1st International conference
on
P-NET fieldbus system
31st October - 1st November 1991
Scandic Hotel, Silkeborg, Denmark

Application example:

Gateway to other systems

By John Johansen, Application engineer, Proces-Data Silkeborg ApS, Denmark.

PD Gateway description

PD Gateway is an advanced option in the high level programming language, PROCESS-PASCAL, which enables you to integrate commmunication protocols for control and regulation equipment (PLC’s etc.) from different manufacturers, directly into the PROCESS-PASCAL programme.

By means of the PD Gateway option, a "non P-NET compatible device" can be regarded as a P-NET interface unit and can be accessed as a "normal P-NET device" including error detection and error handling, from the entire P-NET system. On the other hand, the P-NET system can be regarded as a part of the "non P-NET system", just another unit, when you access it from the "non P-NET" side.

The "non P-NET system" is connected to the P-NET via a PD 3000 P-NET Controller, using either a RS-232 or RS-485 communication port. The PD 3000 P-NET Controller is always connected to the "non P-NET system" at Port1.

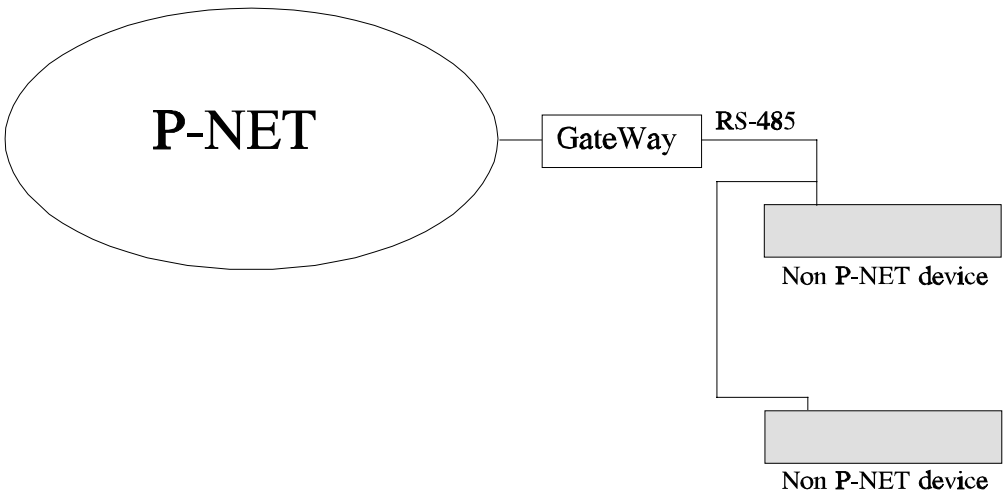


Figure 1: General system layout.

ACCESSING VARIABLES IN THE "NON P-NET SYSTEM" FROM P-NET:
The principle is, that instead of accessing the device directly, following the P-NET standard, a PROCESS-PASCAL programme is activated from the operating system. This programme uses Port1 to communicate with the actual device, using the appropriate protocol. The communication protocol is written in PROCESS-PASCAL.

When the answer from the device is ready, the PROCESS-PASCAL programme returns the answer to the operating system in exactly the same manner as if the answer was recieved in a normal P-NET transmission.

The variables which are to be accessed in the "non P-NET device" are declared in the master units where the access is required by the application, exactly as a normal variable declaration. The variables are declared as global variables with a NET address, including a net list and, depending on the communication protocol, an address specification. The net list for these variables must involve Port1 in the Gateway Controller to activate the PROCESS-PASCAL communication programme.

ACCESSING VARIABLES ON THE P-NET FROM THE "NON P-NET DEVICE":
Depending on the application and the communication facilities in the "non P-NET device", modules and data on the entire P-NET can be accessed from the device.

At softwire number $30 a RECORD is declared, used to transfer data from the operating system to the PROCESS-PASCAL programme which interfaces to the "NON P-NET DEVICE" (CALL). After the PROCESS-PASCAL programme has performed the "NON P-NET" transmission, the same RECORD is used to transfer data from the PROCESS-PASCAL programme to the operating system (ANSWER).

The PD Gateway facility is invoked by inserting 01 in the byte variable ModePort1.IntercomMode, and the interrupt number of the SoftwireInterruptTask in the byte variable ModePort1.IntercomInt. Accessing variables via Port 1 will activate the SoftwireInterruptTask.

NOTE: It is NOT allowed to select IntercomMode and P-NET protocol in ModePort1 at the same time.

