TTControl
HYDAC INTERNATIONAL

I/O Block Library for TTC 580

I/O Block Library for TTC 580 user manual

Version 0.27.3

Original Instructions

Document type:	user manual
Document number:	D-TTCSW-G-20-011
Document revision:	1.0.0
Document date:	2021-09-30
Security:	Corporate confidential
Status:	Released

TTControl GmbH
Schoenbrunner Str. 7, A-1040 Vienna, Austria,Tel. +43 1 585 34 34 – 0, Fax +43 1 585 34 34 – 90, office@ttcontrol.com

Table of Contents

1 Library Blocks

1.1 System Blocks

1.1.1 MainDlg

Block representing main environment-settings of the applications to be run on a TTC580

Input: -

Output:
ErrorCode (uint16): represents the return value of IO_Driver_Init (further details in driver manual)
Execution (uint32): Time needed to execute the application. Time output in us.

Parameters:
baudrate_CANX (CAN channel [0..2]): specify the baudrate of the CAN channel no. 0..2
In case the baudrate was chosen to be *I_CAN_BIT_USER* for a specific CAN channel out of [0..2] an user defined baudrate may be chosen by defining the following quantities:
tseg1 (range [3 ... 16]): Time segment before sample point
tseg2 (range [2 ... 8]): Time segment after sample point
brp (range [1 ... 64]): Baud rate prescaler
Note: Read the documentation of the function *IO_CAN_Init()* to obtain further information how to set the parameters *tseg1, tseg2, sjw, brp* in order define a specific baudrate.
baudrate_CANY (CAN channel no.[3..6]): specify the baudrate of the CAN channel no. 3..6
Duration_ms: specify in ms the cycle time of your application running on the TTC500.
Note: Duration_ms is only relevant for the code generation and is not linked to the fundamental sample time given in the Solver plane of the model configuration.
Show additional information: Check this box to make some additional settings concerning the APDB visible (further details in the driver manual of the TTC580)
Major, Minor, Revision Number: Versioning of the application, which will be visible in the APDB.
Node Number: This number is crusial for the TTC Downloader, in order to address a distict TTC500 on a CAN channel with multiple ECUs.
If there is oly one ECU on a CAN channel you may leave the default value.
Reset by TTC-Downloader: Check the box if you do not want to make a manual power cycle within the TTC-Downloader for flashing an updated program onto the TTC500
Note: For this purpose the standard CAN-ID 0x600 + Node-Number will be used. Make sure that you do not use this CAN-ID within your application, if you checked the 'Reset by TTC-Download' box.
Example: Supose your Node-Number is 15, then you may not use the CAN-ID 0x60F in your application.
Enable ethernet download requests: Activate alternative possibility for download
Note: This is only visible in case "Reset by TTC-Downloader" is not checked
Note further: Download over ethernet is only possible if the actual SW on the ECU has this option already activated.
Enable dedicated actions for error callback: Check the box if you want to define a dedicated action to take when a non-fatal error occurs. The checkboxes define which shut-off group(s) should be disabled upon non-fatal errors. If none of the boxes "Shut-off group 0-2" are checked, the ECU will enter safe state upon non-fatal errors, switching of all outputs.
Note: A list of non-fatal errors can be found in the IO-Driver manual chapter "Diagnostic state machine error"
Note further: If "Enable dedicated actions for error callback" is not checked, non-fatal errors will be ignored.

There are more parameters for the MainDlg block in the AddOn Libraries "CCP" and "J1939DMx" available.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.1.2 Power_Enable

Enables all power outputs

Input:
enable (boolean):
TRUE = activates IO_INT_POWERSTAGE_ENABLE and IO_INT_SAFETY_SW_0..2 in order to drive DO and PWM outputs
FALSE = all powered outputs (DO and PWM) are set to low state

Output: -

Parameters: -

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.1.3 Postrun_Enable

Enables postrun of the ECU

Input:
enable (boolean):
TRUE = independently from the state of KL15 the application keeps running
FALSE = if KL15 is low the ECU powers down

Output: -

Parameters: -

The ECU does not power down automatically when K15 is low. When K15 becomes low, the postrun is still active, and a power down can be now initiated by the application SW. Turning down the ECU is done by setting ”enable” to FALSE.

Further information of the mechanisms related to the behaviour and control of the Power IC of the ECU may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.1.4 EnterSafestate

Allows an application-driven safe state

Input:
enable (boolean): on setting TRUE, ECU activates the safe state. In this way, the ECU can be brought into the safe state from application conditions.

Output: -

Parameters: -

When this function is called the safe state is entered latest after 10ms or on the next call of *IO_ Driver_TaskEnd()*

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.1.5 FPU_Handler

This Block checks for exception reasons (listed below). The corresponding FPU exceptions lead to entering safe state

Input: -

Output: -

Parameters:
division_by_zero (checkbox): is caused if a divide operation has a zero divisor and a dividend that is not zero, an infinity or a NaN
input_denormal (checkbox): is caused if a denormalized input operand is replaced in the computation by a zero
invalid_operation (checkbox): is caused if the result of an operation has no mathematical value or cannot be represented
overflow (checkbox): is caused if the absolute value of the result of an operation, produced after rounding, is greater than the maximum positive normalized number for the destination precision.
underflow (checkbox): is caused if the absolute value of the result of an operation, produced before rounding, is less than the minimum positive normalized number for the destination precision and the rounded result is inexact.

Note: This block may only occur once per model. No code generated if placed in referenced models.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.1.6 SetDateAndTime

Sets the date and time of the real time clock

Input:
trigger (boolean): on rising edge of trigger input the following entities may be set to the real time clock
seconds (uint8): seconds (range [0..59])
minutes (uint8): minutes (range [0..59])
hours (uint8): hours (range [0..23])
days (uint8): days (range [1..31])
months(uint8): months (range [1..12])
years (uint8): years (range [0..99])

Output:
status (uint16): status of the external RTC device

Parameters: -

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.1.7 GetDateAndTime

Returns the date and time of the real time clock

Input:
trigger (boolean): date and time of the real time clock is returned on rising edge of trigger

Output:
Status (uint): corresponds to the result of IO_RTC_GetDateAndTimeStatus() in order to capture busy states or integrity faults of the RTC.
seconds (uint8): seconds (range [0..59])
minutes (uint8): minutes (range [0..59])
hours (uint8): hours (range [0..23])
days (uint8): days (range [1..31])
months(uint8): months (range [1..12])
years (uint8): years (range [0..99])

Parameters: -

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

Document Number:
D-TTCSW-G-20-011

1.1.8 DIAG_Status

Status function for diagnostic state machine and watchdog states

Input: -

Output:
ErrorCode (uint16): return value of internal driver function *DIAG_Status()*
diag_state (uint8): current state of the diagnostic state machine
watchdog_state (uint8): current state of the watchdog CPU
diag_error_code (uint8): error codes of the diagnostic state machine
diag_device_num (uint8): The device number which caused the error
diag_faulty_value (uint32): The value which caused the error
watchdog_error_code (uint8): error codes of the watchdog CPU
watchdog_device_num (uint8): device number which caused the error (of the watchdog CPU)
watchdog_faulty_value (uint32): value which caused the error (of the watchdog CPU)
error_count (uint8): watchdog CPU device error counter
b_FPGA_version_OK (boolean):
TRUE = the version number of the FPGA IP corresponds to the given values in the mask
FALSE = version conflict, please check the FPGA version of your ECU
b_IODrv_version_OK (boolean):
TRUE = the version number of the IO driver corresponds to the given values in the mask
FALSE = version conflict, please update the IO_Driver stack of your ECU

Parameters:
FPGA_Major (uint8): major version number of the FPGA
FPGA_Minor (uint8): minor version number of the FPGA
IODriver_Major (uint8): major version number of the IO_Driver
IODriver_Minor (uint8): minor version number of the IO_Driver
Example: when the IO_Driver version 3.1.17 please enter IODriver_Major 3 IODriver_Minor 1 and ignore the patch number (17)

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.1.9 compare2error

The ErrorCode that is output from an I/O-Block is compared against the given constant in mask (define)

Input:
ErrorCode (uint16): value coming from the ErrorCode output of another block from the IO block library

Output:
b_compare (boolean):
TRUE: value given in Mask and the input Errorcode are identical
FALSE: values are not identical

Parameters:
define (drop-down): define type of ErrorCode

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.1.10 IO_Driver_GetSerialNumber

Returns the ECU's serial number

Input: -

Output:
ErrorCode (uint16): possible Errorcodes from underlying driver function *IO_Driver_GetSerialNumber()*
seriealnumber (uint8): ASCII-Code of the serial number that may be found on the label of the ECU

Parameters: -

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.2 CAN Blocks

1.2.1 ReadMsg_CAN

Reads a message from a given message object.

Input: -

Output:
Read_Error (uint16): indication whether the message is new or not. Specifically:
IO_E_OK (= 0): everything is fine
IO_E_CAN_OVERFLOW (= 40): two or more messages received in the actual cycle
IO_E_CAN_OLD_DATA (= 45): no new data has been received in the actual cycle
Further ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_CAN_WriteMsg()*
data (uint32[2]): received data - 8 bytes stored in a two dimensional array of uint32

Parameters:
channel (drop-down) : CAN channel
id format (drop-down) : format of message identifier (standard / extended)
id (uint32): CAN message identifier

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.2.2 WriteMsg_CAN

Transmits a CAN message, using the given channel and message object.

