Beckhoff and TwinCAT 3 System Development Guide

December 2015

Steven Blair
University of Strathclyde

Contents

Abbreviations

CPU	Central Processing Unit
IO	Input or Output
PC	Personal Computer
PMU	Phasor Measurement Unit
TMC	TwinCAT Module Class
WDK	Windows Driver Kit

1 Introduction

This document is a guide for setting up a Beckhoff hardware controller and a development PC. It is assumed that the development PC runs Windows 7 or Window 8/8.1, although other versions of Windows (including 32-bit and 64-bit) should also work. In particular, guidance is given on using C++ modules and integrating MATLAB Simulink models within TwinCAT 3.

This guide supplements existing Beckhoff documentation:

- Webinar on TwinCAT 3 C++ integration: http://multimedia.beckhoff.com/webinar/en/Webinar_TwinCAT_3_C_plusplus_Integration/default.htm
- C++ Quick Start Guide: https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_c/72057594148625035.html&id=
- The comprehensive manual is available here: ftp://ftp.beckhoff.com/Document/automation/twincat3/TC1300_C_EN.pdf
- The Beckhoff Information System (available at: http://infosys.beckhoff.com/index_en.htm) contains general documentation and some code examples; however, many examples are for TwinCAT 2 and cannot be used in TwinCAT 3.

2 TwinCAT 3 Installation on Development PC

1. Install Visual Studio. There are several versions which are compatible with TwinCAT 3, and provide full functionality:
    a. Free download: Visual Studio Community 2013 (available here: https://www.visualstudio.com/en-us/news/vs2013-community-vs.aspx). Note that TwinCAT 3 does not yet support the 2015 version.
    b. Commercial: Visual Studio 2010, 2012, or 2013 (Professional, Premium, or Ultimate).
    c. Within Strathclyde, the 2013 Professional edition can obtained from the Microsoft DreamSpark agreement, for teaching and non-commercial research purposes.

   Note that if you install TwinCAT 3 without Visual Studio already installed, it will install "Visual Studio 2010 Shell". This version is not suitable for C++ development or debugging.
2. Install Microsoft Windows Driver Kit (WDK) 7 and set up a Windows environment variable, as described by the following instructions: http://infosys.beckhoff.com/english.php?content=../content/1033/tc3_c/54043195639122187.html&id=
3. Download TwinCAT 3 from http://www.beckhoff.co.uk/english.asp?download/tc3-downloads.htm?id=1905053019883865
    a. You need to create an account on the Beckhoff website to access downloads.
    b. The specific download is found in the "Software" > "TE1xxx | Engineering" section, and is called "TwinCAT 3.1 – eXtended Automation Engineering (XAE)".
4. When using TwinCAT 3, you might be asked to create or renew licenses. This is normal. Licenses last for one week, but can be simply renewed, perpetually, when needed. Licenses only need to be purchased for commercial work.
5. For 64-bit targets, two additional steps must be done:
    a. On the development PC, set up digital driver signing. The process is given here: http://infosys.beckhoff.com/english.php?content=../content/1033/tc3_c/63050394893879947.html&id=. Note that step 3 is only required for executing the system on the local development machine.
    b. On the hardware target, execute the "bcdedit /set testsigning yes" command (step 3 from the link above) and reboot the device.
6. Visual Studio with TwinCAT 3 can be opened form the Windows system tray icon, and a new TwinCAT 3 project can be created as described here: http://infosys.beckhoff.com/english.php?content=../content/1033/tc3_c/54043195639143947.html&id=17135.

3 Connecting to a Hardware Controller

A TwinCAT “target” is the hardware device which will ultimately execute a control system. Typically, the target will be one of the Beckhoff hardware controllers (such as the CX1020, etc.), but the local development PC can also be used as the target for testing purposes.

To use a hardware controller:

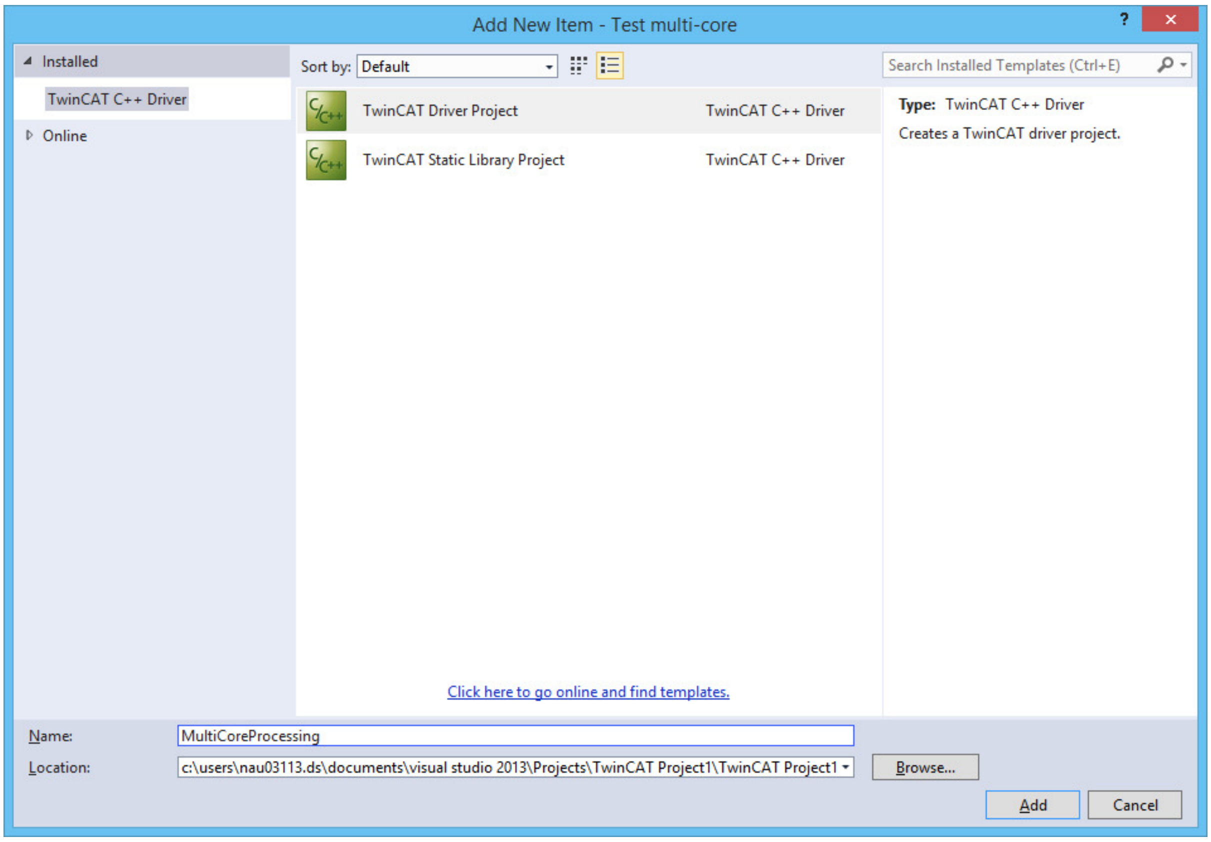
1. Connect one of the Ethernet interfaces on the Beckhoff controller to the development PC. This can be a direct connection, or via a switch. Set the PC's network interface IP address to be on the same subnet as the control. For example, if the controller’s IP address is 192.168.2.10 with a subnet mask of 255.255.255.0, then the PC’s IP address could be set to 192.168.2.20. Note that Beckhoff controllers must not be added to a corporate Windows domain, because this conflicts with the functionality.
2. Create a new TwinCAT 3 project (or use an existing project) in Visual Studio.
3. In the Solution Explorer panel, double-click on the “SYSTEM” for your project and select “Choose target...”. Select “Search (Ethernet)”. Press "Broadcast Search" and select the appropriate network interface (if there is more than one). Select the appropriate controller device in the list. You may need to select the "IP address" radio box before adding the route. Enter the device username and password (by default: "Administrator" and "1"). This process is described in further detail here: http://infosys.beckhoff.com/english.php?content=../content/1033/tc3_system/html/tcsysmgr_addroutedialog.htm&id=.

4 TwinCAT 3 – Typical C++ Project Configuration

This section describes adding a C++ module to a TwinCAT 3 project.

4.1 C++ Module

In the Solution Explorer panel, right click on “C++” and “Add New Item...”. Select “TwinCAT Driver Project”, choose a name, and click on “Add”.


Add New Item - Test multi-core
Installed
TwinCAT C++ Driver
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
Click here to go online and find templates.
Browse...
Add
Cancel

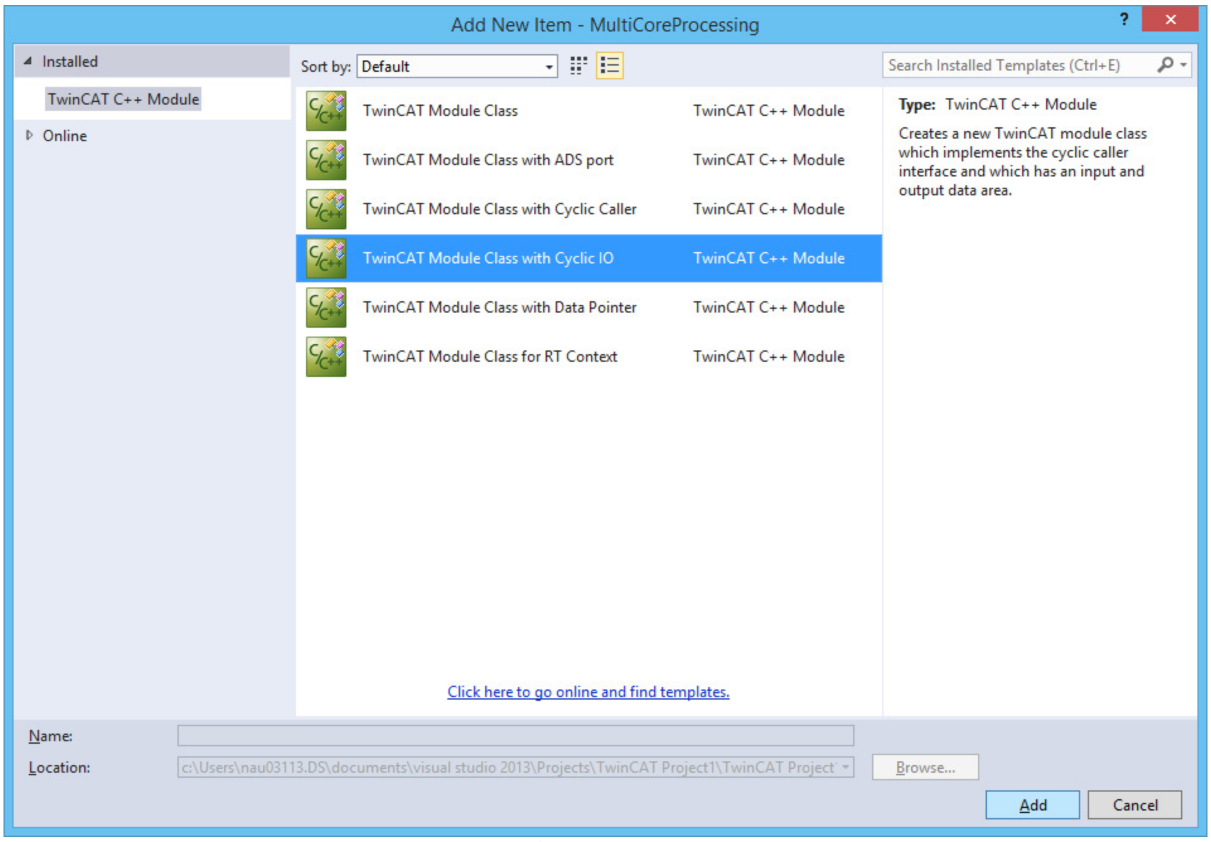
TwinCAT Driver Project	TwinCAT C++ Driver
TwinCAT Static Library Project	TwinCAT C++ Driver

Type: TwinCAT C++ Driver
Creates a TwinCAT driver project.

