UTSD

Programmable DSP-Based Motion Controller

Reference Manual V3.0

Micro Trend Automation Co., Ltd

3F, No.78, Cheng Kung Road, Sec. 1
Nan Kang, Taipei, Taiwan
TEL:(02)27882162 FAX:(02)27857173 2010.06.04

Contents

UTSD105 Reference Manual

UTSD105 Specifications

- Two serial RS232 communication ports; maximum baud rate 38400 bps.
- On board photo coupling isolated inputs and open collector outputs; I/O expansion available.
- Hardware & software positive and negative limits; home flag inputs.
- Total memory 4M Bits. (can be expanded to 16M Bits)
- 4M Bits flash and battery backup SRAM for double security.
- Two +/- 10v analog output/input.
- ASCII command set.
- 8 timers and 4096 variables.
- Buffer available for 256 motion programs and 16 PLC.
- The most advanced servo control algorithm to effectively minimize the number of errors and increase the product stability.
- System working frequency: 32MHz, zero wait state.
- Servo update rate:1ms.
- The highest output pulse speed: 2M pps.
- Capable of electronic gear and electronic cam applications.
- Hardware capture registers for position searching.
- Definable coordinate system. (independent or dependent)
- S-Curve acceleration, rapid move, linear interpolation, circular interpolation, spline move and blended move.
- "Look Ahead" function.
- Back lash compensation.
- Encoder inputs which allows dual feedback.
- Complete online command set and DNC. (direct numerical control)
- Password protection to make your design secured and private.

Application Field

- Sealing Machine
- Lathe Machine
- Brush Maker
- Bar Feeder
- Fly Cutter / Fly Shear
- Printing Machine
- Gilding Machine
- PC Board Maker
- Packing Machine
- Rotary Table
- Drilling Machine
- Electronic Machine
- Spring Coiling Machine
- Glue Dispensing Machine
- Milling Machine / Engraving Machine
- Laser Cutter
- Wood Cutting Machine
- Stamping Machine
- Grinding Machine
- Press Feeder
- Folding and Gluing Machine
- Steel Cutting Machine
- Winding Machine
- Foam Cutting Machine
- Injection Molding Machine

Mathematics Operation Ability

The DSP provides various mathematics operations with very high execution speed in its instruction set. That allows us to prevent suffering the communication delay by running the on board calculations. Following are the variables and operations that UTC-Series accepted.

Numerical values:

All the numerical values are presented in the 32-bit floating-point format.
Acceptable numerical entry as follows:

1234
3
03 (Started with 0 is acceptable)
-27.656
0.001
.001 (The 0 previous to the decimal point could be omitted)
$ff00 (All the hexadecimal value prompted with '$' sign)

Operators:

- Arithmetic +, –, ∗, /
- Modulo %
- Bit-by-bit Boolean &, |, ^

Functions:

- Trigonometric SIN, COS, TAN
- Inverse Trig. ASIN, ACOS, ATAN, ATAN2
- Logarithmic LN, EXP
- Others SQRT, FABS, INT, ROUND

SIN	Function	Standard trigonometric sine function
	Syntax	SIN({expression})
	Domain	All real numbers
	Domain units	Degrees
	Range	-1.0 – 1.0
	Range units	None
	Possible errors	None
COS	Function	Standard trigonometric cosine function
	Syntax	COS({expression})
	Domain	All real numbers
	Domain units	Degrees
	Range	-1.0 – 1.0
	Range units	None
	Possible errors	None
TAN	Function	Standard trigonometric tangent function
	Syntax	TAN({expression})
	Domain	All real numbers except ± 90, 270, …
	Domain units	Degrees
	Range	All real numbers

	Range units	None
	Possible errors	Divide by zero on illegal domain
ASIN	Function	Inverse sine (arc-sine) function
	Syntax	ASIN({expression})
	Domain	-1.0 –1.0
	Domain units	None
	Range	-90 – 90
	Range units	Degrees
	Possible errors	Illegal domain
ACOS	Function	Inverse cosine (arc-cosine) function
	Syntax	ACOS({expression})
	Domain	-1.0 –1.0
	Domain units	None
	Range	0 – 180
	Range units	Degrees
	Possible errors	Illegal domain
ATAN	Function	Inverse tangent (arc-tangent) function
	Syntax	ATAN({expression})
	Domain	All real numbers
	Domain units	None
	Range	-90 – 90
	Range units	Degrees
	Possible errors	None
ATAN2	Function	Expanded arc-tangent function, the cosine value is stored in Q0, and the sine value in parenthesis. It is distinguished from the standard **ATAN** function by the use of two arguments. The advantage of this function is that it has a full 360 degree range.
	Syntax	ATAN2({expression})
	Domain	All real numbers
	Domain units	None
	Range	-180 – 180
	Range units	Degrees
	Possible errors	None
LN	Function	Natural logarithm function (base e)
	Syntax	LN({expression})
	Domain	All positive real numbers
	Domain units	None
	Range	All real numbers
	Range units	None
	Possible errors	Illegal domain
EXP	Function	Exponentiation function (e^x) Note: To implement the y^x function, use $e^{x \ln(y)}$ instead. A sample expression would be EXP(P2*LN(P1)) to implement the function $P1^{P2}$
	Syntax	EXP({expression})

	Domain	All real numbers
	Domain units	None
	Range	All positive real numbers
	Range units	None
	Possible errors	None
SQRT	Function	Square root function
	Syntax	SQRT({expression})
	Domain	All non-negative real numbers
	Domain units	None
	Range	All non-negative real numbers
	Range units	None
	Possible errors	Illegal domain
FABS	Function	Absolute value function
	Syntax	FABS({expression})
	Domain	All real numbers
	Domain units	None
	Range	All non-negative real numbers
	Range units	None
	Possible errors	None
INT	Function	Truncation function, which returns the greatest integer less than or equal to the argument. (INT(3.6) = 3, INT(-3.2) = -4)
	Syntax	INT({expression})
	Domain	All real numbers
	Domain units	None
	Range	All integer numbers
	Range units	None
	Possible errors	None
ROUND	Function	Round off function, which returns the nearest integer to the argument. (ROUND(3.6) = 4, ROUND(-3.2) = -3)
	Syntax	ROUND({expression})
	Domain	All real numbers
	Domain units	None
	Range	All integer numbers
	Range units	None
	Possible errors	None

Expressions: An operation command string that consists of constants, variables, functions and operators. Such as:

```
512
P1
P1-Q18
1000*COS(Q25*180/3.14159)
I101*FABS(M347)/ATAN(P(Q3+1)/6.28)+5
```

Data: The Data could be a constant without bracketing or an expression parenthesized. Such as:

X100
X(P1+250*P2)
X(100) (Also acceptable)

Comparators: Used for the comparison of two numerical values or expressions. Such as:

=	(Equal To)
!=	(Not Equal To)
<	(Less Than)
<= or !>	(Equal To or Less Than)
>	(Greater Than)
>= or !<	(Equal To or Greater Than)

Conditions :

- Simple Conditions - : Consist of 2 expressions and one comparator. Such as:
  WHILE (1 < 2)
  IF (P1 > 5000)
  WHILE (SIN(P2-P1) <= P300/1000)
- Complex Conditions - Two or more conditions linked with AND, OR, Such as :
  IF (P1 > -20 AND P1 < 20)
  WHILE (P80 = 0 OR I102 > 300 AND I102 < 500)
  IF (Q16 !< Q17 AND Q16 !> Q18 OR M136 < 256 AND M137 < 256)

Timers : M0 : increase 1 per millisecond.

M71..M78 : decrease 1 per millisecond.

Variables:

There are 4 kinds of variables in UTC-Series. The characteristic of each variable will be discussed below. The value of each variable could be modified by an online command or by a command in a motion program.

{variable} = **{data}**

Such as:

I[constant]..[constant]=** ;Set I variable to default value
I102 = 45
I101 = I102+P25*3
P200 = SQRT(Q200/10)
Q400..500 = 0

- I-Variables I0 – I499, Each I-variable has specified definition.
  Initialization and set-up:

I0 – I50:	General card setup
I101 – I135:	Motor #1 setup
I150 – I170:	Coordinate System #1 setup
I201 – I235:	Motor #2 setup
I250 – I270:	Coordinate System #2 setup
I301 – I335:	Motor #3 setup
I350 – I370:	Coordinate System #3 setup
I401 – I435:	Motor #4 setup
I450 – I470:	Coordinate System #4 setup

- P-Variables P0 - P1023
  Global user variable for all coordination.
  32-bits floating-point data format.

Matrix Read: Using **P({expression})** instead of **P{constant}** could program a matrix reading process.

Example: The positioning data is pre-stored in P101 to P200. The sample program is to setup to read and execute those positioning data.

```
P1 = 101
WHILE (P1 < 201)
      X(P(P1))
      DWELL100
      P1 = P1 + 1
ENDW
```

Matrix Write: Matrix write should follow a procedure described below.

First set a M-variable point to P0, such as M80->L:1000, (See M-variable section) then set another M-variable point to the lowest 12 bits of the previous M-variable definition word. Such as M80->C50,0,12 ($C50 is the address of M80's definition word). After above setting, M80 will point to P(M81) address when we set a value to M81. That means change value of M80 will make the same change in P(M81). The following example shows how to setup a sine table in P0 to P359 with previous M variable settings.

```
P1000 = 0
WHILE (P1000 < 360)
      M81 = P1000
      M80 = SIN(P1000)
      P1000 = P1000 + 1
ENDW
```

Q-Variables Q0 - Q1023
Local general variable for each coordinate.
Program parameter transfer variable
32-bit floating-point data format.

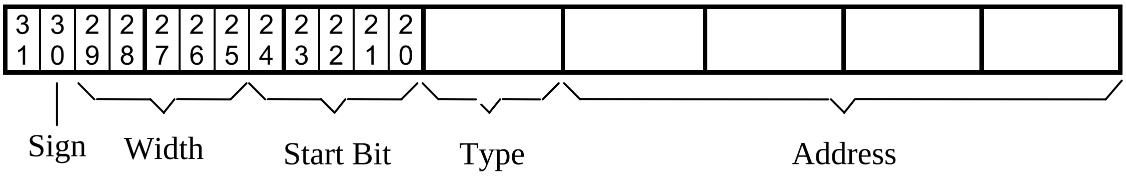
The matrix reading and writing method are the same as P-variables. Following are the actual address list for the Q variable in different coordinate systems. Please beware not to overlap address in multi-coordinate system.(For 2 coordinate systems, only Q0 to Q511 could be used. For 4 coordinate systems, only Q0 to Q 255 could be used)

Mem. Loc.	**Coord. Sys. 1**	**Coord. Sys. 2**	**Coord. Sys. 3**	**Coord. Sys. 4**	**Coord. Sys. 5**	**Coord. Sys. 6**	**Coord. Sys. 7**	**Coord. Sys. 8**
$1400	0	512	768	256	896	384	640	128
…	…	…	…	…	…	…	…	…
$147F	127	639	895	383	1023	511	767	255
$1480	128	640	896	384	0	512	768	256
…	…	…	…	…	…	…	…	…
$14FF	255	767	1023	511	127	639	895	383
$1500	256	768	0	512	128	640	896	384

...	…	…	…	…	…	…	…	…
$157F	383	895	127	639	255	767	1023	511
$1580	384	896	128	640	256	768	0	512
...	…	…	…	…	…	…	…	…
$15FF	511	1023	255	767	383	895	127	639
$1600	512	0	256	768	384	896	128	640
...	…	…	…	…	…	…	…	…
$167F	639	127	383	895	511	1023	255	767
$1680	640	128	384	896	512	0	256	768
...	…	…	…	…	…	…	…	…
$16FF	767	255	511	1023	639	127	383	895
$1700	768	256	512	0	640	128	384	896
...	…	…	…	…	…	…	…	…
$177F	895	383	639	127	767	255	511	1023
$1780	896	384	640	128	768	256	512	0
...	…	…	…	…	…	…	…	…
$17FF	1023	511	767	255	895	383	639	127

- M-Variables M0 - M1023
  Used as a pointer point to the memory address or I/O address:
  Inputs, outputs, counters, A/D, D/A, RAM.

FORMAT:



Address- The address which M variable point to, Range: 0000-FFFF
Type- Specify the memory type.
0: Not used as a pointer, used as a normal variable.
1: Point to (Data Segment), DP = 00
2: Point to I/O address, DP = FF
Start Bit- Point to the start bit of the pointed address, Range: 0-31
Width- Specify the bit width of the pointed address, Range: 0-31
Sign- 1: The content in the pointed address is a signed data.
0: The content in the pointed address is an unsigned data.
Bit 31- 0: The content in the pointed address is an integer data.
1: The content in the pointed address is an floating point data.

Command Specification:

Mxx->*	Not point to any address, used as a normal variable with signed value.
Mxx->addr[, start][, width][,s]	Point to normal data area with integer format.
Mxx->L:addr	Point to normal data area, floating point format.
Mxx->I: addr[, start][, width][,s]	Point to I/O address.

Mxx->	Read current Mxx definition.
Mxx	Read current Mxx value
Mxx = value	Set Mxx value

JOG THE MOTOR

The <Jog> function could be done under the following conditions:

1. No limit switch signals are active on the assigned axis. (The limit signals could be disabled or change polarity by the setting of **Ix09**)
2. No driver error signals are active.
3. The controller is not executing a motion program. (If a motion program is under execution, please stop the motions by an **S** command before jogging the motor).

Addressing Motors:

The active axis should be assigned before the jog function execution. The axis assignment could be done by the instructions #**{const}.** (For example: #1 or #2…). The assignment of axes is modal instructions.

Drivers Enable:

The default condition for all the axes are driver enabled. (Variable **Ix09** could set the enable signals polarity.) The enable conditions will be checked before each motion program execution. An error information will be generated if any addressed motor is disabled.

K: Disable the addressed driver.

J/: Enable the addressed driver.

Jog Acceleration and Deceleration:

The variable **Ix01** could set the jog acceleration and deceleration time. Please do not set this variables to 0.

Jog Speed:

The variable **Ix03** will determine the jog speed. The unit is **counts/msec.** A negative setting will change the direction of motor rotation. To prevent the confusion, we recommend using **Ix09** for direction setting instead and setting **Ix03** to a positive value.

Jog Commands:

J+	Jog to positive direction.
J-	Jog to negative direction.
J/	Jog motion stop.
J={const}	Absolute position jog. Jog to {const} defined position (unit: counts)
J:{const}	Incremental jog. Jog distance = {const} (unit: counts)
J=	Jog to variable defined position. (Suggested Mx63 as the variable pointer)
J:	Variable defined incremental jog. (Suggested Mx63 as the variable pointer)
J*	Jog back to last programming stop position.

Home Searching:

The homing acceleration and deceleration are the same setting as Jog function. (**Ix01**), The speed setting is **Ix07.** The positive value of **Ix07** will run a positive direction home searching while negative settings make negative direction searching. The home searching functions are as following steps.

1. The **HM** Command received.
2. Start searching per **Ix07** setting speed and direction.
3. Detect the home flag trigger signals during searching. (Home flag trigger signals are defined per **Ix09** settings) Decelerate to stop per **Ix01** settings and record the triggered position and trigger signals status.
4. Back to the trigger point per **Ix07** setting speed.
5. Check the home flag status. If the status matches the pre-record condition, jumps to step7. If the status unmatched, generate <Home Flag Error> (Memory address 000D bit 24~bit 27). Then run step 6.
6. Keep forward to search the trigger condition by low speed (about 2k pps) until next trigger happens.
7. Move to the destination per **Ix07** speed setting and **Ix08** offset value settings.
8. Set the current position as mechanical home position and set <Home Function Done> flag. (Memory location 0022)

Motion Program

The powerful UTC-Series instruction set is a kind of high level language. It is similar to BASIC or C language. It also accepts the G, M Code as the motion instructions. The general arithmetic and logical instructions are similar to most of the computer language. Such as WHILE loop, IF… ELSE and other Jump or sequence control instructions.

The UTC-Series allows total 256 programs in the memory. Program 0 is a Rotary Buffer, which is used for DNC feature. Program 1000 is used for defining G CODE. (Please refer to Motion instruction format and STDGCODE.UTC.). Program 1001 is used for M-CODE definition, program 1002 is for defining T-Code. The normal executed programs will be defined in program numbered between 001 to 999.

Each program will start with the instruction **OPEN PROG # (#:** Program number**),** and ended with **CLOSE.** The statement after **';'** mark will not be interpreted could be used as a remark. The total program structure will be as the following:

```
OPEN PROG 1        ; Open program buffer #1
CLEAR              ; Clear previous program
LIN                ; Linear movement
F2.36              ; Speed setting as 2.36 (user unit) / min
X5.346Y0           ; The first movement for a square loop.
X5.346Y5.346       ; The 2nd movement for a square loop.
X0Y5.346           ; The 3rd movement for a square loop.
X0Y0               ; The 4th movement for a square loop.
CLOSE              ; Close the program buffer #1

OPEN PROG 1000 CLEAR ; Defining a G-Code set
RPD                ; G00 – Immediate Movement
RET
N1000 LIN          ; G01 – Linear interpolation.
RET
N2000 CIR1         ; G02 – Circular interpolation with CW.
RET
N3000 CIR2         ; G03 –Circular interpolation with CCW.
RET
…
N90000 ABS         ; G90 – Move to Absolute Position.
RET
N90000 INC         ; G91 – Make an incremental move
RET
N92000             ; G92 – Position define
READ(X,Y,Z)        ; Position read
IF (Q100 & 8388608 > 0)       ; Check if X variable read complete.
      PSET X(Q124)            ; If yes, set X value as Q124.
ENDIF
```

```
IF (Q100 & 16777216 > 0)          ; Check if Y variable read complete.
       PSET Y(Q125)               ; If yes, set Y value as Q125.
ENDIF
IF (Q100 & 33554432 > 0)          ; Check if Z variable read complete.
      PSET Z(Q126)                ; If yes, set Z value as Q126.
ENDIF
RET
```

We may use any simple editor software to edit the entire program and then download to UTC-Series for execution. (The motion instruction format will be described in later section.)

We should define the Coordinate System before we run a NC program. The Coordinate System is a motor group, all the motor inside this group could operate according to the motion instruction of the program at the same time or sequentially. It is possible to define the motors into different Coordination System if their motion is independent and no sequential relation. There are maximum 4 Coordinate Systems in UTC-Series controller. Each motor should not appear in more than one Coordinate Systems.

The instruction **&n** is used to define the Coordinate System **n**. Multi-programs with multi-coordinate systems could be executed simultaneously.

As an example, the first of above program uses X and Y axes in the program. We define 1st motor for X-axis And 2nd motor for Y-axis. Assume the user's unit is mm, encoder feedback is 4000 counts/rev. Ball screw pitch is 5 mm/rev. Then we get ratio of counts/user's unit as:

$$\frac{4000\dfrac{count}{rev}}{5\dfrac{mm}{rev}} = 800\frac{count}{mm}$$

Assume we have only one Coordinate System. We should define the system as:

&1	; Define the coordinate system 1
#1->800X	; 800counts / user's unit (mm) Define the motor to an axis and ratio.
#2->800Y	; 800counts / user's unit (mm) Define the motor to an axis and ratio.

The axis name could be any of the **X,Y,Z,U,V,W,A,B or C.** The ratio could be any positive floating point value. The motor direction is defined per **Ix09** setting. We can not <**Jog**> any motor with axis defined in a program under execution. We should first cancel the definition (**ex: #4->0**) before jogging the motor. Or we can jog a motor that is defined in other axis than the executing one.

There are some instructions to do with auto-execution after we defined the axes:

&1	; Select the Coordinate System one.
B1	; Select the Program 1
R	; Continuous execution.
S	; Single step execution

H	; Halt the execution
A	; Abort the execution
R*	; Continuous execution for all axes.
S*	; Single step execution for all axes.
<Ctrl-O>	; Halt the execution for all axes.
A*	; Abort the execution for all axes.

PLC Program

All the **motion programs** are executed synchronously step by step at highest priority. UTC-Series runs up to 16 PLC programs by scanning asynchronously. Those PLC programs accomplish all the normal PLC features. To create a PLC program and to write an instruction are similar to writing a motion program. We use **OPEN PLC#** to create a new PLC program instead of **OPEN PROG#** for creating a motion program. We can not perform motion instructions in PLC program. We can only move a motor by the <Jog> command.

The PLC program scan frequency is high enough to trace all the logic input signals, set output signal level, set out messages, motor the motions, change parameters or to start another program.

PLC also send out commands just like we send commands from computers. The scan time for normal PLC is about 5 to 10 msec depends on the program length. UTC-Series allows 16 PLC programs stored in the memory. There are numbered from 0 to 15. The stored PLC programs could be enabled by command **ENAPLC#** and disabled by **DISPLC#** (# is the program number). Variable **I6** will determine which programs could be enabled.

The PLC features are described as following:

Calculation Statements:

We may enable a PLC program to calculate an arithmetic operation repeatedly until the expected result appears. EX:

```
P1 = P1+1
P161 = (M161+M164)/M191
```

Conditional Statements:

The conditional statements in PLC are the same as that in motion programs. The follows are some common used statements.

- **IF** (level-triggered conditions)

  To execute a certain block each time when the specified condition are met.

```
IF (M11 = 0)            ; E-stop signal
    CMD"A"
ENDIF
IF (M12 = 0)            ; If a pushbutton is pressed, P1 will count up.
    P1 = P1+1           ; Count frequency is the PLC scan frequency.
ENDIF
```

- **IF** (edge-triggered conditions)

  To execute a certain block once when the specified condition are first met.

```
IF (M11 = 0 AND P11 != 0)       ; Push the JOG button.
    P11 = 0
    CMD"#1J+"
ENDIF
IF (M11 = 1 AND P11 != 1)       ; Release the JOG button.
    P11 = 1
    CMD"#1J/"
ENDIF
IF (M12 = 0)            ; P1 count up when pushbutton pressed once.
```

```
            IF (P12 = 1)
                P1 = P1+1
                P12 = 0
            ENDIF
        ELSE
            P12 = 1
        ENDIF
```

- **WHILE Loop**

  To execute a certain block while a specified condition is met. UTC-Series will jump out this program if an ENDWHILE instruction is executed. Next time the scan will start at the top of this WHILE loop.

Note: The statements located after WHILE loop will not be executed if the WHILE loop condition is met and that certain block in under execution.

```
        IF (M11 = 0 AND P11 != 0)          ; Press E-stop pushbutton
            P11 = 0
            CMD"A"
            WHILE(M60 != 0)                ; Wait for IN POSITION signal
            ENDWHILE
            M1 = 0                         ; Motor brake ON
        ENDIF
        IF (M11 = 1 AND P11 != 1)
            P11 = 1
        ENDIF
```

- **Precise Timer**

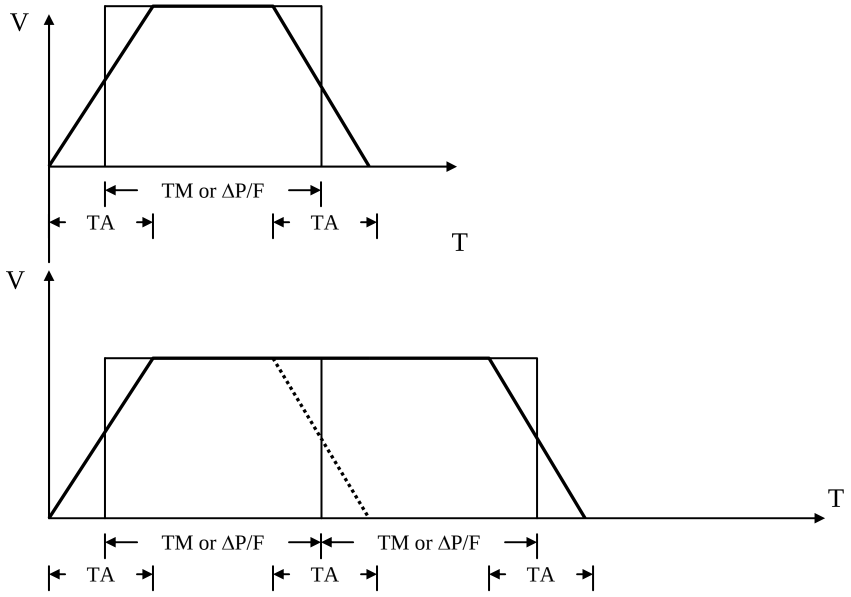
  We can get an accurate timing control using timer of the controller and a WHILE loop. EX:

```
        M71 = 1000                         ; Set timer to 1000 msec
        WHILE(M71 > 0)                     ; Loop until counts to zero
        ENDWHILE
        Where M71->532,S                   ; TIMER1
        M81 = M0                           ; Start of timer
        M82 = M0-M81                       ; Time elapsed so far
        WHILE(M82 < 1000)                  ; Less than specified time ? (in msec)
            M82 = M0-M81                   ; Time elapsed so far
        ENDWHILE
        Where   M0->536,S                  ; Interrupt counter
            M81->C6C0,S                    ; User buffer, signed integer format
            M82->C6C1,S                    ; User buffer, signed integer format
```

Trajectory Generation

UTC-Series has very powerful trajectory generation algorithms. Users can perform variety of difficult maneuvers with simple programs.

The easiest class of moves is Blended Linear Move, without stopping between two linear moves. We only have to set the motion feedrate (F) or move time (TM), acceleration time (TA), S-curve acceleration time (Ts=TA*TS/2), then the linear acceleration time TL = TA-2*Ts. The actual total acceleration will be as TA.



If we set S-curve acceleration time TS = 0(Ts=TA*0=0), we will obtain a constant slope acceleration. The motion profiles will be as Fig 1 on next page. If TS=100, the motion profiles will become a pure S curve as Fig 2 on next page. If 0 > TS>100, the profiles will be as Fig 3 on next page.

Since UTC-Series is very flexible for the acceleration and S-curve acceleration setting. The users can get the maximum acceleration with extremely low jerk. The UTC-Series will handle the multi-axes velocity and acceleration combination calculation. Users do not have to worry about this.

If more than one move is specified in succession without pause in between, the first move will blend into the second with same type of controlled acceleration as is done to and from a stop. We will obtain a smooth curve by calculating the velocities at the beginning of deceleration of first move.

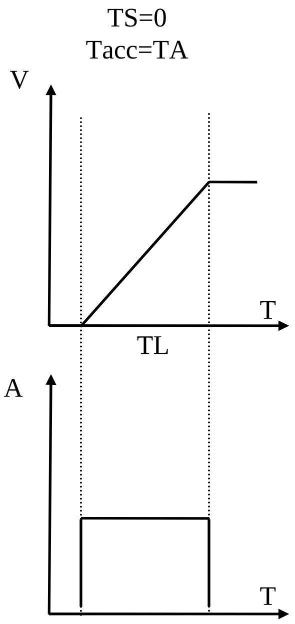


Fig. 1

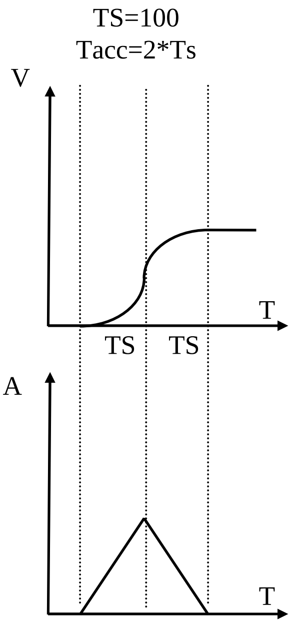


Fig. 2

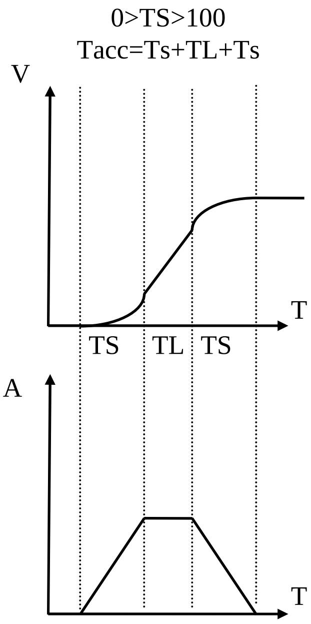
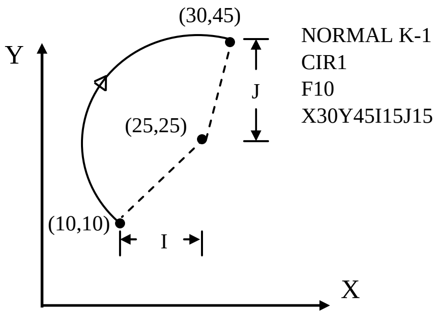
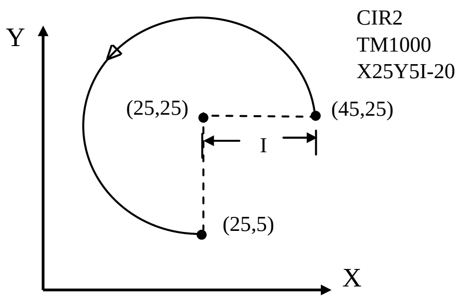


Fig. 3

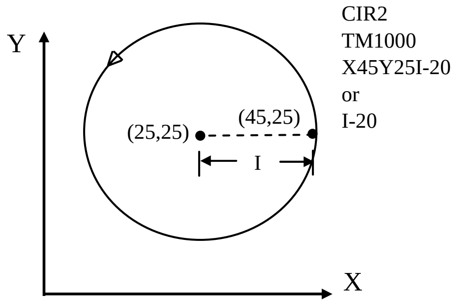
In circular mode, UTC-Series will automatically generate 2-dimentional or 3-dimentional circular moves. The acceleration and velocity settings are the same as linear move. Besides assigning the destination position (endpoint), we have to specify either the arc center or the radius. The circular move could be blended with other circular move or linear motion.



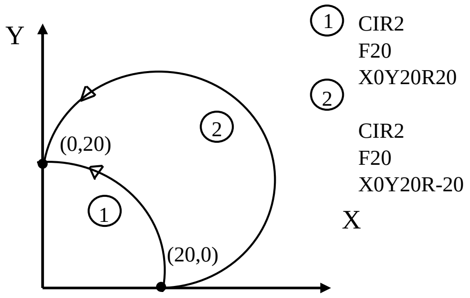
```
NORMAL K-1
CIR1
F10
X30Y45I15J15
```



```
CIR2
TM1000
X25Y5I-20
```

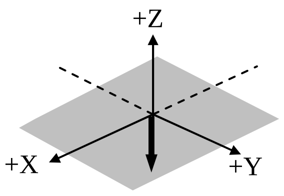


```
CIR2
TM1000
X45Y25I-20
or
I-20
```

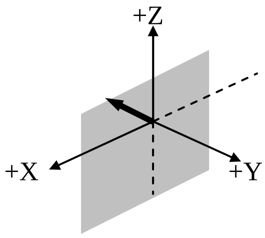


```
1 CIR2
F20
X0Y20R20
2
CIR2
F20
X0Y20R-20
```

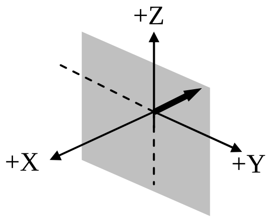
Orientation of the plane



NORMAL K-1



NORMAL J-1



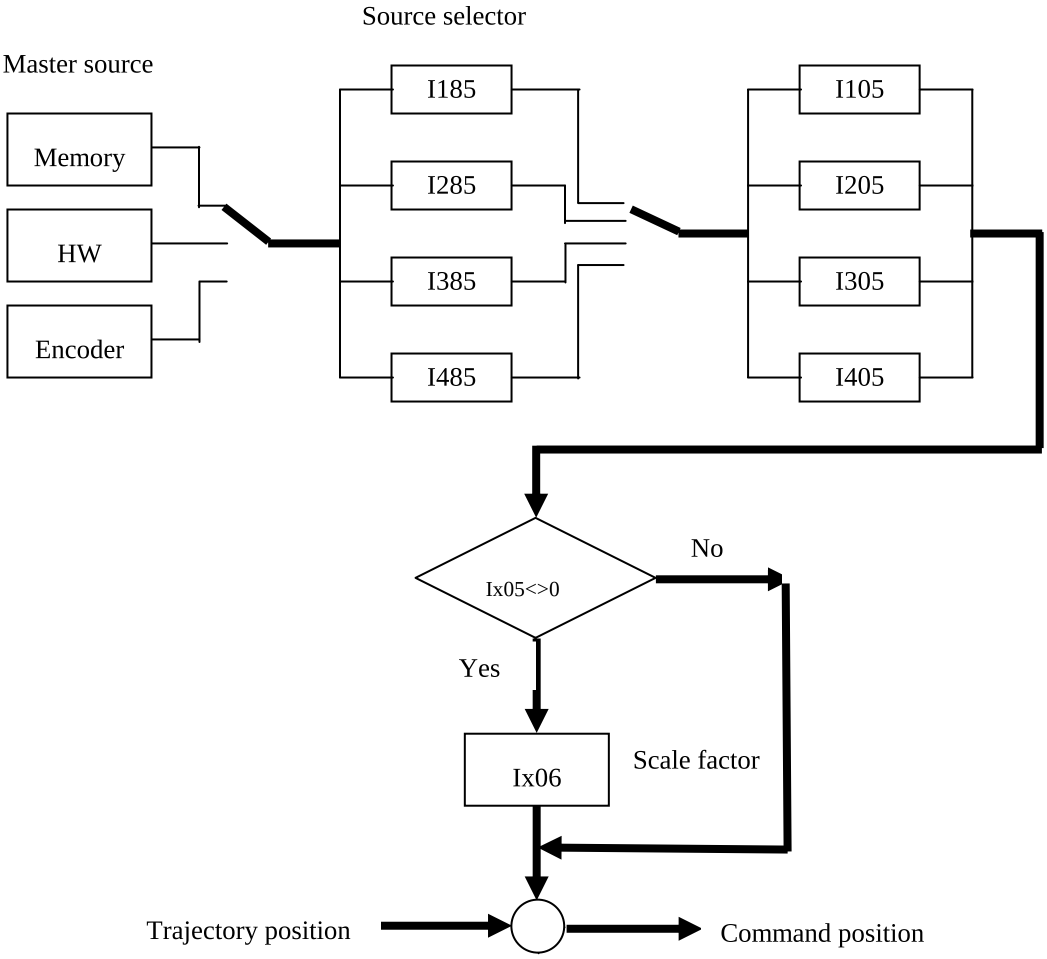
NORMAL I-1

The users might need to generate some very complicated trajectory to perform some difficult work. UTC-Series provide an additional type of motion that allows user to create a smoother and more accurate trajectory. That is called **Spline Move**. In this mode, user has to specify the move time (TM) or feedrate (F) in each segment. Then enter the position or distance step by step. The three dimensional trajectory will be created automatically with no discontinuity of acceleration and velocity.

When the UTC-Series need to follow external signals (normally encoder signals), there are two methods to archive this feature.

- **Position Following – electrical gearing**

When the master signal specified by **Ix85** come in, it would pass by a buffer area specified by **Ix86** (The higher **Ix86** settings the lower change rate of the speed). If **Ix05** set to 1, the **x** motor will follow the master signal from **I185** with the gear ratio of **Ix06** setting. If **Ix05** set to 2, the **x** motor will follow the master signal from **I285** with the gear ratio of **Ix06** setting.

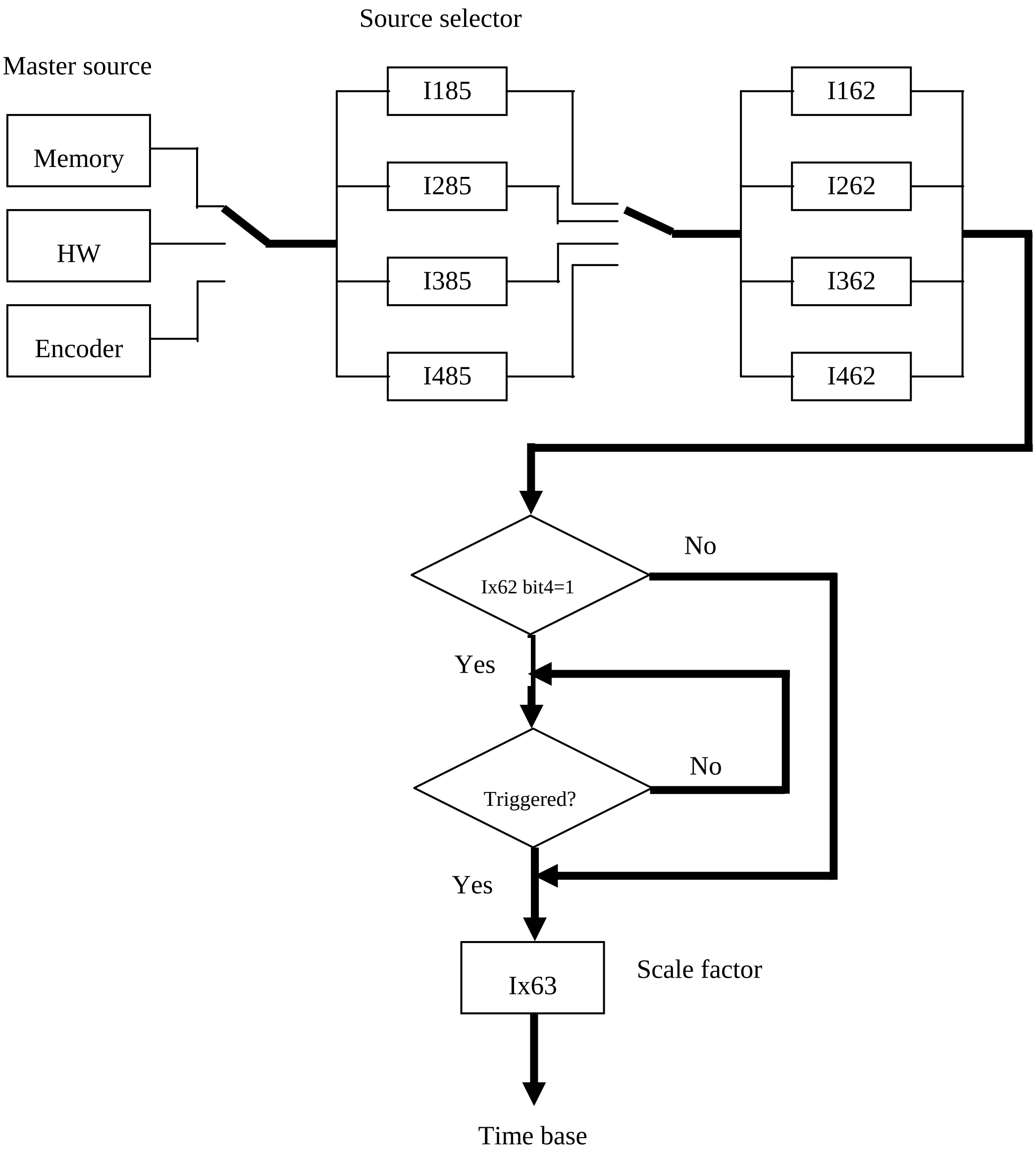


Flow chart of position following (V3.0)

- **Time Based Following – electrical cams**

Besides the position following, there is a special follow method that the master signal will control the time base to archive following purpose. In this mode (**Ix62<>0**), user only have to pre-define a motion profile. The incoming master signal will act as time base. For example, if **Ix63=100,** each count of the master signal will be interpreted as **100** msec. The motion profile will be performed according to the time base of master signal.

If **Ix62** bit4 = 1 also, the following action will start at C signal of 4th axis. Before the trigger signal comes, the control will halt and wait.



