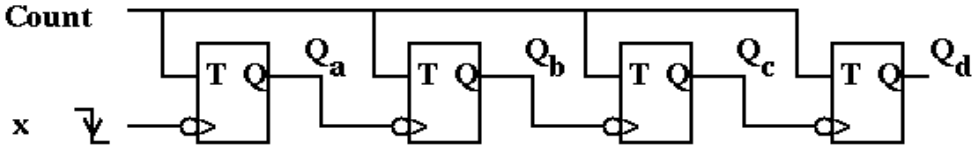
6.111 Lecture # 4

Ss

Counting is a very important function in the digital world, and it is done in a variety of ways

Here is a 'ripple' counter using negative edge triggered T flip flops



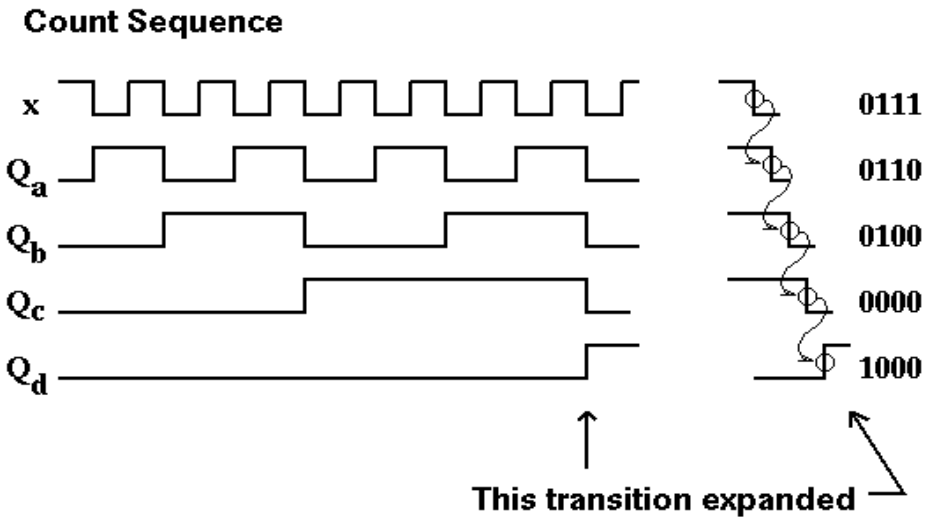
The LSB is on the left in this diagram. It always toggles.

The transition of 1 -> 0 of each 'bit' triggers a toggle of the <u>next</u> most significant bit

Count Sequence:

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

Here is why it is called a 'ripple' counter:



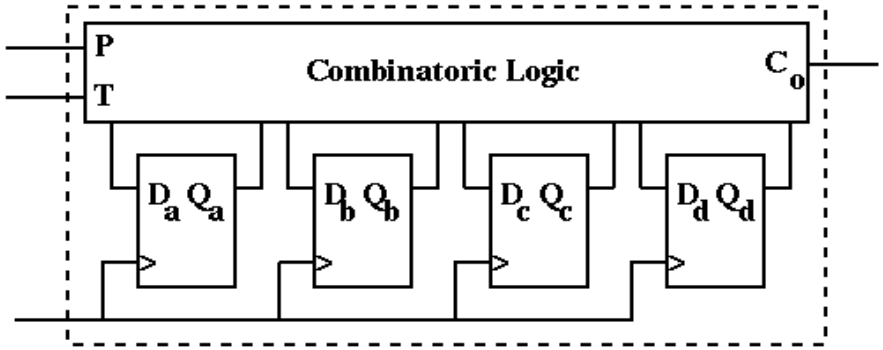
The effect of each input transition must affect all bits, and it does this by rippling through from LSB to MSB

An odd effect is that the transient count is always less than the true count.

Can COUNT fast, but maybe can't be READ fast!

'Synchronous' counters use more logic to reduce the time to stable outputs.

Here is a simplified version of the 4 bit 74LS163 counter



Synchronous Counters: reduce ripple by setting all bits at once

I = P * T
Da = /I * Qa + I * /Qa
Db = /I * Qb + I * Qa * /Qb + /Qa * Qb
Dc = /I * Qc + I * Qa * Qb * /Qc + /Qa * Qc + /Qb * Qc
Dd = /I * Qd + I * Qa * Qb * Qc * /Qd + Qd*/Qa + Qd*/Qb + Qd*/Qc
Rco = T*Qa*Qb*Qc*Qd

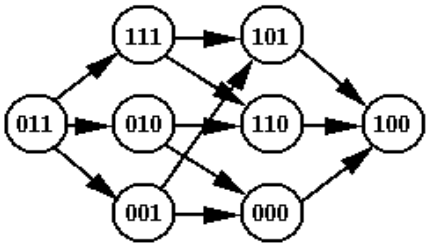
Note that, while all bits of the synchrous counter are set very close to the same time, they may not be set at <u>exactly</u> the same time.

This means that there is a rapidly changing transient state of the counter.

If it passes through all one's it will cause a 'glitch' on the ripple carry out.

You are asked to look for this in Lab 1, but you may not see it!

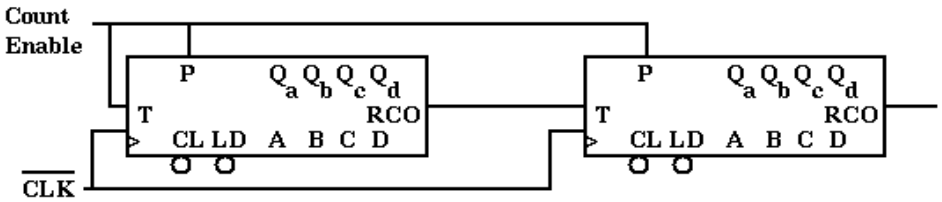
Care is required of the Ripple Carry Output: It can have glitches: Any of these transition paths are possible!



To cascade synchronous counters (to count more bits):

P is "count Enable"

RCO and T are daisy chained

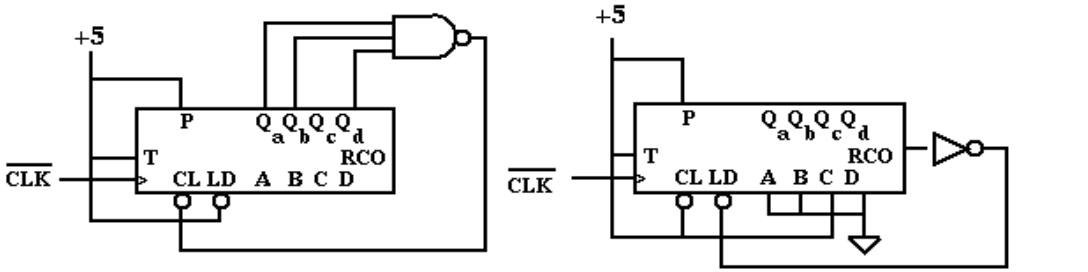


The '163 will 'count' ONLY if P and T are <u>both</u> high

Note that RCO is the AND of all four bits and T.

