Equation and JEDEC Simulators User Manual

SYNARIO

Universal FPGA Design System

Synario is a Data I/O Product

Table of Contents

Preface

1. Equation and JEDEC Simulation

Equation and JEDEC File Simulation 1-1
Test Vector Files 1-1
How to Invoke Simulation 1-2
The Simulator Model 1-3
JEDEC and .tmv Vectors 1-3
Simulation Flow 1-4

2. Report and Trace Types and Break Points

Controlling the Simulation 2-1
Trace Type: Brief 2-2
Trace Type: Clock 2-2
Trace Type: Detailed 2-3
Report Type: None 2-4
Report Type: Tabular 2-6
Report Type: Pins 2-6
Report Type: Macro-cell 2-8
Waveform Viewer 2-10

3. Simulation and Designs with Buffered Outputs

Simulation and Unspecified Inputs 3-1
Simulation for Designs with Feedback 3-2
Register Preloads in the Simulator 3-5
Test Vectors and Simulation 3-5

Debugging State Machines 3-6
Multiple Test Vector Sections 3-7
Automatic Signal Selection 3-8
Don’t Cares in Simulation 3-9
Preset and Preload Registers 3-11
Special Preset Considerations 3-11
TTL Preload 3-13
Supervoltage Preload 3-16
Preset / Reset Controlled by Product Term 3-19
Preset / Reset Controlled by Pin 3-21
Powerup States 3-21
Devices with Clock Inputs 3-21

Preface

The Preface presents an overview of the topics covered in the three chapters that comprise the *Equation and JEDEC Simulators User Manual.* This manual covers the following topics:

- **Chapter 1,** "**Equation and JEDEC Simulation**" — provides a detailed description of the Equation and JEDEC simulators.
- **Chapter 2,** "**Report and Trace Types and Break Points**" — explains how to use Report and Trace Types and Break Points to control simulation.
- **Chapter 3,** "**Simulation and Designs with Buffered Outputs**" — discusses a variety of issues to consider when you simulate designs with buffered outputs.

Note: *Currently, the MACH4 devices do not support JEDEC simulation; however, you can simulate the PLA file with the PLA simulator. The PLA simulator operates in the same manner as the JEDEC simulator.*

Chapter 1
Equation and JEDEC Simulation

This chapter explains the simulation of Equation and JEDEC files for PLD designs.

Equation and JEDEC File Simulation

Test Vector Files

To use the Equation and JEDEC simulators, you will have to create test stimulus with ABEL test vectors. See the *ABEL-HDL Reference* for more information about test vector syntax.

There are two ways to specify test vectors. The most common method is to place test vectors in the ABEL-HDL source file. If you use this method, the Project Navigator will detect the presence of test vectors in the source file and create a "dummy" test vector file. This file indicates to the system that the actual test vectors are in the ABEL-HDL source file.

The other way to specify test vectors is to create a "real" test vector file by selecting the New menu item in the Source menu and then choosing test vectors. Note that test vector files have the ABV file extension and must have the same name as the top level module.

An example test vector file is shown on the following page. Note that you must use the Module and End statements exactly as you do when you create an ABEL-HDL source file. Also note that all pin and node declarations must be included, as well as any constant declarations. It's important to remember to change the pin declarations in the test vector file every time you change the pin declarations in the source. You may wonder if there are any advantages to placing test vectors in the ABV file instead of in the ABEL-HDL source. The most obvious advantage is an improvement in processing time. By placing test vectors in the ABV file you will be able to change the test vectors and re-simulate without having to re-compile the logic. This can make a significant difference in large hierarchical designs.

Figure 1-1
Example ABV File

```
module scan_tv;

c,x = .c.,.x.;

rst, clk      pin;
q5,q4,q3,q2,q1,q0   pin istype 'reg,invert';
up,down       pin istype 'com,invert';

test_vectors
  ([rst,clk] -> [q5,q4,q3,q2,q1,q0,up,down])
   [ 0 , 0 ] -> [ x, x, x, x, x, x, x,  x ];
   [ 1 , c ] -> [ 0, 0, 0, 0, 0, 1, 1,  0 ];
   [ 0 , c ] -> [ 0, 0, 0, 0, 1, 0, 1,  0 ];
   [ 0 , c ] -> [ 0, 0, 0, 1, 0, 0, 1,  0 ];
   [ 0 , c ] -> [ 0, 0, 1, 0, 0, 0, 1,  0 ];
   [ 0 , c ] -> [ 0, 1, 0, 0, 0, 0, 1,  0 ];
   [ 0 , c ] -> [ 1, 0, 0, 0, 0, 0, 0,  1 ];
   [ 0 , c ] -> [ 0, 1, 0, 0, 0, 0, 0,  1 ];
   [ 0 , c ] -> [ 0, 0, 1, 0, 0, 0, 0,  1 ];
   [ 0 , c ] -> [ 0, 0, 0, 1, 0, 0, 0,  1 ];
   [ 0 , c ] -> [ 0, 0, 0, 0, 1, 0, 0,  1 ];
   [ 0 , c ] -> [ 0, 0, 0, 0, 0, 1, 1,  0 ];
   [ 0 , c ] -> [ 0, 0, 0, 0, 1, 0, 1,  0 ];
   [ 0 , c ] -> [ 0, 0, 0, 1, 0, 0, 1,  0 ];
   [ 0 , c ] -> [ 0, 0, 1, 0, 0, 0, 1,  0 ];
   [ 0 , c ] -> [ 0, 1, 0, 0, 0, 0, 1,  0 ];
   [ 0 , c ] -> [ 1, 0, 0, 0, 0, 0, 0,  1 ];
   [ 0 , c ] -> [ 0, 1, 0, 0, 0, 0, 0,  1 ];
   [ 0 , c ] -> [ 0, 0, 1, 0, 0, 0, 0,  1 ];
END
```

How to Invoke Simulation

To make the functional simulation process available, select the test vector (ABV) file in the *Sources in Project* window. The simulation processes will then appear in the *Processes for Current Source* window. If you have not selected a device, only the Equation Simulation process will appear. If you have selected a device, and the process flow for that device supports JEDEC simulation, then the JEDEC Simulation processes will also appear in the *Processes for Current Source* window.

To start a simulation, double-click on either "Simulate Equations" or "Simulate JEDEC File" (depending on which is the appropriate simulator).

The Simulator Model

The Equation simulator uses the Equation file to build a model of the design. The JEDEC simulator uses the JEDEC and device files to build a model of the design. These methods include macrocells, sum-terms, and product terms. Select the Report Type Macro-cell property to display the model.

JEDEC and .tmv Vectors

The Equation simulator uses the tmv file vectors. The JEDEC simulator uses the JEDEC vectors but can optionally use the tmv vectors instead. JEDEC vectors include only test conditions for pins; the .tmv file vectors allow testing of internal nodes. Also, the vectors in the .tmv file can have different values for input and output. For example, the .tmv file allows you to apply a 0 to a bidirectional pin that is an input before the clock and test for an H after the clock. A JEDEC vector could only have the H. In the JEDEC simulator, select the Use .tmv File test vectors property to use the .tmv file for simulation in place of the JEDEC vectors.

