ELEC-H310
Digital Electronics

# Lecture 06
# Sequential logic circuits

Dragomir MILOJEVIC
2021

## Today

1. Asynchronous logic circuits (reminder) & their performance

2. Synchronous circuits (concept)

3. Flip-Flops (reminder) & their usage in sequential circuits synthesis

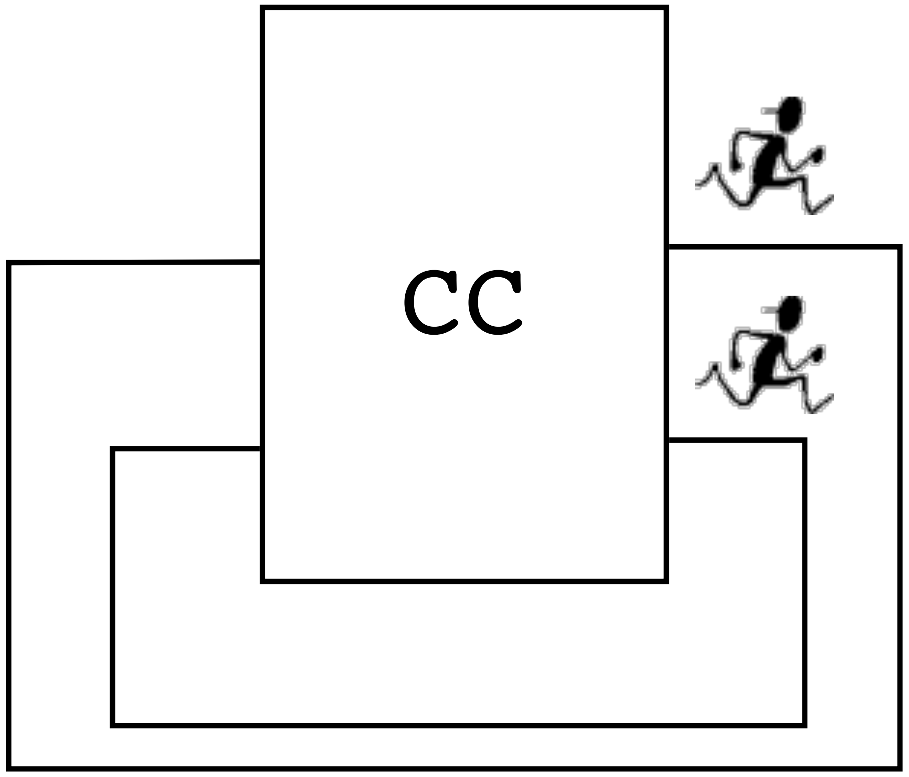
4. Algorithm for synthesis of synchronous logic circuits

5. Example of synchronous sequential system synthesis

## 1. Asynchronous logic circuits (reminder) & their performance

# Race conditions & asynchronous logic

- Problem of **race conditions** will appear when a state machine changes **simultaneously** more than two state variables at a time (i.e. Hamming distance between $y_i$ and $Y_i$ is $> 1$)
- **Race conditions** are the consequence of physical implementation of a circuit: gates & wire have delays and these delays in **concurrent circuits** will be different (one circuit per state variable) → some circuits will be faster, the others will be slower
- In an **asynchronous circuit** race conditions **that are not solved** will cause a non-desired behaviour: system will end up in another state than the one specified by a state table

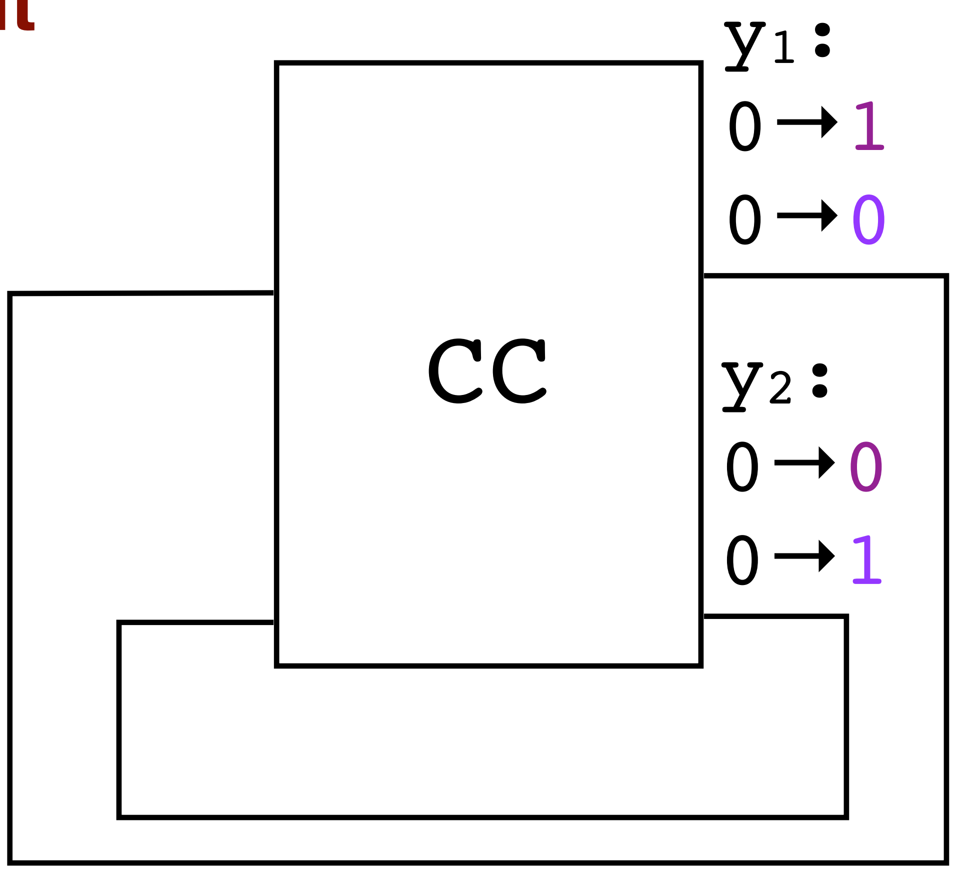


# Asynchronous circuit implementation (synthesis)

- Final encoded state table **can't contain race conditions**
- Solving races using **three different methods** applied in order to allow optimal system:
  - **Method1** – State encoding
  - **Method2** – Transitions modifications
  - **Method3** – Adding an extra state variable (to enable **Method2**)
- Final step of the system synthesis is the output function – to avoid glitches at the output we need to fix the values of the transitions so that only one change appears on the output
- Exact moment of the output toggle is not important; what is important is the change of the value
- We can chose from More and Meally machines, difference is in the state fusion: Meally machines allow fusion of states with different outputs (possibly better, since fewer states in the end)

# Asynchronous circuit performance

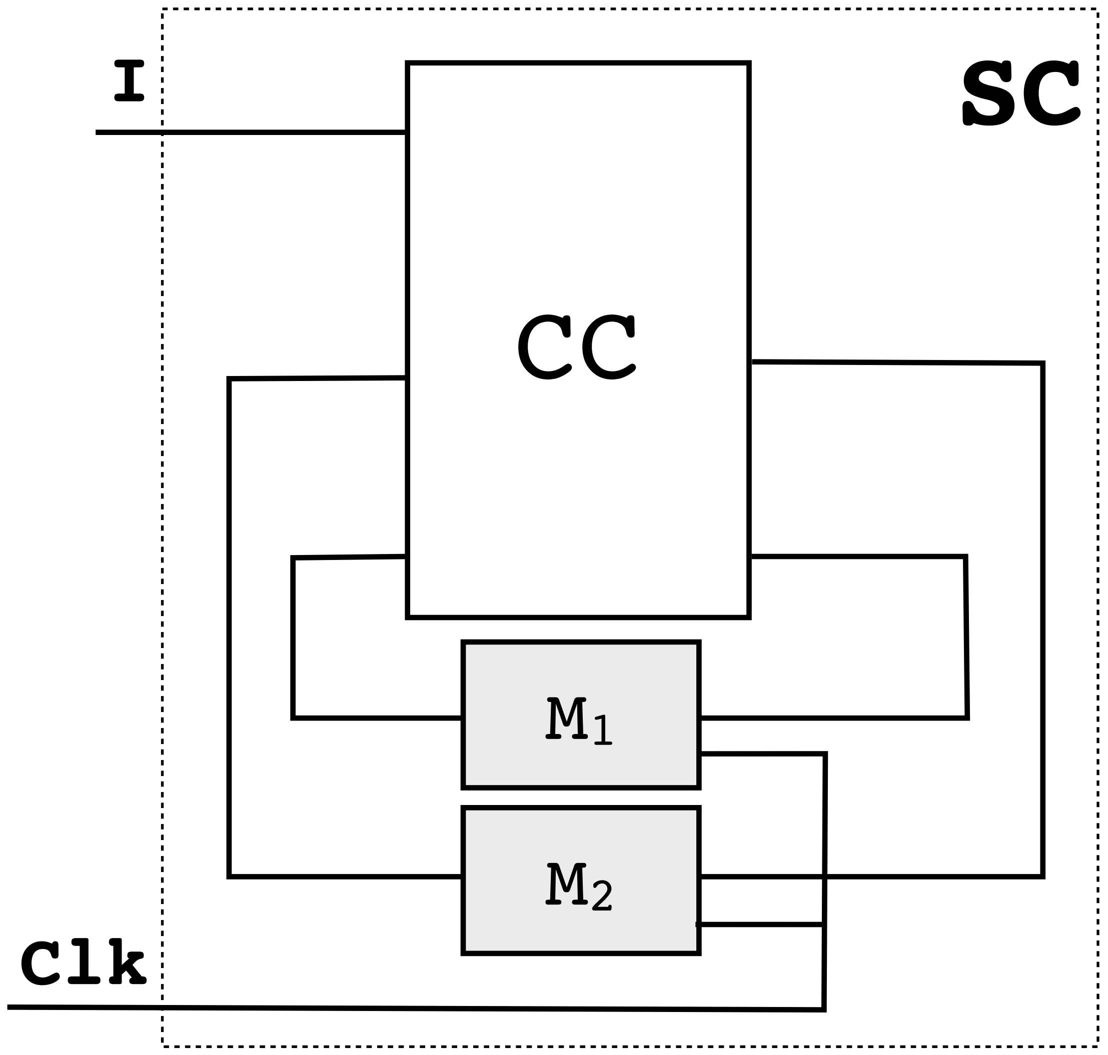
- In an asynchronous machine the state will change whenever **the state variable changes**
- This is because only one state variable changes at time (to allow the physical implementation with no race conditions)
- Consequence: the system will toggle to the future state **whenever the value of the future state arrives at input**
- The **speed** of the system (i.e. transition time between two stable states) will depend on source/destination states (i.e. these times could be different)
- In the example circuit, the system will “switch faster” from the initial state 00 to state 01, than the state 10 ! **Why do I say so?**



