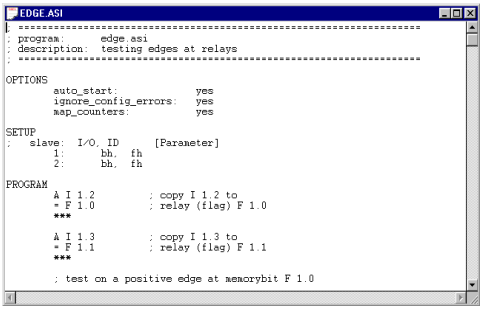
FACTORY AUTOMATION

MANUAL

PROGRAMMING LANGUAGE

AS-I CONTROL TOOLS



CE

PEPPERL+FUCHS

Table of Contents

Issue date 27.07.98

1 Introduction

AS-i Control II, the firmware integrated in Pepperl+Fuchs AS-i Masters , forms together with the AS-Interface a powerful mini-PLC.

Note

Pepperl+ Fuchs is still delivering AS-i masters without the mini-PLC functionality AS-i Control.
AS-i Master is used here as a generic term for AS-i gateways, AS-i PC boards and other AS-i masters.

AS-i Control II is the consistent further development of AS-i Control I which was presented the first time at the end of 1994. AS-i Control II can only be delivered in connection with a PEPPERL+FUCHS AS-i Master as hardware platform.

Integrated in an AS-i Master with serial interface AS-i Control II is the ideal mini-PLC for stand alone solutions for smaller machines or plants. Equipped with commercial AS-i I/O modules it can control to 248 inputs/outputs.

Using AS-i Control II in gateways, i.e. the AS-i/PROFIBUS DP Gateway, is capable to preprocess the actuator-sensor-data within the gateway. This way the hierarchically higher PLC is relieved. Thus AS-i Control II helps decentralizing the control task.

Typical applications are the preprocessing of analog data according to the AS-i analog protocol as well as the fast execution of time critical operations directly within the gateway.

Complete parts of plants or machines can be controlled independently by the gateway. This yields a great advantage when testing or commissioning a bigger plant.

Implemented in PC boards AS-i Control II relieves the PC from the time critical control tasks. With the PC boards AS-i PC2 and AS-i PC104 with AS-i Control II the control program is running on the AS-i board so that the PC processor is not stressed by the hard real time requirements of a control task. The full efficiency of the PC can be used for visualizing data, archiving data, etc.

The PLC program for AS-i Control II can be edited with a commercial PC and is downloaded to the AS-i master afterwards. Following programming tools are available: a DOS software for instruction list (IL with the STEP5™ syntax) and ladder diagram (LD), the easy-to-use Windows software AS-i Control Tools for commissioning and programming AS-i Control II in IL or LD and last but not least the IEC 1131 software AsiGraph.

The control program, i.e. the instructions for the logical combination of process data, is loaded via the serial interface, with PC boards via the ISA bus, and with a gateway via the corresponding fieldbus and stored in a non-volatile EEPROM so that it can restart after a power failure.

Note

As long as it does not process a control program, the AS-i Control master acts just like Pepperl+Fuchs regular AS-i Master without AS-i Control.

If the system recognizes AS-i errors or if the AS-i Master is in an unusual operating mode, the control program is stopped and the process put into a safe condition.

Issue date 27.07.98

2 AS-i Control I and AS-i Control II

This documentation describes both variants of AS-i Control: the original one, now to be referred to as AS-i Control I, and the new version, AS-i Control II.

AS-i Control II is compatible with AS-i Control I, i.e. the newer version includes all the characteristics of the former one in the same syntax.

Features characteristic of AS-i Control II are marked as such in the heading of the chapter concerned with the label "(AS-i Control II only)".

Tabelle 2.1: Tabular comparison of the old and the new versions of AS-i Control

	AS-i Control I	AS-i Control II
program memory	1,5 KB	16 KB
timers and counters	15 each	1024 each
parts of the control program can be written in 8051-assembler	no	yes
byte processing	no	yes
function blocks	no	yes
language	Interpreter	Compiler, hence a lot faster
access to execution control functions	no	yes

Issue date 27.07.98

3 Instruction List Conventions for Control Program Source Files

The AS-i Control program is stored as a normal text file that can be edited by any text editor which produces pure text. This file is read by the software and converted to the format used by the AS-i Control.

You can enter numbers in either hexadecimal or in decimal form. Add an '**h**' for hexadecimal and (optionally) a '**d**' for decimal.

