A DESIGN METHOD OF ASYNCHRONOUS SEQUENTIAL CIRCUITS BASED ON FLOW DIAGRAM

S. TERPLÁN*

Department of Process Control,
Technical University, H-1521 Budapest

Received September 29, 1984.

Summary

A systematic, asynchronous design method based on a flow diagram is shown. The realization utilizes a so-called phase-register coded 1 out of n. A phase consists of so-called phase-register cells, which are elementary asynchronous networks including edge-sensitive integrated circuit flip-flops. The circuits developed by the proposed method are free of critical races and essential hazard faults.

1. Introduction

For the solution of practical problems asynchronous sequential networks must be designed in cases when the operational speed expected from the network cannot be achieved otherwise. In usual cases synchronously operating microprogrammed networks are the main tools for solving sequential control problems. However, with given IC technology—e.g. bipolar TTL—wired asynchronous circuits always permit faster operation than microprogrammed circuits. With these latter the tasks to be performed simultaneously are divided into elementary steps, and the required operation is accomplished by successive steps. Even today, this fact justifies the application of asynchronous control units in fast-working equipment. Such are, e.g. the high-speed mass stores of computers, as well as their interfaces, some real-time acquisition devices, telecommunication and video systems, some real-time physical simulation systems, etc.

The classical asynchronous design procedures starting from a flow table are less applicable to the design of practical control units, since the flow table becomes puzzling by its vast dimensions. At the same time, the necessity of

* The author thanks Prof. Dr. H. M. Lipp and Prof. Dr. J. Beister for the useful consultations within the framework of the scientific cooperation between the Institut für Nachrichtenverarbeitung Universität Karlsruhe and the Department of Process Control Technical University Budapest.

testing for essential hazards and critical races makes the designing work too complicated.

In recent years new procedures which try to solve the above-mentioned problems have been reported in literature. One group of procedures is based on the application of elementary asynchronous edge-sensitive sequential networks. Such edge-sensitive elements are, e.g., the *M* and *G* elements [7], the *I–T* flip-flop and the *D*-TSFF [9].

To describe the operation, various "edge-sensitive" flow tables are used which can be regarded as developed forms of the classical flow table. The edge-sensitive elements exclude errors due to the essential hazards and critical races, therefore, they allow the coding procedures applicable to synchronous networks. However, as has been shown by E. SCHMITT and S. WENDT [12], proper operation is not guaranteed even with synchronous sequential networks, if arbitrary coding and, e.g. edge-sensitive *D* or *J–K* flip-flops are used. Therefore, handling the hazard phenomena in this manner does not yield the final solution. As has been pointed out by J. BEISTER [13, 14], guaranteed proper operation can be achieved, if the paths of signal transmission within the logical network are taken into account separately in the design work even in the case when a path consists of wires only. A detailed analysis and development of the "edge-sensitive design methods" are given in the work of *R.* KIRCHNER [10].

Besides, there is the procedure recommended by R. DAVID [11] using a state graph for the description of the task. An elementary asynchronous sequential network, the so-called CUSA (Cellule Universelle pour Sequences Asynchrones) is assigned to each state. Choosing "1 out of *n*" to encode the states, the CUSA realizes the product of the disjunction of the secondary variables and of the input combination. The author requires certain formal properties of the state graph and has elaborated a systematic verification procedure to check their presence. Errors from essential hazard and critical race conditions are eliminated by the delay conditions of the CUSA built in advance.

A new, interesting method of designing asynchronous sequential networks has been presented by A. DITZINGER and H. M. LIPP [15, 16]. They made experiments with feed-back PROM and FPLA circuits and established that, if certain types are used, operation free of errors from hazards and critical races can be achieved with state coding of synchronous type. They presented this on the basis of the delay conditions of some circuit types. They described the task by means of flow diagrams, from which the functions to be realized can easily be read off.

The two basic ideas that can be derived from the above descriptions and that can be used also in the present work are as follows:

— Application of flip-flops with edge-sensitive inputs as universal building blocks for the design of asynchronous sequential networks
— Use of a graph instead of the flow table.

At the Department of Process Control at the Technical University of Budapest successful investigations have been carried out in systematically designing control units of synchronous nature with the use of flow diagrams [1, 2, 3]. The systematic procedure suitable for designing asynchronous logical networks to be presented in this paper is linked with these investigations.

2. Symbols of the asynchronous flow diagram for the case of changes between adjacent input combinations

The flow chart is constructed by the designer intuitively on the basis of verbal given or other conditions. The operational sequence is partitioned into so-called phases. The phase is a concept similar to that of a state. Phase transitions are triggered by the changes between input combinations (input changes) important from the view-pont of the output sequence. These changes are stated by the designer in the so-called triggering flow diagram instructions in the form of functions F of the input signals. The transition $\emptyset$–1 taking place in the value of function F indicates the phase transition. The next phase is determined by the "yes" branch of the triggering instructions, while the present phase is generated by the triggering instructions connected with each other through the "no" branches (the sequence is irrelevant). The flow chart is normal with respect to the phases, i.e. a given input change can cause only one direct phase transition. The functions F taking part in bounding the developing new phase may have arbitrary value, and even a transition 1–$\emptyset$ may take place in the value of these functions F during the transition to the next phase, since the 1–$\emptyset$ transitions never generate a phase transition.

If the designer finds a condition derived from the input signals that can exclusively enable the respective phase transition, then he may state this in so-called branching instructions in the form of a function G of the input signals. Thus, a condition or enabling function G can be assigned to each of the functions F. The 0–1 transition taking place at the output of function F triggers a phase transition only if, during this change of the input combination, function G has the logical value 1. If the 0–1 transition occurring at the output of function F is not meant to trigger a phase transition, then the value of function G must be $\emptyset$ during this change.

Assignment of a function F and a function G to each other takes place in the way that the YES branch of the triggering instruction corresponding to function F leads to a so-called branching instruction in which function G is defined in the form $G=f(x)$. The YES branch of the branching instruction, which means the validity of $G=1$, leads to the next phase. The two "YES" branches quasi "connected in series" determine the next phase. The "NO" branch of the branching instruction leads to the same place as the "NO" branch of the triggering instruction, i.e. to some other instruction pair taking part in the determination of the current phase. The two "NO" branches together, quasi "connected in parallel" take part in the determination of the current phase. If two functions F leading to different next phases agree with each other and their functions G are complementary, then a simpler notation can be used in the flow

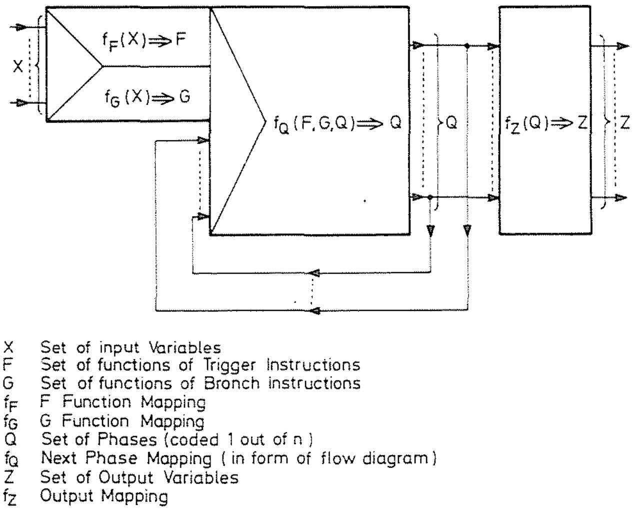
X
$f_F(X) \Rightarrow F$
$f_G(X) \Rightarrow G$
$f_Q(F,G,Q) \Rightarrow Q$
Q
$f_Z(Q) \Rightarrow Z$
Z
X Set of input Variables
F Set of functions of Trigger Instructions
G Set of functions of Bronch Instructions
f_F F Function Mapping
f_G G Function Mapping
Q Set of Phases (coded 1 out of n)
f_Q Next Phase Mapping (in form of flow diagram)
Z Set of Output Variables
f_Z Output Mapping

Fig. 1. X: set of input variables; F: set of functions of trigger instructions; G: set of functions of branch instructions f_F : F function mapping; f_G : G function mapping; 0: set of phases (coded 1 out of n); f_0 : next phase mapping (in form of flow diagram) Z: set of output variables; f_Z output mapping

diagram. The two function pairs F and G can be replaced by one. In such a case, the "NO" branch of the branching instruction also leads to a next phase.

