NOTES ON SHIFT REGISTERS AND COUNTERS

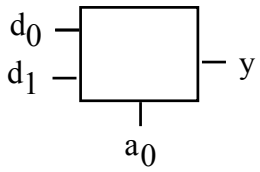
NOTES 1

NOTES ON SHIFT REGISTERS AND COUNTERS

These notes introduce the multiplexer and its uses for selecting signals and in general combinational logic design. They also introduce shift registers and asynchronous and synchronous counters. The introductory concepts of synchronous sequential network design are presented through the design of a counter whose output sequence is in non-numerical order.

A. Multiplexers

A two-line digital multiplexer is a circuit with three inputs d_1, d_0, a_0 and a single output y. The inputs d_1 and d_0 are called the data-inputs while the input a_0 is the selector input.



Two-line to one-line multiplexer.

Word statement: A digital multiplexer is the electronic equivalent of a rotary switch. The selector input(s) is (are) the binary encoding of an integer i, and the output y agrees with data line d_i.

C.2

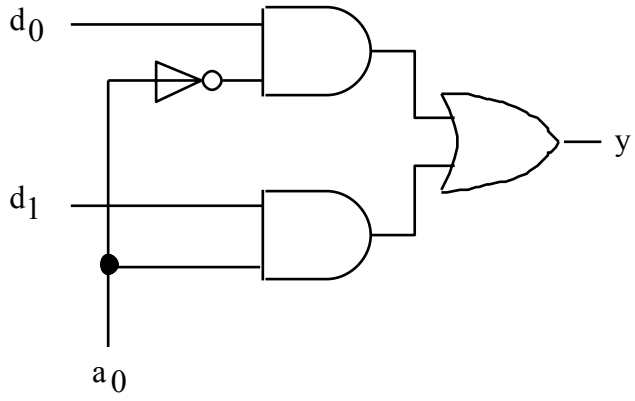
Truth table:

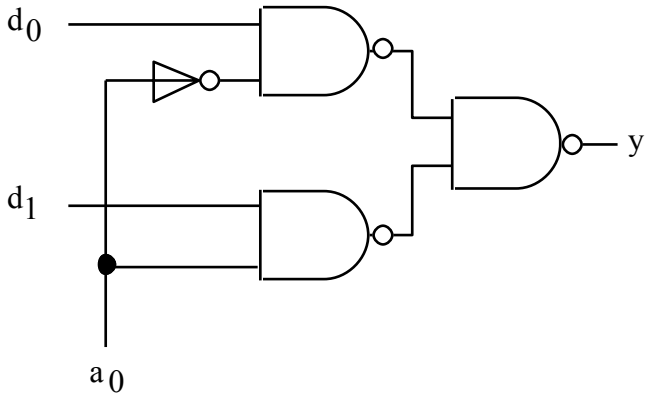
d_1	d_0	a_0	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Logical expressions: $y = d_0\bar{a}_0 + d_1a_0$

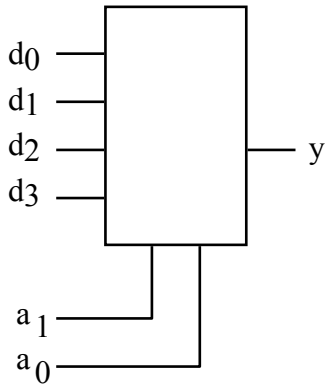
$$y = \overline{\left(\overline{d_0\bar{a}_0}\right)\left(\overline{d_1a_0}\right)}$$

Circuit diagrams:





A four-line to one-line multiplexer is a combinational circuit with six inputs d_3, d_2, d_1, d_0, a_1, a_0 and a single output y.



Four-line to one-line multiplexer

Word description: The selector inputs a_1, a_0 are the binary encoding of an integer i between 0 and 3, and the output y agrees with data input d_i. In other words the selector inputs specify which data line is "connected" to the output y.

Truth table: Since this is a 6 input function there would be $2^6 = 64$ lines in a truth table. To avoid writing a 64 line truth table, a representation which is a mixture of a truth table and a logical expression is often used. Strictly speaking it is not a truth table. Only some of the input variables are listed on the "input" side, and on the "output" side a logical expression of the remaining variables is listed (rather than 0's and 1's as in a truth table).

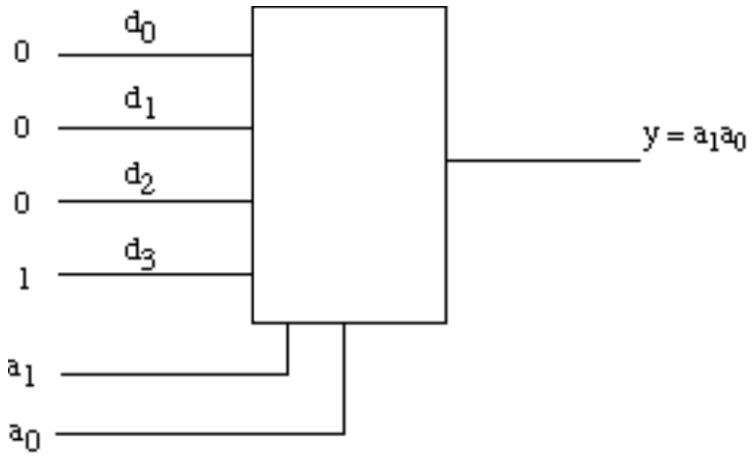
selector inputs a_1 a_0	output y
0 0	d_0
0 1	d_1
1 0	d_2
1 1	d_3

or

a_1 a_0	y
0 0	agrees with d_0
0 1	agrees with d_1
1 0	agrees with d_2
1 1	agrees with d_3

Mixed "truth table" for four-line to one-line multiplexer.

Note that if we wire $d_0 = d_1 = d_2 = 0$ and $d_3 = 1$, the output y is the logical AND of the selector inputs a_1 and a_0.



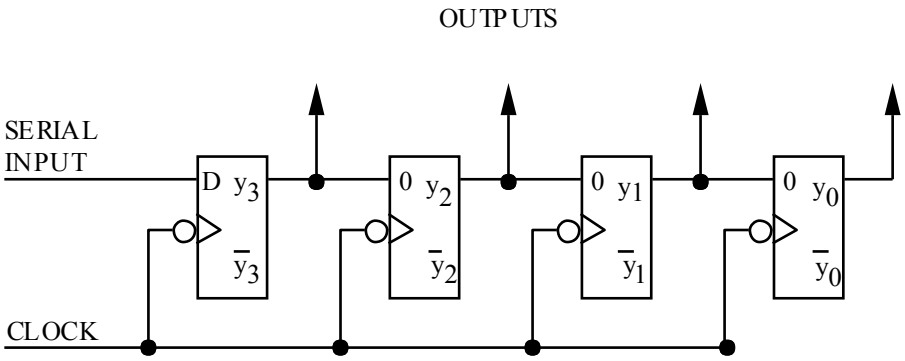
Wiring four-line to one-line multiplexer to realize the AND function.

Note that, in general, if one could connect arbitrary functions of a particular variable, x, namely 0, 1, x, or $\overline{x}$, to the d inputs, any arbitrary 3 variable (a, a_o, x) function could be implemented using a 4:1 multiplexer.

B. Shift Registers

A register is an array of flip-flops which may be used to store information in the form of a binary string. Each flip-flop stores a single bit (binary digit) of the string.

In a shift register each flip-flop transfers its binary value to its right neighbor upon receiving a clock pulse. Whenever such a shift is made, the value in the rightmost flip flop is lost (it "falls off the end") and a serial input provides a new binary value for the leftmost flip-flop. A shift register can be constructed from D flip-flops as shown below. Note: We do not consider shifting to the left here.



