EET 310

Counters

Counters

- A counter is made up of a series of FF's (usually of the same type) which are designed to pass thru a sequence of output states. Sometimes there are one or more inputs associated with the counter as well.

Counters

- A counter is made up of a series of FF's (usually of the same type) which are designed to pass thru a sequence of output states. Sometimes there are one or more inputs associated with the counter as well.

- The count is determined by the # of states in the sequence with the upper limit set by:

$$2^n$$

- where 'n' is the number of FF's.

Ripple Counters

- A **Ripple Counter** is an **Asynchronous Counter.**

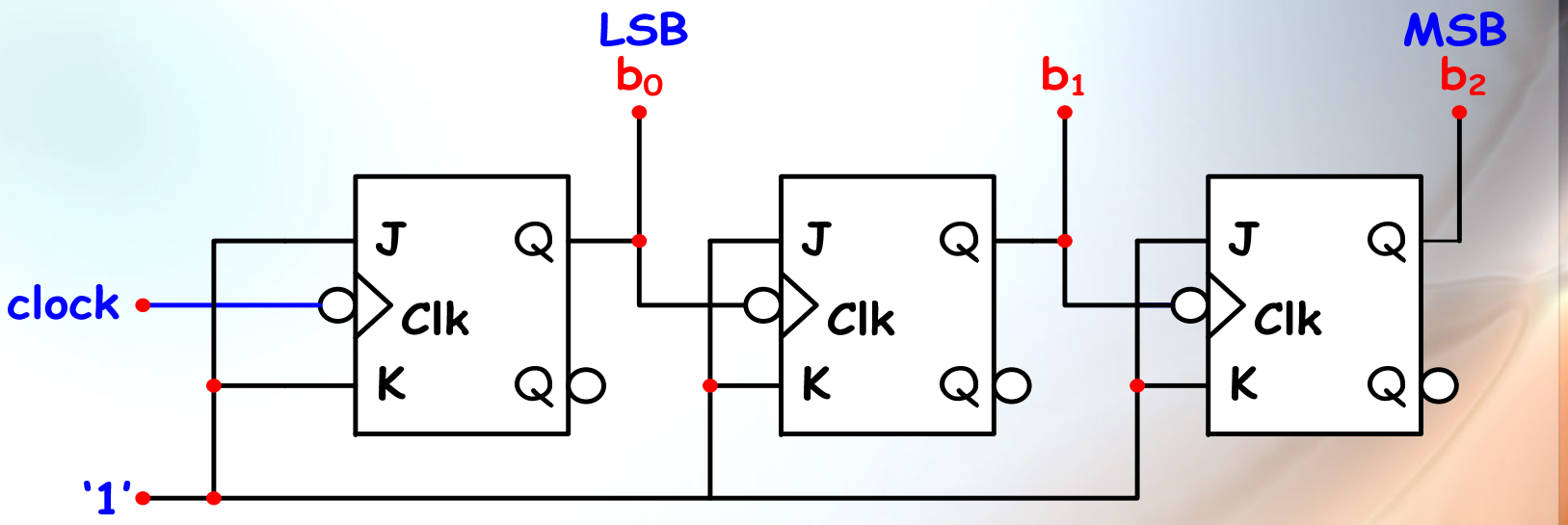
Ripple Counters

- A **Ripple Counter** is an **Asynchronous Counter.**
- An **Asynchronous Counter** is one where **only the first FF is clocked by the system clock**.

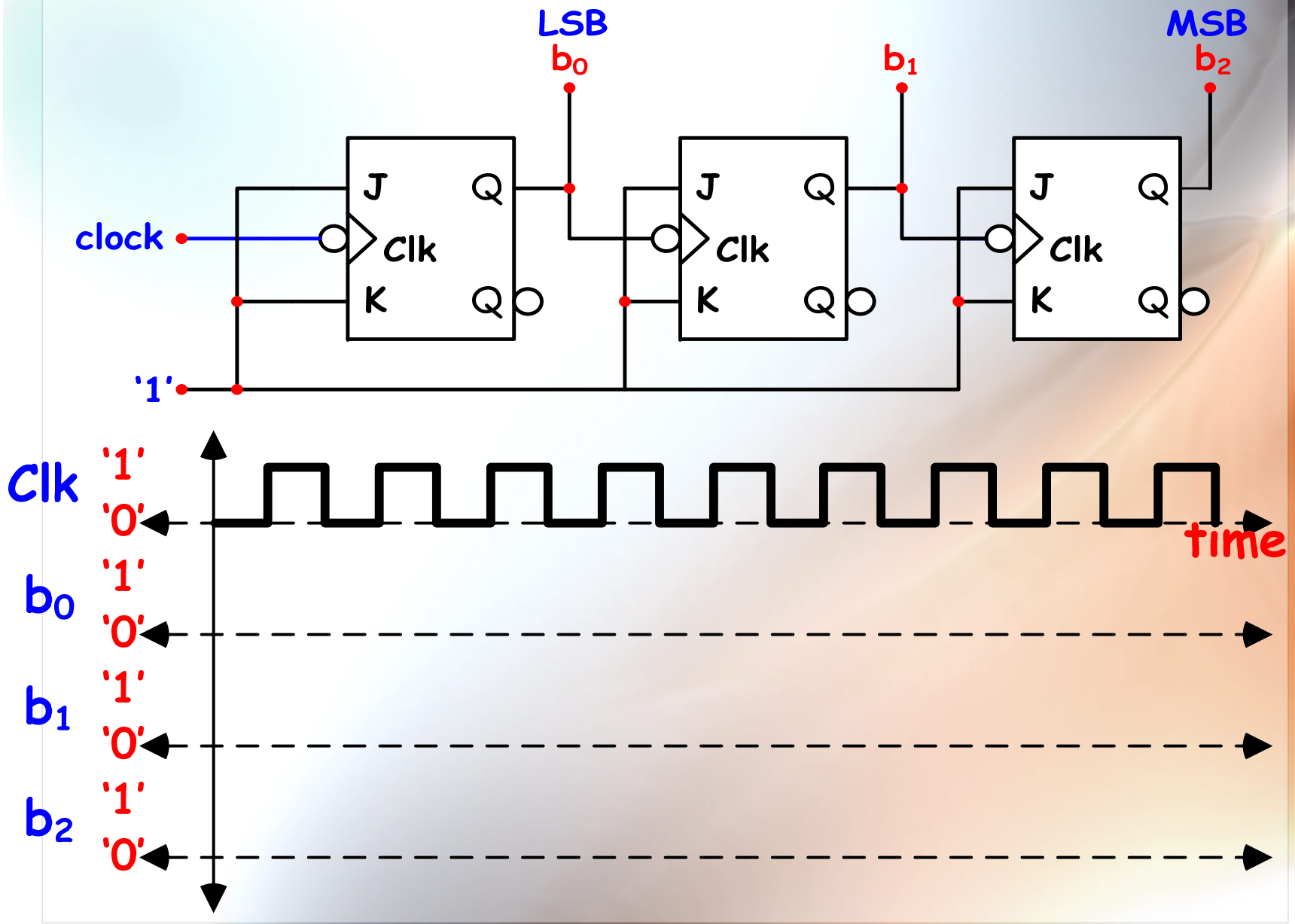
Ripple Counters

- A **Ripple Counter** is an **Asynchronous Counter.**
- An **Asynchronous Counter** is one where **only the first FF is clocked by the system clock.**
- The **next FF is clocked by the output of the previous FF**, and so on and so on……

