INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
I.C.T.P., P.O. BOX 586, 34100 TRIESTE, ITALY, CABLE: CENTRATOM TRIESTE

UNITED NATIONS INDUSTRIAL DEVELOPMENT ORGANIZATION

INTERNATIONAL CENTRE FOR SCIENCE AND HIGH TECHNOLOGY

SMR/542 - 9

ICTP-INFN
SECOND COURSE ON BASIC VLSI DESIGN TECHNIQUES
18 February - 15 March 1991

Digital Design
"Algorithmic State Machine Design"

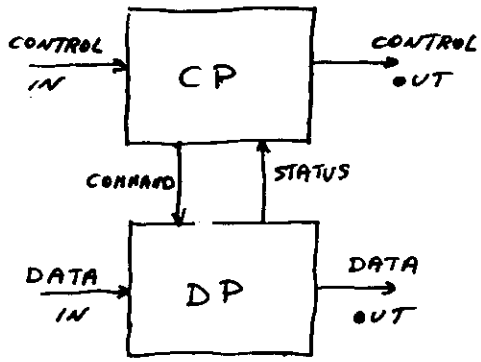
Paolo FARABOSCHI
Dipartimento Ing. Biofisica ed Elettronica
Università di Genova
Via all'Opera Pia 11/A
Genova 16145
Italy

The microarchitecture of any computing unit can be subdivided into 2 parts:

- The control part
- The Data Path.

- The Data Path is composed of hardware structures that perform data transformation and data storage.

- The Control Part performs the control of the flow of the algorithm the computing machine has to accomplish.



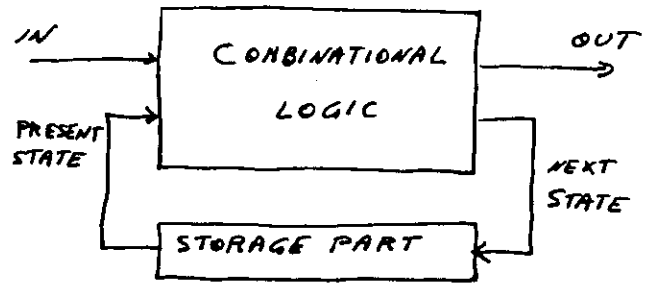
The control part, on the basis of external controls and of the data path status, and of the present machine state, decides what commands have to be sent to the data path in order to perform a given algorithm_

The state of the machine implies the knowledge of present and past conditions to determine future behaviour_

At a given instant the control part performs a logic function which, by using appropriate input information, moves at the proper time to the next state_

A control part is then composed of two parts:

- COMBINATIONAL PART
  That performs the logic function that moves from the current state to the next state and determines the proper output signals for the data path_
- STORAGE PART
  That loads the state of the control part_



THE CONTROL PART CAN BE SYNTHESISED BY USING DIFFENT METHODS_ ONE OF THEM IS THE

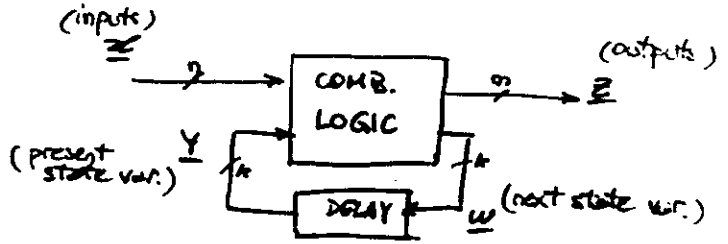
ALGORITHMIC STATE MACHINES METHOD_

SEQUENTIAL LOGIC

- Sequential circuits differ from combinatorial logic in that:

  OUTPUTS ARE DEPENDENT ON PRESENT AND PAST VALUES OF INPUTS

- This is possible because the circuit 'remembers' the sequence of input states, by setting itself in stable configurations individuated by specific values of input and state variables.



- SYNCHRONOUS CIRCUITS
  - state variables change only at certain pre-defined times dictated by an external clock signal
  - The clock is sufficiently long to enable all transients to die down
  - Inputs can be synchronous or asynchronous

ASYNCHRONOUS CIRCUITS

- There is no reference signal
- Time interval are specified by input intervals

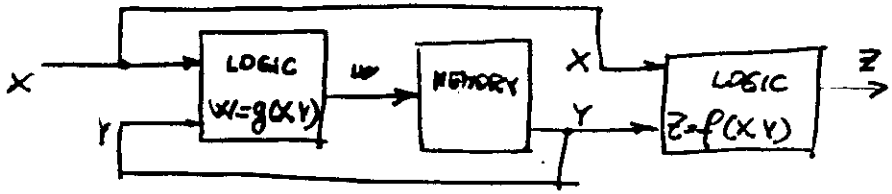
THE ALGORITHMIC STATE MACHINE METHOD

- The description of the behavior of any processing system is provided by an ALGORITHM

  ALGORITHM = sequence of events required to produce a set of actions from a given set of data

  - FINITENESS: must terminate in a finite number of steps
  - DEFINITENESS: each step must be precisely defined and unambiguous

- General State Machine

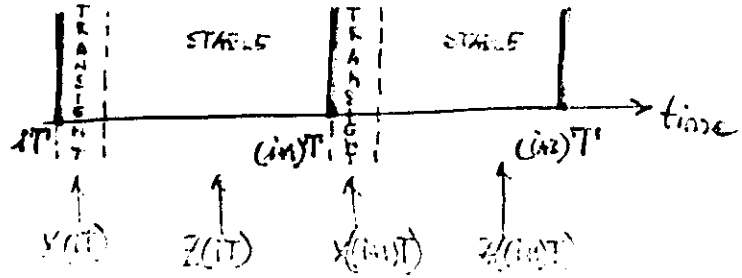


$Y(t+\Delta t) = W(t)$

$W(t) = g[X(t), Y(t)]$

$Z(t) = f[X(t), Y(t)]$

- Time relationship for sync. machines



The design of an ASM implies the determination of the two equations:

$$X_N \leftarrow f(X_P, IN)$$

where X_N = the next state
X_P = the present state
IN = the input signals

$$OUT \leftarrow g(X_P, IN)$$

where OUT = output signals.

The two equations are derived from the algorithm that describes the behaviour of the machine.

An algorithm can be expressed through two types of statements:

* Transformation statements:
  They specify operations performed by the data path, and represent commands issued by the control part.
  A transformation can be determined by:
  - the state of the machine,
  - both the state and inputs

* Control statements:
  They specify how to control the flow of the algorithm on the basis of the present state and of the input signals.

Example :

2 bit up-down synchronous counter
with set and reset.

start:

Sϕ: if reset then out = ϕ, goto Sϕ
else if set then out = 11, goto S3
else if up then out = 1, goto S1
else out = 11, goto S3

S1: if reset then out = ϕ, goto Sϕ
else if set then out = 1, goto S3
else if up then out = ϕ1, goto S2
else out = ϕ, goto Sϕ.

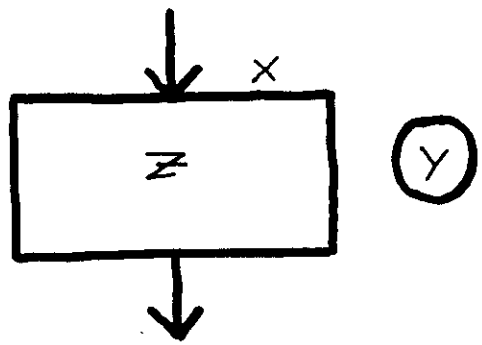
S2:
⋮

S3: - - - -

if ----- decision statement

out = 1 transformation statement.

