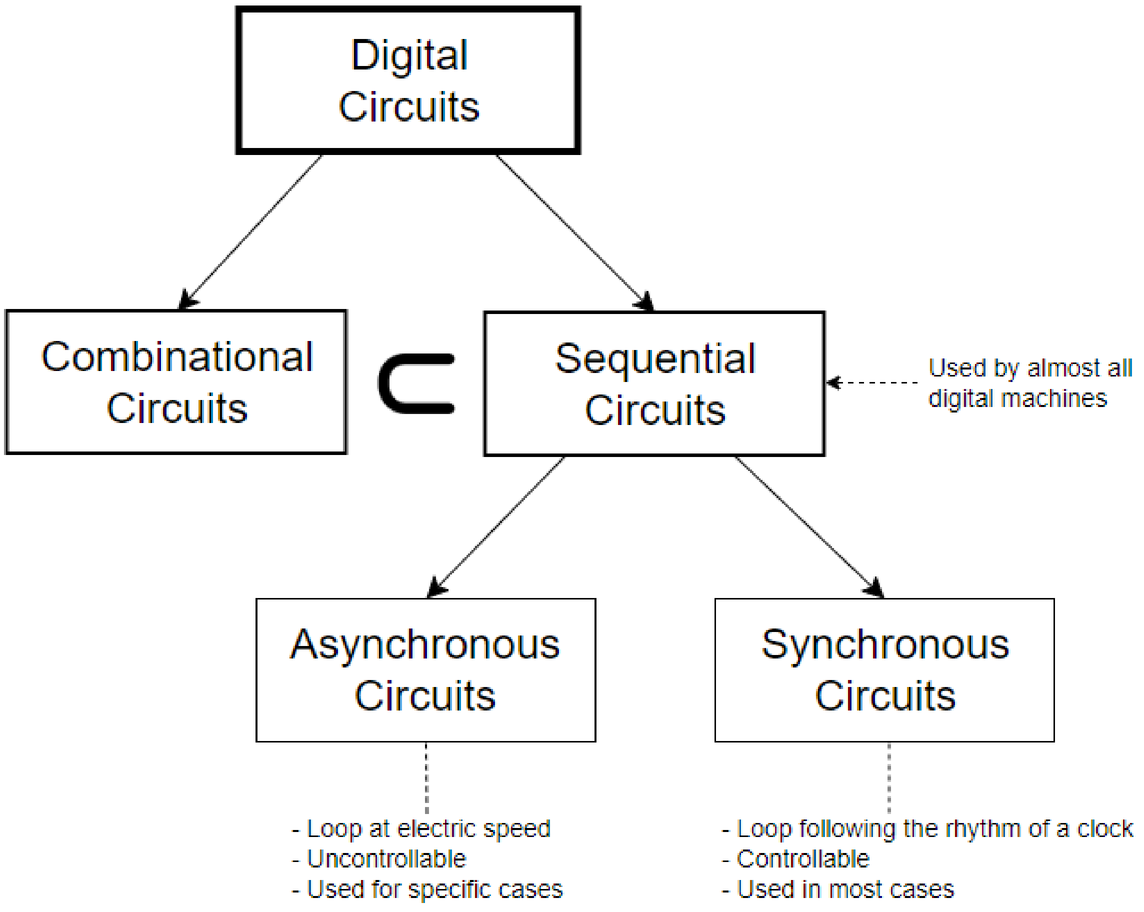
Sétif 1 University
Faculty of Sciences
Common Core Mathematics and Computer Science

2nd semester (year 2023/24)
Machine Structures 2 course
Kara Abdelaziz professor

Chapter 2 : Sequential Circuits

1. Introduction



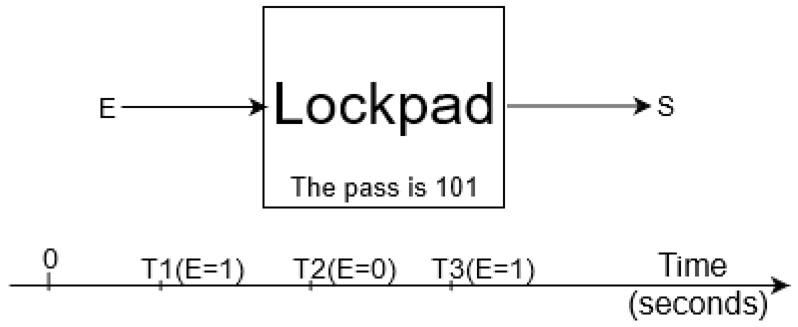
Remark : Throughout this lecture, the term Sequential Circuits (or S.C.) implicitly refers to Synchronous Sequential Circuits.

2. Definition

A Sequential Circuit is a digital circuit that its outputs don't only depends on the inputs, as for combinational circuits, but also on the history of previous inputs.

Example :

We take the example of the Lockpad, but in this time instead of having several inputs like of the Combinational Circuit, it only uses a one single input. You can see the representation in the following diagram.



The pass word 101 must be entered digit by digit over time. If we are at the 3rd digit, the Sequential Circuit must know that the 2 previous digits were 1 followed by 0 to open, otherwise it remains closed. So we say that the Sequential Circuit has a memory to remember the previous sequence. The term *Sequential* comes in relation to this behavior.

3. Sequential Circuits characteristics

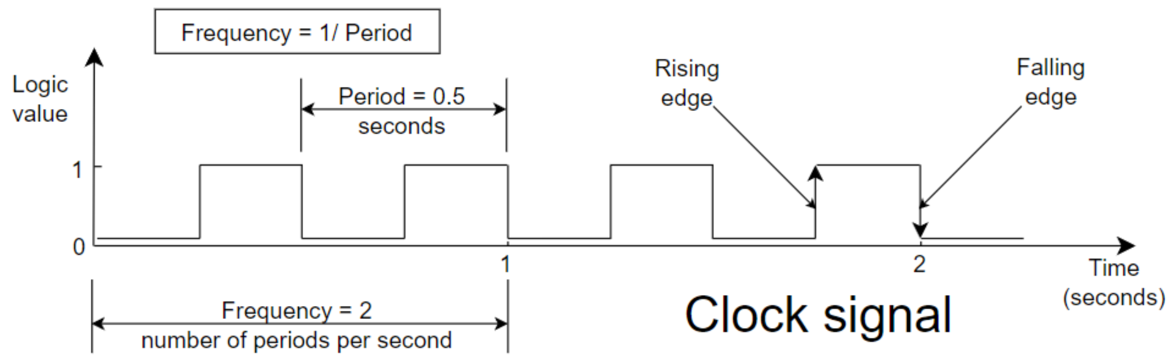
- ➔ They are <u>dynamic</u>, they evolve over time, the output of a Sequential Circuit can change over time even if the inputs remain unchanged.
- ➔ Unlike Combinational Circuits, they must contain a <u>loop</u>. The most common case is the return of a part of the outputs to a part of the inputs.
- ➔ Must have a <u>memory</u> to remember the previous sequence of inputs.
- ➔ The rate of evolution of a Sequential Circuit is punctuated by a <u>clock</u>. Generally one iteration (loop) is performed for each tick (pulse) of the clock.

Remark : It is important to notice that there is also another type of Sequential Circuits in which there are no loops. These are called *Pipelines*, and will be studied later in this chapter.

<u>Clock</u> :

- ➔ It is a periodic digital signal based on 0 and 1, characterized by a Period and a Frequency (as shown in the figure).
- ➔ The signal is generated by digital electronic circuit often also called the Clock.
- ➔ The signal makes it possible to set the pace of operation of a Sequential Circuit.

The diagram represents the evolution of a clock signal over time.

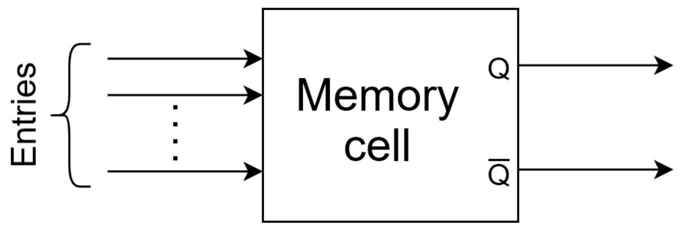


- ➔ Typically, it is the rising edge in the signal that represents the clock tick, telling the Sequential Circuit to complete the current operation and move on to the next one.
- ➔ A single one operation is performed by the Sequential Circuit over a clock period, thus the period is delimited by 2 successive rising edges.
- ➔ Every Sequential Circuit must have a clock input, often called <u>clk</u>.
- ➔ Different form other inputs, the clock signal input is not an information or data input.
- ➔ A processor with a frequency of 4 GHz for example, actually performs 4 billion operations per second. 4 GHz is the frequency of its clock.

4. Memories cells (Latch and FlipFlop)

To understand the memory technology used in Digital Circuits, we must go through the levels of evolution of this technology throughout its history. This evolution is roughly broken down into 4 stages:

1. Bi-stables
2. RS-Latch
3. D-Latch
4. D-FlipFlop

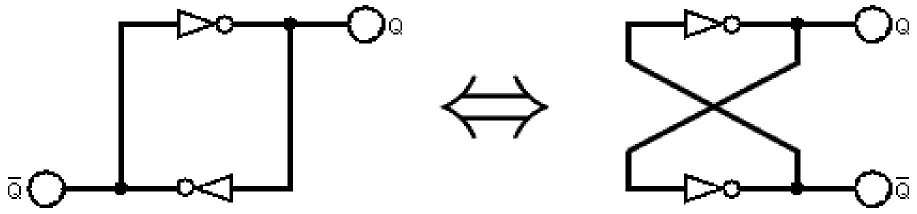


- ➔ All 4 circuits at the top are elementary circuits called memory cells, they represent circuits for saving or memorizing a single bit, they can memorize the value 0 or 1.
- ➔ The memory cell outputs indicate the value stored inside the cell, the value is in Q ($\overline{Q}$ is always the inverse of Q).
- ➔ The inputs allows to enter of a new value and override the old value, this operation is called <u>write</u> in memory, or to command the cell to memorize the current value.

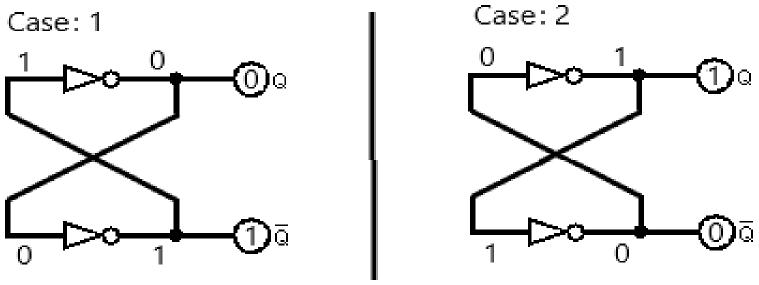
4.1. Bi-stable

- ➔ A Bi-stable circuit is formed according to the diagram at the bottom, where the circuit forms a cycle in which the output of the 1st NOT gate arrives at the input of the 2nd NOT gate, and the output of the 2nd NOT gate NOT arrives at the entrance to the 1st.
- ➔ The Bi-stable circuit does not have any inputs, and have 2 outputs Q and $\overline{Q}$.

Diagram of a Bi-stable circuit :



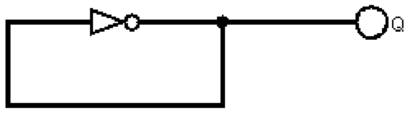
- ➔ When the circuit is turn on (Time = 0 seconds), the circuit can follow 2 possible scenarios. In the diagram below, the circuit can stabilize on Case 1: Q = 0 and $\overline{Q}$ = 1, or Case 2: Q = 1 and $\overline{Q}$ = 0, and the signal loops infinitely in a stable manner on the one of the 2 Cases. Hence the name Bi-stable.
- ➔ Experimentally, when starting up the circuit, it is impossible to know on which case the circuit will stabilize, it is totally random. Sometimes it is in relation to the fastest gate of the 2.
- ➔ But once the circuit has stabilized on a given Case, its configuration will no longer change throughout the all operating time. This phenomenon of permanent stability represents the behavior of memorization.



- ➔ Thus a Bi-stable circuit represents a 1-bit memory cell.
- ➔ By convention, a memory cell must always have 2 outputs named Q and $\overline{Q}$ ($\overline{Q}$ is necessarily the inverse of Q).
- ➔ The Bi-stable circuit is experimental and not real, it demonstrates the concept of memorization (stability) and the construction of memory from logic gates.
- ➔ But it remains impractical in real world, due to the impossibility of controlling the contents of the memory in the absence of inputs to the circuit.
- ➔ This is why instead, RS-Latch is used like memory for real world circuits. Because the content of the cell can be modified using its inputs.

Remark 1: There exists a 3rd State different from the 2 previously mentioned stability states. it is called meta-stable state, in which a value in the middle of 0 and 5 Volts (close to 2.5 Volts) propagates in a loop through the circuit, giving it a prohibited indefined logic value. Although the probability for a Bi-stable to fall into the meta-stable state is outrageously very low.

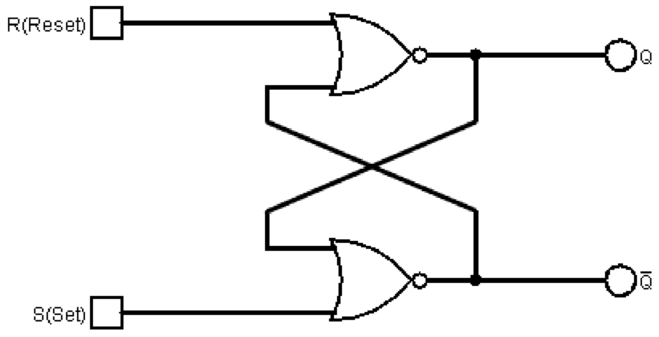
Remark 2: Not all circuits in cycle fall into stable states, in fact it is often the opposite. We can see for example the simple circuit at the bottom, it will never fall into a stable state. The output Q will alternate indefinitely between 0 and 1 for each iteration (loop) of the signal. This kind of circuit is called *Astable*, and they are generally used to produce a clock signal.



