ECE 124 - W22
Digital Circuits and Systems
Full Course Notes

With Prof Otman Basir

My notes cover all the course material; at least, everything on the final was on here. Hope they help!

Josiah Plett

☆ $(x+y)(x+z) = x+yz$ ☆
☆ $x+xy = x$ ☆

ECE 124 Notes 1

① Intro

computer aided design

- We'll use CAD & FPGA's (Field Programmable Logic Devices)
- Everything's binary

0-minterm
1-maxterm

Combination Circuits — Don't care about time :)
Sequential Circuits — Have a sense of time :(

② Analog/Digital, Design, & Number types

- **Motherboard:** ties a computer's components together.

Analog: the input/output IS the number (Physical)
Digital: stuff is in BINARY (logical).

Binary (2), Octal (8), Hex (16)	
Bi → Oct:	groups of 3
Bi → Hex:	groups of 4

Complement: How much you need to add to get to the maximum single digit...

Within Radix R: FOR NEGATIVE NUMBERS
- R's complement: $r^n - N$ for $n \neq 0$ (0→0)

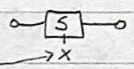
Another way: Each digit is $(r-1) - \langle digit \rangle$, then add 1.
(r-1)'s complement: Just don't add 1

Design Cycle
- Learn what the product requires.
- Specifications & initial design
- Is the design correct? NO
- Prototype implementation and testing
- Meets specifications? NO
  ↳ done! congrats!

Name	Operators	Gates
AND (logical product)	$f \cdot f$	
OR (logical sum)	$f + f$	
NOT	$\overline{f}$, f', $!f$, $\sim f$	
NAND		
NOR		

2.5 pg 53

③ Switches

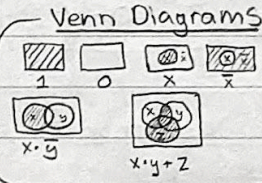


Groups of gates: "Logic Circuit"

$L(x) = x$ is a Logic Function

Precidence: NOT → AND → OR

Venn Diagrams



Boolean Algebra

Axioms:
- AND (1a): $0 \cdot 0 = 0$ (2a): $1 \cdot 1 = 1$ (3a): $0 \cdot 1 = 1 \cdot 0 = 0$
- OR (2b): $0+0=0$ (1b): $1+1=1$ (3b): $0+1=1+0=1$
- NOT (4a): $x=0 \Rightarrow \overline{x}=1$ (4b): $x=1 \Rightarrow \overline{x}=0$

1-Variable Theorems:
- (6a) $x = x \cdot 1$ = (7a) $x \cdot x$ = (6b) $x+0$ = (7b) $x+x$ = (9) $\overline{\overline{x}}$
- (8a) $0 = x \cdot \overline{x}$ = (5a) $x \cdot 0$ (5b) $1 = x+1$ = (8b) $x+\overline{x}$

2- or 3-Variable Theorems:
- (10) $x \cdot y = y \cdot x$, $x+y = y+x$ } Commutativity
- (11) $x(y \cdot z) = (x \cdot y) \cdot z$, $x+(y+z) = (x+y)+z$ } Associativity
- (12) $x \cdot (y+z) = x \cdot y + x \cdot z$, $x+(y \cdot z) = (x+y) \cdot (x+z)$ } Distributivity
- (13) $x + x \cdot y = x$, $x \cdot (x+y) = x$ } Absorbtion
- (14) $x \cdot y + x \cdot \overline{y} = x$, $(x+y) \cdot (x+\overline{y}) = x$ } Combination
- (15) $\overline{x \cdot y} = \overline{x} + \overline{y}$, $\overline{x+y} = \overline{x} \cdot \overline{y}$ } DeMorgan's Theorem
- (16) $x + \overline{x} \cdot y = x + y$, $x \cdot (\overline{x}+y) = x \cdot y$ } DeMorgan's Theorem
- (17) $x \cdot y + y \cdot z + \overline{x} \cdot z = x \cdot y + \overline{x} \cdot z$, $(x+y) \cdot (y+z) \cdot (\overline{x}+z) = (x+y) \cdot (\overline{x}+z)$ } Consensus

④ Synthesis

2.6 pg 60

- **Minterm:** each variable appears once: row 5 | $x_1\ x_2\ x_3$ = 1 0 1 → $m_5 = x_1 \overline{x_2} x_3$
  min = $\overline{max}$
- **Maxterm:** each variable appears once: 6 | 1 1 0 → $M_6 = \overline{x_1} + \overline{x_2} + x_3$