The record at softwire $30 is declared in the following way:

```
{SW $30} InterComRec = RECORD
                         NodeAddress : STRING[25];
                         Instruction : BYTE;
                         DataLength  : BYTE;
                         Addr        : LONGINTEGER;
                         Offset      : INTEGER;
                         BitNo       : BYTE;
                         Data        : ARRAY[1..56] of BYTE;
                         ErrorCode   : WORD;
                         SWNo        : INTEGER;
                         Flags       : ARRAY[0..7] of BOOLEAN;
                       END;
```

NodeAddress

This field is used only to transfer data from the operating system to the PROCESS-PASCAL programme (CALL). The length of the string indicates, how many numbers the string contains. If, for example, a variable is declared - AT NET:(01, 04, 02, 33) the length of the string will be 3, and the contents will be 04, 02, 33 (The first 01 indicates Port1, and is not transferred in the CALL).

Instruction

This field is used to transfer data in CALL as well as ANSWER. First, it is used to transfer the instruction (as it is defined in the P-NET standard) in the CALL. In the ANSWER it is used to transfer instruction/status to the operating system - again according to the P-NET standard, thus enabling the PROCESS-PASCAL programme to return "Actual Data Error" etc. to the operating system.

DataLength

This field is also used both ways. It is used to tell, how many databytes the transmission is concerning. For example, if the transmission is a "LOAD 8 BYTE", datalength will be 8 in CALL as well as ANSWER. If the transmission is a "STORE 4 BYTE", datalength will be 4 in CALL, and 0 in ANSWER. So, the datalength is used to tell the PROCESS-PASCAL programme how many databytes are to be loaded/stored etc., and it is used to tell the operating system the number of databytes in the answer. The datalength must not exceed 56.

Addr

Addr is used only in CALL. Addr contains the address, as it is known from the P-NET standard. According to the P-NET standard, the address can be 2 or 4 bytes long. However, the address in Addr will allways be sign extended to 4 bytes. If the address in the P-NET block was 4 bytes, FLAGS[0] will be TRUE.

Offset

Offset is used only in CALL, according to the P-NET standard. If there was no offset in the P-NET block, the value 0 will be transferred in Offset. If there was an offset in the P-NET block, FLAGS[1] will be TRUE.

BitNo

BitNo is used only in CALL. If the transmission is a bit-transmission (indicated in the P-NET block by MSB in a 4 byte address SET), BitNo will contain the bit number, 0..7, and FLAGS[2] will be TRUE.

DATA

The Data field is used in CALL as well as ANSWER, and contains data as defined in the P-NET standard.

ErrorCode

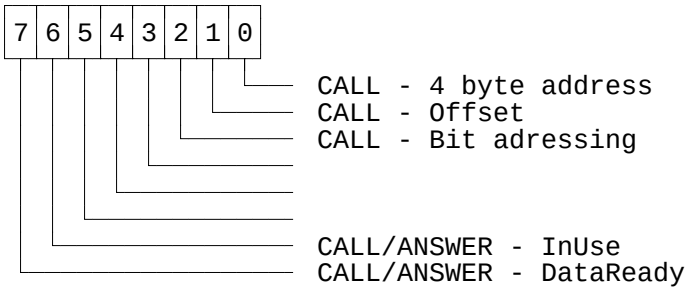
This field is used in ANSWER only. If the "NON P-NET" transmission results in an error of some kind, the PROCESS-PASCAL programme has the possibility to return an errorcode in this field. The operating system inserts 0 in this field in the CALL. If the ErrorCode is not 0 in the ANSWER, the operating system assumes there was a communication error, and does NOT use any data. Instead, the information "P-NET Error" is returned to the master.

SWNo

This field is not used.

FLAGS

This field is used in CALL as well as ANSWER.

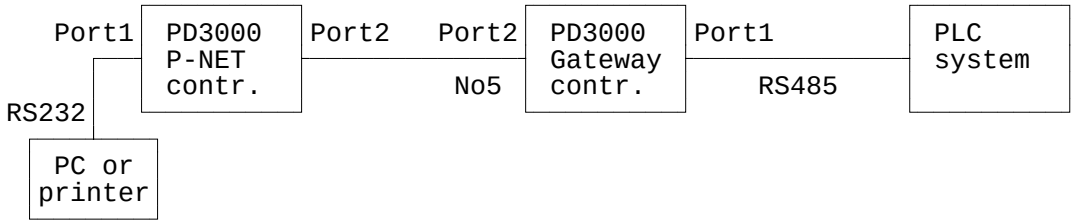


The meaning of the flags 0, 1, 2 is explained under Addr, Offset and BitNo respectively.

FLAGS[6] (InUse) is set TRUE by the operating system before the PROCESS-PASCAL programme is activated. The flag must NEVER be cleared from the PROCESS-PASCAL programme.