JEDEC Vector Conversion

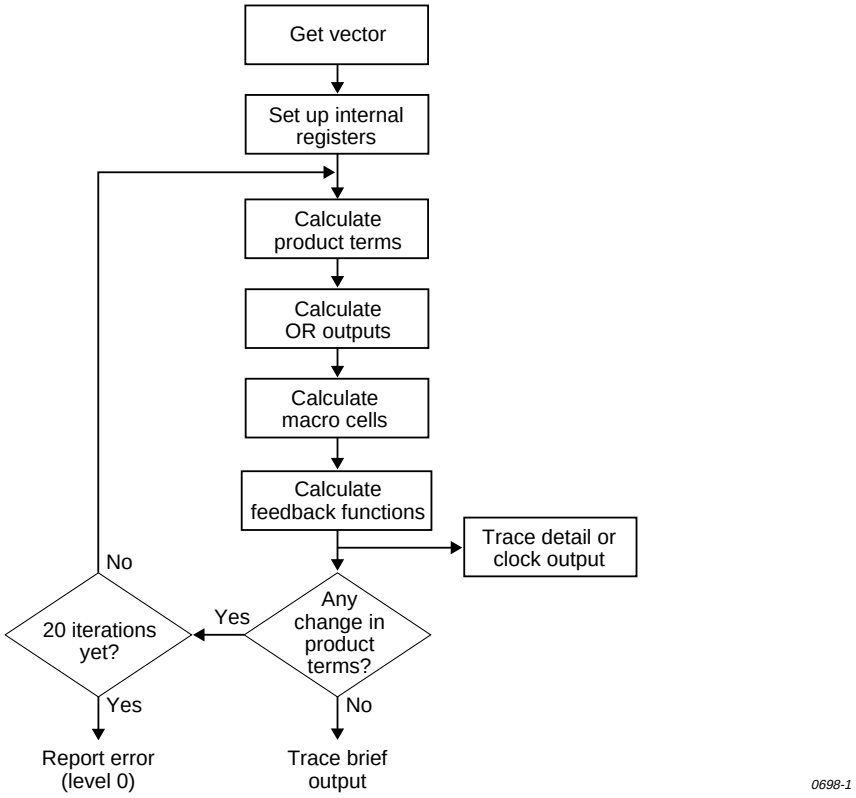
Internally, the JEDEC simulator uses the same test_vector format as the tmv file vectors. The simulator converts JEDEC vectors to tmv format. When reading JEDEC test vectors, the simulator copies the H, L, and Z into the output vector, and all test conditions into the input vector. It also makes the following conversions:

H	Converted to 1.
L	Converted to 0.
.Z.	Converted to 1 or the user-specified value.
.X.	Converted to 0 or the user-specified value.
.C.	Expanded to three vectors with .C. taking on the values 0, 1, and then 0. Use a Trace Type of Clock to observe the clock conversions.
.U.	Expanded to two vectors taking on the values 0 and then 1. Use a Trace Type of Clock to observe the clock conversions.
.K.	Expanded to three vectors with .K. taking on the values 1, 0, and then 1. Use a Trace Type of Clock to observe the clock conversions.
.D.	Expanded to two vectors taking on the values 1 and then 0. Use a Trace Type of Clock to observe the clock conversions.

Simulation Flow

Figure 1-2 shows a flow diagram of simulation during evaluation of the inputs to the output. This flow is the same for both Equation and JEDEC simulation. The simulator applies the first test vector and performs any setup of internal registers that results from the vector applied to the inputs. The simulator then calculates the product terms that result from the test vector, the OR outputs that result from the product terms, any macrocell outputs that result from the OR outputs, and any feedback functions. The results of the simulator calculations are written to the .sim file.

Figure 1-2
Simulation Flow Diagram



The outputs of designs with feedback may require several successive evaluations until the outputs stabilize. After the feedback paths have been calculated, the simulator checks to see if any changes have occurred in the design since the product terms were last calculated. If changes have occurred, due to feedback functions, the calculations are repeated. This iterative process continues until no changes are detected, or until 20 iterations have taken place. If 20 iterations take place and there are still changes, the design is determined to be unstable and an error is reported. More detailed information on simulating devices with feedback, and other advanced uses of the simulation program are presented in the chapters that follow.

Chapter 2
Report and Trace Types and Break Points

This chapter explains how you can use Report and Trace Types and Break Points to control the simulation.

Controlling the Simulation

Report and trace types and break points allow you to control the amount of information the simulation provides. Simulation can provide simple error messages (indicating that the actual outputs differ from the outputs you predicted in your test vectors), or detailed information about the states of internal registers and product terms of a device during simulation. If you simulate a design with Report Type set to None, you can determine if any errors exist. If there are errors, you can use the more detailed Report and Trace types to increase the amount of information provided until you have enough information to solve the problem. Examples of the output produced by the different Report and Trace types are given below.

Note: *In order to see the clock waveform in the waveform viewer, you need to use Trace Type: Detailed, which is the default. Otherwise, you will not see the clock edges displayed.*

With small designs, you can rerun simulation using different Report and Trace types to obtain the information you need. With a more complex design, however, reports may contain so much information that you have difficulty finding information on the error. With break points, you can target the error and limit the amount of information produced. For example, if you run a simulation and detect an error in the twentieth test vector out of 50 vectors, you may want to see more information to determine the cause of the error. You can rerun simulation with a more detailed Trace Type, exclusively for vector 20, using breakpoints in the following manner:

1. Simulate designs at Report Type: None to determine the existence of errors.
2. Once an error is found, increase the Trace and Report Types until you have enough information to correct the error.
3. Use break points to limit simulation results.

Each of the Report and Trace Type properties are discussed below.

Trace Type: Brief

Figure 2-1
Brief Trace Simulation Output

Figure 2-1 shows a brief trace simulation with a Tabular report output for the same source file that produced Figure 2-5.

```
Simulate ABEL 1.00  Date Tue Aug 14 10:56:35 1993
Fuse file: 'regfb.jed'  Vector file: 'regfb.tmv'  Part: 'PLA'
Operation of the simulator on devices with feedback
           I
       C   N
       l O I D D D   F F
       k E T 1 2 3   1 2

V0001  C 0 0 1 1 1   H L
V0002  C 0 0 0 0 1   H L
V0003  C 0 1 1 1 1   L H
V0004  0 0 0 0 0 1   L L
4 out of 4 vectors passed.
```

Trace Type: Clock

The information provided by Trace Type: Clock is similar to the information provided by Brief, except the Clock output displays the device inputs and outputs for each clock pulse. The output shown in Figure 2-2 is an expanded version of the output shown in Figure 2-1. The expanded version shows each clock pulse of the simulate operation.

```
Simulate ABEL 1.00  Date Tue Aug 14 10:58:41 1993
Fuse file: 'regfb.jed'  Vector file: 'regfb.tmv'  Part: 'PLA'
Operation of the simulator on devices with feedback
```

Figure 2-2
Clock Trace Simulation Output

```
                I
          C     N
          l O I D D D     F F
          k E T 1 2 3     1 2

V0001     0 0 0 1 1 1     H L
          1 0 0 1 1 1     H L
          0 0 0 1 1 1     H L
 ...edited...
V0004     0 0 0 0 0 1     L L
4 out of 4 vectors passed.
```

Trace Type: Detailed

The information provided by Trace Type: Detailed is similar to the information provided by Clock, except the Detailed output shows the device inputs and outputs for each iteration of the simulator. Figure 2-3 shows the Detailed output for the same source file that was used to generate Figure 2-1. The output displayed in Figure 2-3 is an expanded version of the output displayed in Figure 2-1 to show each iteration of the simulate operation that takes place to stabilize the device output. In this case, Vector 3 takes an extra iteration of the high clock pulse to stabilize. Detailed trace is the default.

Figure 2-3
Detailed Table Format Simulation Output

```
Simulate ABEL 1.00  Date Tue Aug 14 10:59:21 1993

Fuse file: 'regfb.jed'  Vector file: 'regfb.tmv'  Part: 'PLA'
Operation of the simulator on devices with feedback
       Data I/O Corp.      31 July 1993

           I
       C   N
       l O I D D D   F F
       k E T 1 2 3   1 2

V0001  0 0 0 1 1 1   Z Z
       0 0 0 1 1 1   H L
       1 0 0 1 1 1   H L
       0 0 0 1 1 1   H L
V0002  0 0 0 0 0 1   H L
       1 0 0 0 0 1   H L
       0 0 0 0 0 1   H L
V0003  0 0 1 1 1 1   H L
       1 0 1 1 1 1   L L
       1 0 1 1 1 1   L H
       0 0 1 1 1 1   L H
V0004  0 0 0 0 0 1   L L
4 out of 4 vectors passed.
```

Report Type: None

Figure 2-5 shows the output of JEDEC simulation created during processing of regfb.abl (Figure 2-4) with the Report Type set to None.