Flow chart of time base following (V3.0)

<Example>:

There is a Master encoder with 400 counts/rev, The control should jog 100 mm for each revolution of master encoder. The program will be as:

```
INC                    ; Incremental Mode
      I50=1            ; Blended Move mode
      I185= 5          ; Master from HW Port
      I162=1           ; Start the Time Base Following
      TM(400-(I152+II153)/2) ; Take acceleration time into account.
      X100             ; X-axis move 100 mm first time
      TM400            ; TM=400 ms (Per encoder revolution) after 1st count.
      P0 = 0
      WHILE(P0 = 0)
X100                   ; X-axis move 100 mm each revolution
      ENDW
```

Command Summary

On-line Command (The commands will be executed upon reception)

A. On-line Commands For All Axes

1. Control Type Command:

<Ctrl-A>	Abort all programs execution and motor move.
<Ctrl-D>	Disable all PLC programs.
<Ctrl-G>	Report the status word of this control card.
<Ctrl-K>	Disable all motor driver.
<Ctrl-O>	Hold all the coordination system that is under execution.
<Ctrl-P>	Report the position for all axes(Unit: count)
<Ctrl-Q>	Report the absolute position for all motors. (Unit: count)
<Ctrl-R>	Start to execute all motion programs for all coordinate system.
<Ctrl-S>	Step execution for all motion program in all coordinate system.
<Ctrl-V>	Report velocity of all motors.

2. Addressing Mode Commands

#n	Address the motor (n is a number 1-4)
#	Report the currently addressed motor.
&n	Address a coordinate system(n is a number 1-4)
&	report the currently addressed coordinate system.

3. PLC Control Commands

ENAPLC{const}[,{const}...]	Enable the {const} specified PLC(s)
DISPLC{const}[,{const}...]	Disable the {const} specified PLC(s)

4. Global Variable Commands:

I{data}={expression}	Assign the value {expression} to I{data}
P{data}={expression}	Assign the value {expression} to P{data}
Q{data}={expression}	Assign the value {expression} to Q{data}
M{data}={expression}	Assign the value {expression} to M{data}
M{data}->{definition}	Point the M{data} to the address {definition}
M{data}->*	Assign M{data} as a normal variable.
I{data}	Report I variable value
P{data}	Report P variable value
Q{data}	Report Q variable value
M{data}	Report M variable value
M{data}->	Report M variable definition

5. Buffer Control Commands

OPEN PROG {data}	Open a motion program buffer no. {data}
OPEN ROT	Open the Rotary Buffer
OPEN PLC {data}	Open a specified PLC buffer.
CLOSE	Close the currently opened buffer
CLEAR	Erase the currently opened buffer
SIZE	Report the available unused buffer memory
LIST [{buffer}]	List the contents of specified buffer.

B. Coordinate System On-line Commands

1. Axis Definition Command

#n->{constant}{axis}	Assign the motor n to axis {axis} with {constant} ratio
#n->	Report the axis definition of motor n.

2. Normal Axis Command

%{constant}	Set the Feedrate Override
%	Report Feedrate Override

3. Program Control Command

R	Run the motion program
S	Step execute the motion program
B{constant}	Reset the program pointer
H	Pause the motion programs under execution
A	Abort all the programs under execution.

5. Coordinate Attribute Commands

{axis}={expression}	Re-define the specified axis position
INIT	Axis position shift or rotation

6. Buffer Control Command

PC	Report the current program number
LIST	Report the current program content
PE	Report the program line under execution
LIST PE	Report the program content under execution
DEFROT{constant}	Define a rotary buffer size
DELROT	Delete the rotary buffer
PR	Report the program lines yet to execution

C. Motor On-line Command

1. Normal Command

HM	Motor return to home position
HMZ	Do a zero move homing
O{data}	Manual controlled pulse output

2. Jog Command

J+	Jog to plus direction
J-	Jog to minus direction

J/	Jog stop
J=	Jog to a variable specified position
J={expression}	Jog to {expression} specified position
J:{expression}	Jog an {expression} specified distance.
J:	Jog a variable specified distance
J*	Jog back to last programmed position

3. Report Command

P	Report current motor position
V	Report current motor velocity

D. Program Pointer Control Command

END	Program pointer point to program end
JP{constant}	Program pointer point to specified line no.
NEXT	Program pointer point to next line
UPPER	Program pointer point to last line

Motion Program Command (Stored in the program buffer, executed upon 'R' command)

A. Motion Command

{axis}{data}[{axis}{data}...]	Single step motor linear motion <Example> X100Y100Z200
{axis}{data}[{axis}{data}..][{vector}{data}..]	Single step circular motion
<Example>X100Y100Z200I500J300	
DWELL{data}	Dwell specified time, non time base related
DELAY{data}	Dwell specified time, time base related

B. Modal Command

LIN	Set to linear motion mode
RPD	Set to rapid traverse mode
CIR1	Set to clockwise circular move mode
CIR2	Set to counterclockwise circular move mode
SPLINE	Set to Spline Move mode

C. Coordinate Attribute Command

ABS[({axis}[,{axis},...])]	Absolute move mode
INC[({axis}[,{axis},...])]	Incremental move mode
PSET{axis}{data}[{axis}{data}...]	Redefine the current axis position
NORMAL{vector}{data}	Define normal vector to plane of circular interpolation
ADIS{axis}{data}[{axis}{data}...]	Set axes absolute offset value
IDIS{axis}{data}[{axis}{data}...]	Set axes incremental offset value
AROT X{data}	Set axes absolute rotation angle
IROT X{data}	Set axes incremental rotation angle
ASCL{axis}{data}[{axis}{data}...]	Set axes absolute enlarge ratio

ISCL{axis}{data}[{axis}{data}...]	Set axes incremental enlarge ratio
INIT	Cancel all axes redefinition
R{data}	Set radius for an arc

D. Motion Attribute Command

TM{data}	Set motion time for a single move
F{data}	Set motion feedrate
TA{data}	Set motion acceleration time
TS{data}	Set motion s-curve acceleration time

E. Parameter Setting Command

I{data}={expression}	Assign the {expression} result to I variable
P{data}={expression}	Assign the {expression} result to P variable
Q{data}={expression}	Assign the {expression} result to Q variable
M{data}={expression}	Assign the {expression} result to M variable

F. Program Logic Command

N{constant}	Program line number
GOTO{data}	Jump to a line no. with no return
GOSUB{data}[{letter}{axis}...]	Go to a line & return with variable data
CALL{data}[.{data}][{letter}{axis}...]	Call subroutine [with parameter]
RET	GOSUB return command
READ({letter}[,{letter}...])	Read the parameter for a subroutine
IF({condition}){action}	Execute when the condition met.
ELSE{action}	Execute when the condition not met
ENDIF	Conditional block end
WHILE({condition}){action}	Execute each time when condition met
ENDWHILE	End of WHILE loop
G{data}	NC program G-Code
M{data}	NC program M-Code

G. Other Command

@	Forced On-line command
CMD"{command}"	On-line command from NC program
CMD^{letter}	Control type online command from NC program
SEND"{message}"	Send out messages to PC
DISP[{constant}] ,"{message}"	Shows messages to LCD display
DISP{constant},{constant},{variable}	Shows variable content to LCD display
ENAPLC{constant}[,{constant}...]	Enable specified PLC program
DISPLC{constant}[,{constant}...]	Disable specified PLC program

H. Edit Control Command

INS	Set program line editing as insert mode.
OVR	Set program line editing as replace mode.
DEL	Delete current program pointer content
LEARN[({axis}[,{axis}. . .])]	Motor position teaching

PLC Program Command *(Stored in the buffer executed repeatedly)*

A. Conditional Command

IF({condition})	Execute when condition met
ELSE{action}	Execute when condition not met
ENDIF	End of conditional block
WHILE({condition})	Execute each time when condition met
ENDWHILE	End of WHILE loop
AND({condition})	Condition combinations
OR({condition})	Condition combinations

B. Operation

{variable}={expression}	Assign the {expression} result to a variable
CMD"{command}"	On-line command from PLC
CMD^{letter}	On-line Ctrl-type command from PLC
SEND"{message}"	Send messages to PC
DISP[{constant}],"{message}"	Show messages on LCD
DISP{constant},{constant},{variable}	Show variables on LCD
ENAPLC{constant}[,{constant}...]	Enable specified PLC
DISPLC{constant}[,{constant}...]	Disable specified PLC

I-Parameter Summary (V3.0

Number	Definitions
I0	Card Number
I1	Coordinate System Activation Control
I2	COM2 Baudrate Control
I3	COM2 Handshake Control
I4~ I5	<Reserved>
I6	PLC Programs On/Off Control
I7~ I8	<Reserved>
I9	Maximum Digit for Floating Point Returned
I10	Real Time Interrupt Period
I11~I17	<Reserved>
I18	Extension I/O Board Enable
I19	Digital Inputs Debounce Cycle
I20	Gathered Data Selection
I21	Gathered Data Source 1
I22	Gathered Data Source 2
I23	Gathered Data Source 3
I24	Gathered Data Source 4
I25	Gather Period
I26	Gather Buffer Size
I27	Gather Start and Stop Control
I28	Gather Stop Delay
I50	Current Loop Proportional Gain
I51	Current Loop big step
I52	Current Loop Derivative Gain
I53	Motor Rated Speed
I54	Motor Rated Current
I55	Motor Peak Current
I56	Motor Encoder Feedback Direction
I57	Alarm Checking Mask
I58	Motor Type Selection
I59	Motor Pole Pair Number
I60	Motor Encoder Resolution
I61	Motor Index Position
I62	Motor Magnetization Current
I80	Handwheel Decode Control
I81	<Reserved>
I82	DA1 Bias
I83	DA2 Bias
I84	DA1 Slew Rate
I85	DA2 Slew Rate
I86	DA PWM Frequence
I87~I99	<Reserved>

Ix00	Motor x Activate Control
Ix01	Motor x Jog / Home Acceleration Time
Ix02	Motor Open Loop Command Slew Rate
Ix03	Motor x Jog Speed
Ix04	Motor Deceleration Rate on Position Limit or Abort
Ix05	Motor x Master Following Enable
Ix06	Motor x Master Scale Factor
Ix07	Motor x Homing Speed and Direction
Ix08	Motor x Home Offset
Ix09	Motor x Flag Control
Ix10	Motor x Positive Software Limit
Ix11	Motor x Negative Software Limit
Ix12	Motor x Coordinate Position Displacement
Ix13	Motor x Coordinate Position Scaling
Ix14	<Reserved>
Ix15	Motor x Backlash Size
Ix16	Motor x Backlash Takeup Rate
Ix17	Motor x Rollover Range
Ix18	<Reserved>
Ix19	Motor x Velocity Weighting
Ix20	Motor x PID Proportional Gain
Ix21	Motor x PID Derivative Gain
Ix22	Motor x Velocity Feedforward Gain
Ix23	Motor x PID Integral Gain
Ix24	Motor x PID Integration Mode
Ix25	Motor x Acceleration Feedforward Gain
Ix26	Motor x Position Feedback Address
Ix27	Motor x Velocity Feedback Address
Ix28	Motor x Velocity Feedback Scale
Ix29	Motor x DAC Bias
Ix30	Motor x DAC Limit
Ix31	Motor x Fatal Following Error
Ix32	Motor x Dead Band Size
Ix33	Motor x In Position Band
Ix34	Motor x Big Step Size
Ix35	Motor x Integration Limit
Ix50	Coordinate System x Blended Move Enable Control
Ix51	Coordinate System x Default Program Number
Ix52	Coordinate System x Default Program Acceleration Time
Ix53	Coordinate System x Default Program S-Curve Time
Ix54	Coordinate System x Default Program Feed Rate
Ix55	Coordinate System x Program Acceleration Mode
Ix56	Coordinate System x Feed Rate Override Slew Rate
Ix57	Coordinate System x Program Rapid Move Acceleration Time
Ix58	Coordinate System x Program Rapid Move Feed Rate
Ix59	Coordinate System x Rapid Mode
Ix60	Coordinate System x Maximum Permitted Program Acceleration
Ix61	Coordinate System x Rotate Angle

Ix62	Coordinate System x Master Source
Ix63	Coordinate System x External Time-Base Scale
Ix64	Coordinate System x External Time-Base Offset
Ix65	Coordinate System x Cam Table Rollover
Ix66	Coordinate System x Cam Table Output Mask
Ix80	Encoder Decode Control
Ix81	Encoder Capture Control
Ix82	Encoder Capture Flag Select
Ix85	Master x Source Address
Ix86	Master x Moving Average Buffer Size
Ix87	Master x Backlash Hysteresis

Suggested M-Variable Definition (V3.0)

```
M0->536,S                  ; INTERRUPT COUNTER
;
; GENERAL PURPOSE INPUTS AND OUTPUTS
;
M1->14,16,1                ; MACHINE OUTPUT 1 (CN4-31)
M2->14,17,1                ; MACHINE OUTPUT 2 (CN4-29)
M3->14,18,1                ; MACHINE OUTPUT 3 (CN4-27)
M4->14,19,1                ; MACHINE OUTPUT 4 (CN4-25)
M5->14,20,1                ; MACHINE OUTPUT 5 (CN4-23)
M6->14,21,1                ; MACHINE OUTPUT 6 (CN4-21)
M7->14,22,1                ; MACHINE OUTPUT 7 (CN4-19)
M8->14,23,1                ; MACHINE OUTPUT 8 (CN4-17)
M9->14,16,8                ; MACHINE OUTPUT 1-8 TREATED AS BYTE

M11->I:FF41,16,1           ; MACHINE INPUT 1 (CN4-15)
M12->I:FF41,17,1           ; MACHINE INPUT 2 (CN4-13)
M13->I:FF41,18,1           ; MACHINE INPUT 3 (CN4-11)
M14->I:FF41,19,1           ; MACHINE INPUT 4 (CN4-9)
M15->I:FF41,20,1           ; MACHINE INPUT 5 (CN4-7)
M16->I:FF41,21,1           ; MACHINE INPUT 6 (CN4-5)
M17->I:FF41,22,1           ; MACHINE INPUT 7 (CN4-3)
M18->I:FF41,23,1           ; MACHINE INPUT 8 (CN4-1)
M19->I:FF41,16,8           ; MACHINE INPUT 1-8 TREATED AS BYTE

M20->I:FF41,0,1            ; MACHINE INPUT 9 (CN3-4)
M21->I:FF41,1,1            ; MACHINE INPUT 10 (CN3-6)
M22->I:FF41,2,1            ; MACHINE INPUT 11 (CN3-7)
M23->I:FF41,3,1            ; MACHINE INPUT 12 (CN3-8)
M24->I:FF41,4,1            ; MACHINE INPUT 13 (CN3-9)
M25->I:FF41,5,1            ; MACHINE INPUT 14 (CN3-10)
M26->I:FF41,6,1            ; MACHINE INPUT 15 (CN3-11)
M27->I:FF41,7,1            ; MACHINE INPUT 16 (CN3-12)
M28->I:FF42,0,1            ; MACHINE INPUT 17 (CN3-3)
M29->I:FF42,1,1            ; MACHINE INPUT 18 (CN3-5)
M30->I:FF42,2,1            ; MACHINE INPUT 19 (CN3-13)
M31->I:FF42,3,1            ; MACHINE INPUT 20 (CN3-14)
M32->I:FF42,4,1            ; MACHINE INPUT 21 (RESERVED)
M33->I:FF42,5,1            ; MACHINE INPUT 22 (RESERVED)
M34->I:FF42,6,1            ; MACHINE INPUT 23 (RESERVED)
M35->I:FF42,7,1            ; MACHINE INPUT 24 (RESERVED)
M36->I:FF41,0,8            ; MACHINE INPUT 9-16 TREATED AS BYTE
M37->I:FF42,0,8            ; MACHINE INPUT 17-24 TREATED AS BYTE
;
M40->14,8,1                ; MACHINE OUTPUT 9 (CN2-4)
M41->14,9,1                ; MACHINE OUTPUT 10 (CN2-6)
M42->14,10,1               ; MACHINE OUTPUT 11 (CN2-8)
M43->14,11,1               ; MACHINE OUTPUT 12 (CN2-10)
```

```
M44->14,12,1                 ; MACHINE OUTPUT 13 (CN2-12)
M45->14,13,1                 ; MACHINE OUTPUT 14 (CN2-14)
M46->14,14,1                 ; MACHINE OUTPUT 15 (CN2-16)
M47->14,15,1                 ; MACHINE OUTPUT 16 (CN2-18)
M48->14,8,8                  ; MACHINE OUTPUT 9-16 TREATED AS BYTE
;
M50->I:FF41,8,1              ; MACHINE INPUT 25 (CN2-3)
M51->I:FF41,9,1              ; MACHINE INPUT 26 (CN2-5)
M52->I:FF41,10,1             ; MACHINE INPUT 27 (CN2-7)
M53->I:FF41,11,1             ; MACHINE INPUT 28 (CN2-9)
M54->I:FF41,12,1             ; MACHINE INPUT 29 (CN2-11)
M55->I:FF41,13,1             ; MACHINE INPUT 30 (CN2-13)
M56->I:FF41,14,1             ; MACHINE INPUT 31 (CN2-15)
M57->I:FF41,15,1             ; MACHINE INPUT 32 (CN2-17)
M58->I:FF41,8,8              ; MACHINE INPUT 25-32 TREATED AS BYTE
;
M60->14,0,1                 ; RESERVED MACHINE OUTPUT (CN3-17, IN POS.)
M61->14,1,1                 ; RESERVED MACHINE OUTPUT (CN3-18, BUF. REQ)
M62->14,2,1                 ; RESERVED MACHINE OUTPUT (CN3-19, FAT. FE)
M63->14,3,1                 ; RESERVED MACHINE OUTPUT (CN3-23, WAR. FE)
M64->14,4,1                 ; MACHINE OUTPUT 17 (CN2-19)
M65->14,5,1                 ; MACHINE OUTPUT 18 (CN2-21)
M66->14,6,1                 ; MACHINE OUTPUT 19 (CN2-23)
M67->14,7,1                 ; MACHINE OUTPUT 20 (CN2-26)
M68->14,0,8                 ; MACHINE OUTPUT 17-24 TREATED AS BYTE
;
M70->I:FF53,0,16            ; CONTROL WORD FOR 1ST MT-0123
M71->532,S                  ; TIMER1 COUNT
M72->537,S                  ; TIMER2 COUNT
M73->87,S                   ; MASTER INPUT VALUE (COUNTS)
M74->88,S                   ; MASTER INPUT VELOCITY (COUNTS/MSEC)
;
;Registers associated with axis1
M101->I:FF31,0,12,S         ; #1 12-BIT UPDOWN COUNTER (COUNTS)
M102->AA,S                  ; #1 COMMAND VELOCITY (UNIT: 125 PPS)
M121->I:FF42,12,1           ; -LIM1 INPUT STATUS
M122->I:FF42,8,1            ; +LIM1 INPUT STATUS
M123->I:FF42,20,1           ; FAULT1 INPUT STATUS
M124->I:FF37,1,1            ; HMFL1 INPUT STATUS
M131->D,0,1                 ; #1 POSITIVE LIMIT SET
M132->D,1,1                 ; #1 NEGATIVE LIMIT SET
M139->D,7,1                 ; #1 DRIVER ENABLE BIT
M140->D,3,1                 ; #1 IN-POSITION BIT
M141->22,5,1                ; #1 JOGTO IN PROGRESS
M142->78,S                  ; #1 HOME IN PROGRESS
M143->D,2,1                 ; #1 DRIVER FAULT SET
M145->22,2,1                ; #1 HOME COMPLETE
M146->D,24,1                ; #1 HOME FLAG ERROR
M147->3,S                   ; #1 INC MODE
M161->AF,S                  ; #1 COMMAND POSITION (COUNTS)
M162->56D,S                 ; #1 ACTUAL POSITION (COUNTS)
```

```
M163->L:57F              ; #1 JOG REGISTER POSITION (COUNTS)
M164->L:E4               ; #1 POSITION BIAS (COUNTS)
M165->L:DC               ; #1 PROG. TARGET POSITION (USER UNITS)
M166->L:E0               ; #1 PROG. TARGET POSITION (COUNTS)
M167->L:5A0              ; #1 ACTUAL VELOCITY (COUNTS/MSEC)
M191->L:C6               ; #1 SCALE
;
;Registers associated with coordinate system
M180->15A,S              ; RUN REQUEST
M181->525,S              ; BUFFER OPENED
M182->4CD,S              ; INS MODE
M183->165,S              ; PROGRAM HOLD
M184->1,S                ; PROGRAM NUMBER
M185->15C,S              ; RAPID MODE
M186->166,S              ; LINEAR MODE
M187->L:168              ; CIRCULAR MODE (-1:CIR1/1:CIR2)
M188->48C,S              ; CALL STACK POINTER
M197->L:C2               ; COMMAND FEEDRATE OVERRIDE
M198->L:C0               ; PRESENT FEEDRATE OVERRIDE
;
;Registers associated with axis2
M201->I:FF34,0,12,S      ; #2 12-BIT UPDOWN COUNTER (COUNTS)
M202->AB,S               ; #2 COMMAND VELOCITY
M221->I:FF42,13,1        ; -LIM2 INPUT STATUS
M222->I:FF42,9,1         ; +LIM2 INPUT STATUS
M223->I:FF42,21,1        ; FAULT2 INPUT STATUS
M224->I:FF37,4,1         ; HMFL2 INPUT STATUS
M231->D,4,1              ; #2 POSITIVE LIMIT SET
M232->D,5,1              ; #2 NEGATIVE LIMIT SET
M239->D,29,1             ; #2 DRIVER ENABLE BIT
M240->D,7,1              ; #2 IN-POSITION BIT
M241->22,12,1            ; #2 JOGTO IN PROGRESS
M242->79,S               ; #2 HOME IN PROGRESS
M243->D,6,1              ; #2 DRIVER FAULT SET
M245->22,9,1             ; #2 HOME COMPLETE
M246->D,25,1             ; #2 HOME FLAG ERROR
M247->4,S                ; #2 INC MODE
M261->B0,S               ; #2 COMMAND POSITION (COUNTS)
M262->56E,S              ; #2 ACTUAL POSITION (COUNTS)
M263->L:580              ; #2 JOG REGISTER POSITION (COUNTS)
M264->L:E5               ; #2 POSITION BIAS (COUNTS)
M265->L:DD               ; #2 PROG. TARGET POSITION (USER UNITS)
M266->L:E1               ; #2 PROG. TARGET POSITION (COUNTS)
M267->L:5A1              ; #2 ACTUAL VELOCITY (COUNTS/MSEC)
M291->L:C7               ; #2 SCALE
;
;Registers associated with axis3
M301->I:FF33,0,12,S      ; #3 12-BIT UPDOWN COUNTER (COUNTS)
M302->AC,S               ; #3 COMMAND VELOCITY
M321->I:FF42,14,1        ; -LIM3 INPUT STATUS
M322->I:FF42,10,1        ; +LIM3 INPUT STATUS
```

```
M323->I:FF42,22,1        ; FAULT3 INPUT STATUS
M324->I:FF37,7,1         ; HMFL3 INPUT STATUS
M331->D,8,1              ; #3 POSITIVE LIMIT SET
M332->D,9,1              ; #3 NEGATIVE LIMIT SET
M339->D,30,1             ; #3 DRIVER ENABLE BIT
M340->D,11,1             ; #3 IN-POSITION BIT
M341->22,19,1            ; #3 JOGTO IN PROGRESS
M342->7A,S               ; #3 HOME IN PROGRESS
M343->D,10,1             ; #3 DRIVER FAULT SET
M345->22,16,1            ; #3 HOME COMPLETE
M346->D,26,1             ; #3 HOME FLAG ERROR
M347->5,S                ; #3 INC MODE
M361->B1,S               ; #3 COMMAND POSITION (COUNTS)
M362->56F,S              ; #3 ACTUAL POSITION (COUNTS)
M363->L:581              ; #3 JOG REGISTER POSITION (COUNTS)
M364->L:E6               ; #3 POSITION BIAS (COUNTS)
M365->L:DE               ; #3 PROG. TARGET POSITION (USER UNITS)
M366->L:E2               ; #3 PROG. TARGET POSITION (COUNTS)
M367->L:5A2              ; #3 ACTUAL VELOCITY (COUNTS/MSEC)
M391->L:C8               ; #3 SCALE
;
;Registers associated with axis4
M401->I:FF32,0,12,S      ; #4 12-BIT UPDOWN COUNTER (COUNTS)
M402->AD,S               ; #4 COMMAND VELOCITY
M421->I:FF42,15,1        ; -LIM4 INPUT STATUS
M422->I:FF42,11,1        ; +LIM4 INPUT STATUS
M423->I:FF42,23,1        ; FAULT4 INPUT STATUS
M424->I:FF37,10,1        ; HMFL4 INPUT STATUS
M431->D,12,1             ; #4 POSITIVE LIMIT SET
M432->D,13,1             ; #4 NEGATIVE LIMIT SET
M439->D,31,1             ; #4 DRIVER ENABLE BIT
M440->D,15,1             ; #4 IN-POSITION BIT
M441->29,5,1             ; #4 JOGTO IN PROGRESS
M442->7B,S               ; #4 HOME IN PROGRESS
M443->D,14,1             ; #4 DRIVER FAULT SET
M445->22,23,1            ; #4 HOME COMPLETE
M446->D,27,1             ; #4 HOME FLAG ERROR
M447->6,S                ; #4 INC MODE
M461->B2,S               ; #4 COMMAND POSITION (COUNTS)
M462->570,S              ; #4 ACTUAL POSITION (COUNTS)
M463->L:582              ; #4 JOG REGISTER POSITION (COUNTS)
M464->L:E7               ; #4 POSITION BIAS (COUNTS)
M465->L:DF               ; #4 PROG. TARGET POSITION (USER UNITS)
M466->L:E3               ; #4 PROG. TARGET POSITION (COUNTS)
M467->L:5A3              ; #4 ACTUAL VELOCITY (COUNTS/MSEC)
M491->L:C9               ; #4 SCALE
;
; ACCESSORY I/O M-VARIABLES (1ST MT-0123)
M900->I:FF50,0,1         ; MI/O1
M901->I:FF50,1,1         ; MI/O2
M902->I:FF50,2,1         ; MI/O3
```

```
M903->I:FF50,3,1           ; MI/O4
M904->I:FF50,4,1           ; MI/O5
M905->I:FF50,5,1           ; MI/O6
M906->I:FF50,6,1           ; MI/O7
M907->I:FF50,7,1           ; MI/O8
M908->I:FF50,8,1           ; MI/O9
M909->I:FF50,9,1           ; MI/O10
M910->I:FF50,10,1          ; MI/O11
M911->I:FF50,11,1          ; MI/O12
M912->I:FF50,12,1          ; MI/O13
M913->I:FF50,13,1         ; MI/O14
M914->I:FF50,14,1         ; MI/O15
M915->I:FF50,15,1         ; MI/O16
M916->I:FF50,0,16         ; MI/O1-MI/O16 TREATED AS BYTE
M924->I:FF51,0,1          ; MI/017
M925->I:FF51,1,1          ; MI/O18
M926->I:FF51,2,1          ; MI/O19
M927->I:FF51,3,1          ; MI/O21
M928->I:FF51,4,1          ; MI/O21
M929->I:FF51,5,1          ; MI/O22
M930->I:FF51,6,1          ; MI/O23
M931->I:FF51,7,1          ; MI/O24
M932->I:FF51,8,1          ; MI/O25
M933->I:FF51,9,1          ; MI/O26
M934->I:FF51,10,1         ; MI/O27
M935->I:FF51,11,1         ; MI/O28
M936->I:FF51,12,1         ; MI/O29
M937->I:FF51,13,1         ; MI/O30
M938->I:FF51,14,1         ; MI/O31
M939->I:FF51,15,1         ; MI/O32
M940->I:FF51,0,16         ; MI/O17-MI/O32 TREATED AS BYTE
M948->I:FF52,0,1          ; MI/O33
M949->I:FF52,1,1          ; MI/O34
M950->I:FF52,2,1          ; MI/O35
M951->I:FF52,3,1          ; MI/O36
M952->I:FF52,4,1          ; MI/O37
M953->I:FF52,5,1          ; MI/O38
M954->I:FF52,6,1          ; MI/O39
M955->I:FF52,7,1          ; MI/O40
M956->I:FF52,8,1          ; MI/O41
M957->I:FF52,9,1          ; MI/O42
M958->I:FF52,10,1         ; MI/O43
M959->I:FF52,11,1         ; MI/O44
M960->I:FF52,12,1         ; MI/O45
M961->I:FF52,13,1         ; MI/O46
M962->I:FF52,14,1         ; MI/O47
M963->I:FF52,15,1         ; MI/O48
M964->I:FF52,0,16         ; MI/O33-MI/O48 TREATED AS BYTE
```

I-Variables Definition

I. System I-Variables Online Command Summary

I0 Card Number

Range Nonnegative Integer
Units None
Default 0
Remarks I0 sets the card number of this controller. If more than one UTCs are connected together, the host must use this number to distinguish which one can accept the command issued. Please refer to "**!**" command.

I1 Coordinate System Activation Control

Range 1 - 4
Units None
Default 1
Remarks I1 controls which coordinate systems are activated on a UTC controller. A coordinate system must be activated in order for it to be addressed and accept commands, and to have its automatic capability to run a motion program.

I1 can take values from 1 to 4. The highest numbered coordinate system that is activated is Coordinate System I1. In other words, a given value of I1 activates Coordinate System 1 through Coordinate System I1. Lowering I1 if one or fewer coordinate systems will be used brings one advantage. There is a slight improvement in computational efficiency because de-activated coordinate systems do not have to be checked periodically.

I2 COM2 Baudrate Control

Range $0000 – $00ff
Units None
Default 2
Remarks I2 controls the baudrate and data type through the communication from COM2. Set I2 to 0 will disable COM2. The setting of default value is 19200, N-8-1 (8 data bit, 1 stop bit, no parity check).

Bit 0 .. 3 Baudrate Setting

Value	Baudrate
0	115200
1	9600
2	19200
3	38400

Bit 4 .. 7 Data Format Setting

Value	Format
0	N-8-1
1	O-8-1
2	E-8-1
3	N-7-1
4	O-7-1
5	E-7-1

※ O: Odd parity check, E: Even parity check, N: No parity check

I3 COM2 Handshake Control

Range 0/1
Units None
Default 0
Remarks This parameter determine whether the controller use the RTS/CTS signal for handshake through COM2 or not. If I3 is 1, controller will check this signal before sending the response out.

I6 PLC Programs On/Off Control

Range 0 - 3
Units None
Default 0
Remarks I6 controls which PLC programs may be enabled on power up. There are two types of PLC programs: the foreground programs (PLC 0), which operate at the repetition rate determined by I10 (PLC 0 should be used only for time-critical tasks and should be short); and the background programs (PLC 1 to PLC 15) which cycle repeatedly in background as time allows. I6 controls these as follows:

Setting	Meaning
0	Foreground PLCs off; background PLCs off
1	Foreground PLCs on; background PLCs off
2	Foreground PLCs off; background PLCs on
3	Foreground PLCs on; background PLCs on

Note that an individual PLC program still needs to be enabled to run -- a proper value of I6 merely permits it to be run. Any PLC program that exists at power-up or reset is automatically enabled (even if the saved value of I6 does not permit it to run immediately); also, the **ENAPLC n** command enables the specified program(s). A PLC program is disabled either by the **DISPLC n** command, or by the **OPEN PLC n** command. A **CLOSE** command will automatically re-enable the PLC program at its previous state.

I9 Maximum Digit for Floating Point Returned

Range 0 - 16
Units None
Default 3
Remarks This parameter sets the maximum number of digit for a floating point value to return when the host asked them to response. If the actual digit number of this value exceed the setting of I9, it rounds to the nearest value that the host required.

Example with I9=8
I3
19.99999987
with I9=3
I3
20

I10 Real Time Interrupt Period

Range Any positive integer
Units msec
Default 8
Remarks I10 controls how often certain time-critical tasks, such as PLC 0 and checking for motion program move planning, are performed. These tasks are performed every I10 milliseconds, at a priority level called the “real-time interrupt” (RTI). A value of 2 means that these tasks are performed after 2 milliseconds, 3 means every 3 milliseconds, and so on. The vast majority of users can leave this at the default value. In some advanced applications that push UTC400's speed capabilities, tradeoffs between performance of these tasks and the calculation time they take may have to be evaluated in setting this parameter.

Note: A large PLC0 with a small value of I10 can cause severe problems, because UTC400 will attempt to execute the PLC program every I10 msec. This can starve background tasks, including communications, background PLCs, and the performance will much less than what you expect.

I17 Number of P variable

Range 2048..32768
Units number
Default 2048
Remarks This parameter is used to set number of P variable. The area of P1024 to P2047 occupies the same range of Q variable, i.e. P1024=Q0(&1). These area higher than 2048 will share the area of program (PLC & PROG).

The Indirect addressing should be used on the operation of P variable higher than P1024 in a program.

For Example:

```
P100=2000
CMD"P(P100)=M162/M191"
```

I18 Extension I/O Board Enable

Range 0 – $FF
Units None
Default 0
Remarks UTC controllers can connect to 2*MT0170 or 2*MT0164 or 1*MT0170 1*MT0164 for extension I/O. Each MT0170 has 32 inputs and 16 outputs. Each MT0164 has 4 channels analog output. If we want to use those inputs and outputs, we need to setup I18 properly to enable them.

Setting	Meaning
Bit 0	Port 1 of Board 1(1: Enable 0: Disable)
Bit 1	Port 1 of Board 2(1: Enable 0: Disable)
Bit 2	Port 2 of Board 1(1: Enable 0: Disable)
Bit 3	Port 2 of Board 2(1: Enable 0: Disable)
Bit 4	Port 1 D/A 0~10V(1: Enable 0: Disable)
Bit 5	Port 1 D/A +/-10V(1: Enable 0: Disable)
Bit 6	Port 2 D/A 0~10V(1: Enable 0: Disable)
Bit 7	Port 2 D/A +/-10V(1: Enable 0: Disable)

I19 Digital Inputs Debounce Cycle

Range Nonnegative Integer
Units msec
Default 1
Remarks The users can read the digital inputs status on UTC controller by reading the PGA registers directly, or by reading the memory register $0879 which status is processed by digital filtering. I19 is the debounce period; it means all the inputs must stay in the same status for at least I19 milliseconds to be accepted.

I20 Gathered Data Selection

Range $0000 - $FFFF
Units None
Default 0
Remarks If gather function is enabled, UTC controllers can put the data selected periodically to the gather buffer. There are at most four data can be selected. The x'th data can be set by the x'th digit of I20, and the data comes from the address given in I2x. The definition of each digit of I20 is as following:

Setting	Meaning
0	Disable
1	Decimal Integer Format
2	Floating Point Format
3	Hexdecimal Integer Format

I21 Gathered Data Source 1

Range $000000 - $FFFFFF
Units None
Default 0
Remarks The address of first gather source. $40xxxx for memory buffer and $8100xx for I/O section.

I22 Gathered Data Source 2

Range $000000 - $FFFFFF
Units None
Default 0
Remarks The address of second gather source. $40xxxx for memory buffer and $8100xx for I/O section.

I23 Gathered Data Source 3

Range $000000 - $FFFFFF
Units None
Default 0
Remarks The address of third gather source. $40xxxx for memory buffer and $8100xx for I/O section.

I24 Gathered Data Source 4

Range $000000 - $FFFFFF
Units None
Default 0
Remarks The address of forth gather source. $40xxxx for memory buffer and $8100xx for I/O section.

I25 Gather Period

Range Any Positive Integer
Units msec
Default 1
Remarks If gather function is enabled, UTC controllers can put the data selected periodically to the gather buffer, and the period is set by this parameter.

I26 Gather Buffer Size

Range Any Positive Integer
Units group (set by I20)
Default 0
Remarks If gather function is enabled, UTC controllers can put the data selected periodically to the gather buffer, and the size of gather buffer is set by this parameter.

I27 Gather Start and Stop Control

Range $00 - $FF
Units None
Default 0
Remarks The first digit of this parameter determines the start/stop condition of gather function. If this digit is 0, gather started by “GAT” command and stopped by “ENDG”. If this digit is 1, gather is controlled by the moving of addressed motor. If this digit is 2, gather is controlled by the program running under the addressed coordinate system.

If the second digit is 0($40), gather will be stopped when the gather buffer is full. If this digit is 1, gather will rollover to override the old data.

I28 Gather Stop Delay

Range Nonnegative Integer
Units group (set by I20)
Default 0
Remarks This parameter determines how many data to gather after the stop signal triggered.

1. I50..62 only suitable for UTSD series

I50 Current Loop Proportional Gain

Range Nonnegative Integer
Units None
Default 300 (Depends on motor)
Remarks This parameter sets the proportional gain of current loop control on UTSD series. It is strongly recommended to use the default value for the specified motor.

I51 Current Loop Big Step

Range 0 - 4096
Units None
Default 100 (Depends on motor)
Remarks This parameter sets the big step of current loop control on UTSD series. It is strongly recommended to use the default value for the specified motor.

I52 Current Loop Derivative Gain

Range 0 - 1
Units None
Default 0.8
Remarks This parameter sets the derivative gain of current loop control on UTSD series. It is strongly recommended to use the default value for the specified motor.

I53 Motor Rated Speed

Range 0~6000
Units Revolution per Minute
Default 2000 (Depends on motor)
Remarks This parameter sets the rated speed of motor controlled on UTSD series. It is strongly recommended to use the default value for the specified motor.

I54 Motor Rated Current

Range Nonnegative Floating Point
Units Ampere
Default 2.5 (Depends on motor)
Remarks This parameter sets the rated current of motor controlled on UTSD series. It is strongly recommended to use the default value for the specified motor.

I55 Motor Peak Current

Range Nonnegative Integer
Units Ampere
Default Depends on motor
Remarks This parameter sets the peak current of motor controlled on UTSD series. It is strongly recommended to use the default value for the specified motor.

I56 Motor Encoder Feedback Direction

Range 0/4
Units None
Default 0
Remarks This parameter sets the encoder feedback direction of motor controlled on UTSD series. 0 means clockwise and 4 means counter clockwise.

I57 Alarm Checking Mask

Range $0 - $7FF
Units None
Default 0
Remarks This parameter determines which alarm check will be activated by setting the corresponding bit to 0. The default value will make all the checking active.

Bit	Meaning
0	Over Current Protection
1	DC Bus Under Voltage
2	DC Bus Over Voltage
3	Over Heat
4	Resolver/Encoder Fault
5	Over Load
6	Current Feedback Fault
7	Over Speed
8	Fatal Following Error
9	UVW Phase Error
10	Parameter Fault

I58 Motor Type Selection

Range $00 - $7F
Units None
Default None
Remarks This parameter sets the motor type controlled on UTSD series. Motors now available are as following table.

Setting	Meaning
$0x	Baldor Resolver Motors
$1x	Yaskawa Motors
$2x	Encoder Feedback Motor (Baldor, Yeli, Linear…)
$3x	Sinano Motors
$4x	Sanyo Denki Motors
$5x	ImEnc Motors

I59 Motor Pole Pair Number

Range 4/8/14
Units None
Default None
Remarks This parameter sets the pole pair number of motor controlled on UTSD series.

I60 Motor Encoder Resolution

Range 4096/8000/10000/20000
Units Pulse per Revolution
Default None
Remarks This parameter sets the feedback resolution of motor controlled on UTSD series.

I61 Motor Index Position

Range 0 - 1023
Units None
Default None
Remarks This parameter sets the index position of motor controlled on UTSD series.

I62 Motor Magnetization Current

Range -32768 - 32767
Units None
Default None
Remarks This parameter sets the magnetization current of motor controlled on

UTSD series. This should be set to 0 for non-induction motors.

