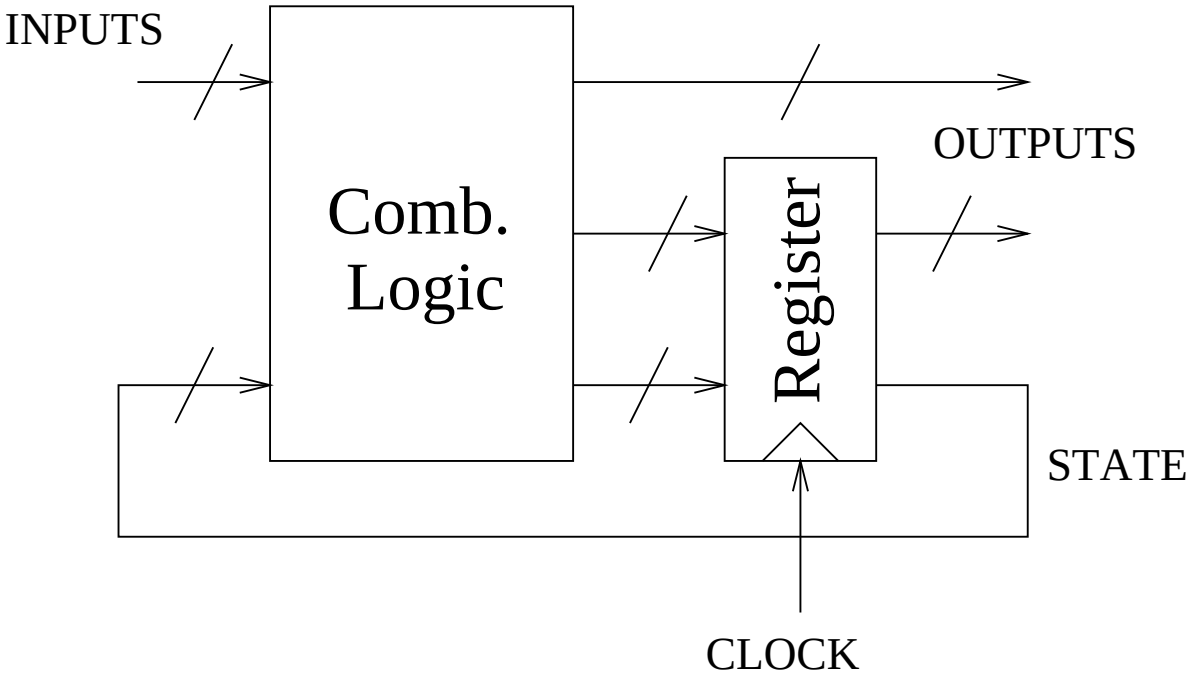
Asynchronous Circuit Design

Chris J. Myers

Lecture 1: Introduction
Preface and Chapter 1

Synchronous Systems

- All events are synchronized to a single global clock.



Synchronous Advantages

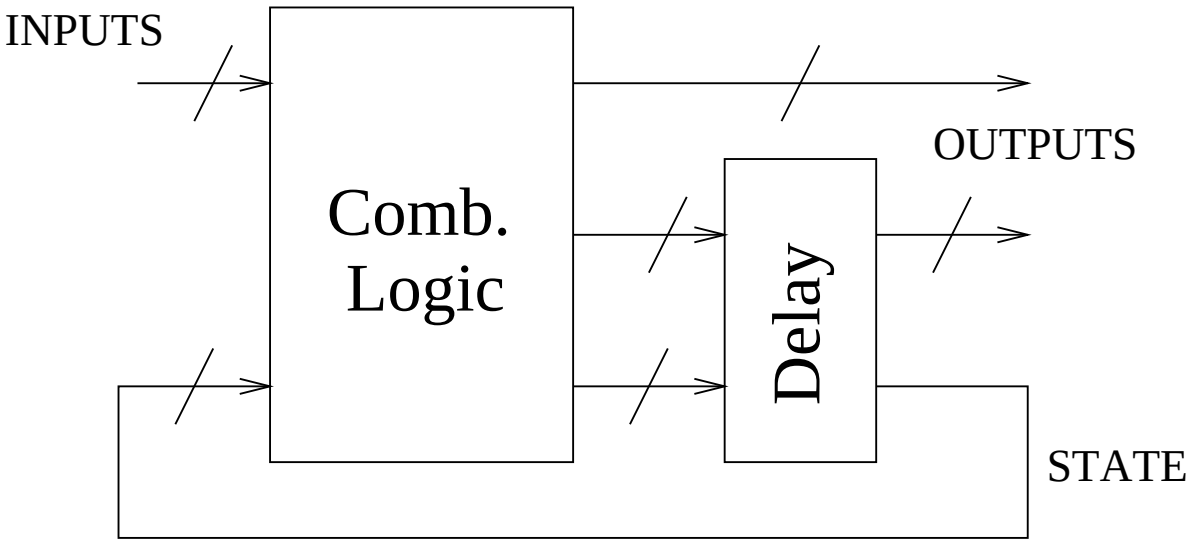
- Simple way to implement sequencing.
- Widely taught and understood.
- Available components.
- Simple way to deal with noise and *hazards*.

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 - Most Often Cited (Al Davis)

1.
2.
3.
4.
5.
6.
7.
8.
9.
10.

Asynchronous Advantages - Most Often Cited (Al Davis)

1.
2.
3.
4.
5.
6.
7.
8.
9.
10. Global synchrony doesn't exist anyway

Asynchronous Advantages - Most Often Cited (Al Davis)

1.
2.
3.
4.
5.
6.
7.
8.
9. Intrinsic elegance
10. Global synchrony doesn't exist anyway

Asynchronous Advantages - Most Often Cited (Al Davis)

1.
2.
3.
4.
5.
6.
7.
8. Intellectual challenge
9. Intrinsic elegance
10. Global synchrony doesn't exist anyway

Asynchronous Advantages - Most Often Cited (Al Davis)

1.
2.
3.
4.
5.
6.
7. Easier to exploit concurrency
8. Intellectual challenge
9. Intrinsic elegance
10. Global synchrony doesn't exist anyway

Asynchronous Advantages - Most Often Cited (Al Davis)

1.
2.
3.
4.
5.
6. Avoid clock distribution costs
7. Easier to exploit concurrency
8. Intellectual challenge
9. Intrinsic elegance
10. Global synchrony doesn't exist anyway

Asynchronous Advantages - Most Often Cited (Al Davis)

1.
2.
3.
4.
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

Asynchronous Advantages - Most Often Cited (Al Davis)

1.
2.
3.
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

Asynchronous Advantages - Most Often Cited (Al Davis)

1.
2.
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

Asynchronous Advantages - Most Often Cited (Al Davis)

1.
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

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

Asynchronous Advantages - NOT Often Cited (Al Davis)

1.
2.
3.
4.
5.
6.
7.
8.
9.
10.

Asynchronous Advantages - NOT Often Cited (Al Davis)

1.
2.
3.
4.
5.
6.
7.
8.
9.
10. It's none of your business

Asynchronous Advantages - NOT Often Cited (Al Davis)

1.
2.
3.
4.
5.
6.
7.
8.
9. Clock radiation causes hair loss
10. It's none of your business

Asynchronous Advantages - NOT Often Cited (Al Davis)

1.
2.
3.
4.
5.
6.
7.
8. Synchronous design gives me gas
9. Clock radiation causes hair loss
10. It's none of your business

Asynchronous Advantages - NOT Often Cited (Al Davis)

1.
2.
3.
4.
5.
6.
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 Advantages - NOT Often Cited (Al Davis)

1.
2.
3.
4.
5.
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 Advantages - NOT Often Cited (Al Davis)

1.
2.
3.
4.
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 Advantages - NOT Often Cited (Al Davis)

1.
2.
3.
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 Advantages - NOT Often Cited (Al Davis)

1.
2.
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 Advantages - NOT Often Cited (Al Davis)

1.
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 Advantages - NOT Often Cited (Al Davis)

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.

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)

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

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;
```

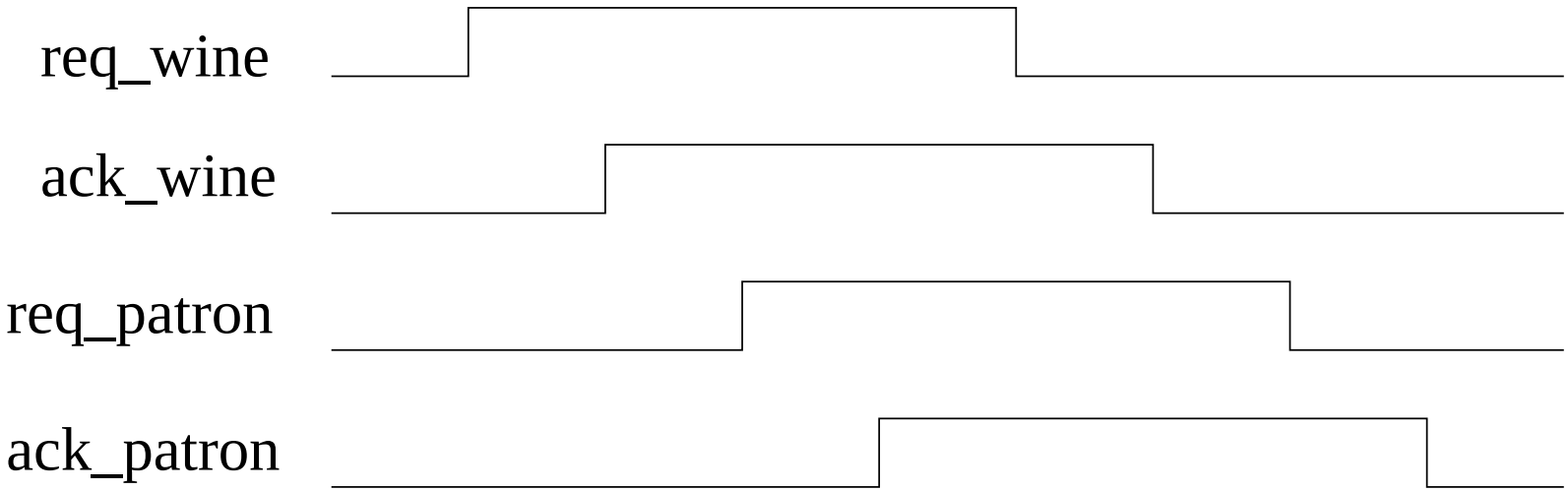
Signal Protocol

```
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;
```

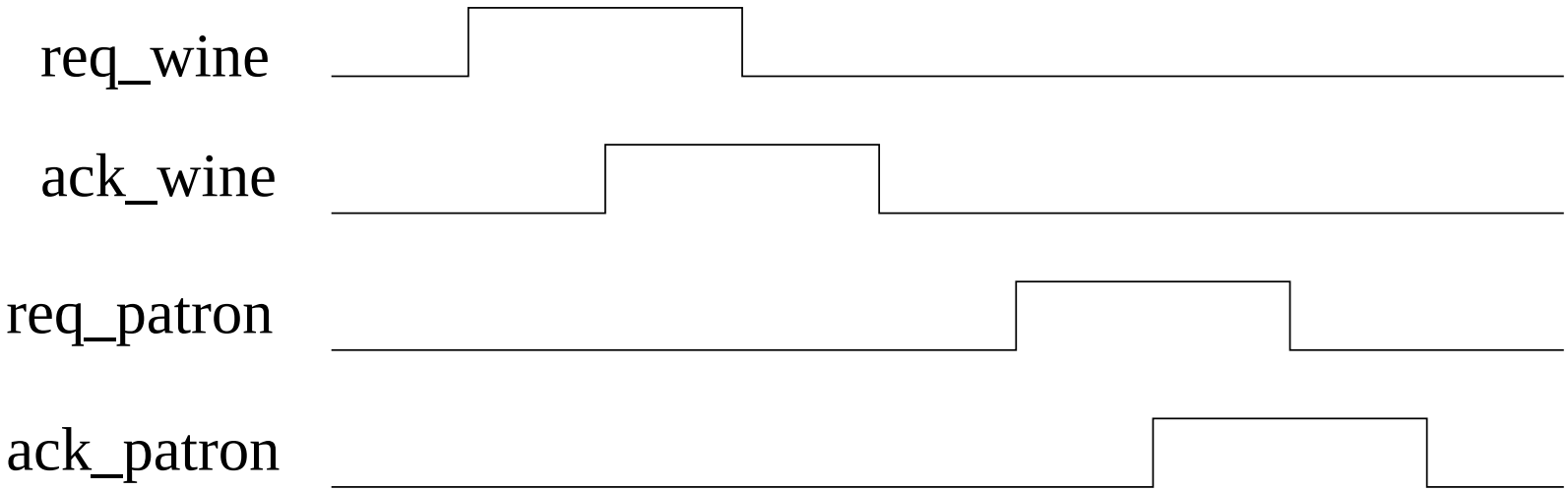
Waveform for 2-Phase Protocol



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

4-Phase Protocol: Passive/Passive

`Shop_PP`:**process**

4-Phase Protocol: Passive/Passive

```
Shop_PP:process
begin




end process;
```

4-Phase Protocol: Passive/Passive

```
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;
```

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

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

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



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;
```

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
  guard(req_patron,'0'); - req_patron resets

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
  guard(req_patron,'0'); - req_patron resets
  assign(ack_patron,'0'); - reset ack_patron
end process;
```

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 Protocol

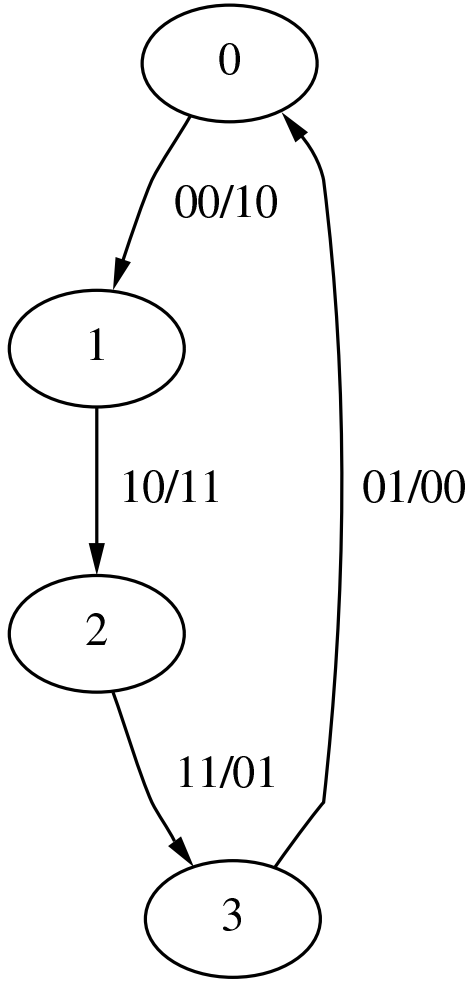
```
Shop_AA:process
begin
  assign(req_wine,'1'); - call winery
  guard(ack_wine,'1'); - wine arrives
  assign(req_wine,'0'); - reset req_wine
  guard(ack_wine,'0'); - ack_wine resets
⇒ state coding problem here
  assign(req_patron,'1'); - call patron
  guard(ack_patron,'1'); - patron buys wine
  assign(req_patron,'0'); - reset req_patron
  guard(ack_patron,'0'); - ack_patron resets
end process;
```

Active/Active Reshuffled

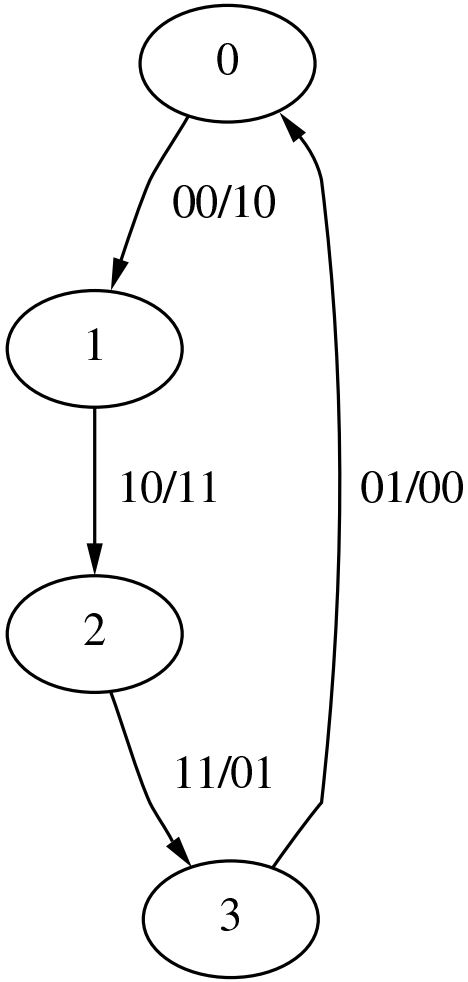
```
Shop_AA_reshuffled:process
begin
  assign(req_wine,'1'); - call winery
  guard(ack_wine,'1'); - wine arrives
  assign(req_patron,'1'); - call patron
  guard(ack_patron,'1'); - patron buys wine
  assign(req_wine,'0'); - reset req_wine
  guard(ack_wine,'0'); - ack_wine resets
  assign(req_patron,'0'); - reset req_patron
  guard(ack_patron,'0'); - ack_patron resets
end process;
```

Asynchronous Finite State Machine (AFSM) (A/A reshuffled)

```
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)

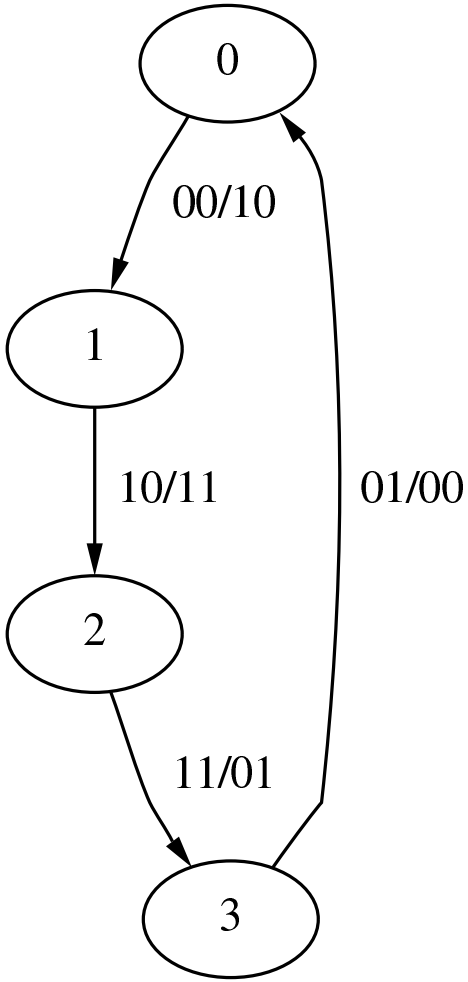


ack_wine / ack_patron

	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

AFSM and Huffman Flow Table (A/A reshuffled)

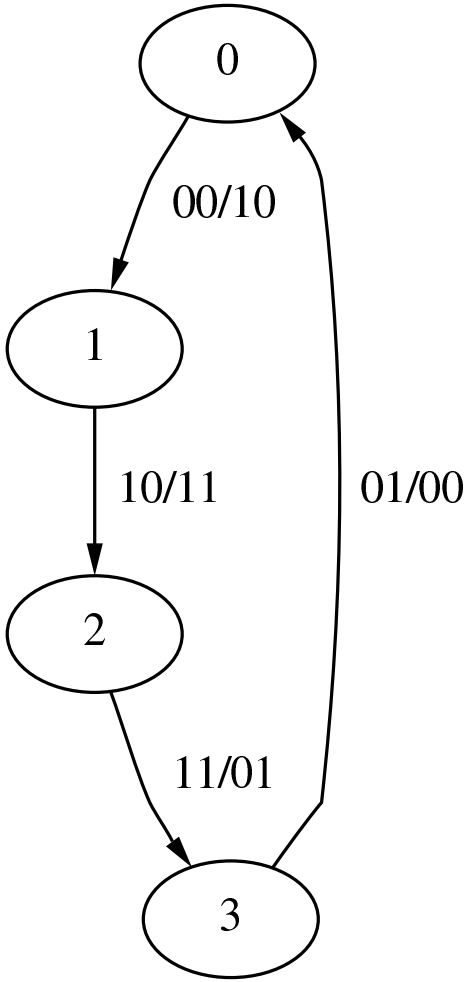


ack_wine / *ack_patron*

	00	01	11	10
0	⓪, 10	⓪, 00	–	2,11
2	–	–	3, 01	②, 11
3	–	0, 00	③, 01	–

req_wine / *req_patron*

AFSM and Huffman Flow Table (A/A reshuffled)

