Problem 9-2

Let the alphabet of seven symbols be {a, b, c, d, e, f, g}. Let the 3 lines used to transmit the symbols be A_2, A_1 and A_0 respectively. Since there are only three lines, the following limitations become clear – it is not possible to implement the encoding in such a way that exactly one line or two lines transitions for all possible combinations of symbol change. Thus when going from symbol a to f we might need to transition all three lines A_2, A_1 and A_0. The following table can be constructed for the symbols and the corresponding transitions on the line.

Current Symbol	A_2 A_1 A_0
A	0 0 1
B	0 1 0
C	0 1 1
D	1 0 0
E	1 0 1
F	1 1 0
G	1 1 1

Thus a new symbol with a different value than the current symbol can be recognized as a transition on one of the three lines and the value of the new symbol decoded from the state of the lines after 1 bit cell time. Thus if the current symbol is b and the next symbol g the lines ($A_2A_1A_0$) go from 010→111. Only the first transition (A_2 or A_0) indicates that a new symbol has started. (The transitions may be skewed due to timing uncertainty). All other transitions are ignored for 1 bit time. The value of the line (111) at the end of the bit time is used as the value of the symbol.

The other case to consider is when the next symbol has the same value as the current symbol i.e we are transmitting f f in a row. In this case all the 3 lines go to the 000 state after the first f. Since 000 has not been chosen for any of the 7 seven symbols this guarantees that there will be a transition indicating that a new symbol (the next f) has started. Since the new symbol has the same value as the previous symbol we ignore the 000 value on the line and use the previous line values (110) as the symbol value. If a third f is transmitted we transition from 000→110 (the value of f) and proceed decoding normally at the other end. So at the receiver, whenever a 000 is received the value of that symbol is interpreted as the value of the previous symbol.

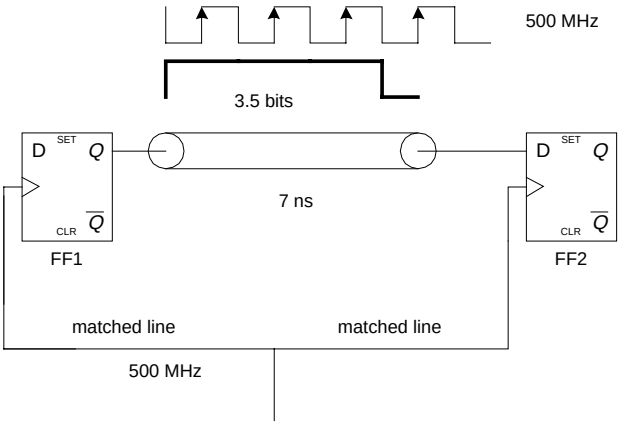
An example:

Transmit: e d d f f a b
Start the line with 000. Line values ($A_2A_1A_0$) in time: 000→101→100→000→110→000→001→010
The decoder at the receiver end detects seven symbols corresponding to seven transitions separated by 7 bit times. The two transitions to 000 indicate that the 100 and 110 symbols have been repeated.

Caveat
The caveat with this implementation is that the skew between lines A_0, A_1 and A_2 cannot exceed the bit time. Thus the transition from d to g (100→111) involves transitions on both A_1 and A_0 which have to take place 'simultaneously' i.e. within the spacing of a bit cell. If A_1 wiggles first then it will indicate the start of a new symbol. The wiggle on A_0 has to happen within a bit cell time, otherwise the current symbol will be incorrectly interpreted as 110. However as mentioned earlier the wiggle on A_0 is ignored as far as timing is concerned since the wiggle on A_1 already indicated the start of a new symbol.

This is just one implementation of the signal encoding. Other algorithms are also possible.

Problem 9-3



Let the data on the link travel from FF1 to FF2. The delay of the link, T_{wire} = 7 ns.
Let t_{add} be the total additional delay in the link due to rise-time, aperture time and timing uncertainty (skew + jitter).

$$T_{add} = 0.5(T_r + T_a) + T_u = 0.5 * (1ns + 300\,ps) + 300\,ps = 950\,ps$$

Looking at the setup above we can write a set of maximum-minimum delay constraints.
In the case when the additional delay is worst case maximum, in order to operate with correct timing we need

$$T_{wire} + T_{add} \le 4 * T_{bit}$$

or $T_{bit} \ge 1988\,ps$ i.e $ClockFrequency \le 503MHz$

If the above constraint is not met, then the clock triggers the (old) same data twice since the data is late at FF2.