If the Moore model is applied, the output combinations can be produced with the aid of the functions formed from the output of the phase flip-flops, i.e., the value of the output signals is assigned to the phases. This can be done simply if the phases are coded "1 out of n". The Moore model network structure corresponding to the flow diagram discussed so far is shown in Fig. 1.

In fact, the phases mean such a partition of the states of a logical problem given by a normal, asynchronous primitive flow table according to the Moore model that each block corresponds to a phase having the following properties:

a) No contradictory output combinations are assigned to those states belonging to the same phase.
b) There cannot be more than one stable state entry in each column of a flow table portion corresponding to an individual phase
c) The columns of each adjacent input combination pair x^k, x^l, where the input change $x^k \rightarrow x^l$ produces during the phase transition a stable next state belonging to the phase, satisfy the following condition: There is no row in which there would be a stable state entry under x^k during the time when under x^l there is an entry meaning a state belonging to some other phase. With symbols:

If $f_q(x^k, y_i) = y_i$,
$f_q(x^l, y_i) = y_j$, and
$f_q(x^l, y_j) = y_j$, where $y_i \quad Q_q$, $y_j \in Q_q$, then there is no row in phase Q_q in which
$f_q(x^k, y_r) = y_r$,
$f_q(x^k, y_r) = y_p$, and
$f_q(x^l, y_p) = y_p$, where $y_r \in Q_q$, for all indices r in question, and where $y_p \quad Q_p$, $Q_q \neq Q_p$

The partition defined in this way constitutes the basis of the procedures for converting a flow table into a flow diagram and of a flow diagram into a flow table, as detailed in [4]. Thereby, it can be demonstrated that the flow diagram describes the operation uniquely only if the initial states of the network are known. Therefore they must be specified separately. Following from the above partition, the specification of the initial state means the specification of the initial phase and an input combination.

As a code "*1* out of *n*" has been chosen to encode the phases, the danger arises that output signals required to remain constant change during the phase transition. This is not permissible. At the same time it is advisable to transform the structure according to the Moore model used so far into a Mealy model so as to decrease the number of states.

For the two reasons given above the network part producing the output signals must be omitted from the block diagram shown in Fig. 1 and the network part shown in Fig. 2 has to be used instead. For this purpose the following instructions for setting the output signals have been introduced:

a) SET instruction
The output signal to which it refers will assume the logical value 1 if K_S is or becomes 1 in the given phase of the network. After this the value of the output signal will remain unchanged until the next RESET instruction pertaining to the output signal occurs.
b) RESET instruction
The output signal to which it refers will assume a logical value 0 if K_R is or becomes 1 in the given phase of the network. After this the value of the

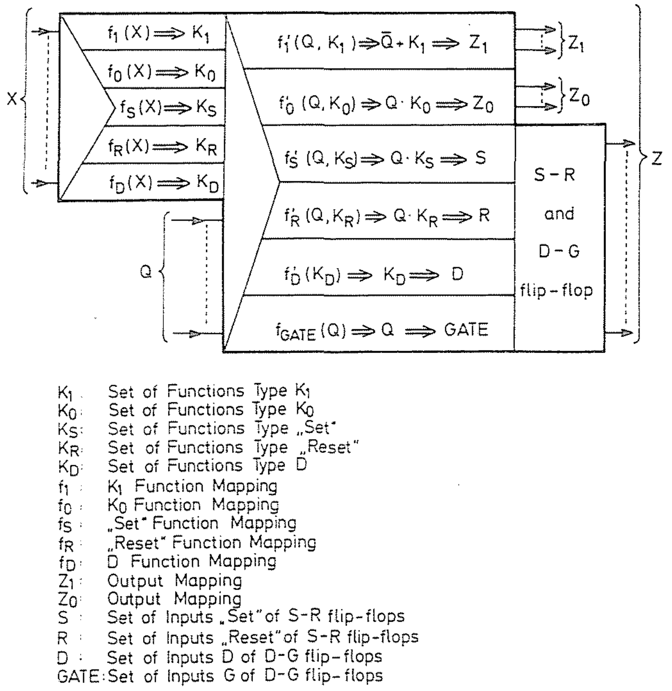
X
f1 (X) ⟹ K1
f0 (X) ⟹ K0
fS (X) ⟹ KS
fR (X) ⟹ KR
fD (X) ⟹ KD
Q
f1' (Q, K1) ⟹ Q̄ + K1 ⟹ Z1
f0' (Q, K0) ⟹ Q · K0 ⟹ Z0
fS' (Q, KS) ⟹ Q · KS ⟹ S
fR' (Q, KR) ⟹ Q · KR ⟹ R
fD' (KD) ⟹ KD ⟹ D
fGATE (Q) ⟹ Q ⟹ GATE
Z1
Z0
S-R and D-G flip-flop
Z
K1 : Set of Functions Type K1
K0 : Set of Functions Type K0
KS : Set of Functions Type „Set"
KR : Set of Functions Type „Reset"
KD : Set of Functions Type D
f1 : K1 Function Mapping
f0 : K0 Function Mapping
fS : „Set" Function Mapping
fR : „Reset" Function Mapping
fD : D Function Mapping
Z1 : Output Mapping
Z0 : Output Mapping
S : Set of Inputs „Set" of S-R flip-flops
R : Set of Inputs „Reset" of S-R flip-flops
D : Set of Inputs D of D-G flip-flops
GATE : Set of Inputs G of D-G flip-flops

Fig. 2. K_1 : set of functions type K_1 ; K_0 : set of functions type K_0 ; K_S : set of functions type "set"; K_R : set of functions type "Reset"; K_D : set of functions type D; f_1 : K_1 function mopping; f_0 : K_0 function mopping; f_S : "Set" function mopping; f_R : "Reset" function mopping; f_D : D function mopping; Z_1 : output mopping; Z_0 : output mopping; S : set of inputs "Set" of $S-R$ flip-flops; R: set of inputs "Reset" of $S-R$ flip-flops; D: set of inputs D of $D-G$ flip-flops; GATE: set of inputs G of $D-G$ flip-flops

output signal will not change until the next SET instruction pertaining to the output signal occurs.

c) D instruction
The output signal to which it refers follows the variation of function K_D in the given phase of the network, then holds the value existing at the moment of the appearance of the next phase until the same instruction comes on.

d) Function instruction (K_1 type, product of sums form)
The output signal to which it refers follows the changes of function K_1 in the given phase of the network: after cessation of the phase, it will assume the value 1 until the next function instruction (of type K_1) pertaining to the output signal.

e) Function instruction (K_0 type, sum of products form)
The output signal to which it refers follows the variation of function K_0 in

the given phase of the network; after cessation of the phase, it will assume the value Ø until the next function instruction (of type K_0) pertaining to the output.

The instructions setting the output signals have to be placed in the phases, in the part bounded by the trigger instructions. In this way they will be assigned to the phases.