Sum-of-Products form

"AND" each minterm with its function value, and "OR" those together:

x	y	f(x, y)	minterm	
0	0	1	$m_0 = \overline{x}\,\overline{y}$	$M_0 = \overline{m_0}$
0	1	1	$m_1 = \overline{x}y$	$M_1 = \overline{m_1}$
1	0	0	$m_2 = x\overline{y}$	$M_2 = \overline{m_2}$
1	1	1	$m_3 = xy$	$M_3 = \overline{m_3}$

$f(x,y) = m_0 \cdot 1 + m_1 \cdot 1 + m_2 \cdot 0 + m_3 \cdot 1$
$= \overline{x}\,\overline{y} + \overline{x}y + xy$

Product-of-Sums form

"AND" each maxterm with NOT its function value, and "AND" those together:

The same function as left becomes:

$f(x,y) = M_0 \cdot \overline{1} \cdot M_1 \cdot \overline{1} \cdot M_2 \cdot \overline{0} \cdot M_3 \cdot \overline{1}$
$= M_2 = (\overline{x} + y)$

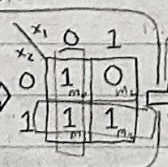
⑤ Karnaugh Maps

↳ <=5 inputs

Canonical & Standard Forms:
- Sum of Products
- Product of sums

KMap: alternative to truth table

x_1	x_2	f	
0	0	1	m_0
0	1	1	m_1
1	0	0	m_2
1	1	1	m_3

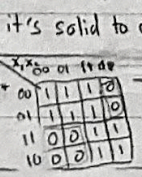


Minimum SOP: $f = \overline{x_1} + x_2$

Minimizing SOP Representations
1. Select rectangles (product terms) with as many 1's as you can (2^n, $n \in \mathbb{Z}$)
2. Cover all the 1's; it's solid to cover a 1 multiple times

Minimizing POS:
- surround 0's
- AND the sum terms

eg.



⇒ $f = (\overline{x_1} + x_2 + x_3)(\overline{x_1} + \overline{x_3})$

ECE 124 Notes 2

$X_n \rightarrow$ uncomplemented (positive) literal

$X: D(n) \rightarrow$ don't-cares

⑥ Function Minimization

$f = \bar{x}w + yw + \bar{z}yx$ ⇨ COST: 1 OR + 3 AND + 10 GATE INPUTS = 14 COST

★ We can always form a cover with only prime implicants ★

Terminology

- Completely Specified: every input has a defined (specific) output.
- Incompletely Specified: opposite
- Sets of minterms:
  - on-set: output is 1
  - off-set: output is 0
  - dc-set: output is X (don't care)

XOR & XNOR

$f_{XOR} = X_1 \oplus X_2$
$= \bar{X}_1 X_2 + X_1 \bar{X}_2$

	00	01	10	11
0	1	0	1	0
1	0	1	0	1

$f_{XNOR} = \overline{X_1 \oplus X_2}$
$= \bar{X}_1 \cdot \bar{X}_2 + X_1 \cdot X_2$

	00	01	10	11
0	0	1	0	1
1	1	0	1	0

TRY TO USE DURING MINIMIZATION!

⑦ Implementations

SOP: - plane of NOT gates
- plane of AND gates
- a single big OR gate

POS: - NOT gates
- OR gates
- an AND gate

- Product term: any AND of literals

More Terminology

- Implicant: product term with outputs always 1 (rectangle in k-map)
- Prime Implicant: removing a literal would result in a (bigger) non-implicant
- Cover: collection of implicants that cover every value of 1
- Essential Prime Implicant: if its the only prime implicant that can include any specific minterm

STANDARD SOP & POS

no requirement for terms to be min or max; instead, only any P or S terms!