STATE SYMBOL



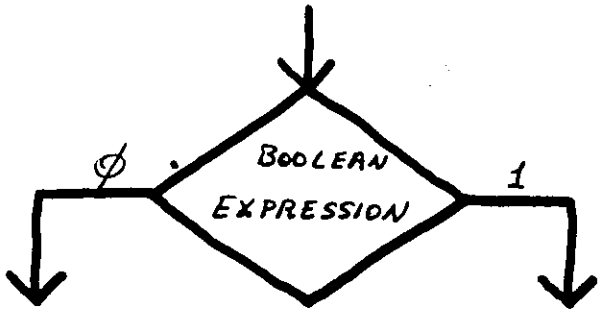
where:

X : is the binary code assigned to the state

Y : is the name of the state

Z : is the list of the active outputs

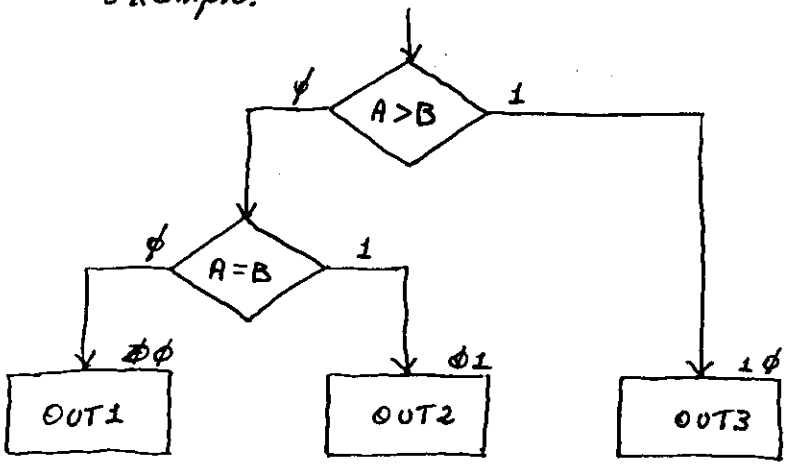
DECISION SYMBOL



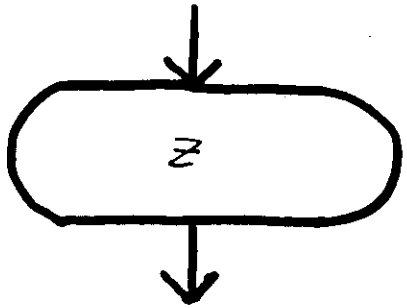
It describes the inputs of the ASM, and has two outputs.

The output is selected according to the value of the boolean expression.

Example:



CONDITIONAL OUTPUT SYMBOL



where :

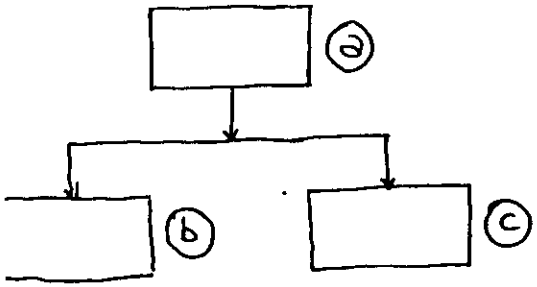
z is the list of the active outputs.

It has been introduced to represent the the direct dependence of the outputs on the inputs.

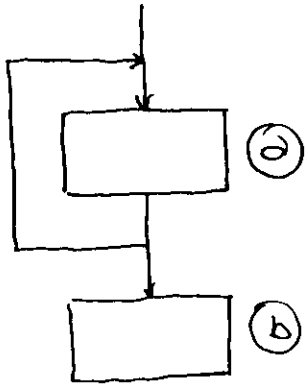
It must be always followed by a state symbol, and must always follow a decision symbol.

SOME COMMON ERRORS OFTEN OCCUR IN THE DESIGN OF ASM CHARTS.

THEY CAN BE:

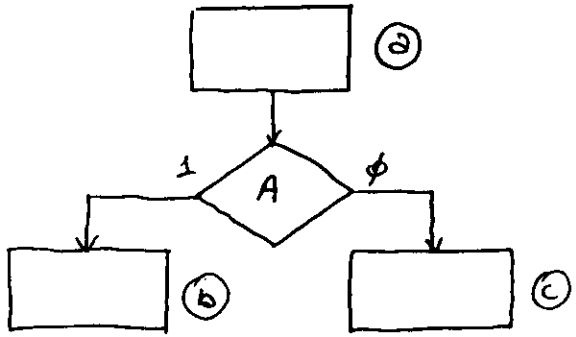


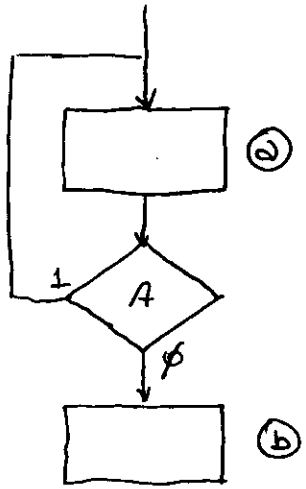
A machine cannot have two states at the same time. No rule is shown to select one state between b and c.



It is not possible to decide what is the next state.

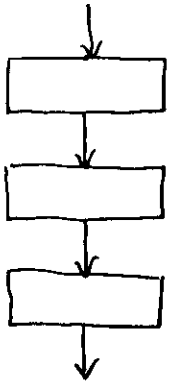
The right configurations are:





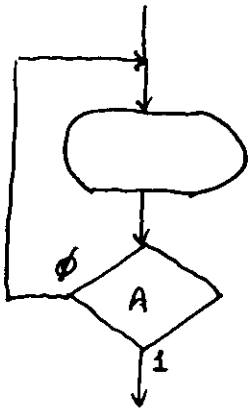
This configuration is called:

WAITING LOOP

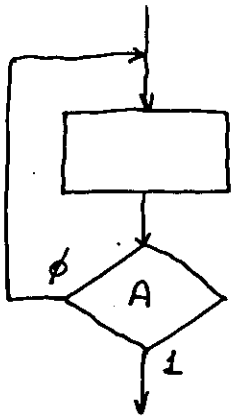


This configuration is valid for synchronous machines only.

In asynchronous machine it produces a continous state change



WRONG



RIGHT

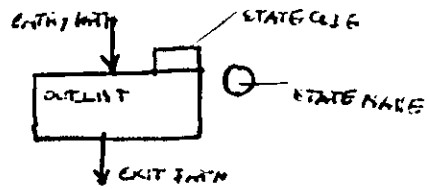
WAITING LOOPS WITH CONDITIONAL OUTPUTS MUST BE AVOIDED.

ASM CHARTS

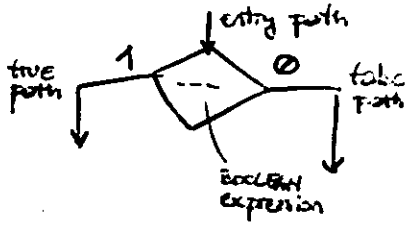
- ASM CHARTS : provide a representation of the state transition and output functions of a state machine

- BASIC BUILDING BLOCKS:

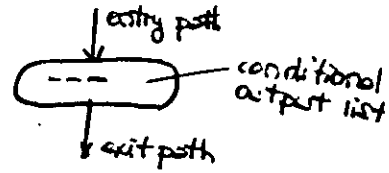
  — STATE BOX



  — DECISION BOX



  — CONDITIONAL OUTPUT BOX

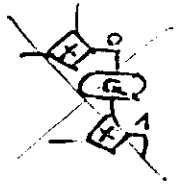


- NOTES
  - only STATE BOXES have timing correspondence
  - each path must be closed in a state box

