTYPE FUNC;

INPUT(1)    data[3:1];

TIMEUNIT     1NS;
CYCLE        100;
TIMEGEN(1)  RZ, 0, 50

SEQUENCE   data[2],
           data[3],
           data[1];
```

```
TESTTYPE FUNC;

INPUT(1)  data[3:1];

TIMEUNIT    1NS;
CYCLE        100;
TIMEGEN(1)  RZ, 0, 50

SEQUENCE data[3:1];
```

```
TESTTYPE FUNC;

INPUT(1) data[3];
INPUT(1) data[2];
INPUT(1) data[1];
TIMEUNIT     1NS;
CYCLE        100;
TIMEGEN(1)  RZ, 0, 50

SEQUENCE     data[3],
             data[2],
             data[1];
```

Figure 9-12. Header File with Bus Notation

Please note that the bus index must be in a serial order, such as b[3], b[2], b[1]. For example, the following is an illegal bus declaration.

```
TESTTYPE FUNC;

INPUT(1) data[4];
INPUT(1) data[2];
INPUT(1) data[1];
TIMEUNIT     1NS;
CYCLE         100;
TIMEGEN(1)  RZ, 0, 50

SEQUENCE     data[4],
             data[2],
             data[1];
```

Figure 9-13. Discontinuous Bus Indexes

The order of a bus declared in the signal section must be the same, such as B[0], B[1], B[2], or B[2], B[1], B[0]. But the order of a bus defined can be random in the sequence. For example, the following is an illegal bus declaration.

```
TESTTYPE FUNC;

INPUT(1) data[3];
INPUT(1) data[1];
INPUT(1) data[2];
TIMEUNIT     1NS;
CYCLE         100;
TIMEGEN(1)  RZ, 0, 50

SEQUENCE     data[3],
             data[2],
             data[1];
```

Figure 9-14. Illegal Bus Order

The order of a bus declared in the signal section must be the same as the order of a bus defined in the Verilog netlist.

9.10 ftrc Rule File

The **ftrc** rule file defines the specification and restriction of the provided IC tester. **ftrc** utilizes this rule to validate the user-defined timegen. Faraday provides several test rule collections for these special purposes. For example, the <DEFAULT> is used for the most general tester among the provided testers, and <FREE> imitates a virtual tester with little limitation on the real speed pattern conversion. The <93KC200>, <93KC400>, and <SC212> are the specified testers. The following table and descriptions show the test rule setting of the <DEFAULT> tester in the **ftrc** rule file.

