Asynchronous Circuit Design

Chris J. Myers

Lecture 3: Communication Protocols
Chapter 3
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.handshake.all;
entity shopPA_dualrail is
  port (bottle1: in std_logic;
        bottle0: in std_logic;
        ack_wine: buffer std_logic:='0';
        shelf1: buffer std_logic:='0';
        shelf0: buffer std_logic:='0';
        ack_patron: in std_logic);
end shopPA_dualrail;
Naive Handshaking Level Representation

shopPA_dualrail: process
begin
  wait until ack_patron = '0';
  wait until bottle0 = '1' or bottle1 = '1';
  if bottle0='1' then shelf0<=‘1’ after delay(1,3);
  elsif bottle1='1' then shelf1<=‘1’ after delay(1,3);
  end if;
  ack_wine <= ‘1’ after delay(1,3);
  wait until ack_patron = '1';
  shelf0 <= ‘0’ after delay(1,3);
  shelf1 <= ‘0’ after delay(1,3);
  wait until bottle0 = '0' and bottle1 <= '0';
  ack_wine <= ‘0’ after delay(1,3);
end process;
Handshaking Level Representation

shopPA_dualrail: process
begin
    guard(ack_patron,'0');
    guard_or(bottle0,'1',bottle1,'1');
    if bottle0 = '1' then assign(shelf0,'1',1,3);
    elsif bottle1 = '1' then assign(shelf1,'1',1,3);
    end if;
    assign(ack_wine,'1',1,3);
    guard(ack_patron,'1');
    vassign(shelf0,'0',1,3,shelf1,'0',1,3);
    guard_and(bottle0,'0',bottle1,'0');
    assign(ack_wine,'0',1,3);
end process;
Handshake Package: *guard* Procedures

- **guard**\((s, v)\) takes a signal, \(s\), and a value, \(v\), and replaces:

  ```
  if (s /= v) then
    wait until s = v;
  end if;
  ```

- **guard**\(_or\)(\(s_1, v_1, s_2, v_2, \ldots\)) takes a set of signals and values, and replaces:

  ```
  if ((s_1 /= v_1) and (s_2 /= v_2) \ldots ) then
    wait until (s_1 = v_1) or (s_2 = v_2) \ldots ;
  end if;
  ```

- **guard**\(_and\)(\(s_1, v_1, s_2, v_2, \ldots\)) takes a set of signals and a set of values, and replaces:

  ```
  if ((s_1 /= v_1) or (s_2 /= v_2) \ldots ) then
    wait until s_1 = v_1 and s_2 = v_2 \ldots ;
  end if;
  ```
Handshake Package: *assign* Procedures

- *assign*(s,v,l,u) takes a signal, s, a value, v, a lower bound of delay, l, and an upper bound of delay, u, and replaces:

  ```
  assert (s /= v)
  report “Vacuous assignment!”
  severity failure;
  s <= v after delay(l,u);
  wait until s = v;
  ```

- *assign*(s1,v1,l1,u1,s2,v2,l2,u2) implements a parallel assignment as follows:

  ```
  assert ((s1 /= v1) or (s2 /= v2))
  report “Vacuous assignment!”
  severity failure;
  s1 <= v1 after delay(l1,u1);
  s2 <= v2 after delay(l2,u2);
  wait until (s1 = v1) and (s2 = v2);
  ```
Handshake Package: `vassign` Procedures

- **Vacuous assign (vassign)** procedure is defined below:

  ```
  if (s /= v) then
    s <= v after delay(l, u);
    wait until s = v;
  end if;
  ```

- `vassign` procedure also allows parallel assignments:

  ```
  if (s1 /= v1) then
    s1 <= v1 after delay(l1, u1);
  end if;
  if (s2 /= v2) then
    s2 <= v2 after delay(l2, u2);
  end if;
  if (s1 /= v1) or (s2 /= v2) then
    wait until s1 = v1 and s2 = v2;
  end if;
  ```
Active and Passive Ports

- Channel has an active and a passive port.
- Active port initiates communication.
- Passive port must patiently wait.
- If a process uses the probe function on a channel, it must connect to the passive port.
- If a channel is not probed, then the assignment is arbitrary.
entity shopPA is
  port (wine_delivery: inout channel:=passive;
       wine_selling: inout channel:=active);
end shopPA;
Passive/Active *wine_shop* using Bundled Data

