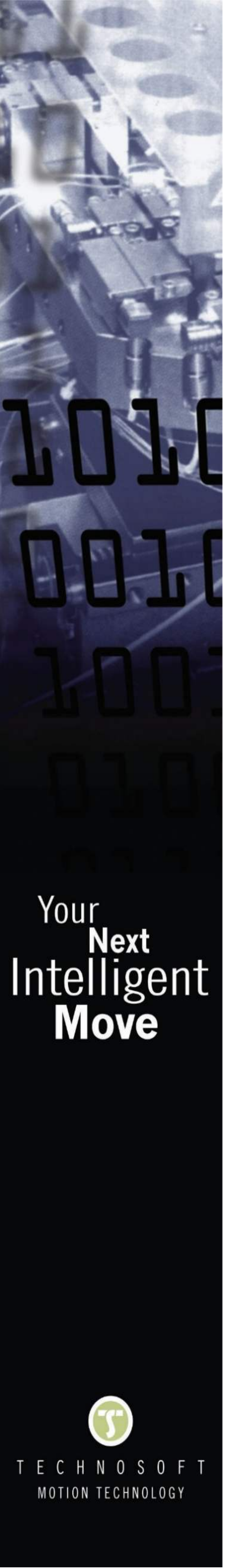
TECHNOSOFT
MOTION TECHNOLOGY



TML interrupt routines usage

Application Note

Easy Motion Studio II

Table of contents

1. Application description

This application note describes how to activate and customize the TML interrupt services routines, using an example that sets the “Int 10 – Time period has elapsed” interrupt, to flash a LED, connected to one of the drive digital outputs.

The TML interrupts are special conditions 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. While an interrupt is active (meaning the ISR code is being executed), all the TML interrupts are globally disabled.

That is why, it is recommended to put in the ISR (Interrupt Status Register) only the minimum needed code. If this is not possible, then other interrupts can be re-enabled using the “Interrupts Settings” dialogue (will be presented in chapter 3).