```
<DEFAULT>
timeunit            = 1 ns ;
min_cycle           = 25 ;
max_cycle           = 40960 ;
tdb_of_dnrz~        = 0 ;
tde_of_dnrz         = 0 ;
tdb_of_rz           = 0 ;
tpw_of_rz           = 5 ;
tde_input_of_rz          = 0 ;
tde_bidirection_of_rz    = 0 ;
tdb_of_ro           = 0 ;
tpw_of_ro           = 5 ;
tde_input_of_ro          = 0 ;
tde_bidirection_of_ro    = 0 ;
tdb_of_sbc               = 0 ;
tpw_of_sbc               = 5 ;
tde_input_of_sbc    = 0;
tde_bidirection_of_sbc = 0;
tsb                 = 0;
tse                 = 0;
tsr                 = 5 ;
tsw~                = 5 ;
max_input_timegen   = 1000 ;
max_output_timegen  = 1000;
max_vector               = 2097152;
Ttol                = 5 ;
```

Figure 9-15. TRC Rule File Example

The followings include the descriptions of each rule:

- timeunit: Unit used in the rule file
- min_cycle: The lower bound of a allowed cycle time
- max_cycle: The upper bound of a allowed cycle time
- tdb_of_dnrz~: The minimum time interval between the beginning of a cycle and the time delay of DNRZ. (A '~' sign means that the value can also be 0.)
- tde_of_dnrz: The minimum time interval between the end of the cycle and the time delay of DNRZ

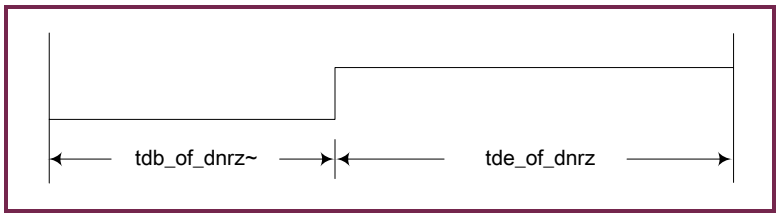


Figure 9-16. DNRZ (Delay Non-Return to Zero) Waveform

- tdb_of_rz: The minimum time interval between the beginning of the cycle and the leading edge of the RZ pulse
- tpw_of_rz: The minimum pulse width of the RZ waveform
- tde_input_of_rz: The minimum time interval between the end of the cycle and the trailing edge of the RZ pulse

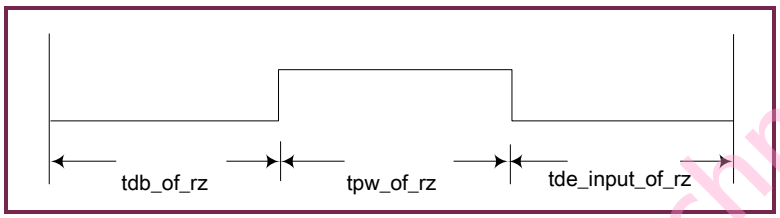


Figure 9-17. RZ (Return to Zero) Waveform

- tdb_of_ro: The minimum time interval between the beginning of the cycle and the leading edge of the RO pulse
- tpw_of_ro: The minimum pulse width of the RO waveform
- tde_input_of_ro: The minimum time interval between the end of the cycle and the trailing edge of the RO pulse

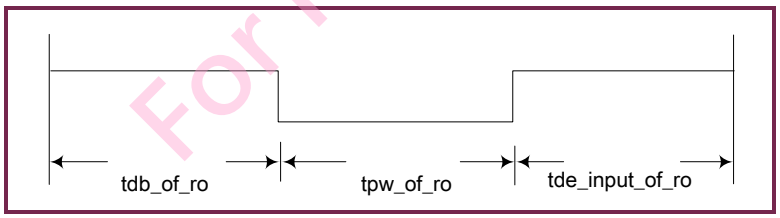


Figure 9-18. RO (Return to One) Waveform

- tsb: The minimum time interval between the beginning of the cycle and the leading edge of the strobe point
- tse: The minimum time interval between the end of the cycle and the trailing edge of the strobe point
- tsr: Front and rear margins of preventing from the skew in the tester
- tsw~: The minimum strobe window. (The '~' sign means that the value can also be 0.)

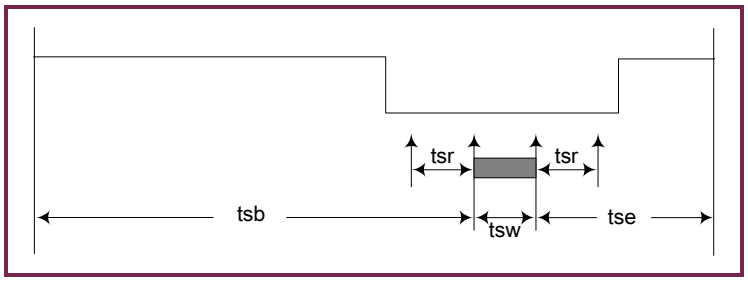


Figure 9-19. Strobe Waveform

- max_input_timegen: The maximum number of the input waveform set
- max_output_timegen: The maximum number of the output strobe set
- max_vector: The maximum number of the test cycles within one test program
- Ttol: The tolerance when **ftrc** is formatting the input waveforms (From the time of changing tabular data). The value is defined as the percentage of the cycle time. For example, if Ttol is 5 and the cycle time is 100 ns, the tolerance is then 100 * 5% = 5 ns.

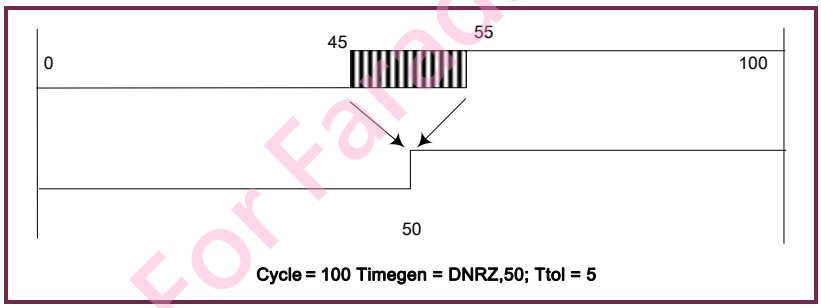


Figure 9-20. Tolerance Region of DNRZ 50

9.11 Error Message List

ftrc has the following messages:

The followings are the message lists of **ftrc**:

9.11.1 Error Messages Related to System

"err_sys001" "can't allocate memory while process line<%d>\n"
"err_sys002" "environment variable<%s> not defined\n"
"err_sys003" "fail to check out license feature<%s>\n"

9.11.2 Error Messages Related to Command File

"err_cmd001" "argument error at command line<%d>\n"
"err_cmd002" "argument<-%s> absent\n"
"err_cmd003" "can't open %s file<%s>\n"
"err_cmd004" "only 0 and 1 are allowed for FLOAT argument\n"
"err_cmd005" "only 0 and 1 are allowed for FTLMODE argument\n"

9.11.3 Error Messages Related to Rule Check

"err_rul001" "start pattern not found in rule file\n"
"err_rul002" "TIMEUNIT statement not found in rule file\n"
"err_rul003" "MIN_CYCLE statement not found in rule file\n"
"err_rul004" "MAX_CYCLE statement not found in rule file\n"
"err_rul005" "value of MIN_CYCLE greater than MAX_CYCLE in rule file\n"
"err_rul006" "TIMEUNIT statement is redefined in rule file\n"

9.11.4 Messages Related to Pad File

"err_pad001" "incorrect format of pad file<%s> - \"Instance Name\" not found\n"
"err_pad002" "incorrect format of pad file<%s> - \"Cell Name\" is missint at line %d\n"
"err_pad003" "incorrect format of pad file<%s> - \"Connected Net\" is missint at line %d\n"
"wng_pad001" "pin<%s> defined in pad file but absent in header file at line<%d>\n"

9.11.5 Error Messages Related to Header File

"err_hdr001" "template type<%s> not found in template file<%s>\n"
"err_hdr002" "TIMEUNIT statement not found in header file\n"
"err_hdr003" "number of declared input timegen<%d> exceeds rule<%d>\n"
"err_hdr004" "number of declared output timegen<%d> exceeds rule<%d>\n"
"err_hdr005" "pin<%s> defined in SEQUENCE statement is not declared before\n"
"err_hdr006" "duplicate declaration of pin<%s>\n"
"err_hdr007" "pin<%s> defined in OUTIF0/1 statement is not declared before\n"
"err_hdr008" "duplicate declaration of TIMEGEN number<%d>\n"
"err_hdr009" "pin<%s> associated with an undeclared TIMEGEN number<%d>\n"
"err_hdr011" "INOUT pin<%s> not declared to OUTIF0 or OUTIF1\n"
"err_hdr012" "pin<%s> : TIMEGEN<%d> violates the rule %s (%d%s %s %d%s)\n"
"err_hdr013" "rule violation of INOUT pin<%s>: input-delay<%d%s> greater than output-delay<%d%s>\n"
"err_hdr014" "bus bits not consecutive in pin<%s>\n"
"err_hdr015" "bus subset order not consistent in pin<%s>\n"
"err_hdr016" "CYCLE statement not found\n"
"err_hdr017" "size of CYCLE<%d%s> violates rule (min:%d%s, max:%d%s)\n"
"err_hdr018" "pin<%s> declared but absent in SEQUENCE statement\n"
"err_hdr019" "duplicate pin<%s> at SEQUENCE statement\n"
"err_hdr021" "one of START_LINE/START_STRING must be declared\n"
"err_hdr022" "no INPUT or INOUT pin defined\n"
"err_hdr023" "no OUTPUT or INOUT pin defined\n"
"err_hdr024" "one of $A_TIME/$R_TIME must be set in FORMAT string\n"
"err_hdr025" "FORMAT statement not declared in TABULAR section\n"
"err_hdr026" "both of START_LINE & START_STRING are declared in header\n"
"err_hdr027" "both of END_LINE & END_STRING are declared in header\n"
"err_hdr028" "both of START_LINE & START_STRING are declared in template\n"
"err_hdr029" "both of END_LINE & END_STRING are declared in template\n"
"err_hdr030" "each element in WHITE_SPACE statement must be one-char string\n"
"err_hdr031" "element<%s> in WHITE_SPACE statement conflicts one of reserved state pattern (0/1/x/X/z/Z)\n"

"err_hdr032" "state word in DEFINE statement must be one-char string at line<%d>\n"
"err_hdr033" "only 0/1/x/X/z/Z is allowed to use as state pattern in DEFINE statement at line<%d>\n"
"err_hdr034" "element in TERMINATOR statement must be one-char string\n"
"err_hdr035" "element<%s> in TERMINATOR statement conflicts one of reserved state pattern (0/1/x/X/z/Z)\n"
"err_hdr036" "TAB_TIMEUNIT statement not found\n"
"err_hdr037" "TIMEUNIT statement is redefined\n"
"err_hdr038" "TAB_TIMEUNIT statement is redefined\n"
"err_hdr039" "CYCLE statement is redefined\n"
"err_hdr040" "SEQUENCE statement is redefined\n"
"err_hdr041" "START_LINE number must be greater than 0 at line<%d>\n"
"err_hdr042" "TERMINATOR statement is redefined\n"
"err_hdr043" "END_TABULAR statement not found\n"
"err_hdr044" "duplicate pin<%s> in OUTIF0 statement\n"
"err_hdr045" "duplicate pin<%s> in OUTIF1 statement\n"