I80 Handwheel Decode Control

Range 0 - 7
Units None
Default 4

Remarks This parameter sets the decode method and direction for the onboard hand wheel encoder input.

Encoder Decode Bits

Bit2	**Bit1**	**Bit0**	**Function**
0	X	0	CW A/B Phase
0	0	1	CW +/- Pulse
0	1	1	CW Pulse/Direction
1	X	0	CCW A/B Phase
1	0	1	CCW +/- Pulse
1	1	1	CCW Pulse/Direction

I82 DA1 Bias

Range -2048 - 2047
Units 10V/2048
Default 0
Remarks This parameter sets the offset for the first channel digital to analog output. It is often used to compensate the zero voltage when the command is zero.

I83 DA2 Bias

Range -2048 - 2047
Units 10V/2048
Default 0
Remarks This parameter sets the offset for the second channel digital to analog output. It is often used to compensate the zero voltage when the command is zero.

I84 DA1 Slew Rate

Range Nonnegative Floating Point
Units 10V/2048
Default 0
Remarks This parameter sets the changing rate for the first channel digital to analog output. If I84 sets to 0, the analog voltage will change to the new

command immediately.

I85 DA2 Slew Rate

Range Nonnegative Floating Point
Units 10V/2048
Default 0
Remarks This parameter sets the changing rate for the second channel digital to analog output. If I85 sets to 0, the analog voltage will change to the new command immediately.

I86 DA PWM Frequency

Range 0 .. 2
Units None
Default 0
Remarks This parameter sets the PWM frequency for the digital to analog output.

0 : 4k	-2048 ~ 2047 for -10V ~ 10V
1 : 8k	-1024 ~ 1023 for -10V ~ 10V
2 : 16k	-512 ~ 511 for -10V ~ 10V

II. Motor I-Variable

x = Motor Number (#x, x = 1 to 4).

I x00 Motor x Activate Control

Range 0 - 2
Units None
Default 1
Remarks This parameter controls whether the motor is active or not. There are three different states to be set. When Ix00 is 0, the motor is disabled. For the motor that is not in use, it can save the computation time. When Ix00 is 1, the motor is active. Each motor need to be in this state before it can be used. When Ix00 is 2, the motor is disabled but it can be used in dry run mode.

Ix01 Motor x Jog / Home Acceleration Time

Range Any positive floating point number
Units msec
Default 50
Remarks This parameter sets the time spent in acceleration in a jogging and homing. A change in this parameter will not take effect until the next move command. For instance, if you wanted a different deceleration time from acceleration time in a jog move, you would specify the acceleration time,

command the jog, change the deceleration time, then command the jog move again (e.g. **J=**), or at least the end of the jog (**J/**).

Ix02 Motor Open Loop Command Slewrate

Range Any positive floating point number
Units bits/msec (32768bits = 10V)
Default 1
Remarks When the motor is in open loop control mode, the users can issue an “O” command to change the output voltage. The rate of change is controlled by this parameter. The actual voltage will add or subtract the value of Ix02 per millisecond until it reaches the new command required. So if you want the command to be changed faster, you should set this value larger, and vice versa.

Ix03 Motor x Jog Speed

Range Any floating point number
Units counts / sec (pps)
Default 10000
Remarks This parameter sets the commanded speed of a jog move. Direction of the jog move is controlled by the jog command.

A change in this parameter will not take effect until the next move command. For instance, if you wanted to change the jog speed on the fly, you would start the jog move, change this parameter, and then issue a new jog command.

Ix04 Motor Deceleration Rate on Position Limit or Abort

Range Any positive floating point number
Units count / msec^2
Default 5
Remarks This parameter sets the rate of deceleration that motor exceeds hardware or software limits, or has its motion aborted by command **A** or **<CTRL-A>**. This value should be set to a value near the maximum physical capability of the motor. Once the deceleration happened, each motor’s command speed was decreased by Ix04 per millisecond until it comes to a completely stop. So if you want to stop the motors faster, increase this value, and vice versa.

Do not set this parameter to zero, or the motor will continue indefinitely after an abort or limit.

Ix05 Motor x Master Following Enable

Range 0 - 12
Units None
Default 0
Remarks This parameter disables or enables motor x's position following function. A value of 0 means disabled. A value from 1 to 4 means position following enabled (electrical gear). Master source is determined by Ix85 and the scale is set by Ix06. A value from 5 to 8 means velocity following enabled. Master source is determined by Ix85 and the scale is set by Ix06. A value from 9 to 12 means position following enabled. It is similar to 1 to 4 but the only difference is it can work during the motion is running.
Bit4~Bit7 means the range of the position following without function(Dead Band)

Ix06 Motor x Master Scale Factor

Range Any floating point number
Units None
Default 1
Remarks This parameter controls the following ratio of motor x. The master input signal was multiplied by this scale factor first then add into the command position. Ix06 can be changed on the fly to permit real-time changing of the following ratio.

Ix07 Motor x Homing Speed and Direction

Range Any signed integer
Units counts / sec (pps)
Default 5000
Remarks This parameter controls the speed and direction of a homing-search move for motor x. Changing the sign reverses the direction of the homing move. A negative value specifies a home search in the negative direction; a positive value specifies the positive direction.

Ix08 Motor x Home Offset

Range Any floating point number
Units counts
Default: -2000
Remarks This parameter is the relative position of the end of the homing cycle to the position at which the home trigger was made. That is, the motor will command a stop at this distance from where it found the home flag and call this commanded location as motor position zero. This permits the motor zero position to be at a different location from the same home trigger position

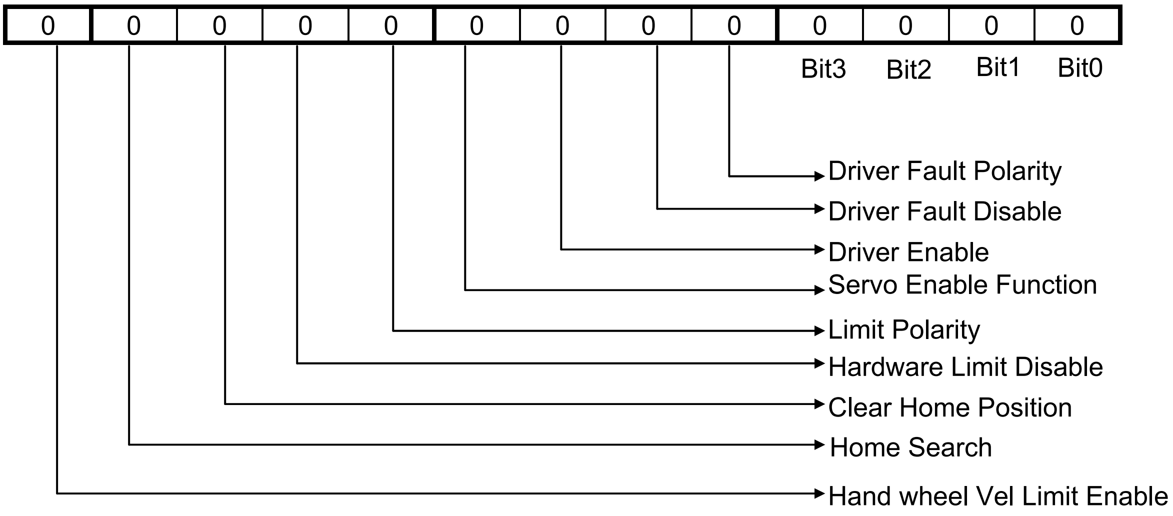
Ix09 Motor x Flag Control

Range Any integer
Units None
Default 0
Remarks This parameter sets the function of flags such as overtravel limit switches, home flag, driver fault flag, and driver enable output.

The overtravel-limit inputs specified by this parameter must be held low in order for motor x to be able to command movement. If this function is disabled, the limit inputs can be used as general-purpose inputs. The polarity of the limit switches is determined by bit-8 of this parameter.

The driver fault function is enabled by held the bit-5 of this parameter low, and the polarity of this signal is determined by bit-4. The same as limit inputs, if this function is disabled, driver fault input can be used as general-purpose input.

The polarity of driver enable output is determined by bit-6 of this parameter.



Hand wheel Vel Limit Enable= 0 The velocity following the hand wheel is unlimited
= 1 The velocity following the hand wheel is limit by Ix03 and FR override
Home Search Disable = 0 Home Move Allowed
= 1 Home Move not Allowed
Clear Home Position = 0 Reset Motor Position After Home Complete
= 1 Keep Motor Position After Home Complete
Limit Disable = 0 Hardware Limit Enable
= 1 Hardware Limit Disable
Limit Polarity = 1 Limit Switch is High True (use B-type switch)
= 0 Limit Switch is Low True (use A-type switch)
Servo Enable Function On = 1 Servo Enable Deactivate (can be used as DOUT)
= 0 Servo Enable Activate
Driver Enable Polarity = 0 Driver Enable is Low True
= 1 Driver Enable is High True

Driver Fault Disable	= 0	Driver Fault Enable
	= 1	Driver Fault Disable
Driver Fault Polarity	= 1	Driver Fault is High True (Normally Close)
	= 0	Driver Fault is Low True (Normally Open)
Bit3	= 0	Servo Disable on Power Up
	= 1	Servo Enable on Power Up
Bit2	= 0	Keep Current Position on Power Up
	= 1	Reset Current Position on Power Up
Bit1	= 0	Jog Affected by Feedrate Override
	= 1	Jog Not Affected by Feedrate Override
Bit0	= 0	DAC or Pulse Command Not Converted
	= 1	DAC or Pulse Command Converted

Ix10 Motor x Positive Software Limit

Range Any integer
Units counts
Default 0
Remarks This parameter sets the position for motor x which if exceeded in the positive direction causes a deceleration to a stop (controlled by Ix04) and allows no further positive position increments or positive output commands as long as the limit is exceeded. If this value is set to zero, there is no positive software limit (if you want 0 as a limit, use 1). This limit is automatically de-activated during homing search moves.

This limit is referenced to the most recent power-up zero position or homing move zero position. The physical position at which this limit occurs is not affected by axis offset commands (e.g. **PSET**, **X=**, …), although these commands will change the reported position value at which the limit occurs.

Ix11 Motor x Negative Software Limit

Range Any integer
Units counts
Default 0
Remarks This parameter sets the position for motor x which if exceeded in the negative direction causes a deceleration to a stop (controlled by Ix04) and allows no further negative position increments or negative output commands as long as the limit is exceeded. If this value is set to zero, there is no negative software limit (if you want 0 as a limit, use -1). This limit is automatically de-activated during homing search moves.

This limit is referenced to the most recent power-up zero position or homing move zero position. The physical position at which this limit occurs is not affected by axis offset commands (e.g. **PSET**, **X=**, …), although these commands will change the reported position value at which the limit occurs.

Ix12 Motor x Coordinate Position Displacement

Range Any floating point
Units user unit
Default 0
Remarks UTC offers a full set of instructions to do the coordinate system transformation very easily, including position rotation and displacement. This parameter is the current coordinate position displacement of motor x.

When motion program is not running, we can set Ix12 by online command to change the coordinate displacement of motor x without any problem. But in a blended motion program, it is strongly recommend not to change this parameter directly, use **ADIS** or **IDIS** instruction instead (this two commands won't stop the blended state). This parameter will be reset to 0 on power up.

Ix13 Motor x Coordinate Position Scaling

Range Any positive floating point
Units None
Default 1
Remarks UTC controller offers a full set of instructions to do the coordinate system transformation very easily, including position rotation and displacement. This parameter is the current coordinate position scaling of motor x.

When motion program is not running, we can set Ix13 by online command to change the coordinate scaling of motor x without any problem. But in a blended motion program, it is strongly recommend not to change this parameter directly, use **ASCL** or **ISCL** instruction instead (this two commands won't stop the blended state). This parameter will be reset to 1 on power up.

Ix15 Motor x Backlash Size

Range Any non-negative integer
Units counts
Default 0
Remarks This parameter allows UTC400 to compensate for backlash in the motor's coupling by adding or subtracting (depending on the new direction) the amount specified in the parameter to the commanded position on direction reversals (this offset will not appear when position is queried or displayed). A value of zero means no backlash. The rate at which this backlash is added or subtracted ("taken up") is determined by Ix16.

Ix16 Motor x Backlash Takeup Rate

Range 1 .. 256
Units counts / msec
Default 1
Remarks This parameter determined how fast backlash (of size Ix15) is “taken up” on direction reversal. If Ix16 is zero, backlash is effectively disabled. Ix16 is usually set as high as possible without creating dynamic problems.

Ix17 Motor x Rollover Range

Range Any non-negative integer
Units counts
Default 0
Remarks This parameter permits position rollover on incremental axes. When Ix17 is greater than zero, rollover is active. If the commanded position greater than the value set in this parameter, it will rollover up from 0. If the commanded position less than zero, it will rollover down from Ix17. Thus the commanded position of motor x will be limited between 0 and Ix17, it prevents the position overflow while the motor is continuously incremental in one direction.

Ix19 Motor x Velocity Weighting

Range Any positive floating point
Units None
Default 1
Remarks When the user unit of the axes that interpolated together is different. Sometimes the feedrate will cause one of those axes moving too slow or too fast. We can use this parameter to adjust the weighting for feedrate calculation to prevent this problem.

Ix20 Motor x PID Proportional Gain

Range Any non-negative integer
Units None
Default 20000
Remarks This parameter provides a DAC command output proportional to the position following error. It acts like an electrical spring, the higher Ix20 is, the stiffer the spring is. If we set the Ix20 too small, the motor will become too weak to follow the command. If we set the value too high, it will cause the motor buzzing and maybe come up with the driver overload.

Ix21 Motor x PID Derivative Gain

Range Any non-negative integer
Units None
Default: 400

Remarks This parameter provides a DAC command output proportional to the measured velocity of motor. It acts like an electrical damper, the higher Ix21 is, the heavier the damping is. If we set the Ix21 too small, the system can not overcome the overshoot caused by the rapid response. If we set the value too high, it will also cause the motor buzzing.

Ix22 Motor x Velocity Feedforward Gain

Range Any non-negative integer
Units None
Default 400
Remarks This parameter provides a DAC command output proportional to the command velocity of motor. It can reduce the following error due to the damping caused by Ix21 or physical effects.

Ix23 Motor x PID Integral Gain

Range Any non-negative integer
Units None
Default 1000
Remarks This parameter provides a DAC command output proportional to the time integral of position following error. It is often used to eliminate the steady state error of the system.

Ix24 Motor x PID Integration Mode

Range 0/1
Units None
Default 1

Remarks This parameter sets two different mode of integral servo loop control. When Ix24 is set to 1, integration is performed only when the command velocity is zero. If Ix24 is set to 0, integration is performed all the time.

Ix25 Motor x Acceleration Feedforward Gain

Range Any non-negative integer
Units None
Default 500
Remarks This parameter provides a DAC command output proportional to the

command acceleration of motor. It can reduce the following error due to the inertial lag.

Ix26 Motor x Position Feedback Address

Range $0000 - $FFFF
Units None
Default I126: $CF16 (UTC-V); $CF24 (UTC-P)
I226: $CF17 (UTC-V); $CF25 (UTC-P)
I326: $CF18 (UTC-V); $CF26 (UTC-P)
I426: $CF19 (UTC-V); $CF27 (UTC-P)

Remarks This parameter sets the controller where to get the position feedback of motor x. There are five onboard channels for the encoder input, all can be chosen for position feedback.

Ix27 Motor x Velocity Feedback Address

Range $0000 - $FFFF
Units None
Default I127: $CF16 (UTC-V); $CF24 (UTC-P)
I227: $CF17 (UTC-V); $CF25 (UTC-P)
I327: $CF18 (UTC-V); $CF26 (UTC-P)
I427: $CF19 (UTC-V); $CF27 (UTC-P)

Remarks This parameter sets the controller where to get the position feedback of motor x. There are five onboard channels for the encoder input, all can be chosen for position feedback.

Ix28 Motor x Velocity Feedback Scale

Range Any positive floating point
Units None
Default 1
Remarks When the controller uses dual feedback for both position and velocity feedback, sometimes the scales of these two encoders are different. In this case, Ix28 can be used to adjust the velocity scale to match the position feedback. Then Ix28= the position counts (from Ix26), the velocity counts (from Ix27) of the same length. In single feedback system, just leave this parameter as default.

Ix29 Motor x DAC Bias

Range -32767 - 32767
Units 10V/32767
Default 0
Remarks This parameter sets the offset for the servo loop command digital to analog output. It is often used to compensate the zero reference between the controller DAC output and driver analog input.

Ix30 Motor x DAC Limit

Range 0 - 32767
Units 10V/32767
Default 20480 (~6.25V)
Remarks This parameter sets the maximum voltage can be output for motor x command. 32767 means ±10V (100%) and 20480 means ±6.25V (62.5%), and so on. We should set this parameter properly in order not to damage the motors.

Ix31 Motor x Fatal Following Error

Range Any non-negative integer
Units Count
Default 2000
Remarks This parameter is very important for the system safety. It defined the maximum following error allowed for motor x. If the following error exceeds Ix31, the motor will be killed immediately. If this motor is under a motion program which is running, the motion program will also be aborted. Set this parameter to 0 will disable following error checking, but it is strongly recommended not to doing so.

Ix32 Motor x Dead Band Size

Range Any non-negative integer
Units Count
Default 0
Remarks This parameter sets the dead band size for motor x. Whenever the position following error within Ix32, controller will ignore the following error and make no DAC output.

Ix33 Motor x In Position Band

Range Any non-negative integer
Units Count
Default 10

Remarks This parameter sets the in position range for motor x. Whenever the position following error within Ix33, there is an in-position bit (Mx40) in controller will be set. If all the motors in the addressed coordinate system are in position, the in position LED indicator will light.

Ix34 Motor x Big Step Size

Range Any non-negative integer
Units Count
Default 100
Remarks This parameter sets the maximum following error allowed to enter the servo filter for motor x. Whenever the position following error exceeds Ix32, controller will clip the following error to this value, thus ensure the system stability.

Ix35 Motor x Integration Limit

Range Any non-negative integer
Units Count
Default 65536
Remarks This parameter limits the value of integration on servo filter when it was activated

III. Coordinate System I-Variable

x = Coordinate System Number (#x, x = 1 to 4)

Ix50 Coordinate System x Blended Move Enable Control

Range 0 / 1
Units None
Default 0
Remarks Ix50 controls whether programmed moves for coordinate system x is automatically blended or not. If this parameter set to 1, programmed moves -- **LIN**, **SPLINE**, and **CIR**-mode -- are blended together with no intervening stop. Upcoming moves are calculated during the current moves. If this parameter is set to 0, there is a brief stop in between each programmed move (it effectively adds a **DWELL 0** command), during which the next move is calculated.

This parameter is only acted upon when the **R** or **S** command is given to start program execution. To change the variable from 0 to 1 while the program is running, the moves start to blended together immediately. But if you change it from 1 to 0, the motion will stop blended while the next **RPD** or **DWELL** command is encountered.

Ix51 Coordinate System x Default Program Number

Range 1 .. 1023
Units None
Default 0
Remarks Before we run a motion program, we need to assign a program number by “B” command. If the ‘R’ or ‘S’ command is issued but no program number has been assigned since power up, Ix51 will be used instead. If this parameter is 0, it uses the previous number before the power is turned off.

Ix52 Coordinate System x Default Program Acceleration Time

Range Any Nonnegative Integer
Units msec
Default 100
Remarks This parameter sets the default time for commanded acceleration for programmed blended **LIN** and **CIR** mode move in coordinate system x. Even though this parameter makes it possible not to specify acceleration time in the motion program, you are strongly encouraged to use **TA** in the program and not to rely on this parameter, unless you must keep to a syntax standard that does not support this (e.g. RS-274 “G-Codes”). Specifying acceleration time in the program along with speed and move modes makes it much easier for later debugging.

The acceleration time is also the minimum time for a blended move; if the distance on a feedrate-specified (**F**) move is so short that the calculated move time is less than the acceleration time, or the time of time-specified (**TM**) move is less than the acceleration time, the move will be done in the acceleration time instead. This will slow down the move.

Ix53 Coordinate System x Default Program S-Curve Time

Range 0 - 100
Units percent
Default 50
Remarks This parameter sets the default time in each “half” of the “S” in S-curve acceleration for programmed blended **LIN** and **CIR** mode move in coordinate system x. Even though this parameter makes it possible not to specify acceleration time in the motion program, you are strongly encouraged to use **TS** in the program and not to rely on this parameter, unless you must keep to a syntax standard that does not support this (e.g. RS-274 “G-Codes”). Specifying acceleration time in the program along with speed and move modes makes it much easier for later debugging.

If Ix53 is zero, the acceleration is constant throughout the Ix52 time and the velocity profile is trapezoidal. If Ix53 is greater than zero, the acceleration will start at zero and linearly increase through Ix53 time.

Ix54 Coordinate System x Default Program Feedrate

Range Any positive floating point number
Units (user unit) / minute
Default 3000
Remarks This parameter sets the default feedrate (command speed) for programmed blended **LIN** and **CIR** mode move in coordinate system x. The first use of an **F** or **TM** statement in a motion program override this value. The velocity unit is defined by the position per minute, as defined by axis definition statements.

Even though this parameter makes it possible not to specify feedrate in the motion program, you are strongly encouraged not to rely on this parameter and to declare your feedrate in the program.

Ix55 Coordinate System x Program Acceleration Mode

Range 0/1
Units None
Default 0
Remarks This parameter sets the acceleration mode for programmed **LIN** and **CIR** mode move in coordinate system x. There are two types of acceleration as following:

When Ix55 set to 1, it's at fix acceleration mode. Every time we start to run the motion program, acceleration will be calculated according to the default feedrate and acceleration time, that is Ix54/Ix52. In order to get a fix acceleration during motion, if the feedrate get higher, the acceleration time must be get longer proportionally, and vice versa. So the **TA** is changing all the time while the program is running.

When Ix55 set to 0, it's at fix acceleration time mode. Each step in the motion program will use the same **TA** assigned in the program. If the feedrate get higher, the acceleration will also get larger, and vice versa. So the acceleration is changing all the time while the program is running.

Ix56 Coordinate System x Time Base Slew Rate

Range 1..100
Units % / msec
Default 1
Remarks Ix56 controls the rate of change of the time base for coordinate system x. It effectively works in two slightly different ways, depending on the source of the time base information. If the source of the time base is the "%" command register, then Ix56 defines the rate at which the "%" (actual time base) value will slew to a newly commanded value. If the rate is too high, and the % value is changed while axes in the coordinate system are moving, there will be a virtual step change in velocity. For these type of applications, Ix56 is set relatively low (often 0.001 to 0.01) to provide smooth changes.

If there is a hardware source (as defined by Ix85), the commanded time-base value changes every servo cycle, and the rate of change of the commanded value is typically limited by hardware considerations (e.g. inertia). In this case, Ix56 effectively defines the maximum rate at which the "%" value can slew to the new hardware-determined value, and the actual rate of change is determined by the hardware. If you wish to keep synchronous to a hardware input frequency, as in a position-lock cam, Ix56 should be set high enough that the limit is never activated. However, following motion can be smoothed significantly with a lower limit if total synchronicity is not required.

Ix57 Coordinate System x Program Rapid Move Acceleration Time

Range Any positive integer
Units msec
Default 100
Remarks This parameter sets the default time for commanded acceleration for programmed **RPD** mode move in coordinate system. The specified S-curve time is also set by Ix53.

Ix58 Coordinate System x Program Rapid Move Feedrate

Range Any positive floating point number
Units (user unit) / minute
Default 5000
Remarks This parameter sets the velocity for programmed **RPD** mode move in coordinate system x. The unit is the same with Ix54 and it can be set to be affected by feedrate override or not by Ix55. Usually we will set a larger value than Ix54 to do a faster rapid move. The acceleration time of rapid move is defined by Ix57.

Ix59 Coordinate System x Rapid Mode

Range 0/1
Units None
Default 0
Remarks This parameter sets whether the RPD move affected by feedrate override or not. When Ix59 set to 0, the rapid move will be affected by feedrate override. If Ix59 set to 1, the rapid move will always on the real speed and won’t be affected by feedrate override.

Ix60 Coordinate System x Maximum Permitted Program Acceleration

Range Any Nonnegative floating point number
Units (user unit) / minute / msec
Default 0
Remarks This parameter sets a limit to the allowed acceleration in motion program moves. If a move command in a motion program requests a higher acceleration by given its TA and TS time settings, the acceleration for all motors in the coordinate system is stretched out proportionately so that the vector acceleration will not exceed this parameter, yet the path will not be changed. The minimum value of TA will be Feedrate / Ix51 (unit : msec).

When moves are broken into small pieces and blended together, this limit can affect the velocity, because it limits the calculated deceleration for each piece, even if that deceleration is never executed, because it blends into the next piece
If this parameter is set to 0, it means there is no limitation on the acceleration.

Ix61 Coordinate System x Rotate Angle

Range Any floating point
Units degree
Default 0
Remarks UTC controller offers a full set of instructions to do the coordinate system transformation very easily, including position rotation and displacement. This parameter is the current rotate angle of coordinate system.

When motion program is not running, we can set Ix61by online command to change the rotate angle without any problem. But in a blended motion program, it is strongly recommend not to change this parameter directly, use **AROT** or **IROT** instruction instead (this two commands won't stop the blended state). This parameter will be reset to 0 on power up.

Ix62 Coordinate System x Master Source

Range $00 - $FF
Units None
Default 0
Remarks UTC controller offers five on-board encoder input ports. We can set the bits 0 ~ 2 of Ix62 properly to choose the source of time base signal coming from:

Setting	Meaning
0	Use internal time base
1	Master signal from address defined in I185
2	Master signal from address defined in I285
3	Master signal from address defined in I385
4	Master signal from address defined in I485

When bit 3($8) of Ix62 is equal to 1, external time base only depends on the counts come in from the master. So the time base is always positive no mater which direction the master is

When bit 4($10) of Ix62 is equal to 1, it means coordinate system x will use the external triggered time-base. It acts just like the external time-base, except it will also use the trigger signal of forth channel to trigger the time-base. Before the trigger signal comes, the time-base will keep frozen. After triggered, Ix62 will be automatically changed this bit to 0 and start the regular external time-base process.

Bit 5($20): Internal operation usage. Do not set.

When bit 6($40) of Ix62 is equal to 1, the position while the cam table enabled is matched by "PreJog", otherwise it was matched by "PSET".

Bit7($80): When this bit is equal to 1, it means no longer time base following for more smooth moving. For example: Ix62=$91 It follows the master1's position and speed upon triggered.

This parameter will be reset to 0 on power up.

Ix63 Coordinate System x External Time-Base Scale

Range Any non-zero floating point
Units count/msec
Default 100
Remarks This parameter sets the scale when we using the external time-base control (when Ix62 is greater than 0, see Ix62 above). The value of 1 means every one count input from master represents one millisecond. That is, when the master frequency is 1KHz, the time-base is equal to 100%. So we need to calculate the scale properly to meet the requirement that the maximum master frequency input won't produce a too large time-base (it is recommended not exceeding 200%).

Ix64 Coordinate System x External Time-Base Offset

Range Any positive integer
Units Count
Default 0
Remarks This parameter sets the offset when we using the external trigger time-base control(Ix62 Bit4=1($10)). The time base will be defrozen on the trigger occurred plus this offset.

Ix65 Coordinate System x Cam Table Rollover

Range Any Nonnegative Integer
Units Count
Default 0
Remarks When the cam table function was enabled in the coordinate system x, we can set a rollover value which when the master exceeds this value, the master value will rollover back to zero.

Ix66 Coordinate System x Cam Table Output Mask

Range $0000 - $FFFF
Units None
Default 0
Remarks When the cam table function was enabled in the coordinate system x, we can set this parameter to determine which output can be controlled by the cam table. Bit0 to bit7 mapped to M01 to M08, bit8 to bit15 mapped to M40 to M47. If the bit is set to 1 means the corresponding outputs are under controlled in the cam table.

IV. Encoder I-Variable

x = Encoder Channel Number (#x, x = 1 to 4)

Ix80 Encoder Decode Control

Range 0 - 7
Units None
Default 4
Remarks This parameter sets the decode method and direction for the onboard encoder input.

Encoder Decode Bits

Bit2	**Bit1**	**Bit0**	**Function**
0	0	0	CW A/B Phase
0	0	1	CW +/- Pulse
0	1	X	CW Pulse/Direction
1	0	0	CCW A/B Phase
1	0	1	CCW +/- Pulse
1	1	X	CCW Pulse/Direction

Ix81 Encoder Capture Control

Range 0 - 15
Units None
Default 3
Remarks This parameter determines which signal triggers a position capture of the counter for encoder x.

Setting	Function
0	Software Control
1	Rising Edge of CHCn
2	Rising Edge of Flag n
3	Rising Edge of [CHCn and Flag n]
4	Software Control
5	Falling Edge of CHCn
6	Rising Edge of Flag n
7	Rising Edge of [CHCn/ and Flag n]
8	Software Control
9	Rising Edge of CHCn
10	Falling Edge of Flag n
11	Rising Edge of [CHCn and Flag n/]
12	Software Control
13	Falling Edge of CHCn
14	Falling Edge of Flag n
15	Rising Edge of [CHCn/ and Flag n/]

Note: To do a software controlled position capture, present this setting to 0 or 4; when the setting is then changed to 8 or 12, the capture is triggered.

Ix82 Encoder Capture Flag Select

Range 0 - 3
Units None
Default 0

Remarks This parameter determines which flag is used for trigger signel for encoder x.

Setting	Meaning
0	Home Flag
1	Positive Limit Flag
2	Negative Limit Flag
3	Driver Fault Flag

Ix85 Master x Source Address

Range Any nonnegative integer
Units None
Default $CF1A (Hand wheel)
Remarks UTC controller offers up to four master sources. Bit 0..16 of Ix83 store the address of master being used. Bit 16..20 define the data length, it must be greater than 1, and it also define the data type is position or velocity.
If bit22 is 0, the address point to memory area, if bit22 is 1, the address point to I/O area. If bit 23 is 0, the data is in binary format, if bit23 is 1, the data is grey code format.
If no any master is needed for the system, this parameter should set to zero to save background scanning time.

Ix86 Master x Moving Average Buffer Size

Range 1 .. 128
Units msec
Default 1
Remarks Once the master source was selected, the frequency of input signals were put into the buffer which size was defined by this parameter once per millisecond. The final result used by the following function is the average value in the whole buffer. This can prevent a sudden change caused by the master source and make the following much smoother than without the buffer does. The buffer is rollover, so we can guarantee that the master signal will never get lost.

Ix87 Master x Backlash Hysteresis

Range Any Nonnegative Integer
Units Count
Default 0

Remarks Once the master source was selected, when it goes on to the opposite direction, it must keep on this way exceeds Ix87 count to affect the slaves.

Online Command Specification

I. Online Command Summary

Notes

spaces	**:** Spaces are not important unless special noted
{ }	**:** Item in **{ }** can be replaced by anything fitting definition
[ ]	**:** Item in **[ ]** is an option
[{item}…]	**:** Repeated syntax
[..{item}]	**:** The periods are used to specify a range

Definitions

constant	**:** Number that is non-changeable
variable	**:** Variable like **I , M , P , Q**
expression	**:**combination of constant, variable, function and operator
data	**:**constant without parentheses or expression with parentheses
axis	**:** element of coordinate system. It can be X, Y, Z, U, V, W, A, B, C

1. Global Online Command

Control Character Command

<Ctrl-D>	Disable all PLC programs
<Ctrl-F>	Report following-rror positions of all motors(count)
<Ctrl-K>	Disable all drivers
<Ctrl-O>	Feed hold for all Coordinate System
<Ctrl-P>	Report positions of all motors(count)
<Ctrl-Q>	Report mechanical positions of all motors(count)
<Ctrl-V>	Report the actual velocities of all motors(count)
<Ctrl-X>	Terminate the message reporting

Address Mode Command

#	Report the currently addressed motor number
#**{constant}**	Address a motor(1-4)
#**{constant}***	Force all command source to address a motor
&**{constant}**	Address a Coordinate System(1-4)
&	Report the currently addressed coordinate system

Global Variable Command

I{constant}[..{constant}]	Report the specified I-variable value
P{expression}	Report the specified P variable value
Q{expression}	Report the specified Q variable value
M{expression}	Report the specified M variable value
M{expression}->	Report M variable definition
I{constant}={expression}	Assign the specified I-variable value and Report the specified I-variable value
P{expression}={expression}	Assign the specified P variable value and report the specified P variable value
Q{expression}={expression}	Assign the specified Q variable value and report the specified Q variable value
M{expression}={expression}	Assign the specified M variable value and report the specified M variable value
M{expression}->{definition}	Assign the specified M variable definition and report the specified M variable definition
I{constant}=*	Reset the specified I-variable value from flash memory(saved value)
I{constant}=**	Reset the specified I-variable value as the default value
M{expression}->*	Assign M{const} as a normal variable
M{expression}->**	Reset the specified M variable as the default value
LISTI{constant}	Report the simple definition of I-variable

PLC Command

ENAPLC{const}[,{const}...]	Enable the specified PLC program
DISPLC{const}[,{const}...]	Disable the specified PLC(s)

Buffer Control Command

OPEN PROG{constant}	Open a motion program buffer
OPEN CAM{constant}	Open a Cam program buffer
OPEN ROT	Open the rotary buffer
OPEN PLC{constant}	Open a PLC program buffer
OPENBUF{address}	Open the addressed buffer memory
CLOSE	Close the currently opened program buffer
CLEAR	Clear currently opened buffer
SIZE	Report the available buffer size
LISTPROG[constant]	Report the existed motion program numbers or specified motion program contents
LISTPLC[constant]	Report the existed PLC program number or specified PLC Program contents
LISTCAM[constant]	Report the existed cam program numbers or

	specified cam program contents
LISTBUF{address}[,{length}]	Report the contents of the addressed buffer memory
LISTGAT	Report the contents of data gathering
GAT	Start to do data gathering
ENDG	Stop data gathering
RD{Type}{address}[,{length}]	Report the contents of the addressed buffer memory
SAVE	Save all the contents from static RAM to FLASH ROM
$$	Software reset
$$$	Card conditions reset
$$$* **	Card reset and re-initialization
PWD={string}	Set the control card password
???	Report system status
??	Report the coordinate system status
?	Report the motor status

2. Coordinate System Online Command

Coordinate Axis Definition Command

#{constant}->{axis}	Assign an axis definition for the specified motor
#{constant}->	Report the specified motor's coordinate system axis definition
CS	Report all the axes specification of the motors

Coordinate System Command

%{constant}	Set Feedrate Override value
%	Report the current Feedrate Override value

Coordinate Axis Related Command

{axis}={constant}	Set the specified axis position
INIT	Cancel all the axes displacement and rotation
ABS	
INC	
NORMAL	

Attention: ABS,INC,NORMAL may be (on line command) also. See(buffer command)

Motion Program Command

B{constant}	Select Motion Program number
B*	Select Motion Program number indirectly by the content of Mx88
R	Execute the Motion Program
S	Execute the motion program step by step
R*	Start to execute Motion Programs for all active Coordinate Systems.
S*	Step execution of Motion Programs for all active Coordinate Systems
ENACAM{constant}	Enable the specified Cam program
H	Program feed hold
A	Abort Program Execution and Motor move
A*	Abort all Program Execution and motor moves

Buffer Control Command

PC	Report program pointer number
PE{constant}	Report the current program line number
PL{constant}	Report the to be executed program line number
PR	Report the lines count in the rotary buffer that have not been executed
LIST	Report the program contents to be executed
LISTPE{constant}[,{length}]	Report the contents of the program(or cam)
DEFROT{constant}	Define Rotary Buffer for motion program
DELROT	Delete the Rotary Buffer

3. Motor Control Online Command

Normal Command

HM	Motor home searching
HMZ	Do a Zero-Move Homing
O{data}	Set the output voltage
@	Specify an Online Command
K	Driver disabled

Jog Command

J+	Jog to positive direction
J-	Jog to negative direction
J/	Jog stop
J=	Jog to a variable specified absolute position
J={constant}	Jog a motor to a specified position

J:{constant}	Jog a distance specified by {constant}
J:	Jog a variable specified distance
J*	Jog the addressed motor back to the motion program stop

Report Command

P	Report the position of a specified motor
V	Report the speed of specified motors

4. Program Pointer Control Online Command

END	End of a program
JP[constant]	Program pointer points to a specified line number
JPN{constant}	Program pointer points to a specified label number
JP*	Program pointer indirectly points to the content of Mx89.
NEXT{constant}	Program pointer point to next program line
UPPER{constant}	Program pointer point to last program line

<CONTROL-D>

Function Disable all PLC programs

Syntax ASCII Value 4

Remarks This command disables the execution of all PLC Programs. It acts as setting I6=0 or executing the command **DISPLC.** Normally we use this command to disable all PLC program when the PLC programs are wrongfully executed or some unexpected results have occurred.

<CONTROL-F>

Function Report following error of all motors

Syntax ASCII Value 16

Remarks This command causes the control to report the following error of all the motors to the host computer. The position unit is **counts**. There is a <SPACE> between each reported position.

Examples <CTRL-F>

10 12 -3 58 0 0 0 0

<CONTROL-K>

Function Disable all drivers

Syntax ASCII Value 11

Remarks This command disables all motor drivers. If a driver is disabled, it will not accept either Jog Command or Program Execution Command. The drivers could be re-enabled by the **J/** or **A** command.

<CONTROL-O>

Function Feed hold for all Coordinate System

Syntax ASCII Value 15

Remarks This command causes all Coordinate System motion feed hold. It acts the same as an **H** command applied to all Coordinate System. The feedrate override value will slow down to zero with a slew rate of parameter **Ix55** settings. The **<CONTROL-R> or <CONTROL-S>** command will make the Feedrate Override ramp-up to the original value.

<CONTROL-P>

Function Report positions of all motors

Syntax ASCII Value 16

Remarks This command causes the control to report the positions of all the motors to the host computer. The position unit is **counts**. There is a <SPACE> between each reported position.

Examples <CTRL-P>

1000 1250 -324 5890 0 0 0 0

<CONTROL-Q>

Function Report mechanical positions of all motors

Syntax ASCII Value 17

Remarks This command causes the control to report the mechanical positions of all the motors to the host computer. The **mechanical positions** are the distances between motors’ actual positions and its mechanical home. The position unit is **counts**. There is a <SPACE> between each reported position.

Examples <CTRL-P>

1000 1250 -324 5890 0 0 0 0

M164

-1000

<CTRL-Q>

0 1250 –324 5890 0 0 0 0

<CONTROL-V>

Function Report the actual velocities of all motors.

Syntax ASCII Value 22

Remarks This command causes the control to report the actual velocities of all the motors to the host computer. The velocities unit is counts/msec. There is a <SPACE> between each reported velocity.

Examples <CTRL-V>

1000 1250 -250 0 0 0 0 0

<CONTROL-X>

Function Terminate the message reporting

Syntax ASCII Value 24

Remarks The command clears the message-reporting buffer, and terminates all the message reporting.

@{command}

Function Specify an Online Command

Syntax @{command}

Remarks Any command prompted with '@' character will be considered as an Online command. When a Program Buffer is opened, this command could be used if we want to have an Online command to get immediate messages or actions.

Examples

```
OPEN PROG1
@JP             ;  Program point to Program begin
@#1J+           ; Jog first motor
@#1J/           ; First motor Jog Stop
LEARN(1)        ; Read the 1st motor position.
```

#

Function Report the currently addressed motor number

Syntax #

Remarks This command causes the control to report the currently addressed motor number to the host computer. We should address the motors before we use the Online <For all motor> commands. The power on default-addressed motor is motor #1.

