ELEC-H-310
Digital electronics

Lecture05
Asynchronous sequential circuits

Dragomir MILOJEVIC

2021

Plan

1. Race conditions

2. Asynchronous circuits: solving race conditions

3. Example of asynchronous implementation

4. Meally machines

5. Synthesis of Moore AND Meally machines
   (for the same problem)

1. Race conditions

State table is first encoded

Let's take the following optimised state table
(**How do we know it is optimised?)**

ab

	00	01	11	10
1	**1**	2	3	2
2	1	**2**	4	**2**
3	1	**3**	**3**	4
4	1	3	**4**	**4**

ab

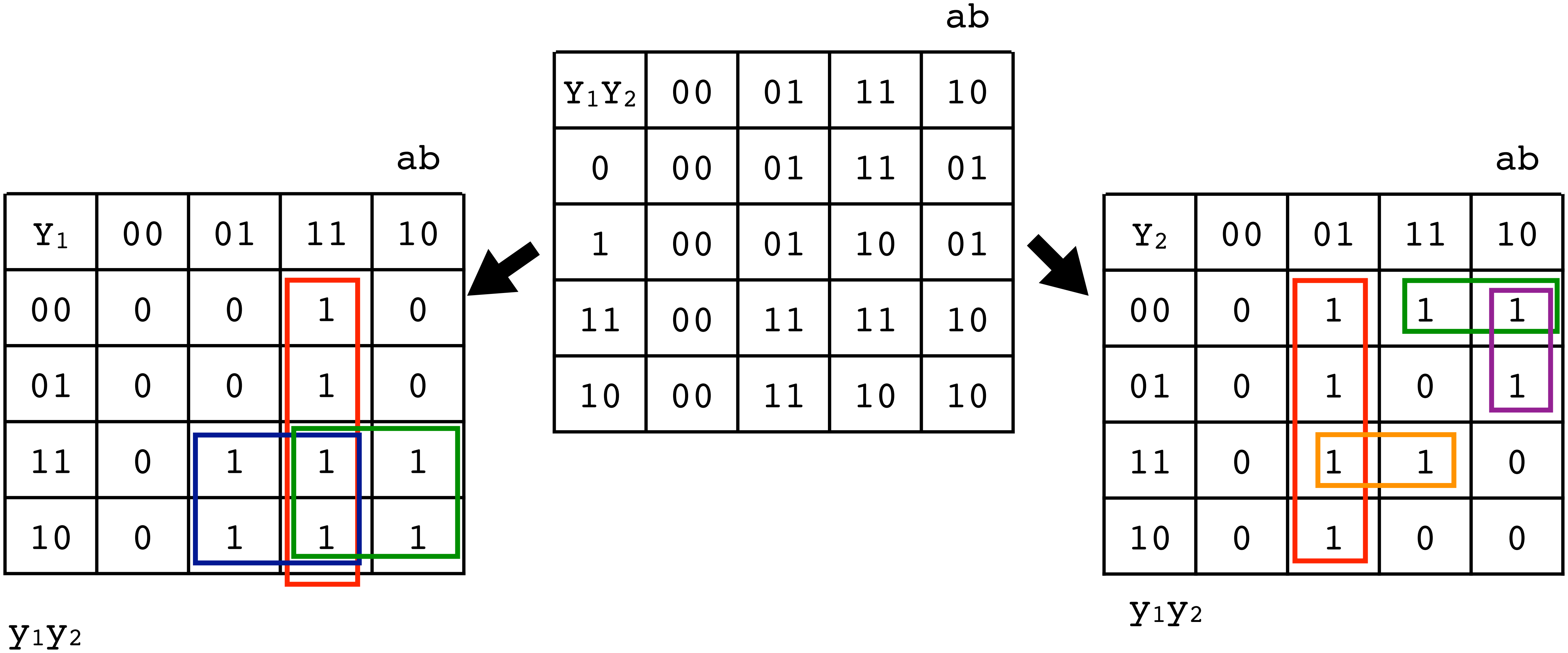
Y_1Y_2	00	01	11	10
00	**00**	01	11	01
01	00	**01**	10	**01**
11	00	**11**	**11**	10
10	00	11	**10**	**10**

Let's use the following encoding:
1→00, 2→01, 3→11, 4→10

Optimised & encoded state table

Then we derive state equations

K-Maps and optimised expressions



ab

Y_1Y_2	00	01	11	10
0	00	01	11	01
1	00	01	10	01
11	00	11	11	10
10	00	11	10	10

ab

Y_1	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	1	1	1
10	0	1	1	1

y_1y_2

ab

Y_2	00	01	11	10
00	0	1	1	1
01	0	1	0	1
11	0	1	1	0
10	0	1	0	0

y_1y_2

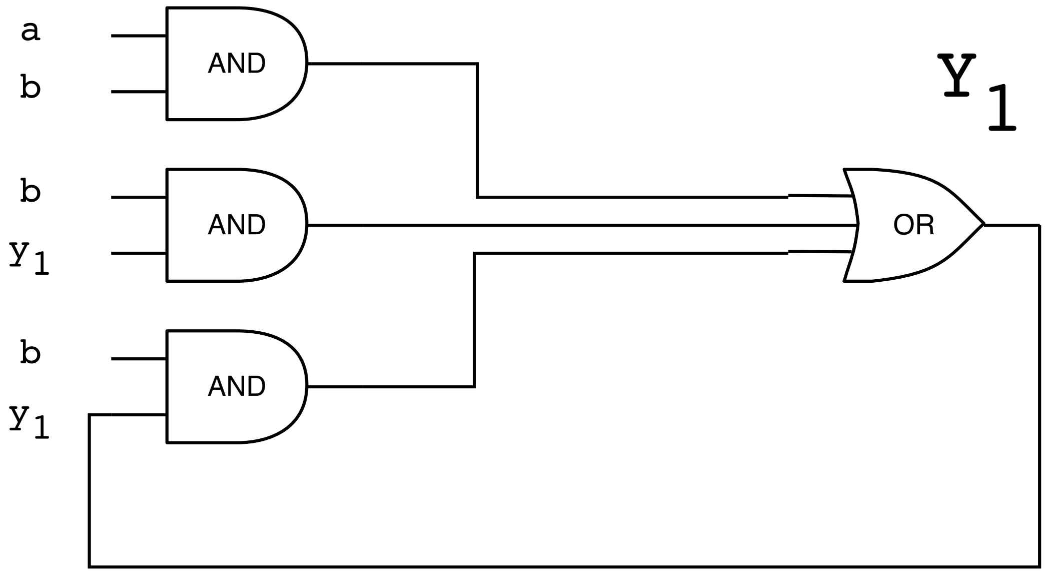
$$Y_1=ab+y_1b+y_1a$$

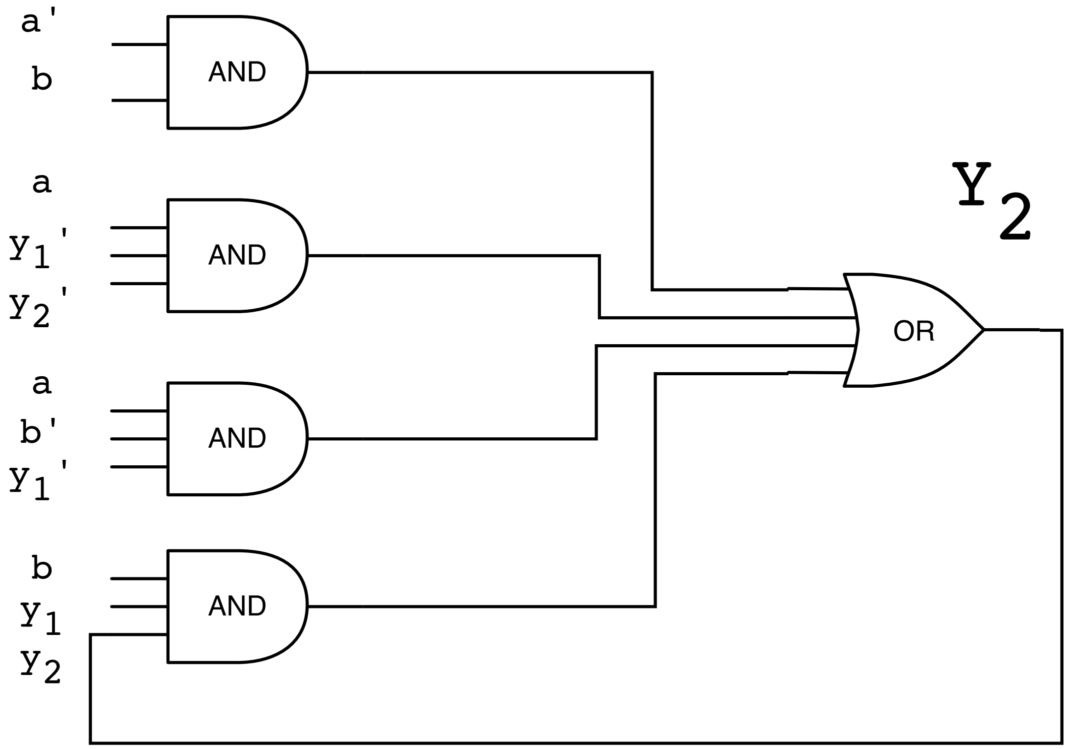
$$Y_2=a'b+y_1'y_2'a$$
$$+\ ab'y_1'+y_1y_2b$$

Equations are used to build the circuit

- Two expressions result in **two independent circuits** with **feedback loops** to allow predicted future to become present:

$$y_i \leftarrow Y_i$$



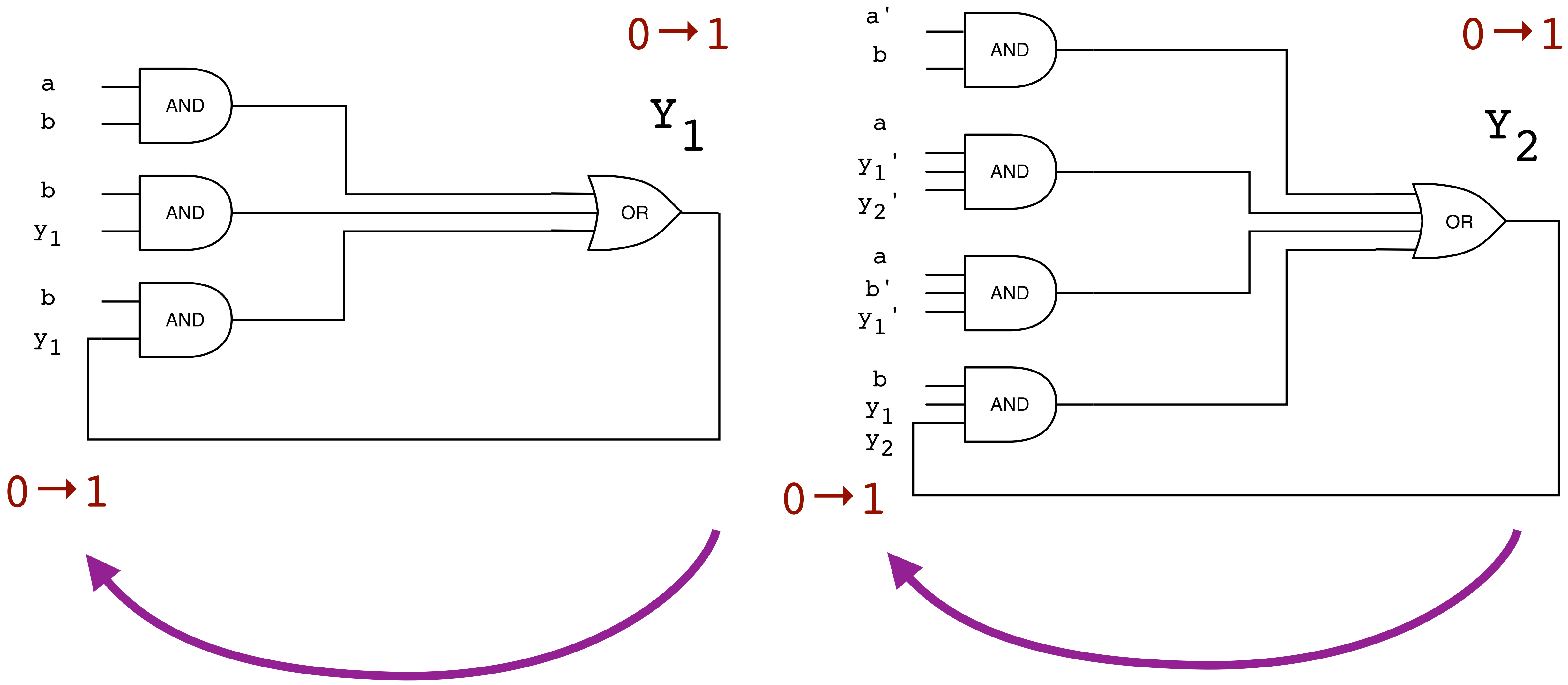


$Y_1 = ab + y_1 b + y_1 a$

$Y_2 = a'b + y_1'y_2'a + ab'y_1' + y_1 y_2 b$

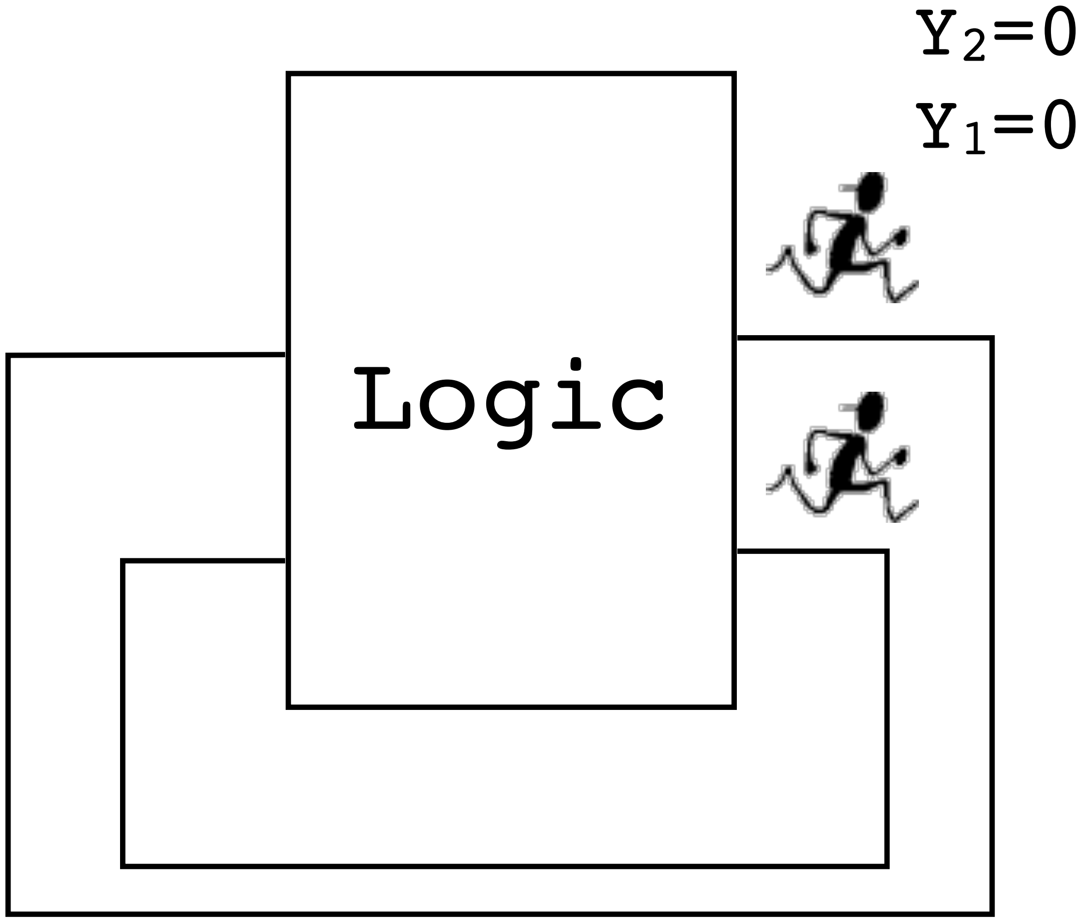
Race conditions at circuit level

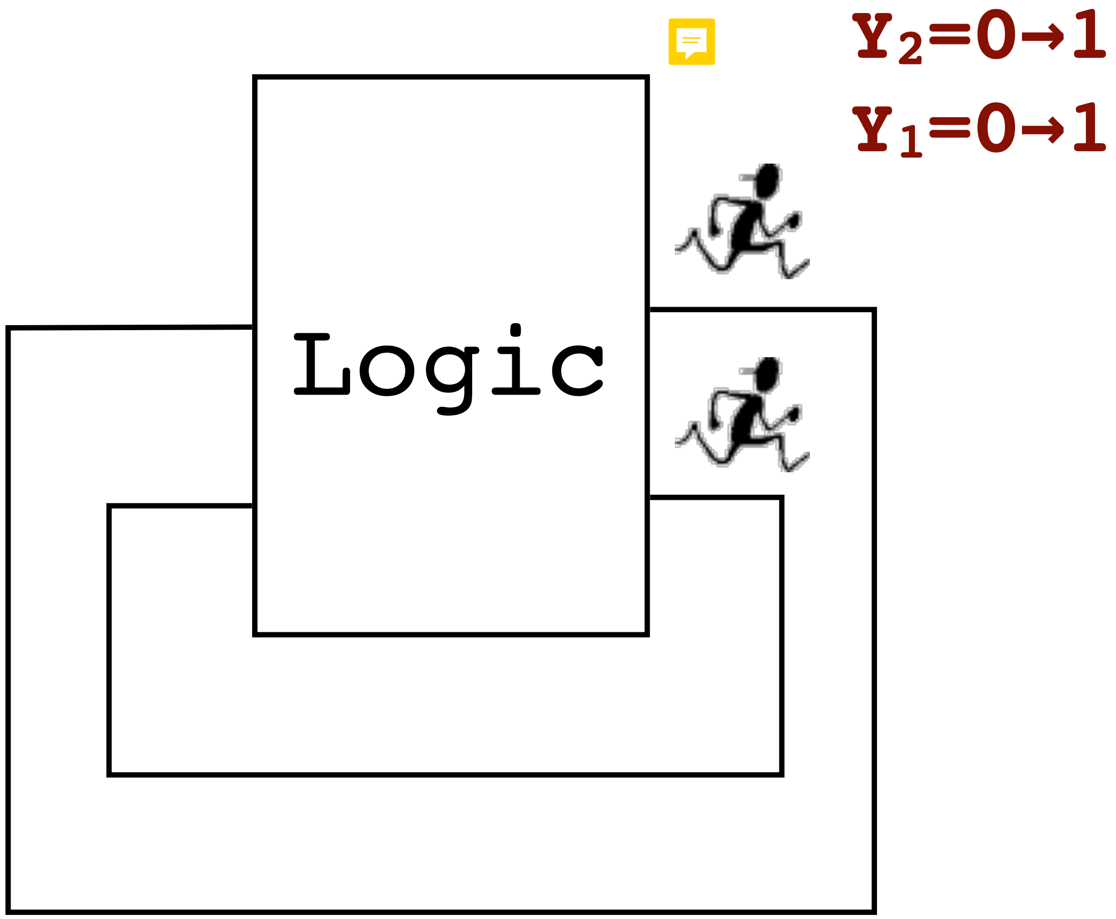
During the update of the state variables, when predicted future becomes present $(y_i \leftarrow Y_i)$ it is possible that **two (or more) state variables** will have to switch from 0 to 1 **at the same time**



Race conditions (analogy) explained (1/2)

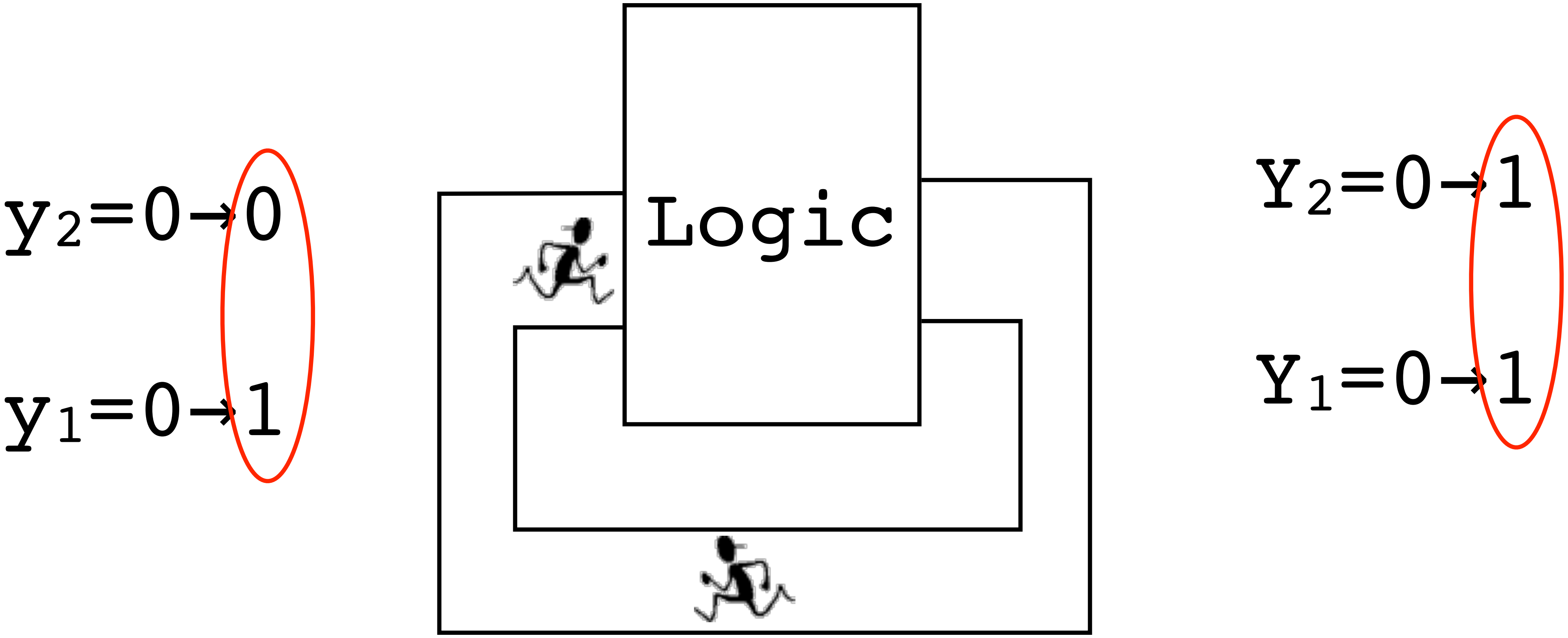
- Our circuit has 2 state variables that we can see as two runners
- When input changes, internal logic will compute **the future state**
- If the current/future state should be the transition from `00` to `11`, both runners should start their “race” at the same time …





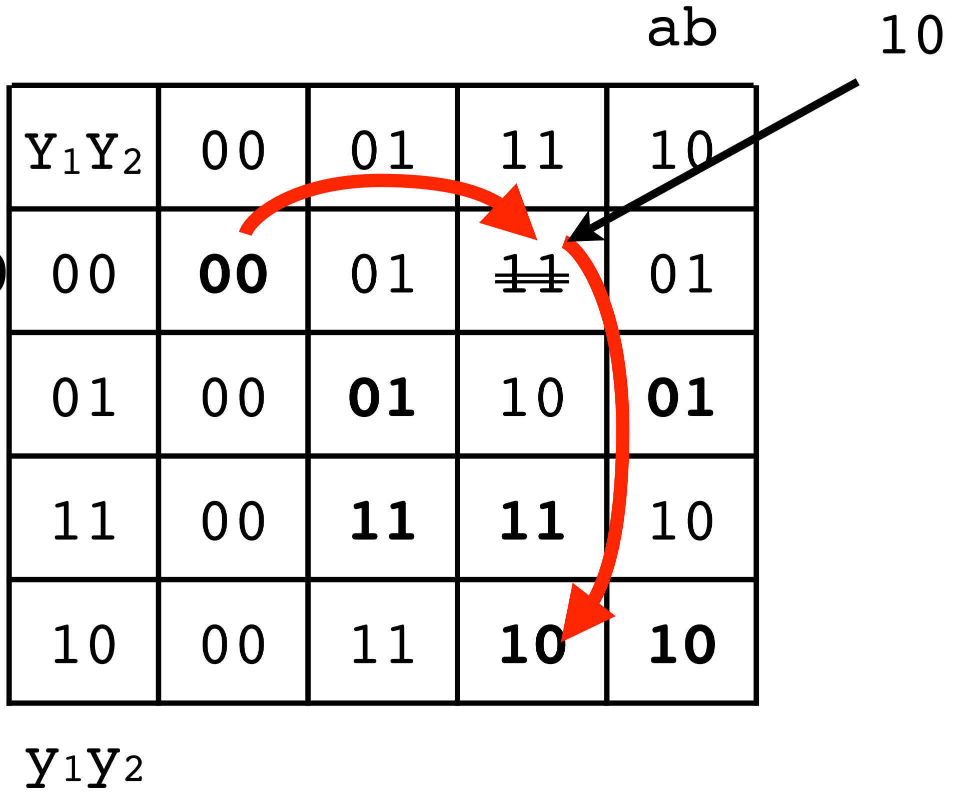
Race conditions (analogy) explained (2/2)

- Two “running lanes” (i.e. wires) don’t have the same length, the speed of two runners is never the same (temperature gradient), etc.
- This means that one of the two runners will arrive **before** the other
- If, instead of `11`, the new destination (seen by the system) becomes `10` `(`or `01)`, **the system doesn’t respect the initial spec !**



Race conditions: back to state table

- System in 00, input has changed, future state should become 11
- Due to race conditions instead of 11 system see's 10
- In our case state 10 is stable, the system will stay there ! (transition towards 01 has the same effect, we end up in another stable state)
- Initially our spec says: "*when system in state* 00 *& when input* ab=11, *then future state should be* 11
- But due to race conditions system ends up in state 10 → final state is altered
- System spect is not followed: **THIS IS BAD !**
- We need to avoid this !!!