FLAGS[7] (DataReady) is set FALSE by the operating system before the PROCESS-PASCAL programme is activated. The flag must be set TRUE by the PROCESS-PASCAL programme, when the "NON P-NET" transmission is completed, i.e. data, errorcode and so on has been transferred to IntercomRec. If FLAGS[7] is not set TRUE within 2 seconds after the PROCESS-PASCAL programme is activated, the operating system assumes that a failure occured, returns "NO ANSWER" to the master, and inserts FALSE in InUse and DataReady.

Example:



The PLC system consists of 2 PLC's, with node numbers 1 and 2.

In the Gateway controller, ModePort1, IntercomMode is set to 01, IntercomInt is set to 5, Protocol is set to 3 (Datamode in+out), and the correct BAUD rate etc. is selected.

In the Gateway controller, the PLC-data are declared this way:

```
PLC1Data: ARRAY[1..100] of INTEGER[DeviceType:3000]
          AT NET:(1,1) SOFTWIRE:$1234;
PLC2Data: ARRAY[1..100] of INTEGER[DeviceType:3000]
          AT NET:(1,2) SOFTWIRE:$1234;
```

In the P-NET controller, the PLC-data can be declared in 2 different ways. This way:

```
PLC1Data: ARRAY[1..100] of INTEGER[DeviceType:3000]
          AT NET:(2,5,1,1) SOFTWIRE:$1234;
PLC2Data: ARRAY[1..100] of INTEGER[DeviceType:3000]
          AT NET:(2,5,1,2) SOFTWIRE:$1234;
```

Or this way:

```
PLC1Data: ARRAY[1..100] of INTEGER[DeviceType:3000]
          AT NET:(2,5) SOFTWIRE:aaaa;
PLC2Data: ARRAY[1..100] of INTEGER[DeviceType:3000]
          AT NET:(2,5) SOFTWIRE:bbbb;
```

where aaaa and bbbb are the softwire numbers for PLC1Data and PLC2Data declared in the Gateway controller (from the Gateway controller MAP-file).

When a LOAD transmission of PLC1Data with index $31 is initiated from the P-NET controller, the Gateway controller returns "answer comes later" to the P-NET controller. Then the following data are transferred to softwire $30 in the Gateway controller:

```
NodeAddress = $01
Instruction = $02              (LOAD)
DataLength  = $02          (INTEGER size)
Addr        = $00001234
Offset      = $0062             (index $31 * 2)
BitNo       NOT DEFINED
Data        NOT DEFINED
ErrorCode   = $0000
SWNo        NOT DEFINED
FLAGS       = 0/1/0/0/0/0/1/0
```

After inserting these data, the task with interrupt number 5 is activated. This task must perform a transmission to the PLC, and insert the result in softwire $30:

```
NodeAddress  Don't care
Instruction = $82              (module error in the PLC)
DataLength  = $02
Address     Don't care
Offset      Don't care
BitNo       Don't care
Data        = 2233             (the data from PLC integer $31)
ErrorCode   = $0000            (no transmission errors occured)
SWNo        Don't care
FLAGS       = 0/1/0/0/0/0/1/0
```

After inserting these data, the PROCESS-PASCAL programme in the Gateway controller must insert TRUE in FLAGS[7]. This makes the operating system return the answer to the P-NET controller, with the data from the PLC.

Thus, the variables PLC1Data and PLC2Data can be accessed from the P-NET controller or from the Gateway controller, just as if they were variables inside a normal P-NET device.

GateWay to PASILAC PLC

Below is described an example on using the Gateway Controller to communicate to a PASILAC PLC system. The PASILAC PLC system is connected to the Gateway Controller at Port1 using the RS-232 communication port.

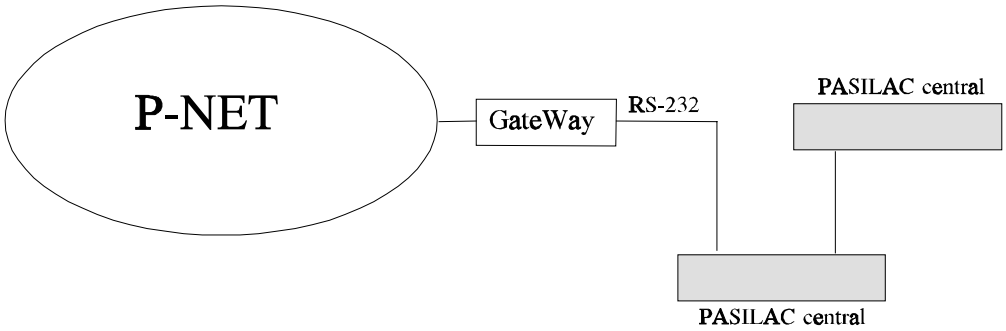


Figure 2: PASILAC system layout.