2. Application flow chart

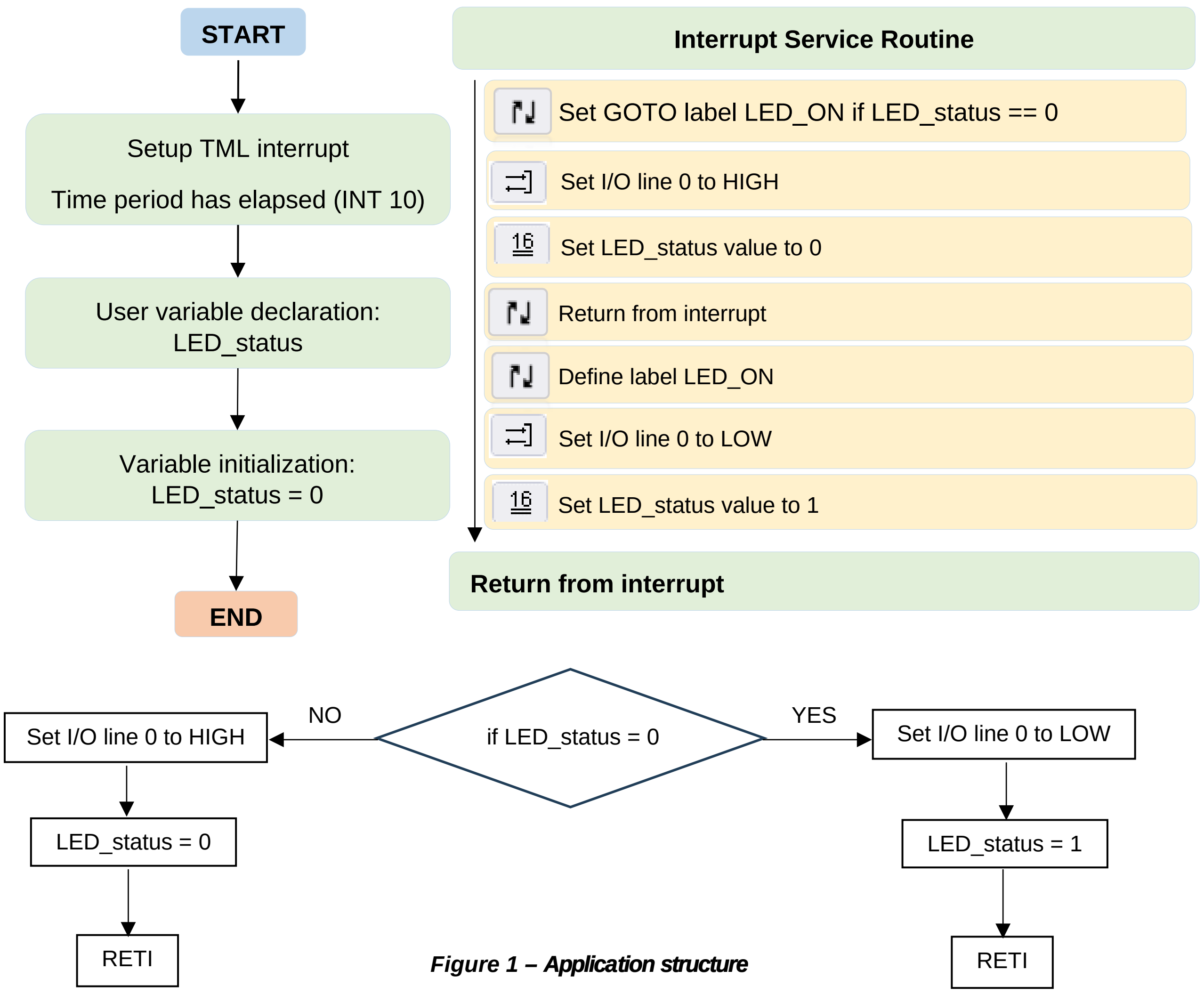


Figure 1 – Application structure

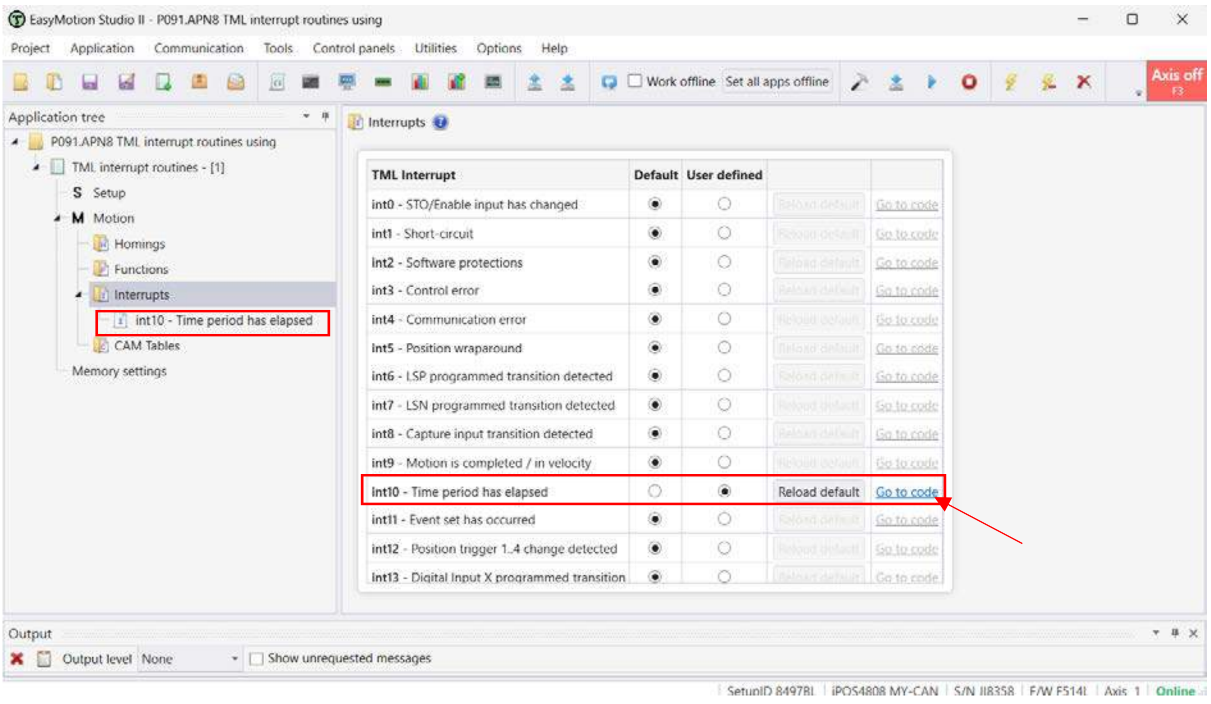
3. EasyMotion Studio II implementation

The application implementation in EasyMotion Studio II, includes 2 main parts:

1) The interrupt ("int10") that contains the TML code to be executed each time the interrupt is triggered.
2) The main program, in the motion branch, that will include the code for using the TML interrupts.

3.1 Time period Interrupt routine

The "Interrupts" section allows you to customize the TML interrupt service routines. Once the "User defined" option is marked the interrupt routine will appear in the project window (left side), under the "Interrupts" section.


EasyMotion Studio II - P091.APN8 TML interrupt routines using
Project Application Communication Tools Control panels Utilities Options Help
Work offline
Set all apps offline
Axis off
Application tree
P091.APN8 TML interrupt routines using
TML interrupt routines - [1]
S Setup
M Motion
Homings
Functions
Interrupts
int10 - Time period has elapsed
CAM Tables
Memory settings
Interrupts
Output
Output level None
Show unrequested messages
SetupID 8497BL | iPOS4808 MY-CAN | S/N JI8358 | F/W F514L | Axis 1 | Online

TML Interrupt	Default	User defined		
int0 - STO/Enable input has changed	●	○	Reload default	Go to code
int1 - Short-circuit	●	○	Reload default	Go to code
int2 - Software protections	●	○	Reload default	Go to code
int3 - Control error	●	○	Reload default	Go to code
int4 - Communication error	●	○	Reload default	Go to code
int5 - Position wraparound	●	○	Reload default	Go to code
int6 - LSP programmed transition detected	●	○	Reload default	Go to code
int7 - LSN programmed transition detected	●	○	Reload default	Go to code
int8 - Capture input transition detected	●	○	Reload default	Go to code
int9 - Motion is completed / in velocity	●	○	Reload default	Go to code
int10 - Time period has elapsed	○	●	Reload default	Go to code
int11 - Event set has occurred	●	○	Reload default	Go to code
int12 - Position trigger 1..4 change detected	●	○	Reload default	Go to code
int13 - Digital Input X programmed transition	●	○	Reload default	Go to code

Figure 2 – Customizing the TML interrupt service routines

This application uses the "Int10 – Time period has elapsed" interrupt routine, to check the value of the "LED_status" user variable and command the OUT(0) digital output.

In order to customize the content of an interrupt service routine, it is necessary to click on the "Go to code" link. This will navigate to the specific section of the application where the desired functionality for the interrupt can be defined and implemented.

The “Jumps and Function Calls” dialogue allows to control the TML program flow through unconditional or conditional jumps and unconditional, conditional or cancelable calls of TML functions. In this application, the “Jumps and Function Calls” dialogue was used to create some conditional jumps, function of the “LED_status” user variable (if “LED_status = 0” the “OUT(0)” output is set to the active level, to switch ON the LED. Otherwise, the “OUT(0)” is set inactive, to switch OFF the LED).