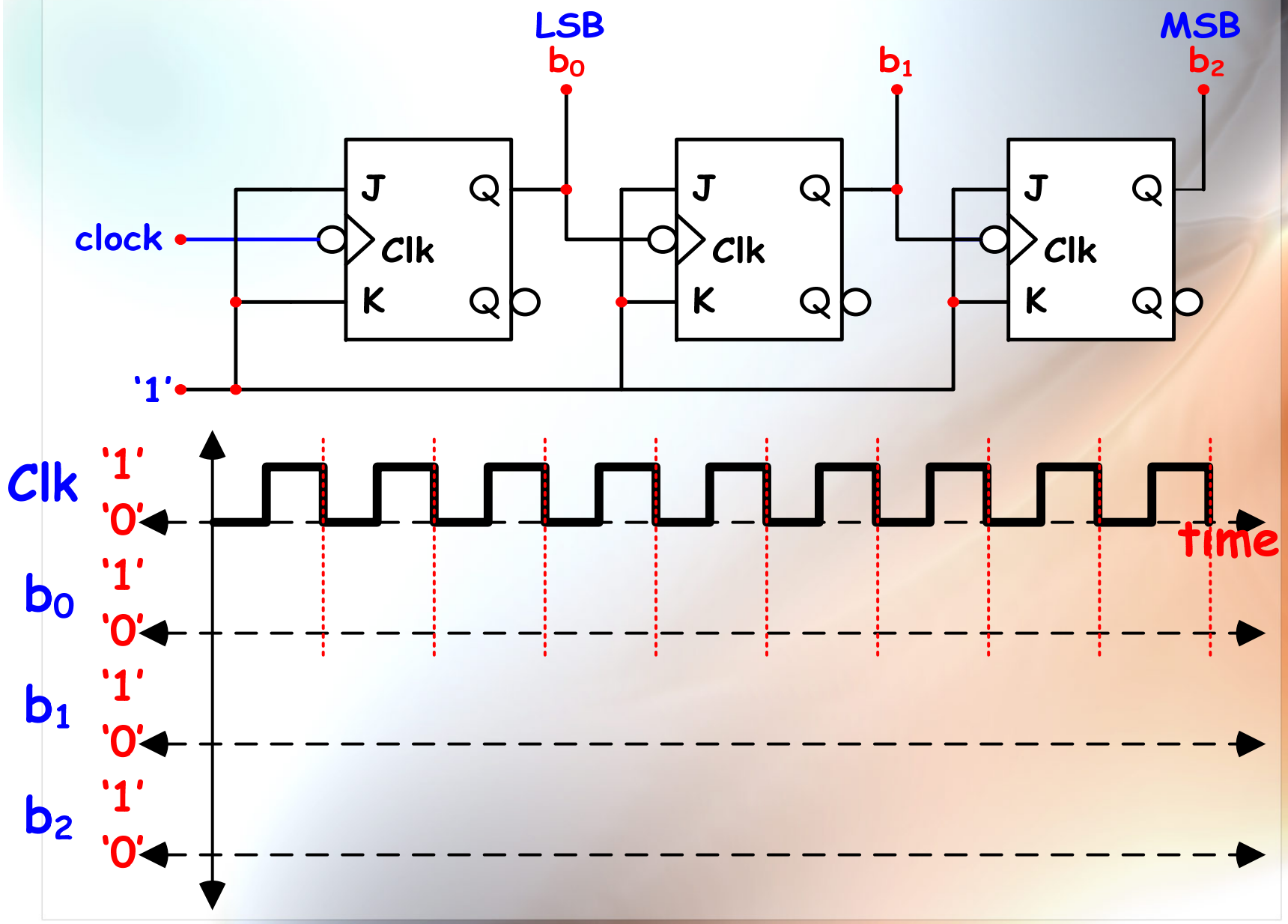
Ripple Counter Timing



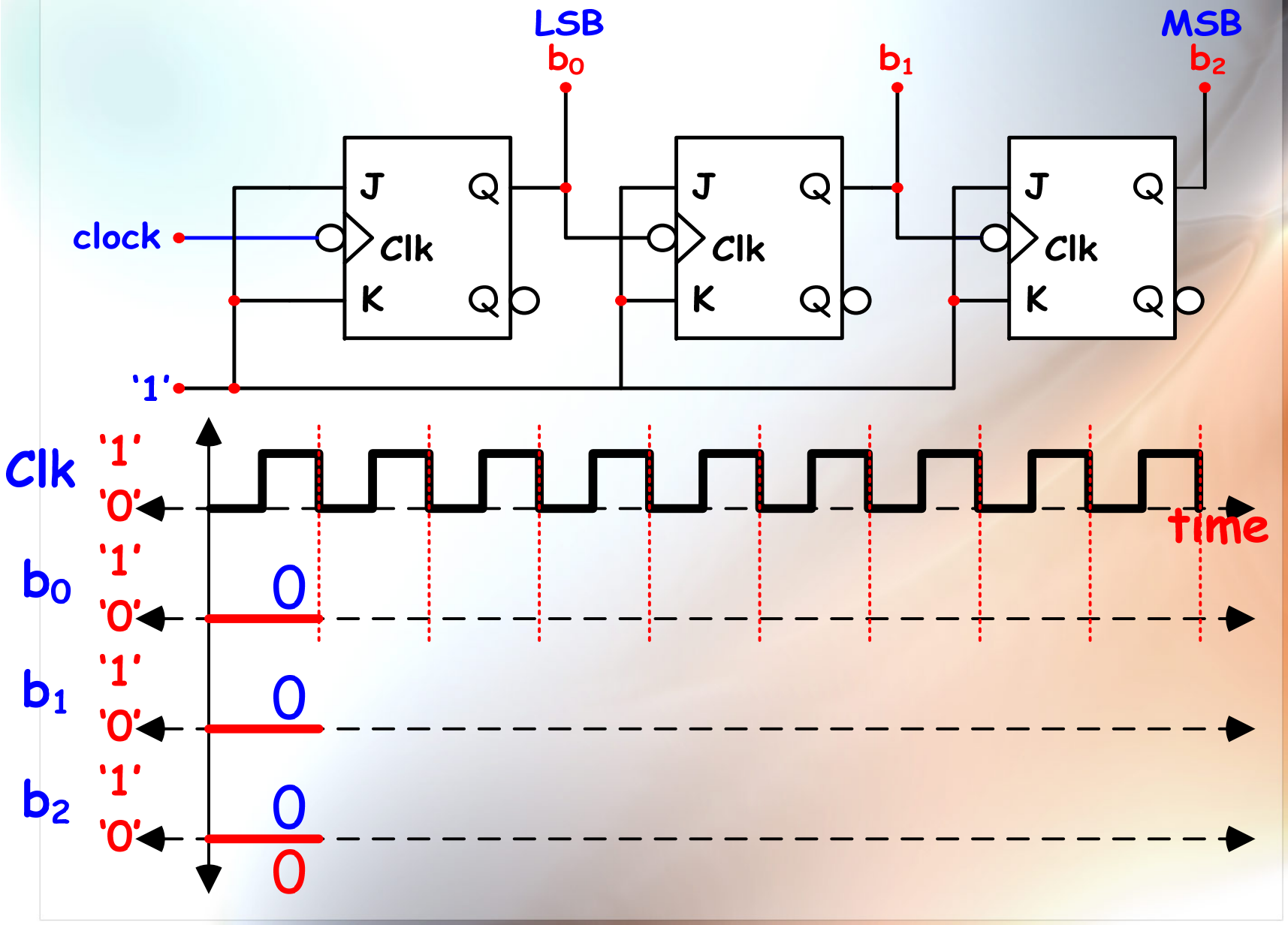
Ripple Counter Timing



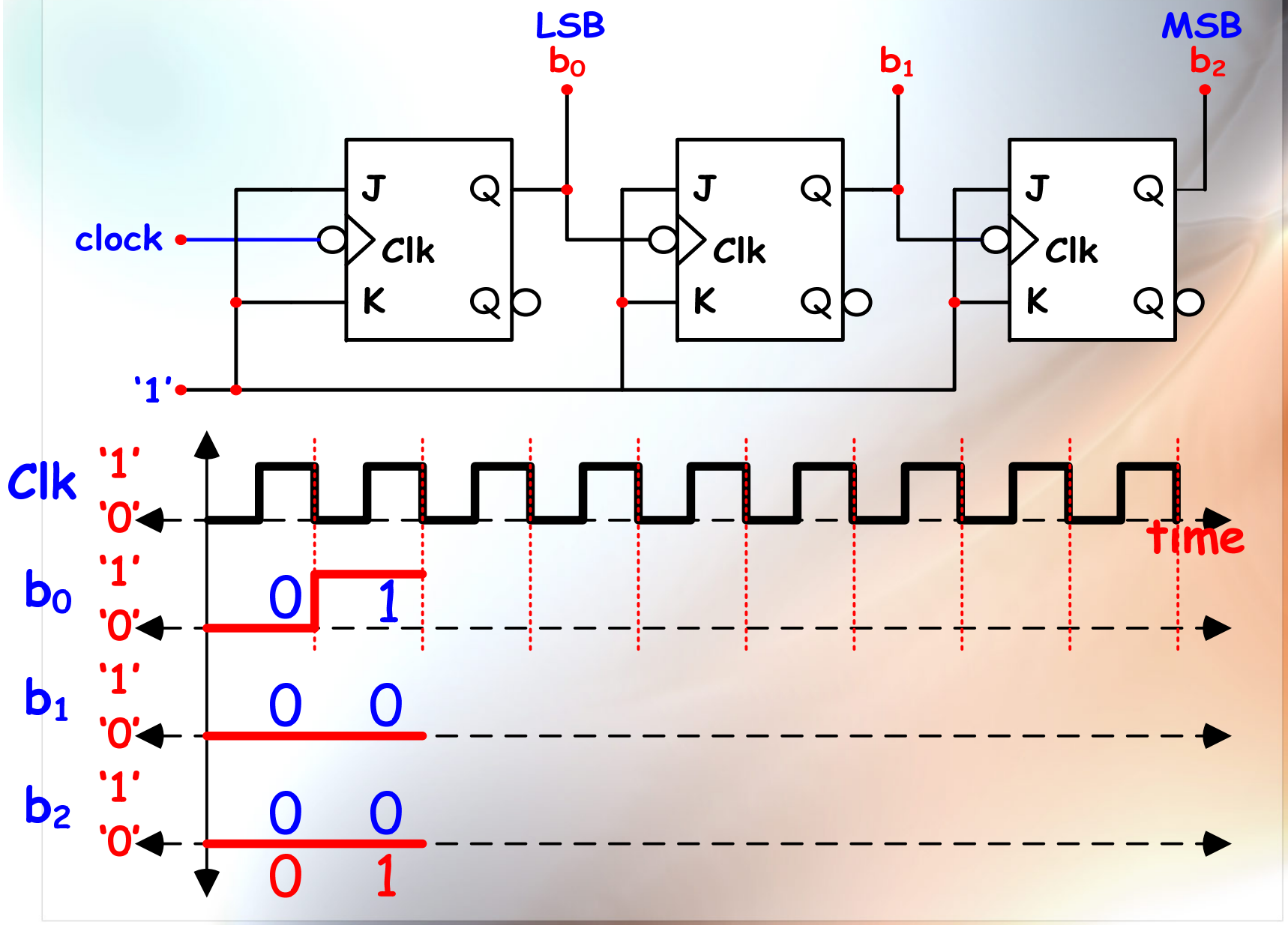
Ripple Counter Timing



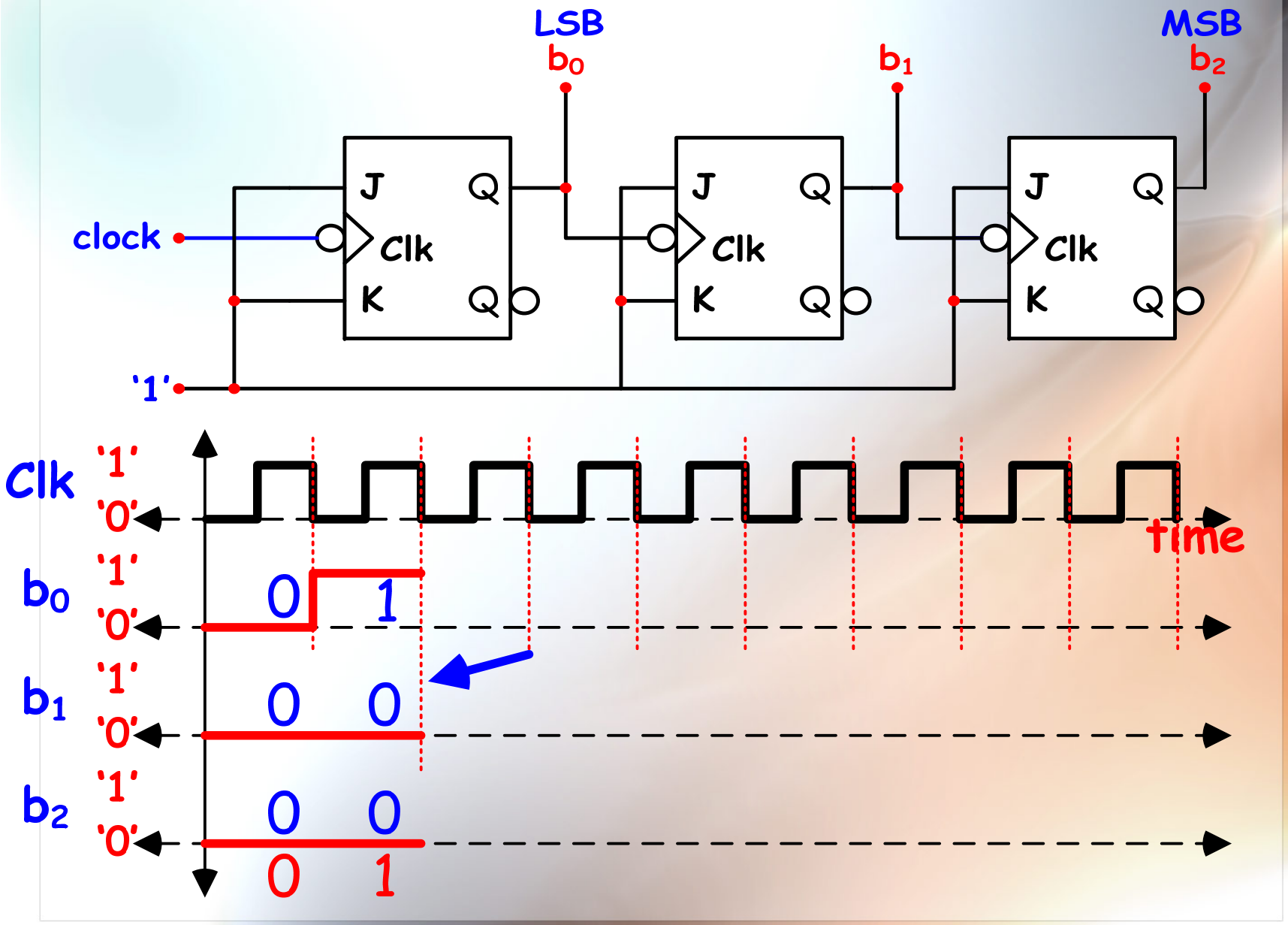
Ripple Counter Timing



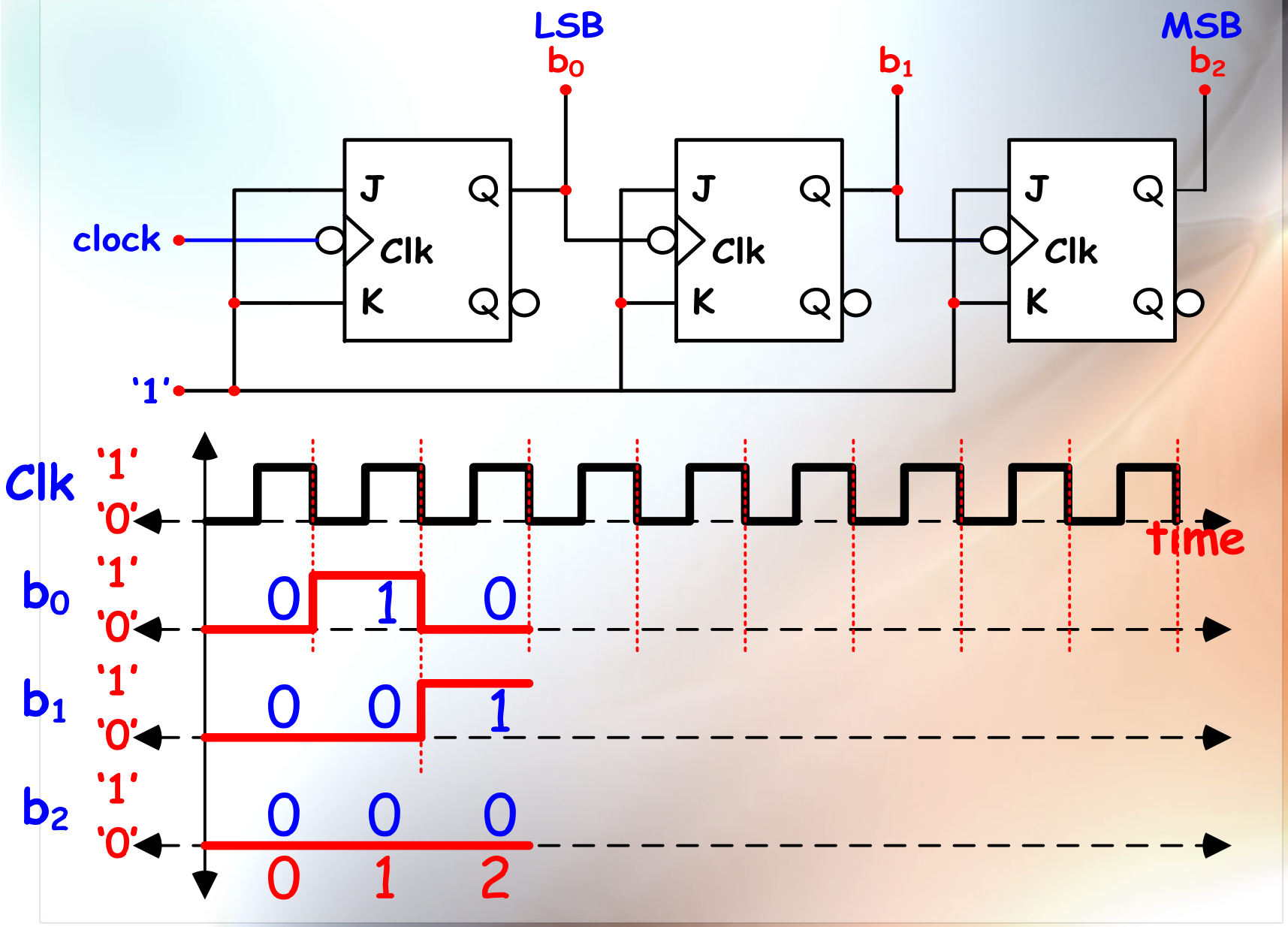
Ripple Counter Timing



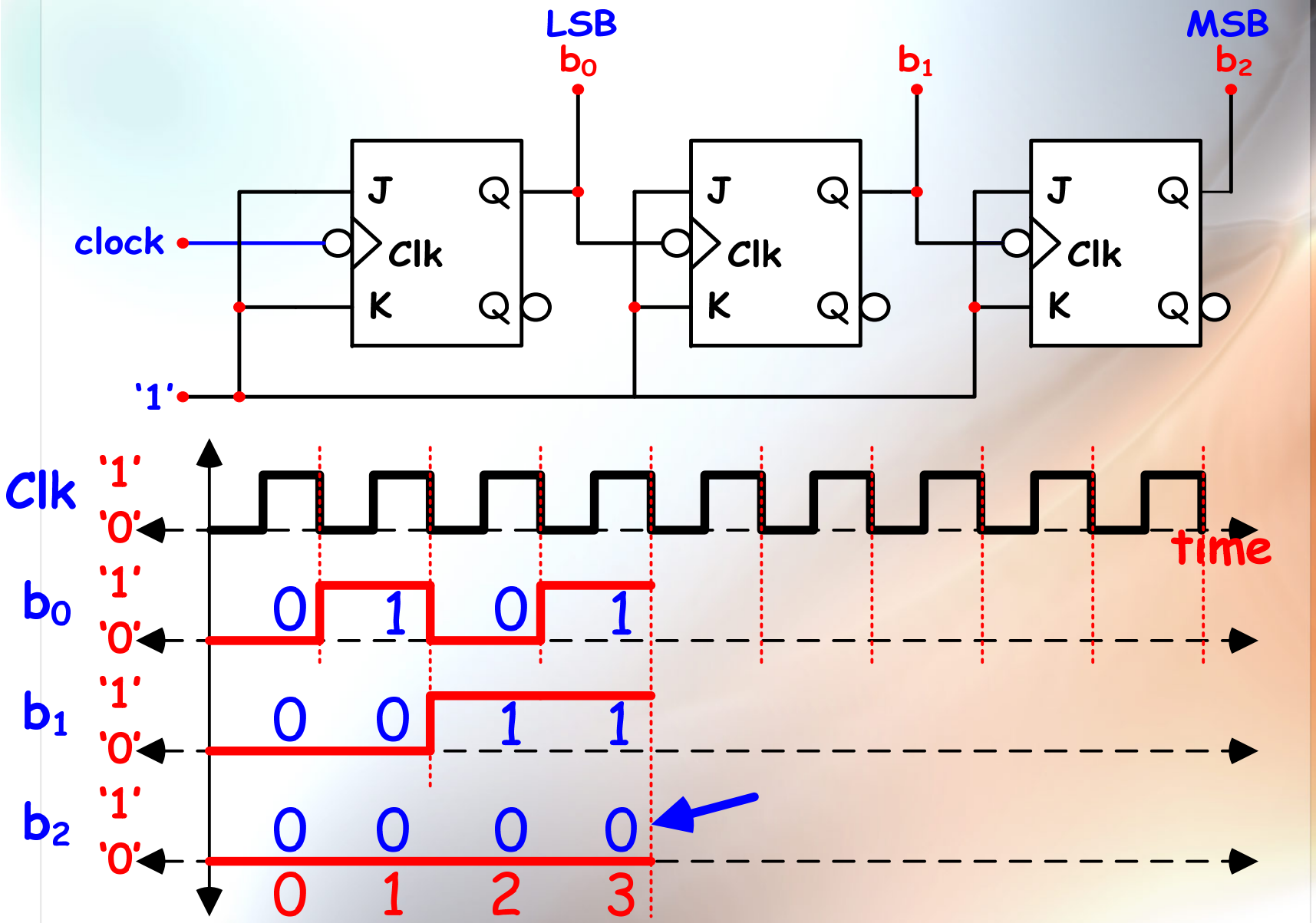
Ripple Counter Timing



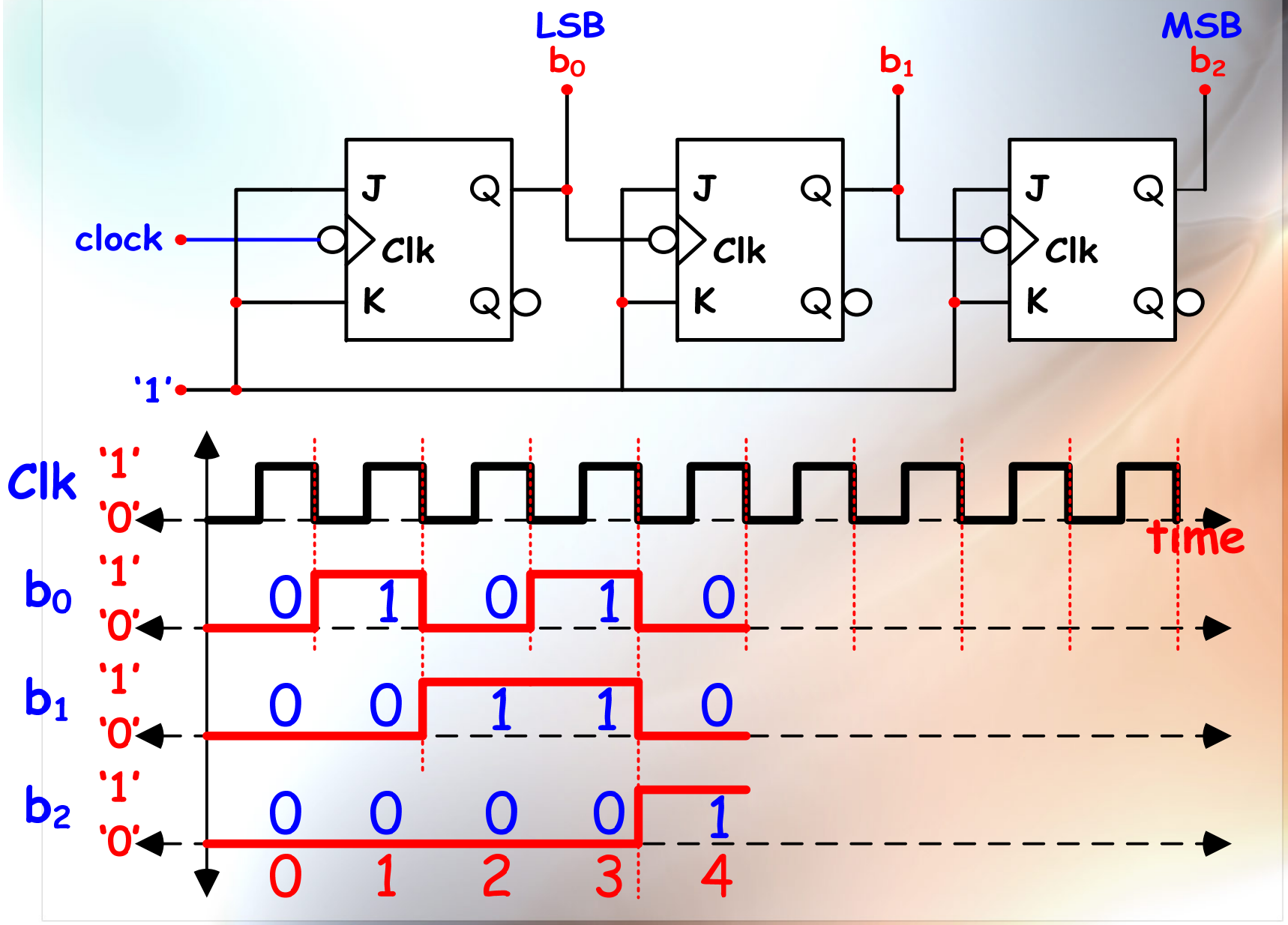
Ripple Counter Timing



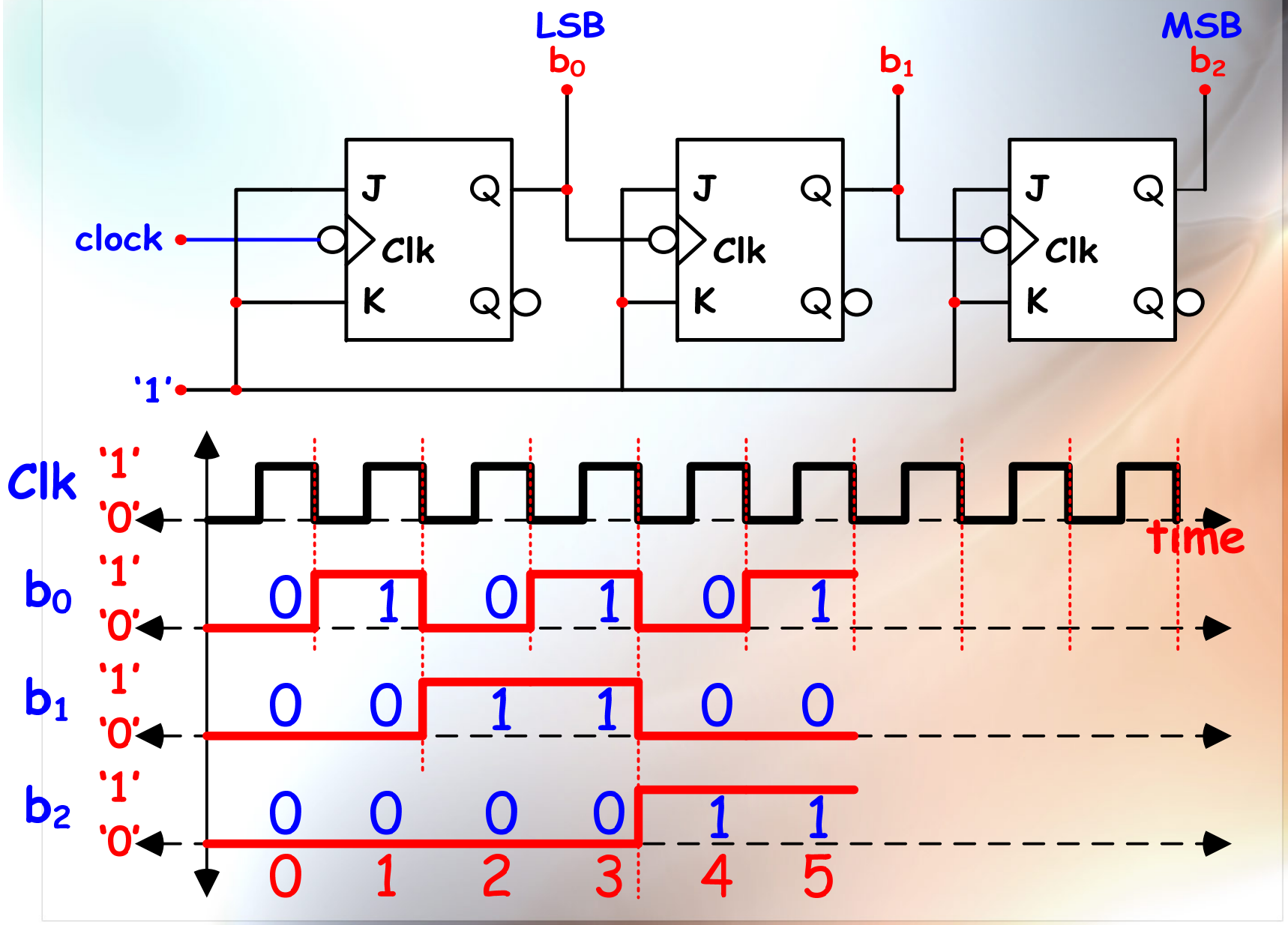
Ripple Counter Timing



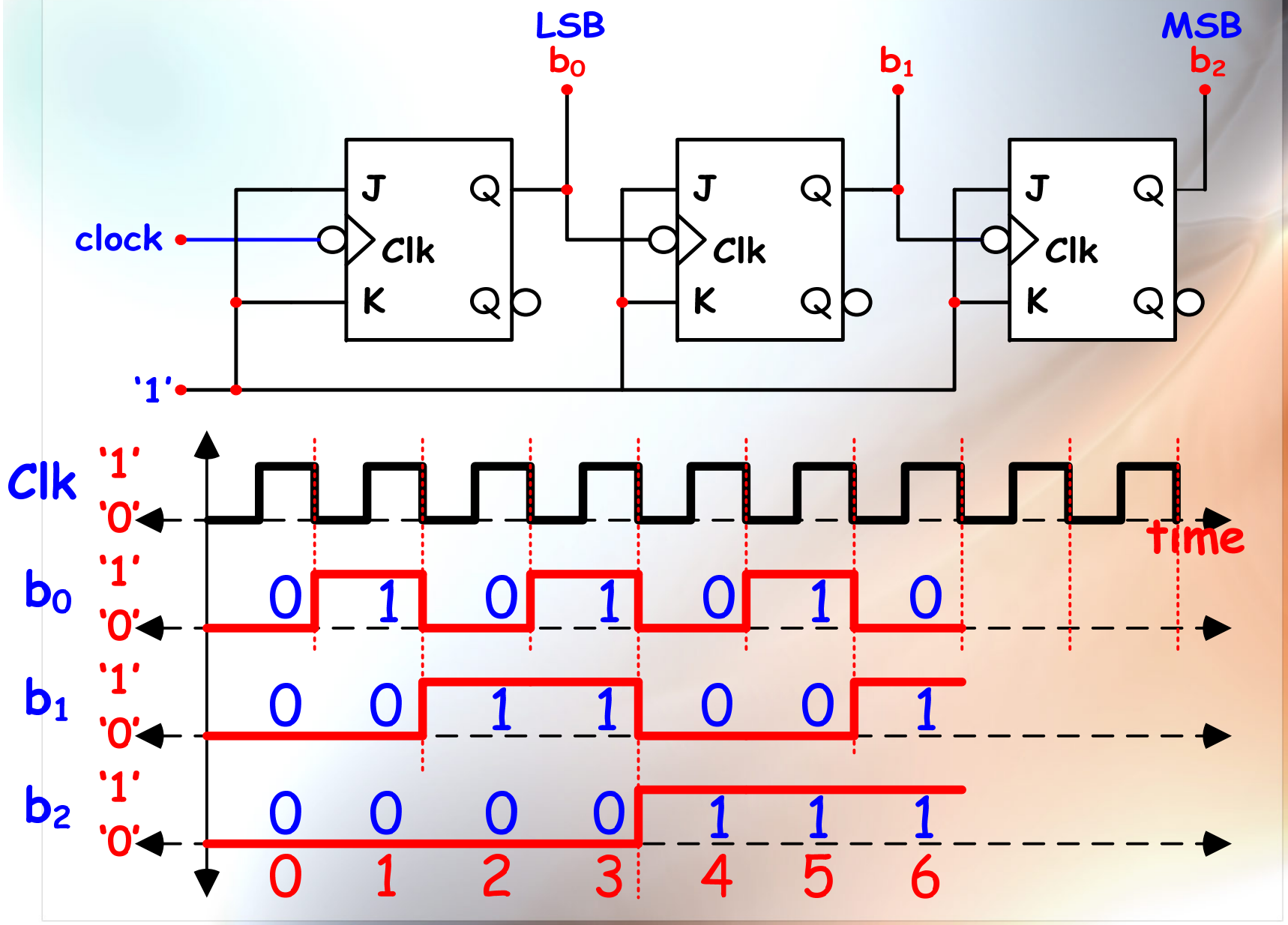
Ripple Counter Timing



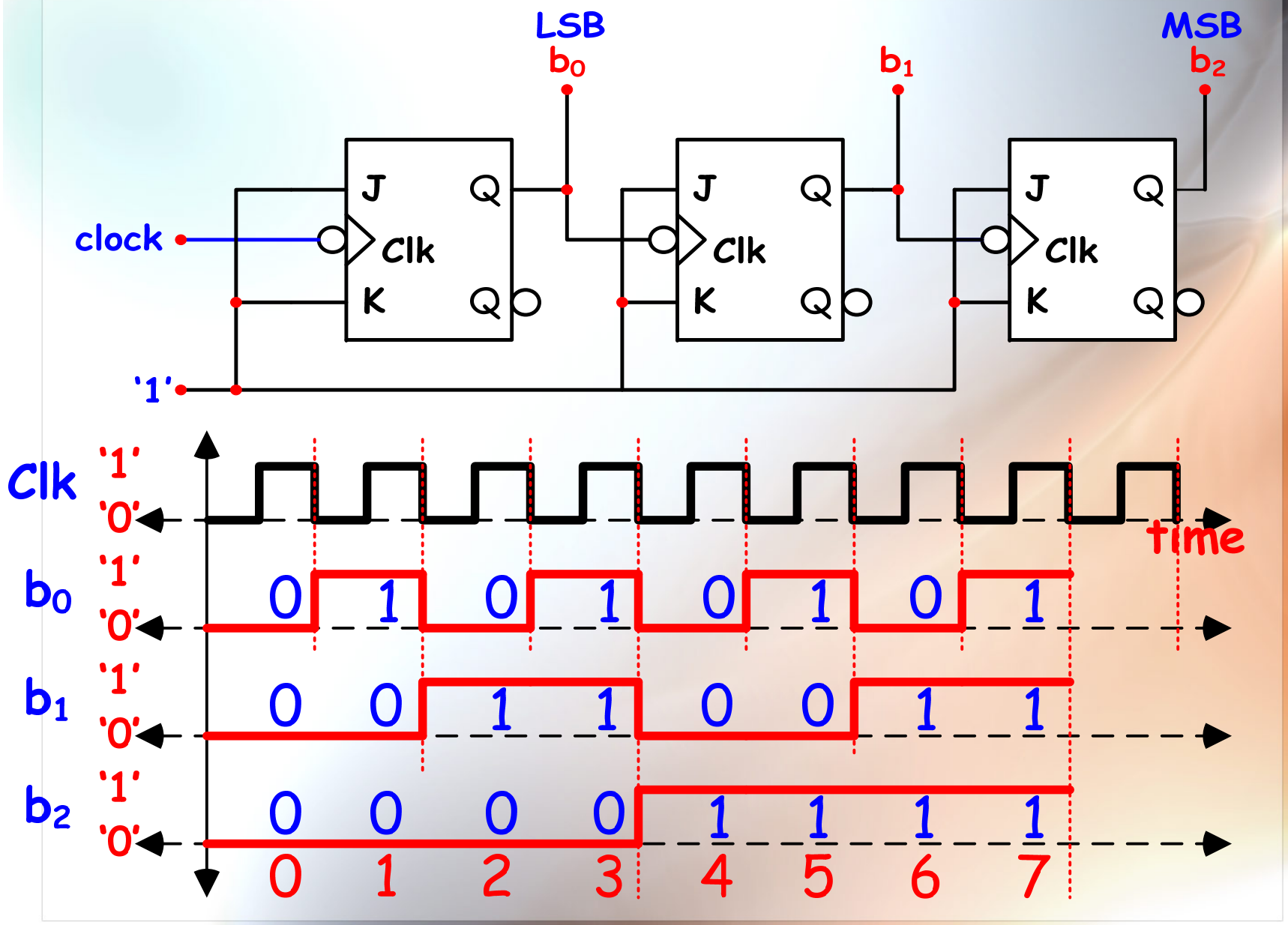
Ripple Counter Timing



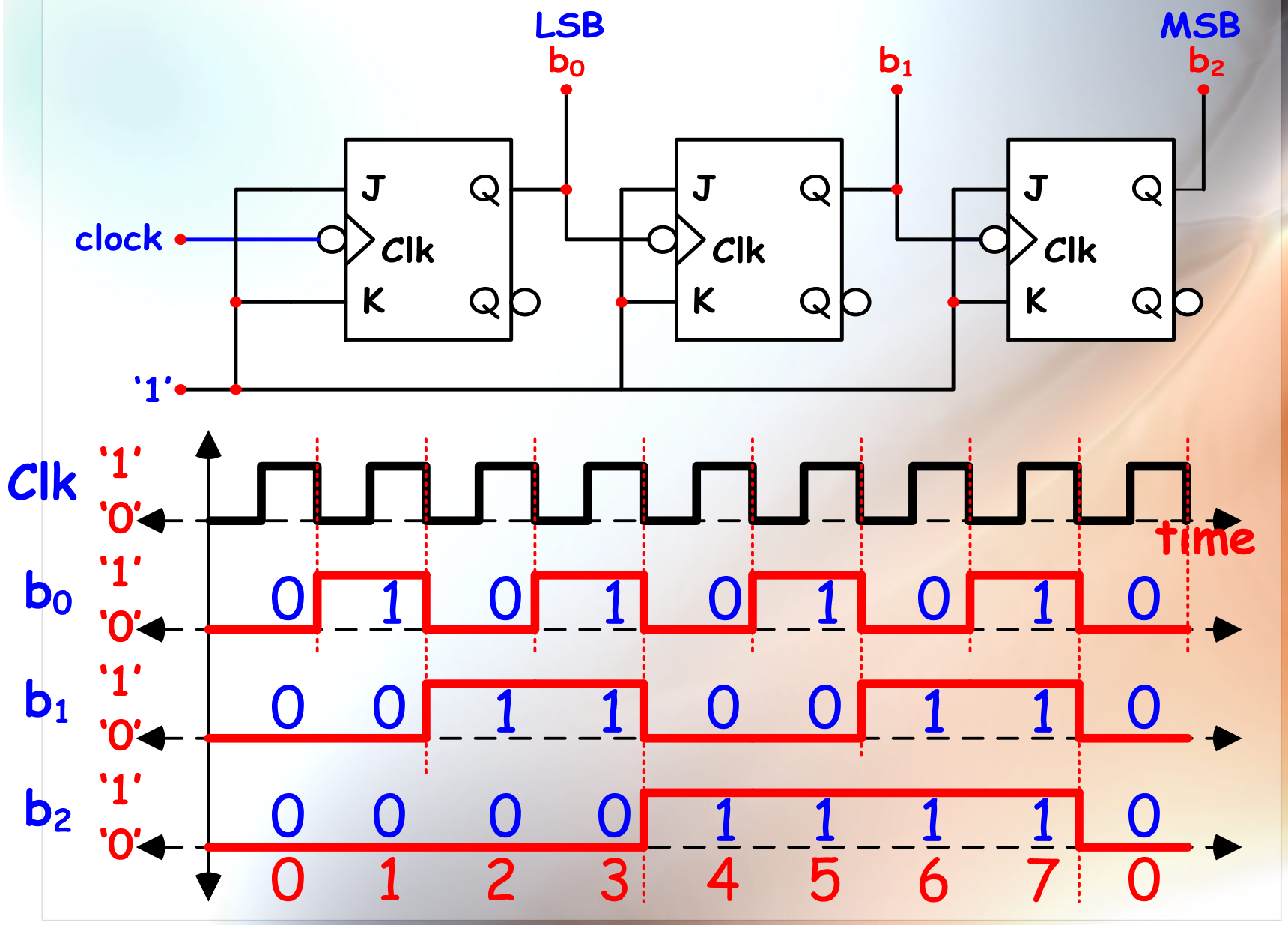
Ripple Counter Timing



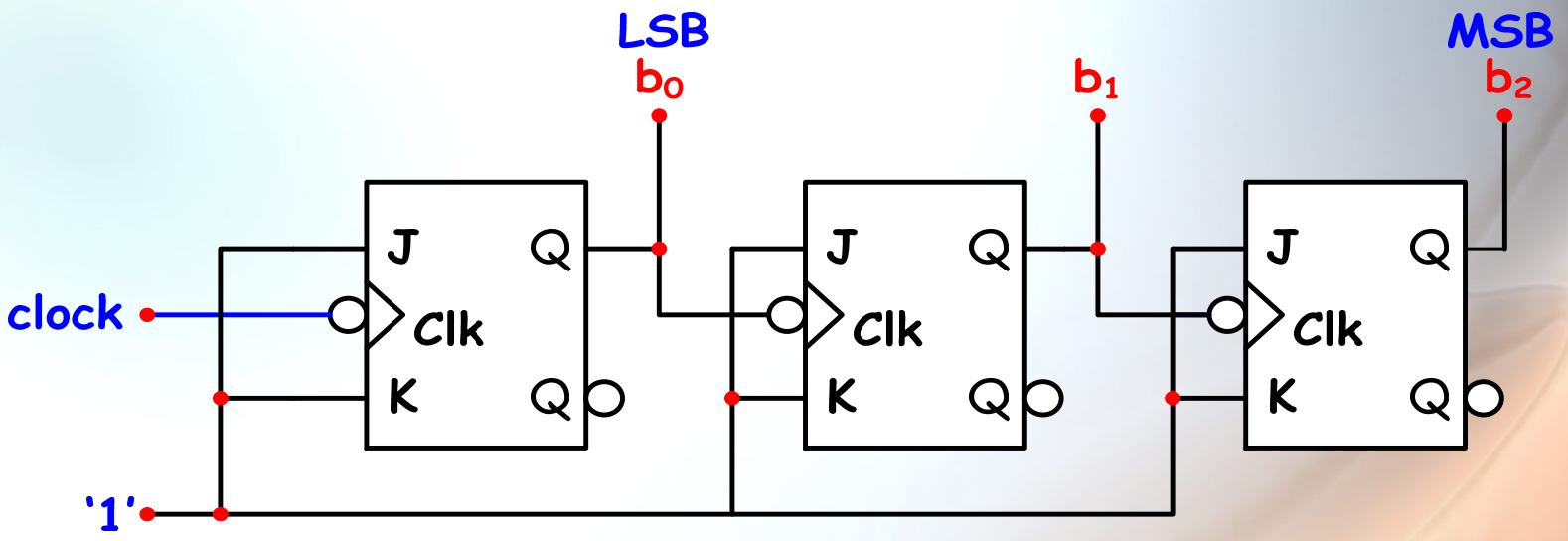
Ripple Counter Timing



Ripple Counter Timing

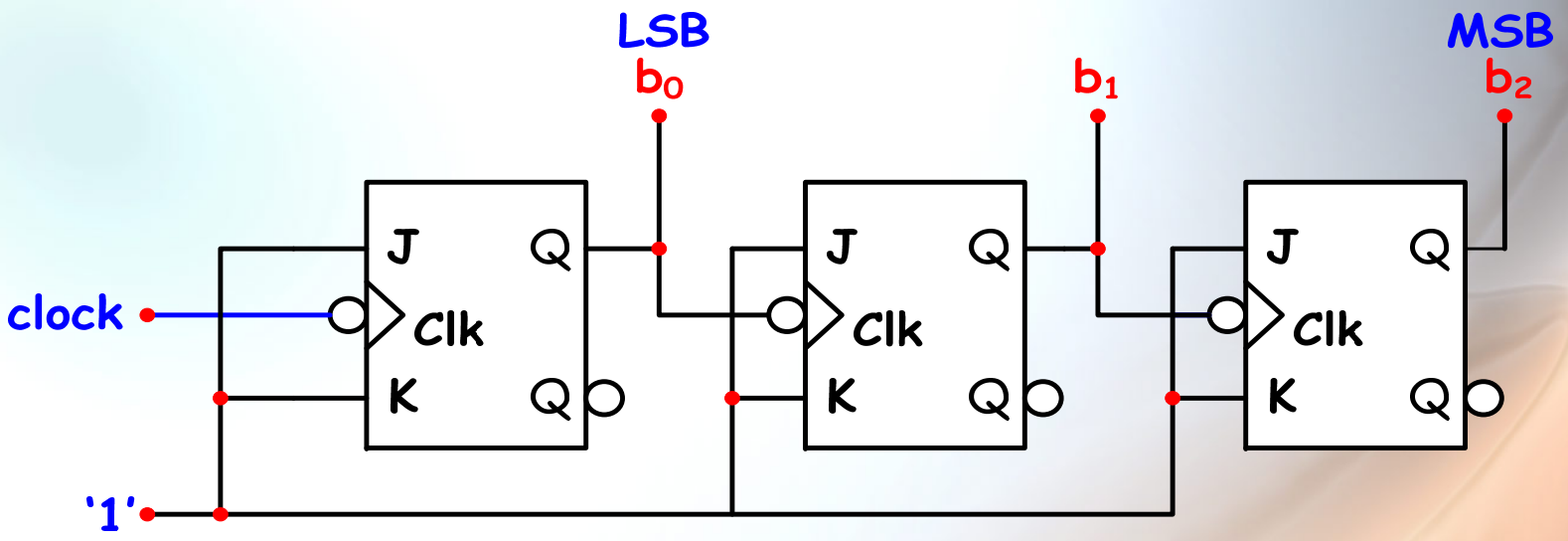


Problem with Ripple Counters



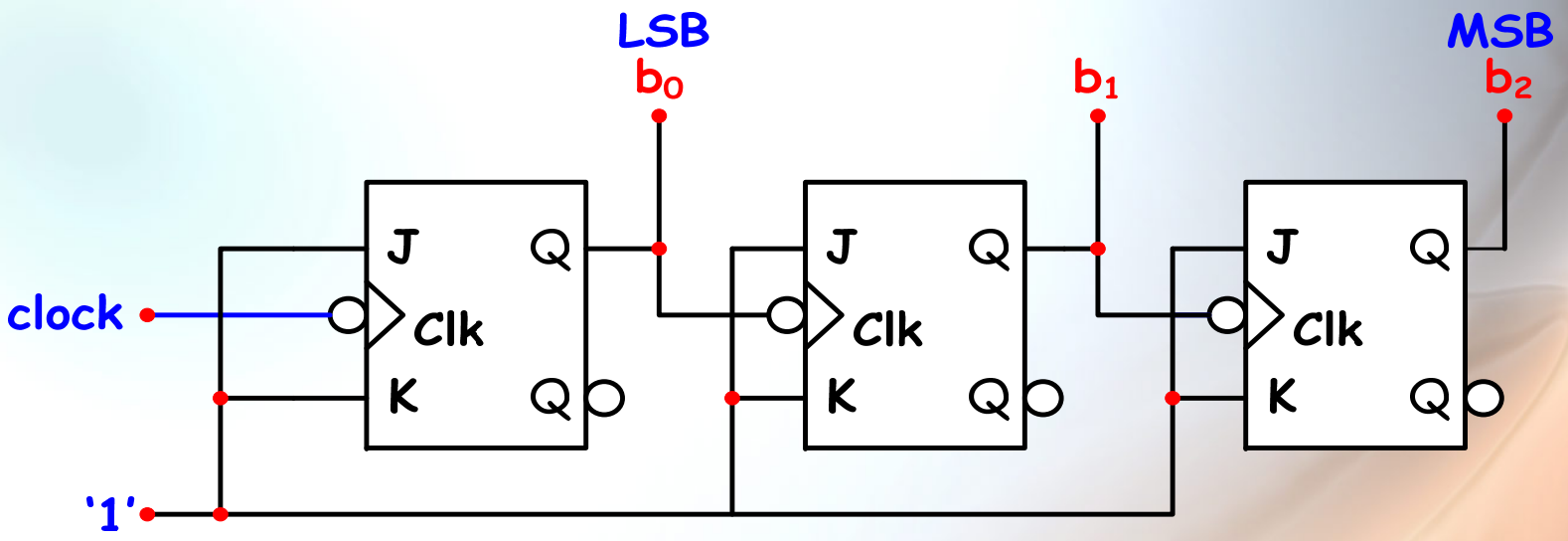
- Ripple counters exhibit a problem at higher frequencies of operation (no longer considering ideal devices).

Problem with Ripple Counters



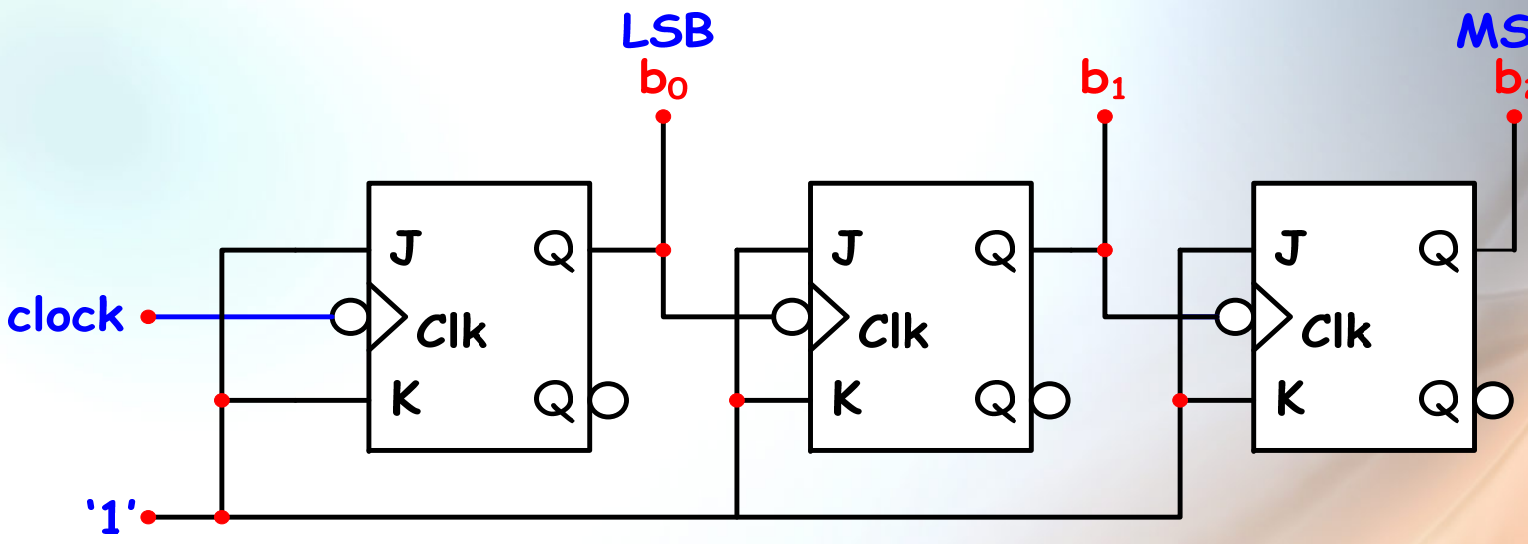
- Ripple counters exhibit a problem at higher frequencies of operation (no longer considering ideal devices).
- Each FF has a propagation delay. The problem is that the **propagation delay (PD)** of **FF #1** will force **FF#2** to be delayed.

Problem with Ripple Counters



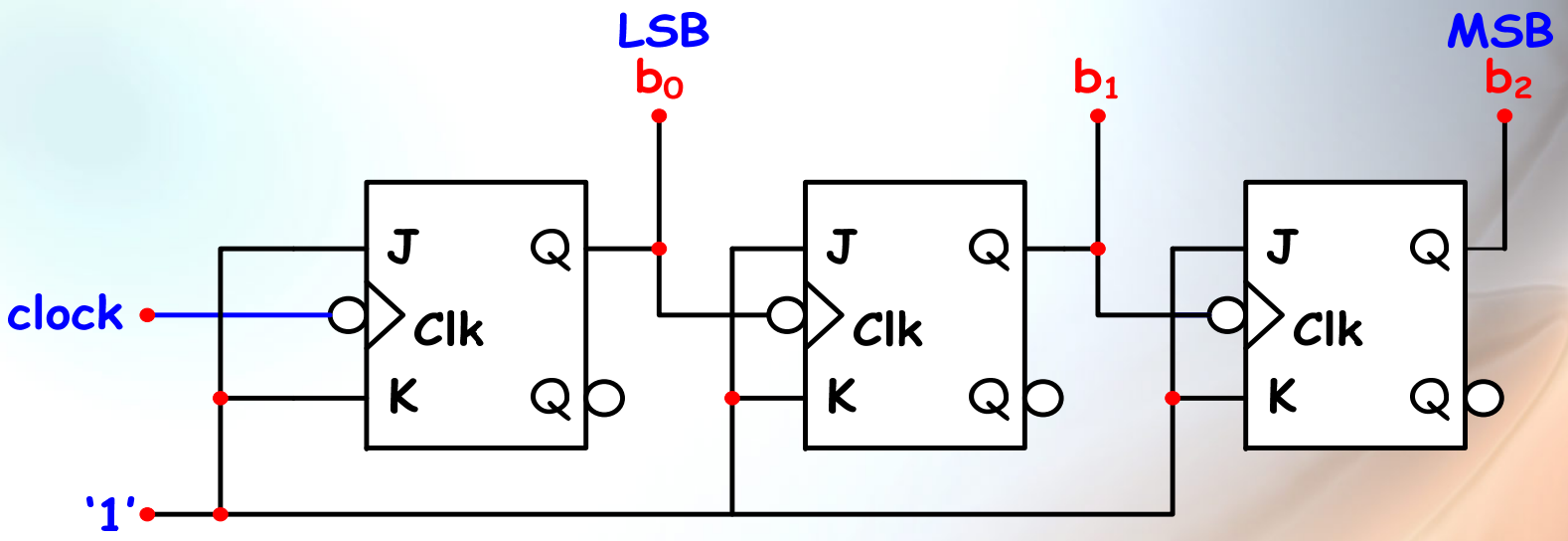
- FF #2's PD will cause FF #3 to be delayed even more ($PD_{\#1} + PD_{\#2}$).

Problem with Ripple Counters



- FF #2's PD will cause FF #3 to be delayed even more ($PD_{\#1} + PD_{\#2}$).
- FF #3's PD will cause its output to be further delayed ($PD_{\#1} + PD_{\#2} + PD_{\#3}$)

Problem with Ripple Counters



- FF #2's PD will cause FF #3 to be delayed even more ($PD_{\#1} + PD_{\#2}$).
- FF #3's PD will cause its output to be further delayed ($PD_{\#1} + PD_{\#2} + PD_{\#3}$)
- This could cause **STATE ERROR's** or **Glitches**. **(spurious states between the actual states)**

Synchronous Counters

