MathWorks Simulink and C++ integration with the new VLT PLC-based standard development platform for instrument control systems

Mario J. Kiekebusch [1a], Nicola Di Lieto [a], Stefan Sandrock [a], Dan Popovic [a], Gianluca Chiozzi [a]
[a]European Southern Observatory, Karl-Schwarzschild-Strasse 2, D-85748 Garching bei München.

ABSTRACT

ESO is in the process of implementing a new development platform, based on PLCs, for upcoming VLT control systems (new instruments and refurbishing of existing systems to manage obsolescence issues). In this context, we have evaluated the integration and reuse of existing C++ libraries and Simulink models into the real-time environment of BECKHOFF Embedded PCs using the capabilities of the latest version of TwinCAT software and MathWorks Embedded Coder. While doing so the aim was to minimize the impact of the new platform by adopting fully tested solutions implemented in C++. This allows us to reuse the in house expertise, as well as extending the normal capabilities of the traditional PLC programming environments.

We present the progress of this work and its application in two concrete cases: 1) field rotation compensation for instrument tracking devices like derotators, 2) the ESO standard axis controller (ESTAC), a generic model-based controller implemented in Simulink and used for the control of telescope main axes.

Keywords: VLT, Instrument Control, PLC, C++, Simulink, TwinCAT

1. TECHNOLOGY OVERVIEW

BECKHOFF Embedded PCs and EtherCAT is the proposed solution as the new development platform for VLT instrument control systems[1]. This solution includes TwinCAT (TC) software which is the implementation of the EtherCAT master that provides deterministic cyclic access to field inputs and outputs as well as to variables in the traditional IEC 61131-3 languages. The TwinCAT version 3 (TC3) launched in 2010 has been a major upgrade which includes the eXtended Automation Technlogy (XAT) expanding its capabilities with many new powerful functions[3]. Among those new features is the support for high-level languages such as C/C++ and Simulink for real-time applications.

C++ Development Environment

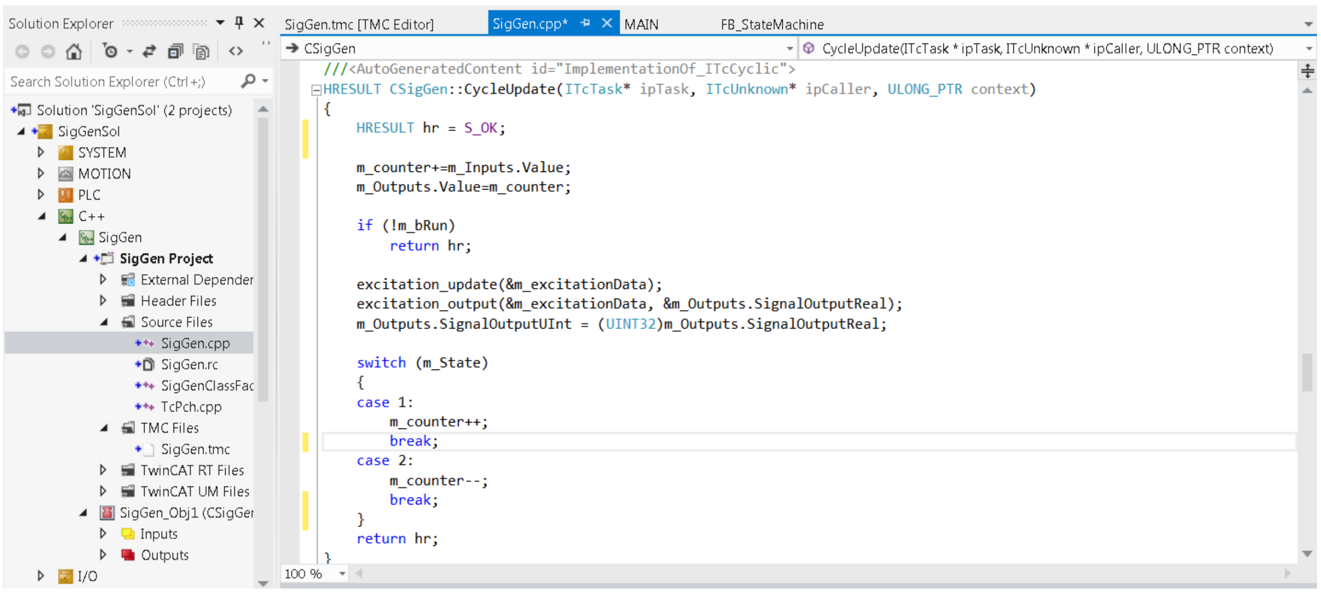
Starting from TC3, the BECKHOFF software is integrated in the *Microsoft Visual Studio (VS)* Integrated Development Environment (IDE). All TC3 components are encapsulated in a so-called "*solution*". Developers can implement PLC and C++ code within the same IDE. Templates are provided to create the various types of projects and files, Figure 1 shows the default directory structure generated when selecting a C++ driver project.

Additional Licenses

The free version of the VS shell that is integrated with the TC3 installation package cannot be used for developing C++ code. The Professional Edition of VS is required in order to be able to compile C++ code for TC3. All our tests were performed using VS Professional 2012. There is also a specific run-time license required for the execution of software developed in C++.

[1] mkiekebu@eso.org; phone +49 89 32006-0; fax: +49 89 320 2362.

Software and Cyberinfrastructure for Astronomy III, edited by Gianluca Chiozzi, Nicole M. Radziwill,
Proc. of SPIE Vol. 9152, 91522B · © 2014 SPIE · CCC code: 0277-786X/14/$18
doi: 10.1117/12.2056381



```
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'SigGenSol' (2 projects)
  SigGenSol
    SYSTEM
    MOTION
    PLC
    C++
      SigGen
        SigGen Project
          External Depender
          Header Files
          Source Files
            SigGen.cpp
            SigGen.rc
            SigGenClassFac
            TcPch.cpp
          TMC Files
            SigGen.tmc
          TwinCAT RT Files
          TwinCAT UM Files
        SigGen_Obj1 (CSigGer
          Inputs
          Outputs
    I/O
```

```
SigGen.tmc [TMC Editor]   SigGen.cpp*   MAIN   FB_StateMachine
CSigGen   CycleUpdate(ITcTask * ipTask, ITcUnknown * ipCaller, ULONG_PTR context)

///<AutoGeneratedContent id="ImplementationOf_ITcCyclic">
HRESULT CSigGen::CycleUpdate(ITcTask* ipTask, ITcUnknown* ipCaller, ULONG_PTR context)
{
    HRESULT hr = S_OK;

    m_counter+=m_Inputs.Value;
    m_Outputs.Value=m_counter;

    if (!m_bRun)
        return hr;

    excitation_update(&m_excitationData);
    excitation_output(&m_excitationData, &m_Outputs.SignalOutputReal);
    m_Outputs.SignalOutputUInt = (UINT32)m_Outputs.SignalOutputReal;

    switch (m_State)
    {
    case 1:
        m_counter++;
        break;
    case 2:
        m_counter--;
        break;
    }
    return hr;
}
```

Figure 1: MS Visual Studio solution explorer and C++ editor.

