Networks for real-time programming

By C. S. E. Phillips*

A method of real-time programming is described which has been used for an experimental computer controlled radar. This method, which may be of general interest, uses a special kind of network analogous to a circuit diagram. The network is used to analyse and specify precisely a real-time system. The network can be drawn up without either a detailed knowledge of programming or the formulation of intricate flow diagrams. The effect of changes to the network can be readily observed and easily performed. The program is broken down into manageable sub-units which facilitate programming.

The computer program or programs required for the experimental automatic surveillance radar have already been described in general terms (Phillips, 1964) as have the main features of the practical system. Since that time we have constructed the various parts of the radar and connected them to an Elliott 920 digital computer. Apart from a change of interface and recoding no changes are anticipated when this computer is replaced by a Marconi Myriad in December 1966. We had intended to use the 920 primarily to establish the interconnections and to feel our way to an understanding of the programming requirements when the Myriad was installed. However, the interface work took less time than expected as did the short programs required to test all the parts of the radar (aerial, modulation and receiver control, plot extractor, etc.). It soon became clear that these small programs were easy to write and debug because (*a*) they were very short and (*b*) they were run separately and (*c*) they were written by people directly concerned with the equipment controlled. It was also clear that the programs would not be able to run together without considerable effort on our part. In fact it was not easy to see how the whole job could be done unless it were done by one person. Furthermore, it was much easier to describe what the system was to do than to say precisely how it was to be programmed. This was due to the complex nature of the program, the experimental nature of the work—the overall program has to be very flexible—and the interactions between parts of the program arising from real-time factors.

Clearly one programmer, even if he were very skilful and farmed out subroutines at a later date, would have taken too long to do the work. The resulting program would almost certainly be out of date before it was finished. What we needed was a way of precisely specifying the system so that the whole job could be divided into small parts, each part worked upon separately and then joined together like Meccano. Parts could then be added or replaced in the light of experience and the effect of these changes immediately visualized. We had originally intended to use subroutines for this purpose, but the complicated interactions between them due to the real-time nature of the problem would have remained. In the method to be described a network is drawn up which can be thought of as specifying these interactions; non-interacting subroutines of these subroutines are "private" and are not mentioned on the network.

* *Royal Radar Establishment, Malvern, Worcestershire.*

Network symbols

The symbols used are shown in **Fig. 1**. There are only two main "components", data and programs, and the connecting "wires" show the direction of data flow. The word data is used here in its most general sense covering numbers, code-words, tags, counts, switches, etc. It is useful at this stage to imagine a group of programmers working under a system analyst who allocates the work. The programmers need not know of each other's existence and must on no account make private arrangements with each other. The data we are concerned with in the network is publicly announced to all programmers. Any private data created by a programmer, that is data considered by the system analyst to be of purely local interest, is "lost" inside the program and is therefore of no concern to the network. However, the system analyst should always try to turn private data into public data since this increases the flexibility of the system.

The simplest data is in the form of a fixed box where words are kept in successive locations. The diagram shows words entering and leaving a box. The programmers know the layout of the words in the box. They do not need to know the absolute addresses since these are arranged by the system analyst. The data boxes are thus addressed by name and the individual words in the box are addressed relative to the start of the box.

We have used two extensions of the data box, viz. circular and chain lists. The circular list contains many sub-blocks of data in successive locations, stored by program in the order in which they are generated. The list is of fixed length so that when it is full the input data is automatically re-directed into the first locations. The newest data thus circulates round the list, overwriting the oldest data. The diagram shows two small data boxes, whose positions are publicly known, con-

