**Chapter 5**

Table of Contents

[Introduction 3](#_Toc44600580)

[SR Latch 4](#_Toc44600581)

[D Latch 9](#_Toc44600582)

[Synchronous and Asynchronous Sequential Circuits 10](#_Toc44600583)

[Flip Flops 12](#_Toc44600584)

[SR Flip Flop 13](#_Toc44600585)

[D Flip Flop 16](#_Toc44600586)

[JK Flip Flop 18](#_Toc44600587)

[T Flip Flop 21](#_Toc44600588)

[Racing Condition 23](#_Toc44600589)

[Master Slave Flip Flop 25](#_Toc44600590)

[Flip Flop Conversion 30](#_Toc44600591)

[Asynchronous Reset 31](#_Toc44600592)

[Analysis of Sequential Circuits 32](#_Toc44600593)

[State Reduction 36](#_Toc44600594)

[Sequential Circuit – Design Procedure 39](#_Toc44600595)

[Sequential Circuits – Analysis Procedure 43](#_Toc44600596)

[Sequence Detector 45](#_Toc44600597)

## Introduction

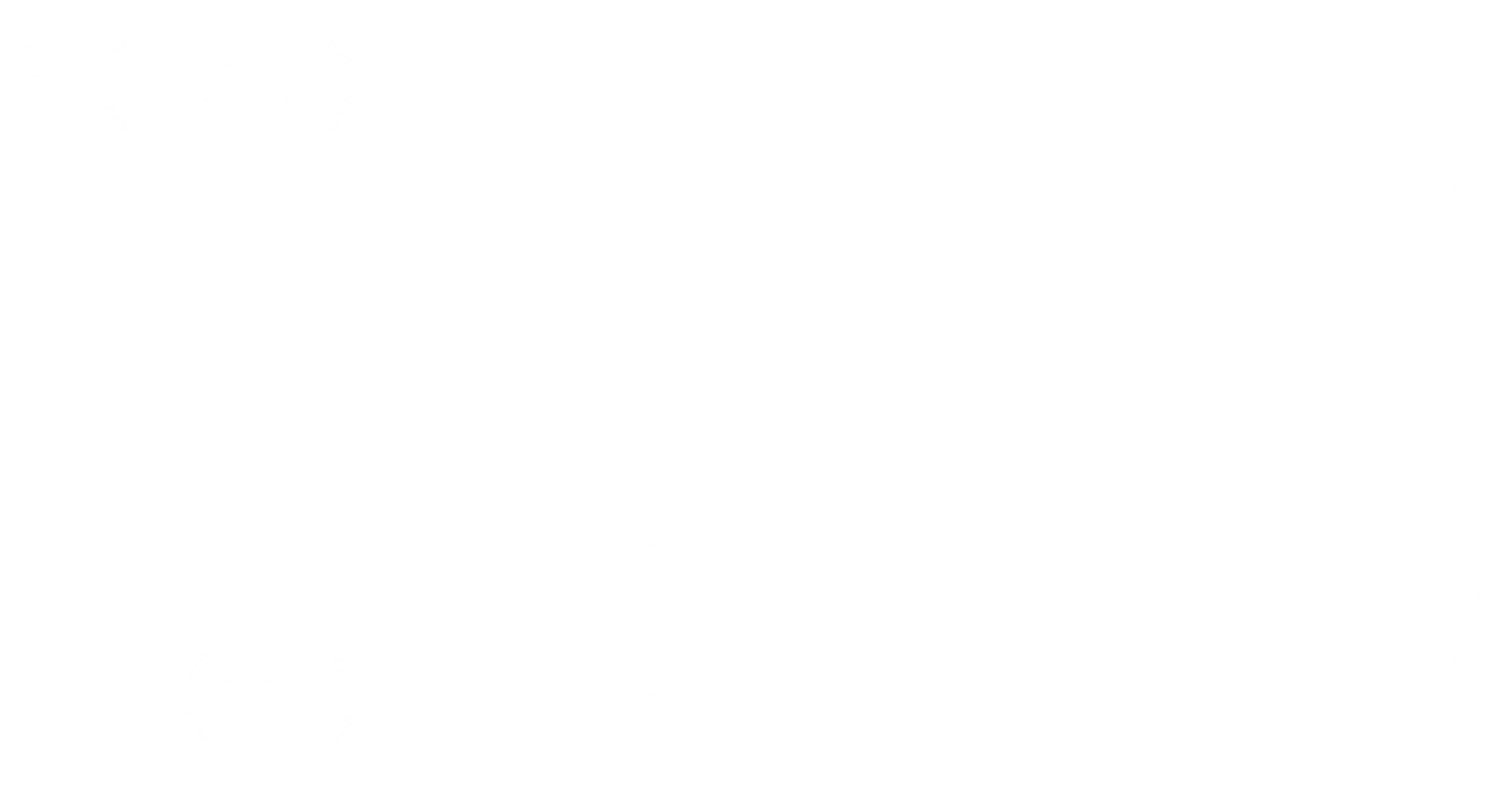
So far, combinational circuits have been used. Their outputs depend on the current input, meaning they have no memory or dependence on previous inputs. Sequential circuits act as storage elements and have memory. They consist of a combinational circuit, from which an output goes to a storage element, from which the path goes back to an input in the combinational circuit. This forms a path called a feedback path.

The storage element stores binary information, and the information stored defines the state of the sequential circuit at that time. External inputs, along with the present state of the storage elements, determine the output. The external inputs also determine the conditions for changing the state of the storage elements, meaning the state of the storage elements is also dependent on the current input and the past states. A sequential circuit is thus specified by a time sequence of inputs, outputs and internal states.

## SR Latch

The SR Latch is the primary base for memory storage. This contains two cross-coupled NAND or NOR gates, along with two inputs, for set and for reset. Reset means output is , and set means output is . Here, the main output is .

Using NOR gates,



NOR gates are controlled by the input of . This means that, if either input is , the NOR gate will immediately know that the output must be , regardless of the other input.

Let and . If we initially set both values to , we cannot know what the output will be. The other inputs, and respectively, are unknown, so the outputs of both NOR gates are also unknown.

Let and (resetting the device). Since , , and since and , . From this, we can see that the device has been reset (since ) and that and have opposite values (thus the naming).

Next, let and . In other words, the inputs are being removed. now, since it is getting an input of from . since and . Thus, and have retained their values, as though remembering them.

Let and (setting the device). Since , . Since and , . Again, we see that the device has been set (since ) and that and have opposite values.

If and is set again, since and , and since and , . Again, this shows that and have retained their values.

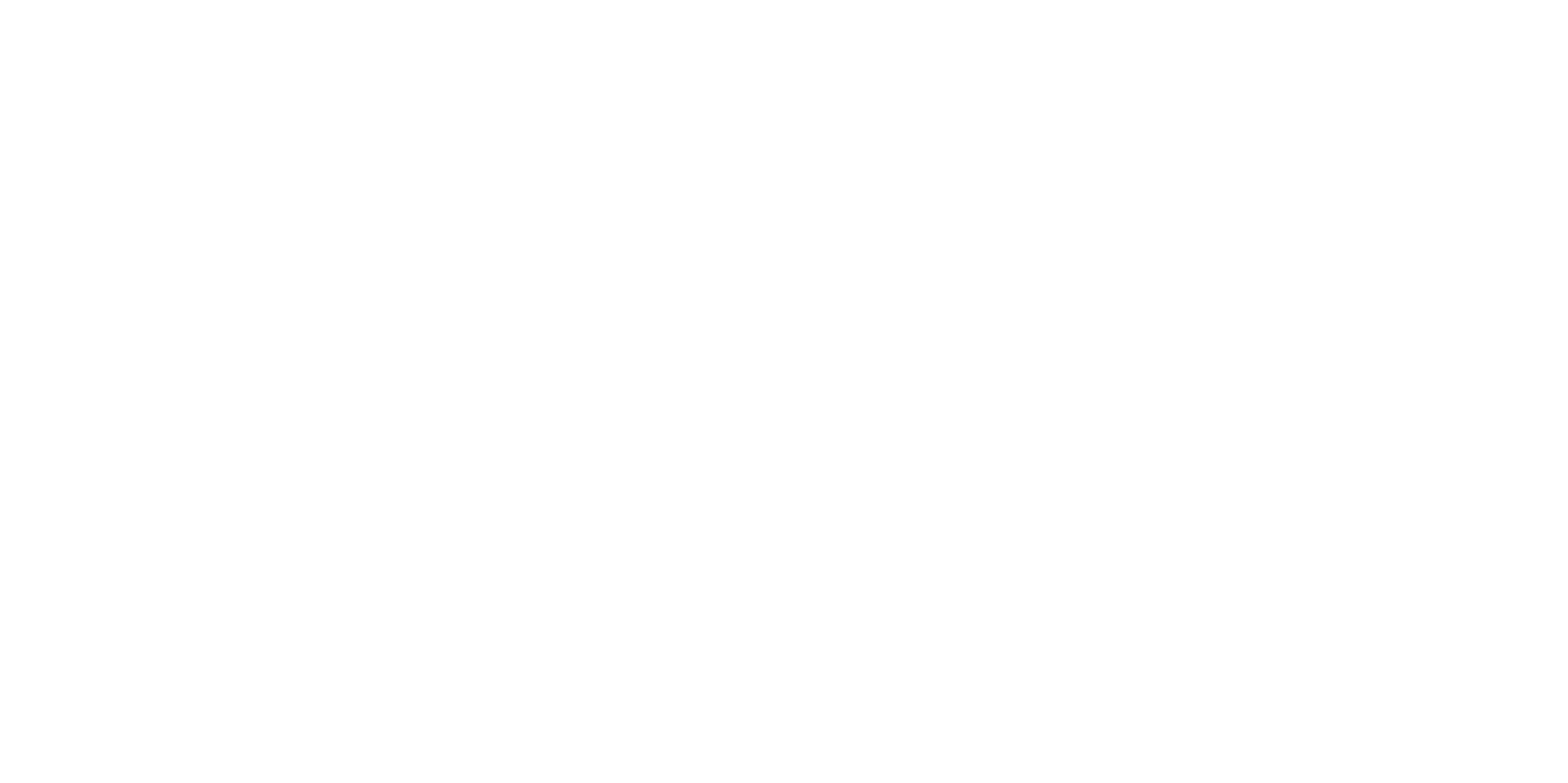
Now, let and . This scenario gives both and a value of . This would seem to contradict the naming used. However, this scenario is known as the forbidden state, and should never be used. The reason behind this is that, if, after setting and , both inputs are simultaneously switched to (the memory state), then the device enters an undefined state.

If we begin by observing , we see that and , so . Since and , . Now let and again (setting and to again), set and to simultaneously, and begin the observation from instead. Since and , . Since and , . This is the opposite of what was found from the previous observation. Neither observation can be considered ‘correct’, since neither has priority over the other. Thus, no proper outputs can be given. In technical language, the output depends on the order in which inputs change, but since both inputs changed simultaneously here, the output cannot be defined. However, switching either input to for even a moment will cause the device to give stable outputs again, since the device will again be either set or reset.

Function Table:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  | MEMORY | |
|  |  | 0 | 1 |
|  |  | 1 | 0 |
|  |  | FOBIDDEN | |

Using NAND gates,



Notice here that the positions of the and inputs have been flipped.

NAND gates are controlled by the input of . This means that if either of the inputs is , the NAND gate immediately knows that the output must be , regardless of the other input.

Let and . Since neither input is , neither NAND gate’s output can be known.

Let and . Thus, and . Note that for the NAND circuit, the reset state gives a value of and not as does the NOR circuit. Thus, this version of the SR latch is also sometimes called an latch.

