6.978 LECTURE # 13. 26 OCTOBER '78.

- TODAY: IMPLEMENTATION (CONT.); THE STORED PROGRAM COMPUTER

- HANDOUT: Project Assignment Schedule:
  26 OCT: Select; 9 NOV. Descr.; 21 NOV. Layout; 7 Dec. Report.
  MY Selection: 21 - 28 NOV.; Files Sent ~ 5 Dec.

POLL ON PROJECT SELECTION

- CONTINUE WITH A BIT MORE re Impl: THE PROCESS:
- BASIC NMOS Process desc. in text. SLIDE

- Process we'll use will have a # of steps to reach the first patterned oxide: these are described in GUIDEBOOK

- Basic idea is to get a p^+ region underneath the thick field oxide region: SLIDE

  This reduces parasitic C's and allows DIFF-DIFF to be 2λ
- WE WILL USE GNDed SUBSTRATE (EXPLAIN)

- INTERACTING WITH MASK/FAB FIRMS: (p.72-73)

  An index of firms is listed in the guidebook. Note, this was intended primarily for instructors & those actually running project chips (i.e. those with money).

  Costs: Making Masks for large proj. set ~ 6K. Depends primarily on the # of flashes (P.G. Time).
  Min run of Wafers: ~ 2K to fab ~ 10 to 20.

  [Note that maskmaking generally takes longer, more expens.]

  Specs: Sample SPEC sheet for proj. set given on p 68-69

WHEN WAFERS COME BACK:

There's a discussion of dicing, chip mounting, wire bonding on p 40-42.

TESTING: We're concerned with two different types:

(a) Electrical Testing: Could be done by probing wafers or by bonding up test patterns in packaged chip:

Special Patterns are included in the starting Frame to allow us to measure T and also the $R/\square$ of the various layers, to extract the MOSFET characteristics, and determine the quality of the process by measuring resistances of long chains of contacts, etc.

A number of such patterns are illustrated and described in p 58-62 by Rick Davies

Show Slides

(b) Functional Testing:

Assuming the process worked, and we've measured T, and assuming that projects are small and ~~thus~~ have high yield,

We now will perform functional tests on our projects. Procedures for this are discussed in p 43-50 by Peter Dobrowolski.

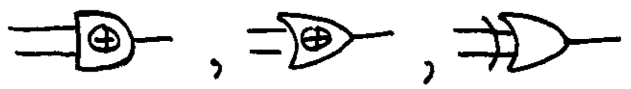
Show Slides and Discuss.

[COULD USE ANY OF THE CURRENTLY POPULAR MICROPROCESSOR DEVELOPMENT KITS. ESPECIALLY IF BUILT UP SOME OUTBOARD HARDWARE]

IF TIME: [DO AT END]

ANOTHER TOPIC WE'VE USED MOSTLY NAND, NOR, INVERT GATES.

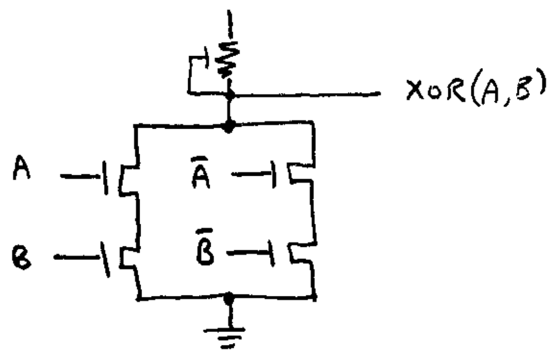
ANOTHER IMPORTANT GATE EASILY IMPL. IN MOS IS THE **XOR** GATE "EXCLUSIVE-OR"

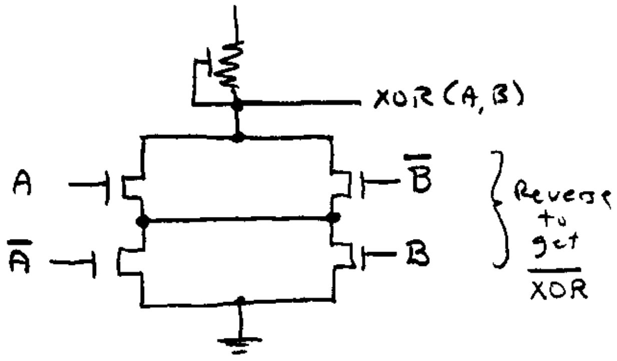


	(XOR)	($\overline{\text{XOR}}$)
A B	$A \oplus B$	$\overline{A \oplus B}$
0 0	0	1
0 1	1	0
1 0	1	0
1 1	0	1

