Asynchronous Circuit Design

Chris J. Myers

Lecture 1: Introduction
Preface and Chapter 1
All events are synchronized to a single global clock.
Synchronous Advantages

- Simple way to implement sequencing.
- Widely taught and understood.
- Available components.
- Simple way to deal with noise and **hazards**.
Synchronous Disadvantages

- Clock distribution is difficult due to *clock skew*.
- Worst-case design.
- Sensitive to variations in physical parameters.
- Not modular.
- Power consumption.
Synchronization is achieved without a global clock.
Asynchronous Advantages - Most Often Cited (Al Davis)
Global synchrony doesn’t exist anyway
Intrinsic elegance

Global synchrony doesn’t exist anyway
Asynchronous Advantages - Most Often Cited (Al Davis)

1. Intellectual challenge
2. Intrinsic elegance
3. Global synchrony doesn’t exist anyway
Easier to exploit concurrency

8. Intellectual challenge

9. Intrinsic elegance

10. Global synchrony doesn’t exist anyway
Avoid clock distribution costs
Easier to exploit concurrency
Intellectual challenge
Intrinsic elegance
Global synchrony doesn’t exist anyway
Asynchronous Advantages - Most Often Cited (Al Davis)

1. Metastability has time to end
2. Avoid clock distribution costs
3. Easier to exploit concurrency
4. Intellectual challenge
5. Intrinsic elegance
6. Global synchrony doesn’t exist anyway
Asynchronous Advantages - Most Often Cited (Al Davis)

1. No clock alignment at the interfaces
2. Metastability has time to end
3. Avoid clock distribution costs
4. Easier to exploit concurrency
5. Intellectual challenge
6. Intrinsic elegance
7. Global synchrony doesn’t exist anyway
Asynchronous Advantages - Most Often Cited (Al Davis)

1. Ease of modular composition
2. No clock alignment at the interfaces
3. Metastability has time to end
4. Avoid clock distribution costs
5. Easier to exploit concurrency
6. Intellectual challenge
7. Intrinsic elegance
8. Global synchrony doesn’t exist anyway
Asynchronous Advantages - Most Often Cited (Al Davis)

1. Power consumed only where needed
2. Ease of modular composition
3. No clock alignment at the interfaces
4. Metastability has time to end
5. Avoid clock distribution costs
6. Easier to exploit concurrency
7. Intellectual challenge
8. Intrinsic elegance
9. Global synchrony doesn’t exist anyway
Asynchronous Advantages - Most Often Cited (Al Davis)

1. Achieve average case performance
2. Power consumed only where needed
3. Ease of modular composition
4. No clock alignment at the interfaces
5. Metastability has time to end
6. Avoid clock distribution costs
7. Easier to exploit concurrency
8. Intellectual challenge
9. Intrinsic elegance
10. Global synchrony doesn’t exist anyway
It's none of your business
Clock radiation causes hair loss
It's none of your business
Synchronous design gives me gas
Clock radiation causes hair loss
It's none of your business
World problems stem from glitches
Synchronous design gives me gas
Clock radiation causes hair loss
It's none of your business
I don’t understand synchronous circuits
World problems stem from glitches
Synchronous design gives me gas
Clock radiation causes hair loss
It’s none of your business
People and circuits need to play by the same rules
I don’t understand synchronous circuits
World problems stem from glitches
Synchronous design gives me gas
Clock radiation causes hair loss
It’s none of your business
Gee - I really don’t know
People and circuits need to play by the same rules
I don’t understand synchronous circuits
World problems stem from glitches
Synchronous design gives me gas
Clock radiation causes hair loss
It’s none of your business
I like to be different
Gee - I really don’t know
People and circuits need to play by the same rules
I don’t understand synchronous circuits
World problems stem from glitches
Synchronous design gives me gas
Clock radiation causes hair loss
It’s none of your business
I like reinventing wheels
I like to be different
Gee - I really don’t know
People and circuits need to play by the same rules
I don’t understand synchronous circuits
World problems stem from glitches
Synchronous design gives me gas
Clock radiation causes hair loss
It’s none of your business
1. It really pisses my boss off
2. I like reinventing wheels
3. I like to be different
4. Gee - I really don’t know
5. People and circuits need to play by the same rules
6. I don’t understand synchronous circuits
7. World problems stem from glitches
8. Synchronous design gives me gas
9. Clock radiation causes hair loss
10. It’s none of your business
Asynchronous Challenges