Y_1Y_2	00	01	11	10
00	**00**	01	~~11~~	01
01	00	**01**	10	**01**
11	00	**11**	**11**	10
10	00	11	**10**	**10**

y_1y_2

Race conditions & solutions

- **Def. Instantaneous change of 2 or more state variables during the state transition**
- In sequential systems it is mandatory that we **don't have no race** conditions to ensure deterministic automata behaviour
- Two different ways to solve this, two different logic circuit classes:
  - **Asynchronous logic circuits** – we make sure there are no races during state transitions (state transitions have up to 1 internal variable change at most); this is what we do first;

  - **Synchronous logic circuits** – we use memory to synchronise state values; any transient will be "filtered" out (ignored) on the input of these memories (think of a D Flip-Flop); we may have races in the state tables they are now harmless; this is what we do in the next lecture;

Race conditions in state tables

Transitions subject to **race conditions** marked in red

Hamming distance between present and future > 1

ab

Y_1Y_2	00	01	11	10
00	**00**		11	
01		**01**		10
11	00		**11**	
10		01		**10**

y_1y_2

Don’t mix things up: inputs **can change instantaneously** (they are random variables)

Below **not a race condition!**

ab

Y_1Y_2	00	01	11	10
00	**00**		11	
01		**01**		10
11	00		**11**	
10		10		**10**

y_1y_2

2. Asynchronous circuits: solving race conditions

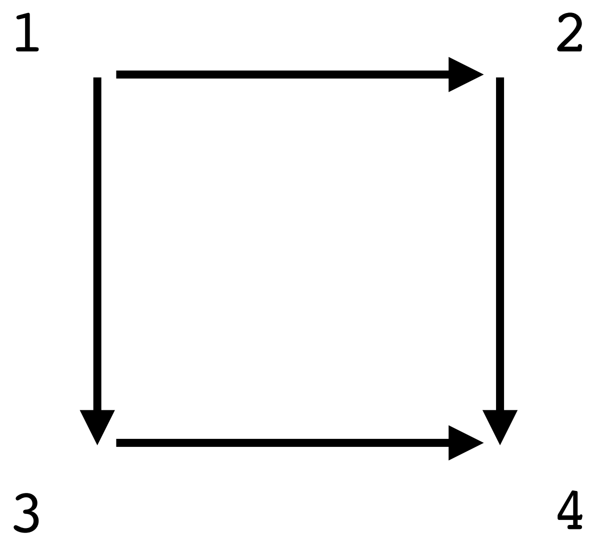
Solving race conditions

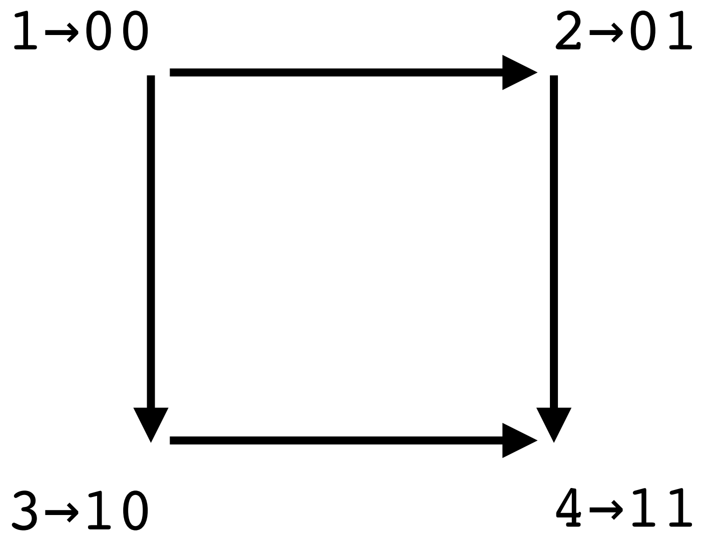
- Different methods called (very originally) here: **Method1**, **Method2** and **Method3**
- To ensure certain optimality of the final system (logic complexity) these methods are applied in the given order:
  - **Method2** applied only if **Method1** doesn't give you the solution
  - **Method3** applied only if **Method1** and **Method2** doesn't give you the solution
- **Race conditions are solved only** when we need to synthesise **asynchronous** systems
- **You can/should leave race** conditions when you are synthesising **synchronous** systems
- Sometimes people are confused about this

Method1 – state encoding

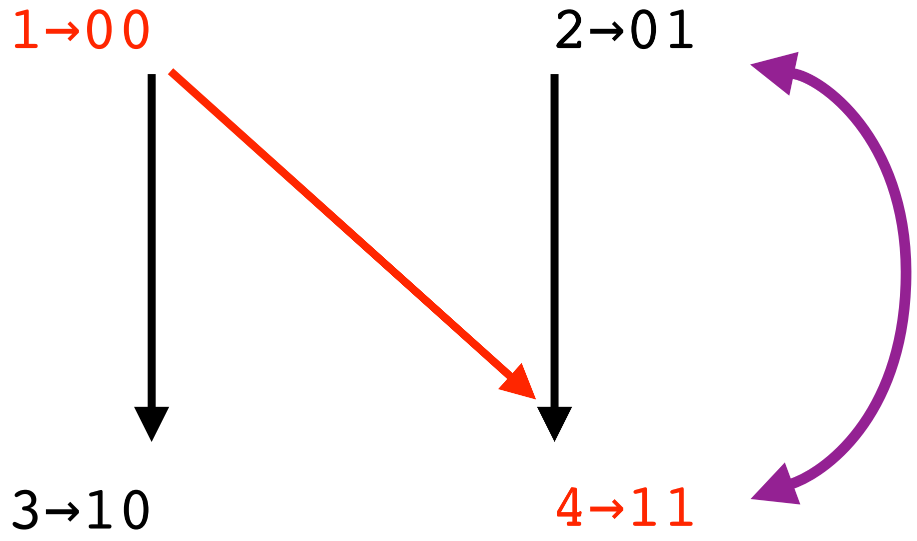
- Encoding choice is free, therefore it is **maybe possible** to find encoding such that encoded table is free of race conditions (this is **not always possible**)
- We want to check in a **systematic way** if this is possible (or not)
- Method using **state encoding graphs** to easily find encoding with no race conditions and show if such encoding exists (or not)
  - Take an n-cube and assign a stable state to each vertex; transitions are shown using arcs (not necessarily oriented); this is similar to state graphs (we omit transition conditions since we are only interested in state connectivity)
  - State assignment respect Hamming distance: adjacent vertices have max Hamming distance = 1!
  - Normal edge – means NO RACE conditions;
    Diagonals – indicate a RACE!

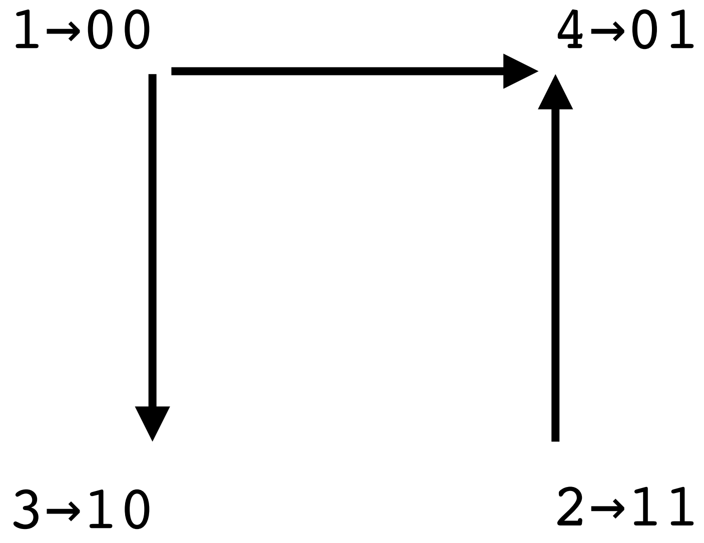
Method1 – state encoding examples





Race condition since we have one diagonal





Here we can swap codes for states 2 & 4 and avoid races

Method1 – state encoding examples

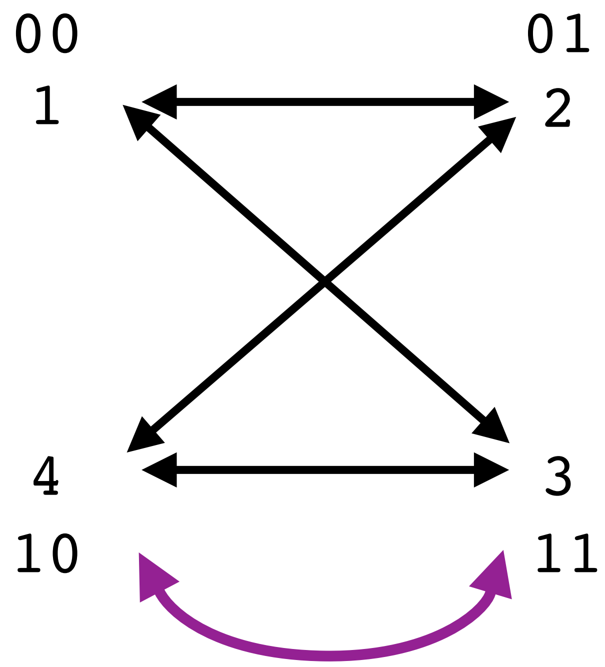
ab

	00	01	11	10
1	**1**	2	3	2
2	**2**	**2**	4	**2**
3	1	**3**	**3**	4
4	2	3	**4**	**4**

	00	01	11	10
00	**00**	01	11	01
01	**01**	**01**	10	**01**
11	00	**11**	**11**	10
10	01	11	**10**	**10**

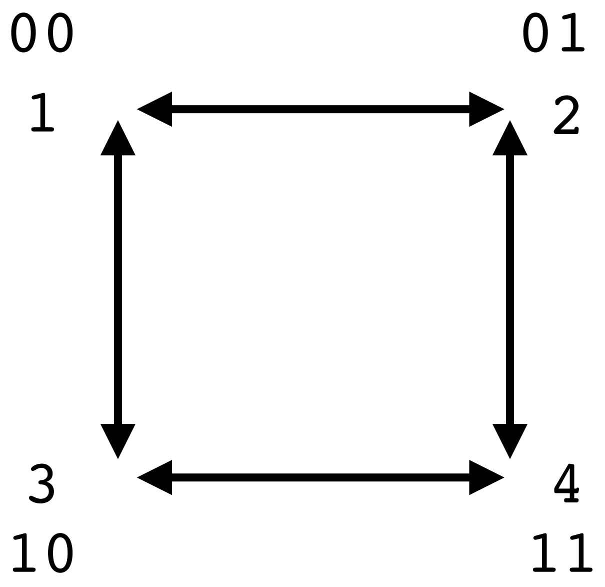
y_1y_2

State encoding graph:



4 race conditions!

Swap two codes:



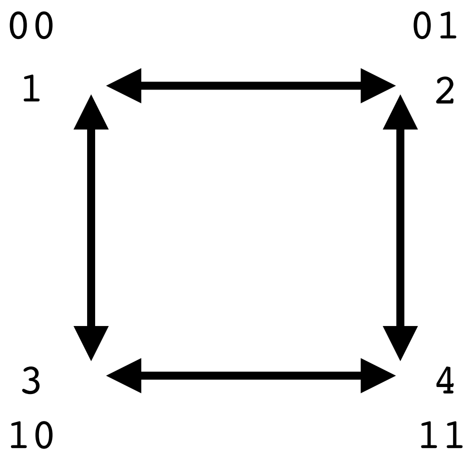
Also 4 other solutions !

Attention !!!

