APPLYING CDL FOR REAL-TIME PROGRAMMING OF MICROPROCESSORS

J. SARBÓ

Department of Instrumentation and Metrology, Technical University, H-1521 Budapest

Received January 12, 1983
Presented by Prof. Dr. L. SCHNELL

Summary

Concrete examples have been presented to illustrate the efficiency of applying CDL and CAL for developing real-time programs for microprocessor-based devices, recently constructed at the Department.

The example is a protection switching equipment, part of a complex telecommunication system developed on commission of the Research Institute of Telecommunication. According to complexity and program size, the equipment belongs to category 2 [4]. The sequential algorithms defining the program size have been prepared in a high-level language (CDL — Compiler Description Language) with inherent advantages, at the same time permitting to achieve storage capacity and speed near the assembly level. The necessary real-time monitor has been made by means of a macro-level monitor generating system (CAL — Concurrent Assembly Language), short-run interrupt handlers have been programmed in assembly language.

This concrete example will illustrate experience obtained in applying CDL-based software technology and CAL macro-library developed at the Department.

Protection switching equipment

A part unit of a multi-channel microwave telecommunication control system has to be developed. The system controls one or more transmission chains composed of sections. The protection switching equipment handles one section of the telecommunication chain. There is a two-way communication on the seven channels of each section. Besides there is a stand-by channel in both directions. The quality of channels is continuously measured by a special hardware that indicates a switching demand if given quality conditions occur. The channels — beside their actual quality characteristics — have a priority, in addition, the operator can instruct channels to configuration, measurement or testing, the equipment protects the channel qualified the most important by quality, priority, etc. characteristics to the reserve channel. Another time, e.g. in case of a demand to measure it transfers the output of a generator of controlled quality to the chosen channel.

Each section consists of receiver-transmitter pairs according to Fig. 1.

In the section the master function is generally supplied by the receiver side except some cases. The information exchange between receiver-transmitter pairs far away proceeds in both directions along a preferential, so-called main channel — one of the controlled channels – in serial mode.

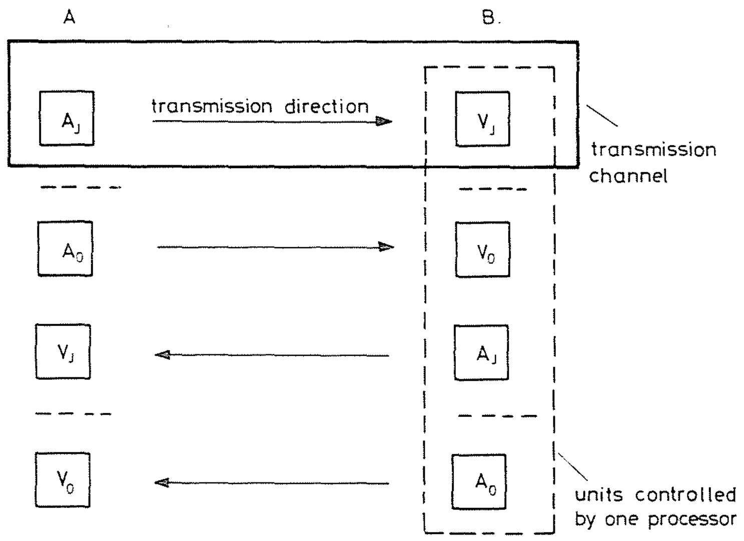

Fig. 1. Receiver-transmitter pairs

Upon random quality changes and operator demands, the transmission channels compete for one resource power (protecting channel). Besides of the system asymmetry, the complexity is increased by the specified short switching time (below 40 msec) and some other special restrictions. It is an important fact that the protection switching equipment is part of a system of high reliability. Every process is protected by timing and in case of deadlock the switching program is restarted by a so-called "software watchdog".

A detail of the real-time structure of the equipment is seen in Fig. 2.

The problem has been found to be typical of those arising in programs of devices category 2, the most important being:

— the program has to function in real-time environment (4–10 processes);

— there exist time-critical parts;

— complexity of the problem increases the size of sequential program parts.