- A **Synchronous counter** is a counter where **the clock is connected to ALL FF's.**

Synchronous Counters

- A **Synchronous counter** is a counter where **the clock is connected to ALL FF's.**
- The design of this type of counter usually requires the creation of a state table.

A 3-bit T-FF Synch Counter Design

Design a 3-bit synchronous counter using T-FF's.

A 3-bit T-FF Synch Counter Design

Y_2	Y_1	Y_0	#
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

A 3-bit T-FF Synch Counter Design

Y_2	Y_1	Y_0	#	$\#^+$
0	0	0	0	1
0	0	1	1	
0	1	0	2	
0	1	1	3	
1	0	0	4	
1	0	1	5	
1	1	0	6	
1	1	1	7	

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	$\#^+$
0	0	0	0	1
0	0	1	1	2
0	1	0	2	
0	1	1	3	
1	0	0	4	
1	0	1	5	
1	1	0	6	
1	1	1	7	

A 3-bit T-FF Synch Counter Design

Y_2	Y_1	Y_0	#	$\#^+$
0	0	0	0	1
0	0	1	1	2
0	1	0	2	3
0	1	1	3	4
1	0	0	4	5
1	0	1	5	6
1	1	0	6	7
1	1	1	7	0

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+			
0	0	0	0	1	0	0	1			
0	0	1	1	2						
0	1	0	2	3						
0	1	1	3	4						
1	0	0	4	5						
1	0	1	5	6						
1	1	0	6	7						
1	1	1	7	0						

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+			
0	0	0	0	1	0	0	1			
0	0	1	1	2	0	1	0			
0	1	0	2	3						
0	1	1	3	4						
1	0	0	4	5						
1	0	1	5	6						
1	1	0	6	7						
1	1	1	7	0						

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+			
0	0	0	0	1	0	0	1			
0	0	1	1	2	0	1	0			
0	1	0	2	3	0	1	1			
0	1	1	3	4	1	0	0			
1	0	0	4	5	1	0	1			
1	0	1	5	6	1	1	0			
1	1	0	6	7	1	1	1			
1	1	1	7	0	0	0	0			

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	#$^+$	y_2^+	y_1^+	y_0^+	T_2		
0	0	0	0	1	0	0	1			
0	0	1	1	2	0	1	0			
0	1	0	2	3	0	1	1			
0	1	1	3	4	1	0	0			
1	0	0	4	5	1	0	1			
1	0	1	5	6	1	1	0			
1	1	0	6	7	1	1	1			
1	1	1	7	0	0	0	0			

We now need to focus on the design of the T FF's. We start with the **MSB T-FF**.

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+	T_2		
0	0	0	0	1	0	0	1			
0	0	1	1	2	0	1	0			
0	1	0	2	3	0	1	1			
0	1	1	3	4	1	0	0			
1	0	0	4	5	1	0	1			
1	0	1	5	6	1	1	0			
1	1	0	6	7	1	1	1			
1	1	1	7	0	0	0	0			

It might help now to remember what the T-FF's truth table looks like.

