

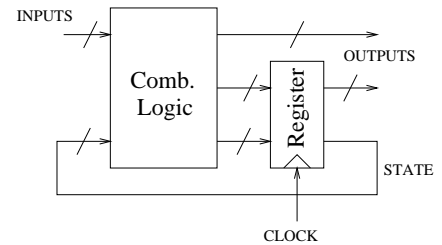
## Asynchronous Circuit Design

Chris J. Myers

Lecture 1: Introduction  
Preface and Chapter 1

## Synchronous Systems

- All events are synchronized to a single global clock.



## Synchronous Advantages

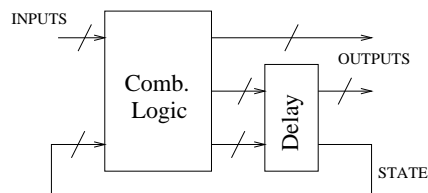
- Simple way to implement sequencing.
- Widely taught and understood.
- Available components.
- Simple way to deal with noise and *hazards*.

## 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

## Asynchronous Startups

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

## 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



## 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;
```

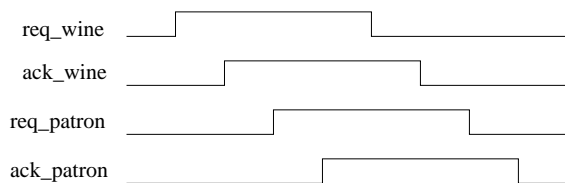
## Signal Protocol

```
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
end process;
```

## 2-Phase Protocol

```
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;
```

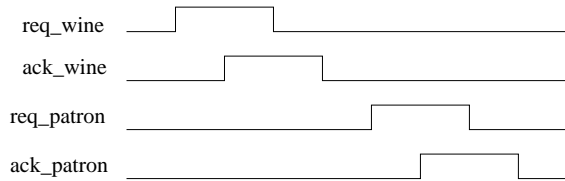
## Waveform for 2-Phase Protocol



## 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



## 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;
```

## 4-Phase Protocol: Passive/Passive

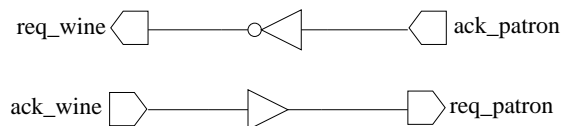
## Active/Active Protocol

```
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;
```

## Active/Active Reshuffled

```
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;
```

## Active/Active Circuit

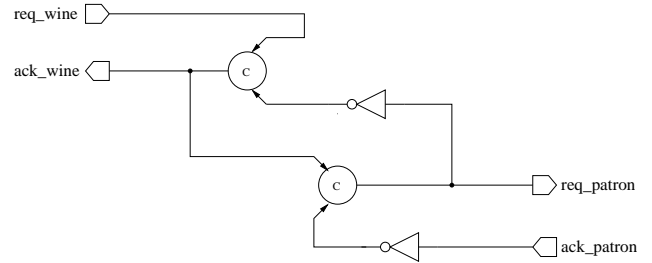


### Passive/Active Reshuffled

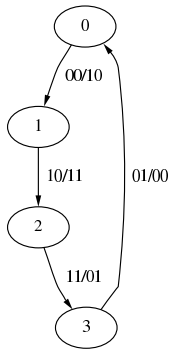
```

Shop_PA_reshuffled: 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;
  
```

### Passive/Active Circuit



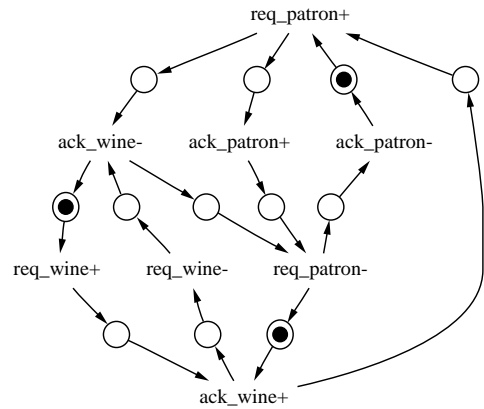
### AFSM and Huffman Flow Table (A/A reshuffled)