TC3 has a modular architecture. All the real-time components are encapsulated in modules which are managed by the run-time system. TC3 uses the concepts from "Component Object Model (COM)" to define the characteristics and behavior of the modules. TwinCAT COM (TcCOM) is the adaptation of COM to the automation technology that allows modules implemented in different languages to interact seamlessly in the real-time context, see Figure 2.

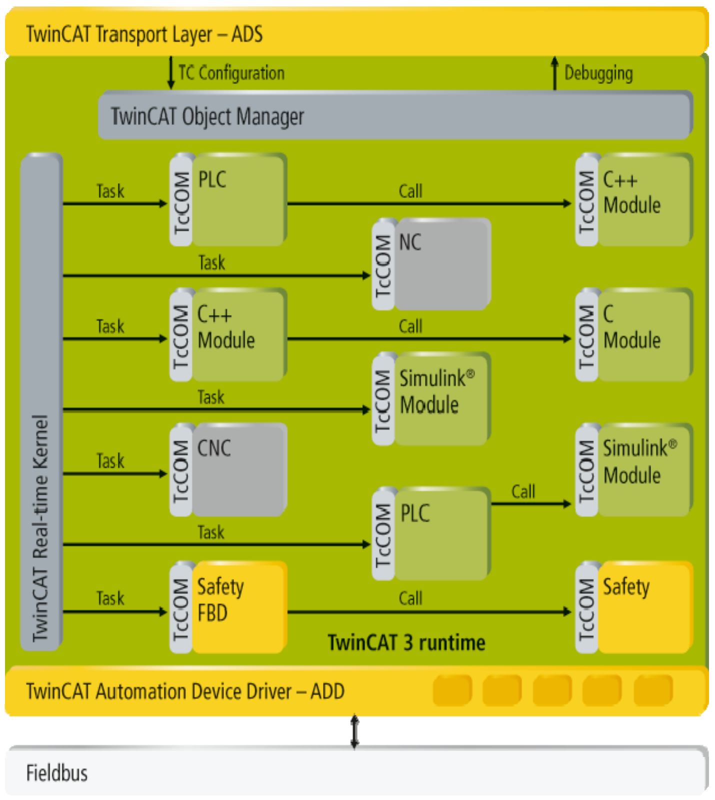


Figure 2: Modular TC3 run-time (*source: BECKHOFF website*)

Each TC3 module has a set of mandatory and optional attributes. The mandatory attributes of each TC3 module are: description, state machine and a generic interface (*ITComObject*). The *ITComObject* interface is used to access basic information and status of the module like name, object ID, parameters and state[3].

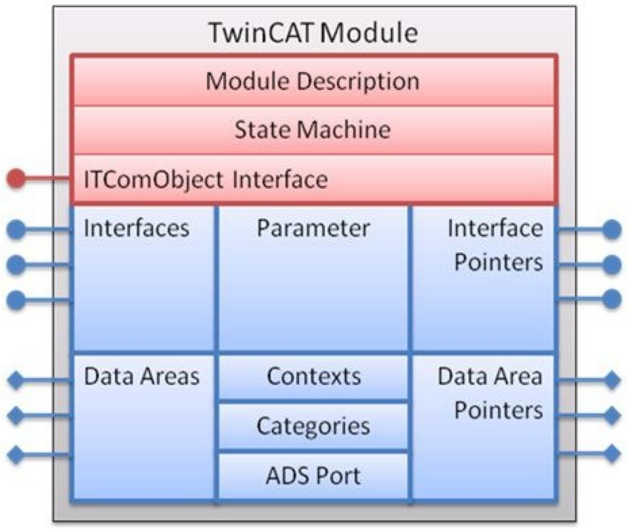


Figure 3: Internal structure of a TC3 module (*source: BECKHOFF website*)

The module state machine, as showed in Figure 4, describes the general state of the module. It controls the initialization, parameterization and the creation of the connection to the other modules[3].

The state and the transition of the C++ modules are consistent with the ones of EtherCAT. The only state taking place in the real-time context is the operational state (OP).

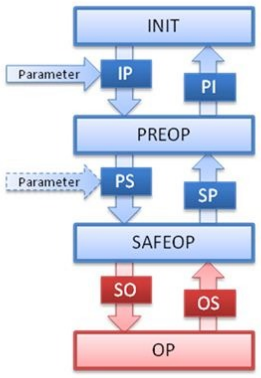


Figure 4: TC3 module state machine (*source: BECKHOFF website*)