Name: MultiCoreProcessing
Location: c:\users\nau03113.ds\documents\visual studio 2013\Projects\TwinCAT Project1\TwinCAT Project1

Figure 1: Initial C++ module creation

There are several C++ module templates to choose from:



Add New Item - MultiCoreProcessing

Installed
TwinCAT C++ Module
Online

Sort by: Default

Search Installed Templates (Ctrl+E)

TwinCAT Module Class	TwinCAT C++ Module
TwinCAT Module Class with ADS port	TwinCAT C++ Module
TwinCAT Module Class with Cyclic Caller	TwinCAT C++ Module
TwinCAT Module Class with Cyclic IO	TwinCAT C++ Module
TwinCAT Module Class with Data Pointer	TwinCAT C++ Module
TwinCAT Module Class for RT Context	TwinCAT C++ Module

Type: TwinCAT C++ Module

Creates a new TwinCAT module class which implements the cyclic caller interface and which has an input and output data area.

Click here to go online and find templates.

Name:

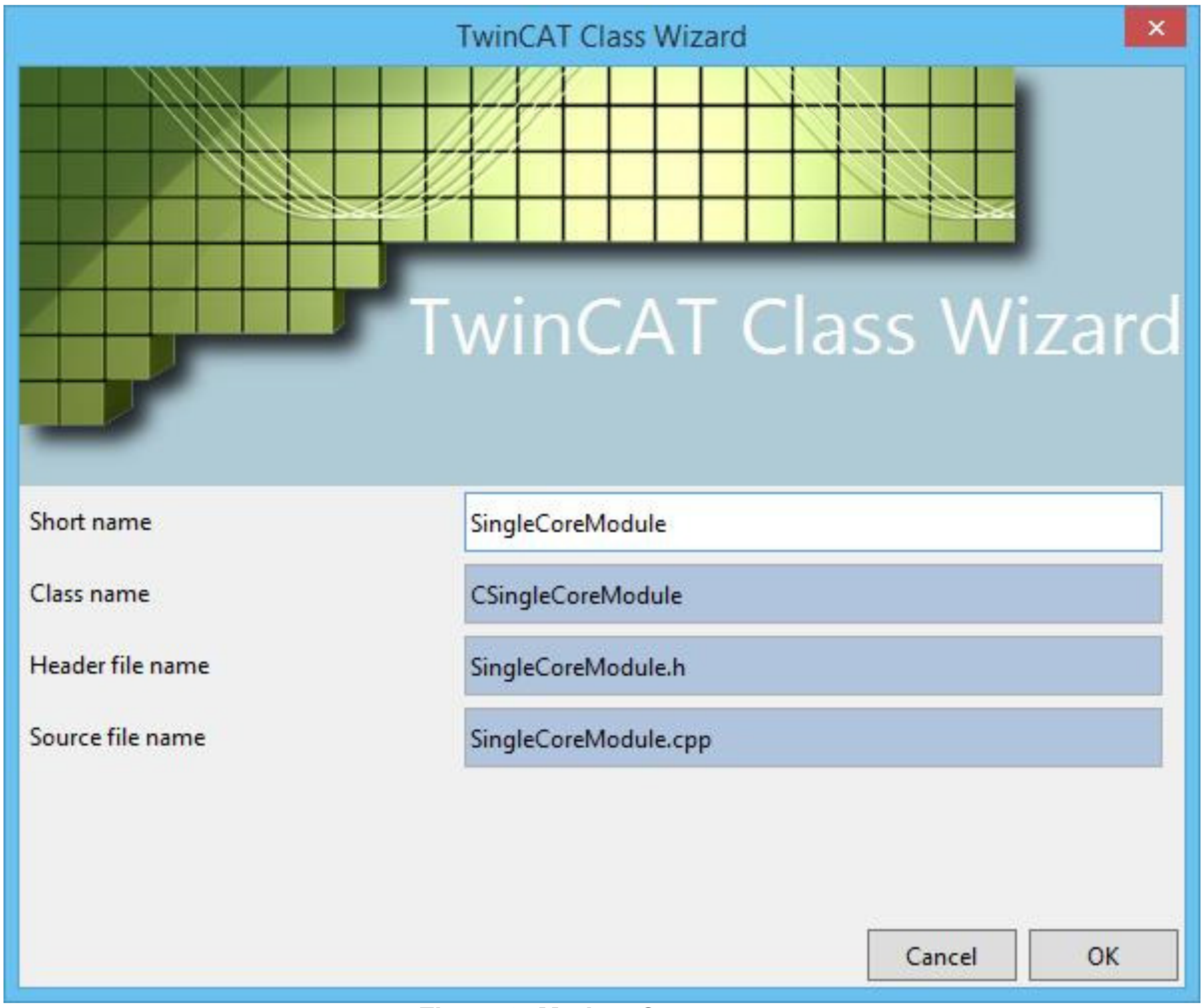
Location: c:\Users\nau03113.DS\documents\visual studio 2013\Projects\TwinCAT Project1\TwinCAT Project

Browse...

Add Cancel

Figure 2: C++ module template

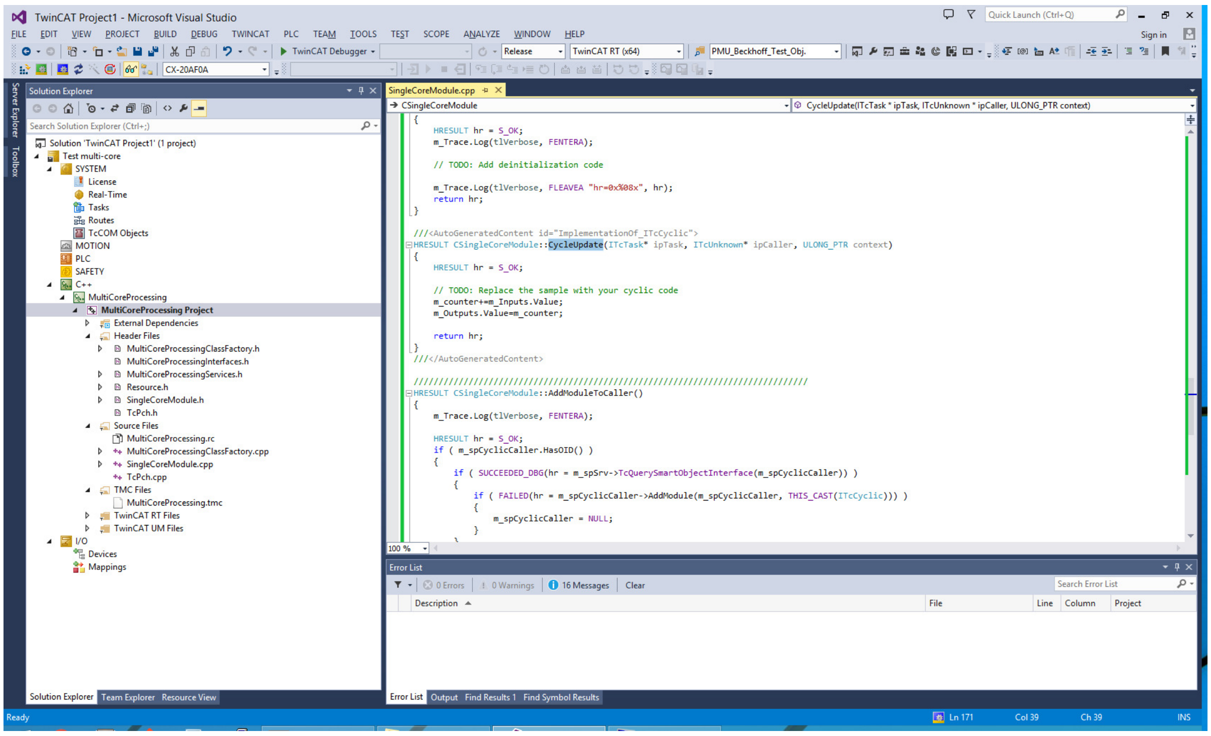
Select the option with “Cyclic IO” and click on “Add”. Select a name for the Module Class:



Short name: SingleCoreModule
Class name: CSingleCoreModule
Header file name: SingleCoreModule.h
Source file name: SingleCoreModule.cpp

Figure 3: Module Class name

This results in a “Module Project” with a TwinCAT Module Class (TMC) file and some pre-generated C++ code. For example a template CycleUpdate() function has been created within the main module C++ file (SingleCoreModule.cpp in this example):

