Lecture #9 — October 12

- **COURSE REGISTRATION**: TO GET LAB ACCT & NEXT BOOK, MUST BE ON LIST
- Variety of things today: Introduction to digitizing/encoding layouts by use of a symbolic layout language.

  Introduce another example subsystem (sorter). Talk over your first project assignment, & prepare to begin the lab work.

- Briefly discuss HW#3: (a soln on whiteboard?)

Many got reasonable solutions to problem #9. Quite a few didn't. The lesson: Word descriptions don't define state machines in real cases. State diagrams or other formal things do. There were things unspecified in the problem: actually how to start the machine. Are one-bit messages allowed? Those interact. There are many solutions. Be sure however to note that you must get a double error at MSG ending in 0110 to satisfy the part that was stated.

The layout problems. I guess most of you visualized the process and design rule material quite well: most of the layouts had no design rule violations. As usual, some of the really compact layouts raised questions about how to really interpret the design rules, especially around the butting contact between gate and source. (Most common errors: incorrect red/green in BC)

Some compact solutions: (subst. smaller than most)

Problem 10:
- Guy Steele $304\lambda^2$
- Dean Brock $324\lambda^2$ (right ✱'s only)
- Lynn Bowen $324\lambda^2$

Problem 11.
- Gerald Roylance $1302\lambda^2$
- Michael Coln $1357\lambda^2$
- Guy Steele $1404\lambda^2$
- Dave Otten $1408\lambda^2$

BRIEF REVIEW OF IMPLEMENTATION:

- How do we get our designs implemented?
  GO THRU SLIDE SEQUENCE:

  PATTERN GENERATION, STEP & REPEAT TO GET MASKS, PROCESS, & PACKAGE

- KEY FIRST STEP IS PATTERN GENERATION. "FLASHING" BOXES ONTO A "PHOTOGRAPHIC PLATE", WITH A SORT OF "PROJECTOR"

- PATTERN GENERATOR DRIVEN BY A MINICOMPUTER, AND SIMPLY FLASHES SEQ OF RECTANGLES EACH HAVING [x, y, h, w, a] values, AS FED TO THE MINI BY A TAPE CONTAINING THE "PG FILE" FOR THE DESIGN, IN AN APPROP. FORMAT

- NOTE: This thing isn't particularly smart: if you had one simple cell repeated over and over, you'd still have a huge PG file.

- THE PG Files for project sets may contain Hundreds of Thousands of rectangle items. LOTS OF DATA.

- So that's an output file. We sure don't want to directly encode our designs in such a way. We need a way of encoding cells as "symbols" and then being able to call and repeat them in some reasonable way.

Symbolic Layout Languages:

~~What we're trying to describe is kind of like two dimensional~~ (~~potentially~~) ~~object code~~ .... We could use something like a macro-assembler where we define Macro's and then Call them to insert code in place.

Let's Look at an informal example:

SLIDE OF SRCELL — SHOW CELL ENCODING.

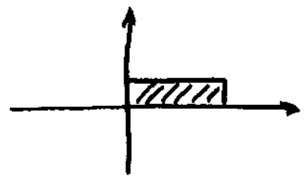
SLIDE OF CODE — SHOW ITERATION, TRANF. TO GET AN ARRAY ...

Mention PLA code. SHOW FIG IS SLIDE

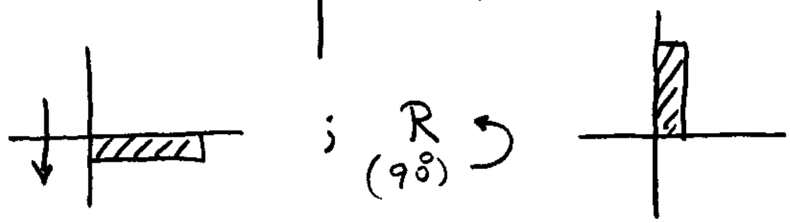
Ah, but before we get carried away:

- There are dangers lurking that you might not anticipate:

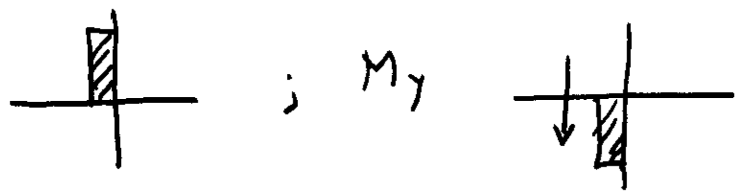
An example:



(i) M_y



(ii) $\underset{(90°)}{R}$



- In general, sequences of MIRRORINGS, ROTATIONS, & TRANSLATIONS produce an overall effect dependent on the order.

- It is non-trivial therefore to specify the semantics of even the simplest Layout languages: i.e. avoid continual misinterpretations of what particular encodings mean.

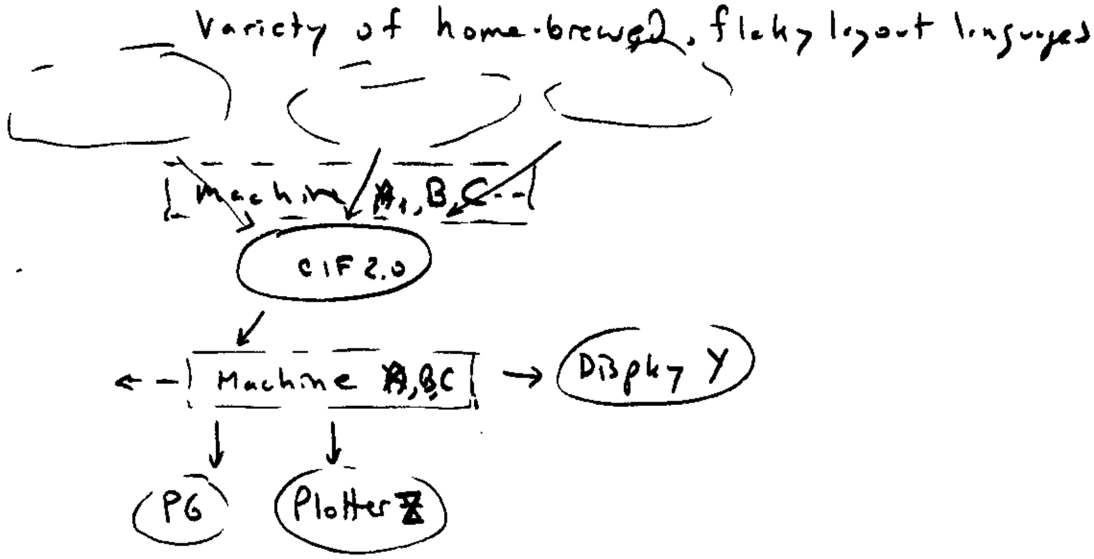
BACKGROUND:

There is no generally accepted, defined, documented layout language. We're about back in the "early 50's" phase of software history.

BUT:

There has been an effort to define a common design interchange language -- to avoid the problem of everybody having to keep writing new file conversion programs ...

So: Where are we?



MORE ABOUT THE INDUSTRY: MOST "ADVANCED" PLACES USE INTERACTIVE LAYOUT SYSTEMS. WE'LL SEE MORE ABOUT THAT LATER: BUT UNDERNEATH THOSE ARE DATA STRUCTURES AND OPERATIONS SOMETHING LIKE THOSE OF SYMBOLIC LAYOUT — ONLY MANIPULATED DIRECTLY BY COMPUTER. ~~GRAPH~~ ACTUALLY, THE SYMBOLIC LAYOUT APPROACH IF DONE AT A HIGH ENOUGH LEVEL, HAS ADVANTAGES NOT DIRECTLY AVAIL. IN STRAIGHT INTER. GRAPHICS SYSTEMS. (SYMBOLIC MANIP CAN BE VERY POWERFUL)

- WE'LL SEE AN EXAMPLE OF A RELATIVELY SOPHISTICATED SYMBOLIC LAYOUT LANGUAGE IN THE NEXT BOOK

- MEANTIME WE'LL GO BACK TO BASICS AND LEARN HOW TO CODE IN THE INTERMEDIATE FORM.

