EtherCAT Force Control

Open-loop current control / Closed-loop force control

Application Note

agito

www.agito-akribis.com

Member of Akribis Systems group

Revision History

Version	Description	Date
1.0	Initial Release	17 October 2025

Contact Information

Manufacturer	Agito Akribis Systems Ltd., Member of Akribis Systems Group
Address	56 Serangoon North Avenue 4, Singapore 555851
Telephone	+65 6484 3357
Website	www.agito-akribis.com

Copyright Notice

Products Rights

AGDx, AGCx, AGMx, AGAx, AGIx, and AGLx are products designed by Agito Akribis Systems Ltd. in Israel. Sales of the products are licensed to Akribis Systems Pte Ltd. under intercompany license agreement.

Agito Akribis Systems Ltd. has full rights to distribute above products worldwide.

Disclaimer

This product documentation was accurate and reliable at the time of its release.

Agito Akribis Systems Ltd. reserves the right to change the specifications of the product described in this manual without notice at any time.

Trademarks

Agito PCSuite is a trademark of Agito Akribis Systems Ltd..

Warranty

This product is warranted to be free of defects in material and workmanship and conforms to the specifications listed in this manual, for a period of 12 months from the shipment date from factory.

Contents

1 Introduction 4
1.1 General Information 4
1.2 Functional Description (CST) 4
2 AGenData User Program 5
2.1 Description of Related EtherCAT Objects 5
2.1.1 Object 2900h: AGenData[1-10] 5
2.1.2 Object 2904h: AGenDataStatus[1-10] 5
2.1.3 Object 6071h: Target Torque 6
2.1.4 Object 6077h: Torque Actual Value 6
2.2 Verification & Creating User Program in PCSuite 6
2.2.1 Slow Approach 6
2.2.2 Mode Switching 7
2.2.3 Command Source 10
2.2.4 User Program (Example) 11
2.3 TwinCAT / PLC Programming Example 13
2.3.1 Modifying PDO List 13
2.3.2 Creating Global Variable List 13
2.3.3 Writing POU 14
2.3.3.1 Customized Function Block 16
3 Using PLCOpen Torque Control (CST Mode) 19
3.1 Description of Related EtherCAT Objects 19
3.1.1 Object 2410h: Analog Input 1 Value 19
3.1.2 Object 2810h: CST Force Loop Enable 19
3.1.3 Object 2811h: CST Command Interpolation 19
3.1.4 Object 6071h: Target Torque 20
3.1.5 Object 6077h: Torque Actual Value 20
3.2 TwinCAT / PLC Programming Example 21
3.2.1 CST Current Loop 21
3.2.1.1 Modifying PDO List 21
3.2.1.2 Writing POU 21
3.2.1.3 Verification 23
3.2.2 CST Force Loop 24
3.2.2.1 Modifying PDO List 24
3.2.2.2 Startup Parameters 24
3.2.2.3 Modifying Output Scaling Factor / CST Interpolation Command Factor 25
3.2.2.4 Writing POU 26
3.2.2.5 Verification 29

1 Introduction

1.1 General Information

This application note describes the various force control methods that can be achieved with our driver. There are two different kind of force control that our drive can achieve, one is through “open-loop” method whereby there is no force sensor feedback connected to the system, and it is outputting the target torque/current according to the ramp. The other method is through “closed-loop” method whereby there is force sensor feedback connected to the system and the drive is trying to close the force loop while outputting the target torque/command. There are currently two methods to achieve this application, one is through “AGenData User Program” and the other is through the conventional way being described in “CiA402-2, CANopen device profile for drives and motion control, Part 2: Operation Modes and Application Data, Chapter 20”, which is the Cyclic Synchronous Torque Mode (CST).

Chapter 2 of this application note describes the use of “AGenData User Program” method and how it can be achieved, and Chapter 3 describes the use of CST mode with examples from TwinCAT.

1.2 Functional Description (CST)

Figure 1 shows the defined input objects as well as the output objects. With this mode, the trajectory generator is located in the control device, not in the drive device. The user may specify the desired target torque (6071_h) and rate of desired ramp and also monitor the outputs torque actual value (6077_h), velocity actual value ($606C_h$) and position actual value (6064_h)

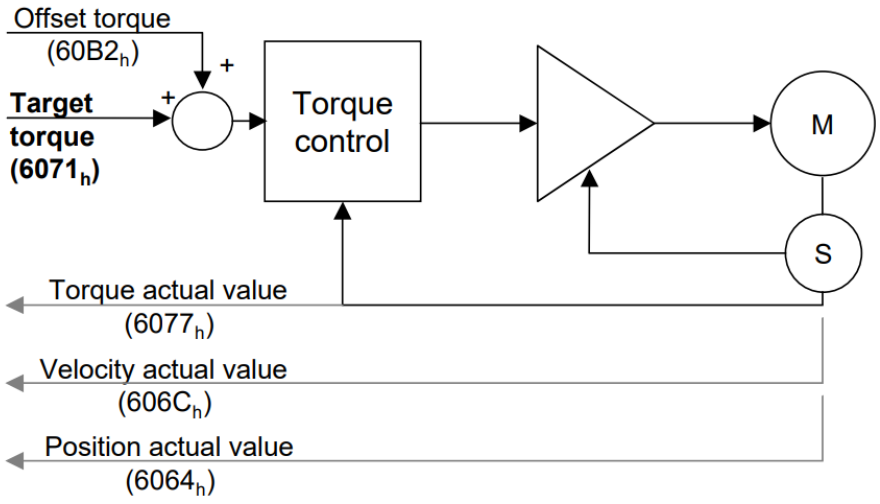


Figure 1 – Cyclic synchronous torque mode overview

2 AGenData User Program

2.1 Description of Related EtherCAT Objects

This section touches on the related EtherCAT Objects (on top of the mandatory ones) that can be / should be mapped to the PDO to ensure cyclic exchanging of information between the master and controller. The mandatory EtherCAT Objects that should have been mapped are “StatusWord”, “Position Actual Value”, “ControlWord” and “Target Position”.

2.1.1 Object 2900h: AGenData[1-10]

AGenData[1-10] consists of 10 arrays which user can freely use in conjunction with any user program that is running inside the drive itself, this is mainly used for assigning the AGenData value to the drive.

Attribute	Value
Sub-index	00_h
Description	Highest sub-index supported
Access	rw
PDO Mapping	Yes
Value Range	10
Default Value	10
Sub-index	01_h
Description	AGenData 1
Access	rw
PDO Mapping	Yes
Value Range	Interger32
Default Value	0
All the subsequent AGenData sub-indices possess the same attribute and value as sub-index 1.	

2.1.2 Object 2904h: AGenDataStatus[1-10]

AGenDataStatus[1-10] consists of 10 arrays which user can freely use in conjunction with any user program that is running inside the drive itself, this is only used for reading back the AGenData value.

Attribute	Value
Sub-index	00_h
Description	Highest sub-index supported
Access	rw
PDO Mapping	Yes
Value Range	10
Default Value	10
Sub-index	01_h
Description	AGenDataStatus 1
Access	rw

PDO Mapping	Yes
Value Range	Interger32
Default Value	0
All the subsequent AGenDataStatus sub-indices possess the same attribute and value as sub-index 1.	

2.1.3 Object 6071h: Target Torque

This object shall indicate the configured input value for the torque controller in profile torque mode/cst mode. The value shall be given per thousand of rated torque.

Attribute	Value
Sub-Index	00_h
Access	rw
PDO Mapping	Yes
Value Range	Interger16
Default Value	0000_h

2.1.4 Object 6077h: Torque Actual Value

This object shall provide the actual value of the torque. It shall correspond to the instantaneous torque in the motor. The value shall be given per thousand of rated torque. Currently, there is no torque feedback in our drive and since T = kt * I and assuming that the motor constant is static, the calculation of the torque feedback will be in terms of the percentage of the continuous current that is currently used.

Attribute	Value
Sub-Index	00_h
Access	rw
PDO Mapping	Yes
Value Range	Interger16
Default Value	0000_h

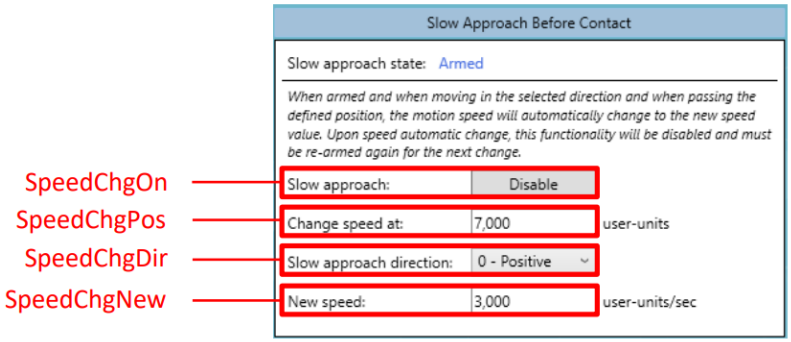
2.2 Verification & Creating User Program in PCSuite

Users can refer to “Agito AN – Force Control” with regards to configuration of force sensor (if any) and tuning of current loop / force loop. This section will be an abstract from the “Agito AN – Force Control” to describe the portion of the UI that the user can use to do their force control verification before writing moving on to writing the force control user program.

2.2.1 Slow Approach

In force control applications, it is common to have the tool move towards the object at a high speed and then slowing down when it approaches so as to reduce the kinetic energy and impulse upon contact.

To configure the slow approach, navigate to the “Slow Approach Before Contact” window and set the relevant parameters.

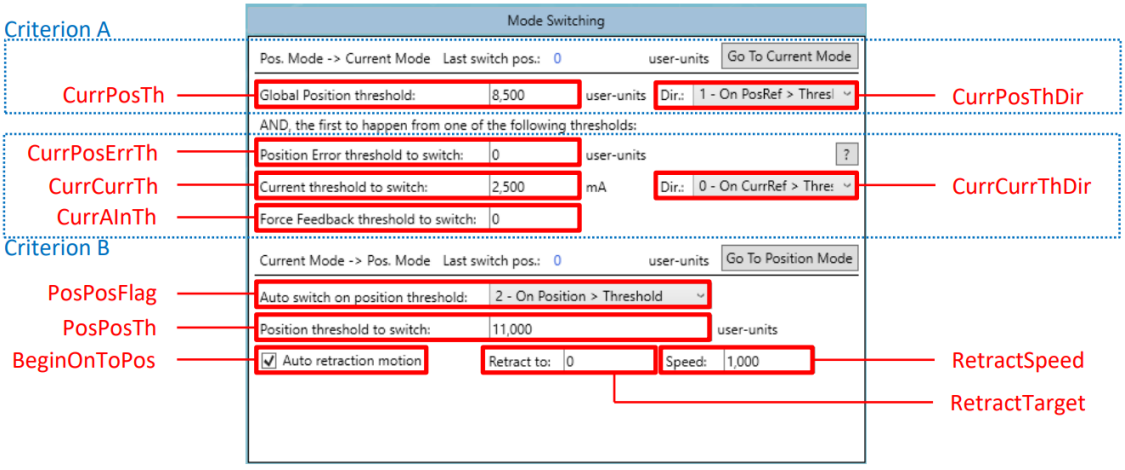


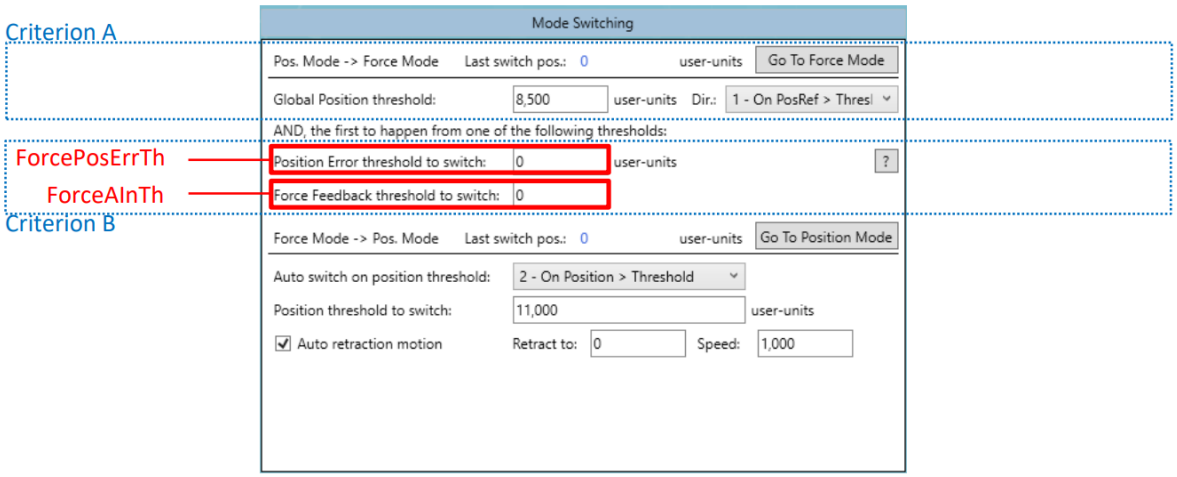
In Figure x, the parameters are set such that when the motor moves past the position of 7,000 counts in the positive direction, it will start to decelerate (or possibly accelerate) to the new speed of 3000 count/s.

Note: This function must be re-enabled after every cycle of force control. It will automatically disable the function at the end of every force control cycle.

2.2.2 Mode Switching

Navigate to “Mode Switching” window; the trigger parameters to switch from position to force/current mode are configured in this window.





Two main criterions have to be fulfilled for switching to occur; in this document it will be referred as Criterion A and Criterion B.

