P2 Digital Electronics

Lecture 7: State machines and sequential logic

Mark Cannon

mark.cannon@eng.ox.ac.uk

Trinity Term 2026

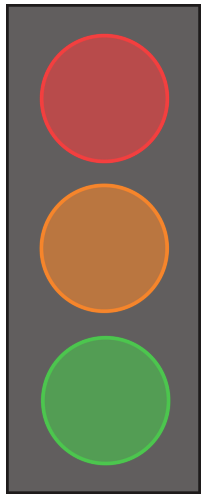
Overview of lectures

1. Logical functions and logic gates
2. Low level logic design
3. Binary number representation
4. Binary arithmetic
5. Integration of digital logic components
6. Memory and sequential circuits
7. **Design of sequential logic**
8. Data converters: analogue to digital / digital to analogue

Please send feedback, comments and corrections to mark.cannon@eng.ox.ac.uk

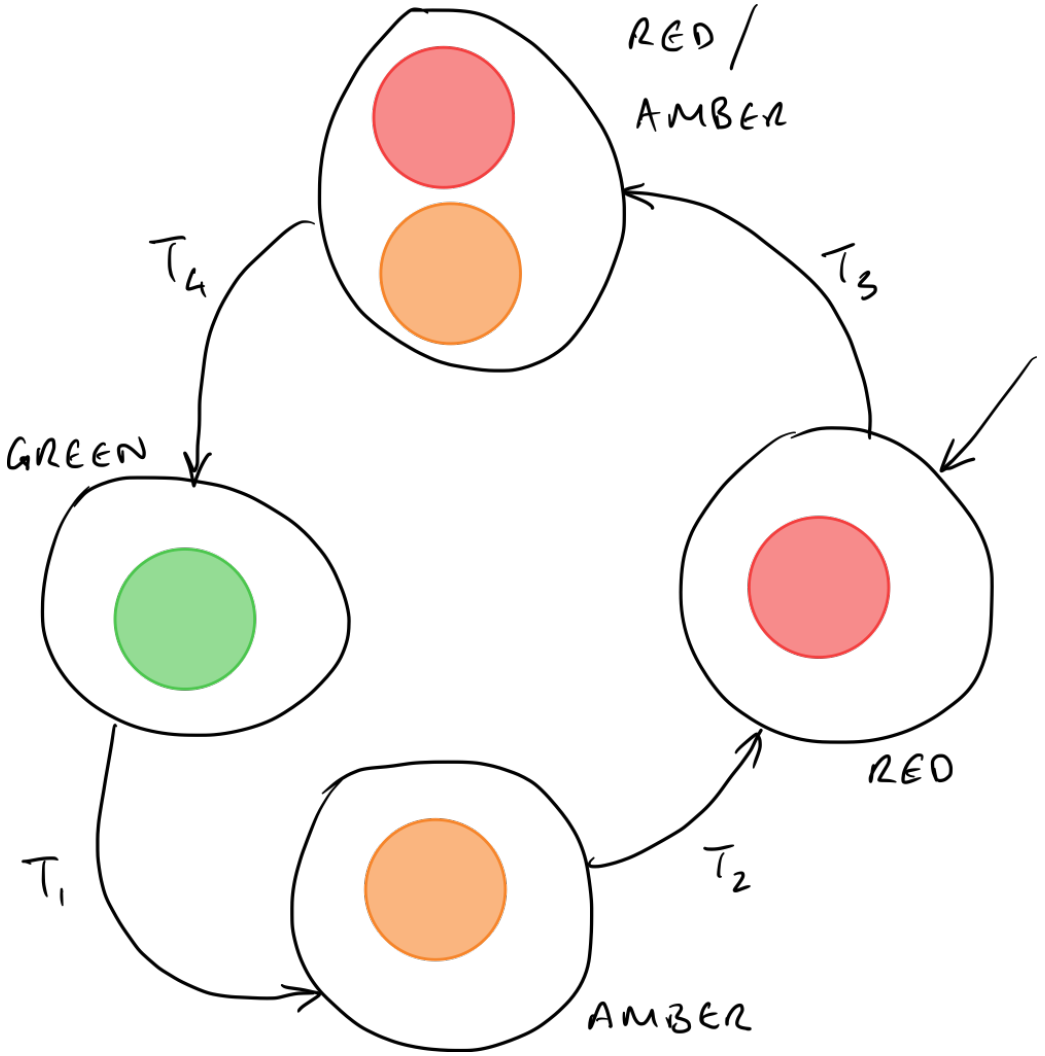
Finite state machine: a simple example

Traffic lights



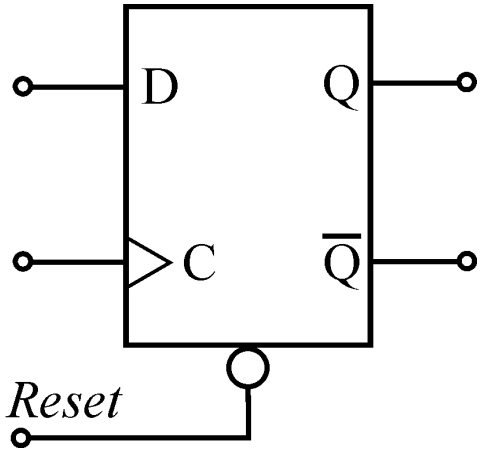
- Here the state and output are the same; in general they can be different
- The next state depends on the current state and the input – we need to store the current state to compute the next state (sequential logic)

State	Input	Next state
green	T_1	amber
amber	T_2	red
red	T_3	red/amber
red/amber	T_4	red



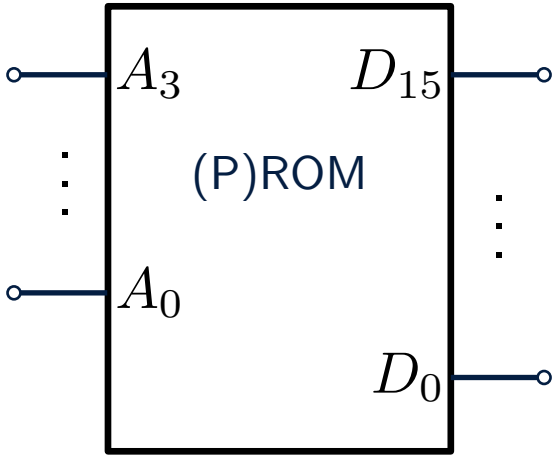
Finite state machine: implementation using digital electronics

Building blocks: D-type flip-flops



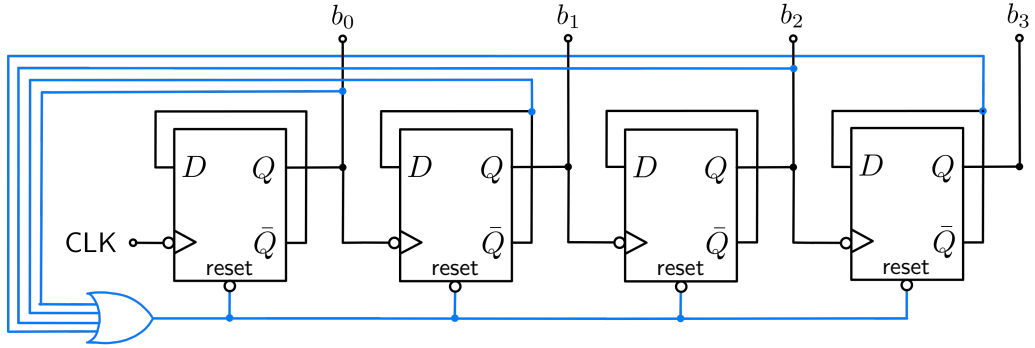
D	C	Reset	Q_{n+1}
X	0	1	Q_n
X	1	1	Q_n
0	↑	1	0
1	↑	1	1
X	X	0	0

