The subject of this paper is a system of programs to aid in the logical design of automata. Figure 1 depicts the experimental system as it presently exists within IBM. There are essentially two internal formats for the system, one called the injective word as shown in Fig. 3, which in essence specifies all the logic blocks in a circuit and how they are linked together. The other is termed a cubical cover (or ON-OFF-ARRAYS) which is a means of describing the behavior of the circuit as if it were a two-level circuit consisting of ANDs followed by ORs (or vice versa). See references 1 and 2 for a more detailed description of these notations.

The programs of the system, the principal ones being shown in boxes (Fig. 1), may be thought of as transformations of these formats into themselves. There are two exceptions. They are the Sequence Chart Analyzer SCA and the program EQIWT, the equation-to-injective-word translator.

We shall first discuss the sequence chart analyzer; a sequence chart is shown in Fig. 2. This is a sequence chart for the operation of the instructions Floating Point Add, Subtract or Compare for an early version of MODEL 60 for System/360. It will be observed that across the top of the chart appear intervals labelled T1, T2, . . . , T10. These refer to time intervals and admit of many interpretations. Operations are written above horizontal line segments in one of the “time columns”. Immediately to the left are written the “immediate” conditions necessary for its execution. For example in column 5 the operation Set Condition Register is performed,
when the sequence chart is "operative" in this area, when the condition holds. The given sequence chart becomes operative at "Chart Entry Conditions" such as (on the far left of the chart) FPADD + FPSUB + COMPARE. Thence in time T1, in the T1 column some, or all, of the operations such as INVERT S SIGN are performed provided the immediate condition written to its left, such as FPSUB + COMPARE, are satisfied. If no immediate condition is there written then the operation is automatically performed. Thus several operations may be simultaneously executed. Thence one moves to the next time interval or else to the next specified interval. For example if after performing any of the operations at time T1 if SAZERO (Serial Adder is Zero) then one jumps to time interval T4. Whichever is the case one proceeds into other phases of the chart. For example after an operation in time T4 has occurred then the next operation and the next time are determined by the immediate conditions written to the left of the line segments emanating from the right half of the T5 column. For example if LEAD ZERO—OVERFLOW and —NORM and ZERO FRACTION is true then the operation SET SIGN PLUS [L] is performed. Thus any set of conditions are prescribed a "path of operations" on the sequence chart ending ultimately in an ENDOP which means end of operations.

It is thus seen that the sequence chart is kind of sequenced flow-chart of machine operations as de-
scribed for some particular data flow structure of a machine.

The Sequence Chart was originally designed by Stanley Pitkowsk and subsequently formalized by the Logic Automation Group mainly by J. M. Galey, P. N. Sholts and Jere Sanborn. Numerous abbreviations for indirect addressing are employed in the description.

The Sequence Chart Analyzer (SCA) first produces a hard copy of the sequence chart itself as illustrated by Fig. 2. The form of this chart is subject to updating and can be changed at will. The second output from SCA is a injective word which prescribes for each gate the total conditions for which it is to be open or to be closed. Figure 3 is one page of several pages of the injective word produced by SCA for the sequence chart of Figure 2. The meaning of this form may be understood by the following example: at left is a circuit and at right is the corresponding injective word.

\[ \begin{align*}
\alpha^* &= A, CE & \text{AND} \\
\alpha &= \alpha^*, T1 & \text{AND} \\
\beta^* &= B, \alpha^* & \text{AND} \\
\beta &= \beta^*, T2 & \text{AND} \\
\delta^* &= A, CE & \text{AND} \\
\delta &= \delta^*, T1 & \text{AND} \\
\eta &= \eta^*, T2 & \text{AND} \\
\epsilon &= A, T3, CE & \text{AND}
\end{align*} \]
Each line or argument is given a label; each line, for example line 3, emanating from a box is expressed as a function of the lines entering the box, together with the name of the logical function which the box performs, and the number of inputs, e.g.

\[ 3 = 4, 5 \quad \text{AND} \quad 2 \]

This method of description course allows feedback in the circuit.

The method of analysis for SCA will be illustrated in the next figure.

The injective word output of SCA is fed directly into the program CIMPL, an acronym for circuit implementation. This program accepts an injective
word described by any set of logic blocks and converts it into an SLT injective word, i.e. one composed of SLT logic modules and obeying fan-in, fan-out powering, etc. of this particular technology (SLT stands for Solid Logic Technology.)

This program was outlined in architectural form by John Earle and programmed by Peter Schneider, now of the University of Wisconsin; Michael Galey, IBM San Jose; Jere Sanborn, IBM Poughkeepsie; and others.

The first operation which CIMPL performs is to search for identical gates or inverters. It eliminates all but one of these and fans out from it. It then does the SLT implementation in the low, medium or high-speed circuit families so as to preserve the general logical structure specified by the designer on the Sequence Chart as well as the delays inherent in the structure. It does this implementation within the SLT circuit constraints manipulating the logic locally so as to satisfy the fan-in constraints and inserting powering whenever called for by the loading equations. It does the fan-in and powering manipulation so as to add the minimum amount of additional hardware to satisfy the constraints while preserving the general logical structure. The program then assigns the circuit types in the specified circuit family and produces an output which can feed directly into DRAW, the program which partitions the injective word produced into pages and assigns print positions for each page, in sum, in effect to draw the ALD sheets (ALD automated logic diagram) for Solid Logic Design Automation; it in fact generates the SLDA logic master tape.