![Diagram showing a flow from Winery to Patron via Shop and WineryShop and ShopPatron](diagram.png)
Passive/Active \textit{wine\_shop} using Bundled Data

![Diagram showing the interaction between Winery, Shop, and Patron]

- **Winery**
  - req\_wine
  - bottle
  - ack\_wine

- **Shop**
  - req\_patron
  - shelf
  - ack\_patron

- **Patron**
Two-Phase Bundled-Data Datapath

Double-edge Triggered Flip-flop

\( \text{ack}_\text{patron} \)

\( \text{req}_\text{patron} \)

\( \text{ack}_\text{wine} \)

\( \text{req}_\text{wine} \)

\( \text{delay} \)

\( \text{bottle} \)

\( \text{shelf} \)

Double-edge Triggered Flip-flop

Chris J. Myers (Lecture 3: Protocols)
library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;
use work.handshake.all;

entity winery_bundled is
  port(req_wine: buffer std_logic:='0';
       ack_wine: in std_logic;
       bottle: buffer
             std_logic_vector(2 downto 0):="000");
end winery_bundled;
architecture two_phase of winery_bundled is
begin
  winery_bundled_2phase: process
  begin
    bottle <= selection(8,3);
    wait for delay(5,10);
    assign(req_wine, not req_wine, 1, 3); -- call shop
    guard(ack_wine, req_wine); -- wine delivered
  end process;
end two_phase;
patronP_bundled_2phase : process
begin
  guard(req_patron, not ack_patron);  -- shop calls
  bag <= shelf after delay(2,4);
  wait for delay(5,10);
  assign(ack_patron, not ack_patron, 1, 3);  -- buys wine
end process;
shop_bundled_2phase: process
begin
  guard(req_wine, not ack_wine);    - winery calls
  shelf <= bottle after delay(2,4);
  wait for delay(5,10);
  assign(req_patron, not req_patron, 1,3);
    - call patron
  guard(ack_patron, req_patron);   - patron buys wine
  assign(ack_wine, not ack_wine, 1,3);  - wine sold
end process;
Four-Phase Bundled-Data Datapath

ack_wine ↔ Ctrl

req_wine → delay

bottle → D

Q → shelf

Level-sensitive Latch

ack_patron → Ctrl

req_patron ↔ Ctrl
```vhdl
winery_bundled_4phase : process
begin
    bottle <= selection(8,3);
    wait for delay(5,10);
    assign(req_wine,'1',1,3);  -- call shop
    guard(ack_wine,'1');      -- wine delivered
    assign(req_wine,'0',1,3);  -- reset req_wine
    guard(ack_wine,'0');      -- ack_wine resets
end process;
```
patronP_bundled_4phase: process
begin
  guard(req_patron,'1'); -- shop calls
  bag <= shelf after delay(2,4);
  wait for delay(5,10);
  assign(ack_patron,'1',1,3); -- patron buys wine
  guard(req_patron,'0'); -- req_patron resets
  assign(ack_patron,'0',1,3); -- reset ack_patron
end process;
shop_bundled_4phase: process
begin
    guard(req_wine,'1');  -- winery calls
    shelf <= bottle after delay(2,4);
    wait for delay(5,10);
    assign(ack_wine,'1',1,3);  -- shop receives wine
    guard(req_wine,'0');  -- req_wine resets
    assign(ack_wine,'0',1,3);  -- reset ack_wine
    assign(req_patron,'1',1,3);  -- call patron
    guard(ack_patron,'1');  -- patron buys wine
    assign(ack_patron,'0',1,3);  -- reset ack_patron
    guard(ack_patron,'0');  -- ack_patron resets
end process;
Shop_PA_reshuffled: process
begin
  guard(req_wine,'1');  -- winery calls
  shelf <= bottle after delay(2,4);
  wait for delay(5,10);
  assign(ack_wine,'1',1,3);  -- shop receives wine
  assign(req_patron,'1',1,3);  -- call patron
  guard(req_wine,'0');  -- req_wine resets
  assign(ack_wine,'0',1,3);  -- reset ack_wine
  guard(ack_patron,'1');  -- patron buys wine
  assign(req_patron,'0',1,3);  -- reset req_patron
  guard(ack_patron,'0');  -- ack_patron resets
end process;
Shop_PA_lazy_active: process
begin
  guard(req_wine,'1');  -- winery calls
  shelf <= bottle after delay(2,4);
  wait for delay(5,10);
  assign(ack_wine,'1',1,3);  -- shop receives wine
  guard(ack_patron,'0');    -- ack_patron resets
  assign(req_patron,'1',1,3);  -- call patron
  guard(req_wine,'0');      -- req_wine resets
  assign(ack_wine,'0',1,3);  -- reset ack_wine
  guard(ack_patron,'1');    -- patron buys wine
  assign(req_patron,'0',1,3);  -- reset req_patron
end process;
Four-Phase Bundled-Data Datapath