and ROMs



... and digital logic

The digital state

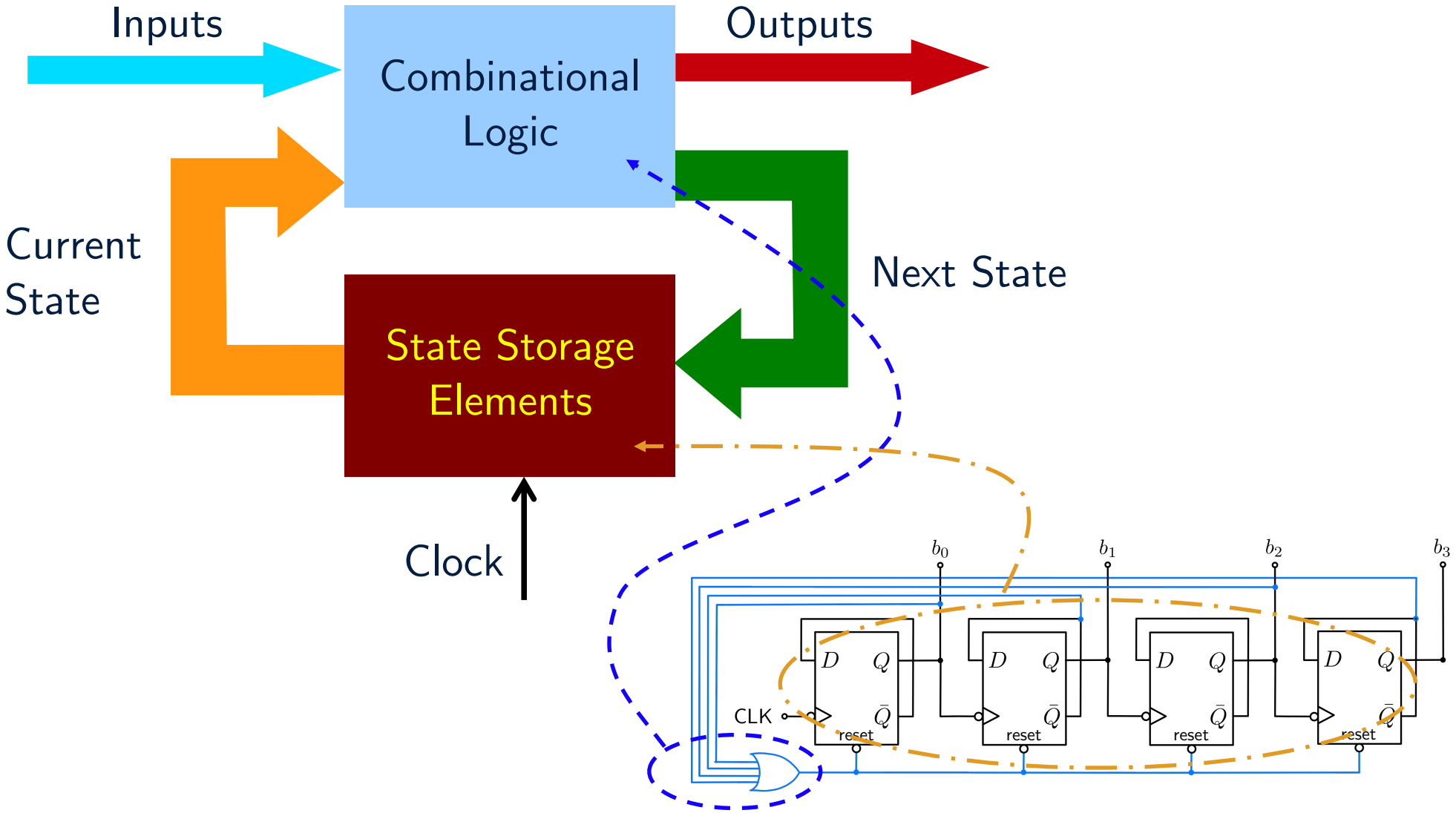


- The counter is a sequential logic device

  It moves through a finite sequence of states, e.g. for a modulo-10 counter: $0, 1, 2, \ldots, 8, 9, 0, 1, 2, \ldots, 8, 9, 0, 1, 2, \ldots$

- The state is defined by a unique combination of the bits of a logic system: *state-defining bits*
- State-defining bits can be outputs, but may also include other bits in order to keep track of events: *state-tracking bits*
- The next state is controlled by the inputs to the flip-flops, while in this circuit current state is determined by their outputs
- Asynchronous feedback can be used, but causes glitches, e.g. counter briefly occupies state 10 – synchronise state changes with the clock to prevent this

Generalised finite state machine

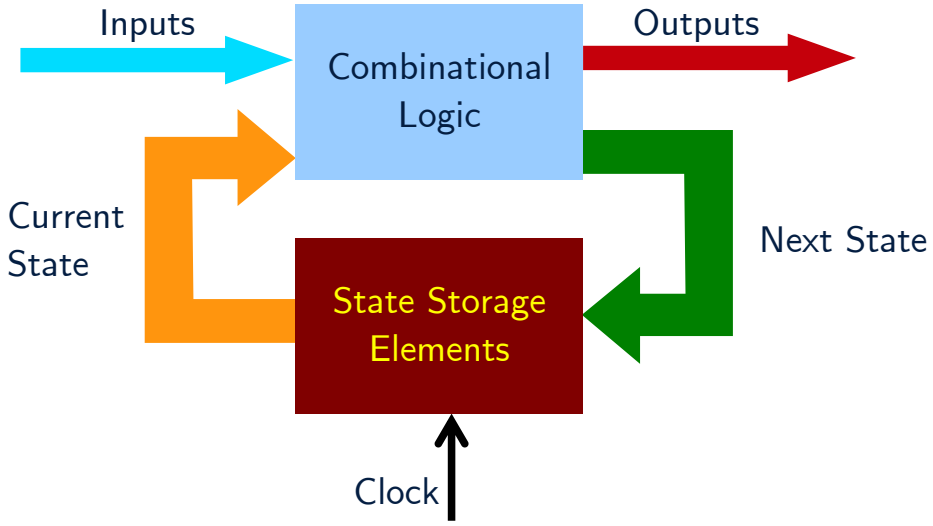


State storage in a register

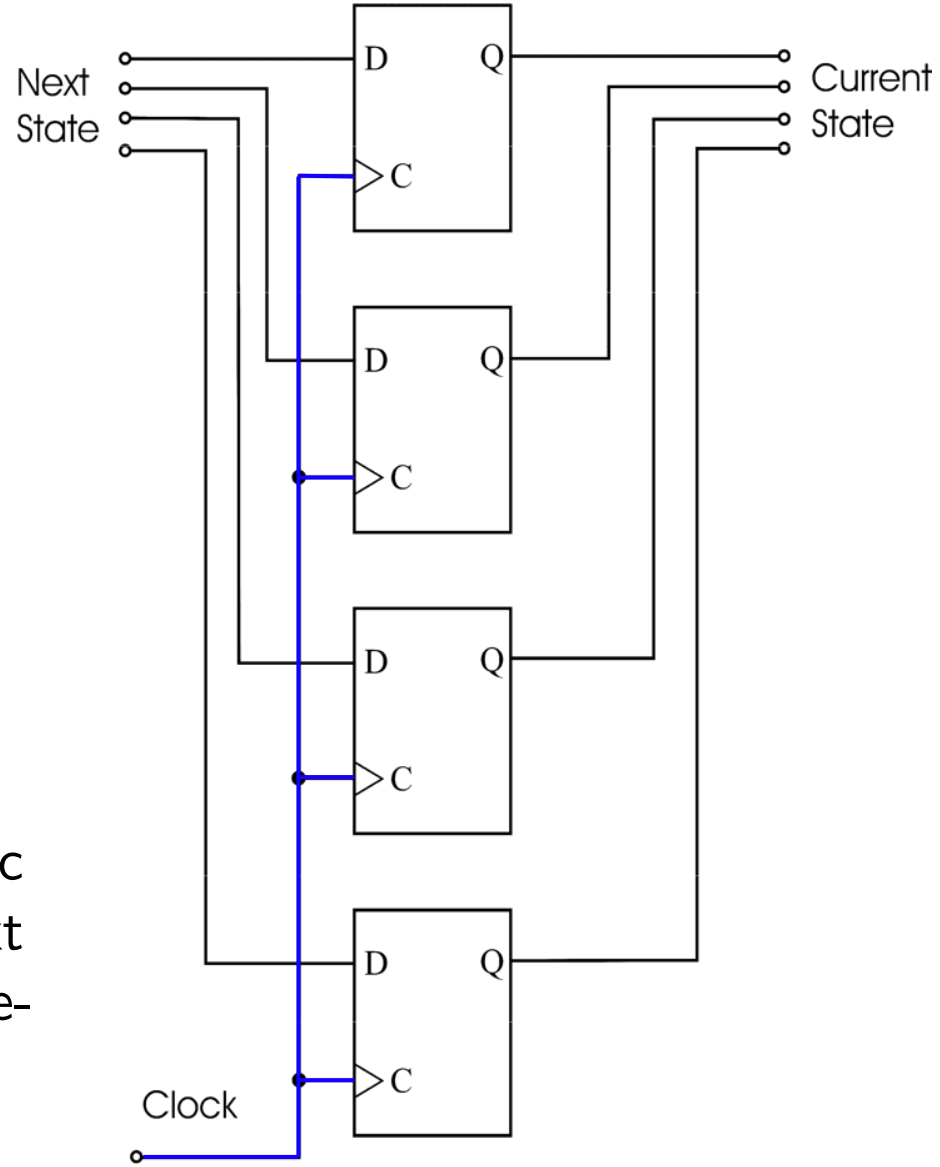
Array of D-type flip-flops forms a register to store the state

The next state is determined by the combination of the inputs and current state

A clock pulse common to all flip-flops transfers the next state though D-types to current state



Combinational logic determines the next state: how do we design this?



