Today: STACK CONTROL, The PLA, Finite State Machines
Next week: The silicon gate nMOS process, Layout design rules, examples of layout from stick diagrams

Lecture #4: Thurs 21 Sept:

- Before we begin today's mtl: talk about References, Projects, Seminars.

  > References: Bring books to class & talk over NEXT TIME

  > Projects: Look at schedule. Talk over.

  Mention looking for 2 people to work hourly - to keep the lab open ~ mid afternoons into evening.

  Mention software packages that must be written to fully support effort: SCAN & PARSE a very limited subset of CIF2.0 (as defined in text), instantiate design file by symbol calls referring to symbol defs, & then plot the resulting boxes on HP plotters. However, it will be several weeks before we absolutely need this software.

  > I'll be handing out another book in ~ 2 weeks which will contain more info, examples of cells & corresponding code, etc.

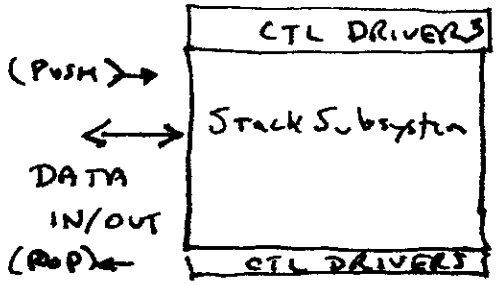
  > SEMINAR SERIES: In addition to Carver Mead lecturing, there are a number of individuals currently actively in research in the area of integrated systems. Some of these people will become very well known as time passes.

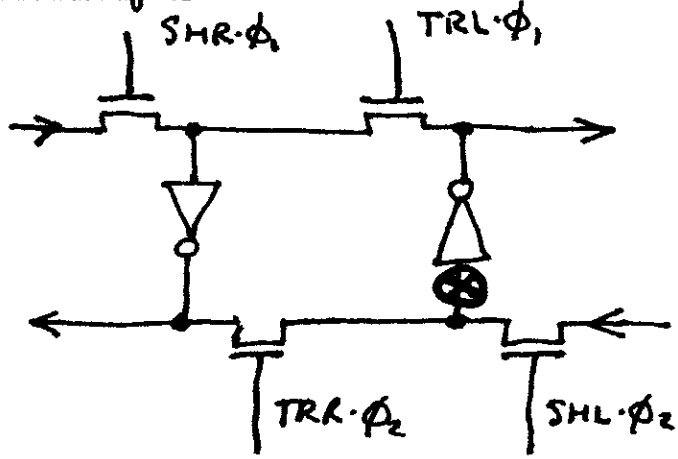
  IF have time

  Mention: Carlo Sequin, Bob Sproull, Chuck Seitz, Doug Fairbairn, Dick Lyon, Wayne Wilner Rich Davies, H.T. Kung, Bob Hon, etc.

- Get a feeling for interest in this series
- Think about times: will hand out questionnaire next week

Continue with Stack Example:





- Note data moves HOR, control lines run vertically.

- Walk thru again: If on ϕ_1, & on ϕ_2, then Fcn:

TRL	,	TRR	→ NOP
SHR	,	TRR	→ PUSH
TRL	,	SHL	→ POP

- **SLIDE: TIMING DIAGRAM:**

> ϕ_1, ϕ_2 always running.

> if NOP most of time, then see that TRL, TRR, occur most of time.

> We need only <u>one</u> control signal (call it "OP") to cause (if <u>on</u> at ϕ_2) the TRR to be followed by $\overline{\text{TRL}}$ and <u>SHR, TRR</u> [PUSH]

