CLM-P843

CLM-P843

Use of FORTH in integrated control system development

R. Endsor

UK ATOMIC ENERGY AUTHORITY

Culham Laboratory

Use of FORTH in integrated control system development

R. Endsor

1. INTRODUCTION

FORTH is a structured computing system based on multi-layered symbolic operators. These operators essentially process data stored on a stack and use this stack directly for parameter passing. Parameter passing is untyped in the sense that there is no syntactic checking of stacked items when an operator is invoked.

At the heart of FORTH is a compiler/interpreter manipulating operator definitions in a single dictionary data structure. New operators are defined in terms of sequences of existing operators or by using assembler code for the underlying computer using FORTH defined assembler operators. There is no distinction between symbols defined in these different ways as each symbol appears in the dictionary followed by a "data structure" defining its behaviour. Execution of FORTH code by the interpreter uses the indirect, threaded code generated by succeeding layers of operator definitions to access the "primitive" operators written in FORTH assembler.

The FORTH system is an open system in that all system code is defined in the dictionary and can be invoked by subsequent additions to the dictionary. This includes the symbols which make up the compiler/interpreter. Usually a FORTH system, supplied with a computer, implements a standard set of basic arithmetic, logical and character operations and performs basic system operations such as controlling terminal input/output, disk operations, printing and task scheduling. The FORTH compiler/interpreter, together with its assembler, is also supplied, as is a screen based editor. A FORTH user application is then built by the addition and invocation of further symbolic operators. This distinction is somewhat blurred by the fact that the user application may include redefinition of any of the underlying FORTH symbols so modifying or extending the computer system.

The multi-layered, open structure of FORTH gives rise to a powerful form of structural abstraction in that an application can be expressed in a user defined abstract language which may be executed in different modes and configurations of underlying hardware. This arises from:-

(a) The ability to redefine FORTH compiler operators to form a compiler for an overlying, user defined language.

(b) Changes to underlying system operators to map onto different hardware configurations; to invoke either internal memory operations or external channel signalling.

These features of FORTH have been exploited in a particular control system project which will be briefly described with particular reference to the relevance of FORTH's structural abstraction.

2. RDS PROJECT

The COMPASS Requirements Definition System (RDS) is briefly described, from the user viewpoint, in an accompanying paper[2]. The aim of the system was to define a distributed control system such that individual sub-system

*FORTH is a trademark of FORTH Inc, CA

components and a central control system could be constructed by contractors. (The sub-system would be based on Programmable Logic Controllers and the Central System on a digital computer.) The operation of the control system was described; partly by an Input/Output Database listing all analogue and digital signals passing between the controlled plant and the sub-system and all signals passing between the sub-systems and the central control system. Operation of the system components was described by concurrent processes expressed as logical operations in a language known as the Requirements Definition Language (RDL). Timing of operations could be expressed by timer variables.

To check the logical completeness of the control system definition, the users could also construct RDL processes exhibiting the responses of the controlled plant and then run these with the processes representing the control system in a simulation mode. A separate language, known as a Display Definition Language (DDL), allowed mimic diagrams, exhibiting total system operation, to be displayed on attached screens. DDL also allowed control panel simulation by allowing attached mice to be used as pointers. A logging file enabled operation of the system to be recorded for later checking.

It was decided that FORTH was a suitable system in which to implement RDS. The implementation divided into three separate components. The Input/Output Database was expressed as FORTH variables and known as the Database Definition Language (DBDL). Systems operations were expressed in RDL and the system operation was displayed using DDL. As the DDL was only used for system checking during specification and was not part of the specification itself it was decided that DDL could be implemented as an addition to the underlying FORTH system and be expressed in a "basic" FORTH style (ie with input parameters being expressed before the operator in reverse Polish form). DBDL, which essentially consisted of a list of sub-system input/output signals of either analogue or digital form, could be expressed as "basic" FORTH variables. However, it was decided that RDL should express the operation of the sub-system in a way which would be immediately comprehensible to both the specifiers and implementers. RDL was, therefore, designed as a "standard" infix language with usual operator precedence which could be overriden by brackets. Assignment statements were allowed; also synchronisation statements using timer variables and structured conditional statements. The structured conditional statements could be nested and parameterless procedures could be defined and invoked. It was decided that the FORTH compiler/interpreter could be modified to act as a compiler for RDL.

