Asynchronous
Circuit Design

Asynchronous Circuit Design

Chris J. Myers

A Wiley-Interscience Publication
JOHN WILEY & SONS, INC.

New York / Chichester / Weinheim / Brisbane / Singapore / Toronto

1

Introduction

Problems

1.1 Describe in words the behavior of the channel-level process given below.

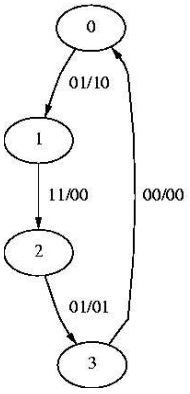
```
Shop:process
begin
  receive(WineryShop,bottle1);
  receive(WineryShop,bottle2);
  send(ShopPatron,bottle2);
  send(ShopPatron,bottle1);
end process;
```

The shelf can now hold two bottles. The last received is the first sold.

1.2 In this problem we design a Huffman circuit for the shop when the winery is passive while the patron is active.

1.2.1. Write a process for a four-phase active/passive shop.

```
Shop_AP: process
begin
  assign(req_wine,'1');   -- call winery
  guard(ack_wine,'1');    -- wine arrives
  assign(req_wine,'0');   -- reset req_wine
  guard(ack_wine,'0');    -- ack_wine resets
  guard(req_patron,'1');  -- patron calls
  assign(ack_patron,'1'); -- sells wine
  guard(req_patron,'0');  -- req_patron resets
  assign(ack_patron,'0'); -- reset ack_patron
end process;
```

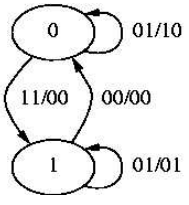


(a)

	00	01	11	10
0	⓪, 00	1, −0	–	–
1	–	①, 10	2, −0	–
2	–	3, 0–	②, 00	–
3	0, 0–	③, 01	–	–

(b)

Fig. 1.1 (a)AFSM and (b)Huffman flow table for active/passive shop.



(a)

	00	01	11	10
0	⓪, 00	⓪, 10	1, −0	–
1	0, 0–	①, 01	①, 00	–

(b)

Fig. 1.2 Reduced (a)AFSM and (b)Huffman flow table for the active/passive shop.

1.2.2. A reshuffled version of an active/passive shop is shown below. Give an AFSM and Huffman flow table that describe its behavior.

```
Shop_AP_reshuffled:process
begin
  guard(req_patron,'1');   -- patron calls
  assign(req_wine,'1');    -- call winery
  guard(ack_wine,'1');     -- wine arrives
  assign(req_wine,'0');    -- reset req_wine
  guard(ack_wine,'0');     -- ack_wine resets
  assign(ack_patron,'1'); -- sells wine
  guard(req_patron,'0');   -- req_patron resets
  assign(ack_patron,'0'); -- reset ack_patron
end process;
```

1.2.3. Combine compatible rows and make a state assignment.

1.2.4. Use K-maps to find a Huffman circuit implementation.

ack_wine/req_patron

x	00	01	11	10
0	0	1	-	–
1	0	0	0	–

req_wine

ack_wine/req_patron

x	00	01	11	10
0	0	0	0	–
1	-	1	0	–

ack_patron

ack_wine/req_patron

x	00	01	11	10
0	0	0	1	–
1	0	1	1	–

x

Fig. 1.3 K-maps for the active/passive shop.

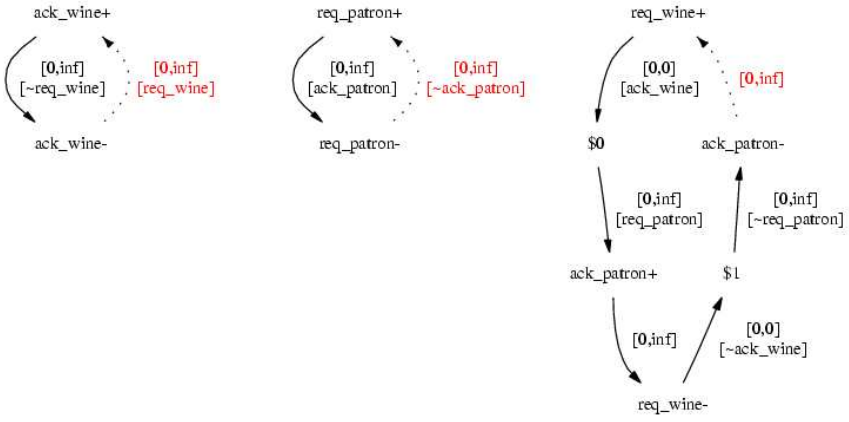


Fig. 1.4 TEL structure for a active/passive shop.

1.3 In this problem we design a Muller circuit for the shop when the winery is passive while the patron is active.

1.3.1. Draw a TEL structure for the reshuffled active/passive shop below.

```
Shop_AP_reshuffled:process
begin
  assign(req_wine,'1');   -- call winery
  guard(ack_wine,'1');    -- wine arrives
  guard(req_patron,'1');  -- patron calls
  assign(ack_patron,'1'); -- sells wine
  assign(req_wine,'0');   -- reset req_wine
  guard(ack_wine,'0');    -- ack_wine resets
  guard(req_patron,'0');  -- req_patron resets
  assign(ack_patron,'0'); -- reset ack_patron
end process;
```

1.3.2. Find the state graph from the TEL structure.

1.3.3. Use K-maps to derive a Muller circuit implementation.

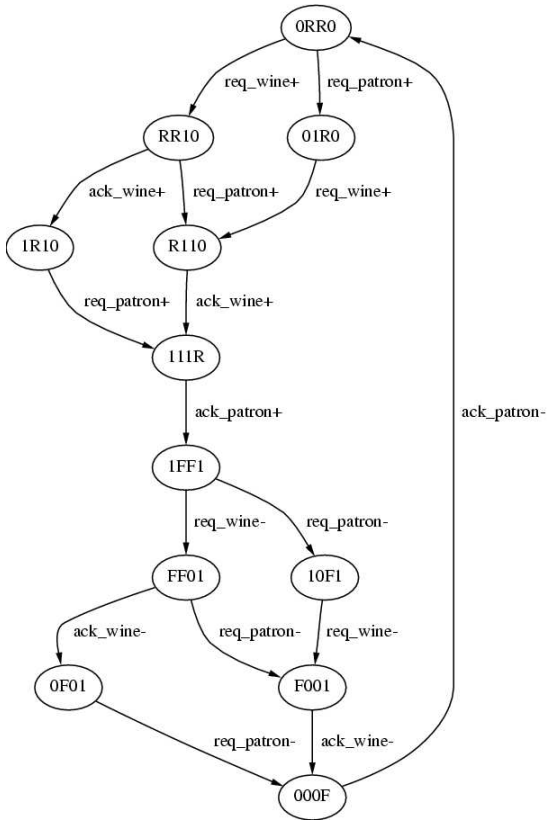


Fig. 1.5 State graph for a active/passive shop (with state vector $\langle$ *ack_wine*, *req_patron*, *req_wine*, *ack_patron* $\rangle$).

req_wine/ack_patron \ *ack_wine/req_patron*	00	01	11	10
00	1	1	–	–
01	0	0	0	0
11	–	–	0	0
10	1	1	1	1

req_wine

ack_wine/req_patron	00	01	11	10
00	0	0	–	–
01	0	1	1	1
11	–	–	1	1
10	0	0	1	0

ack_patron

Fig. 1.6 K-maps for active/passive shop.

2

Communication Channels

Problems

2.1 In this problem you will specify a 4-bit adder using the channel model. The adder has three input ports and two output ports. The input ports are X and Y, which are used to pass in the 4-bit integer operands, and *Cin*, which is a 1-bit carry-in. The output ports are *Sum*, which is a 4-bit integer sum, and *Cout*, which is a 1-bit carry-out.

2.1.1. Specify the 4-bit adder in VHDL using the channel model. The *adder4* process should accept the two operands a and b and the carry-in c from the corresponding input ports (X,Y,*Cin*). It should then compute the sum, s, and carry-out, d, and output the results on the corresponding output ports (*Sum*, *Cout*). Create environment processes to generate random data using the selection procedure and consume the data. Simulate until you are convinced that it works.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.channel.all;
entity adder is
  port(X:inout channel:=init_channel;
       Y:inout channel:=init_channel;
       Cin:inout channel:=init_channel;
       Sum:inout channel:=init_channel;
```

```
      Cout:inout channel:=init_channel);
end entity adder;
architecture behavior of adder is
  signal a:std_logic_vector(3 downto 0);
  signal b:std_logic_vector(3 downto 0);
  signal c:std_logic_vector(0 downto 0);
  signal s:std_logic_vector(4 downto 0);
begin
process
begin
  receive(X,a,Y,b,Cin,c);
  s <= ("0" & a) + ("0" & b) + ("0000" & c);
  wait for 5 ns;
  send(Sum,s(3 downto 0),Cout,s(4 downto 4));
end process;
end behavior;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.channel.all;
entity adder_env is
  port(X:inout channel:=init_channel;
       Y:inout channel:=init_channel;
       Cin:inout channel:=init_channel;
       Sum:inout channel:=init_channel;
       Cout:inout channel:=init_channel);
end entity adder_env;
architecture behavior of adder_env is
  signal a:std_logic_vector(3 downto 0);
  signal b:std_logic_vector(3 downto 0);
  signal c:std_logic_vector(0 downto 0);
  signal s:std_logic_vector(3 downto 0);
  signal d:std_logic_vector(0 downto 0);
  signal t:std_logic_vector(4 downto 0);
begin
process
begin
  a <= selection(16,4);
  b <= selection(16,4);
  c <= selection(1,1);
  wait for 5 ns;
  send(X,a,Y,b,Cin,c);
  receive(Sum,s,Cout,d);
  t <= ("0" & a) + ("0" & b) + ("0000" & c);
  wait for 5 ns;
  assert (s=t(3 downto 0))
```

```
    report "wrong answer"
    severity warning;
  assert (d=t(4 downto 4))
    report "wrong answer"
    severity warning;
end process;
end behavior;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.channel.all;
entity adder_top is
end entity adder_top;
architecture structure of adder_top is
  component adder
    port(X:inout channel:=init_channel;
         Y:inout channel:=init_channel;
         Cin:inout channel:=init_channel;
         Sum:inout channel:=init_channel;
         Cout:inout channel:=init_channel);
  end component;
  component adder_env
    port(X:inout channel:=init_channel;
         Y:inout channel:=init_channel;
         Cin:inout channel:=init_channel;
         Sum:inout channel:=init_channel;
         Cout:inout channel:=init_channel);
  end component;
  signal X:channel:=init_channel;
  signal Y:channel:=init_channel;
  signal Cin:channel:=init_channel;
  signal Sum:channel:=init_channel;
  signal Cout:channel:=init_channel;
begin
  THE_ADDER:adder
    port map(X=>X,Y=>Y,Cin=>Cin,Sum=>Sum,Cout=>Cout);
  THE_ADDER_ENV:adder_env
    port map(X=>X,Y=>Y,Cin=>Cin,Sum=>Sum,Cout=>Cout);
end structure;
```

2.1.2. Specify a 1-bit full adder using VHDL. The *adder1* process should accept two 1-bit operands a and b and the carry-in c from the corresponding ports (X,Y,Cin). It should then compute the 1-bit sum, s, and carry-out, d, and output the results on the corresponding output ports (*Sum*, *Cout*). You may use only logic functions (no arithmetic functions). Create an environment process which communicates with the 1-bit adder and simulate until you are convinced that it works.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.channel.all;
entity adder1 is
  port(X:inout channel:=init_channel;
       Y:inout channel:=init_channel;
       Cin:inout channel:=init_channel;
       Sum:inout channel:=init_channel;
       Cout:inout channel:=init_channel);
end entity adder1;
architecture behavior of adder1 is
  signal a:std_logic;
  signal b:std_logic;
  signal c:std_logic;
  signal s:std_logic;
  signal d:std_logic;
begin
process
begin
  receive(X,a,Y,b,Cin,c);
  s <= a xor b xor c;
  d <= (a and b) or (a and c) or (b and c);
  wait for 5 ns;
  send(Sum,s,Cout,d);
end process;
end behavior;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.channel.all;
entity adder1_env is
  port(X:inout channel:=init_channel;
       Y:inout channel:=init_channel;
       Cin:inout channel:=init_channel;
       Sum:inout channel:=init_channel;
       Cout:inout channel:=init_channel);
end entity adder1_env;
architecture behavior of adder1_env is
  signal a:std_logic;
  signal b:std_logic;
  signal c:std_logic;
  signal s:std_logic;
  signal d:std_logic;
begin
```

```
process
begin
  a <= selection;
  b <= selection;
  c <= selection;
  wait for 5 ns;
  send(X,a,Y,b,Cin,c);
  receive(Sum,s,Cout,d);
  assert (s=(a xor b xor c))
    report "wrong answer" severity warning;
  assert (d=((a and b) or (a and c) or (b and c)))
    report "wrong answer" severity warning;
  wait for 5 ns;
end process;
end behavior;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.channel.all;
entity adder1_top is
end entity adder1_top;
architecture structure of adder1_top is
  component adder1
    port(X:inout channel:=init_channel;
         Y:inout channel:=init_channel;
         Cin:inout channel:=init_channel;
         Sum:inout channel:=init_channel;
         Cout:inout channel:=init_channel);
  end component;
  component adder1_env
    port(X:inout channel:=init_channel;
         Y:inout channel:=init_channel;
         Cin:inout channel:=init_channel;
         Sum:inout channel:=init_channel;
         Cout:inout channel:=init_channel);
  end component;
  signal X:channel:=init_channel;
  signal Y:channel:=init_channel;
  signal Cin:channel:=init_channel;
  signal Sum:channel:=init_channel;
  signal Cout:channel:=init_channel;
begin
  THE_ADDER:adder1
    port map(X=>X,Y=>Y,Cin=>Cin,Sum=>Sum,Cout=>Cout);
  THE_ADDER_ENV:adder1_env
    port map(X=>X,Y=>Y,Cin=>Cin,Sum=>Sum,Cout=>Cout);
end structure;
```

2.1.3. Use structural VHDL to build a 4-bit adder using your 1-bit full adders from Problem 2.1.2. Create an environment process which communicates with the 4 bits of the adder. Convince yourself through simulation that it performs the same function as your 4-bit adder from Problem 2.1.1.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.channel.all;
entity adder4_env is
  port(X3:inout channel:=init_channel;
       Y3:inout channel:=init_channel;
       X2:inout channel:=init_channel;
       Y2:inout channel:=init_channel;
       X1:inout channel:=init_channel;
       Y1:inout channel:=init_channel;
       X0:inout channel:=init_channel;
       Y0:inout channel:=init_channel;
       Cin:inout channel:=init_channel;
       S3:inout channel:=init_channel;
       S2:inout channel:=init_channel;
       S1:inout channel:=init_channel;
       S0:inout channel:=init_channel;
       Cout:inout channel:=init_channel);
end entity adder4_env;
architecture behavior of adder4_env is
  signal a:std_logic_vector(3 downto 0);
  signal b:std_logic_vector(3 downto 0);
  signal c:std_logic_vector(0 downto 0);
  signal t:std_logic_vector(4 downto 0);
  signal s:std_logic_vector(3 downto 0);
  signal d:std_logic;
begin
process
begin
  a <= selection(16,4);
  b <= selection(16,4);
  c <= selection(1,1);
  wait for 5 ns;
  t <= ("0" & a) + ("0" & b) + ("0000" & c);
  wait for 5 ns;
  send(X0,a(0),Y0,b(0));
  send(X1,a(1),Y1,b(1));
  send(X2,a(2),Y2,b(2));
  send(X3,a(3),Y3,b(3));
  send(Cin,c(0));
end process;
```

```
process
begin
  receive(Cout,d);
  receive(S0,s(0));
  receive(S1,s(1));
  receive(S2,s(2));
  receive(S3,s(3));
  assert (s=t(3 downto 0))
    report "wrong answer"
    severity warning;
  assert (d=t(4))
    report "wrong answer"
    severity warning;
  wait for 5 ns;
end process;
end behavior;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.channel.all;
entity adder4_top is
end entity adder4_top;
architecture structure of adder4_top is
  component adder1
    port(X:inout channel:=init_channel;
         Y:inout channel:=init_channel;
         Cin:inout channel:=init_channel;
         Sum:inout channel:=init_channel;
         Cout:inout channel:=init_channel);
  end component;
    component adder4_env
  port(X3:inout channel:=init_channel;
       Y3:inout channel:=init_channel;
       X2:inout channel:=init_channel;
       Y2:inout channel:=init_channel;
       X1:inout channel:=init_channel;
       Y1:inout channel:=init_channel;
       X0:inout channel:=init_channel;
       Y0:inout channel:=init_channel;
       Cin:inout channel:=init_channel;
       S3:inout channel:=init_channel;
       S2:inout channel:=init_channel;
       S1:inout channel:=init_channel;
       S0:inout channel:=init_channel;
       Cout:inout channel:=init_channel);
  end component;
  signal X3:channel:=init_channel;
```

```
  signal Y3:channel:=init_channel;
  signal X2:channel:=init_channel;
  signal Y2:channel:=init_channel;
  signal X1:channel:=init_channel;
  signal Y1:channel:=init_channel;
  signal X0:channel:=init_channel;
  signal Y0:channel:=init_channel;
  signal Cin:channel:=init_channel;
  signal S3:channel:=init_channel;
  signal S2:channel:=init_channel;
  signal S1:channel:=init_channel;
  signal S0:channel:=init_channel;
  signal Cout:channel:=init_channel;
  signal C2:channel:=init_channel;
  signal C1:channel:=init_channel;
  signal C0:channel:=init_channel;
begin
  ADDER_0:adder1
    port map(X=>X0,Y=>Y0,Cin=>Cin,Sum=>S0,Cout=>C0);
  ADDER_1:adder1
    port map(X=>X1,Y=>Y1,Cin=>C0,Sum=>S1,Cout=>C1);
  ADDER_2:adder1
    port map(X=>X2,Y=>Y2,Cin=>C1,Sum=>S2,Cout=>C2);
  ADDER_3:adder1
    port map(X=>X3,Y=>Y3,Cin=>C2,Sum=>S3,Cout=>Cout);
  THE_ADDER_ENV:adder4_env
    port map(X0=>X0,Y0=>Y0,X1=>X1,Y1=>Y1,X2=>X2,
             Y2=>Y2,X3=>X3,Y3=>Y3,Cin=>Cin,S0=>S0,
             S1=>S1,S2=>S2,S3=>S3,Cout=>Cout);
end structure;
```

2.1.4. Optimize your 1-bit full adder from Problem 2.1.2 so that it takes advantage of the fact that for most computations, the longest carry chain is significantly shorter than n, where n is the number of bits in your adder.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.channel.all;
entity adder1_opt is
  port(X:inout channel:=init_channel;
       Y:inout channel:=init_channel;
       Cin:inout channel:=init_channel;
       Sum:inout channel:=init_channel;
       Cout:inout channel:=init_channel);
end entity adder1_opt;
architecture behavior of adder1_opt is
  signal a:std_logic;
  signal b:std_logic;
```

```
  signal c:std_logic;
  signal s:std_logic;
  signal d:std_logic;
begin
process
begin
  receive(X,a,Y,b);
  if (a=b) then
    d <= a;
    wait for 5 ns;
    send(Cout,d);
    receive(Cin,c);
    s <= c;
    wait for 5 ns;
    send(Sum,s);
  else
    receive(Cin,c);
    s <= not c;
    d <= c;
    wait for 5 ns;
    send(Sum,s,Cout,d);
  end if;
end process;
end behavior;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.channel.all;
entity adder4_opt_top is
end entity adder4_opt_top;
architecture structure of adder4_opt_top is
  component adder1_opt
    port(X:inout channel:=init_channel;
         Y:inout channel:=init_channel;
         Cin:inout channel:=init_channel;
         Sum:inout channel:=init_channel;
         Cout:inout channel:=init_channel);
  end component;
  component adder4_env
  port(X3:inout channel:=init_channel;
       Y3:inout channel:=init_channel;
       X2:inout channel:=init_channel;
       Y2:inout channel:=init_channel;
       X1:inout channel:=init_channel;
       Y1:inout channel:=init_channel;
       X0:inout channel:=init_channel;
       Y0:inout channel:=init_channel;
```

```
        Cin:inout channel:=init_channel;
        S3:inout channel:=init_channel;
        S2:inout channel:=init_channel;
        S1:inout channel:=init_channel;
        S0:inout channel:=init_channel;
        Cout:inout channel:=init_channel);
    end component;
    signal X3:channel:=init_channel;
    signal Y3:channel:=init_channel;
    signal X2:channel:=init_channel;
    signal Y2:channel:=init_channel;
    signal X1:channel:=init_channel;
    signal Y1:channel:=init_channel;
    signal X0:channel:=init_channel;
    signal Y0:channel:=init_channel;
    signal Cin:channel:=init_channel;
    signal S3:channel:=init_channel;
    signal S2:channel:=init_channel;
    signal S1:channel:=init_channel;
    signal S0:channel:=init_channel;
    signal Cout:channel:=init_channel;
    signal C2:channel:=init_channel;
    signal C1:channel:=init_channel;
    signal C0:channel:=init_channel;
  begin
    ADDER_0:adder1_opt
      port map(X=>X0,Y=>Y0,Cin=>Cin,Sum=>S0,Cout=>C0);
    ADDER_1:adder1_opt
      port map(X=>X1,Y=>Y1,Cin=>C0,Sum=>S1,Cout=>C1);
    ADDER_2:adder1_opt
      port map(X=>X2,Y=>Y2,Cin=>C1,Sum=>S2,Cout=>C2);
    ADDER_3:adder1_opt
      port map(X=>X3,Y=>Y3,Cin=>C2,Sum=>S3,Cout=>Cout);
    THE_ADDER_ENV:adder4_env
      port map(X0=>X0,Y0=>Y0,X1=>X1,Y1=>Y1,X2=>X2,
               Y2=>Y2,X3=>X3,Y3=>Y3,Cin=>Cin,S0=>S0,
               S1=>S1,S2=>S2,S3=>S3,Cout=>Cout);
  end structure;
```

2.2 Specify and simulate a four-element stack using a channel-level model. The stack should be constructed using a number of identical modules which will each hold a single 8-bit data value. Each module has two passive channels to its left, *Push* and *Pop*, and two active channels to its right, *Put* and *Get*. The module should wait to get either a communication on its *Push* or its *Pop* channel. If it detects a communication on its *Push* channel and it is empty, it should receive the data and store the value received internally. If it is full, it should communicate its data to the right using its *Put* channel, then complete the receive from the *Push* channel. If it gets a communication

on its *Pop* channel and it is full, it should send its data over the *Pop* channel. If it is empty, it should request data over its *Get* channel and then forward the received data out the *Pop* channel. For simplicity, assume that the environment will never push data into a full stack or pop data from an empty stack.

```
library ieee;
use ieee.std_logic_1164.all;
use work.channel.all;
entity stack is
  port(Push:inout channel:=init_channel;
       Pop:inout channel:=init_channel;
       Put:inout channel:=init_channel;
       Get:inout channel:=init_channel);
end entity stack;
architecture behavior of stack is
  signal data:std_logic_vector(7 downto 0);
  signal full:std_logic:='0';
begin
process
begin
  if (probe(Push)) then
    if (full='0') then
      receive(Push,data);
      full <= '1';
      wait for 5 ns;
    else
      send(Put,data);
      receive(Push,data);
    end if;
  elsif (probe(Pop)) then
    if (full='0') then
      receive(Get,data);
      send(Pop,data);
    else
      send(Pop,data);
      full <= '0';
      wait for 5 ns;
    end if;
  end if;
  wait for 5 ns;
end process;
end behavior;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.channel.all;
entity stack_env is
  port(Push:inout channel:=init_channel;
       Pop:inout channel:=init_channel);
end entity stack_env;
architecture behavior of stack_env is
  signal data:std_logic_vector(7 downto 0);
begin
process
  variable z:integer;
  variable entries:integer:=0;
begin
  if (entries=0) then
    data <= selection(255,8);
    wait for 5 ns;
    send(Push,data);
    entries:=entries+1;
  elsif (entries=4) then
    receive(Pop,data);
    entries:=entries-1;
  else
    z:=selection(2);
    if (z=1) then
      data <= selection(255,8);
      wait for 5 ns;
      send(Push,data);
      entries:=entries+1;
    else
      receive(Pop,data);
      entries:=entries-1;
    end if;
  end if;
  wait for 5 ns;
end process;
end behavior;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.channel.all;
entity stack_top is
end entity stack_top;
architecture structure of stack_top is
  component stack
    port(Push:inout channel:=init_channel;
         Pop:inout channel:=init_channel;
         Put:inout channel:=init_channel;
         Get:inout channel:=init_channel);
  end component;
  component stack_env
    port(Push:inout channel:=init_channel;
         Pop:inout channel:=init_channel);
  end component;
  signal Push:channel:=init_channel;
  signal Pop:channel:=init_channel;
  signal Put0:channel:=init_channel;
  signal Get0:channel:=init_channel;
  signal Put1:channel:=init_channel;
  signal Get1:channel:=init_channel;
  signal Put2:channel:=init_channel;
  signal Get2:channel:=init_channel;
  signal Put3:channel:=init_channel;
  signal Get3:channel:=init_channel;
begin
  STACK0:stack
    port map(Push=>Push,Pop=>Pop,Put=>Put0,Get=>Get0);
  STACK1:stack
    port map(Push=>Put0,Pop=>Get0,Put=>Put1,Get=>Get1);
  STACK2:stack
    port map(Push=>Put1,Pop=>Get1,Put=>Put2,Get=>Get2);
  STACK3:stack
    port map(Push=>Put2,Pop=>Get2,Put=>Put3,Get=>Get3);
  THE_STACK_ENV:stack_env
    port map(Push=>Push,Pop=>Pop);
end structure;
```

2.3 Specify and simulate a 4-bit shifter. It should be constructed using three identical modules which each hold a single bit of data, and a special module at the end of the shifter. Each module has a *Load* channel, *Shift_in* channel, *Shift_out* channel, *Done_in* channel, *Done_out* channel, and an *Output* channel. Each module waits to receive a communication on its *Load* channel, at which time it accepts a bit of data. Next, the most significant bit waits to receive a communication from the environment on either its *Shift_in* channel or its *Done_in* channel. If it is on the *Shift_in* channel, it sends its bit to the next most significant bit over the *Shift_out* channel and it accepts a new bit from the *Shift_in* channel. If it receives a communication on its *Done_in* channel, it sends its bit out the *Output* channel and sends a communication on its *Done_out* channel. Note that the *Done_in* and *Done_out* channels do not need to carry any data; they are used only for synchronization. The block at the end does not have a *shift_out* or *Done_out* channel.

```
-----------------------------------------------
-- This is the standard 1 bit cell for a shifter
-----------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.channel.all;
entity shift_base_cell is
  port(load:inout channel:=init_channel;
       shift_in:inout channel:=init_channel;
       shift_out:inout channel:=init_channel;
       done_in:inout channel:=init_channel;
       done_out:inout channel:=init_channel;
       output:inout channel:=init_channel);
end shift_base_cell;
architecture behavior of shift_base_cell is
  signal a_bit:std_logic;
  begin
  process
  begin
    if(probe(load)) then
      receive(load,a_bit);
    elsif(probe(shift_in)) then
      send(shift_out,a_bit);
      receive(shift_in,a_bit);
    elsif(probe(done_in)) then
      receive(done_in);
      send(output,a_bit);
      send(done_out);
    end if;
    wait for delay(5,10);
  end process;
end behavior;
```

```
----------------------------------------
-- This is the end 1 bit cell for a shifter
----------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.channel.all;
entity shift_end_cell is
  port(load:inout channel:=init_channel;
       shift_in:inout channel:=init_channel;
       done_in:inout channel:=init_channel;
       output:inout channel:=init_channel);
end shift_end_cell;
architecture behavior of shift_end_cell is
  signal a_bit:std_logic;
  begin
  process
  begin
    if(probe(load)) then
      receive(load, a_bit);
    elsif(probe(shift_in)) then
      receive(shift_in, a_bit);
    elsif(probe(done_in)) then
      receive(done_in);
      send(output, a_bit);
    end if;
    wait for delay(5,10);
  end process;
end behavior;

-------------------------------
-- 4-bit Shifter for Problem 2.3
-------------------------------
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.channel.all;
entity four_bit_shifter is
  port(main_shift_in:inout channel:=init_channel;
       main_done_in:inout channel:=init_channel;
       load0:inout channel:=init_channel;
       load1:inout channel:=init_channel;
       load2:inout channel:=init_channel;
       load3:inout channel:=init_channel;
       out0:inout channel:=init_channel;
       out1:inout channel:=init_channel;
       out2:inout channel:=init_channel;
       out3:inout channel:=init_channel);
end four_bit_shifter;
```

```
architecture structure of four_bit_shifter is
  component shift_base_cell
    port(load:inout channel:=init_channel;
       shift_in:inout channel:=init_channel;
       shift_out:inout channel:=init_channel;
       done_in:inout channel:=init_channel;
       done_out:inout channel:=init_channel;
       output:inout channel:=init_channel);
  end component;
  component shift_end_cell
    port(load:inout channel:=init_channel;
       shift_in:inout channel:=init_channel;
       done_in:inout channel:=init_channel;
       output:inout channel:=init_channel);
  end component;
  signal s_com01:channel:=init_channel;
  signal s_com12:channel:=init_channel;
  signal s_com23:channel:=init_channel;
  signal d_com01:channel:=init_channel;
  signal d_com12:channel:=init_channel;
  signal d_com23:channel:=init_channel;
begin
  BIT_ZERO:shift_base_cell
    port map(shift_in => main_shift_in,
             shift_out => s_com01,
             load => load0,
             done_in => main_done_in,
             done_out => d_com01,
             output => out0);
  BIT_ONE:shift_base_cell
    port map(shift_in => s_com01,
             shift_out  => s_com12,
             load => load1,
             done_in => d_com01,
             done_out => d_com12,
             output => out1);
  BIT_TWO:shift_base_cell
    port map(shift_in  => s_com12,
             shift_out => s_com23,
             load => load2,
             done_in => d_com12,
             done_out => d_com23,
             output => out2);
  BIT_THREE:shift_end_cell
    port map(shift_in => s_com23,
             load => load3,
             done_in => d_com23,
             output => out3);
end structure;
```

```
-----------------------------
-- Test bench for Problem 2.3
-----------------------------
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.channel.all;
entity env_struc is
  port(done:inout channel:=init_channel;
       shift:inout channel:=init_channel;
       load0_tl:inout channel:=init_channel;
       load1_tl:inout channel:=init_channel;
       load2_tl:inout channel:=init_channel;
       load3_tl:inout channel:=init_channel;
       out0_tl:inout channel:=init_channel;
       out1_tl:inout channel:=init_channel;
       out2_tl:inout channel:=init_channel;
       out3_tl:inout channel:=init_channel);
end env_struc;
architecture testing of env_struc is
  component four_bit_shifter
  port(main_shift_in:inout channel:=init_channel;
       main_done_in:inout channel:=init_channel;
       load0:inout channel:=init_channel;
       load1:inout channel:=init_channel;
       load2:inout channel:=init_channel;
       load3:inout channel:=init_channel;
       out0:inout channel:=init_channel;
       out1:inout channel:=init_channel;
       out2:inout channel:=init_channel;
       out3:inout channel:=init_channel);
  end component;
  signal done_or_shift:std_logic;
    begin
  shifter:four_bit_shifter
    port map(main_shift_in => shift,
             main_done_in => done,
             load0 => load0_tl,
             load1 => load1_tl,
             load2 => load2_tl,
             load3 => load3_tl,
             out0 => out0_tl,
             out1 => out1_tl,
             out2 => out2_tl,
             out3 => out3_tl);
  process
    variable z:integer;
    variable ld:std_logic_vector(3 downto 0);
    variable outp:std_logic_vector(3 downto 0);
```

```
begin
  --randomly choose a value to load into the shifter
  ld:=selection(16,4);
  --load the shifter
  send(load0_tl,ld(0));
  send(load1_tl,ld(1));
  send(load2_tl,ld(2));
  send(load3_tl,ld(3));
  --randomly choose to send a done or shift signal
  z:=selection(2);
  case z is
    when 1 =>
      done_or_shift <= '0';
    when others =>
      done_or_shift <= '1';
  end case;
  wait for delay(5,10);
      while(done_or_shift = '1')
  loop
    ld(3):=ld(2);
    ld(2):=ld(1);
    ld(1):=ld(0);
    --randomly pick a bit to shift in
    ld(0):=selection;
    send(shift,ld(0));
          --randomly choose done or shift before checking the
    --loop condition again
    z:=selection(2);
    case z is
      when 1 =>
        done_or_shift <= '0';
      when others =>
        done_or_shift <= '1';
    end case;
    wait for delay(5,10);
  end loop;
      --send the done signal and then check the output
  --against the expected output
  send(done);
  wait for delay(5,10);
  receive(out0_tl, ld0_out);
  receive(out1_tl, ld1_out);
  receive(out2_tl, ld2_out);
  receive(out3_tl, ld3_out);
      assert ld0_bit = ld0_out
    report "The 0 bit does not match"
    severity error;
  assert ld1_bit = ld1_out
    report "The 1 bit does not match"
```

```
        severity error;
      assert ld2_bit = ld2_out
        report "The 2 bit does not match"
        severity error;
      assert ld3_bit = ld3_out
        report "The 3 bit does not match"
        severity error;
        end process;
end;
```

2.4 In this problem you are going to specify a simple entropy decoder in VHDL using the channel model. An entropy decoder relies on a standard entropy code that represents fixed-length symbols from a source alphabet as variable-length code symbols. The idea is that common symbols are represented using short codes, and uncommon symbols are represented using longer codes. The result is that the average code length is significantly smaller than the symbol length. Such entropy codes are used in numerous audio, video, and data compression schemes (e.g., JPEG, MPEG, etc.). As an example, consider a set of 2-bit symbols (i.e., 00, 01, 10, 11) where 00 has a probability of 90 percent, 01 has a probability of 9 percent, 10 has a probability of 0.9 percent, and 11 has a probability of 0.1 percent. If we encode the symbols as follows:

- 00 - 0
- 01 - 10
- 10 - 110
- 11 - 1110

the average code length would be

$$0.9 \times 1 + 0.09 \times 2 + 0.009 \times 3 + 0.001 \times 4 = 1.11$$

This is nearly half the size of the fixed-length size of the symbols.

