

Variable Entered Mapping (VEM)

This is a powerful Karnaugh Map technique where 1's, 0's, X's (0's) can be plotted as usual but also Boolean variables and expressions can be entered

VEM Plotting Theory

A	B	C	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

General Sum of Product (SOP)
expression for F is:

$$F = \bar{A}.\bar{B}.\bar{C} + \bar{A}.\bar{B}.C + \bar{A}.B.\bar{C} \\ + \bar{A}.B.C + A.\bar{B}.\bar{C} + A.\bar{B}.C \\ + A.B.\bar{C} + A.B.C$$

Consider the following:

$$F.F = F = \bar{A}.\bar{B}.\bar{C}.F + \bar{A}.\bar{B}.C.F + \dots + A.B.C.F$$

Consider this idea for a specific example:

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	$\emptyset(x)$
1	1	1	$\emptyset(x)$

for this example:

$$\begin{aligned}
 f.f = F &= \bar{A}\bar{B}\bar{C}(0) + \bar{A}\bar{B}C(0) \\
 &+ \bar{A}B\bar{C}(1) + \bar{A}BC(1) \\
 &+ A\bar{B}\bar{C}(1) + A\bar{B}C(0) \\
 &+ AB\bar{C}(\emptyset) + ABC(\emptyset)
 \end{aligned}$$

From this idea we plot the values of F in a 2 variable maps

AB \ C	00	01	11	10
0	0	1	$\emptyset$	1
1	0	1	$\emptyset$	0

We develop VEM plotting by extending this idea of plotting the value of a variable into the map.

Consider grouping the least significant variable with the variable F in the general expression for F:

$$\begin{aligned}
 F = F &= \bar{A}.\bar{B}.(\bar{C}.F) + \bar{A}.\bar{B}.(C.F) + \bar{A}.B.(\bar{C}.F) \\
 &+ \bar{A}.B.(C.F) + A.\bar{B}.(\bar{C}.F) + A.\bar{B}.(C.F) \\
 &+ A.B.(\bar{C}.F) + A.B.(C.F)
 \end{aligned}$$

$$\begin{aligned}
 &= \bar{A}.\bar{B}(\bar{C}.F + C.F) + \bar{A}.B(\bar{C}.F + C.F) \\
 &+ A.\bar{B}(\bar{C}.F + C.F) + A.B(\bar{C}.F + C.F)
 \end{aligned}$$

This can result in a much simpler K-Map implementation using VEM plotting.
Consider the following:

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	$\emptyset$
1	1	1	$\emptyset$

Enter the specific values for F into the above equation

$$\begin{aligned}
 F &= \bar{A}.\bar{B}(\bar{C}.0 + C.0) + \bar{A}.B(\bar{C}.1 + C.1) \\
 &+ A.\bar{B}(\bar{C}.1 + C.0) + A.B(\bar{C}.\emptyset + C.\emptyset)
 \end{aligned}$$

$$\begin{aligned}
 &= \bar{A}.\bar{B}.(0) + \bar{A}.B.(C + \bar{C}) \\
 &+ A.\bar{B}(\bar{C}) + A.B(\emptyset)
 \end{aligned}$$

$$= \bar{A}.\bar{B}.(0) + \bar{A}.B.(1) + A.\bar{B}(\bar{C}) + A.B(\emptyset)$$

This expression can be plotted on a 2-Variable K-Map.

		A	
		0	1
B	0	0	$\bar{C}$
	1	1	$\emptyset$

What we are actually plotting into the cell of the map is the value of $\bar{C} \cdot F + C \cdot \bar{F}$.

Example

A	B	C	F ₁	F ₂	F ₃	F ₄
0	0	0	0	1	0	1
0	0	1	1	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	1	1
1	0	0	1	0	0	1
1	0	1	1	0	1	0
1	1	0	0	1	1	0
1	1	1	0	1	0	∅

A	0	1
B	0	1
	C	$\bar{C}$
	0	1
	1	0

F₁

A	0	1
B	0	1
	$\bar{C}$	0
	0	1
	1	C

F₂

A	0	1
B	0	1
	0	C
	0	1
	1	$\bar{C}$

F₃

A	0	1
B	0	1
	0	1
	0	$\bar{C}$
	1	C
	1	C∅

F₄

"special" notation indicating this cell is only to be used if simplification results

EXAMPLE

A	B	C	D	F ₁	F ₂
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	1
0	0	1	1	1	∅
0	1	0	0	∅	1
0	1	0	1	0	0
0	1	1	0	1	∅
0	1	1	1	0	0
1	0	0	0	1	0
1	0	0	1	1	1
1	0	1	0	0	1
1	0	1	1	1	1
1	∅	0	0	∅	∅
1	1	0	1	0	∅
1	1	1	0	0	0
1	1	1	1	0	0

AB \ C	00	01	11	10
0	0	$\bar{D}\emptyset$	$\bar{D}\emptyset$	1
1	1	$\bar{D}$	0	D

F₁

AB \ C	00	01	11	10
0	1	$\bar{D}$	∅	D
1	$\bar{D} + D\emptyset$	$\bar{D}\emptyset$	0	1

F₂

We could let this go a stage further by taking C and D together with F. Let F₂ = F and then we have:

$$F = \bar{A} \cdot \bar{B} \cdot (\bar{C} \cdot \bar{D} \cdot F + \bar{C} \cdot D \cdot F + C \cdot \bar{D} \cdot F + C \cdot D \cdot F) + \bar{A} \cdot B \cdot (\bar{C} \cdot \bar{D} \cdot F + \bar{C} \cdot D \cdot F + C \cdot \bar{D} \cdot F + C \cdot D \cdot F) + A \cdot \bar{B} \cdot (\bar{C} \cdot \bar{D} \cdot F + \bar{C} \cdot D \cdot F + C \cdot \bar{D} \cdot F + C \cdot D \cdot F) + A \cdot B \cdot (\bar{C} \cdot \bar{D} \cdot F + \bar{C} \cdot D \cdot F + C \cdot \bar{D} \cdot F + C \cdot D \cdot F)$$

A \ B	0	1
0	$\bar{C} \cdot \bar{D} + \bar{C} \cdot D + (C \cdot D) \cdot \emptyset$	$\bar{C} \cdot D + C \cdot \bar{D} + C \cdot D$
1	$\bar{C} \cdot \bar{D} + (C \cdot \bar{D}) \cdot \emptyset$	$(\bar{C} \cdot \bar{D}) \cdot \emptyset + (C \cdot D) \cdot \emptyset$

This can lead to difficult to read maps so care needs to be taken when using this method.

VEM Reading Theory

Step 1: I imagine all '1' entries are replaced by the MEV ~~or~~ ORed with its complement (eg: $1 = D + \bar{D}$). Then loop according to the following:

- (a) Loop all single MEV entries that will not loop with another identical MEV in an adjacent cell or with a 1 or a $\emptyset$.
- (b) Loop all MEV's that will loop into duals only with another identical MEV in an adjacent cell.
- (c) Loop all MEV's that will loop into a dual only with a 1.
- (d) Loop all MEV's that will loop into a dual only with a $\emptyset$.
- (e) Any MEV that will loop two ways with another identical MEV, 1, or $\emptyset$ but won't loop into a quad, leave until later.
- (f) Continue looping for quads and groups of eight until every MEV has been looped at least once

Step 2: Once all single MEV entries have been covered, transform the map according to the following transformations:

(a) Replace MEV and $\overline{\text{MEV}}$ with ϕ . (MEV and $\overline{\text{MEV}} \rightarrow 0$)

(b) $0 \rightarrow 0$, $\phi \rightarrow \phi$
 $\rightarrow 1$ { if not completely covered

(c) $1 \rightarrow \phi$ { if completely covered, i.e. looped with both the MEV and $\overline{\text{MEV}}$

(d) $\text{MEV} \phi$ and $\overline{\text{MEV}} \phi \rightarrow 0$

$\rightarrow 1$ { if not covered at all or if just the ϕ is covered

(e) $(\text{MEV} + \overline{\text{MEV}} \phi)$ and $(\overline{\text{MEV}} + \text{MEV} \phi)$

$\rightarrow \phi$ { if completely covered or if just the necessary term is covered

necessary variable

Example 1

Read the following map.

$\begin{matrix} B \\ \backslash A \end{matrix}$	0	1
0	C	0
1	1	C

STEP 1:
replace 1 by $C + \bar{C}$ and group the term.

$\begin{matrix} B \\ \backslash A \end{matrix}$	0	1
0	C	0
1	$C + \bar{C}$	C

So far we have $\bar{A}.C$ and $B.C$ as a result of looping

STEP 2:
consider the covered term $C + \bar{C} \rightarrow 1$; $C \rightarrow 0$

$\begin{matrix} B \\ \backslash A \end{matrix}$	0	1
0	0	0
1	1	0

Put together all the terms extracted from Steps 1 and 2.

$$F = \bar{A}.C + B.C + \bar{A}.B$$

Example 2

Plot the truth table as a VEM and then read the result.

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Plot:

A \ B	0	1
0	0	1
1	$C + \bar{C}\phi$	C

step 1:

A \ B	0	1
0	0	$C + \bar{C}$
1	$C + \bar{C}\phi$	C

$B.C$ (indicated by an arrow pointing to the cell containing C in the bottom-right quadrant)

step 2:

A \ B	0	1
0	0	1
1	ϕ	0

$A.\bar{B}$ (indicated by an arrow pointing to the cell containing 1 in the top-right quadrant)

$$\therefore F = B.C + A.\bar{B}$$

Example 3

A	B	C	D	F ₁	F ₂
0	0	0	0	∅	0
0	0	0	1	∅	0
0	0	1	0	1	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	1	1
0	1	1	0	∅	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	1	∅
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	1	1	∅
1	1	1	0	0	1
1	1	1	1	∅	0

Consider F₁

STEP 1

	AB	00	01	11	10
0	C	D∅ + D̄∅	(D + D̄)	(D)	0
1	C	(D̄)	(D̄∅)	D∅	D + D̄

$$F_1 = \bar{A}.\bar{D} + B.\bar{C}.D + \dots$$

STEP 2

	AB	00	01	11	10
0	C	0	0	0	0
1	C	0	0	0	(1)

$$F_1 = \dots + A.B.C$$

$$F_1 = \bar{A}.\bar{D} + B.\bar{C}.D + A.B.C$$

Consider f_2

STEP 1

		$\bar{A}.B.D$			
		00	01	11	10
C	0	0	D	$D\phi$	0
	1	$D+\bar{D}$	$D+\bar{D}$	$\bar{D}$	$\bar{D}\phi$

$C.\bar{D}$

$$F_2 = \bar{A}.B.D + C.\bar{D}$$

STEP 2

		$\bar{A}.C$			
		00	01	11	10
C	0	0	0	0	0
	1	1	1	0	0

$$F_2 = \dots + \bar{A}.C$$

$$f_2 = \bar{A}.B.D + C.\bar{D} + \bar{A}.C$$

Example 4

STEP 1

AB \ C	00	01	11	10
0	$D\phi$	1	$D + D\phi$	ϕ
1	$\bar{D} + D\phi$	0	$\bar{D}$	1

Annotations: $A \cdot \bar{D}$ (pointing to cell 01), $\bar{D}$ (pointing to cell 11), D (pointing to cell 10)

STEP 2

AB \ C	00	01	11	10
0	0	1	1	ϕ
1	1	0	0	1

Annotations: $\bar{D} \cdot \bar{C}$ (pointing to cell 01), $\bar{D} \cdot C$ (pointing to cell 11)

$$F = A \cdot \bar{D} + \bar{D} \cdot \bar{C} + \bar{D} \cdot C$$

STEP 1

AB \ C	00	01	11	10
0	$\bar{D}$	$\bar{D}$	ϕ	D
1	$D\phi + \bar{D}$	$\bar{D}\phi$	0	1

Annotations: $\bar{A} \cdot \bar{D}$ (pointing to cell 00), $\bar{D}$ (pointing to cell 01), $\bar{D}$ (pointing to cell 10)

STEP 2

AB \ C	00	01	11	10
0	ϕ	0	ϕ	0
1	ϕ	0	0	1

Annotations: $\bar{B} \cdot C$ (pointing to cell 10)

$$F = \bar{B} \cdot D + \bar{A} \cdot \bar{D} + \bar{B} \cdot C$$

STEP 1

AB \ C	00	01	11	10
00	E	ϕ	$\bar{E}$	$\bar{E}$
01	$E + E\phi$	0	E	0
11	1	$E\phi$	$E\phi$	1
10	$E\phi + \bar{E}$	1	$\bar{E}$	0

Annotations: $\bar{A} \cdot \bar{D} \cdot E$ (pointing to cell 00), $A \cdot \bar{D} \cdot \bar{E}$ (pointing to cell 01), $A \cdot \bar{D} \cdot E$ (pointing to cell 11), $\bar{B} \cdot \bar{D} \cdot \bar{E}$ (pointing to cell 10)

STEP 2

AB \ C	00	01	11	10
00	0	ϕ	ϕ	0
01	ϕ	0	0	0
11	1	0	0	1
10	1	1	0	0

Annotations: $\bar{B} \cdot D$ (pointing to cell 10), $\bar{A} \cdot C \cdot \bar{D}$ (pointing to cell 11)

$$F = A \cdot \bar{D} \cdot \bar{C} \cdot E + A \cdot \bar{D} \cdot C \cdot \bar{E} + \bar{A} \cdot \bar{D} \cdot E + \bar{B} \cdot \bar{D} \cdot \bar{E} + \bar{A} \cdot C \cdot \bar{D} + \bar{B} \cdot C \cdot D$$

Example 5

Given:

$$F(A, B, C, G, H) = \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot \bar{H} + \bar{A} \cdot \bar{B} \cdot C \cdot \bar{H} + \bar{A} \cdot B \cdot C \\ + A \cdot \bar{B} \cdot \bar{C} \cdot \bar{G} + A \cdot \bar{B} \cdot C \cdot \bar{G} + A \cdot B \cdot \bar{C} + A \cdot B \cdot C$$

where G, H are "infrequently used variables".

We can group F as:

$$F = (\bar{A} \cdot \bar{B} \cdot \bar{C}) \cdot (\bar{H}) + (\bar{A} \cdot \bar{B} \cdot C) \cdot (\bar{H}) + (\bar{A} \cdot B \cdot C) + (A \cdot \bar{B} \cdot \bar{C}) \cdot (\bar{G}) \\ + (A \cdot \bar{B} \cdot C) \cdot (\bar{G}) + (A \cdot B \cdot \bar{C}) + (A \cdot B \cdot C)$$

This leads to the following map:

AB \ C		00	01	11	10
C	0	$\bar{H}$	0	1	$\bar{G}$
	1	$\bar{H}$	1	1	$\bar{G}$

SYSTEM CONTROLLER DESIGN

A system controller is the "nerve-centre" of the system.

The following sequence is a good design procedure to approach such designs:

- ① DEFINE THE PURPOSE AND ROLE OF THE SYSTEM
- ② DEFINE BASIC OPERATIONS AND LIMITATIONS OF THE SYSTEM THAT THE SYSTEM CONTROLLER IS TO CONTROL. THIS IS DONE WITH A "FIRST-CUT FLOW DIAGRAM" AND A "BASIC BLOCK DIAGRAM".
- ③ USING "TIMING-DIAGRAMS" DEFINE THE TIMING AND FREQUENCY OF THE SYSTEM LEVEL INPUT AND OUTPUT CONTROL SIGNALS. MAKE NOTE OF ANY TIMING CONSTRAINTS.
- ④ DETAIL THE SEQUENTIAL BEHAVIOUR OF THE SYSTEM CONTROLLER AND, AT THE SAME TIME, DETERMINE REGISTER, TEMPORARY STORAGE, SPECIAL CIRCUIT AND OTHER SUB-FUNCTION REQUIREMENTS FOR THE SYSTEM CONTROLLER AND THE OVERALL SYSTEM. ALSO DEVELOP DETAILED TIMING DIAGRAMS DEFINING THE SYSTEM LEVEL AND SUBFUNCTION CONTROL.
- ⑤ DEVELOP AN "MDS DIAGRAM" FOR THE SYSTEM CONTROLLER USING ① TO ④ AS A GUIDE.

6) CHOOSE A SYSTEM CONTROLLER ARCHITECTURE FROM ONE OF THE BASIC VARIATIONS

7) USE RULES TO MAKE STATE ASSIGNMENTS.

The final stages apply to SSI implementations.

8) DESIGN NEXT STATE DECODER LOGIC FOR THE FLIP FLOP OF YOUR CHOICE.

9) DEFINE CONTROLLING OUTPUT REQUIREMENTS AND DESIGN OUTPUT DECODER LOGIC FOR GLITCH-FREE OPERATION.

10) IN LARGE SYSTEMS: DESIGN CONTROL + DISPLAY FEATURES AND FAULT ISOLATION CIRCUITRY.

DEFINING THE PURPOSE AND ROLE OF THE SYSTEM

To design a system controller a designer must first have an intimate knowledge of the system that is to be controlled.

As an example consider the design of a DATA-LOGGER system.

Several questions need to be answered:

- ① What is a data-logger and what does it do?
How does it do it?
- ② Why do you want a data-logger?
- ③ What kind of medium is the logger to be on, and how big are the "logs"?
- ④ What controls the data-logger?
- ⑤ What are the characteristics of this device?
- ⑥ Where do you want the data-logger?
- ⑦ When do you want the data-logger?

The answers to these questions will enable to create a bare bones block diagram of the system and its surroundings.

Example
Specifications for a balloon flight data-logger.

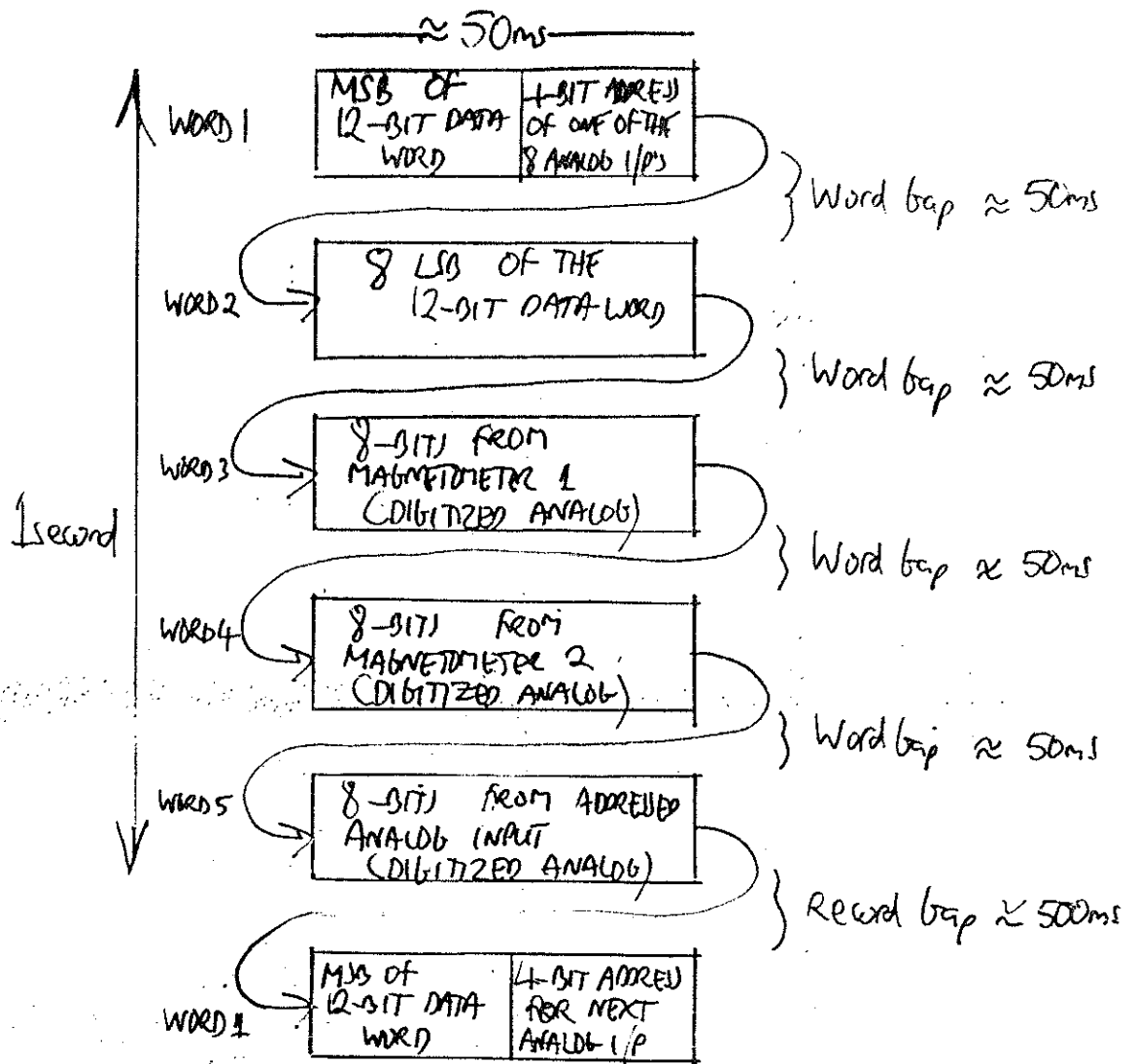
Design a data-logger (REAL TIME DATA RECORDING SYSTEM) for a high altitude (130,000 ft) balloon flight.

