VHDL AND WAVES DESCRIPTIONS FOR A PSEUDO-RANDOM PATTERN GENERATOR

Karen M. Serafino, Capt.
Michael A. Dukes, Capt.
Design Branch
Microelectronics Division

January 10, 1992

Final Report For Period April 1991 to October 1991

Approved for public release; distribution unlimited.

SOLID STATE ELECTRONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6543

92-07725
NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Karen M. Serafini
KAREN M. SERAFINO, CAPT, USAF
Design Branch
Microelectronics Division

Darrrell Barker, Acting Chief
DARRELL BARKER, Acting Chief
Design Branch
Microelectronics Division

Stanley E. Wagner, Chief
STANLEY E. WAGNER, Chief
Microelectronics Division
Solid State Electronics Directorate

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify WL/ELED, WPAFB, OH 45433-6543 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.
A VHDL model for a 32-bit linear feedback shift register is documented, along with a WAVES dataset that provides test vectors for it. The register operates with a leading-edge dual clock, an asynchronous reset, and control bits to enable shifting (to the right) and loading. Other inputs are a polynomial and a seed value; the output is a bit pattern. The feedback feature is controlled by the polynomial. For each polynomial bit position with a value of '1', the value of the register corresponding to that bit is fed through feedback circuitry before being shifted.

The WAVES dataset is used to generate input test vectors for the register, and to provide a utility to compare actual outputs to predicted outputs. There are three different architectures for the register: structural, behavioral, and structural with multiple component instantiations produced by the VHDL generate statement. There are also two architectures for the test bench: one that uses the WAVES dataset, and one that produces input vectors internally. The WAVES test bench uses a built-in utility to check outputs from each of the three register architectures against each other, and the non-WAVES test bench does these checks internally.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. VHDL Source Code</td>
<td>1</td>
</tr>
<tr>
<td>II. WAVES Support Source Code</td>
<td>23</td>
</tr>
<tr>
<td>III. Test Bench Programs</td>
<td>27</td>
</tr>
<tr>
<td>IV. Command File</td>
<td>36</td>
</tr>
<tr>
<td>V. Conclusions</td>
<td>38</td>
</tr>
<tr>
<td>VI. References</td>
<td>39</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Schematic of prpt.stage1.</td>
<td>18</td>
</tr>
<tr>
<td>2. Schematic of prpt.stage.</td>
<td>19</td>
</tr>
<tr>
<td>3. Schematic of prpt.</td>
<td>20</td>
</tr>
<tr>
<td>4. Schematic of 4-Stage prpt.</td>
<td>20</td>
</tr>
</tbody>
</table>
I. VHDL Source Code

--- std.vhd ---

---
--- This file contains VHDL descriptions for the basic components identified by the base_cmos GES system.
--- Descriptions for the transistors are also included.
--- The body of most of the code must be filled in by the user.

entity INV is
  generic(constant tPLH:TIME:=0 ns;
           constant tPHL:TIME:=0 ns);
  port (signal A:in bit;
        signal B:out bit);
end;
architecture inv of inv is
begin
  B <= not A; -- or use this.
end inv;

entity INVZ is
  generic(constant tPLH:TIME:=0 ns;
           constant tPHL:TIME:=0 ns);
  port (signal P:in bit;
        signal N:in bit;
        signal A:in bit;
        signal B:out bit);
end;
architecture invZ of invZ is
begin
  -- your code goes here.
end invZ;

entity NANDGATE is
  generic(constant tPLH:TIME:=0 ns;
           constant tPHL:TIME:=0 ns);
  port (signal A:in bit;
        signal B:in bit;
        signal C:out bit);
end;
architecture nand_gate of nand_gate is
begin
  C <= A nand B; -- or use this.
end nand_gate;

entity NAND3_GATE is
entity XOR_GATE is
generic(constant tPLH:TIME:=0 ns;
constant tPHL:TIME:=0 ns);
port (signal A:in bit;
signal B:in bit;
signal C:out bit);
end;
architecture xor_gate of xor_gate is
begin
--your code goes here.
C <= A xor B; -- or use this.
end xor_gate;

entity XOR_GATE2 is
generic(constant tPLH:TIME:=0 ns;
constant tPHL:TIME:=0 ns);
port (signal A:in bit;
signal Anot:out bit;
signal B:in bit;
signal Bnot:out bit;
signal C:out bit);
end;
architecture xor_gate2 of xor_gate2 is
begin
--your code goes here.
C <= A xor B; -- or use this.
Anot <= not A;
Bnot <= not B;
end xor_gate2;

entity NOR_GATE is
generic(constant tPLH:TIME:=0 ns;
constant tPHL:TIME:=0 ns);
port (signal A:in bit;
signal B:in bit;
signal C:out bit);
end;
architecture nor_gate of nor_gate is
begin
--your code goes here.
C <= A nor B; -- or use this.
end nor_gate;

entity ptrans is
  generic (constant gate_length: integer := 0;
              constant gate_width: integer := 0);
  port ( Gate : in bit;
        Drain : inout bit;
        Source : inout bit);
end ptrans;
architecture ptrans of ptrans is
begin
--your code goes here.
end ptrans;

entity ntrans is
  generic (constant gate_length: integer := 0;
              constant gate_width: integer := 0);
  port ( Gate : in bit;
        Drain : inout bit;
        Source : inout bit);
end ntrans;
architecture ntrans of ntrans is
begin
--your code goes here.
end ntrans;

entity tgate is
  generic (constant tPLH: TIME := 0 ns;
              constant tPHL: TIME := 0 ns);
  port ( p1 : in bit;
        p2 : in bit;
        g : inout bit;
        d : inout bit);
end tgate;
architecture tgate of tgate is
begin
--your code goes here.
end tgate;

-- and_gate.vhd --
entity and_gate is
  port (  
    I, Y    : in bit;  
    Output  : out bit 
  );
end and_gate;

architecture and_gate of and_gate is
  component nand_gate
    generic(constant tPLH:TIME:=0 ns;  
             constant tPHL:TIME:=0 ns);
    port (signal A:in bit;  
           signal B:in bit;  
           signal C:out bit);
    end component;

  for all : nand_gate use entity work.nand_gate( nand_gate );

  component inv
    generic(constant tPLH:TIME:=0 ns;  
             constant tPHL:TIME:=0 ns);
    port (signal A:in bit;  
           signal B:out bit);
    end component;

  for all : inv use entity work.inv( inv );

  signal Interm : bit;

  begin
    C1: NAND_GATE port map (X, Y, Interm);  
    C2: INV port map (Interm,Output);
  end and_gate;

-- or_gate.vhd --

entity or_gate is
  port (  
    X, Y    : in bit;  
    Output  : out bit 
  );
end or_gate;

architecture or_gate of or_gate is
  component nor_gate
    generic(constant tPLH:TIME:=0 ns;
constant tPHL:TIME:=0 ns;
port (signal A:in bit;
     signal B:in bit;
     signal C:out bit);
end component;

for all : nor_gate use entity work.nor_gate( nor_gate );

component inv
    generic(constant tPLH:TIME:=0 ns;
             constant tPHL:TIME:=0 ns);
    port (signal A:in bit;
          signal B:out bit);
end component;

for all : inv use entity work.inv( inv );

signal Interm : bit;

begin
    C1: NOR_GATE port map (X, Y, Interm);
    C2: INV port map (Interm, Output);
end or_gate;