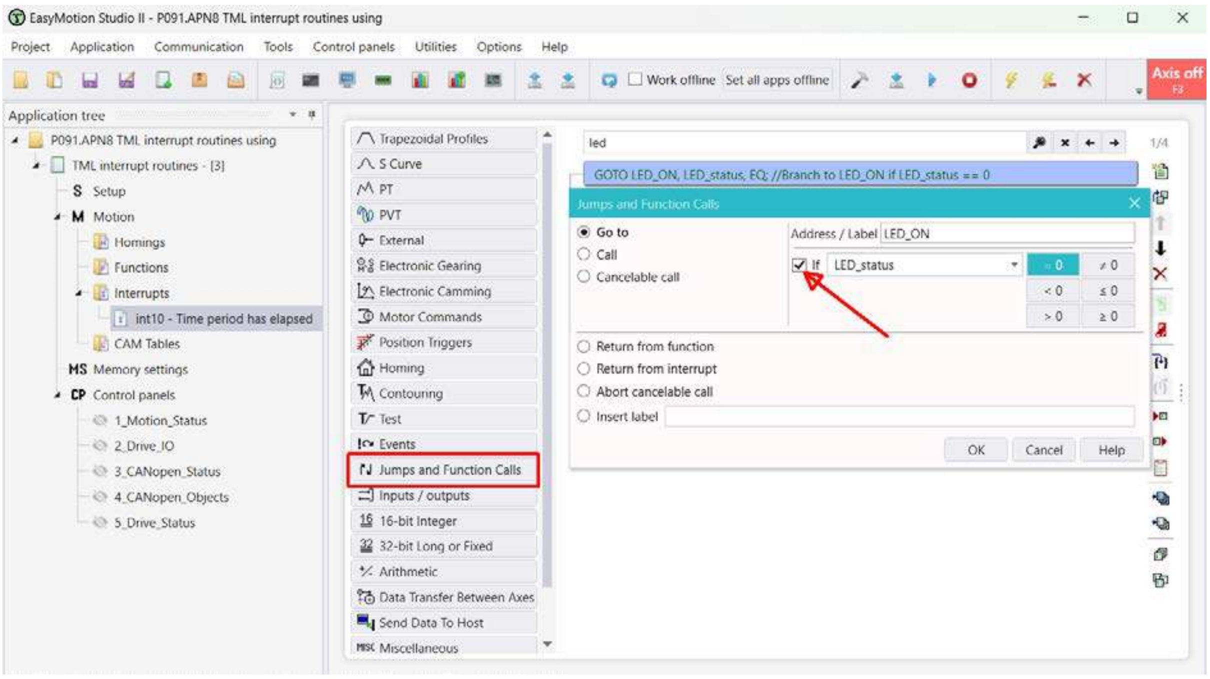
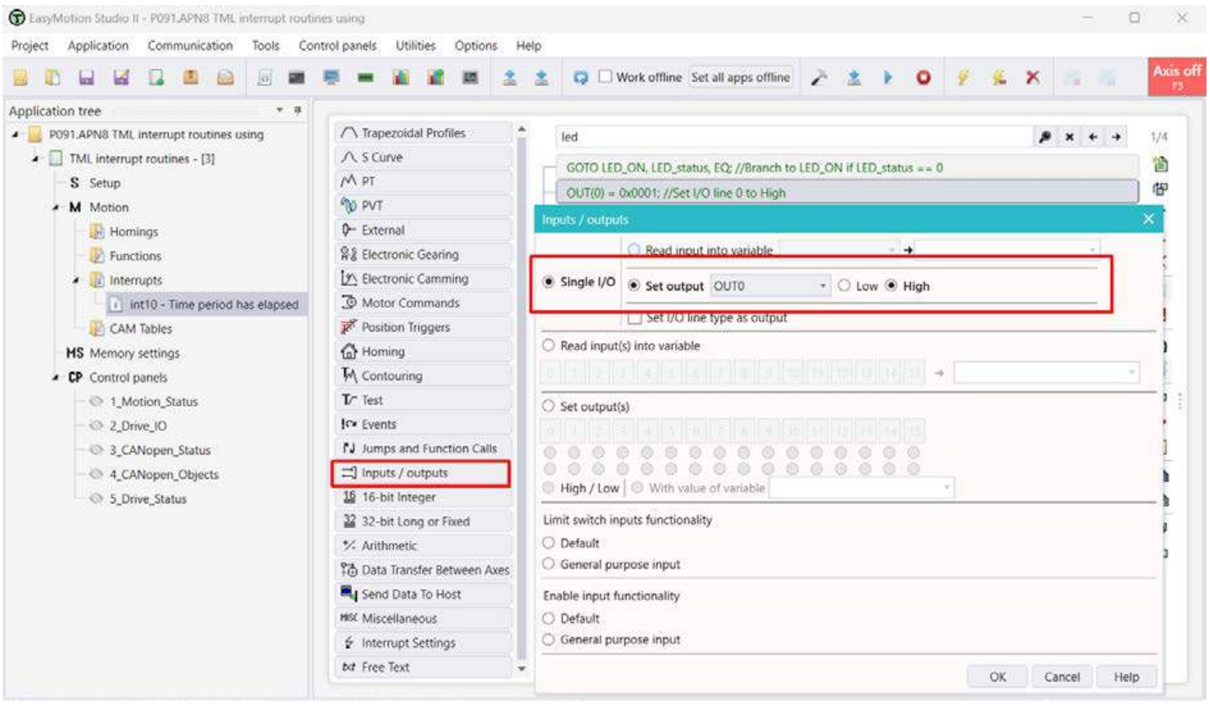

```
GOTO LED_ON, LED_status, EQ; //Branch to LED_ON if LED_status == 0
```
[✓ Go to]
Address / Label LED_ON
[✓ If] LED_status [✓ = 0]