- Lack of mature computer-aided design tools.
- Large area overhead for the removal of hazards.
- Average-case delay can be large.
- Lack of designer experience.
Every design method traces its roots to one of two individuals:
- Huffman - fundamental-mode circuits.
- Muller - speed-independent circuits.
Key Asynchronous Circuit Designs

- ILLIAC (1952) and ILLAC2 (1962) - U. of Illinois
- Atlas (1962) and MU-5 (1966) - U. of Manchester
- Macromodules (60s-70s) - Washington U., St. Louis
- First commercial graphics system (70s) - Evans & Sutherland
- DDM dataflow computer (1978) - U. of Utah
- First asynchronous microprocessor (1989) - Caltech
- First code-compatible processor (1994) - U. of Manchester
- Commercial pager (90s) - Phillips
- RAPPID (1995-9) - Intel
Asynchronous Startups

- Handshake Solutions - Microcontrollers (Phillips)
- Fulcrum - Ethernet Switches (Caltech)
- Silistix - Self-timed interconnect (U. of Manchester)
- Achronix Semiconductor - Asynchronous FPGAs (Cornell)
Asynchronous Startups

- **Handshake Solutions** - Microcontrollers (Phillips)
- **Fulcrum** - Ethernet Switches (Caltech) ← acquired by Intel
- **Silistix** - Self-timed interconnect (U. of Manchester)
- **Achronix Semiconductor** - Asynchronous FPGAs (Cornell) ← founder left
Wine Shop Problem Specification

- Small winery and wine shop in Southern Utah.
- Only a single wine patron.
- Wine shop only has a single small shelf.
- Synchronous versus asynchronous wine shopping.
Channels of Communication

Winery → WineryShop → Shop → ShopPatron → Patron
Winery: process
begin
    send(WineryShop, bottle);
end process;
Shop: process
begin
    receive(WineryShop, shelf);
    send(ShopPatron, shelf);
end process;
Patron: process
begin
    receive(ShopPatron, bag);
end process;
Shop: \texttt{process}
begin
    req\_wine; \text{ - call winery}
    ack\_wine; \text{ - wine arrives}
    req\_patron; \text{ - call patron}
    ack\_patron; \text{ - patron buys wine}
end process;
Shop: process
begin
  assign(req_wine,'1'); - call winery
  guard(ack_wine,'1'); - wine arrives
  assign(req_patron,'1'); - call patron
  guard(ack_patron,'1'); - patron buys wine
end process;
Shop_2Phase: process
begin
assign(req_wine,'1'); - call winery
guard(ack_wine,'1'); - wine arrives
assign(req_patron,'1'); - call patron
guard(ack_patron,'1'); - patron buys wine
assign(req_wine,'0'); - call winery
guard(ack_wine,'0'); - wine arrives
assign(req_patron,'0'); - call patron
guard(ack_patron,'0'); - patron buys wine
end process;
Waveform for 2-Phase Protocol

req_wine

ack_wine

req_patron

ack_patron
4-Phase Protocol: Active/Active

Shop_4Phase: 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
    assign(req_patron,'1'); - call patron
    guard(ack_patron,'1'); - patron buys wine
    assign(req_patron,'0'); - reset req_patron
    guard(ack_patron,'0'); - ack_patron resets
end process;
Waveform for 4-Phase Protocol

req_wine

ack_wine

req_patron

ack_patron
4-Phase Protocol: Passive/Active

Shop_PA: process
begin
  guard(req_wine,'1'); // winery calls
  assign(ack_wine,'1'); // wine is received
  guard(req_wine,'0'); // req_wine resets
  assign(ack_wine,'0'); // reset ack_wine
  assign(req_patron,'1'); // call patron
  guard(ack_patron,'1'); // patron buys wine
  assign(req_patron,'0'); // reset req_patron
  guard(ack_patron,'0'); // ack_patron resets
end process;
Shop_PP: process
Shop_PP: process
begin
end process;
Shop_PP: process
begin
  guard(req_wine,'1');  % winery calls
end process;
4-Phase Protocol: Passive/Passive

