THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 59, No. 9, November 1980
Printed in U.S.A.

A Memory Implementation of a Fully Programmable Digital Controller

By S. V. KARTALOPOULOS

(Manuscript received April 4, 1980)

The design of controllers, and recently of microprogrammed controllers, has been a subject of utmost importance in the development of information-processing machines, or computers. We present a simple implementation of a fully programmable electronic circuit controller using memories of a matrix type, in contrast to the digital controllers usually implemented with sophisticated sequential circuits with random logic or by a microprocessor. The controller consists of two memory parts and generates a set of bivalued waveforms. We develop here a method by which the instruction and program codes are extracted from the latter. The instruction codes are stored in one memory and the program codes in the other. Any change in one or more of the waveforms is easily made by redefining the program and/or instruction codes. The generated waveform is time-compressed or expanded by merely increasing or decreasing the clock frequency of the system. Both codes are entirely user-defined; the user may make the instruction code comprehensive and the program code very simple, or vice versa. We believe this technique reduces the design effort and provides bivalued waveforms with no error, according to the specifications of the application. One circuit fits a variety of applications with a variety of clock frequencies and may also be integrated on one monolithic chip and be reprogrammable if erasable memories are employed.

I. INTRODUCTION

An electronic interface or a data processor, here called a "stored program processing computer" (SPPC), usually consists of two types of circuits: the information circuits and the control circuits. Information circuits are those that store data permanently or temporarily and/or process information (data) performing either logic or algebraic operations. The control circuit, usually called the controller, comprises the

heart of the system and is responsible for the normal flow of information in a well-defined environment with preprescribed control functions in space and in time. The principle of a system is illustrated in Fig. 1. In this illustration, the controller is subject to external conditional signals according to which it controls (regulates) the flow of information through and from circuit to circuit (1 through 3 in Fig. 1). Each information circuit may have, theoretically, any number of inputs and any number of outputs, and each one may be connected with any other information circuit in one given direction or in both.

In this paper, the term SPPC applies to both analog or digital computer, assuming that the controller is digital, as is the usual case. In the design of an electronic system, a considerable amount of the design effort is consumed in the controller circuit. The controller usually is a sequential random logic circuit, and it should be simple, fast, and, above all, error-free or glitchless, thus making it a laborious design of state machines.[1,2] Additionally, in certain applications the controller should be flexible or programmable so that the controlling functions could be altered according to the different requirements of the particular application. In this case, the controller is implemented

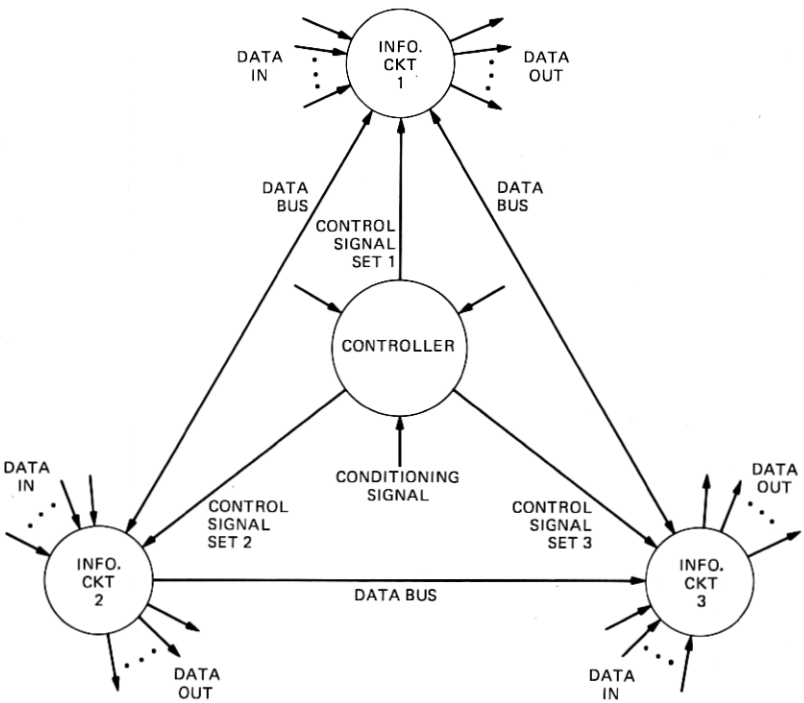

Fig. 1—A model of a data processor. The number of information circuits is not restricted to three, as in the figure.

with microprocessors, where the factor speed now becomes very important, particularly in real-time applications.