Figure 3 – Conditional jump to LED_ON label based on LED_status

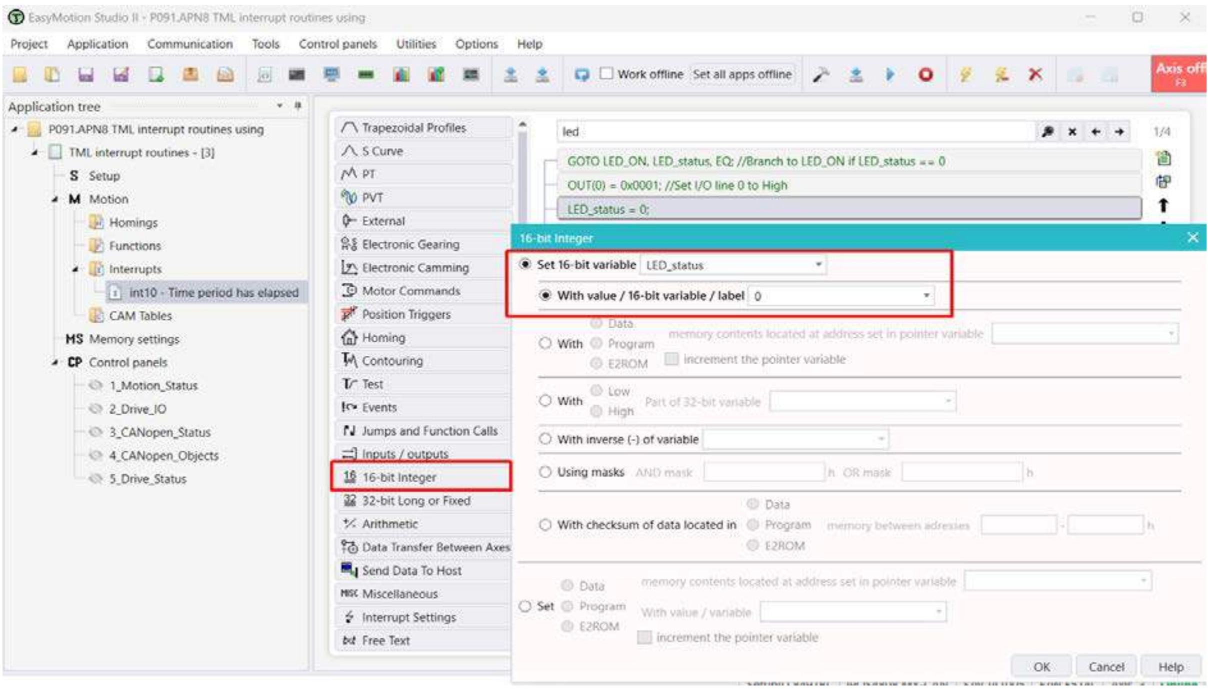
The “I/O” dialogue allows different operations with the drive digital inputs and outputs. It was used here to set the “OUT(0)” digital output LOW or HIGH (function on the “LED_status” variable value). This way, the LED connected to this input is switched ON or OFF.


```
GOTO LED_ON, LED_status, EQ; //Branch to LED_ON if LED_status == 0
OUT(0) = 0x0001; //Set I/O line 0 to High
```
[✓ Single I/O] [✓ Set output] OUT0 Low [✓ High]

EasyMotion Studio II - P091.APN8 TML interrupt routines using
Application tree
int10 - Time period has elapsed
Inputs / outputs
Read input into variable
Set I/O line type as output
Read input(s) into variable
Set output(s)
High / Low
With value of variable
Limit switch inputs functionality
Default
General purpose input
Enable input functionality
Default
General purpose input
OK
Cancel
Help

Figure 4 – Setting the I/O line to HIGH

Once the OUT(0) digital output status is changed, the “Assignment and Data Transfer – 16 bit Integer Data” dialogue is used to modify the “LED_status” variable value. This has the purpose to indicate that the LED is ON (“LED_status = 1”) or OFF (”LED_status = 0”).



```
GOTO LED_ON, LED_status, EQ; //Branch to LED_ON if LED_status == 0
OUT(0) = 0x0001; //Set I/O line 0 to High
LED_status = 0;
```