3. Properties of the functions included in the flow diagram

Functions F and G have the properties listed below:

a) The value of any function G remains unchanged in the current phase when a Ø–1 transition occurs on the output of the associated F function.

b) In any phase, several G functions may be enabling different phase transitions. But under any given input change, only one of the associated F functions can undergo a Ø–1 transition (and actually trigger a change of phase). F functions without a branching instruction are paired with functions $G \overset{\text{id}}{=} 1$.

c) When a phase transition is triggered under a given input change, several G functions in the next phase may be enabling a second phase transition under the same input change, but none of the associated F functions that determine the next phase can trigger the second transition by changing from Ø to 1. Again, F functions without a branching instruction are considered to be paired with functions $G \overset{\text{id}}{=} 1$.

Provided that functions F and G of the flow diagram have the properties listed above, the functions K in the instructions setting the output signals will have the following properties:

a) The output of the function K_D belonging to any instruction D of the flow diagram does not change when the phase in which the instruction occurs is just beginning or ending.

b) No 1–Ø transition takes place on the output of the function K belonging to any instruction of type Set, Reset and K_0 of the flow diagram when the phase in which the instruction occurs is just beginning, and no Ø–1 transition may take place when the phase ends.

c) No Ø–1 transition takes place on the output of the function K_1 belonging to any K_1-type instruction of the flow diagram when the phase in which the instruction occurs is just beginning, and no 1–Ø transition may take place when the phase ends.

The above properties are in fact simple formal requirements imposed on the flow diagram constructed on the basis of verbally given worded conditions.

Drawing a parallel with the flow table, properties a) and b) relative to functions F and G mean that a given flow table cell cannot contain two next state entries. Property c) means that the flow diagram is normal. The properties relative to functions K exclude function hazards. Reference [4] states conditions for checking the properties of the functions F, G, and K of the flow diagram. It also gives a detailed description of a test procedure by which the tests can be performed systematically, and which can be programmed for execution by computer.

4. Realization of an asynchronous phase register network under the single input change condition

4.1 *Realization of functions F and G*

The functions are given explicitly in the flow diagram. In their realization the only requirements is freedom from static hazards. Then, assuming a two-level realization, no peak can arise on the output of the networks, which would lead to malfunctioning. Changes are allowed only between adjacent input combinations; there are no function hazards.

4.2 *Realization of the output signals*

With the assumption that the phases are coded "1 out of n", the mappings indicated in Fig. 2 can directly be accomplished by two-level logical functions realized free of static hazards. Accordingly, the mappings that govern the different types of output signals z will have the following forms:

a) $f'_i(Q, K_1) \Rightarrow z_1$: Type K_1 function instruction (product of sums form)
$z_1 = \prod_{i=0}^{n} (Q_i + K_1^{i, z_1})$, where the Q_i mean the phases in which a K_1-type function instruction relative to output signal z_1 is located.

b) $f'_0(Q, K_0) \Rightarrow z_0$: a K_0-type function instruction (sum of products form)
$z_0 = \sum_{i=0}^{n} (Q_i \cdot K_0^i, z_0)$, where the Q_i mean the phases in which a K_0-type function instruction relative to output signal z_0 is located.

c) $f'_s(Q, K_S) \Rightarrow S$: the function acting on the Set input of the S–R flip-flop having output signal z (Set instruction).
$z_{\text{set}} = \sum_{i=0}^{n} (Q_i \cdot K_s^{i, z})$, where the Q_i mean the phases in which a Set instruction relative to output z is located.

d) $f'_R(Q, K_R) \Rightarrow R$: the function acting on the Reset input of the S–R flip-flop having output signal z (Reset instruction).

$z_{\text{Reset}} = \sum_{i=0}^{n} (Q_i \cdot K_R^{i,z})$, where the Q_i mean the phases in which a Reset instruction relative to output signal z is located.

e) $f'_D(K_D) \Rightarrow D$ and $f'_{\text{GATE}}(Q) \Rightarrow \text{GATE}$: the functions acting on the inputs of D–G flip-flops having output signal z (D instruction).

$D_z = K_D^{i,z}$ and $\text{GATE}_z = Q_i$, where Q_i means the phase in which the instruction D relative to output z is located. It can be imagined that the designer has placed the same instruction D into several phases of the flow diagram. In such a case the function connected to input GATE_z will be modified:

$\text{GATE}_z = \sum_{i=0}^{n}$ where the Q_i mean the phases in which the (same) instruction D relative to output signal Z is located.

4.3 Introduction of the phase register structure

If the phases are coded in "1 out of n", then the part of the network structure characterized by the mapping $f_Q(F, G, Q)^* Q$ leads to the network shown in Fig. 3. The control Q_{iv} of the phase register cell is a set of control signals. The phase register cell can be divided into further network parts by a

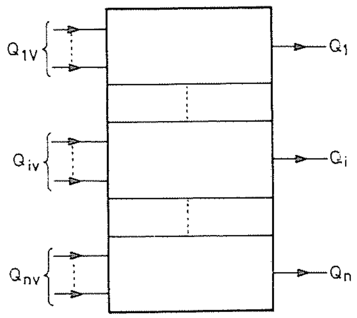

Fig. 3

suitable grouping of the control signals present in the set Q_{iv}. Such a partition can be seen in Fig. 4. To each function F belongs a so-called phase register element, being in itself an asynchronous sequential circuit similar to a flip-flop. The disjunction of the outputs of these phase register elements is the phase itself. The $\emptyset$–1 transition taking place at the output of function F sets the phase

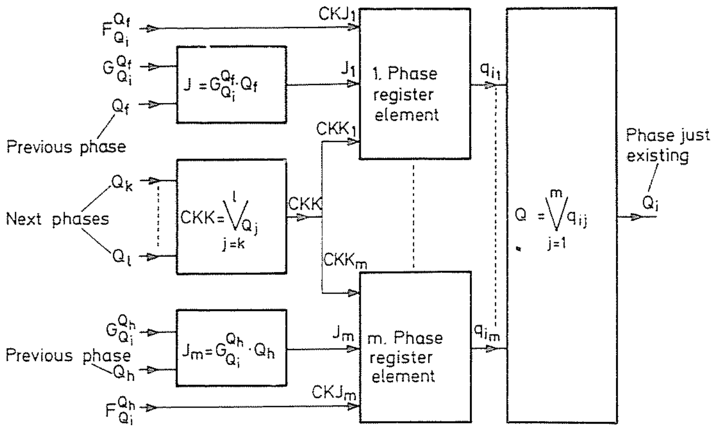

Fig. 4

register element assigned to it if the function G belonging to function F enables the phase transition. The occurrence of any phase following the previous one—i.e. the $\emptyset$–1 transition taking place in the value of signal Q meaning the phase—resets the phase register element. From the properties of the defined function pairs and from the code "1 out of n" of the phases, it follows that during the entire operation only one phase register element at a time is set in the network.

4.3.1 *Realization of the phase register element*

Based on the phase register structure according to Fig. 4 and on the flow diagram operation stated in the foregoing, the operation of the phase register element as an asynchronous sequential network can be described as follows:

Worded condition; Inputs: CKJ, CKK, J

Output: q

1) Output q of the network changes from $\emptyset$ to 1 if and only if input CKJ changes from $\emptyset$ to 1 and during this time there is 1 on input J; the value of input CKK is arbitrary, and even the transition 1–$\emptyset$ is permitted.
2) Output q of the network changes from 1 to $\emptyset$ if and only if input CKK changes from $\emptyset$ to 1; during this time input J may change arbitrarily, the value of input CKJ is arbitrary, and even the transition 1–$\emptyset$ is permitted.