Criterion A is a position criterion which allows for mode switching to occur only when the motor is out of a certain position range. This prevents unintended switching during the acceleration and deceleration phases of the motion profile which tend to have higher currents and higher position errors. Such unintended early switching may cause the motor to accelerate (in current/force mode) over a distance and cause damage to the system or part. See relevant keywords below.

<table>
<tr><th>Keyword</th><th>Description</th></tr>
<tr><td>CurrPosTh</td><td>CurrPosTh defines a position threshold value that PosRef must be greater (or smaller) than, as one of the criteria for switching to Current/Force Mode to occur.

Set this value to a position slightly before your contact point. For example, if the starting position is at 0 counts and the contact position is between 9,000 to 10,000 counts, this value can be set to 8,500 counts.</td></tr>
<tr><td>CurrPosThDir</td><td>CurrPosThDir defines if the criteria of CurrPosTh is positive, negative or bypassed when set to the following values.

<table>
<tr><th>Value</th><th>Criteria</th></tr>
<tr><td>-1</td><td>PosRef < CurrPosThDir</td></tr>
<tr><td>0</td><td>Bypass</td></tr>
<tr><td>1</td><td>PosRef > CurrPosThDir</td></tr>
</table>

Set this value to 1 if moving in a positive direction towards the contact point. E.g., Starting point is at 0 counts and contact point is at 10,000 counts.

Or set the value to -1 if the contact point is in the negative direction. E.g., Starting point is at 20,000 counts and contact point is at 10,000 counts.</td></tr>
</table>

Criterion B is composed of sub-criterions (three sub-criterions **[B1, B2, B3]** for current operation mode and two **[B1**, **B2]** for force operation mode).

Criterion B is fulfilled as long as any single one of the sub-criterions, B1 or B2 or B3 is fulfilled.

Sub-criterion B1: The first sub-criterion checks for position error, if the position error grows bigger than the threshold value, then criterion A is fulfilled. See relevant keywords below.

Keyword	Description
CurrPosErrTh/ ForcePosErrTh	CurrPosErrTh/ForcePosErrTh defines a threshold value for going into Current/Force Operation Mode if position error exceeds the threshold. If the threshold is set to 0, then this trigger is not in use.

Sub-criterion B2: The second sub-criterion checks for analog input, if the analog input grows larger (or smaller) than the threshold value, then criterion A is fulfilled. See relevant keywords below.

Keyword	Description
CurrAInTh/ ForceAInTh	CurrAInTh/ForceAInTh defines a threshold value for going into Current/Force Operation Mode if analog input is greater (less than, if threshold value is negative) than the threshold. If the threshold is set to 0, then this trigger is not in use.

Sub-criterion B3: The third sub-criterion is only applicable for current operation mode and checks for current reference, if current reference is greater than (or less than) the threshold value, then criterion A is fulfilled. See relevant keywords below.

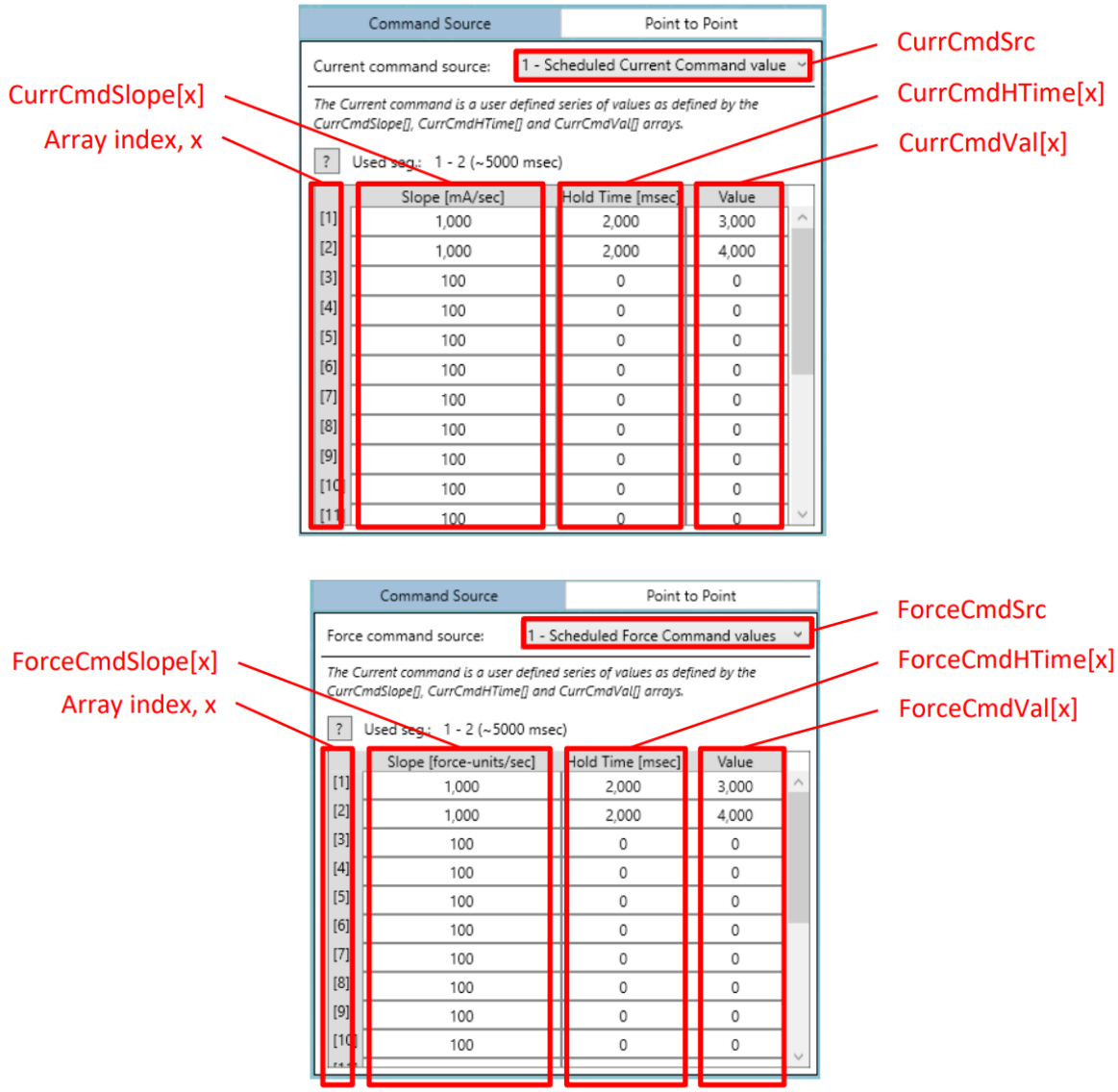
Keyword	Description
CurrCurrTh	CurrCurrTh defines a position threshold value for going into Current Operation Mode if the motor current is greater (or lesser) than the threshold. If the threshold value is set to 0, then the trigger is not in use.
CurrCurrThDir	CurrCurrThDir defines if the criteria of CurrCurrTh is positive or negative when set to the following values.

Value	Criteria
0	CurrRef > CurrCurrTh
1	CurrRef < CurrCurrTh

When both **Criterion A and Criterion B** are fulfilled, operation mode switching will be triggered. **Upon triggering, the threshold value of the sub-criterion B that was triggered will be reset to 0, this is to prevent unintended repeated triggering. Users will have to re-apply the value after every cycle.** The remaining sub-criterions will remain unchanged. Although the controller allows flexibility in defining multiple triggers (sub-criterions of B) at the same time, it is recommended to select and just use one of it.

2.2.3 Command Source

The next step would be to configure the current/force profile, navigate to the “Command Source” window. In the relevant table, assign the current/force profile. Alternatively, the command source can also be configured to come from an analog input, but it will not be shown in the example.



In the example above, the profile is configured to have two steps. Upon the trigger to switch operation modes, the controller will increase/decrease the force/current command to 3,000 units at a rate of 1,000 units/s. Then, the current/force command will then be held at this level for 2,000ms. After which, the controller will increase the current/force command to 4,000mA at a rate of 1,000 units/s and hold at this level for another 2,000ms. Units will be in mA or force-units depending on the mode selected.

Keyword	Description
CurrCmdSrc/ ForceCmdSrc	CurrCmdSrc/ForceCmdSrc defines the command source. Value: 0 — Source: Analog input (In addition to setting this keyword to 0, it is also required to set the analog input to Force Command in IO Page) Value: 1 — Source: Scheduled by table of values
CurrCmdVal[x]/ ForceCmdVal[x]	CurrCmdVal[x]/ForceCmdVal[x] defines the current/force command value for step x.
CurrCmdSlope[x]/ ForceCmdSlope[x]	From the previous step to step x, there 'will' be a difference in the current/force command. Instead of a step jump, the controller will gradually increase/decrease from the current/force command at the rate defined in CurrCmdSlope[x]/ForceCmdSlope[x], in mA/sec or force-unit/sec.
CurrCmdHTime[x]/ ForceCmdHTime[x]	After increasing/decreasing the current/force reference to the command value, the controller will hold the reference at the commanded value for the period of time defined in CurrCmdHTime[]/ForceCmdHTime[].

2.2.4 User Program (Example)

This section shows an example of using the IDE to program the various parameters for both current open loop and force close loop in conjunction with the use of AGenData.

```
//
main([10,30],[5,20],[800,1000])
//
// Place here the main code of your program
    AGenData[1] = 0 //初始化力控开始下压开关
    AGenData[3] = 0 //初始化执行Begin开关
    AGenData[4] = 0 //初始化力控按压完成状态
    AGenData[5] = 0 //初始化线程#2激活状态
    AGenData[6] = 0 //初始化线程#2复位开关
    AProgRun[2], 2  //运行开环力控线程

    //设定门槛值【注意：需要根据实际情况，设定为合理的值】
    AGenData[7]=-19000//400//40500   //设定A轴切换到力控模式的全局位置门槛值（正数：按压方向为编码器正方向；负数：按压方向为编码器负方向）
    AGenData[2]=300     //设定A轴切换到力控模式的压力反馈门槛值

    //设定力控基本参数【注意：EtherCAT主站控制时，应注释掉，交给EtherCAT主站去赋值】
//  AGenData[10]=150   //设定开环压力电流目标值
//  AGenData[9]=10     //设定压力电流保持时间
//  AGenData[8]=20000     //设定运动速度（正值：按压方向为编码器正方向；负值：按压方向为编码器负方向）
    AStop
    while(AMotionStat!=0)
    end
    AMotionMode=19

    while (1)

        if(AProgStat[2]==1)
            AGenData[5]=1
        else if(AProgStat[2]==0)
            AGenData[5]=0
```

Initialize AGenData values at start up.

Settings of parameters to be use in other task for current/force control.

Initialization process, default motionmode is 19 for CSP mode. Not necessary.

```
while (1)

    if(AProgStat[2]==1)
        AGenData[5]=1
    else if(AProgStat[2]==0)
        AGenData[5]=0
    end
```

Checking if the User Program thread 2 is running and indicate in AGenData.

```
    if (AGenData[6]==1) //EtherCAT主站发 "AGenData[6]=1" 重启线程#2
        AGenData[5]=0
        AGenData[1]=0
        AGenData[3] = 0 //初始化执行Begin开关
        AGenData[4] = 0 //初始化力控按压完成状态
        AProgHalt[2]
        AProgReset[2]
        AWaitTime, 10
        AProgRun[2], 2
        if(AProgStat[2]==1)
            AGenData[5]=1
        else if(AProgStat[2]==0)
            AGenData[5]=0
        end

        AStop
        while(AMotionStat!=0)
        end
        AMotionMode=19
        AMotorOn=1

        AGenData[6]=0
```

Resetting of the user program and initialize parameters if user program has an error and indicate that User Program has reset by deasserting AGenData[6].

```
task: OpenLoopFC{2}
    AGoToPosMode    //切换到位置模式
    while(AOperationMode!=3)
    end

    while(1)

        //开环力控下压
        if(AGenData[1]==1)
            AGenData[4] = 0  //初始化力控按压完成状态
            AGoToPosMode    //切换到位置模式
            while(AOperationMode!=3)
            end

            AStop           //停止运动并切换到JOG速度运动模式
            AMotorOn=1
            AMotionMode=0

            //模式切换的门槛值设定
            ACurrPosThDir=-1                    //设定切换到力控模式的全局位置条件方向（1：按压方向为编码器正方向；-1：按压方向为编码器负方向）
            ACurrPosTh=AGenData[7]              //设定切换到力控模式的全局位置门槛值（正数：按压方向为编码器正方向；负数：按压方向为编码器负方向）
            ACurrCurrThDir=1                    //设定切换到力控模式的电机电流条件方向（0：按压方向为编码器正方向；1：按压方向为编码器负方向）

            ACurrCmdSrc=1
            ACurrCmdSlope[1]=(abs(AGenData[10]))*100   //开环力控电流变化斜率（只能是正值）
            ACurrCmdHTime[1]=-1//AGenData[9]          //开环力控压力电流保持实际
            ACurrCmdVal[1]=AGenData[10]           //开环力控压力电流指令值
            ACurrCmdHTime[2]=0

            //AForceCmdSrc = 1


            //AForceCmdSrc = 1
            //AForceCmdSlope[1]=(abs(AGenData[10]))*100
            //AForceCmdHTime[1]=-1 , if its negative its infinite hold time.
            //AForceCmdVal[1]=AGenData[10]
            //AForceCmdHTime[2]=0

            ASpeedChgOn = 0                     //开启或关闭切低速功能，0：关闭；1：开启

            AAccel=(abs(AGenData[8]))*100       //加速度（只能是正值）
            ADecel=(abs(AGenData[8]))*100       //减速度（只能是正值）
            ASpeed=AGenData[8]                  //设定运动速度（正值：按压方向为编码器正方向；负值：按压方向为编码器负方向）
            ABegin
            while (AMotionStat==0)
            end

            ACurrCurrTh=AGenData[2]                 //设定切换到力控模式的电机电流门槛值（正数：按压方向为编码器正方向；负数：按压方向为编码器负方向）

            while(AOperationMode==3)      //首先，等待位置模式下的运动结束
            end
            AGenData[4] = 1 // finish force control力控按压完成
            //while(AOperationMode==1)      //其次，等待开环力控按压结束
            //end
            AGenData[1] = 0
        end


        //执行Begin（开始运动）指令
        if(AGenData[3]==1)
            AGoToPosMode
            AStop
```

Running Thread 2 Task 2.