Setting and now causes to remain since and to remain since and . Thus, and is the memory state for the NAND circuit.

Let and . Thus, and . Again, the set state gives , which is the opposite of what happened with the NOR circuit.

Finally, let and . This is the forbidden state for the NAND circuit. Both and will have a value of , and if and are both now set to simultaneously, the circuit will again enter the indefinitely state with no stable outputs previously described.

Function Table:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  | FORBIDDEN | |
|  |  |  |  |
|  |  |  |  |
|  |  | MEMORY | |

The biggest problem with the SR latch is the forbidden state, that can be acquired by mistake.

Latches are said to be level sensitive, since they operate with signal levels as inputs. They receive input constantly, and the input changes instantly. The output depends on the order in which input changes.

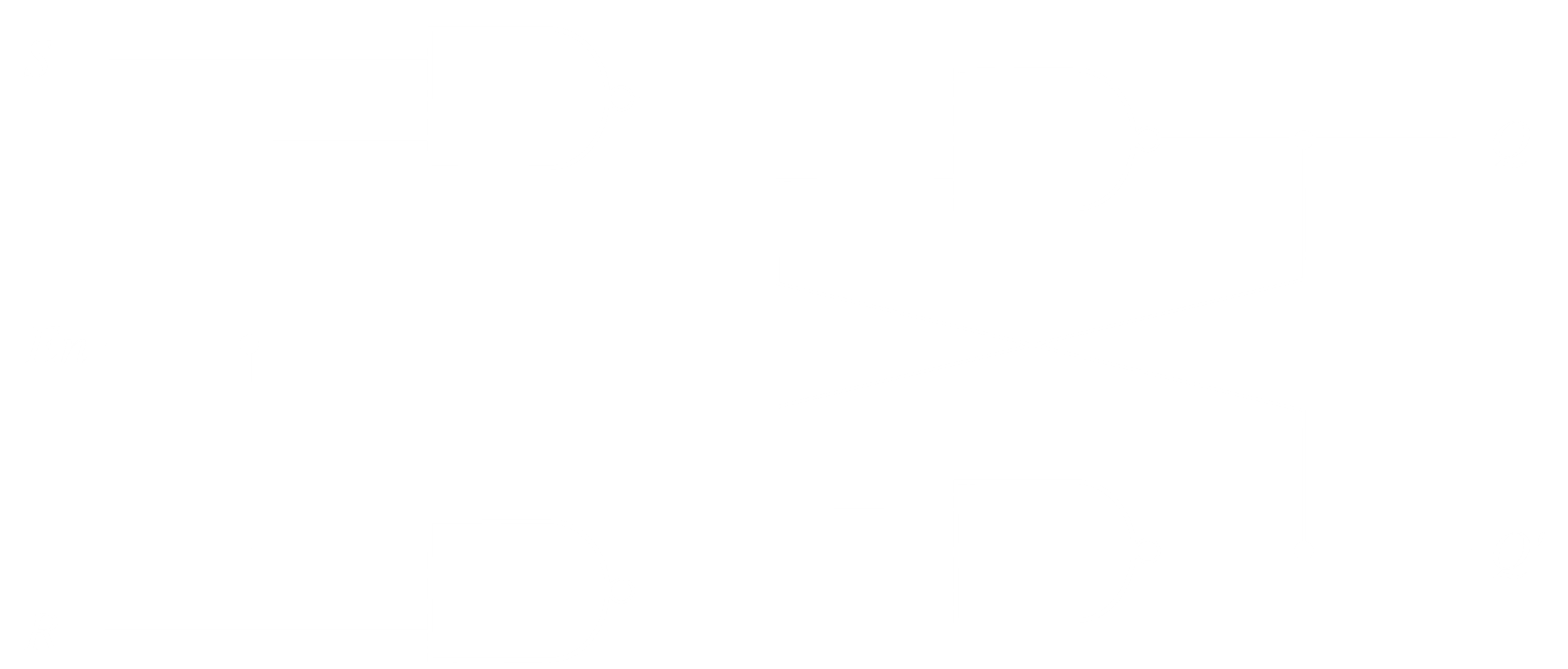
Characteristic Table:

A characteristic table shows the behavior of the next output depending on the current output and the given inputs. We are considering the characteristic table for the NOR SR Latch. The function table will be useful.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  | undefined |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  | undefined |

When and , retains the value of . Otherwise, it changes as expected from the function table.

The SR latch can be modified to include an enable input as shown below.



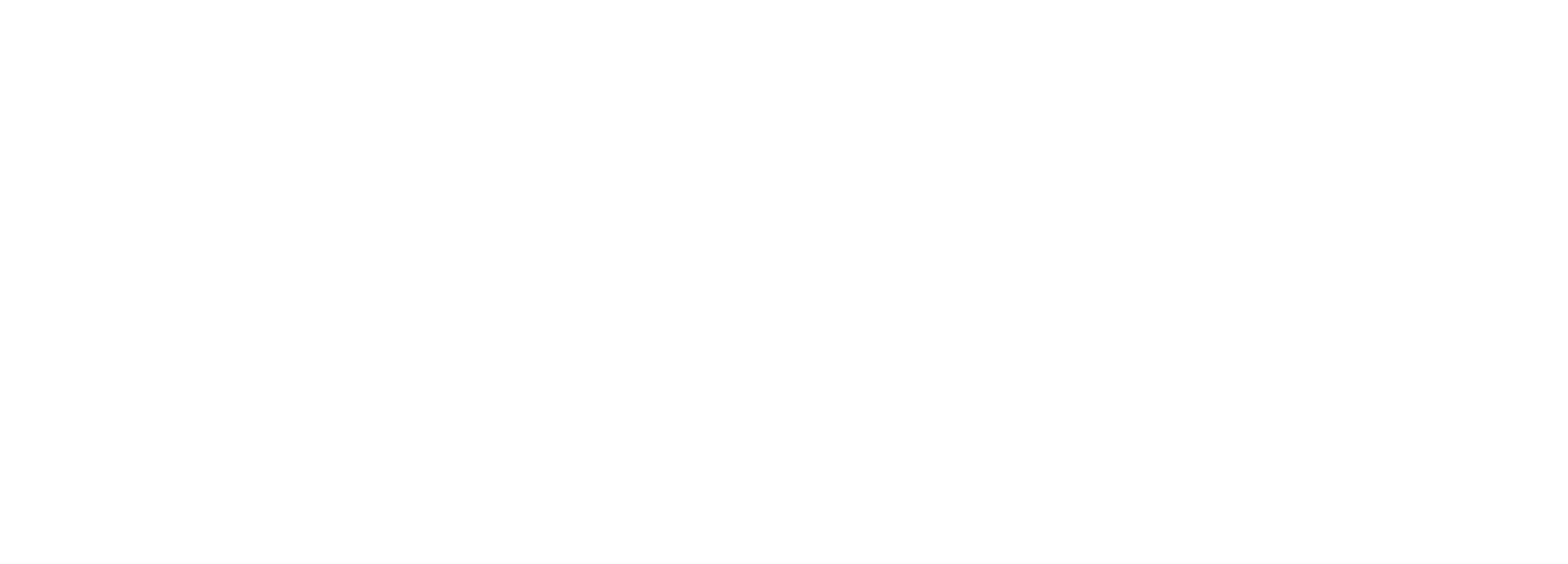
Function Table:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  | MEMORY |
|  |  |  | MEMORY |
|  |  |  |  |
|  |  |  |  |
|  |  |  | FORBIDDEN |

Note that this is an SR latch and not an latch.

## D Latch

One way to fix the problem of the undefined state is to use a D latch. A D latch is just an SR latch with both inputs coming from the same source, and one input being inverted as shown below. Thus, both inputs can never simultaneous be switched to or . This however, is more of a workaround than a solution.



Function Table:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  | Memory |
|  |  |  |
|  |  |  |

## Synchronous and Asynchronous Sequential Circuits

There are two types of sequential circuits, synchronous and asynchronous.

An SR Latch is an asynchronous sequential circuit. The behavior of an asynchronous sequential circuit is based on the order of input signals at any instant of time. Time delay devices are intentionally added, and these work as memory elements. The capability of such a device to store information depends on how long it takes for a signal to propagate through the system. In practice, separate delay units may not be required since the internal propagation delay of logic gates is sufficient. The delay is thus the required memory, meaning an asynchronous circuit can be regarded as a combinational circuit with feedback. This also means that an asynchronous sequential circuit can become unstable at times, due to the feedback.

For example, in a circuit that takes two inputs and and gives one output which is high for inputs and , say the current input is , meaning the output is high, . is now switched from to , and is then switched from to . The correct output should still be . However, if there is a delay in this switch, then an incorrect answer will be shown. If it takes seconds after has been switched to switch , then for that duration of time, the input is , causing to give a low output, . To avoid this, a time delay device could be added, so that the previous state is still shown for some time before allowing the new input through, so that there is enough time to switch . The circuit is essentially, ‘remembering’ the previous state for that duration of time. A remaining drawback of such a system is that, if a state were prohibited for some reason, it could still be acquired by mistake, just like the input was acquired here by mistake. This could lead to unstable states.

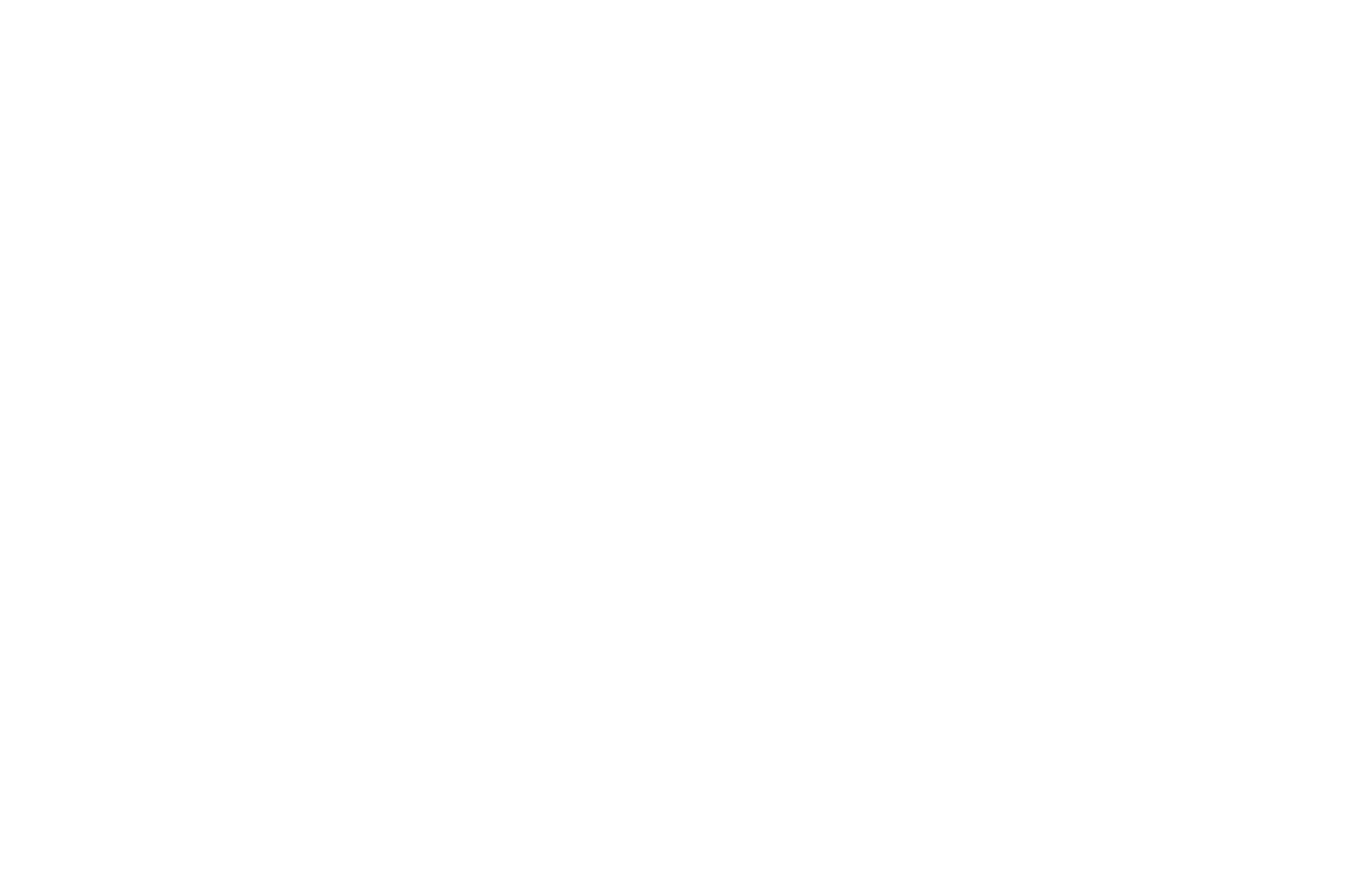
A synchronous sequential circuit on the other hand, handles this problem by using a clock pulse, which sends a signal at discrete instances of time (a pulse). The pulses are distributed in such a way so that elements are only affected by inputs when the pulses arrive synchronizing the input (for example an AND gate with one input being the clock and the other input being external). The clock pulse handles when computational activity occurs, and the external inputs handle what computational activity occurs. Thus, instable states can be avoided. Such circuits are also called clocked sequential circuits.