Each line of the source text file is read up to the first semicolon; any text to the right of the semicolon will be taken as a comment. Empty lines, upper- and lower-case and almost all punctuation symbols are ignored. (The program reads only alphanumeric characters and the symbols '★', '=', '(',')' and '_'. Everything else is interpreted as a space. In the headings of sections in AS-i Control II, square brackets '[' and ']' exist as well.)

Besides the instructions for the logical association of AS-i data (*[PROGRAM]* and *[GATING]* sections), the source text file also contains information about the expected AS-i configuration (*[SETUP]* section) and about the state of the flags at the start of the control program (*[OPTIONS]* section). The file is arranged in five sections, *[OPTIONS]*, *[SETUP]*, *[PROGRAM]*, *[GATING]*, and *[END]*.

With AS-i Control II, the titles of these sections should be put into square brackets to avoid the risk of their being taken for labels. In AS-i Control II, the additional sections *[DECLARATIONS]*, *[ASSEMBLY]*, and *[RAW]* can occur.

The keywords (titles of sections) have to be positioned at the beginning of the section's first line.

[OPTIONS]

Indicates the start of the section of the source text that indicates the status of the "ignore_config_errors", "auto_start" and "map_counters" flags at the start of the control program.

[SETUP]

Indicates the start of the source text section containing the AS-i configuration.

When you download the program, the AS-i Control is configured with the settings in this section. The config_ok AS-i flag indicates that the current configuration matches the information in the *[SETUP]* section.

The source text may contain several *[SETUP]* sections.

[DECLARATIONS] **(AS-i Control II only)**

In this section, variables and function blocks are declared.

[PROGRAM]

Indicates the start of the section with the logic instructions. The syntax of these instructions is similar to STEP5®.
(STEP5 is a registered trademark of Siemens AG.)

The source text can contain several [PROGRAM] sections.

Issue date 27.07.98

[GATING]

Indicates the start of a section with logic instruction in a alternative syntax.

The source text can contain several [GATING] sections.

[PROGRAM] and *[GATING]* sections can be mixed in the source file. The AS-i Control processes the sections in the same order in which appear in the source text file.

[ASSEMBLY] **(AS-i Control II only)**

This section contains those parts of the control program that are written in Assembler.

[END]

Marks the end of the source text. Any text following the [END] statement is ignored.

A missing [END] statement will not cause an error. Concluding the source text with [END] is therefore optional.

[RAW] **(AS-i Control II only)**

After the uploading of a control program from an AS-i Master, this section holds the contents of the EEPROM as Intel hex records.

With **AS-i Control II** it is possible to distribute the source text for a project over several files, although the *[OPTIONS]* and *[SETUP]* sections still have to be located in the file for the main program.

Issue date 27.07.98

3.1 *[Options]* Section

The *[OPTIONS]* section serves to determine the starting behavior of AS-i Control.

"auto_start":

"yes": AS-i Control starts automatically, "no": AS-i Control has to be started manually.

"ignore_config_errors":

Configuration is checked when starting up; if "yes" was chosen, errors may occur during program execution, if slaves fail and no other adequate measures have been taken.

"map_counters":

Causes AS-i Control to map the counter values to flag memory if "yes" was chosen.

These statements have the following syntax:

<flag>: <value>

<value> can have the following syntax:

To set:	**yes**, **on**, or **true**
To not set:	**no**, **off**, or **false**

If no status is given for certain flags, the following defaults are used:

ignore_config_errors:	**no**
auto_start:	**yes**
map_counters:	**no**

The following *[OPTIONS]* section allows AS-i Control to continue with cyclical execution of the control program even if a configuration error has occurred. If the control program is stopped (due to low AS-i voltage, for instance) the master must wait for the "Set" button to be pushed before it restarts the program. No status of "map_counters" is given, so this option remains disabled.

```
[Options]
        ignore_config_errors: yes
        auto_start:           no
```

Issue date 27.07.98

3.2 *[Setup]* Section

Each *[SETUP]* section must begin with a line with the keyword "*[SETUP]*" in it. The *[SETUP]* section contains information about the slaves on the AS-i bus.

For each slave that is to be connected to the AS-i bus while the control program is running, there must be a line in the *[SETUP]* section with the operating address, the I/O code, the configuration ID, and (optionally) the parameters that are to be sent to the slave. The I/O code indicates the direction for the data (in- or output), while the configuration ID describes the kind of slave. A start-up parameter for the slave can be specified here, if necessary. The slave's data sheet says whether the setup is supported by parameters.