4 FFs can represent 2^4 discrete states

Sequential logic design

Synchronous counters

- Generating transition tables
- Using registers
- Sequencers

Synchronous D-type designs

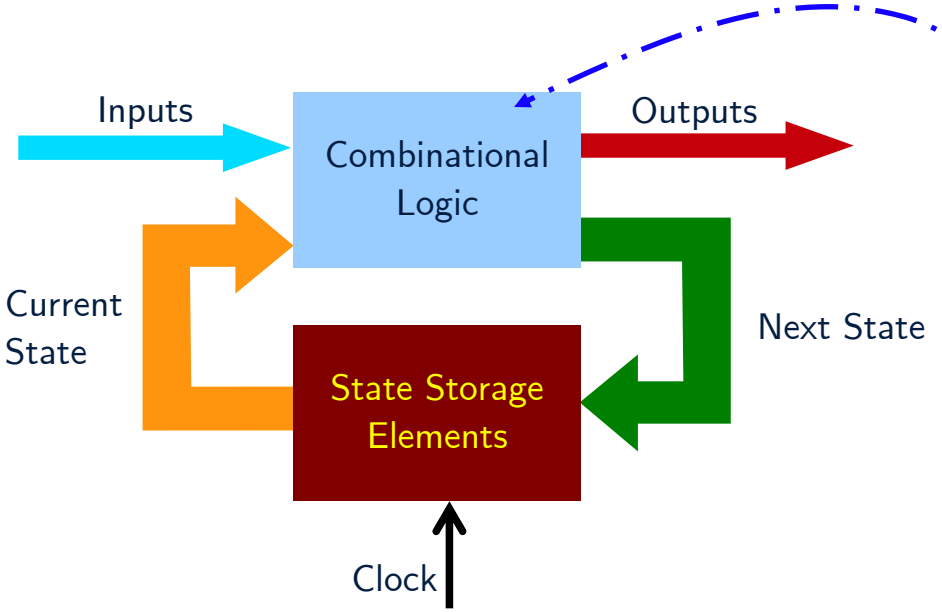
- Steering tables

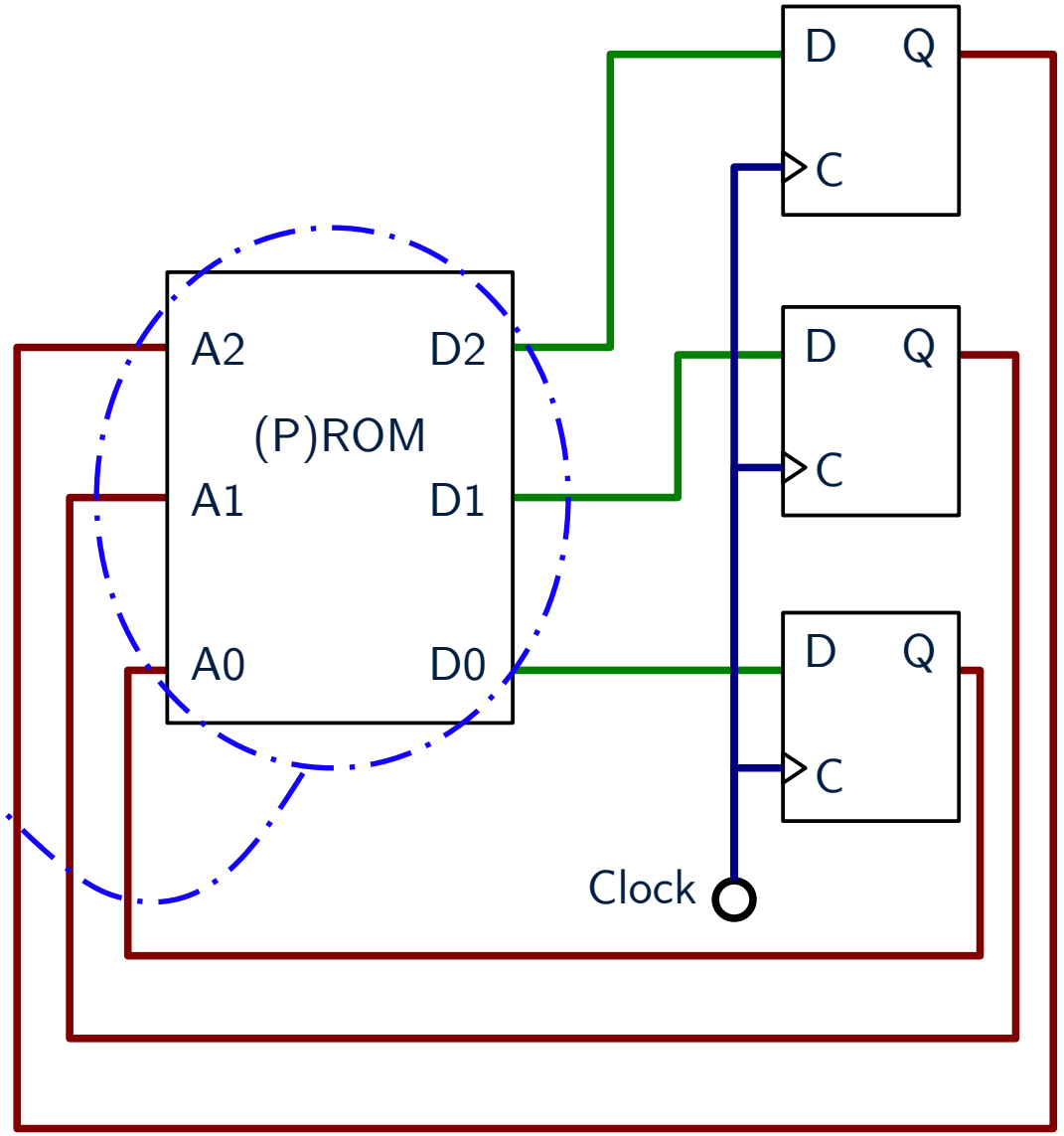
ROM-based counters

Program ROM with required truth tables

Address lines read current state

The content of the ROM address provides required Next State bits (inputs to FFs)





Programming the ROM

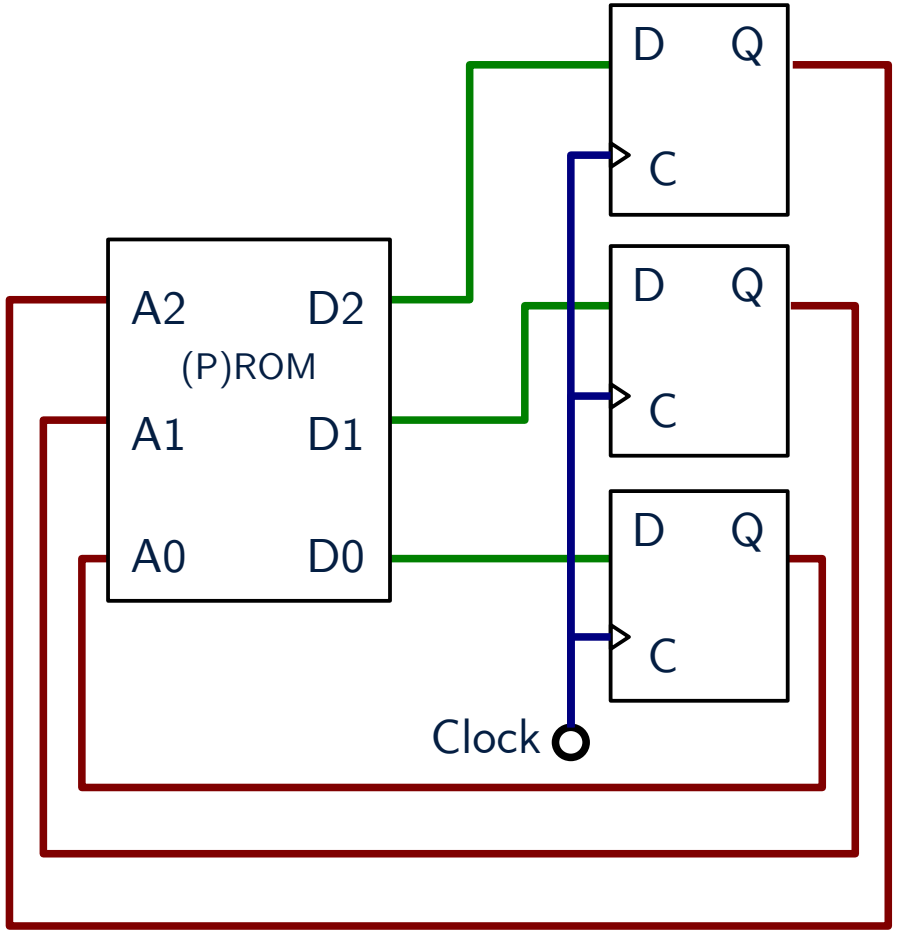
Example: Modulo 6 counter

#	Current state	Next state
0	000	001
1	001	010
2	010	011
3	011	100
4	100	101
5	101	000

$\Leftrightarrow$

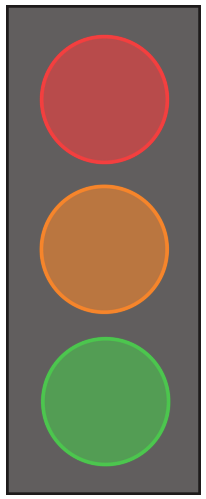
ROM address $A_2\,A_1\,A_0$	Data stored $D_2\,D_1\,D_0$
000	001
001	010
010	011
011	100
100	101
101	000
110	XXX
111	XXX