COMMUNICATION PROTOCOL.

The communication port is set up for full duplex transmission. All data transfers consist of a call and a reply. The call is made by a master and the reply is send by a slave. Both the Gateway controller and the PASILAC PLC system can act as a master and a slave. The communication protocol is based on a command instruction followed by an address and data. The data formats include the types BOOLEAN and INTEGER. A BOOLEAN is represented as an ASCII character "0" for FALSE, or "1" for TRUE. An INTEGER value is represented in 4 ASCII characters, one for each digit.

Communication parameters.

Before the Gateway/PASILAC controller is ready to operate, the P-NET numbers and the right number of masters must be set up to correspond to the entire network configuration. This can be done by editing the file 'GATEWAY.SYS' and compile the GATEWAY program, or by

means of the MONITOR program by changing the variable: ’ModePort2’, and inserting "3" in the variable: ’StartCode’.

Gateway/PASILAC controller has the following default values for setting up Port1 and Port2:

Port1:	RS-232, 9600 baud, 7 databit, 1 startbit and 2 stopbit
Port2:	P-NET, 6 masters, P-NET number 6.

Preset values for the communication programme:

TransPreset	The total time interval allowed for a transmission. The default value is set to 1.4 sec. and this value may not be exceeded.
InputPreset	The total time interval allowed before a character is received from the PASILAC-system. The default value is set to 0.4 sec.
MasterDelayPreset	The time interval in which the Gateway controller must wait from receiving ’CLEAR’ until it may call up again. The default value is set to 0.0 sec.
RetryMax	The maximum number of times PASCOM may try to make a call without generating an error message. The default value is set to 3 times.

Variables in the PASILAC PLC system.

Variables located in the PASILAC PLC system, which you want to load from P-NET, must be declared in the controllers as normal variable declarations. The data, which can be Integer or Boolean, are declared in the following way:

A BOOLEAN at the PASILAC PLC address 45.6:

```
PasilacFlag: BOOLEAN [DeviceType:3000] AT NET: (2, 6, 1, 2) ADDRESS: $456;
```

An ARRAY of PASILAC PLC BOOLEANS:

```
PasilacBitArray: ARRAY [0..$0FFF] OF BOOLEAN [DeviceType:3000] AT NET: (2, 6, 1, 2) ADDRESS: 0;
```

An Integer at the PASILAC PLC address $A010:

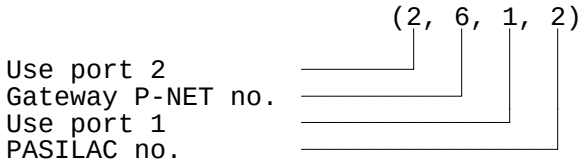
```
PasilacInteger: INTEGER [DeviceType:3000] AT NET: (2, 6, 1, 2) ADDRESS: $0A010;
```

An Array of PASILAC PLC INTEGERS from address $A000 to FFFE:

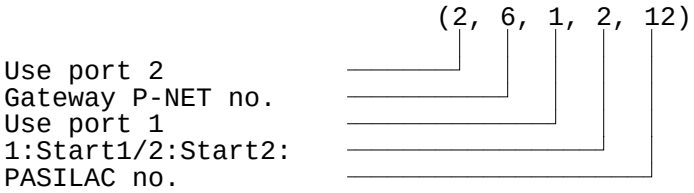
```
PasilacIntArray: ARRAY [$A000..$FFFE] OF INTEGER [DeviceType:3000] AT NET: (2, 6, 1, 2) ADDRESS: 0;
```

The Node address has the following meaning:

The first example is a system with only one PASILAC PLC, i.e. the call will transmit the code Start0:



The second example is a system with a central on the PASILAC PLC system, i.e. the call will transmit the code Start1 or Start2:



Variables at the P-NET.

Variables located at the P-NET, which must be loaded from the PASILAC PLC system must be declared in the Gateway/PASILAC controller before the program is finally compiled.

The variable declaration is done in a normal way in PROCESS-PASCAL, but only the types INTEGER and BOOLEAN are allowed to use. This is because of a restriction in the PASILAC PLC system.

Data can now be loaded from the P-NET into the PASILAC PLC system by using the softwire number as the address for the variables concerned, either an integer or a boolean.

NOTE: it is not possible to declare an array of booleans or integers in the controllers when data are to be loaded from the P-NET and transmitted to the PASILAC PLC system.