A possible realization of the network is given in Fig. 5. This choice is justified by the fact that a part of the network agrees with the IC D flip-flop SN 7474. Instead of D flip-flops IC edge-sensitive J–K flip-flops, e.g., SN 74109 J–K flip-flop, can also be used. In this case the phase register element is simpler, as the J input already exists. The K input has to be connected to 1.

With phased coded "1 out of *n*", the network shown in Fig. 4 is guaranteed to be free from errors due to critical races and essential hazards. This is due to the fact that the phase register element as an independent sequential network is constructed free from hazards and critical races, and within the asynchronous phase register obtained by their coupling the individual elements may have a control permitted by the proper operation of each element. More specifically

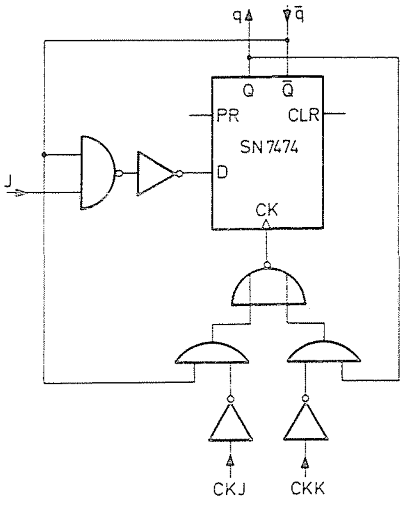

Fig. 5

this means that, apart from single changes of adjacent input combinations, only such multiple changes of input combinations may arise at the inputs of the phase register element which do not affect the operation of the individual phase register elements. If integrated circuits such as *D* flip-flops are used as phase register elements, then the asynchronous phase register network that solves the logical problem can be considered to have an operation independent of the speed. In other words: the delays of the individual building elements (gates, flip-flops) in relation to each other do not affect the proper operation of the whole network, since the proper delay conditions are built into the integrated circuit flip-flops by the factory. After this, one may ask what happens if the logical problem to be realized does not contain any essential hazard, but at the same time the sequential networks (phase register elements) as building blocks, obtained during decomposition, always contain it. In such a case the ability of the phase register "to be free from hazards" is not utilized. The check-up for essential hazards becomes superfluous just by the fact that—with the phase register structure applied—starting from any state of the network, essential hazards may be encountered, which, however, are always neutralized in the same way.

4.4 *An example*

The example chosen is the data transfer block of the control unit connecting a high-speed peripheral having two control signals and a channel IBM 360 working in the selector mode. The data transfer block is part of the channel-side control unit and is subordinated to it.

4.4.1 *Operation*

Input signals:

ZER:	General signal that sets the control unit into initial position
START:	It is displayed by the channel-side control unit when data transfer starts
RDY:	Displayed by the peripheral when it is ready for a new data transfer. An impulse with value 1
INFRDY:	The peripheral indicates by it the end of the byte transfer. An impulse with value 1
WR:	Write when its value is 1; read when its value is 0
PARF:	When it has the value 1, the byte coming from the channel and to be written has a parity error
UC:	When its value is 1, an error has arisen in the data transfer
SERO:	Service Out: a standard interface signal of IBM 360
CMDO:	Command Out: a standard interface signal of IBM 360

Output signals:

SERI:	Service In: a standard interface signal of IBM 360
INFST:	Its value 1 indicates to the peripheral that the data transfer block is ready to transfer the next byte
PH1:	With its value 1 the data transfer block indicates to the channel-side control unit that it is ready to perform data transfer

The output signals listed below are indications to the channel-side control unit. If any of them has the value 1, it takes over control from the data transfer block.

PH5:	The channel has found a parity error in the reading and, therefore, has stopped the data transfer by interface disconnect
PH7:	Data transfer stopped (byte counter full). The channel has responded with Command Out to Service In.
PHS:	The control unit has received data with parity error from the

channel and, therefore, the data transfer block stops transmitting data

PH11: The control unit has found an error in the data transfer and, therefore, stops it.

Other conditions:
— Single input changes only
— The channel-side control unit raises the signal START only if PH1 = 1
— The data transfer speed of the channel is higher than that of the peripherals, i.e., the signal exchange SERI-SERO is sure to take place between two INFRDY impulses
— In the initial state of the data transfer block, the output signal PH1 is 1, the others are ∅. Then, all input signals are ∅, except for the ZER signal, which is 1, and for the WR signal, which may have an arbitrary value.

4.4.2. Construction of the flow diagram

The flow diagram describing the operation of the network to be designed is shown in Fig. 6. The instructions are identified by their graphic symbols according to their types. The output signals beginning with PH are not given separately in the diagram, they agree with the corresponding phase output.

4.4.3 Formal check of the flow diagram

The properties required of the flow diagram functions are sure to be present in the case of this simple example, for the functions to be examined by pairs are either not defined on identical input signals or they are complementary.

4.4.4 Network realization

The control functions of the phase register of the designed network are shown in Fig. 7.a. The initial state of the network is set via the Preset and Clear inputs of the IC flip-flops of the phase register elements.

A possibility of simplification not seen directly from the structure of the network shown in Fig. 4 is as follows:

If the same function *F* triggers the network from given phases into the same next phase under the same conditions, then the next phase can be produced by a single phase under the same conditions, then the next phase can be produced by a single phase register element to the *J* input to which the disjunction of the given phase variables is connected. This could be done with phases PH1, PH7 and PH11, with the latter only because the enabling functions (UC) of the triggering functions also are identical.

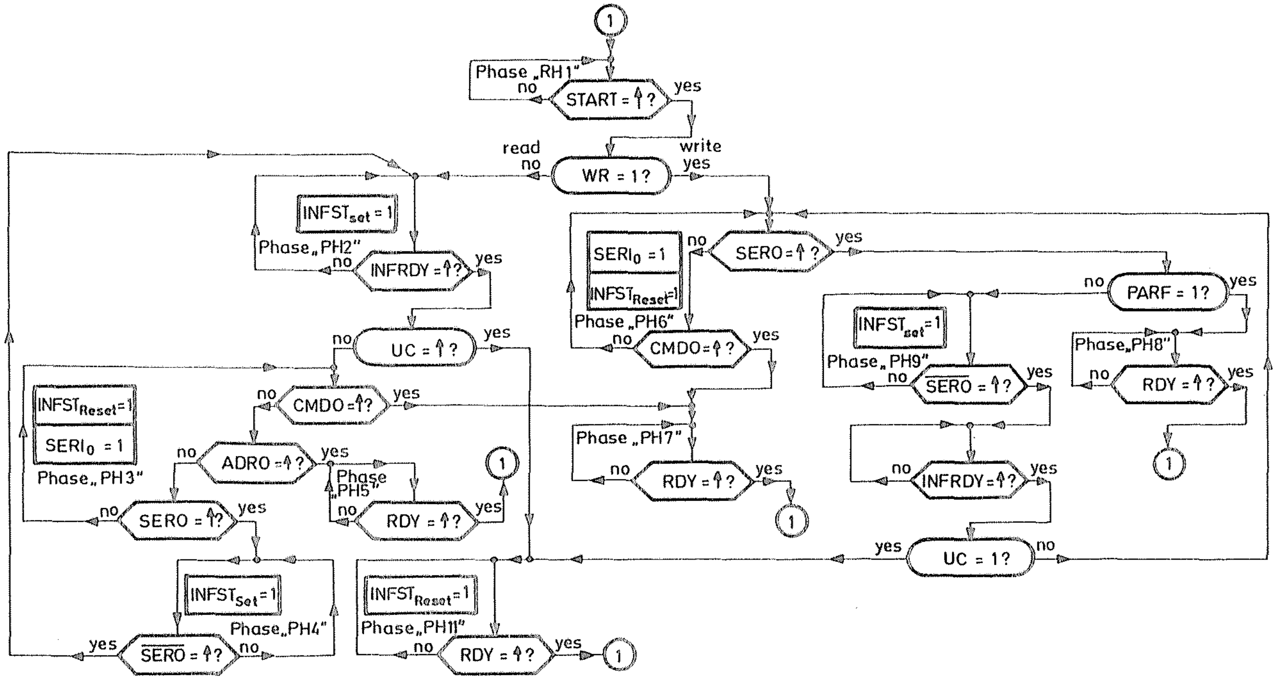
1
Phase „RH1"
START = ↑ ?
WR = 1 ?
read
write
INFST$_{set}$ = 1
Phase „PH2"
INFRDY = ↑ ?
UC = ↑ ?
CMDO = ↑ ?
ADRO = ↑ ?
SERO = ↑ ?
INFST$_{Reset}$ = 1
SERI$_0$ = 1
Phase „PH3"
Phase „PH5"
RDY = ↑ ?
INFST$_{Set}$ = 1
Phase „PH4"
$\overline{SERO}$ = ↑ ?
Phase „PH11"
SERO = ↑ ?
Phase „PH6"
Phase „PH7"
PARF = 1?
Phase „PH8"
Phase „PH9"
INFRDY = ↑ ?
UC = 1?
yes
no