Due to inversion of codes, state table is without races but it is not K-Map!

ab

	00	01	11	10
1	**1**	2	3	2
2	1	**2**	4	**2**
3	1	**3**	**3**	4
4	2	3	**4**	**4**



ab

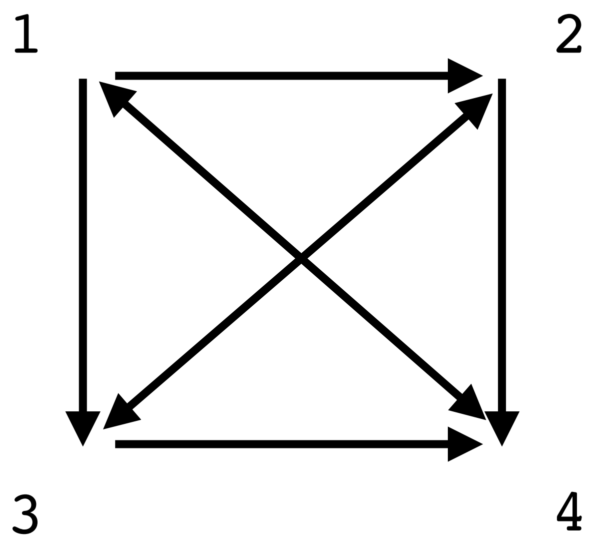
	00	01	11	10
00	**00**	01	10	01
01	00	**01**	11	**01**
10	00	**10**	**10**	11
11	01	10	**11**	**11**

	00	01	11	10
00	**00**	01	10	10
01	00	**01**	11	**10**
10	00	**10**	**10**	11
11	01	10	**11**	**11**

	00	01	11	10
00	**00**	01	10	01
01	00	**01**	11	**01**
11	01	10	**11**	**11**
10	00	**10**	**10**	11

Method2 – modifying transitions

- You can not always find a solution using encoding technique, take the example below:



- No matter what we do, we will always have diagonals
- We need something else → welcome to **Method2** based on the following property: we do not care about what transitions system took, what counts is the source and destination states
- In another words all the following transitions are equivalent since for the same source state we will end up in the same destination state:

1→2→4→**4** or **1**→3→4→**4** or **1**→2→3→5→4→**4**

Method2 – modifying transitions

Using **“don’t care” or existing transitions** – these transitions can be modified to accommodate whatever transition is needed

ab

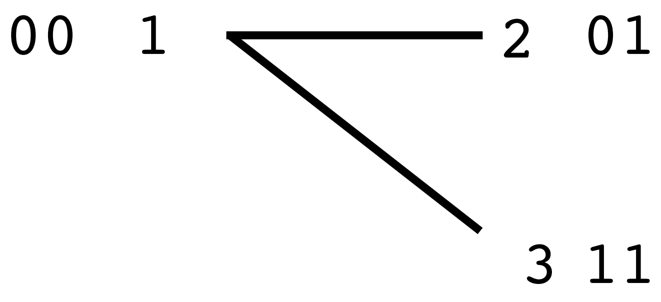
	00	01	11	10
1	**1**	2	3	
2		**2**	–	
3			**3**	
4				

ab

	00	01	11	10
00	**00**	01	~~11~~ 01	
01		**01**	-- 11	
11			**11**	
10				

ab

	00	01	11	10
00	**00**	01	~~11~~ 01	
01		**01**	11	
11			**11**	
10				



00→11→**11**
is now **00**→01→11→**11**
We have extra transition
Is this a problem?

Don’t care could be a valid transition (3 instead of –)

Method3 – adding extra state variable

- If both of the previous methods can't provide solution, we can always add **an extra state variable**
- This is not optimal we know, but there is no other way! (and we can't leave races)
- By doing so we add **extra *2^n* transitions** (*n* is the number of state variables in the beginning)
- These extra transitions are used to solve the race conditions as with **Method2**
- Attention ! We first solve all race conditions that can be solved using **Method2**; only those that still remain will be solved with these new transitions
- This is because if you do not do so, you will over-constraint the system and get much less optimal system in the end

Asynchronous sequential system synthesis

The complete synthesis algorithm:

1. We first establish the state table **from verbal specs**

2. We perform **state reduction** (state optimisation) to reduce the number of states (redundancy during the formalisation process in the previous step); this will save system ressources !

3. State encoding (**Method1**) to chose the appropriate binary codes (eventually without race conditions, not always possible)

4. Solution of the remaining race conditions using, **Method2** or **Method3** (this last method applied only if absolutely necessary, since kills the efforts deployed in step 2)

5. Logic functions for state variables and output synthesis

3. Example of asynchronous implementation

Starting point – optimised state table

ab

Y_1Y	00	01	11	10
1	**1**	2	3	4
2	0	**2**	**2**	4
3	0	4	**3**	**3**
4	0	**4**	**4**	**4**

- First we need to find some encoding
- Blind search or **Method1**
- **One (very important) thing that you can already see in the table on the left?**

State encoding

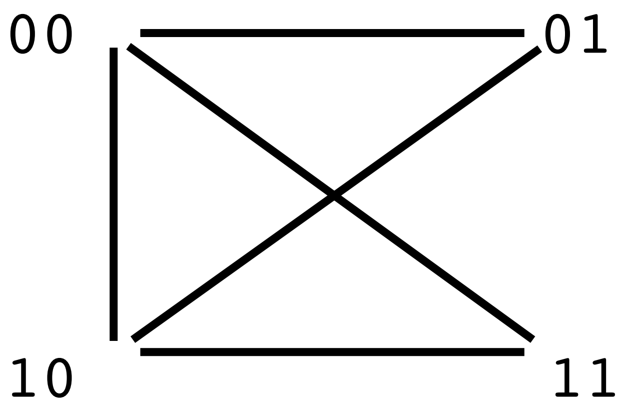
State table analysis: in columns `ab=11` **we have only one transition**

ab

Y_1Y_2	00	01	11	10
00	**00**	01	11	10
01	00	**01**	**01**	10
11	00	10	**11**	**11**
10	00	**10**	**10**	**10**

All states in this column are stable, no mods possible

Method1 – no encoding without races:



Method2 – column `ab=11`, single transition, i.e. it can't be modified

We need to add new state variable !

Adding new state

Adding extra state variable provides new transitions to be used:

ab

Y_1Y_2	00	01	11	10
00	**00**	01	11	10
01	00	**01**	**01**	10
11	00	10	**11**	**11**
10	00	**10**	**10**	**10**

y_1y_2

ab

$Y_1Y_2Y_3$	00	01	11	10
000	**00**	01	11	10
010	00	**01**	**01**	10
110	01	10	**11**	**11**
100	00	**10**	**10**	**10**
001	–	–	–	–
011	–	–	–	–
111	–	–	–	–
101	–	–	–	–

$y_1y_2y_3$

Attention!

This is not a state table!

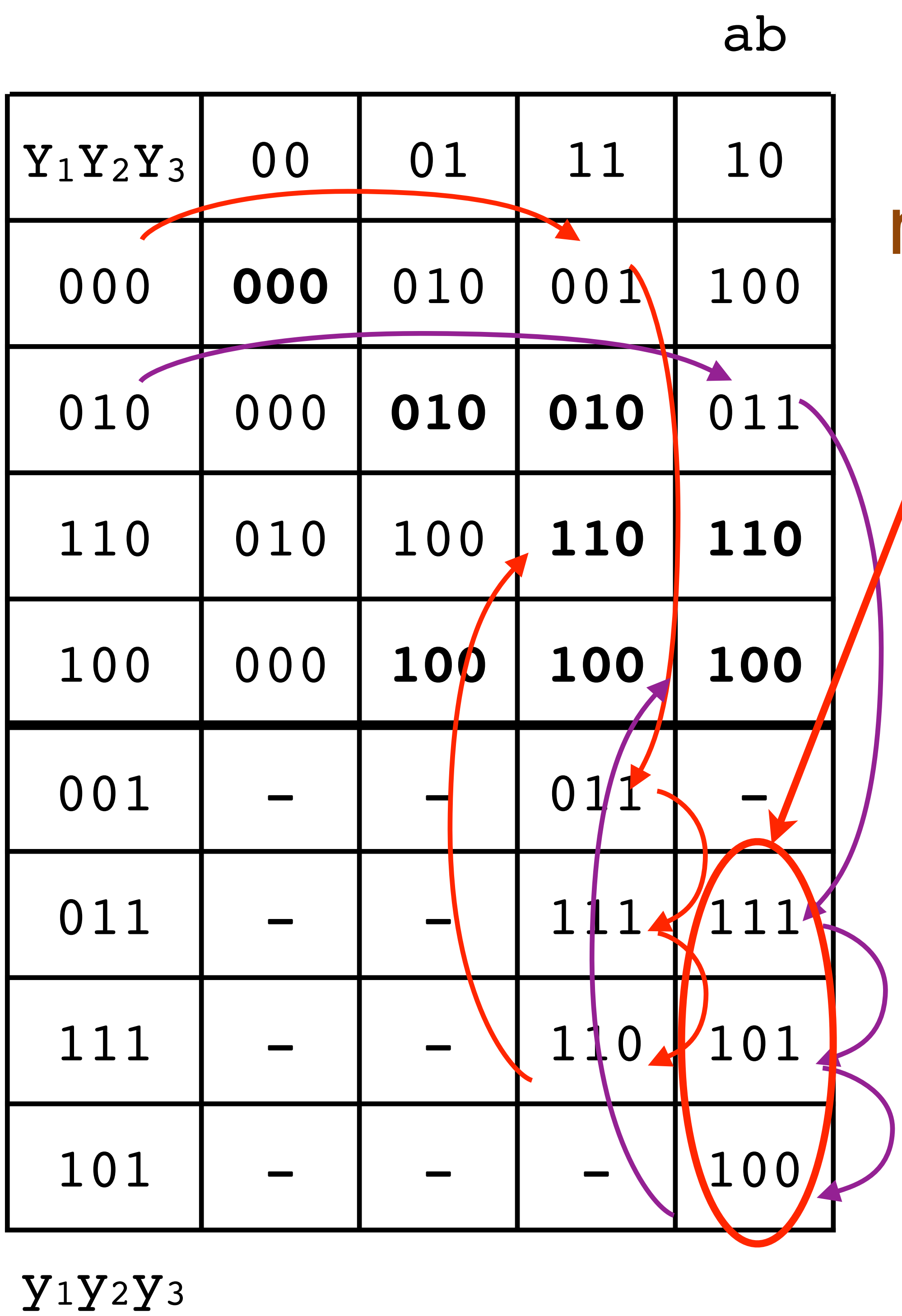
Why?

Using new transitions to solve races

ab

$Y_1Y_2Y_3$	00	01	11	10
000	**000**	010	110	100
010	000	**010**	**010**	100
110	010	100	**110**	**110**
100	000	**100**	**100**	**100**
001	–	–	–	–
011	–	–	–	–
111	–	–	–	–
101	–	–	–	–

$y_1y_2y_3$



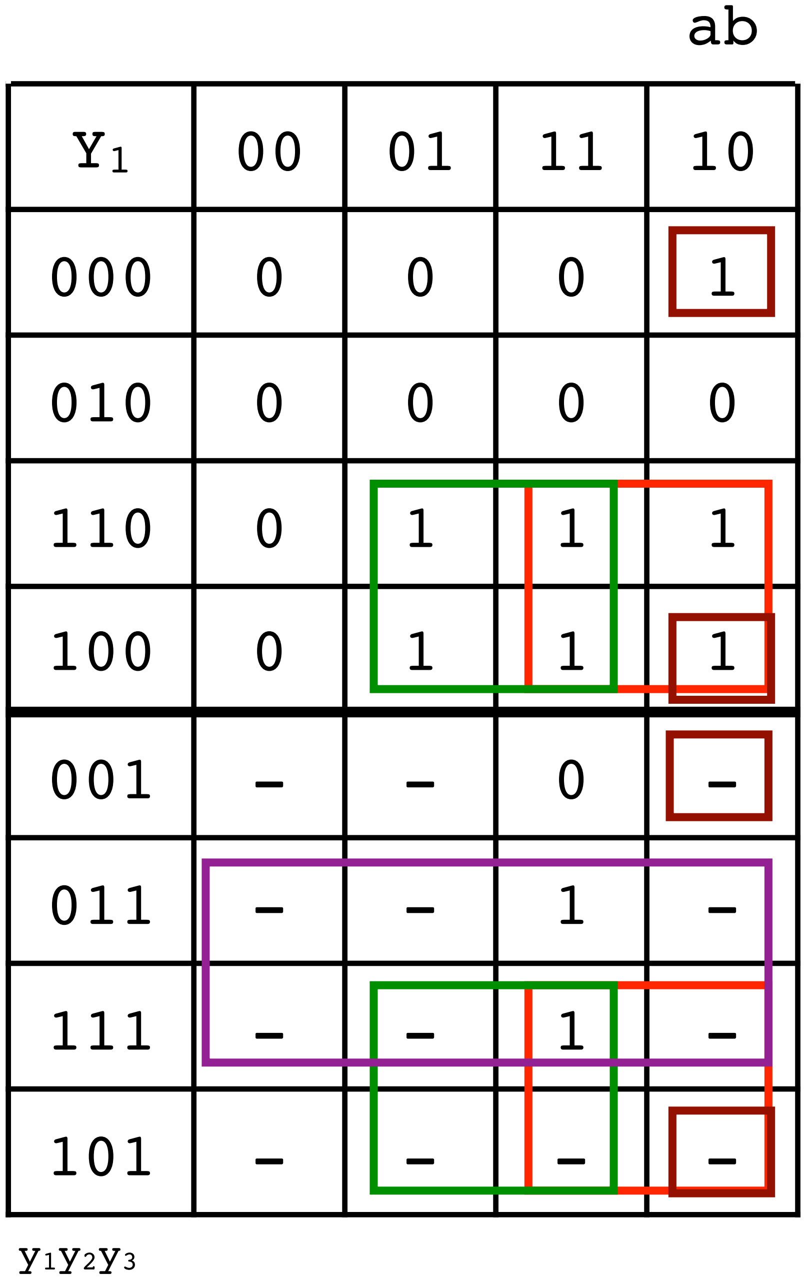
ab

$Y_1Y_2Y_3$	00	01	11	10
000	**000**	010	001	100
010	000	**010**	**010**	011
110	010	100	**110**	**110**
100	000	**100**	**100**	**100**
001	–	–	011	–
011	–	–	111	111
111	–	–	110	101
101	–	–	–	100

$y_1y_2y_3$

Is this necessary?

