UDC 681.3.06:[681.323:62-523]

INTERMEDIATE FUNCTIONAL CONTROL LANGUAGE, FCL

Haruki Tanaka
Yutaka Yoshida
Keiji Ishibashi

1. FOREWORD

Along with the development of the semiconductor technology, the ability of programmable controller (PC) has been enhanced remarkably, and the degree of its popularization has been increased greatly.

Accordingly, multifarious programmable controllers have been developed for various uses, and a number of representative forms coexist as program language according to respective uses or the preference of each user. So far, Instruction List, Ladder Diagram, Function Block Diagram, Decision Table and GRAFCET etc. have been used as PC programming language. This disunited situation of PC programming language is becoming a serious obstacle to the development of process control domain.

In addition, these languages depend greatly upon the hardware of each PC in many cases, thus many programs having high machine dependency are existing nowadays.

Under such situation, we have developed an intermediate language for PC having the functional structure. It enables to make standardized PC software independent of both machine and representative form of program language. Thsu FCL improves the portability of PC programs.

This paper introduces the basic concept, the basic structure of FCL and some examples of its actual applications.

2. WHAT IS AN INTERMEDIATE FUNCTIONAL LANGUAGE

2.1 Purpose of development

A number of representation methods such as Mnemonic description, Ladder diagram, Function Block diagram, SFC (Sequential Function Chart), Mark Flow Graph and Decision table have been used as the program representation of control specification. Each representation method has merits or demerits in representation ability and being easy to see, so they should be used selectively according to the PC application. Generally, a conventional control equipment limits the program representation only to one kind or a few kinds because of the restriction of hardware. Therefore, the users cannot select a representation method freely, and the software which is made up for one certain machine will not run on the other type of machine.

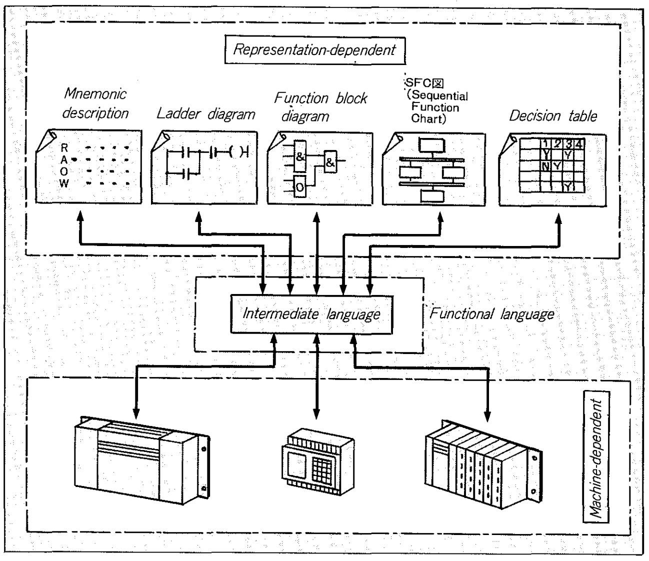
The Functional Control Language FCL has been developed to cope with such difficulties and enable to make the standardized PC software which is independent of both machine and program representation. This is an intermediate language for various program representation to satisfy the following purposes:

(1) Portability of user software
(2) Isolation of program descriptive representation from hardware
(3) Encouragement for progress of program descriptive representation
(4) Encouragement for progress of hardware architecture
(5) High representation ability
(6) Easy conversion from and into graphic representation
(7) Easy execution of intermediate language

2.2 Position of FCL

The Functional Control Language FCL is the intermediate language situated between the representation dependent part and machine dependent part as shown in *Fig. 1.*

Fig. 1 Position of functional language FCL



The program representations such as Ladder diagram, Function block diagram and SFC diagram are converted into the standardized intermediate Functional Language, or the latter is converted into the former vice versa. The various controllers, machine dependent part, execute the machine language converted from the intermediate language or directly the intermediate language, regardless of the program representative form. Therefore, the representation dependent part and machine dependent part are isolated completely from each other, and program having any representation form can be run on any controller.

In addition, it will be assured that all the programs produced so far can be run on any new controller which may be developed in the future, and any program having newly developed representation form can be run on any conventional type controller.