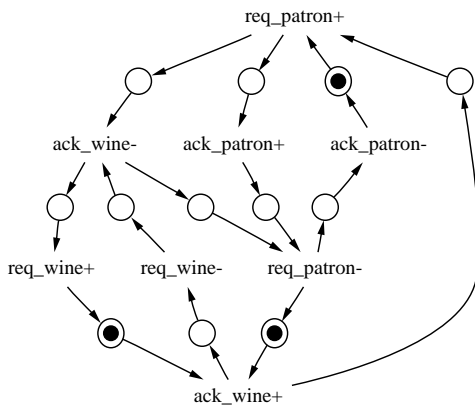
	ack_wine / ack_patron			
	00	01	11	10
0	1, 10	0, 00	-	-
1	1, 10	-	-	2, 11
2	-	-	3, 01	2, 11
3	-	0, 00	3, 01	-

req\_wine / req\_patron

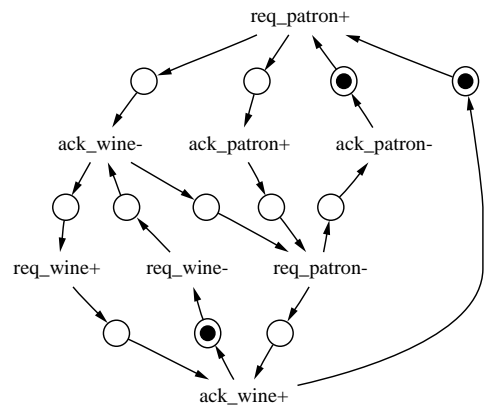
### Petri-net (P/A reshuffled)



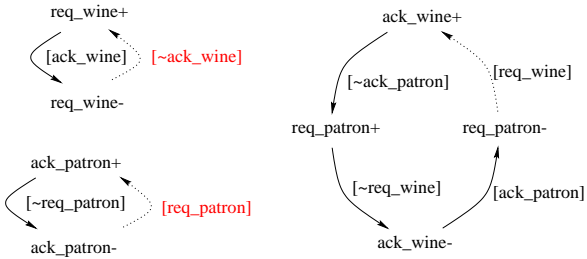
### Petri-net (P/A reshuffled)



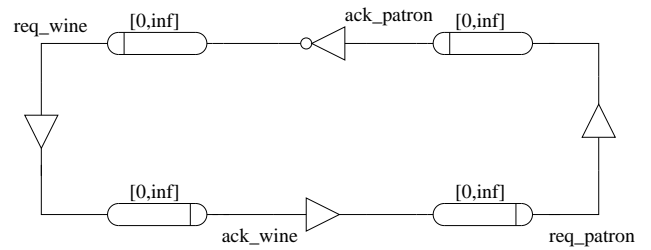
### Petri-net (P/A reshuffled)



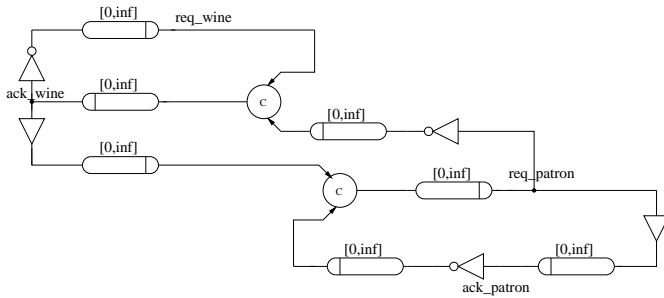
### TEL Structure (P/A reshuffled)