So if this is input to the T input of the next higher nibble, it indicates that all bits below are set, so the next higher nibble should count.

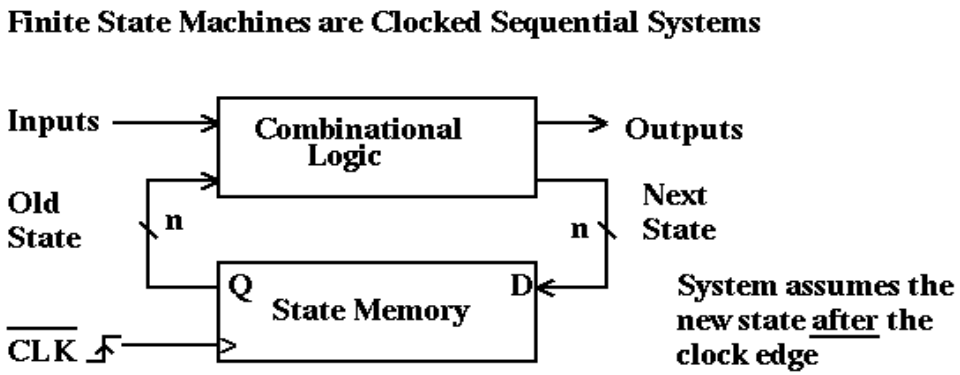
P is 'count enable', and P and T should be tied together ONLY for the least significant 4 bits of a counter.



This one counts 0,1,2, ... , 11, 0, 1 ... **This one counts 4, 5, ... , 15, 4, 5...**

With a little ingenuity, you can achieve all kinds of count sequences. These are both divide by twelve circuits.

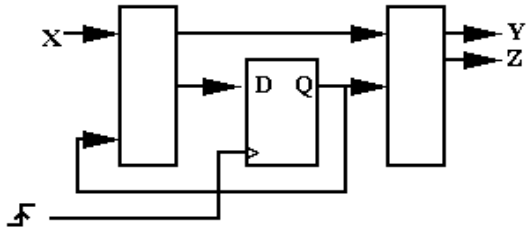
Finite State machines



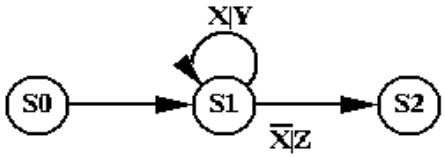
We have already seen simple FSM's in Flip Flops and Counters

But you can do much more complex things with them

> After a clock edge, the 'machine' assumes a state that depends on where it was before the edge and its inputs just before the edge

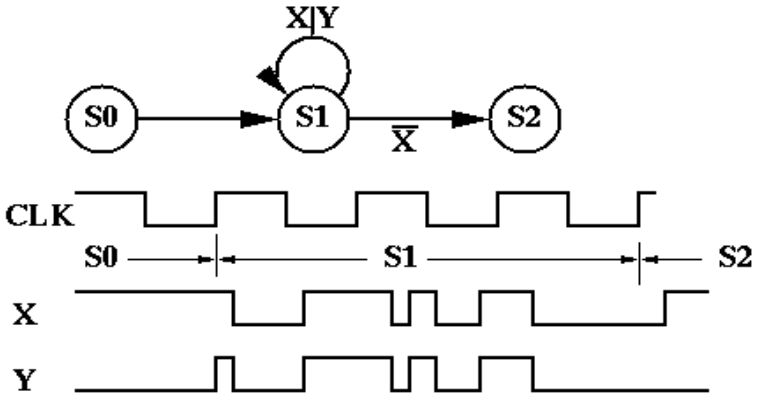


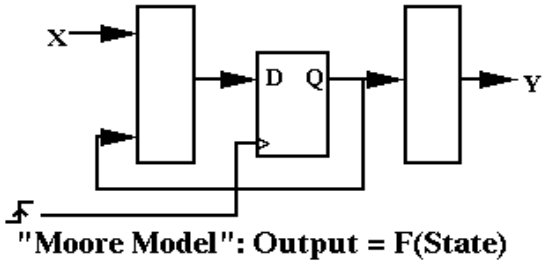
"Mealy Model": Output = F(State, Input)

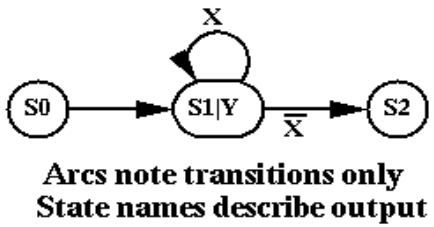


Arcs between states also note output

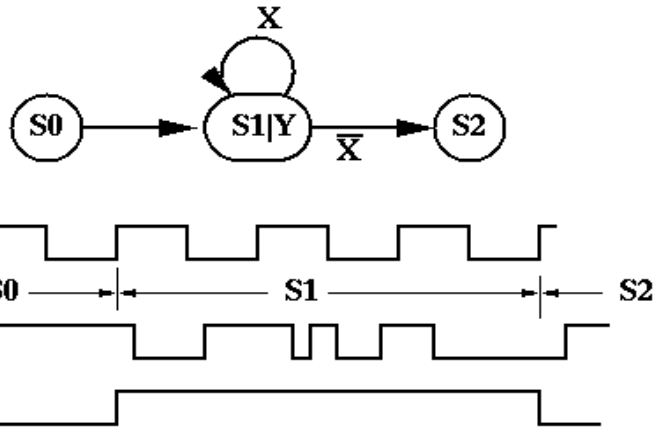
If the input is wired to the output logic, the output can change asynchronously in response to changes in the input.