16-bit Integer

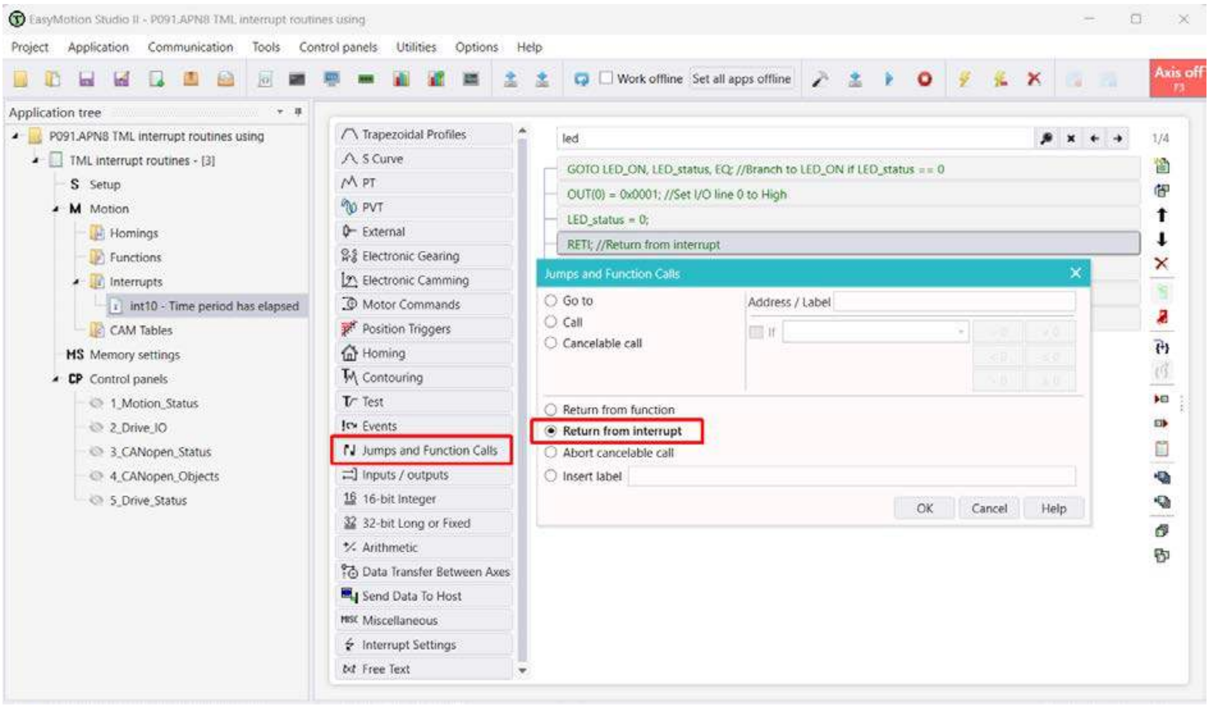
Set 16-bit variable: LED_status

With value / 16-bit variable / label: 0

Figure 5 – Setting the LED_status value to 0

After the “OUT(0)” digital output state is changed and the “LED_status” variable is set accordingly, the program should exit from the interrupt. This is done using the “RETI;” (return from interrupt) instruction. It’s located in the “Jumps and Function Calls” dialogue.

<u>Remark</u>: In the second case (when the LED is switched off), the “RETI” instruction is not added anymore because the EasyMotion Studio II automatically places one at the end of the interrupt routine.



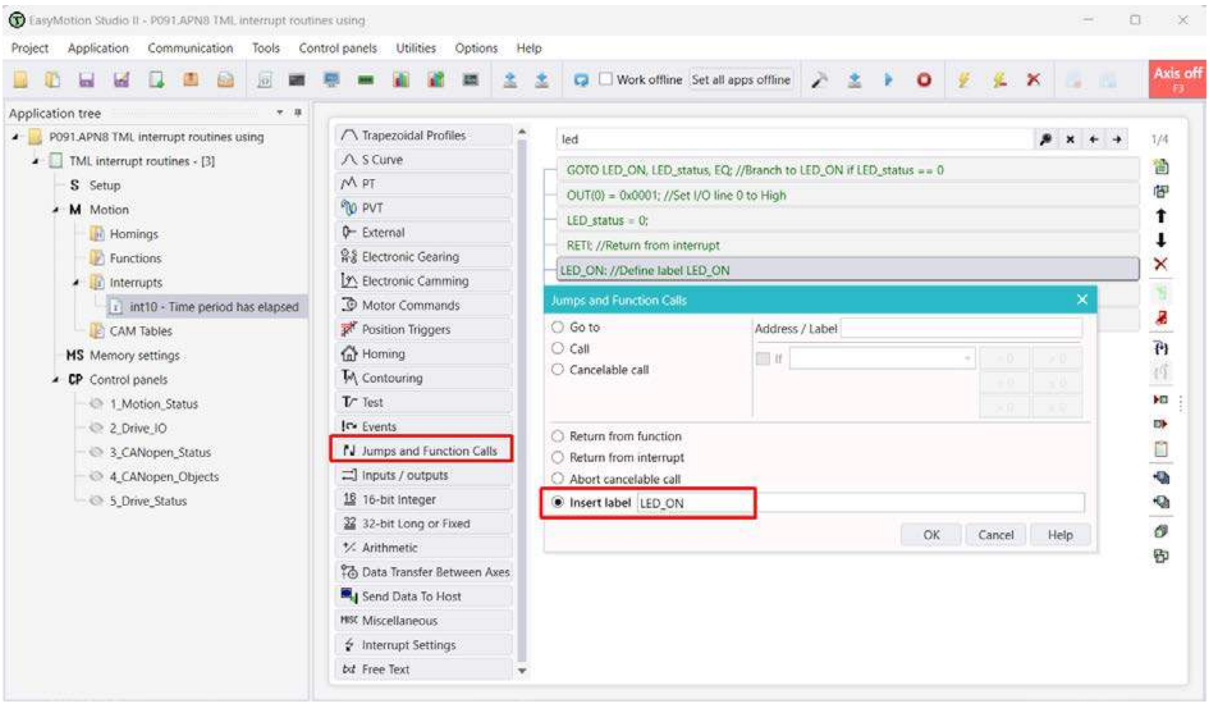
```
GOTO LED_ON, LED_status, EQ; //Branch to LED_ON if LED_status == 0
OUT(0) = 0x0001; //Set I/O line 0 to High
LED_status = 0;
RETI; //Return from interrupt
```

Jumps and Function Calls

Return from interrupt [✓]

Figure 6 – Return from interrupt

The LED_ON label is generated using the "Jumps and Function Calls" wizard and is used to indicate where the program should jump if the “LED_status == 0” condition is true.



```
GOTO LED_ON, LED_status, EQ; //Branch to LED_ON if LED_status == 0
OUT(0) = 0x0001; //Set I/O line 0 to High
LED_status = 0;
RETI; //Return from interrupt
LED_ON: //Define label LED_ON
```