This paper presents a simple technique of generating a set of controlling functions that are well-defined beforehand in the time domain without requiring the knowledge of advanced sequential random logic circuit design. Additionally, the set of the generated controlling functions, in the form of bivalued waveforms, do not exhibit any racing conditions. At this point, it should be recognized that certain concepts of the methods presented here are similar to the concepts of microprogramming.[3]

II. THEORY

Consider a set of bivalued functions of time, i.e., functions that take values 0 or 1.

$$C = \{c_1(nT), c_2(nT), \cdots, c_k(nT)\}; \qquad n = 0, 1, \cdots, n \tag{1}$$

and a sequence of vectors

$$V = [v_n] \tag{2}$$

such that

$$v_n = \sum_{i=1}^{k} c_i(nT)2^{i-1}. \tag{3}$$

Consider that in the sequence of vectors V groups of adjacent vectors may be identified, $g_j(v_i, d_j)$, such that a new set of these unique groups of vectors may be formed

$$G = \{g_j(v_i, d_j)\}, \tag{4}$$

where d_j is the number of vectors in each group called the distance of the group. Each unique group of vectors is called a feature. Now assume that these features are placed in an order starting from location zero and that the starting location L_j of each group and the distance d_j of each group as they are orderly recorded are monitored, thus forming a new set of vectors $o_r(L_j, d_j)$

$$P = \{o_r(L_j, d_j)\}, \tag{5}$$

where each vector $o_r(L_j, d_j)$ consists of two parts, the first location L_j of the group g_j and the distance of the group d_j.

From these definitions, it is obvious that a unique correspondence between the elements of the sets G and P is defined by the particular application,

$$G \Leftrightarrow P.$$

The set of vectors G is denoted as the instruction code and the set of

vectors P as the program. Each vector of the sets G and P is denoted as the instruction microcode or microinstruction, and the program microcode or word, respectively.

III. THE PRINCIPLE OF THE METHOD

The method presented here, in principle, employs two matrix memories; the program code is stored in one memory and the instruction code is stored in the other. Thus the problem of designing a controller reduces to constructing the instruction and program codes from the predefined set of controlling bivalued functions. The latter is accomplished with the following steps: First, arrange the set of the controlling functions

$$C = \{c_i(nT)\}$$

in an order such that the sequence of vectors $[v_n]$ is derived and from this the common features are extracted so that the set of vectors

$$G = \{g_j(v_i, d_j)\}$$

is derived. The latter set comprises the instruction code. Second, place in an order the vectors g_j, starting from location zero (preferably) and record the starting location, L_j, of each vector g_j and its length, d_j, forming a new vector $o_r(L_j, d_j)$ for each vector recorded. Thus a new set of vectors is derived:

$$P = \{o_r(L_j, d_j)\}.$$

This set is the program code. Third, store the instruction code in one matrix memory and the program code in another matrix memory. These two memories will be called the instruction memory and the program memory. How these two memories are used together to implement a programmable controller is explained in detail in the description of the circuit. At this point, it is sufficient to say that the program memory addresses the first location of a feature and simultaneously defines the number of addresses (the distance) this feature takes in the instruction memory.

To generate a set of controlling functions, the program is read sequentially, each word of the program addresses a corresponding feature in the instruction memory and for d_j sequential addresses, thus generating the functions required at the k outputs of the latter memory.