In the case when the additional delay is worst case minimum, to operate with correct timing we need

$$T_{wire} - T_{add} \ge 3 * T_{bit}$$

or $T_{bit} \le 2017\,ps$ ie. $ClockFrequency \ge 495MHz$

If the above constraint is not met then the data at the D input of FF2 is corrupted before the clock has a chance to trigger it to the output.

Thus the link works for $495MHz \le ClockFrequency \le 503MHz$. At clock frequencies outside of this range the above constraints will not be satisfied and the system will fail

<u>Problem 9-6</u>: **Sequential Phase-Only Comparator**

The following table shows the transition table of the comparator logic:

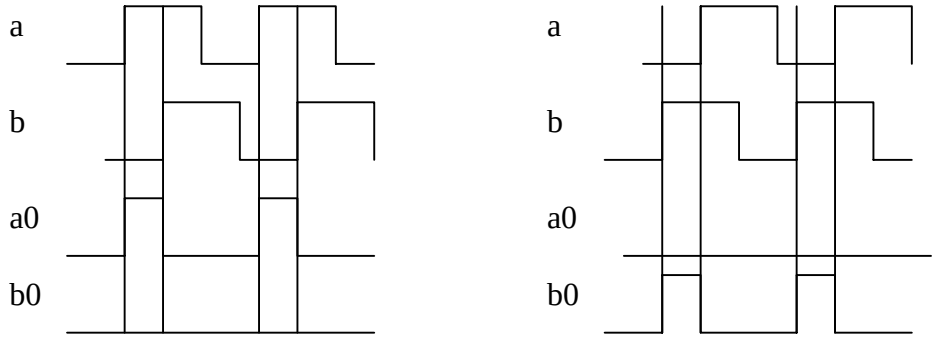
a0 = a * x_b * b0_b
b0 = b * x_b * a0_b
x = (a * b) + ((a + b) * x)

			ab			
a0	b0	x	00	01	10	11
0	0	0	**000**	010	100	111
0	0	1	000	**001**	**001**	**001**
0	1	0	000	**010**	000	011
0	1	1	000	001	001	001
1	0	0	000	000	**100**	101
1	0	1	000	001	001	001
1	1	0	000	000	000	001
1	1	1	000	001	001	001

The each row indicated by (a0, b0, x) indicates the current state, each column of (a,b) indicate the current input, and each entry of the table indicates the next state (a0, b0, x). Since this is an asynchronous system, there're no latches between the current state (a0,b0, x), and the next state (a0, b0, x). Therefore, one change in input may cause multiple state transitions. For example, if the current state is (1,0,0) and the input changes from (1,0) to (1,1), state first changes to (1,0,1), but the entry at row (1,0,1) and column (1,1) suggests that the state will now transition to 001, at which state it will remain until input changes again. Therefore, now there are stable states and unstable ones, and bold entries indicate the stable ones. Basically, if the next state is equal to the current state, the state will not transition any more for the given input, and it is stable. If inputs are constrained so that only one bit changes at a time, the construction and analysis of such an asynchronous system is pretty straightforward. But in this case, two inputs bits can change simultaneously. (In fact, the closed loop system in which this comparator will be used forces them to change simultaneously.) Therefore, the actual behavior is a bit more tricky since it depends on delay of each input transition to output transition etc. As we'll see later, even if the two input bits don't change simultaneously, if the transitions happen in a short amount of time compared to the delay of the logic, it looks as if the bits changed simultaneously. This can be a good thing or bad thing, as will be shown later.

