1. Application note description

The ENABLE input is continuous monitored by the drive firmware. When it switches to the active level, the drive PWM outputs are deactivated, the bit 15 in the MER error register (“Enable is inactive”) is set to 1 and the “int0 – Enable input has changed” interrupt routine is executed.

When the ENABLE input status changes to the inactive state, the bit 15 in the MER error register is reset but the PWM outputs remain deactivated. There is only one exception: if the drive was executing an electronic gearing profile, when the ENABLE input became active, then after the ENABLE input will switch back to the inactive state, the drive power stage and the electronic gearing mode will be enabled automatically.

Remark: The ENABLE input is available only on the drives without STO (safe torque off) functionality.

This application note describes how to program the drive, to reactivate the drive PWM outputs automatically, when the ENABLE input becomes inactive, no-matter what motion profile was in execution, when the ENABLE input changed to the active level.

This functionality was implemented using “int0 – Enable input has changed” interrupt routine. It was modified, to read the bit 15 (“Enable input is inactive”) in the MER error register. If it is 1, the axis is kept disabled. Otherwise, the drive PWM outputs are re-activated.

2. Application flow chart

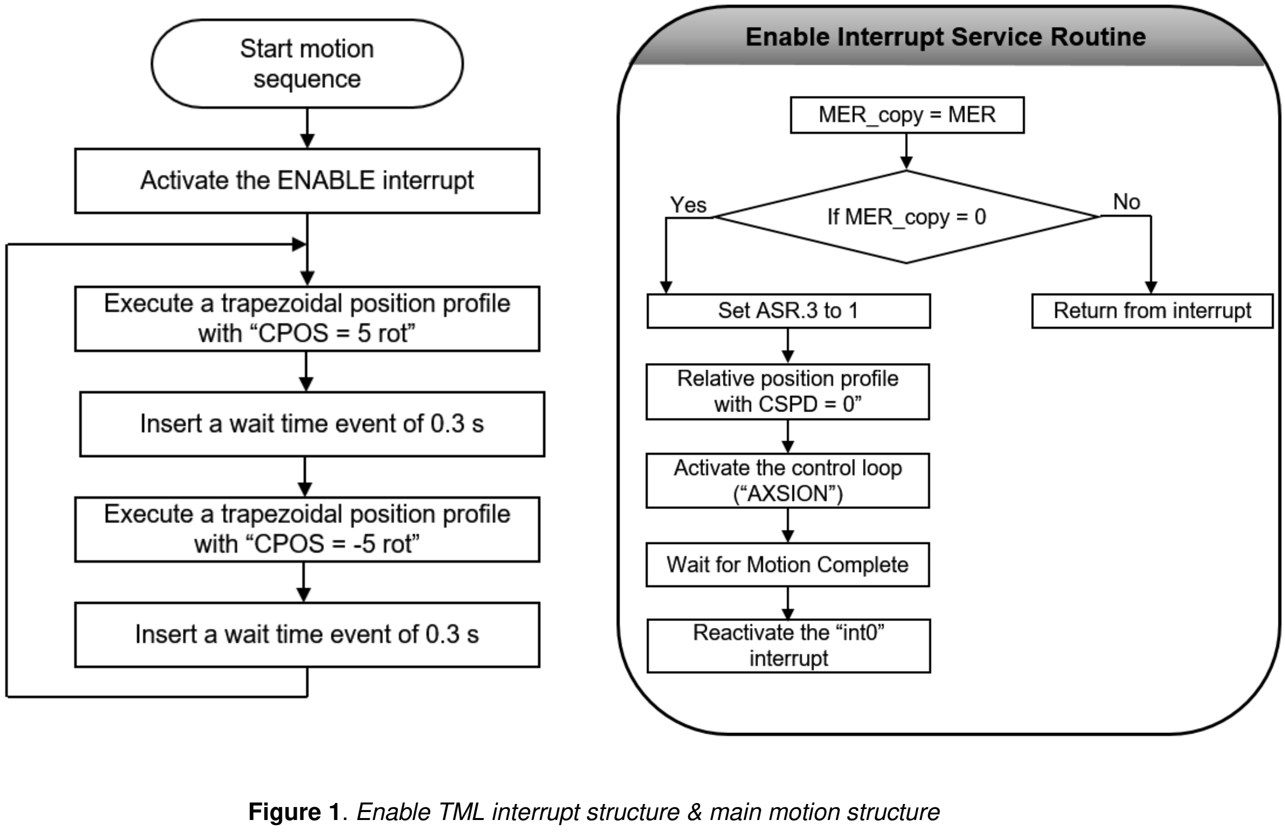
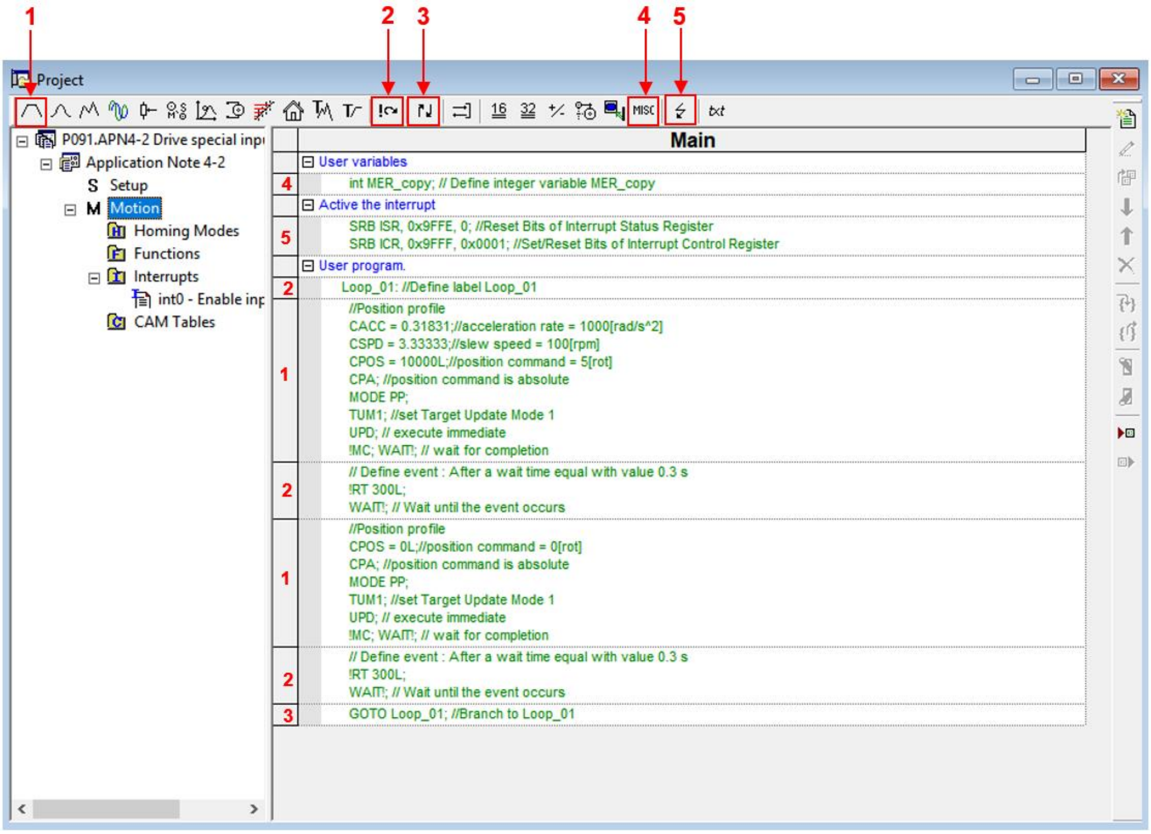

