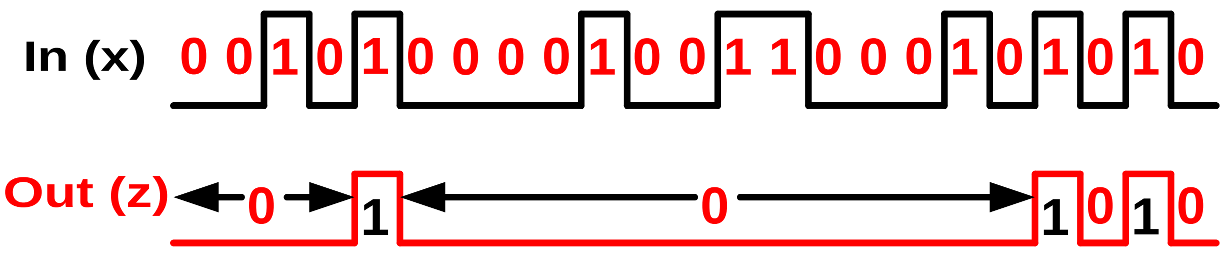
Sequence Detectors

A special type of state machine is the **Sequence Detector**. A sequence detector looks for some kind of pattern in a pulse stream.

EXAMPLE: **Let's observe a bit stream on a wire. They are observed from left to right. For example, the 1st bit seen is a 0, followed by a 0, followed by a 1, and then a 0, etc, etc.** Design a machine which will **output a 'logic 1'** when it observes a **101 sequence**. The machine may **overlap** sequences.



The

A state diagram isn't always necessary in a lot of state machine designs. However, it is **<u>absolutely necessary sequence detector design</u>** for reasons which will become obvious. You might say that it is the "**key**" to the design.

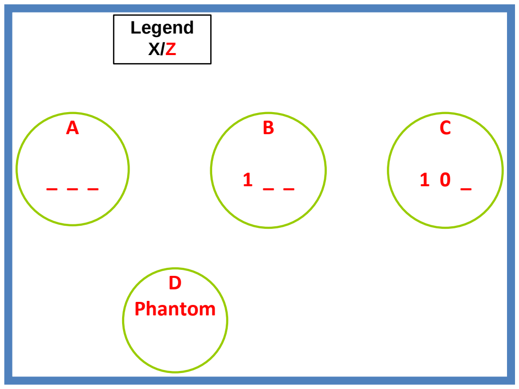
First, the number of states must be determined. **The number of states in a sequence <u>equals</u> the "length of the pattern you are detecting."** For this example, the detector is looking for a **"1 0 1" sequence**, therefore, we need 3 states.

Q: **How many flip-flops will you need in the circuit?**
Ans: **Two**

Q: **How many states do 2 flip-flops produce?**
Ans: **Four**

Even though we only need 3 states, we still have to take the 4th state into account. This 4th state is known as a **Phantom state**.

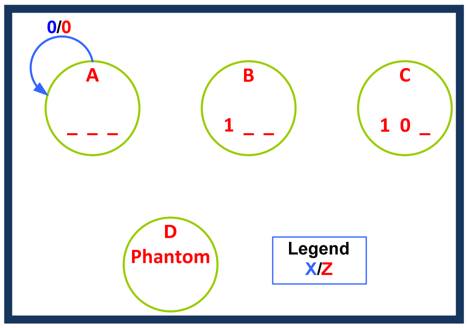
- To start off the design of the state diagram, the **4 states** are shown inside state circles as shown to the right.
- Note that the diagram has a legend. **All diagrams must have a legend** indicating the meaning of the numbers on the wires which will connect the states.
    - Within the legend will be the variables for the input and output variables.
        - The input variable vector will always be on the left side of the **"/"** (Sequence detectors ALWAYS have an input variable).
        - The output variable vectors will be on the right side the **"/"**.