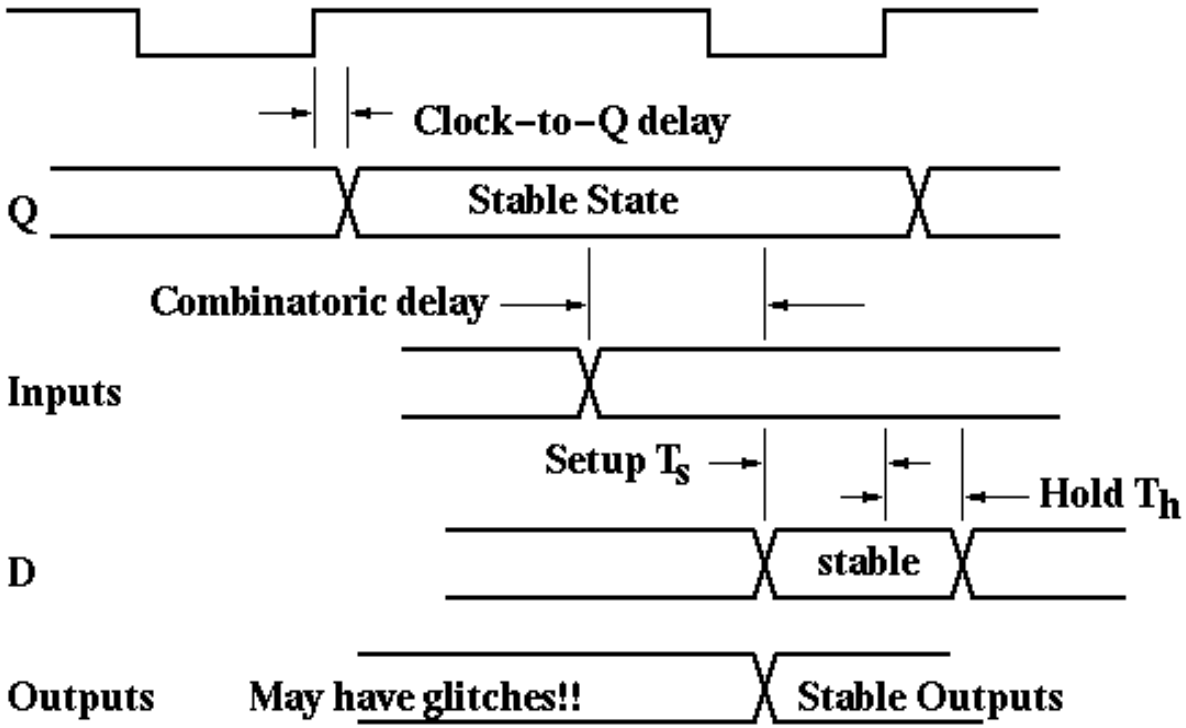




On the other hand, if the input is used ONLY in the next-state logic, the output is fixed during each clock cycle and only changes after the clock edge.

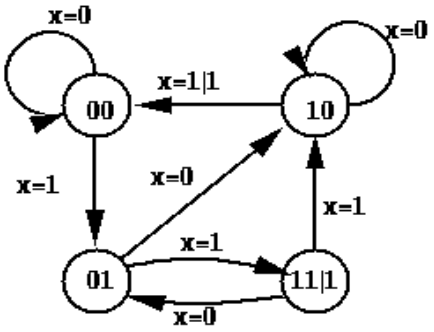


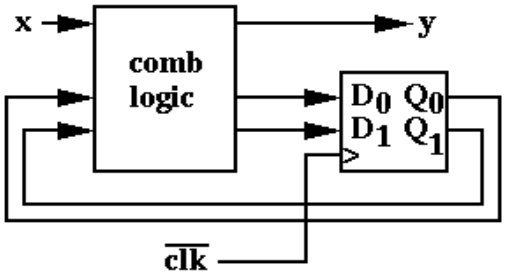
Clock speed is limited by FF delay, Combinatoric delay and setup time.
New state is determined by inputs and old state just BEFORE clock edge



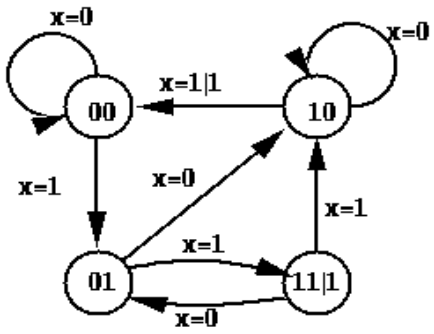
We have automated procedures to build the logic foe finite state machines, but here is an example of a very simple machine.

This is one way of describing an FSM, in terms of transitions on each clock edge.





4 possible states require 2 bits of state. This is a mealey machine



Next State and Output Determination: Logic Specification

Q_1	Q_0	x	D_1	D_0	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	1
1	1	0	0	1	1
1	1	1	1	0	1

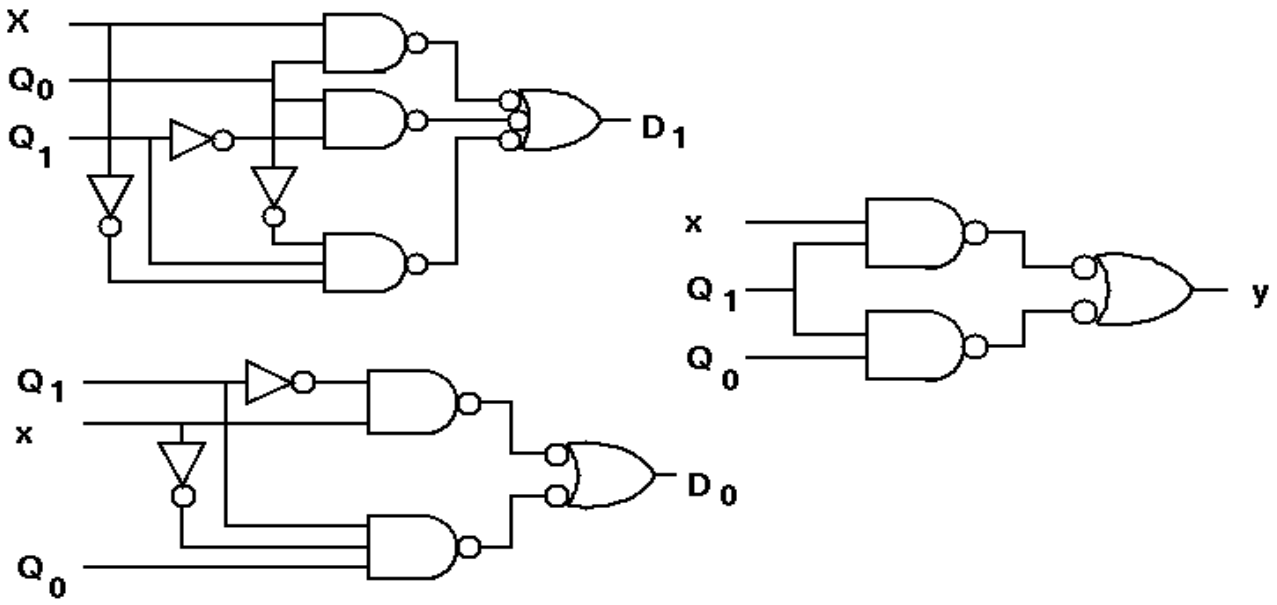
D1 = x*Q0+/Q1*Q0+/x*Q1*/Q0

D0 = x*/Q1+/x*Q0*Q1

y = x*Q1+Q1*Q0

It is straightforward to build a truth table for 'next state' based on 'present state' and input. The output is also derived from the same variables.