Start motion sequence
Activate the ENABLE interrupt
Execute a trapezoidal position profile with “CPOS = 5 rot”
Insert a wait time event of 0.3 s
Execute a trapezoidal position profile with “CPOS = -5 rot”
Insert a wait time event of 0.3 s
Enable Interrupt Service Routine
MER_copy = MER
If MER_copy = 0
Yes
No
Set ASR.3 to 1
Return from interrupt
Relative position profile with CSPD = 0”
Activate the control loop (“AXSION”)
Wait for Motion Complete
Reactivate the “int0” interrupt

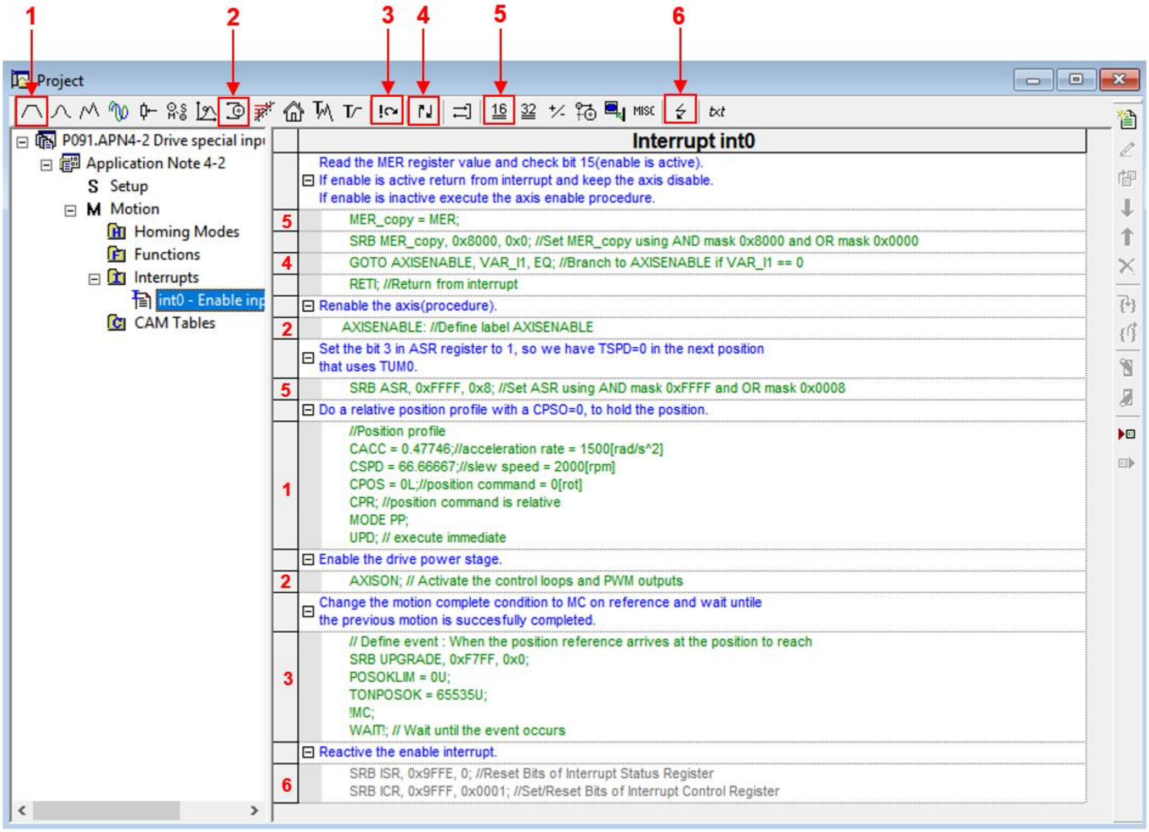
Figure 1. *Enable TML interrupt structure & main motion structure*

3. EasyMotion Studio implementation



```
User variables
    int MER_copy; // Define integer variable MER_copy
Active the interrupt
    SRB ISR, 0x9FFE, 0; //Reset Bits of Interrupt Status Register
    SRB ICR, 0x9FFF, 0x0001; //Set/Reset Bits of Interrupt Control Register
User program.
   Loop_01: //Define label Loop_01
    //Position profile
    CACC = 0.31831;//acceleration rate = 1000[rad/s^2]
    CSPD = 3.33333;//slew speed = 100[rpm]
    CPOS = 10000L;//position command = 5[rot]
    CPA; //position command is absolute
    MODE PP;
    TUM1; //set Target Update Mode 1
    UPD; // execute immediate
    !MC; WAIT!; // wait for completion
    // Define event : After a wait time equal with value 0.3 s
    !RT 300L;
    WAIT!; // Wait until the event occurs
    //Position profile
    CPOS = 0L;//position command = 0[rot]
    CPA; //position command is absolute
    MODE PP;
    TUM1; //set Target Update Mode 1
    UPD; // execute immediate
    !MC; WAIT!; // wait for completion
    // Define event : After a wait time equal with value 0.3 s
    !RT 300L;
    WAIT!; // Wait until the event occurs
    GOTO Loop_01; //Branch to Loop_01
```

Figure 2. *Main application structure*



```
Read the MER register value and check bit 15(enable is active).
If enable is active return from interrupt and keep the axis disable.
If enable is inactive execute the axis enable procedure.
    MER_copy = MER;
    SRB MER_copy, 0x8000, 0x0; //Set MER_copy using AND mask 0x8000 and OR mask 0x0000
    GOTO AXISENABLE, VAR_I1, EQ; //Branch to AXISENABLE if VAR_I1 == 0
    RETI; //Return from interrupt
Renable the axis(procedure).
   AXISENABLE: //Define label AXISENABLE
Set the bit 3 in ASR register to 1, so we have TSPD=0 in the next position
that uses TUM0.
    SRB ASR, 0xFFFF, 0x8; //Set ASR using AND mask 0xFFFF and OR mask 0x0008
Do a relative position profile with a CPSO=0, to hold the position.
    //Position profile
    CACC = 0.47746;//acceleration rate = 1500[rad/s^2]
    CSPD = 66.66667;//slew speed = 2000[rpm]
    CPOS = 0L;//position command = 0[rot]
    CPR; //position command is relative
    MODE PP;
    UPD; // execute immediate
Enable the drive power stage.
    AXISON; // Activate the control loops and PWM outputs
Change the motion complete condition to MC on reference and wait untile
the previous motion is succesfully completed.
    // Define event : When the position reference arrives at the position to reach
    SRB UPGRADE, 0xF7FF, 0x0;
    POSOKLIM = 0U;
    TONPOSOK = 65535U;
    !MC;
    WAIT!; // Wait until the event occurs
Reactive the enable interrupt.
    SRB ISR, 0x9FFE, 0; //Reset Bits of Interrupt Status Register
    SRB ICR, 0x9FFF, 0x0001; //Set/Reset Bits of Interrupt Control Register
```

