Asynchronous Circuit Design. Chris J. Myers
Copyright © 2001 by John Wiley & Sons, Inc.
ISBNs: 0-471-41543-X (Hardback); 0-471-22414-6 (Electronic)

Appendix A
VHDL Packages

I have always imagined that Paradise will be a kind of library.
—Jorge Luis Borges

A.1 NONDETERMINISM.VHD

This package defines functions to model random selection and random delays.

```
library ieee;
use ieee.math_real.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
package nondeterminism is
  shared variable s1:integer:=844396720;
  shared variable s2:integer:=821616997;
  -- Returns a number between 1 and num.
  impure function selection(constant num:in integer) return integer;
  -- Returns a std_logic_vector of size bits between 1 and num.
  impure function selection(constant num:in integer;
                            constant size:in integer) return std_logic_vector;
  -- Returns random delay between lower and upper.
  impure function delay(constant l:in integer;
                        constant u:in integer) return time;
```

```
end nondeterminism;
package body nondeterminism is
  impure function selection(constant num:in integer) return integer is
    variable result:integer;
    variable tmp_real:real;
  begin
    uniform(s1,s2,tmp_real);
    result := 1 + integer(trunc(tmp_real * real(num)));
    return(result);
  end selection;
  impure function selection(constant num:in integer;
                            constant size:in integer)
    return std_logic_vector is
    variable result:std_logic_vector(size-1 downto 0);
    variable tmp_real:real;
  begin
    uniform(s1,s2,tmp_real);
    result := conv_std_logic_vector(integer(trunc(tmp_real * real(num)))
                                    +1,size);
    return(result);
  end selection;
  impure function delay(constant l:in integer;
                        constant u:in integer) return time is
    variable result:time;
    variable tmp:real;
  begin
    uniform(s1,s2,tmp);
    result:=(((tmp * real(u - l)) + real(l)) * 1 ns);
    return result;
  end delay;
end nondeterminism;
```

A.2 CHANNEL.VHD

This package defines a channel data type and send and receive functions.

```
library IEEE;
use IEEE.std_logic_1164.all;
package channel is
  constant MAX_BIT_WIDTH:natural:=32;
  subtype datatype is std_logic_vector((MAX_BIT_WIDTH-1) downto 0 );
  constant dataZ:datatype:=datatype'(others => 'Z');
  constant data0:datatype:=datatype'(others => '0');
  constant dataACK:dataType:=dataType'(others => '1');
  type channel is record
    dataright,dataleft:datatype;
    pending_send,pending_recv,sync:std_logic;
  end record;
  type bools is array (natural range <>) of boolean;
  -- Used to send data on a channel
  procedure send(signal c1:inout channel);
  procedure send(signal c1:inout channel;signal d1:inout std_logic);
  procedure send(signal c1:inout channel;signal d1:inout std_logic;
                 signal c2:inout channel;signal d2:inout std_logic);
  procedure send(signal c1:inout channel;signal d1:inout std_logic;
                 signal c2:inout channel;signal d2:inout std_logic;
                 signal c3:inout channel;signal d3:inout std_logic);
  procedure send(signal c1:inout channel;signal d1:inout std_logic_vector);
  procedure send(signal c1:inout channel;signal d1:inout std_logic_vector;
                 signal c2:inout channel;signal d2:inout std_logic_vector);
  procedure send(signal c1:inout channel;signal d1:inout std_logic_vector;
                 signal c2:inout channel;signal d2:inout std_logic_vector;
                 signal c3:inout channel;signal d3:inout std_logic_vector);
  -- Used to receive data on a channel
  procedure receive(signal c1:inout channel);
  procedure receive(signal c1:inout channel;signal d1:inout std_logic);
  procedure receive(signal c1:inout channel;signal d1:inout std_logic;
                    signal c2:inout channel;signal d2:inout std_logic);
  procedure receive(signal c1:inout channel;signal d1:inout std_logic;
                    signal c2:inout channel;signal d2:inout std_logic;
                    signal c3:inout channel;signal d3:inout std_logic);
  procedure receive(signal c1:inout channel;signal d1:inout std_logic_vector);
  procedure receive(signal c1:inout channel;signal d1:inout std_logic_vector;
                    signal c2:inout channel;signal d2:inout std_logic_vector);
  procedure receive(signal c1:inout channel;signal d1:inout std_logic_vector;
                    signal c2:inout channel;signal d2:inout std_logic_vector;
                    signal c3:inout channel;signal d3:inout std_logic_vector);
  -- Initialization function called in a port declaration
  -- as a default value to initialize a channel.