Figure 2-4
Regfb.abl Source File

```
module regfb
 title 'Operation of the simulator on devices with feedback
       Data I/O Corp.      31 July 1993'

        FB2     device 'P16R4';

        Clk,OE          pin 1,11;
        INIT,D1,D2,D3   pin 2,3,4,5,;
        F1,F2           pin 14,13;

        F1 istype 'reg_D,invert';

equations
        F1.D    = D1 & INIT;
        F2      = D2 & F1.Q;

        F2.OE  = D3;
        F1.C   = Clk;
        F1.OE  = !OE;

test_vectors   ([Clk,OE,INIT,D1,D2,D3] -> [ F1, F2])
                [.C., 0,  0 , 1, 1, 1] -> [ 1 , 0 ];
                [.C., 0,  0 , 0, 0, 1] -> [ 1 , 0 ];
                [.C., 0,  1 , 1, 1, 1] -> [ 0 , 1 ];
                [ 0 , 0,  0 , 0, 0, 1] -> [ 0 , 0 ];
end regfb
```

Figure 2-5
No Report Simulation Output

```
Simulate ABEL 1.00  Date Tue Aug 14 10:54:16 1993

Fuse file: 'regfb.jed'  Vector file: 'regfb.tmv'  Part: 'PLA'
Operation of the simulator on devices with feedback
       Data I/O Corp.      31 July 1993

4 out of 4 vectors passed.
```

In Figure 2-6, one of the test vectors was changed to produce an error, and simulation was run through JEDEC simulation at Report Type: None (only errors are shown). When an error occurs, the simulation output lists the number of the vector that failed, the name and number of the failed output, and the nature of the failure.

Figure 2-6
No Report Simulation Output Showing Error

```
Simulate ABEL 1.00  Date Mon Aug  6 14:48:31 1993

Fuse file: 'fb2.jed'  Vector file: 'fb2.jed'  Part: 'P16R4'

ABEL 1.00 Data I/O Corp. JEDEC file for: P16R4 V8.0
Created on: Mon Aug  6 14:45:51 1993

Operation of the simulator on devices with feedback
       DATA I/O Corp.      31 July 1993

Vector 4
F2 13, 'L' found  'H' expected

3 out of 4 vectors passed.
```

Report Type: Tabular

Report Type: Tabular is the default format. Tabular gives a table with signal levels represented by H, L, and Z for logic high, logic low, and high-impedance state.

Figures 2-1, 2-2, and 2-3 (earlier in this chapter) show the Tabular output for all Trace Types.

Report Type: Pins

The Pins output shows the actual signal outputs and the test vectors used to perform the simulation. The actual output associated with each test vector is shown on one line followed by the input portion of the test vector, Vector In, on the next line. The output portion of the test vector (the expected output of the device) appears on the Vector Out line below the actual output. Figure 2-7 shows an example of a Pins report.

Figure 2-7
Pins Report Simulation Output

```
Simulate ABEL 1.00X  Date Wed Aug 29 10:21:15 1993
Fuse file: 'fb2.jed' Vector file: 'regfb.tmv' Part: 'P16R4'
ABEL 1.00X Data I/O Corp. JEDEC file for: P16R4 V8.0

Created on: Wed Aug 29 10:21:07 1993
Operation of the simulator on devices with feedback

Vector 1
Vector In  [C0111.....0.............]

Device In  [001110000000011110001111]
Device Out [...........ZLHHHHZZ.....]

...edited...

Vector 4
Vector In  [00001.....0.............]

Device In  [000010000000001110000111]
Device Out [...........ZLLHHHZZ.....]

4 out of 4 vectors passed.
```

Report Type: Macro-cell

Macro-cell reports provide the same information as Tabular, with the addition of internal device information such as OR-gate outputs, register outputs, and the final outputs. Figure 2-8 shows one portion of a macro listing with pointers to its various parts and a portion of the corresponding logic diagram. Only a portion of the macro simulation output is shown since macro output files can be quite large.

Figure 2-8
Macro-cell Report Simulation Output

```
Simulate ABEL 1.00X  Date Wed Aug 29 10:22:32 1993

Fuse file: 'fb2.jed' Vector file: 'regfb.tmv' Part: 'P16R4'

ABEL 1.00X Data I/O Corp. JEDEC file for: P16R4 V8.0
Created on: Wed Aug 29 10:21:07 1993

Operation of the simulator on devices with feedback
       Data I/O Corp.      31 July 1993

Vector 1
Vector In  [C0111.....0.............]

Pin  11 [0                      ]----------------------
                                                     O
                                       F1  Pin  14   |\
                                                  --| >O--- H   Vec=H
                                     ----------   |  |/
                                    |   Q = L  |--
PT 1280 [FFFFFFFF               ]---|  OR = L  |
Pin   1 [0                      ]---|  CK = L  |
                                     ----------

PT 1536 [T                      ]----------------------
                                                      |
                                       F2  Pin  13   |\
                                                  --| >O--- L   Vec=L
                                     ----------   |  |/
                                    |          |--
PT 1568 [TFFFFFF                ]---|  OR = H  |
                                     ----------

Vector Out [............LH.........]
 ...
```

As shown in Figure 2-8, the fusemap program assigns fuse and node numbers to the fuses in the device (refer to the Product Term starting at fuse 1536). These numbers are also shown in the Logic Diagrams provided with the PLD Device Kit. The OR-gate and register outputs in the simulation output are internal signals, not available as pin outputs. These OR-gate and register outputs can be useful for debugging designs. The Macro-cell report option produces large files if all pins and nodes are traced for all vectors. To generate a smaller file, use the Watch Signals and Vector Range to Display properties (to specify the pins or nodes for a limited number of vectors). If you do not specify which signals to watch, the first I/O pin in the device is traced. Table 2-1 describes the notation used in the simulation macro output files.

Table 2-1
Notation Used in Simulation Macro-cell Report Files

Notation	Description
Current Nodes	
OE	Output enable
AR	Asynchronous Reset
SR	Synchronous Reset
AP	Asynchronous Preset
SP	Synchronous Preset
LD	Register Load
CK	Register Clock
OR	Normal output OR gate ("D" "T")
IN1	First input to a Flip/Flop ("J" "S")
IN2	Second input to a Flip/Flop ("K" "R")
OR Node Types	
PTnnnn	One or more product term
LOW	Always logic level 0
HIGH	Always logic level 1
Pin nn	Input from pin
Node nn	Input or feedback from a internal node
Pin nn & nn	The AND of two pins
Pin nn # nn	The OR of two pins
PROM nn	Bit nn of a prom output

Notation	Description
PRODUCT Term Display	
Pin nn [1]	Logic level 1 from a pin or node
Pin nn [0]	Logic level 0 from a pin or node
nn & nn [0 & 1]	Logic level 0 from a pin or node
PTnnnnn [TTFFTT]	Multiple product terms
PTnnnnn [TT $ FT]	XOR of two groups of product terms
PTnnnnn [TT # FT]	OR of two groups of product terms
PTnnnnn [T-FF-T]	Shared product terms ('-' term not connected)
PTnnnnn [TTTTTFTFTTTFFFFTTF] [TFFFFFFTTF]	Multiple line display of large OR

T = logic true; F = logic false

Waveform Viewer

The Waveform Viewer provides a graphic representation of the inputs and / or outputs for each of the specified test vectors. You can invoke the Waveform Viewer by double-clicking on the "Equation Simulation Waveform" process or the "JEDEC Simulation Waveform" process in the *Processes for Current Source* window. You must set the Trace Type property to "Detailed" (the default) in order to see the clock edges in the Waveform Viewer. For more information about using the Waveform Viewer, refer to the Waveform Tools manual.

Chapter 3
Simulation and Designs with Buffered Outputs

When a design with 3-state buffered outputs is simulated with wave and/or tabular reports, the states of the outputs are reported as H, L, 1, 0, Z, or X, depending on the test vectors used, whether the pin is bidirectional, and whether the output buffer is enabled.