## 2. Synchronous circuits (concept)

# Feedback delay = memory

- Another way of solving race conditions: use **memories** to store the values of **state variables**
- **State variable update**, or the moment when the predicted future should becomes present, takes place in **regular time intervals**
- These time intervals are driven by an **external control signal, common** to all memories used to store state variables



- This signal is called **Clock (Clk)**
- **Period** of this control signal is sufficiently long to allow all system transients to happen; state values stored in the memory remain stable thought the period of a clock

## Synchronous circuits

- **Synchronous operation**: all state variables do change at the same time since they share same common control signal
- Update process of the memories implement the process of the state variable update
- We note this as follows using Boolean logic equation notation:

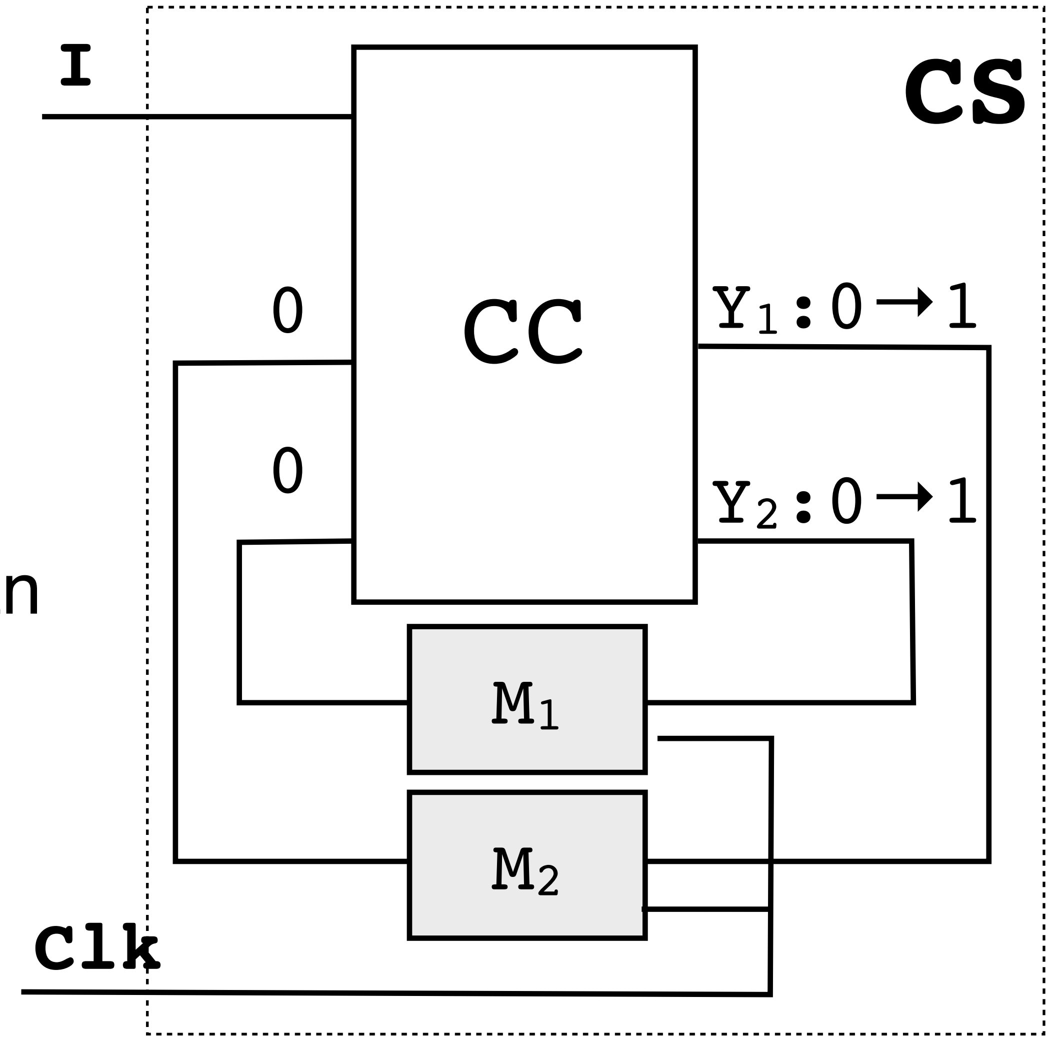
$$Y_i = f(I, y_i)$$

- In reality this is not equality, rather an **assignment**: we assign new state value to the current state
- This is why sometimes we use the following notation (e.g. Hardware Description Languages):

```
CurrentState ← NextState
```

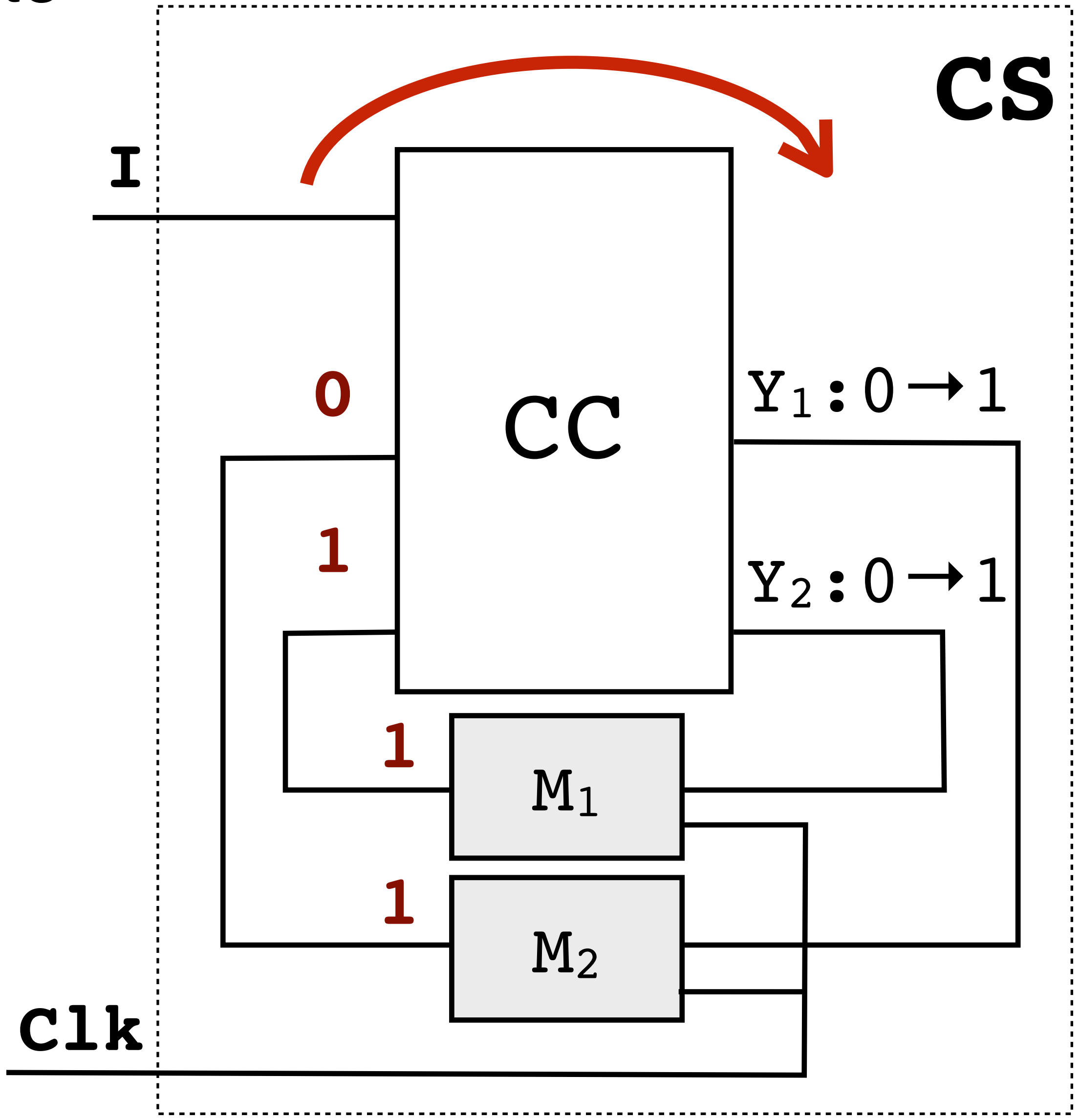
# Synchronisation: how does it work? (1/2)

- **Let's take the following example:** we are in state `00` and we need to move into the state `11` (this is normally a race condition)
- Input changes and will provoke computation process within the CC (state variable logic circuit) that will produce `11`
- However the memory elements will keep their values until the next rising edge of the control signal
- Any variation at input of the M is ignored as long as the synchronisation signal is not active (we can have transients on the input of the memory)
- If there is enough time, we will not latch to an erroneous state
- When rising edge comes, the state updates
- Future predicted by CC becomes present



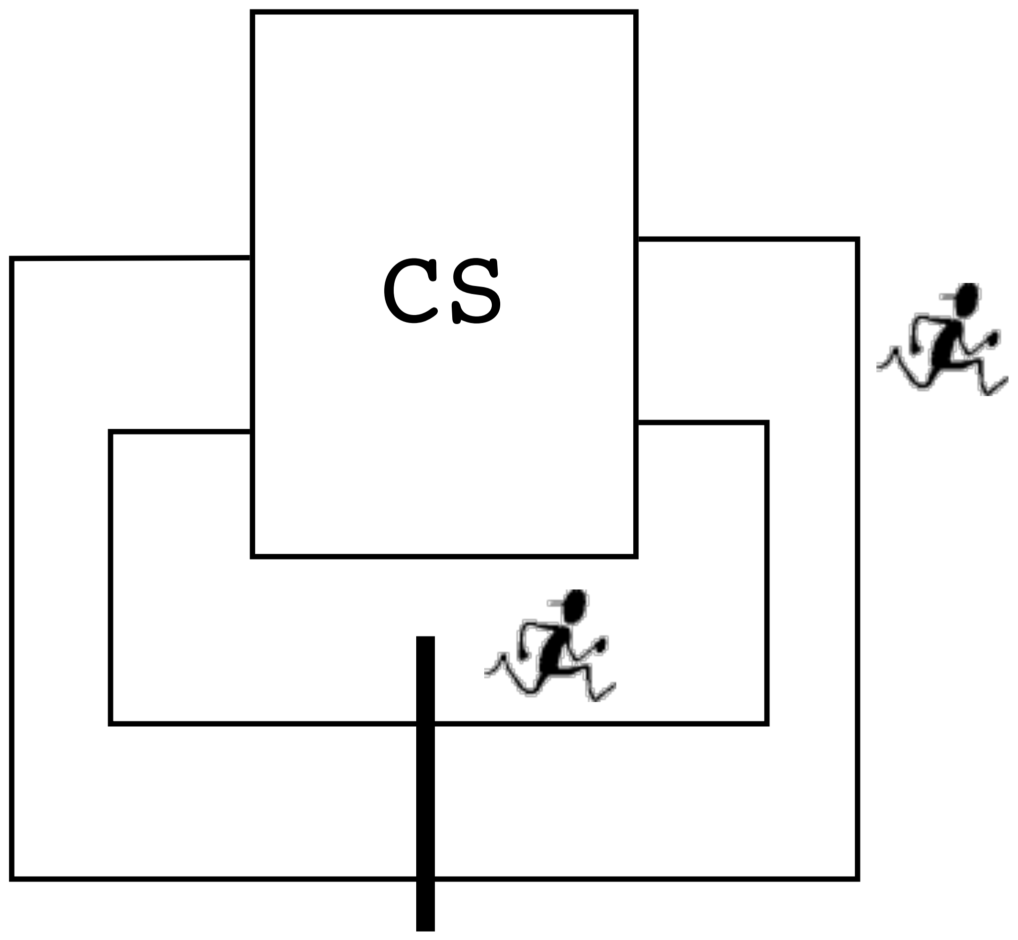
# Synchronisation: how does it work? (2/2)