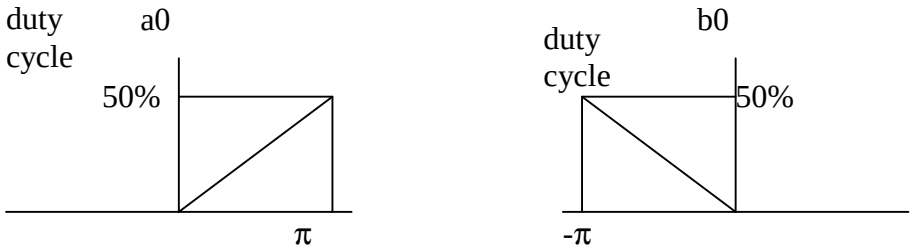
Just by following the transition table above, we can see the general behavior of this comparator. Basically, when input bits are (0,0), the comparator goes into reset state of (0,0,0). It stays there until one of the input bits goes high, at which point one of the a0 and b0 lines is raised. This continues until the other edge goes up. Then both a0 and b0 lines go low, and x goes high, indicating that the comparator has registered both edges. This state is (0,0,1), which may be labeled 'wait' state, since it sits there until both the input bits become zero, at which time the cycle repeats. Careful observation also reveals that this comparator depends on the facts that there's at least some overlap between high a and high b regions and also between low a and low b regions. If the a and b waveforms do not have overlap 1-regions, (which would be the case if the duty cycle is less than 50% and the two input signals have large phase offset,) the table predicts that a0 and b0 will have equal duty cycles. Since the actual phase offset information is gotten by subtracting the b0 duty cycle from a0 duty cycle, this non-overlapping a and b waveforms result in phase reading of zero, which is an error. Also, when a