Soft & Control Technology s.r.o.

Battery powered wireless telemetric systems of BaWiT series

Type series BaWiT- A/B/C/E

SPL ASM

Programming Language User manual

Document version 1.1
Revision 1.10
Dated 18th of June 2014

Content

1 Abbreviations

SPL ASM - SCT PLCprogramming Language ASM.
EVENT - language event that is served by device.
Interpret - execute Bytecode instructions.
Compile - to create bytecode from source file.
ByteCode - coded instructions that can be interpreted by SPL interpreter in device.

2 SPL ASM Language

SPL ASM language is a low-level programming language that can be used for writing event handlers for BaWiT devices. Events are raised during data points processing, device service, system resources activity or are generated directly by language instructions. SPL ASM compiler generates BYTECODE from the source files. This BYTECODE is part of the configuration and is interpreted by SPL ASM interpreter in BaWiT device.

3 Events Description

Events are raised during data points processing:

- Reach or overrun of user data point bounds (analog and digitals)
- New measured values (can be equal to previous measured value)
- Changed value (change is determined by sensibility parameter in device configuration)
- Stabilized value (value is not changed in specified time)

System events:

- Device restart
- Change from SLEEP mode to RUN mode
- Modem turn on, modem failure, ...

Language events:

- Timer elapse
- Alarm
- Direct event raising

4 Language Execution

When the event is raised, it is queued for processing by interpreter. PowerManager allways determines start of event execution (optimalization, etc.). Event is processed parallelly with all other tasks defined by device configuration.

5 Limitations

The number of registers is limited to 8. The number of logical OR is max. 8. The number of inner if-if-else-else conditions can be max. 8. The number of timers is 8. The number of alarms is 8.

5.1 Used Data Types

UINT – 32-bit unsigned integer
INT – 32-bit signed integer
FLOAT – 32-bit float (IEEE754)

6 Preprocessor Commands

Although the described commands are labeled as preprocessor there is no classic preprocessor (as in C language). Source file processing is not divided into preprocessor and compiler; there is only the compiler that processes individual commands and instructions. As an instruction separator in source files is used end-of-line character. Compiler is case insensitive.

6.1 #CONST

Defines constant that can be used in expressions. The value of constant must be computable during compilation process.

```
#CONST constant = constexpression {, constant = constexpression }
```

Constant can be defined only outside the action body. Once defined constant can be redefined and in next source file processing the new value is used. Constants are visible also outside the actual source file (see `#INCLUDE` command).

Constant type is automatically determined from constant expression (uint, int, float, string).

Example:

```
#CONST i = 1, j = i + 1, k = -1
#CONST tlak = "FCGPA_"
#CONST tlak1 = tlak + "1", tlak2 = tlak + "2"
#CONST x = 3.5234, y = x + i + 0.5
```

In the example above the constant have these type:

Constant	Type
i, j	uint
k	int
x, y	float
tlak, tlak1, tlak2	string

6.2 #ACTION … #END

Defines user action.

```
#ACTION ActionName EventId EventIndex
instructions
#END
```

`ActionName` – name of the action. `EventId` – event type. `EventIndex` – event index. As an `EventId` and `EventIndex` user can use integer value or integer constant identifier.

Actions are visible outside the actual source file (see `#INCLUDE`).

6.3 #DEBUG

Turns on/off debug and break bit in instructions.

```
#DEBUG [ON | OFF] [BREAK]
```

Command can be used inside or outside the action body. By using the `BREAK` parameter, the next instruction will be generated with BREAK bit set. By using the parameters `ON` / `OFF` all next instructions will have the DEBUG bit set. Command's domain is only in actual source file.

In the example below the instructions `MOV`, `ADD`, `DIV`, `MUL` will have DEBUG bit set to 1. Instruction `DIV` will have also the BREAK set to 1. Instruction `NEW` and `SUB` will have DEBUG and BREAK bits set to 0. The using of `#DEBUG` without parameters is ignored by compiler.

Example:

```
NEW
#DEBUG ON
MOV CNT[0], 5
```

```
ADD CNT[0], CNT[0], CNT[1]
#DEBUG BREAK
DIV CNT[0], CNT[0], CNT[2]
MUL CNT[0], CNT[0], 2
#DEBUG OFF
SUB CNT[0], CNT[0], CNT[3]
```

6.4 #RESULT

Determines the default result type of instructions `MOV`, `ADD`, `SUB`, `MUL`, `DIV`, `MOD`, `AND`, `IOR` and `XOR`, if the result is not explicitly stated.

```
#RESULT UINT | INT | FLOAT
```

Command can be used inside or outside the action body. In both cases it works as global setting. Command domain is from its occurence till the next `#RESULT` command or end of file.