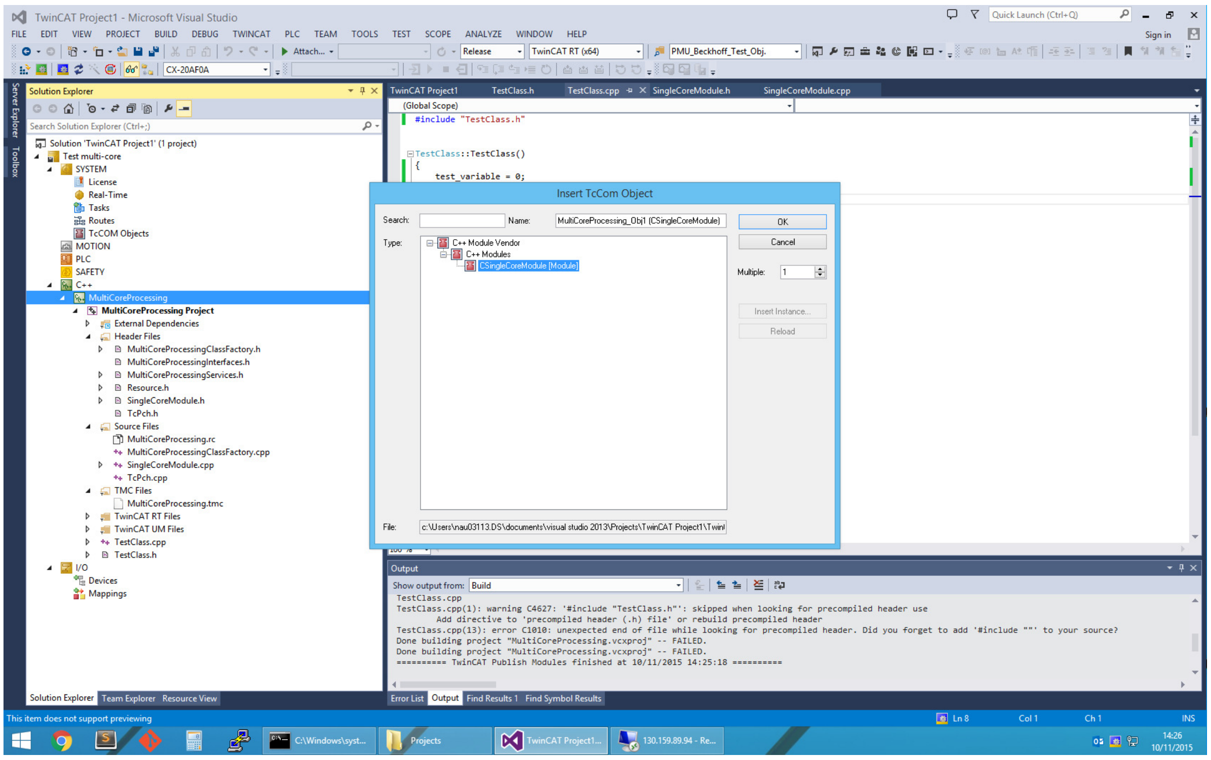


```
{
    HRESULT hr = S_OK;
    m_Trace.Log(tlVerbose, FENTERA);

    // TODO: Add deinitialization code

    m_Trace.Log(tlVerbose, FLEAVEA "hr=0x%08x", hr);
    return hr;
}

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

    // TODO: Replace the sample with your cyclic code
    m_counter+=m_Inputs.Value;
    m_Outputs.Value=m_counter;

    return hr;
}
///</AutoGeneratedContent>

///////////////////////////////////////////////////////////////////////////////
HRESULT CSingleCoreModule::AddModuleToCaller()
{
    m_Trace.Log(tlVerbose, FENTERA);

    HRESULT hr = S_OK;
    if ( m_spCyclicCaller.HasOID() )
    {
        if ( SUCCEEDED_DBG(hr = m_spSrv->TcQuerySmartObjectInterface(m_spCyclicCaller)) )
        {
            if ( FAILED(hr = m_spCyclicCaller->AddModule(m_spCyclicCaller, THIS_CAST(ITcCyclic))) )
            {
                m_spCyclicCaller = NULL;
            }
```

Figure 4: Example of initial generated code

However, this Module Class is merely "template", and you must still create an instance (or "object") of the Module Class to assign to a processor core on a target controller.

Right click on the C++ project, select "Add New Item", and find your "published" module, as shown below:



```
#include "TestClass.h"


TestClass::TestClass()
{
    test_variable = 0;
```

Insert TcCom Object

Name: MultiCoreProcessing_Obj1 (CSingleCoreModule)

Type: C++ Module Vendor > C++ Modules > CSingleCoreModule [Module]

Multiple: 1

File: c:\Users\nau03113.DS\documents\visual studio 2013\Projects\TwinCAT Project1\Twin

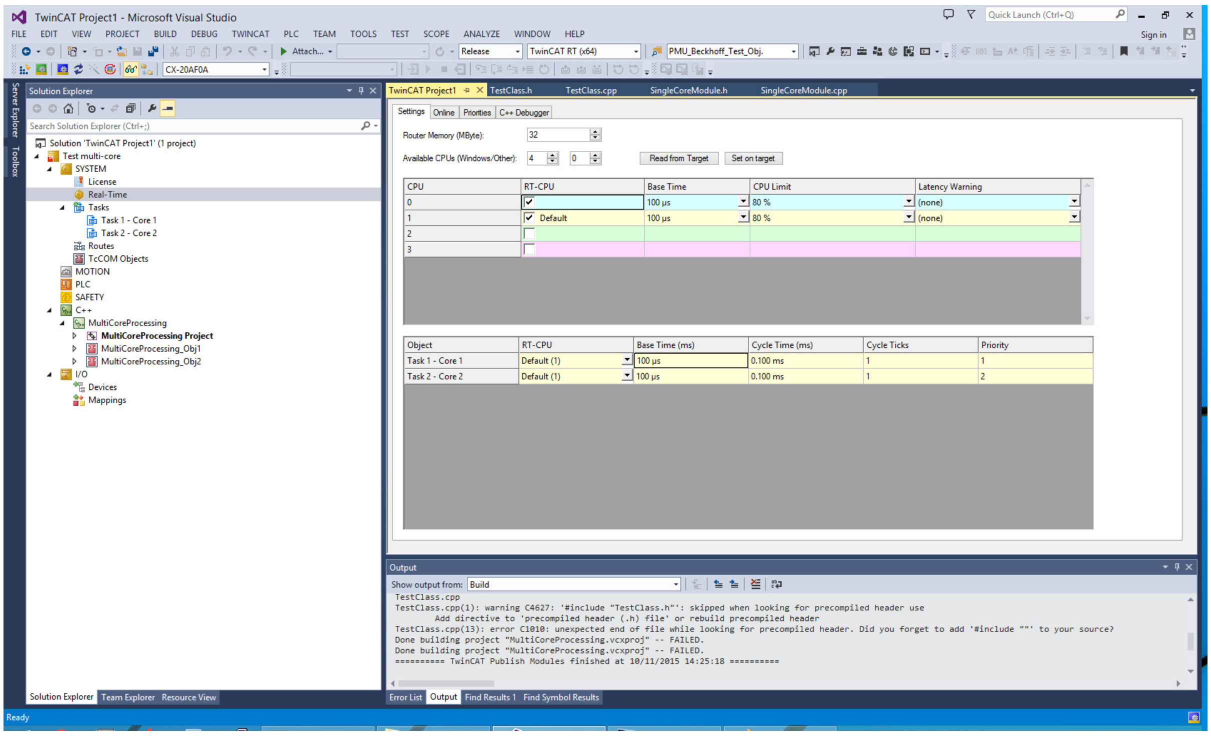
```
TestClass.cpp
TestClass.cpp(1): warning C4627: '#include "TestClass.h"': skipped when looking for precompiled header use
        Add directive to 'precompiled header (.h) file' or rebuild precompiled header
TestClass.cpp(13): error C1010: unexpected end of file while looking for precompiled header. Did you forget to add '#include ""' to your source?
Done building project "MultiCoreProcessing.vcxproj" -- FAILED.
Done building project "MultiCoreProcessing.vcxproj" -- FAILED.
========== TwinCAT Publish Modules finished at 10/11/2015 14:25:18 ==========
```

Figure 5: Creating an instance of a Module Class

This means that you can create multiple object instances from the same Module Class template.

4.2 Assigning Module Instances to Processor Cores

In “SYSTEM” > “Real-Time”, enable the required CPU cores (you may first need to click “Read from Target”) and set the required Base Time:



Solution 'TwinCAT Project1' (1 project)
- Test multi-core
  - SYSTEM
    - License
    - Real-Time
    - Tasks
      - Task 1 - Core 1
      - Task 2 - Core 2
    - Routes
    - TcCOM Objects
  - MOTION
  - PLC
  - SAFETY
  - C++
    - MultiCoreProcessing
      - **MultiCoreProcessing Project**
      - MultiCoreProcessing_Obj1
      - MultiCoreProcessing_Obj2
  - I/O
    - Devices
    - Mappings

Router Memory (MByte): 32

Available CPUs (Windows/Other): 4 0

CPU	RT-CPU	Base Time	CPU Limit	Latency Warning
0	✓	100 µs	80 %	(none)
1	✓ Default	100 µs	80 %	(none)
2	☐			
3	☐			

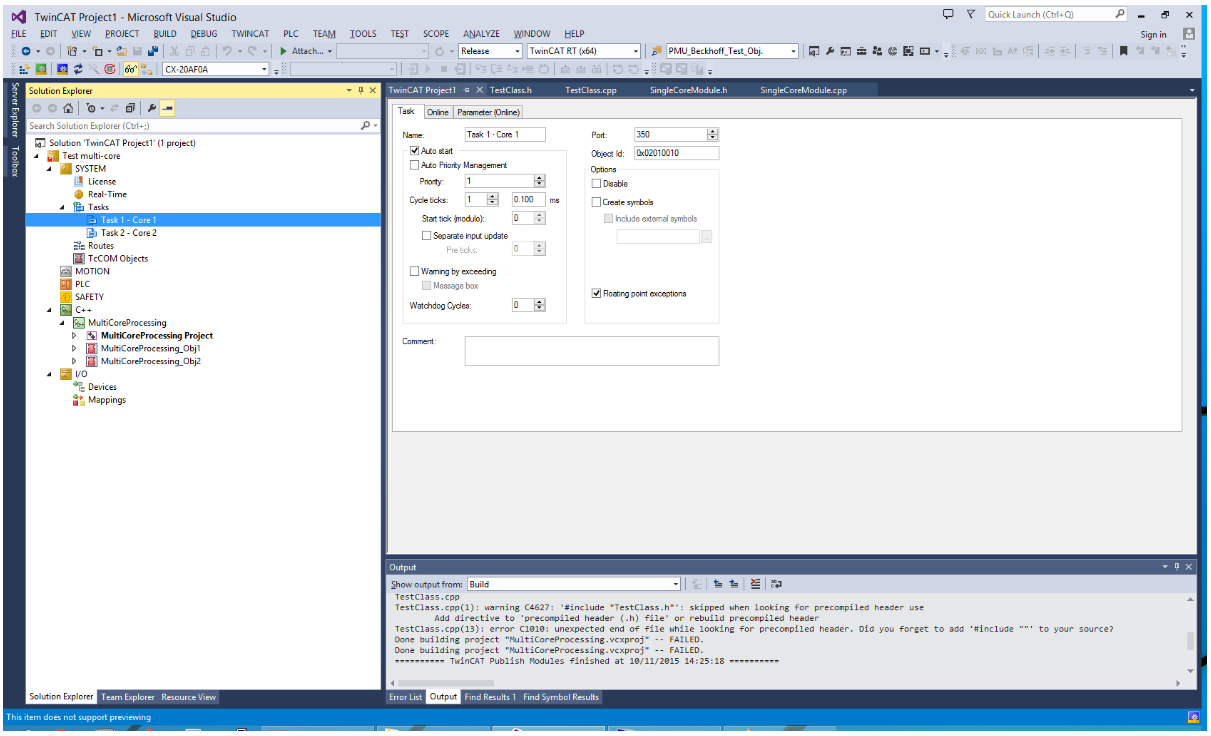
Object	RT-CPU	Base Time (ms)	Cycle Time (ms)	Cycle Ticks	Priority
Task 1 - Core 1	Default (1)	100 µs	0.100 ms	1	1
Task 2 - Core 2	Default (1)	100 µs	0.100 ms	1	2