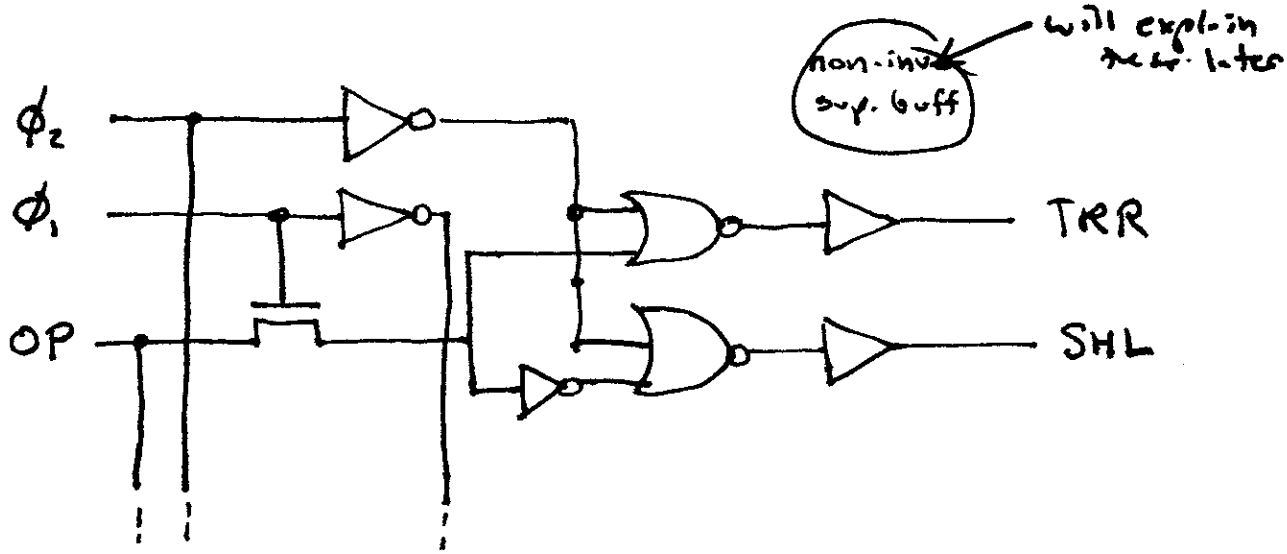
> <u>OR</u> to cause (if <u>on</u> at ϕ_1) the TRL to be followed by $\overline{\text{TRR}}$ and <u>SHL</u> [POP]

Again note: Timing diagrams, even of just sequences of 1's and 0's can be difficult to interpret.

How do we generate TRR, SHL, TRL, SHR ?

From OP, ϕ_1, ϕ_2 to input to drivers which operate the control lines running across the stack:

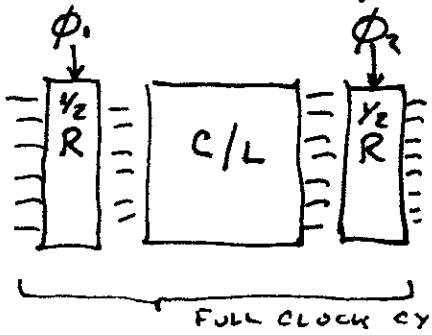
Here's a possible design:



- When ϕ_1 is High, if OP high then drive SHL.
  if OP low then drive TRR.
- When ϕ_2 is high, both TRR & SHL go low, and stay low during the $\phi_2 - \phi_1$ off time.
- note: be careful in these sorts of designs: When ϕ_1 goes off, whichever NOR gate output is high stays high till ϕ_2 comes ON!
- TRL & SHR designed similarly. See Fig 10c. USE FOR ANALYSIS OF OM. BUT, I think it leads to complications
- IMPORTANT POINT: There is a full period between an OP and its next occurrence. Can use to set up counter op with some one line. This can overlap ϕ_1, ϕ_2 ops. Trick used to reduce # micro-code lines.

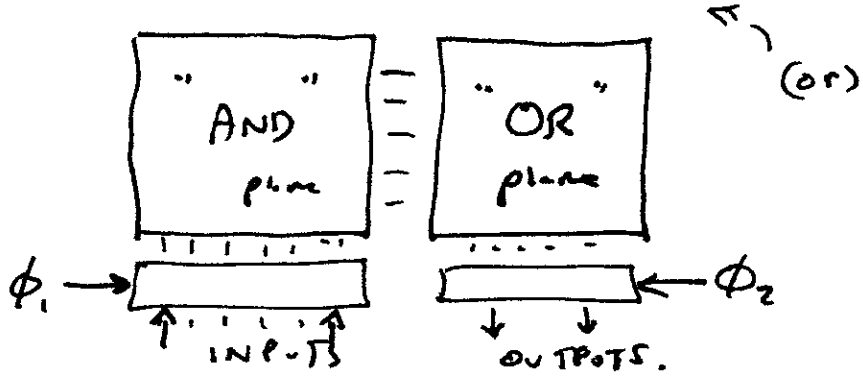
Now, Can make REGs, some forms of C/L. If could only implement arbitrarily irregular C/L in some regular way, we'd have all we need:

The PLA: What we want is C/L to place between Register stages:



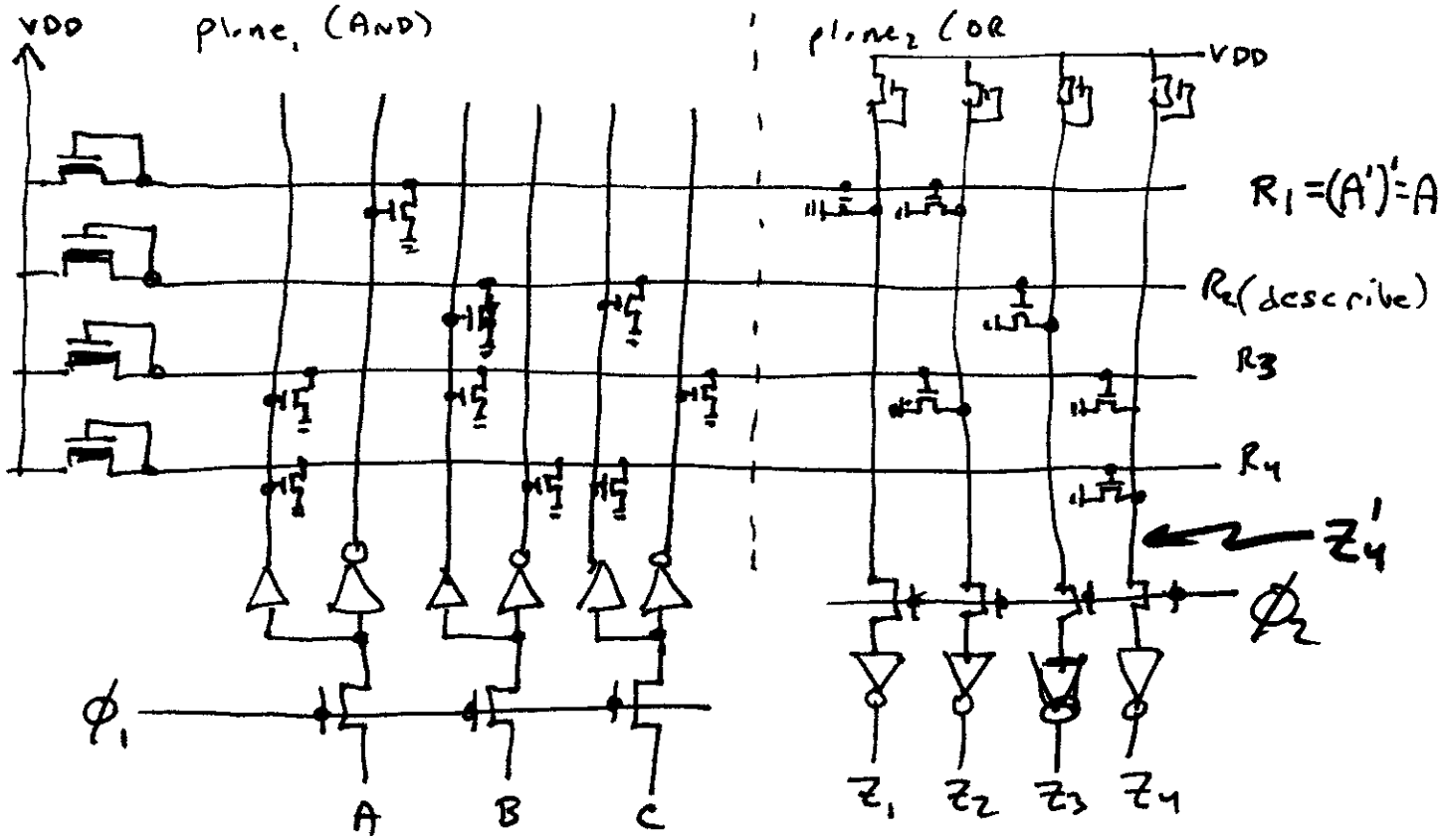
COULD USE A MEMORY, BUT THIS WOULD REQ ALL 2^n poss comb. of inputs × # bits in output. Often wasteful. (mention PROMS)

