1. Application description

This application presents an example on how to restore drive operation after a fault state.

Default behavior in case of an error:

- Switch off green LED
- Switch on red LED
- Cutoff drive PWM outputs and disable the controllers
- Terminate the execution of the TML program.

Proposed recovery sequence:

- Switch off the green LED
- Switch on the red LED
- Cutoff drive PWM outputs
- Wait until the error condition disappears
- Program the motor to hold its current position
- Re-activate the drive PWM outputs
- Switch off the red LED and switch on the green LED
- Resume the main program and process the commands received from the master.

The application uses the software protection interrupt, which monitors the following software protections:

- Over current;
- Over temperature – drive (where a temperature sensor is available);
- Over voltage;
- Under voltage;

An easiest way to evaluate this application is to trigger an over-voltage. This can be done by decreasing the over-voltage protection threshold value under the supply voltage value. The parameter that stores the over voltage protection threshold value is called “UMAXPROT”. It can be modified online from the Command Interpreter window in EasyMotion Studio. For details, see the chapter 4.3.

2. Application flow chart

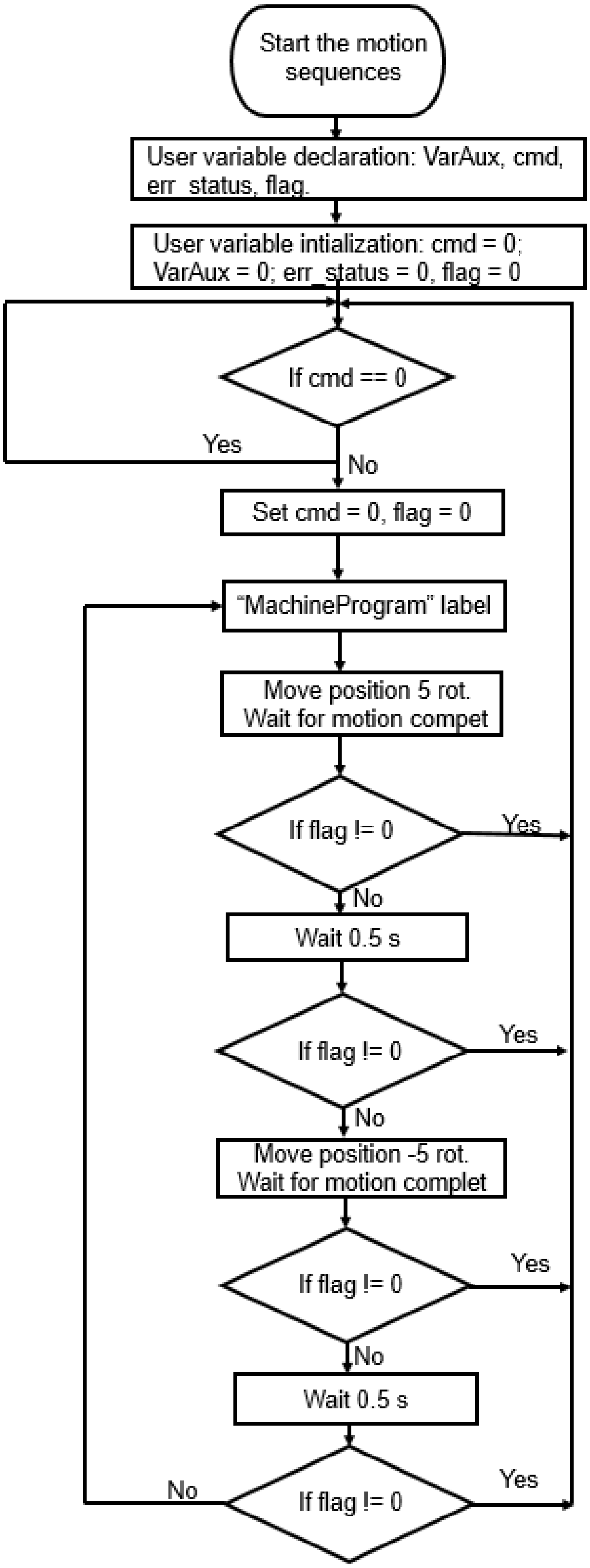

Start the motion sequences
User variable declaration: VarAux, cmd, err_status, flag.
User variable intialization: cmd = 0; VarAux = 0; err_status = 0, flag = 0
If cmd == 0
Yes
No
Set cmd = 0, flag = 0
"MachineProgram" label
Move position 5 rot. Wait for motion compet
If flag != 0
Yes
No
Wait 0.5 s
If flag != 0
Yes
No
Move position -5 rot. Wait for motion complet
If flag != 0
Yes
No
Wait 0.5 s
No
If flag != 0
Yes

Figure 1. *Main application structure*

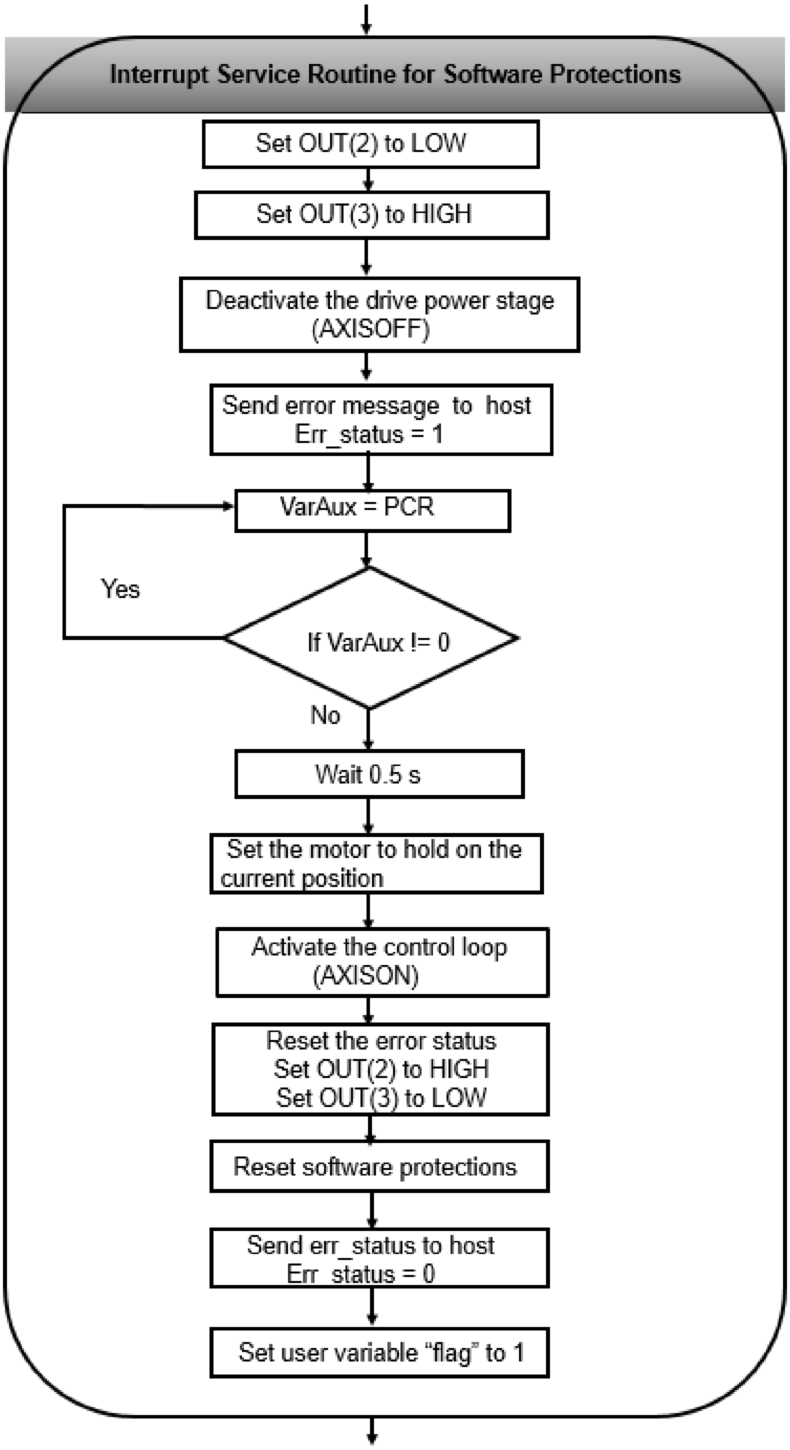
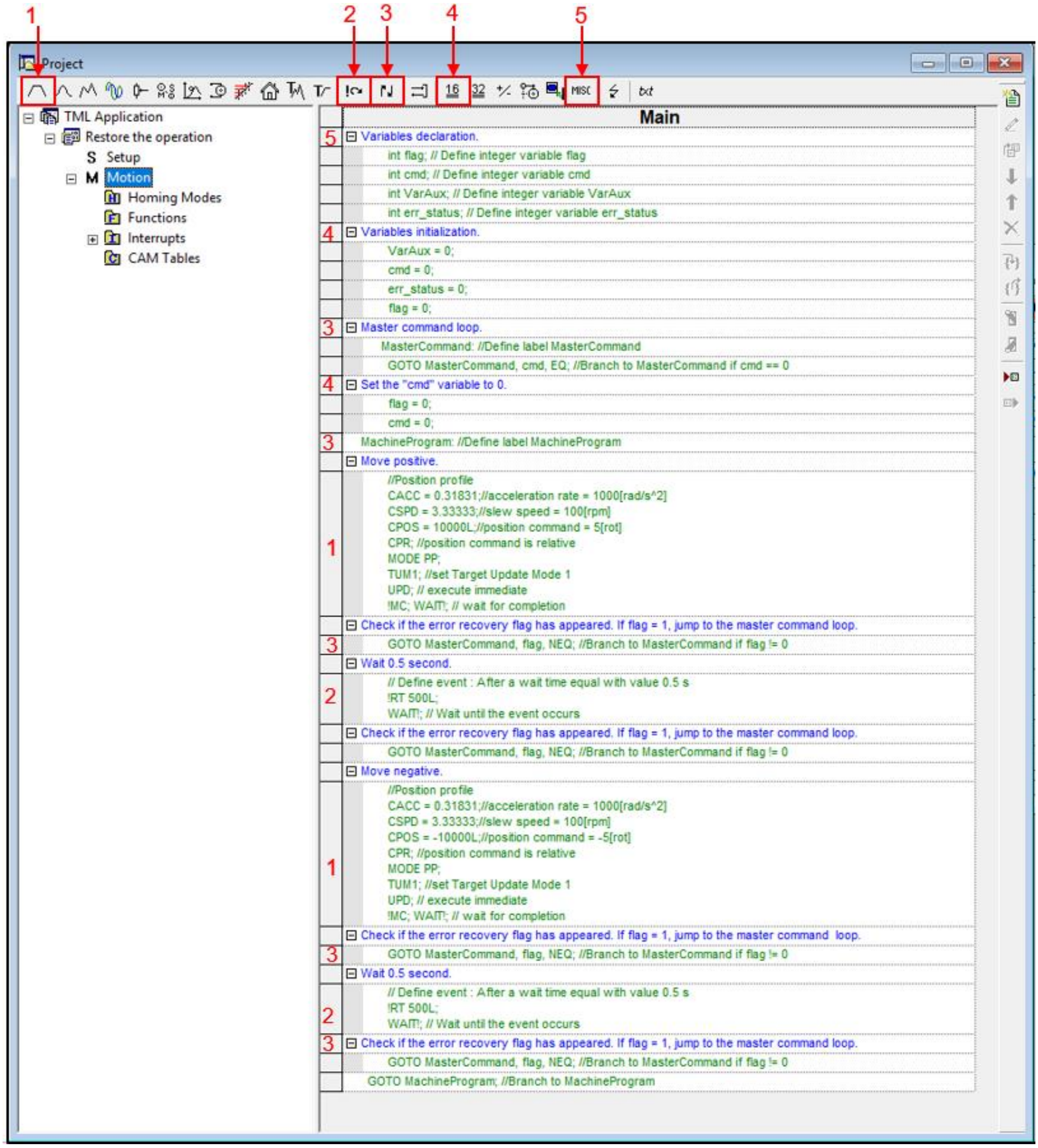

