Asynchronous Circuit Design
Chris J. Myers
Lecture 2: Communication Channels
Chapter 2

Communication Channels
- Channel is used as a point-to-point means of communication between two concurrently operating processes.
- Channel package (see channel.vhd) includes:
  - channel data type
  - init_channel procedure
  - send procedure
  - receive procedure
  - probe procedure

Libraries and Packages
------------------------
-- wine_example.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;

Entities/Architectures
entity wine_example is
end wine_example;
arithmetic behavior of wine_example is
-- declarations
begin
-- concurrent statements
end behavior;
Signal Declarations

type wine_list is (cabernet, merlot, zinfandel,
    chardonnay, sauvignon blanc,
    pinot noir, riesling, bubbly);

signal wine_drunk: wine_list;
signal WineryShop: channel := init_channel;
signal ShopPatron: channel := init_channel;
signal bottle: std_logic_vector(2 downto 0) := "000";
signal shelf: std_logic_vector(2 downto 0);
signal bag: std_logic_vector(2 downto 0);

Concurrent Processes: winery

    winery: process
    begin
        bottle <= selection(8,3);
        wait for delay(5,10);
        send(WineryShop, bottle);
    end process winery;

Concurrent Processes: shop

    shop: process
    begin
        receive(WineryShop, shelf);
        send(ShopPatron, shelf);
    end process shop;

Std_Logic Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'U'</td>
<td>Initialized</td>
</tr>
<tr>
<td>'X'</td>
<td>Forcing unknown</td>
</tr>
<tr>
<td>'0'</td>
<td>Forcing 0</td>
</tr>
<tr>
<td>'1'</td>
<td>Forcing 1</td>
</tr>
<tr>
<td>'Z'</td>
<td>High impedance</td>
</tr>
<tr>
<td>'W'</td>
<td>Weak unknown</td>
</tr>
<tr>
<td>'L'</td>
<td>Weak 0</td>
</tr>
<tr>
<td>'H'</td>
<td>Weak 1</td>
</tr>
<tr>
<td>'-'</td>
<td>Don't care</td>
</tr>
</tbody>
</table>
Concurrent Processes: patron

patron: process
begin
  receive(ShopPatron,bag);
  wine_drunk <= wine_list'val(conv_integer(bag));
end process patron;

Block Diagram for wine shop

Structural Modeling: shop.vhd

entity shop is
  port(wine_delivery: inout channel:=init_channel;
       wine_selling: inout channel:=init_channel);
end shop;
architecture behavior of shop is
  signal shelf: std_logic_vector(2 downto 0);
begin
  shop: process
  begin
    receive(wine_delivery,shelf);
    send(wine_selling,shelf);
  end process shop;
end behavior;

Structural Modeling: winery.vhd

entity winery is
  port(wine_shipping: inout channel:=init_channel);
end winery;
architecture behavior of winery is
  signal bottle: std_logic_vector(2 downto 0):="000";
begin
  winery: process
  begin
    bottle <= selection(8,3);
    wait for delay(5,10);
    send(wine_shipping,bottle);
  end process winery;
end behavior;
**Structural Modeling: patron.vhd**

```
entity patron is
  port (wine_buying: inout channel := init_channel);
end patron;
architecture behavior of patron is
  type wine_list is (cabernet, merlot, zinfandel, chardonnay, sauvignon_blanc, pinot_noir, riesling, bubbly);
  signal wine_drunk: wine_list;
  signal bag: std_logic_vector(2 downto 0);
begin
  patron: process
  begin
    receive (wine_buying, bag);
    wine_drunk <= wine_list'val(conv_integer(bag));
  end process patron;
end behavior;
```

**Component Declarotions**

```
component winery
  port (wine_shipping: inout channel);
end component;
component shop
  port (wine_delivery: inout channel,
        wine_selling: inout channel);
end component;
component patron
  port (wine_buying: inout channel);
end component;
begin
  THE_Winery: winery
    port map (wine_shipping => WineryShop);
  THE_Shop: shop
    port map (wine_delivery => WineryShop,
             wine_selling => ShopPatron);
  THE_Patron: patron
    port map (wine_buying => ShopPatron);
end structure;
```