6.5 #PARAM

Command is used for setting additional configuration parameters of action. If the command is used outside the action body it affects all subsequent actions. If the command is used inside the action body it affects current action and also all subsequent actions. Command domain is only in current source file.

```
#PARAM name constexpression
```

Supported parameters (`name`) are `LogDestination`, `LogNumber`, `LogSize`. Constant expression `constexpression` must be an integer.

6.6 #INCLUDE

Is used for insertion and subsequent processing of other source file from actually processed source file.

```
#INCLUDE <filename>
```

In the included file are visible all defined constants and actions defined up to included file. After finishing the included file processing in the actual source file will be visible all newly defined constants (or redefined constants) and actions. The states of `#DEBUG`, `#PARAM` and `#RESULT` commands are not transfered into actual source file.
The included file will be searched in system specified directories defined by compiler.

Example:

File A

```
#CONST X = 1 << MOC
#DEBUG ON
```

File B

```
#DEBUG OFF
#CONST MOC = 2
#INCLUDE <A>
#CONST Y = X - 1
```

In file *B* user can user constant `X` defined in file *A* from the `#INCLUDE` command. Command `#DEBUG ON` does not affect file *B*, and respectively `#DEBUG OFF` does not affect file *A*. In file A the constant `MOC` is used that is not defined there, but is defined in file *B* before the processing of file *A*.

6.7 #STRING, #SHORTSTRING, #LONGSTRING

Is used for texts definition, which are generated during source code compilation.

```
#STRING id, text
```

```
#SHORTSTRING id, text
#LONGSTRING id, text
```

For every directive compiler generates text with entered id, which is then stored in compiler's output. As `id` user can enter any constant expression that evaluates to non-negative integer. As `text` user can enter any constant expression that evaluates to string.
Directive `#SHORTSTRING` defines that final text can be max. 8 characters long. Similarly for directive `#LONGSTRING` text can be max. 64 characters long. Directive `#STRING` defines that length of final text is determined by compiler (allways short text, allways long text or determined automatically).

6.8 #VALUE

Is used for values definition, which are generated during source code compilation and are stored in device configuration.

```
#VALUE id, ident = (UINT | INT | FLOAT) value
```

For every directive compiler generates text with entered id, which is then stored in compiler's output. As `id` user can enter any constant expression that evaluates to non-negative integer. As `value` user can also enter any constant expression. For every value user must explicitly define type of value which will be used for storage in configuration.

7 Comments

Semicolon character (`;`), or double angle bracket characters (`//`) are used for writing comments. Every character after semicolon or double angle bracket to the end of the row is ignored by the compiler.

8 Constant Expressions

Constant expressions are the expressions that can be computed already in the time of the source file compilation. They are created out of defined constants (`#CONST`) or straightly out of input numerical values. It is possible to use following operators in the constant expressions:

`+ - * /`	Addition, Substraction, Multiplication, and Division.
`\ %`	Integer division and remainder after the integer division.
`& | ^`	Bitwise AND, bitwise OR, and exclusive bitwise OR.
`~`	Bit negation.
`<< >>`	Bit left shift and right shift.

Operators `+ -` `|` have lesser priority than the other operators. Operators with the same priority are processed in the order as they occurred in the expression. The order of the operators processing can be determined with the use of brackets `( )`.
In constant expressions user can also use following functions:

DT (datetime)

Converts entered date and time `datetime` to integer value (number of seconds from 1st January 2000 00:00:00). Parameter datetime must be of type string and must be entered in following format: `yyyy-mm-dd hh:mm:ss`. User can omit date or time part. If user omits date part, compiler will use date 1st January 2000. If user omits time part, compiler will use 00:00:00.

9 Source File Structure

Source file consists of constant definition and single actions. Body of the action can contain several instructions.

9.1 Language Instructions

9.1.1 END

Unconditional end of the action code execution.

9.1.2 NEWIF

Beginning of the new condition.

9.1.3 OR

Logical Or.

9.1.4 NOP

No operation.

9.1.5 ENDC

Conditional end of the action code execution.

9.1.6 EQ [INT | UINT | FLOAT] Src1, Src2

Operand comparison (Src1 = Src2). A register or constant expression can be used as Src.

9.1.7 NE [INT | UINT | FLOAT] Src1, Src2

Operand comparison (Src1 <> Src2). A register or constant expression can be used as Src.