Interrupt Service Routine for Software Protections
Set OUT(2) to LOW
Set OUT(3) to HIGH
Deactivate the drive power stage (AXISOFF)
Send error message to host Err_status = 1
VarAux = PCR
Yes
If VarAux != 0
No
Wait 0.5 s
Set the motor to hold on the current position
Activate the control loop (AXISON)
Reset the error status
Set OUT(2) to HIGH
Set OUT(3) to LOW
Reset software protections
Send err_status to host
Err_status = 0
Set user variable "flag" to 1

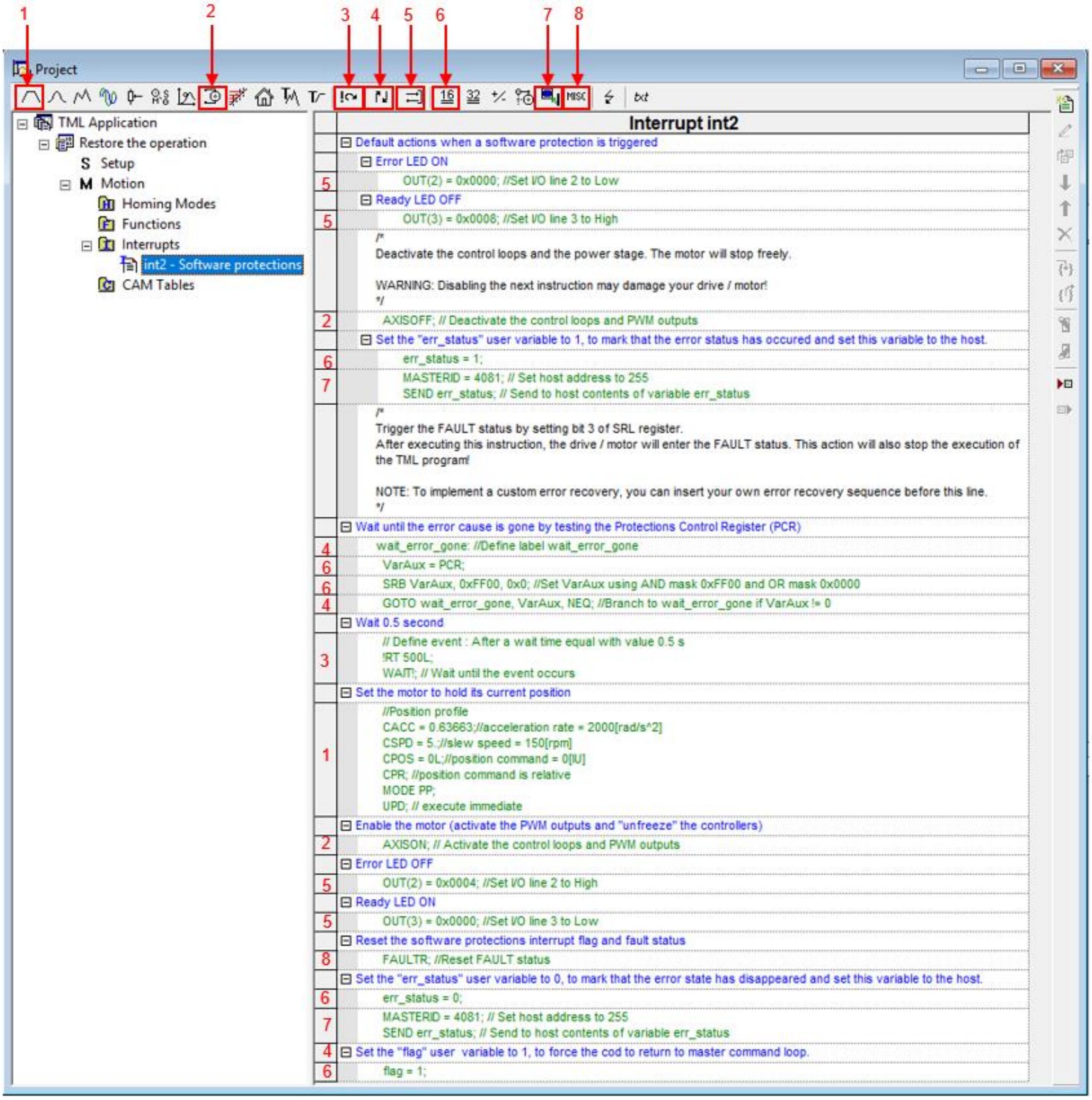
Figure 2. *Software protections interrupt routine structure*

3. EasyMotion Studio implementation



```
Variables declaration.
    int flag; // Define integer variable flag
    int cmd; // Define integer variable cmd
    int VarAux; // Define integer variable VarAux
    int err_status; // Define integer variable err_status
Variables initialization.
    VarAux = 0;
    cmd = 0;
    err_status = 0;
    flag = 0;
Master command loop.
   MasterCommand: //Define label MasterCommand
    GOTO MasterCommand, cmd, EQ; //Branch to MasterCommand if cmd == 0
Set the "cmd" variable to 0.
    flag = 0;
    cmd = 0;
MachineProgram: //Define label MachineProgram
Move positive.
    //Position profile
    CACC = 0.31831;//acceleration rate = 1000[rad/s^2]
    CSPD = 3.33333;//slew speed = 100[rpm]
    CPOS = 10000L;//position command = 5[rot]
    CPR; //position command is relative
    MODE PP;
    TUM1; //set Target Update Mode 1
    UPD; // execute immediate
    !MC; WAIT!; // wait for completion
Check if the error recovery flag has appeared. If flag = 1, jump to the master command loop.
    GOTO MasterCommand, flag, NEQ; //Branch to MasterCommand if flag != 0
Wait 0.5 second.
    // Define event : After a wait time equal with value 0.5 s
    !RT 500L;
    WAIT!; // Wait until the event occurs
Check if the error recovery flag has appeared. If flag = 1, jump to the master command loop.
    GOTO MasterCommand, flag, NEQ; //Branch to MasterCommand if flag != 0
Move negative.
    //Position profile
    CACC = 0.31831;//acceleration rate = 1000[rad/s^2]
    CSPD = 3.33333;//slew speed = 100[rpm]
    CPOS = -10000L;//position command = -5[rot]
    CPR; //position command is relative
    MODE PP;
    TUM1; //set Target Update Mode 1
    UPD; // execute immediate
    !MC; WAIT!; // wait for completion
Check if the error recovery flag has appeared. If flag = 1, jump to the master command  loop.
    GOTO MasterCommand, flag, NEQ; //Branch to MasterCommand if flag != 0
Wait 0.5 second.
    // Define event : After a wait time equal with value 0.5 s
    !RT 500L;
    WAIT!; // Wait until the event occurs
Check if the error recovery flag has appeared. If flag = 1, jump to the master command loop.
    GOTO MasterCommand, flag, NEQ; //Branch to MasterCommand if flag != 0
  GOTO MachineProgram; //Branch to MachineProgram
```

