CALIFORNIA STATE UNIVERSITY
LOS ANGELES

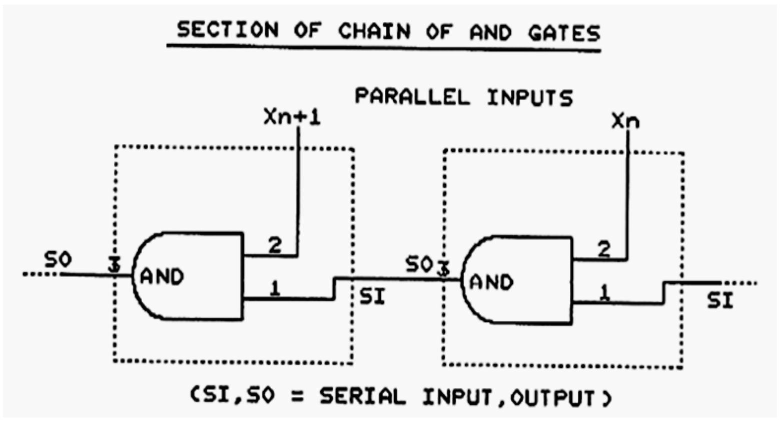
Department of Electrical and Computer Engineering

EE-246 Digital Logic Lab

EXPERIMENT 13
ITERATIVE CIRCUITS

Text: Mano, *Digital Design,* Ch's 3, 4 and 6.
Required chips: **7476:** JK-flipflops (2 chips), and some or all of:
7402: NOR, **7408:** AND, **7432:** OR, **7486:** XOR.

13.1* First read the background discussion in Appendix 1 on iterative circuits, then continue here.

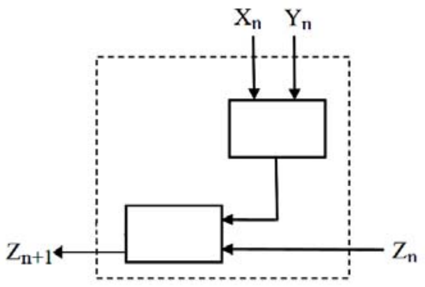


One of the simplest forms of an iterative circuit is a chain in which each stage is just an AND gate with 2 inputs, one parallel and one serial, and a serial output. Similar chains can be constructed of OR or XOR gates. You might use such a chain instead of a single large gate. Thus, a string of 16 AND gates could replace a single 16-input AND, since such large gates are not available in SSI or MSI chips. As discussed in Appendix 1, the relatively long time delay could be a disadvantage.

One possible application for a such a chain of AND gates would be an "All-High" detector. The output of the chain would go high only when all parallel inputs are high. (The serial input to the first gate must always be high; i.e. 5V). This really amounts to single large AND distributed along a chain.

➢➢ Do this in lab: create an "All-Low" detector using a chain of 4 gates, such that the output of the chain goes *low* only when all parallel inputs are low. (Here the serial input to the first gate must always be low; i.e. ground.) Assume inputs, X3..X0 from switches. Use 1 chip (with what kind of gates?). Connect the output of the chain to an LED. *Demonstrate the circuit for your instructor.*

13.2* Build a simplified Iterative Comparator: this circuit will be an inequality detector. It will compare two 4-bit numbers, X = (X3..X0) and Y = (Y3..Y0) using a chain of four identical stages. Each stage contains two different gates, one in parallel, the other in series. The output of the chain should go high if the two numbers are different. (The circuit won't detect which is larger, just whether they are unequal.)



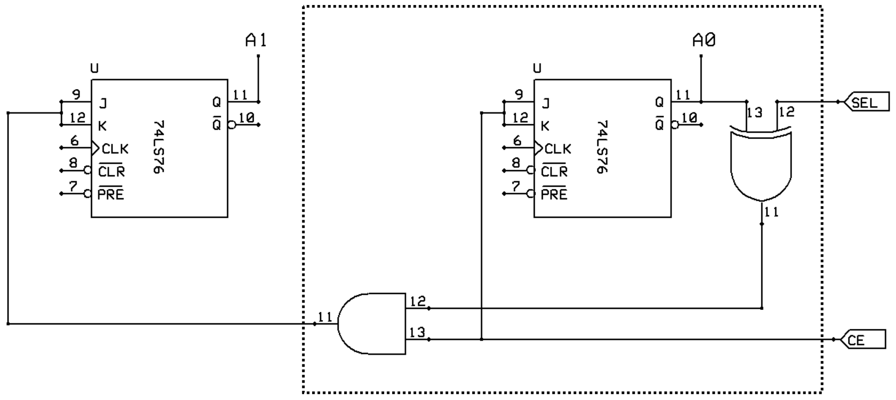
The upper gate checks to see if its two parallel inputs, Xn and Yn, are the same. If so, it outputs a 0; if they are different, it outputs a 1. (What kind of gate will do this?) The lower gate combines this output along with the output of the previous stage, Zn, and passes it up the chain as Zn+1. If even one of the two inputs to the lower gate equals 1, then Zn+1 should equal 1. And if this happens at any stage in the chain, then the output of the entire chain (connected to an LED) will also equal 1 (why?). This indicates that *at least one* pair of Xn-Yn components do not match, so X and Y themselves must be unequal.

