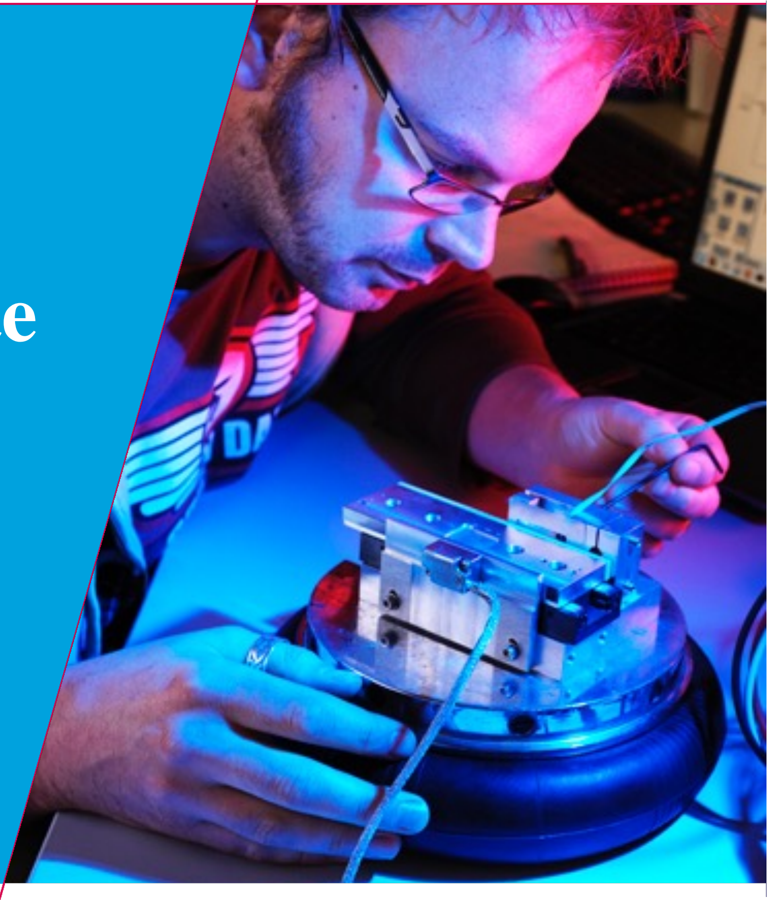
2IC30: Computer Systems
Moore and Mealy Machines; Control of registers via finite State Automata Hardware description languages.

Jan Friso Groote



Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

Example: the On-Off Switch (1)

Specification

If the push button is pressed, the light goes “on”, and stays “on”; if the same button is pressed again the light goes “off”, and stays “off”, and so on indefinitely.

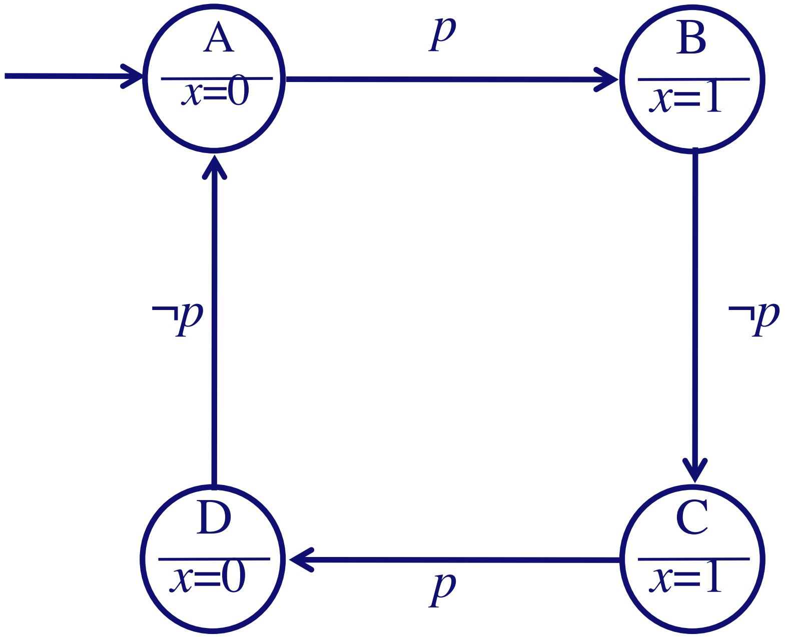
Variables

$\neg p$: the push button is “open”

p : the push button is “closed”

$\neg x$: the light is “off”

x : the light is “on”



Example: the On-Off Switch (2)

Output table and State-transition table

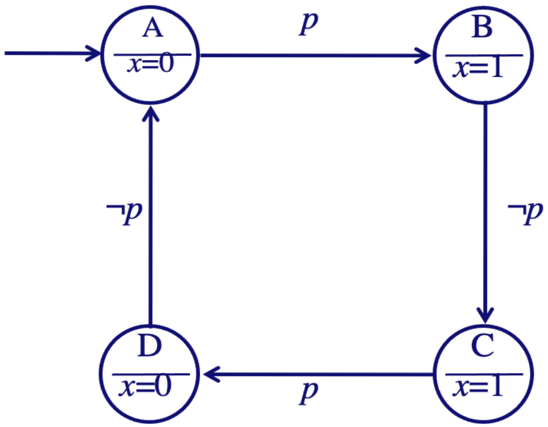
From the state-transition diagram we derive an *output table* and an (abstract) *state-transition table*, in terms of the (abstract) names assigned to the states.

output table

state	x
A	0
B	1
C	1
D	0

state-transition table

state	p	$\text{state}_{\text{new}}$
A	0	A
A	1	B
B	1	B
B	0	C
C	0	C
C	1	D
D	1	D
D	0	A



Technische Universiteit **Eindhoven** University of Technology

Example: the On-Off Switch (3)

State assignment

Several different state assignments are possible; binary representation of 4 states requires *at least* 2 bits, for example:

just counting

state	s	t
A	0	0
B	0	1
C	1	0
D	1	1

Gray code

state	s	t
A	0	0
B	1	0
C	1	1
D	0	1

one-hot encoding

state	s_{A}	s_{B}	s_{C}	s_{D}
A	1	0	0	0
B	0	1	0	0
C	0	0	1	0
D	0	0	0	1

Example: the On-Off Switch (4)

Boolean Functions

We choose the Gray encoding for the states, because this makes the function for the output simple. From the (abstract) state-transition table we derive a (concrete) state-transition table:

output

s	t	x
0	0	0
1	0	1
1	1	1
0	1	0

new states

state	p	$\text{state}_{\text{new}}$
A	0	A
A	1	B
B	1	B
B	0	C
C	0	C
C	1	D
D	1	D
D	0	A

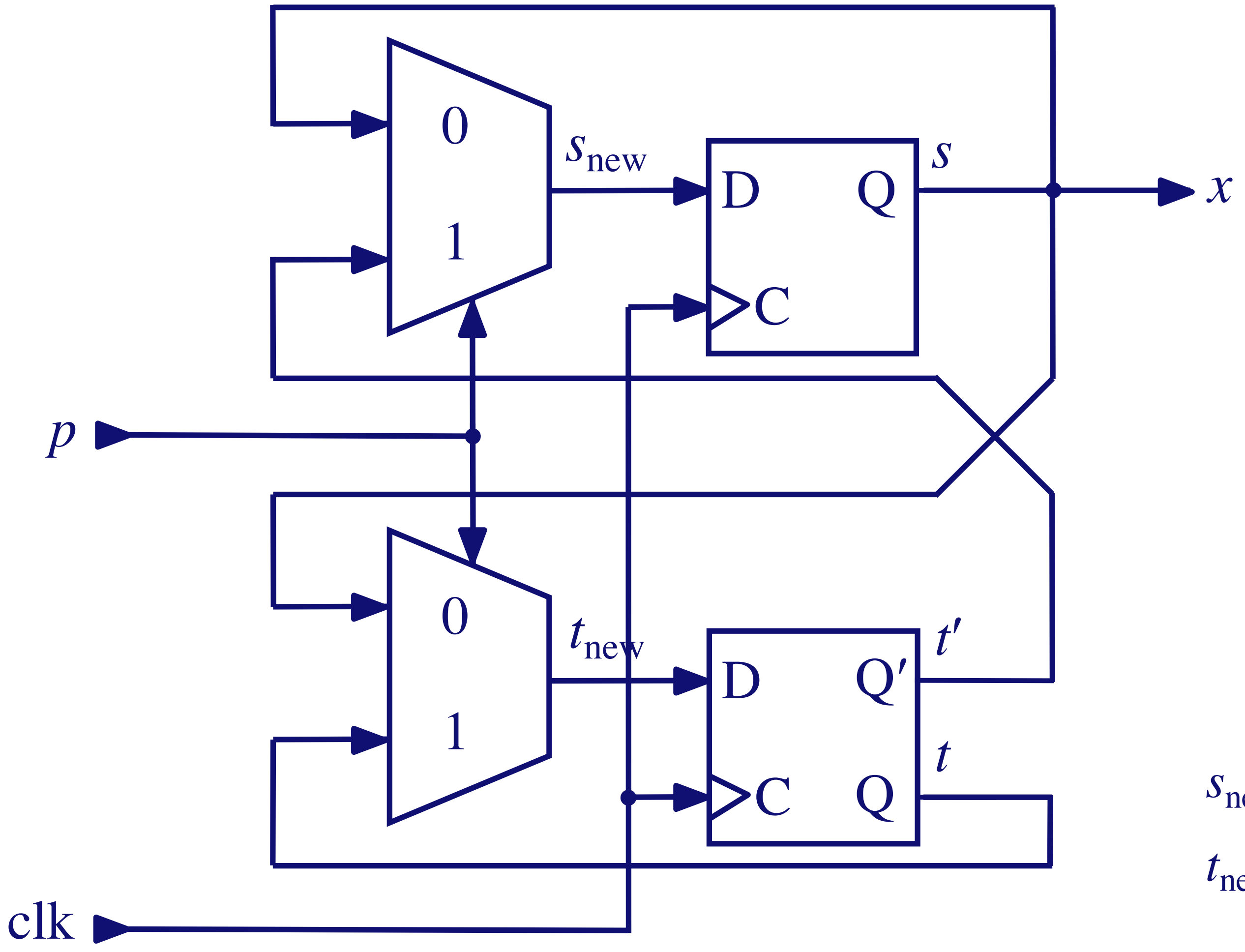
new states

s	t	p	s_{new}	t_{new}
0	0	0	0	0
0	0	1	1	0
1	0	1	1	0
1	0	0	1	1
1	1	0	1	1
1	1	1	0	1
0	1	1	0	1
0	1	0	0	0

Example: the On-Off Switch (7)

Synchronous sequential circuit: driven by a periodic “clock”

The same circuit with feedback, but via D-Flip-Flops:

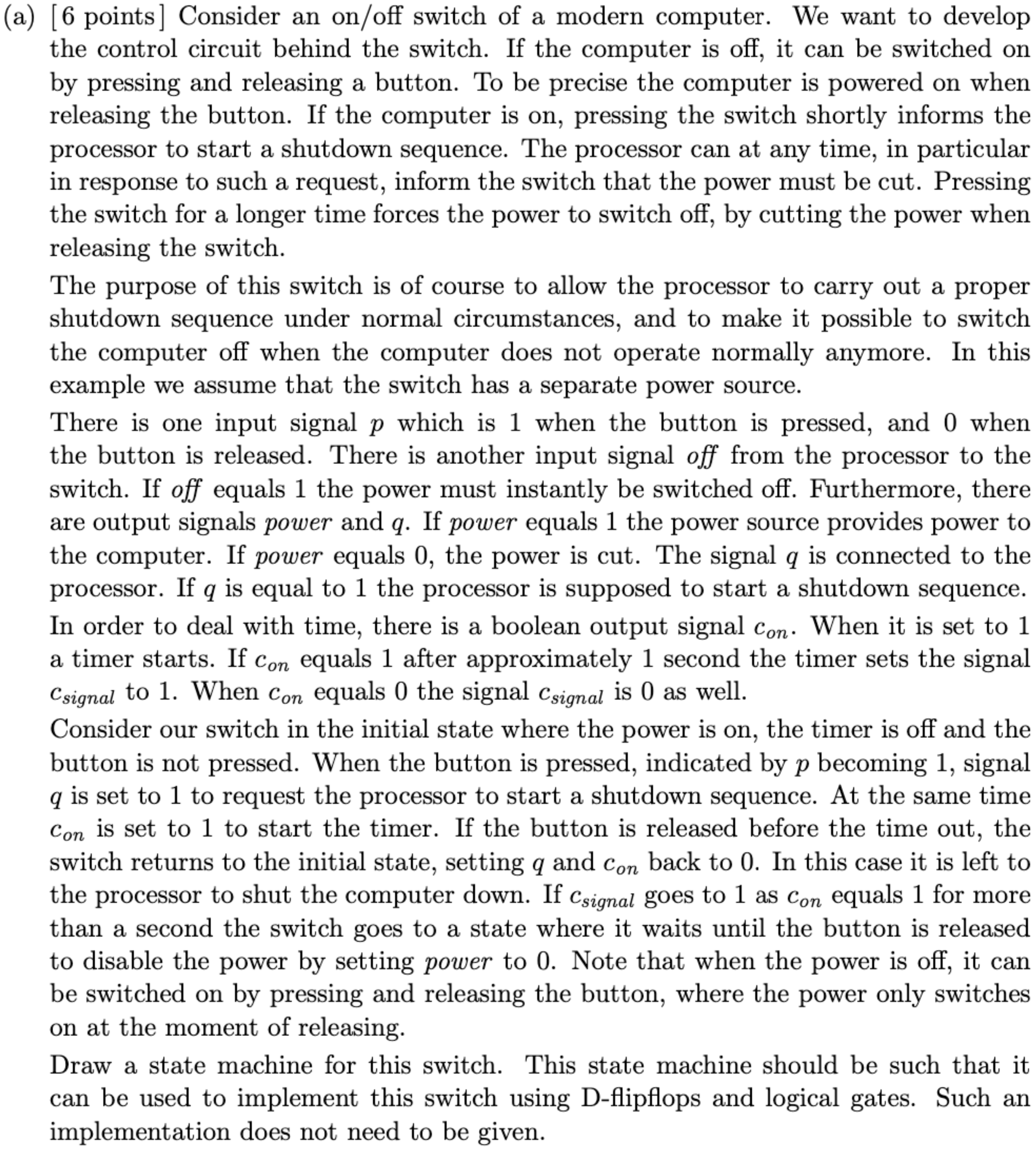


$$s_{\text{new}} = p \wedge \neg t \vee \neg p \wedge s$$

$$t_{\text{new}} = \neg p \wedge s \vee p \wedge t$$

Questions?

How to design an state machine.



(a) [6 points] Consider an on/off switch of a modern computer. We want to develop the control circuit behind the switch. If the computer is off, it can be switched on by pressing and releasing a button. To be precise the computer is powered on when releasing the button. If the computer is on, pressing the switch shortly informs the processor to start a shutdown sequence. The processor can at any time, in particular in response to such a request, inform the switch that the power must be cut. Pressing the switch for a longer time forces the power to switch off, by cutting the power when releasing the switch.