Figure 3. *Main section of the TML program*



```
   ⊟ Default actions when a software protection is triggered
     ⊟ Error LED ON
5          OUT(2) = 0x0000; //Set I/O line 2 to Low
     ⊟ Ready LED OFF
5          OUT(3) = 0x0008; //Set I/O line 3 to High
       /*
       Deactivate the control loops and the power stage. The motor will stop freely.

       WARNING: Disabling the next instruction may damage your drive / motor!
       */
2       AXISOFF; // Deactivate the control loops and PWM outputs
     ⊟ Set the "err_status" user variable to 1, to mark that the error status has occured and set this variable to the host.
6          err_status = 1;
7          MASTERID = 4081; // Set host address to 255
           SEND err_status; // Send to host contents of variable err_status
       /*
       Trigger the FAULT status by setting bit 3 of SRL register.
       After executing this instruction, the drive / motor will enter the FAULT status. This action will also stop the execution of
       the TML program!

       NOTE: To implement a custom error recovery, you can insert your own error recovery sequence before this line.
       */
   ⊟ Wait until the error cause is gone by testing the Protections Control Register (PCR)
4      wait_error_gone: //Define label wait_error_gone
6       VarAux = PCR;
6       SRB VarAux, 0xFF00, 0x0; //Set VarAux using AND mask 0xFF00 and OR mask 0x0000
4       GOTO wait_error_gone, VarAux, NEQ; //Branch to wait_error_gone if VarAux != 0
   ⊟ Wait 0.5 second
        // Define event : After a wait time equal with value 0.5 s
3       !RT 500L;
        WAIT!; // Wait until the event occurs
   ⊟ Set the motor to hold its current position
        //Position profile
        CACC = 0.63663;//acceleration rate = 2000[rad/s^2]
        CSPD = 5.;//slew speed = 150[rpm]
1       CPOS = 0L;//position command = 0[IU]
        CPR; //position command is relative
        MODE PP;
        UPD; // execute immediate
   ⊟ Enable the motor (activate the PWM outputs and "unfreeze" the controllers)
2       AXISON; // Activate the control loops and PWM outputs
   ⊟ Error LED OFF
5       OUT(2) = 0x0004; //Set I/O line 2 to High
   ⊟ Ready LED ON
5       OUT(3) = 0x0000; //Set I/O line 3 to Low
   ⊟ Reset the software protections interrupt flag and fault status
8       FAULTR; //Reset FAULT status
   ⊟ Set the "err_status" user variable to 0, to mark that the error state has disappeared and set this variable to the host.
6       err_status = 0;
7       MASTERID = 4081; // Set host address to 255
        SEND err_status; // Send to host contents of variable err_status
4  ⊟ Set the "flag" user  variable to 1, to force the cod to return to master command loop.
6       flag = 1;
```

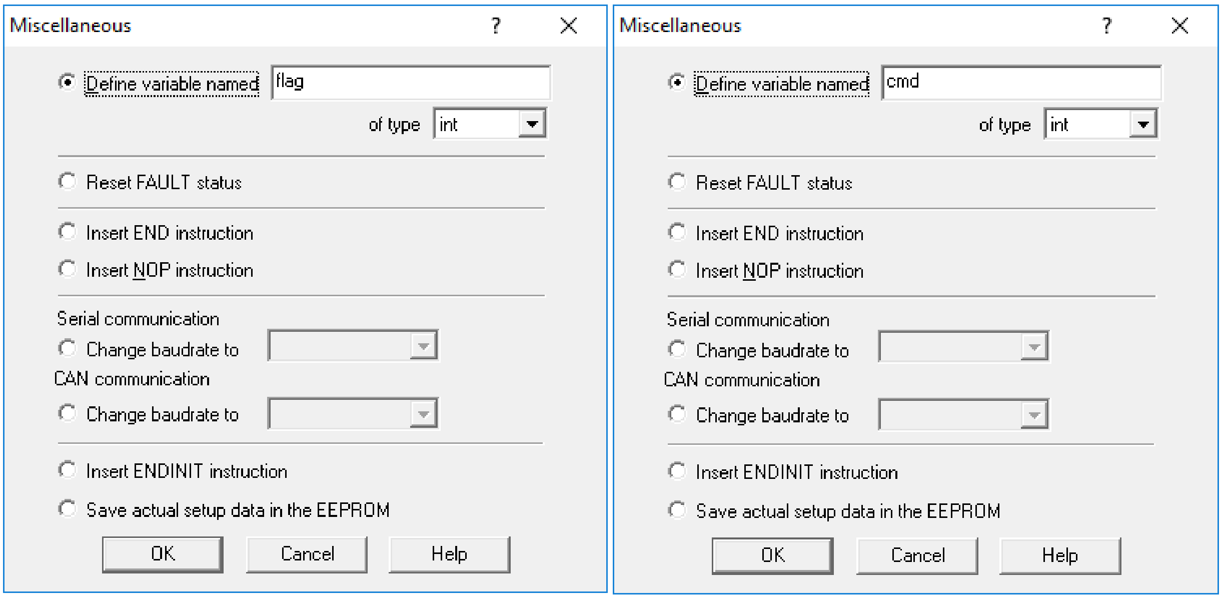
Figure 4. *Software protections interrupt service routine*

4. Detailed description of the EasyMotion Studio implementation

4.1 Motion section

The code sequences from the “Motion” section were generated using the buttons marked with 1 to 5 in Figure 2. Clicking on those buttons the following programming dialogs will open.

• The “Miscellaneous” dialogue (5) allows defining user variables (integer - 16 bit, fixed - 32 bit or long -32 bit).



Miscellaneous

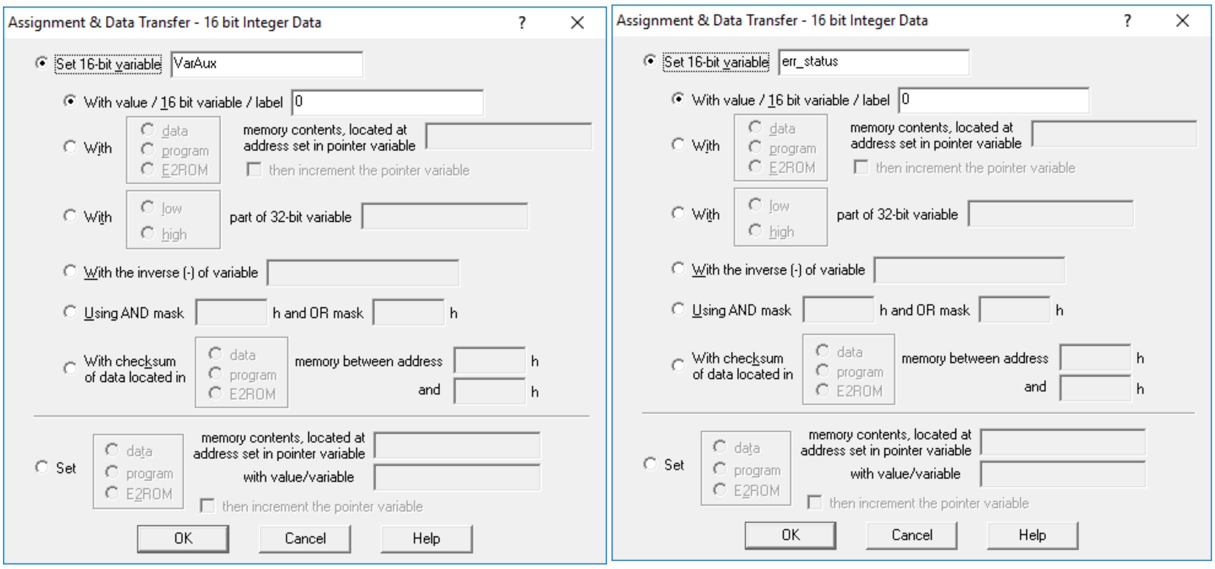
● Define variable named: flag
of type: int
○ Reset FAULT status
○ Insert END instruction
○ Insert NOP instruction
Serial communication
○ Change baudrate to
CAN communication
○ Change baudrate to
○ Insert ENDINIT instruction
○ Save actual setup data in the EEPROM

Miscellaneous

● Define variable named: cmd
of type: int
○ Reset FAULT status
○ Insert END instruction
○ Insert NOP instruction
Serial communication
○ Change baudrate to
CAN communication
○ Change baudrate to
○ Insert ENDINIT instruction
○ Save actual setup data in the EEPROM

Figure 5. *How to define variables*

• The “Assignment and Data Transfer – 16 bit Integer Data” dialogue (6) helps to assign a value to a 16-bit integer variable. This way a variable can be initialized or its value can be modified.



