COO-2118-0014

Report No. 463

CONTROL POINT DESIGN USING MODULAR LOGIC

by

Christian A. Rey

June 1970

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Atomic Energy Commission, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

Department of Computer Science

University of Illinois

Urbana, Illinois 61801

This work was supported by Contract AT(11-1)-2118 with the U. S. Atomic Energy Commission.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Abstract

The purpose of this note is to introduce 4 basic building blocks, or modules, with which one can very easily convert a flow-chart design into a logical network which accomplishes the same coordination of tasks. The implementation utilizes an asynchronous or pseudo-asynchronous "control point" strategy. [1]

The conversion (flow-chart $\rightarrow$ network) is systematic in that one "box" in the flow-chart corresponds to one module. The problem of minimizing the circuit realization of a given coordination is therefore reflected back into the related problem of "minimizing the flow-chart" representing this coordination. Preliminary remarks on this latter process are outlined at the end of the paper.

TABLE OF CONTENTS

	Page
1. Conventions	1
2. Flow-chart conversion	6
3. Four basic modules	7
3.1 The D0-module	7
3.2 The Sequence-module	10
3.3 The Wait-module	10
3.3.1 The W2-module	10
3.3.2 The W1-module	10
3.3.3 Properties and applications of the Wj(n)-module	12
4. Properties and applications of the four basic modules	14
4.1 Asynchronous mode	14
4.2 Pseudo-asynchronous mode	14
4.3 Control sequenced by external signal	17
4.4 Costs	19
4.5 Example	20
5. Simplification of logical network	25
5.1 Unconditional parallel chains	25
5.2 Cascaded Sequence modules	27
5.3 Cascaded Wait modules	29
5.4 Conditional Task	31
5.5 Example	31
6. Conclusion	34

1. Conventions

In our flow-charts, 4 different symbol boxes are used:

- the task box:

- the decision box (the two directional branch):

- the circle:

- the dotted circle:

- the fan-out point:

The task box represents a task to be performed. We mean by "task" a set of "elementary micro-operations" to be done in parallel, (Figure 1). For instance,

"add register R2 to R1"

is a task; similarly,

"call routine SUB and set flip-flop F"

is considered as one task.

It is important to note that we exclude the conditional micro operations from characterizing a task. For example,

"reset F1 if $\alpha = 0$ or reset F2 if $\alpha = 1$ "

is not a task, although the individual operations

"reset F1" and "reset F2".

are tasks. Exceptions to this rule are described in section 5.4.

The decision box (the two-directional branch) has the following meaning: when the control flow reaches the decision

Figure 1 Task Conventions

box, it is directed to the "1 subbranch" if the condition within the box is satisfied and to the "0 subbranch" otherwise. An important assumption is that the condition within the search box must be expressed by one and only one variable. This is not a restrictive assumption since we can always preprocess the conditional variables within the branch box in order to obtain a final variable X which is "1" if the condition is satisfied and "0" otherwise. Some examples are given in Figure 2.

The circle (Figure 3.a): if the control flow hits the circle from any one of the n incoming branches, then the control flow propagates through the outgoing branch.

The dotted circle, (Figure 3.b): if the control flow reaches all of the n incoming branches of a dotted circle, then and only then the flow propagates through the outgoing branch.

Typically, a dotted circle is used to wait for the accomplishment of two or more parallel tasks. Figure 3c shows a complex coordination where circles and dotted circles are combined.

The fan-out point is a point in the flow-chart, where a signal splits into m signals. This point corresponds in the actual design to the output of a gate with a fan-out greater than one. Figure 3d shows some conventions and equivalences concerning fan-out points, circles and dotted circles.

Several non-restrictive assumptions are made in what follows. For instance, we assume that the control signal is normally one and that it becomes a negative pulse (1-0-1) when it activates a device. (This assumption comes from the fact that only NAND gates are used in our design.) We also suppose that the flow-chart is "safe", i.e. it is free of any loops, etc. which could cause actual design malfunction.