## Flip Flops

The storage element used in clocked sequential circuits is called a flip-flop. A flip-flop is a binary storage device capable of storing one bit of information. The flip-flop must reach a stable state ( or ) before receiving new input, so the propagation delay in the circuit and the time between pulses from the clock are very important. The flip-flop can store a binary state indefinitely, given that a power supply is present.

The only difference between a latch and a flip flop is that a flip flop depends on a clock pulse while a latch depends on an enable input.

## SR Flip Flop



The following is the truth table for an SR flip flop circuit:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  | Memory | |
|  |  |  | Memory | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  | Undefined | |

When the clock is set to , and are both . Thus, if and , remains and remains . Similarly, if and , remains and remains . So, this is a memory state.

When clock is set to , and , and are again both . The same thing as above occurs, so this is also a memory state.

For , and , and . This causes to be , which leads to becoming . This is thus the reset state.

For , and , and . This causes to be , which leads to becoming . This is thus the set state.

For , and , both and are , so and are both . We have seen from the SR Latch that this scenario causes an instable state, so this is known as the forbidden state.

The following is the characteristic table for the circuit. A characteristic table shows the expected value of for different combinations of , and .

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  | Undefined |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  | Undefined |

From the characteristic table, we can derive an equation for using a K’Map.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| QnSR | 00 | 01 | 11 | 10 |
| 0 | 0 | 0 |  | 1 |
| 1 | 1 | 0 |  | 1 |

Note that the third column does not contain s, so they do not indicate don’t cares. These are simply left out from all consideration since they are the invalid states.

From this, , given that .

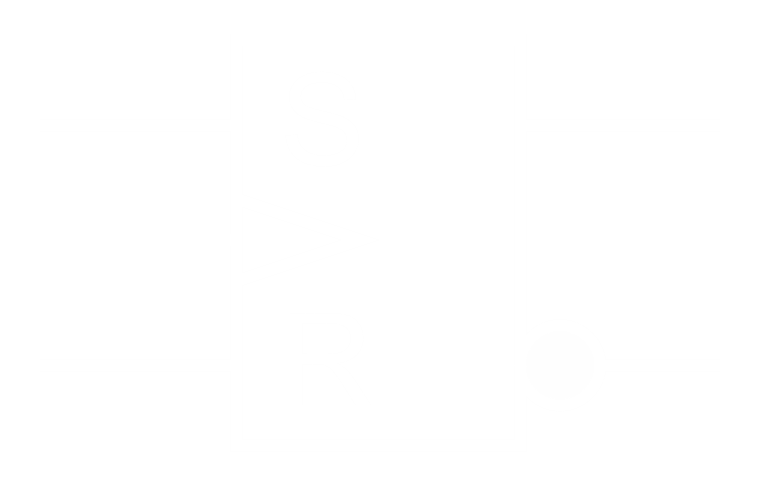
However, in some cases, the invalid states are considered as don’t cares, which leads to the expression .

Next, we make an excitation table. This shows the input values that are required, for different combinations of and .

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

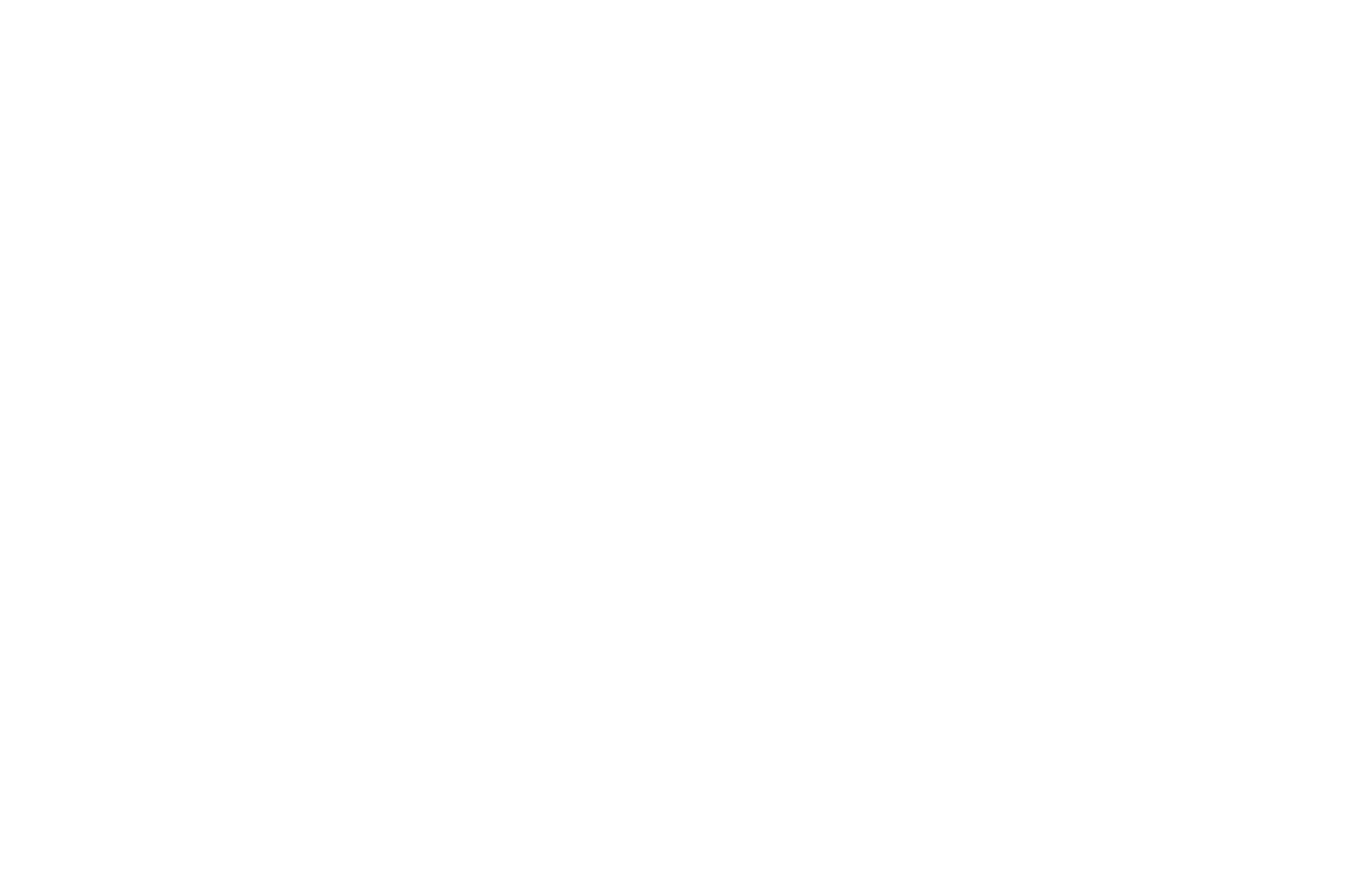
Note here that for two combinations, don’t cares exist. This means the value of does not depend on the don’t care value in those cases.

The block diagram of an SR Flip Flop looks like this:



## D Flip Flop

Similar to how the D-Latch was a workaround for the invalid state present in the SR-Latch, the D-Flip Flop is a workaround for the invalid state present in the SR-Flip Flop. The D-Flip Flop is not however a proper solution to the problem presented by the SR-Flip Flop.

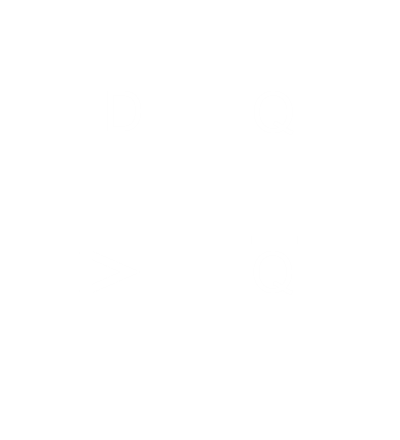


Truth Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  | Memory | |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Thus, . This is why this device is called a D-Flip Flop. It stands for Data Flip Flop.

The block diagram for a D-Flip Flop is shown like this:



Characteristic Table:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

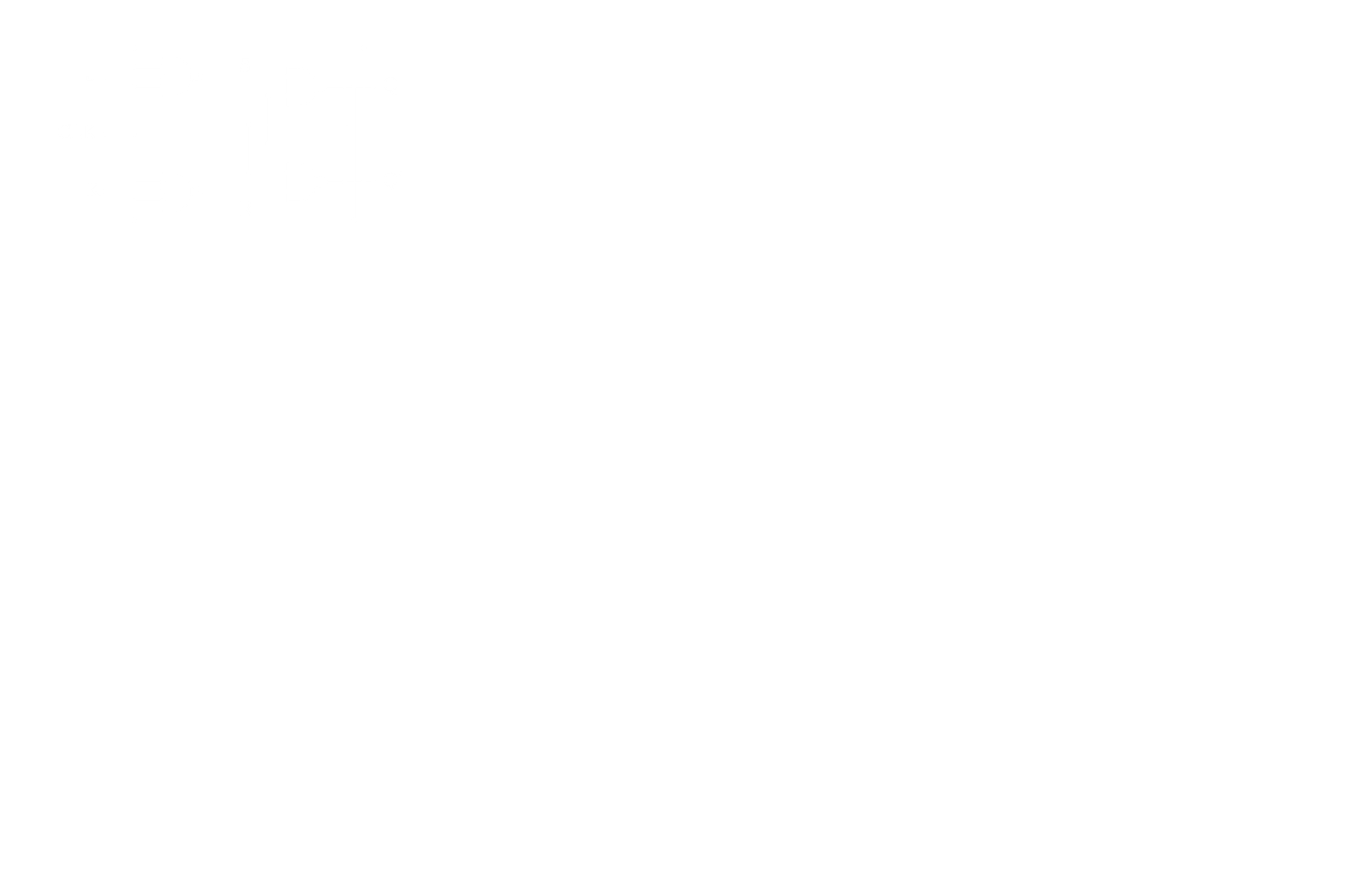
Thus, .

