IMPLEMENTATION OF 4 BIT SYNCHRONOUS SEQUENTIAL UP/DOWN COUNTER USING JKFF AND 7-SEGMENT DISPLAY.

Javohirbek Xatamov
student at Inha university in Tashkent,
j.xatamov@student.inha.uz
https://doi.org/10.5281/zenodo.13375178

ARTICLE INFO

Qabul qilindi: 20-August 2024 yil
Ma'qullandi: 23- August 2024 yil
Nashr qilindi: 26- August 2024 yil

KEYWORDS

Digital counter, synchronous counter, circuit design, karnaugh maps, Logisim, sequential logic.

ABSTRACT

This paper presents the design and implementation of a 4-bit synchronous sequential up/down counter using JK flip-flops. The counter operates in both ascending and descending modes, displaying output on two 7-segment displays. The design process includes state diagram creation, JK state table derivation, and Boolean expression optimization using Karnaugh maps. Implemented in Logisim, the final circuit demonstrates a functional counter capable of displaying hexadecimal values from 0 to F. This work illustrates the effective use of JK flip-flops in synchronous counter design for digital applications.

Introduction. Digital counters are fundamental building blocks in various electronic systems, serving applications ranging from digital clocks to complex microprocessor operations. Among these, synchronous sequential counters are particularly valuable due to their predictable timing characteristics, which result from all flip-flops being triggered simultaneously by a common clock signal. This paper presents the implementation of a 4-bit synchronous sequential up/down counter using JK flip-flops (JKFF). The counter is capable of counting in both ascending and descending order. The result is to be outputted on two 7-segment displays.

The design of synchronous counters using JK flip-flops is of particular interest due to the inherent advantages of JKFFs, such as toggling capabilities and efficient state transitions, which reduce the risk of race conditions. This study outlines the theoretical framework, design process, and practical implementation of the counter in the Logisim app. Design process did not neglect cost effectiveness, however, it did not focus on it.

FSM states. First, since there are two counting modes (up and down), separate state diagrams should be created for each mode(up and down). The selection between these modes is based on the controlling input(w), when w = 0 the counter counts down, and when w = 1 it counts up. It is worth mentioning that a binary system is used in both tables as it is necessary to use binary numbers to work on hardware level which only works on binary numbers. Consequently, there are two state tables w = 1 is shown in Figure 1, w = 0 is illustrated in Figure 2.