If AGenData[1] is set from the master, intialise

Relevant threshold settings to switch from position to force/current mode.

Relevant command source and slope profile for force/current mode.

On the fly speed change option, can input relevant on the fly speed change parameters as well if opted for it.

Motion profile to reach contact point.

Threshold of current to switch, can be put together with the other relevant threshold.

Waiting for current/force control to finish as it will auto switch back to position mode. Once done, it is indicated by asserting AgenData[4].

```
        //执行Begin（开始运动）指令
        if(AGenData[3]==1)
            AGoToPosMode
            AStop
            while (AMotionStat!=0)
            end
            AMotionMode=19 // 切换到CSP模式
            while (AMotionMode!=19)
            end

            AWaitStatus[7],0
            ABegin
            while (AMotionStat!=1)
            end
            AGenData[4] = 0
            AGenData[3] = 0
        end


    end   //end of while(1)
endoftask
```

Waiting for user to give AGendata[3] = 1 to switch back the motionmode to CSP mode in order to move via EtherCAT control.

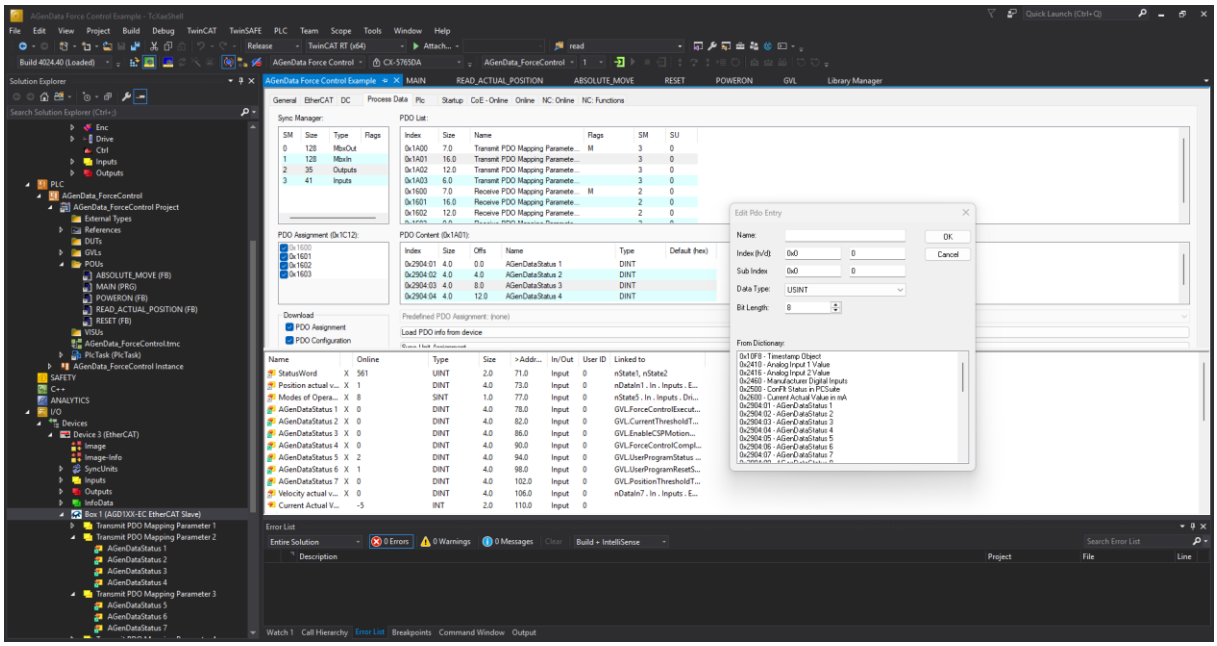
User will have to ensure that the Target Position of the EtherCAT master matches the current actual position of the motor as by using this method we have decoupled the control from the master and will have to realign everything before giving control back to the master.

There will be an example shown in the twincat controls below to show this.

2.3 TwinCAT / PLC Programming Example

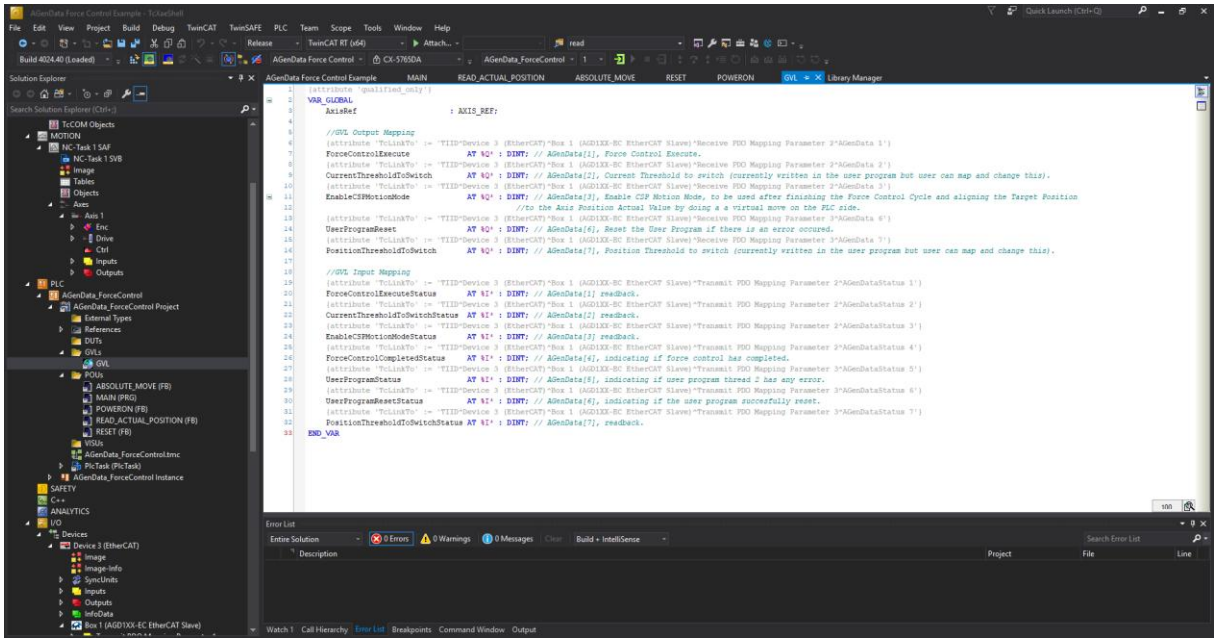
2.3.1 Modifying PDO List

It is important to map the AGenData / AGenDataStatus object (current limit is 10 array size for both object) as an PDO to ensure that the cyclic exchange of information between the master and the slave. AGenData is not mapped by default in the PDO list, users have to navigate to the pages whereby they are able to do the PDO configuration to modify the slave PDO list. Inside the TxPDO section, users will be able to find AGenDataStatus as a PDO object, this is generally use for status readback of the particular AGenData that is located under the RxPDO section.



2.3.2 Creating Global Variable List

Once the PDO list have been modified to include the AGenData/AGenDataStatus , we can create a global variable list and give names to it and link it with the PDO object that we desire so that these variables can be used while creating the customized function block for the force control or any application usage.



2.3.3 Writing POU

The next step will be writing the whole script to test out the force control application. Inside the example below, a customized function block is created to start / indicate when the force/current control has finished and realigning both the master and slave back to move via CSP mode.

```
GVL.JoggingSpeed := 10000;

(*Power On*)
axisPower(axis_Test := GVL.AxisRef, bPowerEnable := bPowerEnable, bPowerStatus => bPowerStatus, bPowerBusy => bPowerBusy, bPowerActive => bPowerActive, bPowerError => bPowerError, nPowerErrorID => nPowe
(*Reset*)
axisReset(axis_Test := GVL.AxisRef, bResetExecute := bResetExecute, bResetDone => bResetDone, bResetBusy => bResetBusy, bResetError => bResetError, nResetErrorID => nResetErrorID);
(*Move Absolute*)
axisMoveAbsolute(axis_Test := GVL.AxisRef, bMoveAbsoluteExecute := bMoveAbsoluteExecute, ABORT_DUE_TO_CST := ABORT_DUE_TO_CST,  nMoveAbsoluteErrorID => nMoveAbsoluteErrorID, bMoveAbsoluteDone => bMoveAb
                bMoveAbsoluteBusy => bMoveAbsoluteBusy, bMoveAbsoluteActive => bMoveAbsoluteActive);
(*Read Actual Position*)
axisReadPosition(axis_Test := GVL.AxisRef, bReadActualPosEnable := TRUE, bReadActualPosValid => bReadActualPosValid, bReadActualPosBusy => bReadActualPosBusy, bReadActualPosError => bReadActualPosError,
                nReadActualPosErrorID => nReadActualPosErrorID, nActualPosition => nActualPosition);
(*Force Control*)
axisForceControl(bForceControlExecute := bForceControlExecute, nCurrPosTh := GVL.PositionThresholdToSwitch, nCurrCurrTh := GVL.CurrentThresholdToSwitch, nJoggingSpeed := GVL.JoggingSpeed,
                bForceControlCompleted => bForceControlCompleted, bForceControlBusy => bForceControlBusy, bForceControlActive => bForceControlActive, bForceControlError => bForceControlError);
(*AlignMasterSlave*)
axisAlignMasterSlave(bAlignMasterSlaveExecute := bAlignMasterSlaveExecute, bAlignMasterSlaveCompleted => bAlignMasterSlaveCompleted, bAlignMasterSlaveBusy => bAlignMasterSlaveBusy,
                bAlignMasterSlaveActive => bAlignMasterSlaveActive, bAlignMasterSlaveError => bAlignMasterSlaveError);
(*User Program Reset*)
axisUserProgramReset(bUserProgramResetExecute := bUserProgramResetExecute, bUserProgramResetCompleted => bUserProgramResetCompleted , bUserProgramResetBusy => bUserProgramResetBusy,
                bUserProgramResetActive => bUserProgramResetActive, bUserProgramResetError => bUserProgramResetError);


CASE seq1State OF
    SEQDEMOSTATE.SEQ1_RESET:
    (*Initialization logic / Reset*)
        bResetExecute := TRUE;
```

Instance of function block used

```
CASE seq1State OF
    SEQDEMOSTATE.SEQ1_RESET:
    (*Initialization logic / Reset*)
        bResetExecute := TRUE;
        bUserProgramResetExecute := TRUE;
        bPowerEnable  := FALSE;

        IF bResetDone AND bUserProgramResetCompleted THEN
            bResetExecute := FALSE;
            bUserProgramResetExecute := FALSE;

            seq1State := SEQDEMOSTATE.SEQ1_POWER_ENABLE;
        ELSIF bResetError THEN
            bResetExecute := FALSE;
            bUserProgramResetExecute := FALSE;
            nResetErrorID := nResetErrorID;

            seq1State := SEQDEMOSTATE.SEQ1_FAULT;
        END_IF

    SEQDEMOSTATE.SEQ1_POWER_ENABLE:
    (*Axis Power*)
        bPowerEnable := TRUE;

        IF bPowerStatus THEN
            seq1State := SEQDEMOSTATE.SEQ1_FORCE_CONTROL_ENABLE;
        ELSIF bPowerError THEN
```

Reset / Initialization Sequence

```
SEQDEMOSTATE.SEQ1_POWER_ENABLE:
(*Axis Power*)
    bPowerEnable := TRUE;

    IF bPowerStatus THEN
        seq1State := SEQDEMOSTATE.SEQ1_FORCE_CONTROL_ENABLE;
    ELSIF bPowerError THEN
        bPowerEnable := FALSE;

        seq1State := SEQDEMOSTATE.SEQ1_FAULT;
    END_IF

SEQDEMOSTATE.SEQ1_FORCE_CONTROL_ENABLE:
(*Force Control Execute*)
    bForceControlExecute := TRUE;

    IF bForceControlBusy AND bForceControlActive THEN
        seq1State := SEQDEMOSTATE.SEQ1_FORCE_CONTROL_WAIT;
    ELSE
        seq1State := SEQDEMOSTATE.SEQ1_FORCE_CONTROL_ENABLE;
    END_IF

SEQDEMOSTATE.SEQ1_FORCE_CONTROL_WAIT:

    IF bForceControlCompleted THEN
        bForceControlExecute := FALSE;

        seq1State := SEQDEMOSTATE.SEQ1_VIRTUAL_ABSOLUTE_MOVE;
```

Axis Power

Force Control & Auto Retract Motion to a certain position (e.g. 2000 counts)

```
        seq1State := SEQDEMOSTATE.SEQ1_VIRTUAL_ABSOLUTE_MOVE;
    ELSIF bForceControlError THEN
        bForceControlExecute := FALSE;

        seq1State := SEQDEMOSTATE.SEQ1_FAULT;
    END_IF

SEQDEMOSTATE.SEQ1_VIRTUAL_ABSOLUTE_MOVE:
(*setting virtual move to current position of the axis*)
    axisMoveAbsolute.nTargetPosition := nTargetPosition_1;
    axisMoveAbsolute.nTargetVelocity := nTargetVelocity_1;
    axisMoveAbsolute.nTargetAcceleration := nTargetAcceleration_1;
    axisMoveAbsolute.nTargetDeceleration := nTargetDeceleration_1;
    axisMoveAbsolute.nTargetJerk := nTargetJerk_1;

    bMoveAbsoluteExecute := TRUE;

    seq1State := SEQDEMOSTATE.SEQ1_VIRTUAL_ABSOLUTE_MOVE_WAIT;

SEQDEMOSTATE.SEQ1_VIRTUAL_ABSOLUTE_MOVE_WAIT:
    IF NOT bMoveAbsoluteBusy AND bMoveAbsoluteDone THEN
        bMoveAbsoluteExecute := FALSE;

        seq1State := SEQDEMOSTATE.SEQ1_ALIGN_MASTER_SLAVE;
    ELSIF NOT bMoveAbsoluteBusy AND (bMoveAbsoluteError OR bMoveAbsoluteCommandAborted) THEN
        bMoveAbsoluteExecute := FALSE;

        seq1State := SEQDEMOSTATE.SEQ1_FAULT;
```

Virtual absolute move of the axis to the position that it was retracted to (e.g. 2000). To align the Target Position on the master and the PosRef of the driver.