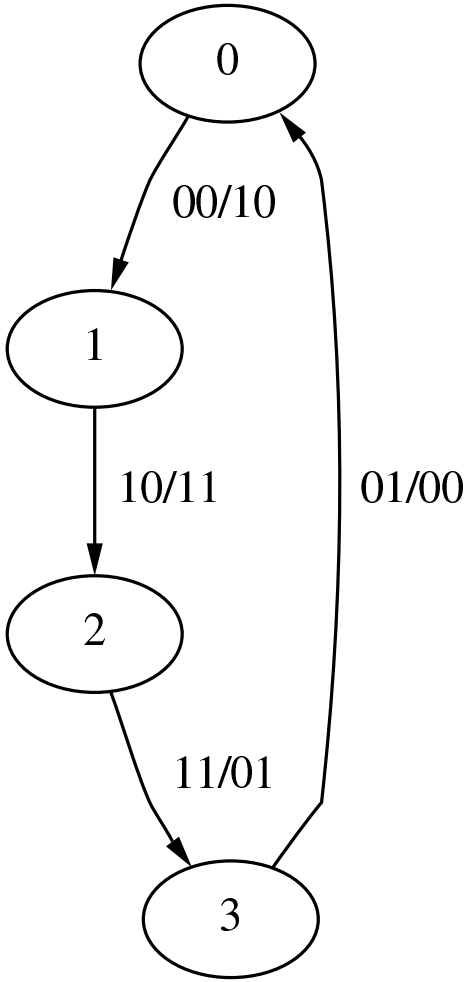


ack_wine / ack_patron

	00	01	11	10
0	⓪, 10	⓪, 00	3,01	⓪, 11
3	–	0, 00	③, 01	–

req_wine / req_patron

AFSM and Huffman Flow Table (A/A reshuffled)



ack_wine / ack_patron

	00	01	11	10
0	⓪, 10	⓪, 00	⓪, 01	⓪, 11

req_wine / req_patron

Karnaugh Maps for Huffman's A/A Reshuffled Circuit

ack_patron

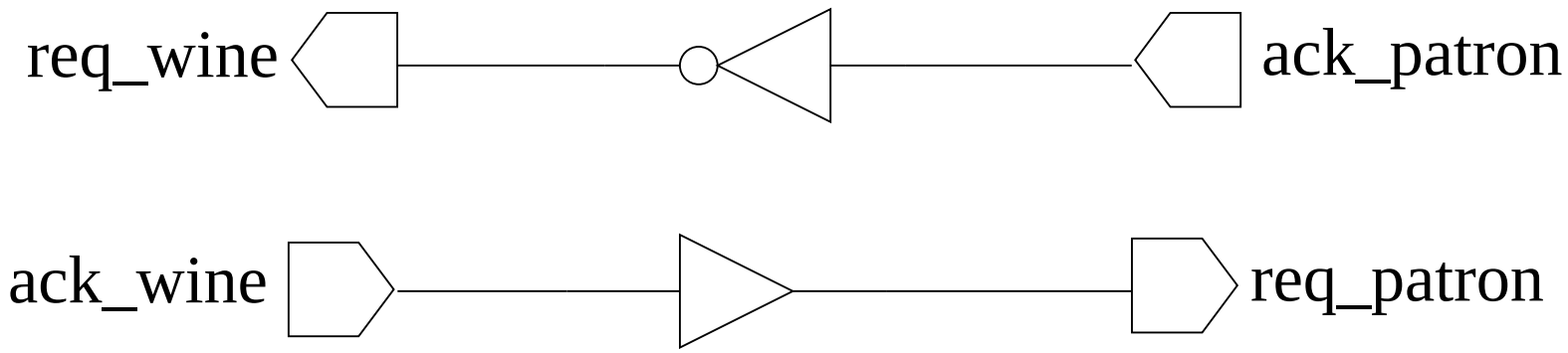
	ack_wine = 0	*ack_wine* = 1
0	1	1
1	0	0

req_wine

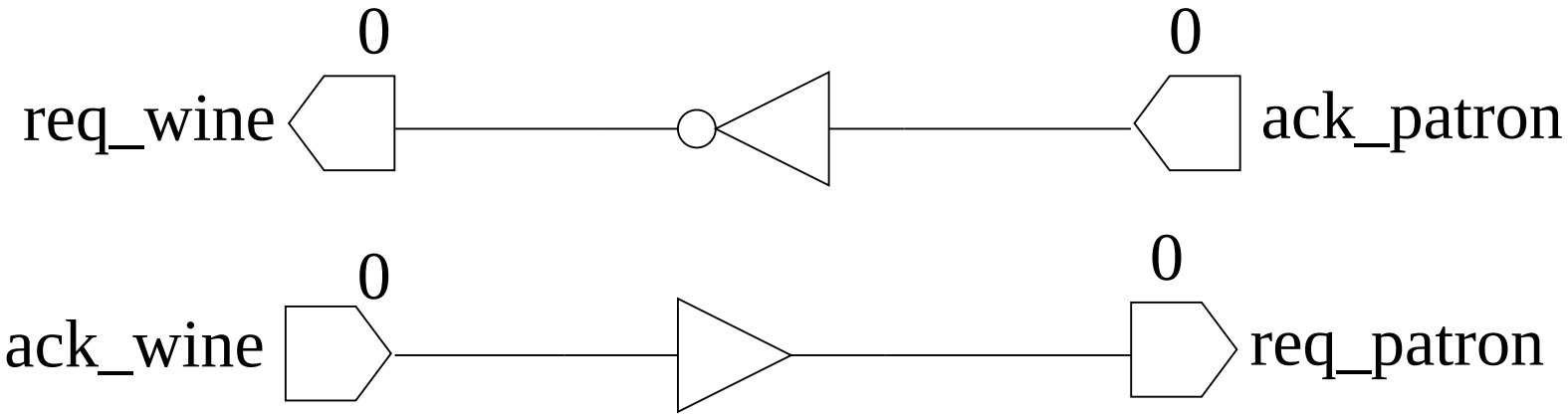
	ack_wine = 0	*ack_wine* = 1
0	0	1
1	0	1