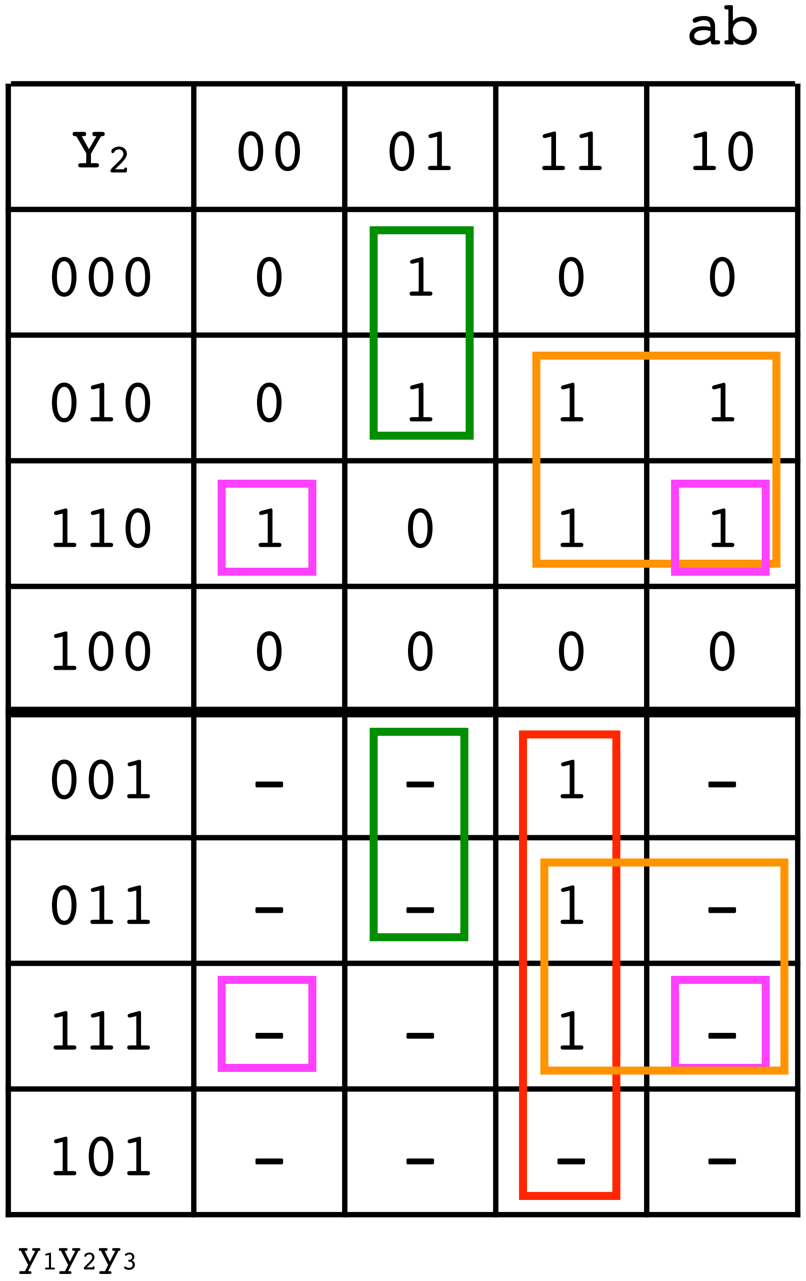
K-Maps optimisation for state functions



ab

Y_1	00	01	11	10
000	0	0	0	1
010	0	0	0	0
110	0	1	1	1
100	0	1	1	1
001	–	–	0	–
011	–	–	1	–
111	–	–	1	–
101	–	–	–	–

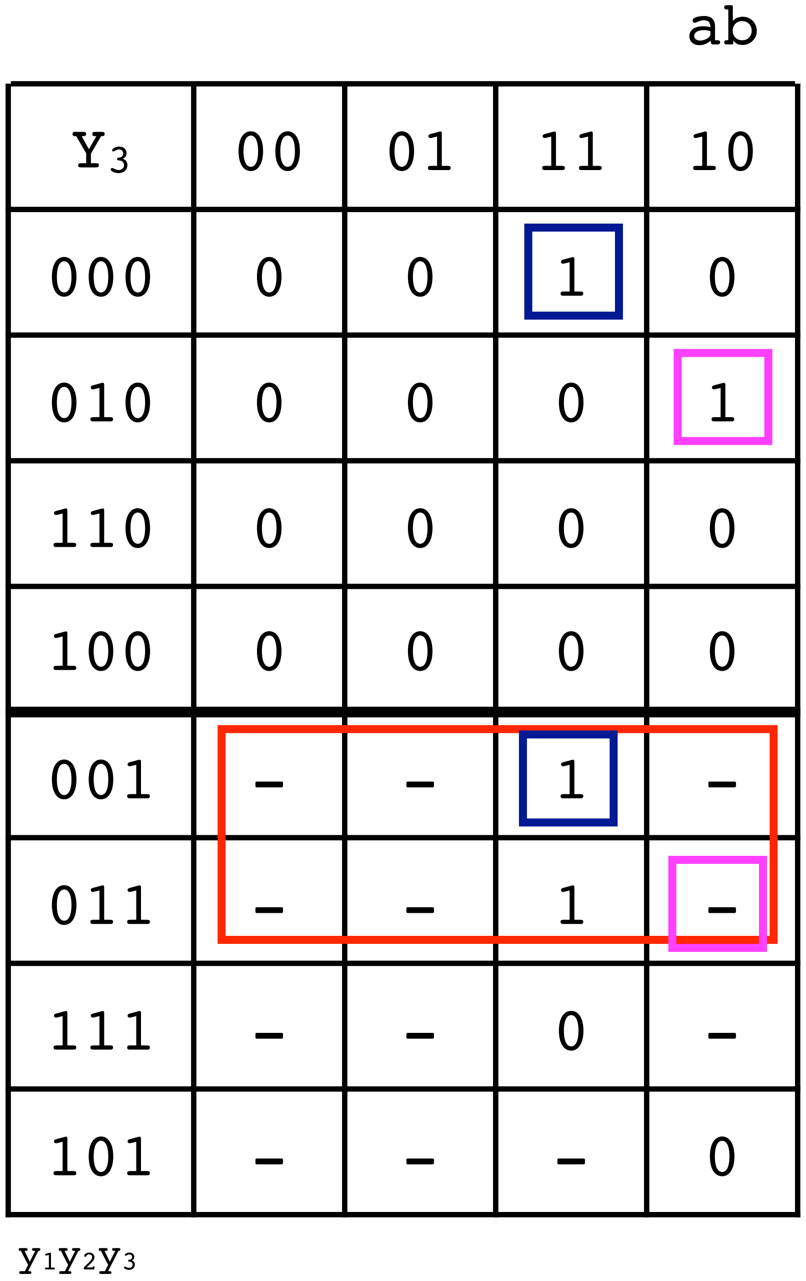
$y_1y_2y_3$



ab

Y_2	00	01	11	10
000	0	1	0	0
010	0	1	1	1
110	1	0	1	1
100	0	0	0	0
001	–	–	1	–
011	–	–	1	–
111	–	–	1	–
101	–	–	–	–

$y_1y_2y_3$



ab

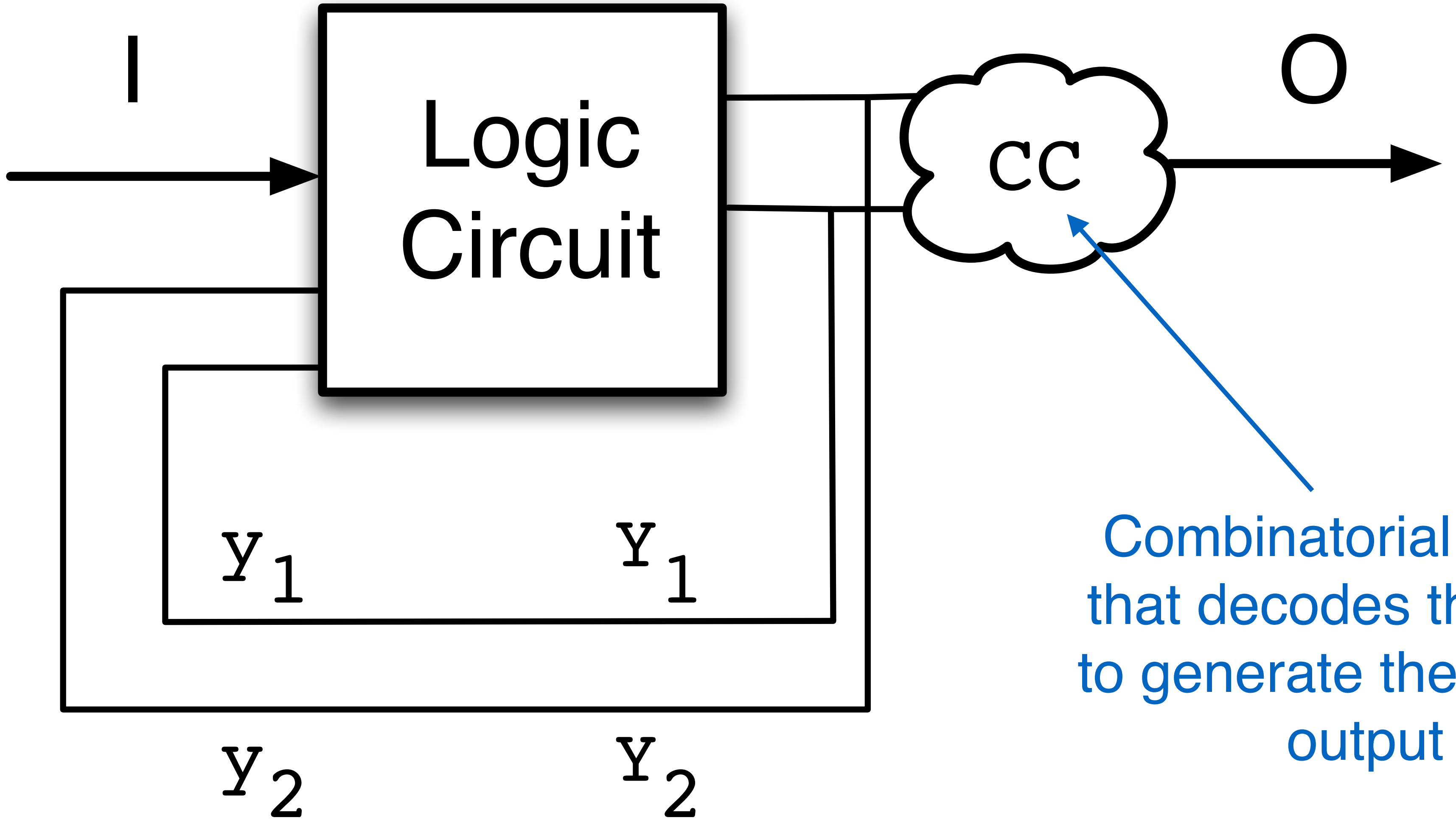
Y_3	00	01	11	10
000	0	0	1	0
010	0	0	0	1
110	0	0	0	0
100	0	0	0	0
001	–	–	1	–
011	–	–	1	–
111	–	–	0	–
101	–	–	–	0