3. IMPLEMENTATION OF A 'HIGH-LEVEL' COMPILER

The RDL language allowed expressions to be written in infix notation. These operated on logical or integer variables and constants. The definitions of standard FORTH operators (such as +, *, AND etc) were changed in two ways. Firstly, a precedence was attached to each operator allowing "normal" infix interpretation of expressions (such as A + B * C) and then the compile time action was altered. Instead of generating the indirect pointer to the operator coding at the current position in the dictionary (this would occur in the definition of the current RDL process), the indirect pointer, together with its precedence, would be stored on an 'alternate' stack (ie not the normal FORTH stack). At the same time, any indirect pointers to operators already stored on the alternate stack, of higher precedence, would be moved to the current position in the dictionary definition. At the termination of the expression, the alternate stack would be

flushed and any remaining entries moved in precedence order to the current dictionary position. Bracket symbols were defined and caused the current precedence to be suitably incremented and decremented to enable correct evaluation of expressions such as (A + B) * C.

FORTH is structured such that redefinition of all existing symbols is straight forward. A new definition of the symbol will always be found, on future reference, in preference to an existing definition. The existing definition, however, still remains in the dictionary and may be referred to during the creation of the new definition. Extra actions can, therefore, always be added to any existing symbol. If, at some future stage, the new definition was removed from the dictionary, then FORTH would revert to using the previous definition.

Overall syntax checking for correct nesting of RDL process and conditional statement usage was performed by creating a state checking symbol operating on a single state variable. The various RDL keywords (such as "process", "operation", "continue-process", "if" etc) were defined, using the state checking symbol to check the existing state and then perform the desired transition if this were valid. Run-time behaviour relied on compiling indirect pointers to underlying FORTH operators, such as IF and THEN.

The RDL compiler using the FORTH compiler/interpreter was relatively succinct occupying about 120 lines of FORTH statements. It had, however, two weaknesses that could have been avoided if a separate RDL parser/code generator (which would have required more FORTH code) had been written. Firstly, the implementation required that all RDL elements were space separated when used to define user processes. This arose from the FORTH interpreter's symbol definition being based on space separation. Secondly, error conditions often produced terse responses. The least helpful was probably a question mark following what FORTH considered to be an undefined symbol. This could arise if the user of RDL had forgotten to define a variable in the accompanying DBDL before using it or failed to separate two elements by a space.

4. CONCURRENCY OF SIMULATION

Once user defined RDL processes had been compiled, these were run in simulation mode, with accompanying DDL screen displays, to ensure that correct system behaviour had been specified. Each process was linked into the FORTH "round-robin" scheduler as a background task. The scheduler worked on a linked list, each task running until it executed a PAUSE statement. This caused the state of the task to be stored and the following task in the list continued after its previous PAUSE. To ensure that execution of RDL processes appeared to be concurrent, PAUSE statements were compiler into code generated by RDL assignment statements. This ensured that a change of a variable in one process could be "immediately" detected and acted upon in other processes. This was especially useful as each process could have an exception condition associated with it. In a control system this could be, for example, an earthing failure. The compiler FORTH code checked the exception condition after each entry from a previous PAUSE. Thus, one RDL process detecting a failure could set a variable which would trigger the exception conditions in other processes and lead to a system "close-down".

The DDL language was interpreted for display on two associated screens. One screen was designated as a Control screen and the other a Status screen. Associated with each screen was a mouse which allowed a software driven pointer to be moved around on the screen. An interactive FORTH process ran for each screen; this

being linked into the FORTH round-robin. DDL statements allowed the values of DBDL variables to be monitored and any changes to be displayed. Also areas of each screen could be designated as "Hit" areas and specified actions performed whenever the button of a mouse pointing to that area was pressed. These actions could simulate the System Controller pressing a "START" button in the simulated system.

5. RDS MEMORY ENHANCEMENT

The original RDS system was implemented on a DEC LSI-11 processor using 16 bit addressing to access the 64 Kbytes of main store. This meant that the basic FORTH system, the definitions for RDS and the user-specified control system simulation all had to fit into 56 Kbytes (the final 4 Kbytes being used for system input/output). This limitation proved to be a restriction and DEC had introduced, for other systems, a hardware memory mapping unit which allowed addressing up to 22 bits. Extra 64 Kbyte main store modules were also available allowing up to 256 Kbytes to be attached to the processor (using an 18 bit backplane). It was decided that RDS should be enhanced to support DBDL variables and RDL user processes running in extended memory with memory mapping.