In addition to the decimal and hexadecimal notations, the I/O code can also be indicated by direct identification of the inputs and outputs ('`i`' = **i**nput, 'o' = **o**utput, 'b' = **b**i-directional).

Parameters are optional. The number fifteen (F_{hex}) is the default value.

Note

Compared with data in the configuration editor of the Windows software AS-i Control Tools and the DOS software ASISHELL the order of IO and ID-code is the other way round.
For example "FB" means IO = B_{hex} and ID = F_{hex}.

The following [SETUP] section assigns a "standard AS-i sensor" (three inputs, one output, ID=1) to address 16. This slave is activated by sending a fourteen (E_{hex}) as a parameter. Furthermore, a "standard actuator" is expected at address 3:

```
     [Setup]
;     slave:     IO,   ID, Power-up Parameter
         16:   iiio,    1, Eh
          3:     Bh,    0
```

Issue date 27.07.98

3.3 *[Declaration]* Section

A *[DECLARATION]* section can also be headed by *[DECL]*. It is here that the variables, timers and counters can be declared and the function blocks can be registered.

The syntax is as follows:

variable:

<symbol>* BYTE *<number of bytes>

sets up a variable with the name *<symbol>* in a "non-visible" part of the memory. The size in byte of the memory reserved has to be specified as parameter (decimal or hexadecimal).

timer:

T*<number>* TIMER

sets up a timer.

counter:

C*<number>* COUNTER

sets up a counter.

function block:

<symbol>* FB *<source text file>

declares a function block with the name <symbol>. The source text for this function block will be expected in a file with the name of which is specified as parameter. This file will only be processed if the function block is called.

The demo file "fb.asi" shows various examples for the use of function blocks.

The following [DECLARATION] section declares a variable with the size of 5 bytes, a timer, a counter, and a function block which is stored in the file mean.fb:

```
[DECL]
    Reading BYTE 5
    T134 TIMER
    C27 COUNTER
    MEAN VALUE FB MEAN.FB
```

Issue date 27.07.98

3.4 [Program] Section

The logic instructions for the AS-i data are defined in *[PROGRAM]* sections that are introduced by a single line with the keyword "*[PROGRAM]*".

The control program's syntax resembles that of STEP5®. However, the whole program must be written in a single program block (PB).

There can be several *[PROGRAM]* sections in a source text file, but a segment cannot span several sections.

3.4.1 Single-bit Processing

A **single-bit register** is used to process single-bit combinations. Conditional operations are executed only if the content of the single-bit register is "**1**".

All binary operations use this register as their first operand; and the result is stored there, as well. A binary operand serves as the second one. Thus, the single-bit register shows the results of all preceding combinations.

A normal instruction line for bit processing in the control program has the following structure:

<operator> <operand>

Operators

A "**A**nd":
logical AND between the single-bit register and the binary operand.

AN "**A**nd **N**ot":
logical AND between the single-bit register and the inversion of the binary operand.

O "**O**r":
logical OR between the single-bit register and the binary operand.

ON "**O**r **N**ot":
logical OR between the single-bit register and the conversion of the binary operand.

= Assignment:
the binary operand is assigned the single-bit register

N "**N**ot":
inversion of the single-bit register.

R "Conditional **R**eset":
the binary operand is allocated a "**0**" if the single-bit register has the content "**1**".

S "Conditional **S**et":
the binary operand is allocated a "**1**" if the single-bit register has the content "**0**".

Issue date 27.07.98

When a timer or a counter is accessed with **S**, the last value loaded with **L** is loaded into the count register of the addressed timer or counter, depending on the contents of the single-bit register. The timer bit remains set for the period defined; a set counter status bit indicates that the counter register does not equal zero.

Binary Operands

A binary operand is described by *<operand mark>*, a space, the *<operand address>*, a dot, and the *<bit address>* (e.g. **`I 21.3`**).

The following binary operands are available:

Input, **I2** (old input), **Q** (output), **F**lag, **T**imer and **C**ounter.
With the two latter, the status bit is available as a binary operand.

1.) **`I`** "**i**nput"image of the AS-i input data.

2.) **`I2`** "old **i**nput"image of the AS-i input data from the last program cycle.
By comparing corresponding bits in I and I2, input data edges can easily be recognized.

3.) **`O`** "**o**utput"image of the AS-i output data.