2.4.1. Specify in VHDL using the channel model an entropy decoder for 2-bit symbols which are encoded as described above. Assume that you have a 1-bit input port, *In*, and a 2-bit output port, *Out*. The module should receive 1 bit of data at a time and output the 2-bit symbol once it has recognized the code word. Specify environment processes to generate and consume the data. Simulate the design until you are convinced that it works.

```
-- Solution problem 2.4.1
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
```

```
use work.channel.all;
entity decoder is
end decoder;
architecture behavior of decoder is
signal Inp:channel:=init_channel;
signal Outp:channel:=init_channel;
signal InpData:std_logic;
signal OutData:std_logic_vector(1 downto 0);
signal Result:std_logic_vector(1 downto 0);
signal Zero:std_logic:='0';
signal One:std_logic:='1';
begin
decoder:process
begin
  OutData <= "00";
  wait for 5 ps;
  receive(Inp,InpData);
  while (InpData='1')
  loop
    OutData <= OutData + "01";
    wait for 5 ps;
    receive(Inp,InpData);
  end loop;
  send(Outp,OutData);
end process;
env:process
variable z:integer;
begin
  z:=selection(4);
  case (z) is
  when 1 =>
    send(Inp,Zero);
  when 2 =>
    send(Inp,One);
    send(Inp,Zero);
  when 3 =>
    send(Inp,One);
    send(Inp,One);
    send(Inp,Zero);
  when others =>
    send(Inp,One);
    send(Inp,One);
    send(Inp,One);
    send(Inp,Zero);
  end case;
  receive(Outp,Result);
  wait for 5 ps;
end process;
end behavior;
```

2.4.2. We have decided to break up the entropy decoder from the first problem into four identical blocks, and we also would like to make it programmable. Each block has a 1-bit input port, L, which receives bits from its left neighbor, a 1-bit output port, R, which transmits bits to its right neighbor, a 2-bit input port, *Load*, which receives the 2-bit symbol this block transmits when it recognizes the code, and a 2-bit output port, *Out*, which transmits its stored symbol when it recognizes the code. The behavior of each block is that it can either receive a 2-bit symbol from the *Load* port or 1 bit from its left neighbor. If it receives a 2-bit symbol from the *Load* port, that symbol is stored in an internal register and becomes the symbol for which this stage is responsible. If it receives a bit from its left neighbor, it checks if it is a 0 or 1. If it is a 0, it will output its stored symbol. If it is a 1, it strips this bit and passes the remaining bits that it receives from its left neighbor one by one to its right neighbor until it sees a 0. At that point it loops back to the beginning of the cycle and waits for the next code to be transmitted. Specify one block in VHDL. Create an environment and simulate until you are convinced that it works.

```
-- Solution problem 2.4.2
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.channel.all;
entity decoder1 is
  port(L:inout channel;
       R:inout channel;
       Load:inout channel;
       Outp:inout channel);
end decoder1;
architecture behavior of decoder1 is
signal LData:std_logic;
signal LoadData:std_logic_vector(1 downto 0);
begin
decoder1:process
begin
  if (probe(Load)) then
    receive(Load,LoadData);
  elsif (probe(L)) then
    receive(L,LData);
    wait for 10 ps;
    if (LData='0') then
      send(Outp,LoadData);
    else
      while (LData='1')
      loop
        receive(L,LData);
```

```
        send(R,LData);
      end loop;
    end if;
  end if;
  wait for 5 ps;
end process;
end behavior;

-- Solution problem 2.4.2
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.channel.all;
entity Prob2_4_2 is
end Prob2_4_2;
architecture behavior of Prob2_4_2 is
component decoder1
  port(L:inout channel;
       R:inout channel;
       Load:inout channel;
       Outp:inout channel);
end component;
signal L:channel:=init_channel;
signal R:channel:=init_channel;
signal Load:channel:=init_channel;
signal Outp:channel:=init_channel;
signal RData:std_logic;
signal LoadData:std_logic_vector(1 downto 0);
signal Result:std_logic_vector(1 downto 0);
signal Zero:std_logic:='0';
signal One:std_logic:='1';
begin
  DEC1:decoder1
    port map(L => L, R => R, Load => Load,
             Outp => Outp);
  env:process
  variable z:integer;
  begin
    LoadData <= "01";
    wait for 5 ps;
    send(Load,LoadData);
    wait for 5 ps;
    while (true)
    loop
      z:=selection(2);
      case (z) is
      when 1 =>
```

```
        send(L,Zero);
        wait for 5 ps;
        receive(Outp,Result);
      when others =>
        send(L,One);
        wait for 5 ps;
        send(L,One);
        wait for 5 ps;
        receive(R,RData);
        send(L,Zero);
        wait for 5 ps;
        receive(R,RData);
      end case;
    end loop;
end process;
end behavior;
```

2.4.3. Use structural VHDL to model an entropy decoder which can recognize codes of length 4.

```
-- Solution problem 2.4.3
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.channel.all;
entity Prob2_4_3 is
end Prob2_4_3;
architecture behavior of Prob2_4_3 is
component decoder1
  port(L:inout channel;
       R:inout channel;
       Load:inout channel;
       Outp:inout channel);
end component;
signal L:channel:=init_channel;
signal C1:channel:=init_channel;
signal C2:channel:=init_channel;
signal C3:channel:=init_channel;
signal R:channel:=init_channel;
signal Load1:channel:=init_channel;
signal Outp1:channel:=init_channel;
signal Load2:channel:=init_channel;
signal Outp2:channel:=init_channel;
signal Load3:channel:=init_channel;
signal Outp3:channel:=init_channel;
signal Load4:channel:=init_channel;
signal Outp4:channel:=init_channel;
signal LoadData:std_logic_vector(1 downto 0);
```

```
signal Result:std_logic_vector(1 downto 0);
signal Zero:std_logic:='0';
signal One:std_logic:='1';
begin
  DEC1:decoder1
    port map(L => L, R => C1, Load => Load1,
             Outp => Outp1);
  DEC2:decoder1
    port map(L => C1, R => C2, Load => Load2,
             Outp => Outp2);
  DEC3:decoder1
    port map(L => C2, R => C3, Load => Load3,
             Outp => Outp3);
  DEC4:decoder1
    port map(L => C3, R => R, Load => Load4,
             Outp => Outp4);
  env:process
  variable z:integer;
  begin
    LoadData <= "00";
    wait for 5 ps;
    send(Load1,LoadData);
    wait for 5 ps;
    LoadData <= "01";
    wait for 5 ps;
    send(Load2,LoadData);
    wait for 5 ps;
    LoadData <= "10";
    wait for 5 ps;
    send(Load3,LoadData);
    wait for 5 ps;
    LoadData <= "11";
    wait for 5 ps;
    send(Load4,LoadData);
    wait for 5 ps;
    while (true)
    loop
      z:=selection(4);
      case (z) is
      when 1 =>
        send(L,Zero);
        wait for 5 ps;
        receive(Outp1,Result);
      when 2 =>
        send(L,One);
        wait for 5 ps;
        send(L,Zero);
        wait for 5 ps;
        receive(Outp2,Result);
```

```
        when 3 =>
          send(L,One);
          wait for 5 ps;
          send(L,One);
          wait for 5 ps;
          send(L,Zero);
          wait for 5 ps;
          receive(Outp3,Result);
        when others =>
          send(L,One);
          wait for 5 ps;
          send(L,One);
          wait for 5 ps;
          send(L,One);
          wait for 5 ps;
          send(L,Zero);
          wait for 5 ps;
          receive(Outp4,Result);
      end case;
    end loop;
end process;
end behavior;
```

2.4.4. The results being transmitted from the *Out* ports from each of the four blocks from the last problem need to be multiplexed together to produce the symbol that has just been recognized. There is a potential race problem with this design as described above. One stage may come to the end of forwarding bits (i.e., it sees the trailing 0) and immediately sees another 0, causing it to transmit its symbol on its *Out* port. This transmission may occur before the transmission by the later stage of the previous symbol. In other words, the symbols may get recognized out of order. Add another channel to forward acknowledgments of transmissions of symbols back to the previous blocks. Try to do this in a way which will prevent transmissions from getting out of order and does not hold up the recognition step.

```
-- Solution problem 2.4.4
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.channel.all;
entity decoder1b is
  port(L:inout channel;
       R:inout channel;
       LAck:inout channel;
       RAck:inout channel;
       Load:inout channel;
       Outp:inout channel);
```

```
end decoder1b;
architecture behavior of decoder1b is
signal LData:std_logic;
signal LoadData:std_logic_vector(1 downto 0);
begin
decoder1:process
begin
  if (probe(Load)) then
    receive(Load,LoadData);
  elsif (probe(L)) then
    receive(L,LData);
    wait for 10 ps;
    if (LData='0') then
      send(Outp,LoadData);
      send(LAck);
    else
      while (LData='1')
      loop
        receive(L,LData);
        send(R,LData);
      end loop;
      receive(RAck);
      send(LAck);
    end if;
  end if;
  wait for 5 ps;
end process;
end behavior;

-- Solution problem 2.4.4
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.channel.all;
entity Prob2_4_4 is
end Prob2_4_4;
architecture behavior of Prob2_4_4 is
component decoder1b
  port(L:inout channel;
       R:inout channel;
       LAck:inout channel;
       RAck:inout channel;
       Load:inout channel;
       Outp:inout channel);
end component;
signal L:channel:=init_channel;
signal C1:channel:=init_channel;
```

```
signal C2:channel:=init_channel;
signal C3:channel:=init_channel;
signal R:channel:=init_channel;
signal LAck:channel:=init_channel;
signal C1Ack:channel:=init_channel;
signal C2Ack:channel:=init_channel;
signal C3Ack:channel:=init_channel;
signal RAck:channel:=init_channel;
signal Load1:channel:=init_channel;
signal Outp1:channel:=init_channel;
signal Load2:channel:=init_channel;
signal Outp2:channel:=init_channel;
signal Load3:channel:=init_channel;
signal Outp3:channel:=init_channel;
signal Load4:channel:=init_channel;
signal Outp4:channel:=init_channel;
signal LoadData:std_logic_vector(1 downto 0);
signal Result:std_logic_vector(1 downto 0);
signal Zero:std_logic:='0';
signal One:std_logic:='1';
begin
  DEC1:decoder1b
    port map(L => L, R => C1, LAck => LAck,
             RAck => C1Ack, Load => Load1,
             Outp => Outp1);
  DEC2:decoder1b
    port map(L => C1, R => C2, LAck => C1Ack,
             RAck => C2Ack, Load => Load2,
             Outp => Outp2);
  DEC3:decoder1b
    port map(L => C2, R => C3, LAck => C2Ack,
             RAck => C3Ack, Load => Load3,
             Outp => Outp3);
  DEC4:decoder1b
    port map(L => C3, R => R, LAck => C3Ack,
             RAck => RAck, Load => Load4,
             Outp => Outp4);
  env:process
  variable z:integer;
  begin
    LoadData <= "00";
    wait for 5 ps;
    send(Load1,LoadData);
    wait for 5 ps;
    LoadData <= "01";
    wait for 5 ps;
    send(Load2,LoadData);
    wait for 5 ps;
    LoadData <= "10";
```

```
    wait for 5 ps;
    send(Load3,LoadData);
    wait for 5 ps;
    LoadData <= "11";
    wait for 5 ps;
    send(Load4,LoadData);
    wait for 5 ps;
    while (true)
    loop
      z:=selection(4);
      case (z) is
      when 1 =>
        send(L,Zero);
        wait for 5 ps;
      when 2 =>
        send(L,One);
        wait for 5 ps;
        send(L,Zero);
        wait for 5 ps;
      when 3 =>
        send(L,One);
        wait for 5 ps;
        send(L,One);
        wait for 5 ps;
        send(L,Zero);
        wait for 5 ps;
      when others =>
        send(L,One);
        wait for 5 ps;
        send(L,One);
        wait for 5 ps;
        send(L,One);
        wait for 5 ps;
        send(L,Zero);
        wait for 5 ps;
      end case;
    end loop;
  end process;
  mux:process
  begin
    if (probe(Outp1)) then
      receive(Outp1,Result);
    elsif (probe(Outp2)) then
      receive(Outp2,Result);
    elsif (probe(Outp3)) then
      receive(Outp3,Result);
    elsif (probe(Outp4)) then
      receive(Outp4,Result);
    end if;
```

```
    wait for 5 ps;
  end process;
  LAckProc:process
  begin
    receive(LAck);
    wait for 5 ps;
  end process;
end behavior;
```

2.5 Add a jump and link (*jal*) instruction to the MiniMIPS example. This instruction is of the jump format, and the opcode is 000011. This instruction should send the old PC+1 to register 31 (effectively register 7 in our scaled-down design), and set the PC to the address.

```
--------------------
-- Name:  fetch.vhd
--------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.channel.all;
entity fetch is
  port(imem_address:inout channel:=init_channel;
       imem_data:inout channel:=init_channel;
       decode_instr:inout channel:=init_channel;
       branch_decision:inout channel:=init_channel;
       PCout:inout channel:=init_channel);     end fetch;
architecture behavior of fetch is
  signal PC:std_logic_vector( 31 downto 0 ):=(others => '0');
  signal instr:std_logic_vector( 31 downto 0);
  signal bd:std_logic_vector( 0 downto 0 );
  alias opcode is instr(31 downto 26);
  alias offset is instr(15 downto 0);
  alias address is instr(25 downto 0);
begin
process
  variable branch_offset:std_logic_vector( 31 downto 0);
begin
  send(imem_address,PC);
  receive(imem_data,instr);
  PC <= PC + 1;
  wait for 5 ns;
  case opcode is
  when "000100" => -- beq
    send(decode_instr,instr);
    receive(branch_decision,bd);
    if (bd = "1") then
```

```
      branch_offset(31 downto 16):=(others => instr(15));
      branch_offset(15 downto 0):=offset;
      PC <= PC + branch_offset;
      wait for 5 ns;
    end if;
  when "000110" => -- j
    PC <= (PC(31 downto 26) & address);
    wait for 5 ns;
  when "000011" => -- jal
    send(decode_instr,instr);
    send(PCout,PC);
    PC <= (PC(31 downto 26) & address);
    wait for 5 ns;
  when others =>
    send(decode_instr,instr);
  end case;
end process;
end behavior;

---------------------
-- Name:  decode.vhd
---------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.channel.all;
entity decode is
  port(decode_instr:inout channel:=init_channel;
       execute_op:inout channel:=init_channel;
       execute_rs:inout channel:=init_channel;
       execute_rt:inout channel:=init_channel;
       execute_rd:inout channel:=init_channel;
       execute_func:inout channel:=init_channel;
       execute_offset:inout channel:=init_channel;
       dmem_datain:inout channel:=init_channel;
       dmem_dataout:inout channel:=init_channel;
       PCin:inout channel:=init_channel);     end decode;
architecture behavior of decode is
  type registers is array (0 to 7) of
       std_logic_vector(31 downto 0);
  signal instr:std_logic_vector( 31 downto 0);
  alias op is instr(31 downto 26);
  alias rs is instr(23 downto 21);
  alias rt is instr(18 downto 16);
  alias rd is instr(13 downto 11);
  alias func is instr(5 downto 0);
  alias offset is instr(15 downto 0);
```

```
  signal reg:reg_array := (X"00000000",
                           X"11111111",
                           X"22222222",
                           X"33333333",
                           X"44444444",
                           X"55555555",
                           X"66666666",
                           X"77777777");
  signal reg_rs:std_logic_vector(31 downto 0);
  signal reg_rt:std_logic_vector(31 downto 0);
  signal reg_rd:std_logic_vector(31 downto 0);
begin
process
begin
  receive(decode_instr,instr);
  case op is
  when "000000" => -- ALU op
    send(execute_op,op);
    send(execute_func,func);
    reg_rs <= reg(conv_integer(rs));
    reg_rt <= reg(conv_integer(rt));
    wait for 5 ns;
    send(execute_rs,reg_rs);
    send(execute_rt,reg_rt);
    receive(execute_rd,reg_rd);
    reg(conv_integer(rd)) <= reg_rd;
    wait for 5 ns;
  when "000100" => -- beq
    send(execute_op,op);
    reg_rs <= reg(conv_integer(rs));
    reg_rt <= reg(conv_integer(rt));
    wait for 5 ns;
    send(execute_rs,reg_rs);
    send(execute_rt,reg_rt);
  when "100011" => -- lw
    send(execute_op,op);
    reg_rs <= reg(conv_integer(rs));
    wait for 5 ns;
    send(execute_rs,reg_rs);
    send(execute_offset,offset);
    receive(dmem_dataout,reg_rt);
    reg(conv_integer(rt)) <= reg_rt;
    wait for 5 ns;
  when "101011" => -- sw
    send(execute_op,op);
    reg_rs <= reg(conv_integer(rs));
    reg_rt <= reg(conv_integer(rt));
    wait for 5 ns;
    send(execute_rs,reg_rs);
```

```
    send(execute_offset,offset);
    send(dmem_datain,reg_rt);
  when "000011" => -- jal
    receive(PCin,reg_rd);
    reg(7) <= reg_rd;
    wait for 5 ns;
  when others => -- undefined
    assert false
      report "Illegal instruction"
      severity error;
  end case;
end process;
end behavior;
```

2.6 Extend the MiniMIPS to include the set less than (*slt*) instruction. This instruction is of the register format, and the opcode is 000000 with function code 101010. It checks if register *rs* is less than *rt*. If so, it sets *rd* to 1. Otherwise, it sets *rd* to 0. This instruction typically is used before a branch.

```
----------------------
-- execute.vhd
----------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.channel.all;
entity execute is
  port(execute_op:inout channel:=init_channel;
       execute_rs:inout channel:=init_channel;
       execute_rt:inout channel:=init_channel;
       execute_rd:inout channel:=init_channel;
       execute_func:inout channel:=init_channel;
       execute_offset:inout channel:=init_channel;
       dmem_addr:inout channel:=init_channel;
       dmem_rw:inout channel:=init_channel;
       branch_decision:inout channel:=init_channel);
end execute;
architecture behavior of execute is
  signal rs:std_logic_vector(31 downto 0);
  signal rt:std_logic_vector(31 downto 0);
  signal rd:std_logic_vector(31 downto 0);
  signal op:std_logic_vector(5 downto 0);
  signal func:std_logic_vector(5 downto 0);
  signal offset:std_logic_vector(15 downto 0);
  signal rw:std_logic;
  signal bd:std_logic;
begin
```

```
process
  variable addr_offset:std_logic_vector(31 downto 0);
begin
  receive(execute_op,op);
  case op is
  when "000000" => -- ALU op
    receive(execute_func,func,execute_rs,rs,
            execute_rt,rt);
    case func is
    when "100000" => -- add
      rd <= rs + rt;
    when "100010" => -- sub
      rd <= rs - rt;
    when "100100" => -- and
      rd <= rs and rt;
    when "100101" => -- or
      rd <= rs or rt;
    when "101010" => -- slt
      if (rs < rt) then
        rd <= conv_std_logic_vector(1,32);
      else
        rd <= conv_std_logic_vector(0,32);
      end if;
    when others =>
      rd <= (others => 'X'); -- undefined
    end case;
    wait for delay(5,10);
    send(execute_rd,rd);
  when "000100" => -- beq
    receive(execute_rs,rs,execute_rt,rt);
    if (rs = rt) then bd <= '1';
    else bd <= '0';
    end if;
    wait for delay(5,10);
    send(branch_decision,bd);
  when "100011" => -- lw
    receive(execute_rs,rs,execute_offset,offset);
    addr_offset(31 downto 16):=(others => offset(15));
    addr_offset(15 downto 0):=offset;
    rd <= rs + addr_offset;
    rw <= '1';
    wait for delay(5,10);
    send(dmem_addr,rd);
    send(dmem_rw,rw);
  when "101011" => -- sw
    receive(execute_rs,rs,execute_offset,offset);
    addr_offset(31 downto 16):=(others => offset(15));
    addr_offset(15 downto 0):=offset;
    rd <= rs + addr_offset;
```

```
      rw <= '0';
      wait for delay(5,10);
      send(dmem_addr,rd);
      send(dmem_rw,rw);
    when others => -- undefined
      assert false
        report "Illegal instruction"
        severity error;
    end case;
    end process;
end behavior;
```

2.7 Extend the MiniMIPS to include immediate instructions: add immediate (*addi*, op = 001000), and immediate (*andi*, op = 001100), or immediate (*ori*, op = 001101), and set less than immediate (*slti*, op = 001010). These instructions are of the load/store/branch format. The *rs* field points to one operand and the offset is the other operand. The offset is sign extended for *addi* and *slti*, while it is zero extended for *andi* and *ori*. The result is stored in the register pointed to by the *rt* field.

```
---------------------
-- decode.vhd
---------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.channel.all;
entity decode is
  port(decode_instr:inout channel:=init_channel;
       execute_op:inout channel:=init_channel;
       execute_rs:inout channel:=init_channel;
       execute_rt:inout channel:=init_channel;
       execute_rd:inout channel:=init_channel;
       execute_func:inout channel:=init_channel;
       execute_offset:inout channel:=init_channel;
       dmem_datain:inout channel:=init_channel;
       dmem_dataout:inout channel:=init_channel);
end decode;
architecture behavior of decode is
  type registers is array (0 to 7) of
       std_logic_vector(31 downto 0);
  type booleans is array (natural range <>) of boolean;
  signal instr :  std_logic_vector( 31 downto 0);
  alias op:std_logic_vector(5 downto 0) is
    instr(31 downto 26);
  alias rs:std_logic_vector(2 downto 0) is
    instr(23 downto 21);
```

```
  alias rt:std_logic_vector(2 downto 0) is
    instr(18 downto 16);
  alias rd:std_logic_vector(2 downto 0) is
    instr(13 downto 11);
  alias func:std_logic_vector(5 downto 0) is
    instr(5 downto 0);
  alias offset:std_logic_vector(15 downto 0) is
    instr(15 downto 0);
  signal reg:registers:=(X"00000000",
                         X"11111111",
                         X"22222222",
                         X"33333333",
                         X"44444444",
                         X"55555555",
                         X"66666666",
                         X"77777777");
  signal reg_rs:std_logic_vector(31 downto 0);
  signal reg_rt:std_logic_vector(31 downto 0);
  signal reg_rd:std_logic_vector(31 downto 0);
  signal reg_locks:booleans(0 to 7):=(others => false);
  signal decode_to_wb:channel:=init_channel;
  signal wb_instr:std_logic_vector(31 downto 0);
  alias wb_op:std_logic_vector(5 downto 0) is
    wb_instr(31 downto 26);
  alias wb_rt:std_logic_vector(2 downto 0) is
    wb_instr(18 downto 16);
  alias wb_rd:std_logic_vector(2 downto 0) is
    wb_instr(13 downto 11);
  signal lock:channel:=init_channel;
begin
decode:process
begin
  receive(decode_instr,instr);
  if ((reg_locks(conv_integer(rs))) or
      (reg_locks(conv_integer(rt)))) then
    wait until ((not reg_locks(conv_integer(rs))) and
                (not reg_locks(conv_integer(rt))));
  end if;
  reg_rs <= reg(conv_integer(rs));
  reg_rt <= reg(conv_integer(rt));
  send(execute_op,op);
  wait for delay(5,10);
  case op is
  when "000000" => -- ALU op
    send(execute_func,func,execute_rs,reg_rs,
         execute_rt,reg_rt);
    send(decode_to_wb,instr);
    receive(lock);
  when "000100" => -- beq
```

```
      send(execute_rs,reg_rs,execute_rt,reg_rt);
    when "100011" => -- lw
      send(execute_rs,reg_rs,execute_offset,offset);
      send(decode_to_wb,instr);
      receive(lock);
    when "101011" => -- sw
      send(execute_rs,reg_rs,execute_offset,offset,
           dmem_datain,reg_rt);
    when "001000" |
         "001100" |
         "001101" |
         "001010" => --addi, andi, ori, slti
      send(execute_rs,reg_rs,execute_offset,offset);
      send(decode_to_wb,instr);
      receive(lock);
    when others => -- undefined
      assert false
        report "Illegal instruction"
        severity error;
    end case;
  end process;
  writeback:process
  begin
    receive(decode_to_wb,wb_instr);
    case wb_op is
    when "000000" => -- ALU op
      reg_locks(conv_integer(wb_rd)) <= true;
      wait for 1 ns;
      send(lock);
      receive(execute_rd,reg_rd);
      reg(conv_integer(wb_rd)) <= reg_rd;
      wait for delay(5,10);
      reg_locks(conv_integer(wb_rd)) <= false;
      wait for delay(5,10);
    when "100011" => -- lw
      reg_locks(conv_integer(wb_rt)) <= true;
      wait for 1 ns;
      send(lock);
      receive(dmem_dataout,reg_rd);
      reg(conv_integer(wb_rt)) <= reg_rd;
      wait for delay(5,10);
      reg_locks(conv_integer(wb_rt)) <= false;
      wait for delay(5,10);
    when "001000" |
         "001100" |
         "001101" |
         "001010" => --addi, andi, ori, slti
      reg_locks(conv_integer(wb_rt)) <= true;
      wait for 1 ns;
```

```
    send(lock);
    receive(execute_rd,reg_rd);
    reg(conv_integer(wb_rt)) <= reg_rd;
    wait for delay(5,10);
    reg_locks(conv_integer(wb_rt)) <= false;
    wait for delay(5,10);
  when others => -- undefined
    wait for delay(5,10);
  end case;
end process;
end behavior;

----------------------
-- execute.vhd
----------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.channel.all;
entity execute is
  port(execute_op:inout channel:=init_channel;
       execute_rs:inout channel:=init_channel;
       execute_rt:inout channel:=init_channel;
       execute_rd:inout channel:=init_channel;
       execute_func:inout channel:=init_channel;
       execute_offset:inout channel:=init_channel;
       dmem_addr:inout channel:=init_channel;
       dmem_rw:inout channel:=init_channel;
       branch_decision:inout channel:=init_channel);
end execute;
architecture behavior of execute is
  signal rs:std_logic_vector(31 downto 0);
  signal rt:std_logic_vector(31 downto 0);
  signal rd:std_logic_vector(31 downto 0);
  signal op:std_logic_vector(5 downto 0);
  signal func:std_logic_vector(5 downto 0);
  signal offset:std_logic_vector(15 downto 0);
  signal rw:std_logic;
  signal bd:std_logic;
  signal sign_ext:std_logic_vector(31 downto 0);
begin
process
  variable addr_offset:std_logic_vector(31 downto 0);
begin
  receive(execute_op,op);
  case op is
  when "000000" => -- ALU op
```

```
    receive(execute_func,func,execute_rs,rs,
            execute_rt,rt);
    case func is
    when "100000" => -- add
      rd <= rs + rt;
    when "100010" => -- sub
      rd <= rs - rt;
    when "100100" => -- and
      rd <= rs and rt;
    when "100101" => -- or
      rd <= rs or rt;
    when others =>
      rd <= (others => 'X'); -- undefined
    end case;
    wait for delay(5,10);
    send(execute_rd,rd);
  when "000100" => -- beq
    receive(execute_rs,rs,execute_rt,rt);
    if (rs = rt) then bd <= '1';
    else bd <= '0';
    end if;
    wait for delay(5,10);
    send(branch_decision,bd);
  when "100011" => -- lw
    receive(execute_rs,rs,execute_offset,offset);
    addr_offset(31 downto 16):=(others => offset(15));
    addr_offset(15 downto 0):=offset;
    rd <= rs + addr_offset;
    rw <= '1';
    wait for delay(5,10);
    send(dmem_addr,rd);
    send(dmem_rw,rw);
  when "101011" => -- sw
    receive(execute_rs,rs,execute_offset,offset);
    addr_offset(31 downto 16):=(others => offset(15));
    addr_offset(15 downto 0):=offset;
    rd <= rs + addr_offset;
    rw <= '0';
    wait for delay(5,10);
    send(dmem_addr,rd);
    send(dmem_rw,rw);
  when "001000" => --addi
    receive(execute_rs,rs,execute_offset,offset);
    if offset(15) = '1' then
      sign_ext <= X"FFFF" & offset;
    else
      sign_ext <= X"0000" & offset;
    end if;
    wait for delay(5,10);
```

```
      rd <= rs + sign_ext;
      wait for delay(5,10);
      send(execute_rd,rd);
    when "001100" => --andi
      receive(execute_rs,rs,execute_offset,offset);
      sign_ext <= X"0000" & offset;
      wait for delay(5,10);
      rd <= rs and sign_ext;
      wait for delay(5,10);
      send(execute_rd,rd);
    when "001101" => --ori
      receive(execute_rs,rs,execute_offset,offset);
      sign_ext <= X"0000" & offset;
      wait for delay(5,10);
      rd <= rs or sign_ext;
      wait for delay(5,10);
      send(execute_rd,rd);
    when "001010" => --slti
      receive(execute_rs,rs,execute_offset,offset);
      if offset(15) = '1' then
        sign_ext <= X"FFFF" & offset;
      else
        sign_ext <= X"0000" & offset;
      end if;
      wait for delay(5,10);
      if rs < sign_ext then
        rd <= X"00000000";
      else
        rd <= X"00000001";
      end if;
      wait for delay(5,10);
      send(execute_rd,rd);
    when others => -- undefined
      assert false
        report "Illegal instruction"
        severity error;
    end case;
    end process;
  end behavior;
```

2.8 Extend the MiniMIPS to include shift instructions: shift left logical (*sll*), shift right logical (*srl*), and shift right arithmetic (*sra*). These instructions are of the register format with opcode 000000. The function codes are 000000 for *sll*, 000010 for *srl*, and 000011 for *sra*. They shift *rt* by the amount specified in the 5-bit *shamt* field and store the result in *rd*. In an arithmetic shift, the high-order bit is shifted in. In an arithmetic shift, the high-order bit is shifted in. In logical shifts, a 0 is shifted in.

2.9 Extend the MiniMIPS to include a *trap* and a return from exception (*rfe*) instruction.

2.10 Extend the MiniMIPS to support exceptions. In particular, *add* and *sub* instructions can cause an arithmetic overflow exception, and illegal instructions should also cause an exception.

```
--------------------
-- imem.vhd
--------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.channel.all;
entity imem is
  port(address:inout channel:=init_channel;
       data:inout channel:=init_channel);
end imem;
architecture behavior of imem is
  type memory is array (0 to 7) of
    std_logic_vector(31 downto 0);
  signal addr:std_logic_vector(31 downto 0);
  signal instr:std_logic_vector(31 downto 0);
begin
process
  variable imem:memory:=(
    X"8c220000", -- L: lw r2,0(r1)
    X"8c640000", --    lw r4,0(r3)
    X"00a42020", --    add r4,r5,r4
    X"20601111", --    addi r0,r3, X"1111"
    X"3001f04b", --    andi r1,r0, X"f04b"
    X"3422870f", --    ori r2,r1, X"870f"
    X"28435555", --    slti r3,r2, X"5555"
    X"2844cccc"); --   slti r4,r2 X"cccc"
begin
  receive(address,addr);
  instr <= imem(conv_integer(addr(2 downto 0)));
  wait for delay(5,10);
  send(data,instr);
end process;
end behavior;

--------------------
-- fetch.vhd
--------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
```

```
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.channel.all;
entity fetch is
  port(imem_address:inout channel:=init_channel;
       imem_data:inout channel:=init_channel;
       decode_instr:inout channel:=init_channel;
       branch_decision:inout channel:=init_channel;
       inst_PC:inout channel:=init_channel);
end fetch;
architecture behavior of fetch is
  signal PC:std_logic_vector(31 downto 0):=(others=>'0');
  signal instr:std_logic_vector(31 downto 0);
  signal bd:std_logic;
  alias opcode:std_logic_vector(5 downto 0) is
    instr(31 downto 26);
  alias offset:std_logic_vector(15 downto 0) is
    instr(15 downto 0);
  alias address:std_logic_vector(25 downto 0) is
    instr(25 downto 0);
  signal EPC:std_logic_vector(31 downto 0);
  signal Cause:std_logic_vector(31 downto 0);
begin
process
  variable branch_offset:std_logic_vector(31 downto 0);
begin
  send(imem_address,PC);
  receive(imem_data,instr);
  PC <= PC + 1;
  wait for delay(5,10);
  case opcode is
  when "000100" => -- beq
    send(decode_instr,instr);
    receive(branch_decision,bd);
    if (bd = '1') then
      branch_offset(31 downto 16):=(others=>instr(15));
      branch_offset(15 downto 0):=offset;
      PC <= PC + branch_offset;
      wait for delay(5,10);
    end if;
  when "000110" => -- j
    PC <= (PC(31 downto 26) & address);
    wait for delay(5,10);
  when "101011" |
       "100011" |
       "000000"  => --sw, ld, alu op
    send(decode_instr,instr);
    send(inst_PC, PC);
  when others =>
```

```
    EPC <= PC - 1; --PC location of bad inst
    Cause <= X"c0000000"; --cause code for
                          --the undefined instruction
    assert false
      report "Exception:  Invalid Instruction"
      severity error;
    assert false
      severity failure; -- kill the program
  end case;
end process;
end behavior;

---------------------
-- decode.vhd
---------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.channel.all;
entity decode is
  port(decode_instr:inout channel:=init_channel;
       execute_op:inout channel:=init_channel;
       execute_rs:inout channel:=init_channel;
       execute_rt:inout channel:=init_channel;
       execute_rd:inout channel:=init_channel;
       execute_func:inout channel:=init_channel;
       execute_offset:inout channel:=init_channel;
       dmem_datain:inout channel:=init_channel;
       dmem_dataout:inout channel:=init_channel;
       inst_PC_in:inout channel:=init_channel;
       inst_PC_out:inout channel:=init_channel);
end decode;
architecture behavior of decode is
  type registers is array (0 to 7) of
       std_logic_vector(31 downto 0);
  type booleans is array (natural range <>) of boolean;
  signal instr :  std_logic_vector( 31 downto 0);
  alias op:std_logic_vector(5 downto 0) is
    instr(31 downto 26);
  alias rs:std_logic_vector(2 downto 0) is
    instr(23 downto 21);
  alias rt:std_logic_vector(2 downto 0) is
    instr(18 downto 16);
  alias rd:std_logic_vector(2 downto 0) is
    instr(13 downto 11);
  alias func:std_logic_vector(5 downto 0) is
    instr(5 downto 0);
```

```
  alias offset:std_logic_vector(15 downto 0) is
    instr(15 downto 0);
  signal reg:registers:=(X"00000000",
                         X"11111111",
                         X"22222222",
                         X"33333333",
                         X"44444444",
                         X"55555555",
                         X"66666666",
                         X"77777777");
  signal reg_rs:std_logic_vector(31 downto 0);
  signal reg_rt:std_logic_vector(31 downto 0);
  signal reg_rd:std_logic_vector(31 downto 0);
  signal reg_locks:booleans(0 to 7):=(others => false);
  signal decode_to_wb:channel:=init_channel;
  signal wb_instr:std_logic_vector(31 downto 0);
  alias wb_op:std_logic_vector(5 downto 0) is
    wb_instr(31 downto 26);
  alias wb_rt:std_logic_vector(2 downto 0) is
    wb_instr(18 downto 16);
  alias wb_rd:std_logic_vector(2 downto 0) is
    wb_instr(13 downto 11);
  signal lock:channel:=init_channel;
  signal PC:std_logic_vector(31 downto 0);
begin
decode:process
begin
  receive(decode_instr,instr);
  if ((reg_locks(conv_integer(rs))) or
      (reg_locks(conv_integer(rt)))) then
    wait until ((not reg_locks(conv_integer(rs))) and
                (not reg_locks(conv_integer(rt))));
  end if;
  reg_rs <= reg(conv_integer(rs));
  reg_rt <= reg(conv_integer(rt));
  receive(inst_PC_in, PC);
  send(execute_op,op);
  send(inst_PC_out, PC);
  wait for delay(5,10);
  case op is
  when "000000" => -- ALU op
    send(execute_func,func,execute_rs,reg_rs,
         execute_rt,reg_rt);
    send(decode_to_wb,instr);
    receive(lock);
  when "000100" => -- beq
    send(execute_rs,reg_rs,execute_rt,reg_rt);
  when "100011" => -- lw
    send(execute_rs,reg_rs,execute_offset,offset);
```

```
      send(decode_to_wb,instr);
      receive(lock);
    when "101011" => -- sw
      send(execute_rs,reg_rs,execute_offset,offset,
           dmem_datain,reg_rt);
    when others => -- undefined
      assert false
        report "Illegal instruction"
        severity error;
    end case;
  end process;
  writeback:process
  begin
    receive(decode_to_wb,wb_instr);
    case wb_op is
    when "000000" => -- ALU op
      reg_locks(conv_integer(wb_rd)) <= true;
      wait for 1 ns;
      send(lock);
      receive(execute_rd,reg_rd);
      reg(conv_integer(wb_rd)) <= reg_rd;
      wait for delay(5,10);
      reg_locks(conv_integer(wb_rd)) <= false;
      wait for delay(5,10);
    when "100011" => -- lw
      reg_locks(conv_integer(wb_rt)) <= true;
      wait for 1 ns;
      send(lock);
      receive(dmem_dataout,reg_rd);
      reg(conv_integer(wb_rt)) <= reg_rd;
      wait for delay(5,10);
      reg_locks(conv_integer(wb_rt)) <= false;
      wait for delay(5,10);
    when others => -- undefined
      wait for delay(5,10);
    end case;
  end process;
  end behavior;

  ----------------------
  -- execute.vhd
  ----------------------
  library ieee;
  use ieee.std_logic_1164.all;
  use ieee.std_logic_arith.all;
  use ieee.std_logic_unsigned.all;
  use work.nondeterminism.all;
  use work.channel.all;
  entity execute is
```

```
  port(execute_op:inout channel:=init_channel;
       execute_rs:inout channel:=init_channel;
       execute_rt:inout channel:=init_channel;
       execute_rd:inout channel:=init_channel;
       execute_func:inout channel:=init_channel;
       execute_offset:inout channel:=init_channel;
       dmem_addr:inout channel:=init_channel;
       dmem_rw:inout channel:=init_channel;
       branch_decision:inout channel:=init_channel;
       inst_PC:inout channel:=init_channel);
end execute;
architecture behavior of execute is
  signal rs:std_logic_vector(31 downto 0);
  signal rt:std_logic_vector(31 downto 0);
  signal rd:std_logic_vector(31 downto 0);
  signal op:std_logic_vector(5 downto 0);
  signal func:std_logic_vector(5 downto 0);
  signal offset:std_logic_vector(15 downto 0);
  signal rw:std_logic;
  signal bd:std_logic;
  signal sign_ext:std_logic_vector(31 downto 0);
  signal PC:std_logic_vector(31 downto 0);
  signal EPC:std_logic_vector(31 downto 0);
  signal Cause:std_logic_vector(31 downto 0);
  signal overflow:std_logic_vector(31 downto 0);
begin
process
  variable addr_offset:std_logic_vector(31 downto 0);
begin
  receive(execute_op,op);
  receive(inst_PC, PC);
  case op is
  when "000000" => -- ALU op
    receive(execute_func,func,execute_rs,rs,
            execute_rt,rt);
    case func is
    when "100000" => -- add
      overflow <= rs + rt;
      wait for delay(5,10);
      if (rs(31) = '1' or rt(31) = '1') and
        overflow(31) = '0' then
        EPC <= PC - 1; --PC location of bad inst
        Cause <= X"c0000020"; --cause code for
                              --the undefined instruction
        assert false
          report "Exception:  Addition Overflow"
          severity error;
        assert false
          severity failure; -- kill the program
```

```
      end if;
      rd <= rs + rt;
    when "100010" => -- sub
      rd <= rs - rt;
      overflow <= rs + rt;
      wait for delay(5,10);
      if rs(31) = '0' and overflow(31) = '1' then
        EPC <= PC - 1; --PC location of bad inst
        Cause <= X"c0000020"; --cause code for
                              --the undefined instruction
        assert false
          report "Exception:  Subtraction Overflow"
          severity error;
        assert false
          severity failure; -- kill the program
      end if;
    when "100100" => -- and
      rd <= rs and rt;
    when "100101" => -- or
      rd <= rs or rt;
    when others =>
      EPC <= PC - 1; --PC location of bad inst
      Cause <= X"c0000000"; --cause code for
                            --the undefined instruction
      assert false
        report "Exception:  Invalid Instruction"
        severity error;
      assert false
        severity failure; -- kill the program
      --rd <= (others => 'X'); -- undefined
    end case;
    wait for delay(5,10);
    send(execute_rd,rd);
  when "000100" => -- beq
    receive(execute_rs,rs,execute_rt,rt);
    if (rs = rt) then bd <= '1';
    else bd <= '0';
    end if;
    wait for delay(5,10);
    send(branch_decision,bd);
  when "100011" => -- lw
    receive(execute_rs,rs,execute_offset,offset);
    addr_offset(31 downto 16):=(others => offset(15));
    addr_offset(15 downto 0):=offset;
    rd <= rs + addr_offset;
    rw <= '1';
    wait for delay(5,10);
    send(dmem_addr,rd);
    send(dmem_rw,rw);
```

```
  when "101011" => -- sw
    receive(execute_rs,rs,execute_offset,offset);
    addr_offset(31 downto 16):=(others => offset(15));
    addr_offset(15 downto 0):=offset;
    rd <= rs + addr_offset;
    rw <= '0';
    wait for delay(5,10);
    send(dmem_addr,rd);
    send(dmem_rw,rw);
  when others => -- undefined
    assert false
      report "Illegal instruction"
      severity error;
  end case;
  end process;
end behavior;
```

3

Communication Protocols

Problems

3.1 The patron has decided that he is going to choose which of the two shops to patronize, but first he will give that shop a call to tell them that he is coming. The chosen shop must then call the winery to request a bottle of wine to be delivered. The configuration is again as shown in Figure 3.4 using bundled data. The channel-level models of the *winery*, *shop*, and *patron* are shown below. Derive the handshaking expansion, and simulate until you are convinced that it is correct.

```
winery:process
begin
  if (probe(wine_to_new_shop)) then
    bottle <= selection(8,3);
    wait for delay(5,10);
    send(wine_to_new_shop,bottle);
  elsif (probe(wine_to_old_shop)) then
    bottle <= selection(8,3);
    wait for delay(5,10);
    send(wine_to_old_shop,bottle);
  else
    wait for delay(5,10);
  end if;
end process winery;

shop:process
```

```
begin
  if (probe(wine_selling)) then
     receive(wine_delivery,shelf);
     send(wine_selling,shelf);
  end if;
  wait for delay(1,2);
end process shop;

patron:process
variable z:integer;
begin
  z := selection(2);
  if (z = 1) then
    receive(old_shop_to_patron,bag);
    wine_drunk<=wine_type'val(conv_integer(bag));
  else
    receive(new_shop_to_patron,bag);
    wine_drunk<=wine_type'val(conv_integer(bag));
  end if;
  wait for delay(1,2);
end process patron;

-------------------------------------
-- This is the winery for problem 3.1
-------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.handshake.all;
entity winery is
  port(req_wine1:in std_logic:='0';
       ack_wine1:buffer std_logic:='0';
       bottle1:buffer std_logic_vector(2 downto 0);
       req_wine2:in std_logic:='0';
       ack_wine2:buffer std_logic:='0';
       bottle2:buffer std_logic_vector(2 downto 0));
end winery;
architecture behavioral of winery is
    signal bottle:std_logic_vector(2 downto 0);
  begin
  winery:process
    begin
      if(req_wine2 = '1') then
        bottle <= selection(8,3);
        wait for delay(5,10);
        bottle2 <= bottle;
        wait for delay(5,10);
        assign(ack_wine2,'1',5,10);
        guard(req_wine2,'0');
```

```
        assign(ack_wine2,'0',5,10);
      elsif(req_wine1 = '1') then
        bottle <= selection(8,3);
        wait for delay(5,10);
        bottle1 <= bottle;
        wait for delay(5,10);
        assign(ack_wine1,'1',5,10);
        guard(req_wine1,'0');
        assign(ack_wine1,'0',5,10);
      else
        wait for delay(5,10);
      end if;
    end process winery;
  end;


-----------------------------------
-- This is the shop for problem 3.1
-----------------------------------
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.handshake.all;
entity shop is
  port(req_wine:buffer std_logic:='0';
       ack_wine:in std_logic:='0';
       bottle:in std_logic_vector(2 downto 0);
       req_patron:in std_logic:='0';
       ack_patron:buffer std_logic:='0';
       shelf:buffer std_logic_vector(2 downto 0));  end shop;
architecture behavioral of shop is
  begin
  shop:process
    begin
      guard(req_patron,'1');
      assign(req_wine,'1',5,10);
      guard(ack_wine,'1');
      shelf <= bottle;
      wait for delay(5,10);
      assign(req_wine,'0',5,10);
      guard(ack_wine,'0');
      assign(ack_patron,'1',5,10);
      guard(req_patron,'0');
      assign(ack_patron,'0',5,10);
      wait for delay(5,10);
    end process shop;
  end;

-----------------------------------
```

```
-- This is the patron for problem 3.1
-------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.handshake.all;
entity patron is
  port(req_patron1:buffer std_logic:='0';
       ack_patron1:in std_logic:='0';
       shelf1:in std_logic_vector(2 downto 0);
       req_patron2:buffer std_logic:='0';
       ack_patron2:in std_logic:='0';
       shelf2:in std_logic_vector(2 downto 0));
end patron;
architecture behavioral of patron is
  signal bag:std_logic_vector(2 downto 0);
begin
  patron:process
    variable z:integer;
      begin
    z := selection(2);
    if(z=1) then
      assign(req_patron1,'1',5,10);
      guard(ack_patron1,'1');
      bag <= shelf1;
      wait for delay(5,10);
      assign(req_patron1,'0',5,10);
      guard(ack_patron1,'0');
    else
      assign(req_patron2,'1',5,10);
      guard(ack_patron2,'1');
      bag <= shelf2;
      wait for delay(5,10);
      assign(req_patron2,'0',5,10);
      guard(ack_patron2,'0');
    end if;
    wait for delay(5,10);
  end process patron;
end;

--------------------------------------
-- Top-Level Structure for Problem 3.1
--------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.handshake.all;
entity two_shop_struc is
end two_shop_struc;
```

```
architecture structural of two_shop_struc is
  component patron
  port(req_patron1:buffer std_logic:='0';
       ack_patron1:in std_logic:='0';
       shelf1:in std_logic_vector(2 downto 0);
       req_patron2:buffer std_logic:='0';
       ack_patron2:in std_logic:='0';
       shelf2:in std_logic_vector(2 downto 0));
  end component;
  component shop
  port(req_wine:buffer std_logic:='0';
       ack_wine:in std_logic:='0';
       bottle:in std_logic_vector(2 downto 0);
       req_patron:in std_logic:='0';
       ack_patron:buffer std_logic:='0';
       shelf:buffer std_logic_vector(2 downto 0));
  end component;
  component winery
  port(req_wine1:in std_logic:='0';
       ack_wine1:buffer std_logic:='0';
       bottle1:buffer std_logic_vector(2 downto 0);
       req_wine2:in std_logic:='0';
       ack_wine2:buffer std_logic:='0';
       bottle2:buffer std_logic_vector(2 downto 0));
  end component;
    signal s_req_wine1:std_logic;
  signal s_ack_wine1:std_logic;
  signal s_bottle1:std_logic_vector(2 downto 0);
  signal s_req_wine2:std_logic;
  signal s_ack_wine2:std_logic;
  signal s_bottle2:std_logic_vector(2 downto 0);
  signal s_req_patron1:std_logic;
  signal s_ack_patron1:std_logic;
  signal s_shelf1:std_logic_vector(2 downto 0);
  signal s_req_patron2:std_logic;
  signal s_ack_patron2:std_logic;
  signal s_shelf2:std_logic_vector(2 downto 0);
begin
  THE_WINERY:winery
    port map(req_wine1 => s_req_wine1,
             ack_wine1 => s_ack_wine1,
             bottle1 => s_bottle1,
             req_wine2 => s_req_wine2,
             ack_wine2 => s_ack_wine2,
             bottle2 => s_bottle2);
  OLD_SHOP:shop
    port map(req_wine => s_req_wine1,
             ack_wine => s_ack_wine1,
             bottle => s_bottle1,
```

```
            req_patron => s_req_patron1,
            ack_patron => s_ack_patron1,
            shelf => s_shelf1);
  NEW_SHOP:shop
    port map(req_wine => s_req_wine2,
            ack_wine => s_ack_wine2,
            bottle => s_bottle2,
            req_patron => s_req_patron2,
            ack_patron => s_ack_patron2,
            shelf => s_shelf2);
  THE_PATRON:patron
    port map(req_patron1 => s_req_patron1,
            ack_patron1 => s_ack_patron1,
            shelf1 => s_shelf1,
            req_patron2 => s_req_patron2,
            ack_patron2 => s_ack_patron2,
            shelf2 => s_shelf2);
end structural;
```

3.2 Find two legal reshufflings for the handshaking expansion from Problem 3.1. Simulate until you are convinced that they are correct.