req_patron

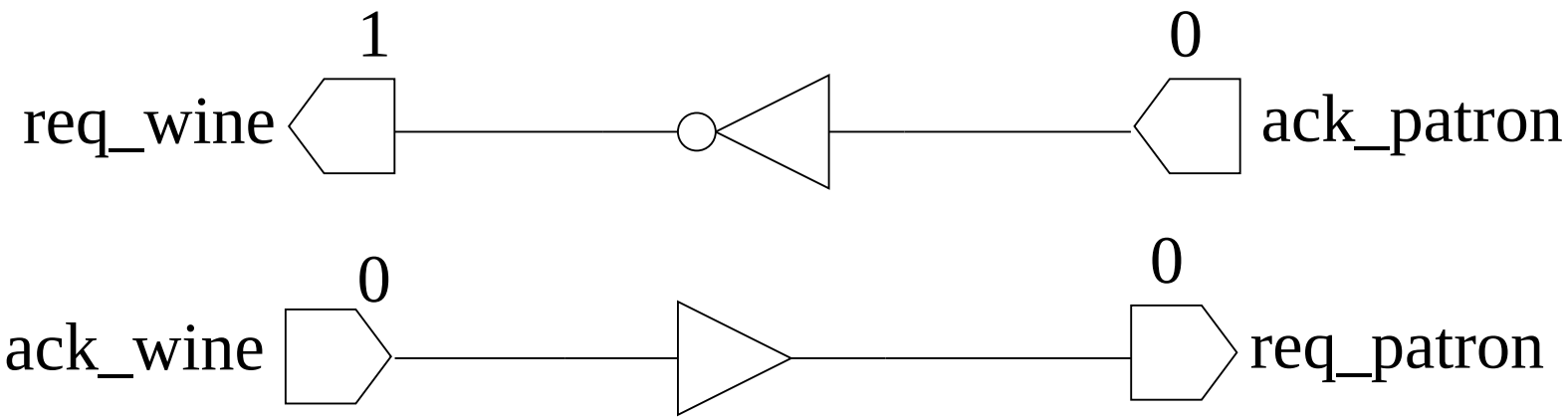
Active/Active Reshuffled Circuit



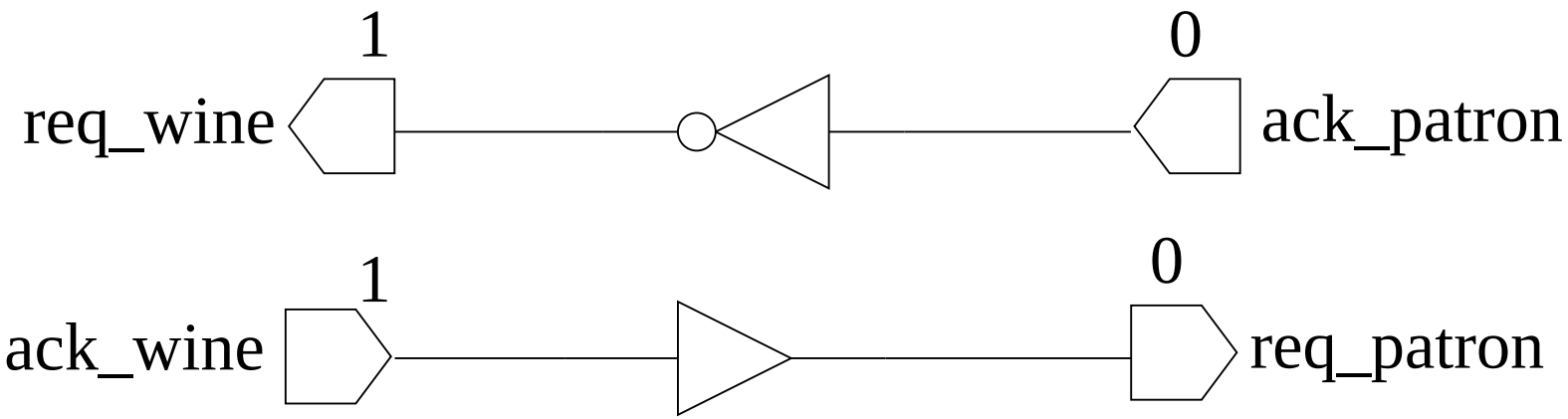
Active/Active Reshuffled Circuit



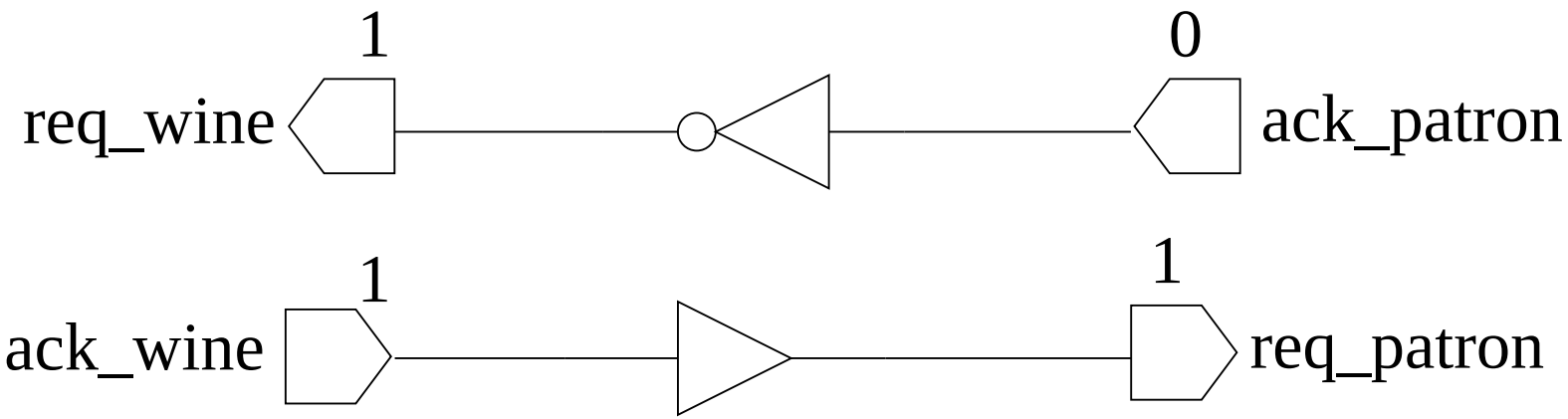
Active/Active Reshuffled Circuit



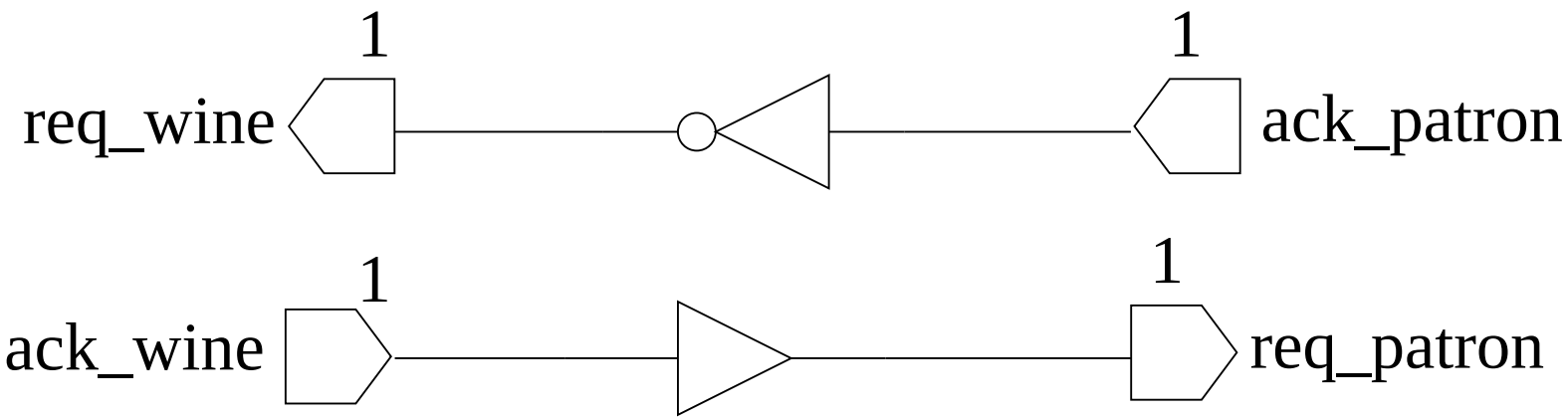
Active/Active Reshuffled Circuit



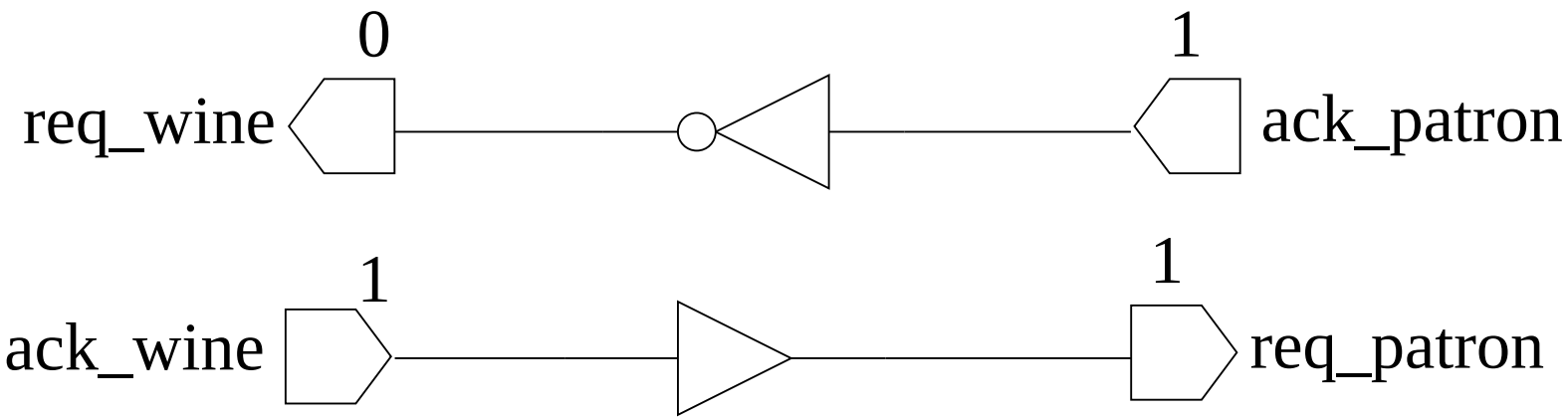
Active/Active Reshuffled Circuit



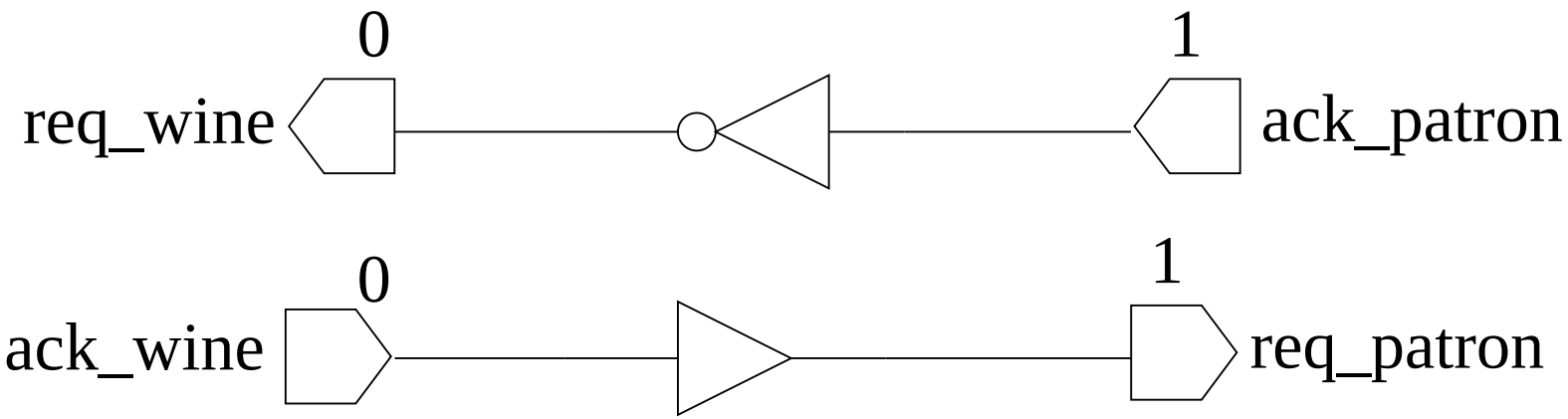
Active/Active Reshuffled Circuit



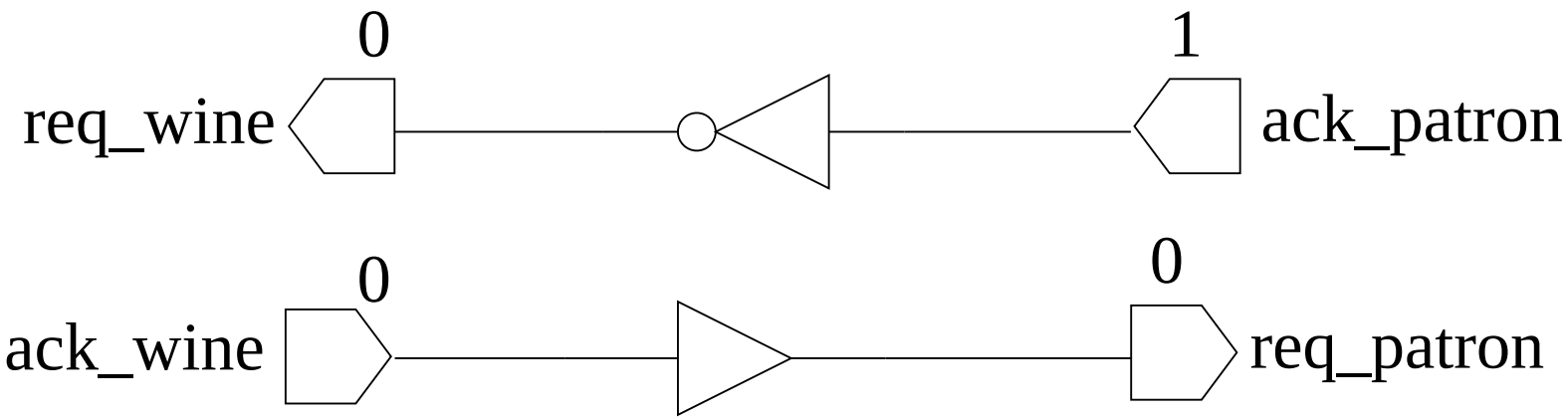
Active/Active Reshuffled Circuit



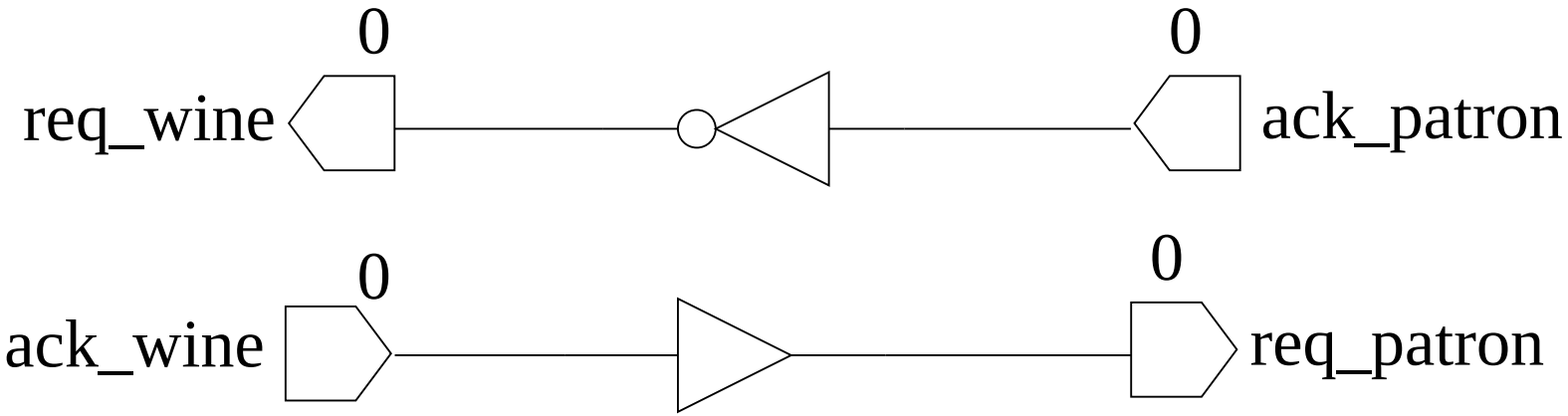
Active/Active Reshuffled Circuit



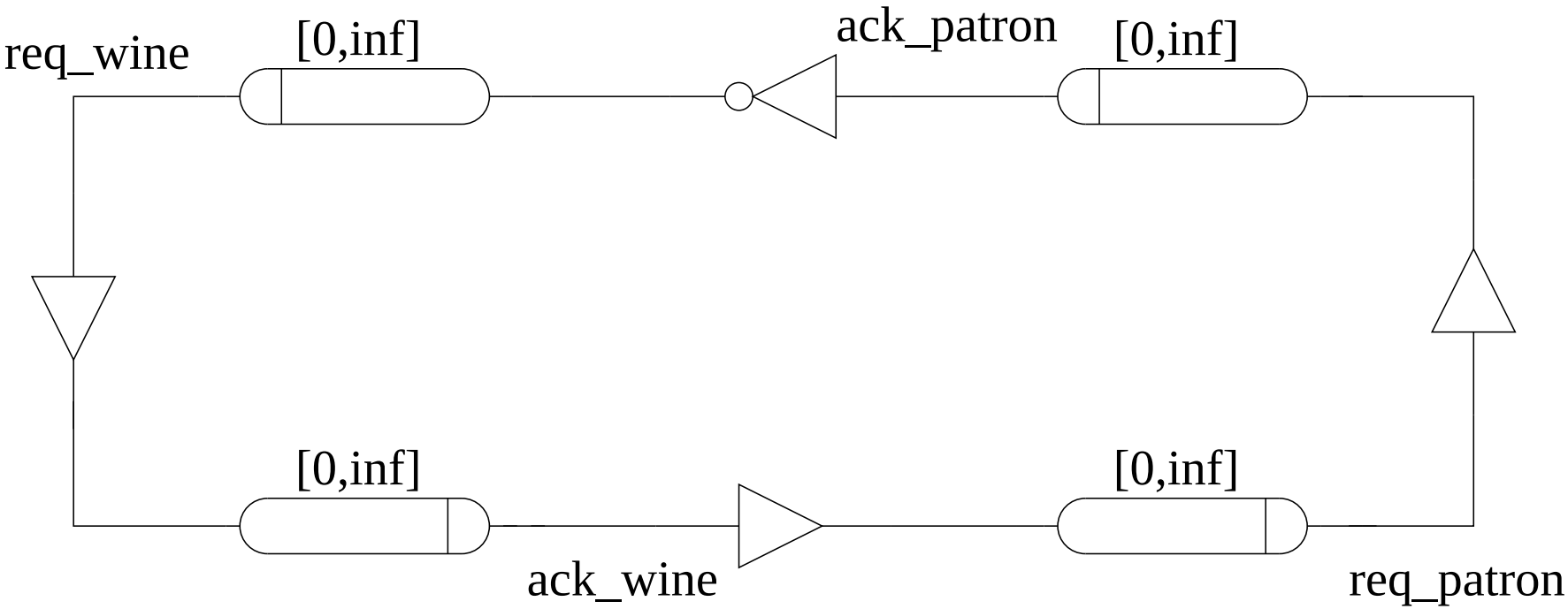
Active/Active Reshuffled Circuit



Active/Active Reshuffled Circuit



Active/Active Reshuffled Circuit



This circuit is *delay-insensitive*.

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;
```

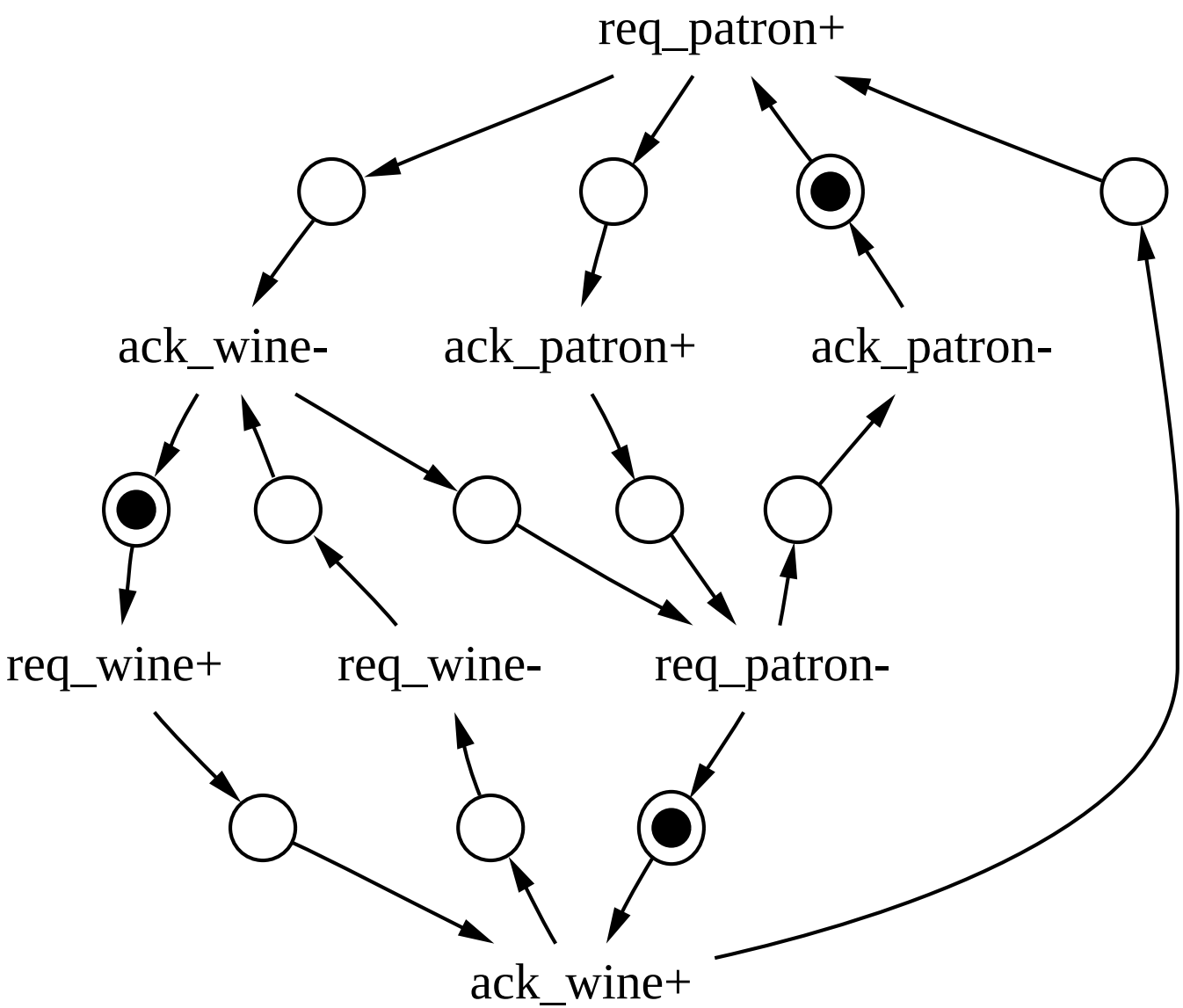
Passive/Active Reshuffled

```
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;
```

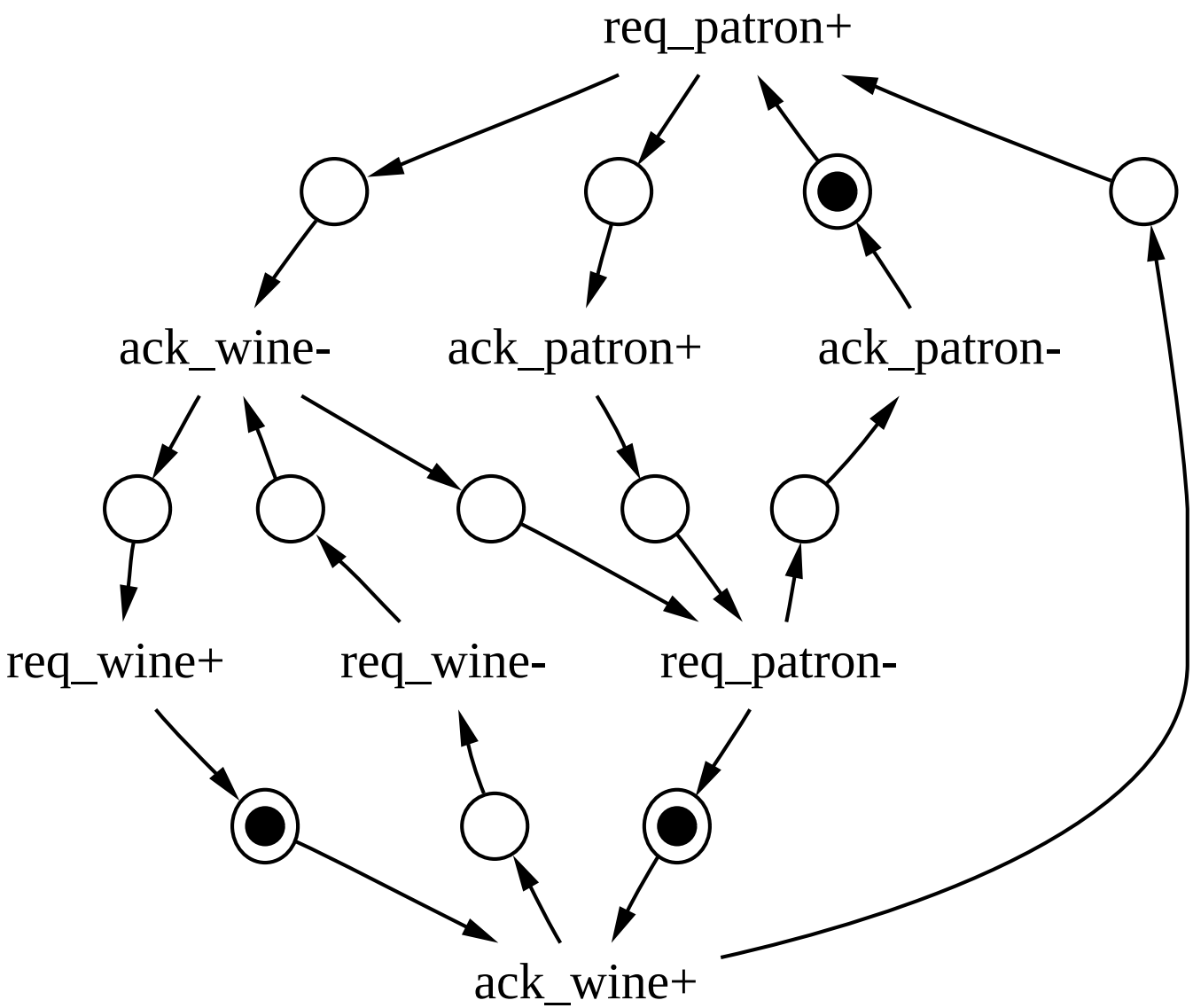
Passive/Lazy-Active Reshuffled