Here is the logic that would be required to implement that FSM, if it were made out of discrete gates.



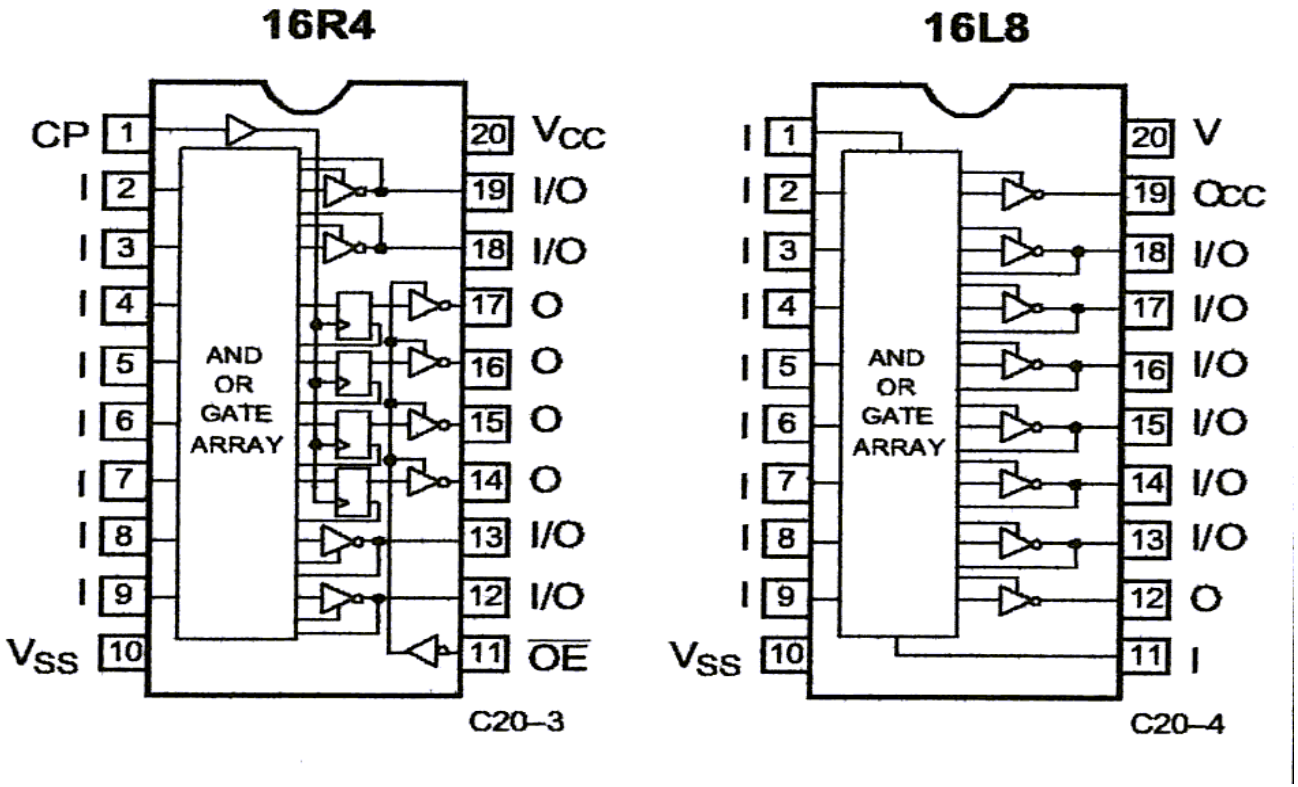
Programmable Logic: Here are two old PALs

Note I, O and I/O pins

Power and Ground are consistently upper right and lower left

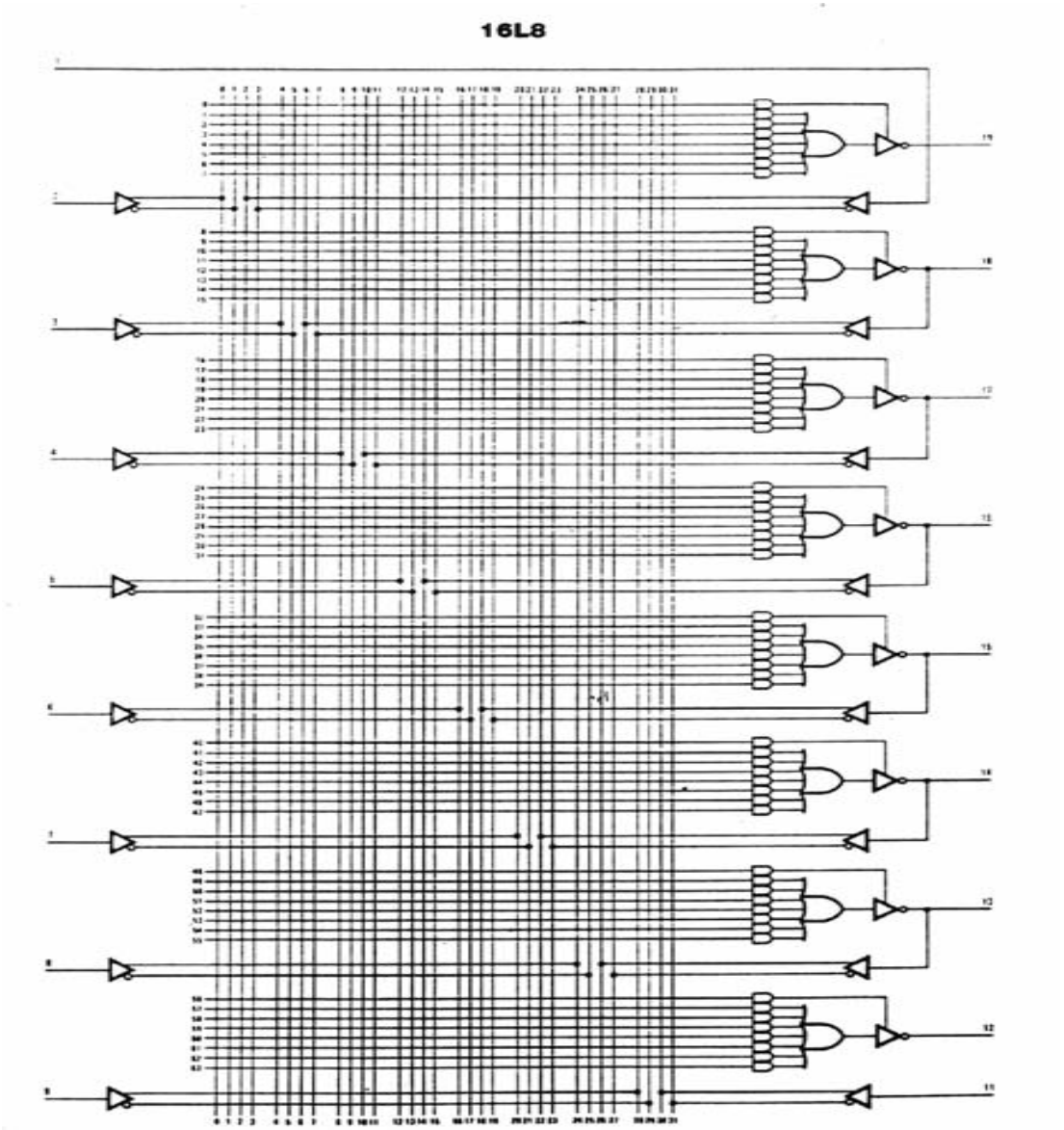
Clock is pin 1 and /OE is lower right, if those are required