Excitation Table:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## JK Flip Flop

The JK Flip Flop solves the undefined output problem presented in an SR Flip Flop.



When , or when , and , the circuit acts exactly the same way the SR Flip Flop did, giving the memory as output.

When , and , . Let us assume that . Since and are both , . Since , remains . Now let us assume that . This causes to become . Since , and are all , , and thus becomes . As described earlier, this then leads to and . Thus, regardless of the previous states, setting and will lead to and .

When , and , . This time let us assume . Since and are both , . Since , remains . Now let us assume . Thus, . Since , and are all , , which flips to . The process described earlier occurs again, ending with and . Thus, regardless of previous outputs, for and , and .

Finally, let us consider , and . Let us consider and . At this moment, since , . Since , and are all , , which in turn flips to . Now, since and , becomes . The outputs have been flipped to and . At this moment, since , . Since , and are all , . This in turn flips to . Since and are both , is flipped to . Again, the outputs have been toggled. This state is thus known as the toggle state. The previous output, regardless of what it was, is flipped in this state.

A problem presented by the toggle state is that, if the state is left as is, i.e. the clock is not switched off, the outputs will keep toggling. This leads to a condition known as the racing condition which will be discussed later.

The truth table for the JK Flip Flop:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  | Memory | |
|  |  |  | Memory | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  | Toggle | |

From here, we can derive the characteristic table:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

The K Map can now be drawn and can be used to derive an equation for .

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| QnJK | 00 | 01 | 11 | 10 |
| 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |

Thus, .

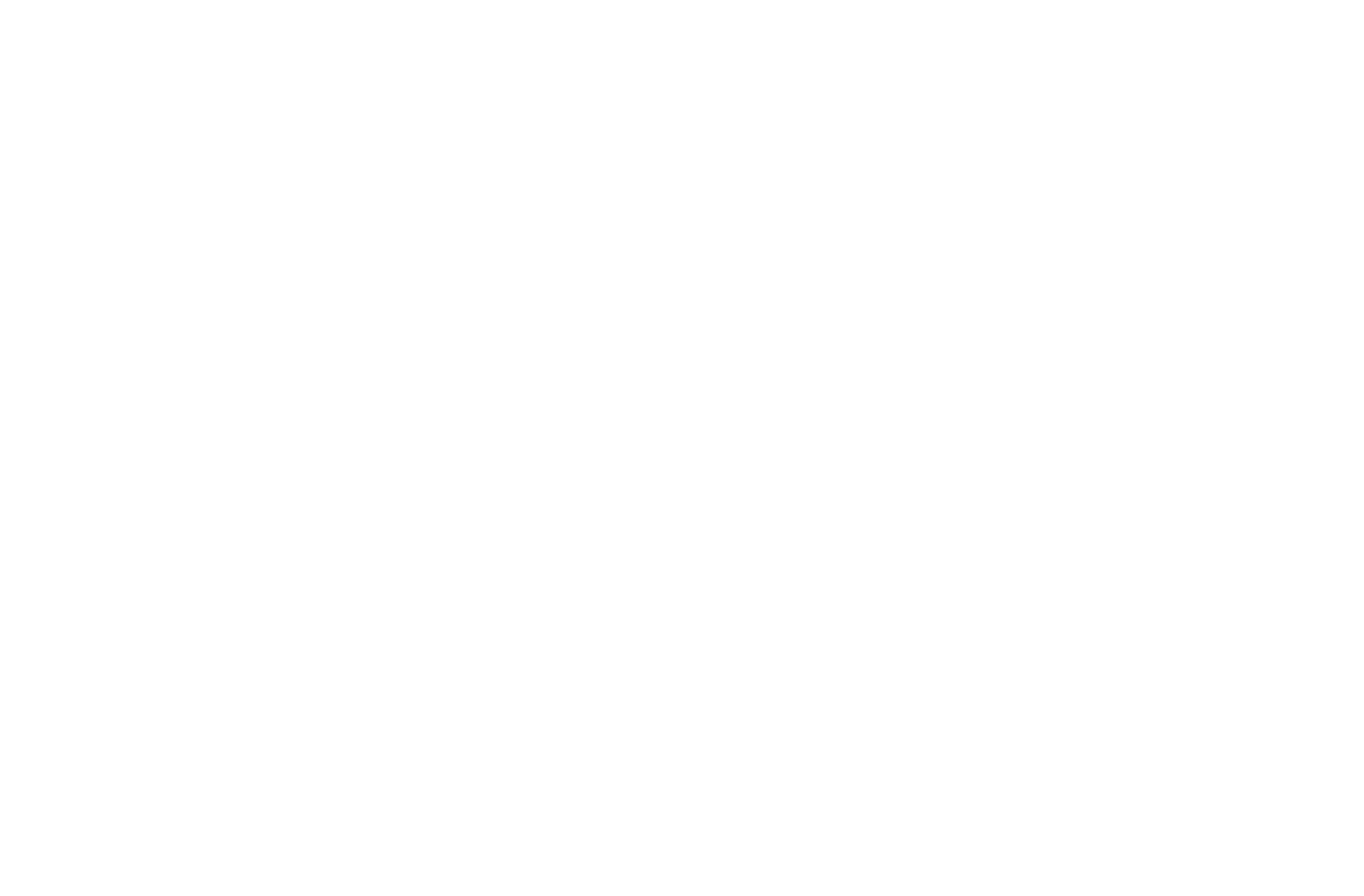
The excitation table:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

From here, since we can assume the don’t care values in the bottom two cases match the values of , and , again since we can assume the don’t care values in the top two cases match the values of .

## T Flip Flop

The T Flip Flop, is made by joining the two inputs of the JK Flip Flop.



This means, only the case and or and are valid here.

Thus, the truth table:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  | Memory | |
|  |  | Memory | |
|  |  | Toggle | |

From here, we can see that have or set to will retain the memory condition, while setting both to will toggle the output. This is thus known as a Toggle Flip Flop.

Characteristic Table:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Excitation Table:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Racing Condition

As mention earlier, leaving a JK Flip Flop in the toggle state leads to a condition known as the racing condition.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | | | |  |  |
|  |  |  | | | |  |
|  |  |  | | | |  |  |
|  |  |  |  | | | |  |  |
|  |  |  | | | |  |
|  |  |  | | | |  |  |
|  |  |  |  | | | |  |  |
|  |  |  | | | |  |
|  |  |  | | | |  |  |
|  |  |  |  | | | |  |  |
|  |  |  |  |  |  |  |
|  |  |  | | | |  |  |

In the above signal diagram, and are left at . Every time become , starts toggling from to and back. This happens rapidly and multiple times while . When becomes , the last state obtained by is preserved.

This state is thus instable, and it is difficult to take readings properly. There are a few ways to solve this problem.

One way would be to increase the clock pulse’s frequency, so that the output does not get enough time to toggle two times. However, this is impractical. In reality, the toggling happens very quickly and frequently, making it difficult to prevent it in this method while also leaving enough time to get a proper reading.

Another method is to use edge triggering (as opposed to level triggering where the output is active for the duration during which ). Edge triggering means that the output becomes active at the moment when changes from to or from to . This takes an extremely short amount of time, thus only allowing the toggle to occur once. There are two types of edge triggering, positive edge triggering, which activates the output when switches from to , and negative edge triggering, which activates the output when switched from to .

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

Here, is the output under positive edge trigger and is the output under negative edge trigger. Notice that the outputs are exactly the same, but shifted by , where is the time period of the clock pulse.

## Master Slave Flip Flop

The last method to solving the racing condition problem is to use a Master Slave Flip Flop. A Master Slave Flip Flop takes two flip flops one after another, with the output of one flip flop becoming the input of the other. However, both flip flops are not activated at the same time. This is done by using one input, but inverting the input before it enters the second flip flop.

The circuit for a Master Slave JK Flip Flop looks like this:



Notice that the clock input to the slave has been inverted. Thus, when the Master is active, we get an output , which goes to the Slave, but when the output from the Slave, , is made available, which occurs when , and is taken back to the input of the Master, it does not have any effect, thus removing the racing condition.

The signal diagram looks like this:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |  |

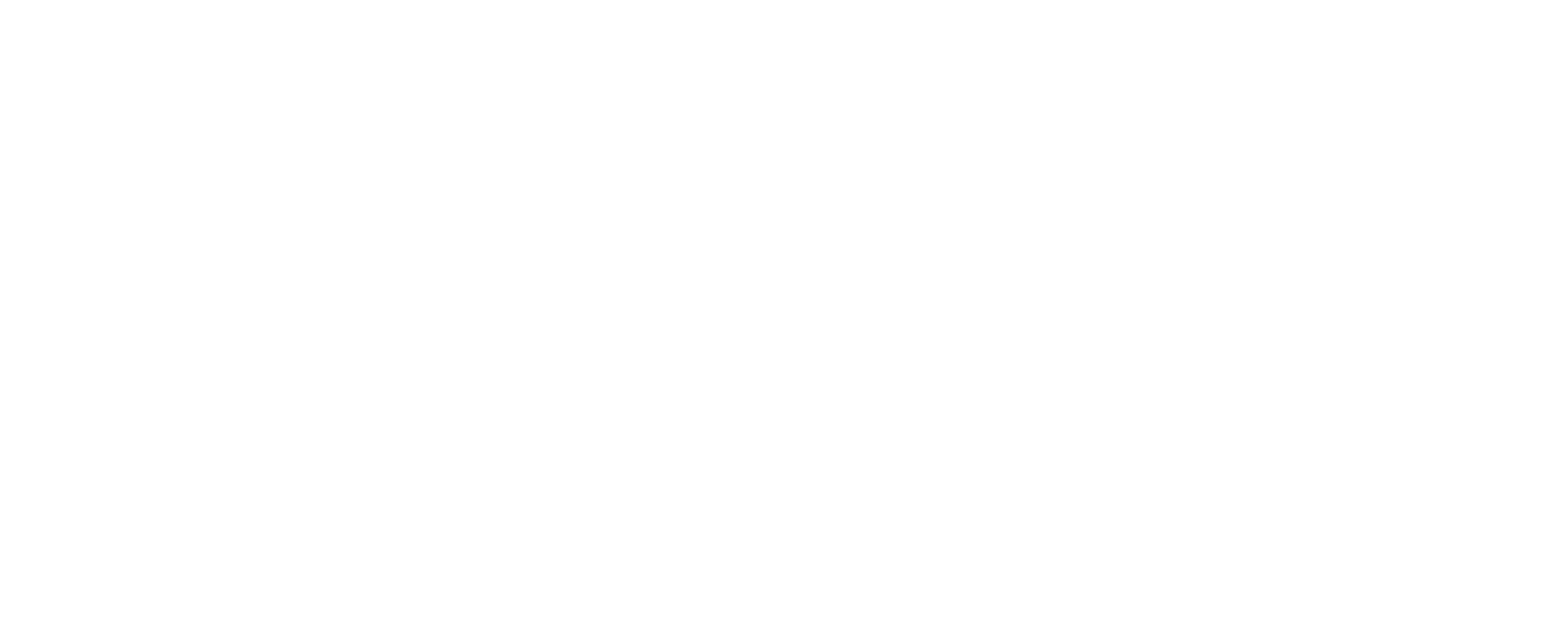
Here, is toggled every time becomes , but this output does not toggle again since the input from does not change (since the Slave is inactive). The output is then stored when becomes and is toggled again when .

on the other hand, is toggled every time becomes , and does not change until again.