Could store 000 in these locations to ensure sequence entry on random start

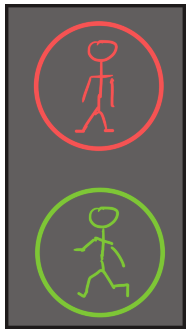


State machine with internal states: a simple example

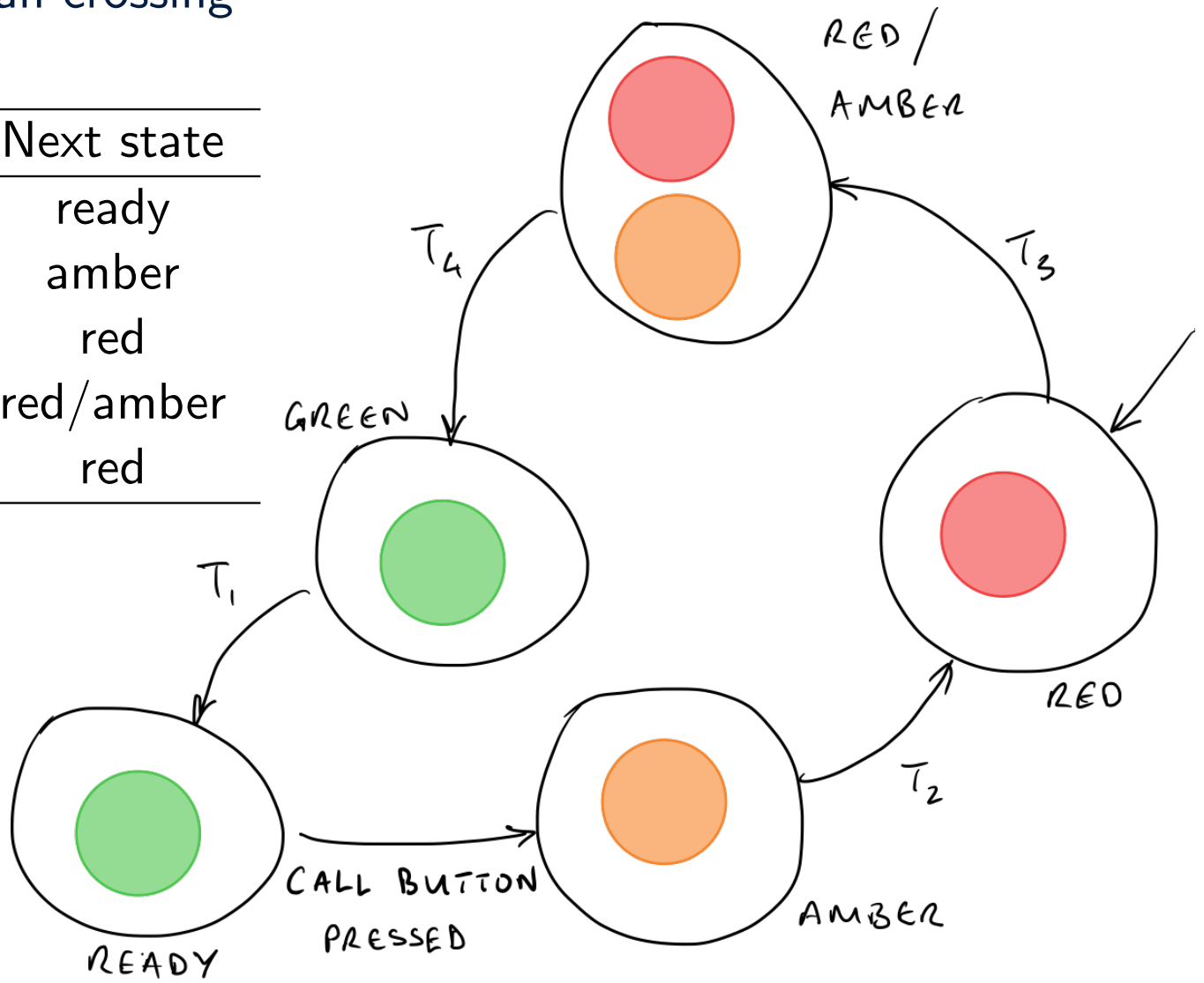
Traffic lights with call button at pedestrian crossing



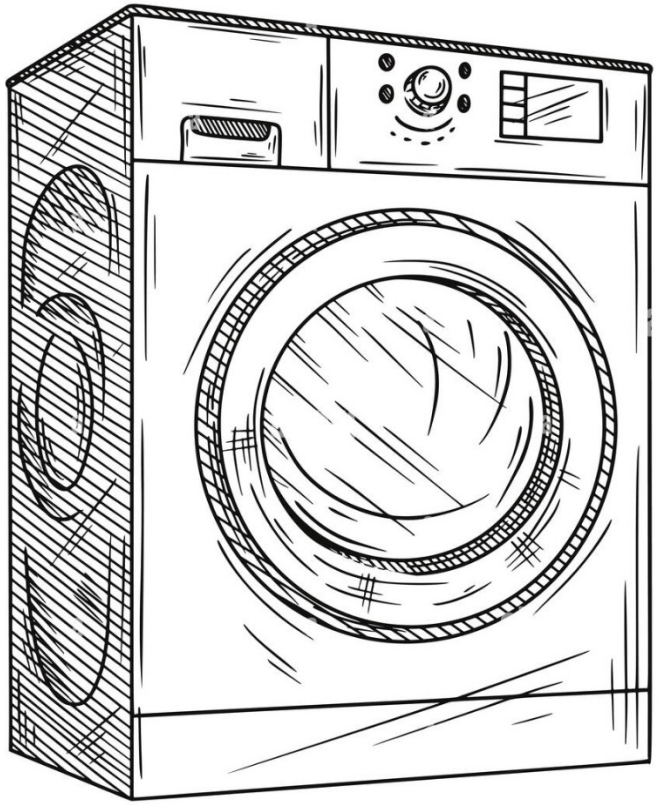
State	Input	Next state
green	T_1	ready
ready	call button	amber
amber	T_2	red
red	T_3	red/amber
red/amber	T_4	red



Output (green light) is the same for two states (green and ready)



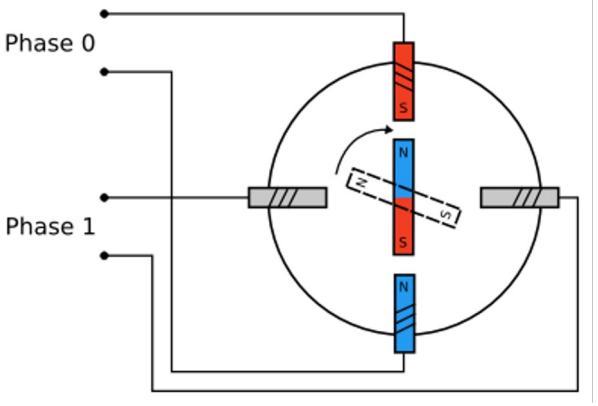
Generalised sequencers



- Sequencers are not limited to conventional counting
- The sequence could be generating a series of outputs to control e.g. a washing machine
  - controller must switch on and off motors, pumps, heaters, door latches, indicator lights...
- The state sequence can be arbitrary
  - there is no need for numerical correlation between the current and next state

Sequence generator example

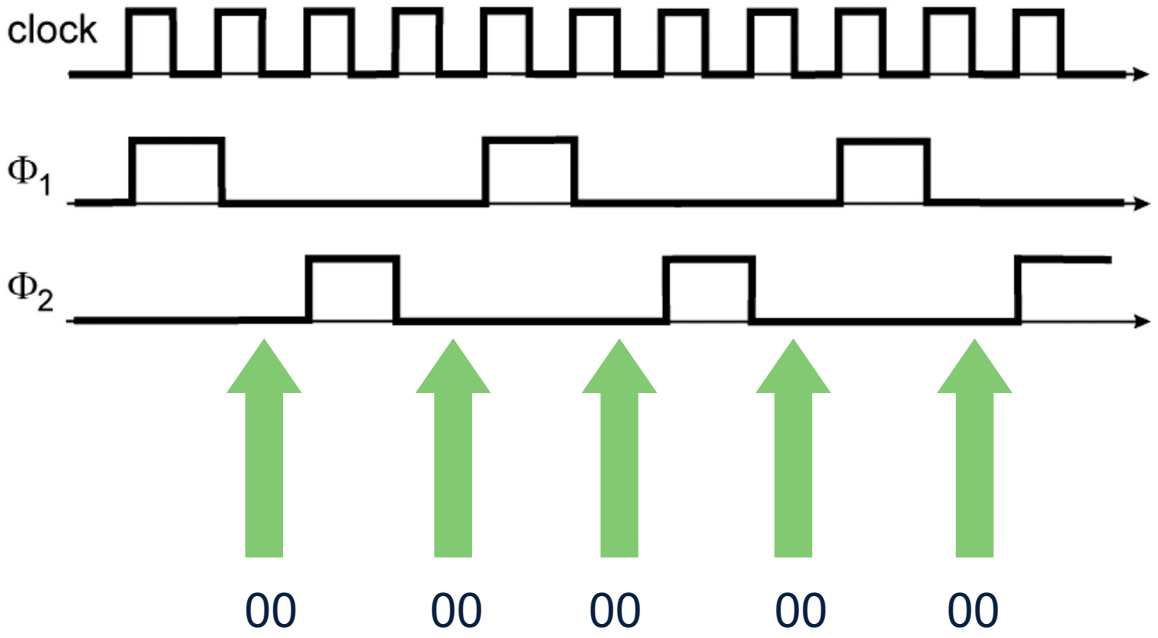
Stepper motor



Φ_1	Φ_2
0	1
0	0
1	0
0	0
0	1
⋮	⋮

Aim – to generate two phase separated pulse streams from a single clock signal

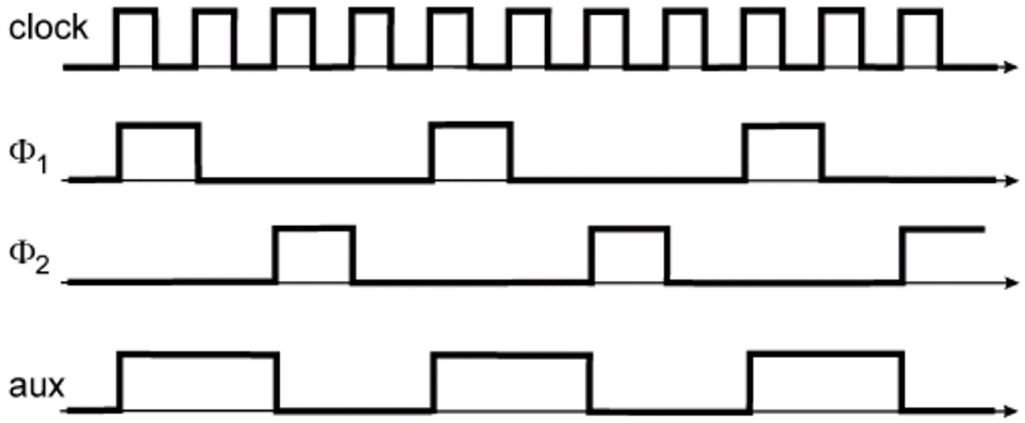
e.g. as the drive signals for a stepper motor