NAND	NOR	XOR	XNOR
$\overline{xyz}$	$\overline{x+y+z}$	$x \oplus y$	$\overline{x \oplus y}$

$X_1 \oplus X_2 \oplus X_3 \oplus X_4 \rightarrow$ "odd operation" $(X_1+X_2+X_3+X_4)\%2$

BUFFER (boosting signal strength)

Associative	NOT Associative
AND, OR, XOR	NAND, NOR, XNOR

SOP with NAND

NOT

POS with NOR

NOT

⑧ CAD(D) & Physics

Computer Aided Design (& Drafting)

VHDL
VHSIC Hardware Description Language

VHSIC
Very High Speed Integrated Circuit

★ Variable names are CASE INsensitive

★ Concurrency: everything in parallel

VHDL Circuit Description

- Entity Declaration: inputs & outputs
  - ↳ describes "outside world" interface
- Architecture Definition
  - ↳ describes the circuit's implementation

DATA OBJECTS
- signals
- constants
- variables

} See Slides For Syntax

IN | OUT | INOUT | BUFFER

TRANSISTORS as SWITCHES

X="low" X="high"

(Type of transistor) MOSFET
Metal Oxide Semiconductor Field-Effect Transistor
(n-channel or p-channel)

Functional Decomposition

eg 2

$g = X_1 + X_2 + X_5$

⑨ Tabular Method

Functional Decomposition

If $f(x_1,\ldots,x_4) = \bar{X}_1X_2X_3 + X_1\bar{X}_2X_3 + X_1X_2X_4 + \bar{X}_1\bar{X}_2X_4$

Then $f = (\bar{X}_1X_2 + X_1\bar{X}_2)X_3 + (X_1X_2 + \bar{X}_1\bar{X}_2)X_4$

Let $g(x_1,x_2) = \bar{X}_1X_2 + X_1\bar{X}_2$ so $g'(x_1,x_2) = X_1X_2 + \bar{X}_1\bar{X}_2$

$f = gX_3 + g'X_4$

Quine-McCulskey Method

- eliminate literals with absorbtion
- Minimum # of prime implicants (chart)

CAD

TABULAR METHOD

Minterm List (list 1)

- List all applicable minterms (1 or X)
- Group them based on # of 1's in binary representation

Prime Implications (list 2)

- Combine implicants who differ by 1 with preceding.
- Check off terms of list 1 that appear in list 2

List 3

- Same like list 2, but make sure X's align.

Essential PIs

- PI: not checked off ☺
- EPI: only ☑ in a column
- FINAL ANSWER: Sum the EPIs! (min cost!)

EXAMPLE

$f = \Sigma m(0, 4, 8, 10, 11, 12, 13, 15)$

# of 1's	m_i	4-literal Implicants	
0	0	0000	✓
1	4	0100	✓
	8	1000	✓
2	10	1010	✓
	12	1100	✓
3	11	1011	✓
	13	1101	✓
4	15	1111	✓

combination	3L - Implicants	
0, 4	0X00	✓
0, 8	X000	✓
8, 10	10X0	✓
4, 12	X100	✓
8, 12	1X00	✓
10, 11	101X	
12, 13	110X	
11, 15	1X11	
13, 15	11X1	

combo	2L
0,4,8,12	XX00

We would repeat 2 times if we could

PIs	0	4	8	10	11	12	13	15
$P_1 = 10X0$			✓	✓				
$P_2 = 101X$				✓	✓			
$P_3 = 110X$						✓	✓	
$P_4 = 1X11$					✓			✓
$P_5 = 11X1$							✓	✓
$P_6 = XX00$	✓	✓	✓			✓		

ECE 124 Notes 3

$\oplus \leftrightarrow$ XOR subtraction: addition of two's complement form

(10) Petrick's Method (branch & bound method)

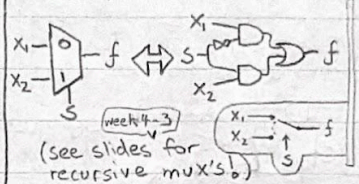
1. Form a function that's true when all columns are covered
   - (a) Make a SUM of PI's that cover the minterms
   - (b) AND all those sums together
2. Reduce that function algebraically, to a min SOP.
   - (a) $(x+y)(x+z) = x+yz$ & $x+xy = x$
   - (b) Be aware: each product term represents a solution!
3. Determine minimum solutions (fewest # of PI's & literals)
4. Choose the term(s) with min # of literals; write out the corresponding minimum sum of PI's!

ROM → Read-only Memory (changable)
PROM → Programmable Memory

m inputs, n outputs } $2^m \times n$ ROM ("by")

(11) Multi-Level Combinational Circuits (Modules)

Multiplexer (MUX)



(see slides for recursive mux's!)

