TG drives

TGMcontroller

- Control system for motion control -

Document revisions:

REVISION	COMMENT
02/2023	Initial document version

Note: Technical changes that improve the properties of the device may be made without prior notice!

Printed in the Czech Republic.

CONTENT

1 General information

As software and operating systems evolve, it is becoming increasingly difficult to run real-time extensions on PCs. On the other hand the communication speeds are more than sufficient using standard Ethernet gigabit connection. An external device with a powerful real-time processor and a fast connection can easily replace the hard to maintain software extension to an operating system. The **TGMcontroller** device perfectly meets the above mentioned requirements and features.

2 Features

The **TGMcontroller** implements the complete TGMotion control system – version 502 or higher. It works as EtherCAT master and through the fieldbus supports up to:

- 64 servo axes
- 16 I/O units
- 16 additional user defined devices
- 3 independent CNC interpolators, each with 10 axes.

The oscilloscope memory has size of 32 MB, which allows storing information about a very large process. The virtual PLC programs are programmed in C or C++ language, complete gnu tool chain is available. PLC development can be done in any popular IDE system, like Visual Studio or VS Code.

The minimal cycle time is 100 µs.

EtherCAT master has extremely low jitter of several nanoseconds. It means that the EtherCAT packets are sent to the fieldbus very precisely in time domain and allow controlling the motion process with very high quality.

Full set of communication and diagnostics tools are available – Windows and partially Linux.

The TGMotion control system is described in its own manual.

3 Hardware

The system is based on the proven and powerful FPGA Zynq processor system – a double core Cortex A9 processor running at 667 MHz. The device has 128 MB of RAM memory and 16 MB of flash storage (used for the safety golden firmware version). The actual firmware, configuration, and user-defined PLC are stored on an 8 GB SD card. Communication with other devices can be done by 3 Ethernet ports and/or CAN bus. The system has 10 digital inputs, 6 digital outputs and 2 analog inputs. There are also 3 feedback connectors for a movement monitoring. The supported digital interfaces are Hiperface DSL, EnDAT 2.2, SSI and incremental encoder IRC.

3.1 Connectors

3.1.1 X1 - Power supply (AUX supply)

X1 - AUX SUPPLY VOLTAGE	
pin	signal
EI1	Extended input 1 (could be also labeled as STOA)
EI2	Extended input 2 (STOB)
+24	+24 V Power supply voltage
N.C.	Not connected (STOC)
GND	GND (0 V)

Extended inputs works as standard digital inputs 0 – 24 V (see also below about digital I/O mapping).

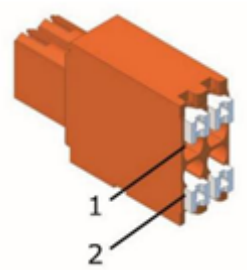
Used connector type is 5pin WEIDMÜLLER SC 3.81/05.

3.1.2 Ethernet connectors

- **X11** – It is used for connecting the TGMotion system (TCP or UDP protocol) by Control Observer and other user applications. Additional supported protocols are Modbus/TCP and Profinet IO (see below).
- **X12** – This port is used for so called Fast Service Port. Serves for very fast peer-to-peer connection between **TGMcontroller** and PC. Special custom raw protocol is used. The PC must install the winpcap or npcap driver (www.winpcap.org or www.npcap.com). No setting is necessary, the communication DLL finds the correct PC network adapter. For best performance, use the built-in or PCIe NIC adapters. The USB-Ethernet adapters can be also used, but suffer of worse performance. Some low cost USB to Ethernet adapters don't work at all with the FSP protocol. The adapter must have space for at least 32 packets at one time.
- **X13** – EtherCAT master connector – use this port for connecting the EtherCAT devices in the EtherCAT fieldbus. No setting is necessary. The port is capable of 100 Mbit or 1 Gbit speeds, depending of the first connected device. There are a few 1 Gbit EtherCAT devices available, e.g. TGZ servo drives or several EtherCAT bridges from other manufacturers. If using the 1 Gbit fieldbus, all the devices in the chain must be using the 1 Gbit speed.

Used connectors type are standard RJ45.

3.1.3 CAN bus connector X10

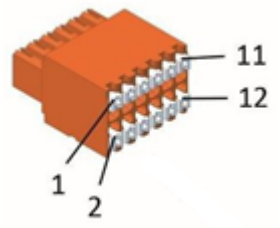


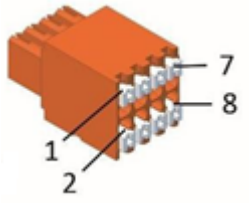
X10 – CAN CONECTOR	
pin	signal
1	CAN-H
2	CAN-L
3	CAN GND
4	N.C.

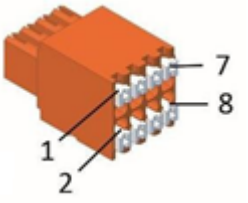
Note that there is no 120 Ω terminator resistor.

Used connector type is 4pin WEIDMÜLLER S2C-SMT 3.50.