The bit addresses in the input/output data fields consist of two numeric values: the operating address of the AS-i slave to be addressed and the number of the data connection. Since there are 32 possible slave addresses with AS-i, with each slave possessing four data connections, "**`X 0.0`**" through "**`X 31.3`**" are valid binary operands in the input or output data image (**X** represents Input or Output).
There are 128 bytes of flag memory (8 bits each). "**`F 0.0`**" through "**`F 127.7`**" are valid binary operands in the flag memory.

4.) **`F`** "**f**lag":
flag memory in the control program.

Flags are sometimes called "relays" as their equivalent in electrical engineering.
If the *map_counters* flag was set at the start of the control program, the addresses **`F 96.0`** to **`F 125.7`** give access to the registers of the fifteen counters.

The following *[PROGRAM]* section causes AS-i Control to set output 3 on slave 17 exactly when input 0 is set on either slave 4 or on slave 5, and bit **`F 42.7`** is not set in the flag memory:

```
[Program]
     O  I    4.0
     O  I    5.0
     AN F   42.7
     =  Q   17.3
     ★★★
```

There are fifteen independent timers from "T 0" through "T 14" available.

5.) **`T`** "**t**imer":
status bits of the timer.

Issue date 27.07.98

If the time set on the timer is not yet over, the timer status bit remains set.
There are fifteen independent counters from "**C 0**" through "**C 14**" available.
If the map_counters flag was set, access to the counter registers is possible via the flag memory (see chap. 3.4.4).

6.) **C** "**c**ounter"
status bits of the counters.

If a counter register holds a value other than zero, the corresponding counter status bit is set.

Note

With AS-i Control II, the number of timers and counters is restricted only by the size of the RAM. With the new version, it is also possible to access a bit of a variable declared in [DECLARATIONS]:
<symbol> <byte address> . <bit address>

3.4.2 Multi-Bit Processing

Multi-bit processing is possible for timers, counters and parameters, with AS-i Control II also for byte- and word addresses. The syntax again is very similar to STEP5®.

Only one register can be pre-loaded as a multi-bit operand with each counter-, timer- and parameter constant. The actual loading and, with timers and counters, controlling as well as the reading is done by the associated status bits as binary operands.

The following loading operations are available:

L	**KT**	***<time in milliseconds>***	(load constant into timer)
L	**KC**	***<counter value>***	(load constant into counter)
L	**KP**	***address>.<data>***	(load constant into parameter)

With AS-i Control II, the following additional loading operations for byte- and word operands are available:

L	**KB**	***<8-bit-number>***	(load constant into byte)
L	**KW**	***<16-bit-number>***	(load constant into word)
L		***<byte address>***	(load contents of byte address)
L		***<word address>***	(load contents of word address)

plus copy operations for byte- and word operands:

T	***<byte address>***	(transfer to byte address)
T	***<word address>***	(transfer to word address)

Issue date 27.07.98

plus arithmetical operations and comparative operations for byte operands:

+	***F***	addition
-	***F***	subtraction
!=	***F***	equality (!)
><	***F***	inequality
>	***F***	greater than
>=	***F***	greater than or equal
<	***F***	smaller than
<=	***F***	smaller than or equal

The following [PROGRAM] section executes a subtraction:
flag byte 3 = flag byte 1 - flag byte 2.

```
[Program]
    L FY 1
    L FY 2
    - F
    T FY 3
    ★★★
```

Timers

The timer syntax again closely follows the STEP5® syntax:

L KT ***<time in milliseconds>*** (constant in timer)
•••
S T ***<timer number>***

The first instruction writes a time value into a 12-bit register independent of previous operations. The time is stored in units of 10 ms internally, but to simplify the usage, the time values in control programs are specified in single milliseconds and transferred to the internal values. Therefore the width of 12 bits limits the timer value to 40.95 sec. You can get around this limitation by combining the timer with a counter. (see demo *1h_timer.asi*).

If the program later accesses the timer with operator **S** (as a conditional operation), the last loaded timer value is loaded into this timer's count register, and the timer status bit remains set according to the time set.

Note

AS-i Control II writes the timer value into the upper 12 bits of a given 16-bit register. This means that the timer values that are written into a master during the downloading of a program with AS-i Control I get misrepresented when the program is uploaded with AS-i Control II, and vice versa.

Issue date 27.07.98

Counters

Loading a counter's count register works in a similar way:

```
L    KC   <counter value>          (constant in counter)
•••
S    C    <counter number>
```

A counter's status bit is set if the contents of the count register does not equal zero.

In addition to that, the counter contents can be increased or decreased with the following commands:

CU C ***<counter number>***

Conditional command to **c**ount **u**p (increase) the count register by 1.