NOTE: Two solutions are given below though many others exist. Any protocol which keeps the order of the 4-phase handshake intact as well as the order of the guard on ackwine high before ackpatron high is okay.

```
-- This is a reshuffled shop for problem 3.2
architecture behavioral of shop is
  begin
  shop:process
  begin
    guard(req_patron,'1');
    assign(req_wine,'1',5,10);
    guard(ack_wine,'1');
    shelf <= bottle;
    wait for delay(5,10);
    assign(ack_patron,'1',5,10);
    assign(req_wine,'0',5,10);
    guard(ack_wine,'0');
    guard(req_patron,'0');
    assign(ack_patron,'0',5,10);
    wait for delay(5,10);
  end process shop;
end;
```

```
-- This is a reshuffled shop for problem 3.2
architecture behavioral of shop is
  begin
  shop:process
  begin
```

```
    guard(req_patron,'1');
    assign(req_wine,'1',5,10);
    guard(ack_wine,'1');
    shelf <= bottle;
    wait for delay(5,10);
    assign(ack_patron,'1',5,10);
    guard(req_patron,'0');
    assign(ack_patron,'0',5,10);
    assign(req_wine,'0',5,10);
    guard(ack_wine,'0');
    wait for delay(5,10);
  end process shop;
end;
```

3.3 Perform handshaking expansion on the simple fetch process shown below using bundled data.

```
process
begin
  if (probe(increment)) then
    send(pcadd,'1');
    receive(pcadd_result,pc);
    send(imem_address,pc);
    receive(imem_data,instr);
    send(instr_decode,instr);
    receive(increment);
  elsif (probe(jump)) then
    send(pcadd,offset);
    receive(pcadd_result,pc);
    send(imem_address,pc);
    receive(imem_data,instr);
    send(instr_decode,instr);
    receive(jump);
  end if;
end process;
```

```
process
begin
  if (increment_snd='1') then
    -- send(pcadd,'1');
    --@synthesis_off
    pcadd_data <= '1';
    --@synthesis_on
    wait for 5 ns;
    assign(pcadd_snd,'1',1,3);
    guard(pcadd_rv,'1');
    assign(pcadd_snd,'0',1,3);
    guard(pcadd_rv,'0');
    -- receive(pcadd_result,pc);
    guard(pcadd_result_snd,'1');
```

```
      --@synthesis_off
      pc <= pcadd_result_data;
      wait for 5 ns;
      --@synthesis_on
      assign(pcadd_result_rv,'1',1,3);
      guard(pcadd_result_snd,'0');
      assign(pcadd_result_rv,'0',1,3);
      -- send(imem_address,pc);
      --@synthesis_off
      imem_address_data <= pc;
      --@synthesis_on
      wait for 5 ns;
      assign(imem_address_snd,'1',1,3);
      guard(imem_address_rv,'1');
      assign(imem_address_snd,'0',1,3);
      guard(imem_address_rv,'0');
      -- receive(imem_data,instr);
      guard(imem_data_snd,'1');
      --@synthesis_off
      instr <= imem_data_data;
      wait for 5 ns;
      --@synthesis_on
      assign(imem_data_rv,'1',1,3);
      guard(imem_data_snd,'0');
      assign(imem_data_rv,'0',1,3);
      -- send(instr_decode,instr);
      --@synthesis_off
      instr_decode_data <= instr;
      --@synthesis_on
      wait for 5 ns;
      assign(instr_decode_snd,'1',1,3);
      guard(instr_decode_rv,'1');
      assign(instr_decode_snd,'0',1,3);
      guard(instr_decode_rv,'0');
      -- receive(increment);
      guard(increment_snd,'1');
      assign(increment_rv,'1',1,3);
      guard(increment_snd,'0');
      assign(increment_rv,'0',1,3);
    elsif (jump_snd='1') then
      -- send(pcadd,offset);
      --@synthesis_off
      pcadd_data <= offset;
      --@synthesis_on
      wait for 5 ns;
      assign(pcadd_snd,'1',1,3);
      guard(pcadd_rv,'1');
      assign(pcadd_snd,'0',1,3);
      guard(pcadd_rv,'0');
```

```
    -- receive(pcadd_result,pc);
    guard(pcadd_result_snd,'1');
    --@synthesis_off
    pc <= pcadd_result_data;
    wait for 5 ns;
    --@synthesis_on
    assign(pcadd_result_rv,'1',1,3);
    guard(pcadd_result_snd,'0');
    assign(pcadd_result_rv,'0',1,3);
    -- send(imem_address,pc);
    --@synthesis_off
    imem_address_data <= pc;
    --@synthesis_on
    wait for 5 ns;
    assign(imem_address_snd,'1',1,3);
    guard(imem_address_rv,'1');
    assign(imem_address_snd,'0',1,3);
    guard(imem_address_rv,'0');
    -- receive(imem_data,instr);
    guard(imem_data_snd,'1');
    --@synthesis_off
    instr <= imem_data_data;
    wait for 5 ns;
    --@synthesis_on
    assign(imem_data_rv,'1',1,3);
    guard(imem_data_snd,'0');
    assign(imem_data_rv,'0',1,3);
    -- send(instr_decode,instr);
    --@synthesis_off
    instr_decode_data <= instr;
    --@synthesis_on
    wait for 5 ns;
    assign(instr_decode_snd,'1',1,3);
    guard(instr_decode_rv,'1');
    assign(instr_decode_snd,'0',1,3);
    guard(instr_decode_rv,'0');
    -- receive(jump);
    guard(jump_snd,'1');
    assign(jump_rv,'1',1,3);
    guard(jump_snd,'0');
    assign(jump_rv,'0',1,3);
  end if;
end process;
```

3.4 Find four legal reshufflings of the handshaking expansion from Problem 3.3.

Many possibilities. Any solution is fine as long as each handshake on a channel has its order of operations preserved. Also, all assign and guards to high values should remain in same order.

3.5 Perform handshaking expansion on your 4-bit adder design from Problem 2.1.1 using bundled data. Also, perform handshaking expansion on your environment processes. Simulate your design with its environment.

```
-- Adder for Problem 3.5
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.handshake.all;
use work.nondeterminism.all;
entity adder is
  port(X:in std_logic_vector(3 downto 0);
       Y:in std_logic_vector(3 downto 0);
       Cin:in std_logic_vector(0 downto 0);
       req_x:in std_logic;
       ack_x:buffer std_logic:='0';
       Sum:buffer std_logic_vector(3 downto 0):="0000";
       Cout:buffer std_logic_vector(0 downto 0):="0";
       req_s:buffer std_logic:='0';
       ack_s:in std_logic);
end entity adder;
architecture behavior of adder is
begin
process
variable t :  std_logic_vector(4 downto 0):="00000";
begin
  guard(req_x, '1');
  t := ("0" & X) + ("0" & Y) + ("0000" & Cin);
  assign(ack_x, '1', 1, 3);
  guard(req_x, '0');
  assign(ack_x, '0', 1, 3);
  Sum <= t(3 downto 0) after delay(1, 3);
  Cout <= t(4 downto 4) after delay(1, 3);
  wait for delay(4, 6);
  assign(req_s, '1', 1, 3);
  guard(ack_s, '1');
  assign(req_s, '0', 1, 3);
  guard(ack_s, '0');
end process;
end behavior;

-- Adder environment for Problem 3.5
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
```

```
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.handshake.all;
entity adder_env is
  port(X:buffer std_logic_vector(3 downto 0):="0000";
       Y:buffer std_logic_vector(3 downto 0):="0000";
       Cin:buffer std_logic_vector(0 downto 0):="0";
       req_x:buffer std_logic:='0';
       ack_x:in std_logic;
       Sum:in std_logic_vector(3 downto 0);
       Cout:in std_logic_vector(0 downto 0);
       req_s:in std_logic;
       ack_s:buffer std_logic:='0');
end entity adder_env;
architecture behavior of adder_env is
begin
process
variable t :  std_logic_vector(4 downto 0):="00000";
begin
  X <= selection(16,4);
  Y <= selection(16,4);
  Cin <= selection(2,1);
  wait for 5 ns;
  assign(req_x, '1', 1, 3);
  guard(ack_x, '1');
  assign(req_x, '0', 1, 3);
  guard(ack_x, '0');
  guard(req_s, '1');
  t := ("0" & X) + ("0" & Y) + ("0000" & Cin);
  assign(ack_s, '1', 1, 3);
  guard(req_s, '0');
  assign(ack_s, '0', 1, 3);
  assert (Sum=t(3 downto 0))
    report "wrong answer" severity warning;
  assert (Cout=t(4 downto 4))
    report "wrong answer" severity warning;
end process;
end behavior;

-- Top component for Problem 3.5
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity adder_top is
end entity adder_top;
architecture structure of adder_top is
  signal X:std_logic_vector(3 downto 0);
  signal Y:std_logic_vector(3 downto 0);
```

```
  signal Cin:std_logic_vector(0 downto 0);
  signal req_x:std_logic;
  signal ack_x:std_logic;
  signal Sum:std_logic_vector(3 downto 0);
  signal Cout:std_logic_vector(0 downto 0);
  signal req_s:std_logic;
  signal ack_s:std_logic;
begin
  THE_ADDER:entity work.adder(behavior)
    port map(X => X,Y => Y,Cin => Cin, req_x => req_x,
             ack_x => ack_x, Sum => Sum, Cout => Cout,
             req_s => req_s, ack_s => ack_s);
  THE_ADDER_ENV:entity work.adder_env(behavior)
    port map(X => X,Y => Y,Cin => Cin, req_x => req_x,
             ack_x => ack_x, Sum => Sum, Cout => Cout,
             req_s => req_s, ack_s => ack_s);
end structure;
```

3.6 Perform handshaking expansion on your 1-bit adder design from Problem 2.1.2. Use dual-rail for data encoding. Also, perform handshaking expansion on your environment processes. Simulate your design. Using these single-bit blocks, build a 4-bit adder with its environment and simulate.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.handshake.all;
entity adder1 is
  port(x0 :  in std_logic;
       x1 :  in std_logic;
       xack :  inout std_logic:='0';
       y0 :  in std_logic;
       y1 :  in std_logic;
       yack :  inout std_logic:='0';
       cin0 :  in std_logic;
       cin1 :  in std_logic;
       cinack :  inout std_logic:='0';
       sum0 :  inout std_logic:='0';
       sum1 :  inout std_logic:='0';
       sumack :  in std_logic;
       cout0 :  inout std_logic:='0';
       cout1 :  inout std_logic:='0';
       coutack :  in std_logic);
end entity adder1;
architecture behavior of adder1 is
begin
process
```

```
begin
  guard(sumack,'0');
  guard(coutack,'0');
  if (x0 /= '1' and x1 /= '1') then
      wait until x0 = '1' or x1 = '1';
  end if;
  if (y0 /= '1' and y1 /= '1') then
      wait until y0 = '1' or y1 = '1';
  end if;
  if (cin0 /= '1' and cin1 /= '1') then
      wait until cin0 = '1' or cin1 = '1';
  end if;
  sum1 <= (x0 and y0 and cin1) or
          (x0 and y1 and cin0) or
          (x1 and y0 and cin0) or
          (x1 and y1 and cin1) after delay(1,5);
  sum0 <= (x1 and y1 and cin0) or
          (x1 and y0 and cin1) or
          (x0 and y1 and cin1) or
          (x0 and y0 and cin0) after delay(1,5);
  cout1 <= (x1 and y1) or
           (x1 and cin1) or
           (y1 and cin1) after delay(1,5);
  cout0 <= (x0 and y0) or
           (x0 and cin0) or
           (y0 and cin0) after delay(1,5);
  wait until (sum1 = '1' or sum0 = '1') and
             (cout1 = '1' or cout0 = '1');
  xack <= '1' after delay(1,5);
  yack <= '1' after delay(1,5);
  cinack <= '1' after delay(1,5);
  wait until xack = '1' and yack = '1' and cinack = '1';
  guard(sumack,'1');
  guard(coutack,'1');
  sum1 <= '0' after delay(1,5);
  sum0 <= '0' after delay(1,5);
  cout1 <= '0' after delay(1,5);
  cout0 <= '0' after delay(1,5);
  wait until (sum1 = '0' and sum0 = '0' and
              cout1 = '0' and cout0 = '0');
  guard(x0,'0');
  guard(x1,'0');
  guard(y0,'0');
  guard(y1,'0');
  guard(cin0,'0');
  guard(cin1,'0');
  xack <= '0' after delay(1,5);
  yack <= '0' after delay(1,5);
  cinack <= '0' after delay(1,5);
```

```
  wait until xack = '0' and yack = '0' and cinack = '0';
end process;
end behavior;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.handshake.all;
entity adder_env4 is
  port(x0_0 :  inout std_logic:='0';
       x1_0 :  inout std_logic:='0';
       xack_0 :  in std_logic;
       y0_0 :  inout std_logic:='0';
       y1_0 :  inout std_logic:='0';
       yack_0 :  in std_logic;
       cin0_0 :  inout std_logic:='0';
       cin1_0 :  inout std_logic:='0';
       cinack_0 :  in std_logic;
       sum0_0 :  in std_logic;
       sum1_0 :  in std_logic;
       sumack_0 :  inout std_logic:='0';
       x0_1 :  inout std_logic:='0';
       x1_1 :  inout std_logic:='0';
       xack_1 :  in std_logic;
       y0_1 :  inout std_logic:='0';
       y1_1 :  inout std_logic:='0';
       yack_1 :  in std_logic;
       sum0_1 :  in std_logic;
       sum1_1 :  in std_logic;
       sumack_1 :  inout std_logic:='0';
       x0_2 :  inout std_logic:='0';
       x1_2 :  inout std_logic:='0';
       xack_2 :  in std_logic;
       y0_2 :  inout std_logic:='0';
       y1_2 :  inout std_logic:='0';
       yack_2 :  in std_logic;
       sum0_2 :  in std_logic;
       sum1_2 :  in std_logic;
       sumack_2 :  inout std_logic:='0';
       x0_3 :  inout std_logic:='0';
       x1_3 :  inout std_logic:='0';
       xack_3 :  in std_logic;
       y0_3 :  inout std_logic:='0';
       y1_3 :  inout std_logic:='0';
       yack_3 :  in std_logic;
       sum0_3 :  in std_logic;
       sum1_3 :  in std_logic;
```

```
        sumack_3 :  inout std_logic:='0';
        cout0_3 :  in std_logic;
        cout1_3 :  in std_logic;
        coutack_3 :  inout std_logic:='0');
end entity adder_env4;
architecture behavior of adder_env4 is
  signal x :  std_logic_vector(3 downto 0);
  signal y :  std_logic_vector(3 downto 0);
  signal c :  std_logic_vector(0 downto 0);
  signal s :  std_logic_vector(3 downto 0);
  signal d :  std_logic;
begin
process
begin
  x <= conv_std_logic_vector(selection(16),4);
  y <= conv_std_logic_vector(selection(16),4);
  c <= conv_std_logic_vector(selection(1),1);
  wait for 5 ns;
  x1_0 <= x(0) after delay(1,5);
  x1_1 <= x(1) after delay(1,5);
  x1_2 <= x(2) after delay(1,5);
  x1_3 <= x(3) after delay(1,5);
  x0_0 <= not x(0) after delay(1,5);
  x0_1 <= not x(1) after delay(1,5);
  x0_2 <= not x(2) after delay(1,5);
  x0_3 <= not x(3) after delay(1,5);
  y1_0 <= y(0) after delay(1,5);
  y1_1 <= y(1) after delay(1,5);
  y1_2 <= y(2) after delay(1,5);
  y1_3 <= y(3) after delay(1,5);
  y0_0 <= not y(0) after delay(1,5);
  y0_1 <= not y(1) after delay(1,5);
  y0_2 <= not y(2) after delay(1,5);
  y0_3 <= not y(3) after delay(1,5);
  cin1_0 <= c(0) after delay(1,5);
  cin0_0 <= not c(0) after delay(1,5);
  guard(xack_0,'1');
  guard(xack_1,'1');
  guard(xack_2,'1');
  guard(xack_3,'1');
  guard(yack_0,'1');
  guard(yack_1,'1');
  guard(yack_2,'1');
  guard(yack_3,'1');
  guard(cinack_0,'1');
  x1_0 <= '0' after delay(1,5);
  x1_1 <= '0' after delay(1,5);
  x1_2 <= '0' after delay(1,5);
  x1_3 <= '0' after delay(1,5);
```

```
  x0_0 <= '0' after delay(1,5);
  x0_1 <= '0' after delay(1,5);
  x0_2 <= '0' after delay(1,5);
  x0_3 <= '0' after delay(1,5);
  y1_0 <= '0' after delay(1,5);
  y1_1 <= '0' after delay(1,5);
  y1_2 <= '0' after delay(1,5);
  y1_3 <= '0' after delay(1,5);
  y0_0 <= '0' after delay(1,5);
  y0_1 <= '0' after delay(1,5);
  y0_2 <= '0' after delay(1,5);
  y0_3 <= '0' after delay(1,5);
  cin1_0 <= '0' after delay(1,5);
  cin0_0 <= '0' after delay(1,5);
  guard(xack_0,'0');
  guard(xack_1,'0');
  guard(xack_2,'0');
  guard(xack_3,'0');
  guard(yack_0,'0');
  guard(yack_1,'0');
  guard(yack_2,'0');
  guard(yack_3,'0');
  guard(cinack_0,'0');
end process;
process
begin
  if ((sum0_0 /= '1' and sum1_0 /= '1') or
      (sum0_1 /= '1' and sum1_1 /= '1') or
      (sum0_2 /= '1' and sum1_2 /= '1') or
      (sum0_3 /= '1' and sum1_3 /= '1') or
      (cout0_3 /= '1' and cout1_3 /= '1')) then
    wait until (sum0_0 = '1' or sum1_0 = '1') and
                (sum0_1 = '1' or sum1_1 = '1') and
                (sum0_2 = '1' or sum1_2 = '1') and
                (sum0_3 = '1' or sum1_3 = '1') and
                (cout0_3 = '1' or cout1_3 = '1');
  end if;
  s <= sum1_3 & sum1_2 & sum1_1 & sum1_0;
  d <= cout1_3;
  wait for 5 ns;
  sumack_0 <= '1' after delay(1,5);
  sumack_1 <= '1' after delay(1,5);
  sumack_2 <= '1' after delay(1,5);
  sumack_3 <= '1' after delay(1,5);
  coutack_3 <= '1' after delay(1,5);
  wait until sumack_0 = '1' and sumack_1 = '1' and
              sumack_2 = '1' and sumack_3 = '1' and
              coutack_3 = '1';
  if ((sum0_0 /= '0' or sum1_0 /= '0') or
```

```
      (sum0_1 /= '0' or sum1_1 /= '0') or
      (sum0_2 /= '0' or sum1_2 /= '0') or
      (sum0_3 /= '0' or sum1_3 /= '0') or
      (cout0_3 /= '0' or cout1_3 /= '0')) then
    wait until (sum0_0 = '0' and sum1_0 = '0') and
               (sum0_1 = '0' and sum1_1 = '0') and
               (sum0_2 = '0' and sum1_2 = '0') and
               (sum0_3 = '0' and sum1_3 = '0') and
               (cout0_3 = '0' and cout1_3 = '0');
  end if;
  sumack_0 <= '0' after delay(1,5);
  sumack_1 <= '0' after delay(1,5);
  sumack_2 <= '0' after delay(1,5);
  sumack_3 <= '0' after delay(1,5);
  coutack_3 <= '0' after delay(1,5);
  wait until sumack_0 = '0' and sumack_1 = '0' and
             sumack_2 = '0' and sumack_3 = '0' and
             coutack_3 = '0';
end process;
end behavior;

-- Entity/architecture declarations
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity adder4 is
end adder4;
architecture adder4 of adder4 is
    -- Component declarations
    component adder_env4
        port(
            cout1_3 :  in std_logic;
            cout0_3 :  in std_logic;
            sum1_3 :  in std_logic;
            sum0_3 :  in std_logic;
            yack_3 :  in std_logic;
            xack_3 :  in std_logic;
            sum1_2 :  in std_logic;
            sum0_2 :  in std_logic;
            yack_2 :  in std_logic;
            xack_2 :  in std_logic;
            sum1_1 :  in std_logic;
            sum0_1 :  in std_logic;
            yack_1 :  in std_logic;
            xack_1 :  in std_logic;
            sum1_0 :  in std_logic;
            sum0_0 :  in std_logic;
            cinack_0 :  in std_logic;
            yack_0 :  in std_logic;
```

```
            xack_0 :  in std_logic;
            y1_0 :  inout std_logic;
            y0_0 :  inout std_logic;
            x1_0 :  inout std_logic;
            x0_0 :  inout std_logic;
            coutack_3 :  inout std_logic;
            sumack_3 :  inout std_logic;
            y1_3 :  inout std_logic;
            y0_3 :  inout std_logic;
            x1_3 :  inout std_logic;
            x0_3 :  inout std_logic;
            sumack_2 :  inout std_logic;
            y1_2 :  inout std_logic;
            y0_2 :  inout std_logic;
            x1_2 :  inout std_logic;
            x0_2 :  inout std_logic;
            sumack_1 :  inout std_logic;
            y1_1 :  inout std_logic;
            y0_1 :  inout std_logic;
            x1_1 :  inout std_logic;
            x0_1 :  inout std_logic;
            sumack_0 :  inout std_logic;
            cin1_0 :  inout std_logic;
            cin0_0 :  inout std_logic
        );
    end component;
    component adder1
        port(
            coutack :  in std_logic;
            sumack :  in std_logic;
            cin1 :  in std_logic;
            cin0 :  in std_logic;
            y1 :  in std_logic;
            y0 :  in std_logic;
            x1 :  in std_logic;
            x0 :  in std_logic;
            cout0 :  inout std_logic;
            sum1 :  inout std_logic;
            sum0 :  inout std_logic;
            cinack :  inout std_logic;
            yack :  inout std_logic;
            xack :  inout std_logic;
            cout1 :  inout std_logic
        );
    end component;
    -- Signal declarations
    signal XSIG010001 :  std_logic;
    signal XSIG010002 :  std_logic;
    signal XSIG010003 :  std_logic;
```

```
    signal XSIG010004 :  std_logic;
    signal XSIG010005 :  std_logic;
    signal XSIG010006 :  std_logic;
    signal XSIG010007 :  std_logic;
    signal XSIG010008 :  std_logic;
    signal XSIG010009 :  std_logic;
    signal XSIG010010 :  std_logic;
    signal XSIG010011 :  std_logic;
    signal XSIG010012 :  std_logic;
    signal XSIG010013 :  std_logic;
    signal XSIG010014 :  std_logic;
    signal XSIG010015 :  std_logic;
    signal XSIG010016 :  std_logic;
    signal XSIG010017 :  std_logic;
    signal XSIG010018 :  std_logic;
    signal XSIG010019 :  std_logic;
    signal XSIG010020 :  std_logic;
    signal XSIG010021 :  std_logic;
    signal XSIG010022 :  std_logic;
    signal XSIG010023 :  std_logic;
    signal XSIG010024 :  std_logic;
    signal XSIG010025 :  std_logic;
    signal XSIG010026 :  std_logic;
    signal XSIG010027 :  std_logic;
    signal XSIG010028 :  std_logic;
    signal XSIG010029 :  std_logic;
    signal XSIG010030 :  std_logic;
    signal XSIG010031 :  std_logic;
    signal XSIG010032 :  std_logic;
    signal XSIG010033 :  std_logic;
    signal XSIG010034 :  std_logic;
    signal XSIG010035 :  std_logic;
    signal XSIG010036 :  std_logic;
    signal XSIG010037 :  std_logic;
    signal XSIG010038 :  std_logic;
    signal XSIG010039 :  std_logic;
    signal XSIG010040 :  std_logic;
    signal XSIG010041 :  std_logic;
    signal XSIG010042 :  std_logic;
    signal XSIG010043 :  std_logic;
    signal XSIG010044 :  std_logic;
    signal XSIG010045 :  std_logic;
    signal XSIG010046 :  std_logic;
    signal XSIG010047 :  std_logic;
    signal XSIG010048 :  std_logic;
    signal XSIG010052 :  std_logic;
    signal XSIG010053 :  std_logic;
    signal XSIG010054 :  std_logic;
begin
```

```
-- Signal assignments
-- Component instances
adder_env41 :  adder_env4
    port map(
        cout1_3 => XSIG010047,
        cout0_3 => XSIG010046,
        sum1_3 => XSIG010044,
        sum0_3 => XSIG010043,
        yack_3 => XSIG010042,
        xack_3 => XSIG010039,
        sum1_2 => XSIG010032,
        sum0_2 => XSIG010031,
        yack_2 => XSIG010030,
        xack_2 => XSIG010027,
        sum1_1 => XSIG010023,
        sum0_1 => XSIG010022,
        yack_1 => XSIG010021,
        xack_1 => XSIG010018,
        sum1_0 => XSIG010011,
        sum0_0 => XSIG010010,
        cinack_0 => XSIG010003,
        yack_0 => XSIG010009,
        xack_0 => XSIG010006,
        y1_0 => XSIG010008,
        y0_0 => XSIG010007,
        x1_0 => XSIG010005,
        x0_0 => XSIG010004,
        coutack_3 => XSIG010048,
        sumack_3 => XSIG010045,
        y1_3 => XSIG010041,
        y0_3 => XSIG010040,
        x1_3 => XSIG010038,
        x0_3 => XSIG010037,
        sumack_2 => XSIG010033,
        y1_2 => XSIG010029,
        y0_2 => XSIG010028,
        x1_2 => XSIG010026,
        x0_2 => XSIG010025,
        sumack_1 => XSIG010024,
        y1_1 => XSIG010020,
        y0_1 => XSIG010019,
        x1_1 => XSIG010017,
        x0_1 => XSIG010016,
        sumack_0 => XSIG010012,
        cin1_0 => XSIG010002,
        cin0_0 => XSIG010001
    );
adder11 :  adder1
    port map(
```

```
        coutack => XSIG010015,
        sumack => XSIG010012,
        cin1 => XSIG010002,
        cin0 => XSIG010001,
        y1 => XSIG010008,
        y0 => XSIG010007,
        x1 => XSIG010005,
        x0 => XSIG010004,
        cout0 => XSIG010013,
        sum1 => XSIG010011,
        sum0 => XSIG010010,
        cinack => XSIG010003,
        yack => XSIG010009,
        xack => XSIG010006,
        cout1 => XSIG010014
    );
adder12 :  adder1
    port map(
        coutack => XSIG010054,
        sumack => XSIG010024,
        cin1 => XSIG010014,
        cin0 => XSIG010013,
        y1 => XSIG010020,
        y0 => XSIG010019,
        x1 => XSIG010017,
        x0 => XSIG010016,
        cout0 => XSIG010052,
        sum1 => XSIG010023,
        sum0 => XSIG010022,
        cinack => XSIG010015,
        yack => XSIG010021,
        xack => XSIG010018,
        cout1 => XSIG010053
    );
adder13 :  adder1
    port map(
        coutack => XSIG010036,
        sumack => XSIG010033,
        cin1 => XSIG010053,
        cin0 => XSIG010052,
        y1 => XSIG010029,
        y0 => XSIG010028,
        x1 => XSIG010026,
        x0 => XSIG010025,
        cout0 => XSIG010034,
        sum1 => XSIG010032,
        sum0 => XSIG010031,
        cinack => XSIG010054,
        yack => XSIG010030,
```

```
            xack => XSIG010027,
            cout1 => XSIG010035
        );
    adder14 :  adder1
        port map(
            coutack => XSIG010048,
            sumack => XSIG010045,
            cin1 => XSIG010035,
            cin0 => XSIG010034,
            y1 => XSIG010041,
            y0 => XSIG010040,
            x1 => XSIG010038,
            x0 => XSIG010037,
            cout0 => XSIG010046,
            sum1 => XSIG010044,
            sum0 => XSIG010043,
            cinack => XSIG010036,
            yack => XSIG010042,
            xack => XSIG010039,
            cout1 => XSIG010047
        );
end adder4;
```

3.7 Perform handshaking expansion on your stack design from Problem 2.2 using bundled data. Create an environment and simulate a four-stage stack.

```
library ieee;
use ieee.std_logic_1164.all;
use work.handshake.all;
entity stack is
  port(
    -- push :  inout channel := init_channel;
    push_data :  in std_logic_vector(7 downto 0);
    push_rv :  inout std_logic:='0';
    push_snd :  in std_logic;
    -- pop :  inout channel := init_channel;
    pop_data :  out std_logic_vector(7 downto 0);
    pop_rv :  in std_logic;
    pop_snd :  inout std_logic:='0';
    -- put :  inout channel := init_channel;
    put_data :  out std_logic_vector(7 downto 0);
    put_rv :  in std_logic;
    put_snd :  inout std_logic:='0';
    -- get :  inout channel := init_channel;
    get_data :  in std_logic_vector(7 downto 0);
    get_rv :  inout std_logic:='0';
    get_snd :  in std_logic
  );
end stack;
```

```
architecture behavior of stack is
  signal data:std_logic_vector(7 downto 0);
  signal full:std_logic:='0';
begin
process
begin
  if (push_snd='1') then
    if (full='0') then
      -- receive(push,data);
      guard(push_snd,'1');
      --@synthesis_off
      data <= push_data;
      wait for 5 ns;
      --@synthesis_on
      assign(push_rv,'1',1,3);
      guard(push_snd,'0');
      assign(push_rv,'0',1,3);
      full <= '1';
      wait for 5 ns;
    else
      -- send(put,data);
      --@synthesis_off
      put_data <= data;
      --@synthesis_on
      wait for 5 ns;
      assign(put_snd,'1',1,3);
      guard(put_rv,'1');
      assign(put_snd,'0',1,3);
      guard(put_rv,'0');
      -- receive(push,data);
      guard(push_snd,'1');
      --@synthesis_off
      data <= push_data;
      wait for 5 ns;
      --@synthesis_on
      assign(push_rv,'1',1,3);
      guard(push_snd,'0');
      assign(push_rv,'0',1,3);
    end if;
  elsif (pop_rv='1') then
    if (full='0') then
      -- receive(get,data);
      assign(get_rv,'1',1,3);
      guard(get_snd,'1');
      --@synthesis_off
      data <= get_data;
      wait for 5 ns;
      --@synthesis_on
      assign(get_rv,'0',1,3);
```

```
      guard(get_snd,'0');
      -- send(pop,data);
      --@synthesis_off
      pop_data <= data;
      --@synthesis_on
      wait for 5 ns;
      assign(pop_snd,'1',1,3);
      guard(pop_rv,'0');
      assign(pop_snd,'0',1,3);
    else
      -- send(pop,data);
      --@synthesis_off
      pop_data <= data;
      --@synthesis_on
      wait for 5 ns;
      assign(pop_snd,'1',1,3);
      guard(pop_rv,'0');
      assign(pop_snd,'0',1,3);
      full <= '0';
      wait for 5 ns;
    end if;
  end if;
  wait for 5 ns;
end process;
end behavior;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.handshake.all;
entity stack_env is
  port(
    -- push :  inout channel := init_channel;
    push_data :  out std_logic_vector(7 downto 0);
    push_rv :  in std_logic;
    push_snd :  inout std_logic:='0';
    -- pop :  inout channel := init_channel;
    pop_data :  in std_logic_vector(7 downto 0);
    pop_rv :  inout std_logic:='0';
    pop_snd :  in std_logic
  );
end stack_env;
architecture behavior of stack_env is
  signal data:std_logic_vector(7 downto 0);
begin
process
  variable z:integer;
```

```
  variable entries:integer:=0;
begin
  if (entries=0) then
    data <= selection(255,8);
    wait for 5 ns;
    -- send(push,data);
    --@synthesis_off
    push_data <= data;
    --@synthesis_on
    wait for 5 ns;
    assign(push_snd,'1',1,3);
    guard(push_rv,'1');
    assign(push_snd,'0',1,3);
    guard(push_rv,'0');
    entries:=entries+1;
  elsif (entries=4) then
    -- receive(pop,data);
    assign(pop_rv,'1',1,3);
    guard(pop_snd,'1');
    --@synthesis_off
    data <= pop_data;
    wait for 5 ns;
    --@synthesis_on
    assign(pop_rv,'0',1,3);
    guard(pop_snd,'0');
    entries:=entries-1;
  else
    z:=selection(2);
    if (z=1) then
      data <= selection(255,8);
      wait for 5 ns;
      -- send(push,data);
      --@synthesis_off
      push_data <= data;
      --@synthesis_on
      wait for 5 ns;
      assign(push_snd,'1',1,3);
      guard(push_rv,'1');
      assign(push_snd,'0',1,3);
      guard(push_rv,'0');
      entries:=entries+1;
    else
      -- receive(pop,data);
      assign(pop_rv,'1',1,3);
      guard(pop_snd,'1');
      --@synthesis_off
      data <= pop_data;
      wait for 5 ns;
      --@synthesis_on
```

```
      assign(pop_rv,'0',1,3);
      guard(pop_snd,'0');
      entries:=entries-1;
    end if;
  end if;
  wait for 5 ns;
end process;
end behavior;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.handshake.all;
entity stack_top is
end stack_top;
architecture structure of stack_top is

  component stack
  port (
    push_data :  in std_logic_vector(7 downto 0);
    push_rv :  inout std_logic;
    push_snd :  in std_logic;
    pop_data :  out std_logic_vector(7 downto 0);
    pop_rv :  in std_logic;
    pop_snd :  inout std_logic;
    put_data :  out std_logic_vector(7 downto 0);
    put_rv :  in std_logic;
    put_snd :  inout std_logic;
    get_data :  in std_logic_vector(7 downto 0);
    get_rv :  inout std_logic;
    get_snd :  in std_logic
  );
  end component;
  component stack_env
  port (
    push_data :  out std_logic_vector(7 downto 0);
    push_rv :  in std_logic;
    push_snd :  inout std_logic;
    pop_data :  in std_logic_vector(7 downto 0);
    pop_rv :  inout std_logic;
    pop_snd :  in std_logic
  );
  end component;
  --signal Push:channel:=init_channel;
  signal Push_data :  std_logic_vector(7 downto 0);
  signal Push_rv :  std_logic;
  signal Push_snd :  std_logic;
  --signal Pop:channel:=init_channel;
```

```
  signal Pop_data :  std_logic_vector(7 downto 0);
  signal Pop_rv :  std_logic;
  signal Pop_snd :  std_logic;
  --signal Put0:channel:=init_channel;
  signal Put0_data :  std_logic_vector(7 downto 0);
  signal Put0_rv :  std_logic;
  signal Put0_snd :  std_logic;
  --signal Get0:channel:=init_channel;
  signal Get0_data :  std_logic_vector(7 downto 0);
  signal Get0_rv :  std_logic;
  signal Get0_snd :  std_logic;
  --signal Put1:channel:=init_channel;
  signal Put1_data :  std_logic_vector(7 downto 0);
  signal Put1_rv :  std_logic;
  signal Put1_snd :  std_logic;
  --signal Get1:channel:=init_channel;
  signal Get1_data :  std_logic_vector(7 downto 0);
  signal Get1_rv :  std_logic;
  signal Get1_snd :  std_logic;
  --signal Put2:channel:=init_channel;
  signal Put2_data :  std_logic_vector(7 downto 0);
  signal Put2_rv :  std_logic;
  signal Put2_snd :  std_logic;
  --signal Get2:channel:=init_channel;
  signal Get2_data :  std_logic_vector(7 downto 0);
  signal Get2_rv :  std_logic;
  signal Get2_snd :  std_logic;
  --signal Put3:channel:=init_channel;
  signal Put3_data :  std_logic_vector(7 downto 0);
  signal Put3_rv :  std_logic;
  signal Put3_snd :  std_logic;
  --signal Get3:channel:=init_channel;
  signal Get3_data :  std_logic_vector(7 downto 0);
  signal Get3_rv :  std_logic;
  signal Get3_snd :  std_logic;
begin
    STACK0 :  stack
  port map(
    Push_data =>Push_data,
    Push_rv =>Push_rv,
    Push_snd =>Push_snd,
    Pop_data =>Pop_data,
    Pop_rv =>Pop_rv,
    Pop_snd =>Pop_snd,
    Put_data =>Put0_data,
    Put_rv =>Put0_rv,
    Put_snd =>Put0_snd,
    Get_data =>Get0_data,
    Get_rv =>Get0_rv,
```

```
  Get_snd =>Get0_snd
);
STACK1 :  stack
port map(
  Push_data =>Put0_data,
  Push_rv =>Put0_rv,
  Push_snd =>Put0_snd,
  Pop_data =>Get0_data,
  Pop_rv =>Get0_rv,
  Pop_snd =>Get0_snd,
  Put_data =>Put1_data,
  Put_rv =>Put1_rv,
  Put_snd =>Put1_snd,
  Get_data =>Get1_data,
  Get_rv =>Get1_rv,
  Get_snd =>Get1_snd
);
STACK2 :  stack
port map(
  Push_data =>Put1_data,
  Push_rv =>Put1_rv,
  Push_snd =>Put1_snd,
  Pop_data =>Get1_data,
  Pop_rv =>Get1_rv,
  Pop_snd =>Get1_snd,
  Put_data =>Put2_data,
  Put_rv =>Put2_rv,
  Put_snd =>Put2_snd,
  Get_data =>Get2_data,
  Get_rv =>Get2_rv,
  Get_snd =>Get2_snd
);
STACK3 :  stack
port map(
  Push_data =>Put2_data,
  Push_rv =>Put2_rv,
  Push_snd =>Put2_snd,
  Pop_data =>Get2_data,
  Pop_rv =>Get2_rv,
  Pop_snd =>Get2_snd,
  Put_data =>Put3_data,
  Put_rv =>Put3_rv,
  Put_snd =>Put3_snd,
  Get_data =>Get3_data,
  Get_rv =>Get3_rv,
  Get_snd =>Get3_snd
);
THE_STACK_ENV : stack_env
port map(
```

```
    Push_data =>Push_data,
    Push_rv =>Push_rv,
    Push_snd =>Push_snd,
    Pop_data =>Pop_data,
    Pop_rv =>Pop_rv,
    Pop_snd =>Pop_snd
  );
end structure;
```

3.8 Perform handshaking expansion on your stack design from Problem 2.2 assuming only 1 bit of data encoded using dual-rail. Create an environment and simulate a four-stage stack.