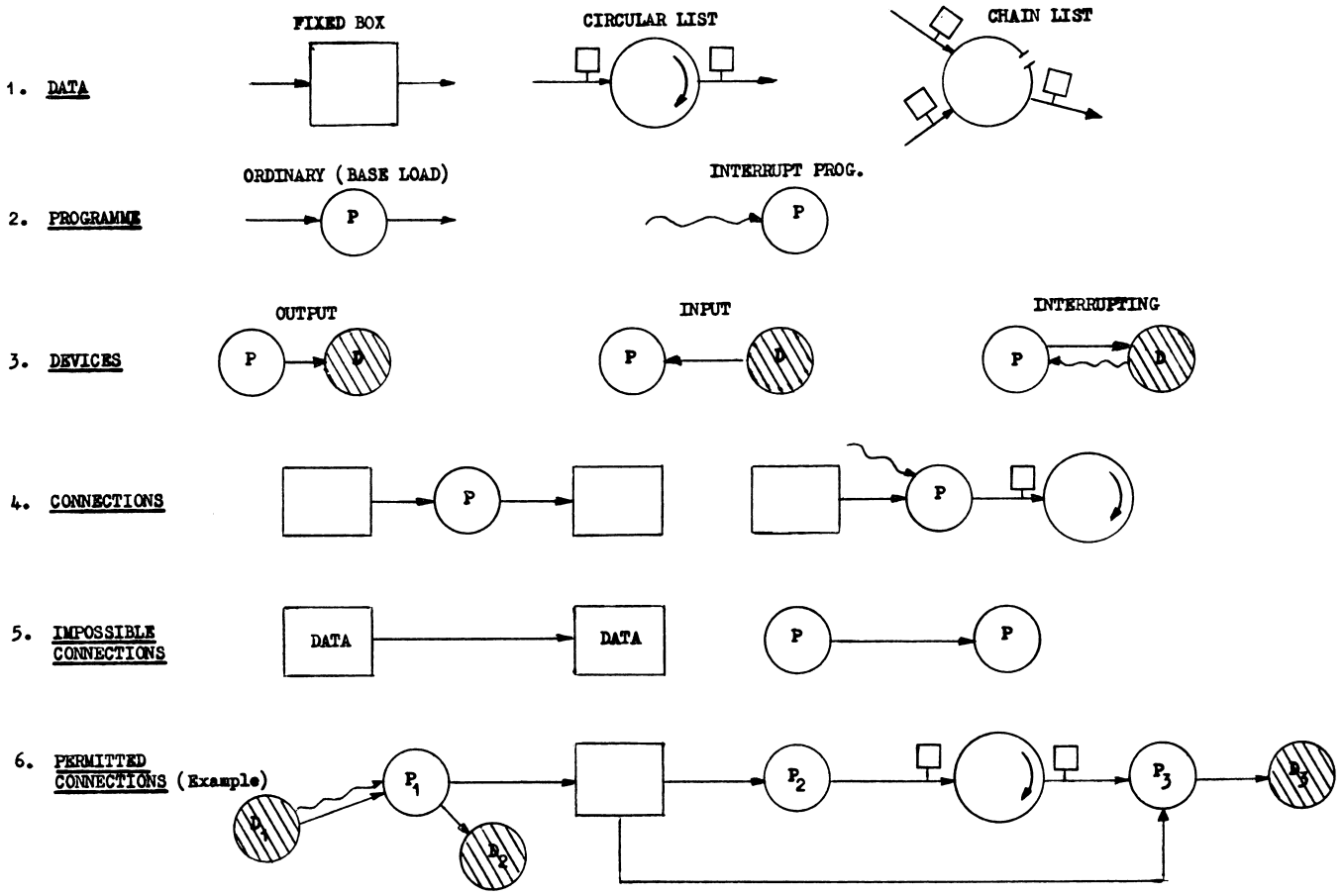

Fig. 1.—System symbols for real-time programming

taining input and output pointers, i.e. the current addresses for input or output of the next data sub-block. The output pointer box could in practice be grouped with all other output pointer boxes. This larger box would then be the public list of output pointers. The input box could contain other data required to read from or write into the list. This information might consist of the list length and the size of each sub-block of data in the list. It is therefore possible to change the sub-block size and list length at a later stage without necessitating program changes. Knowledge of the input pointer also enables other programs to read independently from the list without running ahead of the input.

The chain lists are similar except that the successive sub-blocks of data contain pointers to the succeeding and preceding sub-blocks. This type of list is used to facilitate the storing of data sub-blocks so that they may be addressed in an order dependent on some quantity (e.g. azimuth angle) in the sub-block rather than the time of generation of the data itself. Since, in a chain list, data sub-blocks can be interposed, it is possible to have more than one input to these lists. This facility is shown in the diagram.