Note: The axis will not physically move as the motionmode for the axis is not put to 19 (motionmode used to move in EtherCAT Mode)

```
        seq1State := SEQDEMOSTATE.SEQ1_FAULT;
    END_IF

SEQDEMOSTATE.SEQ1_ALIGN_MASTER_SLAVE:
(*This is solely changing the driver motion mode to work in CSP in EtherCAT, can be integrated into the MoveAbsFunction logic as well, as we did a virtual move to align driver position reference
 and Target Position on EtherCAT*)
    bAlignMasterSlaveExecute := TRUE;

    seq1State := SEQDEMOSTATE.SEQ1_ALIGN_MASTER_SLAVE_WAIT;

SEQDEMOSTATE.SEQ1_ALIGN_MASTER_SLAVE_WAIT:
    IF bAlignMasterSlaveCompleted THEN
        bAlignMasterSlaveExecute := FALSE;

        seq1State := SEQDEMOSTATE.SEQ1_MOVE_TO_TARGET;
    ELSIF bAlignMasterSlaveError THEN
        bAlignMasterSlaveExecute := FALSE;

        seq1State := SEQDEMOSTATE.SEQ1_FAULT;
    END_IF
```

Once the TargetPosition and the PosRef of the driver is aligned, we can trigger this function to turn motionmode = 19 and prepare for any movement from the master thereafter.

NOTE: Only call this once the TargetPosition and the posref of the driver is aligned.

```
SEQDEMOSTATE.SEQ1_MOVE_TO_TARGET:
    axisMoveAbsolute.nTargetPosition := nTargetPosition_2;
    axisMoveAbsolute.nTargetVelocity := nTargetVelocity_2;
    axisMoveAbsolute.nTargetAcceleration := nTargetAcceleration_2;
    axisMoveAbsolute.nTargetDeceleration := nTargetDeceleration_2;
    axisMoveAbsolute.nTargetJerk := nTargetJerk_2;
```

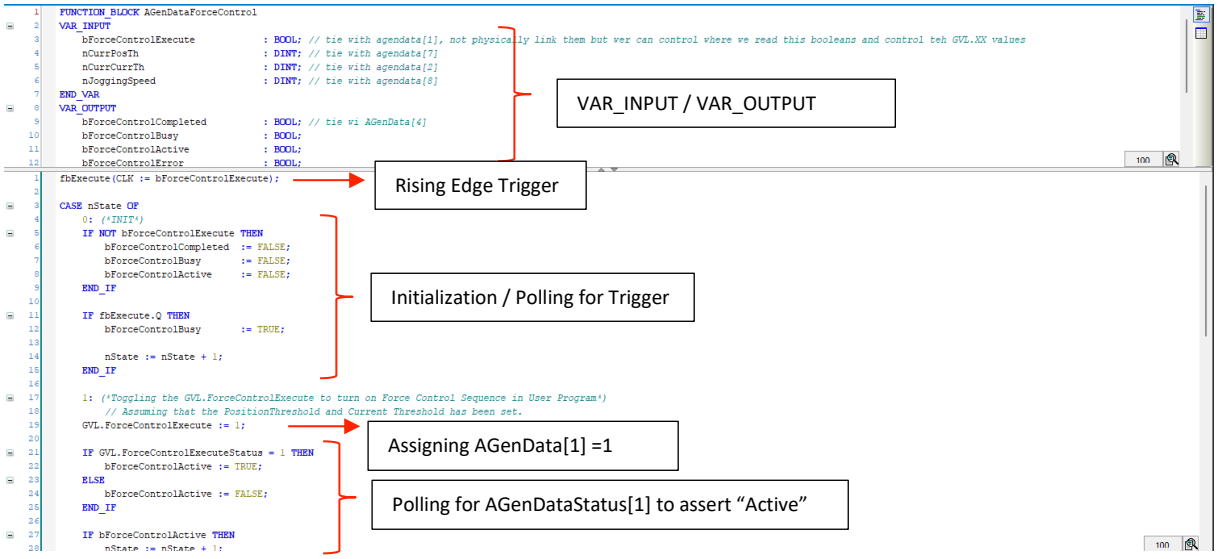
```
SEQDEMOSTATE.SEQ1_MOVE_TO_TARGET:
    axisMoveAbsolute.nTargetPosition := nTargetPosition_2;
    axisMoveAbsolute.nTargetVelocity := nTargetVelocity_2;
    axisMoveAbsolute.nTargetAcceleration := nTargetAcceleration_2;
    axisMoveAbsolute.nTargetDeceleration := nTargetDeceleration_2;
    axisMoveAbsolute.nTargetJerk := nTargetJerk_2;

    bMoveAbsoluteExecute := TRUE;

    seq1State := SEQDEMOSTATE.SEQ1_MOVE_TO_TARGET_WAIT;

SEQDEMOSTATE.SEQ1_MOVE_TO_TARGET_WAIT:
    IF NOT bMoveAbsoluteBusy AND bMoveAbsoluteDone THEN
        bMoveAbsoluteExecute := FALSE;

        seq1State := SEQDEMOSTATE.SEQ1_FORCE_CONTROL_ENABLE;
    ELSIF NOT bMoveAbsoluteBusy AND (bMoveAbsoluteError OR bMoveAbsoluteCommandAborted) THEN
        bMoveAbsoluteExecute := FALSE;

        seq1State := SEQDEMOSTATE.SEQ1_FAULT;
    END_IF
END_CASE
```

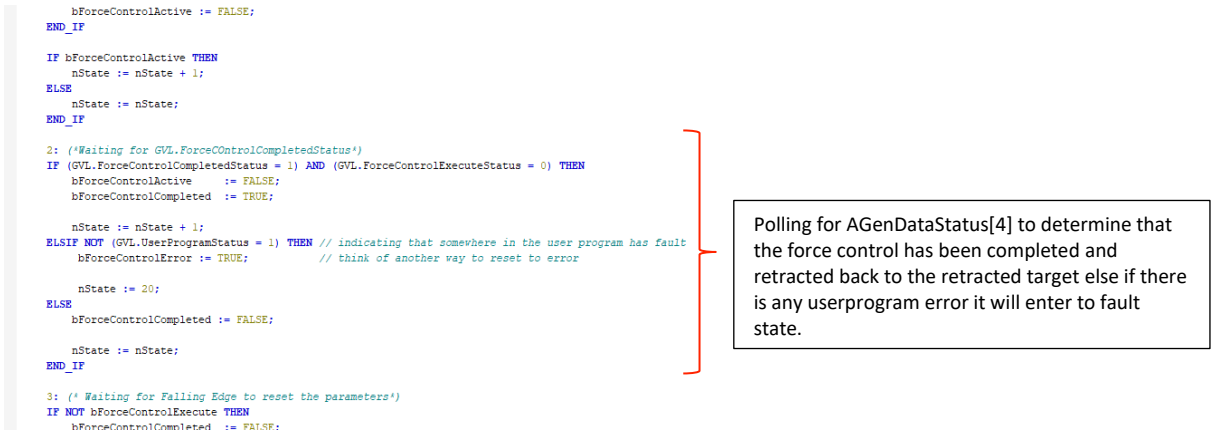
Absolute Move to Target Point 2 / datum point.

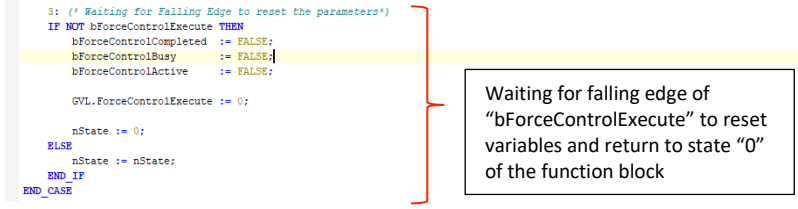
2.3.3.1 Customized Function Block

The example shown above is the “MAIN Program”. This section will show how the AGenData and AGenDataStatus can be integrated inside the customized function block. Only “AGenDataForceControl”, “AlignMasterSlave” and “UserProgramReset” will be shown in this section. The rest of the function blocks are using standard PLCOpen function blocks.

AGenDataForceControl







AlignMasterSlave

```
FUNCTION_BLOCK AlignMasterSlave
VAR_INPUT
    bAlignMasterSlaveExecute        : BOOL; // tie with agendata[3], not physically link them but wer can control where we read this booleans and control teh GVL.XX values
END_VAR
VAR_OUTPUT
    bAlignMasterSlaveCompleted      : BOOL;
    bAlignMasterSlaveBusy           : BOOL;
    bAlignMasterSlaveActive         : BOOL;
    bAlignMasterSlaveError          : BOOL;
END_VAR
VAR
    fbExecute                       : R_TRIG;
```

VAR_INPUT / VAR_OUTPUT

```
fbExecute(CLK := bAlignMasterSlaveExecute);
fbAlignMasterSlaveDone (CLK := bAlignMasterSlaveActive);

CASE nState OF
    0: (*INIT*)
    IF NOT bAlignMasterSlaveExecute THEN
        bAlignMasterSlaveCompleted  := FALSE;
        bAlignMasterSlaveBusy       := FALSE;
        bAlignMasterSlaveActive     := FALSE;
    END_IF

    IF fbExecute.Q THEN
        bAlignMasterSlaveBusy       := TRUE;

        nState := nState +1;
    END_IF

    1: (*Toggling GVL.EnableCSPmotionMode to align back with master , but first master will have to do a virtual move to the axis current position first before calling this function*)
        // Or give a TargetPosition to move to let say Point A TargetPosition is 1000. Do AbsMove virtually before triggering this function on the master, and in the user program trigger a absolute m
        // to Point A Target Position
    GVL.EnableCSPMotionMode := 1;

    IF (GVL.EnableCSPMotionModeStatus = 1) THEN (*Currently switching to drive to CSP Mode*)
        bAlignMasterSlaveActive     := TRUE;
    ELSE
        bAlignMasterSlaveActive     := FALSE;
    END_IF
```

Initialization / Polling for Trigger

Assigning AGenData[3] =1

```
        // Or give a TargetPosition to move to let say Point A TargetPosition is 1000. Do AbsMove virtually before triggering this function on the master, and in the user program trigger a ab
        // to Point A Target Position
    GVL.EnableCSPMotionMode := 1;

    IF (GVL.EnableCSPMotionModeStatus = 1) THEN (*Currently switching to drive to CSP Mode*)
        bAlignMasterSlaveActive     := TRUE;
    ELSE
        bAlignMasterSlaveActive     := FALSE;
    END_IF

    IF fbAlignMasterSlaveDone.Q THEN
        bAlignMasterSlaveCompleted  := TRUE;

        nState := nState + 1;
    ELSIF NOT (GVL.UserProgramStatus = 1) THEN
         bAlignMasterSlaveError := TRUE;

         nState := 20;
    ELSE
        nState := nState;
    END_IF

    2:
    IF NOT bAlignMasterSlaveExecute THEN
        bAlignMasterSlaveCompleted  := FALSE;
        bAlignMasterSlaveBusy       := FALSE;
        bAlignMasterSlaveActive     := FALSE;
```

Checking for AGenDataStatus[3] to indicate “Active” and also checking the falling edge of “Active” to indicate that it has been completed.

```
    2:
    IF NOT bAlignMasterSlaveExecute THEN
        bAlignMasterSlaveCompleted  := FALSE;
        bAlignMasterSlaveBusy       := FALSE;
        bAlignMasterSlaveActive     := FALSE;

        GVL.EnableCSPMotionMode     := 0;

        nState := 0;
    END_IF
END_CASE
```

Waiting for falling edge of “bAlignMasterSlaveExecute” to reset variables and return to state “0” of the function block.

