Digital Electronics and Microprocessor Systems (ELEC211)

Dave McIntosh and Valerio Selis

dmc@liv.ac.uk

v.selis@liv.ac.uk

Digital 11: ASM design and serial communications

UNIVERSITY OF LIVERPOOL

Outline

- **ASM design**
  - Dice game controller example
  - ***(from Roth & Kinney 6th Edition: Sections 19.2-3, p.631 on..***
    … and in 7th Edition, p.667 on)
- **Serial coding schemes**
  - NRZ, NRZI, RZ, Manchester
  - Code conversion with Moore/Mealy machines
- **Extra notes about shift registers** (Lec 20-21/Dig 7-8)

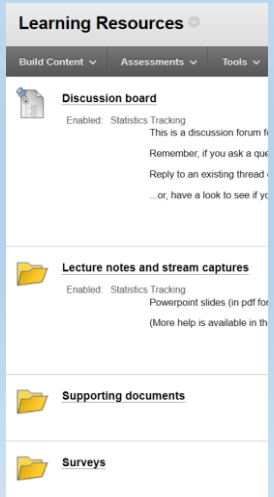
Previous material

State machines ✓
Moore and Mealy machines ✓
ASM charts ✓
One Hot and Encoding methods ✓
State tables ✓
State graphs ✓
PLA; PLA tables ✓
Serial and parallel ✓
Timing diagrams ✓
Glitches ✓

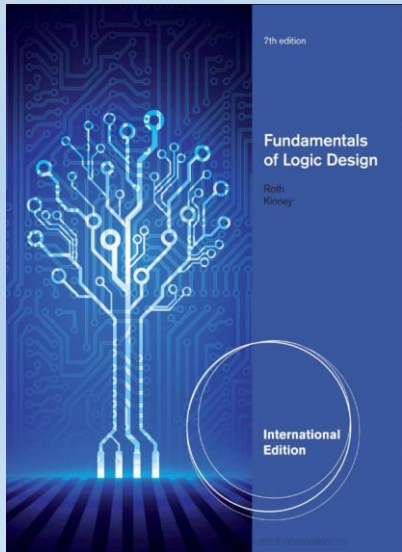
Use VITAL!:

- Stream lectures
- Handouts
- Notes and Q&A each week
- Discussion Board
- Exam resources

www.liv.ac.uk/vital



Course textbook – 7th ed. available as e-book!

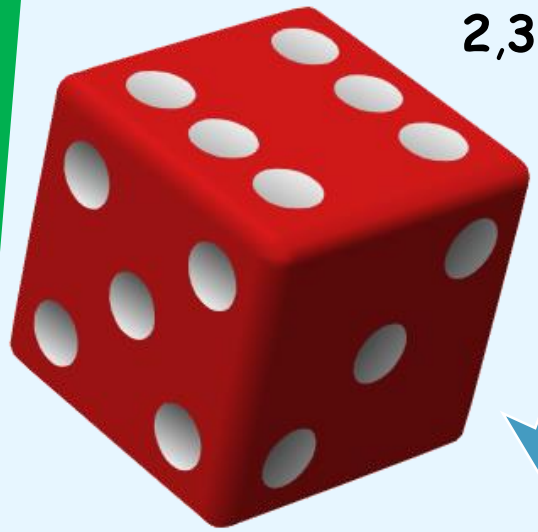


UNIVERSITY OF LIVERPOOL

Roll & add

7 or 11 - Win

2,3 or 12 - lose



Other - Points (roll again)

Next score = Points - Win

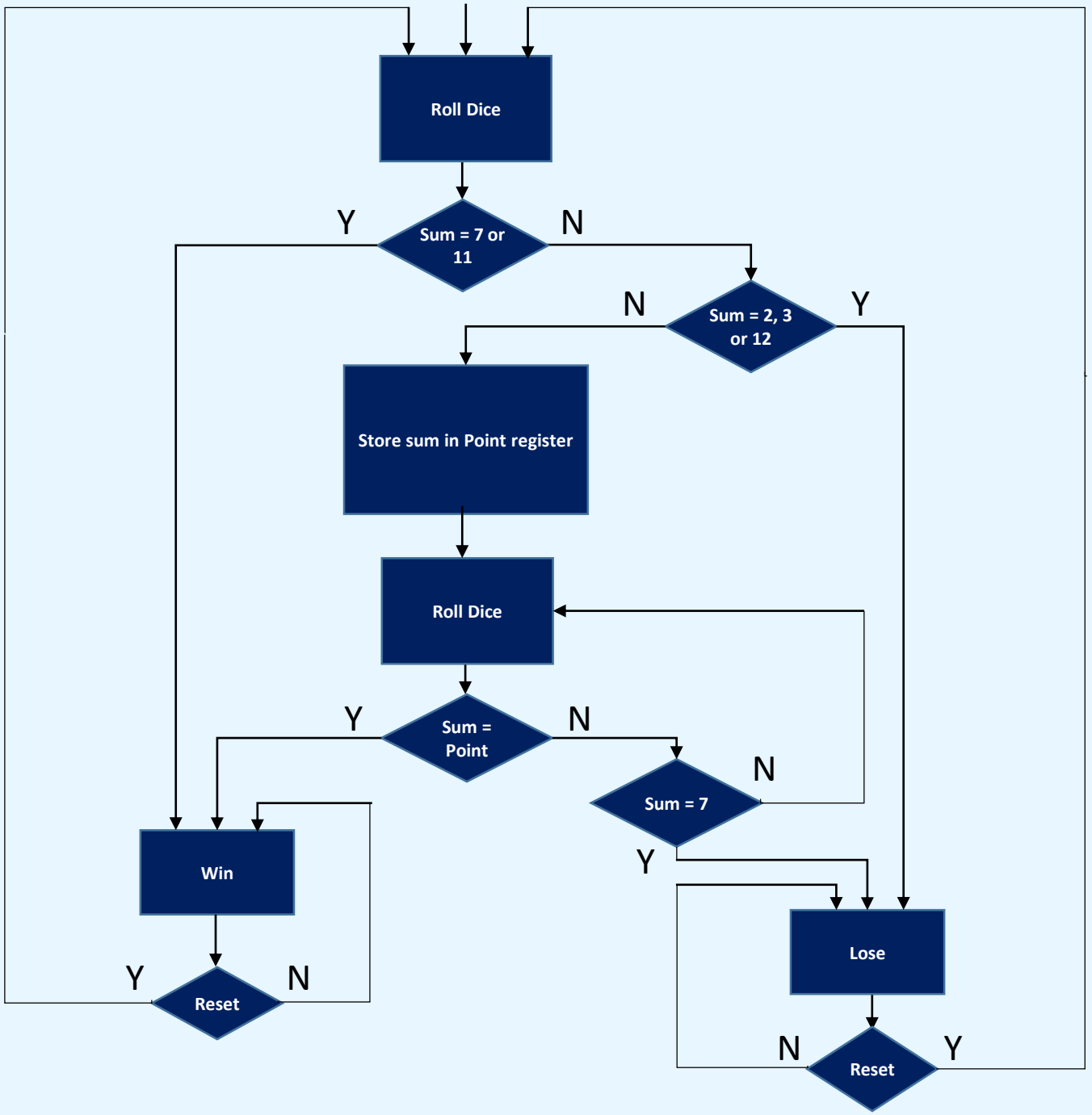
Next score = 7 - lose

Next score = other – roll again

https://en.wikipedia.org/wiki/File:Two_red_dice_01.svg

UNIVERSITY OF LIVERPOOL

(Dice game ‘flowchart’)



THIS IS NOT AN ASM CHART!!!

Some digital design principles

General sequential circuit design principles:

(**Synthesis** is the opposite of **analysis**)

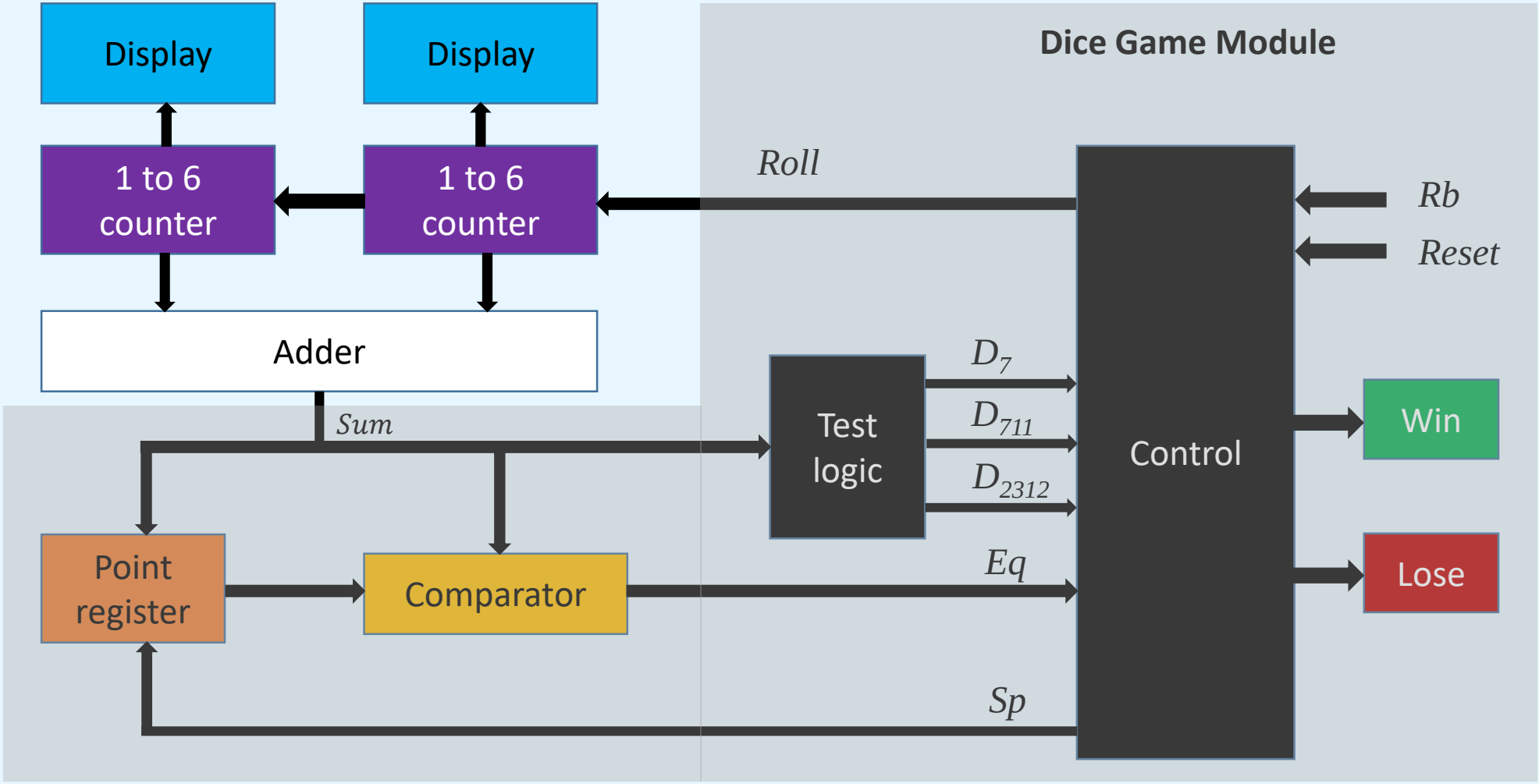
- [State graph →] State table
- Calculate number of flip-flops required
- State transition table
- Flip-flop next-state maps, input maps
- Derive flip-flop input eqns.; Output fns.
- Realise with gates or equivalent
- Check by signal tracing / simulation / testing