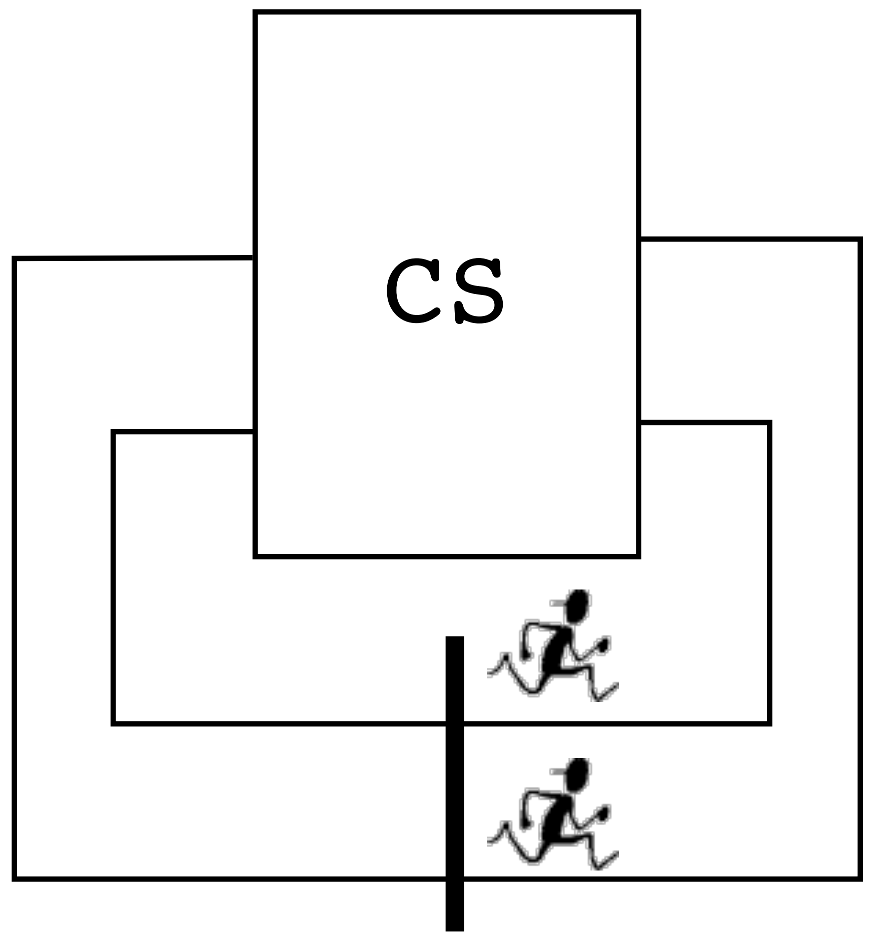
- Just right after the rising edge of `Clk` memories have as content `11`
- After this rising edge the content of the memories is locked and any variation on the input will be ignored until the next rising edge
- This new value of the state will still have to travel until the CC input and this will take some time depending on wire length
- But this is not a problem, since all these transients will be ignored at input of the memory
- Input can change at any time, and circuit can take the time it needs to settle down and compute next state, as long as it is less then `Clk` period (this time will be computed so that the worse path have enough time to travel)



# Synchronous circuits: race analogy

Period of the common control signal (Clk) is computed so that the **slowest** signal in the circuit, **has enough time to travel across**; memory element acts like a synchronisation barrier





## Performance of sequential synchronous system

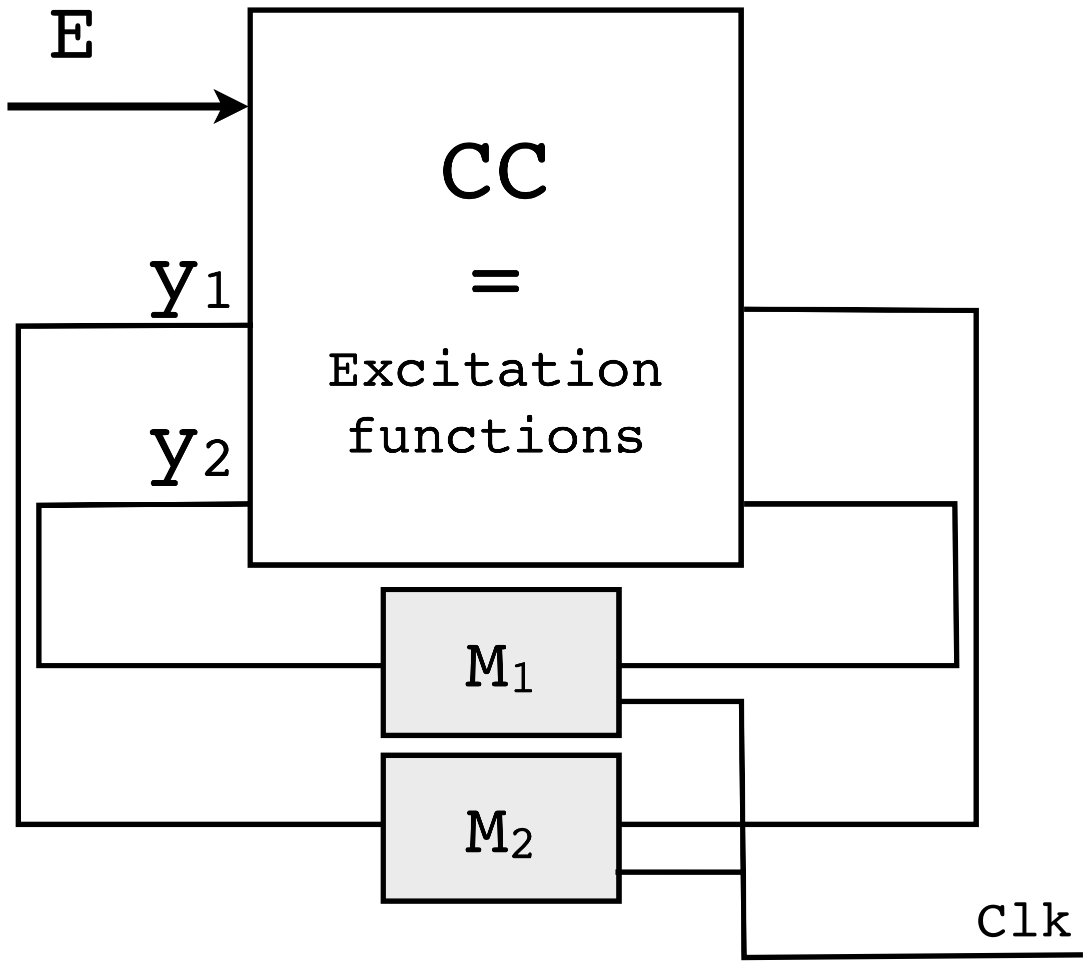
- We need to wait for the **slowest signal** in the circuit (in fact not only wires but logic gates too; this is called a **logic path**)
- Slowest signal of all for a given circuit is called **critical path**
- The speed with which the system will switch from one stable state to another will be determined by this critical path (the slowest racer)
- Because of the synchronisation, all other, even faster logic paths, will need to wait for the slowest one (this does not sound optimal)
- Hence the synchronous systems are going to be less performant than asynchronous systems (this is a necessary evil)
- Asynchronous systems will change state as quick as the transition takes place (absolute switching time from one state to another could be different)
- In sequential systems all states switch in the same amount of time (slowest one)

## Memory elements are Flip-Flop (FF)

- In previous slides we spoke about memory elements as a very general devices (we haven't said anything about how do they work)
- For the real circuit, the memory used will be one of the FFs we have seen
- These FFs will be driven by the control signals that should mimic the behaviour defined by the state table
- So instead of storing the actual state value, we will be driving the content of the FFs by providing the appropriate control of these devices
- Logic functions that will control the FF content are called **excitation functions**
- And the circuit synthesis process becomes then the process of the **excitation functions synthesis**
- Because FFs are different, these functions will depend on the type of the FF used (this is going to be the input of the problem)

# Synchronous circuits

- Memory elements $M_i$ are **standard memory elements** (so one of the flip-flops SR, JK, D, T)
- **Excitation functions** are used to steer the content of these memories
- Different state bits concatenated together will indicate the binary code of the current state
- So if $M_i$ is a flip-flop JK, we are going to derive logic functions that will **drive** J & K of each memory element