Memory mapping was based on two sets of eight 16 bit registers, known as the Kernel and User registers. The FORTH system was modified to initialise these registers with seven Kernel registers pointing to the first 56 Kbytes of main store and the final Kernel register pointing to the last 4 Kbytes. FORTH system loading then continued in Kernel mode. The basic FORTH system and a modified RDS system were then loaded into the first 40 Kbytes of main store. The next 16 Kbytes of main store were reserved for loading a FORTH defined screen editor and for "Hit" and "Scan" tables for the Control and Status screens.

The modified RDS system contained renamed definitions of the FORTH dictionary handling routines which allowed access to a further directory to be stored in extended main store. When compiling the user defined control system, the dictionary routines would switch the processor from Kernel to User state and create user DBDL definitions in 56 Kbytes of extended memory. Data storage for variables associated with these definitions was assigned in a further 24 Kbytes of extended memory. Code generated by the RDL processes was mapped into 16 Kbytes of extended memory. Further areas of extended memory were assigned for run-time workspace for each RDL process. All these areas had previously been mapped into the DBDL variable and RDL process dictionary entries.

When the simulation was to be run the FORTH round-robin included the foreground tasks for the Control and Status screens and a single entry for a small, specially written extended memory scheduler. When entered this scheduler would either immediately exit or map three Kernel registers to run each RDL defined process in turn. Once every process had its turn the scheduler reset the Kernel registers to point to the initialised FORTH state and re-entered the FORTH round-robin. When an RDL process was running it accessed variable values by special fetch and store actions which switched the processor to User state, accessed the variable and then returned the processor to Kernel state.

It should be emphasised that as far as the users of RDS were concerned, the extended memory system was identical to the original RDS system and allowed RDS defined systems to be run without modification (of course the extension allowed far more extensive systems to be defined). The important point of the extension was the flexibility of the FORTH system and the relatively short period in which the modified

system was produced by an experienced FORTH system programmer.

6. TEST SYSTEM MODE

The final area in which the RDS system was extended is of more general interest. RDS was used to create specifications for supply of a distributed control system based on Programmable Logic Controllers and a Central Computer. These were to be produced by an external contractor and there was a need for the specifiers to check that the delivered system met the specification. The LSI-11 system used for RDS had the Hytec modifications allowing access to a CAMAC crate. This was the specified communication medium for the control system; it was, therefore, decided to link the controlled plant models, written in RDL for the original simulation, to the delivered control system. The operation of the delivered system could then be checked using the Control or Status screens to monitor the operation. This method of testing allowed operation of the control system to be tested before the plant which it was to control had been delivered.

The required modification to RDS was to map specified DBDL variables to signals on Digital and Analogue channels in input/output modules accessible through the CAMAC crate. Some modification to the specification DBDL was required as the user had to intersperse a few statements giving the module addresses to which the following DBDL input/output variables would correspond (the specifications have already been drawn up with a clear definition of the grouping in which control signals would be wired on the control system). Code to initialise and perform input/output to CAMAC modules was written in about 100 lines of FORTH.

It remained for RDL and DDL to be modified to handle both CAMAC variables and memory variables in main store. One way would have been to test for variable type when accessing the variable and then perform different actions. FORTH offers instead a more structured solution. With the definition of each variable type it is possible to store references to separate store and fetch actions. Generic store and fetch operators may then be defined which are called whenever a DBDL variable is to be accessed. These generic operators then use the indirect pointers in the FORTH-generated data structures to access the actions needed for the current variable and return, if necessary, the required result on the stack (in the case of Analogue input/output appropriate scaling is also performed). Using this method a Test Mode RDS system could be constructed from the Simulation Mode coding with changes to about 40 lines of code.

7. CONCLUSIONS

FORTH offers an open, multi-layered structure on which flexible, abstract systems may be built. This has been demonstrated by RDS where an initial Requirements and Simulation system has been transformed into a tightly-coupled Validation and Acceptance system. The ability to base different stages of system development on an abstract, high-level specification is seen as an advance in system development techniques.

ACKNOWLEDGEMENTS

The FORTH implementation of RDS was performed mainly by Chris Stevens and Brian Mercer of Computer Solutions Ltd (UK) under contract to the UKAEA.

REFERENCES

1. L Brodie, Thinking FORTH, (Prentice-Hall 1986).

2. D C Edwards et al, COMPASS Electrical Systems Development and Commissioning, Proc. 15th SOFT Conference.