3.1.4 Feedbacks







X5 - Encoder	
Pin	Incremental encoder
1	GND
2	+5V
3	Z-
4	Z+
5	N.C.
6	N.C.
7	A-
8	A+
9	B-
10	B+
11	GND
12	+12 V

X7 – Feedback 2		
pin	Hiperface DSL	SSI
1	FBSEL-	N.C.
2	FBSEL+	N.C.
3	FBSEL-	DATA-
4	FBSEL+	DATA+
5	N.C.	CLK-
6	N.C.	CLK+
7	DSL-	GND
8	DSL+	+12 V

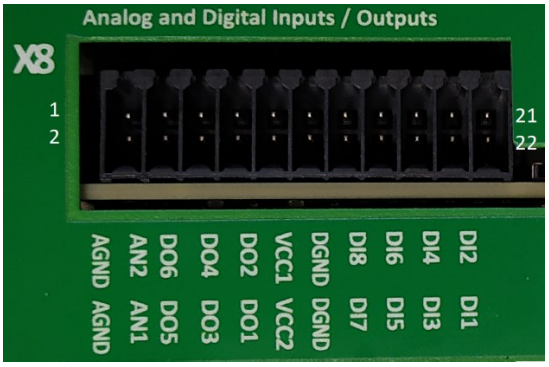
X6 – Feedback 1		
pin	Hiperface DSL	EnDat 2.2
1	FBSEL-	N.C.
2	FBSEL+	N.C.
3	FBSEL-	DATA-
4	FBSEL+	DATA+
5	N.C.	CLK-
6	N.C.	CLK+
7	DSL-	GND
8	DSL+	+12 V

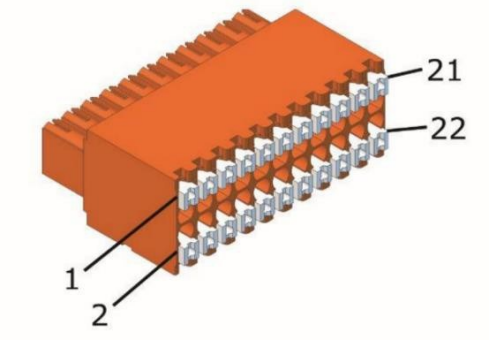
For DSL, pins 1 – 3 and 2 – 4 must be connected together.

TGMcontroller supports the following feedbacks:

- The **X5** connector is used for an incremental encoder (IRC). Connector type is 12pin WEIDMÜLLER S2C-SMT 3.50.
- The **X6** connector supports either Hiperface DSL feedback or EnDAT 2.2 standard. The type of the used feedback communication standard is given in the TGM.INI file (see also below): `Servo[xx].FeedbackType=1` is for **DSL** and `Servo[xx].FeedbackType=2` is for **EnDAT**. Connector type is 8pin WEIDMÜLLER S2C-SMT 3.50.
- The **X7** connector can be used with DSL or SSI feedback. Use the following settings in the TGM.INI file: `Servo[xx].FeedbackType=1` is for **DSL** and `Servo[xx].FeedbackType=3` is for **SSI**. Connector type is 8pin WEIDMÜLLER S2C-SMT 3.50.

3.1.5 Inputs and outputs





X8 – DIGITAL INPUTS / OUTPUTS ANALOG INPUTS	
pin	signal
1	ANALOG INPUT GND
2	ANALOG INPUT GND
3	ANALOG INPUT 2
4	ANALOG INPUT 1
5	DIGITAL OUTPUT 6
6	DIGITAL OUTPUT 5
7	DIGITAL OUTPUT 4
8	DIGITAL OUTPUT 3
9	DIGITAL OUTPUT 2
10	DIGITAL OUTPUT 1
11	VCC DO 1-3
12	VCC DO 4-6
13	DIGITAL GND
14	DIGITAL GND
15	DIGITAL INPUT 8
16	DIGITAL INPUT 7
17	DIGITAL INPUT 6
18	DIGITAL INPUT 5
19	DIGITAL INPUT 4
20	DIGITAL INPUT 3
21	DIGITAL INPUT 2
22	DIGITAL INPUT 1

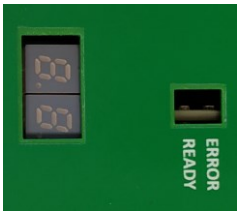
Analog inputs are 0 – 10 V.

Digital inputs are 24 VDC, low level 0 – 10 V, high level 12 – 24 V, 20 mA.

Digital outputs are 5 – 24 VDC (depending on VCC DO voltage), 300 mA per output.

Used connector type is 22pin WEIDMÜLLER S2C-SMT 3.50.

3.2 LED and status display



The LED display shows the device IP address during startup. The text “IP” is followed by numbers. Since the numbers can have up to 3 digits, each complete number is separated by a dot. For example to display the number “192”, first a single “1” is displayed (without a dot) and then “92.” (with the dot). Complex IP addresses like 192.168.128.179 can be displayed in this way.