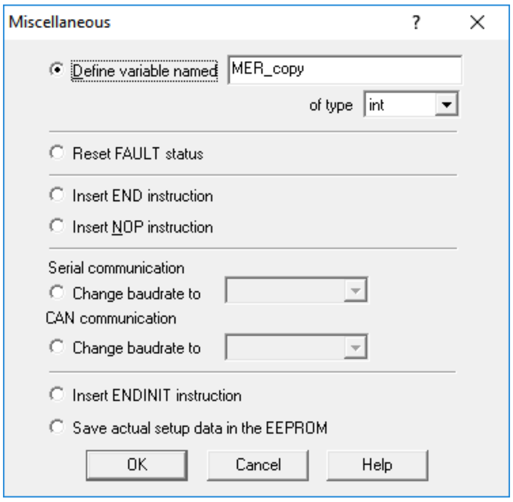
Figure 3. *Enable Interrupt routine structure*

4. Detailed description

4.1 Motion sequences

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 (4), allows declaring the user variables and inserting different TML instructions. Here, it was used to declare the “MER_copy” user variable that will be used later, in the “int0 – Enable input has changed” interrupt service routine.



● Define variable named MER_copy
of type int

○ Reset FAULT status

○ Insert END instruction

○ Insert NOP instruction

Serial communication
○ Change baudrate to

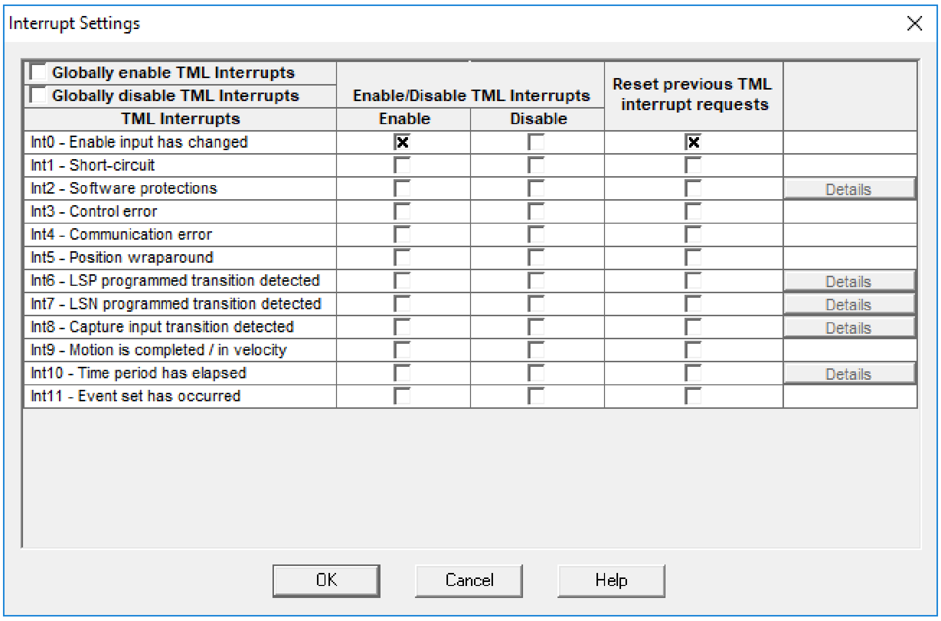
CAN communication
○ Change baudrate to

○ Insert ENDINIT instruction

○ Save actual setup data in the EEPROM

Figure 4. *How to declare a user variable*

• The “Interrupt Settings” dialog (5) allows to activate and/or deactivate the TML (Technosoft Motion Language) interrupts. In this case it was used to activate the “Int0 – Enable input has changed” interrupt routine, that was used to implement the functionality described in the first chapter.