Function Synthesis (w/MUX)

Shannon's Expansion

$f(w_1, w_2 \cdots w_n) = w_1' f(0, w_2 \cdots w_n) + w_1 f(1, w_2 \cdots w_n)$

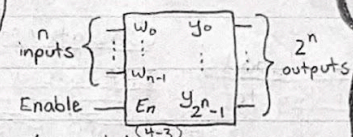
$f = w_1' \cdot f_{w_1'} + w_1 \cdot f_{w_1}$

- "cofactors" of f, respect to w_1

eg $f_{w_1} = w_2' f_{w_1 w_2'} + w_2 f_{w_1 w_2}$ ← 4 INPUT :)

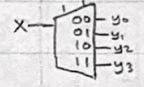
eg $f_{w_1' w_2} = f(0, 1, w_3 \cdots)$

Decoder (binary decoder)



(see slides for recursive decoders!)

De-Multiplexer

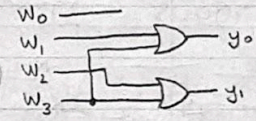


S_0	S_1	y_0	y_1	y_2	y_3
0	0	X	0	0	0
0	1	0	X	0	0
1	0	0	0	X	0
1	1	0	0	0	X

Encoder (binary decoder)

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

priority encoder has (X)



- generates error when more than 1 input is high
- "priority" encoder doesn't!

(12) Combinatorial Circuits

Binary half-adder

x	y	sum	cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$sum = \bar{x}y + \bar{y}x$
$cout = xy$

Binary full-adder

$sum = x \oplus y \oplus c_{in}$
$cout = xy + c_{in}(x \oplus y)$

Ripple adder for n-bits

Linking many full-adders!
Performance: $\boxed{2n+1}$
(if made of half-adders)

Carry look-ahead adder

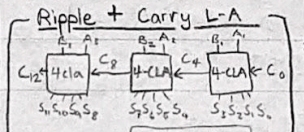
- p(i): propagate; sum
- g(i): generate; cout

use functional substitution

$c_1 = g_0 + c_0 p_0$
$c_2 = g_1 + c_1 p_1 = g_1 + p_1 g_0 + p_1 p_0 c_0$
$c_3 = g_2 + c_2 p_2 = g_1 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$
$c_4 = \ldots$

Performance: $\boxed{3 \text{ units}}$

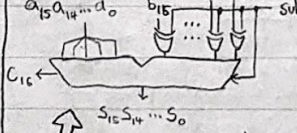
Ripple + Carry L-A



3+2+2 = $\boxed{7 \text{ units}}$

Binary subtracter

subtraction is same as addition of two's complement



Can add or subtract!

Magnitude Comparators

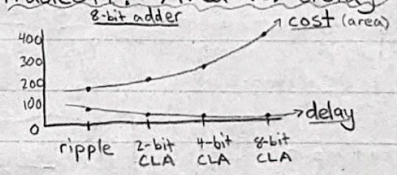
equality of bits: $e_i = a_i b_i + \bar{b}_i \bar{a}_i$
→ overall equality: $(A=B) = e_n e_{n-1} \cdots e_0$
inequality of bits: ★conceptual algorithm★
also, remember: $(A<B) = \overline{(A=B)+(A>B)}$

Performance

1 logic gate ≡ 1 "unit" of delay

- Circuit's performance is determined by the longest combinatorial path.

≡ Tradeoff: Area vs. delay

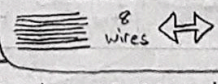


Overflow Detection → if you need n+1 bits

≡ Unsigned numbers: if there is a carry-out from the MSB (Most Significant Bit), then there's overflow.

Busses

A bundle of wires that carries data someplace! "16-bit bus"

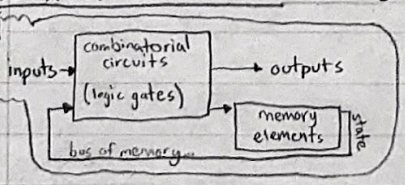


in this course we'll never have >1 sources driving at the same time; reuse multiplexers as control!

(13) Sequential Circuits (& recognizers)

"synchronous" ≡ "clocked sequential"

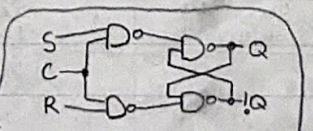
Combinational vs sequential



Synchronous vs Asynchronous
- Synchronous → circuit behaviour is determined from knowledge of values at discrete instances in time
- Asynchronous → circuit behaviour is determined by signals at any instance in time, and the order the inputs change.

CLOCK SIGNAL

used to control behaviour of a circuit at discrete points in time.