CD C ***<counter number>***

Conditional command to **c**ount **d**own (decrease) the count register by 1.

STP C

STP C ***<counter number>***

This command alters the content of the counter register by one **st**e**p**. The counting direction depends on the content of the single-bit register:

single-bit register = 1:	count up (increment)
single-bit register = 0:	count down (decrement)

The width of 12 bits limits the counter value to 4095. You can get around this limitation by combining the counter with another counter.

If the *map_counters* flag was set at the start of the control program, the addresses **F 96.0** to **F 125.7** give access to the counter registers of the fifteen counters.

Issue date 27.07.98

The following *[PROGRAM]* section causes output 0 of slave 31 to blink at a frequency of 1 Hz:

```
[Program]
   A    T  0          ;timer 0 still running
   JC   = end         ;skip following lines
   N
   L    KT 500        ;timer 0 no longer running:
   S    T  0          ;restart timer 0 with 500 ms
   ***

   AN   Q  31.0       ;invert output Q 31.0
   =    Q  31.0
end
   BE
```

3.4.3 Byte- and Word Addresses (AS-i Control II only)

The syntax for the byte addresses is as follows:

QB	***<even number>***	output
IB	***<even number>***	input
I2B	***<even number>***	old input
FY	***<number>***	flag
<symbol>	***<number>***	variable declared in *[DECLARATIONS]*

Only even numbers can be used for the in- and outputs. As one in- or output spans four bits, only complete bits are addressed.

The syntax for the word addresses is as follows:

FW	***<number>***	flag word
T	***<number>***	timer (also AS-i Control I)
C	***<number>***	counter (also AS-i Control I)

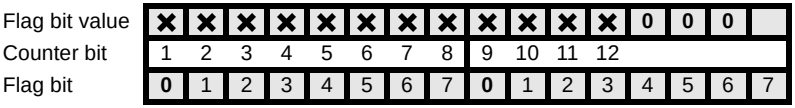
Issue date 27.07.98

3.4.4 Access to Counter Values

Counter values can be accessed by selecting the appropriate flag byte and -bit. The first counter with the number 0 is mapped to flag byte 96. Each counter is twelve bits wide and therefore takes up two bytes.

How to calculate the address of a counter:

Multiply the counter number by two and add 96. Example: Counter 7: 7 * 2 + 96 = 110. Thus you can access counter 7 by selecting the bits of flag byte 110 and 111. The syntax for accessing a single bit is **F**<byte>.<bit> (e.g. for the 6th bit of the 3rd counter: 2 * 2 + 96 => F100.5). The four most significant bits of the second byte are always zero.



Counter	Flag bytes	
0	96	97
1	98	99
2	100	101
3	102	103
4	104	105
5	106	107
6	108	109
7	110	111
8	112	113
9	114	115
10	116	117
11	118	119
12	120	121
13	122	123
14	124	125

Issue date 27.07.98

3.4.5 Execution Control

There are seven instructions available to control the program execution:

******* segment end

concludes a block of logical operations. The result of previous logical operations is taken to be "undefined". The next **A** or **O** operations correspond to loading the single-bit register while the next **AN** or **ON** operations correspond to the loading of its inversion.

BE Block **E**nd

Since AS-i Control can have only one block, BE marks the end of the control program. Control program data is exchanged with the AS-i and a new control program cycle is started.

BEC **B**lock **E**nd **C**onditional

If the contents of the single-bit register equals 1, program execution is aborted at this point and a new control program cycle is started. Otherwise the program is continued.

BEU **B**lock **E**nd **U**nconditional **(AS-i Control II only)**

program execution is aborted at this point and a new program cycle is started, no matter what the content of the single-bit register is.

If the commands BEC and BEU are used in a function block, then program execution is aborted there and the program turns back to the routine that had called the function block.

JU **J**ump (**U**nconditional) to the indicated label

If a program line is marked with *<label>*: (*<instructions>*...), program execution can jump to this label from any position in the control program with the command

JU = *<label>*

<label> can be any string consisting of alphanumeric characters, and the symbols "*****", "**=**", "**(**", "**)**" and "_". A label must be unique and different from all other labels in the source text.

JC **J**ump (**C**onditional) to the indicated label

Similar to JU, but the jump depends on contents of the single-bit register, i.e. the jump is only executed if the single-bit register contains "1".

Pay attention to the program flow not to create endless loops. In this case, the program is aborted with a program cycle time-out.