-- dff_reset_beh.vhd --

entity dff_reset is
    port(clock : in bit;
         clock_bar : in bit;
         D : in bit;
         reset : in bit;
         Q_bar : out bit);
end dff_reset;

architecture beh of dff_reset is
begin
    process(clock,D,reset)
    begin
        if reset = '1' then
            Q_bar <= '1';
        elsif clock = '1' then
            Q_bar <= not D;
        end if;
    end process;
end beh;

-- prpt.stage1.vhd --
entity prpt_stage1 is
port(seed    : in bit;
load_seed  : in bit;
reset      : in bit;
input      : in bit;
clock1     : in bit;
clock1_bar : in bit;
clock2     : in bit;
clock2_bar : in bit;
test_enable: in bit;
output     : out bit);
begin
  assert ((not clock1 = clock1_bar) or (not clock1'stable(2 ns)))
  or (now = 0 ns)) report
  "bad clock1" severity failure;
  assert ((not clock2 = clock2_bar) or (not clock2'stable(2 ns))) report
  "bad clock2" severity failure;
end prpt_stage1;
architecture structural of prpt_stage1 is
component dff_reset
port(clock    : in bit;
      clock_bar : in bit;
      D         : in bit;
      reset     : in bit;
      Q_bar     : out bit);
end component;
for all : dff_reset use entity work.dff_reset(beh);
component and_gate
port (signal X,Y:in bit;
      signal Output:out bit);
end component;
for all : and_gate use entity work.and_gate( and_gate );
component Xor_Gate
  generic(constant tPLH:TIME:=0 ns;
          constant tPHL:TIME:=0 ns);
port (signal A:in bit;
      signal B:in bit;
      signal C:out bit);
end component;
for all : Xor_Gate use entity work.Xor_Gate( xor_gate );
signal interm1,interm2,interm3,interm4 : bit;
signal zero : bit := '0';
begin

dff_reset1 : dff_reset port map(clock1,clock1_bar,intern2,reset,intern3);
dff_reset2 : dff_reset port map(clock2,clock2_bar,intern3,zero,output);
and_gate1 : and_gate port map(load_seed,seed,intern1);
and_gate2 : and_gate port map(input,test_enable,intern4);
xor_gate1 : xor_gate port map(intern4,intern1,intern2);
end structural;

-- prpt.stage.vhd --

entity prpt_stage is
port(seed : in bit;
    load_seed : in bit;
    polynomial : in bit;
    feed_in : in bit;
    reset : in bit;
    input : in bit;
    clock1 : in bit;
    clock1_bar : in bit;
    clock2 : in bit;
    clock2_bar : in bit;
    test_enable : in bit;
    feed_out : out bit;
    output : out bit);
begin
assert (((not clock1 = clock1_bar) or (not clock1'stable(2 ns)))
    or (now = 0 ns)) report
"bad clock1" severity failure;
-- assert ((not clock2 = clock2_bar) or (not clock2'stable(2 ns))) report
-- "bad clock2" severity failure;
end prpt_stage;
architecture structural of prpt_stage is
component prpt_stage1
port(seed : in bit;
    load_seed : in bit;
    reset : in bit;
    input : in bit;
    clock1 : in bit;
    clock1_bar : in bit;
    clock2 : in bit;
    clock2_bar : in bit;
    test_enable : in bit;
    output : out bit);
end component;

for all : prpt_stage1 use entity work.prpt_stage1(structural);
component and_gate
  port (signal X,Y:in bit;
        signal Output:out bit);
end component;

for all : and_gate use entity work.and_gate( and_gate );

component Xor_Gate
  generic(constant tPLH:TIME:=0 ns;
           constant tPHL:TIME:=0 ns);
  port (signal A:in bit;
        signal B:in bit;
        signal C:out bit);
end component;

for all : Xor_Gate use entity work.Xor_Gate( xor_gate );

signal interm : bit;

begin
  prpt_stagel_1 : prpt_stagel
  port map(seed,load_seed,reset,input,clock1,clock1_bar,clock2,
           clock2_bar,test_enable,output);
  and_gate1 : and_gate port map(polynomial,input,interm);
  xor_gate1 : xor_gate port map(feed_in,interm,feed_out);

end structural;

-- prpt.vhd --

entity prpt is
  port(seed : in bit_vector(31 downto 0);
       load_seed : in bit;
       polynomial : in bit_vector(31 downto 0);
       reset : in bit;
       clock1 : in bit;
       clock1_bar : in bit;
       clock2 : in bit;
       clock2_bar : in bit;
       test_enable: in bit;
       pattern : out bit_vector(31 downto 0));
begin
  assert (((not clock1 = clock1_bar) or (not clock1'stable(2 ns)))
  or (now = 0 ns)) report
  "bad clock1 result" severity failure;
  -- assert (((not clock2 = clock2_bar) or (not clock2'stable(2 ns))) report
  -- "bad clock2 result" severity failure;
end prpt;
architecture structural of prpt is

component prpt_stage
port(seed : in bit;
     load_seed : in bit;
     polynomial : in bit;
     feed_in : in bit;
     reset : in bit;
     input : in bit;
     clock1 : in bit;
     clock1_bar : in bit;
     clock2 : in bit;
     clock2_bar : in bit;
     test_enable: in bit;
     feed_out : out bit;
     output : out bit);
end component;

for all : prpt_stage use entity work.prpt_stage(structural);

component prpt_stage1
port(seed : in bit;
     load_seed : in bit;
     reset : in bit;
     input : in bit;
     clock1 : in bit;
     clock1_bar : in bit;
     clock2 : in bit;
     clock2_bar : in bit;
     test_enable: in bit;
     output : out bit);
end component;

for all : prpt_stage1 use entity work.prpt_stage1(structural);

component and_gate
port (signal X,Y:in bit;
      signal Output:out bit);
end component;

for all : and_gate use entity work.and_gate( and_gate );

component inv
    generic(constant tPLH:TIME:=O ns;
            constant tPHL:TIME:=O ns);
    port (signal A:in bit;
          signal B:out bit);
end component;

for all : inv use entity work.inv( inv );
signal feedback : bit_vector(31 downto 0);
signal i_o_temp : bit_vector(31 downto 0);

begin

prpt_stage1 : prpt_stage
port map(seed(31),load_seed,reset,feedback(31),clock1,clock1_bar,
clock2,clock2_bar,test_enable,i_o(31));

prpt_stage2 : prpt_stage
port map(seed(30),load_seed,polynomial(31),feedback(30),reset,i_o(30),
clock1,clock1_bar,clock2,clock2_bar,test_enable,feedback(31),i_o(30));

prpt_stage3 : prpt_stage
port map(seed(29),load_seed,polynomial(30),feedback(29),reset,i_o(29),
clock1,clock1_bar,clock2,clock2_bar,test_enable,feedback(30),i_o(29));

prpt_stage4 : prpt_stage
port map(seed(28),load_seed,polynomial(29),feedback(28),reset,i_o(28),
clock1,clock1_bar,clock2,clock2_bar,test_enable,feedback(29),i_o(28));

prpt_stage5 : prpt_stage
port map(seed(27),load_seed,polynomial(28),feedback(27),reset,i_o(27),
clock1,clock1_bar,clock2,clock2_bar,test_enable,feedback(28),i_o(27));