The Master Slave JK Flip Flop is what is actually used in toggling circuits like counters.

Master Slave D Flip Flop

The Master Slave Flip Flop is used for other flip flops as well, such as the D Flip Flop. The block diagram of a Master Slave D Flip Flop looks like this:



The following is a signal diagram for the Master Slave D Flip Flop. The outputs for a positive edge trigger and a negative edge trigger (these are unrelated to the Master Slave Flip Flop) have also been included to show the effect of using the edge triggering method once again. The signal for is random.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | | |  |  |  |  | | |  |
|  |  |  |  |  |  |  | | |  |  |  |  | | |  |
|  |  |  |  |  |  | | |  |  |  |  | | |
|  |  |  |  |  |  | | |  |  |  |  | | |  |
|  |  |  |  |  |  |  | | |  |  |  |  | | |  |
|  |  |  |  |  |  | | |  |  |  |  | | |
|  |  |  |  |  |  | | |  |  |  |  | | |  |
|  |  |  |  |  |  |  | | |  |  |  |  | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | | |  |  |  |  | | |  |
|  |  |  |  |  |  |  | | |  |  |  |  | | |  |
|  |  |  |  |  |  | | |  |  |  |  | | |
|  |  |  |  |  |  | | |  |  |  |  | | |  |
|  |  |  |  |  |  |  | | |  |  |  |  | | |  |
|  |  |  |  |  |  | | |  |  |  |  | | |
|  |  |  |  |  |  | | |  |  |  |  | | |  |
|  |  |  |  |  |  |  | | |  |  |  |  | | |  |
|  |  |  |  |  |  | | |  |  |  |  | | |
|  |  |  |  |  |  | | |  |  |  |  | | |  |

Remember that the output for a D Flip Flop is just the input at that time.

Let us first observe the output of the Master, . For the time period to , , so . switches to just after the clock becomes , so does not reflect that change. When the clock becomes high again at , realizes that , so becomes . switches to just after the clock becomes low at , so does not reflect that change. At , switches to since at that time, however, switches to immediately after that, and since the clock is still high, immediately reflects that change. This results in a very quick change from to and back to for as shown in the diagram. This very fast change is a glitch in the results. From here, next becomes after , but since the clock is low then, remains . At , another glitch appears, as is again forced to switch very quickly from to and back again, due to the fact that switched to just after the clock became high.

Now let us observe , the output of the slave. This output depends on the output from the master, and not directly on , and detects changes while the clock is low instead of high. first changes from to at . When the clock next becomes low at , detects the change and becomes . only changes to a low output 2 times after this, at the two glitches. However, since both of these glitches occur while the clock is high, and returns to before the clock becomes low again, does not detect these changes. This means that the output signal given by is glitch free.

Next, we observe , the output of the positive edge trigger. This output is dependent directly on and is only affect at the exact moment the clock switches from low to high. These switches occur at , , , and . At , so remains . At , so switches to . At , becomes , so . then immediately switches back to , but since the clock has already switched, does not reflect this change. At , , so switches to . At , , so . Again, soon switches to but since the clock has already switches, the change is not reflected in . Thus, is also glitch free.

Lastly, for , the output of the negative edge trigger. also depends directly on and is only affect at the exact moment the clock switches from high to low. These switches occur at , , , and . At , , so . soon switches to but the change is not reflected on as the clock has already switched. At , so . soon switches but remains high. At , and is high. does not change at the exact moment the clock switches from high to low on any of these instances, so the changes are not reflected on .

A point to note is that and give the exact same outputs. This is always true for any flip flop.

## Flip Flop Conversion

In times of need, when we inevitably run out of flip flops, flip flops we require can be created from other flip flops. In order to do this, we must first establish what our requirements are, and what materials are available. Then, we must create the characteristic table for the required flip flop, and the excitation table for the available flip flop. This essentially tells us what outputs are expected from the required flip flop, and what inputs from the available flip flop will give us those outputs. From here, we can create a Boolean expression and then build the circuit.

For example, to create a D Flip Flop from a JK Flip Flop,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Characteristic Table for D Flip Flop | | | Excitation Table for JK Flip Flop | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Thus, and .

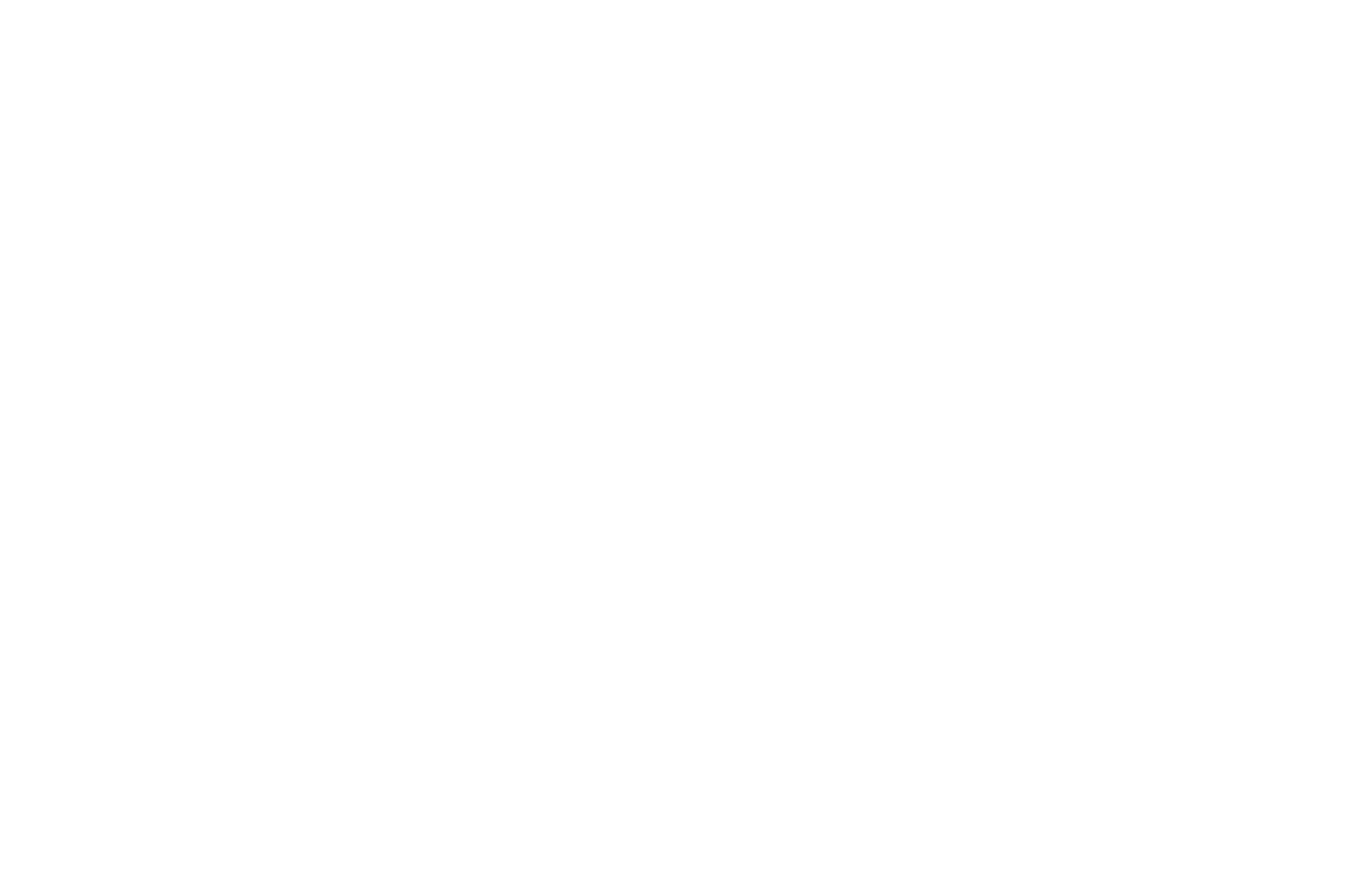
The block diagram will look like this:



## Asynchronous Reset

Throughout the chapter, multiple references have been made to the output being Memory. However, for memory to exist, an initial output must be given. The initial outputs are set by the Preset and Clear switches.

The following is the diagram for an SR Flip Flop with Preset and Clear:



Present and Clear are direct, asynchronous inputs. Their truth table is given below.

|  |  |  |
| --- | --- | --- |
| Preset | Clear |  |
|  |  | Invalid |
|  |  |  |
|  |  |  |
|  |  | Normal |

If Preset and Clear are both set to , and both become . This is thus the invalid state. For Preset and Clear , which causes to become (setting the output). For Preset and Clear , the opposite occurs (clearing the output). For Preset and Clear , The output will depend on the SR Flip Flop itself. The device is then in the working state.

## Analysis of Sequential Circuits

The analysis of a sequential circuit involves a state table, a state diagram and a state equation for the sequence of inputs, outputs and internal states. State tables and state diagrams can be compared to the truth tables used for combinational circuits, while a state equation can be compared to the input-output equation used with combinational circuits.