"err_hdr046" "pin<%s> defined in both of OUTIF0 and OUTIF1 statement\n"
"err_hdr047" "pin<%s> associated to TIMEGEN<%d> that has invalid waveform type\n"
"err_hdr048" "one and only one $STATES must be set in FORMAT string\n"
"err_hdr049" "null string defined in WHITE_SPACE statement\n"
"err_hdr050" "SEQUENCE statement is not defined\n"
"err_hdr051" "signal name \"%s\" is a reserved keyword of FTL format\n"

9.11.6 Messages Related to VCD File

"err_vcd001" "unknown timeunit<%s> in VCD file at line<%d>\n"
"err_vcd002" "scope depth over bottom in VCD file at line<%d>\n"
"err_vcd003" "INOUT pin<%s> has no control net definition in pad file\n"
"err_vcd004" "control net<%s> of pin<%s> not defined in VCD dump file\n"
"err_vcd005" "%s pin<%s> not defined in VCD dump file\n"
"err_vcd006" "Unsupport scope type in VCD file at line<%d>\n"
"err_vcd007" "Bus width mismatch in VCD file at line<%d>\n"
"err_vcd008" "Real value change is not support in current version of TRC in VCD file at line<%d>\n"
"err_vcd009" "Timeunit not defined in VCD file\n"

"err_vcd010" "More than half of pins declared in the header file are not defined in VCD dump file. Probably the VCD test top module is setting wrong.\n"
"wng_vcd001" "variable<%s> defined in VCD file but absent in header file\n"
"wng_vcd002" "pin<%s> not defined in pad file, the hierarchical name is assigned to %s\n"
"wng_vcd003" "Unsupport variable type in VCD file at line<%d>\n"

9.11.7 Messages Related to Tabular Data File

"err_dat001" "unknown TABULAR TIMEUNIT<%s> at data line<%d>\n"
"err_dat002" "start pattern not found in data file\n"
"err_dat003" "<\\t> found in skip area of data file at line<%d>\n"
"err_dat004" "number of parsed signal states<%d> at data line<%d> less than declared signals<%d> in SEQUENCE\n"
"err_dat005" "time not found at data line<%d>\n"
"err_dat006" "unknown pattern state<%c> at data line<%d>\n"
"err_dat007" "Total vector length (cycle number) exceeds the limit of MAX_VECTOR<%d> \n"
"err_dat008" "content of data line<%d> is not consistent with TABULAR FORMAT string\n"
"err_dat009" "%d extra token(s) left while matching FORMAT statement finished at data line<%d>\n"
"err_dat010" "number of parsed skip char<%d> is less than $SKIP<%d> at data line<%d>\n"
"err_dat011" "can't decide state of input pin<%s> at cycle<%d>\n"
"err_dat012" "can't decide state of output pin<%s> at cycle<%d>\n"
"err_dat013" "X or Z occurs within input cycle of pin<%s> at cycle<%d>\n"
"err_dat014" "time decrease at data line<%d>\n"
"wng_dat001" "input skew occurs at pin<%s> during cycle<%d>\n"
"wng_dat002" "X or Z occurs within input cycle of pin<%s> at cycle<%d>\n"
"wng_dat003" "X occurs within strobe region on pin<%s> at cycle<%d>\n"
"wng_dat004" "pin<%s> has transition within Tsr- at cycle<%d>\n"
"wng_dat005" "pin<%s> has transition within Tsw at cycle<%d>\n"
"wng_dat006" "pin<%s> has transition within Tsr+ at cycle<%d>\n"
"wng_dat007" "control pin<%s> is X within strobe region at cycle<%d>\n"
"wng_dat008" "control pin<%s> has transition within strobe region at cycle<%d>\n"
"wng_dat009" "Z occurs within strobe region on pin<%s> at cycle<%d>\n"

9.11.8 Error Messages Related to WGL File

"err_wgl001" "unknown direction of signal<%s> in Pattern block\n"
"err_wgl002" "duplicate pin<%s> in Pattern block\n"
"err_wgl003" "%s pin<%s> is not allowed to be declared separately in Pattern block\n"
"err_wgl004" "pin<%s> defined in Pattern block is not declared before\n"
"err_wgl005" "Pattern vector length is less than signal count at line<%d>\n"
"err_wgl006" "Pattern vector length is greater than signal count at line<%d>\n "
"err_wgl007" "Track time stamp should start from 0 at line<%d>\n"
"err_wgl008" "Non increasing time stamp in track definition at line<%d>\n"
"err_wgl009" "Timeplate definition not recognized at line<%d>\n"
"err_wgl010" "pin<%s> in timeplate section at line<%d> is not declared before\n"
"err_wgl011" "pin<%s>'s direction is different with its original direction at line<%d>\n"
"err_wgl012" "unknown direction of timeplate at line<%d>\n"
"err_wgl013" "member<%s> in scan cell group<%s> is not declared before at line<%d>\n"
"err_wgl014" "member<%s> in scan cell group<%s> is a group at line<%d>\n"
"err_wgl015" "duplicate cell<%s> in Scan Cell section\n"
"err_wgl016" "duplicate cell<%s> in scan cell group<%s>\n"
"err_wgl017" "first signal<%s> in scan chain<%s> not input direction\n"
"err_wgl018" "last signal<%s> in scan chain<%s> not output direction\n"
"err_wgl019" "scan chain<%s> has no input and output signal\n"
"err_wgl020" "scan cell<%s> in scan chain<%s> is not declared before\n"
"err_wgl021" "scan cell<%s> in scan state<%s> is not declared before\n"
"err_wgl022" "duplicate scan state name<%s> at line<%d>\n"
"err_wgl023" "duplicate scan cell<%s> in scan state<%s>\n"
"err_wgl024" "scan chain<%s> at line<%d> is not declared before\n"
"err_wgl025" "scan state<%s> at line<%d> is not declared before\n"
"err_wgl026" "signal in scan chain<%s> does not match scan vector direction at line<%d>\n"
"err_wgl027" "Pattern vector size<%d> exceeds internal buffer size<%d> at line<%d>\n"
"err_wgl028" "vector size of State<%s> less than ALL Scan count at line<%d>\n"
"err_wgl029" "pattern for bi-directional pin<%s> not paired at line<%d>\n"

9.11.9 Error Messages Related to FSDB File

"err_fsd001" "<%s> is not verilog type fsdb.\n"
"err_fsd002" "<%s> is not a fsdb file.\n"
"err_fsd003" "Open fsdb file <%s> fail.\n"
"err_fsd004" "control net<%s> of pin<%s> not defined in FSDB dump file\n"
"err_fsd005" "%s pin<%s> not defined in FSDB dump file\n"
"err_fsd006" "Timeunit not defined in FSDB file\n"
"err_fsd007" "un-supported timeunit<%s> defined in FSDB file\n"
"err_fsd008" "unknown pattern state of pin<%s> at time<%.0f>\n"

Chapter 10

Generate Verilog Test Bench with ftl2ver

This chapter contains the following sections:

The followings are the verilog test bench generation flow with **ftl2ver**:

1. Prepare the ftl file
2. Prepare the pad file
3. Prepare command file
4. Execute the **ftl2cae.lnx <id>.cmd**
5. Examine the output data

After generating the *.vt* and *.rom* file, user can perform the verilog simulation again to check whether the ftl meets original *fsdb/vcd* file.

10.1 Bus Format Support

The bus signal declaration in the *ftl* file is supported in this release. To support the bus signal declaration, the *ftl* file must obey certain rules:

1. The bus signals can be defined in the following three different formats and each signal can have its own timegen. The followings are the examples of the three acceptable bus notations:
   a. INPUT(2) mybus[5:0];
   b. INPUT(2) mybus[5:3];
      INPUT(2) mybus[2:0];
   c. INPUT(2) mybus[5];
      INPUT(2) mybus[4];
      INPUT(2) mybus[3];
      INPUT(2) mybus[2];
      INPUT(2) mybus[1];
      INPUT(2) mybus[0];

2. The orientation of the declared bus signal in the *ftl* header section must be identical, i.e., all from the low bit to the high bit or all from the high bit to the low bit.
3. The indices of a bus signal must be continuous.
4. The bus width of the bus signal in the *ftl* file must be the same as the one in the Verilog netlist.
5. The bus orientation of the bus signal in the *ftl* file must be the same as the one in the Verilog netlist. The following is an example of a Verilog module:

   ```
   module test(a,b,c) ;
   input [5:0] a;
   ```

