Real-time programming using a real-time language of intermediate level

B. EICHENAUER
Elektronik System GmbH, München, W. Germany

1. Introduction

The higher programming languages such as ALGOL 60, FORTRAN and PL/1 for digital computer systems used in business, technology and science have enabled users with little knowledge of the systems themselves to use them cost-effectively in practice. On the other hand, users who wish to solve real-time problems by digital computers must have comprehensive experience in handling their particular types of machines, and have to spend much time in formulation and debugging of programs.

The last few years have seen several attempts to simplify the use of digital computers in the field of real-time applications. Today, there are essentially three views on how the difficulties can be overcome.

The first way, pursued since the beginning of the sixties, involves the development of special-purpose packages [1, 2, 3] and problem-oriented languages [4, 5, 6, 7] for special fields in real-time applications.

Experience in the use of special-purpose packages and of problem-oriented languages shows that differences between problems in one application field are usually so substantial that only a small part of the field for which a package or a language was made can be covered. With the development of technology, packages and problem-oriented languages have to be modified, resulting in a great number of packages, problem-oriented languages and dialects of these languages. In the field of automatic check-out, for instance, there are at least 30 languages and dialects, but to my knowledge not one of these languages can completely formulate the simple function test problem treated below on language level.

The second way to overcome the difficulties involves the development of intermediate-level languages like CORAL66 [8], allowing only for the algorithmic program parts which can be translated in efficient code. In these languages most of the features relevant to real-time programming, i.e. tasking, timing and process-I/O, are left quite open. Real-time programming is carried out by calling the functions of the target-computer operating system.

This means of simplifying the problem may allow faster formulation of programs, as compared with assembler programming, but the main disadvantages of current methods are still present. For instance, every user must know the actual surrounding manufacturer software nearly as well as when using assembler languages. The documentation and the portability of programs is restricted by the dependence of the target computer.

The third method, used, for example, in INDAC8 [9], PAS1 [10] and PEARL [11], consists in adding to algorithmic language features of intermediate level a sufficient set of basic real-time and I/O features of comparable language level. As the algorithmic features of intermediate-level languages such as ALGOL or FORTRAN can be used to formulate any numerical algorithms, the basic real-time features should allow separation of application problems in loosely connected processes [12], and formulation of any correspondence of a process with the surrounding world.

A detailed explanation of the above set of language elements cannot be given here. Instead, we describe the solution of a simplified automation problem in the process and experiment automation real-time language PEARL, which we believe to be the most advanced language for real-time programming at present.