Question: should the serial input to the first stage be a 1 or a 0? Be prepared to justify your choice. Your design should use 2 chips only. Try to make all stages identical so additional stages could be added at *either* end. (Can you see how this circuit incorporates an "All-Low" detector?)

Draw the circuit and then build it with the 4-bit inputs X and Y coming from switches. Test it first with X = Y (all bit pairs equal) and then with X ≠ Y (at least one pair different). *Demonstrate circuit behavior for your instructor.*

13.3 Iterative Up/Down Counter: (*design only*). Read Appendix 2 on up-counters, then continue here.

The circuit at right is a 2-bit up-down counter; it counts up 0-1-2-3-0-etc. or down 3-2-1-0-3-etc..



- When CE = 0, counting stops; when CE = 1, counting proceeds.
- When SEL = 0, the counter counts up; when SEL = 1, it counts down.

(NOTE: For something similar to the SEL control used here, refer to Exp 8.1. There a select input changes a circuit from an adder to a subtractor.)

- These flip-flops act like T's because J=K.
- Assume flip-flop CLKs are connected to a pulser. Also, except when flip-flop outputs A1 A0 are initially cleared to 00, the CLR inputs should be connected to Vcc (i.e. made inactive) along with the PRE inputs.

The circuit inside the dotted block (flip-flop A0 and the two gates) is the iterative module. If the counter had more than two flipflops, then each one, except the last (at the left), would look exactly like A0's circuit. This would require that the wire from the SEL input be continued across the whole counter. Because this is only a 2-bit counter, the last flipflop, A1, doesn't need any gates since there is nothing to its left.

Note that if count is enabled (CE=1), then T0 must equal 1 since the least-significant bit of any counter (A0 in this case) always toggles when the clock comes. The question is what happens to A1 when count is enabled and the clock comes--when does it toggle and when not. The answer depends both on A0 and SEL.

1) In the state table at the right, assume CE = 1, which means that T0 is always 1. Fill in the values for T1 and for the next-state values of A1 A0 based on an analysis of the given circuit. Your answers should show whether the circuit is an up-down counter.

Explain your results to your instructor. Next-state values for A1 A0 must be consistent with the values of T1 you show in the table.

PS					NS
A1	A0	SEL	T1	T0	A1 A0
0	0	0		1	
0	1	0		1	
1	0	0		1	
1	1	0		1	
1	1	1		1	
1	0	1		1	
0	1	1		1	
0	0	1		1	

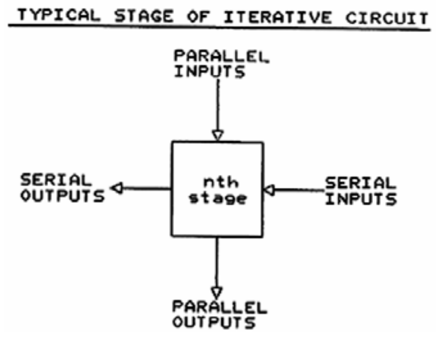
2) Find the rule for down-counters, similar to the one in Appendix 2 for up-counters, from the following count sequence, shown in binary and decimal:

1 0 1 1 0 0 0 0	176
$1\ 0\ 1\ \underline{0\ 1\ 1\ 1\ 1}$	175

Here, again, the flipflops that toggled are underlined. In a down-counter, the last one always toggles (same as in an up-counter). But what rule explains why the other four toggled? Make sure you give a ***clear*** answer to this question in your report.

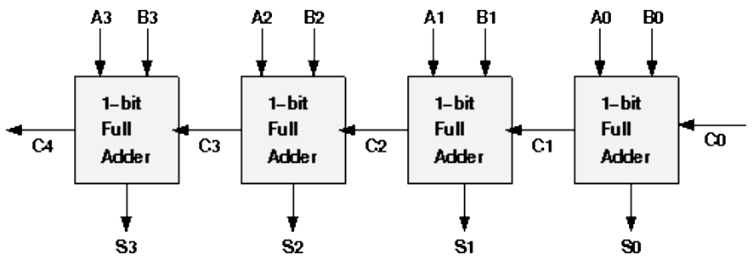
APPENDIX 1

ITERATIVE CIRCUITS



An iterative circuit consists of a chain of identical modules or input stages. Stages can be sequential or combinatorial (i.e. with or without flipflops). Each module can have parallel and serial inputs and outputs. The serial inputs and outputs propagate information from one stage to the next along the chain. A simple example (without parallel outputs) is the chain of AND gates on the first page.