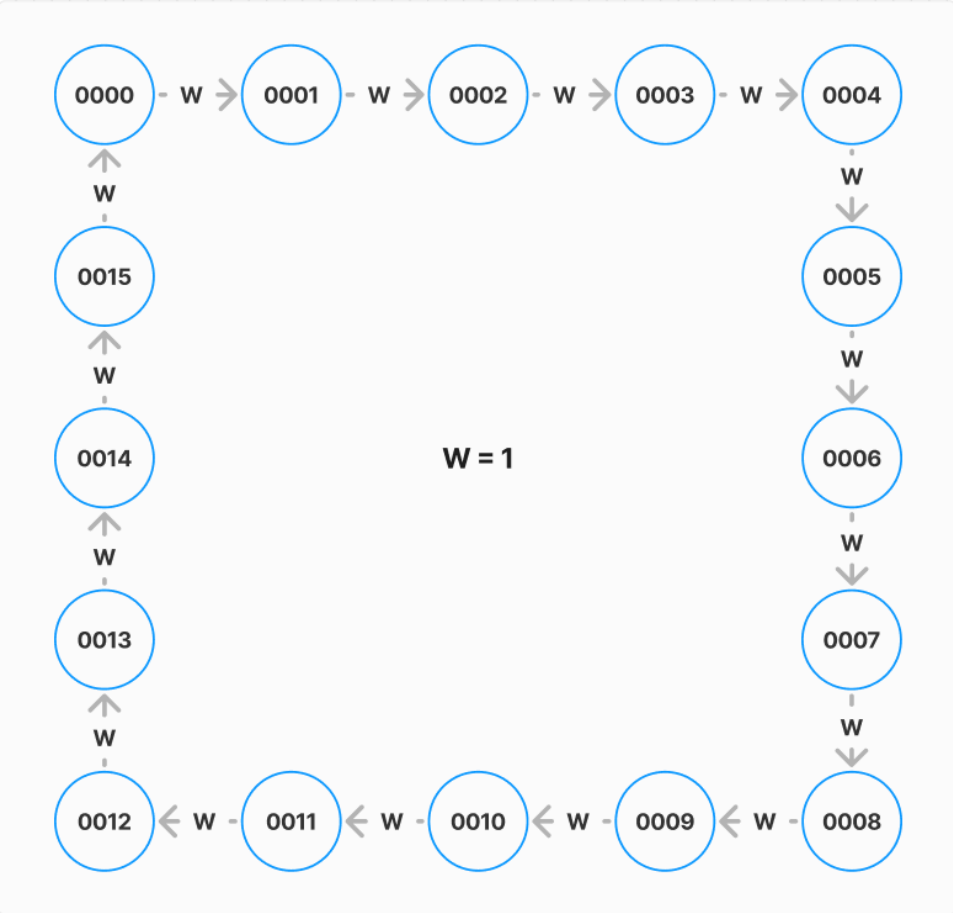

Figure 1

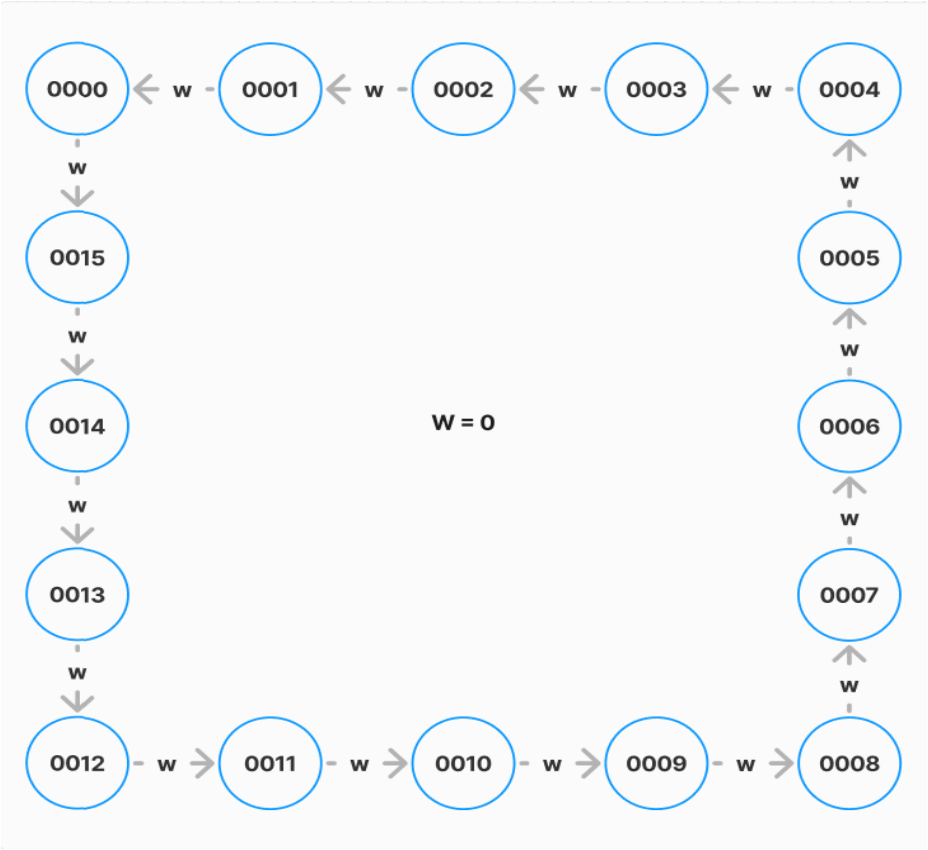

Figure 2

State table derivation. As the state diagrams are present, state tables can be built based on them. In these tables numbered "y" letters are the representations of each bit in a binary number, small letter "y"s are for the current state and capital letter "Y"s are for the next state. The one below is for w=0 which sets the circuit to count down, i.e. all values' next state is their decrement, except for the first value which changes 0000 to 1111 this is done as our counter counts in loops(Figure 3).