FB subroutine call **(AS-i Control II only)**

Syntax: `JU FB` *<symbol>* or `JC FB` *<symbol>*

Call of the function block with the name <symbol>. The function block must have been declared in a *`[DECLARATION]`* section.

Issue date 27.07.98

3.4.6 AS-i Parameter

Sending parameters to an AS-i slave is available as an AS-i specific instruction:

PARAM Send parameters to an AS-i slave

(With **AS-i Control II**, the command PARAM calls the firmware funktion block SendPrm.)

This is a conditional instruction, carried out only if the content of the single-bit register is "1".

The address of the slave and the value to be sent are read out of the 12-bit register. This has to be loaded in advance, accordingly:

```
                L   KP  <address>.<data>      (constant in parameters)
Example:        L   KP  13.9
```

The slave's response is stored in the lower 4 bits of flag memory 127. Flag **F** 127.7 is set if the execution control does not indicate an error while sending the parameters.

The sending of the AS-i parameters is carried out during the AS-i master's management phase. This is reached every five milliseconds, even if the maximum number of slaves is connected. The sending of the **PARAM** command has to wait for the management phase. The number of slaves is significant both for the wait and for the cycle time of the program: the less slaves there are, the shorter the wait, and the shorter the program's cycle time.

Every rising edge at input 12.0 causes a "3" to be sent as parameter to slave 13. A "12" is sent on every falling edge.

```
[Program]
   A    I  12.0          ;rising edge
   AN   I2 12.0
   L    KP 13.3          ;parameter value "3"
   PARAM
   ***

   A    I2 12.0          ;falling edge
   AN   I  12.0
   L    KP 13.12         ;parameter value "12"
   PARAM
   ***
```

3.4.7 ASM Operator (AS-i Control II only)

With the ASM-operator, assembler commands can be inserted into the *[PROGRAM]* section:

ASM the rest of the line is passed on from the compiler to the assembler without further examination, as if it was part of the assembly section (see below).

Issue date 27.07.98

3.5 *[Gating]* Sections

The *[GATING]* sections can contain instructions for the bit-by-bit logical association of AS-i data in an alternative, simpler syntax.

An instruction in such a section must have the following structure:

<target> = **[not]** ***<source>*** [***<operator>*** **[not]** ***<source>*** **[...]]**

<target> and *<source>* are indicated just like operands in the ***Program*** **sections** as <***operand mark> <operand address>.<bit address>***.

<target>: Address where the result of the logical operation to the right of the equals sign is to be stored.

<source>: Indication of the AS-i data addresses that are to be logically associated.

not: (Optional) - The information indicated by *<source>* is inverted prior to the logic operation when it is preceded by "**not**".

"**not**" can be abbreviated as "**n**".

<operator> Indicates the type of logical operation.

"**and**" and "**or**" are possible logical operaton. "**and**" and "**or**" are possible logical operators.

The following Gating section causes AS-i Control to set output 3 on slave 17 exactly when input 0 is set either on slave 4 or slave 5, and bit **F 42.7** is not set in the flag memory. In addition to that, input 5.0 is mapped directly to output **17.2**.

```
[Gating]
   Q 17.3 = I 4.0 or I 5.0 and not F 42.7
   Q 17.2 = I 5.0
```

Issue date 27.07.98

3.6 *[Assembly]* Section (AS-i Control II only)

As the compiler first produces assembler code which, in a further step, is translated into assembler, it would seem reasonable to be able to enter parts of the control program directly in assembler.

This is what the *[ASSEMBLY]* section is used for. Its contents are transferred from the compiler to the assembler without further examination. It is part of the programmer's job to decide which operations will not interfere with the running of the AS-i master and with the host interface. (It would make no sense, for example, to block all interrupts.)

An *[ASSEMBLY]* section can also be headed with *[ASM]*.

3.6.1 Pre- and Suffixes for Symbols in the Assembler Text

To avoid the possibility of mixing up names, the symbols get different pre- and suffixes in the source texts, depending on the category in the assembler text.

Symbols in the Code Area:

<*symbol*>:	function block of the firmware
U?<*symbol*>:	reloaded function block
?<*X*> <*hex-number*>:	automatically produced jump addresses
?U<*symbol*>:	jump adresses for JU and JC

Symbols in the Data Area:

<*symbol*>:	data fields from *[DECLARATION]* sections
<*symbol*>:	data fields of the firmware
?<*symbol*>:	automatically produced data fields, e.g.:
?Cci<*number*>:	counter
?Tci<*number*>:	timer
?PSeg:	parameter passing segment

3.7 *[Raw]* Section (AS-i Control II only)

The *[RAW]* section is produced during the upload and gives the content of the EEPROM as Intel-hex-records. There is no use for this section in a user program.