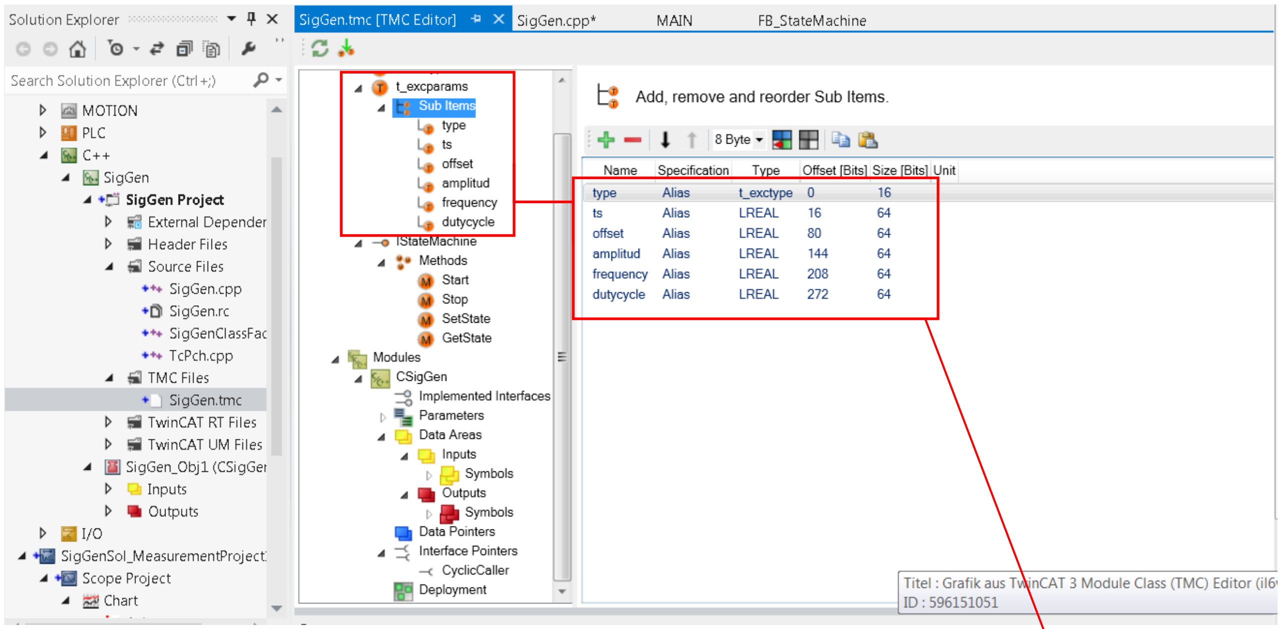
There are two types of files to describe the modules:

- Class description files, *.tmc
- Instance description files, *.tmi

The class description contains the description of the module and its interface together with the general information of the module: vendor information, class ID, etc. The instance description file contains the concrete settings of the module like parameters, interface pointers, etc.

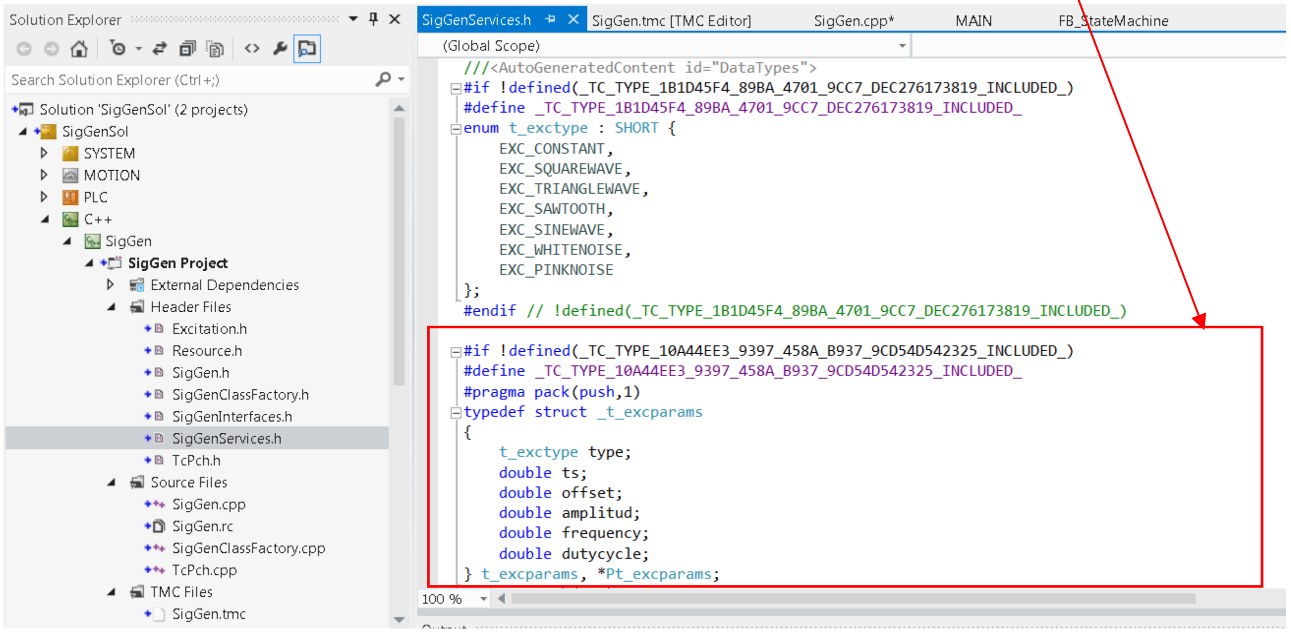
C++ Code Generation

TC3 provides a code generator for implementing the interfaces of module classes such as type definitions, parameters, data areas, etc. The generator is based on the TMC file which is produced by the TMC editor, a graphical application embedded in TC3. Once data types, parameters and inputs and outputs of a module are defined through the graphical editor, the code generation transforms them from TMC file to the C++ code, see Figure 5 and Figure 6.



Name	Specification	Type	Offset [Bits]	Size [Bits]	Unit
type	Alias	t_exctype	0	16	
ts	Alias	LREAL	16	64	
offset	Alias	LREAL	80	64	
amplitud	Alias	LREAL	144	64	
frequency	Alias	LREAL	208	64	
dutycycle	Alias	LREAL	272	64	

Figure 5: TMC editor



```
///<AutoGeneratedContent id="DataTypes">
#if !defined(_TC_TYPE_1B1D45F4_89BA_4701_9CC7_DEC276173819_INCLUDED_)
#define _TC_TYPE_1B1D45F4_89BA_4701_9CC7_DEC276173819_INCLUDED_
enum t_exctype : SHORT {
    EXC_CONSTANT,
    EXC_SQUAREWAVE,
    EXC_TRIANGLEWAVE,
    EXC_SAWTOOTH,
    EXC_SINEWAVE,
    EXC_WHITENOISE,
    EXC_PINKNOISE
};
#endif // !defined(_TC_TYPE_1B1D45F4_89BA_4701_9CC7_DEC276173819_INCLUDED_)

#if !defined(_TC_TYPE_10A44EE3_9397_458A_B937_9CD54D542325_INCLUDED_)
#define _TC_TYPE_10A44EE3_9397_458A_B937_9CD54D542325_INCLUDED_
#pragma pack(push,1)
typedef struct _t_excparams
{
    t_exctype type;
    double ts;
    double offset;
    double amplitud;
    double frequency;
    double dutycycle;
} t_excparams, *Pt_excparams;
```

Figure 6: Extract of a code generation.