and b have no overlapping low regions, (which would be the case if duty cycle > 50% and phase offset so that (a,b) is not equal to (0,0) at any time,) we can see that the system will get stuck in the 'wait' state, since the comparator is reset by (a,b) being (0,0). Other than these two requirements, the frequencies of the two inputs should be matched as well. The feedback loop will probably correct any small frequency error, but the phase reading will be meaningless unless the two inputs are at similar frequencies.

Two example waveforms predicted by the table are shown below.



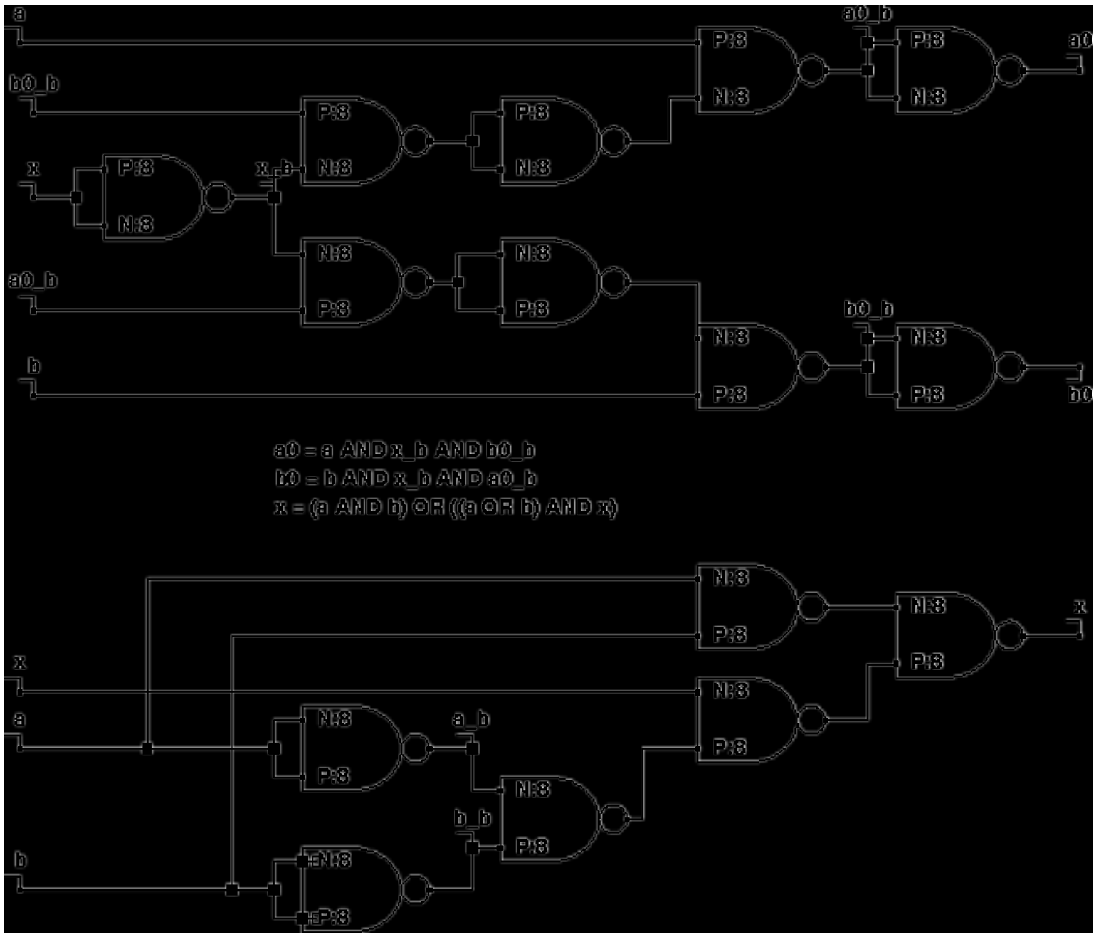
As shown, a0 will indicate that a0 is leading (starting from the reset state), and b0 will indicate that b0 is leading. Also, the duty cycle of these two signals show the amount of phase offset. When they're exactly matched, meaning a and b go high and low at the same time, a0 and b0 will both stay at 0, if we assume the logic delay is none and transitions are instantaneous. On the table, this corresponds to state transitions of (0,0,0) -> (1,1,1) -> (0,0,1) -> (0,0,0), which confirms that a0 and b0 are zero at all stages. But again, the logic delay will show slightly different behavior, as will be shown on the HSPICE simulation.