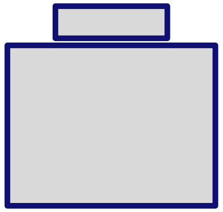
The purpose of this switch is of course to allow the processor to carry out a proper shutdown sequence under normal circumstances, and to make it possible to switch the computer off when the computer does not operate normally anymore. In this example we assume that the switch has a separate power source.

There is one input signal p which is 1 when the button is pressed, and 0 when the button is released. There is another input signal *off* from the processor to the switch. If *off* equals 1 the power must instantly be switched off. Furthermore, there are output signals *power* and q. If *power* equals 1 the power source provides power to the computer. If *power* equals 0, the power is cut. The signal q is connected to the processor. If q is equal to 1 the processor is supposed to start a shutdown sequence.

In order to deal with time, there is a boolean output signal c_{on}. When it is set to 1 a timer starts. If c_{on} equals 1 after approximately 1 second the timer sets the signal c_{signal} to 1. When c_{on} equals 0 the signal c_{signal} is 0 as well.

Consider our switch in the initial state where the power is on, the timer is off and the button is not pressed. When the button is pressed, indicated by p becoming 1, signal q is set to 1 to request the processor to start a shutdown sequence. At the same time c_{on} is set to 1 to start the timer. If the button is released before the time out, the switch returns to the initial state, setting q and c_{on} back to 0. In this case it is left to the processor to shut the computer down. If c_{signal} goes to 1 as c_{on} equals 1 for more than a second the switch goes to a state where it waits until the button is released to disable the power by setting *power* to 0. Note that when the power is off, it can be switched on by pressing and releasing the button, where the power only switches on at the moment of releasing.

Draw a state machine for this switch. This state machine should be such that it can be used to implement this switch using D-flipflops and logical gates. Such an implementation does not need to be given.



Computer power button

Off: press → Computer on.

On: short press → activate shutdown.

On: long press → force power off.

How to design an state machine.

Inputs: p press button.

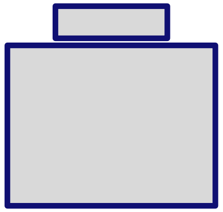
off computer asks to switch off.

c_{signal} timer times out.

Outputs: $power$ power to computer.

q start shutdown sequence.

c_{on} activate timer.

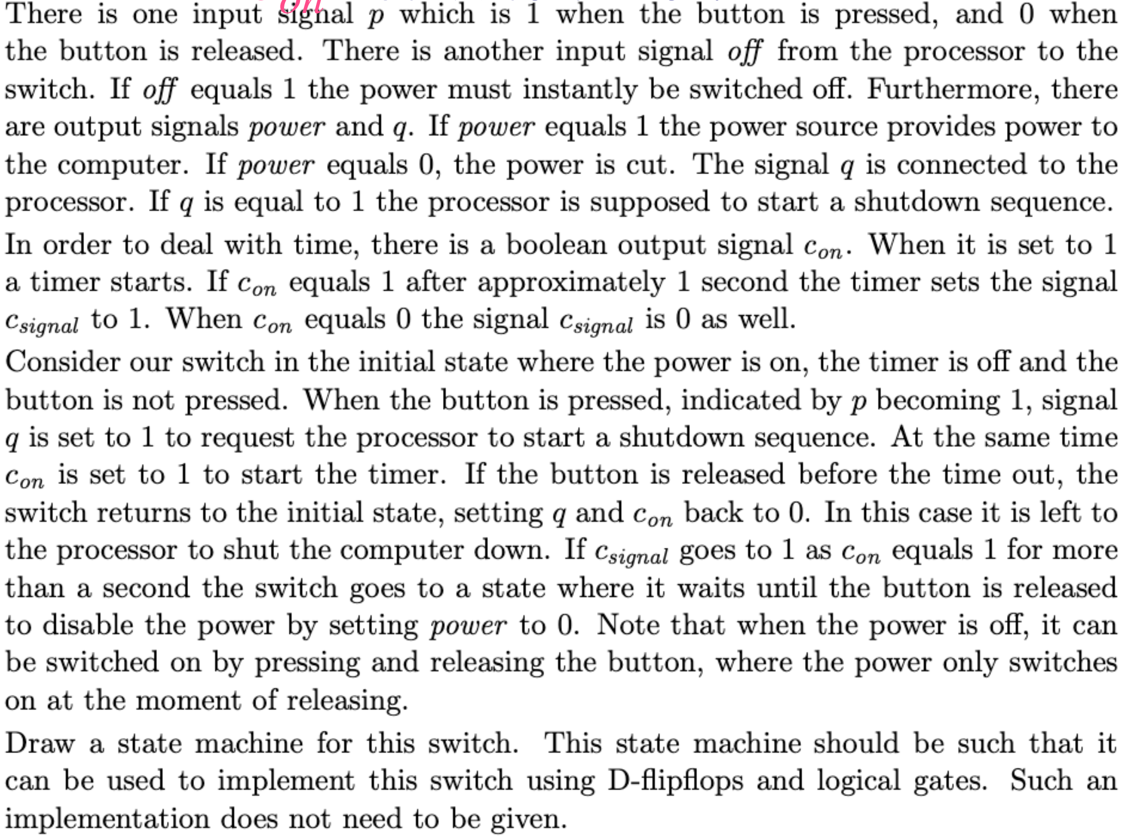


Computer power button

Off: press → Computer on.

On: short press → activate shutdown.

On: long press → force power off.

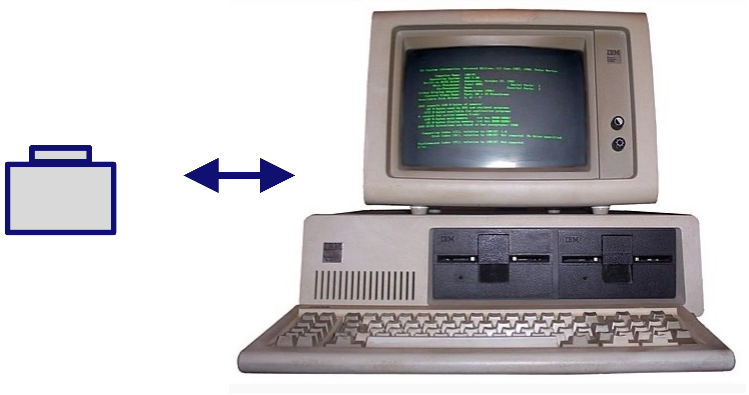


There is one input signal p which is 1 when the button is pressed, and 0 when the button is released. There is another input signal *off* from the processor to the switch. If *off* equals 1 the power must instantly be switched off. Furthermore, there are output signals *power* and q. If *power* equals 1 the power source provides power to the computer. If *power* equals 0, the power is cut. The signal q is connected to the processor. If q is equal to 1 the processor is supposed to start a shutdown sequence.

In order to deal with time, there is a boolean output signal c_{on}. When it is set to 1 a timer starts. If c_{on} equals 1 after approximately 1 second the timer sets the signal c_{signal} to 1. When c_{on} equals 0 the signal c_{signal} is 0 as well.

Consider our switch in the initial state where the power is on, the timer is off and the button is not pressed. When the button is pressed, indicated by p becoming 1, signal q is set to 1 to request the processor to start a shutdown sequence. At the same time c_{on} is set to 1 to start the timer. If the button is released before the time out, the switch returns to the initial state, setting q and c_{on} back to 0. In this case it is left to the processor to shut the computer down. If c_{signal} goes to 1 as c_{on} equals 1 for more than a second the switch goes to a state where it waits until the button is released to disable the power by setting *power* to 0. Note that when the power is off, it can be switched on by pressing and releasing the button, where the power only switches on at the moment of releasing.

Draw a state machine for this switch. This state machine should be such that it can be used to implement this switch using D-flipflops and logical gates. Such an implementation does not need to be given.



How to design an state machine.

Inputs: p press button.

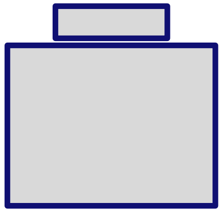
off computer asks to switch off.

c_{signal} timer times out.

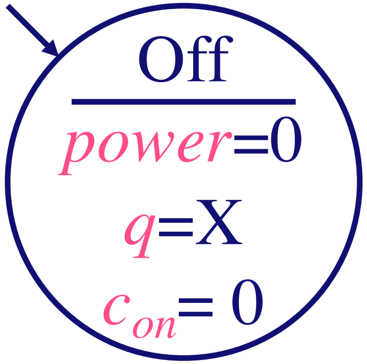
Outputs: $power$ power to computer.

q start shutdown sequence.

c_{on} activate timer.



Computer
power button



Off

$power$=0

q=X

c_{on}= 0

Off: press → Computer on.

On: short press →
activate shutdown.

On: long press →
force power off.

How to design an state machine.

Inputs: p press button.

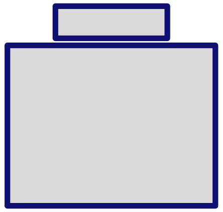
off computer asks to switch off.

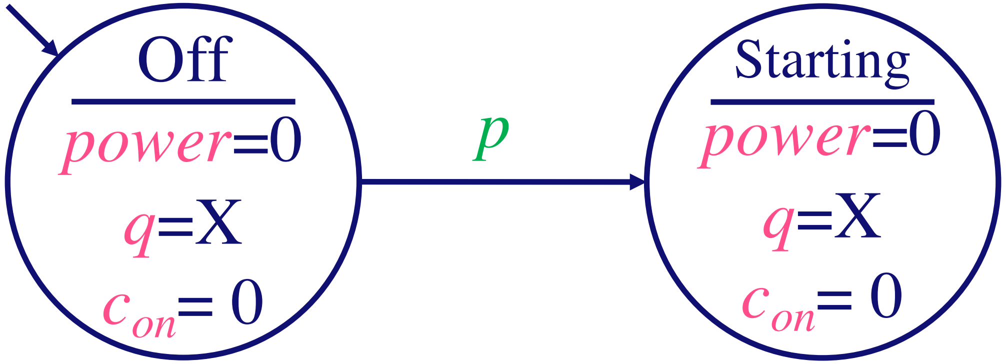
c_{signal} timer times out.

Outputs: $power$ power to computer.

q start shutdown sequence.

c_{on} activate timer.



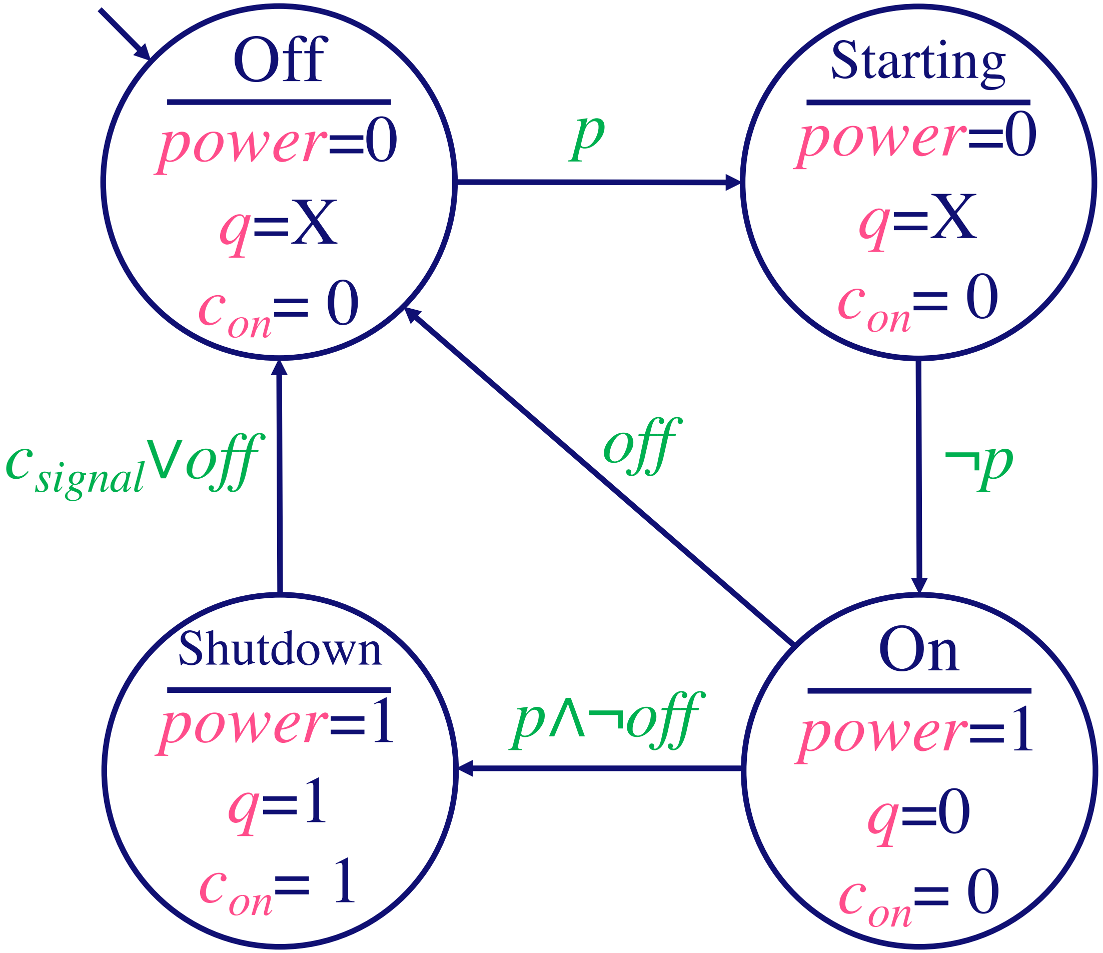


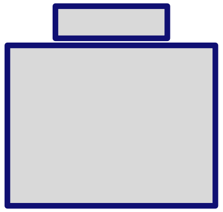
Off: press → Computer on.

On: short press → activate shutdown.

On: long press → force power off.

How to design an state machine.





Computer power button

Inputs: p press button.

off computer asks to switch off.