The data-logger receives 12 parallel lines of digital data from a special digital system on board called a photon counter.

Also the logger receives 10 lines of analog data from special atmospheric monitoring transducers. This analog data must be digitized and recorded on a single track of a cassette tape, together with the 12 bits of digital data.

A complete data record is to be made once each second and recorded in 8-bit serial words on the tape.

The diagram shows the bit formatting.



FIVE 8-bit words (40 bits) make up a record and each record includes the data for just one of the 8 out of 10 analog inputs that must be digitized. So it takes 8 records to make a file of all 8 multiplexed analog inputs. Note the timing in the chart. Each word takes 50ms to record and then a 50ms space is generated. This is repeated for 5 words. A 500ms space indicates a record gap.

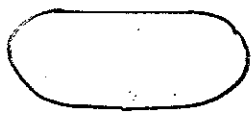
The system starts by a START signal from the balloon system or will continue until commanded to stop (STOP).

BASIC OPERATIONAL CHARACTERISTICS OF THE SYSTEM

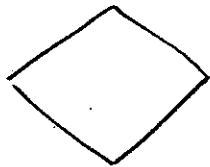
Now the designer must start to define the system level control functions and start a bare-bones block diagram for the complete system.

Then we develop the "FIRST CUT FLOW DIAGRAM" and "TIMING DIAGRAMS". You must ensure that no timing or sequential incompatibilities exist.

Flow diagram symbols are as below:



TERMINAL



DECISION
(YES OR NO)

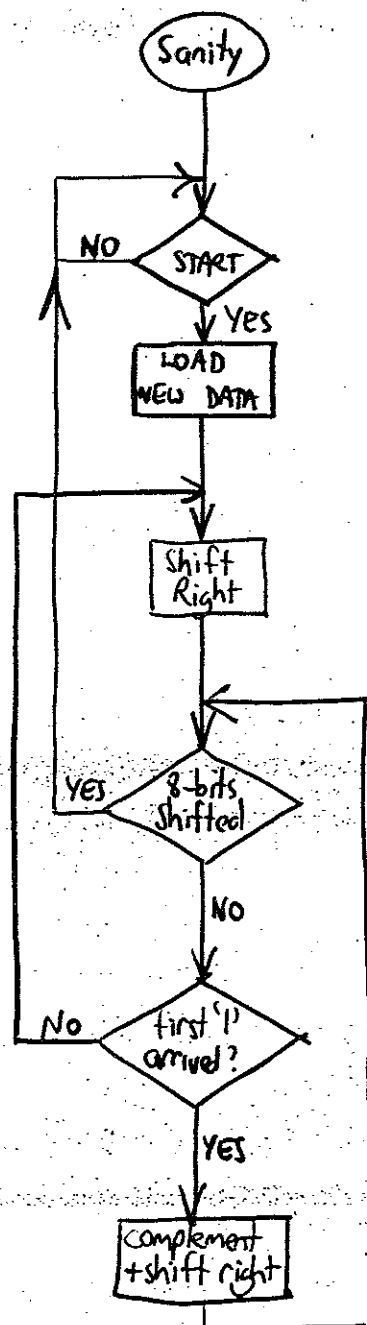
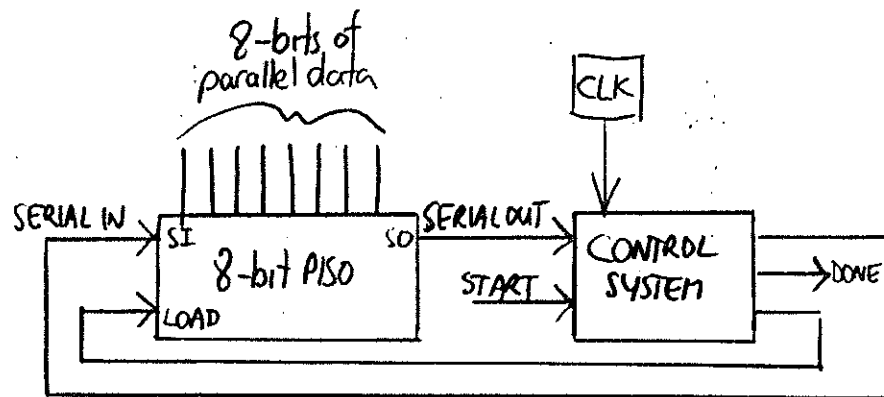


ACTION/PROCESS

Example

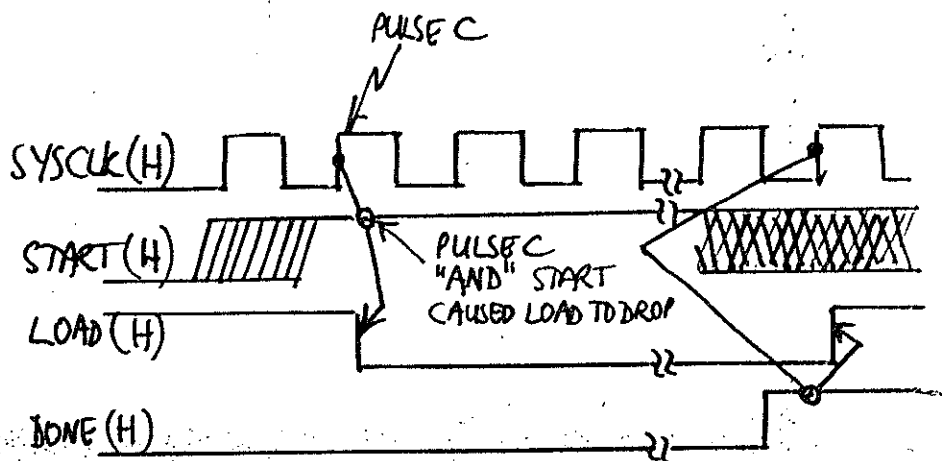
Design a flow diagram for a system that loads a register and sequentially replaces the contents of the register with its 2's complement. You are given the bare-bones block diagram and the first cut flow diagram.

Bare bones block diagram



TIMING AND FREQUENCY CONSIDERATIONS

The 2's complement has the following timing diagram:



TIMING DIAGRAM NOTATION



SIGNIFIES SIGNAL WILL BE CHANGING FROM A HIGH TO A LOW VOLTAGE



SIGNIFIES "DONT CARE" CHANGING VOLTAGE PERMITTED



--- LOW TO A HIGH VOLTAGE



SIGNIFIES "AND"

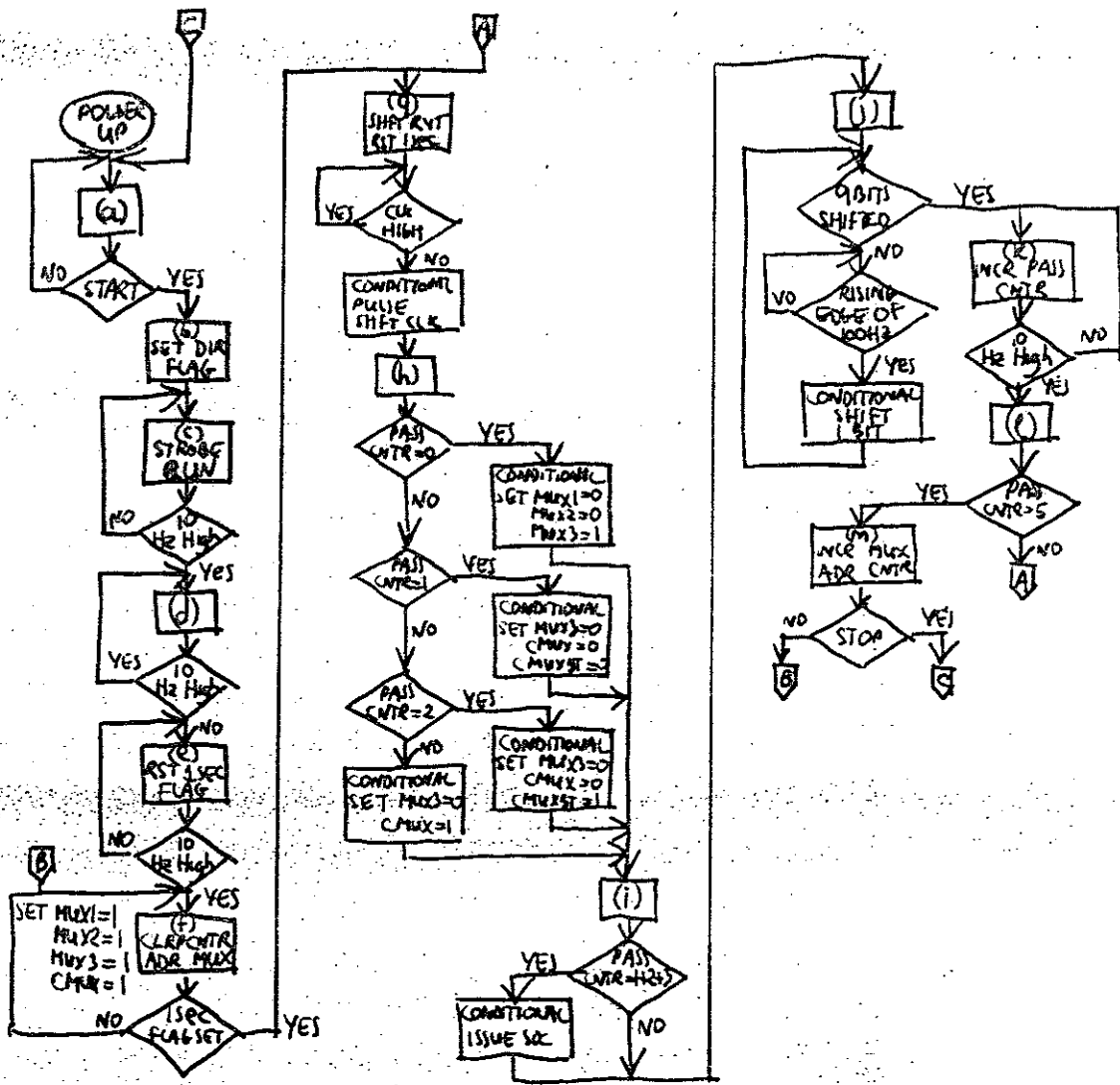
FUNCTIONAL PARTITION AND DETAILED FLOW DIAGRAM

You now need to consider the system controller:

A DETAILED FLOW DIAGRAM in conjunction with

a FUNCTIONAL PARTITION (BLOCK DIAGRAM)

Along with these a detailed timing diagram is needed



DETAILED FLOW DIAGRAM FOR DATA LOGGER

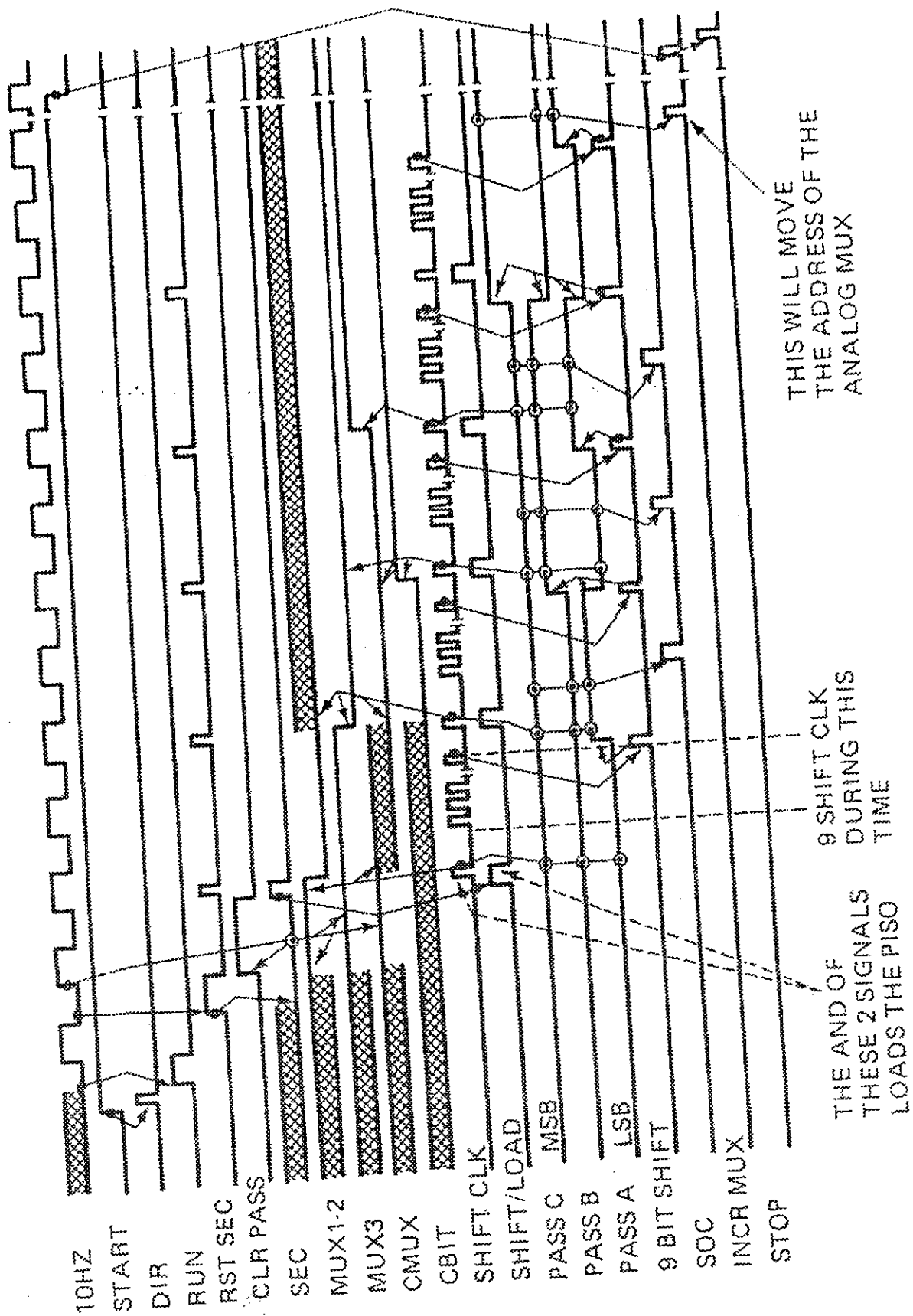


Fig. 7-6(b). Detailed timing diagram for data logger.

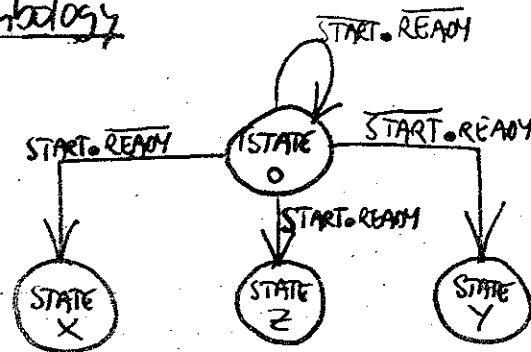
SYSTEM CONTROLLER STATE SPECIFICATION

To specify the system controller hardware a number of methods exist, two such techniques are:

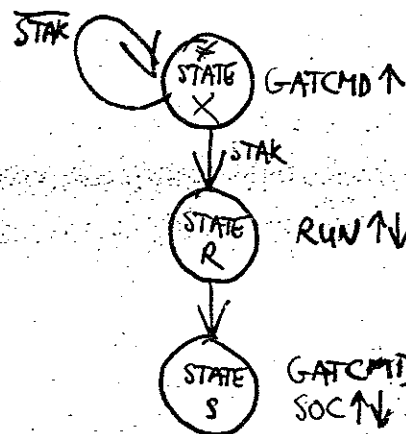
- ① ALGORITHMIC STATE MACHINE (ASM) CHART
- ② MNEMONIC DOCUMENTED STATE (MDS) DIAGRAM.

The MDS diagram method will be used here.

MDS Diagram Symbology



(a) FOUR-BRANCH STATE



(b) OUTPUT CONTROL GENERATION

MDI diagrams are state diagrams with the transition governed by equations instead of simply 1's and 0's in the familiar Input/output format.

(a) show a simple diagram while (b) introduces extra terminology.

↑ is ASSERT with no voltage level implied

↓ is defined as DE-ASSERT (negate) with no voltage level implied.

* Mean branching from this state is controlled by an asynchronous variable.

Consider (b).

① The * in state X means that branching from X is controlled by a variable that may or may not be synchronised to the system clock.

② GATCND ↓ associated with X means that GATCND is asserted upon entrance into X. It must be DE-ASSERTED (↓) either as it leaves the state or elsewhere. IN this case upon entry to S.

3) State R has RUN $\uparrow\downarrow$ associated with it.
RUN is ASSERTED($\uparrow$) on entry to R and DE-ASSERTED($\downarrow$) when leaving R. This is a state pulse of width equal to the time spent in the state.

