Debugging and assessment of control programs for an automatic radar

By K. Jackson and J. R. Prior*

The development of a program for an experimental automatic surveillance and tracking radar is described. The problems of debugging a real time system are discussed. The technique of testing the real-time system in four stages is of interest to users in other fields including business data processing, where on-line systems may be under development.

(Received January 1969)

During the last three years the authors have been concerned with program development for an experimental automatic surveillance and tracking radar. This radar has been described in detail elsewhere (Phillips, 1966, 1967a), but in order to put the rest of this paper in perspective a brief description of the system is given below.

In our experience there are two main problems associated with the programming of a complex experimental system. The first of these is the difficulty of subdividing a large program, the details of which are not at first known precisely, so that more than one programmer can work on the system. This subject has already been discussed by Phillips (1967b) and the methods he suggests were used in all our work. The second problem arises once the system has been partially programmed and concerns the methods whereby subsequent development of the real-time system can be speeded. The difficulties here are the recognition and elimination of faults occurring in the system whilst it is operating in real-time, and the assessment of the performance of the system.

We are now using a method of program testing which greatly reduces these problems and which has general applications to on-line systems.

The automatic radar

The aim of this equipment is to use a fast, multilevel interrupt computer to control various parameters of the radar so that it can survey through 360° in azimuth and up to 60° in elevation. Tracks are automatically initiated on targets within about 50 km of the radar and fresh information about these tracks is obtained at regular intervals by controlling the aerial beam pointing direction and the transmission.

We are using an aerial which rotates at several manually variable rates between six and forty revs per minute. The aerial provides a beam three degrees high in elevation which can be positioned rapidly under computer control to any one of twenty-three elevation positions between zero and sixty degrees.

As this aerial rotates the computer must decide at each point the elevation setting, and the transmitter and receiver characteristics necessary for correct performance both of surveillance for new targets and of redetection of known tracks.

In addition to the peripherals concerned directly with the radar there are two which are used to control the timing of computer operations. One of these is the master azimuth clock. This is a counter which is driven by the rotation of the aerial. It can be read at any time by program to give azimuth as a fraction of 360°. Since we have found it less troublesome to arrange the timing of operations with reference to an unambiguous clock rather than one that repeats every 360°, the counter interrupts the computer regularly so that a revolution count can be kept within the computer. This count can then be combined with the counter reading to give an unambiguous measure of azimuth.

The other timing peripheral is an angular interval counter. This device can be set to any angle up to 360°. It will then provide an interrupt when the aerial has rotated through that angle.

The control programs

One object of the programs is to provide data to control the radar aerial, transmitter and receiver so that the system will perform the tasks of surveillance and tracking of many targets. The control data is stored in a time ordered control list since the aerial is rotating mechanically at a constant rate. This list is a buffer holding the control data for a period of up to 135° ahead of the radar aerial azimuth at any time. Control data is output from the control list by an interrupt program initiated by the external angular interval counter. The interrupt program also sets the counter to the angular interval before the next control so that when that angle has elapsed another control will be output with negligible time error.

Detections are made by recognising the presence, in any of the receiver channels, of a peak of energy which exceeds a predetermined level for the channel. These channel levels are controlled by the computer. When a detection occurs the computer program is interrupted and the target data is stored in an input buffer within the computer. Subsequently the data is used by the non-interrupt (or base level) program to generate a track list which contains the current positions, velocities, priorities, etc., of all detectable tracks. The data from this track list, together with that from various preset lists of data within the computer is then used to generate the control list.

* *Royal Radar Establishment, Malvern, Worcestershire*

The master azimuth clock interrupt has been allotted the highest priority level since it is important in deciding the timing of other programs that there should be no confusion as to the azimuth at any time. The next two lower levels hold the control output and data input programs respectively. The remaining lower levels are used for displays and manual controls.

Since the greater part of the work is done on the base level the program on this level has been subdivided into roughly 20 sub-programs which interact with each other and with programs on interrupt levels only through data stores. These base level sub-programs all operate under the control of a master program which has two sections.

The first section contains the setting up procedure. This is entered once only each time the program is run. Its purpose is to initialise all modifiers and data which might lead to malfunctioning if left undefined.