UserProgramReset

```
FUNCTION_BLOCK UserProgramReset
VAR_INPUT
    bUserProgramResetExecute        : BOOL;
END_VAR
VAR_OUTPUT
    bUserProgramResetCompleted      : BOOL;
    bUserProgramResetBusy           : BOOL;
    bUserProgramResetActive         : BOOL;
    bUserProgramResetError          : BOOL;
END_VAR
VAR
    fbExecute                       : R_TRIG;
```

VAR_INPUT / VAR_OUTPUT

```
fbExecute(CLK := bUserProgramResetExecute);
fbUserProgramResetDone(CLK := bUserProgramResetActive);

CASE nState OF
    0: (*INIT*)
    IF NOT bUserProgramResetExecute THEN
        bUserProgramResetCompleted  := FALSE;
        bUserProgramResetBusy       := FALSE;
        bUserProgramResetActive     := FALSE;
    END_IF

    IF fbExecute.Q THEN
        bUserProgramResetBusy       := TRUE;

        nState := nState + 1;
    END_IF

    1:
    GVL.UserProgramReset := 1;

    IF GVL.UserProgramResetStatus = 1 THEN
        bUserProgramResetActive := TRUE;
    ELSE
        bUserProgramResetActive := FALSE;
    END_IF

    IF bUserProgramResetActive THEN
        nState := nState + 1;
    ELSE
        nState := nState;
    END_IF

    2:
    IF GVL.UserProgramStatus = 1 THEN
        bUserProgramResetCompleted  := TRUE;

        nState := nState + 1;
    ELSE
        bUserProgramResetCompleted := FALSE;

        nState := nState;
    END_IF

    3: (* Waiting for Falling Edge to reset the parameters*)
    IF NOT bUserProgramResetExecute THEN
        bUserProgramResetCompleted  := FALSE;
        bUserProgramResetBusy       := FALSE;
        bUserProgramResetActive     := FALSE;

        GVL.UserProgramReset := 0;

        nState := 0;
    ELSE
        nState := nState;
    END_IF
END_CASE
```

Initialization / Polling for Trigger

Assigning AGenData[6] =1

Checking for AGenDataStatus[6] to indicate “Active” and also checking the falling edge of “Active” to indicate that it has been completed.

Waiting for falling edge of “bUserProgramResetExecute” to reset variables and return to state “0” of the function block.

3 Using PLCOpen Torque Control (CST Mode)

The conventional way to do force control over EtherCAT is to use the provided Torque Control function block in the PLCOpen. The difference between using this method and the method above is that the sampling of the switching of mode is tied to the cycle time of the PLC task whereas the User Program method is sampling at the driver interrupt which is at 62.5us. There are certain conditions or SDO objects to be set before enabling the torque control to enter current/force control mode in the drive and that will be covered in the description of related EtherCAT Objects.

3.1 Description of Related EtherCAT Objects

This section touches on the related EtherCAT Objects (on top of the mandatory ones) that can be / should be mapped to the PDO to ensure cyclic exchanging of information between the master and controller. The mandatory EtherCAT Objects that should have been mapped are “StatusWord”, “Position Actual Value”, “ControlWord” and “Target Position”.

3.1.1 Object 2410_h: Analog Input 1 Value

This object shall indicate the value that is connected to the analog input port of the driver i.e. force sensor. Calibration should have been done on the PCSuite UI. The value that is reported back will be in terms of force units.

Attribute	Value
Sub-Index	00_h
Access	rw
PDO Mapping	Yes
Value Range	Interger32
Default Value	00000000_h

3.1.2 Object 2810_h: CST Force Loop Enable

This object shall indicate if current / force mode is to be used. “0” indicates that current mode shall be used and “1” indicates force mode. By default, the value of the object is set as “0”, if user has analog force feedback connected, this object should set as “1”.

This object can be set through SDO and does not be to be included in the cyclic exchange. If the master allows startup configurations, this object can be set during the start-up parameters.

Attribute	Value
Sub-Index	00_h
Access	rw
PDO Mapping	Yes
Value Range	Unsigned8
Default Value	00_h

3.1.3 Object 2811_h: CST Command Interpolation

This object shall indicate the configured value to be used to interpolate the CST Command when force mode is enabled. By default, the value of the object is set as “1”, if the user finds that 1mv/10 force units is too coarse, they can further interpolate by changing this to a value of “2” or any other value. It is scaled down by 1/ (values set in this object).

This object can be set through SDO and does not be to be included in the cyclic exchange. If the master allows startup configurations, this object can be set during the start-up parameters.

Attribute	Value
Sub-Index	00_h
Access	rw
PDO Mapping	Yes
Value Range	Unsigned8
Default Value	00_h

3.1.4 Object 6071_h: Target Torque

This object shall indicate the configured input value for the torque controller in profile torque mode/cst mode. The value shall be given per thousand of rated torque.

If current mode (0x2810: “CST Force Loop Enable” = 0) is selected, the Target Torque will be given per thousand of the continuous torque/current set in the driver. (E.g. if the continuous current in the driver is set as 2000mA, 5.0% of Target Torque would indicate a current reference of 100mA and 0.1% of it would indicate 2mA).

If force mode (0x2810: “CST Force Loop Enable” = 1) is selected, the Target Torque will be given per thousand of +- 10V (10000mV) as normally there will be an analog force feedback connected to the driver and during the initial set up it has already done certain calibration to indicate that 1mV represents 1g/5g or 10mv/ represents 1g/5g. The force unit is not necessary to be in g, it will be according to the calibration that was set. Hence, 5% of the Target Torque would give a force reference of 500 force units and 0.1% of it would give 10 force units. It is also possible to scale down even more by changing the scaling factor (0x2810: “CST Command Interpolation”)

Attribute	Value
Sub-Index	00_h
Access	rw
PDO Mapping	Yes
Value Range	Interger16
Default Value	0000_h

3.1.5 Object 6077_h: Torque Actual Value

This object shall provide the actual value of the torque. It shall correspond to the instantaneous torque in the motor. The value shall be given per thousand of rated torque. Currently, there is no torque feedback in our drive and since T = kt * I and assuming that the motor constant is static, the calculation of the torque feedback will be in terms of the percentage of the continuous current that is currently used.

If continuous current is set as 2000ma and 500mA is currently being used, a value of 25% would be reported back in this object.

If force mode is used, please refer to object 0x2410 “Analog Input Value 1” to see the force units that is being reported back.

Attribute	Value

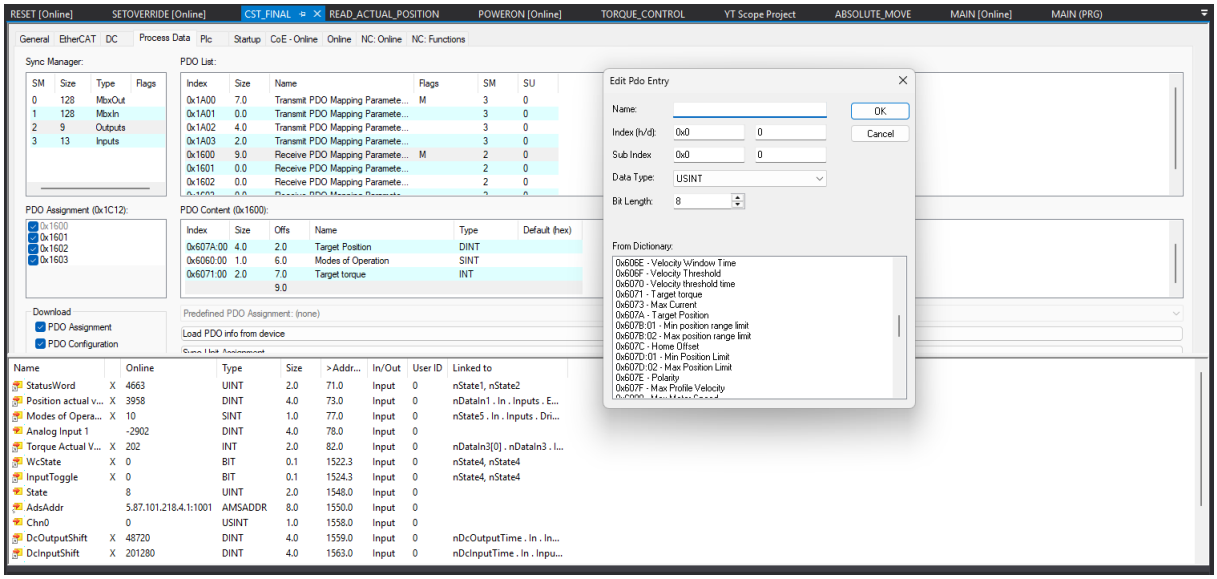
Sub-Index	00_h
Access	rw
PDO Mapping	Yes
Value Range	Interger16
Default Value	0000_h

3.2 TwinCAT / PLC Programming Example

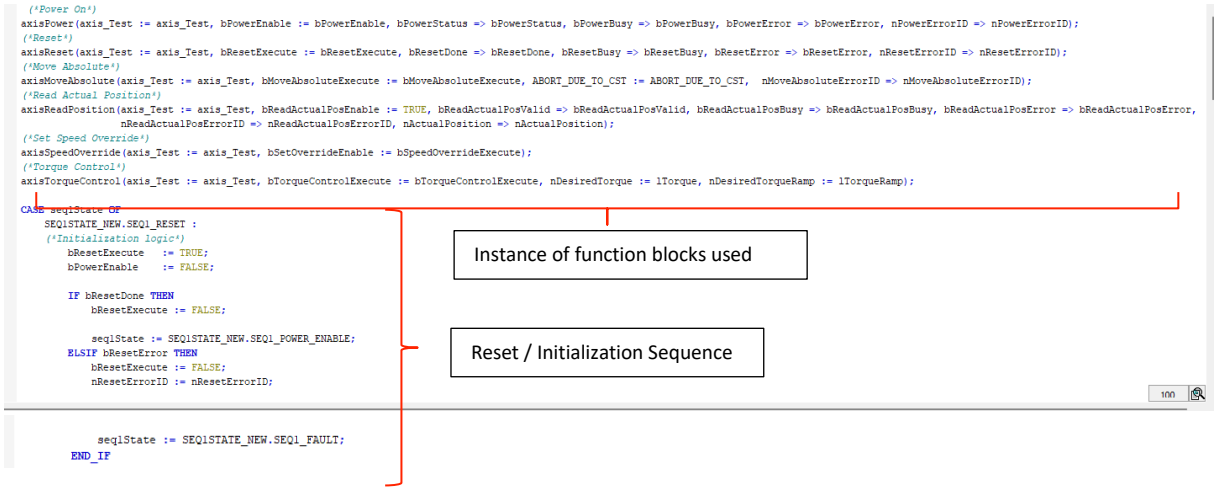
3.2.1 CST Current Loop

3.2.1.1 Modifying PDO List

It is important to map the Target Torque ($0x6071_h$) and Torque Actual Value ($0x6077_h$) object into the PDO list to ensure the cyclic exchange of information between the master and the slave. Descriptions of the individual objects are described in the section above and how they are being used in different modes. Target Torque ($0x6071_h$) and Torque Actual Value ($0x6077_h$) are not mapped in the PDO list by default hence users have to manually add the objects into the PDO configuration list.



3.2.1.2 Writing POU



```
SEQ1STATE_NEW.SEQ1_POWER_ENABLE:
(*Axis Power*)
    bPowerEnable := TRUE;

    IF bPowerStatus THEN
        seq1State := SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_1;
    ELSIF bPowerError THEN
        bPowerEnable := FALSE;

        seq1State := SEQ1STATE_NEW.SEQ1_FAULT;
    END_IF
```

Axis Power

```
SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_1:
(*Setting Target 1 Parameters*)
    axisMoveAbsolute.nTargetPosition := nTargetPosition_1;
    axisMoveAbsolute.nTargetVelocity := nTargetVelocity_1;
    axisMoveAbsolute.nTargetAcceleration := nTargetAcceleration_1;
    axisMoveAbsolute.nTargetDeceleration := nTargetDeceleration_1;
    axisMoveAbsolute.nTargetJerk := nTargetJerk_1;

    bMoveAbsoluteExecute := TRUE;

    seq1State := SEQ1STATE_NEW.SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_1;
```

Setting Target Position past the contact point position, will use speed override to slow the speed down after crossing some position threshold.

```
SEQ1STATE_NEW.SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_1:
    IF axisReadPosition.nActualPosition > -1000 THEN //-100000 THEN
        axisSpeedOverride.nVelocityFactor := nVelocityFactorChange_Slow;
```

```
SEQ1STATE_NEW.SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_1:
    IF axisReadPosition.nActualPosition > -1000 THEN //-100000 THEN
        axisSpeedOverride.nVelocityFactor := nVelocityFactorChange_Slow;
        bSpeedOverrideExecute := TRUE;
    END_IF

    IF axisSpeedOverride.bSetOverrideBusy AND axisSpeedOverride.bSetOverrideEnabled THEN
        seq1State := SEQ1STATE_NEW.SEQ1_CST_MODE;
    END_IF
```

Reading the axis actual position and trigger speed override upon crossing the position. Users have to set this area to ensure that there is enough stroke to let it travel at slow speed.

```
SEQ1STATE_NEW.SEQ1_CST_MODE:
    IF axis_Test.NcToPlc.ActTorque >= 20 THEN
        bTorqueControlExecute := TRUE;
        ABORT_DUE_TO_CST := TRUE;
        bMoveAbsoluteExecute  := FALSE;
        bSpeedOverrideExecute := FALSE;
        seq1State := SEQ1STATE_NEW.SEQ1_CST_FORCE_HOLDING_WAIT;
    END_IF
```

Polling the ActTorque that the PLC reads and if its more than 20% then trigger the MC_TorqueControl function block. Inside the MC_TorqueControl, there is a manual enable start option. If it is FALSE, the TargetTorque command will start from 0, else it will start from the point it captured the ActTorque.

```
SEQ1STATE_NEW.SEQ1_CST_FORCE_HOLDING_WAIT:
    IF axisTorqueControl.bTorqueCommandDone AND NOT axisTorqueControl.bTorqueControlActive THEN
        bTorqueControlExecute := FALSE;
        ABORT_DUE_TO_CST := FALSE;
        seq1State := SEQ1STATE_NEW.SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_2;
    END_IF

SEQ1STATE_NEW.SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_2:
```