4) STATE S has:

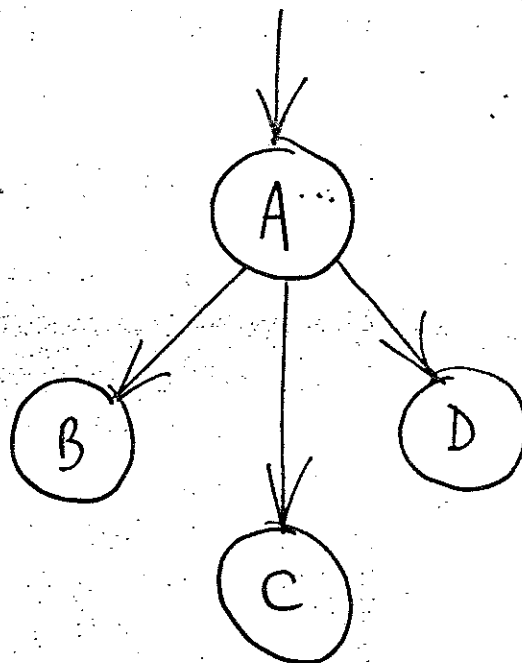
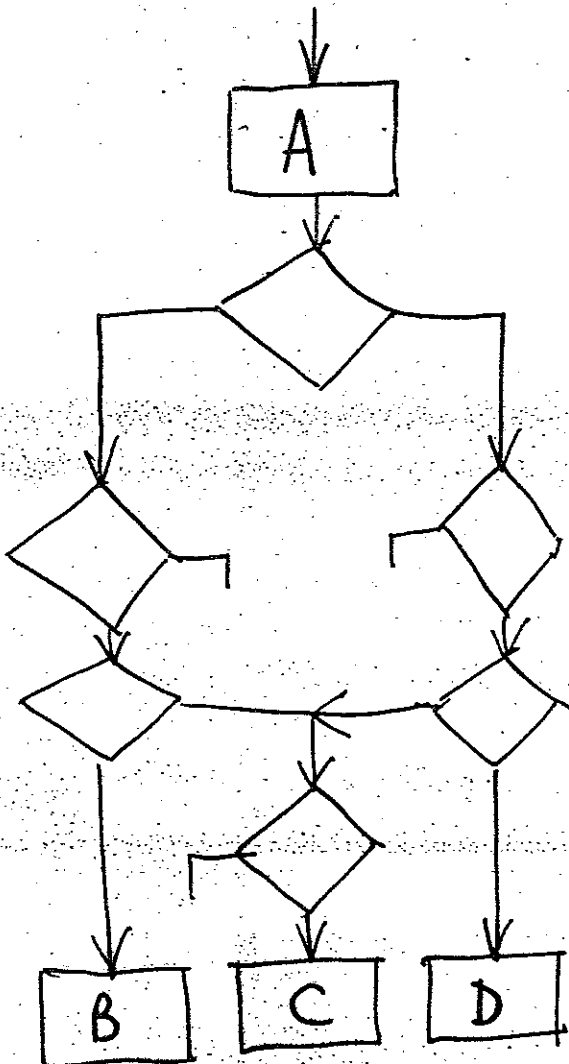
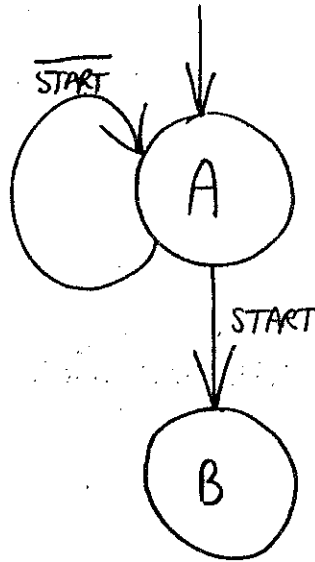
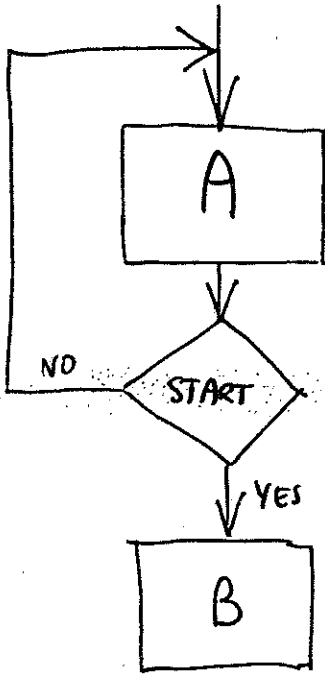
$$SOC \uparrow\downarrow = (StateS \cdot EOC \cdot \overline{CLK})$$

this is used for "conditional outputs".

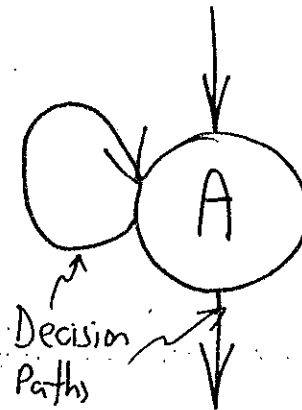
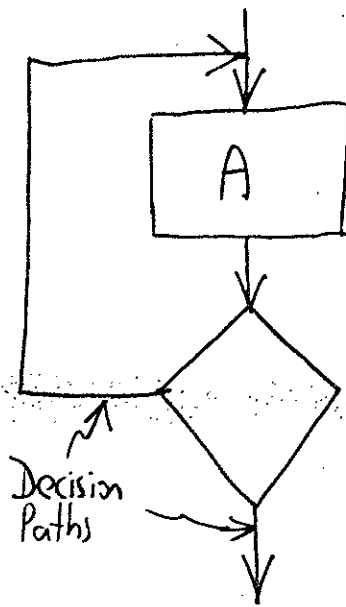
The output SOC is asserted conditionally in S;
the conditions are EOC must be asserted and
the inactive period of the clock must be present.

ADAPTING FLOW DIAGRAMS FOR MDS DIAGRAMS

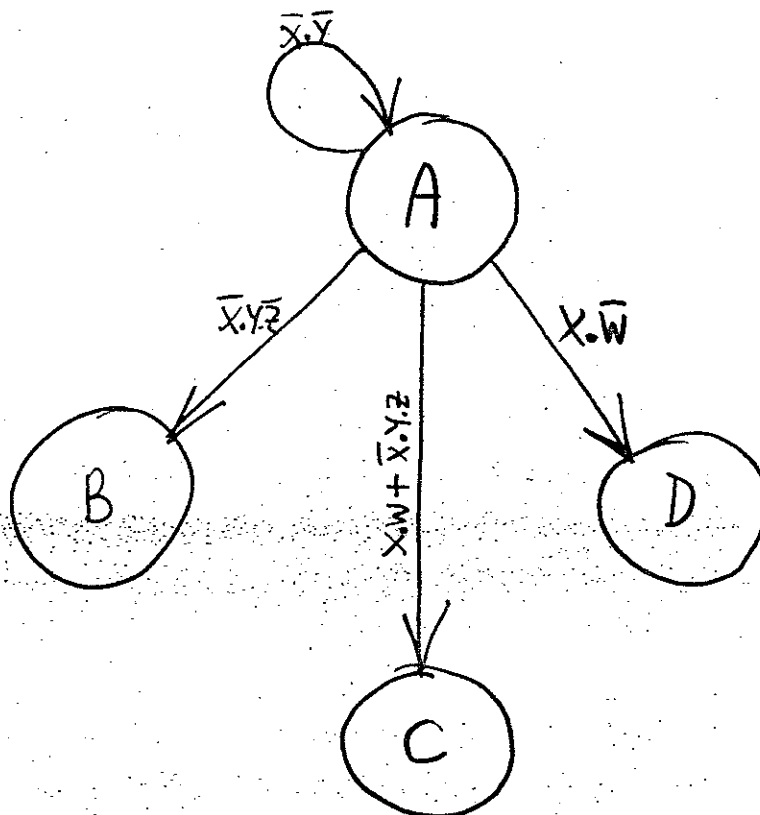
Concept 1: An action block in a flow diagram defines a state in an MDS diagram



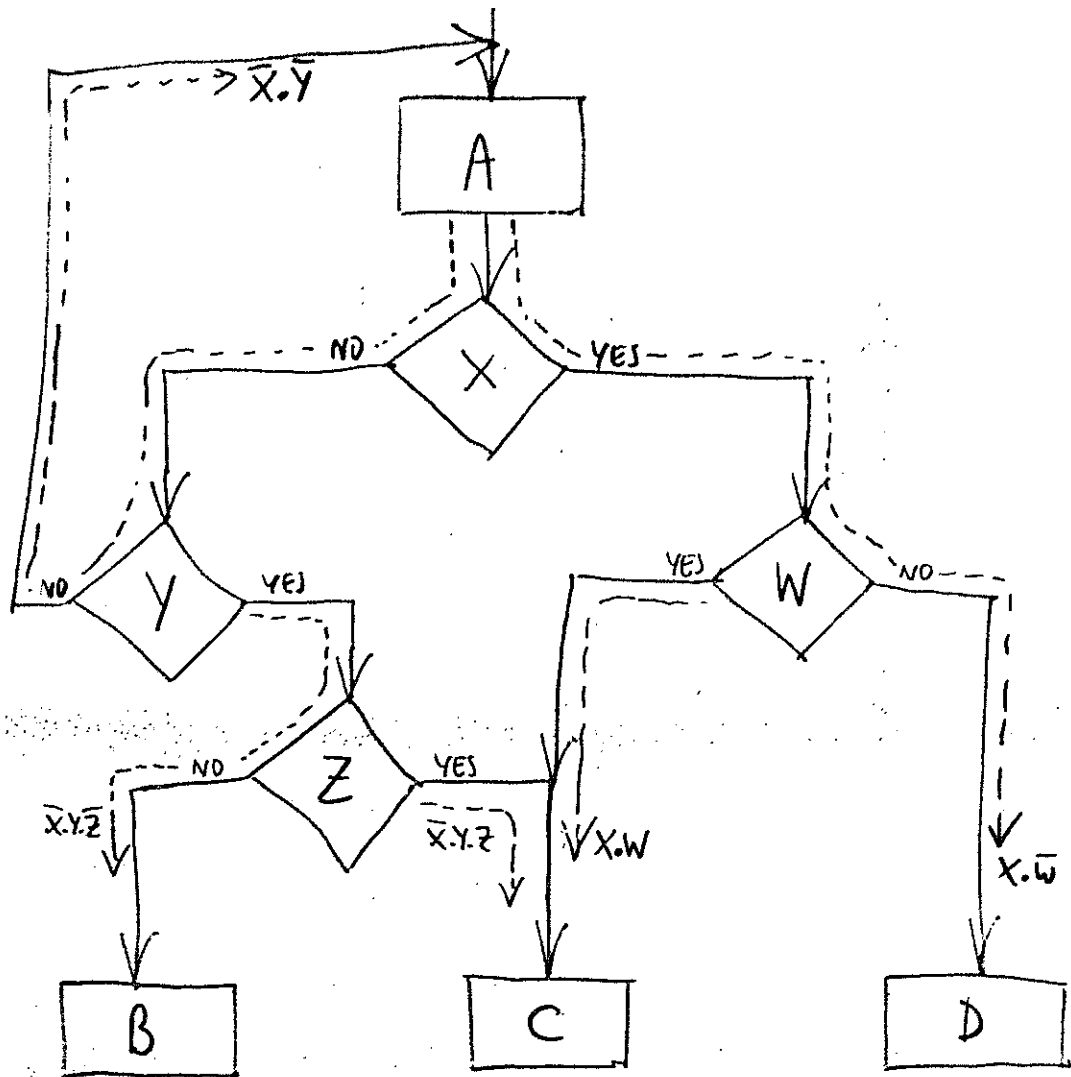
Concept 2: The branching conditions (branching expressions) for a state in an MDS diagram are derived by tracing the decision paths from the state definition action block to all other action blocks.



BRANCHING
CONDITIONS

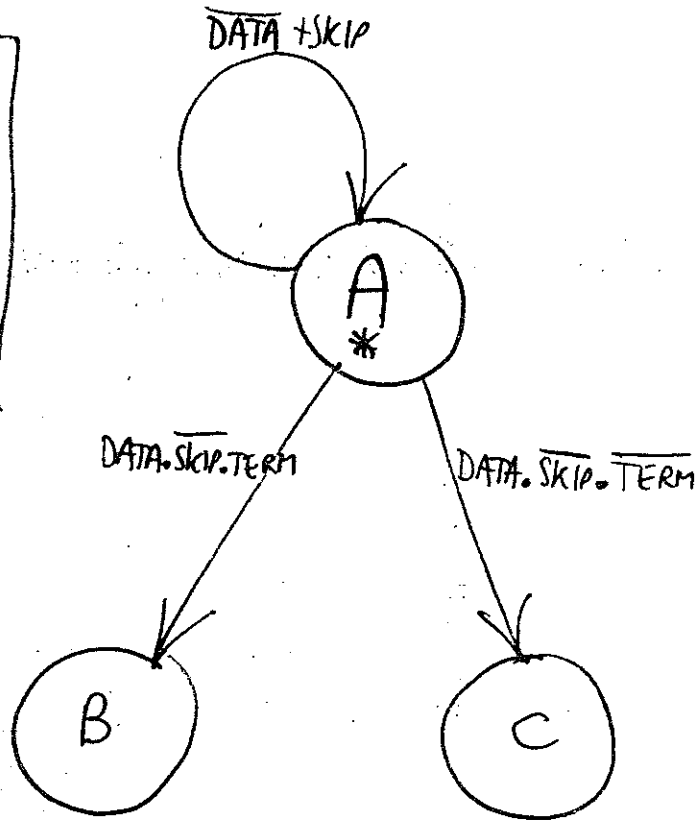
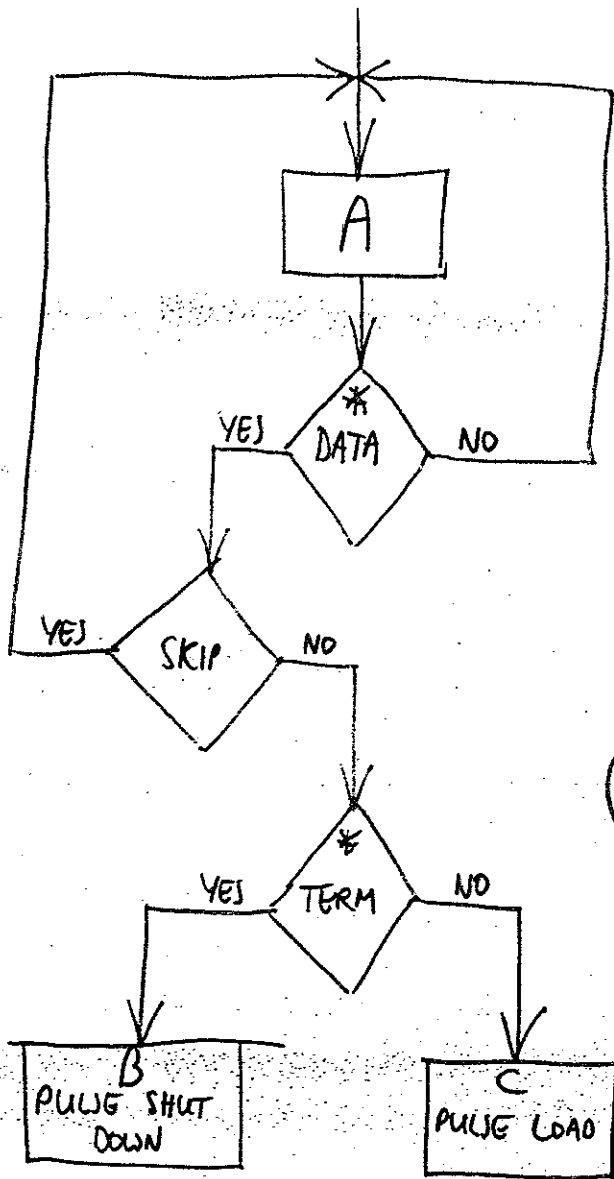


MDS DIAGRAM
FOR FOLLOWING
FLOW DIAGRAM

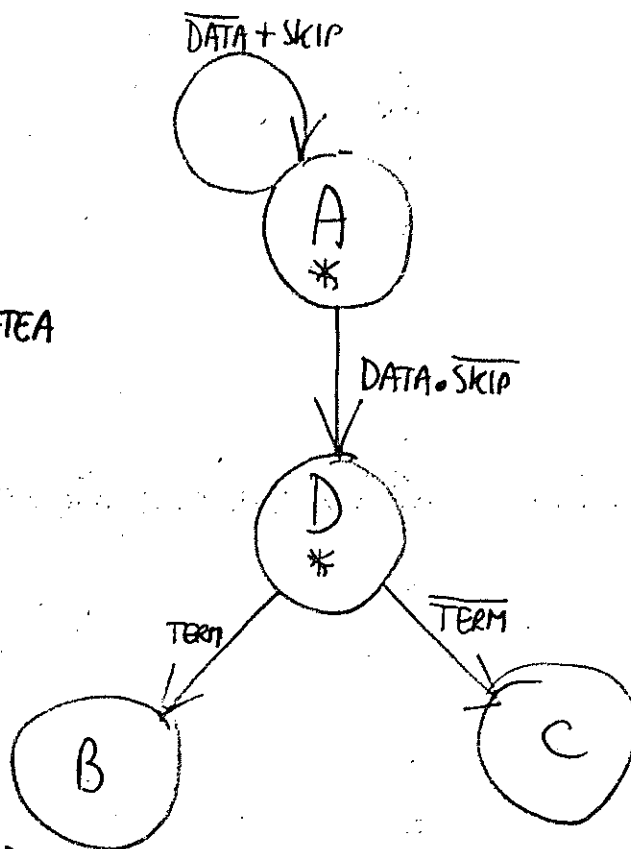
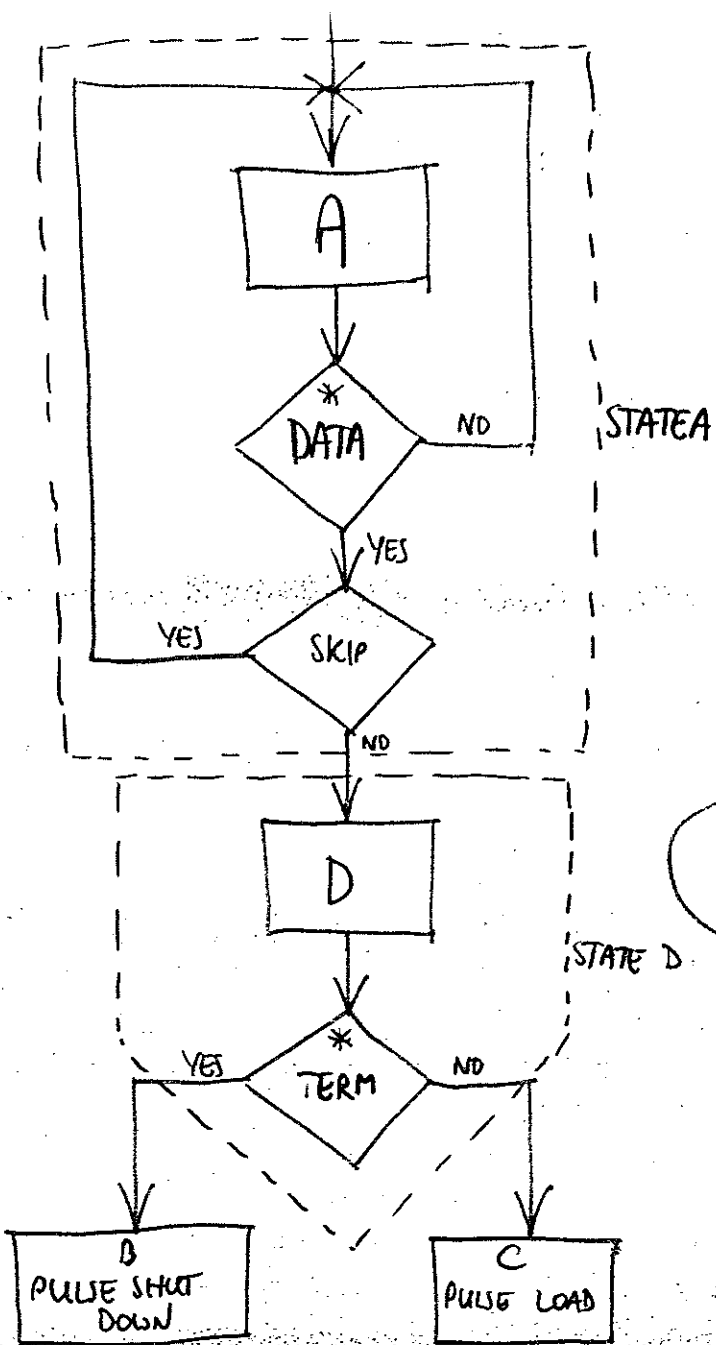


Concept 3: Avoid making branching decisions on more than one purely asynchronous decision variable.