Programs

Programs are drawn as a circle with a P inside, the different programs being distinguished by a suffix. Interrupt programs are distinguished by a wiggly or fertility line from a device whereas ordinary base load (non-interrupt) programs are assumed to be continuously operating under control of a master program. The diagrams show (1) an ordinary program taking in data from a single data source, operating upon it in some way and transferring the result elsewhere, and (2) an interrupt program being fertilized, but performing no useful work.

Devices

The computer communicates with the outside world, taking data in from or putting data out to devices of one kind or other (punches, readers, tape-decks, A/D and D/A equipment, etc.). These devices are represented by a cross-hatched circle with a suitable letter inside. The diagrams show (1) a program generating data, and transferring it to a device, (2) a program taking data autonomously from a device, and (3) a device

which triggers an interrupt program which subsequently transfers data to the same device.

Connections

Thin lines represent the flow of data, the direction of flow being highly significant. The diagrams show (1) a program taking data from the left-hand box, operating upon the data in some way and transferring results to the the right-hand box, (2) an interrupt program transferring data from a box to a circular list.

It should be self-evident at this point that direct connections between data boxes and between programs, as shown in the next diagrams, are quite impossible and must never be used. The final diagram (6) of Fig. 1 shows a simple example of a possible program "circuit". Device D1 triggers off program P1 which takes in data from D1, operates upon it in such a way as to transfer one set of data to the box and another set of data to device D2. (The fact that P1 is not further sub-divided means that some of these operations are common and that both operations are required to be performed consecutively.) Program P2 reads data from the box, though not necessarily all of it, operates and transfers data to the circular list. Program P3 also takes data from the box, combines it with data from the circular list and transfers the result to device D3. The arrangement of data in the box and the circular list is known to the system analyst and to the P2 and P3 programmers, but if a separate programmer writes P1, the P1 programmer needs to know only the arrangement of data in the box.

Multi-level interrupts

A great convenience for real-time work is the external interrupt. This is a wire or wires which, when energized by an external device, causes the computer to link out to special interrupt programs. The program jumped to is usually reached in this way only and the program jumped from is either another interrupt program or a base level program. The base level program is presumed to be always operating when no interrupt calls are present, and is reverted to after all the interrupts have been dealt with.

The simplest form of multi-level interrupt is a single wire which calls for a general multi-level interrupt program whose first job is to decide which device is calling. A word is read which names the device and thus determines which branch of the interrupt program is to be used according to a predetermined priority. This is a case of multiple priority interrupting on one level.

If the interrupt rate is high the time spent sorting out the calls can become significant. For this reason the Elliott 920 and Marconi Myriad (among others) provide a more sophisticated multi-level system where several wires are available. In this case each wire directly initiates a different interrupt program (each one of which could of course control multiplexed priorities as in the single-level case). Clearly it could arise that more than one device calls simultaneously, or that a high priority device interrupts one of lower priority. Various hardware and program methods which vary from one machine to another are available to deal with these circumstances. The exact details do not concern us here so long as it is realized that in a multi-level system each level constitutes an independent program (i.e. a separate program sequencer is used) and the computer is time shared between each level and a non-interrupt or base level according to some system of priorities.

This does not mean that programs on one level cannot communicate with programs on another. In the example given in Fig. 1, the interrupt program P1 fills the data box. However, programs P2 and P3, which use this data, can be written on any other level. The advantage of the interrupt in a real-time system is that the times at which programs are run can be arranged to be dependent on the devices themselves, and the advantage of the fast multi-level interrupt system is that special-purpose buffers and computing equipment need not be built into the external devices. Of course, some devices can be considered as mere slaves of the computer and can be driven (via direct data transfer) without recourse to interrupt programs.