Input:
data (uint32 [2]): 8 bytes, stored in a two dimensional array of uint32, which will be transmitted

Output:
Write_Error (uint16): indication whether the transmission has been started successfully or not. Specifically:
IO_E_OK (= 0): everything is fine
IO_E_BUSY (= 2) : no data has been transmitted. There is a problem with the CAN line
(eg: no acknowledge from any receiving node)
Further ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_CAN_WriteMsg()*

Parameters:
channel (drop-down) : CAN channel
id format (drop-down) : format of message identifier (standard / extended)
id (uint32): CAN message identifier
length (range[0..8]): length of data to be transmitted
Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

Document Number:
D-TTCSW-G-20-011

1.2.3 ReadMsg_CAN_derated

Representation of the ReadMsg_CAN function call, with derate functionality. This can be used for CAN Tx-messages with message cycle times larger than the application cycle time which can be defined in the mask of the MainDlg block.

Input: -

Output:
Read_Error_debcd (uint16): Passes the debounced ErrorCode of the CAN-driver to the application. When the driver ErrorCode are different from zero (IO_E_OK) twice or more times consecutively, the last ErrorCode is output to the application.
(eg: the ErrorCodes from IO_CAN_ReadMsg have the sequence 0, 45, 0
the output **Read_Error_debcd** is 0, 0, 0.
In case the sequence is 0, 45, 45, 0 the output **Read_Error_debcd** is 0, 0, 45, 0)
Some specific ErrorCodes:
IO_E_OK (= 0): everything is fine
IO_E_CAN_OVERFLOW (= 40) : two or more messages have been received in the actual message cycle time
IO_E_CAN_OLD_DATA (= 45) : no new data has been received in the actual message cycle time
Further ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_CAN_WriteMsg()*
data (uint32 [2]): received data, 8 bytes stored in a two dimensional array of uint32

Parameters:
channel (drop-down) : CAN channel
id format (drop-down) : format of message identifier (standard / extended)
id (uint32): CAN message identifier
derate (uint16) : The message buffer is only read out every n-th cycle, where n is given by this parameter. (Message cycle time) = derate * (application cycle time)
Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.2.4 WriteMsg_CAN_derated

Representation of the WriteMsg_CAN function call with derate functionality. This can be used for CAN Rx-messages with message cycle times larger than the application cycle time which can be defined in the mask of the MainDlg block.

Input:
data (uint32 [2]): 8 bytes stored in a two dimensional array of uint32 each which will be transmitted

Output:
Write_Error (uint16): indication whether the transmission has been started successfully or not.
Specifically:
IO_E_OK (= 0): everything is fine
IO_E_BUSY (= 2) : no data has been transmitted. There is a problem on the CAN line (eg: no acknowledge from any receiving node)
Further ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_CAN_WriteMsg()*

Parameters:
channel (drop-down) : CAN channel

id format (drop-down) : format of message identifier (standard / extended)
id (uint32): CAN message identifier
length (range[0..8]): length of input data
derate (uint16) : The message buffer is only sent out every n-th cycle, where n is given by this parameter. (Message cycle time) = derate * (application cycle time)
Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.2.5 bit_packing

Prepares CAN-transmition of a signal at a specified position within the CAN buffer.

Input:
Data_in (any native datatype): ***Physical value*** of the signal to be transmitted on the CAN bus

Output:
Data_out (uint32 [2]): CAN buffer filled with zeros except for the placeholder containing raw value

Parameters:
Endianness (drop-down): edianness of the signal within the CAN buffer
Format (drop-down): format of the signal within the CAN buffer
Startbit (range [0..63]): posititon of the placeholder for the signal in the CAN buffer
Width_bits (range [1..32]): size of the placeholder for the signal in the CAN buffer.
Needs to be big enough to hold the given datatype of **Data_in**
Factor: Parameter of the formula to calculate physical value
Offset: Parameter of the formula to calculate physical value

The following definition applies:
(Raw value) = ((***Physical value***) - (**Offset**)) / (**Factor**)
Raw value … the value of the signal as it is transmitted on the CAN-network.

Note: multiple bit_packing outputs may be merged using the bitwise operator OR_Tx

1.2.6 bit_unpacking

reads in CAN signal from a specified position within the CAN Buffer

Input:
Data_in (uint32 [2]): CAN buffer with the placeholder containing raw value

Output:
Data_out (double): ***Physical value*** of the signal to be transmitted on the CAN bus

Parameters:
Format (drop-down): format of the signal within the CAN buffer
Startbit (range [0..63]): posititon of the placeholder for the signal in the CAN buffer
Width_bits (range [1..32]): size of the placeholder for the signal in the CAN buffer.
Needs to be big enough to hold the given datatype of **Data_in**
Factor: Parameter of the formula to calculate physical value
Offset: Parameter of the formula to calculate physical value
The following definition applies:
(***Physical value***) = ((Raw value) * (**Factor**)) + (**Offset**)
Raw value … the value of the signal as it is transmitted on the CAN-network.´
Verification capabilities are supported.

1.3 ADC Blocks

1.3.1 ADC_Absolute_5V

Returns the value of the given ADC channel: 0V..5V

Input: -

Output:
ErrorCode (uint16) : Indication whether the ADC measurement is correct (=0).
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_ADC_Get()*
adc_value (uint32) : value of given ADC channel in mV (range [0..5000])

Parameters:
adc_channel (drop-down): ADC channel
The PIN numbers may be found in table below

Note: Pull resistor is fixed to pull-down

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.3.2 ADC_Absolute_10V

Returns the value of the given ADC channel: 0V..10V

Input: -

Output:
ErrorCode (uint16): Indication whether the ADC measurement is correct (=0).
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_ADC_Get()*
adc_value (uint32) : value of given ADC channel in mV (range [0..10000])

Parameters:
adc_channel (drop-down): ADC channel
The PIN numbers may be found in table below

Note: Pull resistor is fixed to pull-down

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.3.3 ADC_Absolute_32V

Returns the value of the given ADC channel: 0V..32V

Input: -

Output:
ErrorCode (uint16): Indication whether the ADC measurement is correct (=0).

ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_ADC_Get()*
adc_value (uint32): value of given ADC channel in mV (range [0..32000])

Parameters:
adc_channel (drop-down): ADC channel The PIN numbers may be found in table below

Note: Pull resistor is fixed to pull-down

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.3.4 ADC_Current

Returns the value of the given ADC channel: 0mA..25mA

Input: -

Output:
ErrorCode (uint16): Indication whether the ADC measurement is correct (=0).
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_ADC_Get()*
adc_value (uint32): value of given ADC channel in uA (range [0..25000])

Parameters:
adc_channel (drop-down): ADC channel
The PIN numbers may be found in table below

Note: An internal switch (FET) protects the ADC channel from damage. Reset of the FET protection after a certain time is performed automatically. The protection can be reset 10 times, afterwards the input will remain permanently protected, which is indicated by the ErrorCode.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.3.5 ADC_UBAT

Returns the value of the battery ADC channel: 0V..55 V

Input: -

Output:
ErrorCode (uint16) : Indication whether the ADC measurement is correct (=0).
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_ADC_Get()*
adc_value (uint32) : value of given ADC channel in mV (range [0..55000])

Parameters: -

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.3.6 ADC_Ratiometric_5V

Returns the value of the given ADC channel: 0V..5V
Use this configuration if the connected sensor is supplied by one of the sensor supplies

Input: -

Output:
ErrorCode (uint16) : Indication whether the ADC measurement is correct (=0).
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_ADC_Get()*
adc_value (uint32) : value of given ADC channel in mV (range [0..5000])

Parameters:
adc_channel (drop-down) : ADC channel
The PIN numbers may be found in table below
sensor_supply (drop-down) : sensor supply channel

Note: For ratiometric ADC measurements the sensor supply is switched on automatically, you don't need to insert the corresponding SensorSupply_5V block explicitly or configure the *SensorSupply_Modif* block

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.3.7 ADC_Ratiometric_10V

Returns the value of the given ADC channel: 0V..10V Use this configuration if the connected sensor is supplied by the variable sensor supply which can be configured in the range from 5V to 10V.

Input: -

Output:
ErrorCode (uint16) : Indication whether the ADC measurement is correct (=0).
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_ADC_Get()*
adc_value (uint32) : value of given ADC channel in mV (range [0..10000])

Parameters:
adc_channel (drop-down) : ADC channel
The PIN numbers may be found in table below
volt_sensor_supply (drop-down) : configuration of the variable sensor supply

Note: The sensor has to be supplied by IO_ADC_SENSOR_SUPPLY_2. For ratiometric ADC measurements the sensor supply is switched on automatically, you don't need to insert the corresponding SensorSupply_Var block explicitly.
Restriction: All ADC_Ratiometric_10V blocks in the application must use the same voltage level for SensorSupply_10V within their masks otherwise a code generation error is issued.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.3.8 ADC_Resistive

Returns the value of the given ADC channel: 0Ohm..100000Ohm
use this configuration to measure an resistive signal.