  function init_channel return channel;
  function active return channel;
  function passive return channel;
  -- Test for pending communication on a channel
  function probe(signal chan:in channel) return boolean;
end channel;
package body channel is
  procedure validate(signal data:in std_logic_vector) is
  begin
    assert( data'LENGTH <= MAX_BIT_WIDTH )
      report "Bit width is too wide" severity failure;
  end validate;
  function init_channel return channel is
  begin
    return(dataright=>dataZ,dataleft=>dataZ,
           pending_send=>'Z',pending_recv=>'Z',sync=>'Z');
  end init_channel;
  function active return channel is
  begin
    return(dataright=>dataZ,dataleft=>dataZ,
           pending_send=>'Z',pending_recv=>'Z',sync=>'Z');
  end active;
  function passive return channel is
  begin
    return(dataright=>dataZ,dataleft=>dataZ,
           pending_send=>'Z',pending_recv=>'Z',sync=>'Z');
  end passive;
  procedure send_handshake(variable done:inout boolean;
                           variable reset:inout boolean;
                           signal chan:inout channel) is
  begin
```

```
    if (done=false) then
      if (reset=false) then
        if (chan.pending_send='Z') then
          chan.pending_send<='1';
        end if;
        if (chan.sync='1') then
          chan.pending_send<='Z';
          reset:=true;
        end if;
      elsif (chan.sync='Z') then
        done:=true;
      end if;
    end if;
  end send_handshake;
  function zero_extend(signal data:in std_logic) return datatype is
    variable extdata:datatype;
  begin
    extdata := data0;
    extdata(0) := data;
    return extdata;
  end zero_extend;
  function zero_extend(signal data:in std_logic_vector)
    return datatype is
    variable extdata:datatype;
  begin
    validate(data);
    extdata := data0;
    extdata( data'length - 1 downto 0 ) := data;
    return extdata;
  end zero_extend;
  procedure send(signal c1:inout channel) is
    variable done:bools(1 to 1) := (others => false);
    variable reset:bools(1 to 1) := (others => false);
  begin
    loop
      send_handshake(done(1),reset(1),c1);
      exit when (done(1)=true);
      wait on c1.sync;
    end loop;
  end send;
  procedure send(signal c1:inout channel;signal d1:inout std_logic) is
    variable done:bools(1 to 1) := (others => false);
    variable reset:bools(1 to 1) := (others => false);
  begin
    c1.dataright <= zero_extend(d1);
    loop
      send_handshake(done(1),reset(1),c1);
      exit when (done(1)=true);
      wait on c1.sync,c1.pending_send,c1.pending_recv;
    end loop;
  end send;
  procedure send(signal c1:inout channel;signal d1:inout std_logic;
                 signal c2:inout channel;signal d2:inout std_logic) is
    variable done:bools(1 to 2) := (others => false);
    variable reset:bools(1 to 2) := (others => false);
  begin
```

```
    c1.dataright <= zero_extend(d1);
    c2.dataright <= zero_extend(d2);
    loop
      send_handshake(done(1),reset(1),c1);
      send_handshake(done(2),reset(2),c2);
      exit when ((done(1)=true) and (done(2)=true));
      wait on c1.sync,c2.sync;
    end loop;
  end send;
  procedure send(signal c1:inout channel;signal d1:inout std_logic;
                 signal c2:inout channel;signal d2:inout std_logic;
                 signal c3:inout channel;signal d3:inout std_logic) is
    variable done:bools(1 to 3) := (others => false);
    variable reset:bools(1 to 3) := (others => false);
  begin
    c1.dataright <= zero_extend(d1);
    c2.dataright <= zero_extend(d2);
    c3.dataright <= zero_extend(d3);
    loop
      send_handshake(done(1),reset(1),c1);
      send_handshake(done(2),reset(2),c2);
      send_handshake(done(3),reset(3),c3);
      exit when ((done(1)=true) and (done(2)=true) and (done(3)=true));
      wait on c1.sync,c2.sync,c3.sync;
    end loop;
  end send;
  procedure send(signal c1:inout channel;
                 signal d1:inout std_logic_vector) is
    variable done:bools(1 to 1) := (others => false);