IV. AN EXAMPLE

This technique is summarized with an example of four controlling functions as in Fig. 2. In this figure, four functions are assumed, C, and from them the set of vectors, V, is formed. For the latter, the features, F, are extracted and the program is written. Notice that, in general,

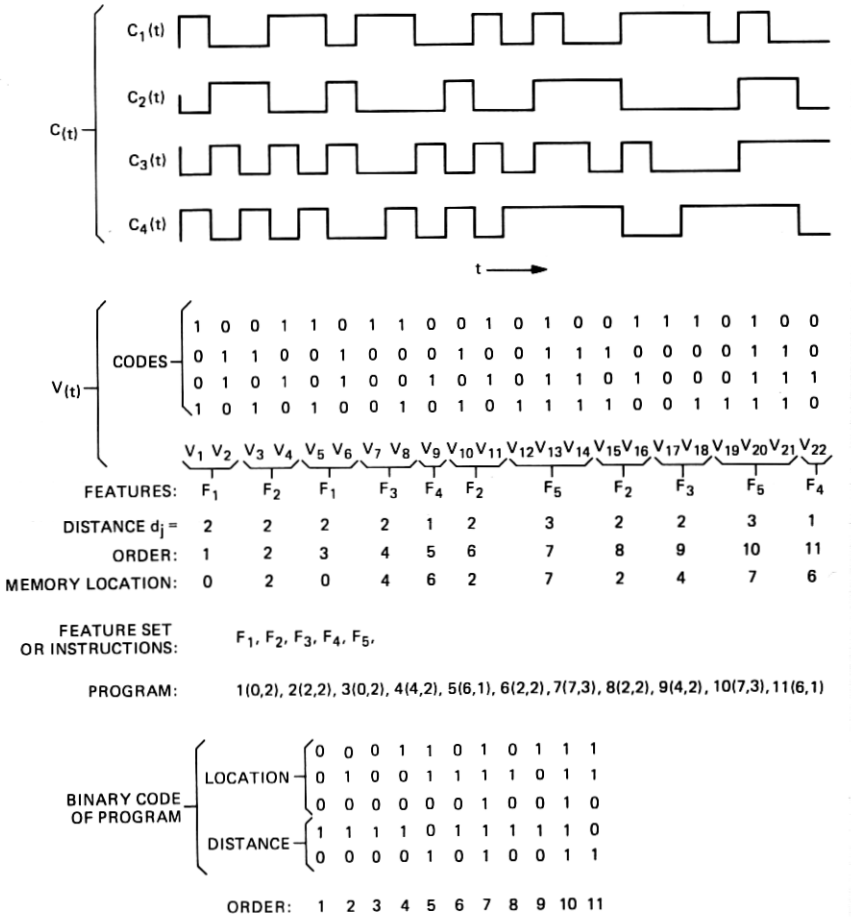

Fig. 2—An example to summarize the technique of converting the given signaling $C(t)$ into the instruction and program sets.

extraction of the features is constrained by the requirements and specifications of the particular application. As has already been said, a separate feature may be attributed to each individual vector. In this case, the set of features is equal to the sequence of vectors, V, and the program is a null set, i.e., it contains the location 0 as the first address of the program. Thus, in this case the program memory is redundant and in many applications such a case may be preferable. Although the latter case may require less memory elements, in applications where the features are permanent, such as in a microprocessor or microcomputer, the former case may be more attractive. Then the instruction memory may be a low cost nonerasable memory, such as a ROM (read only memory), while the program memory may be of an erasable type such as RAM (random-access memory), EPROM, etc. No special empha-

sis is given in any of the two cases here; a description of the technique is given as well as the variations it may have, depending on the requirements of the particular application.

V. CIRCUIT DESCRIPTION

Before we describe a more comprehensive circuit to implement the presented method, we give a brief description of the operation of a matrix memory for clarity.

A matrix memory consists of memory elements organized in l rows and k columns. A decoder circuit of $p = \sqrt{\log_2 l}$ inputs, where $\sqrt{x}$ means the lower integer above x, accepts a binary code of length p and, depending on its decimal value, it addresses (enables) the corresponding row so that the contents of the row appear at the k outputs of the memory. Symbolically, a matrix memory is illustrated in Fig. 3.