[✓ Insert label] LED_ON

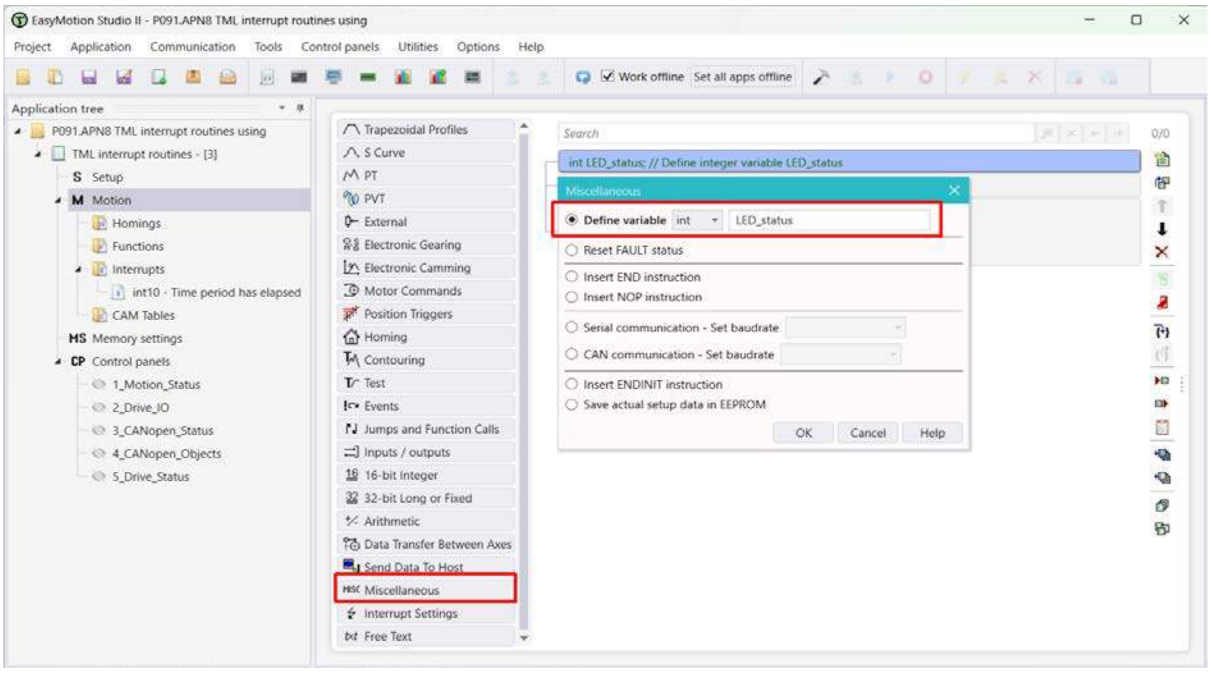
Figure 7 – Defining LED_ON label

The remaining instructions from the ISR10 are complementary to what was presented already.

Each time the ISR10 gets executed the state of the LED will be toggled depending on the value of the LED_status variable.

3.2 Main motion section

The “Miscellaneous” dialogue allows to declare user variables, reset the drive/motor fault status, execute the “END” / “NOP” / “ENDINIT” TML instructions, change the CAN / RS-232 baudrate and save the actual setup into the drive memory. In this case the “Miscellaneous” dialogue was used to declare “LED_status” the 16bit integer user variable, that is used to track the output status (active or inactive).

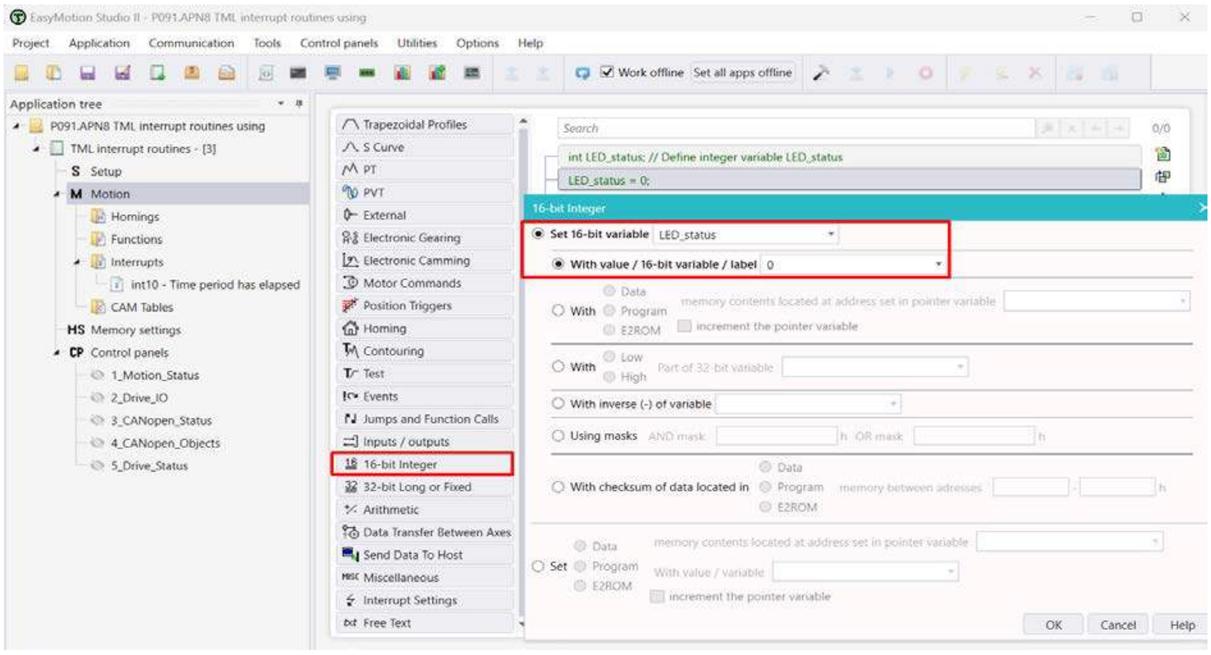


```
int LED_status; // Define integer variable LED_status
```

Miscellaneous:
- [✓ Define variable] int LED_status
- Reset FAULT status
- Insert END instruction
- Insert NOP instruction
- Serial communication - Set baudrate
- CAN communication - Set baudrate
- Insert ENDINIT instruction
- Save actual setup data in EEPROM

Figure 8 – Defining variable LED_status

The “Assignment and Data Transfer – 16 bit Integer Data” dialogue allows different operations with the 16-bit integer variables / parameters / registers. Here it was used to initialize the “LED_status” user variable with 0.