Software documentation

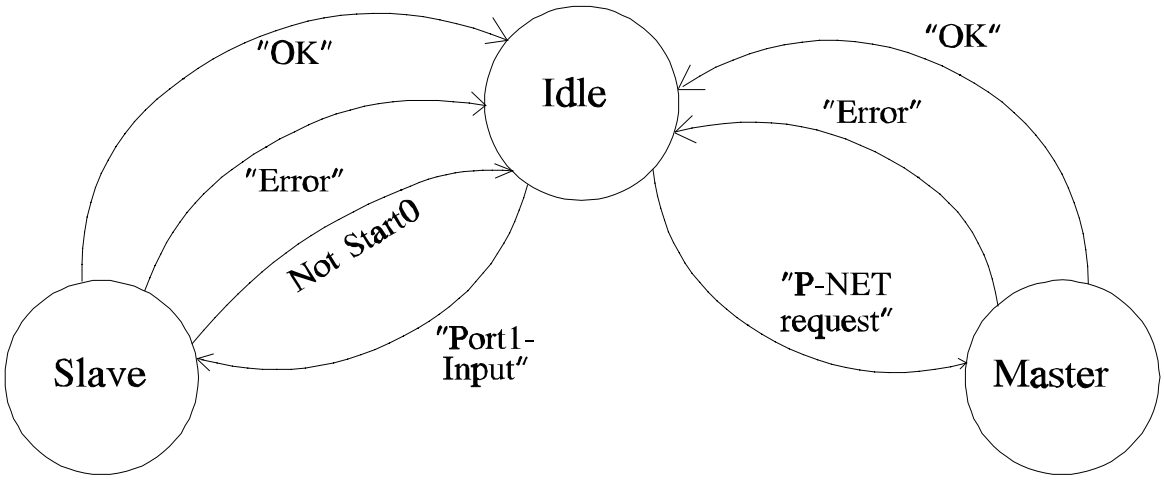
The PROCESS-PASCAL program is structured like a state machine. Furthermore the program consist of two small procedures, a initialization procedure and a procedure which transforms address and instruction in a P-NET call into the appropriate address and instruction format on a PASILAC-system.

The structure of the system.

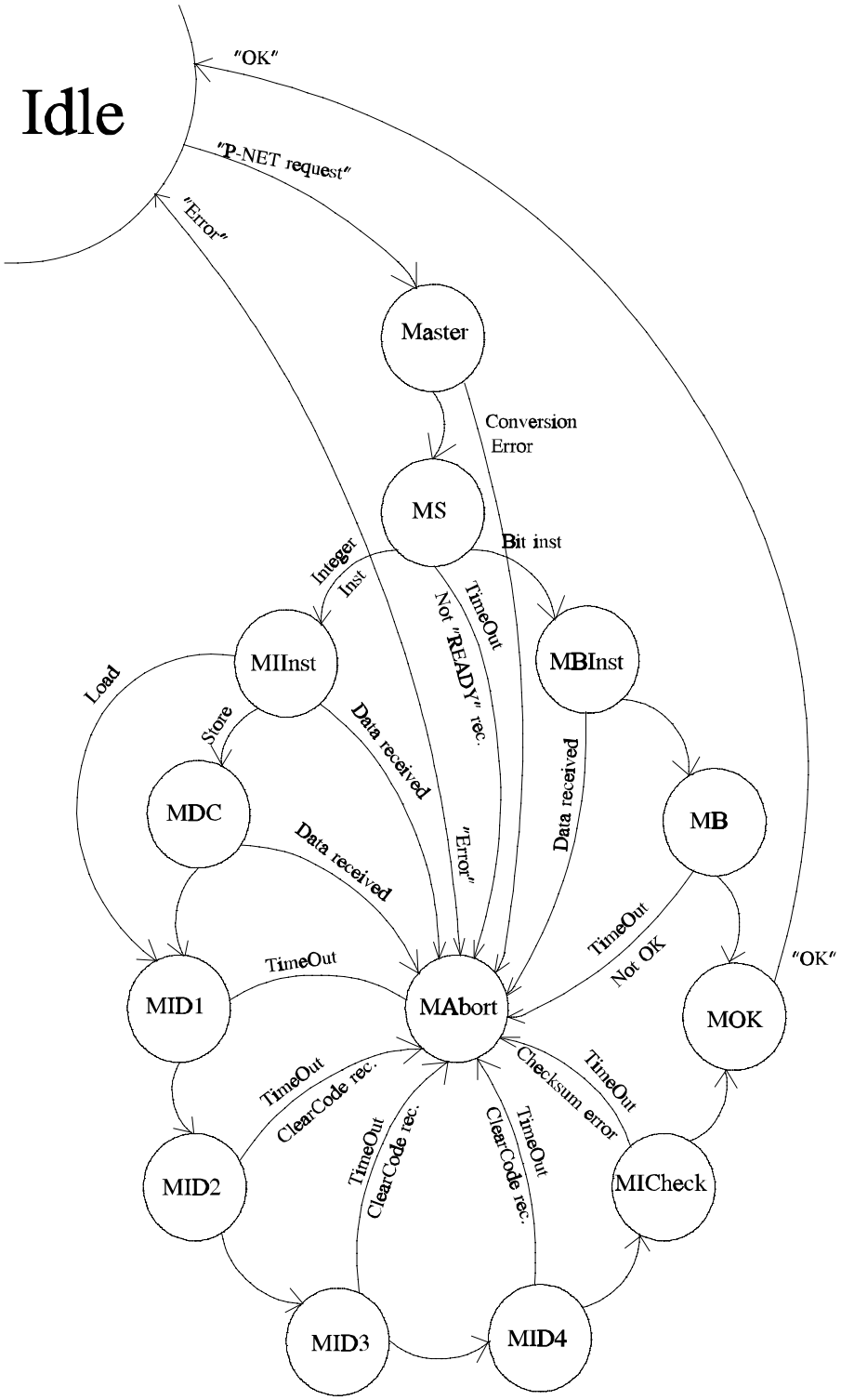
PASCOM waits for a call either from the P-NET or from the PASILAC-system. When PASCOM detects a call from the P-NET it generates an InterCom interrupt. When PASCOM detects a call from the PASILAC-system, it will send data to the RS-232 port (Port1Input).