C++ Debugging

Traditional debugging capabilities for C++ are supported in TC3. Additionally, TC3 provides a mechanism for monitoring C++ variables at run-time and without stopping the normal execution of the modules. This facility is called

LiveWatch and brings a new way of debugging and troubleshooting C++ code more similar to the tools provided within the IEC61131-3 editor.

SVN Integration

VS can be extended through plug-ins. We have been using the *AnkhSVN* plug-in which is an open source SVN client that is supported and well integrated into VS.

SIMULINK Integration

MathWorks enables the code generation from Simulink models to various targets through the Embedded Coder [5]. With the Embedded Coder plus the TC3 Target for *MatLab/Simulink (TE1400)*, supplementary software from BECKHOFF, it is possible to generate C++ code encapsulated in a standard TC3 module format that can be instantiated or loaded into the TC3 development environment.

In order to perform the code generation it is necessary to customize the coder settings before triggering the building process within Simulink. The generation process delivers two outputs:

- C++ code generated by the MathWorks Embedded Coder/TE1400 in the form of a VS C++ project. The generated project contains all source code files, compiler and linker settings that are necessary to successfully compile the module within TC3 environment.
- Binary (object file) produced by the MS VS C++ compiler.

Binaries can be added as TcCOM objects to the VS solution under the TC3 System. The inputs and outputs of the TC3 module are matching the ones in the Simulink model from where it was generated.

One of the most interesting capabilities of TC3 is the possibility to display and navigate through the Simulink block diagram including the display of parameters and signals values that can be monitored and/or modified at run-time. The user can adjust these parameters within the TC3 environment without to change the original model.

The TC3 block diagram keeps the same layout and model element names, see Figure 7 and Figure 8.

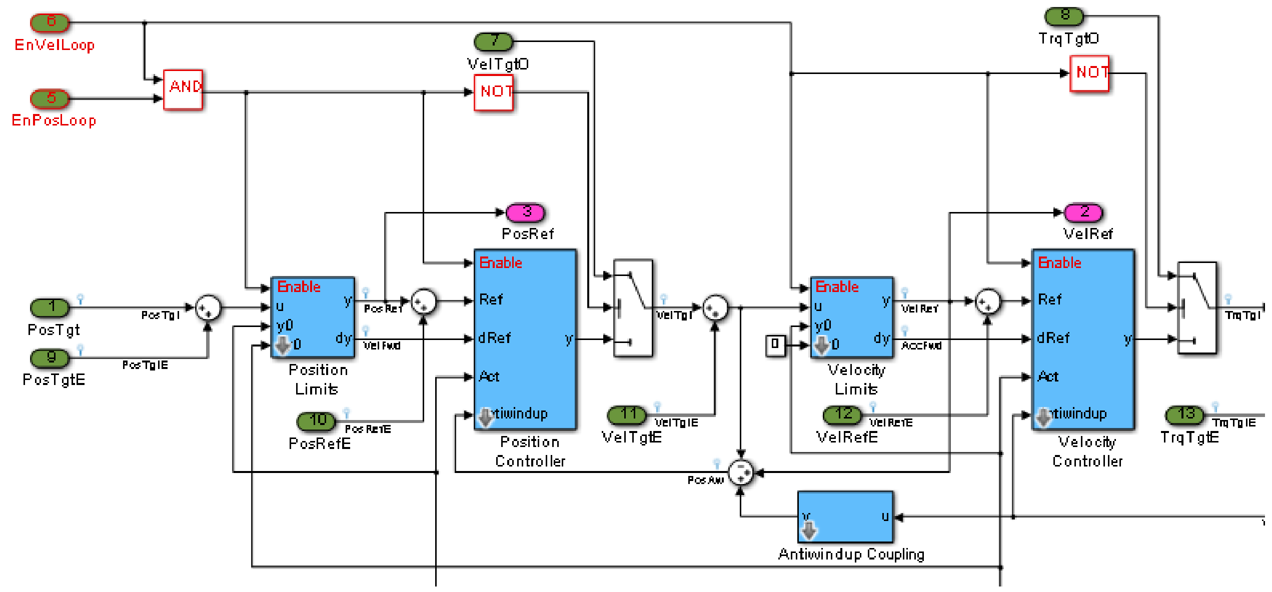


Figure 7: Extract of a simplified version of the ESTAC Simulink model.

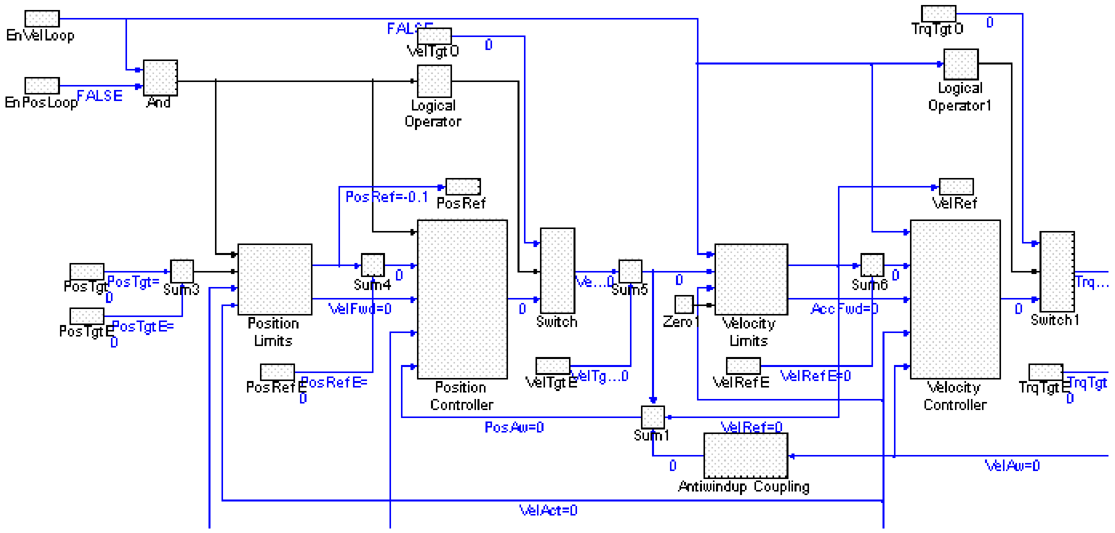


Figure 8: TC3 version of the original Simulink model.