Mutual translation is also permitted between program representations, if they have equivalent representation ability.

2.3 Basic structure of FCL

The basic form of FCL uses a set of parentheses "(   )" as one unit, shown in *Fig. 2.*

A function and arguments are aligned within "(   )" to form one functional expression. One functional expression is sure to have one value, thus permitting the functional expression itself to be described as an argument. The FCL is a language to describe specifications in combination of such functions, and it complies with the basic form of the LISP language. (G.L. Steel Jr. et al., 1984).

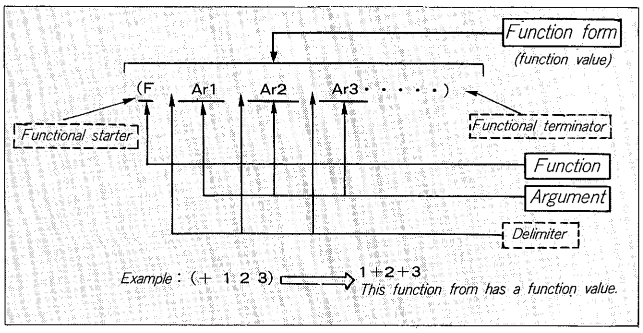
The major rules for the basic form are as follows:

(1) A functional expression starts with a function starter "(" and ends with a function terminator ")".

(2) The function is placed immediately after the function starter "(".

(3) The function is provided with 0 or more argument.

Fig. 2 Basic form of FCL



Function form
(function value)
(F Ar1 Ar2 Ar3 · · · · ·)
Functional starter
Functional terminator
Function
Argument
Delimiter
Example : (+ 1 2 3) ⟹ 1+2+3
This function from has a function value.

Fig. 3 Example of function combination

```
(FF1 (F2 Ar1 Ar2) (F3 Ar3 Ar4))

Example : (*  (+ 3 4)  (−7 5) )
                    ⟹ (3+4) * (7−5) のこと。
```

(4) A constant, veriable or functional expression is permitted as an argument.

(5) The function and argument, or two argument are separated from each other by a separator "space".

(6) When a functional expression is executed, one function value is sent back.

Since a functional expression is allowed as an argument, functions can be combined together to permit a flexible representation, as shown in *Fig. 3.*

2.4 Features of functional representation

The functional representation has the following features:

2.4.1 Simple structure

The functional representatio has a simple structure, that is, function(s) and argument(s) are bracketed with parentheses. Even a complicated functional description can be realized by combination of such simple basic structures. Therefore, the compiler and discompiler of Functional Language can be easily produced, and executing process of Functional Language is very simple. Thus, not only inter-

Fig. 4 Representation ability of functional structure

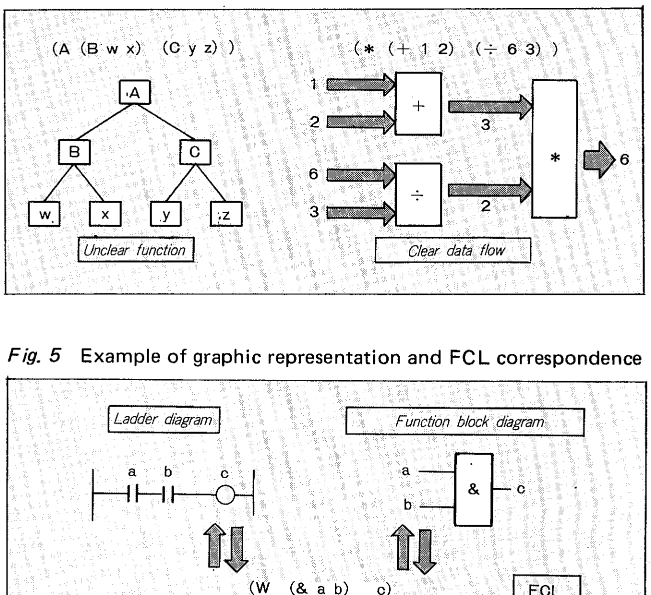
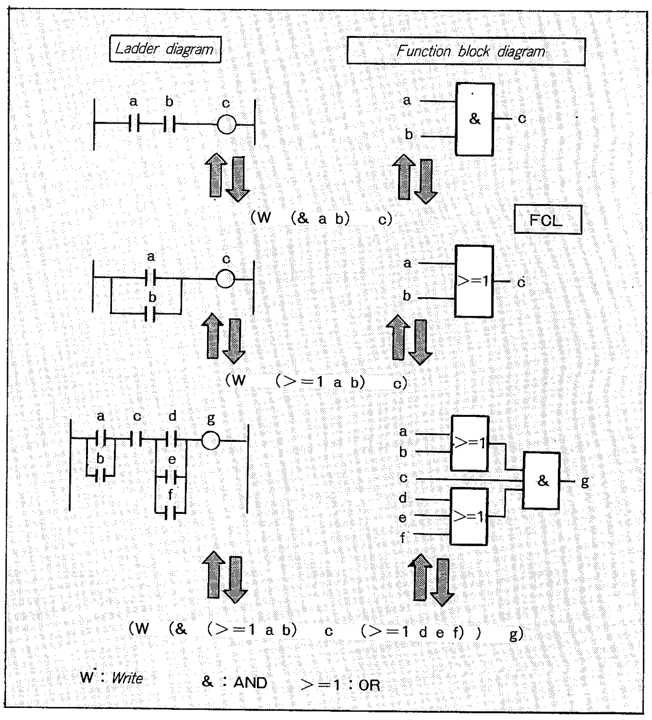