The following flow and MDI diagrams have two variables independent of each other and our control (asynchronous)

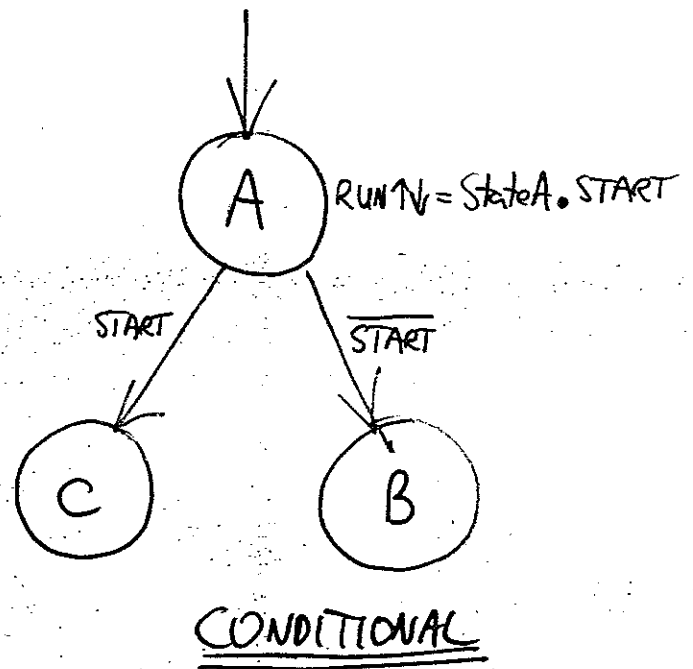
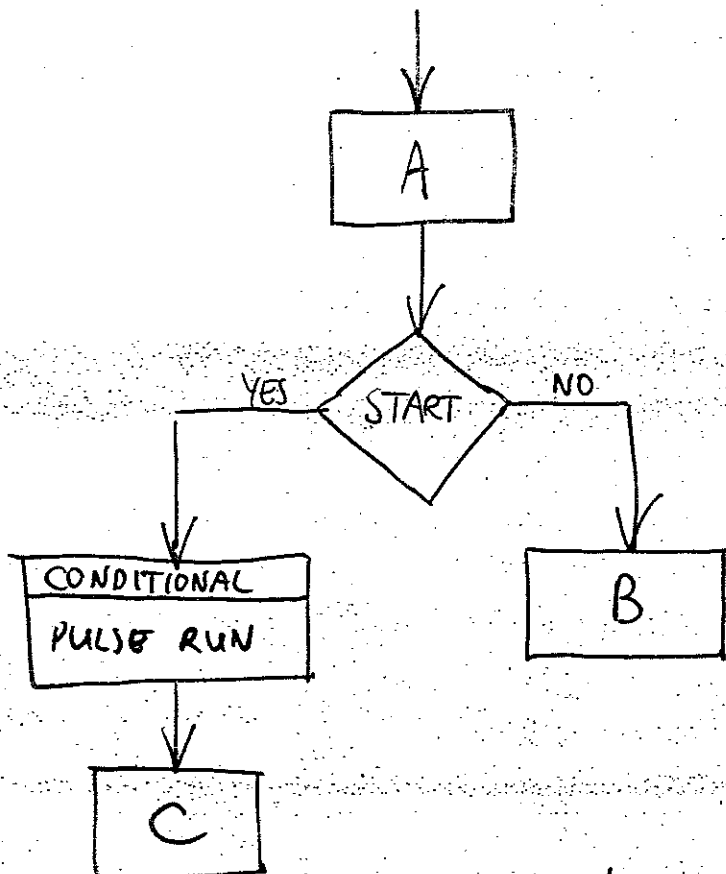
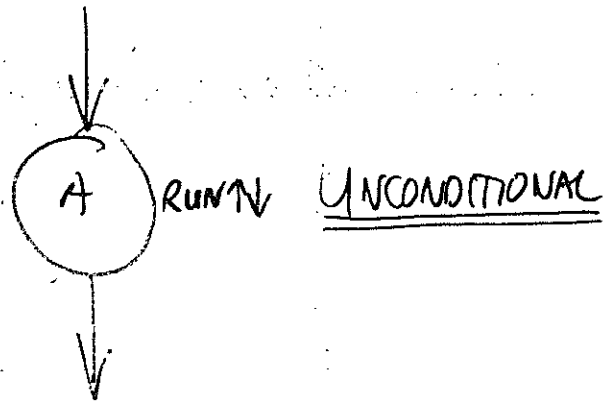
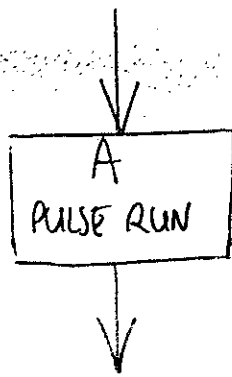


The correct method is to define a new state:



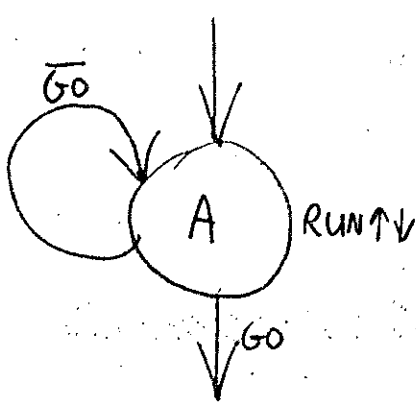
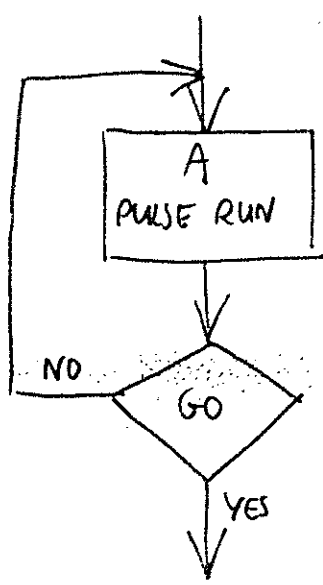
Concept 4: Generated outputs are symbolized by action notation in the action blocks:
Outputs are defined as:

- ① Unconditional (immediate)
- ② Conditional (input dependent)

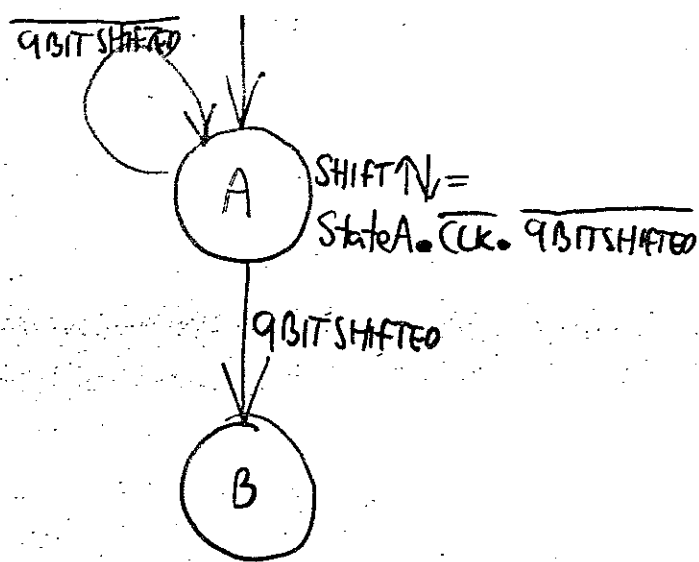
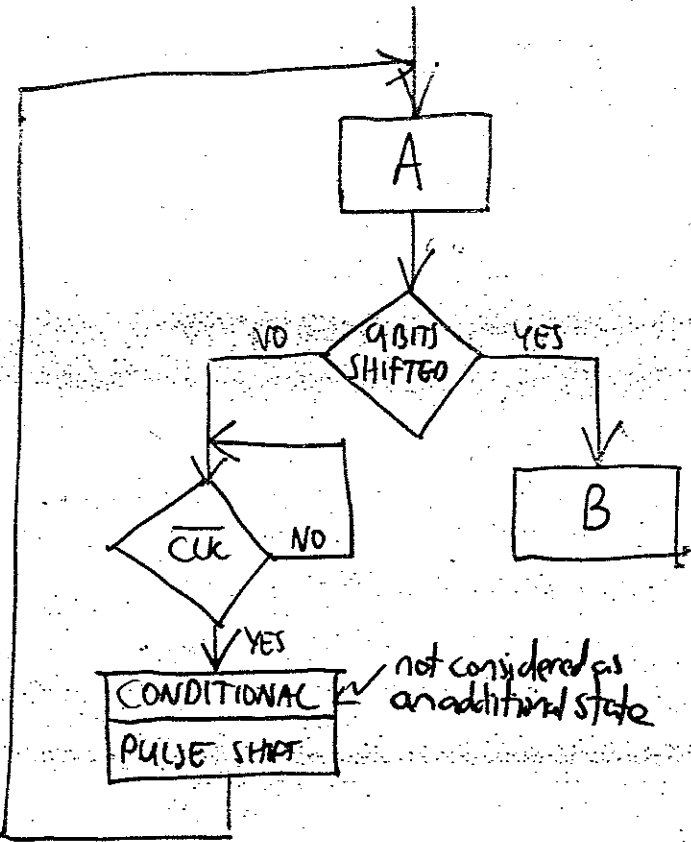


WARNING: Never generate a conditional output on an asynchronous variable it could be changing during the generation of the output.

Concept 5: Unconditional and conditional outputs can be specified with duration time dependent on a decision variable, that is ASSERT and WAIT for response.



Unconditional Output With an Input Time Duration Dependency



Conditional Output with an Input Time Duration Dependency

SYNCHRONIZING TWO SYSTEMS

The design techniques discussed depend the 'marriage' of independent sub systems.

This requires the synchronization of:

- ① sub-systems
- ② controlled systems
- ③ controlling system with the system controller

This can be a big problem so we begin with a simple definition.

The synchronization problem is simply one of the system controller knowing when or within what time frame it can expect input changes from its controlled and controlling systems.

This simple definition does not mention the system clock but this is critical in such tasks.

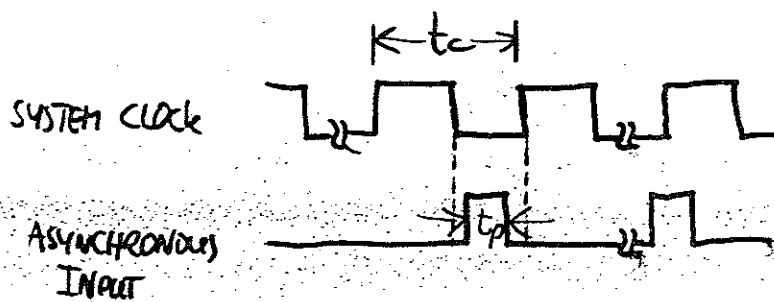
Improper synchronisation can lead to two faults:

- ① INPUT CONTROL CHANGE COULD BE "MISSED" AND HENCE IGNORED BY THE SYSTEM CONTROLLER
- ② The INTRODUCTION OF UNDEFINED STATE TRANSITIONS.

Consider FAULT ① where the input is ASSERTED for a short time relative to the system clock. By short we mean t_p does not satisfy:

$$f_c > \frac{1}{t_p}$$

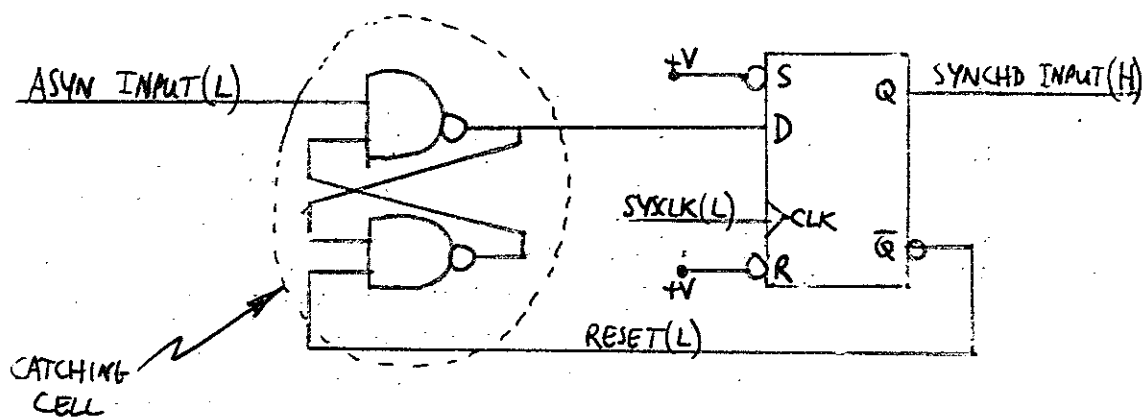
where f_c is the system clock frequency and t_p is the input pulse width.



There may be cases when it is impractical to increase the clock frequency to meet this criterion and so we must "catch" the pulse and hold it until the system controller can service it. If this can be done then the system is said to be SYNCHRONIZED.

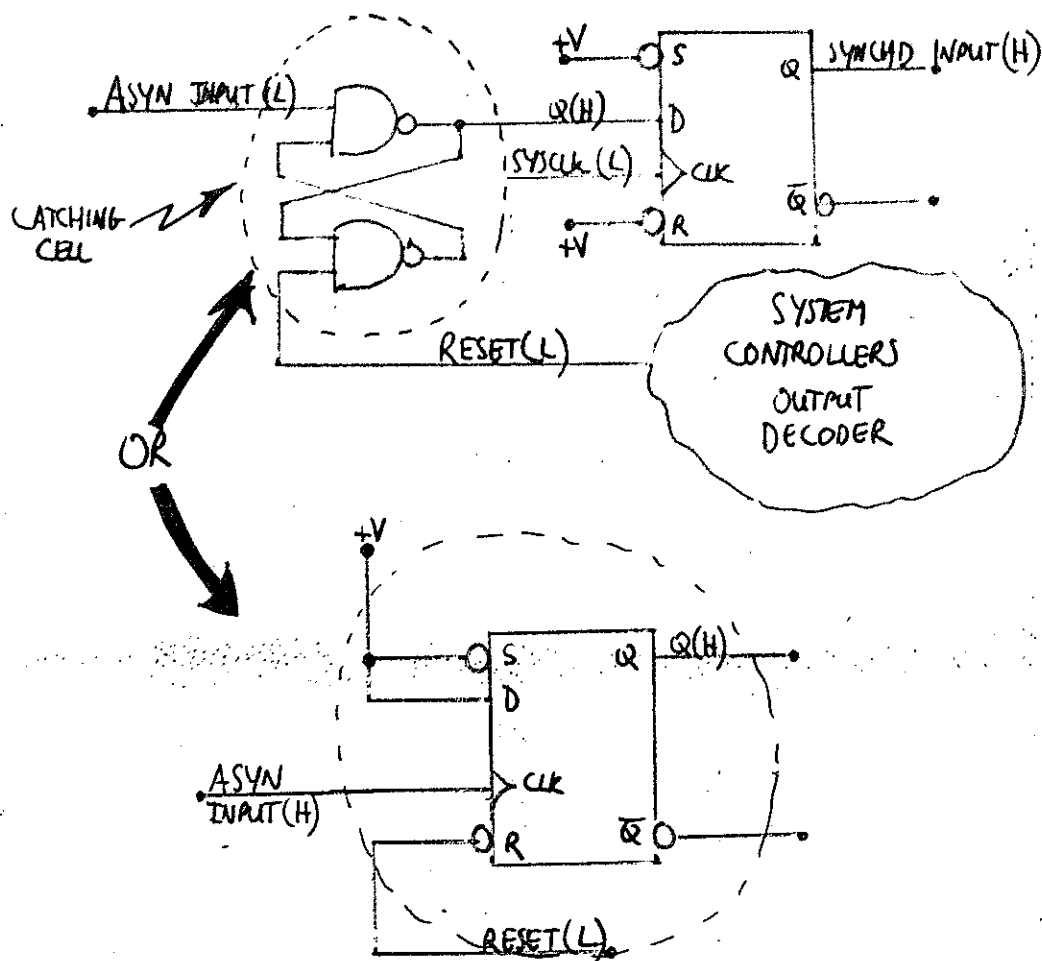
There are a number of techniques for this all dependent on:

$$f_p < f_c$$



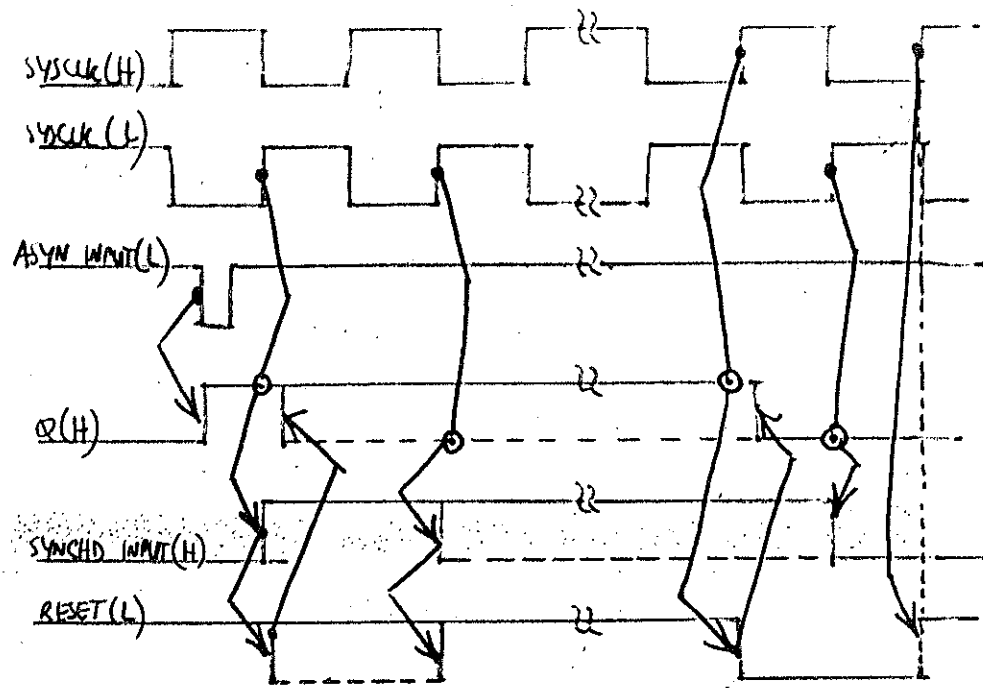
This network will catch the short asynchronous pulse and hold it until the falling edge of SYCLK. It is then transferred to the output of the D-Type flipflop and held for one more complete system clock cycle.

Note: the setting of the D-Type resets the cell.