Inside the MC_TorqueControl function block, a hold timer was added after the Set Torque have been reached before asserting TorqueCommandDone.

```
SEQ1STATE_NEW.SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_2:
    axisSpeedOverride.nVelocityFactor := nVelocityFactorChange_Fast;
    bSpeedOverrideExecute := TRUE;

    IF axisSpeedOverride.bSetOverrideBusy AND axisSpeedOverride.bSetOverrideEnabled THEN
        bSpeedOverrideExecute := FALSE;
        seq1State := SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_2;
    END_IF
```

Resetting the speed override to the original velocity factor to retract the axis at a faster speed.

```
SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_2:
(*Setting Target 2 Parameters*)
    axisMoveAbsolute.nTargetPosition := nTargetPosition_2;
    axisMoveAbsolute.nTargetVelocity := nTargetVelocity_2;
    axisMoveAbsolute.nTargetAcceleration := nTargetAcceleration_2;
    axisMoveAbsolute.nTargetDeceleration := nTargetDeceleration_2;
    axisMoveAbsolute.nTargetJerk := nTargetJerk_2;

    bMoveAbsoluteExecute := TRUE;

seq1State := SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_2_WAIT;
```

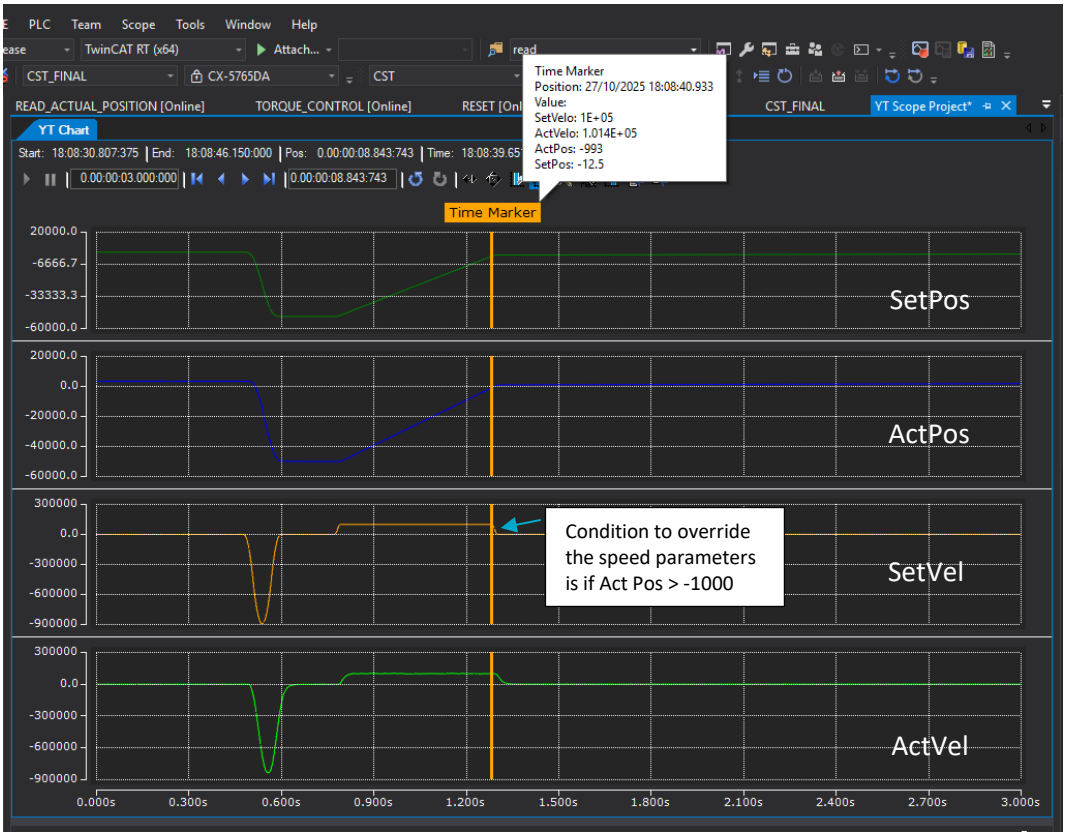
Retracting to Target Position 2.

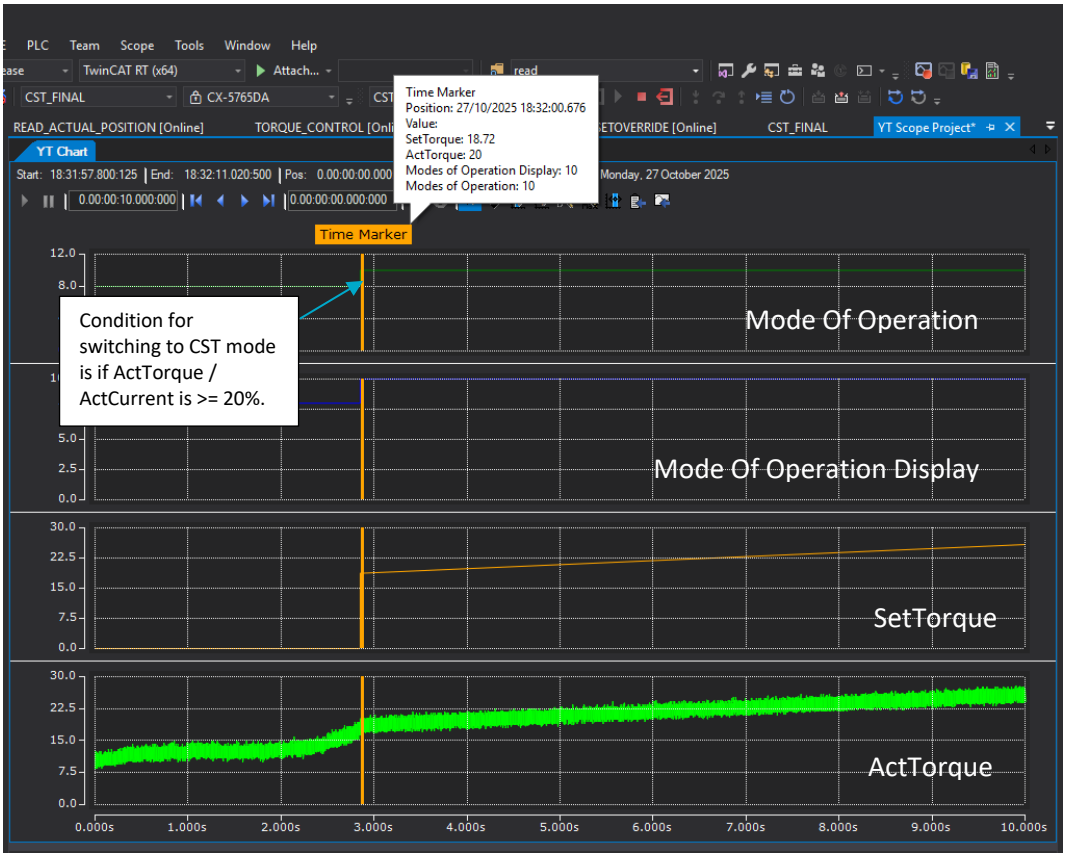
```
SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_2_WAIT:
    IF NOT axisMoveAbsolute.bMoveAbsoluteBusy AND axisMoveAbsolute.bMoveAbsoluteDone THEN
        bMoveAbsoluteExecute := FALSE;

        seq1State := SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_1;
    ELSIF NOT axisMoveAbsolute.bMoveAbsoluteBusy AND axisMoveAbsolute.bMoveAbsoluteError THEN
        nMoveAbsoluteErrorID := axisMoveAbsolute.nMoveAbsoluteErrorID;

        seq1State := SEQ1STATE_NEW.SEQ1_FAULT;
    END_IF


SEQ1STATE_NEW.SEQ1_FAULT:
    bPowerEnable := FALSE;

END_CASE
```

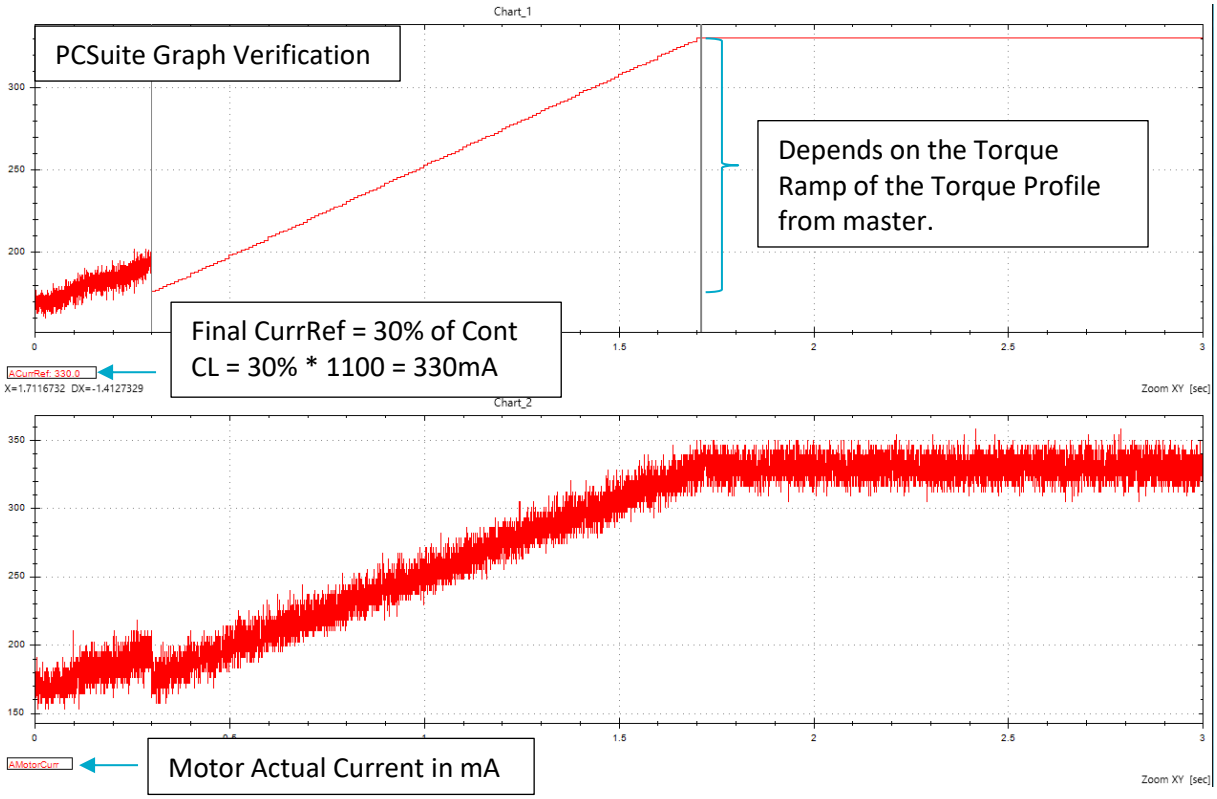
Waiting for done signal.

End of cycle / back to the start of the force control cycle.

3.2.1.3 Verification







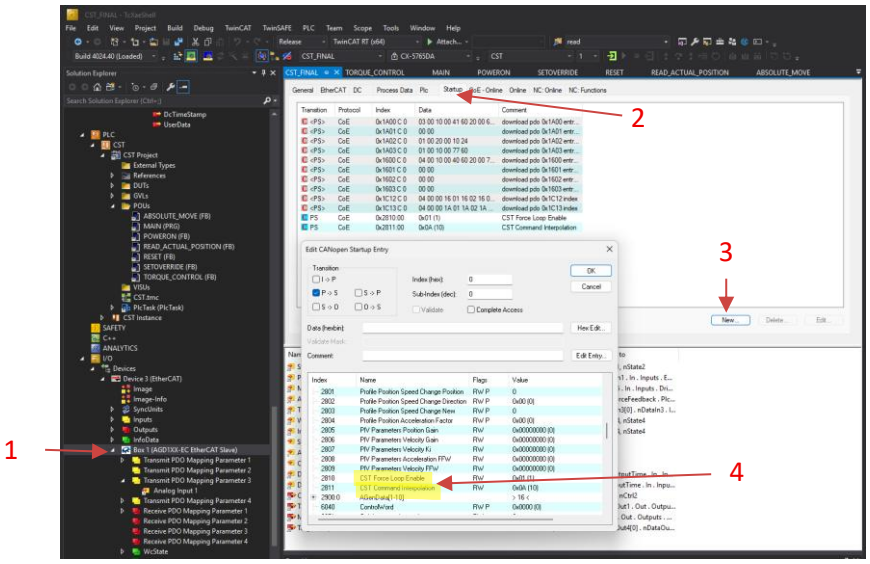
3.2.2 CST Force Loop

3.2.2.1 Modifying PDO List

Similar to CST Current Loop, it is necessary to map the objects that are mentioned above as well as **Analog Input 1 Value (0x2410$_h$)** into the PDO object list as this object will be used as the force feedback of the force sensor.

