

Documentation Report

MATLAB real-time control development for Windows 10

Fakultät IV: Elektrotechnik und Informatik  
Institut für Energie- und Automatisierungstechnik  
Fachgebiet Regelungssysteme

Vorgelegt von  
Böttcher, Nils m-nr.: 313335  
Fröhlich, Andre m-nr.: 0348517  
Nowak, Stanislaw m-nr.: 0346690

am   
19.10.2017

Betreut von  
Constantin Wiesener

1. Inhaltsverzeichnis

[I Inhaltsverzeichnis 1](#_Toc504841624)

[1 Introduction 2](#_Toc504841625)

[1.1 Motivation 2](#_Toc504841626)

[2 Compiler Setup 2](#_Toc504841627)

[2.1 Installation 2](#_Toc504841628)

[3 Coding 3](#_Toc504841629)

[3.1 Code comparison between linux and mac 3](#_Toc504841630)

[3.2 Code adjustments for Windows 10 3](#_Toc504841631)

[4 Necessary configuration for creating new model 5](#_Toc504841632)

[II Bibliography 6](#_Toc504841633)

[III Table of figures 8](#_Toc504841634)

[IV Tabellenverzeichnis 9](#_Toc504841635)

[V Attachment 10](#_Toc504841636)

[V.I windows\_file\_process.tlc 10](#_Toc504841637)

[V.II windows\_main.tlc 11](#_Toc504841638)

# Introduction

## Motivation

MATLAB Real-Time open source protocols and libraries for controlling external Hardware (i.e. neuro-stimulators), are only available under Mac and Linux operating systems. Most companies and research facilities use Windows Operating Systems. Therefore, a suitable solution has to be implemented under the newest Windows 10 OS. In order to port the Mac/Linux model over to Windows 10, the given code has to be analyzed and modified with regard to their suitability. These includes Simulink Coder with the Generic Real-Time Target, Simulink Real-Time, Simulink Desktop Real-Time and Soft-Real-Time Simulation. The final MATLAB-code should enable a user to use any device interface port such as USB in Real-Time mode under Windows 10.

# Compiler Setup

To be able to modify the model, changing the Simulink settings or code of the core data, and compile it to a new executable one needs an appropriate c/c++ compiler. For windows the MinGW compiler is chosen.

## Installation

To install it you either do it by downloading it from the Add-Ons menu in MATLAB called “MATLAB Support for MinGW-w64 C/C++ Compiler” or install it manually. When the compiler from the Add-Ons menu was installed, nothing has to be done. If the compiler was installed outside MATLAB, the environment variable has to be set manually. The following instruction is working for MATLAB R2015b through R2017a. One needs at least the version 4.9.2 of minGW-w64.

(I) Setting the system environment variable:

1. Administrative privileges are required.  
2. Select Computer from Start menu.  
3. Choose System properties from the Context menu.  
4. Then advanced system settings into advanced tab (normally first tab opened).  
5. Go to environment variables.  
6. Under system variables, select New.  
7. In the new system variable dialog box, type MW\_MINGW64\_LOC in the variable name field.  
8. In the variable value field, type the location of the MinGW-w64 compiler installation (for example: ‘C: \programs\TDM-GCC-64’).  
9. Now click on “OK” button to close the dialog box. Close the control Panel dialog box.

(II) Setting the MW\_MINGW64 environment variable for MATLAB following command line has to be entered:

*setenv(‘MW\_MINGW64\_LOC’, ‘destination’)*

Instead of destination, one has to write the installation folder of MinGW-w64 like in (I) point 8. This command has to be run every time MATLAB is started anew. If this is to be avoided, creating a ‘startup.m’ file is advised.

To verify, that the compiler was recognized by MATLAB run the following command:

*mex -setup*

Now your system should be set to be able to compile the needed data and create your own executables. The original MinGW setup guide and further information can be found under following link: “https://de.mathworks.com/MATLABcentral/answers/313298-i-already-have-mingw-on-my-computer-how-do-i-configure-it-to-work-with-MATLAB.