Remark 3: Bi-stable latch circuits, like RS-Latch, which comes just after, are asynchronous circuits. They are not controlled by a clock and they loop at the speed of electricity (close to the speed of light).

4.2. RS-Latch

➔ The RS-Latch circuit is shown in the diagram below. Tt is formed by 2 crossed NOR gates, such that the output of one is connected to the input of the other and vice versa, and thus forming a cycle.



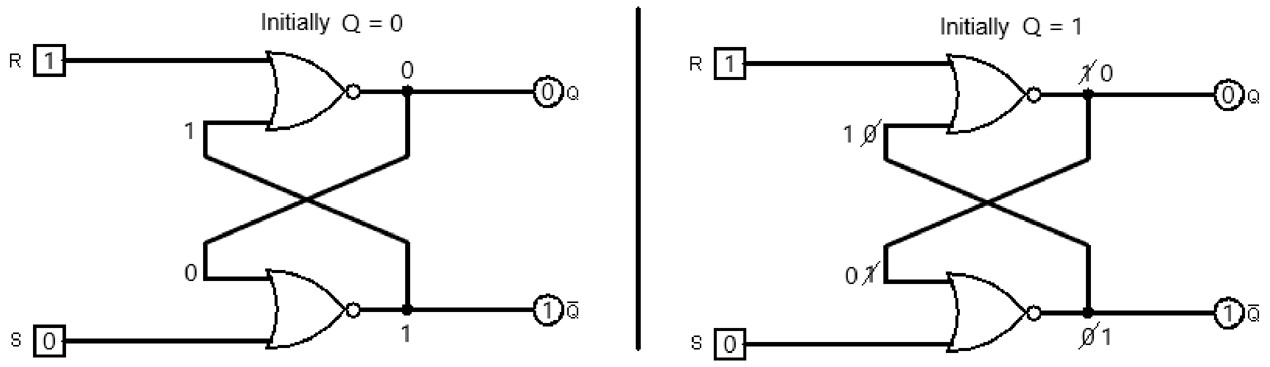
➔ To understand the operation of the circuit, an analysis of its behavior on all possible cases of inputs is made on the 4 following diagrams.

➔ The diagrams successively represent the case R=1 S=0, R=0 S=1, R=0 S=0, and the case R=1 S=1. R for *Reset* and S for *Set*.

➔ Initially at T=0, while turning on the circuit. The State of the circuit is random and unknown, although the two possible scenarios are Q=0 or Q=1, for each Input Case, these 2 cases must be studied.

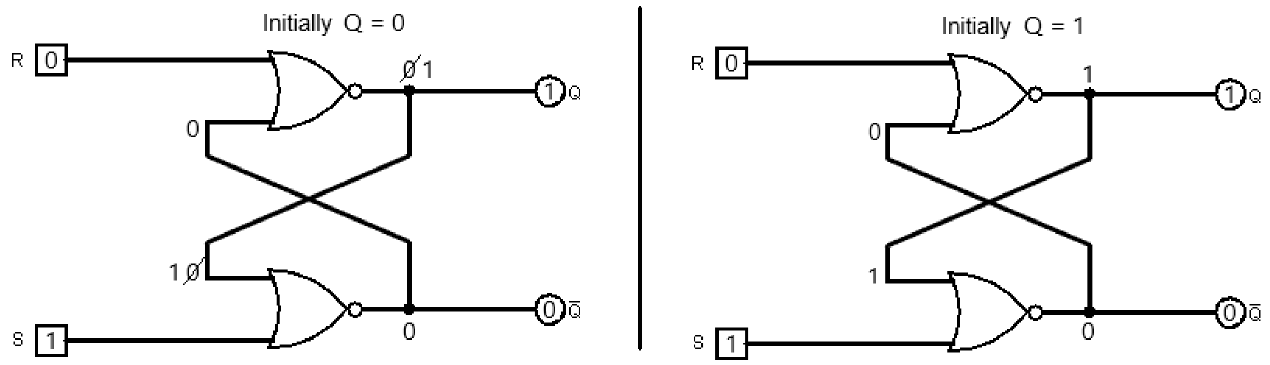
- ➔ It is also possible to track the values of $\overline{Q}$ instead of Q for the Initial state, but in the end it would be the same thing.

For R = 1 and S = 0



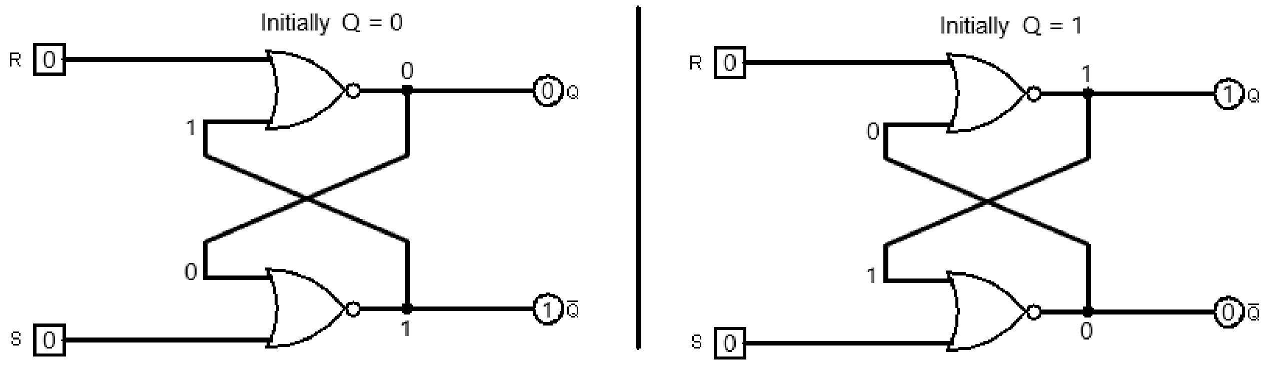
- ➔ For Case R=1 S=0 and initially Q=0, the value of Q=0 enters the 2nd NOR at the bottom and produces $\overline{Q}=\overline{S+0}$=1. The value of $\overline{Q}$=1 enters the 1st NOR and produces Q=$\overline{R+1}$=0. The cycle is completed without the starting point Q changing its value. Then the signal will loop like this infinitely, so Q and $\overline{Q}$ are stabilized on the values 0 and 1 respectively.
- ➔ For the case where initially Q=1, the value of Q is used in the 2nd NOR, producing $\overline{Q}$=0. In the same way $\overline{Q}$ is used by the first upper NOR and this time gives 0 which will overwrite the old initial state 1 of Q. At this point, the circuit is not yet stabilized and we must continue the execution. The 2nd button gate with Q=0, this time produces 1 overwriting the old 0, which is in turn used by the 1st gate and produces 0. So at this point, in the 2nd iteration, Q is not overwritten and remains 0 for the next loops, and the circuit has just stabilized on Q=0 and Q=1.

For R = 0 and S = 1



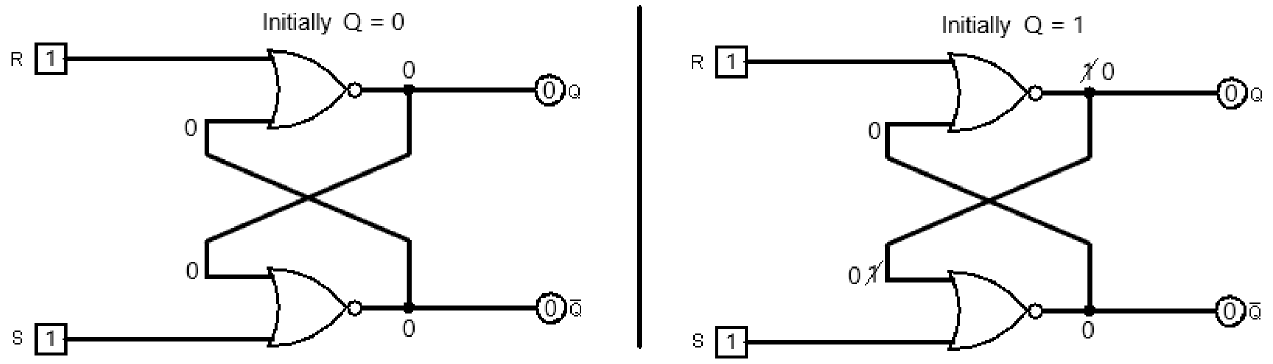
- ➔ In the same way, the execution of the circuits for the remaining cases are executed.
- ➔ There are cases where the circuit stabilizes in the 1st iteration, and sometimes it stabilizes in the 2nd iteration.
- ➔ A circuit stabilizes when the execution performs a complete cycle and the starting point value don't changed.

For R = 0 and S = 0



- ➔ We notice that for R=0 S=1, whatever the initial state, the RS-Latch stabilizes on Q=0 $\overline{Q}$=1. And vice versa for R=1 S=0 stabilizes on Q=1 $\overline{Q}$=0.
- ➔ For R=0 S=0 it is different, in the sense that the latch stabilizes depending of the initial state. If the initial state is Q=0 the circuit stabilizes on Q=0 $\overline{Q}$=1, and conversely if Q=1 the circuit stabilizes on Q=1 $\overline{Q}$=0.
- ➔ Another way to analyze the circuit R=0 S=0 is to generalize this observation and to deduce that the circuit preserves the value of the previous loop Q\` just before the circuit passes to R=0 S=0. If before passing to R=0 S=0 the circuit was looping on Q\`=0, the cell will keep this value (Q=0 and $\overline{Q}$=1) for next cycles. And vice versa, if Q\`=1 the cell will save Q=1 and $\overline{Q}$=0.
- ➔ This last property is very important, it shows the mechanism for memorizing a newly changed value in the memory cell. What was missing in the previous a Bi-stable latch.

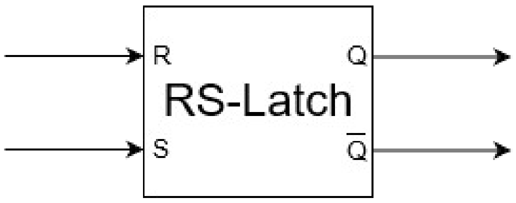
For R = 1 and S = 1



- ➔ For the circuits R=1 S=1, whatever the initial state, the outputs always produces Q=0 $\overline{Q}$=0, which represents an incoherent output. Q must always be the inverse of $\overline{Q}$. This is why the input combination R=1 S=1 is prohibited for RS-Latch circuits, it causes them to enter an inconsistent state.
- ➔ A summary of the 4 input combinations explaining the working of an RS-Latch cell is shown in the table below. The R=1 S=0 (Reset) and R=0 S=1 (Set) modes allows the modification of the cell value (it is a *write* operation). The R=0 S=0 (Memorize) mode is symbolized in the table by Q\` and $\overline{Q}$\`, which respectively represent the value of Q and $\overline{Q}$ in the previous iteration. The combination R=1 S=1 is prohibited.

R	S	Q	$\overline{Q}$	
1	0	0	1	The Reset mode resets Q to 0
0	1	1	0	The Set mode sets Q to 1
0	0	Q`	$\overline{Q}$`	The Memorize (00) entry stores Q
1	1	0	0	Entry (11) is prohibited

➔ The Global Schem of an RS-Latch cell is shown in the figure as follows.

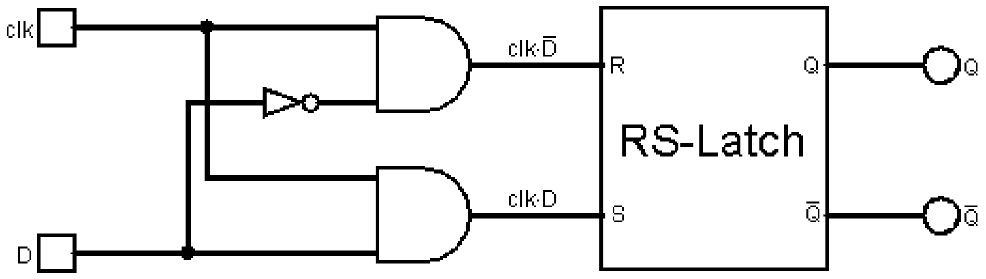


4.3. D-Latch

➔ The circuit diagram of the D-Latch is presented below. We can notice that a D-Latch uses inside an RS-Latch to work.

➔ In the name of the D-Latch, D comes from *Data*.

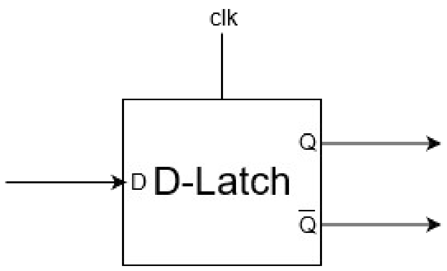
➔ The D-Latch is an improvement upon the RS-Latch, in the sense that only one input is dedicated to changing the value of the memory cell, instead of 2 for the RS-Latch. The 2nd input is dedicated to the clock *clk*, an essential input for any Synchronous Sequential Circuit.