We impl. the PLA in the following overall subsystem structure:



An Example Circuit: TO ILLUSTRATE FCN OF THE AND & OR Planes. THEY ARE REALLY NOR planes, but as you know NOR-NOR logic (if inputs are avail in TRUE & comp form) can generate all C/L functions of the inputs, just as AND-OR logic can.

EXAMPLE: Note: AM showing the ϕ_1, ϕ_2 registers to indicate how it is <u>imbedded in system</u>. Also to anticipate the Finite State Machine.



<u>AND Plane</u>: If line across plane is high, & FET present, it pulls output low
<u>Notice how the plane, forms the "AND" of the inputs</u>

i.e. (ROW,)

$R_1 = (A')' = A$

$R_2 = (B + C)' = B'C'$

$R_3 = (A + B + C)' = A'B'C$

$R_4 = (A + B' + C)' = A'BC'$

<u>OR Plane</u>: If <u>row is high</u> it <u>pulls vert in. low</u>, <u>so output high</u> <u>(if Transistor)</u>

Notice how the OR plane + Reg row forms the "OR" of input rows

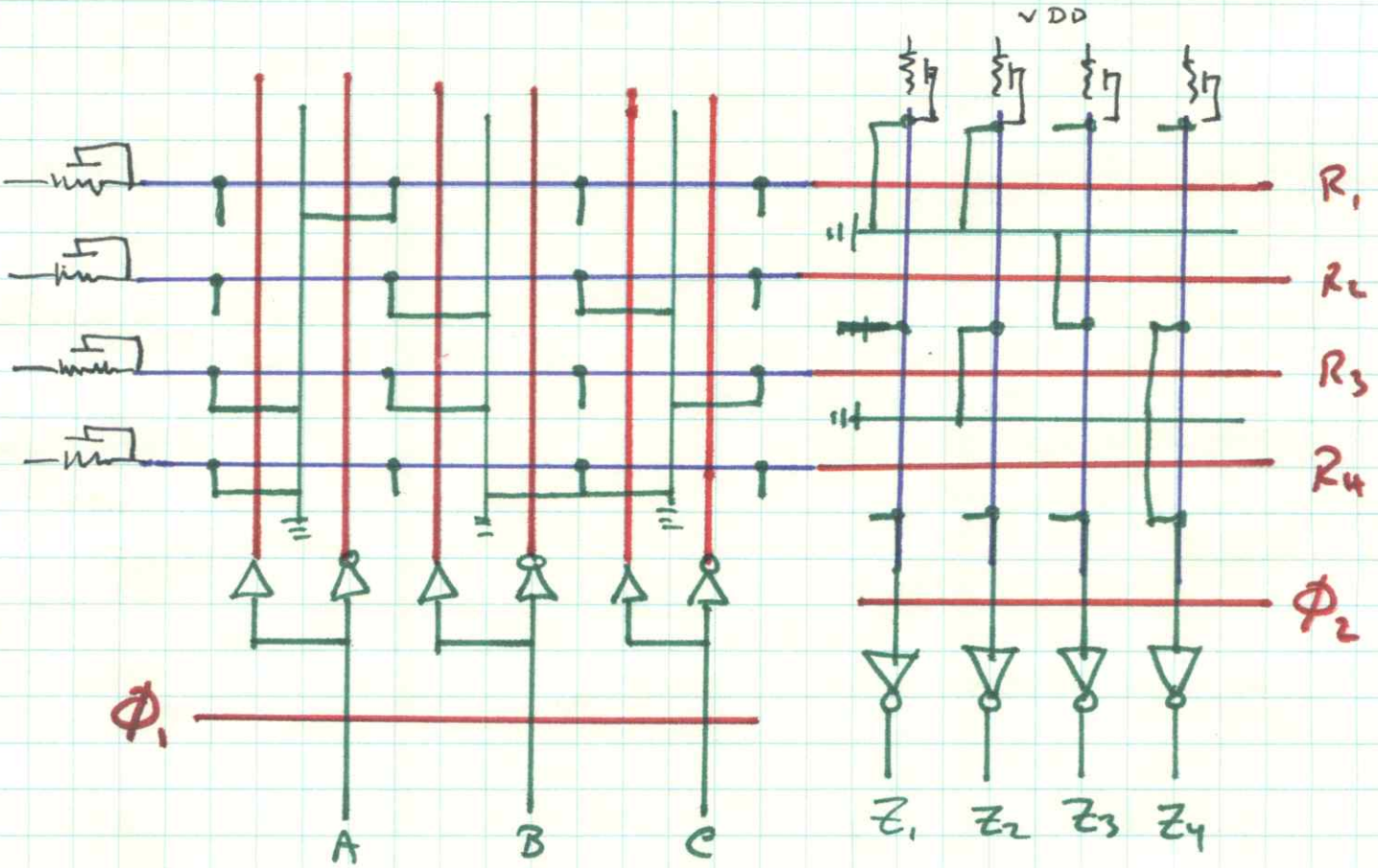
<u>EX:</u> $Z_4' = NOR(R_3, R_4) = (A'B'C + A'BC')'$