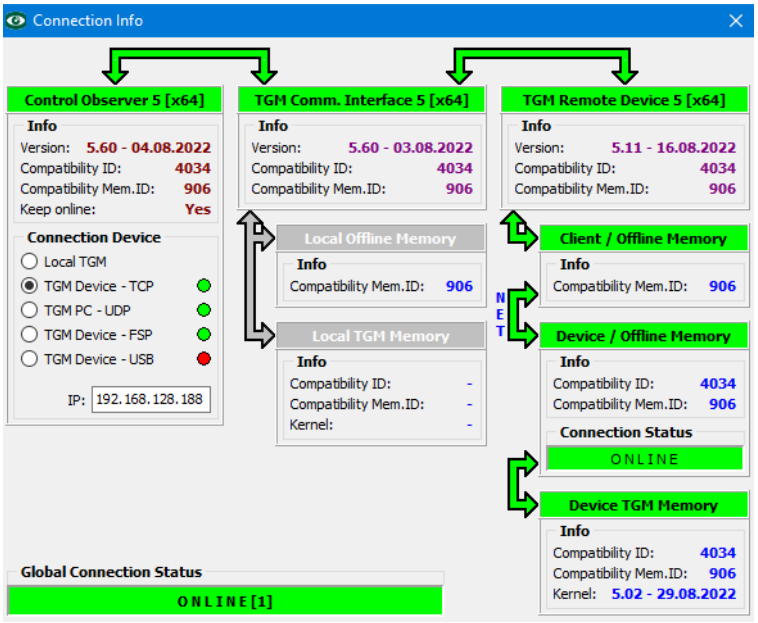
There are four additional LED diodes labeled as ERROR (red color) and READY (green color) for two axes. The READY shows that this particular axis is enabled in the TGM.INI file, on the other hand ERROR means the axis is not mapped to TGMotion by the TGM.INI file. See also below discussion about the entries in the TGM.INI file.

4 Software

The **TGMcontroller** is in the fact the complete real-time TGMotion control system in the box. Detailed description of the TGMotion is given in its own manual. There are no functional differences between TGMotion on the PC and on the **TGMcontroller**. The virtual PLC can be developed and tested completely on PC if necessary and then only compiled for the **TGMcontroller** and run on it without any additional changes.

4.1 Control Observer

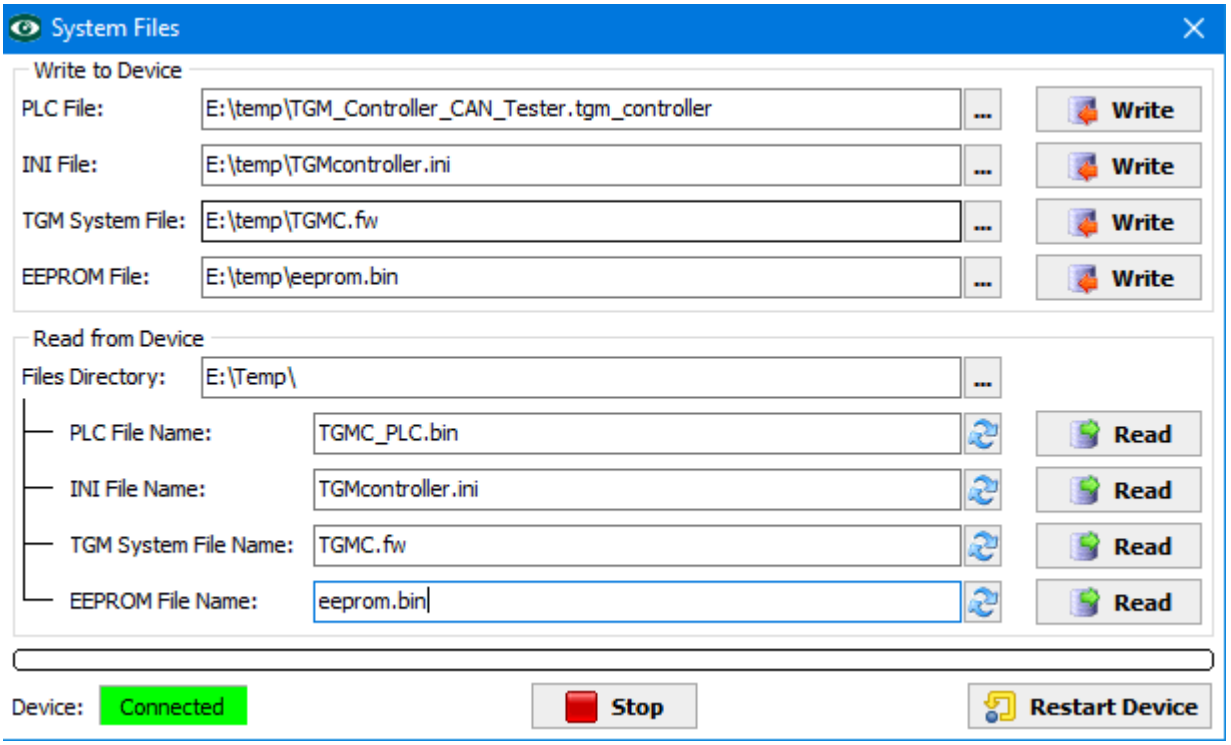
The Control Observer is a service and monitoring tool running on a Windows PC and plays a big role in the maintenance, firmware update, PLC development and TGMcontroller commissioning. First of all it must be connected to the device. Use the Connection Info window for that:



Select the wanted connection type in the Connection Device group box. Use TCP or UDP connection if the IP address is already known (displayed during startup in the LED display) and valid within an Ethernet segment. Use the FSP type if the Ethernet cable is connected from PC directly (peer to peer) to the FSP (X12) connector. Wait for the connection status to be ONLINE. Then all the features are available and the Control Observer works directly with the device.