# Coding

The upcoming text explaines where, what and why certain code lines have been changed or added (compared to Linux-/mac-version) to be able to build the Simulink-model, which is then to be used on external hardware. It is also to note, that the three files “windows\_file\_process.tlc”, “windows\_main.tlc” and “test.slx” have to be in the same folder in order for the compiler to be able to create the needed executable.

## Code comparison between Linux and mac

First, the already done mac and Linux versions were compered to each other. The found differences were then taken as reference points for needed changes to be able to run it under windows. The vary command lines were used to find the corresponding windows commands. Which is not quite trivial, as many of them could not be easily translated as they did not have a one code line solution, like in Linux/mac.

## Code adjustments for Windows 10

First adjustment was made in the *windows\_file\_process.tlc*. MATLAB-Function *%<DisableGenerateExampleMain()>* in line 35 is added compared to Linux/Mac. It overwrites the example main file created by MATLAB. This has to be done in avoidance of generating two main functions. For the given Linux/Mac target the compiling process was managed by the *ert\_linux.tmf (ert\_mac.tmf)* file. This file contained all information for the compiler to build the desired target with an external MinGw compiler. Since there was the function mentioned above it is not necessary on windows to create a .tmf-file.

The *windows\_main.tlc* consist of several functions and classes, which have to be correctly translated from Linux syntax to windows one. For that, the first *%function generateDeclarations() Output* in Line-range 21-170 has been adjusted. First, following includes were added: *#include "windows.h", #include <ntstatus.h>* and *#include <winternl.h>* in Lines 32, 34, 36. The last two guarantee timer interrupt and real time under Windows. The additional function, commented as *Realtime functions, Timer function, TSC timestamp function, Global Variables* ranging from 76 to 135. *Realtime functions* set the priority class to real time. *Typedef* in Line 88 and *TSC timestamp function* in Line 105 create access to ntDLL.dll which serves as a timer for the executable. *pSetTimerResolution* and *pQueryTimerResolution* functions added to control windows timer as done in lines 132, 378, 388. This is done, because the timer in windows variates between 0.5 milliseconds and 15.6 milliseconds, depending on the management of the windows scheduler. Because the timer value fluctuates between the mentioned period it is not classified as real time. The *setFunction* in lines 201, 336, 344, 388, 389, 395 is being used to reset the timer to the lowest value of 0.5 milliseconds after every execution of *sub-rate*, *base rate* and *main class*. This way it is insured, that the named timer has a constant value and the model works in real time with a frequency up to 2 kHz.

Next, *sub-rate* functions are in lines 184 to 211. *Sub-rate* has been slightly expended, in comparison to Linux version, so the semaphores work with the additionally created timer.

The *base-rate* function ranging from line 216 to 358. The biggest code addition starting in line 253. Here it was necessary to replace the *sub-rate* overrun check function as a one-line solution like it is done in Linux isn´t working. Also, the “Trigger sub-rate threads” part starting in line 283 was slightly expanded, due to different command control of semaphores under windows. The windows semaphores are well documented as well as the thread priority classes and the time measurement functions. For that reason, the detailed function is not explained in this document. The sleeptime calculation starting in line 322 is changed from Linux/Mac towards the counterpart in windows. The measured sleeptime fills up the time between the desired step time and the code execution time. In Addition, as stated before, the *setFunction* is being called in the end of the *base-rate* function.

Last is the *main function* going from line 363 to 499. The actual *pSetTimerResolution* and *pQueryTimerResolution* functions are accessed through the library *NTDLL.dll*. In lines from 371 to 406 the *NTDLL.dll* is being loaded in and the *setFunction* initialized. Also a few print command lines were added for debugging purposes. As seen in lines 374, 381, 386, 391, 394, 398, 404. Furthermore, in the main function, starting in line 431, the semaphores are initialized and the threads created. The termination of the program with the *CTRL+C-key* combination was taken from the Linux/Mac-version.