Input: -

Output:
ErrorCode (uint16) : Indication whether the ADC measurement is correct (=0).
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_ADC_Get()*
adc_value (uint32) : value of given ADC channel in Ohm (range [0..100000])

Parameters:
adc_channel (drop-down) : ADC channel
The PIN numbers may be found in table below

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.3.9 ADC_SensorSupply

Returns the value of the associated sensor supply ADC channel: 0V..10.560V

Input: -

Output:
ErrorCode (uint16) : Indication whether the ADC sensor supply measurement is correct (=0).
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_ADC_Get()*
adc_value (uint32) : value of given sensor supply ADC channel in mV (range [0..10560])

Parameters:
adc_channel (drop-down) : sensor supply channel
The PIN numbers may be found in table below

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.3.10 ADC_BoardTemp

Returns the value of the board temperature: -63°C .. 152°C

Input: -

Output:
ErrorCode (uint16) : Indication whether the ADC temperature measurement is correct (=0).
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_ADC_Get()*
Temperature (uint32) : bord temperature in degree Celsius (range [-63..152])

Parameters: -

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.3.11 ADC_K15_

Returns the value of the Terminal 15 Input channel: 0V..55V

Document Number:
D-TTCSW-G-20-011

Input: -

Output:
ErrorCode (uint16) : Indication whether the ADC measurement is correct (=0).
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_ADC_Get()*
adc_value (uint32) : value of Terminal 15 input channel in mV (range [0..55000])

Parameters: -

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.3.12 ADC PINs

Connector PIN	Pin aliases
PIN_103	IO_ADC_00
PIN_127	IO_ADC_01
PIN_104	IO_ADC_02
PIN_128	IO_ADC_03
PIN_105	IO_ADC_04
PIN_129	IO_ADC_05
PIN_106	IO_ADC_06
PIN_130	IO_ADC_07
PIN_107	IO_ADC_08
PIN_131	IO_ADC_09
PIN_108	IO_ADC_10
PIN_132	IO_ADC_11
PIN_109	IO_ADC_12
PIN_133	IO_ADC_13
PIN_110	IO_ADC_14
PIN_134	IO_ADC_15
PIN_111	IO_ADC_16
PIN_135	IO_ADC_17
PIN_112	IO_ADC_18
PIN_136	IO_ADC_19
PIN_113	IO_ADC_20
PIN_137	IO_ADC_21
PIN_114	IO_ADC_22
PIN_138	IO_ADC_23
PIN_115	IO_ADC_24
PIN_139	IO_ADC_25
PIN_116	IO_ADC_26
PIN_140	IO_ADC_27
PIN_117	IO_ADC_28
PIN_141	IO_ADC_29
PIN_122	IO_ADC_30
PIN_146	IO_ADC_31
PIN_123	IO_ADC_32
PIN_147	IO_ADC_33

PIN_124	IO_ADC_34
PIN_148	IO_ADC_35
PIN_149	IO_ADC_36
PIN_173	IO_ADC_37
PIN_152	IO_ADC_38
PIN_176	IO_ADC_39
PIN_155	IO_ADC_40
PIN_179	IO_ADC_41
PIN_158	IO_ADC_42
PIN_182	IO_ADC_43
PIN_251	IO_ADC_44
PIN_238	IO_ADC_45
PIN_252	IO_ADC_46
PIN_239	IO_ADC_47
PIN_253	IO_ADC_48
PIN_240	IO_ADC_49
PIN_254	IO_ADC_50
PIN_241	IO_ADC_51
PIN_161	IO_ADC_52
PIN_185	IO_ADC_53
PIN_188	IO_ADC_54
PIN_164	IO_ADC_55
PIN_191	IO_ADC_56
PIN_167	IO_ADC_57
PIN_194	IO_ADC_58
PIN_170	IO_ADC_59

Note: this table is valid for HY-TTC580, other variants of the HY-TTC500 family may have fewer PINs.

1.4 DIO Blocks

1.4.1 Get_DI

Gets the value of a digital input.

Input: -

Output:
ErrorCode (uint16) : Indication whether the DI measurement is correct (=0).
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_DI_Get()*
di_value (boolean): Input value
TRUE: High level
FALSE: Low level

Parameters:
di_channel (drop-down) : digital input channel
The PIN numbers may be found in table below
pupd (drop-down) : pull up/down configuration

Note: The threshold between high and low is at 2.5V
Note: Only pull-up (IO_DI_PU_10K) option for
IO_DI_00.. 35,
IO_DI_48.. 55,
IO_DI_80.. 87

Note: Only pull-down (IO_DI_PD_10K) option for
IO_DI_56.. 71

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.4.2 Get_DI_VarThreshold

Gets the value of a digital input with an user definable voltage threshold.

Input: -

Output:
ErrorCode (uint16) : Indication whether the DI measurement is correct (=0).
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_DI_Get()*
di_value (boolean) : Input value
TRUE: High level
FALSE: Low level

Parameters:
di_channel (drop-down) : digital input channel
The PIN numbers may be found in table below
pupd (drop-down) : pull up/down configuration
low_thresh1_mV: Defines the lower voltage limit of valid low signal (0 .. 32000mV)

low_thresh2_mV: Defines the upper voltage limit of valid low signal (0 .. 32000mV)
high_thresh1_mV: Defines the lower voltage limit of valid high signal(1 .. 32000mV)
high_thresh2_mV: Defines the upper voltage limit of valid high signal (1 .. 32000mV)

Note: Only pull-up option for
IO_DI_80..87
Note: Only pull-down option for
IO_DI_56..71

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.4.3 Get_K15_

Returns the state of K15

Input: -

Output:
ErrorCode (uint16) : Indication whether the underlying measurement is correct (=0).
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*
description of Driverfunction *IO_POWER_Get()*
state (uint8): state of K15:
OFF = 0
ON = 1

Parameters: -

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.4.4 Get_WakeUp

Returns the state of WakeUp

Input: -

Output:
ErrorCode (uint16) : Indication whether the underlying measurement measurement is correct (=0).
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*
description of Driverfunction *IO_POWER_Get()*
state (uint8) : state of WakeUp
OFF = 0
ON = 1

Parameters: -

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.4.5 Set_DO_HighSide

Sets the value of a digital output.

Input:
do_value (boolean) : value to be output on the PIN (given by channel *do_channel*)
TRUE: High level
FALSE: Low level
Output:
ErrorCode (uint16): ErrorCode returned by the driver function *IO_DO_Set ()* used for setting the digital output.
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_DO_Set()*
Parameters:
do_channel (drop-down) : digital output channel
The PIN numbers may be found in table below
diagnostic (checkbox):Output configuration:
checked: diagnostic pull-up enabled
unchecked: diagnostic pull-up disabled
Note: Diagnostic capabilities only for
IO_DO_00..07,
IO_DO_52..59

Note: The output is protected from overcurrent. In case the protection is activated, a reset is triggered automatically after a defined timespan (details in driver manual). The protection can be reset 10 times, afterwards the output will remain permanently protected, which is indicated by the ErrorCode.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.4.6 Set_DO_HighSide_Fullfb

Sets the value of a digital output.

Input:
do_value (boolean) : value to be output on the PIN (given by channel *do_channel*)
Output:
DO_ErrorCode (uint16): ErrorCode returned by the driver function *IO_DO_Set()* used for setting the digital output.
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_DO_Set()*
Volt_fb (uint16): measured voltage of the digital output in mV (range [0..32000]) **Current_fb** (uint16): measured current of the digital output in mA (range [0..7500])

Parameters:
do_channel (drop-down) : digital output channel
The PIN numbers may be found in table below
diagnostic (checkbox):Output configuration:
checked: diagnostic pull-up enabled
unchecked: diagnostic pull-up disabled

Note: The output is protected from overcurrent. In case the protection is activated, a reset is triggered automatically after a defined timespan (details in driver manual). The protection can be reset 10 times,

afterwards the output will remain permanently protected, which is indicated by the ErrorCode.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.4.7 Set_DO_LowSide

Sets the value of a digital output.

Input:
do_value (boolean): value to be output on the PIN (given by channel do_channel)

Output:
DO_ErrorCode (uint16): ErrorCode returned by the driver function *IO_DO_Set()* used for setting the digital output.
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_DO_Set()*

Parameters:
do_channel (drop-down) : digital output channel
The PIN numbers may be found in table below

Note: No diagnostic capabilities

Note: The output is protected from overcurrent. In case the protection is activated, a reset is triggered automatically after a defined timespan (details in driver manual). The protection can be reset 10 times, afterwards the output will remain permanently protected, which is indicated by the ErrorCode.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.4.8 DIO PINs

Connector PINs	Pin alias
PIN_103	IO_DI_48
PIN_127	IO_DI_49
PIN_104	IO_DI_50
PIN_128	IO_DI_51
PIN_105	IO_DI_52
PIN_129	IO_DI_53
PIN_106	IO_DI_54
PIN_130	IO_DI_55
PIN_107	IO_DI_56
PIN_131	IO_DI_57
PIN_108	IO_DI_58
PIN_132	IO_DI_59
PIN_109	IO_DI_60
PIN_133	IO_DI_61
PIN_110	IO_DI_62
PIN_134	IO_DI_63
PIN_111	IO_DI_64
PIN_135	IO_DI_65
PIN_112	IO_DI_66
PIN_136	IO_DI_67
PIN_113	IO_DI_68
PIN_137	IO_DI_69
PIN_114	IO_DI_70
PIN_138	IO_DI_71
PIN_115	IO_DI_36
PIN_139	IO_DI_37
PIN_116	IO_DI_38
PIN_140	IO_DI_39
PIN_117	IO_DI_40
PIN_141	IO_DI_41
PIN_122	IO_DI_42
PIN_146	IO_DI_43
PIN_123	IO_DI_44
PIN_147	IO_DI_45

PIN_124	IO_DI_46	
PIN_148	IO_DI_47	
PIN_149	IO_DO_00	IO_DI_72
PIN_173	IO_DO_01	IO_DI_73
PIN_152	IO_DO_02	IO_DI_74
PIN_176	IO_DO_03	IO_DI_75
PIN_155	IO_DO_04	IO_DI_76
PIN_179	IO_DO_05	IO_DI_77
PIN_158	IO_DO_06	IO_DI_78
PIN_182	IO_DO_07	IO_DI_79
PIN_251	IO_DO_08	IO_DI_80
PIN_238	IO_DO_09	IO_DI_81
PIN_252	IO_DO_10	IO_DI_82
PIN_239	IO_DO_11	IO_DI_83
PIN_253	IO_DO_12	IO_DI_84
PIN_240	IO_DO_13	IO_DI_85
PIN_254	IO_DO_14	IO_DI_86
PIN_241	IO_DO_15	IO_DI_87
PIN_153	IO_DO_16	IO_DI_00
PIN_177	IO_DO_17	IO_DI_01
PIN_156	IO_DO_18	IO_DI_02
PIN_180	IO_DO_19	IO_DI_03
PIN_159	IO_DO_20	IO_DI_04
PIN_183	IO_DO_21	IO_DI_05
PIN_186	IO_DO_22	IO_DI_06
PIN_162	IO_DO_23	IO_DI_07
PIN_189	IO_DO_24	IO_DI_08
PIN_165	IO_DO_25	IO_DI_09
PIN_192	IO_DO_26	IO_DI_10
PIN_168	IO_DO_27	IO_DI_11
PIN_195	IO_DO_28	IO_DI_12
PIN_171	IO_DO_29	IO_DI_13
PIN_154	IO_DO_30	IO_DI_14
PIN_178	IO_DO_31	IO_DI_15
PIN_157	IO_DO_32	IO_DI_16
PIN_181	IO_DO_33	IO_DI_17
PIN_160	IO_DO_34	IO_DI_18