prpt_stage6 : prpt_stage
port map(seed(26),load_seed,polynomial(26),feedback(26),reset,i_o(26),
clock1,clock1_bar,clock2,clock2_bar,test_enable,feedback(25),i_o(26));

prpt_stage7 : prpt_stage
port map(seed(25),load_seed,polynomial(25),feedback(25),reset,i_o(25),
clock1,clock1_bar,clock2,clock2_bar,test_enable,feedback(24),i_o(25));

prpt_stage8 : prpt_stage
port map(seed(24),load_seed,polynomial(24),feedback(24),reset,i_o(24),
clock1,clock1_bar,clock2,clock2_bar,test_enable,feedback(23),i_o(24));

prpt_stage9 : prpt_stage
port map(seed(23),load_seed,polynomial(23),feedback(23),reset,i_o(23),
clock1,clock1_bar,clock2,clock2_bar,test_enable,feedback(22),i_o(23));

prpt_stage10 : prpt_stage
port map(seed(22),load_seed,polynomial(22),feedback(22),reset,i_o(22),
clock1,clock1_bar,clock2,clock2_bar,test_enable,feedback(21),i_o(22));

prpt_stage11 : prpt_stage
port map(seed(21),load_seed,polynomial(21),feedback(21),reset,i_o(21),
clock1,clock1_bar,clock2,clock2_bar,test_enable,feedback(20),i_o(21));

prpt_stage12 : prpt_stage
port map(seed(20),load_seed,polynomial(20),feedback(20),reset,i_o(20),
clock1,clock1_bar,clock2,clock2_bar,test_enable,feedback(19),i_o(20));

prpt_stage13 : prpt_stage
port map(seed(19),load_seed,polynomial(19),feedback(19),reset,i_o(19),
clock1,clock1_bar,clock2,clock2_bar,test_enable,feedback(18),i_o(19));

prpt_stage14 : prpt_stage
port map(seed(18),load_seed,polynomial(18),feedback(18),reset,i_o(18),
clock1,clock1_bar,clock2,clock2_bar,test_enable,feedback(17),i_o(17));
prpt-stage15 : prpt-stage
    port map(seed(16), load_seed, polynomial(17), feedback(16), reset, i_o(17),
              clock1, clock1_bar, clock2, clock2_bar, test_enable, feedback(17), i_o(16));
prpt-stage16 : prpt-stage
    port map(seed(15), load_seed, polynomial(16), feedback(15), reset, i_o(16),
              clock1, clock1_bar, clock2, clock2_bar, test_enable, feedback(16), i_o(16));
prpt-stage17 : prpt-stage
    port map(seed(14), load_seed, polynomial(15), feedback(14), reset, i_o(15),
              clock1, clock1_bar, clock2, clock2_bar, test_enable, feedback(15), i_o(15));
prpt-stage18 : prpt-stage
    port map(seed(13), load_seed, polynomial(14), feedback(13), reset, i_o(14),
              clock1, clock1_bar, clock2, clock2_bar, test_enable, feedback(14), i_o(14));
prpt-stage19 : prpt-stage
    port map(seed(12), load_seed, polynomial(13), feedback(12), reset, i_o(13),
              clock1, clock1_bar, clock2, clock2_bar, test_enable, feedback(13), i_o(12));
prpt-stage20 : prpt-stage
    port map(seed(11), load_seed, polynomial(12), feedback(11), reset, i_o(12),
              clock1, clock1_bar, clock2, clock2_bar, test_enable, feedback(12), i_o(12));
prpt-stage21 : prpt-stage
    port map(seed(10), load_seed, polynomial(11), feedback(10), reset, i_o(11),
              clock1, clock1_bar, clock2, clock2_bar, test_enable, feedback(11), i_o(11));
prpt-stage22 : prpt-stage
    port map(seed(9), load_seed, polynomial(10), feedback(9), reset, i_o(10),
              clock1, clock1_bar, clock2, clock2_bar, test_enable, feedback(10), i_o(10));
prpt-stage23 : prpt-stage
    port map(seed(8), load_seed, polynomial(9), feedback(8), reset, i_o(9),
              clock1, clock1_bar, clock2, clock2_bar, test_enable, feedback(9), i_o(9));
prpt-stage24 : prpt-stage
    port map(seed(7), load_seed, polynomial(8), feedback(7), reset, i_o(8),
              clock1, clock1_bar, clock2, clock2_bar, test_enable, feedback(8), i_o(8));
prpt-stage25 : prpt-stage
    port map(seed(6), load_seed, polynomial(7), feedback(6), reset, i_o(7),
              clock1, clock1_bar, clock2, clock2_bar, test_enable, feedback(7), i_o(7));
prpt-stage26 : prpt-stage
    port map(seed(5), load_seed, polynomial(6), feedback(5), reset, i_o(6),
              clock1, clock1_bar, clock2, clock2_bar, test_enable, feedback(6), i_o(6));
prpt-stage27 : prpt-stage
    port map(seed(4), load_seed, polynomial(5), feedback(4), reset, i_o(5),
              clock1, clock1_bar, clock2, clock2_bar, test_enable, feedback(5), i_o(5));
prpt-stage28 : prpt-stage
    port map(seed(3), load_seed, polynomial(4), feedback(3), reset, i_o(4),
              clock1, clock1_bar, clock2, clock2_bar, test_enable, feedback(4), i_o(4));
prpt-stage29 : prpt-stage
    port map(seed(2), load_seed, polynomial(3), feedback(2), reset, i_o(3),
              clock1, clock1_bar, clock2, clock2_bar, test_enable, feedback(3), i_o(3));
prpt-stage30 : prpt-stage
    port map(seed(1), load_seed, polynomial(2), feedback(1), reset, i_o(2),
              clock1, clock1_bar, clock2, clock2_bar, test_enable, feedback(2), i_o(2));
prpt-stage31 : prpt-stage
    port map(seed(0), load_seed, polynomial(1), feedback(0), reset, i_o(1),
              clock1, clock1_bar, clock2, clock2_bar, test_enable, feedback(1), i_o(1));
and gate1: and_gate port map(polyomial(0),i_o(0),feedback(0));