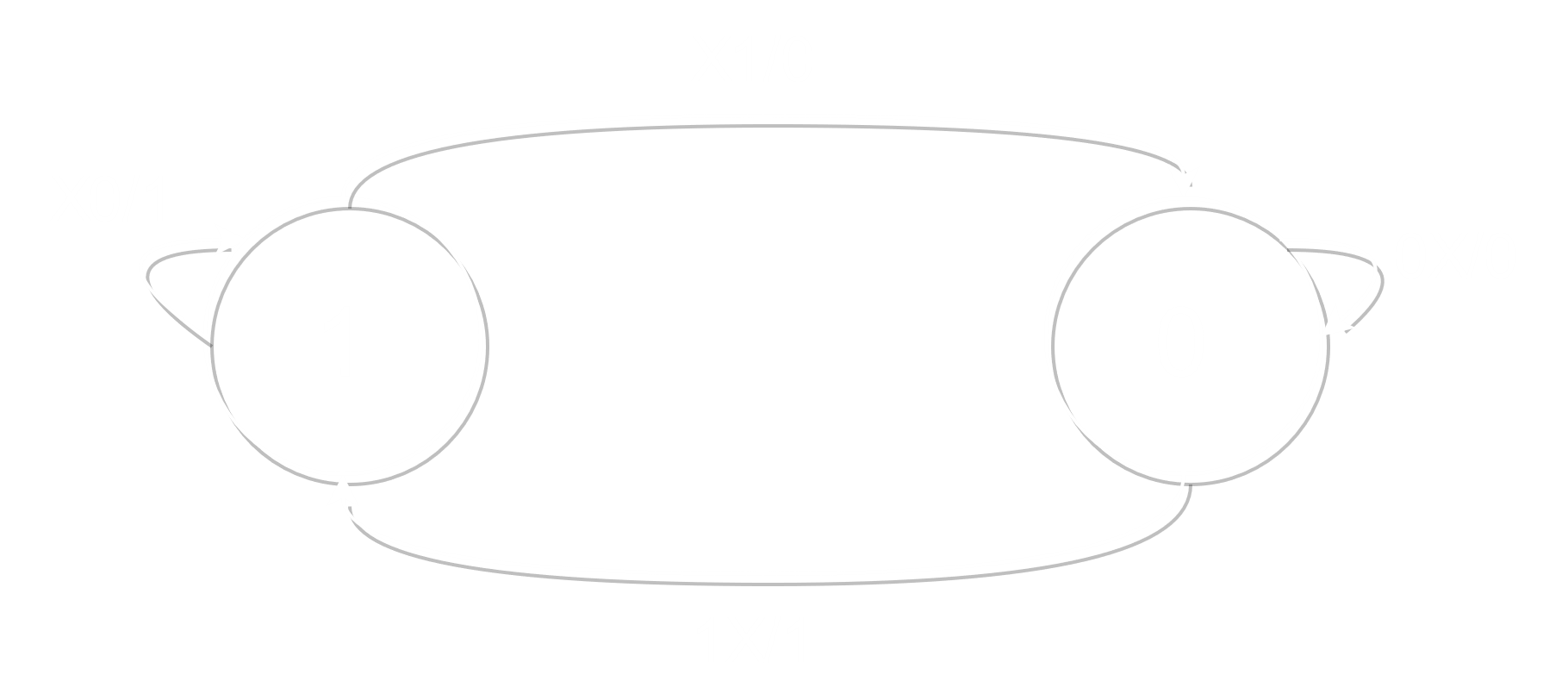
As an example, let us consider a single JK Flip Flop. Note that a sequential circuit is not usually this simple.

First, we make the state table. This consists of the present state, which is here, the present inputs, and , the next state, , and the output, which also happens to be here.

State Table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Present State | Present Inputs | | Next State | Output |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

From here, we can derive the state diagram:



The state diagram is simply a graphical representation of the state table. It shows how the system changes from one state to another for a certain input. For the single JK Flip Flop we are discussing, there are only two possible states, and . The values along the arrows show the input/output. For example, if the current state is , for the inputs and , the next state is also . So, a self-loop is shown and the input is written as (since the value of is irrelevant). The output is for both cases and is shown along with the input.

The state equation can be derived from the state table using a K Map.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| QnJK | 00 | 01 | 11 | 10 |
| 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |

Thus, .

Now let us consider a more complicated sequential circuit. The following diagram consists of two separate JK Flip Flops.



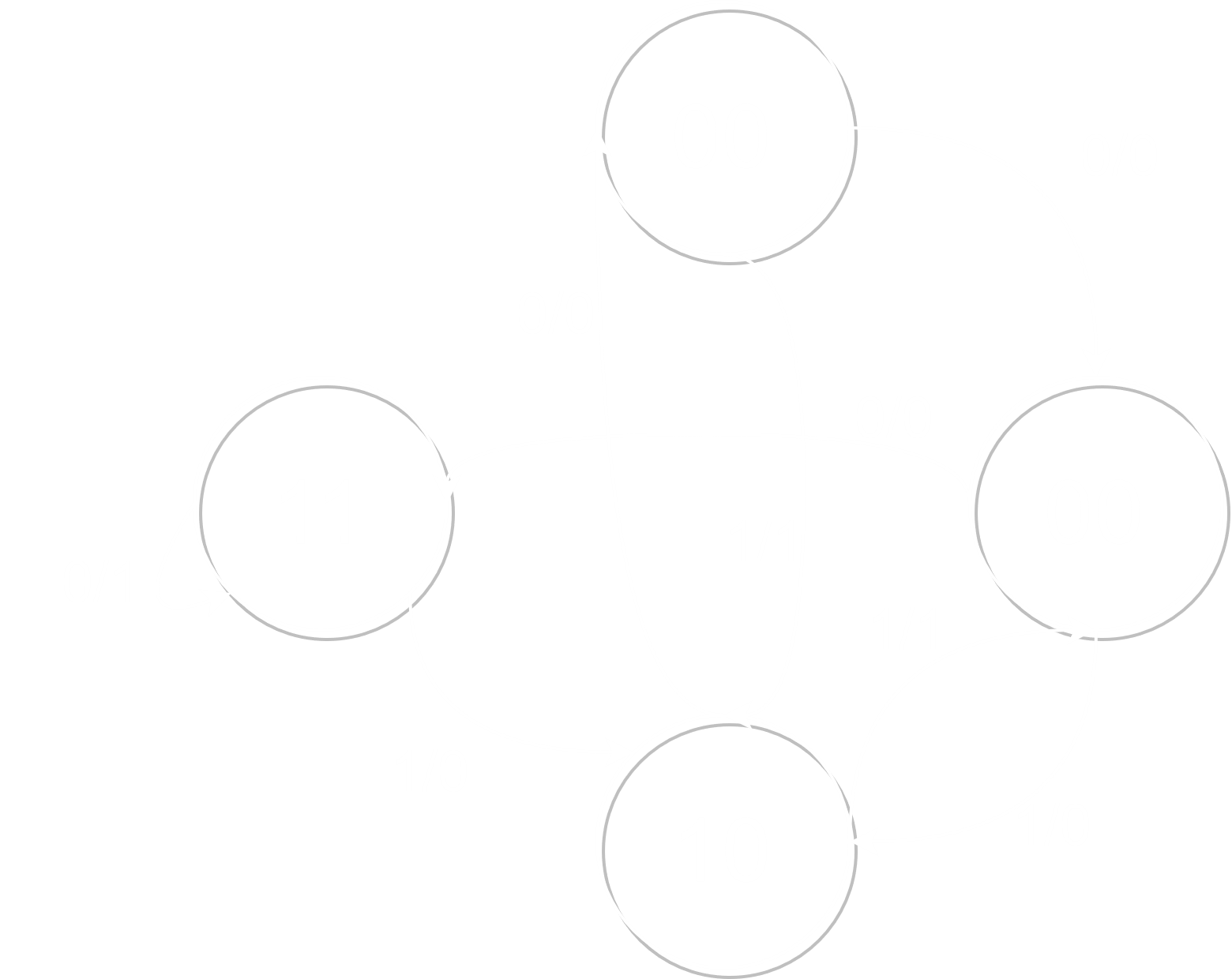
For simplicity, we are ignoring the inner workings of the circuit and are only concentrating on the two flip flops. We are also considering that there is only one input, , and one output, . The values for next state and output are given randomly and do not represent an actual circuit.

State Table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Present State | | Present Input | Next State | | Output |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

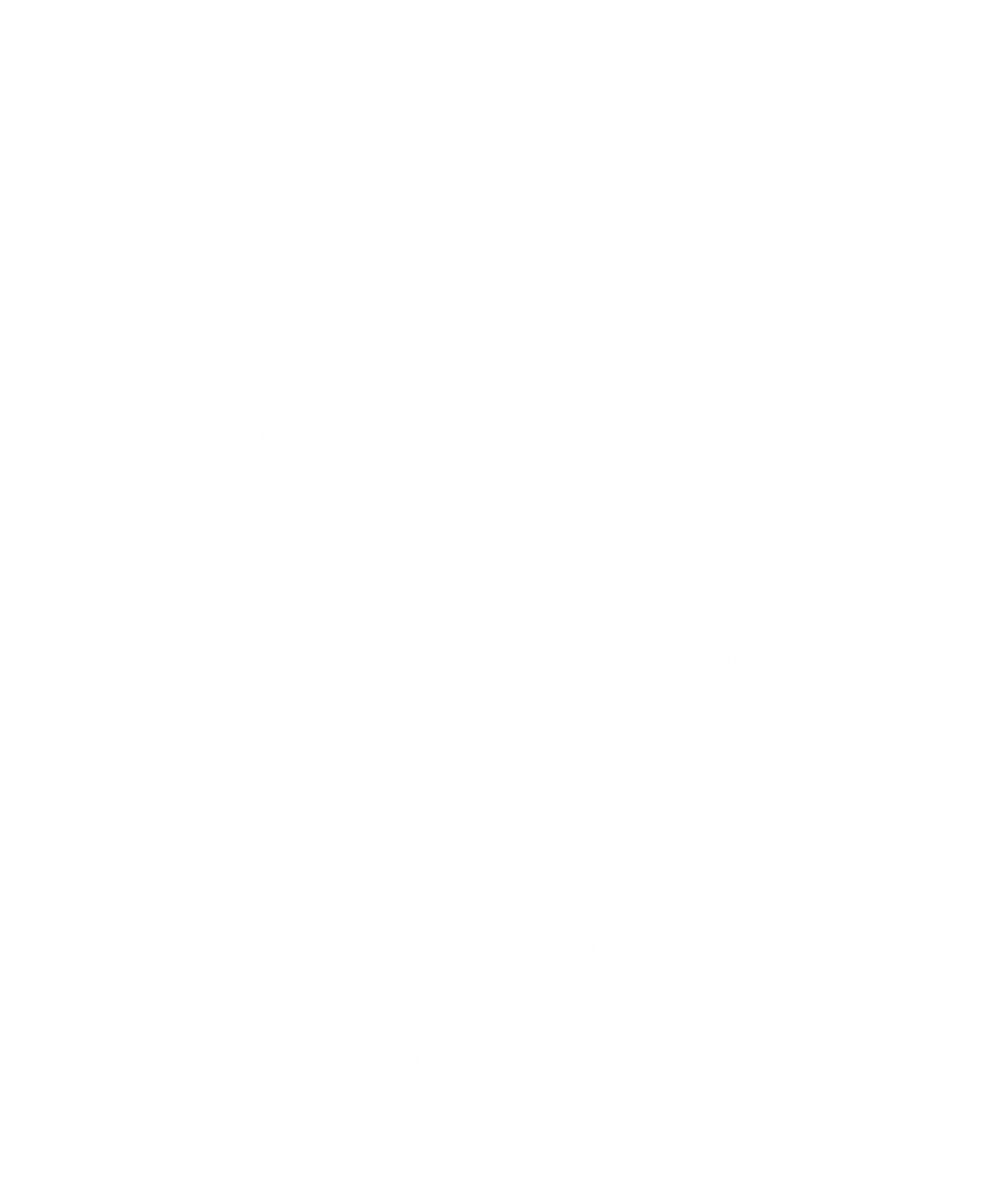
State Diagram:

Since there are 2 JK Flip Flops involved, there are 4 possible states.



## State Reduction

For the following state diagram, there are possible states. Since , flip flops are required to achieve this state diagram.