**Structural Modeling: wine_example2.vhd**

```
-- wine_example2.vhd
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.channel.all;
entity wine_example is
  end wine_example;
architecture structure of wine_example is
begin
  -- component and signal declarations
  begin
    -- component instantiations
  end structure;
```

**Component Instantiations**

```
begin
  THE_Winery: winery
    port map (wine_shipping => WineryShop);
  THE_Shop: shop
    port map (wine_delivery => WineryShop,
             wine_selling => ShopPatron);
  THE_Patron: patron
    port map (wine_buying => ShopPatron);
end structure;
```
Block Diagram Including New Wine Shop

VHDL with New Wine Shop

architecture new.structure of wine_example is
  -- component declarations
  -- channel declarations
begin
  -- winery
  OLD_SHOP: shop
    port map(wine_delivery => WineryShop, wine_selling => ShopNewShop);
  NEW_SHOP: shop
    port map(wine_delivery => ShopNewShop, wine_selling => NewShopPatron);
  -- patron
end new.structure;

Deterministic Selection: if-then-else

winery2: process
begin
  bottle <= selection(8,3);
  wait for delay(5,10);
  if (wine_list'val(conv_integer(bottle)) = merlot) then
    send(WineryNewShop,bottle);
  else
    send(WineryOldShop,bottle);
  end if;
end process winery2;
**Deterministic Selection: case**

slide21

```plaintext
Deterministic Selection: case

winery3: process
begin
  bottle <= selection(8,3);
  wait for delay(5,10);
  case (wine_list'val(conv_integer(bottle))) is
  when merlot =>
    send(WineryNewShop,bottle);
  when others =>
    send(WineryOldShop,bottle);
  end case;
end process winery3;
```

**Non-deterministic Selection: case**

slide23

```plaintext
Non-deterministic Selection: case

winery5: process
variable z: integer;
begin
  bottle <= selection(8,3);
  wait for delay(5,10);
  z:= selection(2);
  case z is
  when 1 =>
    send(WineryNewShop,bottle);
  when others =>
    send(WineryOldShop,bottle);
  end case;
end process winery5;
```

**Non-deterministic Selection: if-then-else**

slide22

```plaintext
Non-deterministic Selection: if-then-else

winery4: process
variable z: integer;
begin
  bottle <= selection(8,3);
  wait for delay(5,10);
  z:= selection(2);
  if (z = 1) then
    send(WineryNewShop,bottle);
  else
    send(WineryOldShop,bottle);
  end if;
end process winery4;
```

**Repetition: for loops**

slide24

```plaintext
Repetition: for loops

winery6: process
begin
  for i in 1 to 4 loop
    bottle <= selection(8,3);
    wait for delay(5,10);
    send(WineryOldShop,bottle);
  end loop;
  for i in 1 to 3 loop
    bottle <= selection(8,3);
    wait for delay(5,10);
    send(WineryNewShop,bottle);
  end loop;
end process winery6;
```
Repetition: while loops

```
winery7: process
begin
    while (wine_list\'val(conv_integer(bottle)) /= merlot)
        loop
            bottle <= selection(8,3);
            wait for delay(5,10);
            send(WineryOldShop,bottle);
        end loop;
        bottle <= selection(8,3);
        wait for delay(5,10);
        send(WineryNewShop,bottle);
    end process winery7;
```

Repetition: infinite loops

```
winery8: process
begin
    bottle <= selection(8,3);
    wait for delay(5,10);
    send(WineryOldShop,bottle);
    loop
        bottle <= selection(8,3);
        wait for delay(5,10);
        send(WineryNewShop,bottle);
    end loop;
end process winery8;
```

Deadlock

```
producer: process
begin
    send(X,x);
    send(Y,y);
end process producer;

consumer: process
begin
    receive(Y,a);
    receive(X,b);
end process consumer;
```