Issue date 27.07.98

3.8 Firmware Function Blocks (AS-i Control II only)

The AS-i master's host-interface functions can also be called as 'function block of the firmware' by the control program.

Among other things, this makes it possible to send new parameters to an AS-i slave or to control the list of active slaves from the control program.

Prior to every translation, the compiler reads the entry point of these functions out of the AS-i master.

3.8.1 Parameter Passing

The memory segment PSeg is used for passing the function parameters to the function block and read their return values. The parameters have to be copied into PSeg before calling the function block. After the execution of the function block the return values can be read from PSeg.

3.8.2 Description of the AS-i Master Function Blocks

The following table shows how the firmware function blocks are called up.

symbol	description	calling parameters	return value
SetPP	write power-up para-meter (PP)	PSeg[0]: slave address PSeg[1]: power-up para-meter of this AS-i slave	PSeg[0]: 00_{hex} (success) 80_{hex} (error)
GetPP	read power-up para-meter (PP)	PSeg[0]: slave address	PSeg[0]: 00_{hex} (success) 80_{hex} (error) PSeg[1]: power-up parame-ter of this AS-i slave
SendPrm	send AS-i slave para-meter (PI) (equivalent to PARAM)	PSeg[0]: slave address PSeg[1]: parameter of this AS-i slave	PSeg[0]: 00_{hex} (success) 80_{hex} (error) PSeg[1]: AS-i Slave response to para-meter call
GetPI	read image of AS-i Slave para-meter (PI)	PSeg[0]: slave address	PSeg[0]: 00_{hex} (success) 80_{hex} (error) PSeg[1]: parameter of this AS-i Slave
GetPCD	read slave profile from configu-ration data (PCD)	PSeg[0]: slave address	PSeg[0]: 00_{hex} (success) 80_{hex} (error) PSeg[1]: expected slave pro-file at the slave address
GetCDI	read image of AS-i slave profiles (CDI)	PSeg[0]: slave address	PSeg[0]: 00_{hex} (success) 80_{hex} (error) PSeg[1]: profile of this AS-i slave

Issue date 27.07.98

symbol	description	calling parameters	return value
GetLPS	read list of AS-i slaves to be expected (LPS)	none	PSeg[0]: 00_{hex} (success) 80_{hex} (error) PSeg[1]: list of slaves (slaves 24 to 31) PSeg[2]: list of slaves (slaves 16 to 23) PSeg[3]: list of slaves (saves 8 to 15) PSeg[4]: list of slaves (slaves 0 to 7)
GetLAS	read list of AS-i slaves in data exchange (LAS)	none	PSeg[0]: 00_{hex} (success) 80_{hex} (error) PSeg[1]: list of slaves (slaves 24 bis 31) PSeg[2]: list of slaves (slaves 16 bis 23) PSeg[3]: list of slaves (slaves 8 bis 15) PSeg[4]: list of slaves (slaves 0 bis 7)
GetLDS	read list of detected AS-i slaves (LDS)	none	PSeg[0]: 00_{hex} (success) 80_{hex} (error) PSeg[1]: list of slaves (slaves 24 bis 31) PSeg[2]: list of slaves (slaves 16 bis 23) PSeg[3]: list of slaves (slaves 8 bis 15) PSeg[4]: list of slaves (slaves 0 bis 7)
DataEx	turn off data exchange	PSeg[0]: 0 (activate data exchange) ≠0 (deactivate data exchange)	PSeg[0] 00_{hex} (success) 80_{hex} (error)
Address	change AS-i slave address	PSeg[0]: (old) slave address PSeg[1]: destination address	PSeg[0]: 00_{hex} (success), 82_{hex} (slave not found), 83_{hex} (slave with adresse 0 detec-ted), 84_{hex} (destination address already exists), 85_{hex} (slave address could not get deleted), 86_{hex} (destination address could not get program-med) 87_{hex} (destination address has been stored non-volatile)