Problem – there are different states in the sequence that are not uniquely identified by the outputs alone

State tracking variable

aux	Φ_1	Φ_2
1	0	1
1	0	0
0	1	0
0	0	0
1	0	1

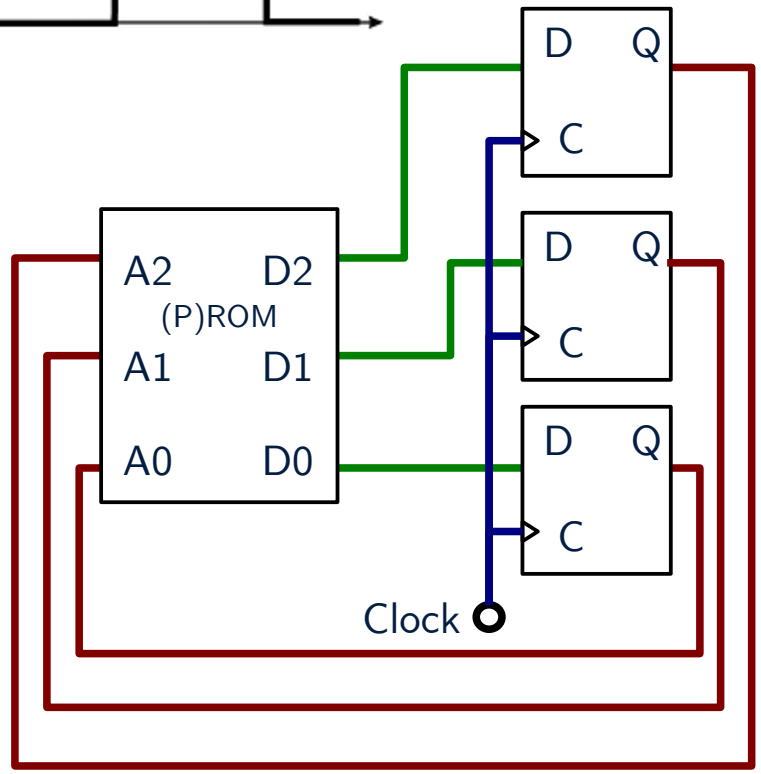


Transition table

Current State			Next state		
aux	Φ_1	Φ_2	aux	Φ_1	Φ_2
0	0	0	1	0	1
1	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	0	0

$\Leftrightarrow$

$A_2A_1A_0$	$D_2D_1D_0$
000	101
001	XXX
010	000
011	XXX
100	010
101	100
110	XXX
111	XXX



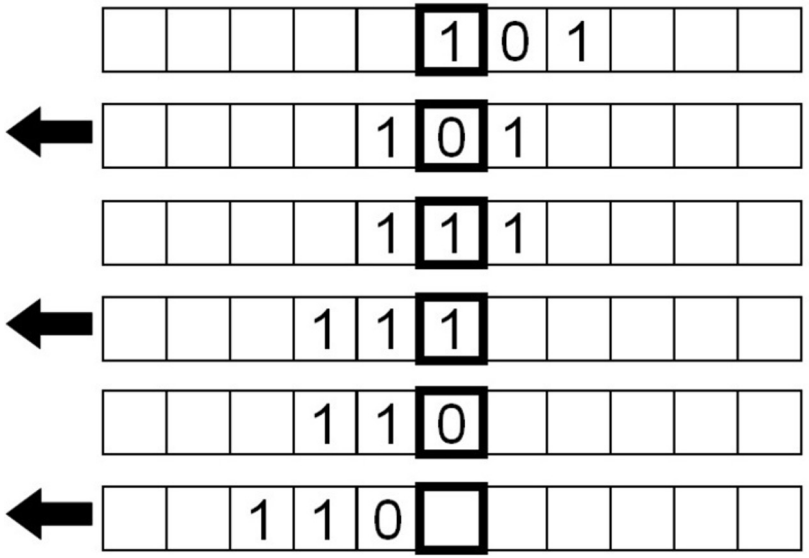
Universal state machines

Turing machine:

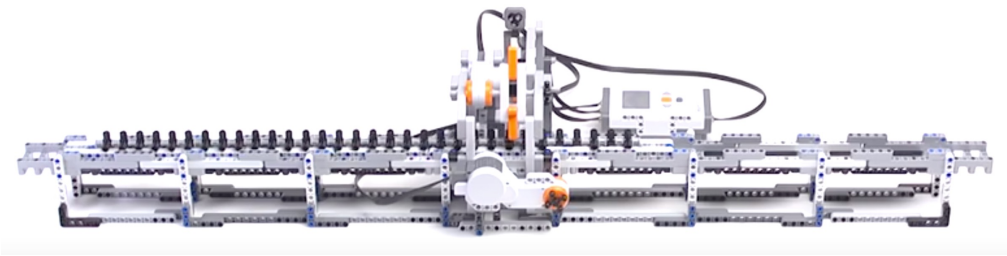
Introduced by Alan Turing in article about computation of numbers in 1936

‘The machine is supplied with a “tape” (the analogue of paper) running through it, and divided into sections (called “squares”) each capable of bearing a “symbol”.’

Alan Turing, **On Computable Numbers, with an Application to the Entscheidungsproblem**, 1936



https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/turing-machine/one.html

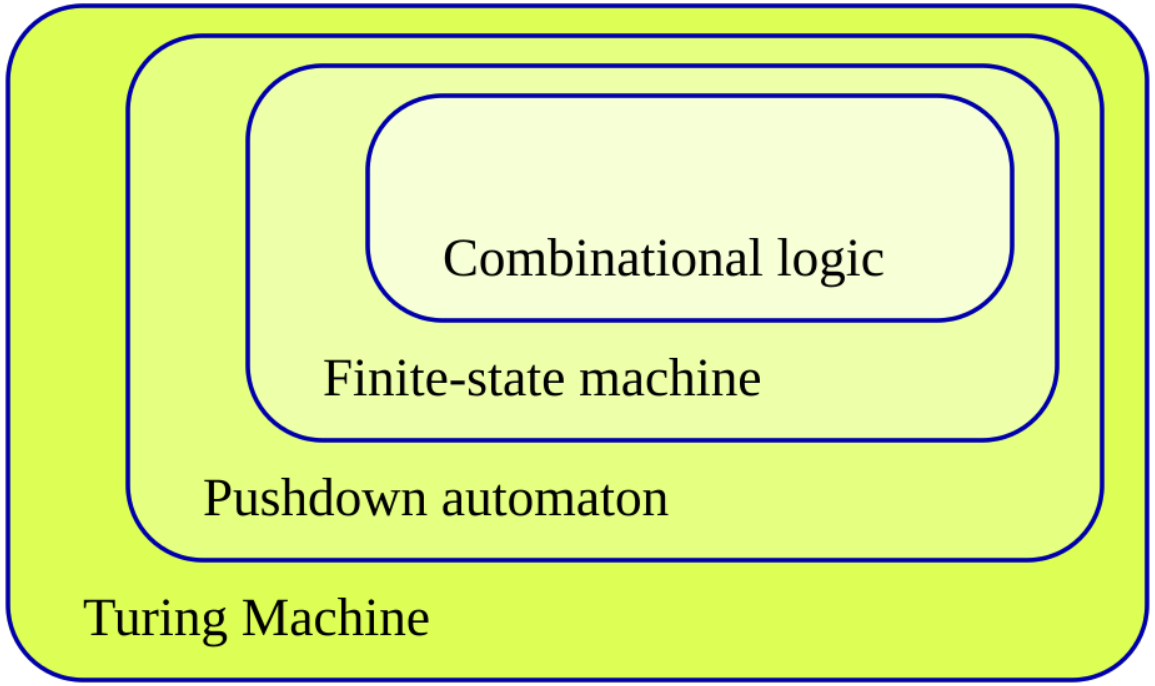


https://www.youtube.com/watch?v=FTSAiF9AHN4

Universal state machines

Design examples in this course

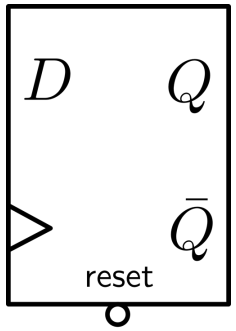
More sophisticated computers
with read/write memory (e.g.RAM)
-- see A2 Intro to Computing