9.1.8 LT [INT | UINT | FLOAT] Src1, Src2

Operand comparison (Src1 < Src2). A register or constant expression can be used as Src.

9.1.9 LE [INT | UINT | FLOAT] Src1, Src2

Operand comparison (Src1 <= Src2). A register or constant expression can be used as Src.

9.1.10 GT [INT | UINT | FLOAT] Src1, Src2

Operand comparison (Src1 > Src2). A register or constant expression can be used as Src.

9.1.11 GE [INT | UINT | FLOAT] Src1, Src2

Operand comparison (Src1 >= Src2). A register or constant expression can be used as Src.

9.1.12 ELSE

Beginning of the instructions that will be executed when the condition is not evaluated as true.

9.1.13 ENDIF

End of condition instructions.

9.1.14 TRUE

Allways true.

9.1.15 EE Ident Value

Raising event (action event handler) with the name Ident and parameter Value. Event handler is not executed immediately; it is queued to the raised events queue, and it is waiting to be processed.

9.1.16 MOV [INT | UINT | FLOAT] Dest, Src1

It sets operand value Dest to the operand value Src1. A register can be used as Dest; a register or constant expression can be used as Src. If requested type of result is entered. If the requested type of the result is specified, the resulting value will be saved with the specified type. Otherwise, the instructions will be used based on the last command `#RESULT`.

9.1.17 ADD [INT | UINT | FLOAT] Dest, Src1, Src2

Counts up operand values Src1 and Src2, and saves the result into Dest. A register can be used as Dest; a register or a constant expression can be used as Src. If the requested type of the result is specified, the resulting value will be saved with the specified type. Otherwise, the instructions will be used based on the last command `#RESULT`.

9.1.18 SUB [INT | UINT | FLOAT] Dest, Src1, Src2

Subtracts operand values Src1 and Src2, and saves the result into Dest. A register can be used as Dest; a register or a constant expression can be used as Src. If the requested type of the result is specified, the resulting value will be saved with the specified type. Otherwise, the instructions will be used based on the last command `#RESULT`.

9.1.19 MUL [INT | UINT | FLOAT] Dest, Src1, Src2

Multiplies operand values Src1 and Src2, and saves the result into Dest. A register can be used as Dest; a register or a constant expression can be used as Src. If the requested type of the result is specified, the resulting value will be saved with the specified type. Otherwise, the instructions will be used based on the last command `#RESULT`.

9.1.20 DIV [INT | UINT | FLOAT] Dest, Src1, Src2

Divides operand values Src1 and Src2, and saves the result into Dest. A register can be used as Dest; a register or a constant expression can be used as Src. If the requested type of the result is specified, the resulting value will be saved with the specified type. Otherwise, the instructions will be used based on the last command `#RESULT`.

9.1.21 MOD [INT | UINT | FLOAT] Dest, Src1, Src2

Saves the reminder after the deduction of Src1 by Src2 into Dest. A register can be used as Dest; a register or a constant expression can be used as Src. If the requested type of the result is specified, the resulting value will be saved with the specified type. Otherwise, the instructions will be used based on the last command `#RESULT`.

9.1.22 AND [INT | UINT | FLOAT] Dest, Src1, Src2

Saves the result of bit multiplication of Src1 and Src2 into Dest. A register can be used as Dest; a register or a constant expression can be used as Src. If the requested type of the result is specified, the resulting value will be saved with the specified type. Otherwise, the instructions will be used based on the last command `#RESULT`.

9.1.23 IOR [INT | UINT | FLOAT] Dest, Src1, Src2

Saves the result of bitwise-or of Src1 and Src2 into Dest. A register can be used as Dest; a register or a constant expression can be used as Src. If the requested type of the result is specified, the resulting value will be saved with the specified type. Otherwise, the instructions will be used based on the last command `#RESULT`.

9.1.24 XOR [INT | UINT | FLOAT] Dest, Src1, Src2

Saves the result of exlusive bitwise-or of Src1 and Src2 into Dest. A register can be used as Dest; a register or a constant expression can be used as Src. If the requested type of the result is specified, the resulting value will be saved with the specified type. Otherwise, the instructions will be used based on the last command `#RESULT`.

9.1.25 DIF [INT | UINT | FLOAT] Dest, Src1, Src2

Subtracts operand values Src1 and Src2, and saves the absolute value of the result into Dest. A register can be used as Dest; a register or a constant expression can be used as Src. If the requested type of the result is specified, the resulting value will be saved with the specified type. Otherwise, the instructions will be used based on the last command `#RESULT`.