The Probe

```
patron2: process
begin
    if (probe(OldShopPatron)) then
        receive(OldShopPatron,bag);
        wine_drunk <= wine_list\'val(conv_integer(bag));
    elsif (probe(NewShopPatron)) then
        receive(NewShopPatron,bag);
        wine_drunk <= wine_list\'val(conv_integer(bag));
    end if;
    wait for delay(5,10);
end process patron2;
```
**Parallel Send**

```plaintext
winery9: process
begin
  bottle1 <= selection(8,3);
  bottle2 <= selection(8,3);
  wait for delay(5,10);
  send(WineryOldShop,bottle1,WineryNewShop,bottle2);
end process winery9;
```

**Parallel Receive**

```plaintext
patron3: process
begin
  receive(OldShopPatron,bag1,NewShopPatron,bag2);
  wine_drunk1 <= wine_list\'val(conv_integer(bag1));
  wine_drunk2 <= wine_list\'val(conv_integer(bag2));
end process patron3;
```

---

**MiniMIPS: ISA**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>rd := rs + rt</td>
<td>add r1, r2, r3</td>
</tr>
<tr>
<td>sub</td>
<td>rd := rs - rt</td>
<td>sub r1, r2, r3</td>
</tr>
<tr>
<td>and</td>
<td>rd := rs &amp; rt</td>
<td>and r1, r2, r3</td>
</tr>
<tr>
<td>or</td>
<td>rd := rs</td>
<td>rt</td>
</tr>
<tr>
<td>lw</td>
<td>rt := mem[rs + offset]</td>
<td>lw r1, (32)r2</td>
</tr>
<tr>
<td>sw</td>
<td>mem[rs + offset] := rt</td>
<td>sw r1, (32)r2</td>
</tr>
<tr>
<td>beq</td>
<td>if (rs==rt) then (PC := PC + offset)</td>
<td>beq r1, r2, Loop</td>
</tr>
<tr>
<td>j</td>
<td>(PC := address)</td>
<td>(j) Loop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Opcode</th>
<th>Func</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>0</td>
<td>32</td>
</tr>
<tr>
<td>sub</td>
<td>0</td>
<td>34</td>
</tr>
<tr>
<td>and</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>or</td>
<td>0</td>
<td>37</td>
</tr>
<tr>
<td>lw</td>
<td>35</td>
<td>n/a</td>
</tr>
<tr>
<td>sw</td>
<td>43</td>
<td>n/a</td>
</tr>
<tr>
<td>beq</td>
<td>4</td>
<td>n/a</td>
</tr>
<tr>
<td>j</td>
<td>6</td>
<td>n/a</td>
</tr>
</tbody>
</table>
### MiniMIPS: Instruction Formats

#### Register Instructions

<table>
<thead>
<tr>
<th>Field</th>
<th>6</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shamt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Func</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Load/store/branch instructions

<table>
<thead>
<tr>
<th>Field</th>
<th>6</th>
<th>5</th>
<th>5</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Offset</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Jump instructions

<table>
<thead>
<tr>
<th>Field</th>
<th>6</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

### Block Diagram for MiniMIPS

![Block Diagram](image)

---

### MiniMIPS: minimips.vhd

```vhdl
-- Entity/architecture declarations
-- ieee stuff
use work.channel.all;
entity minimips is
end minimips;
architecture structure of minimips is
begin
-- Component declarations
-- Signal declarations
begin
-- Component instantiations
end structure;
```
MiniMIPS: imem.vhd

-- iee stuff
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"00000000", ..., X"00000000");
    begin
      receive(address,addr);
      instr <= imem(conv_integer(addr(2 downto 0)));
      wait for delay(5,10);
      send(data,instr);
    end process;
  end behavior;
end

MiniMIPS: fetch.vhd

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);
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);
begin
  process
    variable imem:memory:=
      (X"8c220000", -- L: lw r2,0(r1)
        X"00000000", ..., X"00000000");
    begin
      receive(address,addr);
      instr <= imem(conv_integer(addr(2 downto 0)));
      wait for delay(5,10);
      send(data,instr);
    end process;
  end behavior;
end
MiniMIPS: fetch.vhd

process
  variable branch_offset: std_logic_vector(31 downto 0);
begin
  send(mem address, PC);
  receive(mem data, instr);
  PC <= PC + 1;
  wait for delay(5,10);
  case opcode is
    when "000110" => -- j
      PC <= PC(31 downto 26) & address);
    wait for delay(5,10);
  when others =>
    send(mem instr, instr);
  end case;
end process;
end behavior;