Issue date 27.07.98

symbol	description	calling parameters	return value
AutoAddr	enable automatic programming of AS-i slave adresses	PSeg[0]: 0 (disable automatic programming) ≠0 (enable automatic programming)	PSeg[0]: 00_{hex} (success) 80_{hex} (error)
Command	call AS-i command	PSeg[0]: slave address PSeg[1]: information part in the command call (5 bit)	PSeg[0]: 00_{hex} (success) 80_{hex} (error) PSeg [1]: information part of the slave response (4 bit)
RdEeprom	read EEPROM	PSeg[0]: number of bytes to read (1 to 8) PSeg[1]: EEPROM start address (high byte) PSeg[2]: EEPROM start address (low byte)	PSeg[0..7]:up to 8 byte data
WrEeprom	write EEPROM	PSeg[0]: number of bytes to write(1 to 8) PSeg[1]: EEPROM start address (high byte) PSeg[2]: EEPROM start address (low byte) PSeg[3..10]:up to 8 byte data	PSeg[0..7]:up to 8 byte data
GetId	read firmware version-ID	none	PSeg[0..7]: 8 byte version-ID

The following [PROGRAM] section writes the power-up parameter 0A(hex) to slave 17.

```
[Program]
   L KB 17          ;load slave address
   T PSeg 0         ;write parameter passing segment

   L KB 0Ah         ;load power-up parameter
   T PSeg 1         ;write parameter passing segment

   JU FB SetPP      ;call firmware function block SetPP
```

Issue date 27.07.98

4 Control Program Demos

The following list shows you the control program demos which are available in the directory "demo" of the software.

Use them to train yourself and feel free to edit them to suit your demands.

program file	description
counter.asi	A counter is set and counted up or down by rising edges at inputs.
edge.asi	Testing edges at flags.
fb.asi	Use of function blocks. **(AS-i Control II only)**
first.asi	Input/output with 2I-/2O-moduls.
flipflop.asi	An edge triggered and a state triggered flip-flop.
gating.asi	Alternative syntax for logical functions.
jump.asi	Conditional and unconditional jumps.
logic.asi	Simple logical functions.
offdelay.asi	Switch-off delay.
ondelay.asi	Switch-on delay.
oscil.asi	Pulse generator by using conditional jumps.
param.asi	Send parameter bits to a slave.
pulse.asi	High pulse (duration of one second) at an output.
timer.asi	Use a timer and test if it ran out.
1h_timer.asi	Timer for time ranges longer than 40 seconds.

Issue date 27.07.98

Ausgabedatum 27.07.98

Pepperl+Fuchs GmbH · 68301 Mannheim · Telefon (06 21) 7 76-11 11 · Telefax (06 21) 7 76-10 00 · Internet http://www.pepperl-fuchs.com

Issue date 27.07.98

One Company, Two Divisions.

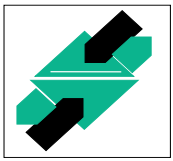
Factory Automation Division

Product Range

- Binary and analog sensors
- in different technologies
  - Inductive and capacitive sensors
  - Magnetic sensors
  - Ultrasonic sensors
  - Photoelectric sensors
- Incremental and absolute rotary encoders
- Counters and control equipment
- ID systems
- AS-Interface

Areas of Application

- Machine engineering
- Conveyor or transport
- Packaging and bottling
- Automobile industry



Process Automation Division

Product Range

- Signal conditioners
- Intrinsically safe interface modules
- Remote process interface
- Intrinsically safe field bus solutions
- Level control sensors
- Process measuring and control systems engineering at the interface level
- Intrinsic safety training

Areas of Application

- Chemical industry
- Industrial and community sewage
- Oil, gas and petrochemical industry
- PLC and process control systems
- Engineering companies for process systems

Service Area

Worldwide sales, customer service and consultation via competent and reliable Pepperl+Fuchs associates ensure that you can contact us wherever or whenever you need us. We have subsidiaries worldwide for your convenience.

The Pepperl+Fuchs Group

USA Headquarter
Pepperl+Fuchs Inc.• 1600 Enterprise Parkway
Twinsburg, Ohio 44087 • Cleveland-USA
Tel. (330) 4 25 35 55 • Fax (330) 4 25 93 85
e-mail: sales@us.pepperl-fuchs.com

Asia Pacific Headquarter
Pepperl+Fuchs Pte Ltd. • P+F Building
18 Ayer Rajah Crescent • Singapore 139942
Tel. (65) 7 79 90 91 • Fax (65) 8 73 16 37
e-mail: sales@sg.pepperl-fuchs.com

Worldwide Headquarter
Pepperl+Fuchs GmbH • Königsberger Allee 87 68307
Mannheim • Germany
Tel. +49 621 7 76-0 • Fax +49 621 7 76-10 00
http://www.pepperl-fuchs.com
e-mail: info@de.pepperl-fuchs.com

PEPPERL+FUCHS