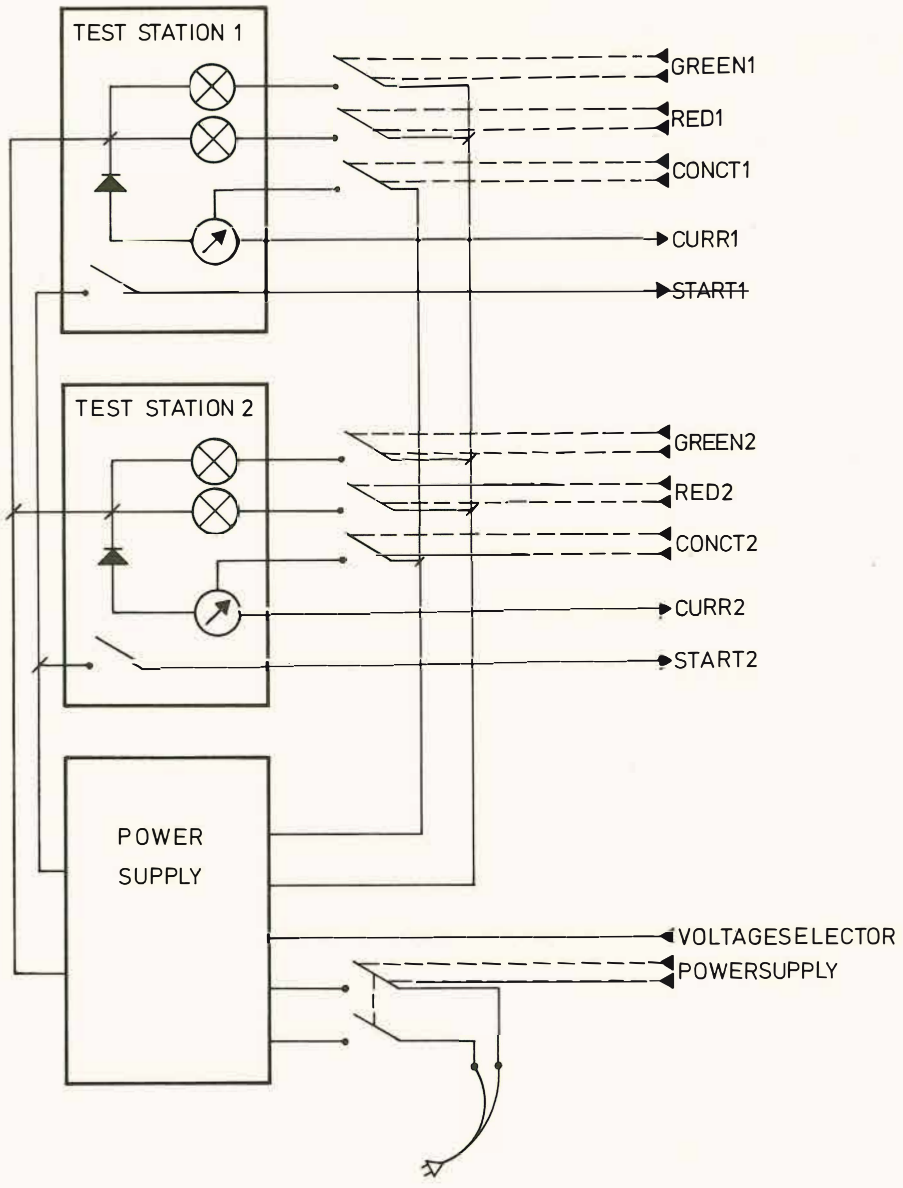

Fig. 1 Test installation

2. Programming example

The example is taken from the PEARL report [11].

2.1 Problem

Figure 1 shows a test installation consisting of two identical test stations and of a programmable power supply. The function test for four different types of diode has to be carried out at the test stations.

After the start of the automation program by the operator the computer switches on the power supplies of the two test stations (output to POWERSUPPLY) and adjusts the reverse voltage depending on the type of diode to be tested (output to VOLTAGESELECTOR).

The first step of the function test is to connect the diode to one of the test stations. Then the start button of the chosen test station is pressed to start the test program of the station (input from START1 resp. START2).

The test program applies the reverse voltage to the diode (output to CONCT1 resp. CONCT2), measures the reverse current (input from CURR1 resp. CURR2) and removes the reverse voltage (output to CONCT1 resp. CONCT2).

If the measured reverse current exceeds a type-dependent maximum, a red lamp on the instrument panel has to be switched on for 2 sec (output to RED1 resp. RED2). Otherwise a green lamp is switched on (output to GREEN1 resp. GREEN2).

The number of diodes not operating and the total number of diodes are counted for every test station and must be recorded at the end of the test period, or the type of diode must be changed.