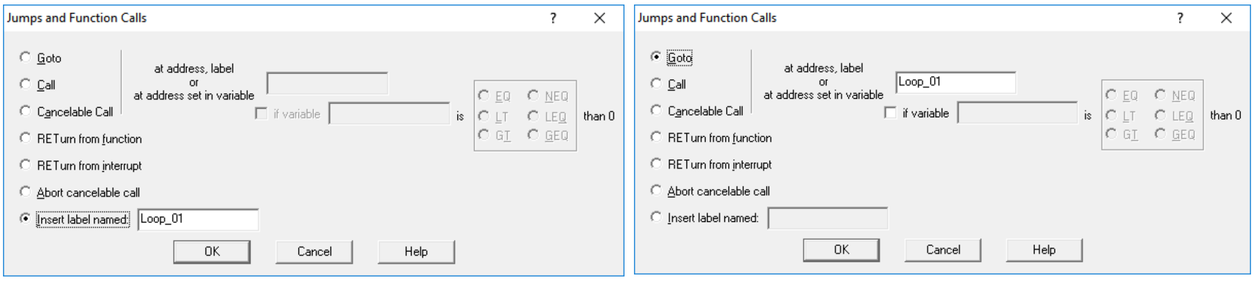
□ Globally enable TML Interrupts
□ Globally disable TML Interrupts

TML Interrupts	Enable/Disable TML Interrupts Enable	 Disable	Reset previous TML interrupt requests	
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	□	□	□	

Figure 5. *How active the enable input*

• The “Jumps and Function Calls” dialogue (3) 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 a loop where the motor runs 5 rotations in the positive direction and then 5 rotations in the negative direction.



Jumps and Function Calls

○ Goto
○ Call
○ Cancelable Call
○ RETurn from function
○ RETurn from interrupt
○ Abort cancelable call
● Insert label named: Loop_01

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

Jumps and Function Calls

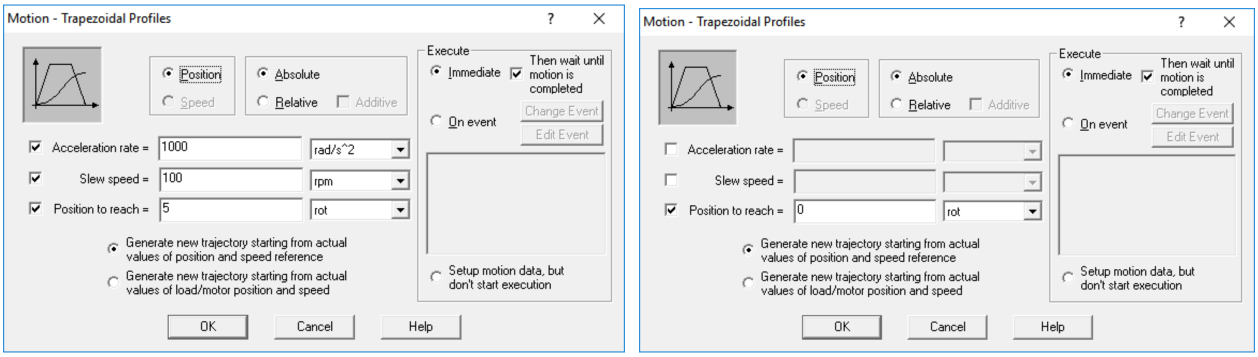
● Goto
○ Call
○ Cancelable Call
○ RETurn from function
○ RETurn from interrupt
○ Abort cancelable call
○ Insert label named:

at address, label or at address set in variable: Loop_01
□ if variable is ○ EQ ○ NEQ ○ LT ○ LEQ ○ GT ○ GEQ than 0

Figure 6. *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.

In this case it was used to move the motor for 5 rotations and then to return to 0.



Motion - Trapezoidal Profiles

● Position ○ Speed
● Absolute ○ Relative □ Additive
☑ Acceleration rate = 1000 rad/s^2
☑ Slew speed = 100 rpm
☑ Position to reach = 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

Motion - Trapezoidal Profiles

● Position ○ Speed
● Absolute ○ Relative □ Additive
□ Acceleration rate =
□ Slew speed =
☑ Position to reach = 0 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

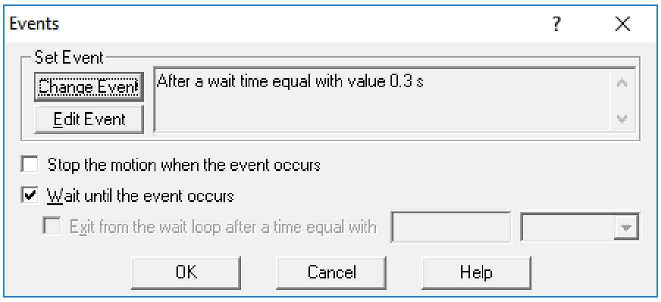
Execute: ● Immediate ☑ Then wait until motion is completed
○ On event
○ Setup motion data, but don't start execution

Figure 7. *How to configure and start motion using a trapezoidal position profile*

• The “Events” dialogue (2) allows defining 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.

In this case, the “Events” dialog was used to insert a 0.3 s delay between the two movements. This reduces the shock when motion is reversed.