The second section controls the frequency of operation and priorities of the base level programs. It arranges that these programs provide and process data at a suitable rate to match the demands of the external equipment. Whenever one of the sub-programs operates there is an optional print-out of the relevant information about the program and its data interfaces. After the operation of any of the programs on the base level the priority list is again studied to see if a more important sub-program is demanding attention. If not then lower priority programs are allowed to operate.

Debugging

One of the greatest difficulties in the development of a real time system of this kind is debugging the on-line programs. This difficulty is more pronounced in a real-time system than an off-line system for the following reasons. In contrast to most off-line programs the data being processed is often of a random nature; hence fault conditions cannot be repeated in an identical manner. Owing to the logically complex nature of the operations, faults are frequently only recognised some time after they have occurred, by which time any evidence of use for fault diagnosis may have been destroyed. It is often impracticable to insert extra print into the program for fault diagnosis, or even to use the facilities built into the master program, since the printing rate may not be fast enough to keep up with all the data output required and, in any event, vast amounts of data may be printed before a very infrequent fault occurs. Additional difficulties arise in an experimental system in which new equipment is also being developed since this equipment can frequently provide incorrect data.

As a result of the above difficulties the task of on-line debugging becomes one that can occupy both the computer programmer and all the peripheral radar equipment for long periods. It is clearly preferable therefore to do as much debugging as possible off-line and in the absence of equipment. We have developed a four stage process of debugging, the aim in the first stage being to test small parts of the system in isolation by simulating the input data to those parts. The output data can then be studied at leisure to ensure that the correct actions have taken place. In this way debugging becomes an off-line process. As more of the programs are developed they can be combined to make groups of programs to be tested together off-line. This is the second stage of development. The third stage, which must occur before finally running the programs on-line, is the operation of the whole system in pseudo-real-time. This is in effect an extension of the above methods to the point where all the programs on all the different interrupt levels are combined together.

Pseudo-real-time operation

The aim of this part of the debugging is to remove many of the faults which occur due to the variable timing of the operation of one program with respect to another. It also enables the programmer to operate the complete system in the absence of peripheral radar equipment, thereby both removing the possibility of equipment faults as a source of error and simplifying the final and most difficult stage—that of true on-line debugging.

In order to operate in pseudo-real-time without equipment one must simulate (1) the passage of time, (2) the timing and action of interrupts and (3) the sources and sinks of data.

It is important that time and interrupts are simulated in such a way that situations are precisely repeatable. For this reason they are simulated by software. Hardware simulation can only lead to repeatable sequences if the timing is driven from the computer logic clock so that interrupts occur at the same instruction on repeated runs of the program.

We have simulated the passage of time (which in our case is proportional to aerial azimuth angle) by adding an additional sub-program to those operating on the base-level. This program is entered from the master program once for each cycle of the rest of the programs on the base level. The additional sub-program contains both a master clock store to simulate absolute time and an interval timer store for each of the interrupt levels. The master clock is incremented by unity each time the program is entered. This store is read by any program which in real-time operation would read the peripheral (azimuth) clock to determine the present time.

All the interval timer stores are decremented by unity each time the program is entered. Whenever an interval timer store becomes zero the interval has elapsed and the corresponding interrupt program is entered on base level. The particular counter is reset to the next interval appropriate to the level of interrupt. In this way the event by event behaviour of the real system is simulated, though not necessarily at the 'true' rate. The system can therefore be described as operating in pseudo-real-time.

The incremental times to which the counters are reset depend upon the nature of the interrupt. Generally, sources of interrupt can be divided into four classes:

1. Those which provide interrupts at regular intervals not under computer control (e.g. the interrupt from our master azimuth clock which occurs every 180° of aerial rotation). Obviously these are simulated by replacing a fixed number into the relevant elapsed angle counter store on each occasion.

2. Those which provide interrupts at times specified by the computer (e.g. interrupts calling for more controls for the radar). These must be simulated

by extracting the relevant time interval from within the computer.

3. Those occurring at random time intervals (e.g. noise inputs to the computer from the radar) can be simulated by selection from a table of random numbers.
4. Those whose time interval is defined by laws not known precisely to the computer (e.g. inputs or target data). This last group is the most difficult to simulate since the timing here requires a knowledge of the environment external to the computer. In the past we have used a list of times calculated by hand to simulate the timing of track detection interrupts but, as explained later, we currently use a programmed model of this environment to calculate these times by computer whilst the control programs are running.