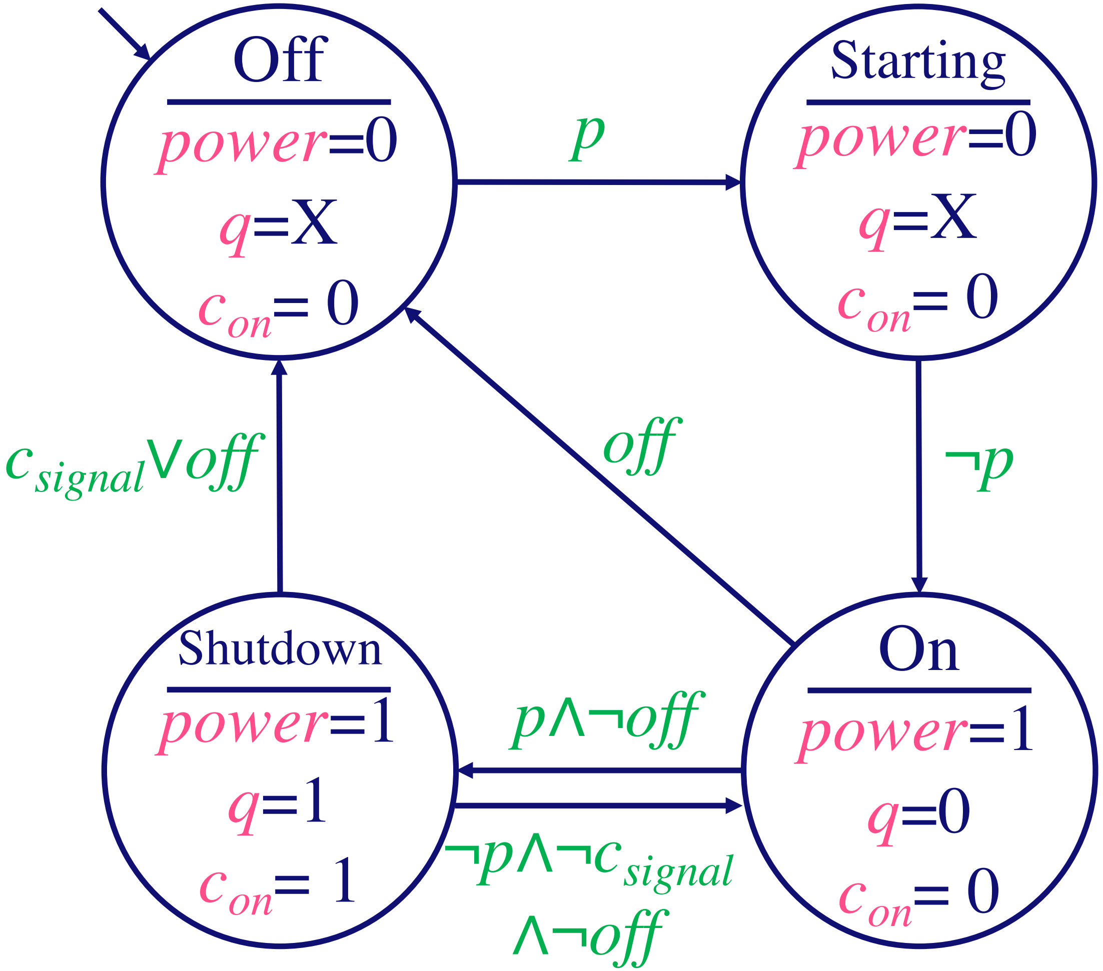
c_{signal} timer times out.

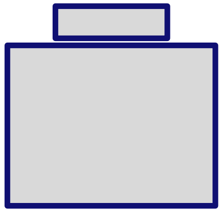
Outputs: $power$ power to computer.

q start shutdown sequence.

c_{on} activate timer.

How to design an state machine.





Computer power button

Inputs: p press button.

off computer asks to switch off.

c_{signal} timer times out.

Outputs: $power$ power to computer.

q start shutdown sequence.

c_{on} activate timer.

Example: the On-Off Switch (2)

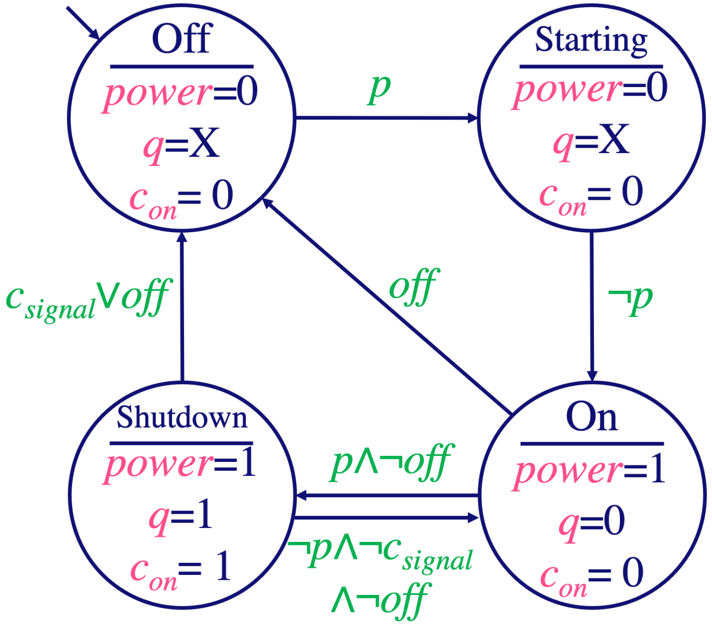
Output table and State-transition table

output table

state	*power*	*q*	c_{on}
Off	0	X	0
Starting	0	X	0
Shutdown	1	1	1
On	1	0	0

state-transition table

state	*p*	c_{signal}	*off*	$\text{state}_{\text{new}}$
Off	0	0	0	Off
Off	0	0	1	Off
Off	0	1	0	Off
Off	0	1	1	Off
Off	1	0	0	Starting
Off	1	0	1	Starting
Off	1	1	0	Starting
Off	1	1	1	Starting



Example: the On-Off Switch (2)

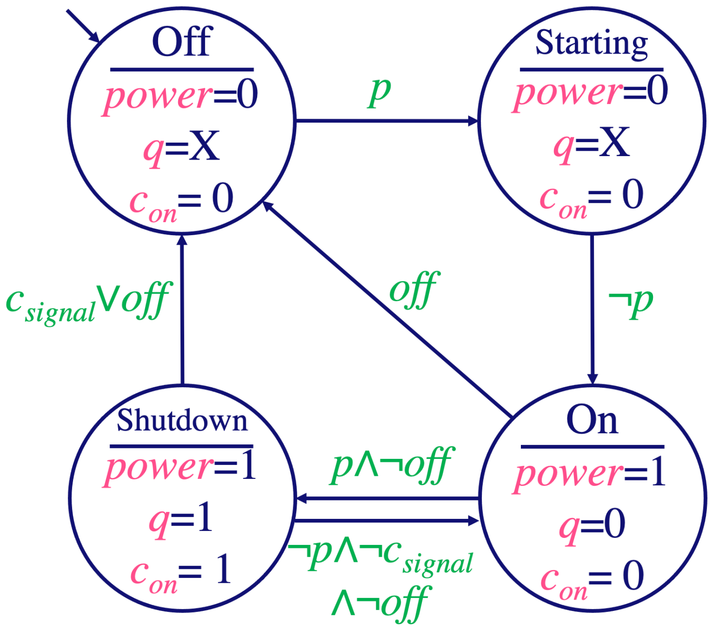
Output table and State-transition table

output table

state	$power$	q	c_{on}
Off	0	X	0
Starting	0	X	0
Shutdown	1	1	1
On	1	0	0

state-transition table (incomplete, 32 entries)

state	p	c_{signal}	off	$\text{state}_{\text{new}}$
Off	0	0	0	Off
Off	0	0	1	Off
Off	0	1	0	Off
Off	0	1	1	Off
Off	1	0	0	Starting
Off	1	0	1	Starting
Off	1	1	0	Starting
Off	1	1	1	Starting
Starting	0	0	0	On
Starting	0	0	1	On
Starting	0	1	0	On
Starting	0	1	1	On
Starting	1	0	0	Starting
Starting	1	0	1	Starting
Starting	1	1	0	Starting



Example: the On-Off Switch (3)

State assignment

Encoding

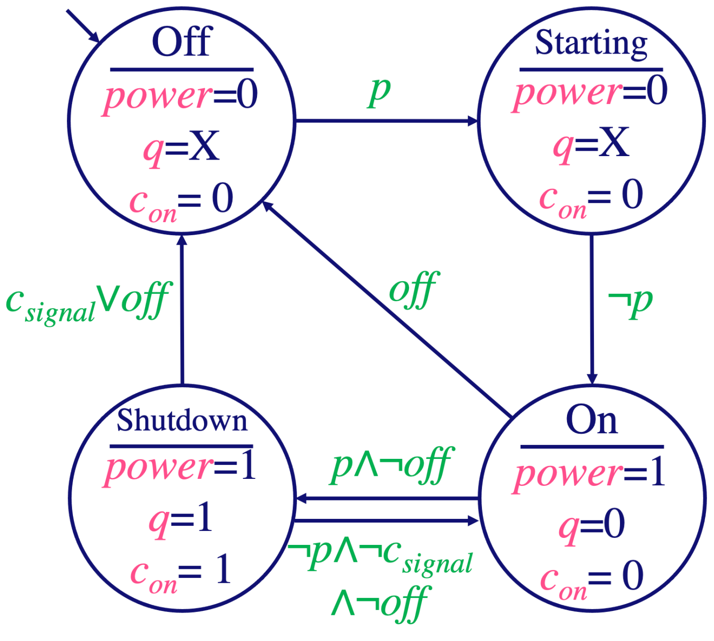
state	s	t
Off	0	0
Starting	0	1
Shutdown	1	0
On	1	1

output table

s	t	*power*	q	c_{on}
0	0	0	X	0
0	1	0	X	0
1	0	1	1	1
1	1	1	0	0

state-transition table (32 entries)

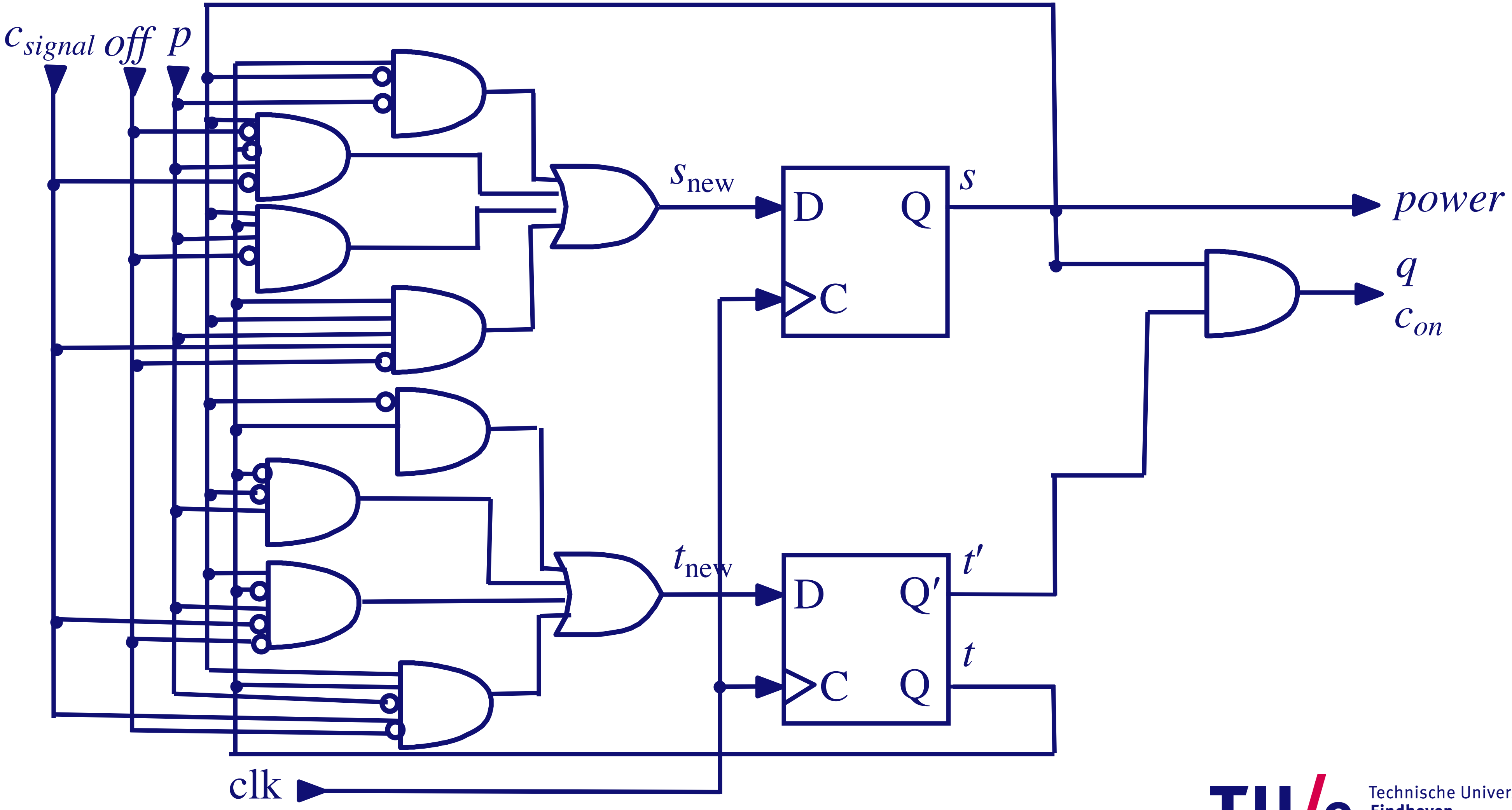
s	t	p	c_{signal}	off	s_{new}	t_{new}
0	0	0	0	0	0	0
0	0	0	0	1	0	0
0	0	0	1	0	0	0
0	0	0	1	1	0	0
0	0	1	0	0	0	1
0	0	1	0	1	0	1
0	0	1	1	0	0	1
0	0	1	1	1	0	1
0	1	0	0	0	1	1
0	1	0	0	1	1	1
0	1	0	1	0	1	1
0	1	0	1	1	1	1
0	1	1	0	0	0	1
0	1	1	0	1	0	1
0	1	1	1	0	0	1



Example: the On-Off Switch (7)

Synchronous sequential circuit:

- $s_{\text{new}} \;=\; \neg s \wedge t \wedge \neg p \vee s \wedge \neg t \wedge p \wedge \neg c_{signal} \wedge \neg off \vee s \wedge t \wedge p \wedge \neg off \vee s \wedge t \wedge p \wedge c_{signal} \wedge \neg off$
  $t_{\text{new}} \;=\; \neg s \wedge t \vee \neg s \wedge \neg t \wedge p \vee s \wedge \neg t \wedge p \wedge \neg c_{signal} \wedge \neg off \vee s \wedge t \wedge \neg p \wedge c_{signal} \wedge \neg off$



Questions?

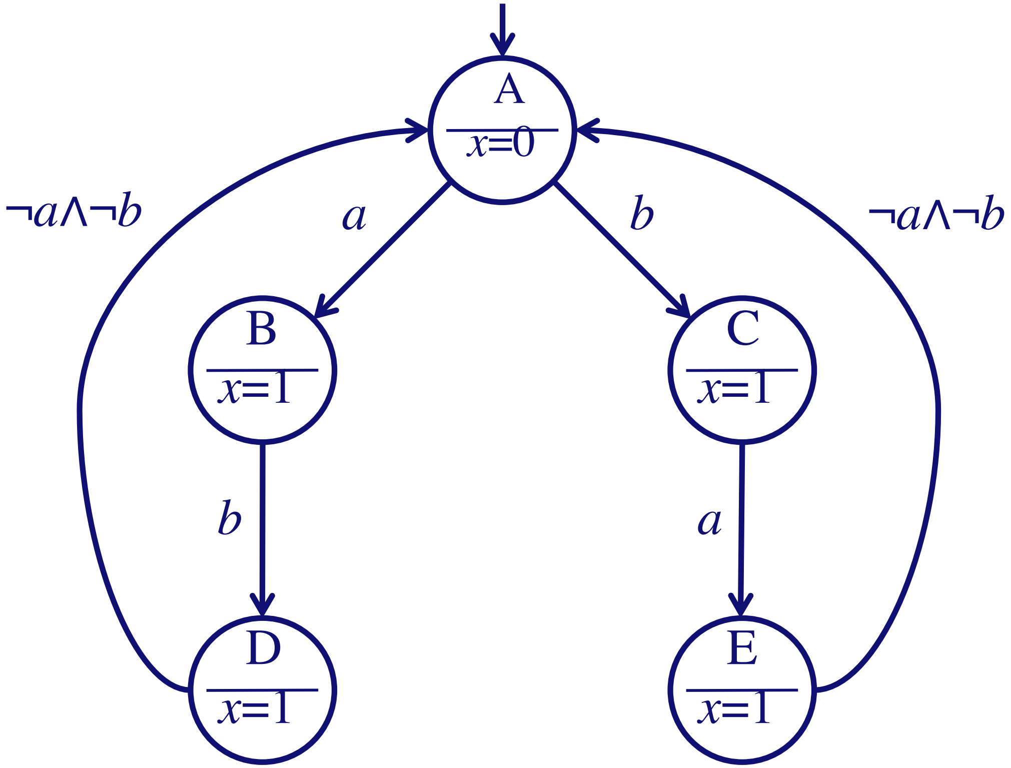
Example: minimizing an automaton (1)

Specification

- Two input variables a and b; one output x.
- Both inputs become 1 and then 0 again, in any order.
- The two inputs become 1 one after the other, so *not simultaneously*.
- If the *first* of the inputs becomes 1, the output x becomes 1, too.
- If *both* inputs becomes 0, the output x becomes 0, too.
- The *first* input to become 1 may even become 0 *before* the *second* one becomes 1: the circuit must be designed must “remember” this.