Note: Values has nothing to do with each other vertically, each value represents independent particular state from 0 to 15 and on the right it is shown what will be the next state if w is 0 or 1, in the first case the next state will be less by 1, in the second more by 1

Current State				Next state (w=0)			
y3	y2	y1	y0	Y3	Y2	Y1	Y0
0	0	0	0	1	1	1	1
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	1
0	0	1	1	0	0	1	0
0	1	0	0	0	0	1	1
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	1
0	1	1	1	0	1	1	0
1	0	0	0	0	1	1	1
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	1
1	0	1	1	1	0	1	0
1	1	0	0	1	0	1	1
1	1	0	1	1	1	0	0
1	1	1	0	1	1	0	1
1	1	1	1	1	1	1	0

Figure 3

Next, the state table for the up counter is similar, the only difference from the previous table is that for every value of the current state, the next state will be incremented.

Current State				Next state (w=1)			
y3	y2	y1	y0	Y3	Y2	Y1	Y0
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	1	0	1	0
1	0	1	0	1	0	1	1
1	0	1	1	1	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	0	0	0

Figure 4

JK state table derivation. First, here is the table of JK flip flop in Figure 5.

J	K	Q
0	0	Q
0	1	0
1	0	1
1	1	~Q

Figure 5

Q is the state. Some of its properties: if J and K are both 1, then the next state will be the opposite of the current one, e.g. if the current value of the state is 1, next is 0 or vice versa. If J and K are both 0, then the next state will stay the same. In the above tables in Figure 3 and 4, the values of the current and the next state are given, so the values of J and K can be reverse engineered.

As the values of y and Y are basically current and next state, they can be used to determine the values of J and K, below is the table which will be used for this (Figure 6)

Q(t)	Q(t+1)	J	K
0	0	0	d
0	1	1	d
1	0	d	1
1	1	d	0

Figure 6

Explanation for some of these properties: when the current and the next state is 0, it means even after the values of J or K changed the state stayed the same, according to the table in Figure 5, there are two possibilities, either J and K were 0 and 1, which cased the state to become 0, but it were initially 0, so nothing changed, or J and K were 0's which causes the state to stay the same. So, J and K are either 0 0 or 0 1, in other words, J is definitely 0 while K is correct in both cases so it doesn't matter, that's why K is "d" which stands for "doesn't matter". Because J and K work on the bit level they affect only the individual bits, which y and Y were used to mark. Furthermore, Based on the values in Figure 3 and 4, two tables will be created for w = 0(Figure 7) and w = 1(Figure 8) to represent four "Y"s and J&K values which potentially caused their value change.

w=0							
Y3		Y2		Y1		Y0	
J	K	J	K	J	K	J	K
1	d	1	d	1	d	1	d
0	d	0	d	0	d	d	1
0	d	0	d	d	1	1	d
0	d	0	d	d	0	d	1
0	d	d	1	1	d	1	d
0	d	d	0	0	d	d	1
0	d	d	0	d	1	1	d
0	d	d	0	d	0	d	1
d	1	1	d	1	d	1	d
d	0	0	d	0	d	d	1
d	0	0	d	d	1	1	d
d	0	0	d	d	0	d	1
d	0	d	1	1	d	1	d
d	0	d	0	0	d	d	1
d	0	d	0	d	1	1	d
d	0	d	0	d	0	d	1

Figure 7

w=1							
Y3		Y2		Y1		Y0	
J	K	J	K	J	K	J	K
0	d	0	d	0	d	1	d
0	d	0	d	1	d	d	1
0	d	0	d	d	0	1	d
0	d	1	d	d	1	d	1
0	d	d	0	0	d	1	d
0	d	d	0	1	d	d	1
0	d	d	0	d	0	1	d
1	d	d	1	d	1	d	1
d	0	0	d	0	d	1	d
d	0	0	d	1	d	d	1
d	0	0	d	d	0	1	d
d	0	1	d	d	1	d	1
d	0	d	0	0	d	1	d
d	0	d	0	1	d	d	1
d	0	d	0	d	0	1	d
d	1	d	1	d	1	d	1

Figure 8

K-map derivation. The above tables in Figure 7 and 8 are the values of J and K which were extracted from the state table. The next step would be to make karnaugh maps out of these J&K's in order to find minimum SOPs. There are two tables for J&K values containing the same Y's and this Y contains J and K tables, thus there should be four k-maps for every Y(different w's, J's and K's).[1]

One k-map as an example(Figure 9):