9.1.26 MEQ [INT | UINT | FLOAT] Dest, Src1, Src2

Saves the result of expression Src1==Src2 into Dest. A register can be used as Dest; a register or a constant expression can be used as Src. If the requested type of the result is specified, the resulting value will be saved with the specified type. Otherwise, the instructions will be used based on the last command `#RESULT`.

9.1.27 MNE [INT | UINT | FLOAT] Dest, Src1, Src2

Saves the result of expression Src1!=Src2 into Dest. A register can be used as Dest; a register or a constant expression can be used as Src. If the requested type of the result is specified, the resulting value will be saved with the specified type. Otherwise, the instructions will be used based on the last command `#RESULT`.

9.1.28 MGT [INT | UINT | FLOAT] Dest, Src1, Src2

Saves the result of expression Src1>Src2 into Dest. A register can be used as Dest; a register or a constant expression can be used as Src. If the requested type of the result is specified, the resulting value will be saved with the specified type. Otherwise, the instructions will be used based on the last command `#RESULT`.

9.1.29 MGE [INT | UINT | FLOAT] Dest, Src1, Src2

Saves the result of expression Src1>=Src2 into Dest. A register can be used as Dest; a register or a constant expression can be used as Src. If the requested type of the result is specified, the resulting value will be saved with the specified type. Otherwise, the instructions will be used based on the last command `#RESULT`.

9.1.30 MLT [INT | UINT | FLOAT] Dest, Src1, Src2

Saves the result of expression Src1<Src2 into Dest. A register can be used as Dest; a register or a constant expression can be used as Src. If the requested type of the result is specified, the resulting value will be saved with the specified type. Otherwise, the instructions will be used based on the last command `#RESULT`.

9.1.31 MLE [INT | UINT | FLOAT] Dest, Src1, Src2

Saves the result of expression Src1<=Src2 into Dest. A register can be used as Dest; a register or a constant expression can be used as Src. If the requested type of the result is specified, the resulting value will be saved with the specified type. Otherwise, the instructions will be used based on the last command `#RESULT`.

9.1.32 JMP label

Jumps to the the labeled instruction. Jumps are possible only within one action. In two different actions, it is possible to label the jump place identically. Instruction `JMP` will be translated as

```
MOV UINT STS[0], absolute_offset
```

Value `absolute_offset` will be calculated by the compiler. At the moment, no instruction for relative offset jump exists. It is necessary to use `ADD UINT STS[0], STS[0], relative_offset`.
In the example, the instruction `JMP` makes jump to instruction `ADD`.

JMP example:

```
MOV CNT[0], DP[0]       // sets Counter0 to the value of data point addr. 0
someplace:              // label
ADD CNT[0], CNT[0], 5   // Adds 5 to Counter0
MUL CNT[0], CNT[0], 2   // Multiplies Counter0 by constant 2
NEW                     // New condition
LT CNT[0], 100          // Test if Counter0 is less than 100
JMP miesto              // Jump to someplace
NEW
MOV DP[0], CNT[0]       // Sets data point addr.0 value to Counter0
```

9.2 Registers

NO	No operand. Implicitly evaluated as UINT 0.
W[*const***]**	Temporary action registers. *const* – integer constant expression
DP[*addr***]** **DP[***shortname***]**	Valus of data points. *addr* – 2-byte data point address (integer constant expression) *shortname* – shortened label of the data point (string or string constant identifier)
DP[*addr***].***constident*** **DP[***shortname***].***constident***	Configuration values of data points. *addr* – 2-byte address of data point (integer constant expression) *shortname* – shortened label of the data point (string or string constant identifier) *constident* – parameter identification number of data point configuration item (positive integer or integer constant identifier) • 0 – own value • 1 – SCT time • 2 – Quality Descriptor • Other – data point parameters based on the numeration
CFG[*area,item_index,parameter***]**	Value of configuration table *parameter* in the *area* and the item with the number *item_index*. *area* – configuration area identification number (integer constant expression) • 1 – Objects • 2 – Slave devices • Etc. *item_index* – item serial number in the configuration (integer constant expression) *parameter* – parameter identification number in the configuration (integer or integer constant identifier)
STS[*table,item,parameter***]**	Value of *parameter* of the state table with id number

	table in the item with id number *item*. *table* – state table identification number (integer constant expression) *item* – identification number of the item in the state table (integer constant expression) *parameter* – identification number of the parameter in the configuration (integer or integer constant identifier)
CNT[*const*]	Counter.
CTC[*const*]	Timer.
ALR[*const*]	Alarm.