    variable reset:bools(1 to 1) := (others => false);
  begin
    c1.dataright <= zero_extend(d1);
    loop
      send_handshake(done(1),reset(1),c1);
      exit when (done(1)=true);
      wait on c1.sync,c1.pending_send,c1.pending_recv;
    end loop;
  end send;
  procedure send(signal c1:inout channel;signal d1:inout std_logic_vector;
                 signal c2:inout channel;signal d2:inout std_logic_vector) is
    variable done:bools(1 to 2) := (others => false);
    variable reset:bools(1 to 2) := (others => false);
  begin
    c1.dataright <= zero_extend(d1);
    c2.dataright <= zero_extend(d2);
    loop
      send_handshake(done(1),reset(1),c1);
      send_handshake(done(2),reset(2),c2);
      exit when ((done(1)=true) and (done(2)=true));
      wait on c1.sync,c2.sync;
    end loop;
  end send;
  procedure send(signal c1:inout channel;signal d1:inout std_logic_vector;
                 signal c2:inout channel;signal d2:inout std_logic_vector;
                 signal c3:inout channel;signal d3:inout std_logic_vector) is
    variable done:bools(1 to 3) := (others => false);
  variable reset:bools(1 to 3) := (others => false);
begin
  c1.dataright <= zero_extend(d1);
  c2.dataright <= zero_extend(d2);
  c3.dataright <= zero_extend(d3);
  loop
    send_handshake(done(1),reset(1),c1);
    send_handshake(done(2),reset(2),c2);
    send_handshake(done(3),reset(3),c3);
    exit when ((done(1)=true) and (done(2)=true) and (done(3)=true));
    wait on c1.sync,c2.sync,c3.sync;
  end loop;
end send;
procedure recv_hse(variable done:inout boolean;
                   variable reset:inout boolean;
                   signal chan:inout channel) is
begin
  if (done = false) then
    if (reset = false) then
      if ((chan.pending_recv='Z') and
          (chan.sync='Z')) then
        chan.pending_recv<='1';
      end if;
      if (chan.sync='1') then
        chan.pending_recv<='Z';
      end if;
      if ((chan.pending_send='1') and
          (chan.pending_recv='1') and
          (chan.sync='Z')) then
        chan.sync<='1';
      end if;
      if ((chan.pending_send='Z') and
          (chan.pending_recv='Z') and
          (chan.sync='1')) then
        chan.sync <= 'Z';
        reset:=true;
      end if;
    elsif (chan.sync='Z') then
      done:=true;
    end if;
  end if;
end recv_hse;
procedure recv_hse(variable done:inout boolean;
                   variable reset:inout boolean;
                   signal chan:inout channel;signal data:out std_logic) is
begin
  if (done=false) then
    if (reset=false) then
      if ((chan.pending_recv='Z') and
          (chan.sync= 'Z')) then
        chan.pending_recv<='1';
      end if;
      if (chan.sync='1') then
        chan.pending_recv<='Z';
      end if;
      if ((chan.pending_send='1') and
          (chan.pending_recv='1') and
          (chan.sync='Z')) then
        chan.sync<='1';
        data<=chan.dataright(0);
      end if;
      if ((chan.pending_send='Z') and
          (chan.pending_recv='Z') and
          (chan.sync='1')) then
        chan.sync<='Z';
        reset:=true;
      end if;
    elsif (chan.sync='Z') then
      done:=true;
    end if;
  end if;
end recv_hse;
procedure recv_hse(variable done:inout boolean;
                   variable reset:inout boolean;
                   signal chan:inout channel;
                   signal data:out std_logic_vector) is
begin
  if (done=false) then
    if (reset=false) then
      if ((chan.pending_recv='Z') and
          (chan.sync='Z')) then
        chan.pending_recv<='1';
      end if;
      if (chan.sync='1') then
        chan.pending_recv<='Z';
      end if;
      if ((chan.pending_send='1') and
          (chan.pending_recv='1') and
          (chan.sync='Z')) then
        chan.sync<='1';
        data<=chan.dataright(data'length - 1 downto 0);
      end if;
      if ((chan.pending_send='Z') and
          (chan.pending_recv='Z') and
          (chan.sync='1')) then
        chan.sync<='Z';