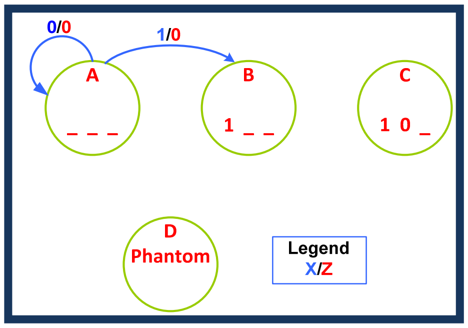
- Note the blanks and numbers inside the state circles. They are placed there as a **reminder** of what **must have happened**(in the immediate past) for the machine to be in the current state.
    - For example, **state B** contains a **"1 _ _".** That means that the **last bit observed must have been a "1"** or the machine wouldn't be in state B. **State C** contains a **"1 0 _".** This means that the very **last TWO bits must have been a "1 0"** in order for the machine to be in state C. This will become clearer later.

- The phantom state(s) must always be included in the design (if they exist).
    - Even though the machine isn't supposed to ever be in those states, **a logic designer must always design for the unlikely.** The machine must know what to do if it ever gets into that Phantom state or it will just sit there.
    - Just as we did in earlier state machine designs, we call the phantom states "**illegal**" states. But this time, **we are constrained in where we can send them**. **In general, we send the phantom states to State A**. There can be exceptions to this which we will cover later.

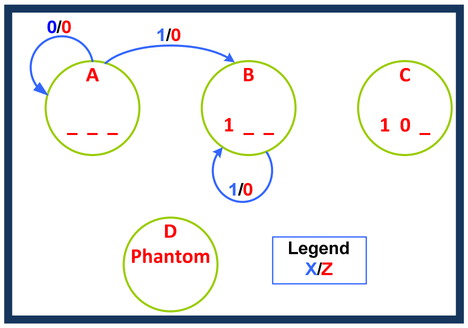
1. Start by observing the data stream for the desired sequence. The machine always starts in **state A** which has **no preconditions (note that the State A bubble is empty)**. When in **state A**, if a **"0"** is observed, it should **stay** in **state A**, and the output **(z)** should **remain low** since the **sequence hasn't been detected yet.**



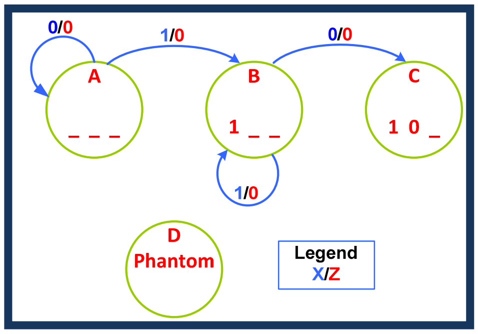
2. It will continue to **loop** within **State A** in this way until the 1st **'1'** of the sequence is detected. At this point, the machine will go to **state B** which has the **precondition that the "last bit detected had to be a '1'** (Note the '**1 _ _**' in **State B's** bubble). **The output (z) will remain low as shown to on the arrow leading to State B.**



3. Now, with the machine in **state B**, the **next bit** can either be a '**1**' or a '**0**'. If the **next bit** is a '**1**', we note that the **precondition for being in state B** is that the **LAST bit seen had to be a '1'**. That condition **is still met** so the machine will stay in **state B** and the **output will remain low (z)**.

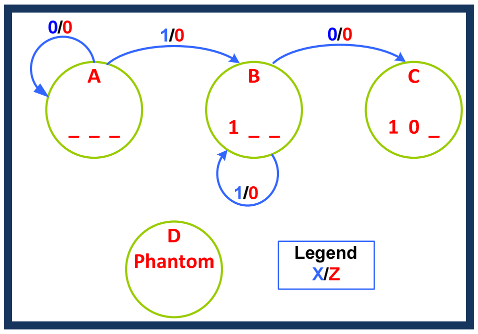


4. At this point, as soon as a **'0'** is detected, the machine will go to **state C** where the **precondition** is that the **LAST TWO bits detected MUST have been a '1 0'** which is true at this point.



