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 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.

While an interrupt is active, the other interrupts are deactivated. That is why, it is recommended to keep the ISR as short as possible. If this is not possible, then the other interrupts should be re-enabled using the "Interrupts Settings" dialogue (will be presented in chapter 4).

2. Application flow chart



Figure 1. Application structure

3. EasyMotion Studio implementation

		1 2 3	
Project			
ハハベ物 ┡ ╬ ២ Ͽ ♥ Ѧ Խ T/ !~	۲J	☐ 1일 월 ½ 월 ■ Misc 🗲 bat	10
TMLinterrupt		Main	
🖃 👘 Untitled Application		Variable declaration	li.
S Setup	2	int LED_status; // Define integer variable LED_status	E.
Motion		Variable initialization	Ļ
Homing Modes	1	LED_status = 0;	^
Functions		Enable TML Interrupt	·
🖃 🛅 Interrupts		SRB ISR, 0x9BFF, 0; //Reset Bits of Interrupt Status Register	<u> </u>
🚹 int10 - Time period has elapsed	3	SRB ICR, 0x9FFF, 0x0400; //Set/Reset Bits of Interrupt Control Register	F
CAM Tables		EINT; //Enable TML INTerrupts	R
2			<u></u>
			19
			æ
J	1		







4. Detailed description of the EasyMotion Studio implementation

4.1 Motion section

The code sequences in the "Motion" section were generated using the buttons marked with 1 to 3 in Figure 2. Clicking on those buttons the following programming dialogues will open.

• The "Miscellaneous" dialogue (2) allows to declare user variables, reset/exit the drive/motor from the 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" user variable, that is use to track the output status (active or inactive).

• The "Assignment and Data Transfer – 16 bit Integer Data" dialogue (1) allows different operations with the 16-bit integer variables / parameters / registers. Here it was sued to initialize the "LED_status" user variable with 0.

Miscellaneous	?	\times
Define variable named LED_status		
of type int	•	
C Reset FAULT status		
C Insert END instruction		
C Insert NOP instruction		
Serial communication		
C Change baudrate to 📃 🚬		
CAN communication		
C Change baudrate to		
C Insert ENDINIT instruction		
C Save actual setup data in the EEPROM		
OK Cancel Help		



Figure 4. Defining the user variables

Figure 5. Set the a user variable

• The "Interrupt Settings" dialogue (3) 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 0.5 s.

TML Interrupts Enable Disable Int0 - Enable input has changed Import of the second of	Globally disable TML Interrupts	Enable/Disable	TML Interrupts	Reset previous TML	
Into - Enable input has changed	TML Interrupts	Enable	Disable	interrupt requests	
ntl - Short-circuit Image: Control end of the short-circuit Image: Control end of the short-circuit ntl 2 - Software protections Image: Control end of the short-circuit Image: Control end of the short-circuit Details ntl 2 - Control end of the short-circuit Image: Control end of the short-circuit Image: Control end of the short-circuit Details ntl 2 - Control end of the short-circuit Image: Control end of the short-circuit Image: Control end of the short-circuit Details ntl 2 - Software input transition detected Image: Control end of the short-circuit Image: Control end of the short-circuit Details ntl 3 - Control end of the short-circuit Image: Control end of the short-circuit Image: Control end of the short-circuit Details ntl 0 - Time period has elapsed Image: Control end of the short-circuit It 1 - Event set has occurred Image: Control end of the short-circuit Image: Control end of the short-circuit Image: Control end of the short-circuit It 2 - Position trigger 14 change detected Image: Control end of the short-circuit Image: Control end of the short-circuit Image: Control end of the short-circuit It 2 - Position trigger 14 change detected Im	nt0 - Enable input has changed			Г	
12 - Software protections Image: Control error Image: Control error 13 - Control error Image: Control error Image: Control error 14 - Communication error Image: Control error Image: Control error 14 - Control error Image: Control error Image: Control error 14 - Control error Image: Control error Image: Control error 14 - Control error Image: Control error Image: Control error 14 - Software programmed transition detected Image: Control error Image: Control error 14 - LSN programmed transition detected Image: Control error Image: Control error 14 - Software input transition detected Image: Control error Image: Control error 14 - Software input transition detected Image: Control error Image: Control error 14 - Software input transition detected Image: Control error Image: Control error 14 - Software input transition detected Image: Control error Image: Control error 14 - Event set has occurred Image: Control error Image: Control error 14 - Position trigger 14 change detected Image: Control error Image: Control error	nt1 - Short-circuit	Γ	Г	Г	
nt3 - Control error	nt2 - Software protections	Г	Г	Г	Details
Intl - Communication error Imst - Position wraparound Imst - Sup programmed transition detected Imst - Sup progra	Int3 - Control error	Г	Г	Г	
Int5 - Position wraparound	Int4 - Communication error	Г	—	F	
Int8 - LSP programmed transition detected Imit P Imit P Imit P Details Int7 - LSN programmed transition detected Imit P Imit P Details Int8 - Capture input transition detected Imit P Imit P Details Int9 - Motion is completed /in velocity Imit P Imit P Imit P Int10 - Time period has elapsed Imit Imit P Imit P Imit P Int11 - Event set has occurred Imit P Imit P Imit P Int12 - Position trigger 14 change detected Imit P Imit P	nt5 - Position wraparound	Ē	E	Г	
Int7 - LSN programmed transition detected IM7 - LSN programmed transition detected IM8 - Capture input transition transiti	Int6 - LSP programmed transition detected				Details
Int8 - Capture input transition detected	nt7 - LSN programmed transition detected			E [Details
Int9 - Motion is completed / in velocity T T T Details Int10 - Time period has elapsed IX T Details Int11 - Event set has occurred T T T Int12 - Position trigger 14 change detected T T T T	Int8 - Capture input transition detected	Π		Г	Details
Intl0 - Time period has elapsed Image: Constraint of the second	int9 - Motion is completed / in velocity	Г	Г	Г _	
Int11 - Event set has occurred	nt10 - Time period has elapsed	X	Г	X	Details
Int12 - Position trigger 14 change detected T T T T	ht11 - Event set has occurred	Г		E F	1
OK. Cancel Help	Int12 - Position trigger 14 change detected	Γ	F	Г	