Design via ASM chart…

- Block diagram
- Define control circuit inputs/outputs
- Properly sequenced ASM chart
- Continue to implement (realise):
  - Combinational circuit and..
  - ..Flip-flops store the state

Dice game block diagram

- Counter
- Adder
- Test logic
- Control circuit
- Register
- Comparator



UNIVERSITY OF LIVERPOOL

Assumptions:

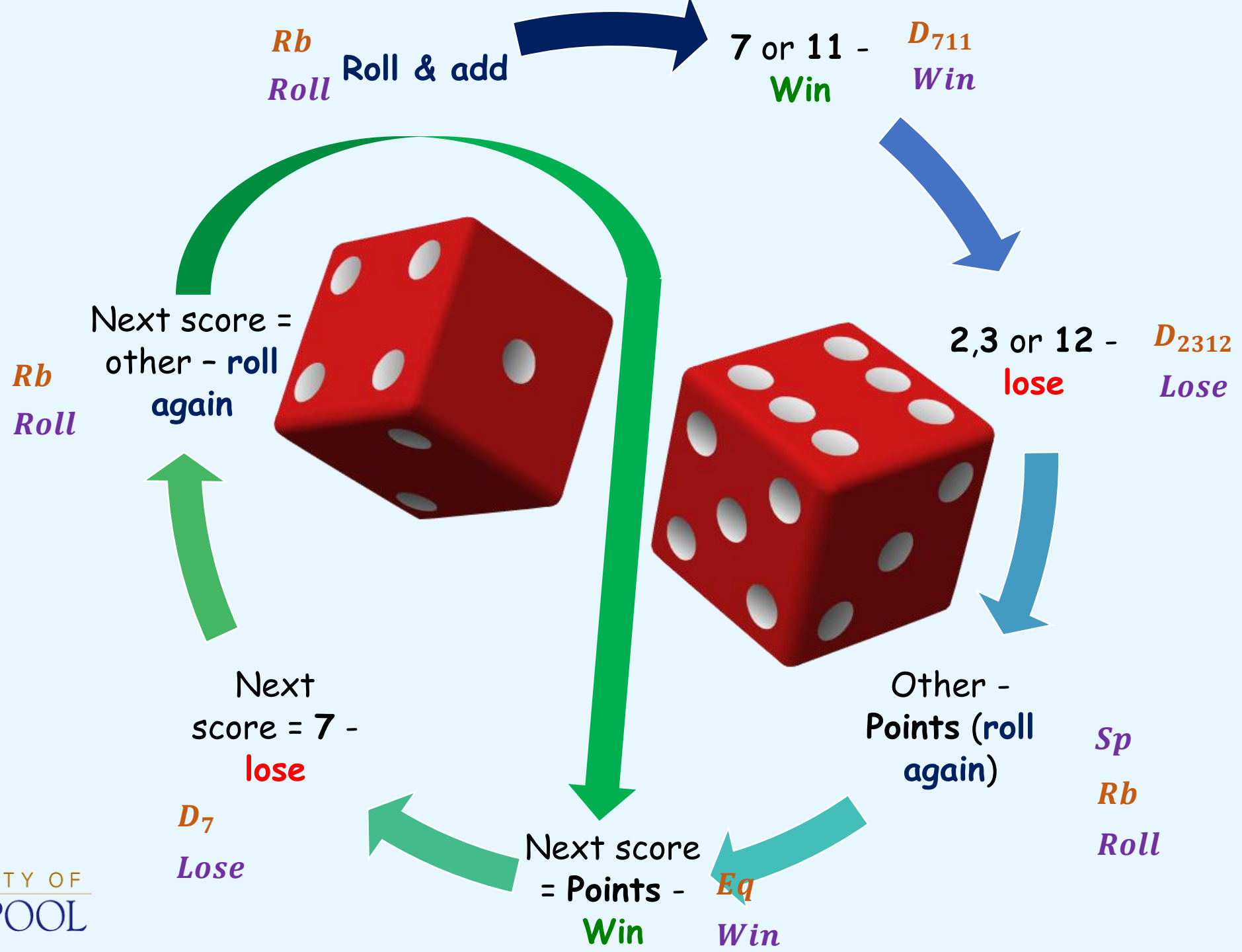
- ***push buttons are properly debounced***
- ***Rb changes are properly synchronized with clock***

INPUTS to control circuit

D_7 = 1 if dice sum is 7
D_{711} = 1 if dice sum is 7 or 11
D_{2312} = 1 if dice sum is 2, 3, or 12
Eq = 1 if dice sum equals number stored in Point register
Rb = 1 when Roll button pressed
Reset = 1 when Reset button pressed

OUTPUTS from control circuit

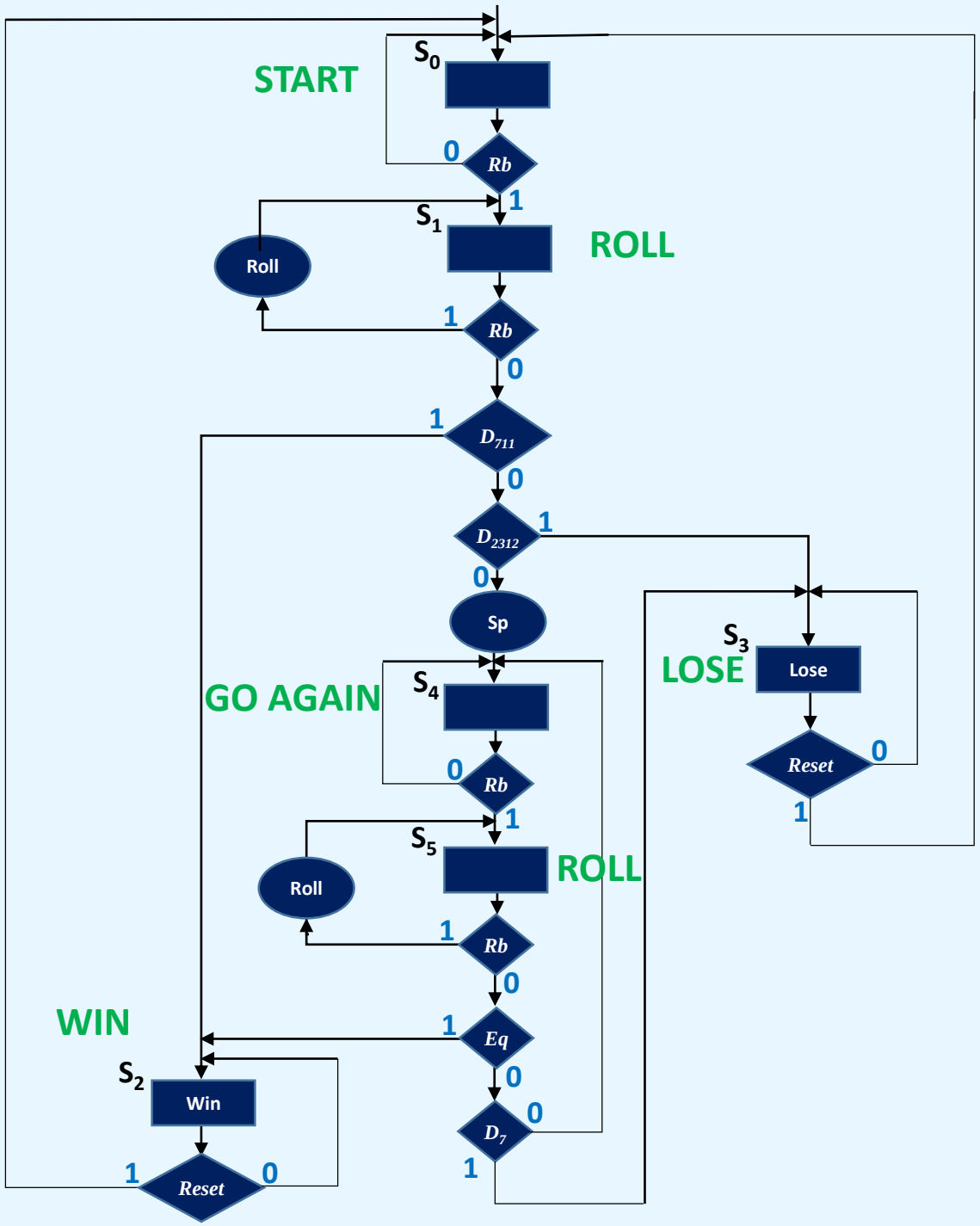
Roll = 1 enables dice counters
Sp = 1 → sum is stored in Point register
Win = 1 turns on Win light
Lose = 1 turns on Lose light



https://en.wikipedia.org/wiki/File:Two_red_dice_01.svg

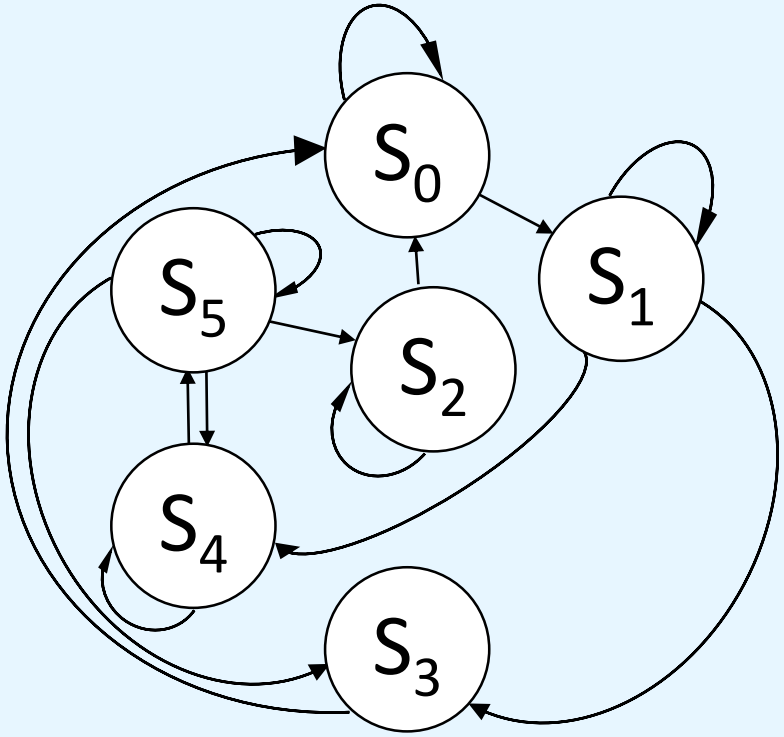
UNIVERSITY OF LIVERPOOL

Dice game
ASM chart



UNIVERSITY OF LIVERPOOL

State graph???