Examples

```
#
1
#3
#
3
```

#{constant}

Function Address a Motor

Syntax #{constant}

{constant}representing the addressed motor number, ranging 1~4

Remarks This command will address a motor to accept the future motor command. We should address the motors before we use the Online <For all motor> command. The power on default-addressed motor is motor #1.

Examples

```
#1J+        ; Motor #1 Jog +
#2J+        ; Motor #1 Jog +
J/          ; Motor #2 Jog Stop
#1J/        ; Motor #1 Jog Stop
```

#{constant}*

Function Force all command source to address a motor

Syntax #{constant}*

{constant}representing the addressed motor number, ranging 1~4

Remarks Different command source (Com1, Com2, PLC) will not affect motor addressing each other. This command #{constant}* will force all the command source address at this {constant} motor.

#{constant}->

Function Report the specified motor's coordinate system axis definition

Syntax #{constant}->

{constant} represents the specified motor number, ranging 1~4

Remarks This command causes the control to report the current axis definition of the specified motor. The message **0** will be reported if the motor has not been defined to any axis.

Examples #1->

1000X

#2->

0

#{constant}->{axis definition}

Function Assign an axis definition for the specified motor

Syntax #{constant}->{scale factor}{axis}[,{cs}]

{constant} represents the specified motor, Ranging 1~4

{scale factor} is a positive ratio, representing the encoder counts number of each user's unit (axis unit).

{axis} Specifying axis, could be any of X, Y, Z, U, V, W, A, B, C

{cs} represents the specified Coordinate System number, ranging 1-4.

Remarks This command assigns the specified motor to a coordinate system. If the axis definition is zero **0,** it represents the specified motor axis definition is cleared.

If the {coordinate system} has already been specified, the motor will be redefined to a new Coordinate System; no matter what coordinate system it previously defined.

PS: The ratio must be a positive floating-point value, the moving direction could be specified by Parameter **Ix09 bit0**.

Examples

```
&1            ; Define #1 Coordinate System
#1->1000X     ; Assign motor #1 to X axis with scale 1000.
#2->333.333Y  ; Assign motor #2 to Y axis with scale 333.333.
```

```
#3->A,2        ; Assign motor #3 to A axis with scale 1 in. 2nd
               ; Coordinate System
#4->0          ; Clear axis definition for motor #4
```

$$

Function Software Reset

Syntax $$

Remarks This Command resets the controller to the power on status.

$$$

Function Card conditions reset

Syntax $$$

Remarks This Command resets the control to the power-on default conditions. It is just like re-turned on the power. For the software version later than **V1.33 or V1.43,** if a **SAVE** has been done, the information back-upped in Flash Memory will be reloaded to the working memory area.

$$$***

Function Card reset and re-initialization

Syntax $$$***

Remarks This command causes the control to do the following action:

1. Reset all parameters to the Factory settings.
2. Clear all program buffers.

This command allows users to re-program the control. If a control card has troubles when power just turned on or cannot communicate with host computer, please turn SW1 digit 1 to ON position then re-turn on the power. This is to re-initialize the control.

???

Function Report system status

Syntax ???

Remarks This command causes the control to report the system status. Following is the definition:

12	**2**	**2**	**1**	**3**	**4**	**4**	**1**	**1**	**2**
Reserved	**MsgSend**	**Gath**	**GathRov**	**BufOpen**	**Com2Req**	**Com1Req**	**Save**	**HandShake**	**Baud**

Baud	0	115200 bps(COM1)
	1	9600 bps
	2	19200 bps
	3	38400 bps
HandShake	0	Com1 No Handshake Signal(RTS/CTS)
	1	Com1 Needs Hardware Handshake
Saved	0	Under Flash Save Or Not Complete
	1	Flash Save Complete
Cmd1Req		Data In Queue of Com1 Received Buffer
Cmd2Req		Data In Queue of Com2 Received Buffer
BufOpen	0	No Buffer Open
	1	Program Buffer Open
	2	PLC Buffer Open
	3	Rot Buffer Open
	4	Cam Buffer Open
	5	Memory Buffer Open
GathRov	0	Not Rollover
	1	Rollover
Gather	0	No Gathered Data(Delete Buffer)
	1	Stop Gather
	2	Gather Enable
	3	Gather On Going
MsgSend	0	Com1&Com2 No Message Return
	1	Com1 Has Message To Send
	2	Com2 Has Message To Send
	3	Both Com1&Com2 Have Message To Send

??

Function Report the coordinate system status

Syntax ??

Remarks This command causes the control to report the coordinate system status. Following is the definition:

1	**1**	**1**	**1**	**1**	**3**	**1**	**3**	**1**	**1**
Reserved	**MaxAcc**	**Homing**	**InPos**	**DataRdy**	**Norm2**	**AbsR**	**Norm1**	**Rad**	**FixSlew**
2	**1**	**3**	**1**	**3**	**1**	**3**	**1**	**3**	**16**
Postlude	**TMMode**	**Modal**	**Hold**	**RunReq**	**NbBlend**	**Moving**	**Blend**	**Status**	**for PMPANEL**

Status	0	Not Program Run
	1	Program Halted
	2	Program Held
	3	Program_Run/Cam_Enable In Progress
	4	Program Bstart In Progress
	5	Program Step In Progress
	6	PSet In Progress
	7	MUT In Progress
Blend	0	Non_Blended
	1	Blended
Moving	0	Under Calculated Or Not_Motion Execution
	1	Dwell In Progress
	2	RPD, Lin, Cir1, Cir2, Spline, MUT, Delay In Progress (Time Filter)**RPD Always In Non_Blend Mode
	3	Program Home In Progress
	4	Cam Enable
	5	PVT, DNC, Spline(No Time Filter)
NbBlend	0	Next Block Non_Blended
	1	Next Block Blended
RunReq	0	StandBy
	1	Program Request Next Move
	2	Cam Request Next Block
	3	Cam Request Previous Block
Hold	0	Not Hold
	1	Feed Hold In Progress

Modal	0	Null(Not Modal Command)
	1	Rapid
	2	Linear
	3	Cir1
	4	Cir2
	5	Spline
	6	MUT(Not Modal Command)
	7	Delay(Not Modal Command)
TmMode	0	Feed Mode
	1	Tm Mode
PostLude	0	PostLude Disable
	1	PostLude Enable
	2	PostLude In Progress
	3	PostLude Return
FixSlew	0	Fix Ta
	1	Fix Acc. Slew Rate
Rad	0	Vector Assigned (I, J, K)
	1	Radius Assigned (R)
Norm1	0	#1
	1	#2
	2	#3
	3	#4
AbsR	0	Incremental Vector(I, J, K)
	1	Absolute Vector(I, J, K)
Norm2	0	#1
	1	#2
	2	#3
	3	#4
DataRdy	0	NextBlock Not Ready
	1	NextBlock Data Ready
InPos	0	At Least One CSAxis Not InPos
	1	All CSAxis Are InPos
Homing	0	No CSAxis Is In Homing Process
	1	At Least One CSAxis Is In Homing Process
MaxAcc	0	Default Acceleration Not Exceed A_Max(Ix60)
	1	Default Acceleration Exceed A_Max

?

Function Report the motor status

Syntax ??

Remarks This command causes the control to report the motor status. Following is the definition:

14	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**3**	**1**
Reserved	**FEFatal**	**Fault**	**MLimit**	**PLimit**	**InPOs**	**Inc**	**DirCmd**	**Jog**	**Home**

3	**4**
Servo	**Status**

Status	0	Servo Off
	1	Open Loop
	2	In Position(Zero Command Velocity)
	3	Abort In Progress
	4	Jog In Progress
	5	Homing In Progress
	6	Assigned CS Program Run Or Cam Enable
	7	Assigned CS Program Step End
Servo	0	No Servo Change
	1	From Open Loop To Close Loop
	2	From Close Loop To Open Loop
	3	From Close Loop To Close Loop
	4	Servo Off Pset
	5	On Line Servo On Pset
	6	Program Pset
Home	0	Home Not Complete
	1	Home Completed
Jog	0	Jog Continuously
	1	Homing, Wait To Leave Flag Signal
	2	Homing, Wait For Trigger
	3	J=, Wait For Deceleration Point
	4	J=, Taxi To Target In progress
Dir	0	+ Command Direction
	1	– Command Direction
Inc	0	Axis In Absolute Mode
	1	Axis In Incremental Mode

InPos	0	Not In Pos.
	1	In Position
PLimit	0	Positive Limit Not Set
	1	Positive Limit Set
MLimit	0	Negative Limit Not Set
	1	Negative Limit Set
Fault	0	No Driver Fault Not Set
	1	Driver Fault Set
FEFatl	0	No Fatal FE
	1	Fatal FE Set

%

Function Report the current feedrate override value

Syntax %

Remarks This command causes the control to report the feedrate override value of the currently addressed coordinate system.

Examples %

100

%50

%

50

%{constant}

Function Set feedrate override value

Syntax %{constant}

{constant} is a non-negative value representing the new feedrate override value.

Remarks This command is used to set the feedrate override value. The value 100 represents 'real-time' and 0 represents a stop. The feedrate override slew rate is determined by parameter **I7** (V1.3x) or **Ix55** (V1.4x).

Examples %0

%33.333

%150

&

Function Report the currently addressed coordinate system

Syntax &

Remarks This command causes the control to report the number of currently addressed coordinate system. We should address the active coordinate system before any coordinate system related online command is executed. The power on default-addressed coordinate system is #1.

Examples

```
&
1
&3
&
3
```

&{constant}

Function Address a Coordinate System

Syntax &{constant}

{constant} is representing the number of coordinate system to addressed, ranging 1~4

Remarks This command set the currently addressed coordinate system number. We should address the active coordinate system before any coordinate system related online command. The power on default-addressed coordinate system is #1.

Examples:

```
&1B1R      ; #1 Coordinate System to execute Motion Program #1
S          ; #1 Coordinate System to step execute Motion Program
&2B10R     ; #2 Coordinate System to execute Motion Program #10
A          ; #2 Coordinate System to abort Motion Program execution.
```

{axis}={constant}

Function Set the specified axis position.

Syntax {axis}={constant}

{axis} Specifying axis, could be any of X, Y, Z, U, V, W, A, B or C

{constant} is a floating-point value representing the axis value (motor position).

Remarks This command does not cause the specified axis to move; it only assigns a new value to the axis position. It works the same as command **PSET** in motion program.

Examples X=0 ; Set X axis position to 0
Z=-123.5 ; Set Z axis position to -123.5

A

Function Abort Program Execution and Motor move

Syntax A

Remarks This command stops all Program Execution and Motor moves; Motor deceleration is specified by Parameter **I15**. The program cannot continue from the position where it is stopped by this command. We normally use this command only when an emergency happens. For temporary stop the program execution, we may use the **H** command for immediate stop or **S** Command to stop at end of current moving.

A*

Function Abort all Program Execution and motor moves

Syntax ASCII Value 1

Remarks This command stops all Program Execution and Motor moves, The motor deceleration is specified by Parameter **Ix04**. The program cannot continue execution from the position where it has stopped by this command. We normally use this command only when an emergency situation happens. For temporary stopping the program execution, we may use **H** command for immediate stop or **S** Command to stop at end of current moving. The A* command enables all the drivers, which are currently disabled.

B{constant}

Function Select Motion Program number

Syntax B{constant}
{constant} is a non-negative value ranging 1.0~999.9999

Remarks This command sets the motion program pointer to point to a program numbered INT({constant}), the pointed line number is ({constant}-INT({constant}))*10000. If the {constant} is an integer, the pointer will point to the beginning line of program number **{constant}.**

Examples	B1	; Point to Program #1 beginning line.
	B10.35	; Point to Program #10, line number 3500.

PS: B0 causes the program pointer point to the beginning of the rotary buffer. Normally we use it following the command DEF ROT. If the rotary buffer has been opened by the command OPEN ROT, B0 will be interpreted as B-axis motion command. It causes the B-axis back to zero position.

B*

Function Select Motion Program number indirectly by the content of Mx88

Syntax B*

Remarks This command sets the motion program pointer to point to a program numbered INT(Mx88). The pointed line number is (Mx88-INT(Mx88))*10000.

CLEAR

Function Clear currently opened buffer

Syntax CLEAR
CLEARALL

Remarks Normally we use this command after **OPEN {buffer}** command to clear the old contents before downloading a new program. Otherwise, the new downloaded program will be added to follow the last line of previous program.

Examples

```
OPEN PROG 1    ;Open Program #1
CLEAR          ;Clear current contents
……….
CLOSE          ; Close Program #1
```

CLOSE

Function Close the currently opened program buffer

Syntax CLOSE

Remarks If a buffer has been opened by the command **OPEN {buffer}**, should be closed by the command **CLOSE**. All the command entered before the close command will be saved in the buffer for future execution.

For the versions later than V1.33 & V1.43, when the command **CLOSE** is executed, the controller will re-build a checksum value. The checksum value will be verified at each power-on; if the checksum value unmatched, the control will reload data from Flash Memory.

Examples

```
OPEN PROG 1    ; Open Program #1
CLEAR          ; Clear current contents
. . .
CLOSE          ;  Close Program #1
```

CS

Function Report all the axes specification of the motors.

Syntax CS

Remarks This command cause the control to report all the axes specification of the motors, including the information of axis name, unit ratio, and the coordinate system.

Examples

```
CS
#1->400X,1
#2->400Y,1
#3->33.333A,2
#4->0
```

DEFROT

Function Define Rotary Buffer for motion program

Syntax DEFROT[{constant}]

{constant} is a positive integer representing the buffer size with unit of "word".

Remarks This command causes the control to reserve a memory space with {constant} specified size for Rotary Buffer. If the buffer size is not specified, all the memory left will be defined as the Rotary Buffer.

Examples

```
DEF ROT 1000      ; Define a rotary buffer of 1000 words
B0R               ; Program pointer point to the top of the buffer.
                  ; and ready for execution(PROG 0)
OPEN ROT          ; Open the rotary buffer
```

```
LIN X100Y100          ; Execute the step immediately after it is entered.
```

DELROT

Function Delete the Rotary Buffer

Syntax DELROT

Remarks This command deletes the currently reserved rotary buffer. All the reserved buffer memory will be released. If there is a program under execution, the rotary buffer could not be deleted. We have to abort the execution by the command **A** before deleting the rotary buffer.

Examples

```
DEF ROT 1000          ; Define a rotary buffer of 1000 words
DEL ROT               ; Delete the currently defined buffer.
```

DISPLC

Function Disable the specified PLC(s)

Syntax DISPLC{constant}[,{constant}. . .]

{constant} is a positive integer representing PLC number, ranging 0~15

Remarks This command disables a specified PLC program. The PLC program is specified by program number ranging from 0 to 15. The disabled PLC program will stop execution at next scan cycle, not an immediate stop.

Examples

```
DISPLC1
DISPLC1,2,4,5
```

ENAPLC

Function Enable the specified PLC program.

Syntax ENAPLC{constant}[,{constant}. . .]

{constant} is a positive integer representing PLC number, ranging 0~15

Remarks This command enables a specified PLC program. The PLC program is specified by program number, ranging from 0 to 15. The enabled PLC program will start execution at next scan cycle. The change of parameter **I6** will also change the enable condition of a PLC program. The change of **I6** from 0 to 3 will enable all PLC programs, while **I6** from 3 to 0 will disable all PLC programs.

Examples

```
ENAPLC1
ENAPLC1,2,4,5
```

ENACAM

Function Enable the specified Cam program.

Syntax ENAPLC{constant}[,{constant}. . .]

{constant} is a positive integer representing Cam number, ranging 1~15

Remarks This command enables a specified Cam program. The Cam program is specified by program number, ranging from 1 to 15. The enabled Cam program will start execution at next scan cycle. You can use ***A** to stop running Cam program.

Examples ENACAM1

ENACAM1,2,4,5

ENDG

Function Stop data gathering

Syntax ENDG

Remarks This command causes the control to stop data gathering.

GAT

Function Start to do data gathering

Syntax GAT

Remarks This command causes the control to do data gathering according to **I20..28**.

H

Function Program feed hold

Syntax H

Remarks This command causes all Coordinate System motion feed hold. It is the same as the execution of **%0** command for all Coordinate System. The feedrate override value will slow down to zero with a slew rate of parameter **I7** settings. The **R or S** command will make the Feedrate Override back to original value.

HM

Function Motor home searching

Syntax HM

Remarks This command causes a home searching operation. The homing acceleration and deceleration are the same setting as Jog function. (Ix01 and Ix02), The speed setting is at **IX07.** The positive value of Ix07 will run a positive direction home searching while negative settings make negative direction searching. The home searching functions work as following steps.

1. The 'HM' Command received.
2. Start searching per **Ix07** speed and direction settings, **Ix01 & Ix02** acceleration setting.
3. Detect the home flag trigger signals until it matches. (Home flag trigger signals are defined per **Ix09** settings) Decelerate to stop per **Ix01&Ix02** settings then record the triggered position and trigger signals status.
4. Backward to search the trigger condition with low speed until the real trigger point is found.
5. Move to a position where it matches **Ix08** offset value setting.
6. Set the current position as mechanical home position.

The home flag signal is specified by **Ix09** bit0 and bit1:

bit1	bit0	
0	1	; External home flag signal (edge trigger)
1	0	; Encoder C signal (Rising edge trigger)
1	1	; External signal and encoder C signal trigger.

There is an associated bit will be set to 1 after a home searching is finished. The user may check the associated bit condition for searching finish confirm.
We suggest using the **M** variable **Mx45** for the bit test.

Examples HM

```
HM
#1HM#2HM
```

HMZ

Function Do a Zero-Move Homing

Syntax HMZ

Remarks This command will not cause a motor to move, only set the current motor position as mechanical home position.

I{constant}

Function Report the specified I-variable value

Syntax I{constant}[..{constant}]

{constant} is an integer representing I variable number, ranging 0~1023

Remarks This command causes the control to report the specified I-variable value

PS: If a program buffer or rotary buffer has been opened, I{constant} will be recognized as a vector definition of X-axis and stored in the buffer for future execution. (Please refer to Motion Program Format).

Examples

```
I5
2000
I2..4
500
20
20
```

I{constant}=*[?]

Function Reset the specified I-variable value from flash memory(saved value)

Syntax I{constant}[..{constant}]=*[?]

{constant} is an integer representing I variable number, ranging 0~1023

The second {constant} should be greater than the first one; it represents the ending I-variable number.

[?] means to report the value of this I-variable value.

Remarks This command causes the control to reset the specified I-variable value from flash memory. This means that this command returns the saved specified I-variable value.

I{constant}=**[?]

Function Reset the specified I-variable value as the default value

Syntax I{constant}[..{constant}]=**[?]

{constant} is an integer representing I variable number, ranging 0~1023.

The second {constant} should be greater than the first one; it represents the ending I-variable number.

[?] means to report the value of this I-variable value.

Remarks This command causes the control to reset the specified I-variable value as the default value.

I{constant}={expression}[?]

Function Assign The specified I-variable value.

Syntax I{constant}[..{constant}]={expression}[?]

{constant} is an integer representing I variable number, ranging 0~1023

The second {constant} should be greater than the first one; it represents the ending I-variable number.

{expression} is an expression

[?] means to report the value of this I-variable value.

Remarks This command will assign the {expression} value to the {constant} specified I variable. If a program buffer or rotary buffer has been opened, I{constant}={expression} will be stored in the buffer for future execution.

Examples I5=2

I109=1.25 * I108

I102=$17

I104=I103

INIT

Function Cancel all the axes displacement and rotation

Syntax INIT

Remarks The **INIT** command will set the motor zero position as axis position, set the axis rotation angle as zero and set the axis ratio to 1. (This is the power on default condition.)

Examples I159

45

I112

150

INIT

I159

0

I112

0

J+

Function Jog to positive direction.

Syntax J+

Remarks This command causes currently specified motor jog to positive direction. The jog acceleration time and speed are specified by variable **Ix01-Ix03**.

If a motion program is under execution, Jog function will not be performed. We may first stop the execution by an '**S**' command, and then do the Jog command after motor stops. The command **J*** will move the motor back to the position where motion program stop. Then, next **R or S** command could continue the motion program execution.

Examples J+

```
J+
#4J+
#1J+#3J+
```

J-

Function Jog to negative direction.

Syntax J-

Remarks This command causes currently specified motor jog to negative direction. The jog acceleration time and speed are specified by variable **Ix01-Ix03**.

If a motion program is under execution, Jog function will not be performed. We may first stop the execution by an '**S**' command, and then do the Jog command after motor stops. The command **J*** will move the motor back to the position where motion program stop. Then, next **R or S** command could continue the motion program execution.

Examples

```
J-
#1J-
#2J-#4J-
```

J/

Function Jog stop.

Syntax J/

Remarks This command stops a jog moving or enables a motor driver, which is currently disabled.

The deceleration time and speed are specified by variable **Ix01-Ix03**.

If a motion program is under execution, the Jog function will not be performed.

Examples	#1J+	; Motor #1 Jog positive direction
	J/	; Motor #1 stop jog
	K	; Driver Disable
	J/	; Driver Enable

J:

Function Jog a variable specified distance

Syntax J:

Remarks This command jogs a motor with a distance of counts specified in the target memory. The memory location is **L:2E** for #1 motor, **L:2F** for #2 motor and so on. It is suggested using an **M** variable to point to the memory location. That is **M163-> L:2E** for #1 motor, **M263** for #2 motor and so on.

The acceleration time and speed are specified by variable **Ix01-Ix03**.

If a motion program is under execution, Jog function will not be performed. We may first stop the execution by an '**S**' command, then do the Jog command after motor stops. The command **J*** will move the motor back to the position where motion program stop. Then, next **R or S** command could continue the motion program execution.

Examples	M463->**	; Point the M463 to the 4th motor jog destination(Default).
	#4HM	; Motor #4 back home.
	M463=100	;
	#4J:	; Jog the #4 motor to the position of 100 counts
	J:	; Jog the #4 motor to another distance of 100 counts (the motor position becomes 200 counts)

J:{constant}

Function Jog a distance specified by {constant}

Syntax J:{constant}

{constant} is a floating point value representing the distance to jog in count..

Remarks This command jogs a motor with a distance of counts specified by {constant}.

The acceleration/deceleration time and speed are specified by variable **Ix01-Ix03**.

If a motion program is under execution, the Jog function will not be performed. We may first stop the execution by an '**S**' command, and then do the Jog command after motor stops. The command **J*** will move the motor back to the position where motion program stopped. Then, next **R or S** command could continue the motion program execution.

Examples	#1HM	; Motor #1 back HOME
	J:2000	; Jog a distance of 2000 counts
	J:2000	; Jog a distance of 2000 counts (The motor position becomes 2000 counts)

J=

Function Jog to a variable specified absolute position.

Syntax J=

Remarks This command jogs a motor to the target memory specified absolute position. The memory location is **L:2E** for #1 motor, **L:2F** for #2 motor and so on. It is suggested using an **M** variable to point to the memory location. That is **M163-> L:2E** for #1 motor, **M263** for #2 motor and so on.

The acceleration/deceleration time and speed are specified by variable **Ix01-Ix03**.

If a motion program is under execution, the Jog function will not be performed. We may first stop the execution by an '**S**' command, and then do the Jog command after motor stops. The command **J*** will move the motor back to the position where motion program stopped. Then, next **R or S** command could continue the motion program execution.

Examples	M363->**	; Point M363 to the motor #3 target memory location(Default Value).

```
M363=10          ;
#3J=             ; Jog motor #3 to the absolute position 10.
```

J={constant}

Function Jog a motor to a specified position

Syntax J={constant}

{constant} is a floating point value representing the absolute position to jog in count..

Remarks This command jog a motor to an absolute position specified by {constant}. The acceleration/deceleration time and speed are specified by variable **Ix01-Ix03**.

If a motion program is under execution, the Jog function will not be performed. We may first stop the execution by an '**S**' command, then progress the Jog command after motor stops. The command **J*** will move the motor back to the position where motion program stop. This will allow the user to continue their normal program execution by next **R or S** command.

Examples

```
J=0              ; Jog the currently addressed motor back to 0
#4J=5000         ; Motor #4 jog to the position 5000 counts.
#2J=-32000       ; Motor #2 jog to the position -32000 counts.
```

J*

Function Jog the addressed motor back to the motion program stop

Syntax J*

Remarks This command will move the motor back to the position where motion program stop, if the motor has been jog to some other place after a motion program stop. This command will allow users to continue their normal program execution by next **R or S** command.

Examples

```
S                ; Stop a program under execution
#1J=0#2J=0       ; Jog the motors back to zero
#1J*#2J*     ; Jog the motors back to the program stop position
R                ; Continue the Execution
```

K

Function Driver disabled

Syntax K

Remarks This command disables the currently specified motor driver. The Jog commands and program execution commands will not be accepted. We may use the **J/ or A** command to re-enable the specified driver.

Examples

```
K          ; Disable the currently addressed motor driver
#1K        ; Motor #1 driver disable
J/         ; Motor #1 driver enable
#2K#3K     ; Motor #1 & #2 drivers disable
A          ; All motor driver enable
```

LIST

Function Report the program(or cam) contents to be executed

Syntax LIST

Remarks This command causes the control to report the program(or cam) contents to be executed

Examples

```
B1R        ; Program #1 start to execute.
LIST       ; Report the program contents to be executed
Q300=1
```

LISTBUF**{address}[,{length}]**

Function Report the contents of the addressed buffer memory(hex).

Syntax LISTBUF{address}[,{length}]

{address} points to the buffer memory you want to list. It should be 16 bit, ranging from $0~FFFF

{length} means the length of the buffer memory you want to list.

Remarks This command is used to report the contents of the addressed buffer memory .

Examples

```
CLOSE
OPENBUF $2000      :open buffer
LISTBUF$2000,2     ;list the addressed buffer memory
```

38
39

LISTI**{constant}**

Function Report the simple definition of I-variable

Syntax LISTI{constant}[..{constant}]

Remarks This command causes the control to Report the simple definition of I-variable

Examples LISTI120 ; Report the simple definition of I-variable

I120 kp

LISTPE**{constant}[,{length}]**

Function Report the contents of the program(or cam)

Syntax LISTPE{constant}[,{length}]

{constant} means the number of program call. 0 means main program

{length} means the length of the contents you want to list. If this is omitted, it will list all the program content until end of program.

Remarks This command causes the control to report the contents of the program(or cam)

LISTCAM

Function Report the existed cam program numbers or specified cam program contents

Syntax LISTCAM{constant}

{constant} is positive integer representing the number of the cam program to be reported.

Remarks **LISTCAM** command will report the existed cam program numbers and their start and end address.

The **LISTCAM{constant}** will report all the contents of the specified cam program in detail.

Examples LISTCAM ;Report the existed cam program and their start and ;end address.

1 0 29

```
2   108   116
LISTCAM 1          ; Report all program contents of cam program #1
X0Y0
X20Y20
. . .
```

LISTGAT

Function Report the contents of data gathering

Syntax LISTGAT

Remarks This command causes the control report the contents of data gathering if data gathering is finished.

Examples

```
I25=2              ;gather data every 2ms
I20=$8D            ;Source 1,3,4($xD)
                   ;Source 4 floating point number
I21=$A9B           ;Gather Source 1 1msec interrupt counter
I23=$CEB4          ;Gather Source 3  following error of motor#1
I24=$1001          ;Gather Source 4 P1
GAT                ;Start to gather data
ENDG               ;Stop data gathering
LISTGAT            ;Report the content
GATH=[a1    b1   c1   d1
  a2 b2   c2   d2
       …
  an bn   cn   dn
  ];
```

LIST PLC

Function Report the existed PLC program number or specified PLC Program contents

Syntax LIST PLC {constant}

{constant} is positive integer representing the number of the PLC Program to reported, ranging 0~15.

Remarks This command reports the existed PLC program number, start address, end address, and enable status.

The **LIST PLC {constant}** reports all the contents of the specified PLC program in detail.

Examples LIST PLC ; Report the exist PLC program number, start and end address, and enable status.

```
1    120   194   YES
2    195   442   NO
3    443   779  YES
LIST      PLC 3          ; Report all program contents of PLC program #3.
IF(M20=0ANDM180=0)
. . .
ENDI
RET
```

LISTPROG

Function Report the existed motion program numbers or specified motion program contents

Syntax LIST PROG{constant}

{constant} is positive integer representing the number of the motion program to be reported.

Remarks **LISTPROG** command will report the existed motion program numbers and their start and end address.

The **LISTPROG{constant}** will report all the contents of the specified motion program in detail.

Examples LIST PROG ;Report the existed motion program and their start and end address.

```
1    0     29
1000 30    107
2    108   116
LISTPROG 1       ; Report all program contents of motion program #1
WHILE(Q300<=P300)
. . .
ENDW
RET
```

M{expression}

Function Report the specified M variable value

Syntax M{expression}[..{expression}]

{expression} is an integer representing the variable number to be specified or the starting variable number, ranging 0 ~ 1023.

The 2nd {expression} representing the end number of variable to be reported, its value should be larger than the 1st one.

Remarks This command causes the control to report the specified M variable value.

If the variable is a signed number, the reported value is in decimal.

If the variable is an unsigned number, the reported value is in hexadecimal.

Examples M0

623316

M165

0

M1..3

0

1

1

PS: If the program buffer or rotary buffer has been opened, **M{expression}** will be recognized as an M-code and stored in the program for future execution. (Refer to the motion program command specification.)

M{expression}={expression}

Function Set the specified M variable value

Syntax M{expression}[..{expression}]={expression}{?}

{expression} is an integer representing the variable number to be specified or the starting variable number, ranging 0 ~ 1023.

The 2nd {expression} representing the end number of variable to be set, its value should be larger than the 1st one.

The 3rd {expression} representing the value to be set.

[?] means to report the value of this M variable.

Remarks This command sets the specified M variable value.

If the program buffer or rotary buffer has been opened,

M{expression}={expression} will be stored in the program for future execution.

(Refer to the motion program command specification.)

Examples M1=1

M9=M9 & $20

M102=-16384

M1..8=0

M(P100)=M162+M164

M**{expression}**->

Function Report M variable definition

Syntax M{expression}[..{expression}]->

{expression} is an integer representing the variable number to be specified or the starting variable number, ranging 0 ~ 1023.

The 2nd {expression} representing the end number of variable to be reported, its value should be larger than the 1st one.

Remarks This command causes the control to report the specified M variable definition.

Examples M1->

14,16,1

M180->

13F,S

M191->

L:B6

M2..M4->

14,17,1

14,18,1

14,19,1

M{expression}->*

Function Assign M{expression} as a normal variable.

Syntax M{expression}[..{expression}]->*[{Type}][?]

{expression} is an integer representing the variable number to be specified or the starting variable number, ranging 0 ~ 1023.

The 2nd {expression} representing the end number of variable to be assigned, its value should be larger than the 1st one.

{Type} can be L/S/U. Default is S(See M{expression}-> addr[,start][,width][,s])

[?] means to report the value of this M variable.

Remarks This command cancels the M variable definition to be used as a normal variable.

Examples

```
M90..91->*          ; Cancel M90,M91 definition
M90..91->           ; Report M90,M91 definition
*
*
M90..91             ; Report M90,M91 contents
1234
567
```

M**{expression}->****

Function Reset the specified M variable as the default value

Syntax M{expression}[..{expression}]->**[?]

{expression} is an integer representing I variable number, ranging 0~1023

The 2nd {expression} representing the end number of variable to be assigned, its value should be larger than the 1st one.

[?] means to report the value of this M variable.

Remarks This command causes the control to reset the specified M variable as the default value.

M{expression}->addr[,start][,width][,s]

Function Set the specified M variable definition

Syntax M{expression}->addr[,start][,width][,s][?]

{expression} is an integer representing the M variable number to be specified, ranging 0 ~ 1023.

addr is the address in hexadecimal that the M variable pointed, ranging 0000~ FFFF

[start] is the start bit of the pointed address, ranging 0~31.

[width] is the bits length of pointed address, ranging 1~31.

But, [start] + [width] < 32.

[s] Representing a signed value

If [s] is not specified, it represents an unsigned value.

[?] means to report the value of this M variable definition.

Remarks This command will point a specified M variable to the normal data area. The addressed contents is an integer, If [s] is not specified, it represents an unsigned value, while the [s] specifies a signed value.

Examples

```
M90->1049,0,8,S      ;  Define M90 address and data format
M91->1049,0,10       ; Define M91 address and data format
M91=255
M90..91              ; Report M90~M91 value
-1                   ; Report M90 value in decimal
FF                   ; Report M91 value in hexadecimal
```

PS: It is allowed that multiple M variables point to the same address. If the assigned data formats are different, the reported value will be different.

M{expression}->L : addr

Function Set the M variable definition

Syntax M{expression}-> L:addr[?]

{expression} is an integer representing the M variable number to be specified, ranging 0 ~ 1023.

addr is the address in hexadecimal that the M variable pointed, ranging 0000~ FFFF

[?] means to report the value of this M variable definition.

Remarks This command points the specified M variable to the data area.

The addressed data is a floating-point value.

Examples

```
M90->L:1049       ; Define M90 address and data format.
```

M{expression}->I: addr[,start][,width][,s]

Function Set the M variable definition

Syntax M{expression}-> I:addr[,start][,width][,s][?]

{expression} is an integer representing the M variable number to be specified, ranging 0 ~ 1023.

addr is the address in hexadecimal that the M variable pointed, ranging 0000~ FFFF

[start] is the start bit of the pointed address, ranging 0~31.

[width] is the bits length of pointed address, ranging 1~31.
But, [start] + [width] < 32.
[s] Representing a signed value
If [s] is not specified, it represents an unsigned value.
[?] means to report the value of this M variable definition.

Remarks This command will point a specified M variable to the I/O address area. The addressed contents is an integer, If [s] is not specified, it represent an unsigned value, while the [s] specifies a signed value.

Examples

```
M11->I:FF41,16,1      ; DefineM11 pointed I/O address and data format.
M19->I:FF41,16,8      ; DefineM19 pointed I/O address and data format.
```

O{constant} For Voltage Command(UTCx00V)

Function Set the output voltage

Syntax O{constant}

{constant} is an integer representing the ratio of **Ix30** voltage, ranging -100~100.

Remarks This command causes the control send out the voltage.

Examples

```
I130=20480   ;6.25V
O10              ; The addressed axes send out 10/100*6.25 volts .
O-20             ; The addressed axes send out –20/100*6.25 volts.
```

O{constant} For Pulse Command(UTCx00P)

Function Set the output pulse frequency in the unit Pulses/msec

Syntax O{constant}

{constant} is an integer representing the pulse number per msec, ranging -250~250.

Remarks This command causes the control send out the setting number of pulse in one msec.

Examples

```
O10              ; The addressed axes send out 10 pulses .
O-20             ; The addressed axes send out -20 pulses.
```

OPENBUF**{address}**

Function Open the addressed buffer memory.

Syntax OPENBUF{address}

{address} point to the buffer memory. It should be 16 bit, ranging from $0~FFFF

Remarks This command is used for opening buffer memory. The buffer must be closed by the command **CLOSE** when the execution is finished.

No other program buffer or rotary buffer could be opened when buffer memory is currently opened. It is suggested to execute a **CLOSE** command before we want to open a buffer.

Examples

```
CLOSE
OPENBUF $2000
…
CLOSE
```

OPEN CAM

Function Open a Cam program buffer.

Syntax OPEN PROG {constant}

{constant} is a positive integer representing the cam program number, ranging 0~15.

Remarks This command is used for opening a cam program buffer. We may edit any legal cam program lines after a buffer has been opened. The buffer must be closed by the command **CLOSE** when the editing is finished.

No other program buffer or rotary buffer could be opened when a cam program buffer is currently opened. It is suggested to execute a **CLOSE** command before we want to open a buffer.

Opening a cam program will terminate the execution if it is currently executed; however, it can re-start after the buffer is closed.

The cam program begins with N{constant}. Constant ranges 0 to 999. The content can be any motion action or mold command.

Examples

```
CLOSE
OPEN CAM 1
CLEAR
N100        X10 Y20
```

CLOSE

OPEN PLC

Function Open a PLC program buffer.

Syntax OPEN PLC {constant}

{constant} is a positive integer representing the PLC program number, ranging 0~15.

Remarks This command is used for opening a PLC program buffer. We may edit any legal program lines after a buffer has been opened. The buffer must be closed by the command **CLOSE** when the editing is finished.

No other program buffer or rotary buffer could be opened when a PLC program buffer is currently opened. It is suggested executing a **CLOSE** command before we want to open a buffer.

Opening a PLC program will terminate the execution if it is currently executed. And, it will automatically re-start after the buffer is closed.

Examples

```
CLOSE
OPEN PLC 7
CLEAR
IF(M11=1)
…
CLOSE
```

OPEN PROG

Function Open a motion program buffer.

Syntax OPEN PROG {constant}

{constant} is a positive integer representing the motion program number, ranging 0~999.

Remarks This command is used for opening a motion program buffer. We may edit any legal program lines after a buffer has been opened. The buffer must be closed by the command **CLOSE** when the editing is finished.

No other program buffer or rotary buffer could be opened when a motion program buffer is currently opened. It is suggested to execute a **CLOSE** command before we want to open a buffer.

Opening a motion program will terminate the execution if it is currently executed; however, it can re-start after the buffer is closed.

Examples

```
CLOSE
OPEN PROG 1
CLEAR
X10 Y20 …
CLOSE
B1
R
```

OPEN ROT

Function Open the rotary buffer.

Syntax OPEN ROT

Remarks This command is used for opening rotary buffer. We may edit any legal program lines for execution after a buffer has been opened. The buffer must be closed by the command **CLOSE** when the execution is finished. (The branch and loop commands are not allowed in the rotary buffer.)

No other program buffer or rotary buffer could be opened when rotary buffer is currently opened. It is suggested to execute a **CLOSE** command before we want to open a buffer.

Examples

```
CLOSE
DEF ROT 1000
B0R
OPEN ROT
X10 Y20
…
CLOSE
```

P

Function Report the position of a specified motor

Syntax P

Remarks This command causes the control to report the position of currently specified motors.

Examples

```
P                   ; Report the position of currently addressed motor.
1995
```

```
#1P          ; Report position of Motor #1
-5
#2P#4P       ; Report position of Motor #2 & #4
9998
10002
```

P{expression}

Function Report specified P variable value

Syntax P{expression}[..{expression}]

{expression} is a positive integer representing the P variable, ranging 0~(I17-1).

The 2nd {expression} representing the end number of variable to be assigned, its value should be larger than the 1st one.

Remarks This command causes the control to report the {expression} specified P variable value or the P variable range.

Examples

```
P1
25
P1005
3.44
P100..102
17.5
-373
0.005
```

P{expression}={expression}

Function Set the specified P variable value

Syntax P{expression}[..{expression}]={expression}[?]

{expression} is a positive integer representing the P variable, ranging 0~(I17-1).

The 2nd {expression} representing the end number of variable to be assigned, its value should be larger than the 1st one.

{expression} is the value to be assigned.

[?] means to report the value of this P variable.

Remarks This command sets the specified P variable value.

If the program buffer or rotary buffer has been opened, the

P{constant}={expression} will be stored in the program for future execution. (Refer to the motion program command specification.)

Examples P1=1

P75=P329 + P10

P102=-16384

P100..199=0

P(P100)=M191＊SIN(M167)

*******Block move: P100..P199=P120 means to move from P120..219 to P100..199

PC

Function Report program pointer number.

Syntax PC

Remarks This command causes the control to report current program pointer value.

Examples PC

10

B123

PC

123

PE**{constant}**

Function Report the current program(or Cam) line number.

Syntax PE{constant}

{constant} means the number of program call. 0 means main program.

Remarks This command causes the control to report the current program(or Cam) line number.

PL**{constant}**

Function Report the to be executed program(or Cam) line number.