Figure 2 Sequence Conventions

Figure 3 Circle and Dotted Circle Conventions

2. Flow chart conversion

What can we consider to be the simplest way to convert a flow-chart into an actual logic design? The response to this question should take into account factors such as:

- conversion duration
- operator qualifications
- ease of resulting logic drawing interpretation

Further considerations concerning the design itself are:

- speed of resulting logical network
- degree of modularity
- minimality (cost)

The idea of converting a flow-chart into actual logic using a "box-to-box" conversion scheme has several of the advantages emphasized above. In particular, it shortens drastically the design duration by giving the logic the same overall topology as the flow-chart. Moreover, since the conversion is systematic, the designer no longer needs to be an "artist" as before. As far as the quality of the design itself is concerned, its modularity is unquestionable, due to the presence of a limited number of building blocks. We will see later that a factor such as speed may be improved in the final logical design if some preprocessing of the flow-chart is done first.

It is now our purpose to introduce four basic logical hardware modules corresponding to the four basic flow-chart "boxes".

3. Four basic modules

3.1 The DO-module

This module is the logical equivalent of a task box initiated from n places (Figure 4).

There are two possible designs for the DO-modules, which correspond to two different modes: asynchronous and pseudo-asynchronous.

In the first case the DO-module, called DOA(n), does the following. When one of the advance-in lines $\overline{A_{i1}}, \dots, \overline{A_{in}}$ drops to "0", the memory element is set and the DO-module is said to be primed. When $\overline{A_{i1}}, \dots, \overline{A_{in}}$ return back to "1", the task signal $\overline{DO}$, becomes "0". This resets the whole system: $\overline{DO}$ goes back to "1". In this way a negative pulse appears at the $\overline{DO}$ and $\overline{A_0}$ lines. The length of this pulse depends primarily on the propagation delay through gate (3). In conclusion, the DOA(n) module sends a negative pulse on the $\overline{DO}$ and $\overline{A_0}$ lines when a negative pulse has been seen at one of the $\overline{A_{i1}}, \dots, \overline{A_{in}}$ lines.

In the second case, the DO-module, called DO(n), not only does what the DOA(n) does but also "simulates" the task duration with a "timing stage", as shown. When $\overline{DO}$ goes to "0", the upper input of gate (5) becomes "1" and C is charged. After a delay Δ depending on R and C , the second input of gate (5) becomes "1" thus making $\overline{A_0} = "0"$ which resets the whole system as seen previously. Again a negative pulse appears at the $\overline{DO}$ line and at the $\overline{A_0}$ line. The length of the $\overline{DO}$ pulse depends on Δ whereas that of the $\overline{A_0}$ pulse depends only on the propagation delay through the various gates.

(a) FLOWCHART

(b) ASYNCHRONOUS MODE

(c) PSEUDO-ASYNCHRONOUS MODE

Figure 4 DO-Modules

An interesting property of the DO-module is that the $\overline{Ai}$'s, $\overline{DO}$ and $\overline{Ao}$ signals are all of the same polarity (negative pulse), and thus can be used for the same purpose. For instance, the $\overline{Ao}$ line is normally used as the advance-in $\overline{Ai}$ of a next DO-module. This can also be done using the $\overline{DO}$ line. In this case, this signal acts as a calling signal and is, in general, used (as in software) by a main sequence (program) to initiate some other sequence (routine) which will later "return" using a reply signal, (Figure 11).

Another useful property is that Δ is selectable according to the choice of the capacitor C. It is shown [1] that Δ obeys the relation:

$$\Delta \text{ (in nanosecond)} = 1.5 \times C_{pf}$$

if $R = 8.2k$ (IC 7400 Series).

A more powerful property of the DO-module is that it exhibits the same topology as the flow-chart box which represents it: both have the same number of control (flow) input/output lines (n input lines : $\overline{Ail}, \dots, \overline{Ain}$; one output line : $\overline{Ao}$).

3.2 Sequence-module

This module assumes the function of a two-directional branch. According to external condition(s), it directs an input negative-going pulse towards one of two directions. The flow-chart and logical design of the Sequence-module, termed $S(n)$, are shown in Figure 5. Again, input and output signals are all of the same nature. Thus, a $S(n)$ -module is compatible with a $DO(n)$ -module and with other $S(n)$ -modules. Notice that the Sequence Condition, S , controlling the action of the module must be reached before any signal $\overline{S_i}$ hits gate (1). Also, there is the same number of control input/output lines in the flow-chart box as in the module (input S is an input condition).

3.3 Wait-module

3.3.1 W2-module

The third module to be introduced here is called the $W2(n)$ module. The flow-chart of this module together with its design is shown in Figure 6. When a negative pulse has appeared at all the lines $\overline{W_1}, \dots, \overline{W_n}$ all the flip-flops are set and the output gate (3) is sensitized to the Wait Condition W . When $W = "1"$, the $W2(n)$ -module resumes its cycle by resetting all the flip-flops, thus producing a negative pulse at line $\overline{W_o}$.