A Tabular report lists device pins that are output-only, or bidirectional and configured as outputs. These outputs are reported (in order of significance) in the following manner:

- If the buffer is enabled, the active state (H or L) of the output is reported. This output results from the levels that are applied at the input pins by the input test vector.
- If the buffer is not enabled, the same value (1, 0, Z, or X) that was applied to that output by the input test vector is reported.
- If the output is not enabled, and no 1 or 0 is applied to the output by the input test vector, Z is reported.

Simulation and Unspecified Inputs

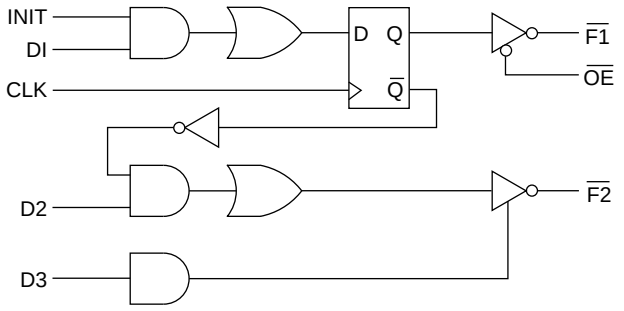
When the input test vector does not specify a logic level to be applied to a particular input, or set of inputs, simulation uses the default value assigned by the -x option. Using don't cares (Xs) in the input test vector can cause the input(s) to be unspecified. The simulator does not propagate unknown values.

Simulation for Designs with Feedback

Logic designs containing feedback present a unique simulation problem because the current output on one or more gates depends on the outputs of other gates. Thus, determining the outputs of a design with feedback is not a simple input-to-output determination. Propagation delays, the number of gates in the feedback path, and (in synchronous feedback circuits) clock inputs must be taken into account. When an input to the design changes, the outputs may not stabilize immediately. Synchronous circuits must be clocked before the outputs reflect changes in the inputs.

The simulator determines the final outputs of feedback circuits through iteration, calculation, and monitoring of the outputs until they stabilize or are clocked out to give the final outputs. If outputs do not stabilize after 20 iterations, an error message is given. The iterations, final outputs, and the states of the internal register are provided in the simulation output file, depending on the report and trace type you choose. Figure 3-1 shows a simple synchronous circuit with feedback. One clock pulse is required after the inputs change to cause a corresponding change in the outputs. The source file describing this circuit and the simulation output for a Tabular report are shown in Figures 2-3 and 2-4 in the previous chapter.

Figure 3-1
Synchronous Feedback Circuit



The Tabular report shows the test vectors and the final outputs after the clock pulse. A tabular report with clock trace shows the test vectors and the value of the outputs before and after the clock. A macro-cell report results in a large simulation output file. If you wish to examine the trace output for this circuit, you can run JEDEC simulation on regfb.abl with macro-cell report and examine the simulation results.

The second feedback example in Figure 3-2 shows an asynchronous circuit that requires more than one simulation iteration before the outputs stabilize. Figure 3-3 shows the source file describing the circuit and Figures 3-4 and 3-5 show the simulation output for brief trace and detailed trace.

Brief trace shows the final outputs after they have stabilized, as well as the test vectors. Detailed trace shows the output values at the different iterations as the outputs stabilize, as well as the final outputs and the test vectors. Notice that for the inputs provided in vector 2, three iterations are needed before the outputs stabilize. Vector 1 requires only one iteration to provide stable outputs. Macrocell report is not shown but can be generated by running JEDEC simulation with "feedback.abl" shown in Figure 3-3.

Figure 3-2
Asynchronous Feedback Circuit

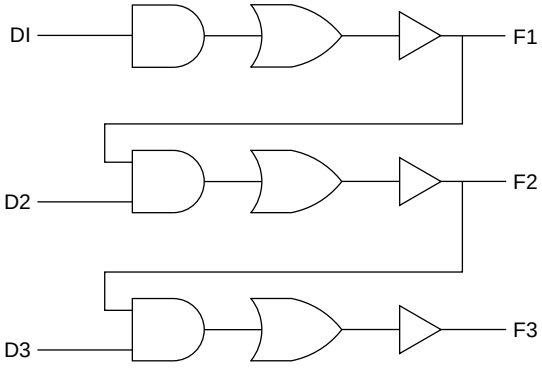


Figure 3-3
Source File: Asynchronous Feedback Circuit

```
module feedback
 title 'Operation of the simulator on devices with feedback'
        FB1     device 'P16HD8';
        D1,D2,D3        pin 1,2,3;
        F1,F2,F3        pin 13,14,15;
equations
        F1      = D1;
        F2      = D2 & F1;
        F3      = D3 & F2;
test_vectors   ([D1,D2,D3] -> [F1,F2,F3])
                [ 0, 0, 0] -> [ 0, 0, 0];
                [ 1, 1, 1] -> [ 1, 1, 1];
end feedback
```

Figure 3-4
Brief Trace Simulation Output: Asynchronous Feedback Circuit

```
Simulate ABEL 1.00  Date Mon Aug  6 14:55:57 1993

Fuse file: 'feedback.jed' Vector file: 'feedback.tmv'

Operation of the simulator on devices with feedback
                    DATA I/O Corp.      24 Feb 1993

       D D D   F F F
       1 2 3   1 2 3

V0001  0 0 0   L L L
V0002  1 1 1   H H H
```

Figure 3-5
Detailed Trace Simulation Output: Asynchronous Feedback Circuit

```
2 out of 2 vectors passed.

Simulate ABEL 1.00  Date Mon Aug  6 14:57:01 1993
Fuse file: 'feedback.jed'  Vector file: 'feedback.tmv'

Operation of the simulator on devices with feedback
                    DATA I/O Corp.      24 Feb 1993

       D D D   F F F
       1 2 3   1 2 3

V0001  0 0 0   L L L
V0002  1 1 1   H L L
       1 1 1   H H L
       1 1 1   H H H
2 out of 2 vectors passed.
```

Register Preloads in the Simulator

When you use a preload vector as the first test vector in the simulator, and the device you are using has asynchronous presets, you should include a "dummy" vector (all "don't cares") before the preload vector. The dummy vector prevents the possibility that the simulator initialization may destroy the preload data. You should be cautious about setting don't cares to 1. When you preload registers to the 1 state, be sure to define values for the resets of the preloaded registers. If the resets are not defined, the registers will be reset after the preload operation, and the preloaded data will be lost.

Test Vectors and Simulation

Equation simulation uses design test vectors, that you supply, to simulate the design (independent of any device). JEDEC simulation uses design test vectors to simulate a programmed device. The more comprehensive and detailed your test vectors are, the more useful your simulation results will be.

When using test vectors, you specify the required input pattern and the expected outputs at the device pins. The simulator will apply the inputs from the test vectors to the simulated circuit and compare the simulated output with the output specified in the test vectors. If there is any difference, an error is indicated.

Note that the simulators cannot test a mode in your design if you do not write a test vector to force that mode of operation. It is to your advantage to create complete sets of test vectors that test all functions of your logic design, and to use simulation regularly as you implement design changes.

Note: *The nodes shown on logic diagrams that are shipped with the PLD Device Kit are outputs for writing equations (for example, the OR terms for the RS flip/flops in an F167). These nodes cannot be used as direct flip/flop inputs for simulation test vectors.*

Debugging State Machines

State machines can be difficult to debug when an error occurs, because each state depends on previous states and affects subsequent states. An error in the description of one state transition can cause the state machine to follow a sequence different from the test vectors. The loss of synchronization can cause a cascade of errors that makes the original error difficult to isolate.

To test and debug larger state machines:

- Add test vectors that periodically force the machine to known states. This will reset the state machine and eliminate the possibility of a loss of synchronization.
- Write small sets of test vectors that test individual functions of the state machine, then gradually add the test vectors to the simulation.

By periodically forcing (with your test vectors) the state machine to a known state, and then letting the state transitions take place, you limit the cascading of errors to a smaller number of states. This will make it much easier to find initial errors.

Starting with a small set of test vectors, that test only part of the state machine's function, also helps isolate any errors. When the operation of one function has been verified, add a set of test vectors that test another function, then add another, and so on, until you have tested the full function of the state machine. Combining this technique of gradual simulation with the use of test vectors that periodically force the machine to known states will make errors easier to pinpoint and simplify the testing of large state machines.