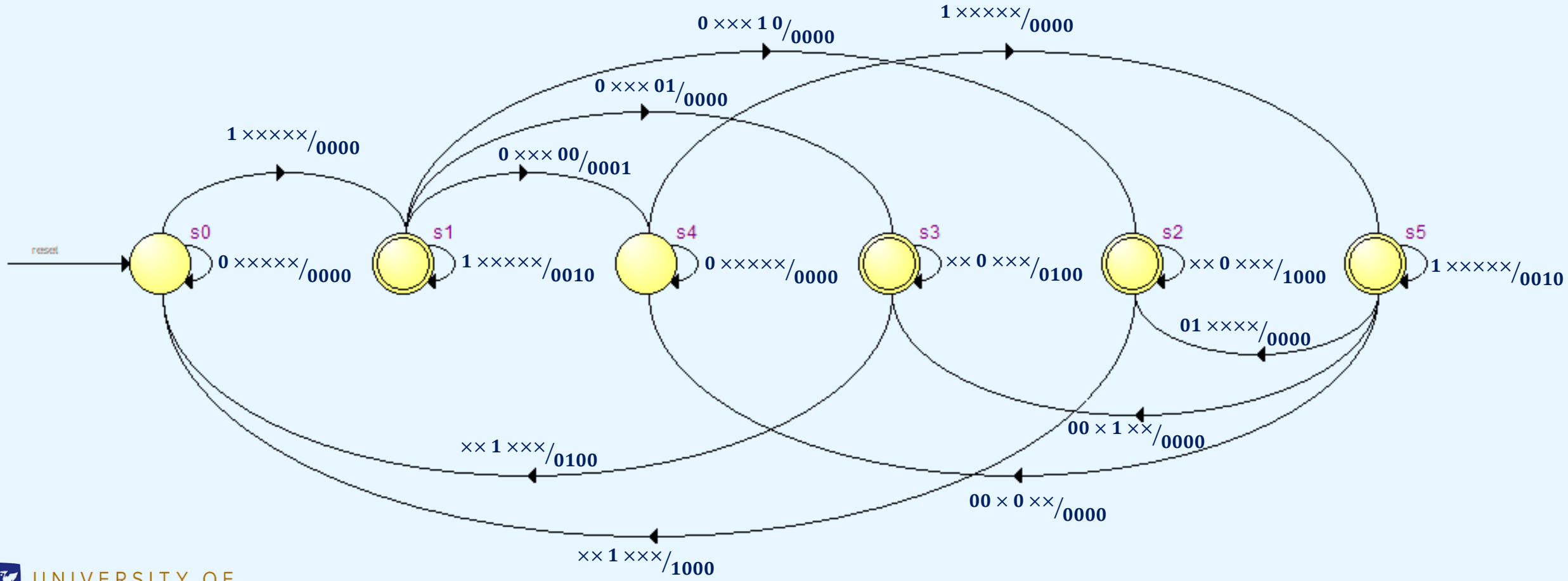


Dice game: **State table**… (Quartus II)

%Current state	Current inputs						=> Current outputs				Next state%
ss,	Rb,	Eq,	reset,	D7,	D711,	D2312	=> Win,	Lose,	Roll,	Sp,	ss;
s0,	0,	X,	X,	X,	X,	X	=> 0,	0,	0,	0,	s0;
s0,	1,	X,	X,	X,	X,	X	=> 0,	0,	0,	0,	s1;
s1,	0,	X,	X,	X,	0,	0	=> 0,	0,	0,	1,	s4;
s1,	0,	X,	X,	X,	0,	1	=> 0,	0,	0,	0,	s3;
s1,	0,	X,	X,	X,	1,	0	=> 0,	0,	0,	0,	s2;
s1,	1,	X,	X,	X,	X,	X	=> 0,	0,	1,	0,	s1;
s2,	X,	X,	1,	X,	X,	X	=> 1,	0,	0,	0,	s0;
s2,	X,	X,	0,	X,	X,	X	=> 1,	0,	0,	0,	s2;
s3,	X,	X,	0,	X,	X,	X	=> 0,	1,	0,	0,	s3;
s3,	X,	X,	1,	X,	X,	X	=> 0,	1,	0,	0,	s0;
s4,	0,	X,	X,	X,	X,	X	=> 0,	0,	0,	0,	s4;
s4,	1,	X,	X,	X,	X,	X	=> 0,	0,	0,	0,	s5;
s5,	0,	0,	X,	0,	X,	X	=> 0,	0,	0,	0,	s4;
s5,	0,	0,	X,	1,	X,	X	=> 0,	0,	0,	0,	s3;
s5,	0,	1,	X,	X,	X,	X	=> 0,	0,	0,	0,	s2;
s5,	1,	X,	X,	X,	X,	X	=> 0,	0,	1,	0,	s5;

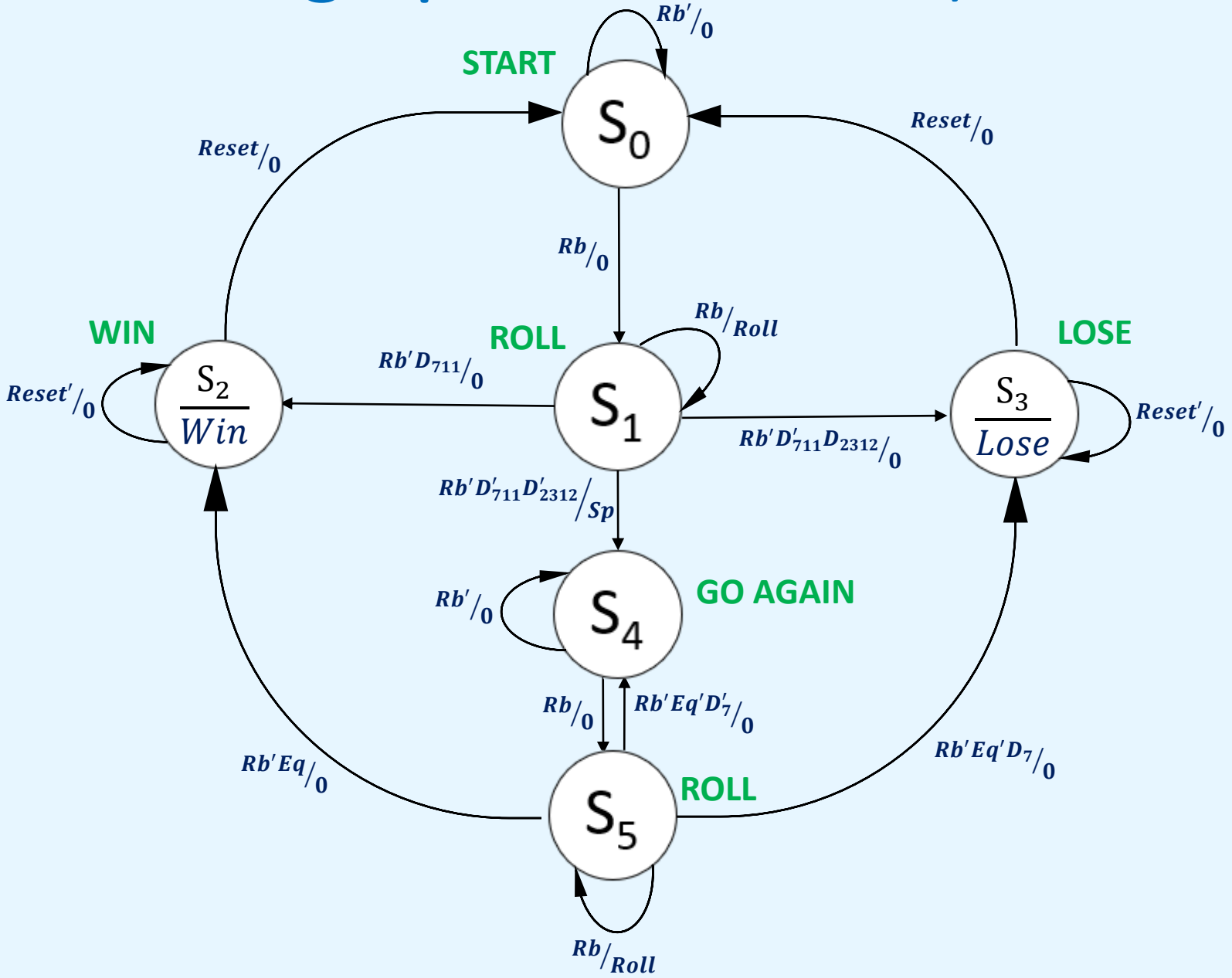
Dice game: **State graph** using Quartus II