```
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;
```

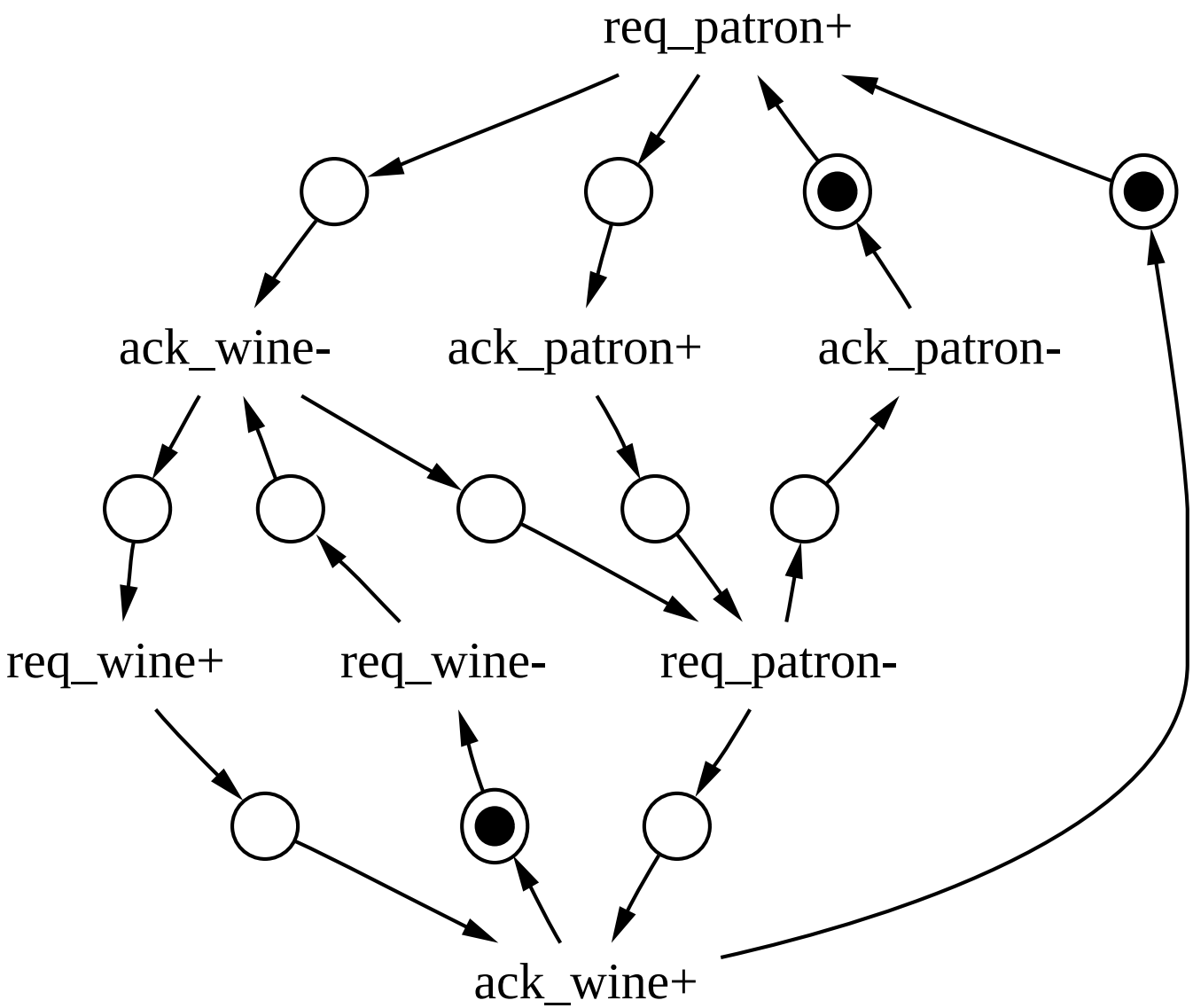
Petri-net (P/LA reshuffled)



Petri-net (P/LA reshuffled)

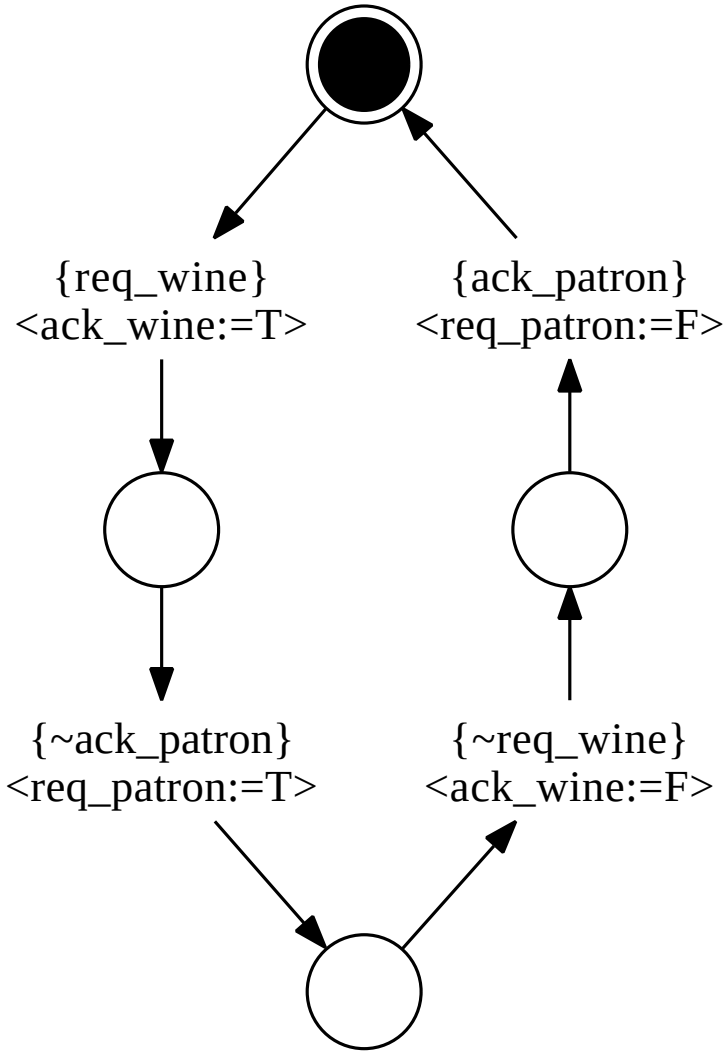


Petri-net (P/LA reshuffled)

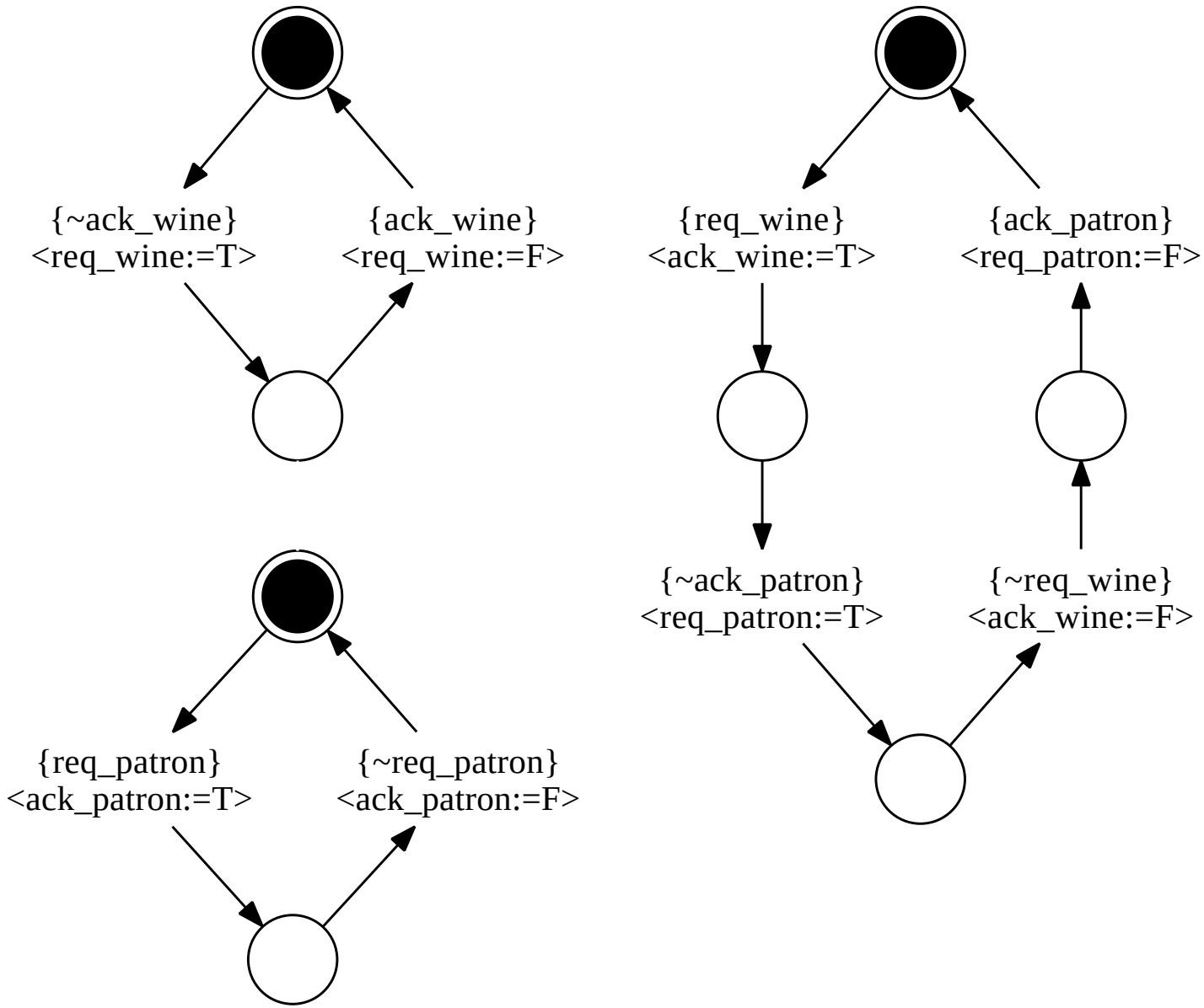


Labeled Petri Net (LPN) (P/LA reshuffled)

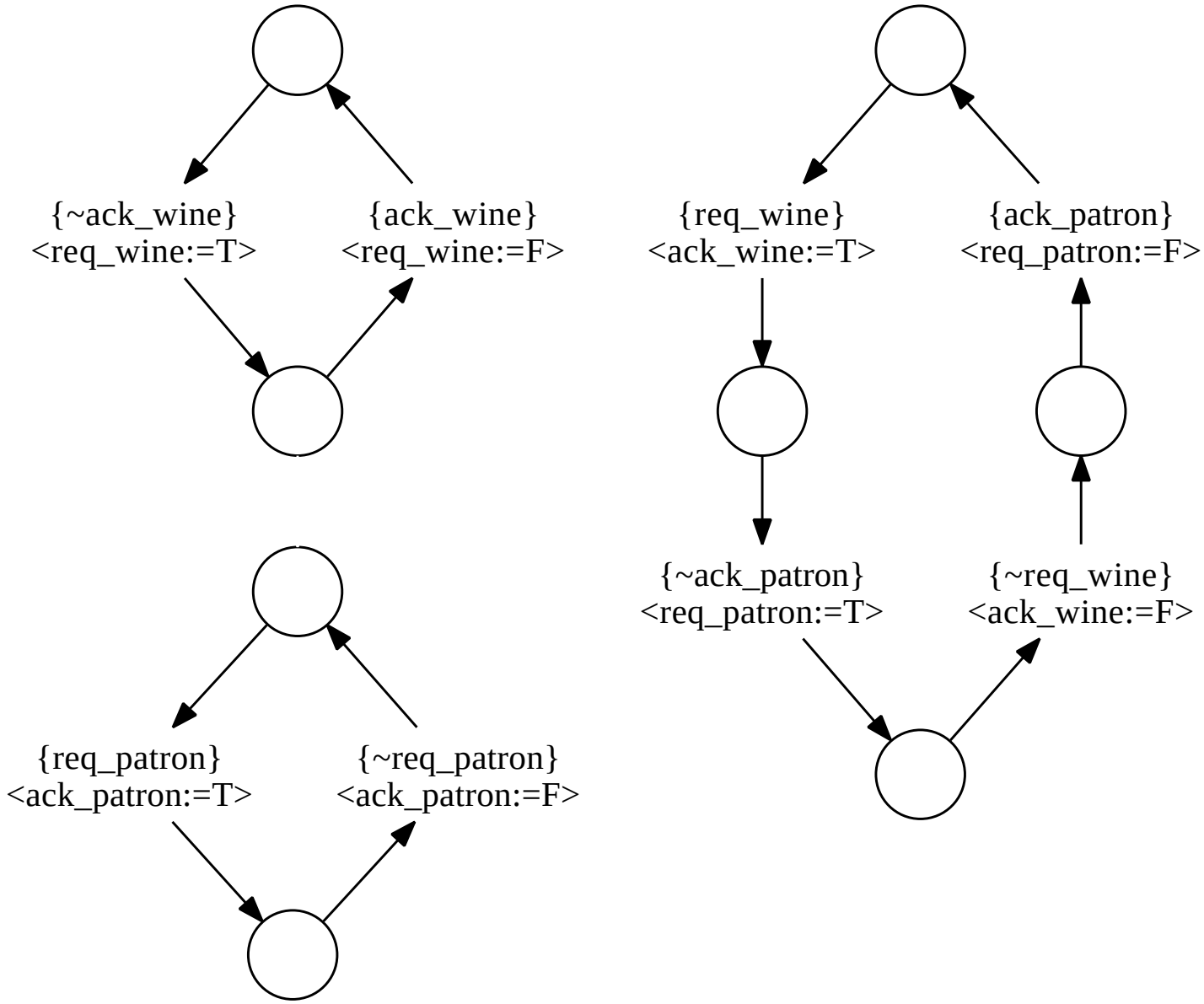
```
Shop_PA_lazy_active:process
begin
   guard(req_wine,'1');
   assign(ack_wine,'1');
   guard(ack_patron,'0');
   assign(req_patron,'1');
   guard(req_wine,'0');
   assign(ack_wine,'0');
   guard(ack_patron,'1');
   assign(req_patron,'0');
end process;
```



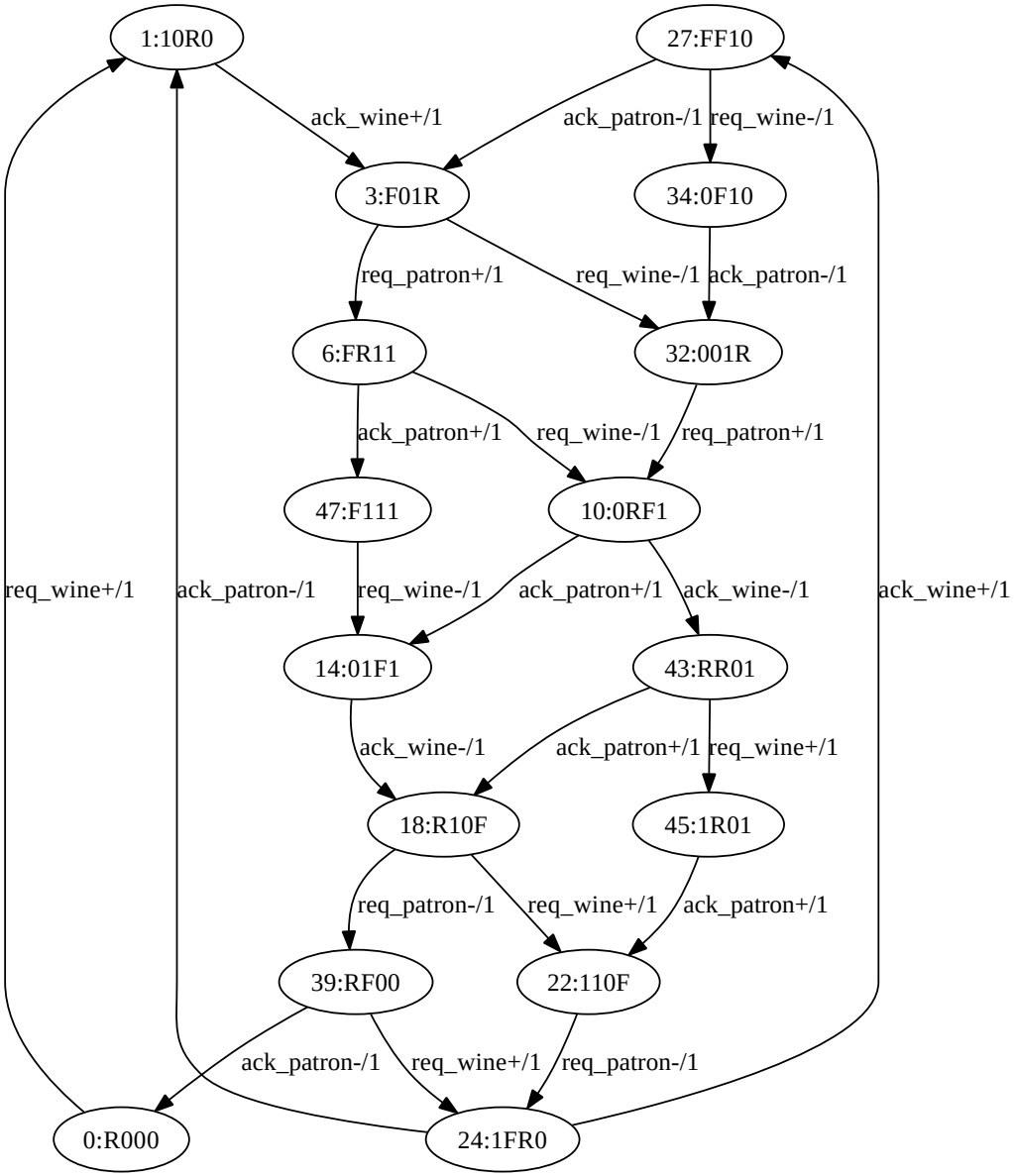
LPN (P/LA reshuffled)



From LPN to State Graph to Circuit



State Graph for P/LA Reshuffled (⟨ rw, ap, aw, rp ⟩)



Karnaugh Maps for Passive/Lazy-Active Reshuffled

req_wine/ack_patron

ack_wine/ req_patron	00	01	11	10
00	0	0	1	1
01	0	0	0	0
11	0	0	1	1
10	1	1	1	1

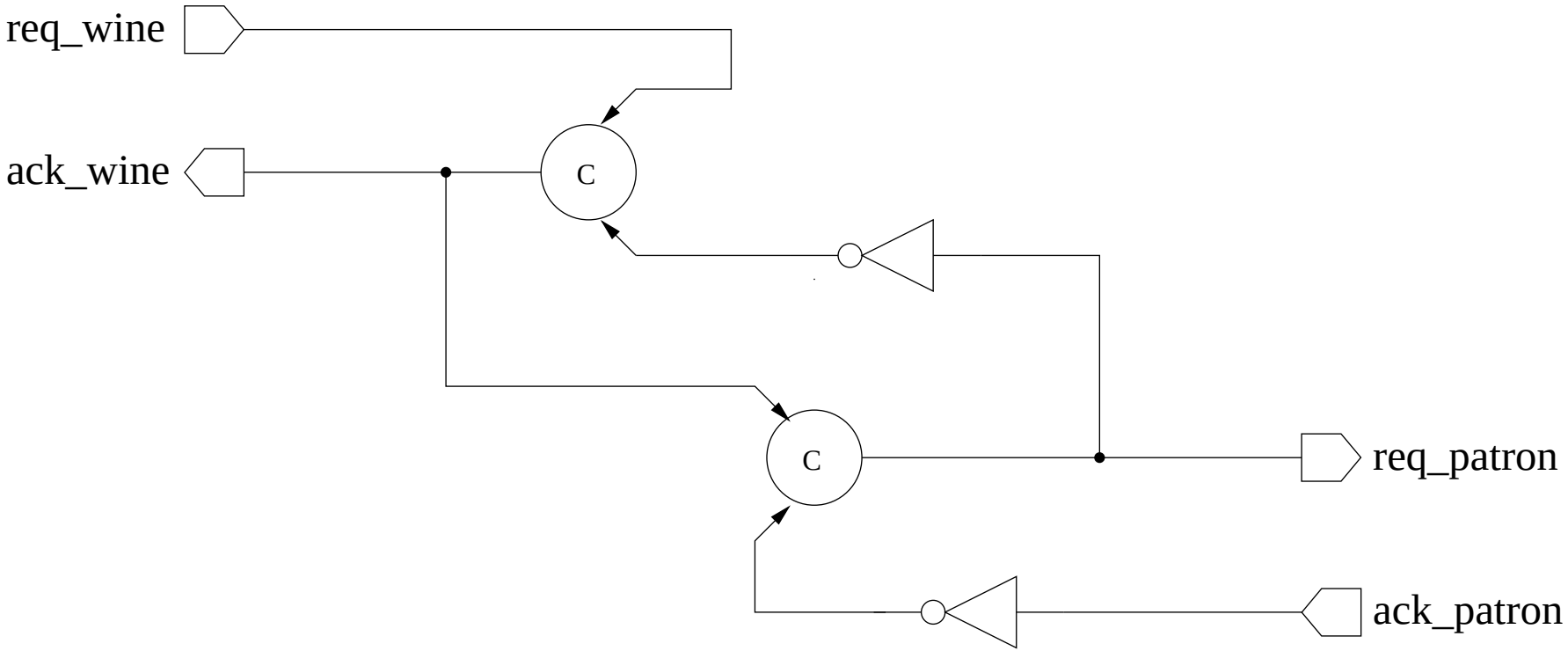
ack_wine

req_wine/ack_patron

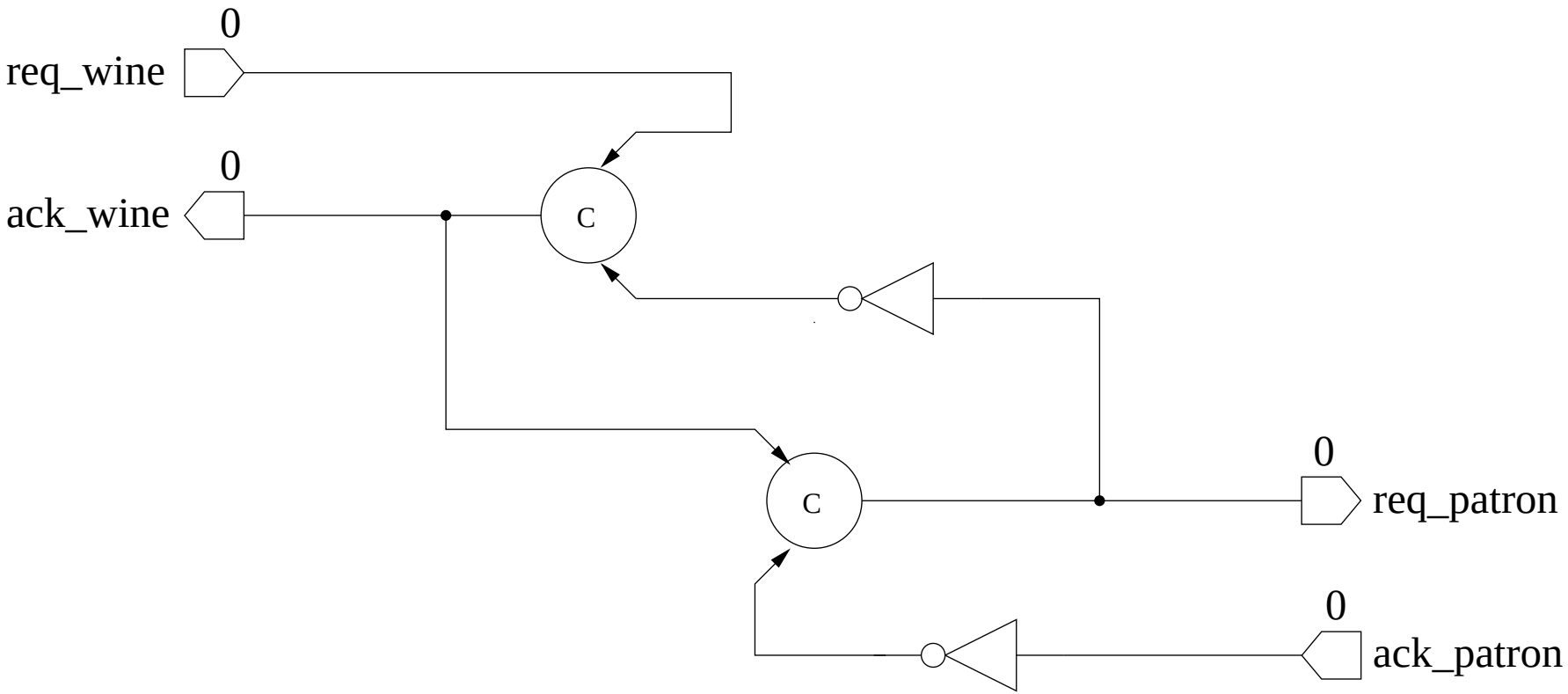
	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	1	1	1	1
10	1	0	0	1

req_patron

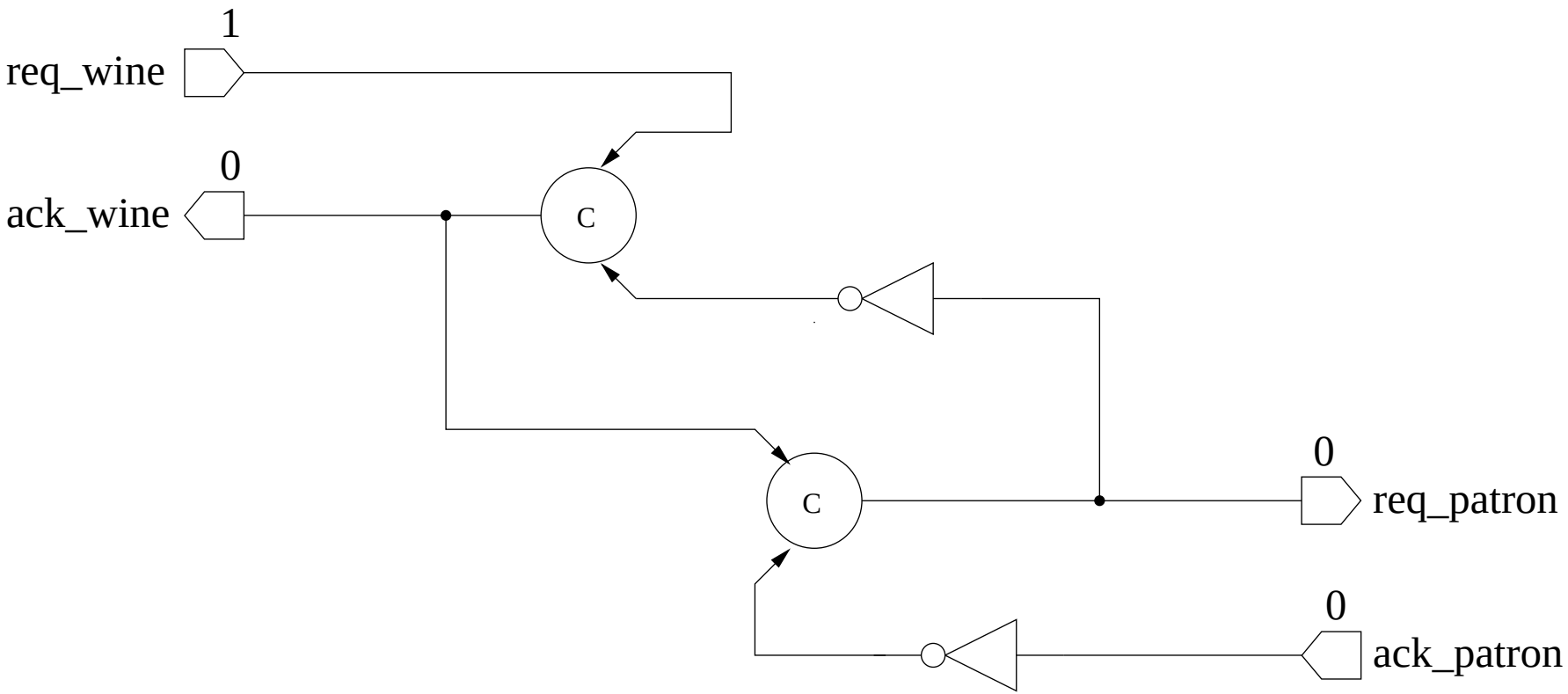
Passive/Lazy-Active Reshuffled Circuit



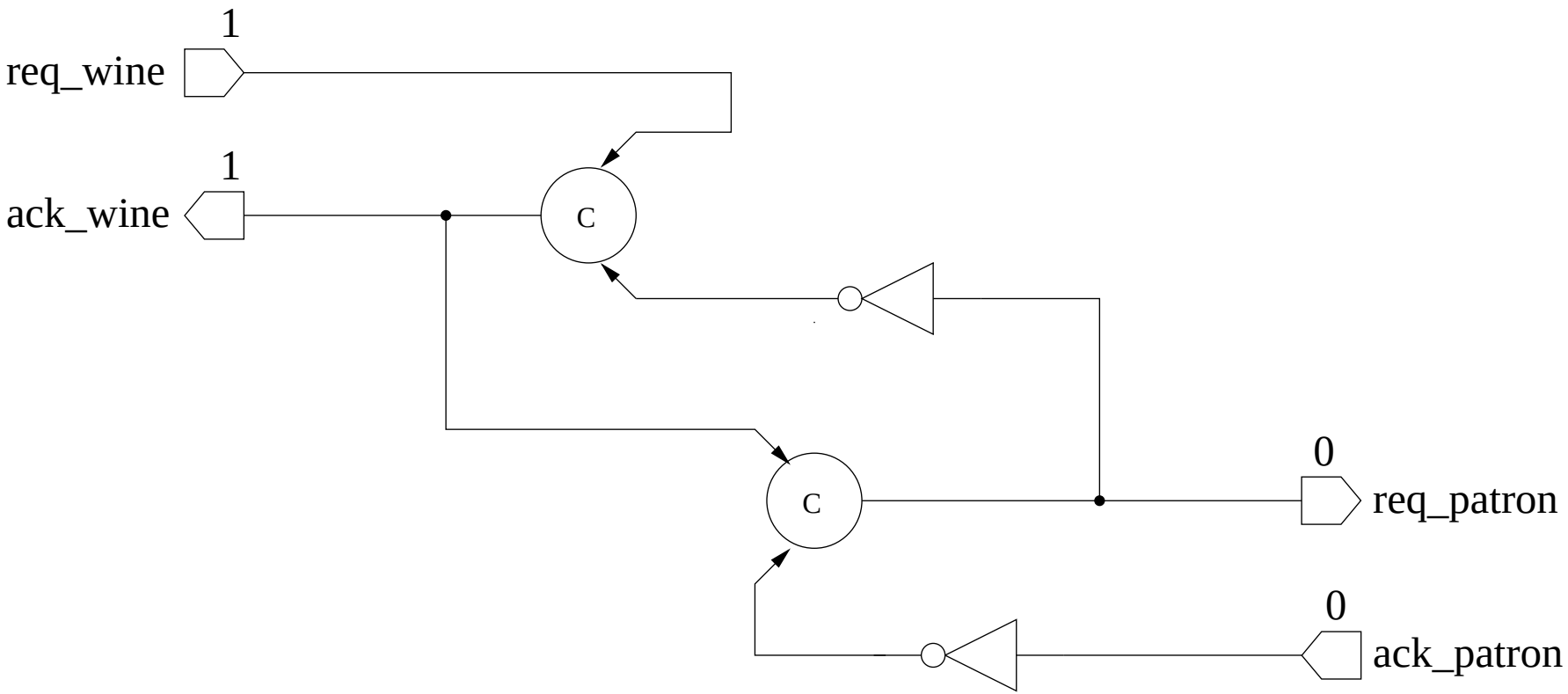
Passive/Lazy-Active Reshuffled Circuit



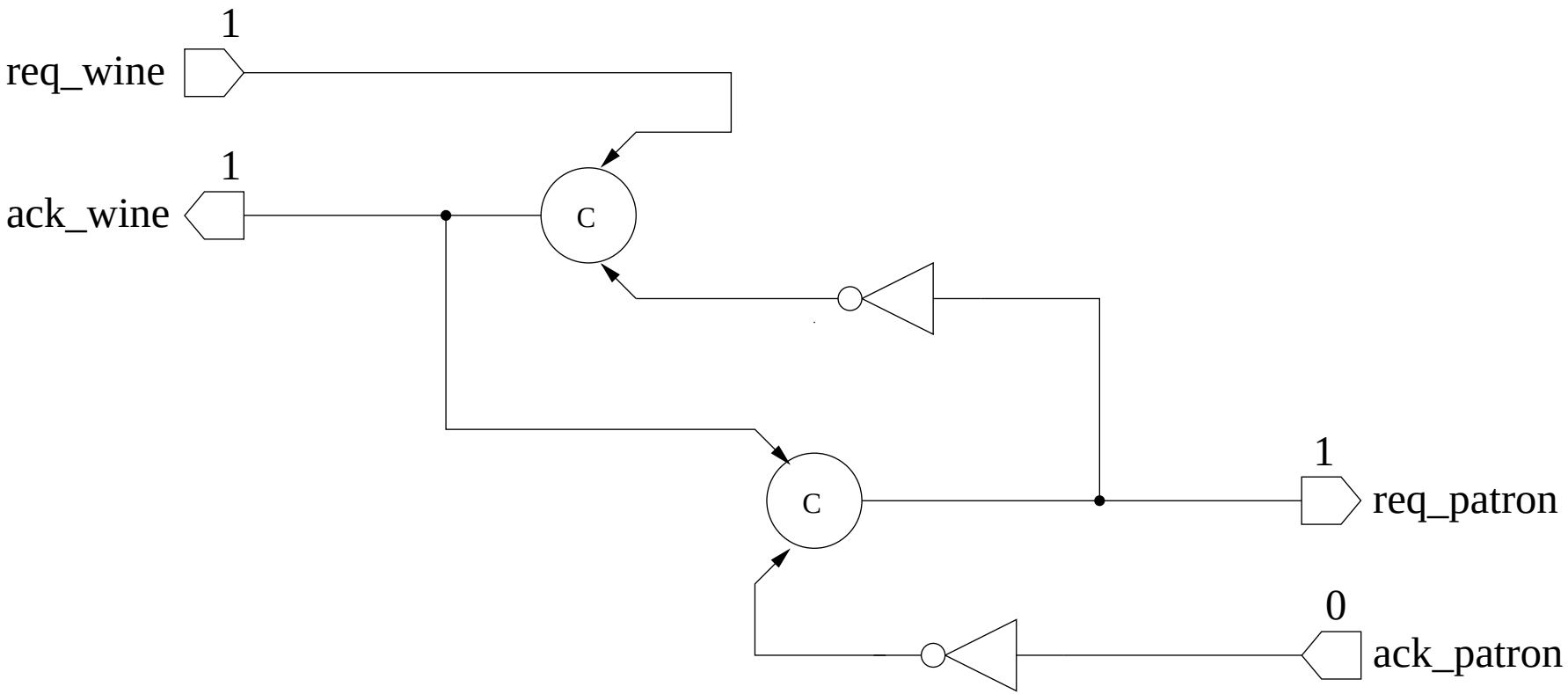
Passive/Lazy-Active Reshuffled Circuit



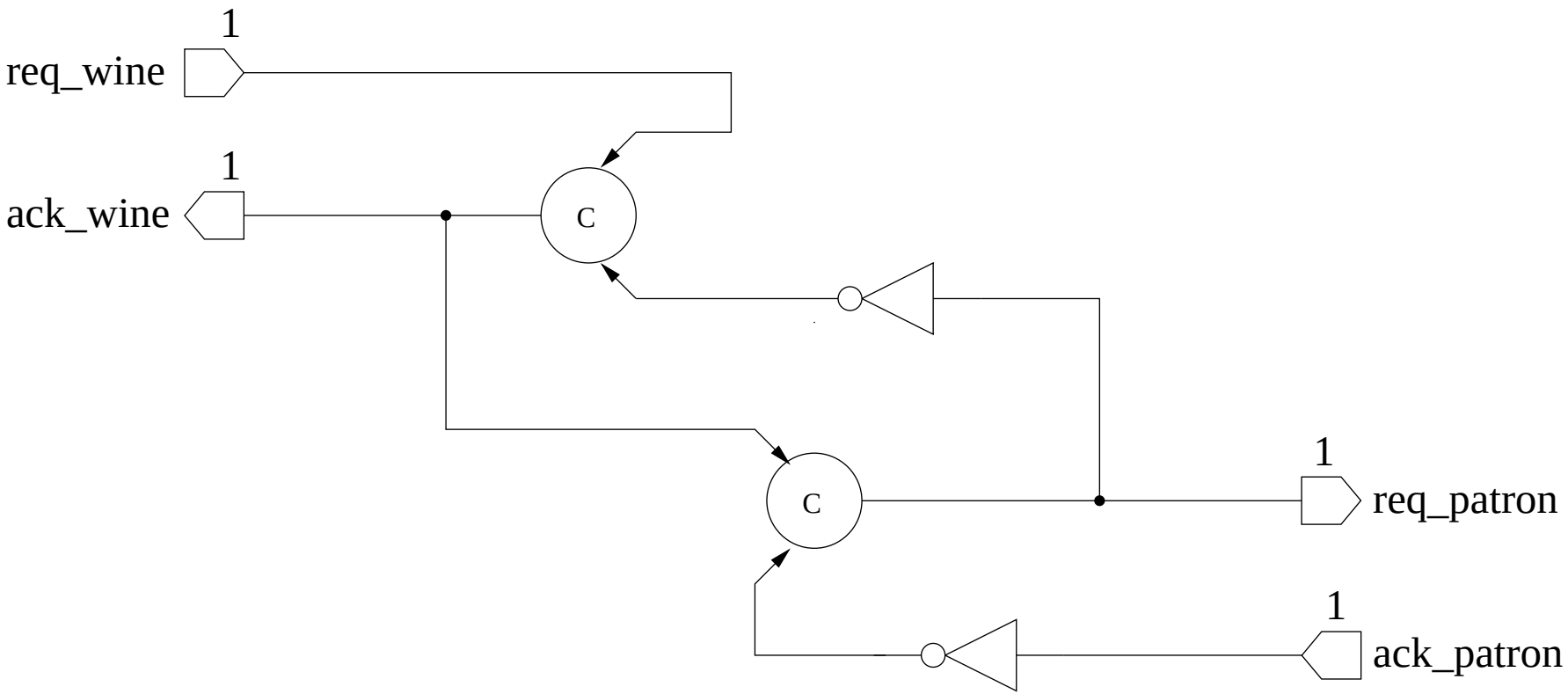
Passive/Lazy-Active Reshuffled Circuit



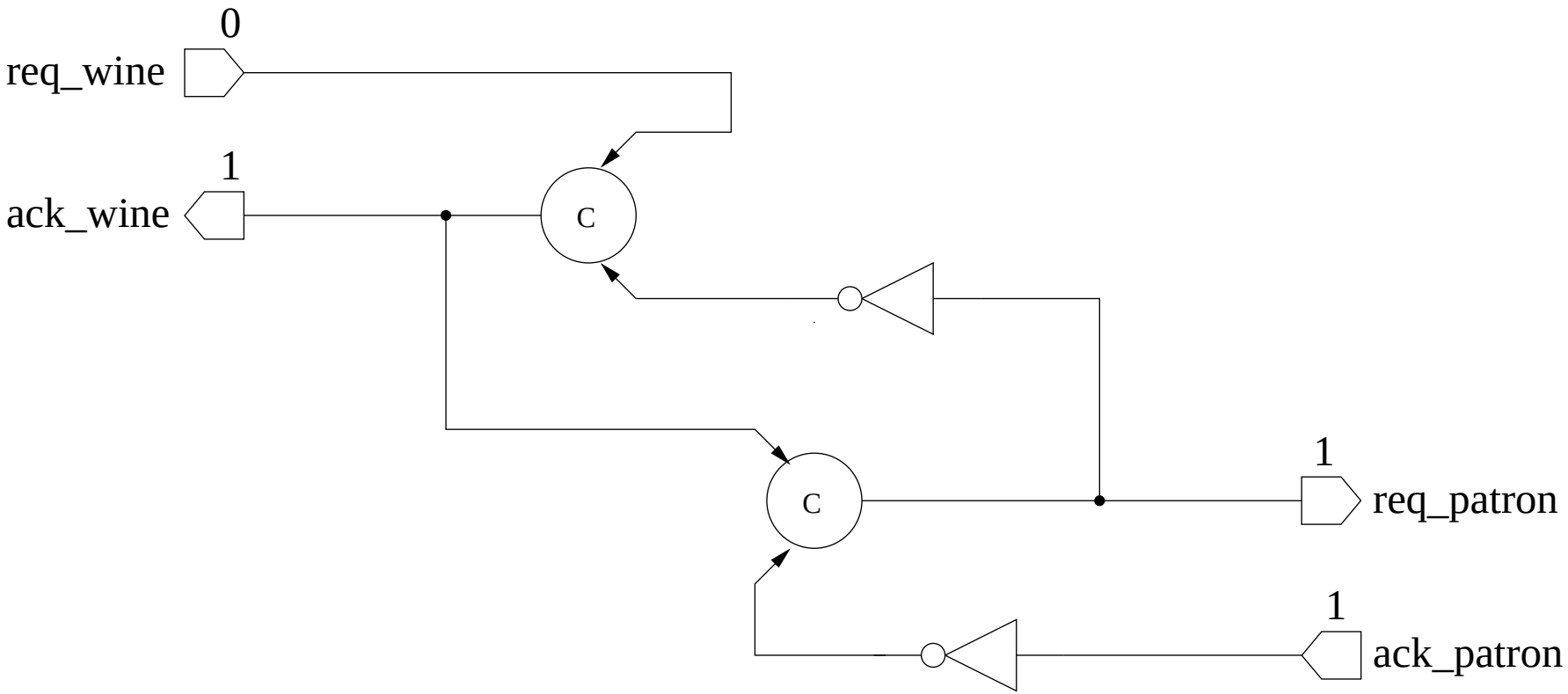
Passive/Lazy-Active Reshuffled Circuit



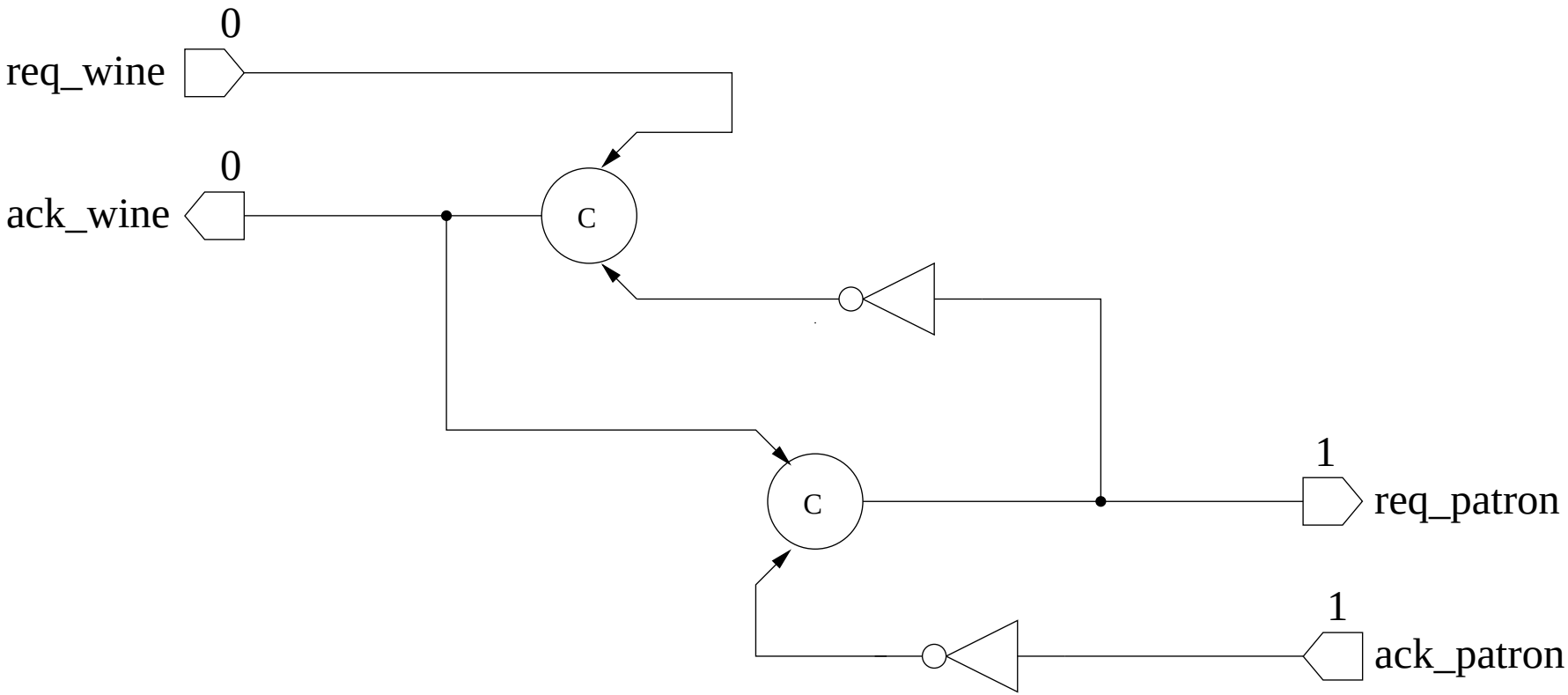
Passive/Lazy-Active Reshuffled Circuit



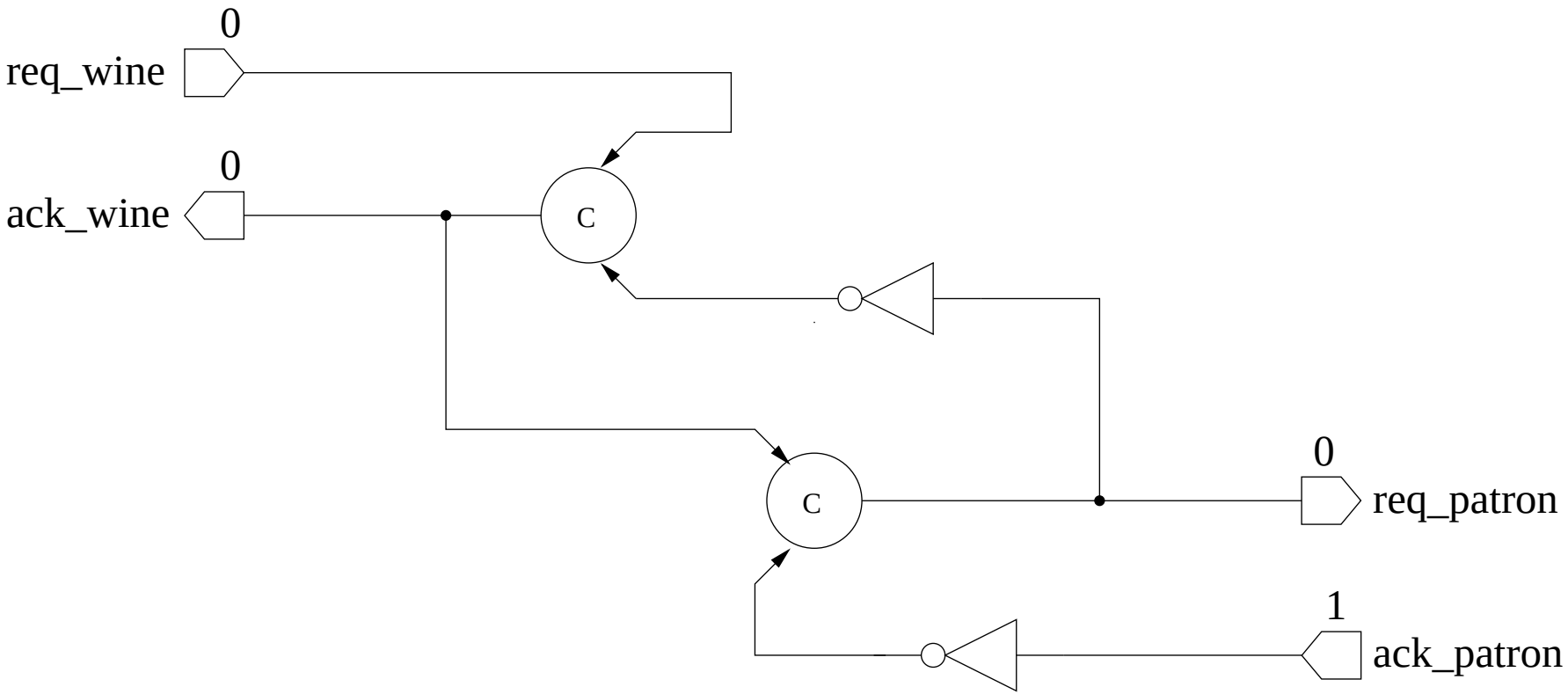
Passive/Lazy-Active Reshuffled Circuit



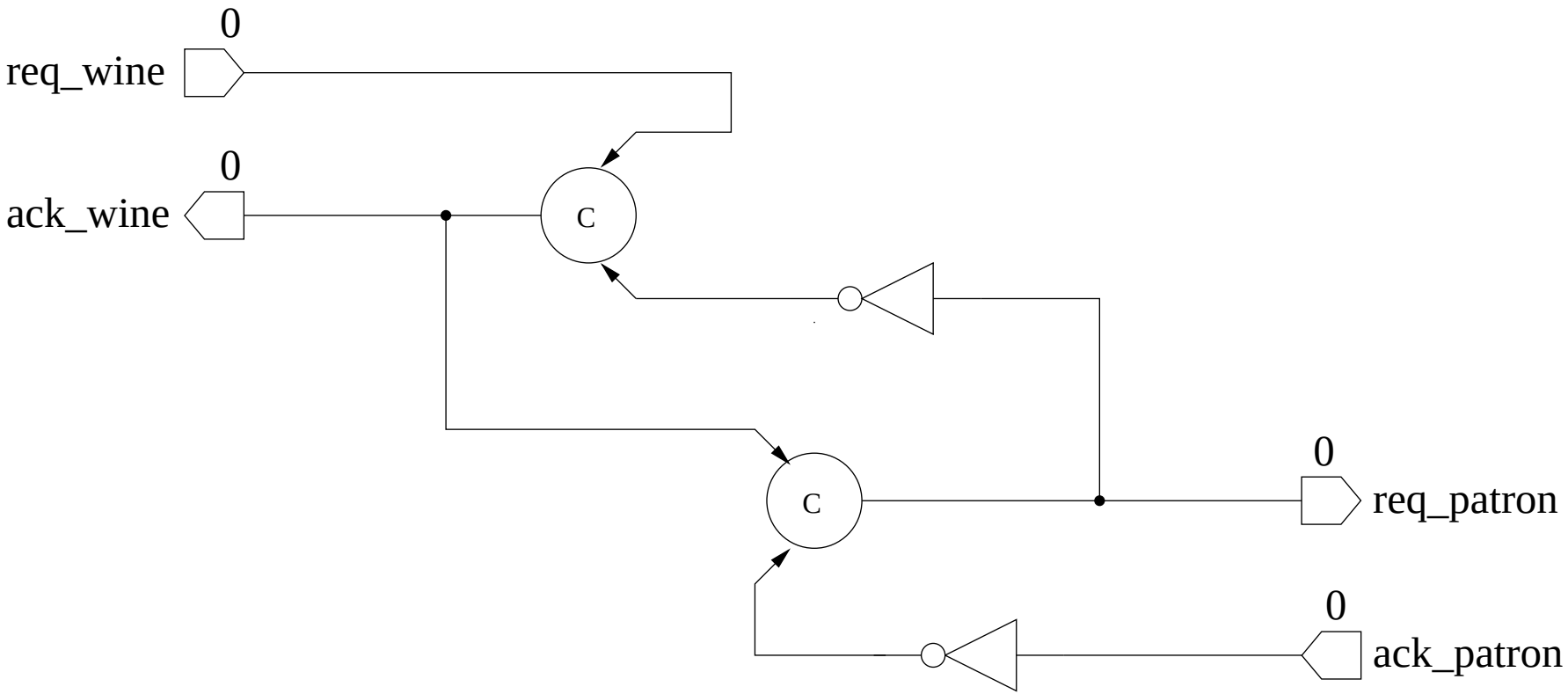
Passive/Lazy-Active Reshuffled Circuit



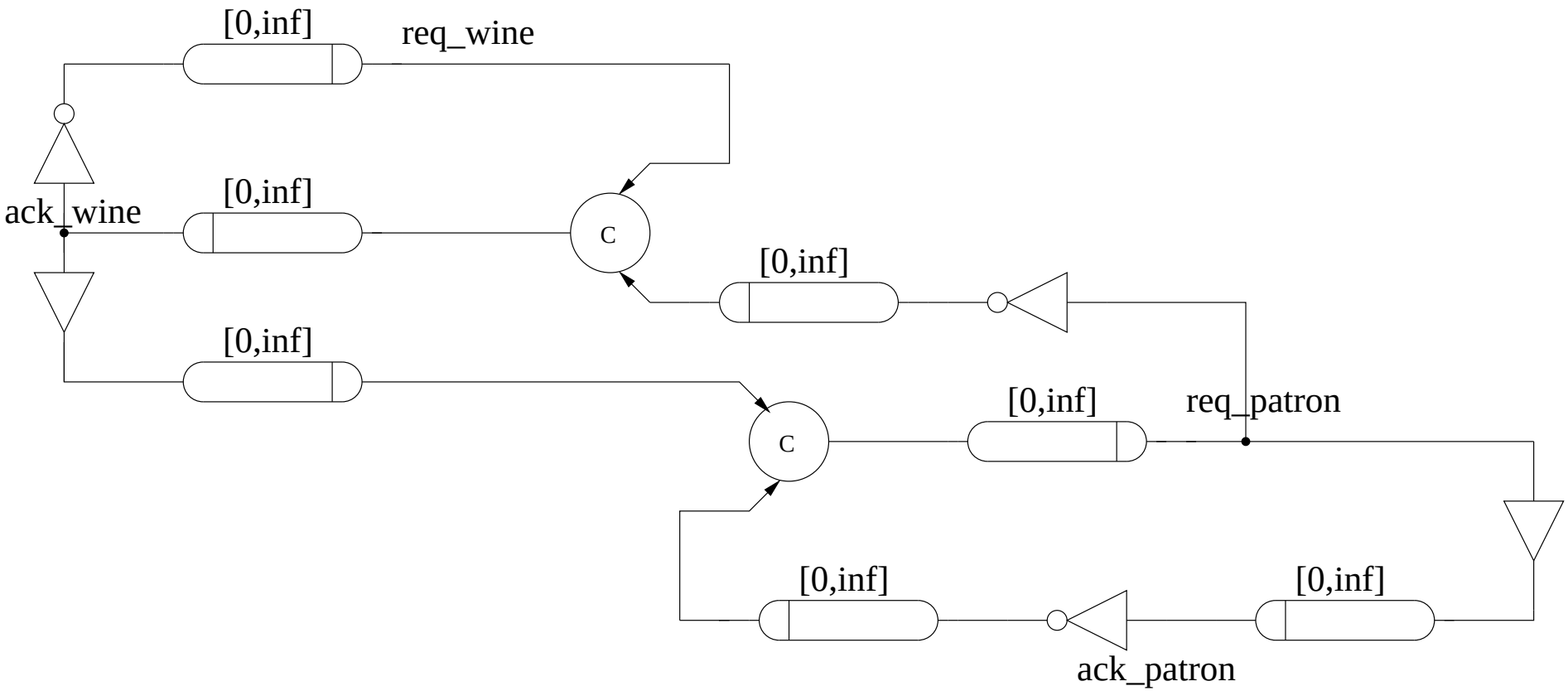
Passive/Lazy-Active Reshuffled Circuit



Passive/Lazy-Active Reshuffled Circuit



Passive/Lazy-Active Reshuffled Circuit



This circuit is *delay-insensitive*.

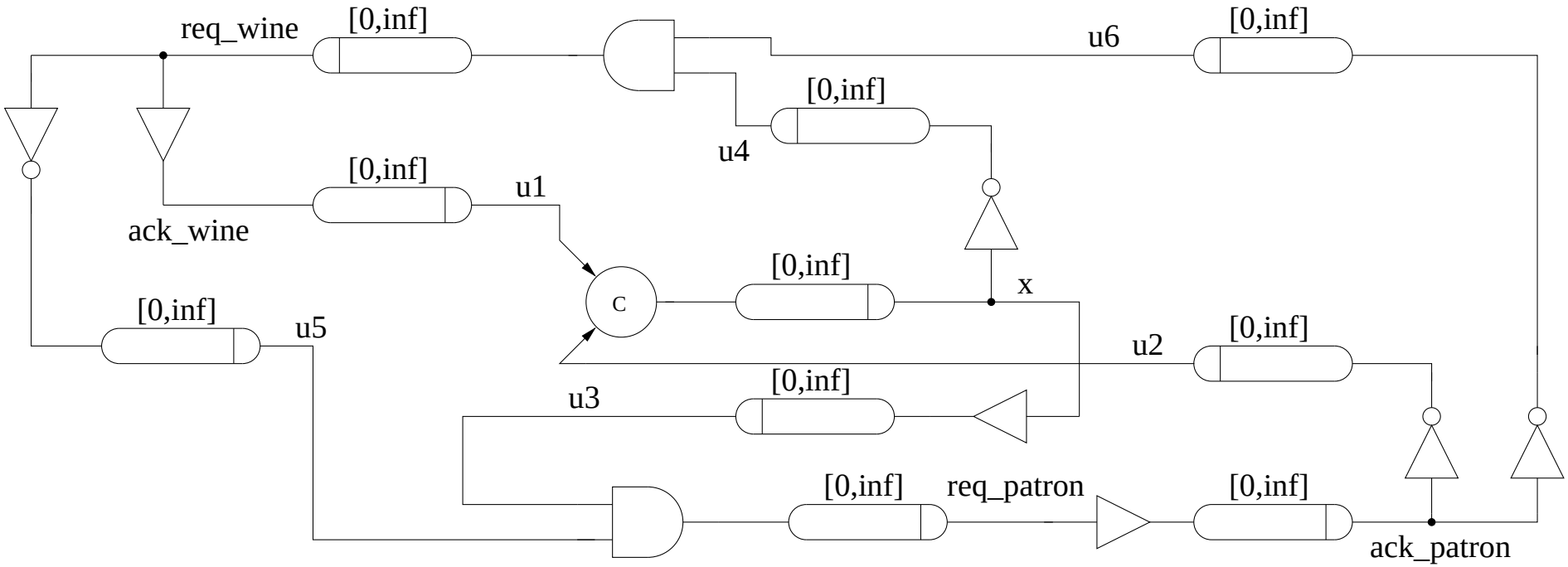
Active/Active Protocol