Multiple Test Vector Sections

You can use more than one set of test vectors to simulate the function of a device. This may be useful in the representation of the test in the source file. In a similar manner, any time you have two or more distinct functions being performed by the same device, you may want to describe the vectors in separate sections for each function.

Take, for example, the source file presented in Figure 3-6 that describes AND and NAND gates that are implemented on the same device, as well as the test vectors used to simulate the operation of the device. The test vectors are described in two separate sections. The first section lists the test vectors that simulate the operation of the AND portion of the design, and the second section tests the NAND function. With the test vectors written as they are, in two separate sections, the correspondence between test vectors and the function being tested is readily apparent in the source file.

Figure 3-6
Source File with Multiple Test Vector Sections

```
module simple
 title 'Simple ABEL-HDL example Dan Poole   Data I/O Corp'
                U7      device  'P14H4';
        A1,A2,A3        pin 1,2,3;
        N1,N2,N3        pin 4,5,6;
        AND,NAND        pin 14,15;
equations
         AND    = A1 & A2 & A3;
        !NAND   = N1 & N2 & N3;
test_vectors  'Test And Gate'
              ( [A1,A2,A3] -> AND )
                [ 0, 0, 0] ->  0;
                [ 1, 0, 0] ->  0;
                [ 0, 1, 0] ->  0;
                [ 0, 0, 1] ->  0;
                [ 1, 1, 1] ->  1;
test_vectors  'Test Nand Gate'
              ( [N1,N2,N3] -> NAND )
                [ 0, 0, 0] ->  1;
                [ 1, 0, 0] ->  1;
                [ 0, 1, 0] ->  1;
                [ 0, 0, 1] ->  1;
                [ 1, 1, 1] ->  0;
end simple
```

Automatic Signal Selection

The simulator shows only signals that are used in test vectors for all report formats (except for pins), unless you use specific signals with the Watch Signal property. (The Pins report type shows all inputs and outputs.) If you run the module shown above (module simple), with the default tabular format, the simulation results are given for only the signals used in the test vectors (Figure 3-7):

Figure 3-7
Simulation Results Showing Automatic Signal Selection

```
Simulate ABEL 1.00  Date Thu Aug  9 14:45:02 1993

Fuse file: 'simple.jed'  Vector file: 'simple.tmv'

Simple ABEL-HDL example

***** Test And Gate *****

                A
       A A A    N
       1 2 3    D

V0001  0 0 0    L
V0002  1 0 0    L
V0003  0 1 0    L
V0004  0 0 1    L
V0005  1 1 1    H

***** Test Nand Gate *****

                N
                A
       N N N    N
       1 2 3    D

V0006  0 0 0    H
V0007  1 0 0    H
V0008  0 1 0    H
V0009  0 0 1    H
V0010  1 1 1    L
10 out of 10 vectors passed.
```

Don’t Cares in Simulation

In ABEL-HDL, you can use the special constant .X. in a test vector to denote a don’t-care input or output. The .X. tells the simulator to choose a value for the input designated by the .X. in the test vector(s), or to disregard an output signal’s state. The default value used in the simulator for the don’t care inputs is zero. However, you can use the Don’t Care (X) Value property to specify zero or one (0 or 1) for the don’t care value.

Input pins that are not specified in the test vectors are given a default don’t care value of zero (0) by the simulator unless the don’t care property is set to 1. In this case, all unspecified pins will be assigned a value of 1 in the test vectors.

If you experience trouble with devices not working in a circuit or programmer / tester, it may be helpful to recheck the don’t care assumptions. There may be incorrectly set 1s and 0s in a test vector. If this is the case, you will need to use the simulator to check the test vector.

Note: *Logic programmers use JEDEC vectors to test the device. Occasionally vectors that pass in JEDEC simulation will fail on the programmer. Data I/O logic programmers such as the UniSite, 2900 or 3900 display the failed pin and vector on the operator’s terminal.*

Also, the simulator checks the design with a single level for the don’t care inputs, while the target circuit may place other levels on the input during actual operation of the device. For complete simulation, you must run the simulation operation with the don’t cares set to 0, and then again with them set to 1.

The simulators ignore output pins that are not specified in the test vectors and will not indicate an error due to conflict between a specified value and the value determined by the simulator. The .X. constant at an output pin tells the simulators not to compare the outputs (the output produced by the design and the output specified in the test vector) but still allow them to be displayed. Figure 3-8 shows how the .X. value can be assigned to the outputs prior to the JEDEC simulation step of the language processor.

Figure 3-8
Assignment of Don't Care Value (.x.) to Design Outputs

```
module findout
title 'The JEDEC simulator will find the output levels
Ngoc Nicholas    Data I/O Corp    9 Aug 1993'

        F1      device 'P16L8';
        A,B,Y1,Y2       pin 1,2,14,15;
        X = .X.;

equations
        !Y1 = A # B;
        !Y2 = A $ B;

test_vectors
       ([A,B] -> [Y1,Y2])
        [0,0] -> [ X, X];
        [0,1] -> [ X, X];
        [1,0] -> [ X, X];
        [1,1] -> [ X, X];
end
```

Using Pins, Wave, and Tabular report types, you can observe the actual output values determined by the simulator. In Figure 3-9, the X entries (in the test vectors) for pins 14 and 15 allow the simulator to display an H or L to indicate the output value for the specified inputs.

Figure 3-9
JEDEC Simulation Results with Outputs Specified as Don't Care

```
Simulate ABEL 1.00X  Date Tue Aug 28 15:07:54 1993

Fuse file: 'findout.jed' Vector file: 'findout.tmv'

The JEDEC simulator will find the output levels
Ngoc Nicholas    Data I/O Corp    9 Aug 1993

             Y Y
       A B   1 2

V0001  0 0   H H
V0002  0 1   L L
V0003  1 0   L L
V0004  1 1   L H
4 out of 4 vectors passed.
```

Preset and Preload Registers

Preset, reset, and preload are terms used to define a specific action and resultant output of one or more registers contained in a programmable logic device. Preset forces all register outputs to one, reset forces all register outputs to zero, and preload forces all registers to specified states. Synchronous preset, reset, and preload functions require a clock input. Asynchronous functions require no clock input.

To verify the operation of these devices, appropriate test vectors must be written and placed in the source file. These test vectors allow the simulation steps of the language processor to verify operation of the design by performing the operations required of these registers.

Note: *For preload, the simulator assumes that devices have inversion between the register outputs and the device outputs. When you preload devices that have noninverting outputs, or that have outputs that are programmable to noninverting, you must complement the data to be preloaded, to obtain the desired preload condition.*

Also note that it is not possible to preset, reset, and preload at the same time. Preset and preload must not contend during preload with other inputs, preset, or register functions.

Special Preset Considerations

Certain programmable logic devices, such as the F105 and F167, do not respond to the first clock pulse following a preset condition (power-on or the preset input). These devices allow normal clocking only after a high-to-low transition of the clock input, following the preset condition. Therefore, simulation for these devices requires an additional test vector, following the preset condition, to provide the high-to-low transition of the clock input.

To illustrate the preset considerations for these devices, a four-state counter with clock and preset inputs is presented in Figure 3-10, along with the test vectors required to properly verify the design. The equation for the preset condition is written using the dot extension for the two registers. This counter is targeted for a circuit that provides a power-on preset condition; so the test vectors must verify operation of the counter after power-on preset as well as after the preset input has been active.