Thus: $Z_4 = A'B'C + A'BC'$

[FLIP OVER SHEET/USE WHITEBOARD]

STICK DIAGRAM THE PLA EXAMPLE

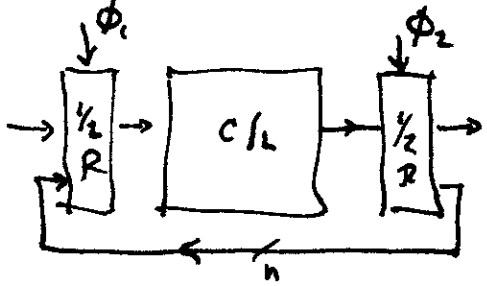
- Run control lines in POLY
- Pullup to output lines in Metal
- Run Ground paths in Diff between alternate poly lines
- Both planes the same, just tilt over an AND plane to get an OR plane.
- Put in input regs/drivers. Pullups.
- Program with transistors at appropriate places.



- The overall size of the PLA is a function of:
  - > # INPUTS, # PRODUCT TERMS, # OUTPUTS,
  - > and the length unit to which we scale our design rules on wire widths/separations (λ)
- Since use NOR form, delays are not too bad.

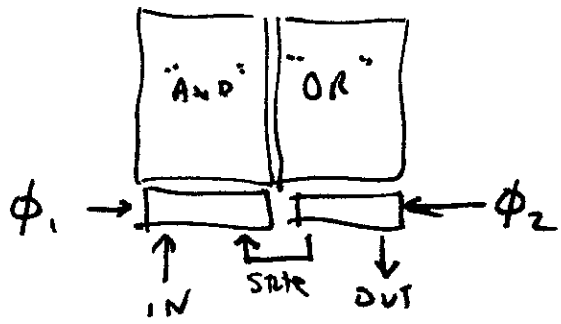
Finite State Machines

- In many cases in the processing of data, it is necessary to know the outcome of the current proc. step, before proceding with the next.
- The results of the present may be used as inputs to the next. They may determine which of several possible next steps we select to do.
- The following configuration can be used to implement a processing step having this regmt:



Some of the outputs are fed back around to the input register.

- This implements a Finite State Machine. The machine has a finite number of states as encoded by feedback paths (# states = 2^n). Now the output and next state are C/L functions not only of the input, but also of the present state.
- We'll usually use the form: PLA finite State Machine