        reset:=true;
      end if;
    elsif (chan.sync='Z') then
      done:=true;
    end if;
  end if;
end recv_hse;
procedure receive (signal c1:inout channel) is
  variable done:bools(1 to 1) := (others => false);
  variable reset:bools(1 to 1) := (others => false);
begin
  loop
    recv_hse(done(1),reset(1),c1);
    exit when (done(1)=true);
    wait on c1.pending_send,c1.sync,c1.pending_recv;
  end loop;
end receive;
procedure receive(signal c1:inout channel;signal d1:inout std_logic) is
  variable done:bools(1 to 1) := (others => false);
  variable reset:bools(1 to 1) := (others => false);
begin
  loop
    recv_hse(done(1),reset(1),c1,d1);
    exit when (done(1)=true);
    wait on c1.pending_send,c1.sync,c1.pending_recv;
  end loop;
end receive;
procedure receive(signal c1:inout channel;signal d1:inout std_logic;
                  signal c2:inout channel;signal d2:inout std_logic) is
  variable done:bools(1 to 2) := (others => false);
  variable reset:bools(1 to 2) := (others => false);
begin
  loop
    recv_hse(done(1),reset(1),c1,d1);
    recv_hse(done(2),reset(2),c2,d2);
    exit when ((done(1)=true) and (done(2)=true));
    wait on c1.pending_send,c1.sync,c1.pending_recv,
      c2.pending_send,c2.sync,c2.pending_recv;
  end loop;
end receive;
procedure receive(signal c1:inout channel;signal d1:inout std_logic;
                  signal c2:inout channel;signal d2:inout std_logic;
                  signal c3:inout channel;signal d3:inout std_logic) is
  variable done:bools(1 to 3) := (others => false);
  variable reset:bools(1 to 3) := (others => false);
begin
  loop
    recv_hse(done(1),reset(1),c1,d1);
    recv_hse(done(2),reset(2),c2,d2);
    recv_hse(done(3),reset(3),c3,d3);
    exit when ((done(1)=true) and (done(2)=true) and (done(3)=true));
    wait on c1.pending_send,c1.sync,c1.pending_recv,c2.pending_send,
      c2.sync,c2.pending_recv,c3.pending_send,c3.sync,c3.pending_recv;
  end loop;
end receive;
procedure receive(signal c1:inout channel;signal d1:inout std_logic_vector
                 ) is
  variable done:bools(1 to 1) := (others => false);
  variable reset:bools(1 to 1) := (others => false);
begin
  validate( d1 );
  loop
    recv_hse(done(1),reset(1),c1,d1);
    exit when (done(1)=true);
    wait on c1.pending_send,c1.sync,c1.pending_recv;
  end loop;
end receive;
procedure receive(signal c1:inout channel;signal d1:inout std_logic_vector;
                  signal c2:inout channel;signal d2:inout std_logic_vector
                  ) is
  variable done:bools(1 to 2) := (others => false);
  variable reset:bools(1 to 2) := (others => false);
  begin
    validate( d1 );
    validate( d2 );
    loop
      recv_hse(done(1),reset(1),c1,d1);
      recv_hse(done(2),reset(2),c2,d2);
      exit when ((done(1)=true) and (done(2)=true));
      wait on c1.pending_send,c1.sync,c1.pending_recv,
        c2.pending_send,c2.sync,c2.pending_recv;
    end loop;
  end receive;
  procedure receive(signal c1:inout channel;signal d1:inout std_logic_vector;
                    signal c2:inout channel;signal d2:inout std_logic_vector;
                    signal c3:inout channel;signal d3:inout std_logic_vector
                    ) is
    variable done:bools(1 to 3) := (others => false);
    variable reset:bools(1 to 3) := (others => false);
  begin
    validate( d1 );
    validate( d2 );
    validate( d3 );
    loop
      recv_hse(done(1),reset(1),c1,d1);
      recv_hse(done(2),reset(2),c2,d2);
      recv_hse(done(3),reset(3),c3,d3);
      exit when ((done(1)=true) and (done(2)=true) and (done(3)=true));
      wait on c1.pending_send,c1.sync,c1.pending_recv,c2.pending_send,
        c2.sync,c2.pending_recv,c3.pending_send,c3.sync,c3.pending_recv;
    end loop;
  end receive;
  function probe( signal chan:in channel ) return boolean is
  begin
    return ((chan.pending_send='1') or (chan.pending_recv='1'));
  end probe;
end channel;
```

A.3 HANDSHAKE.VHD

This package defines assign and guard functions on signals.