Y1 K1	y0			
y1y0\y3y2	00	01	11	10
00	d	d	d	d
01	d	d	d	d
11	1	1	1	1
10	0	0	0	0

Figure 9

The above table is for Y1 bit, and its K1 values shown in Figure 8, as it contains 16 values that can be broken into a 4 by 4 table, each slot of this table represents a value of different variable, there are 16 variables these variables are represented by 4 bits of y(y1,y2,y3,y4), vertical is y1y0, horizontal y3y2, when combined they make up values from 0000 to 1111. Main task is to find the maximum number of 1's and d's located in close proximity, group them and find the y bit which has only one value in this group, but the formed group of numbers' quantity must be divisible by 2. In this case y0 is always 1, this means the function of K1 is y0. Note: if there were other y bits they would be simply added into the function by "or" operand.

w=0				

Y0 J0	1			
y1y0\y3y2	00	01	11	10
00	1	1	1	1
01	d	d	d	d
11	d	d	d	d
10	1	1	1	1

Y0 K0	1			
y1y0\y3y2	00	01	11	10
00	d	d	d	d
01	1	1	1	1
11	1	1	1	1
10	d	d	d	d

Y1 J1	~y0			
y1y0\y3y2	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	d	d	d	d
10	d	d	d	d

Y1 K1	~y0			
y1y0\y3y2	00	01	11	10
00	d	d	d	d
01	d	d	d	d
11	0	0	0	0
10	1	1	1	1

Y2 J2	~y1*~y0			
y1y0\y3y2	00	01	11	10
00	1	d	d	1
01	0	d	d	0
11	0	d	d	0
10	0	d	d	0

Y2 K2	~y1*~y0			
y1y0\y3y2	00	01	11	10
00	d	1	1	d
01	d	0	0	d
11	d	0	0	d
10	d	0	0	d

Y3 J3	~y1*~y2*~y0			
y1y0\y3y2	00	01	11	10
00	1	0	d	d
01	0	0	d	d
11	0	0	d	d
10	0	0	d	d

Y3 K3	~y2*~y1*~y0			
y1y0\y3y2	00	01	11	10
00	d	d	0	1
01	d	d	0	0
11	d	d	0	0
10	d	d	0	0

Figure 10

w=1									
Y0 J0	1				Y0 K0	1			
y1y0\y3y2	00	01	11	10	y1y0\y3y2	00	01	11	10
00	1	1	1	1	00	d	d	d	d
01	d	d	d	d	01	1	1	1	1
11	d	d	d	d	11	1	1	1	1
10	1	1	1	1	10	d	d	d	d
Y1 J1	y0				Y1 K1	y0			
y1y0\y3y2	00	01	11	10	y1y0\y3y2	00	01	11	10
00	0	0	0	0	00	d	d	d	d
01	1	1	1	1	01	d	d	d	d
11	d	d	d	d	11	1	1	1	1
10	d	d	d	d	10	0	0	0	0
Y2 J2	y1*y0				Y2 K2	y1*y0			
y1y0\y3y2	00	01	11	10	y1y0\y3y2	00	01	11	10
00	0	d	d	0	00	d	0	0	d
01	0	d	d	0	01	d	0	0	d
11	1	d	d	1	11	d	1	1	d
10	0	d	d	0	10	d	0	0	d
Y3 J3	y2*y1*y0				Y3 K3	y2*y1*y0			
y1y0\y3y2	00	01	11	10	y1y0\y3y2	00	01	11	10
00	0	0	d	d	00	d	d	0	0
01	0	0	d	d	01	d	d	0	0
11	0	1	d	d	11	d	d	1	0
10	0	0	d	d	10	d	d	0	0

Figure 11

Notes on the above tables Figure and 11:

1) Areas marked with blue are the grouped values.

2)In this case, "*" = "multiplication" means "AND", "+" = "addition" means "OR" and "~" = "negation" means "NOT".

3)y1y0 are for columns and y3y2 are for rows.

Calculation of the functions' minimum SOPs. Furthermore, as all the SOP functions were found from K-map, it is time to combine. This is done by multiplying functions by w(or ~w if functions are from tables where w=0), and then adding the same J's or K's together.

$$J_0 = 1$$
$$K_0 = 1$$
$$J_1 = \neg w * \neg y_0 + y_0 * w = \neg(\text{xor}(y_0, w))$$
$$K_1 = \neg w * \neg y_0 + y_0 * w = \neg(\text{xor}(y_0, w))$$
$$J_2 = \neg w * \neg y_0 * \neg y_1 + w * y_0 * y_1$$
$$K_2 = \neg w * \neg y_0 * \neg y_1 + w * y_0 * y_1$$
$$J_3 = \neg w * \neg y_0 * \neg y_1 * \neg y_2 + w * y_0 * y_1 * y_2$$
$$K_3 = \neg w * \neg y_0 * \neg y_1 * \neg y_2 + w * y_0 * y_1 * y_2$$

Figure 12

Figure 12 also contains further simplification.