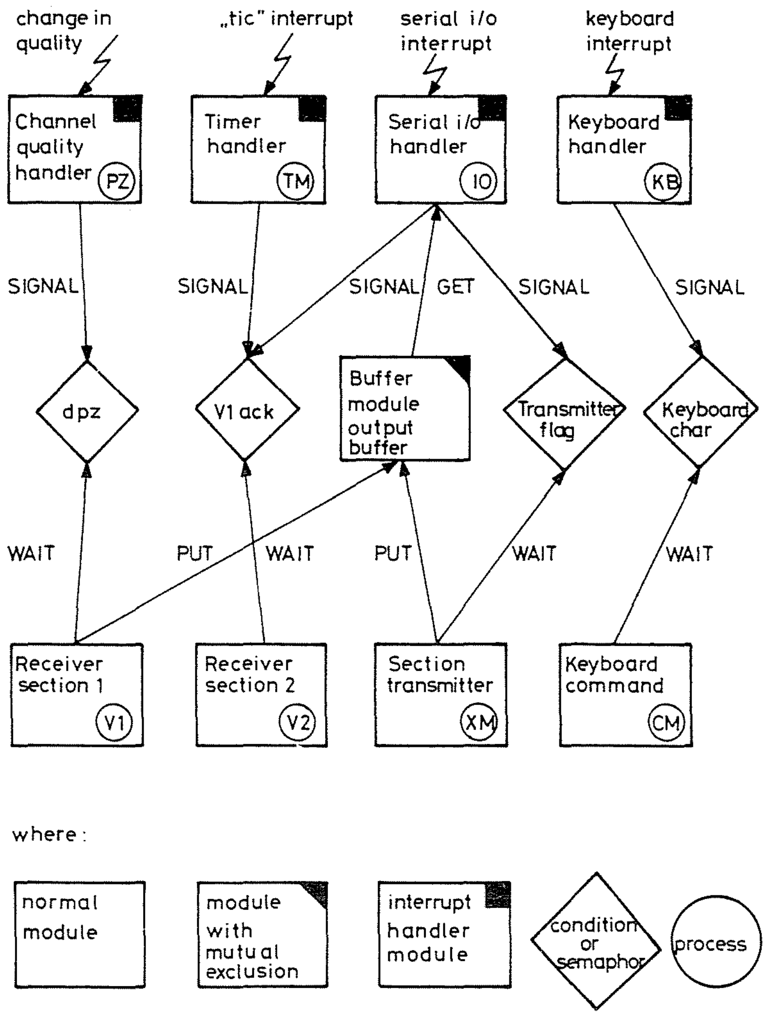

Fig. 2. Real-time structure

Application experiences

The device program has been realized in four, distinct phases:

1. Programming and debugging of version CDL→PDP-11 of the switching algorithm.
2. Adaptation of the CDL program to microprocessor Z80.
3. Generating and testing CAL monitor running on Z80.
4. Fitting and testing the switching program and the CAL monitor.

ad 1.

Coding of the channel switching algorithm was preceeded by a long and deep-going specification process. Meanwhile it became clear that the score of unique boundary conditions would excessively complicate the testing algorithm, involving control all of meaningful input combinations of about 40 inputs. The high number of input combinations induced us to functional testing. To this aim an interactive simulator system assuring simple control of the 40 inputs had to be produced.

The simulator, again written in CDL, maps the hardware handled by the channel switching program onto the display. All kinds of meaningful state transitions can be produced in simulated hardware. Also the system responses on state transitions (generations) appear on the display. For both CDL programs an experienced CDL-programmer needed less than four weeks to get from coding to the syntactically correct versions of the programs.

Program sizes:

CST (channel switching)	1700 lines (5.5 Kb)
SZM (simulator)	1500 lines (4.5 Kb).

From the aspect of algorithm, the simulator is essentially simpler than the channel switching program. Its relatively large size is due to the comprised algorithms for display handling, command processing and hardware simulation. As a matter of fact its programming did not strive for reducing its size.