Assignment & Data Transfer - 16 bit Integer Data

● Set 16-bit variable: VarAux
● With value / 16 bit variable / label: 0
○ With ○ data ○ program ○ E2ROM memory contents, located at address set in pointer variable
□ then increment the pointer variable
○ With ○ low ○ high part of 32-bit variable
○ With the inverse (-) of variable
○ Using AND mask h and OR mask h
○ With checksum of data located in ○ data ○ program ○ E2ROM memory between address h and h
○ Set ○ data ○ program ○ E2ROM memory contents, located at address set in pointer variable with value/variable
□ then increment the pointer variable

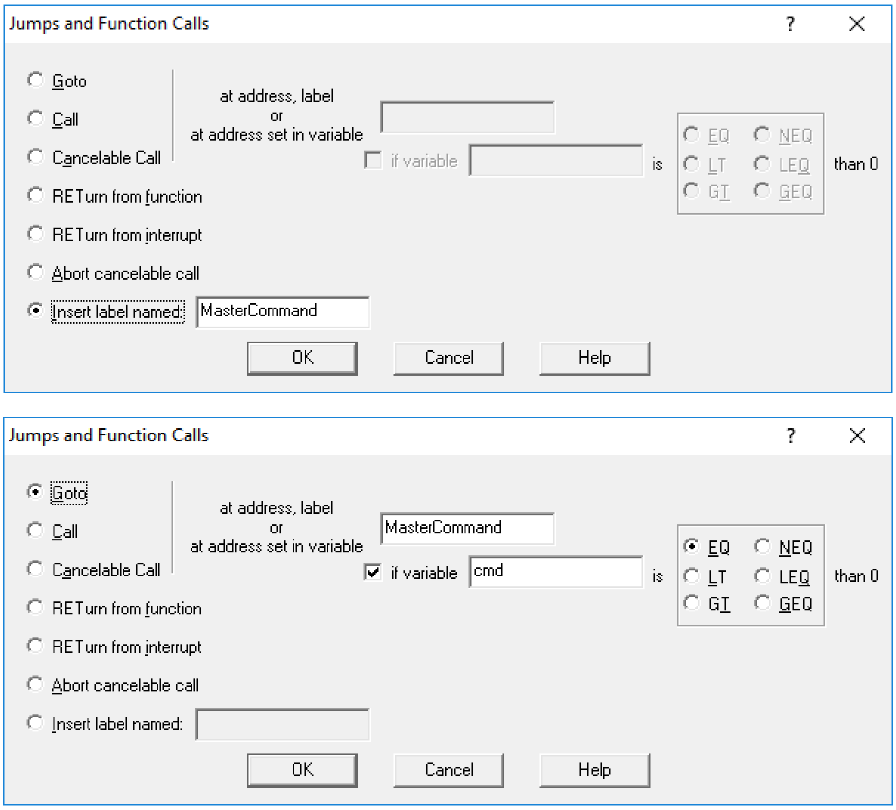
Assignment & Data Transfer - 16 bit Integer Data

● Set 16-bit variable: err_status
● With value / 16 bit variable / label: 0
○ With ○ data ○ program ○ E2ROM memory contents, located at address set in pointer variable
□ then increment the pointer variable
○ With ○ low ○ high part of 32-bit variable
○ With the inverse (-) of variable
○ Using AND mask h and OR mask h
○ With checksum of data located in ○ data ○ program ○ E2ROM memory between address h and h
○ Set ○ data ○ program ○ E2ROM memory contents, located at address set in pointer variable with value/variable
□ then increment the pointer variable

Figure 6. *How to set a value for a variable*

• The “Jumps and Function Calls” dialogue (4) allows controlling the TML program flow through unconditional or conditional jumps and unconditional, conditional or cancelable calls of TML functions.

In this case this dialog was used to create the “MasterCommand” loop where the program waits for the master to start the motion sequence, by setting the “cmd” user variable to a value different than 0.



Jumps and Function Calls

○ Goto
○ Call
○ Cancelable Call
○ RETurn from function
○ RETurn from interrupt
○ Abort cancelable call
● Insert label named: MasterCommand

at address, label or at address set in variable:
□ if variable is ○ EQ ○ NEQ ○ LT ○ LEQ ○ GT ○ GEQ than 0

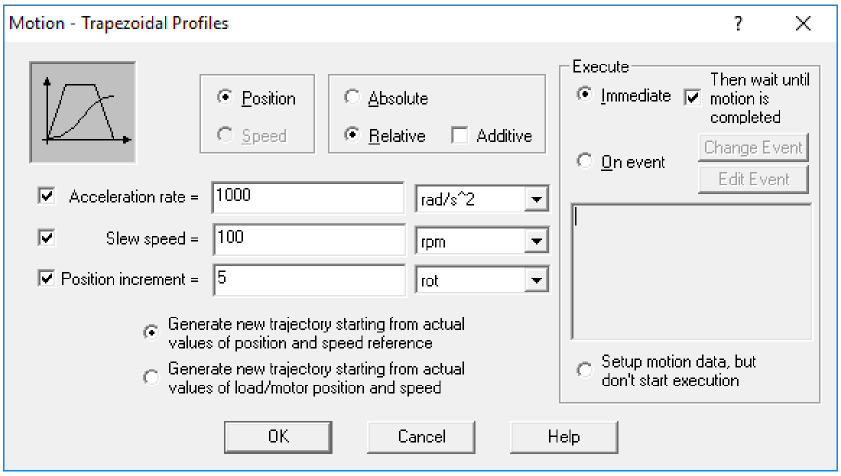
Jumps and Function Calls

● Goto
○ Call
○ Cancelable Call
○ RETurn from function
○ RETurn from interrupt
○ Abort cancelable call
○ Insert label named:

at address, label or at address set in variable: MasterCommand
☑ if variable cmd is ● EQ ○ NEQ ○ LT ○ LEQ ○ GT ○ GEQ than 0

Figure 7. *How to implement a TML loop*

• The “Motion – Trapezoidal Profiles” dialogue (1) allows to program a position or speed profile with a trapezoidal shape of the speed, due to a limited acceleration.



Motion - Trapezoidal Profiles

● Position ○ Speed
○ Absolute ● Relative □ Additive

☑ Acceleration rate = 1000 rad/s^2
☑ Slew speed = 100 rpm
☑ Position increment = 5 rot

● Generate new trajectory starting from actual values of position and speed reference
○ Generate new trajectory starting from actual values of load/motor position and speed

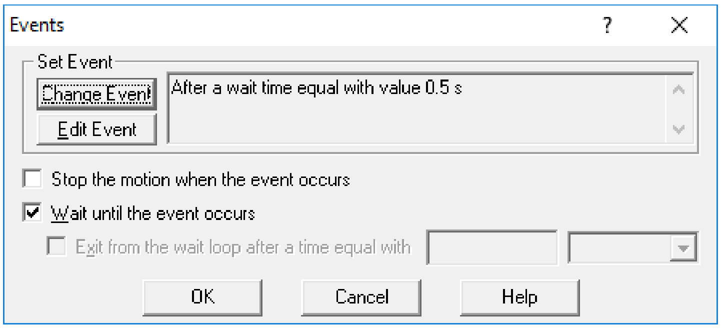
Execute
● Immediate ☑ Then wait until motion is completed
○ On event
○ Setup motion data, but don't start execution

Figure 8. *How to set a motion profile*

This dialog was used to create the motion profiles inside the “MachineProgram” loop.

• The “Events” dialogue (2) allows to define events. An event is a programmable condition, which once set, is monitored for occurrence.

The following actions can be connected to an event:
- stop the motion when the event occurs;
- wait for the programmed event to occur.



After a wait time equal with value 0.5 s

☐ Stop the motion when the event occurs

☑ Wait until the event occurs

☐ Exit from the wait loop after a time equal with

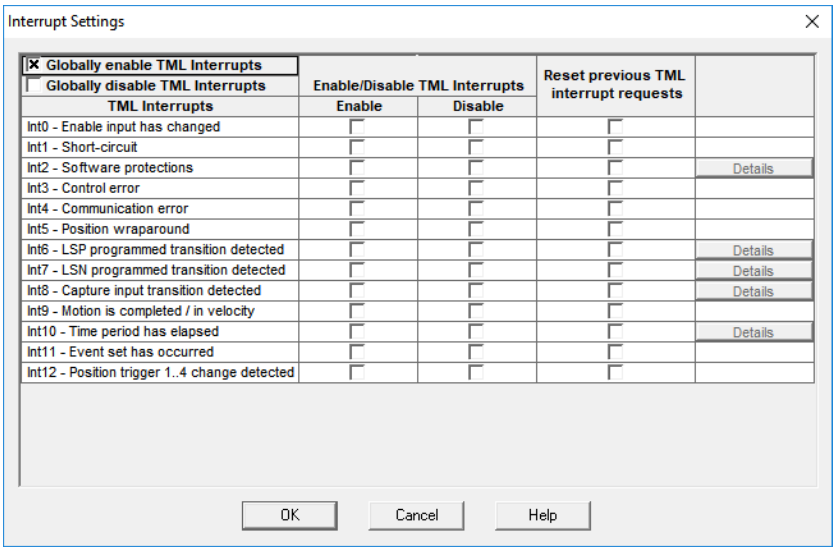
Figure 9. *How to create an event*

The “Events” dialog is used to insert a delay of 0.5 s between the two motion profiles.

4.2 Software Protections Interrupt

The TML interrupts are special functions that are continuously monitored by the drive firmware. When a TML interrupt occurs, the main TML program execution is suspended and the TML code associated with the interrupt, called Interrupt Service Routine (in short ISR), is executed.

Remark: While an interrupt is active, the other interrupts are deactivated. It is recommended to keep the ISR as short as possible. If is not possible, then the other interrupts should be re-enabled using the “Interrupts Settings” dialogue.