➔ To understand how the D-Latch works, an execution of the 4 possible combinations of inputs is performed and its result is presented in the following table :

clk	D(Data)	Q	$\overline{Q}$	
0	0	Q`	$\overline{Q}$`	Memorization when clk=0
0	1	Q`	$\overline{Q}$`	Memorization when clk=0
1	0	0	1	Q=0 for D=0 when clk=1
1	1	1	0	Q=1 for D=1 when clk=1

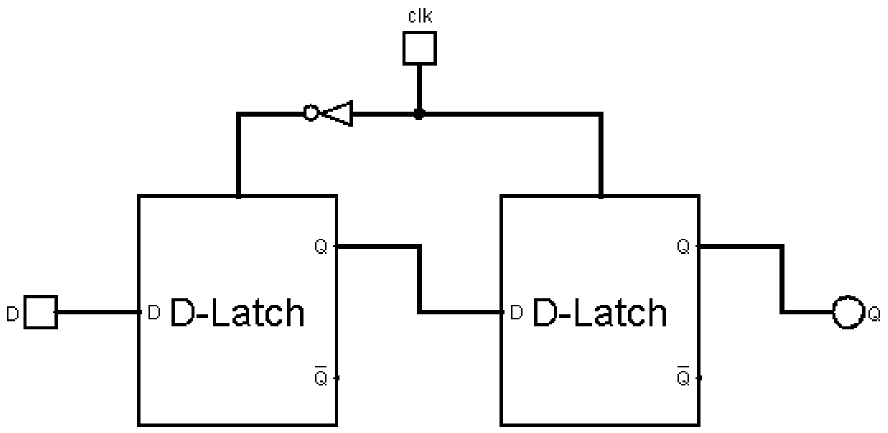
- ➔ By observation, the clock puts the latch in 2 modes; Memorize when the clock is at 0. And Change (or Write) a new value if the clock is at 1. In the latter case, the value D override the old value of the latch.
- ➔ When in *Memorize* mode, the D-Latch is sometimes said to be closed (or *opaque*) because the D value cannot be changed inside the cell. While during *Change* mode, the D-Latch is said to be open (or *transparent*), the value of D can enter and modify the inside of the cell (which is by the way always accessible on Q and $\overline{Q}$).
- ➔ Another advantage of the D-Latch compared to the RS-Latch is that there is no risk of falling into the forbidden case 11 as in the RS-Latch.
- ➔ The representation of the Global Scheme of a D-Latch cell is presented as follows.



Remark : It's also very simple to create a D-Latch with a single 2-1 Mux, which I let you guess the solution by yourselves.

4.4. D-FlipFlop

- ➔ A D-FlipFlop is a memory cell internally made up of 2 D-Latches placed in series one after the other. One of the 2 is has an inverted clock, as shown in the diagram below.
- ➔ A D-FlipFlop works in a similar manner than a D-Latch. The only difference is that the time of transparency, the modification of the value inside the cell is very reduced, it happen during the rising edge of the clock.



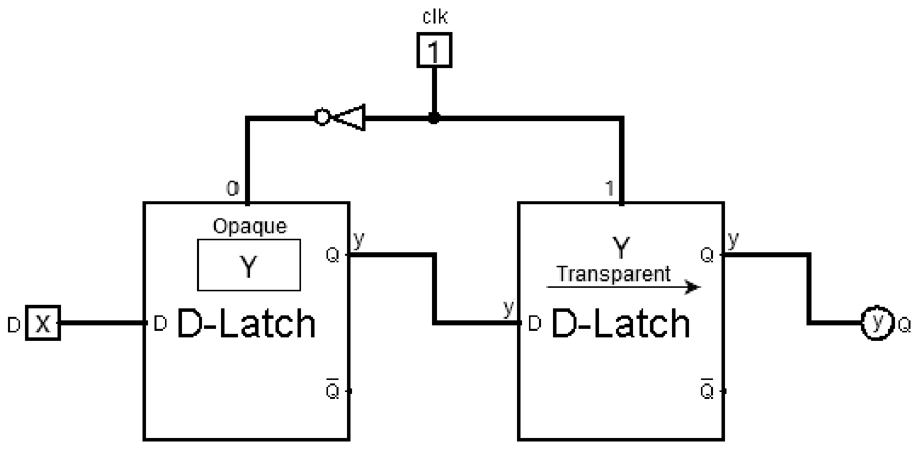
- ➔ So, the few picoseconds of the clock's rising edge time, which represents the change of the clock from the value 0 to 1 (see the clock diagram), is the only instant where the inside of the cell can be modified.

➔ According to the internal diagram of the D-FlipFlop. Due to the inverted clock, the 2 D-Latch always work alternately. If one is transparent the other is opaque, and vice versa.

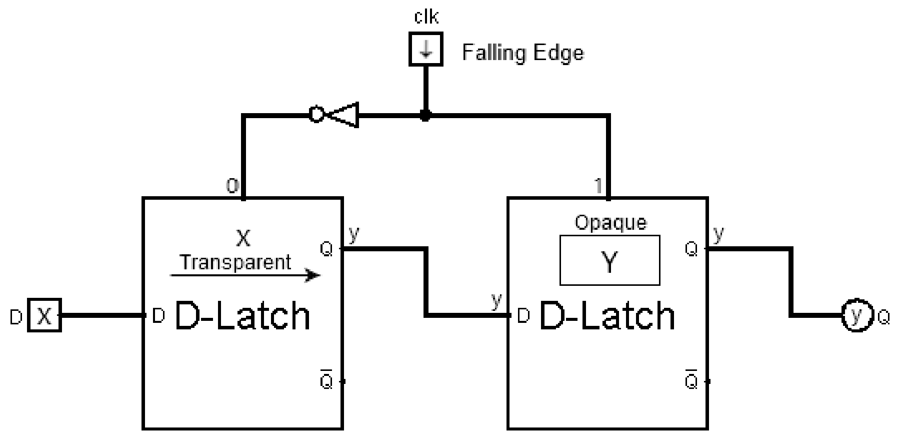
➔ To understand how D-FlipFlop works, we will execute an example crossing the successive phases in a clock period, which are ; the value 1, the falling edge, the value 0, and the rising edge.

➔ Let's assume on the example that the cell contains Y inside (binary value), and the input D provides the value X (binary too). The most important thing about this example is to observe the moment of entering the X value inside the cell and overwriting the Y value.

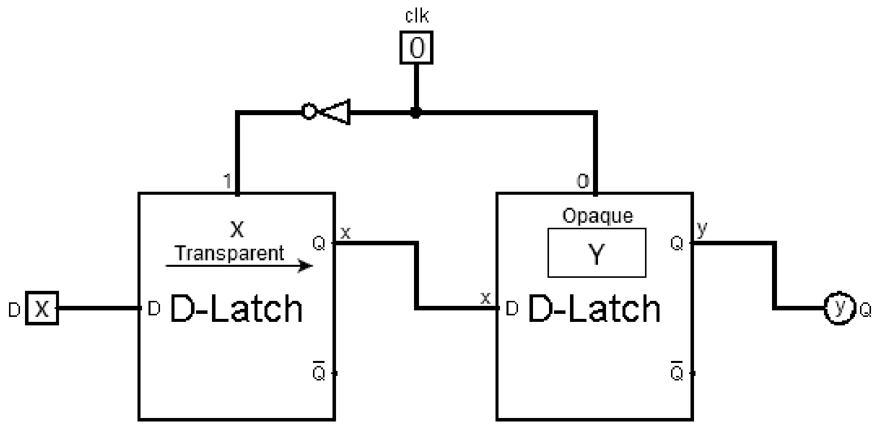
➔ When the clock is at 1 as in the diagram below, the 1st D-Latch (left) is closed and contains Y, and the 2nd is transparent, which passes Y from the output of the 1st D-Latch towards the exit of the 2nd and thus the D-FlipFlop.



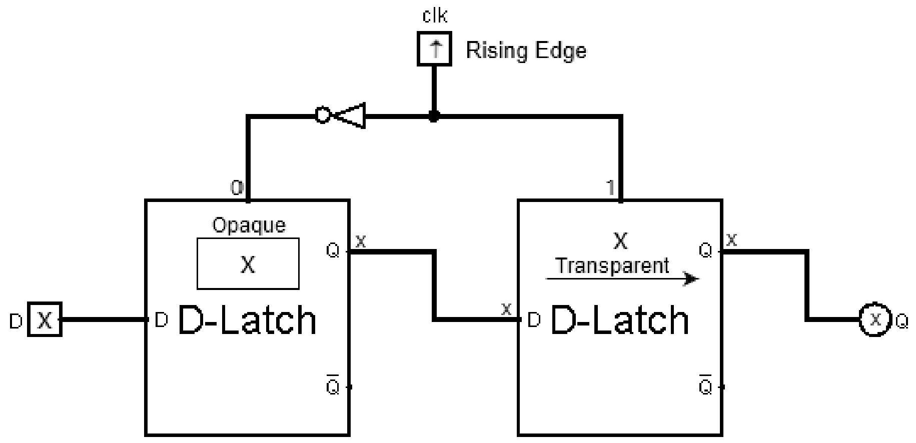
➔ The 2nd step in the clock period is the falling edge. In the diagram with the clock at 0, the 1st D-Latch becomes transparent and passes X from input D. <u>At the same time</u> the 2nd D-Latch closes, since its clock becomes 1, and will memorize the Y value before it transforms into X since the 1st D-Latch is opening. And the D-FlipFlop always holds the Y value according to its output.



➔ The 3rd step in the period is the value 0. The the following diagram shows that when the clock is at 0, the 1st transparent D-Latch passes X from the D input to the input of the 2nd D-Latch. The latter is opaque and memorizes the Y value previously saved during the falling edge of the clock.



➔ Finally, during the rising edge and only at this moment. The output Q of the D-FlipFlop (and also the output of the 2nd D-Latch) will change from Y to X (diagram at the bottom). At this phase, the value X is stored in the 1st D-Latch that becomes opaque, and X passes to the output the 2nd D-Latch which becomes transparent.



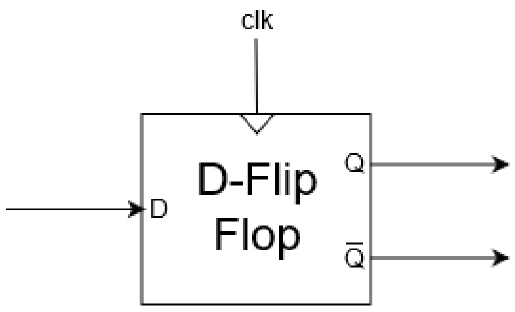
➔ The clock then goes to value 1 of the next period, in which X will no longer change. And those 4 phases will repeat for the next clock periode, and so on.

➔ In conclusion, the only time a D-FlipFlop can change is during the rising edge, the rest of the clock period it memorizes the value inside.

➔ The term FlipFlop refers to the alternation between Opaque and Transparency of the 2 D-Latch.

➔ The Global Scheme of the D-FlipFlop is shown as follows.

➔ Note the triangle in the clock input. It symbolizes the synchronisation with the rising edge on the clock signal.



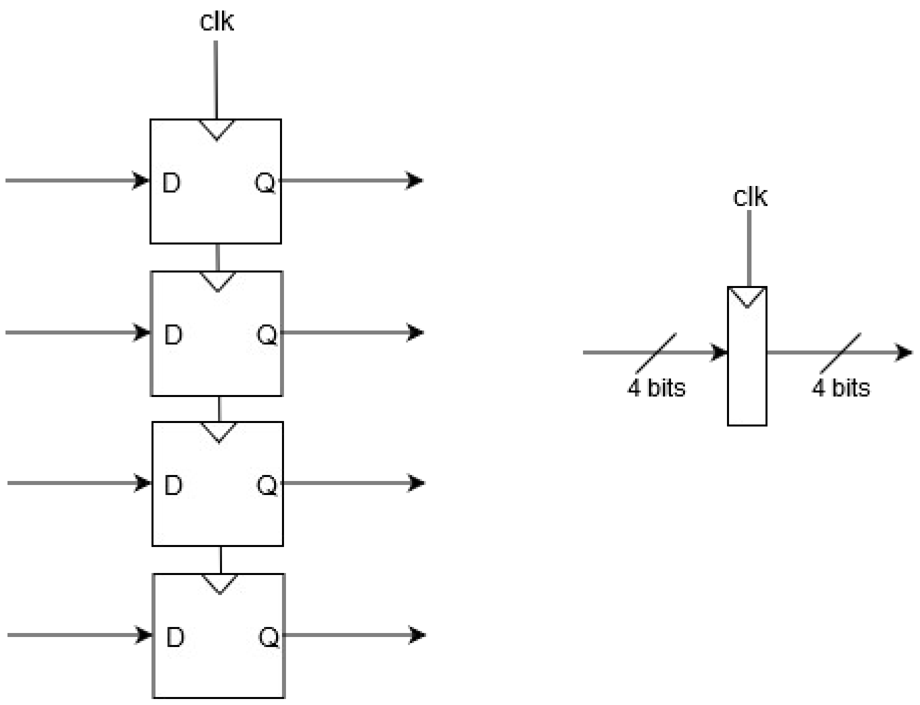
Remark 1: The majority of current integrated circuits (Processors, GPUs, Input/Output Units...etc.) use D-FlipFlop as internal memory. Casually, it is possible to find digital circuits that use the RS-Latch and D-Latch, and this is especially true for older systems. The Bi-Stable is a non-practical theoretical circuit, its purpose is to demonstrate the concept of stability and memorization.

Remark 2: FlipFlop and Latch technology memories should not be confused with memories such as RAM, ROM, or external storage memories; Hard Disk, CD-Rom, Flash Memory...etc, which use different memory technologies.

4.5. Registers

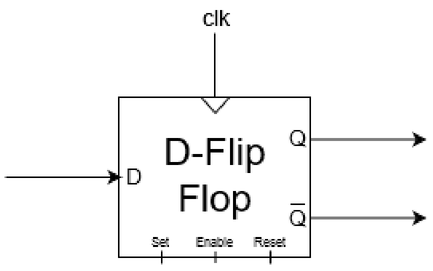
Registers represent a means of storing an information word (or memory word) over multiple bits; 4-bit, 8-bit, 16-bit, 32-bit...etc. They are created like an array of multiple D-FlipFlops synchronized to the same clock signal.