### A/A Reshuffled Circuit



### P/A Reshuffled Circuit



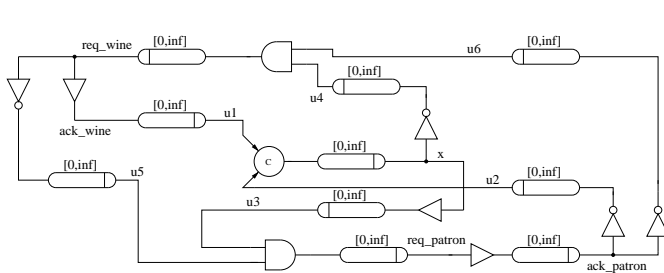
### Active/Active State Variable

Shop\_AA\_state\_variable:process  
begin

```

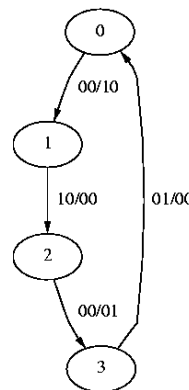
assign(req_wine,'1'); - call winery
guard(ack_wine,'1'); - wine arrives
assign(x,'1'); - set state variable
assign(req_wine,'0'); - reset req_wine
guard(ack_wine,'0'); - ack_wine resets
assign(req_patron,'1'); - call patron
guard(ack_patron,'1'); - patron buys wine
assign(x,'0'); - reset state variable
assign(req_patron,'0'); - reset req_patron
guard(ack_patron,'0'); - ack_patron resets
end process;
```

### A/A SV Circuit



req\_wine+, ack\_wine+, x+, req\_wine-, ack\_wine-, req\_patron+, ack\_patron+  
u2-, x-, u6-  
req\_wine glitches!

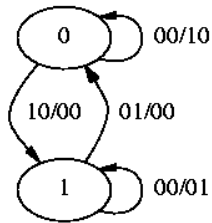
### AFSM and Huffman Flow Table (A/A SV)



		ack_wine / ack_patron			
		00	01	11	10
0	1, -0	① 00	-	-	-
1	① 10	-	-	-	2, -0
2	3, 0-	-	-	-	② 00
3	③ 01	0, 0-	-	-	-

req\_wine / req\_patron

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



	ack_wine / ack_patron			
	00	01	11	10
0	0, 10	0, 00	-	1, -0
1	1, 01	0, 0-	-	1, 00

	req_wine / req_patron			
	00	01	11	10
0	0, 10	0, 00	-	1, -0
1	1, 01	0, 0-	-	1, 00

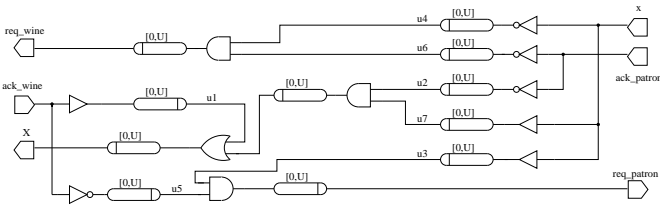
## Karnaugh Maps for Huffman's A/A SV Circuit

ack_wine/ack_patron					ack_wine/ack_patron				
x	00	01	11	10	x	00	01	11	10
0	1	0	-	-	0	0	0	-	0
1	0	0	-	0	1	1	-	-	0

req_wine					req_wine				
x	00	01	11	10	x	00	01	11	10
0	0	0	-	1	0	0	0	-	1
1	1	0	-	1	1	1	0	-	1

x

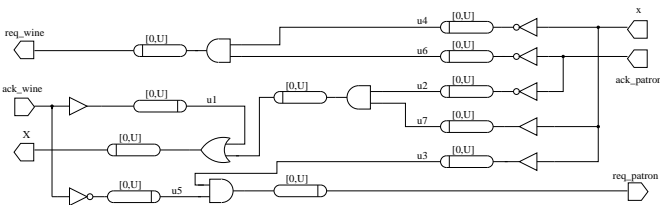
## Huffman's A/A SV Circuit



## Huffman's Assumptions

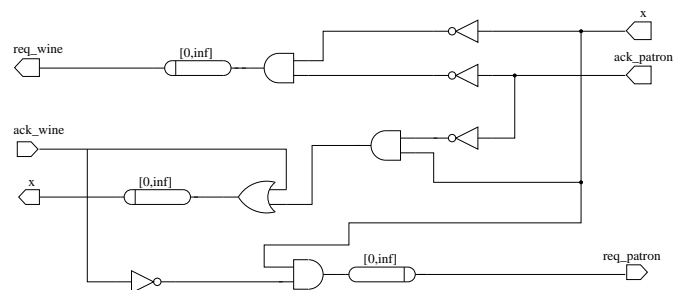
- Bounded gate and wire delay model.
- Circuit does not need to be closed.
- Single-input change fundamental mode.
- One input changes → output changes → state changes.
- May need to add delay in fed back state variables.