```
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;
```

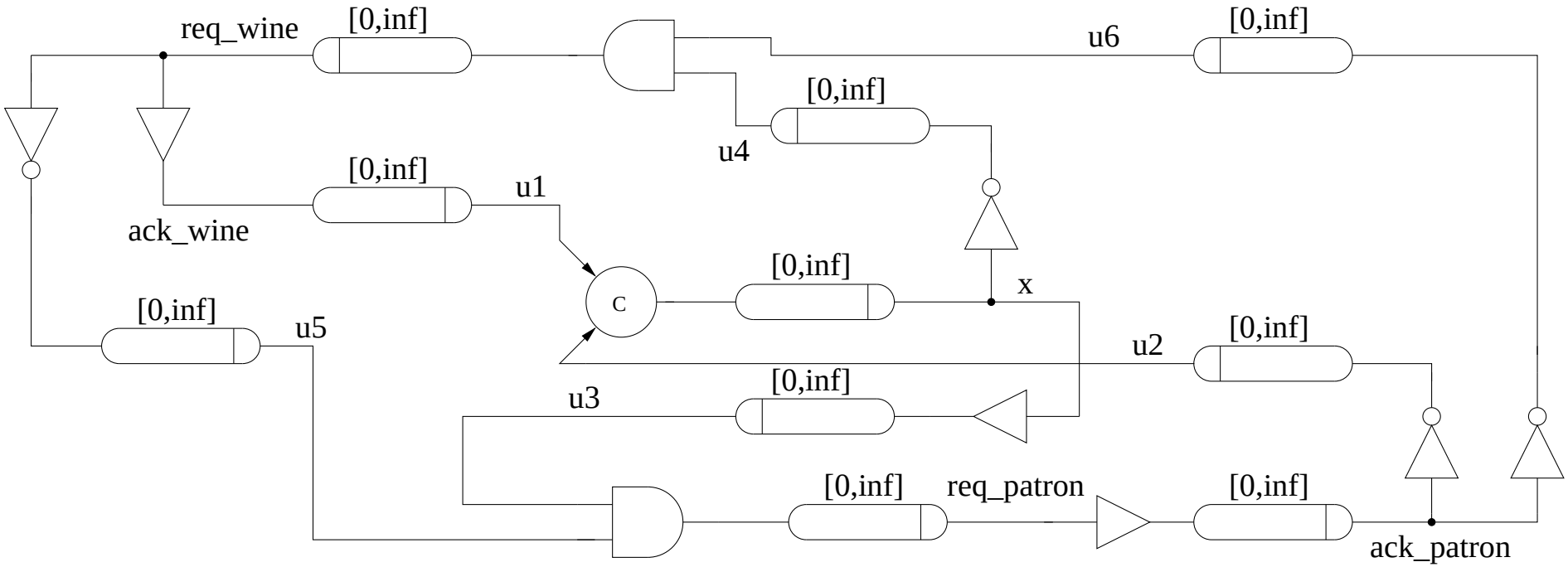
Active/Active State Variable