Fig. 5 Example of graphic representation and FCL correspondence



preting by software, but also direct execution by LSI can be realized.

2.4.2 High representation ability

Functional representation can show the explicit "relations" among the individual functions, the explicit "data flow" indicating data flow between functions, and processing parallelism permitting parallel processing in the future. Therefore, the functional representation is suitable for semantic representation and conceptual representation, and permits the "high representation ability" and "affinity with graphic representation" required for intermediate language.

2.5 Example of correspondence with graphic representation

An example of correspondence between function block diagram and ladder diagram and FCL is shown in *Fig. 5*.

The important thing is that the graphic representations and FCL can be converted into each other. As can be seen from this, translation between graphic representation with the same representation ability is possible via intermediate language FCL.

In addition, the SFC which will be adopted as IEC standard can also be bidirectionally converted with FCL.

3. FCL LANGUAGE SPECIFICATIONS

3.1 Features of FCL

The features of FCL are as follows:

(1) There are a number of available basic functions such as logic operation, and bit processing suitable for control, as well as arithmetic floating operation.

(2) State variables can be defined, so not only the combinational logic, but also sequential logic can be described easily.

(3) Owing to the structured functions (if-then-else, do-while etc.), even the program of complicated flow can be made up visibly in a form easy to represent graphically.

(4) Special functions useful in the PC application can be defined as macro function, and thereby the descriptive ability can be enhanced.

(5) Real time description such as Task enter and Task priority definition is permitted.

3.2 Classification of FCL

The language classification of FCL is shown in *Fig. 6*.

3.3 Types of data handled by FCL

FCL can handle various kinds of data types, as shown in *Fig. 7*.

3.4 FCL functions

About 120 functions are available with FCL. Typical functions are listed in *Table 1*.