Another input to CIMPL is Boolean equations: a preprocessor called EQIWT (equation to injective word translator) accepts Boolean equations and converts it to the injective word format. This gives the designer the ability to convert his Boolean equations directly into hardware. This path within the logical automation complex was used in Hurley, England, particularly by K. A. Duke, in the design of the Model 40 of System/360. Specifically the ALU, the Arithmetic and Logic Unit was so designed.

An alternate program called Position Draw or P-DRAW was written by Galey: this accepts an injective word in which the pages and print positions have been specified by the designer in a list structure. It generates the ALD sheets in accordance with the pattern prescribed by the engineer and also generates the Logic Master Tape of Solid Logic Design Automation. It thus saves the designer from making careful drawings for the keypunching operation and substantially cuts down on the magnitude and difficulty of the keypunch operation itself.

Figure 4 shows one page of 24 ALD sheets produced by SCA followed by CIMPL followed by DRAW. These ALD sheets would have, in the manual version, been originally computed and hence drawn by hand for insertion into the Solid Logic Design Automation System. Total IBM 7094 time for this operation was about 15 minutes.

Another entry into the system of logic automation programs besides the sequence chart and Boolean equation is through a translation code, as shown in Figure 5. This is the translation from NPL to typewriter coding. The code translation program assembles this into a set of Boolean functions, one for each output. This is then formulated as a two-level minimization problem with many outputs.

MIN is a program of the extraction algorithm\(^2, 3\) which works with cubical covers and has a domain of applicability much wider than those algorithms using canonical terms. For the NPL-typewriter code translation a minimum was obtained. This minimum two-level solution was then converted by a decomposition algorithm DECO\(^4, 5\) applied by a human computer (the program was unavailable at the time). This result was then fed into CIMPL and DRAW to produce on 13 ALD sheets an SLT design of the code translator. Total machine time did not exceed 10 minutes on the IBM 7094.

Two more programs of the system will be mentioned. These programs can be run in conjunction with the other programs of the system. The first is called \(\pi^*\)\(^2, 3\) and is a program for analyzing circuits; precisely it accepts an injective word and translates it to a cubical cover, i.e. a normal form expression, for each output of the circuit in terms of its primary inputs. This program requires that all feedback loops be cut. It enables us to apply minimization procedures to already designed circuits in an effort to achieve cost reduction.

This approach is especially useful in the case of design of low-cost circuits for which many copies will be made and wherein the savings in diodes or transistors are particularly significant. These programs have had extensive use in IBM Endicott, e.g. on the MICR (magnetic ink character recognition)
circuitry by Billy N. Carr, on the IBM 1030, a device to transmit data from terminals, and on the Serial Wire Printer. In each case, \( \pi \) was followed by MIN followed by factorization and thence (in two cases) by CIMPL. Incidentally mistakes by the engineer were detected in the design and corrected. In each case a very small amount of machine time was used.

Another program which has had extensive usage is a set of diagnostic programs, labelled DIAG in Fig. 1. This is a program which accepts a circuit in injective word format and produces a set of input patterns termed tests which when applied to the circuits is capable of determining whether or not a failure has occurred of a given variety. The variety treated by the program basically is a failure for each line wherein the line may be fixed in value, either stuck-at-1 or stuck-at-0. This program was used specifically in the design of the Memory Protect and Relocation MPR for the 7094 configuration going to M. I. T. By an addition of only 2.5 percent extra hardware it was possible by means of the tests generated for this piece of hardware to detect any single failure of the stuck-at-1 or stuck-at-0 variety. Approximately one hour was required of 7090 time to compute the tests. DIAG was a predecessor of the SLT set of programs FLT (Fault Location Technology) used on some models of System/360, written principally by Frank and Martha Evans.

A new program is being planned\(^8\) for an improved algorithm called d-DIAG for generating diagnostic tests. This algorithm is substantially faster than other known methods and is based essentially on a calculus of injective words rather than by a calculus of cubical covers, as is DIAG. Reference 8 describes this algorithm in considerable detail and makes comparisons with previous methods.

A profitable way to utilize this complex of design programs is in the redesign mode: A computer system is designed for one technology and it is desired to redesign it in another. For this purpose a program EXTRACT is needed to take from the SLDA Logic Master Tape the logical essentials to

---

Figure 4.
be fed into the analysis and synthesis programs. This process has been used partially automated, that is to say, the EXTRACT program as it now stands requires another routine before being completely automatic. This program is in an experimental state.

FUTURE SYSTEM

The Iverson notation has been used to present a very complete specification of the architecture of the IBM System/360 (Ref. 9) and this immediately opens the possibility of adjoining this description to our Logic Automation Complex. The design for the machine would thus be specified initially by means of a program or system of programs written in the Iverson notation. For each model of System/360 the data flow would also be given, essentially as a list of facilities available plus their interconnections. This part of the experimental design system remains to be done.

REFERENCES