Example : a 4-bit Register is made and represented as follows :



4.6. Enable, Set et Reset

It is possible to find additional command inputs on D-FlipFlops, such as Set, Reset and Enable, as shown in the figure below. They allow the user to have more control over the D-FlipFlop.



- ➔ *Enable* command allows with 1 to activate or with 0 to deactivate a FlipFlop.
- ➔ If FlipFlop is activated it works normally. Otherwise, deactivating it involves freezing and saving the data. This way, making it impossible to modify the value of the cell during the rising edge of the clock.
- ➔ The *Set* and the *Reset* commands behave like RS-Latch, they allow respectively the internal modification of the saved value, to 1 with Set and 0 with Reset.
- ➔ 2 ways are possible to apply the Set and the Reset in FlipFlops; *Synchronous* and *Asynchronous*.
- ➔ A Synchronous Set/Reset involves a modification to 1/0 of the cell only during the rising edge of the clock. While in the Asynchronous, the modification is done instantly, at any moment of the clock period.

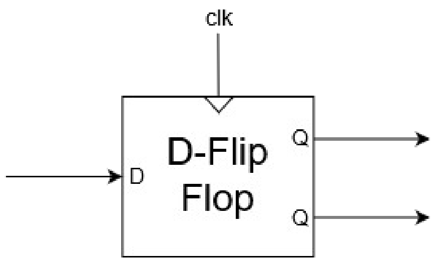
4.7. Other types of FlipFlops

Although the D-FlipFlop remains the best known and most used in digital electronics domain. There are other types of FlipFlops which are much more suitable for some particular situations.

<u>D-FlipFlop :</u>

- ➔ The best known and easiest to use.

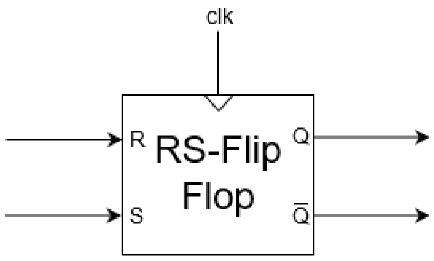
D	▽	Q	$\overline{Q}$
0		0	1
1		1	0



RS-FlipFlop :

- ➔ Follow the same usage as an RS-Latch.
- ➔ Not to be confused with RS-Latch.

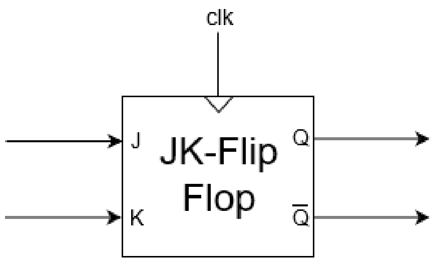
R	S	▽	Q	$\overline{Q}$
0	0		Q`	$\overline{Q}$`
0	1		1	0
1	0		0	1
1	1		-	-



JK-FlipFlop :

- ➔ Similar to RS-FlipFlop in its usage, but in addition, it offers a correction of the forbidden case (11) by reversing the saved content of the cell.
- ➔ The letters JK have no particular meaning, it is just the sequence in the alphabet of the 2 letters JK.
- ➔ But to remember their uses, some electrinics refer to K as *Kill,* which kills the 1 to become 0, and J to *Jump* which jumps from 0 to 1.

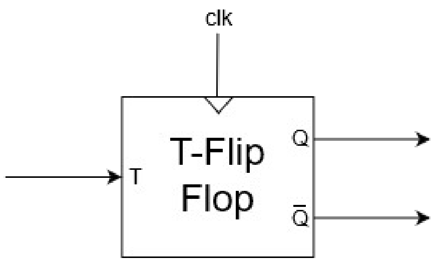
J	K	▽	Q	$\overline{Q}$
0	0		Q`	$\overline{Q}$`
0	1		0	1
1	0		1	0
1	1		$\overline{Q}$`	Q`



T-FlipFlop :

- ➔ T refers to the word *Trigger*.
- ➔ It has a single input, if it is set to 1 it will invert the saved value.
- ➔ To control the initial value of the T-FlipFlip, the *Set* and *Reset* commands are often used.

T	▽	Q	$\overline{Q}$
0		Q`	$\overline{Q}$`
1		$\overline{Q}$`	Q`



Important remark : The truth tables described here are only applicable during the rising edge of the clock. In the rest of the period, the FlipFlop saves its value. Hence the use of the triangle symbol in between the 2 separating lines in those truth tables.

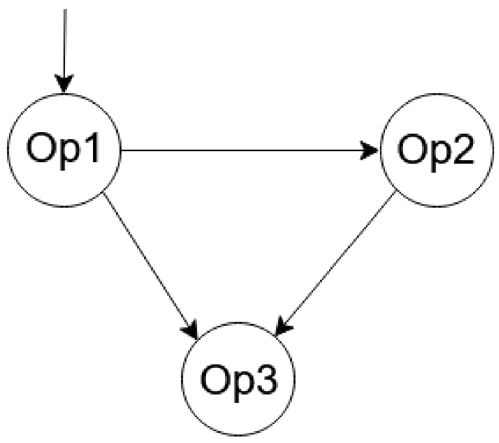
remark : There are a lot of different ways, using logic gates, to implement Latches and FlipFlops.

5. Sequential Circuit specification

The description of how a Sequential Circuit operate is provided formally by what we call mathematically a Finite State Machine (F.S.M.) or Automaton. The word Automaton is close to the term <u>automatic</u> and it is no coincidence. The Automaton makes it possible to model the operation of automatic machines, most often controlled by a Sequential Circuit.

Example :

Let assume a machine that can perform 3 different operations, each operation could be processed by a different Combinational Circuit. Following a process of manufacturing a product by this machine, operation 1 must always be the 1st operation to be performed, operation 2 must follow operation 1, operation 3 could follow operation 1 or operation 2. We can easily model the sequence of those operations by a Finite State Machine as follows:



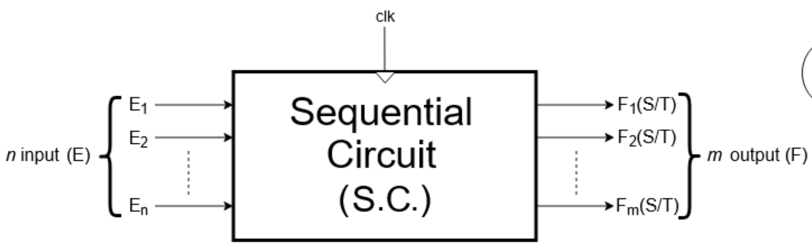
The control of the machine could be manual, and it is up to the human to ensure that the sequencing of operations is correct. but if we want an automatic functionning of the machine, a use of a Sequential Circuit which replacing the 3 Combinatorial Circuits is possible. The Sequential Circuit must have sensor inputs allowing it to know the end of an operation, and user have to choose the desired sequence; op1, op2, op3 or op1, op3.

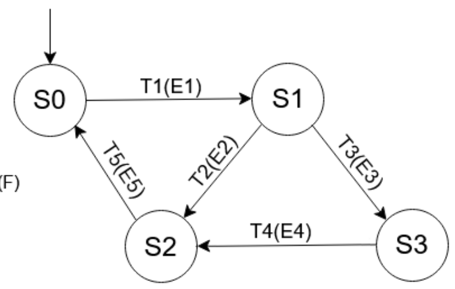
5.1. F.S.M. definition :

An F.S.M. is an abstract mathematical model widely used in the domain of engineering. It makes it possible to describe a system in different states and its transition from one state to another. A definition of F.S.M. must respect the following few points:

- The F.S.M. is a graph where nodes are called States (S) and the links called Transitions (T). As described in the diagram below.
- The State does not represent a component of the system but the entire system in a given State, ex: If a car is a system, its wheel is not a State but the entire car. When the engine is stopped the car is in the *Idle* state.
- Transition represents a condition or event that moves the system from one State to another, ex: Turning the car key is the event that moves the car from the *Idle* state to the *Started* state.
- The F.S.M. works in such a way that the system receives different events, these events trigger the appropriate Transitions and the system goes from one State to another for each event in a repeated perpetual manner.
- The system cannot be in 2 or more States at a given time. Whatever the time, only one State is active at a time upon the entire F.S.M.

The diagram on the left represents the Global Scheme of a Sequential Circuit, and the one on the right, the F.S.M. describing the internal working process of the Sequential Circuit:





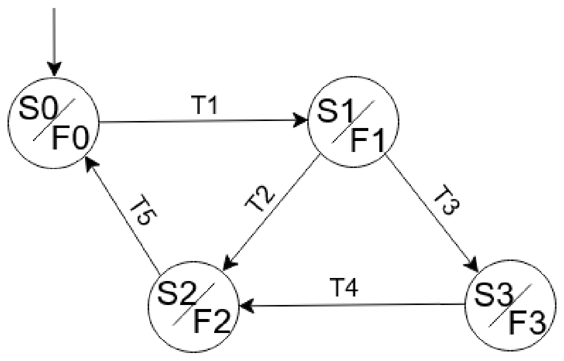
- The Orphan Transition at the top of S0, indicates that S0 is the initial State, at T0 (Time = 0) the system starts from this State.
- Normally F.S.M. has a Final State, in which the Automaton and the System stops, but Sequential Circuits are supposed to never stop.
- The Transition from one State to another is triggered by a value E (on *n* bits) at the circuit input. The values on the circuit inputs are the Transition events or conditions.

Remark : E1 should not confuse E1 with E_1 (on the diagram). The first is a combination of *n* bits on the circuit input, the second is the first bit of this combination.

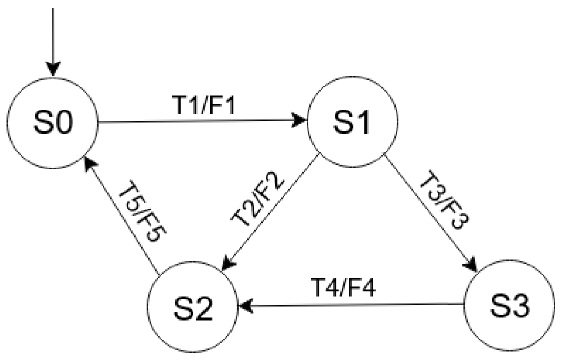
5.2. Sequential Circuit Outputs :

- The output of a Combinational Circuit is a function of its inputs. But the output of a Sequential Circuit can take 2 forms on 2 different machine models, *Moore* and *Mealy*, as described in the following diagram.

The figure represents the output F of a Moore Machine on the left, and the output of a Mealy Machine on the right.



Moore Machine



Mealy Machine

- The output F on Moore Machine is represented by /F inside the State circle of the F.S.M. Each output is a direct function F(S) of the current State of the system.
- However an output F in Mealy Machine is represented by /F on the Transitions arrows of the F.S.M. In this case, the output is a direct function F(T) of the Transition taken by the system.
- In summary, the output in a Moore Machine is related to the current State of the system. And the output in a Mealy Machine is related to the Transition taken by the system while passing from one State to another.

5.3. F.S.M. Transition Table :

Any F.S.M. could be represented by a table called Transition Table, defined by the following few points:

- A Transition Table is a collection of all Transitions making up an F.S.M. Each row in the table represents a Transition T. As shown in the table below.
- Each Transition to be taken in an F.S.M. needs 2 parts of information; Current State and Triggering Event (Input/Condition).
- Thus, the Transition Table mathematically defines Transitions as a function (or application), described by T(S,E) = S` (S` is the next State).
- In other words, the Transition Table has as input the current State and Input E (Event/Condition) of the Transitions, and produces as output the next State.
- The Transition Table allows, among other things, to represent the F.S.M. in a form that the Sequential Circuit can use it electronically for it implementation.

The Automaton Transition Table representing the figure above :

Transitions	Current state	Input	Next State
T1	S0	E1	S1
T2	S1	E2	S2
T3	S1	E3	S3
T4	S3	E4	S2
T5	S2	E5	S0

Remark : A Sequential Circuit being defined before as a circuit which produces its outputs according to the present inputs, plus the history of previous sequence of inputs. In reality, it is the current State that that repesent this sequence, knowing that to arrive at the Current State, it was necessary to go through a determined sequence of inputs. This is why we state that the Current State stores a formatted previously entred sequence.

5.4. F.S.M. execution by a Sequential Circuit :

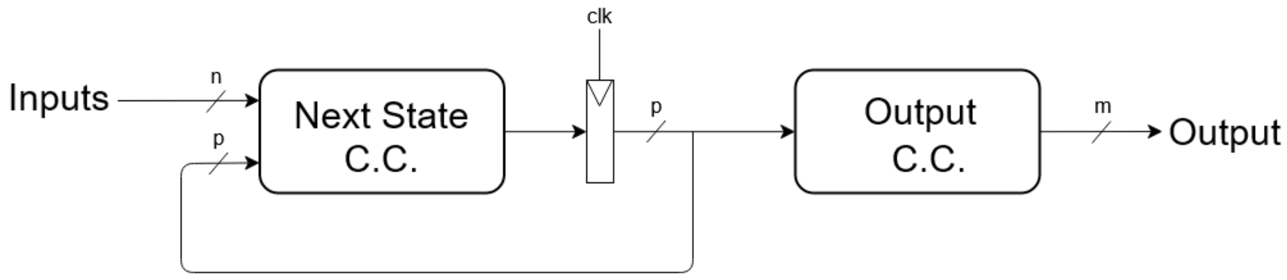
An F.S.M. specifying a Sequential Circuit is executed according to the following few rules:

- The clock is mandatory for a Sequential Circuit, as it has been shown previously by a triangle on the Global Scheme.
- For each clock period, the system remains stable in a given state, and passes to the Next State at the end of the clock period.
- A Transition from one State to another occurs during the rising edge of the clock.
- Thus, the output data of a Moore Machine persists throughout the period. And practically taken at the ending rising edge of the period.
- While in Mealy Machine, the output data of the circuit is taken at the starting rising edge of the current State period.
- Input E for a Sequential Circuit is always read during the rising edge at the beginning of the period.