- This will be supported in the LAB: i.e. Text files of CIF2.0 code can be plotted on the plotters. Also, all the software necessary to generate/sort PG files is ready to go at PARC — CIF files thus can be shipped off for actual implementation

- HOWEVER, ANYONE WISHING TO EASE THEIR LAYOUT ~~EN~~CODING TASK COULD WRITE A PREPROCESSOR TO TRANS. X INTO CIF

CIF 2.0 [A Human Readable Intermediate Object Code]

- On reading at this time, not necessary to go into details unless you wish to. We will use only a small subset - to get our projects going. We'll learn mainly by examples.

- **But first correct ERRORS / HANDOUTS** 2

The Subset: You should learn how to specify (interpret) in CIF2.0:

- Distances. Integer values in units of hundredths of a μm

- Coordinates: Right handed coord. system, incr. y up, increasing x right. Interpreted as front surface of chip (not intermediate artifacts)

- Directions: Specified by 2 integers: a direction vector

  First is component along X, second along Y.
  Thus: (1 0) is in + ~~X~~ direction.

- BOXES: Box Length 25 Width 60 Center 80,40 Direction -20,20;
  (in X)

  [Note: COMMANDS FOLLOWED BY ;
  Note: CIF file is Sequence of Commands terminated by an END marker E]

- Recommend using Shorter encoding:

  BOX: B25 60 80 40 -20 20;

  [Note: Direction defaults to +X. We will use only Boxes at right ∡s. So don't need direction]

LAYER SPECIFICATION:

- A MODE IS SET WHICH APPLIES TO ALL SUBSEQUENT GEOMETRIC PRIMITIVES (BOXES) UNTIL SET AGAIN.

Layer ND;	or	LND;
		LNP;
(etc.)		LNC;
		LNM;
		LNI;
		LNB;
		LNG;

- **SYMBOLS**: This facility in CIF provides a means to greatly reduce the size of most intermediate form files compared to the PG file to be generated.

- The symbol facility in CIF is deliberately limited in order to avoid mushrooming difficulties of implementing programs that process CIF files. For example:

  > Symbols have no parameters.
  > Calling a symbol does not allow the symbol geometry to be scaled up or down.
  > There are no direct facilities for iteration. This is primarily due to the difficulty of defining a standard method of specifying iterations without introducing machine dependent computation problems.

- However, it is still possible to achieve much compaction by defining several levels of symbols:

  i.e. cell, row, double-row, array, etc.

DEFINING SYMBOLS:

- Precede the symbol geometry with a DS command;
  Follow " " " with a DF "

- $\boxed{\text{Definition Start \#57 A/B = 100/; ---; Definition Finish;}}$
  > OR: $\boxed{\text{DS 57 100 1; ---; DF;}}$
  > The first argument is the symbol's identifying number.
  > The DS provides a way of scaling distances using literal values of A & B:

  As the form is read, each distance (position or size) is scaled to: $\boxed{(a * \text{distance})/b}$

  > Thus if the designer wishes to use a grid of 1 micron, the symbol definition might cite distances in microns, and specify $a = 100$, $b = 1$ to convert to integers in units of 1/100ths μm. Or, use $\lambda = 3\mu m$, grid. Be careful: Integer distances only. This reduces the number of characters in files, and may improve their legibility. This isn't Scaling. A symbol is defined with absolute distances.

- DS's may not NEST.

- However, DS's may contain CALL's of other symbols, which may in turn CALL other symbols.

- A Symbol must be defined before it is called.

  (Put symbol defs first)

CALLING SYMBOLS:

- The CALL command takes a specified symbol, and specifies transformations (TRANSLATE, MIRROR, ROTATE) to be applied to it to "place an instance" (instantiate) of the symbol at a particular location in a particular orientation.