The above circuit has advantages because of a controlled reset of the catching cell

Timing Diagram



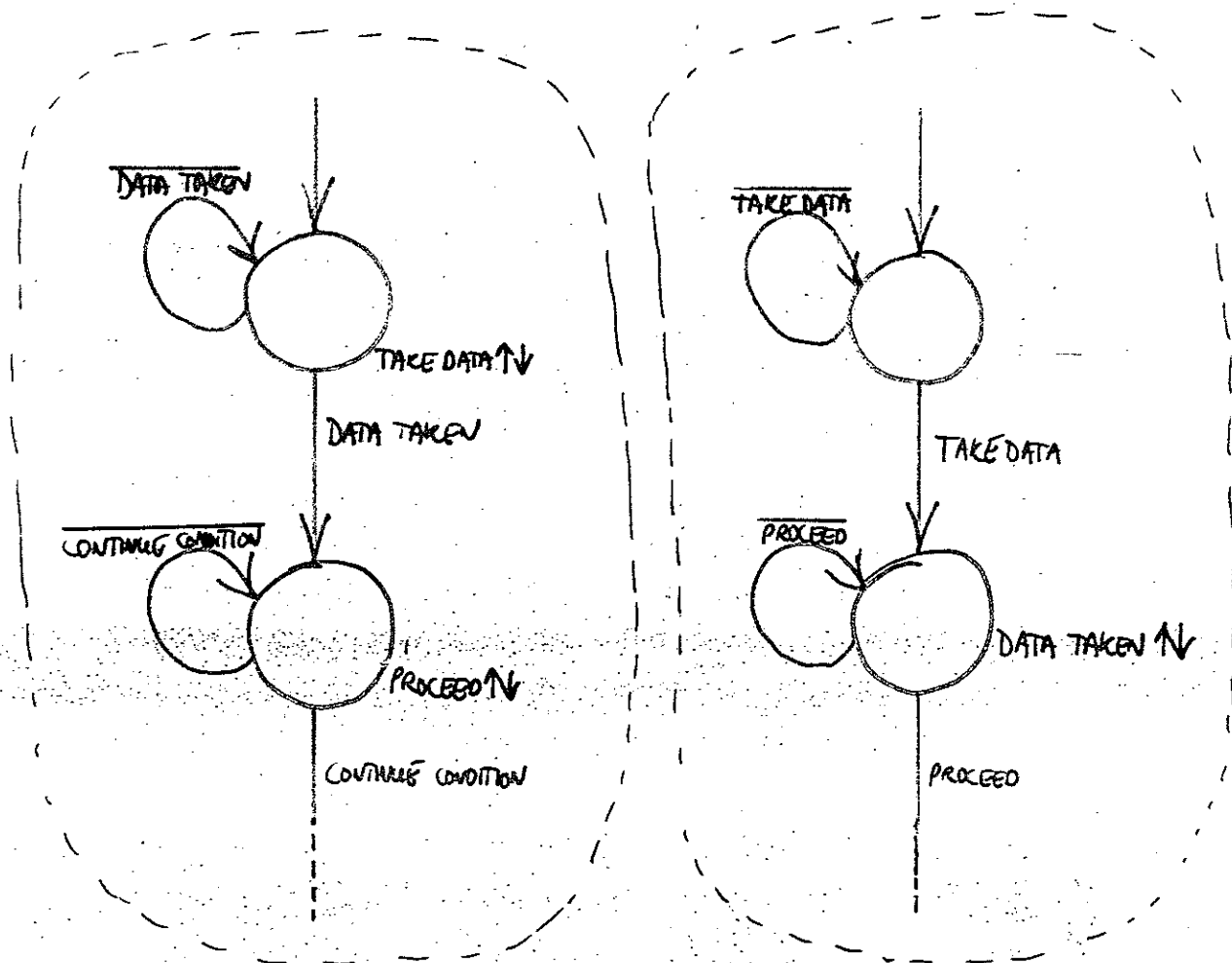
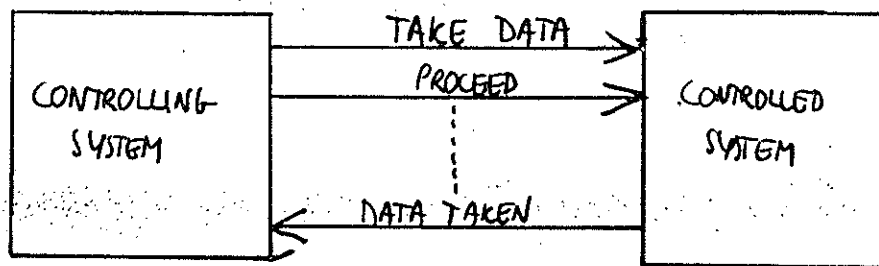
Another synchronization method is needed when

$$t_p \gg t_c$$

This is called 'level synchronization'. The input is defined as a catching level, but transition take place asynchronously with respect to the system clock. The same circuits are used but without the "catching cell". This locks an asynchronous input with the falling edge of the system clock.

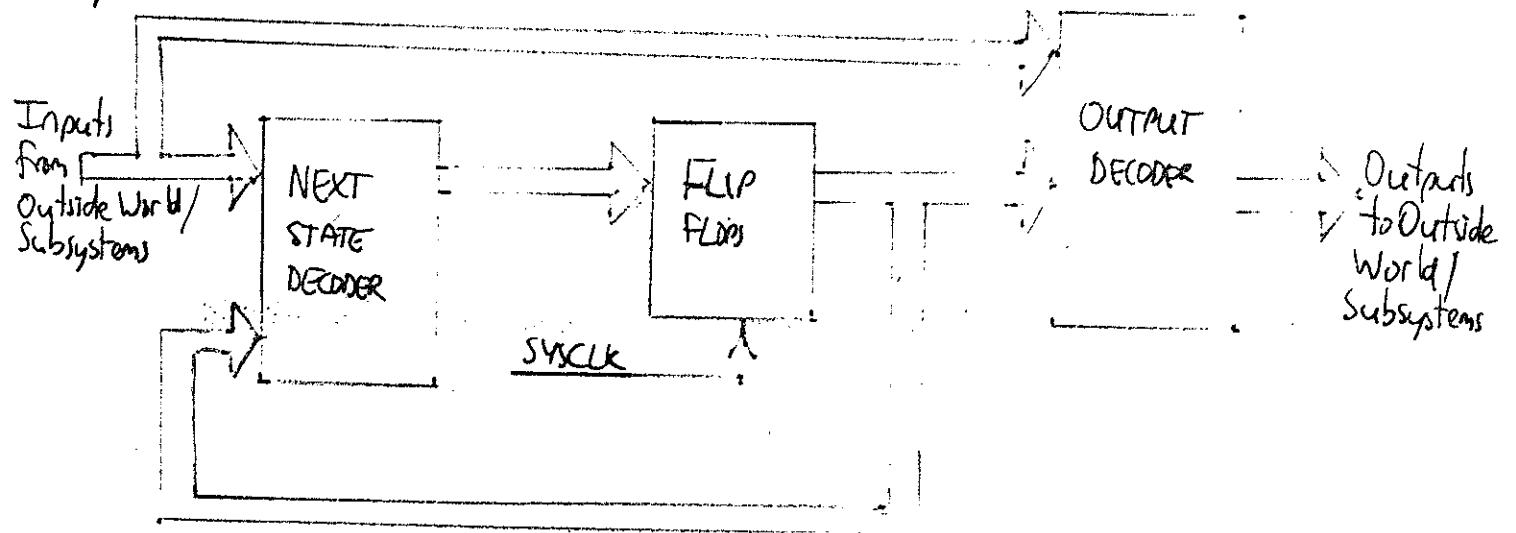
The Handshake

The handshake is a sort of mutual agreement to synchronize. The two systems will treat the others inputs as asynchronous.

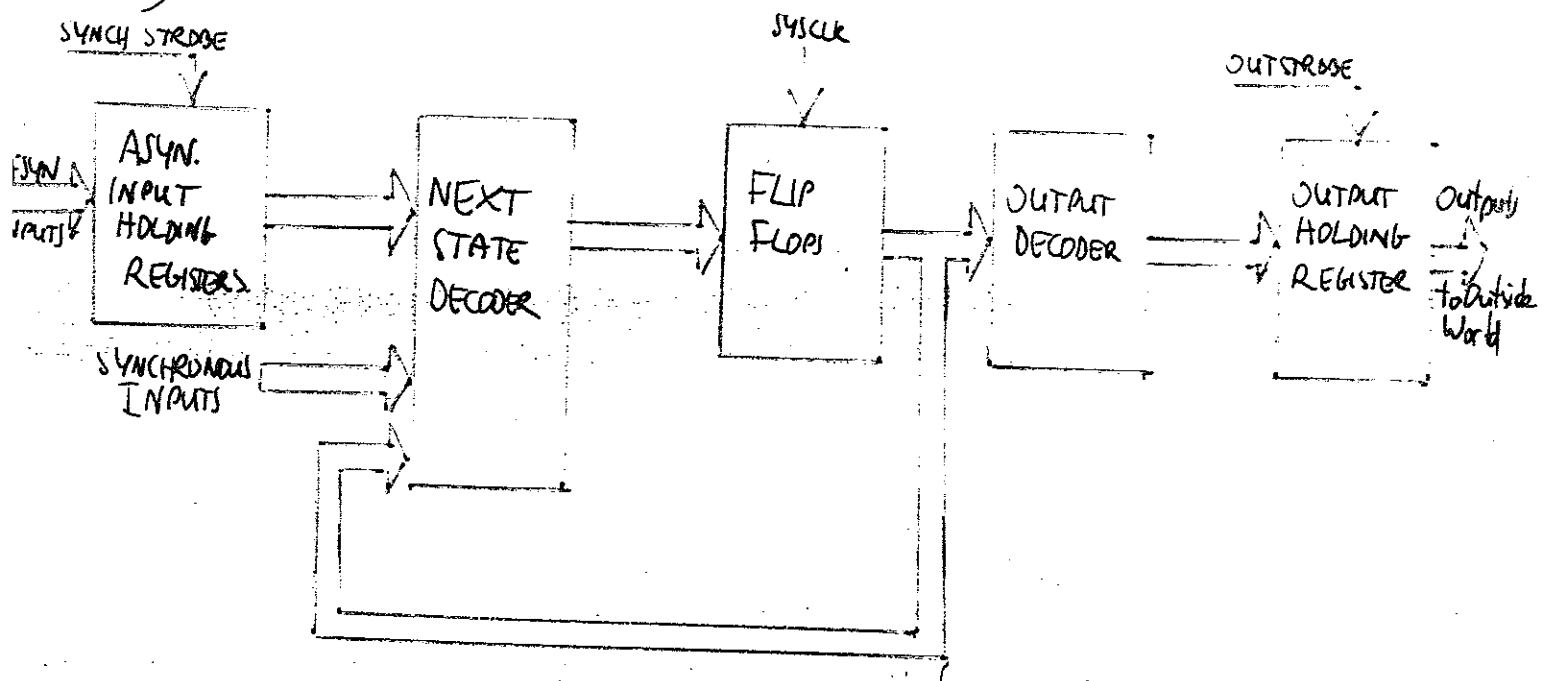


CHOICE OF CONTROLLER ARCHITECTURE

The addition of several holding registers can improve the practical performance of the fundamental or classic sequential circuit shown below:



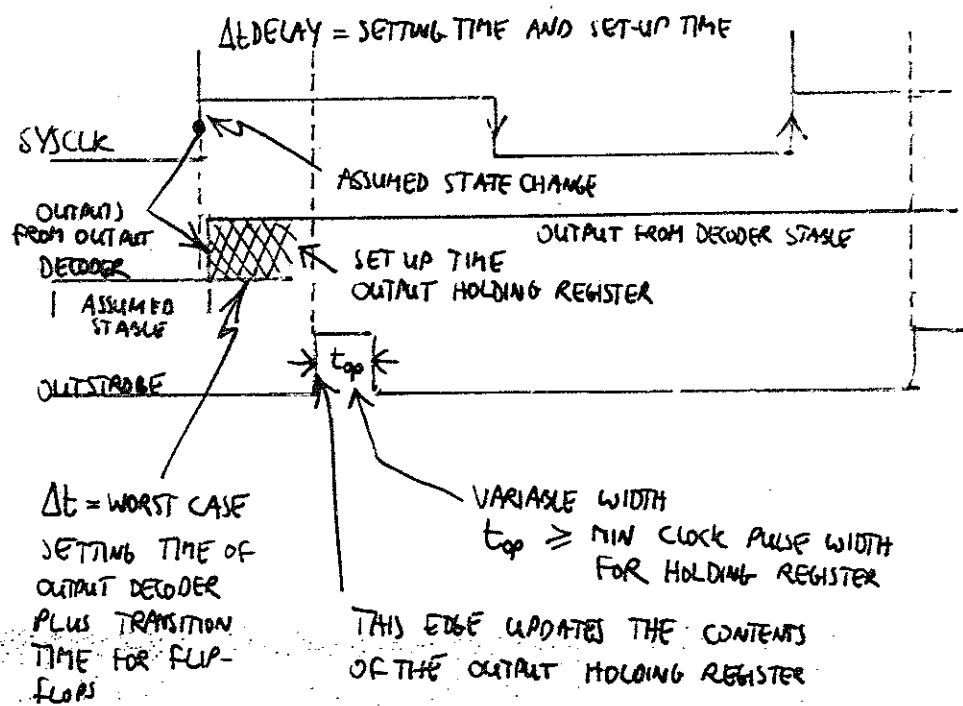
An output holding register and an asynchronous input holding register could be used.



GLITCH-FREE OUTPUT GENERATION

The output holding register is used to remove "glitches" from signals emerging from the output decoder. A glitch is an unwanted "spike" or "transient" that can increment counters, ^{clear} registers or start unwanted processes.

When using an output holding register the register must be updated after each state change.



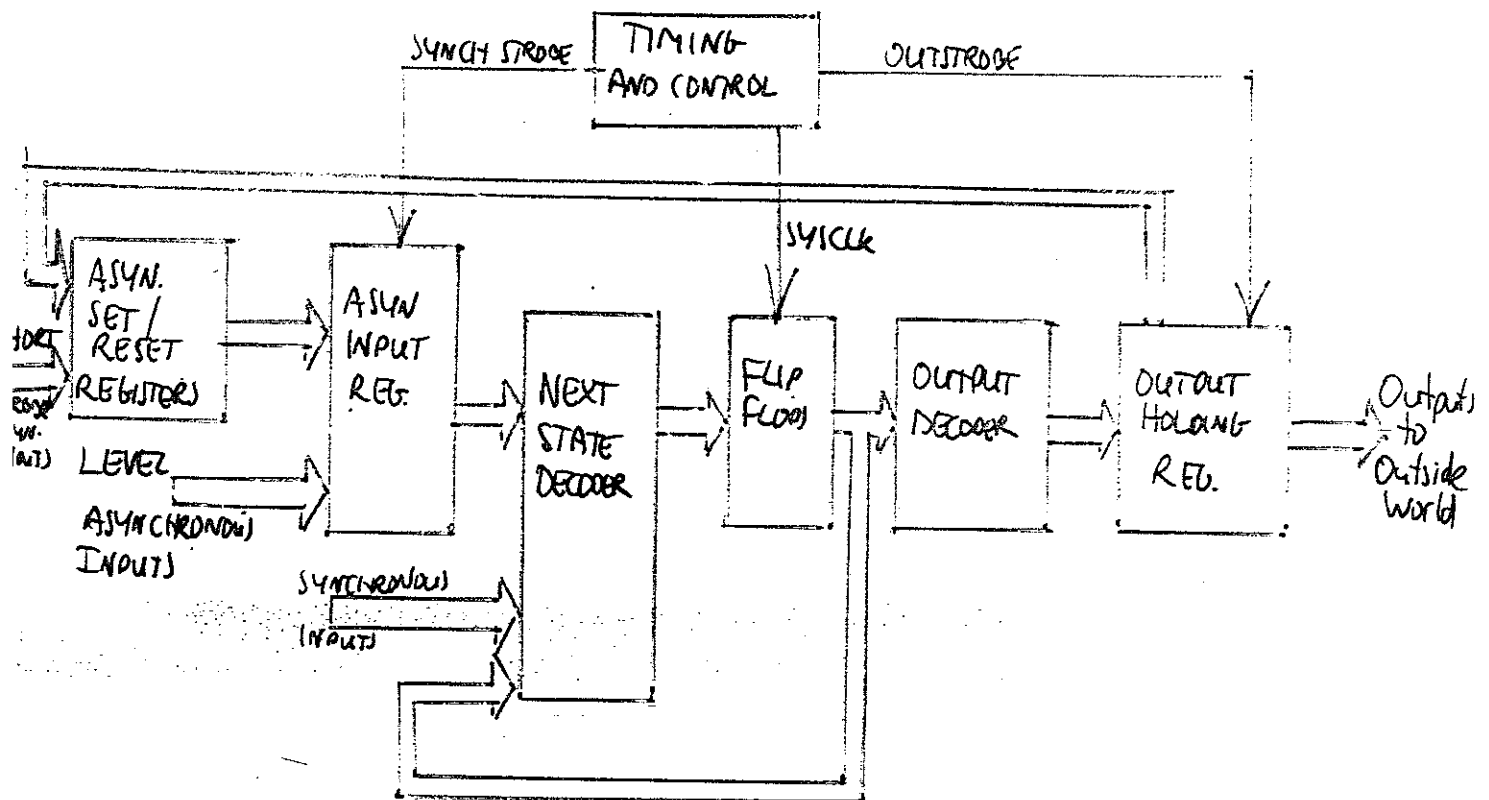
The diagram shows how the properly timed outstrobe signal will load the output holding register and guarantee glitch-free outputs.

More hardware is needed and the output changes are delayed by Δt but we now have guaranteed operation.

The ASYNCHRONOUS INPUT HOLDING REGISTER is the collection of all the D-Type flip flop needed to synch-in the ^(changing level) asynchronous inputs.

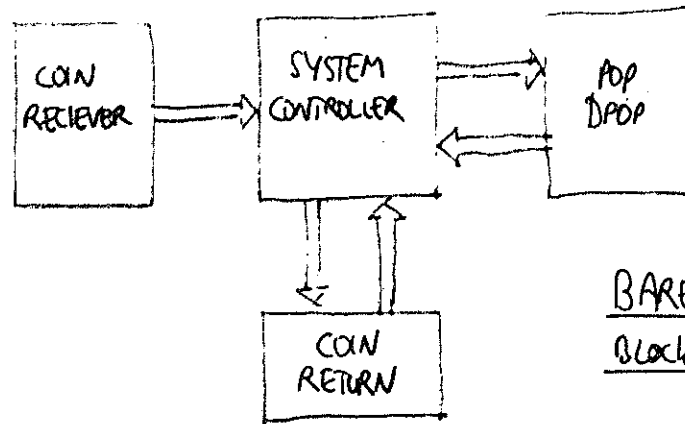
The signal SYNC STROBE is rather than a phase of sysclk to show that any properly chosen pulse train can be used to synchronize the level asynchronous inputs.

The following sequential machine has asynchronous SET/RESET registers.

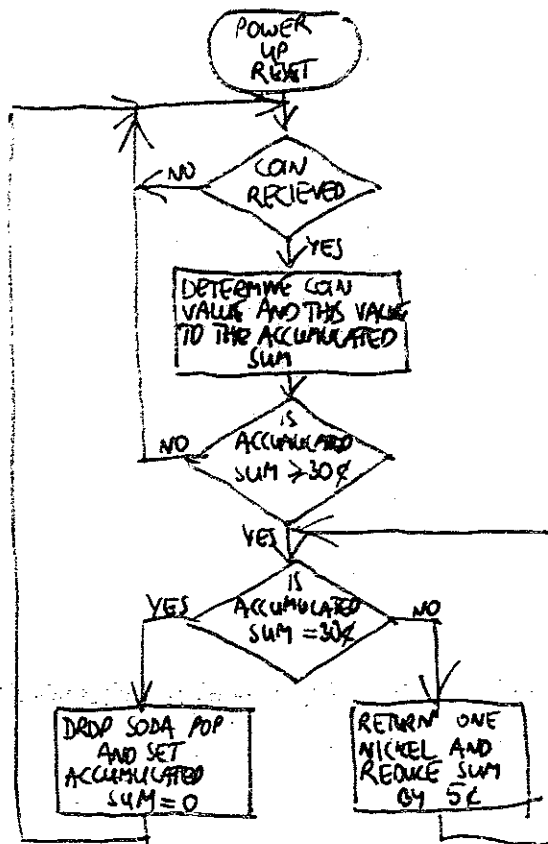


This is the set of cells for catching short input pulses.