- ack_wine
- req_wine
- delay
- D
- Q
- bottle
- shelf
- Level-sensitive Latch
- ack_patron
- req_patron

Chris J. Myers (Lecture 3: Protocols) Asynchronous Circuit Design

22 / 66
Four-Phase Bundled-Data Early Protocol
Lazy-Active Shop (Early Protocol)

Shop_PA_lazy_active: process
begin
  guard(req_wine,'1');  -- winery calls
  shelf <= bottle after delay(2,4);
  wait for delay(5,10);
  guard(ack_patron,'0');  -- ack_patron resets
  assign(req_patron,'1',1,3);  -- call patron
  guard(ack_patron,'1');  -- patron buys wine
  assign(ack_wine,'1',1,3);  -- shop receives wine
  assign(req_wine,'0');  -- req_wine resets
  assign(ack_wine,'0',1,3);  -- reset ack_wine
end process;
Four-Phase Bundled-Data Late Protocol
Shop_PA_lazy_active: process
begin
  guard(req_wine,'1');  -- winery calls
  assign(req_patron,'1',1,3);  -- call patron
  guard(ack_patron,'1');  -- patron buys wine
  assign(ack_wine,'1',1,3);  -- shop receives wine
  guard(req_wine,'0');  -- req_wine resets
  shelf <= bottle after delay(2,4);
  wait for delay(5,10);
  assign(req_patron,'0',1,3);  -- reset req_patron
  guard(ack_patron,'0');  -- ack_patron resets
  assign(ack_wine,'0',1,3);  -- reset ack_wine
end process;
Four-Phase Bundled-Data Broad Protocol

(c)

req

ack

broad

1st data

2nd data
Shop_PA_lazy_active: process begin
  guard(req_wine,'1');       - winery calls
  shelf <= bottle after delay(2,4);
  wait for delay(5,10);
  assign(ack_wine,'1',1,3);  - shop receives wine
  guard(ack_patron,'0');     - ack_patron resets
  assign(req_patron,'1',1,3); - call patron
  guard(req_wine,'0');       - req_wine resets
  guard(ack_patron,'1');     - patron buys wine
  assign(ack_wine,'0',1,3);  - reset ack_wine
  assign(req_patron,'0',1,3); - reset req_patron
end process;

Need edge-triggered flip-flop
Winery_Patron : process
begin
  bottle <= selection(8,3);
  wait for delay(5,10);
  assign(req_wine,'1',1,3); - call shop
  guard(ack_wine,'1'); - wine delivered
  guard(req_patron,'1'); - shop calls patron
  bag <= shelf after delay(2,4);
  wait for delay(5,10);
  assign(ack_patron,'1',1,3); - patron buys wine
  guard(req_patron,'0'); - req_patron resets
  assign(ack_patron,'0',1,3); - reset ack_patron
  assign(req_wine,'0',1,3); - reset req_wine
  guard(ack_wine,'0'); - ack_wine resets
end process;
Shop_PA_SV: process begin
  guard(req_wine,'1');       - winery calls
  shelf <= bottle after delay(2,4);
  wait for delay(5,10);
  assign(ack_wine,'1',1,3);  - shop receives wine
  assign(x,'1',1,3);         - set x
  guard(req_wine,'0');       - req_wine resets
  assign(ack_wine,'0',1,3);  - reset ack_wine
  assign(req_patron,'1',1,3); - call patron
  guard(ack_patron,'1');     - patron buys wine
  assign(x,'0',1,3);         - reset x
  assign(req_patron,'0',1,3); - reset req_patron
  guard(ack_patron,'0');     - ack_patron resets
end process;
Passive/Active wine_shop using Dual-Rail (1 bit)
```vhdl
winery_dual_rail : process
  variable z : integer;
begin
  z := selection(2);
  case z is
    when 1 =>
      assign(bottle1,'1',1,3);
    when others =>
      assign(bottle0,'1',1,3);
  end case;
  guard(ack_wine,'1');
  vassign(bottle1,'0',1,3,bottle0,'0',1,3);
  guard(ack_wine,'0');
end process;
```
shopPA_dual_rail: process