Syntax PL{constant}

{constant} means the number of program call. 0 means main program.

Remarks This command causes the control to report the to be executed program(or Cam) line number.

PR

Function Report the lines count in the rotary buffer that have not been executed.

Syntax PR

Remarks This command causes the control to report the lines count in the rotary buffer that have not been executed.

PWD

Function Set the control card password

Syntax PWD={string}

{string}a string of maximum 16 characters

Remarks This command allows users to setup their own passwords to protect their programs. Once the password is been set, each time when uploading a program to the host computer, the password must first be re-entered. Otherwise, the control will report a error message. The password can be changed by next password setup, or use **PWD=** to clear the password setting. If the password is forgotten, the only way to clear the password is using the reset command ($$$***) to clear the entire program memory.

Examples PWD=MicroTrend

PWD=~!@#$%^&*()

Q{expression}

Function Report specified Q variable value

Syntax Q{expression}[..{expression}]

{expression} is a positive integer representing the Q variable, ranging 0~1023

The 2nd {expression} representing the end number of variable to be assigned, its value should be larger than the 1st one.

Remarks This command causes the control to report the {expression} specified Q variable value or the Q variable range.

Examples Q10

35

Q255

-3.4578
Q101..103
0
98.5
-0.0333

Q{expression}={expression}

Function Set the specified Q variable value

Syntax Q{expression}[..{expression}]={expression}[?]

{expression} is a positive integer representing the Q variable, ranging 0~1023

The 2nd {expression} representing the end number of variable to be assigned, its value should be larger than the 1st one.

{expression} is the value to be assigned.

[?] means to report the value of this Q variable.

Remarks This command sets the specified Q variable value.

If the program buffer or rotary buffer has been opened, Q{constant}={expression} will be stored in the program for future execution. (Refer to the motion program command specification.)

Examples Q100=2.5
Q1..10=0
Q(P100)=INT(M162/M191＊100)＊0.01

R

Function Execute the Motion Program.

Syntax R

Remarks This command runs a motion program according to the motion program pointer, the pointer will back to the beginning after all lines are finished.

The **H or A** command could terminate the program execution.

Examples

B1R	; Program pointer point to Program #1 and Execute.
H	; Terminate the Program execution
R	; Execute the Program again
A	; Abort the Program Execution

R*

Function Start to execute Motion Programs for all active Coordinate Systems.

Syntax R*

Remarks This command causes the control to start executing the Motion Programs from the pointed program line for all active Coordinate Systems. The execution will stop at each program end, and then the pointer will point to first line of the program.

RD{type}{address}[,{length}]

Function Report the contents of the addressed buffer memory.

Syntax RD{D(dec)/H(hex)/I(input)/L(float)}{address}[,{length}]

{address} points to the buffer memory you want to list. It should be 16 bit, ranging from $0~FFFF

{length} means the length of the buffer memory you want to list.

Remarks This command is used to report the contents of the addressed buffer memory.

Examples

```
CLOSE
OPENBUF $2000       :open buffer
LISTBUF$2000,2      ;list the addressed buffer memory
38
39
```

S

Function Execute the motion program step by step.

Syntax S

Remarks This command will run a motion program step by step according to the motion program pointer, the pointer will back to the beginning of that program after all lines are finished. If a **BSTART** command is executed, it will continue to execute until a **BSTOP** command.

S*

Function Step execution of Motion Programs for all active Coordinate Systems.

Syntax S*

Remarks This command causes the control to execute one step of the Motion Program from the pointed line for all active Coordinate Systems. Then, the pointer will point to next line of each Motion Program. If a **BSTART** command is executed the whole block between **BSTART** and **BSTOP** will be executed as one step.

SAVE

Function Save all the contents from static RAM to FLASH ROM.

Syntax SAVE

Remarks This command saves all the contents, including Variables, Motion Programs, PLC Programs and so on, to a non-volatile Flash Memory. This process could protect all the important program contents or data from been destroyed. If the contents in the SRAM is unexpectedly lost, we may make the control to recover from the damage by using the **$$$** Online Command to re-load the contents from the Flash Memory.

Each time when the power is turned on, the control runs a checksum process. If the checksum result shows an unmatched situation, the control will automatically do the re-load. The re-load process takes about 10 seconds; the controller does not response to any command before it is finished. Please do not turn the power off when this reload is processing. Otherwise, the reload process cannot be completed.

The Watch Dog LED is turned on when the **SAVE** command is executing. Please do not run this command when any motor is running, no matter running a motion program or just jogging the motor.

SIZE

Function Report the available buffer size.

Syntax SIZE

Remarks This command reports the available buffer size, with unit (word).

Examples SIZE

43188

V

Function Report the speed of specified motors.

Syntax V

Remarks This command causes the control to report the speed of specified motors, with unit counts / msec.

Examples

```
V              ; Report the addressed motor speed.
0
#1V            ; Report the speed of Motor #1.
10
#2V#4V         ; Report the speed of Motor #2 & #4.
123
-100
```

VER

Function Report the control software version.

Syntax VER

Remarks This command reports the current software version. The controller checks version each time when the power is turned on. If the software version is changed, the control will reset automatically. (The same as executing a **$$$***** command.)

Examples

```
VER      ; Report the control software version..
V3.0
```

Motion Command Format

I. Summary of Buffer Command:

1. Motion Action Command

{axis}{data}[{axis}{data}...]	Motor linear motion Example:X100Y100Z200
{axis}{data}[{axis}{data}..] **[{vector}{data}..]**	Motor arc motion Example:X100Y100Z200 I500J300
DWELL{data}	Program stop for a specified time
DELAY{data}	The program execution will delay for a specified time
HM[({axis}[,{axis}…])]	Home searching

2. Motion Mode Command

LIN	Linear motion mode
RPD	Rapid motion mode
CIR1	Clockwise arc motion mode
CIR2	Counterclockwise arc motion mode
MUT	Move Until Trigger
POSTLUDE	Automatic subroutine call after a programming move
SPLINE	Spline move mode

3. Axis Command

ABS[({axis}[,{axis},...])]	Absolute axis mode
INC[({axis}[,{axis},...])]	Incremental move mode
PSET{axis}{data}[{axis}{data}...]	Define the current axis position
NORMAL{vector}{data}	Specifies the rotation plane for a circular interpolation move
ADIS{axis}{data}[{axis}{data}...]	Absolute displacement for the specified axes
IDIS{axis}{data}[{axis}{data}...]	Set the incremental displacement of axes
AROT X{data}	Specifies the incremental rotation angle of a coordinate plane
IROT X{data}	Specifies the absolute rotation angle of a coordinate plane
ASCL{axis}{data}[{axis}{data}...]	Specifies the absolute scaling of axes
ISCL{axis}{data}[{axis}{data}...]	Specifies the incremental scaling of axes
INIT	Cancel all the axes displacement and rotation

R{data}	Set circle radius

4. Motion Related Command

TM{data}	Set move time
F{data}	Specifying motor feed rate
TA{data}	Set the total acceleration time
TS{data}	Set the percentage of S-curve acceleration time

5. Setting Variable Command

I{expression}={expression}	Set I variable value
P{expression}={expression}	Set P variable value
Q{expression}={expression}	Set Q variable value
M{expression}={expression}	Set M variable value

6. Logical Control Command

N{constant}	Program line number
GOTO{data}	Unconditional jump Without Return
GOSUB{data}[{letter}{axis}...]	Unconditional jump With Return
CALL{data}[.{data}][{letter}{axis}...]	Subroutine Call
RET	Return from subroutine
READ({letter}[,{letter}...])	Read the variable for subroutine
IF({condition}){action}	The block-start-point of a conditional branch
ELSE{action}	Start fault condition branch
ENDIF	End of conditional block
WHILE({condition}){action}	Start point of a conditional loop
ENDWHILE	End of conditional loop
G{data}	NC program G-Code
M{data}	NC program M-Code

7. Miscellaneous Command

CMD"{command}"	Run an immediate command
CMD^{letter}	
SEND”{message}”	Send message to computer(PC)
SEND{C1/C2},{ " variable=}",{ variable}	Send message controller to controller
DISP[{constant}] ,"{message}"	Shows messages on LCD panel
DISP{constant},{constant},{variable}	
DISP{constant},{len},{PE/LIST}[0]	
ENAPLC{constant}[,{constant}...]	Enable specified PLC
DISPLC{constant}[,{constant}...]	Disable specified PLC(s)

8. Program Pointer Control Command

INS	Set the program editing to the

	insertion mode
OVR	Set the program editing process as the write over mode
DEL	Delete program command line
LEARN[{axis}[,{axis}. . .]	Motor position teaching

{axis}{data}[{axis}{data}. . .]

Function Motor linear motion setting

Type Motion program

Syntax {axis}{data}[{axis}{data}. . .]

{axis} Specifying axis, could be any choice of X, Y, Z, U, V, W, A, B, or C.

{data} could be a constant (no parentheses) or an expression (in parentheses) representing the end position in absolute mode or the distance of moving in incremental mode.

[{axis}{data}. . .] is an option, multi-axes could be specified to move simultaneously.

Remarks This is a basic motion command, including an axis label and its associated value. The value presents a position in absolute mode, or a distance in incremental mode. The value is user scaled by axis definition command.

Example

```
X10000
X(P1+P2)
X1000Y1000
Y(Q100+50)Z35W(P100)
X(Q1*SIN(Q2/Q3))W500
```

{axis}{data}[{axis}{data}. . .]{vector}{data}[{vector}{data}. . .]

Function: Motor arc motion setting

Type: Motion program

Syntax: {axis}{data}[{axis}{data}. . .]{vector}{data}[{vector}{data}. . .]

{axis} Specifying axis, could be any choice of X, Y, Z, U, V, W, A, B, or C.

{data} could be a constant (no parentheses) or an expression (in parentheses) representing the end position in absolute mode or the distance of moving in incremental mode.

[{axis}{data}. . .] is an option, specifying the axes moving simultaneously.

{vector} is a character (I,J or K) specifying a vector component ,parallel to the X,Y or Z axis respectively, to the center of arc.

{data}specifies the magnitude of the vector component.

[{vector}{data}. . .] is specifying multiple vector components.

Remarks For an arc motion setting, we should specify the endpoint (Same as in linear motion) and the vector to the arc center. The direction of an arc motion is specified by the commands **CIR1**(CW) or **CIR2**(CCW).

Example
```
X5000Y3000I1000J1000
X(P101)Y(P102)I(P201)J(P202)
X10I5
X10Y20Z5I5J5
J10
```

ABS

Function: Absolute move mode

Type: Motion program

Syntax: ABS[({axis}[,{axis}. . .])]

{axis} Specifying axis, could be any of X, Y, Z, U, V, W, A, B or C

Remarks The command **ABS** with no axes specified will cause all axes set to absolute mode. The **ABS** command with axes specified will cause only the assigned axes to be set to absolute mode; all others will keep as their original mode.

Example
```
ABS
ABS(X,Y,Z)
ABS(X,Y,Z,A,B,C,U,V,W)
```

ADIS

Function: Absolute displacement for the specified axes

Type: Motion program

Syntax: ADIS

{axis} Specifying axis, could be any of X, Y, Z, U, V, W, A, B or C.

{data} could be a floating point constant or an expression presenting the displacement distance in user units for the axis.

Remarks This command set the specified value as the offset value of each axis. That is X' = X{data}, Y' = Y{data}, Z' = Z{data}, W' = W{data}. The axes not specified will keep their original offset value. If no axis is specified, the current motor positions will be specified as their offset values, it act as **PSET**X0Y0Z0W0 but will not cancel Blended Move function. This command will not cause any movement; only cause the future movements to take the offset value into account.

Example

```
ADISX10Y5Z3.5W0    ;New coordinate center (X', Y', Z', W') = (10, 5,
                     3.5, 0)
ADISX20Y-5         ;New coordinate center (X', Y', Z', W') = (20,
                   -5, 3.5, 0) ADISX(P1)Z(P2+3)
ADIS               ;New coordinate center will be the current
                   motor positions.
```

AND

Function: Logic condition AND

Type: Motion program and PLC PROGRAM

Syntax: AND {condition}

{condition} is a simple or compound condition.

Remarks This command use only in **IF** or **WHILE** command; it performs the Boolean operator logically.

Example

```
IF(M11 = 1 AND P11 != 1)
CMD"R"
P11=1
ENDIF
```

AROT

Function: Specifies the absolute rotation angle of a coordinate plane

Type: Motion program

Syntax: AROT X{data}

{data} could be a floating point constant or an expression presenting the absolute rotation angle in the unit of an angle.

Remarks This command will load the specified X-axis value as coordinate absolute rotation angle. That is θ_{rot} = X{data}. If specified axis is not X-axis, the specified value will be given up. This command will not cause any movement; only cause the future movements to take the rotation angle into account.

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} \cos(\theta_{rot}) & \sin(\theta_{rot}) \\ -\sin(\theta_{rot}) & \cos(\theta_{rot}) \end{bmatrix} \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Example

```
AROTX45     ; 45 degree rotation
AROTX(ASIN(P2))
```

ASCL

Function: Specifies the absolute scaling of axes

Type: Motion program

Syntax: ASCL{axis}{data}[{axis}{data}. . .]

{axis} Specifying axis, could be any of X, Y, Z, U, V, W, A, B or C

{data} could be a floating point constant or an expression presenting the absolute scale of axis in the unit of ratio.

Remarks This command will load the specified axes value as coordinate axes ratio. That is Xratio = X{data}, Yratio = Y{data}, Zratio = Z{data}, Wratio = W{data}. This command will not cause any movement, only cause the future movements to take the ratio into account.

Example

```
ASCLX2          ; X axis multiplier is 2
ASCLX1.5        ; X axis multiplier is 1.5
ASCLZ0.5        ; Z axis multiplier is0.5
ASCLW1          ; W axis multiplier is1
ASCLX(P1)Y(P2)Z(P3)W(P4)
```

BSTART

Function: Program Block Start

Type: Motion program

Syntax: BSTART

Remarks In the single step execution, all the motion commands after **BSTART** will be continuously executed until the **BSTOP** command is found.

Example

```
X100Y100
BSTART
Z(P100)
Z(-P100)
BSTOP
X0Y0Z0
```

BSTOP

Function: Program Block End

Type: Motion program

Syntax: BSTOP

Remarks This command is used as a pair with **BSTART**. In the single step execution, all the motion commands after **BSTART** will be continuously executed until next **BSTOP** command is found.

Example

```
X100Y100
BSTART
Z(P100)
Z(-P100)
BSTOP
X0Y0Z0
```

CALL

Function: Subroutine Call

Type: Motion program

Syntax: CALL{data}[{letter}{data}. . .]

{data} is a floating point constant, the integer part representing the motion program number to be called, the fractional part multiplied by 10000 will representing the line number of that motion program being called.

{letter} is an English alphabet (except G, M, N, T) used for passing a parameter to the subroutine being called.

{data}is the specified parameter value

Remarks Motion programs are labeled as **PROGn** (n represents the program number), total 256 programs could be stored in memory. Each program under execution could call other programs as a subroutine. Maximum 16 subroutine loops (including G, M code) could be nested.

If the **CALL** command followed by an integer, the subroutine will be started at top of the program being called. The subroutine will be ended by the **RET** command or program end. Following end of a subroutine, the control will jump back to the next line of the previous **CALL** command.

If the number following the **CALL** command contains fractional part, the subroutine will start at the line number specified by the fractional part multiply by 10000.

If the **CALL** command followed by an English alphabet except G, M, N or T, the alphabet is used for passing a parameter to the subroutine being called. And there should be a **READ** command at the first line of the subroutine, the Alphabet will be recognized as a variable and its value will be stored in the associate Q variable for later use. (Refer to **READ command)**

Example

```
CALL500             ;to PROG500 at the top
CALL500.1           ;to PROG500 label N10000
CALL500.12345       ;to PROG500 label N12345
CALL500 X10Y20`     ;to PROG500 passing X and Y
```

CIR1

Function: Clockwise arc motion mode

Type: Motion program

Syntax: CIR1

Remarks This is a mold command setting the current circular motion mode as clockwise Arc Motion Mode until next mode setting command found.

The arc motion only performs on the plane of 1st and 2nd axes. The radius (**R**) or the vector component to the center of arc **(I, J)** should also been specified. Otherwise, the statements will be recognized as a linear motion.

Example

```
LIN
X10Y10F200
CIR1
X20Y20I10
X25Y15J-5
X10Y10R10
LIN
X0Y0
```

CIR2

Function: Counterclockwise arc motion mode

Type: Motion program

Syntax: CIR2

Remarks This is a mold command setting the current motion mode as counterclockwise Arc Motion Mode until next mode setting command.

The arc motion only performs on the plane of 1st and 2nd axes. The radius (**R**) or the vector component to the center of arc **(I, J)** should also been specified. Otherwise, the statements will be recognized as a linear motion.

Example

```
LIN
X10Y10F200
CIR2
X20Y20I10
X25Y15J-5
X10Y10R-10
LIN
X0Y0
```

CMD

Function: Run an immediate command

Type: Motion program, PLC Program

Syntax: CMD"{command}"

CMD^{letter}

{command} is an any legal immediate command.

^{letter} is recognized as an on-line control code command.

Remarks In the motion program, we use the **CMD** command to perform an immediate instruction. The immediate command could be a string or a control code type command prompted by '^'. These commands will not report any message to computer. The **SEND** command can be used for sending messages back to computer.

Example

```
CMD"#1J+"
CMD"#4HM"
```

CMD"A*"

CMD^D

- Do not use CMD continuously in motion or PLC program

DELAY

Function: The program execution will delay for a specified time

Type: Motion program

Syntax: DELAY{data}

{data} is a floating point constant or an expression representing the delay time in msec.

Remarks The control will keep the command position of all axes in the coordinate system constant for the time specified by {data}.

There are 3 differences between **DELAY** and **DWELL**:

(1). In Blended Move, the **DELAY** timing includes the deceleration time of the previous motion but **DWELL** does not.

(2). **DELAY** is a function of Time Base (% value), while **DWELL** performs a fixed time.

(3). In Blended Move, **DELAY** command will perform next step calculation but **DWELL** will not.

Example DELAY2000

DELAY(P1+200)

DISP

Function: Shows messages on LCD pannel

Type: Motion program or PLC Program

Syntax: DISP{const},"message"

DISP{const},{const}.{const}, {data}

{const} is the start location of the LCD panel, an integer of 0~79

"message" The message string for display

{const}.{const}: The 1st constant is an integer 1~80, specifying the total inter digits, the 2nd constant is also an integer 0~15, specifying the total fractional digits.

{data} could be a variable or an expression.

Remarks This command is used for displaying a formatted value or a string on a standard 40×2 LCD panel. The start location 0 is at left top, 39 at right top, 40 at left bottom and 79 at right bottom. The total digits of the value for both integer and fractional part can be specified. The unspecified portion will be filled with blank.

Example

```
DISP0,"Hello World!"
DISP40, "X Pos:          mm"
DISP10,6.2,P100
DISP10,6.4,10*SIN(Q2)+Q1/3
```

DISPLC

Function: Disable specified PLC(s)

Type: Motion program, PLC PROGRAM

Syntax: DISPLC{const}[,{const}. . .]

{const} bis PLC program number, an integer ranging 0~15

Remarks This command is used for disable a {const} specified PLC program. The program number is between 0 and 15. The disabled PLC program will stop at next scan instead of an immediate stop.

Example

```
DISPLC1
DISPLC1,2,4,5
```

DWELL

Function: Program stop for a specified time

Type: Motion program

Syntax: DWELL{data}

DWE{data}

{data} is a floating point constant or expression representing the delay time in msec.

Remarks The control will stop the command position of all axes in the coordinate system for the time specified by {data}. The **DWELL** command will stop the Blended Motion even a zero time dwell. (**DWELL 0**). The differences between **Dwell** and **Delay** commands are described in the **Delay** command section.

Example

```
DWELL2000
DWELL(P1+P2)
```

DWE0

ELSE

Function: Start fault condition branch

Type: Motion program, PLC Program

Syntax: ELSE

Remarks This instruction must be matched with an **IF** instruction (**ELSE** requires a preceding **IF**, **IF** does not require a following **ELSE**.) It is followed by the instruction to be executed upon a false **IF** condition.

Example

```
IF(M11 = 1)
X(P100)
ELSE
X(P200)
ENDIF
```

ENAPLC

Function: Enable the specified PLC

Type: Motion program, PLC PROGRAM

Syntax: ENAPLC{const}[,{const}. . .]

{const} is PLC program number, an integer ranging 0~15

Remarks This command is used for enable a **{const}** specified PLC program. The program number is between 0 and 15. The enabled PLC program will start at next scanning cycle. Changing **I 6** will also change the PLC enable status. If **I 6** value is changed from 0 to 3, will enable all the PLC programs, while changing value from 3 to 0 will disable all the PLC programs.

Example

```
ENAPLC1
ENAPLC1,2,4,5
```

ENDIF

Function: End of conditional block

Type: Motion program, PLC Program

Syntax: ENDIF

ENDI

Remarks Each of the conditional blocks started with **IF** should have an **ENDIF** instruction for closing. The program will generate an error message when it is **CLOSE**, if the **IF** and **ENDIF** are not matched with pair.

Example

```
IF(M11 = 1)
X(P100)
ELSE
X(P200)
ENDIF
```

ENDWHILE

Function: End of conditional loop

Type: Motion program, PLC Program

Syntax: ENDWHILE

ENDW

Remarks Each of the conditional loops started with **WHILE** should have an **ENDWHILE** instruction for closing. The program will generate an error message when it is closed (**CLOSE**), if the **WHILE** and **ENDWHILE** are not matched with pair. When the command **ENDWHILE** is executed in a program, the control will jump back to associated **WHILE** for next condition evaluation. It is not allowed putting the associated **ENDWHILE** in front of the **WHILE** command.

Example

```
WHILE(P10 < 10)
P10=P10+1
ENDWHILE
```

F

Function: Specifying motor feed rate

Type: Motion program

Syntax: F{data}

{data} could be a floating point constant or an expression presenting the motor feed rate with the user units/minute.

Remarks This instruction sets the commanded velocity for motor linear or arc motion. The motor velocity will follow last commanded value until next commanded velocity executed. If the motion distance is too short as unable to reach the setting velocity, the actual velocity will be lower than the setting value. In fact, in the condition of TM<TA+TS, the TM will be set to TA+TS.

Example F2000
F12.56
F(P100*SIN(SQRT(Q1)))

FRAX

Function: Specify the axes that the motor feed rate instruction (**F**) covered.

Type: Motion program

Syntax: FRAX[({axis}[,{axis}. . .])]

{axis} Specifying axis, could be any choice of X, Y, Z, U, V, W, A, B, or C.

Remarks This instruction specifies the axes that will follow the **F** instruction setting. UTC400 calculates the motion time on each step according to the **F** setting and **FRAX** specified axes. If the instruction **FRAX** stands alone with no axis specified, all the axes in current coordinate system follow the **F** setting.

Caution: When a step with non-**FRAX**-specified-axes is executed, the amplitude of the velocity vector will be set to zero. And, the motion time will also be counted as zero. In this case, the actual motion time TM will be set to the value of TA+TS. Therefore, if the motion distance is too long, the velocity of that step might become too high as to over the rating speed base on the hardware limitation. Then, an error bit will be set (at bit 1 of memory location 0001) and the program execution will be terminated.

Example FRAX
FRAX(X,Y,Z)
FRAX(A)

G

Function: Reserved codes (G-CODE)

Type: Motion program

Syntax: G{data}[{letter}{data}. . .]

{data} could be a floating point constant or an expression, 2 digits of fractional part ranging 0.00~1048.99

{letter} is an alphabet character except G, M, N or T, used for passing a variable to PROG1000.

{data} is the specified variable value

Remarks The command **G**{data} will be recognized as **CALL 1000.**{data}. This structure allows users to define their own **G code** for special purpose easily. Just like the **CALL** instruction, we may pass variables by the **READ** command at the 1st line of subroutine and the {letter}{data} in **G** command.

Example		
	G01	;jump to N1000 of PROG1000
	G1.1	;jump to N1100 of PROG1000
	G92X0Y0	;jump to N92000 of PROG1000 passing X and Y

GOSUB

Function: Unconditional jump With Return

Type: Motion program

Syntax: GOSUB{data}

{data} is an integer specifying the destination line number in the same program ranging 0~4096.

Remarks This command causes the motion program execution to jump to the line number specified by {data}, the first **RET** will cause the control to jump back to the next line of the **GOSUB** commend. It is similar to the **CALL** command but not an inter-program call.

Example GOSUB300 ;jump to N300 of this program, to jump back on RET
GOSUB(P100)

GOTO

Function: Unconditional jump Without Return

Type: Motion program

Syntax: GOTO{data}

{data} is an integer specifying the destination line number in the same program ranging 0~4096.

Remarks This command causes the motion program execution to jump to the line number specified by {data}, the first RET or Program END will cause the control to stop. If the line number specified could not be found, the command will be ignored.

Example GOTO300 ;jump to N300 of this program
GOTO(P100)

HM

Function: Mechanical Home searching

Type: Motion program

Syntax: HM[({axis}[,{axis}…])]

{axis} Specifying axis, could be any of X, Y, Z, U, V, W, A, B or C .

Remarks The command **HM** with no axes specified will cause home searching of all the axes of the whole coordinate system. The **HM** command with axes specified will cause only the assigned axes to do home searching. This instruction will disable the blended move.

Example HM
HM(X,Y,Z)
HM(X,Y,Z,A,B,C,U,V,W)

I

Function: I-Vector setting for circular moves

Type: Motion program

Syntax: I{data}

{data} is a floating-point constant or expression representing the magnitude of the I-vector in scaled user axis units.

Remarks In circular motions, this **I{data}** value specifies the component of the vector to the arc center that is parallel to the X-axis. Either in absolute mode or incremental mode, the reference point is at the start position of X-axis.

Example X10Y20I10J5
X(P10)I(P10/2)
I33.33

I{const}={data}

Function: Set I variable value

Type: Motion program, PLC Program

Syntax: I{const}={data}

{const} is an integer representing the I-variable number ranging 0~1023

{data} is a floating-point constant or expression represents the value to be assigned to the specified I-Variable.

Remarks This command is to assign a value to a specified I variable. In the Blended Move mode, when executing a motion command, it looks one step ahead. So, this I-variable will be assigned when the previous step just start.

Example

```
I1=1
I4=P131+1000
```

IDIS

Function: Set the incremental displacement of axes

Type: Motion program

Syntax: IDIS{axis}{data}[{axis}{data}. . .]

{axis} Specifying axis, could be any choice of X, Y, Z, U, V, W, A, B or C.

{data} could be a floating point constant or an expression presenting the offset value of the specified axis in the user scaled axis units.

Remarks This command adds an offset to the current axes value. That is X' = X' + X{data}, Y' = Y'+Y{data}, Z' = Z'+Z{data}, W' = W'+W{data}, The unspecified axes will not be affected and keep the original axis value. This command does not cause any movement; it only causes the future movements to take the offset value into account.

Example ;The original axes value (X_0, Y_0, Z_0, W_0)

IDISX10Y5Z3.5

;New axes value becomes (X', Y', Z', W') = (X_0+10, Y_0+5, Z_0+3.5, W_0)

IDISX(P100)Y(Q200)

IF

Function: The block-start-point of a conditional branch

Type: Motion program, PLC Program

Syntax: IF({condition})

{condition} is a simple or compound conditions.

Remarks 1.For the true {condition}, the statements following the **IF** line will be executed step by step. The control will jump over **ENDIF** to execute the next line when the **ENDIF** or **ELSE** appears.

2. For the false condition, it will execute the line following **ELSE** if there is an associated **ELSE** command in this conditional block. If there is no associated **ELSE** command in this block, the control will jump over the associated **ENDIF** to the line following **ENDIF**.

IF command does not require a following **ELSE,** but must pair matched with an **ENDIF** command. The program will generate an error message when it is closed if the **IF** and **ENDIF** is not matched with pair. There is no limit for the multi-layer nested **IF** conditions if the memory size is adequate.

Example

```
IF(M11 = 1)
X(P100)
ELSE
X(P200)
ENDIF
IF(M12=1ANDP12!=1)
CMD"#4J+"
ENDIF
```

INC

Function: Set to incremental mode

Type: Motion program

Syntax: INC[({axis}[,{axis}. . .])]

{axis} Specifying axis, could be any of X, Y, Z, U, V, W, A, B, C

Remarks If the **INC** command with no axis specified, all axes will be set to incremental mode. If the **INC** command with axes specified, only the specified axes will be set to the incremental mode, other axes will keep their original mode.

Example INC

INC(X,Y,Z)

INC(X,Y,Z,U,V,W,A,B,C)

INIT

Function: Cancel all the axes displacement and rotation

Type: Motion program

Syntax: INIT

Remarks **INIT** command will reset all axes value to the motor zero position, the rotation angle reset to zero and the axes ratio reset to 1. (This is the default factory setting for the UTC400.)

Example INIT

IROTX45

IROT

Function: Specifies the incremental rotation angle of a coordinate plane

Type: Motion program

Syntax: AROT X{data}

{data} could be a floating point constant or an expression presenting the incremental rotation angle in the unit of angle.

Remarks This command will add the specified X-axis value as coordinate absolute rotation angle. That is θ_{rot} = θ_{rot} +X{data}. If specified axis is not X-axis, the specified value will be given up. This command will not cause movement of any motor; only cause the future movements to take the incremental rotation angle into account.

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} \cos(\theta_{rot}) & \sin(\theta_{rot}) \\ -\sin(\theta_{rot}) & \cos(\theta_{rot}) \end{bmatrix} \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Example IROTX45 ; 45 degree rotation

IROTX90 ; 90 degree rotation

ISCL

Function: Specifies the incremental scaling of the axes

Type: Motion program

Syntax: ASCL{axis}{data}[{axis}{data}. . .]

{axis} Specifying axis, could be any of X, Y, Z, U, V, W, A, B, C

{data} could be a floating point constant or an expression presenting the incremental scale of axis in the unit of ratio.

Remarks This command will multiply the specified axes value to coordinate axes ratio. That is Xratio = Xratio × X{data}, Yratio = Yratio × Y{data}, Zratio = Zratio × Z{data}, Wratio = Wratio × W{data}. This command will not cause the movement of any motor, only cause the future movements to take the multiplied ratio into account.

Example

```
ISCLX2          ; X-axis value multiplied by 2
ISCLX1.5        ; X-axis value multiplied by 1.5
ISCLX(1/3)      ; X-axis value multiplied by 1/3
ISCLX(P1)Y(P2)Z(P3)W(P4)
```

J

Function: J-Vector setting for circular moves

Type: Motion program

Syntax: J{data}

{data} is a floating-point constant or expression representing the magnitude of the J-vector in scaled user axis units.

Remarks In circular motions, this specifies the component of the vector to the arc center that is parallel to the Y-axis. Either in absolute mode or in incremental mode, the reference points are both at the start position of Y-axis.

Example

```
X10Y20I10J5
Y(P10)J(P10/2)
J33.33
```

K

Function: K-Vector setting for circular moves

Type: Motion program

Syntax: K{data}

{data} is a floating-point constant or expression representing the magnitude of the K-vector in scaled user axis units.

Remarks In circular motions, this specifies the component of the vector to the arc center that is parallel to the Z-axis. Either in absolute mode or in incremental mode, the reference points are both at the start position of Z-axis.

Example

```
#1->400X
#2->400Z
NORMAL J-1      ; Specified the X-Z plane as the rotation plane.
X10Z20I10K5
Z(P10)K(P10/2)
K33.33
```

LIN

Function: Set to linear interpolation move mode

Type: Motion program

Syntax: LIN

Remarks This command sets the current program to the Linear Interpolation Move mode, and keeps in this mode until next mode command being executed. The velocity of the linear interpolation mode is determined by the latest **F** or **TM** command.

Example

```
LIN
X10Y20Z30W40
OPENPROG1000CLEAR
N1000 LIN RET
```

M{const}={data}

Function: Assign the M variable value

Type: Motion program, PLC Program

Syntax: M{const}={data}

{const}is an integer representing the M-variable number, ranging 0~1023

{data}is a floating-point constant or expression representing the value to be assigned to this M-variable.

Remarks This instruction assigns a value to a specified M-variable. In the Blended Move mode, the execution looks one step ahead; therefore, this M-variable will be assigned when the previous step just started.

Example

```
M1=1
M4=P131+1000
```

M

Function: Machine Code (M-CODE)

Type: Motion program

Syntax: M{data}[{letter}{data}. . .]

{data} could be a floating point constant or an expression, 2 digits for fractional part ranging 0.00~1048.99

{letter}is an alphabet character except G, M, N or T, used for passing a variable to PROG1001.

{data}is the specified variable value

Remarks The command **M**{data} will be recognized as **CALL 1001.**{data}. This structure allows users to define their own **M-code** for the special purpose easily. Just like the **CALL** instruction, we may pass variables by the **READ** command in the 1st line of subroutine and the {letter}{data} in **M** command.

Example

```
M01        ;jump to N1000 of PROG1001
M1.1       ;jump to N1100 of PROG1001
M92X0Y0    ;jump to N92000 of PROG1001 passing X and Y
```

N

Function: Program line number

Type: Motion program

Syntax: N{const}

{const}is an integer representing the program line number, ranging 1~1,048,575(2^{20}-1)

Remarks This is a label for a line in the program could be easily specified by the command **GOTO, GOSUB, CALL, G** or **M**. Line numbers neither has to be in numerical order, nor have to be existed in every line. Since the line number also takes one word memory space each line, it is suggested that we put it only when it is needed.

Example

```
N1000
N2000 X200
```

NORMAL

Function: Specifies the rotation plane for a circular interpolation move

Type: Motion program

Syntax: NORMAL{vector}{data}

{vector} is one of the letter I,J, and K, representing components of the total vector parallel to the X, Y, and Z axes, respectively.

{data}only take the polarity of this value for the vector direction.

Remarks This command allows uses to specify the rotation plane, the vector K representing the X-Y plane, the vector J representing the Z-X plane while vector I representing the Y-Z plane as the rotation plane. The positive or negative value will determine the CW or CCW of the rotation. The power on default rotation plane is **NORMAL K-1** for all the coordinate system.

Example

```
NORMAL K-1     ; X-Y plane
NORMAL J-1     ; Z-X plane
NORMAL I-1     ; Y-Z plane
```

OR

Function: Conditional OR

Type: Motion program, PLC Program

Syntax: OR{condition}

{condition} is a simple or compound conditions.

Remarks This command only used in **IF** or **WHILE** command, it performs the Boolean operator logical OR.

Example

```
IF(M11 = 1 OR M12 = 1)
CMD"R"
P11=1
ENDIF
```

P{const}={data}

Function: Set P variable value

Type: Motion program, PLC Program

Syntax: P{const}={data}

{const} is an integer representing the P-variable number ranging 0~1023

{data}is a floating-point constant or expression represents the value to be assigned to the specified P-Variable.

Remarks This command assigns a value to a specified P-variable. In the Blended Move mode, the execution looks one step ahead. So, the P-variable will be assigned when the previous step just start.

Example

```
P1=1
P4=P131+1000
P123=ROUND(100*SIN(Q100))
```

POSTLUDE

Function: Automatic subroutine call after a programming move

Type: Motion program

Syntax: POSTLUDE1{command}

POSTLUDE0

{command} could be the following types:

CALL{const}

G{const}

M{const}

Remarks **POSTLUDE1** command will cause the future execution of a motion command to insert a subroutine call after that movement. The subroutine call will not be executed following a non-motor-moved command. The type of subroutine call could only be a constant, not an expression or variable.

POSTLUDE0 will cancel all the automatic inserted subroutine executions.

Example

```
POSTLUDE1 CALL200     ;Insert a CALL200 after subsequent moves
LIN
X100Y100            ; Implicit CALL200 after this move
X200Y200            ; Implicit CALL200 after this move
…….
POSTLUDE0           ; Disable postlude
CLOSE
OPENPROG200 CLEAR
Z-1
Z1
CLOSE
```

PSET

Function: Define the current axis position

Type: Motion program

Syntax: PSET

Remarks This command re-defines the motor position of the axes. The actual motor position will not be changed by this command. Insert this command between any two movements will cause the Blended Move to stop. If this effect is not permitted in the program execution, please use the command **ADIS** or **IDIS** instead.

Example

```
PSETX10Y10          ; Set the axes position X=10, Y=10
PSET
X10Y10              ; The same effect as previous statement.
```

PSETX(Q124)Y(Q125)Z(Q126)

Q{const}={data}

Function: Set Q variable value

Type: Motion program, PLC Program

Syntax: Q{const}={data}

{const} is an integer representing the Q-variable number ranging 0~1023

{data}is a floating-point constant or expression represents the value to be assigned to the specified Q-Variable.

Remarks This command assigns a value to a specified Q-variable. In the Blended Move mode, the execution looks one step ahead. This Q-variable will be assigned to a new value when the previous step just start.

Example

```
Q1=1
Q4=P131+1000
Q123=ROUND(100*SIN(Q100))
```

R

Function: Set circle radius

Type: Motion program

Syntax: R{data}

{data}is a floating-point constant or expression representing the radius of an arc move with user length units.

Remarks In the circular interpolation mode, if we define the endpoint and radius **R** for next move, there are two paths with different circle center meet the specified condition. When we give **R** a positive value, the circular move will take the path with the arc less than or equal to 180°. If the radius value specified in {data} is a negative value, the circular move will take the path with the arc greater than 180°.

If the assigned radius is less than half the distance from start point to endpoint, there is no circular move possible. In this case UTC400 will assign the radius as the half distance from start point to endpoint and perform a half circle move accordingly.

If there is no **R** and no **IJK** vector specified on a command line in the circular move mode, the command will be recognized as a linear motion.

PS: If there are **AROT**, **ASCL**, …commands for axes rotation or ratio settings, the **R** settings will not follow the change. Please use **I, J or K** for the circular setting.

Example CIR1

```
X10Y10R10          ;1/4 circle arc to (10, 10)
X0Y0R-10           ;3/4 circle arc to (0, 0)
X(P100)R(P100/2)   ; half circle arc to (P100, 0)
```

READ

Function: Read the variable for subroutine

Type: Motion program

Syntax: READ({letter}[,{letter}. . .])

{letter}is an alphabet character except G, M, N or T, used for passing a variable to the subroutine.

Remarks **READ** command is executed in a subroutine. It will cause a letters scanning at the associated **subroutine call** instruction for all the possible variables until an English alphabet not in the list of letters to **READ**. If a letter value is successfully read into the associated Q-variable, the bit N-1 of Q100 will be set to 1. (N noted for Nth letter of the alphabet). The value been read is stored in the variable Q(100+N). The **READ** command will be ignored if it appears at somewhere not in the subroutine.

Following table is the Q-variable and flag bit of Q100 associated with each letter:

Letter	**Target Variable**	**Q100 Bit**	**Bit Value Decimal**	**Bit Value Hex**
A	Q101	0	1	$01
B	Q102	1	2	$02
C	Q103	2	4	$04
D	Q104	3	8	$08
E	Q105	4	16	$10
F	Q106	5	32	$20
G*	Q107	6	64	$40
H	Q108	7	128	$80
I	Q109	8	256	$100

J	Q110	9	512	$200
K	Q111	10	1,024	$400
L	Q112	11	2,048	$800
M*	Q113	12	4,096	$1000
N*	Q114	13	8,192	$2000
O	Q115	14	16,384	$4000
P	Q116	15	32,768	$8000
Q	Q117	16	65,536	$10000
R	Q118	17	131,072	$20000
S	Q119	18	262,144	$40000
T*	Q120	19	524,288	$80000
U	Q121	20	1,048,576	$100000
V	Q122	21	2,097,152	$200000
W	Q123	22	4,194,304	$400000
X	Q124	23	8,388,608	$800000
Y	Q125	24	16,777,216	$1000000
Z	Q126	25	33,554,432	$2000000

*Not available for user

Example

```
N4000 READ(X)
DWE(Q124)
RET
N92000 READ(X,Y)
IF(Q100&8388608>0)
PSET X(Q124)
ENDIF
IF(Q100&16777216>0)
PSET Y(Q125)
ENDIF
```

RET

Function: Return from subroutine

Type: Motion program

Syntax: RET

Remarks The **RET** command will cause the motion program to jump back to the next line of original call function, if this routine starts from a **CALL, GOSUB, G, M, code.** If this command appears in the main program, it represents a program end.