Figure 6. Interrupt Settings dialogue

4.2. Time period Interrupt routine

The "Interrupts" section allows 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.

Project				
🖃 🌇 APN - TML interrupt routines using		Interrupt	ts	
🖃 📳 Untitled Application	int0 - Enable input has changed	Default	OUser defined	Reload default
S Setup	int1 - Short-circuit	Default	O User defined	Reload default
M Motion	int2 - Software protections	Default	OUser defined	Reload default
Homing Modes	int3 - Control error	Default	OUser defined	Reload default
E Functions	int4 - Communication error	Default	OUser defined	Reload default
	int5 - Position wraparound	Default	OUser defined	Reload default
int10 - Time period bas elansed	int6 - LSP programmed transition detected	Default	OUser defined	Reload default
a CAM Tables	int7 - LSN programmed transition detected	Default	OUser defined	Reload default
CAW Tables	int8 - Capture input transition detected	Default	OUser defined	Reload default
	int9 - Motion is completed / in velocity	Default	OUser defined	Reload default
	int10 - Time period has elapsed	O Default	User defined	Reload default
	int11 - Event set has occurred	Default	OUser defined	Reload default
	int12 - Position trigger 14 change detected	Default	OUser defined	Reload default

Figure 7. How to customize 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.

The code sequence inside the "Int10 – Time period has elapsed" interrupt was generated using the buttons marked with 1 to 3 in Figure 3. Clicking on those buttons the following programming dialogues will open.

• The "Jumps and Function Calls" dialogue (1) 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 on 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).

Jumps and Function Calls	?	×	Jumps and Function Calls	?	×
Gota At address, label or at address, label at address set in variable C Cancelable Call C RETurn from function C RETurn from pterupt C Abort cancelable call C Insert label named.	D <u>N</u> EQ DLE <u>Q</u> H D <u>G</u> EQ	nan O	C Goto at address, label C Gall or C Cancelable Call it variable C RETurn from function IF it variable C RETurn from interrupt △ Abort cancelable call		than 0
OK Cancel Help			OK Cancel Help		



The same dialogue was used to create the "LED_ON" and "LED_OFF" labels.

Jumps and Function Calls ? X	Jumps and Function Calls ? X
Goto at address, label or at address, label or at address, label or at address, label or at address are in variable Cancelable Call Cancelable call Cancel DK Cancel Help	C goto at address, label C gall at address, label at address, label at address, label C Gancelable Call if variable C RETurn from junction if variable C Abort cancelable call if wariable C Insert label named LED_OFF OK Cancel

Figure 9. Create the "LED_ON" / "LED_OFF" label

• 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.

1/0 ×	1/0 ×
Single I/0 Read an input Read input line into variable	Read an input Read an input C Read input line into variable
Set an output Set output line OUTO status C low C high Set I/D line type Set as output	Set an output Set output line OUTO status I low Chigh Set I/O line type Set as output
C Read input(s) 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 into variable	C Read input(s) 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 into variable
C Set output(s) 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 C as High C <t< th=""><th>C Set output(s) 15 14 13 12 11 10 3 8 7 6 5 4 3 2 1 0 C as High C <t< th=""></t<></th></t<>	C Set output(s) 15 14 13 12 11 10 3 8 7 6 5 4 3 2 1 0 C as High C <t< th=""></t<>
OK Cancel Help	OK Cancel Help

Figure 10. How to set output line status

• Once the OUT(0) digital output status is changed, the "Assignment and Data Transfer – 16 bit Integer Data" dialogue (3) 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").