begin
  guard(ack_patron,'0');
  guard_or(bottle0,'1',bottle1,'1');
  if bottle0 = '1' then assign(shelf0,'1',1,3);
  elsif bottle1 = '1' then assign(shelf1,'1',1,3);
  end if;
  assign(ack_wine,'1',1,3);
  guard(ack_patron,'1');
  vassign(shelf0,'0',1,3,shelf1,'0',1,3);
  guard_and(bottle0,'0',bottle1,'0');
  assign(ack_wine,'0',1,3);
end process;
patronP_dualrail : process
begin
  guard_or(shelf1,'1',shelf0,'1');
  assign(ack_patron,'1',1,3);
  guard_and(shelf1,'0',shelf0,'0');
  assign(ack_patron,'0',1,3);
end process;
Passive/Active wine_shop using Dual-Rail

Winery

Shop0

Shop1

Shop2

Patio

bottle0_1

bottle0_0

ack_wine0

bottle1_1

bottle1_0

ack_wine1

bottle2_1

bottle2_0

ack_wine2

shelf0_1

shelf0_0

ack_wine0

shelf1_1

shelf1_0

ack_wine1

shelf2_1

shelf2_0

ack_wine2

ack_patron0

ack_patron1

ack_patron2
winery_dual_rail: process
  variable z: integer;
begin
  z := selection(8);
  case z is
    when 1 =>
      assign(bottle2_0,'1',1,3,bottle1_0,'1',1,3,
             bottle0_0,'1',1,3);
    when 2 =>
      assign(bottle2_0,'1',1,3,bottle1_0,'1',1,3,
             bottle0_1,'1',1,3);
    when 3 =>
      assign(bottle2_0,'1',1,3,bottle1_1,'1',1,3,
             bottle0_0,'1',1,3);
when 4 =>
    assign(bottle2_0,'1',1,3,bottle1_1,'1',1,3,
        bottle0_1,'1',1,3);
when 5 =>
    assign(bottle2_1,'1',1,3,bottle1_0,'1',1,3,
        bottle0_0,'1',1,3);
when 6 =>
    assign(bottle2_1,'1',1,3,bottle1_0,'1',1,3,
        bottle0_1,'1',1,3);
when 7 =>
    assign(bottle2_1,'1',1,3,bottle1_1,'1',1,3,
        bottle0_0,'1',1,3);
when others =>
    assign(bottle2_1,'1',1,3,bottle1_1,'1',1,3,
        bottle0_1,'1',1,3);
end case;
guard_and(ack_wine2,'1',ack_winel,'1',
         ack_wine0,'1');
vassign(bottle2_0,'0',1,3,bottle1_0,'0',1,3,
        bottle0_0,'0',1,3);
vassign(bottle2_1,'0',1,3,bottle1_1,'0',1,3,
        bottle0_1,'0',1,3);
guard_and(ack_wine2,'0',ack_winel,'0',
         ack_wine0,'0');
end process;
patronP_dualrail : process
begin
  guard_or(shelf2_1,'1',shelf2_0,'1');
guard_or(shelf1_1,'1',shelf1_0,'1');
guard_or(shelf0_1,'1',shelf0_0,'1');
assign(ack_patron2,'1',1,3,ack_patron1,'1',1,3,
    ack_patron0,'1',1,3);
guard_and(shelf2_1,'0',shelf2_0,'0');
guard_and(shelf1_1,'0',shelf1_0,'0');
guard_and(shelf0_1,'0',shelf0_0,'0');
assign(ack_patron2,'0',1,3,ack_patron1,'0',1,3,
    ack_patron0,'0',1,3);
end process;
Two Wine Shops