## Huffman's A/A SV Circuit



req\_wine+, ack\_wine+, X+, x+, req\_wine-, ack\_wine-, req\_patron+, ack\_patron+  
 x will not go low until after both u2- and u6- due to feedback delay assumption.

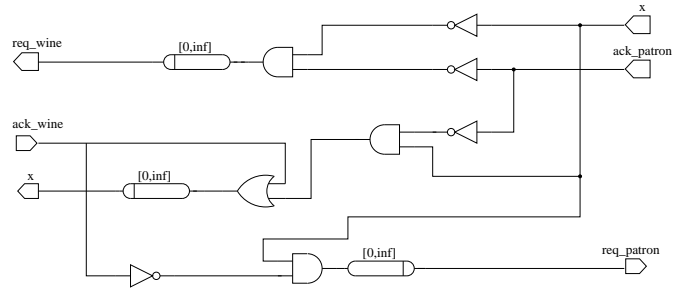
## Muller's Active/Active SV Circuit



## Muller's Assumptions

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

## Muller's Active/Active SV Circuit



*req\_wine+*, *ack\_wine+*, *x+*, *req\_wine-*, *ack\_wine-*, *req\_patron+*, *ack\_patron+*  
*ack\_patron-* change felt at both *x* and *req\_wine* gates simultaneously due to isochronic fork assumption.

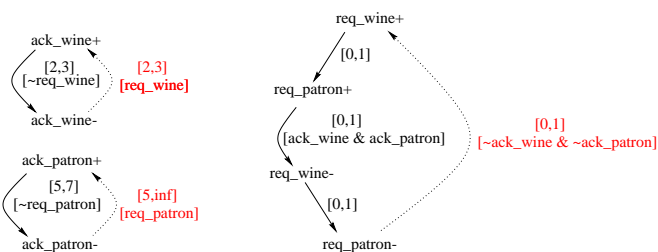
## Timed Wine Shop

```
Shop_AA_timed: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;
```

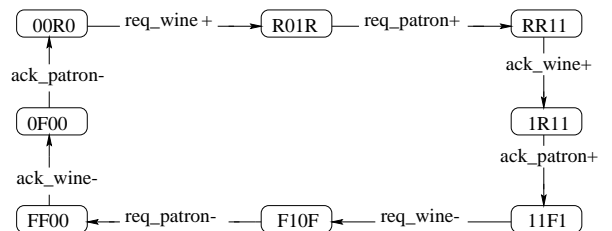
## Timed Winery and Patron

```
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;
```

## TEL Structure for Timed Wine Shop Example



## State Graph for Timed Wine Shop Example

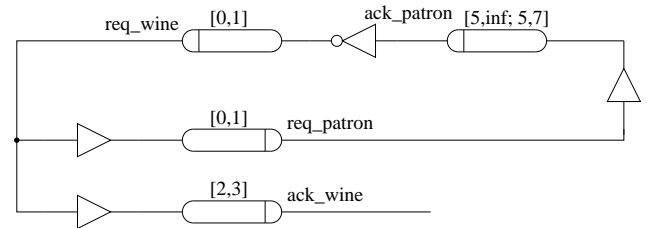


State vector: (*ack\_wine*, *ack\_patron*, *req\_wine*, *req\_patron*)

## Karnaugh Maps for Timed Circuit

		ack_wine/ack_patron				ack_wine/ack_patron				
		00	01	11	10	00	01	11	10	
req_wine/ req_patron	00	1	0	0	—	00	0	0	0	—
	01	—	—	0	—	01 <td>—</td> <td>—</td> <td>0</td> <td>—</td>	—	—	0	—
	11	1	—	0	1	11 <td>1</td> <td>—</td> <td>1</td> <td>1</td>	1	—	1	1
	10	1	—	—	—	10 <td>1</td> <td>—</td> <td>—</td> <td>—</td>	1	—	—	—
		req_wine				req_patron				

## 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

## 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:
  - $\text{Always}(\text{ack\_patron} \Rightarrow \text{ack\_wine})$
- When the wine is requested, it eventually arrives:
  - $\text{req\_wine} \Rightarrow \text{Eventually}(\text{ack\_wine})$

## Summary of Course Topics

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