Let us now consider the problem of simulating the sources and sinks of data. There is no difficulty with regard to outputs since no 'real' equipment is being controlled. We have found it useful, however, to replace each output instruction of the program by the option to punch the data if required. This can be used to give a diagnostic facility during debugging.

Input of data is more complicated to arrange. At present there are only two sources of data for input. The first of these is the absolute time (or azimuth clock) The second is the data about the external environment and in the past this was read from a precalculated list of tracks. Currently, this data is provided by the environmental model and is influenced by the data which is output.

A limitation of the third stage of debugging in which all the programs operate together with pseudo interrupts occurring in the correct time relationship to each other is that these interrupts always occur after the operation of any one of the base-level sub-programs, rather than during the operation of these sub-programs. However, even with this limitation, the real-time interaction of programs can be tested comprehensively. We have set up various lists of input data and interrupt times for targets which have tested many functions of the logic. Removal of any faults now becomes a very much easier process since time can be made to 'stand still' when necessary. The computer may be stopped at any point and the contents of stores examined or printed. Data may be printed if desired whilst the system is working in pseudo-real-time with no danger of upsetting the operation. Also by running on the same input data the identical situation may be repeated with different tests until the fault can be located. The ability to run the system without associated equipment removes all equipment faults from consideration.

In some applications it might be possible to pass through a further intermediate stage in which some parts of the peripheral equipment were associated with the system working off-line.

Program layout

Full advantage of pseudo-real-time operation can be obtained if the organisation of the program tapes is made in such a way that the transition from on-line to off-line working involves the minimum of change. All transfers to and from radar equipment have therefore been arranged to take place via short sub-routines (or macros). These routines each read a switch store, which can be simply set at the start of any program run, to define on-line or off-line operation of that routine. **Fig. 1** shows an example of the programs associated with on-line timing and off-line simulation of time. This shows how the input and output routines are arranged

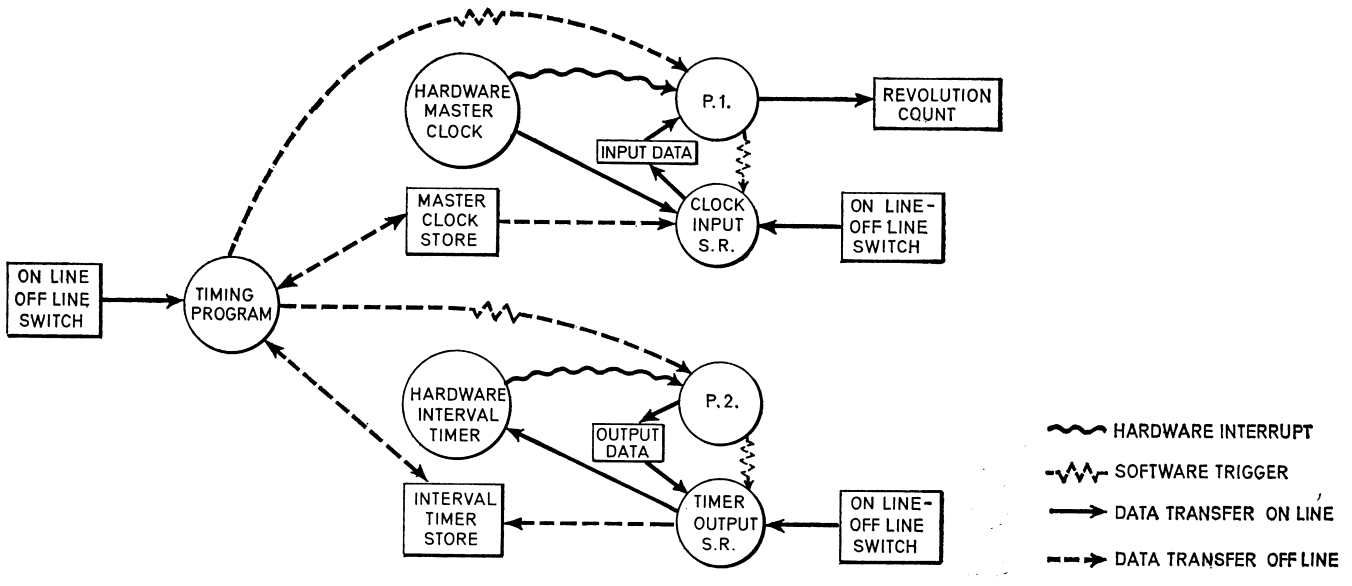

Fig. 1. Simulation of time and interrupts

to access either the hardware devices or the software stores. It also indicates how the programs P_1 & P_2 can be entered on interrupt from the hardware devices or can be triggered from the timing program.