Basic Shift Register

At each clock pulse, the binary values are shifted one position to the right, and the leftmost flip-flop assumes the values of the serial input. Most computers have several different right shift instructions. In a logical right shift, a logical 0 is shifted into the leftmost position and the value in the rightmost position is shifted out and lost. In a circular right shift, the value in the rightmost position is circularly returned and shifted into the leftmost position rather than being discarded. In an arithmetic right shift, the contents of the register are interpreted as a 2's complement integer representation. The number in the register after a shift should be half the number before the shift. Actually there may be a truncation error in the result since dividing an odd number by 2 will leave a fractional part of 1/2 which is lost. When the initial contents are positive, a 0 is shifted into the leftmost position. However, when the initial contents are negative, determining the binary value to be shifted into the leftmost position requires some thought – shifting a 0 in would be incorrect since presumably a negative number was divided by 2 yet the result looks positive because of the leading 0.

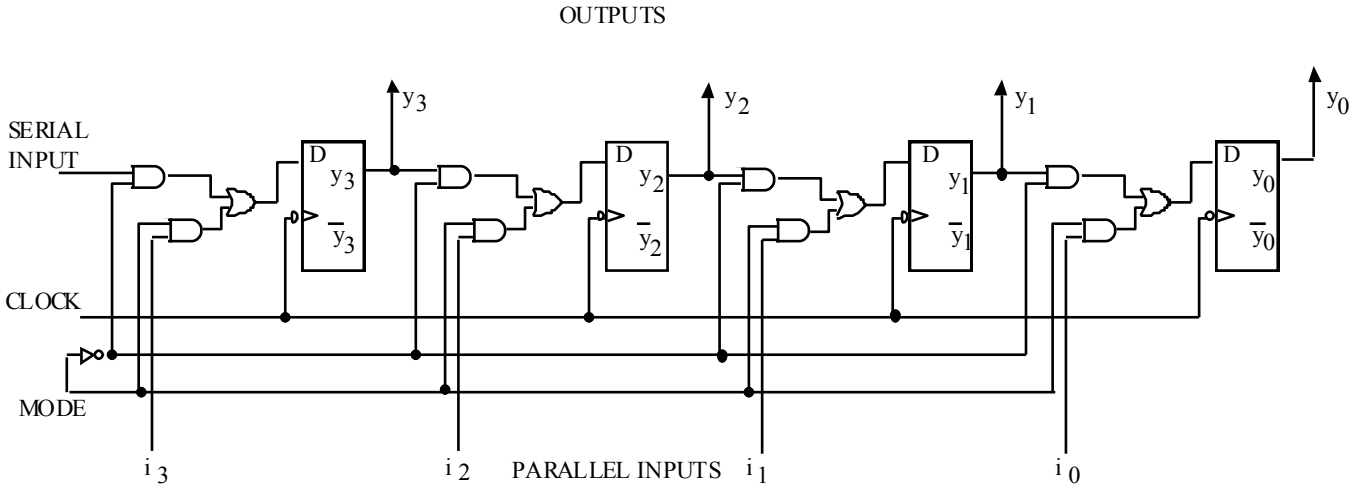
LOGICAL RIGHT SHIFTS	CIRCULAR RIGHT SHIFTS	ARITHMETIC RIGHT SHIFTS
$0110 \rightarrow 0011$	$0110 \rightarrow 0011$	$0110 \rightarrow 0011$ $(6 \rightarrow 6/2 = 3)$
$0011 \rightarrow 0001$	$0011 \rightarrow 1001$	$0011 \rightarrow 0001$ $(3 \rightarrow 3/2 = 1)$
$1100 \rightarrow 0110$	$1100 \rightarrow 0110$	$1100 \rightarrow 1110$ $(-4 \rightarrow -4/2 = -2)$

Similarly there are three types of left shift. The logical left shift and the circular left shift are the obvious modifications of the corresponding right shifts. In an arithmetic left shift, the result is twice the initial contents. Determining the value shifted into the least significant bit requires some thought for the positive and negative cases. An additional complication arises in the arithmetic left shift. Since the initial contents are multiplied by 2, the result may be too large to store in the register. In most computers, the shift is performed anyway and a special overflow flip-flop is set to 1 to indicate that the register contents are incorrect – the result was too large for the register and it overflowed. It is then the user's responsibility to check the overflow flip-flop after every shift to make sure he didn't tell the computer to do something dumb.

LOGICAL LEFT SHIFTS	CIRCULAR LEFT SHIFTS	ARITHMETIC LEFT SHIFTS
$0011 \rightarrow 0110$	$0011 \rightarrow 0110$	$0011 \rightarrow 0110$ $(3 \rightarrow 2 \bullet 3 = 6)$
$0110 \rightarrow 1100$	$0110 \rightarrow 1100$	$0110 \rightarrow 1100$ $(6 \rightarrow 2 \bullet 6 = -4)$ OVERFLOW
$1100 \rightarrow 1000$	$1100 \rightarrow 1001$	$1100 \rightarrow 1000$ $(-4 \rightarrow 2 \bullet -4 = -8)$

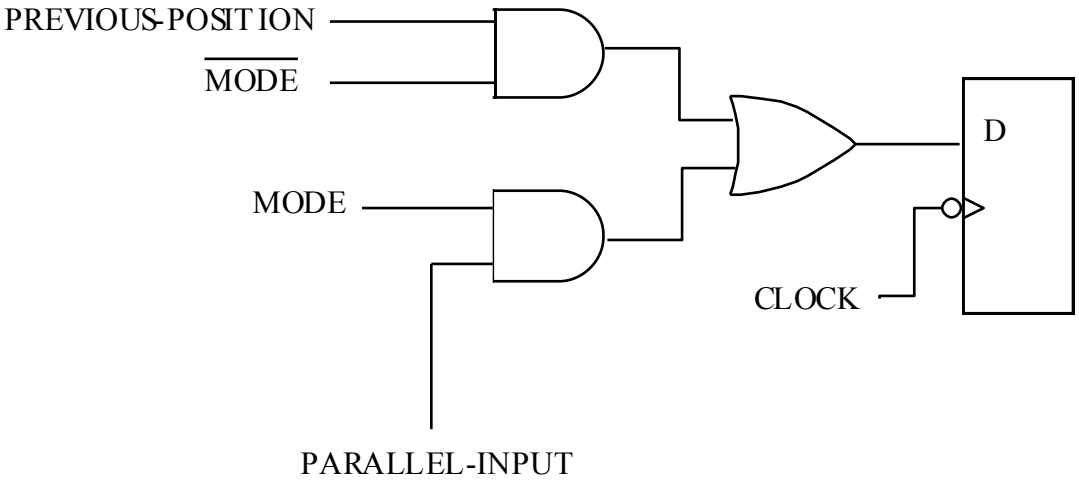
Parallel-load Shift Register

Some shift registers have additional logic which permits all flip-flops to be loaded simultaneously in parallel with a single clock pulse.