```
input [0:5] b;
output [3:0] c;
...
endmodule
```

The following is the corresponding bus declaration in the *ftl* pattern.

```
TESTTYPE FUNC;
INPUT(1) a[5:0];
INPUT(2) b[0:5];
OUTPUT(3) c[3:0];
```

6. The bus notation is also supported in the SEQUENCE declaration of an *ftl* pattern. However, the orientation of the bus doesn't have any limitations. For example, the SEQUENCE of the above example can be declared as follow:

```
SEQUENCE a[3], a[2] ,a[5], a[4], a[0] , b[5:0], c[1:0], c[2:3];
```

10.2 Message list for flt2ver

The followings are the message list reported by **flt2ver**.

- 10.2.1 Error Messages Related to System
- 10.2.2 Error Messages Related to Command File
- 10.2.3 Messages Related to Pad File
- 10.2.4 Messages Related to *ppa* File
- 10.2.5 Error Messages Related to Rule File
- 10.210.2.6 Messages Related to *ftl* File

10.2.1 Error Messages Related to System

"err_sys001" "can't allocate memory while processing line<%d>\n"
"err_sys002" "environment variable<%s> not defined\n"
"err_sys003" "can't open bus parameter file<%s>\n"
"err_sys004" "section<%s> not existed in bus parameter file<%s>\n"
"err_sys005" "no BUSRANGE or BUSINDEX in section<%s> of bus parameter file<%s>\n"
"err_sys006" "fail to check out license feature<%s>\n"

10.2.2 Error Messages Related to Command File

"err_cmd001""required argument<%s> for %s absent\n"
"err_cmd002""can't open %s file<%s>\n"
"err_cmd003""unknown target format<%s>\n"
"err_cmd004""unknown argument<%s>\n"

10.2.3 Messages Related to Pad File

"err_pad001" "pin<%s> declared in *ftl* file but has no match in pad file\n"
"err_pad002" "pin<%s> type mismatch between *pad* file<%s> with *ftl* file<%s>\n"
"err_pad004" "not an illegal pad file<%s>\n"
"err_pad005" "error open pad file<%s>\n"
"err_pad006" "unknown type of pin<%s> in *pad* file\n"
"wng_pad001" "pin<%s> declared in *pad* file but has no match in *ftl* file\n"
"wng_pad002" "INOUT pin<%s> declared in *pad* file is assigned as %s pin in *ftl* file\n"

10.2.4 Messages Related to *ppa* File

"err_ppa001" "error open *ppa* file<%s>\n"
"err_ppa002" "no PPA_BEGIN found in *ppa* file<%s>\n"
"wng_ppa001" "pin<%s> defined in *ppa* file but absent in FTL\n"

10.2.5 Error Messages Related to Rule File

"err_rul001" "can't open rule file<%s>\n"
"err_rul002" "unknown field name<%s> in rule file at line<%d>\n"

10.2.6 Messages Related to *ftl* File

"wng_ftl001" "Stable region reaches %s cycle boundary and will be trimed.\n"
"err_ftl001" "size of CYCLE<%.f%s> violates rule (min:%.f%s max:%.f%s)\n"
"err_ftl002" "total vector length<%d> (Cycle number) exceeds the limit of MAX_VECTOR<%.f>"

"err_ftl003" "number of declared input *timegen*<%d> exceeds rule<%.f>\n"
"err_ftl004" "number of declared output *timegen*<%d> exceeds rule<%.f>\n"
"err_ftl005" "pin<%s> with *timegen*<%d> violates the rule %s (%.f%s %s %.f%s)\n"
"err_ftl006" "pin<%s> associated with an undeclared *timegen* number<%d>\n"
"err_ftl007" "decalred *timegen* index<%d> exceeds the limit<%d> at line<%d>\n"
"err_ftl008" "pin<%s> declared but absent in SEQUENCE statement\n"
"err_ftl009" "no FTL pattern found in *ftl* file\n"
"err_ftl010" "number of declared pins<%d> mismatches SEQUENCE pins<%d>\n"
"err_ftl011" "CYCLE not defined in *ftl* file\n"
"err_ftl012" "TIMEUNIT not defined in *ftl* file\n"
"err_ftl013" "SEQUENCE not defined in *ftl* file\n"
"err_ftl014" "bit stream length<%d> not equal to pin count<%d> at line<%d>\n"
"err_ftl015" "unknown pattern bit<%c> assigned to input signal<%s> at line<%d>\n"
"err_ftl016" "unknown pattern bit<%c> assigned to output signal<%s> at line<%d>\n"
"err_ftl017" "unknown pattern bit<%c> assigned to inout signal<%s> at line<%d>\n"
"err_ftl018" "declared input timegen numbers exceeds limitation<%d> of STS tester\n"
"err_ftl019" "declared output timegen numbers exceeds limitation<%d> of STS tester\n"
"err_ftl020" "duplicate pin<%s> in signal declaration\n"
"err_ftl021" "bus subset order not consistent in pin<%s>\n"
"err_ftl022" "bus subset type not consistent in pin<%s>\n"
"err_ftl023" "duplicate bus index<%d> in pin<%s>\n"
"err_ftl024" "duplicate SEQUENCE section in FTL\n"
"err_ftl025" "pin<%s> used in SEQUENCE but not declared before at line<%d>\n"
"err_ftl026" "pin<%s> defined as a bus pin in SEQUENCE but declared as non-bus pin in signal section at line<%d>\n"
"err_ftl027" "duplicate pin<%s> in SEQUENCE\n"
"err_ftl028" "bus bits of pin<%s> defined in SEQUENCE exceeds its illegal range\n"
"err_ftl029" "pin<%s> used in MASK but not declared before at line<%d>\n"
"err_ftl030" "input pin<%s> can not be masked at line<%d>\n"
"err_ftl031" "No support SBC waveform\n"
"err_ftl032" "bus pin<%s> is not continuous or has duplicate bits at pad file\n"
"err_ftl033" "input timegen of pin<%s> binds to invalid timegen category\n"
"err_ftl034" "output timegen of pin<%s> binds to invalid timegen category\n"

"err_ftl035" "scan signal<%s> not declared before at line<%d>\n"

"err_ftl036" "null scan pattern for signal<%s> at cycle<%d>\n"

"err_ftl037" "Strobe time %d ps of timegen number %d equals to clock period, this may cause simulation error.

"err_ftl038" "File name %s is not support by tester (Too many dots), please change it.

Appendix

Appendix.A Faraday Tester Interface Language

A1.1 What is ftl?

FTL stands for the Faraday tester interface language. It is defined as the intermediate data for both the simulator and tester. Based on this language format, Faraday has developed various utilities of supporting the *ftl*, the utility to convert the simulation result into *ftl* (**ftrc**), utility to convert FTL to the simulation input commands related to third party CAD tools (**ftl2ver**, etc.), and utility to convert FTL to the test programs. *ftl* file can be created through:

- Any kind of text editor, such as the vi or textedit, whichever is available on your computer
- **ftrc** programs – The tester rule checker and test pattern extractor provided by Faraday

A1.2 Example of *ftl* File

```
// Example of FTL file

TESTTYPE FUNC;