```
library ieee;
use ieee.std_logic_1164.all;
use work.handshake.all;
entity stack is
  port(
    -- push :  inout channel := init_channel;
    push_rv :  inout std_logic:='0';
    push0 :  in std_logic;
    push1 :  in std_logic;
    -- pop :  inout channel := init_channel;
    pop_rv :  in std_logic;
    pop0 :  inout std_logic:='0';
    pop1 :  inout std_logic:='0';
    -- put :  inout channel := init_channel;
    put_rv :  in std_logic;
    put0 :  inout std_logic:='0';
    put1 :  inout std_logic:='0';
    -- get :  inout channel := init_channel;
    get_rv :  inout std_logic:='0';
    get0 :  in std_logic;
    get1 :  in std_logic
  );
end stack;
architecture behavior of stack is
  signal data:std_logic:='0';
  signal full:std_logic:='0';
begin
process
begin
  guard_or(push0,'1',push1,'1',pop_rv,'1');
  if (push0='1' or push1='1') then
    if (full='0') then
      -- receive(push,data);
      if (push0='1') then vassign(data,'0',1,3);
      else vassign(data,'1',1,3);
      end if;
```

```
        assign(push_rv,'1',1,3);
        guard_and(push0,'0',push1,'0');
        assign(push_rv,'0',1,3);
        assign(full,'1',1,3);
      else
        -- send(put,data);
        if (data='0') then assign(put0,'1',1,3);
        else assign(put1,'1',1,3);
        end if;
        guard(put_rv,'1');
        vassign(put0,'0',1,3,put1,'0',1,3);
        guard(put_rv,'0');
        -- receive(push,data);
        if (push0='1') then vassign(data,'0',1,3);
        else vassign(data,'1',1,3);
        end if;
        assign(push_rv,'1',1,3);
        guard_and(push0,'0',push1,'0');
        assign(push_rv,'0',1,3);
      end if;
    elsif (pop_rv='1') then
      if (full='0') then
        -- receive(get,data);
        assign(get_rv,'1',1,3);
        guard_or(get0,'1',get1,'1');
        if (get0='1') then vassign(data,'0',1,3);
        else vassign(data,'1',1,3);
        end if;
        assign(get_rv,'0',1,3);
        guard_and(get0,'0',get1,'0');
        -- send(pop,data);
        if (data='0') then assign(pop0,'1',1,3);
        else assign(pop1,'1',1,3);
        end if;
        guard(pop_rv,'0');
        vassign(pop0,'0',1,3,pop1,'0',1,3);
      else
        -- send(pop,data);
        if (data='0') then assign(pop0,'1',1,3);
        else assign(pop1,'1',1,3);
        end if;
        guard(pop_rv,'0');
        vassign(pop0,'0',1,3,pop1,'0',1,3);
        assign(full,'0',1,3);
      end if;
    end if;
    wait for 5 ns;
  end process;
  end behavior;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.handshake.all;
entity stack_env is
  port(
    -- push :  inout channel := init_channel;
    push_rv :  in std_logic;
    push0 :  inout std_logic:='0';
    push1 :  inout std_logic:='0';
    -- pop :  inout channel := init_channel;
    pop_rv :  inout std_logic:='0';
    pop0 :  in std_logic;
    pop1 :  in std_logic
  );
end stack_env;
architecture behavior of stack_env is
  signal data:std_logic;
begin
process
  variable z:integer;
  variable entries:integer:=0;
begin
  if (entries=0) then
    z:=selection(2);
    -- send(push,data);
    if (z=1) then
      assign(push0,'1',1,3);
    else
      assign(push1,'1',1,3);
    end if;
    guard(push_rv,'1');
    vassign(push0,'0',1,3,push1,'0',1,3);
    guard(push_rv,'0');
    entries:=entries+1;
  elsif (entries=4) then
    -- receive(pop,data);
    assign(pop_rv,'1',1,3);
    guard_or(pop0,'1',pop1,'1');
    if (pop0='1') then vassign(data,'0',1,3);
    else vassign(data,'1',1,3);
    end if;
    assign(pop_rv,'0',1,3);
    guard_and(pop0,'0',pop1,'0');
    entries:=entries-1;
  else
    z:=selection(2);
```

```
      if (z=1) then
        z:=selection(2);
        -- send(push,data);
        if (z=1) then
          assign(push0,'1',1,3);
        else
          assign(push1,'1',1,3);
        end if;
        guard(push_rv,'1');
        vassign(push0,'0',1,3,push1,'0',1,3);
        guard(push_rv,'0');
        entries:=entries+1;
      else
        -- receive(pop,data);
        assign(pop_rv,'1',1,3);
        guard_or(pop0,'1',pop1,'1');
        if (pop0='1') then vassign(data,'0',1,3);
        else vassign(data,'1',1,3);
        end if;
        assign(pop_rv,'0',1,3);
        guard_and(pop0,'0',pop1,'0');
        entries:=entries-1;
      end if;
    end if;
    wait for 5 ns;
  end process;
  end behavior;

  library ieee;
  use ieee.std_logic_1164.all;
  use ieee.std_logic_arith.all;
  use ieee.std_logic_unsigned.all;
  use work.handshake.all;
  entity stack_top is
  end stack_top;
  architecture structure of stack_top is

    component stack
    port (
      push_rv :  inout std_logic:='0';
      push0 :  in std_logic;
      push1 :  in std_logic;
      pop_rv :  in std_logic;
      pop0 :  inout std_logic:='0';
      pop1 :  inout std_logic:='0';
      put_rv :  in std_logic;
      put0 :  inout std_logic:='0';
      put1 :  inout std_logic:='0';
      get_rv :  inout std_logic:='0';
```

```
  get0 :  in std_logic;
  get1 :  in std_logic
);
end component;
component stack_env
port (
  push_rv :  in std_logic;
  push0 :  inout std_logic:='0';
  push1 :  inout std_logic:='0';
  pop_rv :  inout std_logic:='0';
  pop0 :  in std_logic;
  pop1 :  in std_logic
);
end component;
--signal Push:channel:=init_channel;
signal Push_rv :  std_logic;
signal Push0 :  std_logic;
signal Push1 :  std_logic;
--signal Pop:channel:=init_channel;
signal Pop_rv :  std_logic;
signal Pop0 :  std_logic;
signal Pop1 :  std_logic;
--signal Put0:channel:=init_channel;
signal Put0_rv :  std_logic;
signal Put00 :  std_logic;
signal Put01 :  std_logic;
--signal Get0:channel:=init_channel;
signal Get0_rv :  std_logic;
signal Get00 :  std_logic;
signal Get01 :  std_logic;
--signal Put1:channel:=init_channel;
signal Put1_rv :  std_logic;
signal Put10 :  std_logic;
signal Put11 :  std_logic;
--signal Get1:channel:=init_channel;
signal Get1_rv :  std_logic;
signal Get10 :  std_logic;
signal Get11 :  std_logic;
--signal Put2:channel:=init_channel;
signal Put2_rv :  std_logic;
signal Put20 :  std_logic;
signal Put21 :  std_logic;
--signal Get2:channel:=init_channel;
signal Get2_rv :  std_logic;
signal Get20 :  std_logic;
signal Get21 :  std_logic;
--signal Put3:channel:=init_channel;
signal Put3_rv :  std_logic;
signal Put30 :  std_logic;
```

```
  signal Put31 :  std_logic;
  --signal Get3:channel:=init_channel;
  signal Get3_rv :  std_logic;
  signal Get30 :  std_logic;
  signal Get31 :  std_logic;
begin
    STACK0 :  stack
  port map(
    Push_rv =>Push_rv,
    Push0 =>Push0,
    Push1 =>Push1,
    Pop_rv =>Pop_rv,
    Pop0 =>Pop0,
    Pop1 =>Pop1,
    Put_rv =>Put0_rv,
    Put0 =>Put00,
    Put1 =>Put01,
    Get_rv =>Get0_rv,
    Get0 =>Get00,
    Get1 =>Get01
  );
  STACK1 :  stack
  port map(
    Push_rv =>Put0_rv,
    Push0 =>Put00,
    Push1 =>Put01,
    Pop_rv =>Get0_rv,
    Pop0 =>Get00,
    Pop1 =>Get01,
    Put_rv =>Put1_rv,
    Put0 =>Put10,
    Put1 =>Put11,
    Get_rv =>Get1_rv,
    Get0 =>Get10,
    Get1 =>Get11
  );
  STACK2 :  stack
  port map(
    Push_rv =>Put1_rv,
    Push0 =>Put10,
    Push1 =>Put11,
    Pop_rv =>Get1_rv,
    Pop0 =>Get10,
    Pop1 =>Get11,
    Put_rv =>Put2_rv,
    Put0 =>Put20,
    Put1 =>Put21,
    Get_rv =>Get2_rv,
    Get0 =>Get20,
```

```
    Get1 =>Get21
  );
  STACK3 :  stack
  port map(
    Push_rv =>Put2_rv,
    Push0 =>Put20,
    Push1 =>Put21,
    Pop_rv =>Get2_rv,
    Pop0 =>Get20,
    Pop1 =>Get21,
    Put_rv =>Put3_rv,
    Put0 =>Put30,
    Put1 =>Put31,
    Get_rv =>Get3_rv,
    Get0 =>Get30,
    Get1 =>Get30
  );
  THE_STACK_ENV : stack_env
  port map(
    Push_rv =>Push_rv,
    Push0 =>Push0,
    Push1 =>Push1,
    Pop_rv =>Pop_rv,
    Pop0 =>Pop0,
    Pop1 =>Pop1
  );
end structure;
```

3.9 Perform handshaking expansion on your shifter design from Problem 2.3 using bundled data. Create an environment and simulate a four-element shifter.

```
------------------------------------------------
-- This is the standard 1 bit cell for a shifter
------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.handshake.all;
entity shift_base_cell is
  port(req_load:in std_logic;
       ack_load:buffer std_logic:='0';
       req_shift_in:in std_logic;
       ack_shift_in:buffer std_logic:='0';
       req_shift_out:buffer std_logic:='0';
       ack_shift_out:in std_logic;
       req_done_in:in std_logic;
       ack_done_in:buffer std_logic:='0';
       req_done_out:buffer std_logic;
```

```
        ack_done_out:in std_logic:='0';
        output:buffer std_logic;
        ld_bit:in std_logic;
        sbit:out std_logic;
        a_bit:in std_logic);
end shift_base_cell;
architecture behavior of shift_base_cell is
  signal the_bit:std_logic;
  signal done_bit:std_logic:='0';
  begin
  process
  begin
      if(req_load = '1') then
      wait for delay(5,10);
      the_bit <= ld_bit;
      wait for delay(5,10);
      assign(ack_load,'1',5,10);
      guard(req_load, '0');
      assign(ack_load,'0',5,10);
      done_bit <= '0';
      wait for delay(5,10);
    end if;
        if(req_shift_in = '1' or req_done_in = '1') then
      while(done_bit = '0') loop
        guard_or(req_shift_in,'1',req_done_in,'1');
        if(req_shift_in = '1') then
          assign(req_shift_out,'1',5,10);
          sbit <= the_bit;
          wait for delay(5,10);
          guard(ack_shift_out,'1');
          assign(req_shift_out,'0',5,10);
          guard(ack_shift_out,'0');
          the_bit <= a_bit;
          wait for delay(5,10);
          assign(ack_shift_in,'1',5,10);
          guard(req_shift_in,'0');
          assign(ack_shift_in,'0',5,10);
        else                  assign(ack_done_in,'1',5,10);
          guard(req_done_in,'0');
          done_bit <= '1';
          output <= the_bit;
          wait for delay(5,10);
          assign(req_done_out,'1',5,10);
          guard(ack_done_out,'1');
          assign(req_done_out,'0',5,10);
          guard(ack_done_out,'0');
          assign(ack_done_in,'0',5,10);
        end if;
        wait for delay(5,10);
```

```
      end loop;
    end if;
        wait for delay(5,10);
  end process;
end behavior;

-----------------------------------------
-- This is the end 1 bit cell for a shifter
-----------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.handshake.all;
entity shift_end_cell is
  port(req_load:in std_logic;
       ack_load:buffer std_logic:='0';
       req_shift_in:in std_logic;
       ack_shift_in:buffer std_logic:='0';
       req_done_in:in std_logic;
       ack_done_in:buffer std_logic:='0';
       output:buffer std_logic;
       ld_bit:in std_logic;
       a_bit:in std_logic);
end shift_end_cell;
architecture behavior of shift_end_cell is
  signal the_bit:std_logic;
  signal done_bit:std_logic:='0';
  begin
  process
  begin
        if(req_load = '1') then
      wait for delay(5,10);
      the_bit <= ld_bit;
      wait for delay(5,10);
      assign(ack_load,'1',5,10);
      guard(req_load, '0');
      assign(ack_load,'0',5,10);
      done_bit <= '0';
      wait for delay(5,10);
    end if;
    if(req_shift_in = '1' or req_done_in = '1') then
      while(done_bit = '0') loop
        guard_or(req_shift_in,'1',req_done_in,'1');
        if(req_shift_in = '1') then
          wait for delay(5,10);
          the_bit <= a_bit;
          wait for delay(5,10);
          assign(ack_shift_in,'1',5,10);
          guard(req_shift_in,'0');
```

```
          assign(ack_shift_in,'0',5,10);
        else
          assign(ack_done_in,'1',5,10);
          guard(req_done_in,'0');
          assign(ack_done_in,'0',5,10);
          done_bit <= '1';
          output <= the_bit;
          wait for delay(5,10);
        end if;
        wait for delay(5,10);
      end loop;
    end if;
    wait for delay(5,10);
  end process;
end behavior;

---------------------------------
-- 4-bit Shifter for Problem 3.9
---------------------------------
library ieee;
use ieee.std_logic_1164.all;
entity four_bit_shifter is
  port(req_main_shift_in:in std_logic:='0';
       ack_main_shift_in:buffer std_logic:='0';
       req_main_done_in:in std_logic:='0';
       ack_main_done_in:buffer std_logic:='0';
       req_load0:in std_logic:='0';
       ack_load0:buffer std_logic:='0';
       req_load1:in std_logic:='0';
       ack_load1:buffer std_logic:='0';
       req_load2:in std_logic:='0';
       ack_load2:buffer std_logic:='0';
       req_load3:in std_logic:='0';
       ack_load3:buffer std_logic:='0';
       out0:buffer std_logic;
       out1:buffer std_logic;
       out2:buffer std_logic;
       out3:buffer std_logic;
       in0:in std_logic;
       in1:in std_logic;
       in2:in std_logic;
       in3:in std_logic;
       input_bit:in std_logic);
end four_bit_shifter;
architecture structure of four_bit_shifter is
  component shift_base_cell
    port(req_load:in std_logic;
       ack_load:buffer std_logic;
       req_shift_in:in std_logic;
```

```
    ack_shift_in:buffer std_logic;
    req_shift_out:buffer std_logic;
    ack_shift_out:in std_logic;
    req_done_in:in std_logic;
    ack_done_in:buffer std_logic;
    req_done_out:buffer std_logic;
    ack_done_out:in std_logic;
    output:buffer std_logic;
    ld_bit:in std_logic;
    sbit:out std_logic;
    a_bit:in std_logic);
end component;
component shift_end_cell
  port(req_load:in std_logic;
    ack_load:buffer std_logic;
    req_shift_in:in std_logic;
    ack_shift_in:buffer std_logic;
    req_done_in:in std_logic;
    ack_done_in:buffer std_logic;
    output:buffer std_logic;
    ld_bit:in std_logic;
    a_bit:in std_logic);
end component;
signal req_s_com01:std_logic;
signal ack_s_com01:std_logic;
signal req_s_com12:std_logic;
signal ack_s_com12:std_logic;
signal req_s_com23:std_logic;
signal ack_s_com23:std_logic;
signal req_d_com01:std_logic;
signal ack_d_com01:std_logic;
signal req_d_com12:std_logic;
signal ack_d_com12:std_logic;
signal req_d_com23:std_logic;
signal ack_d_com23:std_logic;
signal bit01:std_logic;
signal bit12:std_logic;
signal bit23:std_logic;
signal ld0:std_logic;
signal ld1:std_logic;
signal ld2:std_logic;
signal ld3:std_logic;
begin
BIT_ZERO:shift_base_cell
  port map(req_shift_in => req_main_shift_in,
           ack_shift_in => ack_main_shift_in,
           req_shift_out => req_s_com01,
           ack_shift_out => ack_s_com01,
           req_load => req_load0,
```

```
            ack_load => ack_load0,
            req_done_in => req_main_done_in,
            ack_done_in => ack_main_done_in,
            req_done_out => req_d_com01,
            ack_done_out => ack_d_com01,
            output => out0,
            ld_bit => in0,
            sbit => bit01,
            a_bit => input_bit);
    BIT_ONE:shift_base_cell
      port map(req_shift_in => req_s_com01,
            ack_shift_in => ack_s_com01,
            req_shift_out  => req_s_com12,
            ack_shift_out  => ack_s_com12,
            req_load => req_load1,
            ack_load => ack_load1,
            req_done_in => req_d_com01,
            ack_done_in => ack_d_com01,
            req_done_out => req_d_com12,
            ack_done_out => ack_d_com12,
            output => out1,
            ld_bit => in1,
            sbit => bit12,
            a_bit => bit01);
    BIT_TWO:shift_base_cell
      port map(req_shift_in  => req_s_com12,
            ack_shift_in  => ack_s_com12,
            req_shift_out => req_s_com23,
            ack_shift_out => ack_s_com23,
            req_load => req_load2,
            ack_load => ack_load2,
            req_done_in => req_d_com12,
            ack_done_in => ack_d_com12,
            req_done_out => req_d_com23,
            ack_done_out => ack_d_com23,
            output => out2,
            ld_bit => in2,
            sbit => bit23,
            a_bit => bit12);
    BIT_THREE:shift_end_cell
      port map(req_shift_in => req_s_com23,
            ack_shift_in => ack_s_com23,
            req_load => req_load3,
            ack_load => ack_load3,
            req_done_in => req_d_com23,
            ack_done_in => ack_d_com23,
            output => out3,
            ld_bit => in3,
            a_bit => bit23);
```

```
end structure;

------------------------------
-- Test bench for Problem 3.9
------------------------------
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.handshake.all;
entity env_struc is
  port(req_done:inout std_logic:='0';
       ack_done:buffer std_logic:='0';
       req_shift:inout std_logic:='0';
       ack_shift:buffer std_logic:='0';
       req_load0_tl:inout std_logic:='0';
       ack_load0_tl:buffer std_logic:='0';
       req_load1_tl:inout std_logic:='0';
       ack_load1_tl:buffer std_logic:='0';
       req_load2_tl:inout std_logic:='0';
       ack_load2_tl:buffer std_logic:='0';
       req_load3_tl:inout std_logic:='0';
       ack_load3_tl:buffer std_logic:='0';
       out0_tl:buffer std_logic;
       out1_tl:buffer std_logic;
       out2_tl:buffer std_logic;
       out3_tl:buffer std_logic;
       ld0:inout std_logic;
       ld1:inout std_logic;
       ld2:inout std_logic;
       ld3:inout std_logic;
       the_in_bit:inout std_logic);
end env_struc;
architecture testing of env_struc is
  component four_bit_shifter
  port(req_main_shift_in:inout std_logic;
       ack_main_shift_in:buffer std_logic;
       req_main_done_in:inout std_logic;
       ack_main_done_in:buffer std_logic;
       req_load0:inout std_logic;
       ack_load0:buffer std_logic;
       req_load1:inout std_logic;
       ack_load1:buffer std_logic;
       req_load2:inout std_logic;
       ack_load2:buffer std_logic;
       req_load3:inout std_logic;
       ack_load3:buffer std_logic;
       out0:buffer std_logic;
       out1:buffer std_logic;
       out2:buffer std_logic;
```

```
      out3:buffer std_logic;
      in0:inout std_logic;
      in1:inout std_logic;
      in2:inout std_logic;
      in3:inout std_logic;
      input_bit:inout std_logic);
end component;
signal ld0_bit:std_logic;
signal ld1_bit:std_logic;
signal ld2_bit:std_logic;
signal ld3_bit:std_logic;
signal ld0_out:std_logic;
signal ld1_out:std_logic;
signal ld2_out:std_logic;
signal ld3_out:std_logic;
signal done_or_shift:std_logic:='0';
  begin
shifter:four_bit_shifter
  port map(req_main_shift_in => req_shift,
           ack_main_shift_in => ack_shift,
           req_main_done_in => req_done,
           ack_main_done_in => ack_done,
           req_load0 => req_load0_tl,
           ack_load0 => ack_load0_tl,
           req_load1 => req_load1_tl,
           ack_load1 => ack_load1_tl,
           req_load2 => req_load2_tl,
           ack_load2 => ack_load2_tl,
           req_load3 => req_load3_tl,
           ack_load3 => ack_load3_tl,
           out0 => out0_tl,
           out1 => out1_tl,
           out2 => out2_tl,
           out3 => out3_tl,
           in0 => ld0,
           in1 => ld1,
           in2 => ld2,
           in3 => ld3,
           input_bit => the_in_bit);
process
  variable temp:integer;
begin
  --randomly choose a value to load into the shifter
  temp:=selection(2);
  case temp is
    when 1 =>
      ld0_bit <= '0';
    when others =>
      ld0_bit <= '1';
```

```
end case;
temp:=selection(2);
case temp is
  when 1 =>
    ld1_bit <= '0';
  when others =>
    ld1_bit <= '1';
end case;
temp:=selection(2);
case temp is
  when 1 =>
    ld2_bit <= '0';
  when others =>
    ld2_bit <= '1';
end case;
temp:=selection(2);
case temp is
  when 1 =>
    ld3_bit <= '0';
  when others =>
    ld3_bit <= '1';
end case;
wait for delay(5,10);
--load the shifter
assign(req_load0_tl,'1',5,10);
    ld0 <= ld0_bit;
wait for delay(5,10);
guard(ack_load0_tl,'1');
assign(req_load0_tl,'0',5,10);
guard(ack_load0_tl,'0');
assign(req_load1_tl,'1',5,10);
    ld1 <= ld1_bit;
wait for delay(5,10);
guard(ack_load1_tl,'1');
assign(req_load1_tl,'0',5,10);
guard(ack_load1_tl,'0');
assign(req_load2_tl,'1',5,10);
    ld2 <= ld2_bit;
wait for delay(5,10);
guard(ack_load2_tl,'1');
assign(req_load2_tl,'0',5,10);
guard(ack_load2_tl,'0');
assign(req_load3_tl,'1',5,10);
    ld3 <= ld3_bit;
wait for delay(5,10);
guard(ack_load3_tl,'1');
assign(req_load3_tl,'0',5,10);
guard(ack_load3_tl,'0');
--randomly choose to send a done or shift signal
```

```
temp:=selection(2);
case temp is
  when 1 =>
    done_or_shift <= '0';
  when others =>
    done_or_shift <= '1';
end case;
wait for delay(5,10);
    while(done_or_shift = '1')
loop
  ld3_bit <= ld2_bit;
  ld2_bit <= ld1_bit;
  ld1_bit <= ld0_bit;
  --randomly pick a bit to shift in
  temp:=selection(2);
  case temp is
    when 1 =>
      ld0_bit <= '0';
    when others =>
      ld0_bit <= '1';
  end case;
  wait for delay(5,10);
  assign(req_shift,'1',5,10);
  the_in_bit <= ld0_bit;
  wait for delay(5,10);
  guard(ack_shift,'1');
  assign(req_shift,'0',5,10);
  guard(ack_shift,'0');
        --randomly choose done or shift before checking the
  --loop condition again
  temp:=selection(2);
  case temp is
    when 1 =>
      done_or_shift <= '0';
    when others =>
      done_or_shift <= '1';
  end case;
  wait for delay(5,10);
end loop;
    --send the done signal and then check the
--output against the expected output
assign(req_done,'1',5,10);
guard(ack_done,'1');
assign(req_done,'0',5,10);
guard(ack_done,'0');
    wait for delay(5,10);
ld0_out <= out0_tl;
ld1_out <= out1_tl;
ld2_out <= out2_tl;
```

```
    ld3_out <= out3_tl;
    wait for delay(5,10);
        assert ld0_bit = ld0_out
      report "The 0 bit does not match"
      severity error;
    assert ld1_bit = ld1_out
      report "The 1 bit does not match"
      severity error;
    assert ld2_bit = ld2_out
      report "The 2 bit does not match"
      severity error;
    assert ld3_bit = ld3_out
      report "The 3 bit does not match"
      severity error;
      end process;
end;
```

3.10 Perform handshaking expansion on your shifter design from Problem 2.3 using dual-rail data encoding. Create an environment and simulate a four-element shifter.

```
-------------------------------------------------
-- This is the standard 1 bit cell for a shifter
-------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.handshake.all;
entity shift_base_cell is
  port(load0:in std_logic;
       load1:in std_logic;
       ack_load:buffer std_logic:='0';
       shift_in0:in std_logic;
       shift_in1:in std_logic;
       ack_shift_in:buffer std_logic:='0';
       shift_out0:buffer std_logic:='0';
       shift_out1:buffer std_logic:='0';
       ack_shift_out:in std_logic;
       req_done_in:in std_logic;
       ack_done_in:buffer std_logic:='0';
       req_done_out:buffer std_logic;
       ack_done_out:in std_logic:='0';
       output:buffer std_logic);
end shift_base_cell;
architecture behavior of shift_base_cell is
  signal the_bit:std_logic;
  signal done_bit:std_logic:='0';
  begin
  process
```

```
begin
    if(load0 = '1' or load1 = '1') then
    if(load0 = '1') then
      the_bit <= '0';
      wait for delay(5,10);
    elsif(load1 = '1') then
      the_bit <= '1';
      wait for delay(5,10);
    end if;
    assign(ack_load,'1',5,10);
    guard_or(load0,'0',load1,'0');
    assign(ack_load,'0',5,10);
    done_bit <= '0';
    wait for delay(5,10);
  end if;
      if(shift_in0 = '1' or shift_in1 = '1' or
     req_done_in = '1') then
    while(done_bit = '0') loop
      guard_or(shift_in0,'1',shift_in1,'1',req_done_in,'1');
      if(shift_in0 = '1' or shift_in1 = '1') then
        if(the_bit = '0') then
          assign(shift_out0,'1',5,10);
        elsif(the_bit = '1') then
          assign(shift_out1,'1',5,10);
        end if;
        wait for delay(5,10);
        guard(ack_shift_out,'1');
        vassign(shift_out0,'0',5,10,shift_out1,'0',5,10);
        guard(ack_shift_out,'0');
        if(shift_in0 = '1') then
          the_bit <= '0';
        elsif(shift_in1 = '1') then
          the_bit <= '1';
        end if;
        wait for delay(5,10);
        assign(ack_shift_in,'1',5,10);
        guard_or(shift_in0,'0',shift_in1,'0');
        assign(ack_shift_in,'0',5,10);
      else                  assign(ack_done_in,'1',5,10);
        guard(req_done_in,'0');
        done_bit <= '1';
        output <= the_bit;
        wait for delay(5,10);
        assign(req_done_out,'1',5,10);
        guard(ack_done_out,'1');
        assign(req_done_out,'0',5,10);
        guard(ack_done_out,'0');
        assign(ack_done_in,'0',5,10);
      end if;
```

```
        wait for delay(5,10);
      end loop;
    end if;
        wait for delay(5,10);
  end process;
end behavior;

-----------------------------------------
-- This is the end 1 bit cell for a shifter
-----------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.handshake.all;
entity shift_end_cell is
  port(load0:in std_logic;
       load1:in std_logic;
       ack_load:buffer std_logic:='0';
       shift_in0:in std_logic;
       shift_in1:in std_logic;
       ack_shift_in:buffer std_logic:='0';
       req_done_in:in std_logic;
       ack_done_in:buffer std_logic:='0';
       output:buffer std_logic);
end shift_end_cell;
architecture behavior of shift_end_cell is
  signal the_bit:std_logic;
  signal done_bit:std_logic:='0';
  begin
  process
  begin
    if(load0 = '1' or load1 = '1') then
      if(load0 = '1') then
        the_bit <= '0';
        wait for delay(5,10);
      elsif(load1 = '1') then
        the_bit <= '1';
        wait for delay(5,10);
      end if;
      wait for delay(5,10);
      assign(ack_load,'1',5,10);
      guard_or(load0,'0',load1,'0');
      assign(ack_load,'0',5,10);
      done_bit <= '0';
      wait for delay(5,10);
    end if;
    if(shift_in0 = '1' or shift_in1 = '1' or
       req_done_in = '1') then
      while(done_bit = '0') loop
```

```
        guard_or(shift_in0,'1',shift_in1,'1',
                 req_done_in,'1');
        if(shift_in0 = '1' or shift_in1 = '1') then
          if(shift_in0 = '1') then
            the_bit <= '0';
          elsif(shift_in1 = '1') then
            the_bit <= '1';
          end if;
          wait for delay(5,10);
          assign(ack_shift_in,'1',5,10);
          guard_or(shift_in0,'0',shift_in1,'0');
          assign(ack_shift_in,'0',5,10);
        else
          assign(ack_done_in,'1',5,10);
          guard(req_done_in,'0');
          assign(ack_done_in,'0',5,10);
          done_bit <= '1';
          output <= the_bit;
          wait for delay(5,10);
        end if;
        wait for delay(5,10);
      end loop;
    end if;
    wait for delay(5,10);
  end process;
end behavior;

---------------------------------
-- 4-bit Shifter for Problem 3.10
---------------------------------
library ieee;
use ieee.std_logic_1164.all;
entity four_bit_shifter is
  port(main_shift_in0:in std_logic:='0';
       main_shift_in1:in std_logic:='0';
       ack_main_shift_in:buffer std_logic:='0';
       req_main_done_in:in std_logic:='0';
       ack_main_done_in:buffer std_logic:='0';
       load00:in std_logic:='0';
       load01:in std_logic:='0';
       ack_load0:buffer std_logic:='0';
       load10:in std_logic:='0';
       load11:in std_logic:='0';
       ack_load1:buffer std_logic:='0';
       load20:in std_logic:='0';
       load21:in std_logic:='0';
       ack_load2:buffer std_logic:='0';
       load30:in std_logic:='0';
       load31:in std_logic:='0';
```

```
      ack_load3:buffer std_logic:='0';
      out0:buffer std_logic;
      out1:buffer std_logic;
      out2:buffer std_logic;
      out3:buffer std_logic);
end four_bit_shifter;
architecture structure of four_bit_shifter is
  component shift_base_cell
    port(load0:in std_logic;
       load1:in std_logic;
       ack_load:buffer std_logic:='0';
       shift_in0:in std_logic;
       shift_in1:in std_logic;
       ack_shift_in:buffer std_logic:='0';
       shift_out0:buffer std_logic:='0';
       shift_out1:buffer std_logic:='0';
       ack_shift_out:in std_logic;
       req_done_in:in std_logic;
       ack_done_in:buffer std_logic:='0';
       req_done_out:buffer std_logic;
       ack_done_out:in std_logic:='0';
       output:buffer std_logic);
  end component;
  component shift_end_cell
    port(load0:in std_logic;
         load1:in std_logic;
         ack_load:buffer std_logic:='0';
         shift_in0:in std_logic;
         shift_in1:in std_logic;
         ack_shift_in:buffer std_logic:='0';
         req_done_in:in std_logic;
         ack_done_in:buffer std_logic:='0';
         output:buffer std_logic);
  end component;
  signal s_com010:std_logic;
  signal s_com011:std_logic;
  signal ack_s_com01:std_logic;
  signal s_com120:std_logic;
  signal s_com121:std_logic;
  signal ack_s_com12:std_logic;
  signal s_com230:std_logic;
  signal s_com231:std_logic;
  signal ack_s_com23:std_logic;
  signal req_d_com01:std_logic;
  signal ack_d_com01:std_logic;
  signal req_d_com12:std_logic;
  signal ack_d_com12:std_logic;
  signal req_d_com23:std_logic;
  signal ack_d_com23:std_logic;
```

```
signal bit01:std_logic;
signal bit12:std_logic;
signal bit23:std_logic;
signal ld0:std_logic;
signal ld1:std_logic;
signal ld2:std_logic;
signal ld3:std_logic;
begin
BIT_ZERO:shift_base_cell
  port map(shift_in0 => main_shift_in0,
           shift_in1 => main_shift_in1,
           ack_shift_in => ack_main_shift_in,
           shift_out0 => s_com010,
           shift_out1 => s_com011,
           ack_shift_out => ack_s_com01,
           load0 => load00,
           load1 => load01,
           ack_load => ack_load0,
           req_done_in => req_main_done_in,
           ack_done_in => ack_main_done_in,
           req_done_out => req_d_com01,
           ack_done_out => ack_d_com01,
           output => out0);
BIT_ONE:shift_base_cell
  port map(shift_in0 => s_com010,
           shift_in1 => s_com011,
           ack_shift_in => ack_s_com01,
           shift_out0  => s_com120,
           shift_out1  => s_com121,
           ack_shift_out  => ack_s_com12,
           load0 => load10,
           load1 => load11,
           ack_load => ack_load1,
           req_done_in => req_d_com01,
           ack_done_in => ack_d_com01,
           req_done_out => req_d_com12,
           ack_done_out => ack_d_com12,
           output => out1);
BIT_TWO:shift_base_cell
  port map(shift_in0  => s_com120,
           shift_in1  => s_com121,
           ack_shift_in  => ack_s_com12,
           shift_out0 => s_com230,
           shift_out1 => s_com231,
           ack_shift_out => ack_s_com23,
           load0 => load20,
           load1 => load21,
           ack_load => ack_load2,
           req_done_in => req_d_com12,
```

```
            ack_done_in => ack_d_com12,
            req_done_out => req_d_com23,
            ack_done_out => ack_d_com23,
            output => out2);
  BIT_THREE:shift_end_cell
    port map(shift_in0 => s_com230,
            shift_in1 => s_com231,
            ack_shift_in => ack_s_com23,
            load0 => load30,
            load1 => load31,
            ack_load => ack_load3,
            req_done_in => req_d_com23,
            ack_done_in => ack_d_com23,
            output => out3);
end structure;

-------------------------------
-- Test bench for Problem 3.10
-------------------------------
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.handshake.all;
entity env_struc is
  port(req_done:inout std_logic:='0';
       ack_done:buffer std_logic:='0';
       shift0:inout std_logic:='0';
       shift1:inout std_logic:='0';
       ack_shift:buffer std_logic:='0';
       load0_tl0:inout std_logic:='0';
       load0_tl1:inout std_logic:='0';
       ack_load0_tl:buffer std_logic:='0';
       load1_tl0:inout std_logic:='0';
       load1_tl1:inout std_logic:='0';
       ack_load1_tl:buffer std_logic:='0';
       load2_tl0:inout std_logic:='0';
       load2_tl1:inout std_logic:='0';
       ack_load2_tl:buffer std_logic:='0';
       load3_tl0:inout std_logic:='0';
       load3_tl1:inout std_logic:='0';
       ack_load3_tl:buffer std_logic:='0';
       out0_tl:buffer std_logic;
       out1_tl:buffer std_logic;
       out2_tl:buffer std_logic;
       out3_tl:buffer std_logic);
end env_struc;
architecture testing of env_struc is
  component four_bit_shifter
  port(main_shift_in0:in std_logic:='0';
```

```
      main_shift_in1:in std_logic:='0';
      ack_main_shift_in:buffer std_logic:='0';
      req_main_done_in:in std_logic:='0';
      ack_main_done_in:buffer std_logic:='0';
      load00:in std_logic:='0';
      load01:in std_logic:='0';
      ack_load0:buffer std_logic:='0';
      load10:in std_logic:='0';
      load11:in std_logic:='0';
      ack_load1:buffer std_logic:='0';
      load20:in std_logic:='0';
      load21:in std_logic:='0';
      ack_load2:buffer std_logic:='0';
      load30:in std_logic:='0';
      load31:in std_logic:='0';
      ack_load3:buffer std_logic:='0';
      out0:buffer std_logic;
      out1:buffer std_logic;
      out2:buffer std_logic;
      out3:buffer std_logic);
end component;
signal ld0_bit:std_logic;
signal ld1_bit:std_logic;
signal ld2_bit:std_logic;
signal ld3_bit:std_logic;
signal ld0_out:std_logic;
signal ld1_out:std_logic;
signal ld2_out:std_logic;
signal ld3_out:std_logic;
signal done_or_shift:std_logic:='0';
  begin
shifter:four_bit_shifter
  port map(main_shift_in0 => shift0,
           main_shift_in1 => shift1,
           ack_main_shift_in => ack_shift,
           req_main_done_in => req_done,
           ack_main_done_in => ack_done,
           load00 => load0_tl0,
           load01 => load0_tl1,
           ack_load0 => ack_load0_tl,
           load10 => load1_tl0,
           load11 => load1_tl1,
           ack_load1 => ack_load1_tl,
           load20 => load2_tl0,
           load21 => load2_tl1,
           ack_load2 => ack_load2_tl,
           load30 => load3_tl0,
           load31 => load3_tl1,
           ack_load3 => ack_load3_tl,
```

```
          out0 => out0_tl,
          out1 => out1_tl,
          out2 => out2_tl,
          out3 => out3_tl);
           process
  variable temp:integer;
begin
  --randomly choose a value to load into the shifter
  temp:=selection(2);
  case temp is
    when 1 =>
      ld0_bit <= '0';
    when others =>
      ld0_bit <= '1';
  end case;
  temp:=selection(2);
  case temp is
    when 1 =>
      ld1_bit <= '0';
    when others =>
      ld1_bit <= '1';
  end case;
  temp:=selection(2);
  case temp is
    when 1 =>
      ld2_bit <= '0';
    when others =>
      ld2_bit <= '1';
  end case;
  temp:=selection(2);
  case temp is
    when 1 =>
      ld3_bit <= '0';
    when others =>
      ld3_bit <= '1';
  end case;
  wait for delay(5,10);
  --load the shifter
  if(ld0_bit = '1') then
    assign(load0_tl1,'1',5,10);
  elsif(ld0_bit = '0') then
    assign(load0_tl0,'1',5,10);
  end if;
  wait for delay(5,10);
  guard(ack_load0_tl,'1');
  vassign(load0_tl0,'0',5,10,load0_tl1,'0',5,10);
  guard(ack_load0_tl,'0');
      if(ld1_bit = '1') then
    assign(load1_tl1,'1',5,10);
```

```
elsif(ld1_bit = '0') then
  assign(load1_tl0,'1',5,10);
end if;
wait for delay(5,10);
guard(ack_load1_tl,'1');
vassign(load1_tl0,'0',5,10,load1_tl1,'0',5,10);
guard(ack_load1_tl,'0');
        if(ld2_bit = '1') then
  assign(load2_tl1,'1',5,10);
elsif(ld2_bit = '0') then
  assign(load2_tl0,'1',5,10);
end if;
wait for delay(5,10);
guard(ack_load2_tl,'1');
vassign(load2_tl0,'0',5,10,load2_tl1,'0',5,10);
guard(ack_load2_tl,'0');
        if(ld3_bit = '1') then
  assign(load3_tl1,'1',5,10);
elsif(ld3_bit = '0') then
  assign(load3_tl0,'1',5,10);
end if;
wait for delay(5,10);
guard(ack_load3_tl,'1');
vassign(load3_tl0,'0',5,10,load3_tl1,'0',5,10);
guard(ack_load3_tl,'0');
--randomly choose to send a done or shift signal
temp:=selection(2);
case temp is
  when 1 =>
    done_or_shift <= '0';
  when others =>
    done_or_shift <= '1';
end case;
wait for delay(5,10);
    while(done_or_shift = '1')
loop
  ld3_bit <= ld2_bit;
  ld2_bit <= ld1_bit;
  ld1_bit <= ld0_bit;
  --randomly pick a bit to shift in
  temp:=selection(2);
  case temp is
    when 1 =>
      ld0_bit <= '0';
    when others =>
      ld0_bit <= '1';
  end case;
  wait for delay(5,10);
  if(ld0_bit = '0') then
```

```
        assign(shift0,'1',5,10);
      elsif(ld0_bit = '1') then
        assign(shift1,'1',5,10);
      end if;
      wait for delay(5,10);
      guard(ack_shift,'1');
      vassign(shift0,'0',5,10,shift1,'0',5,10);
      guard(ack_shift,'0');
            --randomly choose done or shift before checking the
      --loop condition again
      temp:=selection(2);
      case temp is
        when 1 =>
          done_or_shift <= '0';
        when others =>
          done_or_shift <= '1';
      end case;
      wait for delay(5,10);
    end loop;
        --send the done signal and then check the
    --output against the expected output
    assign(req_done,'1',5,10);
    guard(ack_done,'1');
    assign(req_done,'0',5,10);
    guard(ack_done,'0');
        wait for delay(5,10);
    ld0_out <= out0_tl;
    ld1_out <= out1_tl;
    ld2_out <= out2_tl;
    ld3_out <= out3_tl;
    wait for delay(5,10);
        assert ld0_bit = ld0_out
      report "The 0 bit does not match"
      severity error;
    assert ld1_bit = ld1_out
      report "The 1 bit does not match"
      severity error;
    assert ld2_bit = ld2_out
      report "The 2 bit does not match"
      severity error;
    assert ld3_bit = ld3_out
      report "The 3 bit does not match"
      severity error;
      end process;
end;
```

3.11 Perform handshaking expansion on your entropy decoder design from Problem 2.4.1 using bundled data. Create an environment and simulate.