From the example waveforms shown, we can see the dynamic range of this comparator. Assuming one period of the periodic waveform corresponds to 2*π, and positive phase means a is leading, the response looks as follows:



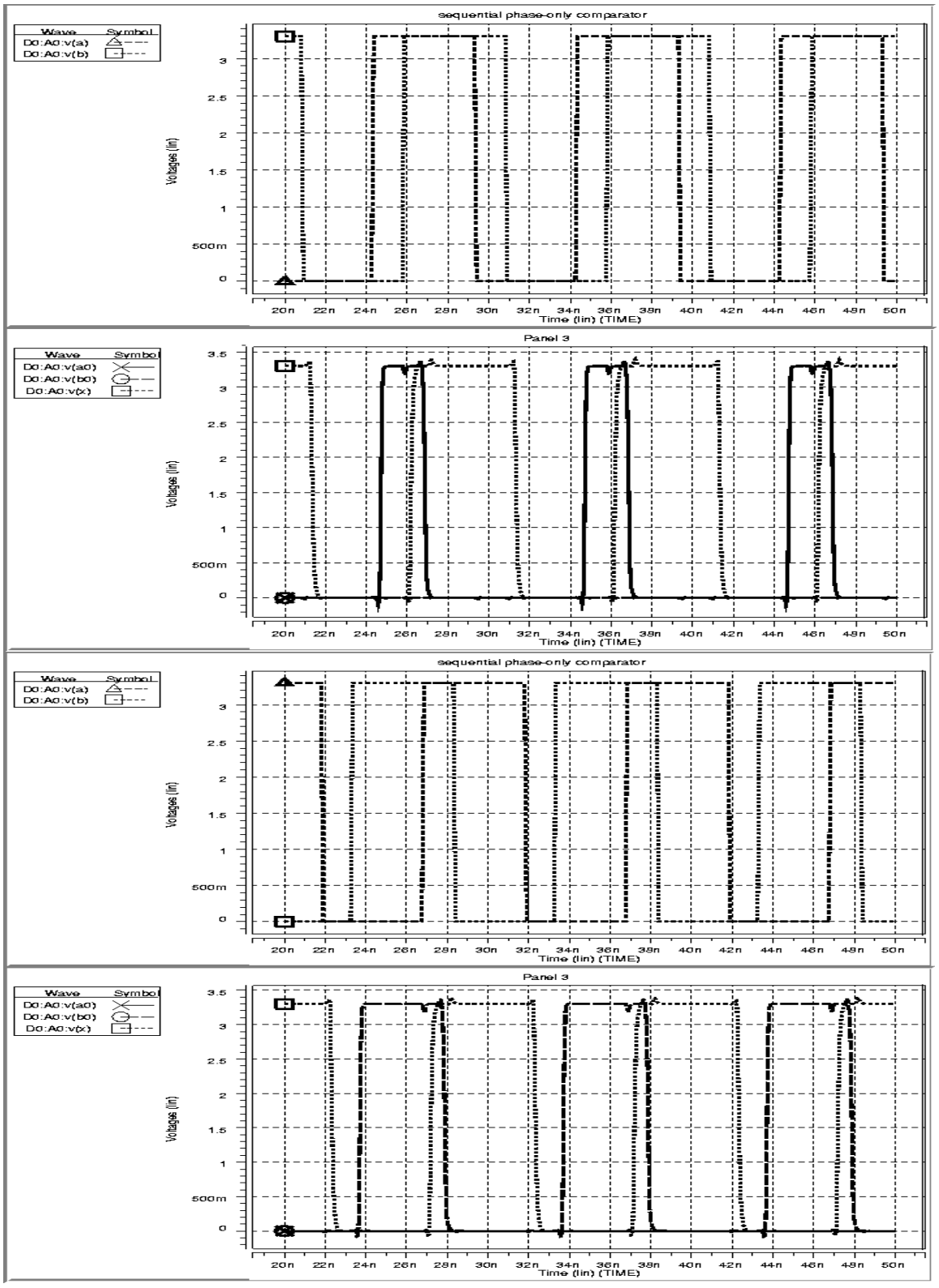
Therefore, a0 – b0 gives a linear phase response of this comparator. By the way, when the phase offset is exactly ±π, meaning they're complement of each other, funny thing happens. Over a broader region around that area, it can be deduced that the phase response will wrap around, so that a0's duty cycle of 50% will be small offset away from b0's duty cycle of 50%. However, a metastable point exists when they're exactly π apart. In the table, it means that the comparator will stay in 'wait' state forever, since the inputs are never (0,0) and they switch from (1,0) and (0,1) only. 'Wait' state is a stable state for both of these inputs, and the comparator never gets out of it. This has nothing to do with the logic delay, since it strictly depends on the input transition. This is a bad thing unless phase lock of ±π radians is also okay, but hopefully, this exact phase offset will not occur when the closed loop control is doing its work well. Notice that this state is called meta-stable, since any deviation from it will force the system toward phase offset of zero, which is the only truly stable point of the system.