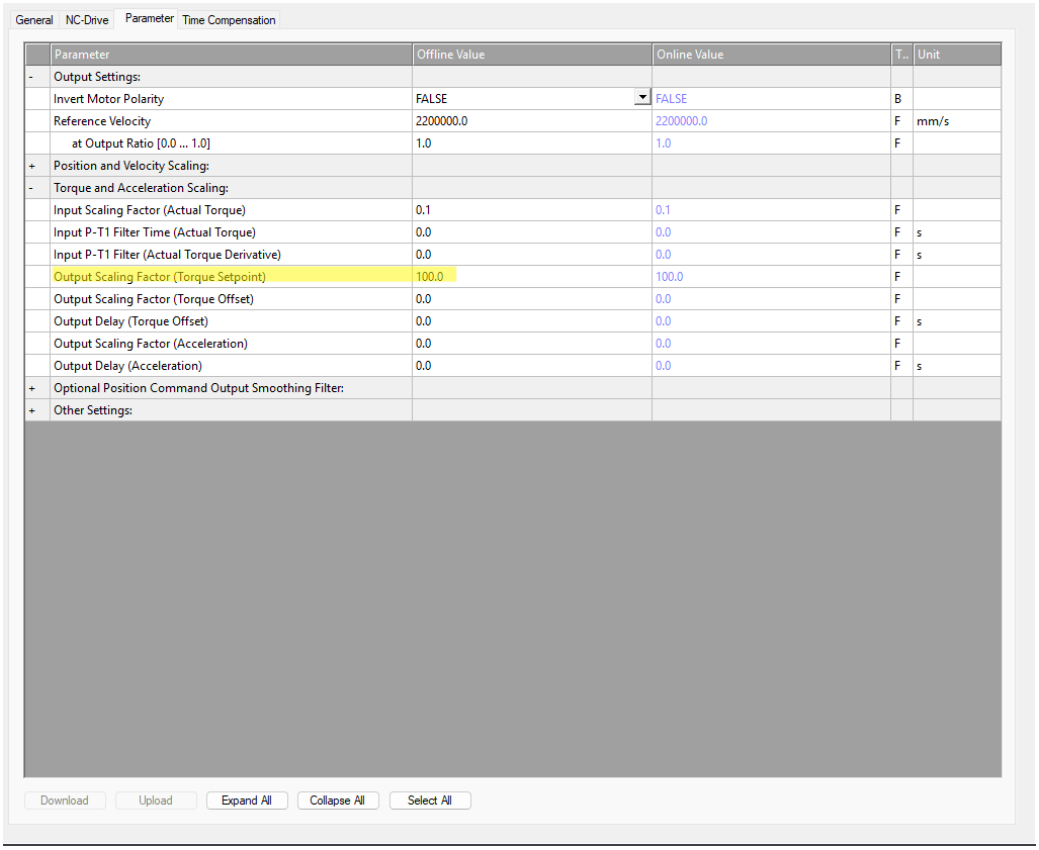
3.2.2.2 Startup Parameters

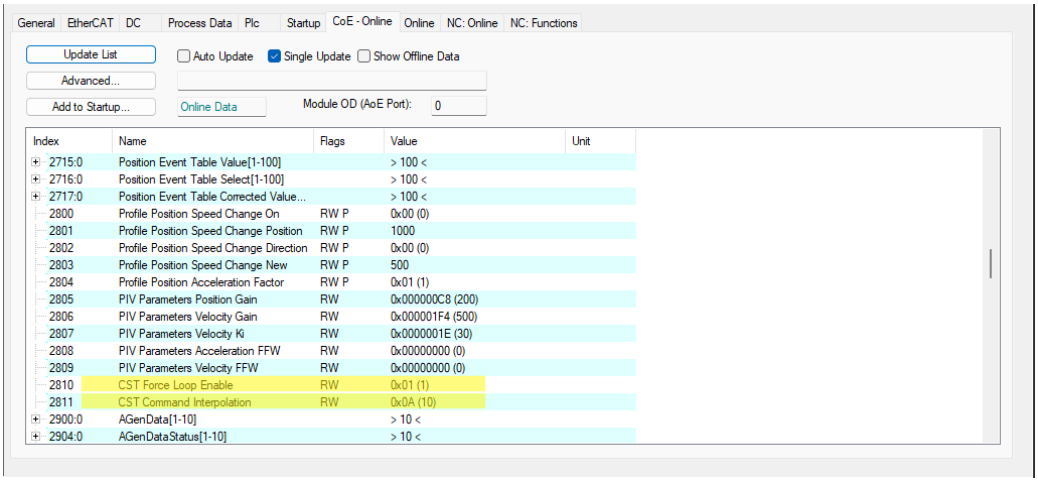
If the master is able to pre-configure some start up parameters before going into OP mode, they can pre-configure objects **CST Force Loop Enable (0x2810$_h$) and CST Command Interpolation (0x2811$_h$)** in the startup parameters page shown below.



3.2.2.3 Modifying Output Scaling Factor / CST Interpolation Command Factor

By default, the torque output scaling factor is set at 10.0 on the master. This output scaling will in turn use the % command from the MC_TorqueControl function block and scale it to feed into Target Torque Object ($0x6071_h$). If the user determines that it is too coarse i.e. the force feedback that the user wants to switch at is 50 force units but due to jitter the manual start torque of the MC_TorqueControl function block captured at 48 force units, this will in turn translate to 0.48% and the 0.08% portion will be truncated. Hence, the user can bump up the torque output scaling factor to 100.0 (on the master) and also scale down on the driver side as well by a factor of 10 by inputting "10" in the CST Command Interpolation ($0x2811_h$) object.





3.2.2.4 Writing POU

```
(*Power On*)
axisPower(axis_Test := axis_Test, bPowerEnable := bPowerEnable, bPowerStatus => bPowerStatus, bPowerBusy => bPowerBusy, bPowerError => bPowerError, nPowerErrorID => nPowerErrorID);
(*Reset*)
axisReset(axis_Test := axis_Test, bResetExecute := bResetExecute, bResetDone => bResetDone, bResetBusy => bResetBusy, bResetError => bResetError, nResetErrorID => nResetErrorID);
(*Move Absolute*)
axisMoveAbsolute(axis_Test := axis_Test, bMoveAbsoluteExecute := bMoveAbsoluteExecute, ABORT_DUE_TO_CST := ABORT_DUE_TO_CST,  nMoveAbsoluteErrorID => nMoveAbsoluteErrorID);
(*Read Actual Position*)
axisReadPosition(axis_Test := axis_Test, bReadActualPosEnable := TRUE, bReadActualPosValid => bReadActualPosValid, bReadActualPosBusy => bReadActualPosBusy, bReadActualPosError => bReadActualPosError,
                 nReadActualPosErrorID => nReadActualPosErrorID, nActualPosition => nActualPosition);
(*Set Speed Override*)
axisSpeedOverride(axis_Test := axis_Test, bSetOverrideEnable := bSpeedOverrideExecute);
(*Torque Control*)
axisTorqueControl(axis_Test := axis_Test, bTorqueControlExecute := bTorqueControlExecute, nDesiredTorque := lTorque, nDesiredTorqueRamp := lTorqueRamp);

CASE seq1State OF
    SEQ1STATE_NEW.SEQ1_RESET :
    (*Initialization logic*)
        bResetExecute   := TRUE;
        bPowerEnable    := FALSE;

        IF bResetDone THEN
            bResetExecute := FALSE;

            seq1State := SEQ1STATE_NEW.SEQ1_POWER_ENABLE;
        ELSIF bResetError THEN
            bResetExecute := FALSE;
            nResetErrorID := nResetErrorID;

            seq1State := SEQ1STATE_NEW.SEQ1_FAULT;
        END_IF

    SEQ1STATE_NEW.SEQ1_POWER_ENABLE:
    (*Axis Power*)
        bPowerEnable := TRUE;

        IF bPowerStatus THEN



        IF bPowerStatus THEN
            seq1State := SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_1;
        ELSIF bPowerError THEN
            bPowerEnable := FALSE;

            seq1State := SEQ1STATE_NEW.SEQ1_FAULT;
        END_IF

SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_1:
(*Setting Target 1 Parameters*)
    axisMoveAbsolute.nTargetPosition := nTargetPosition_1;
    axisMoveAbsolute.nTargetVelocity := nTargetVelocity_1;
    axisMoveAbsolute.nTargetAcceleration := nTargetAcceleration_1;
    axisMoveAbsolute.nTargetDeceleration := nTargetDeceleration_1;
    axisMoveAbsolute.nTargetJerk := nTargetJerk_1;

    bMoveAbsoluteExecute := TRUE;

    seq1State := SEQ1STATE_NEW.SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_1;

SEQ1STATE_NEW.SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_1:
    IF axisReadPosition.nActualPosition > -1000 THEN //-100000 THEN
        axisSpeedOverride.nVelocityFactor := nVelocityFactorChange_Slow;
        bSpeedOverrideExecute := TRUE;
    END_IF

    IF axisSpeedOverride.bSetOverrideBusy AND axisSpeedOverride.bSetOverrideEnabled THEN
        seq1State := SEQ1STATE_NEW.SEQ1_CST_MODE;
    END_IF

SEQ1STATE_NEW.SEQ1_CST_MODE:
    IF GVL.ForceFeedback >= 50 THEN //axis_Test.NcToPlc.ActTorque >= 20.0 THEN
        bTorqueControlExecute := TRUE;
        ABORT_DUE_TO_CST := TRUE;



        ABORT_DUE_TO_CST := TRUE;
        bMoveAbsoluteExecute  := FALSE;
        bSpeedOverrideExecute := FALSE;
        seq1State := SEQ1STATE_NEW.SEQ1_CST_FORCE_HOLDING_WAIT;//SEQ1STATE_NEW.SEQ1_CST_FORCE_HOLDING_WAIT;
    END_IF

SEQ1STATE_NEW.SEQ1_CST_FORCE_HOLDING_WAIT:
    IF axisTorqueControl.bTorqueCommandDone AND NOT axisTorqueControl.bTorqueControlActive THEN
        bTorqueControlExecute := FALSE;
        ABORT_DUE_TO_CST := FALSE;
        seq1State := SEQ1STATE_NEW.SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_2;
    END_IF

SEQ1STATE_NEW.SEQ1_OVERRIDE_SPEED_PARAMETERS_TARGET_2:
    axisSpeedOverride.nVelocityFactor := nVelocityFactorChange_Fast;
    bSpeedOverrideExecute := TRUE;

    IF axisSpeedOverride.bSetOverrideBusy AND axisSpeedOverride.bSetOverrideEnabled THEN
        bSpeedOverrideExecute := FALSE;
        seq1State := SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_2;
    END_IF

SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_2:
(*Setting Target 2 Parameters*)
    axisMoveAbsolute.nTargetPosition := nTargetPosition_2;
    axisMoveAbsolute.nTargetVelocity := nTargetVelocity_2;
    axisMoveAbsolute.nTargetAcceleration := nTargetAcceleration_2;
    axisMoveAbsolute.nTargetDeceleration := nTargetDeceleration_2;
    axisMoveAbsolute.nTargetJerk := nTargetJerk_2;

    bMoveAbsoluteExecute := TRUE;

seq1State := SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_2_WAIT;

SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_2_WAIT:
    IF NOT axisMoveAbsolute.bMoveAbsoluteBusy AND axisMoveAbsolute.bMoveAbsoluteDone THEN
```

Instance of function blocks used

Reset / Initialization Sequence

Axis Power

Setting Target Position past the contact point position, will use speed override to slow the speed down after crossing some position threshold.

Reading the axis actual position and trigger speed override upon crossing the position. Users have to set this area to ensure that there is enough stroke to let it travel at slow speed.

Polling the Analog Input Feedback and if its more than 50 force units then trigger the MC_TorqueControl function block. Inside the MC_TorqueControl, there is a manual enable start option. If it is FALSE, the TargetTorque command will start from 0, else it will start from the point it captured the in terms of %.

Inside the MC_TorqueControl function block, a hold timer was added after the Set Torque have been reached before asserting TorqueCommandDone.

Resetting the speed override to the original velocity factor to retract the axis at a faster speed.

Retracting to Target Position 2.

```
        IF NOT axisMoveAbsolute.bMoveAbsoluteBusy AND axisMoveAbsolute.bMoveAbsoluteDone THEN
            bMoveAbsoluteExecute := FALSE;

            seq1State := SEQ1STATE_NEW.SEQ1_ABSOLUTE_MOVE_TARGET_1;
        ELSIF NOT axisMoveAbsolute.bMoveAbsoluteBusy AND axisMoveAbsolute.bMoveAbsoluteError THEN
            nMoveAbsoluteErrorID := axisMoveAbsolute.nMoveAbsoluteErrorID;

            seq1State := SEQ1STATE_NEW.SEQ1_FAULT;
        END_IF


    SEQ1STATE_NEW.SEQ1_FAULT:
        bPowerEnable := FALSE;


END_CASE
```

Waiting for done signal.

End of cycle / back to the start of the force control cycle.

```
fbExecute(CLK:= bTorqueControlExecute);

CASE nState OF
    0:
        IF NOT bTorqueControlExecute THEN
            bTorqueCommandDone := FALSE;
            bInTorque := FALSE;
            bTorqueControlBusy := FALSE;
            bTorqueControlActive := FALSE;
            bTorqueControlCommandAborted := FALSE;
            bTorqueControlError := FALSE;
            nTorqueControlErrorID := 0;
            DelayTimer(IN:=FALSE);
        END_IF

        IF fbExecute.Q THEN
            nState := nState + 1;
        END_IF

    1:
        Options.EnableManualTorqueStartValue := TRUE;
        Options.ManualTorqueStartValue := (GVL.ForceFeedback * 0.01);//axis_Test.NcToPlc.ActTorque;

        fbTorqueControl(
            Axis                := axis_Test,
            Execute             := bTorqueControlExecute,
            ContinuousUpdate    := bTorqueControlContinuousUpdate,
            Relative            := bTorqueControlRelative,
            Torque              := nDesiredTorque,
            TorqueRamp          := nDesiredTorqueRamp,
            VelocityLimitHigh   := nVelocityLimitHigh,
            VelocityLimitLow    := nVelocityLimitLow,
            Options             := Options);

        IF fbTorqueControl.Active THEN
            bTorqueControlActive := fbTorqueControl.Active;
            nState := nState + 1;
        ELSIF fbTorqueControl.Error THEN
```

Inside the MC_TorqueControl Function Block

Scaled the Force Feedback depending on “CST Command Interpolation” or the Output Torque Scaling on the master.