INPUT(1) ACKN, ADI, BIS ;
INPUT(3) PSIORN, PSIOWN;
INPUT(4) CK18M;
OUTPUT(2) ADO, BDIRO, CHRDY, CK18MO ;
INOUT(1,2) D0,D1,D2,D3,D4,D5,D6,D7 ;

TIMEUNIT 1 NS;
CYCLE 100;
TIMEGEN(1) DNRZ,0;
TIMEGEN(3) DNRZ,30;
TIMEGEN(4) RZ,25,50;
TIMEGEN(2) STROBE,80,10;
SEQUENCE     ACKN, ADI, BIS, CK18M, PSIORN, PSIOWN,
    ADO, BDIRO, CHRDY, CK18MO,
    D0, D1, D2, D3, D4, D5, D6, D7 ;

BEGIN
001111LHLHLLLLLLLL; // 1 : 0
001011LHHLHHHHHHHH; // 2 : 100
001011LHHLHHHHHHHH; // 3 : 200
110001HLLHLLLLLLLX; // 4 : 300
111011HLHH00000001; // 5 : 400
END
```

A1.2.1 Test Type Definition

- **Format**

```
TESTTYPE      < FUNC > ;
```

- **Description**

  The TESTTYPE is to define the type of test vector. It can be:

  FUNC : Indicate a functional test is performed.

- **Example**

```
TESTTYPE    FUNC ;
```

A1.2.2 INPUT Statement

- **Format**

```
INPUT(m)    pin_name1, pin_name2, . . . ;
```

- **Description**

  The INPUT statement declares the names of the input pins and their corresponding timing generators. *m* stands for the number to the desired input timing generator which is described in the TIMEGEN statement. *m* must be an integer between 0 and 99.

  When the multiple pins are described, a comma must be used to delimit each pin name. No blank is required to follow the comma, but the added blank may improve the readability.

Note: Currently, the maximum number of the input timing generator is 100.

- **Example**

```
INPUT(0)    CLK, RESET, WRB;
INPUT(97)   RDB, AD1, AD0;
INPUT(46)   CLR;
```

A1.2.3 OUTPUT Statement

- **Format**

```
OUTPUT(n)   pin_name1, pin_name2, . . . ;
```

- **Description**

  The OUTPUT statement declares the names of the output pins and their corresponding timing generators. *n* stands for the number to the desired output timing generator which is described in the TIMEGEN statement. *n* must be an integer between 0 and 99.

  When the multiple pins are described, a comma must be used to delimit each pin name. No blank is required to follow the comma, but the added blank may improve the readability.

Note: Currently, the maximum number of the output timing generator is 100.

- **Example**

```
OUTPUT(0)     ALE, HA1, HA0;
OUTPUT(17)    AOUT;
```

A1.2.4 INOUT Statement

- **Format**

```
INOUT(m,n)    pin_name1, pin_name2, . . . ;
```

- **Description**

  The INOUT statement declares the names of the bi-directional pins and their corresponding timing generators. The *m* stands for the number to the desired input timing generator and *n* stands for the number to the desired output timing generator. Both *m* and *n* must be the integers between 0 and 99.

  If no bi-directional pin is used in a circuit, the INOUT statement is not required. When the multiple bi-directional pins are described, a comma must be used to delimit each pin name. No blank is required to follow the comma, but the added blank may improve the readability.

- **Example**

```
INOUT(0, 99)  AD7, AD6, AD5, AD4, AD3, AD2, AD1, AD0;
```

A1.2.5 TIMEUNIT Statement

- **Format**

```
TIMEUNIT      n < PS | NS | US >;
```

- **Description**

  The TIMEUNIT declares the time unit of the CYCLE and TIMEGEN statements mentioned below. It must be an integer with the unit. The unit can be either picosecond (ps), nanosecond (ns), or microsecond (μs).

  No blank is required, but the added blanks can improve the readability.

- **Example**

```
TIMEUNIT    1 ns;
TIMEUNIT    100 ps;
TIMEUNIT    10 ps;
```

A1.2.6 CYCLE Statement

- **Format**

```
CYCLE     test_cycle;
```

- **Description**

  The CYCLE statement describes the test cycle time in the unit declared in the TIMEUNIT statement. The test cycle must be greater than MIN_CYCLE described in the test rule file.

  The test cycle reflects how often the input patterns are applied to the circuit. For example, if the test cycle is 1000 ns, the first pattern in the pattern data block represents the signal is applied in the 0 ~ 1000 ns period, and the second test pattern signal is applied in the 1000 ~ 2000 ns period, and so on.

- **Example**

```
CYCLE     100;
CYCLE     50;
```

A1.2.7 TIMEGEN Statement (Input Timing)

- **Format**

```
TIMEGEN(m)DNRZ, delay;
TIMEGEN(m)RZ, delay, pulse_width;
TIMEGEN(m)RO, delay, pulse_width;
```

- **Description**

The TIMEGEN statement defines the timing format of the signals. Users may specify the format and timing of the signals through the TIMEGEN declaration. The input signals may change their value at the onset of each test cycle, or after a fixed delay from the beginning of the test cycle. There are four types of the input waveforms as illustrated below:

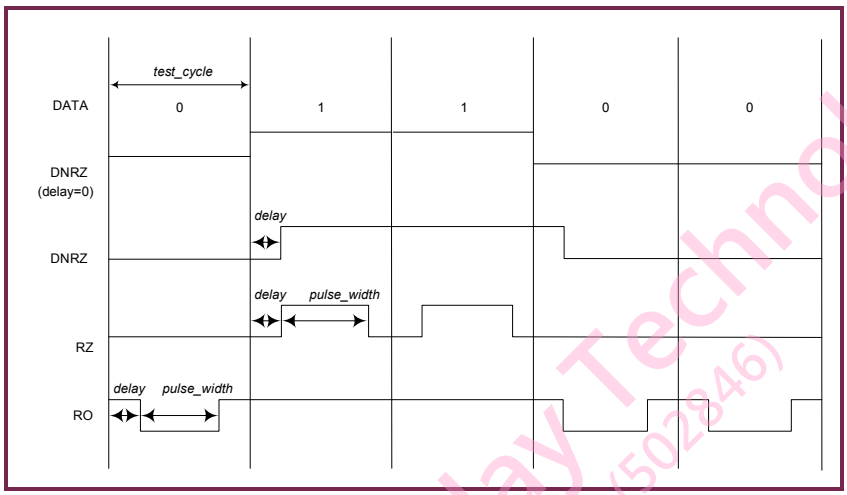


Figure 10-1. 4 Types of Input Waveform

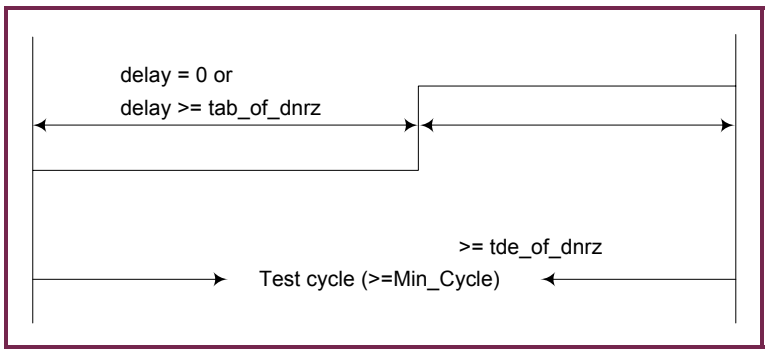
- **DNRZ format**

There are two types of DNRZ (Delayed Non-Return to Zero) formats, the non-delayed and delayed formats.

The non-delayed DNRZ waveform format changes their values at the beginning of the test cycle, if the value in this cycle differs from the one in the previous cycle.

The delayed DNRZ waveforms change their values after a fixed delay from the beginning of the test cycle. It is not necessary for the delayed DNRZ waveforms to change their values in every test cycle. But if they do, the delay must be identical for all the test cycles.

The delay of the non-delayed DNRZ waveform is 0. The change of the delayed DNRZ waveform type must be no less than tdb_of_dnrz from the beginning of the test cycle and no later than tde_of_dnrz before the end of the test cycle. The values of tdb_of_dnrz and tde_of_dnrz are described in the test rule file.



- **RZ and RO format**

  The waveform in the RZ (Return to Zero) format has a single positive pulse within one test cycle. It is not necessary for the RZ waveform to have a pulse in every test cycle. If it does, its values will stay at the logic 0 throughout the test cycle.

  The waveform in the RO (Return to One) format has a single negative pulse within one test cycle. It is not necessary for the RO waveform to have a pulse in every test cycle. If it does, its values will stay at the logic 1 throughout the test cycle.

  For both the RZ and RO waveforms, the leading edge must be occurred no less than Tdb_of_RZ (RO) from the beginning of the test cycle. The trailing edge must occur no later than Tde_input_of_RZ (RO) for the inputs and Tde_bidirection_of_RZ (RO) for the bi-directions, before the end of the test cycle. The minimum pulse width is Tpw_of_RZ (RO). All the values of Tdb_of_RZ (RO), Tde_bidirection_of_RZ (RO), Tde_input_of_RZ (RO), and Tpw_of_RZ (RO) are described in the test rule file.

Some restrictions to the types of the RZ/RO waveforms must be highlighted herein, that is:
RZ and RO formats can not be applied to the bi-directional signals.

A1.2.8 TIMEGEN Statement (Output Timing)

- **Format**