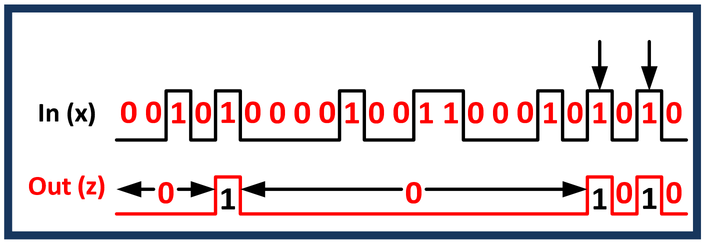
Design continued on the next page)

5. With the machine in **state C**, the **next bit detected** can either be a **0** or a **1**. Remember that we are looking for a **101** sequence and that in order to remain in **state C**, the **precondition that the "last TWO bits must have been a '1 0'"** MUST be met (as is demonstrated by the State C Bubbles contents).



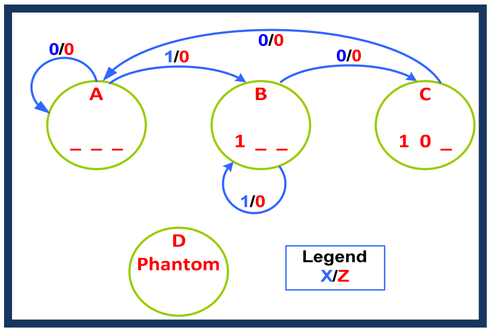
If the **next bit detected** is a **0**, that **precondition is no longer being met** so it needs to be determined to which state the machine will go. At this point we need to remember that the problem statement indicated that the **overlapping was allowed**. What does this mean?

Note the '**1**' pointed to by the first arrow in the input stream to the below.

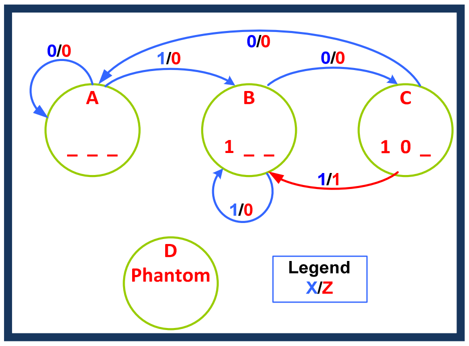


It is not only the **second '1'** in a desired sequence, but due to **overlap**, it is also the **first '1'** in the next occurring desired sequence. The same can be said for the '1' pointed to by the second arrow. ***At the end of this example we will look at the same state machine where overlapping isn't allowed and note the differences.***

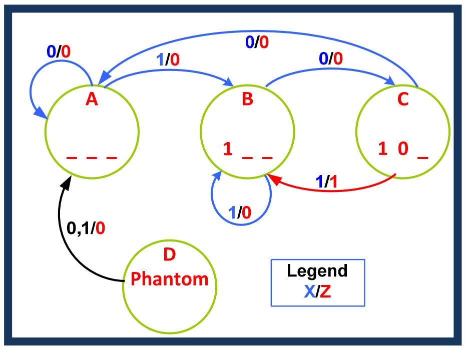
6. Ok, the machine is in **state C** and has just received a '0'. It can't stay in **state C since the precondition for being in state C, the last two bits detected must have been a 1 0, is no longer being met.** So, we need to determine just where it can go. The **preconditions** for **state B**, **the last bit seen must have been a 1**, hasn't been met either, so it can't go back to **state B**. The only thing left is for the machine to go back to **state A**, **no preconditions**, and **the output remains low.**