On the topic of scaling for Force Feedback, CST command interpolation and also Output Torque Scaling (not all masters support this option).

In the case of default scaling of CST Command Interpolation and Output Torque Scaling:

If CST Command Interpolation is default (value of 1)
and Output Target Torque Scaling is default (value of 10)

1% of the commanded "Torque" would give a Target Torque (0x6071)
value of 10 due to the default output target torque scaling (default value of 10)

Analog Input represents 1mv/g (depends on the setting of the force sensor and also calibration)

Equation of Force Ref = TargetTorque - TargetTorquePrev * 0.001 (in terms of thousand of percentage) * 10000 (+- 10V range) * (1/CST_Command_Interpolation) + fractions (if any)

So a value of 1% equates to Target Torque 10 = Force Ref of 100 Force units
Hence, it is needed to scale the force feedback by a factor of 0.01 inside the MC_TorqueControl -> ManualTorqueStartValue

Switching to CST Condition = 50g = 50 * 0.01 = 0.5% on the PLC giving a Target Torque Value of 5.
However, due to jitter it could have capture the start value at 0.48% and due to truncation, the Target Torque Value is input as 4 instead of 4.8 and under these circumstances it is advised to do some scaling to acquire the truncated values as well.

In the case of scaling of CST Command Interpolation and Output Torque Scaling by a factor of 10:

If CST Command Interpolation is scaled by 10 (value of 10)
and Output Target Torque Scaling is scaled by 10 (value of 100)

1% of the commanded "Torque" would give a Target Torque (0x6071)
value of 100 after the output target torque scaling.

Analog Input represents 1mv/g (depends on the setting of the force sensor and also calibration)

Equation of Force Ref = TargetTorque - TargetTorquePrev * 0.001 (in terms of thousand of percentage) * 10000 (+- 10V range) * (1/CST_Command_Interpolation) + fractions (if any)

So a value of 1% equates to Target Torque 100 = Force Ref of 100 Force units (assuming that TargetTorquePrev is 0)
Hence, the scaling set for the force feedback is a factor of 0.01 as well inside the MC_TorqueControl -> ManualTorqueStartValue

Switching to CST Condition = 50g = 50 * 0.01 = 0.5% on the PLC giving a Target Torque Value of 50 (due to scaling).
However, due to jitter it could have capture the start value at 0.48% and it will give a value of Target Torque Value of 48 (due to output target torque scaling)

In the case of scaling of CST Command Interpolation (by 10) and default scaling for Output Torque Scaling:

If CST Command Interpolation is scaled by 10 (value of 10)
and Output Target Torque Scaling is default (value of 10)

1% of the commanded "Torque" would give a Target Torque (0x6071)
value of 10 after the output target torque scaling.

Analog Input represents 1mv/g (depends on the setting of the force sensor and also calibration)

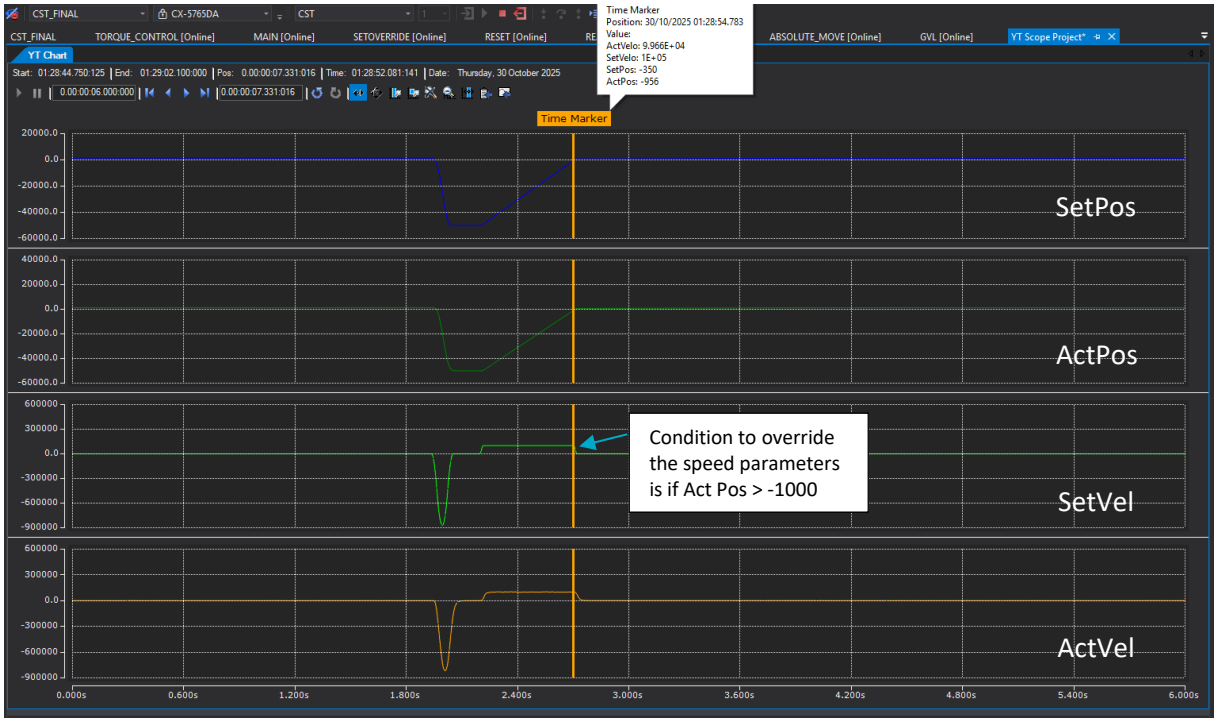
Equation of Force Ref = TargetTorque - TargetTorquePrev * 0.001 (in terms of thousand of percentage) * 10000 (+- 10V range) * (1/CST_Command_Interpolation) + fractions (if any)

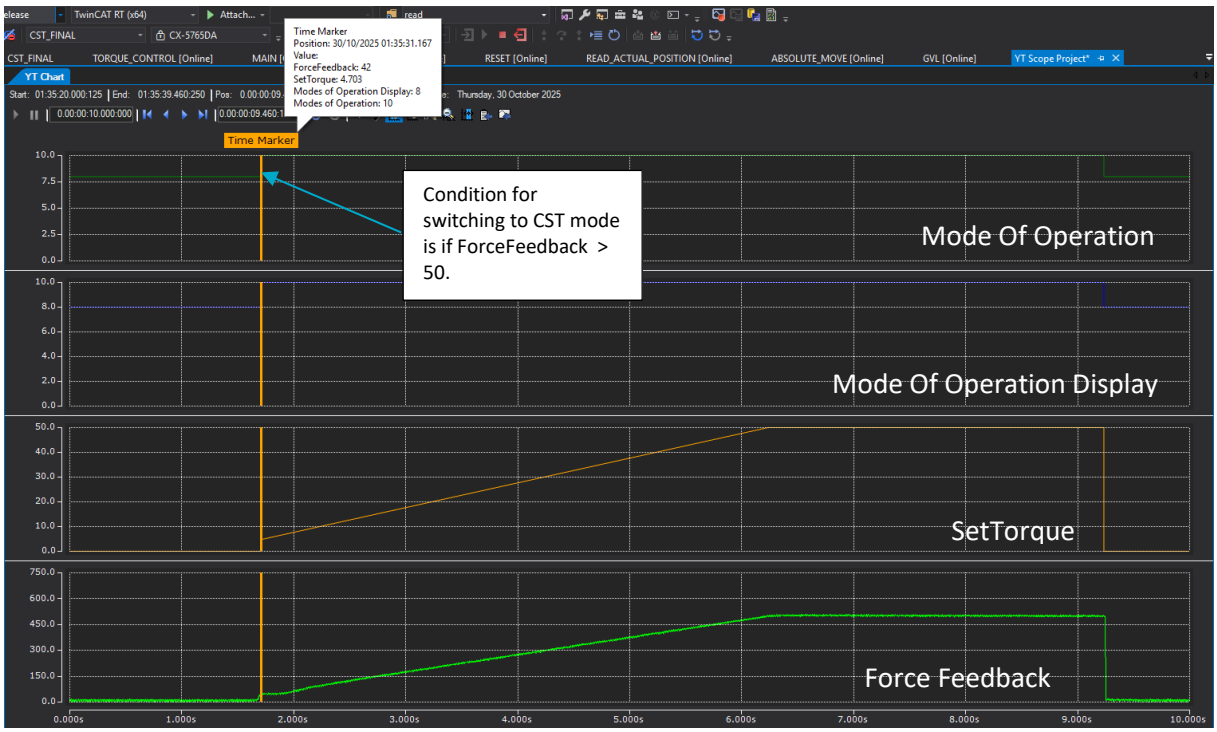
So a value of 1% equates to Target Torque 10 = Force Ref of 10 Force units (assuming that TargetTorquePrev is 0)
Hence, the scaling set for the force feedback is a factor of 0.1 inside the MC_TorqueControl -> ManualTorqueStartValue

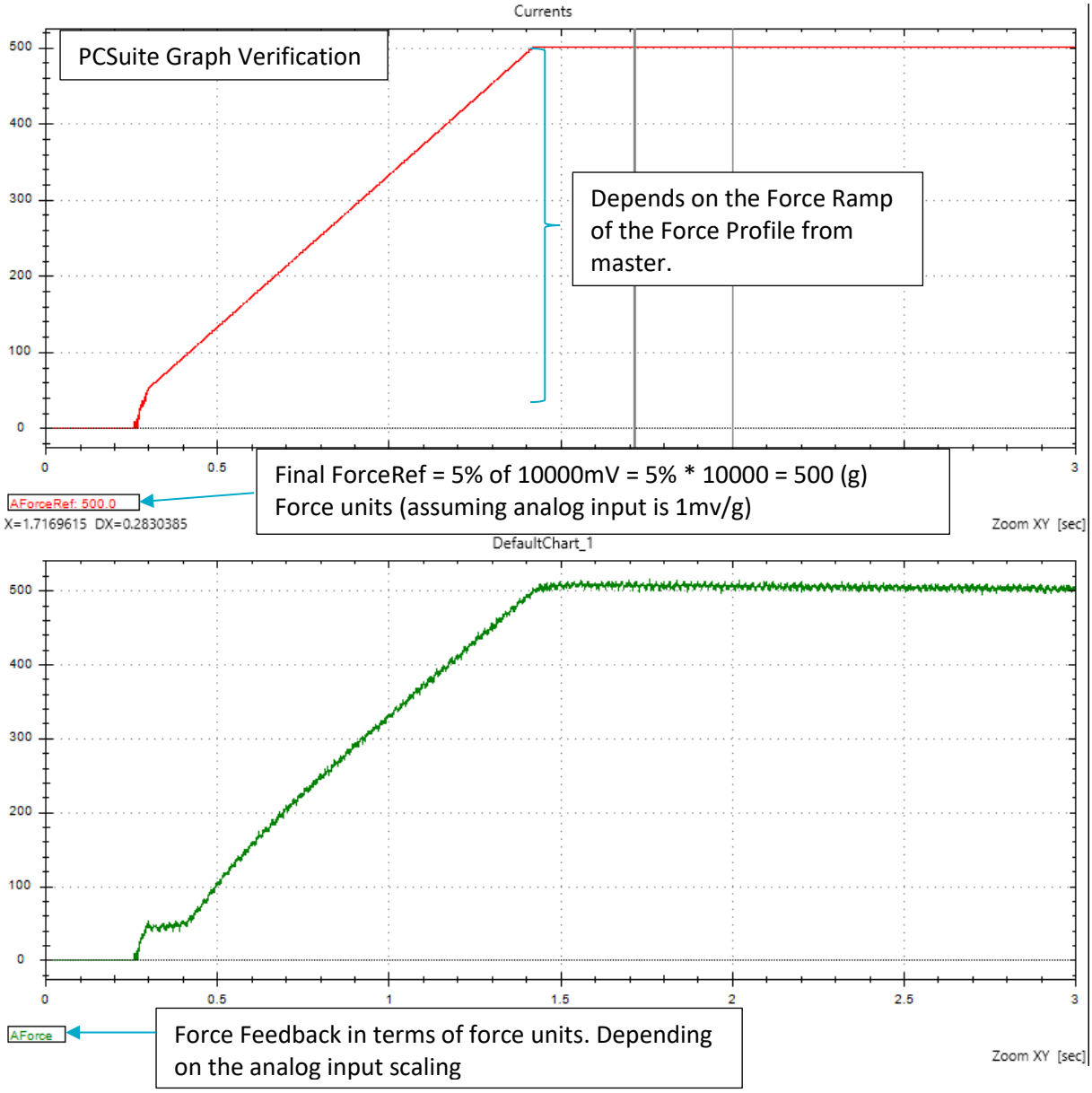
Switching to CST Condition = 50g = 50 * 0.1 = 5% on the PLC giving a Target Torque Value of 50 (due to scaling).
However, due to jitter it could have capture the start value at 4.8% and it will give a value of Target Torque Value of 48 (due to output target torque scaling)

In this case, it is necessary to bump the % Target Torque command by a factor of 10 as well.

3.2.2.5 Verification






Currents
PCSuite Graph Verification
Depends on the Force Ramp of the Force Profile from master.
Final ForceRef = 5% of 10000mV = 5% * 10000 = 500 (g)
Force units (assuming analog input is 1mv/g)
AForceRef: 500.0
X=1.7169615 DX=0.2830385
Zoom XY [sec]
DefaultChart_1
AForce
Force Feedback in terms of force units. Depending on the analog input scaling
Zoom XY [sec]