$Rb\ Eq\ reset\ D7\ D711\ D2312 / Win\ Lose\ Roll\ Sp$



Dice game: **State graph** another way…

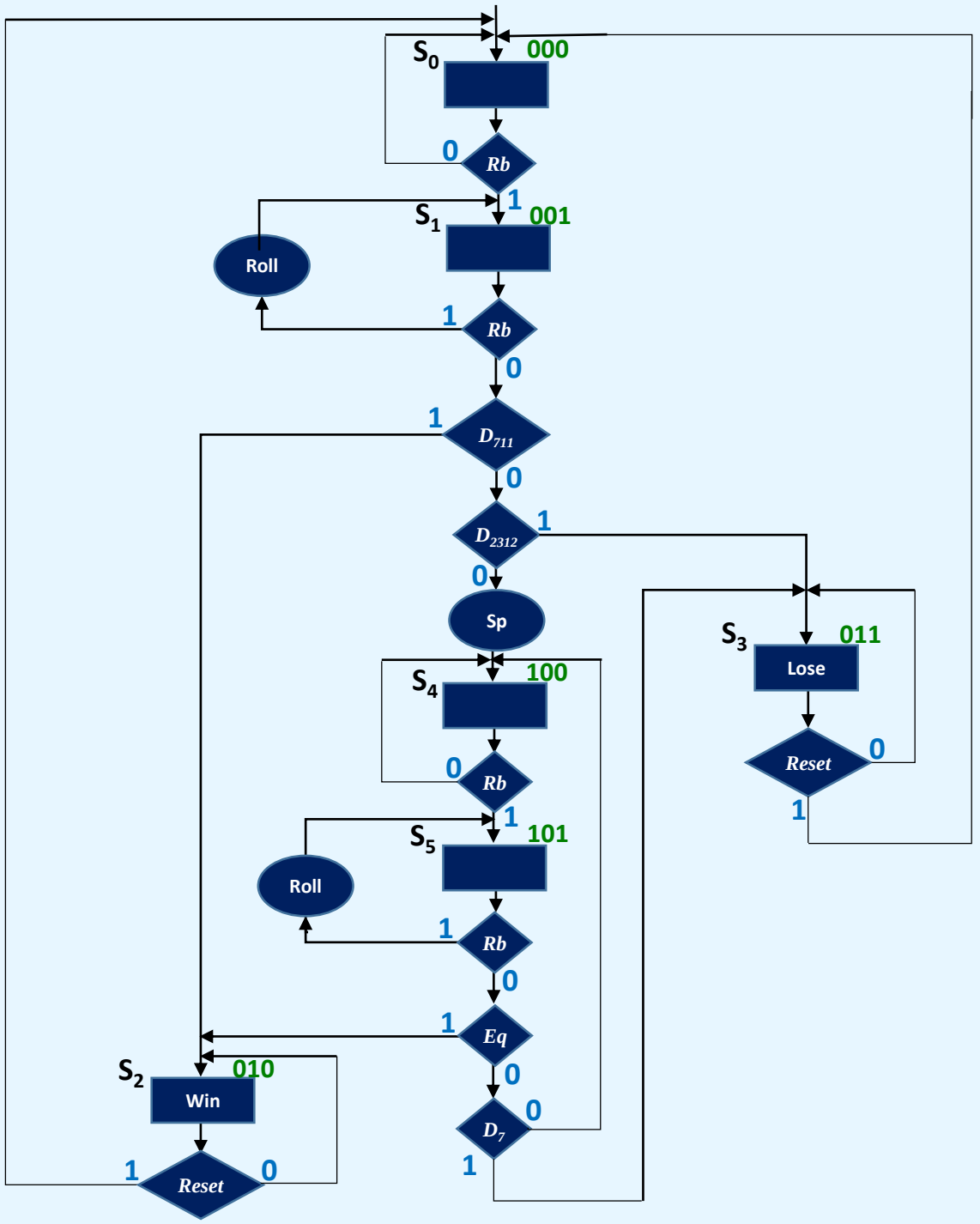
$Inputs/Outputs$



- Has Moore and Mealy **outputs**…
- But it's still a **Mealy machine!**

UNIVERSITY OF LIVERPOOL

Dice game
ASM chart



Note ‘adjacent’ assignments, e.g. S_2, S_3 (010 and 011)

… these two states are both possible next states of S_1

Dice game: State transition table…

State	ABC
S_0	000
S_1	001
S_2	010
S_3	011
S_4	100
S_5	101

	Present state	Present Inputs						Next state	Present Outputs			
	ABC	Rb	Reset	D_7	D_{711}	D_{2312}	Eq	$A^+B^+C^+$	Win	Lose	Roll	Sp
START S_0	000	0	X	–	–	–	–	**000**	0	0	0	0
	000	1	X	–	–	–	–	**001**	0	0	0	0
ROLL S_1	001	1	X	–	–	–	–	**001**	0	0	1	0
	001	0	X	X	0	0	–	**100**	0	0	0	1
	001	0	X	X	0	1	–	**011**	0	0	0	0
	001	0	X	X	1	–	–	**010**	0	0	0	0
WIN S_2	010	X	0	–	–	–	–	**010**	1	0	0	0
	010	X	1	–	–	–	–	**000**	1	0	0	0
LOSE S_3	011	X	1	–	–	–	–	**000**	0	1	0	0
	011	X	0	–	–	–	–	**011**	0	1	0	0
GO AGAIN S_4	100	0	X	–	–	–	–	**100**	0	0	0	0
	100	1	X	–	–	–	–	**101**	0	0	0	0
ROLL S_5	101	0	X	0	X	X	0	**100**	0	0	0	0
	101	0	X	1	X	X	0	**011**	0	0	0	0
	101	0	X	X	X	X	1	**010**	0	0	0	0
	101	1	X	–	–	–	–	**101**	0	0	1	0
	110	–	–	–	–	–	–	---	–	–	–	–
	111	–	–	–	–	–	–	---	–	–	–	–

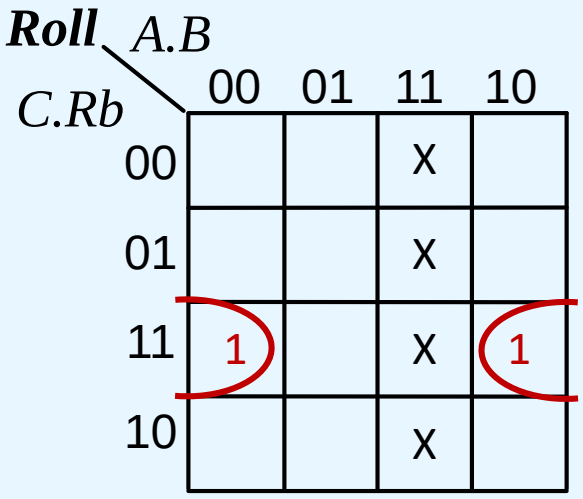
16 functions, 9 (possible) variables…

Next state / output equations…?

E.g.

$$\textbf{Roll} = A'.B'.C.Rb + A.B'.C.Rb$$
$$= (A'.B'.C + A.B'.C).Rb$$
$$= (A' + A).B'.C.Rb$$
$$= B'.C.Rb$$