3.3.2 W1-module

Another module, called $W1(n)$ -module, is introduced here. It may appear to be redundant with one of those already defined, since it can be realized using a $S(n+1)$ -module. However, since the flow-chart configuration it implements is

(a) FLOWCHART

(b) LOGIC DESIGN

Figure 5 Sequence Module

(a) FLOWCHART

(b) LOGIC DESIGN

Figure 6 W2-Module

quite frequent, a cheaper design is proposed (Figure 7). When any of $\overline{W_1}, \dots, \overline{W_n}$ goes to "0" the flip-flop is set. If $W = "1"$, $\overline{W_0}$ drops to "0" thus resetting the flip-flop which in turn causes $\overline{W_0}$ itself to go back to "1".

Let us notice that if $n = 1$, a $W_1(1)$ -module is equivalent to a $W_2(1)$ -module. Since this case often occurs, we give the name W-module to this kind of module.

3.3.3 Properties and applications of the $W_j(n)$ -module

The W_1 and W_2 modules have a very similar structure and therefore exhibit very similar properties. First of all, inputs and outputs are negative pulses which give these modules a full compatibility with the modules previously defined. However, special attention must be given to studying the speed of such units as well as the output pulse length they produce. Assuming $W = "1"$, the propagation delay through a $W_j(n)$ unit is only a function of gates (1) and (3) whereas the pulse length is related to the delay in gates (1), (2) and (3). Hence it is advisable to use fast gates (1) and (3) and a slow gate (2) which will alone give the output pulse its required length. Typically if gates (1) and (3) are taken from the IC series 7400 and gate (2) is a 74L00, a propagation delay through the $W_j(n)$ unit of 20ns may be expected together with a pulse length of 50 ns at the output. Also, modules W_1 and W_2 have the same topology as the flow-chart they implement: same number of inputs ($\overline{W_1}, \dots, \overline{W_n}$) and outputs ($\overline{W_0}$) (W is a conditional input and $\overline{CC}$ is the "common clear" line used to reset the system at the very beginning).

(a) FLOWCHART

(b) $S(n+1)$ REALIZATION

(c) $W1(n)$ - MODULE

Figure 7 $W1$ -Module

4. Properties and applications of the four basic modules

It is shown in what follows that these blocks may be used to implement control logic using any of the three basic modes: asynchronous, pseudo-asynchronous or clocked. The main difference between these 3 modes is the way in which task boxes are implemented.

4.1 Asynchronous mode

In asynchronous logic, a task is activated when some reply from the previous task is received. Two cases are possible: this reply is a level (produced by a reply flip-flop) or it is a pulse (produced by the last DO-module in the routine. In the first case, the configuration of Figure 8b may be used efficiently if Δ is such that flip-flop F1 is reset by CALL SUB before F2 is set. It is important to notice that the same configuration may be used to call an entire routine or to do a simple micro operation.

When the reply signal is a pulse, the configuration of Figure 8d may be used. Note that in this case a DOA module is needed instead of a DO-module.

A more complicated situation is illustrated in Figure 9. Note that in this case, two replies are pulses (1-0-1), and one is a level. As said before, some preprocessing on these replies is necessary to obtain a single composite variable X which is "1" when all replies have been received and "0" otherwise. A W2-module may be used, as shown, to obtain this variable X.

4.2 Pseudo-asynchronous mode

In this mode, the DO-module (with timing stage) is used. As indicated by the mode name, a pseudo-reply is produced by this

(a) FLOWCHART

(b) DESIGN (REPLY = LEVEL)

(c) FLOWCHART

(d) DESIGN (REPLY = PULSE)

Figure 8 Calling Program Implementations

timing stage after a certain delay Δ which merely simulates duration of the task performed.

If this task is a micro operation, Δ can be easily computed and using a DO-module to perform the task is straight forward. If the task is an entire routine, its duration may be variable and hence it may be difficult to assign a specific duration Δ to this routine. In this case, the pseudo-asynchronous mode cannot be used and the problem is solved in an asynchronous manner, as shown in Figures 8 and 9.

4.3 Control sequenced by external signal

A slight variation of the previous mode can be devised with the task duration simulated by a clock instead of a timing stage. In this case, the system becomes synchronous and may be implemented as in Figure 10.