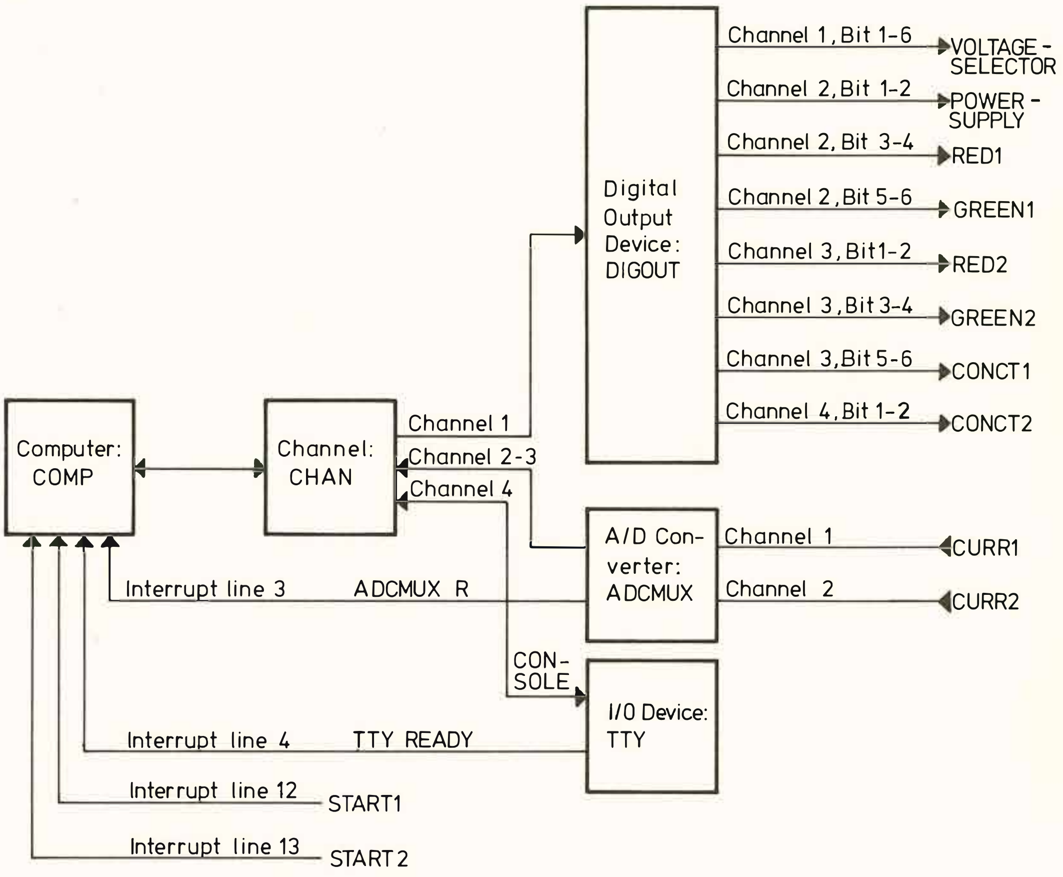

Fig. 2 Configuration of process periphery

2.2 *Configuration of the computing system*

Figure 2 shows the configuration of a hypothetical computing system servicing the test installation.

2.3 *Automation program*

The PEARL automation program consists of two program sections – the so-called SYSTEM-section and the PROBLEM-section. In the system section the configuration of the computing system shown in Fig. 2 is described. To allow for configuration-independent programming of automation algorithms in the problem division, all process end-points have to be identified by names.

The automation algorithm for the function test problem has been divided into three parallel resp. quasi-parallel executable program parts named TESTPROB, TEST1 and TEST2.

The main-task TESTPROB is activated by the operator by input to the standard-I/O-device CONSOLE (e.g. with the control statement: ACTIVATE TESTPROB PRIO 3).