inv1 : inv port map(i_o(31),temp(31));
inv2 : inv port map(i_o(30),temp(30));
inv3 : inv port map(i_o(29),temp(29));
inv4 : inv port map(i_o(28),temp(28));
inv5 : inv port map(i_o(27),temp(27));
inv6 : inv port map(i_o(26),temp(26));
inv7 : inv port map(i_o(25),temp(25));
inv8 : inv port map(i_o(24),temp(24));
inv9 : inv port map(i_o(23),temp(23));
inv10 : inv port map(i_o(22),temp(22));
inv11 : inv port map(i_o(21),temp(21));
inv12 : inv port map(i_o(20),temp(20));
inv13 : inv port map(i_o(19),temp(19));
inv14 : inv port map(i_o(18),temp(18));
inv15 : inv port map(i_o(17),temp(17));
inv16 : inv port map(i_o(16),temp(16));
inv17 : inv port map(i_o(15),temp(15));
inv18 : inv port map(i_o(14),temp(14));
inv19 : inv port map(i_o(13),temp(13));
inv20 : inv port map(i_o(12),temp(12));
inv21 : inv port map(i_o(11),temp(11));
inv22 : inv port map(i_o(10),temp(10));
inv23 : inv port map(i_o(9),temp(9));
inv24 : inv port map(i_o(8),temp(8));
inv25 : inv port map(i_o(7),temp(7));
inv26 : inv port map(i_o(6),temp(6));
inv27 : inv port map(i_o(5),temp(5));
inv28 : inv port map(i_o(4),temp(4));
inv29 : inv port map(i_o(3),temp(3));
inv30 : inv port map(i_o(2),temp(2));
inv31 : inv port map(i_o(1),temp(1));
inv32 : inv port map(i_o(0),temp(0));
inv33 : inv port map(temp(31),pattern(31));
inv34 : inv port map(temp(30),pattern(30));
inv35 : inv port map(temp(29),pattern(29));
inv36 : inv port map(temp(28),pattern(28));
inv37 : inv port map(temp(27),pattern(27));
inv38 : inv port map(temp(26),pattern(26));
inv39 : inv port map(temp(25),pattern(25));
inv40 : inv port map(temp(24),pattern(24));
inv41 : inv port map(temp(23),pattern(23));
inv42 : inv port map(temp(22),pattern(22));
inv43 : inv port map(temp(21),pattern(21));
inv44 : inv port map(temp(20),pattern(20));
inv45 : inv port map(temp(19),pattern(19));
inv46 : inv port map(temp(18),pattern(18));
inv47 : inv port map(temp(17),pattern(17));
inv48 : inv port map(temp(16),pattern(16));
inv49 : inv port map(temp(16),pattern(15));
inv50 : inv port map(temp(14),pattern(14));
inv51 : inv port map(temp(13),pattern(13));
inv52 : inv port map(temp(12),pattern(12));
inv53 : inv port map(temp(11),pattern(11));
inv54 : inv port map(temp(10),pattern(10));
inv55 : inv port map(temp(9),pattern(9));
inv56 : inv port map(temp(8),pattern(8));
inv57 : inv port map(temp(7),pattern(7));
inv58 : inv port map(temp(6),pattern(6));
inv59 : inv port map(temp(5),pattern(5));
inv60 : inv port map(temp(4),pattern(4));
inv61 : inv port map(temp(3),pattern(3));
inv62 : inv port map(temp(2),pattern(2));
inv63 : inv port map(temp(1),pattern(1));
inv64 : inv port map(temp(0),pattern(0));
end structural;

-- prpt_bch.vhd --

--entity prpt is
-- port(seed : in bit_vector(31 downto 0);
-- load_seed : in bit;
-- polynomial : in bit_vector(31 downto 0);
-- reset : in bit;
-- clock1 : in bit;
-- clock1_bar : in bit;
-- clock2 : in bit;
-- clock2_bar : in bit;
-- test_enable : in bit;
-- pattern : out bit_vector(31 downto 0));
--begin
-- assert ((not clock1 = clock1_bar) or (not clock1'stable(2 ns))) report
-- "bad clock1 result" severity failure;
-- assert ((not clock2 = clock2_bar) or (not clock2'stable(2 ns))) report
-- "bad clock2 result" severity failure;
--end prpt;

architecture beh of prpt is

signal work_pattern : bit_vector(31 downto 0);

begin

process(seed,load_seed,polynomial,reset,clock1,test_enable)

variable temp_pattern : bit_vector(31 downto 0);
variable old_temp_pattern : bit_vector(31 downto 0);
variable feedback : bit_vector(31 downto 0);
begin

if reset = '1' then
    for i in temp_pattern'range loop
        temp_pattern(i) := '0';
    end loop;
else
    if clock1 = '0' and not clock1'stable then
        if load_seed = '1' then
            temp_pattern := seed;
        end if;
    elsif clock1 = '1' and not clock1'stable then
        if test_enable = '1' then
            old_temp_pattern := temp_pattern;
            for i in temp_pattern'low to (temp_pattern'high - 1) loop
                temp_pattern(i) := temp_pattern(i + 1);
            end loop;
            feedback(0) := polynomial(0) and old_temp_pattern(0);
            for i in (feedback'low + 1) to feedback'high loop
                feedback(i) := (polynomial(i) and old_temp_pattern(i)) xor
                feedback(i-1);
            end loop;
            temp_pattern(31) := feedback(31);
        end if;
    end if;
end if;
work_pattern <= temp_pattern;
end process;
process(clock2)
begin
    if clock2 = '1' and not clock2'stable then
        pattern <= work_pattern;
    end if;
end process;
end beh;

-- prpt_gen.vhd --

--entity prpt is
-- port(seed : in bit_vector(31 downto 0);
-- load_seed : in bit;
-- polynomial : in bit_vector(31 downto 0);
-- reset : in bit;
-- clock1 : in bit;
-- clock1_bar : in bit;
-- clock2 : in bit;
-- clock2_bar : in bit;
-- test_enable : in bit;
-- pattern : out bit_vector(31 downto 0));
--begin
-- assert (((not clock1 = clock1_bar) or (not clock1'stable(2 ns)))
-- or (now = 0 ns)) report
-- "bad clock1 result" severity failure;
---- assert (((not clock2 = clock2_bar) or (not clock2'stable(2 ns))) report
---- "bad clock2 result" severity failure;
--end prpt;

architecture struct_generate of prpt is

component prpt_stage
    port(seed : in bit;
         load_seed : in bit;
         polynomial : in bit;
         feed_in : in bit;
         reset : in bit;
         input : in bit;
         clock1 : in bit;
         clock1_bar : in bit;
         clock2 : in bit;
         clock2_bar : in bit;
         test_enable: in bit;
         feed_out : out bit;
         output : out bit);
end component;

component prpt_stagel
    port(seed : in bit;
         load_seed : in bit;
         reset : in bit;
         input : in bit;
         clock1 : in bit;
         clock1_bar : in bit;
         clock2 : in bit;
         clock2_bar : in bit;
         test_enable: in bit;
         output : out bit);
end component;

for all : prpt_stagel use entity work.prpt_stagel(structural);

component and_gate
    port (signal X,Y:in bit;
         signal Output:out bit);
end component;

for all : and_gate use entity work.and_gate( and_gate );

component inv
    generic(constant tPLH:TIME:=0 ns;
             constant tPHL:TIME:=0 ns);

15
port (signal A: in bit;
    signal B: out bit);
end component;

signal feedback : bit_vector(31 downto 0);
signal i_o, temp : bit_vector(31 downto 0);

begin

prpt_stage1 : prpt_stage1
    port map(seed(31), load_seed, reset, feedback(31), clock1, clock1_bar,
    clock2, clock2_bar, test_enable, i_o(31));

prpt : for i in 31 downto 1 generate
    prpt_stage_i : prpt_stage
        port map(seed(i-1), load_seed, polynomial(i), feedback(i-1),
        reset, i_o(i), clock1, clock1_bar, clock2, clock2_bar,
        test_enable, feedback(i), i_o(i-1));
    end generate;

and_gate1 : and_gate port map(polynomial(0), i_o(0), feedback(0));

inv_inv : for i in 0 to 31 generate
    invA : inv port map(i_o(i), temp(i));
    invB : inv port map(temp(i), pattern(i));
    end generate;

end struct_generate;

-- master.vhd --

entity master_clock is
    generic
        ( constant clock_low_time : time := 60 ns;
          constant duty_cycle_time : time := 40 ns);
    port
        ( master_go : in bit;
          Phi : buffer bit := '1');
end master_clock;

architecture master_clock of master_clock is

begin

process(master_go, phi)
begin
    if (master_go = '1') then
        case Phi is
            when '1' => Phi <= '0' after duty_cycle_time;
        end case;
when '0' => Phil <= '1' after clock_low_time;
when others => assert false
    report "Master clock in undefined state"
    severity ERROR;
end case;
end if;
end process;
end master_clock;