The state machine.

The state machine consist of two side branches, one where PASCOM is the master (seen from the P-NET side), and one where PASCOM is a slave. The two side branches are tied together by a common condition 'Idle'. During the 'Idle'-state one of two events is expected: a) data on port1 (a call from the PASILAC-system), or b) PNETrequest is set by a InterCom interrupt.



Master part, states.



Master: Master. In this condition the start code and the µP-no is sent to the PASILAC-system. The start code depends on the NodeAddress in InterComRecord. If there is only a single number - or if the second number is 70H - PASCOM will send 'START0'. Otherwise it will send 'START1' or 'START2' depending on wheter the first number is 1 or 2. If there is a syntax error in the call, next condition is MAbort, otherwise the next condition is MS.

MS: MasterSend. In this condition 'READY' is expected from the PASILAC-system. When 'READY' comes, the instruction is sent. If the received data is not 'READY', or if the InputTimer runs out, next condition is MAbort. If the instruction is a bit instruction on the PASILAC-system, (ResetBit, SetBit or TestBit) next condition is MIInst.

MIInst: MasterIntegerInstruction. In this condition the Integer address is sent to the PASILAC-system. If the call is a Load-call it is followed by the check sum, and the next condition is MID1. If the call is a Store-call,the next condition is MDC. If, contrary to expectation, data is received from the PASILAC-system, the next condition is MAbort.

MDC: MasterDataCheck. In this condition data and check sum in connection with the store-call is sent to the PASILAC-system. The next condition is MID1, unless data is received from the PASILAC-system. In this case the next condition is MAbort.

MID1: MasterIntegerData1. If it was a Load-call, first data byte is expected and the next condition is MID2. If it was a Store-call, 'OK' is expected. If 'Store' is not followed by 'OK', or if TransTimer runs out before data comes, the next condition is MAbort.

MID2: MasterIntegerData2. The second data byte in the reply is expected, and the next condition is MID3. If 'CLEAR' is received from the PASILAC-system or TransTimer runs out, the next condition is MAbort.

MID3: MasterIntegerData3. The third data byte in the reply is expected, and the next condition is MID4. If 'CLEAR' is received from the PASILAC-system or TransTimer runs out, the next condition is MAbort.

MID4: MasterIntegerData4. The forth data byte in the reply is expected, and the next condition is MICheck. If 'CLEAR' is received from the PASILAC-system or TransTimer runs out, the next condition is MAbort

MICheck: MasterIntegerChecksum. In this condition the check sum is expected. If it is OK data is placed in InterComRecord and the next condition is MOK. If there is a check sum error or the TransTimer runs out, the next condition MAbort.