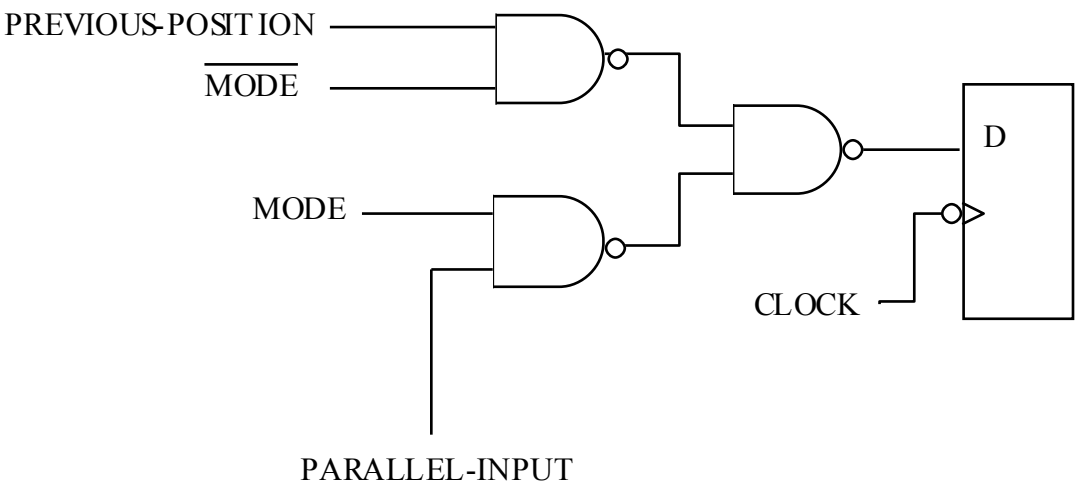


Parallel-load Shift Register

To analyze the parallel-load shift register shown above, consider the networks of two AND gates feeding an OR gate on the D flip-flop inputs.



Control Inputs for Parallel-load Shift Register



Control Inputs for Parallel-load Shift Register (NAND Version)

When the MODE signal is low, the lower AND (or NAND) gate is blocked and the D flip-flop input value is simply PREVIOUS-POSITION. Thus when the flip-flops are clocked, the register shifts right in the normal manner. When the MODE signal is high, the upper AND (or NAND) gate is blocked and the D flip-flop input value is PARALLEL-INPUT. When the flip-flops are clocked, each flip-flop is loaded from its parallel input.

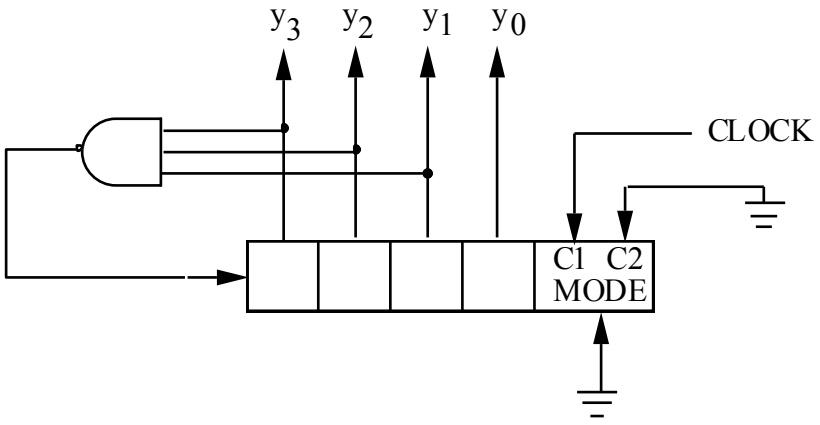
When actually building the circuit, the two AND gates feeding an OR gate would be replaced by the equivalent two NAND gates feeding a NAND gate, as shown above. It is fairly common to use a 2-level NAND network instead of a 2-level AND-OR network because it is more convenient to stock just one type of gate, and NAND gates are more readily available than AND gates and OR gates. Equivalently, a 2:1 multiplexer can be used for the input logic with the MODE signal connected as a select input.

Ring counter. A n-bit ring counter is a circuit with n outputs of which exactly one is "active" at any time. Each of the outputs has a turn at being "active" exactly once in each n clock pulses. If "active" is interpreted as meaning logical 1, the output sequence of a 4-bit ring counter with outputs y_3,y_2,y_1,y_0 would be

$y_3y_2y_1y_0$

1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
1 0 0 0
0 1 0 0

Sequence Generators. Feedback connections from a shift register's outputs to its serial input can be used to produce different output sequences.



An Example of a Sequence Generator Circuit

If we assume that the shift register above initially contains $y_3y_2y_1y_0 = 1101$, the output sequence generated is the following:

y_3	y_2	y_1	y_0
1	1	0	1
1	1	1	0
0	1	1	1
1	0	1	1
1	1	0	1

•
•
•
•

Once a shift register state is repeated, the output sequence must start to repeat. The period, the number of clock pulses before the output sequence repeats, is 4 for the example. Here, "active" means logical 0. Some mathematical techniques for analyzing sequence generators are developed in ECE/Math 319, Applied Modern Algebra.

C. Counters

The toggle is a synchronous logic element whose binary output (state) complements each time it is clocked. It is readily verified that a toggle can be constructed from a JK flip-flop by wiring J=K=1 as shown in Figure 2.

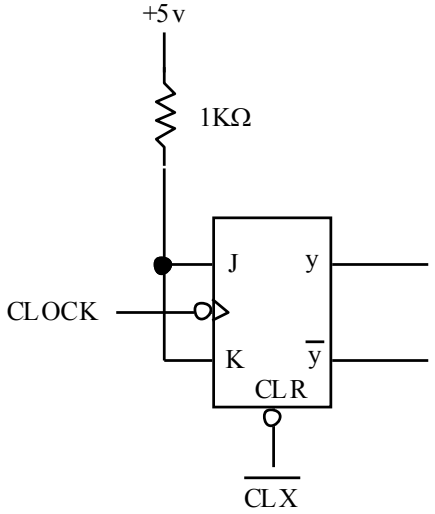


Figure 1. Toggle Constructed from JK Flip-Flop

Asynchronous Counters. In asynchronous counters (ripple counters) there is not a common system clock to every flip-flop. In the 4-bit ripple counter shown in Figure 2, the output of each stage is used as the clock for the next most significant stage.

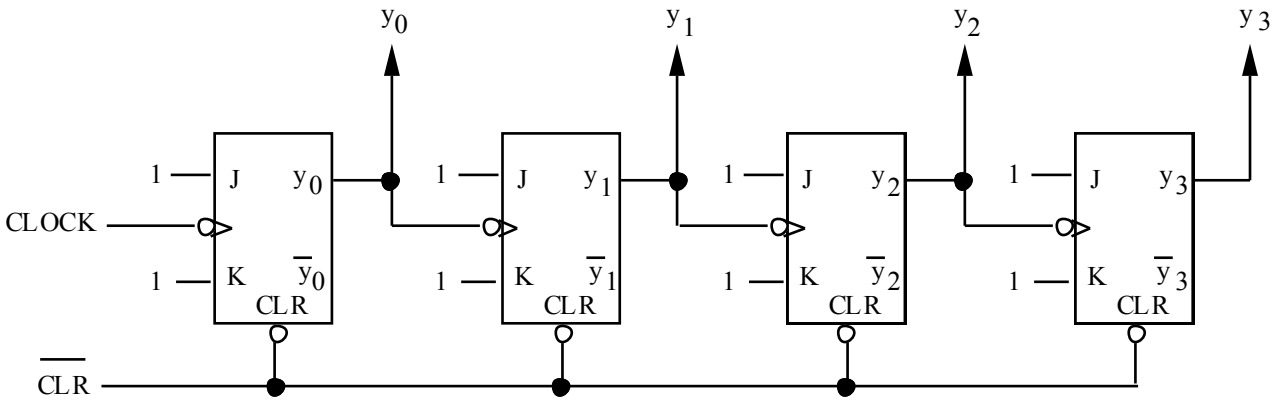


Figure 2. Ripple Counter

The least significant bit, y_0, changes with every clock pulse. Since y_0 is the"clock" for the toggle of the next most significant stage, y_1 changes on the negative transition every time y_0 goes from 0 to 1 and back to 0. More generally since y_i is the clock for the toggle in the $i+1^{st}$ stage, y_{i+1} changes on the negative transition every time y_i goes from 0 to 1 and back to 0. An asynchronous counter of any length can be constructed by using the output of the i^{th} stage as the clock for the toggle of the $i+1^{st}$ stage.