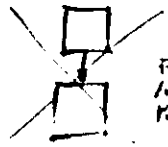
- INVALID REPRESENTATION

LOOPING

LINK PATHS

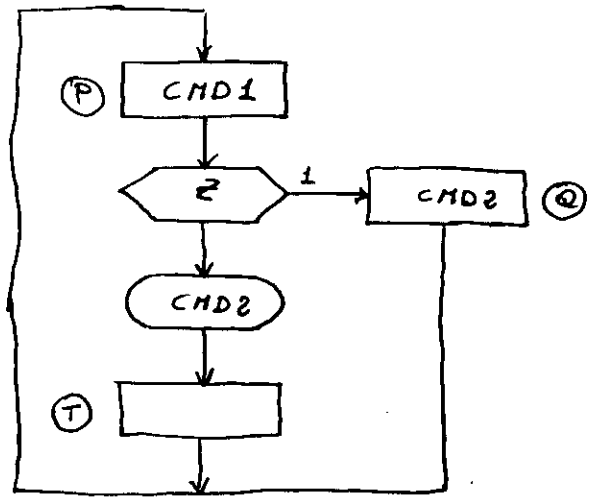


ASYNC. GLITCHES

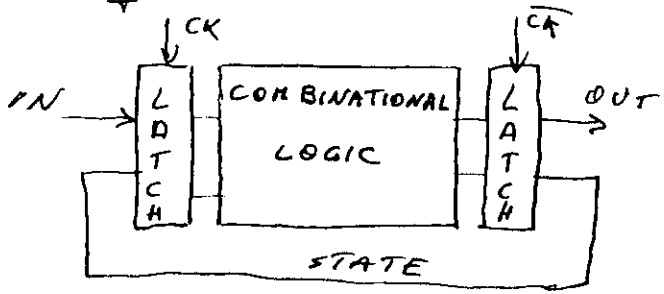


SYNTHESIS FROM AN ASM CHART

Consider the following ASM CHART:

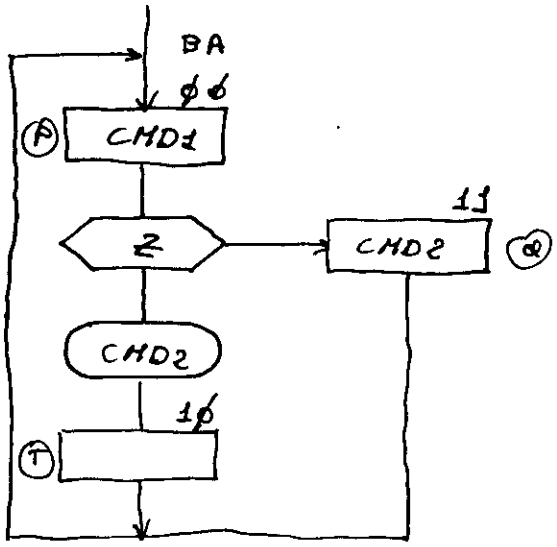


The synthesis process allows to design the function performed by the combinational logic of an ASM.



- The first step is to perform the state assignment of binary state variable values to states.
- In synchronous state machines with synchronous inputs the assignment can be arbitrary.

We choose to have two state variables (A, B) with the following state assignment.



The second step is to design the truth table of the combinational logic.

INPUT		OUTPUT		
z	BA	BA	CMD1	CMD2
ϕ	$\phi\phi$	1ϕ	1	1
1	$\phi\phi$	11	1	ϕ
—	1ϕ	$\phi\phi$	ϕ	ϕ
—	11	$\phi\phi$	ϕ	1

$$B = \overline{B}\cdot\overline{A}\cdot\overline{z} + \overline{B}\cdot\overline{A}\cdot z = \overline{B}\cdot\overline{A}$$

$$A = \overline{B}\cdot\overline{A}\cdot z$$

$$CMD1 = \overline{z}\cdot\overline{A}\cdot\overline{B} + B\cdot A$$

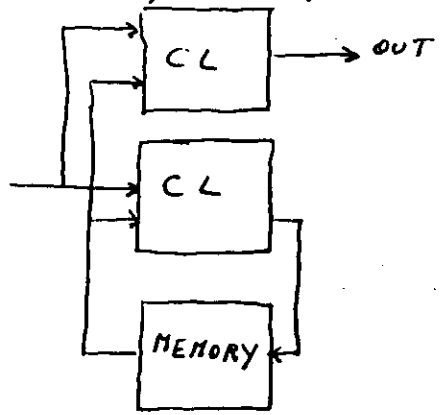
$$CMD2 = \overline{B}\cdot\overline{A}$$

The KARNAUGH MAPS CAN BE USED TO MINIMIZE LOGIC EQUATIONS

CLASSIFICATION OF THE ALGORITHMIC STATE MACHINES

The ASM's have been sub-divided into 5 classes.

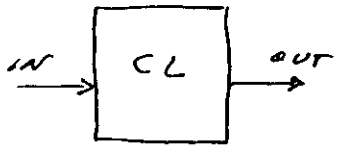
The schematic diagram of an ASM is the following:



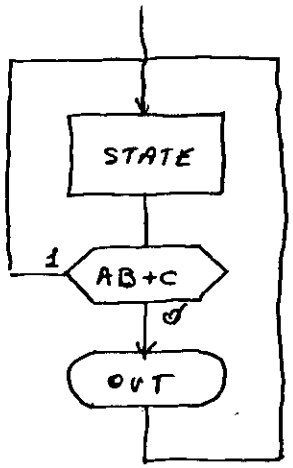
This schematic includes the all 5 classes and it represents the more general ASM.

CLASS ϕ MACHINE:

The class ϕ machine is composed of a combinational circuit:



It has only one state. The output are always of conditional type.



CLASS 1 MACHINE

the next state does not depend on the current one and the outputs do not depend on inputs.

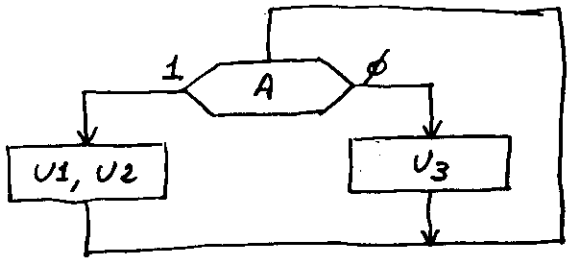
A clan 1 machine is described by the following equations:

$$X \leftarrow f(IN)$$

$$OUT \leftarrow g(x)$$

An ASM CHART for a CLASS ϕ1 machine does not contain any conditional outputs.

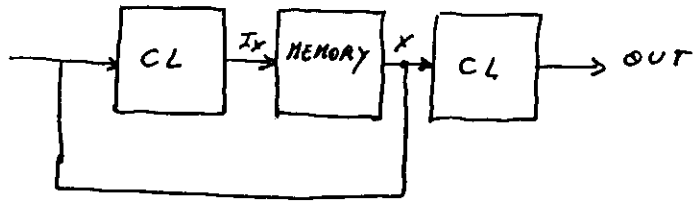
Example:



Truth table

INPUTS A	NEXT STATE x	NEXT STATE y	OUTPUTS $U_1 U_2 U_3$
ϕ	1	1	ϕ ϕ 1
1	ϕ	ϕ	1 1 ϕ

CLASS 2 MACHINE



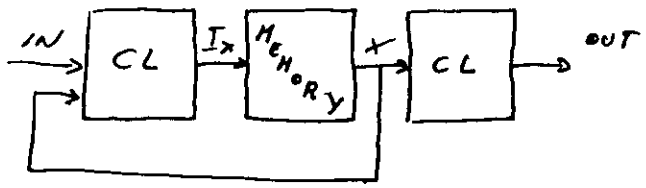
The class-2 machines do not have any input, the outputs only depend on the state.

$$OUT = g(x)$$
$$x = f(x)$$

These machines execute sequences of states, an example is given by the synchronous counter.

An ASM CHART for this class does not contain any decision symbol.