or…

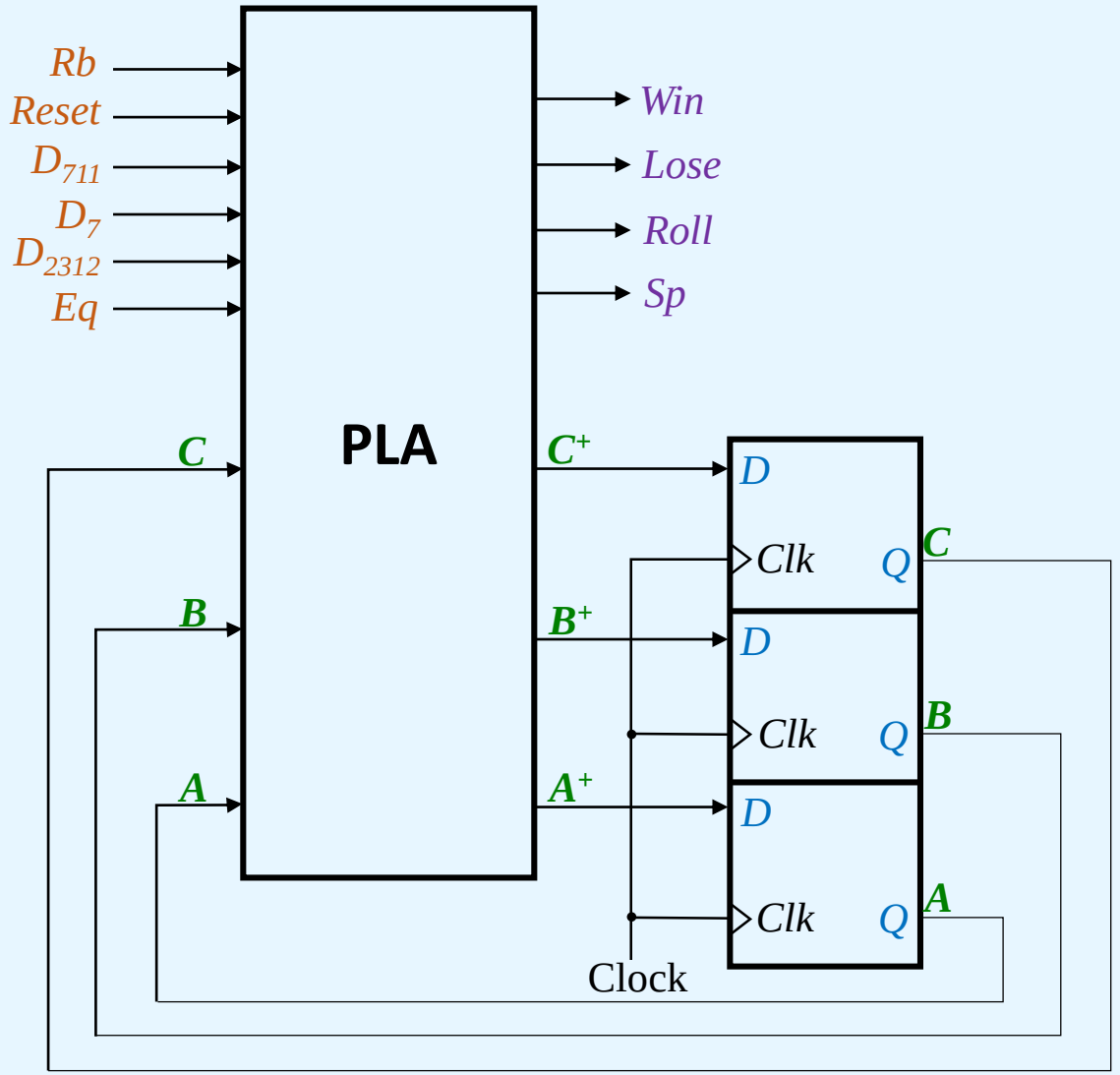


$$\textbf{Roll} = B'.C.Rb$$

UNIVERSITY OF LIVERPOOL

Dice game: **realisation** with a PLA and flip-flops…

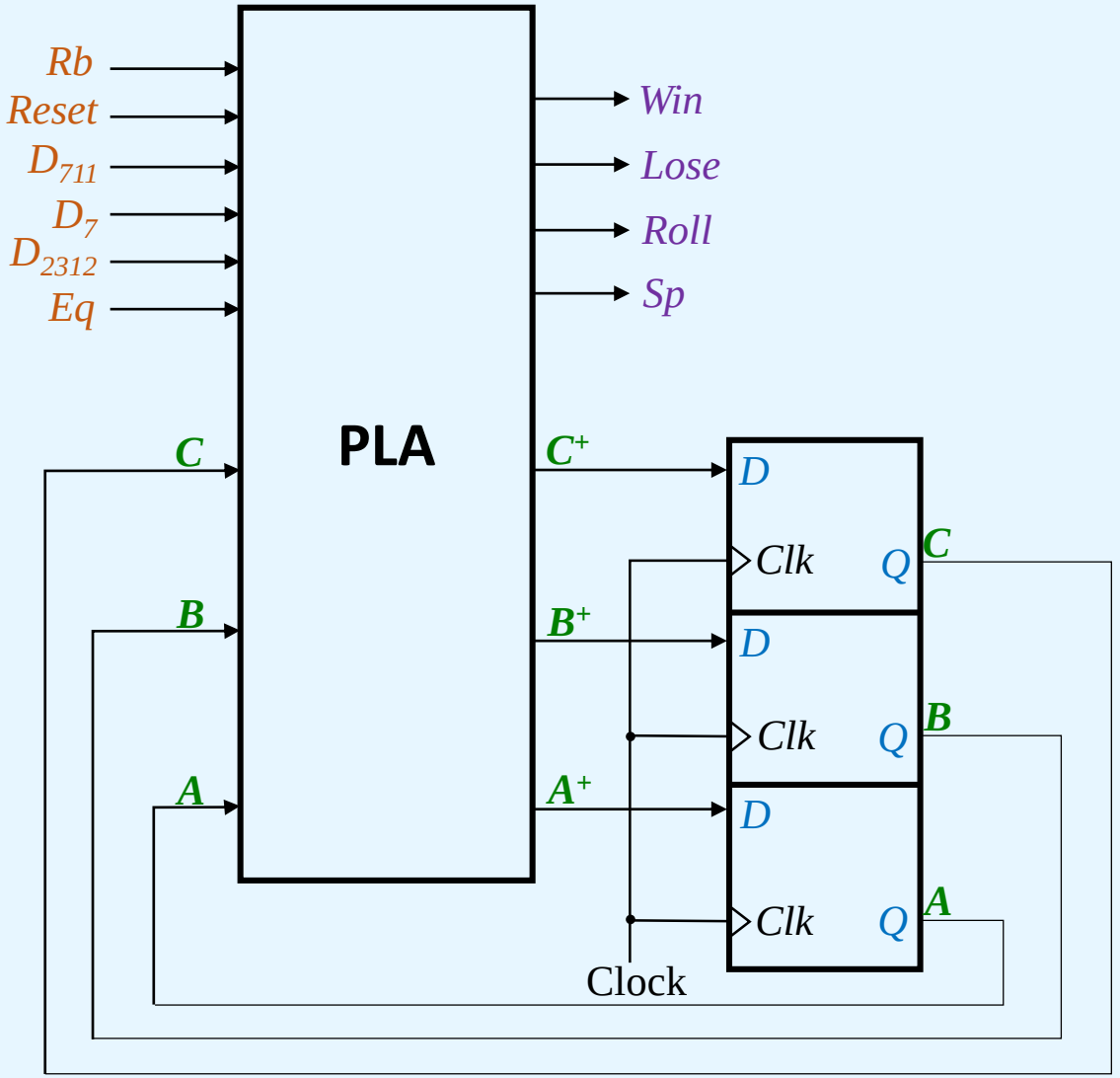
- Combinational circuit
- Flip-flops to store the state



Need to program the logic array … PLA table …

UNIVERSITY OF LIVERPOOL

Dice game: **realisation** with a PLA and flip-flops…



PLA table

		Inputs							Outputs				
		ABC	Rb	Reset	D_7	D_{711}	D_{2312}	Eq	$A^+B^+C^+$	Win	Lose	Roll	Sp
S_0	1	000	0	–	–	–	–	–	000	0	0	0	0
	2	000	1	–	–	–	–	–	001	0	0	0	0
S_1	3	001	1	–	–	–	–	–	001	0	0	1	0
	4	001	0	–	–	0	0	–	100	0	0	0	1
	5	001	0	–	–	0	1	–	011	0	0	0	0
	6	001	0	–	–	1	–	–	010	0	0	0	0
S_2	7	010	–	0	–	–	–	–	010	1	0	0	0
	8	010	–	1	–	–	–	–	000	1	0	0	0
S_3	9	011	–	1	–	–	–	–	000	0	1	0	0
	10	011	–	0	–	–	–	–	011	0	1	0	0
S_4	11	100	0	–	–	–	–	–	100	0	0	0	0
	12	100	1	–	–	–	–	–	101	0	0	0	0
S_5	13	101	0	–	0	–	–	0	100	0	0	0	0
	14	101	0	–	1	–	–	0	011	0	0	0	0
	15	101	0	–	–	–	–	1	010	0	0	0	0
	16	101	1	–	–	–	–	–	101	0	0	1	0
	17	110	–	–	–	–	–	–	---	–	–	–	–
	18	111	–	–	–	–	–	–	---	–	–	–	–

(This is **not** a state transition table!)

Dice game: **realisation** with a PLA and flip-flops…

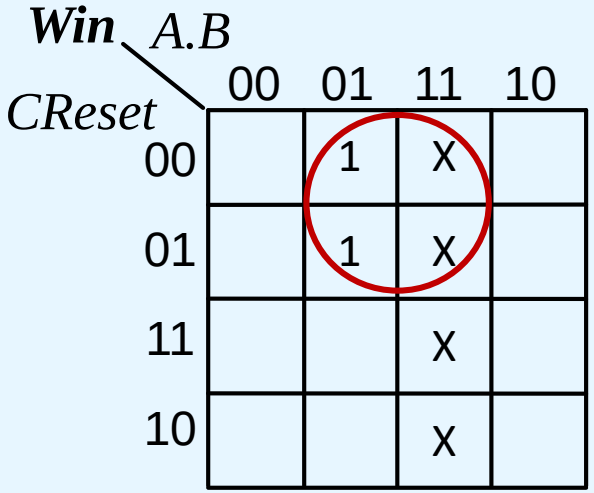
		Inputs							Outputs				
		ABC	*Rb*	*Reset*	D_7	D_{711}	D_{2312}	*Eq*	$A^+B^+C^+$	*Win*	*Lose*	*Roll*	*Sp*
S_0	1	000	0	–	–	–	–	–	000	0	0	0	0
	2	000	1	–	–	–	–	–	001	0	0	0	0
S_1	3	001	1	–	–	–	–	–	001	0	0	1	0
	4	001	0	–	–	0	0	–	100	0	0	0	1
	5	001	0	–	–	0	1	–	011	0	0	0	0
	6	001	0	–	–	1	0	–	010	0	0	0	0
S_2	7	010	–	0	–	–	–	–	010	1	0	0	0
	8	010	–	1	–	–	–	–	000	1	0	0	0
S_3	9	011	–	1	–	–	–	–	000	0	1	0	0
	10	011	–	0	–	–	–	–	011	0	1	0	0
S_4	11	100	0	–	–	–	–	–	100	0	0	0	0
	12	100	1	–	–	–	–	–	101	0	0	0	0
S_5	13	101	0	–	0	–	–	0	100	0	0	0	0
	14	101	0	–	1	–	–	0	011	0	0	0	0
	15	101	0	–	–	–	–	1	010	0	0	0	0
	16	101	1	–	–	–	–	–	101	0	0	1	0
	17	110	–	–	–	–	–	–	---	–	–	–	–
	18	111	–	–	–	–	–	–	---	–	–	–	–

PLA table

Next state / output equations…?

E.g.

Win = *B.C'*



Win = *B.C'*

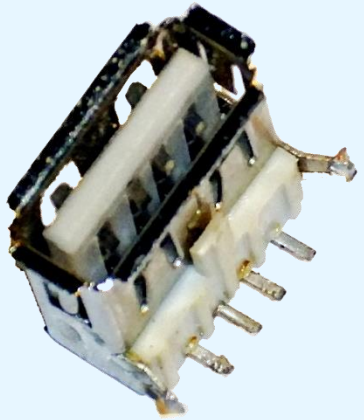
UNIVERSITY OF LIVERPOOL

Serial Data Transmission

Binary data is frequently transmitted between computers as a serial stream of bits…

- bit by bit
- one bit at a time

USB 1 and 2 (Universal Serial Bus)



Vcc, **Data+, Data-**, Gnd

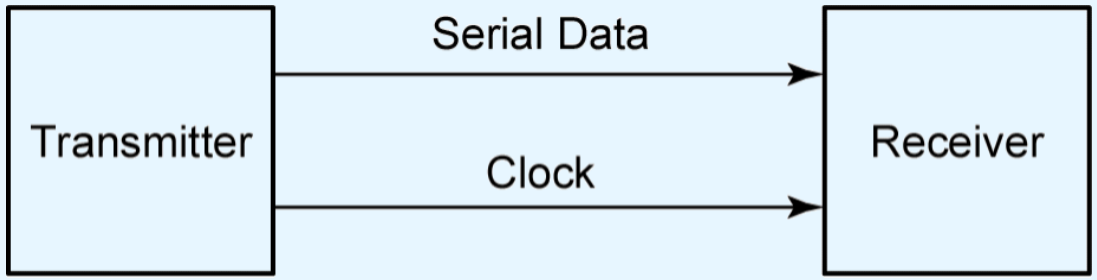
RS232

https://commons.wikimedia.org/wiki/File:USB_2.0_female_01.jpg

https://www.flickr.com/photos/79053562@N00/2235412540

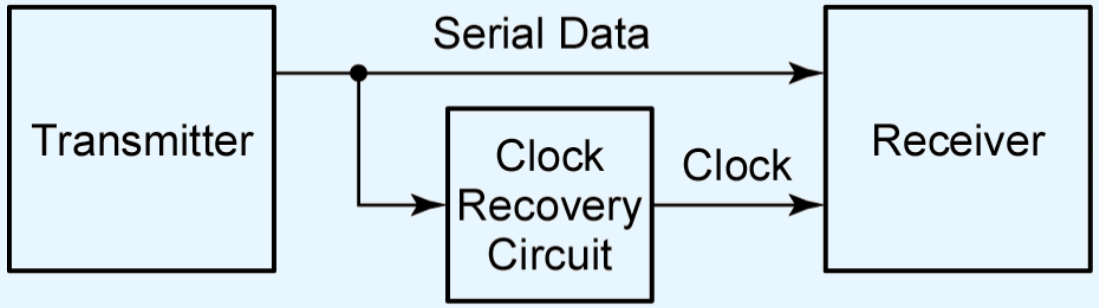
Serial Data Transmission

A clock signal is often transmitted along with the data, so the receiver can read the data at the proper time.