Consider the situation in which the state is 1111 and a clock pulse is received. Since a flip-flop requires a small but finite time to change states, y_0 changes to 0 shortly after the clock pulse, y_1 changes to 0 shortly after y_0 drops, y_2 changes to 0 shortly after y_1 drops, … , and the individual bit changes "ripple" from least significant position to most. The time taken by a state transition depends upon the size of the counter (number of positions) as well as the particular state transition to be made.

Synchronous Counters. In synchronous counters all flip-flops are clocked by the same system clock. Hence all flip-flops make their transitions to their next states at the same time and there is no "ripple." Figures 3 and 4 show examples of the two basic types of synchronous counters which produce state sequences in numerical order.

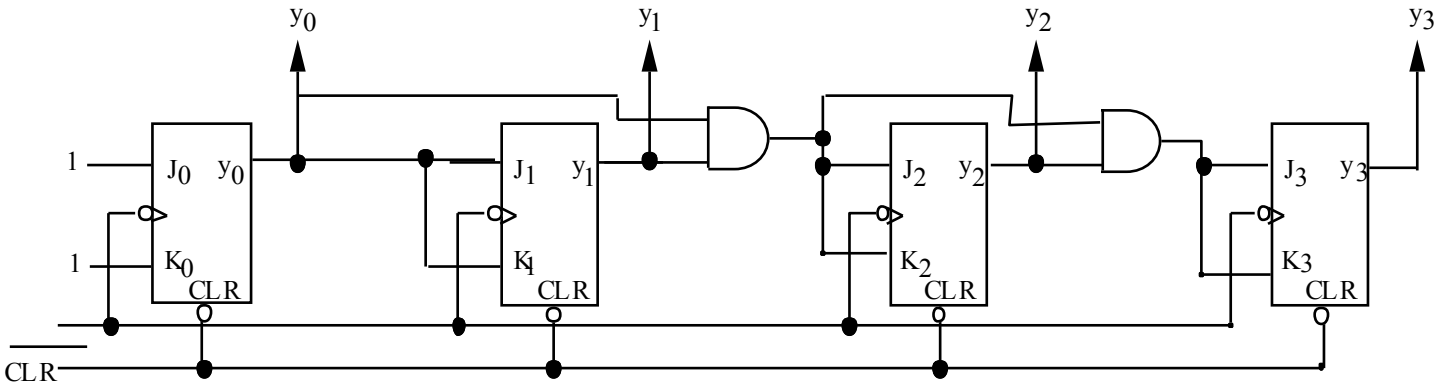


Figure 3. Serial Synchronous Counter

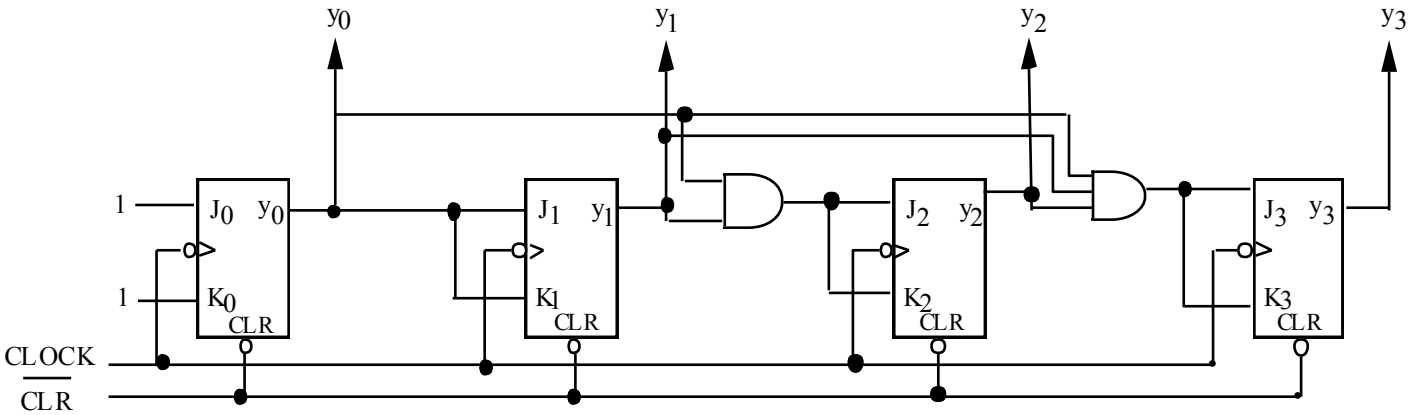


Figure 4. Parallel Synchronous Counter

The least significant stage is simply a toggle which changes with every clock pulse. Since $J_1=K_1=y_0$, the next most significant stage, y_1, will change state whenever $y_0=1$ and will remain the same otherwise. More generally, since $J_i=K_i = y_{i-1} \bullet y_{i-2} \bullet \ldots \bullet y_1 \bullet y_0$, the i^{th} stage will change state only when all positions of less significance are 1 and will remain the same otherwise.

In each counter, the control inputs to the i^{th} stage, J_i and K_i, are the same logical function. The only difference between the counters is the logical circuit for generating the function. At the i^{th} stage, the serial counter uses the already constructed function $J_{i-1} = y_{i-2} \bullet \ldots y_1 \bullet y_0$ to get $J_i = J_{i-1} \bullet y_{i-1}$ while the parallel counter constructs its control inputs

directly from all less significant positions $J_i = y_{i-1} \cdot y_{i-2} \cdot \ldots \cdot y_1 \cdot y_0$. Following a state transition the parallel counter control input J_i will stabilize sooner than the serial counter control input J_i due to the finite delay through the long serial string of AND gates used to generate J_i in the serial counter.

Parallel-load Counters. In many applications it is desirable to be able to load an initial value into a counter and have it count beginning at that value. This is accomplished by the same techniques used in parallel-load shift registers.

Gray Code Counter. A Gray code counter is one where only a single bit changes with each successive count. In the binary counter, the circuit outputs were simply the four individual flip-flop states. It is possible to include combinational logic so that the circuit outputs are some combinational function of the flip-flop states. For example, combinational logic can be added to the circuit of Figure 3 so that the system outputs go through the Gray code sequence while the flip-flop states go through the binary sequence. See below. However, for reasons you will discuss in Experiment 3, we will not design a Gray code counter using this method. Instead, we will design a synchronous sequential machine to count through the Gray code.