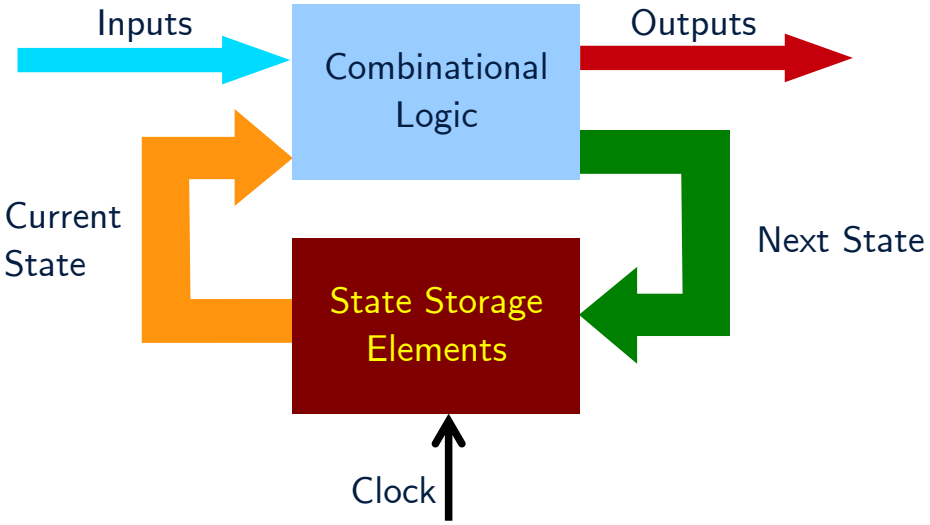
https://en.wikipedia.org/wiki/Finite-state_machine

Synchronous D-type design

A steering table tells you what input is required for a desired output transition



D-type flip-flop



D	Q_n	Q_{n+1}	Transition
0	0	0	$0 \rightarrow 0 = \mathbf{0}$
1	0	1	$0 \rightarrow 1 = \mathbf{H}$
1	1	1	$1 \rightarrow 1 = \mathbf{1}$
0	1	0	$1 \rightarrow 0 = \mathbf{L}$

truth table

$\Rightarrow$

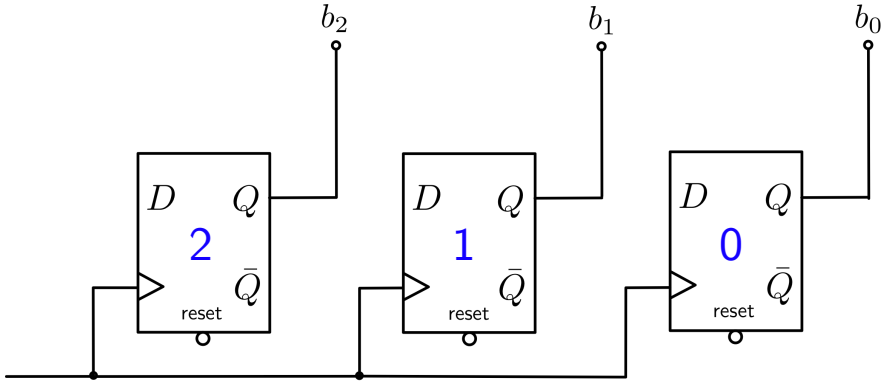
Transition	D
$0 \rightarrow 0 = \mathbf{0}$	0
$0 \rightarrow 1 = \mathbf{H}$	1
$0 \rightarrow 0 = \mathbf{1}$	1
$0 \rightarrow 0 = \mathbf{L}$	0

steering table

Synchronous D-type design

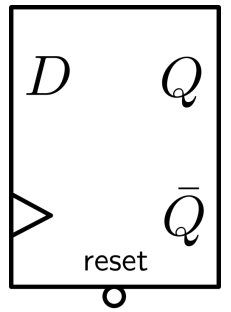
Modulo 6 counter:

#	Current state	Next state
0	000	001
1	001	010
2	010	011
3	011	100
4	100	101
5	101	000



D-type flip-flop:

D	Clock	Q_{n+1}
0	↑	0
1	↑	1



$$Q_{n+1} = D_n$$

K-map for next b_2:

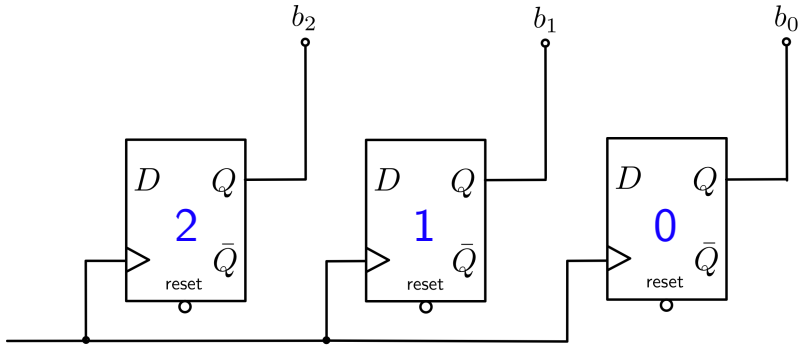
b_2 \ b_1b_0	00	01	11	10
0	0	0	1	0
1	1	0	X	X

$$\Rightarrow \quad D_2 = b_1.b_0 + b_2.\bar{b}_0$$

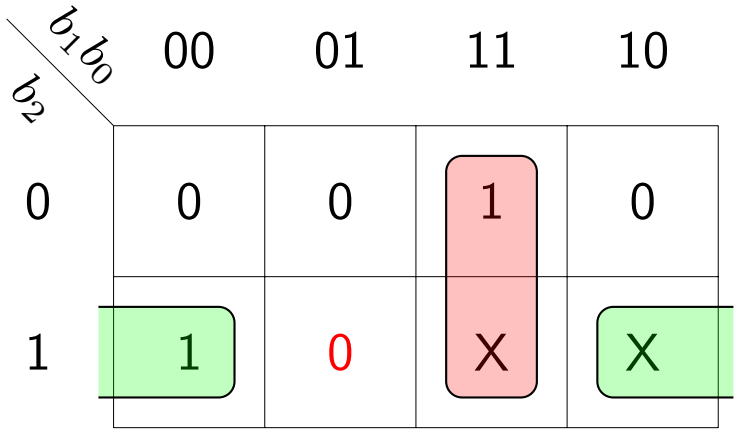
Completing the design

Modulo 6 counter:

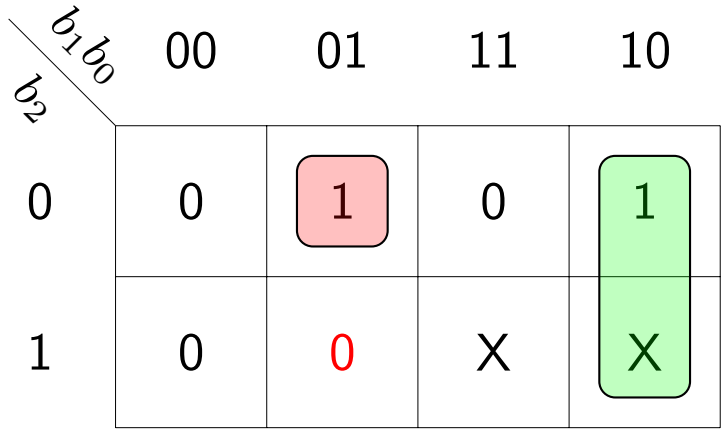
#	Current state	Next state
0	000	001
1	001	010
2	010	011
3	011	100
4	100	101
5	101	000



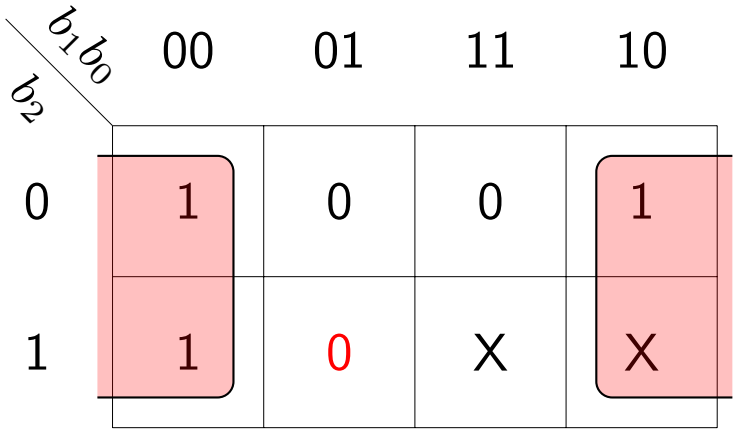
D inputs need to be set to next value of b output