# Necessary configuration for creating new model

If a new Simulink model is to be used with the given files or the old model is to be changed then it is important that following settings are present. Under model *configuration parameters -* *solver* - *solver options,* set *type* to *fixed-step* as shown below in Figure 1.

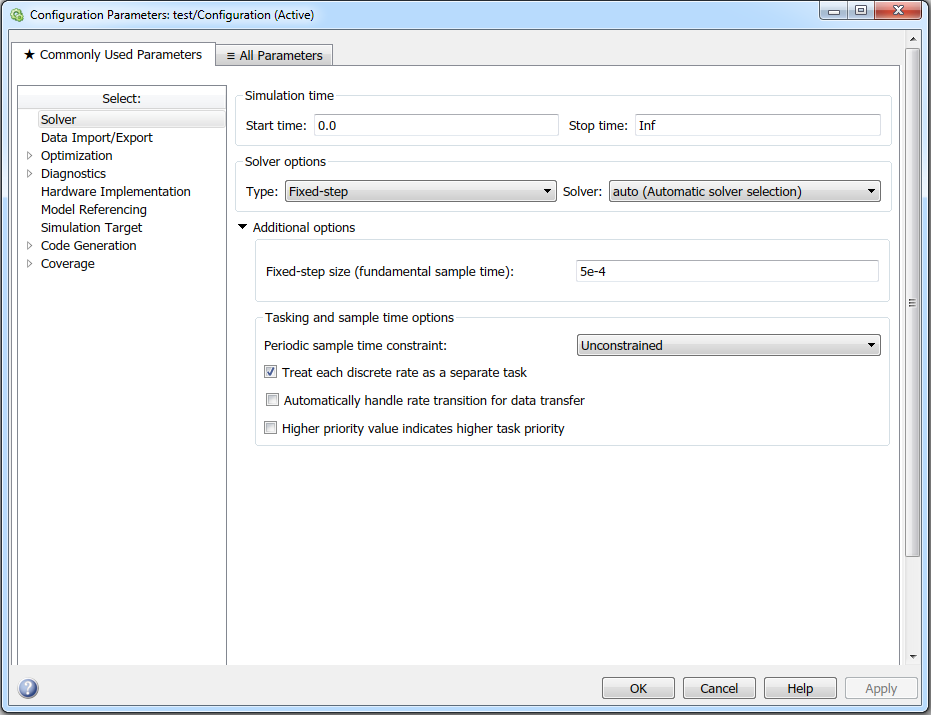


Figure 1: Solver settings

Next, under *code generation* set *System target file* to *ert.tlc* as in Figure 2. Furthermore, in *toolchain settings* set *toolchain* to MinGW64 compiler.