4.2 Configuration file TGM.INI

The TGM.INI file is stored on the SD card. It can be edited directly on the card by PC or it can be read from the device by the Control Observer, modified and then written back. Use the System Files window, adjust the names in the edit boxes accordingly and use [Read] and then [Write] buttons (INI file rows). The file names on the PC can be arbitrary, **TGMcontroller** internally uses the correct SD card's file names.



After the TGM.INI file modification, it is necessary to restart the **TGMcontroller** (by Observer's button [Restart Device] or by power off/on sequence).

The internal feedbacks and inputs/outputs must be mapped to the TGMotion in the TGM.INI file if necessary. The values are visible in the SERVO structure. Use the servo type 92:

```
[Servo_Configuration]

Servo[00].Type=92
Servo[00].Node=1
Servo[00].Axis=1
Servo[00].FeedbackType=1

Servo[01].Type=92
Servo[01].Node=1
Servo[01].Axis=2
Servo[01].FeedbackType=1
```

The Node value is irrelevant and is ignored, but should be unique in the INI file and in the range of 1 – 64. The Axis=1 supports FeedbackType 1 (DSL) and 2 (EnDAT), while Axis=2 supports FeedbackType 1 (DSL) and 3 (SSI). Other values are invalid. If the FeedbackType is not specified, value 1 (DSL) is used. The indexes in square brackets (00 or 01 above) can be of course different, in the range of 0 – 63.

Actual position values from feedbacks are stored to the SERVO[xx].Position variable (TGMotion SERVO shared memory, accessible by PLC or PC), where the "xx" is the index from the TGM.INI file above. Value from the incremental encoder (connector X5) is always mapped to axis 1, variable SERVO[xx].ExtPosition.

Likewise, the digital inputs and outputs are found in the SERVO[xx].DigitalIn and SERVO[xx].DigitalOut variables. The table below shows the used inputs mapping. It assumes the TGM.INI entries as given above (SERVO[00].Axis=1 and SERVO[01].Axis=2).

Input pin	Axis	Appears at bit	Code example
DI1	1	DigitalIn.0	`SERVO[00].DigitalIn & 0x001`
DI2	2	DigitalIn.0	`SERVO[01].DigitalIn & 0x001`
DI3	1	DigitalIn.1	`SERVO[00].DigitalIn & 0x002`
DI4	2	DigitalIn.1	`SERVO[01].DigitalIn & 0x002`
DI5	1	DigitalIn.2	`SERVO[00].DigitalIn & 0x004`
DI6	2	DigitalIn.2	`SERVO[01].DigitalIn & 0x004`
DI7	1	DigitalIn.3	`SERVO[00].DigitalIn & 0x008`
DI8	2	DigitalIn.3	`SERVO[01].DigitalIn & 0x008`
EI1	1	DigitalIn.8	`SERVO[00].DigitalIn & 0x100`
EI2	1	DigitalIn.9	`SERVO[00].DigitalIn & 0x200`

Outputs mapping is similar:

Output pin	Axis	Used bit	Code for set	Code for clear
DO1	1	DigitalOut.0	`SERVO[00].DigitalOut |= 0x01`	`SERVO[00].DigitalOut &= ~0x01`
DO2	2	DigitalOut.0	`SERVO[01].DigitalOut |= 0x01`	`SERVO[01].DigitalOut &= ~0x01`
DO3	1	DigitalOut.1	`SERVO[00].DigitalOut |= 0x02`	`SERVO[00].DigitalOut &= ~0x03`
DO4	2	DigitalOut.1	`SERVO[01].DigitalOut |= 0x02`	`SERVO[01].DigitalOut &= ~0x03`
DO5	1	DigitalOut.2	`SERVO[00].DigitalOut |= 0x04`	`SERVO[00].DigitalOut &= ~0x04`
DO6	2	DigitalOut.2	`SERVO[01].DigitalOut |= 0x04`	`SERVO[01].DigitalOut &= ~0x04`

Analog inputs values AN1, AN2 appears in the SERVO[00].AnalogIn and SERVO[01].AnalogIn variables.

4.3 Communication with PC

TGMcontroller uses two gigabit Ethernet ports for communicating with outer world. The so called service port (**X11**) implements standard protocols: TCP, UDP, Profinet I/O and Modbus TCP. All these protocols can be used simultaneously if desired. The important role for these protocols plays the IP address (see below). This service port can be used in larger networks where Ethernet switches, routers or bridges are necessary. Up to 16 external clients can be connected at the same time.

As the **TGMcontroller** can be also used as a replacement for real-time extension of a PC, the Fast Service Port (**X12**) is implemented. It uses a custom protocol and should be used only for peer to peer Ethernet connection, i.e. a direct cable between PC and **TGMcontroller**. The FSP can be characterized by a very fast response time and allows very high data bandwidth. It does not use any IP address and therefore is very easy to setup – just plug the Ethernet cable to X12 FSP port and to any free Ethernet NIC adapter on a PC (preferably a built-in adapter on the motherboard or a PCIe adapter). The supporting communication Control Observer's DLL (TGM_DEV_FSP_5.dll or TGM_DEV_FSP_5_x64.dll) scans all the available PC's adapters and choose the connected one during startup. The disadvantage is that a winpcap (www.winpcap.org) or npcap (www.npcap.com) driver must be installed on the PC.