Serial Data Transmission

Alternatively only the serial data is transmitted, and a **clock recovery circuit** (“digital phase-locked loop”) is used to regenerate the clock signal at the receiver.



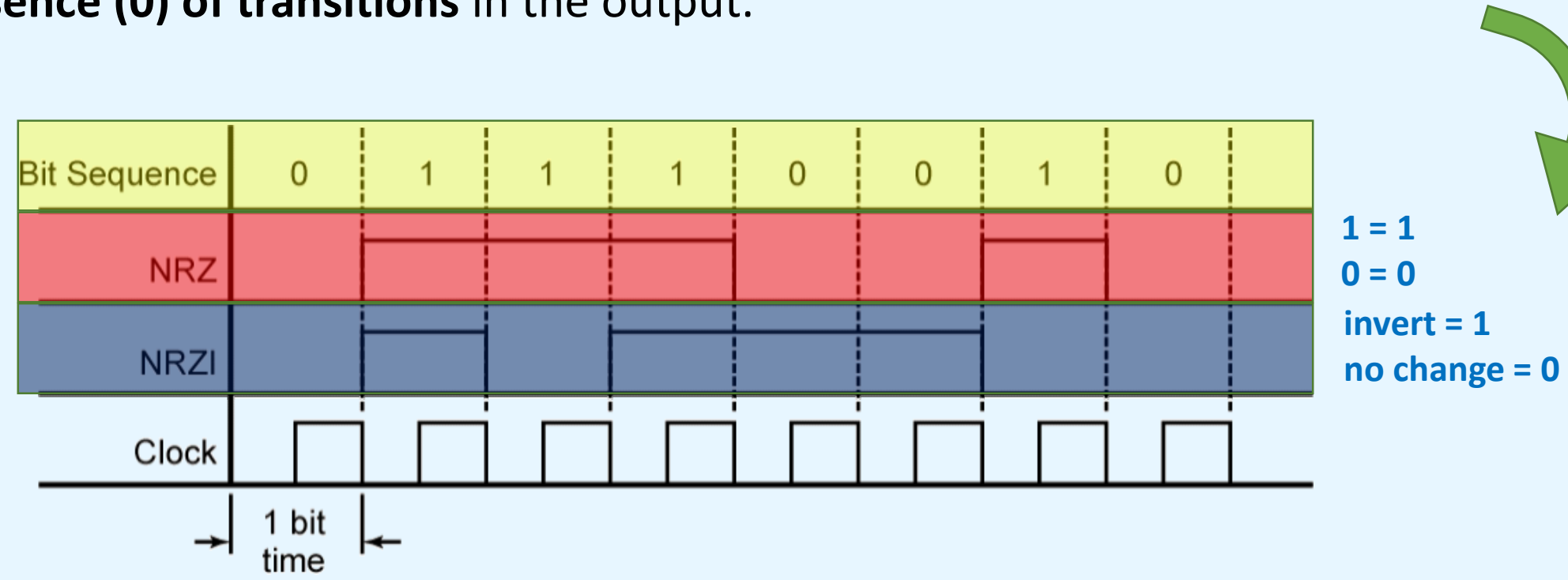
More transitions (0 to 1, 1 to 0) make it easier to recover the clock.

Serial Coding Schemes

Here are four different coding schemes for serial data:

NRZ (non-return-to-zero) code: each bit is transmitted for one bit time, without any change.

NRZI (non-return-to-zero-inverted) code: data is encoded by the **presence (1) or absence (0) of transitions** in the output.



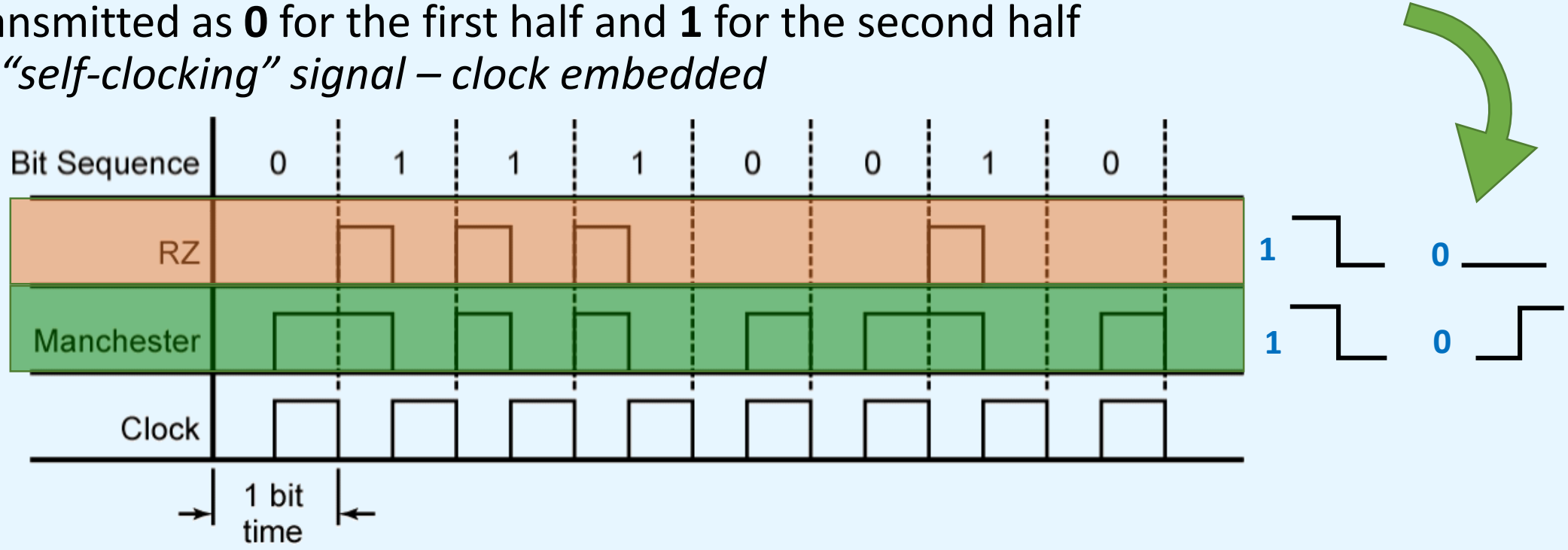
Serial Coding Schemes

RZ (return-to-zero) code:

- **"1"** is transmitted as **1** for the first half of the bit time and **0** for the second half
- **"0"** is transmitted as **0** for full bit-time

Manchester code:

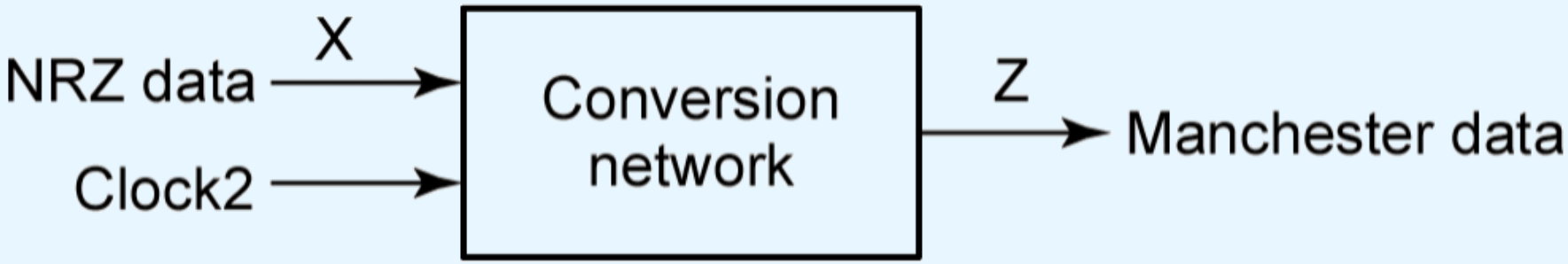
- **"1"** is transmitted as **1** for the 1st half of the bit time and **0** for the 2nd half of the cycle
- **"0"** is transmitted as **0** for the first half and **1** for the second half
- *This is a "self-clocking" signal – clock embedded*



NRZ to Manchester Code Conversion

Advantage: when the original bit sequence has a long string of 1's or 0's, Manchester code has more transitions and this makes it easier to recover the clock signal.

We may wish to convert NRZ to Manchester code.



Use a clock 'Clock2' that is twice the frequency of the basic clock.

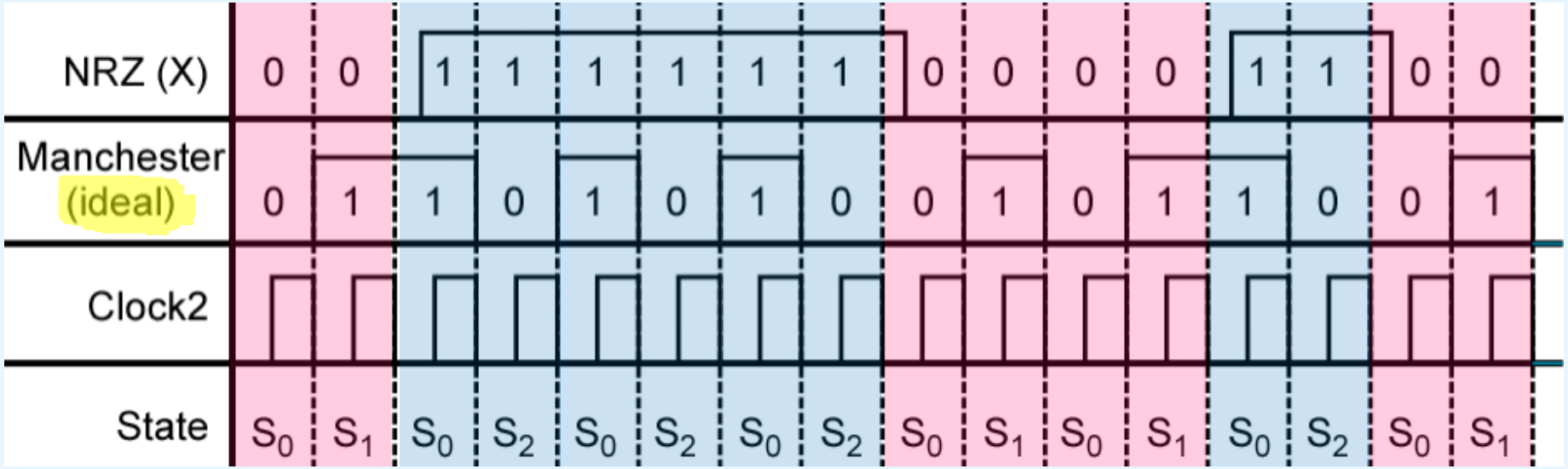
Now, all changes in state will occur on the same edge of Clock2.