Only constant expressions must be used as indexes for access to the actual register values (that is, they must be computable in the time of compilation).

10 Grammar

Grammar is of LL(1) type and is written in slightly modified EBNF. Starting non-terminal of grammar is **SPLAsm**.

```
EOLN      = '\n' | '\r' .
Sign      = "+" | "-" .
Digit     = "0".."9" .
HexDigit  = "0".."9" | "A".."F" | "a".."f" .
Letter    = "_" | "A".."Z" | "a".."z" .
Exponent  = "E" | "e" .
noQuote   = ANY - '"' - EOLN .
noAngleBr = ANY - '<' - '>' - EOLN .
ident     = Letter { Letter | Digit } .
integer   = (Digit { Digit }) | ("0x" HexDigit {HexDigit}) .
real      = [ Digit { Digit } ] "." Digit { Digit } [ Exponent [ Sign ] Digit { Digit } ] .
string    = '"' { noQuote | '\"' } '"' .
angleStr  = '<' { noAngleBr } '>' .
SPLAsm  = { Block } EOF .
Comment = (";" | "//") {ANY} .
Block = [ ConstStmt | ValueStmt | StringStmt | ActionStmt | DebugStmt | ResultStmt | ParamStmt |
        IncludeStmt ] [Comment] EOLN .
ConstStmt = "#CONST" ConstDecl { "," ConstDecl } .
ConstDecl = ident "=" ConstExpr .
ConstExpr = ConstTerm { ("+" | "-" | "|") ConstTerm } .
ConstTerm = ConstFactor { ("*" | "/" | "%" | "&" | "^" | "<<" | ">>" ) ConstFactor } .
ConstFactor = ident | integer | real | string | "DT" "(" ConstExpr ")" |
        ( "(" ConstExpr ")" ) | ("~" ConstFactor) |  ( ("+" | "-" ) ConstFactor ) .
ValueStmt = "#VALUE" ConstExpr , ident = ResultType ConstExpr .
StringStmt = ("#STRING" | "#SHORTSTRING" | "#LONGSTRING") ConstExpr "," ConstExpr .
ResultStmt = "#RESULT" ResultType .
ResultType = "UINT" | "INT" | "FLOAT" .
DebugStmt = "#DEBUG" ["ON" | "OFF"] ["BREAK"] .
ParamStmt = "#PARAM" ident ConstExpr .
IncludeStmt = "#INCLUDE" angleStr .
ActionStmt = "#ACTION" ActionHeader { Command } "#END" .
ActionHeader = ident (ident | integer) (ident | integer) [Comment] EOLN .
Command = [ (ident ":") | Instruction | DebugStmt | ResultStmt | ParamStmt ] [Comment] EOLN .
Instruction = "END" | "NEWIF" | "OR" | "ELSE" | "TRUE"
    | ( ("EQ" | "NE" | "LT" | "LE" | "GT" | "GE") [ResultType] Operand "," Operand)
    | "NOP" | "ENDC"
    | ( "EE" ident Operand )
    | ( "MOV" [ResultType] DestOperand "," Operand )
    | ( ("ADD" | "SUB" | "MUL" | "DIV" | "MOD" | "AND" | "IOR" | "XOR"
        | "DIF" | "MEQ" | "MNE" | "MGT" | "MGE" | "MLT" | "MLE" )
        [ResultType] DestOperand "," Operand "," Operand )
```

```
    | ( "JMP" ident ) .
Operand = DestOperand | ConstExpr .
DestOperand = RegisterDP | RegisterCFG | RegisterSTS | RegisterOther | RegisterNO .
RegisterArray = "[" ConstExpr "," ConstExpr "," ConstExpr "]" .
RegisterDP = "DP" "[" ConstExpr "]" ["." (ident | integer)] .
RegisterCFG = "CFG" RegisterArray .
RegisterSTS = "STS" RegisterArray .
RegisterOther = ("W" | "CNT" | "CTC" | "ALR") "[" ConstExpr "]" .
RegisterNO = "NO" .
```

any character	`ANY`
end of file	`EOF`
non-terminal definition	`=`
or	`|`
optional	`[...]`
repetition >= 0	`{...}`
group	`(...)`
terminal	`"..." '...'`
rule end	`.`
interval / set	`..`
exclusion	`-`