2. TRACKING DEVICES

We have implemented a simple prototype of an instrument tracking device (derotator) with the aim of learning and validating the C++ support within TC3. While doing this, we have reused the computation of the field rotation compensation from the existing LCU code which has been used by several VLT instruments. The adaptation of the C code was a fairly simple and straightforward process. The changes to be done were minor, mainly to convert the base time between the two systems and the name of the trigonometric functions which have a different name under the TC3 environment.

As outlined in Figure 9, the tracking device prototype consists of two TC3 modules. One TC3 module doing the field rotation computation (C++), and the other one doing the interaction with the motion controller through the Beckhoff NC task (ST).

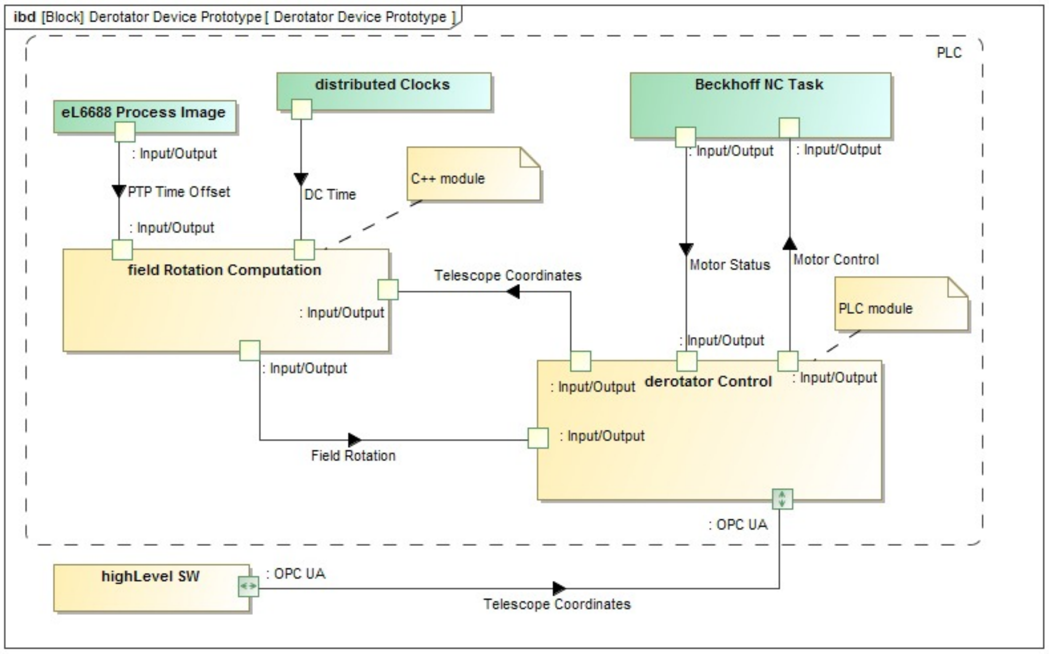


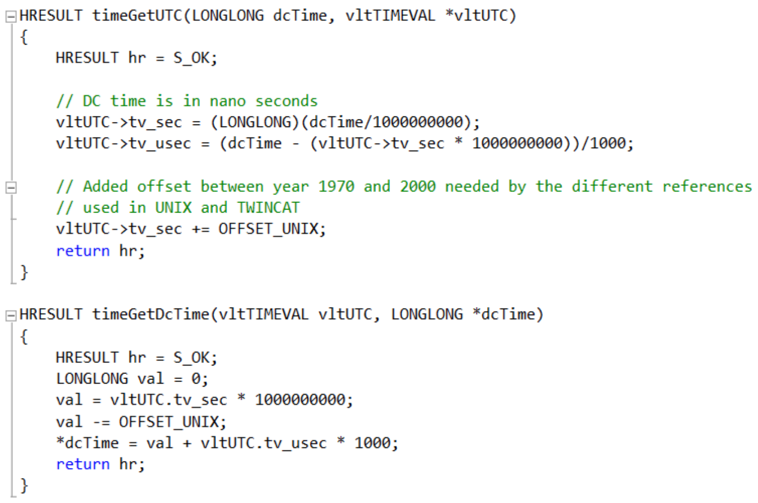
Figure 9: Derotator device architecture (prototype).

2.1 C++ Module

The C++ module is executed cyclically in the context of an independent task. The PLC code gets the information from the C++ module via the mapping of input/output variables. The C++ module obtains the UTC time required to compute the field rotation based on the internal distributed clocks time and the external reference time (IEEE 1588). We have used the interface of the system real-time task (ITcRTime) to inquire the distributed clocks time inside the C++ module:

```
m_spRTime->GetCurDcTime(GETCURDCTIME_ACTUAL, &m_Outputs.TaskDcTime );
```

The distributed clocks time is expressed in nanoseconds since January 1st , 2000. The maximum resolution is 100 ns. We have implemented some C++ functions to do the conversion between UTC and distributed clock time, see Figure 10.



```
HRESULT timeGetUTC(LONGLONG dcTime, vltTIMEVAL *vltUTC)
{
    HRESULT hr = S_OK;

    // DC time is in nano seconds
    vltUTC->tv_sec = (LONGLONG)(dcTime/1000000000);
    vltUTC->tv_usec = (dcTime - (vltUTC->tv_sec * 1000000000))/1000;

    // Added offset between year 1970 and 2000 needed by the different references
    // used in UNIX and TWINCAT
    vltUTC->tv_sec += OFFSET_UNIX;
    return hr;
}

HRESULT timeGetDcTime(vltTIMEVAL vltUTC, LONGLONG *dcTime)
{
    HRESULT hr = S_OK;
    LONGLONG val = 0;
    val = vltUTC.tv_sec * 1000000000;
    val -= OFFSET_UNIX;
    *dcTime = val + vltUTC.tv_usec * 1000;
    return hr;
}
```

Figure 10: Time conversion functions using C++ in TC3 environment.