Fig. 6

Phase	Phase reg element	J	CKJ	CKK	Preset	Clear
PH1	1	PH5 · PH7 + PH8 + PH11	ROY	PH2 + PH6	$\overline{\text{ZER}}$	„1"
PH2	1	PH1 · $\overline{\text{WR}}$	START	PH3 + PH11	„1"	$\overline{\text{ZER}}$
	2	PH4	$\overline{\text{SERO}}$		„1"	
PH3	1	PH2 $\overline{\text{UC}}$	INFROY	PH4 + PH5 + PH7	„1"	$\overline{\text{ZER}}$
PH4	1	PH3	SERO	PH2	„1"	$\overline{\text{ZER}}$
PH5	1	PH3	ADRO	PH1	„1"	$\overline{\text{ZER}}$
PH6	1	PH1 · WR	START	PH7 + PH8 + PH9	„1"	$\overline{\text{ZER}}$
	2	PH10 · $\overline{\text{UC}}$	INFRDY		„1"	$\overline{\text{ZER}}$
PH7	1	PH3 + PH6	CMDO	PH1	„1"	$\overline{\text{ZER}}$
PH8	1	PH6 · PARF	SERO	PH1	„1"	$\overline{\text{ZER}}$
PH9	1	PH6 · $\overline{\text{PARF}}$	SERO	PH10	„1"	$\overline{\text{ZER}}$
PH10	1	PH9	$\overline{\text{SERO}}$	PH6 + PH11	„1"	$\overline{\text{ZER}}$
PH11	1	(PH2 + PH10) · UC	INFRDY	PH1	„1"	$\overline{\text{ZER}}$

a)

$$\text{INFST}_{\text{Set}} = \text{PH2} + \text{PH4} + \text{PH9} \qquad \text{SERI} = \text{PH3} + \text{PH6}$$
$$\text{INFST}_{\text{Reset}} = \text{PH3} + \text{PH6} + \text{PH11}$$

b)

Fig. 7

The control functions of the output signals are shown in Fig. 7.b, and the scheme of the network in Fig. 8. For simplicity, the Preset and Clear inputs are not indicated in this figure.

5. Consideration of multiple input changes

The inputs that are able to change simultaneously can also be taken into account, and, with certain restrictions (e.g., the use of running clock signals, application of auxiliary networks), there exist realizations for such cases, too: This is possible because the *J* input of the phase register element agrees with the *D* input of the *D* flip-flop, and its CKJ input with the input of the flip-flop clock signal, and thus the synchronizing property of the *D* flip-flop can be utilized. Furthermore, input signals changing asynchronously produce multiple input changes. In general, the sequence of change of these signals is not known. The same concept is also meant by the terms "asynchronous signal change" and "asynchronism of signals". These can also refer to the outputs of logical functions if the functions depend on signals changing asynchronously. The term "asynchronous version of the flow diagram instruction" means that the value of the function associated with it may change asynchronously with

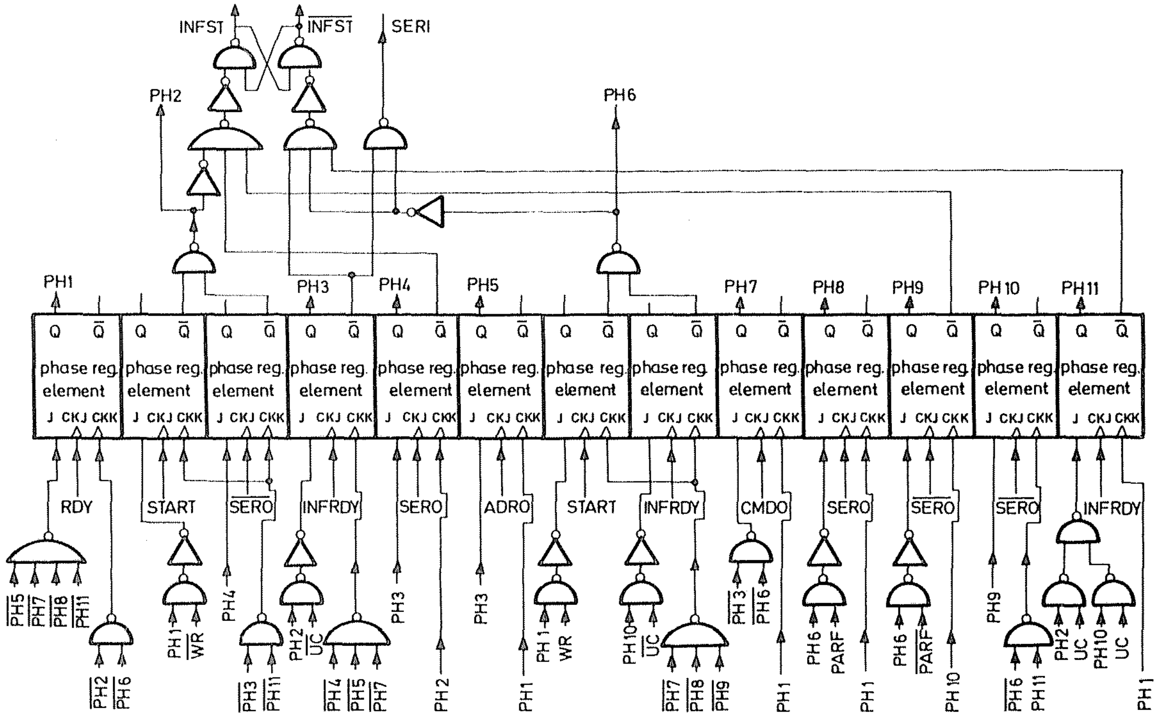
INFST
$\overline{INFST}$
SERI
PH2
PH6
PH1
PH3
PH4
PH5
PH7
PH8
PH9
PH10
PH11
Q
$\overline{Q}$
phase reg. element
J CKJ CKK
RDY
START
$\overline{SERO}$
INFRDY
SERO
ADRO
CMDO
$\overline{PH5}$ $\overline{PH7}$ $\overline{PH8}$ $\overline{PH11}$
$\overline{PH2}$ $\overline{PH6}$
PH1 $\overline{WR}$
PH4
$\overline{PH3}$ $\overline{PH11}$
PH2 $\overline{UC}$
$\overline{PH4}$ $\overline{PH5}$ $\overline{PH7}$
PH3
PH2
PH1 WR
PH10 $\overline{UC}$
$\overline{PH7}$ $\overline{PH8}$ $\overline{PH9}$
$\overline{PH3}$ $\overline{PH6}$
PH6 PARF
PH6 $\overline{PARF}$
PH10
PH9
$\overline{PH6}$ PH11
PH2 UC
PH10 UC

Fig. 8

respect to another function of the flow diagram. The term "asynchronous flow diagram instruction" has the same meaning. A possible approach to the problems of multiple input changes is to group the asynchronous signal changes according to their effects on the logical network.