Shop_PP: process
begin
  guard(req_wine,'1');  - winery calls
  assign(ack_wine,'1');  - wine is received
end process;
Shop_PP: process
begin
  guard(req_wine,'1');  - winery calls
  assign(ack_wine,'1');  - wine is received
  guard(req_wine,'0');  - req_wine resets
end process;
Shop_PP : process
begin
  guard(req_wine,'1'); - winery calls
  assign(ack_wine,'1'); - wine is received
  guard(req_wine,'0'); - req_wine resets
  assign(ack_wine,'0'); - reset ack_wine
end process;
Shop_PP: `process`

begin
  guard(req_wine,'1');  // winery calls
  assign(ack_wine,'1');  // wine is received
  guard(req_wine,'0');   // req_wine resets
  assign(ack_wine,'0');  // reset ack_wine
  guard(req_patron,'1'); // patron calls

end process;
4-Phase Protocol: Passive/Passive

Shop_PP: process
begin
  guard(req_wine,'1'); - winery calls
  assign(ack_wine,'1'); - wine is received
  guard(req_wine,'0'); - req_wine resets
  assign(ack_wine,'0'); - reset ack_wine
  guard(req_patron,'1'); - patron calls
  assign(ack_patron,'1'); - sells wine
end process;
Shop_PP : process
begin
  guard(req_wine,'1'); - winery calls
  assign(ack_wine,'1'); - wine is received
  guard(req_wine,'0'); - req_wine resets
  assign(ack_wine,'0'); - reset ack_wine
  guard(req_patron,'1'); - patron calls
  assign(ack_patron,'1'); - sells wine
  guard(req_patron,'0'); - req_patron resets
end process;
Shop_PP: process
begin
  guard(req_wine,'1'); - winery calls
  assign(ack_wine,'1'); - wine is received
  guard(req_wine,'0'); - req_wine resets
  assign(ack_wine,'0'); - reset ack_wine
  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;
Shop_AA: 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
assign(req_patron,'1'); // call patron
guard(ack_patron,'1');  // patron buys wine
assign(req_patron,'0'); // reset req_patron
guard(ack_patron,'0');  // ack_patron resets
end process;
Shop_AA: 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
⇒ state coding problem here
    assign(req_patron,'1'); // call patron
    guard(ack_patron,'1'); // patron buys wine
    assign(req_patron,'0'); // reset req_patron
    guard(ack_patron,'0'); // ack_patron resets
end process;
Shop_AA_reshuffled: process
begin
assign(req_wine,'1');  - call winery
guard(ack_wine,'1');  - wine arrives
assign(req_patron,'1');  - call patron
guard(ack_patron,'1');  - patron buys wine
assign(req_wine,'0');  - reset req_wine
guard(ack_wine,'0');  - ack_wine resets
assign(req_patron,'0');  - reset req_patron
guard(ack_patron,'0');  - ack_patron resets
end process;
Shop_AA_reshuffled: process
begin
    assign(req_wine,'1');
    guard(ack_wine,'1');
    assign(req_patron,'1');
    guard(ack_patron,'1');
    assign(req_wine,'0');
    guard(ack_wine,'0');
    assign(req_patron,'0');
    guard(ack_patron,'0');
end process;
AFSM and Huffman Flow Table (A/A reshuffled)

\[\begin{array}{cccc}
0 & 1,10 & 0,00 & 2,11 \\
1 & 1 & 2,11 & \\
2 & 3,01 & 2 & \\
3 & 0,00 & 3,01 & \\
\end{array}\]
AFSM and Huffman Flow Table (A/A reshuffled)