Tab. 1 Character meanings used in EBNF

11 Debugging

For each instruction it is possible to turn on whether the given instruction participates in detail debug output. When executing user event, the file is generated in directory 0:/SPL of BaWiT file system with the name splABDC.log. ABCD contains hexadecimal number of user defined event. Listing is as follows:

```
SPL Run Action ID=1 DT=2009-09-04 10:07:27 BRK=0 STS=0 Name=ZmenaPoctuBatt
    Event=48, Index=44, Param=1
SPL INS: EXE_MOV DEBUG CndTst=1 ExeTst=1 Run=1
    DST: W[3].F=1.000000
    SRC1:DP[A=1,O=0,T=0,B=0,I=0].F=1.000000 Text=batts VelID=5 <Batts>
    SRC2:NO.F=0.000000
    W:   [0]=unk [1]=unk [2]=unk [3]=1.000000 [4]=unk [5]=unk [6]=unk [7]=unk
    CNT: [0]=unk [1]=unk [2]=unk [3]=unk [4]=unk [5]=unk [6]=unk [7]=unk
    CTC: [0]=0 [1]=0 [2]=0 [3]=0 [4]=0 [5]=0 [6]=0 [7]=0
    ALR: [0]=0 U [1]=0 U [2]=0 U [3]=0 U [4]=0 U [5]=0 U [6]=0 U [7]=0 U
      T: [0]=x01.0
SPL INS: EXE_DIV DEBUG CndTst=1 ExeTst=1 Run=1
    DST: W[3].F=0.010000
    SRC1:W[3].F=1.000000
    SRC2:CONS.F=100.000000
    W:   [0]=unk [1]=unk [2]=unk [3]=0.010000 [4]=unk [5]=unk [6]=unk [7]=unk
    CNT: [0]=unk [1]=unk [2]=unk [3]=unk [4]=unk [5]=unk [6]=unk [7]=unk
    CTC: [0]=0 [1]=0 [2]=0 [3]=0 [4]=0 [5]=0 [6]=0 [7]=0
    ALR: [0]=0 U [1]=0 U [2]=0 U [3]=0 U [4]=0 U [5]=0 U [6]=0 U [7]=0 U
      T: [0]=x01.0
SPL INS: EXE_MOV DEBUG CndTst=1 ExeTst=1 Run=1
    DST: DP[A=1,O=0,T=1,B=0,I=0].F=0.010000 Text=ndb VelID=12 <NDB>
    SRC1:W[3].F=0.010000
    SRC2:NO.F=0.000000
    W:   [0]=unk [1]=unk [2]=unk [3]=0.010000 [4]=unk [5]=unk [6]=unk [7]=unk
    CNT: [0]=unk [1]=unk [2]=unk [3]=unk [4]=unk [5]=unk [6]=unk [7]=unk
    CTC: [0]=0 [1]=0 [2]=0 [3]=0 [4]=0 [5]=0 [6]=0 [7]=0
    ALR: [0]=0 U [1]=0 U [2]=0 U [3]=0 U [4]=0 U [5]=0 U [6]=0 U [7]=0 U
      T: [0]=x01.0
SPL action Time=1709msec
```

If the logging is disabled, only record of user event execution is stored:

```
SPL Run Action ID=0 DT=2009-09-02 14:39:58 BRK=0 STS=0 Name=Reset
    Event=16, Index=1, Param=0
SPL action Time=0msec
```

12 Examples

12.1 Cooling control

Layout
Supervise user set temperature limit. When exceeding limit turn cooling fan, control fan activity and indicate cooling error after each fan start (at latest 5 seconds since start).
Description of technical solution
Temperature measuring by PT1000 on analog input 0. Activity of cooling fan is indicated on digital input 0. Fan is switched by digital output 0. Error indication is on digital output 1.
Configuration description
Configuration must have correctly configured 4 data points according to technical description of the solution. It is necessary to set following parameters of device configuration for correct operation according to layout:

- Data point for actual temperature
  - must have the same signature (for example **Temperature**) as in the source program (see bellow)
  - set condition of user event generating to **New value**
  - index of user event within range 0..65535 set by user (for example 160 – when measuring new value of data point the event with set number 160 will be generated)
  - correctly set high limit (based on user requirements)
- Data point to indicate fan operation
  - must have the same signature (for example **CoolRun**) as in the source program (see bellow)
- Data point for cooling error indication
  - must have the same signature (for example **CoolErr**) as in the source program (see bellow)
- Data point for controlling of fan operation
  - Must have the same signature (for example **Cooler**) as in the source program (see bellow)