```
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
package handshake is
  procedure assign(signal sig:inout std_logic;constant val:std_logic;
                   constant l:integer;constant u:integer);
  procedure assign(signal sig1:inout std_logic;constant val1:std_logic;
                   constant l1:integer;constant u1:integer;
                   signal sig2:inout std_logic;constant val2:std_logic;
                   constant l2:integer;constant u2:integer);
  procedure assign(signal sig1:inout std_logic;constant val1:std_logic;
                   constant l1:integer;constant u1:integer;
                   signal sig2:inout std_logic;constant val2:std_logic;
                   constant l2:integer;constant u2:integer;
```

```
                 signal sig3:inout std_logic;constant val3:std_logic;
                 constant l3:integer;constant u3:integer);
  procedure vassign(signal sig:inout std_logic;constant val:std_logic;
                  constant l:integer;constant u:integer);
  procedure vassign(signal sig1:inout std_logic;constant val1:std_logic;
                  constant l1:integer;constant u1:integer;
                  signal sig2:inout std_logic;constant val2:std_logic;
                  constant l2:integer;constant u2:integer);
  procedure vassign(signal sig1:inout std_logic;constant val1:std_logic;
                  constant l1:integer;constant u1:integer;
                  signal sig2:inout std_logic;constant val2:std_logic;
                  constant l2:integer;constant u2:integer;
                  signal sig3:inout std_logic;constant val3:std_logic;
                  constant l3:integer;constant u3:integer);
  procedure guard(signal sig:in std_logic;constant val:std_logic);
  procedure guard_or(signal sig1:in std_logic;constant val1:std_logic;
                   signal sig2:in std_logic;constant val2:std_logic);
  procedure guard_or(signal sig1:in std_logic;constant val1:std_logic;
                   signal sig2:in std_logic;constant val2:std_logic;
                   signal sig3:in std_logic;constant val3:std_logic);
  procedure guard_and(signal sig1:in std_logic;constant val1:std_logic;
                    signal sig2:in std_logic;constant val2:std_logic);
  procedure guard_and(signal sig1:in std_logic;constant val1:std_logic;
                    signal sig2:in std_logic;constant val2:std_logic;
                    signal sig3:in std_logic;constant val3:std_logic);
end handshake;
package body handshake is
  procedure assign(signal sig:inout std_logic;constant val:std_logic;
                 constant l:integer;constant u:integer) is
  begin
    assert (sig /= val) report "Vacuous assignment" severity failure;
    sig <= val after delay(l,u);
    wait until sig = val;
  end assign;
  procedure assign(signal sig1:inout std_logic;constant val1:std_logic;
                 constant l1:integer;constant u1:integer;
                 signal sig2:inout std_logic;constant val2:std_logic;
                 constant l2:integer;constant u2:integer) is
  begin
    assert (sig1 /= val1 or sig2 /= val2)
      report "Vacuous assignment" severity failure;
    sig1 <= val1 after delay(l1,u1);
    sig2 <= val2 after delay(l2,u2);
    wait until sig1 = val1 and sig2 = val2;
  end assign;
  procedure assign(signal sig1:inout std_logic;constant val1:std_logic;
                 constant l1:integer;constant u1:integer;
                 signal sig2:inout std_logic;constant val2:std_logic;