-- slave.vhd --

entity slave_clock is
generic
  ( constant pulse_width : time := 40 ns;
    constant pulse_delay : time := 10 ns);
port
  ( external_trigger : in bit;
    Clock_out : out bit := '0');

-- This is a leading edge triggered clock!
end slave_clock;

architecture slave_clock of slave_clock is
begin
  clock:block ((external_trigger = '0') and not(external_trigger'stable))

  begin
    process (guard)
    begin
      if GUARD then
        Clock_out <= '1' after pulse_delay,
        '0' after pulse_width+pulse_delay;
      end if;
    end process;
  end block;
end slave_clock;

-- prptconf.vhd --

configuration prptconf of work.prpt is
  for struct_generate
  for prpt
The file std.vhd contains a group of basic gates: two types of inverter, a two-input nand gate, a three-input nand gate, a two-input exclusive or gate, a three-input exclusive or gate, a nor gate, a p-type transistor, an n-type transistor, and a transmission gate. and_gate.vhd builds a two-input and gate from an inverter and a two-input nand gate. or_gate.vhd builds a two-input or gate from an inverter and a nor gate. diff_reset_beh.vhd is a behavioral description of a D latch with reset.

prpt_stage1.vhd contains the component representing the MSB of the linear feedback shift register. As shown in Figure 1, the component consists of two D latches connected to form a dual-clocked latch. The input to the second latch is the output from the first latch, clocked in as clock2 goes high. The input to the first latch is dependent on whether or not the signals load_seed and test_enable are set. If load_seed is set, then seed gets latched in after one clock cycle.

If test_enable is set, input gets latched in after one clock cycle. If reset is set, the first latch is set to '0' immediately, and the second is set to '0' as clock2 goes high. To ensure proper behavior of this component, one must either not set load_seed and test_enable at the same time, or else reset the latch immediately before loading a seed.

Figure 1. Schematic of prpt_stage1.

Shown in Figure 2 is prpt_stage.vhd. prpt_stage.vhd contains the component representing the 31 LSBs of the shift register. The component consists of the prpt_stage1 component, plus two gates to control the feedback feature of the register. If polynomial is set, then input and feed_in will be exclusive or'd to produce feed_out. If polynomial is not set, feed_out receives the value of feed_in.
prpt.vhd contains the structural description of a 32-bit linear feedback shift register. It consists of one prpt.stage1 component, 31 prpt.stage components, two invertors per stage, and an AND gate. The invertors are needed to produce the pattern output. The AND gate is required to produce the last stage's feedback signal. Figure 2 is a schematic of the arrangement of the components. Since the output from each stage is also used as input to the next stage, pattern (mode out), the output from each stage, could not be produced directly from the stage components. An internally declared signal, Lo, is used in the stage component instantiations, and then passed through a two-invertor buffer to produce pattern.

prpt.beh.vhd contains the behavioral description of the 32-bit linear feedback shift register. It consists of two processes. The first is triggered whenever an event occurs on seed, load_seed, polynomial, reset, clock1, or test_enable. This process controls resetting, loading new seed values, and shifting the register bit values. The second process is triggered upon an event on clock2. It controls assigning the pattern value calculated in the first process to pattern at the appropriate time, when clock2 goes high. The first attempt at a behavioral description used a binary division scheme [3] to generate the bits to shift into the register. The seed was the dividend and the polynomial was the divisor, and as each bit of the quotient was calculated, it was shifted into the left side of the pattern. The quotient bits are calculated as follows. If the remainder (or dividend, if this is the first step of the division) has an MSB of '1', then the quotient bit is '1'. Then, the polynomial is copied underneath the remainder (or dividend), the two bit vectors are exclusive or'd, and the result (remainder) is written on the next line. If the remainder (or dividend) has an MSB of '0', then the quotient bit is '0'. Then, "00000" is written underneath the remainder, the polynomial and "00000" are exclusive or'd and the result is written (or the polynomial is simply copied) on the next line. No matter what the quotient bit is, the leading '0' of the remainder is discarded, and the next bit of the dividend is brought down and written as the LSB of the remainder. The goal of an n-bit linear feedback shift register is to produce \((2^n - 1)\) unique bit patterns (each pattern except all zeroes) before repeating the set of patterns. For example, the binary division using a seed of "1000" and a polynomial of "10011" (four-bit shift register, shown in Figure 4 [3]) would be as follows:
Figure 3. Schematic of prpt.

Figure 4. Schematic of 4-Stage prpt.
The example above works out correctly, since no patterns are repeated until all 15 \((2^n - 1)\) patterns are produced. However, the following example shows that this binary division algorithm does not work for all seeds.
The pattern produced after the third quotient bit is calculated is hex(A), the same value as the seed.

prpt\_gen.vhd contains an alternate structural description of the register. It is identical to prpt.vhd except that the 31 LSB stages and the 64 invertors are instantiated with `generate` statements.

master.vhd and `slave.vhd` comprise a leading-edge triggered dual clock. The master clock stays high for 60 nanoseconds, transitions to low, and stays low for 40 nanoseconds. The output value from the master clock is the input for the slave clock. The slave clock goes high 10 nanoseconds after the master clock goes low, and stays high for 40 nanoseconds. It then transitions to low, and remains low for 60 nanoseconds. The clocks stop cycling when the master clock is given an input of ‘0’.

prptconf.vhd is a configuration specification needed when prpt\_gen.vhd is used. The `generate` statements make it necessary to configure those components outside of the architecture in which they are instantiated.
II. WAVES Support Source Code

--- waves_header.vhd ---

-- This WAVES dataset uses MATCH to compare the output of the structural
-- LFSR to the output of the behavioral model.

TITLE Pseudo-Random Pattern Generator
CORPORATE_AUTHOR WL/ELED
INDIVIDUAL_AUTHOR Captain Karen M. Serafino
RELEASE_DATE_AND_TIME 19-March-1991
ORIGIN Intermetrics, Inc. Standard VHDL 1076 Support Environment Version 2.1
DEVICE_ID 32-bit linear feedback shift register

WAVES_FILENAME waves_logic WORK
WAVES_UNIT WAVES_INTERFACE WORK
WAVES_FILENAME waves_codes WORK
WAVES_FILENAME waves_device WORK
WAVES_UNIT WAVES_OBJECTS WORK
WAVES_FILENAME waves_form WORK

EXTERNAL_FILENAME waves_external waves_external

WAVEFORM_GENERATOR.PROCEDURE WORK.WAVES_GENERATOR.WAVEFORM

--- waves_logic.vhd ---

use work.waves.standard.all;
package waves_logic is
  type logic_value is (logic_i0, logic_i1, logic_o0, logic_o1);
  function value_dictionary(value : logic_value) return event_value;
end waves_logic;

package body waves_logic is
  function value_dictionary(value : logic_value) return event_value is
    begin
      case value is
        when logic_i0 => return (state = low and strength = drive and
direction = stimulus and relevance = required);
        when logic_i1 => return (state = high and strength = drive and
direction = stimulus and relevance = required);
        when logic_o0 => return (state = low and strength = drive and
direction = response and relevance = required);
        when logic_o1 => return (state = high and strength = drive and
direction = response and relevance = required);
      end case;
    end value_dictionary;
end waves_logic;
end waves_logic;