Solution description
Temperature measurement is running based on user configured time diagram for analog input (see configuration on CD). When reading new value the event is generated. Servicing of this event will compare if actual temperature is higher or equal to configured high limit of this data point. When meeting the criteria the value of data point **Cooler** will be set to 1 and this will turn on the fan. In the same time this state will be registered in internal language register. Finally the timer will start and after 5 seconds invoke event to control cooling operation. In case that the temperature is not higher than configured limit, the value of data point **Cooler** to 0 (turns off the fan). This state will be again registered in register and executes timer for cooling operation control.
After elapsing time for cooling control the event is generated. This event will control if the registered state in internal register is equal to value of data point **CoolRun** (indication, cooling on/off). If the condition is fulfilled, the value 0 will be registered to data point **CoolErr** (error is not indicated). In case that condition is not fulfilled the error is indicated (value 1 on data point **CoolErr**) and in the same time it will again starts the time for further cooling control.

Program

```
// file with device system constants
#INCLUDE <system.spi>

// constants
#CONST TurnOn = 1
#CONST TurnOff = 0
```

```
// datapoint names constants
#CONST Temperature = "Temper"
#CONST Cooler = "Cooler"
#CONST CoolerRun = "CoolRun"
#CONST CoolerError = "CoolErr"

// constant for register id
#CONST CoolerId = 0

// no debug information
#DEBUG OFF
#ACTION AfterResetEvent Action_SYS Action_SYS_RST
// device "Reset" handler
#END

// debug information on
#DEBUG ON
// ID debug log
#PARAM LogNumber 160
// maximum debug log size
#PARAM LogSize 4096
// new data point value event handler
#ACTION NewTemperature Action_DP_NEW 160
    // test, if the value of data point Teplota higher or equal
    // to high limit of data point Temperature
      GE FLOAT DP[Temperature], DP[Temperature].cfgUserHI
            // registering value of constant Zapnut to data point
            MOV FLOAT DP[Cooler], TurnOn
            // registering value of constant Zapnut to register of interpreter W
            MOV W[CoolerId], TurnOn
            // setinng timer to 5 seconds
            MOV CTC[CoolerId], 5
      // if the test was false (Teplota is lower than high limit)
      ELSE
            // set value of constant TurnOff to data point
            MOV FLOAT DP[Cooler], TurnOff
            // register value of constant Vypnut to interpreter W register
            MOV W[CoolerId], TurnOff
            // setting timer to 5 seconds
            MOV CTC[CoolerId], 5
      // end condition (does not have to be, because the action finishes)
      ENDIF
#END

#PARAM LogNumber CoolerId
#PARAM LogSize 2048
// servicing of event after elapsing timer CoolerId
#ACTION CoolingTest Action_CTC CoolerId
      // test, if the value of CoolerId register is equal to value of data point
CoolerRun
      EQ FLOAT W[CoolerId], DP[CoolerRun]
            // registers value TurnOff to data point CoolerError
            MOV FLOAT DP[CoolerError], TurnOff
      // if the test was false
      ELSE
            // registers constant TurnOn value to data point CoolerError
            MOV FLOAT DP[CoolerError], TurnOn
            // setting timer CoolerId to 1 second
            MOV CTC[CoolerId], 1
      // end of condition (does not have to, because the action finishes)
      ENDIF
```

```
#END
```

Notes

Whole example with complete configuration is available on istallation CD for application **K2config** in directory **Examples\Ex1**.

12.2 Measuring of weight

Layout

Based on measured weight it is needed to set value of current loop 4-20 mA according to hardware limits of weight sensor.

Technical solution description

Weight measuring by Tensometric Bridge on analog input 0.

Configuration description

Configuration must have 2 correctly configured data points based on layout and technical solution description. It is necessary to set following parameters of device configuration for correct operation according to layout:

- Data point for actual weight
  - Must have the same signature (for example **Weight**) as in the source program (see bellow)
  - set condition of user event generating to **New value**
  - index of user event within range 0..65535 set by user (for example 150 – when measuring new value of data point the event with set number 150 will be generated)
  - correctly set low limit (based on user requirements)
  - correctly set high limit (based on user requirements)
- Data point for corrected weight value
  - must have the same signature (for example **AOUT**) as in the source program (see bellow)
  - correctly set low limit (based on user requirements)
  - correctly set high limit (based on user requirements)

Solution description

Weight measuring is running based on user configured time diagram for analog input. When reading new value the event is generated. Servicing of this event will calculate new value of data point **AOUT** based on set low and high limits of data points **Weight** and **AOUT** by linear transformation.