PIN_184	IO_DO_35	IO_DI_19
PIN_187	IO_DO_36	IO_DI_20
PIN_163	IO_DO_37	IO_DI_21
PIN_190	IO_DO_38	IO_DI_22
PIN_166	IO_DO_39	IO_DI_23
PIN_193	IO_DO_40	IO_DI_24
PIN_169	IO_DO_41	IO_DI_25
PIN_196	IO_DO_42	IO_DI_26
PIN_172	IO_DO_43	IO_DI_27
PIN_101	IO_DO_44	IO_DI_28
PIN_125	IO_DO_45	IO_DI_29
PIN_150	IO_DO_46	IO_DI_30
PIN_174	IO_DO_47	IO_DI_31
PIN_102	IO_DO_48	IO_DI_32
PIN_126	IO_DO_49	IO_DI_33
PIN_151	IO_DO_50	IO_DI_34
PIN_175	IO_DO_51	IO_DI_35
PIN_161	IO_DO_52	IO_DI_88
PIN_185	IO_DO_53	IO_DI_89
PIN_188	IO_DO_54	IO_DI_90
PIN_164	IO_DO_55	IO_DI_91
PIN_191	IO_DO_56	IO_DI_92
PIN_167	IO_DO_57	IO_DI_93
PIN_194	IO_DO_58	IO_DI_94
PIN_170	IO_DO_59	IO_DI_95

Note: this table is valid for HY-TTC580, other variants of the HY-TTC500 family may have fewer PINs.

1.5 PWM Blocks

1.5.1 Set_PWM_Simple

Set the duty cycle for a PWM channel without diagnostic margin and with the high output signal set variable

Input:
duty_cycle (uint16) : Duty cycle for the channel (range [0..65535] corresponding to [0%..100%])

Output:
ErrorCode (uint16) : Indication whether the PWM is correctly set (=0).
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_PWM_SetDuty()*

Parameters:
pwm_channel (drop-down) : PWM channel
The PIN numbers may be found in table below
frequency (drop-down) : PWM frequency in Hz (range [50 .. 1000], only predefined frequencies with a period of an integral multiple of 1ms, 0.5ms or 0.25ms are possible)

Note: The output is protected from overcurrent. In case the protection is activated, a reset is triggered automatically after a defined timespan (details in driver manual). The protection can be reset 10 times, afterwards the output will remain permanently protected, which is indicated by the ErrorCode.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.5.2 Set_PWM_CurrFb

Set the duty cycle for a PWM channel with the high output signal set variable and returning the measured current.

Input:
duty_cycle (uint16): Duty cycle for the channel. Range: 0..65535 corresponding to (0%..100%)

Output:
ErrorCode (uint16): ErrorCode returned by the driver function IO_PWM_SetDuty () used for setting the PWM output.
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_PWM_SetDuty()*
current (uint16): Measured current of the given channel in mA Range (range [0..7500])

Parameters:
pwm_channel (drop-down) : PWM channel
The PIN numbers may be found in table below
frequency (drop-down) : PWM frequency in Hz (range [50 .. 1000], only predefined frequencies with a period of an integral multiple of 1ms, 0.5ms or 0.25ms are possible)
diag_margin (checkbox): application of a diagnostic margin to the PWM output. If the checkbox is ticked, the PWM output can be checked upon short circuit and open load as a minimum high and low time of the PWM signal is guaranteed.

Note: The output is protected from overcurrent. In case the protection is activated, a reset is triggered automatically after a defined timespan (details in driver manual). The protection can be reset 10 times,

afterwards the output will remain permanently protected, which is indicated by the ErrorCode.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.5.3 Set_PWM_FullFb

Set the duty cycle for a PWM channel.

Input:
duty_cycle (uint16): Duty cycle for the channel (range [0..65535] corresponding to [0%..100%])

Output:
ErrorCode (uint16): ErrorCode returned by the driver function IO_PWM_SetDuty () used for setting the PWM output.
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_PWM_SetDuty()*
current (uint16): Measured current of the given channel in mA Range (range [0..7500])
h_time_fb (uint16): High time feedback for the channels. Returns high-time in us
period_fb (uint16): Period feedback for the channels. Returns period in us

Parameters:
pwm_channel (drop-down) : PWM channel
The PIN numbers may be found in table below
frequency (drop-down) : PWM frequency in Hz (range [50 .. 1000],
only predefined frequencies with a period of an integral multiple of 1ms, 0.5ms or 0.25ms are possible)
polarity_high (checkbox): Polarity of output signal
not checked (FALSE): Low output signal is variable
checked (TRUE): High output signal is variable
diag_margin (checkbox): application of a diagnostic margin to the PWM output.
If the checkbox is ticked, the PWM output can be checked upon short circuit and open load
as a minimum high and low time of the PWM signal is guaranteed.

Note: The output is protected from overcurrent. In case the protection is activated, a reset is triggered automatically after a defined timespan (details in driver manual). The protection can be reset 10 times, afterwards the output will remain permanently protected, which is indicated by the ErrorCode.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.5.4 PWM PINs

Connector PIN	Pin aliases
PIN_153	IO_PWM_00
PIN_177	IO_PWM_01
PIN_156	IO_PWM_02
PIN_180	IO_PWM_03
PIN_159	IO_PWM_04
PIN_183	IO_PWM_05
PIN_186	IO_PWM_06
PIN_162	IO_PWM_07
PIN_189	IO_PWM_08
PIN_165	IO_PWM_09
PIN_192	IO_PWM_10
PIN_168	IO_PWM_11
PIN_195	IO_PWM_12
PIN_171	IO_PWM_13
PIN_154	IO_PWM_14
PIN_178	IO_PWM_15
PIN_157	IO_PWM_16
PIN_181	IO_PWM_17
PIN_160	IO_PWM_18
PIN_184	IO_PWM_19
PIN_187	IO_PWM_20
PIN_163	IO_PWM_21
PIN_190	IO_PWM_22
PIN_166	IO_PWM_23
PIN_193	IO_PWM_24
PIN_169	IO_PWM_25
PIN_196	IO_PWM_26
PIN_172	IO_PWM_27
PIN_101	IO_PWM_28
PIN_125	IO_PWM_29
PIN_150	IO_PWM_30
PIN_174	IO_PWM_31
PIN_102	IO_PWM_32
PIN_126	IO_PWM_33

PIN_151	IO_PWM_34
PIN_175	IO_PWM_35

Note: this table is valid for HY-TTC580, other variants of the HY-TTC500 family may have fewer PINs.

1.6 PWD Blocks

1.6.1 PWD_Inc

Setup a single incremental interface and get the counter value.Two PWD channels are needed for one incremental interface.

Input: -

Output:
ErrorCode (uint16): Indication whether the PWD measurement is correct (=0).
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_PWD_IncGet ()*
count (uint16): Value of the incremental counter (0..65535)

Parameters:
inc_channe (drop-down) : Channel of the incremental interface
The PIN numbers may be found in table below
secondary_channel (drop-down) : greyed out, corresponding secondary channel
mode (drop-down) : Defines the counter behavior
IO_PWD_INC_2_COUNT: Counts up/down on any edge of the two input channels
IO_PWD_INC_1_COUNT: Counts up/down on any edge of the 1st input channel only
pupd (drop-down) : Pull up/down interface:
IO_PWD_PU_10K: Pull up 10 kOhm
IO_PWD_PD_10K: Pull down 10 kOhm

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.6.2 PWD_IncReset

Set the counter value of an incremental interface

Input:
b_triggger (boolean): counter is taken over on rising edge of trigger

Output:
ErrorCode (uint16): Indication whether the reset of the incremental value is successful (=0).
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_PWD_IncSet ()*

Parameters:
inc_channel (drop-down) : Channel of the incremental interface
The PIN numbers may be found in table below
count (uint16): Value of the incremental counter (0..65535) to be set

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.6.3 Get_Pulse_Freq_Curr

Get the frequency and the pulse-width from the specified timer channel connected to a current sensor (7mA/14mA)

Input: -

Output:
ErrorCode (uint16): Indication whether the PWD measurement is correct (=0).
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_PWD_IncSet ()*
frequency (uint32): Accumulated frequency in mHz (1/1000 Hz)
pulse_width (uint32): Accumulated pulse-width in us

Parameters:
timer_channel (drop-down): timer channel
The PIN numbers may be found in table below
pulse_mode (drop-down): Specifies the pulse mode
IO_PWD_HIGH_TIME: measure pulse-hightime
IO_PWD_LOW_TIME: measure pulse-low-time
IO_PWD_PERIOD_TIME: measure pulse-high and low-time (Period)
freq_mode (drop-down): Specifies the variable edge
IO_PWD_RISING_VAR: frequency is measured on falling edges
IO_PWD_FALLING_VAR: frequency is measured on rising edges

Note: The output is protected from overcurrent. In case the protection is activated, a reset is triggered automatically after a defined timespan (details in driver manual). When re-enabling the channel after it has been in protection state, it will do a transition to the startup state. This is reflected by the ErrorCode.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.6.4 Get_Pulse_Freq_Volt

Get the frequency and the pulse-width from the specified timer channel driven by a voltage signal.