- $\boxed{\text{Call Symbol \# 57 Mirrored in X Rotated to -1,1 then Translated to 10,20.}}$

- alternatively: $\boxed{\text{C 57 M X R -1, 1 T 10, 20;}}$

- NOTE:

C 1 T500 0 MX	adds 500 to the X coords, then mirror in X.
C 1 MX T500 0	Mirrors in X, then adds 500.

- The order is important. Intuitively, each transformation is applied in sequence.

- SYMBOL CALLS MAY NEST. A symbol DEFINITION may contain a CALL of another symbol. However, no direct or indirect recursion (calling itself)

- WHEN CALLS NEST, it is necessary to "Concatenate" the effects of transformations specified in the sequence of CALLS.

- LAYER SETTINGS PRESERVED ACROSS SYMBOL CALLS & DEFS.

- WITHIN A SYMBOL DEF, LAYER MODE IS IMPLICITLY RESET BY THE DS, DF COMMANDS: DS to NULL (must be specified), DF to previous value.

- BUT I'D USE SIMPLE PROCEDURE OF CLOSELY COUPLING LAYER SPECIFICATIONS & THE ENTITIES THEY ARE FUNCTIONALLY ASSOCIATED WITH, TO AVOID ERRORS.

<u>TRANSFORMATIONS</u>: The primitive transformation are:

- <u>T point</u>: Translate the current symbol origin to this point (translate and place an instance of the symbol (origin) at this point)
- <u>M X</u> : Mirror in X : Multiply X coordinates by -1.
- <u>M Y</u> : Mirror in Y : Multiply Y coordinates by -1.
- R point : Rotate Symbol's X axis to this direction.

- <u>Transformations are applied in sequence</u>. However, don't need to do them all separately. Can compute the effect of a concatenated sequence as follows:

- Each point (X Y) is transformed to (X' Y') in the chip coordinate system by a 3×3 transformation matrix:

$$\boxed{[x' \; y' \; 1] = [x \; y \; 1] \, T}$$

- The transformation matrix T is simply the <u>product</u> of all the primitive transformation specified by the call: i.e. $T = T1 \; T2 \; T3$, etc.

- The primitive transformation matrizes ~~are~~ obtained by using the following templates:

(cont.)

Primitive Transformation Templates:

T a b $\qquad T_n = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & 1 \end{bmatrix}$

MX $\qquad T_n = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

MY $\qquad T_n = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

R a b $\qquad T_n = \begin{bmatrix} a/c & b/c & 0 \\ -b/c & a/c & 0 \\ 0 & 0 & 1 \end{bmatrix}$ where $c = \sqrt{a^2+b^2}$

Transformation of Direction Vectors: $(x \;\; y)$

We form the vector $[x \;\; y \;\; 0]$ and transform it by T into $[x' \; y' \; 0]$.

The new direction vector is then simply $(x' \;\; y')$.

Read Section on transformations carefully.

COMMENTS: Enclose in parentheses: (comment);

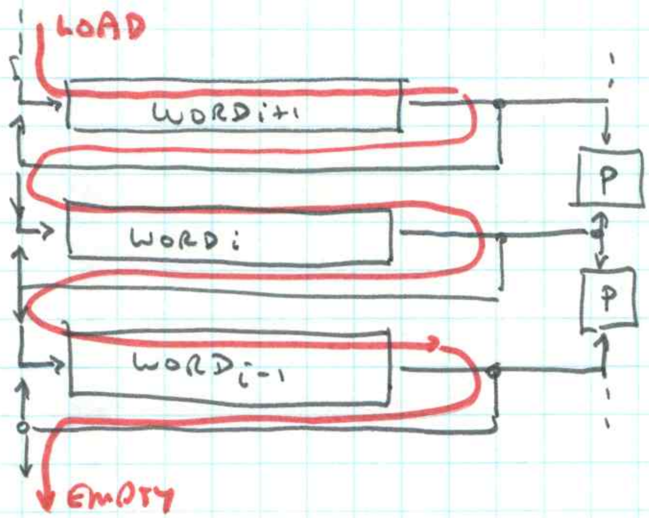
END COMMAND: E signals the end of the CIF file.