The master programs

It should be evident from the foregoing that in a more complex network, such as these given in **Figs. 2** and **3**, there will be a number of programs grouped in levels. Although it is a simple matter to redistribute the programs (possibly following system changes), each group of programs, once constituted, is independent of all other groups in the way described. Each interrupting level will cause its own group of programs to run, and the allocation of programs to groups must depend on this fact. However, we must now consider the frequency of running programs within the groups—particularly the base level group. The simplest method is to run them one after another in a fixed order. This hardly requires an executive or master program at all. Usually a more complicated arrangement is required which varies the order according to the quantity of data in a list, to the time elapsed since the program was last performed or according to an external operator controlled switch. Following this procedure "important" programs are not kept unnecessarily up to date and "unimportant" programs such as self-test and calibrations are not omitted altogether. In general, therefore, one may require a separate master program for each level. No master program will be complicated, since it gets all its information from the network, and on some levels master programs can be dispensed with altogether.

FMCW automatic surveillance system

The networks as envisaged at present are shown in Figs. 2 and 3. The only change proposed for the Myriad is to make use of the larger number of interrupt

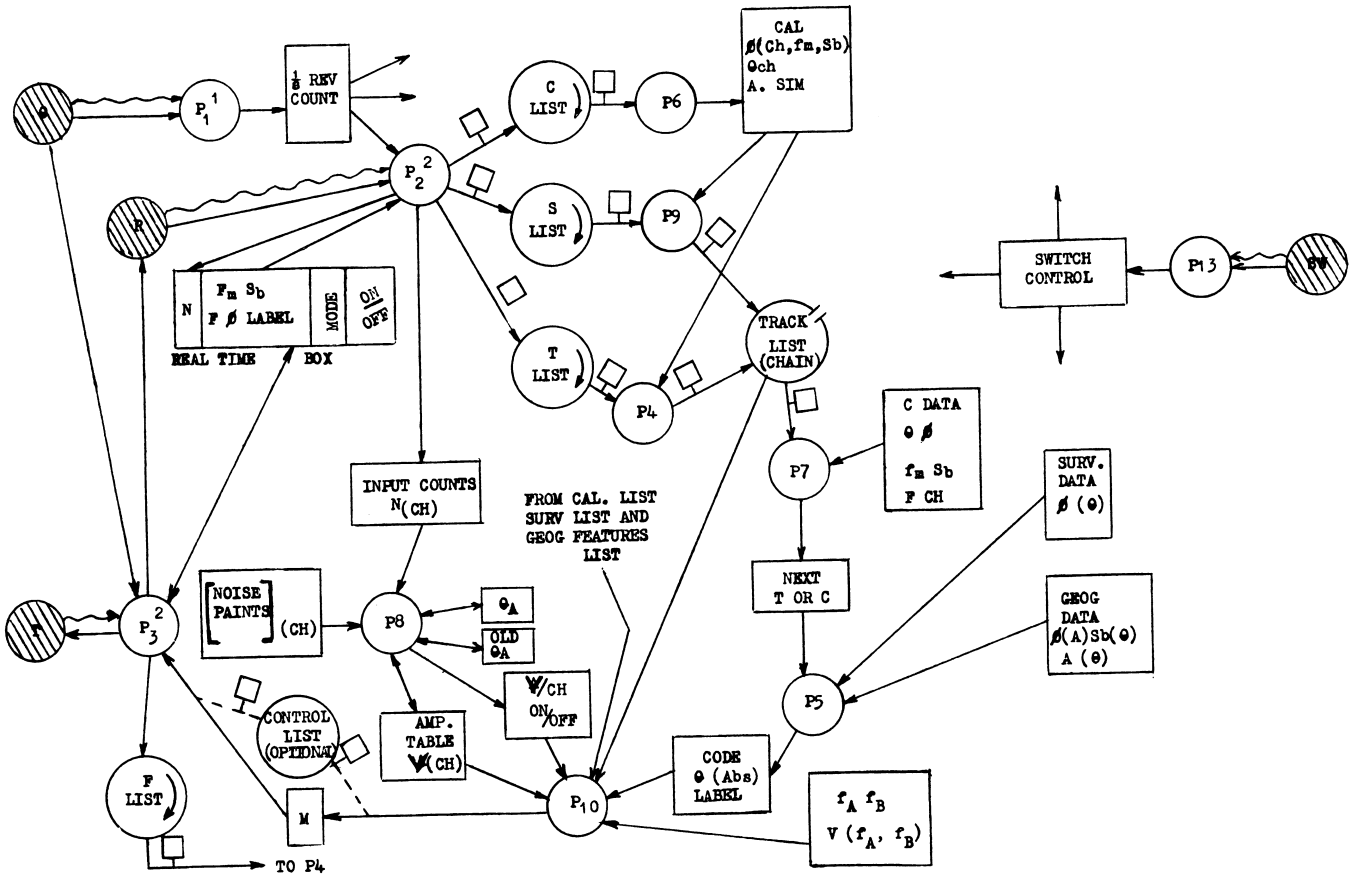

Fig. 2.—FMCW surveillance system analysis

levels. The multi-level programs of Fig. 2 are as follows:

Level 1. P_1^1 triggered by azimuth pulses from rotating aerial.

Level 2. P^2 triggered by radar plot extractor.
P_3^2 triggered by Timer.

(owing to shortage of levels P_2^2 and P_3^2 have to be run on the same level).

Level 3. P_{13}^3 triggered by switches (and also on-line teleprinter, digital PPI and digital strobe not shown on the diagram).

Level 4. P4–P10 operated by Master program.

The main omissions from the network of Fig. 2 are the outputs which vary with the particular experiment and consist of printing the contents of one of the data blocks or lists. Fig. 3 shows the output arrangements as part of a more general purpose control system described in a following section.

Data lists (Fig. 2)

C List Fed by P2. Contains calibration plots.

S List Fed by P2. Contains plots obtained when the radar is operating under surveillance mode.

T List Fed by P2. Contains plots obtained when the radar is looking for plots of tracks.

F List Fed by P3. Contains plots that were expected, but the radar failed to detect.

Control List Fed by P10. Contains all sets of data necessary for the control of the radar in the future (depending on list length).

Track List Fed by P9 and P4. Contains all information on tracks and initial detections. This list may be considered as the "output" of the radar system.

Data Boxes (Reading from left to right and starting at the top left-hand corner.)

Rev count Fed by P1. Contains angular count measured in eighths of a revolution up to a maximum of 2^{15} revolutions (Elliott 920) or 2^{21} revolutions (Marconi Myriad). This data is used by many programs to obtain "absolute" azimuth.

Cal Fed by P6. Contains the most recent (smoothed) measurement of the calibration in range, and azimuth of each radar channel as a function of modulation characteristics. Also contains the amplitude of the calibration signals for maintenance purposes.

Real-Time Box Fed by P2 and P3. Contains the number (N) of plots since the last control change, all information about the waveform transmitted, the mode (surveillance, tracking or calibration) and an ON/OFF switch (this switches off P2 until transients have died away).

Switch Control Fed by P13. Contains number in switches when button last pressed by the radar operator.

Input Counts Fed by P2. Contains numerical total of inputs on each channel during S mode since P8 last used.

C Data Fed by initial input. Contains angular positions of calibration simulators. Also list of all modulation frequencies, sideband numbers, doppler frequencies and channels over which calibrations are required.

Surv. Data Fed by initial input. Contains list of elevation beams and order in which they are to be used during surveillance periods.

Noise Paints (Ch.) Fed by initial input. Contains number of noise paints permitted on each channel during a given period of time.

θ_A *and Old* θ_A Fed by P8. Contain absolute azimuth and beginning and end of given period (see Noise Paints above).

Next T or C Fed by P7. Contains address of track (in Track list) or calibration plot which is next in azimuth order to be placed in Control list.

Geog. Data Fed by initial input. Contains elevation contour of surrounding territory, special side-band requirements and false alarm variations as a function of azimuth.

Amp. Table Fed by P8. Contains list of voltage levels at present used on each channel (i.e. when last set in) determining detectability.

V/Ch ON/OFF Fed by P8. Contains channel number of any alteration required. If none required switch to OFF.

Code Label Fed by P5. Contains information on next control required before adding any voltage level changes.

f_A, f_B $V(f_A, f_B)$ Fed by initial input. Contains calibration of main modulating oscillators. Calibration monitoring may be added later.

Devices

T Timer, P3 arranges to call itself in a given period of time (measured in azimuth pulses).

R Radar.

θ Azimuth pulses from rotating antenna.