Show output from: Build

```
TestClass.cpp
TestClass.cpp(1): warning C4627: '#include "TestClass.h"': skipped when looking for precompiled header use
        Add directive to 'precompiled header (.h) file' or rebuild precompiled header
TestClass.cpp(13): error C1010: unexpected end of file while looking for precompiled header. Did you forget to add '#include ""' to your source?
Done building project "MultiCoreProcessing.vcxproj" -- FAILED.
Done building project "MultiCoreProcessing.vcxproj" -- FAILED.
========== TwinCAT Publish Modules finished at 10/11/2015 14:25:18 ==========
```

Figure 6: Assigning module instances to processor cores

Then create a task for each C++ module instance:



TwinCAT Project1 - Microsoft Visual Studio

Task | Online | Parameter (Online)

Name: Task 1 - Core 1
[✓] Auto start
[ ] Auto Priority Management
Priority: 1
Cycle ticks: 1 0.100 ms
Start tick (modulo): 0
[ ] Separate input update
Pre ticks: 0
[ ] Warning by exceeding
[ ] Message box
Watchdog Cycles: 0

Port: 350
Object Id: 0x02010010
Options
[ ] Disable
[ ] Create symbols
[ ] Include external symbols
[✓] Floating point exceptions

Comment:

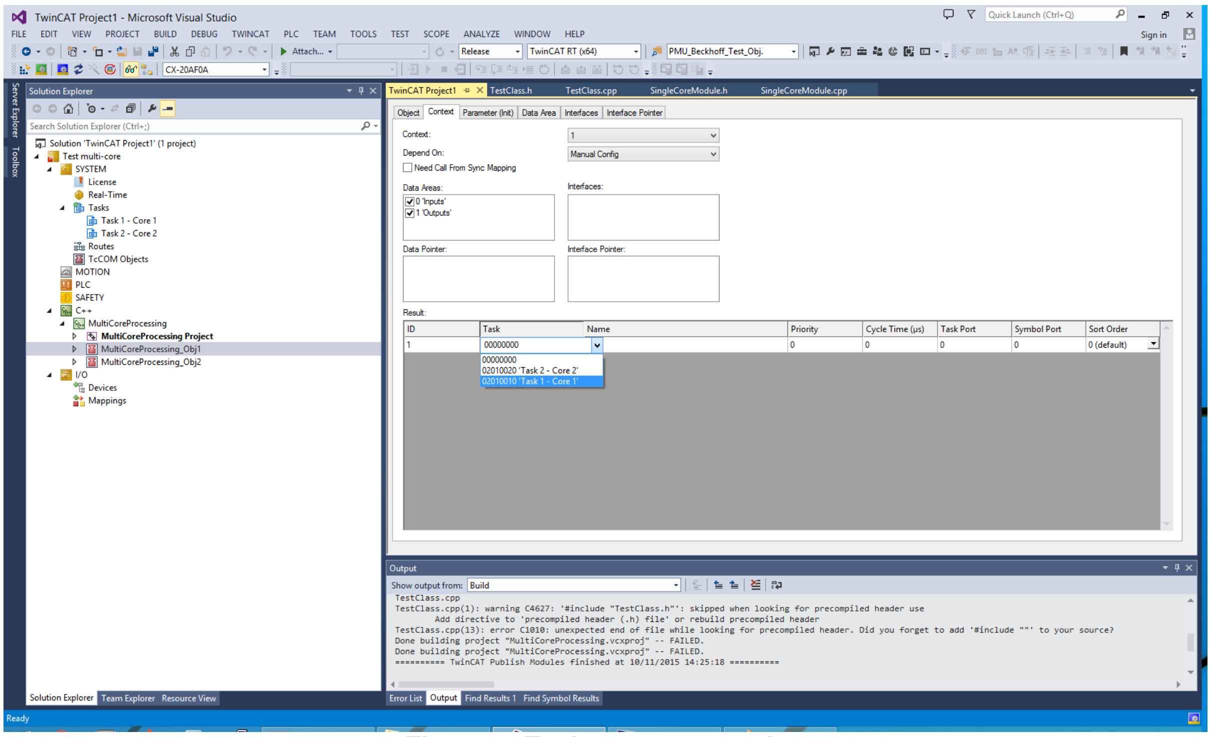
Output — Show output from: Build

```
TestClass.cpp
TestClass.cpp(1): warning C4627: '#include "TestClass.h"': skipped when looking for precompiled header use
        Add directive to 'precompiled header (.h) file' or rebuild precompiled header
TestClass.cpp(13): error C1010: unexpected end of file while looking for precompiled header. Did you forget to add '#include ""' to your source?
Done building project "MultiCoreProcessing.vcxproj" -- FAILED.
Done building project "MultiCoreProcessing.vcxproj" -- FAILED.
========== TwinCAT Publish Modules finished at 10/11/2015 14:25:18 ==========
```

This item does not support previewing

Figure 7: Task configuration

For each module instance, select the “Context” tab and assign the module to a task:



TwinCAT Project1 - Microsoft Visual Studio

Object | Context | Parameter (Init) | Data Area | Interfaces | Interface Pointer

Context: 1
Depend On: Manual Config
[ ] Need Call From Sync Mapping

Data Areas:
[✓] 0 'Inputs'
[✓] 1 'Outputs'

Interfaces:

Data Pointer:

Interface Pointer:

Result:

ID	Task	Name	Priority	Cycle Time (µs)	Task Port	Symbol Port	Sort Order
1	00000000		0	0	0	0	0 (default)

00000000
02010020 'Task 2 - Core 2'
02010010 'Task 1 - Core 1'

Output — Show output from: Build

```
TestClass.cpp
TestClass.cpp(1): warning C4627: '#include "TestClass.h"': skipped when looking for precompiled header use
        Add directive to 'precompiled header (.h) file' or rebuild precompiled header
TestClass.cpp(13): error C1010: unexpected end of file while looking for precompiled header. Did you forget to add '#include ""' to your source?
Done building project "MultiCoreProcessing.vcxproj" -- FAILED.
Done building project "MultiCoreProcessing.vcxproj" -- FAILED.
========== TwinCAT Publish Modules finished at 10/11/2015 14:25:18 ==========
```

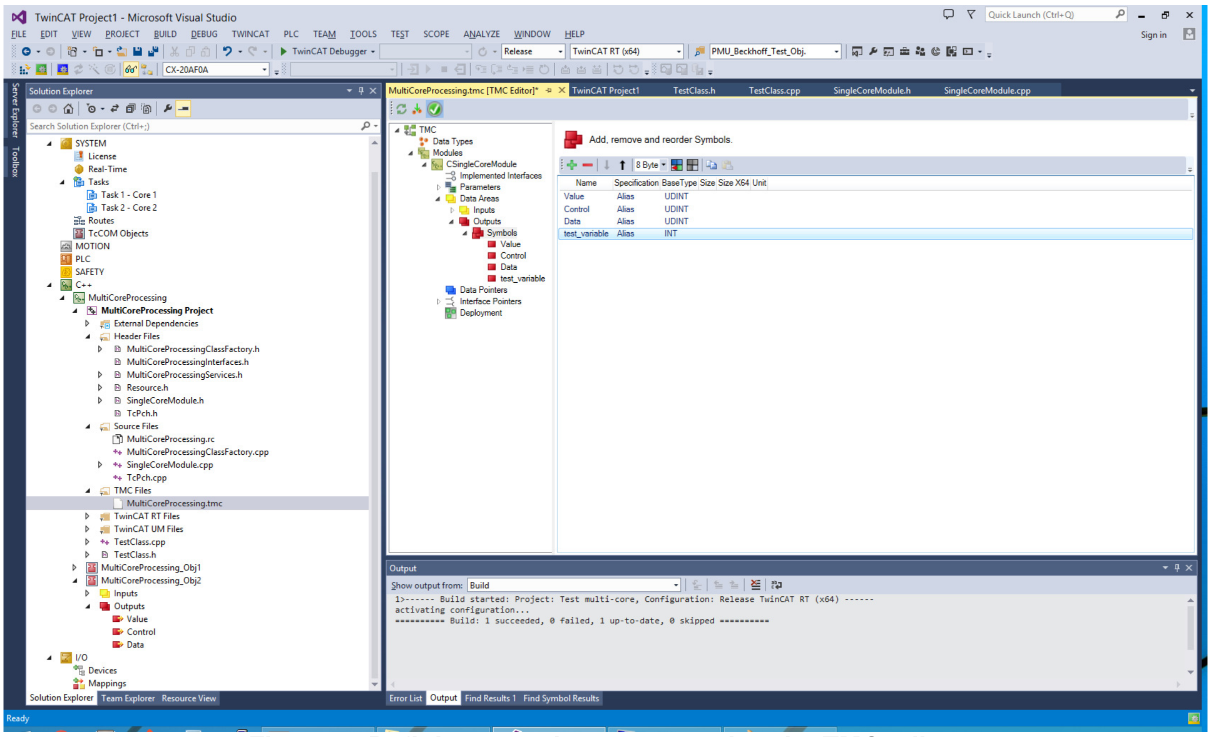
Ready

Figure 8: Task context mapping

More details on task configuration are available here: http://infosys.beckhoff.com/english.php?content=../content/1033/tc3_system/html/tcsysmgr_systemnode_subnodes_tasks.htm&id=15611.