\[
\begin{array}{c|c|c|c|c}
   & 00 & 01 & 11 & 10 \\
\hline
0 & \textcircled{10} & \textcircled{00} & - & 2,11 \\
2 & - & - & 3,01 & \textcircled{2}11 \\
3 & - & 0,00 & \textcircled{3}01 & - \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
   & 00 & 01 & 11 & 10 \\
\hline
0 & \textcircled{10} & \textcircled{00} & - & 2,11 \\
2 & - & - & 3,01 & \textcircled{2}11 \\
3 & - & 0,00 & \textcircled{3}01 & - \\
\end{array}
\]

req_wine / req_patron

ack_wine / ack_patron
AFSM and Huffman Flow Table (A/A reshuffled)

<table>
<thead>
<tr>
<th>的状态</th>
<th>00/10</th>
<th>01/00</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ack_wine / ack_patron</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>req_wine / req_patron</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>
AFSM and Huffman Flow Table (A/A reshuffled)

```
<table>
<thead>
<tr>
<th></th>
<th>00/10</th>
<th>01/00</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>10/11</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>11/01</td>
<td></td>
</tr>
</tbody>
</table>
```

<table>
<thead>
<tr>
<th>ack_wine / ack_patron</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>req_wine / req_patron</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>
Karnaugh Maps for Huffman’s A/A Reshuffled Circuit

| ack_patron | req_wine | | req_patron |
|------------|----------| |----------|
| 0          | 1        | | 0        |
| 0          | 1        | | 0        |
| 1          | 0        | | 1        |

- \(ack_{\text{wine}}\)
- \(ack_{\text{wine}}\)
- \(req_{\text{wine}}\)
- \(req_{\text{patron}}\)
Active/Active Reshuffled Circuit

req_wine → ack_patron

ack_wine → req_patron
Active/Active Reshuffled Circuit

```
req_wine  
\[0\]  
ack_wine  
\[0\]  
ack_patron  
\[0\]  
req_patron  
\[0\]
```
Active/Active Reshuffled Circuit

req_wine

ack_wine

ack_patron

req_patron

0 1

0

0
Active/Active Reshuffled Circuit

req_wine 1 0
ack_wine 1 0
ack_patron 0
req_patron 0
Active/Active Reshuffled Circuit

\[
\begin{align*}
\text{req\_wine} & \quad 1 \quad \rightarrow \quad 0 \quad \text{ack\_patron} \\
\text{ack\_wine} & \quad 1 \quad \rightarrow \quad 1 \quad \text{req\_patron}
\end{align*}
\]
Active/Active Reshuffled Circuit

req_wine \rightarrow 1 \rightarrow \text{ack_paton}

ack_wine \rightarrow 1 \rightarrow \text{req_paton}
Active/Active Reshuffled Circuit

\[ \text{req\_wine} \quad 0 \quad \text{ack\_patron} \quad 1 \]

\[ \text{ack\_wine} \quad 1 \quad \text{req\_patron} \quad 1 \]
Active/Active Reshuffled Circuit