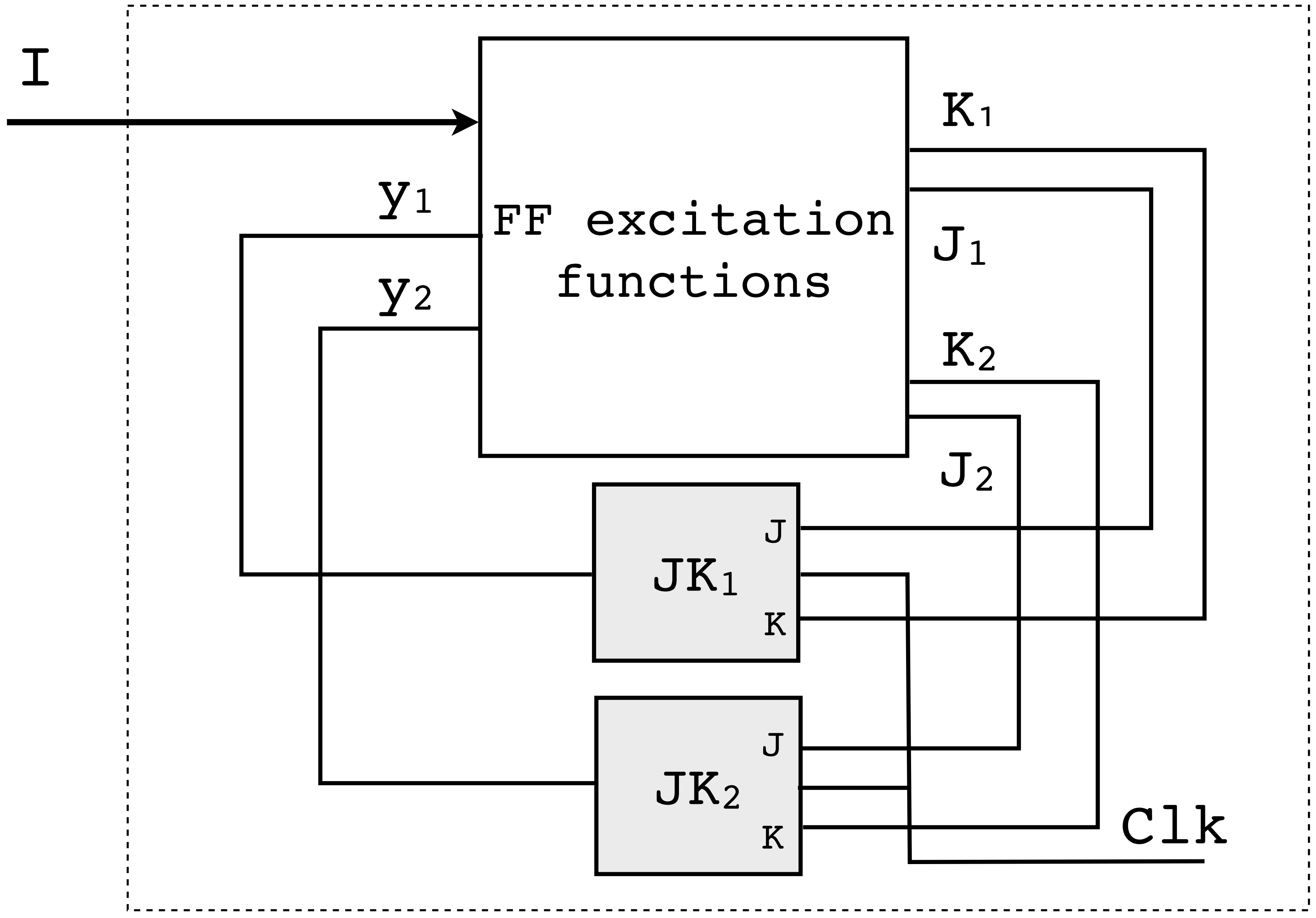


- This means 4 different logic functions, 4 independents logic circuits that are connected to the same inputs and control memory content (in case of D or T FFs this means only one logic function per memory element)

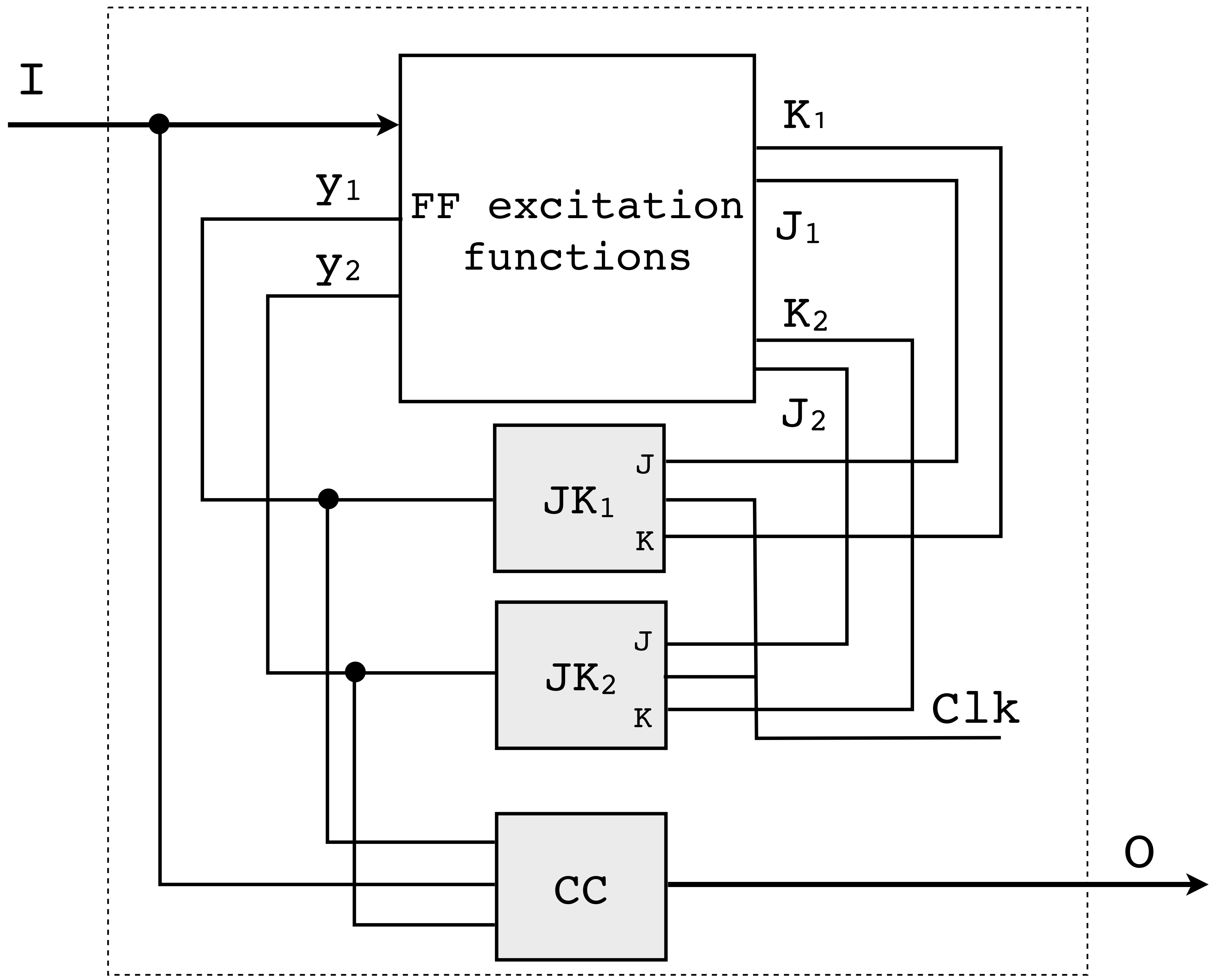
## Synchronous circuit with JK-FF

For a system with two state variables memorised using two JK flip-flops:

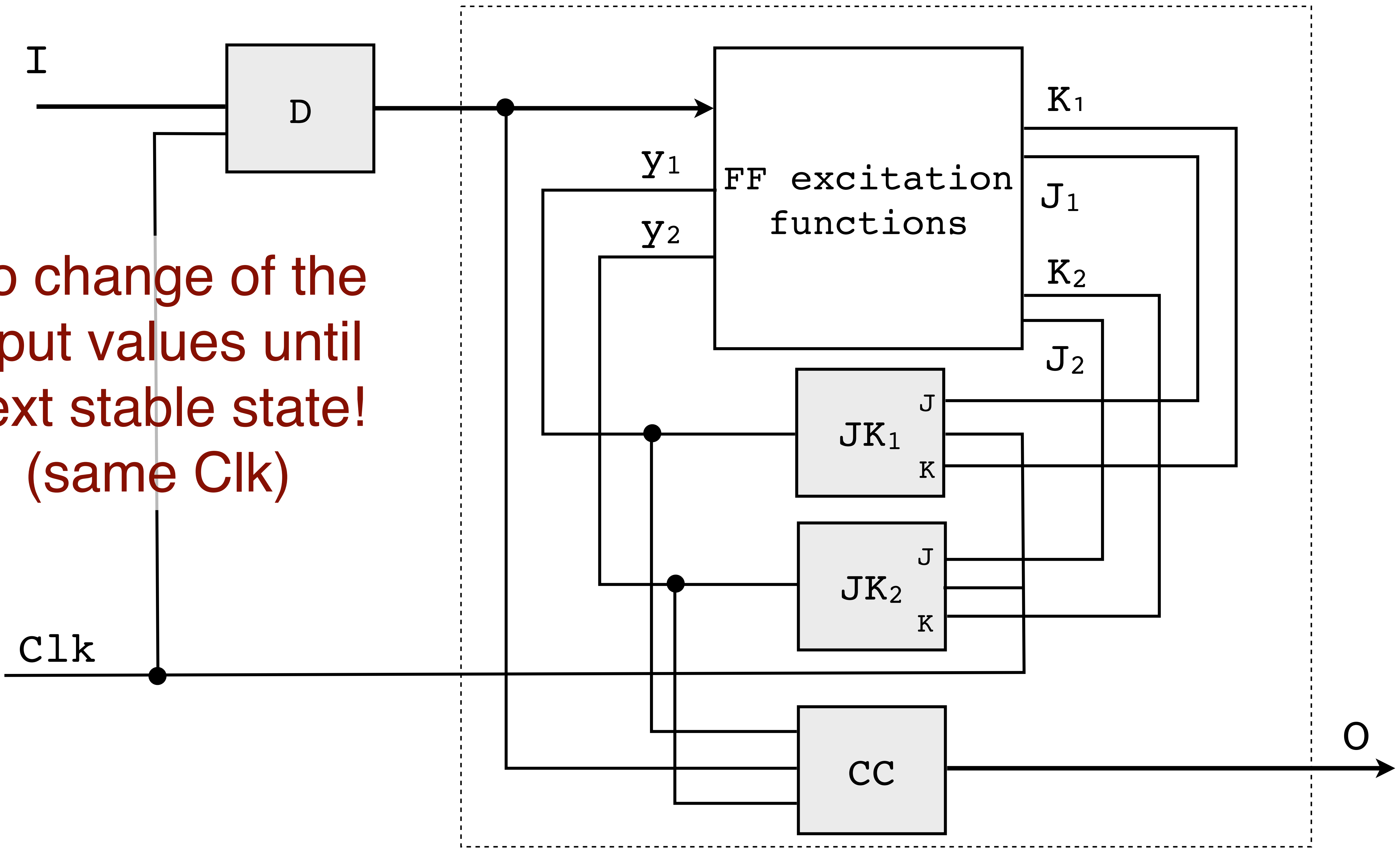
$$M_i = f(I, y_i)$$



# Complete system with output circuitry



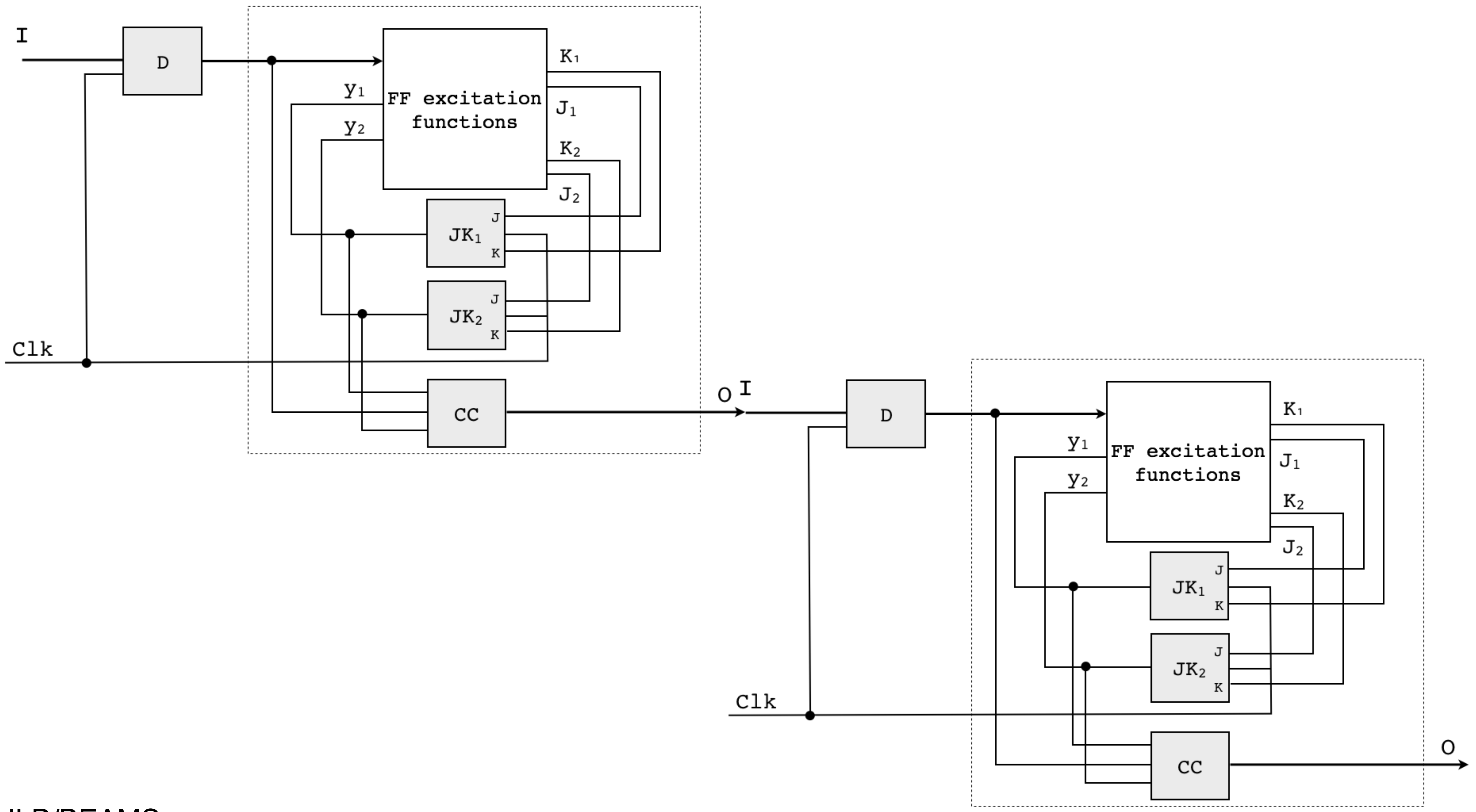
## Complete system with input synchronisation



No change of the input values until next stable state! (same Clk)

## Register Transfer Logic (RTL)

- Multiple systems connected in series with output that becomes the input of the next stage



## 3. Flip-Flops (reminder) & their usage in sequential circuits synthesis

## Flip-Flops (FFs) SR, JK, D & T: state tables

Any of the FFs: present state Q & future state $Q^+$

Different state tables:

SR

| $Q^+$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | **0** | **0** | – | 1 |
| 1 | **1** | 0 | – | **1** |

D

| $Q^+$ | 0 | 1 |
|---|---|---|
| 0 | **0** | 1 |
| 1 | 0 | **1** |

JK

| $Q^+$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | **0** | **0** | 1 | 1 |
| 1 | **1** | 0 | 0 | **1** |

T

| $Q^+$ | 0 | 1 |
|---|---|---|
| 0 | **0** | 1 |
| 1 | **1** | 0 |

**You don't need to know this by heart**

## Excitation tables of FFs

- During the synthesis of a sequential circuit, instead of state encoded tables we will use **excitation tables**
- They reflect the way how the state variables should change
- Four possibilities since we have present & future for a state, and the value could be 0 or 1:

| Q | $Q^+$ | Operation | Code |
|---|---|---|---|
| 0 | 0 | Maintain 0 | $\mu_0$ |
| 0 | 1 | Enable | $\varepsilon$ |
| 1 | 0 | Disable | $\delta$ |
| 1 | 1 | Maintain 1 | $\mu_1$ |

## State table → excitation table: SR & JK

**SR**

| | | | | SR |
|---|---|---|---|---|
| $Q^+$ | 00 | 01 | 11 | 10 |
| 0 | **0** | **0** | – | 1 |
| 1 | **1** | 0 | – | **1** |

| | Q | $Q^+$ | S | R |
|---|---|---|---|---|
| $\mu_0$ | 0 | 0 | 0 | – |
| ε | 0 | 1 | 1 | 0 |
| δ | 1 | 0 | 0 | 1 |
| $\mu_1$ | 1 | 1 | – | 0 |

**JK**

| | | | | JK |
|---|---|---|---|---|
| $Q^+$ | 00 | 01 | 11 | 10 |
| 0 | **0** | **0** | 1 | 1 |
| 1 | **1** | 0 | 0 | **1** |

| | Q | $Q^+$ | J | K |
|---|---|---|---|---|
| $\mu_0$ | 0 | 0 | 0 | – |
| ε | 0 | 1 | 1 | – |
| δ | 1 | 0 | – | 1 |
| $\mu_1$ | 1 | 1 | – | 0 |

# State table → excitation table: SR & JK

**T**

|  |  | T |
|---|---|---|
| $Q^+$ | 0 | 1 |
| 0 | **0** | 1 |
| 1 | **1** | 0 |

|  | Q | $Q^+$ | T |
|---|---|---|---|
| $\mu_0$ | 0 | 0 | 0 |
| $\varepsilon$ | 0 | 1 | 1 |
| $\delta$ | 1 | 0 | 1 |
| $\mu_1$ | 1 | 1 | 0 |

**D**

|  |  | D |
|---|---|---|
| $Q^+$ | 0 | 1 |
| 0 | **0** | 1 |
| 1 | 0 | **1** |

|  | Q | $Q^+$ | D |
|---|---|---|---|
| $\mu_0$ | 0 | 0 | 0 |
| $\varepsilon$ | 0 | 1 | 1 |
| $\delta$ | 1 | 0 | 0 |
| $\mu_1$ | 1 | 1 | 1 |

## 4. Algorithm for synthesis of synchronous logic circuits

## Steps

1. **Primitive state table** (1 stable state per line of the state table) from verbal specs (key for the correctness of this first formal spec)
2. Depending on the machine model adopted (Moore or Meally) **state table optimisation – optimised state table**
3. State encoding (**any encoding!**) will result in **encoded stat table**
4. **Excitation table** of the system based on the encoded state table
5. Depending on the memory element choice (SR, JK, D or T) memory excitation tables & finally K-Maps of the **memory excitation functions**
6. Logic optimisation and final Boolean expression for the above
7. Synthesis of the system output (solve the system transitions)

**Because of the strong dependency, we will not ask you to do all this in a single exercise for the exam**

## Synthesis asynchronous vs. synchronous

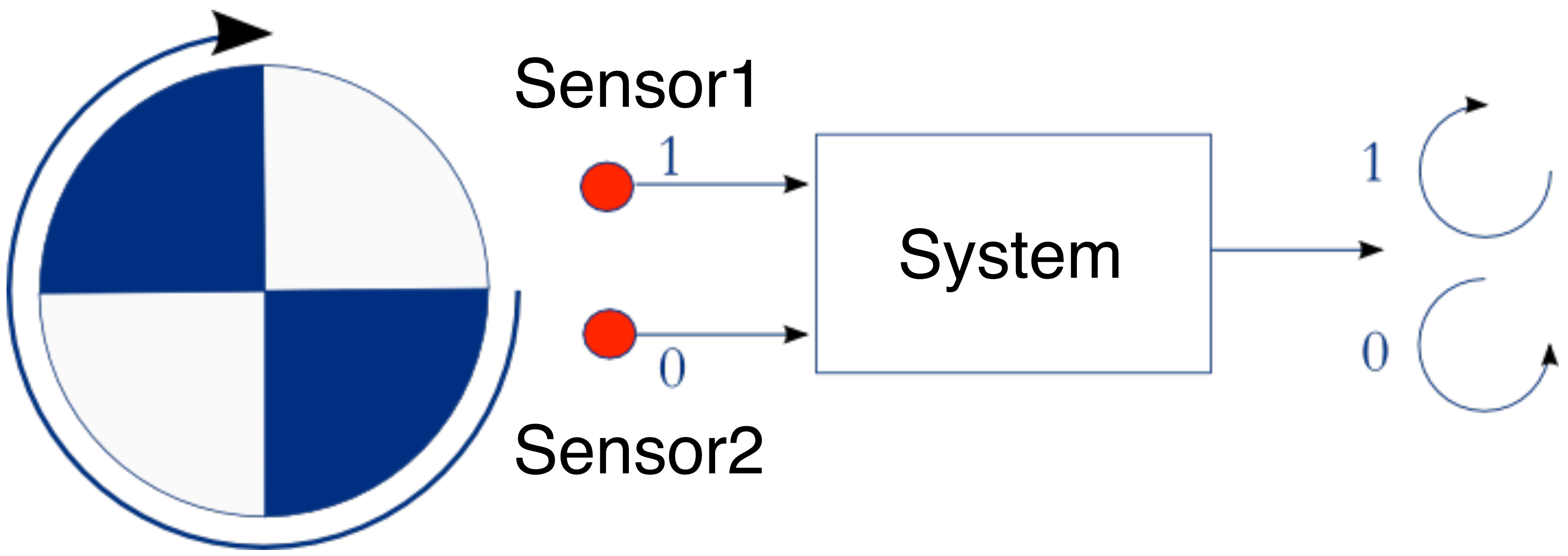
**Same algorithm, but we skip race conditions problem solving and we add synthesis of memory excitations functions!**

- State table can have race conditions!!!
- Choice of a FF could influence the complexity of the circuit in the end; we will not consider this (as said FF choice will be the input to the problem)
- You have all the elements required to do a complete system synthesis … so let's do that!