Example: minimizing an automaton (2)

State-transition diagram



Example: minimizing an automaton (3)

State Equivalence [*simple approach: better use bisimulation, but this is not in the scope of this course*]

Two states are equivalent iff:

- both states have the *same values* for the outputs, and:
- both states have the *same outgoing transitions*, with the *same guards*, and leading to the *same states*.

In short: from both states onwards the automaton exhibits the *same behaviour*.

Theorem

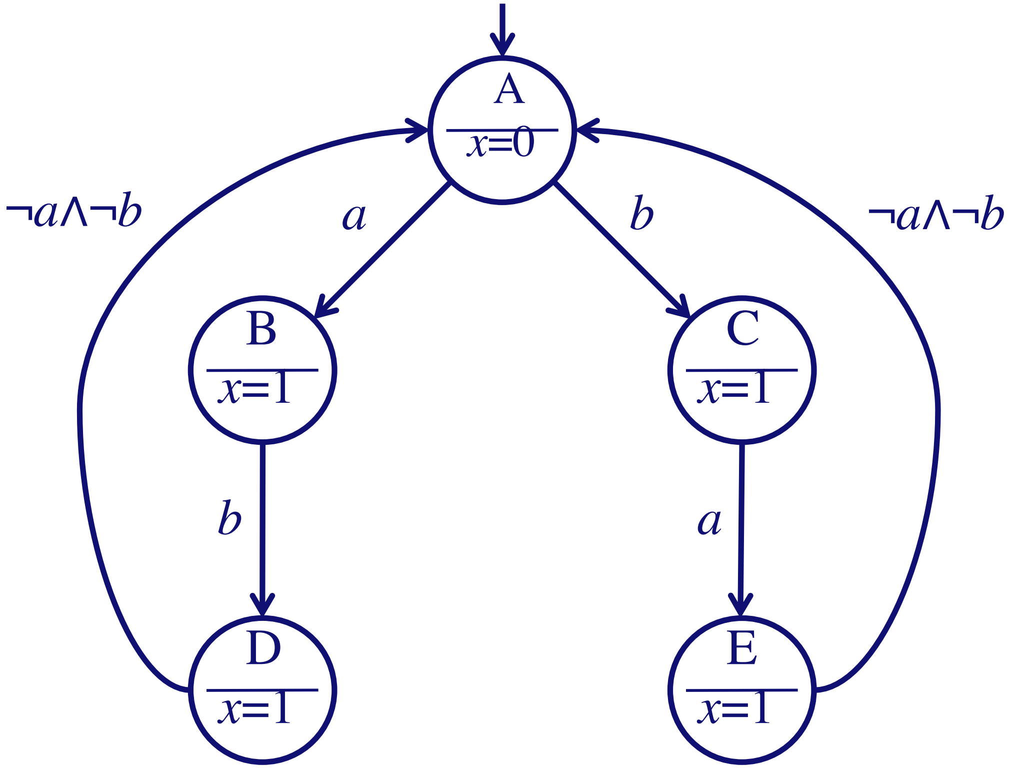
Equivalent states may be *combined* into a single state.

Application

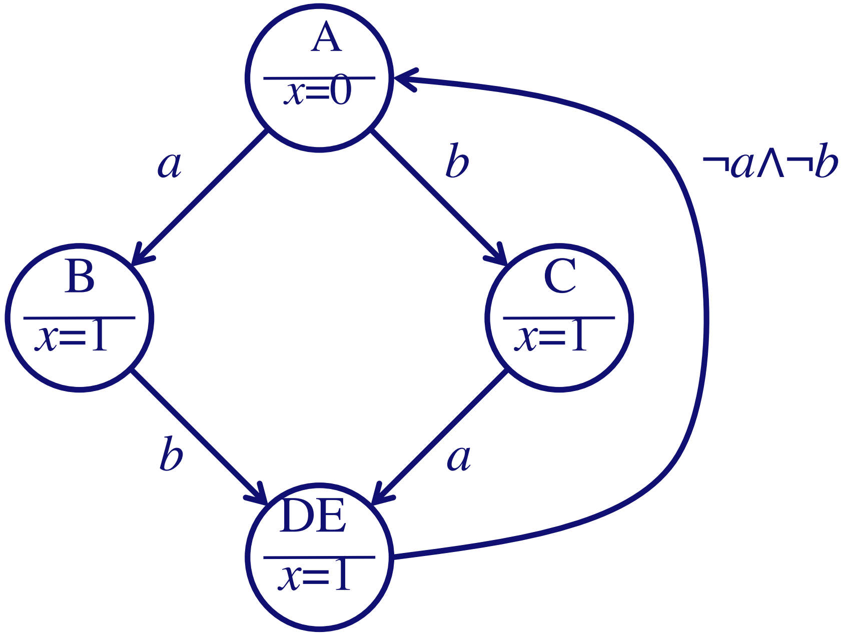
In our example: states D and E are equivalent, so they may be combined.

Example: minimizing an automaton (2)

State-transition diagram

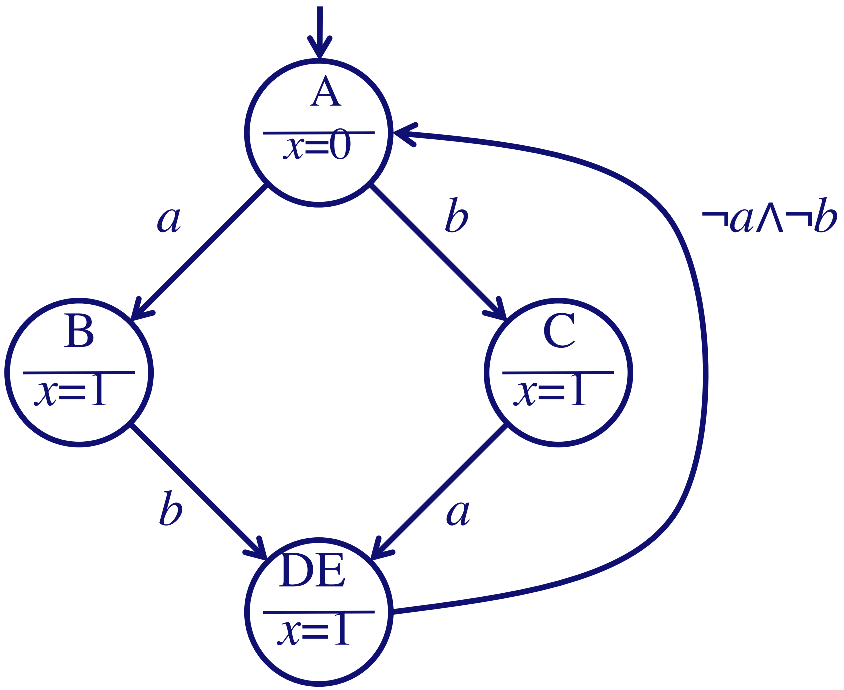


Minimized State-transition Diagram



Example: minimizing an automaton (4)

Minimized State-transition Diagram



state transitions

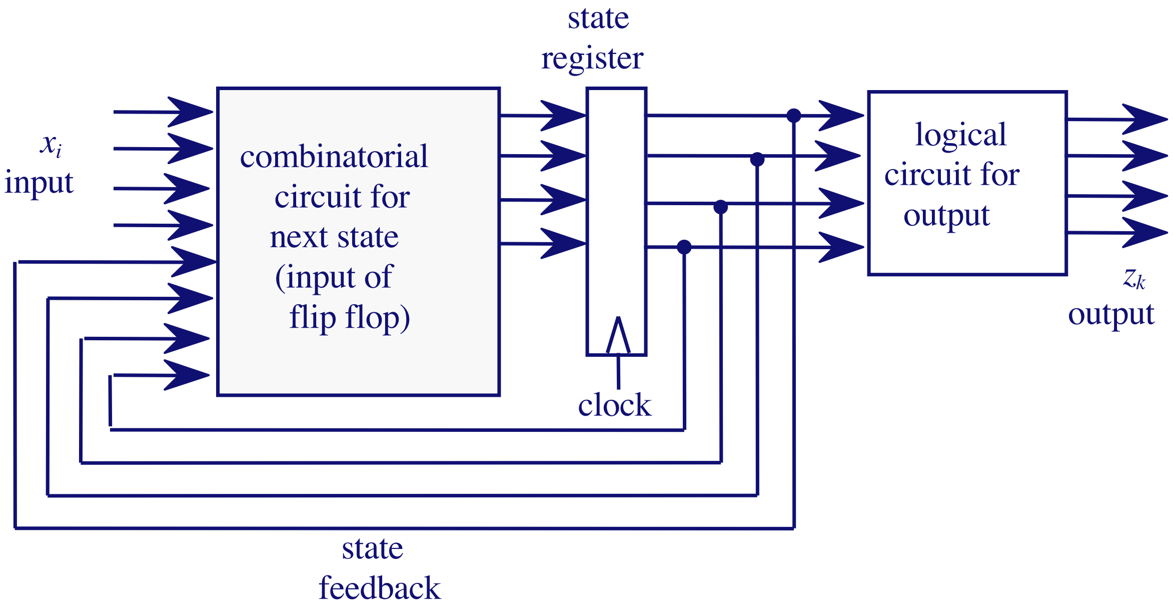
state	a	b	$\text{state}_{\text{new}}$	
A	0	0	A	
A	0	1	C	
A	1	0	B	
A	1	1	X (don't care)	
B	0	0	B	
B	0	1	DE	
B	1	0	B	
B	1	1	DE	X (don't care)
C	0	0	C	
C	0	1	C	
C	1	0	DE	
C	1	1	DE	X (don't care)
DE	0	0	A	
DE	0	1	DE	
DE	1	0	DE	
DE	1	1	DE	X (don't care)

Summary: Design Flow for Finite Automata

1. Analyse and understand the specification of the automaton (states and guarded transitions!).
2. Minimize the automaton (if possible).
3. Create an abstract specification of the automaton (state diagram / state transition table).
4. Assign (binary) codes to the states of the automaton.
5. Choose types of memory elements and implement the state register.
6. Design combinatorial circuits to compute the next-state and output functions (Karnaugh maps!).
7. Connect the state register and the combinatorial circuits.

Questions?

Moore and Mealy machines

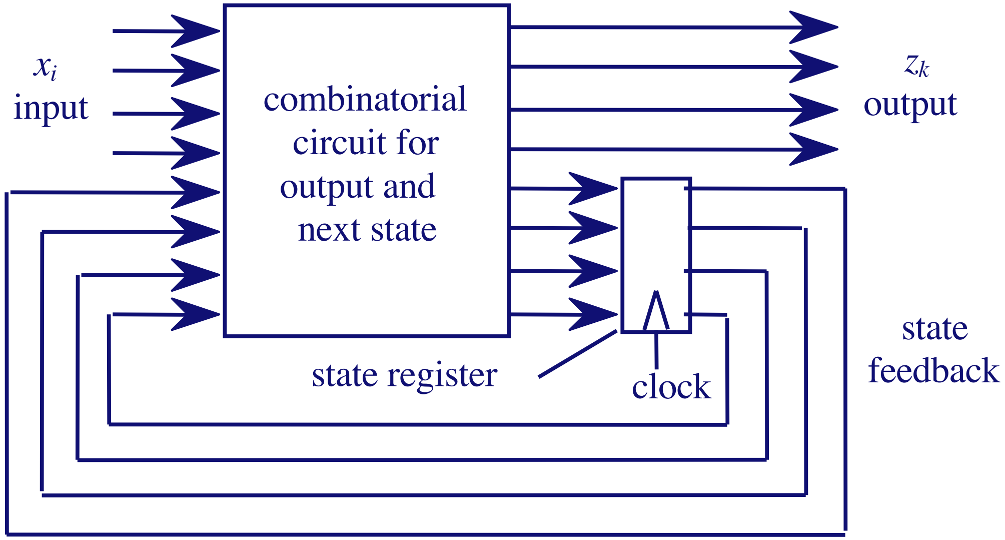


state register
x_i
input
combinatorial circuit for next state (input of flip flop)
clock
logical circuit for output
z_k
output
state feedback

Moore machine

output depends only on current state of the machine

output changes along with state



Mealy machine

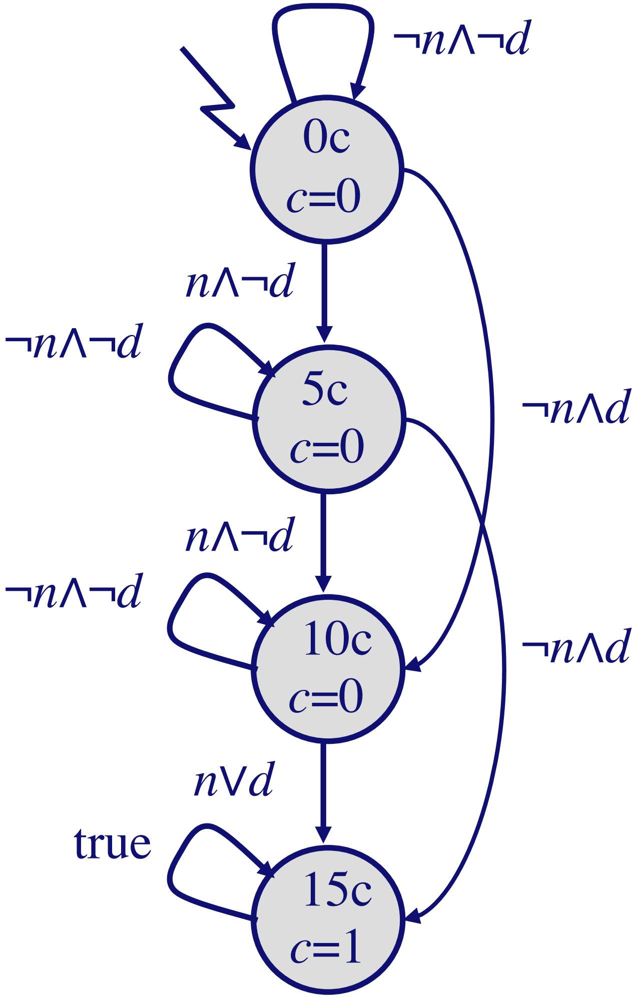
output depends on both current state and input

changes of input values have direct impact on the output value

asynchronous signals

Example: Coffee machine

Design a control circuit for a machine accepting 5c and 10c coins which outputs coffee after (at least) 15c has been paid.