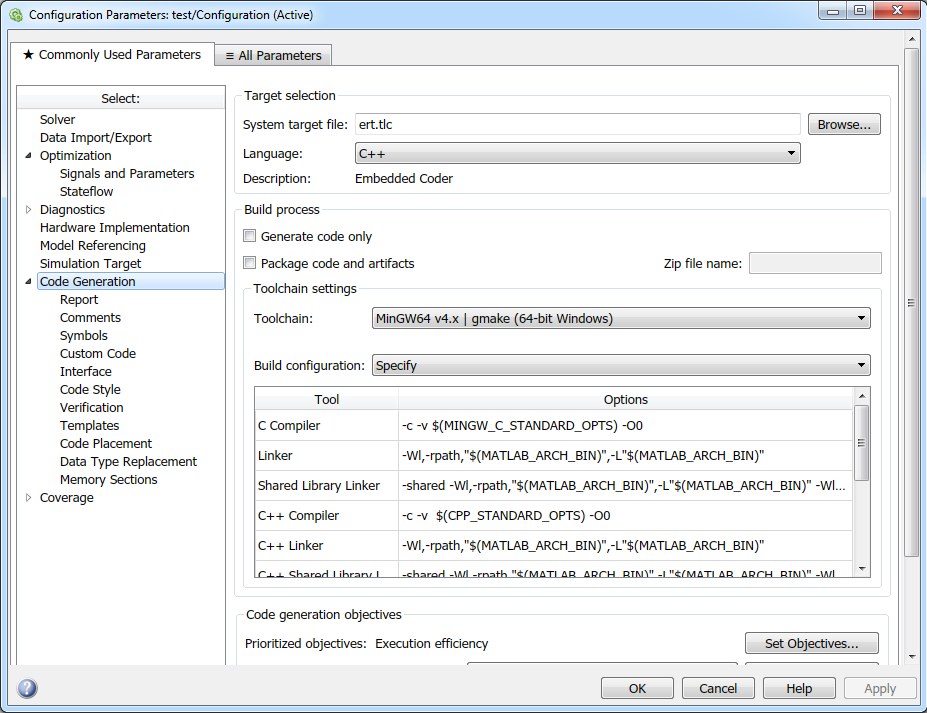


Figure 2: Code Generation settings

Accordingly, to Figure 3, under *Code Generation – Interface – Code Interface* set *Code Interface packaging* to *Nonreusable function*.

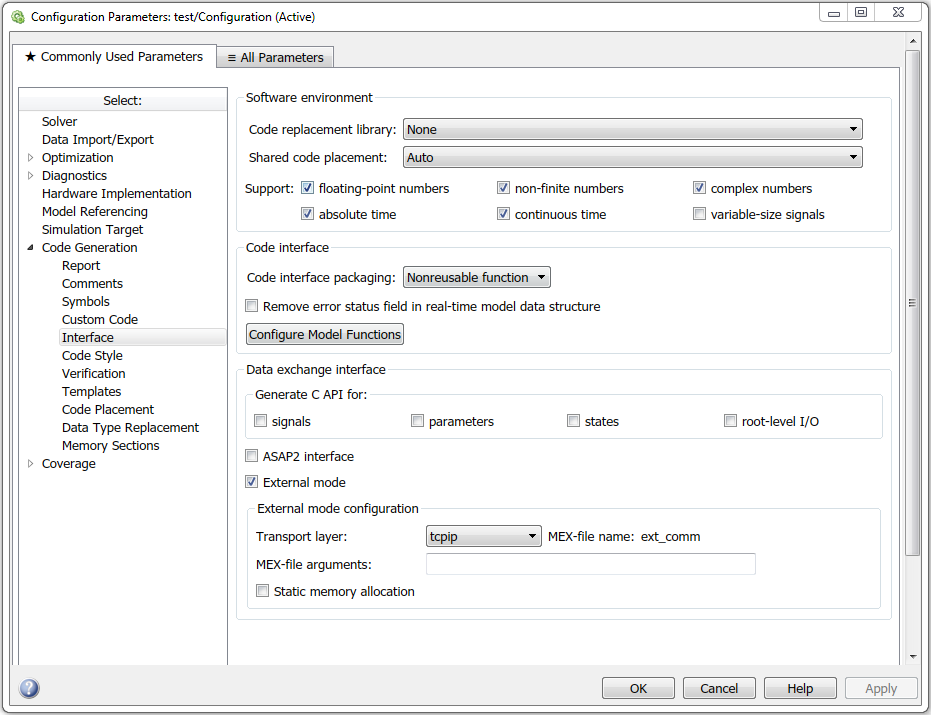


Figure 3: Code Generation - Interface settings

Finally, the right template has to be selected. Under *Code Generation – Templates – Custom templates* set *File customization template* to *windows\_file\_process.tlc*. For reference see Figure 4. It ensures that the created executable of the current model is made with the function given by *windows\_main.tlc*.