- Winery
  - WineryOldShop
  - WineryNewShop
  - NewShop

Old Shop
- OldShop
- OldShopPatron
- NewShopPatron

Patron
Two Wine Shops

Winery

OldShop

NewShop

Patron

req_wine1

ack_wine1

bottle1

OldShop

ack_wine1

req_patron1

ack_patron1

shelf1

NewShop

req_wine2

ack_wine2

bottle2

NewShop

req_patron2

ack_patron2

shelf2
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;
Winery for Two Wine Shops (part I)

\[
\text{winery:process} \\
\text{variable } z : \text{ integer;} \\
\text{begin} \\
\quad z := \text{selection}(2); \\
\quad \text{bottle } <= \text{selection}(8,3); \\
\quad \textbf{wait for delay(5,10);} \\
\quad \textbf{case } z \textbf{ is} \\
\quad \quad \textbf{when } 1 \Rightarrow \\
\quad \quad \quad \text{bottle1 } <= \text{bottle after delay(2,4);} \\
\quad \quad \quad \textbf{wait for } 5 \text{ ns;} \\
\quad \quad \quad \text{assign(req_wine1,'1',1,3);} \quad \text{ - call winery} \\
\quad \quad \quad \text{guard(ack_wine1,'1');} \quad \text{ - wine delivered} \\
\quad \quad \quad \text{assign(req_wine1,'0',1,3);} \quad \text{ - reset req_wine} \\
\quad \quad \quad \text{guard(ack_wine1,'0');} \quad \text{ - ack_wine resets}
\]
when others =>
  bottle2 <= bottle after delay(2,4);
  wait for 5 ns;
  assign(req_wine2,'1',1,3);  -- call winery
  guard(ack_wine2,'1');       -- wine delivered
  assign(req_wine2,'0',1,3);  -- reset req_wine
  guard(ack_wine2,'0');       -- ack_wine resets
end case;
end process;
shop: process
begin
  receive(WineryShop, shelf);
  send(ShopPatron, shelf);
end process  shop;
Shop_PA_lazy_active: process
begin
  guard(req_wine, '1'); // winery calls
  shelf <= bottle after delay(2,4);
  wait for delay(5,10);
  assign(ack_wine, '1', 1, 3); // shop receives wine
  guard(ack_patron, '0'); // ack_patron resets
  assign(req_patron, '1', 1, 3); // call patron
  guard(req_wine, '0'); // req_wine resets
  guard(ack_patron, '1'); // patron buys wine
  assign(ack_wine, '0', 1, 3); // reset ack_wine
  assign(req_patron, '0', 1, 3); // reset req_patron
end process;
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;
patronP: process
begin
  if (req_patron1 = '1') then
    bag <= shelf1 after delay(2,4);
    wait for delay(5,10);
    assign(ack_patron1,'1',1,3); - patron buys wine
  guard(req_patron1,'0'); - req_patron resets
  assign(ack_patron1,'0',1,3); - reset ack_patron
  wine_drunk <= wine_list'val(conv_integer(bag));
elsif (req_patron2 = '1') then
  bag <= shelf2 after delay(2,4);
  wait for delay(5,10);
  assign(ack_patron2,'1',1,3);  -- patron buys wine
  guard(req_patron2,'0');       -- req_patron resets
  assign(ack_patron2,'0',1,3);  -- reset ack_patron
  wine_drunk <= wine_list'val(conv_integer(bag));
end if;
wait for delay(1,2);
end process;
shop: process
begin
    receive(WineryShop, shelf);
    send(ShopPatron, shelf);
end process shop;
Circuit for Looping Constructs

Asynchronous Circuit Design
Circuit for Assignment to *shelf*

Asynchronous Circuit Design

Chris J. Myers (Lecture 3: Protocols)
Circuit for Assignment from Two Locations

In1

EN

req₁

ack₁

Call

REG

Out

In2

EN

req₂

ack₂

Chris J. Myers (Lecture 3: Protocols)
Circuit for Receive Procedure

Asynchronous Circuit Design

Chris J. Myers (Lecture 3: Protocols)
Circuit for Send Procedure

Asynchronous Circuit Design
if (cond1) then
  S1;
elsif (cond2) then
  S2;
else
  S3;
end if;
Circuit for Selection Statement

Asynchronous Circuit Design

Chris J. Myers  (Lecture 3: Protocols)
Circuit for Sequential Composition

req → S1 → S2 → ack
Circuit for Receive followed by Send

Asynchronous Circuit Design
Circuit for Parallel Composition

S1

S2

C

req

ack
Unoptimized Circuit for the wine_shop

Asynchronous Circuit Design
Circuit after CALL Module Optimization
Circuit after SEL and Merge Module Optimizations
Final Circuit for the \textit{wine\_shop}
Summary

- guard, assign, and delay functions
- Active and passive protocols
- Handshaking expansion
- Reshuffling
- State variable insertion
- Dual-rail data encoding
- Syntax-directed translation