Input: -

Output:
ErrorCode (uint16) : Indication whether the PWD measurement is correct (=0).
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_PWD_IncSet ()*
frequency (uint32): Accumulated frequency in mHz (1/1000 Hz)
pulse_width (uint32): Accumulated pulse-width in us

Parameters:
timer_channel (drop-down) : timer channel
The PIN numbers may be found in table below
pulse_mode (drop-down) : Specifies the pulse mode
IO_PWD_HIGH_TIME: measure pulse-hightime
IO_PWD_LOW_TIME: measure pulse-low-time
IO_PWD_PERIOD_TIME: measure pulse-high and low-time (Period)
freq_mode (drop-down) : Specifies the variable edge
IO_PWD_RISING_VAR: frequency is measured on falling edges
IO_PWD_FALLING_VAR: frequency is measured on rising edges
pupd (drop-down) : Pull up/down interface:
IO_PWD_NO_PULL: fixed pull resistor
IO_PWD_PU_10K: Pull up 10 kOhm
IO_PWD_PD_10K: Pull down 10 kOhm
IO_PWD_PD_90: Pull down 90 Ohm (for 7mA/14mA sensors)

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.6.5 PWD PINs

Connector PIN	Pin aliases
PIN_115	IO_PWD_00
PIN_139	IO_PWD_01
PIN_116	IO_PWD_02
PIN_140	IO_PWD_03
PIN_117	IO_PWD_04
PIN_141	IO_PWD_05
PIN_122	IO_PWD_06
PIN_146	IO_PWD_07
PIN_123	IO_PWD_08
PIN_147	IO_PWD_09
PIN_124	IO_PWD_10
PIN_148	IO_PWD_11
PIN_101	IO_PWD_12
PIN_125	IO_PWD_13
PIN_150	IO_PWD_14
PIN_174	IO_PWD_15
PIN_102	IO_PWD_16
PIN_126	IO_PWD_17
PIN_151	IO_PWD_18
PIN_175	IO_PWD_19

Note: this table is valid for HY-TTC580, other variants of the HY-TTC500 family may have fewer PINs.

1.7 Sensor Supply Blocks

1.7.1 SensorSupply_5V

Switch on sensor supply 0 or sensor supply 1 with fixed 5 volts.

Input:
enable (boolean): sensor supply is switched on if enable = 1

Output:
ErrorCode (uint16) : ErrorCode returned by the driver function *IO_POWER_Set ()* used for setting the sensor supply output.
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_POWER_Set ()*

Parameters:
sensor_supply (drop-down) : sensor supply (0 or 1)

Note: For ratiometric ADC measurements the sensor supply is switched on automatically (no need to insert this block explicitely)

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.7.2 SensorSupply_Modif

Operate sensor supply 2 for configurable voltage

Input:
Volt (uint8): specifies the output voltage: possible values [0,5,6,7,8,9,10] volts

Output:
ErrorCode (uint16) : ErrorCode returned by the driver function *IO_POWER_Set ()* used for setting the sensor supply output.
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_POWER_Set ()*

Parameters: -

Note: For ratiometric ADC measurements the sensor supply is switched on automatically (no need to insert this block explicitely)

Note: in case the input does not match to possible values the sensor supply is off. Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

Document Number:
D-TTCSW-G-20-011

1.8 PVG and VOUT Blocks

1.8.1 Set_PVG_Simple

Sets the output value of one PVG channel in percent

Input:

percent00 (uint): Output value in percent * 100 (range [1000..9000])

Output:

ErrorCode (uint16): ErrorCode returned by the driver function *IO_PVG_Set ()* used for setting the PVG output.
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_POWER_Set ()*

Parameters:

pvg_channel (drop-down) : PVG channel
The PIN numbers may be found in table below

Note: Initial value of output set to 50

Note: The output is protected from overcurrent. In case the protection is activated, a reset is triggered automatically after a defined timespan (details in driver manual). The protection can be reset 10 times, afterwards the output will remain permanently protected, which is indicated by the ErrorCode.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.8.2 Set_PVG_VoltFb

Sets the output value of one PVG channel in percent

Input:

percent00 (uint16): Output value in percent * 100 (range [1000..9000])

Output:

ErrorCode (uint16): ErrorCode returned by the driver function *IO_PVG_Set ()* used for setting the PVG output.
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_POWER_Set ()*
voltage_fb (uint8): measured voltage in mV (range [0..32000])

Parameters:

pvg_channel (drop-down) : PVG channel
The PIN numbers may be found in table below
output_value_init (drop-down) : output value with which the PVG channel will be initialized
in percent * 100 (range [1000..9000])

Note: The output is protected from overcurrent. In case the protection is activated, a reset is triggered automatically after a defined timespan (details in driver manual). The protection can be reset 10 times, afterwards the output will remain permanently protected, which is indicated by the ErrorCode.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.8.3 Set_VOUT_Simple

Sets the output voltage of one voltage output channel

Input:
voltage (uint16): Output voltage in mV (range [0..32000])

Output:
ErrorCode (uint16):): ErrorCode returned by the driver function *IO_VOUT_Set ()* used for setting the VOUT output.
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_VOUT_SetVoltage ()*

Parameters:
vout_channel (drop-down) : VOUT channel
The PIN numbers may be found in table below

Note: The output is protected from overcurrent. In case the protection is activated, a reset is triggered automatically after a defined timespan (details in driver manual). The protection can be reset 10 times, afterwards the output will remain permanently protected, which is indicated by the ErrorCode.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.8.4 Set_VOUT_VoltFb

Sets the output voltage of one voltage output channel, returning the measured voltage in mV

Input:
voltage (uint16): Output voltage in mV (range [0..32000])

Output:
ErrorCode (uint16): ErrorCode returned by the driver function *IO_VOUT_Set ()* used for setting the VOUT output.
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_VOUT_SetVoltage ()*
voltage_fb (uint16) : measured voltage in mV (range [0..32000])

Parameters:
vout_channel (drop-down) : VOUT channel
The PIN numbers may be found in table below

Note: The output is protected from overcurrent. In case the protection is activated, a reset is triggered automatically after a defined timespan (details in driver manual). The protection can be reset 10 times, afterwards the output will remain permanently protected, which is indicated by the ErrorCode.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

Document Number:
D-TTCSW-G-20-011

1.8.5 PVG and VOUT PINs

Connector PIN	Pin aliases	
PIN_161	IO_PVG_00	IO_VOUT_00
PIN_185	IO_PVG_01	IO_VOUT_01
PIN_188	IO_PVG_02	IO_VOUT_02
PIN_164	IO_PVG_03	IO_VOUT_03
PIN_191	IO_PVG_04	IO_VOUT_04
PIN_167	IO_PVG_05	IO_VOUT_05
PIN_194	IO_PVG_06	IO_VOUT_06
PIN_170	IO_PVG_07	IO_VOUT_07

Note: this table is valid for HY-TTC580, other variants of the HY-TTC500 family may have fewer PINs.

1.9 UART Blocks

1.9.1 Write_RS232_

Writes a value with a comment to UART interface.

Input:
value (any native datatype): value to be displayed

Output: -

Parameters:
Comment (String): used as description of the value to be displayed. The length of the string together with the length of the value to be displayed may not exceed 80 characters.

Used settings are: 115.200 baud, 8 databits, 1 stopbit, no parity

The buffer is only sent out every n-th cycle, where n is 100 by default. In order to change this default, define a variable DERATE_PRINTF in the MATLAB command window with a different value.
If the model contains reference models with Write_RS232 blocks, it is necessary to add at least one *Write_RS232* block to the root model. If not, the cursor wouldn't get reset to the top-left position.

Note: Do not use any escape sequences nor format characters in the comment string.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*
in chapter *IO_UART_Write ()*

1.9.2 Write_RS232_green

Triggers writing of a non zero value with green font to UART interface.

Input:
value (any native datatype) : value to be displayed

Output: –

Parameters:
comment (String): used as description of the value to be displayed in green font. The length of the string together with the length of the value to be displayed may not exceed 80 characters.

Used settings are: 115.200 baud, 8 databits, 1 stopbit, no parity

The buffer is only sent out if value is not zero. This happens every n-th cycle, where n is 100 by default. In order to change this default, define a variable DERATE_PRINTF in the MATLAB command window with a different value.
If the model contains reference models with Write_RS232 blocks, it is necessary to add at least one *Write_RS232* block to the root model. If not, the cursor wouldn't get reset to the top-left position.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.9.3 Write_RS232_red

Triggers writing of a non zero value with red comment to UART interface.

Input:
value (any native datatype) : value to be displayed

Output: -

Parameters:
comment (String): used as description of the value to be displayed. The length of the string together with the length of the value to be displayed may not exceed 80 characters.

Used settings are: 115.200 baud, 8 databits, 1 stopbit, no parity

The buffer is only sent out if value is not zero. This happens every n-th cycle, where n is 100 by default. In order to change this default, define a variable DERATE_PRINTF in the MATLAB command window with a different value.
If the model contains reference models with Write_RS232 blocks, it is necessary to add at least one *Write_RS232* block to the root model. If not, the cursor wouldn't get reset to the top-left position.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.9.4 UART_Write

Writes multiple values with a comment to UART interface.

Input:
value (array): This quantity is the output via serial interface for every single item, if the parameter **Comment** contains the corresponding type qualifier (%d for integers and %f for floats, can be added multiple times.)

Output:
ErrorCode (uint16): Return value of the IO-driver *IO_UART_Write()*
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_UART_Write ()*

Parameters:
Comment (String): used as a description of the value to be displayed.
(length of the string + length of the values) may not exceed 80 characters.

Used settings are: 115.200 baud, 8 databits, 1 stopbit, no parity

In order to display the values you need to use type specifiers (e.g. %d or %f). It is not checked whether these specifiers are correct. You may use multiple type specifiers if you have a multidimensional input signal.