The input/output variables of the C++ module are described here:

Input:

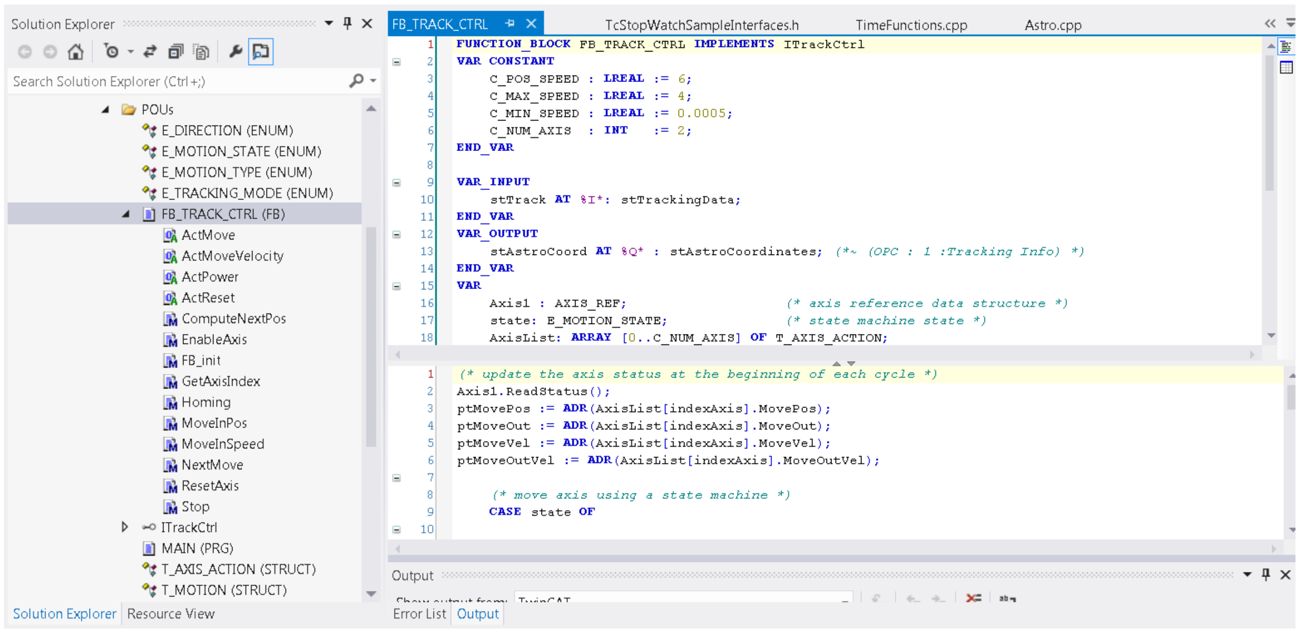
- Structure containing the telescope pointing coordinates (ra, dec, posang, equinox).
- Structure containing the time reference coming from terminal IEEE1588.
- Distributed Clocks Time.

Output:

- Structure containingµ the computation of the field rotation (altitude, azimuth, hour angle, parallactic angle, pupil rotation, field rotation, etc.)

2.2 PLC Module

The PLC module consists of two Program Organization Units (POUs), the main program and the function block FB_TRACK_CTRL. The FB_TRACK_CTRL implements a set of methods encapsulating the motion control functionalities and the state machine handling (see Figure 11). The main program is just instantiating the FB_TRACK_CTRL.



```
FUNCTION_BLOCK FB_TRACK_CTRL IMPLEMENTS ITrackCtrl
VAR CONSTANT
    C_POS_SPEED : LREAL := 6;
    C_MAX_SPEED : LREAL := 4;
    C_MIN_SPEED : LREAL := 0.0005;
    C_NUM_AXIS  : INT   := 2;
END_VAR

VAR_INPUT
    stTrack AT %I*: stTrackingData;
END_VAR
VAR_OUTPUT
    stAstroCoord AT %Q* : stAstroCoordinates; (*~ (OPC : 1 :Tracking Info) *)
END_VAR
VAR
    Axis1 : AXIS_REF;                     (* axis reference data structure *)
    state: E_MOTION_STATE;                (* state machine state *)
    AxisList: ARRAY [0..C_NUM_AXIS] OF T_AXIS_ACTION;
```

```
(* update the axis status at the beginning of each cycle *)
Axis1.ReadStatus();
ptMovePos := ADR(AxisList[indexAxis].MovePos);
ptMoveOut := ADR(AxisList[indexAxis].MoveOut);
ptMoveVel := ADR(AxisList[indexAxis].MoveVel);
ptMoveOutVel := ADR(AxisList[indexAxis].MoveOutVel);

    (* move axis using a state machine *)
    CASE state OF

```

Figure 11: PLC module prototype implementation.

Input:

- Telescope pointing coordinates (ra, dec, posang, equinox) received from the high-level software via OPC UA[6].
- NC Structure with the motor status.
- Field rotation information received from the C++ module via the input and output variables.

Output: µ

- Updated motor position/velocity.
- Telescope coordinates. This is required since C++ modules do not support direct interface to OPC UA.

2.3 Evaluation

The implementation of the prototype of the tracking device has been the first attempt to use the C++ language to implement software artifacts under the PLC platform at ESO. After the familiarization with the technology, it was rather simple to implement and adapt existing pieces of the VLT code to the new environment.

The required correction frequency of the existing derotator devices running on the LCU platform is at maximum 10 Hz. The limiting factor is just the requirement and not the capabilities of the LCU. The achieved correction frequency of prototype running on the PLC was 1 kHz (on a CX2030) given by the PLC cycle time. At each cycle, the PLC part of the prototype is computing a new set point using the information obtained from the C++ part. The computation (in C++) to fill up the field rotation structure took less than 50 microseconds. The overall CPU load of the PLC running the prototype of the tracking device was very low (2-3%).

3. ESO STANDARD AXIS CONTROLLER (ESTAC)

ESTAC is the new standard controller for the control of the main axes of the auxiliary and unit telescopes in Paranal[2]. It has a model based design implemented using MathWorks Simulink. Our goal was to validate the feasibility of integrating this ESO product on the new PLC based development environment and use it to control the telescope simulator and some DC motors. In order to achieve our goal, we have built a prototype application combining three TC3 modules:

- ESTAC (Simulink module): a simplified version of the ESTAC model which has been reduced to include only the controller part, see Figure 7.