☒ Globally enable TML Interrupts ☐ Globally disable TML Interrupts	Enable/Disable TML Interrupts		Reset previous TML interrupt requests	
TML Interrupts	Enable	Disable		
Int0 - Enable input has changed	☐	☐	☐	
Int1 - Short-circuit	☐	☐	☐	
Int2 - Software protections	☐	☐	☐	Details
Int3 - Control error	☐	☐	☐	
Int4 - Communication error	☐	☐	☐	
Int5 - Position wraparound	☐	☐	☐	
Int6 - LSP programmed transition detected	☐	☐	☐	Details
Int7 - LSN programmed transition detected	☐	☐	☐	Details
Int8 - Capture input transition detected	☐	☐	☐	Details
Int9 - Motion is completed / in velocity	☐	☐	☐	
Int10 - Time period has elapsed	☐	☐	☐	Details
Int11 - Event set has occurred	☐	☐	☐	
Int12 - Position trigger 1..4 change detected	☐	☐	☐	

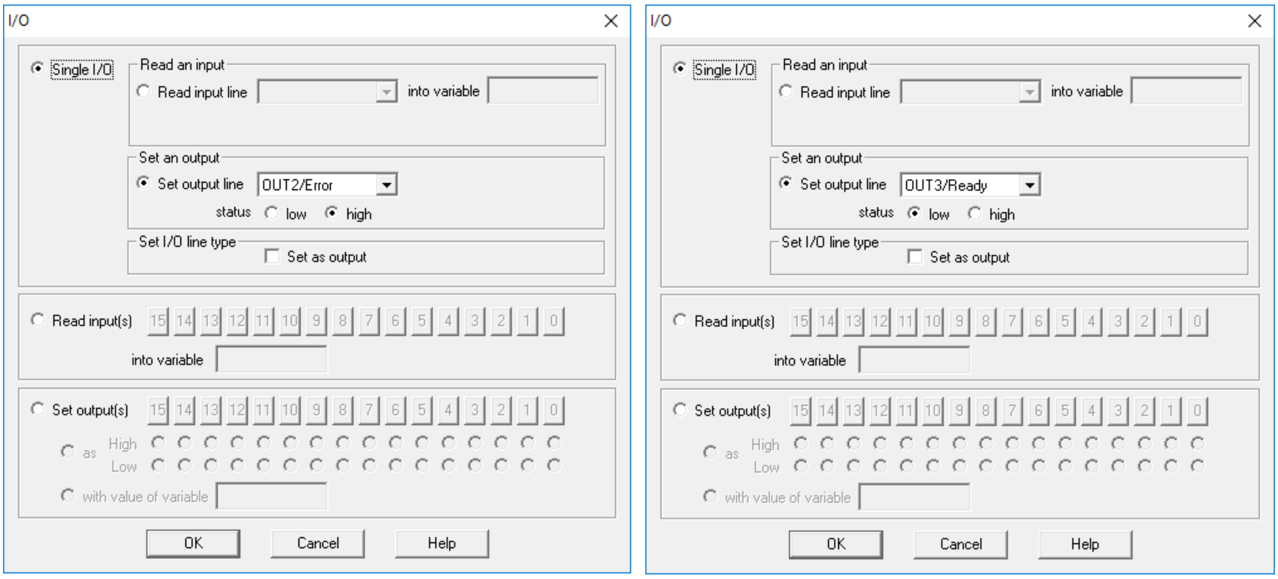
Figure 10. *Interrupt Settings dialog*

This application was implemented using the “Int2 - Software protections” interrupt, that was customized to contain the code that corresponds to the recovery sequence proposed in the first chapter of this document.

The code sequence inside the “Int2 - Software protections” interrupt was generated using the buttons marked with 1 to 8 in Figure 3. Clicking on those buttons the following programming dialogs will open.

- The “I/O” dialogue allows programming the following operations with the digital inputs and outputs:
    - read and save the status of a digital input into a variable
    - set low or high a digital output
    - read and save the status of multiple digital inputs into a variable
    - set multiple digital outputs according with an immediate value or the value of 16-bit variable

In this application, the “I/O” dialogue (1) is used to set LOW / HIGH the “Ready” and “Error” digital output, that are also associated to the green and respective red LED on the drive.



I/O

● Single I/O

Set an output: ● Set output line OUT2/Error, status ○ low ● high

Set I/O line type: □ Set as output

I/O

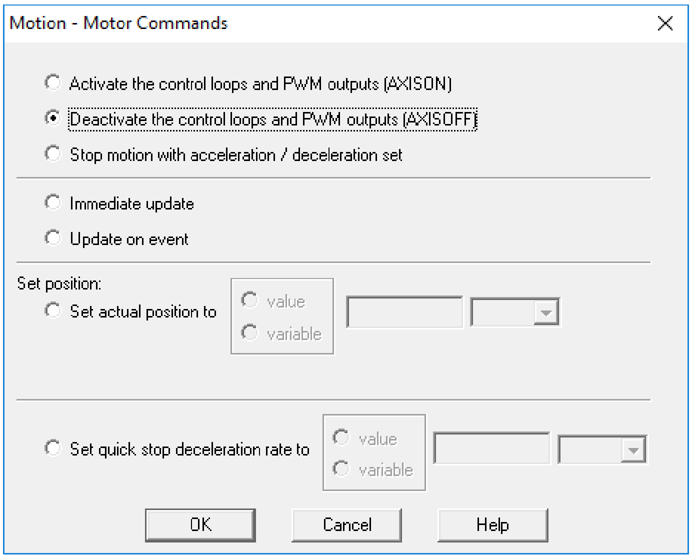
● Single I/O

Set an output: ● Set output line OUT3/Ready, status ● low ○ high

Set I/O line type: □ Set as output

Figure 11. *How to set output line status*

- The “Motion - Motor Commands” dialogue (2) was used to deactivate the control loops and the power stage PWM output commands (AXISOFF) when the error has occurred.



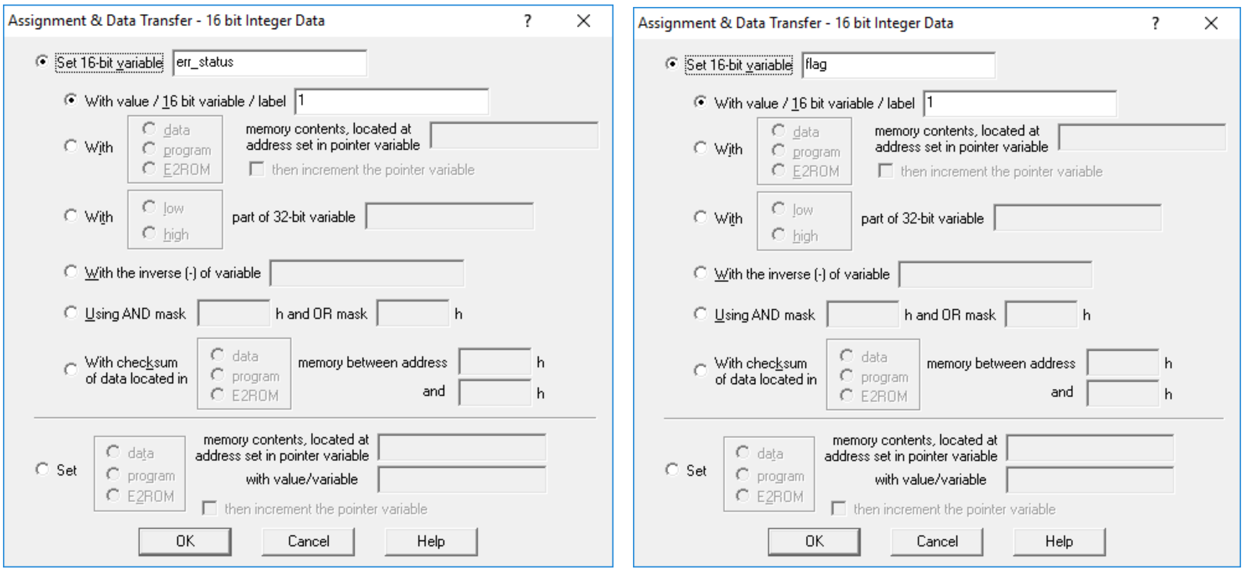
Motion - Motor Commands

○ Activate the control loops and PWM outputs (AXISON)
● Deactivate the control loops and PWM outputs (AXISOFF)
○ Stop motion with acceleration / deceleration set

○ Immediate update
○ Update on event

Figure 12. *How to deactivate the drive power stage*

• The “Assignment and Data Transfer – 16 bit Integer Data” dialogue (6) was used to modify the value of the “err_status” and “flag” user variables.



Assignment & Data Transfer - 16 bit Integer Data

● Set 16-bit variable: err_status

● With value / 16 bit variable / label: 1

Assignment & Data Transfer - 16 bit Integer Data

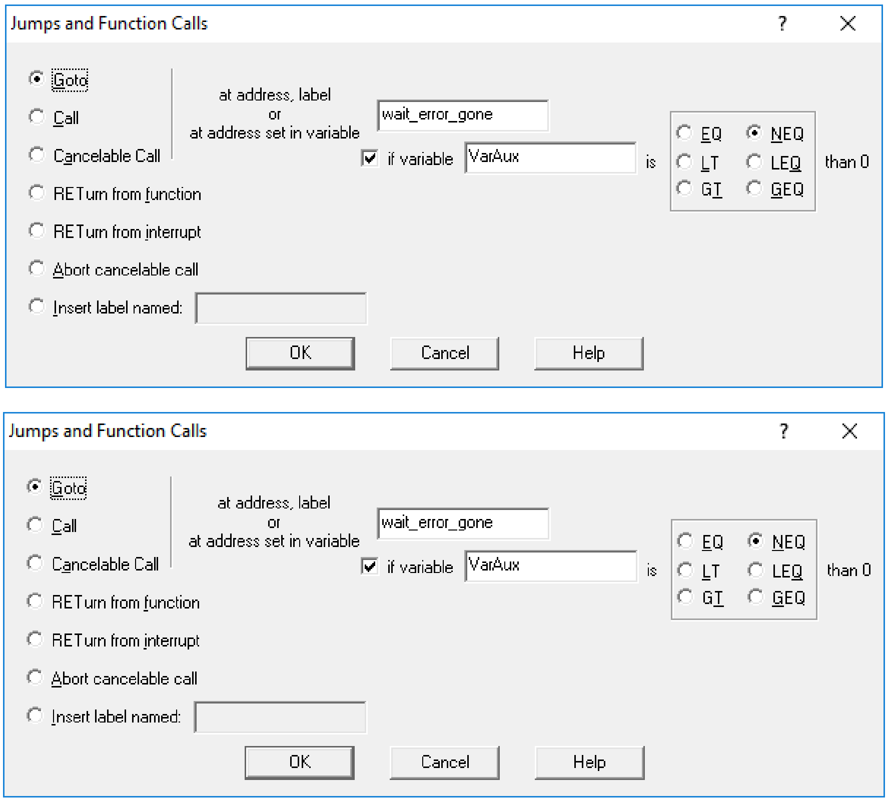
● Set 16-bit variable: flag

● With value / 16 bit variable / label: 1

?
With
data
program
E2ROM
memory contents, located at address set in pointer variable
then increment the pointer variable
low
high
part of 32-bit variable
With the inverse (-) of variable
Using AND mask
h and OR mask
h
With checksum of data located in
memory between address
and
Set
with value/variable
OK
Cancel
Help