Program

```
// file containing device system constants
#INCLUDE <system.spi>

// constants for marking the data points in configuration
#CONST Weight = "Weight"
#CONST WeightOut = "AOUT"

// enabling debug information for all next actions
#DEBUG  ON
// debug log ID
#PARAM LogNumber 150
// maximal size of debug log
#PARAM LogSize 4096
#ACTION NewWeight Action_DP_NEW 150
    // calculation of coefficients for linear transformation
    // k = (y2-y1) / (x2-x1)
      SUB  FLOAT W[0], DP[WeightOut].cfgHwMax, DP[WeightOut].cfgHwMin
      SUB  FLOAT W[1], DP[Weight].cfgHwMax, DP[Weight].cfgHwMin
      // k = W[0]
      DIV  FLOAT W[0], W[0], W[1]

      // q = y1 - k * x1
```

```
    MUL   FLOAT W[1], W[0], DP[Weight].cfgHwMin
    // q = W[1]
    SUB   FLOAT W[1], DP[WeightOut].cfgHwMin, W[1]

    // y = k*x + q
    MUL   FLOAT W[2], W[0], DP[Weight]
    ADD   FLOAT W[2], W[2], W[1]
    // new value is set to data point - calculated value y = W[2]
    MOV   FLOAT DP[WeightOut], W[2]
#END
```

Notes

Whole example with complete configuration is available on istallation CD for application **K2config** in directory **Examples\Ex2**.

12.3 GSM gate

Layout

Allow to multiple users control opening/closing a gate. If configured user rings through, this will execute sequence of gate opening. It is requested to automatically close the gate after 2 minutes, if it is still opened.

Technical solution description

Gate controlling is connected to digital output 0. Position switch of gate closing is connected to digital input 0.

Configuration description

Configuration must have 2 correctly configured data points based on layout and technical solution description. It is necessary to set following parameters of device configuration for correct operation according to layout:

- Data point of position switch for closing the gate
  - Must have the same signature (for example **GateSt**) as in the source program (see bellow)
  - set condition of user event generating to **Changed value**
  - index of user event within range 0..65535 set by user (for example 200 – when measuring new value of data point the event with set number 200 will be generated)
- Data point for gate controlling
  - Must have the same signature (for example **GateIm**) as in the source program (see below)

Solution description

Sequence of gate opening is executed by ring through event and verifying the user. Servicing will set impulse on data point **GateIm** to 1 and starts 1 second timer. After elapsing time service of timer will set impulse on data point **GateIm** to 0.

Each change of data point **GateSt** value will verifies if value of data point **GateSt** is equal 0 (gate is opened). In case of equal the 2 minute countdown will be started (by timer). After elapsing the given time will be again generated impulse on data point **GateIm** for 1 second.

Program

```
// file containing device system constants
#INCLUDE <system.spi>

#PARAM LogSize 16384

#CONST Opened = 0
#CONST Closed = 1

// constants for marking the data points in configuration
#CONST GateImpulse = "GateIm"
#CONST GateState = "GateSt"

// constants for timers
```

```
#CONST GateImpulseWait = 0
#CONST GateOpened = 1

// timeout for how long the gate have to be opened (in sec)
#CONST GateOpenedTimeout = 10

#DEBUG ON
// servicing event when calling and verifying user number
#ACTION User Action_USR Action_USR_PHONEOK
    // set impulse
      MOV FLOAT DP[GateImpulse], 1
      // start timer GateImpulseWait
      MOV CTC[GateImpulseWait], 1
#END

// service of timer GateImpulseWait
#ACTION TimerGateOpen Action_CTC GateImpulseWait
      // drop impulse
      MOV FLOAT DP[GateImpulse], 0
#END

// service of data point new value event
#ACTION GateStateChanged Action_DP_SENSE 200
    // if the gate is opened
      EQ FLOAT DP[GateState], Opened
            // start 2 minutes countdown
            MOV CTC[GateOpened], GateOpenedTimeout
      // end of condition (not necessary)
      ENDIF
#END

// service of casovaca GateOpened
#ACTION TimerGateOpened Action_CTC GateOpened
    // if the gate is opened
      EQ FLOAT DP[GateState], Opened
            // set impulse
            MOV FLOAT DP[GateImpulse], 1
            // start timer GateImpulseWait
            MOV CTC[GateImpulseWait], 1
      // end of condition (not necessary)
      ENDIF
#END
```

Notes

Whole example with complete configuration is available on istallation CD for application **K2config** in directory **Examples\Ex3**.