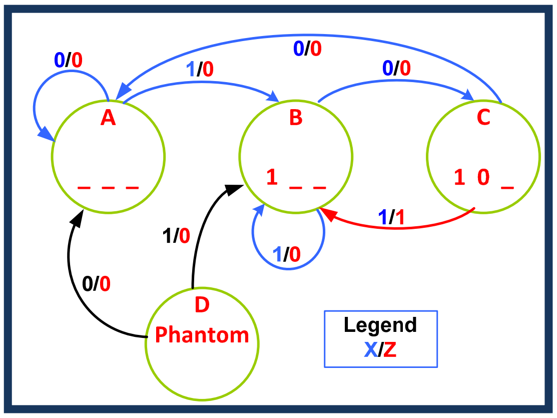
7. Finally, if the machine is in **state C** and the **next bit observed is a '1'**, it is noted that the **last two bits observed were 0 1** so it can't stay in **state C (with the required change in output) since the preconditions have not been met**. However, the **preconditions** for **state B**, (**last bit observed was a '1'**), have been met, so the machine can go to **state B** and the **output (z)** goes **high** since the **101 sequence was detected**.



8. Finally, we have to take the Phantom (**illegal**) state into account. We have sent the two different input possibilities from the phantom state to **state A**. In this case, a single arrow is used and the line contents have a **0,1** on the input **side indicating that in reality there are two arrows.**



9. Another way to handle this phantom state is to realize that if the machine is in state D and a 1 is received, the preconditions have been met to be in state B. So, it is allowed to do just that as is shown.



Why not always do this? Well, the decision to do this will be made in the design phase after the state table has been created. Whichever method results in the simplest Boolean expressions (just as we did earlier in the state machine tutorial) will be the method used.

Now that the state diagram has been created, the next step is the state table. Two different representations of the table will be shown below. The illegal states will be given in red.

You need to be able to both understand and create either of the state table representation.

Before we go any further in our design, we need to make a state assignment. Normally, the logic designer would make this assignment in an order which would minimize the circuit. This topic is covered in Chapter 9 and in EET 420 if we don't get to it in this course.

	y_1	y_0
A	0	0
B	0	1
D	1	0
C	1	1

y_1y_0	x=0 Y_1Y_0	x=1 Y_1Y_0	x=0 z	x=1 z
A	A	B	0	0
B	C	B	0	0
D	A	B	0	0
C	A	B	0	1

Input	Present State	Next State	Output
x	y_1y_0	Y_1Y_0	z
0	A	A	0
0	B	C	0
0	D	A	0
0	C	A	0
1	A	B	0
1	B	B	0
1	D	B	0
1	C	B	1

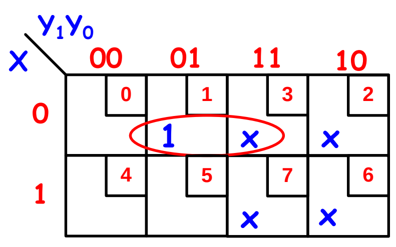
Now, let's look at the numbers in states B and C in the table. They represent what must have been found in the sequence in order to have been in that state. For instance, in order to be in state B, the last bit to be detected must have been a '1', "the 1st '1' in the sequence."

Input	Present State	Next State	Logic				Output
x	y_1y_0	Y_1Y_0	J_1	K_1	J_0	K_0	z
0	0 A 0	0 A 0	0	x	0	x	0
0	0 B 1	1 C 1	1	x	x	0	0
0	1 D 0	0 A 0	x	1	0	x	0
0	1 C 1	0 A 0	x	1	x	1	0
1	0 A 0	0 B 1	0	x	1	x	0
1	0 B 1	0 B 1	0	x	x	0	0
1	1 D 0	0 B 1	x	1	1	x	0
1	1 C 1	0 B 1	x	1	x	0	1

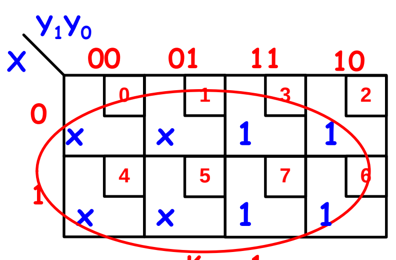
JK Transition Table

y	⇒	Y	J	K
0	⇒	0	0	x
0	⇒	1	1	x
1	⇒	0	x	1
1	⇒	1	x	0

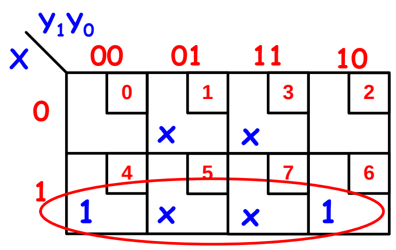
Note that the assigned state numbers have been added to the state variables. This enables us to add on the logic columns.