Circuit Implementation. Functions are known, so the next step is to build a circuit. It is shown below(Figure 13), CLK is clock, necessary for JK flip flops, constant '1' located in bottom left represents input into the J0K0 based on its function, Logisim app is used for implementation[2]:

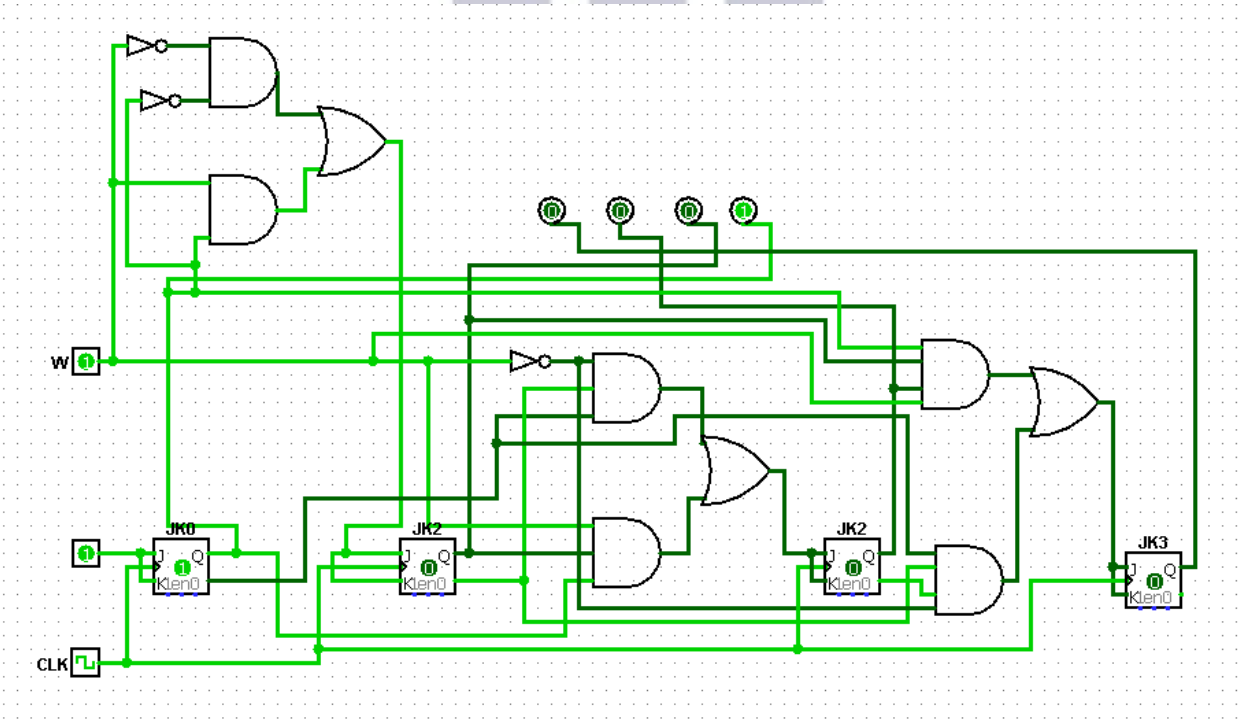

Figure 13

The above circuit's four output displays increment or decrement by 1 based on the value of w counting from 0000 to 1111 or vice versa and then start over in a loop.