-- waves_codes.vhd --

use work.waves_logic.all;
use work.waves_interface.all;
package waves_codes is
  constant pin_codes : string := "01";
end waves_codes;

-- waves_device.vhd --

package waves_device is
  type test_pins is (st, te, ls, re,
    y31, y30, y29, y28, y27, y26, y25, y24, y23, y22, y21, y20, y19, y18, y17, y16,
    y15, y14, y13, y12, y11, y10, y9, y8, y7, y6, y5, y4, y3, y2, y1, y0,
    s31, s30, s29, s28, s27, s26, s25, s24, s23, s22, s21, s20, s19, s18, s17, s16,
    s15, s14, s13, s12, s11, s10, s9, s8, s7, s6, s5, s4, s3, s2, s1, s0,
    p15, p14, p13, p12, p11, p10, p9, p8, p7, p6, p5, p4, p3, p2, p1, p0);
end waves_device;

-- waves_form.vhd --

use std.textio.all;
use work.waves_logic.all;
use work.waves_interface.all;
use work.waves_codes.all;
use work.waves_device.all;
use work.waves_objects.all;
package waves_generator is
  procedure waveform(signal connect : inout waves_port_list;
    signal connect_m : in waves_match_list);
end waves_generator;
package body waves_generator is
  procedure waveform(signal connect : inout waves_port_list;
    signal connect_m : in waves_match_list) is
    constant t0 : event_time := etime(0 ns);
    constant input_pins : pinset := new_pinset((
      st, te, ls, re,
      y31, y30, y29, y28, y27, y26, y25, y24, y23, y22, y21, y20, y19, y18, y17, y16,
      y15, y14, y13, y12, y11, y10, y9, y8, y7, y6, y5, y4, y3, y2, y1, y0,
      s31, s30, s29, s28, s27, s26, s25, s24, s23, s22, s21, s20, s19, s18, s17, s16,
      s15, s14, s13, s12, s11, s10, s9, s8, s7, s6, s5, s4, s3, s2, s1, s0));
    constant output_pins : pinset := all_pins and not input_pins;
constant input_fs : frame_set :=
  frame_event((logic_i0,t0)) +
  frame_event((logic_i1,t0));
constant output_fs : frame_set :=
  frame_event((logic_o0,t0)) +
  frame_event((logic_o1,t0));
constant fsa : frame_set_array :=
  new_frame_set_array(input_fs,input_pins) +
  new_frame_set_array(output_fs,output_pins);
variable td : time_data := new_time_data(fsa);
variable slice : file_slice := new_file_slice;
file lfsr_data : text is in "waves_external";
constant match_pins : pinset := all_pins;
begin
  match(connect, hold, input_pins);
  match(connect, sample, output_pins);
  loop
    read_file_slice(lfsr_data,slice);
    exit when slice.end_of_file;
    apply(connect,slice.codes.all_delay(slice.fs_time),td);
    assert (match(connect_m,match_pins)) report "no match" severity failure;
  end loop;
end waveform;
end waves_generator;

waves_external. —

waves_header.vhd is the WAVES header file that documents the order of analysis of the packages, the name of the waveform generator procedure, and the names of any external files.

waves_logic.vhd contains the package that defines the WAVES-required type logic_value and the function value_dictionary. There are four logic values in this model, '0' and '1' bit values for both the input and output pins. value_dictionary takes a parameter of type logic_value and returns information about the value's state, strength, direction, and relevance. State and strength are either high and drive (for '1' values) or low and drive (for '0' values). Direction is stimulus
for input pins and response for output pins. Relevance is required for all pins here. Relevance is used by the test bench program to determine the pins for which to check the actual values against the predicted values.

waves_codes.vhd contains the package that defines the WAVES-required constant pin_codes. In this model, only bit values '0' and '1' are used, so pin_codes is the string "01".

waves_device.vhd contains the package that defines the WAVES-required enumerated type test_pins. test_pins consists of one unique name for each input and output pin of the shift register. The order of values in test vector waveforms must match the order of pin names in test_pins. The pin names are abbreviated forms of stop, test_enable, load_seed, reset, polynomial(31) downto polynomial(0), seed(31) downto seed(0), and pattern(31) downto pattern(0), respectively.

waves_form.vhd contains the package that defines the WAVES-required waveform generator procedure. There is no specific name required; this procedure is named waveform. The procedure supplies waveforms for the shift register by reading data from an external file waves_external. Each record (line) of the external file supplies a value for each test pin at each given simulation time. First, a time_data variable (called td here) must be built. Several intermediate constants are declared to avoid having one huge declaration. td is formed from a frame set array fsa. A frame set array is like a template, with each pin having a slot for each logic_value it can have, at time 0 nanoseconds. The two match procedure calls are for setting the m_control value for each pin. Input pins get a value of hold, meaning no checking of actual versus predicted pin values is done by the function match. Output pins get a value of sample, meaning checking of actual versus predicted values will be done on these pins by function match. The loop in the procedure reads a record from the external data file, applies the given bit values for each specified pin, and waits for the amount of time specified by the delay time value in each record. There is also a call to the function match in the loop. After each waveform is applied, and simulation time is advanced appropriately according to the given delay time, each pin with an m_control value of sample gets its actual value checked against the value predicted for it in the external data file. If there is a discrepancy, the assertion statement will cause simulation to stop. This continues until the end-of-file of waves_external is reached.
III. Test Bench Programs

- test_prpt.vhd This test bench does not use WAVES —

entity test is end;
architecture test of test is

component prpt
port(seed : in bit_vector(31 downto 0);
load_seed : in bit;
polynomial : in bit_vector(31 downto 0);
reset : in bit;
clock1 : in bit;
clock1_bar : in bit;
clock2 : in bit;
clock2_bar : in bit;
test_enable: in bit;
pattern : out bit_vector(31 downto 0));
end component;

for prpt1 : prpt use entity work.prpt(structural);
for prpt2 : prpt use entity work.prpt(beh);
for prpt3 : prpt use configuration work.prptconf;

component inv
generic(constant tPLH:TIME:=0 ns;
constant tPHL:TIME:=0 ns);
port (signal A:in bit;
      signal B:out bit);
end component;

for all : inv use entity work.inv( inv );

component master_clock
generic
 ( constant clock_low_time : time := 60 ns;
   constant duty_cycle_time : time := 40 ns);
port
 ( master_go : in bit;
   Phi1 : buffer bit := '1');
end component;

for all : master_clock use entity work.master_clock(master_clock );

component slave_clock
generic
 ( constant pulse_width : time := 40 ns;
   constant pulse_delay : time := 10 ns);
port
 ( external_trigger : in bit;
   Clock_out : out bit := '0');