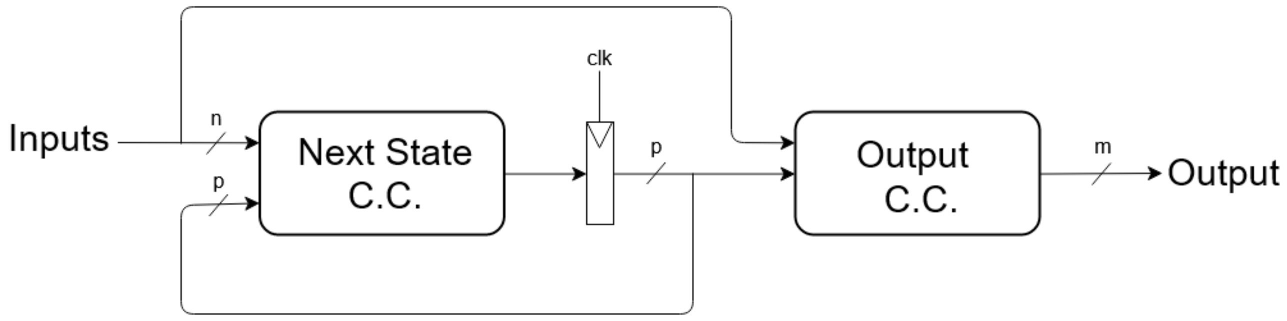
5.5. Schematics logic models to construct an F.S.M. :

To design the Schematics of a Sequential Circuit based on an F.S.M. One needs to follow one of the 2 logic models presnted on the figures below. The first is Moore's Machine model for the Moore F.S.M. and the second Mealy Machine model for the Mealy F.S.M.

The following figure illustrates the Moore Machine model :



The following figure illustrates the Mealy Machine model :



The Machine is chosen according to the appropriate F.S.M. and the process of designing a Sequential Circuit will be limited to the search for the following 3 unknowns represented on the diagram :

- The value of *p* (the bit size of the Registre).
- The Next State Combinational Circuit.
- The Output Combinational Circuit.

Solving these 3 unknowns imply the drawing of the Sequential Circuit Schematics. To do so, a method called 7-step method is used.

6. Conventional method to design a Sequential Circuits

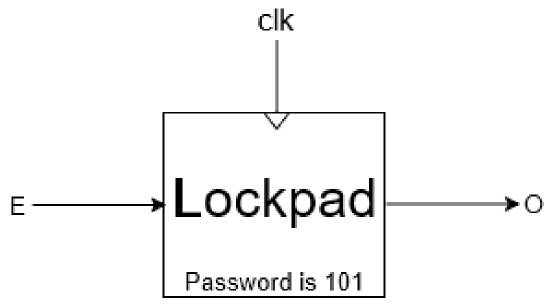
The process of constructing a Sequential Circuits is based on the 7-step Method, and it involves the application of the following consecutive 7 steps :

- Global Scheme
- F.S.M.
- Transition Table
- Encoded States Table and Outputs Table
- Encoded Transition Table
- Logic Formulas
- Schematics

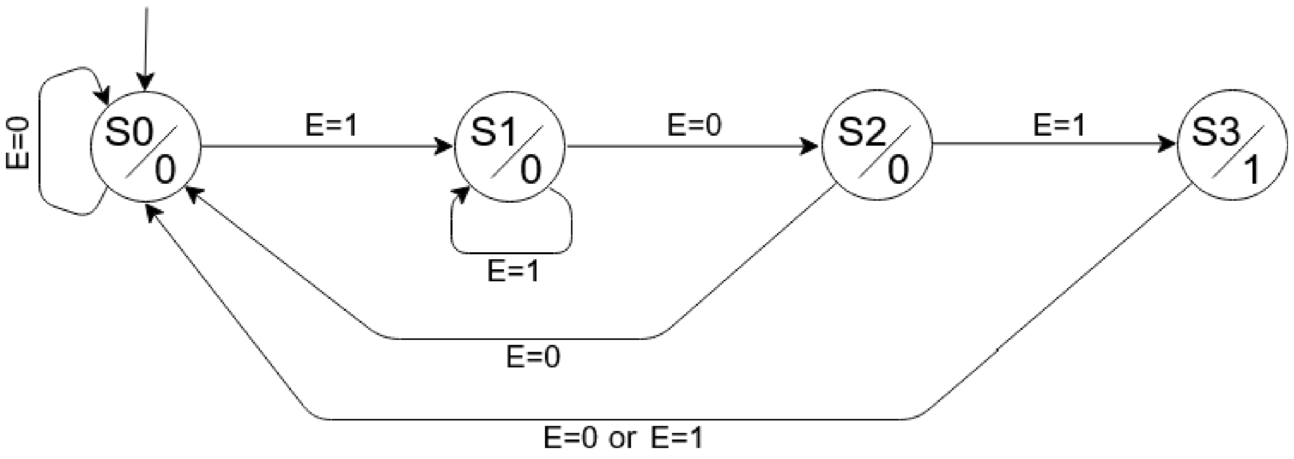
Example 1: (Moore Machine)

The example is of the electronic Padlock seen previously in this chapter.

<u>Step 1</u> : Global Scheme



<u>Step 2</u> : F.S.M.



Remark 1: The F.S.M. is practically the most delicate and important step among the 7 steps of building a Sequential Circuit.

Remark 2: The design of an F.S.M. usually imply giving a meaning for each State. For instance, S0 in our F.S.M. means that the circuit has not recognized any digit of the correct code at this stage, while S1 means that the circuit recognizes the first digit 1 which is correct, S2 that sequence 10 entered so far is correct, and S3 that all code 101 is correct.

Remark 3: The events represent the value entered in E (1 or 0). But the most optimal way, when the circuit contains several inputs, is to express the Transition condition by a logical formula of the variables. **ex** : a Transition with the condition $\overline{E}_2 \cdot E_1 + \overline{E}_0$, can only be taken when ($E_2=0$ and $E_1=1$) or $E_0=0$.

Remark 4: The F.S.M.'s solution also remains correct when modifying Transition S3 with E=1 going to State S1 instead of S0. This means that the 1 just after the recognition of the code is taken as the first 1 recognized for the next password.

Step 3 : Transition Table

Current State	Input (E)	Next State
S0	0	S0
S0	1	S1
S1	0	S2
S1	1	S1
S2	0	S0
S2	1	S3
S3	0	S0
S3	1	S0

Step 4 : Encoded States Table and Outputs Table

States	S_1	S_0	O
S0	0	0	0
S1	0	1	0
S2	1	0	0
S3	1	1	1

To encode 4 States we need 2 bits ; S_1 et S_0.

Remark 1: Remember that to get the number of bits necessary to encode N values in binary, you will need to use $Log_2(N)$ formula, with a ceil rounding.

Remark 2: Be careful not to confuse qualifiers like S1 and S_1, the first designates State S1 in the F.S.M. the second designates the second bit in the encoding of the States.

Remark 3: Finding the number of bits to encode the States, is considered the resolution of the first unknown *p* (*p*=2), so the size of the Register for this Machine is 2 bits.

Remark 4: The Register in the Machines (Moore or Mealy) has the role of memorizing in binary the State in which the F.S.M. is settled. and to provide this information to the 2 Combinational Circuits (Next State C.C. and Output C.C.).

Remark 5: The Output Table has the 2 columns S_1 and S_0 as inputs, and column O as output. Indeed it represents the Truth Table of the Output Combinational Circuit for the Moore Machine.

Remark 6: Finding the Output Table correspond to the finding the second unknown, which is the Output circuit. It is a Combinational Circuit that has as input the current State of the F.S.M. and produces the corresponding output for that State.

Step 5 : Encoded Transition Table

S_1	S_0	Input (E)	$S`_1$	$S`_0$
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

It is the same Transition Table but with the encoding of the States

Remark 1: $S`_1$ and $S`_0$ represent the encoding bits of the next State.

Remark 2: La Table de Transition Encodée est une Table de Vérité avec comme entrées S_1 ,S_0 et E, et comme sortie $S`_1$ et $S`_0$. Elle représente réellement le troisième inconnu de la Machine de Moore, qui est le Circuit Combinatoire de l'État suivant. Et réellement c'est la table qui représente l'Automate du Circuit Séquentiel.

The Encoded Transition Table is a Truth Table with S1, S0 and E as inputs, and S`1 and S`0 as outputs. It actually represents the third unknown of the Moore Machine, which is the next State Combinatorial Circuit. And represents effectively, the figuration of the Sequential Circuit F.SM.

Remark 3: The Next State circuit according to the Moore Machine diagram, lies at the heart of the Sequential Circuit loop. It takes 2 types of information as input; the Current State of the circuit from the Register, and the event (Transition condition) from input E. And produces the Transition jumping to the next State. Which will be saved in the Register on the next rising edge of the clock.

Remark 4: The Sequential Circuit passes from one State to another during each period or rising edge of the clock. it is the Register that memorizes the code of the Current State in which the circuit is settled. The Next State circuit is responsible for transitioning to the next state by combining input E and the current state. The Sequential Circuit produces its output depending on the current state of the circuit, it is of the Output circuit responsiblity to perform this link function between the Current State and the Output.

Remark 5: For a Synchronous Sequential Circuit it is the clock which sets the rhythm of the looping rate on the Moore Machine or Mealy Machine, as well as the rate of passing from one State to another on the F.S.M.

Step 6 : Logical Formulas

The Logic Formulas are the reduced functions of the study of the 2 Combinatorial Circuits, Output Circuit and Next State Circuit.

Visually, we can directly deduce the formula from the Output T.T. of : **$O(S_1,S_0) = S_1 \cdot S_0$**

And performe the Karnaugh map reduction to Next State T.T. :

E \ $S_1 S_0$	00	01	11	10
0	0	$\boxed{1}$	0	0
1	0	0	0	$\boxed{1}$

$S`_1(S_1,S_0,E) = \overline{S}_1 \cdot S_0 \cdot \overline{E} + S_1 \cdot \overline{S}_0 \cdot E$

E \ $S_1 S_0$	00	01	11	10
0	0	0	0	0
1	1	1	0	1

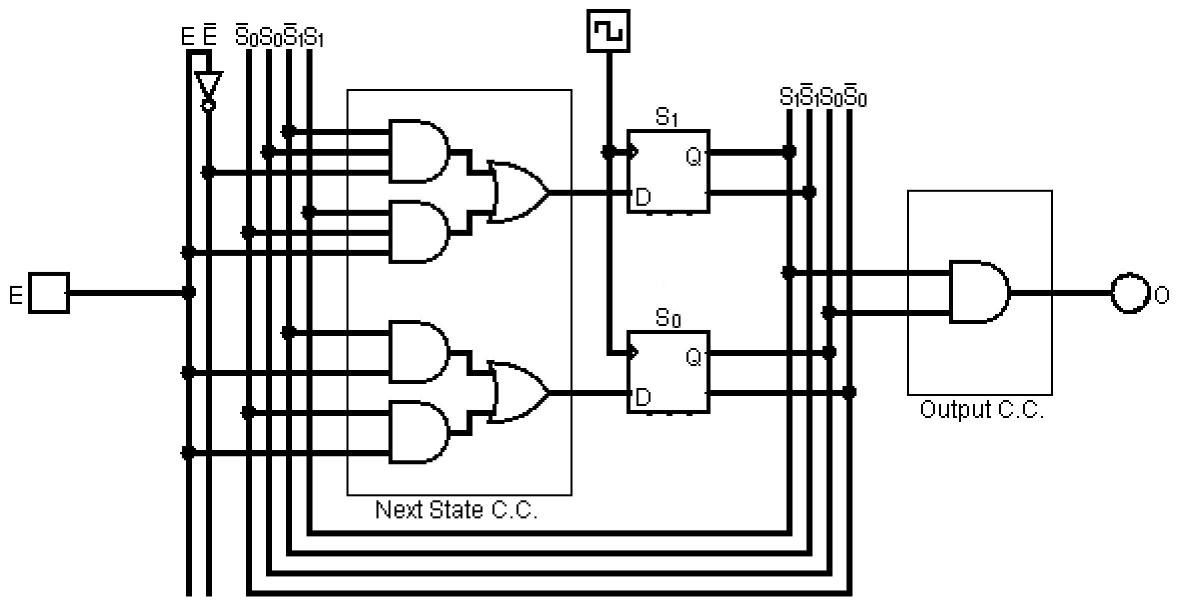
$S`_0(S_1,S_0,E) = \overline{S}_1 \cdot E + \overline{S}_0 \cdot E$

Remark 1: This step is equivalent to steps 3 and 4 of the 5-step method for the construction of a Combinatorial Circuit. We can neglect the Canonical Forms step and concentrate on the reduction step.

Remark 2: The function of O(S1,S0) was deduced directly due to its inherent simplicity, which does not require the development of a Karnaugh map.

Remark 3: Like the 5-step method for the design of a Combinational Circuit, it is possible in some situations to choose algebraic reduction instead of Karnaugh map. And even using *don't care* for the impossible outputs.

Step 7 : Schematics



$E\ \overline{E}\ \overline{S}_0 S_0 \overline{S}_1 S_1$
S_1
Q
D
S_0
Q
D
$S_1 \overline{S}_1 S_0 \overline{S}_0$
E
O
Output C.C.
Next State C.C.