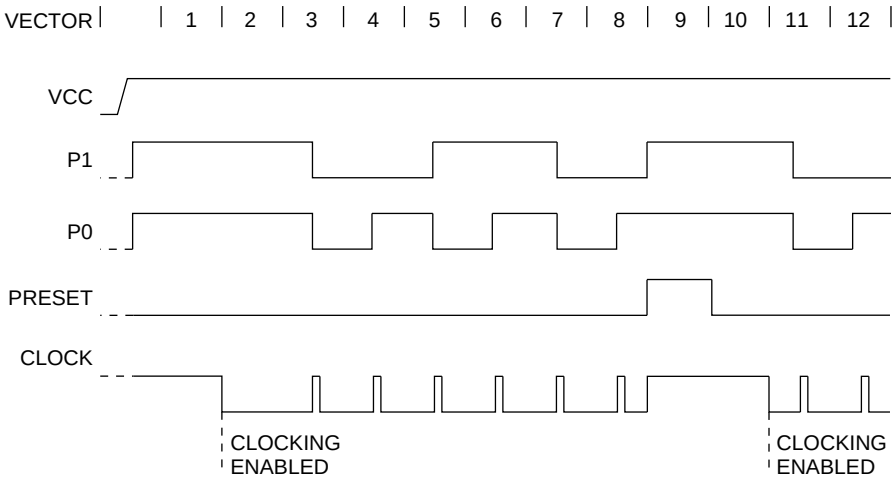
Figure 3-10
Test Vectors for Special Preset Conditions

```
 module preset
 title '2-bit counter to demonstrate power on preset Bob Hamilton   Data I/O Corp'
        preset  device 'F167';
        Clk,Hold        pin  1,2;
        PR              pin  16;        "Preset/Enable
        P1,P0           pin  15,14;
        Ck,X = .C.,.X.;
equations
        [P1,P0].PR  =  PR;
        [P1,P0].C   =  Clk;
        [P1.R,P0.S] = !P1 & !P0 & !Hold;   " state 0
        [P1.S,P0.R] = !P1 &  P0 & !Hold;   " state 1
        [P1.S,P0.S] =  P1 & !P0 & !Hold;   " state 2
        [P1.R,P0.R] =  P1 &  P0 & !Hold;   " state 3
test_vectors
       ([Clk,PR,Hold] -> [P1,P0])
        [ 1 , 1,  0 ] ->    3;
        [ 1 , 0,  0 ] ->    3; " Provides a High-to-Low on clock
        [ 0 , 0,  0 ] ->    3; " to enable clocking
        [ Ck, 0,  0 ] ->    0;
        [ Ck, 0,  0 ] ->    1;
        [ Ck, 0,  0 ] ->    2; " Hold count
        [ Ck, 0,  1 ] ->    2;
        [ Ck, 0,  0 ] ->    3;
        [ Ck, 0,  0 ] ->    0; " Roll over
        [ Ck, 0,  0 ] ->    1;
        [ 1 , 1,  0 ] ->    3; " Preset high
        [ 1 , 0,  0 ] ->    3; " Preset low
        [ Ck, 0,  0 ] ->    0;
        [ Ck, 0,  0 ] ->    1;
" Notes edited...
end
```

Figure 3-11 is a timing diagram that shows the action of the test vectors in Figure 3-10. As indicated in the timing diagram, the preset input overrides the clock input and, when held high, inhibits clocking of the counter. Assuming that the device is powered up in the preset condition, the first test vector pulls the clock input high while the second vector pulls it low to provide the transition required for normal clocking.

The next six test vectors provide clock inputs to increment the counter through all states and back to state one. As shown in the timing diagram, the 9th test vector invokes the preset function, and the 10th test vector pulls the preset input low and maintains the clock input high. The 10th test vector allows the preset line to go low before the high-to-low transition of the clock. The preset line must go low before the clock so the high-to-low clock transition can enable the clock pulse of the 11th test vector. The high-to-low transition that follows the 10th test vector resumes normal clocking of the device.

Figure 3-11
Timing Diagram Showing Test Vector Action



Preset overrides clock, and when held high, clocking is inhibited and the registers are high. Normal clocking resumes with the first clock pulse following a high-to-low clock transition after preset goes low.

0748-2

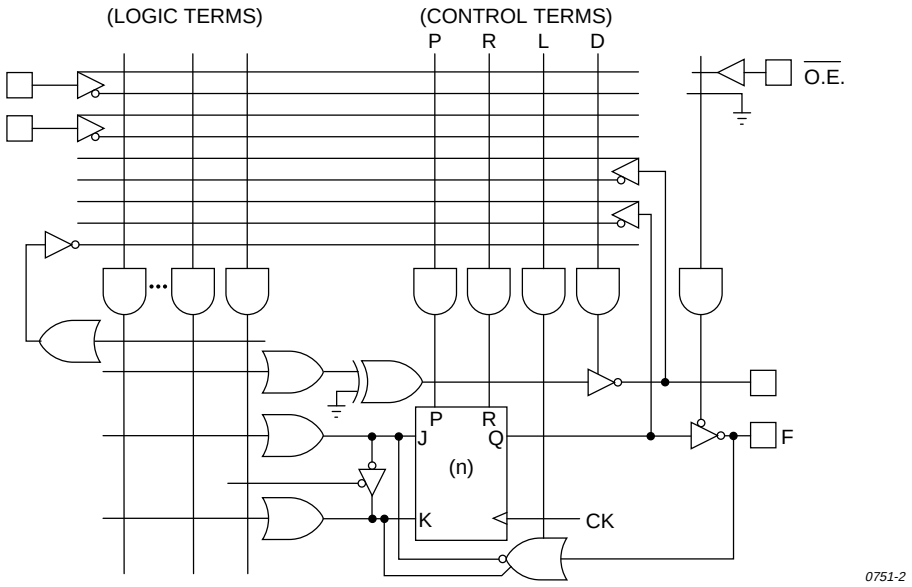
If the 2nd and 10th test vectors were not included in the source file, the clock pulse of the 3rd and 11th vectors would be lost. That is, the high-to-low transition of the clock pulses in these vectors would cause the resumption of normal clocking, but would not increment the counter as required by the design.

TTL Preload

Certain programmable logic devices, such as the F159 FPLS (field programmable logic sequencer) allow internal registers to be forced (preloaded) to a known state by means of the TTL preload function. Figure 3-12 shows a typical FPLS layout. To preload the output register in such a device, the following four conditions must be present:

- The output must be placed in the high-impedance state.
- The desired register state must be placed on the output pin.
- The load control term must be activated.
- A clock pulse must be applied to the clock input.

Figure 3-12
Internal Register of the F159



The fifth test vector in Figure 3-13 shows how each of the four conditions are met as follows:

- The OE (Ena) pin is held at 1.
- The output (F0) is pulled low by inserting a 0 in the input side of the test vector.
- A 1 is applied to the LOAD input, which activates the load control term (LA).
- A clock pulse is provided by the C (.C.) applied to the clock input

Figure 3-13
Invoking the TTL Preload Function

```
module TTLload
 title 'TTL load example
Dave Kohlmeier   Data I/O Corp  9 Aug 1993'

        TTL59   device  'F159';

        C,L,H,X,Z       = .C.,0,1,.X.,.Z.;

        Clk,J_IN,K_IN,LOAD,Ena,F0       pin 1,2,3,4,11,12;

        F0 istype 'reg_JK,invert';

equations
         F0.OE  = !Ena;
         F0.J   = J_IN;
         F0.K   = K_IN;
         F0.L   = LOAD;
         F0.C   = Clk;

test_vectors
       ([Clk,Ena,J_IN,K_IN,LOAD,F0] -> F0)
        [ C , L ,  1 ,  0 ,  0 , X] ->  0 ; "Set
        [ C , L ,  0 ,  1 ,  0 , X] ->  1 ; "Reset
        [ C , L ,  1 ,  1 ,  0 , X] ->  0 ; "Toggle
        [ C , L ,  1 ,  1 ,  0 , X] ->  1 ; "Toggle
        [ C , H ,  0 ,  0 ,  1 , 0] ->  X ; "Load
        [ 0 , L ,  0 ,  0 ,  0 , X] ->  0 ; "Test
        [ C , L ,  1 ,  1 ,  0 , X] ->  1 ; "Toggle
end
```

The sixth test vector tests to make sure the register was loaded with the 0 applied to the output by the fifth test vector. The sixth test vector enables the output (Ena at 0) and allows the output to be tested (F0 = X on the input side and 0 on the output side of the vector) while holding the clock input at 0.