DISCUSS NEXT HW ASSIGNMENT:

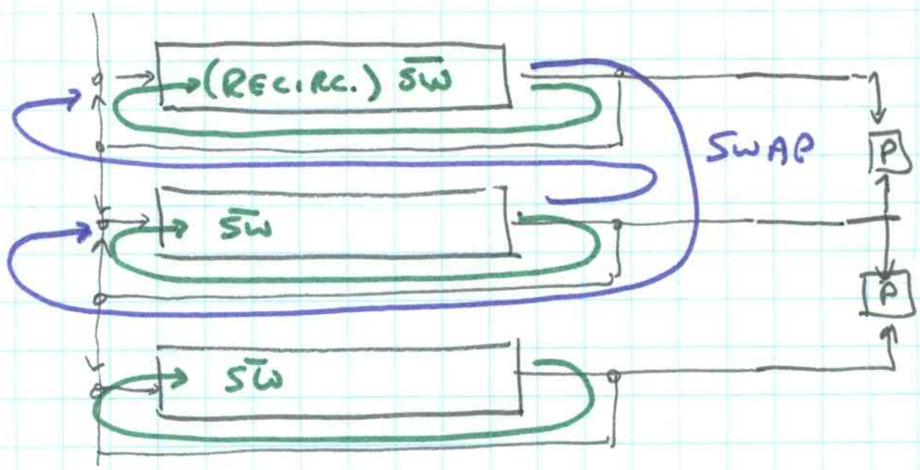
- Problem 13 is a CIF coding exercise. Keep a copy of your solution. You'll need it for a lab exercise.

- THE SORTER SUBSYSTEM. Problem 14

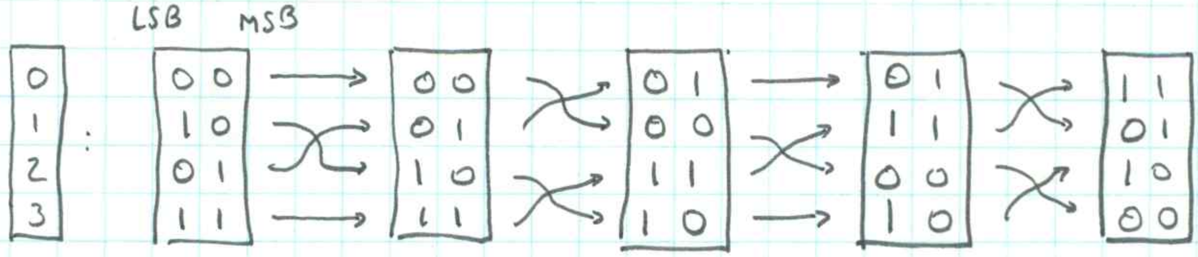
USE HW SET AS NOTES. DEVELOP STRUCTURE:



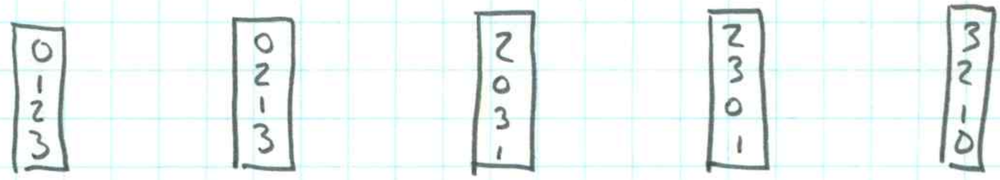
"A SMART MEMORY"



THE CONCEPT OF THE BUBBLE SORT: SUPPOSE LOADED:

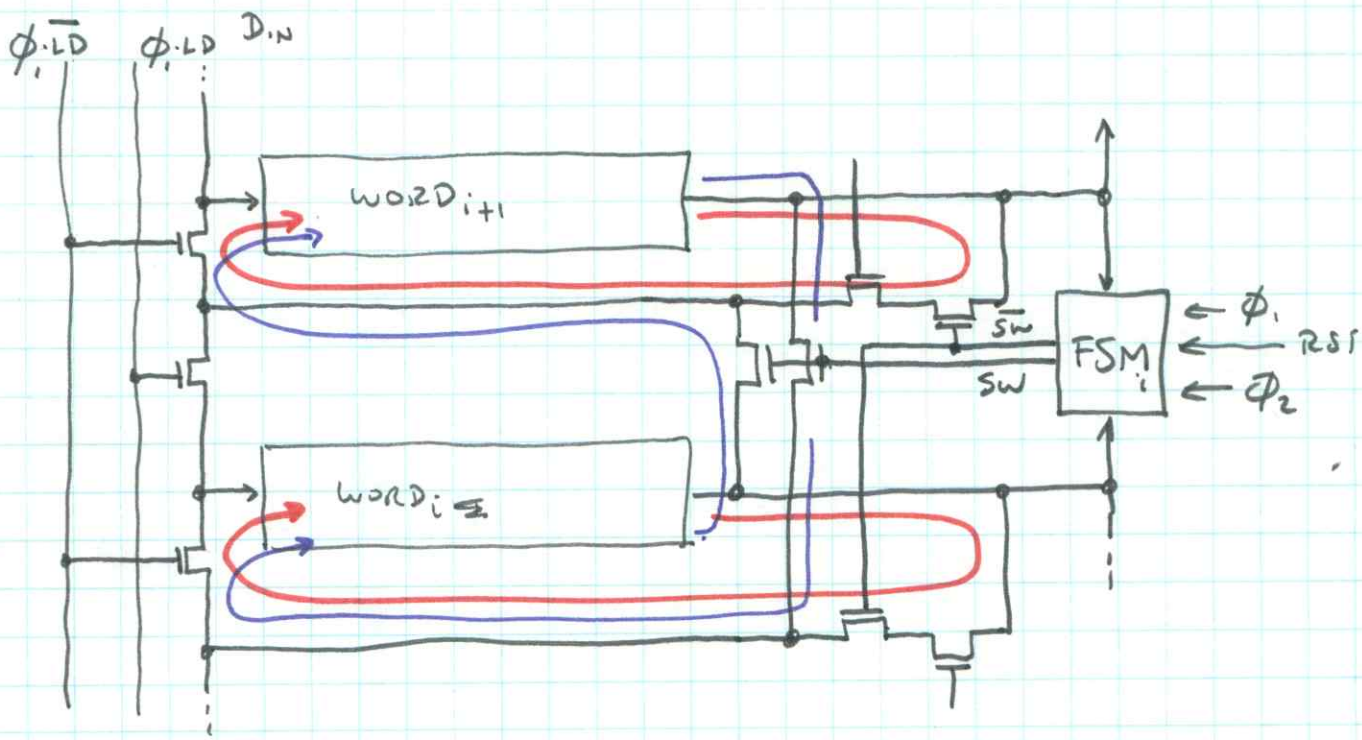


i.e:



ALGORITHM: SHIFT OUT AT RIGHT. START SWAPPING ADJ. PAIR OF ROWS IF DIGITS ARE DIFFERENT, AND LOWER = 1, UPPER = 0. COMPLETES IN N "CYCLES"

MORE DETAIL OF THE SWAPPING SWITCHES:



Think thru FSM, in some detail.

But all you should do for this assignment is the circuit design of FSM$_i$.

<u>Be sure to</u> carefully apply the 2-ϕ clocking methodology, doing the right things in the right places.

- <u>PROJECT ASSIGNMENT</u> #1

  (See Assignment sheet)

- <u>PROJECT LAB STATUS/QUESTIONS</u>

  (Jon Allen)