CLASS 3 MACHINE



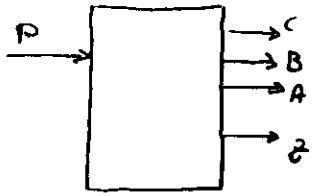
A state depend on both the previous state and the inputs.
It is described by the following equations:

$$x \leftarrow f(x, in)$$
$$OUT \leftarrow g(x)$$

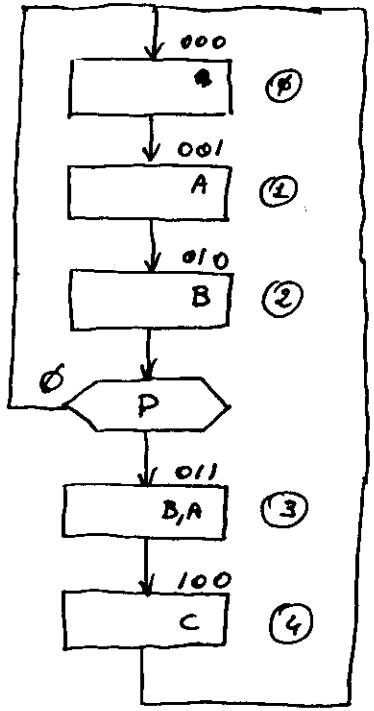
Example:

Design a synchronous counter with an input P and an output z.
If P is true the counter counts up to five,
otherwise the counter counts up to 3.

z signals when the counter reaches the value "ϕ_4".



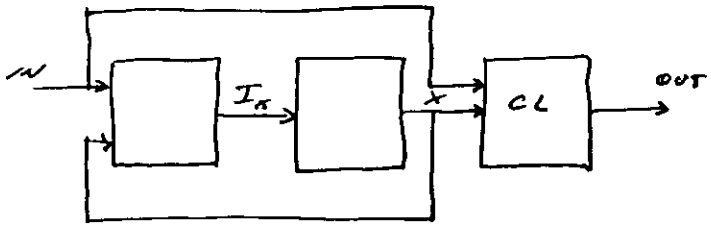
A, B, C = state variables



Truth table

INPUT	P.STATE	N.STATE	OUT
P	CBA	CBA	z
—	000	001	1
—	001	010	0
0	010	000	0
1	010	011	0
—	011	100	0
—	100	000	0

CLASS-4 MACHINE

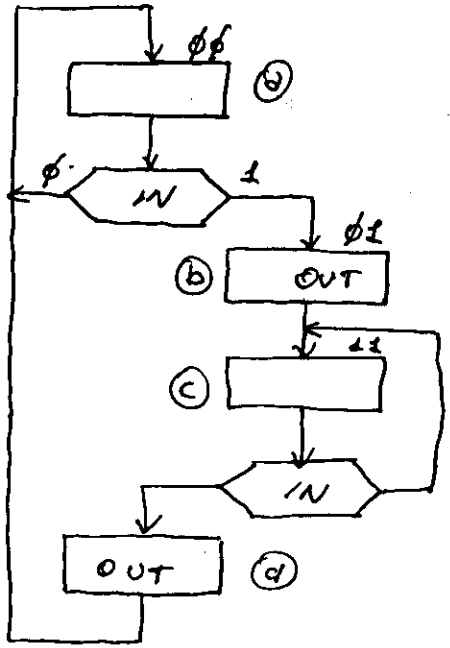


A class-4 machine CHART can contain conditional outputs.

Design an ASM which with the input of the form:

outputs the followings waveform:

We can use a SYNCHRONOUS ASM with clock signal whose period is very short with respect to the input signal period.



INPUTS	PRESENT STATE	NEXT STATE	OUTPUTS
IN	A B	A B	OUT
ϕ	0 0	0 0	0
1	0 0	0 1	0
–	0 1	1 0	1
1	1 0	1 0	0
ϕ	1 0	1 1	0
–	1 1	0 0	1

state assignment

A \ B	0	1
ϕ	a	b
1	d	c

ASM STATE ASSIGNMENT

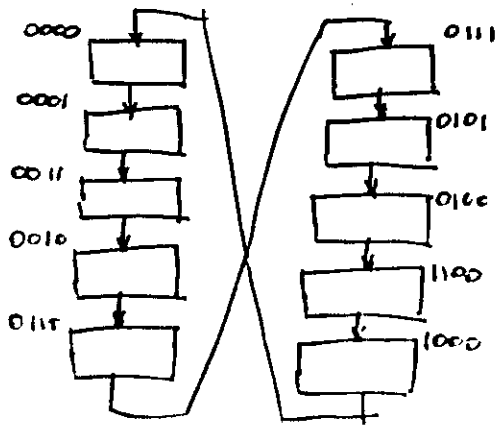
- It is the most difficult step to solve
  - Codes must be unique.
  - Different assignments lead to different complexity.
  - Invalid assignment might lead to races.
- The state is kept in a STATE REGISTER of k memory elements.
  If n is the number of states, it must be $2^k \geq n$

OPTIMIZATION CRITERIA

- Definition:
  HAMMING DISTANCE: number of different bits in two different configurations

1. MINIMIZE THE HAMMING DISTANCE OF LINKED STATES
2. MINIMIZE THE HAMMING DISTANCE OF STATES DEPENDING FROM SAME INPUT VARIABLES

Example: modulo 10 counter (sync.)



- binary sequence coding is not the best ($\Sigma H = 14$)
- with a Gray coding we have the optimum ($\Sigma H = 10$)

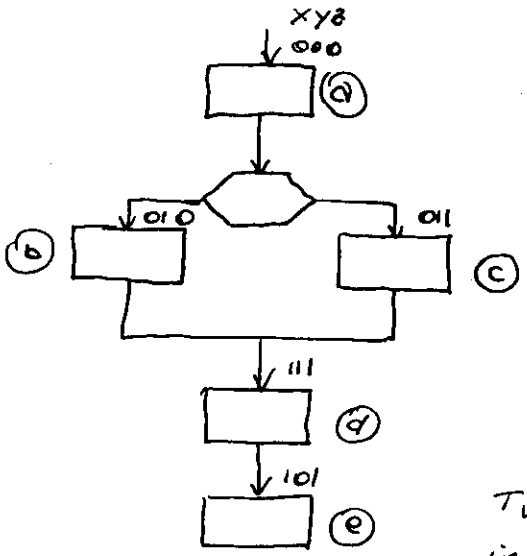
STATE ASSIGNMENT

There are two general criteria that can be used to assign states in an ASM CHART.

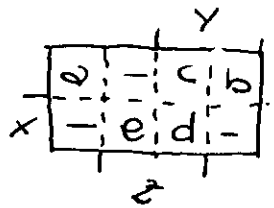
1) Minimization of the number of variable changes

These criterium use the Hamming distance. It is defined Hamming distance between two states the number of variables which change moving from one state to the other.

Consider the following ASM CHART:



Ex: a: 000, c: 011
Hamming distance a − c = 2



Two contiguous states in the map have Hamming distance = 1

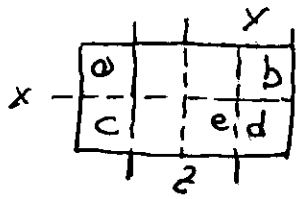
The criteria consist in evaluating the Hamming distance for all the possible state transitions:

TRANSITION	DISTANCE	TRANSITION	DISTANCE
a − b	1	c − d	1
a − c	2	d − e	1
b − d	2	e − a	2

The sum of the distances is 9.

The optimum assignment is that where the sum is minimum.

In our case it is possible to have a better assignment



In fact:

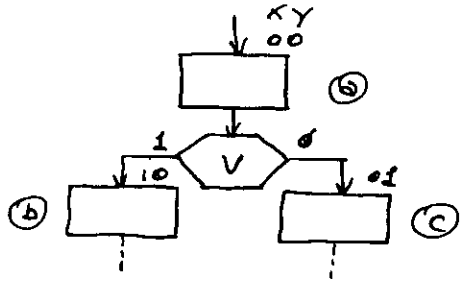
TRANSITION	DISTANCE	TRANSITION	DISTANCE
a − b	1	c − d	1
a − c	1	d − e	1
b − d	1	e − a	3

The sum is 8.