5.1 Alignment of signals (trivial asynchronism)

The change of signals varying asynchronously with respect to each other does not affect the operation of the network. However, when a value complex arises that produces a change in the value of an F or G function, then this value complex will exist as long as desired by the network structure. In other words, the outputs of functions F and G continue to change as if only single input changes occurred at the network input.

The signals changing in the way described above do not affect the symbol system of the flow diagram, since their asynchronism does not appear at the outputs of functions F and G.

5.2 Initialization

Certain signal changes cause the network to assume its initial position irrespective of how the other input signals change at the same time. It will remain in the initial position, irrespective of how the other input signals change, until the initializing signal change anew. Thus, from among the signals changing asynchronously, those initializing the network have absolute priority over the others.

Initialization is indicated in the flow diagram by means of the initialization instruction, in which A means a logical function of the input signals of the network. If $A = 1$, the network is set into its initial position, irrespective of the output changes of the other instruction functions (F and G). (See Fig. 9). The initial phase can end when A becomes $\emptyset$. The "YES" branch of the instruction leads to the initial phase, and its "NO" branch to a triggering instruction which takes part in the determination of that phase in which the initializing instruction occurs. As function A has absolute priority over the other functions, it must be present in each phase except where the verbal specification of the task to be solved indicates that in certain phases function A will never be "1". In the initial phase itself, the initializing instruction means that the "YES" branch ($A = 1$) maintains the phase independently of the changes in the value of function F of the triggering instructions, while the "NO" branch ($A = \emptyset$) enables the phase transitions.

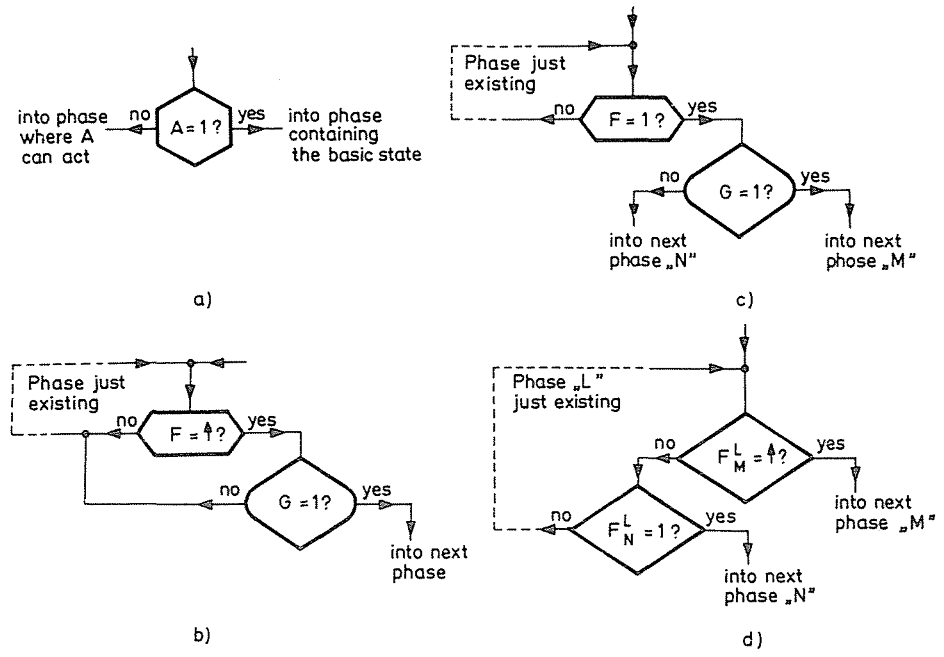

Fig. 9

5.3 Waiting for the asynchronous signal with clock signal

This is the case where *G* changes asynchronously with respect to its associated function *F* and the "NO" branch of function *G* does not lead to a new phase (see Fig. 9.b). The asynchronism of the two functions with respect to each other indicates a corresponding asynchronism of two or more input signals. The term derives from the fact that similar tasks often occur when networks are being synchronized.

5.4 Branching from an asynchronous signal

The difference to the previous section is that the "NO" branch of function *G* also leads to a new phase (see Fig. 9.c). In other words: the phase transition produced by the transition ∅–1 arising on the output of function *F* depends on the instantaneous value of function *G*. Formally, the previous section can be regarded as a special case of this section. The distinction is justified by the essential differences in realization that will be discussed later.

5.5 Asynchronism of the signals causing a phase transition

The outputs of the functions *F* taking part in the development of the given phase may change from ∅ to 1 asynchronously with respect to each other as a result of multiple input changes (see Fig. 9.d). In the network, the next phase is determined by the first transition ∅–1 just arising on the output of function *F*. The asynchronism of a given function *F* is to be interpreted as follows: Its output may change from ∅ to 1 with respect to a similar change of the other functions *F* taking part in the development of the phase, but cannot change asynchronously with respect to a similar change of functions *F* taking part in the development of the phases that precede and follow the given phase. All of the changes mentioned in the previous sentence are enabled by a corresponding function *G*. Thus, the particular function *F* whose output changes asynchronously from ∅ to 1 only when its function *G* does not enable these changes is not considered to be asynchronously changing. An asynchronous trigger instruction means the asynchronism of at least two functions *F* with respect to each other. The asynchronism of a given function *F* alone does not mean that it can change asynchronously with respect to its associated function *G*, since the asynchronism of these function pairs is taken into account according to Sections 5.3 and 5.4.

The asynchronism of the functions *F* of two phases following one another is not enabled because this would be meaningless from the viewpoint of the output combination sequence. Otherwise, some output signals could last for a very short period of time, for ∅ time in principle, which is not permissible. From the viewpoint of the designer, all of the functions *F* whose asynchronism with respect to each other must be reckoned with in the given situation, have to be placed in a single phase.

These cases of asynchronism do not affect the procedure of checking the flow diagram, as mentioned in Section 3, except that also functions A must also be considered when functions *F* are examined.

Realization possibilities of the cases of asynchronism

The trivial asynchronism does not affect the realization by its character.

The initializing instruction is implemented by use of the Preset and Clear inputs of the IC flip-flops built into the phase register elements. If the phase is not initial, the network for the function A located there has to be connected to the Clear input of the phase register elements constituting the phase. If the phase is initial, then the A network located there has to be connected to the Preset input of the phase register elements constituting the phase. Owing to the disjunction of the outputs of the phase register elements, it is sufficient to control only a single Preset input. The Preset and Clear inputs of the IC flip-flops have priority over the other inputs. Thereby, the function A of the initializing instruction assumes priority, considering that there is no other sequential part in the asynchronous phase register network. The $S-R$ and $D-G$ flip-flops placed in the output part are controlled by means of the phases, so that they, too, assume their initial positions. When initialization arises during a multiple input change, then it assumes priority for similar reasons, but freedom from "peaks" on the network outputs is not guaranteed. In the same way, at the cessation of initialization, a phase transition due to a multiple input change may occur.