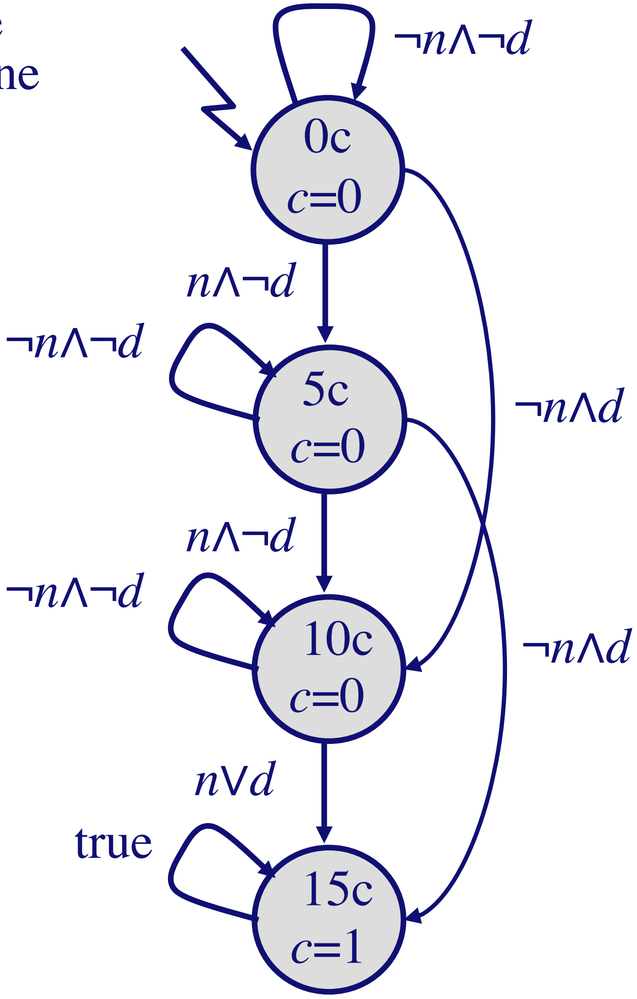
input coding:

n, d = 5c, 10c

$n \wedge d$ is an invalid input

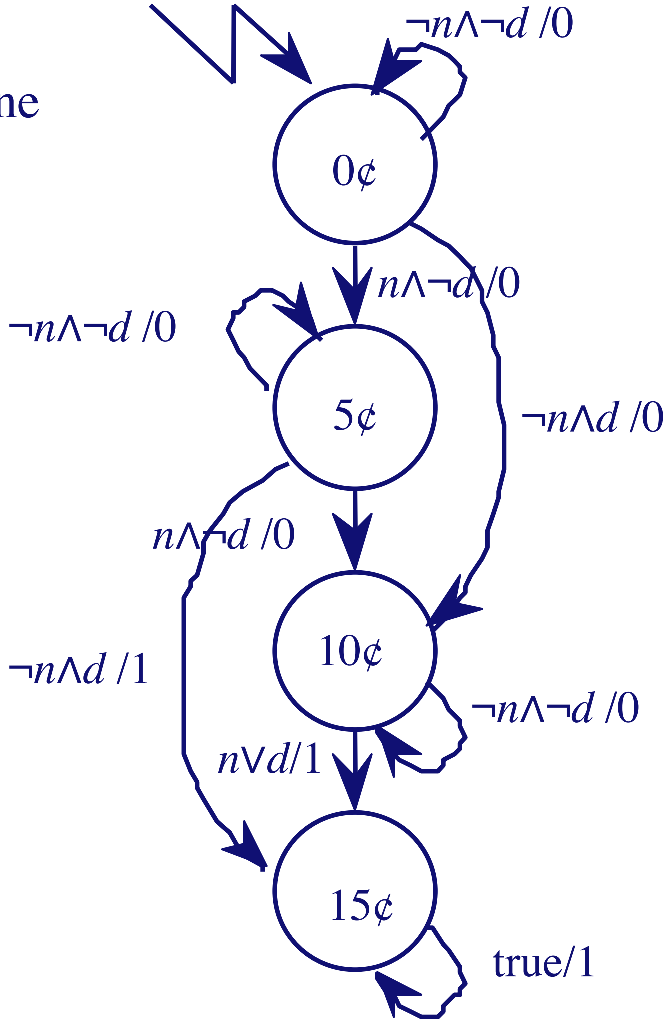
Coffee machine: Moore and Mealy machines

Moore
machine



output mentioned
in states

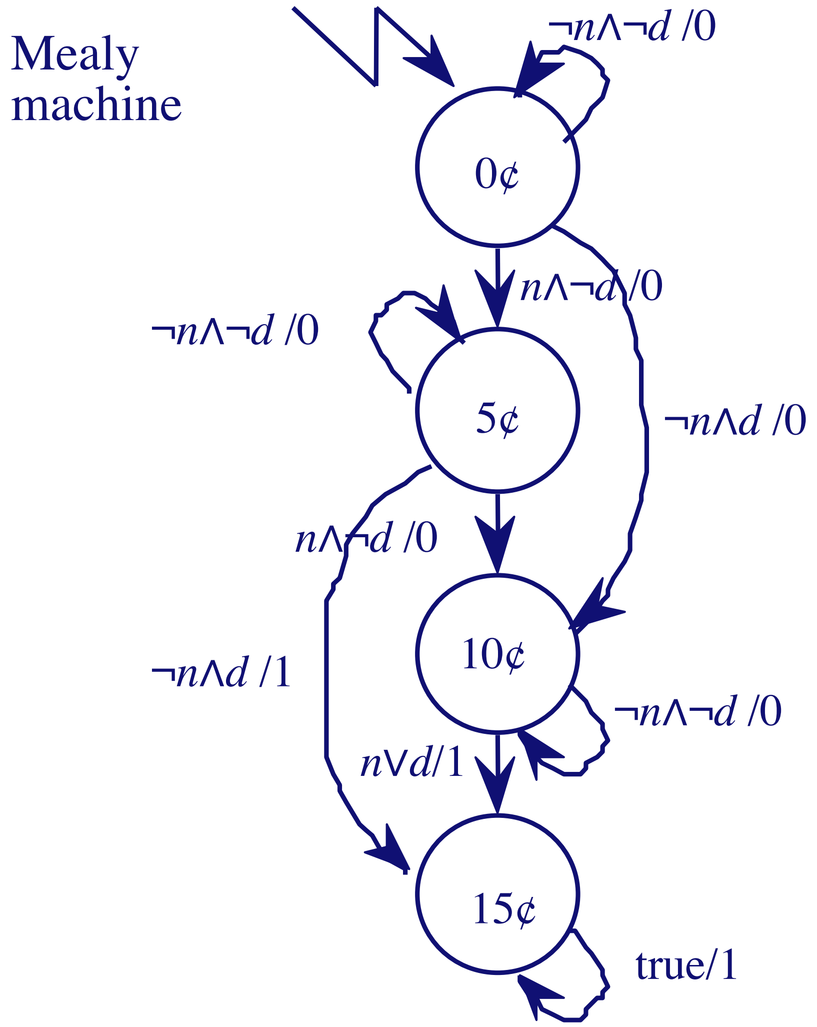
Mealy
machine



output mentioned
in transitions

Coffee machine (Mealy): transition table

PS	n	d	NS	c
0¢	0	0	0¢	0
	0	1	10¢	0
	1	0	5¢	0
	1	1	x	x
5¢	0	0	5¢	0
	0	1	15¢	1
	1	0	10¢	0
	1	1	x	x
10¢	0	0	10¢	0
	0	1	15¢	1
	1	0	15¢	1
	1	1	x	x
15¢	0	0	15¢	1
	0	1	15¢	1
	1	0	15¢	1
	1	1	x	x



Coffee machine (Mealy): truth tables

transition table

PS	*n*	*d*	NS	*c*
0¢	0	0	0¢	0
	0	1	10¢	0
	1	0	5¢	0
	1	1	x	x
5¢	0	0	5¢	0
	0	1	15¢	1
	1	0	10¢	0
	1	1	x	x
10¢	0	0	10¢	0
	0	1	15¢	1
	1	0	15¢	1
	1	1	x	x
15¢	0	0	15¢	1
	0	1	15¢	1
	1	0	15¢	1
	1	1	x	x

state encoding

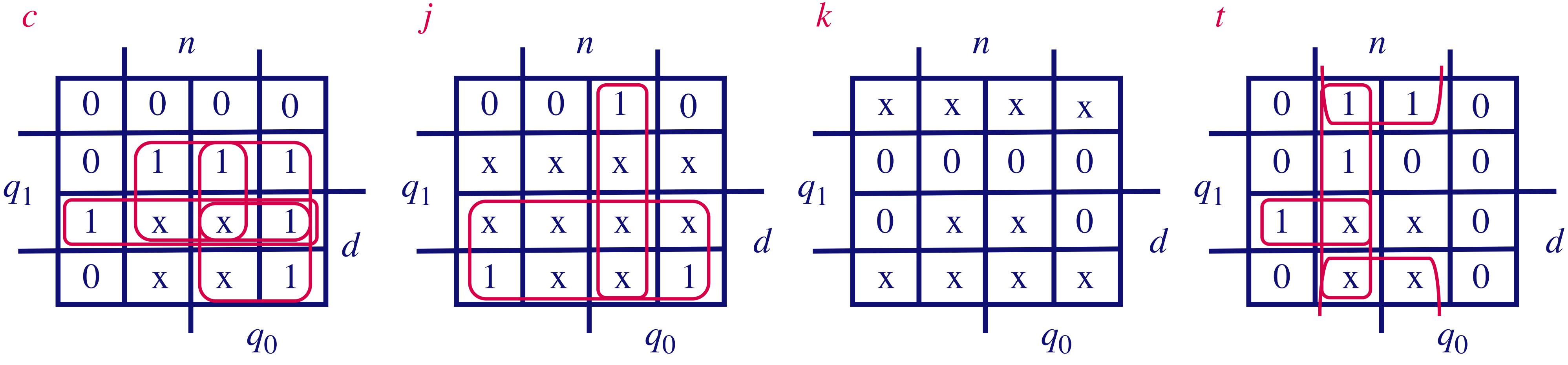
PS	$q_1\ q_0$
0¢	00
5¢	01
10¢	10
15¢	11

truth table

q_1q_0	*n*	*d*	$q_1^+q_0^+$	*c*	j k	t
0 0	0	0	0 0	0	0 x	0
	0	1	1 0	0	1 x	0
	1	0	0 1	0	0 x	1
	1	1	x x	x	x x	x
0 1	0	0	0 1	0	0 x	0
	0	1	1 1	1	1 x	0
	1	0	1 0	0	1 x	1
	1	1	x x	x	x x	x
1 0	0	0	1 0	0	x 0	0
	0	1	1 1	1	x 0	1
	1	0	1 1	1	x 0	1
	1	1	x x	x	x x	x
1 1	0	0	1 1	1	x 0	0
	0	1	1 1	1	x 0	0
	1	0	1 1	1	x 0	0
	1	1	x x	x	x x	x

implementation: q_1 with j-k flipflop (toggles when both inputs are 1)
q_0 with t flipflop (toggles when input is 1)

Coffee machine (Mealy): implementation



c

			n	n	
		0	0	0	0
q_1		0	1	1	1
q_1	d	1	x	x	1
	d	0	x	x	1
				q_0	q_0

j

			n	n	
		0	0	1	0
q_1		x	x	x	x
q_1	d	x	x	x	x
	d	1	x	x	1
				q_0	q_0

k

			n	n	
		x	x	x	x
q_1		0	0	0	0
q_1	d	0	x	x	0
	d	x	x	x	x
				q_0	q_0

t

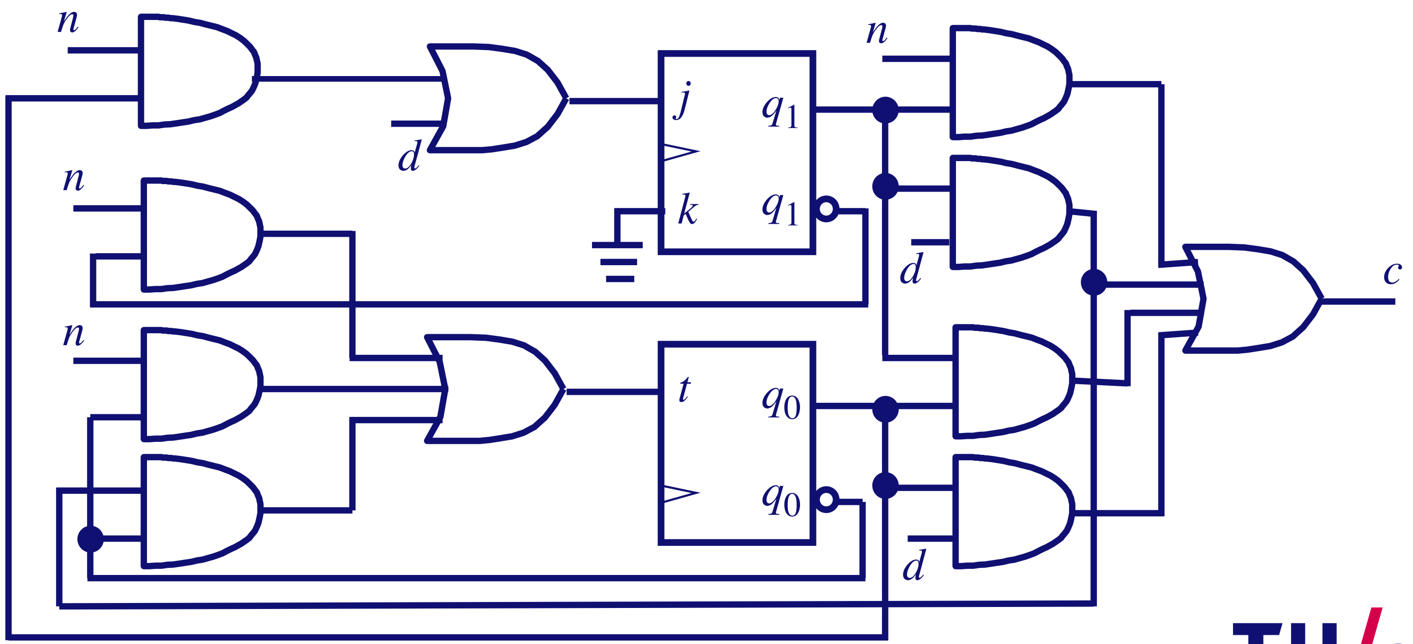
			n	n	
		0	1	1	0
q_1		0	1	0	0
q_1	d	1	x	x	0
	d	0	x	x	0
				q_0	q_0

$c = q_1 \wedge d \vee q_1 \wedge n \vee q_0 \wedge q_1 \vee q_0 \wedge d$

$j = d \vee q_0 \wedge n$

$k = 0$

$t = \neg q_1 \wedge n \vee \neg q_0 \wedge n \vee \neg q_0 \wedge q_1 \wedge d$



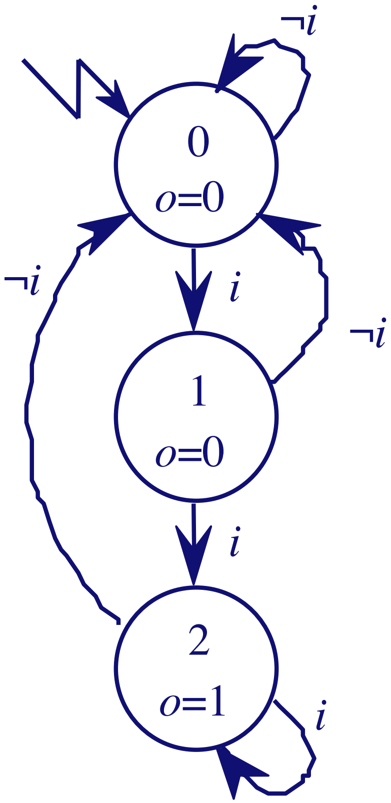
Moore versus Mealy machines

A Mealy machine usually has less states than a Moore machine given the same output (but more complicated timing).