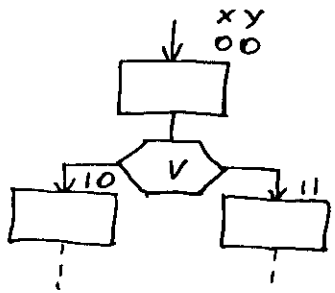
2) Reduction of the dependency on input variables

EX:

The following chart:



can be transformed in:



where x does not depend on V.

In an ASM chart with many decision blocks it is convenient to limit the dependency of the state variables on input variables.

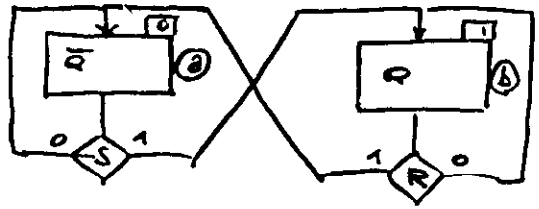
MEMORY ELEMENTS: SR FLIP FLOP

ASYNCHRONOUS memory element
SET-RESET Flip-Flop

1. Transition table

R	S	Q_{n+1}	$\bar{Q}_{n+1}$
0	0	Q_n	$\bar{Q}_n$
1	0	0	1
0	1	1	0
1	1	–INVALID–	

2. ASM CHART

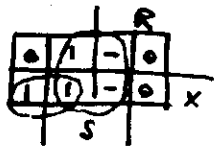


1 state var (X)
2 states S

3. NEXT STATE FUNCTION $X_{next} = f(X, S, R)$

We can suppose that (11) is an invalid configuration and introduce a don't care (R=1, S=1)

- map for next X:



$X = S + X \cdot \bar{R}$

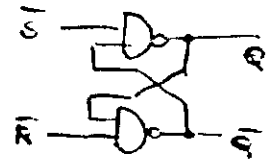
(or $\bar{X} = R + \bar{X} \cdot \bar{S}$ if we consider circling of 0's)

4. OUTPUT FUNCTION

Trivial: $Q = X$, $\bar{Q} = \bar{X}$

5. CIRCUIT IMPLEMENTATION

- using NAND gates $X = \overline{\bar{S} \cdot (X \cdot \bar{R})}$

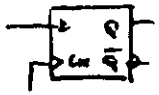


- using NOR gates $X = \overline{R + (X + \bar{S})}$

MEMORY ELEMENTS: D Flip Flop

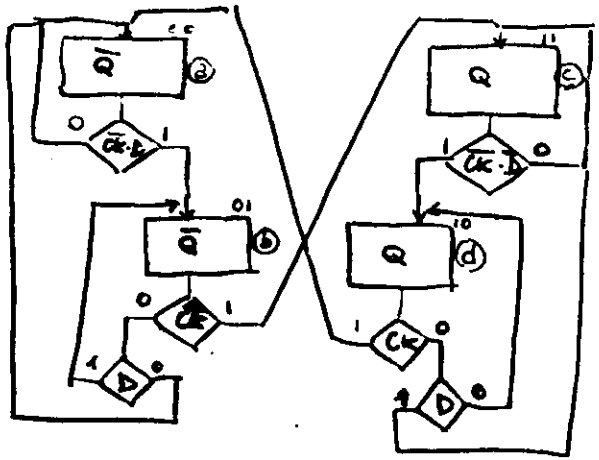
- SYNCHRONOUS D Flip Flop

  Described as an async. machine (with ck as input)

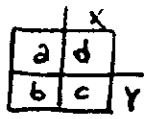


ck	D	Q_{n+1}	$\bar{Q}_{n+1}$
0	–	Q_n	$\bar{Q}_n$
1	–	Q_n	$\bar{Q}_n$
↑	0	0	1
↑	1	1	0

- ASM CHART

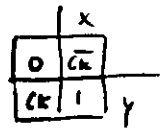


- STATE ASSIGNMENT

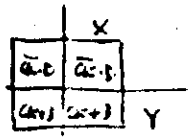


(2 STATE VARS)
(4 STATES)

- NEXT STATE



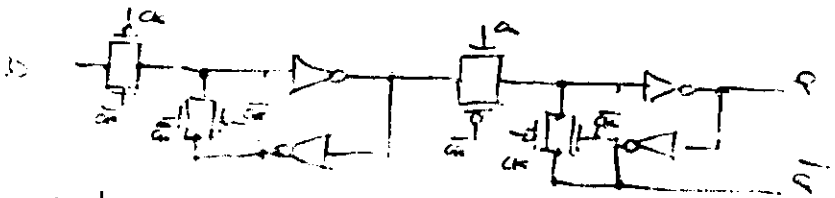
X_{NEXT}: $X = XY + ck \cdot Y + \bar{ck} \cdot X$



Y_{NEXT}: $Y = \bar{ck} \cdot D \cdot \bar{Y} + ck \cdot Y + D \cdot Y$

- CIRCUIT IMPLEMENTATION

  In CMOS technology: it's need a completely different approach:



(only 16 trans.)

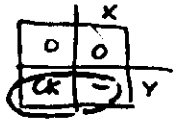
SYNTHESIS OF ASM CHARTS WITH FLIP-FLOPS

- Memory elements can be used to synthetize ASM charts. Next state functions are expressed in less complex ways

Ex.: for the D flipflop (as an ASYNC. circuit) we can use 2 SR for next state.

We have to compute SET and RESET maps for all state variables:

S_X

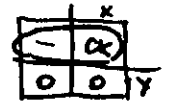


$ck \cdot Y$

S_Y

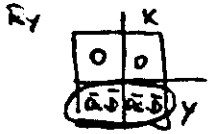
$\bar{ck} \cdot D \cdot \bar{Y}$

R_X



$ck \cdot \bar{Y}$

R_Y

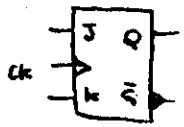


$\bar{ck} \cdot \bar{D} \cdot Y$

- Much easier function are derived

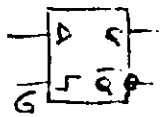
- OTHER MEMORY ELEMENTS:

JK- FLIP FLOP



J	K	ck	Q_{n+1}	$\bar{Q}_{n+1}$
–	–	↓		
–	–	↑	Q_n	$\bar{Q}_n$
1	0	↑	1	0
0	1	↑	0	1
0	0	↑	Q_n	$\bar{Q}_n$
1	1	↑	$\bar{Q}_n$	Q_n

LATCH



D	G	Q_{n+1}	$\bar{Q}_{n+1}$
–	0	Q_n	$\bar{Q}_n$
1	1	1	0
0	1	0	1

RACES

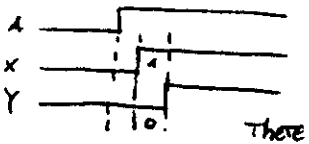
ASYNCHRONOUS MACHINES

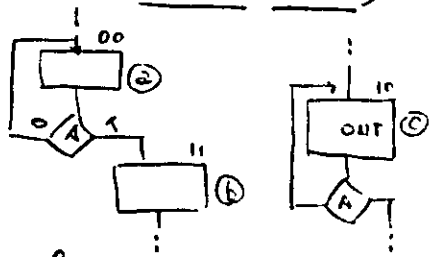
- STATE RACES

  when an intermediate transient state can be captured by the input values and converted into an incorrect stable state (CRITICAL RACE)

Ex.:

If the transitions are delayed





There is a spurious configuration (10) between (00) and (11)

The machine goes to state © instead than state (b) from which it oscillates back to (b)

- TO AVOID CRITICAL RACES:

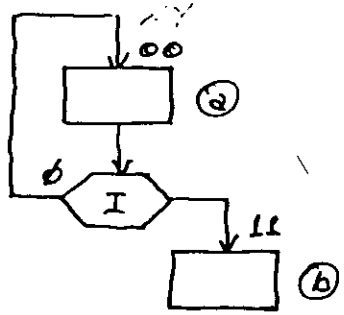
  [SUBSEQUENT STATES CODES MUST HAVE DISTANCE EQUAL TO 1 (ASYNC. MACHINES)]