NRZ to Manchester Code Conversion: **Mealy Machine**

00 = 0 11 = 1

01 = 0 10 = 1

Falling-edge-triggered



State assignment:

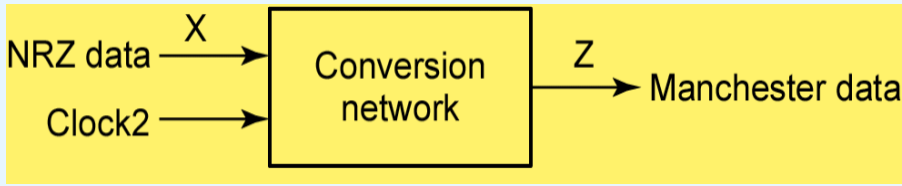
$S_0 \rightarrow S_1$ is associated with a Manchester 0; $S_0 \rightarrow S_2$ is associated with a Manchester 1.
S_0: "reset" state (follows S_1 or S_2, awaits next state S_1 or S_2)
*Output (Manchester 1 or 0) is a function of not just the **state**, but of the **input***

If NRZ bit is 0, will be 0 for two+ Clock2 periods. If it is 1, will be 1 for two+ Clock2 periods.

So, starting from the reset state S_0 there are only two possible **input (NRZ)** sequences: 00 or 11

UNIVERSITY OF LIVERPOOL

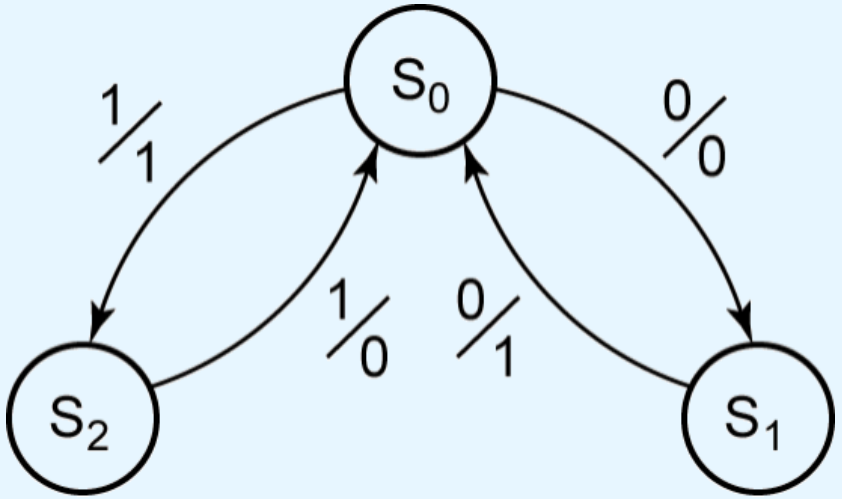
NRZ to Manchester Code Conversion **(Mealy Machine)**



If the first input is 0 the output is 0 and at the end of the Clock2 period, the circuit goes to S_1 . The second input must be 0 & the circuit comes back to S_0 & output becomes 1.

Nearly the same story for the 11 input.

State graph



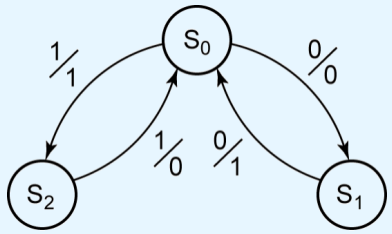
State table

Present State	Next State		Present Output Z	
	X=0	X=1	X=0	X=1
S_0	S_1	S_2	0	1
S_1	S_0	-	1	-
S_2	-	S_0	-	0

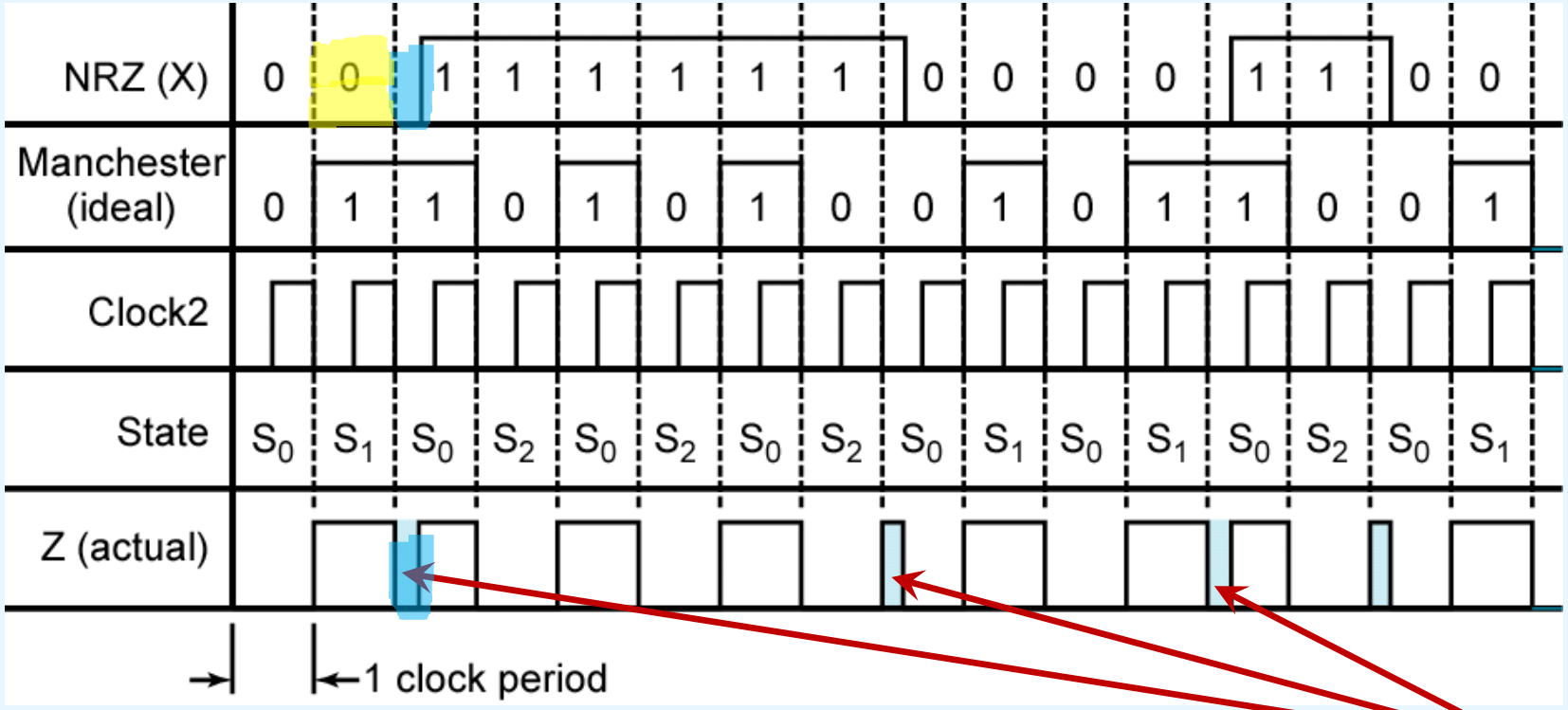
(Dash – we don't care; input sequence never occurs)

UNIVERSITY OF LIVERPOOL

NRZ to Manchester Code Conversion **(Mealy Machine)**



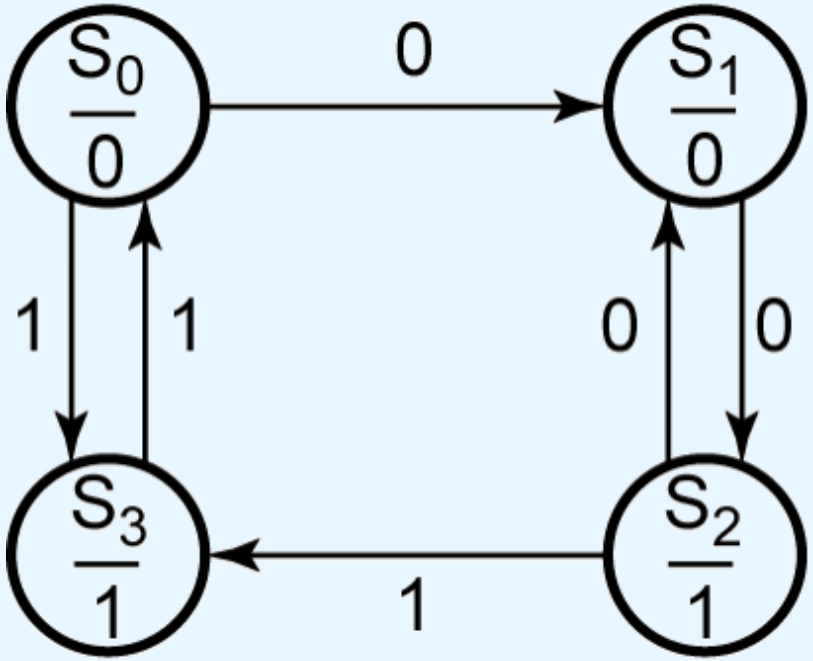
Present State	Next State		Present Output Z	
	X=0	X=1	X=0	X=1
S_0	S_1	S_2	0	1
S_1	S_0	-	1	-
S_2	-	S_0	-	0



If input signal is not exactly synchronised with the clock … glitches in output.

Solution: redesign circuit in **Moore** form.