Now, let's consider how to construct this phase comparator using only two input NAND gates. One possible solution is shown below, which is gotten by pushing bubbles around. Notice that inverters were obtained by tying both inputs of NAND gates together. I didn't really worry about gate sizing at all, and simply used the default gates sizes.

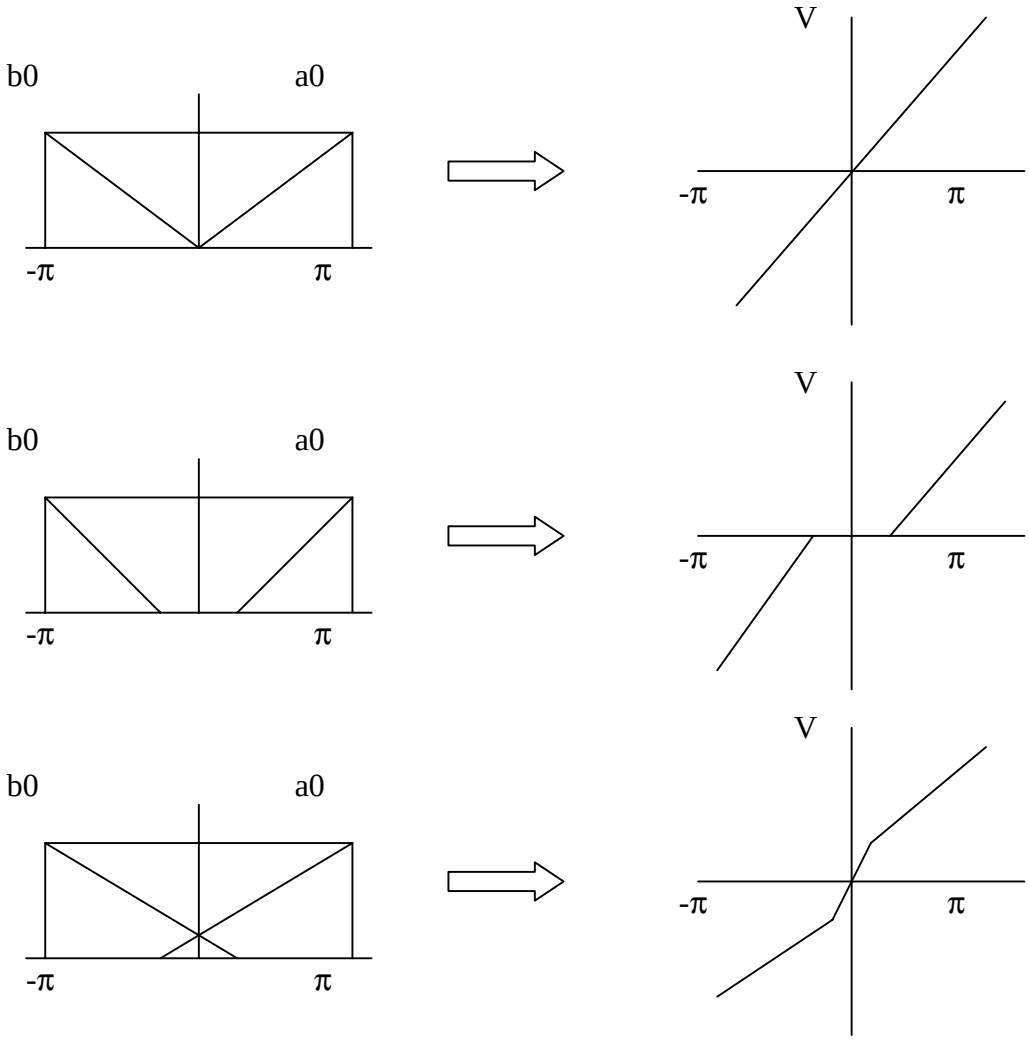


The following page shows the HSPICE simulation results. The upper plot shows the waveforms when a arrives earlier than b by about 0.15*π, and the lower plot shows the waveforms when b arrives earlier then a by about 0.35*π. For both of these plots, the top graph is the input waveforms, and the bottom one is the output (state) waveforms. As shown, the period of the pulse waveforms is 10nsec, corresponding to 100Mhz frequency.

Even though the edges of the output waveforms are not very sharp due to non-optimal gate sizings, and there're some noticeable delays, the waveforms closely resembles the expected waveforms. That is, when a is early, a0 goes high for a while, and when b is early, b0 goes high for a while. Also, the duty cycle is approximately proportional to the offset amount.