These are historic parts: fast, cheap and you probably won't ever see one



Here is a schematic diagram for the 16L8: we can learn more about this by considering its parts.

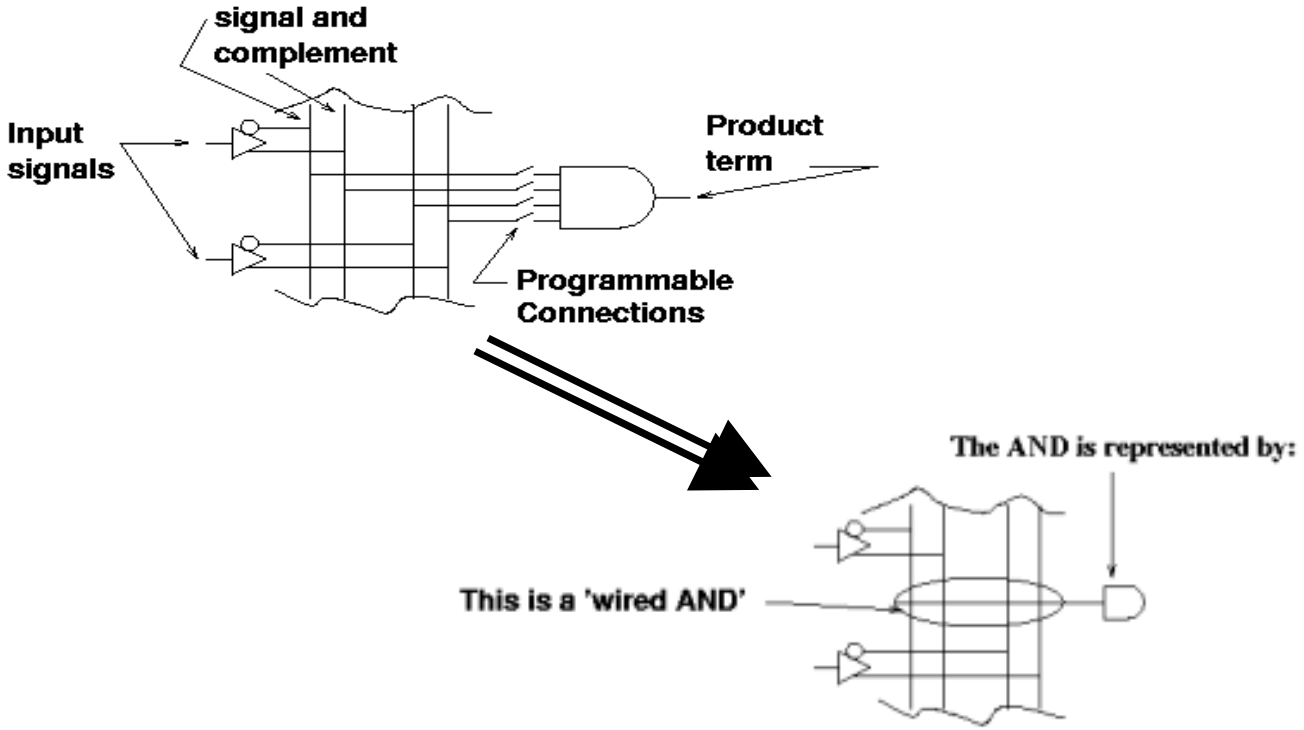
It more complex parts it is not usual to see the whole wiring diagram as you do here.



Programmable Array Logic (PAL)

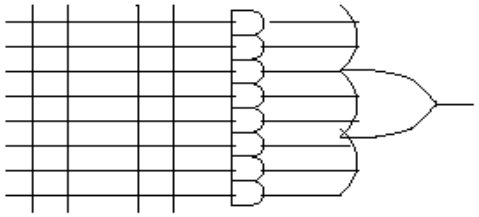
The basic element is the 'product term': essentially a wired AND of input signals and their complements

You can make things like a*b*/c



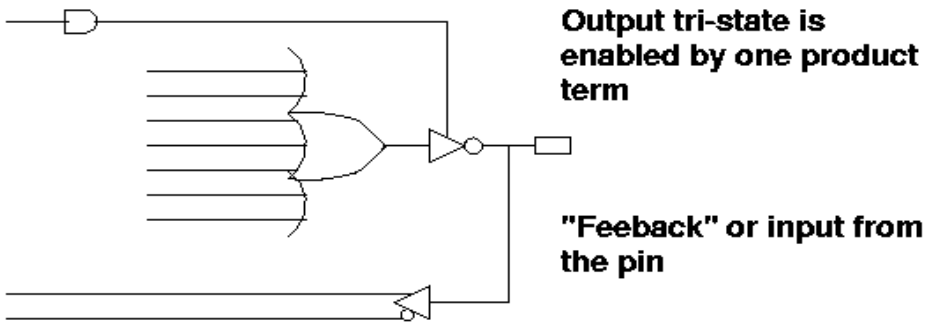
All of these devices synthesize a large OR of ANDs