Following from its character, the phase register solves the case termed "waiting for the asynchronous signal with clock signal" (see 5.3).

In the phase register elements, function G controls the D input of the IC D flip-flops, and function F controls the Clock input. Thus, in fact the D flip-flop samples the instantaneous value of function G. One of the main applications of such flip-flops is the solution of similar synchronizing problems in networks built on different principles. The present case is a case of sampling: when the $\emptyset-1$ transition occurs on the output of function F, the D flip-flop in the phase register element either triggers or not, depending on the instantaneous value of function G just changing. The solution of this problem of asynchronism applies with certain restrictions. As is known from literature [17, 18], if the D input of a D flip-flop changes just when the Clock input changes from $\emptyset$ to 1, the D flip-flop may oscillate. Nothing can be said about the duration of the self-excitation, i.e. it may last for a very long period. It has been demonstrated in the literature that the self-excitation has an inherent cause, i.e., it is not caused by technological or logical design errors. From this it follows that a logical network which contains a synchronizing part made necessary by its purpose may work incorrectly, no matter which principle it is built on; this property cannot be regarded as a special disadvantage of the asynchronous phase register network being discussed. Proper functioning is guaranteed if the prescribed setup and hold times relative to the D input can be kept. As, for example in the SN74 series, this period lasts about 5 ns, the probability of an incorrect operation is not great, since, owing to other circuit delays, the input and output signal sequences of the networks are considerably slower.

As there is no generalizable method, a concrete example will illustrate the solution of the problem.

The case termed "branching from an asynchronous signal" and discussed in Section 5.4 cannot be solved by means of a single asynchronous phase register network because there is a danger that either both or neither of two phase register elements taking part in producing different phases will be set, since either G or $\bar{G}$ of the same function G is connected to their inputs J.

The case termed "asynchronism of signals causing phase transition" and discussed in Section 5.5 cannot be solved by a single asynchronous phase register network, for the coincidence of the edges arising on the outputs of functions F may result in several phases being set. Fig. 10 shows the simplified flow diagram of the channel selection register of a control unit. The control unit can work simultaneously with two IBM 360 channels. When it is transferring data over one channel at the moment when the other requests an initial selection, then it responds to the second one with a short busy sequence, then, after having finished the data transfer, indicates by service request that it is free. If it does not communicate with any of the channels, then it grants data transfer to the first of the initial selection initiatives. The connection of the channels and of two signal flow controllers of the interface capable of operating simultaneously is performed by the channel selection register. It has two output signals. A = 1 indicates that channel A is transferring data, B = 1 indicates the same for channel B. The value combination A = B = 1 is excluded while A = B

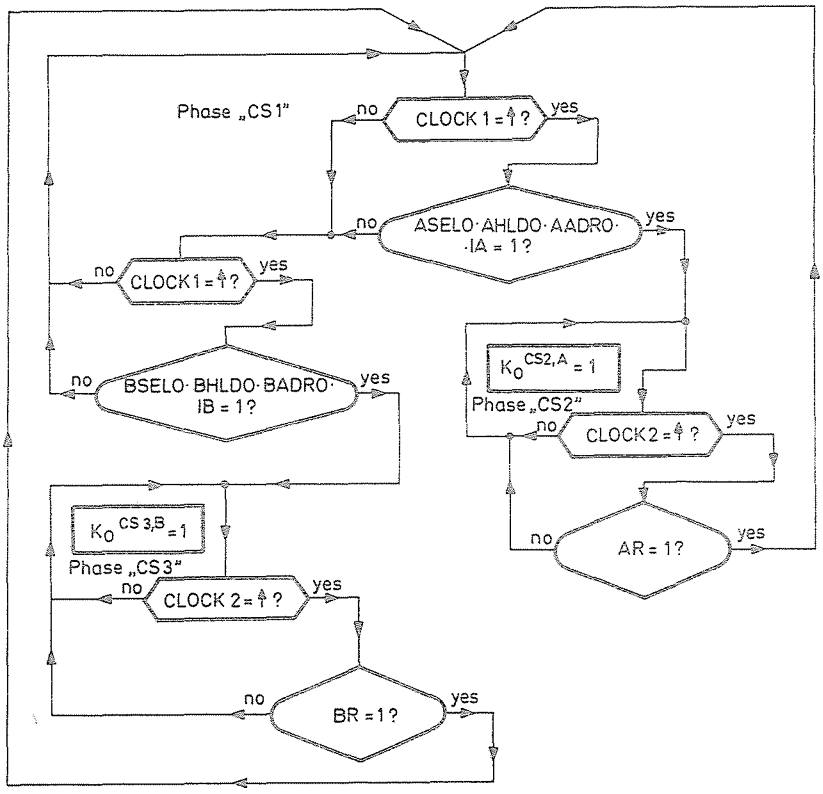

Fig. 10

=∅ indicates the initial state of the control unit, i.e. readiness for the initiation of the channels. In Fig. 1∅ the interface signals of channel A begin with the letter A, those of channel B with the letter B. Signals IA and IB indicate the recognition of the addresses sent by channel A and channel B, respectively. Signals AR and BR are generated by the controller of the data transfer flow after transfer of the final status. As there is no connection of any kind between the two computers, except for the interface, the beginning of the initial selection sequences started by the channels may coincide in time. This makes it necessary to sample the channel signals controlling the selection in the asynchronous triggering instructions in phase CS1.

The solution of the problem is shown in Fig.11.a. The solution of the asynchronism problem requires two-phase running clock signals, which are generated from the square CLOCK signal by the network shown in Fig. 11.b. The solution, in fact, leads to a synchronous network, since the phase transitions are controlled by the two-phase clock signal. The role of CLOCK 1 is unique. CLOCK 2 is necessary because the signals AR and BR may appear at any time with respect to CLOCK 1. The condition of proper operation is that the period of the two-phase clock signals be at most half of the duration of the