```
-- Solution problem 2.4.1
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.handshake.all;
entity decoder is
end decoder;
architecture behavior of decoder is
--signal Inp:channel:=init_channel;
signal Inp_data :  std_logic;
signal Inp_rv :  std_logic;
signal Inp_snd :  std_logic;
--signal Outp:channel:=init_channel;
signal Outp_data :  std_logic_vector(1 downto 0);
signal Outp_rv :  std_logic;
signal Outp_snd :  std_logic;
signal InpData:std_logic;
signal OutData:std_logic_vector(1 downto 0);
signal Result:std_logic_vector(1 downto 0);
signal Zero:std_logic:='0';
signal One:std_logic:='1';
begin
decoder:process
begin
  OutData <= "00";
  wait for 5 ps;
  -- receive(inp,inpdata);
  guard(inp_snd,'1');
  --@synthesis_off
  inpdata <= inp_data;
  wait for 5 ns;
  --@synthesis_on
  assign(inp_rv,'1',1,3);
  guard(inp_snd,'0');
  assign(inp_rv,'0',1,3);
  --@synthesis_off
  while (InpData='1')
  loop
  --@synthesis_on
    OutData <= OutData + "01";
    wait for 5 ps;
    -- receive(inp,inpdata);
    guard(inp_snd,'1');
    --@synthesis_off
    inpdata <= inp_data;
    wait for 5 ns;
    --@synthesis_on
```

```
    assign(inp_rv,'1',1,3);
    guard(inp_snd,'0');
    assign(inp_rv,'0',1,3);
  --@synthesis_off
  end loop;
  -- send(outp,outdata);
  outp_data <= outdata;
  --@synthesis_on
  wait for 5 ns;
  assign(outp_snd,'1',1,3);
  guard(outp_rv,'1');
  assign(outp_snd,'0',1,3);
  guard(outp_rv,'0');
end process;
env:process
variable z:integer;
begin
  z:=selection(4);
  case (z) is
  when 1 =>
    -- send(inp,zero);
    --@synthesis_off
    inp_data <= zero;
    --@synthesis_on
    wait for 5 ns;
    assign(inp_snd,'1',1,3);
    guard(inp_rv,'1');
    assign(inp_snd,'0',1,3);
    guard(inp_rv,'0');
  when 2 =>
    -- send(inp,one);
    --@synthesis_off
    inp_data <= one;
    --@synthesis_on
    wait for 5 ns;
    assign(inp_snd,'1',1,3);
    guard(inp_rv,'1');
    assign(inp_snd,'0',1,3);
    guard(inp_rv,'0');
    -- send(inp,zero);
    --@synthesis_off
    inp_data <= zero;
    --@synthesis_on
    wait for 5 ns;
    assign(inp_snd,'1',1,3);
    guard(inp_rv,'1');
    assign(inp_snd,'0',1,3);
    guard(inp_rv,'0');
  when 3 =>
```

```
      -- send(inp,one);
      --@synthesis_off
      inp_data <= one;
      --@synthesis_on
      wait for 5 ns;
      assign(inp_snd,'1',1,3);
      guard(inp_rv,'1');
      assign(inp_snd,'0',1,3);
      guard(inp_rv,'0');
      -- send(inp,one);
      --@synthesis_off
      inp_data <= one;
      --@synthesis_on
      wait for 5 ns;
      assign(inp_snd,'1',1,3);
      guard(inp_rv,'1');
      assign(inp_snd,'0',1,3);
      guard(inp_rv,'0');
      -- send(inp,zero);
      --@synthesis_off
      inp_data <= zero;
      --@synthesis_on
      wait for 5 ns;
      assign(inp_snd,'1',1,3);
      guard(inp_rv,'1');
      assign(inp_snd,'0',1,3);
      guard(inp_rv,'0');
    when others =>
      -- send(inp,one);
      --@synthesis_off
      inp_data <= one;
      --@synthesis_on
      wait for 5 ns;
      assign(inp_snd,'1',1,3);
      guard(inp_rv,'1');
      assign(inp_snd,'0',1,3);
      guard(inp_rv,'0');
      -- send(inp,one);
      --@synthesis_off
      inp_data <= one;
      --@synthesis_on
      wait for 5 ns;
      assign(inp_snd,'1',1,3);
      guard(inp_rv,'1');
      assign(inp_snd,'0',1,3);
      guard(inp_rv,'0');
      -- send(inp,one);
      --@synthesis_off
      inp_data <= one;
```

```
    --@synthesis_on
    wait for 5 ns;
    assign(inp_snd,'1',1,3);
    guard(inp_rv,'1');
    assign(inp_snd,'0',1,3);
    guard(inp_rv,'0');
    -- send(inp,zero);
    --@synthesis_off
    inp_data <= zero;
    --@synthesis_on
    wait for 5 ns;
    assign(inp_snd,'1',1,3);
    guard(inp_rv,'1');
    assign(inp_snd,'0',1,3);
    guard(inp_rv,'0');
  end case;
  -- receive(outp,result);
  guard(outp_snd,'1');
  --@synthesis_off
  result <= outp_data;
  wait for 5 ns;
  --@synthesis_on
  assign(outp_rv,'1',1,3);
  guard(outp_snd,'0');
  assign(outp_rv,'0',1,3);
  wait for 5 ps;
end process;
end behavior;
```

3.12 Perform handshaking expansion on your entropy decoder design from Problem 2.4.2 using dual-rail data encoding. Create an environment and simulate a 4-bit decoder.

NOTE: This solution uses bundled data for Load and Outp channels while it uses dual-rail for L and R.

```
-- Solution problem 3.12
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.handshake.all;
entity decoder1_hs is
  port(L0:in std_logic;
       L1:in std_logic;
       LAck:buffer std_logic:='0';
       R0:buffer std_logic;
       R1:buffer std_logic;
       RAck:in std_logic:='0';
```

```
        LoadReq:in std_logic;
        LoadAck:buffer std_logic:='0';
        LoadData:in std_logic_vector(1 downto 0);
        OutpReq:buffer std_logic:='0';
        OutpAck:in std_logic;
        OutpData:buffer std_logic_vector(1 downto 0));
end decoder1_hs;
architecture behavior of decoder1_hs is
begin
decoder1:process
begin
  guard_or(LoadReq,'1',L0,'1',L1,'1');
  if (LoadReq='1') then
    OutpData <= LoadData after delay(2,3);
    wait for delay(4,5);
    assign(LoadAck,'1',0,1);
    guard(LoadReq,'0');
    assign(LoadAck,'0',0,1);
  else
    if (L0='1') then
      assign(Lack,'1',0,1);
      guard_and(L0,'0',L1,'0');
      assign(Lack,'0',0,1);
      assign(OutpReq,'1',0,1);
      guard(OutpAck,'1');
      assign(OutpReq,'0',0,1);
      guard(OutpAck,'0');
    else
      while (L1='1')
      loop
        assign(Lack,'1',0,1);
        guard_and(L0,'0',L1,'0');
        assign(Lack,'0',0,1);
        guard_or(L0,'1',L1,'1');
        if (L0='1') then
          assign(R0,'1',0,1);
        else
          assign(R1,'1',0,1);
        end if;
        guard(RAck,'1');
        vassign(R0,'0',0,1,R1,'0',0,1);
        guard(RAck,'0');
      end loop;
      assign(Lack,'1',0,1);
      guard_and(L0,'0',L1,'0');
      assign(Lack,'0',0,1);
    end if;
  end if;
end process;
```

```
end behavior;

-- Solution problem 3.12
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.nondeterminism.all;
use work.handshake.all;
entity Prob3_12 is
end Prob3_12;
architecture behavior of Prob3_12 is
component decoder1_hs
  port(L0:in std_logic;
       L1:in std_logic;
       LAck:buffer std_logic;
       R0:buffer std_logic;
       R1:buffer std_logic;
       RAck:in std_logic;
       LoadReq:in std_logic;
       LoadAck:buffer std_logic;
       LoadData:in std_logic_vector(1 downto 0);
       OutpReq:buffer std_logic;
       OutpAck:in std_logic;
       OutpData:buffer std_logic_vector(1 downto 0));
end component;
signal L0:std_logic:='0';
signal L1:std_logic:='0';
signal LAck:std_logic:='0';
signal C10:std_logic:='0';
signal C11:std_logic:='0';
signal C1Ack:std_logic:='0';
signal C20:std_logic:='0';
signal C21:std_logic:='0';
signal C2Ack:std_logic:='0';
signal C30:std_logic:='0';
signal C31:std_logic:='0';
signal C3Ack:std_logic:='0';
signal R0:std_logic:='0';
signal R1:std_logic:='0';
signal RAck:std_logic:='0';
signal Load1Req:std_logic:='0';
signal Load1Ack:std_logic:='0';
signal Outp1Req:std_logic:='0';
signal Outp1Ack:std_logic:='0';
signal Load2Req:std_logic:='0';
signal Load2Ack:std_logic:='0';
signal Outp2Req:std_logic:='0';
signal Outp2Ack:std_logic:='0';
```

```
signal Load3Req:std_logic:='0';
signal Load3Ack:std_logic:='0';
signal Outp3Req:std_logic:='0';
signal Outp3Ack:std_logic:='0';
signal Load4Req:std_logic:='0';
signal Load4Ack:std_logic:='0';
signal Outp4Req:std_logic:='0';
signal Outp4Ack:std_logic:='0';
signal LoadData:std_logic_vector(1 downto 0);
signal Outp1Data:std_logic_vector(1 downto 0);
signal Outp2Data:std_logic_vector(1 downto 0);
signal Outp3Data:std_logic_vector(1 downto 0);
signal Outp4Data:std_logic_vector(1 downto 0);
signal Result:std_logic_vector(1 downto 0);
begin
  DEC1:decoder1_hs
    port map(L0 => L0, L1 => L1, LAck => LAck,
             R0 => C10, R1 => C11, RAck => C1Ack,
             LoadReq => Load1Req, LoadAck => Load1Ack,
             LoadData => LoadData, OutpReq => Outp1Req,
             OutpAck => Outp1Ack, OutpData => Outp1Data);
  DEC2:decoder1_hs
    port map(L0 => C10, L1 => C11, LAck => C1Ack,
             R0 => C20, R1 => C21, RAck => C2Ack,
             LoadReq => Load2Req, LoadAck => Load2Ack,
             LoadData => LoadData, OutpReq => Outp2Req,
             OutpAck => Outp2Ack, OutpData => Outp2Data);
  DEC3:decoder1_hs
    port map(L0 => C20, L1 => C21, LAck => C2Ack,
             R0 => C30, R1 => C31, RAck => C3Ack,
             LoadReq => Load3Req, LoadAck => Load3Ack,
             LoadData => LoadData, OutpReq => Outp3Req,
             OutpAck => Outp3Ack, OutpData => Outp3Data);
  DEC4:decoder1_hs
    port map(L0 => C30, L1 => C31, LAck => C3Ack,
             R0 => R0, R1 => R1, RAck => RAck,
             LoadReq => Load4Req, LoadAck => Load4Ack,
             LoadData => LoadData, OutpReq => Outp4Req,
             OutpAck => Outp4Ack, OutpData => Outp4Data);
  env:process
  variable z:integer;
  begin
    LoadData <= "00" after delay(2,3);
    wait for 5 ns;
    assign(Load1Req,'1',0,1);
    guard(Load1Ack,'1');
    assign(Load1Req,'0',0,1);
    guard(Load1Ack,'0');
    LoadData <= "01" after delay(2,3);
```

```
wait for 5 ns;
assign(Load2Req,'1',0,1);
guard(Load2Ack,'1');
assign(Load2Req,'0',0,1);
guard(Load2Ack,'0');
LoadData <= "10" after delay(2,3);
wait for 5 ns;
assign(Load3Req,'1',0,1);
guard(Load3Ack,'1');
assign(Load3Req,'0',0,1);
guard(Load3Ack,'0');
LoadData <= "11" after delay(2,3);
wait for 5 ns;
assign(Load4Req,'1',0,1);
guard(Load4Ack,'1');
assign(Load4Req,'0',0,1);
guard(Load4Ack,'0');
while (true)
loop
  z:=selection(4);
  case (z) is
  when 1 =>
    assign(L0,'1',0,1);
    guard(LAck,'1');
    assign(L0,'0',0,1);
    guard(LAck,'0');
    guard(Outp1Req,'1');
    Result <= Outp1Data after delay(2,3);
    wait for 5 ns;
    assign(Outp1Ack,'1',0,1);
    guard(Outp1Req,'0');
    assign(Outp1Ack,'0',0,1);
  when 2 =>
    assign(L1,'1',0,1);
    guard(LAck,'1');
    assign(L1,'0',0,1);
    guard(LAck,'0');
    assign(L0,'1',0,1);
    guard(LAck,'1');
    assign(L0,'0',0,1);
    guard(LAck,'0');
    guard(Outp2Req,'1');
    Result <= Outp2Data after delay(2,3);
    wait for 5 ns;
    assign(Outp2Ack,'1',0,1);
    guard(Outp2Req,'0');
    assign(Outp2Ack,'0',0,1);
  when 3 =>
    assign(L1,'1',0,1);
```

```
            guard(LAck,'1');
            assign(L1,'0',0,1);
            guard(LAck,'0');
            assign(L1,'1',0,1);
            guard(LAck,'1');
            assign(L1,'0',0,1);
            guard(LAck,'0');
            assign(L0,'1',0,1);
            guard(LAck,'1');
            assign(L0,'0',0,1);
            guard(LAck,'0');
            guard(Outp3Req,'1');
            Result <= Outp3Data after delay(2,3);
            wait for 5 ns;
            assign(Outp3Ack,'1',0,1);
            guard(Outp3Req,'0');
            assign(Outp3Ack,'0',0,1);
          when others =>
            assign(L1,'1',0,1);
            guard(LAck,'1');
            assign(L1,'0',0,1);
            guard(LAck,'0');
            assign(L1,'1',0,1);
            guard(LAck,'1');
            assign(L1,'0',0,1);
            guard(LAck,'0');
            assign(L1,'1',0,1);
            guard(LAck,'1');
            assign(L1,'0',0,1);
            guard(LAck,'0');
            assign(L0,'1',0,1);
            guard(LAck,'1');
            assign(L0,'0',0,1);
            guard(LAck,'0');
            guard(Outp4Req,'1');
            Result <= Outp4Data after delay(2,3);
            wait for 5 ns;
            assign(Outp4Ack,'1',0,1);
            guard(Outp4Req,'0');
            assign(Outp4Ack,'0',0,1);
        end case;
      end loop;
  end process;
  end behavior;
```

3.13 Using syntax-directed translation, find a circuit to implement Euclid's greatest common divisor (gcd) algorithm given below.

```
receive(A,a,B,b);
```

```
while (a != b) loop
  if (a > b) then
    a <= a - b;
    wait for 5 ns;
  else
    b <= b - a;
    wait for 5 ns;
  end if;
end loop;
end if;
send(C,a);
```

You may assume that the user is not allowed to send in operands a or b equal to 0. You can also assume the existence of a comparator which has two input buses, and after a request sets a condition which is true when operand 1 is greater than operand 2. After the condition is stable, an acknowledgment is provided. You may also assume the existence of a subtractor which has two input buses and one output bus. After getting a request, it returns an acknowledgment to indicate a stable result on the output bus.

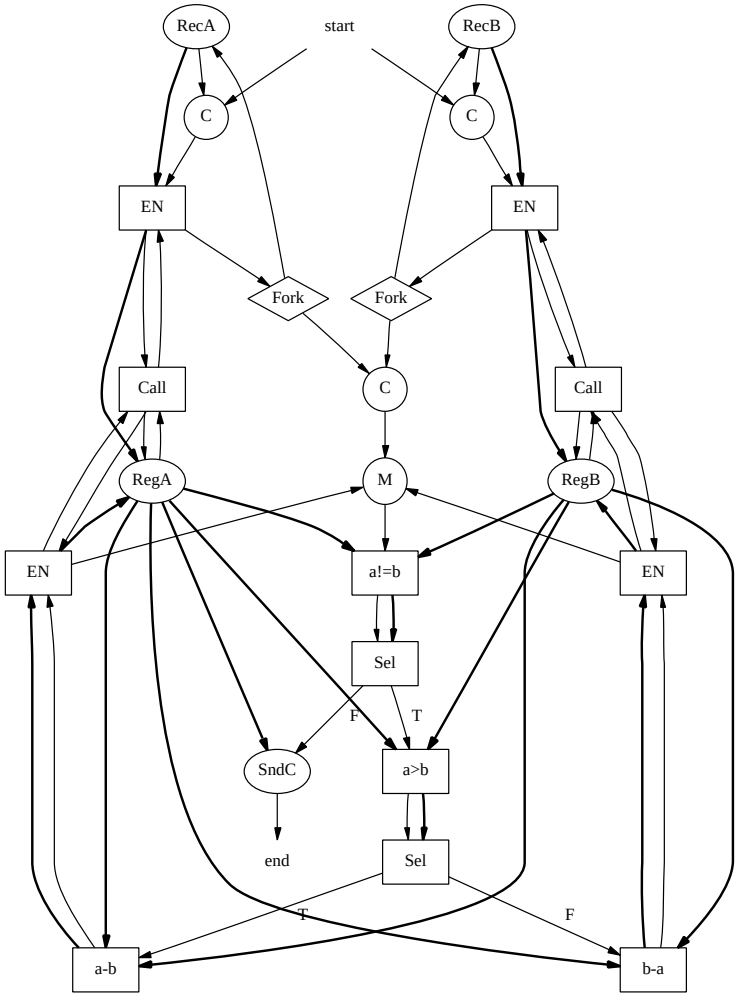


Fig. 3.1 Solution to Problem 3.13.

4 Graphical Representations

Problems

4.1 Asynchronous Finite State Machines

The AFSM for a *quick-return linkage circuit* (QRL) is shown in Figure 4.1.

4.1.1. Find the flow table for the AFSM.

4.1.2. Draw a burst-mode state machine for the QRL circuit.

Solution is shown in Figure **??**.

4.2 Burst-Mode State Machines

For each BM state diagram in Figure 4.2, determine whether:

4.2.1. It has the maximal set property.

They all violate this property in state 2.

4.2.2. It represents a BM machine.

(a) is not a BM machine because the path state 0, 1, 4, 0, results in a high when it started low. (b) is not because the path state 0, state 1, state 4, state 0 results in x being low at the beginning but high at the end in the same state. Also, y is set low between state 4 and 0 when it is already low. (c) is not because the path state 0, state 1, state 4, state 0 results in y being set low when it was never set high. (d) is a BM machine.

4.3 Burst-Mode to Flow Table

Translate the BM machine shown in Figure 4.3(a) into a flow table.

Solution is shown in Figure 4.4.

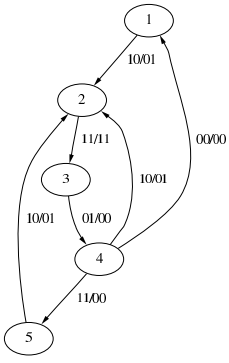


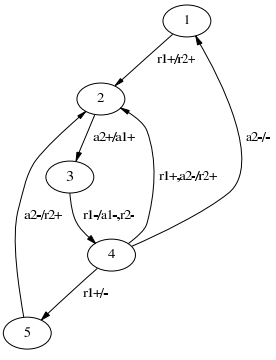
Fig. 4.1 AFSM for Problem 4.1.

r1 / a2

	00	01	11	10
s1	(s1), 00	–	–	s2, 01
s2	–	–	s3, 11	(s2), 01
s3	–	s4, 00	(s3), 11	–
s4	s1, 00	(s4), 00	s5, 00	s2, 01
s5	–	–	(s5), 00	s2, 01

a1 / r2

(a)



(b)

Fig. 4.2 (a) Flow table and (b) burst-mode state machine for Problem 4.1.

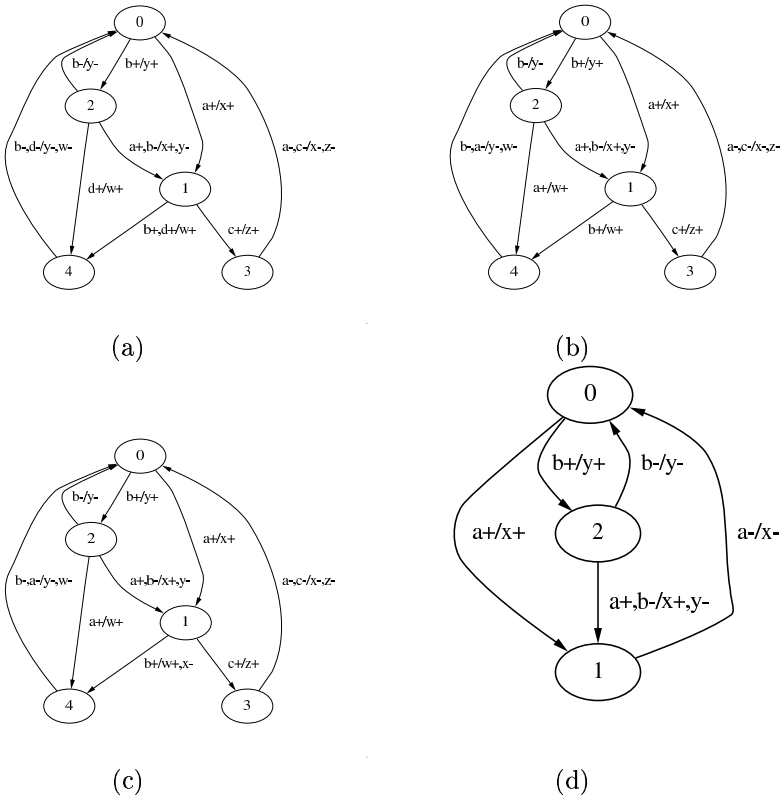


Fig. 4.3 BM machines for Problem 4.2.

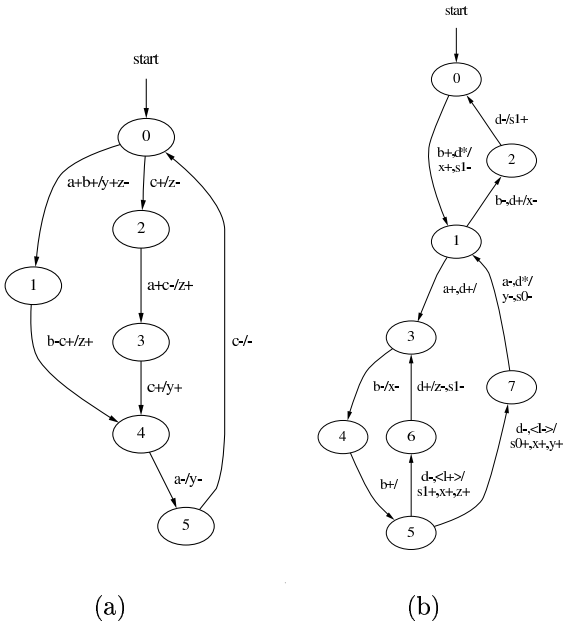


Fig. 4.4 (a) BM machine for Problem 4.3. Note that $abc = 000$ and $yz = 01$ initially. (b) XBM machine for Problem 4.5. Note that $abdl = 000-$ and $s0s1xyz = 01000$ initially.

	a / b / c							
	000	001	011	010	110	111	101	100
s0	(s0), 01	s2, 00	–	(s0), 01	(s1), 10	–	–	(s0), 01
s1	–	–	–	–	(s1), 10	(s1), 10	s4, 11	(s1), 10
s2	(s2), 00	(s2), 00	–	–	–	–	(s2), 00	s3, 01
s3	–	–	–	–	–	–	s4, 11	(s3), 01
s4	–	s5, 01	–	–	–	–	(s4), 11	–
s5	s0, 01	(s5), 01	–	–	–	–	–	–
	y / z							

Fig. 4.5 Flow table for Problem 4.3(a)

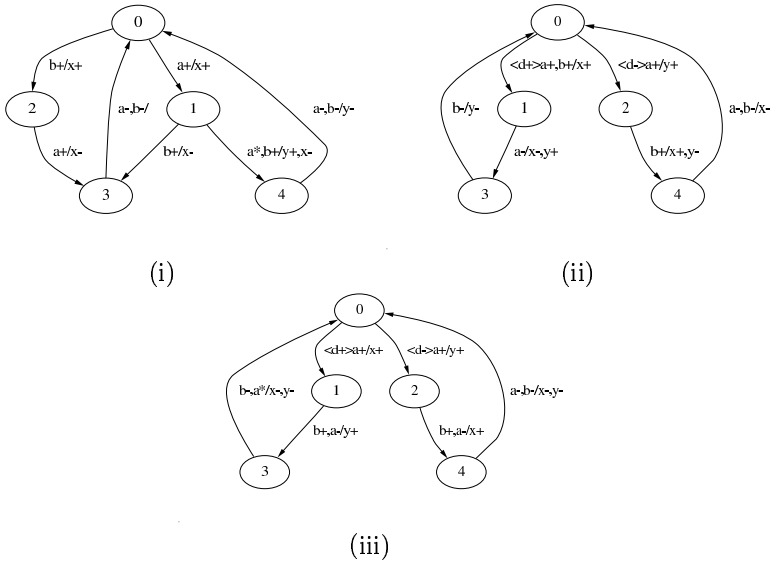


Fig. 4.6 Extended burst-mode machines for Problem 4.4.

4.4 Extended Burst-Mode State Machines

For the extended burst-mode state machines in Figure 4.5, determine whether they satisfy the maximal set property, and if not, explain why not.

(i) Doesn't satisfy maximal set property because one input burst $b+$ is included in the other input burst $a*, b+$ in state 1.

(ii) Satisfies maximal set property.

(iii) Doesn't satisfy maximal set property because there is no compulsory transition for $a+$ in state 0.

4.5 Extended Burst-Mode to Flow Table

Translate the XBM machine shown in Figure 4.3(b) into a flow table.

4.6 Petri net Properties

For the Petri nets in Figure 4.7, determine whether they have the following properties:

4.6.1. k-bounded, and if so, for what value of k? Is it safe?

(a) Unbounded. Therefore, it is not safe.

(b) k–bounded and $k = 1$. Therefore, it is safe.

(c) k–bounded and $k = 1$. Therefore, it is safe.

	0000	0001	0011	0010
s0	(s0), 01000	(s0), 01000	(s0), 01000	(s0), 01000
s1	(s1), 00100	(s1), 00100	s2, 00000	s2, 00000
s2	s0, 00100	s0, 00100	(s2), 00000	(s2), 00000
s3	–	–	–	–
s4	–	–	–	–
s5	–	–	–	–
s6	–	–	–	–
s7	–	–	–	–

	0110	0111	0101	0100
s0	s1, 00100	s1, 00100	s1, 00100	s1, 00100
s1	(s1), 00100	(s1), 00100	(s1), 00100	(s1), 00100
s2	–	–	–	–
s3	–	–	–	–
s4	–	–	–	–
s5	–	–	–	–
s6	–	–	–	–
s7	s1, 00100	s1, 00100	s1, 00100	s1, 00100

	1100	1101	1111	1110
s0	–	–	–	–
s1	(s1), 00100	(s1), 00100	s3, 00100	s3, 00100
s2	–	–	–	–
s3	–	–	(s3), 00100	(s3), 00100
s4	–	–	s5, 00000	s5, 00000
s5	s7, 10110	s6, 01101	(s5), 00000	(s5), 00000
s6	(s6), 01101	(s6), 01101	s3, 00100	s3, 00100
s7	(s7), 10110	(s7), 10110	(s7), 10110	(s7), 10110

	1010	1011	1001	1000
s0	–	–	–	–
s1	–	–	–	–
s2	–	–	–	–
s3	s4, 00000	s4, 00000	–	–
s4	(s4), 00000	(s4), 00000	–	–
s5	–	–	–	–
s6	–	–	–	–
s7	–	–	–	–

Fig. 4.7 Flow table for Problem 4.5 Inputs are $abdl$, and outputs are $s0S1xyz$.

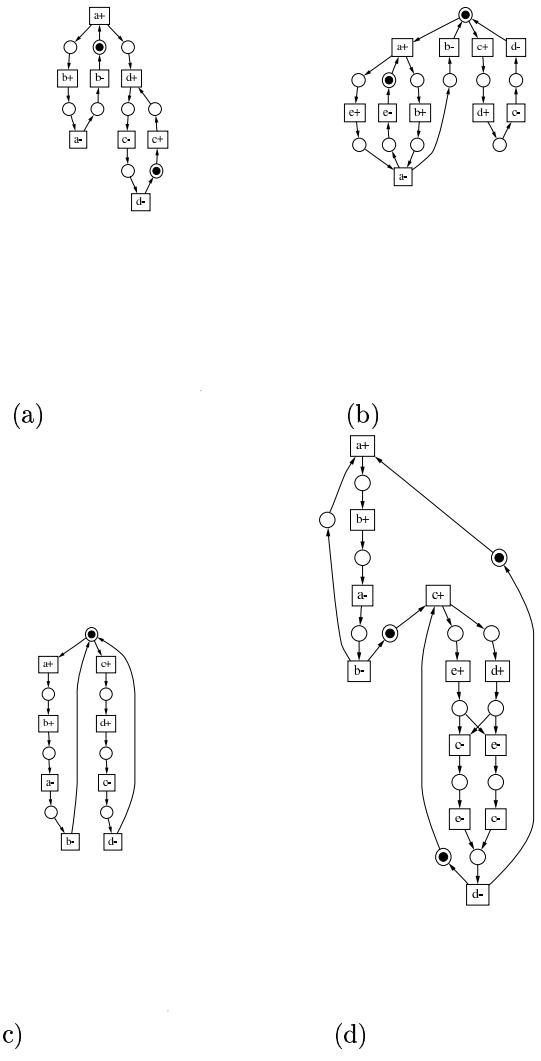


Fig. 4.8 Petri nets for Problems 4.6, 4.7, and 4.8.

(d) k–bounded and $k = 2$. Therefore, it is not safe.

4.6.2. Live, and if not, classify each transition's degree of liveness.

(a), (b), and (c) are live. (d) is not live. For (d) $a+, b+, a-, b-$ are dead, and $c+, e+, d+, c-, e-, d-$ are L1-live.

4.7 Petri net Classifications

For the Petri nets in Figure 4.7, determine whether they fall into the following classifications:

4.7.1. State machine: (c)
4.7.2. Marked graph: (a)
4.7.3. Free–choice net: (a), (c)
4.7.4. Extended free–choice net: (a), (c), (d)
4.7.5. Asymmetric choice net: (a), (b), (c), (d)

4.8 Reachability Graphs

For the Petri net in Figure 4.7(b), determine its reachability graph.

Solution is shown in Figure 4.8.

4.9 Signal Transition Graphs

For the signal transition graphs in Figure 4.9, state which of the following properties are satisfied or not, and if not, say why:

4.9.1. Live

(a) and (b) are live. (c) is not for trace $a+$, $b+$.

4.9.2. Safe

(a) and (b) are safe. (c) is not for trace $c+$, $b+$, $c-$, $b-$, $c+$, $b+$.

4.9.3. Persistent

(a) is persistent. (b) is not persistent for arc $c- \rightarrow b-$ since $d-$, $c+$ can fire while $c- \rightarrow b-$ remains marked. (c) is not persistent for arc $b+ \rightarrow a-$ since $c-$, $b-$ can fire while this arc remains marked.

4.9.4. Single-cycle transitions

(a) and (b) have single-cycle transitions. (c) does not since $b+$ and $b-$ each appear twice.

4.10 State Graphs

For the signal transition graphs in Figure 4.9, find their state graphs and determine if they have the following properties, and if not, say why:

The state graphs are shown in Figure 4.10, 4.11, and 4.12.

4.10.1. Consistent state assignment

They all have consistent state assignment.

4.10.2. Unique state assignment

(a) has unique state assignment. (b) does not hae unique state assignment. Starting in state R0R0, after firing $a+$, $b+$, $a-$, it is in state 01R0. Continuing with firing $c+$, $d+$, $c-$, $d-$, it is in state 0FR0. These two states have the same state code 0100. (c) does not have unique state assignment. Starting in state R0R, after firing $c+$, $b+$, $c-$, $b-$, we have a new marking which includes a token now also on the $b+ \rightarrow a-$ arc.

4.11 Signal Transition Graphs

For the signal transition graphs in Figure 4.13, state which of the following properties are satisfied or not, and if not, say why:

4.11.1. Live

(b), (c), and (e) are live. (a) is not live because it is dead after a firing sequence $a+, b+, a-, b-$. (d) is not live because it is dead after a firing sequence $c+, a-$.

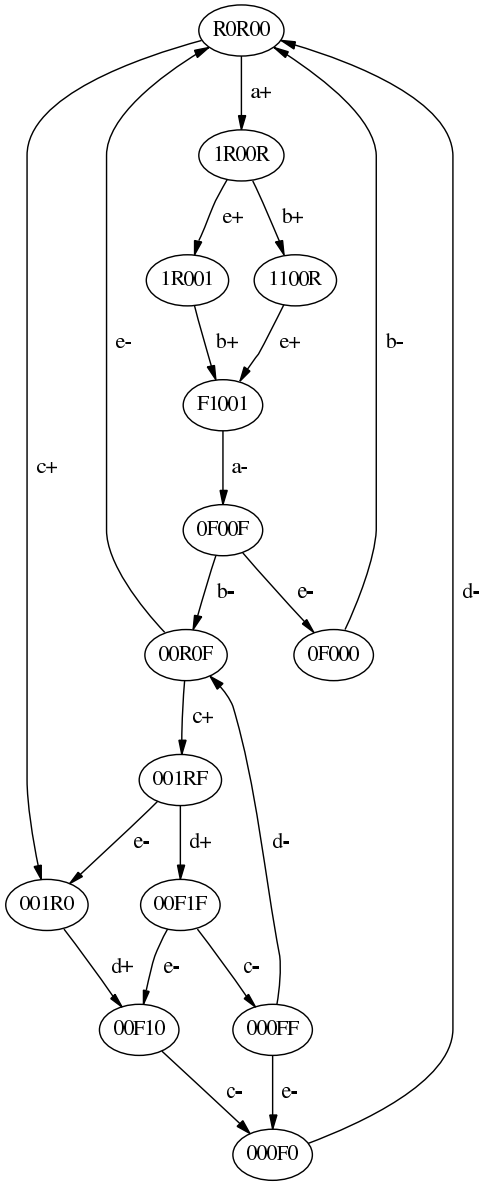
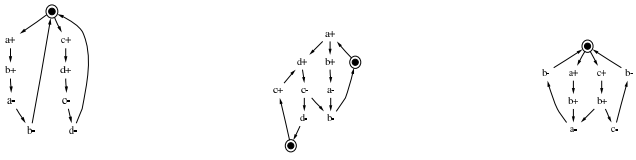


Fig. 4.9 Solution for Problem 4.8.



(a) (b) (c)

Fig. 4.10 Signal transition graphs for problems 4.9 and 4.10.

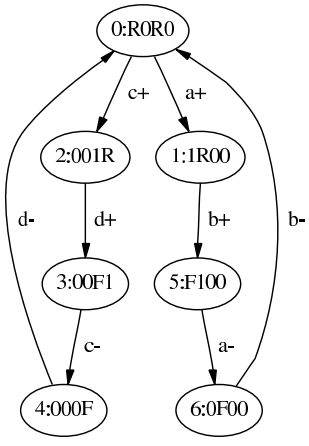


Fig. 4.11 State graphs for Problems 4.10(a).

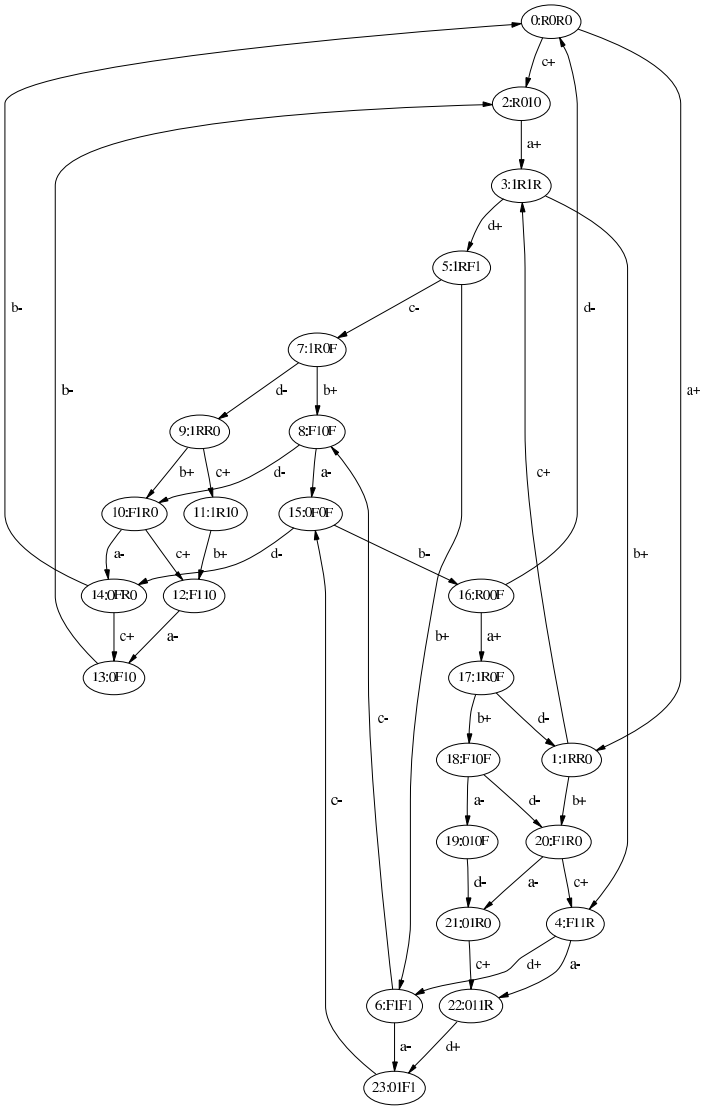


Fig. 4.12 State graphs for Problems 4.10(b).

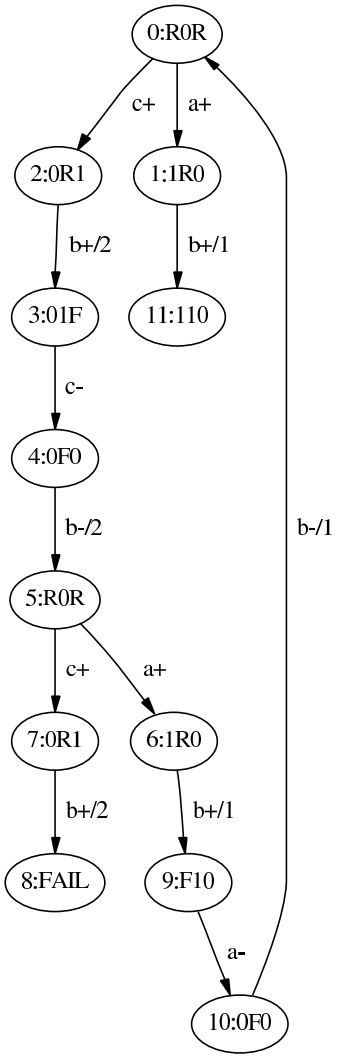


Fig. 4.13 State graphs for Problems 4.10(c).

4.11.2. Safe

(a), (c), (d), and (e) are safe. (b) is not safe after firing of $z-$.

4.11.3. Persistent

(a), (c), and (d) are persistent. (b) is not persistent because the arc $y- \rightarrow w+$ can remain marked while $y+$ fires. (e) is not persistent because the arc $d+ \rightarrow b-$ can remain marked while $d-$ fires.

4.11.4. Single-cycle transitions

They all have single-cycle transitions.

4.12 State Graphs

For the signal transition graphs in Figure 4.13, find their state graphs and determine if they have the following properties, and if not, say why:

State graphs are shown in Figure 4.14.

4.12.1. Consistent state assignment

They all have consistent state assignment.

4.12.2. Unique state assignment

Only (d) has unique state assignment.

4.13 Signal Transition Graphs

For the signal transition graphs in Figure 4.15, state which of the following properties are satisfied or not, and if not, say why:

4.13.1. Live

(a), (b), and (c) are live. (d) is not live after $r1+$.

4.13.2. Safe

(a), (c), and (d) are safe. (b) is not safe. After the trace $r1+$, $r2+$, $a2+$, $a1+$, the arc $a1+ \rightarrow r1-$ has two tokens.

4.13.3. Persistent

(c) and (d) are persistent. (a) is not persistent. After the trace $r1+$, $r2+$, $a2+$, $a1+$, $r1-$, $a1-$, $r1+$, the arc $r1- \rightarrow r2-$ is marked but $r1$ has gone high. (b) is not persistent. After the trace $r1-$, $r1+$, the arc $r1- \rightarrow r2-$ is marked, but $r1$ has gone high.

4.13.4. Single-cycle transitions

(a), (b), and (d) all have single-cycle transitions. (c) has two $r1+$ transitions.

4.14 State Graphs

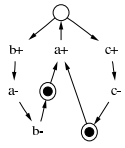
For the signal transition graphs in Figure 4.15, find their state graphs and determine if they have the following properties, and if not, say why:

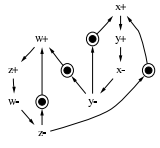
State graphs are shown in Figure 4.16,4.17, 4.18, and 4.19.

4.14.1. Consistent state assignment

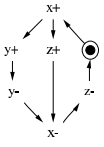
(b) is not consistent. The trace $r1-$ fires $r1$ low although it is already low at the start. (c) does not have a consistent state assignment. After firing, $r1+$, $r2+$, $a2+$, $a1+$, the transition $r1+$ can fire again although it is already high.

4.14.2. Unique state assignment

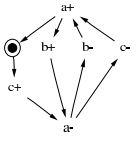




(a)

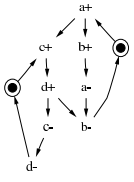


(b)



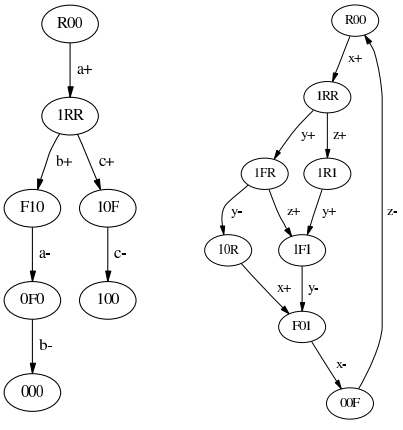
(c)

(d)



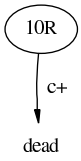
(e)

Fig. 4.14 Signal transition graphs for problems 4.11 and 4.12.

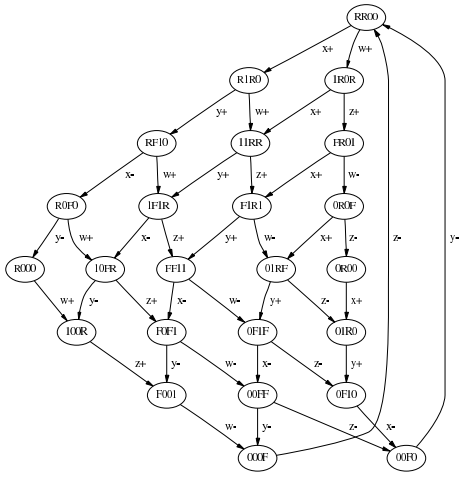


(a) $\langle a, b, c \rangle$

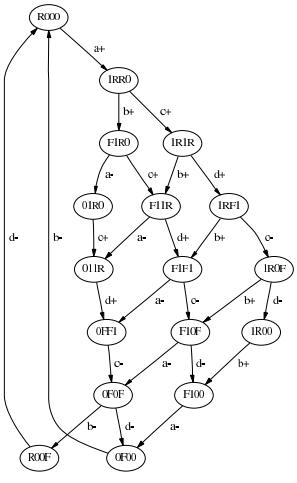
(c) $\langle x, y, z \rangle$



(d) $\langle a, b, c \rangle$



(b) $\langle w, x, y, z \rangle$



(e) $\langle a, b, c, d \rangle$

Fig. 4.15 State graphs for Problem 4.12.

(a) (b) (c) (d)

Fig. 4.16 Signal transition graphs for Problems 4.13 and 4.14.

(d) has a unique state assignment. (a) does not have USC. The trace $r1+$, $r2+$, $a2+$ results in the state 1101. If we continue from here with the trace $a1+$, $r1-$, $a1-$, $r1+$, the state is again 1101, but for a different marking. (b) does not have USC. The trace $r1-$, $a1-$, $r1+$ leads to state 1000. If we fire $r1+$ again from here, the state is still 1000. (c) also does not have USC. The trace $r1+$ leads to state 1000. If we continue from here with the trace $r2+$, $a2+$, $a1+$, $r1+$, $r2-$, $a2-$, $a1-$, we are again in state 1000 but for a different marking.

4.15 State Graphs

Find the state graph from the STG representation of the quick return linkage (QRL) circuit shown in Figure 4.20(a).

State graph is shown in Figure 4.21.

4.16 State Graphs

Assuming that all signals are initially low, find the SG from the STG in Figure 4.20(b).

State graph is shown in Figure 4.22.

4.17 State Graphs

Assuming that all signals are initially low, find the SG from the TEL structure in Figure 4.23. The events $a+$, $a-$, $b+$, and $b-$ all conflict with each other.

4.18 Translation

For the following handshaking expansion, draw graphical representations using the following methods:

- BM machine

  Solution shown in Figure 4.25.

- STG

  Solution shown in Figure 4.26.

- TEL structure

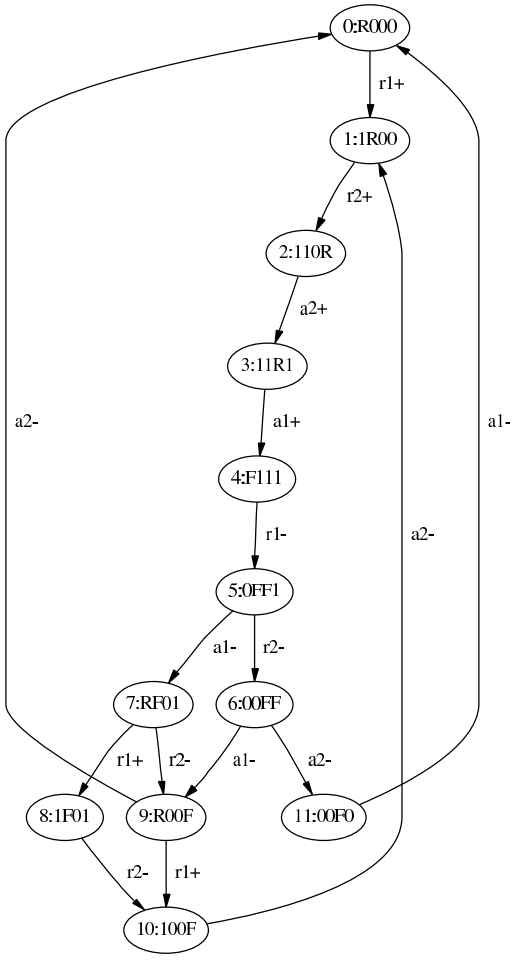


Fig. 4.17 State graph for Problems 4.14(a).

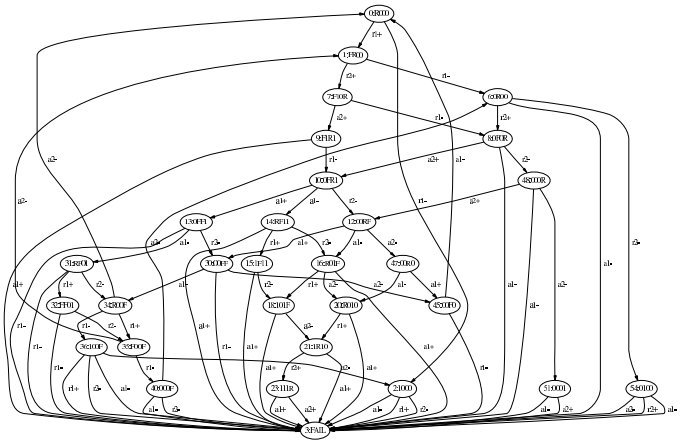


Fig. 4.18 State graph for Problems 4.14(b).

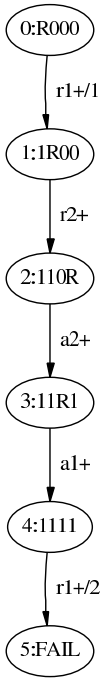


Fig. 4.19 State graph for Problems 4.14(c).

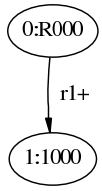


Fig. 4.20 State graph for Problems 4.14(d).

(a) (b)

Fig. 4.21 (a) QRL circuit for Problem 4.15. (b) STG for Problem 4.16.

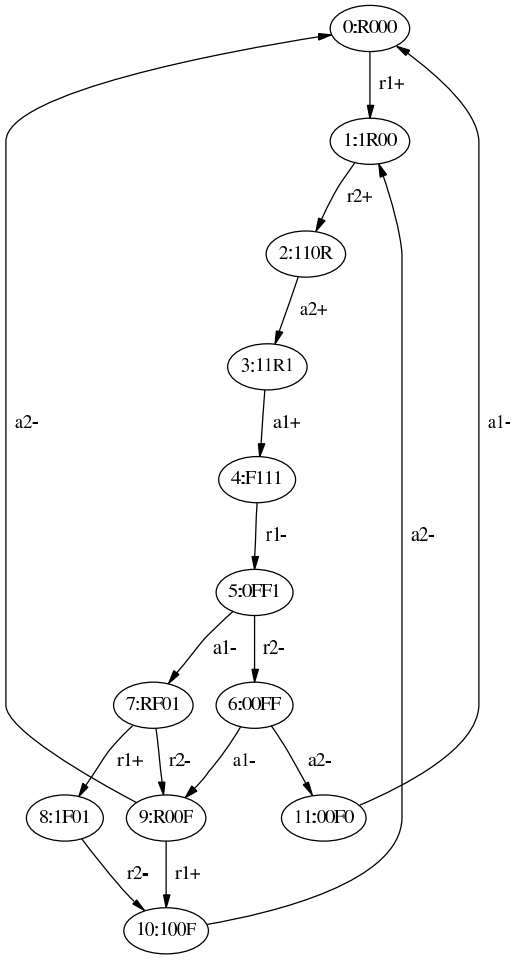


Fig. 4.22 State graph for Problems 4.15.

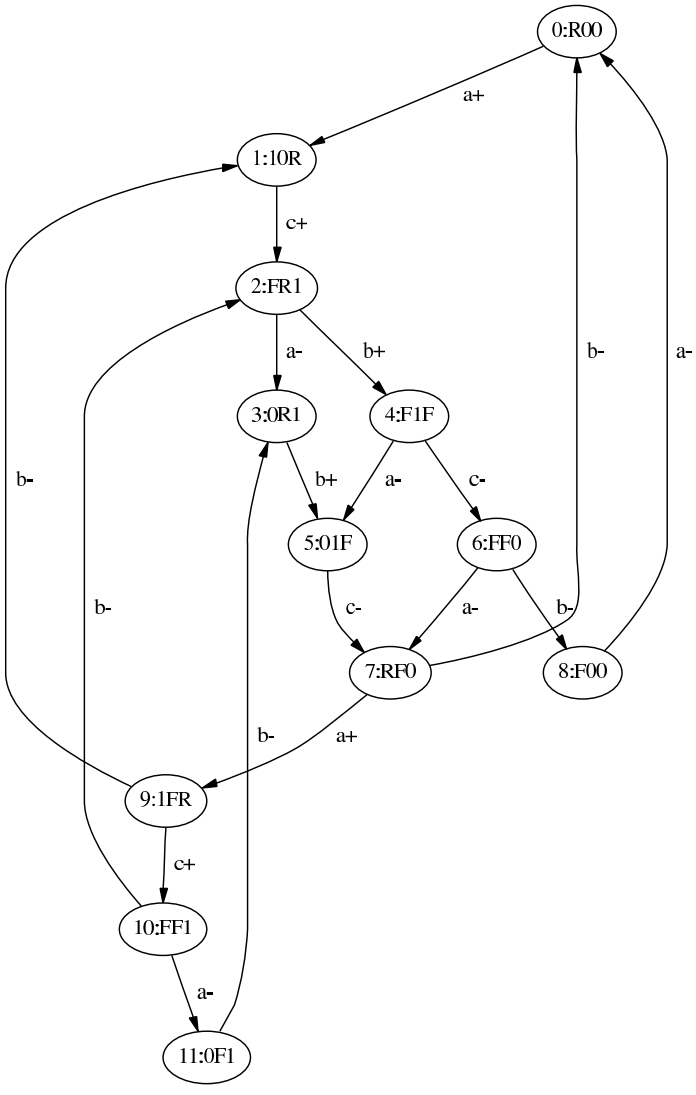


Fig. 4.23 State graph for Problems 4.16.

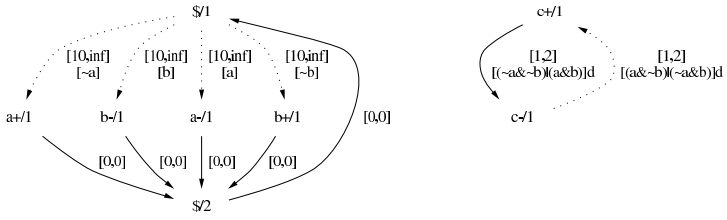


Fig. 4.24 TEL for Problem 4.17.

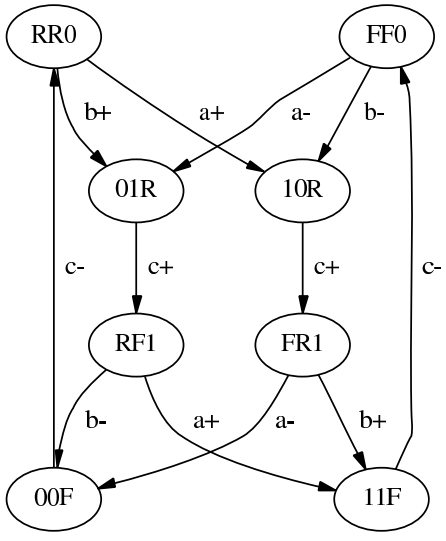


Fig. 4.25 State graph for Problem 4.17

Solution shown in Figure 4.27.