T	Q_p	Q_N	
0	0	0	Hold
0	1	1	
1	0	1	Toggle
1	1	0	

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+	T_2		
0	0	0	0	1	0	0	1			
0	0	1	1	2	0	1	0			
0	1	0	2	3	0	1	1			
0	1	1	3	4	1	0	0			
1	0	0	4	5	1	0	1			
1	0	1	5	6	1	1	0			
1	1	0	6	7	1	1	1			
1	1	1	7	0	0	0	0			

Or better yet, lets remodel the truth table into some-thing called the **Transition Table**.

Q_p	Q_N	T
0	0	0
0	1	1
1	0	1
1	1	0

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+	T_2
0	0	0	0	1	0	0	1	
0	0	1	1	2	0	1	0	
0	1	0	2	3	0	1	1	
0	1	1	3	4	1	0	0	
1	0	0	4	5	1	0	1	
1	0	1	5	6	1	1	0	
1	1	0	6	7	1	1	1	
1	1	1	7	0	0	0	0	

Or better yet, lets remodel the truth table into some-thing called the **Transition Table**.

Q_P	Q_N	T
0 →	0	0
0	1	1
1	0	1
1	1	0

The T-FF **Transition Table** tells us that in order for a transition from a **present state of 0** to a **next state of 0**, the **T-FF's input must have been a 0**.

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+	T_2
0	0	0	0	1	0	0	1	
0	0	1	1	2	0	1	0	
0	1	0	2	3	0	1	1	
0	1	1	3	4	1	0	0	
1	0	0	4	5	1	0	1	
1	0	1	5	6	1	1	0	
1	1	0	6	7	1	1	1	
1	1	1	7	0	0	0	0	

Or better yet, lets remodel the truth table into some-thing called the **Transition Table**.

Q_P	Q_N	T
0	0	0
0	1	1
1	0	1
1	1	0

Or that in order to have a **Transition** from 0 to 1 or from 1 to 0, the **T-FF's input must have been** a 1.

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+	T_2
0	0	0	0	1	0	0	1	
0	0	1	1	2	0	1	0	
0	1	0	2	3	0	1	1	
0	1	1	3	4	1	0	0	
1	0	0	4	5	1	0	1	
1	0	1	5	6	1	1	0	
1	1	0	6	7	1	1	1	
1	1	1	7	0	0	0	0	

Or better yet, lets remodel the truth table into some-thing called the **Transition Table**.

Q_P	Q_N	T
0	0	0
0	1	1
1	0	1
1	1	0

And finally, in order to have a **Transition** from **1** to **1**, the **T-FF's input must have been a 0**.

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+	T_2
0	0	0	0	1	0	0	1	0
0	0	1	1	2	0	1	0	0
0	1	0	2	3	0	1	1	0
0	1	1	3	4	1	0	0	
1	0	0	4	5	1	0	1	
1	0	1	5	6	1	1	0	
1	1	0	6	7	1	1	1	
1	1	1	7	0	0	0	0	

Now, back to the MSB columns of both Present and Next state.

We note that each of the 0 to 0 transitions must have required the MSB T-FF's input to have been a 0.

Q_p	Q_N	T
0	0	0
0	1	1
1	0	1
1	1	0

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+	T_2
0	0	0	0	1	0	0	1	0
0	0	1	1	2	0	1	0	0
0	1	0	2	3	0	1	1	0
0	1	1	3	4	1	0	0	
1	0	0	4	5	1	0	1	0
1	0	1	5	6	1	1	0	0
1	1	0	6	7	1	1	1	0
1	1	1	7	0	0	0	0	

We note the same thing with each of the 1 to 1 transitions.

Q_p	Q_N	T
0	0	0
0	1	1
1	0	1
1	1	0

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+	T_2
0	0	0	0	1	0	0	1	0
0	0	1	1	2	0	1	0	0
0	1	0	2	3	0	1	1	0
0	1	1	3	4	1	0	0	1
1	0	0	4	5	1	0	1	0
1	0	1	5	6	1	1	0	0
1	1	0	6	7	1	1	1	0
1	1	1	7	0	0	0	0	

Now we see that the transition from 0 to 1 required an input of 1 to have occurred.

Q_p	Q_N	T
0	0	0
0	1	1
1	0	1
1	1	0

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+	T_2
0	0	0	0	1	0	0	1	0
0	0	1	1	2	0	1	0	0
0	1	0	2	3	0	1	1	0
0	1	1	3	4	1	0	0	1
1	0	0	4	5	1	0	1	0
1	0	1	5	6	1	1	0	0
1	1	0	6	7	1	1	1	0
1	1	1	7	0	0	0	0	1

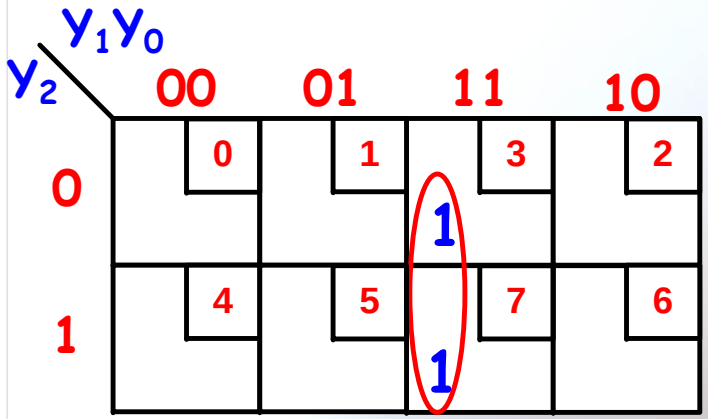
The same can be said about the transition from 1 to 0 .

Q_p	Q_N	T
0	0	0
0	1	1
1	0	1
1	1	0

A 3-bit T-FF Synch Counter Design

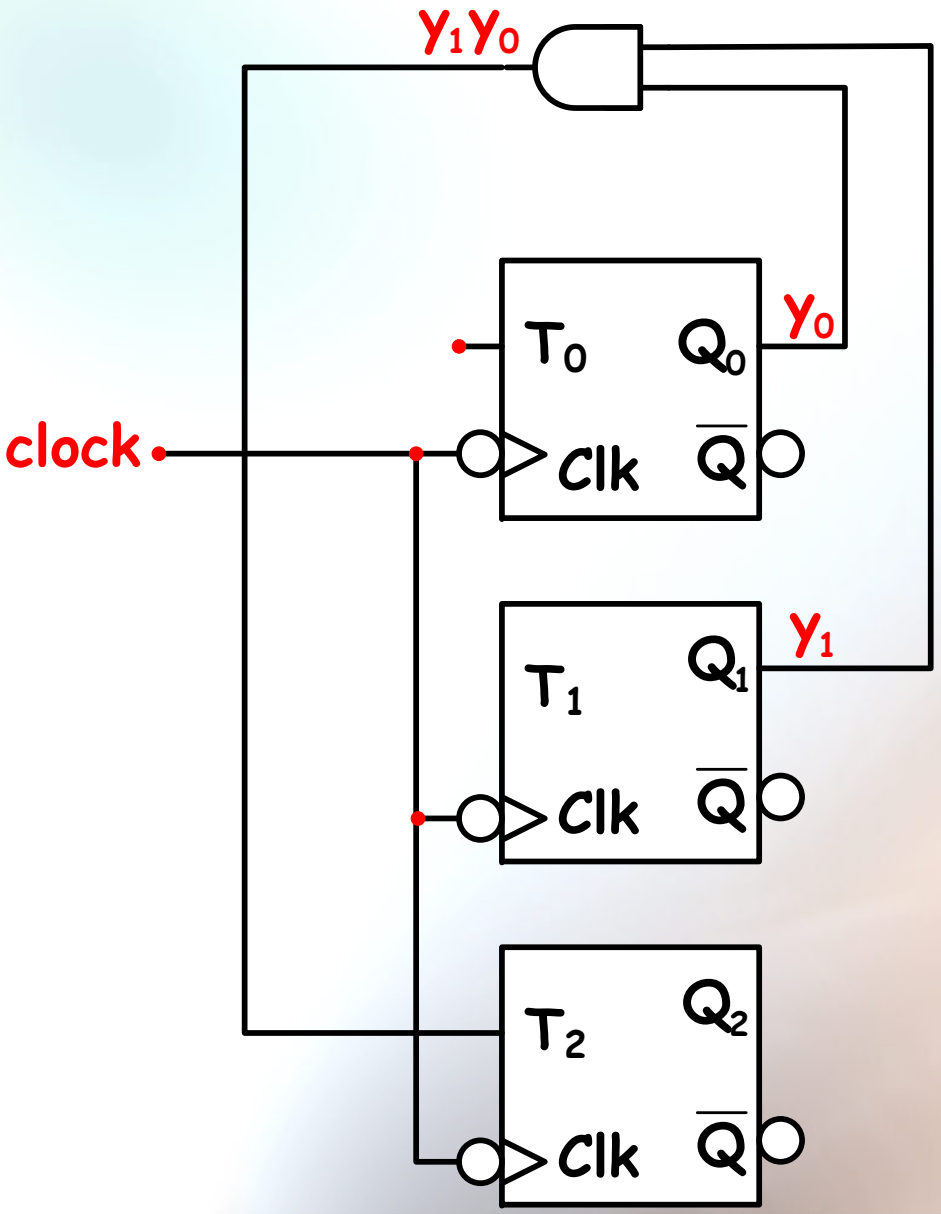
y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+	T_2
0	0	0	0	1	0	0	1	0
0	0	1	1	2	0	1	0	0
0	1	0	2	3	0	1	1	0
0	1	1	3	4	1	0	0	1
1	0	0	4	5	1	0	1	0
1	0	1	5	6	1	1	0	0
1	1	0	6	7	1	1	1	0
1	1	1	7	0	0	0	0	1

Now, K-map the T2 column.



$T_2 = y_1y_0$

A 3-bit T-FF Synch Counter Design



With the results of the K-map, start the construction of the circuit diagram.

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+	T_2	T_1
0	0	0	0	1	0	0	1	0	
0	0	1	1	2	0	1	0	0	
0	1	0	2	3	0	1	1	0	
0	1	1	3	4	1	0	0	1	
1	0	0	4	5	1	0	1	0	
1	0	1	5	6	1	1	0	0	
1	1	0	6	7	1	1	1	0	
1	1	1	7	0	0	0	0	1	

Lets repeat the process for the middle bit.

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+	T_2	T_1
0	0	0	0	1	0	0	1	0	0
0	0	1	1	2	0	1	0	0	
0	1	0	2	3	0	1	1	0	0
0	1	1	3	4	1	0	0	1	
1	0	0	4	5	1	0	1	0	0
1	0	1	5	6	1	1	0	0	
1	1	0	6	7	1	1	1	0	0
1	1	1	7	0	0	0	0	1	

Using the same process as we did with the MSB, we note that all 0 to 0 and 1 to 1 transitions required a 0 as the input of the middle T-FF.

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	#$^+$	y_2^+	y_1^+	y_0^+	T_2	T_1
0	0	0	0	1	0	0	1	0	0
0	0	1	1	2	0	1	0	0	1
0	1	0	2	3	0	1	1	0	0
0	1	1	3	4	1	0	0	1	
1	0	0	4	5	1	0	1	0	0
1	0	1	5	6	1	1	0	0	1
1	1	0	6	7	1	1	1	0	0
1	1	1	7	0	0	0	0	1	

And all 0 to 1 transitions required a 1 as the input of the middle T-FF.

A 3-bit T-FF Synch Counter Design

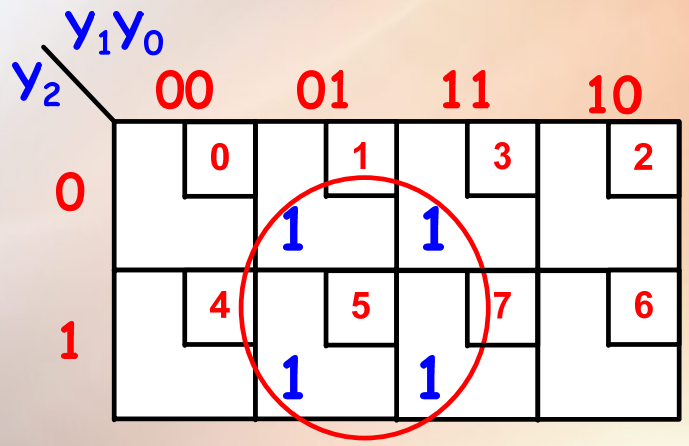
y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+	T_2	T_1
0	0	0	0	1	0	0	1	0	0
0	0	1	1	2	0	1	0	0	1
0	1	0	2	3	0	1	1	0	0
0	1	1	3	4	1	0	0	1	1
1	0	0	4	5	1	0	1	0	0
1	0	1	5	6	1	1	0	0	1
1	1	0	6	7	1	1	1	0	0
1	1	1	7	0	0	0	0	1	1

And all 1 to 0 transitions required a 1 as the input of the middle T-FF.

A 3-bit T-FF Synch Counter Design

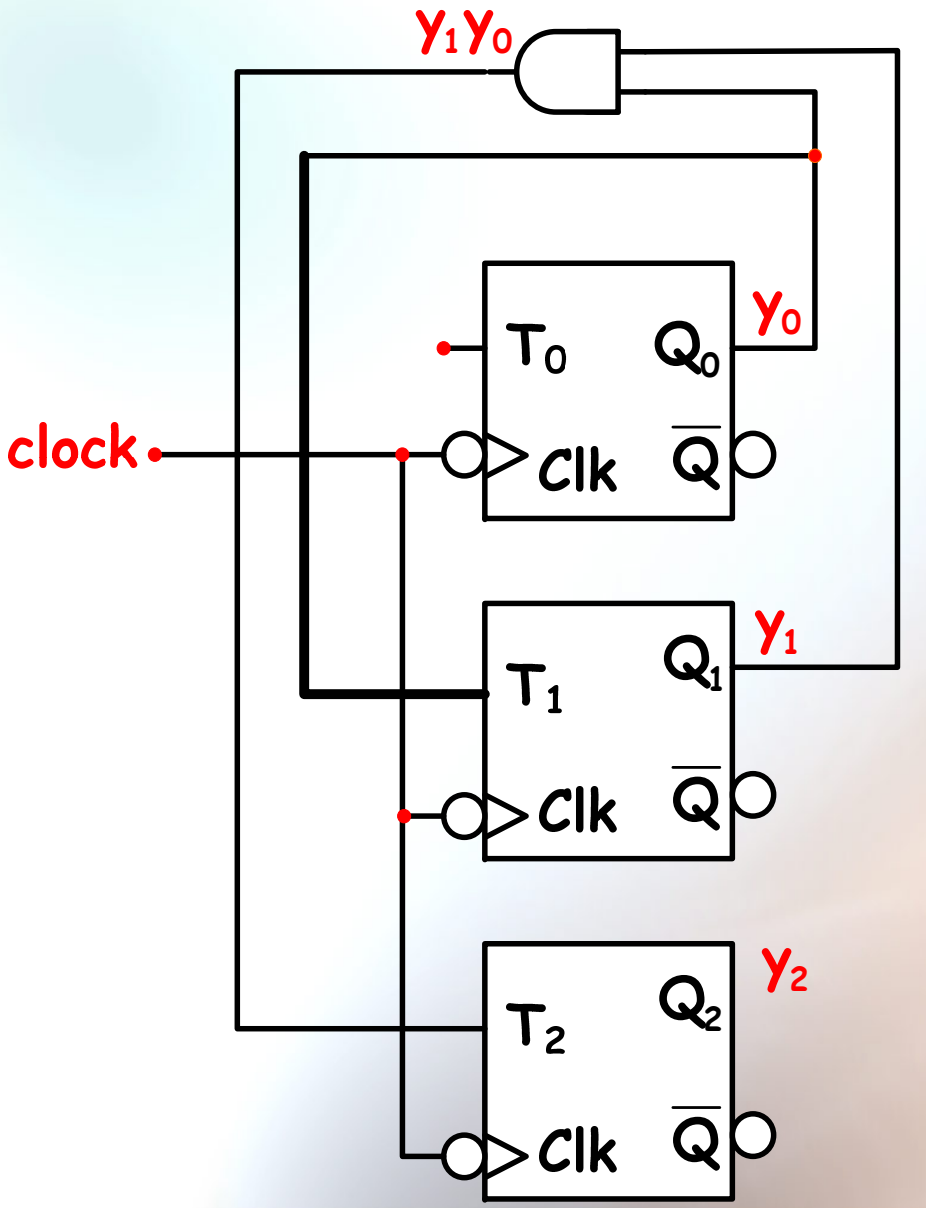
y_2	y_1	y_0	#	#$^+$	y_2^+	y_1^+	y_0^+	T_2	T_1
0	0	0	0	1	0	0	1	0	0
0	0	1	1	2	0	1	0	0	1
0	1	0	2	3	0	1	1	0	0
0	1	1	3	4	1	0	0	1	1
1	0	0	4	5	1	0	1	0	0
1	0	1	5	6	1	1	0	0	1
1	1	0	6	7	1	1	1	0	0
1	1	1	7	0	0	0	0	1	1

Now, **K-map** the **T1** column.



$T_1 = y_0$

A 3-bit T-FF Synch Counter Design



Add the new connection to the circuit.

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+	T_2	T_1	T_0
0	0	0	0	1	0	0	1	0	0	
0	0	1	1	2	0	1	0	0	1	
0	1	0	2	3	0	1	1	0	0	
0	1	1	3	4	1	0	0	1	1	
1	0	0	4	5	1	0	1	0	0	
1	0	1	5	6	1	1	0	0	1	
1	1	0	6	7	1	1	1	0	0	
1	1	1	7	0	0	0	0	1	1	

Finally we focus on the LSB transitions.

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+	T_2	T_1	T_0
0	0	0	0	1	0	0	1	0	0	1
0	0	1	1	2	0	1	0	0	1	
0	1	0	2	3	0	1	1	0	0	1
0	1	1	3	4	1	0	0	1	1	
1	0	0	4	5	1	0	1	0	0	1
1	0	1	5	6	1	1	0	0	1	
1	1	0	6	7	1	1	1	0	0	1
1	1	1	7	0	0	0	0	1	1	

All of the 0 to 1 transitions require that a 1 be placed on the LSB T-FF's input.

A 3-bit T-FF Synch Counter Design

y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+	T_2	T_1	T_0
0	0	0	0	1	0	0	1	0	0	1
0	0	1	1	2	0	1	0	0	1	1
0	1	0	2	3	0	1	1	0	0	1
0	1	1	3	4	1	0	0	1	1	1
1	0	0	4	5	1	0	1	0	0	1
1	0	1	5	6	1	1	0	0	1	1
1	1	0	6	7	1	1	1	0	0	1
1	1	1	7	0	0	0	0	1	1	1

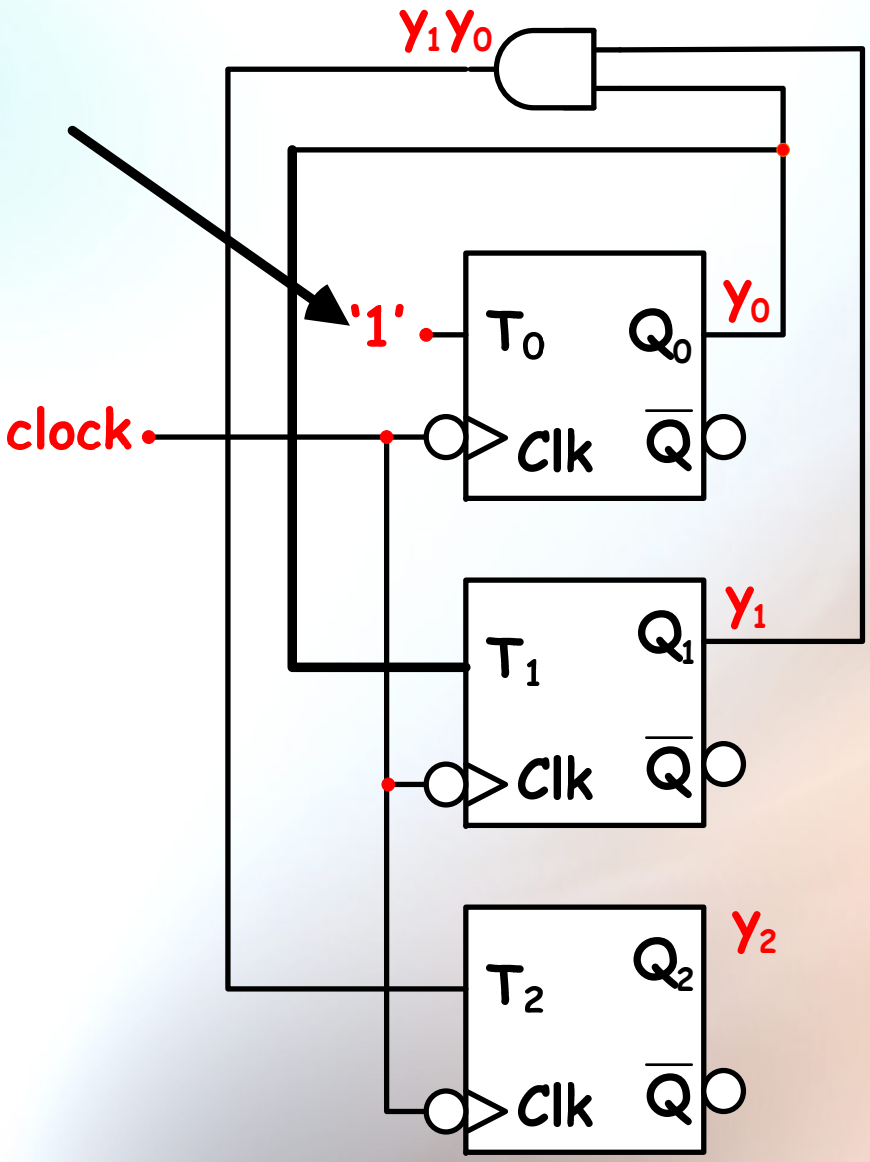
All of the 1 to 0 transitions require that a 1 be placed on the LSB T-FF's input.