For example:
pressure: %f , temperature: %5.2f
terminal Output: pressure: 0.358703 , temperature: 32.04

The buffer is only sent out every n-th cycle, where n is 100 by default. In order to change this default define a variable DERATE_PRINTF in the MATLAB command window with a different value.

Note: Differently to the behaviour of Write_RS232, the cursor will not get reset to the top-left position and it is possible to use special characters.

Note: LF is added automatically at the end of the string which means you do not need to add *\r\n* at the end.

Note: mixing UART_Write with Write_RS232 blocks within the same mode is not advisable, as Write_RS232 blocks perform a reset of the cursor within the terminal. Depending on the code placement by the EmbeddedCoder this might wipe some outputs from the terminal.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.9.5 UART_Read

Read data from serial interface into an array of uint8 with a user defined length.

Input: -

Output:
ErrorCode (uint16): Output of the IO-driver function *IO_UART_Read()*
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_UART_Read ()*
data (uint 8 [user defined length]) : array of uint8 with user defined length given by the parameter *Size of data array*
rx_len (uint16): Actually read bytes

Parameters:
Size of data array (range[1..512]): defines the width of the output data.
Show additional information (checkbox): further configure:
Baud rate: Baud rate in baud/s (range [1200 .. 115200])
default: 115.200 baud
Bits per frame (drop-down):number of data bits per frame
default: 8 databits
Parity configuration (drop-down) : possible parity configuration
default: no parity
Stop bits (drop-down) : for number of stop bits per frame
default: 1 stopbit

Note: mixing UART_Read blocks with any UART_Write or Write_RS232 blocks within the same mode is not advisable, as UART_Read blocks may initialize or require a different setting for baudrate, databits, parity or stopbits, unless default settings are kept.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.10 EEPROM Blocks

1.10.1 EEPROM_Read

Read data from the EEPROM.

Input: -

Output:
b_ready (boolean): when reading from EEPROM is not accomplished the output is FALSE.
data (uint32): Actual value that was retrieved from EEPROM.

Parameters:
offset (uint16): place in the EEPROM where data will be retieved from.
[0..65535] for TTC580 and [0..32767] for TTC590
EEPROM_type (drop-down): defines the length of data to be read from EEPROM.
Possible values: uint8 (=1byte), uint16 (=2bytes), uint32 (=4bytes)

Note: When the expression *(offset + length)* of Bytes exceed the limit of 65535 no reading from EEPROM takes place and a constant zero is output as *data* and *b_ready* is FALSE.
When *b_ready* is not TRUE do not trust data as reading of EEPROM is not finished at startup of the ECU.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.10.2 EEPROM_Read_Raw

Read data from the EEPROM (raw)

Input: -

Output:
ErrorCode(uint16): Output of EEPROM read driver function.
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_EEPROM_Read ()*
data (corresponding to *EEPROM_type*): Value read from EEPROM.

Parameters:
offset (uint16): place in the EEPROM where data will be retieved from.
[0..65535] for TTC580 and [0..32767] for TTC590
EEPROM_type (drop-down): defines the length of data to be read from EEPROM.
Possible values: uint8 (=1byte), uint16 (=2bytes), uint32 (=4bytes)

Note: a semaphore for avoiding concurrent EEPROM access needs to be implemented in the application

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.10.3 EEPROM_Read_OffsetIn

Read data from the EEPROM.

Input:
offset (uint16): place in the EEPROM where data will be retieved from.
[0..65535] for TTC580 and [0..32767] for TTC590

Output:
b_ready (boolean): when reading from EEPROM is not accomplished the output is FALSE.
data (corresponding to *EEPROM_type*): Actual value that was retrieved from EEPROM.

Parameter:
EEPROM_type (drop-down): defines the length of data to be read from EEPROM.
Possible values: uint8 (=1byte), uint16 (=2bytes), uint32 (=4bytes)

Note: When the expression *(offset + length)* of Bytes exceed the EEPROM-limit no reading from EEPROM takes place and a constant zero is output as data and *b_ready* is FALSE. When *b_ready* is not TRUE do not trust data as reading of EEPROM is not finished at startup of the ECU.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.10.4 CRC32_EEPROM_Read

Calculate and compare the crc32 (data integrity check) for a given section

Input: -

Output:
b_ready (boolean):
TRUE = read operation accomplished
FALSE = read operation still processing
CRC_OK (int8): may hold three possible values:
-1 => after EEPROM section was read, CRC check failed (terminal state)
0 => reading of EEPROM section not completed yet, CRC check is pending
+1 => after EEPROM section was read, CRC check was successful (terminal state)

Parameters:
BasicAddr (uint16): starting-address of the section for which crc32 is calculated
EndAddr (uint16): end-address of the section for which crc32 is calculated
offset(uint16): EEPROM/FRAM memory offset (0..65535 for EEPROM, 0..32767 for FRAM)

Note: the crc32 is calculated for the given section [BasicAddr .. EndAddr] out of the EEPROM and compares the result to the content of a given address in EEPROM.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.10.5 IO_EEPROM_GetStatus

Returns the status of the EEPROM driver.

Input: -

Output:
ErrorCode (uint16): status of the EEPROM driver. If the ErrorCode returns *IO_E_OK*, the EEPROM is free to be accessed.
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_EEPROM_GetStatus ()*

Parameters: -

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.10.6 EEPROM_Write

Writes data to the EEPROM.

Input:
trigger (boolean): Data is written to EEPROM on the risinge edge of the trigger input.
data (uint32): Actual value that needs to be written to EEPROM.

Output:
b_ready (boolean): when writing to EEPROM is not accomplished *b_ready* is FALSE

Parameters:
offset (uint16): index in of the EEPROM-dataspace where data will be placed.
[0..65535] for TTC580 and [0..32767] for TTC590
EEPROM_type (drop-down): defines the length of data to be written to EEPROM.
Possible values: uint8 (=1byte), uint16 (=2bytes), uint32 (=4bytes)

Note: When the expression *(offset + length)* of Bytes exceed the limit of *65535* no writing to EEPROM takes place, and *b_ready* is FALSE.
On a positive edge of the input *trigger* data is captured and stored internally into a mirror. Actual writing is performed when EEPROM is accessable. This mechanism is handled internally therefore the output *b_ready* doesn't need to be fed back to the tigger.
Output *b_ready* may be used for subsequent actions like switch-off logic for the ECU, in order to make sure that no EEPROM write activity is pending while shutdown.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.10.7 EEPROM_Write_Raw

Writes data to the EEPROM (raw)

Input:
data (uint32): Actual value that needs to be written to EEPROM.

Output: **ErrorCode** (uint16): Output of EEPROM write driver function.
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_EEPROM_Write ()*

Parameters:
offset (uint16): index of the EEPROM-dataspace where data shall be placed.
[0..65535] for TTC580 and [0..32767] for TTC590
EEPROM_type (drop-down): defines the length of data to be written to EEPROM.
Possible values: uint8 (=1byte), uint16 (=2bytes), uint32 (=4bytes)

Note: a semaphore for avoiding concurrent EEPROM access needs to be implemented in the application.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.10.8 EEPROM_Write_OffsetIn

Writes data to the EEPROM.

Input:
trigger (boolean): Data is written to EEPROM on the risinge edge of the trigger input.
data (uint32): Actual value that needs to be written to EEPROM.
offset (uint16): index of the EEPROM-dataspace where data will be placed.
[0..65535] for TTC580 and [0..32767]for TTC590

Output:
b_ready (boolean): when writing to EEPROM is not accomplished *b_ready* is FALSE.

Parameters:
EEPROM_type (drop-down): defines the length of data to be written to EEPROM.
Possible values: uint8 (=1byte), uint16 (=2bytes), uint32 (=4bytes).

Note: When the expression *(offset + length)* of Bytes exceed the EEPROM-limit, no writing to EEPROM takes place, and *b_ready* is FALSE.
On a positive edge of the input *trigger* data is captured and stored internally into a mirror.
Actual writing is performed when EEPROM is accessable. This mechanism is handled internally, therefore the output *b_ready* doesn't need to be fed back to the tigger.
Output *b_ready* may be used for subsequent actions like switch-off logic for the ECU, in order to make sure that no EEPROM write activity is pending.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.10.9 CRC32_EEPROM_Write

Calculate and write the crc32 (data integrity check) for a given section

Input:
trigger (boolean): triggers calculation of crc32

Output:
b_ready (boolean): is TRUE when crc32 is completed and written to EEPROM at given address (Parameter *offset*)

Parameters:
BasicAddr (uint16): starting-point/address of the section for which crc32 is calculated
EndAddr (uint16): ending-point/address of the section for which crc32 is calculated
offset (uint16): EEPROM/FRAM memory offset (0..65535 for EEPROM, 0..32767 for FRAM) to which the crc32 is written. 4 bytes are used for the crc32 value.

Note: the crc32 is calculated out of the EEPROM for the given section [BasicAddr..EndAddr]. The resulting crc32 is written to the given address in EEPROM.

Note: The input *trigger* has to be connected to the same trigger as for all EEPROM_Write blocks accessing the interval [BasicAddr .. EndAddr]; otherwise data inconsistencies may occur.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.11 LIN Blocks

1.11.1 IO_LIN_Init

Initialization of the LIN communication driver.

Input: -

Output:
ErrorCode (uint16): Indication whether the Initialization of the communication driver has been successful ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_LIN_Init ()*

Parameters:
baudrate (uint16): Baud rate in bit/s (range [1000..20000])
checksum_type (drop-down): Checksum type

Note: This block may only occur once in the application.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.11.2 LIN_Read

Reads a LIN frame with the given *id* and of the given *length*.

Input:
b_request (boolean): If true, sends a reading request to a LIN-slave with given *id* and *length*.