## 5. Example of synchronous sequential system synthesis

## System verbal specification

A disc, attached to a rotating axe, is divided into four equal segments alternatively painted in white and black. In front of the disc we have two sensors being able to detect the colour of the disc: 1 for white, 0 for black. We want to design a system that will tell in which direction the axe rotates: if clock wise the output Z=1, and if counter clock wise Z=0.



**Is this combinatorial or sequential circuit? Explain**

## What are we going to do?

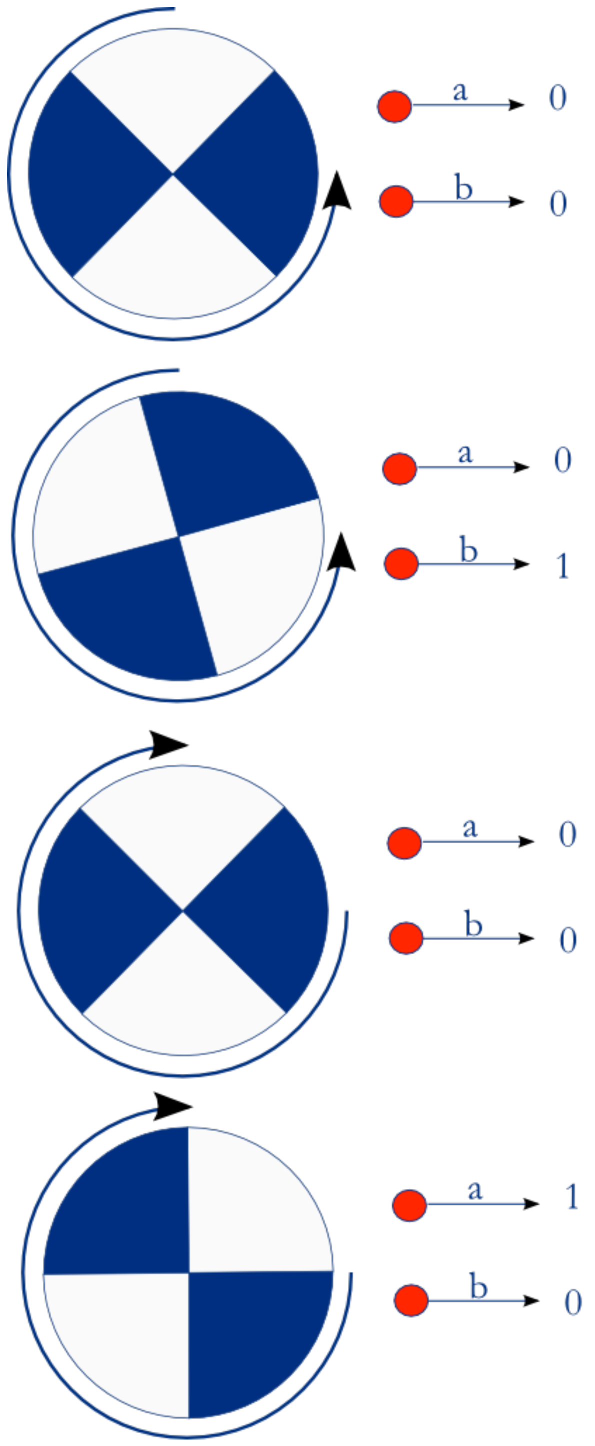
- We can choose between:
  - **Moore machine** (we **need** to respect the output)
  - **Meally machine** (we **don't need** to respect the output)
- A priori Meally machine could be more optimal because more states could be merged, but this is not guaranteed, so let's check both
- And solve race conditions by building either:
  - **Asynchronous logic circuit –** no race conditions in the state table
  - **Synchronous logic circuit – we use memories to store state values, that are updated only using a rising edge of a common control signal (Clk)**
- We can eventually compare these two solutions for logic complexity

# Problem analysis for primitive state table

- Simultaneous variations are not allowed (sensor sampling is much faster then the axe rotation speed)
- This means that in primitive state table any transition from `00` to `11`, `01` to `10`, `10` to `01` and `11` to `00` will be marked with a **don't care** (this in reality is **don't happen** because of the above)
- We make a hypothesis: initial state input is `00` and the axe rotates clock wise
- Taking anything else as hypothesis for initial state won't change anything
- As usual we analyse the possible outcomes from this state:
  - Either the axe changes the rotation direction
  - Or it doesn't (it keeps rotating as assumed initially)

# Primitive state table construction 1/2

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



**State1** is the initial state and the inputs are 00; we assume that the axe rotates counter clock wise, **rotation changes**, we end in **State2**, and then **State4** (input sequence is 00, 01, 11)

From **State1** if the rotation does not changes (the axe keeps **rotating clockwise** rotation) input sequence is 00, 10 we end in **State3**

From **State1** 11 input is not possible, so don’t care

# Primitive state table construction 2/2

|  | ab 00 | 01 | 11 | 10 | z |
|---|---|---|---|---|---|
| 1 | **1** | 2 | - | 3 | 1 |
| 2 | 1 | **2** | 4 | - | 0 |
| 3 | 5 | - | 6 | **3** | 1 |
| 4 | - | 7 | **4** | 8 | 0 |
|  |  |  |  |  |  |
| 5 | **5** | 2 | - | 3 | 0 |
| 6 | - | 7 | **6** | 8 | 1 |
| 7 | 1 | **7** | 4 | - | 1 |
| 8 | 5 | - | 6 | **8** | 0 |

- From **State3**:
  - **State5**: change of rotation (input: 00,10,00)
  - **State6**: same rotation direction (input is now: 00,10,11)
- From **State4**:
  - **State7**: rotation direction change
  - **State8**: same rotation direction
- There are two parts in the table …

## A. Moore machine – state table optimisation

Construction of the state equivalence table and **1st pass**

|  | ab 00 | 01 | 11 | 10 | z |
|---|---|---|---|---|---|
| 1 | **1** | 2 | – | 3 | 1 |
| 2 | 1 | **2** | 4 | – | 0 |
| 3 | 5 | – | 6 | **3** | 1 |
| 4 | – | 7 | **4** | 8 | 0 |
| 5 | **5** | 2 | – | 3 | 0 |
| 6 | – | 7 | **6** | 8 | 1 |
| 7 | 1 | **7** | 4 | – | 1 |
| 8 | 5 | – | 6 | **8** | 0 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | X | | | | | | |
| 3 | 1–5 | X | | | | | |
| 4 | X | 1–7 | X | | | | |
| 5 | X | 1–5 | X | 2–7<br>3–8 | | | |
| 6 | 2–7<br>3–8 | X | 3–8 | X | X | | |
| 7 | 2–7 | X | 1–5 | X | X | 4–6 | |
| 8 | X | 1–5 | X | 4–6 | 3–8 | X | X |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

## A. Moore machine

2nd pass

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | X | | | | | | |
| 3 | ~~1-5~~ | X | | | | | |
| 4 | X | ~~1-7~~ | X | | | | |
| 5 | X | ~~1-5~~ | X | ~~2-7 3-8~~ | | | |
| 6 | ~~2-7 3-8~~ | X | ~~3-8~~ | X | X | | |
| 7 | ~~2-7~~ | X | ~~1-5~~ | X | X | ~~4-6~~ | |
| 8 | X | ~~1-5~~ | X | ~~4-6~~ | ~~3-8~~ | X | X |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

1-5 can't be merged, so 2-5, 2-8, 3-7 can not be merged either

1-6 if 2-7 et 3-8;
2-7 can't be merged; so 4-5 either

**There are no possible simplifications !!!**

# B. Meally machine

States 1-5, 2-7, 3-8 and 4-6 are not equivalents
(two stable states in the same column with different outputs)

|  | ab |  |  |  |  |
|---|---|---|---|---|---|
|  | 00 | 01 | 11 | 10 | Z |
| 1 | **1** | 2 | - | 3 | 1 |
| 2 | 1 | **2** | 4 | - | 0 |
| 3 | 5 | - | 6 | **3** | 1 |
| 4 | - | 7 | **4** | 8 | 0 |
| 5 | **5** | 2 | - | 3 | 0 |
| 6 | - | 7 | **6** | 8 | 1 |
| 7 | 1 | **7** | 4 | - | 1 |
| 8 | 5 | - | 6 | **8** | 0 |

|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| 2 | OK |  |  |  |  |  |  |
| 3 | 1-5 | 1-5 |  |  |  |  |  |
| 4 | 3-8 | 2-7 | 4-6 3-8 |  |  |  |  |
| 5 | X | 1-5 | OK | 2-7 3-8 |  |  |  |
| 6 | 2-7 3-8 | 2-7 | 3-8 | X | 2-7 |  |  |
| 7 | 2-7 | X | 1-5 | OK | 2-7 | 4-6 |  |
| 8 | 1-5 3-8 | 1-5 | X | 4-6 | 3-8 | OK | 1-5 |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

## B. Meally machine

2nd pass & state fusion

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | OK | | | | | | |
| 3 | 1 X 5 | 1 X 5 | | | | | |
| 4 | 3 X 8 | 2 X 7 | 4–6 X 3–8 | | | | |
| 5 | X | 1 X 5 | OK | 2–7 X 3–8 | | | |
| 6 | 2–7 X 3–8 | 2 X 7 | 3 X 8 | X | 2 X 7 | | |
| 7 | 2 X 7 | X | 1 X 5 | OK | 2 X 7 | 4 X 6 | |
| 8 | 1–5 X 3–8 | 1 X 5 | X | 4 X 6 | 3 X 8 | OK | 1 X 5 |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

| | |
|---|---|
| **1 —— 2** | **1** |
| **3 —— 5** | **2** |
| **4 —— 7** | **3** |
| **6 —— 8** | **4** |

# B. Meally machine

## Final optimised state table

| | ab | | | | |
|---|---|---|---|---|---|
| | 00 | 01 | 11 | 10 | z |
| 1 | **1** | 2 | – | 3 | 1 |
| 2 | 1 | **2** | 4 | – | 0 |
| 3 | 5 | – | 6 | **3** | 1 |
| 4 | – | 7 | **4** | 8 | 0 |
| 5 | **5** | 2 | – | 3 | 0 |
| 6 | – | 7 | **6** | 8 | 1 |
| 7 | 1 | **7** | 4 | – | 1 |
| 8 | 5 | – | 6 | **8** | 0 |