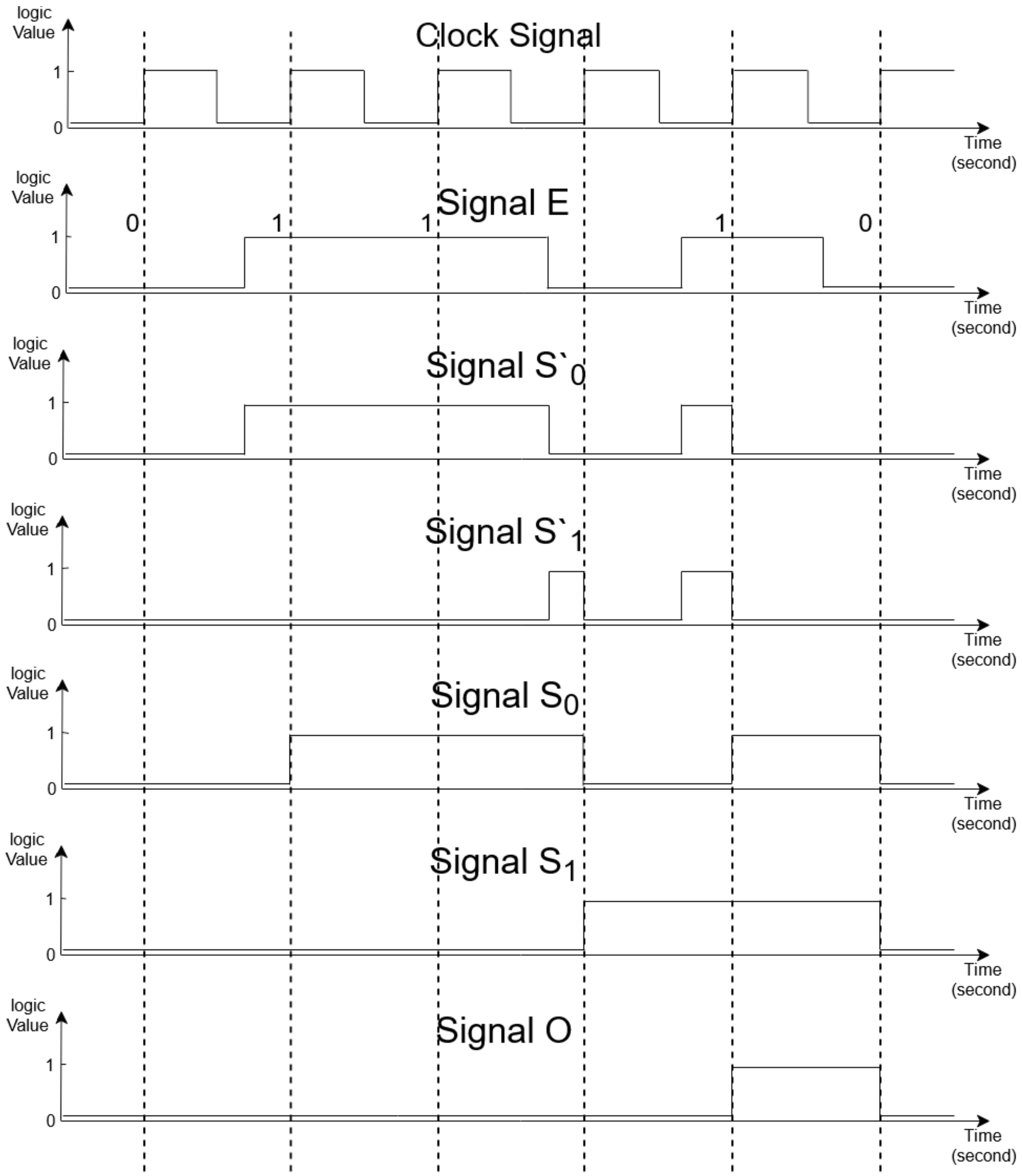
Remark 1: The construction of the Schematics is modeled on the diagram of Moore's Machine. All the elements of the Schematics are arranged in the same location as in the Machine diagram.

Remark 2: When executing the circuit, it is possible for the user at any time to know which State the F.S.M. is located, simply by reading the State code in the Register with the 2 cells S_1 and S_0.

Remark 3: The main characteristics of a Sequential Circuit are directly visible on the Schematics; the loop, the memory and the clock. The dynamism is observable during an execution, an execution is performed at the bottom.

Remark 4: The Register memory does not directly save the history of the previous entered sequence. But it does so indirectly through the current State code, knowing that the evolution towards a certain State in an F.S.M. requires some form of entered sequence.

Execution : To fully understand the internal functioning of the Sequential Circuit, we will perform an execution with a scenario of simple inputs with the sequence 011010 (going from left to right). By tracking on the timing diagram, the evolution per clock period of the signals $S`_0$, $S`_1$ (the outputs of the Next State Combinational Circuit), S_0, S_1 (the outputs of the Register) and O (the output of the Output Combinational Circuit, and at the same time the entire Sequential Circuit). We must follow the propagation of the signal through the logic gates in the 2 Combinational Circuits. We must not forget that during the Initial State the values in the Register are $S_0 = 0$ and $S_1 = 0$.



Important remark 1: The input signals (E in our case), must always arrive at the input of the Sequential Circuit before the rising edge of the clock. This gives enough time for its propagation inside the Combinational Circuits and would conform the *setup-time* timing of the Flipflops.

Important remark 2: The Sequential Circuit remains stable in a given State throughout all the period, and transits to another State during the rising edge of the clock. Which implies that the output of the circuit passing through the Output Combinational Circuit also remains stable throughout all the period. The output is collected by the next circuit at the ending rising edge of the period.

Remark 1: FlipFlops are characterized by what is called *setup-time*, this is a minimum time in which the signal in D must remain stable before the arrival of the rising edge.

Remark 2: There are concrete formulas for the temporal calculation of the signal in combinational and sequential circuits. This belongs to the the temporal study domain of logic circuits (it is not approached this course).

One-hot Encoding : Is a form of state encoding which specify that the value *p* is equal to the number of states. Where each state is represented by a unique bit in the Register. At any time there is only only one active bit (set to 1) in the entire Register, the one representing the current State, the other bits are 0.

Example 2: (One-hot Encoding)

We will repeat the previous example 1 of the electric lockpad with One-hot encoding. The first 3 steps are exactly the same, no need to repeat them, we continue from step 4.

Step 4 : Encoded One-hot States Table and Outputs Table

States	S_0	S_1	S_2	S_3	O
S0	1	0	0	0	0
S1	0	1	0	0	0
S2	0	0	1	0	0
S3	0	0	0	1	1

To encode 4 States in One-hot format we need 4 bits.

Remark 1: The Output Table remains practically the same, except for the encoding of the States which it become different.

Remark 2: Actually the Table is made up of 16 input lines knowing that it has 4 input variables (2^4 = 16 possibilities). States encoded with more than a single 1 (such as 0111 or 0011) or no 1 (such as 0000) are assumed to be impossible case states in One-hot encoding. They are not represented on the table and assumed to be *don't care*.

<u>Step 5</u> : One-hot Encoded Transition Table

S_0	S_1	S_2	S_3	E	$S`_0$	$S`_1$	$S`_2$	$S`_3$
1	0	0	0	0	1	0	0	0
1	0	0	0	1	0	1	0	0
0	1	0	0	0	0	0	1	0
0	1	0	0	1	0	1	0	0
0	0	1	0	0	1	0	0	0
0	0	1	0	1	0	0	0	1
0	0	0	1	0	1	0	0	0
0	0	0	1	1	1	0	0	0

One-hot Encoded Transition Table

Remark : The table has 5 variables as inputs, so we expect 32 lines. Most of them have don't care outputs, since they do not respect One-hot encoding.

<u>Step 6</u> : Logical Formulas

Visually we can deduce from the Output T.T. that : $\mathbf{O(S_0,S_1,S_2,S_3) = S_3}$

We use algebraic minimization on the One-hot Encoded Transition Table :

$S`_0(S_0,S_1,S_2,S_3,E) = S_0\cdot\overline{S_1}\cdot\overline{S_2}\cdot\overline{S_3}\cdot\overline{E} + \overline{S_0}\cdot\overline{S_1}\cdot S_2\cdot\overline{S_3}\cdot\overline{E} + \overline{S_0}\cdot\overline{S_1}\cdot\overline{S_2}\cdot S_3\cdot\overline{E} + \overline{S_0}\cdot\overline{S_1}\cdot\overline{S_2}\cdot S_3\cdot E$
$S`_0(S_0,S_1,S_2,S_3,E) = (S_0\cdot\overline{S_1}\cdot\overline{S_2}\cdot\overline{S_3}\cdot\overline{E} + \overline{S_0}\cdot\overline{S_1}\cdot S_2\cdot\overline{S_3}\cdot\overline{E}) + (\overline{S_0}\cdot\overline{S_1}\cdot\overline{S_2}\cdot S_3\cdot\overline{E} + \overline{S_0}\cdot\overline{S_1}\cdot\overline{S_2}\cdot S_3\cdot E)$
$S`_0(S_0,S_1,S_2,S_3,E) = \overline{S_1}\cdot\overline{S_3}\cdot\overline{E}\cdot(S_0\cdot\overline{S_2} + \overline{S_0}\cdot S_2) + \overline{S_0}\cdot\overline{S_1}\cdot\overline{S_2}\cdot S_3$
$\mathbf{S`_0(S_0,S_1,S_2,S_3,E) = \overline{S_1}\cdot\overline{S_3}\cdot\overline{E}\cdot(S_0\oplus S_2) + \overline{S_0}\cdot\overline{S_1}\cdot\overline{S_2}\cdot S_3}$

$S`_1(S_0,S_1,S_2,S_3,E) = S_0\cdot\overline{S_1}\cdot\overline{S_2}\cdot\overline{S_3}\cdot E + \overline{S_0}\cdot S_1\cdot\overline{S_2}\cdot\overline{S_3}\cdot E$
$\mathbf{S`_1(S_0,S_1,S_2,S_3,E) = (S_0\oplus S_1)\cdot\overline{S_2}\cdot\overline{S_3}\cdot E}$

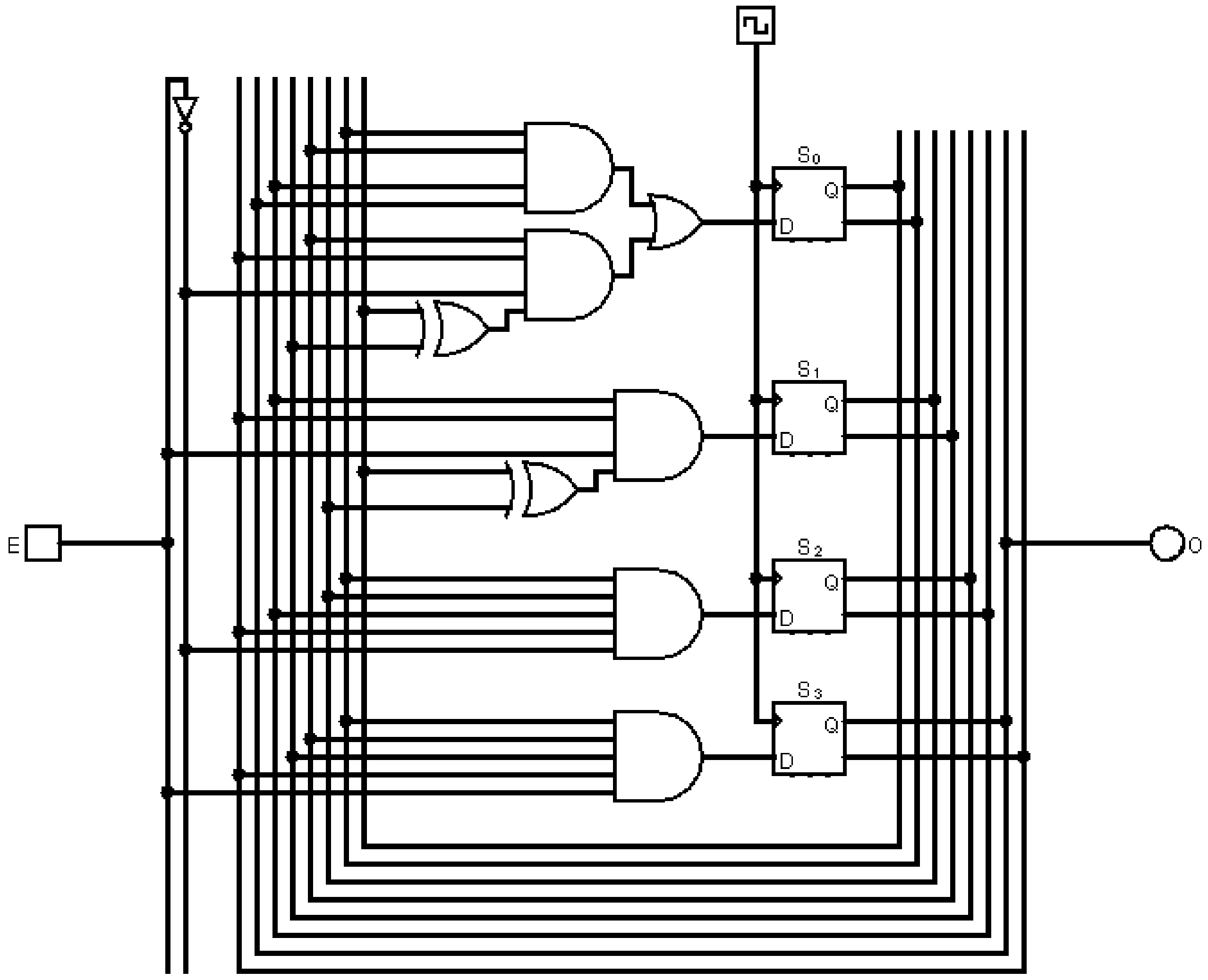
$\mathbf{S`_2(S_0,S_1,S_2,S_3,E) = \overline{S_0}\cdot S_1\cdot\overline{S_2}\cdot\overline{S_3}\cdot\overline{E}}$

$\mathbf{S`_3(S_0,S_1,S_2,S_3,E) = \overline{S_0}\cdot\overline{S_1}\cdot S_2\cdot\overline{S_3}\cdot E}$

Remark 1: We were forced to choose to use Algebraic reduction instead of Karnaugh map. Because of the minimization with a Karnaugh map on 5 variables is significantly different and more difficult (in addition we didn't study that in class).