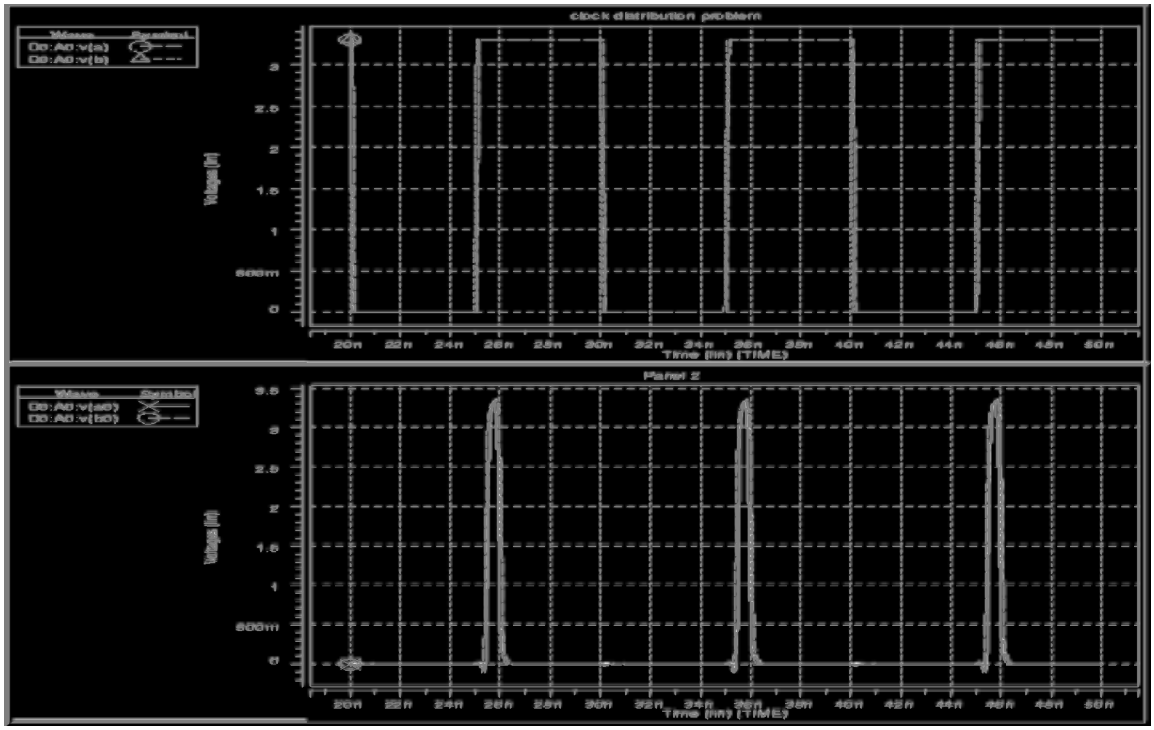

sequential phase-only comparator
Wave
Symbol
D0:A0:v(a)
D0:A0:v(b)
Voltages (lin)
3
2.5
2
1.5
1
500m
0
20n
22n
24n
26n
28n
30n
32n
34n
36n
38n
40n
42n
44n
46n
48n
50n
Time (lin) (TIME)
Panel 3
Wave
Symbol
D0:A0:v(a0)
D0:A0:v(b0)
D0:A0:v(x)
Voltages (lin)
3.5
3
2.5
2
1.5
1
500m
0
20n
22n
24n
26n
28n
30n
32n
34n
36n
38n
40n
42n
44n
46n
48n
50n
Time (lin) (TIME)
sequential phase-only comparator
Wave
Symbol
D0:A0:v(a)
D0:A0:v(b)
Voltages (lin)
3
2.5
2
1.5
1
500m
0
20n
22n
24n
26n
28n
30n
32n
34n
36n
38n
40n
42n
44n
46n
48n
50n
Time (lin) (TIME)
Panel 3
Wave
Symbol
D0:A0:v(a0)
D0:A0:v(b0)
D0:A0:v(x)
Voltages (lin)
3.5
3
2.5
2
1.5
1
500m
0
20n
22n
24n
26n
28n
30n
32n
34n
36n
38n
40n
42n
44n
46n
48n
50n
Time (lin) (TIME)

Even for relatively ideal response as shown on the previous page, the effect of logic delay is evident. For example, when a is leading, a0 doesn't go high at the same moment a goes high, and a0 doesn't go low at the same moment b goes high. Also, the overlap between x and a0 is quite visible, which is not predicted by naïve interpretation of the state transition table. Now, my concern is whether this comparator has a dead zone or not. Dead zone is where there is non-zero offset which the comparator can't see. We've already seen one kind of dead zone around phase offset of $\pm\pi$. But since this is a meta-stable state, we can count on the fact that small disturbance will eventually bring the system out of this state. Besides, assuming the closed loop does its job well, we will not spend much time around this region at all. The more crucial region is around phase of zero. Since we want zero offset, the comparator response around this region must be very sensitive. Now, three possibilities of a0 and b0 response and its effect on the comparator response is shown below. Note that comparator response is simply a0 response minus b0 response.



The first case shows the ideal case, when we neglect any delay. This gives a perfectly linear comparator response (which is not necessarily good, as explained in the lecture.) However, due to process variations, it's not a good idea to rely on this ideal behavior. For example, something like the second case can happen, where there's some area when neither a0 and b0 are high for small offset. For this case, the comparator will say that there's no phase error even when there is, which prevents accurate phase locking from happening. So, second case is definitely a bad thing. What we want is something like the third case, which gives some margin to tolerate adverse process variation toward direction of the second case. For this, we

have steeper slope around zero phase region. High gain around this area is generally a good thing since it's more immune to jitter. In summary, for a comparator with no dead zone around zero phase region, we want both a0 and b0 to go high for small phase offset. Before investigating what this means in terms of delay requirement, let's see the SPICE simulation of this case. The following plot shows the result when a leads b by 0.01*π.