NOTES:

- Not always a state assignment is feasible
  Sometimes it may be necessary to
  - INTRODUCE EMPTY STATES
  - INCREASE THE NUMBER OF STATE VARIABLES

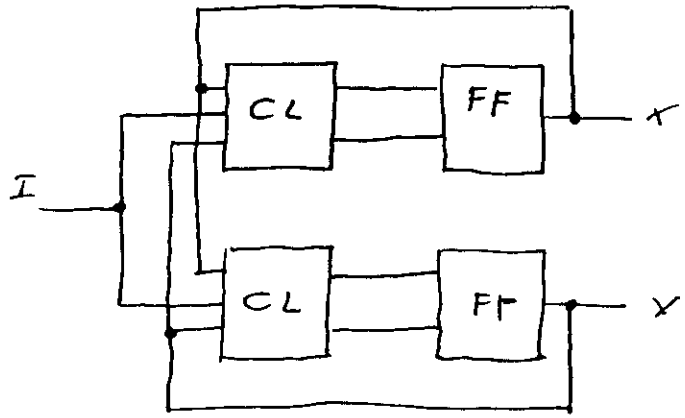
STATE ASSIGNMENT IN ASYNCHRONOUS ASM

Consider the following chart:

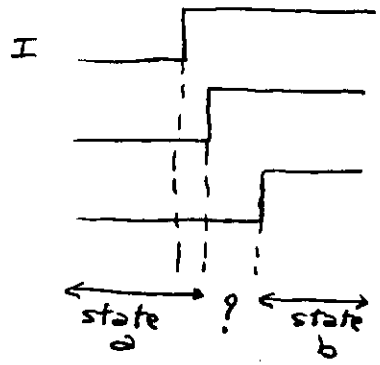


which is part of an ASM CHART with 4 states.

The asynchronous ASM will be composed of two SR flip-flops and two combinational logics.



When I has a transition from ϕ to 1, the flip-flops have to set to 1. But we cannot expect that x and y reach the 1 at the same time.



we have 3 different situations:

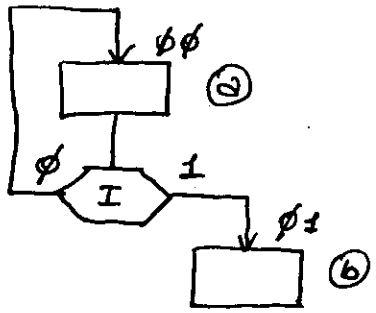
1) (a) → (?) → (b): From state (a) the machine goes to state (?) and then to state (b).
   If in the ASM the (?) exists we can have:

2) (a) → (?) : The machine assumes the (?) state as a stable state.

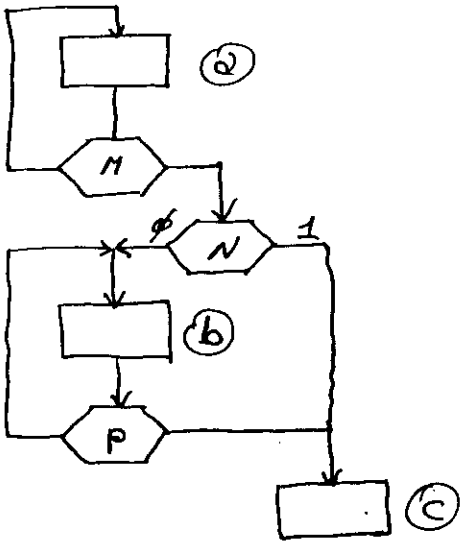
3) (a) → (?) → (??): The state of the machine is unpredictable.

To avoid the situations shown above it is necessary that the distances among the states are always 1.

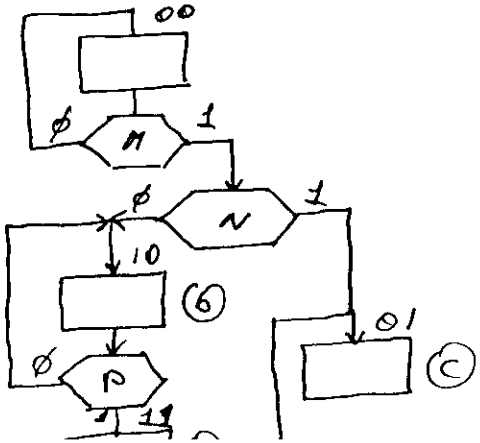
In this way a state transition causes the change of only one variable and spurious states cannot reached



It is not always possible to have the distance equal to 1.
Consider the following chart:



It is necessary to add a transition state to insure that the distance between state is 1.



transition

An alternative to the addition of transition states is the introduction of new state variables.
In this way larger maps are obtained and is easier to have distance equal 1.

RACES

SYNCHRONOUS MACHINES

- **STATE RACES**

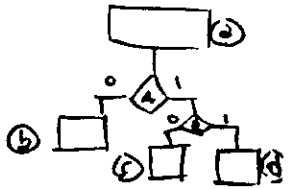
  SYNC. INPUT : there is never a problem : transients die down before the next edge

  ASYNC. INPUT : it is possible that a ck edge latches a variable in a transient.
  We must pay attention that only ONE STATE variable depends on an async. input

TO AVOID STATE RACES :

> ALL STATES DEPENDING FROM THE VALUE OF ASYNC. INPUTS MUST HAVE DISTANCE EQUAL TO 1
> (SYNC. MACHINES)

NOTE :
- sometimes impossible situations may arise

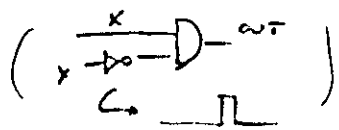


Then it's necessary
- to MODIFY THE CHART
- to SYNCHRONIZE INPUT VARIABLES

b-d, b-c, c-d } with distance 1 IT IS IMPOSSIBLE

- **OUTPUT RACES**

  when state variable transition occurr with finite delays : glitches may be present in some outputs



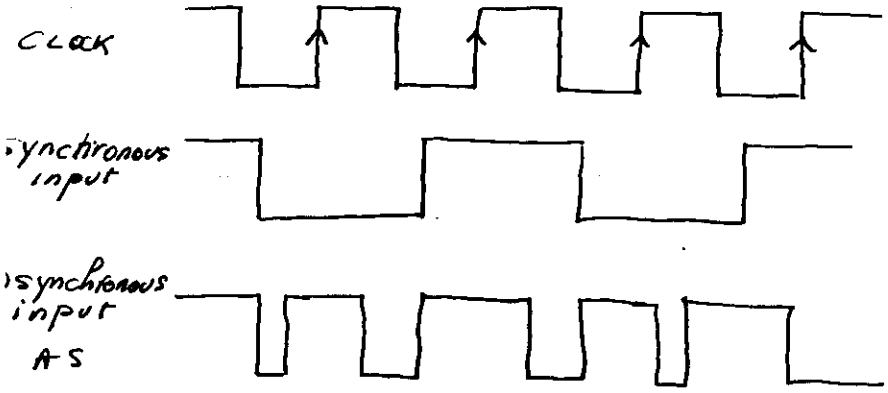
TO AVOID OUTPUT RACES :

1. Impose distance 1 to successive states (adjacent)
2. synchronise outputs (but are delayed)
3. Validate outputs with a STROBE
5. Acknowledge when next circuits are synchronous

STATE ASSIGNMENT IN SYNCHRONOUS ASM

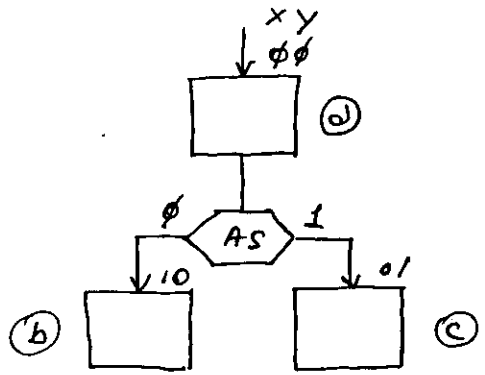
TRANSITION RACES :

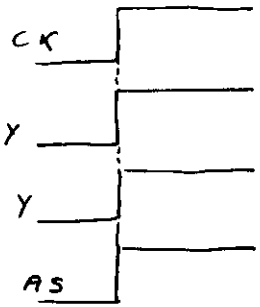
SUPPOSE to have an ASM active on the rising edge of the clock. We know that synchronous inputs have a fixed relation with the clock, while asynchronous inputs are random.