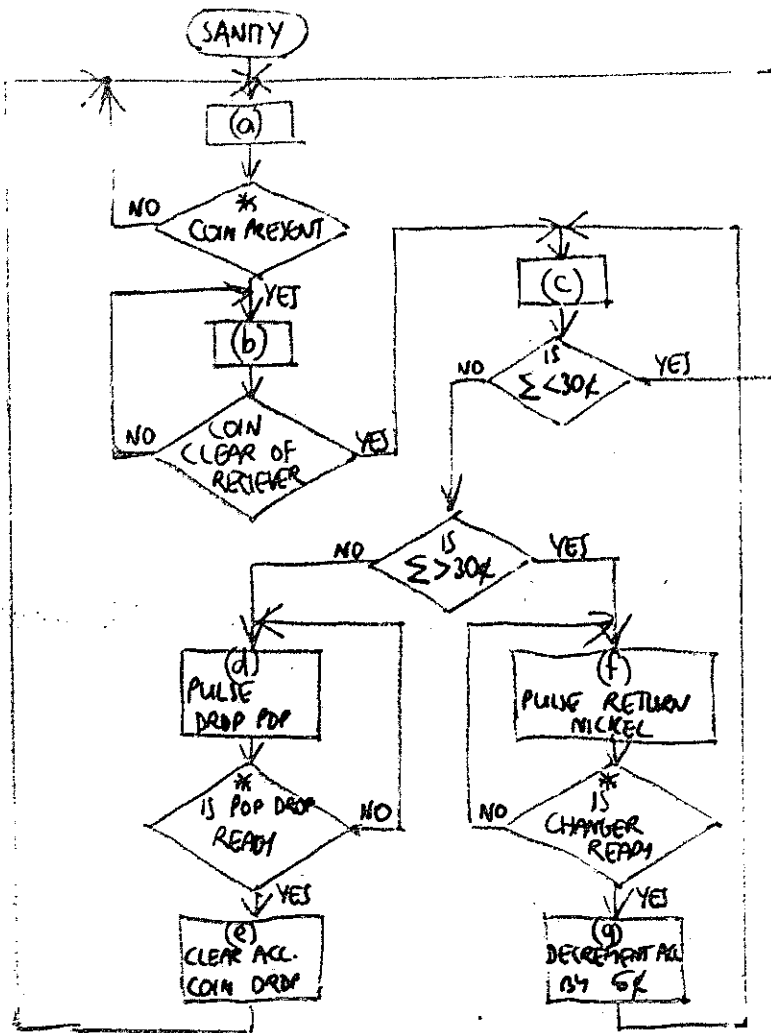
WORKED EXAMPLE: POP MACHINE CONTROLLER



BARE BONES
BLOCK DIAGRAM

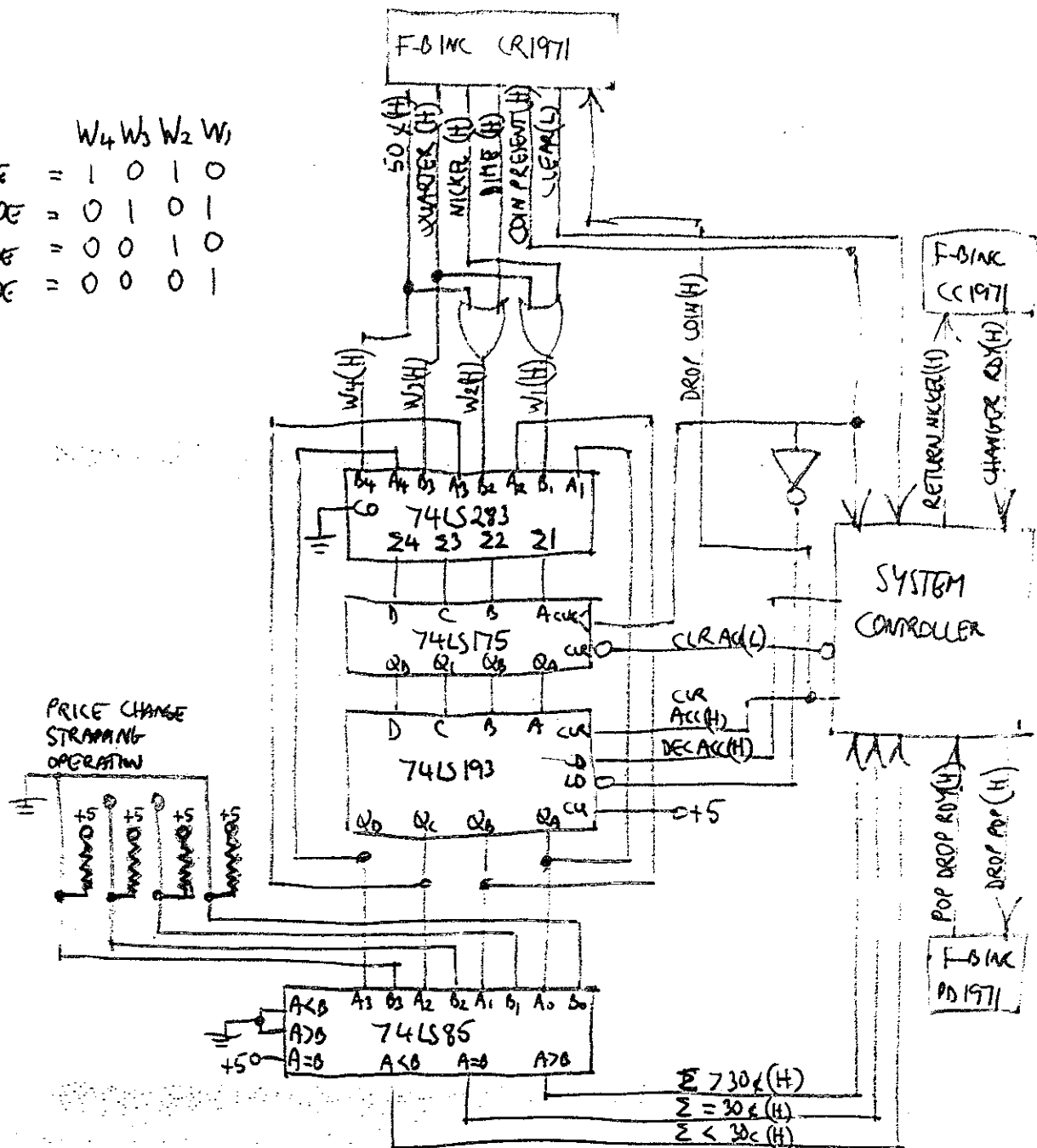


FIRST CUT
FLOW DIAGRAM



DETAILED
FLOW
DIAGRAM

	W_4	W_3	W_2	W_1
50¢ CODE	= 1	0	1	0
25¢ CODE	= 0	1	0	1
10¢ CODE	= 0	0	1	0
5¢ CODE	= 0	0	0	1



FUNCTIONAL
PARTITION

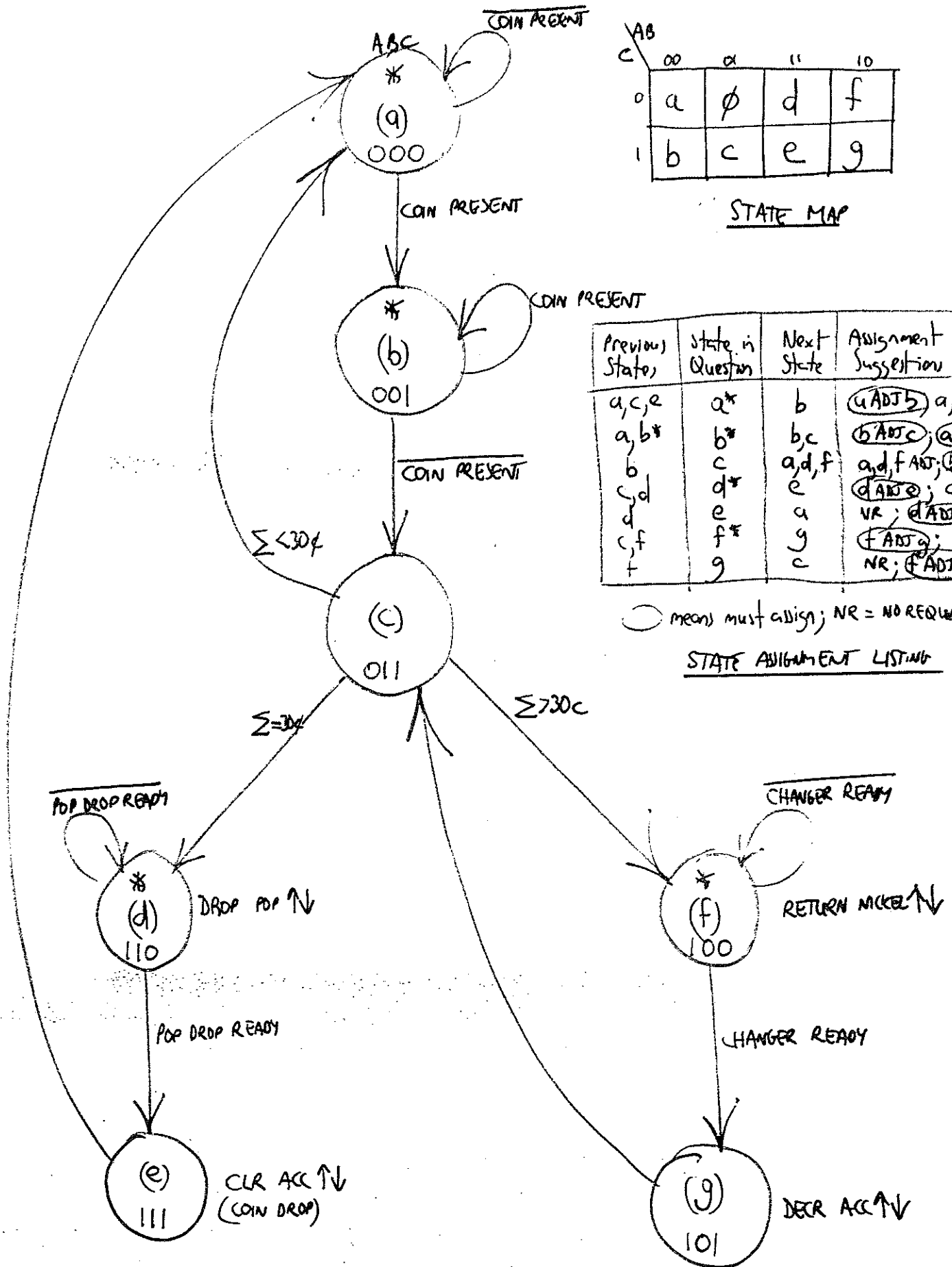
AB				
	00	01	11	10
0	a	∅	d	f
1	b	c	e	g

STATE MAP

Previous State,	State in Question	Next State	Assignment Suggestion
a, c, e	a*	b	(a ADJ b) a, c, e ADJ
a, b*	b*	b, c	(b ADJ c) (a ADJ b)
b	c d*	a, d, f	a, d, f ADJ; (b ADJ c)
c, d	d*	e	(d ADJ e) c ADJ d
d, f	e f*	a	NR; (d ADJ e)
f	f*	g	(f ADJ g) c ADJ f
	g	c	NR; (f ADJ g)

○ means must assign; NR = NO REQUIREMENT

STATE ASSIGNMENT LISTING



MDS DIAGRAM

Next State Maps for D-Type Implementation

AB \ C		00	01	11	10
C	0	0	~~0~~	!	!
	1	0	=+>	0	0

↑
read, '=' OR '>'

Notes:

CP = COIN PRESENT

=+> = ($\Sigma=30c$) + ($\Sigma>30c$)

CR = CHAMBER READY

PDR = PDP DROP READY

$$D_A = \bar{A}.B.(=+>) + A.\bar{C}$$

AB \ C		00	01	11	10
C	0	0	~~0~~	!	0
	1	$\bar{C}P$	=	0	!

$$D_B = \bar{A}.B.(=) + \bar{B}.C.(\bar{C}P) + B.\bar{C} + A.\bar{B}.C$$

AB \ C		00	01	11	10
C	0	CP	~~0~~	PDR	CR
	1		0	0	

$$D_C = \bar{A}.\bar{B}.(CP) + B.\bar{C}.(PDR) + A.\bar{B}.(CR) + \bar{B}.C$$

47

Next State Maps for J-k Type Implementation

AB \ C	00	01	11	10
0	0	$\emptyset$	$\emptyset$	$\emptyset$
1	0	=+>	$\emptyset$	$\emptyset$

$$J_A = B \cdot (= + >)$$

AB \ C	00	01	11	10
0	$\emptyset$	$\emptyset$	0	0
1	$\emptyset$	$\emptyset$	1	1

$$K_A = C$$

AB \ C	00	01	11	10
0	0	$\emptyset$	$\emptyset$	0
1	$\overline{CP}$	$\emptyset$	$\emptyset$	1

$$J_B = C \cdot (\overline{CP}) + A \cdot C$$

AB \ C	00	01	11	10
0	$\emptyset$	$\emptyset$	0	$\emptyset$
1	$\emptyset$	=	1	$\emptyset$

$$K_B = C \cdot (=) + A \cdot C$$

AB \ C	00	01	11	10
0	CP	$\emptyset$	PDR	CR
1	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

$$J_C = \overline{A} \cdot (CP) + B \cdot (PDR) + A \cdot \overline{B} \cdot (CR)$$

AB \ C	00	01	11	10
0	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
1	0	1	1	0

$$K_C = B$$

The Output Decoder

" Make up a map for each output.

Plot the map using MEV for the conditional variables, then simplify and implement "

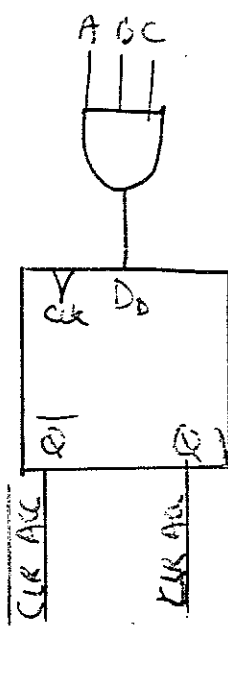
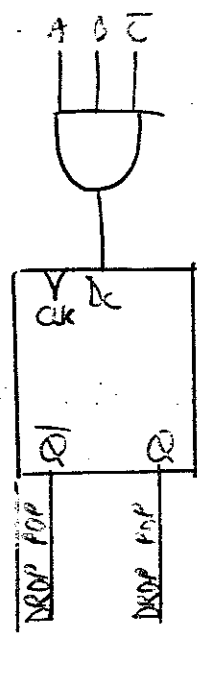
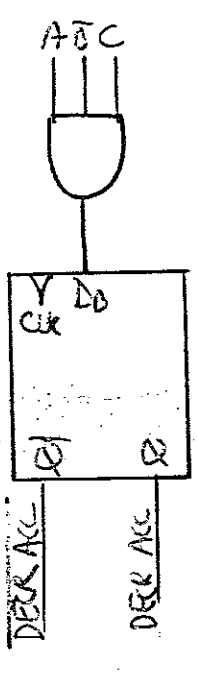
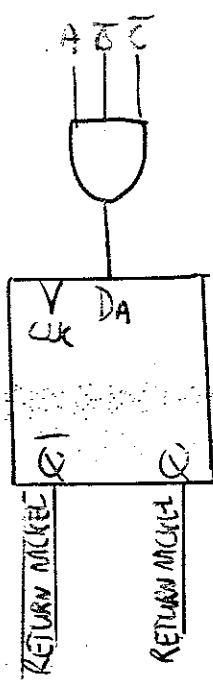
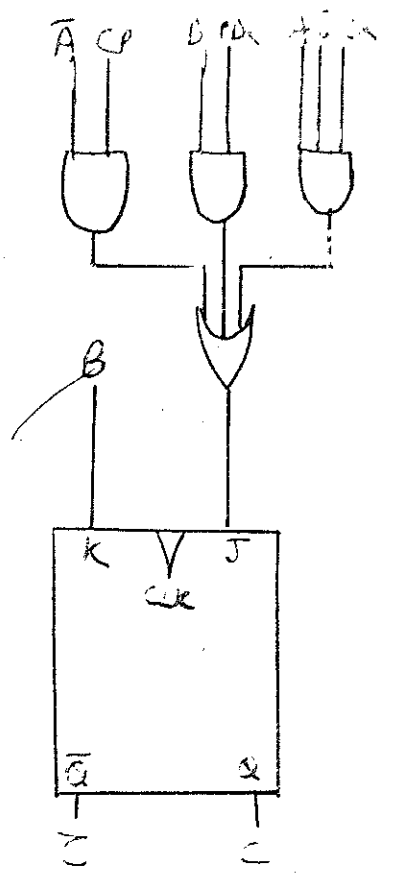
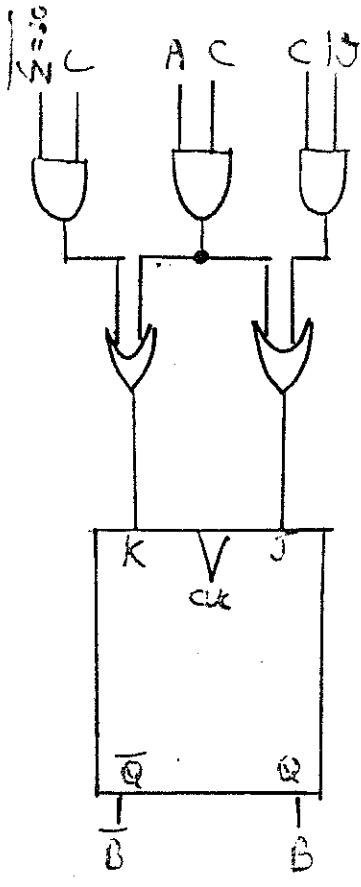
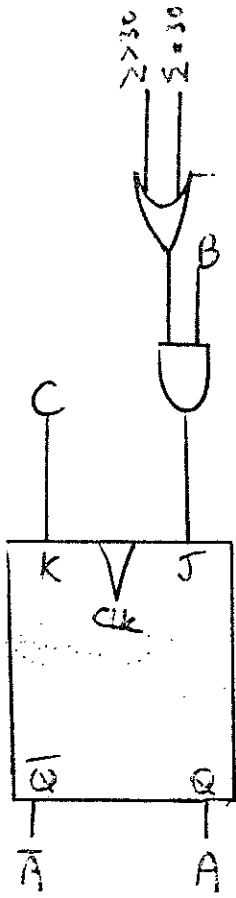
In this case this method is redundant since no output is a function of more than one state and so we can directly write the outputs by inspecting the MDS diagram.

$$\text{RETURN MICKEL} = A \cdot \bar{B} \cdot \bar{C}$$

$$\text{DECR ACC} = A \cdot \bar{B} \cdot C$$

$$\text{DROP POP} = A \cdot B \cdot \bar{C}$$

$$\text{CLR ACC} = A \cdot B \cdot C$$



Output
telling
Register

FLAG=1 and CLK, all of which can be traced through the detailed flow diagram.

Now consider one more design problem illustrating the use of the design phases (1-9). This problem requires you to interpret a set of detailed specifications and further illustrate the functionality of the system controller concept. Special thanks to Prof. Richard Ohran of Brigham Young University for suggesting a problem such as this back in 1971.

EXAMPLE 7-5: The Pop Machine Controller

Introduction and specifications. The El-Rip-O Vending Machine Company wishes to update its Model 1909 mechanical vending machine. It is desired that a first generation digital controlled prototype system be developed for test and evaluation; nothing really fancy, just an evaluation prototype. The El-Rip-O Company has entered into an OEM (Original Equipment Manufacturers) agreement with the Futzel-Boopsdink Pipe and Die Company to provide the coin receiver, coin changer, and pop dropping systems for the new El-Rip-O Model 1971 system when it is fully developed.

Preliminary specifications. The initial digital control system should be developed such that it will direct the control of the coin receiver, coin changer, and pop drop mechanics and provide the El-Rip-O Company with a system capable of automatically dispensing soda pop at 30¢ per can and making the proper change retrieval for the following coin sequences of nickels, dimes, quarters, and half-dollars.