27
end component;

for all : slave_clock use entity work.slave_clock(slave_clock);

component or_gate
  port (X, Y : in bit;
        Output : out bit);
end component;

for all : or_gate use entity work.or_gate(or_gate);

component and_gate
  port (signal X,Y:in bit;
        signal Output:out bit);
end component;

for all : and_gate use entity work.and_gate( and_gate );

signal pattern : bit_vector(95 downto 0);
signal polynomial : bit_vector(31 downto 0) :=
"00110000000000000000000000000000"
signal seed : bit_vector(31 downto 0) :=
"01010000000000000000000000000000"
signal load_seed : bit := '0';
signal reset : bit := '1';
signal clock1 : bit := '1';
signal clock1_bar : bit := '0';
signal clock2 : bit := '0';
signal clock2_bar : bit := '1';
signal CP : bit := '1';
signal stop : bit := '0';
alias pat1 : bit_vector(31 downto 0) is pattern(95 downto 64);
alias pat2 : bit_vector(31 downto 0) is pattern(63 downto 32);
alias pat3 : bit_vector(31 downto 0) is pattern(31 downto 0);
signal interm1,interm2 : bit;
signal test_enable : bit := '1';

begin

inv1 : inv port map(clock1,clock1_bar);
inv2 : inv port map(clock2,clock2_bar);
prpt1 : prpt
  port map(seed,load_seed,polynomial,reset,clock1,clock1_bar,
           clock2,clock2_bar,test_enable,pat1);
prpt2 : prpt
  port map(seed,load_seed,polynomial,reset,clock1,clock1_bar,
           clock2,clock2_bar,test_enable,pat2);
prpt3 : prpt
  port map(seed,load_seed,polynomial,reset,clock1,clock1_bar,
           clock2,clock2_bar,test_enable,pat3);

28
master_clock1 : master_clock port map(stop,CP);
slave_clock1 : slave_clock port map(CP,clock2);
and_gate1 : and_gate port map(CP,test_enable,interm1);
and_gate2 : and_gate port map(load_seed,CP,interm2);
or_gate1 : or_gate port map(interm1,interm2,clock1);

load_seed <= '1' after 50 ns,
   '0' after 150 ns,
   '1' after 2400 ns,
   '0' after 2495 ns,
   '1' after 3400 ns,
   '0' after 3495 ns,
   '1' after 4400 ns,
   '0' after 4495 ns,
   '1' after 5400 ns,
   '0' after 5495 ns;

reset <= '0' after 30 ns,
   '1' after 2200 ns,
   '0' after 2390 ns,
   '1' after 3200 ns,
   '0' after 3390 ns,
   '1' after 4200 ns,
   '0' after 4390 ns,
   '1' after 5200 ns,
   '0' after 5390 ns;

stop <= '1' after 100 ns,
   '0' after 7000 ns;

test_enable <= '1',
   '0' after 2400 ns,
   '1' after 4395 ns;

seed <= "101000000000000000000000000000000" after 2400 ns,
    "1111000000000000000000000000000000" after 3400 ns,
    "0101000000000000000000000000000000" after 4400 ns,
    "0011000000000000000000000000000000" after 5400 ns;

process
begin
   variable wait_var : boolean := true;
   if wait_var then
      wait for 40 ns;
      wait_var := false;
      assert (not pat1'stable) or (not pat2'stable) or (not pat3'stable) or
      ((pat1 = pat2) and (pat1 = pat3)) report "patterns do not match"
         severity failure;
      wait for 55 ns;
   else

   end process;
wait for 100 ns;
assert (not pat1'stable) or (not pat2'stable) or (not pat3'stable) or
((pat1 = pat2) and (pat1 = pat3)) report "patterns do not match"
severity failure;
assert (stop = '1') report "done" severity failure;
end if;
end process;
end test;

-- test_bench.vhd  WAVES test bench --

use std.textio.all;
use work.waves_system.all;
use work.waves_logic.all;
use work.waves_interface.all;
use work.waves_device.all;
use work.waves_objects.all;
use work.waves_generator.all;
entity test is end;
architecture test of test is

component prpt
port(seed : in bit_vector(31 downto 0);
    load_seed : in bit;
    polynomial : in bit_vector(31 downto 0);
    reset : in bit;
    clock1 : in bit;
    clock1_bar : in bit;
    clock2 : in bit;
    clock2_bar : in bit;
    test_enable : in bit;
    pattern : out bit_vector(31 downto 0)));
end component;

for prpt1 : prpt use entity work.prpt(structural);
for prpt2 : prpt use entity work.prpt(beh);
for prpt3 : prpt use configuration work.prptconf;

component inv
generic(constant tPLH:TIME:=0 ns;
        constant tPHL:TIME:=0 ns);
port (signal A:in bit;
      signal B:out bit);
end component;

for all : inv use entity work.inv( inv );

component master_clock
generic
  ( constant clock_low_time : time := 60 ns;
    constant duty_cycle_time : time := 40 ns);
  port
    ( master_go : in bit;
      Phi : buffer bit := '1');
end component;

for all : master_clock use entity work.master_clock(master_clock);

component slave_clock
  generic
    ( constant pulse_width : time := 40 ns;
      constant pulse_delay : time := 10 ns);
  port
    ( external_trigger : in bit;
      Clock_out : out bit := '0');
end component;

for all : slave_clock use entity work.slave_clock(slave_clock);

component or_gate
  port (X, Y : in bit;
        Output : out bit);
end component;

for all : or_gate use entity work.or_gate(or_gate);

component and_gate
  port (signal X,Y:in bit;
        signal Output:out bit);
end component;

for all : and_gate use entity work.and_gate( and_gate );

function data_to_char(data : logic_value) return character is
begin
  case data is
    when logic_i0 => return '0';
    when logic_i1 => return '1';
    when logic_o0 => return '0';
    when logic_o1 => return '1';
  end case;
end data_to_char;

function data_to_bit(data : logic_value) return bit is
begin
  case data is
    when logic_i0 => return '0';
    when logic_i1 => return '1';
    when logic_o0 => return '0';
    when logic_o1 => return '1';
end case;

end case;
end data_to_bit;
subtype test_pins_range is integer range
test_pins'pos(test_pins'left) + 1 to
    test_pins'pos(test_pins'right) + 1;
signal test_vector : bit_vector(test_pins_range);
alias pat1 : bit_vector(31 downto 0) is test_vector(69 to 100);
alias polynomial : bit_vector(31 downto 0) is test_vector(5 to 36);
alias seed : bit_vector(31 downto 0) is test_vector(37 to 68);
alias load_seed : bit is test_vector(3);
alias reset : bit is test_vector(4);
signal clock1 : bit;
signal clock1_bar : bit;
signal clock2 : bit;
signal clock2_bar : bit;
signal CP : bit;
alias stop : bit is test_vector(1);
signal pat2 : bit_vector(31 downto 0);
signal pat3 : bit_vector(31 downto 0);
signal interm1, interm2 : bit;
alias test_enable : bit is test_vector(2);
signal connect : waves_port_list(test_pins_range);
signal connect_m : waves_match_list(test_pins_range);
begin
waves : waveform(connect, connect_m);
read : process(connect)
begin
    stop <= transport data_to_bit(logic_value'val(connect(1).l_value));
    test_enable <= transport data_to_bit(logic_value'val(connect(2).l_value));
    load_seed <= transport data_to_bit(logic_value'val(connect(3).l_value));
    reset <= transport data_to_bit(logic_value'val(connect(4).l_value));
    for i in polynomial'high downto polynomial'low loop
        polynomial(i) <= transport data_to_bit(logic_value'val(connect(36-i).l_value));
    end loop;
    for i in seed'high downto seed'low loop
        seed(i) <= transport data_to_bit(logic_value'val(connect(68-i).l_value));
    end loop;
end process;
inv1 : inv port map(clock1,clock1_bar);
inv2 : inv port map(clock2,clock2_bar);
prpt1 : prpt
    port map(seed,load_seed,polynomial,reset,clock1,clock1_bar,
        clock2,clock2_bar,test_enable,pat1);
prpt2 : prpt
    port map(seed,load_seed,polynomial,reset,clock1,clock1_bar,
clock2,clock2_bar,test_enable,pat2); 