Assignment & Data Transfer - 16 bit Integer Data	?	×	Assignment & Data Transfer - 16 bit Integer Data	?	Х
Set 16-bit variable LED					
With value / 16 bit variable / label			With value / <u>1</u> 6 bit variable / label		
C With C data C program C E2ROM I then increment the pointer variable			C With C data memory contents, located at address set in pointer variable C E2ROM T then increment the pointer variable		
C With C low part of 32-bit variable			C With C jow part of 32-bit variable		
C With the inverse (-) of variable			C With the inverse (-) of variable		
C Using AND mask h and OR mask h			C Lising AND mask h and OR mask h		
C With checksum C data C program C data located in C E2ROM and	h h		C With checksum C data C program C E2ROM and	h h	
C Set C data C program C E2ROM Memory contents, located at address set in pointer variable with value/variable then increment the pointer variable			C Set C data C program C E2ROM with value/variable		
			OK Cancel Help		

Figure 11. Variable's value corresponding to the LED ON / OFF status

• After the "OUT(0)" digital output state is changed and the "LED_status" variable is set accordingly, the program should return from the interrupt. This is done using the "RETI;" (return from interrupt) instruction. It can be inserted from the "Jumps and Function Calls" dialogue (1).

<u>**Remark**</u>: In in the second case (when the LED is switched off), the "RETI" instruction is not used anymore because the program returns naturally to the "Motion section" (the interrupt routine ends after "LED_status = 0" instruction).

Jumps and Function	Calls	?	×
C <u>G</u> oto C <u>C</u> all C <u>Ca</u> ncelable Call C RET urn from <u>f</u> un RET urn from <u>inte</u> C <u>A</u> bort cancelable C <u>Insert label name</u>	at address, label or at address set in variable if variable is call c call	C NEQ C LEQ C GEQ	than O
	OK Cancel Help		

Figure 12. How to insert a "RETI" TML instruction

5. Application evaluation

This application requires to connect a LED to the "OUT(0)" digital output, according to the schematics in the drive user manual. If this is not possible, then the "2_Drive IO" control panel in EasyMotion Studio can be used.

ect Application Communi	cation View Control Panel Wind		OFT STAT STOP	
☞ ⊌ ☺ ฃ 😐 🖗 ೫	Project	Ctrl+J	📾 📅 📆 🙀 🕺	
	Command Interpreter	Ctrl+I		
	Logger	Ctrl+L		
	Multi-Axis Logger	Ctrl+A		
	Scope	Ctrl+E		
	Control Panel	>	1_Motion Status	Ctrl+1
	Memory	Ctrl-M	2_Drive IO	Ctrl+2
	Output		3_CANopen Status	Ctrl+3
	V Toolbar		4_Drive Status	Ctrl+4
	🗸 Status Bar			
	View Graph Plot			
Digita	e IO	Digital Outp	J puts	
Piez 2_Driv Digita Genera N0 H L	e IO 1 Inputs 1 purpose Enable G IN1 H H H H H C C C C C C C C C C C C C C	Digital Outp ieneral purpos UTO DUT1 OUT T H C H C I C L C L C I	See Error Ready H C L C H L C L C L	b
Pigita Genera N0 ↓ ↓	e IO al Inputs I purpose IN1 H H H H L L L L	Digital Outp ieneral purpos uro puri lour H C H C I C L C I Analog Inpu	Se Error Ready H OUT2/ OUT3/ H C L OUT2/ Its	D
Pigita Genera N0 ↓ Limit s	e IO al Inputs I purpose Enable IN1 H H H H L L L L witches Capture	Digital Outp ieneral purpos H O H O H C L C I Analog Inpu Feedback(AI	Duts Se Error Ready I4 0UT2/ 0UT3/ H ⊂ L ⊂ H C L ⊂ L Its D2) Reference(AD5)	B
Limit s	e IO 1 Inputs 1 purpose Enable IN1 H H H H L L L L witches Capture LSN Indx1 Indx2	Digital Outp ieneral purpos UTO DUT1OUT H C H C I C L C I Analog Inpu Feedback(At	Error Ready From Ready CH CH CH CH<	Ĩ ₽ 0
Limit s	e IO 1 Inputs 1 purpose Enable IN1 IN4 H H H L L L L witches Capture IN3/LSP Indx1 Indx2 H H H H L L	Digital Outp ieneral purpos UTO DUT1 OUT H H H H I I L C L C I Analog Inpu Feedback(AI 0.137 5 [] U 0	Duts se Error Ready I 0UT2/ OUT3/ H L C L C H C L Its D2) Reference(AD5) 3.53 5 M 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0	0

Figure 13. How to insert a "RETI" TML instruction

During the program execution the "OUT(0) digital output will switch High (H) and Low (L) each 0.5 respecting the algorithm presented in the previous chapter.