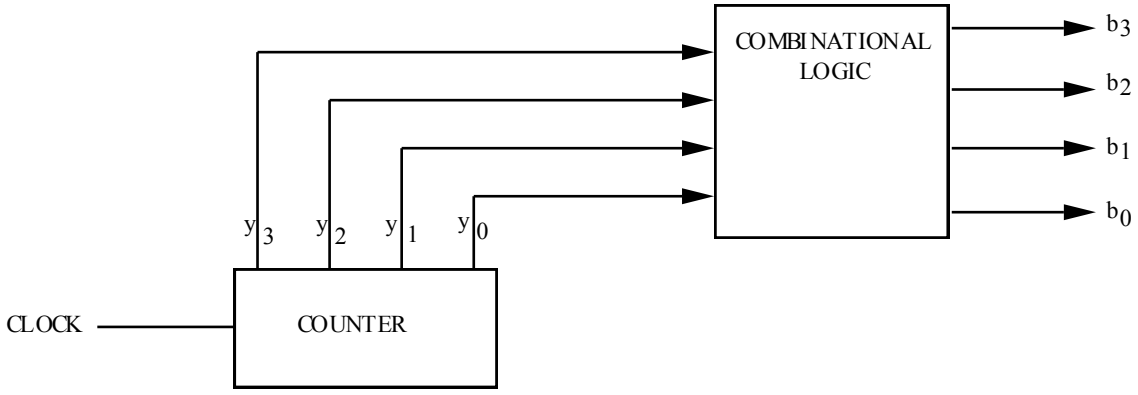


Figure 5a. Gray Code Counter from a Binary Counter

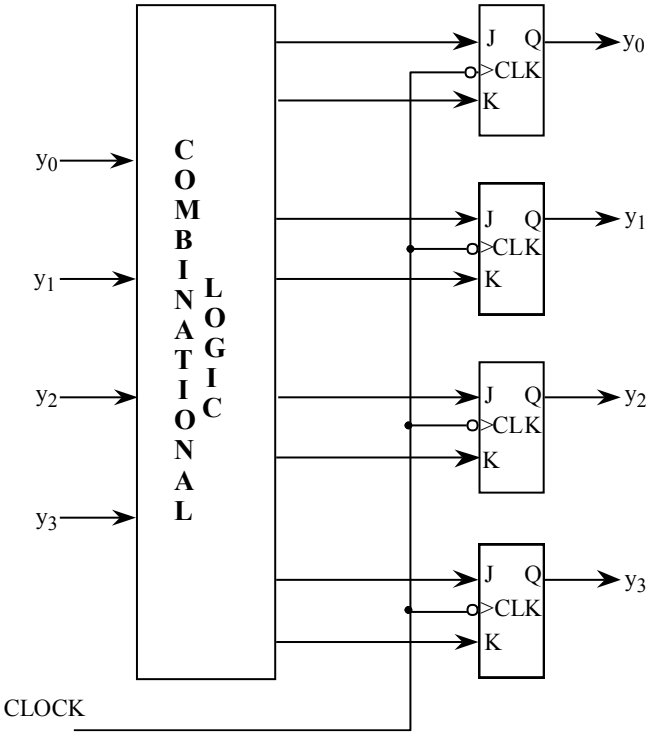
(current) state $y_3y_2y_1y_0$	(current) output $b_3b_2b_1b_0$
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 0	0 0 1 1
0 0 1 1	0 0 1 0
0 1 0 0	0 1 1 0
0 1 0 1	0 1 1 1
0 1 1 0	0 1 0 1
0 1 1 1	0 1 0 0
1 0 0 0	1 1 0 0
1 0 0 1	1 1 0 1
1 0 1 0	1 1 1 1
1 0 1 1	1 1 1 0
1 1 0 0	1 0 1 0
1 1 0 1	1 0 1 1
1 1 1 0	1 0 0 1
1 1 1 1	1 0 0 0

Figure 5b. Combinational Logic must Produce Outputs $b_3b_2b_1b_0$ from Flip-Flop States $y_3y_2y_1y_0$

Design for a 4-bit synchronous Gray Code counter (the flip flop states of this counter must sequence through the Gray Code).

Note that EXCLUSIVE-OR gates are extremely useful for reducing the amount of combinational logic for this circuit.

Note that if $f(x, y, z, \ldots) = x \oplus g(y, z, \ldots)$, then $f(x, y, z, \ldots) = \overline{x}\ g(y, z, \ldots) + x\ \overline{g}(y, z, \ldots)$ and the Karnaugh map for f can be divided into an x half and an $\overline{x}$ half whose cell entries are complements of those in the x half. One should thus look for a checkerboard-like pattern of "1"s and "0"s in a Karnaugh map to see if EXCLUSIVE-ORs are useful.



Transition Table

Present state				NEXT STATE			
y_3	y_2	y_1	y_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	1
0	0	1	1	0	0	1	0
0	0	1	0	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	0	1	0	1
0	1	0	1	0	1	0	0
0	1	0	0	1	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	0
1	1	1	0	1	0	1	0
1	0	1	0	1	0	1	1
1	0	1	1	1	0	0	1
1	0	0	1	1	0	0	0
1	0	0	0	0	0	0	0

C.16

Transition Maps

$y_3 y_2$ \ $y_1 y_0$	00	01	11	10
00	1	1	0	0
01	0	0	1	1
11	1	1	0	0
10	0	0	1	1

Y_0

$y_3 y_2$ \ $y_1 y_0$	00	01	11	10
00	0	1	1	1
01	0	0	0	1
11	0	1	1	1
10	0	0	0	1

Y_1

$y_3 y_2$ \ $y_1 y_0$	00	01	11	10
00	0	0	0	1
01	1	1	1	1
11	1	1	1	0
10	0	0	0	0

Y_2

$y_3 y_2$ \ $y_1 y_0$	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	1	1
10	0	1	1	1

Y_3

transition	J	K
0→0	0	d
0→1	1	d
1→0	d	1
1→1	d	0

Karnaugh Maps

$y_3 y_2$ \ $y_1 y_0$	00	01	11	10
00	1	d	d	0
01	0	d	d	1
11	1	d	d	0
10	0	d	d	1

J_0

$y_3 y_2$ \ $y_1 y_0$	00	01	11	10
00	d	0	1	d
01	d	1	0	d
11	d	0	1	d
10	d	1	0	d

K_0

$y_3 y_2$ \ $y_1 y_0$	00	01	11	10
00	0	1	d	d
01	0	0	d	d
11	0	1	d	d
10	0	0	d	d

J_1

$y_3 y_2$ \ $y_1 y_0$	00	01	11	10
00	d	d	0	0
01	d	d	1	0
11	d	d	0	0
10	d	d	1	0

K_1

$y_3 y_2$ \ $y_1 y_0$	00	01	11	10
00	0	0	0	1
01	d	d	d	d
11	d	d	d	d
10	0	0	0	0

J_2

$y_3 y_2$ \ $y_1 y_0$	00	01	11	10
00	d	d	d	d
01	0	0	0	0
11	0	0	0	1
10	d	d	d	d

K_2

$y_3 y_2$ \ $y_1 y_0$	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	d	d	d	d
10	d	d	d	d