The additional sub-program entered from the master program to simulate time and interrupts is also activated by a switch, hence while working in real-time the presence of the programs to simulate time and peripherals causes negligible increase in running time.

System assessment

Once the logical errors have been removed from an on-line program one is still left with the problem of performance assessment. This is particularly true of an experimental radar. It is frequently expensive and impractical to test the performance of the system using genuine aircraft, hence simulation of the data sources of the external environment is to be desired.

As mentioned earlier we have written programs for the simulation of the track environment. These programs operate by calculating the co-ordinates of many manoeuvring tracks as a function of time once they have been given some initial information about the targets. At the times that the aerial is pointing on the correct azimuth direction to detect the targets the conditions of the radar are examined to determine whether the target is detectable and also what the input information should be. In this way tracking of many manoeuvring targets can be simulated. Input of this artificial track data is synchronised to the radar azimuth by using either the external hardware interval timer for real-time operation or the internal software interval timer for pseudo-real-time operation.

Thus the input of track data can be regulated. The track simulation programs can be run with the systems working in pseudo-real-time or in real-time. Also when the system is operating in real-time they can work while true radar detections are being made. In this way it is possible to have a gradual build up of complexity in the testing and assessment of performance since early work can then take place in the absence of equipment. This is then followed by tests with equipment and pure track data provided by the simulator. Finally we can simulate a more realistic radar environment by adding clutter returns and noise from the radar.

The program allows the testing of the complete computer system in real-time with track situations which are repeatable. The simulated information can be outputted to displays in the normal way whether the program is running in real or pseudo-real-time. If any faults are noticed one can revert to the more simple preceding steps using the same data to eliminate the fault.

Future plans

In the near future our computing facilities are to be increased by the addition of a disc backing store, an on-line flexowriter and an alpha-numeric display. Using this equipment our methods of program development will be extended and improved. We will be able to store the past history of the real-time situation for several minutes of operation so that diagnosis of faults and assessment of programs can be performed by analysing the stored data. This analysis will be done using the display and associated keyboard to show the contents of various lists on demand.

Future programming will be done using a high level language (CORAL 66) so that the writing time for new programs will be reduced. Since the use of such a language eliminates many of the simple logical errors made in machine code the emphasis of the debugging will shift towards faults which prevent the program compiling correctly. This work, which at present involves much paper tape handling, will be speeded considerably since the editing and compilations will be done using the programs from the disc. The display will be used to show fault reports and the text requiring correction, and the keyboard will allow input of commands for rapid correction of texts and to initiate recompilations.

Conclusions

The method described here for the debugging and assessment of an automatic radar has general applications in the field of process control, since whatever the application it is possible to program a model which can be used to provide the data for input to the control programs, and the times at which interrupts would occur. The ability to remove faults due to timing interaction of the program whilst working off line allows more rapid debugging. Also the fact that the program can be operated in the absence of peripheral equipment eliminates the need for the equipment and the possibility of equipment errors. The programming effort required to implement a system of simulated time and interrupts is very small. In our system the timing program consists of less than one hundred instructions. Simulation of the environment and data input sources was initially done by using tables of data and a short input program. The more flexible simulation now implemented is roughly 1000 instructions long.

The effort involved in developing these programs has been well justified by the ease of testing additions and alterations to the radar program.

A further advantage of pseudo-real-time working with simulated data sources could be the possibility of extensive development of the programs either before or in parallel with the equipment. Hence the effects of the programming on the equipment could be available at the development stage rather than after production had started. If the early development is done before the computer is purchased it should preferably take place on a computer of the same type though some advantage can be obtained even if this is not so. Development work should also be done in the same high-level language, but a general purpose machine might be used, in which case only minor modifications to the program would be needed before transfer to the process control computer.

References

PHILLIPS, C. S. E. (1966). *Radar Techniques for Detection, Tracking and navigation*, Gordon and Breach, pp. 253–265.
PHILLIPS, C. S. E. (1967). Computer controlled adaptive radar, *Proc IEE*, Vol. 144, No. 7.
PHILLIPS, C. S. E. (1967). Networks for real time programming, *The Computer Journal*, Vol. 10, No. 1, pp. 46–52.