$y_1y_2y_3$

4. Meally machines

Moore Machine

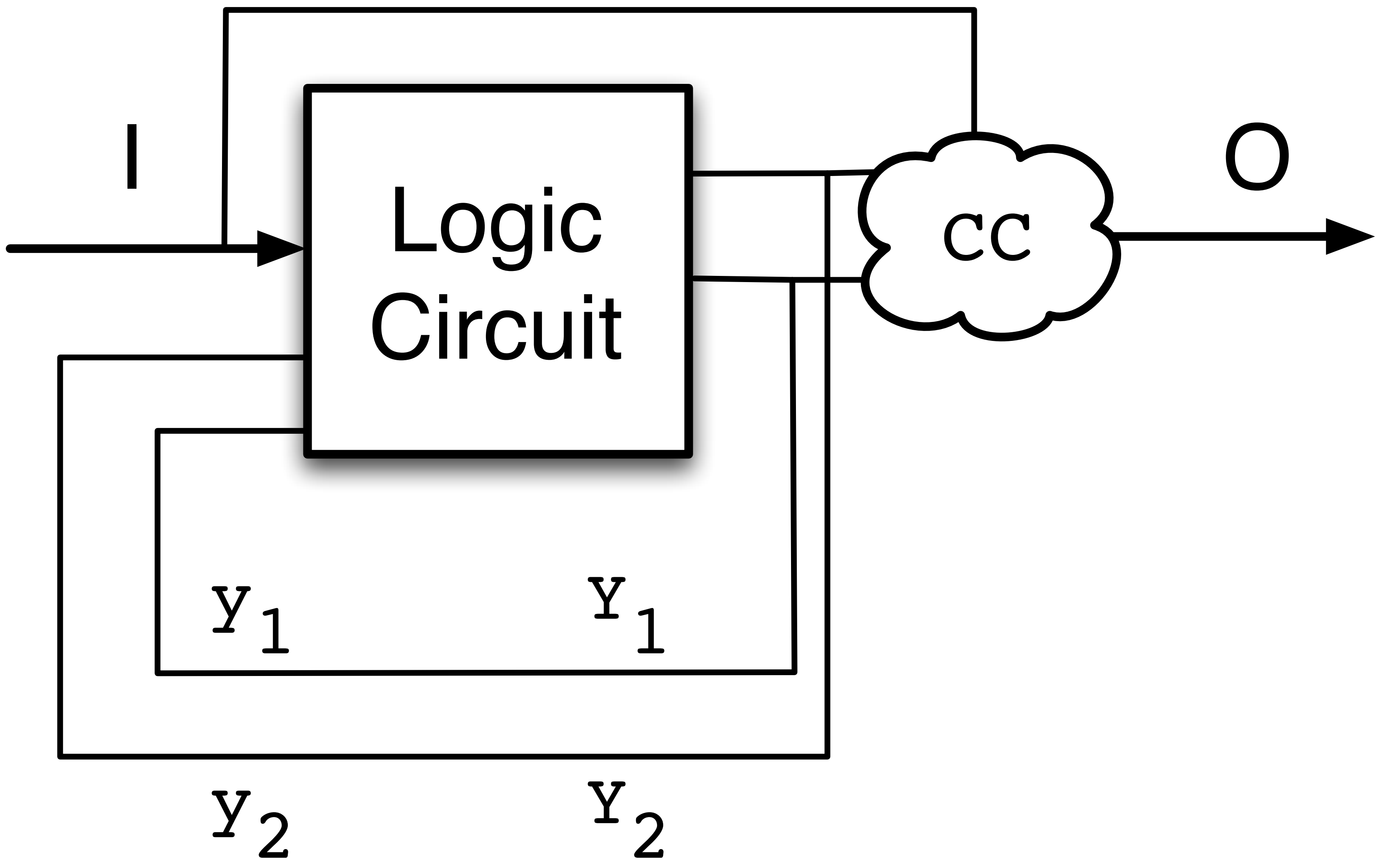
Output (combinatorial) is function of **state variables** only;
(we focus on this type of machines for a moment)



Combinatorial Circuit that decodes the state to generate the system output

Meally machine

Output (combinatorial) is function of **state variables AND the system INPUT**



Moore Machine – output synthesis

Easiest way: using a simple **state decoder**; we look into the values of the output at stable states & we write simple sum of products for outputs that are = 1

ab

	00	01	11	10	z
00	**00**	01	01	–	0
01	00	**01**	11	–	1
11	**11**	01	**11**	10	0
10	–	**10**	11	**10**	1

State 01

State 10

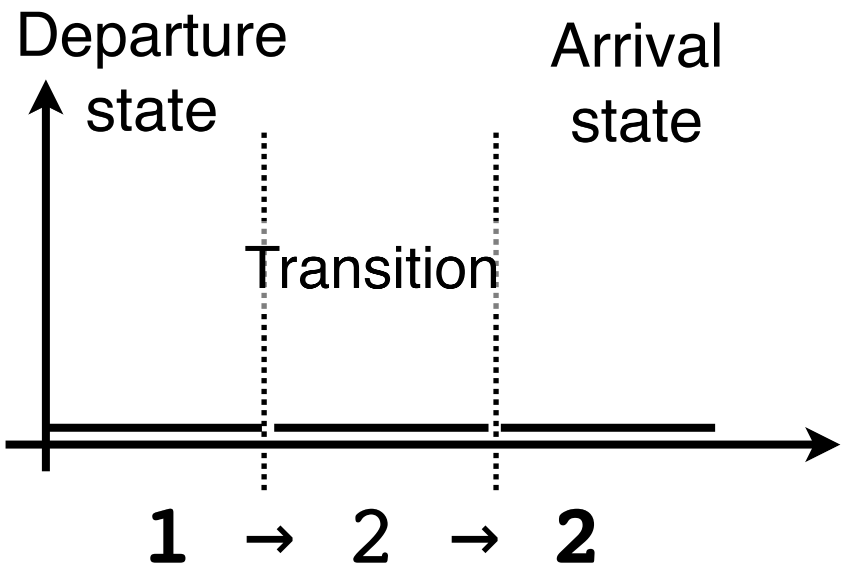
$$z(y_1,y_2) = y_1'y_2 + y_1y_2'$$

Not a good way to do things!

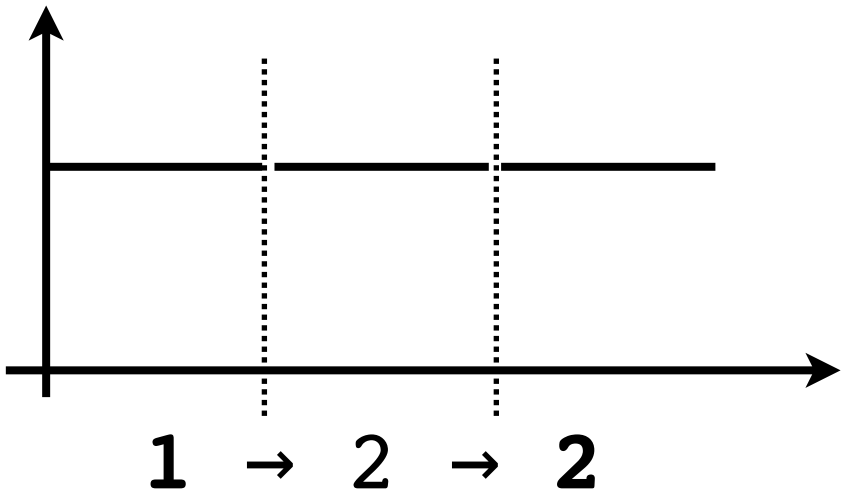
Look at transition **00**→01→11→**11** in the table above and follow the output **0**→1→0→**0** this does look like glitch that we want to avoid !

One output transition rule – **single transition**

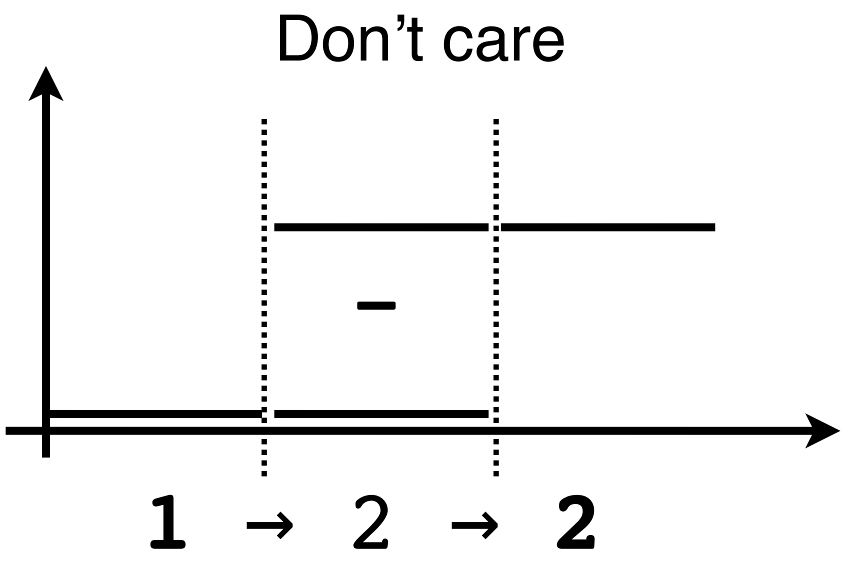
a) Departure state & destination state outputs are the same:

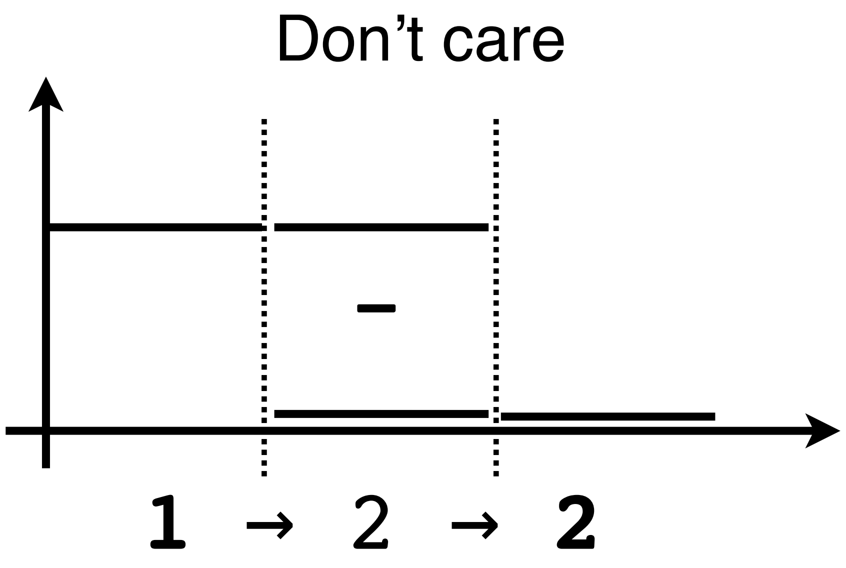


Transition keeps the same value



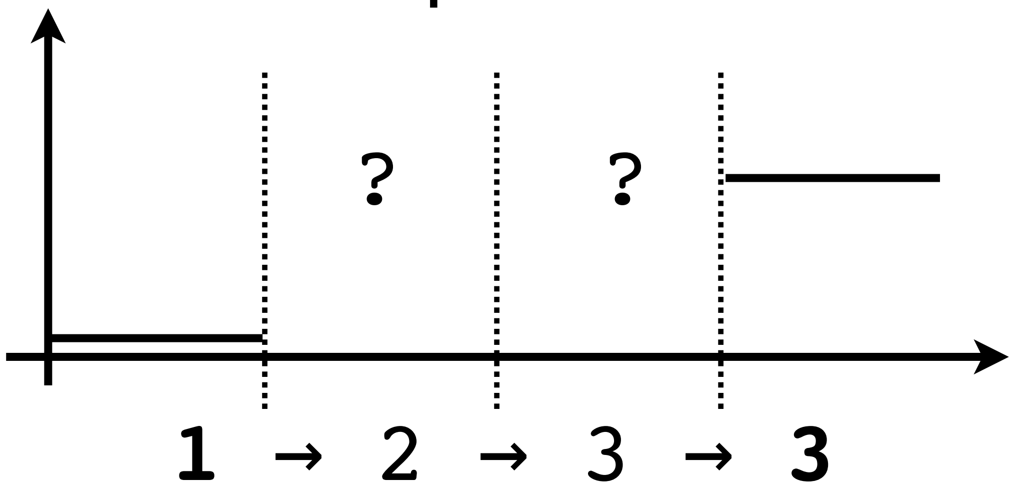
b) Departure state & destination state outputs are the different:



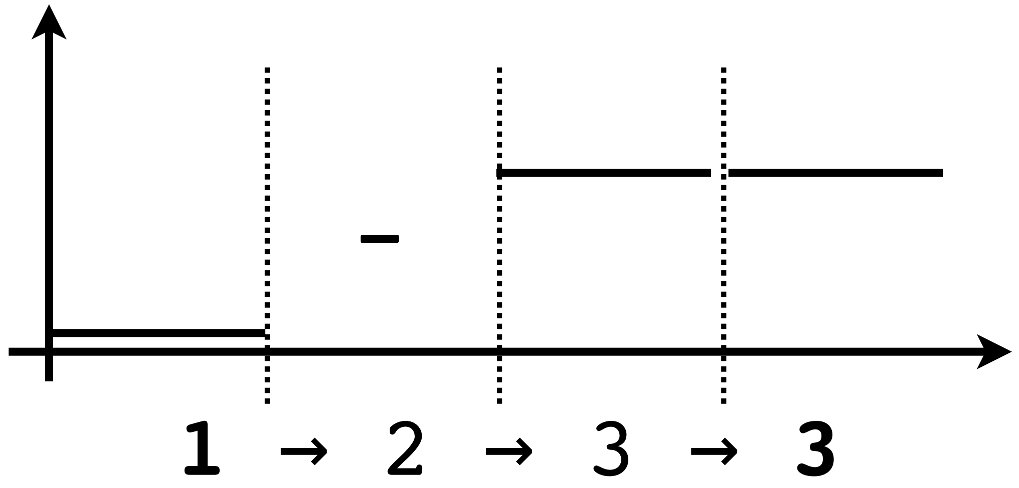


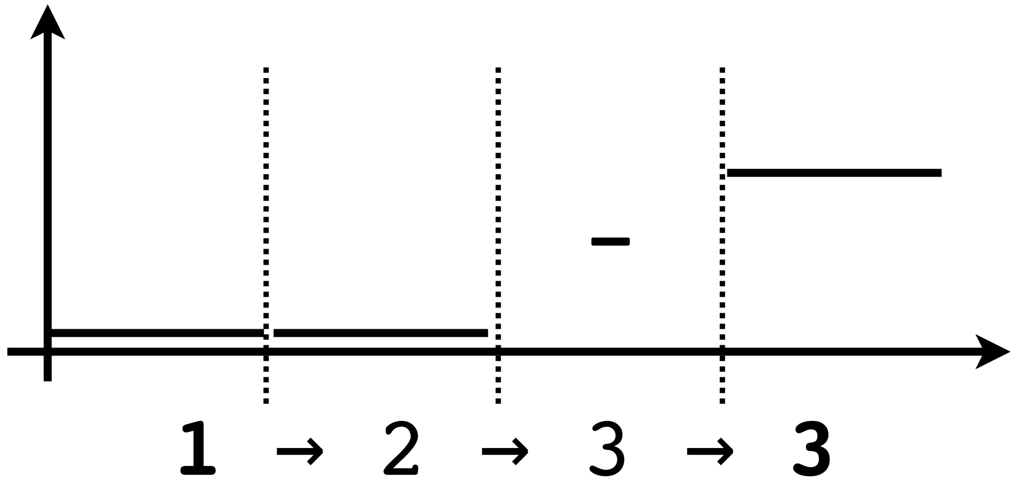
One output transition rule – **multiple transitions**

When solving race conditions we can introduce **multiple transitions**, the question is how to fix the output in such cases?



Rule is simple – **only a single transition is allowed**, so:

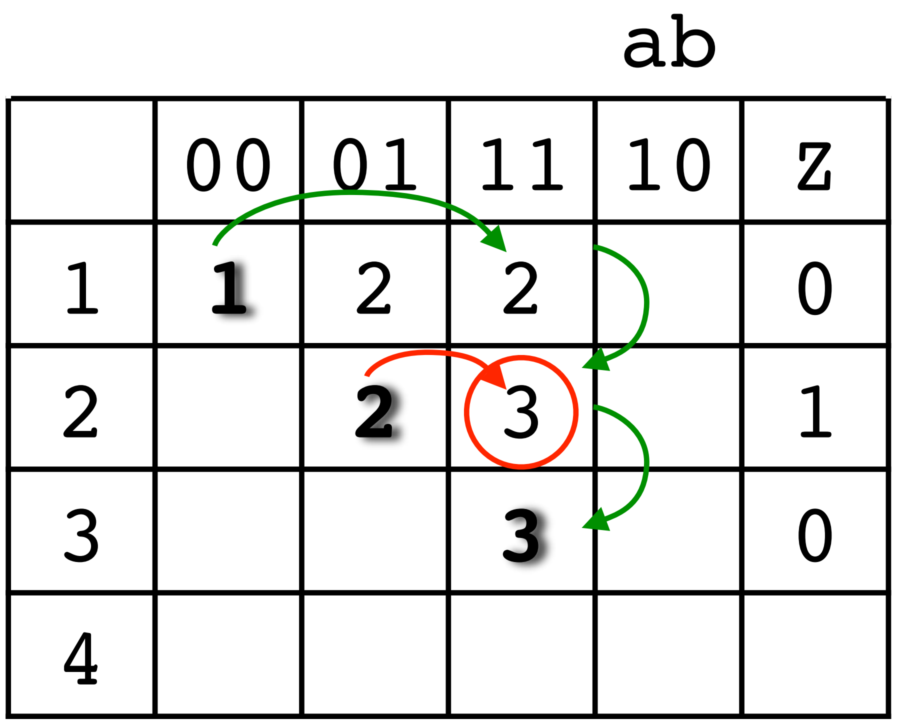




Only one don’t care is allowed; **can you explain why?**

One output transition rule – **shared transitions**

Attention! A transition between two stable states can be shared (especially after we applied **Method2** to solve race conditions)



ab

	00	01	11	10	Z
1	**1**	2	2		0
2		**2**	3		1
3			**3**		0
4					

Here we have:

2→3→**3**

but also:

1→2→3→**3**

3 is shared transition

It is the transition **1**→2→3→**3** that will fix the value for 3 and not **2**→3→**3** …

Moore Machine – output synthesis example

Output K-Map base on a single output transition rule:

ab

	00	01	11	10	z
1	**1**	2	2	–	0
2	1	**2**	3	–	1
3	**3**	2	**3**	4	0
4	–	**4**	3	**4**	1

ab

z	00	01	11	10
00	**0**	–	0	–
01	–	**1**	0	–
11	**0**	–	**0**	–
10	–	**1**	–	**1**

Two solutions since one essential and two (simple) prime implicants

$z_1 = a'b+y_1y_2'$ or

$z_2 = a'b+ab'$

What does this input does here?

Meally machine as opposed to Moore

- In Moore machine the output is a function of system states **only**
- For this reason during state optimisation we could not fuse states with different outputs
- **But to avoid glitches at system output we limit the number of transitions to only one!** (1 change from 0 to 1 and inversely)
- This means that we explicitly need to set-up output during transitions which in turn might appear inputs in the final output
- Is this then Meally machine? → **NO!**
- Difference between the two should appear early in the synthesis process
- Because we can use inputs, we can make a difference between different stable states in a single state table line, and that has a very strong consequence

Meally machine: state fusion example

For the system below: states **1** & **4** can't be fused in the case of the Moore machine (How to differentiate between the two?)

ab

	00	01	11	10	Z
1	**1**	3	2	-	1
2					0
3					0
4	1	3		**4**	0

In case of Meally we could do that (because we can use inputs to distinguish between the two 1)

ab

	00	01	11	10
1	**1/1**	3	2	**1/0**
2				
3				

Meally machine – output

- Output is function of inputs too, **same stable state can have different outputs** (impossible in Moore)
- Rather than a single output column (one per state), one output column per input combination
- To simplify things we use a simple slash symbol next to the state

ab

	00	01	11	10
1	**1/1**		2	**1/0**
2			**2/0**	
3				
4				

Here there is a difference between two stable states **1/1** & **1/0** due to input difference ab=00,ab=10

Meally machine: states that can’t fuse

Exception

We can not merge two stable states having different outputs and being stable for the same combination of inputs (when these states are in the same column)

ab

	00	01	11	10	z
1	**1**		2		1
2			**2**		0
3	**3**				0
4					

In the example on the left we can not make a difference between states 1 et 3 if they were merged! (what do we put as output value?)

Machine de Meally – output synthesis

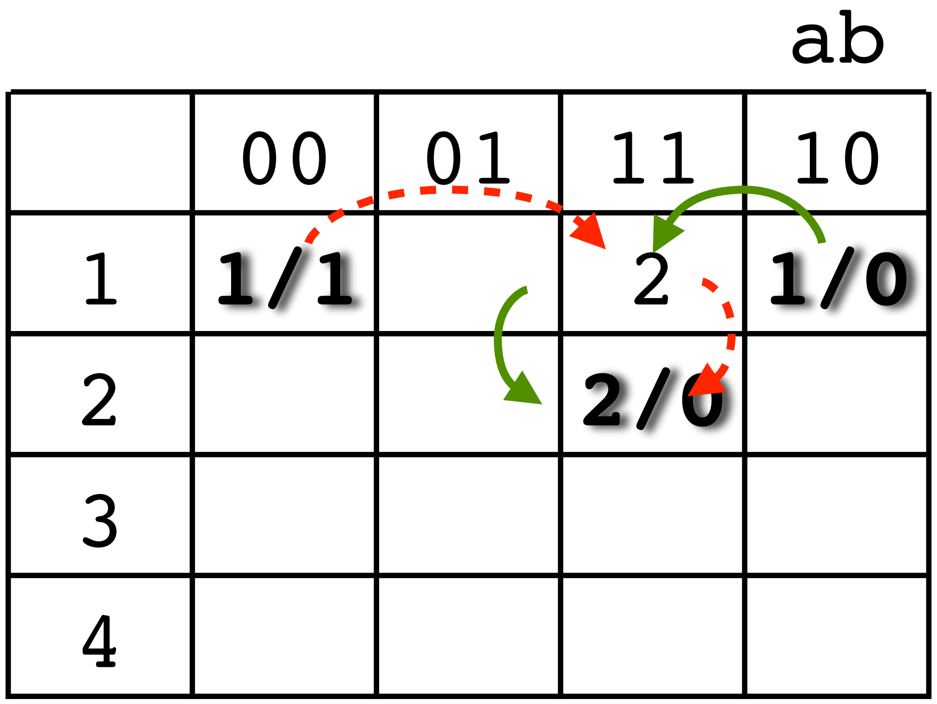
- Same idea is applied as for Moore machine:
  - A single transition (from 0 to 1 or 1 to 0) is allowed between two stable states (no matter how many transitions we have)
- **Attention!**
  Things can be a bit more complicated in the case of Meally machines since we can have two (or more) outputs (that might differ) for a same stable state
- Meally output could be therefore more constrained than for Moore machine (and thus have more complex final expressions)
- Two cases (as previously):
  - Single transitions
  - Multiple transitions that could be eventually shared

Meally machine – output synthesis (1/2)

- Transition 2 from state **1** when input ab=11 can be taken :

  a) from state **1** when ab=00, z=1 (discontinued line) or

  b) from state **1** when ab=10, z=0 (continuous line)



ab

	00	01	11	10
1	**1/1**		2	**1/0**
2			**2/0**	
3				
4				

ab

z	00	01	11	10
1	**1**		0	**0**
2			**0**	
3				
4				

It is the transition from ab=10 with z=0 that will force the output value for transition 2

Meally machine – output synthesis (2/2)

Attention! Shared transitions – In the example below transition 3 for ab=11 is used to move from **1** to **3**, but also to move from **2** to **3**

We need to fix the transition from **2** to **3** to 0 **first!** Transition **2** will have to be fixed later, accordingly to what has been set for 2

ab

	00	01	11	10
1	**1/1**		2	**1/1**
2		**2/0**	3	
3			**3/0**	
4				

ab

z	00	01	11	10
1	**1**		–	**1**
2		**0**	**0**	
3			**0**	
4				

5. Synthesis of Moore AND Meally machines (for the same problem)

System specification

- System behaviour described using primitive state table
- We need to perform **state optimisation** & system synthesis
- We consider both:
  - **Moore machine**
  - **Meally machine**
- Different solutions could be then compared for logic complexity
- A priori Meally machine could have less states due to possibly better state optimisation (because we can merge states having different outputs)

ab

	00	01	11	10	Z
1	**1**	2	-	3	0
2	1	**2**	4	-	1
3	1	-	4	**3**	0
4	-	5	**4**	-	1
5	6	**5**	4	-	0
6	**6**	5	-	3	1

A. Moore machine

We exclude all states with different outputs: all cells marked with x during the 1st pass when building **equivalences conditions table**

ab

	00	01	11	10	z
1	**1**	2	-	3	0
2	1	**2**	4	-	1
3	1	-	4	**3**	0
4	-	5	**4**	-	1
5	6	**5**	4	-	0
6	**6**	5	-	3	1

1st pass

2	x				
3	OK	x			
4	x	2-5	x		
5	1-6 2-5	x	1-6	x	
6	x	1-6 2-5	x	OK	x
	1	2	3	4	5

A. Moore machine – state fusions

2nd pass: 1-6, 2-5, NOK...