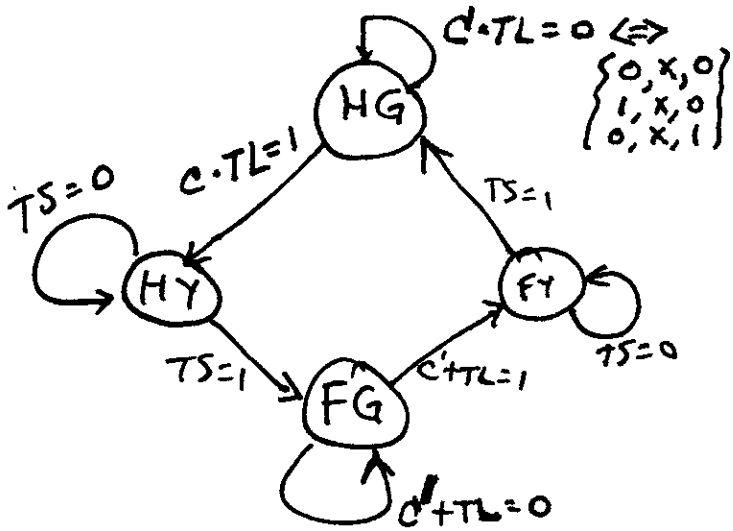
Show part #

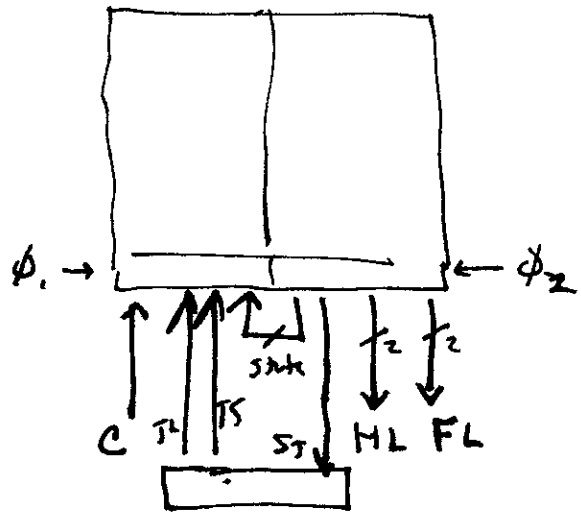
Design a Traffic light controller:

Lets go thru a complete example: **SLIDE**.

- Busy highway intersected by a seldom used farmroad.
- Detectors installed which cause a signal C to go high when cars are at either point (C).
- If no cars are at position C, we wish to control the traffic light so that it remains green.
- If cars are present, want highway lights to cycle thru caution to red, and then farmroad light to green.
- Farmroad light to remain green only while detectors signal cars present, but never longer than some timeout. Farmroad light then cycles thru caution to red, & HW light returns to green.
- Highway light is seen not interruptable again for some fraction of a minute.
- We usually begin by sketching out a **state diagram** consisting of circles and arrows, circles indicating states, arrows indicating poss. transitions, what input causes them, and what output results:

In this case: C, TS, TL





- Describe The Symbolic Transition Table:

An alternative form is shown in (Fig 15d).
This tabular form begins a procedure for mapping the function of the state diagram into a PLA Finite State Machine.

The idea is: In Pres. State IF input is (next state will be) and output will be

(In Pres. State IF input is) — Symbolic Input in ϕ_1

(next state will be and output will be) — Symbolic State Output in ϕ_2

- Describe the encoded transition Table:

We now simply assign binary codes to the states, inputs, outputs to form an encoded transition table. (Fig 15e)

- Now, an algorithm described in Text Ch3 p 24-25 indicates how to construct the stick diagram of a PLA FSM having correct # inputs, state lines, outputs, rows & how to Program / Place the transistors as a function of the encoded transition table entries

You should study this example an convince yourself that the stick diag. impl. the tabulated function.

Do Before Traffic Light Controller

- Mention Problem 8: Draw state diagram, and walk thru possible transitions, clarifying notation.
- Problem 8 is just to implement in a PLA, FSM for already defined problem. We'll do more designs in future assignments where you'll have to create the starting state diagram, given a written description of the problem.

- SIZE: Note: We'll find that the Traffic light controller FSM even in '78, occupies only ~ $1/125^{th}$ of a chip. It can run at a clock rate $\sim 10^7$ times as fast as the real-time prob. requires.

  By late 80's, it might occupy ~ $1/25000^{th}$ of a chip, and run 10^4 times faster.

- Discuss Two Motivational Subjects:

  use of arrays of FSMs, FSM vs "microprocessors"

  > H.T. Kung's array processor algorithms.

  > Image processing right in a display.