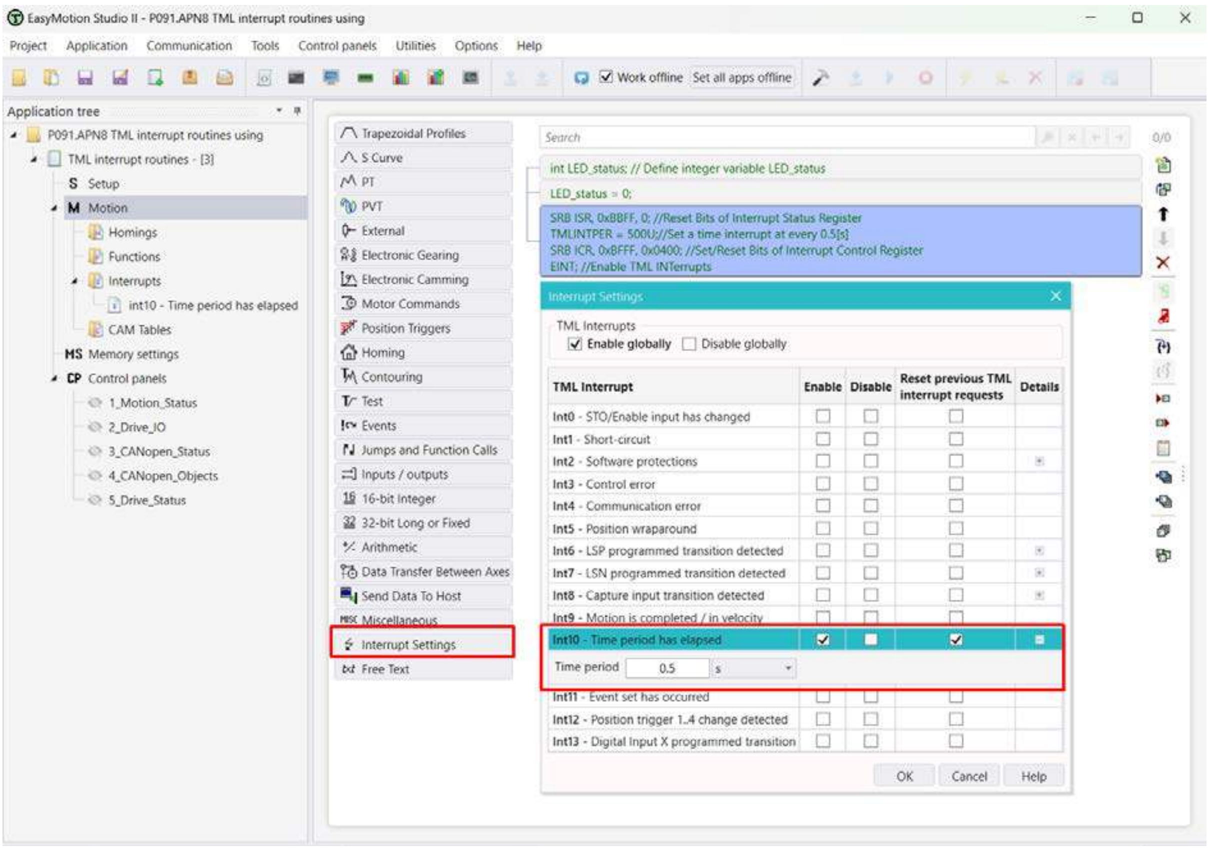


```
int LED_status; // Define integer variable LED_status
LED_status = 0;
```

16-bit Integer:
- [✓ Set 16-bit variable] LED_status
  - [✓ With value / 16-bit variable / label] 0
  - With Data / Program / E2ROM memory contents located at address set in pointer variable; increment the pointer variable
  - With Low / High Part of 32-bit variable
  - With inverse (-) of variable
  - Using masks AND mask h OR mask h
  - With checksum of data located in Data / Program / E2ROM memory between adresses h
- Set Data / Program / E2ROM memory contents located at address set in pointer variable With value / variable; increment the pointer variable

Figure 9 – Setting the LED_status value to 0

The “Interrupt Settings” dialogue allows to activate and/or deactivate the TML (Technosoft Motion Language) interrupts. In this case, it was used to activate the “int10 – Time period has elapsed” interrupt routine and set it to a time period of 0.5 s.



```
int LED_status; // Define integer variable LED_status
LED_status = 0;
SRB ISR, 0xBBFF, 0; //Reset Bits of Interrupt Status Register
TMLINTPER = 500U;//Set a time interrupt at every 0.5[s]
SRB ICR, 0xBFFF, 0x0400; //Set/Reset Bits of Interrupt Control Register
EINT; //Enable TML INTerrupts
```