2	X				
3	OK	X			
4	X	~~2-5~~	X		
5	~~1-6~~ ~~2-5~~	X	~~1-6~~	X	
6	X	~~1-6~~ ~~2-5~~	X	OK	X
	1	2	3	4	5

Fusion graphs

1 — 3

4 — 6

2

State fusion

ab

	00	01	11	10	Z
1	**1**	2	-	3	0
2	1	**2**	4	-	1
3	1	-	4	**3**	0
4	-	5	**4**	-	1
5	6	**5**	4	-	0
6	**6**	5	-	3	1

2	→	2
1-3	→	1
4-6	→	3
5	→	4

Fused table

ab

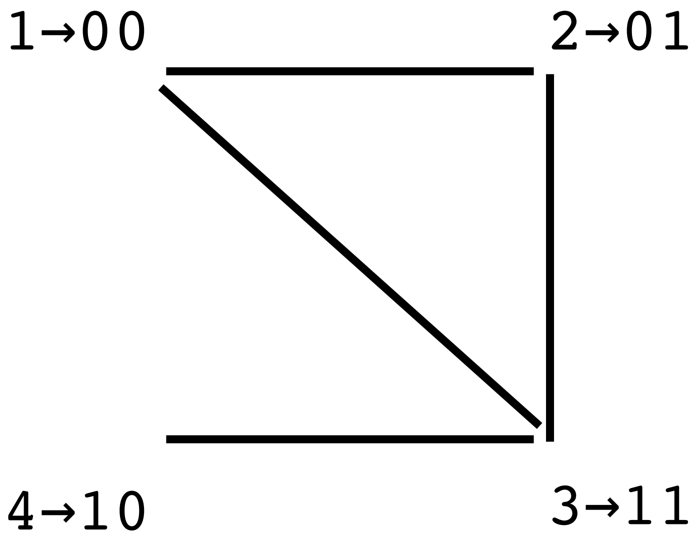
	00	01	11	10	Z
1	**1**	2	-	**1**	0
2	1	**2**	3	-	1
3	**3**	4	**3**	1	1
4	3	**4**	3	-	0

A. Moore machine – state encoding

State encoding graph: we will always have one race condition

ab

	00	01	11	10	z
1	**1**	2	–	**1**	0
2	1	**2**	3	–	1
3	**3**	4	**3**	1	1
4	3	**4**	3	–	0

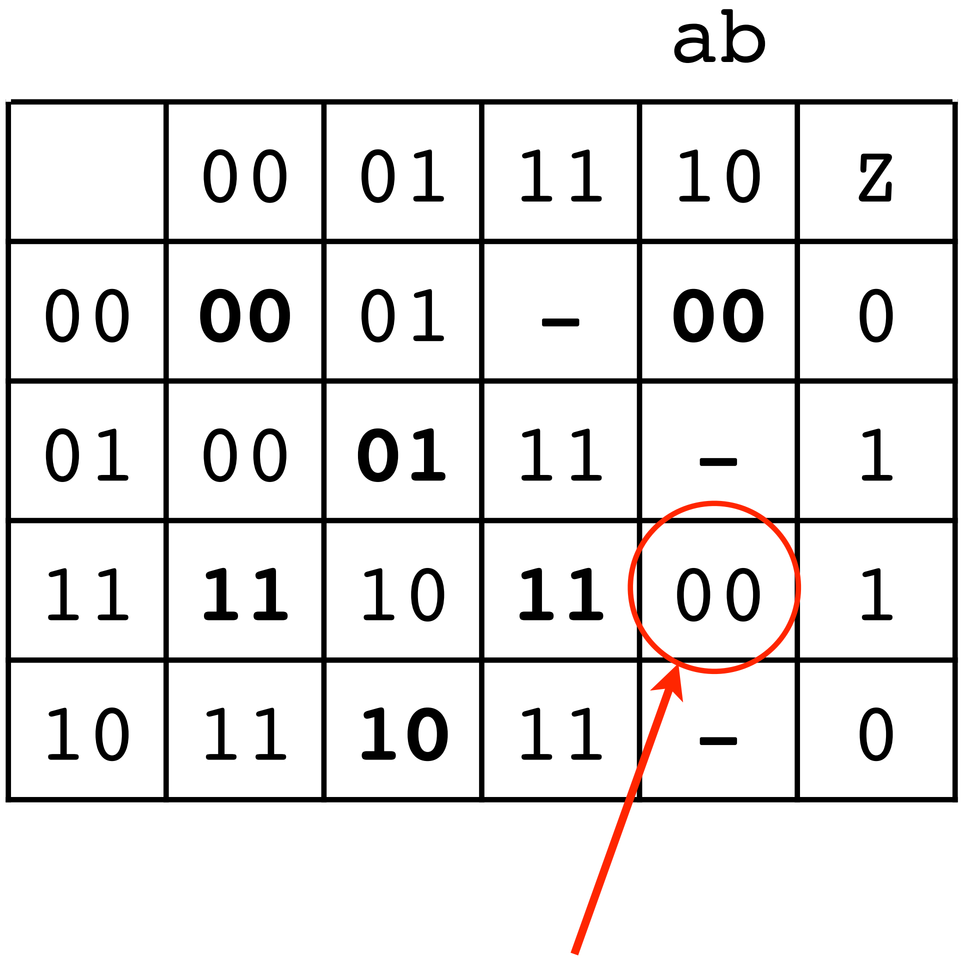


ab

	00	01	11	10
00	**00**	01	–	**00**
01	00	**01**	11	–
11	**11**	10	**11**	00
10	11	**10**	11	–

Race

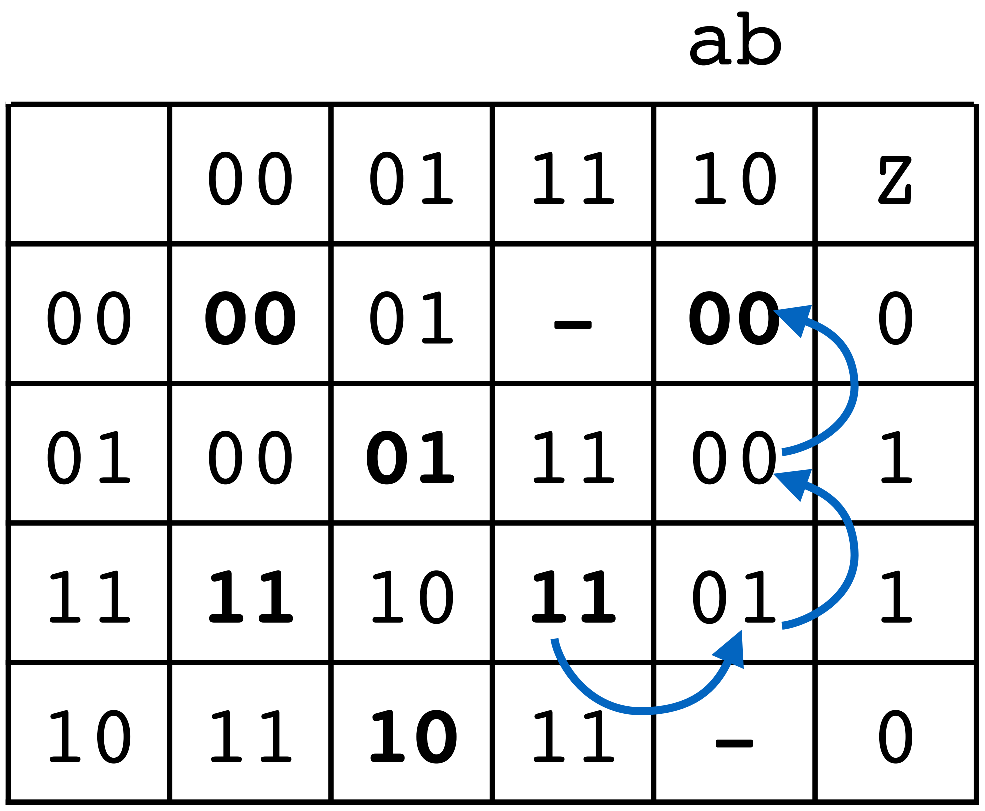
A. Moore machine – race condition



ab

	00	01	11	10	Z
00	**00**	01	–	**00**	0
01	00	**01**	11	–	1
11	**11**	10	**11**	00	1
10	11	**10**	11	–	0

Race



ab

	00	01	11	10	Z
00	**00**	01	–	**00**	0
01	00	**01**	11	00	1
11	**11**	10	**11**	01	1
10	11	**10**	11	–	0

Solution – transition

A. Moore machine – feedback loops

ab

	00	01	11	10	Z
00	**00**	01	–	**00**	0
01	00	**01**	11	00	1
11	**11**	10	**11**	01	1
10	11	**10**	11	–	0

Y_1	00	01	11	10
00	**0**	0	–	**0**
01	0	**0**	1	0
11	**1**	1	**1**	0
10	1	**1**	1	–

$Y_1=a'y_1+ab$

Y_2	00	01	11	10
00	**0**	1	–	**0**
01	0	**1**	1	0
11	**1**	0	**1**	1
10	1	**0**	1	–

$Y_2=ab+b'y_1+by_1'$

A. Moore machine – output function synthesis

State table with one output per state

ab

	00	01	11	10	z
00	**00**	01	–	**00**	0
01	00	**01**	11	00	1
11	**11**	10	**11**	01	1
10	11	**10**	11	–	0

Outputs for stable states

ab

z	00	01	11	10
00	**0**		–	**0**
01		**1**		
11	**1**		**1**	
10		**0**		–

Transitions 1 change allowed

ab

z	00	01	11	10
00	**0**	–	–	**0**
01	–	**1**	1	–
11	**1**	–	**1**	1
10	–	**0**	–	–

$z=y_2$

B. Meally machine

New state equivalence table in which we can merge **states with different outputs** (all except **1** et **6** et **2** et **5**)

	ab				
	00	01	11	10	Z
1	**1**	2	–	3	0
2	1	**2**	4	–	1
3	1	–	4	**3**	0
4	–	5	**4**	–	1
5	6	**5**	4	–	0
6	**6**	5	–	3	1

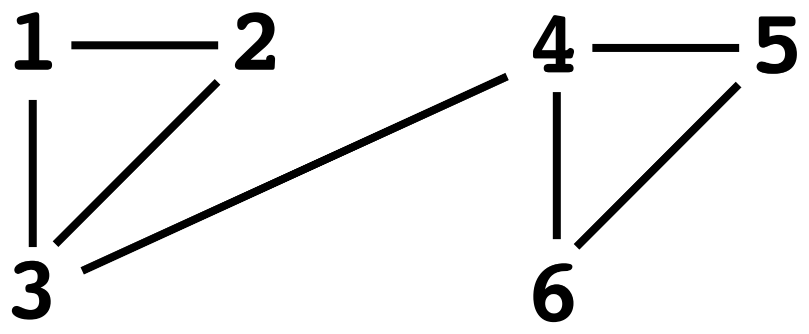
1st pass

2	OK				
3	OK	OK			
4	2–5	2–5	OK		
5	1–6	X	1–6	OK	
6	X	1–6 2–5	1–6	OK	OK
	1	2	3	4	5

B. Meally machine – state fusion

2nd pass

2	OK				
3	OK	OK			
4	~~2-5~~ X	~~2-5~~ X	OK		
5	~~1-6~~ X	X	~~1-6~~ X	OK	
6	X	~~1-6~~ ~~2-5~~ X	~~1-6~~ X	OK	OK
	1	2	3	4	5



State fusion

1,2,3 → 1
4,5,6 → 2

rather than !!!

1,2 → 1
3,4 → 2
5,6 → 3

ab

	00	01	11	10	Z
1	**1**	2	–	3	0
2	1	**2**	4	–	1
3	1	–	4	**3**	0
4	–	5	**4**	–	1
5	6	**5**	4	–	0
6	**6**	5	–	3	1

Fused state table

	00	01	11	10
1	**1/0**	**1/1**	2	**1/0**
2	**2/1**	**2/0**	**2/1**	1

B. Meally machine – state encoding and output

ab

	00	01	11	10
1	**1/0**	**1/1**	2	**1/0**
2	**2/1**	**2/0**	**2/1**	1

One state bit is enough

ab

Y_1	00	01	11	10
0	**0**	**0**	1	**0**
1	**1**	**1**	**1**	0

$Y_1=y_1a'+ab$

ab

Z	00	01	11	10
0	**0**	**1**	1	**0**
1	**1**	**0**	**1**	0

$Z=y_1a'b'+y_1'b+ab$

Result comparison

Moore

$Y_1=a'y_1+ab$

$Y_2=ab+b'y_1+a'by_1'$

$Z=y_2$

Meally

$Y_1=y_1a'+ab$

$Z=y_1a'b'+y_1'b+ab$

Less circuits for Meally (2 instead of 3) but with a bit more complicated output function…

We can not say in advance which one is better!