**1 — 2 → 1**

**3 — 5 → 2**

**4 — 7 → 3**

**6 — 8 → 4**

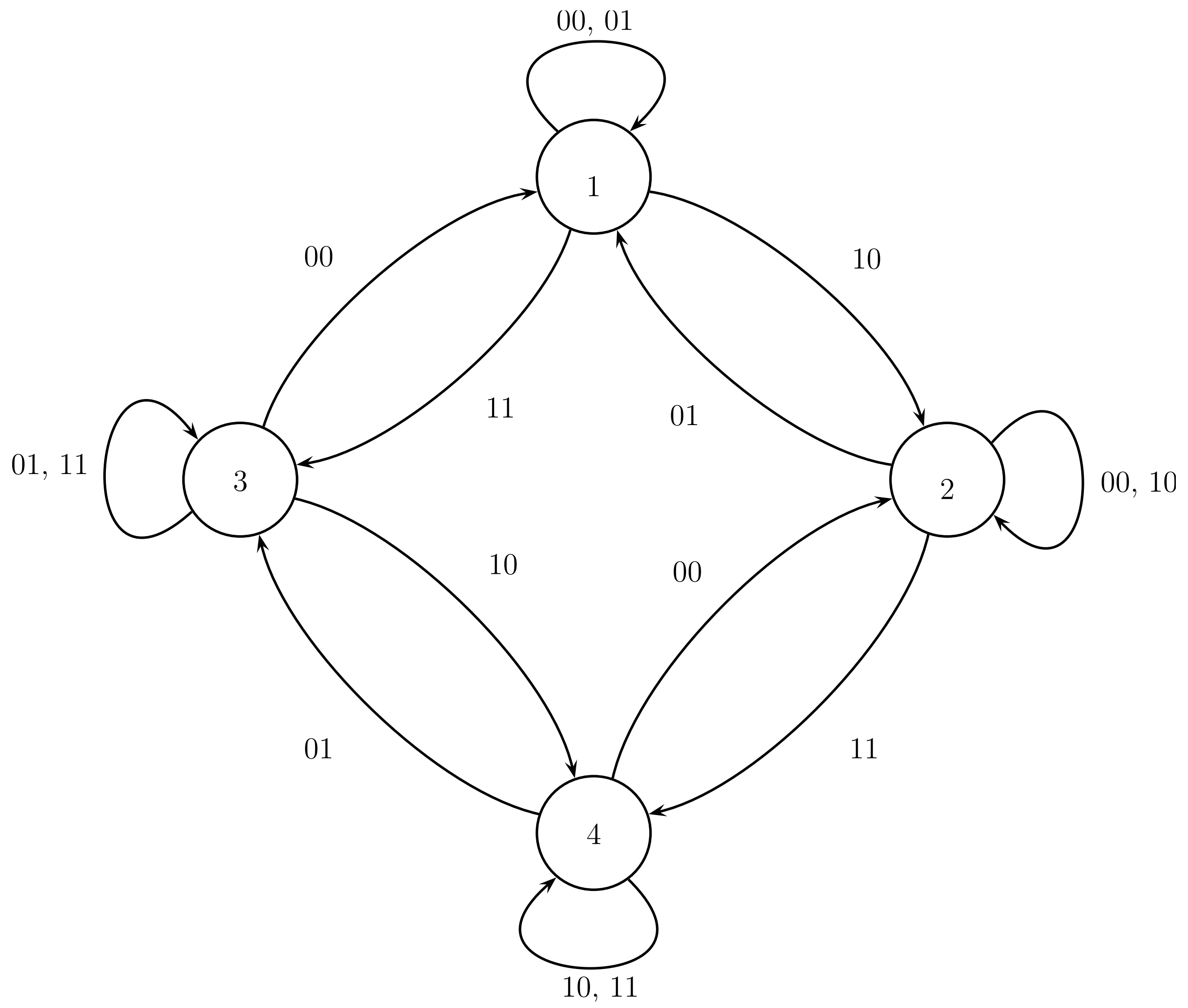
| | ab | | | |
|---|---|---|---|---|
| | 00 | 01 | 11 | 10 |
| 1 | **1/1** | **1/0** | 3 | 2 |
| 2 | **2/0** | 1 | 4 | **2/1** |
| 3 | 1 | **3/1** | **3/0** | 4 |
| 4 | 2 | 3 | **4/1** | **4/0** |

# B. Meally machine

## Optimised state t**able and state graph**

ab

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

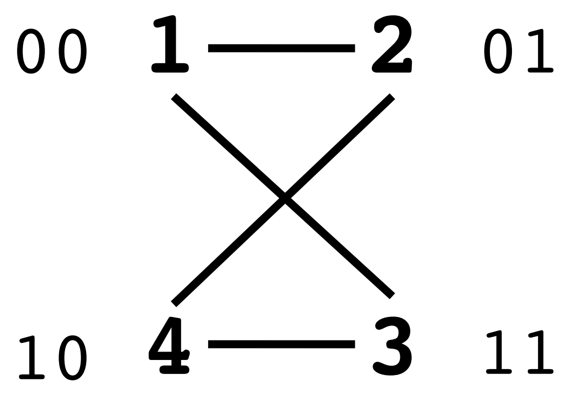


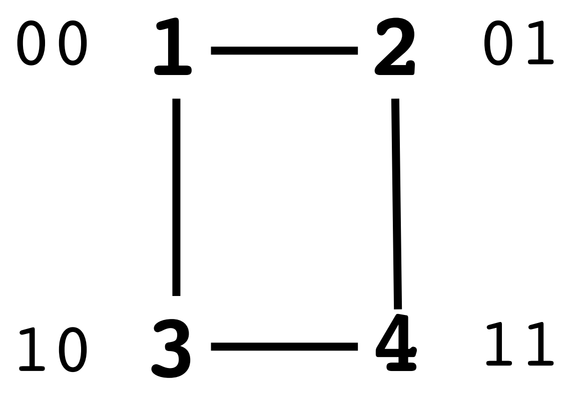
## Asynchronous implementation (Moore machine only)

# B. Mealy machine: encoding

State encoding graph, encoding choice and encoded state table:

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





| | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | **00** | **00** | 10 | 01 |
| 01 | **01** | 00 | 11 | **01** |
| 11 | 01 | 10 | **11** | **11** |
| 10 | 00 | **10** | **10** | 11 |

# B. Meally machine: feedback functions

ab

| $Y_1Y_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | **00** | **00** | 10 | 01 |
| 01 | **01** | 00 | 11 | **01** |
| 11 | 01 | 10 | **11** | **11** |
| 10 | 00 | **10** | **10** | 11 |

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 |

ab

| $Y_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | **0** | **0** | 0 | 1 |
| 01 | **1** | 0 | 1 | **1** |
| 11 | 1 | 0 | **1** | **1** |
| 10 | 0 | **0** | **0** | 1 |

$Y_1 = ab+y_1(a+b)$

$Y_2 = ab'+y_2(a+b')$

## B. Meally machine: outputs

Attention !!! we have inverted the state 3 and 4 during encoding, so this has to be taken into account; This is Meally machine beware of shared transitions (not here) & multiple source states (all 4 states)

ab

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

ab

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

$$Z = b'\, y'_1 y'_2 + a\, y_1' y_2 + b y_1 y_2 + a' y_1 y_2'$$

## Implementation using JK-FF

# 1. State encoding

We start with the optimised state table and we pick the first encoding we can think of, leaving the race conditions as they are (red coloured cells in the table on the right); we can & should do this because we are implementing a **synchronous circuit !!!**

ab

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

ab

|  | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | **00/1** | **00/0** | 11 | 01 |
| 01 | **01/0** | 00/0 | 10 | **01/1** |
| 11 | 00 | **11/1** | **11/0** | 10 |
| 10 | 01 | 11 | **10/1** | **10/0** |

## 2. Excitation table of the system

This one is independent of the memory element choice:

|  | ab |  |  |  |
|---|---|---|---|---|
|  | 00 | 01 | 11 | 10 |
| 00 | **00/1** | **00/0** | 11 | 01 |
| 01 | **01/0** | 00 | 10 | **01/1** |
| 11 | 00 | **11/1** | **11/0** | 10 |
| 10 | 01 | 11 | **10/1** | **10/0** |

$\Rightarrow$

|  | ab |  |  |  |
|---|---|---|---|---|
|  | 00 | 01 | 11 | 10 |
| 00 | $\mu_0\mu_0$ | $\mu_0\mu_0$ | $\varepsilon\varepsilon$ | $\mu_0\varepsilon$ |
| 01 | $\mu_0\mu_1$ | $\mu_0\delta$ | $\varepsilon\delta$ | $\mu_0\mu_1$ |
| 11 | $\delta\delta$ | $\mu_1\mu_1$ | $\mu_1\mu_1$ | $\mu_1\delta$ |
| 10 | $\delta\varepsilon$ | $\mu_1\varepsilon$ | $\mu_1\mu_0$ | $\mu_1\mu_0$ |

# 3. Excitation table of FF

System excitation table is split in two tables: one per memory element

ab

| | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | $\mu_0\mu_0$ | $\mu_0\mu_0$ | $\varepsilon\varepsilon$ | $\mu_0\varepsilon$ |
| 01 | $\mu_0\mu_1$ | $\mu_0\delta$ | $\varepsilon\delta$ | $\mu_0\mu_1$ |
| 11 | $\delta\delta$ | $\mu_1\mu_1$ | $\mu_1\mu_1$ | $\mu_1\delta$ |
| 10 | $\delta\varepsilon$ | $\mu_1\varepsilon$ | $\mu_1\mu_0$ | $\mu_1\mu_0$ |

ab

| $M_1$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | $\mu_0$ | $\mu_0$ | $\varepsilon$ | $\mu_0$ |
| 01 | $\mu_0$ | $\mu_0$ | $\varepsilon$ | $\mu_0$ |
| 11 | $\delta$ | $\mu_1$ | $\mu_1$ | $\mu_1$ |
| 10 | $\delta$ | $\mu_1$ | $\mu_1$ | $\mu_1$ |

ab

| $M_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | $\mu_0$ | $\mu_0$ | $\varepsilon$ | $\varepsilon$ |
| 01 | $\mu_1$ | $\delta$ | $\delta$ | $\mu_1$ |
| 11 | $\delta$ | $\mu_1$ | $\mu_1$ | $\delta$ |
| 10 | $\varepsilon$ | $\varepsilon$ | $\mu_0$ | $\mu_0$ |