A 3-bit T-FF Synch Counter Design

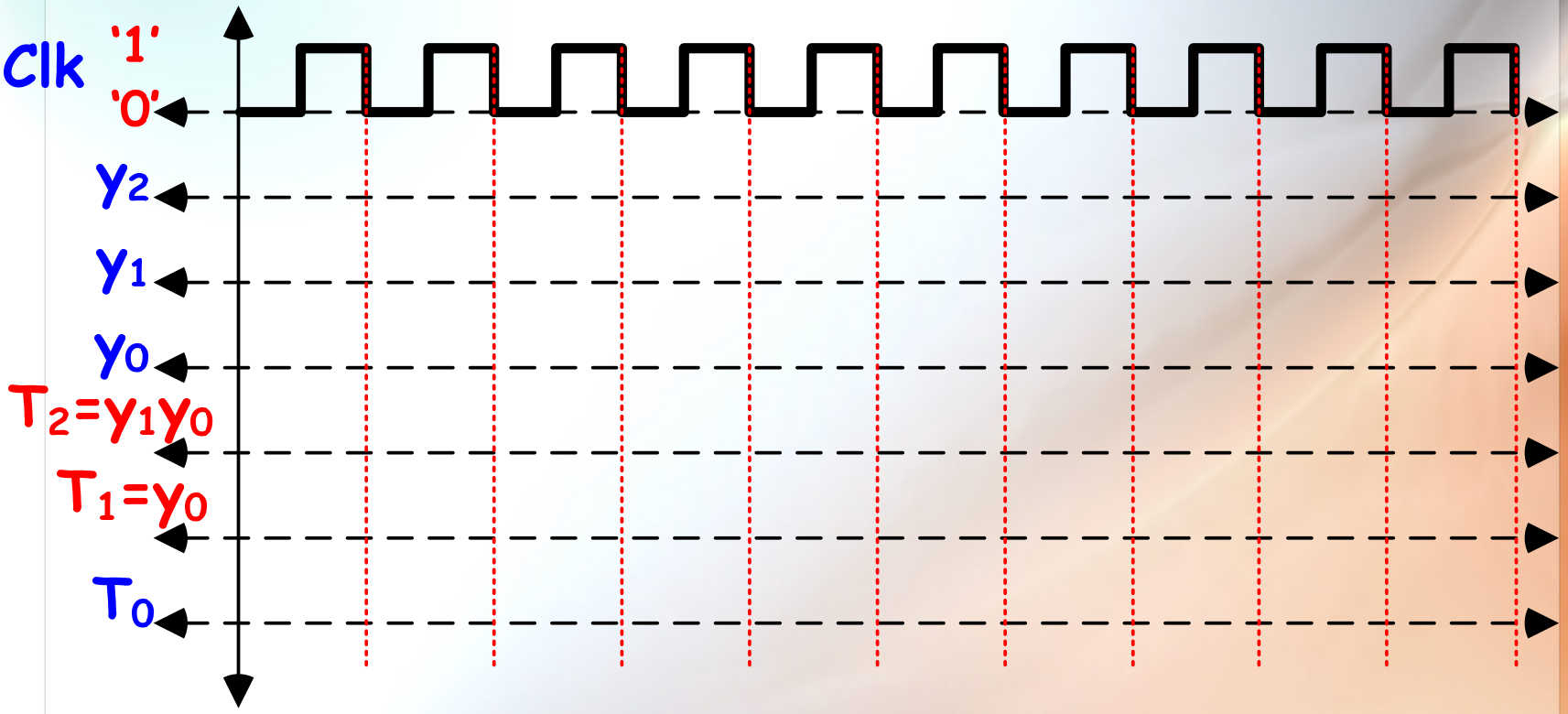
y_2	y_1	y_0	#	$\#^+$	y_2^+	y_1^+	y_0^+	T_2	T_1	T_0
0	0	0	0	1	0	0	1	0	0	1
0	0	1	1	2	0	1	0	0	1	1
0	1	0	2	3	0	1	1	0	0	1
0	1	1	3	4	1	0	0	1	1	1
1	0	0	4	5	1	0	1	0	0	1
1	0	1	5	6	1	1	0	0	1	1
1	1	0	6	7	1	1	1	0	0	1
1	1	1	7	0	0	0	0	1	1	1

There is no need to actually K-map the LSB column because it is obvious "by inspection" that the result would be $T_0 = 1$

The Circuit

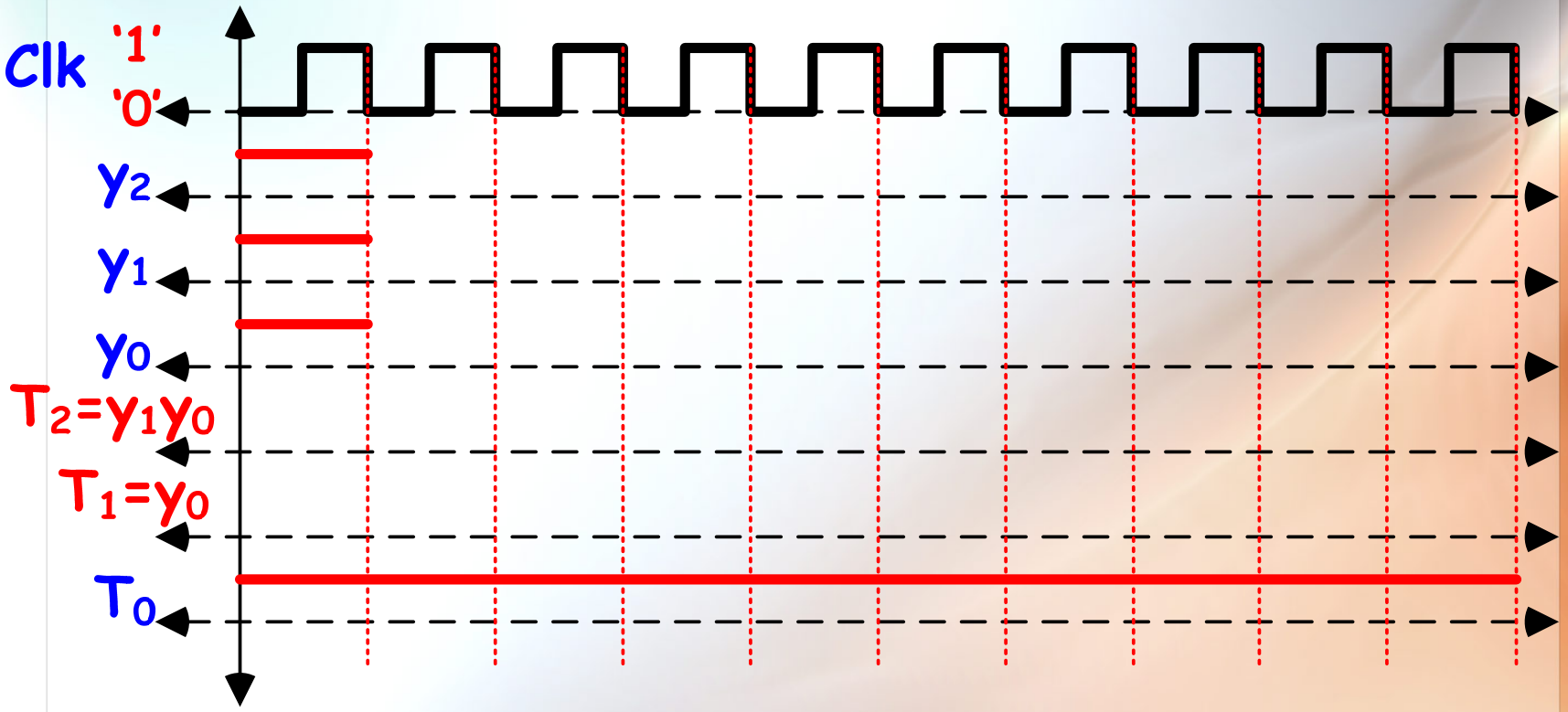

Y_1Y_0
'1'
T_0
Q_0
Y_0
clock
Clk
$\overline{Q}$
T_1
Q_1
Y_1
Clk
$\overline{Q}$
T_2
Q_2
Y_2
Clk
$\overline{Q}$

The Timing Diagram



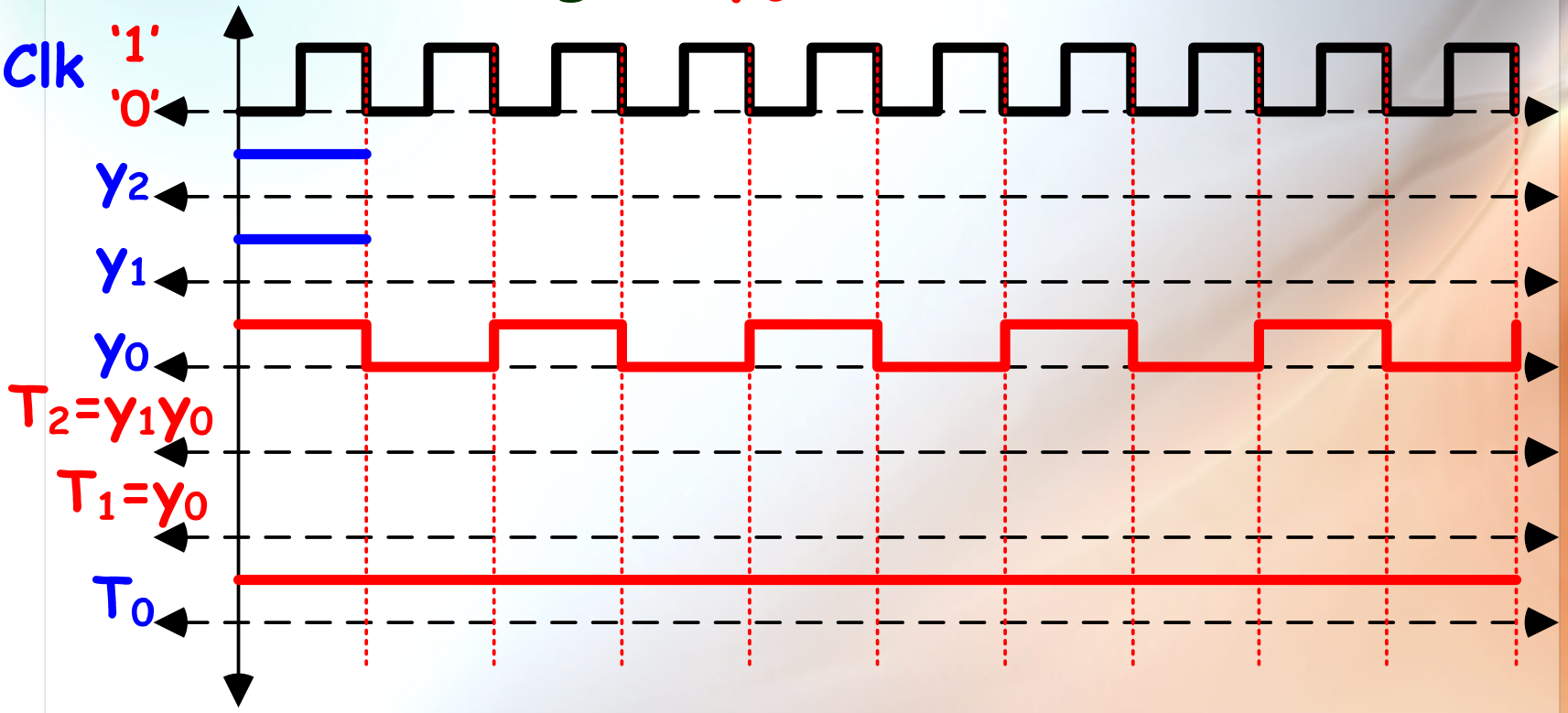
The Timing Diagram

Add in the $T_0=1$ line and the initial states of 111



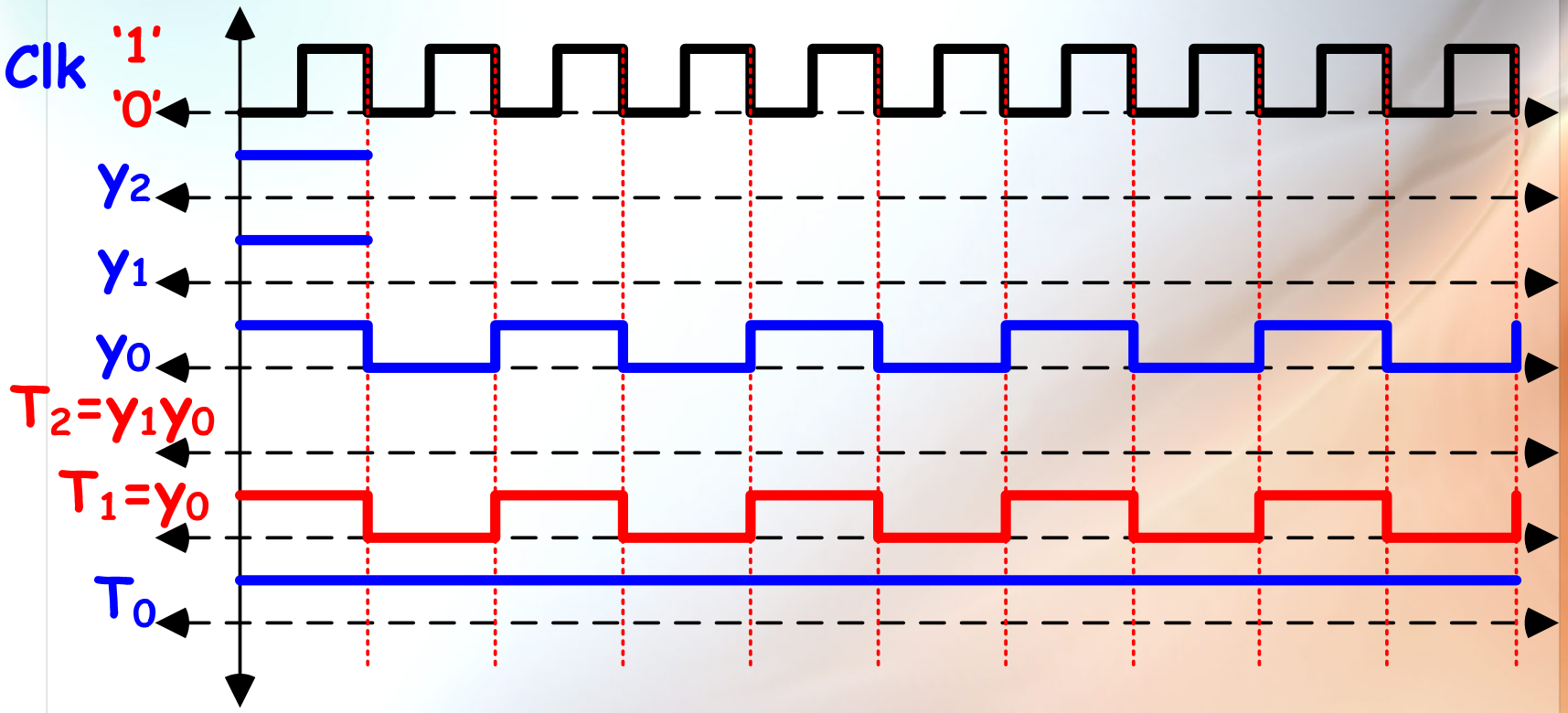
The Timing Diagram

Based on the T_0 signal, note that T1 is the same thing as y_0.



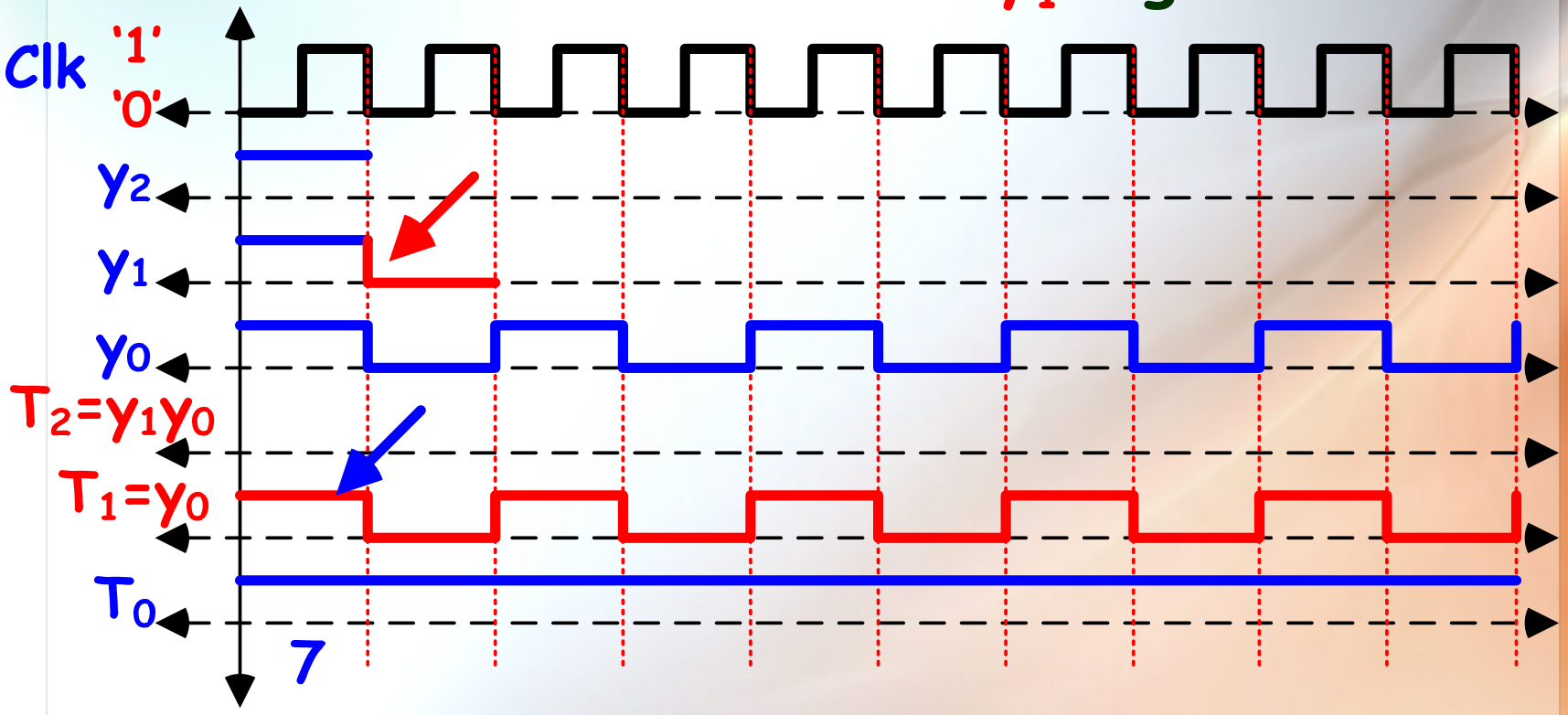
The Timing Diagram

Based on y_0 add in the T_1 signal.



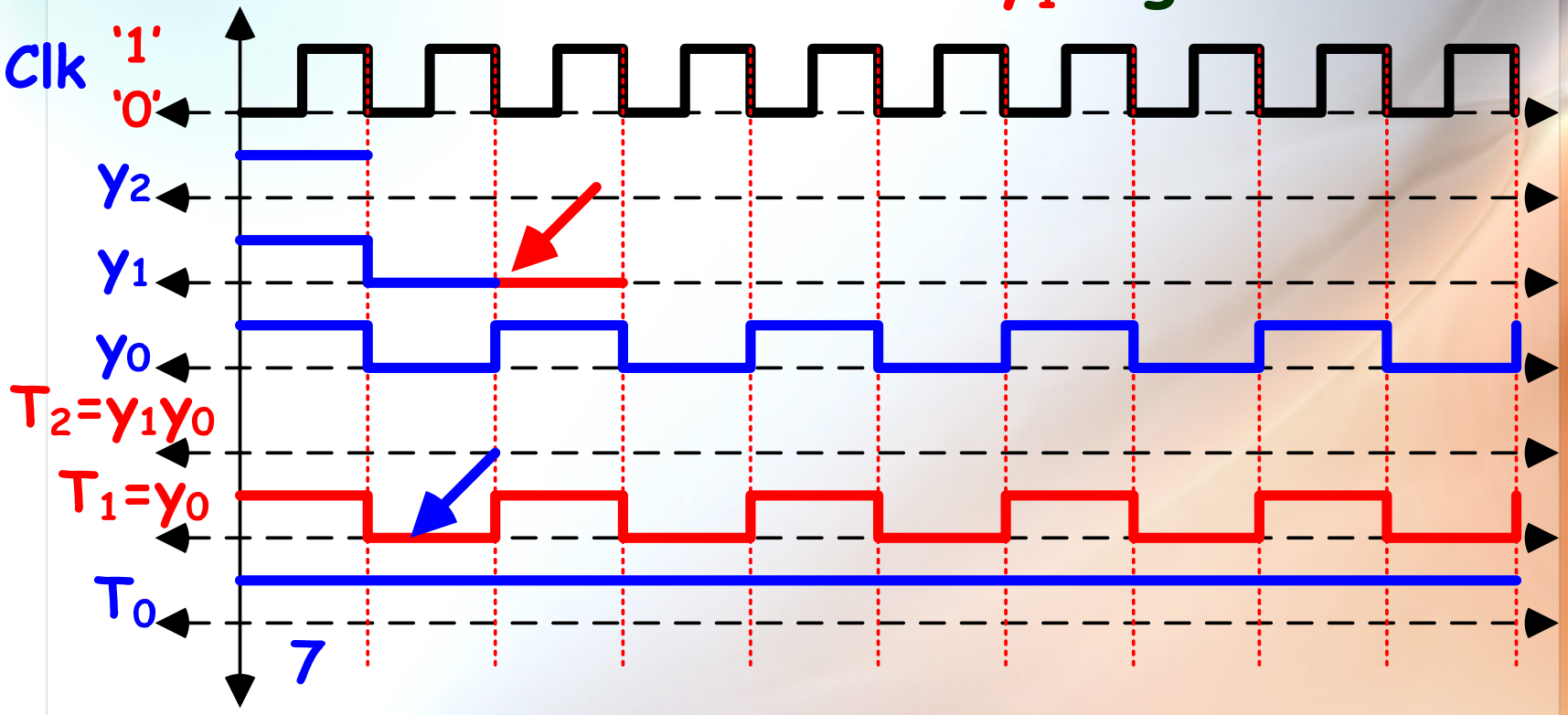
The Timing Diagram

Since y_1 is solely dependent on T_1, we can now create the entire y_1 signal.