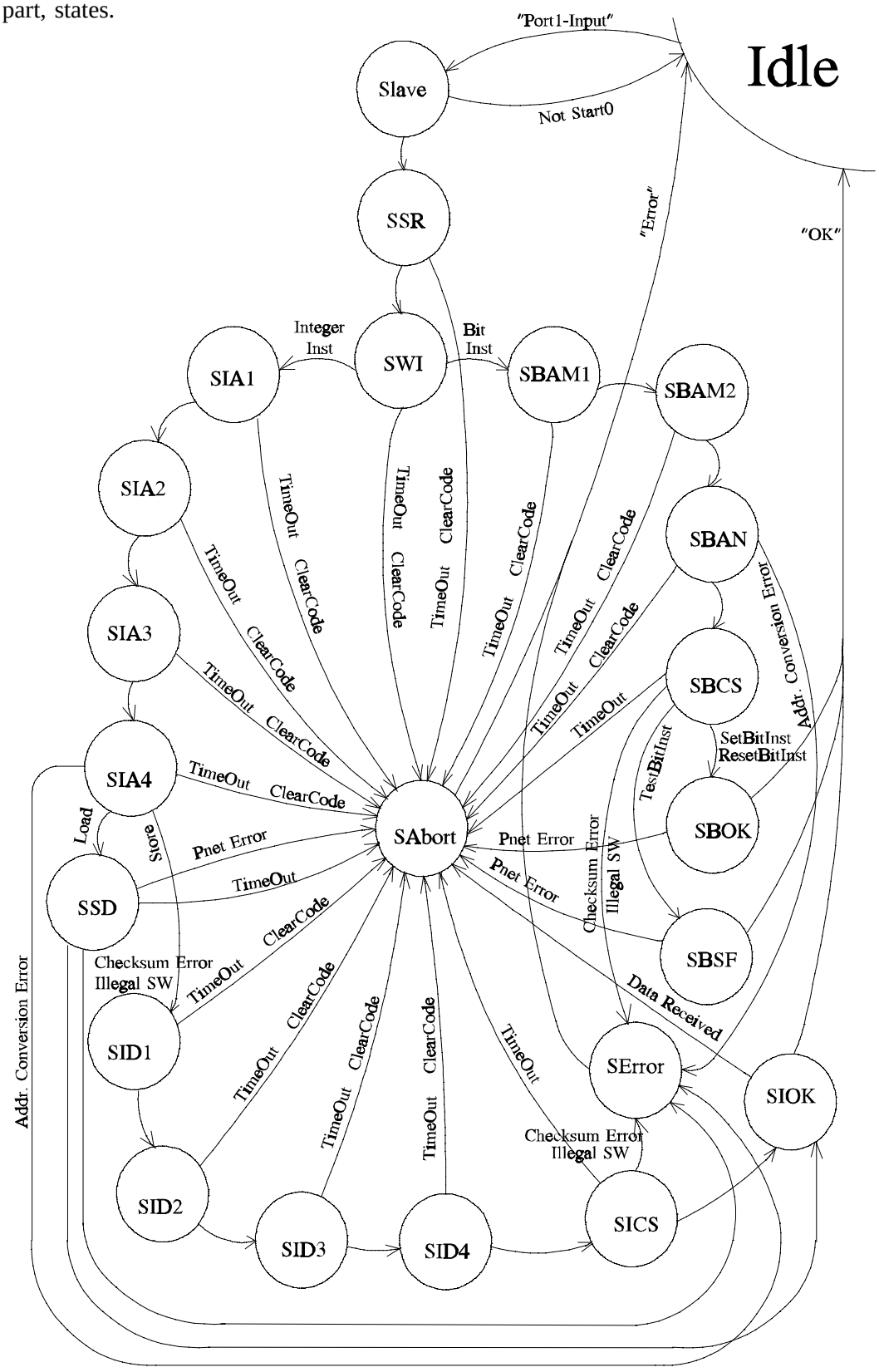
MOK: MasterOK. This is a final condition. A 'CLEAR' is sent to the PASILAC-system and OK to the P-NET. The next condition is Idle.

MBInst: MasterBitInstruction. In this condition the check sum of the call is calculated. The bit address is sent to the PASILAC-system followed by the check sum. The next condition is MB, but, if contrary to expectation, data is received from the PASILAC-system, the next condition is MAbort.

MB: MasterBit. In this condition the reply to the Bit-call is expected. If the call was a BitTest call, it is placed in InterComRecord, otherwise 'OK' is expected. The next condition is MOK. If 'OK' is not received or TransTimer runs out, the next condition is MAbort.

MAbort: MasterAbort. This is an error condition. In this condition the data flow from PASCOM to the PASILAC-system is terminated and replaced by a 'CLEAR'. If having tried a number of times equal to RetryMax, 'NoAnswer' is sent to the P-NET. The next condition is Idle.

Slave part, states.



Slave: Slave. In this condition the first character is read from the PASILAC-system. If it is a 'START0', the next condition is SSR, otherwise the next condition is Idle.

SSR: SlaveSendReady. In this condition the µP-no. is expected. This no. is not used. Then 'READY' is sent to the PASILAC-system. The next condition is SWI. If a 'CLEAR' is received or the TransTimer runs out, the next condition is SAbort.