In preparing the programs the types, date of their occurrence time, etc. of committed errors have been recorded. Only errors found during debugging the programs were recorded until all the developed programs began to normally function. Therefore no data on how the programs work are available. Neither the number of errors subsisting in each program can be told, although some publications recommend estimation by error statistics taken during program debugging. Errors can be detected on the basis of program source text or program run (answer given on some input). The ratio of errors from either source has been detailed in Tables 1 and 2. The most of errors are seen in Table 1

Table 1

Error types of switching program (CST)

error type	work day						
	1	2	3	4	5	6	7
misprint:	(0.15)	(4.3)	(1.4)	(2.2)	(1.0)	(2.0)	–
assembly error:	(0.9)	(3.2)	(1.3)	(2.0)	(2.2)	(2.4)	(2.0)
CDL alg. error:	(0.7)	(1.2)	(7.0)	(1.3)	(2.1)	(2.1)	(1.0)
change in specif.:	–	–	–	(0.1)	–	–	–
initiation error:	–	–	(3.0)	(2.0)	(1.0)	(1.1)	–
number of error groups:		⟨..10..⟩		⟨....15....⟩			

where (x, y): x run-time; y number of errors found in the source list

Table 2

Error types of simulator program

error type	work day						
	1	2	3	4	5	6	7
misprint:	(0.13)	–	(1.1)	(3.2)	–	(1.0)	–
assembly error:	(0.6)	–	(1.0)	–	–	–	–
CDL alg. error:	(0.4)	(0.3)	(3.3)	(3.2)	(3.0)	(4.1)	–
change in specific.:	–	–	–	–	(0.1)	–	–
initiation error:	–	–	–	(0.2)	–	–	–
number of error groups:			⟨..5..⟩	⟨....4....⟩			

to have been found during the first analysis — based on a source program list — of the syntactically correct program. There after the number of errors recognized in the channel switching program has essentially diminished, attributable to a severe error causing at the first run the program to, "kill" the operating system RT-11, inhibiting its localization.

Again, misprints were found even by the 6th time. Most of misprints could have been screened out on the basis of the first program list by having had the program text (especially the assembly algorithms) checked.

Programming errors are generally not evenly distributed in the program text. An error found — in the initial stage of debugging — may eliminate several errors (group of errors), thus the running time is correlated with the number of error-groups found, rather than with the total number of corrected errors.*

ad 2.

Transfer means a semantically equivalent transcript of CDL-macros. Size of the programs (after debugging on PDP-11):

Channel switching program is 1800 lines, among them 300 lines (cca 15%) are CDL-macros. Final size of the simulator program: 1700 lines, among them 200 lines are CDL-macros.

This ratio is 10 to 30% in case of other CDL-programs made by us.

Transfer of macros is a mechanical work but easy to go wrong. Due to the small number of macros, transcript involved no error, version Z80 of the CDL program worked immediately. The situation, however, is usually not so good. In case of bigger programs in the rewriting phase about 10 to 20% of macros are bungled. Essentially, if the program doesn't work on the target-system after porting, the error resides exclusively in the transcript of macros. It should be emphasized since the target-system (Z80) has only assembly-level debugging facility.

* Error group means mainly typical errors (e.g. misprints), different errors made in a small logical unit and consequences of errors.

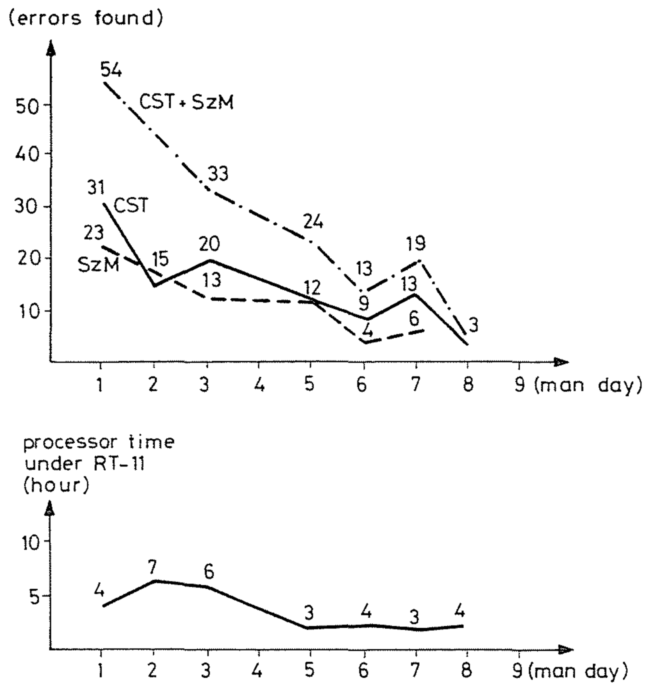

Fig. 3. Processor-time and error statistics of CDL-PDP-11 programs

ad 3.

CAL was not applied before by the author for programming real-time monitors, a fact responsible for difficulties in preparing the monitor.