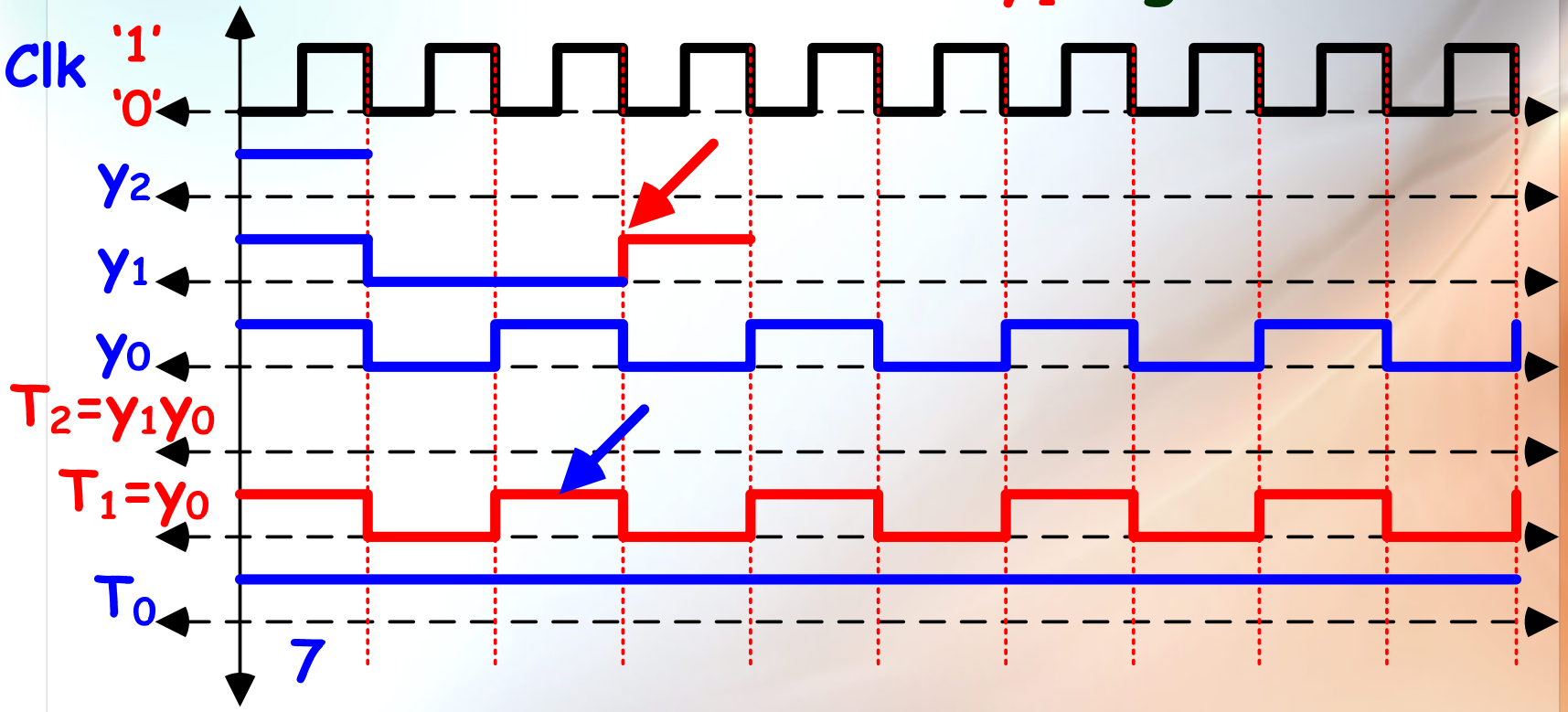
The Timing Diagram

Since y_1 is solely dependent on T_1, we can now create the entire y_1 signal.



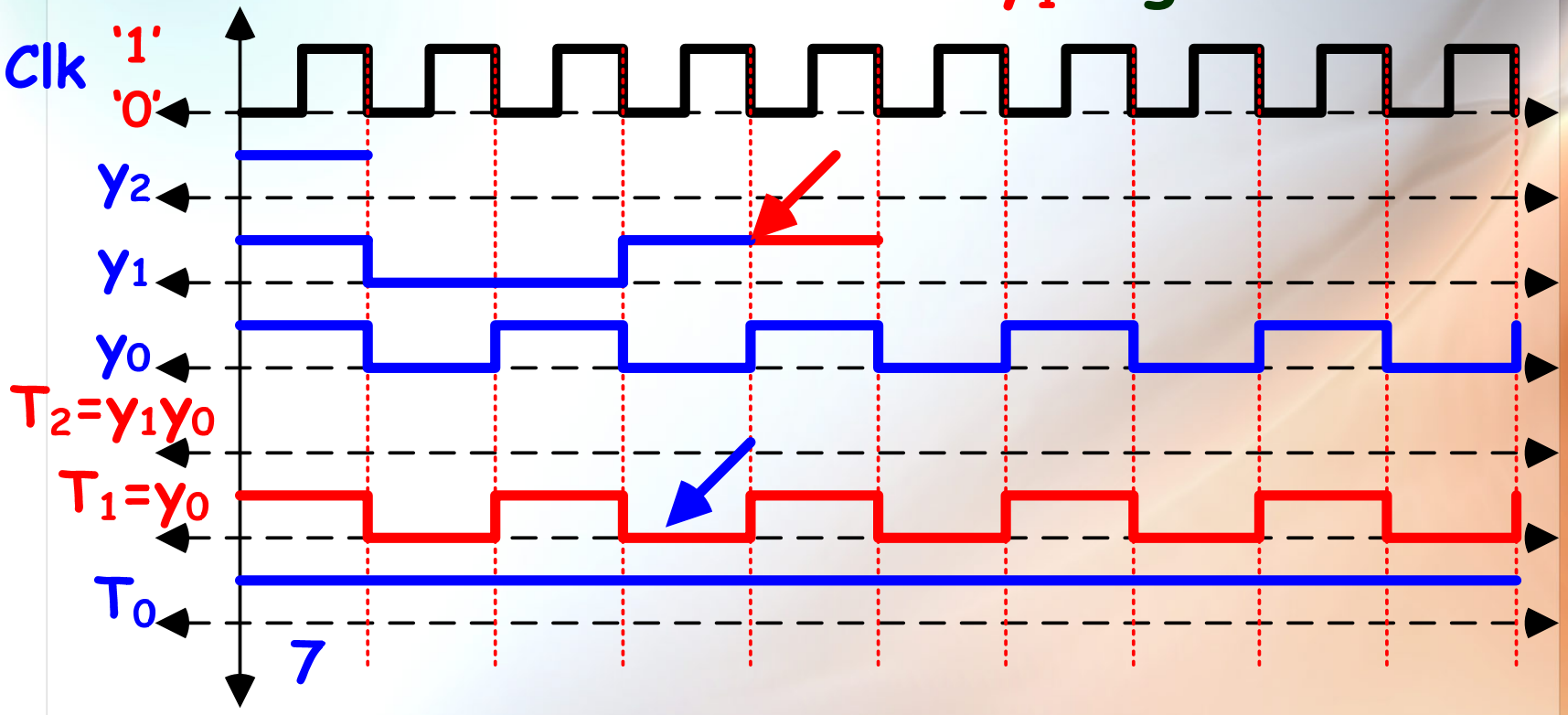
The Timing Diagram

Since y_1 is solely dependent on T_1, we can now create the entire y_1 signal.



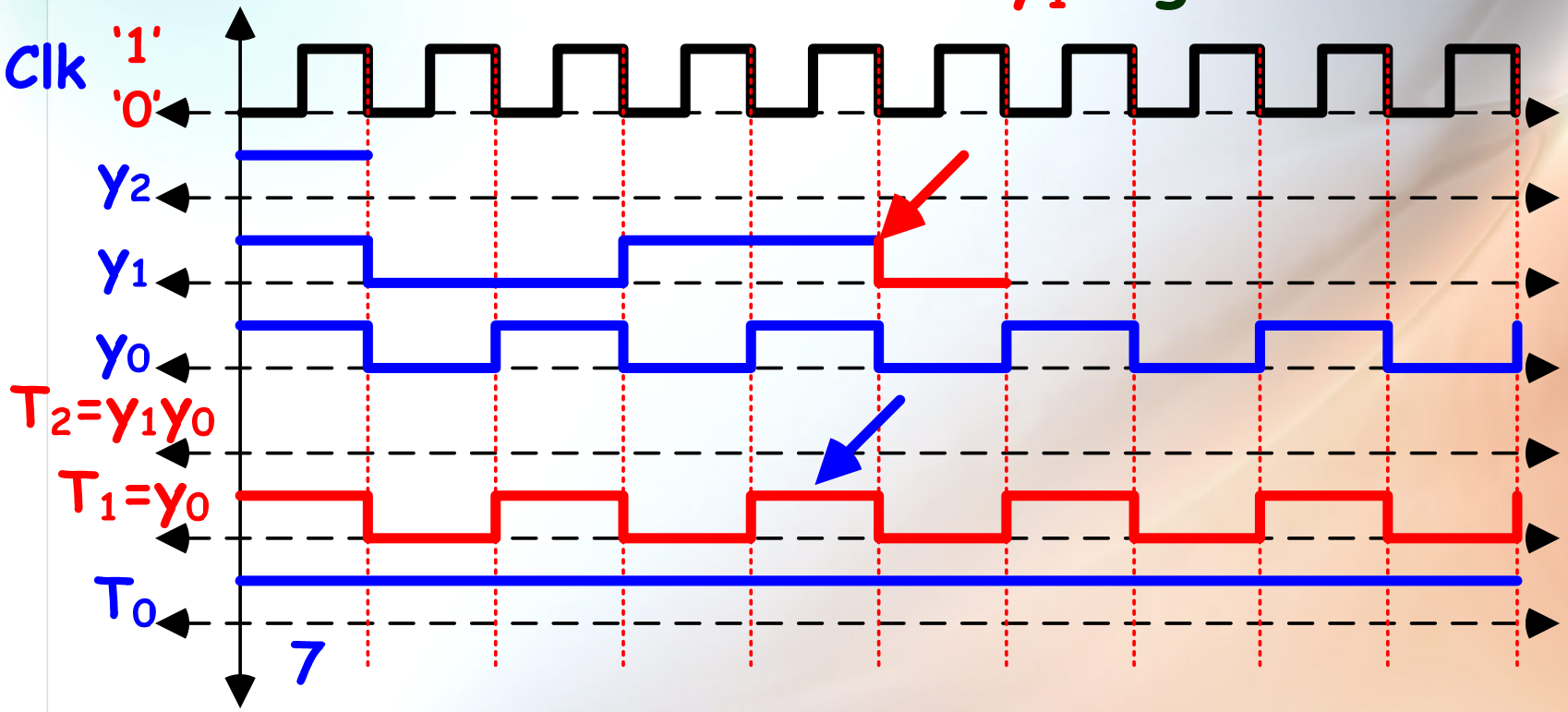
The Timing Diagram

Since y_1 is solely dependent on T_1, we can now create the entire y_1 signal.



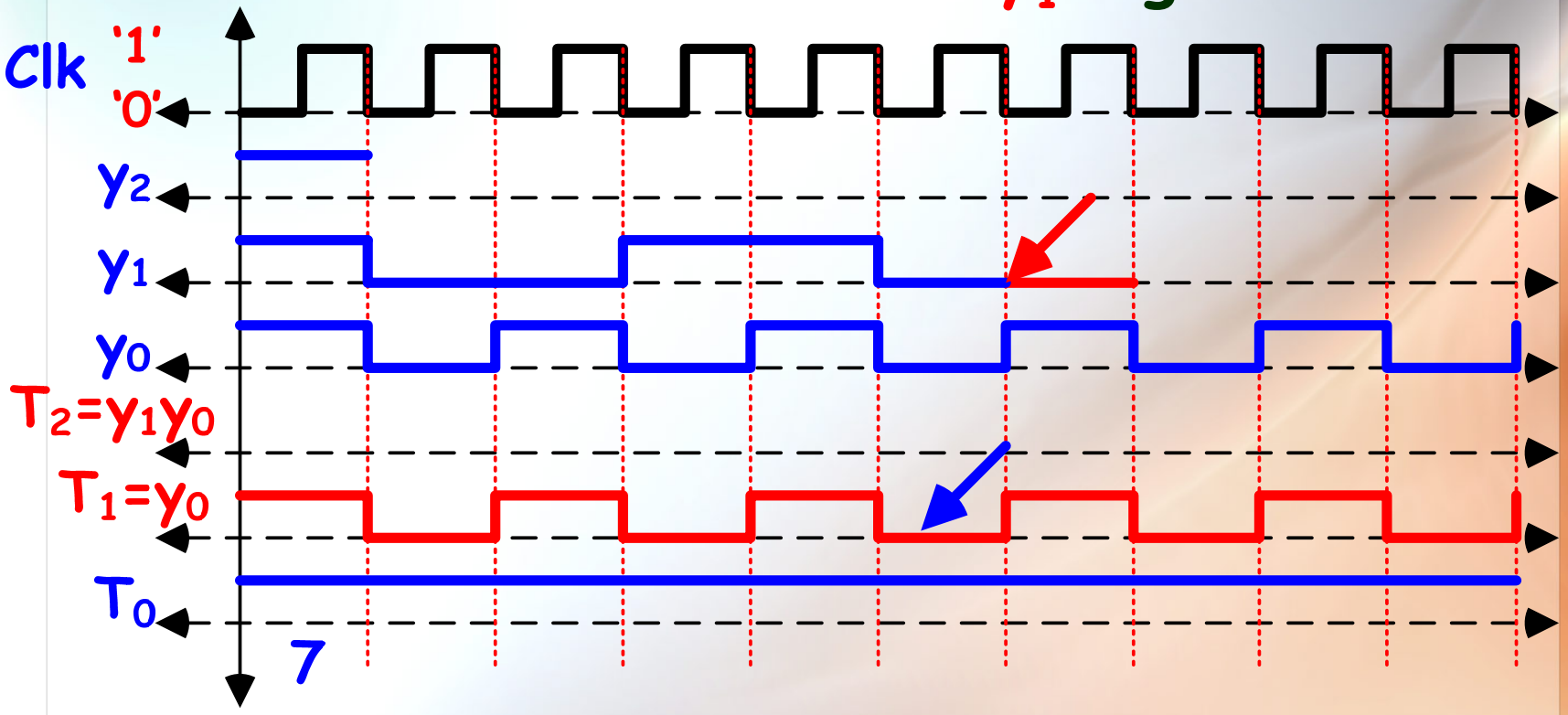
The Timing Diagram

Since y_1 is solely dependent on T_1, we can now create the entire y_1 signal.



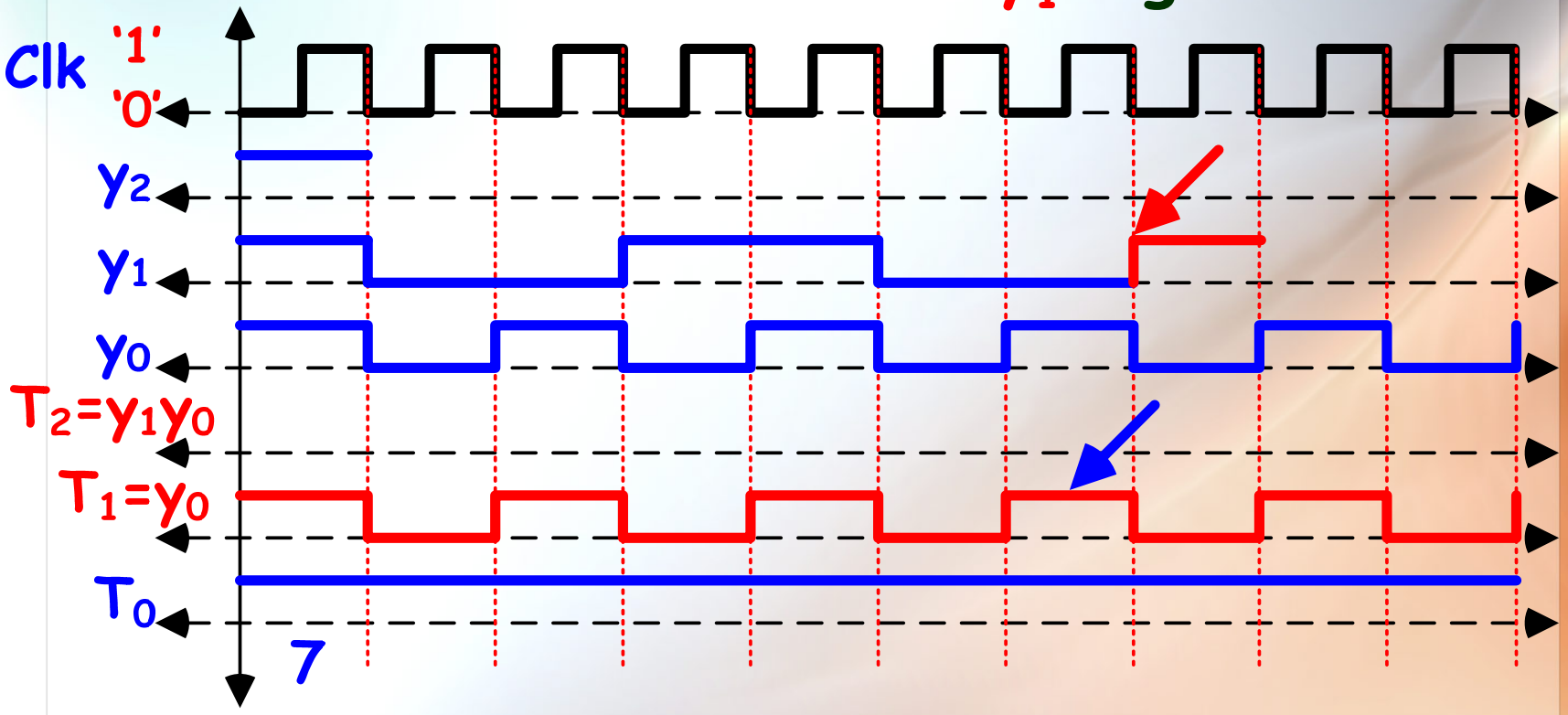
The Timing Diagram

Since y_1 is solely dependent on T_1, we can now create the entire y_1 signal.



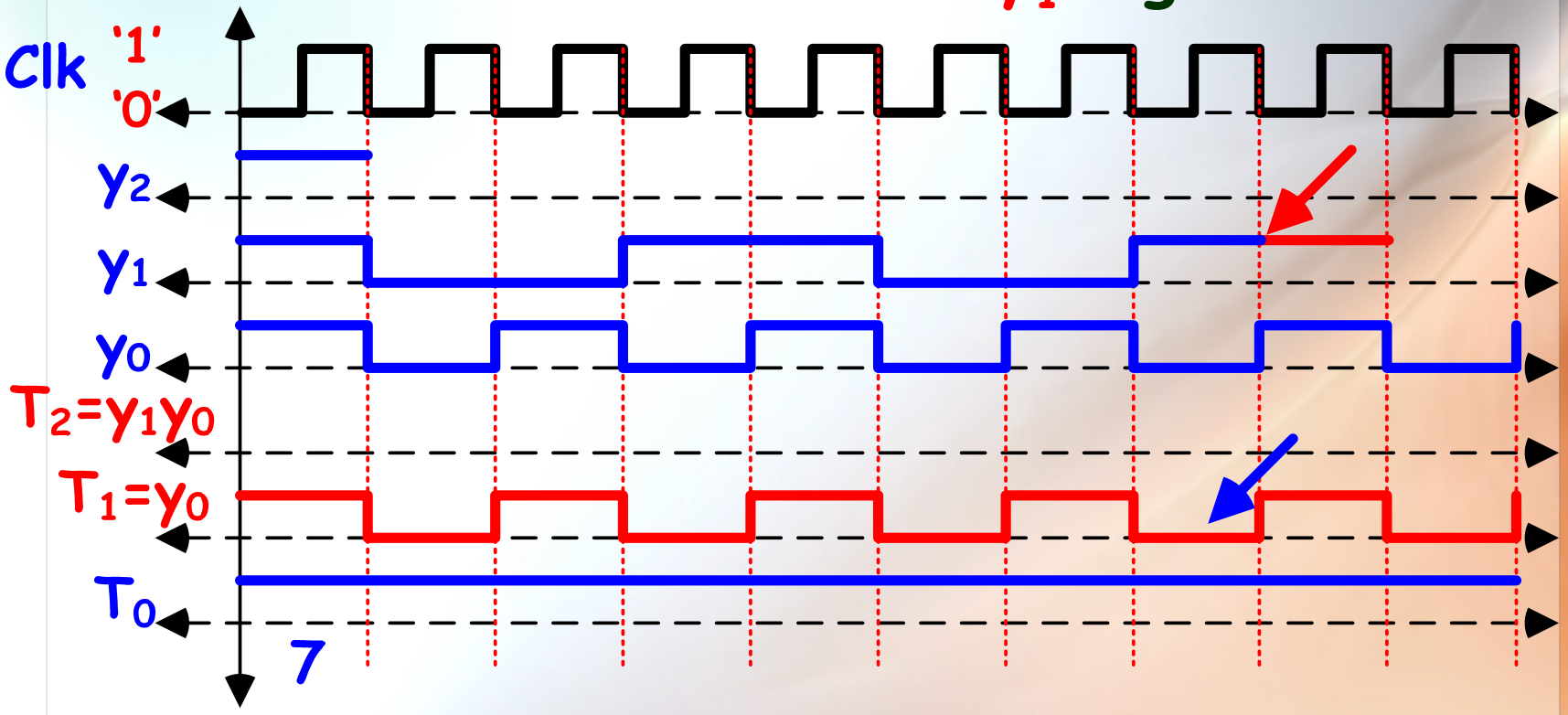
The Timing Diagram

Since y_1 is solely dependent on T_1, we can now create the entire y_1 signal.