$\boxed{\text{XOR} \Leftrightarrow \text{"MOD 2 SUM"}}$

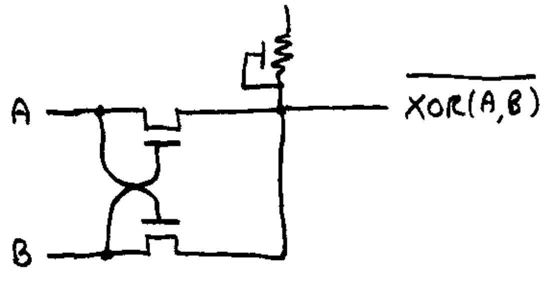
AS IN MULTICOMPARATOR:





ANOTHER VERY SIMPLE CIRCUIT, REQUIRING ONLY A & B (NOT COMPL)

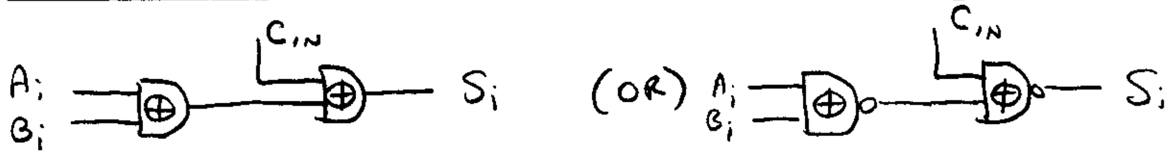
IS:



Describe fcn:
If $A = B = 1$, OUT = 1
$A = B = 0$, OUT = 1
If $A \neq B$, Then "ON" FET connects to zero input.

(NOTE: CANNOT BE FED BY CLOCKED PASS-TRANSISTORS)

EXAMPLE USE: IN ADDERS:



[NOTING: $(A \oplus B) = \bar{A} \oplus \bar{B}$]

THE STORED PROGRAM MACHINE:

- CH 5 & 6 Desc. an LSI comp. sys. des & impl. at Caltech.
- Provides many ex. of LSI circuit & subsystem design. Particularly of 2 chips of System: DATA PATH, CONT. SLIDES
- In later lectures I'll be describing this system in some more detail.
- It is architectually a general purp, stored program computer, using microprogrammed control.
- So that we can share a common terminology, and really understand the details of this system, I'd like today to review the basic ideas of the stored program computer --- the classical general purpose computer sometimes referred to as the von Newman machine (as a concept).
- What is a general purpose, St. prog. Computer? [First ½ CH 6]

  i.e., what are the key ideas, from which we synthesize and instantiate such machines.

[> By the way, that is an often embarrassing question to ask of "computer architects". Try it sometime!]

- Consider the OM Data Path: SLIDE

  It is claimed that this regular looking structure can perform a rich variety of operations on data stored within it.

  How can we visualize this — I tend to think of it using a piano analogy --- control lines as keys that are struck --- sequence of notes and chords can over time build up a very complex, abstract, piece of music.

  DATA FLOW vs CONTROL ... So the data chip is only at best ½ a computer. We must generate the control sequence somehow.

- **WHAT'S THE SIMPLEST FORM OF CTL. SEQUENCER:**

$\boxed{\text{FIG 1}}$ A Finite State Machine. Here, no inputs to FSM just runs thru a fixed seq of outputs indep of act. in Data Path. Could be used for impl. a digital filter - data in at left, fixed set of ops, data out at right.

- **Enhance this a bit:** $\boxed{\text{Fig 2}}$

To make control sequencing possibly be fcn of some event in data path, some LOG FCN of data called FLAGS are fed as inputs to the FSM.

Note: these diagrams "cover" large sets of possible FSM's, and overall structures. They are meant to clarify the KEY IDEAS which enhance functional capability as we move up thru a hierarchy to the SPM.

Any one of these can "cover" machines of great "complexity in detail." But the key ideas are few in number

Note: For simplicity, we'll not bother to show the clocks and PLA regs. Assume everything is Synchronous

- **Enhance Further:** $\boxed{\text{FIG. 3}}$

Fig 2 is quite general --- but we can make improvements such as adding or dedicating a register to hold the flags, loaded by output from the FSM.

Thus, the Flags can be used as control inputs for many cycles after their generation.

> A basic limitation however is the small amount of information provided by the few flags generated by the data path operations.

- THE STORED PROG. MACH. FIG 4a

A very powerful & general arrangement is shown in Fig 4a.

FSM sequencing not only controlled by last state & flags but also by data coming from a memory.
This provides a completely new dimension of possibilities:

- The fundamental idea is:

Rather than have th FSM perform one predefined operation (no matter how complex), we design it to perform any of a set of predefined operations

Called the "Machine instruction set".

The Mach. Inst. Set is carefully defined so that with the (CONT, DATAPATH, MEM) we can mechanize any of a number of different algorithms of interest to a number of diff users.

These algorithms are ~~implemented~~ encoded as programs composed of sequences of machine instructions loaded into the MEM.

Programs operate on data also located in the memory.

- The Machine fcns as follows

> One register in DP is selected to hold pointer into the program. Call this the PROGRAM COUNTER (PC).