Figure 13. *How to set a value for a variable*

• The “Jumps and Function Calls” dialogue (4) was used to create the “wait_error_gone” loop, where the program will stay as long as the “VarAux” variable will be different than 0.



Jumps and Function Calls

● Goto

at address, label or at address set in variable: wait_error_gone

☑ if variable VarAux is ● NEQ than 0

Jumps and Function Calls

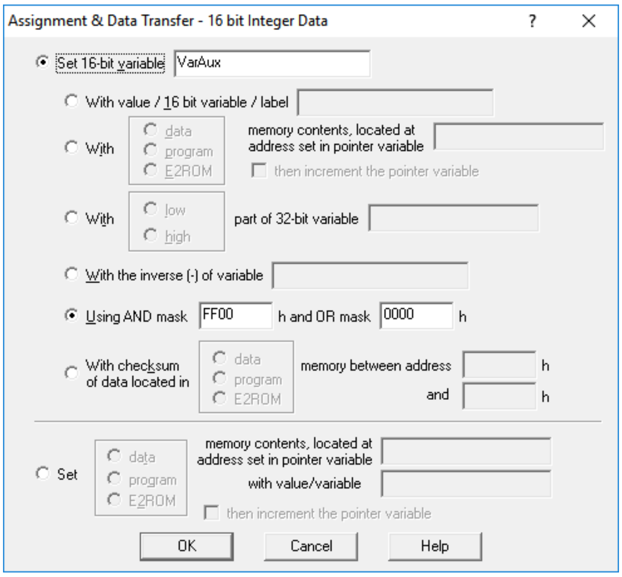
● Goto

at address, label or at address set in variable: wait_error_gone

☑ if variable VarAux is ● NEQ than 0

Figure 14. *How to implement a TML loop*

• Inside the "wait_error_gone" loop the "Assignment and Data Transfer – 16-bit Integer Data" dialogue (6) was used, to apply an "AND" and "OR" mask to the "VarAux" user variable (copy of the PCR register). The mask isolates bits 15 to 8 in the "VarAux" variable, to check if the error state has disappeared.



Assignment & Data Transfer - 16 bit Integer Data

● Set 16-bit variable VarAux

○ With value / 16 bit variable / label

○ With ○ data ○ program ○ E2ROM memory contents, located at address set in pointer variable □ then increment the pointer variable

○ With ○ low ○ high part of 32-bit variable

○ With the inverse (-) of variable

● Using AND mask FF00 h and OR mask 0000 h

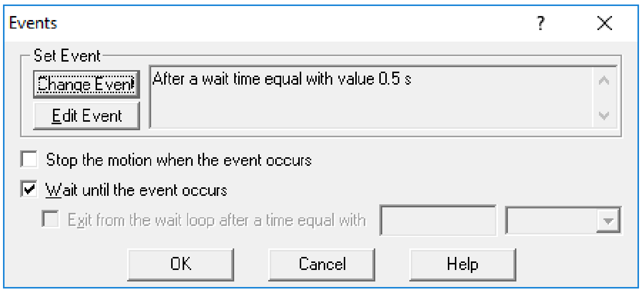
○ With checksum of data located in ○ data ○ program ○ E2ROM memory between address h and h

○ Set ○ data ○ program ○ E2ROM memory contents, located at address set in pointer variable with value/variable □ then increment the pointer variable

Figure 15. *How to apply an AND and OR mask to a 16-bits variable*

Remark: PCR (Protections Control Register) is a 16-bit command and status register, containing the status information of the TML protections. A detailed description of the PCR register in presented in the EasyMotion Studio help topics (link).

• The "Events" dialogue (3) was used to hold the program execution for 0.5 s.



Events

Set Event

After a wait time equal with value 0.5 s

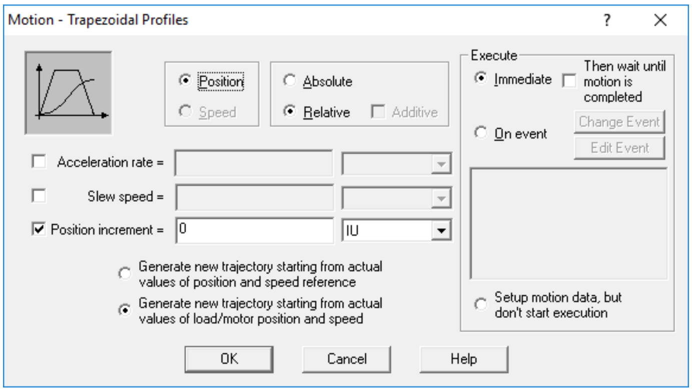
□ Stop the motion when the event occurs

☑ Wait until the event occurs

□ Exit from the wait loop after a time equal with

Figure 16. *How to set a time event*

• The "Motion – Trapezoidal Profiles" dialogue (1) was used this time to force the motor to hold the current position, after the error state disappears and the axis is re-enabled.



Motion - Trapezoidal Profiles

● Position ○ Speed

○ Absolute ● Relative □ Additive

Execute

● Immediate □ Then wait until motion is completed

○ On event

□ Acceleration rate =

□ Slew speed =

☑ Position increment = 0 IU

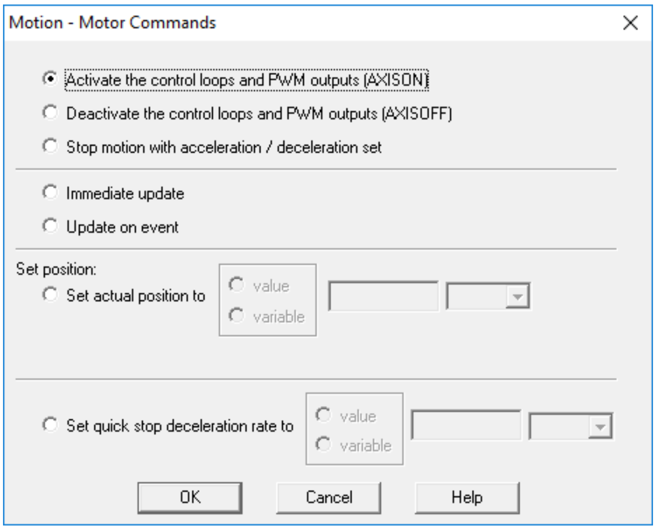
○ Generate new trajectory starting from actual values of position and speed reference

● Generate new trajectory starting from actual values of load/motor position and speed

○ Setup motion data, but don't start execution

Figure 17. *How to configure and start a position profile with CPOS = 0, using the TUM0 mode*

- The “Motion - Motor Commands” dialogue (2) is used again to reactivate the drive PWM outputs and allow the position profile above to start being executed.



Motion - Motor Commands

● Activate the control loops and PWM outputs (AXISON)
○ Deactivate the control loops and PWM outputs (AXISOFF)
○ Stop motion with acceleration / deceleration set

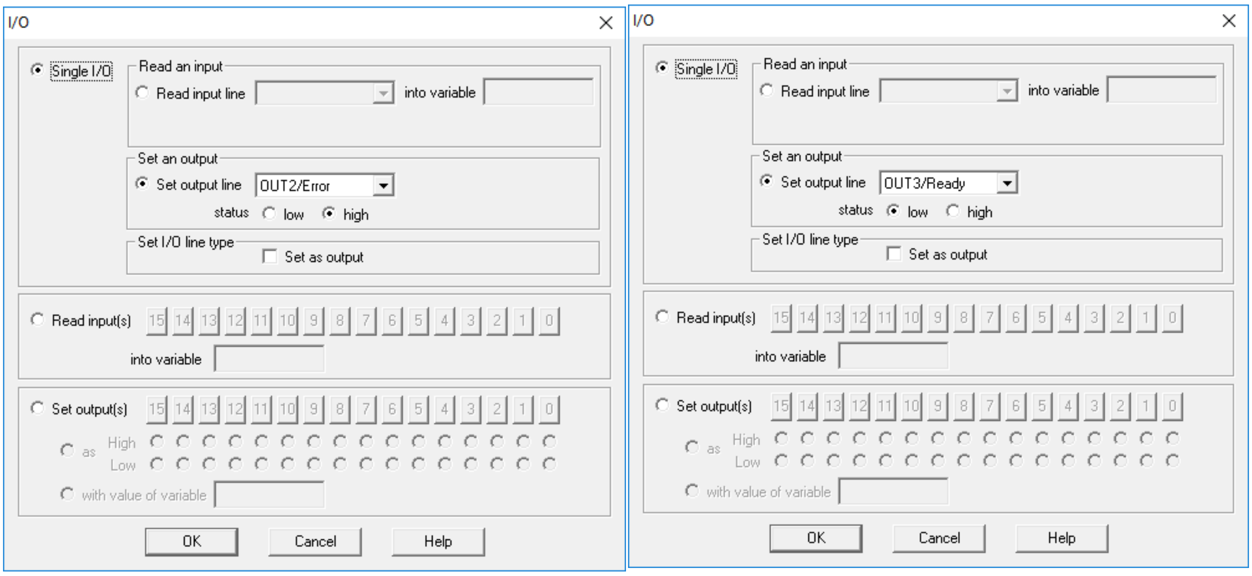
○ Immediate update
○ Update on event

Set position:
○ Set actual position to ○ value ○ variable

○ Set quick stop deceleration rate to ○ value ○ variable

Figure 18. *How to activate the control loop*

- The green LED (Ready – OUT3) is turned on and the red LED (Error – OUT2) is switched OFF, using the “I/O” dialogue.