Example

```
OPENPROG1000CLEAR
RPD RET
N1000 LIN RET
. . .
```

RPD

Function: Set to rapid move mode

Type: Motion program

Syntax: RPD

Remarks This command sets the current motion mode to the rapid move mode until next motion mode command is executed.

Example

```
RPD
X10Y20Z30W40
OPENPROG1000CLEAR
RPD RET
N1000 LIN RET
….
```

S

Function: Spindle velocity setting

Type: Motion program

Syntax: S{data}

{data} is floating-point constant or expression representing the spindle velocity.

Remarks This command loads the {data} content into variable Q127. We may detect the change of Q127 through PLC program and change the spindle velocity accordingly.

Example

```
S2000
S12.56
S(P100*SIN(SQRT(Q1)))
```

SEND

Function: Send message to computer.

Type: Motion program, PLC Program

Syntax: SEND"{message}"

"message" is the messages to be sent to computer.

SEND{C1/C2},"{ variable=}",{ variable}

C1/C2 means Com1/Com2, "{ variable=}" can be "I(p/Q/M)xxx=", { variable} can be I(p/Q/M)xxx

Remarks This command will send out {message} through RS232 to the computer or controller to controller for error indication or for status reporting.

Example

```
IF(M180 = 1 AND P180 != 1)
     SEND"PROGRAM RUNNING"
     P180=1
ENDIF
IF(M180 = 0 AND P180 != 0)
     SEND"PROGRAM STOP"
     P180=0
ENDIF
```

SPLINE

Function: Spline move mode

Type: Motion program

Syntax: SPLINE

Remarks This command sets the current motion in Spline move mode.

In the spline move mode, all the motion trajectory will be generated through a 3-dimentional calculation. By this way, we can obtain a smooth trajectory passing by all the position we specified in the program.

In order to get the smooth trajectory, we need to add an additional time segment (TM1) for the acceleration at the starting point, and also a time segment (TM2) at the endpoint. The total motion time will be the total time for program line motion plus TM1 and TM2.

The motion commands after the **SPLINE** command line, could specify the velocity **(F)** or motion time **(TM)** step by step. The Spline move mode will be terminated at program end or if other motion mode (**RPD,LIN, CIR1** or **CIR2**) command appears.

Example

```
RPDX10Y10
SPLINE
X20Y15F2000
TM500
X30Y45
. . .
RPDX0Y0
```

STOP

Function: Stop program execution

Type: Motion program

Syntax: STOP

Remarks This command causes the executing program to a stop. The **R** or **S** command will make the program execution re-started from the line following the **STOP** command.

Example

```
X100Y100
Z200
STOP
X0Y0Z0
```

T

Function: Tool select Code (T-CODE)

Type: Motion program

Syntax: T{data}[{letter}{data}. . .]

{data} could be a floating point constant or an expression, 2 digits for fractional part ranging 0.00~1048.99

{letter}is an alphabet character except G, M, N or T, used for passing a variable to PROG1002.

{data}is the specified variable value

Remarks The command **T**{data} will be recognized as **CALL 1002.**{data}. This structure allows users to define their own **T-code** for special purpose easily. Just like the **CALL** instruction, we may pass variables by the **READ** command in the 1st line of subroutine and the {letter}{data} in **T** command.

Example

```
T01        ; jump to N1000 of PROG1002
T1.1       ; jump to N1100 of PROG1002
T92X0Y0    ; jump to N92000 of PROG1002 passing X and Y
```

TA

Function: Set the total acceleration time

Type: Motion program

Syntax: TA{data}

{data} could be a floating point constant or an expression presenting the acceleration time with the units msec.

Remarks The acceleration time consists of 2 portions, TS is the S-curve acceleration time and TL is the linear acceleration time. The total acceleration time TA=TS+TL.

If we set S Curve Acceleration Time TS=0, we will obtain a constant slope acceleration. If TS=100, the motion profiles will become pure S curve. If 0>TS>100, the profile will have both linear (constant acceleration) and S-curve portion.

Example

```
TA100
TA(P10*2)
TA(INT(SQRT(30)*Q200+0.5))
```

TM

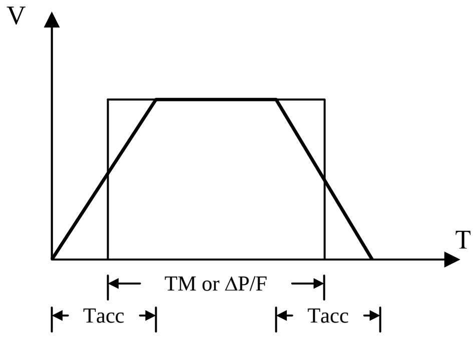
Function: Set move time

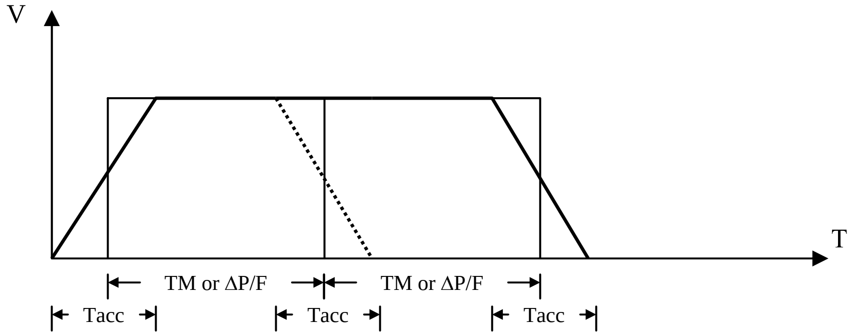
Type: Motion program

Syntax: TM{data}

{data} could be a floating point constant or an expression presenting the move time with the units msec.

Remarks This command sets the move time of a motion in the motion program. The setting will keep to the next **F** or **TM** command executed. The **TM** value will be set to **TA** automatically if the setting is less than TA. (Also see **TA** command)





Example TM100

TM(P10*2)

TM(INT(SQRT(30)*Q200+0.5))

TS

Function: Set the percentage of S-curve acceleration time

Type: Motion program

Syntax: TS{data}

{data} is an integer between 0~100.

Remarks This command is the percentage of S-curve acceleration time over the overall acceleration time **TA**.

Example

```
TS100
TS(P10*2)
TS(INT(SQRT(30)*Q200+0.5))
```

UNIT

Function: Set the axes units ratio

Type: Motion program

Syntax: UNIT{axis}{data}[{axis}{data}. . .]

{axis} Specifying axis, could be any of X, Y, Z, U, V, W, A, B, C

{data} could be a floating point constant or an expression presenting the ratio with the units is the ratio value

Remarks This command will multiply a ratio to the axes units scales. (The axes unit scales are defined by the on-line command **#{number}->**.) This command will neither move the motor nor change the axes unit's scales. It only causes the future movement to multiply a ratio to the unit scale. The ratio setting will be ignored if the setting is 0. It will generate a mirror trajectory if the ratio setting is a negative value.

Example

```
#1->1000X    ; X axis Unit Scale = 1000 (counts/mm)
X10          ; X axis actual position: 10000 counts
X0           ; X axis actual position: 0 counts
UNITX1.5
X10          ; X axis actual position: 15000 counts
UNITX1       ; Cancel the X-axis ratio setting
X10          ; X axis actual position: 10000 counts
UNITX1.5
DWE2         ; re-PMATCH
```

CIR1X(10/1.5)I10 ; To path an ellipse, the X endpoint should be divided by the ratio 1.5

WHILE

Function: Start point of a conditional loop

Type: Motion program, PLC Program

Syntax: WHILE({condition})

{condition}is a simple or compound conditions

Remarks **WHILE** is the only loop command in UTC400. It is used to do a block of program command repeatedly until the condition is not true. If there is no move command or **DWELL, DELAY** in the **WHILE** loop and the condition keeps true, the control will leave this loop after each scan. This action prevents all the background timing from being used in the **WHILE** loop. It also cancels the Blended Move Function.

Each of the conditional loop starts with **WHILE** should end with a **ENDW** instruction. The program will generate an error message when it is closed (**CLOSE**) if the **WHILE** and **ENDW** do not match with pair. When the command **ENDW** is executed in a program, the control will jump back to associated **WHILE** for next condition evaluation. It is not allowed putting the associated **ENDW** in front of the **WHILE** command.

The maximum nested **WHILE** loop is 16. There is also an error message be generated if it is over limitation

Example

```
WHILE(M11 = 1)        ;Wait until M11 equal to 0
ENDWHILE
INC
P1=0
WHILE(P1 < 10)
    X1000
    P1=P1+1
ENDW
```

Program Pointer Control Command

CLEAR

Function: Clear the current program contents

Type: Motion program pointer control (Program command, On-line Command)

Syntax: CLEAR

Remarks This command erases the current opened buffer and points the program pointer to the beginning of this program. Normally, we put this command following the **OPEN {buffer}** command to override the original contents.

Example

```
OPEN PROG1 CLEAR
LINX10Y10
CIR1I5
. . .
CLEAR
LIST
<End>
```

DEL

Function: Delete program command line

Type: Motion program pointer control (Program Command)

Syntax: DEL[{const}]

DEL*

{const} is an integer representing the number of lines to be deleted after current line.

Remarks If without a number specified in this command, the current program line will be deleted. If a number specified in this command, it specifies that the total number of lines after the current program line will be deleted. If the command followed by a '*', all the program lines after the current line will be deleted.

Example

```
OPEN PROG1 CLEAR
LINX10Y10
CIR1I5
```

```
CLOSE
OPEN PROG1
@JP0
DEL
CLOSE
LISTPROG1
```
CIR1I5

END

Function: End of a program

Type: Motion program pointer control (On-line Command)

Syntax: END

Remarks In the program editing process, it is possible to move the program pointer to any line in the program for insertion, modification or deletion. This command will cause the pointer point to end of the program. (The pointer point to the end of a program when it is just opened.)

Example

```
OPEN PROG1
LINX10Y10
CIR1I5
. . .
@JP0
LIST
```
LINX10Y10
```
@END
LIST
```
<*End*>

INS

Function: Set the program editing to the insertion mode

Type: Motion program pointer control (Program Command)

Syntax: INS

Remarks In the program editing insertion mode, all the entered new contents will be inserted to the current content after the program pointer. The original content after the pointer will be push to follow the new entered contents.

Example

```
OPEN PROG1 CLEAR
LINX10Y10
CIR1I5
CLOSE
OPEN PROG1
@JP0
INS
RPDX0Y0
CLOSE
LISTPROG1
RPDX0Y0
LINX10Y10
CIR1I5
```

JP

Function: Program pointer points to a specified line number.

Type: Motion program pointer control (On-line Command)

Syntax: JP{const}

{const} is an integer representing the program line number raging 1~4096

Remarks In the program editing process, it is possible to point the program pointer to any line in the program for insertion, modification or deletion. This command will cause the pointer point to the **{const}** specified line number. The pointer will point to the start of this program if the line number is not specified. (The same as **JP0**)

Example

```
OPEN PROG1
LINX10Y10
N100CIR1I5
. . .
JP100
LIST
N100CIR1I5
```

JPN

Function: Program pointer points to a specified label number.

Type: Motion program pointer control (On-line Command)

Syntax: JPN{const}

N{const} is an integer representing the program label number raging 1~4096

Remarks In the program editing process, it is possible to point the program pointer to any label number in the program for insertion, modification or deletion. This command will cause the pointer point to the **N{const}** specified label number.

Example

```
OPEN PROG1
LINX10Y10
N100CIR1I5
. . .
JPN100
LIST
N100CIR1I5
```

JP*

Function: Program pointer indirectly points to the content of Mx89.

Type: Motion program pointer control (On-line Command)

Syntax: JP*

Remarks This command will cause the program pointer point to the content of Mx89.

LEARN

Function: Motor position teaching

Type: Motion program pointer control (Program Command)

Syntax: LEARN[({axis}[,{axis}. . .])]

{axis}represents the axis for teaching, range:1~4

Remarks This command will cause the new axes position equal to the original setting position plus the current motor position. We may first move the motors to a desire position then execute the **LEARN** command to add the motor position value the original setting axes value. All the motor position value will be read if there is no axis specified in the **LEARN** command.

Example

```
#1->1000X
#2->1000Z
OPEN PROG1
<Ctrl-P>
2500 -2000 0 0 0 0 0 0
LEARN(1,2)
@UPPER
LIST
X2.5Z-2
```

LIST

Function: Report the program content which the pointer pointed.

Type: Motion program pointer control (Program Command, On-line Command)

Syntax: LIST

Remarks In the program editing, it is possible to check he current content of the pointed program line. If the pointer point to program end, the reported message will be an *<End>* message.

Example

```
OPEN PROG1
LINX10Y10
CIR1I5
LIST
<End>
@UPPER
LIST
CIR1I5
```

NEXT

Function: Program pointer point to next program line

Type: Motion program pointer control (On-line Command)

Syntax: NEXT

Remarks In the program editing process, it is possible to point the program pointer to any line in the program for insertion, modification or deletion. This command will cause the pointer points to the next program line.

Example

```
OPEN PROG1
LINX10Y10
CIR1I5
. . .
@JP0
LIST
LINX10Y10
@NEXT
LIST
CIR1I5
```

OVR

Function: Set the program editing process as the write over mode.

Type: Motion program pointer control (Program Command)

Syntax: OVR

Remarks In the editing write over mode, the new edited program contents will over-write the current contents, other contents will keep unchanged.

Example

```
OPEN PROG1 CLEAR
LINX10Y10
CIR1I5
CLOSE
OPEN PROG1
@JP0
OVR
RPDX0Y0
CLOSE
LISTPROG1
RPDX0Y0
CIR1I5
```

UPPER

Function: Program pointer point to last program line

Type: Motion program pointer control (On-line Command)

Syntax: UPPER

Remarks In the program editing process, it is possible to point the program pointer to any line in the program for insertion, modification or deletion. This command will cause the pointer point to the previous program line.

Example

```
OPEN PROG1
LINX10Y10
CIR1I5
LIST
<End>
@UPPER
LIST
CIR1I5
```

Appendix

I. Example program of communication

```
// The most common used baud rate are 9600 or 38400
// If use COM1 of PC, define iComBase as 0x3f8
// If use COM2 of PC, define iComBase as 0x2f8

// Use this function to setup RS232 of PC first
void SetupRS232()
{
        _outp(iComBase + 3, 0x83);
        _outp(iComBase, (int)(115200 / lBaudRate));
        _outp(iComBase + 1, (int)(450 / lBaudRate));
        _outp(iComBase + 3, 0x03);
}

// Use this function to verify the controller is there or not
BOOL CardOnLine()
{
        int    i = 0, n = 0;
        char buf[255];

        while(n < 3 && i < 11) {
               _outp(iComBase, 24);   // Ctrl-X
               Sleep((DWORD)500);
               while(GetLine(buf));
               _outp(iComBase, 7);     // Ctrl-G
               i = GetLine(buf);
               n++;
        }
        if(i > 11 && i < 15) return TRUE;
        else return FALSE;
```

```
}

// Use this function to send one character to controller
BOOL SendChar(char ch)
{
        long   i = 0;

        while(i++ < lTimeOut && !(_inp(iComBase+5) & 0x20));
        while(i++ < lTimeOut && !(_inp(iComBase+6) & 0x10));
        // If don’t want to use the handshake signal, delete one line above
        if(i < lTimeOut) {
                _outp(iComBase, ch);
                return TRUE;
        }
        _outp(iComBase, 24);    // Ctrl-X
        return FALSE;
}

// Use this function to receive response from controller
int GetLine(char* linebuf)
{
        char        ic;
        int         nc;
        long        i;

        ic = 0; nc = 0; i = 0;
        _disable();
        _outp(iComBase+4, 2);
        while (i++ < lTimeOut && ic != '\r' && nc < 255) {
                if((_inp(iComBase+5) & 1) == 1) {
                        linebuf[nc++] = _inp(iComBase);
                        i=0;
                }
        }
        _outp(iComBase+4,0);
        _enable();
        linebuf[nc] = 0;
        return(nc);
    }

    // Use this function to send command line to controller
    void SendLine(const char* ch)
    {
        int    i = 0;

        while(ch[i] != 0) SendChar(ch[i++]);
        SendChar('\r');
   }
```

II. Suggested M-Variable Definition (V3.0)

```
M0->&MAINCOUNT,S;               ; INTERRUPT COUNTER

; GENERAL PURPOSE INPUTS AND OUTPUTS
M1->&PGA1 DOUT DATA,0,1;            ; MACHINE OUTPUT 1
M2->&PGA1 DOUT DATA,1,1;            ; MACHINE OUTPUT 2
M3->&PGA1 DOUT DATA,2,1;            ; MACHINE OUTPUT 3
M4->&PGA1 DOUT DATA,3,1;            ; MACHINE OUTPUT 4
M5->&PGA1 DOUT DATA,4,1;            ; MACHINE OUTPUT 5
M6->&PGA1 DOUT DATA,5,1;            ; MACHINE OUTPUT 6
M7->&PGA1 DOUT DATA,6,1;            ; MACHINE OUTPUT 7
M8->&PGA1 DOUT DATA,7,1;            ; MACHINE OUTPUT 8
M9->&PGA1 DOUT DATA,0,8;            ; MACHINE OUTPUT 1-8 BYTE

M11->DIBUFFER,0,1;                  ; MACHINE INPUT 11
M12->DIBUFFER,1,1;                  ; MACHINE INPUT 12
M13->DIBUFFER,2,1;                  ; MACHINE INPUT 13
M14->DIBUFFER,3,1;                  ; MACHINE INPUT 14
M15->DIBUFFER,4,1;                  ; MACHINE INPUT 15
M16->DIBUFFER,5,1;                  ; MACHINE INPUT 16
M17->DIBUFFER,6,1;                  ; MACHINE INPUT 17
M18->DIBUFFER,7,1;                  ; MACHINE INPUT 18
M19->DIBUFFER,0,8;                  ; MACHINE INPUT 11~18

M21->DIBUFFER,8,1;                  ; MACHINE INPUT 21
M22->DIBUFFER,9,1;                  ; MACHINE INPUT 22
M23->DIBUFFER,10,1;                 ; MACHINE INPUT 23
M24->DIBUFFER,11,1;                 ; MACHINE INPUT 24
M25->DIBUFFER,12,1;                 ; MACHINE INPUT 25
M26->DIBUFFER,13,1;                 ; MACHINE INPUT 26
M27->DIBUFFER,14,1;                 ; MACHINE INPUT 27
M28->DIBUFFER,15,1;                 ; MACHINE INPUT 28
```

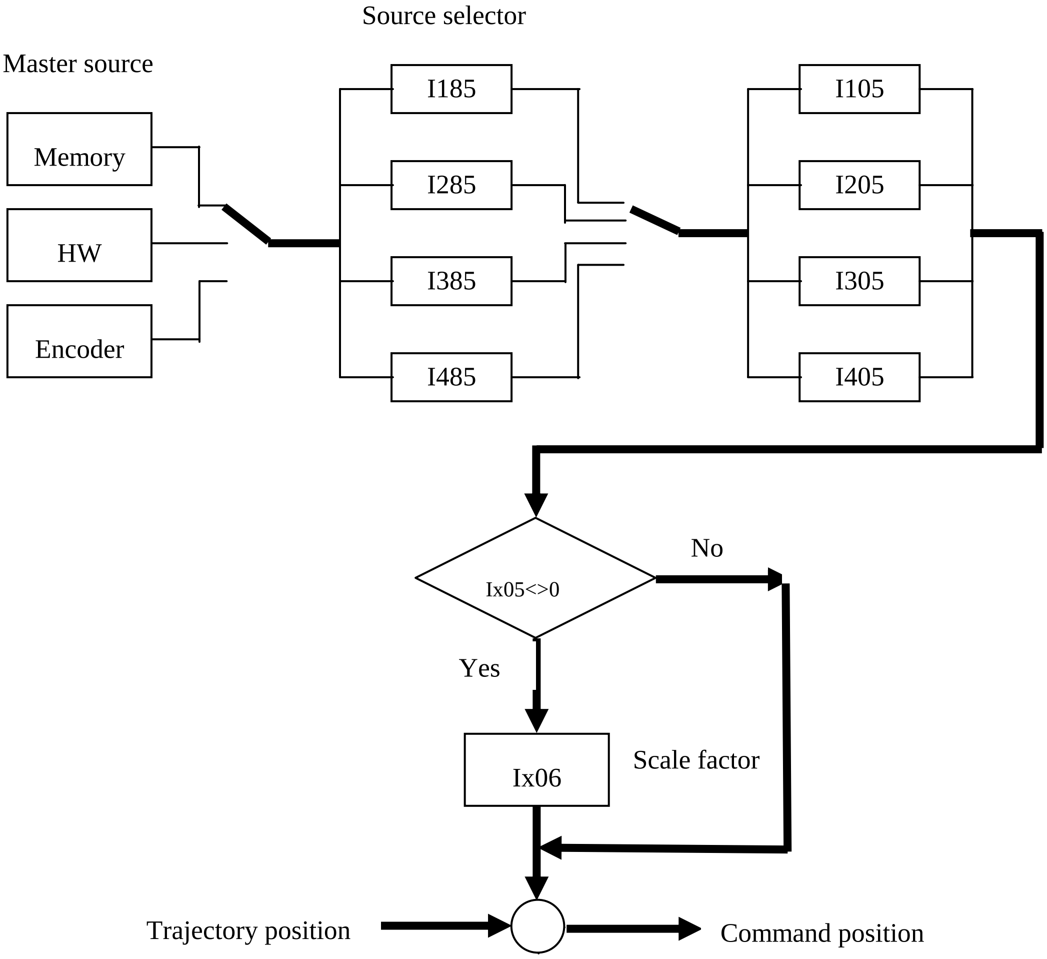
```
M29->DIBUFFER,8,8    ;                          ; MACHINE INPUT 21~28

;----- MULTIPLEXER OUTPUT
M40->&PGA1 DOUT1 DATA,0,1;       ; MULTIPLEXER OUTPUT 40
M41->&PGA1 DOUT1 DATA,1,1;       ; MULTIPLEXER OUTPUT 41
M42->&PGA1 DOUT1 DATA,2,1;       ; MULTIPLEXER OUTPUT 42
M43->&PGA1 DOUT1 DATA,3,1;       ; MULTIPLEXER OUTPUT 43
M44->&PGA1 DOUT1 DATA,4,1;       ; MULTIPLEXER OUTPUT 44
M45->&PGA1 DOUT1 DATA,5,1;       ; MULTIPLEXER OUTPUT 45
M46->&PGA1 DOUT1 DATA,6,1;       ; MULTIPLEXER OUTPUT 46
M47->&PGA1 DOUT1 DATA,7,1;       ; MULTIPLEXER OUTPUT 47
M48->&PGA1 DOUT1 DATA,0,8;       ; MULTIPLEXER OUTPUT 40~47


;----- MULTIPLEXER INPUT
M50->DIBUFFER,16,1;                 ; MULTIPLEXER INPUT 50
M51->DIBUFFER,17,1;                 ; MULTIPLEXER INPUT 51
M52->DIBUFFER,18,1;                 ; MULTIPLEXER INPUT 52
M53->DIBUFFER,19,1;                 ; MULTIPLEXER INPUT 53
M54->DIBUFFER,20,1;                 ; MULTIPLEXER INPUT 54
M55->DIBUFFER,21,1;                 ; MULTIPLEXER INPUT 55
M56->DIBUFFER,22,1;                 ; MULTIPLEXER INPUT 56
M57->DIBUFFER,23,1;                 ; MULTIPLEXER INPUT 57
M58->DIBUFFER,16,8;                 ; MULTIPLEXER INPUT 50~57

M60->&GLOSTATUS,12,3;               ; GLOSTATUS.BBUFOPEN
M61->&GLOSTATUS,16,2;               ; GLOSTATUS.BGATH
M62->&GLOSTATUS,15,1;               ; GLOSTATUS.BGATHROV
M63->&GLOSTATUS,3,1;                ; GLOSTATUS.BSAVED
M64->&GLOSTATUS,4,4;                ; GLOSTATUS.BCMD1REQ
M65->&GLOSTATUS,8,4;                ; GLOSTATUS.BCMD2REQ
M66->&GLOSTATUS;                    ; GLOSTATUS
M67->&PLCSCANCOUNT,S;
```

```
M71->&TIMERCOUNT1,S;                 ; TIMER1 COUNT
M72->&TIMERCOUNT2,S;                 ; TIMER2 COUNT
M73->&TIMERCOUNT3,S;                 ; TIMER3 COUNT
M74->&TIMERCOUNT4,S;                 ; TIMER4 COUNT
M75->&TIMERCOUNT5,S;                 ; TIMER5 COUNT
M76->&TIMERCOUNT6,S;                 ; TIMER6 COUNT
M77->&TIMERCOUNT7,S;                 ; TIMER7 COUNT
M78->&TIMERCOUNT8,S;                 ; TIMER8 COUNT

M80->PGA1 AD1 BUF,S;                 ; AD1
M81->PGA1 AD2 BUF,S;                 ; AD2
M82->PGA1 DA1 BUF,S;                 ; DA1
M83->PGA1 DA2 BUF,S;                 ; DA2
M84->&HWCOUNT,0,16,S;

;REGISTERS ASSOCIATED WITH AXIS1
MX01->ENCCOUNT,0,16,S;               ; 16-BIT UPDOWN COUNTER
                                     (COUNTS)
MX02->L:DACOUT;                      ; DAC COMMAND
(MX02->PULSEOUT,S;)                 ; PULSE
MX03->PLSCOUNT,0,16,S;           ; 16-BIT CAPTURE COUNTER (COUNTS)
MX04->L:DACCMD;                      ; DAC
MX05->L:DACREAL;                     ; DAC

MX20->PGA1 FLAG BUF,0,1;             ; +LIM1
MX21->PGA1 FLAG BUF,1,1;             ; -LIM1
MX22->PGA1 FLAG BUF,2,1;             ; HMFL FLAG
MX23->PGA1 FLAG BUF,3,1;             ; DRIVER FAULT
MX24->PGA1 HSTS,0,1;                 ; TRIGGEREED
MX25->PGA1 HSTS,1,1;                 ; FLAG
MX26->PGA1 HSTS,2,1;                 ; CHC
MX27->PGA1 HSTS,3,1;                 ; LATCH C
MX40->&MOTORSTATUS,0,4;              ; MOTORSTATUS.BSTATUS
MX41->&MOTORSTATUS,4,3;              ; MOTORSTATUS.BSERVO
```

```
MX42->&MOTORSTATUS,8,3;              ; MOTORSTATUS.BJOG
MX43->&MOTORSTATUS,14,2;             ; MOTORSTATUS.BLIMIT
MX44->&MOTORSTATUS,17,1;             ; MOTORSTATUS.BFEFATL
MX45->&MOTORSTATUS,7,1;              ; MOTORSTATUS.BHOME
MX46->&MOTORSTATUS,11,1;             ; MOTORSTATUS.BDIRCMD
MX47->&MOTORSTATUS,12,1;             ; MOTORSTATUS.BINC
MX48->&MOTORSTATUS,13,1;             ; MOTORSTATUS.BINPOS
MX49->&MOTORSTATUS,16,1;             ; MOTORSTATUS.BFAULT

MX61->&CMDPOS,3,29,S;            ; COMMAND POSITION (COUNTS)
MX62->&CURPOS,3,29,S;            ; ACTUAL POSITION (COUNTS)
MX63->&JOGREG,3,29,S;            ; JOG REGISTER POSITION (COUNTS)
MX64->&POSBIAS,3,29,S;           ; POSITION BIAS (COUNTS)
MX65->&JOGTARGET,3,29,S;             ;
MX66->&HW_POS,3,29,S;            ; PRESENT MASTER POSITION(COUNTS)
MX67->L:&AXISPOS;                ;
MX68->L:CMDVEL;                  ; REAL VELOCITY(UNIT: COUNTS/MSEC)
MX69->&CURVEL,3,29,S;             ; ACTUAL VELOCITY (UNIT: COUNTS/MSEC)

;REGISTERS ASSOCIATED WITH COORDINATE SYSTEM 1
MX80->&CSSTATUS,0,3;             ; CSSTATUS.BSTATUS
MX81->&CSSTATUS,4,3;             ; CSSTATUS.BMOVING
MX82->&CSSTATUS,8,3;             ; CSSTATUS.BRUNREG
MX83->&CSSTATUS,12,3;            ; CSSTATUS.BMODEL
MX88->L:&PROGNUMREG;             ;
MX89->&LINENUMREG,S;             ;

MX91->L:&SCALE;                  ; SCALE
MX97->L:&NEWOVERRIDE;            ;
MX98->L:&FROVERRIDE;             ;

M800..999->&EXTERNALIO;          ;
```

III. *Position Following – electrical gearing*

When the master signal specified by **Ix85** come in, it would pass by a buffer area specified by **Ix86** (The higher **Ix86** settings the lower change rate of the speed). If **Ix05** set to 1, the **x** motor will follow the master signal from **I185** with the gear ratio of **Ix06** setting. If **Ix05** set to 2, the **x** motor will follow the master signal from **I285** with the gear ratio of **Ix06** setting.

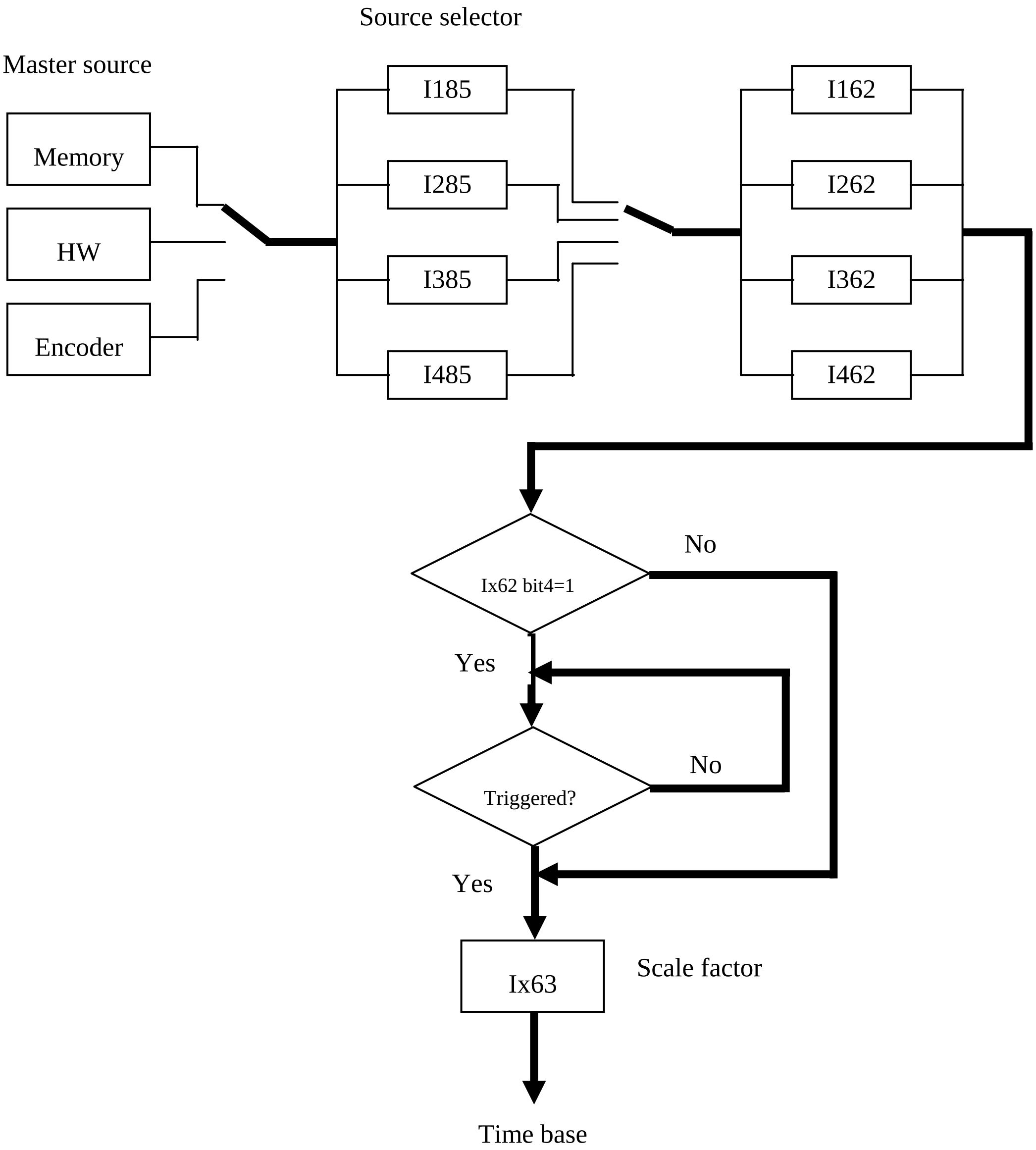


Flow chart of position following (V3.0)

IV. *Time Based Following – electrical cams*

Besides the position following, there is a special follow method that the master signal will control the time base to archive following purpose. In this mode (**Ix62<>0**), user only have to pre-define a motion profile. The incoming master signal will act as time base. For example, if **Ix63=100,** each count of the master signal will be interpreted as **100** msec. The motion profile will be performed according to the time base of master signal.

If **Ix62** bit4 = 1 also, the following action will start at C signal of 4th axis. Before the trigger signal comes, the control will halt and wait.