Fig. 6 Classification of FCL

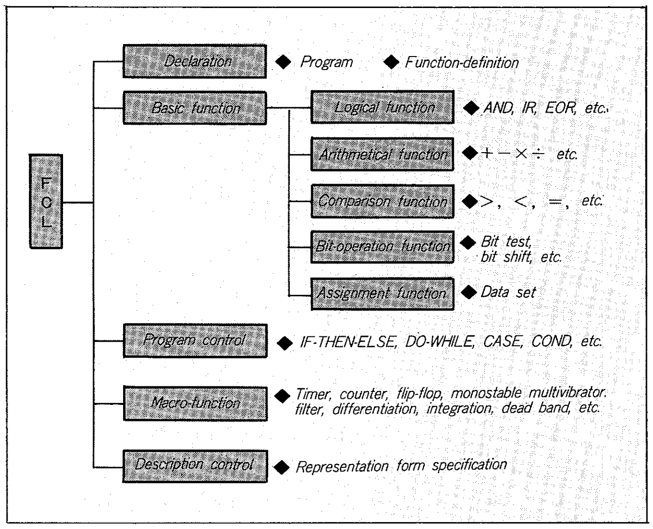
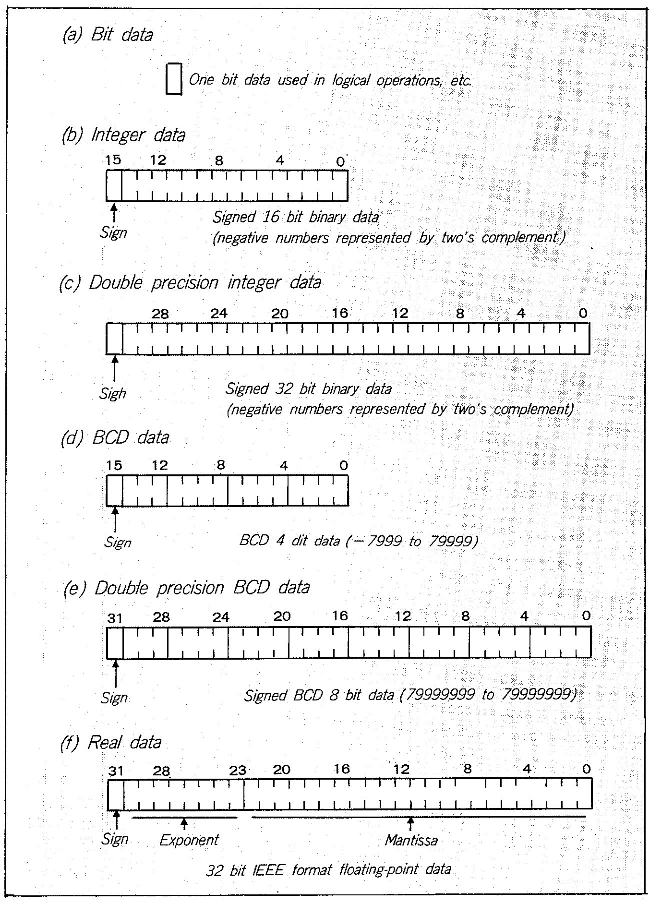


Fig. 7 Types of data handled by FCL


(a) Bit data
One bit data used in logical operations, etc.
(b) Integer data
15 12 8 4 0
Sign
Signed 16 bit binary data
(negative numbers represented by two's complement)
(c) Double precision integer data
28 24 20 16 12 8 4 0
Sigh
Signed 32 bit binary data
(negative numbers represented by two's complement)
(d) BCD data
15 12 8 4 0
Sign
BCD 4 dit data (−7999 to 79999)
(e) Double precision BCD data
31 28 24 20 16 12 8 4 0
Sign
Signed BCD 8 bit data (79999999 to 79999999)
(f) Real data
31 28 23 20 16 12 8 4 0
Sign
Exponent
Mantissa
32 bit IEEE format floating-point data

4. ACTUAL APPLICATION

The controller MICREX-F500 developed at this time is a kind of high level language machine which can execute the FCL itself as the machine language. Thus programs of various representations can be available for F500 only by converting them to FCL.

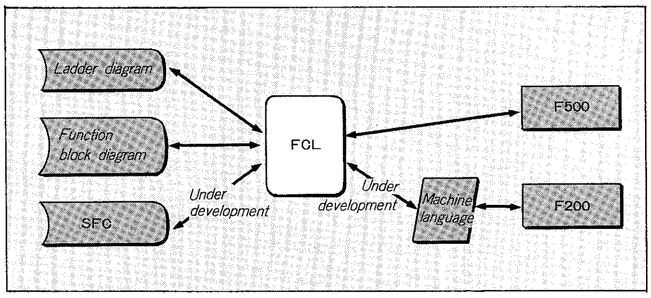
Table 1 Main functions of FCL (1/2)