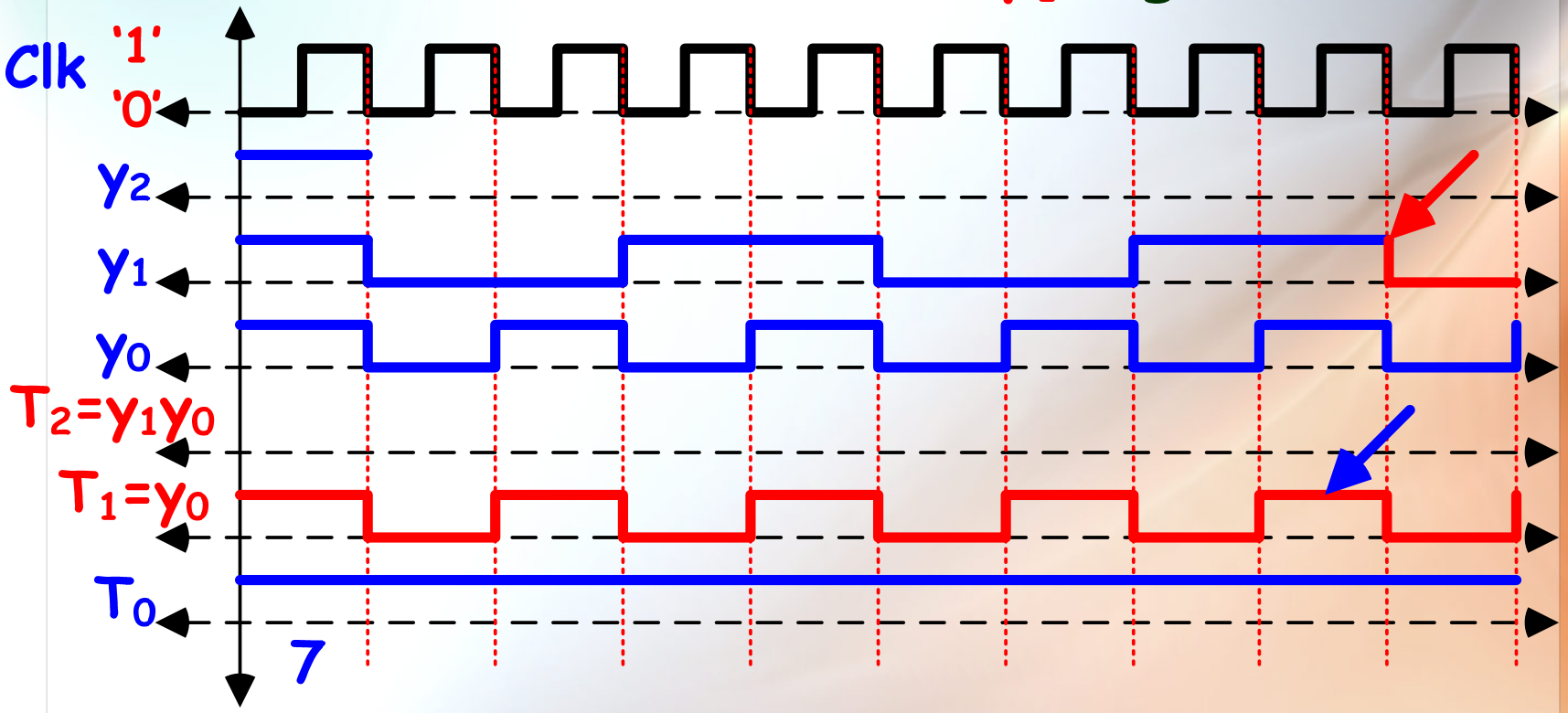
The Timing Diagram

Since y_1 is solely dependent on T_1, we can now create the entire y_1 signal.



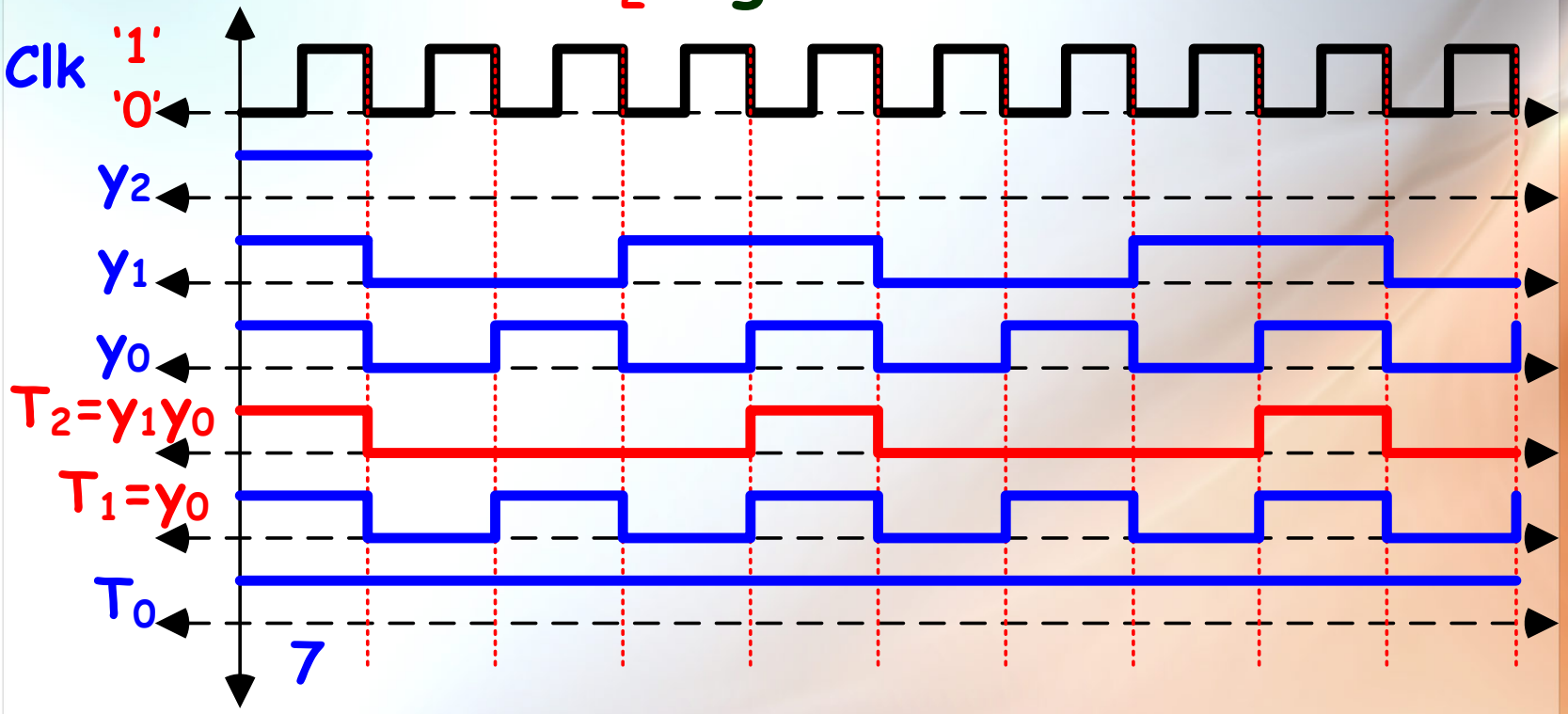
The Timing Diagram

Since y_1 is solely dependent on T_1, we can now create the entire y_1 signal.



The Timing Diagram

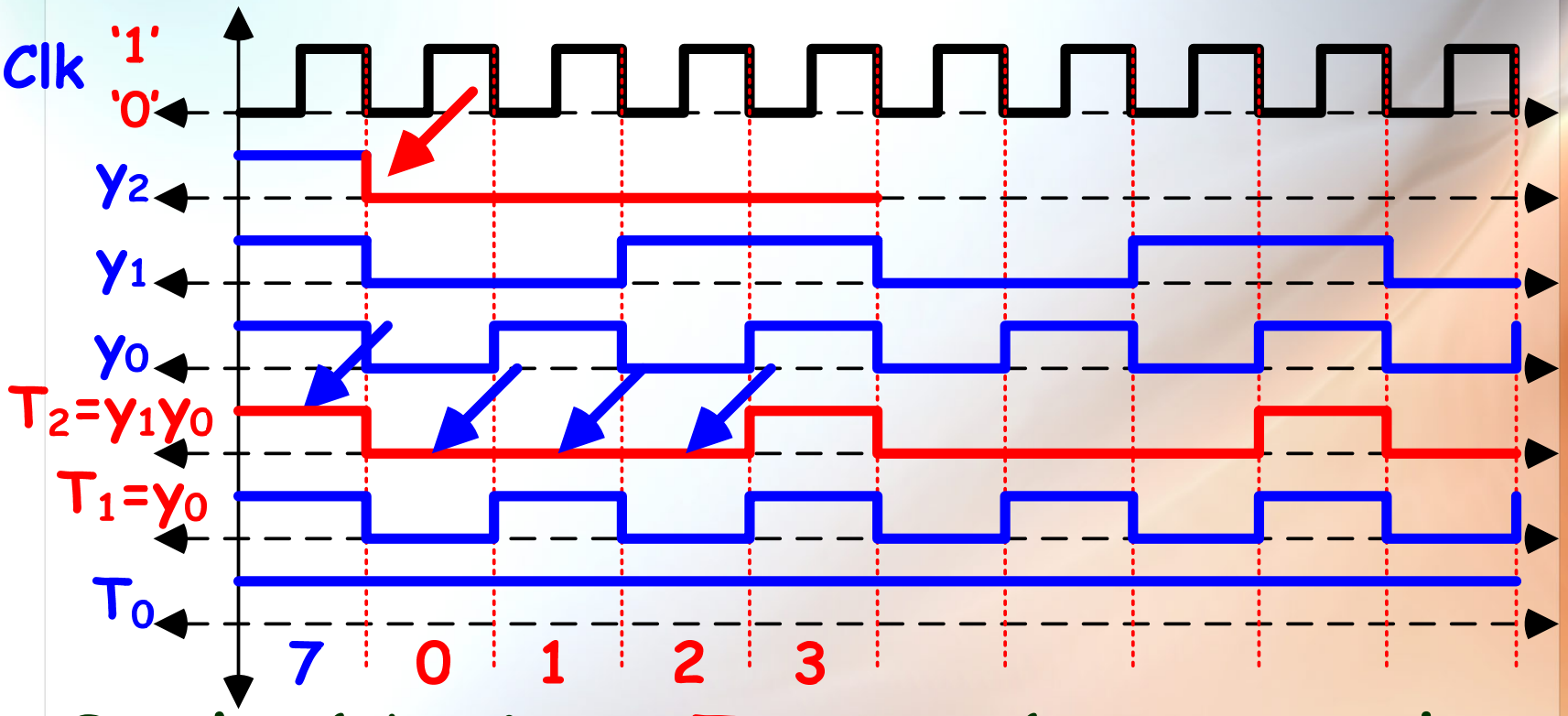
Since y_1 and y_0 are known, we can now determine the T_2 signal.



The only time that T_2 can be high is if both y_1 AND y_0 are high.

The Timing Diagram

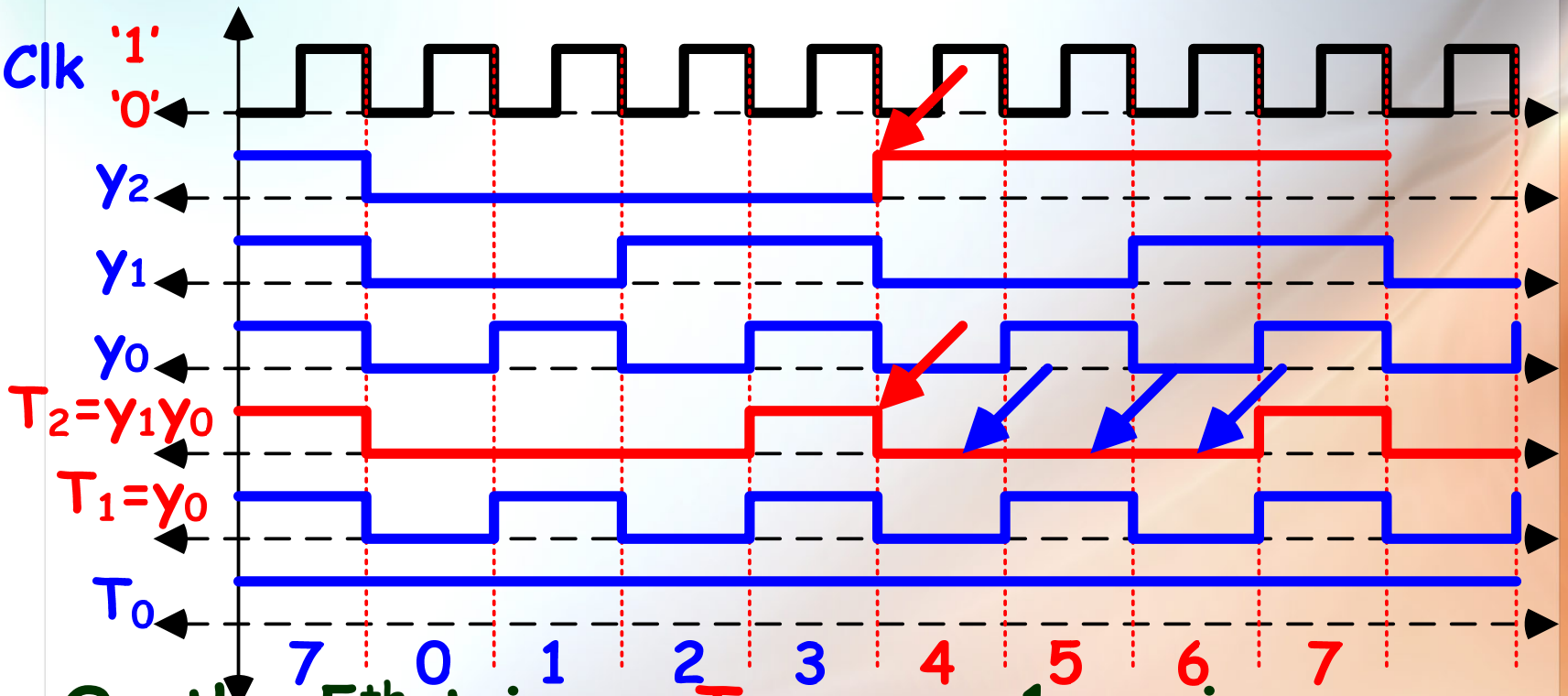
With T_2 known, we can now determine y_2.



On the 1st trigger T_2 was a 1 so y_2 toggles. It stays low for 3 clock cycles because T_2 is low during that time.

The Timing Diagram

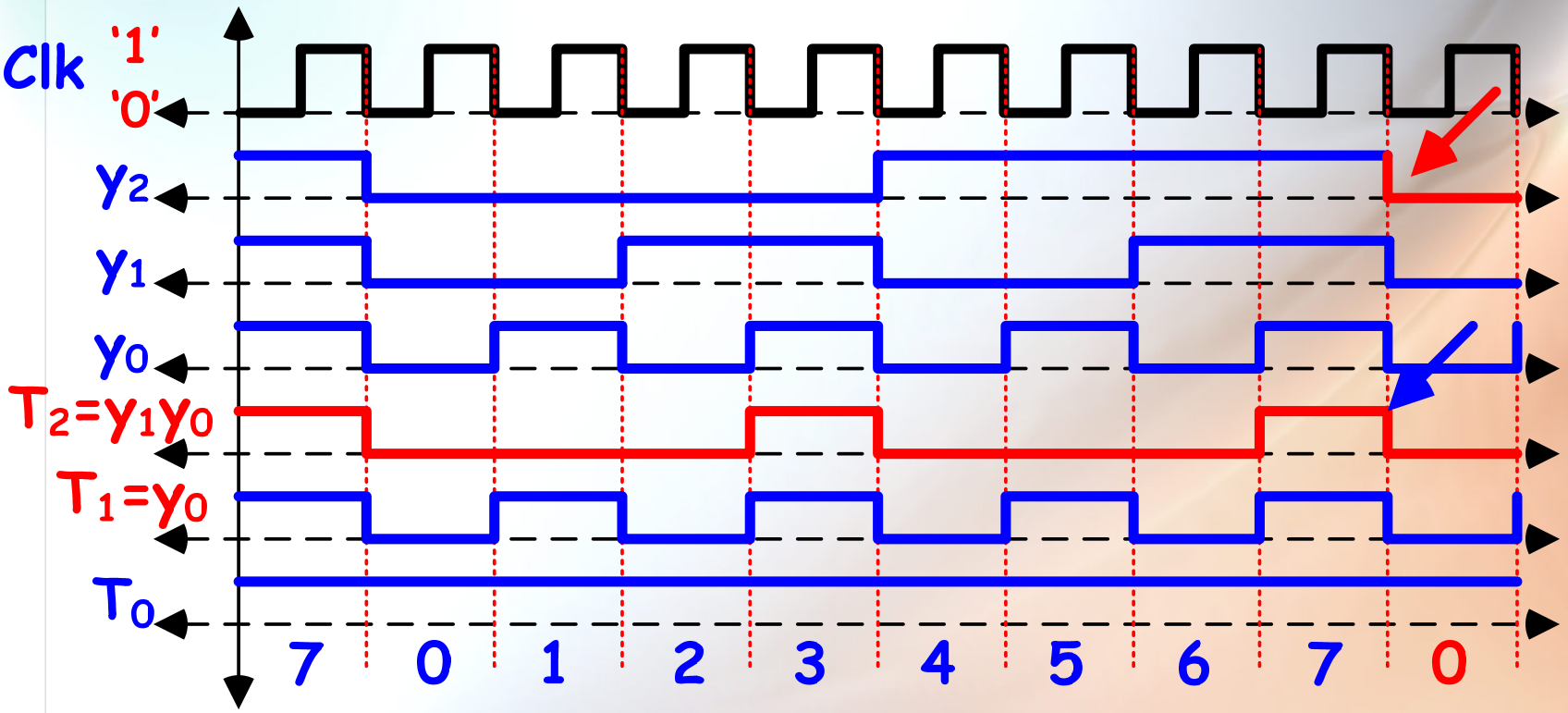
With T_2 known, we can now determine y_2.



On the 5th trigger T_2 was a 1 again so y_2 toggles high. Again it stays there for. 3 clock cycles because $T_2$is low during that time.

The Timing Diagram

With T_2 known, we can now determine y_2.



Finally, T_2 is high again and y_2 toggles low so the state goes back to 000.

The 'Mod-n' Counter

- **A counters Modulus or 'Mod' number is the total number of states which are counted.**

The 'Mod-n' Counter

- A counters Modulus or 'Mod' number is the total number of states which are counted.
- For example, a 4-bit counter will be a Mod-16 counter (unless prevented from being so) since it counts from 0-15.

The 'Mod-n' Counter

- A counters Modulus or 'Mod' number is the total number of states which are counted.
- For example, a 4-bit counter will be a Mod-16 counter (unless prevented from being so) since it counts from 0-15.
- We have seen this before in the form of:

$$2^n = 2^4 = 16$$

$$(n = \#\ \text{of FF's})$$

The 'Mod-n' Counter

- **A counter does not have to be allowed to count up to its maximum possible count however.**

The 'Mod-n' Counter

- A counter does not have to be allowed to count up to its maximum possible count however.
- A circuit designer can add in a decoder circuit to 'decode' the output of the counter and send a reset signal (or Clear) to the counter when the desired count is reached.

Mod-5 example

- **Design a Mod-5 counter using a 4-bit ripple counter.**

Mod-5 example

- Design a Mod-5 counter using a 4-bit ripple counter.
  - First step is to understand that the counter will count from 0 to 4 and then reset on a 5.

Mod-5 example

- Design a Mod-5 counter using a 4-bit ripple counter.
  - First step is to understand that the counter will count from 0 to 4 and then reset on a 5.
  - Since all states above a 5 will never be allowed to occur, the designer can use them as 'Don't Cares'.

Mod-5 example

- Design a Mod-5 counter using a 4-bit ripple counter.
  - First step is to understand that the counter will count from 0 to 4 and then reset on a 5.
  - Since all states above a 5 will never be allowed to occur, the designer can use them as 'Don't Cares'.
  - The "active-level" of the reset or clear needs to be known. In this case we are using a JK FF with active-low CLR's.

Mod-5 example

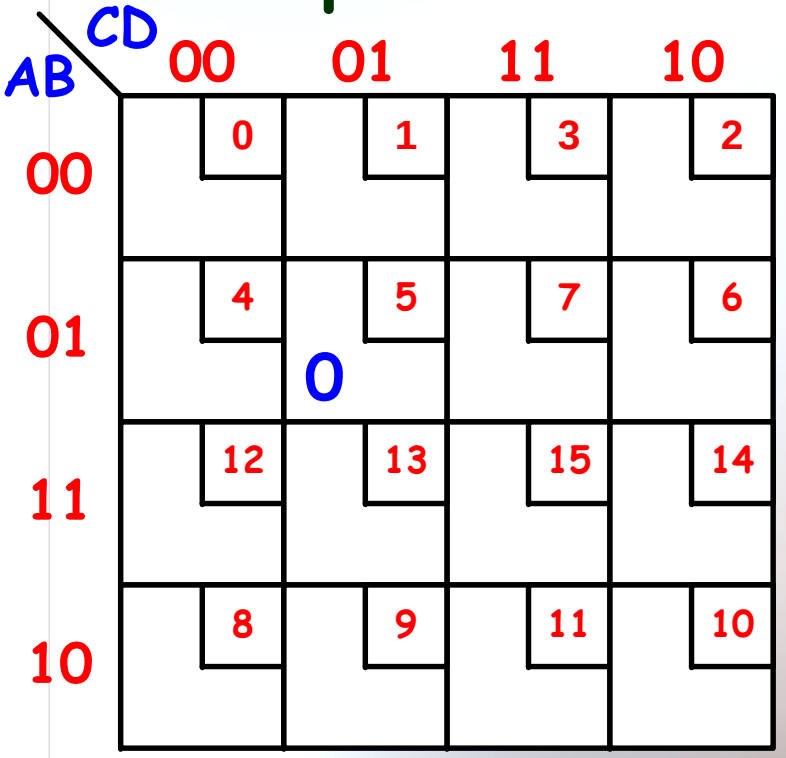
- **Design a Mod-5 counter using a 4-bit ripple counter.**
  - **Create a 4-bit K-map.**

Mod-5 example

- Design a Mod-5 counter using a 4-bit ripple counter.
  - Create a 4-bit K-map.
  - We are looking for an active-low signal to reset the circuit then we are going to be looking for a 0 out when a count of 5 is reached.