I/O

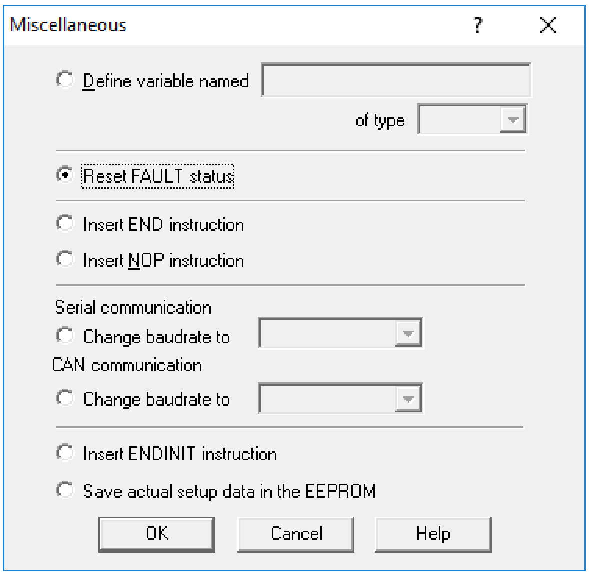
● Single I/O
Read an input
○ Read input line into variable
Set an output
● Set output line OUT2/Error
status ○ low ● high
Set I/O line type
□ Set as output
○ Read input(s) 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
into variable
○ Set output(s) 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
○ as High / Low
○ with value of variable

I/O

● Single I/O
Read an input
○ Read input line into variable
Set an output
● Set output line OUT3/Ready
status ● low ○ high
Set I/O line type
□ Set as output
○ Read input(s) 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
into variable
○ Set output(s) 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
○ as High / Low
○ with value of variable

Figure 19. *How to reset the output line status*

- The fault status is reset using the “**FAULTR;**” TML instruction (see the “Miscellaneous” (8) dialog).



Miscellaneous

○ Define variable named

of type

● Reset FAULT status

○ Insert END instruction

○ Insert NOP instruction

Serial communication

○ Change baudrate to

CAN communication

○ Change baudrate to

○ Insert ENDINIT instruction

○ Save actual setup data in the EEPROM

Figure 20. *How to reset fault status*

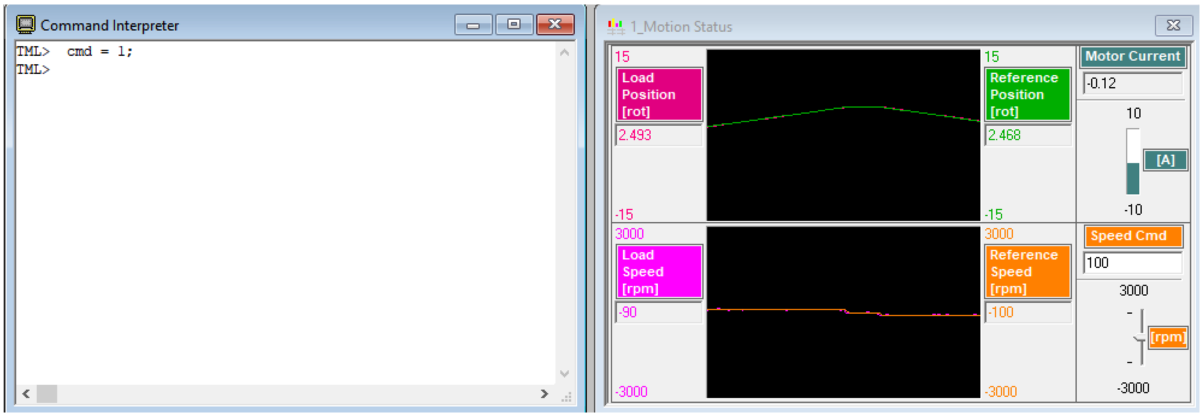
Remark*:* The drive/motor will return to FAULT state if there are errors when the FAULTR command is executed

• Additionally, the drive can be configured to send automatically the error status, to a host. For that the "Send Data To Host" dialog (7) can be used.

4.3 Application evaluation, using the over-voltage protection

After the application runs, the drive will wait for the master to send the start command. To simulate the master, the "Command Interpreter" window can be used, to set the "cmd" variable to a value different than 0.

The motor will start to perform the motion profiles inside the "MachineProgram" loop. This can be checked, using the "1_Motion Status" control panel.



Command Interpreter

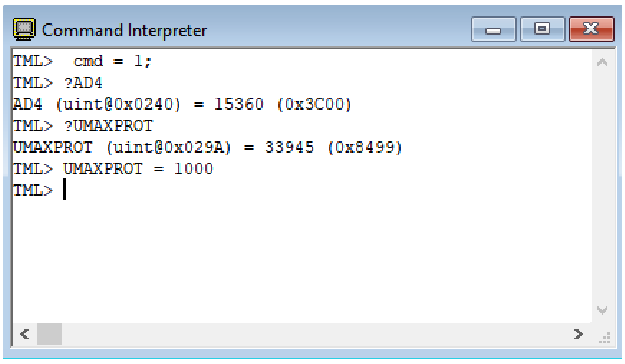
```
TML>  cmd = 1;
TML>
```

1_Motion Status

Figure 21. *Start the motion when "cmd" become different than 0*

The over-voltage protection is continuously monitoring the motor supply voltage (AD4). When this value goes over the set threshold (UMAXPORT), the protection triggers and the drive executes the TML code inside the "Int 2. Software protections" interrupt routine.

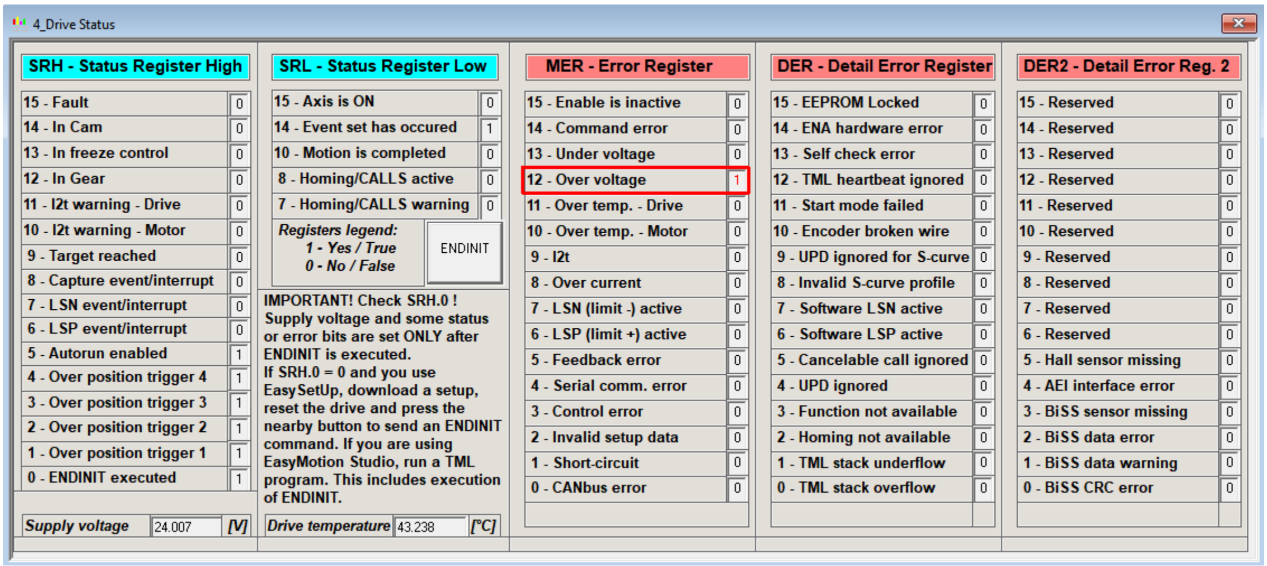
To simulate an over-voltage situation, "UMAXPROT" should be set lower than AD4.



```
TML>  cmd = 1;
TML> ?AD4
AD4 (uint@0x0240) = 15360 (0x3C00)
TML> ?UMAXPROT
UMAXPROT (uint@0x029A) = 33945 (0x8499)
TML> UMAXPROT = 1000
TML> |
```

Figure 22. *Trigger the over-voltage protection*

The bit 12 in the MER error register (over-voltage) will be triggered and the drive will execute the code in the software protections interrupt.