```
shopPA_dual_rail:process
begin
  guard_or(bottle0,'1',bottle1,'1');
  if bottle0 = '1' then assign(shelf0,'1',1,2);
  elsif bottle1 = '1' then assign(shelf1,'1',1,2);
  end if;
  guard(ack_patron,'1');
  assign(ack_wine,'1',1,2,shelf0,'0',1,2,shelf1,'0',1,2);
  guard_and(bottle0,'0',bottle1,'0',ack_patron,'0');
  assign(ack_wine,'0',1,2);
end process;
```

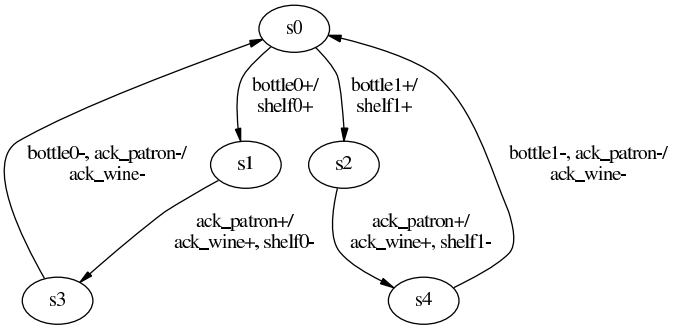


Fig. 4.26 BM machine for Problem 4.18

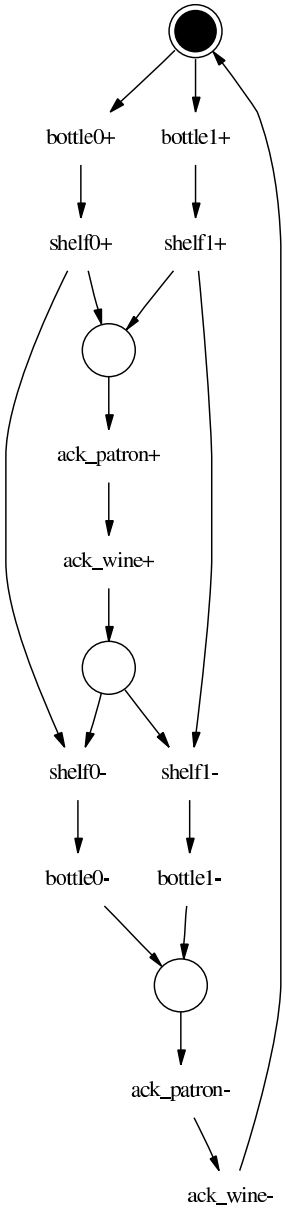


Fig. 4.27 STG for Problem 4.18

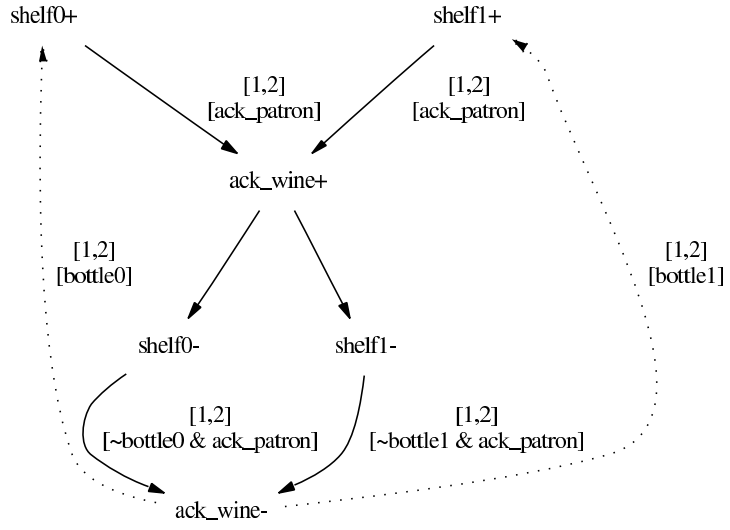


Fig. 4.28 TEL structure for Problem 4.18. It has conflicts $shelf0+\#shelf1+$ and $shelf0-\#shelf1-$.

5

Huffman Circuits

Problems

5.1 Binate Covering

Solve the following constraint matrix using the *bcp* algorithm.

$$\mathbf{A} = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} & c_{11} \end{matrix} \\ \left[\begin{array}{ccccccccccc} - & - & - & - & - & - & - & - & - & 1 & - \\ 1 & - & - & 1 & - & - & - & - & - & - & - \\ 1 & 1 & - & 1 & 1 & 1 & 1 & - & - & - & - \\ 1 & - & 1 & - & 1 & - & - & - & 1 & - & - \\ - & - & 1 & - & - & 1 & - & - & - & - & - \\ - & 1 & - & - & - & 1 & - & 1 & - & - & 1 \\ - & 1 & - & - & - & - & 1 & 1 & - & 1 & - \\ - & - & 1 & - & - & - & - & - & 1 & - & - \\ 0 & - & - & - & - & - & - & 1 & - & - & - \\ 0 & - & - & - & - & - & - & - & - & 1 & - \\ 0 & - & 1 & - & - & - & - & - & 1 & - & - \\ 1 & 0 & - & - & 1 & - & - & - & - & - & - \\ - & 0 & 1 & - & - & - & - & - & - & - & 1 \\ - & - & 0 & - & - & - & - & - & - & 1 & - \\ - & - & 0 & - & - & - & 1 & - & - & - & - \\ - & - & - & - & 0 & - & - & - & - & 1 & - \\ - & - & 1 & - & 0 & - & - & - & 1 & - & - \\ 1 & - & - & - & 1 & 0 & - & - & - & - & - \\ 1 & - & - & - & 1 & - & 0 & - & - & - & - \\ - & 1 & - & - & - & - & 0 & 1 & - & - & - \\ - & - & 1 & - & - & - & - & 0 & - & - & 1 \end{array}\right] \end{array} \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ 16 \\ 17 \\ 18 \\ 19 \\ 20 \\ 21 \end{array}$$

Solution: c_1, c_3, c_7, c_8, c_{10}.

5.2 Binate Covering

Implement the BCP algorithm in your favorite programming language. It should read in a constraint matrix and output a list of columns used in the best solution found.

C-code solution available on request from author.

5.3 State Minimization

For the flow table shown in Figure 5.1:

	00	01	11	10
1	–	2,0	–	5,1
2	–	1,0	3, –	–
3	4,1	3,–	1,1	5,–
4	–	–	2,–	1,–
5	6,–	3,–	2,1	–,0
6	5,–	6,1	1,–	2,–

Fig. 5.1 Flow table for Problem 5.3.

5.3.1. Find compatible pairs using a pair chart.

2	~				
3	2,3	1,3			
4	×	2,3	×		
5	×	1,3 2,3	1,2 4,6	~	
6	×	×	2,5 4,5	1,2	1,2 3,6
	1	2	3	4	5

5.3.2. Compute the maximal compatibles.
{ 123, 235, 245, 356, 456 }
5.3.3. Set up and solve BCP using only the maximal compatibles.

$$\mathbf{A} = \begin{array}{c} \begin{matrix} 123 & 235 & 245 & 356 & 456 \end{matrix} \\ \left[\begin{array}{ccccc} 1 & - & - & - & - \\ 1 & 1 & 1 & - & - \\ 1 & 1 & - & 1 & - \\ - & - & 1 & - & 1 \\ - & 1 & 1 & 1 & 1 \\ - & - & - & 1 & 1 \\ 1 & 0 & - & - & - \\ 1 & - & 0 & - & - \\ 1 & 1 & 0 & - & - \\ 1 & - & - & 0 & - \\ - & - & - & 0 & 1 \\ - & 1 & 1 & 0 & - \\ - & - & 1 & 0 & 1 \\ 1 & - & - & - & 0 \\ - & - & - & 1 & 0 \end{array}\right] \end{array} \begin{matrix} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{matrix}$$

{ 123, 245, 356, 456 }
5.3.4. Compute the prime compatibles.
{ 123, 235, 245, 356, 456, 24, 35, 36, 45, 46, 6 }
5.3.5. Set up and solve BCP using the prime compatibles.

$$\mathbf{A} = \begin{array}{c} \begin{array}{cccccccccc} 123 & 235 & 245 & 356 & 456 & 24 & 35 & 36 & 45 & 46 \end{array} \\ \left[\begin{array}{cccccccccc} 1 & - & - & - & - & - & - & - & - & - \\ 1 & 1 & 1 & - & - & 1 & - & - & - & - \\ 1 & 1 & - & 1 & - & - & 1 & 1 & - & - \\ - & - & 1 & - & 1 & 1 & - & - & 1 & 1 \\ - & 1 & 1 & 1 & 1 & - & 1 & - & 1 & - \\ - & - & - & 1 & 1 & - & - & 1 & - & 1 \\ 1 & 0 & - & - & - & - & - & - & - & - \\ - & 0 & - & - & 1 & - & - & - & 1 & - \\ 1 & - & 0 & - & - & - & - & - & - & - \\ 1 & 1 & 0 & - & - & - & - & - & - & - \\ 1 & - & - & 0 & - & - & - & - & - & - \\ - & - & - & 0 & 1 & - & - & - & 1 & - \\ - & 1 & 1 & 0 & - & - & - & - & - & - \\ - & - & 1 & 0 & 1 & - & - & 1 & - & - \\ 1 & - & - & - & 0 & - & - & - & - & - \\ - & - & - & 1 & 0 & - & 1 & - & - & - \\ 1 & 1 & - & - & - & 0 & - & - & - & - \\ 1 & - & - & - & - & - & 0 & - & - & - \\ - & - & - & - & 1 & - & 0 & - & 1 & - \\ - & 1 & 1 & - & - & - & - & 0 & - & - \\ - & - & 1 & - & 1 & - & - & 0 & 1 & - \\ 1 & - & - & - & - & - & - & - & 0 & - \end{array}\right] \end{array} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ 16 \\ 17 \\ 18 \\ 19 \\ 20 \\ 21 \end{array}$$

{ 123, 245, 46 }

5.3.6. Form the reduced table.

	00	01	11	10
A(123)	C,1	A,0	A,1	B,1
B(245)	C,–	A,0	A,1	A,0
C(46)	B,–	C,1	A,–	A,–

5.3.7. Compare the results from 5.3.3 and 5.3.5.

Using primes allows a three state solution as opposed to a four state solution.

5.4 State Minimization

For the flow table shown in Figure 5.2:

5.4.1. Find compatible pairs using a pair chart.

2	1,4 3,6				
3	2,3	~			
4	2,3	×	~		
5	2,3	1,4	×	~	
6	3,4 1,5	×	~	×	1,5
	1	2	3	4	5

	00	01	11	10
1	3,–	2,–	1,1	1,–
2	6,–	–	4,1	1,–
3	–	3,1	–	–,0
4	2,1	–	–	–,0
5	–	3,0	1,–	–
6	4,0	–	5,–	–

Fig. 5.2 Flow table for Problem 5.4.

5.4.2. Compute the maximal compatibles.
{ 123, 125, 134, 136, 145, 156 }
5.4.3. Set up and solve BCP using only the maximal compatibles.

$$\mathbf{A} = \begin{array}{c} \begin{array}{cccccc} 156 & 145 & 134 & 136 & 123 & 125 \end{array} \\ \left[\begin{array}{cccccc} 1 & 1 & 1 & 1 & 1 & 1 \\ – & – & – & – & 1 & 1 \\ – & – & 1 & 1 & 1 & – \\ – & 1 & 1 & – & – & – \\ 1 & 1 & – & – & – & 1 \\ 1 & – & – & 1 & – & – \\ 0 & – & 1 & – & – & – \\ 0 & – & – & – & 1 & – \\ – & 0 & – & – & 1 & – \\ – & – & 0 & – & 1 & – \\ – & – & 1 & 0 & – & – \\ – & – & – & 0 & 1 & – \\ 1 & 1 & – & 0 & – & 1 \\ – & – & – & 1 & 0 & – \\ – & 1 & 1 & – & 0 & – \\ – & – & – & 1 & – & 0 \\ – & – & – & – & 1 & 0 \\ – & 1 & 1 & – & – & 0 \end{array}\right] \end{array} \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ 16 \\ 17 \\ 18 \end{array}$$

{ 156, 134, 136, 123 }
5.4.4. Compute the prime compatibles.
{ 156, 145, 134, 136, 123, 125, 16, 56, 45, 34, 36, 23, 25, 1 }
5.4.5. Set up and solve BCP using the prime compatibles.

$$\mathbf{A} = \begin{array}{c} \begin{array}{cccccccccccccc} 156 & 145 & 134 & 136 & 123 & 125 & 16 & 56 & 45 & 34 & 36 & 23 & 25 & 1 \end{array} \\ \left[\begin{array}{cccccccccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & - & - & - & - & - & - & 1 \\ - & - & - & - & 1 & 1 & - & - & - & - & - & 1 & 1 & - \\ - & - & 1 & 1 & 1 & - & - & - & - & 1 & 1 & 1 & - & - \\ - & 1 & 1 & - & - & - & - & - & 1 & 1 & - & - & - & - \\ 1 & 1 & - & - & - & 1 & - & 1 & 1 & - & - & - & 1 & - \\ 1 & - & - & 1 & - & - & 1 & 1 & - & - & 1 & - & - & - \\ 0 & - & 1 & - & - & - & - & - & - & 1 & - & - & - & - \\ 0 & - & - & - & 1 & - & - & - & - & - & - & 1 & - & - \\ - & 0 & - & - & 1 & - & - & - & - & - & - & 1 & - & - \\ - & - & 0 & - & 1 & - & - & - & - & - & - & 1 & - & - \\ - & - & 1 & 0 & - & - & - & - & - & 1 & - & - & - & - \\ - & - & - & 0 & 1 & - & - & - & - & - & - & 1 & - & - \\ 1 & 1 & - & 0 & - & 1 & - & - & - & - & - & - & - & - \\ - & - & - & 1 & 0 & - & - & - & - & - & 1 & - & - & - \\ - & 1 & 1 & - & 0 & - & - & - & - & - & - & - & - & - \\ - & - & - & 1 & - & 0 & - & - & - & - & 1 & - & - & - \\ - & - & - & - & 1 & 0 & - & - & - & - & - & 1 & - & - \\ - & 1 & 1 & - & - & 0 & - & - & - & - & - & - & - & - \\ - & - & 1 & - & - & - & 0 & - & - & 1 & - & - & - & - \\ 1 & 1 & - & - & - & 1 & 0 & - & - & - & - & - & - & - \\ 1 & 1 & - & - & - & 1 & - & 0 & - & - & - & - & - & - \\ - & 1 & 1 & - & - & - & - & - & - & - & - & 0 & - & - \end{array}\right] \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ 16 \\ 17 \\ 18 \\ 19 \\ 20 \\ 21 \\ 22 \end{array} \end{array}$$

{ 145, 36, 23 }

5.4.6. Form the reduced table.

	00	01	11	10
A (145)	C,1	C,0	A,1	A,0
B (36)	A,0	C,1	A,–	–,0
C (23)	B,–	C,1	A,1	A,0

5.4.7. Compare the results from 5.4.3 and 5.4.5.

The result using primes requires only three states as opposed to four when using maximals.

5.5 State Minimization

For the flow table shown in Figure 5.3:

5.5.1. Find compatible pairs using a pair chart.

2	3,6				
3	×	×			
4	×	1,6	×		
5	~	1,2	×	1,5	
6	1,2	×	4,6	4,5	×
	1	2	3	4	5

	00	01	11	10
1	3,0	1,–	–	–
2	6,–	2,0	1,–	–
3	–,1	–	4,0	–
4	1,0	–	–	5,1
5	–	5,–	2,1	1,1
6	–	2,1	6,–	4,1

Fig. 5.3 Flow table for Problem 5.5.

5.5.2. Compute the maximal compatibles.
{ 125, 16, 245, 36, 46 }
5.5.3. Compute the prime compatibles.
{ 125, 15, 16, 2, 24, 25, 245, 3, 36, 4, 45, 46, 6 }
5.5.4. Set up and solve BCP using the prime compatibles.

$$\mathbf{A} = \begin{array}{c} \begin{array}{cccccccccccccc} 125 & 15 & 16 & 2 & 24 & 25 & 245 & 3 & 36 & 4 & 45 & 46 & 6 & \end{array} \\ \left[\begin{array}{cccccccccccccc} 1 & 1 & 1 & – & – & – & – & – & – & – & – & – & – \\ 1 & – & – & 1 & 1 & 1 & 1 & – & – & – & – & – & – \\ – & – & – & – & – & – & – & 1 & 1 & – & – & – & – \\ – & – & – & – & 1 & – & 1 & – & – & 1 & 1 & 1 & – \\ 1 & 1 & – & – & – & 1 & 1 & – & – & – & 1 & – & – \\ – & – & 1 & – & – & – & – & – & 1 & – & – & 1 & 1 \\ 0 & – & – & – & – & – & – & – & 1 & – & – & – & – \\ 1 & – & 0 & – & – & – & – & – & – & – & – & – & – \\ – & – & – & 1 & 0 & – & – & – & – & – & – & – & – \\ 1 & – & – & – & – & 0 & – & – & – & – & – & – & – \\ 1 & – & – & – & – & – & 0 & – & – & – & – & – & – \\ 1 & 1 & – & – & – & – & 0 & – & – & – & – & – & – \\ – & – & 1 & – & – & – & 0 & – & – & – & – & – & – \\ – & – & – & – & – & – & – & – & 0 & – & – & 1 & – \\ 1 & 1 & – & – & – & – & – & – & – & – & 0 & – & – \\ – & – & – & – & – & – & – & – & – & – & 1 & 0 & – \end{array}\right] \end{array} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ 16 \end{array}$$

{ 125, 36, 45, 46 }
5.5.5. Form the reduced table.

	00	01	11	10
A (125)	B,0	A,0	A,1	A,1
B (36)	–,1	A,1	D,0	D,1
C (45)	A,0	A,–	A,1	A,1
D (46)	A,0	A,1	D,–	C,1

5.6 State Assignment

For the flow table shown in Figure 5.4:

	00	01	11	10
1	1,0	2,0	3,0	1,0
2	1,0	2,0	3,0	2,1
3	3,1	5,0	3,0	4,0
4	1,0	–,–	4,0	4,0
5	3,1	5,0	5,1	4,0

Fig. 5.4 Flow table for Problem 5.6.

5.6.1. Find a state assignment without using the outputs.
Partition list:

$$
\begin{aligned}
\pi_1 &= \{12;35\} \\
\pi_2 &= \{14;35\} \\
\pi_3 &= \{13;4\} \\
\pi_4 &= \{13;5\} \\
\pi_5 &= \{23;4\} \\
\pi_6 &= \{23;5\} \\
\pi_7 &= \{34;1\} \\
\pi_8 &= \{34;2\} \\
\pi_9 &= \{45;1\} \\
\pi_{10} &= \{45;2\} \\
\pi_{11} &= \{1;2\}
\end{aligned}
$$

Boolean matrix:

	1	2	3	4	5
π_1	1	1	0	–	0
π_2	1	–	0	1	0
π_3	0	–	0	1	–
π_4	0	–	0	–	1
π_5	–	0	0	1	–
π_6	–	0	0	–	1
π_7	1	–	0	0	–
π_8	–	1	0	0	–
π_9	1	–	–	0	0
π_{10}	–	1	–	0	0
π_{11}	0	1	–	–	–

Pairwise intersectibles:

	π_1	π_2	π_3	π_4	π_5	π_6	π_7	π_8	π_9	π_{10}	π_{11}	$\overline{\pi_9}$	$\overline{\pi_{10}}$
π_2	∼												
π_3	×	×											
π_4	×	×	∼										
π_5	×	∼	∼	∼									
π_6	×	×	∼	∼	∼								
π_7	∼	×	×	×	×	∼							
π_8	∼	×	×	∼	×	×	∼						
π_9	∼	×	×	×	×	×	∼	∼					
π_{10}	∼	×	×	×	×	×	∼	∼	∼				
π_{11}	×	×	∼	∼	×	×	×	∼	×	∼			
$\overline{\pi_9}$	×	×	∼	∼	∼	∼	×	×	×	×	∼		
$\overline{\pi_{10}}$	×	×	∼	∼	∼	∼	×	×	×	×	×	∼	
$\overline{\pi_{11}}$	×	∼	×	×	∼	∼	∼	×	∼	×	×	×	∼

Maximal intersectibles:

x_1	(π_1, π_2)
x_2	$(\pi_1, \pi_7, \pi_8, \pi_9, \pi_{10})$
x_3	$(\pi_2, \pi_5, \overline{\pi_{11}})$
x_4	(π_3, π_{11})
x_5	$(\pi_3, \pi_4, \pi_5, \pi_6, \overline{\pi_9}, \overline{\pi_{10}})$
x_6	$(\pi_3, \pi_4, \overline{\pi_9}, \pi_{11})$
x_7	(π_3, π_6, π_{11})
x_8	(π_4, π_8, π_{11})
x_9	$(\pi_5, \pi_6, \overline{\pi_{10}}, \overline{\pi_{11}})$
x_{10}	$(\pi_6, \pi_7, \overline{\pi_{11}})$
x_{11}	$(\pi_8, \pi_{10}, \pi_{11})$

Setting up the covering problem:

$$\mathbf{A} = \begin{array}{c} \begin{matrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & x_{11} \end{matrix} \\ \left[\begin{array}{ccccccccccc} 1 & 1 & - & - & - & - & - & - & - & - & - \\ 1 & - & 1 & - & - & - & - & - & - & - & - \\ - & - & - & 1 & 1 & 1 & 1 & - & - & - & - \\ - & - & - & - & 1 & 1 & - & 1 & - & - & - \\ - & - & 1 & - & 1 & - & - & - & 1 & - & - \\ - & - & - & - & 1 & - & 1 & - & 1 & 1 & - \\ - & 1 & - & - & - & - & - & - & - & 1 & - \\ - & 1 & - & - & - & - & - & 1 & - & - & - \\ - & 1 & - & - & 1 & 1 & - & - & - & - & - \\ - & 1 & - & - & 1 & - & - & - & 1 & - & 1 \\ - & - & 1 & 1 & - & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \end{matrix}$$

Solution: x_2, x_3, x_5
Reduced Boolean Matrix:

	1	2	3	4	5
x_2	1	1	0	0	0
x_3	1	0	0	1	0
x_5	0	0	0	1	1

State assigned flow table:

	00	01	11	10
110	110,0	100,0	000,0	110,0
100	110,0	100,0	000,0	100,1
000	000,1	001,0	000,0	011,0
011	110,0	−,−	011,0	011,0
001	000,1	001,0	001,1	011,0

5.6.2. Find a state assignment using the outputs as state variables.
This has no use, so the state assignment is the same.

5.7 State Assignment

For the flow table shown in Figure 5.5:

	00	01	11	10
1	1,0	2,0	3,0	1,0
2	4,0	2,0	2,0	2,1
3	4,1	2,0	3,0	3,0
4	4,0	5,0	3,0	4,0
5	1,1	5,0	5,1	5,0

Fig. 5.5 Flow table for Problem 5.7.

5.7.1. Find a state assignment without using the outputs.
Partition list:

$$\begin{aligned}
\pi_1 &= \{15;24\} \\
\pi_2 &= \{15;34\} \\
\pi_3 &= \{45;12\} \\
\pi_4 &= \{45;23\} \\
\pi_5 &= \{13;2\} \\
\pi_6 &= \{34;2\}
\end{aligned}$$

Boolean matrix:

	1	2	3	4	5
π_1	0	1	–	1	0
π_2	0	–	1	1	0
π_3	0	0	–	1	1
π_4	–	0	0	1	1
π_5	0	1	0	–	–
π_6	–	0	1	1	–

Pairwise intersectibles:

π_2	$\sim$					
π_3	$\times$	$\times$				
π_4	$\times$	$\times$	$\sim$			
π_5	$\sim$	$\times$	$\times$	$\times$		
π_6	$\times$	$\sim$	$\sim$	$\times$	$\times$	
$\overline{\pi_5}$	$\times$	$\times$	$\times$	$\times$	$\times$	$\sim$
	π_1	π_2	π_3	π_4	π_5	π_6

Maximal intersectibles:

x_1	(π_1, π_2)
x_2	(π_1, π_5)
x_3	(π_2, π_6)
x_4	(π_3, π_4)
x_5	$(\overline{\pi_5}, \pi_6)$
x_6	(π_3, π_6)

Setting up the covering problem:

$$\mathbf{A} = \begin{array}{c} \begin{matrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \end{matrix} \\ \begin{bmatrix} 1 & 1 & – & – & – & – \\ 1 & – & 1 & – & – & – \\ – & – & – & 1 & – & 1 \\ – & – & – & 1 & – & – \\ – & 1 & – & – & 1 & – \\ – & – & 1 & – & 1 & 1 \end{bmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix}$$

Solution: x_1, x_4, x_5
Reduced Boolean Matrix:

	1	2	3	4	5
x_1	0	1	1	1	0
x_4	0	0	0	1	1
x_5	1	0	1	1	0

State assigned flow table:

	00	01	11	10
001	001,0	100,0	101,0	001,0
100	111,0	100,0	100,0	100,1
101	111,1	100,0	101,0	101,0
111	111,0	010,0	101,0	111,0
010	001,1	010,0	010,1	010,0

5.7.2. Find a state assignment using the outputs as state variables.
This has no use, so the state assignment is the same.

5.8 State Assignment

For the flow table shown in Figure 5.6:

	00	01	11	10
1	1,0	3,0	7,0	1,0
2	5,0	2,0	6,0	2,1
3	1,1	3,0	7,0	3,0
4	4,0	7,0	6,0	4,0
5	5,1	7,0	6,1	1,0
6	1,0	2,0	6,0	3,0
7	5,1	7,1	7,1	2,1

Fig. 5.6 Flow table for Problem 5.8.

5.8.1. Find a state assignment without using the outputs.
THIS PROBLEM IS TOO LARGE. IT SHOULD NOT BE ASSIGNED.

5.8.2. Find a state assignment using the outputs as state variables.
THIS PROBLEM IS TOO LARGE. IT SHOULD NOT BE ASSIGNED.

5.9 Two-Level Logic Minimization

Do the following for the Karnaugh map shown in Figure 5.7:

		ab			
		00	01	11	10
cd	00	1	1	1	1
	01	1	1	1	0
	11	0	1	1	0
	10	0	0	1	1

Fig. 5.7 Karnaugh map for Problem 5.9.

5.9.1. Find all prime implicants using the recursive procedure.
primes $= a\,\overline{d} + ab + \overline{a}\,\overline{c} + \overline{c}\,\overline{d} + b\,\overline{c} + bd$

5.9.2. Set up and solve a covering problem to pick the minimal number of prime implicants, ignoring hazards.
$f = a\,\overline{d} + \overline{a}\,\overline{c} + bd$

5.9.3. Set up and solve a covering problem to pick the minimal number of prime implicants for a hazard-free cover assuming SIC.
primes $= a\,\overline{d} + ab + \overline{a}\,\overline{c} + \overline{c}\,\overline{d} + bd$

5.9.4. Assume that the only transitions possible are

$$\begin{aligned} t_1 &= [\overline{a}\,b\,\overline{c}\,d, \overline{a}\,\overline{b}\,\overline{c}\,\overline{d}] \\ t_2 &= [a\,b\,\overline{c}\,\overline{d}, a\,\overline{b}\,\overline{c}\,d] \\ t_3 &= [\overline{a}\,\overline{b}\,c\,\overline{d}, \overline{a}\,b\,c\,d] \\ t_4 &= [a\,b\,c\,\overline{d}, a\,\overline{b}\,c\,d] \end{aligned}$$

Identify the type of each transition and its transition cube.
t_1 is a static $1 \to 1$ with transition cube $\overline{a}\,\overline{c}$.
t_2 is a dynamic $1 \to 0$ with transition cube $a\,\overline{c}$.
t_3 is a dynamic $0 \to 1$ with transition cube $\overline{a}\,c$.
t_4 is a dynamic $1 \to 0$ with transition cube ac.

5.9.5. Determine all required cubes for the transitions above.
t_1's required cube is $\overline{a}\,\overline{c}$.
t_2's required cubes are $a\,\overline{c}\,\overline{d}$ and $ab\,\overline{c}$.
t_3's required cube is $\overline{a}\,bcd$.
t_4's required cubes are abc and $ac\,\overline{d}$.

5.9.6. Determine all privileged cubes for the transitions above.
t_2 has privileged cube $a\,\overline{c}$.
t_4 has privileged cube ac.
t_3 has privileged cube $\overline{a}\,c$, though this one is optional since this $0 \to 1$ transition has a single minterm in the ON-set, so cannot be illegally intersected.

5.9.7. Find the dhf-prime implicants.
dhf-primes $= a\,\overline{d} + ab + \overline{a}\,\overline{c} + \overline{c}\,\overline{d} + \overline{a}\,bd + b\,\overline{c}$

5.9.8. Set up and solve a covering problem to pick the minimal number of prime implicants for a hazard-free cover assuming that the only transitions are those given above.
$f = a\,\overline{d} + ab + \overline{a}\,\overline{c} + \overline{a}\,bd$

5.10 Two-Level Logic Minimization

Do the following for the Karnaugh map shown in Figure 5.8:

5.10.1. Find all prime implicants using the recursive procedure.
$a\,\overline{b}$, $\overline{b}\,c$, and $\overline{a}\,c\,\overline{d}$.

5.10.2. Set up and solve a covering problem to pick the minimal number of prime implicants, ignoring hazards.
$a\,\overline{b}$, $\overline{b}\,c$, and $\overline{a}\,c\,\overline{d}$.

5.10.3. Set up and solve a covering problem to pick the minimal number of prime implicants for a hazard-free cover assuming SIC.
$a\,\overline{b}$, $\overline{b}\,c$, and $\overline{a}\,c\,\overline{d}$.

cd \ ab	00	01	11	10
00	0	0	0	1
01	0	0	0	1
11	1	0	0	1
10	1	1	0	1

Fig. 5.8 Karnaugh map for Problem 5.10.

5.10.4. Assume that the only transitions possible are:

$$\begin{aligned} t_1 &= [\overline{a}\,\overline{b}\,\overline{c}\,\text{-}, \overline{a}\,\overline{b}\,c\,\text{-}] \\ t_2 &= [\overline{a}\,b\,c\,\overline{d}, \overline{a}\,b\,\overline{c}\,\overline{d}] \\ t_3 &= [a\,b\,\text{-}\,\text{-}, a\,\overline{b}\,\text{-}\,\text{-}] \end{aligned}$$

Identify the type of each transition and its transition cube. Assume that d is a level signal and that c is a falling directed don't care in t_3.

t_1 is a dynamic $0 \rightarrow 1$ transition with transition cube $[\overline{a}\,\overline{b}\,\text{-}\,\text{-}]$

t_2 is a dynamic $1 \rightarrow 0$ transition with transition cube $[\overline{a}\,b\,\text{-}\,\overline{d}]$.

t_3 is a dynamic $0 \rightarrow 1$ transition with transition cube $[a\,\text{-}\,\text{-}\,\text{-}]$.

5.10.5. Determine all required cubes for the transitions above.

t_1's required cube is $[\overline{a}\,\overline{b}\,c\,\text{-}]$

t_2's required cube is $[\overline{a}\,b\,c\,\overline{d}]$.

t_3's required cube is $[a\,\overline{b}\,\text{-}\,\text{-}]$.

5.10.6. Determine all privileged cubes for the transitions above.

t_1 priveledged cube is $[\overline{a}\,\overline{b}\,\text{-}\,\text{-}]$

t_2 priveledged cube is $[\overline{a}\,b\,\text{-}\,\overline{d}]$.

t_3 priveledged cube is $[a\,\text{-}\,\text{-}\,\text{-}]$.

5.10.7. Find the dhf-prime implicants.

$a\,\overline{b}$, $\overline{a}\,\overline{b}\,c$, and $\overline{a}\,b\,c\,\overline{d}$.

5.10.8. Set up and solve a covering problem to pick the minimal number of prime implicants for a hazard-free cover assuming that the only transitions are those given above.

$a\,\overline{b}$, $\overline{a}\,\overline{b}\,c$, and $\overline{a}\,b\,c\,\overline{d}$.

5.11 Burst-Mode Synthesis

Do the following for the BM machine shown in Figure 5.9:

5.11.1. Translate the BM machine into a flow table.

See solution in Figure 5.10.

5.11.2. Perform state minimization on the flow table. Be sure to consider BM operation.

See solution in Figure 5.11.

5.11.3. Perform state assignment on the reduced flow table.

Many possible solutions. One example is $s0 = 001$, $s1 = 011$, $s2 = 100$, and $s3 = 010$.

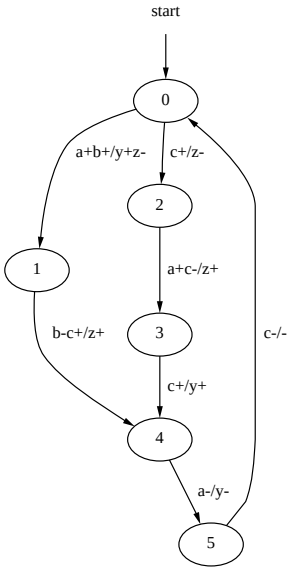


Fig. 5.9 BM machine for Problem 5.11. Note that $abc = 000$ and $yz = 01$ initially.

	a / b / c							
	000	001	011	010	110	111	101	100
s0	(s0), 01	s2, 00	–	(s0), 01	s1, 10	–	–	(s0), 01
s1	–	–	–	–	(s1), 10	(s1), 10	s4, 11	(s1), 10
s2	(s2), 00	(s2), 00	–	–	–	–	(s2), 00	s3, 01
s3	–	–	–	–	–	–	s4, 11	(s3), 01
s4	–	s5, 01	–	–	–	–	(s4), 11	–
s5	s0, 01	(s5), 01	–	–	–	–	–	–
	y / z							

Fig. 5.10 Flow table for Problem 5.11.1

	a / b / c							
	000	001	011	010	110	111	101	100
s0	(s0), 01	s2, 00	–	(s0), 01	s1, 10	–	–	(s0), 01
s1	–	–	–	–	(s1), 10	(s1), 10	s3, 11	(s1), 10
s2	(s2), 00	(s2), 00	–	–	–	–	(s2), 00	s3, 01
s3	s0, 01	(s3), 01	–	–	–	–	(s3), 11	(s3), 01
	y / z							

Fig. 5.11 Flow table for Problem 5.11.2

5.11.4. Perform two-level logic minimization to find hazard-free logic to implement the output signals.

THIS PROBLEM IS TOO LARGE. IT SHOULD NOT BE ASSIGNED.

5.12 Extended Burst-Mode Synthesis

Do the following for the XBM machine shown in Figure 5.12:

5.12.1. Translate the XBM machine into a flow table.

5.12.2. Perform state minimization on the flow table. Be sure to consider XBM operation.

5.12.3. Perform state assignment on the reduced flow table.

5.12.4. Perform two-level logic minimization to find hazard-free logic to implement the output signals.

THIS PROBLEM IS TOO LARGE. IT SHOULD NOT BE ASSIGNED.

5.13 Multilevel Logic Synthesis

Apply hazard-preserving transformations to find a minimum literal factored form for your solution of Problem 5.9.

$f = \overline{a}(\overline{c} + bd) + a(b + \overline{d})$

5.14 Multilevel Logic Synthesis

Apply hazard-preserving transformations to find a minimum literal factored form for your solution of Problem 5.10.

$f = \overline{a}\, c(\overline{b} + \overline{d}) + a\, \overline{b}$

5.15 Multilevel Logic Synthesis

Apply hazard-preserving transformations to find a minimum literal factored form for your solution of Problem 5.11.

Problem 5.11.4 should not be assigned, so neither should this one.

5.16 Multilevel Logic Synthesis

Apply hazard-preserving transformations to find a minimum literal factored form for your solution of Problem 5.12.

Problem 5.12.4 should not be assigned, so neither should this one.

5.17 Technology Mapping

Map your multilevel logic from Problem 5.13 using library from Figure 5.42.

Decomposition, $\overline{\overline{\overline{a}\,\overline{c\,\overline{bd}}}}\;\overline{\overline{a\,\overline{\overline{b}\,b}}}$, is a minimum cost solution.

5.18 Technology Mapping

Map your multilevel logic from Problem 5.14 using library from Figure 5.42.

Decomposition, $\overline{\overline{a\, c(b\,\overline{d})\, a\, \overline{b}}}$ is a minimum cost solution.

5.19 Technology Mapping

Map your multilevel logic from Problem 5.15 using library from Figure 5.42.

Problem 5.11.4 should not be assigned, so neither should this one.

5.20 Technology Mapping

Map your multilevel logic from Problem 5.16 using library from Figure 5.42.

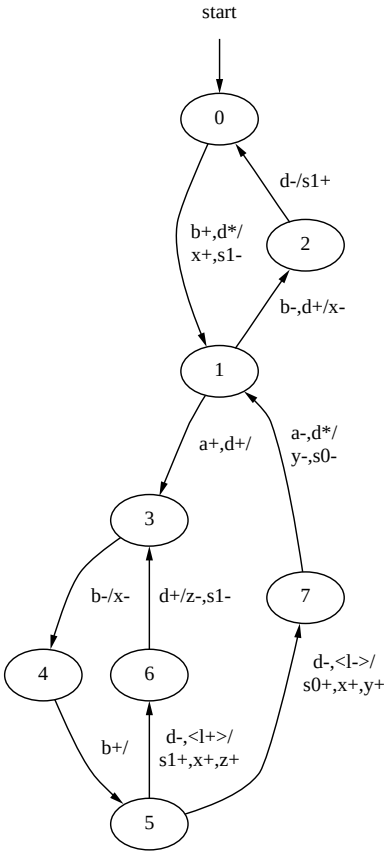


Fig. 5.12 XBM machine for Problems 5.12 and 5.21. Note that $abdl = 000-$ and $s0s1xyz = 01000$ initially.

	00	01	11	10
1	3,1	1,0	2,0	1,1
2	2,0	2,0	2,0	4,0
3	3,1	1,0	3,0	4,0
4	4,0	2,0	3,0	4,0

Fig. 5.13 Flow table for Problem 5.22.

Problem 5.12.4 should not be assigned, so neither should this one.

5.21 Generalized C-Element Synthesis

Do the following for the XBM machine shown in Figure 5.12, targeting a gC implementation.

5.21.1. Translate the XBM machine into a flow table.

5.21.2. Perform state minimization on the flow table. Be sure to consider XBM operation.

5.21.3. Perform state assignment on the reduced flow table.

5.21.4. Perform two-level logic minimization to find hazard-free logic to implement the output signals using generalized C-elements.

5.22 Sequential Hazards

Find all essential hazards in the flow table shown in Figure 5.13.

- Starting in state 1 with input 10, you can end up in either state 3 or state 4 after x goes low due to an essential hazard.
- Starting in state 3 with input 11, you can end up in either state 1 or state 2 after x goes low due to an essential hazard.
- Starting in state 1 with input 10, you can end up in either state 2 or state 4 after y goes high due to an essential hazard.

6

Muller Circuits

Problems

6.1 Speed Independence
For the SG shown in Figure 6.1(a), find its corresponding partial order of equivalence classes. A property is said to hold in a state if it holds in all states that follow it. For each state in the SG, determine which of the following properties hold:

Partial order of equivalence classes shown in Figure 6.1(b).

1. Speed independence

   Holds in all states except 00RR.

2. Semi-modularity

   Holds in all states except 00RR.

3. Distributivity

   Holds in all states except 00RR and 0RR1.

4. Totally sequential

   Holds only in states R010, 1R10, F110, and 0110.

6.2 Complete State Coding
For the STG in Figure 6.2(a), find the state graph and then find all state pairs

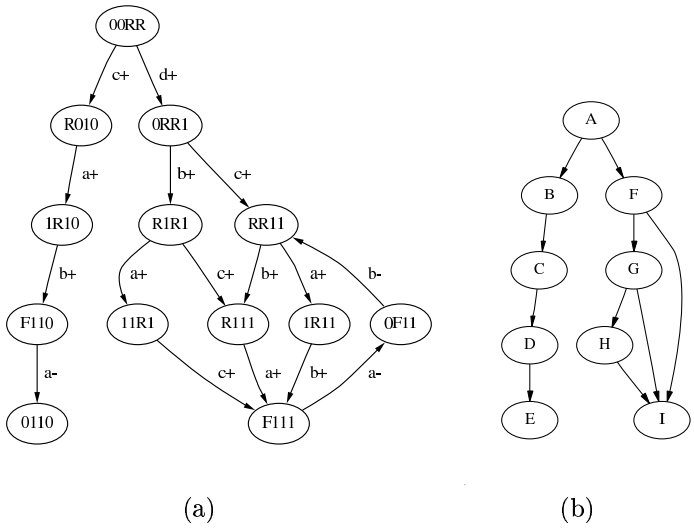


Fig. 6.1 (a) SG for Problem 6.1. (b) Partial order of equivalence classes for Problem 6.1.

which violate complete state coding. Solve the CSC violations by adding a new state variable. Show the new state graph.

The state graph is shown in Figure 6.2(a). The CSC violation pairs are (01F,0F1), (10R,F00), (FF0,1FR), and (F1F,FF1). The new STG and its SG are shown in Figure 6.3.

6.3 Hazard-Free Logic Synthesis

From the SG in Figure 6.4, find a hazard-free logic implementation for the output signals x, d, and c using the:

6.3.1. Atomic gate approach.