J_3

$y_3 y_2$ \ $y_1 y_0$	00	01	11	10
00	d	d	d	d
01	d	d	d	d
11	0	0	0	0
10	1	0	0	0

K_3

AN ALTERNATIVE METHOD USING TRANSITION NOTATION

Current State				Next State				Transition Table				y3		y2		y1		y0	
y_3	y_2	y_1	y_0	y'_3	y'_2	y'_1	y'_0	y_3^T	y_2^T	y_1^T	y_0^T	J_3	K_3	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	0	0	1	0	0	0	↑	0	d	0	d	0	d	1	d
0	0	0	1	0	0	1	1	0	0	↑	1	0	d	0	d	1	d	d	0
0	0	1	1	0	0	1	0	0	0	1	↓	0	d	0	d	d	0	d	1
0	0	1	0	0	1	1	0	0		1	0	0	d	1	d	d	0	0	d
0	1	1	0	0	1	1	1	0	1	1	↑	0	d	d	0	d	0	1	d
0	1	1	1	0	1	0	1	0	1	↓	1	0	d	d	0	d	1	d	0
0	1	0	1	0	1	0	0	0	1	0	↓	0	d	d	0	0	d	d	1
0	1	0	0	1	1	0	0	↑	1	0	0	1	d	d	0	0	d	0	d
1	1	0	0	1	1	0	1	1	1	0	↑	d	0	d	0	0	d	1	d
1	1	0	1	1	1	1	1	1	1	↑	1	d	0	d	0	1	d	d	0
1	1	1	1	1	1	1	0	1	1	1	↓	d	0	d	0	d	0	d	1
1	1	1	0	1	0	1	0	1	↓	1	0	d	0	d	1	d	0	0	d
1	0	1	0	1	0	1	1	1	0	1	↑	d	0	0	d	d	0	1	d
1	0	1	1	1	0	0	1	1	0	↓	1	d	0	0	d	d	1	d	0
1	0	0	1	1	0	0	0	1	0	0	↓	d	0	0	d	0	d	d	1
1	0	0	0	0	0	0	0	↓	0	0	0	d	1	0	d	0	d	0	d

Transition	J	K
1	d	0
↑	1	d
↓	d	1
0	0	d

C.18

y_3

J_3	y_1y_0 00	01	11	10
y_3y_2 00	0	0	0	0
01	1	0	0	0
11	d	d	d	d
10	d	d	d	d

K_3	y_1y_0 00	01	11	10
y_3y_2 00	d	d	d	d
01	d	d	d	d
11	0	0	0	0
10	1	0	0	0

y_2

J_2	y_1y_0 00	01	11	10
y_3y_2 00	0	0	0	1
01	d	d	d	d
11	d	d	d	d
10	0	0	0	0

K_2	y_1y_0 00	01	11	10
y_3y_2 00	d	d	d	d
01	0	0	0	0
11	0	0	0	1
10	d	d	d	d

y_1

J_1	y_1y_0 00	01	11	10
y_3y_2 00	0	1	d	d
01	0	0	d	d
11	0	1	d	d
10	0	0	d	d

K_1	y_1y_0 00	01	11	10
y_3y_2 00	d	d	0	0
01	d	d	1	0
11	d	d	0	0
10	d	d	1	0

y_0

J_0	y_1y_0 00	01	11	10
y_3y_2 00	1	d	d	0
01	0	d	d	1
11	1	d	d	0
10	0	d	d	1

K_0	y_1y_0 00	01	11	10
y_3y_2 00	d	0	1	d
01	d	1	0	d
11	d	0	1	d
10	d	1	0	d

Expressions

$J_0 = \bar{y}_1 \bar{y}_2 \bar{y}_3 + y_1 y_2 \bar{y}_3 + \bar{y}_1 y_2 y_3 + y_1 \bar{y}_2 y_3$

$= (\bar{y}_1 \bar{y}_2 + y_1 y_2) \bar{y}_3 + (\bar{y}_1 y_2 + y_1 \bar{y}_2) y_3$

$= \overline{(y_2 + y_1) \oplus y_3}$

$J_0 = \overline{y_1 \oplus y_2 \oplus y_3}$

$K_0 = y_1 \bar{y}_2 \bar{y}_3 + \bar{y}_1 y_2 \bar{y}_3 + y_1 y_2 y_3 + \bar{y}_1 \bar{y}_2 y_3$

$= (y_1 y_2 + \bar{y}_1 \bar{y}_2) y_3 + (y_1 \bar{y}_2 + \bar{y}_1 y_1) \bar{y}_3$

$= (y_1 + y_2) \oplus y_3$

$K_0 = y_1 \oplus y_2 \oplus y_3$

$J_1 = y_0 \bar{y}_2 \bar{y}_3 + y_0 y_2 y_3$

$= y_0 (\bar{y}_2 \bar{y}_3 + y_2 y_3)$

$J_1 = y_0 \overline{(y_2 \oplus y_3)}$

$K_1 = y_0 y_2 \bar{y}_3 + y_0 \bar{y}_2 y_3$

$= y_0 (y_2 \bar{y}_3 + \bar{y}_2 y_3)$

$K_1 = y_0 (y_2 \oplus y_3)$

$J_2 = \bar{y}_0 y_1 \bar{y}_3$

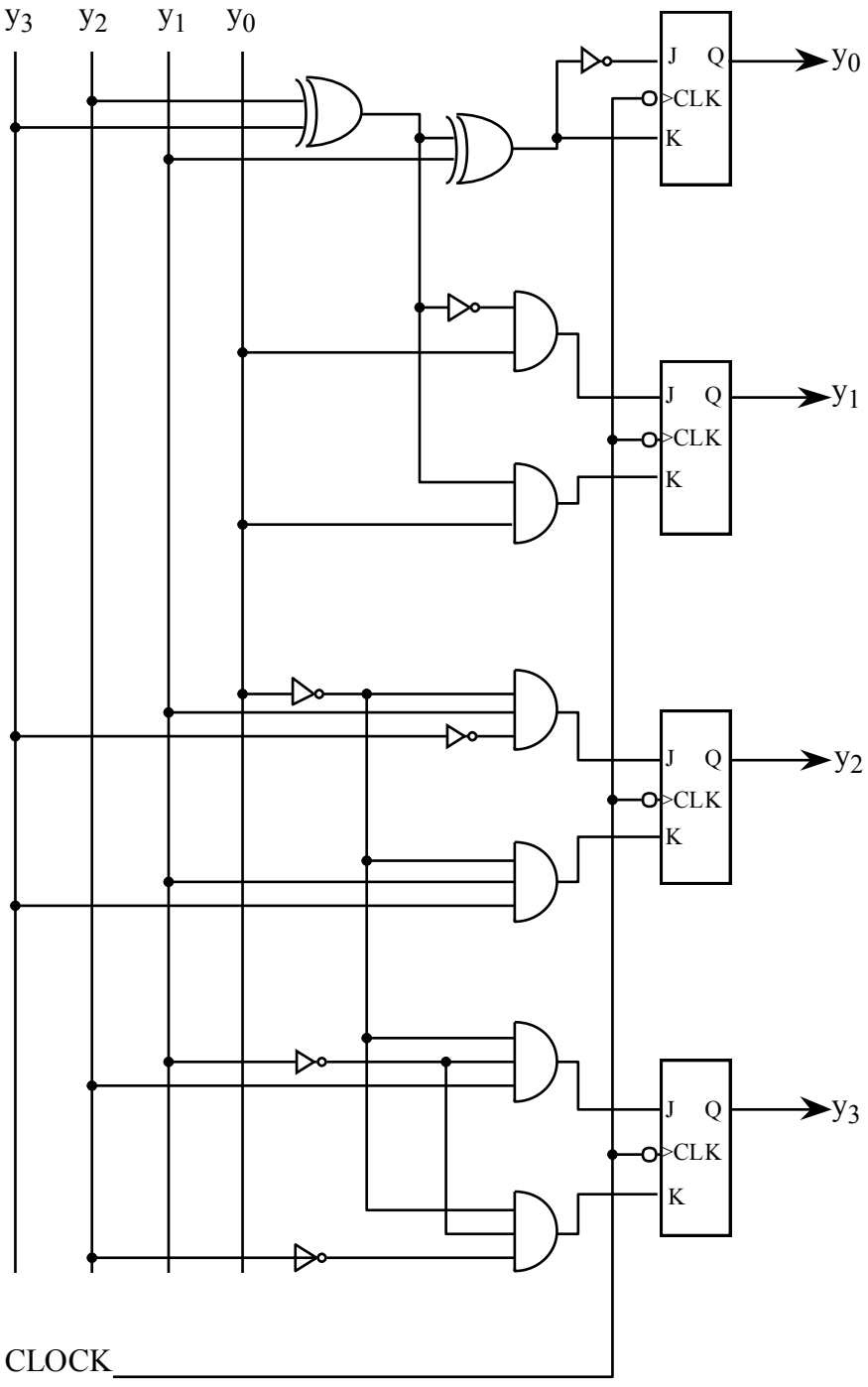
$K2 = \bar{y}_0 y_1 y_3$

$J_3 = \bar{y}_0 \bar{y}_1 y_2$

$K_3 = \bar{y}_0 \bar{y}_1 \bar{y}_2$

C.20

Logic Diagram



Counter Moduli. The modulus of a counter is the length or period of its output sequence. For example, the counters producing the following output sequences have moduli of 3.