req_wine

ack_wine

1

ack_patron

req_patron

0

0

1
Active/Active Reshuffled Circuit

req_wine

ack_wine

ack_patron

req_patron

0

1

0

0
Active/Active Reshuffled Circuit

```
req_wine 0

ack_wine 0

ack_patron

req_patron
```
This circuit is *delay-insensitive*. 
Shop_PA: \textbf{process} \begin{verbatim}
begin
  guard(req_wine,'1'); - winery calls
  assign(ack_wine,'1'); - wine is received
  guard(req_wine,'0'); - req_wine resets
  assign(ack_wine,'0'); - reset ack_wine
  assign(req_patron,'1'); - call patron
  guard(ack_patron,'1'); - patron buys wine
  assign(req_patron,'0'); - reset req_patron
  guard(ack_patron,'0'); - ack_patron resets
end process;
\end{verbatim}
Shop_PA_reshuffled: process
begin
  guard(req_wine,'1');  -- winery calls
  assign(ack_wine,'1');  -- receives wine
  assign(req_patron,'1');  -- call patron
  guard(req_wine,'0');  -- req_wine resets
  assign(ack_wine,'0');  -- reset ack_wine
  guard(ack_patron,'1');  -- patron buys wine
  assign(req_patron,'0');  -- reset req_patron
  guard(ack_patron,'0');  -- ack_patron resets
end process;
Shop_PA_lazy_active: process
begin
  guard(req_wine,'1');  -- winery calls
  assign(ack_wine,'1');  -- receives wine
  guard(ack_patron,'0');  -- ack_patron resets
  assign(req_patron,'1');  -- call patron
  guard(req_wine,'0');  -- req_wine resets
  assign(ack_wine,'0');  -- reset ack_wine
  guard(ack_patron,'1');  -- patron buys wine
  assign(req_patron,'0');  -- reset req_patron
end process;
Petri-net (P/LA reshuffled)
Petri-net (P/LA reshuffled)

Asynchronous Circuit Design

Chris J. Myers  (Lecture 1: Introduction)
Shop_PA_lazy_active: process
begin
  guard(req_wine,'1');
  assign(ack_wine,'1');
  guard(ack_patron,'0');
  assign(req_patron,'1');
  guard(req_wine,'0');
  assign(ack_wine,'0');
  guard(ack_patron,'1');
  assign(req_patron,'0');
end process;
LPN (P/LA reshuffled)
From LPN to State Graph to Circuit

{~ack_wine}
<req_wine:=T>
{ack_wine}
<req_wine:=F>

{req_wine}
<ack_wine:=T>
{ack_patron}
<req_patron:=F>

{~ack_patron}
<req_patron:=T>
{~req_wine}
<ack_wine:=F>

{req_patron}
<ack_patron:=T>
{~req_patron}
<ack_patron:=F>
State Graph for P/LA Reshuffled (⟨ rw, ap, aw, rp ⟩)
### Karnaugh Maps for Passive/Lazy-Active Reshuffled

**Ack_Wine**

<table>
<thead>
<tr>
<th>req_wine/ack_patron</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Ack_Wine**

<table>
<thead>
<tr>
<th>req_wine/ack_patron</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Passive/Lazy-Active Reshuffled Circuit

req_wine

ack_wine

C

C

req_patron

ack_patron
Passive/Lazy-Active Reshuffled Circuit

```
req_wine 0
ack_wine 0
```

```
0 req_patron
0 ack_patron
```
Passive/Lazy-Active Reshuffled Circuit

Asynchronous Circuit Design

Chris J. Myers (Lecture 1: Introduction)
Passive/Lazy-Active Reshuffled Circuit

req_wine \rightarrow c \rightarrow req_patron

ack_wine \rightarrow c

\begin{array}{c}
1
\end{array}

\begin{array}{c}
0
\end{array}

\begin{array}{c}
1
\end{array}

\begin{array}{c}
0
\end{array}
Passive/Lazy-Active Reshuffled Circuit

\[ \text{req\_wine} \quad 1 \]
\[ \text{ack\_wine} \quad 1 \]

\[ \text{c} \]

\[ \text{req\_patron} \quad 1 \]
\[ \text{ack\_patron} \quad 1 \]
Passive/Lazy-Active Reshuffled Circuit
Passive/Lazy-Active Reshuffled Circuit

```
0
req_wine

0
ack_wine

0

1
req_patron

1
ack_patron
```
Passive/Lazy-Active Reshuffled Circuit

req_wine \( \rightarrow \) 0
ack_wine \( \rightarrow \) 0

\( c \)

\( \rightarrow \) \( \rightarrow \) \( \rightarrow \) \( \rightarrow \) \( \rightarrow \)

\( \rightarrow \) \( \rightarrow \) \( \rightarrow \) \( \rightarrow \) \( \rightarrow \)

req_patron \( \rightarrow \) 0

ack_patron \( \rightarrow \) 1
This circuit is **delay-insensitive**.
Shop_AA: 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
  assign(req_patron,'1'); - call patron
  guard(ack_patron,'1'); - patron buys wine
  assign(req_patron,'0'); - reset req_patron
  guard(ack_patron,'0'); - ack_patron resets
end process;
Shop_AA_state_variable: process
begin
  assign(req_wine,'1'); - call winery
  guard(ack_wine,'1'); - wine arrives
  assign(x,'1'); - set state variable
  assign(req_wine,'0'); - reset req_wine
  guard(ack_wine,'0'); - ack_wine resets
  assign(req_patron,'1'); - call patron
  guard(ack_patron,'1'); - patron buys wine
  assign(x,'0'); - reset state variable
  assign(req_patron,'0'); - reset req_patron
  guard(ack_patron,'0'); - ack_patron resets
end process;
Active/Active State Variable Circuit

\[\text{req\_wine}^+, \text{ack\_wine}^+, x^+, \text{req\_wine}^-, \text{ack\_wine}^-, \text{req\_patron}^+, \text{ack\_patron}^+\]
Active/Active State Variable Circuit

\[ \text{req}_\text{wine}+, \text{ack}_\text{wine}+, \ x+, \ \text{req}_\text{wine}-, \ \text{ack}_\text{wine}-, \ \text{req}_\text{patron}+, \ \text{ack}_\text{patron}+, \ u1-, \ u2-, \ x-, \ u4+, \ u6- \]
req_wine+, ack_wine+, x+, req_wine-, ack_wine-, req_patron+, ack_patron+, u1-, u2-, x-, u4+, u6-, req_wine glitches!
Huffman Circuits

- Bounded gate and wire delay model.
- Circuit does not need to be closed.
- Single-input change fundamental mode.
- One input changes $\rightarrow$ output changes $\rightarrow$ state changes.
- May need to add delay in fed back state variables.

David Huffman
Active/Active Protocol

Shop_AA: 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
assign(req_patron,'1'); - call patron
guard(ack_patron,'1'); - patron buys wine
assign(req_patron,'0'); - reset req_patron
guard(ack_patron,'0'); - ack_patron resets

end process;
AFSM and Huffman Flow Table (A/A)

\[
\begin{array}{c|c|c|c|c|c}
\text{req\_wine / req\_patron} & \text{00} & \text{01} & \text{11} & \text{10} \\
\hline
0 & 1, \quad \text{0} & \text{0} & \text{0} & \text{0} \\
1 & \text{1} & \text{1} & \text{1} & \text{2} \\
2 & \text{3} & \text{0} & \text{0} & \text{2} \\
3 & \text{3} & \text{0} & \text{0} & \text{0} \\
\end{array}
\]

\text{ack\_wine / ack\_patron}

\[
\begin{array}{c|c|c|c|c|c}
\text{req\_wine / req\_patron} & \text{00} & \text{01} & \text{11} & \text{10} \\
\hline
0 & 1, \quad \text{0} & \text{0} & \text{0} & \text{0} \\
1 & \text{1} & \text{1} & \text{1} & \text{2} \\
2 & \text{3} & \text{0} & \text{0} & \text{2} \\
3 & \text{3} & \text{0} & \text{0} & \text{0} \\
\end{array}
\]
Reduced AFSM and Huffman Flow Table (A/A)

\[\begin{array}{c|cccc}
\text{req\_wine / req\_patron} & 00 & 01 & 11 & 10 \\
\hline
0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 \\
\end{array}\]

\[\begin{array}{c|cccc}
\text{ack\_wine / ack\_patron} & 00 & 01 & 11 & 10 \\
\hline
0 & 10 & 00 & - & 1, -0 \\
1 & 01 & 0, 0 & - & 00 \\
\end{array}\]
### Karnaugh Maps for Huffman’s A/A Circuit

#### ack_wine/ack_patron

<table>
<thead>
<tr>
<th>$x$</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

#### req_wine

<table>
<thead>
<tr>
<th>$x$</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

#### req_patron

<table>
<thead>
<tr>
<th>$x$</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>
Huffman’s Active/Active Circuit

req_wine, ack_wine+, X+, x+, req_wine-, ack_wine-, req_patron+, ack_patron+
Huffman’s Active/Active Circuit

\[ \text{req}_\text{wine}, \text{ack}_\text{wine}, \ X+, \ x+, \ \text{req}_\text{wine}^-, \ \text{ack}_\text{wine}^-, \ \text{req}_\text{patron}^+, \ \text{ack}_\text{patron}^+ \]

x will not go low until after both \( u2^- \) and \( u6^- \) due to feedback delay assumption.
Muller Circuits

- Unbounded gate delay model.
- Wire delays are assumed to be negligible.
- Forks are assumed to be isochronic.
- Model called speed-independent.

David Muller
Muller’s Active/Active Circuit

req_wine [0,inf] ack_patron

ack_wine

x [0,inf] req_patron

Chris J. Myers (Lecture 1: Introduction)
Muller’s Active/Active Circuit

\( \text{req\_wine}^+, \text{ack\_wine}^+, x^+, \text{req\_wine}^-, \text{ack\_wine}^-, \text{req\_patron}^+, \text{ack\_patron}^+ \)
Muller’s Active/Active Circuit

\[ \text{req}_\text{wine}, \text{ack}_\text{wine}, x+, \text{req}_\text{wine}-, \text{ack}_\text{wine}-, \text{req}_\text{patron}+, \text{ack}_\text{patron}+ \]

\text{ack}_\text{patron} \text{ change felt at both } x \text{ and } \text{req}_\text{wine} \text{ gates simultaneously due to isochronic fork assumption.}
Shop_AA_timed:

```vhdl
process
begin
  assign(req_wine,'1',0,1); -- call winery
  assign(req_patron,'1',0,1); -- call patron
  -- wine arrives and patron arrives
  guard_and(ack_wine,'1',ack_patron,'1');
  assign(req_wine,'0',0,1);
  assign(req_patron,'0',0,1);
  -- wait for ack_wine and ack_patron to reset
  guard_and(ack_wine,'0',ack_patron,'0');
end process;
```
Timed Winery and Patron

```
winery: process
begin
  guard(req_wine,'1');  — wine requested
  assign(ack_wine,'1',2,3);  — deliver wine
  guard(req_wine,'0');
  assign(ack_wine,'0',2,3);
end process;
patron: process
begin
  guard(req_patron,'1');  — shop called
  assign(ack_patron,'1',5,inf);  — buy wine
  guard(req_patron,'0');
  assign(ack_patron,'0',5,7);
end process;
```
LPN for Timed Wine Shop Example

Asynchronous Circuit Design
State vector: \((\text{ack\_wine}, \text{ack\_patron}, \text{req\_wine}, \text{req\_patron})\)
### Karnaugh Maps for Timed Circuit

#### req_wine/req_patron

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>01</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

#### req_wine

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>01</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

#### req_patron

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>01</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Cycle time is the delay from when the patron gets one bottle of wine until he can get another.

Assuming the timed circuit delays are uniformly distributed except that the patron is extremely unlikely to take more than 10 minutes, we obtain the following cycle times:

- Muller and Huffman’s circuits (A/A SV) - 21.5 minutes
- Original (A/A reshuffled) - 20.6 minutes
- Timed circuit - 15.8 minutes
**Validation versus Verification**

- **Validation** is simulation of interesting situations.
- **Verification** is exhaustive checks of all possible situations.
  - Can check that circuit *conforms* to the specification.
  - Can check that protocol has certain properties.
Sample Properties

- The wine arrives before the patron:
  - Always($ack_{\text{patron}} \Rightarrow ack_{\text{wine}}$)
- When the wine is requested, it eventually arrives:
  - $req_{\text{wine}} \Rightarrow \text{Eventually}(ack_{\text{wine}})$
Summary of Course Topics

- Communication Channels
- Communication Protocols
- Graphical Representations
- Huffman Circuits
- Muller Circuits
- Timed Circuits
- Verification
- Applications