```
[ c: (x)] Combinational
[ c: (b & c)] Combinational
[ c: (b & ~a)] Combinational
[ d: (a & ~c & ~x)] Combinational
[ x: (b & d)] Combinational
[ x: (~c & x)] Combinational
```

6.3.2. Generalized C-element approach (use the single-cube algorithm).

```
[+c: (x)]
[+c: (b & ~a)]
[-c: (~b)]
[ d: (a & ~c & ~x)] Combinational
[+x: (b & d)]
[-x: (c & ~d)]
```

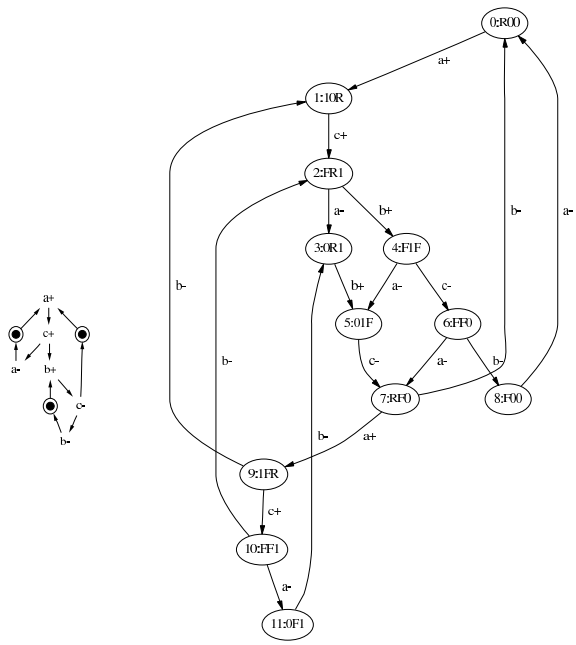


Fig. 6.2 (a) STG for Problem 6.2. (b) SG for Problem 6.2.

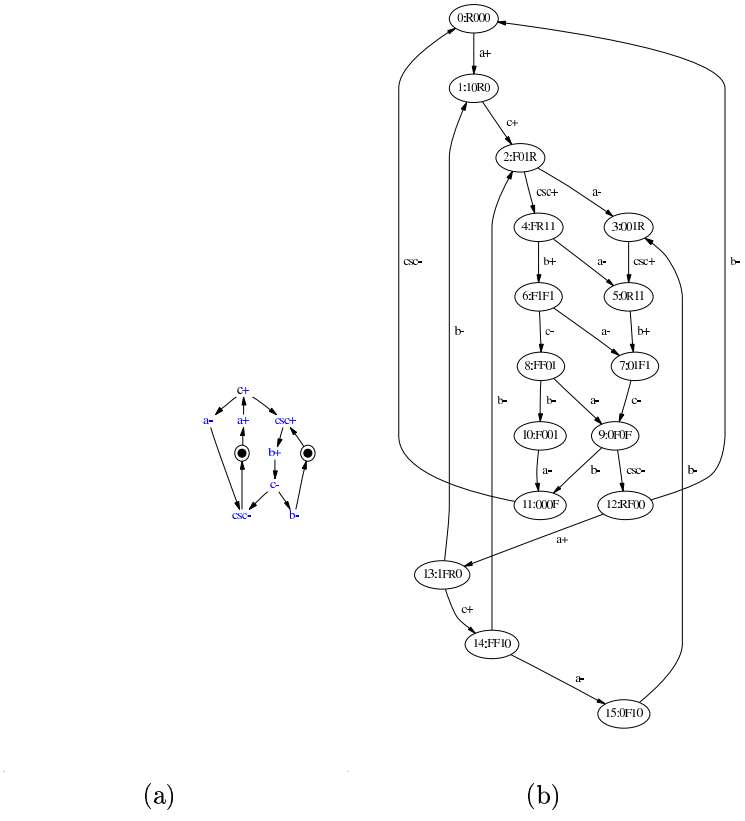


Fig. 6.3 (a) New STG for Problem 6.2. (b) New SG for Problem 6.2.

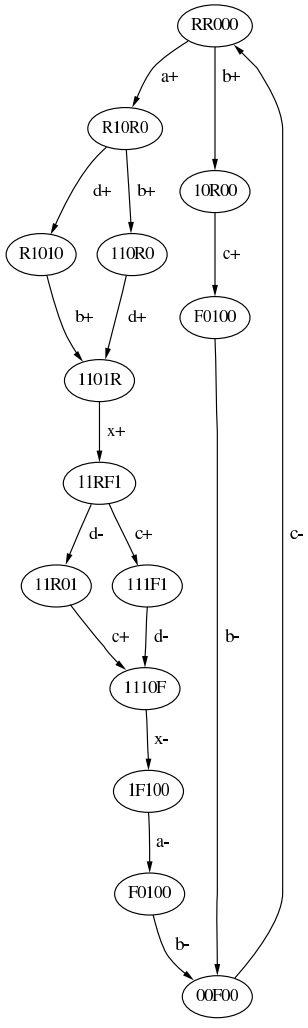


Fig. 6.4 SG for Problem 6.3.

6.3.3. Standard C-element approach (use the single-cube algorithm).

```
[+c: (x)]
[+c: (b & ~a & ~c)]
[-c: (~b)]
[ d: (a & ~c & ~x)] Combinational
[+x: (b & d)]
[-x: (c & ~d & x)]
```

6.4 Exceptions

From the SG shown in Figure 6.5 for the output signals $r2$, $a1$, and x:

6.4.1. Find the excitation cubes and trigger cubes.

```
REGIONS:
STATEVECTOR:INP a2 INP r1 a1 r2 x
EVENT : ENABLED CUBE : TRIGGER CUBE
+a1 : 11010 : 1ZZZZ
-a1 : X01XX : Z0ZZ1
+r2 : 01000 : 01ZZ0
-r2 : 1XX11 : Z0ZZ1
+x : 1X110 : ZZ1ZZ
-x : XXX01 : ZZZ0Z
```

6.4.2. Use the single-cube algorithm to find a standard C-implementation, if possible. If not possible, explain why not.

```
[+x: (a1 & r2)]
[-x: (~r2)]

The signal x is non-persistent for a1-
The signal r1 is non-persistent for r2-
```

6.4.3. Use the more general algorithm to find a multicube cover for the unimplemented signals from 6.4.2.

```
[+a1: (a2 & r2 & ~x)]
[-a1: (~r1 & x)]
[-a1: (~r2)]
[+r2: (~a2 & r1 & ~x)]
[-r2: (~a1 & x)]
[-r2: (~r1 & x)]
[+x: (a1 & r2)]
[-x: (~r2)]
```

6.5 Speed-Independent Design

Perform the following on the VHDL in Figure 6.6.

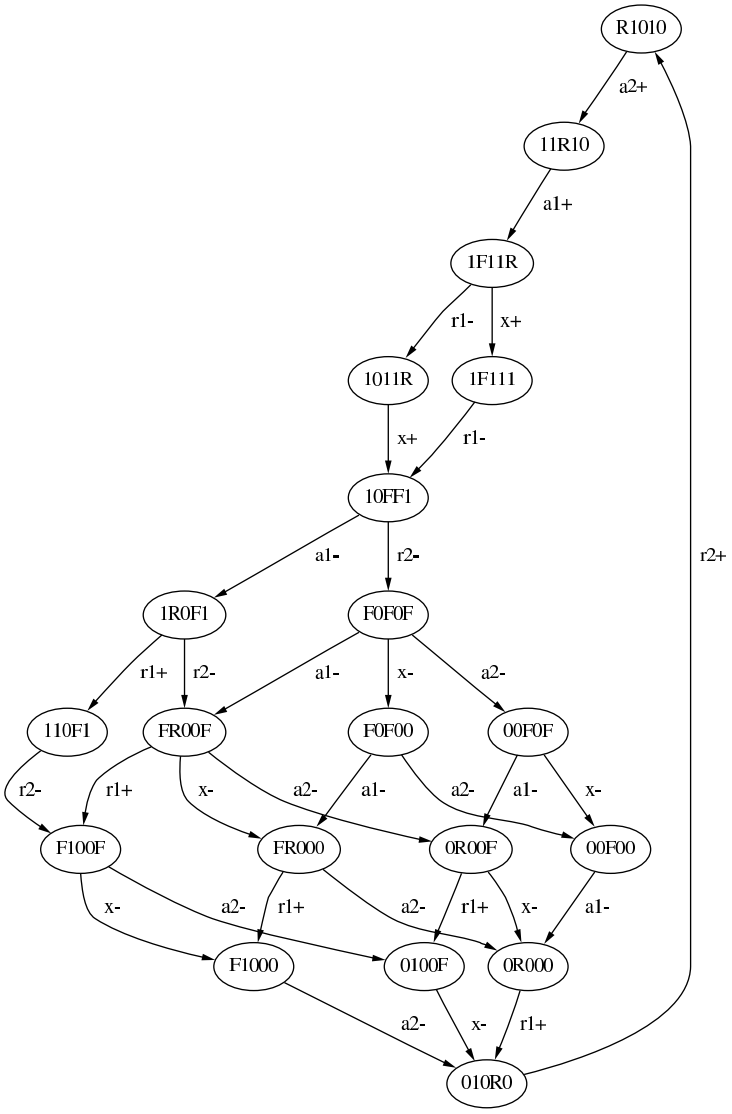


Fig. 6.5 SG for Problem 6.4.

```
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.handshake.all;
entity p6 is
end entity;
architecture hse of p6 is
  signal ai,bi,x:std_logic; --@ in
  signal ao,bo:std_logic;
begin
main:process
begin
  guard(ai,'1');
  if (x = '1') then
    assign(ao,'1',1,2);
    guard(ai,'0');
    assign(ao,'0',1,2);
  else
    assign(bo,'1',1,2);
    guard(bi,'1');
    assign(bo,'0',1,2);
    guard(bi,'0');
    assign(ao,'1',1,2);
    guard(ai,'0');
    assign(ao,'0',1,2);
  end if;
end process;
ai:process
  variable z:integer;
begin
  z:=selection(2);
  if (z=1) then
    assign(x,'1',1,2);
    assign(ai,'1',1,2);
    guard(ao,'1');
    assign(ai,'0',1,2);
    assign(x,'0',1,2);
    guard(ao,'0');
  else
    assign(ai,'1',1,2);
    guard(ao,'1');
    assign(ai,'0',1,2);
    guard(ao,'0');
  end if;
end process;  bi:process
begin
  guard(bo,'1');
  assign(bi,'1',1,2);
  guard(bo,'0');
  assign(bi,'0',1,2);
end process;
end hse;
```

Fig. 6.6 VHDL for Problem 6.5.

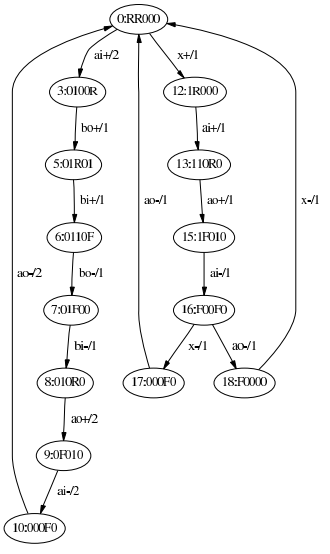


Fig. 6.7 SG for Problem 6.5.1.

6.5.1. Assuming that all signals are initially low, find the state graph.
See Figure 6.7.

6.5.2. Find all state pairs which violate complete state coding.

```
CSC VIOLATIONS:
STATEVECTOR:INP x INP ai INP bi ao bo
STATE1 : STATE2
3:0100R <-> 9:010R0
```

6.5.3. Solve the CSC violations by using reshuffling. Show the new state graph.
See VHDL in Figure 6.8 and SG in Figure 6.9.

6.5.4. Solve the CSC violations by adding a new state variable q. Show the new state graph.
See Figure 6.10.

6.5.5. Use Boolean minimization to find the circuit from either of the solutions above.

```
[ ao: (x & ai)] Combinational
[ ao: (ai & ao)] Combinational
[ ao: (bi)] Combinational
[ bo: (~x & ai & ~ao)] Combinational

[ ao: (x & ai)] Combinational
```

```
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.handshake.all;
entity p6 is
end entity;
architecture hse of p6 is
  signal ai,bi,x:std_logic; --@ in
  signal ao,bo:std_logic;
begin
main:process
begin
  guard(ai,'1');
  if (x = '1') then
    assign(ao,'1',1,2);
    guard(ai,'0');
    assign(ao,'0',1,2);
  else
    assign(bo,'1',1,2);
    guard(bi,'1');
    assign(ao,'1',1,2);
    assign(bo,'0',1,2);
    guard(bi,'0');
    guard(ai,'0');
    assign(ao,'0',1,2);
  end if;
end process;
ai:process
  variable z:integer;
begin
  z:=selection(2);
  if (z=1) then
    assign(x,'1',1,2);
    assign(ai,'1',1,2);
    guard(ao,'1');
    assign(ai,'0',1,2);
    assign(x,'0',1,2);
    guard(ao,'0');
  else
    assign(ai,'1',1,2);
    guard(ao,'1');
    assign(ai,'0',1,2);
    guard(ao,'0');
  end if;
end process;  bi:process
begin
  guard(bo,'1');
  assign(bi,'1',1,2);
  guard(bo,'0');
  assign(bi,'0',1,2);
end process;
end hse;
```

Fig. 6.8 VHDL for Problem 6.5.3.

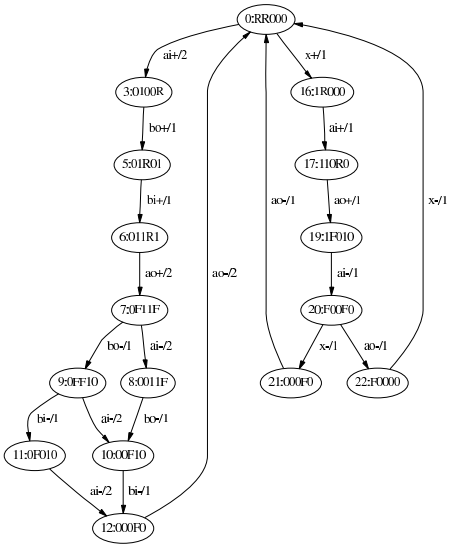


Fig. 6.9 SG for Problem 6.5.3.

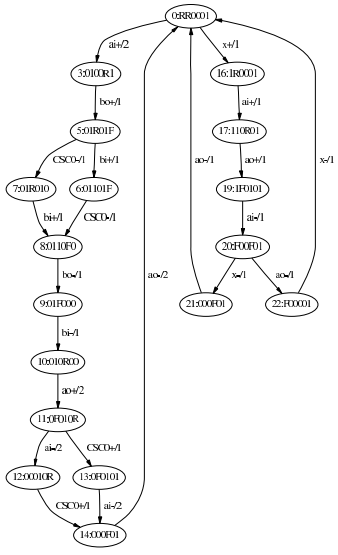


Fig. 6.10 SG for Problem 6.5.4.

```
[ ao: (ai & ao)] Combinational
[ ao: (~bi & ~bo & ~CSC0)] Combinational
[ bo: (~x & ai & ~ao & CSC0)] Combinational
[ bo: (~bi & bo)] Combinational
[ CSC0: (ao)] Combinational
[ CSC0: (~bo & CSC0)] Combinational
```

6.5.6. Find the circuit from your reshuffled and state variable solutions using a generalized C-element implementation technique. Comment on which is best and why.

```
[+ao: (x & ai)]
[+ao: (bi)]
[-ao: (~ai & ~bi)]
[ bo: (~x & ai & ~ao)] Combinational
```

```
[+ao: (x & ai)]
[+ao: (~bi & ~bo & ~CSC0)]
[-ao: (~ai & CSC0)]
[+bo: (~x & ai & ~ao & CSC0)]
[-bo: (bi & ~CSC0)]
[+CSC0: (ao)]
[-CSC0: (bo)]
```

The reshuffled one is likely simpler and perhaps even faster since it sets Ao high sooner.

6.6 Standard C-Implementation

Find a standard C-implementation for the circuit specified in Figure 6.11.

```
[+ao: (x & ai)]
[+ao: (bi)]
[-ao: (~ai & ~bi)]
[ bo: (~x & ai & ~ao)] Combinational
```

6.7 Hazard-Free Decomposition

For the state graph shown in Figure 6.12 and output signals *lo*, *ro*, and x:

6.7.1. Find a gC implementation using the single-cube algorithm.

```
[+ro: (ri & x & ~lo)]
[-ro: (~ri & ~x)]
[+x: (lo)]
[-x: (ro)]
[+lo: (~li & ~ro & ~x)]
[-lo: (li & x)]
```

6.7.2. Use the insertion point method to decompose any gates which have a fanin greater than two.

```
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.handshake.all;
entity p6 is
end entity;
architecture hse of p6 is
  signal ai:std_logic; --@ in
  signal bi:std_logic; --@ in
  signal x:std_logic; --@ in
  signal ao:std_logic;
  signal bo:std_logic;
begin
main:process
begin
  guard(ai,'1');
  if (x = '1') then
    assign(ao,'1',1,2);
    guard(ai,'0');
    assign(ao,'0',1,2);
  else
    assign(bo,'1',1,2);
    guard(bi,'1');
    assign(ao,'1',1,2);
    assign(bo,'0',1,2);
    guard_and(bi,'0',ai,'0');
    assign(ao,'0',1,2);
  end if;
end process;
ai:process
  variable z:integer;
begin
  z:=selection(2);
  if (z=1) then
    assign(x,'1',1,2);
    assign(ai,'1',1,2);
    guard(ao,'1');
    assign(x,'0',1,2);
    assign(ai,'0',1,2);
    guard(ao,'0');
  else
    assign(ai,'1',1,2);
    guard(ao,'1');
    assign(ai,'0',1,2);
    guard(ao,'0');
  end if;
end process;  bi:process
begin
  guard(bo,'1');
  assign(bi,'1',1,2);
  guard(bo,'0');
  assign(bi,'0',1,2);
end process;
end hse;
```

Fig. 6.11 VHDL for Problem 6.6.