```
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;
```

Active/Active State Variable Circuit

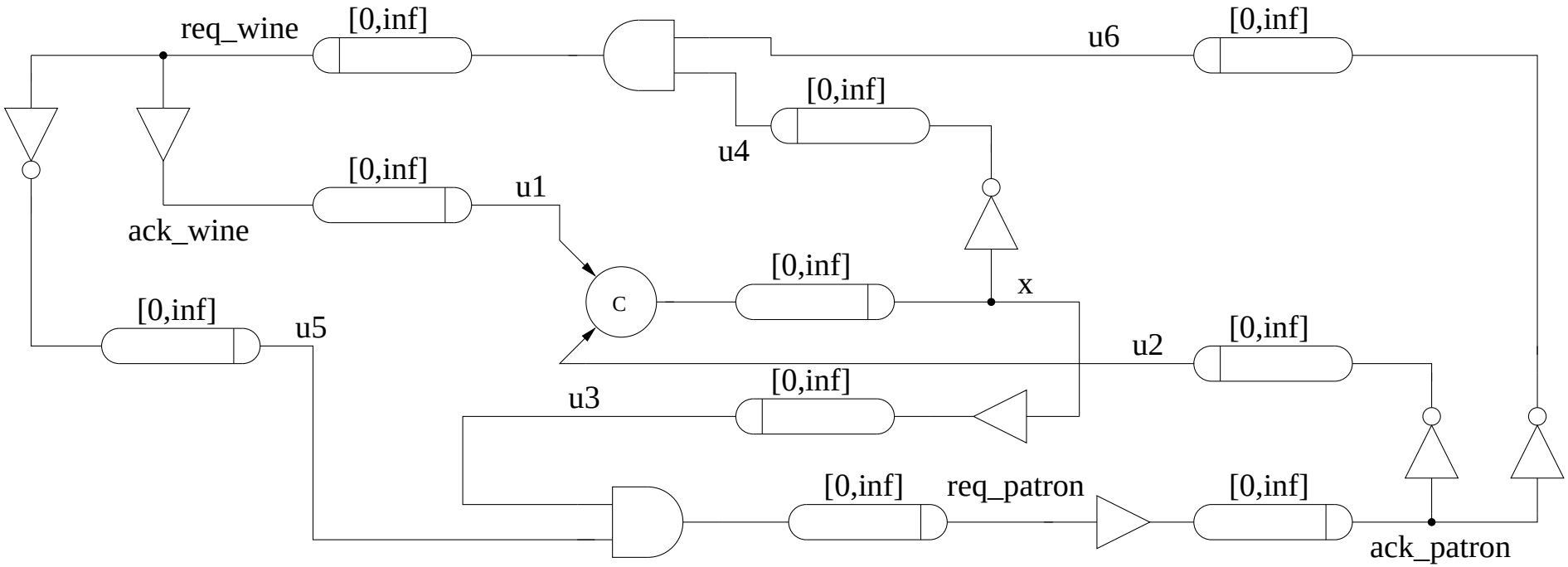


Active/Active State Variable Circuit



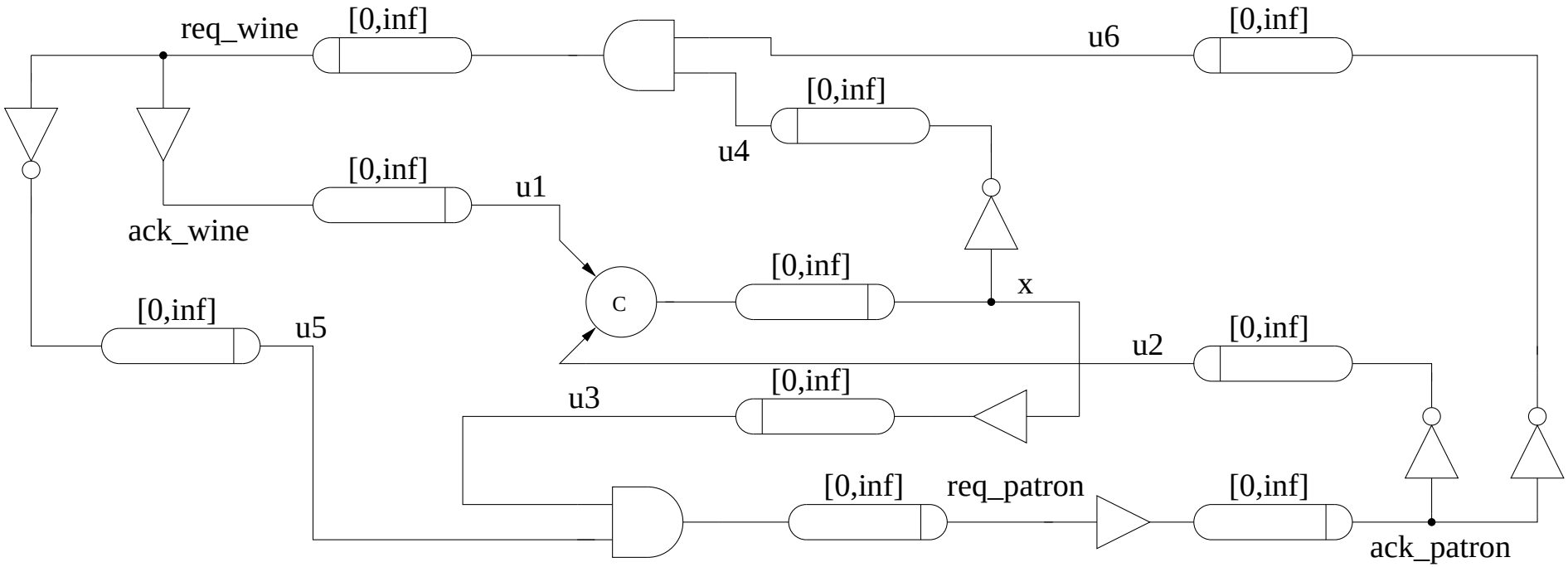
req_wine+, *ack_wine+*, *x+*, *req_wine-*, *ack_wine-*, *req_patron+*, *ack_patron+*

Active/Active State Variable Circuit



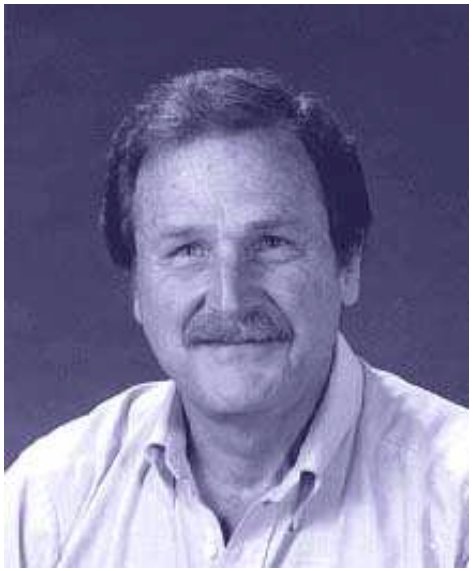
req_wine+, *ack_wine+*, *x+*, *req_wine-*, *ack_wine-*, *req_patron+*, *ack_patron+*
u1-, *u2-*, *x-*, *u4+*, *u6-*

Active/Active State Variable Circuit



req_wine+, *ack_wine+*, *x+*, *req_wine-*, *ack_wine-*, *req_patron+*, *ack_patron+*
u1-, *u2-*, *x-*, *u4+*, *u6-*
req_wine glitches!

Huffman Circuits



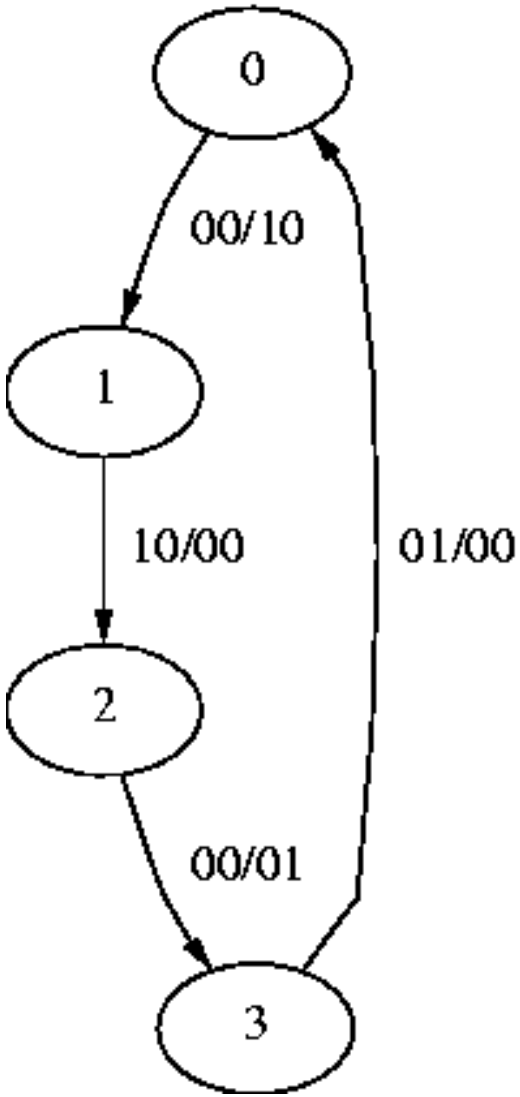
David Huffman

- *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.

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)

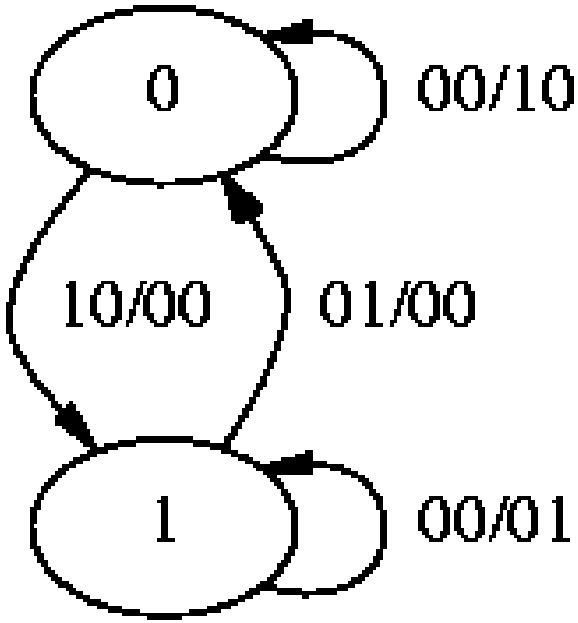


ack_wine / *ack_patron*

	00	01	11	10
0	1, –0	⓪, 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)



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*

Karnaugh Maps for Huffman's A/A Circuit

ack_wine/ack_patron

x	00	01	11	10
0	1	0	–	–
1	0	0	–	0

req_wine

ack_wine/ack_patron

x	00	01	11	10
0	0	0	–	0
1	1	–	–	0

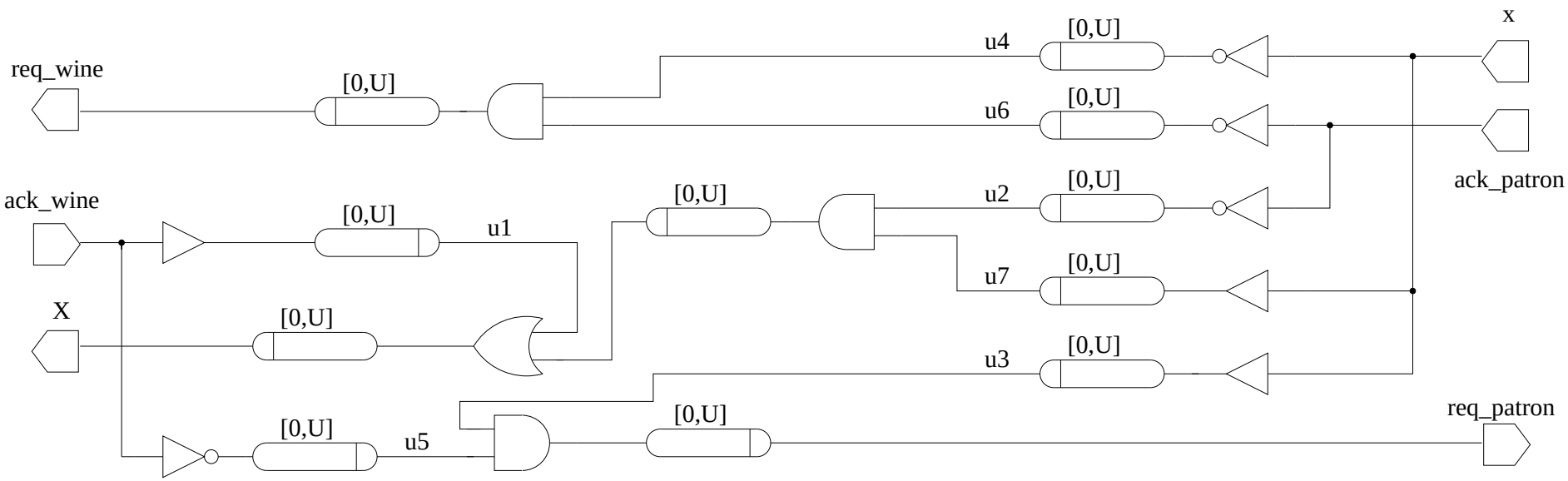
req_patron

ack_wine/ack_patron

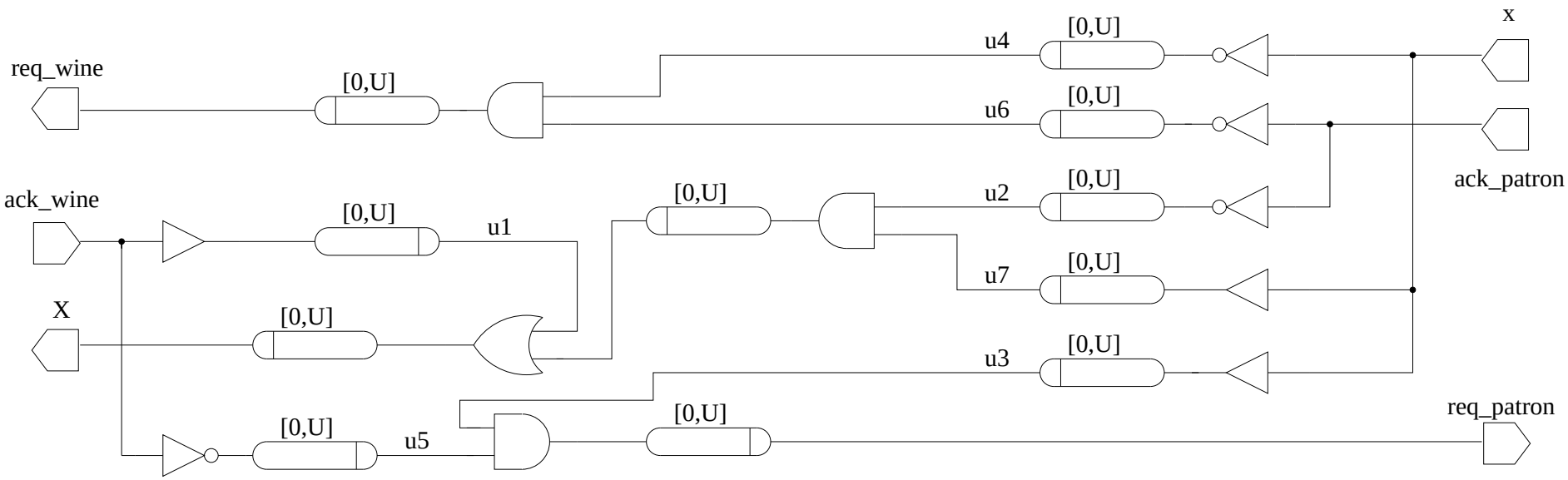
x	00	01	11	10
0	0	0	–	1
1	1	0	–	1

x

Huffman's Active/Active Circuit

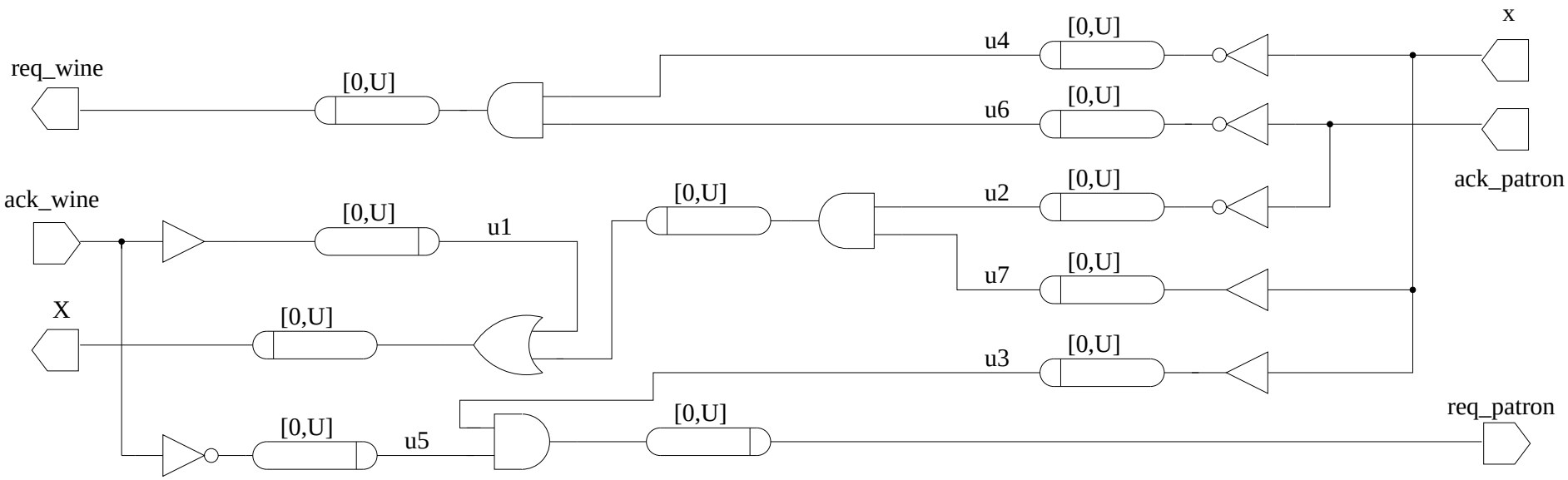


Huffman's Active/Active Circuit



req_wine+, *ack_wine+*, *X+*, *x+*, *req_wine-*, *ack_wine-*, *req_patron+*, *ack_patron+*

Huffman's Active/Active Circuit



req_wine+, *ack_wine+*, *X+*, *x+*, *req_wine-*, *ack_wine-*, *req_patron+*, *ack_patron+*

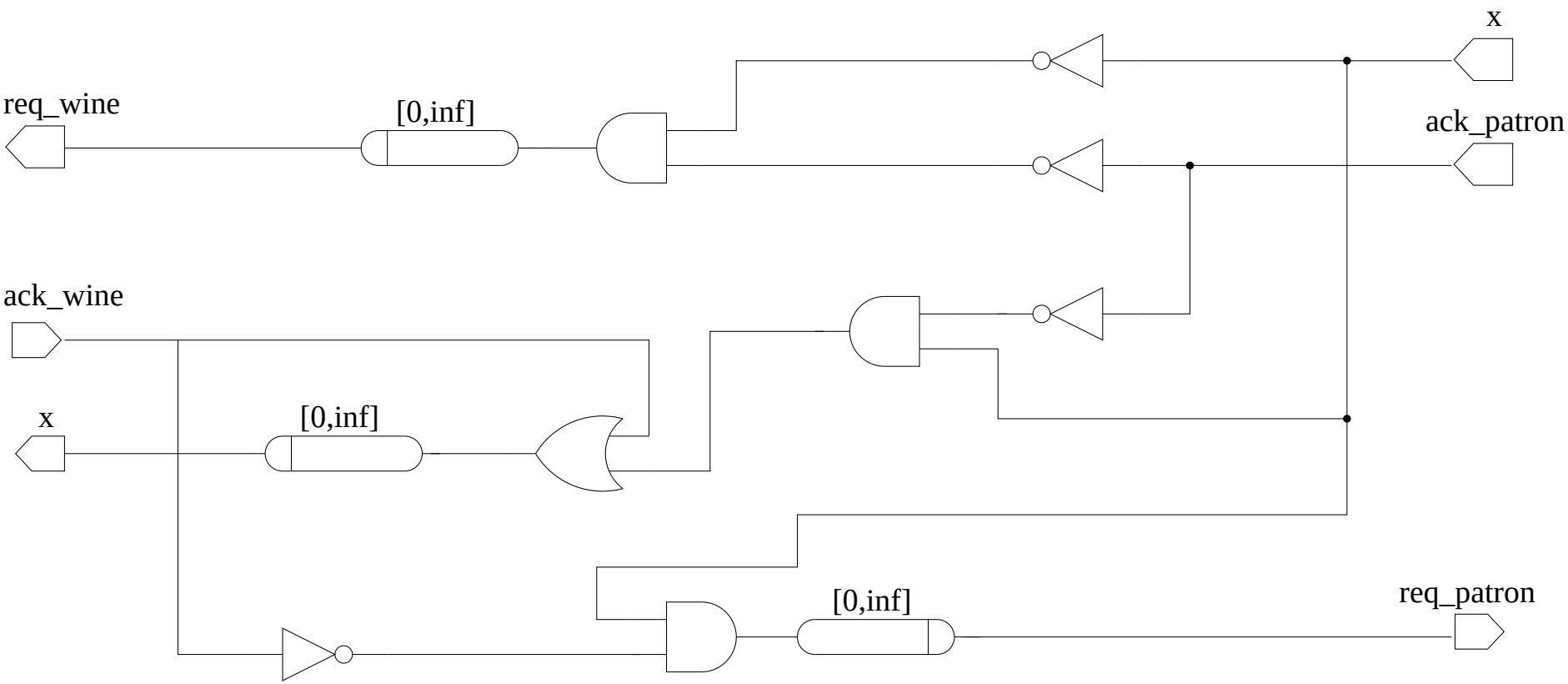
x will not go low until after both *u2-* and *u6-* due to feedback delay assumption.

Muller Circuits

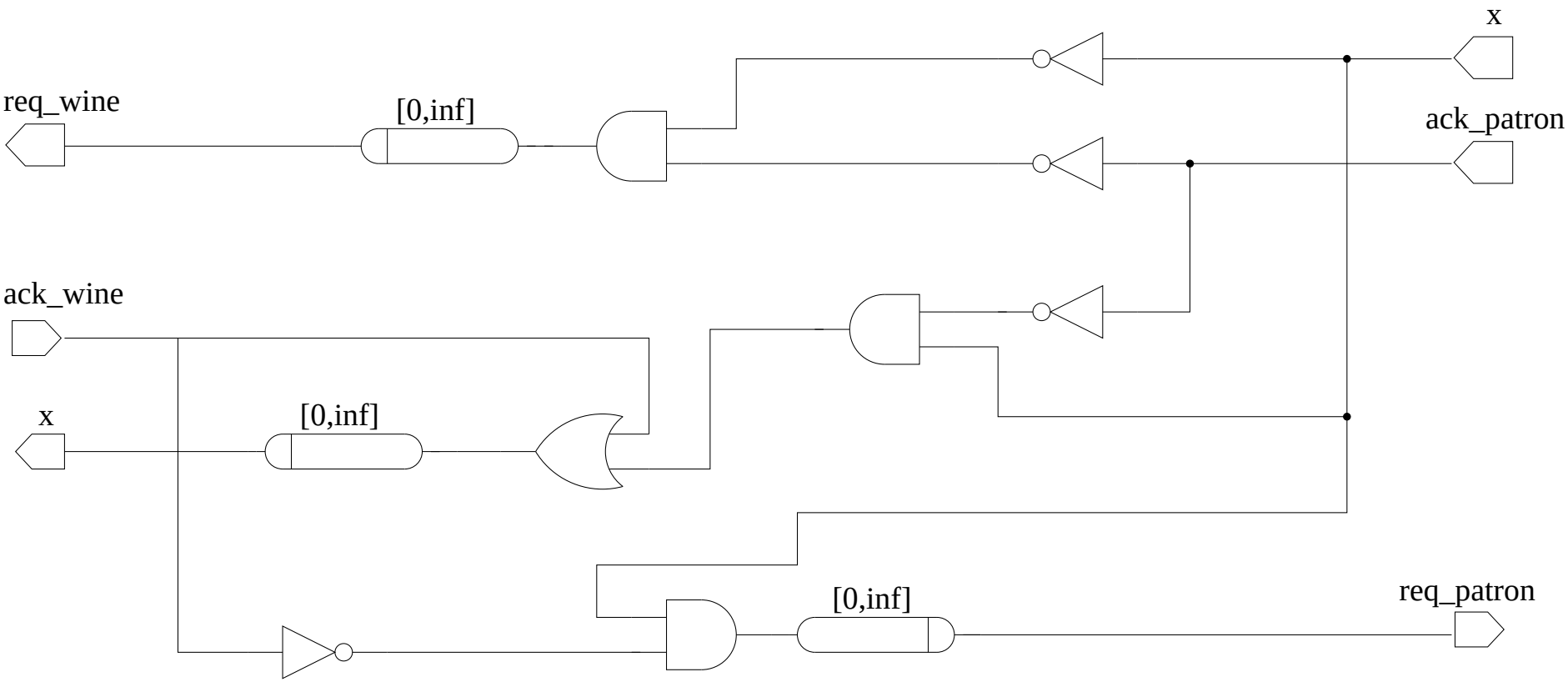
David Muller

- *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 Circuit

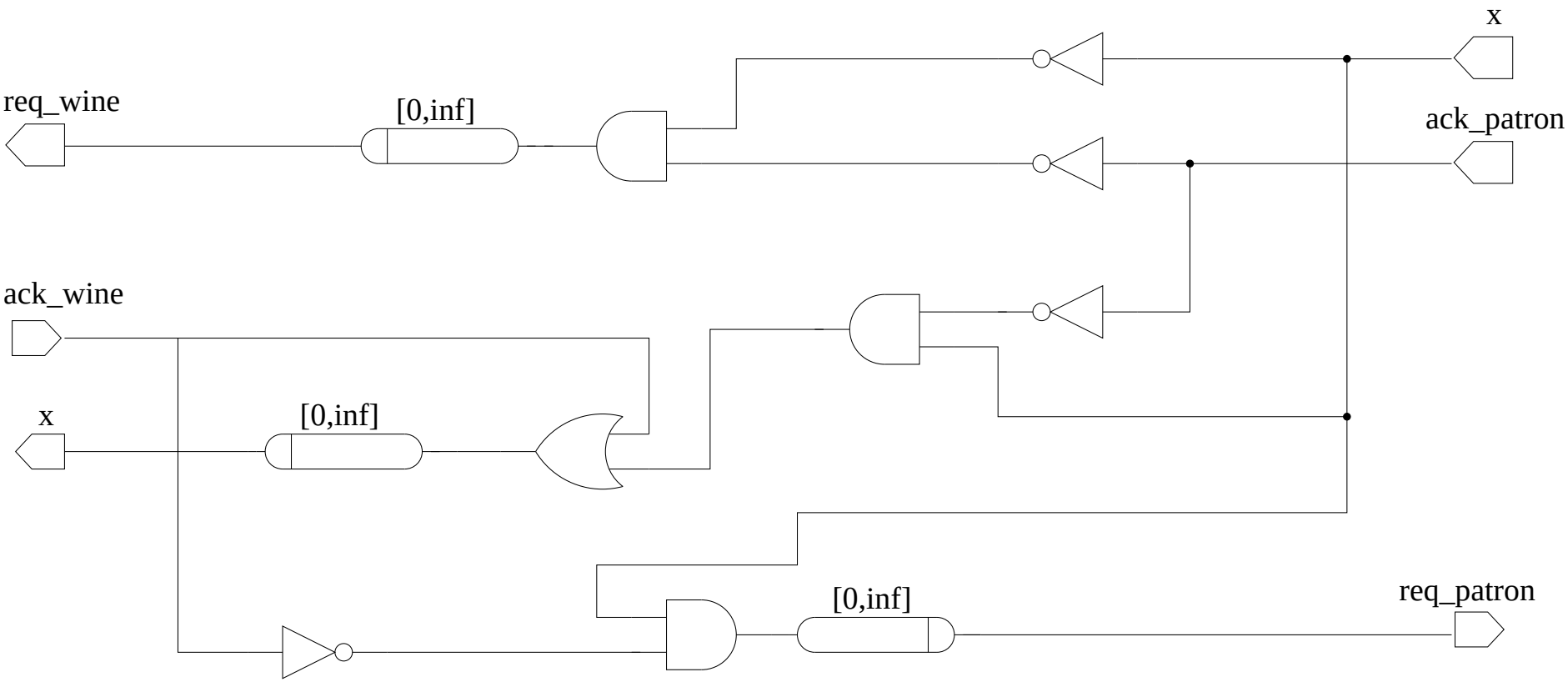


Muller's Active/Active Circuit



req_wine+, *ack_wine+*, *x+*, *req_wine-*, *ack_wine-*, *req_patron+*, *ack_patron+*

Muller's Active/Active 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.

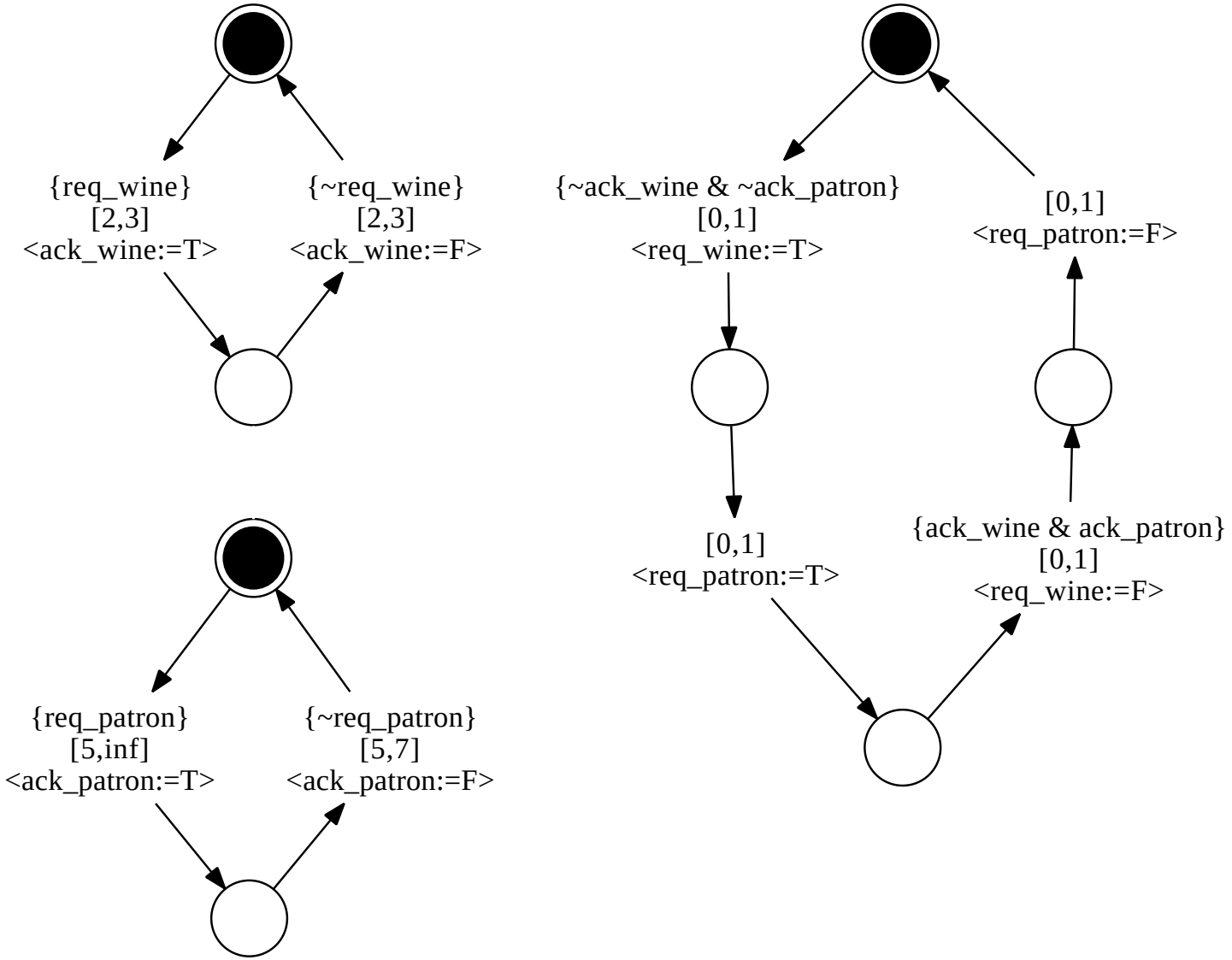
Timed Wine Shop