> In Partic FSM state, called Fetch Nx Inst (FNI)

the FSM causes the memory to be read at location PC.

The resulting fetched inst. then places the FSM into first state of sequence to execute that instruction type.

- The FSM sequences thru # of states to execute that inst., at some point incrementing /calc. next PC value, & finally returning to the FNI state.

- Problem with Fig 4a: Most steps of INST need as FSM input some details of inst. or its encoded fields. We'll get more effective use of PLA if we dedicate a register to hold the instruction:

- **FIG 4b** In Fig 4b, an Inst. Reg. (IR) holds the inst. fetched during FNI from mem loc'n PC.

- NOW LETS FURTHER STRUCTURE THINGS BY NAMING THE STAGES of EXEC. OF INSTRUCTIONS:

Suppose have mach inst set incl. ALU ops, BR ops, MEM ops. & Data Path like OM. What must controller do to execute typical machine instructions?

Typically six Stages: **ON BOARD**

①	Fetch Next Inst	inst at PC fetched into IR
②	Decode Inst	"branch to starting state" for OP type
③	Fetch Operands	state seq. to fetch opnds ...
④	Perform Op	ex: Add Reg 1, Reg 2 ---
⑤	Store Result	to dest: Regs, Memory
⑥	Calc Next PC, go to FNI	Most incr., branches do more.

- <u>HOW DO WE DESIGN SUCH A CONTROLLER?</u>

  We construct a state diagram and impl. in PLA FSM. But, many states, 50 - 100$^+$, so can be very complex.

- <u>To structure this problem:</u> Fig 5

  Form state diagram as matrix. Vertically we stack up the processing stages FNI, Decode, etc.

  Then, we have one column for each instruction type, as many columns horizontally as instructions.

  [While many states, transitions are usually simple and local.]

- <u>Figure 5 shows INFORMAL EXAMPLES</u>

  of control sequencing to execute some different instruction types, to give a feeling for the details.

  (TALK THRU THESE A BIT)

- IN TEXT THERE ARE CORRESPONDING "TABULATED" SEQUENCES.

  THE # CYCLES TO DO THESE DEP. ON DATA PATH CAPABILITY, i.e. HOW MANY THINGS CAN GO ON IN PARALLEL.

  (SHOW THESE 3 SLIDES)

- YOU'LL FIND IT INSTRUCTIVE TO EXAMINE THESE SEQUENCES.

- <u>Note that they look like "programs" written in a very low level "machine language."</u>

- This anticipates the concept of "Micro-Programmed Control"

- SOMETIMES: > Entire Mach. Inst. Set not Definable ~~when~~ when machine is being designed.
  > Some users might want to run prog. for another machine type. Could sim., but inefficient.

- Thus, often wish we could have some sort of writeable controls ... so we could change them. Would help in debugging also.

- So, computer designers have often used MEMORY to hold control sequences, i.e. implemented the FSM with a memory FIG 6.

  This is inefficient: F flag bits, n next state lines, & i-bit instructions, then need $2^{(i+f+n)}$ words.
  But can usually easily reduce by inserting logic in feedback path:

- In figure Fig 7, some logic we'll call the "Micro program counter Path" is inserted in the path between the first memory and decoder.

  This type of control is generally referred to as MICRO PROGRAMMED control, whether writeable or Read only memories are used.

- Now the design of the control logic is reduced to encoding the sequences of control bit patterns to be stored in the MICRO-CODE memory.

- THE "MICRO-PROG CNTR PATH" similar to the data path:

  IT IS CONTROLLED BY OUTPUTS OF THE μ-CODE MEMORY.
  ITS PURPOSE: TO REDUCE THE AMOUNT OF μ-CODE MEMORY REQ'D.
  DOES THIS BY: ● MAPPING THE F + n bits of state into smaller #, then decoded to address μ-code memory.
  ● Reduces n by allowing complex fcns within μPC to be specified with just a few bits of control info.

- THE CONTROLLER CHIP DESC. IN CH6 IS THE μPC PATH PORTION OF A MICRO-PROG. CONTROLLER FOR OM2.
  (GO BACK TO OV. SLIDE)

- ALTERNATIVE WAY TO VIEW FIG 7:

  > EXAMINE LOOP FORMED BY μPC, Decoder, μ-CODE MEM

  > VIEW μ-code mem address as an "Inst ADDRESS" and wires from μ-code mem to μPC as "INSTRUCTION"

  > This alternative view is shown in FIG 8

- What we've really done is create another STORED PROGRAM machine within our STORED PROGRAM machine,

  So as to put as much fcn capability as possible in the path between machine inst and decoder of the state machine.

- ONE NOTE OF WARNING: VERY LITTLE IS EVER GOING ON AT ONE TIME WITHIN THE SPM --- THINK OF ALL THAT MEMORY, THE FETCH, EXEC, STORE,

  etc.