Prescribed operation. Upon the insertion of each coin, the controller is to record the coin value and issue the proper change, if required, in nickels only, then drop a can of soda pop. At this time a solenoid is to be activated that lets the coins collected in the coin sequence drop into a common collector box. This system is to have a coin release feature for the manual release of jammed coins and coins collected in "short" sequences, and the coin release feature is also intended to reset the system to an initial condition.

Coin receiver specification. The Futzel-Boopsdink Pipe and Die Model CR-1971 coin receiver is described by the following features:

- (1) single slot coin entry;
- (2) electronic coin detection;
- (3) guaranteed coin detection for U.S. half dollars, quarters, dimes, and nickels;
- (4) automatic nonvalid and bent coin rejection;
- (5) all control inputs and outputs are standard TTL compatible;
- (6) coin-catching mechanism and manual coin release feature;
- (7) a special mechanical mechanism that prevents coin over-run.

Electrical specifications. See Figure 7-32.

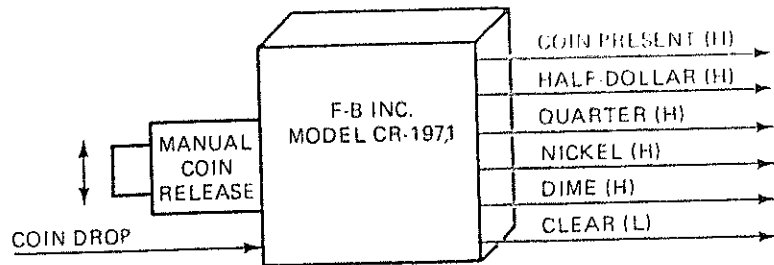


Fig. 7-32. Block diagram of CR-1971.

Signal description.

COIN DROP—an input that commands the CR-1971 to drop collected coins into the common collection box.

CLEAR—an output that will go to a 5 volt ± 0.5 level for the duration of the depression of the coin release.

COIN PRESENT—an output signifying that a coin is present in the coin receiver and denomination has been determined.

HALF DOLLAR, QUARTER, DIME and NICKEL—outputs signifying the denomination of coin present.

The timing relationship between COIN PRESENT, HALF-DOLLAR, QUARTER, DIME, and NICKEL is described by the timing diagram shown in Figure 7-33.

Coin changer. The Futzel-Boopsdink Pipe and Die Model CC-1971 coin changer (see Figure 7-34), developed by the Stuck-Muzzey subsidiary, features:

- (1) fast, reliable, electro-mechanical nickel-ejecting system (100 msec per nickel maximum);
- (2) special READY status output which indicates when another coin ejection sequence can be started;
- (3) automatic load of 50 nickels reserve.

EJECT NICKEL—TTL compatible input, pulse triggered with minimum width of 10 msec restriction.

CHANGER READY—an output status line indicating the changer will respond to EJECT NICKEL.

Timing relations. See Figure 7-35.

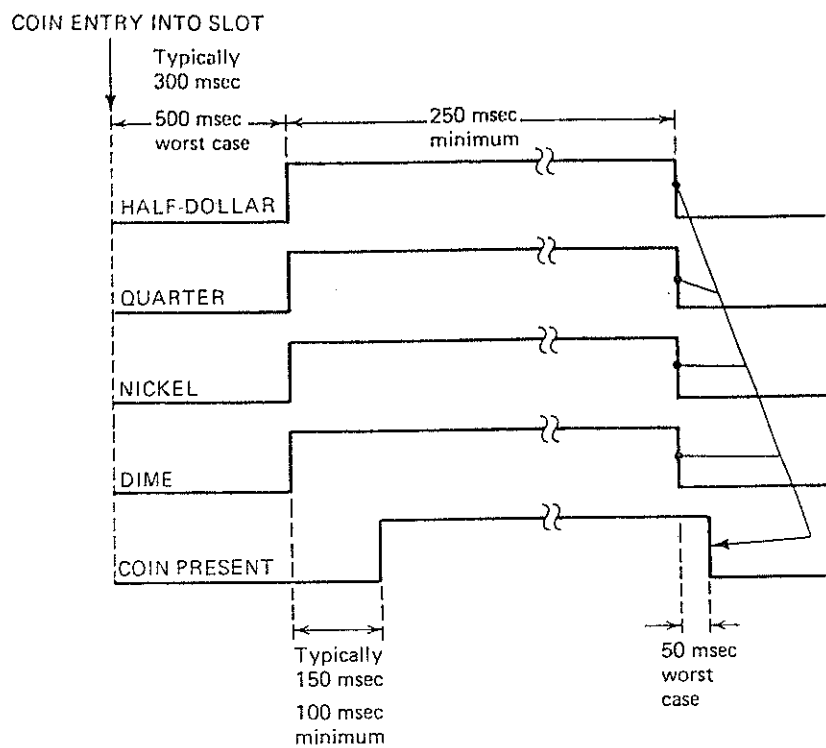


Fig. 7-33. Timing detail of the CR-1971.

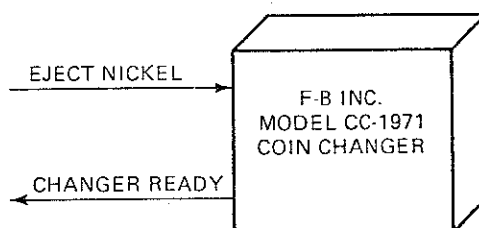


Fig. 7-34. The block diagram of the CC-1971.

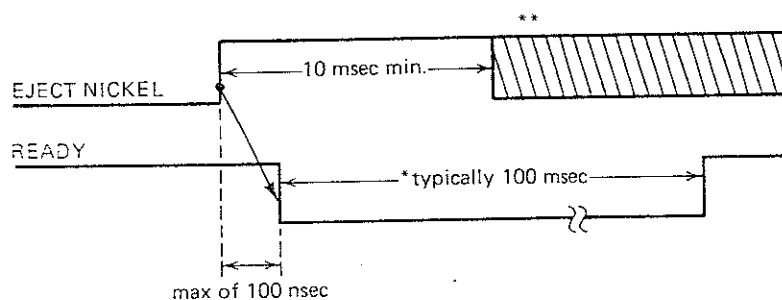


Fig. 7-35. Timing detail of the CC-1971.

Pop-drop mechanism The Futzel-Boopsdink Model PD-1971 pop-drop mechanism developed by Bend-A-Can Division (see Figure 7-36) is a fast solenoid operated device featuring:

- (1) single TTL compatible drop command input;
- (2) pulse operated with 10 msec minimum pulse width;
- (3) status READY line

Timing and operation is identical to that of the coin changer.

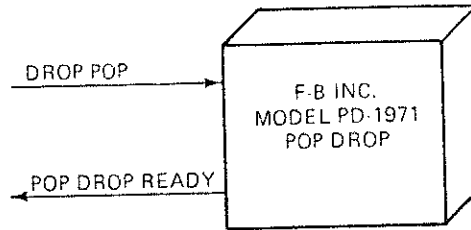


Fig. 7-36. The block diagram of the PD-1971. Note that POP DROP READY will not return to the READY status if the mechanism is jammed.

7-11.3 SYSTEM DEVELOPMENT

First let it be noted that the system to be designed has certain frailties related to its cheating the customer as well as the customer cheating it. However, since this development is only an evaluation prototype, these frailties can be temporarily overlooked. The specifications for a foolproof system are given as a design exercise and left for you to work out the fine details.

Phase I. Because of your intimate knowledge of vending machines derived from the everyday encounters with these beasts (see Figure 7-37), along with the specifications given, Phase I is pretty well covered.



Fig. 7-37. Man and the vending machine.

Phase II. From the knowledge accounted for in Phase I, the bare-bones block diagram as shown in Figure 7-38 can be readily developed. This bare-bones block diagram is supported by the first-cut flow diagram shown in Figure 7-39. Note that this flow diagram clearly defines the proposed big picture sequential behavior of the controller operation. Now keep in mind that this proposed operation is but one of several possible alternatives. However, it was selected on the basis of the following criteria:

- (1) It provides a practical and effective system.
- (2) The author feels that it nicely demonstrates the logical sequence of events.
- (3) It also demonstrates how properly designed "hard wired" systems can have a reasonable degree of flexibility.

Though these criteria are somewhat biased, the concept of making a selection from several possible alternatives should not be considered as such. It is very important that you derive several alternatives and then select one of these based on some criteria. Now these criteria can vary depending on the design requirements, but the major cornerstones should be:

- (1) Practicality
- (2) Effectiveness
- (3) Efficiency
- (4) Degree of flexibility

The flexibility facet should be treated in the light of possible future applications and modifications. This is in keeping with the invariable fact that somebody will want something changed at the most inopportune time. However, don't let these anticipations completely override your preliminary preparations; just keep them in mind. For example, though there is no mention of price changes or multi-pop selection requirements in the specifications for the pop machine, these features are included with no real extra effort or cost. They are simply a natural fallout of the proposed system.

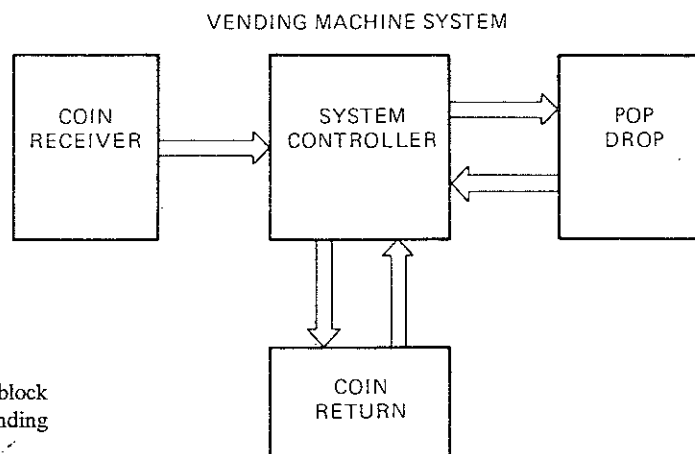


Fig. 7-38. The bare-bones block diagram of the prototype vending machine.

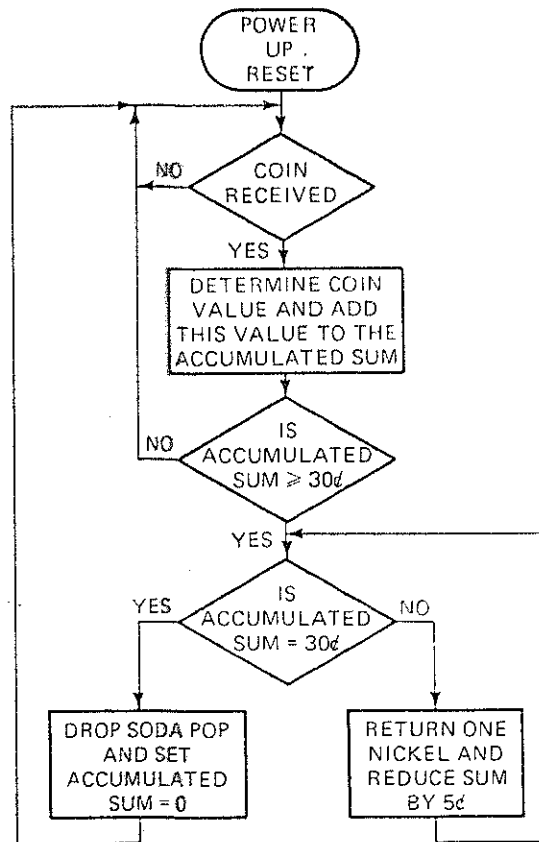


Fig. 7-39. A proposed first-cut flow diagram defining the vending machine's basic operation.

7-11.4 THE DETAILED FLOW DIAGRAM, FUNCTIONAL PARTITION AND MDS DIAGRAM DESCRIPTION

The detailed flow diagram shown in Figure 7-40 defines the exact sequential behavior of the functional partition shown in Figure 7-41. It should be clear that this flow diagram is an extension to the first-cut diagram. Along with this it also defines the timing relations of the control signal to and from the subsystems. The functional partition typifies the system controller in its presiding role. It should be noted that the arithmetic operations called for are carried out by the interconnection of the ADDER, PIPO REGISTER, DOWN COUNTER, and COMPARATOR. The PIPO REGISTER was added to insure the edge-triggered loading of the COUNTER. This register/counter complex is referred to as the ACCUMULA

TOR (ACC) in this example. Further, it should be noted that a "strappable" compare word allows for easy price changing up to 75¢. The value of each coin is then encoded to the binary value of the number of nickels it represents in accordance with the indicated encoding (w_4, w_3, w_2, w_1) specification table. This encoded value is then added to the present value of the accumulator on the rising-edge of COIN PRESENT. Thus the basic functions called for by the first-cut and detailed flow diagram are brought to light with a hardware description. Note further that the IC package count is up to six packages, all with 16 pins or less.

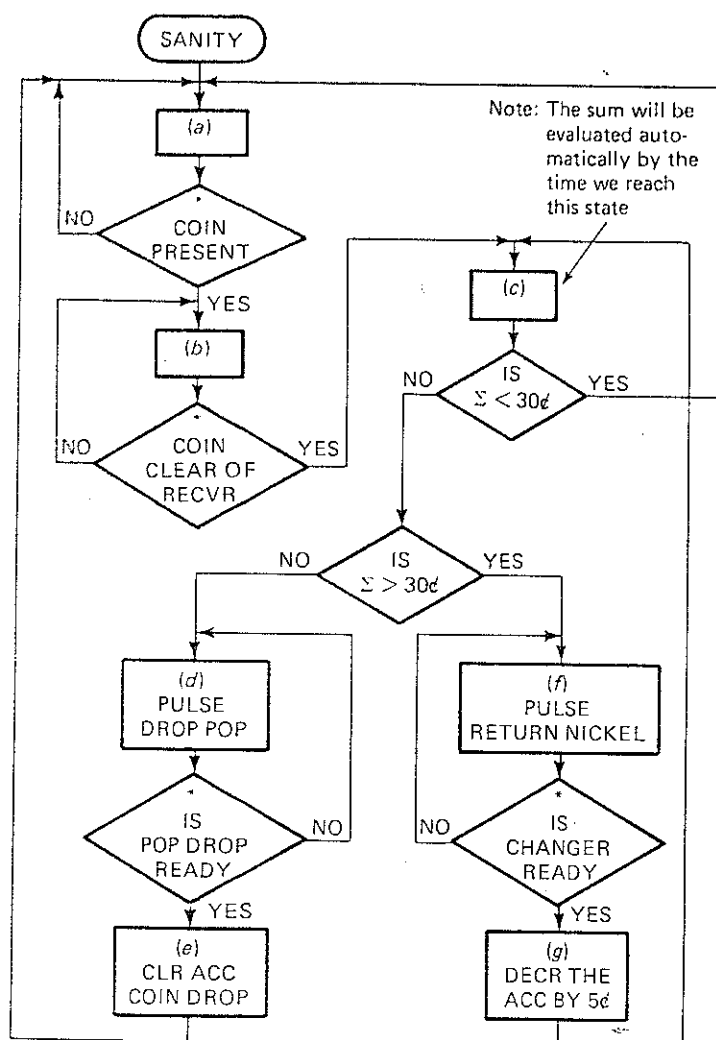


Fig. 7-40. The detailed flow diagram for the prototype pop machine system controller.

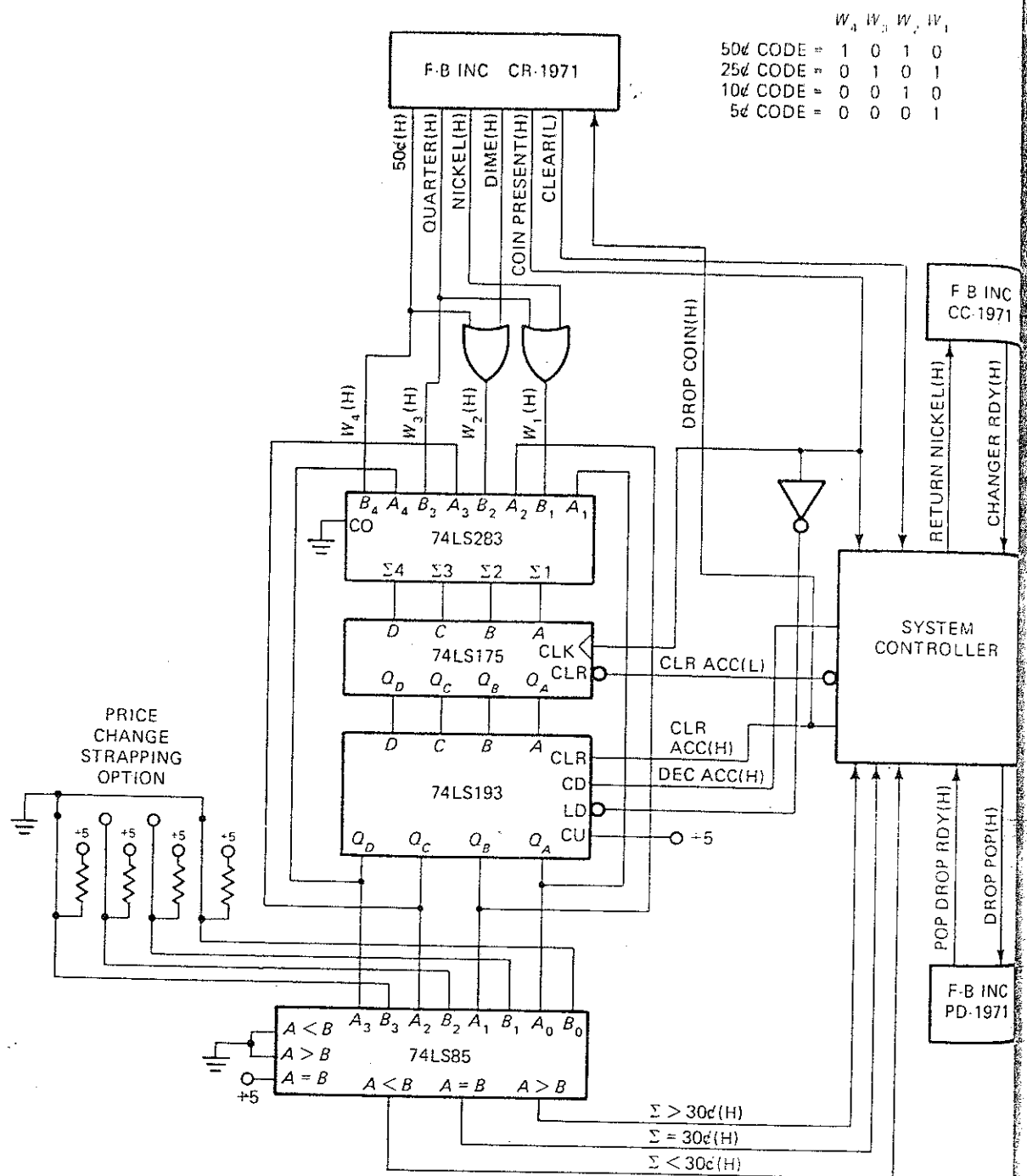


Fig. 7-41. The functional partition of the pop vending machine control system.