4.4 IP address

The default IP address is 192.168.1.188. It can be modified in the TGM.INI file:

```
[DHCP]
Enable=0
IPAddress=192.168.1.188
```

```
Mask=255.255.255.0
Gateway=192.168.1.1
DNS=8.8.8.8
```

After changing the IP address, be sure to set also the right Gateway address in the same Ethernet segment.

It is also possible to use DHCP for automatic IP address assignment (in the networks with a router). Setting Enable=1 activates this feature. The assigned IP address is displayed on the LED display when the DHCP process succeeds. Note that it could take some time. If the DHCP is not successful (after timeout of several tenths of seconds), the static IP address from TGM.INI file is used instead.

4.5 MAC address

TGMcontroller reads its internal unique DNA value and invents a MAC address from it. The automatic MAC address always begins with 00:0A. The value can be overridden by an entry in the TGM.INI file in the [DHCP] section:

```
[DHCP]
…
MacAddress=00.xx.xx.xx.xx.xx
```

The first number must be always 00, otherwise the entry is ignored. Other values “xx” can be as desired. Note that hexadecimal numbers are used.

4.6 PLC development and upload

The source code of the TGMotion’s PLC is always compatible with all the TGMotion derivatives (Windows PC with real-time extension, TGMmini, TGZ+Motion, **TGMcontroller** and possible future systems). It means that the PLC can be programmed and tested e.g. on PC and then just recompiled for the **TGMcontroller** system. The PLC must be written in C/C++ language.

The PLC program can be compiled on any PC with Windows or Linux. All needed applications are open source. A cross compiler is necessary, it must be GCC compiler called gcc-arm-none-eabi. The latest version can be downloaded from ARM website (developer.arm.com at the time of writing this manual). Choose the latest ZIP version and extract it to any folder. The compiler must support at least the C++17 standard.

The project for PLC uses the GNU standard makefile. It is necessary to use the mingw32-make program to execute the makefile commands. The mingw32-make can be downloaded e.g. from sourceforge.net and must be accessible from command line (e.g. by adding its path to the PATH system variable or using its filename together with the path on the command line). Note that there is no need to have the GCC compiler in PATH.

The makefile has almost fixed structure, only the list of source/header files and the path to the compiler is necessary to edit or modify. When the makefile is correct, the PLC can be created by command

```
mingw32-make all
```

executed in the directory where the makefile and source files are located. Optionally use the full path of the mingw32-make.exe file, like

```
C:\PLC\projects\gnu\mingw32-make all
```

To rebuild the PLC, first clean the intermediate files by

```
mingw32-make clean
```

and then build it by

```
mingw32-make all
```

Use the –j<number> option to compile it using more CPU cores, e.g.

```
mingw32-make –j4 all
```

(Uses four threads for project compile, speeding up the build process.)

Example of the makefile:

```
#path to TGM include files
TGM_PATH=..\include
#path to compiler
CC_PATH=..\..\gnu\gcc-arm-none-eabi\bin

#source files, must be in the same directory as this makefile
SOURCES=main.cpp                \
        DI_Capture.cpp          \
        Program_01.cpp          \
        Program_02.cpp          \
        Program_03.cpp          \
        Program_04.cpp          \
        Program_Ini.cpp         \
        Servo_Func.cpp          \
        $(TGM_PATH)\fmt\format.cpp

#used header file. these files are checked for dependencies
HEADERS=Definition.h             \
        DI_Capture.h             \
        PLC_Func.h               \
        Servo_Func.h             \
        stdafx.h                 \
        User_Definitions.h       \
        User_Variables.h


USER_PATH=.\
#set the output filename
OUT_FILE_NAME=TGM_Controller_CAN_Tester

#================================
#there is no need to change the rest of makefile


#compiler file names
CC=$(CC_PATH)\arm-none-eabi-g++.exe
OBJCOPY=$(CC_PATH)\arm-none-eabi-objcopy.exe
OBJDUMP=$(CC_PATH)\arm-none-eabi-objdump.exe
#stamper is special utility to mark the resulting PLC as for TGMcontroller
STAMPER=tgm_bin_stamper.exe


OPTIMIZE=-O2

BSP_PATH=..\BSP

EXECUTABLE=$(OUT_FILE_NAME).elf
BIN_FILE=$(OUT_FILE_NAME).tgm.bin
```

```
STAMPED_FILE=$(OUT_FILE_NAME).tgm_controller
DUMP_FILE=$(OUT_FILE_NAME).elf.dbg

MAX_SERVO=64
MAX_DIO=16
MAX_INTERPOLATOR=3
SIZE_OSCILLOSCOPE=33554432
TGM_SETTINGS=-DMAX_SERVO_PROJECT_SETTINGS=$(MAX_SERVO) \
             -DMAX_DIO_PROJECT_SETTINGS=$(MAX_DIO) \
             -DMAX_INTERPOLATOR_PROJECT_SETTINGS=$(MAX_INTERPOLATOR) \
             -DSIZE_OSCILLOSCOPE_MEMORY_PROJECT_SETTINGS=$(SIZE_OSCILLOSCOPE)

DEFINES=$(TGM_SETTINGS) \
        -DFREERTOS \
        -DZYNQ \
        -DNDEBUG \
        -DGLOBAL_TIMER_PRESCALER=1 \
        -DNO_FILE_FUNCTIONS

CFLAGS=$(OPTIMIZE) -std=c++17 -I. -I$(BSP_PATH)\include -I$(TGM_PATH) \
       -I$(USER_PATH) $(DEFINES) -g -Wall -Wextra -mcpu=cortex-a9 \
       -mfpu=vfpv3 -mfloat-abi=hard -ffunction-sections -fdata-sections \
       -Wno-unknown-pragmas
LDFLAGS=-g -mlittle-endian -Wl,-T -Wl,lscript.ld \
       -Wl,--start-group,-lxil,-lgcc,-lc,-lstdc++,--end-group \
       -L$(BSP_PATH)\lib -mcpu=cortex-a9 -mfpu=vfpv3 -mfloat-abi=hard \
       -Wl,-build-id=none -specs=Xilinx.spec -Wl,--gc-sections

OBJECTS=$(SOURCES:.cpp=.o)

$(EXECUTABLE): $(OBJECTS) $(SOURCES) $(HEADERS) makefile
       $(CC) $(OBJECTS) $(LIB_OBJECTS_C) $(LIB_OBJECTS_CPP) $(LIB_OBJECTS_ASM) \
        -o $@ $(LDFLAGS)
       cmd /C $(OBJDUMP) -C -x -S $(EXECUTABLE) >$(DUMP_FILE)
       $(OBJCOPY) -O binary $(EXECUTABLE) $(BIN_FILE)
       $(STAMPER) $(BIN_FILE) $(STAMPED_FILE) -oPLC -b0x07800000 -s0x00800000

%.o: %.cpp $(HEADERS) makefile | $(OBJDIR)
       $(CC) -c $< $(CFLAGS) -o $@

all: $(SOURCES) $(HEADERS) $(EXECUTABLE) makefile

clean:
       del /Q $(OBJECTS) $(EXECUTABLE) $(BIN_FILE) $(DUMP_FILE)
```