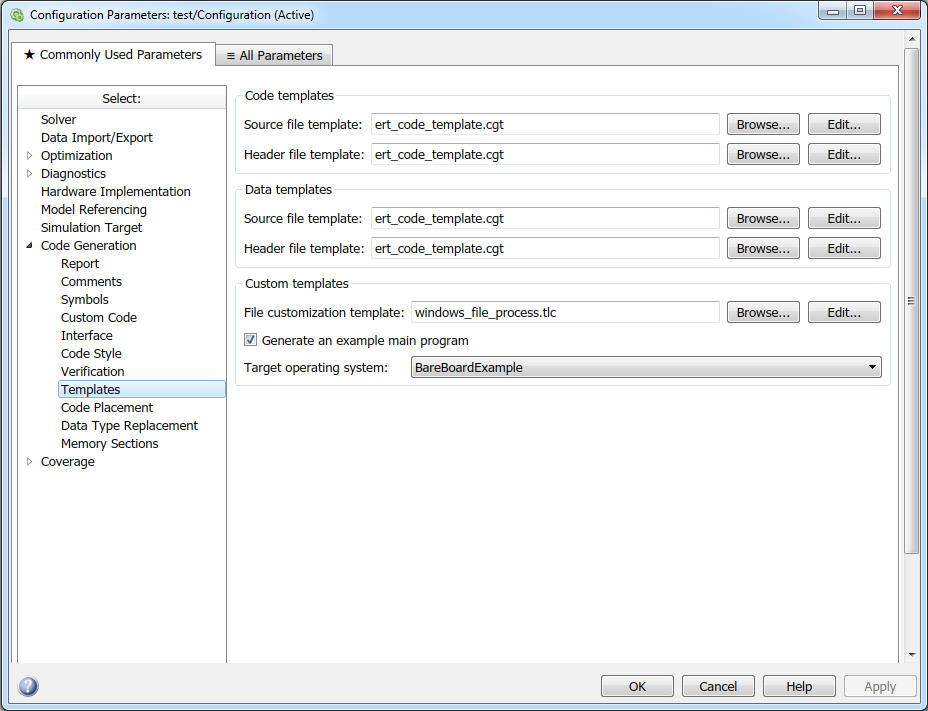


Figure 4: Code Generation - Template settings

As far as it was doable, other settings has been found out not to hinder the building process of the model. This may vary after manipulation of the main files.

1. Bibliography

[CIE82] Internationale Beleuchtungskommission: CIE N° 30-2 (TC-4.6), calculation and measurement of luminance and illuminance in road lighting, Bureau central de la CIE, 52. Boulevard Malesherbes, 75008 Paris – France, 1982

[DIN-EN 13201-3] Deutsches Institut für Normung e. V.: DIN EN 13201-3:2015, Straßenbeleuchtung – Teil 3: Berechnung der Gütemerkmale, Beuth Verlag GmbH, Berlin, 2015

[Bae96] Baer, R.: Beleuchtungstechnik. [1], Grundlagen, Verlag Technik, Berlin, 1996

[LiTG67] Fachausschuss Außenbeleuchtung: Methoden der Leuchtdichteberechnung für Straßenbeleuchtung, Lichttechnische Gesellschaft e. V., Karlsruhe, 1967

1. Table of figures

[Figure 1: Solver settings 5](#_Toc504822106)

[Figure 2: Code Generation settings 5](#_Toc504822107)

[Figure 3: Code Generation - Interface settings 6](#_Toc504822108)

[Figure 4: Code Generation - Template settings 6](#_Toc504822109)

1. Tabellenverzeichnis

[Tabelle 1: Standard Format der r-Tabelle [DIN-EN 13201-3, S.18] **Fehler! Textmarke nicht definiert.**](#_Toc496022357)

[Tabelle 2: gemessene r-Tabelle der Teppichprobe (Laboranalyse) **Fehler! Textmarke nicht definiert.**](#_Toc496022358)

[Tabelle 3: Berechnete r-Tabelle des Kellerteppichs (Labormessung) **Fehler! Textmarke nicht definiert.**](file:///C:\Users\Andre\Desktop\Bachelor\neu\Schrift\Bachelorarbeit_Andre_Fröhlich.docx#_Toc496022359)