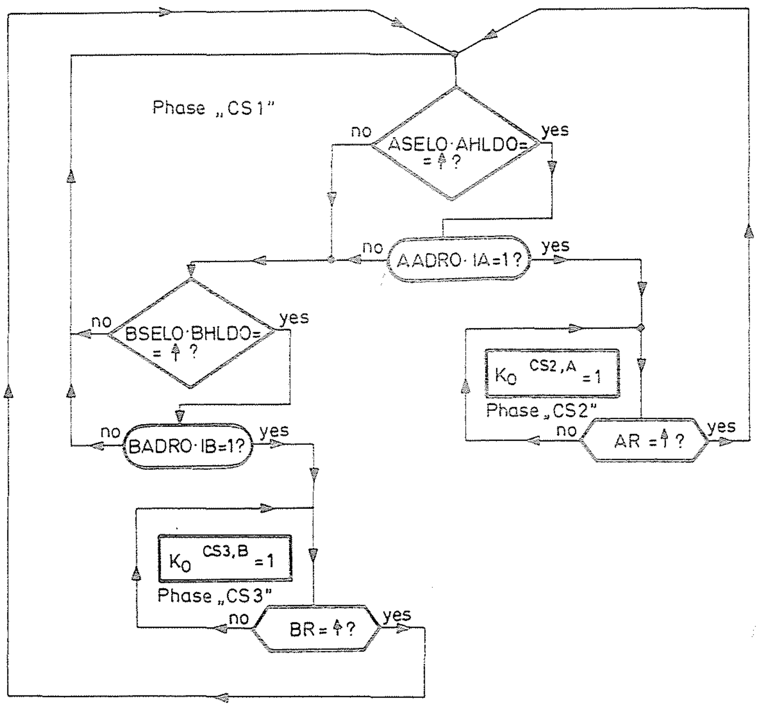

Fig. 11.a

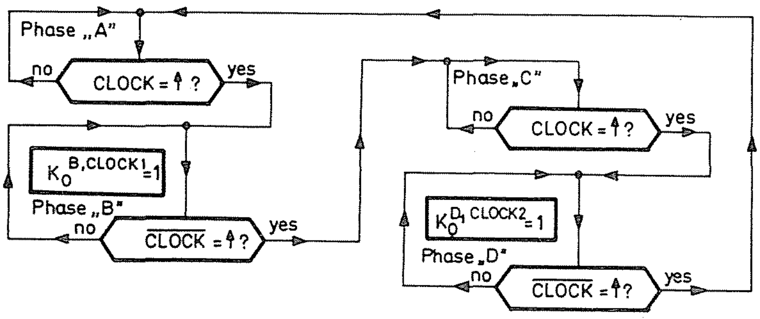

Fig. 11.b

other input signals. CLOCK 2 could be omitted if a startable clock signal generator would have been used, whose output signal could substitute for CLOCK 1. It could be started by the appearance of the phase CS1.

As the flow diagrams shown in Fig. 11.a and 11.b contain only the cases of asynchronism mentioned in Section 5.3, they are realized in the same way as in Example 4.4 (Section 4.4.5).

Reference [4] also discusses the co-operation of asynchronous phase register networks. To co-operate with equipment build on different principles, clocked (synchronous) flip-flops and monostable, too, can be used in the network part producing the output signals.

Conclusions

The steps of the design process can be summarized as follows:
— Construction of the flow diagram on the basis of a verbal or other specification of the problem.
— Checking the properties of functions F and G of the flow diagram. If any of them is missing, modification of the flow diagram must be done according to a new interpretation of the specification.
— Checking the properties of functions K of the flow diagram. In case of any error, modification of the flow diagram and repeated execution of the previous step have to be performed.
— Based on the flow diagram, and according to the network structure shown in Fig. 4, determination of the control functions of the phase register.
— Determination of the control functions of the output part.
The discussed design method has the following features:
— The operation is described by means of a flow diagram, which is the basis of the design.

— The network structure is based on a phase register.
— It is possible to take multiple input changes into account.
— Once the flow diagram has been constructed, the design work is completely systematic.
— The network is free from errors due to essential hazards and critical race conditions, which does not need any special examinations since it is ensured by the ICs present in the phase register elements.
— The network realizes can be considered speed-independent. Here, this means that the signal-delaying effect of the circuits of the network with respect to each other does not affect the proper operation. Of course, the speed of the input signal sequence must not exceed the maximum operational speed determined by the longest path of signal transmission.

This latter property is very important, since today by dividing a global design into steps, the work of the logical designer and that of the production engineer tend to become more and more separated. Thus, the designer has less and less influence upon the topology and arrangement of the IC boards, and upon the solution of the buffering problems caused by the dimensions. Another difficulty arises from the increasing number of IC manufacturers and from changing purchasing possibilities. Switching from one IC source to another may cause problems because of deviations in delay values.

As for the fields of application, the design method discussed can be used in constructions involving a great number of input signals and output signals and long control sequences (a large number of phases), where only a small number of input signals is relevant at any time and where it is not necessary to branch from one phase into many different directions. For the functions of the flow diagram this means that each function depends on a few input signals and does not contain many terms. For the phases it means that their composition does not involve many functions *F*. Tasks of this kind have a rather wide range, owing to the general trend that control sequences are executed with the use of control signals not coded in the binary system, but, partly or wholly, in the system of "1 out of n". This is the way the output and input units of computers and many peripherals are usually organized. In the control of production processes, too, the signals coming from physical sensors usually have independent logical effects and many of their combinations are excluded.

Acknowledgements

I thank Professor Dr. A Frigyes, Head of Department, for supporting and sponsoring the research work carried out to reach the results published in the present paper, and for his ensuring favourable working conditions to it.

For similar reasons I am grateful to Associate Professor Dr. P. Arató for his valuable advice and great help and Associate Professor Dr. P. Kalmár for his assistance.

I thank Professor Dr. H. M. Lipp and Professor Dr. J. Beister for their help in completing this paper.

References

1. LÁSZLÓ, Z.–ARATÓ, P.: Microprogrammed Logic Network Design. Microprocessors, *2*, 73 (1978)
2. ARATÓ, P.–GRANTNER, J.–KALMÁR, P.–TERPLÁN, S.: Methoden des logischen Entwurfs auf der Grundlage von Ablaufplänen. Z. elektr. Inform. & Energietechnik, Leipzig *7* 426 (1977)
3. KALMÁR, P.: A phase-state reduction and assigment method based on the flow chart for the logical design of control units. Periodica Polytechnica vol. 20. No. 3. 1976
4. TERPLÁN, S.: A Method of Designing Asynchronous Logic Networks, Based on Flow Diagram (in Hungarian) Doctoral dissertation, 1980, Budapest Technical University
5. UNGER, S. H.: Asynchronous Sequential Switching Circuits. Wiley-Interscience, New York, 1969
6. McCLUSKEY, E. J.: Introduction to the Theory of Switching Circuits. New York: McGraw-Hill 1965
7. CHUANG, Y. H.: Transition Logic Circuits and a Synthesis Method. IEEE Transactions on Computers, vol. C-18, No. 2, February 1969. p. 154
8. SMITH, J. R. Jr.–ROTH, CH. H. Jr.: Analysis and Synthesis of Asynchronous Sequential Networks Using Edge-Sensitive Flip-Flops. IEEE Transactions on Computers, vol. C-20, No. 8, August. 1971. p. 847
9. BREDESON, J. K.–HULINA, P. T.: Synthesis of Multiple-Input Change Asynchronous Circuits Using Transition-Sensitive Flip-Flops. IEEE Transactions on Computers, vol. C-22, No. 5. May 1973. p. 524
10. KIRCHNER, R.: Asynchrone Schaltwerke mit Flankensteuerung. Dissertation, Universität Karlsruhe, 1978
11. DAVID, R.: Modular Designing of Asynchronous Circuits Defined by Graphs. IEEE Transactions on Computers, vol. C-26, No. 8, August. 1977. p. 727
12. SCHMITT, E.–WENDT, S.: Critical Feedback Analysis of Clocked Digital Systems. NTZ, Heft 4. 1972. S. 196
13. BEISTER, J.: Ein Totzeitmodell für asynchrone Schaltwerke. NTZ 28, Heft 1, 1975. S. 13
14. BEISTER, J.: A Unified Approach to Combinational Hazards. IEEE Transactions on Computers, vol. C-23, No. 6, June 1974. p. 565
15. DITZINGER, A.–LIPP, H. M.: Asynchrone Schaltwerke mit dem FPLA IM 5200. Applikationsbericht T. U. Karlsruhe
16. DITZINGER, A.–LIPP, H. M.: Use of Memories and Programmable Logic Arrays for Asynchronous Sequential Circuits. Computer and Digital Techniques, October 1979, *2*, No. 5
17. PECHOUCEK, M.: Anomalous Response Times of Input Synchronizers. IEEE Transactions on Computers, vol. C-25, No. 2, February 1976. p. 133
18. CHANEY, TH. J.–MOLNAR, CH. E.: Anomalous Behavior of Synchronizer and Arbiter Circuits. IEEE Transactions on Computers, vol. C-22, No. 4, 1973. p. 421
19. The TTL Data Book for Design Engineers CC-411C. Texas Instruments 1975

Sándor TERPLÁN H-1521 Budapest