- Signal Generator (C++ module): a set of C++ routines reused from the ESTAC development which produce test signals that can be injected in the controller in order to carry out performance measurements[4].
- PLC module: a component doing the gluing of all TC3 modules and the interaction with the hardware.

3.1 Evaluation

We have successfully deployed the ESTAC application on a BECKHOFF Embedded PC. We have tested it by controlling the telescope simulator, an existing mockup based on NI hardware that is used to test ESTAC for telescope main axes in the VLT control model. Additionally, we have used ESTAC to control, in position and velocity, two different DC motors using the DC controller EL7341 and the encoder interface EL5101-0010. In both cases the CPU load remained below 10% running at cycle time of 1 ms.

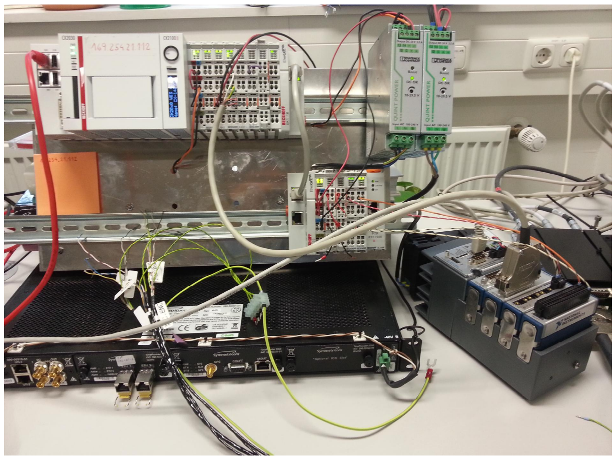
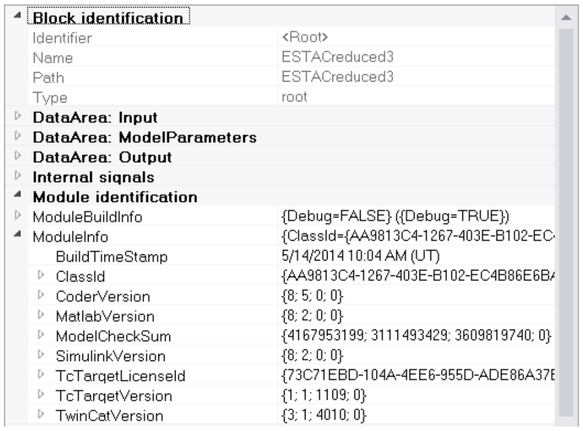


Figure 12: Laboratory setup for the integration of ESTAC on a BECKHOFF PLC.

There is a tight integration between TC3 and Simulink, the TC3 target for the MathWorks Embedded Coder includes makefiles and build scripts that together with VS compiler/linker produce a TC3 object file directly from the Simulink block diagram, without manual intervention.

In TC3 is possible to navigate through the different levels of the Simulink model which are presented hierarchically in a tree view together with the block diagram (see Figure 14). The values of the model parameters and the internal signals can be accessed and modified at run-time.

The TC3 module general information can be also accessed from the list of properties of the model.



Block identification	
Identifier	<Root>
Name	ESTACreduced3
Path	ESTACreduced3
Type	root
DataArea: Input	
DataArea: ModelParameters	
DataArea: Output	
Internal signals	
Module identification	
ModuleBuildInfo	{Debug=FALSE} ({Debug=TRUE})
ModuleInfo	{ClassId={AA9813C4-1267-403E-B102-EC
BuildTimeStamp	5/14/2014 10:04 AM (UT)
ClassId	{AA9813C4-1267-403E-B102-EC4B86E6B
CoderVersion	{8; 5; 0; 0}
MatlabVersion	{8; 2; 0; 0}
ModelCheckSum	{4167953199; 3111493429; 3609819740; 0}
SimulinkVersion	{8; 2; 0; 0}
TcTargetLicenseId	{73C71EBD-104A-4EE6-955D-ADE86A37
TcTargetVersion	{1; 1; 1109; 0}
TwinCatVersion	{3; 1; 4010; 0}

Figure 13: TC3 Simulink module properties.

During the integration we faced some difficulties bringing the TC3 into run mode. The system simply remained in config mode and no error was reported. Later on we discovered that the problem was due to a wrong configuration of the parameters of the task calling the TC3 Simulink module. These parameters (task priority, step size, etc.) should match the ones defined in the MathWorks coder Tc advanced configuration.