4.3 Using a TMC File

The TMC file, with its associated editor GUI, defines the functionality of a C++ module. For example, Outputs – which are visible to other TwinCAT modules – can be defined:


TwinCAT Project1 - Microsoft Visual Studio
MultiCoreProcessing.tmc [TMC Editor]*
Add, remove and reorder Symbols.
Solution Explorer
Output

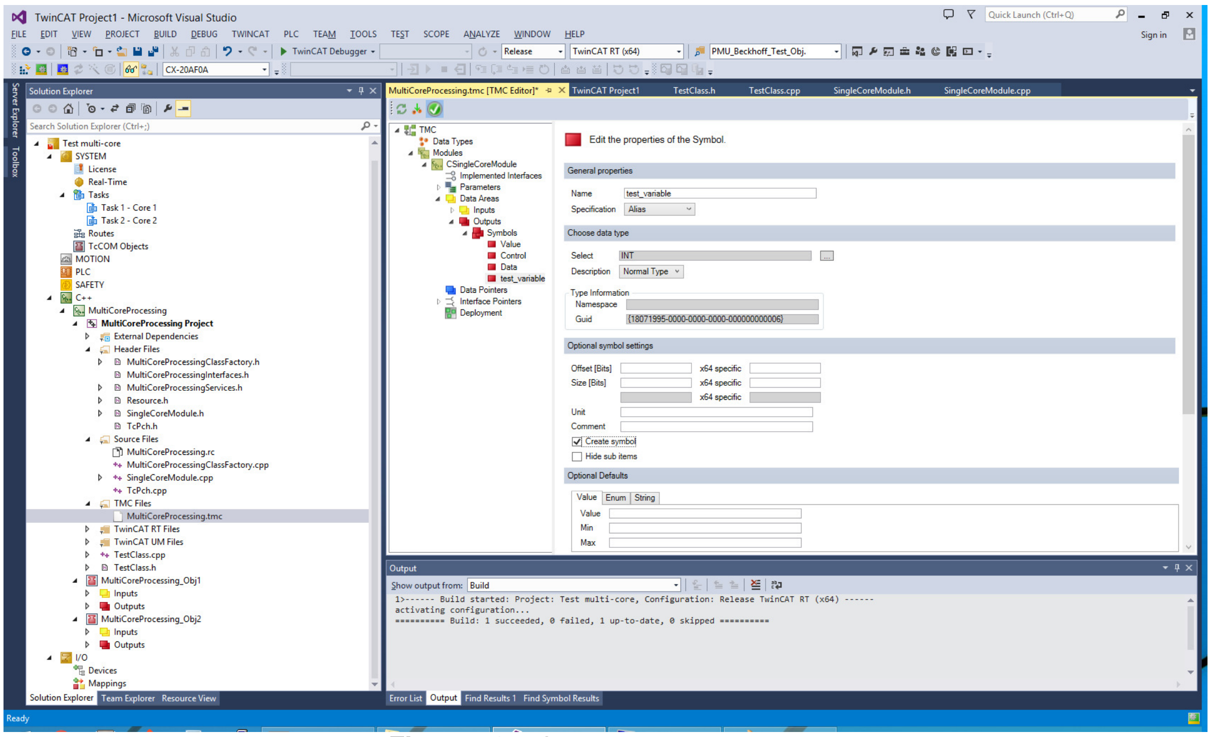
Name	Specification	BaseType	Size	Size X64	Unit
Value	Alias	UDINT			
Control	Alias	UDINT			
Data	Alias	UDINT			
test_variable	Alias	INT			

Show output from: Build

```
1>------ Build started: Project: Test multi-core, Configuration: Release TwinCAT RT (x64) ------
activating configuration...
========== Build: 1 succeeded, 0 failed, 1 up-to-date, 0 skipped ==========
```

Figure 9: Defining a module output using the TMC editor

In order to be able to monitor variables in real-time, select “Create symbol” in the editor:


TwinCAT Project1 - Microsoft Visual Studio
MultiCoreProcessing.tmc [TMC Editor]*
Edit the properties of the Symbol.
General properties
Choose data type
Optional symbol settings
Optional Defaults
Solution Explorer
Output

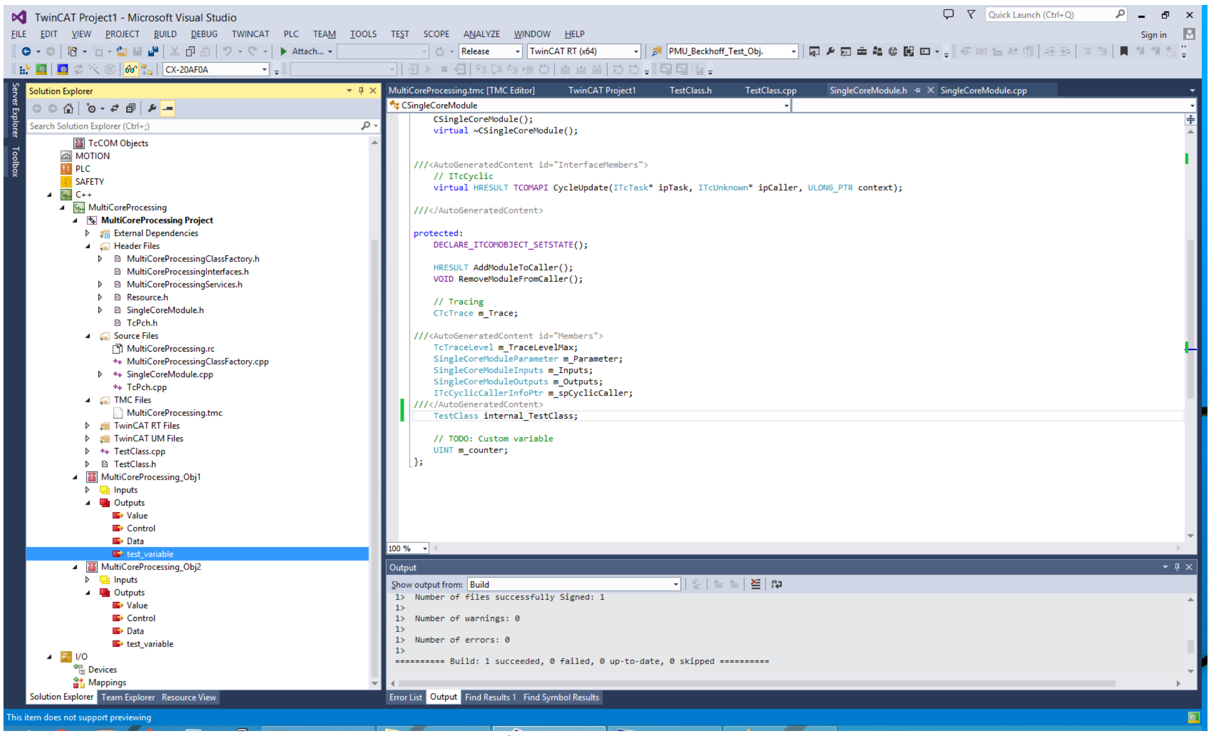
Name: test_variable
Specification: Alias
Select: INT
Description: Normal Type
Guid: {18071995-0000-0000-0000-000000000006}
[✓] Create symbol
[ ] Hide sub items

Show output from: Build

```
1>------ Build started: Project: Test multi-core, Configuration: Release TwinCAT RT (x64) ------
activating configuration...
========== Build: 1 succeeded, 0 failed, 1 up-to-date, 0 skipped ==========
```

Figure 10: “Create symbol” option

After making changes to a TMC file, save the file, right click on the module, and select “TwinCAT TMC Code Generator”. This updates the generated code. The changes should automatically appear for all module instances:

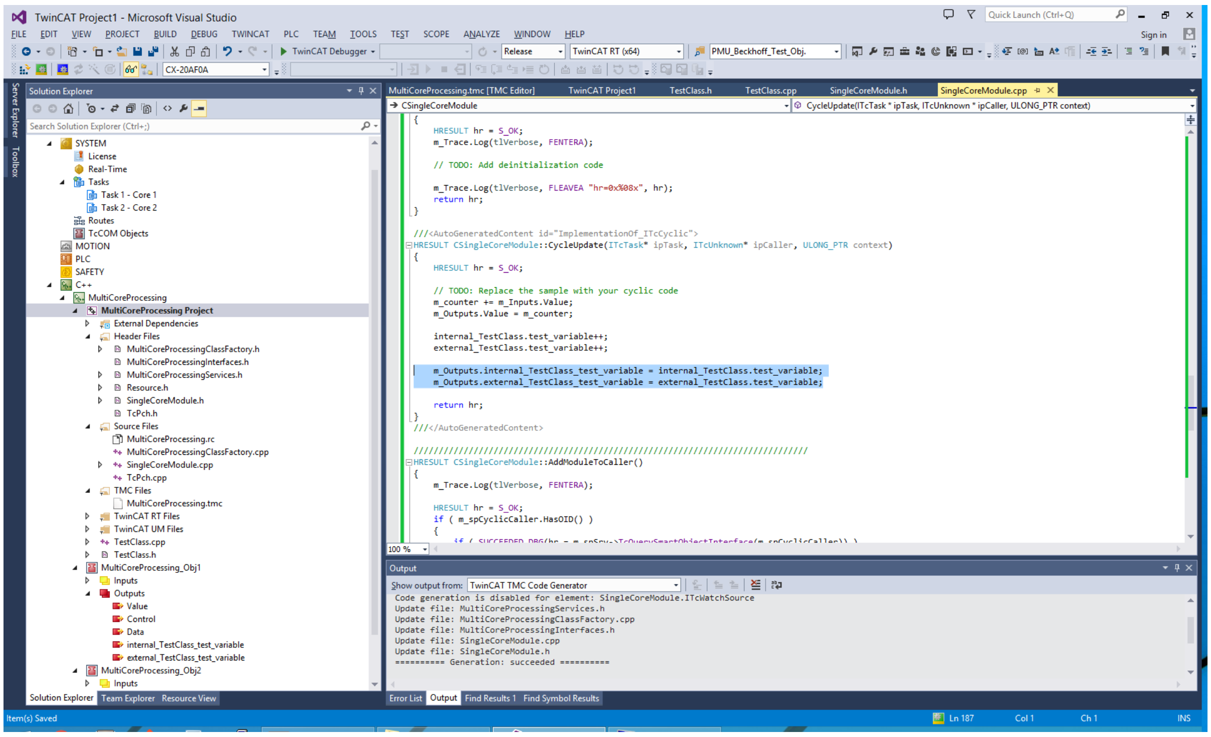


```
    CSingleCoreModule();
    virtual ~CSingleCoreModule();


///<AutoGeneratedContent id="InterfaceMembers">
    // ITcCyclic
    virtual HRESULT TCOMAPI CycleUpdate(ITcTask* ipTask, ITcUnknown* ipCaller, ULONG_PTR context);

///</AutoGeneratedContent>

protected:
    DECLARE_ITCOMOBJECT_SETSTATE();

    HRESULT AddModuleToCaller();
    VOID RemoveModuleFromCaller();

    // Tracing
    CTcTrace m_Trace;

///<AutoGeneratedContent id="Members">
    TcTraceLevel m_TraceLevelMax;
    SingleCoreModuleParameter m_Parameter;
    SingleCoreModuleInputs m_Inputs;
    SingleCoreModuleOutputs m_Outputs;
    ITcCyclicCallerInfoPtr m_spCyclicCaller;
///</AutoGeneratedContent>
    TestClass internal_TestClass;

    // TODO: Custom variable
    UINT m_counter;
};
```

```
1>  Number of files successfully Signed: 1
1>
1>  Number of warnings: 0
1>
1>  Number of errors: 0
1>
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Figure 11: Two Module Class objects, with a common definition

The definitions can be accessed from C++ code. For example, two output variables can be assigned values as follows:



```
{
    HRESULT hr = S_OK;
    m_Trace.Log(tlVerbose, FENTERA);

    // TODO: Add deinitialization code

    m_Trace.Log(tlVerbose, FLEAVEA "hr=0x%08x", hr);
    return hr;
}

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

    // TODO: Replace the sample with your cyclic code
    m_counter += m_Inputs.Value;
    m_Outputs.Value = m_counter;

    internal_TestClass.test_variable++;
    external_TestClass.test_variable++;

    m_Outputs.internal_TestClass_test_variable = internal_TestClass.test_variable;
    m_Outputs.external_TestClass_test_variable = external_TestClass.test_variable;

    return hr;
}
///</AutoGeneratedContent>

///////////////////////////////////////////////////////////////////////////////
HRESULT CSingleCoreModule::AddModuleToCaller()
{
    m_Trace.Log(tlVerbose, FENTERA);

    HRESULT hr = S_OK;
    if ( m_spCyclicCaller.HasOID() )
    {
```

```
Code generation is disabled for element: SingleCoreModule.ITcWatchSource
Update file: MultiCoreProcessingServices.h
Update file: MultiCoreProcessingClassFactory.cpp
Update file: MultiCoreProcessingInterfaces.h
Update file: SingleCoreModule.cpp
Update file: SingleCoreModule.h
========== Generation: succeeded ==========
```

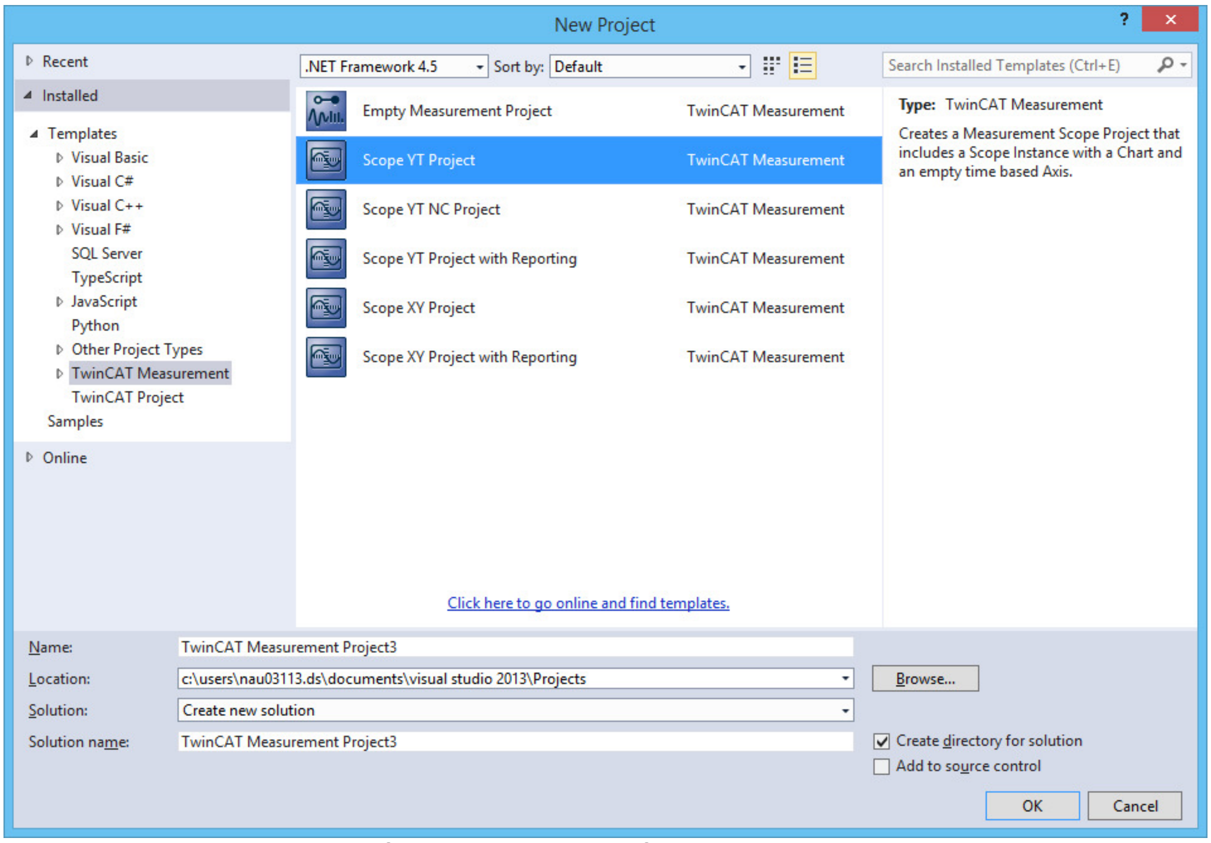
Figure 12: Assignment of output values

Once all required definitions have been added to the TMC file and the required C++ code has been added, the module can now be built, although this will also be done automatically when transferring to a target. Select “Activate Configuration” in the toolbar to run the project on the TwinCAT target.

Further TMC documentation is available here: https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_c/27021598485651723.html&id=.

4.4 Real-Time Monitoring

Data in TwinCAT projects can be monitored with high granularity in real-time. Go to “File” > “Add” > “New Project...” > “TwinCAT Measurement”, then select the appropriate scope type:


New Project

Empty Measurement Project	TwinCAT Measurement
Scope YT Project	TwinCAT Measurement
Scope YT NC Project	TwinCAT Measurement
Scope YT Project with Reporting	TwinCAT Measurement
Scope XY Project	TwinCAT Measurement
Scope XY Project with Reporting	TwinCAT Measurement

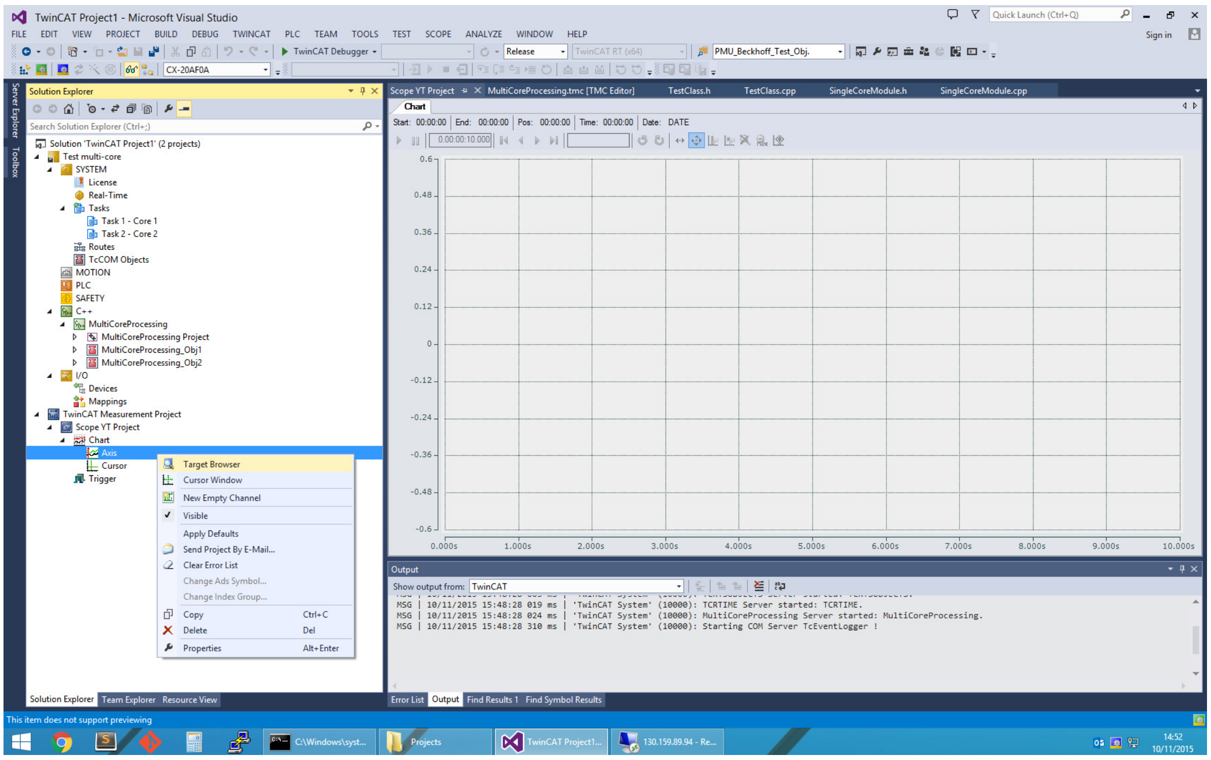
Type: TwinCAT Measurement

Creates a Measurement Scope Project that includes a Scope Instance with a Chart and an empty time based Axis.

Name: TwinCAT Measurement Project3
Location: c:\users\nau03113.ds\documents\visual studio 2013\Projects
Solution: Create new solution
Solution name: TwinCAT Measurement Project3
[✓] Create directory for solution
[ ] Add to source control

Figure 13: Creating a new TwinCAT Measurement project

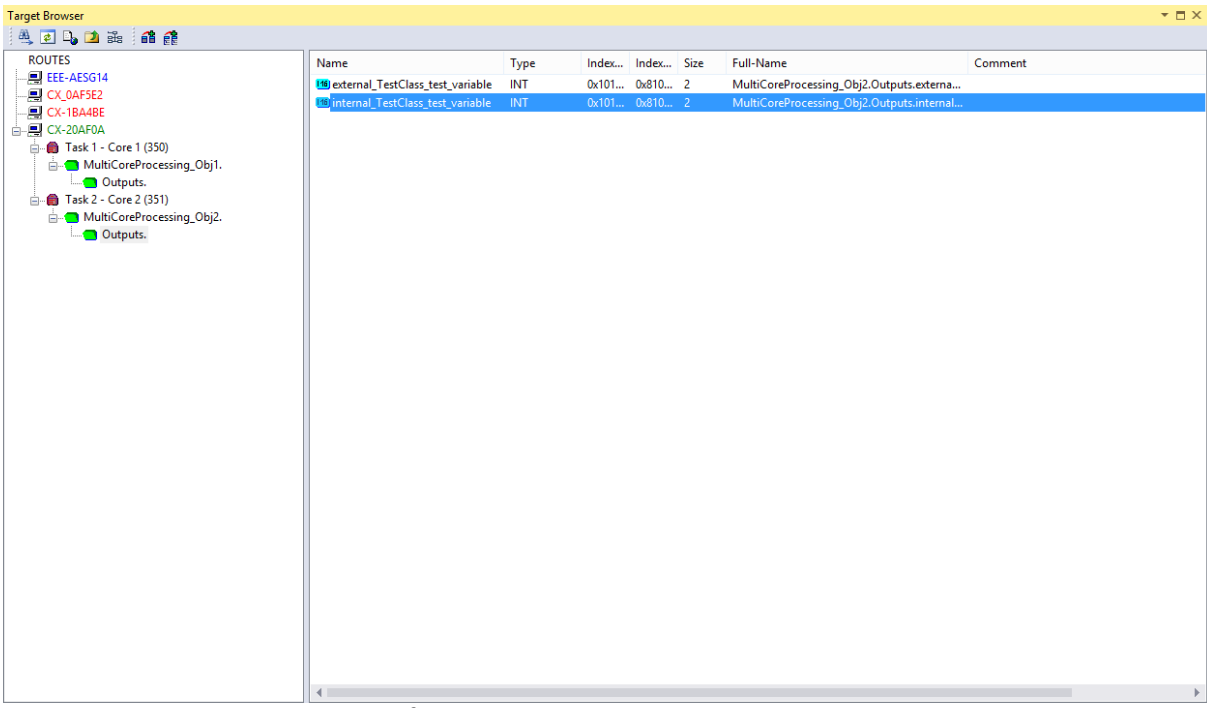
Open the Target Browser by right-clicking on the newly-created “Axis” element:



```
MSG | 10/11/2015 15:48:28 019 ms | 'TwinCAT System' (10000): TCRTIME Server started: TCRTIME.
MSG | 10/11/2015 15:48:28 024 ms | 'TwinCAT System' (10000): MultiCoreProcessing Server started: MultiCoreProcessing.
MSG | 10/11/2015 15:48:28 310 ms | 'TwinCAT System' (10000): Starting COM Server TcEventLogger !
```

Figure 14: Adding data to plots

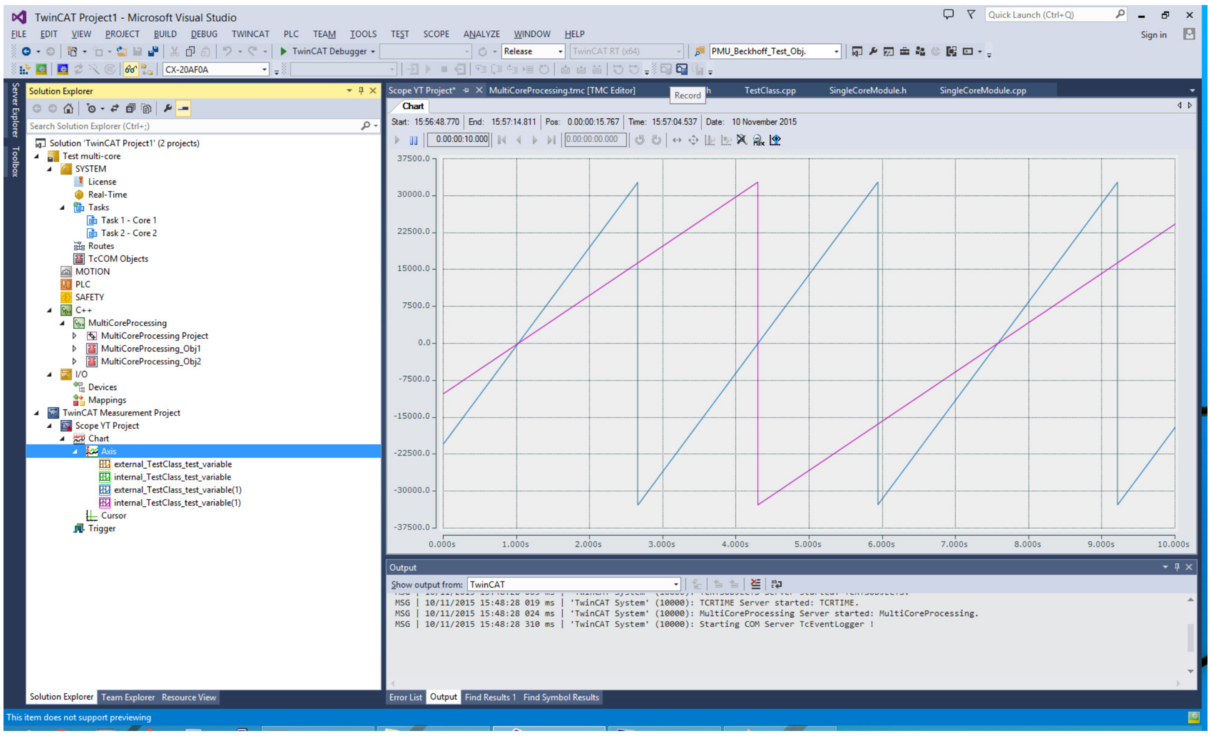
Add the relevant signals to your plot:



Name	Type	Index...	Index...	Size	Full-Name	Comment
external_TestClass_test_variable	INT	0x101...	0x810...	2	MultiCoreProcessing_Obj2.Outputs.externa...	
internal_TestClass_test_variable	INT	0x101...	0x810...	2	MultiCoreProcessing_Obj2.Outputs.internal...	

Figure 15: Selecting signals from the Target Browser

Press the Record button on the toolbar to capture data:



```
MSG | 10/11/2015 15:48:28 019 ms | 'TwinCAT System' (10000): TCRTIME Server started: TCRTIME.
MSG | 10/11/2015 15:48:28 024 ms | 'TwinCAT System' (10000): MultiCoreProcessing Server started: MultiCoreProcessing.
MSG | 10/11/2015 15:48:28 310 ms | 'TwinCAT System' (10000): Starting COM Server TcEventLogger !
```

TwinCAT Project1 - Microsoft Visual Studio
Solution Explorer
Solution 'TwinCAT Project1' (2 projects)
Test multi-core
SYSTEM
License
Real-Time
Tasks
Task 1 - Core 1
Task 2 - Core 2
Routes
TcCOM Objects
MOTION
PLC
SAFETY
C++
MultiCoreProcessing
MultiCoreProcessing Project
MultiCoreProcessing_Obj1
MultiCoreProcessing_Obj2
I/O
Devices
Mappings
TwinCAT Measurement Project
Scope YT Project
Chart
Axis
external_TestClass_test_variable
internal_TestClass_test_variable
external_TestClass_test_variable(1)
internal_TestClass_test_variable(1)
Cursor
Trigger
Start: 15:56:48.770 End: 15:57:14.811 Pos: 0.00:00:15.767 Time: 15:57:04.537 Date: 10 November 2015
Show output from: TwinCAT
This item does not support previewing

Figure 16: Viewing plots of module data

For the trial license of TwinCAT 3 (which can be renewed perpetually), up to one hour of data, per signal, can be captured.

5 Integrating Simulink Models

There are two approaches for integrating an algorithm or control system implemented in Simulink (which may include performance-critical components written directly in C) into the Beckhoff platform, and these approaches are described in the following subsections. Both approaches involve the automatic generation of a C or C++ code implementation of a Simulink model, using Simulink Coder [1], [2].

5.1 Manual Compilation of Generated C++ Code within TwinCAT

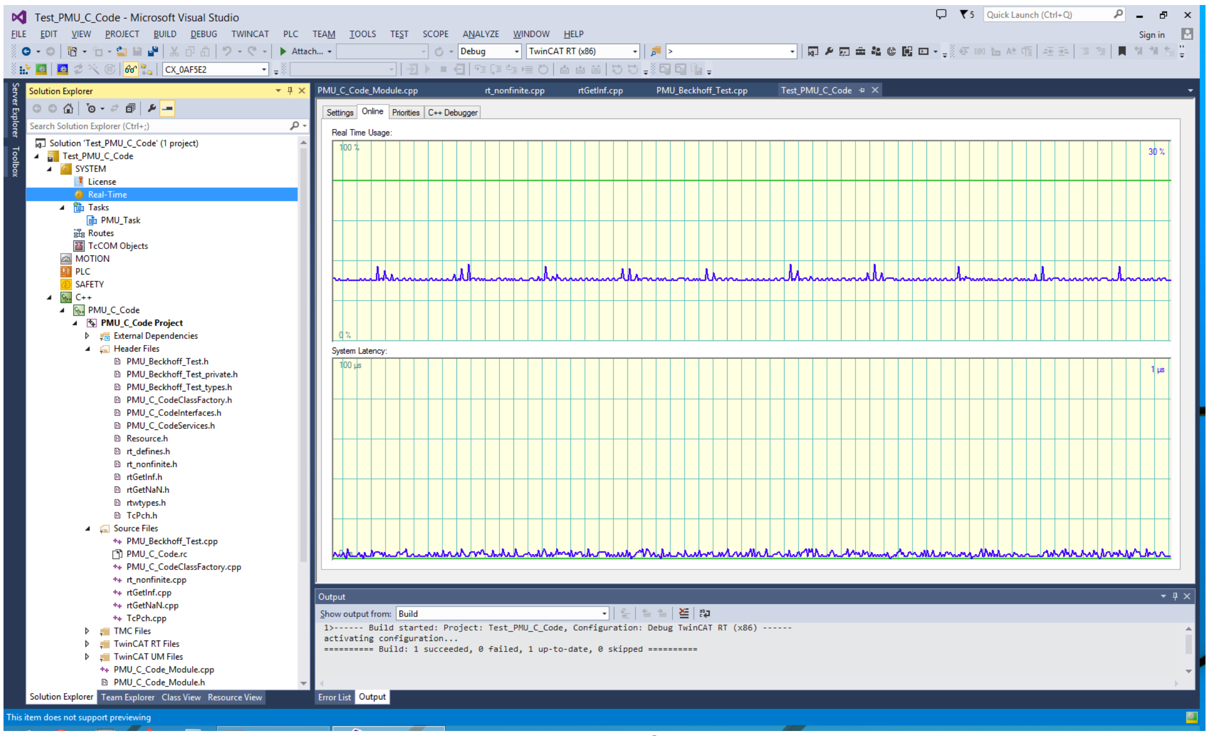
Simulink Coder can be used to create a C or C++ implementation of a Simulink model, which is intended to be optimised for embedded, real-time applications. For integration with TwinCAT 3, Simulink Coder should be configured to generate C++ code for a 32-bit x86 platform (unless a different controller unit architecture is being targeted). Double-precision floating-point numbers can be used.

The generated C++ code can be included in a TwinCAT 3 project within a C++ Module Class, using the template for “cyclic” tasks (see Figure 2). The code from Simulink Coder should wrap the Simulink model in a C++ class. An instance of this class should be defined as a `protected` member in the header file of the Module Class (i.e., not simply declared arbitrarily in the C++ code). The Simulink model should be initialised within the `SetObjStatePS()` function within the main Module Class C++ file, for example:

```
// State transition from PREOP to SAFEOP
//
// Initialize input parameters
// Allocate memory
HRESULT CPMU_M_class::SetObjStatePS(PTComInitDataHdr pInitData)
{
      m_Trace.Log(tlVerbose, FENTERA);
      HRESULT hr = S_OK;
      IMPLEMENT_ITCOMOBJECT_EVALUATE_INITDATA(pInitData);

      // Simulink model initialization
      PMU_Beckhoff_Test_Obj.initialize();

      m_Trace.Log(tlVerbose, FLEAVEA "hr=0x%08x", hr);
      return hr;
}
```

The task should be configured to execute on a particular time-base, such as 1 ms or 100 µs, which matches the time-step of the Simulink model. For example, the CX1020 controller unit – a relatively basic controller with a 1 GHz single-core CPU – can execute the phasor measurement (PMU) algorithm documented in [3] in real-time with a 100 µs time-base, requiring 30-40% of the CPU time[1]. This is illustrated in Figure 17.

[1] This only includes the CPU time for the PMU algorithm; however, the additional CPU time requirements for communications processing are not significant, and can be allocated to a lower priority task than the time-critical algorithm processing.