Brief description of process

Starting with the radar control program P3, this changes the radar so that it adopts the new mode required, all details being noted in the Real Time Box. This program would operate at variable speeds up to 100 times per second. Any plots obtained are taken in by P2 and after consultation with the Real Time Box, counted in "input counts" and passed to either C, S or T list. Programs P6, P9 and P4 read these lists. Program P6 reads the C list and updates calibrations. Program P9 reads the S list, eliminates any non-random interference, recalibrates the remainder and puts them on the track list. Program P4 compares the track inputs with the track, finds the best correlation, calibrates, updates and smooths the track, predicts the position when the radar could see it again and works out the most suitable transmission. Depending on the use to which the radar is put, outputs would normally be selected from the track list.

Proceeding in a clockwise direction, program P7 takes either the "next" track from the track list or the next calibration plot required. Having decided which comes first, the track having precedence in the event of conflict, program P5 mixes in the surveillance and geographical data and finally decides on the next control operation. Program P10 is used to add in any amplitude selection level changes, and the control data is put on the control list. Some time later P3 picks up this data, marks the track in the track list and the cycle is complete. If no returns are obtained the expected plot is placed on the Failed list (F) and the relevant track is informed of the failure. All lists are arranged to be in azimuth order (including the track list, where the azimuth order is based on predicted and not present positions). This together with the use of labels ensures negligible time wasted searching through lists.

Program P2 in conjunction with P8 arranges that the surveillance false alarm rate is kept constant. At infrequent periods P8 compares the surveillance input counts on each channel with the desired rate given in "Noise Paints" and adjusts the voltage level accordingly. This is a form of long term a.g.c. Short time level changes are dealt with in P5 using "Geog. data". This is an example of a type of program which, it is believed, will have a profound effect on the design of the radar equipment itself.

Switch control network

The purpose of "switch control" is to allow manual control of the overall system. As envisaged at present, these manual controls are for two purposes. One is to permit on-line display or printing of any quantity in any list or box. The other will permit manual introduction of new programs and alteration of the "fixed" lists of basic data. Although these manual controls are essential for the experimental development of the automatic radar as a whole, some would be required in a developed system, since conditions will vary from one application to another.

The detailed system, devised by Dr. J. R. Prior, is shown in Fig. 3. A desired manual operation is set up