Class	Name	Symbols and description example	Remarks
Sequence	AND	(& $P_0 P_1 \ldots P_3$)	
	OR	(> = 1 $P_0 P_1 \ldots P_3$)	
	WRITE	(W $P_0 P_1 \ldots P_3$)	$(P_0) \rightarrow P_1 \ldots P_3$
	not	(N P_0)	$\overline{P_0}$: Bit inversion
	Set	(S $P_0 P_1$)	P_0 : Input data P_1 : Coil No.
	Reset	(R $P_0 P_1$)	P_0 : Input data P_1': Coil No.
	Flip-flop	(FF $P_0 P_1 P_2$)	P_0 : F/F No. P_1 : Set input P_2 : Reset input
	Rising edge differential	(D $P_0 P_1$)	P_0 : Input data P_1 : Differential coil No.
	Falling edge differential	(DN $P_0 P_1$)	
Counter	Up counter	(CTU $P_0 P_1 P_2 P_3$)	P_0 : Counter No. P_1 : Counter pulse input P_2 : Reset input P_3 : Set valve
	Down counter	(CTD $P_0 P_1 P_2 P_3$)	
	Ring counter	(CTR $P_0 P_1 P_2 P_3$)	
	Up and down counter	(CTUD $P_0 P_1 P_2 P_3 P_4$)	P_0 : Counter No. P_1 : Up counter pulse P_2 : Down counter pulse P_3 : Reset input P_4 : Set value
Timer	On-delay timer	(TON $P_0 P_1 P_2$)	P_0 : Timer No. P_1 : Input data P_2 : Set value
	Off-delay timer	(TOF $P_0 P_1 P_2$)	
	Monostable timer	(MON $P_0 P_1 P_2$)	
	Retriggerable timer	(MOR $P_0 P_1 P_2$)	
	Integrating timer	(TMR $P_0 P_1 P_2 P_3$)	P_0 : Timer No. P_1 : Input data P_2 : Reset input P : Set value
Comparison	>	(> $P_0 P_1$)	If (P_0 : P_1) is satisfied, the result is 1 (true). If it is not satisfied, the result is 0 (false).
	≧	(> = $P_0 P_1$)	
	<	(< $P_0 P_1$)	
	≦	(<= $P_0 P_1$)	
	=	(= $P_0 P_1$)	
	≠	(< > $P_0 P_1$)	
Logical operation	AND	(AND $P_0 P_1 \ldots P_3$)	
	OR	(OR $P_0 P_1 \ldots P_3$)	
	EOR	(EOR $P_0 P_1 \ldots P_3$)	
	INV	(INV P_0)	$\overline{P_0}$: 1's complement
	Set bit	(SBIT $P_0 P_1$)	P_0 : Bit No. P_1 : Set data
	Reset bit	(RBIT $P_0 P_1$)	P_0 : Bit No. P_1 : Set data
	Test bit	(TBIT $P_0 P_1$)	P_0 : Bit No. P_1 : Tested data
	Shift right logical Shift left logical	(SRL $P_0 P_1$) (SLL $P_0 P_1$)	P_0 : Number of shift bits P_1 : Shifted data
Fixed-point arithmetic	Add	(+ $P_0 P_1 \ldots P_3$)	
	Subtract	(− $P_0 P_1 \ldots P_3$)	
	Multiply	(∗ $P_0 P_1 \ldots P_3$)	
	Divide	(/ $P_0 P_1 \ldots P_3$)	
	Remainder	(REM $P_0 P_1$)	$P_0 \div P_1$ remainder operation.
	Square root	(SQRT P_0)	$\sqrt{P_0}$
	Absolute value	(ABS P_0)	$\lvert P_0 \rvert$
	Sign inversion	(NEG P_0)	$\overline{P_0}$: 2's complement
Floating-point arithmetic	Add	(+F $P_0 P_1 \ldots P_3$)	(32 bit IEEE format)
	Subtract	(−F $P_0 P_1 \ldots P_3$)	
	Multiply	(∗F $P_0 P_1 \ldots P_3$)	
	Divide	(/F $P_0 P_1 \ldots P_3$)	
	Square root	(FSQR P_0)	$\sqrt{P_0}$ (32 bit IEEE format)
	Absolute value	(FABS P_0)	$\lvert P_0 \rvert$ (32 bit IEEE format)
	Sign inversion	(FNEG P_0)	$\overline{P_0}$ (32 bit IEEE format)

Table 1 Main functions of FCL (2/2)