$$D_2 = b_1.b_0 + b_2.\bar{b}_0$$



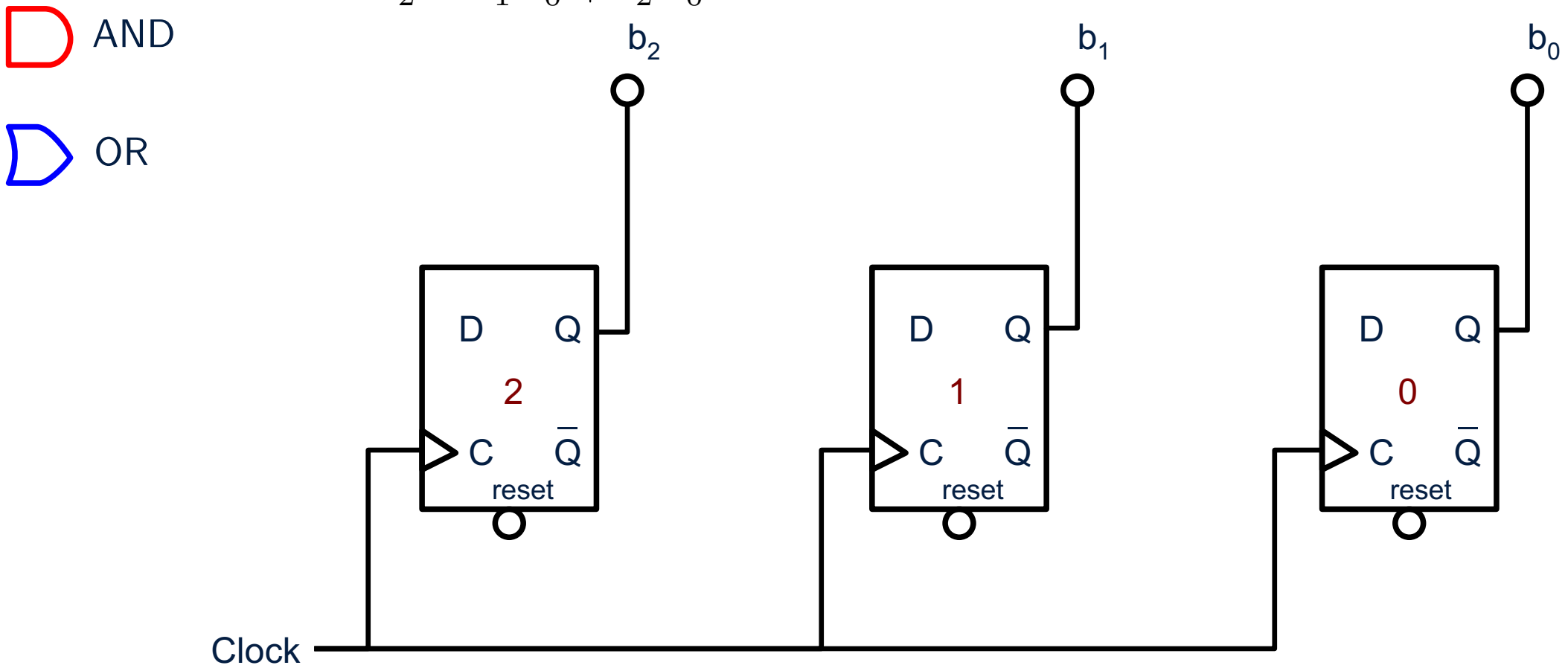
$$D_1 = b_1.\bar{b}_0 + \bar{b}_2.\bar{b}_1.b_0$$