output sequence of counter 1	output sequence of counter 2	output sequence of counter 3
0000	0000	00
0001	0101	11
0010	1011	10
0000	0000	00
•	•	•
•	•	•
•	•	•

The modulus of an n-stage counter similar to those of Figures 2, 3, or 4 is 2^n. Any output sequence of period 2^n can be produced by the proper combinational logic circuit with the binary counter flip-slop states as inputs.

To construct a counter whose modulus is not a power of 2, two different design methods are commonly used. The first method is to modify a counter of the form of Figures 2, 3, and 4 to have a modulus of 15 (rather than 2^4=16) by adding the circuit of Figure 6.

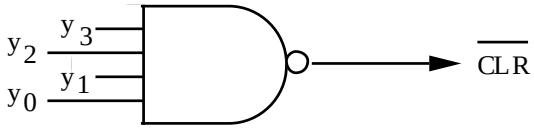


Figure 6. Counter Modification Circuit with a state detector NAND gate

If the counter's initial state is 0000, the first fourteen clock pulses will produce normal state transitions finishing with the state 1110. On the falling edge of the 15th clock pulse, the counter will momentarily go to the state 1111. However, this will cause the output of the NAND gate of Figure 6 to go low, which in turn will asynchronously clear all of the flip-flops to the 0 state before the 16th clock pulse. Thus 15 clock pulses cause the counter to return to the initial state, or in other words, the counter has a period, or modulus, of 15. This method may be somewhat risky due to races and transients in the resulting circuit.

The second method for constructing a counter whose modulus is not a power of two involves a direct design from flip-flops and gates.

D. Introduction to Synchronous Sequential Network Design

Synchronous sequential network design is best illustrated by example. Consider the design of a two position mod 3 counter that counts through the state sequence $00 \rightarrow 11 \rightarrow 10 \rightarrow 00 \rightarrow \ldots$

The first step in designing such a circuit is to graphically represent the desired operation with a state transition diagram in which there is a node for each state and a directed arc for each possible transition.

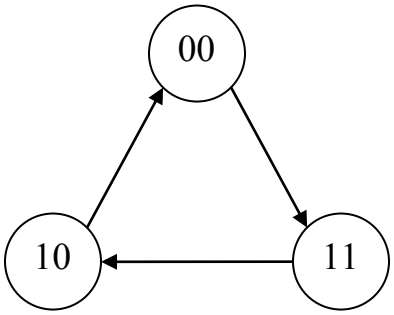


Figure 7. State Transition Diagram

Next the system operation described by the state transition diagram is summarized in a state transition table. The state transition table gives the next system state as a function of the current system state (and the system inputs if there are any).

CURRENT STATE y_1y_0	NEXT STATE y_1y_0
00	11
01	dd
10	00
11	10

Figure 8. State Transition Table

Note that the table includes 01 as a possible system state and indicates that in this case we don't care what the next state is.

Finally we need to find flip-flop control input functions which produce the desired transitions. The state transition table shows that when the current state is 00, flip-flop y_1 goes from an initial state of 0 to a next state of 1. Control inputs $J_1 = 1$ and $K_1 = 0$ or 1 will produce this transition. Thus when the current state is 00, we don't care what value K_1 has but J_1 must have value 1. This information can be entered in a truth table giving J_1 and K_1 as functions of the current state. To save steps it is usually entered directly into a Karnaugh map, as shown below.

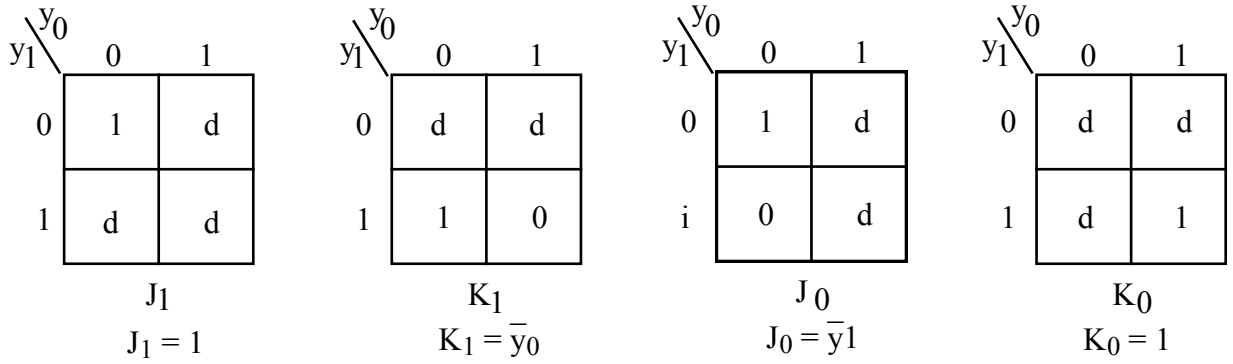


Figure 9. JK Flip-flop Excitation Functions

Other entries are made in a similar manner. If the initial flip-flop state is 0 and a next state of 0 is needed, we can use either J=0, K=1 or J=0, K=0; so we must have J=0 and we don't care what K is. A similar analysis yields J=d, K=1 for the transition $1 \rightarrow 0$ and J=d, K=0 for the transition $1 \rightarrow 1$. For reference in future designs, this is summarized in the table below.

transition	J K
$0 \rightarrow 0$	0 d
$0 \rightarrow 1$	1 d
$1 \rightarrow 0$	d 1
$1 \rightarrow 1$	d 0

Figure 10. JK Flip-flop Inputs Needed to Produce State Transitions

Returning to the mod 3 counter design, when the system state is 01, we don't care what the next state is, and hence we don't care about the values of either the J or the K control inputs. The Karnaugh maps of Figure 9 yield the following logical expressions for the flip-flop control inputs.

$$J_1 = 1$$

$$K_1 = \overline{y}_0$$

$$J_0 = \overline{y}_1$$

$$K_0 = 1$$

A circuit counting through the desired state sequence is shown below.