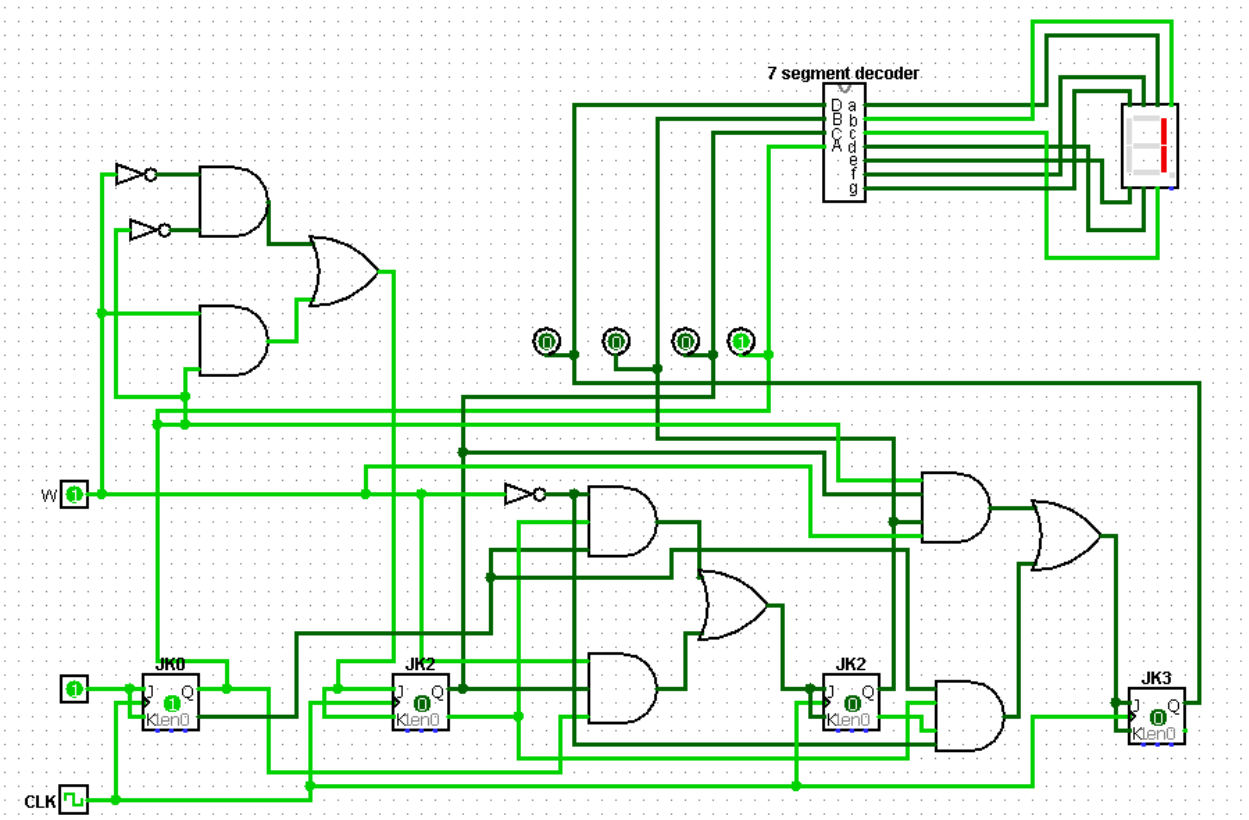

Figure 14

Figure 14 is the same as the circuit in Figure 13 but contains 2 additional components: 7-segment display and a decoder, necessary to decode BCD into hex[3].In the end, final circuit's display can count from 0 to F, F representing 15.

Conclusion. This paper presents the design and implementation of a 4-bit synchronous sequential up/down counter using JK flip-flops (JKFFs). By leveraging the toggling capabilities and efficient state transitions of JKFFs, we achieved a reliable and versatile counter capable of counting in both ascending and descending order. The design process involved creating distinct state diagrams for the up and down counting modes, followed by the derivation of state tables for both modes. These state tables were instrumental in determining the JK inputs required for proper counter operation. Through the use of Karnaugh maps, we optimized the Boolean expressions for the JK inputs, ensuring minimal logic and accurate functionality. The practical implementation of the counter was conducted using the Logisim app, where the counter's operation was verified through simulation. The final design integrates two 7-segment displays, allowing the counter to output values in hexadecimal format from 0 to F. This implementation demonstrates the practical application of theoretical concepts and validates the effectiveness of the proposed design. In summary, the paper highlights the efficiency of JK flip-flops in synchronous counter design and provides a comprehensive approach to implementing a functional 4-bit up/down counter. The successful integration of the counter with 7-segment displays underscores the utility of this design in various digital applications.

References:

1. David Money Harris and Sarah L. Harris, “Digital Design and Computer Architecture”, pp. 75-87.
2. Carl Burch, “Logisim”, https://sourceforge.net/projects/circuit/ .
3. Texas Instruments, “SN54LS47 BCD to 7-Segment Decoder/Drivers”, https://www.ti.com/lit/ds/symlink/sn54ls47.pdf?ts=1724371511246&ref_url=https%253A%252F%252Fwww.google.com%252F .