MiniMIPS: decode.vhd

entity decode is
  port(decoded 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;

MiniMIPS: decode.vhd

type reg array is array (0 to 7) of std_logic_vector(31 downto 0);
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(6 downto 0);
alias offset: std_logic_vector(15 downto 0) is instr(15 downto 0);
signal registers reg array := (1"00000000"...);
signal reg op: std_logic_vector(31 downto 0);
signal reg rt: std_logic_vector(31 downto 0);
signal reg rd: std_logic_vector(31 downto 0);
MiniMIPS: decode.vhd

process
begin
  receive(decode_instr,instr);
  reg_rs <= reg(conv_integer(rs));
  reg Rt <= reg(conv_integer(rt));
  wait for delay(5,10);
  send(execute_op,op);
  case op is
    when "000000" => -- ALU op
      send(execute_func,func,execute Rs,reg Rs, 
          execute Rt,reg Rt);
      receive(execute Rd,reg Rd);
      reg(conv_integer(rd)) <= reg Rd;
      wait for delay(5,10);
when others => -- undefined
  assert false report "Illegal instruction" severity error;
end case;
end process;
end behavior;

MiniMIPS: execute.vhd

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;
```
MiniMIPS: execute.vhd
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
```

```
MiniMIPS: execute.vhd
process
variable addr_offset: std_logic_vector(31 downto 0);
begin
receive(execute, op, op);
case op is
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);
```

```
MiniMIPS: execute.vhd
when "000000" => -- ALU op
receive(execute, func, execute, 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);
```

```
MiniMIPS: execute.vhd
when "100011" => -- lw
receive(execute, 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);
```

```
MiniMIPS: execute.vhd

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;

MiniMIPS: dmem.vhd

entity dmem is
  port(address:inout channel:=init_channel;
       data_in:inout channel:=init_channel;
       data_out:inout channel:=init_channel;
       read_write:inout channel:=init_channel);
end dmem;
architecture behavior of dmem is
  type memory is array (0 to 7) of
    std_logic_vector(31 downto 0);
  signal addr:std_logic_vector(31 downto 0);
  signal d:std_logic_vector(31 downto 0);
  signal rw:std_logic;
  signal dmem:memory:=(X"00000000",...);
begin

MiniMIPS: dmem.vhd

 receive(address,addr);
 receive(read_write, rw);
 case rw is
 when '1' =>
   d <= dmem(conv_integer(addr(2 downto 0)));
   wait for delay(5,10);
   send(data_out,d);
 when '0' =>
   receive(data_in,d);
   dmem(conv_integer(addr(2 downto 0))) <= d;
   wait for delay(5,10);
 when others =>
   wait for delay(5,10);
end case;
end case;

RAW Hazards

- $r1$ contains 1.
- $r2$ contains 2.
add $r1,r2,r2$
add $r4,r1,r1$
Pipelined MiniMIPS: decode.vhd

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_pp, op);
wait for delay(5, 10);
end;

Pipelined MiniMIPS: decode.vhd

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_pp, op);
wait for delay(5, 10);
end;

Pipelined MiniMIPS: decode.vhd

begin
receive(decode_to wb, instr);
case wb_op is
when "000000" => -- ALU op
send(execute_func, func, execute_reg_rs, execute_reg_rt);
send(decode_to wb, instr);
receive(lock);
when "100011" => -- lw
send(execute_reg_rs, execute_reg_rt, execute_offset, offset);
send(decode_to wb, instr);
receive(lock);
end;

Pipelined MiniMIPS: decode.vhd

begin
receive(decode_to 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_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);
end;
Pipelined MiniMIPS: decode.vhd

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;

Summary

- Channel package
- Send and receive procedures
- Structural modeling
- Selection and repetition
- The probe
- Parallel composition
- MiniMIPS