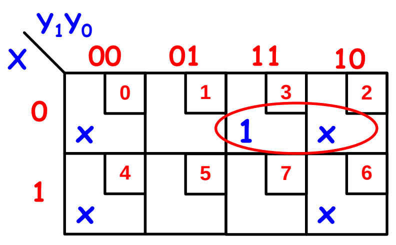
$J_1 = \overline{x}y_0$



$K_1 = 1$

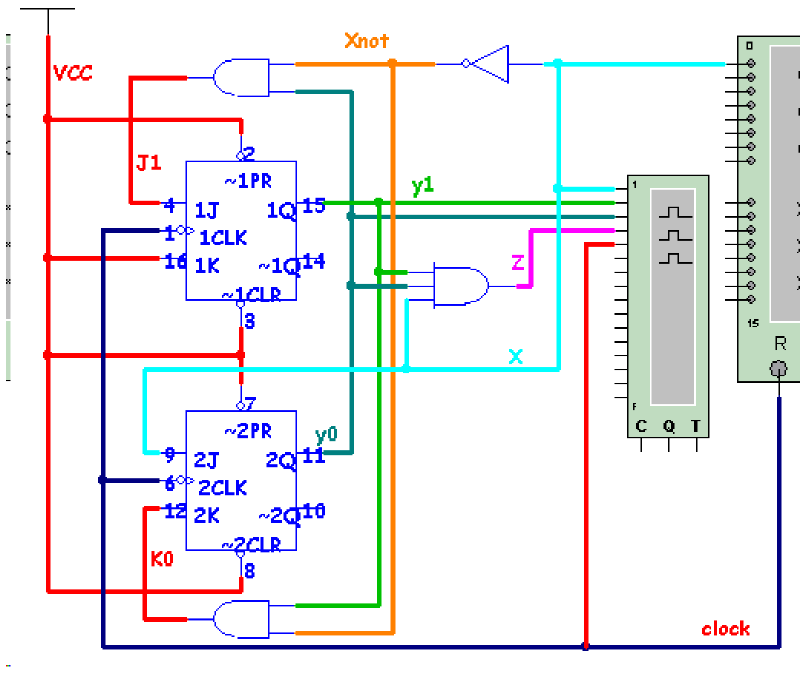


$J_0 = x$

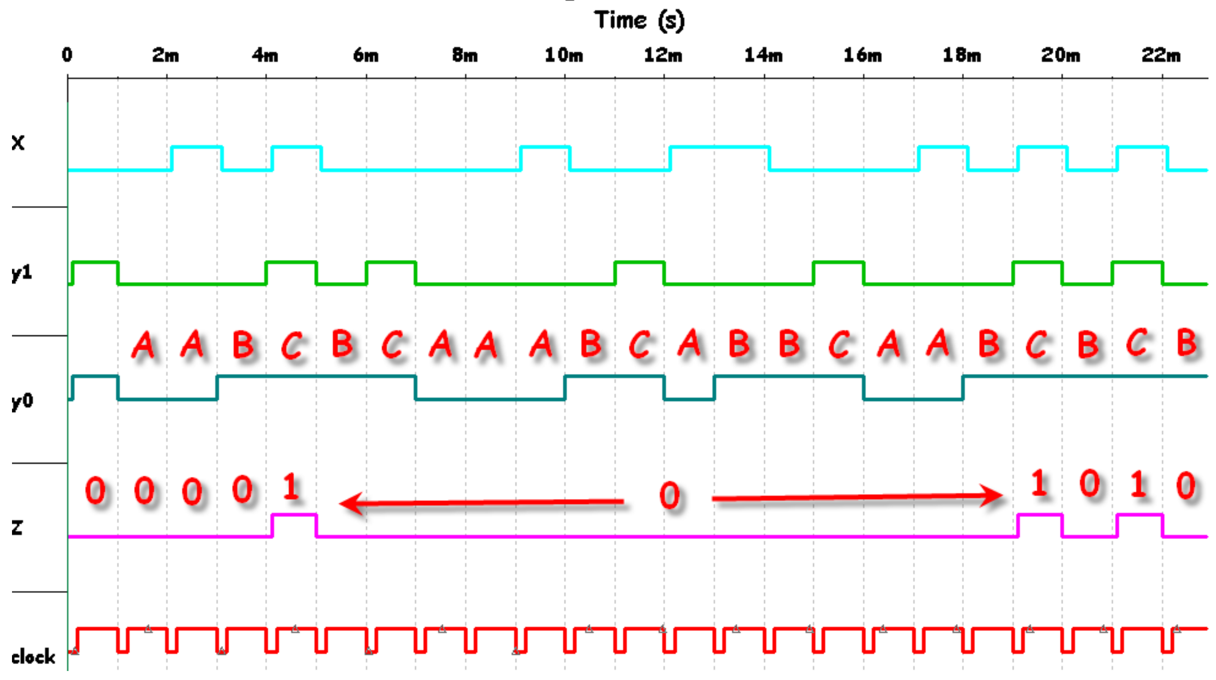


$K_0 = \overline{x}y_1$

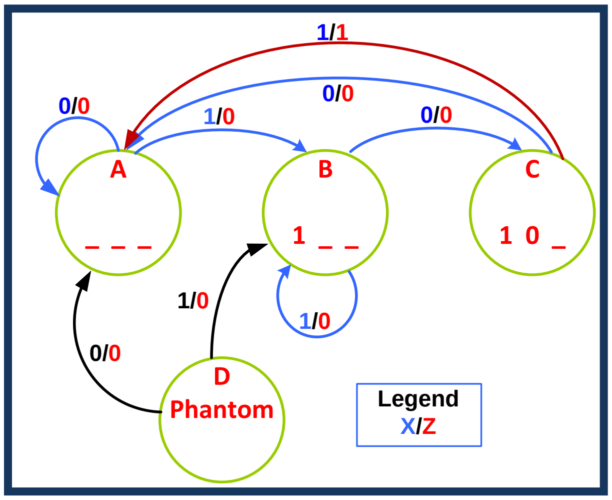
Circuit and Graph follow on the next page)


VCC
Xnot
J1
y1
~1PR
1J
1Q
1CLK
1K
~1Q
~1CLR
Z
X
~2PR
y0
2J
2Q
2CLK
2K
~2Q
~2CLR
K0
clock
C Q T
R

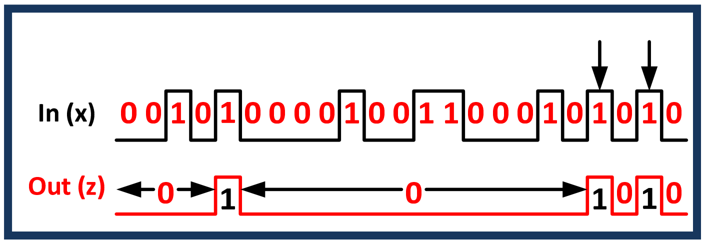
Figure_b_4


Time (s)
0
2m
4m
6m
8m
10m
12m
14m
16m
18m
20m
22m
X
y1
A A B C B C A A A B C A B B C A A B C B C B
y0
0 0 0 0 1
0
1 0 1 0
Z
clock

Earlier we discussed the subject of 'OVERLAPPING'. Now, let's take a look at what the state diagram will look like if OVERLAPPING is not allowed.



Note that the last arrow (in red) went all the way back to State A vice being able to stop a State B since we are not allowed to let the last one of the last sequence be the 1st one of the next sequence.


In (x)
0 0 1 0 1 0 0 0 0 1 0 0 1 1 0 0 0 1 0 1 0 1 0
Out (z)
0
1
0
1 0 1 0