Figure 10 Operations Sequenced by External Signal

4.4 Costs

In Table 1, modules are listed together with the NAND gates and cost required to implement them. A cost is also computed based on the following costs: Inverter: 5 units; 2 input NAND: 6 units; 3 input NAND: 8 units; 4 input NAND: 13 units. (1 unit $\approx$ 5¢).

module	inverter	2-NAND	3-NAND	4-NAND	Cost/module
DO(1)	1	3	1	0	31
DO(2)	1	1	3	0	35
DO(3)	2	3	2	0	44
DO(4)	2	3	1	1	49
DOA(1)	0	2	1	0	20
DOA(2)	0	0	3	0	24
DOA(3)	1	2	2	0	33
DOA(4)	1	2	1	1	38
S(1)	2	2	0	0	22
S(2)	1	3	0	0	23
S(3)	1	2	1	0	25
S(4)	1	2	0	1	30
W=W1(1)	0	2	1	0	20
W1(2)	0	1	2	0	22
W1(3)	0	1	1	1	27
W2(2)	0	2	3	0	36
W2(3)	0	3	3	1	55

Table 1: Module List

For each of the modules in this table, the cost is computed for the cheapest design shown in Figure 11.

We are now able to compare the efficiency of this new method with the efficiency of any other method.

4.5 Example

To make this comparison meaningful, we have chosen to implement a routine which was already designed and implemented using the "artist" approach instead of the systematic approach presented here. The routine 'SAMPLE' occurs in the Beam Motion Control of ILLIAC III SMV Scan-Display Controller. The flow-chart is shown in Figure 12.

Use of the four basic modules described earlier to convert the flow-chart is quite straight forward. What one needs to do is just to try to match the flow-chart of each of the four basic modules with the segments of the flow-chart of the system to be realized. This is done in Figure 12.

Costs are computed in Table 2 for the new and old designs of SAMPLE routine (Figure 13).

Cost (Modular design):		Cost (Old design)	
2xDOA(1)	= 40	1xD0(2)	= 35
1xDOA(2)	= 24	8xD0(1)	= 248
1xD0(1)	= 31	8xinverters	= 40
1xD0(2)	= 35	10x2-NAND	= 60
3xW	= 60	2x3-NAND	= <u>16</u>
1xW1(2)	= 22	Total	= 399
3xS(1)	= <u>66</u>		
Total	= 278		

Table 2: Comparison of Old Design and New Design

(a) DO OR DOA MODULES

(b) SEQUENCE MODULES

(c) W1-MODULES

(d) W2-MODULES

Figure 11 Lowest Cost Designs for Modules

Figure 12 Flow chart of SAMPLE and its Modular Logic Segmentation

Figure 13 Old Design of SAMPLE

Table 2 shows that a substantial savings is realized as well in time as in money when the systematic method proposed in this paper is applied. Even greater savings can be made if the logic drawing resulting from the use of the modules is closely inspected for the presence of obvious redundant logic. Although the modules are individually optimal, a set of converted modules may not represent an optimal implementation. For this reason some of the most frequent flow-chart configurations are considered next with suggestions to optimize their realization.

5. Simplification of logical networks

Although the last example has shown that a considerable savings in time and money can be realized by applying the above systematic procedure, the design itself is not claimed to be "minimal" since it is only composed of locally minimal subnetworks. For example, another flow-chart representing the same routine might lead to a cheaper realization. It is, therefore, believed that some preprocessing must be done on the flow-chart in order for this one to lead to a "near minimal" realization.

Since a "box" in a flow-chart corresponds to a module in the design, it is reasonable to try first to minimize the number of such boxes. The essential rules for doing this are stated next.

5.1 Unconditional parallel chains

Unconditional parallel chains of tasks are sequences of task which are performed at the same time (in parallel). An example is shown in Figure 14 where three chains are entered at the same time. In order to reduce the amount of logic needed for such a coordination, it may be advisable to follow the next rule (the operator is supposed to have some information about the relative duration of each of the tasks mentioned in this figure before applying this rule: the number in parenthesis represents these durations).

Rule 1: If boxes located at the top or at the bottom of n parallel branches represent tasks which have approximately the same duration, these tasks can be merged into one box (and therefore be performed by only one DO, or DOA module). In our case,

(a) DESIGN A

THE NUMBER IN PARENTHESES
REPRESENTS ESTIMATED DURATION
OF TASK

(b) DESIGN B

tasks A, B and C have approximately the same duration, and so are merged into one box (duration = 11). Let us notice that if we had "duration of A" = "duration of B" = 10 and "duration of C" = 30, then we could have merged only A and B. This is done for tasks G, F and E.