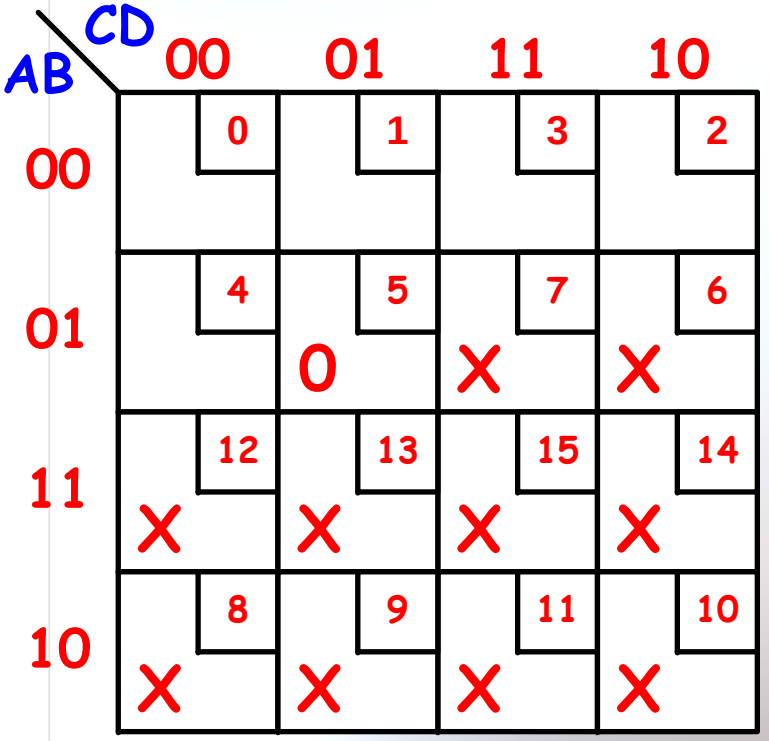
Mod-5 example

- **Design a Mod-5 counter using a 4-bit ripple counter.**
  - **So, a 0 is placed in Cell 5 of the k-map.**



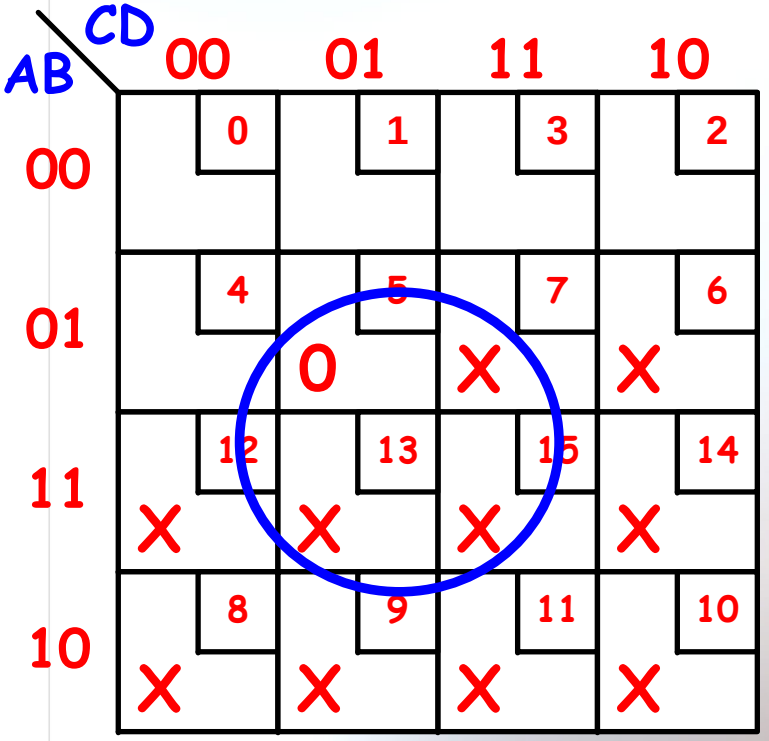
Mod-5 example

- Design a Mod-5 counter using a 4-bit ripple counter.
  - Since 6 thru 15 will never occur, we add in the don't cares



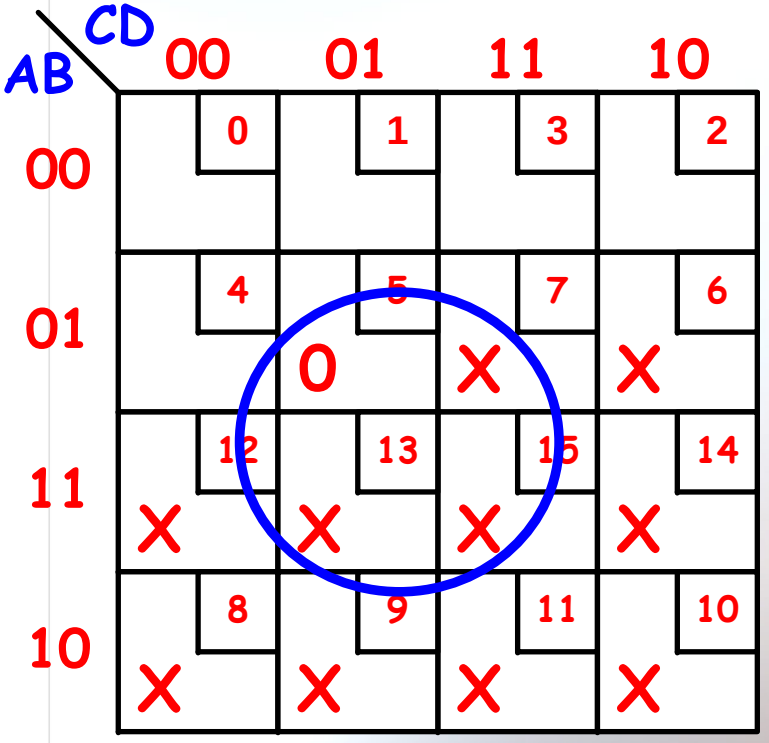
Mod-5 example

- **Design a Mod-5 counter using a 4-bit ripple counter.**
  - **Now group the answer.**



Mod-5 example

- **Design a Mod-5 counter using a 4-bit ripple counter.**
  - **And the answer is:**

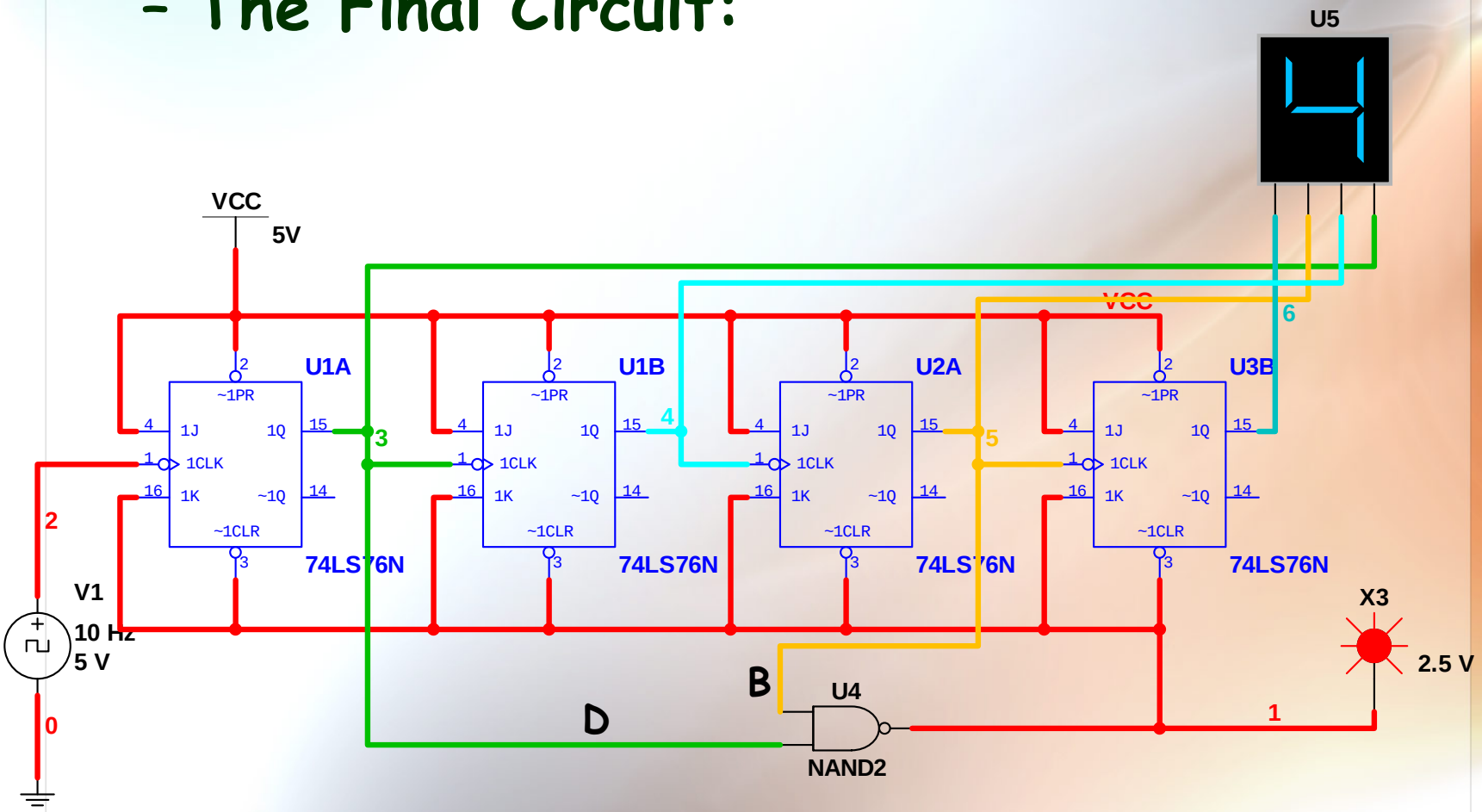


$$f(A,B,C,D) = \overline{B} + \overline{D}$$

$$= \overline{B\overline{D}}$$

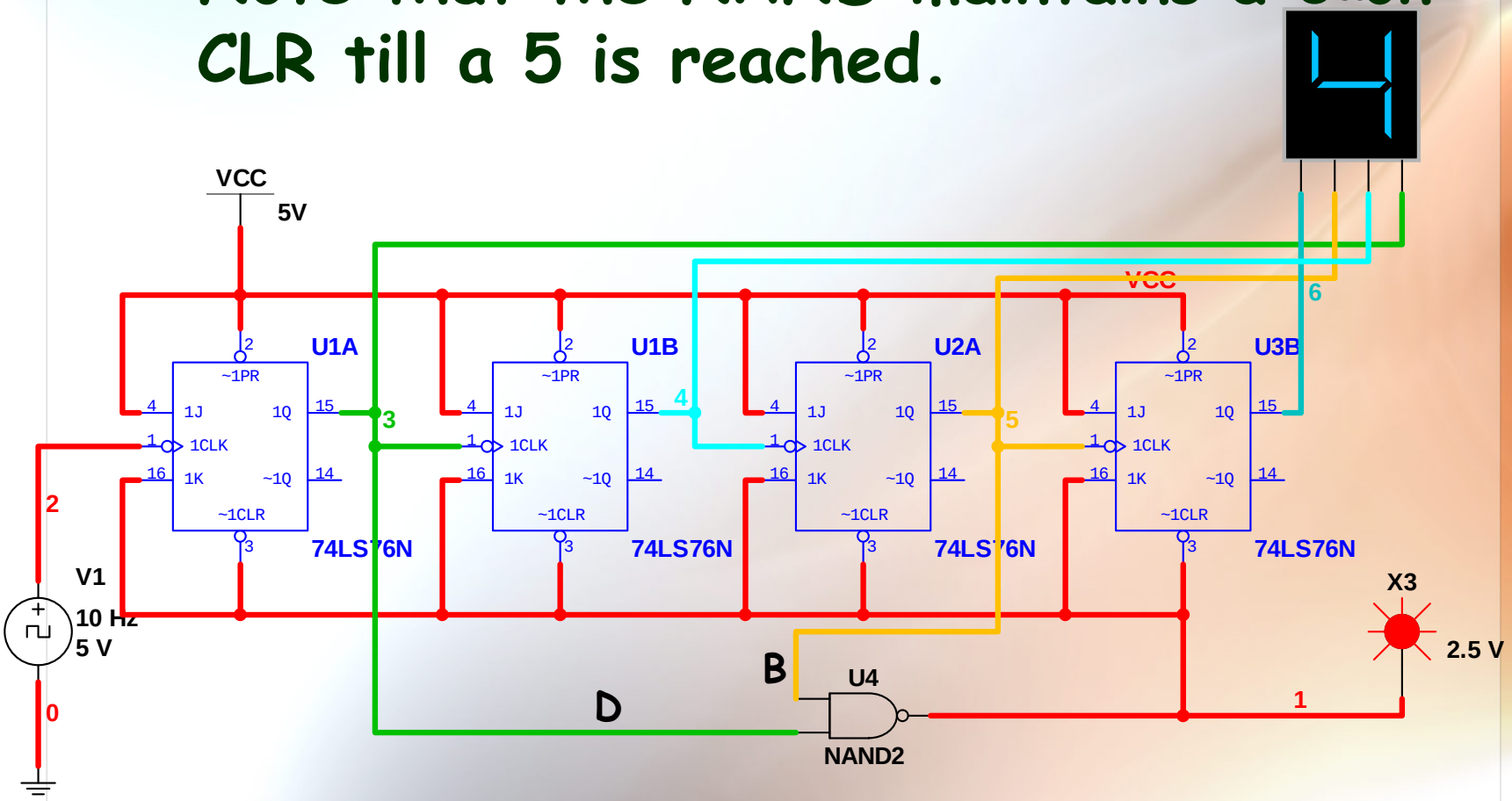
Mod-5 example

- **Design a Mod-5 counter using a 4-bit ripple counter.**
  - **The Final Circuit:**



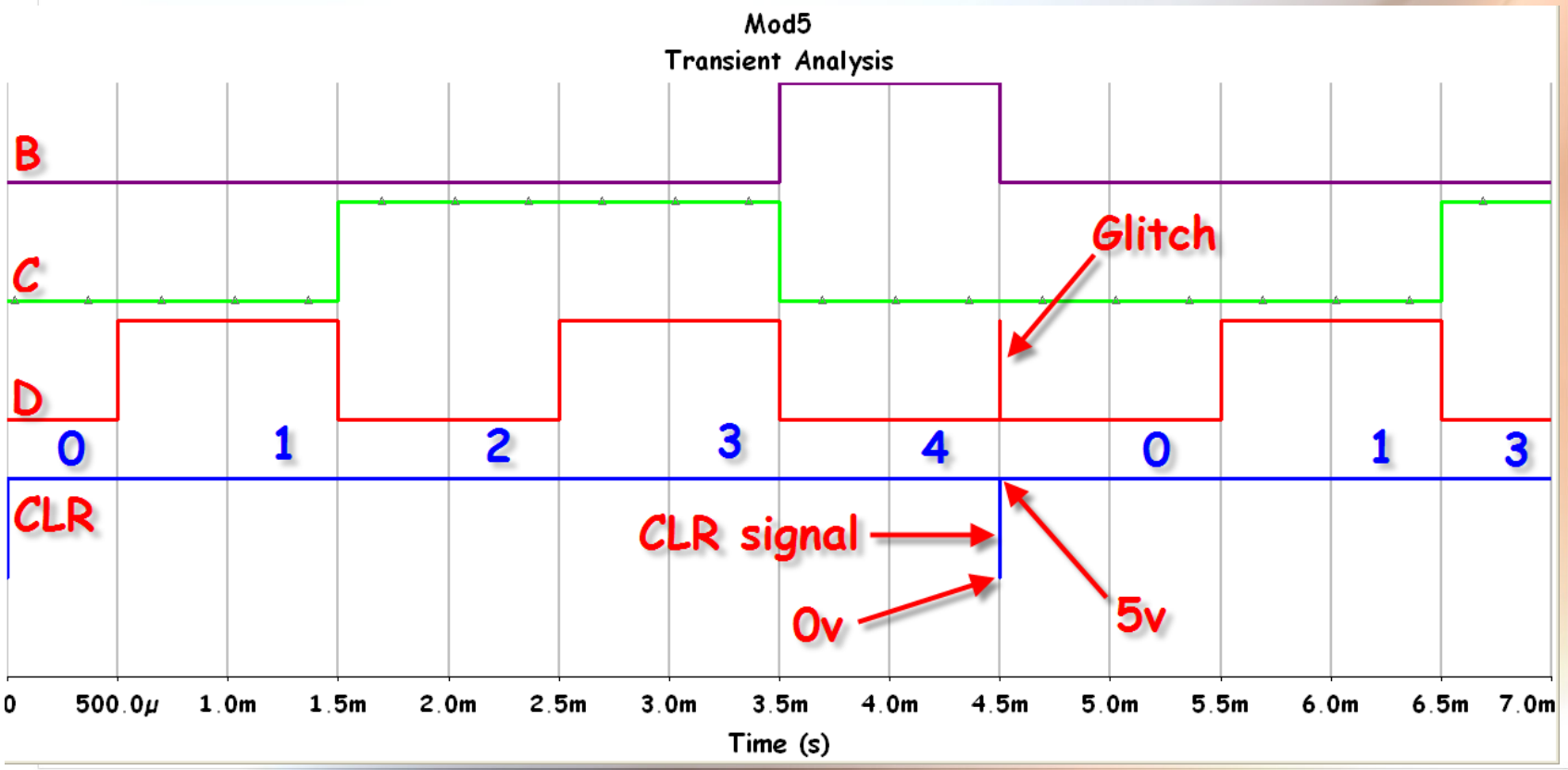
Mod-5 example

- Design a Mod-5 counter using a 4-bit ripple counter.
  - Note that the NAND maintains a 0 on CLR till a 5 is reached.



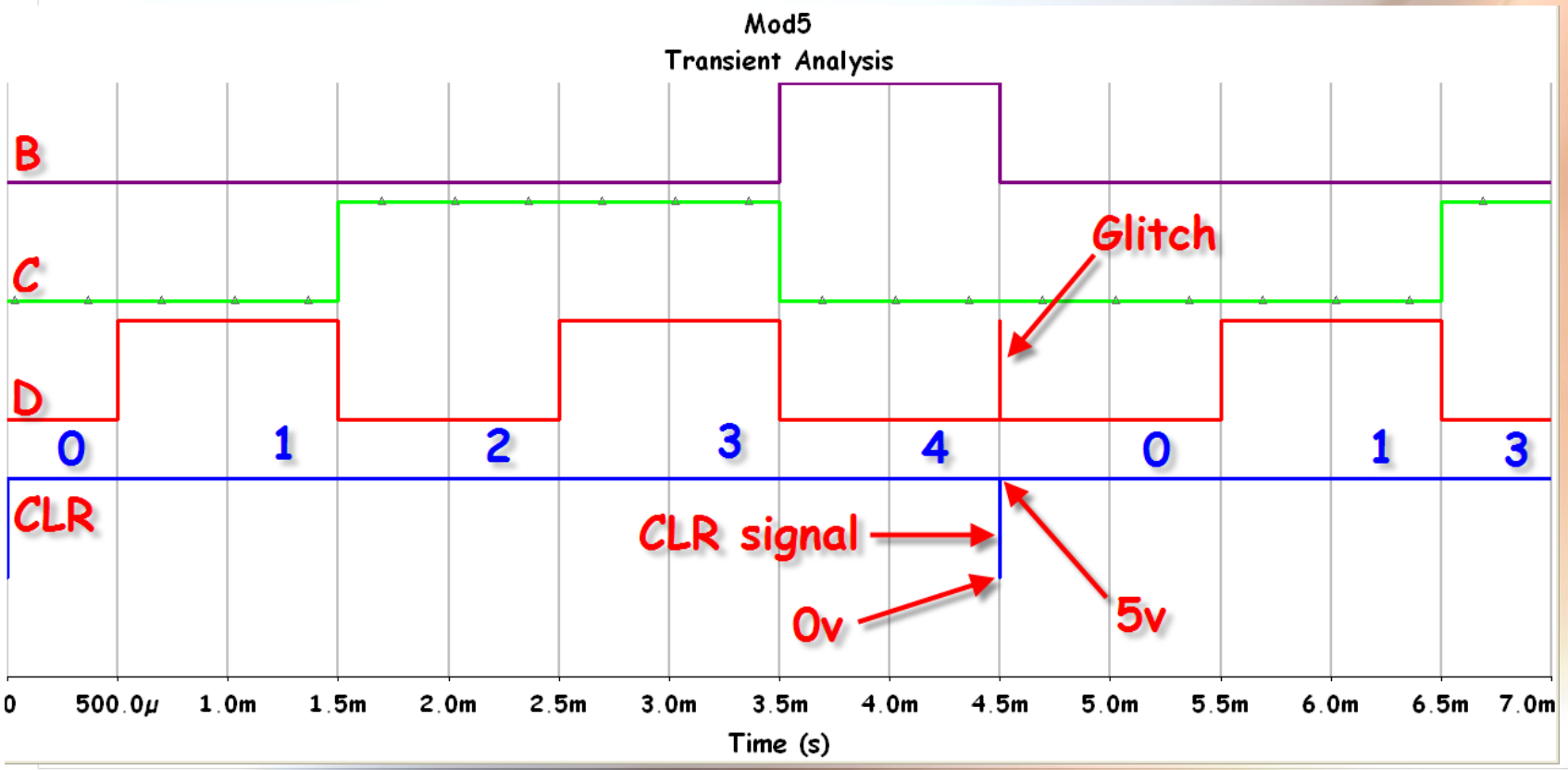
Mod-5 example

- **The Timing diagram:**
  - **Shows the count from 0 – 4, resetting on 5 and starting over.**



Mod-5 example

- **The Timing diagram:**
  - **Note the GLITCH which quite often is a result of this type of MOD-Counter design.**



Mod-n design warning

- Be aware that some counters have a delay between a count and the reset.

Mod-n design warning

- Be aware that some counters have a delay between a count and the reset.
- For instance, lab 7 in EET 315 has a circuit with a D-FF in the reset circuitry. This causes a delay of 1 clock pulse so a different value is needed when you are looking for the count value to decode.

Synchronous Circuit Disadvantage

- Because of the **propagation delay** of the gates which are supplying the control signals to each FF input, there can be delays which cause state errors.

Synchronous Circuit Disadvantage

- Because of the **propagation delay** of the gates which are supplying the control signals to each FF input, there can be delays which cause state errors.
- The problem gets even worse if the **propagation delay** seen by each input is different from the other inputs.

Synchronous Circuit Disadvantage

- Because of the **propagation delay** of the gates which are supplying the control signals to each FF input, there can be delays which cause state errors.
- The problem gets even worse if the **propagation delay** seen by each input is different from the other inputs.
- These problems can cause **state glitches. (spurious states between the actual states)**