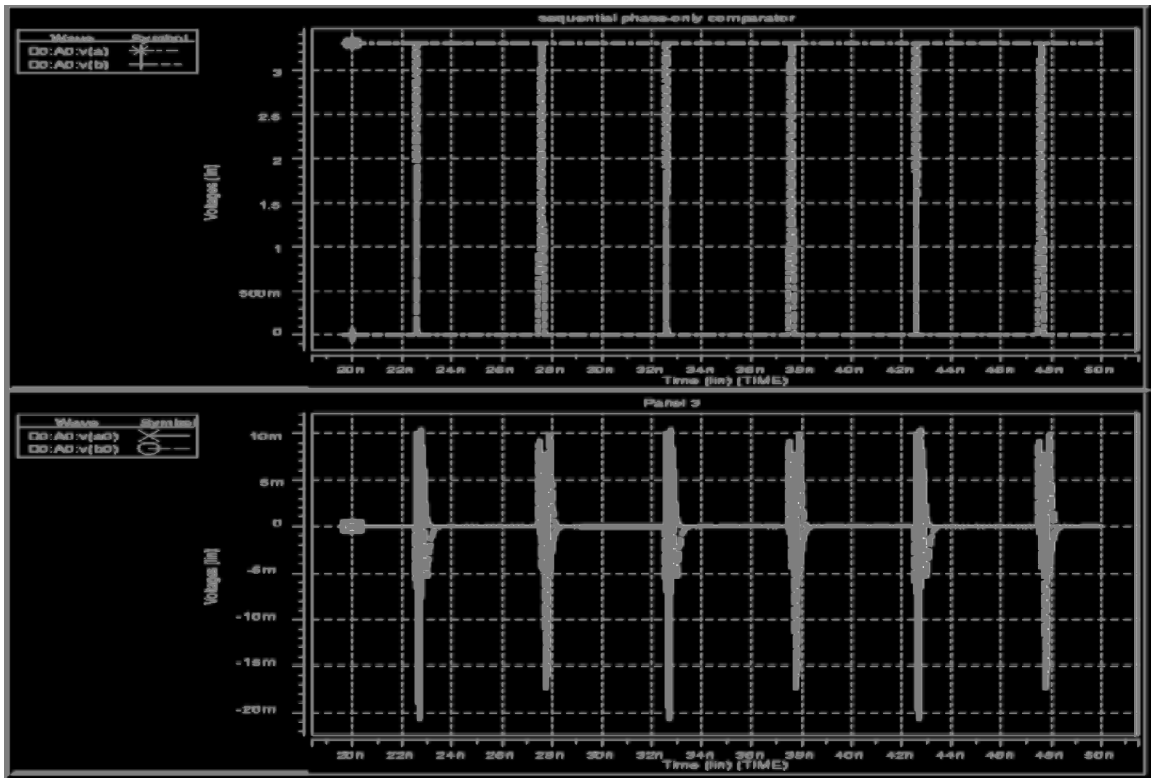


Luckily, we're on the safe side. Note that a0 and b0 goes high when the phase offset is very small. b0 going high is not predicted by the no-delay interpretation of the table.

Given the result, we can go back to the table to see whether it makes sense or not. Consider a case where a leads b by a small amount. Starting from the reset state, when a goes high, the logic will try to go to (a0, b0, x) = (1,0,0) state. However, before a0 has time to change to the new value, b goes high. This is as if the input transitions from (0,0) to (1,1) abruptly in the reset state. So, the state will now become (1,1,1). But this is a non-stable state; thus when the state becomes (1,1,1), the system goes to (0,0,1) 'wait' state right away. If this were a non-delay system, these transitions would have occurred instantaneously, and we wouldn't have seen such noticeable pulse as shown in above figure. But due to finite delay of gates, we see the pulse. Notice that even though I talked of state transition as discrete, (0,0,0) -> (1,1,1) -> (0,0,1) above, this is just a simple way to interpret the result. In reality, all the delays will interact with each other at the same time to produce the waveform above.

In summary, it seems that we want the delay from input change to a0 or b0 to be as big as required for the desired margin so that inputs transitioning close to each other will be interpreted as if they transitioned at the same time. Also, we want the state transition to be slow enough so that there's noticeable amount of time when a0 and b0 are both high. (State stays at (1,1,1) for some time.)

As an aside, the waveforms at the meta stable point (around phase of $\pm\pi$) is shown on the following page. Since there's no noise in our simulation and no feedback to adjust the phase, the system stays in this meta stable state. Notice that a0 and b0 actually shows some activity when the inputs transition, but the magnitude shows that this is negligible, and a0 and b0 largely stay around 0V.


sequential phase-only comparator
Wave
Symbol
D3:A0:v(a)
D3:A0:v(b)
Voltage (in)
3
2.5
2
1.5
1
500m
0
20n
22n
24n
26n
28n
30n
32n
34n
36n
38n
40n
42n
44n
46n
48n
50n
Time (lin) (TIME)
Panel 3
Wave
Symbol
D3:A0:v(a0)
D3:A0:v(b0)
Voltage (in)
10m
5m
0
-5m
-10m
-15m
-20m
20n
22n
24n
26n
28n
30n
32n
34n
36n
38n
40n
42n
44n
46n
48n
50n
Time (lin) (TIME)