# 4. Excitation tables for M1 & M2 memories (JK)

ab

| $M_1$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | $\mu_0$ | $\mu_0$ | $\varepsilon$ | $\mu_0$ |
| 01 | $\mu_0$ | $\mu_0$ | $\varepsilon$ | $\mu_0$ |
| 11 | $\delta$ | $\mu_1$ | $\mu_1$ | $\mu_1$ |
| 10 | $\delta$ | $\mu_1$ | $\mu_1$ | $\mu_1$ |

| | J | K |
|---|---|---|
| $\mu_0$ | 0 | – |
| $\varepsilon$ | 1 | – |
| $\delta$ | – | 1 |
| $\mu_1$ | – | 0 |

ab

| $M_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | $\mu_0$ | $\mu_0$ | $\varepsilon$ | $\varepsilon$ |
| 01 | $\mu_1$ | $\delta$ | $\delta$ | $\mu_1$ |
| 11 | $\delta$ | $\mu_1$ | $\mu_1$ | $\delta$ |
| 10 | $\varepsilon$ | $\varepsilon$ | $\mu_0$ | $\mu_0$ |

ab

| $J_1K_1$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0– | 0– | 1– | 0– |
| 01 | 0– | 0– | 1– | 0– |
| 11 | –1 | 0 | 0 | 0 |
| 10 | –1 | 0 | 0 | 0 |

ab

| $J_2K_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0– | 0– | 1– | 1– |
| 01 | 0 | –1 | –1 | 0 |
| 11 | –1 | 0 | 0 | –1 |
| 10 | 1– | 1– | 0– | 0– |

# 5. K-Maps & optimised excitation functions

ab

| $J_1K_1$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0- | 0- | 1- | 0- |
| 01 | 0- | 0- | 1- | 0- |
| 11 | -1 | 0 | 0 | 0 |
| 10 | -1 | 0 | 0 | 0 |

ab

| $J_2K_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0- | 0- | 1- | 1- |
| 01 | 0 | -1 | -1 | 0 |
| 11 | -1 | 0 | 0 | -1 |
| 10 | 1- | 1- | 0- | 0- |

ab

| $J_1$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 1 | 0 |
| 01 | 0 | 0 | 1 | 0 |
| 11 | - | - | - | - |
| 10 | - | - | - | - |

ab

| $K_1$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | - | - | - | - |
| 01 | - | - | - | - |
| 11 | 1 | 0 | 0 | 0 |
| 10 | 1 | 0 | 0 | 0 |

ab

| $J_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 1 | 1 |
| 01 | - | - | - | - |
| 11 | - | - | - | - |
| 10 | 1 | 1 | 0 | 0 |

ab

| $K_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | - | - | - | - |
| 01 | 0 | 1 | 1 | 0 |
| 11 | 1 | 0 | 0 | 1 |
| 10 | - | - | - | - |

$J_1=ab$

$K_1=a'b'$

$J_2=y_1a'+y_1'a$

$K_2=y_1b'+y_1'b$

# 6. Circuit output

ab

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

ab

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

$$
\begin{aligned}
Z = \; & b'y'_1y'_2 \\
& + a y_1' y_2 \\
& + a' y_1 y_2 \\
& + b y_1 y_2'
\end{aligned}
$$

# 7. Circuit diagram

$J_1 = ab$

$K_1 = a'b'$

$J_2 = y_1a'+y_1'a$

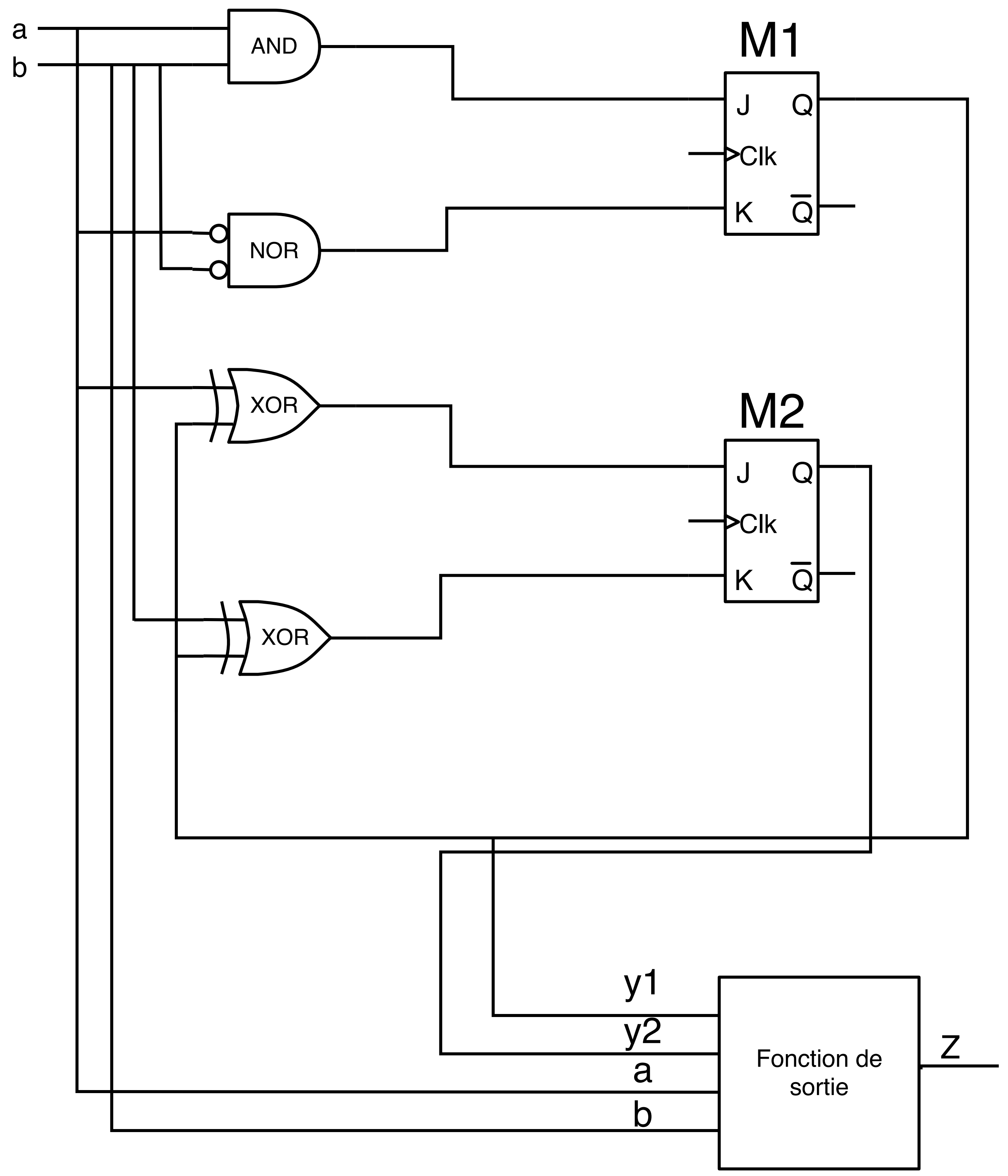
$K_2 = y_1b'+y_1'b$

$Z = b'y_1y_2$

$+ a\ y_1'y_2$

$+ a'by_1$

$+ by_1y_2'$



## Implementation using D-FF

# Excitation tables assuming D Flip-Flop

ab

| $M_1$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | $\mu_0$ | $\mu_0$ | $\varepsilon$ | $\mu_0$ |
| 01 | $\mu_0$ | $\mu_0$ | $\varepsilon$ | $\mu_0$ |
| 11 | $\delta$ | $\mu_1$ | $\mu_1$ | $\mu_1$ |
| 10 | $\delta$ | $\mu_1$ | $\mu_1$ | $\mu_1$ |

|  | D |
|---|---|
| $\mu_0$ | 0 |
| $\varepsilon$ | 1 |
| $\delta$ | 0 |
| $\mu_1$ | 1 |

ab

| $M_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | $\mu_0$ | $\mu_0$ | $\varepsilon$ | $\varepsilon$ |
| 01 | $\mu_1$ | $\delta$ | $\delta$ | $\mu_1$ |
| 11 | $\delta$ | $\mu_1$ | $\mu_1$ | $\delta$ |
| 10 | $\varepsilon$ | $\varepsilon$ | $\mu_0$ | $\mu_0$ |

ab

| $D_1$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 1 | 0 |
| 01 | 0 | 0 | 1 | 0 |
| 11 | 0 | 1 | 1 | 1 |
| 10 | 0 | 1 | 1 | 1 |

$D_1=ab+y_1(b+a)$

ab

| $D_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 1 | 1 |
| 01 | 1 | 0 | 0 | 1 |
| 11 | 0 | 1 | 1 | 0 |
| 10 | 1 | 1 | 0 | 0 |

$D_2 = y_1y_2'a' + y_1y_2b + y_1'y_2b' + y_1'y_2'a$

## Output function synthesis

As for the asynchronous circuit

ab

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

ab

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

$Z = b'y'_1y'_2 + ay_1'y_2$

$+a'by_1 + by_1y_2'$

# Comparison of different implementations

**JK**

$J_1 = ab$

$K_1 = a'b'$

$J_2 = y_1a'+y_1'a$

$K_2 = y_1b'+y_1'b$

$Z = b'\ y'_1y'_2 + a\ y_1'y_2 + a'by_1 + by_1y_2'$

**D**

$D_1 = ab+y_1(b+a)$

$D_2 = y_1y_2'a'+ y_1y_2b + y_1'y_2b'+ y_1'y_2'a$

$Z = b'y'_1y'_2 + ay_1'y_2 + a'by_1 + by_1y_2'$

If you know the exact price of the gates you could make the right choice

# Beware of the following errors (0 at exam)

- Not know how to derive the excitation tables for different FFs
- Solve the race conditions before synchronous implementation
- Make the excitation table for the system based on inputs and not for states !!! (yes I do see these from time to time …)

| | ab | | | |
|---|---|---|---|---|
| | 00 | 01 | 11 | 10 |
| 00 | **00/1** | **00/0** | 11 | 01 |
| 01 | **01/0** | 00 | 10 | **01/1** |
| 11 | 00 | **11/1** | **11/0** | 10 |
| 10 | 01 | 11 | **10/1** | **10/0** |

| | ab | | | |
|---|---|---|---|---|
| | 00 | 01 | 11 | 10 |
| 00 | $\mu_0\mu_0$ | $\mu_0\varepsilon$ | $\varepsilon\varepsilon$ | $\varepsilon\mu_0$ |
| 01 | | | | |
| 11 | | | | |
| 10 | | | | |