Remark 2: The Karnaugh map remains applicable for a number of variables no greater than 6, although it remains more delicate for 5 and 6 variables. The Quine–McCluskey method (not seen in class) is more suitable for this type of situation.

<u>Step 7</u> : Schematics



Remark 1: Although this is not the case for this example, as a general rule One-hot encoding has the advantage, compared to normal encoding, of minimizing the number of logic gates in the 2 Combinational Circuits. in the other hand, it increases the number of memory cells.

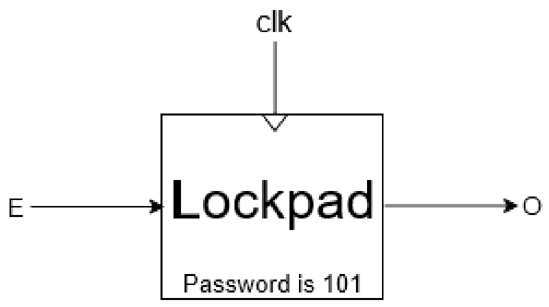
Remark 2: Il existe aussi une autre variante semblable au One-hot, appelée One-Cold (*1-bit-à-0* en français). C'est le complément de One-hot, ça veut dire le bit représentant l'État est à 0 et les autres à 1, **ex** : l'État S0 est encodé 0111.

There is also another variation similar to One-hot, called *One-Cold*. It is the complement of One-hot. That means the bit representing the State is at 0 and the others at 1, **ex** : State S0 is encoded 0111.

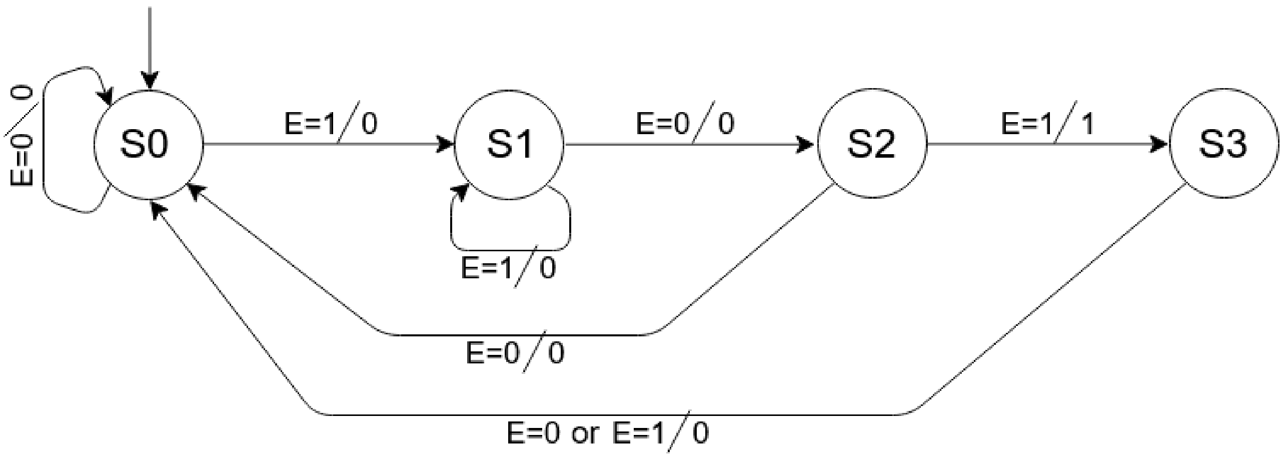
Exemple 3: (la Machine de Mealy)

We will repeat the same example of the padlock, this time with the Mealy Machine.

<u>Step 1</u> : Global Scheme



<u>Step 2</u> : F.S.M.



Remark : There is no difference between the 2 machines of Moore and Mealy, the only difference is that the output in Moore is in *States*, while in Mealy it is in *Transitions*.

<u>Step 3</u> : Transition Table

Current State	Input (E)	Next State
S0	0	S0
S0	1	S1
S1	0	S2
S1	1	S1
S2	0	S0
S2	1	S3
S3	0	S0
S3	1	S0

Remark : The Transition Table is exactly the same as that of the Moore Machine.

<u>Step 4</u> : Encoded States Table and Outputs Table

States	S_1	S_0	E	O
S0	0	0	0	0
S0	0	0	1	0
S1	0	1	0	0
S1	0	1	1	0
S2	1	0	0	0
S2	1	0	1	1
S3	1	1	0	0
S3	1	1	1	0

Encoded States and Outputs Table for Mealy Machine

Remark : By observing the difference between Moore's and Mealy's Machine on the diagram of the 2 machines (page 19), the only difference between the 2 is in their Output circuits. The Moore machine only needs the current State coming from the Register to produce the output, while the Mealy Machine needs to emit its output during the Transition of the 2 pieces of information; input E and current State. On the basis of these 2 pieces of information, the Mealy Output combinational circuit can recognize the Transition and thus the corresponding output at the rising edge of the clock.

<u>Step 5</u> : Encoded Transition Table

S_1	S_0	Input (E)	$S`_1$	$S`_0$
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

Remark : The Encoded Transition Table is exactly the same as that of the Moore Machine.

Step 6 : Logical Formulas

The Next State Combinational Circuit produces the same formulas as of the Moore Machine, knowing that the Encoded Transition Table has not changed.

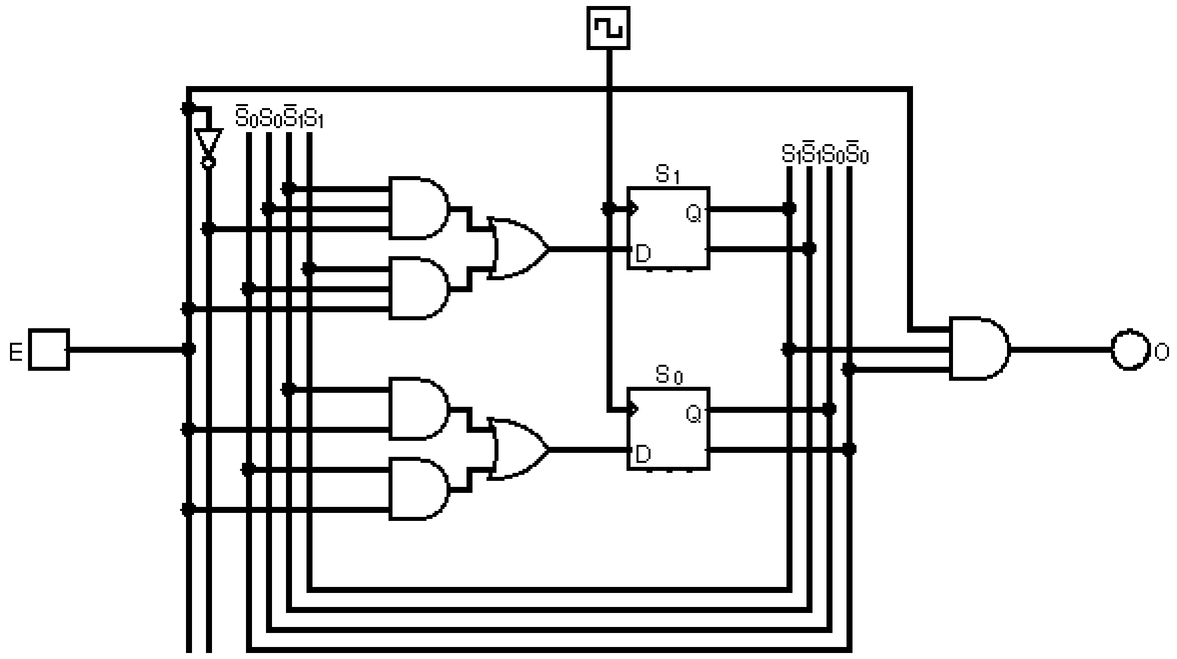
$S`_0(S_1,S_0,E) = \overline{S_1}\cdot E + \overline{S_0}\cdot E$

$S`_1(S_1,S_0,E) = \overline{S_1}\cdot S_0\cdot \overline{E} + S_1\cdot \overline{S_0}\cdot E$

The Output Table is so simple that the formula can be extracted directly :

$O(S_1,S_0,E) = S_1\cdot \overline{S_0}\cdot E$

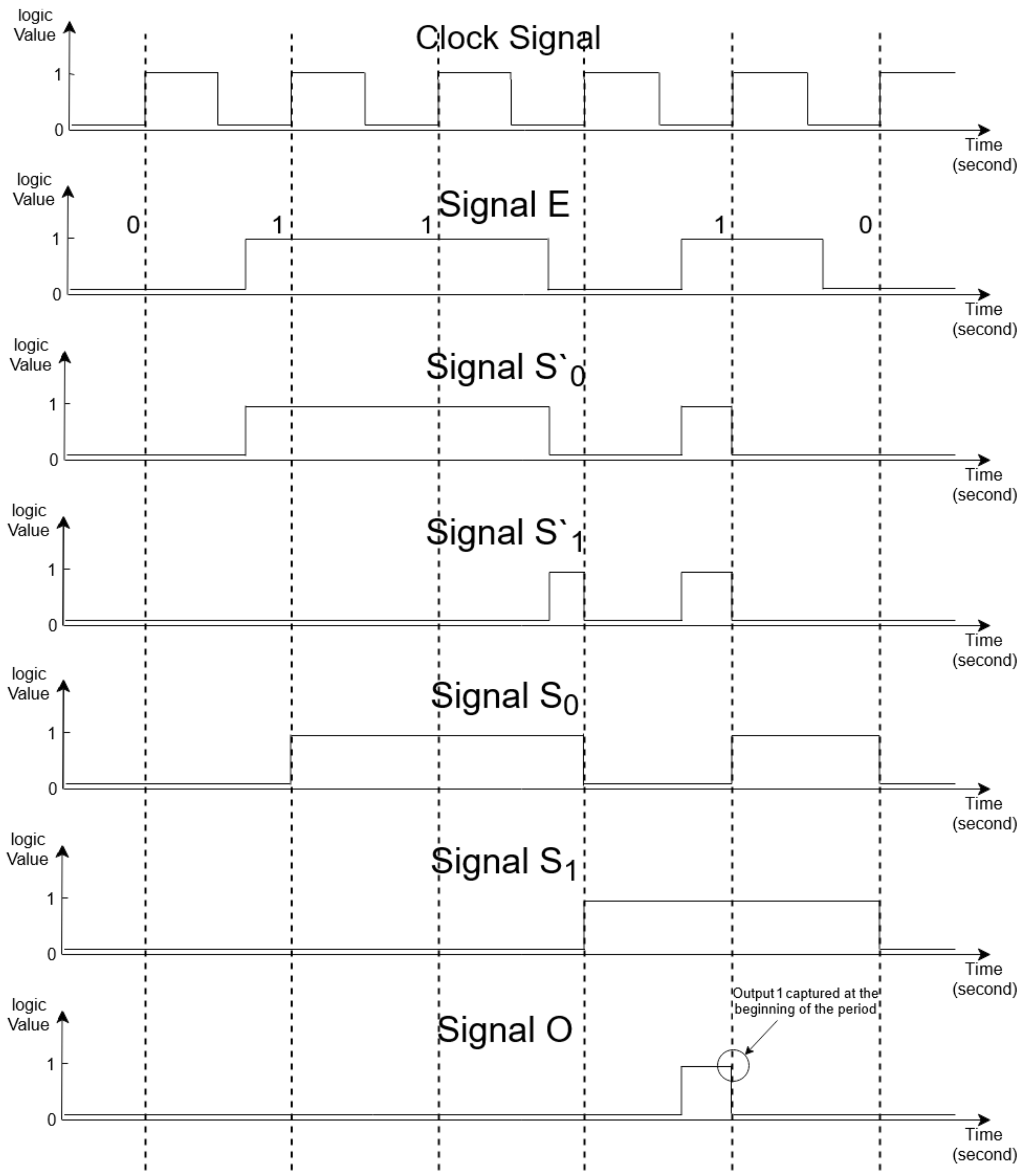
Step 7 : Logigramme



Remark 1: Encoding with One-hot or One-cold is also possible on the Mealy Machine.

Remark 2: To design of a Sequential Circuit, the choice between the Moore Machine and the Mealy Machine, with each machine 3 forms of encoding; normal, One-hot and One-cold, is generally oriented by constraints such as: the number of gates available, the number of memory cells available, the speed or frequency desired for the circuit, the maximum power consumption...etc. . All these constraints generally influence the design decisions of a digital electronic circuit.

Execution : We repeat an execution identical to example 1 with a scenario of inputs in sequence 011010. the execution is shown on the timing diagram at the bottom.



Remark 1: The main difference in execution between the Moore Machine and the Mealy Machine is that the output value 1 is captured on the rising edge ending the S3 State period in Moore Machine. While the output 1 is captured during the rising edge at the beginning of the State S3 period in Mealy Machine.

Remark 2: It is always possible to capture the output in Mealy Machine for the duration of a period, by simply adding a register to the output of the machine.