Flow chart of time base following (V3.0)

<Example>:

There is a Master encoder with 400 counts/rev, The control should jog 100 mm for each revolution of master encoder. The program will be as:

```
INC                 ; Incremental Mode
      I50=1               ; Blended Move mode
      I185= 5             ; Master from HW Port
      I162=1              ; Start the Time Base Following
      TM(400-(I152+II153)/2) ; Take acceleration time into account.
      X100                ; X-axis move 100 mm first time
      TM400               ; TM=400 ms (Per encoder revolution) after 1st count.
      P0 = 0
      WHILE(P0 = 0)
X100                ; X-axis move 100 mm each revolution
      ENDW
```

V. *Example Program*

```
; ********************** START.UTC **************************
; These programs provide examples of simple programs that
; can be used to get acquainted with writing motion programs
; on UTC400.    These programs will run indefinitely once started.
; An 'A' command can be used to stop them.    By changing
; the WHILE condition or eliminating the WHILE statement,
; a program can be given a finite run length.
; Note: Command lines whose comments have an '*' are not
; actually part of the program -- they are commands to control
; the program buffers.
; ***********************************************************

CLOSE OPEN PROG 1 CLEAR        ; *open UTC400 program buffer #1

F5000                          ; specify the feedrate in encoder cts/sec.
WHILE(1<2)                     ; begin an indefinite loop
        X(P1)                  ; move to position as set by variable
        DWELL300               ; pause for 300 msec
        X0                     ; move back to position 0 (home)
        DWELL300               ; pause again for 300 msec
ENDWHILE                       ; end the indefinite loop

CLOSE                          ; *close UTC400 program buffer

CLOSE OPEN PROG 2 CLEAR        ; open UTC400 program buffer #2

WHILE(1<2)                     ; begin an indefinite loop
        F(P1*2)                ; set speed according to variable
        Y10000                 ; move motor #2 to position 10000
        DWELL300               ; pause for 300 msec
        F(P1*2)                ; reread the variable
        Y0                     ; go back to position 0
        DWELL300               ; pause again for 300 msec
ENDWHILE                       ; end the indefinite loop

CLOSE                          ; close UTC400 program buffer

CLOSE OPEN PROG 3 CLEAR        ; open UTC400 program buffer #3

F5000                          ; specify feedrate in encoder counts/sec
WHILE(1<2)                     ; begin an indefinite loop
        TA(P1/5)               ; set accel. time according to variable
```

```
          TS50                   ; set "S" curve time to 50 msec
          Z50000                 ; move to position 50000
          DWELL300               ; pause for 300 msec
          TS50                   ; "S" curve again (actually not needed)
          TA(P1/5)               ; reread variable to update accel.
          Z0                     ; return to position zero
          DWELL300               ; pause again
ENDWHILE                         ; end indefinite loop

CLOSE                            ; close UTC400 program buffer

CLOSE OPEN PROG 4 CLEAR          ; open UTC400 program buffer #4

F5000                            ; specify feedrate in encoder counts/sec
TA200                            ; specify acceleration time in msec
TS50                             ; set "S" curve time in msec
WHILE(1<2)                       ; begin indefinite loop
          A50000                 ; jog to position 50000
          DWELL(P1/100)          ; pause according to variable
          A0                     ; go back to position 0
          DWELL(P1/100)          ; pause again...
ENDWHILE                         ; end the indefinite loop

CLOSE                            ; close UTC400 program buffer

CLOSE OPEN PROG 5 CLEAR          ; open UTC400 program buffer #5

F1000                            ; specify feedrate in encoder counts/min
TA200                            ; set acceleration time
TS50                             ; set "S" curve time
WHILE(1<2)                       ; begin indefinite while loop
          X(P1)Y(P1)Z(P1)A(P1)   ; move all 4 motors to variable pos.
          DWELL300               ; pause for 300 msec
          X0Y0Z0A0               ; move all motors back to position 0
          DWELL300               ; pause another 300 msec
ENDWHILE                         ; end indefinite loop

CLOSE                            ; close UTC400 program buffer

CLOSE OPEN PROG 6 CLEAR          ; open UTC400 program buffer #6

TA200                            ; set acceleration time
TS50                             ; set "S" curve time
WHILE(1<2)                       ; begin indefinite loop
          F(P1)                  ; read variable for feedrate
          X40000                         ; move motor #1 first
          Y40000                         ; move motor #2 second
          Z40000                         ; move motor #3 third
          A40000                         ; move motor #4 fourth
          DWELL300                       ; pause for 300 msec
          F(P1)                          ; read variable again
          A0                             ; return motor #4 to position 0
          Z0                             ; return motor #3 to position 0
          Y0                             ; return motor #2 to position 0
          Z0                             ; return motor #1 to position 0
          DWELL300                       ; pause another 300 msec
ENDWHILE                                 ; end indefinite loop

CLOSE                                    ; close UTC400 program buffer

CLOSE OPEN PROG 7 CLEAR                  ; open UTC400 program buffer #7

F10000                                   ; specify feedrate in encoder counts/sec
TA200                                    ; set acceleration time
TS50                                     ; set "S" curve time
WHILE(1<2)                               ; begin indefinite loop
          X(P1) Y(P1) Z(P1) A(P1)                  ; move all to first position
          DWELL300                                 ; pause for 300 msec
          X(P1*2) Y(P1*2) Z(P1*2) A(P1*2)          ; move all to 2nd position
          DWELL300                                 ; pause again
          X(P1*3) Y(P1*3) Z(P1*3) A(P1*3)          ; move all to 3rd position
          DWELL300                                 ; pause again
          A0 Z0 Y0 X0                              ; go back to 0
          DWELL300                                 ; pause once more...
ENDWHILE                                           ; end indefinite loop

CLOSE                                    ; close UTC400 program buffer

CLOSE OPEN PROG 8 CLEAR                  ; open UTC400 program buffer #8

TA200                                    ; set acceleration time
TS50                                     ; set "S" curve time
M1=0                                     ; clear velocity change flags
WHILE(1<2)                               ; begin indefinite loop
          F(P1*3)                                  ; 1st feedrate from variable
          M1=0                                     ; clear vel. change flags
          X300000 Y300000 Z300000 A300000          ; move all motors to 1st pos.
          F(P1)                                    ; read 2nd feedrate
          X400000 Y400000 Z400000 A400000          ; move all motors to 2nd pos.
          F(P1/3)                                  ; read 3rd feedrate
```

```
        X433333 Y433333 Z433333 A433333        ; move all motors to 3rd pos.
        M1=1                                   ; set 1st vel. change bit
        DELAY700                               ; pause for 700 msec
        F(P1*4)                                ; read fourth feedrate
        M1=3                                   ; set 2nd vel. change bit
        A0 Z0 Y0 X0                            ; move all back to 0
        DELAY700                               ; pause again...
ENDWHILE                                       ; end indefinite loop

CLOSE                                          ; close UTC400 program buffer


;******************** ALTDEST.UTC *********************
; This program allows altered destination in moves.
; The card is always moving toward an X position of
; P1 and a Y position of P2, but it does it in small
; increments so when P1 or P2 is changed, it will
; change direction within a few milliseconds.   P1
; and P2 can be changed from the host on the fly.
; *********************************************************

CLOSE
OPEN PROG 2
CLEAR
P9=100                          ; Vector distance per increment
P11=0                           ; Starting X position
P12=0                           ; Starting Y position
I3=10                           ; C.S accel time
I4=0                            ; C.S S-curve time
TM10                            ; Increment time in msec
WHILE (M1=1)                    ; Whatever condition you like
   P21=P1-P11                   ; X distance to go
   P22=P2-P12                   ; Y distance to go
   P0=SQRT(P21*P21+P22*P22)     ; Vector distance to go
   IF(P0>P9)                    ; If longer than increment
      P0=P9/P0                  ; Fraction of distance
      P11=P11+P21*P0            ; X increment
      P12=P12+P22*P0            ; Y increment
   ELSE                         ; Distance < increment
      P11=P1                    ; Just go to destination
      P12=P2
   ENDIF
   X(P11) Y(P12)                ; Make move
ENDWHILE
CLOSE
```

```
;********************** CHGSPEED.UTC *******************
; This program is an example of moving at high speed
; until some sort of external command, then reacting
; very quickly to the command, but changing speed
; very slowly.    It works by splitting the move into
; very small segments, and using WHILE loops to
; continue in one mode until the signal is received.
; ************************************************************

CLOSE
OPEN PROG 10
CLEAR
P4=150000
P5=3000
P6=600
INC(X)                          ; X-axis in incremental mode
TM(P5/P6)                       ; move segment time (5 msec here!)
TA(P5/P6)                       ; accel time (<= TM)
P3=P4/(P6*P6)                   ; multiplier to get distance
P1=0                            ; start at zero speed
P2=0                            ; start at zero distance
N10
WHILE (P0 != 0)                 ; P0=0 is signal to decelerate
   IF (P1<P6)                   ; not up to full speed?
      P1=P1+1                   ; increment to increase speed
      P2=P1*P3                  ; distance to travel in segment
   ENDIF
   X(P2)                        ; incremental move segment
ENDWHILE
WHILE (P0=0)                    ; P0=1 is signal to accelerate
   IF (P1>0)                    ; still moving?
      P1=P1-1                   ; decrement to decrease speed
      P2=P1*P3                  ; distance to travel in segment
   ENDIF
   X(P2)                        ; incremental move segment
ENDWHILE
GOTO 10                         ; loops back up when P0=1 again
CLOSE
```

```
;********************** CIRCTRY.UTC *********************
; This example contains subroutines that perform
; circular interpolation with linear moves.    The
; subroutine starting at N10000 does all the cal-
; culations for the circular move.    If the host
; program has not specified the center, the routine
; calls the subroutine at N20000 to compute the
; center point.    For each circularly interpolated
; move, the main program must specify the starting
; point, the ending point, the radius, the direc-
; tion, and the increment size.    Also, either the
; long-arc/short-arc toggle or the center point
; must be specified.    Since these specifications
; are variables that hold their value, any that do
; not change between moves do not have to be re-
; specified.    The main program in this example
; provides a thorough exercise of the subroutines.

; Most people will use UTC400's own circular moves
; for this type of move, but there are several
; reasons why you might want to use this type of
; routine.    First, this is not limited to the
; X, Y, and Z axes.    Second, with slight variations
; the moves can be made non-circular.    Also, it
; is possible to change directions on the fly by
; changing the sign of Q8.
; ************************************************************

CLOSE
OPEN PROG 1 CLEAR
; Main section of program is to exercise subroutine
TA50                              ; should be shorter than time for increment
TS0                               ; zero for best blending
F1000                             ; feedrate specification is easiest
X10000Y0                          ; linear move
; Long arc clockwise
Q1=10000 Q2=0 Q3=5000 Q4=-5000 Q5=5000
Q6=1 Q7=1 Q8=2 GOSUB 10000
; Linear move
X5000Y0
; Short arc clockwise
Q1=5000 Q2=0 Q3=10000 Q4=5000 Q5=5000
Q6=1 Q7=0 Q8=2 GOSUB 10000
; Short arc counterclockwise
```

```
Q1=Q3 Q2=Q4 Q3=12500 Q4=7500 Q5=2500
Q6=0 Q7=0 Q8=2 GOSUB 10000
; Long arc counterclockwise
Q1=Q3 Q2=Q4 Q3=10000 Q4=5000 Q5=2500
Q6=0 Q7=1 Q8=2 GOSUB 10000
; Half-circle clockwise
Q1=Q3 Q2=Q4 Q3=10000 Q4=0 Q5=2500
Q6=1 Q7=1 Q8=2 GOSUB 10000
; Half-circle counterclockwise; dist > 2R
Q1=Q3 Q2=Q4 Q3=10000 Q4=-5005 Q5=2500
Q6=0 Q7=0 Q8=2 GOSUB 10000
DELAY 500
X0Y-5000
DELAY 500
;Now an multi-part circle to test off angles
Q1=0 Q2=-5000 Q3=3535.5 Q4=-3535.5 Q5=5000
Q6=0 Q7=0 Q8=2 GOSUB 10000
Q1=Q3 Q2=Q4 Q3=3535.5 Q4=3535.5 GOSUB10000
Q1=Q3 Q2=Q4 Q3=-3535.5 Q4=3535.5 GOSUB10000
Q1=Q3 Q2=Q4 Q3=-3535.5 Q4=-3535.5 GOSUB10000
Q1=Q3 Q2=Q4 Q3=0 Q4=-5000 GOSUB10000
DELAY 500
X0Y0
RETURN

; UTC400 CIRCULAR ARC MOVE SUBROUTINE
; This subroutine performs moves on a circular arc
; automatically. The section of the program calling
; this subroutine simply needs to set the following
; variables as controlling parameters for the move:
; Q1: X-coordinate of the starting point of arc
; Q2: Y-coordinate of the starting point of arc
; Q3: X-coordinate of the ending point of arc
; Q4: Y-coordinate of the ending point of arc
; Q5: Radius of the arc
; Q6: 0 for counterclockwise; 1 for clockwise
; Q7: 0 for short arc; 1 for long arc;
;       -1 if have specified center
; Q8: subdivide size in degrees
; If host has the center location, the following
; can be specified also (with Q7=-1); otherwise
; they will be calculated by the routine
; Q9: X-coordinate of the center
; Q10: Y-coordinate of the center
```

```
N10000 ; CIRCULAR ARC MOVE SUBROUTINE
IF (Q7>-1)
      GOSUB 20000                 ; MUST COMPUTE CENTER
ENDIF
Q0=Q1-Q9                          ; PREP FOR ATAN2
Q26=ATAN2(Q2-Q10)                 ; ANGLE FROM CENTER TO START
Q0=Q3-Q9                          ; PREP FOR ATAN2
Q27=ATAN2(Q4-Q10)                 ; ANGLE FROM CENTER TO END
IF (Q6=0 AND Q27<Q26)
   Q27=Q27+360                    ; PROPER DIRECT.
ENDIF
IF (Q6=1 AND Q27>Q26)
   Q27=Q27-360                    ; PROPER DIRECT.
ENDIF
IF (Q6=0)                         ; IF COUNTERCLOCKWISE
   Q28=Q26+Q8                     ; INCREASING ANGLE
   WHILE (Q28<Q27)                ; NOT PAST END ANGLE
      Q11=Q9+Q5*COS(Q28)          ; X-COORD ON ARC
      Q12=Q10+Q5*SIN(Q28)         ; Y-COORD ON ARC
      X(Q11)Y(Q12)                ; MAKE PARTIAL MOVE
      Q28=Q28+Q8                  ; INCREMENT ANGLE
   ENDWHILE                       ; LOOP UNTIL END ANGLE
   X(Q3)Y(Q4)                     ; MAKE FINAL MOVE
ELSE                              ; CLOCKWISE ARC
   Q28=Q26-Q8                     ; DECREASING ANGLE
   WHILE (Q28>Q27)                ; NOT PAST END ANGLE
      Q11=Q9+Q5*COS(Q28)          ; X-COORD ON ARC
      Q12=Q10+Q5*SIN(Q28)         ; Y-COORD ON ARC
      X(Q11)Y(Q12)                ; MAKE PARTIAL MOVE
      Q28=Q28-Q8                  ; DECREMENT ANGLE
   ENDWHILE                       ; LOOP UNTIL END ANGLE
   X(Q3)Y(Q4)                     ; MAKE FINAL MOVE
ENDIF
RETURN

N20000                            ; SUBROUTINE TO CALCULATE CIRCLE CENTER
Q20=SQRT((Q3-Q1)*(Q3-Q1)+(Q4-Q2)*(Q4-Q2))      ; DISTANCE
Q21=Q5*Q5-Q20*Q20/4
IF (Q21<0)                        ; MOVE DISTANCE > 2R: MUST ADJUST
   Q21=0                          ; CENTER ON LINE FR START TO END
   Q5=Q20/2                       ; ADJUST RADIUS TO HALF DISTANCE
ENDIF
Q23=SQRT(Q21)                     ; DISTANCE FR MIDPOINT TO CENTER
Q0=Q3-Q1                          ; PREP FOR ATAN2
Q24=ATAN2(Q4-Q2)                  ; ANGLE FROM START TO END
```

```
Q0=Q20/2                          ; PREP FOR ATAN2
Q25=ATAN2(Q23)                    ; ANGLE OF CENTER OFF MIDLINE
IF (Q6=1 AND Q7=1 OR Q6=0 AND Q7=0)
   Q26=Q24+Q25                    ; ADD TO GET ANG FR START TO CENT
ELSE
   Q26=Q24-Q25                    ; SUB TO GET ANG FR START TO CENT
ENDIF
Q9=Q1+Q5*COS(Q26)                 ; CENTER X-COORD
Q10=Q2+Q5*SIN(Q26)                ; CENTER Y-COORD
RETURN
CLOSE


;******************** PLCDISP2.UTC *******************
; This PLC program monitors the motor position registers,
; scales them into counts, and sends the resulting values
; to the LCD display.

; This example illustrates several important features:
;     1. The use of D: M-variables for double-word values
;     2. The use of the servo cycle counter as a timer
;     3. The use of an open 24-bit M-variable containing
;          difference information to handle rollover properly
;     4. The display of messages on the LCD
;     5. The display of variable values on the LCD
;     6. The disabling of a condition that caused a display
;          command so that the next scan through the PLC, the
;          display command will not automatically be done again
;     7. How to convert motor positions to axis positions
;*********************************************************
; Setup and Definitions

CLOSE
; The PLC programs assume that the motors have assigned to
; axes as shown here, although they do not assume what the
; coefficient in the assignment is; they use the actual
; variable.
#1->1000X
#2->1000Y
#3->1000Z
#4->1000A

M85->C6C0,S                       ; Open memory for start time
M86->C6C1,S                       ; Open memory for elapsed time
M87->C680,1                       ; Bit that is zero on poweron/reset
```

```
;*****************************************************
; PLC programs to do the task
;*****************************************************

; This first PLC program is just executed once on power-up
; or reset to set up the display function.    By disabling
; itself, it keeps from executing repeatedly.

OPEN PLC 1 CLEAR
DISP 0,"#1 POS =                  "   ; Display labels
DISP 20,"#2 POS =                 "
DISP 40,"#3 POS =                 "
DISP 60,"#4 POS =                 "
P86=333                               ; 1/3 second
DISABLE PLC 1
CLOSE

OPEN PLC 2 CLEAR
IF (M87=0)                            ; Need to mark starting time?
   M85=M0                             ; Record starting time
   M87=1                              ; No longer first time thru
ENDIF
; These statements take the actual position, add in the
; position bias, then scale into counts and user units
P161=(M161+M164)/M191
P261=(M261+M264)/M291
P361=(M361+M364)/M393
P461=(M461+M464)/M491

M86=M0-M85                            ; Compute time since last display
IF (M86>P86)                          ; More than update time?
   DISP    9, 11.0, P161              ; Display the positions
   DISP 29, 11.0, P261
   DISP 49, 11.0, P361
   DISP 69, 11.0, P461
   M87=0                              ; So will mark starting time
ENDIF
CLOSE

I6=3
ENAPLC 1,2
```

```
;********************** DAISY.UTC *********************
; This program "draws" a daisy-shaped figure in the
; XY-plane.    It illustrates circular interpolation,
; the use of subroutines, and output setting.
; **************************************************************

OPEN PROG 10 CLEAR
F20000                          ; Set feedrate
TA 10                           ; Accel time
LIN                             ; Start with linear moves
GOTO 1000                       ; Jump over subroutines
N62                             ; Label to start subroutine
DWELL01
M1=1                            ; Turn off output
F2000                           ; Faster feedrate
DWELL 100                       ;
RET                             ; Return to line after GOSUB
N63                             ; Label to turn on subroutine
DWELL01
M1=0                            ; Turn on output
F10000                          ; Slower feedrate
DWELL 100
RET                             ; Return to line after GOSUB
N1000                           ; Start of main program area
GOSUB 62
X2082.4000 Y1778.0000           ; Linear offset move
GOSUB 63                        ; To turn on output (eg laser)
; Full circle move (end point = start point)
CIR 1 X2082.4000 Y1778.0000 I-606.2000 J0.0000
LIN
GOSUB 62                        ; To turn off output
X1880.9716 Y2232.2616
GOSUB 63
; First "petal"
CIR 2 X1070.7003 Y2233.1163 I-404.7716 J345.5384
LIN
GOSUB 62
X1930.4225 Y1383.0427
GOSUB 63
; Second "petal"
CIR 2 X1937.9537 Y2179.4222 I345.5775 J394.9573
LIN
GOSUB 62
X1012.0638 Y2163.0278
```

```
GOSUB 63
; Third "petal"
CIR 2 X1013.4259 Y1394.1638 I-335.6638 J-385.0278
LIN
GOSUB 62
X1071.4245 Y1323.7339
GOSUB 63
; Fourth "petal"
CIR 2 X1885.0458 Y1318.9081 I404.7755 J-345.5339
LIN
GOSUB 62
X74.0000 Y178.2000
GOSUB 63
X2870.6000 Y178.2000
X2870.6000 Y3391.2000
X74.0000 Y3391.2000
X74.0000 Y178.2000
GOSUB 62
X0.0000 Y0.0000
CLOSE


;********************** PLCIO.UTC *********************
; UTC400 I/O PLC Program Examples
; These examples show how you can use PLC programs
; with UTC400's general-purpose I/O
; ; *************************************************************

; Definitions and Setup

CLOSE                          ; To ensure these are on-line
M20..39->*                     ; Self-referenced flag variables

; PLC Program
CLOSE                          ; Make sure other buffers closed
OPEN PLC 5                     ; Open buffer for editing
CLEAR                          ; Erase existing contents
; This first branch sets a variable based on
; the state of Machine Input 1.    This variable
; could be a destination, speed, time, etc.
IF (M11=0)                     ; Machine Input 1 true?
   P1000=5000                  ; If so, set to this value
ELSE
   P1000=500                   ; If not set to this value
ENDIF
```

```
; This next branch increments a counter every time
; Machine Input 2 goes true (edge triggered)
IF (M12=0)                          ; Machine Input 2 true?
   IF (M22=0)                       ; Not true last time thru?
      P8=P8+1                       ; Have rise edge, so increment
      M22=1                         ; Note as true for next time thru
   ENDIF
ELSE                                ; MI2 is not true
   M22=0                            ; Note as not true for next time thru
ENDIF
; This next branch sets an output only if all four
; of a set of flag variables are true
IF (M23=1 AND M24=1)                ; First two flags true?
   IF (M25=1 AND M26=1)             ; Second two flags true?
      M1=0                          ; Then set output
   ENDIF
ELSE
   M1=1                             ; Otherwise clear output
ENDIF
CLOSE                               ; Close the buffer
ENAPLC 5                            ; Enable operation of program


;******************** PLCPULSE.UTC ********************
; This example uses a PLC program to pulse Machine
; Output 1 every time a distance of 250 counts is
; passed (vector distance) by the axes using
; Encoders 1 and 2.    The pulse is turned off the
; next time through the PLC (~1 msec).
; **********************************************************

; Definitions and setup

M20->*                              ; Self-referenced flag var

; PLC program contents
CLOSE                               ; Close any open buffer
OPEN PLC 2                          ; Open buffer for editing
CLEAR                               ; Erase old contents
IF (M20=1)                          ; M20 sets the mode
   IF (M1=1)                        ; Output now off
      P102=(M101-P101)*(M101-P101)       ; X-dist squared
      P202=(M201-P201)*(M201-P201)       ; Y-dist squared
      IF (SQRT(P102+P202)>250)           ; dist > 250?
         M1=0                       ; Set output
         P101=M101                    ; Reset 'last pos' vars
         P201=M201                    ; Ditto
      ENDIF
   ELSE
      M1=1                            ; Clear output next time thru
   ENDIF
ENDIF
CLOSE                                 ; Close the buffer
ENAPLC 2                              ; Enable the action


;*********************** JOGSWTCH.UTC *************************
; This program shows how to use the thumbwheel input
; lines to create jog switches that are dedicated to
; a particular motor.
; Note in particular the use of "latching flags" so
; that commands are only given on a change of state
; of the input line, making them edge-triggered.
; **********************************************************

; Setup and definitions

CLOSE

; PLC Program to implement function
OPEN PLC 15 CLEAR
IF (M50=0 AND P50 != 0)               ; Motor 1 jog plus switch on
   P50=0                              ; Set latching flag
   CMD"#1J+"                          ; Issue command
ENDIF                                 ; Motor 1 jog plus switch off
IF (M50=1 AND P50 != 1)               ; Motor 1 jog plus switch off
   P50=1                              ; Set latching flag
   CMD"#1J/"                          ; Issue command
ENDIF                                 ; Motor 1 jog plus switch off
IF (M51=0 AND P51 != 0)               ; Motor 1 jog minus switch on
   P51=0                              ; Set latching flag
   CMD"#1J-"                          ; Issue command
ENDIF                                 ; Motor 1 jog minus switch off
IF (M51=1 AND P51 != 1)               ; Motor 1 jog minus switch off
   P51=1                              ; Set latching flag
   CMD"#1J/"                          ; Issue command
ENDIF                                 ; Motor 1 jog minus switch off
```

```
;************************* INVERT.UTC *************************
; This example shows how UTC400 can be used to perform the inverse
; kinematic problem -- the conversion of user coordinates (usually
; cartesian) to motor or joint coordinates.   This program works only
; in two axes -- it shows the conversion of XY data into radial and
; angular axes.   This would permit the cartesian specification of
; points on a rotary table with a radial arm.   The concept can be
; extended to 6 or even 8 axes -- the calculations simply get more
; involved.   Note that logical branching, not just pure computation,
; can be done in the conversion (here to handle C-axis rollover).
;****************************************************************
; Set-up and definitions
#3->5000U                 ; Radial axis definition; units here should
                          ; be the same as for X and Y user units
#4->13.889C               ; Angular (theta) axis definition; units are
                          ; degrees or radians, but must match I15.
                          ; Here 5000 cts/rev yields 13.889 cts/degree.
Q21=5                     ; X zero position in U-axis coordinates
Q22=0                     ; C-axis offset


;****************************************************************
; Program that performs the inverse kinematic problem and actually
; does the move in joint coordinates.   This is called as a subprogram
; and expects 'X' data (in Q124) and 'Y' data (in Q125).   It converts
; this data into radius and angle coordinates (the inverse kinematic
; problem) and commands the move -- here of the U (radius) and C
; (angle) axes.   Q21 and Q22 should be preset as offsets for the U
; and C axes, respectively (if needed).   The move is broken into
; small segments and the conversion is done at each segment boundary.
; The resulting joint trajectories are then splined together so the
; movement is smooth and accurate.
; **********************************************************

CLOSE
OPEN PROG 500 CLEAR
READ(X,Y)                 ; X endpoint->Q124; Y endpoint->Q125
Q12=Q10*Q11/1000          ; Distance per segment (user units)
Q14=Q24                ; X working point is X starting point
Q15=Q25                ; Y working point is Y starting point
SPLINE                    ; Cubic spline mode
TA(Q11)                   ; Segment time
Q16=0                     ; To start loop
WHILE (Q16<1)
```

```
                                              ; This section breaks the linear move into
                                              ; small segments.   The coordinate transform-
                                              ; ation is done at each segment boundary.
                                              ; This next calculation should only be done
                                              ; on "feedrate axes".
  Q34=Q124-Q14                                ; X distance to go
  Q35=Q125-Q15                                ; Y distance to go
  Q36=SQRT(Q34*Q34+Q35*Q35)                   ; Vector distance to go
  IF (Q36>Q12)                                ; Longer than segment size?
    Q37=Q12/Q36                               ; Fraction of distance to go
                                              ; The interpolation below should be done on
                                              ; all axes, feedrate or not
    Q14=Q14+Q34*Q37                           ; X location at end of next segment
    Q15=Q15+Q35*Q37                           ; Y location at end of next segment
  ELSE                                        ; Last segment of move
    Q14=Q124                                  ; End of segment is end of move
    Q15=Q125                                  ; End of segment is end of move
    Q24=Q124                                  ; X starting point for next move
    Q25=Q125                                  ; Starting point for next move
    Q16=1                                     ; To mark last segment
  ENDIF
; Now convert to joint coordinates
  Q17=Q21-SQRT(Q14*Q14+Q15*Q15)   ; Radial axis coordinate
  Q28=Q18                               ; Last angular coordinate
  Q0=Q14                                      ; Second argument for ATAN2
  Q18=ATAN2(Q15)+Q22                          ; Angular axis coordinate
  IF (Q18-Q28>180)
    Q8=Q8-360                                 ; Rollover going negative
  ENDIF
  IF (Q18-Q28<-180)
    Q8=Q8+360                           ; Rollover going positive
  ENDIF
  U(Q17)C(Q18+Q8)                             ; Make segment move
ENDWHILE
CLOSE
```

```
;*********************************************************************
; The following is a sample main program that can use the above
; program.    This program simply specifies a square in user coord-
; inates.    The inverse kinematic problem is basically invisible to
; it.    This shows that the actual inverse kinematics can be largely
; hidden from the final parts programmer.
; ***************************************************************

; Start-up data: needed only before the first running.
Q24=-Q21                          ; Starting X-value
Q25=-Q22                          ; Starting Y-value
Q8=0                              ; Starting C-axis rollover

OPEN PROG 25 CLEAR
F(Q10)
Q10=2                             ; Desired feedrate (XY units/sec)
Q11=30                            ; Spline piece time
CALL 500 X2 Y2                    ; Move to start of square
DWELL100
M1=0                              ; Turn on output (e.g. laser)
CALL 500 X2 Y-2                   ; Do first side of square
DWELL100
CALL 500 X-2 Y-2                  ; Do second side of square
DWELL100
CALL 500 X-2 Y2                   ; Do third side of square
DWELL100
CALL 500 X2 Y2                    ; Do last side of square
DWELL100
M1=1                              ; Turn off output
CALL 500 X-5 Y0                   ; Return to retract position
CLOSE
```

```
;**************************** GCODE EXAMPLE **************************
;* This program demonstrates use use of G-codes in a UTC400 *
;* program.   The program uses a few commonly used G-codes to cut *
;* a part (an L-shaped bracket) using 3 axes.   In UTC400, G-codes *
;* actually treated as subroutine calls which contain true UTC400 *
;* commands.   However, you use the G-codes exactly the same way *
;* you would in a true G-code machine, such as a CNC machine. *
;* The fact that they are subroutine calls gives you the added *
;* flexibility to customize your G-codes to perform whatever *
;* tasks you need. *
;*********************************************************************

CLOSE                   ;close any previously opened buffer
OPEN PROG 6 CLEAR       ;open motion program buffer 6 and clear it out
G20                     ;select inches for units
G90                     ;select absolute mode
G94                     ;select inches per minute mode
F100                    ;set feedrate of 100 inches per minute
G00 X1 Y4               ;go RAPIDly to starting position
M3 S1800                ;start spindle at 1800 RPM clock wise
G01 Z.1                 ;lower the cutting tool
M08                     ;turn coolant on
Z0                      ;plunge cutting tool into material
Y12                     ;cut
G02 X2 Y13 I1 J0        ;do rounded corner
G01 X3                  ;cut across
G02 X4 Y12 I0 J-1       ;do rounded corner
G01 Y6                  ;cut downward
G03 X7 Y4 I3 J0
G01 X13                 ;cut across
G02 X14 Y3 I0 J-1
G01 Y2                  ;cut downward
G02 X13 Y1 I-1 J0
G01 X4                  ;cut across backward
G02 X1 Y4 I0 J4         ;last rounded edge
G01 Z.1                 ;remove cutting tool from material
X3 Y3                   ;move to next point
Z0                      ;plunge cutting tool into material
G02 I.5 J.5             ;cut first hole
G01 Z.1                 ;remove cutting tool from material
X2 Y11                  ;move to next point
Z0                      ;plunge cutting tool into material
G02 I.5 J.5             ;cut first hole
G01 Z.1                 ;remove cutting tool from material
```

```
X12 Y2              ;move to next point
Z0                  ;plunge cutting tool into material
G02 I.5 J.5         ;cut first hole
G01 Z.1             ;remove cutting tool from material
M09                 ;turn off coolant
Z1                  ;lift cutting tool
G00 X0 Y0           ;return to home position
M05                 ;turn off spindle
CLOSE               ;close this program buffer
```

Contents

UTSD105 Hardware Specifications

I. Electric Specifications

UTSD105-XX Type	Unit	-03	-05	-07	-10	-15	-20
Input Voltage	VAC	220V(±10%)50/60Hz					
AC Servo Motor Voltage	VAC	220V(±10%)50/60Hz(3-phase)					
Nominal DC-BUS Voltage	VDC	310					
DC-BUS Voltage Range	VDC	90~350					
Output Voltage	VRMS	0~250					
Cons. Phase Current (±10%)	ARMS	2.5	3.5	5.5	7.1	11	14
Max. Peak Current (±10%)	ARMS	6.3	8.8	13.4	17.7	26.8	35.7
Regeneration Resistor		50Ω/100W			20Ω/500W		
Protections	Over voltage/Undervoltage/Over current/Over temperature						
Control	IGBT PWM(3-phase)						
Cooling	Air(internal fan)w/Heat Sink						

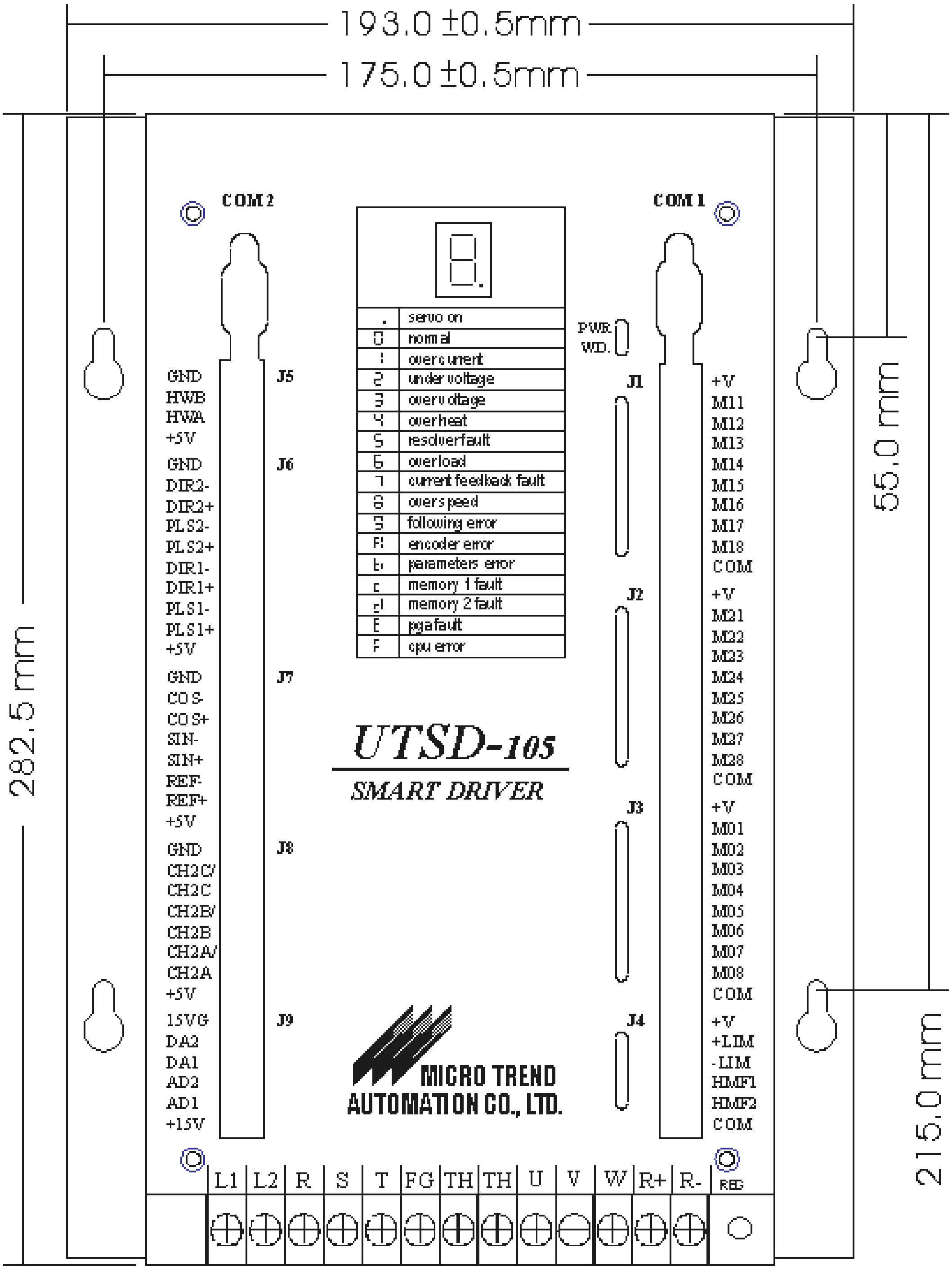
II. Environmental Specifications

UTSD105-XX Type	Unit	-03	-05	-07	-10	-15	-20
Ambient temperature	°C	+0~+45°C					
Storahe temperature	°C	-25~+70°C					
Humidity	%	10~90RH or less(non-condensing)					
Altitud		Max.1000m					

III. Mechanical Specifications

UTSD105-XX Type	Unit	-03	-05	-07	-10	-15	-20
Mounting		Panel					
Dimensions	mm	282.5*193*111			316.5*193*140		

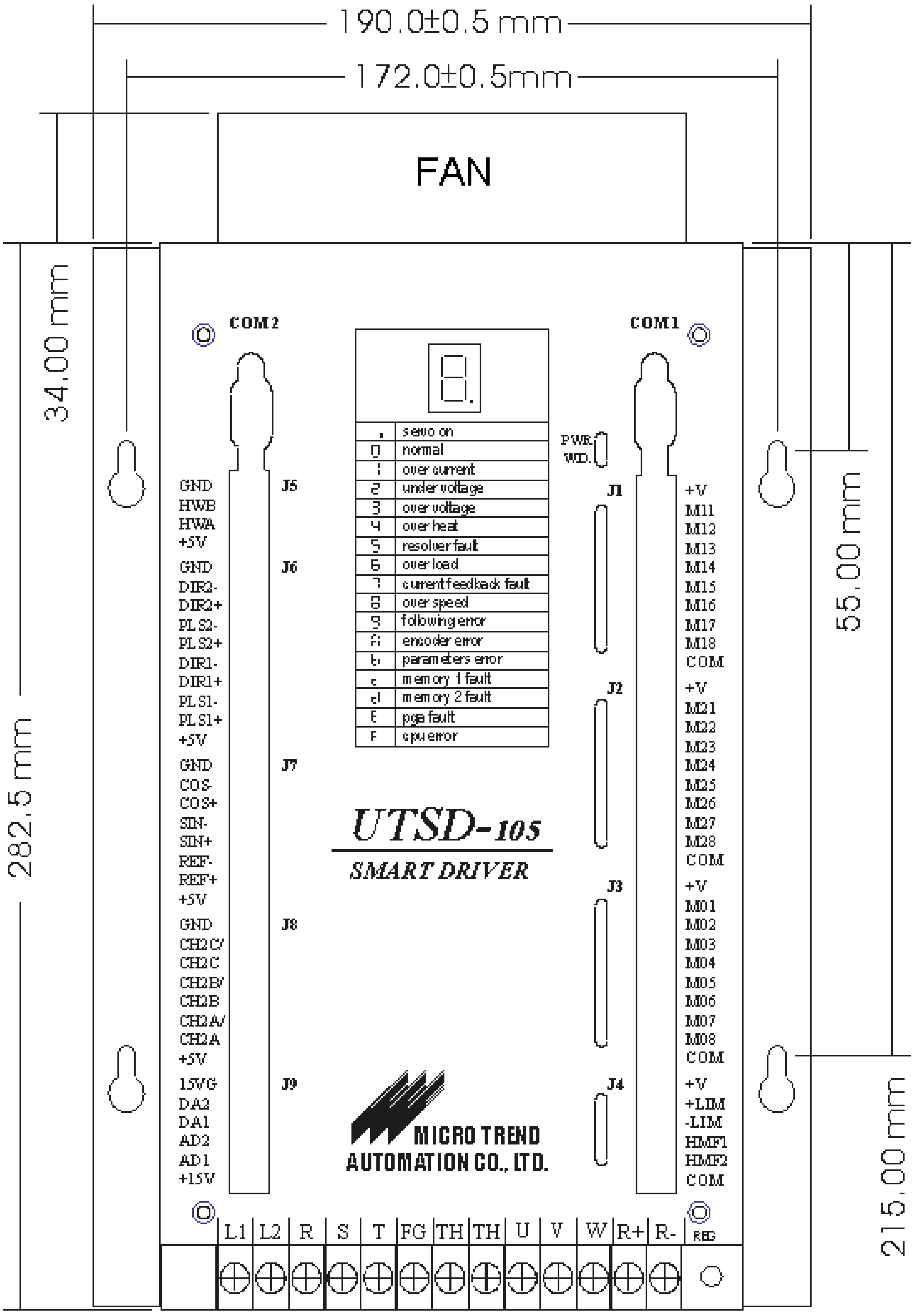
1. UTSD105-(03/05/07/10) Dimension Diagram



.	servo on
0	normal
1	overcurrent
2	under voltage
3	overvoltage
4	overheat
5	resolverfault
6	overload
7	current feedback fault
8	overspeed
9	following error
A	encoder error
b	parameters error
c	memory 1 fault
d	memory 2 fault
E	pgafault
F	cpu error

Height:111.0mm

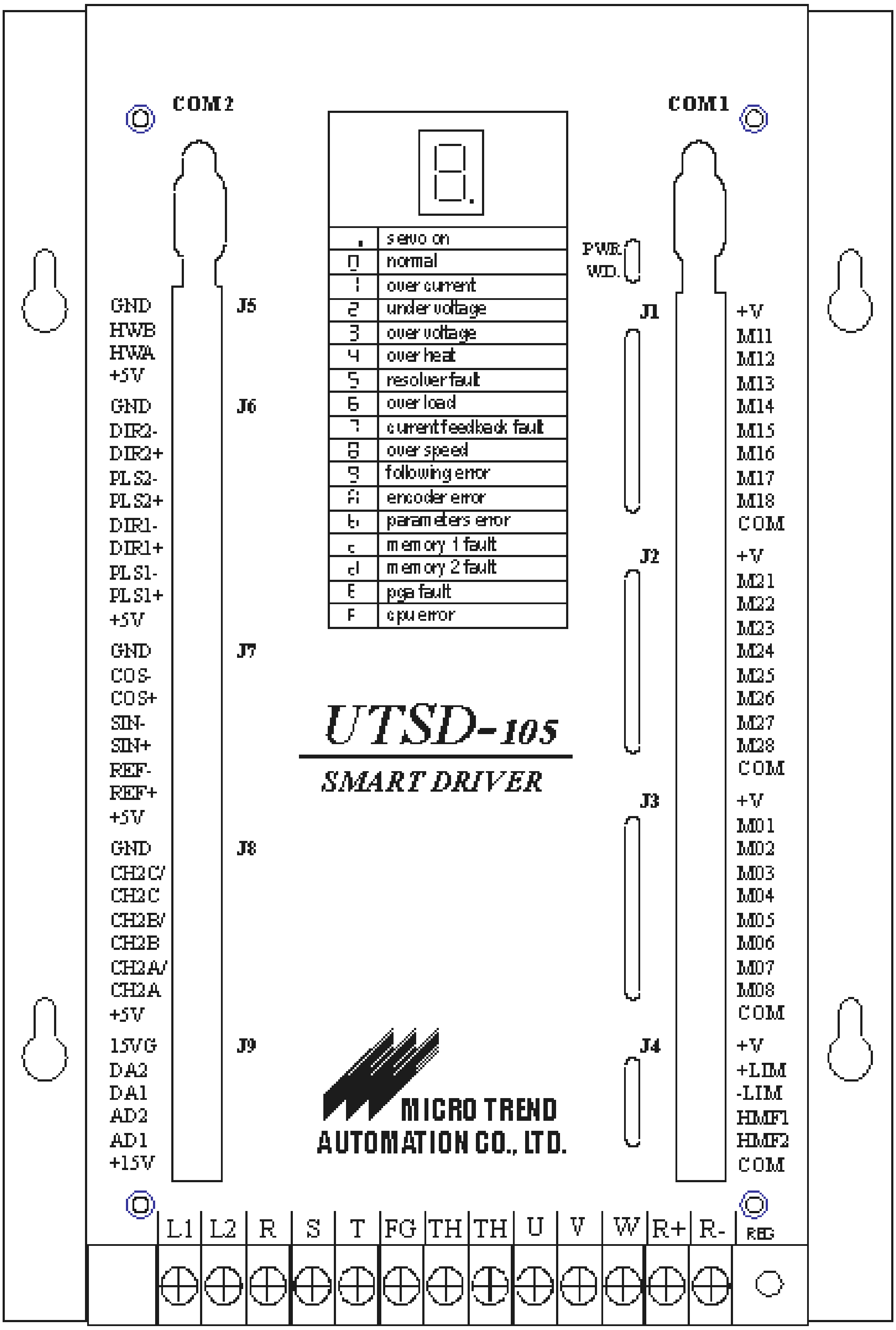
2. UTSD105-(15/20) Dimension Diagram



.	servo on
0	normal
1	over current
2	under voltage
3	over voltage
4	over heat
5	resolver fault
6	over load
7	current feedback fault
8	over speed
9	following error
A	encoder error
b	parameters error
c	memory 1 fault
d	memory 2 fault
E	pga fault
F	cpu error