To allow for parallel working of the test stations the service programs for the stations are isolated by introducing the tasks TEST1 and TEST2. TEST1 resp. TEST2 will be activated after interrupt START1 resp. START2 has occurred.

```
MODULE LIBRARY DIODETEST;
       /* PEARL EXAMPLE PROGRAM: FUNCTION TEST PROBLEM */
SYSTEM;
/*  CONNECTION OF STANDARD DEVICES:  */
   COMP              <->   CHAN;
   CHAN  *  1         ->   DIGOUT;
         *  2 * 1,12 <-    ADCMUX;
         *  4        <->   CONSOLE:  TTY;
```

```
/*  PROCESS END POINTS:  */
   VOLTAGESELECTOR:      <-     DIGOUT * 1 * 1,6;
   POWERSUPPLY:          <-            * 2 * 1,2;
   RED1:                 <-            * 2 * 3,2;
   GREEN1:               <-            * 2 * 5,2;
   RED2:                 <-            * 3 * 1,2;
   GREEN2:               <-            * 3 * 3,2;
   CONCT1:               <-            * 3 * 5,2;
   CONCT2:               <-            * 4 * 1,2;
   CURR1:                 ->    ADCMUX * 1;
   CURR2:                 ->           * 2;
/*  CONNECTIONS OF INTERRUPTS:  */
           CONTACT(3)     <-     ADCMUX * R;
           CONTACT(4)     <-     TTY * READY;
  START1:  CONTACT(12)    <-     ;
  START2:  CONTACT(13)    <-     ;
/*  END OF SYSTEM DIVISION  */
PROBLEM;
DCL (START1,START2) VAL INTERRUPT;
TASK TESTPROB GLOBAL HANDLE:
  BEGIN
  DCL            /*  STANDARD OUTPUT DEVICE:  */
       STANDARDWRITE VAL DVC = CONSOLE,
                 /*  VARIABLES TO COUNT THE NUMBER OF TESTOBJECTS:  */
       (TOTAL1,TOTAL2,NOGO1,NOGO2) INT := Ø,
                 /*  VARIABLE TO STORE THE UPPER LIMIT OF REVERSE CURRENT:  */
       TYPECURR INT,
                 /*  STRINGS TO ADJUST THE POWERSUPPLY TO REVERSE VOLTAGE:  */
       ADJUST(1:4) VAL BIN(6) = ('1ØØØ'B,'1ØØ'B,'1Ø'B,'1'B),
                 /*  UPPER LIMITS OF REVERSE CURRENT:  */
       MAXCURR(1:4) VAL INT = (5Ø,35,28,2Ø),
                 /*  COMMON CHARACTER STRINGS:  */
       TEXT1 VAL CHAR(13) = 'TEST STATION ',
       TEXT2 VAL CHAR(28) = ': NUMBER OF TESTED DIODES = ',
       TEXT3 VAL CHAR(27) = 'NUMBER OF INFERIOR GOODS = ',
                 /*  COMMON FORMATS:  */
       COM1 VAL FORMAT = F'S(13),S(1),S(28),,L',
       COM2 VAL FORMAT = F'(16)C,S(27)';
                -/*  TEST PROCEDURE:  */
  DCL  TEST PROC REENTRANT =
    ((UAMP,CONNECTOR,REDLAMP,GREENLAMP) VAL DVC,
            (TOTALNUMBER,NOGONUMBER)        INT )
    BEGIN  DCL CURRENT INT;
           TOTALNUMBER := TOTALNUMBER + 1;
               /*  MEASURE REVERSE CURRENT:  */
        MOVE '1'B TO CONNECTOR;
        MOVE UAMP TO CURRENT;
        MOVE '1Ø'B TO CONNECTOR;
```

```
               /*  CONTROL OF REVERSE CURRENT:  */
          IF CURRENT > TYPECURR OR CURRENT < 2 THEN
              /* NOGO BRANCH: */  MOVE '1'B TO REDLAMP;
                                  DELAY 2 SEC;
                                  MOVE '10'B TO REDLAMP;
                                  NOGONUMBER:=NOGONUMBER + 1;
                                                ELSE
              /* GO BRANCH: */    MOVE '1'B TO GREENLAMP;
                                  DELAY 2 SEC;
                                  MOVE '10'B TO GREENLAMP;
                                                       FI;
           END  /* OF TEST PROCEDURE  */ ;
              /* DECLARATION OF TASKS TO CONTROL THE TEST STATIONS:  */
TASK TEST1: CALL TEST(CURR1,CONCT1,RED1,GREEN1,TOTAL1,NOGO1);
TASK TEST2: CALL TEST(CURR2,CONCT2,RED2,GREEN2,TOTAL2,NOGO2);
              /* GET THE TYPENUMBER OF DIODE UNDER TEST:  */
TYPEDEF: WRITE (DATE,TIME,'TYPENUMBER := ')
                          FORMAT((2)(,(2)C),S(14));
         READ TYPECURR FROM CONSOLE FORMAT(,(2)L);
         IF TYPECURR < 1 OR TYPECURR > 4 THEN
            WRITE 'INPUT ERROR: UNDEFINED TYPE'
                          FORMAT((10)C,S(34),(2)L);
            GOTO TYPEDEF;                           FI;
              /* SWITCH ON AND ADJUST POWERSUPPLY: */
MOVE '1'B TO POWERSUPPLY;
DELAY 2 MIN;
MOVE ADJUST(TYPECURR) TO VOLTAGESELECTOR;
              /* STORE LIMIT OF REVERSE CURRENT AND INDICATE TEST BEGIN:  */
TYPECURR := MAXCURR(TYPECURR);
WRITE (TIME, 'TEST BEGIN') FORMAT (,S(10),(2)L);
              /* CONNECT START INTERRUPTS TO CONTROL TASKS: */
ON START1 ACTIVATE TEST1;
ON START2 ACTIVATE TEST2;
              /* SUSPEND MAIN TASK TESTPROB UNTIL OPERATORS' COMMAND: CONTINUE TESTPROB;  */
SUSPEND EXEPT TEST1,TEST2;
              /* RECORD OF TEST RESULTS:  */
WRITE (TIME,'  TEST RESULTS:',TEXT1,'1',TEXT2,TOTAL1,TEXT3,NOGO1,
                             TEXT1,'2',TEXT2,TOTAL2,TEXT3,NOGO2)
             FORMAT(,S(15),(2)((2)L,COM1,COM2),P);
END  /* OF MAIN TASK TESTPROB  */ ;
MODEND  /* OF MODULE DIODETEST  */ ;
```