We will now consider whether it is possible to reduce the number of states. For this, we will require the state table. However, the state table is being drawn in a different manner this time. The inputs are being given a lower priority, and only the present state, the next state, and the output are being prioritized.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Present State | Next State | |  | |
|  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

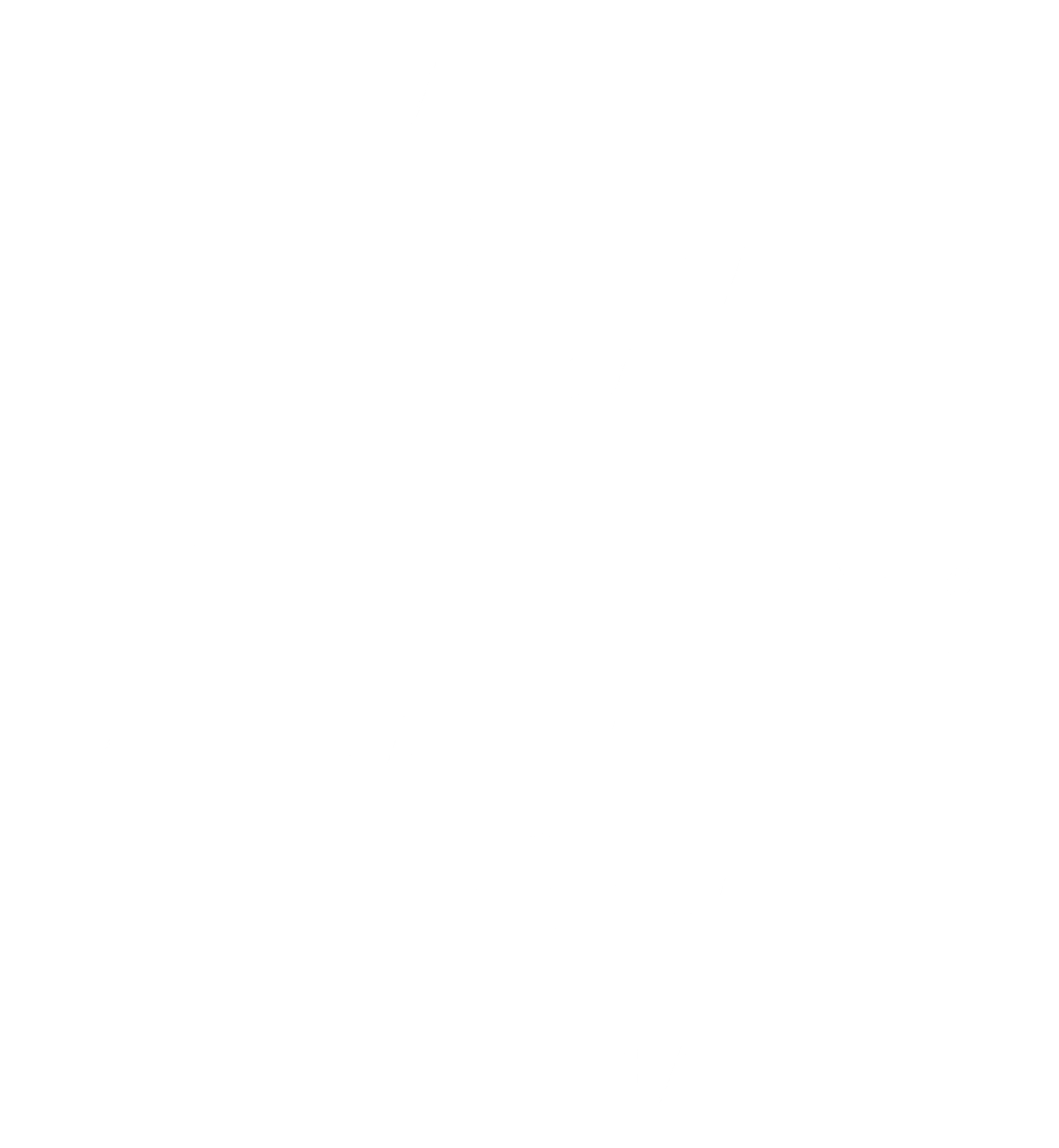
Next, we check whether any present states give the same next states and outputs. This condition Is met by and . can thus be removed from the table. Any places where appears in the table is replaced with an .

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Present State | Next State | |  | |
|  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Now we see that and meet the condition required for reduction. So is removed and any places where appears is replaced with a .

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Present State | Next State | |  | |
|  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

No further reduction is possible. The new state diagram looks like this:



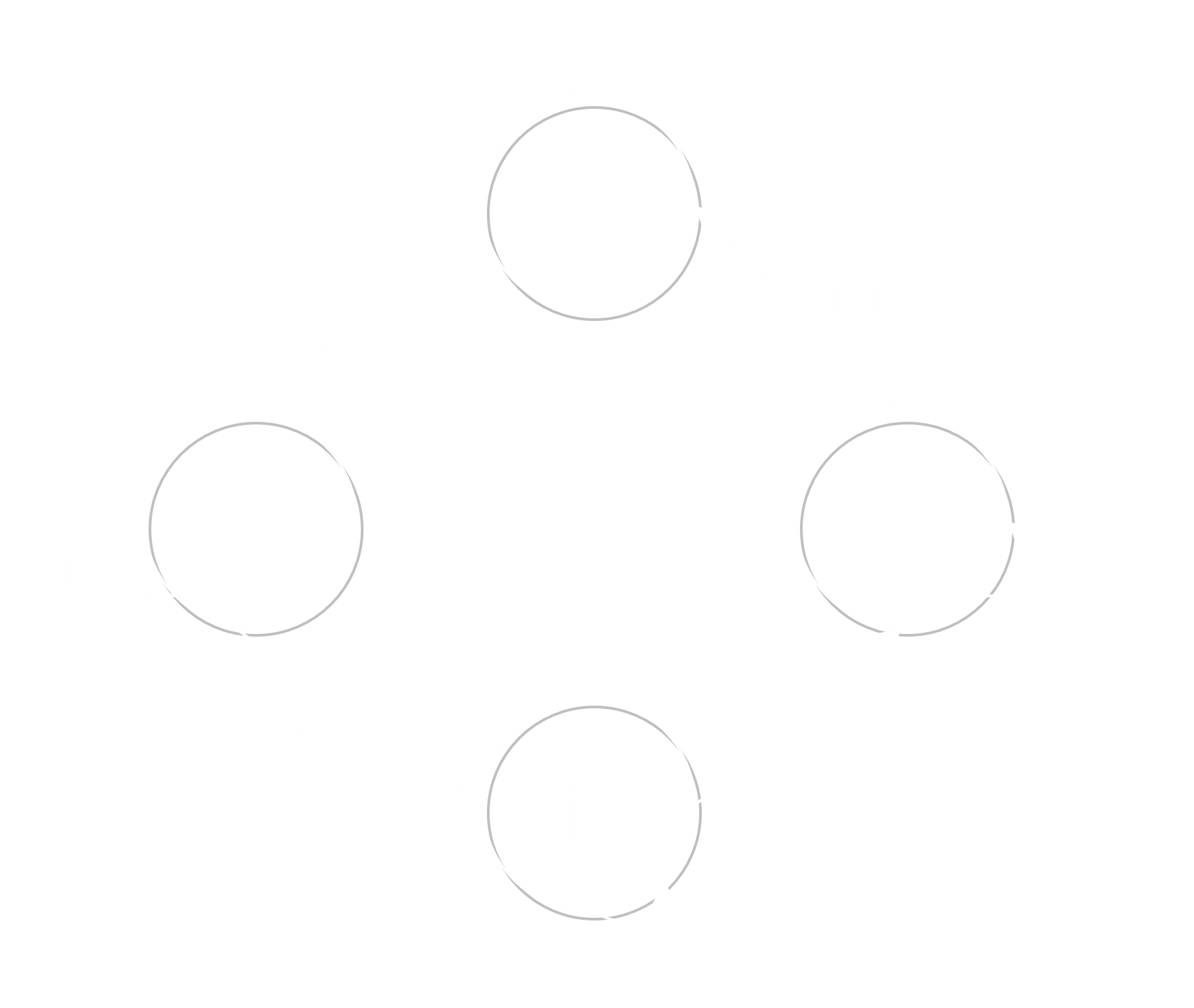
## Sequential Circuit – Design Procedure

When designing a sequential circuit, the following steps are followed:

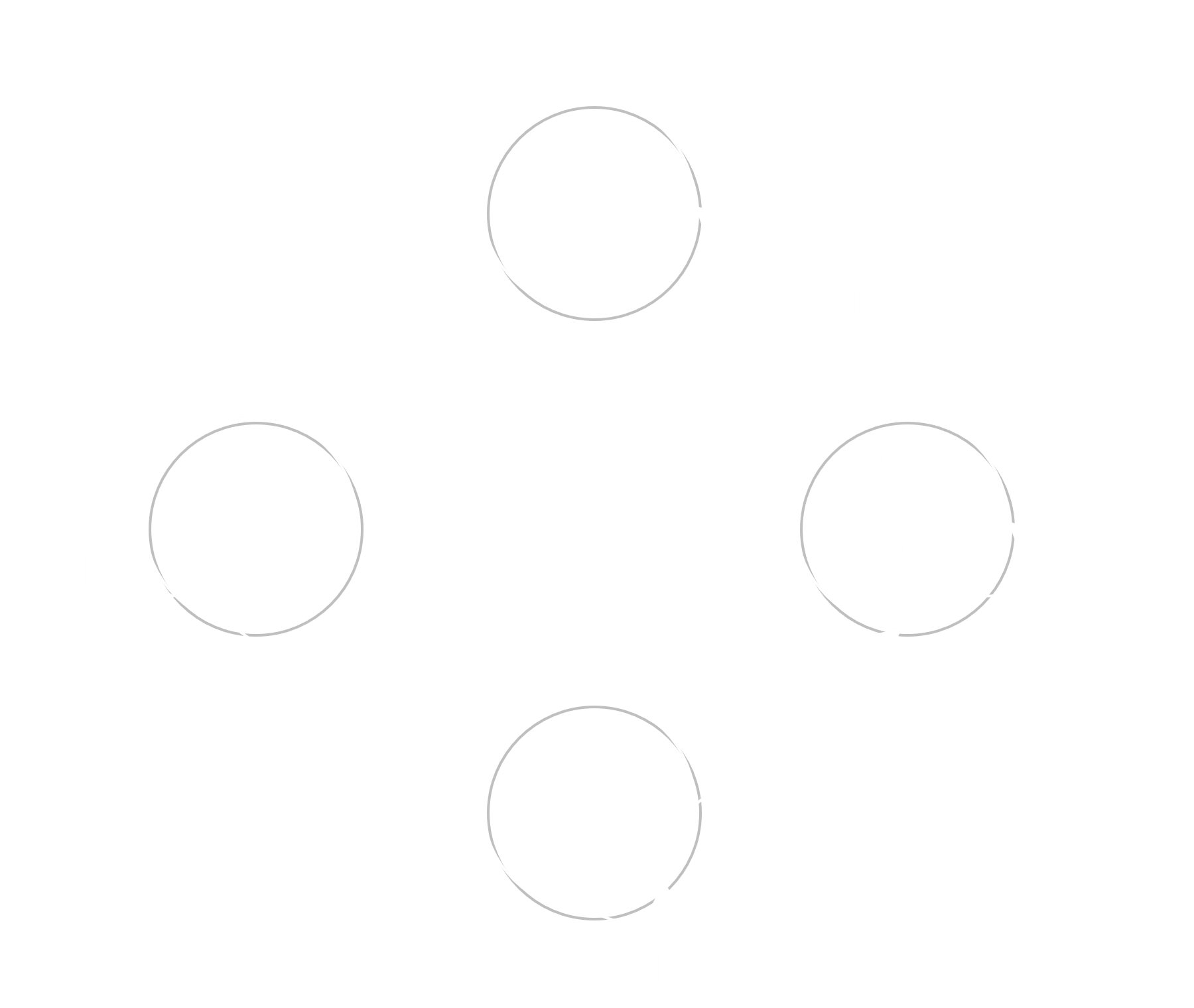
* Specify the problem.
* Create a state diagram from the problem.
* Create a state table.
* Create a reduced state diagram.
* Determine the number and type of flip flops required.
* Create circuit excitation table.
* Create output and flip flop equations.
* Create a circuit Diagram.