The product terms produced by the ANDs are combined in large ORs

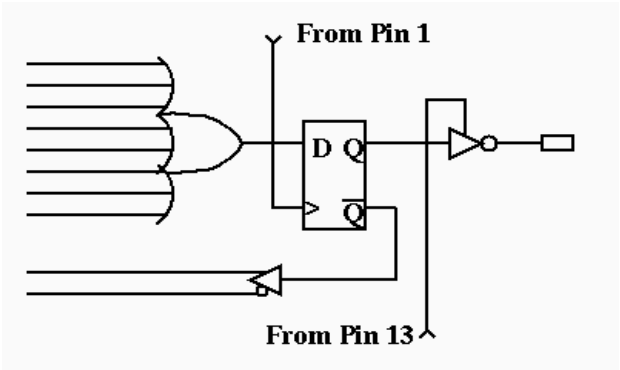


And the output from the OR can be treated in a number of ways

Some older PALs: Output is sent directly to a pin. (16L8)



Or the output can be registered, as in the 16R4



Note that in this case the 'feedback' is from the register, not the pin.
Pin 1 is now dedicated to being the clock input and is not available as a regular input.
Pin 13 (or the lower right hand pin) is output enable and is not available as a regular input

Programmable Logic Devices have become more complex

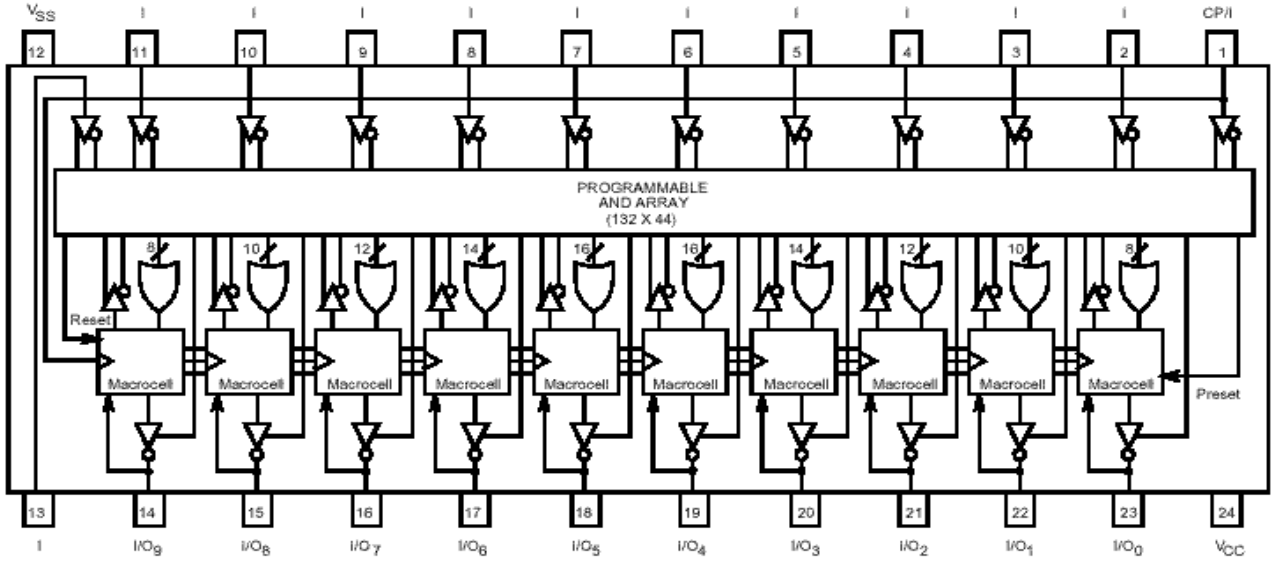
Here is the block diagram level diagram of the 22v10

The Programmable Array is familiar

Note the ORs employ different numbers of product terms

And here the output architecture is also programmable

Logic Block Diagram (PDIP/CDIP)



This is the output logic macrocell for the 22V10

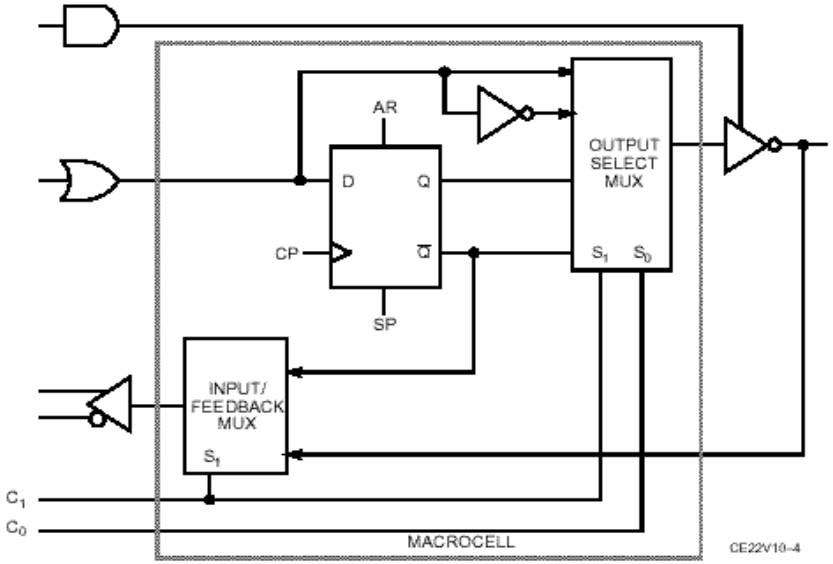
Output enable is derived froma single product term

Output Select has 4 choices:

Direct or inverted

Registered direct or inverted

'Feedback' input is either from the register or from the pin



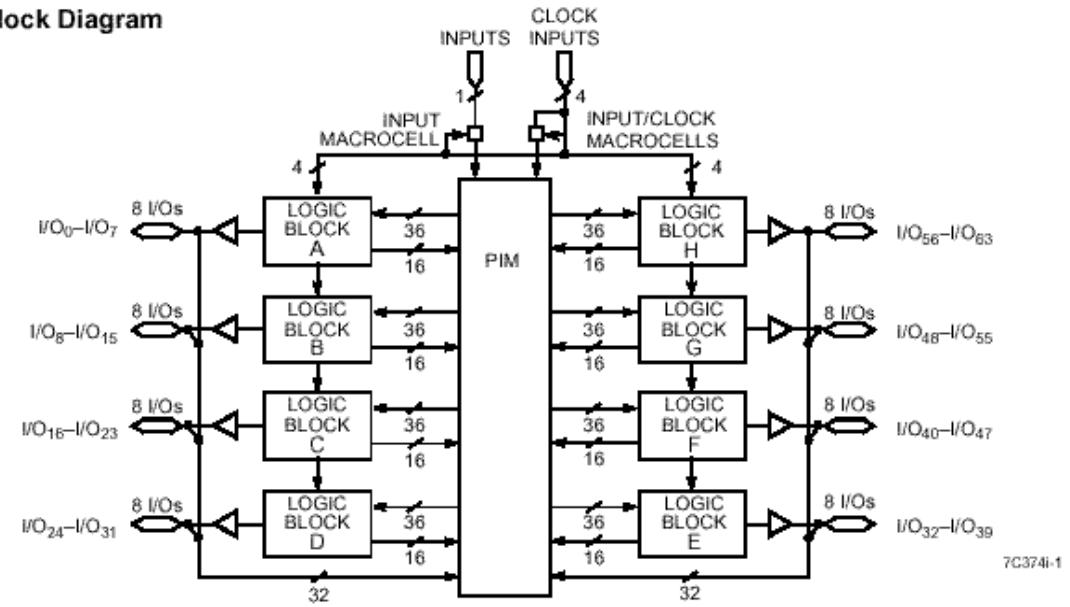
The clock is still from Pin 1

The select bits are programmed

CPLD's are just more complicated PLD's

Here is a diagram for the Cypress '374i part

Logic Block Diagram



Here is a program logic block

Note there are both I/O and 'buried' macrocells

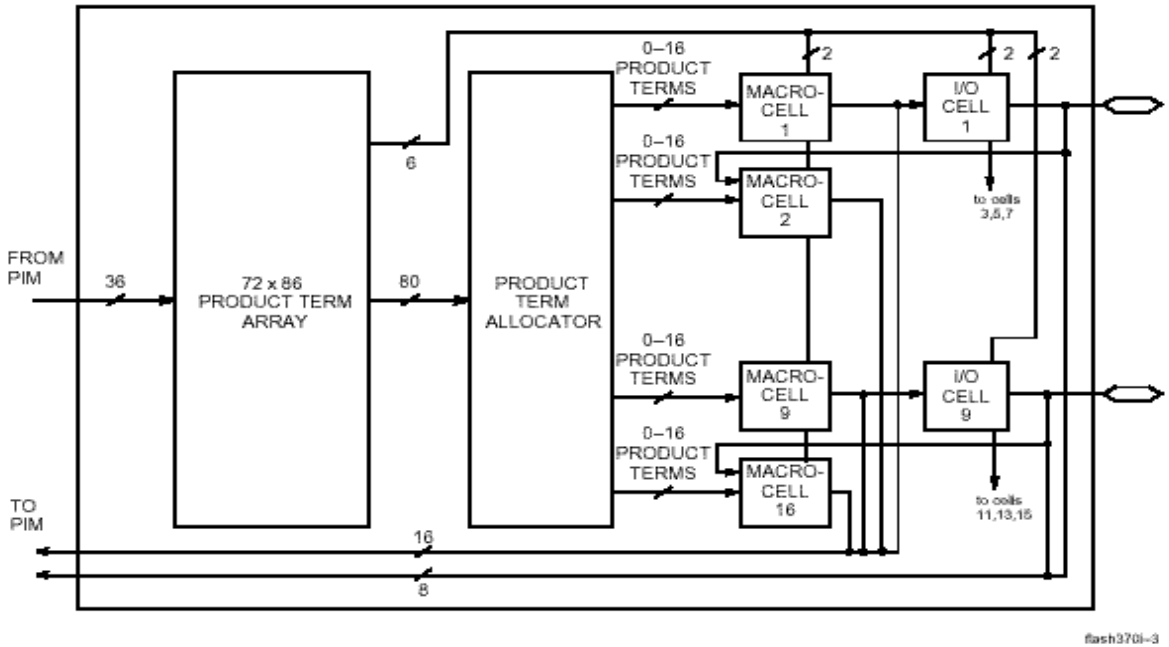


Figure 3. Logic Block for CY7C372i and CY7C374i (Register Intensive)

Input/Output Macrocell (programmable architecture)