Events

Set Event: After a wait time equal with value 0.3 s

□ Stop the motion when the event occurs
☑ Wait until the event occurs
□ Exit from the wait loop after a time equal with

Figure 8. *How to create an event*

4.2 Enable input has changed 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.

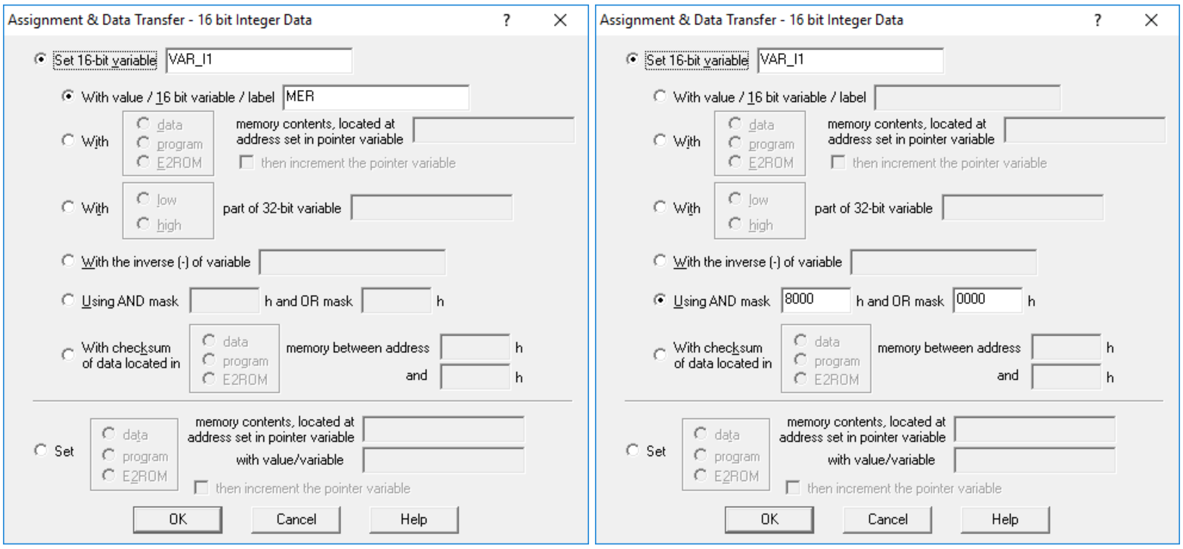
Remark: While an interrupt is active, the other interrupts are deactivated. It is recommended to keep the ISR as short as possible. If is not possible, then the other interrupts should be re-enabled using the "Interrupts Settings" dialogue.

This application was implemented using the "Int0 – Enable input has changed" interrupt.

The code sequence inside the "Int0 – Enable input has changed" interrupt was generated using the buttons marked with 1 to 6 in Figure 3. Clicking on those buttons the following programming dialogs will open.

- The "Assignment and Data Transfer – 16 bit Integer Data" dialogue (5) allows different operations with a 16 bit integer variable/parameter/register.

Here, the "Assignment and Data Transfer – 16 bit Integer Data" dialogue was used to set the "VAR_I1" internal variable with the value of the MER error register and then, to apply an "AND" and "OR" mask, that isolates the bit 15 in the MER error register ("Enable is inactive").



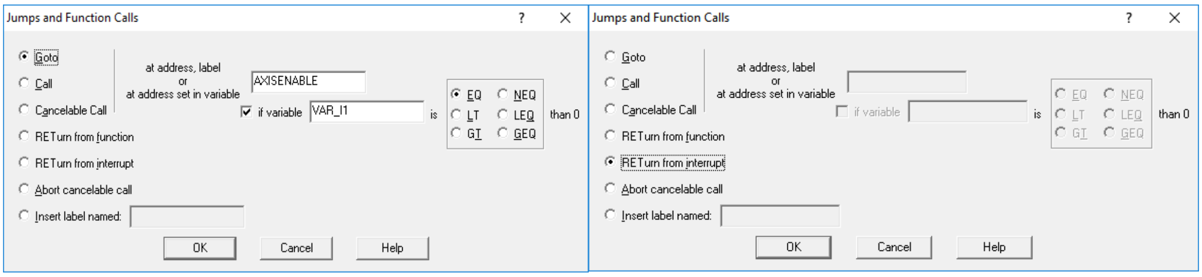
Assignment & Data Transfer - 16 bit Integer Data
● Set 16-bit variable: VAR_I1
● With value / 16 bit variable / label: MER

Assignment & Data Transfer - 16 bit Integer Data
● Set 16-bit variable: VAR_I1
● Using AND mask: 8000 h and OR mask: 0000 h

? ×
○ 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
OK
Cancel
Help

Figure 9. *Operations using the "Assignment and Data Transfer – 16 bit Integer Data" dialog*

- The "Jumps and Function Calls" dialogue (4) was used to generate the "GOTO AXSIENABLE;" and "RETI;" instructions that makes the program to jump to the "AXISENABLE;" label, if MER.15 is 1 or to return from the interrupt ("RETI;") if MER.15 is 0.