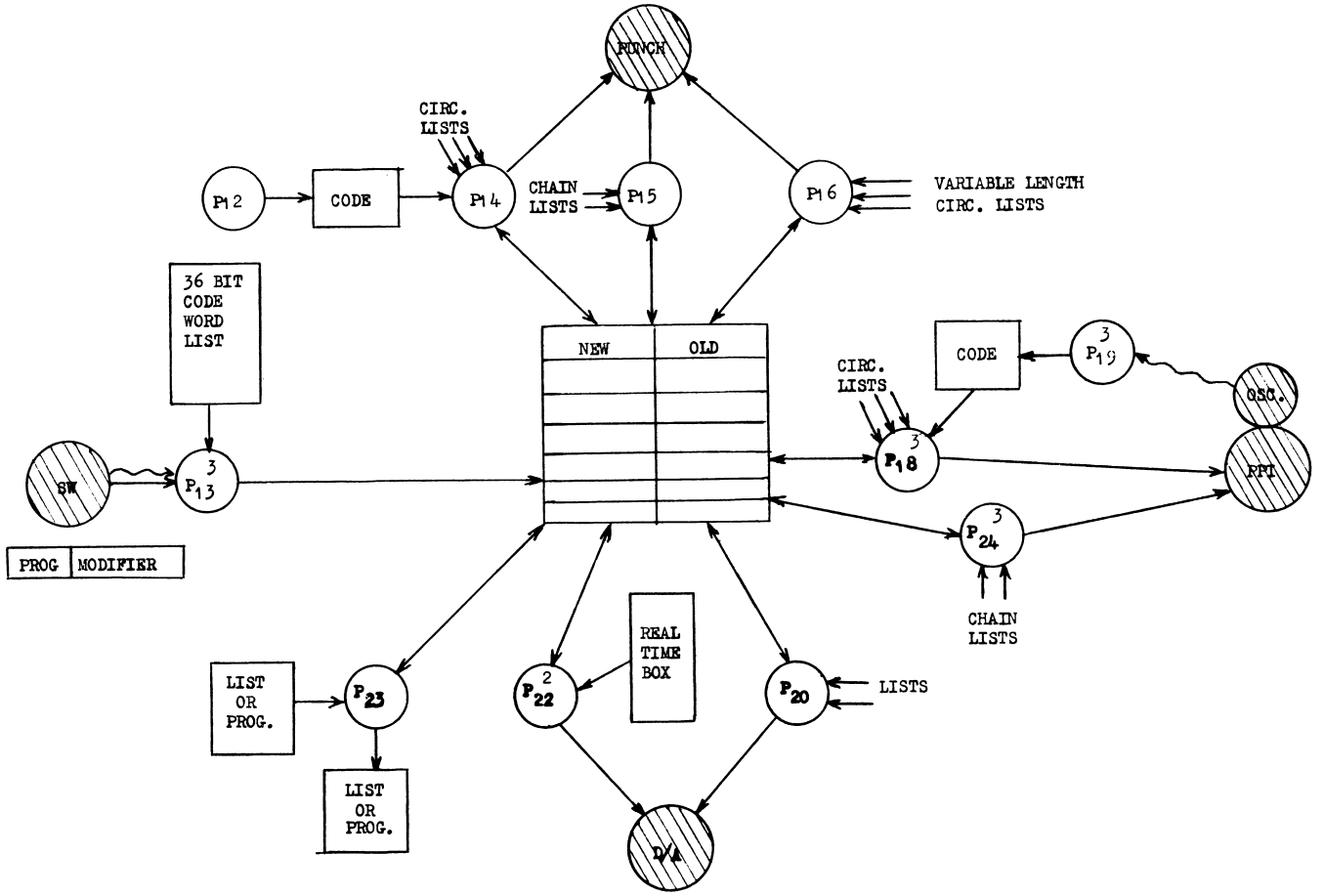

Fig. 3.—Switch control network

on the switches in coded form. This code is divided into two parts, one part, "PROG", defining the additional program requested and the other part, "MODIFIER" specifying the information required by that program. For example, PROG could refer to a general program for printing the surveillance list, and the bit pattern of MODIFIER could determine a certain part of that list. A zero MODIFIER signifies that the program must be switched off.

When the operator presses the switches button, interrupt program P13 enters a number into the NEW/OLD box at a location determined by PROG. Each PROG location is divided into a NEW and an OLD part and P13 always feeds the NEW part. The number fed in is in some cases the MODIFIER itself, but certain MODIFIERS are detected by P13 as "indirect". In these cases the MODIFIER is used to access the number from the "36 BIT CODE WORD LIST".

Programs P14, P15, P16 for the punch, P18 and P24 for the P.P.I., P20 and P22 for the digital to analogue converter and P23 for list or program changes operate independently of the switches interrupt program P13. Each individual program first checks whether there are new operator demands, by comparing the contents of NEW and OLD locations. If they are different the OLD contents are replaced by the NEW contents and the program is changed.

The boxes labelled "code", which are used by P14 and P18, enable these programs to operate on different lists when these are members of a closely similar group. This code is fed by a master program P12 or P19 which is unaffected by the switches. The digital P.P.I. master program P19 controls the sequence P18 and P24, all three programs being on level 3 interrupt. The interrupt arises from the device labelled "osc" which controls the frequency of operation of this particular group of programs. The frequency of this oscillator can be manually set at a rate which provides a sufficiently good picture without stealing too much computer time.

Of particular interest is the display of the "Real-Time Box" (see Fig. 2) via program P22. This program is entered on level 2 after P3, thus ensuring that the actual "instantaneous" changes in radar control are correctly displayed in real time via a D/A converter and oscilloscope.

Conclusion

At the moment of writing (December 1966) all the programs of Figs. 2 and 3 have been written and run together satisfactorily. A simplified P4 is being used to close the main loop of Fig. 2 and the radar control

program P23 of Fig. 3 has been greatly expanded. The network diagram has proved admirably suitable as a means of bridging the gap between intentions and specification. Thus its first advantage lies in its inherently simple yet precise manner of specifying a real-time computer problem. In the very short time it has been in use, the rate of completion of programs has been very high so that its second advantage lies in the breaking down of a complex system into smaller units, thereby enabling a group of programmers to work in parallel on relatively simple programs. Its third advantage is its flexibility as shown by the ease with which additions and modifications have been made. It is thus particularly suited to experimental programming and to those systems which may require development. The fourth and overriding advantage is its ability to facilitate the running together of separate programs. Here we have found that a partially completed network can be used as a vehicle for testing a new program.