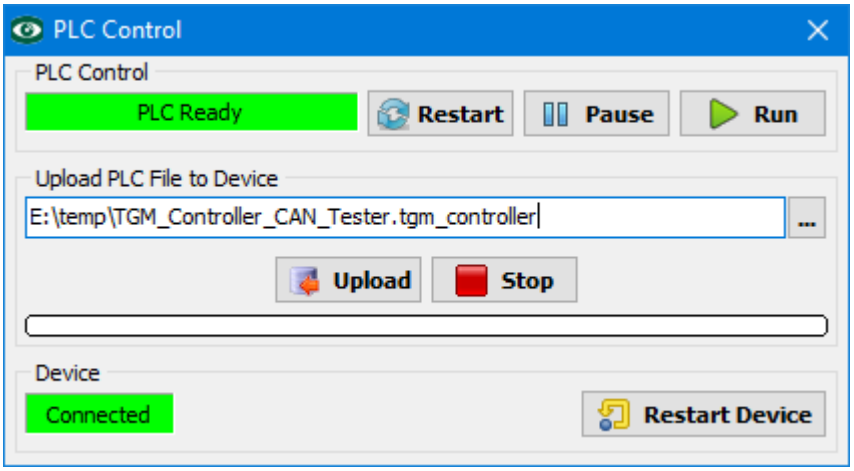
After creating executable .ELF file, the makefile creates a binary file and subsequently stamp it by a standalone utility tgm_bin_stamper.exe.

Ask TG Drives representatives for an example PLC program. There is also an example program on the SD card together with other auxiliary files. The project contains all the necessary files, compiler tools, linker script file, stamper program, BSP (board support package) and TGMotion public include files. The file main.cpp contains the necessary access function GetProcAddress() which is used to get placement addresses of the main PLC functions: Program_Ini(), Program_01(), Program_02(), Program_03(), Program_04(). The function GetProcAddress() must be placed at startup address 0x07800000 and must not be optimized. It also initializes the C standard library and all the dependencies like pointers to virtual functions and initialization of local and global variables.

See also the separate manual about PLC programing with TGMotion.

4.7 PLC start and autostart

In addition to Control Observer's System Files window, there is a dedicated one for PLC programming – PLC Control. Just set the correct PLC filename and Upload it to the TGMcontroller. The PLC is started by clicking the button [Run].



The Run PLC process performs the following sequence:

1. Stops any running PLC and wait for all the Program_XX() functions to finish.
2. Clear the PLC DATA memory.
3. Clear (set to zero) all the digital outputs of connected I/O devices as well as servo drives. Set the mode of all servo drives to zero.
4. Set also the internal digital outputs to zero (if they are mapped to TGMotion).
5. The main servo loop is stopped and no additional EtherCAT messages are transmitted.
6. The PLC is loaded to memory from SD card.
7. The main servo loop starts again, the EtherCAT communication is active.
8. The PLC initialization function, Program_Ini() is called and when succeeds (returns 1), the periodic call of Program_XX() functions is established, i.e. the PLC is started.

The PLC can be started after the TGMcontroller boots up. Set the TGM.INI file entry

```
[PLC_Configuration]
…
Autostart_PLC=1
```

There is no synchronization between PLC start and EtherCAT communication. Usually the PLC starts up faster than the EtherCAT communication. The programmer must count with it and wait for example for EtherCAT state value (operational = 8) of the attached devices, or the requested number of found EtherCAT devices on the fieldbus (variable SYSTEM.MAIN. Found_Total_Number_Of_ECAT_Devices).