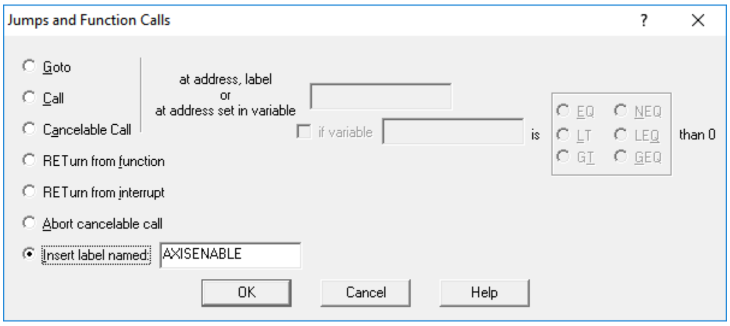


Jumps and Function Calls
● Goto
at address, label or at address set in variable: AXISENABLE
☑ if variable VAR_I1 is ● EQ than 0

Jumps and Function Calls
● RETurn from interrupt

Figure 10. *Insert a "GOTO;" and "RETI;" TML instruction*

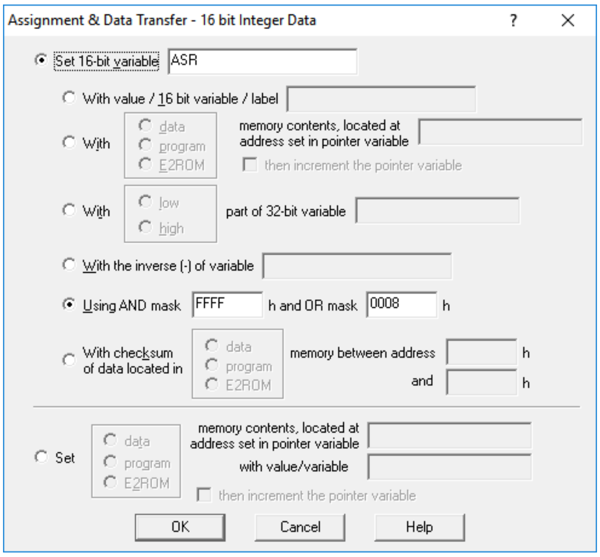
The “AXISENABLE” label was also created using the “Jumps and Function Calls” dialogue (4).


Jumps and Function Calls

● Insert label named: AXISENABLE

Figure 11. *How to create a label*

• The “Assignment and Data Transfer – 16 bit Integer Data” dialogue (5) was used again, to apply an “AND” and “OR” mask, that sets the bit 3 in the ASR register is set to 1. This will make the target speed 0, when the next position profile (with TUMO - the reference is generated starting from the actual value of load/motor position and speed) is executed. The purpose of this instruction is to prevent the motor from jumping to the last imposed position, when the drive power stage is reactivated.


Assignment & Data Transfer - 16 bit Integer Data

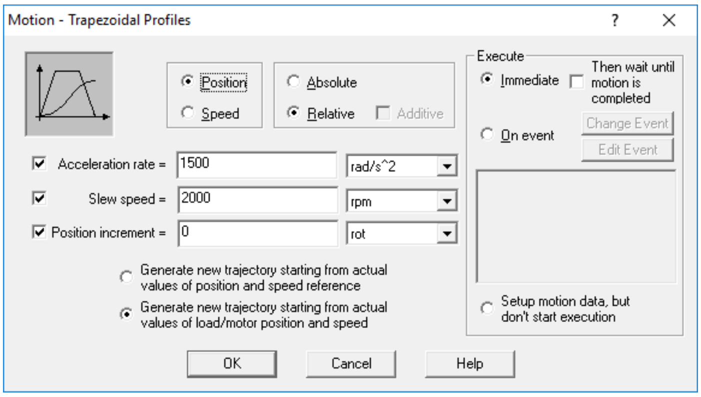
● Set 16-bit variable: ASR

● Using AND mask FFFF h and OR mask 0008 h

? ×
○ 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
○ 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
OK
Cancel
Help

Figure 12. *Set the bit 3 in the ASR register to 1*

• The “Motion – Trapezoidal Profiles” dialog (1) was used to execute a relative position profile, with “TUM0” and a position increment of 0 rot. This way the motor will hold the current position, after the error state disappears and the axis is re-enabled.