```
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;
```

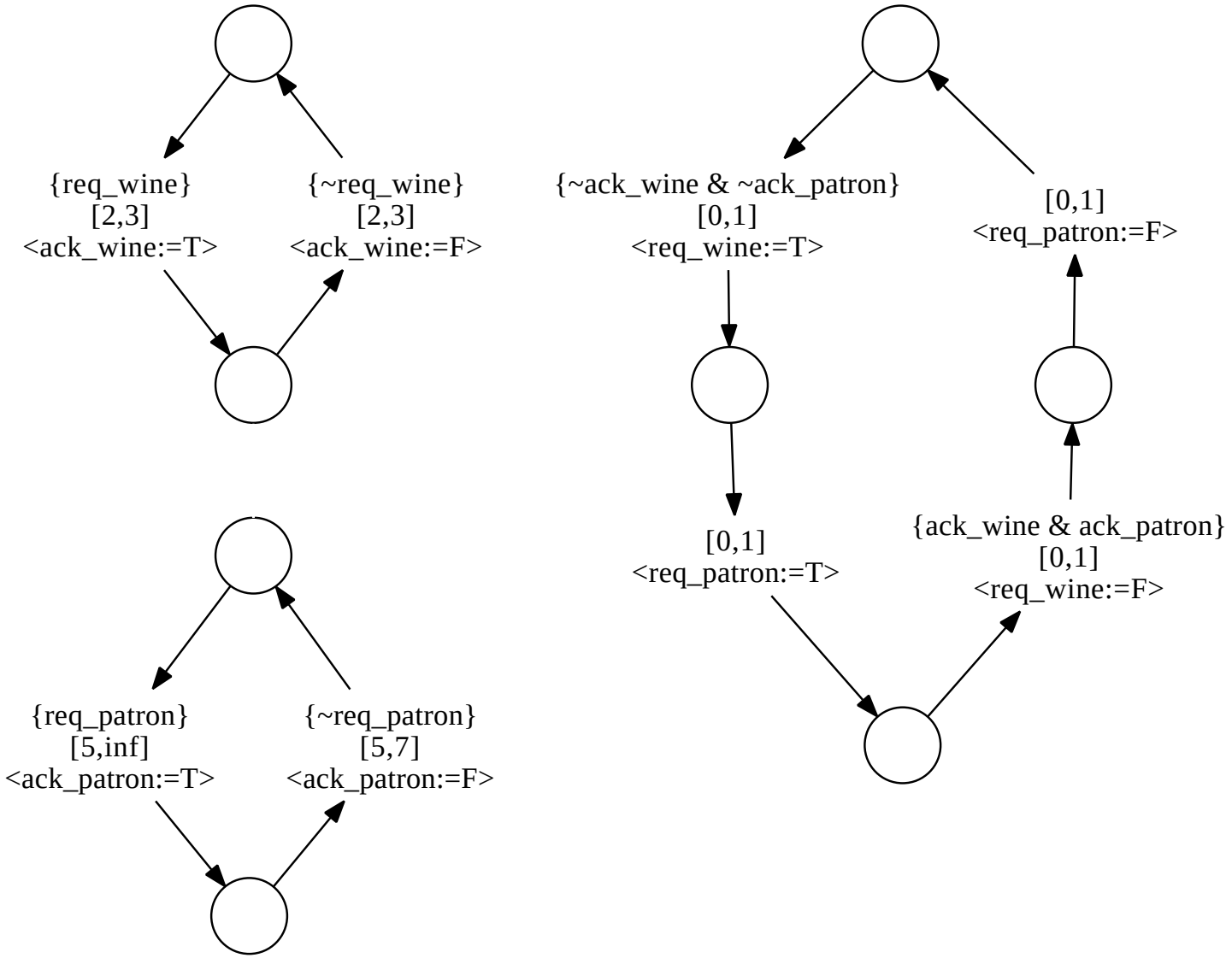
Timed Winery and Patron