```
TIMEGEN(n)  STROBE, delay [, strobe_width ];
```

- **Description**

n is the indicator number of this timing generator. At the timing strobes, the output values are monitored for the comparison against the expected values. The starting time of the strobe operation is a delay (*delay*) from the beginning of the test cycle, and the ending time of the strobe operation is a delay (*strobe_width*) with respect to the starting time. If *strobe_width* is omitted, the edge strobes are generated.

The strobe operation must start to be greater than T_{sb} measured from the onset of the test cycle and must end to be no later than T_{se} before the end of the test cycle. The values of the T_{sb} and T_{se} are defined in the test rule file.

- **Example**

```
TIMEGEN(0)       STROBE, 80, 10;
TIMEGEN(99)      STROBE, 190;
```

A1.2.9 SEQUENCE Statement

- **Format**

```
SEQUENCE      pin_name1, pin_name2, . . . ;
```

- **Description**

The SEQUENCE statement declares the pin name sequence corresponding to the bit order in the pattern data. A comma is used to delimit the pin names. All the external I/O pins used in the circuit must be specified in the SEQUENCE statement.

To insert one blank column in the pattern data, place one additional comma between the pin names and insert an underscore "_" between the signal bits in the vector pattern. When the patterns are too wide and the vector continues to be on the next line, place a colon ":" after the last pin name of the line instead of a comma.

- **Example**

```
SEQUENCE    XTAL1P, XTAL2P, RSTPN, NEAPN, ALEPN, NPSENP, ,
    PRT07, PRT06, PRT05, PRT04, PRT03, PRT02, PRT01, PRT00, ,
    PRT17, PRT16, PRT15, PRT14, PRT13, PRT12, PRT11, PRT10, ,
    PRT27, PRT26, PRT25, PRT24, PRT23, PRT22, PRT21, PRT20, ,
    PRT37, PRT36, PRT35, PRT34, PRT33, PRT32, PRT31, PRT30;
BEGIN
0L00HH_XXXXXXXX_HHHHHHHH_HHHHHHHH_HHHHHHHH;    // 1 : 0
0L00LH_XXXXXXXX_XXXXXXXX_HHHHHHHH_XXXXXXXX;    // 2 : 25
0L00LH_XXXXXXXX_11111111_HHHHHHHH_11111111;    // 3 : 50
0L00HH_XXXXXXXX_11111111_HHHHHHHH_11111111;    // 4 : 75
0L00HH_LLLLLLLL_11111111_LLLLLLLL_11111111;    // 5 : 100
0L00LH_LLLLLLLL_11111111_LLLLLLLL_11111111;    // 6 : 125
END
```

```
SEQUENCE    ACKN, ADI, BIS, CK18M, CTSN, DCDN, DSRN, EEMDAIN,
            ERRORN, FAST2S, OPDANS, PAP3S, PBUSY, PCA0, CA1, PCA10, PCA11,PCA2, PCA3, PCA4,
            PCA5, PCA6, PCA7, PCA8, PCA9, PCAEN, PCKS, PEND,PIRQS0, PIRQS1, PIRQS2, PP2S,
            PS2AS0, PS2AS1, PSIORN, PSIOWN, PU2S,PUPAS, RESET, RIN, SER, SIRQS0, SIRQS1,
            SLCT, SPAS0, SPAS1, SPAS2, SPD0, SPD1, STFAST, ZPPS :
            ADO, BDIRO, CHRDY, K18MO, BOCN, DTRN, EMCK, EEMCSH, EEMDO,PCENBUFN, PCIRQ10,
            CIRQ11, CIRQ15, PCIRQ3, PCIRQ4, PCIRQ5, PCIRQ7,PCIRQ9, RTSN,,
            AUTOFD, D0, D1, D2, D3, D4, D5, D6, D7, INIT, D0, D1, PD2, PD3, PD4,PD5, PD6,
            PD7, SLCTIN, STRB;
BEGIN
11111111101111000101110101111111110100101110001110
XXXXXXXLLLXXXXXXXXX_X11000001XXXXXXXXXXX;        // 1 : 0
11111111101100000101110101111111110100101110001110
XXXXXXXLLLXXXXXXXXX_X00001000XXXXXXXXXX;          // 2 : 200
11111111101110000101110101111111110100101110001110
XXXXXXXLLLXXXXXXXXX_X00000000XXXXXXXXXX;        // 3 : 400
END
```

A1.2.10 Test Pattern Data

The test pattern data is a string of characters indicating the logic state values corresponding to the order defined in the SEQUENCE statement. There must be a semicolon ";" placed at the end of the pattern vectors.

A1.2.10.1 Input Patterns

Only '0' and '1' are the valid states of the input patterns. These characters, often called the pattern mnemonics, have the following meanings.

- **DNRZ waveforms**

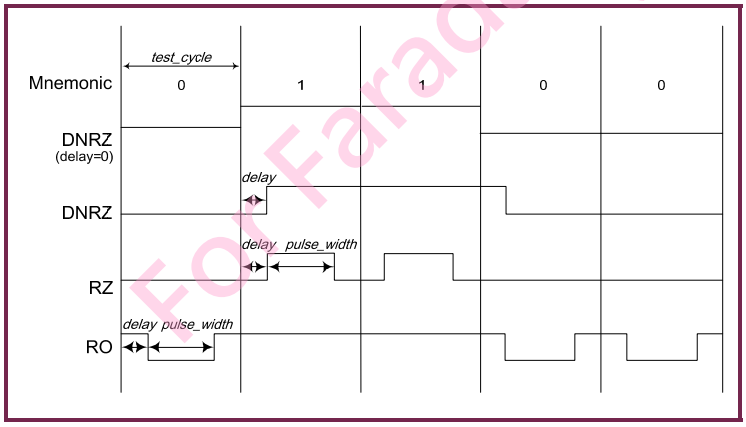
  The signal represented by a DNRZ waveform changes its value from 0 to 1 after the specified delay from the test boundary when the pattern mnemonic changes from 0 to 1. Conversely, the signal value changes from 1 to 0 when the pattern mnemonic changes from 1 to 0. The signal does not change its value throughout the test cycle if the pattern mnemonic is the same as the previous cycle. The delay of the DNRZ signal may be 0. That means the signal will change its state on the boundary of the test cycle.

- **RZ waveforms**

  The signal represented by an RZ waveform generates a positive pulse of a specified delay when the pattern mnemonic is 1. The signal stays at 0 throughout the test cycle when the pattern mnemonic is 0.

- **RO waveforms**

  The signal represented by an RO waveform generates a negative pulse of a specified delay when the pattern mnemonic is 0. The signal stays at 1 throughout the test cycle when the pattern mnemonic is 1.



A1.2.10.2 Output Patterns:

The L, H, Z, and X are the legal characters of describing the expected output patterns. These mnemonics have the following meanings:

L : The output signal is expected to be low (Logic 0) at the strobe point (Window) at the test cycle.

H : The output signal is expected to be high (Logic 1) at the strobe point (Window) at the test cycle.

Z : The tri-state output signal is expected to be in the high-impedance state at the strobe point (Window) at the test cycle.

X : Don't care.

A1.2.11 Remark Statement

- **Format**