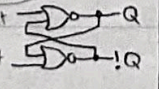
Latches - type of storage element (helps: asynchronous)

- Level sensitive: operate at 0 or 1, not 0→1 or 1→0 "rising edge"

NOR SR Latch:

S	R	Q	$\bar{Q}$
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0

(1, 0) means Q = 1
(0, 0) is storage!
(0 1) means Q = 0
(1, 1) is undesirable!



NAND SR Latch:
same as NOR but:
(S, R) (1, 0) → Q = 0
(0, 1) → Q = 1
(1, 1) → storage
(0, 0) → bad!

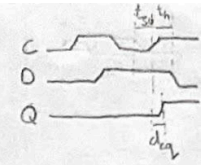
Gated latch

control C	S	R	next Q value
0	X	X	no change
1	0	0	no change
1	0	1	0
1	1	0	1
1	1	1	undefined

to control whether it can function!

ECE 124 Notes 4

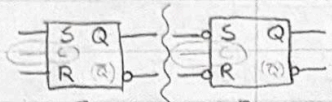
Definition: Propagation Delays:
- time held stable before edge → Setup time: t_{su}
- time held stable after edge → Hold time: t_h
- time till output is stable → Clock-to-output time: t_{cq}



Definition: Synchronous: same clock
Asynchronous: not same with clock

(13.2) Latches (ct'd)

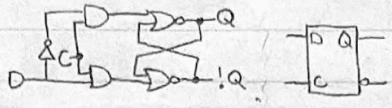
Schematic Symbols



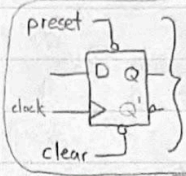
active-high inputs (S=1 → set) Q=1

active-low inputs (S=0 → set) Q=1

D-latch captures the logic level that's present on the Data line when the Control is high;
C=1 → Q=D
C=0 → Q = it's previous value

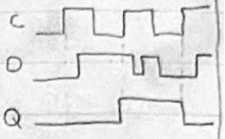


(14) Flip Flops

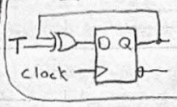


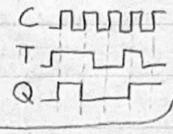
D-Flip-Flop (DFF)

- Updates on rising edge
- "Clear" updates instantly
- negated clock? falling edge!



Toggle FF

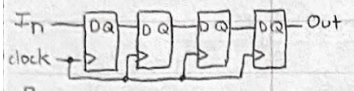




JK-FF

T-FF if J=K=1
D-FF otherwise
(J=S) ★
(K=R) ★

Multi-Cell Storage Components



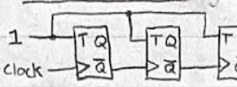
↳ 4-bit shift register
- data point goes right by 1 on every rising edge

Parallel Load Shift Register

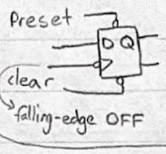
- series & parallel input (& output)

Counters

TFF-based sync? Counter



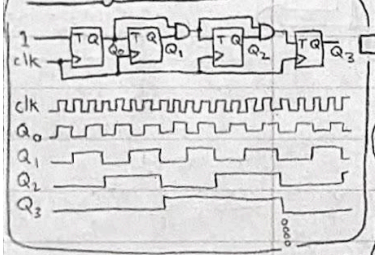
DFF with clear/preset



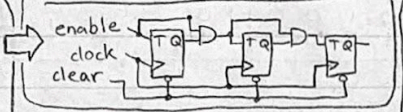
- clear/preset are asynchronous
- clear/preset are active-low
- to make "clear" synchronous, do: clear, data → D

(15) Synchronous Counters, Analysis, & Design

4-bit Sync Counter

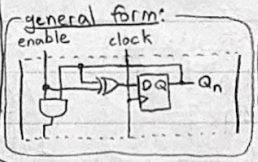


Now with Enable & Clear



DFF-based counters

general form:



Parallel-load capability counter

see lecture notes week-6-3 pg 6

Modulo-N counter

see lecture notes week-6-3 pg 7

Johnson Counter

AKA "inverse feedback counter" "twisted ring counter"

- Modified ring counter to produce the following:

000
100
110
111
011
001
000
100
...

See lecture notes for schematic (week-6-3 pg 8)

Ring counter

000
100
010
001
000
...

implemented similar to a Johnson counter

BCD counter

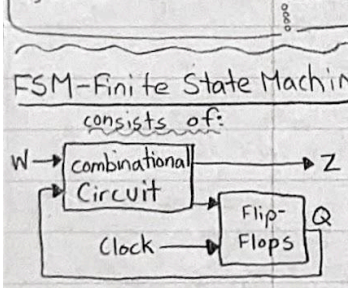
counts from zero to nine!