Output:
ErrorCode (uint16): indication whether data of the last retrieve request were successfully read from the LIN bus.
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_LIN_Read ()*
data (uint8): LIN frame array. Parameters *Id* and *length* must be set before. The received frame will be stored in the data part.

Parameters:
id (drop-down): Corresponds to the LIN subscription ID.
length (drop-down): Data read from LIN have the width given statically by length.

Note: Read request is processed only if the LIN-Bus is not blocked by a previous uncomplete action.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.11.3 LIN_Write

Transmit a LIN frame with the given id and of the given length.

Input:
b_send (boolean): if true, writes data on LIN bus with given *ID* and *length*
data (uint8): LIN frame array. *ID* and *length* and data parts must be provided

Output:
ErrorCode (uint16): indication whether the request has been submitted.

ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_LIN_Write ()*

Parameters:
id (drop-down): Corresponds to the LIN subscription ID.
length (drop-down): Data written on LIN have the width given statically by length.

Note: Send command is only processed if the LIN-Bus is not blocked by a previous uncomplete action.

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

1.11.4 IO_LIN_GetStatus

Returns the status of the LIN channel.

Input: -

Output:
ErrorCode (uint16): status of the LIN channel
ErrorCodes can be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf* description of Driverfunction *IO_LIN_GetStatus ()*

Parameters: -

Further information may be obtained from *HY-TTC_500_IO_Driver_Manual_<ver>.pdf*

2 Add on Libraries

2.1 AddOn_Lib_TTC580_J1939DMx

2.1.1 MainDlg_J1939DMx

Block representing main environment-settings of the applications to be run on a TTC580 enableing the J1939 functionality additionally.

Parameters: (additionally to the Parameters of ”MainDlg”)
Enable DMx services (J1939) (checkbox): activates sending DM1 and DM2/DM3 services out of J1939 standard
J1939 channel (drop-down): use the dedicated CAN channel for sending DM1 messages and receiving DM2/DM3 request messages. Note: The chosen J1939 CAN channel has to be run with 250kbit/s according to J1939 standard **J1939SorceAddress** (uint8): define source address to be sent in J1939 DM1 messages and retreiving from by DM2/DM3 services
Basis Address DM2 (range [0..63488]): Definition of the basis address in EEPROM to be used for DM2 storage. The amount of storage space is 2048 bytes. Note: In this storage space no other data than DM2 should be stored in. That needs to be considered when using blocks out of the EEPROM block family out of the IOLib-Blockset. Note: Default is 63488 which places the persistant DM2 data right at the end of the EEPROM storage space.

2.1.2 DMx_IO_Block

Sending J1939 DM1 messages according to the input *ErrorCodeIn*.

Input:
ErrorCodeIn (uint16[2]): has to be connected to the ErrorCode output of an IO-Block

Output:
ErrorCode (uint16): feed-through ErrorCode of the preceding IO-Block

Parameters:
J1939 SPN_Offset (range [0..524032]): *J1939_SPN_Offset* + *PIN-define* (see IO_Pin.h); if chosen beyond pssible limits, no DTC will be sent.

DM1.FMI: Desired class of faults may be enabled by the check boxes:
Enable FMI: ”DATA ERRATIC INTERMITTENT OR INCORRECT” (checkbox):
Enable FMI: ”VOLTAGE ABOVE NORMAL OR SHORTED TO HIGH SOURCE” (checkbox):
Enable FMI: ”VOLTAGE BELOW NORMAL OR SHORTED TO LOW SOURCE” (checkbox):
Enable FMI: ”CURRENT BELOW NORMAL OR OPEN CIRCUIT” (checkbox):
Enable FMI: ”CURRENT ABOVE NORMAL OR GROUNDED CIRCUIT” (checkbox):
Enable FMI: ”ROOT CAUSE NOT KNOWN” (checkbox):
Enable FMI: ”BAD INTELLIGENT DEVICE OR COMPONENT” (checkbox):

Note: DM1 messages are sent over the channel defined in the MainDlg-block.

Example: Example_DMx_IO_Block_For_User_Manual.txt

Document Number:
D-TTCSW-G-20-011

2.1.3 DMx_Custom_Block

Sending J1939 DM1 messages according to the input *ErrorCodeIn*.

Input:
ErrorCodeIn (uint16): J1939 DM1 messages are sent according to this input

Output:
ErrorCode (uint16): The output ErrorCode is the feed-through ErrorCode of the preceding Block

Parameters:
J1939 SPN (range [0..524287]): If *J1939 SPN* is chosen beyond possible limits, no DTC will be sent

DM1.FMI: According to the mapping given in *"IOLib500_FMI_map.c"*. This file may be adapted by the user in the labeled section of *"IOLib580_Environment/src/IOLib500_FMI_map.c"*.

Note: If the input ErrorCode is not found in *"IOLib500_FMI_map.c"* then DM1.FMI = 0x12 is output which stand for *"BAD INTELLIGENT DEVICE OR COMPONENT"*. The same holds if the FMI defined by the user is outside the interval [0..31].

Note: DM1 messages are sent over the channel defined in the MainDlg-block.

Note: A DTC will be transmitted over DM1 as long as it is not set inactive (ErrorCode = 0). This have to be taken into account if this block is used within ported subsystems (e.g.: enabled, triggered, switched).

Please check the example "Example_IOBlock_Library_DMx_Appl.mdl" out of the examples collection for further information.

2.1.4 DMx_Application_Mask

Process DM1 and DM2 according to mask entries. Input:
error_active (boolean): J1939 DM1 messages are sent when the *error_active* is *TRUE*.

Output: -

Parameters:
J1939 SPN (range [0..524287]): If J1939 SPN is chosen beyond possible limits, no DTC will be sent.
J1939_FMI (range [0..31]): the following assignment holds:

FMI=2 DATA ERRATIC, INTERMITTENT OR INCORRECT
FMI=3 VOLTAGE ABOVE NORMAL, OR SHORTED TO HIGH SOURCE
FMI=4 VOLTAGE BELOW NORMAL, OR SHORTED TO LOW SOURCE
FMI=5 CURRENT BELOW NORMAL OR OPEN CIRCUIT
FMI=6 CURRENT ABOVE NORMAL OR GROUNDED CIRCUIT
FMI=11 ROOT CAUSE NOT KNOWN
FMI=12 BAD INTELLIGENT DEVICE OR COMPONENT

DM1 messages are sent over the channel defined in the MainDlg-block.
DM2 are stored in EEPROM at the location defined in the MainDlg-block and are serviced upon reception of DM2-Request message and deleted by DM3-Request message.

Note: If the FMI defined by the user is outside the interval [0..31] then *DM1.FMI* = 0x12 is output which stand for *"BAD INTELLIGENT DEVICE OR COMPONENT"*.

Note: A DTC will be transmitted over DM1 as long as it is not set inactive (*error_active* = FALSE). This have to be taken into account if this block is used within ported subsystems (e.g.: enabled, triggered,

switched)

Please check the example ”Example_IOBlock_Library_DMx_Appl.mdl” out of the examples collection for further information.

2.1.5 DMx_Application_Input

Process DM1 and DM2 according to mask entries to given input.

Input:
error_active (boolean): J1939 DM1 messages are sent when the *error_active* input is *TRUE*.
J1939_FMI (range [0..31): If the FMI defined by the user is beyond possible limits then *DM1.FMI* = 0x12 is output which stand for *”BAD INTELLIGENT DEVICE OR COMPONENT”*
the following assignment holds:

FMI=2 DATA ERRATIC, INTERMITTENT OR INCORRECT
FMI=3 VOLTAGE ABOVE NORMAL, OR SHORTED TO HIGH SOURCE
FMI=4 VOLTAGE BELOW NORMAL, OR SHORTED TO LOW SOURCE
FMI=5 CURRENT BELOW NORMAL OR OPEN CIRCUIT
FMI=6 CURRENT ABOVE NORMAL OR GROUNDED CIRCUIT
FMI=11 ROOT CAUSE NOT KNOWN
FMI=12 BAD INTELLIGENT DEVICE OR COMPONENT

Output: -

Parameters:
J1939 SPN (range [0..524287]): If J1939 SPN is chosen beyond possible limits, no DTC will be sent.

DM1 messages are sent over the channel defined in the MainDlg-block.
DM2 are stored in EEPROM at the location defined in the MainDlg-block and are serviced upon reception of DM2-Request message and deleted by DM3-Request message.

Note: This block is kept in the library for legacy reasons. For new applications please use ”DMx_Application_AllInput” instead.

Note: A DTC will be transmitted over DM1 as long as it is not set inactive (*error_active* = FALSE). This have to be taken into account if this block is used within ported subsystems (e.g.: enabled, triggered, switched)

2.1.6 DMx_Application_AllInput

Process DM1 and DM2 according to mask entries to given input.

Input:
error_active (boolean): when TRUE = J1939 DM1 messages are sent
J1939_SPN (range [0..524287]): If J1939 SPN is chosen beyond possible limits, no DTC will be sent.
J1939_FMI (range [0..31): If the FMI defined by the user is beyond possible limits then *DM1.FMI* = 0x12 is output which stand for *”BAD INTELLIGENT DEVICE OR COMPONENT”*
the following assignment holds:

FMI=2 DATA ERRATIC, INTERMITTENT OR INCORRECT
FMI=3 VOLTAGE ABOVE NORMAL, OR SHORTED TO HIGH SOURCE
FMI=4 VOLTAGE BELOW NORMAL, OR SHORTED TO LOW SOURCE
FMI=5 CURRENT BELOW NORMAL OR OPEN CIRCUIT
FMI=6 CURRENT ABOVE NORMAL OR GROUNDED CIRCUIT
FMI=11 ROOT CAUSE NOT KNOWN
FMI=12 BAD INTELLIGENT DEVICE OR COMPONENT

Output: -

Parameters: -

DM1 messages are sent over the channel defined in the MainDlg-block.
DM2 are stored in EEPROM at the location defined in the MainDlg-block and are serviced upon reception of DM2-Request message and deleted by DM3-Request message.