$$D_0 = \bar{b}_0$$

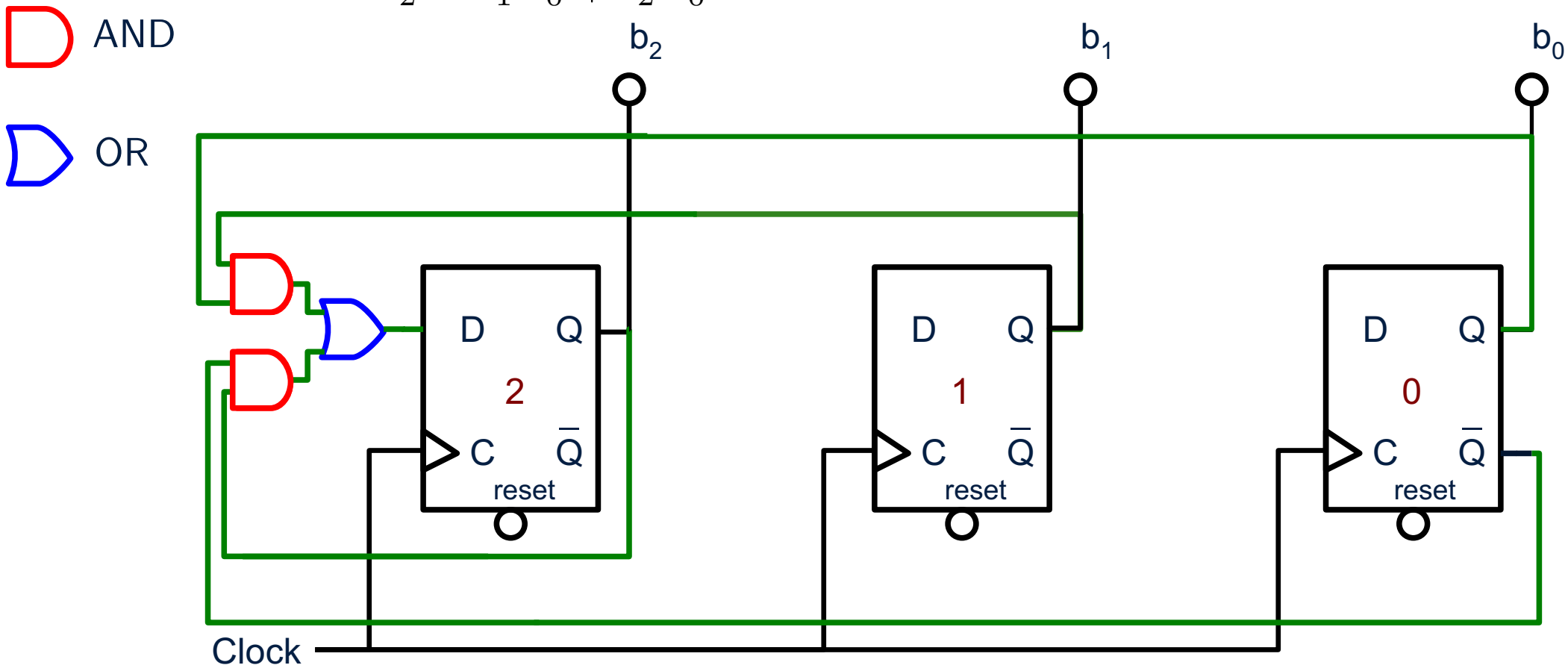
Final logic design

$$D_2 = b_1.b_0 + b_2.\bar{b}_0$$



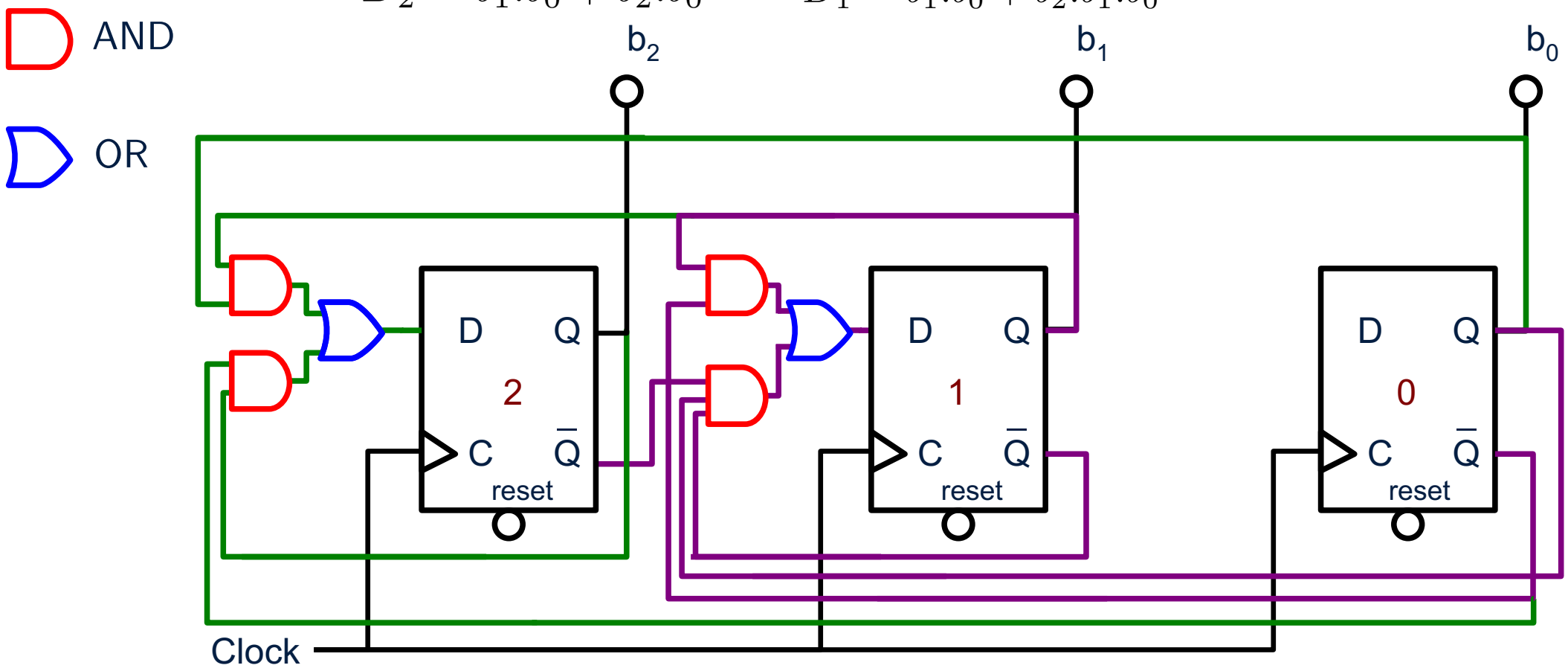
Final logic design

$$D_2 = b_1.b_0 + b_2.\bar{b}_0$$



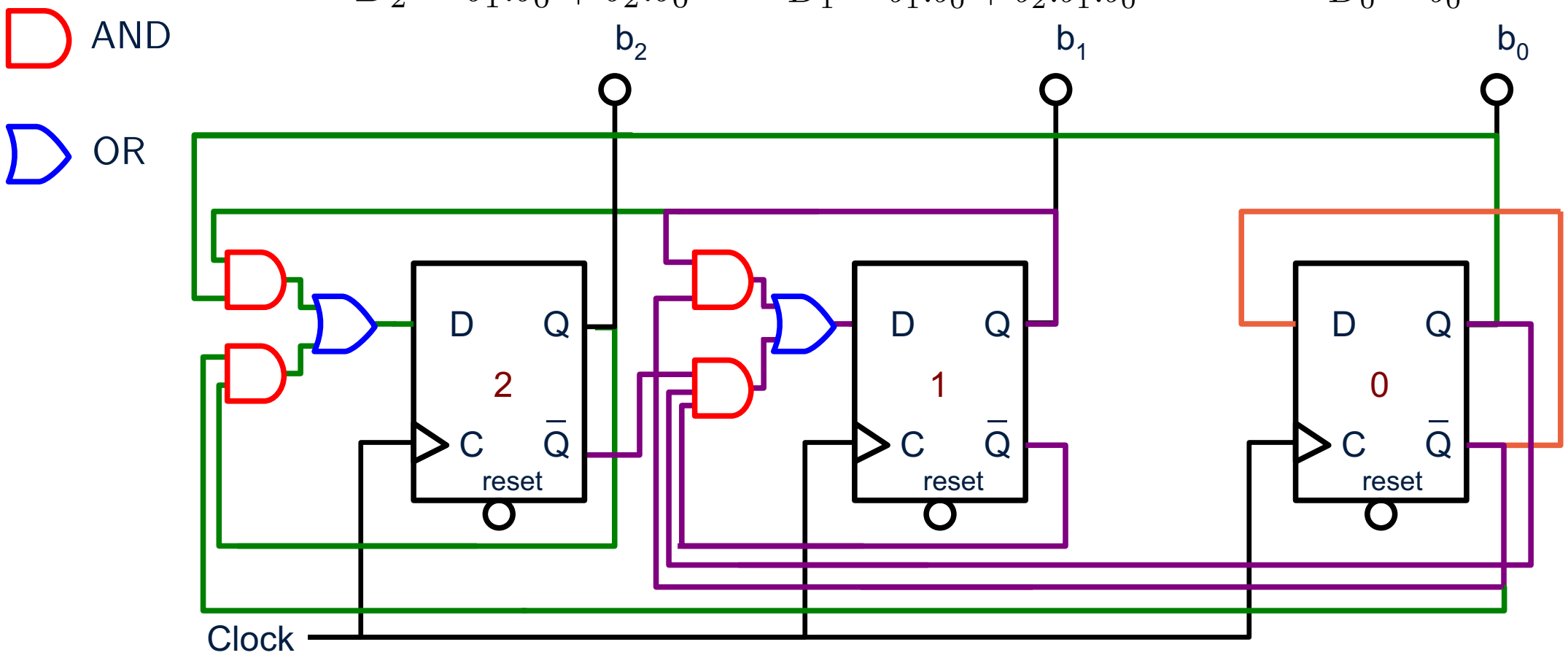
Final logic design

$$D_2 = b_1.b_0 + b_2.\bar{b}_0 \qquad D_1 = b_1.\bar{b}_0 + \bar{b}_2.\bar{b}_1.b_0$$



Final logic design

$$D_2 = b_1.b_0 + b_2.\bar{b}_0 \qquad D_1 = b_1.\bar{b}_0 + \bar{b}_2.\bar{b}_1.b_0 \qquad D_0 = \bar{b}_0$$



A simple modulo 6 counter – needs quite a lot of wiring and extra asynchronous gates

... but there is no possibility of runt pulses – the system can never enter unwanted states

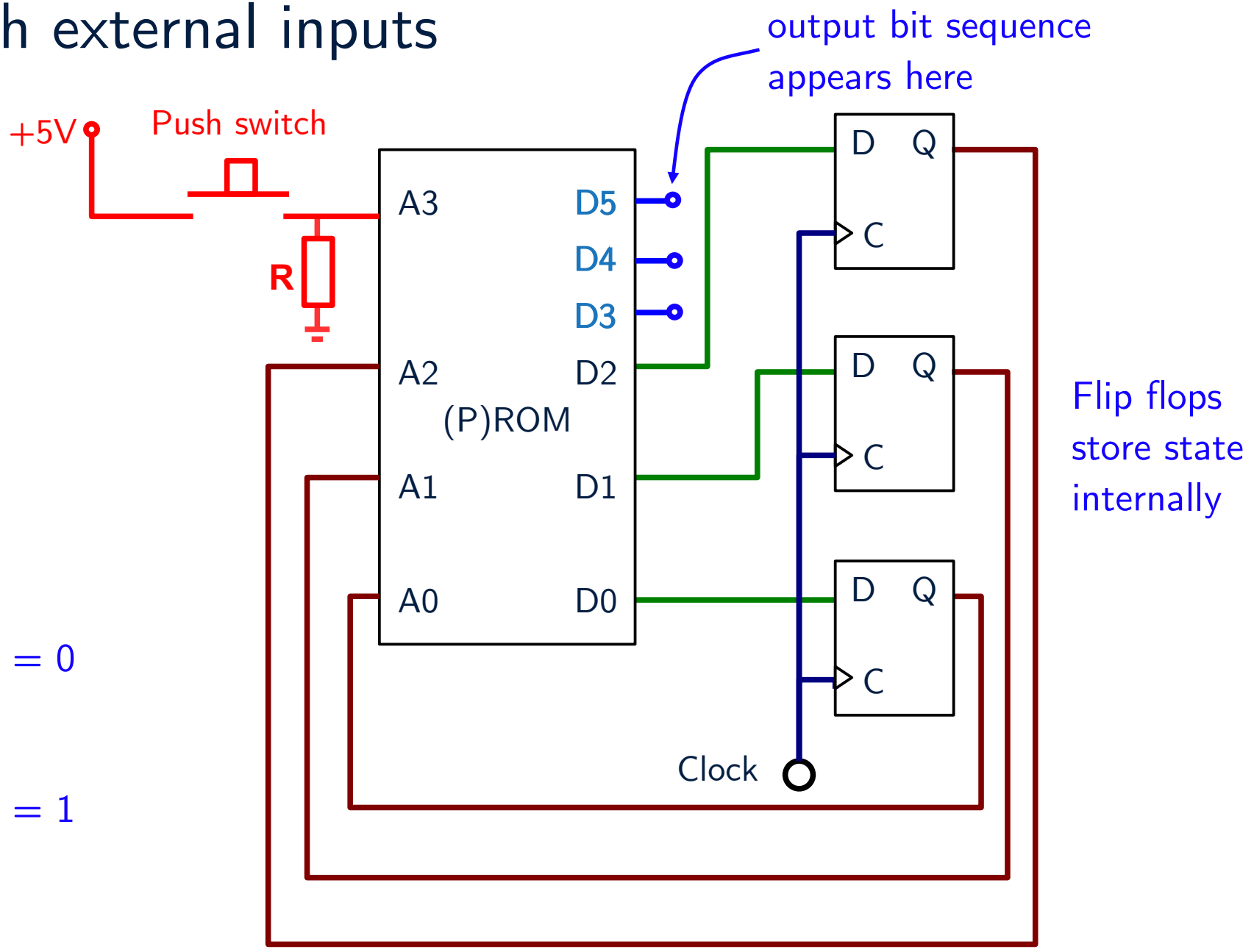
State machine with external inputs

Example state machine with a single input signal

Two modes of operation determined by switch:

Count odd numbers:
Sequence 1,3,5,7 if input A3 = 0

Count even numbers:
Sequence 0,2,4,6 if input A3 = 1



State machine with external inputs

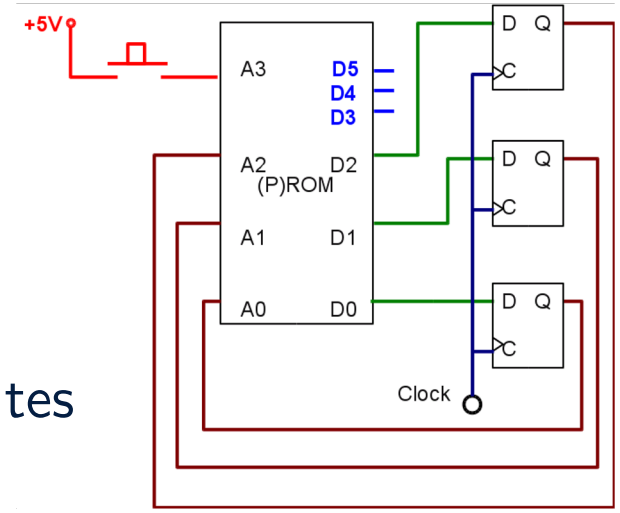
ROM contents:

#	Address A3	A2	A1	A0	Data stored at Address D2	D1	D0	D5	D4	D3	
0	0	0	0	0	0	0	1	0	0	1	**1**
1	0	0	0	1	0	1	0	0	1	1	**3**
2	0	0	1	0	0	1	1	1	0	1	**5**
3	0	0	1	1	0	0	0	1	1	1	**7**
.	.	.	.	.	.	.	.	.	.	.	
8	1	0	0	0	0	0	1	0	0	0	**0**
9	1	0	0	1	0	1	0	0	1	0	**2**
10	1	0	1	0	0	1	1	1	0	0	**4**
11	1	0	1	1	0	0	0	1	1	0	**6**

External input — Current internal state — Next state — Outputs

Count 1,3,5,7
if A3 = 0

Count 0,2,4,6
if A3 = 1



States defined internally, not by outputs

ROM contains all information about outputs and transitions between states – no external combinatorial logic required

Overview of lectures

1. Logical functions and logic gates
2. Low level logic design
3. Binary number representation
4. Binary arithmetic
5. Integration of digital logic components
6. Memory and sequential circuits
7. **Design of sequential logic**
8. Data converters: analogue to digital / digital to analogue

Please send feedback, comments and corrections to mark.cannon@eng.ox.ac.uk