Now consider the circuit of Fig. 4. This consists of two matrix memories, the program memory and the instruction memory with the associated decoders, and three counter circuits, C-1, C-2, and C-3. The three gates A-1, A-2, and O-2 merely indicate the AND and OR operations and not the number of logic gates in the circuit. The counter C-1 is a binary counter of minimum length $\sqrt{\log_2 r}$, where r is the number of program words and constitutes the address of the program memory. The minimum number of rows of the program memory is also r. The counter circuit C-2 is a circuit that includes a variable modulo q binary counter, where $q = \sqrt{\log_2 d}$. The minimum value of the variable modulo counter is q_{max}. This counter circuit advances the binary counter, C-1, every $\langle d_j \rangle$ clock pulses and, for $\langle d_i \rangle \cdot T$ clock periods, enables the gate A-2 so that $\langle d_j \rangle$ clock pulses will be counted by the counter. C-3. The notation $\langle x \rangle$ means the decimal value of x, an integer number. The counter C-3 is a presettable binary counter of minimum length $\sqrt{\log_2(L_{max} + d_{max})}$. This counter is preset to the value $\langle L_j \rangle$ and counts $\langle d_j \rangle$ pulses which the counter C-2 enables, starting from $\langle L_j \rangle$ and

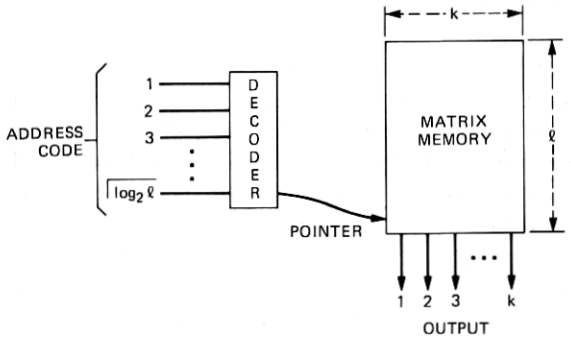

Fig. 3—Model of a matrix memory.

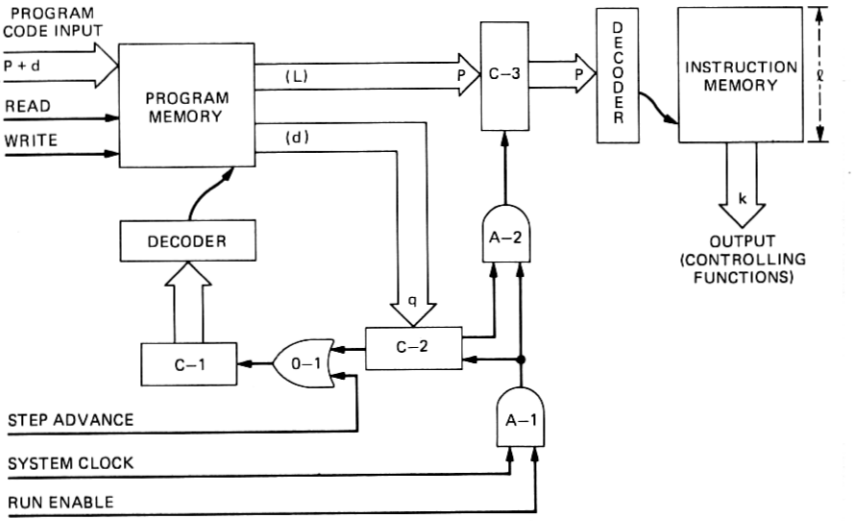

Fig. 4—A programmable controller with two matrix memories. The program and the instruction.

advancing to $\langle L_j \rangle + \langle d_j \rangle$, thus addressing the microinstructions stored in locations from $\langle L_j \rangle$ to $\langle L_j \rangle + \langle d_j \rangle$.

In brief, the counter circuit C-2 determines when the program memory will be advanced and for how many clock periods the counter C-3 will be counting, while counter C-3 provides the address of the corresponding instructions to be read out.

The abovedescribed circuit is greatly simplified in the case where the set of features $F(v_i)$ equals the set of vectors V. Then, as has already been mentioned, the program is the null set and the program

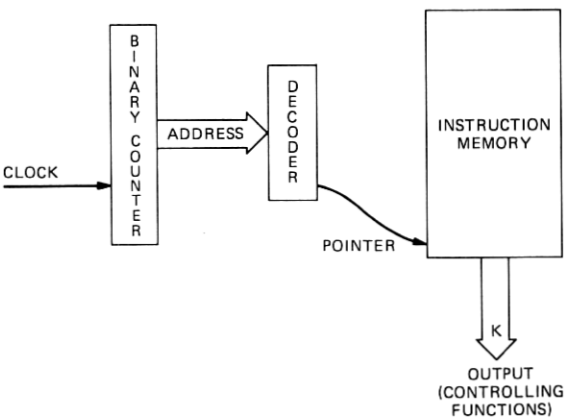

Fig. 5—A reduced-to-one matrix memory programmable controller. Here, the program set may be considered a null set.