Interrupt Settings

TML Interrupts
[✓ Enable globally] Disable globally

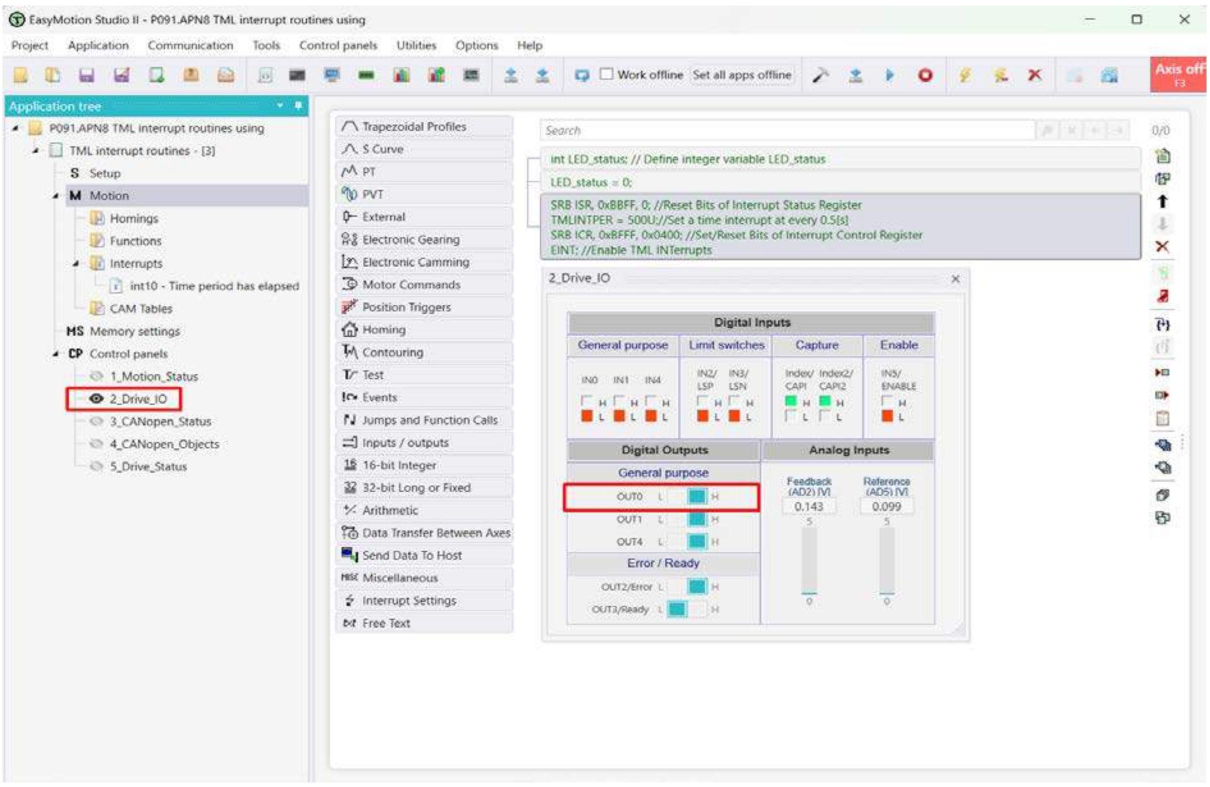
TML Interrupt	Enable	Disable	Reset previous TML interrupt requests	Details
Int0 - STO/Enable input has changed	☐	☐	☐	
Int1 - Short-circuit	☐	☐	☐	
Int2 - Software protections	☐	☐	☐	⊞
Int3 - Control error	☐	☐	☐	
Int4 - Communication error	☐	☐	☐	
Int5 - Position wraparound	☐	☐	☐	
Int6 - LSP programmed transition detected	☐	☐	☐	⊞
Int7 - LSN programmed transition detected	☐	☐	☐	⊞
Int8 - Capture input transition detected	☐	☐	☐	⊞
Int9 - Motion is completed / in velocity	☐	☐	☐	
Int10 - Time period has elapsed	✓	☐	✓	⊟
Time period 0.5 s				
Int11 - Event set has occurred	☐	☐	☐	
Int12 - Position trigger 1..4 change detected	☐	☐	☐	
Int13 - Digital Input X programmed transition	☐	☐	☐	

OK Cancel Help

Figure 10 – Interrupt settings

4. Application evaluation

After running the TML program, we can use the “2_Drive IO” control panel in EasyMotion Studio II to monitor the status of the OUT0 digital output.



```
int LED_status; // Define integer variable LED_status
LED_status = 0;
SRB ISR, 0xBBFF, 0; //Reset Bits of Interrupt Status Register
TMLINTPER = 500U;//Set a time interrupt at every 0.5[s]
SRB ICR, 0xBFFF, 0x0400; //Set/Reset Bits of Interrupt Control Register
EINT; //Enable TML INTerrupts
```

Figure 11 – Using the ‘2_Drive IO” control panel

To start a control panel, right-click on it, and then select "START" from the context menu that appears.

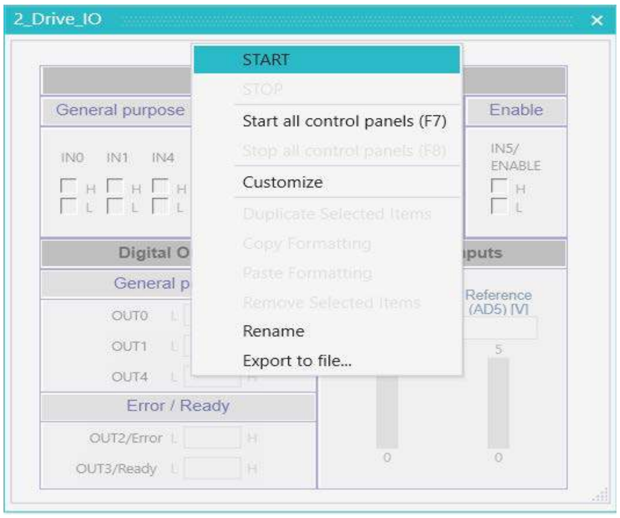


Figure 12 – Starting the ‘2_Drive IO” control panel