SWI: SlaveWaitInstruction. In this condition an instruction from the PASILAC-system is expected. If it is a Bit-instruction the next condition is SBAM, otherwise the next condition is SIA1. If the TransTimer runs out the next condition is SAbort.

SIA1: SlaveIntegerAddress1. In this condition the first address byte from the PASILAC-system is expected. The next condition is SIA2. If the TransTimer runs out or a 'CLEAR' is received, the next condition is SAbort.

SIA2: SlaveIntegerAddress2. In this condition the second address byte from the PASILAC-system is expected. The next condition is SIA3. If the TransTimer runs out or a 'CLEAR' is received, the next condition is SAbort.

SIA3: SlaveIntegerAddress3. In this condition the third address byte from the PASILAC-system is expected. The next condition is SIA4. If the TransTimer runs out or a 'CLEAR' is received, the next condition is SAbort.

SIA4: SlaveIntegerAddress4. In this condition the fourth address byte from the PASILAC-system is expected. The next condition is SSD if it is a Load-instruction, otherwise the next condition is SID1. If the TransTimer runs out or a 'CLEAR' is received, the next condition is SAbort. If there is a conversion error on the address the next condition is SError.

SID1: SlaveIntegerData1. In this condition the first data byte from the PASILAC-system is expected. The next condition is SID2. If the TransTimer runs out or a 'CLEAR' is received, the next condition is SAbort.

SID2: SlaveIntegerData2. In this condition the second data byte from the PASILAC-system is expected. The next condition is SID3. If the TransTimer runs out or a 'CLEAR' is received, the next condition is SAbort.

SID3: SlaveIntegerData3. In this condition the second data byte from the PASILAC-system is expected. The next condition is SID4. If the TransTimer runs out or a 'CLEAR' is received, the next condition is SAbort.

SID4: SlaveIntegerData4. In this condition the second data byte from the PASILAC-system is expected. The next condition is SICS. If the TransTimer runs out or a 'CLEAR' is received, the next condition is SAbort.

SICS: SlaveIntegerCheckSum. In this condition the check sum is expected. If this is OK, IntPtr is used for pointing out the variable containing the received softwire address. Then data is stored on the P-NET, and an OK is sent to the PASILAC-system. The next condition is SIOK. If the TransTimer runs out the next condition is SAbort. If there is a check sum error, conversion error or a pointer error, the next condition is SError.

SSD: SlaveSendData. In this condition the check sum is expected. If this is OK, IntPtr is used for pointing out the variable containing the received softwire address. Then data is retrieved from the P-NET, and a reply for the PASILAC-system is made and sent. The next condition is SIOK. If the TransTimer runs out the next condition is SAbort. If there is a check sum error or a pointer error, the next condition is SError. If there is a transmission error on the P-NET, the next condition is ASbort.

SIOK: SlaveIntegerOK. In this condition the reply block is expected. The next condition is Idle. If any data is received from the PASILAC-system, the next condition is SAbort.

SBAM1: SlaveBitAddressM1. In this condition the first bit address byte (M1) from the PASILAC-system is expected. The next condition is SBAM2. If the TransTimer runs out or 'CLEAR' is received, the next condition is SAbort.

SBAM2: SlaveBitAddressM2. In this condition the second bit address byte (M2) from the PASILAC-system is expected. The next condition is SBAN. If the TransTimer runs out or 'CLEAR' is received, the next condition is SAbort.

SBAN: SlaveBitAddressN. In this condition the third bit address byte (N) from the PASILAC-system is expected. The next condition is SBCS. If the TransTimer runs out or 'CLEAR' is received, the next condition SAbort. If there is a conversion error on the address, the next condition is SError.

SBCS: SlaveBitCheckSum. In this condition the check sum is expected. If this is OK, BoolPtr is used for pointing out the variable with the received softwire address. If the instruction is TestBit the next condition is SBSF. Otherwise the next condition is SBOK. If TransTimer runs out, the next condition is SAbort. If there is check sum error, conversion error or pointer error, the next condition is SError.

SBOK: SlaveBitOK. In this condition the variable pointed out on the P-NET by BoolPtr is set or reset. Then OK is sent to the PASILAC-system and the next condition is Idle. If there is a transmission error on the P-NET, the next condition is SAbort.

SBSF: SlaveBitSendFlag. In this condition the variable on the P-NET pointed out by BoolPtr is read. Then the value of the variable is sent to the PASILAC-system and the next condition is Idle. If there is a transmission error on the P-NET, the next condition is SAbort.

SAbort: SlaveAbort. In this condition the buffers and port1 are initialized. Then 'CLEAR' is sent to the PASILAC-system. The next condition is Idle.

SError: SlaveError. In this condition the buffers and port1 are initialized. Then ERROR is sent to the PASILAC-system. The next condition is Idle.