NRZ to Manchester Code Conversion: **Moore Machine**

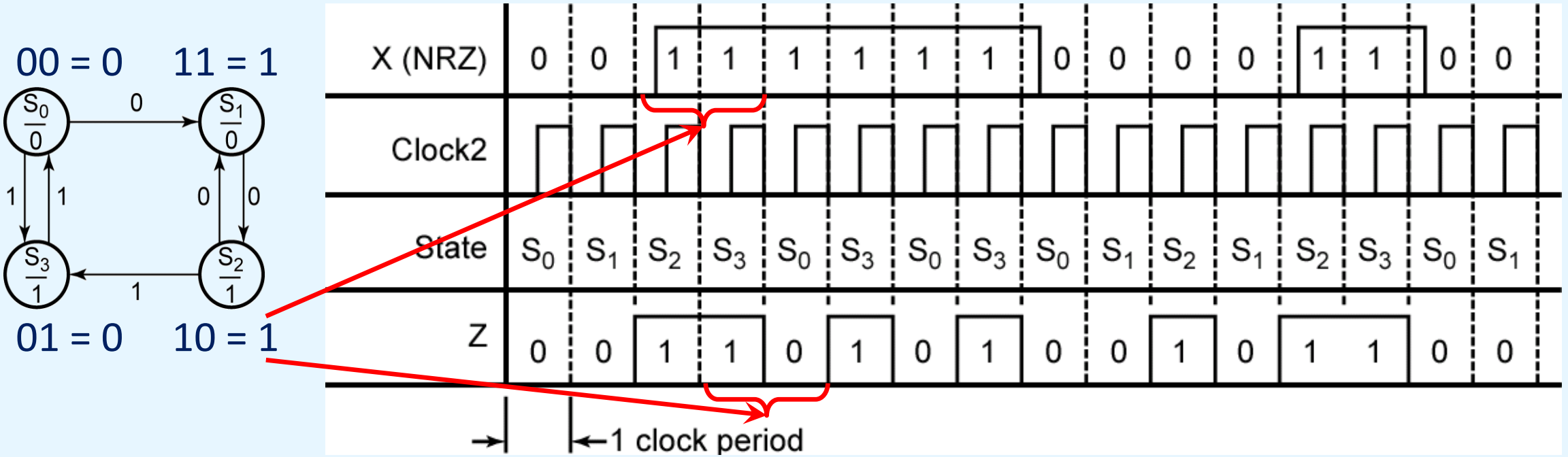


State Diagram

Present State	Next State		Present Output
	X=0	**X=1**	**Z**
S_0	S_1	S_3	0
S_1	S_2	-	0
S_2	S_1	S_3	1
S_3	-	S_0	1

State Table

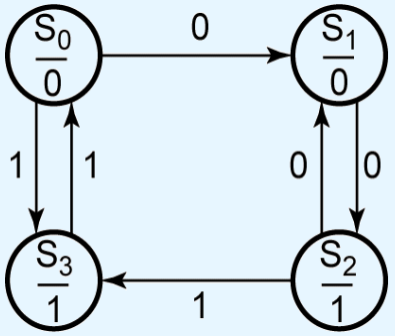
NRZ to Manchester Code Conversion: **Moore Machine**



Moore circuit output waits until after the active edge of the clock, so glitches are avoided, but the output is delayed by a single **Clock2** period.

NRZ to Manchester Code Conversion: **Moore Machine Encoded**

State	BA
S_0	00
S_1	01
S_2	10
S_3	11



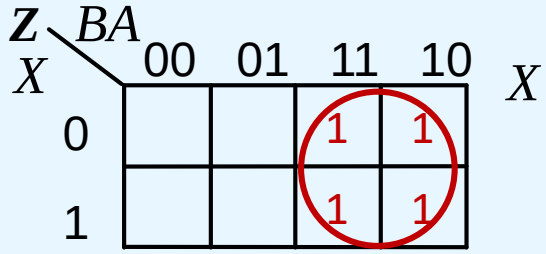
State transition table

Present State	Present input	Next State	Present Output
BA	***X***	***B^+A^+***	**Z**
00	0	01	0
00	1	11	0
01	0	10	0
01	1	-	0
10	0	01	1
10	1	11	1
11	0	-	1
11	1	00	1

Next state / output equations…?

Output:

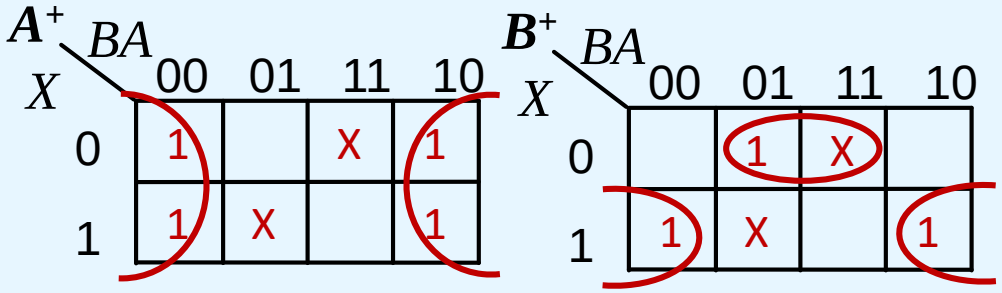
$$Z = B$$



Next state:

$$A^+ = A'$$

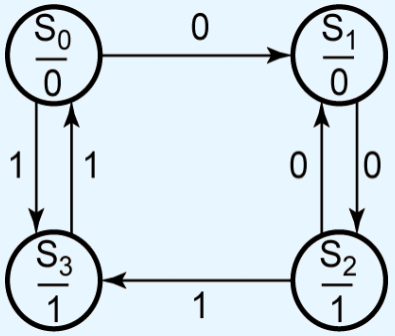
$$B^+ = A'.X + A.X'$$



UNIVERSITY OF LIVERPOOL

NRZ to Manchester Code Conversion: Moore Machine One Hot

State	DCBA
S_0	0001
S_1	0010
S_2	0100
S_3	1000



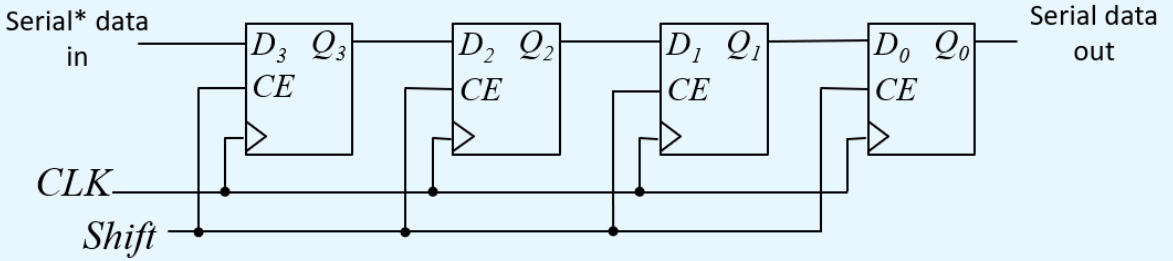
State transition table

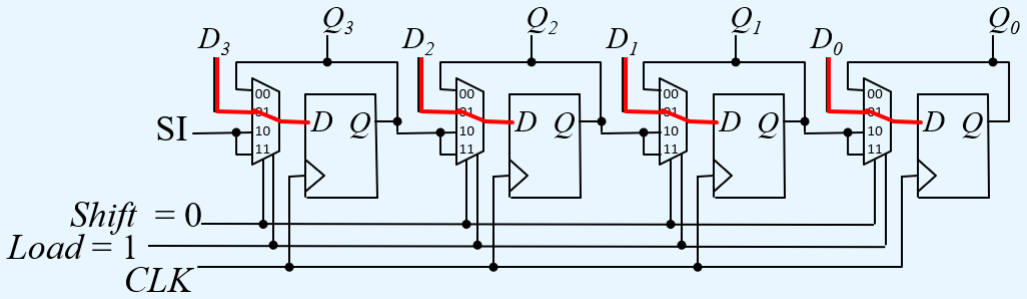
Present State	Present input	Next State	Present Output
DCBA	***X***	$D^+C^+B^+A^+$	**Z**
0001	0	0010	0
0001	1	1000	0
0010	0	0100	0
0010	1	-	0
0100	0	0010	1
0100	1	1000	1
1000	0	-	1
1000	1	0001	1

UNIVERSITY OF LIVERPOOL

Supplementary notes about shift registers (Lec20-21 / Digital 7-8 reprise)

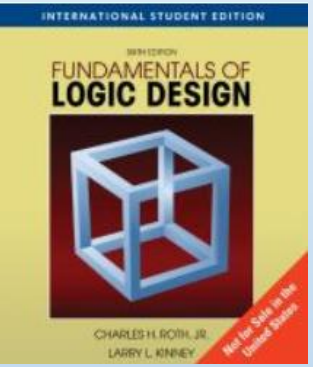
- Serial in Serial out (SISO)
  - Data loading and retrieval are bit-by-bit
  - E.g. data can only be read out at the last flip-flop, one bit at a time
- Parallel in Parallel out (PIPO)
  - Data loading and retrieval can be done in parallel
  - E.g. all bits in the word can be loaded in one clock cycle
- Data shifting to left/right
  - A possible application:
    - Left shift → $\times 2$
    - Right shift → $\div 2$





UNIVERSITY OF LIVERPOOL

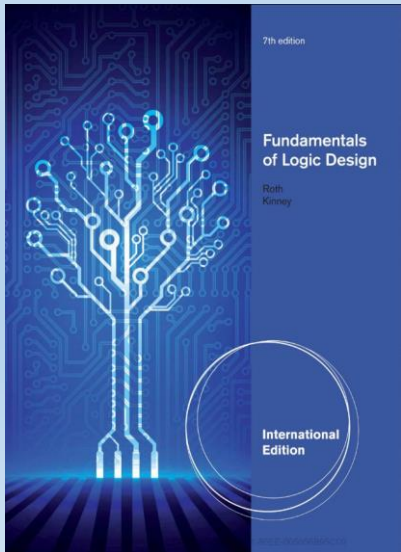
Summary and suggested reading



- Section 19.2 Deriving ASM charts
- Section 19.3 Realising ASM charts
- Section 14.4 Code conversion (serial data)

Next lecture
(Lec 25 / Digital 12):

The Quine-McCluskey method.



Roth and Kinney *Fundamentals of Logic Design*

….. 7th ed. is available as an e-book!

UNIVERSITY OF LIVERPOOL