We have problems with asynchronous inputs when they change at the same time of the clock.

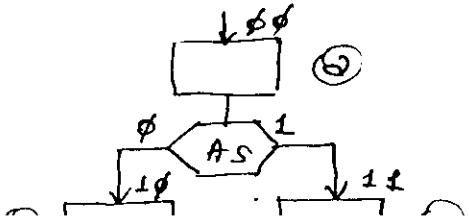
Consider the following ASM chart:





If AS changes on the rising edge of CK we don't know how it will be interpreted (if AS = φ, or AS = 1).
Therefore from state (a) the machine can indifferently move to state (b) or to state (c).
But it is also possible that, for the delays of the flip-flops, the machine reaches the states φ φ or 1 1.

To insure that the machine reaches the states b or c it is necessary that the two states have distance equal to 1.

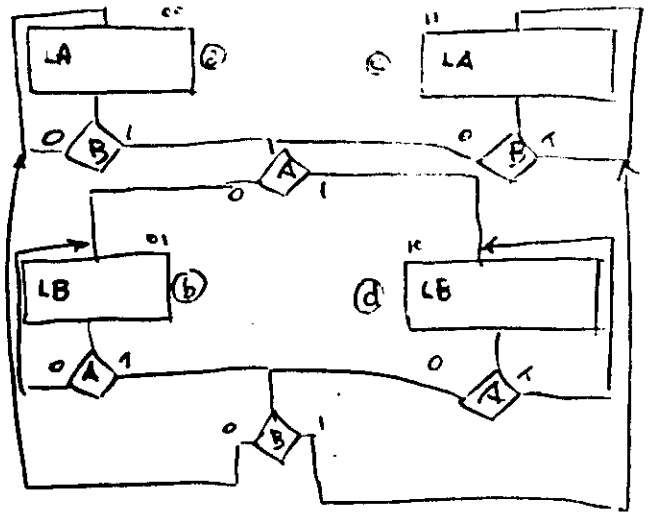


EXAMPLE 1
ASYNCHRONOUS CIRCUIT

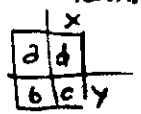
Problem: design of a circuit which finds which of two variables was the last to toggle.

LA = last A
LB = last B = $\bar{A}$

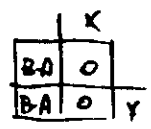
• ASM CHART:



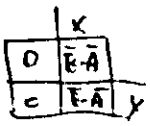
STATE ASSIGNM.:



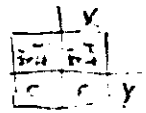
NEXT STATE:
we use 2 SR flip flops



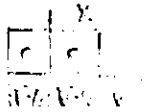
$S_x = \bar{x} \cdot B \cdot A$



$R_x = x \cdot \bar{B} \cdot \bar{A}$



$S_y = B \cdot \bar{A} \cdot \bar{B}$



$R_y =$

• OUTPUT FUNCTIONS:

LA

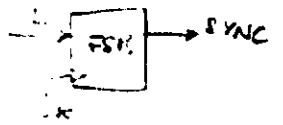
$LA = \bar{x} \cdot \bar{y} + x \cdot y = \overline{x \oplus y}$

LB

$LB = x \cdot \bar{y} + \bar{x} \cdot y = x \oplus y$

EXAMPLE: SYNCHRONOUS CIRCUIT

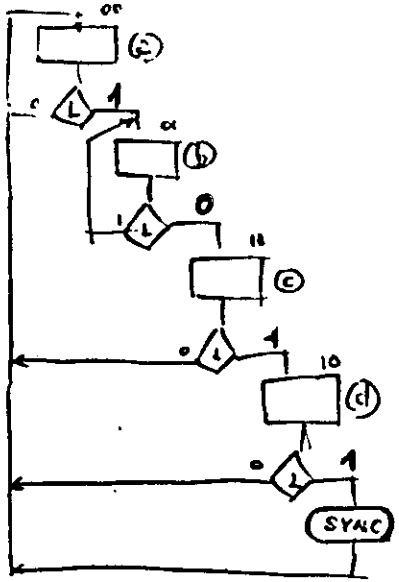
Bit stream decoder, with synchronous line output



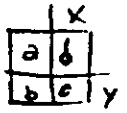
we have to identify the sequence

1011

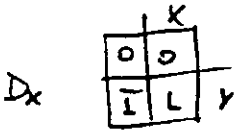
ASM Chart:



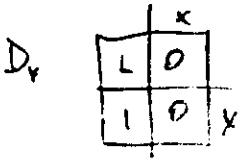
- STATE ASSIGNMENT:



- NEXT STATE:
  we use 2 D flip-flops



$D_x = \bar{1}\bar{x}Y + LXY$

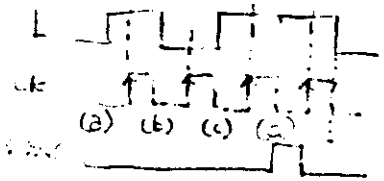


$D_y = \bar{x}y + L\bar{x}$

- OUTPUT FUNCTIONS:

$SYNC = x\bar{y} \cdot L$

NOTE: the sync signal lasts only for half clk



If we want an output pulse we must add another state (and another state var.)

To avoid output glitches:

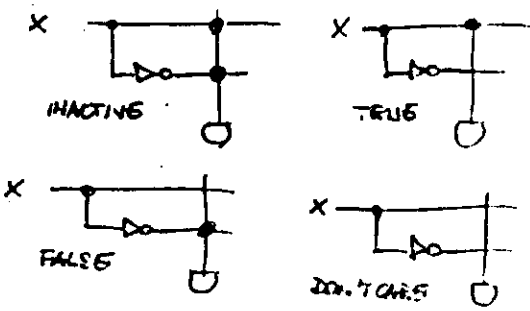
$SYNC = x\bar{y} \cdot L\overline{clk}$ (strobe)

SYNTHESIS OF LOGIC FUNCTIONS WITH PROGRAMMABLE DEVICES

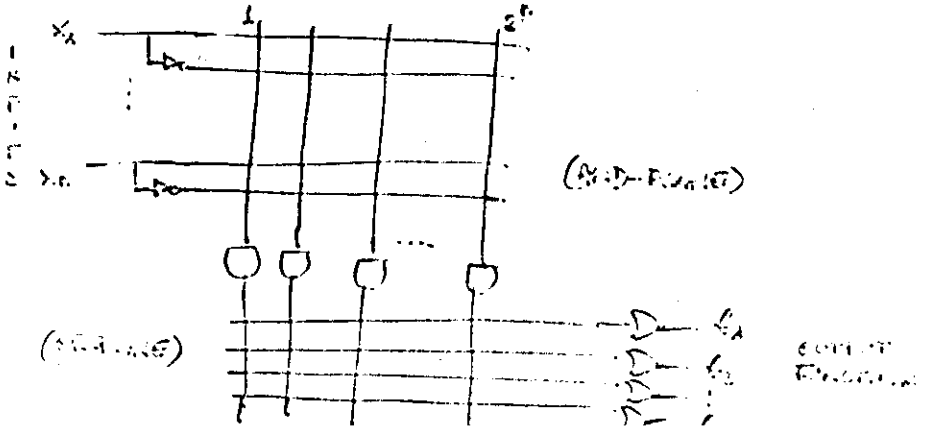
- Development in IC technology → low cost, flexible programmable devices
  - PLA programmable logic arrays
  - ROM read only memory
- LOGIC ARRAYS: consist of a two-level array an AND ARRAY, connected to an OR ARRAY