Although the main spur for the work was the need to find a manageable method for the real-time on-line programming of an automatic surveillance radar, it is clear that the method could be applied in many other fields. Although the present work is being carried out in machine code a further improvement could be obtained if the detailed programming were to make use of a high level language, such as JOVIAL or CORAL. The language must, of course, be suitable for on-line work, i.e. it must be able to handle interrupts and preferably have other special features as well, but these requirements do not in any way arise from the use of a network. One can imagine in the future the combination of a network and a high-level language so that quite complex control systems could be planned and developed with comparative ease. As far as future ground surveillance radars are concerned, this may lead to a general-purpose method of data handling and control, for radars with different, and variable, physical characteristics.

Reference

PHILLIPS, C. S. E. (1964). "An Automatic F.M.C.W. Surveillance Radar", *Proc.* Eighth AGARD Avionics Panel Symposium on Radar Techniques for Detection, Tracking and Navigation (September 1964). Also R.R.E. Memorandum No. 2324.

Book Review

Numerical Processes in Differential Equations, by Ivo Babuska, Milan Práger and Emil Vitásek, 1966; 351 pages. (London and New York: *John Wiley and Sons Ltd.*, 63s.)

This is a translation of a book published in Prague in 1964. A short introductory chapter is followed by one on the stability of numerical processes in general. This starts with examples of simple calculations performed on different computers; the diversity of the results for the same calculations is illuminating. The authors then introduce their concepts of local and global stability, which are illustrated by examples drawn from recurrence relations and matrix processes.

Chapter III, on initial-value problems for ordinary differential equations, concentrates chiefly on classical recurrence methods and Runge-Kutta methods. The problem of deriving satisfactory error estimates is also discussed. There follows a chapter of 150 pages on boundary-value problems for ordinary linear equations; an interesting range of topics is treated, including factorization and variational methods.

The fifth chapter, on elliptic partial differential equations, is devoted to the derivation of the relevant finite-difference equations, and the solution of the resulting linear systems by direct or iterative methods. The book concludes with a short chapter on parabolic equations.

There are two disturbing features which recur throughout the many numerical results displayed. First, the graphs of errors, in various processes, due to round-off, against the reciprocal of the step length, indicate that progressive reduction of the interval eventually gives rise to larger errors. The implications of this phenomenon are disturbing, since it appears to rule out the possibility of ever ensuring that the numerical results are independent of the interval used. In fact, however, it is possible to prepare programs based on Runge-Kutta formulae, for example, which do not possess this defect. Secondly, the numerical results presented seldom agree with the exact results to more than a few of the figures quoted, and the uninitiated reader may falsely conclude that higher accuracy is unattainable for problems of the type considered. Again one may cite the example of Runge-Kutta methods which may be programmed to yield an accuracy comparable with the working accuracy.

The reviewer is also not entirely happy about the definitions of global and local stability of numerical processes. Each of the recurrence relations used for illustration (on pages 28 and 29) arose "from the solution of a simple system of differential equations". Failing an exact solution of the recurrence relation, values of the solution to the differential equation would have made the results more revealing.

The book is generally well produced, although the reviewer found a few misprints. Two coefficients are wrong in the formula of Huta on page 88. Incidentally, this is a thoroughly bad formula in practice in view of the size of some of the coefficients, and has been completely superseded by the formulae of J. C. Butcher (*Journal of the Australian Mathematical Society*, 1964). The last term in the second equation of (2.4.1) has an index missing. There is an error in the scale for the abscissa of the graph of Figure 3.4, and the coefficient of y_k in Example 4.14 should read $-2{\cdot}0225$. None the less this book can be commended to research workers in the field as it contains many stimulating ideas and treats some topics not easily found in extant literature.

A. R. CURTIS