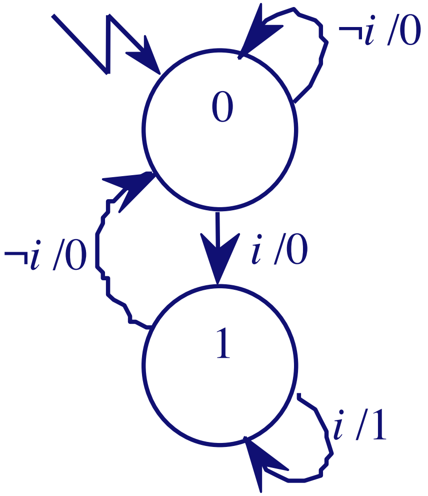
Example

Output o is 1 if the last two input values i were 1.

Moore

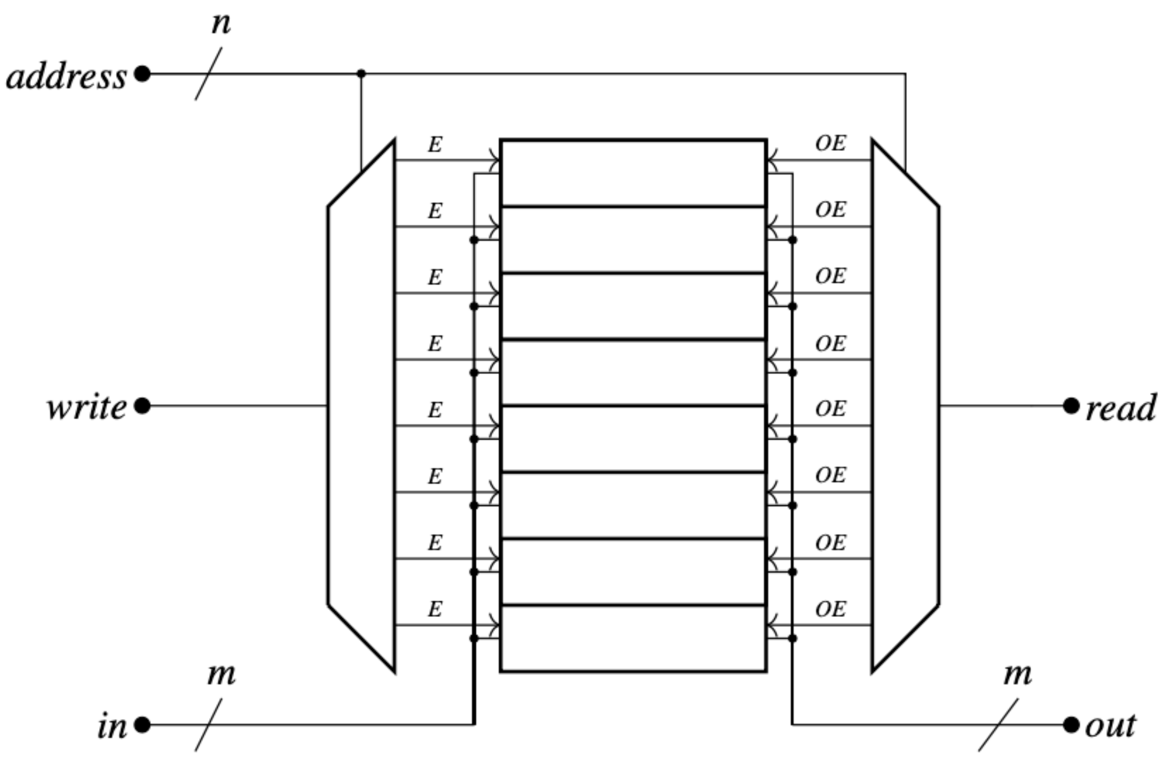


Mealy



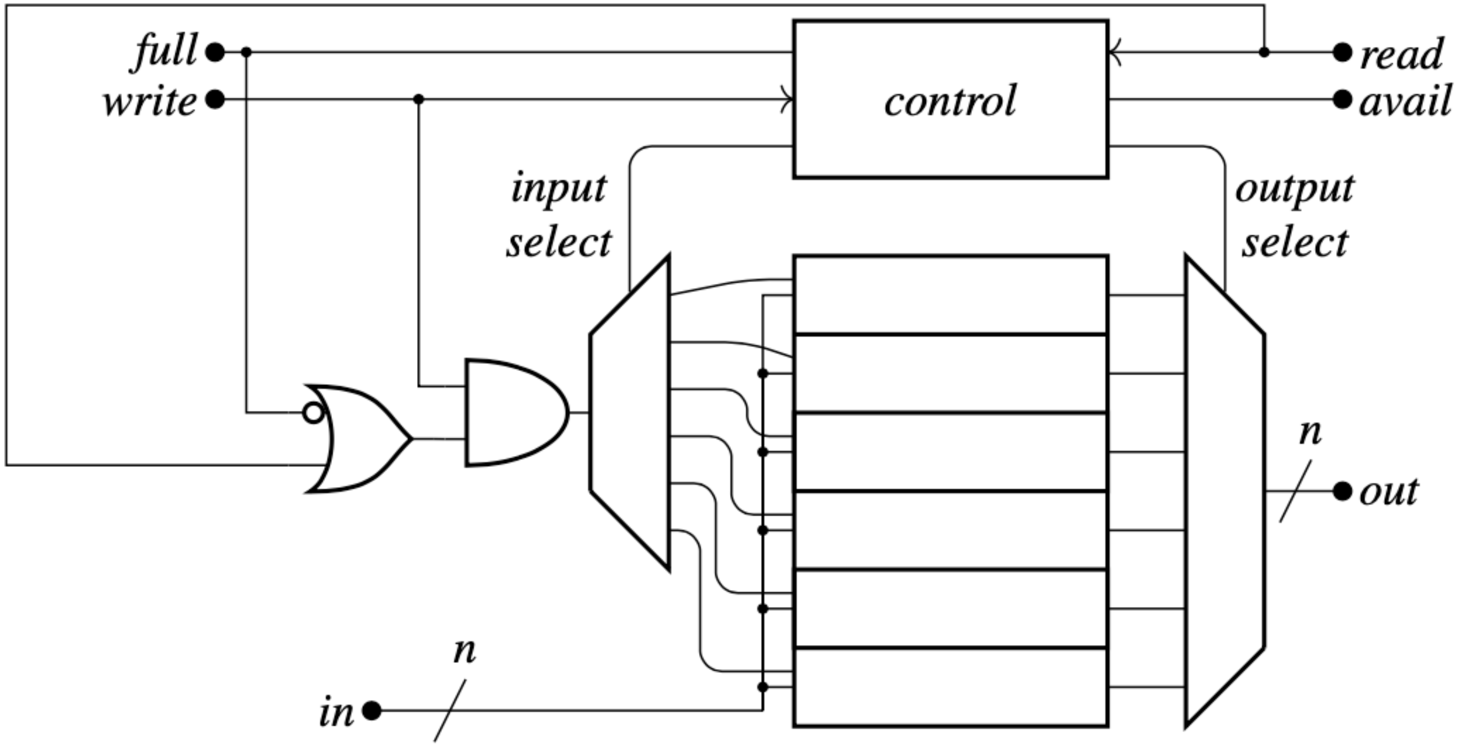
Questions?

Automata controlling registers.



How to transform m-bit memory of size 2^n into a hardware buffer?

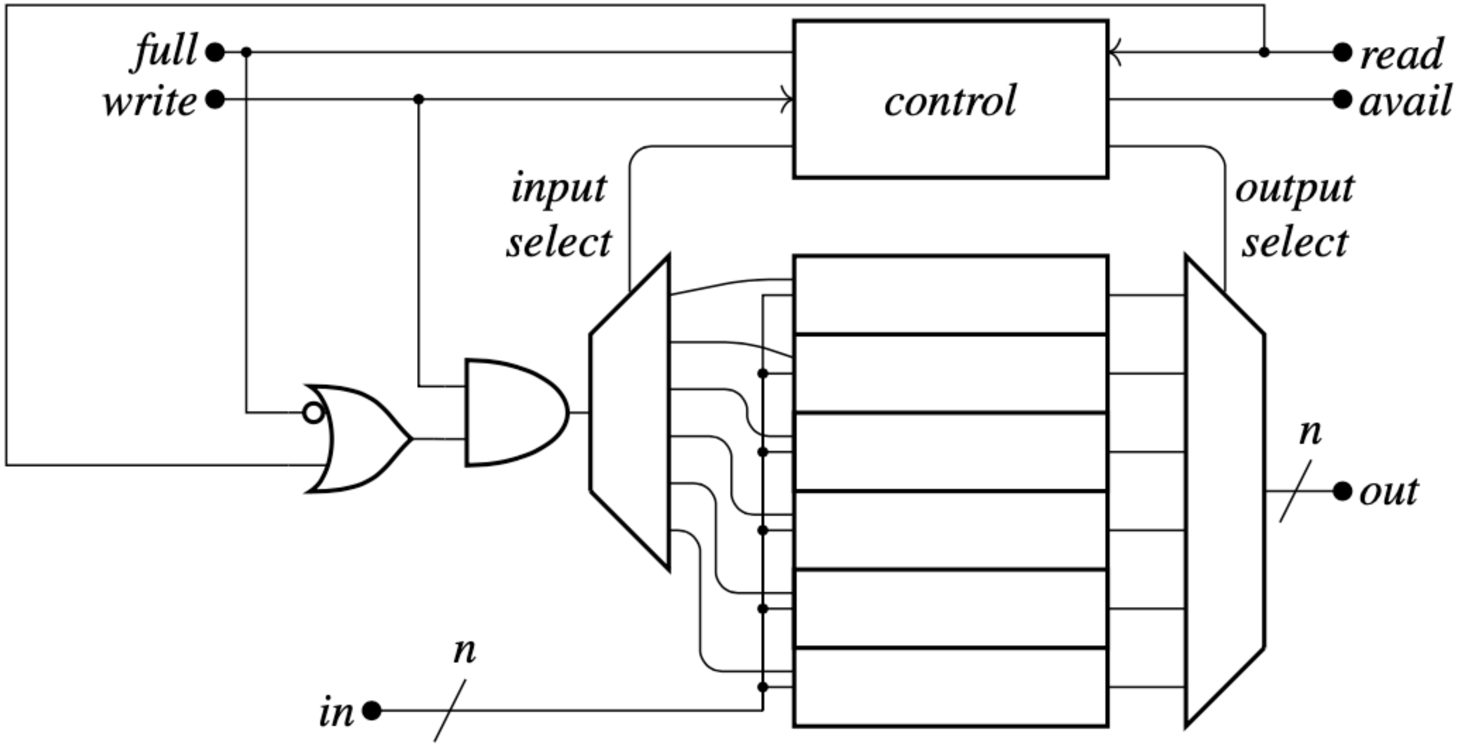
Automata controlling registers.



How to transform *m*-bit memory of size 2^n into a hardware buffer?

Add a finite state controller.

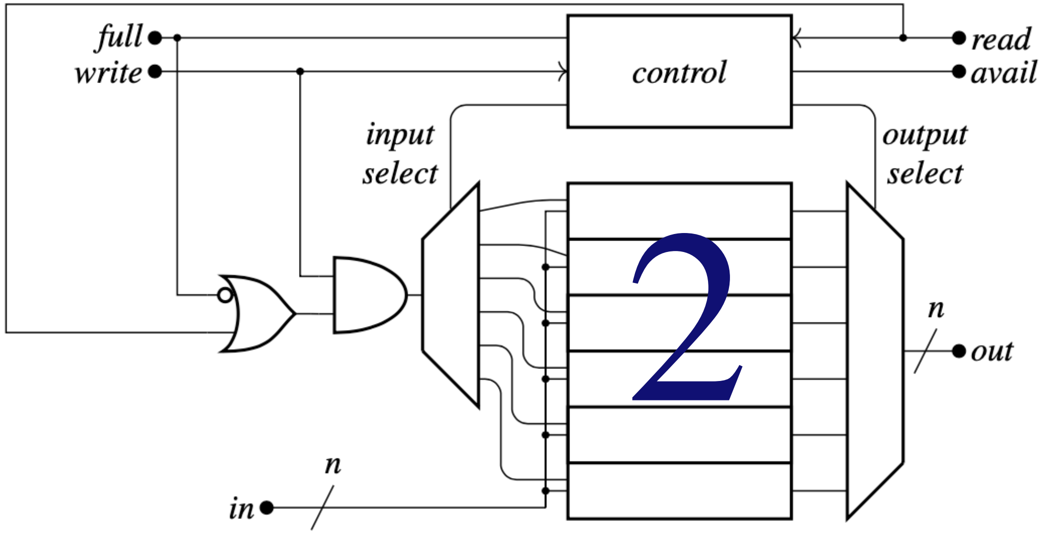
Automata controlling registers.

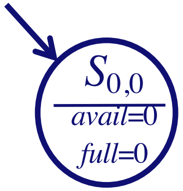


How to transform *m*-bit memory of size 2*n* into a hardware buffer?

Add a clocked finite state controller. Restrict to 2 registers.

Automata controlling registers.