  ENABLE LOGIC FUNCTIONS IN SUM-OF-PRODUCTS FORM TO BE EASILY IMPLEMENTED

  In all devices programming involves breaking or creating connections in strategic points in the array
- POSSIBLE CONNECTIONS

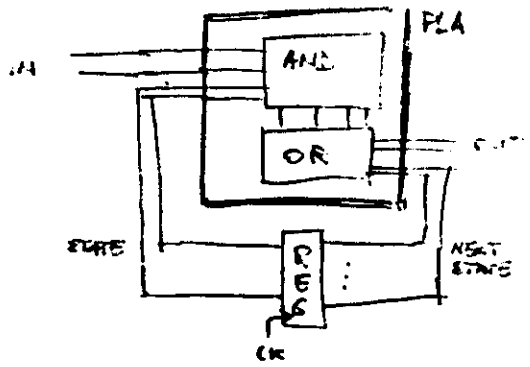


- ARRAY ORGANIZATION



ASM WITH PLA's

- possible to organize PLA's and memory elements for synthesize ASM.



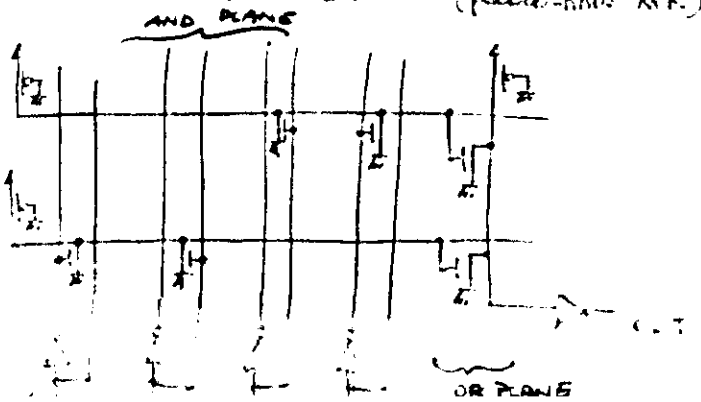
NOTES

- PLA's have intrinsic hazards
  then ONLY SYNC. MACHINES can be implemented with them
- All inputs must be synchronous (or synchronized)
- It's not possible to avoid OUTPUT RACES

ADVANTAGES

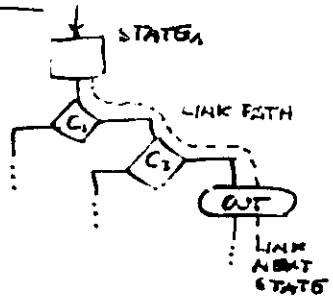
- low design and circuit costs
- easy to implement
- there is no need for optimization (all MINTERMS)

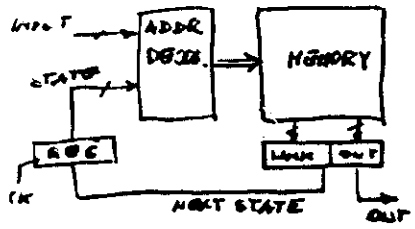
EXAMPLE OF IMPLEMENTATION OF PLA (pseudo-nMOS M.F.)



TECHNIQUES FOR ASM DESIGN WITH ROMs

1. CONTROL WORD PER ASM LINK PATH



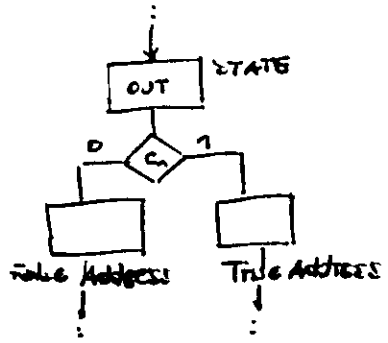


- for each path we keep in memory next state and outputs
- All outputs must be conditional in order to avoid one cycle delay

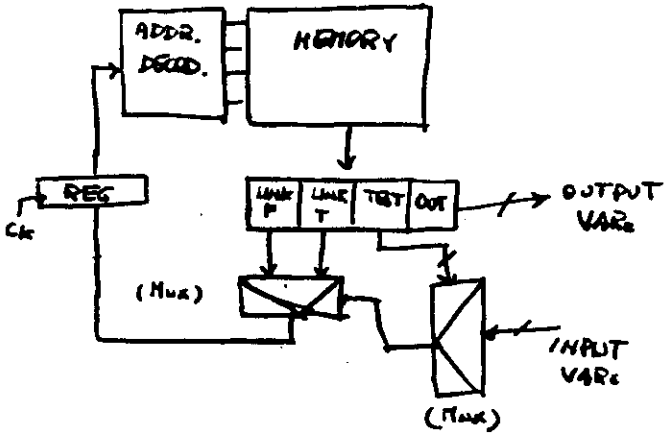
- Drawbacks:
  - If there are many input variables, the memory grows considerably, while there are few relevant selections (e.g.: introducing 1 variable doubles the memory size).
- There are more efficient implementations, considering some limiting hypothesis on the ASM charts.
- Memory requirements
  for an ASM with k state variables
  n inputs
  m outputs
  The fully decoded approach requires
  $(k+m) \cdot 2^{k+n}$ bits

2. ONE WORD PER STATE-LAST FAR

Hypothesis:
- No conditional outputs
- Only one decisional block per state.
  (2 alternatives)



We can organize memory:
1. INPUT VARIABLES (CODED)
2. TRUE STATE
3. FALSE STATE
4. OUTPUTS



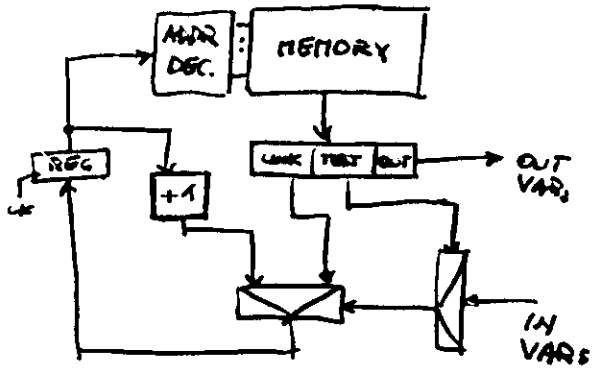
- k state variables
  n inputs
  m outputs

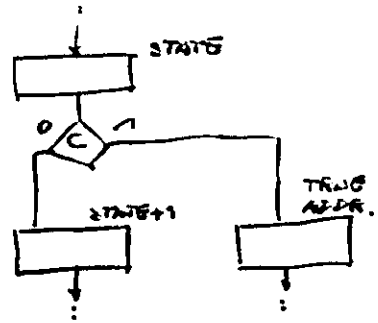
  Memory requirements $(e_0 n + 2k + m) \cdot 2^k$ bits

- Some more hardware is required
  - k multiplexer 2-to-1 to choose false or true link path
  - one multiplexer n-to-1 to choose input variables
- + Large amount of memory is saved ($\sim 2^n$) with respect to the fully decoded approach

3. DEFAULT PATH (STATE+1)

Hypotheses:
- We can save more memory assuming that one of the 2 possible addresses is the previous one +1.
- one address is redundant and can be eliminated





- Memory requirements

  k state
  n input
  m output

  $(e_0 n + k + m) 2^k$ bits

- Hardware requirements: Incrementer (and the same multiplexers)

NOTES:

1. As not all states have outputs, it is sometimes convenient to define 2 FORMATS for the memory

   where 1 bit specifies the kind of format

1	OUTPUT	
0	TEST	LINK

2. The structure becomes more and more similar to the CONTROL PART of microprocessors (with microprograms)