Shortest clock that works?

find propagation delay:
↳ Find longest "trickle" time...
↳ and add t_{su} + t_{cq}

FSM-Finite State Machine

consists of:



W: circuit input
Z: circuit output
Q: current state

sequential circuits

Moore is Less

Moore Machine

FSM except Z depends only on Q (sync w/clock)
- Z changes synchronously

Mealy Machine

FSM except Z depends on Q and W (sync w/input)
- Z changes asynchronously

(16) Sequential Circuits

← Types of Finite State Machines

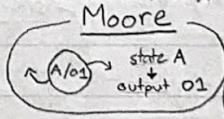
State Diagram

- Possible states: state bubbles
- Possible transitions: directed edges
- Circuit outputs: labelled on those

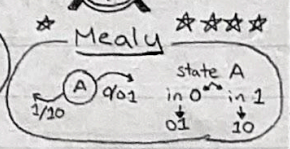
Edge-in State Diagram

Indicates the transitions based on state when the rising edge returns

★★★★★★★★★ Moore



★ ★★★ Mealy



Any sequential circuit can be described with a state diagram that's either Mealy or Moore.

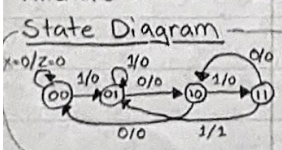
Purpose of Reset:

We want to know what the system's at when it starts/restarts

(17) State/Transition Tables

Alternate form to state diagrams:

State Diagram

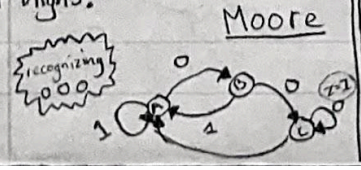


State Table

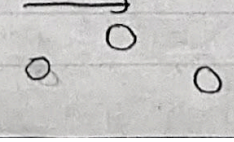
curr state Q_0Q_1	next state Q_0Q_1 X=0	next state Q_0Q_1 X=1	output Z X=0	output Z X=1
00	00	01	0	0
01	10	01	1	1
10	00	11	0	0
10	10	01	0	1

Master vs Slave: slave gives output
↓ rising edge ↓ falling edge

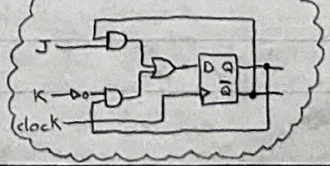
XOR: odd number of highs!



Moore | Mealy



JK-FF Digram



ECE 124 Notes 5

(18) Synchronous Circuit Design

DESIGN

If DFF, these are equal!

Use cases (word problem) → State Diagram (Mealy: State + In; Moore: State) → State Table (current state, inputs, next state, Flip-flop values) → FF choice (Each FF is one bit; DFF => good choice!; JK => complex choice!; Excitation Tables) → FF input equations → Output equations → Circuit Schematic!

Q_2 \ Q_0Q_1	00	01	10	11
0	0	0	1	1
1	1	X	X	0

# of Karnaugh maps == # of inputs to flip-flop

(19 (17!)) Circuit Analysis

ANALYSIS

THAT BUT BACKWARDS

Using sentences to describe to left

Procedure:

- Identify the FFs used to hold the current state
- Identify the outputs of the circuit
- Write the logic equations for circuit outputs & FF inputs ("next state equations") — "excitation tables" ← small + simple
- Use the "next state equations" to derive a state table w/ next state and outputs
- (optional) Make a state diagram from the state table.

(19) State Reduction

The State Assignment problem... ← binary value of state

- make the output the curr state; perhaps adding "extra bits" ← encoding for outputs
- more than minimum FFs
- potentially lots of unused states

One-Hot Encoding

of FFs = # of States

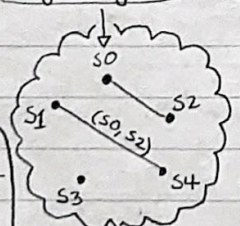
1 output is on for each state.