SRH - Status Register High	
15 - Fault	0
14 - In Cam	0
13 - In freeze control	0
12 - In Gear	0
11 - I2t warning - Drive	0
10 - I2t warning - Motor	0
9 - Target reached	0
8 - Capture event/interrupt	0
7 - LSN event/interrupt	0
6 - LSP event/interrupt	0
5 - Autorun enabled	1
4 - Over position trigger 4	1
3 - Over position trigger 3	1
2 - Over position trigger 2	1
1 - Over position trigger 1	1
0 - ENDINIT executed	1

Supply voltage 24.007 *[V]*

SRL - Status Register Low	
15 - Axis is ON	0
14 - Event set has occured	1
10 - Motion is completed	0
8 - Homing/CALLS active	0
7 - Homing/CALLS warning	0

Registers legend:
1 - Yes / True
0 - No / False

IMPORTANT! Check SRH.0 ! Supply voltage and some status or error bits are set ONLY after ENDINIT is executed. If SRH.0 = 0 and you use EasySetUp, download a setup, reset the drive and press the nearby button to send an ENDINIT command. If you are using EasyMotion Studio, run a TML program. This includes execution of ENDINIT.

Drive temperature 43.238 *[°C]*

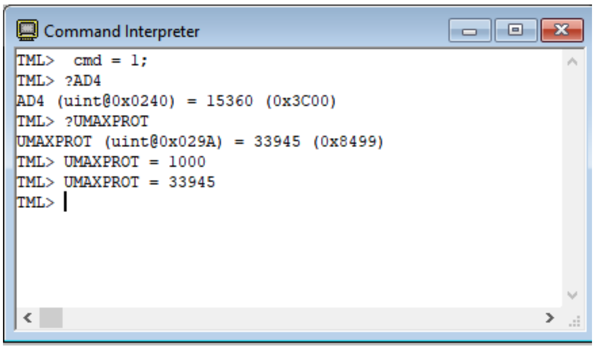
MER - Error Register	
15 - Enable is inactive	0
14 - Command error	0
13 - Under voltage	0
12 - Over voltage	1
11 - Over temp. - Drive	0
10 - Over temp. - Motor	0
9 - I2t	0
8 - Over current	0
7 - LSN (limit -) active	0
6 - LSP (limit +) active	0
5 - Feedback error	0
4 - Serial comm. error	0
3 - Control error	0
2 - Invalid setup data	0
1 - Short-circuit	0
0 - CANbus error	0

DER - Detail Error Register	
15 - EEPROM Locked	0
14 - ENA hardware error	0
13 - Self check error	0
12 - TML heartbeat ignored	0
11 - Start mode failed	0
10 - Encoder broken wire	0
9 - UPD ignored for S-curve	0
8 - Invalid S-curve profile	0
7 - Software LSN active	0
6 - Software LSP active	0
5 - Cancelable call ignored	0
4 - UPD ignored	0
3 - Function not available	0
2 - Homing not available	0
1 - TML stack underflow	0
0 - TML stack overflow	0

DER2 - Detail Error Reg. 2	
15 - Reserved	0
14 - Reserved	0
13 - Reserved	0
12 - Reserved	0
11 - Reserved	0
10 - Reserved	0
9 - Reserved	0
8 - Reserved	0
7 - Reserved	0
6 - Reserved	0
5 - Hall sensor missing	0
4 - AEI interface error	0
3 - BiSS sensor missing	0
2 - BiSS data error	0
1 - BiSS data warning	0
0 - BiSS CRC error	0

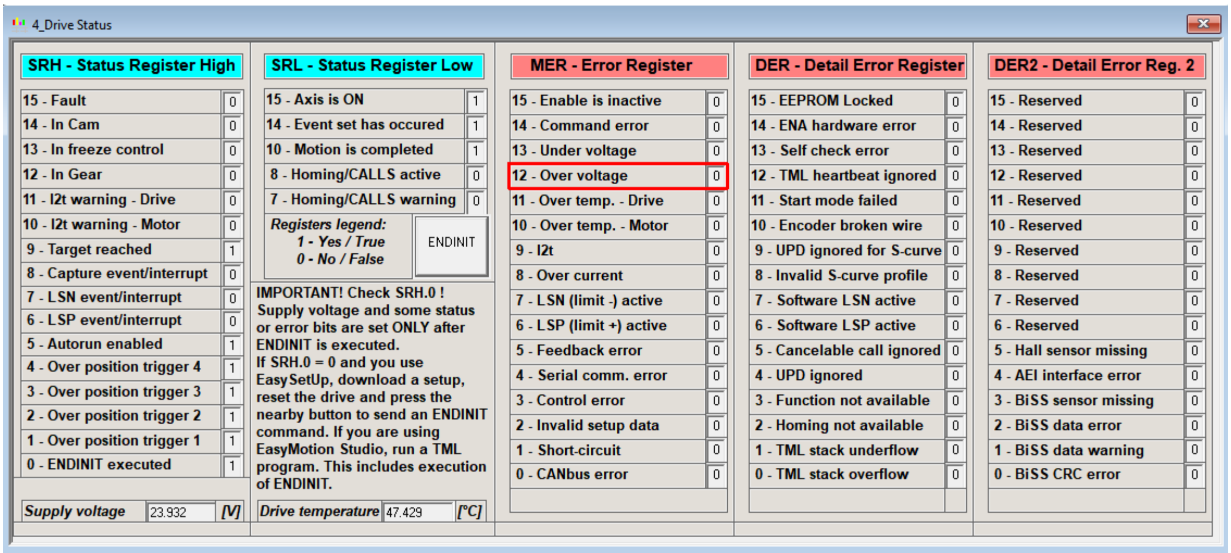
Figure 23. *Over-voltage protection*

Setting the “UMAXPROT” parameter value to the default value, the over-voltage state will disappear and the drive will execute the error recovery procedure.



```
TML>  cmd = 1;
TML> ?AD4
AD4 (uint@0x0240) = 15360 (0x3C00)
TML> ?UMAXPROT
UMAXPROT (uint@0x029A) = 33945 (0x8499)
TML> UMAXPROT = 1000
TML> UMAXPROT = 33945
TML> |
```

Figure 24. *Reset UMAXPROT to the initial value*



4_Drive Status

SRH - Status Register High	
15 - Fault	0
14 - In Cam	0
13 - In freeze control	0
12 - In Gear	0
11 - I2t warning - Drive	0
10 - I2t warning - Motor	0
9 - Target reached	1
8 - Capture event/interrupt	0
7 - LSN event/interrupt	0
6 - LSP event/interrupt	0
5 - Autorun enabled	1
4 - Over position trigger 4	1
3 - Over position trigger 3	1
2 - Over position trigger 2	1
1 - Over position trigger 1	1
0 - ENDINIT executed	1

Supply voltage 23.932 *[V]*

SRL - Status Register Low	
15 - Axis is ON	1
14 - Event set has occured	1
10 - Motion is completed	1
8 - Homing/CALLS active	0
7 - Homing/CALLS warning	0

Registers legend:
1 - Yes / True
0 - No / False

IMPORTANT! Check SRH.0 ! Supply voltage and some status or error bits are set ONLY after ENDINIT is executed. If SRH.0 = 0 and you use EasySetUp, download a setup, reset the drive and press the nearby button to send an ENDINIT command. If you are using EasyMotion Studio, run a TML program. This includes execution of ENDINIT.

Drive temperature 47.429 *[°C]*

MER - Error Register	
15 - Enable is inactive	0
14 - Command error	0
13 - Under voltage	0
12 - Over voltage	0
11 - Over temp. - Drive	0
10 - Over temp. - Motor	0
9 - I2t	0
8 - Over current	0
7 - LSN (limit -) active	0
6 - LSP (limit +) active	0
5 - Feedback error	0
4 - Serial comm. error	0
3 - Control error	0
2 - Invalid setup data	0
1 - Short-circuit	0
0 - CANbus error	0

DER - Detail Error Register	
15 - EEPROM Locked	0
14 - ENA hardware error	0
13 - Self check error	0
12 - TML heartbeat ignored	0
11 - Start mode failed	0
10 - Encoder broken wire	0
9 - UPD ignored for S-curve	0
8 - Invalid S-curve profile	0
7 - Software LSN active	0
6 - Software LSP active	0
5 - Cancelable call ignored	0
4 - UPD ignored	0
3 - Function not available	0
2 - Homing not available	0
1 - TML stack underflow	0
0 - TML stack overflow	0

DER2 - Detail Error Reg. 2	
15 - Reserved	0
14 - Reserved	0
13 - Reserved	0
12 - Reserved	0
11 - Reserved	0
10 - Reserved	0
9 - Reserved	0
8 - Reserved	0
7 - Reserved	0
6 - Reserved	0
5 - Hall sensor missing	0
4 - AEI interface error	0
3 - BiSS sensor missing	0
2 - BiSS data error	0
1 - BiSS data warning	0
0 - BiSS CRC error	0

Figure 25. *The error state resetting*

Once the error recovery procedure is executed and the fault state is reset, the drive will return to the "MasterCommand" loop (in the main program) and wait again for the master command.

5. Conclusions

In case of the complex systems, controlled by a PC / PLC master, is recommended to distributed the intelligence between the master and the slave devices.

In this application, the master sends only the start command and needs to take some decisions when a fault occurs. The rest of the tasks were implemented on the Technosoft drive.

This solution minimizes the communication channel traffic and the tasks that needs to be executed by the master.

In case of the masters where the TML protocol needs to be implemented, this structure has also the advantage that requires to implement (at the master level) only a minimum set of TML instructions.