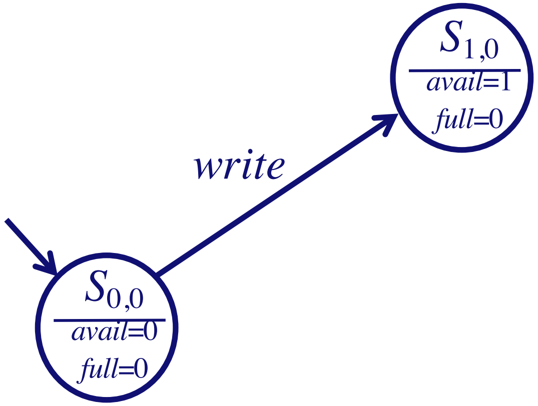


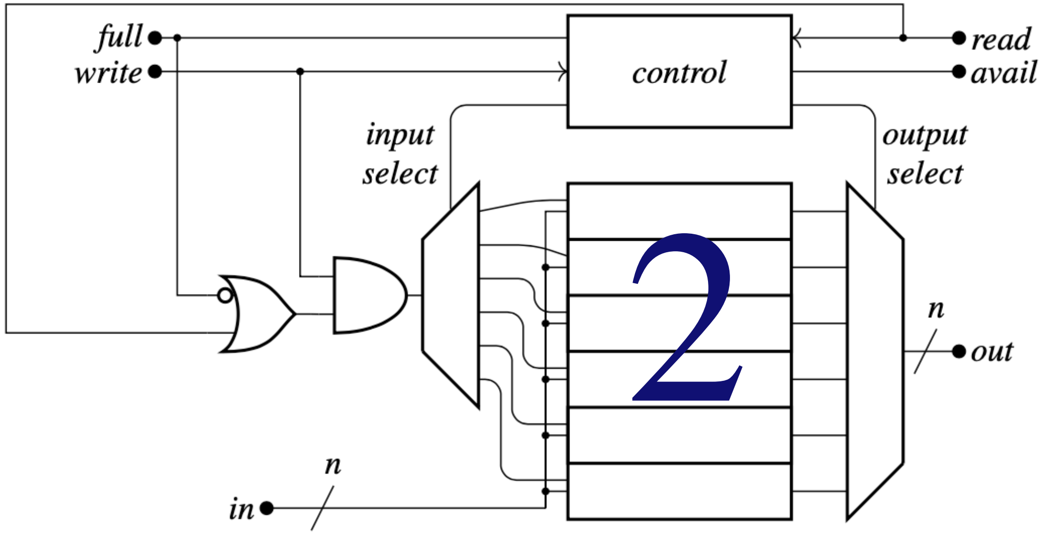


Restrict to 2 registers.

$S_{i,j}$: i registers are occupied, j is the next to be output.

Automata controlling registers.

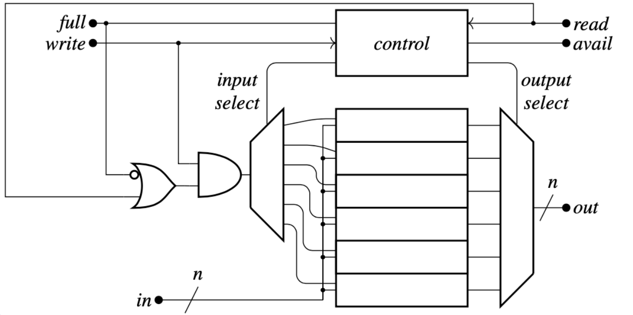


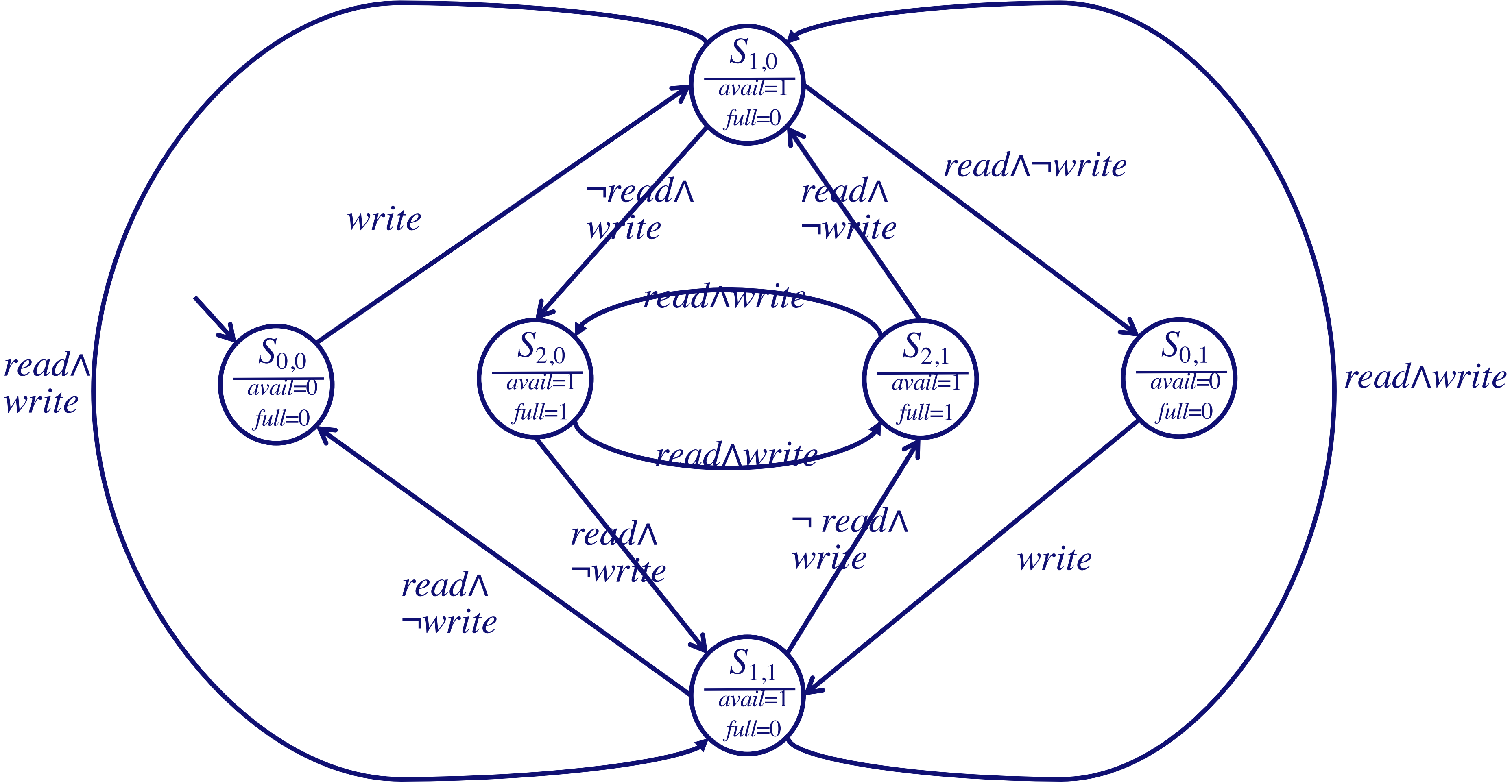


Restrict to 2 registers.

$S_{i,j}$: i registers are occupied, j is the next to be output.

Automata controlling registers.



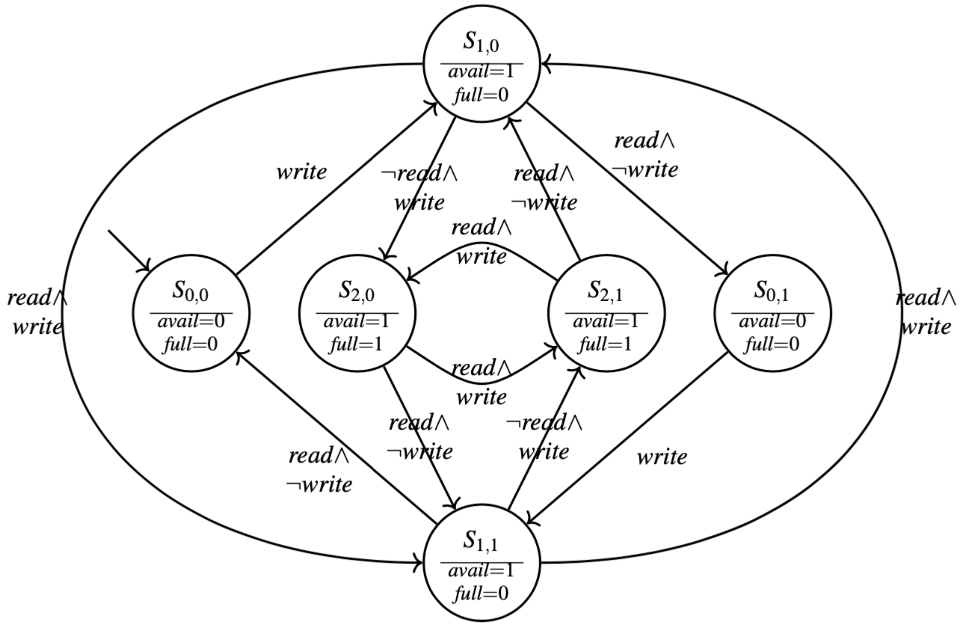


$S_{i,j}$: i registers are occupied, j is the next to be output.

Automata controlling registers.

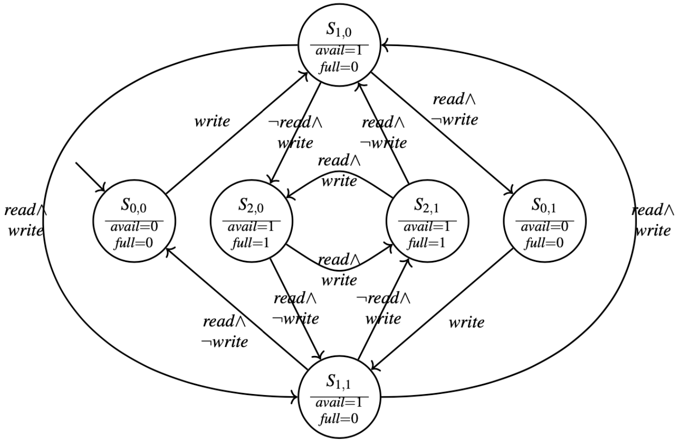
Output table.

	avail	*full*
$S_{0,0}$	0	0
$S_{1,0}$	1	0
$S_{0,1}$	0	0
$S_{1,1}$	1	0
$S_{2,0}$	1	1
$S_{2,1}$	1	1



Restrict to 2 registers.

Automata controlling registers.



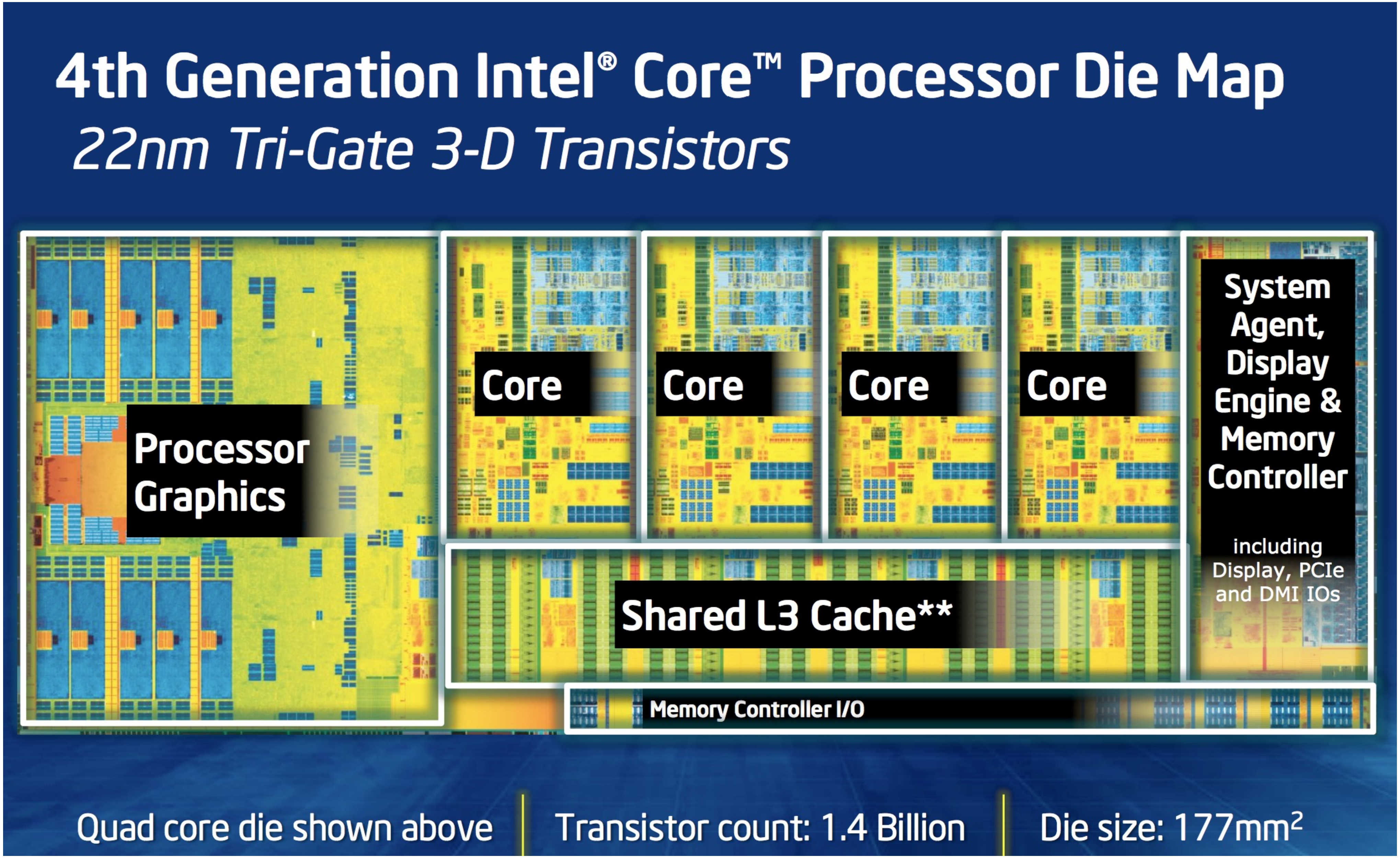
	read	*write*	
$S_{0,0}$	0	0	$S_{0,0}$
	0	1	$S_{1,0}$
	1	0	$S_{0,0}$
	1	1	$S_{1,0}$
$S_{1,0}$	0	0	$S_{1,0}$
	0	1	$S_{2,0}$
	1	0	$S_{0,1}$
	1	1	$S_{1,1}$
$S_{1,1}$	0	0	$S_{1,1}$
	0	1	$S_{2,1}$
	1	0	$S_{0,0}$
	1	1	$S_{1,0}$
$S_{0,1}$	0	0	$S_{0,1}$
	0	1	$S_{1,1}$
	1	0	$S_{0,1}$
	1	1	$S_{1,1}$

	read	*write*	
$S_{2,0}$	0	0	$S_{2,0}$
	0	1	$S_{2,1}$
	1	0	$S_{1,1}$
	1	1	$S_{1,0}$
$S_{2,1}$	0	0	$S_{2,1}$
	0	1	$S_{1,0}$
	1	0	$S_{2,1}$
	1	1	$S_{2,0}$

Restrict to 2 registers.

Questions?

Hardware on a chip.