4.8 PLC programs priority and cycle time

The programs in the PLC runs with different priority. It is up to programmer to choose the right function for PLC tasks.

1. **Program_04()** has the highest priority and runs in the context of servo cycle function. It is called

periodically at the interval given by `Cycle_Time` value in the TGM.INI file (in microseconds). The allowed values are 100, 200, 250, 500, 1000, 2000, and then 3000, …, 10000 (in 1000 steps). It is very important that the Program_04() does not last more than approx. half of the cycle time, otherwise TGMotion will not have chance to make its own calculations. The are no checks for too busy Program_04() function, but the problem appears very quickly on the EtherCAT side of the system: the desired servo positions will not be updated and a following error arises in the servo amplifier iteself. If it is absolutely necessary to do a heavy calculation in the Program_04() function, the only solution is to increase the cycle time.

2. **Program_03()** has a priority just one step below the Program_04() and likewise **Program_02()** two steps below Program_04(). The programs are called repeatedly with the cycle time given in the TGM.INI file. For example:

```
[PLC_Configuration]
…
Cycle_Time_Program_02=400
Cycle_Time_Program_03=800
```

The values are given in microseconds and must be a multiple of 200. The minimal allowed value is 200 µs. To disable the given function completely, use value of zero. TGMcontroller checks the elapsed time in each function, and if less than 100 microseconds remain before the next function call, the actual call will not occur until the next 200-microsecond tick. Thus TGMcontroller can schedule other lower priority tasks (communication, etc.) and the Program_02() or Program_03() cannot overload the system.

3. **Program_01()** runs at the lowest priority possible within the system. It means that other tasks, such as communication, Profinet handling, etc. will run with higher priority and will interrupt the Program_01() quite often. On the other hand, the Program_01() get all the remaining CPU time, which is usually more than 90 %. Additionally, the Program_01() is called as fast as possible (if the function is almost empty, the call interval is about 200 – 400 ns). This conception allows to implement any time consuming calculations to the Program_01() without the fear of the system overload. Note that there is no real-time behavior for the Program_01(). The cycle time in the TGM.INI file must be set to zero, no other value is allowed. The Program_01() is always called – it cannot be disabled.

```
[PLC_Configuration]
Cycle_Time_Program_01=0
```

The Control Observer's System Timers window shows the elapsed time of various TGMotion's tasks.

System Timers [microsecond]

Reset	Setting in	Actual	Maximum	Actual
Servo	500	8	25	
PLC 1	0	0	25	
PLC 2	400	0	9	
PLC 3	600	0	9	
PLC 4 (Servo)	500	0	1	
Interplolator 1	500	1	18	
Interplolator 2	500	0	0	
Interplolator 3	500	0	0	

Note that the maximal elapsed time of the Program_01() (labeled as PLC 1) includes also the time of all the

other tasks which interrupted the Program_01(). The similar thing happens for Program_02() which can be interrupted by Program_03() and the main servo cycle routine; and for Program_03(), interrupted by the main cycle function. So the maximal value could be the sum of elapsed time of the PLC function and the tasks of higher priority. The only exact time measurement is given for Program_04(), because it cannot be interrupted.

4.9 Profinet I/O device

Profinet I/O device is disabled by default. Use the following settings in the TGM.INI file:

```
[Profinet]
Enable=1
```

Each Profinet device is specified by its MAC address, IP address and device name. These attributes should be unique within the Profinet network. While the MAC address is fixed, the name and IP address can be set by a commissioning tool (TIA Portal, Proneta, etc.). The default TGMcontroller's Profinet name is empty and IP address is set to 0.0.0.0. The organization of the modules and slots is given by GSDML configuration file. There are nine slots: 1 × DAP, 4 inputs slots (size 32, 64, 128 or 256 bytes) and 4 output slots (with the same sizes). Each slot has an associated parameter which gives the data offset to PLC DATA memory. This parameter can be set by the commissioning tool or by Profinet PLC program.

Be aware that changing the IP address of the Profinet device also changes that address of the service port X11. It means that a possible loss of TCP/UDP or other communication is possible. The solution is to use the same address for service protocols and for Profinet or use the FSP (X12) port for direct communication with the PC, independently to the Profinet network.

4.10 Modbus TCP

The Modbus TCP protocol is disabled by default. To enable it, use the following TGM.INI configuration:

```
[Modbus]
Enable=1
```

The PLC DATA memory is used for Modbus data store or receive. The Modbus register number is used as an offset to the memory. Because the register numbers count by 1, there is a value multiplier available with the default value of 4. Also a global offset can be used to shift the Modbus data in the DATA memory (default value 0). It is recommended that the global offset is divisible by 4. The final position of the Modbus data is calculated by:

```
final_offset_to_data_memory = global_offset + (multiplier × modbus_register_number)
```

The global offset and multiplier can be set also in the TGM.INI file. For example:

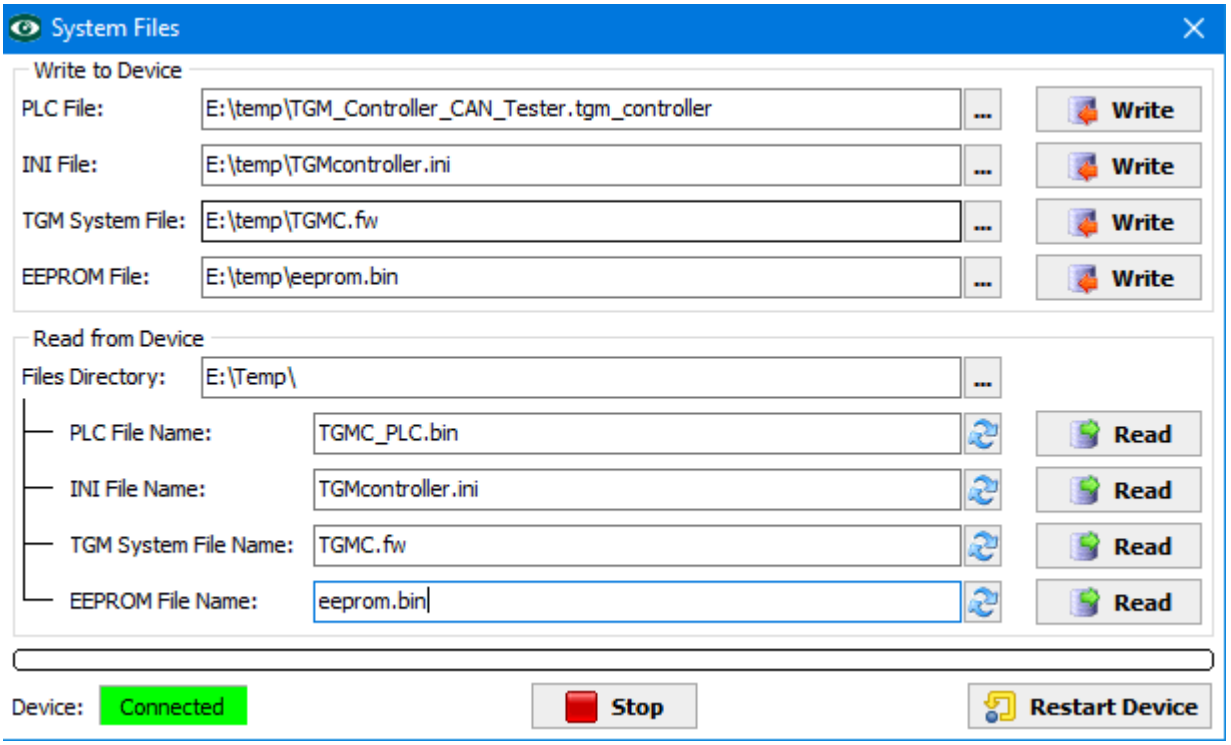
```
[Modbus]
Enable=1
Offset=16384
Multiplier=4
```

The following Modbus commands are supported:

READ_COILS	1
READ_DISCRETE_INPUTS	2
READ_HOLDING_REGISTERS	3
READ_INPUT_REGISTERS	4
WRITE_SINGLE_REGISTER	6
WRITE_MULTIPLE_COILS	15
WRITE_MULTIPLE_REGISTERS	16

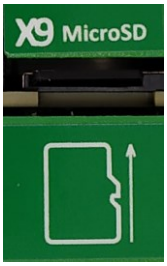
4.11 Firmware update

The firmware can be easily upgraded by Control Observer. Just select the correct file in the TGM System File edit box and use the [Write] button. Note that the file can be stored on the PC under any file name. **TGMController** uses the correct file name when saving the file to the SD card. After the file transfer, the device will be automatically restarted. The updated firmware is stored to the SD card. It is also possible to store the file directly to the SD card by PC, in that case the name must be TGMC.fw and the file must be in the root directory of the card. See also the next chapter about the SD card contents.



It could happen that a Device Offline error appears during the read back of the files from the device, especially when using FSP protocol. This usually means that used Ethernet adapter has not enough packet buffers (32 or more are recommended). The FSP protocol is designed for high performance adapters, namely the PCIe ones.

4.12 SD card contents



Two important and mandatory files with fixed names must be stored on the SD card:

- TGMC.fw – firmware file (in special binary format)
- TGM.INI – configuration text file

There are also possible additional files:

- TGM_PLC.bin – virtual PLC created by user
- Eeprom.bin – binary configuration file for PLC

All the four above files can be written to SD card by PC service program Control Observer. These files can be also read back. Since the SD card uses standard FAT32 file system, it can be also directly accessed by a PC and SD card reader. When the modified SD card is inserted back to the TGMcontroller, it must be restarted by Control Observer or by power off/on sequence. These file names are fixed.

4.13 Safe boot

TGMcontroller normally boots from SD card. In the case of faulty card or broken firmware, it is possible to start the device from internal flash memory. Remove SD card and restart the system. It takes a long time to start it without the SD card. The system loads a default INI file contents with the IP address 192.168.1.188. Use the FSP port if possible since no settings on the PC side is necessary. Insert a new SD card formatted to FAT32 and upload all the necessary files by Control Observer: TGM.INI file and then **TGMcontroller** firmware. The device will be automatically restarted and should boot correctly from the SD card.

The TGM.INI file is stored on the SD card in its natural text form, so it is also possible edit it directly by PC in the case of wrong settings. A restart of the **TGMcontroller** device is necessary after insertion of the SD card back to the device.