The monitor had to be defined by CAL macrocalls. A reference book was available to this. Programming of the monitor is a mechanical work after having specified the processes. On the basis of the specification an experienced programmer does it in a few hours; a syntactically correct monitor can be produced in 1 or 2 days.

The monitor has been debugged without building in sequential algorithms and interrupt-handlers. The operation of interrupts has been checked on the empty monitor and the scheduling algorithm of the monitor tested by adjusting the synchronizing means (semaphors, conditions, etc.). The number and types of errors retrieved have been compiled in Fig. 4 and Table 3.

The first trial produced few errors but acquaintance with the monitor. On the second occasion more than 50% of all the errors were found.

The monitor has been compiled under RSTS-E, much slower than the operating system RT-11. As CAL hides the monitor's code before the user and little experience with CAL was available, the monitor was re-compiled after each error (or group of error) found, protracting the development. Size of the monitor: 650 lines (2.5 Kbyte).*

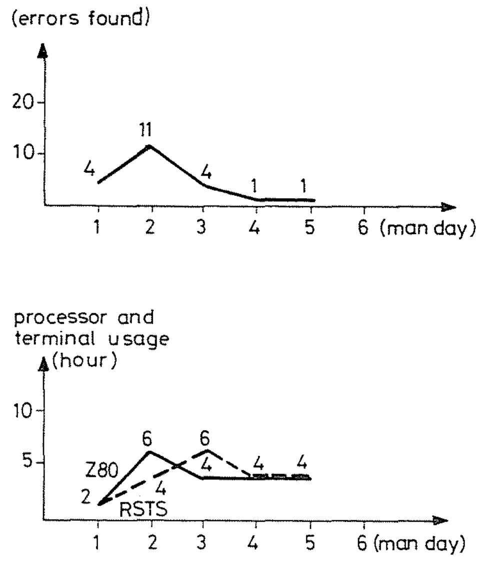

Fig. 4. Error and processor-time statistics of CAL program

Table 3

Error types of CAL program

error type	work day				
	1	2	3	4	5
misprint:	1	2	–	1	2
assembly error:	1	1	1	–	1
CAL applic. error:	–	5	2	1	1
change in specific.:	–	1	–	–	–
initiation error:	1	2	–	–	–
application error of system Z80:	1	–	1	–	–

Evaluation of results

The measured data permitted to compute some characteristics of importance. Speed of error correction as a function of re-compilations:

CST: 103/12 = 8.5 errors/compilation
i.e. 103/25 = 4 errors/group of errors
25/12 = 2 group of errors/compilation

* RSTS-E: Time-sharing operating system of PDP-11, RT-11: single-user real-time operating system of PDP-11

SZM: 58/5 = 12 errors/compilation
i.e. 58/9 = 6.5 errors/group of errors
9/5 = 2 group of errors/compilation
CAL: 21/9 = 2.3 errors/compilation, unique errors.

The number of errors corrected in each compilation point out different complexities of the two CAL programs. In the monitor unique errors were prevalent owing to its small size.

In conformity with the above and with the speed difference between processors PDP-11 and Z80, efficiency of used processor time is the following:

RT-11: 103/34 = 3 errors/hour

Z80: 21/20 = 1 error/hour.

It is also worth mentioning that, referred to a finished CDL program, one error has occurred each 27 lines, and in a CAL program, each 32 lines.

Portability-based software technology resulted in a program of good quality in a reasonable time. According to our experience the same in assembly (by using CAL) would have lasted about 2–3 times longer, mainly because of the lengthy debugging, testing work.

The adaptation process itself was fairly quick. Also other CDL programs made at the Department worked in at most 3 to 4 weeks on microprocessor.

References

1. KOSTER, C. H. A.: A Compiler, Compiler Matematisch Centrum Amsterdam, MR 127
2. Concurrent Assembly Language (CAL) Programming Handbook, Department of Instrumentation and Metrology, Technical University, Budapest (1980)
3. HANÁK, P.–RÁCZ, G.–SARBÓ, J.: Microprocessor Software Technology System, Programming System '81 Conference, Szeged
4. SARBÓ, J.–RÁCZ, G.: An Approach to Real-Time Microprocessor Programming, Periodica Polytechnica, El. Eng. **27** (1983)

János SARBÓ H-1521 Budapest