4th Generation Intel® Core™ Processor Die Map

22nm Tri-Gate 3-D Transistors

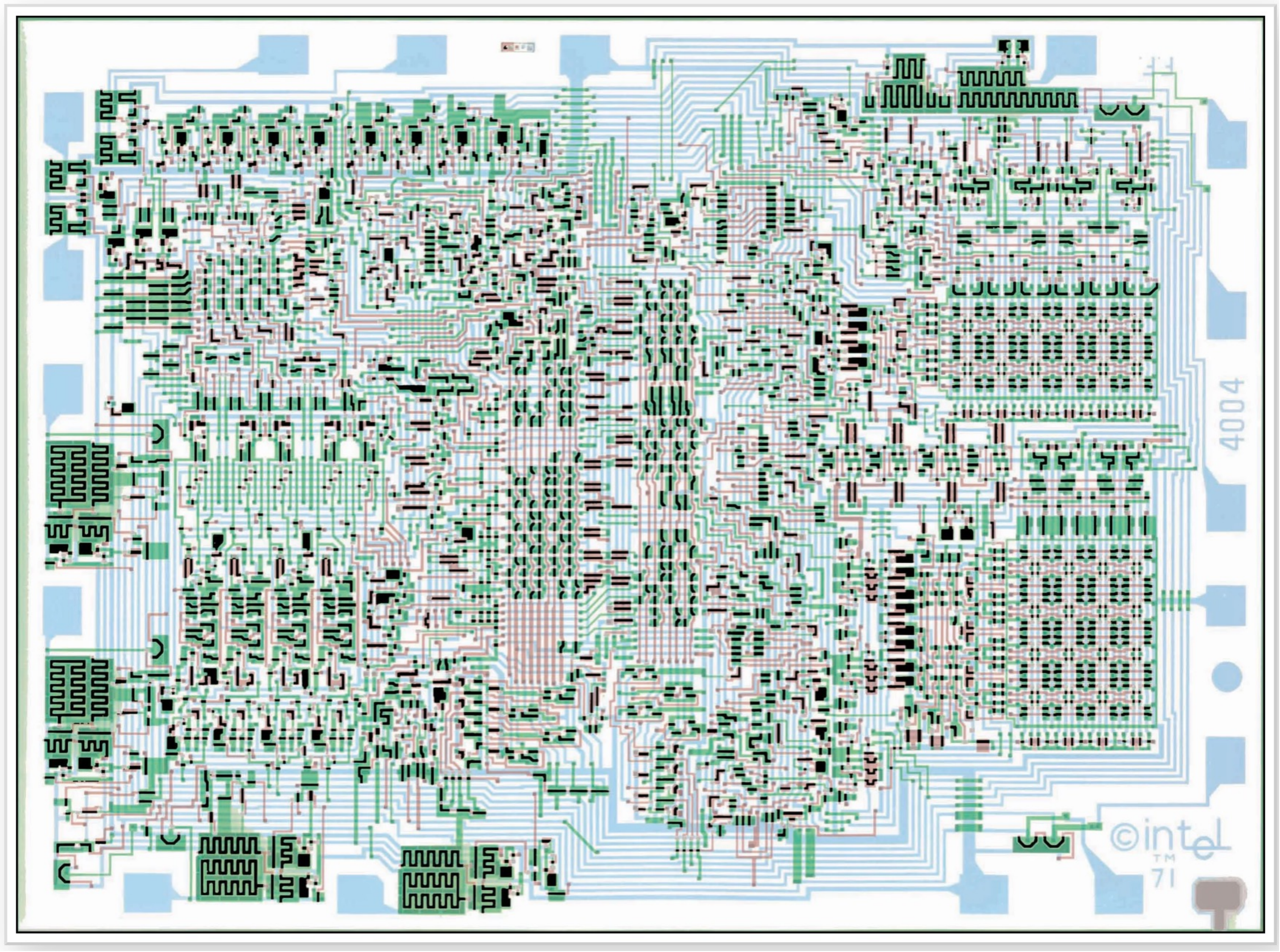
Processor Graphics
Core
Core
Core
Core
System Agent, Display Engine & Memory Controller
including Display, PCIe and DMI IOs
Shared L3 Cache**
Memory Controller I/O

Quad core die shown above | Transistor count: 1.4 Billion | Die size: 177mm^2

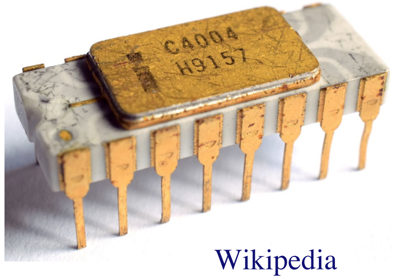
VHDL (VHSIC Hardware Description Language)

```
process
begin
  wait until START = '1'; -- wait until START is high

  for i in 1 to 10 loop -- then wait for a few clock periods...
    wait until rising_edge(CLK);
  end loop;

  for i in 1 to 10 loop       -- write numbers 1 to 10 to DATA,
                              -- 1 every cycle
    DATA <= to_unsigned(i, 8);
    wait until rising_edge(CLK);
  end loop;

  -- wait until the output changes
  wait on RESULT;

  -- now raise ACK for clock period
  ACK <= '1';
  wait until rising_edge(CLK);
  ACK <= '0';

  -- and so on...
end process;
```

The first microprocessor: Intel 4004.



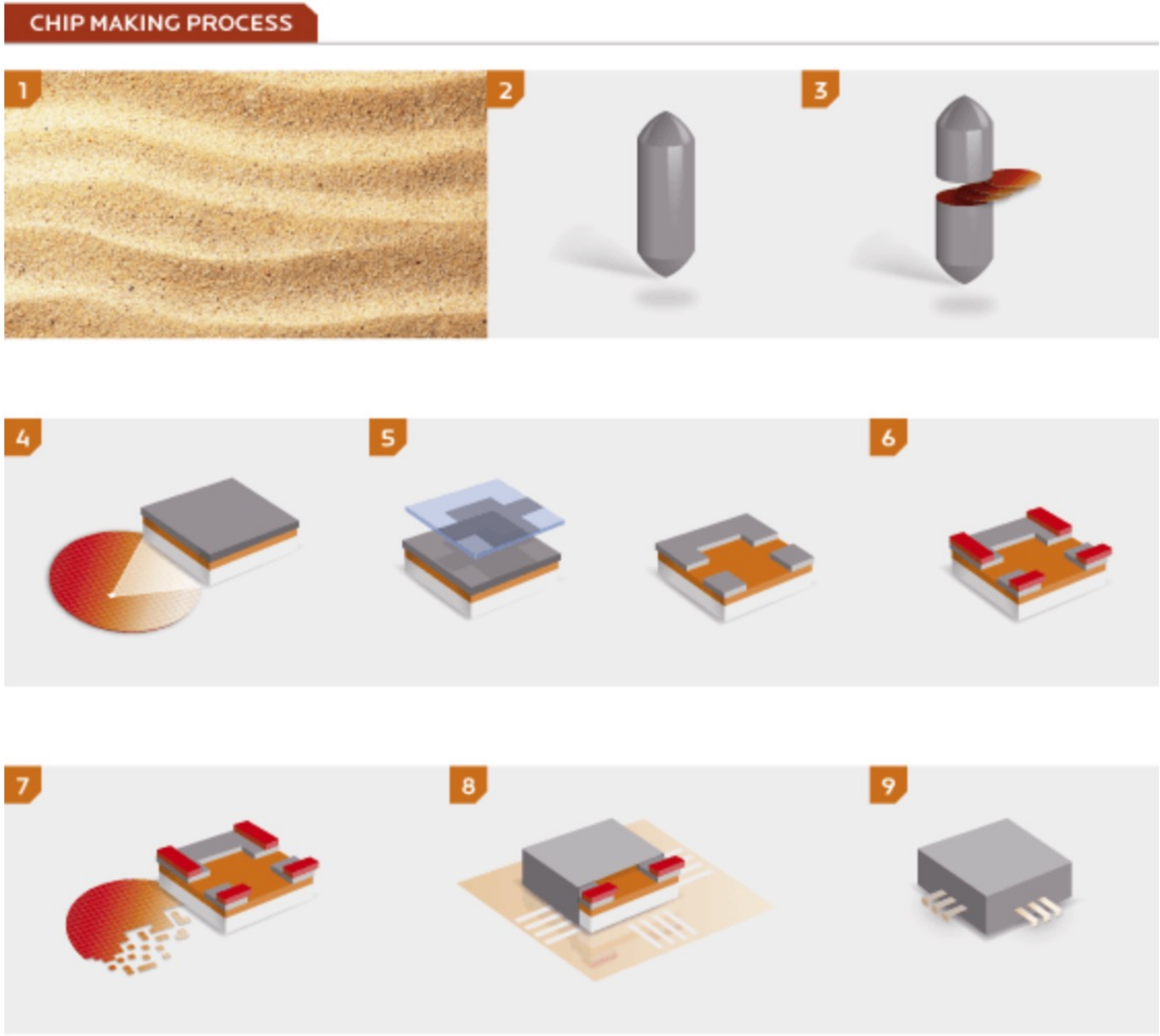
Picture: rickrutt.wordpress.com



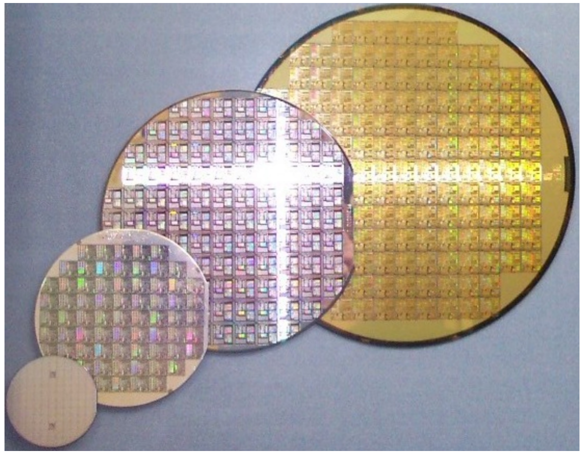
Wikipedia

4-bit computer
2300 transistors

Chip making process



picture taken from the ASM website



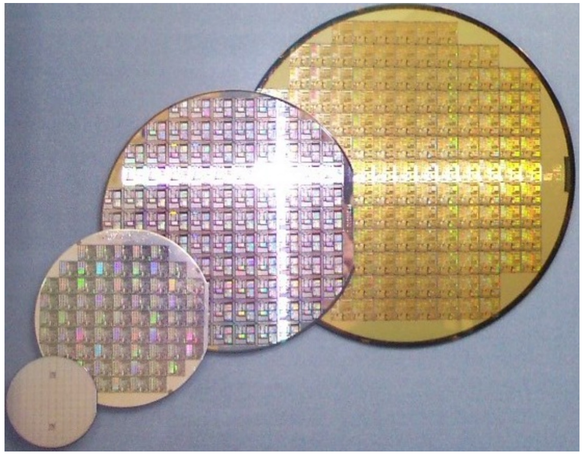
Source: Wikipedia

Project image on a wafer: market leader ASML.

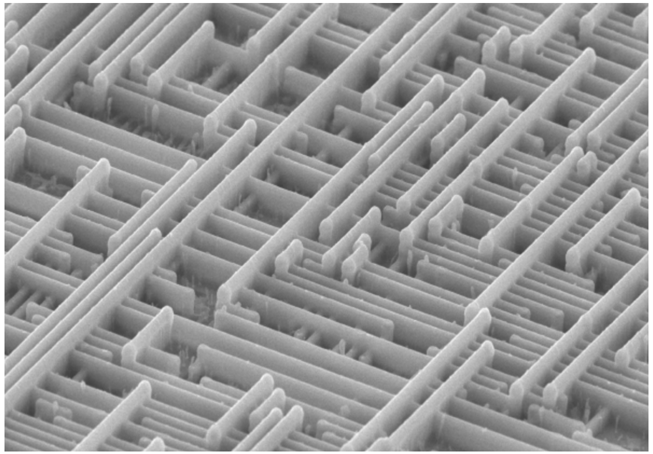
ASML
PAS 5500/400

ASML

Printing of the Netherlands.



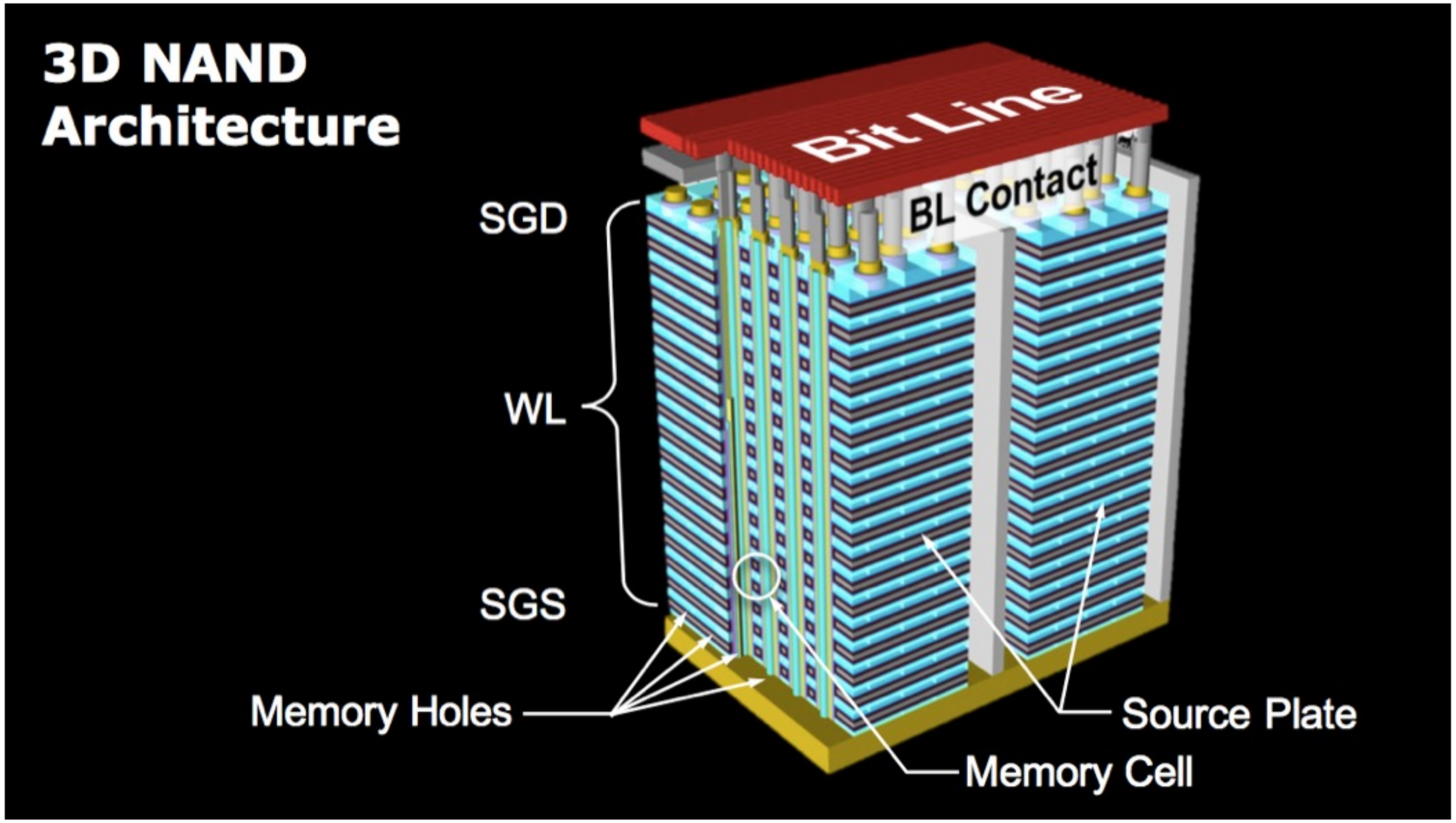
Picture: Wikipedia



Picture: www.ial-fa.com

Picture: Wikipedia

Chips are becoming 3-dimensional



Source: Western digital

Summary

What did you learn:

- Specify machines with memory as state machines.
- Implement state machines using flip-flops and combinatorial circuit.
- Moore vs Mealy machines.
- State machines can control systems with registers.
- Hardware description languages and the production of wafers.