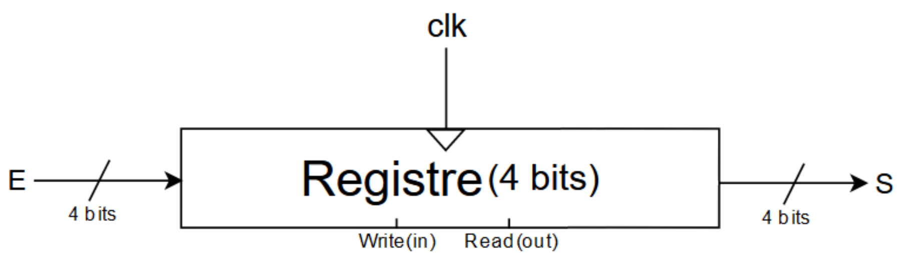
7. The most known Sequential Circuits

As with Combinational Circuits there are some Sequential Circuits which are very well known and widely used.

7.1. Registers

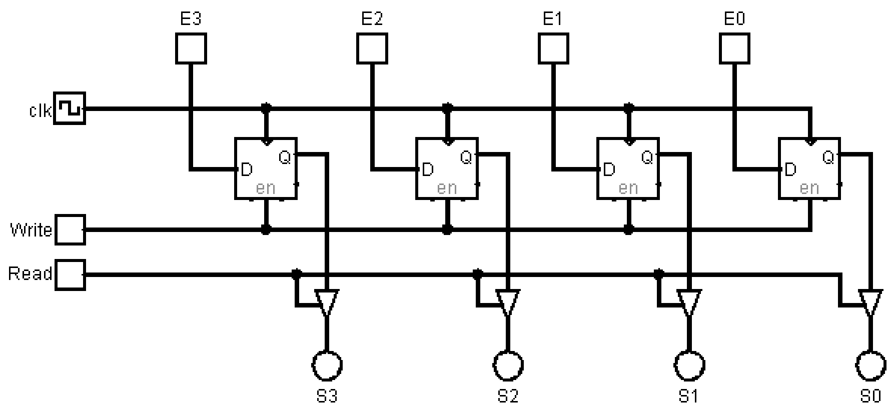
- ➔ We have already seen the simplest form of a Register, it is a D-FlipFlop array with a common clock.
- ➔ The Register may have more features, such as *reading* and *writing*.
- ➔ Reading and Writing are implemented by 2 control inputs as in the figure below.

The figure shows a 4-bit Register with additional Read and Write control inputs.



- ➔ The Write (or *in*) if set to 1 will allow the value in E to enter during the Rising Edge to overwrite the old value in the Register. On the other hand if it is set to 0, the value will not enter and the saved value remains unchanged.
- ➔ The Read (*out*) command gives the Register the option of not outputting the value saved inside. If Read is set to 1, the value in the Register outputs in S. On the other hand if it is set to 0,the floating value Z will come out on the output S.

Une implémentation basique de la lecture et de l'écriture est détaillée sur la figure suivante

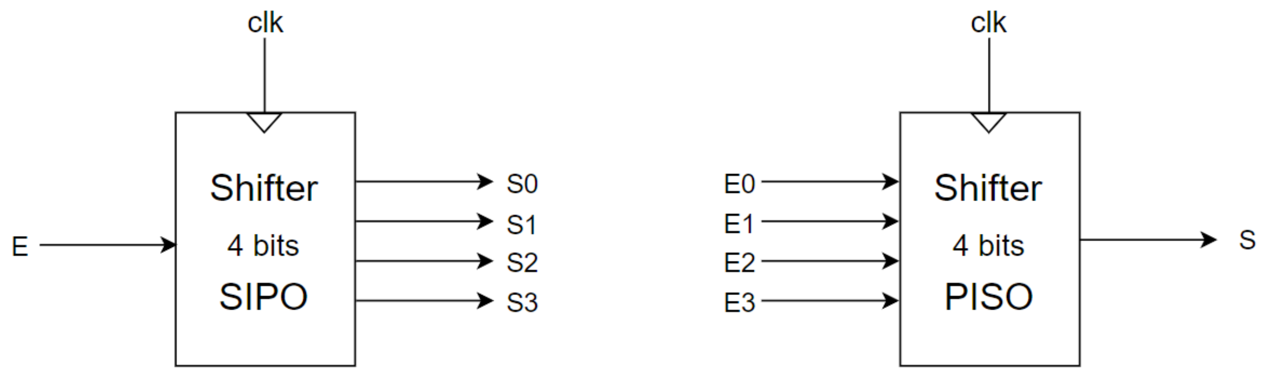


- ➔ The Write in the figure is implemented by the Enable input of the D-FlipFlops. So if it is put to 0 the values of E will not enter during the Rising Edge.
- ➔ Read is implemented using the Tristate buffer gate between the Q outputs of the D-FlipFlop and the S output, so if the Tristate buffer control is set to 0 its output produces Z, which is interpreted as a neutral signal.

7.2. Shifters

- ➔ As there is a Combinational Shifter there is also a Sequential Shifter.
- ➔ The difference between a Combinational Shifter and a Sequential Shifter is that in the Combinational we must specify the number of bits to be shifted. Whereas in Sequential, one bit is automatically shifted for each Rising Edge of the Clock.
- ➔ Two types of Sequential Shifters exist, Serial-In Parallel-Out (or SIPO) and Parallel-In Serial-Out (or PISO).

The following figure illustrates the 2 types of Shifters, on the left the 4-bit Serial- Input Parallel-Output Sequential Shifter (SIPO). And on the right, the 4-bit Parallel-Input Serial-Output Sequential Shifter (PISO).



- ➔ A SIPO Shifter works in such a way that at each Rising Edge the input E enters a new value, the values entered in a chain one after the other will output in the same order in parallel on the outputs S0 until S4.
- ➔ For instance, if at the 1st Rising Edge we entered the value 1, it would come out in S0. And if in the 2nd Rising Edge we entered the value 0, the output would be 1 in S1 and 0 in S0. And the same for the 3rd Rising Edge, if we enter the value 1 it is the value 101 that will come out respectively in S2 S1 S0, and so on.
- ➔ That means that the new bit entered will shift the other bits by one position to create its own place in the first position, and the same for the bit that follows it...etc.
- ➔ In a PISO Shifter it is the opposite, we enter the 4 values at the same time in parallel. And those will then output successively one after the other in the output E for each Rising Edge of the Clock.

- ➔ There is an essential command in the PISO Shifter which is not shown in the previous diagram.It is the command that chooses between the operation of reading the 4 values in parallel at the Rising Edge, or the operation to release them one after the other in series for each Rising Edge.
- ➔ Both PISO and SIPO Shifters have the ability to adhere to the Slicing mechanism, making them very flexible for creating Shifters of varying sizes.

Remark 1: For Sequential Shifters, knowing the orientation of the left or right shift is not as important as for Combinational Shifters, since it is a bit-by-bit transmission used mainly for Serial/Parallel transmission.

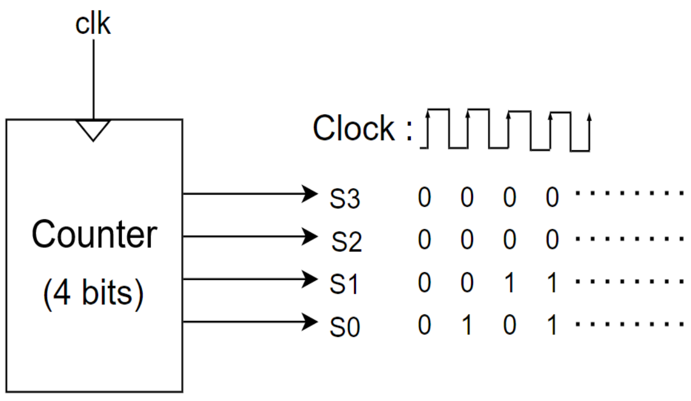
Remark 2: Sequential Shifters have an important role in serving as a Serial/Parallel or Parallel/Serial conversion interface.

Remark 3: SIPO Shifters are also widely used by processors to extend their number of outputs. For example, for a calculator with a display of 10 7-segment displays, normally its processor would need 10 x 7 = 70 output ports, which is enormous. In reality, the processor can use a single output, on which it will output the 70 segment values in series, then supplied in parallel using a SIPO for the 10 7-segment displays.

7.3. Counters

- ➔ The Counter is a Sequential Circuit that can count in binary.
- ➔ It is a circuit that has no inputs but has n outputs to represent an n-bit unsigned binary value.
- ➔ For each Rising Edge it will count, starting with the value 0, then 1, then 2...until (2^n-1), then returning to the initial value 0 and starting again.

In the figure below we can see how the 4-bit Counter can count in binary by synchronizing with each Rising Edge of the clock signal.



- ➔ We can see on the diagram that on the 1st Rising Edge the Counter outputs the value 0 (0000), the 2nd the value 1 (0001), the 3rd the value 2 (0010)...etc. .
- ➔ The Counter can have a control input called Reset, if set to 1 it will reset the count to 0, even if the Counter has not yet reached its last value (2^n-1).
- ➔ The Counters also adhere to the slicing composition.

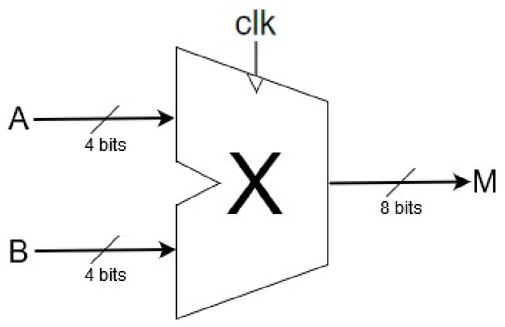
Remark 1: It is not uncommon to find Counters with an n-bit input, this allows them to force an initial value other than 0 if the need arises.

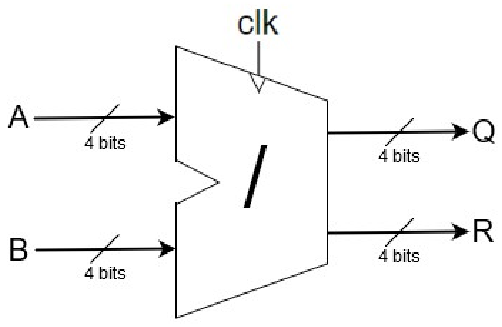
Remark 2: The counter is a very famous circuit. It is an essential component for example in digital watches, stopwatches, timers...etc.

7.4. Multipliers and Dividers

- ➔ Sequential Multipliers and Dividers are identical to Combinatorial Multipliers and Dividers, the only difference is that the latter also have a clock input.
- ➔ Because the calculation in these circuits is not direct, it needs to consume a little time before finishing.

A 4-bit Multiplier is shown left the figure, and a 4-bit Divider is shown right figure.





- ➔ The multiplication operation needs to wait *m* clock periods to complete, and the division also needs *n* clock periods.
- ➔ In reality, the Multiplier and Divider calculate a single bit for each Rising Edge of the Clock.
- ➔ Remeber that combinatorial multiplication and division perform their calculations instantly.

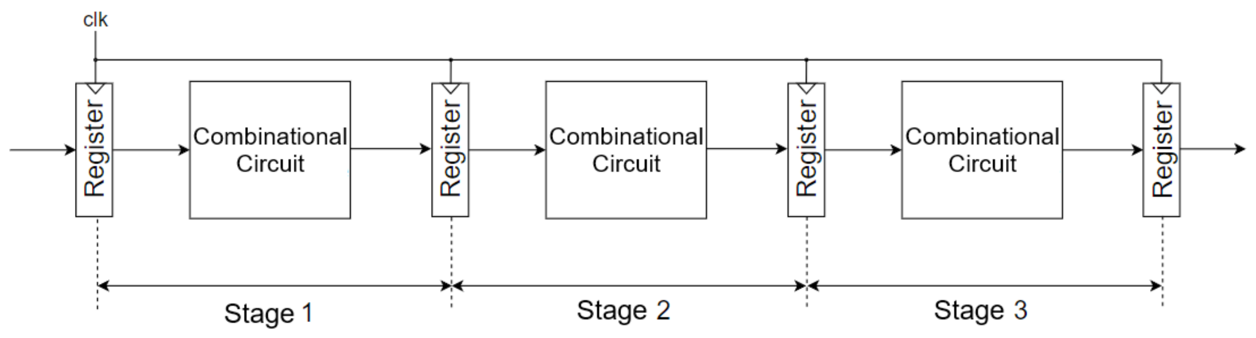
Remark 1: The difference between Combinatorial and Sequential Multipliers/Dividers is that Combinatorial have the advantage of being much faster than Sequential ones. On the other hand, the complexity and the number of gates for the construction of Sequentials are much lower compared ot Combinatorial.

Remark 2: In practice, to benefit from the speed of Combinatorial Multipliers/Dividers and the simplicity of Sequential Multipliers/Dividers, often a hybrid implementation is used.

Remark 3: Among the well-known Sequential Circuits, there are also Control Units (CU), they responsible of controlling Processors from inside.

8. Pipeline

Pipeline is a form of Sequential Circuits that do not use Loops in their operation. The Pipeline circuit consists of a fixed number of stages. We can see in the figure below a 3-stage pipeline for instance.



- ➔ The Pipeline can be considered as a special case of Sequential Circuits with constant and fixed processing of data, without variation in processing as for Sequential Circuits based on loops.
- ➔ The Pipeline is generally used for data streaming applications, with fixed and stepwise processing of this data.
- ➔ The Pipeline exercises parallel processing, in the way that several pieces of data are executed at the same time but each at a different stage.

Remark : The most obvious example of the use of Pipelines is undoubtedly the Hardware decoders of audio and video streams, generally found inside Graphics Cards, they allows the decoding of video and audio stream.