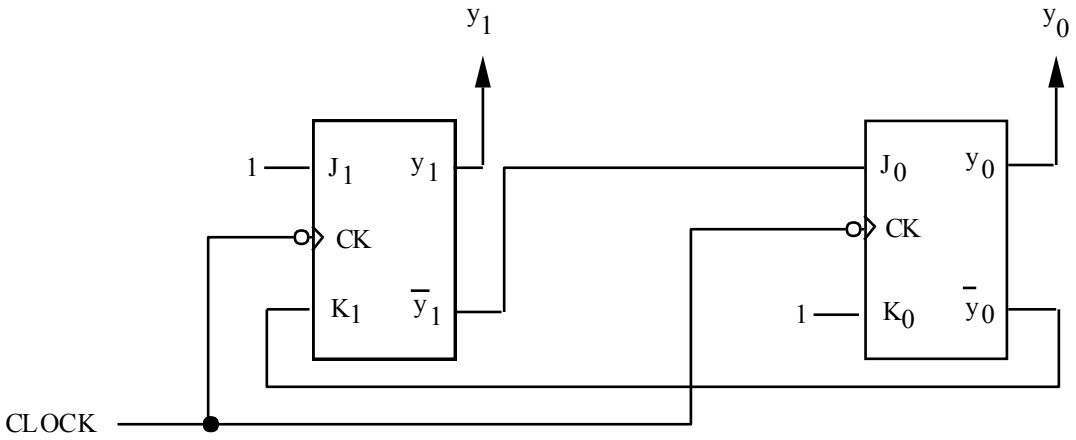


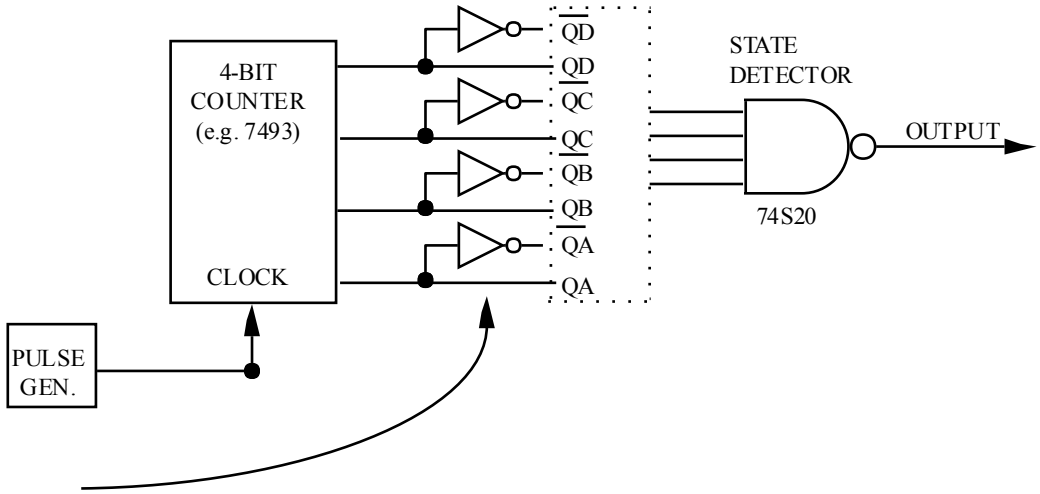
Figure 11. Circuit for Mod 3 Counter

Finally we should check to see what the circuit does if it powers up in the 01 state. For $y_1y_0 = 01$, the equations show J_1 will be 1, K_1 will be 0, J_0 will be 1 and K_0 will be 1. With these values the next state will be 10. If this state had not been in the "normal" state sequence (or led to the "normal" state sequence much as in the "self-starting property for shift registers") we could redesign the counter to force particular behavior in some of the previous "don't care" situations. Alternatively we could use the asynchronous clear inputs to initialize all flip-flops to the 0 state. If the all 0 state was not in the "normal" state sequence, we would have needed to redesign the circuit. For this reason the all 0 state is usually included in the normal state sequence of a synchronous sequential network.

You should be careful, in your design for this and all later experiments, to use the established techniques for synchronous and asynchronous sequential network design. In particular, design errors should never be corrected by inserting RC networks or monostable multivibrators (one-shots) within a sequential circuit simply to provide a delay of one internal signal with respect to another. The solution for such timing problems is to redesign your logic correctly. No credit will be added for making a design

work by adding such delay elements. The use of such delay elements is only proper when designing special signals with specified frequencies or delays relative to one another.

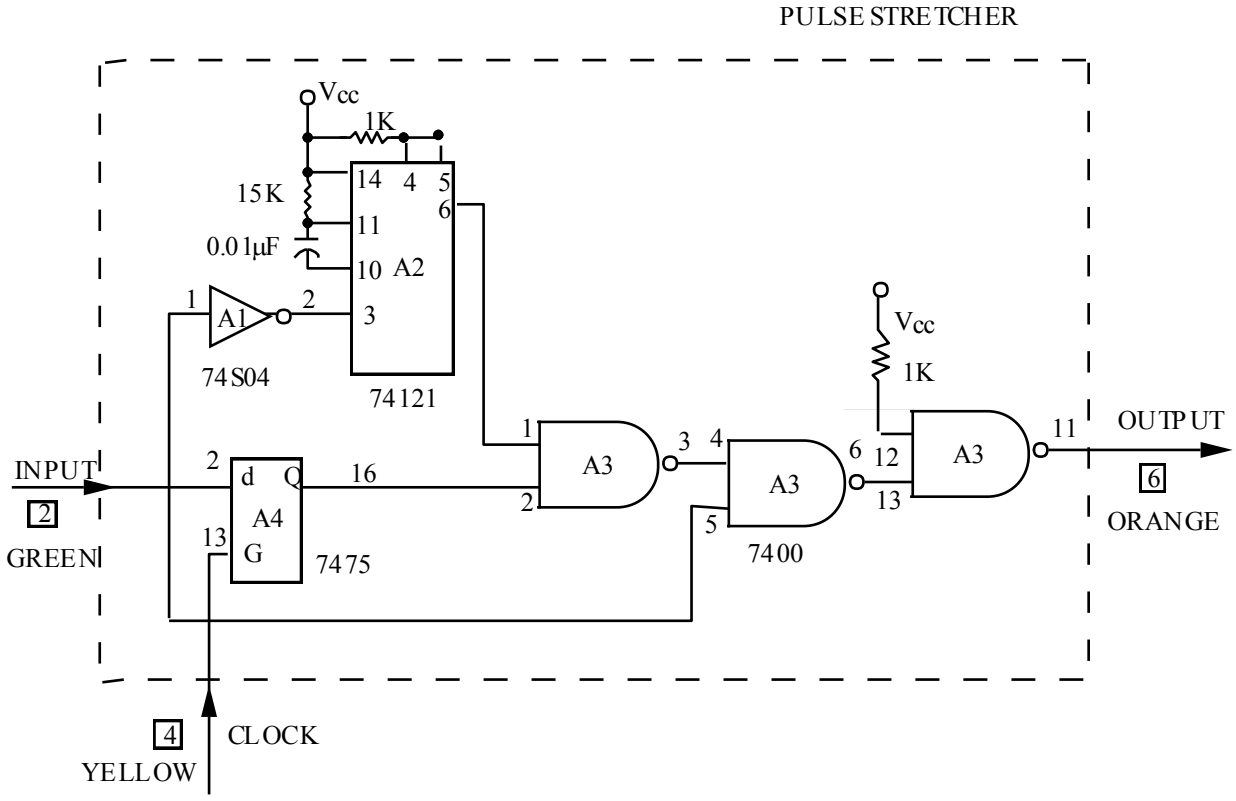
E. State Detector Circuit



Note: Any inverters used here should be from a 74S04 chip or 74F04, not a 7404. [If Q and $\overline{Q}$ of each FF are directly available, those inverters will not be required.] Is there a better way to do this? Discuss.

F. Pulse Stretcher

Although you should not need it for the experiment, the circuit documentation for a pulse stretcher, as wired on your lab station, has been provided below.



The function of the pulse stretcher circuit is to extend short negative going glitches into wider pulses. Thus a typical input/output waveform may look a follows: