Asynchronous Circuit Design

Chris J. Myers
Lecture 1: Introduction
Preface and Chapter 1

Synchronous Systems

- 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.
Asynchronous Systems

- Synchronization is achieved without a global clock.

Asynchronous Advantages

- Elimination of clock distribution problems.
- Average-case performance.
- Adaptivity to processing and environmental variations.
- Component modularity.
- Lower system power requirements.

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.

Asynchronous Circuit History

- 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

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

- Channel connections:
  - Winery to WineryShop
  - Shop to ShopPatron
  - Patron

Channels of Communication in VHDL

```
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;
```
Event Protocol

Shop: process
begin
  req_wine; -- call winery
  ack_wine; -- wine arrives
  req_patron; -- call patron
  ack_patron; -- patron buys wine
end process;

Signal Protocol

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
  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;

2-Phase Protocol

Shop2Phase: 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
\[\text{波形图}\]

ack_wine
\[\text{波形图}\]

req_patron
\[\text{波形图}\]

ack_patron
\[\text{波形图}\]
4-Phase Protocol: Active/Active

ShopA: 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;

4-Phase Protocol: Passive/Active

ShopA: 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;

Waveform for 4-Phase Protocol

req_wine
ack_wine
req_patron
ack_patron
Active/Active Reshuffled

Shop\text{AA}\text{reshuffled}:\text{process}
begin
assign(req\text{wine},'1'); -- call winery
guard(ack\text{wine},'1'); -- wine arrives
assign(req\text{patron},'1'); -- call patron
guard(ack\text{patron},'1'); -- patron buys wine
assign(req\text{wine},'0'); -- reset req\text{wine}
guard(ack\text{wine},'0'); -- ack\text{wine resets}
assign(req\text{patron},'0'); -- reset req\text{patron}
guard(ack\text{patron},'0'); -- ack\text{patron resets}
end process;

Passive/Active Reshuffled

Shop\text{PA}\text{reshuffled}:\text{process}
begin
guard(req\text{wine},'1'); -- winery calls
assign(ack\text{wine},'1'); -- receives wine
guard(ack\text{patron},'0'); -- ack\text{patron resets}
assign(req\text{patron},'1'); -- call patron
guard(req\text{wine},'0'); -- req\text{wine resets}
assign(ack\text{wine},'0'); -- reset ack\text{wine}
guard(ack\text{patron},'1'); -- patron buys wine
assign(req\text{patron},'0'); -- reset req\text{patron}
end process;

Active/Active Circuit

Slide 21

Passive/Active Circuit

Slide 23
AFSM and Huffman Flow Table (A/A reshuffled)

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

req_wine / req_patron

Petri-net (P/A reshuffled)

TEL Structure (P/A reshuffled)

A/A Reshuffled Circuit
P/A Reshuffled Circuit

Active/Active State Variable

Slide 30

Active/Active 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;

A/A SV Circuit

AFSM and Huffman Flow Table (A/A SV)
Reduced AFSM and Huffman Flow Table (A/A SV)

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

Karnaugh Maps for Huffman’s A/A SV Circuit

<table>
<thead>
<tr>
<th></th>
<th>ack_wine/ack_patron</th>
<th>req_wine/req_patron</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Huffman’s A/A SV Circuit

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>req_wine</td>
<td>req_patron</td>
</tr>
<tr>
<td>x</td>
<td>0 0</td>
</tr>
<tr>
<td>req_wine</td>
<td>req_patron</td>
</tr>
<tr>
<td>x</td>
<td>1 1</td>
</tr>
</tbody>
</table>

Huffman’s Assumptions

- **Bounded gate and wire delay model.**
- Circuit does not need to be closed.
- **Single-input change fundamental mode.**
- One input changes → output changes → state changes.
- May need to add delay in feedback state variables.
Muller’s Active/Active SV Circuit

Slide 37

Muller’s Assumptions

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

Slide 38

Timed Wine Shop

Slide 39

Timed Winery and Patron

Slide 40
TEL Structure for Timed Wine Shop Example

State Graph for Timed Wine Shop Example

Karnaugh Maps for Timed Circuit

Timed Circuit
Performance Analysis

- 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

Sample Properties

- The wine arrives before the patron:
  - Always(ack\_patron \Rightarrow ack\_wine)
- When the wine is requested, it eventually arrives:
  - req\_wine \Rightarrow Eventually(ack\_wine)

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.

Summary of Course Topics

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