Supervoltage Preload

Supervoltage preload allows the setting of registers within certain devices, such as the P16R4, to the logic levels placed on their registered outputs. Supervoltage preload is accomplished by means of the .P. test condition (.P. special constant) that is used to "jam load" registers, within the logic device, to the desired state. When the .P. test condition is applied to the clock pin, the logic level applied to the register output is loaded into the register. Devices with separate banks of registers require that the P test condition be applied to each clock pin. Also, during preload certain device pins, such as the output enable pin, may have to be in a defined state.

To verify the preload operation, use a separate test vector to test the outputs. This vector must follow the vectors that perform the preload operation.

Note: *For preload, the simulators assume devices have inversion between the register and device outputs. When you preload devices that have non inverting outputs (or outputs that are programmable to non inverting), you must complement the data to be preloaded. Also, it is not possible to preset, reset, and preload at the same time. Preset and preload must not contend with other inputs (or preset or register functions) during preload.*

Supervoltage preload can be used to test state machine designs that may assume one or more illegal states, or to test designs that contain branch conditions. An illegal state for a state machine is a state that the design does not allow, but the device is capable of assuming under certain conditions (such as powerup or noise). A typical decade counter having states 0 through 9 and four registers, could possibly assume six additional (and illegal) states (10 through 15). The decade counter should be designed so that when an illegal state is reached, the next clock pulse returns the counter to the 0 state. During simulation it is necessary to not only test the counter for normal up/down/clear operation (performed by the test vectors), but also to make sure it will clock to state 0 from any illegal state.

To test that a decade counter will clock to state 0 from any illegal state, the following two procedures are required:

1. Define all illegal states to be tested.
2. Create test vectors that verify the return to state 0 from any illegal state.

To test the illegal states for a decade counter, it is necessary to define the illegal states (that is, 10 through 15). Figure 3-14 shows that the following additional entries are included to define the six illegal states:

```
S10= ^b0101;
S11= ^b0100;
S12= ^b0011;
S13= ^b0010;
S14= ^b0001;
S15= ^b0000;
```

To verify that the design will recover from each of the illegal states, appropriate test vectors are included. This group of test vectors preloads the device to each possible illegal state and then verifies that the device clocks to state 0. The test vectors preload the counter by means of the .P. special constant applied to the clock pin and a logic high applied to the output enable (OE) pin. The test vector that follows the preload test vector verifies the result of the preload operation. The following test vector verifies the clocking of the counter from the illegal state to state 0.

Figure 3-14
Defining Illegal States and Test Vectors for Illegal States

```
 module CNT10P
 title 'decimal counter
Note: preload the data on pins into the registers, Denny Siu   Data I/O Corp'

        cnt10p          device 'P16R4';

        Clk,Clr,OE      pin 1,2,11;
        Q3,Q2,Q1,Q0     pin 14,15,16,17 istype 'reg_D,invert';

        Ck,X,Z,P        = .C. , .X., .Z., .P.;

" Counter States
        S0 = ^b1111;    S4 = ^b1011;    S8 = ^b0111;    S12= ^b0011;
        S1 = ^b1110;    S5 = ^b1010;    S9 = ^b0110;    S13= ^b0010;
        S2 = ^b1101;    S6 = ^b1001;    S10= ^b0101;    S14= ^b0001;
        S3 = ^b1100;    S7 = ^b1000;    S11= ^b0100;    S15= ^b0000;

equations
        [Q3,Q2,Q1,Q0].c = Clk;
        [Q3,Q2,Q1,Q0].oe = !OE;

state_diagram [Q3,Q2,Q1,Q0]
        State S0:       IF !Clr THEN S1 ELSE S0;
        State S1:       IF !Clr THEN S2 ELSE S0;
        State S2:       IF !Clr THEN S3 ELSE S0;
        State S3:       IF !Clr THEN S4 ELSE S0;
        State S4:       IF !Clr THEN S5 ELSE S0;
        State S5:       IF !Clr THEN S6 ELSE S0;
        State S6:       IF !Clr THEN S7 ELSE S0;
        State S7:       IF !Clr THEN S8 ELSE S0;
        State S8:       IF !Clr THEN S9 ELSE S0;
        State S9:       GOTO S0;
```

```
"Ensure return from illegal state
        State S10:      GOTO S0;
        State S11:      GOTO S0;
        State S12:      GOTO S0;
        State S13:      GOTO S0;
        State S14:      GOTO S0;
        State S15:      GOTO S0;

@page
test_vectors 'Test Counter'
      ( [Clk ,OE, Clr ] -> [Q3,Q2,Q1,Q0])
        [ Ck , 0,  1  ] ->    S0;
        [ Ck , 0,  0  ] ->    S1;
        [ Ck , 0,  0  ] ->    S2;
        [ Ck , 0,  0  ] ->    S3;
        [ Ck , 0,  0  ] ->    S4;
        [ Ck , 0,  0  ] ->    S5;
        [ Ck , 1,  0  ] ->    Z ;
        [ Ck , 0,  0  ] ->    S7;
        [ 0  , 0,  0  ] ->    S7;
        [ Ck , 0,  0  ] ->    S8;
        [ Ck , 0,  0  ] ->    S9;
        [ Ck , 0,  0  ] ->    S0;
        [ Ck , 0,  0  ] ->    S1;
        [ Ck , 0,  0  ] ->    S2;
        [ Ck , 0,  1  ] ->    S0;

test_vectors 'preload to illegal states'
      ( [Clk ,OE, Clr,[Q3,Q2,Q1,Q0]] -> [Q3,Q2,Q1,Q0])
        [ P  , 1,  0 ,      S10    ] ->     X ;
        [ 0  , 0,  0 ,       X     ] ->    S10;
        [ Ck , 0,  0 ,       X     ] ->    S0 ;
        [ P  , 1,  0 ,      S11    ] ->     X ;
        [ 0  , 0,  0 ,       X     ] ->    S11;
        [ Ck , 0,  0 ,       X     ] ->    S0 ;
        [ P  , 1,  0 ,      S12    ] ->     X ;
        [ 0  , 0,  0 ,       X     ] ->    S12;
        [ Ck , 0,  0 ,       X     ] ->    S0 ;
        [ P  , 1,  0 ,      S13    ] ->     X ;
        [ 0  , 0,  0 ,       X     ] ->    S13;
        [ Ck , 0,  0 ,       X     ] ->    S0 ;
        [ P  , 1,  0 ,      S14    ] ->     X ;
        [ 0  , 0,  0 ,       X     ] ->    S14;
        [ Ck , 0,  0 ,       X     ] ->    S0 ;
        [ P  , 1,  0 ,      S15    ] ->     X ;
        [ 0  , 0,  0 ,       X     ] ->    S15;
        [ Ck , 0,  0 ,       X     ] ->    S0 ;
end
```

An example of a state machine, containing branch conditions, is given in the blackjack machine in the ABEL-HDL Reference. If it was impossible to preload the state machine to each branch condition, it would be necessary to repeat the test vectors down to the Test22 state for each branch of the Test22 state. The test vectors in Figure 3-15 show how this state machine can be preloaded to test these three branches of the design.

Figure 3-15
Using Test Vectors to Preload a State Machine

```
 test_vectors ' Test 3 way branch at Test22'
([Ena,Clk,LT22,Ace,Qstate] -> [Ace,Qstate  ])
 [ 1  ,.P.,  X  1 ,Test22] -> [ X,   X     ];
 [ 0  , O ,  1  X ,   X  ] -> [ H,Test22   ];"Verify preload
 [ 0  , C ,  1  X ,   X  ] -> [ H,ShowStand];
 [ 1  ,.P.,  X  0 ,Test22] -> [ X,   X     ];
 [ 0  , O ,  0  X ,   X  ] -> [ L,Test22   ];
 [ 0  , C ,  0  X ,   X  ] -> [ L,ShowStand];
 [ 1  ,.P.,  X  1 ,Test22] -> [ X,   X     ];
 [ 0  , C ,  0  X ,   X  ] -> [ H,Sub_10   ];
```

Preset/Reset Controlled by Product Term

In programmable logic devices, such as the P22V10, preset and reset functions are controlled by product terms. An example of controlling the preset and reset functions is given in Figure 3-16.