```
//      comment
/*      comment        */
```

- **Description**

  You can use "//" as the remark statement. All the statements after "//" will be ignored until the encountering of a new-line symbol. You can use the pair "/*" and "*/" to start and end a remark statement, too. Any characters between "/*" and "*/" are ignored; they may be used freely to make a program easier to understand.

- **Example**

```
// FTC_tRC  1.0
// Header    File: tx.h
// Input     File: tx.tab
// Rule      File: /home1/pctool/victor/ftc/ETC/TRC/trc.rul
// Template File: /home1/pctool/victor/ftc/ETC/TRC/trc.tmp
/* Total_Pin_No.: 0090
 Input: 0051    Output: 0019    Bidirection: 0020 */

TESTTYPE FUNC;
INPUT(0)         ACKN, ADI, BIS, CTSN, DCDN, DSRN, EEMDAIN, ERRORN;
INPUT(5)         CK18M;
OUTPUT(7)    ADO, BDIRO, HRDY, CK18MO, DBOCN, DTRN;
```

A2.1 Why Header File of TRC?

Only the information of timing and signal states can be obtained in the simulation result file. TRC needs more information to successfully generate the cycle-based test pattern file. An additional header file is required for this purpose. A header file includes the declaration of the pin names, timing format of signals, data format of simulation result file, etc. TRC will extract the test pattern from the simulation result file based on the declarations in the header file.

A2.2 Example of Header File

```
TESTTYPE      FUNC;
INPUT(1)      SPAS0, SPAS1, SPAS2, SPD0, SPD1, STFAST, ZPPS;
INPUT(8)      ACKN, ADI, BIS, CTSN, DCDN, DSRN, EEMDAIN;
INPUT(19)         PSIORN, PSIOWN;
INPUT(2)      CK18M;
OUTPUT(3)         ADO,BDIRO,CHRDY,CK18MO;
OUTPUT(4)         DBOCN, DTRN, EEMCK, EEMCSH, EEMDO;
INOUT(1,4)        D0, D1, D2, D3, D4, D5, D6, D7;
OUTIF1        D0, D1, D2, D3, D4;
OUTIF0        D4, D5, D6, D7;
TIMEUNIT      1 NS;
CYCLE             200;
TIMEGEN(1)        DNRZ, 0;
TIMEGEN(8)        DNRZ, 70;
TIMEGEN(2)        RZ, 50, 100;
TIMEGEN(19)   RO, 30, 100;
TIMEGEN(3)        STROBE, 180;
TIMEGEN(4)        STROBE, 170, 10;
SEQUENCE      SPAS0, SPAS1, SPAS2, SPD0, SPD1, STFAST, ZPPS, ACKN,ADI, BIS, CTSN, DCDN, DSRN, EEMDAIN,
PSIORN, PSIOWN,CK18M,  ,ADO, BDIRO, CHRDY, CK18MO, DBOCN, DTRN,EEMCK, EEMCSH, EEMDO,  ,D0, D1, D2,
D3, D4, D5, D6, D7;
DEFINE        "St1" = "1", "We0" = "0", "HiZ" = "Z";
TABULAR
TAB_TIMEUNIT      10 PS;
START_LINE            1;
END_LINE          $ - 3;
WHITE_SPACE       ",";
TERMINATOR        ";"
FORMAT            "$a_time  =  $states";
END_TABULAR
```

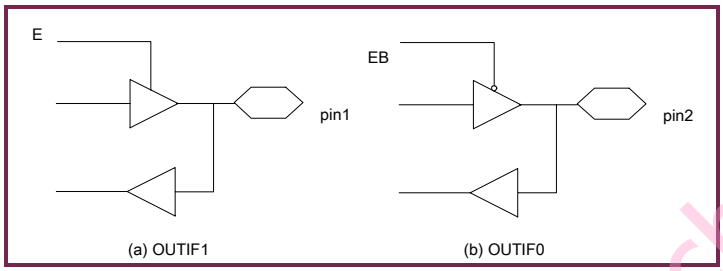
A2.2.1 BOUTIF1/OUTIF0 Statement

- **Format**

```
OUTIF1        pin_name1, pin_name2, . . . ;
OUTIF0        pin_name1, pin_name2, . . . ;
```

- **Description**

  OUTIF1/OUTIF0 declares the control type of a bi-directional pin. A bi-directional pin must be declared as OUTIF1 if it is in the output mode when the control signal is in high (Logic 1). An OUTIF0 statement defines that a bi-directional pin is in the output mode when the control signal is in low (Logic 0). Each bi-directional pin must only have one corresponding OUTIF1/OUTIF0 declaration. It is shown as follows:



- **Example**

```
OUTIF1        DA7, DA6, DA5, DA4, DA3, DA2, DA1, DA0;
OUTIF0        AD, BID;
```

A2.2.2 DEFINE Statement

- **Format**

```
DEFINE        ”state1”=”valid-state1”,
              ”states2” = ”valid-state2”, ... ;
```

- **Description**

  Without the extra definitions, TRC only recognizes 4 valid states from the simulation result file, the 0 (Logic 0), 1 (Logic 1), Z (High-impedance), and X (Unknown). If your simulator generates an equivalent state as 0, 1, Z or X, but the format is in different character (For example, state 0 stands for the logic 0, state 1 stands for the logic 1, etc.), the DEFINE statement can be used

to make your own state characters or string recognizable with TRC. For example, if the unknown state is expressed as "U" in some simulators, you must define 'U" as "X" in the header file, and the TRC will treat "U" as an unknown state.

- **Example**

```
DEFINE        "St1" = "1";
DEFINE        "We0" = "0", "U" = "X", "HiZ" = "Z";
```

A2.2.3 TABULAR/END_TABULAR Statement

- **Format**