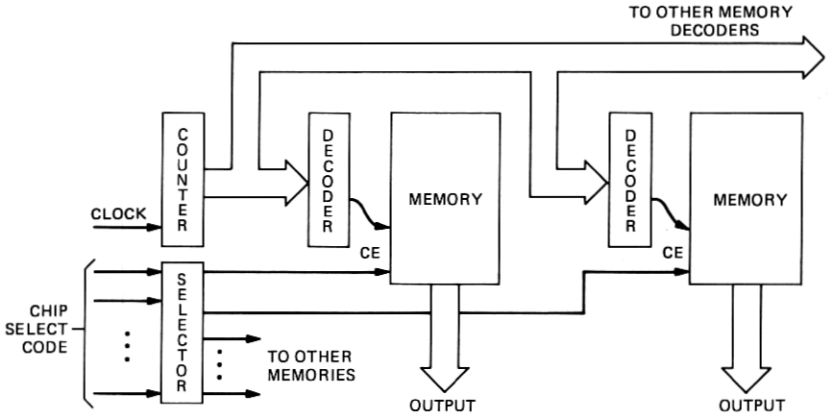

Fig. 6—More than one instruction and/or program sets may be incorporated by paralleling matrix memories.

memory is redundant. In this case, the circuit reduces to only a counter and the instruction memory, as illustrated in Fig. 5. Here it is said that the set of vectors is mapped into the instruction memory.

A circuit that takes advantage of the chip enable (CE) may parallel more than one instruction memory, as in Fig. 6. In this case, the program code consists of one more variable in addition to L_j and d_j, the CE_i instruction memory. Following the same guidelines, the program memory may be paralleled with more than one program memory.

VI. POSSIBLE REDUCTION OF THE MEMORY SIZE

Under certain circumstances, the memory size or alternatively the number of the controlling functions may be reduced or increased, respectively, as is illustrated in the following.

Consider as an example that the microinstructions are one of the following eight-bit codes in hexadecimal notation: FC, 60, DA, F2, 66, B6, BE, EO, FE, F6, EF, 3F, 9D, 7B, 9F, and 8F. Then, by looking at the manufacturer's specifications of BCD to seven segment decoders, it will be noticed that the above eight-bit codes are generated if the four-bit binary codes 0000 to 1111 are applied to the inputs of the decoder (e.g., MC14495). Thus, instead of storing eight-bit codes, four-bit codes will be sufficient at the expense of one decoder. This technique, under the circumstances illustrated above, may decrease the width of the matrix memory or expand the number of the controlling functions.

The above example is not unique. A decoder differently designed from the one in the example may provide different codes.

VII. CONCLUSION

It has been demonstrated that designing a controller circuit does not require any design of sequential random logic circuit or microprocessor

implementation with the technique presented. The problem is simplified to converting the *a priori* known controlling functions to a set of vectors and from there on to either extracting the features or not, depending upon the application, and to keeping the logistics of occurrence of the features. From this, the instruction code and the program code are derived. Then mapping of these codes into the corresponding memories of the circuit completes the design and programming of the controller.

VIII. ACKNOWLEDGMENTS

The author expresses his appreciation to B. S. Bosik and the reviewers for their suggestions and critical reading of the manuscript.

REFERENCES

1. T. C. Bartee et al, *Theory and Design of Digital Machines*, New York: McGraw-Hill, 1962.
2. J. Hartmanis and R. E. Stearns, *Algebraic Structure Theory of Sequential Machines*, Englewood Cliffs, N.J.: Prentice-Hall, 1966.
3. M. V. Wilkes, "Microprogramming," Proc. Eastern Joint Computer Conference, December 1956, pp. 18–20.
4. J. W. Tukey, *Exploratory Data Analysis*, Reading, Mass.: Addison-Wesley, 1977.