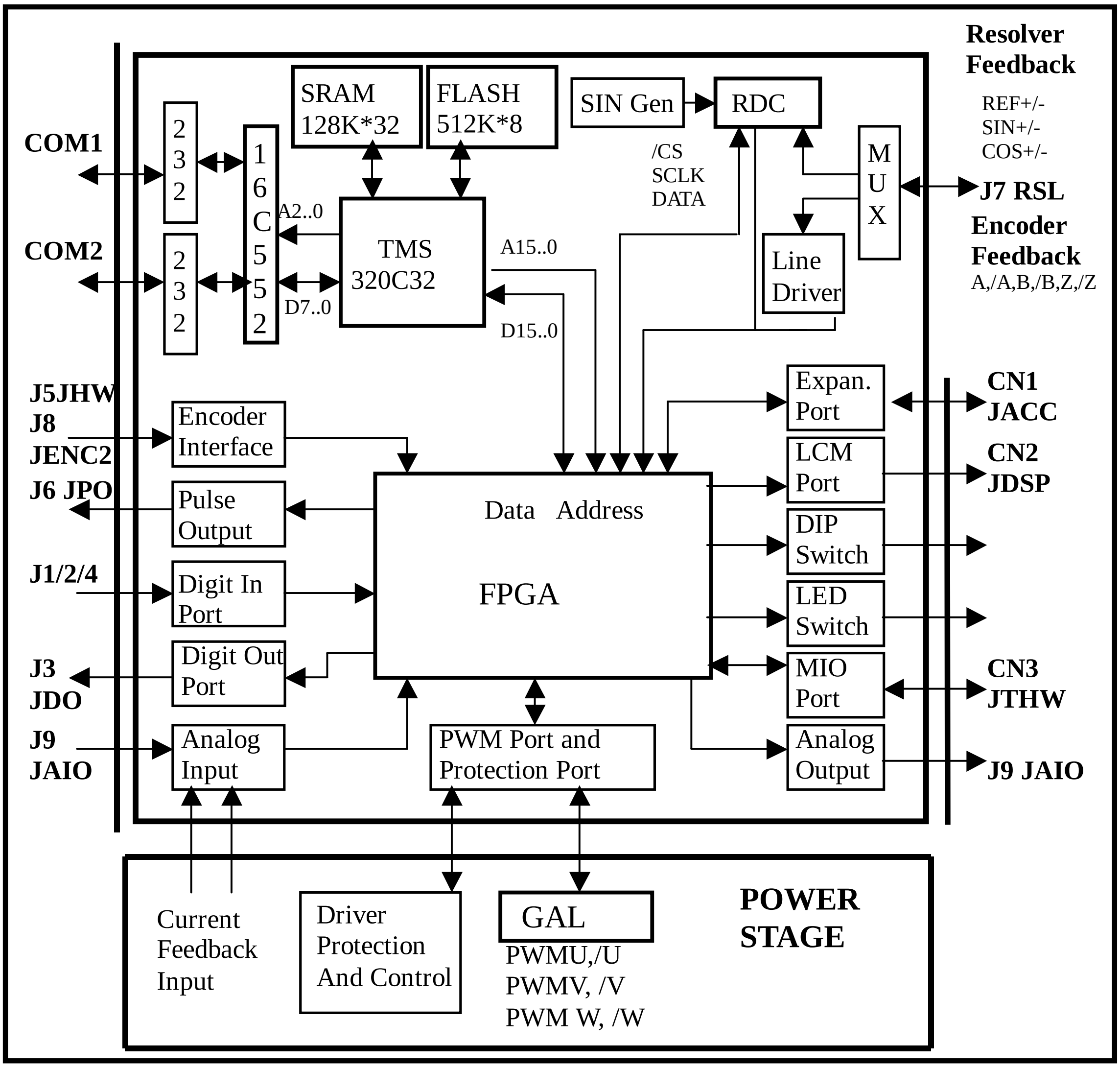
Height：140.0mm

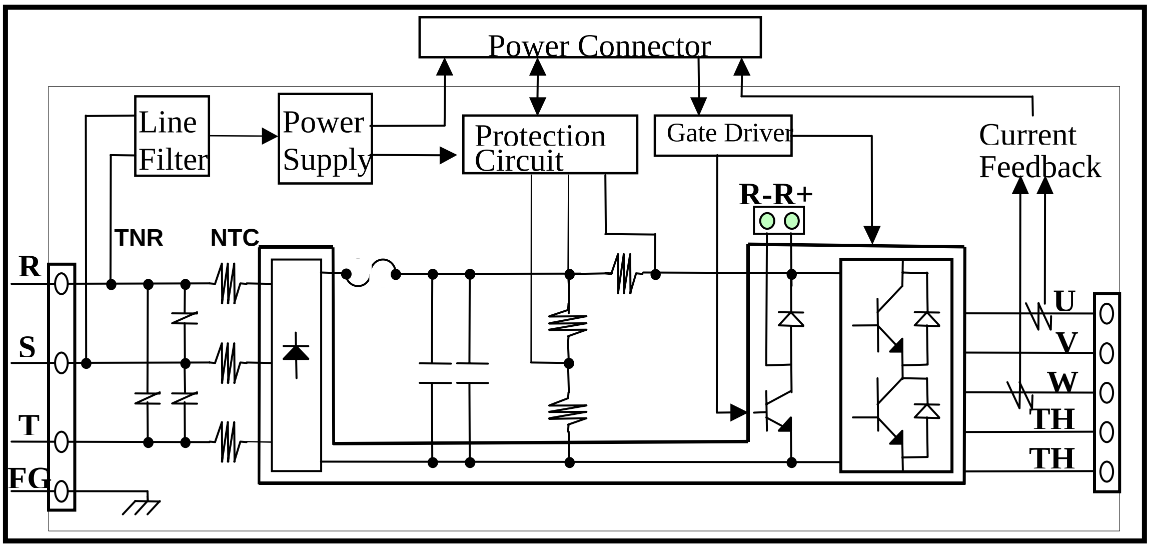
3. UTSD105 Front View



.	servo on
0	normal
1	over current
2	under voltage
3	over voltage
4	over heat
5	resolver fault
6	over load
7	current feedback fault
8	over speed
9	following error
A	encoder error
b	parameters error
c	memory 1 fault
d	memory 2 fault
E	pga fault
F	cpu error

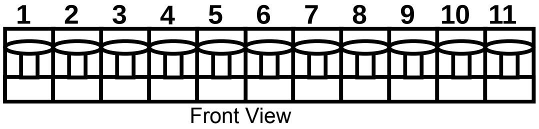
IV. Construction





V. Connector Pinouts

- **TB1 (11-PIN Terminal Block)**

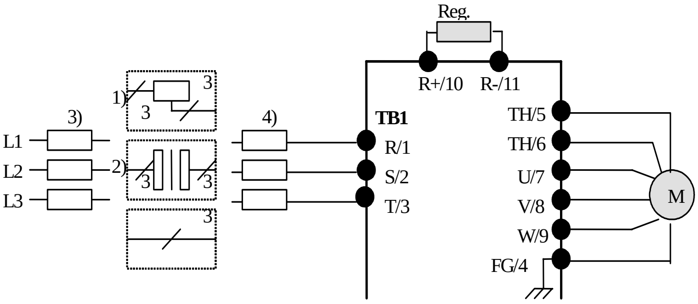


Pin	Signal	Descrptions
1 2 3	R S T	AC Power input 220VAC 60Hz 220V/3-phase is recommended
4	FG	Frame ground (resistor100Ω)
5 6	TH TH	Thermo Relay input (Connect to motor Thermo Switch)
7 8 9	U V W	Power of Motor (Connect to motor U/V/W)
10 11	R+ R-	External regeration resistor (50Ω/100 W)

Status monitoring

CPU	PWR	Green Power LED
	W.D	Red CPU LED
Power stage	PWR	Green Power LED
	REG	Reg.LED

The Power offer diagram

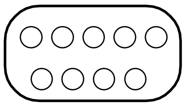


1) autotransformer
2) isolating transformer
3) 4) the size of the fuse must fit for the power of the transformer

> The motor's Thermal relay is Normal Close without polarity. Please make a short circuit on two contacts with a small wire, if the motor is without a Thermal relay

> The regeneration resistor must be installed by the customer.

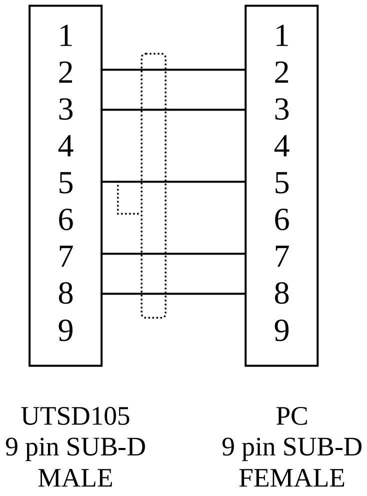
- **JCOM1 (9-PIN D-sub CONNECTOR)**

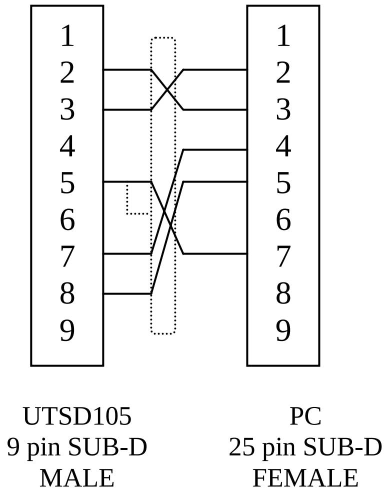


Front View

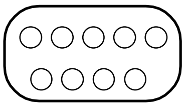
PIN#	SYMBOL	FUNCTION	Description	UTSD105 <---> PC
2	TXD	OUTPUT	Transmit data	2 (TXD) ---> 2 (RXD)
3	RXD	INPUT	Receive data	3. (RXD) <--- 3 (TXD)
5	GND	COMMON	Common	5. (GND) <-->5 (GND)
7	CTS	CTS	Clear to send	7. (CTS) <---- 7 (RTS)
8	RTS	RTS	Request to send	8. (RTS) ----> 8 (CTS)
9	+5V	+5V		

Remark: 9P D-sub connector, one to one to PC. The default setting is 38400, N, 8, 1





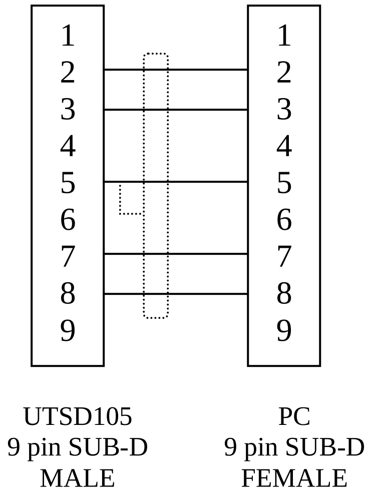
- **JCOM2 (9-PIN D-sub CONNECTOR)**

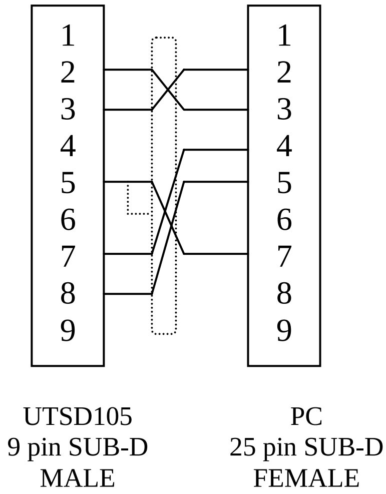


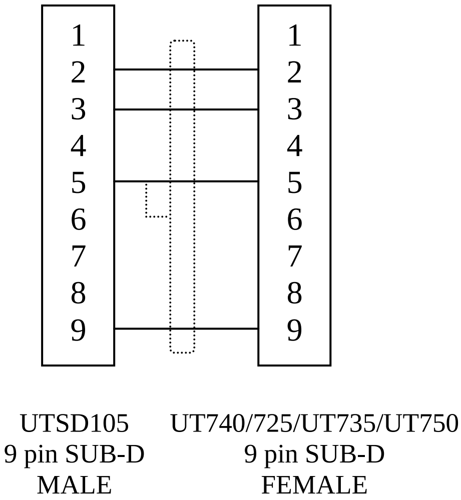
Front View

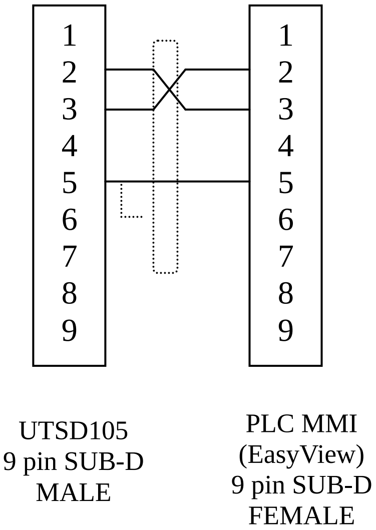
PIN#	SYMBOL	FUNCTION	Description	UTSD105 <---> PC
2	TXD	OUTPUT	Transmit data	2 (TXD) ---> 2 (RXD)
3	RXD	INPUT	Receive data	3. (RXD) <--- 3 (TXD)
5	GND	COMMON	Common	5. (GND) <-->5 (GND)
7	CTS	CTS	Clear to send	7. (CTS) <---- 7 (RTS)
8	RTS	RTS	Request to send	8. (RTS) ----> 8 (CTS)
9	+5V	+5V		

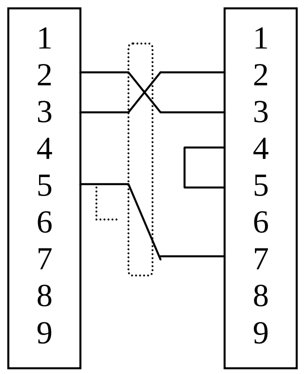
Remark: 9P D-sub connector, one to one to PC. The default setting is 38400, N, 8, 1





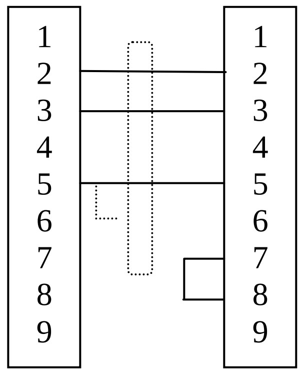






UTSD105
9 pin SUB-D
MALE

PLC MMI(Hitech)- FEMALE
(Digital ProFace)-MALE
(Fuji) -MALE
25 pin SUB-D



UTSD105
9 pin SUB-D
MALE

PLC MMI(Hitech)
9 pin SUB-D
FEMALE

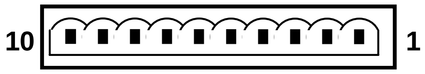
Panel setting Facon FB series

Baut rate 19200,N,8,1

(Fuji // ProFace Baut rate 19200,E,7,1(I2=$52))

Parameter	PLC		UTC-series
R(HR)	0- 1023	→	P0-1023
R(HR)	1024- 2047	→	Q0-1023
M	0- 1023	→	M0-1023(for bit)
WM	0- 1023	→	M0-1023(for value)

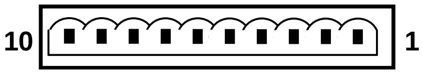
- **J1 JDI1 (10-PIN Terminal Block CONNECTOR)**



Front View

PIN#	SYMBOL	FUNCTION	Description	NOTES
1	+V	POWER	+13V POWER	+13V isolation power
2	M11	INPUT	Machine Input 11	
3	M12	INPUT	Machine Input 12	
4	M13	INPUT	Machine Input 13	
5	M14	INPUT	Machine Input 14	
6	M15	INPUT	Machine Input 15	
7	M16	INPUT	Machine Input 16	
8	M17	INPUT	Machine Input 17	
9	M18	INPUT	Machine Input 18	
10	COM	COMMON	+13VG Common	+13VG common

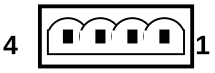
- **J2 JDI2 (10-PIN Terminal Block CONNECTOR)**



Front View

PIN#	SYMBOL	FUNCTION	Description	NOTES
1	+V	POWER	+13V Power	+13V isolation power
2	M21	INPUT	Machine Input 21	
3	M22	INPUT	Machine Input 22	
4	M23	INPUT	Machine Input 23	
5	M24	INPUT	Machine Input 24	
6	M25	INPUT	Machine Input 25	
7	M26	INPUT	Machine Input 26	
8	M27	INPUT	Machine Input 27	
9	M28	INPUT	Machine Input 28	
10	COM	COMMON	+13VG Common	+13VG Common

- **J5 JHW (4-PIN Terminal Block CONNECTOR)**

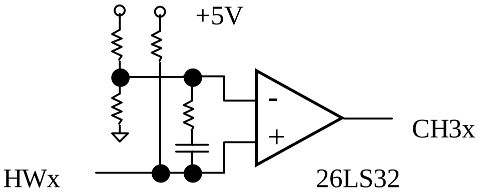


Front View

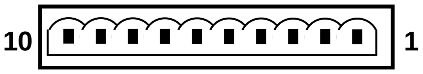
PIN#	SYMBOL	FUNCTION	Description	NOTES
4	GND	COMMON	+5V Common	
3	HWB	INPUT	HW B Channel Input	
2	HWA	INPUT	HW A Channel Input	
1	+5V	POWER	+5V Power	

UTC-series has one ports encoder input. It can be line driver or open collector.

Handwheel (x=A,B)



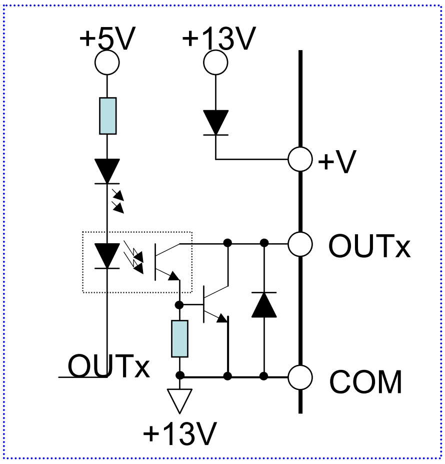
- **J3 JDO (10-PIN Terminal Block CONNECTOR)**



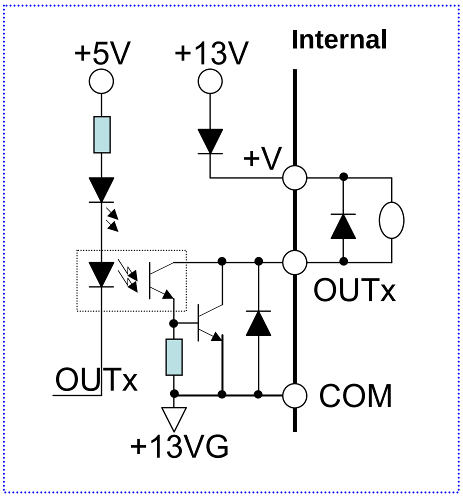
Front View

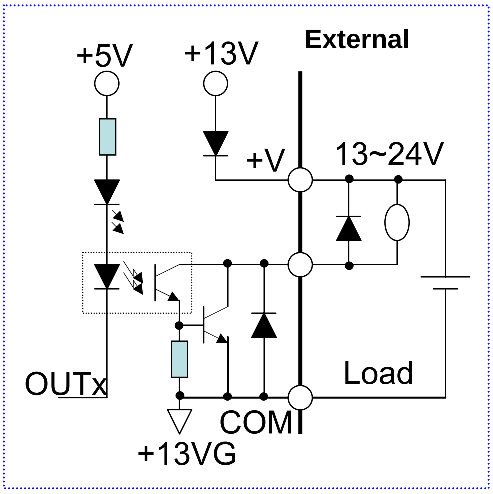
PIN#	SYMBOL	FUNCTION	Description	NOTES
1	+V	POWER	+13V Power	+13V isolation power
2	MO1	OUTPUT	Machine Output 1	
3	MO2	OUTPUT	Machine Output 2	
4	MO3	OUTPUT	Machine Output 3	
5	MO4	OUTPUT	Machine Output 4	
6	MO5	OUTPUT	Machine Output 5	
7	MO6	OUTPUT	Machine Output 6	
8	MO7	OUTPUT	Machine Output 7	
9	MO8	OUTPUT	Machine Output 8	
10	COM	COMMON	+13VG Common	+13VG Common

UTSD105 offers on board 8 photo couple isolation digital output points.. There is a isolated +13V on board. The customer can use this +13V or external +24V as follows

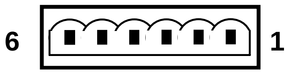


Suggested connection method:





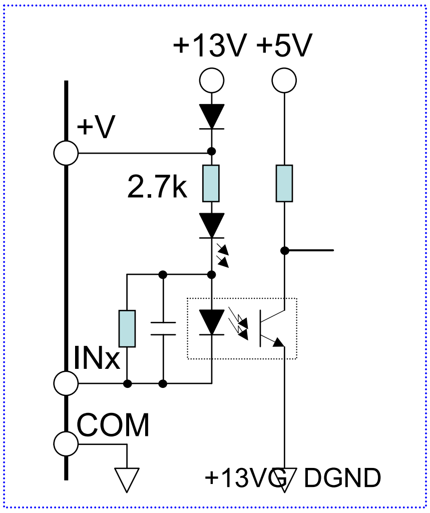
- **J4 JDDI (6-PIN Terminal Block CONNECTOR)**



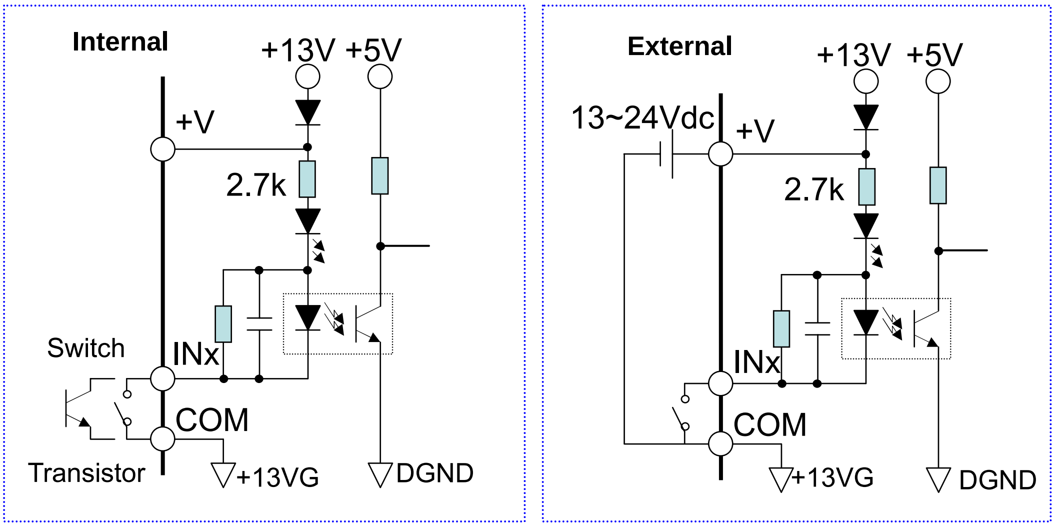
Front View

PIN#	SYMBOL	FUNCTION	Description	NOTES
1	+V	POWER	+13V Power	+13V isolation power
2	+LIM	INPUT	Positive Limit Input	
3	-LIM	INPUT	Negative Limit Input	
4	HMF1	INPUT	Home Flag #1	
5	HMF2	INPUT	Home Flag #2	
6	COM	COMMON	+13VG COMMON	+13VG Common

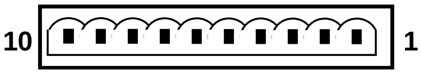
UTSD105 offers on board 20 photo-isolation digital inputs. Among these points, there are 16 (J1, JDI1,J2,JDI2) general purposed inputs and 4(+LIM/-LIM/HMF1/HMF) dedicated inputs. There is an isolated +13V on board. The customer can use this +13V or external +24V as follows:



suggested connection method:



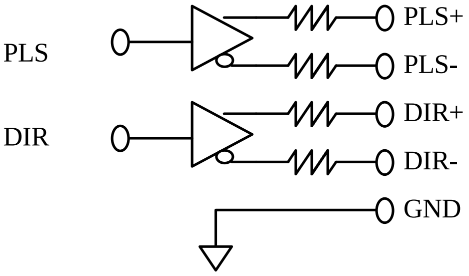
- **J6 JPO (10-PIN Terminal Block CONNECTOR)**



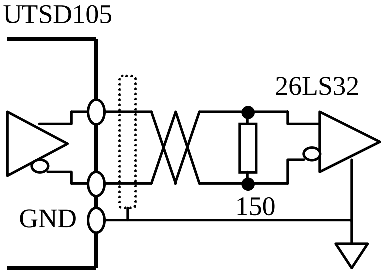
Front View

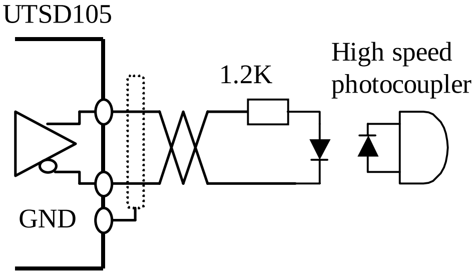
PIN#	SYMBOL	FUNCTION	Description	NOTES
10	GND	COMMON	+5V Common	
9	DIR2-	OUTPUT	Neg Dir. Output	CH#2
8	DIR2+	OUTPUT	Pos Dir. Output	CH#2
7	PLS2-	OUTPUT	Neg Pulse Output	CH#2
6	PLS2+	OUTPUT	Pos Pulse Output	CH#2
5	DIR1-	OUTPUT	Neg Dir. Output	CH#1
4	DIR1+	OUTPUT	Pos Dir. Output	CH#1
3	PLS1-	OUTPUT	Neg Pulse Output	CH#1
2	PLS1+	OUTPUT	Pos Pulse Output	CH#1
1	+5V	POWER	+5V Power	

Pulse Output

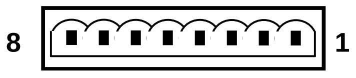


***Example* Pulse Output:**





- **J7 JRSL (8-PIN Terminal Block CONNECTOR)**



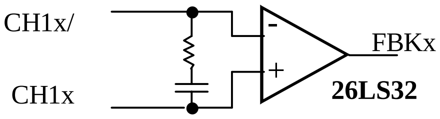
Front View

PIN#	SYMBOL	FUNCTION	Description	NOTES
8	GND	COMMON	+5V Common	
7	COS-	INPUT	Feedback Input	
6	COS+	INPUT	Feedback Input	
5	SIN-	INPUT	Feedback Input	
4	SIN+	INPUT	Feedback Input	
3	REF-	INPUT	Feedback Input	
2	REF+	INPUT	Feedback Input	
1	+5V	POWER	+5V Power	

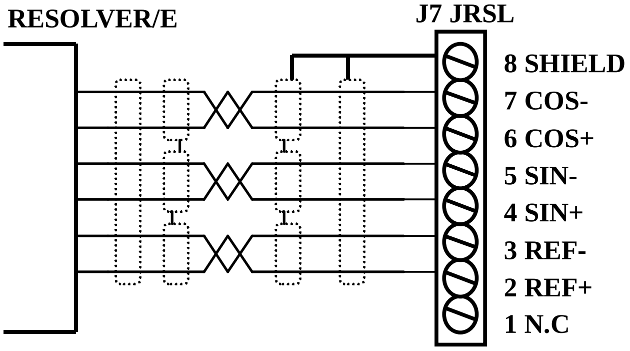
UTSD105 offer resolver and encoder feed back ports,
Resolver // Encoder feed back diagram **:**

J8		J7	
Encoderk	**Resolver**	Encoder	**Resolver**
8. DNGD	**8. DGND**	8. DGND	**8. Shield**
7. CHW/	**7. CH2C/**	7. CHC/	**7.COS-**
6. CHW	**6. CH2C**	6. CHC	**6. COS+**
5. CHV/	**5. CH2B/**	5. CHB/	**5. SIN-**
4. CHV	**4. CH2B**	4. CHB	**4. SIN+**
3. CHU/	**3. CH2A/**	3. CHA/	**3. REF-**
2. CHU	**2. CH2A**	2. CHA	**2. REF+**
1. +5V	**1. +5V**	1. +5V	**1. N.C**

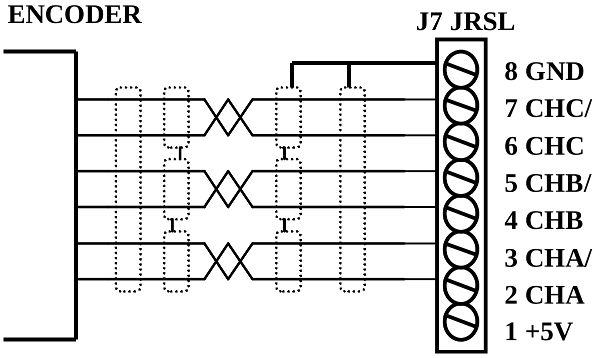
Encoder Feed Back J7 Ports:



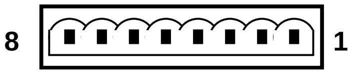
Resolver Feed Back Offer Diagram :



Encoder Feed Back Offer Diagram:



- **J8 JENC2 (10-PIN Terminal Block CONNECTOR)**



Front View

PIN#	SYMBOL	FUNCTION	Description	NOTES
8	GND	COMMON	Digit Common	
7	CH2C/	INPUT	Encoder C Ch Neg	#2 Encoder Input
6	CH2C	INPUT	Encoder C Ch Pos	#2 Encoder Input
5	CH2B/	INPUT	Encoder B Ch Neg	#2 Encoder Input
4	CH2B	INPUT	Encoder B Ch Pos	#2 Encoder Input
3	CH2A/	INPUT	Encoder A Ch Neg	#2 Encoder Input
2	CH2A	INPUT	Encoder A Ch Pos	#2 Encoder Input
1	+5V	POWER	+5V Power	

UTSD105 No.2 encoder feed back can adopt the standard output of line driver or open collector, and the interface as following:

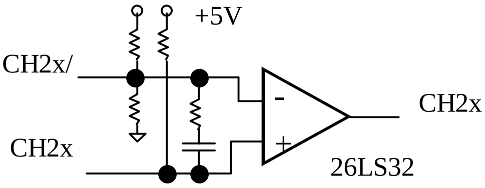
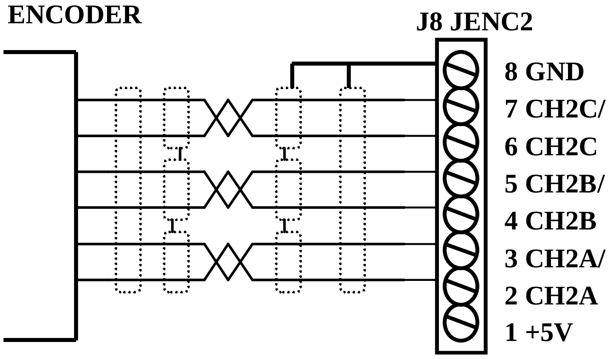
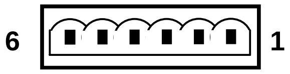


Diagram:



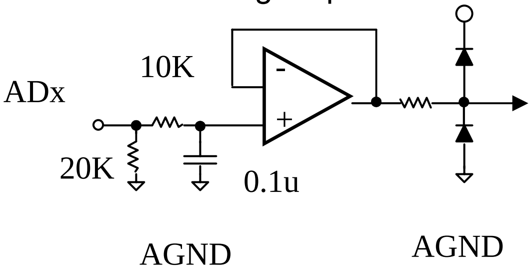
- **J9 JAIO (6-PIN Terminal Block CONNECTOR)**



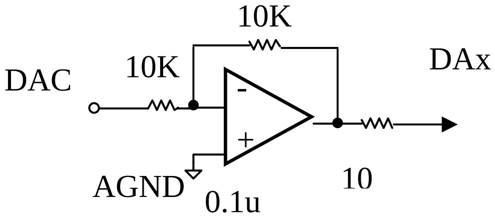
Front View

PIN#	SYMBOL	FUNCTION	Description	NOTES
6	+15VG	COMMON	+15V Common	
5	DA2	OUTPUT	Analog Output	DA #2
4	DA1	OUTPUT	Analog Output	DA #1
3	AD2	INPUT	Analog Input	AD #2
2	AD1	INPUT	Analog Input	AD #1
1	+15V	POWER	+15V Power	

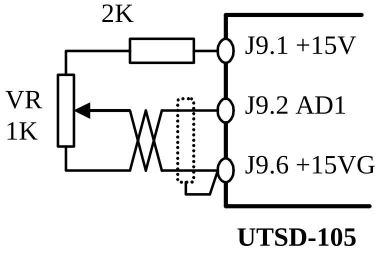
UTSD105 offers two analog output 0~5V. The resolution is 12 bits.



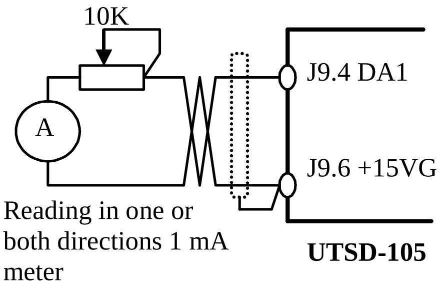
UTSD105 offers two analog output -10~10V. The resolution is 16 bits.



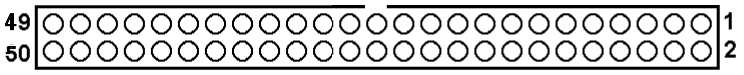
Example **Analog Input:**



Example Analog Output:



- **CN1 JACC (50-PIN HEADER)**

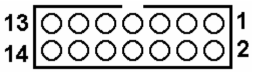


Front View

PIN#	SYMBOL	FUNCTION	Description	NOTES
1	D0	BIDIRECT	Data Bus Bit 0	
2	D1	BIDIRECT	Data Bus Bit 1	
3	D2	BIDIRECT	Data Bus Bit 2	
4	D3	BIDIRECT	Data Bus Bit 3	
5	D4	BIDIRECT	Data Bus Bit 4	
6	D5	BIDIRECT	Data Bus Bit 5	
7	D6	BIDIRECT	Data Bus Bit 6	
8	D7	BIDIRECT	Data Bus Bit 7	
9	D8	BIDIRECT	Data Bus Bit 8	
10	D9	BIDIRECT	Data Bus Bit 9	
11	D10	BIDIRECT	Data Bus Bit 10	
12	D11	BIDIRECT	Data Bus Bit 11	
13	D12	BIDIRECT	Data Bus Bit 2	
14	D13	BIDIRECT	Data Bus Bit 3	
15	D14	BIDIRECT	Data Bus Bit 4	
16	D15	BIDIRECT	Data Bus Bit 5	
17	D16	BIDIRECT	Data Bus Bit 6	
18	D17	BIDIRECT	Data Bus Bit 7	
19	D18	BIDIRECT	Data Bus Bit 8	
20	D19	BIDIRECT	Data Bus Bit 9	
21	D20	BIDIRECT	Data Bus Bit 20	
22	D21	BIDIRECT	Data Bus Bit 21	
23	D22	BIDIRECT	Data Bus Bit 22	
24	D23	BIDIRECT	Data Bus Bit 23	
25	GND	COMMON		
26	GND	COMMON		
27	A0	OUTPUT	Addr. Bus Bit 0	
28	A1	OUTPUT	Addr. Bus Bit 1	
29	A2	OUTPUT	Addr. Bus Bit 2	
30	A3	OUTPUT	Addr. Bus Bit 3	
31	/IOSEL1	OUTPUT	UTSD Chip Sel.	

32	/IOSEL2	OUTPUT	UTSD Chip Sel.	
33	/IOSEL3	OUTPUT	UTSD Chip Sel.	
34	/IOSEL4	OUTPUT	UTSD Chip Sel.	
35	/IOSEL5	OUTPUT	UTSD Chip Sel.	
36	/IOSEL6	OUTPUT	UTSD Chip Sel.	
37				
38				
39				
40				
41	/RESET.	OUTPUT		
42	+5V	POWER		
43	/IOSTRB_W	OUTPUT	Write Strobe	LOW TRUE
44	/IOSTRB_R	OUTPUT	Read Strobe	LOW TRUE
45	GND	COMMON		
46	GND	COMMON		
47	RESET	OUTPUT	Internal Reset	HIGH TRUE
48	+5V	POWER		
49	EXP_CLK			
50	+5V	POWER		

- **CN2 JDISP (14-PIN HEADER)**



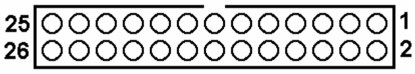
Front View

PIN#	SYMBOL	FUNCTION	DESCRIPTION	NOTES
1	VDD	OUTPUT	+5V power	Power output
2	Vss	COMMON	Common	
3	Rs	OUTPUT	Read strobe	TTL signal out
4	VEE	OUTPUT	Contrast adjust	**0 to +5VDC*(Remark)**
5	E	OUTPUT	Display enable	High to enable
6	R/W	OUTPUT	Read/write strobe	TTL signal out
7	DB0	OUTPUT	Display data 0	
8	DB1	OUTPUT	Display data 1	
9	DB2	OUTPUT	Display data 2	
10	DB3	OUTPUT	Display data 3	
11	DB4	OUTPUT	Display data 4	
12	DB5	OUTPUT	Display data 5	
13	DB6	OUTPUT	Display data 6	
14	DB7	OUTPUT	Display data 7	

JDISP can be connected to 2 x 40 LCD module to send the message on the display.

*Remark**:**Can be controlled by VR1

- **CN3 JTHW (26-PIN HEADER)**

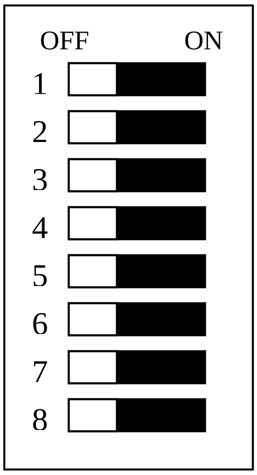


Front View

PIN#	SYMBOL	FUNCTION	DESCRIPTION	NOTE
1	GND	COMMON		
2	GND	COMMON		
3	MI1	INPUT		
4	MO1	OUTPUT		
5	MI2	INPUT		
6	MO2	OUTPUT		
7	MI3	INPUT		
8	MO3	OUTPUT		
9	MI4	INPUT		
10	MO4	OUTPUT		
11	MI5	INPUT		
12	MO5	OUTPUT		
13	MI6	INPUT		
14	MO6	OUTPUT		
15	MI7	INPUT		
16	MO7	OUTPUT		
17	MI8	INPUT		
18	MO8	OUTPUT		
19	N.C			
20	GND	COMMON		
21	N.C			
22	GND	COMMON		
23	N.C			
24	GND	COMMON		
25	+5V	+5VDC		
26	GND	COMMON		

JTHW offers 8 TTL lever inputs and 8 TTL lever inputs. The main function is to read BCD thumbwheel switches.

- **DIP_SWITCHES**



S

NO.	FUNCTION	OFF	ON
1	Power on re-initializing (The same as **$$$*** ** command)	NO	YES
2 3	Com1 baud rate	**00:**115200 **10:**19200	**01:** 9600 **11:** 38400
4	Com1 handshake	YES	NO
5	Current loop frequency	4K	6K
6	Wait state	1 wait state	0 wait state
7	Power on load from FLASH	NO	YES
8	Seven-Segment Display Mode	Standard	Other