```
TABULAR
<tabular-statements>
END_TABULAR
```

- **Description**

It is necessary to define the format of the target tabular simulation result file by using the TABULAR statement. The tabular statement block is enclosed between the TABULAR and END_TABULAR keywords.

- **Example**

```
TABULAR
TAB_TIMEUNIT     10PS;
START_LINE       1;
END_LINE         $-2;
FORMAT           "$a_time = $states";
END_TABULAR
```

A.2.2.4 TAB_TIMEUNIT Statement

- **Format**

```
TAB_TIMEUNIT     n <PS | NS | US>;
```

- **Description**

  TAB_TIMEUNIT defines the time unit in the tabular simulation result file. The unit of the timing stamp can also be defined by using the $timeunit in the FORMAT statement. If the TAB_TIMEUNIT and $timeunit are both defined in the header file, the TRC will adopt the $timeunit as the first priority.

- **Example**

```
TAB_TIMEUNIT            1 ns;
TAB_TIMEUNIT            10 ps;
START_LINE/END_LINE
```

A2.2.5 START_STRING/END_STRING Statement

- **Format**

```
START_LINE          starting-line;
END_LINE            ending-line;
START_STRING        starting-string;
END_STRING          ending-string;
```

- **Description**

  Sometimes there are the redundant paragraphs in the simulation result file, such as the header and tailor in recording the pin names, timing parameters, or other information. You have to tell TRC to skip these redundant paragraphs to capture the correct patterns. The START_LINE/END_LINE statements and the START_STRING/END_STRING statements are designed for this purpose.

  START_LINE defines the starting line of the actual pattern block. END_LINE defines the ending line of the actual pattern block. For example, the statement of START_LINE 19 and END_LINE 100 will force the TRC to read the tabular pattern file from the 19th line to the 100th line as the actual pattern block. When using the END_LINE statement, "$" represents the last line, and "$-*n*" represents the "last *n*th line."

START_STRING defines a string token as the beginning of the actual pattern block. TRC will ignore the content of the tabular data file until reaching the starting string token, and then start reading the actual pattern data from the next line. If the pattern data follows the starting string immediately, a "+" must be declared following the starting string token. For example, START_STRING "starting-string" +.

END_STRING defines a string as the ending of the actual pattern block. The TRC will stop reading the actual pattern data once a defined ending string is encountered.

- **Example**

```
START_LINE          15;
END_LINE            987;
END_LINE            $;
END_LINE            $-5;
START_STRING        "begin";
START_STRING        "start"+;
END_STRING          "end";
```

A.2.2.6 WHITE_SPACE Statement

- **Format**

```
WHITE_SPACE         "white-space1",  "'white-space2", ... ;
```

- **Description**

  Sometimes, there are meaningless characters in a vector line which are treated as a white space. You have to define these white space characters by using the WHITE_SPACE statement to guide TRC in reading the pattern data. Additionally, only single character can be defined as a white space.

- **Example**

```
WHITE_SPACE      ",";
WHITE_SPACE      "_", "+";
```

A.2.2.7 TERMINATOR Statement

- **Format**

```
TERMINATOR      "terminate-character";
```

- **Description**

  TRC reads a vector line and processes one time. By default, TRC treats a new-line (\n) character as the terminator of one vector line. In other cases, you have to define the terminator character by using the TERMINATOR statement to guide TRC in reading the pattern data. For example, to have meaning by placing a ";" character at the end of vector line, you have to define TERMINATOR ";" in the header file. There is no doubt that the legal state characters (0, 1, X, Z, and defined state character in the DEFINE statement) are not the candidate characteristics of the TERMINATOR statement. For example, if DEFINE "U" = "X" is declared in the header file, the character "U" will not be allowed to use again in the TERMINATOR statement.

- **Example**

```
TERMINATOR      ";";
TERMINATOR      "\";
```

A.2.2.8 FORMAT Statement

- **Format**

```
FORMAT      "$a_time  = $states";
FORMAT      "$skip15 data: $states/$r_time$timeunit";
```

- **Description**

  The FORMAT statement can describe the structure of a vector line, including the state of the pins and timing information, in the Original Tabular data File (OTF). Typically, a vector line in OTF contains the time stamps and the states for all the pins in a fixed format. The time stamp always increases from one vector to the next. This is probably the most common file format of the simulation result.

  The FORMAT statement allows you to describe the data order and format in the tabular data file with the help of some special $keywords. TRC scans each line in the tabular data file and attempts to match the data with one of the string specified in the FORMAT statement. A successful matching results in loading a new time and state data into the TRC; otherwise, neglects the vector line with an error message.

Within the FORMAT statement, there are three special characters, the space, the "$" character, and a back slash "\." The space character tells TRC that there may be 0 or more white spaces at this location. Thus, the single space in the FORMAT statement can match 0 or more white spaces in the vector line of the tabular data file. Characters interpreted as the white spaces in the vector line include the space and tab characteristics. Other characters can be added to the list of the white space characters by using the WHITE_SPACE statement.

The "$" character prefixes one of the several keywords that tell the TRC what kind of data are expected at this location in this vector line, or what kind of action to take. Any other character encountered in a format string is interpreted as a "hard" character and must have a corresponding identical character at this location in the OTF vector line for the format string to match. To distinguish whether the "$" character is a prefix of the keywords or a hard character, TRC uses "\" as the escape character. For example, if there is a "$" hard character in the vector line, you must declare it in the format string as \$. Because $ \. To have the special purpose FORMAT statement, you have to escape these characters if you use them as a pure state character, for example "\\."

Currently, the legal $keyword and their meanings are as follows:

$a_time: Absolute time stamp
This is the most common case, positive integer time stamp for the vector.

$r_time: Relative time stamp
For some OTF, the time stamp represents a differential time from the time of the previous vector. TRC thus adds this value to the current time to get an absolute time stamp of this vector.

$timeunit: Unit of time stamp

$states: State data of the pins
When this $keyword is used in the FORMAT statement, TRC looks for the states of all pins. TRC processes the states by using the signal order described in the SEQUENCE statement of the header file.

$skip*n*: Skip *n* character positions in the vector line, where *n* is a positive integer. It is used to guide TRC to skip the specified length of texts that are not relevant to the time stamp and state information in the vector line of OTF, but regularly changed between the vector lines; for example, the cycle number in the head of each vector line.

Please note that the "Tab" character is not allowed in the text that you want to skip. You may lose the absolute position in the vector line if you insist to use "Tab."

The example below gives the vector lines from OTF and the corresponding FORMAT statement.

- **Example**

```
OTF1:
       230315        0 0 0 1 0 1 0 1 1 x x 0 0 z z z z

TAB_TIMEUNIT         10 PS ;
FORMAT               "$a_time  $states ";

OTF2:
$1982.785            1, 0 , 0, 1, x , x, 0, 0, z, z, z, z, 0, 1;

TAB_TIMEUNIT         1ns;
WHITE_SPACE          "," ;
FORMAT               "\$$a_time  $states ;";

OTF3:
100.0010_10001 \ 128
100.0010_00001 \ 56

WHITE_SPACE          "." "_" ;
TAB_TIMEUNIT         100ps ;
FORMAT               "$states \\ $r_time" ;

OTF4:
Vector        3      1639747ps1001001101000
Vector        4      1639804ps0001001101000

FORMAT               "$skip20  $a_time$timeunit  $states" ;

OTF5:
1004ps        10011001110001; 3
432ps         00011001110001;  4

TERMINATOR           ";" ;
FORMAT               "$r_time$timeunit  $states" ;
```