The seven-state MDS diagram for the system controller is shown in Figure 7-42. All branching conditions and outputs are clearly specified in close accordance with the specification set forth in the detailed flow diagram. The branching decisions based on asynchronous inputs are properly identified and are defined by the GO, NO-GO decision construct discussed in Section 7-10.1

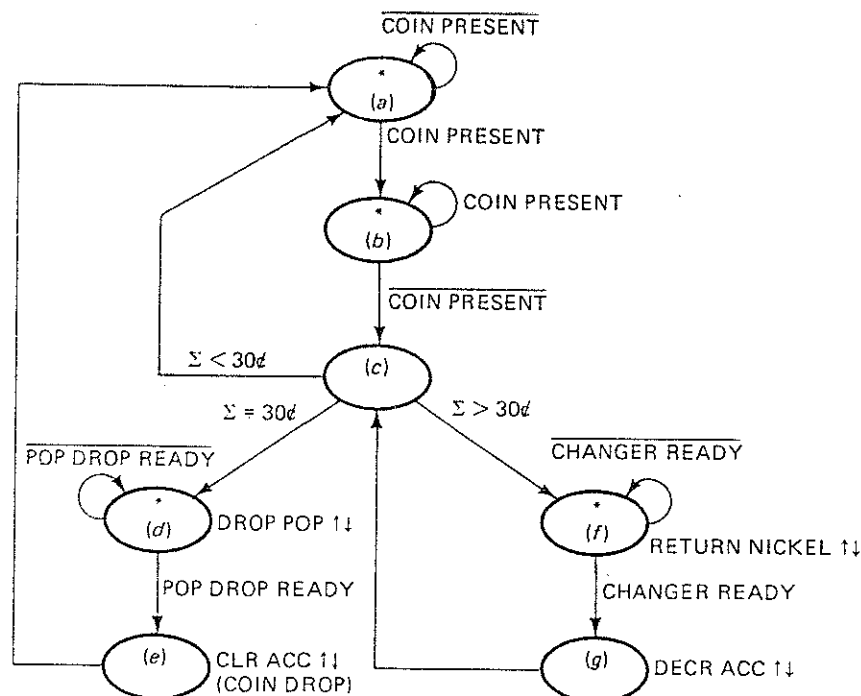


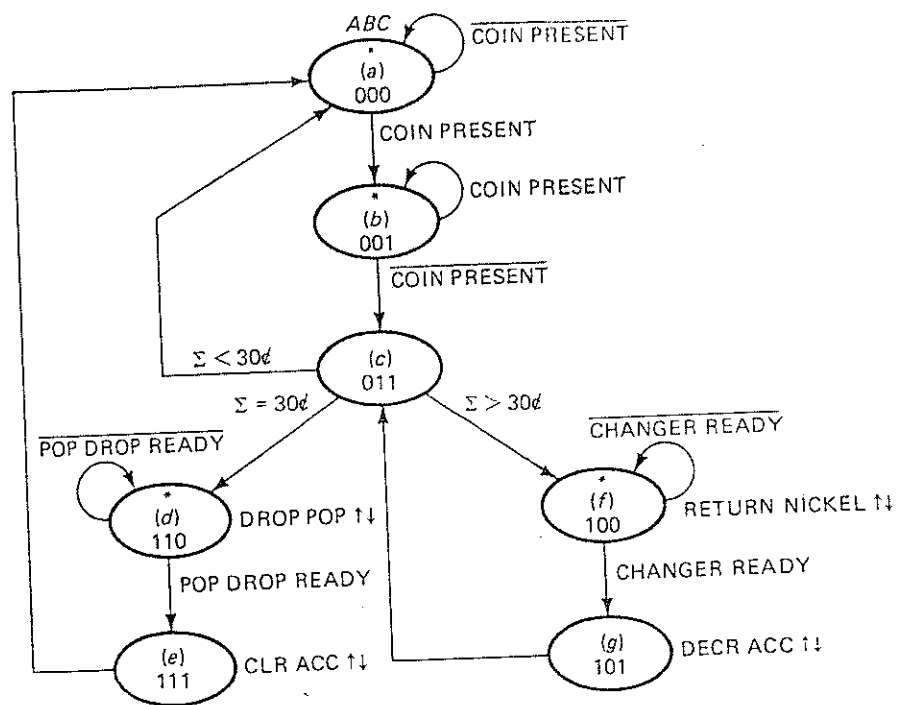
Fig. 7-42. The MDS diagram for the prototype pop machine system controller. Note that it is assumed CLR ACC ↑↓ causes the coins to drop.

7-11.5 THE HARDWARE IMPLEMENTATION OF THE NEXT STATE AND OUTPUT DECODER

Up to this point the major effort has been directed toward the system's three D's: Definition, Description, and Documentation. Now some of the practical aspects of an actual hardware development process are presented. The first step in this development process is to make a state assignment in accordance with the rules set forth earlier, paying strict attention to asynchronous branching (see Figure 7-43). A state map (regular K-map) is used to aid the state assignment process. Here the rules related to asynchronous branching variables are given the highest priority.

This priority calls for the following state assignment constraints:

a adjacent to b
 b adjacent to c
 d adjacent to e
 f adjacent to g



C \ AB				
	00	01	11	10
0	a	∅	d	f
1	b	c	e	g

State Map

Fig. 7-43. The system controller MDS diagram for the prototype pop machine complete with a state assignment and state map.

The rest of the states are assigned with some reference to the simple rules set forth in Chapter 6 and the list in Table 7-1. However, no exhaustive attempt is made to optimize the state assignment.

TABLE 7-1 State assignment listing

Previous States	State in Question	Next State	Assignment Suggestions
<i>a, c, e</i>	<i>a*</i>	<i>b</i>	<i>a ADJ b*</i> ; <i>a, c, e ADJ</i>
<i>a, b*</i>	<i>b*</i>	<i>b, c</i>	<i>b ADJ c*</i> ; <i>a ADJ b*</i>
<i>b</i>	<i>c</i>	<i>a, d, f</i>	<i>a, d, f ADJ</i> ; <i>b ADJ c*</i>
<i>c, d</i>	<i>d*</i>	<i>c</i>	<i>d ADJ c*</i> ; <i>c ADJ d</i>
<i>d</i>	<i>e</i>	<i>a</i>	NR; <i>d ADJ e*</i>
<i>c, f</i>	<i>f*</i>	<i>g</i>	<i>f ADJ g*</i> ; <i>c ADJ f</i>
<i>f</i>	<i>g</i>	<i>c</i>	NR; <i>f ADJ g*</i>

*→a must assignment; NR→NO Requirement

7-12 NEXT STATE DECODER MAPS

Once a state assignment is made, the next design step determines the first real commitment to any specific type of hardware. This example demonstrates the ease with which small sequential machines can be implemented at the SSI gate and Flip-Flop level, knowing well that other alternatives do exist. This ease is brought about in part by taking advantage of the inherent application of the VARIABLE-ENTERED MAP. Using VEM's bypasses the tedious detail of filling out a PRESENT/NEXT STATE TABLE by going *directly from the MDS diagram to the NEXT STATE maps*. The plotting of the VEM's is a simple process. However, it does require a *properly documented MDS diagram* and some ability to recognize, by inspection, the *CONDITIONAL SET OPERATIONS* called for in this MDS diagram. This process is described in the following paragraph.

Next State Map Plotting Process

Using a MDS diagram with a state assignment, select a state, then examine the branching mnemonics to *all* NEXT STATES of the selected state; and enter into the STATE CELL of the NEXT STATE MAP the Boolean expression that defines all of the SET CONDITIONS ($0 \rightarrow 1, 1 \rightarrow 1$) that are required for the chosen state variable in that state. Thus these *CONDITIONAL SET EXPRESSIONS* become *MAP ENTERED VARIABLES* in the NEXT STATE MAPS.

This process is repeated for each STATE VARIABLE of each STATE until the NEXT STATE maps are completed. Remember, the NEXT STATE maps are the D-maps from which JK and T maps can be readily obtained. The described processes are clearly demonstrated in Figures 7-44 and 7-45. Pay particular attention to the entry in state cell *c* in the D_A map. Here you see:

(= + >)

which indicates that the A Flip-Flop is set by the next clock edge if the machine is in: STATE c AND ($=$) OR ($>$) are ASSERTED. The rest of the MEV ENTRIES are obvious. However, it is recommended that a map be generated for each Flip-Flop and these maps be plotted in parallel, one state at a time, using the state map as a reference for the state cell location.

From Figures 7-44 and 7-45 notice that the JK does yield a circuit implementation requiring fewer gates (11 gates or four IC packages). Thus the new grand total for the system equals nine IC packages, some with spare gates, which really isn't too bad considering the capabilities of the overall system. However, the design for the OUTPUT DECODER must be completed.

C \ AB	00		01		11		10	
	0	1	2	3	6	7	4	5
0	0	$\emptyset$	1	1				
1	0	$=+>$	0	0				

$$D_A = \bar{A}B(=+>) + A\bar{C}$$

C \ AB	00		01		11		10	
	0	1	2	3	6	7	4	5
0	0	$\emptyset$	1	0				
1	$\overline{CP}$	$=$	0	1				

$$D_B = \bar{A}B(=) + \bar{B}C(\overline{CP}) + B\bar{C} + A\bar{B}C$$

C \ AB	00		01		11		10	
	0	1	2	3	6	7	4	5
0	CP	$\emptyset$	PDR	CR				
1	1	0	0	1				

$$D_C = \bar{A}\bar{B}(CP) + B\bar{C}(PDR) + A\bar{B}(CR) + \bar{B}C$$

Legend: CP = COIN PRESENT
 $=+> = (\Sigma = 30¢) + (\Sigma > 30¢)$
 CR = CHANGER READY
 PDR = POP DROP READY

Fig. 7-44. The NEXT STATE or D maps for the proto-type pop machine system controller.

C \ AB	00	01	11	10
	0	1	2	3
0	0	0	0	0
1	0	1	0	0

$$J_A = B(\bar{C} + C)$$

C \ AB	00	01	11	10
	0	1	2	3
0	0	0	0	0
1	0	0	1	1

$$K_A = C$$

C \ AB	00	01	11	10
	0	1	2	3
0	0	0	0	0
1	CP	0	0	1

$$J_B = C(\bar{C}P) + AC$$

C \ AB	00	01	11	10
	0	1	2	3
0	0	0	0	0
1	0	0	1	0

$$K_B = C(\bar{C}) + AC$$

C \ AB	00	01	11	10
	0	1	2	3
0	CP	0	PDR	CR
1	0	0	0	0

$$J_C = \bar{A}(CP) + B(PDR) + \bar{A}\bar{B}(CR)$$

C \ AB	00	01	11	10
	0	1	2	3
0	0	0	0	0
1	0	1	1	0

$$K_C = B$$

Requires 11 gates, or four IC packages with spare gates.

Fig. 7-45. The NEXT STATE DECODER maps for a JK implementation.

7-13 THE OUTPUT DECODER

The design of the OUTPUT DECODER is like any other combination decoder design problem that can be summed up quickly by the following process:

Make up a map for *each* output. Plot the map using MEV for the conditional variables, then simplify and implement.

However, many times maps serve no useful purpose. For example, in the present system controller there is no output that is a function of more than one (1) state. This information can be quickly gleaned from inspecting the MDS diagram. Therefore, in cases such as these, it is recommended that you make up a simple output list. This output list is nothing more than a list of every required output and

the states and conditions related to the generation of these outputs. For example, the MDS diagram in Figure 7-43 would have the following list:

$$\text{RETURN NICKEL} = A\bar{B}\bar{C}$$

$$\text{DECR ACC} = A\bar{B}C$$

$$\text{DROP POP} = AB\bar{C}$$

$$\text{CLR ACC} = ABC$$

From this expression list observe that none can be further simplified. Therefore, each requires a simple three-input gate (AND function) with the inputs connected to the PRESENT STATE VARIABLES (ABC). However, the ever present glitch problem exists because of our state assignment. The transition from 111→000 introduces the possibility of a glitch in three states. But the real question is which three states. Previous studies indicate that all six of the remaining states are vulnerable when the 111→000 transition is made. Therefore, some sort of glitch prevention must be introduced. In cases with a smaller number of bits changing (two in this case), the possible transitions can be traced in a state map for easy identification of possible glitch problems. However, the 111→000 transition overrides the need for any further glitch studies.

Since there are no outputs which must remain ASSERTED over several states, an OUTPUT HOLDING REGISTER would appear to be an overkill. However, DROP POP and RETURN NICKEL generated in states D and F must be held stable all during these states. Therefore, a holding register is probably called for. Therefore, four three-input NAND gates (74LS10) are chosen for the output decoder and a 74LS175 is chosen as the OUTPUT HOLDING REGISTER. For the completed schematic of the total system controller including the OUTPUT DECODER, see Figure 7-46. From this figure observe that the grand total IC package count is 14, all of which are standard or established 14 or 16 pin IC's (see parts list in Table 7-2).

TABLE 7-2 Parts List

Quantity	Number	Description
1	74LS283	Four-bit adder
1	74LS193	Hex up/down counter
2	74LS175	Quad D PIPO
2	74LS112	Dual edge-triggered JK Flip-Flop
1	74LS85	Four-bit comparator
2	74LS32	Quad of two-input OR
2	74LS10	Triple three-input NAND
1	74LS08	Quad of two-input AND
*1	7404	Hex of inverters
$\frac{1}{14}$	74LS00	Quad of two-input NAND

*The standard 7404 was chosen because of the high drive requirement of the clock input on the 74LS112 (four std LS loads per input)

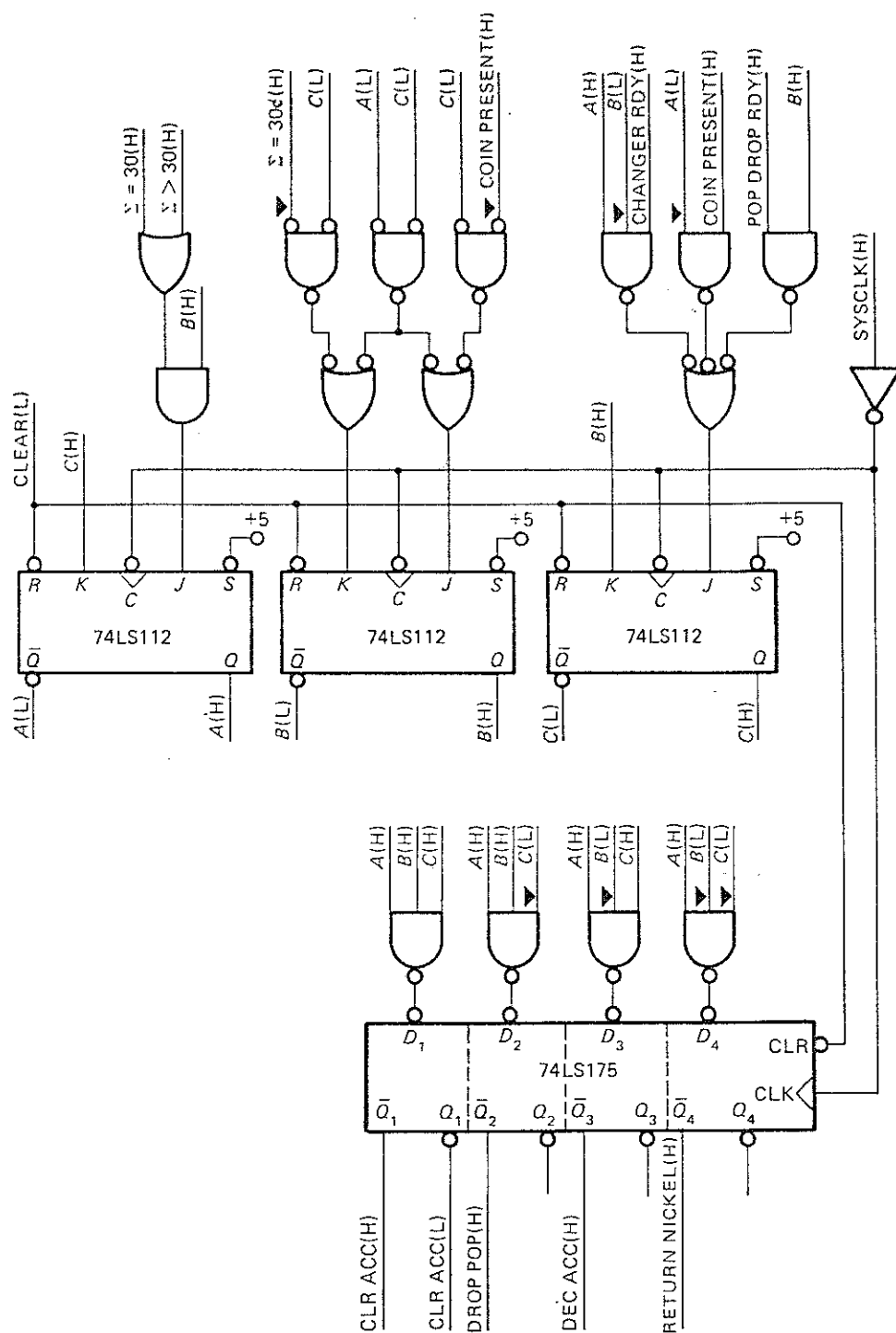


Fig. 7-46. The complete JK implementation of the system controller for the pop vending machine system.

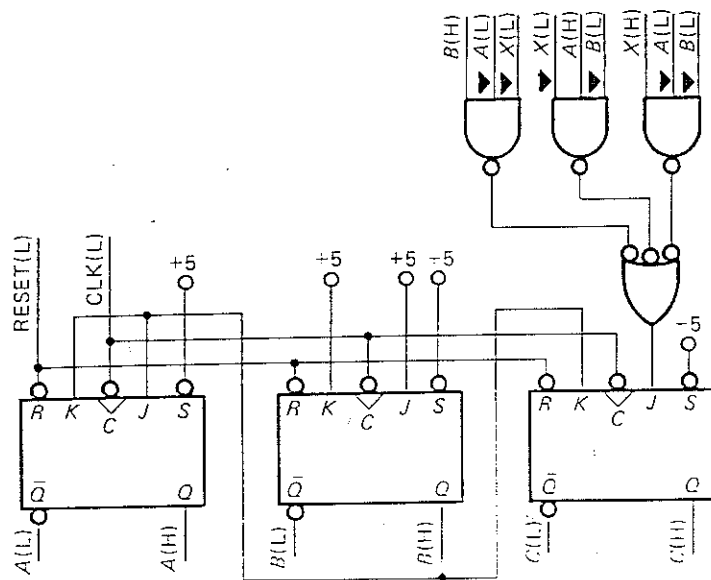


Fig. P6-4.

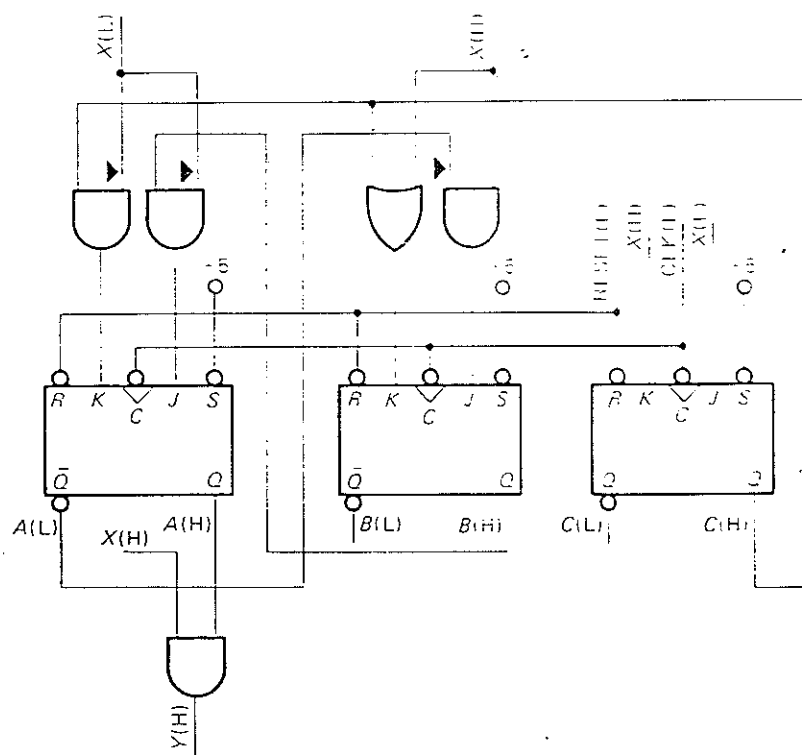


Fig. P6-5.