Note: A DTC will be transmitted over DM1 as long as it is not set inactive (*error_active* = FALSE). This have to be taken into account if this block is used within ported subsystems (e.g.: enabled, triggered, switched)

Please check the example ”Example_IOBlock_Library_DMx_Appl.mdl” out of the examples collection for further information.

2.1.7 DM2_KL15shutdown

Holds output high for DM2 synchronization

Input:
bKL15 (boolean): can be connected to *Get_K15*, in order to detect power down request

Output:
bPowerSelfHold (boolean): By default FALSE. As soon as *bKL15* goes low, *bPowerSelhold* becomes high (TRUE) as long as synchronization of DM2 is active. After sychronization is done the output *bPowerSelfhold* goes to FALSE again and the ECU may be switched off safely

Parameters: -

Purpose of use is to synchronize and store DM2 persistently before shutdown of the ECU.

Note: the block *DM2_KL15shutdown* may be placed in parallel (combined by OR) to any other afterrun block like e.g. *IO_EEPROM_GetStatus()*

Document Number:
D-TTCSW-G-20-011

2.2 AddOn_Lib_TTC580_CCP

2.2.1 MainDlg_CCP

Block representing main environment-settings of the applications to be run on a TTC580 enableing the J1939 functionality additionally.

Parameters: (additionally to the Parameters of ”MainDlg”)
Enable CCP (checkbox): By checking this box you activate the CCP driver on the TTC500 and you may define the following quantities:
CRO ID (): Standard CAN identifier for CROs (Host -> ECU)
DTO ID (): Standard CAN identifier for DTOs (ECU -> Host)
CCP Station Address (): Logical station address in little-endian byte order
CCP Station ID (): String containing the station ID (Maximum length 8 characters)
CCP channel (drop-down): select the channel for CCP. This channel needs to run a minimum baudrate of 250kBaud.
CRO DTO Id format (drop-down): Select if Ids of CRO and DTO shall be in standard or extended format.
Enable DAQ sync (checkbox): The CCP-Driver on the TTC500 shall exhibit synchronous functionallity based on application cycle time (event No.0)
Enable DAQ alternate (checkbox): The CCP-Driver on the TTC500 shall exhibit synchronous functionality based on double application cycle time with alternating events (event No.1 and 2)
Total ODTs for DAQ lists (range [1..254]): The given amount of ODTs are equally distributed among the DAQ lists.

Note: the left over time is used to transmit DAQ-lists for synchronous mode in CCP

Note: if total amount of ODTs are chosen too high, the ECU may not process all DAQ lists

Note: if the amount of ODTs is chosen higher than 126, overrun indication for the CCP master is turned off

Document Number:
D-TTCSW-G-20-011

3 Tools

3.1 EEPROM Importer

function [*ErrorCode*] = EEPROM_Import(*sXls_File*)

Name
EEPROM_Import.m

In
sXls_File : [Path\]*Filename.xls*

Purpose
Import EEPROM configuration data and generate a library containing blocks for all EEPROM access
Every block has the parameters defined in *Filename.xls* as ports.
Every block has a busy output indicating that the EEPROM access is ongoing
Generated EEPROM-Read blocks have an output *CRC_OK*
which may hold three possible values:
-1 => after EEPROM section was read, CRC check failed (terminal state)
0 => reading of EEPROM section not completed yet, CRC check is pending
+1 => after EEPROM section was read, CRC check was successful (terminal state)
Generated EEPROM-Write blocks have a input trigger, on a rising edge parameters
are written physically to EEPROM

Example
Open the xls-File *"eeprom_example.xls"* and generate the corresponding EEPROM blocks within the library *LibEEPROM_eeprom_example.mdl* by calling the command
"ErrorCode = EEPROM_Import('*eeprom_example.xls*')" in the command window of MATLAB

function [ErrorCode] = hex_read(*sXls_File*, *sIhex_File*)

Name
hex_read.m

In
sXls_File : [Path\]*ExcelFilename.xls*
sIhex_File : [Path\]*IhexFilename.ihex*

Out
ErrorCode : 0 ... at least one paramter group stored successfully
1 ... no parameter group processed successfully

Purpose
Store EEPROM data into an Excel file.

Prerequisite
EEPROM data have to be already stored into an ihex-file.
This action is accomplished by the TTC-Downloader.
The ihex-file has to cover at least all addresses for data defined in the Excel-file

Example
The following needs to be placed in the MATLAB command window:
ErrorCode = hex_read('*eeprom_example.xls*','*eeprom_example.ihex*')

Document Number:
D-TTCSW-G-20-011

3.2 DBC Importer

function [ErrorCode] = DBC_Import(*sDBC_File*, sNetworkNodeName [, CAN_Channel [, CycleTime [,bDebounce])

Name
DBC_Import.m

In
sDBC_File : [Path\]*Filename.dbc*
sNetworkNodeName : Name of the Network Node that is defined in the DBC-File.
In case sNetworkNodeName is a empty string, libraries for every Network Node are generated.
CAN_Channel: out of [0..6].
Default is 0 (meaning CAN_0)
CycleTime: Application runs on this CycleTime [ms]. Default is 10[ms]
This value is taken for derate parameter of CAN-TX and CAN-RX messages.
bDebounce: out of [0..1].
Default is 0 (meaning *"no debounce"*)
Debouncing holds only for cyclic RX-messages, for other messages no effect.
Switching on means that an Rx-error is only passed to the application if the CAN-driver issues a *Not_OK* on two consecutive cycles.

Purpose
Import CAN-DBC and generate a library containing blocks for all rx/tx messages

Example
Open the DBC-File *"hermes_mini_codings.dbc"* and generate the corresponding network node *"measure"* mapped onto the CAN0 of the ECU assuming a cycle time of the application being 10ms and debouncing is switched on.
The following needs to be placed in the MATLAB command window:
ErrorCode = DBC_Import('*hermes_mini_codings.dbc*','*measure*', 0, 10, 1)

Document Number:
D-TTCSW-G-20-011

Legal Disclaimer

THE INFORMATION GIVEN IN THIS DOCUMENT IS GIVEN AS A SUPPORT FOR THE USAGE OF THE PRODUCT AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE PRODUCT. THE RECIPIENT OF THIS DOCUMENT MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. THIS DOCUMENT WAS MADE TO THE BEST OF KNOWLEDGE OF TTCONTROL GMBH. NEVERTHELESS AND DESPITE GREATEST CARE, IT CANNOT BE EXCLUDED THAT MISTAKES COULD HAVE CREPT IN. TTCONTROL GMBH PROVIDES THE DOCUMENT FOR THE PRODUCT “AS IS” AND WITH ALL FAULTS AND HEREBY DISCLAIMS ALL WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ACCURACY OR COMPLETENESS, OR OF RESULTS TO THE EXTENT PERMITTED BY APPLICABLE LAW. THE ENTIRE RISK, AS TO THE QUALITY, USE OR PERFORMANCE OF THE DOCUMENT, REMAINS WITH THE RECIPIENT. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW TTCONTROL GMBH SHALL IN NO EVENT BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOSS OF DATA, DATA BEING RENDERED INACCURATE, BUSINESS INTERRUPTION OR ANY OTHER PECUNIARY OR OTHER LOSS WHATSOEVER) ARISING OUT OF THE USE OR INABILITY TO USE THE DOCUMENT EVEN IF TTCONTROL GMBH HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

IF THE PRODUCT IS MARKED AS “PROTOTYPE”, THE DELIVERED PRODUCT IS A DEVELOPMENT SAMPLE (“SAMPLE”). THE RECIPIENT ACKNOWLEDGES THAT THEY ARE ALLOWED TO USE THE SAMPLE ONLY IN A LABORATORY FOR THE PURPOSE OF DEVELOPMENT. IN NO EVENT IS THE RECIPIENT ALLOWED TO USE THE SAMPLE FOR THE PURPOSE OF SERIES MANUFACTURING.

TTCONTROL GMBH PROVIDES NO WARRANTY FOR ITS PRODUCTS OR ITS SAMPLES, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW DISCLAIMS ALL LIABILITIES FOR DAMAGES RESULTING FROM OR ARISING OUT OF THE APPLICATION OR USE OF THESE PRODUCTS OR SAMPLES. THE EXCLUSION OF LIABILITY DOES NOT APPLY IN CASES OF INTENT AND GROSS NEGLIGENCE. MOREOVER, IT DOES NOT APPLY TO DEFECTS WHICH HAVE BEEN DECEITFULLY CONCEALED OR WHOSE ABSENCE HAS BEEN GUARANTEED, NOR IN CASES OF CULPABLE HARM TO LIFE, PHYSICAL INJURY AND DAMAGE TO HEALTH. CLAIMS DUE TO STATUTORY PROVISIONS OF PRODUCT LIABILTY SHALL REMAIN UNAFFECTED.

ANY DISPUTES ARISING OUT OF OR IN CONNECTION WITH THIS DOCUMENT SHALL BE GOVERNED SOLELY BY AUSTRIAN LAW, EXCLUDING ITS CONFLICT OF LAW RULES AND THE UNITED NATIONS CONVENTION ON CONTRACTS FOR THE INTERNATIONAL SALE OF GOODS. SUCH DISPUTES SHALL BE DECIDED EXCLUSIVELY BY THE COURTS OF VIENNA, AUSTRIA.

ALL PRODUCT NAMES AND TRADEMARKS MENTIONED IN THIS USER MANUAL ARE THE PROPERTY OF THEIR RESPECTIVE OWNERS, WHICH ARE IN NO WAY ASSOCIATED OR AFFILIATED WITH TTCONTROL GMBH.

I/O Block Library for TTC 580 user manual

Document version **1.0.0** of 2021-09-30

Document number: **D-TTCSW-G-20-011**