Not a lot different from the PAL

Note there are four available clock lines: chosen by a MUX

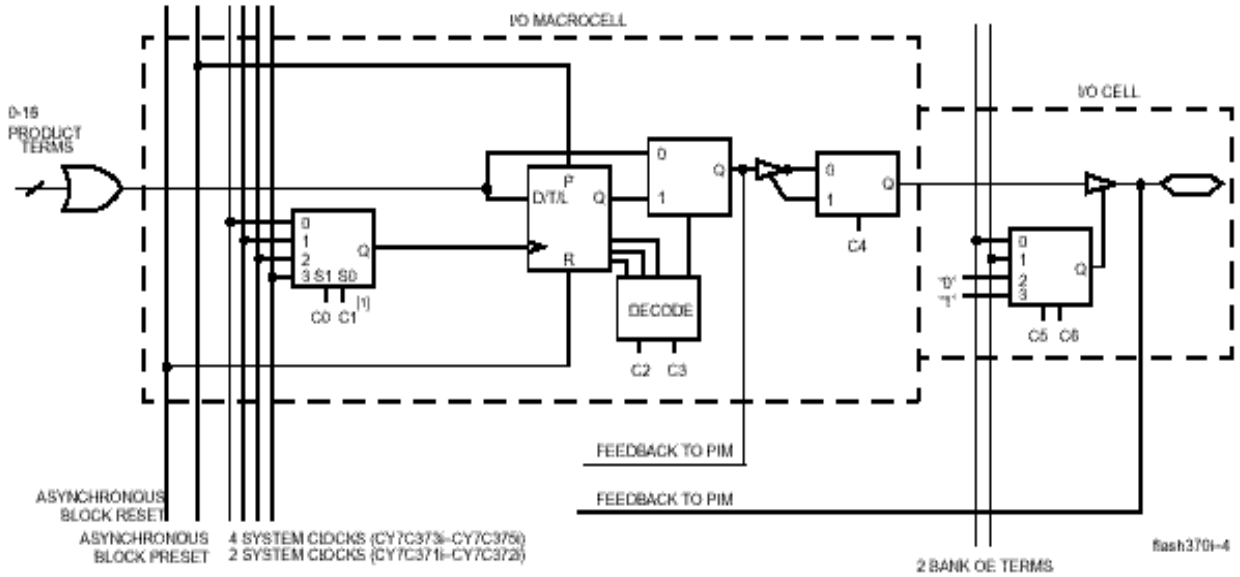


Figure 6. I/O Macrocell

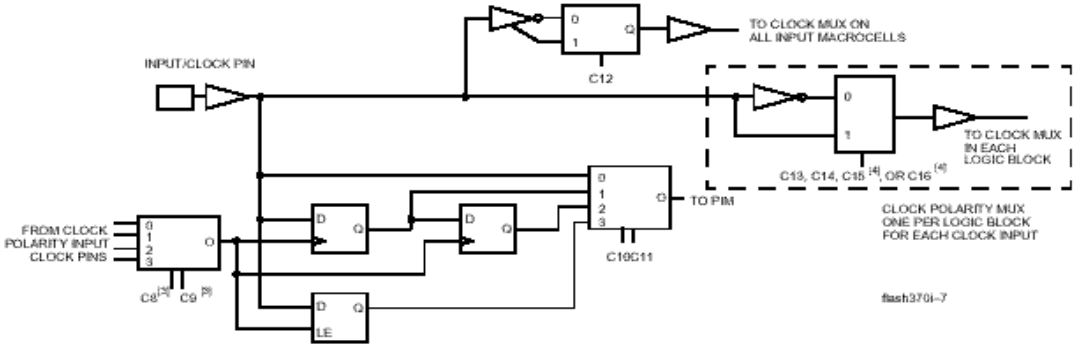


Figure 7. Input/Clock Pins

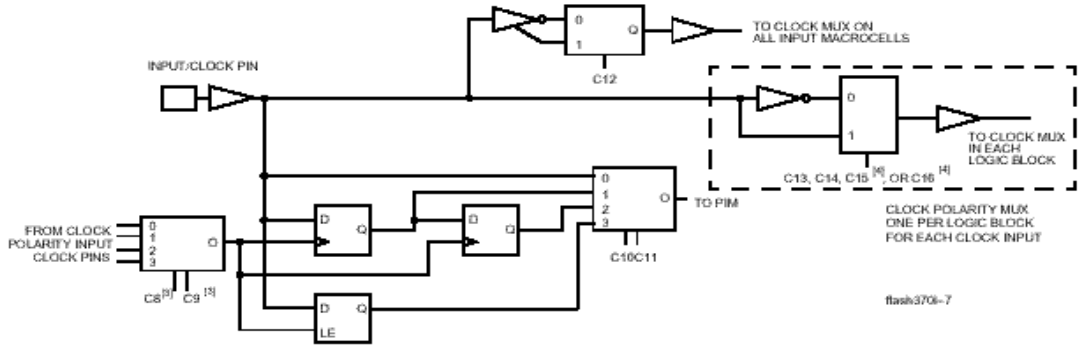
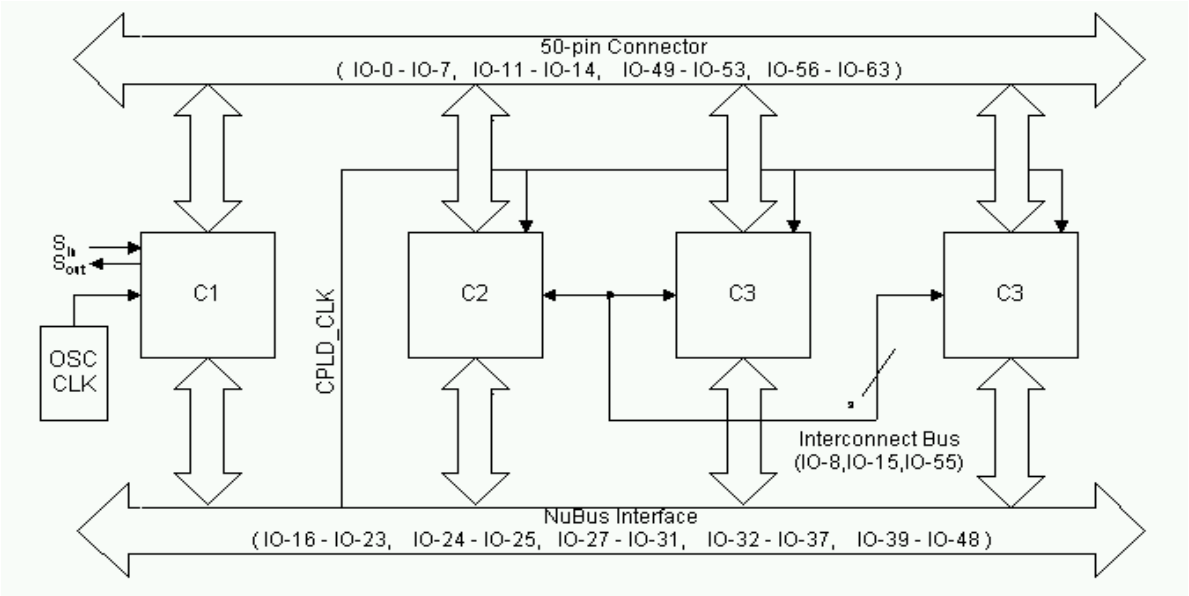
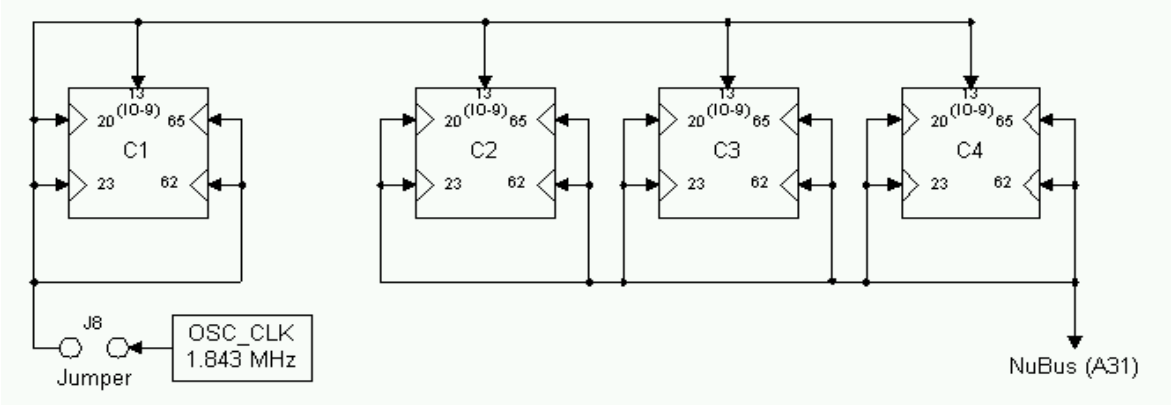


Figure 5. Input Pins

Lab Kits have four '374i parts

Note that interconnections limit flexibility of signal allocation




13
20
(10-9)
65
C1
23
62
C2
C3
C4
J8
Jumper
OSC_CLK
1.843 MHz
NuBus (A31)