The steps will be followed to design the next circuit.

For a given problem, say the state diagram looks like this:



Here, notice that both outputs from any given state are the same, i.e. the outputs from the state are both , regardless of the input, the outputs from the state are both and so on. In such a scenario, the state diagram can also be drawn like this:



State Table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Present State | | Next State | | | |  | |
|  | |  | |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

We can see that the problem requires flip flops. Let us assume that it is demanded that the circuit be created using flip flops.

Now we must create a circuit excitation table. This is nearly the same as the state table, but also adds columns for determination of the inputs to the flip flops required for a particular set of next states to be achieved.

Circuit Excitation Table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Present State | | Input | Next State | | Flip Flop Inputs | | Output |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

From here, we can determine the equations for the inputs to the flip flops and the output of the circuit.

Thus, the circuit diagram:

T

Q

Q

T

Q

Q

CLK

X

Y

A

B

An important thing to note is that for the given circuit, the output is completely independent of the input , and is only dependent on the current states and . A circuit giving such an output is know as a Moore State Machine. A circuit where the output is dependent on both the present states and on the inputs is called a Mealy State Machine.

## Sequential Circuits – Analysis Procedure

While analyzing a sequential circuit, we go backwards, from the circuit, to equations for the inputs to the flip flops and the circuits output, to the circuit excitation table and finally the state diagram.

Consider the following circuit:

CLK

X

Y

A

B

D

Q

Q

D

Q

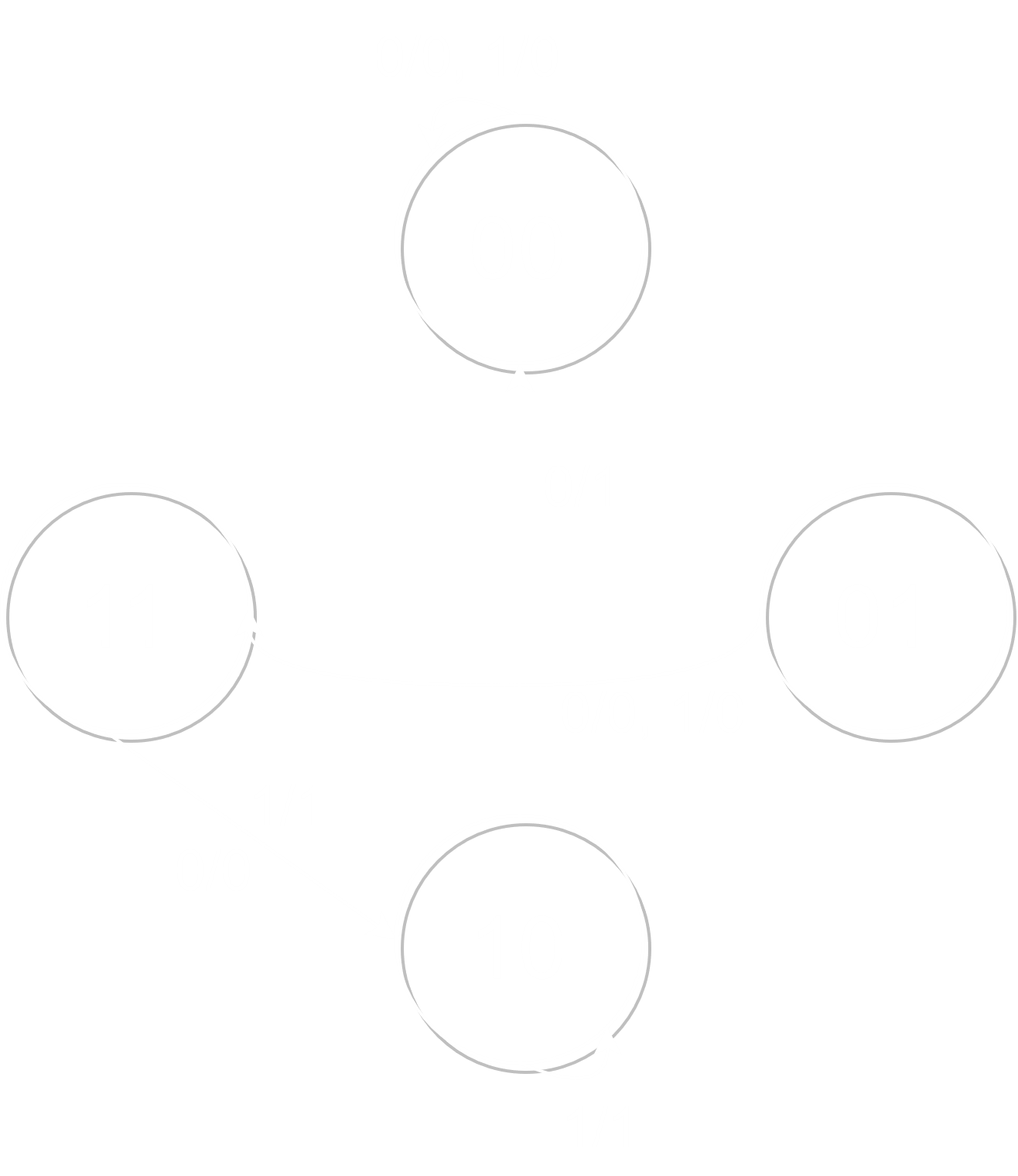
Q

Here,

Circuit Excitation Table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

State Diagram:

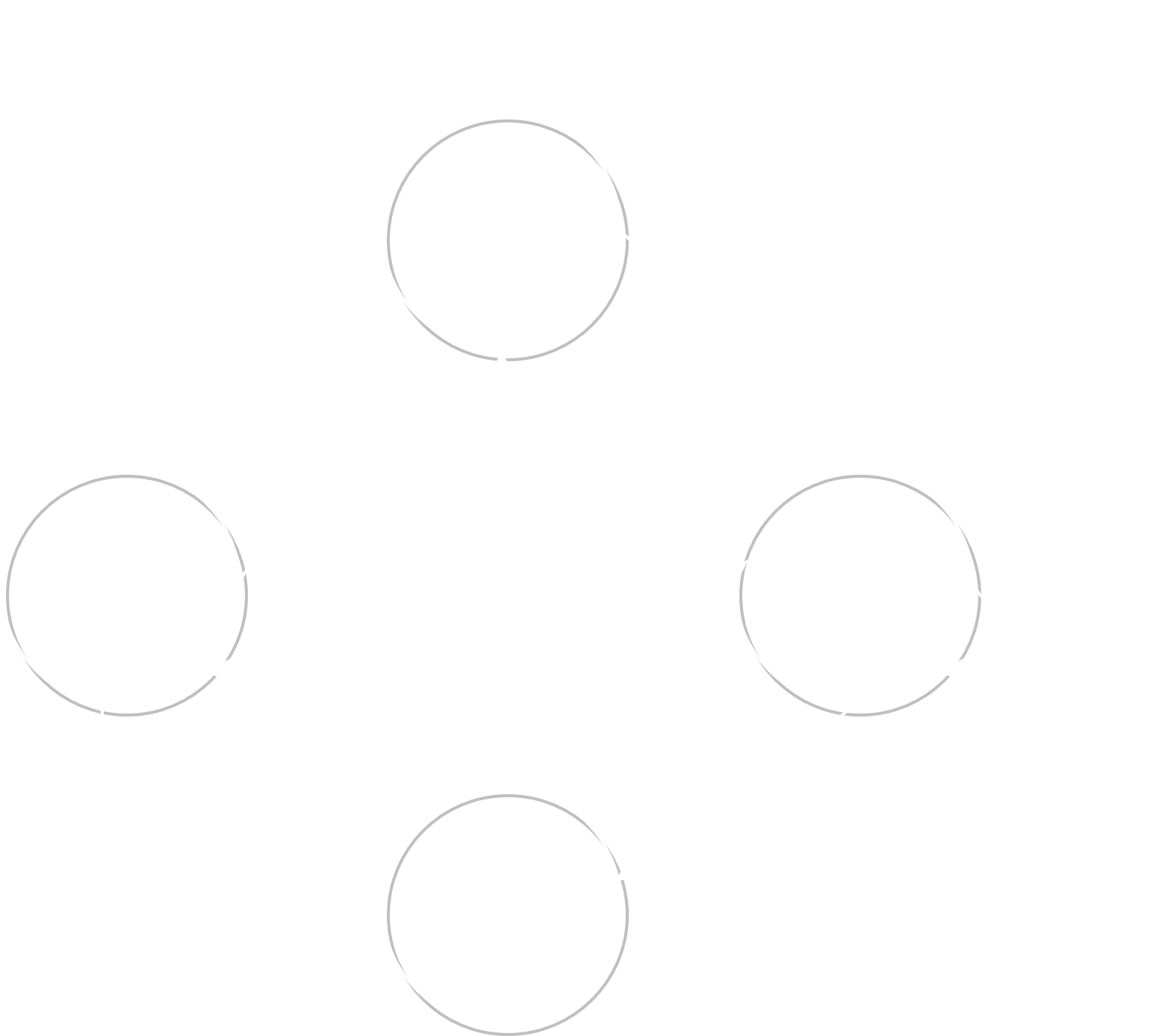


## Sequence Detector

We will now design a sequence detector using the design procedure. The circuit must determine a particular sequence ‘’ from a pattern . Starting from , the output will be . This is because when the circuit detects the fifth input , it finds that the two inputs immediately before it are and respectively. Thus, the fifth output is . The same thing happens for the 11th and 13th output.

This pattern is for a circuit that ignores overlaps. What this means is, for the 13th output, the first of the sequence detected is the last from the sequence detected by the 11th output. This is an overlap. For a circuit that will only accept non-overlapping outputs, .

For the particular sequence we are trying to detect (the diagram for a different sequence will be different), the state diagram looks like this:



The diagram tells the circuit to remain in , the reset state, until it detects a is detected. When a is detected, the circuit goes to . If the next input is , the circuit goes on to , since the pattern is being continue, but if the input is , the circuit goes back to , resetting itself. From , if the next input is , the circuit goes to . Otherwise, it goes back to . At , the function gets a little more complicated. If the next input is , then the sequence is obviously broken (), but it could be the beginning of another sequence, so the circuit goes to . If the next input is , the circuit goes to , since the previous input was and the current input is . If the next input is , then the sequence will be matched again and the circuit will go from to .

The outputs given along with each state shows that for every input, if the circuit reaches , or , it will give an output of . If the circuit reaches , it will give an output of .

Since there are 4 states, we will be requiring 2 flip flops. To keep things simple, D Flip Flops are being used. The state table will thus look like this. Again, the state table is specific to the particular sequence we are trying to detect.

State Table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Present State | | | Input | Next State | | | Output |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

The following K Maps are drawn up for , , and :

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

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

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

Thus,

Remember that the next states and are obtained through a combination of the current input and the current states and . This means that the output from the flip flops currently are and and the next states will be obtained by sending the current outputs back to the input.

Circuit Diagram:

CLK

X

Y

A

B

D

Q

Q

D

Q

Q

If any other flip flops are used instead of a D Flip Flop, the process becomes more complicated since the equations for the flip flop inputs cannot be directly derived from the state table. An excitation table must also be drawn up. Such a process is shown later in the design procedure for a 2-bit synchronous up counter, in Chapter 6.