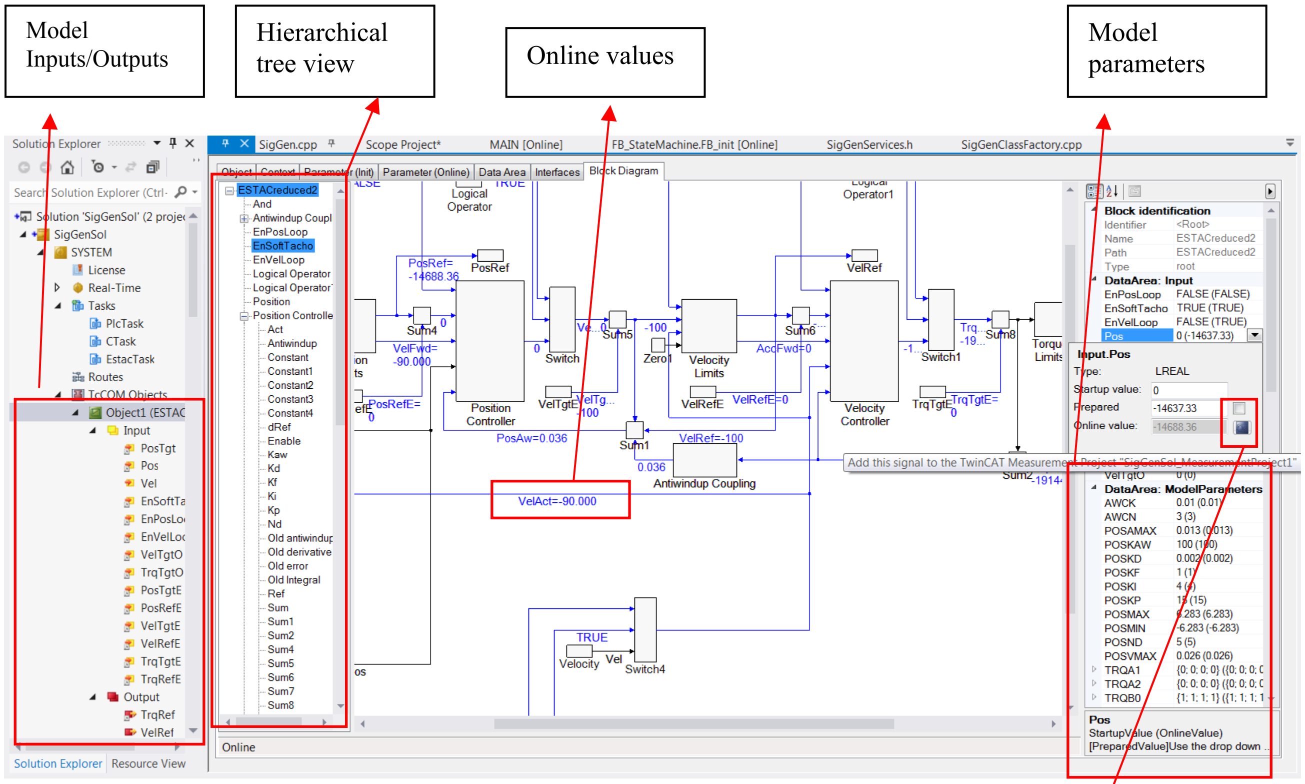


Figure 14: ESTAC Simulink model running in TC3 real-time environment.

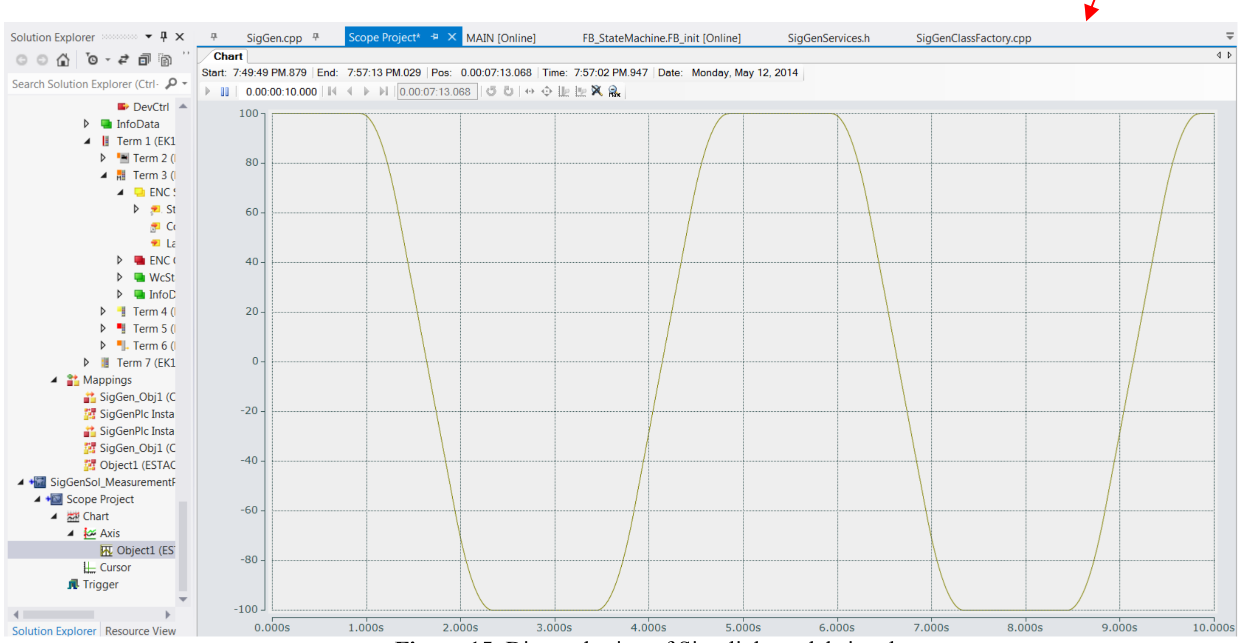


Figure 15: Direct plotting of Simulink model signals.

These tests demonstrated the versatility and the potential of integrating C++ and Simulink application into the TC3 environment. The direct benefits are:

- Re-use of existing and previously tested software components. Some minor modifications are required to adapt the code to the new platform though.
- Simplification of the development by using a well known programming language in our environment. Most of our software engineers are experienced C/C++ developers.
- Implementation of more advanced applications requiring higher order of performance.

The suggested architecture for applications combining PLC and C++ code is to encapsulate the interaction between the hardware and the high-level software within the PLC part, and the computation of intensive tasks or complex algorithms within the C++ part. Nevertheless, in the scope of instrument control and in order to keep it simple and to avoid additional licenses, we intend to use C++ and Simulink only for those cases where this is justified, e.g. for the computation of field rotation for tracking devices.

4. CONCLUSIONS

With the latest version of TwinCAT it is possible to develop PLC code not only with the traditional standard languages specified in IEC61131-3, but also in C/C++ and Matlab/Simulik. This is a major advantage that opens the door to implementing more advanced applications with a higher level of computational complexity, beyond what is covered by traditional PLCs. Our laboratory tests of the new TC3 functionalities demonstrated the feasibility of reusing existing VLT software code implemented in C++. This will certainly ease the implementation of instrument control software for this new development platform.

The feasibility of integrating Simulink models running in the PLC and without any modification was also verified confirming the expansion of the possibilities for this development platform not only for using specialized controllers like ESTAC but also for implementing simulation capabilities.

REFERENCES

[1] Kiekebusch, M., Lucuix, C., Erm, T., Chiozzi, G., Zamparelli, M., Kern, L., Brast, R., Pirani, W., Reiss, R., Popovic, D., Knudstrup, J., Duchateau, M., Sandrock, S., Di Lieto, N., “PC based PLCs and Ethernet based fieldbus: the new standard platform for future VLT instrument control”, Proc. SPIE 9152 (2014)
[2] Sandrock, S., Di Lieto, N.,Pettazzi, L., Erm, T., “Design and implementation of a general main axis controller for the ESO telescopes”, Proc. SPIE 8451 (2012).
[3] http://www.beckhoff.de
[4] Di Lieto, N., "Standard Telescope Axes Controller, Control Algorithm Specification", ESO internal document, (2011)
[5] http://www.mathworks.de/
[6] http://www.opcfoundation.org