5.2 Cascaded sequence modules

In the definition of a two-directional branch, we have said that conditions within branches-boxes were expressed with only one variable (C_i). This corresponds as it is shown next to a cheaper design. Figure 15 represents two designs implementing the same sequence. In design A, the condition $C_i = A + BC$ is, in fact, realized with a set of cascaded sequence modules, whereas in design B, variables A, B and C are preprocessed and the resulting variable X is the variable used in the sequence module. Design B is obviously cheaper than design A. This can be generalized in the following way. Let $f = f(A_1, A_2, \dots, A_n)$ be the boolean function characterizing the sequence condition. Let n be the number of variables in f and p the number of minterm in the expression of f . It can be shown that $1 \leq p \leq 2^{n-1}$. In Table 3, we evaluate designs A and B in terms of their cost as functions of n and p .

(a) DESIGN ①: THREE S(1)-MODULES NEEDED
COST = 66

(b) DESIGN ②: ONE S(1)-MODULE
PLUS ADDITIONAL LOGIC
COST = 39

Figure 15 Rule 2: Cascaded Sequence Modules

n	p	Design A	Design B	Cost of A	Cost of B
2	1	2xS(1)	S(1)+1x2inpN+1xinv	44	33
2	2	3xS(1)	S(1)+3x2 " "	66	40
3	1	3xS(1)	S(1)+1x3 " +1xinv	66	35
3	2	4xS(1)	S(1)+2x3 " +1x2inpN	88	44
3	3	6xS(1)	S(1)+4x3 " + "	132	46
3	4	7xS(1)	S(1)+4x3 " +1x4inpN	154	67

Table 3: Evaluation of Designs A and B

From this table we deduce:

Rule 2: When a set of sequence modules exists such that there exists one input line to this set and no more than two output lines, such a set may be replaced by only one sequence module.

This property can be generalized:

<u>Number of output lines in the set</u>	<u>Sufficient number of Sequence module</u>
2	1
3	2
n	n-1

5.3 Cascaded Wait-modules

A similar rule may be applied to cascaded Wait-modules.

Rule 3: In any case, cascaded Wait-modules may be replaced by only one fed by a preprocessed variable X. Figure 16 shows an example of application of this rule. It is interesting to notice that Rules 2 and 3 not only decrease the cost, but also increase the speed of the control.

Figure 16 Rule 3: Cascaded Wait Modules

5.4 Conditional Task

In the definition of a task (paragraph 1) we have excluded a case such as

"reset F1 if $\alpha = 0$ or F2 if $\alpha = 1$ "

because this task can be expanded using a two-directional branch and two task boxes (Figure 17). However, if the two tasks

"reset F1" "reset F2"

have approximately the same duration, such an expansion technique might not be advisable from an economy standpoint since it would result in the use of

one S(1)-module

two DO-modules

whereas one DO-module (plus additional "task logic") is sufficient as shown in Figure 17. Hence the following rule:

Rule 4: If boxes located at the output of a multi-directional branch correspond to tasks having approximately the same duration, these boxes and this branch may be merged into one single box representing a "conditional task".

5.5 Example

An example which shows how efficiently these rules may be applied is represented in Figure 18. The original flow-chart represents a cost of 407, whereas the final one has a cost of around 180 (part of the reduced logic is "transformed" into task logic).

Figure 17 Rule 4: Using Task Logic

Figure 18 Flow chart condensation using rules 1-4: An Example

6. Conclusion

We have presented a modular approach to the problem of designing control logic using the so-called "Control Point Technique." Compared to the "old" heuristic method, this approach has the advantage of being systematic and of leading to an economical realization.

For the last point to be achieved, some "preprocessing" of the initial flow-chart is sometimes necessary. A set of heuristics can be formulated to do that. Although these heuristics can be applied systematically, in certain cases it might not be advisable to apply them. For instance, applying Rule 1 can destroy a certain modularity in the design because of the merging of two parallel chains. Therefore, at this stage of the design, the skill and experience of the designer are still needed.

REFERENCE

- [1] Rey, Christian A., "Control Point Strategy and Its Automated Diagnosis", Department of Computer Science Report No. 450, June 1971.