Consider, for example, an N-bit iterative adder consisting of N identical full-adder stages. Each stage includes parallel inputs An, Bn, and an output sum bit, Sn. There are also serial input and output carries, Cn and Cn+1, which carry information from stage to stage.



The advantage to an iterative circuit is ease of design and construction. Designing a full-adder stage with 3 inputs and 2 outputs requires a truth-table with only 3+2 = 5 columns and 2^3 = 8 rows. Once designed, one has only to string together as many stages as desired; i.e. a 4-bit adder would require 4 identical full-adder stages--that's the whole design. Contrast this with designing the entire 4-bit adder at once. It would take a truth-table with 9 input columns (A3..A0, B3..B0, and C0), 5 output columns (S3..S0 and C4), and 2^9 = 512 rows!

The price you pay for ease of design is lower speed of operation. The carries must propagate up the chain of full adders and there is a small time delay associated with each one, say 10 nanosec's. Now consider the case of a 16-stage iterative adder performing the following sum:

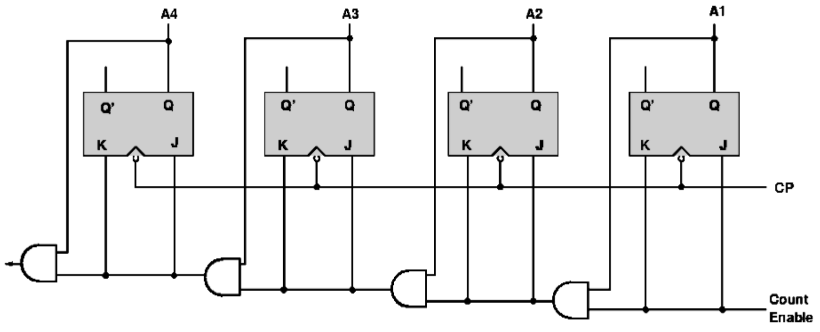
```
  A   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 +B   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
      -------------------------------
      0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

The carry generated by A0+B0 = 1+1 must "ripple" up the chain through 15 more stages before the sum stabilizes at all 0's. This can take something like 16 · 10 = 160 nanosec's or 0.16 μs, which is a long time for many applications. It means that none of the inputs (An or Bn) should change more often than once per 0.16 μs or 6 times per μs. In other words, the data rate into the adder must be less than 6 MHz. For high-speed circuits (like microprocessors) this is much too slow.

To avoid this kind of problem, most commercial adder chips, like the 7483 used in Experiment 8, have extra "look-ahead" carry circuitry which brings carry information immediately to all stages so there is no need to wait for it to ripple up the chain. One could chain four 7483's together and thereby take advantage of this fast look-ahead carry feature to drastically reduce overall delay in a 16-bit design.

APPENDIX 2
ANALYSIS OF AN UP-COUNTER

The following diagram is a 4-bit iterative up-counter, each stage of which consists of a JK-flipflop connected as a "T" (i.e. J = K) plus a 2-input AND which is part of a chain of gates. The serial input to each stage comes from the chain of ANDs and goes to the J-K inputs. (The output-carry from the last AND gate is available if more flipflops are to added at the left.)



Now look at any stage in the chain. When the output of the chain of gates going into that stage is high, the flipflop will toggle on the next clock (CP) since J=K=1. This happens only when the outputs of all previous flipflops are high (Count Enable must be high as well).
For example, A3 toggles only when A2=A1=A0=1. On the other hand, if at least one of the previous flipflops is low, A3 will not change state, since J=K=0.

To see where this rule comes from, observe the following two consecutive count states in an 8-bit counter:

Binary	Decimal
1 0 1 $\underline{0\ 1\ 1\ 1\ 1}$	175
1 0 1 $\underline{1\ 0\ 0\ 0\ 0}$	176

The underlined 5 bits represent flipflops which changed state. The least-significant bit always toggles in a counter, but the other 4 toggled because, *in each case*, all the bits to their right were 1's. The left-hand 3 flipflops did not toggle because each had at least one 0 to its right. Since this is the way counting is done in base-2, the counter must be designed accordingly.