```
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;
```

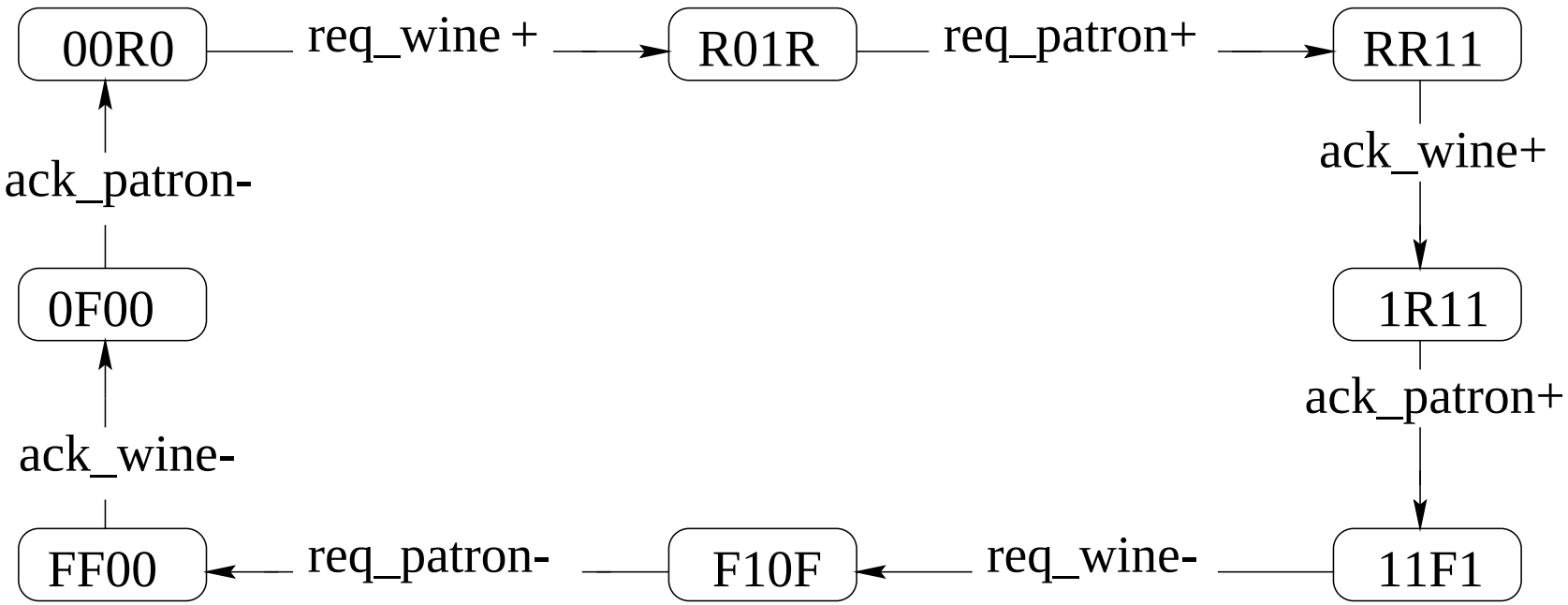
LPN for Timed Wine Shop Example



LPN 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

req_wine/ req_patron

ack_wine/ack_patron

	00	01	11	10
00	1	0	0	–
01	–	–	0	–
11	1	–	0	1
10	1	–	–	–

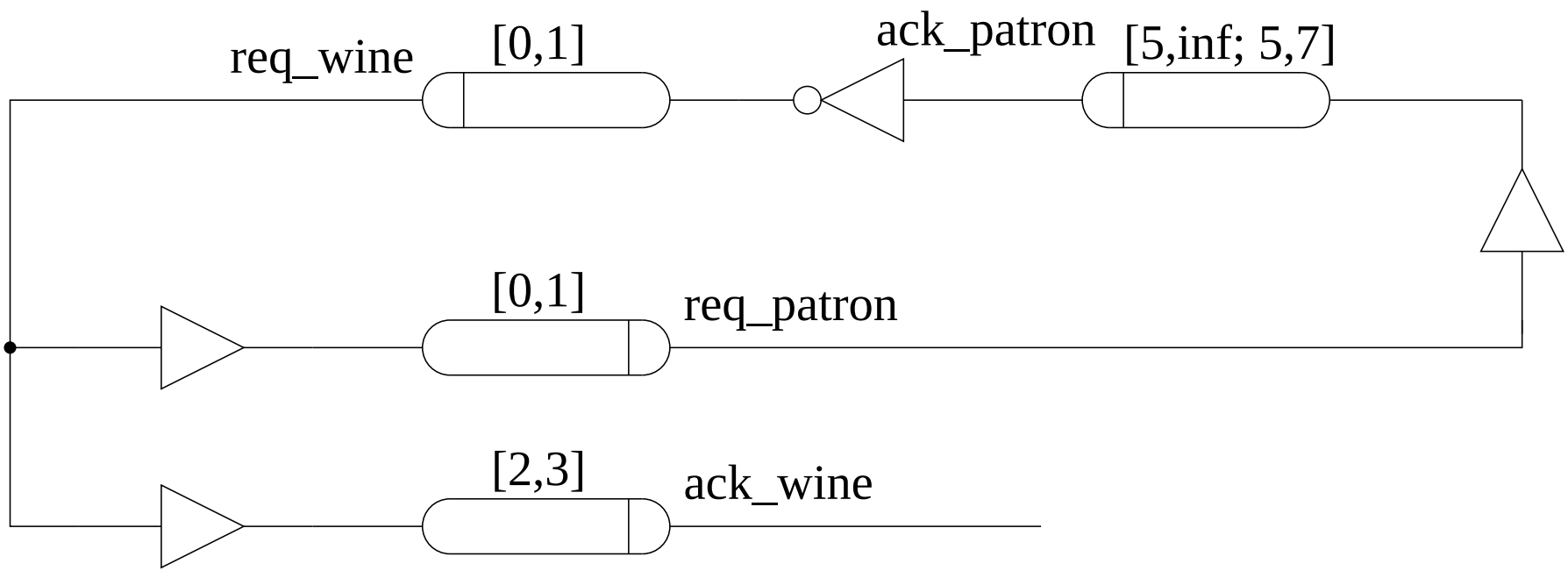
req_wine

ack_wine/ack_patron

	00	01	11	10
00	0	0	0	–
01	–	–	0	–
11	1	–	1	1
10	1	–	–	–

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 then 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($ack\_patron \Rightarrow ack\_wine$)
- When the wine is requested, it eventually arrives:
  - $req\_wine \Rightarrow$ Eventually(ack_wine)

Summary of Course Topics

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