Figure 3-16
Controlling Reset / Preset by Product Term

```
module reset22a
title 'Demonstrates Asynchronous Reset and Synchronous Preset'
        reset22a        device  'P22V10';
        Clk,I1,I2,R,S,T Pin 1,2,3,4,5,6;
        Q1,Q2           Pin 14,15 istype 'buffer';
        Ck,Z,H,L        = .C., .Z., 1, 0;
        Input           = [I2,I1];
        Output          = [Q2,Q1];
equations
        Output    := Input;   "Registered buffer
        Output.AR  = R & !T;
        Output.SP  = S & !T;
        Output.Clk = Clk;
test_vectors
       ([Clk,Input,R,S,T] -> Output)
        [ Ck,  0  ,0,0,0] ->   0;
        [ Ck,  1  ,0,0,0] ->   1;
        [ Ck,  2  ,0,0,1] ->   2;
        [ 0 ,  3  ,0,0,1] ->   2;    "Hold
        [ Ck,  3  ,0,0,1] ->   3;
        [ 0 ,  3  ,1,0,1] ->   3;    "Reset = R & !T
        [ 0 ,  3  ,1,0,0] ->   0;    "Async Reset
        [ 0 ,  0  ,0,1,0] ->   0;    "Preset requires clock
        [ Ck,  0  ,0,1,0] ->   3;    "Sync Preset
end
```

Note: *In devices like the P22V10, there is a common reset for all registers. While a reset equation for one output will reset all of the registers in the P22V10, you should write individual reset equations to ensure architecture-independence.*

Note: *For devices that have programmable polarity at the output of the registers, preset and reset functions may complement the output pins.*

In the P22V10, the reset is asynchronous while preset is synchronous (to the clock input). The test vectors in Figure 3-16 verify the reset and preset functions. The first three vectors verify the loading of input data. (Note that the third vector pulls the T input high. This is in preparation for T to be pulled low later in the simulation.) The fourth vector verifies operation of the clock input by changing the input data, without loading a new clock input. The sixth vector verifies that the R input, without a low T input, will not provide the asynchronous reset.

The seventh vector verifies operation of the asynchronous reset by pulling the T input low. The next vector verifies that a high S input and low T input do not preset the device without the clock input. The final vector verifies the synchronous preset by providing the clock pulse and testing the output for both output pins at logic high (decimal 3).

Preset/Reset Controlled by Pin

Some devices have a direct Preset or Preload coming from a pin. Be sure to include this in the test vectors, or simulation errors may occur.

Powerup States

Some devices power up with registers set to 1, some with registers set to 0, and some with registers set to an unknown value. For example, some TI devices power up with registers set and outputs low while some AMD devices power up with registers clear and outputs high. The first test vectors should always place the device in a known state. For example:

```
test_vectors
([clk,clear,input]->[output])
 [  0 ,   0   ,   0  ]->[   .X.    ];  "Power up to don't care
test
 [  0 ,   1   ,   0  ]->[    0     ];  "Clear the f/f and put
output to known value
 [  0 ,   0   ,   0  ]->[    0     ];  "Relax clear signal and
test for steady state
 [ .C.,   0   ,   0  ]->[    0     ];  "Begin clocking
```

Devices with Clock Inputs

Since devices with registered outputs must be clocked before the outputs reflect any change in inputs, a clock pulse must be specified as one of the test vector inputs. A clock input is indicated by a .C. in the test vector for a low-high-low pulse, and a .K. for a high-low-high pulse. The clock input causes JEDEC simulation to evaluate the inputs to the outputs prior to the first clock pulse transition (low-to-high or high-to-low depending on the polarity of the clock signal). The evaluation consists of the iterative steps described in "Simulation Program Operation." The inputs to outputs are then evaluated with the clock input at its active state, and then again with the clock input at its inactive state.

When running simulation with Trace Type of Clock or Detailed, simulation data will be written for all three evaluations. That is, internal test vectors are generated to evaluate the design before the first clock transition, after the first clock transition, and after the second clock transition, thus effectively expanding the number of test vectors. An example of JEDEC simulation output for a device with a clock input is shown in Figure 2-2 in the "Trace Type: Clock" section of Chapter 2, *Report and Trace Types and Break Points.* The clock input is represented by the .C. on the Vector In line. On the four subsequent Device In lines, the .C. input goes from 0 to 1 and back to 0, to provide one complete clock pulse.

Index

A

Architecture independence
    and reset 3-20
Automatic signal selection 3-8

B

Branch conditions, testing 3-19
Brief trace 2-2
Buffered outputs and simulation 3-1

C

Clock inputs 3-21
    See also feedback
    and synchronous functions 3-11
Clock trace 2-3
    and test vectors 3-22
cnt10p.abl 3-17

D

Debugging state machines 3-6
Detailed trace 2-4
Devices with clock inputs 3-21
Don't cares and simulation 3-9

E

Errors, pinpointing in simulation 2-1

F

Feedback
    and asynchronous circuits 3-3
    and simulation 3-2
    and synchronous circuits 3-2
feedback.abl 3-4
findout.abl 3-10
FPLS, preloading internal registers 3-13

I

Illegal states
    defining 3-16
    example 3-16
    and supervoltage preload 3-16
Initialization, and preload vectors 3-5
Inputs, unspecified in simulation 3-1
Internal registers, TTL preload 3-13
Inverting outputs and preload 3-11

J

JEDEC simulation
    macrocell report 2-8
    model of 1-3
    no report 2-5
    and test vectors 3-5

L

Limiting simulation output 2-1

M

Macro-cell report 2-8
Macrocell trace, notation 2-9

N

No report 2-5
Non-inverting outputs and preload 3-11
Notation, macrocell trace 2-9

O

Outputs
buffered 3-1
stabilizing in simulation 3-2

P

P22V10, example test vectors 3-20
Pins trace 2-7
Pins, controlling preset / reset 3-21
PLA simulation (MACH4) v
Powerup state 3-21
Preload registers 3-11
assume inverting 3-11
supervoltage 3-16
Preset
controlling with pins 3-21
controlling with product terms 3-20
synchronous 3-20
Preset registers 3-11
example 3-11
special considerations 3-11
timing diagram 3-12
preset.abl 3-12
Product terms, controlling preset / reset 3-20
Programmable polarity and reset 3-20
Propagation delays, feedback 3-2

R

regfb.abl 2-5
Registers
asynchronous presets 3-5
preload 3-5
preset and preload 3-11
Report Type
macro-cell 2-8
none 2-5
Reset 3-11
and architecture-independence 3-20
asynchronous 3-20
controlling with pins 3-21
controlling with product terms 3-20
and programmable polarity 3-20
reset22a.abl 3-20

S

Signals, automatic selection of 3-8
simple.abl 3-7
Simulation
automatic signal selection 3-8
brief trace 2-2
and buffered outputs 3-1
clock trace 2-3
detailed trace 2-4
don't care values 3-9
and feedback 3-2
limiting data 2-1
macrocell trace notation 2-9
multiple test vector sections 3-7
pinpointing errors 2-1
pins trace 2-7
register preloads 3-5
stabilizing outputs 3-2
Tabular trace 2-6
and unspecified inputs 3-1
State machine example 3-16
State machines
debugging 3-6
illegal states 3-16
Structured vector failure 3-9
Supervoltage preload 3-16

T

Tabular trace 2-6
Terms
product 3-20
Test vectors
and branch conditions 3-19
conversion of JEDEC 1-3
and illegal states 3-16
in .tmv file 1-3
multiple sections 3-7
and powerup states 3-21
and preset, reset, and preload registers 3-11
preset considerations 3-11
for reset / preset 3-20
and simulation 3-5
and state machines 3-6
and supervoltage preload 3-16

Trace
brief 2-2
clock 2-3
detailed 2-4
pins 2-7
Tabular 2-6
TTL preload 3-13
ttlload.abl 3-15

U

Unspecified inputs in simulation 3-1
Unspecified pins 3-9