Reduction

States are equivalent if:

- they give identical outputs
- they give same/equivalent next states

methods for reduction: Implication charts; Merger Diagrams; Partitioning



see "state reduction" slides

curr	next A=0	next A=1	out A=0	out A=1
S0	S3	S2	1	1
S1	S0	S4	0	0
S2	S3	S0	1	1
S3	S1	S3	0	0
S4	S2	S1	0	0

	S0	S1	S2	S3
S1				
S2				
S3				
S4				

① "X" if different output values
② "√" if definitely equivalent
③ "(S0, S1)" if equiv. under the case that those 2 are equivalent
④ check!

↳ under which conditions are any of these equivalent?

(20) Algorithm State Machines ASM

- Take advantage of registers & counters.
- use decoders & muxes for logic.

☆ Data Paths (X)
↳ information reception + delivery

☆ Control Paths (D, T, JK...)
↳ logic between things

- Modular SyDe
- RTL → between registers
  Register transfer level
- Status signals
  ↳ for control paths
- ASM charts
  ↳ flow of computation

R2 ← R1 Transfer from register 1 to 2

if (T1=1) then R2 ← R1 (conditional)

R4 ← R4 Shift right

State Box

- operation of sequential circuit
- indicates FSM states

name → binary code → register operation or output

Decision Box

- impact of input on the control system
- implemented using a magnitude comparator (MC)

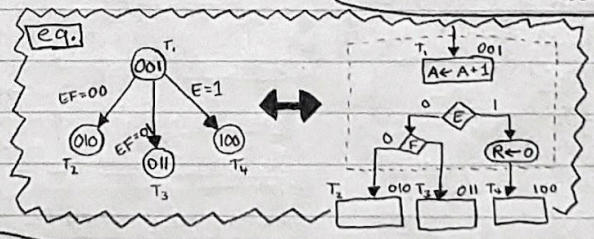
condition: 0 → exit path; 1 → exit path

Conditional Box

- indicates assignments following a decision box; data transfer usually

exit path from decision box → Register operation or output

eg.



(21) ASM 2

Problem → ASM chart (flowchart) → state diagram → datapath: state ≡ ckt ≡ data; control path: inputs (prev. state) ≡ ckt ≡ state

Problem → state diagram/state table

Hilroy

"Less is Moore" — depends on less; ie. not on prev. state or input

"NOR is Normal" — NOR latches are the normal ones

ECE 124 Notes 6

transition table ⟷ flow table

$y_1 y_2$	$Y_1 Y_2$, Z
00	01, 0
01	10, 0
10	10, 0
11	01, 0

$y_1 y_2$	$Y_1 Y_2$, Z
a	b, 0
b	c, 0
c	c, 1
d	b, 0

would be multiple based on X

"Loop" through logic gates

(22) Asynchronous Sequential Circuits

- Feeback loops that resemble latches

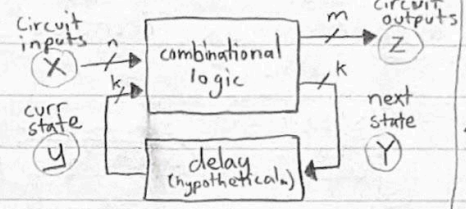
NO FF
NO CLK
YES MEMORY

Asynchronous Analysis

Identify: has latches? Async! (or equivalent...)

Generally: stable state is unpredictable

(23) Async Circuits



- Stability: stable means steady state (unchanging).
- Fundamental Mode Operation:
  - only one input can change at a time
  - it changes only if circuit is stable
- circle stable states in your transition tables:

curr y_1	next $x=0$ Y_1	next $x=1$ Y_1	output
0	(0)	1	whatever lol
1	1	(0)	

- Primitive flow table
  - one stable state per row

DEFINITIONS

(24) Races, Cycles, State Assignment

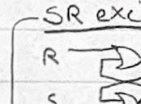
eg

$y_1 y_2$	x=0	x=1
00	00	11
01	00	11
11	00	11
10	00	10

This is a critical race. (different stable states)

00→11 RACE
y_1 changes first
00→10 (stable)
y_2 changes first
00→01→11 (stable)
change simultaneously
00→11 (stable

SR excitation table



S R	y Y
0 X	0 0
1 0	0 1
0 1	1 0
X 0	1 1

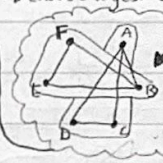
assuming never SR=11

(25) State Reduction

A and B are compatible if, for each input combo:
1. They have same outputs AND
2. They have compatible next states

} where specified ← [X]

Implication Chart → Merger Diagram
- choose largest "cliques" (loops)



Assuming (A, D) doesn't require (A, B), for example

(26) State Assignment (Race Free)