Motion - Trapezoidal Profiles

● Position

● Relative

☑ Acceleration rate = 1500 rad/s^2

☑ Slew speed = 2000 rpm

☑ Position increment = 0 rot

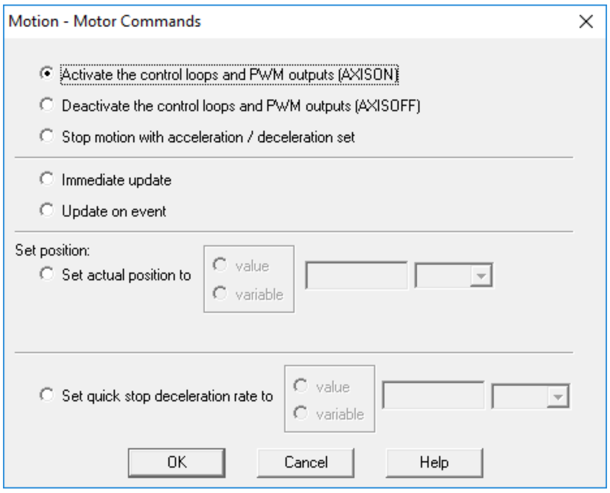
● Generate new trajectory starting from actual values of load/motor position and speed

Execute: ● Immediate

? ×
○ Speed
○ Absolute
□ Additive
○ Generate new trajectory starting from actual values of position and speed reference
□ Then wait until motion is completed
○ On event
Change Event
Edit Event
○ Setup motion data, but don't start execution
OK
Cancel
Help

Figure 13. *How to configure and start motion using a position profile using TUM0*

- The “Motion - Motor Commands” dialogue (2) is used 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

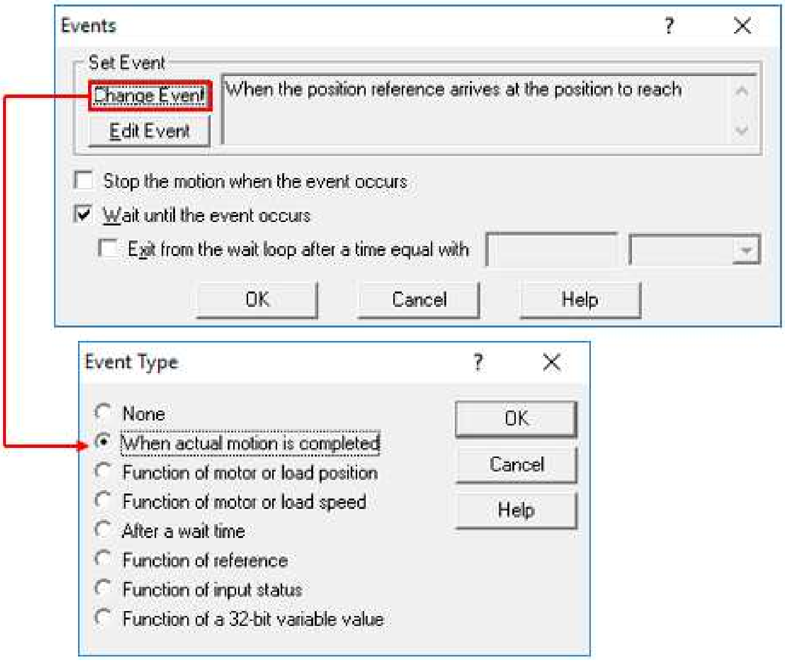
○ Immediate update
○ Update on event

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

○ Set quick stop deceleration rate to ○ value ○ variable

Figure 14. *How to set AXISON*

- The “Events” dialogue (3) was used here to hold the program execution until the previous motion profile (with CPOS = 0 rpm) is executed and the Motion Complete bit is set to 1.



Events

Set Event
Change Event
Edit Event
When the position reference arrives at the position to reach

□ Stop the motion when the event occurs
☑ Wait until the event occurs
□ Exit from the wait loop after a time equal with

Event Type

○ None
● When actual motion is completed
○ Function of motor or load position
○ Function of motor or load speed
○ After a wait time
○ Function of reference
○ Function of input status
○ Function of a 32-bit variable value

Figure 15. *Set an event on Motion Complete.*

Once the code under the “AXISENABLE” label is executed, the drive power stage (the PWM outputs) will be reactivated and the motor will hold the current position.

The program will return from the “Int0 – Enable input has changed” interrupt routine, to the last executed instruction in the main TML program. As most probably this instruction is one of the two trapezoidal position profiles in the “Loop_01” loop, the motor will start to spin again, performing the back and forward motion inside the loop.

Remark: In case the ENABLE input needs to be used as a general purpose input, its default behavior, can be disabled by setting the “ENABLEOFF” parameter to 1.