Class	Name	Symbol and description example	Remarks
Conversion	Integer/floating conversion	(IFC P_0)	Fixed-point → floating-point conversion (32 bit IEEE format)
	Floating/integer conversion	(FIC P_0)	Floating-point → fixed-point conversion (32 bit IEEE format)
	BCD/BIN conversion	(DBC P_0)	Decimal → binary conversion
	BIN/BCD conversion	(BDC P_0)	Binary → decimal conversion
	Decode	(DECO P_0)	P_0: Input data
	Encode	(ENCO P_0)	P_0: Input data
Transfer	Assignment	(: = P_0 P_1 ...P_3)	(P_0) → P_1 ...P_3
	Block transfer	(BT P_0 P_1 P_2)	P_0 :Size P_1 : Transfer source address, P_2 : Trasnfer destination address
	Digit transfer	(DT P_0 P_1 P_2 P_3)	P_0 : Input data, P_1 : Transfer source address, P_2 : Bit size, P_3 : Transfer destination address
	Pattern clear	(PC P_0 P_1 P_2)	P_0 : Size, P_1 : Transfer source address, P_2 : Pattern
	Search	(SRCH P_0 P_1 P_2 P_3)	P_0 :Size, P_1 : Search data, P_2 : Test source address, P_3 : Detection address storage area address
	Switch	(SW P_0 P_1 P_2)	P_0 : Switch input, P_1 : On data, P_2 : Off data
	Sort	(SORT P_0 P_1 P_2)	P_0 : Size, P_1 : Test source address, P_2 : Transfer address
	Work swap	(WSWP P_0)	P_0 : Input data
Analog operation	Upper limit	(HL P_0 P_1)	P_0 : Input data, P_1 : Upper limit value
	Lower limit	(LL P_0 P_1)	P_0 : Input data, P_1 : Lower limit value
	Upper and lower limits	(HLL P_0 P_1 P_2)	P_0 : Input data, P_1 : Upper limit value, P_2 : Lower limit value
	Dead band	(DB P_0 P_1)	P_0 : Input data, P_1 : Bias value
	Bias	(DZ P_0 P_1)	P_0 : Input data, P_1 : Bias value
	Filter	(FIL P_0 P_1 P_2 P_3)	P_0 : Filter No., P_1 : Enable input, P_2 : Input data, P_3 : Filter time constant
	Differential	(DIF P_0 P_1 P_2 P_3)	P_0 : Differential/integration No., P_1 : Enable input,
	Integration	(INT P_0 P_1 P_2 P_3)	P_2 : Input data, P_3 : Time constant
Program control	Program start	(PGn expression... expression)	n: Program No.
	FM start	(FMSn expression... expression)	n: Function module No. P_0 P_1 ... P_2 :Input parameters
	FM call	(FMn P_0 P_1 ... P_0)	
	IF statement	(IF condition expression expression 1 expression 2)	Expression 1: THEN processing, expression 2: ELSE processing
	WHILE statement	(WHLE condition expression expression)	
	REPEAT statement	(REPEAT expression con- dition expression	
	CONDITION statement	(COND (condition expression expression) (condition expression expression) ⋮ ⋮ (condition expression expression)	
	CASE statement	(CASE (expression) ((expression expression expression) expression) ((expression expression ...expression) expression) ⋮ ⋮ ((expression expression ...expression) expression) (OTHRW expression)	

Fig. 8 Example of FCL use



By converting FCL to the other machine languages, the these various programs can be executed by other model of PCS and by the controllers of other manufacturers.

As an example, software represented by the three kinds of programming language ladder diagram, function block diagram, and SFC can be run on both the MICREX-F500 series and F200 series, which have a different machine language, as shown in *Fig. 8*.

5. CONCLUSION

The intermediate functional language FCL enables PC software to be described in common way independent of many program representative methods (PC programming languages) or target PC machines. Therefore, PC software can be produced in any programming language, and it can be utilized for any kind of PC machine, even if it is made by other manufacturer. Thus the portability of PC software will be improved, and plenty of PC software will be accumulated as common property, resulting in the progress of process control domain.

We will continue to make every effort hereafter in order to refine the FCL further, and to make software engendering system for FCL more complete and comfortable by using AI technology.