```
[+z: (~li & ~x)]
[-z: (li & x)]
[ ro: (~lo & y)] Combinational
[+x: (lo)]
[-x: (ro)]
[ lo: (z & ~ro)] Combinational
[+y: (ri & x)]
[-y: (~ri & ~x)]
```

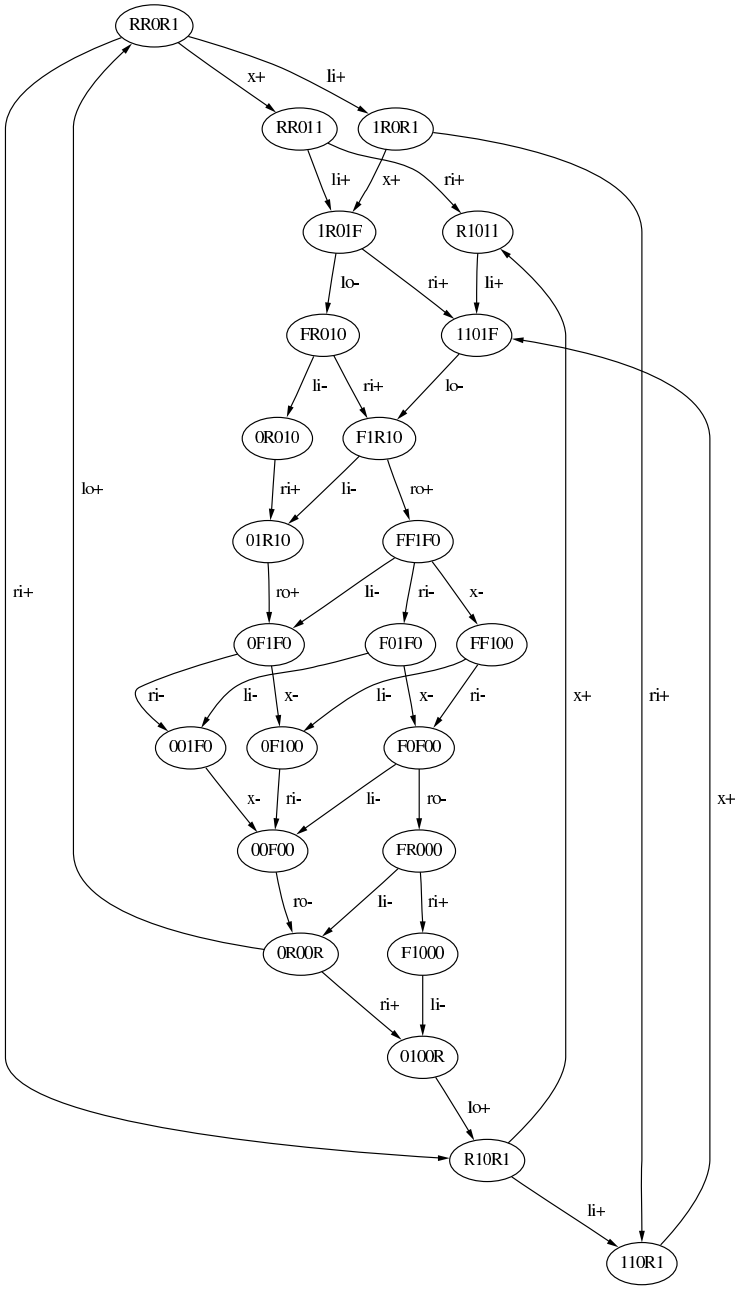


Fig. 6.12 SG for Problem 6.7.

7

Timed Circuits

Problems

7.1 Modeling Timing

Give two timed firing sequences for the TEL structure shown in Figure 7.1(a) that end with the *Wine Is Purchased* event.

First sequence:
([{$r5, r6$},$t5 = t6 = 0$],0), ([{$r5, r6$},$t5 = t6 = 0.5$],0.5), ([{$r1, r6$},$t1 = 0$,$t6 = 0.5$],0.5), ([{$r1, r6$},$t1 = 0.2$,$t6 = 0.7$],0.7), ([{$r1, r2$},$t1 = 0.2$,$t2 = 0$],0.7), ([{$r1, r2$},$t1 = 1.2$,$t2 = 1.0$],1.7), ([{$r2, r3$},$t2 = 1.0$,$t3 = 0$],1.7), ([{$r2, r3$},$t2 = 1.5$,$t3 = 0.5$],2.2), ([{$r3, r4$},$t3 = 0.5$,$t4 = 0$],2.2), ([{$r3, r4$}, $t3 = 0.7$, $t4 = 0.2$],2.4), ([{$r5, r6$},$t5 = t6 = 0$],2.4)

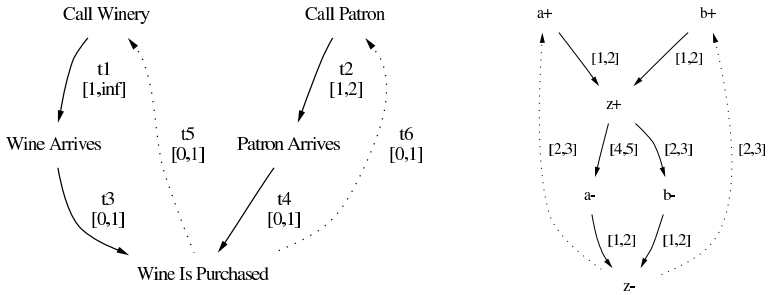
Second sequence:
([{$r5, r6$},$t5 = t6 = 0$],0), ([{$r5, r6$},$t5 = t6 = 0.5$],0.5), ([{$r2, r5$},$t2 = 0, t5 = 0.5$],0.5), ([{$r2, r5$},$t2 = 0.2, t5 = 0.7$],0.7), ([{$r1, r2$},$t1 = 0, t2 = 0.2$],0.7), ([{$r1, r2$},$t1 = 1.0, t2 = 1.2$],1.7), ([{$r2, r3$},$t2 = 1.2, t3 = 0$],1.7), ([{$r2, r3$},$t2 = 1.7, t3 = 0.5$],2.2), ([{$r3, r4$},$t3 = 0.5, t4 = 0$],2.2), ([{$r3, r4$}, $t3 = 0.7$, $t4 = 0.2$],2.4), ([{$r5, r6$},$t5 = t6 = 0$],2.4)

Many other solutions exist.

7.2 Modeling Timing

Give two timed firing sequences for the TEL structure shown in Figure 7.1(b) that end with the $z+$ event.

Assume that $(a+, z+)$ is t1, $(b+, z+)$ is t2, $(z+, a-)$ is t3, $(z+, b-)$ is t4, $(a-, z-)$ is t5, $(b-, z-)$ is t6, $(z-, a+)$ is t7, and $(z-, b+)$ is t8.



(a) (b)

Fig. 7.1 TEL structures for homework problems.

First sequence:
([{$t7, t8$}, $t7 = t8 = 0$], t=0), ([{$t7, t8$}, $t7 = t8 = 2.5$], t=2.5), ([{$t1, t8$}, $t1 = 0$, $t8 = 2.5$], t=2.5), ([{$t1, t8$}, $t1 = 0.5$, $t8 = 3.0$], t=3.0), ([{$t1, t2$}, $t1 = 0.5$, $t2 = 0$], t=3.0), ([{$t1, t2$}, $t1 = 2.0$, $t2 = 1.5$], t=4.5), ([{$t3, t4$}, $t3 = t4 = 0$], t=4.5)

Second sequence:
([{$t7, t8$}, $t7 = t8 = 0$], t=0), ([{$t7, t8$}, $t7 = t8 = 2.5$], t=2.5), ([{$t2, t7$}, $t2 = 0$, $t7 = 2.5$], t=2.5), ([{$t2, t7$}, $t2 = 0.5$, $t7 = 3.0$], t=3.0), ([{$t1, t2$}, $t1 = 0$, $t2 = 0.5$], t=3.0), ([{$t1, t2$}, $t1 = 1.5$, $t2 = 2.0$], t=4.5), ([{$t3, t4$}, $t3 = t4 = 0$], t=4.5)

Many other solutions exist.

7.3 Regions

Using regions to represent timing, find the timed state space for one timed firing sequence for the TEL structure shown in Figure 7.1(a) that ends with the *Wine Is Purchased* event.

One possible solution shown in Figure 7.2(a).

7.4 Regions

Using regions to represent timing, find the timed state space for one timed firing sequence for the TEL structure shown in Figure 7.1(b) that ends with the $z+$ event.

Assume that $(a+, z+)$ is t1, $(b+, z+)$ is t2, $(z+, a-)$ is t3, $(z+, b-)$ is t4, $(a-, z-)$ is t5, $(b-, z-)$ is t6, $(z-, a+)$ is t7, and $(z-, b+)$ is t8.

One possible solution shown in Figure 7.2(b).

7.5 Discrete Time

Using discrete-time states, find the timed state space for one timed firing sequence for the TEL structure shown in Figure 7.1(a) that ends with the *Wine Is Purchased* event.

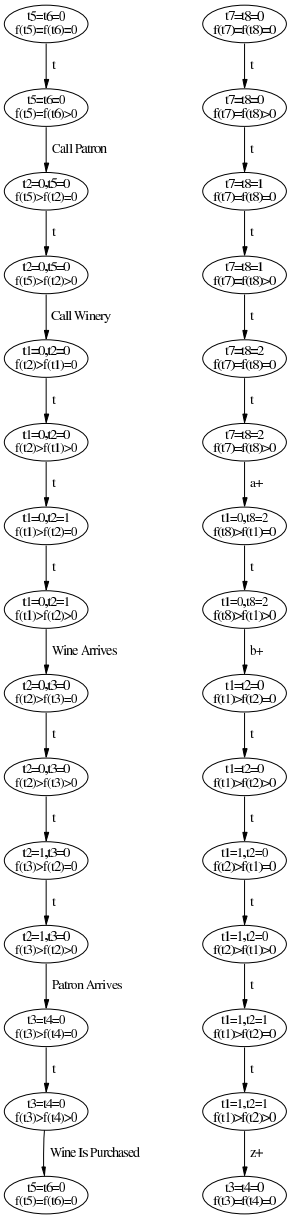


Fig. 7.2 (a)Solution for 7.3. (b) Solution for 7.4.

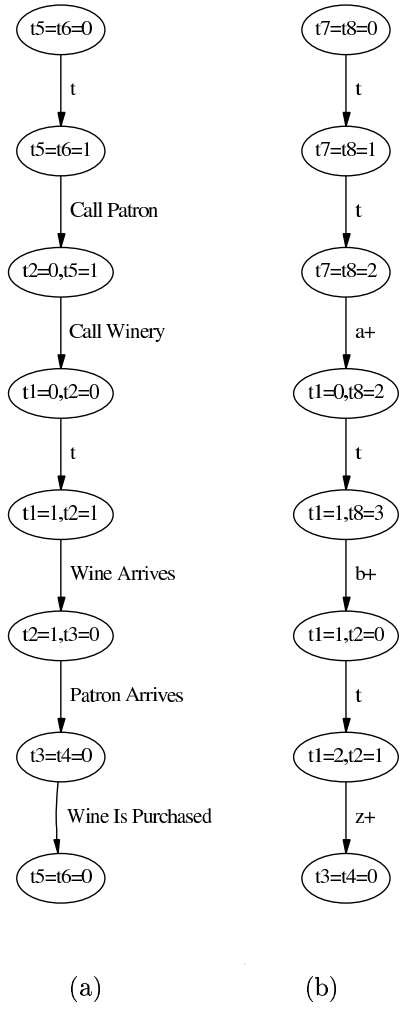


Fig. 7.3 (a)Solution for 7.5. (b) Solution for 7.6.

One possible solution shown in Figure 7.3(a).

7.6 Discrete Time

Using discrete-time states, find the timed state space for one timed firing sequence for the TEL structure shown in Figure 7.1(b) that ends with the $z+$ event.

Assume that $(a+, z+)$ is t1, $(b+, z+)$ is t2, $(z+, a-)$ is t3, $(z+, b-)$ is t4, $(a-, z-)$ is t5, $(b-, z-)$ is t6, $(z-, a+)$ is t7, and $(z-, b+)$ is t8.

One possible solution shown in Figure 7.3(b).

7.7 Zones

Using zones to represent timing, find the entire timed state space for the TEL structure shown in Figure 7.1(a).

```
Marking: {(Wine_Is_Purchased,Call_Winery),
          (Wine_Is_Purchased,Call_Patron)}
r1: (Wine_Is_Purchased,Call_Patron)
r2: (Wine_Is_Purchased,Call_Winery)
          r0   r1   r2
       ------------------
   r0|     0     1     1
   r1|     0     0     0
   r2|     0     0     0
Fired Clocks: {}

Marking: {(Call_Patron,Patron_Arrives),
          (Wine_Is_Purchased,Call_Winery)}
r1: (Wine_Is_Purchased,Call_Winery)
r2: (Call_Patron,Patron_Arrives)
          r0   r1   r2
       ------------------
   r0|     0     1     1
   r1|     0     0     0
   r2|     0     1     0
Fired Clocks: {}

Marking: {(Call_Patron,Patron_Arrives),
          (Call_Winery,Wine_Arrives)}
r1: (Call_Patron,Patron_Arrives)
r2: (Call_Winery,Wine_Arrives)
          r0   r1   r2
       ------------------
   r0|     0     2     1
   r1|     0     0     0
   r2|     0     1     0
Fired Clocks: {}

Marking: {(Call_Winery,Wine_Arrives),
          (Patron_Arrives,Wine_Is_Purchased)}
r1: (Call_Winery,Wine_Arrives)
r2: (Patron_Arrives,Wine_Is_Purchased)
          r0   r1   r2
       ------------------
   r0|     0     1     1
   r1|     0     0     0
```

```
   r2|     0     1     0
Fired Clocks: {}

Marking: {(Wine_Arrives,Wine_Is_Purchased),
          (Patron_Arrives,Wine_Is_Purchased)}
r1: (Wine_Arrives,Wine_Is_Purchased)
          r0   r1
       ------------
   r0|     0     1
   r1|     0     0
Fired Clocks: {(Patron_Arrives,Wine_Is_Purchased)}

Marking: {(Call_Patron,Patron_Arrives),
          (Wine_Arrives,Wine_Is_Purchased)}
r1: (Call_Patron,Patron_Arrives)
r2: (Wine_Arrives,Wine_Is_Purchased)
          r0   r1   r2
       ------------------
   r0|     0     2     1
   r1|    -1     0    -1
   r2|     0     2     0
Fired Clocks: {}

Marking: {(Wine_Arrives,Wine_Is_Purchased),
          (Patron_Arrives,Wine_Is_Purchased)}
r1: (Patron_Arrives,Wine_Is_Purchased)
          r0   r1
       ------------
   r0|     0     1
   r1|     0     0
Fired Clocks: {(Wine_Arrives,Wine_Is_Purchased)}

Marking: {(Patron_Arrives,Wine_Is_Purchased),
          (Wine_Is_Purchased,Call_Winery)}
r1: (Wine_Is_Purchased,Call_Winery)
r2: (Patron_Arrives,Wine_Is_Purchased)
          r0   r1   r2
       ------------------
   r0|     0     1     0
   r1|    -1     0    -1
   r2|     0     1     0
Fired Clocks: {}

Marking: {(Call_Winery,Wine_Arrives),
          (Patron_Arrives,Wine_Is_Purchased)}
```

```
r1: (Call_Winery,Wine_Arrives)
          r0   r1
       ------------
   r0|     0     1
   r1|     0     0
Fired Clocks: {(Patron_Arrives,Wine_Is_Purchased)}

Marking: {(Call_Winery,Wine_Arrives),
          (Wine_Is_Purchased,Call_Patron)}
r1: (Wine_Is_Purchased,Call_Patron)
r2: (Call_Winery,Wine_Arrives)
          r0   r1   r2
       ------------------
   r0|     0     1     1
   r1|     0     0     0
   r2|     0     1     0
Fired Clocks: {}

Marking: {(Call_Patron,Patron_Arrives),
          (Call_Winery,Wine_Arrives)}
r1: (Call_Winery,Wine_Arrives)
r2: (Call_Patron,Patron_Arrives)
          r0   r1   r2
       ------------------
   r0|     0     2     2
   r1|     0     0     0
   r2|     0     1     0
Fired Clocks: {}

Marking: {(Call_Patron,Patron_Arrives),
          (Wine_Arrives,Wine_Is_Purchased)}
r1: (Call_Patron,Patron_Arrives)
r2: (Wine_Arrives,Wine_Is_Purchased)
          r0   r1   r2
       ------------------
   r0|     0     2     1
   r1|     0     0     0
   r2|     0     2     0
Fired Clocks: {}

Marking: {(Call_Winery,Wine_Arrives),
          (Patron_Arrives,Wine_Is_Purchased)}
r1: (Call_Winery,Wine_Arrives)
r2: (Patron_Arrives,Wine_Is_Purchased)
          r0   r1   r2
```

```
       ------------------
   r0|     0     2     1
   r1|    -1     0    -1
   r2|     0     2     0
Fired Clocks: {}

Marking: {(Wine_Arrives,Wine_Is_Purchased),
          (Wine_Is_Purchased,Call_Patron)}
r1: (Wine_Is_Purchased,Call_Patron)
r2: (Wine_Arrives,Wine_Is_Purchased)
          r0   r1   r2
       ------------------
   r0|     0     1     0
   r1|    -1     0    -1
   r2|     0     1     0

Marking: {(Call_Patron,Patron_Arrives),
          (Wine_Arrives,Wine_Is_Purchased)}
r1: (Call_Patron,Patron_Arrives)
          r0   r1
       ------------
   r0|     0     2
   r1|     0     0
Fired Clocks: {(Wine_Arrives,Wine_Is_Purchased)}
```

7.8 Zones

Using zones to represent timing, find the entire timed state space for the TEL structure shown in Figure 7.1(b).

```
0:RR0
Marking: {(z-/1,b+/1),(z-/1,a+/1)}
r1: (z-/1,a+/1)
r2: (z-/1,b+/1)
          r0   r1   r2
       ------------------
   r0|     0     3     3
   r1|     0     0     0
   r2|     0     0     0
Fired Clocks: {}

1:1R0
Marking: {(a+/1,z+/1),(z-/1,b+/1)}
r1: (z-/1,b+/1)
r2: (a+/1,z+/1)
          r0   r1   r2
```

```
       ------------------
   r0|     0     3     1
   r1|    -2     0    -2
   r2|     0     3     0
Fired Clocks: {}

2:11R
Marking: {(a+/1,z+/1),(b+/1,z+/1)}
r1: (b+/1,z+/1)
          r0   r1
       ------------
   r0|     0     2
   r1|     0     0
Fired Clocks: {(a+/1,z+/1)}

3:FF1
Marking: {(z+/1,b-/1),(z+/1,a-/1)}
r1: (z+/1,b-/1)
r2: (z+/1,a-/1)
          r0   r1   r2
       ------------------
   r0|     0     3     3
   r1|     0     0     0
   r2|     0     0     0
Fired Clocks: {}

4:F01
Marking: {(b-/1,z-/1),(z+/1,a-/1)}
r1: (z+/1,a-/1)
r2: (b-/1,z-/1)
          r0   r1   r2
       ------------------
   r0|     0     5     2
   r1|    -2     0    -2
   r2|     0     3     0

Fired Clocks: {}

5:00F
Marking: {(b-/1,z-/1),(a-/1,z-/1)}
r1: (a-/1,z-/1)
          r0   r1
       ------------
   r0|     0     2
   r1|     0     0
```

```
Fired Clocks: {(b-/1,z-/1)}

6:R10
Marking: {(b+/1,z+/1),(z-/1,a+/1)}
r1: (z-/1,a+/1)
r2: (b+/1,z+/1)
          r0   r1   r2
       ------------------
   r0|     0     3     1
   r1|    -2     0    -2
   r2|     0     3     0
Fired Clocks: {}

7:11R
Marking: {(a+/1,z+/1),(b+/1,z+/1)}
r1: (a+/1,z+/1)
          r0   r1
       ------------
   r0|     0     2
   r1|     0     0
Fired Clocks: {(b+/1,z+/1)}
```

7.9 POSET Timing

Using POSETs and zones, find the entire timed state space for the TEL structure shown in Figure 7.1(a).

```
Marking: {(Wine_Is_Purchased,Call_Winery),
          (Wine_Is_Purchased,Call_Patron)}
r1: (Wine_Is_Purchased,Call_Patron)
r2: (Wine_Is_Purchased,Call_Winery)
          r0   r1   r2
       ------------------
   r0|     0     1     1
   r1|     0     0     0
   r2|     0     0     0
Fired Clocks: {}

Marking: {(Call_Patron,Patron_Arrives),
          (Wine_Is_Purchased,Call_Winery)}
r1: (Wine_Is_Purchased,Call_Winery)
r2: (Call_Patron,Patron_Arrives)
          r0   r1   r2
       ------------------
   r0|     0     1     1
   r1|     0     0     0
```

```
   r2|     0     1     0
Fired Clocks: {}

Marking: {(Call_Patron,Patron_Arrives),
          (Call_Winery,Wine_Arrives)}
r1: (Call_Patron,Patron_Arrives)
r2: (Call_Winery,Wine_Arrives)
          r0   r1   r2
       ------------------
   r0|     0     2     1
   r1|     0     0     1
   r2|     0     1     0
Fired Clocks: {}

Marking: {(Call_Winery,Wine_Arrives),
           (Patron_Arrives,Wine_Is_Purchased)}
r1: (Call_Winery,Wine_Arrives)
r2: (Patron_Arrives,Wine_Is_Purchased)
          r0   r1   r2
       ------------------
   r0|     0     1     1
   r1|     0     0     0
   r2|     0     3     0
Fired Clocks: {}

Marking: {(Wine_Arrives,Wine_Is_Purchased),
          (Patron_Arrives,Wine_Is_Purchased)}
r1: (Wine_Arrives,Wine_Is_Purchased)
          r0   r1
       ------------
   r0|     0     1
   r1|     0     0
Fired Clocks: {(Patron_Arrives,Wine_Is_Purchased)}

Marking: {(Call_Patron,Patron_Arrives),
          (Wine_Arrives,Wine_Is_Purchased)}
r1: (Call_Patron,Patron_Arrives)
r2: (Wine_Arrives,Wine_Is_Purchased)
          r0   r1   r2
       ------------------
   r0|     0     2     1
   r1|     0     0     0
   r2|     0     2     0
Fired Clocks: {}
```

```
Marking: {(Wine_Arrives,Wine_Is_Purchased),
          (Patron_Arrives,Wine_Is_Purchased)}
r1: (Patron_Arrives,Wine_Is_Purchased)
          r0   r1
       ------------
   r0|     0     1
   r1|     0     0
Fired Clocks: {(Wine_Arrives,Wine_Is_Purchased)}

Marking: {(Patron_Arrives,Wine_Is_Purchased),
          (Wine_Is_Purchased,Call_Winery)}
r1: (Wine_Is_Purchased,Call_Winery)
r2: (Patron_Arrives,Wine_Is_Purchased)
          r0   r1   r2
       ------------------
   r0|     0     1     0
   r1|    -1     0    -1
   r2|     0     1     0
Fired Clocks: {}

Marking: {(Call_Winery,Wine_Arrives),
          (Patron_Arrives,Wine_Is_Purchased)}
r1: (Call_Winery,Wine_Arrives)
          r0   r1
       ------------
   r0|     0     1
   r1|     0     0
Fired Clocks: {(Patron_Arrives,Wine_Is_Purchased)}

Marking: {(Call_Winery,Wine_Arrives),
          (Wine_Is_Purchased,Call_Patron)}
r1: (Wine_Is_Purchased,Call_Patron)
r2: (Call_Winery,Wine_Arrives)
          r0   r1   r2
       ------------------
   r0|     0     1     1
   r1|     0     0     0
   r2|     0     1     0
Fired Clocks: {}

Marking: {(Wine_Arrives,Wine_Is_Purchased),
          (Wine_Is_Purchased,Call_Patron)}
r1: (Wine_Is_Purchased,Call_Patron)
r2: (Wine_Arrives,Wine_Is_Purchased)
          r0   r1   r2
```

```
       ------------------
   r0|     0     1     0
   r1|    -1     0    -1
   r2|     0     1     0
Fired Clocks: {}

Marking: {(Call_Patron,Patron_Arrives),
          (Wine_Arrives,Wine_Is_Purchased)}
r1: (Call_Patron,Patron_Arrives)
          r0   r1
       ------------
   r0|     0     2
   r1|     0     0
Fired Clocks: {(Wine_Arrives,Wine_Is_Purchased)}
```

7.10 POSET Timing

Using POSETs and zones, find the entire timed state space for the TEL structure shown in Figure 7.1(b).

```
0:RR0
Marking: {(z-/1,b+/1),(z-/1,a+/1)}
r1: (z-/1,a+/1)
r2: (z-/1,b+/1)
          r0   r1   r2
       ------------------
   r0|     0     3     3
   r1|     0     0     0
   r2|     0     0     0
Fired Clocks: {}

1:1R0
Marking: {(a+/1,z+/1),(z-/1,b+/1)}
r1: (z-/1,b+/1)
r2: (a+/1,z+/1)
          r0   r1   r2
       ------------------
   r0|     0     3     1
   r1|    -2     0    -2
   r2|     0     3     0
Fired Clocks: {}

2:11R
Marking: {(a+/1,z+/1),(b+/1,z+/1)}
r1: (b+/1,z+/1)
          r0   r1
```

```
       ------------
   r0|     0     2
   r1|     0     0
Fired Clocks: {(a+/1,z+/1)}

3:FF1
Marking: {(z+/1,b-/1),(z+/1,a-/1)}
r1: (z+/1,b-/1)
r2: (z+/1,a-/1)
          r0   r1   r2
       ------------------
   r0|     0     3     3
   r1|     0     0     0
   r2|     0     0     0
Fired Clocks: {}

4:F01
Marking: {(b-/1,z-/1),(z+/1,a-/1)}
r1: (z+/1,a-/1)
r2: (b-/1,z-/1)
          r0   r1   r2
       ------------------
   r0|     0     5     2
   r1|    -2     0    -2
   r2|     0     3     0

Fired Clocks: {}

5:00F
Marking: {(b-/1,z-/1),(a-/1,z-/1)}
r1: (a-/1,z-/1)
          r0   r1
       ------------
   r0|     0     2
   r1|     0     0
Fired Clocks: {(b-/1,z-/1)}

6:R10
Marking: {(b+/1,z+/1),(z-/1,a+/1)}
r1: (z-/1,a+/1)
r2: (b+/1,z+/1)
          r0   r1   r2
       ------------------
   r0|     0     3     1
   r1|    -2     0    -2
```

```
   r2|     0     3     0
Fired Clocks: {}

7:11R
Marking: {(a+/1,z+/1),(b+/1,z+/1)}
r1: (a+/1,z+/1)
          r0   r1
       ------------
   r0|     0     2
   r1|     0     0
Fired Clocks: {(b+/1,z+/1)}
```

7.11 Timed Circuits

Consider the TEL structure shown in Figure 7.1(b).

7.11.1. Find a SG ignoring timing and use it to find a Muller circuit for output signal z.

SG shown in Figure 7.4(a).

```
[+z: (a & b)]
[-z: (~a & ~b)]
```

7.11.2. Find a SG considering timing and use it to find a timed circuit for output signal z.

SG shown in Figure 7.4(b).

```
[+z: (a & b)]
[-z: (~a)]
```

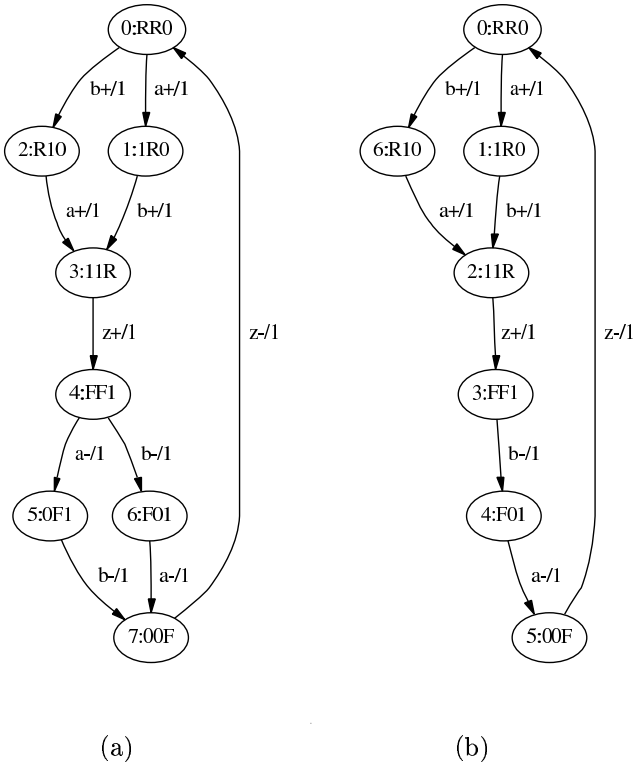


Fig. 7.4 (a)Untimed SG. (b) Timed SG for Figure 7.1.

8

Verification

Problems

8.1 Linear-Time Temporal Logic
Consider the situation where there are two shops to which the winery can deliver wine. It communicates with the first using the wires *req_wine1* and *ack_wine1* and the second with *req_wine2* and *ack_wine2*. Write an LTL formula that says that the winery is *fair*. In other words, the winery will deliver wine to both shops.

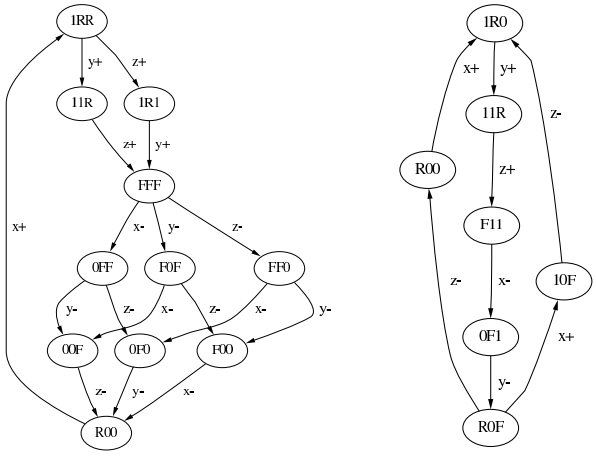
$$\Box(\Diamond(req\_wine1) \wedge \Diamond(req\_wine2))$$

8.2 Linear-Time Temporal Logic
Again consider the situation where there are two shops to which the winery can deliver wine. Write an LTL formula that says that the winery at some point decides to sell only to one shop in the future.

$$\Diamond(\Box(\neg req\_wine1) \vee \Box(\neg req\_wine2))$$

8.3 Linear-Time Temporal Logic
If either the winery can stop producing wine or the patron loses interest in buying wine, which of LTL formulas 8.1 to 8.8 would be violated? How could they be fixed?

If the winery stops producing wine, then at some point *req_wine* would stop going high. This would cause 8.1 to be violated since a state where *req_wine* is



(a) (b)

Fig. 8.1 SGs for Problems 8.4 and 8.5 (state vector $\langle x, y, z \rangle$).

high would never be reached. This can be fixed using the weak until operator in place of the strong until operator. Similarly, if the patron loses interest in buying wine, then at some point *ack_patron* would stop going high. This would cause 8.3 to be violated. This also can be fixed using the weak until operator in place of the strong until operator.

8.4 Protocol Verification

Check if the SG in Figure 8.1(a) satisfies the following LTL formula:

$$\Box(y \Rightarrow (x \mathbf{U} z))$$

If not, indicate which states violate the formula.

This formula is not true in states FF0 and 0F0.

8.5 Protocol Verification

Check if the SG in Figure 8.1(b) satisfies the following LTL formula:

$$\Box(y \Rightarrow (x \mathbf{U} z))$$

If not, indicate which states violate the formula.

The SG satisfies this formula.

8.6 Timed Linear-Time Temporal Logic

Again consider the situation where there are two shops to which the winery can deliver wine. Write a timed LTL formula that says that the winery is fair in bounded time. In other words, the winery will not stop delivering wine to one shop for more than 60 minutes.

$$\Box(\Diamond_{<60}(req\_wine1) \wedge \Diamond_{<60}(req\_wine2))$$

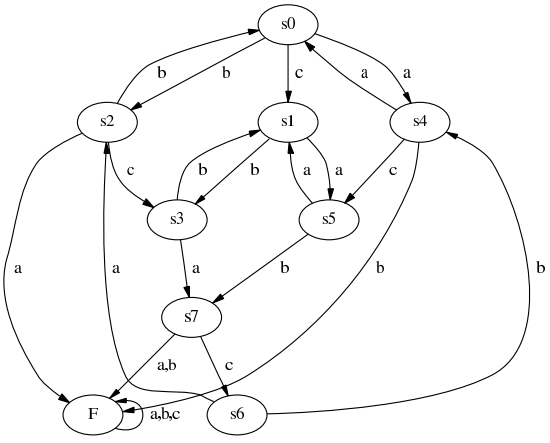


Fig. 8.2 Solution for Problem 8.7.

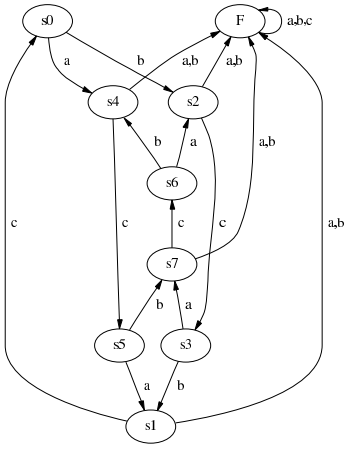


Fig. 8.3 Solution for Problem 8.8.

8.7 Trace Structures

Give a receptive trace structure for a NAND gate.

Solution shown in Figure 8.2.

8.8 Trace Structures

Give a receptive trace structure for an XOR gate.

Solution shown in Figure 8.3.

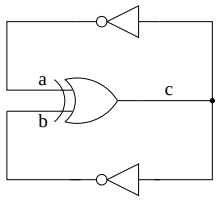


Fig. 8.4 Circuit for Problem 8.9.

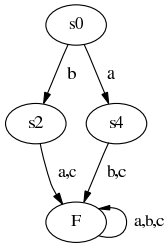


Fig. 8.5 Solution for Problem 8.9.

8.9 Trace Structure Composition

Use composition of trace structures to determine whether or not the circuit shown in Figure 8.4 is failure-free.

The trace structure after composition is shown in Figure 8.5. This circuit is not failure-free.

8.10 Trace Structure Composition

Use composition of trace structures to determine whether or not the circuit shown in Figure 8.6 is failure-free.

NEED A SOLUTION HERE

8.11 Canonical Trace Structures

Transform the trace structure shown in Figure 8.7 to a canonical prefix-closed trace structure.

Should assume that a and b are inputs and c and d are outputs. Solution is shown in Figure 8.8.

8.12 Mirrors and Verification

Use composition to create a trace structure for the circuit shown in Figure 8.9. Find the mirror of the specification for a C-element shown in Figure 8.10. Compose the trace structure for the circuit and the mirror of the specification to determine if the circuit conforms to the specification (i.e., the circuit is a

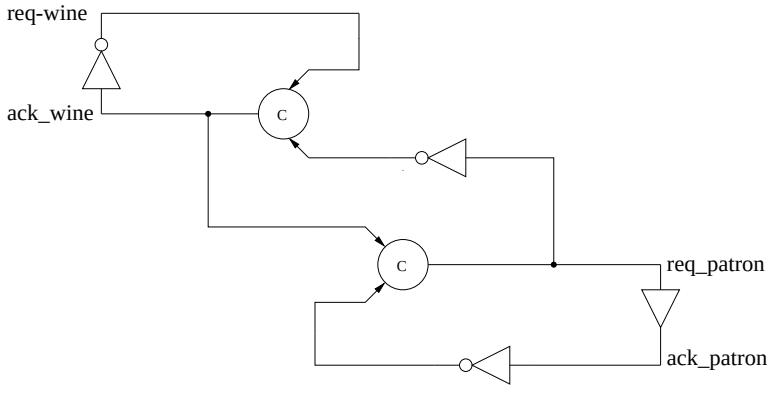


Fig. 8.6 Circuit for Problem 8.10.

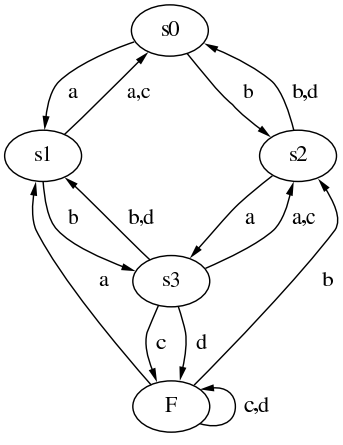


Fig. 8.7 Trace structure for Problem 8.11.

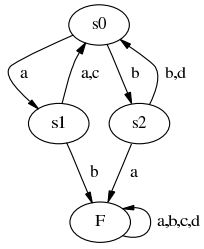


Fig. 8.8 Solution for Problem 8.11.

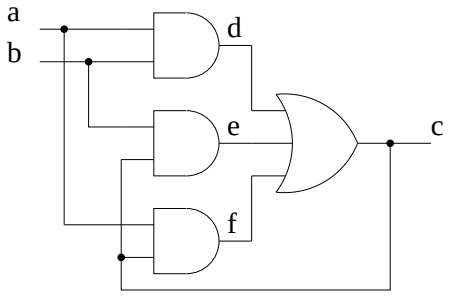


Fig. 8.9 Circuit for Problem 8.12.

correct implementation of a C-element). If the circuit does not conform, give a sequence of transitions that causes the circuit to fail.

8.13 Mirrors and Verification

Use composition to create a trace structure for the circuit shown in Figure 8.11. Find the mirror of the specification for a C-element shown in Figure 8.10. Compose the trace structure for the circuit and the mirror of the specification to determine if the circuit conforms to the specification (i.e., the circuit is a correct implementation of a C-element). If the circuit does not conform, give a sequence of transitions that causes the circuit to fail.

8.14 Strong Conformance

A sequencer receives an input a and generates an output b followed by another output c. The trace structure for a sequencer is shown in Figure 8.12(a). A fork receives an input a and generates outputs b and c in parallel. The trace structure for a fork is shown in Figure 8.12(b). Show that a sequencer conforms to a fork, but a fork does not conform to a sequencer. Does a sequencer strongly conform to a fork?

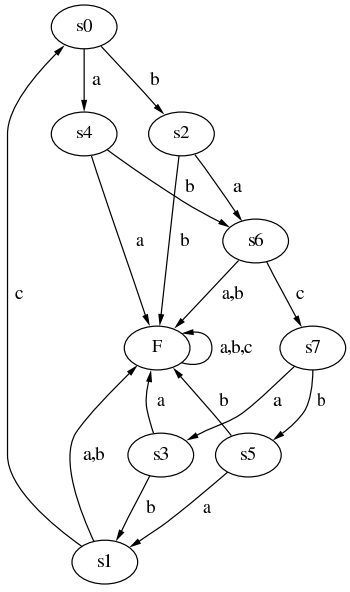


Fig. 8.10 Specification for Problems 8.12 and 8.13.

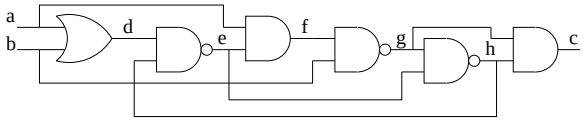


Fig. 8.11 Circuit for Problem 8.13.

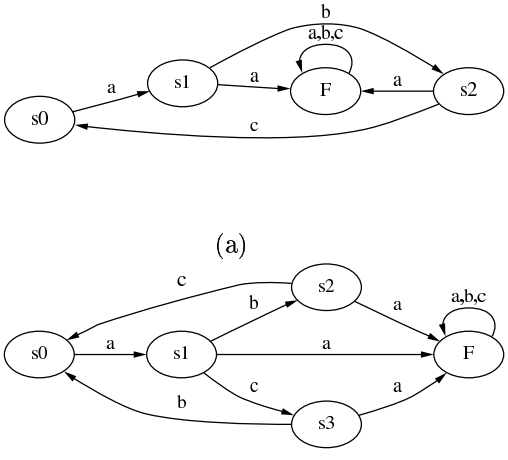


Fig. 8.12 (a) Sequencer and (b) fork.

9

Applications

Problems

9.1 Probability of Synchronization Failure
Assume that $\tau = 20$ ps, $T_0 = 8$ ns, the clock frequency is 4 GHz, asynchronous inputs arrive at an average rate of 2 GHz, and $t_a = 200$ ps.

9.1.1. What is the MTBF?

344ns

9.1.2. How many latches does it take to reduce MTBF to 10,000 years?

Assuming that t_{pd} is 120 ps, it takes 7 latches.

9.1.3. If no extra latchs are used, at what rate can asynchronous inputs arrive such that the MTBF is over 1 year?

$2.18e-5Hz$

9.1.4. For what value of τ does the MTBF exceed 1 year?

4.75ps

9.2 Probability of Synchronization Failure
Assume that $\tau = 10$ ps, $T_0 = 4$ ns, the clock frequency is 5 GHz, asynchronous inputs arrive at an average rate of 100 MHz, and $t_a = 20$ ps.

9.2.1. What is the MTBF?

3.7ns

9.2.2. How many latches does it take to reduce MTBF to 100,000 years?

Assuming that t_{pd} is 120 ps, it takes 7 latches.

9.2.3. If no extra latchs are used, at what clock frequency does the MTBF exceed one year? (Assume that t_a is 10 percent of the cycle time.)

9.2.4. What clock frequency would produce an MTBF of more than 1 year if t_a is 50 percent of the cycle time?