                 constant l2:integer;constant u2:integer;
                 signal sig3:inout std_logic;constant val3:std_logic;
                 constant l3:integer;constant u3:integer) is
  begin
    assert (sig1 /= val1 or sig2 /= val2 or sig3 /= val3)
      report "Vacuous assignment" severity failure;
    sig1 <= val1 after delay(l1,u1);
    sig2 <= val2 after delay(l2,u2);
  sig3 <= val3 after delay(l3,u3);
  wait until sig1 = val1 and sig2 = val2 and sig3 = val3;
end assign;
procedure vassign(signal sig:inout std_logic;constant val:std_logic;
                  constant l:integer;constant u:integer) is
begin
  if (sig /= val) then
    sig <= val after delay(l,u);
    wait until sig = val;
  end if;
end vassign;
procedure vassign(signal sig1:inout std_logic;constant val1:std_logic;
                  constant l1:integer;constant u1:integer;
                  signal sig2:inout std_logic;constant val2:std_logic;
                  constant l2:integer;constant u2:integer) is
begin
  if (sig1 /= val1) then
    sig1 <= val1 after delay(l1,u1);
  end if;
  if (sig2 /= val2) then
    sig2 <= val2 after delay(l2,u2);
  end if;
  if (sig1 /= val1 or sig2 /= val2) then
    wait until sig1 = val1 and sig2 = val2;
  end if;
end vassign;
procedure vassign(signal sig1:inout std_logic;constant val1:std_logic;
                  constant l1:integer;constant u1:integer;
                  signal sig2:inout std_logic;constant val2:std_logic;
                  constant l2:integer;constant u2:integer;
                  signal sig3:inout std_logic;constant val3:std_logic;
                  constant l3:integer;constant u3:integer) is
begin
  if (sig1 /= val1) then
    sig1 <= val1 after delay(l1,u1);
  end if;
  if (sig2 /= val2) then
    sig2 <= val2 after delay(l2,u2);
  end if;
  if (sig3 /= val3) then
    sig3 <= val3 after delay(l3,u3);
  end if;
  if (sig1 /= val1 or sig2 /= val2 or sig3 /= val3) then
    wait until sig1 = val1 and sig2 = val2 and sig3 = val3;
  end if;
end vassign;
procedure guard(signal sig:in std_logic;constant val:std_logic) is
begin
  if (sig /= val) then
    wait until sig = val;
  end if;
end guard;
procedure guard_or(signal sig1:in std_logic;constant val1:std_logic;
                   signal sig2:in std_logic;constant val2:std_logic) is
begin
  if (sig1 /= val1 and sig2 /= val2) then
```

```
      wait until sig1 = val1 or sig2 = val2;
    end if;
  end guard_or;
  procedure guard_or(signal sig1:in std_logic;constant val1:std_logic;
                     signal sig2:in std_logic;constant val2:std_logic;
                     signal sig3:in std_logic;constant val3:std_logic) is
  begin
    if (sig1 /= val1 and sig2 /= val2 and sig3 /= val3) then
      wait until sig1 = val1 or sig2 = val2 or sig3 = val3;
    end if;
  end guard_or;
  procedure guard_and(signal sig1:in std_logic;constant val1:std_logic;
                      signal sig2:in std_logic;constant val2:std_logic) is
  begin
    if ( sig1 /= val1 or sig2 /= val2 ) then
      wait until sig1 = val1 and sig2 = val2;
    end if;
  end guard_and;
  procedure guard_and(signal sig1:in std_logic;constant val1:std_logic;
                      signal sig2:in std_logic;constant val2:std_logic;
                      signal sig3:in std_logic;constant val3:std_logic) is
  begin
    if (sig1 /= val1 or sig2 /= val2 or sig3 /= val3) then
      wait until sig1 = val1 and sig2 = val2 and sig3 = val3;
    end if;
  end guard_and;
end handshake;
```