```
1>------ Build started: Project: Test_PMU_C_Code, Configuration: Debug TwinCAT RT (x86) ------
activating configuration...
========== Build: 1 succeeded, 0 failed, 1 up-to-date, 0 skipped ==========
```

Figure 17: PMU algorithm CPU execution time

There are two caveats with this approach:

1. In TwinCAT 3, precompiled headers must be manually disabled for the imported Simulink source files. This is a simple change within the C++ compiler settings.
2. It is required to use platform-specific library calls for mathematical functions, which are better suited for real-time operation, rather than those defined in the standard C "math.h" library. A simple method for achieving this is to replace the "`#include <math.h>`" line in the main C++ header file generated by Simulink Coder with:

```
#include <fpu87.h>
#include "math_h_defines.h"
```

where "`math_h_defines.h`" provides a mapping of the standard C versions to the TwinCAT versions:

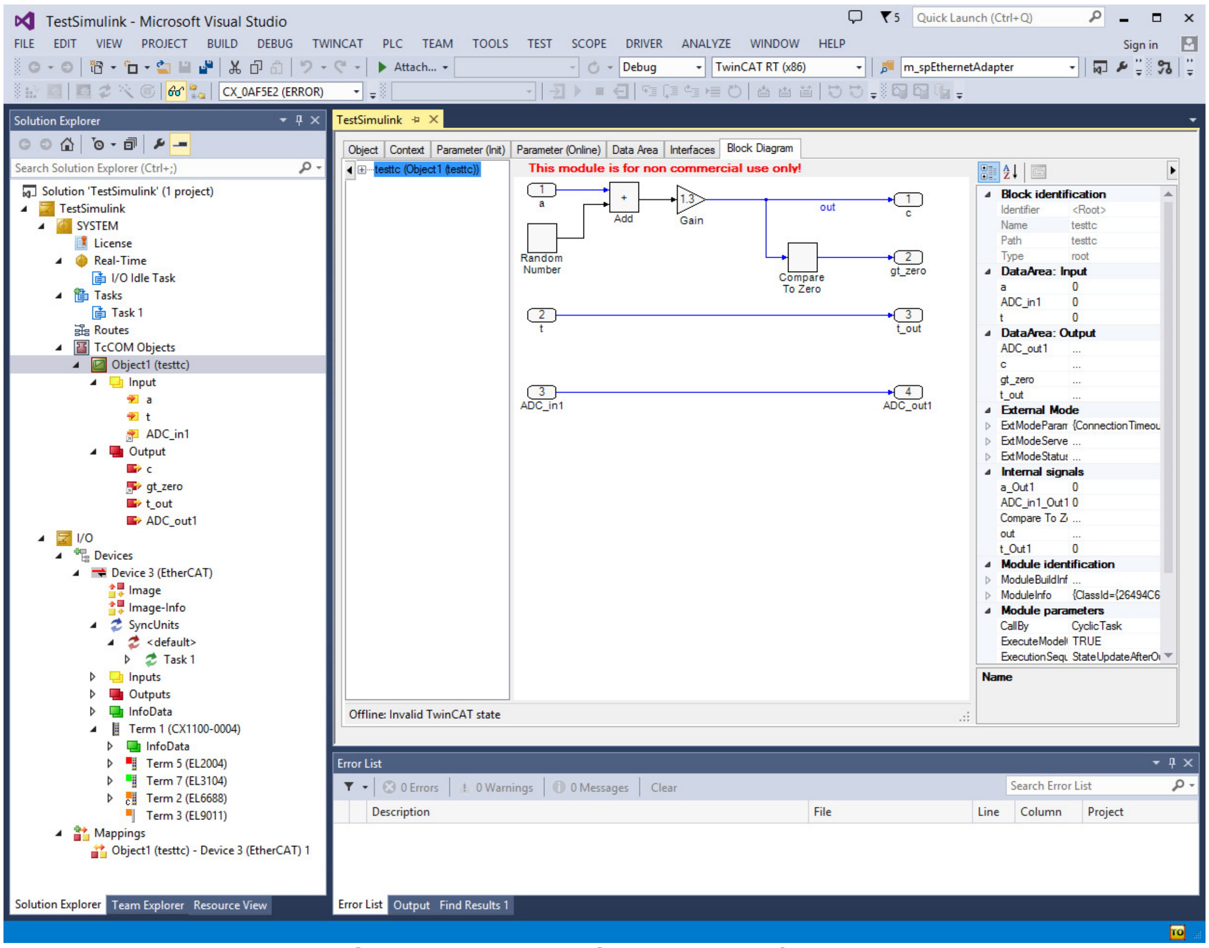
```
#define sqr sqr_
#define sqrt sqrt_
#define sin sin_
#define cos cos_
#define tan tan_
#define atan atan_
#define atan2 atan2_
#define asin asin_
#define acos acos_
#define exp exp_
#define log log_
#define log10 log10_
#define fabs fabs_
#define fmod fmod_
#define ceil ceil_
#define floor floor_
#define pow pow_
#define sincos sincos_
```

```
#define fmodabs fmodabs_
#define round round_
#define rounddigits rounddigits_
#define coubic coubic_
#define ldexp ldexp_
#define ldexpf ldexpf_
#define sinh sinh_
#define cosh cosh_
#define tanh tanh_
#define finite finite_
#define isnan isnan_
#define rands rands_
```

Note that this will result in compiler warnings, but allows the module to be built and executed.

5.2 Using Simulink-TwinCAT 3 Integration

There is a Beckhoff software tool for directly integrating Simulink models with TwinCAT 3. This is achieved simply by selecting the TwinCAT 3 “target” with Simulink Coder (rather than the normal embedded target); once the code is generated, the module is automatically available within TwinCAT 3. A simple example, which illustrates the integration of a Simulink “block” model into the TwinCAT 3 user interface, is shown in Figure 18.



TestSimulink - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TWINCAT PLC TEAM TOOLS TEST SCOPE DRIVER ANALYZE WINDOW HELP
Sign in
Attach...
Debug
TwinCAT RT (x86)
m_spEthernetAdapter
CX_0AF5E2 (ERROR)
Solution Explorer
Search Solution Explorer (Ctrl+;)
TestSimulink
Object Context Parameter (Init) Parameter (Online) Data Area Interfaces Block Diagram
testtc (Object1 (testtc))
1 a
Random Number
Add
1.3 Gain
out
Compare To Zero
1 c
2 gt_zero
2 t
3 t_out
3 ADC_in1
4 ADC_out1
Solution Explorer Team Explorer Resource View
Error List
0 Errors 0 Warnings 0 Messages Clear
Search Error List
Description File Line Column Project
Error List Output Find Results 1

Solution 'TestSimulink' (1 project)
- TestSimulink
  - SYSTEM
    - License
    - Real-Time
      - I/O Idle Task
    - Tasks
      - Task 1
    - Routes
    - TcCOM Objects
      - Object1 (testtc)
        - Input
          - a
          - t
          - ADC_in1
        - Output
          - c
          - gt_zero
          - t_out
          - ADC_out1
  - I/O
    - Devices
      - Device 3 (EtherCAT)
        - Image
        - Image-Info
        - SyncUnits
          - <default>
            - Task 1
        - Inputs
        - Outputs
        - InfoData
        - Term 1 (CX1100-0004)
          - InfoData
          - Term 5 (EL2004)
          - Term 7 (EL3104)
          - Term 2 (EL6688)
          - Term 3 (EL9011)
    - Mappings
      - Object1 (testtc) - Device 3 (EtherCAT) 1

This module is for non commercial use only!

Block identification	
Identifier	<Root>
Name	testtc
Path	testtc
Type	root
DataArea: Input	
a	0
ADC_in1	0
t	0
DataArea: Output	
ADC_out1	...
c	...
gt_zero	...
t_out	...
External Mode	
ExtModeParam	{ConnectionTimeou
ExtModeServe	...
ExtModeStatus	...
Internal signals	
a_Out1	0
ADC_in1_Out1	0
Compare To Ze	...
out	...
t_Out1	0
Module identification	
ModuleBuildInf	...
ModuleInfo	{ClassId={26494C6
Module parameters	
CallBy	CyclicTask
ExecuteModel	TRUE
ExecutionSequ	StateUpdateAfterO

Name

Offline: Invalid TwinCAT state

Figure 18: Simple example of Simulink-TwinCAT 3 integration

This tool is intended to be more convenient and to allow easier debugging than direct integration of C++ code as described in Section 5.1. Simulink model inputs and outputs are automatically identified in TwinCAT 3, as illustrated in the left of Figure 18. Signals in the Simulink model can be monitored in real-time whilst the project is executed on the target hardware. However, this approach requires a separate software licence and the code generation process typically takes significantly longer than for manual C++ code integration.

Furthermore, the process in Section 5.1 allows additional "wrapper" C++ code to be added to the Module Class, without needing to add an additional task. Conversely, the direct Simulink-TwinCAT integration forces all task functionality to be implemented within Simulink.

6 References

[1] MathWorks, “Simulink Coder - Generate C and C++ code from Simulink and Stateflow models,” 2012. [Online]. Available: http://www.mathworks.co.uk/products/simulink-coder/.

[2] A. J. Roscoe, S. M. Blair, and G. M. Burt, “Benchmarking and optimisation of Simulink code using Real-Time Workshop and Embedded Coder for inverter and microgrid control applications,” in *Universities Power Engineering Conference (UPEC)*, 2009, pp. 1–5.

[3] A. J. Roscoe, “Exploring the Relative Performance of Frequency-Tracking and Fixed-Filter Phasor Measurement Unit Algorithms Under C37.118 Test Procedures, the Effects of Interharmonics, and Initial Attempts at Merging P-Class Response With M-Class Filtering,” *IEEE Trans. Instrum. Meas.*, vol. 62, no. 8, pp. 2140–2153, Aug. 2013.