1. '1800 process supervisory program (PROSPRO/1800)', IBM no. H 20-0473-1 (1968).
2. BATES, D.G., 'PROSPRO/1800', IEEE Transactions on Industrial Electronics and Control Instrumentation, Vol. IECI-15, No. 2, pp. 70-75 (December 1968).
3. 'BICEPS summary manual/BICEPS supervisory control', GE Proc. Comp. Dept., A GET-3539 (1969).
4. 'ATLAS abbreviated test language for avionics systems' ARINC Specification 416-1, Aeronautical Radio Inc. (June 1969).

5. METSKER, G. S., 'Checkout test language: an interpretive language designed for aerospace checkout tasks', Fall Joint Comp. Conf., pp.1329-1336 (1968).
6. 'Bendix OPTOL programming system', The Bendix Corporation, Teterboro, New Jersey (March 1968).
7. 'PLACE The compiler for the programming language for automatic checkout equipment', Battelle Memorial Institute, Columbus Laboratories, Technical Report AFAPL-TR-68-27 (May 1968).
8. 'Official definition of CORAL66', Inter-Establishment Committee for Computer Applications (February 1970).
CALLAWAY, A. A., 'A guide to CORAL programming', Royal Aircraft Establishment, Technical Report 70102 (June 1970).
9. 'INDAC8', Digital Equipment Corporation, Maynard, Mass.
10. 'PAS1 Prozess-Automationssprache 1', BBC Mannheim.
11. BRANDES, J., et al., 'PEARL: A proposal for a process and experiment automation real-time language', to be published by Projekt PDV, GFK Karlsruhe.
12. DIJKSTRA, E.W., 'Co-operating sequential processes', Programming Languages, F. Genuys (Ed.), Academic Press, London (1968).