Shared Row SA | MultiRow SA | One-hot SA

PROCEDURE

Analysis w/ Latches

1. Label latch output (Y_j) & feedback path (y_j)
2. Logic equations for S_j and R_j
3. [X] Must have, to work correctly:
   $S \cdot R = 0$ ← NOR ("normal") | NAND → $\bar{S} + \bar{R} = 0$
4. Use known behaviour for outputs:
   $Y = S + \bar{R} \cdot y$ ← NOR | NAND → $Y = \bar{S} + R \cdot y$
   "logic equations"
5. Make transition table
6. (perhaps) Make flow table

Design

1. Make Primitive Flow Table
2. Reduce flow table
3. Do State Assignment (no race conditions)
4. Next & Output logic equations
5. Draw circuit

(27) RAM & ROM

EPROMS, EEPROMS } store data and Functions and Design Building blocks

(ROM → read only)

PAL: Programmable Array Logic
- Products followed by sums

Erasable Electrically Programmable Read-Only Memory
- "programming" means blowing fuses to give some bits = 0
  - only do it once!
- erasable w/ UV light
- electrically erasable

important

ROM
- Permanent
- (k) addresses
- (n) bits
- ($2^k \cdot n$) info

k inputs → $2^k \cdot n$ ROM → n outputs

RAM ↔ (soft) Random access memory

READ and WRITE

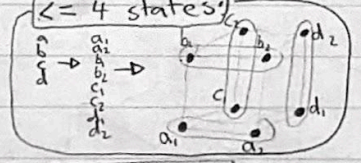
PLA — Programable Logic array
implements SOP — input buffers & inverters → AND → OR

PAL — Programmable array logic
PLA but the OR plane is fixed

CPLD → very big SPLDs connected by Programmable Routing Fabric

Hazard: inside Glitch: output

<= 4 states!



HAZARDS

static & dynamic

Static-0 hazard: 0→0, but instead 0→1→0

Static-1 hazard: opposite of ↑

Dynamic hazard: ⎍ or ⎍

multi-level circuits

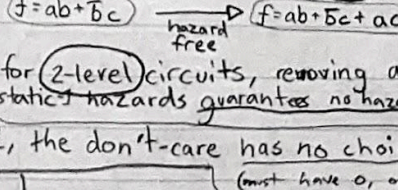
SR → tolerates momentary 00

S'R' → tolerates momentary 11

Generate function: always SET or RESET

output glitches: if both are 0 or both 1, the don't-care has no choice (must have 0, or 1)

If adjacent minterms aren't covered, a HAZARD EXISTS



$f = ab + \bar{b}c$ → hazard free → $f = ab + \bar{b}c + ac$

for 2-level circuits, removing all static-1 hazards guarantees no hazards

SPLD — Simple programmable logic device.
- add some FFs after a PAL or PLA

gotta program the multiplexer select lines

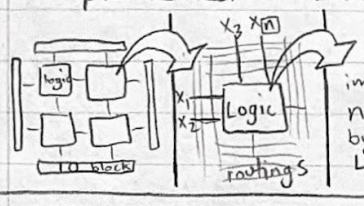
ECE 124 Notes 7

practice

- convert to JK FF (from state diagram)
- draw circuit out of state table
- Decoder, and memory stuff
- multiplexer system (w/ enable)
- Counter system (w/ load)
- Full Adder system
- merger diagrams
- partitioning (SA)
- critical races (RF)
- recognize a pattern
- ASM chart

(28) FPGAs

Field Programmable Gate Arrays → Implemented with an LUT (look up table)



implements any n-input function by configuring LUT bits

asymmetric clock cycle == non-uniform duty cycle

inverted clocks make it harder to function

(29) Timing Analysis

Cycle Times

FF outputs take time to arrive back as inputs

To reach in time, our maximum frequency for 2 FFs is:

$$T_{cycle} \geq T_{clock\ out} + T_{data\ logic\ between\ (FF2)} + T_{su} + \underbrace{(T_{clk1} - T_{clk2})}_{\text{clock skew } (+ \text{ or } -)}$$

$T_{cycle} = \frac{1}{f_{cycle}}$

without violating the set-up time

Pins of a Chip

su — cf → time when FF CLK INPUT active
cc → CLK PIN

hold — df → data FF time
dc → data PIN time
tsu → su w.r.t. FF CLK (for FF input)
th → FF hold time
tsetup → PIN su
thold → PIN hold

its a garbage meme but its kinda on point. you got this :)