prpt3 : prpt  
  port map(seed,load_seed,polynomial,reset,clock1,clock1_bar,  
clock2,clock2_bar,test_enable,pat3); 

master_clock1 : master_clock port map(stop,CP); 
slave_clock1 : slave_clock port map(CP,clock2); 
and_gate1 : and_gate port map(CP,test_enable,intern1); 
and_gate2 : and_gate port map(load_seed,CP,intern2); 
or_gate1 : or_gate port map(intern1,intern2,clock1); 

match_proc : process  
  variable first_time : boolean := true;  
  type ev_ptr is access system_event_value;  
  variable ev : ev_ptr;  
  variable rel_flag,dir_flag : boolean;  
--variable outline : line; -- for debugging  
--file大纲 : text is out "match.out";  
begin  
  if first_time then  
    for j in test_pin_range loop  
      connect_m(j).m_value <= true;  
    end loop;  
    first_time := false;  
  end if;  
  wait until patl'stable;  
  for i in test_pin_range loop  
    ev :=  
      new system_event_value'(value_dictionary(logic_value'val(connect(i).l_value))));  
    dir_flag := false;  
    rel_flag := false;  
    for k in ev.all'range loop  
      if ev(k).kind = direction_kind then  
        if ev(k).value = direction_type'pos(response) then  
          dir_flag := true;  
        end if;  
      elsif ev(k).kind = relevance_kind then  
        if ev(k).value = relevance_type'pos(required) then  
          rel_flag := true;  
        end if;  
      end if;  
    end loop;  
    if (connect(i).m_control = sample) and dir_flag and rel_flag then  
      if (test_vector(i) = pat2(100-i)) and (test_vector(i) = pat3(100-i)) then  
        connect_m(i).m_value <= connect_m(i).m_value and true;  
      else  
        connect_m(i).m_value <= connect_m(i).m_value and false;  
      end if;  
    end if;  
  end loop; 
end match_proc;
end if;
end if;
end loop;
end process;

view : process(test_enable,load_seed,stop,reset,polynomial,seed)
  variable outline : line;
  file outfile : text is out "waves.out";
begin
  write(outline,now);
  writeln(outfile,outline);
  write(outline,string("stop : "));
  write(outfile,stop);
  writeln(outfile,outline);
  write(outline,string("test_enable : "));
  write(outfile,test_enable);
  writeln(outfile,outline);
  write(outline,string("load_seed : "));
  write(outfile,load_seed);
  writeln(outfile,outline);
  write(outline,string("reset : "));
  write(outfile,reset);
  writeln(outfile,outline);
  write(outfile,string("polynomial : "));
  for i in polynomial'high downto polynomial'low loop
    write(outfile,polynomial(i));
  end loop;
  writeln(outfile,outline);
  write(outfile,string("seed : "));
  for i in seed'high downto seed'low loop
    write(outfile,seed(i));
  end loop;
  writeln(outfile,outline);
  write(outfile,string("pattern : "));
  for i in pat1'high downto pat1'low loop
    write(outfile,pat1(i));
  end loop;
  writeln(outfile,outline);
end process;

lfsr_out : process(pat1)
  variable outline : line;
  file out_lfsr : text is out "lfsr.out";
begin
  write(outline,stop);
  write(outline,test_enable);
  write(outline,load_seed);
  write(outline,reset);
  for i in polynomial'high downto polynomial'low loop
    write(outline,polynomial(i));
  end loop;
for i in seed'high downto seed'low loop
    write(outline,seed(i));
end loop;
write(outline,string'(" : "));
write(outline,now);
writeline(out-lfsr,outline);
for i in patl'high downto patl'low loop
    write(outline,patl(i));
end loop;
writeline(out-lfsr,outline);
end process;
end test;

File test_prpt.vhd is a pattern generator test bench that does not use a WAVES dataset. It generates test vectors identical to those built by the waveform generator procedure in chapter 2. This test bench uses three different architectures for the entity prpt, a structural one, a behavioral one, and a structural model that uses generate statements to instantiate the prpt-stage and inv components. It compares the outputs from the three prpt components against each other. If there is a discrepancy, simulation is stopped with an assertion statement. The test bench also contains the dual clock and its associated gates.

test_bench.vhd is the test bench that uses a WAVES dataset. It contains the same components as test_prpt.vhd, along with a concurrent call to the waveform generator procedure, a process to set the match values for the output pins, a process that feeds waveform information to internal signals, and processes that write input and output data to external files. It also has functions that convert logic_value objects to characters and bits. It does not manually generate test vectors, but uses the waveform generator procedure to build them from the external file.

These two test benches produce the same input waveforms and output vectors.
IV. Command File

--- Intermetrics, Inc. command file ---

vls define waves_standard work
vhdl std.vhd
vhdl and_gate.vhd
vhdl or_gate.vhd
vhdl dff_reset_beh.vhd
vhdl prpt_stage1.vhd
vhdl prpt_stage.vhd
vhdl prpt.vhd
vhdl prpt_beh.vhd
vhdl prpt_gen.vhd
vhdl master.vhd
vhdl slave.vhd
vhdl prptconf.vhd
mag inv"(inv)"
mg invz"(invz)"
mg nand_gate"(nand_gate)"
mg nand3_gate"(nand3_gate)"
mg xor_gate"(xor_gate)"
mg xor_gate2"(xor_gate2)"
mg nor_gate"(nor_gate)"
mg ptrans"(ptrans)"
mg ntrans"(ntrans)"
mg tgate"(tgate)"
mg and_gate"(and_gate)"
mg or_gate"(or_gate)"
mg dff_reset"(beh)"
mg prpt_stage1"(structural)"
mg prpt_stage"(structural)"
mg prpt"(structural)"
mg prpt"(beh)"
mg prpt"(struct_generate)"
mag master_clock"(master_cl .)"
mag slave_clock"(slave_clock"
mag prptconf
vhdl waves_system.wav
vhdl waves_standard.wav
vhdl waves_standard.wav
vhdl waves_logic.vhd
vhdl waves_interface.wav
vhdl waves_interface.wav
vhdl waves_codes.vhd
vhdl waves_device.vhd
vhdl waves_objects.wav
vhdl waves_objects.wav
vhdl waves_form.vhd
vhdl test_bench.vhd  # use test_prpt.vhd for non-WAVES test bench
This is a command file for analyzing, model generating, building, and simulating the pseudo-random pattern generator using the Intermetrics, Inc. Standard VHDL 1076 Support Environment. First, the low-level components comprising the pattern generator are analyzed and model generated. Next, the WAVES packages are analyzed and model generated. Finally, the test bench is analyzed, model generated, built and simulated. The general way to execute the file is:

```
% csh -v command_file_name >& log_file_name &
```
V. Conclusions

The VHDL models were simulated with and without WAVES. The results from both were found to be equivalent. Since the WAVES manual was incomplete and contained errors, there was difficulty learning and implementing the standard. Recommended corrections and additions have been submitted. The integrity of the existing WAVES software was found to be intact. Sufficient WAVES software existed for completing this project.
VI. References