[Tabelle 4: r-Tabelle des Laufstegbohrkerns (Bohrkernanalyse von BK1) **Fehler! Textmarke nicht definiert.**](file:///C:\Users\Andre\Desktop\Bachelor\neu\Schrift\Bachelorarbeit_Andre_Fröhlich.docx#_Toc496022360)

[Tabelle 5: r-Tabelle des Laufstegbohrkerns (Bohrkernanalyse von BK2) **Fehler! Textmarke nicht definiert.**](file:///C:\Users\Andre\Desktop\Bachelor\neu\Schrift\Bachelorarbeit_Andre_Fröhlich.docx#_Toc496022361)

[Tabelle 6: berechnete r-Tabelle des Laufstegs (erste Messung) **Fehler! Textmarke nicht definiert.**](#_Toc496022362)

[Tabelle 7: berechnete r-Tabelle des Laufstegs (zweite Messung) **Fehler! Textmarke nicht definiert.**](#_Toc496022363)

[Tabelle 8: berechnete r-Tabelle des Laufstegs (dritte Messung) **Fehler! Textmarke nicht definiert.**](file:///C:\Users\Andre\Desktop\Bachelor\neu\Schrift\Bachelorarbeit_Andre_Fröhlich.docx#_Toc496022364)

[Tabelle 9: Kosten-/Zeitaufwand für Bohrkernentnahme und -messung **Fehler! Textmarke nicht definiert.**](file:///C:\Users\Andre\Desktop\Bachelor\neu\Schrift\Bachelorarbeit_Andre_Fröhlich.docx#_Toc496022365)

[Tabelle 10: kosten-/Zeitaufwand für die Leuchtdichtemethode **Fehler! Textmarke nicht definiert.**](file:///C:\Users\Andre\Desktop\Bachelor\neu\Schrift\Bachelorarbeit_Andre_Fröhlich.docx#_Toc496022366)

1. Attachment
   1. windows\_file\_process.tlc
2. %% Abstract:
3. %%   Example Embedded Coder custom file processing **template**.
4. %%
5. %%   Note: This file can contain any valid TLC code, which Embedded Coder
6. %%   executes just prior to writing the generated source files to disk.
7. %%   Using **this** **template** "hook" file, you are able to augment the generated
8. %%   source code and create additional files.
9. %%
10. %% Copyright 1994-2010 The MathWorks, Inc.
11. %%
12. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
13. %selectfile NULL\_FILE
15. %assign ert\_mainC = LibCreateSourceFile("Source", "Custom", "ert\_main")

18. %assign ::modelName = CompiledModel.Name
19. %assign ::solverMode = CompiledModel.FixedStepOpts.SolverMode
20. %assign ::fundamentalStepSize = CompiledModel.FundamentalStepSize
21. %assign ::numSampleTimes = CompiledModel.NumSynchronousSampleTimes
22. %assign ::extMode = CompiledModel.ConfigSet.ExtMode
24. %% parse sample times and offsets
25. %assign j = 0
26. %foreach i = CompiledModel.NumSampleTimes
27. %**if** CompiledModel.SampleTime[i].Asynchronous == "no"
28. %assign ::sampleTime%<j> = CompiledModel.SampleTime[i].PeriodAndOffset[0]
29. %assign ::offset%<j> = CompiledModel.SampleTime[i].PeriodAndOffset[1]
30. %assign j = j+1
31. %endif
32. %endforeach
34. %include "windows\_main.tlc"
36. %<DisableGenerateExampleMain()>
38. %openfile buff
39. %<generateDeclarations()>
40. %closefile buff
41. %<LibSetSourceFileSection(ert\_mainC, "Declarations", buff)>
43. %openfile buff
44. %<generateRtOneStep()>
45. %<generateMain()>
46. %closefile buff
47. %<LibSetSourceFileSection(ert\_mainC, "Functions", buff)>
    1. windows\_main.tlc
48. %% Abstract:
49. %%   Example main **for** bare board target (multirate model)
50. %%
51. %% Copyright 1994-2013 The MathWorks, Inc.
52. %%
53. %selectfile NULL\_FILE
55. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DisableGenerateExampleMain \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/
56. %function DisableGenerateExampleMain() **void**
58. %**if** GenerateSampleERTMain
59. %assign ::CompiledModel.GenerateSampleERTMain = TLC\_FALSE
60. %warning Overriding example ert\_main.c!
61. %endif
63. %endfunction
64. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/
66. %assign ::tid01Eq = LibGetTID01EQ()
68. %function generateDeclarations() Output

71. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Includes \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/
72. // Multirate - Multitasking case main file
74. #include <stdio.h>                     // This ert\_main.c example uses printf/fflush
75. #include <signal.h>
76. #include <stdlib.h>
77. #include <stdbool.h>
78. #include <unistd.h>
79. #include "windows.h"
80. #pragma warning(disable: 4005)
81. #include <ntstatus.h>
82. #pragma warning(default: 4005)
83. #include <winternl.h>
84. #include <tlhelp32.h>
85. #include <cstdlib>
86. #include "%<modelName>.h"              // Model's header file
87. #include "rtwtypes.h"                  // MathWorks types
88. %**if** extMode == 1
89. #include "ext\_work.h"                  // External mode header file
90. #include <intrin.h>
91. %endif
92. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/


96. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Defines \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/
97. #ifndef TRUE
98. #define TRUE true
99. #define FALSE false
100. #endif
102. //!  ==================\*
103. //! \* Required defines \*
104. //! \*==================\*/
106. #ifndef MODEL
107. # error Must specify a model name.  Define MODEL=name.
108. #else
109. // create generic macros that work with any model
110. # define EXPAND\_CONCAT(name1,name2) name1 ## name2
111. # define CONCAT(name1,name2) EXPAND\_CONCAT(name1,name2)
112. # define MODEL\_INITIALIZE CONCAT(MODEL,\_initialize)
113. # define MODEL\_STEP       CONCAT(MODEL,\_step)
114. # define MODEL\_TERMINATE  CONCAT(MODEL,\_terminate)
115. # define RT\_MDL           CONCAT(MODEL,\_M)
116. #endif
117. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/




123. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Realtime functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/
124. **BOOL** RealTimePriorityClass()
125. {
126. **return** SetPriorityClass(GetCurrentProcess(), REALTIME\_PRIORITY\_CLASS);
127. }
129. **BOOL** RealTimeThreadPriority()
130. {
131. **return** SetThreadPriority(GetCurrentThread(), THREAD\_PRIORITY\_TIME\_CRITICAL);
132. }
133. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/
135. **typedef** NTSTATUS (NTAPI\* pSetTimerResolution)(**ULONG** RequestedResolution, **BOOLEAN** Set, **PULONG** ActualResolution);
136. **typedef** NTSTATUS (NTAPI\* pQueryTimerResolution)(**PULONG** MinimumResolution, **PULONG** MaximumResolution, **PULONG** CurrentResolution);
138. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Timer function \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/
139. **HANDLE** timer;
140. LARGE\_INTEGER li;

143. **DWORD** WINAPI nanosleep(**LPVOID** arg){
144. wprintf(L"The nanosleep() thread ID is %u\n", GetCurrentThreadId());
145. timer = CreateWaitableTimer(NULL, TRUE, NULL);
147. }
148. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/


152. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TSC timestamp function \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/
153. **LONGLONG** rdtsc(){
154. **return** \_\_rdtsc();
155. }
156. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/


160. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Signal handler for ABORT during simulation \*\*\*\*\*\*\*\*\*\*\*\*\* \*/
161. **void** abortHandler(**int** sig) {
162. fprintf(stderr, "Simulation aborted by pressing CTRL+C\n");
163. rtmSetStopRequested(%<modelName>\_M, 1);
164. }
165. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

168. //!
169. //! Thread handle of the base rate thread.
170. //! Fundamental sample time = %<fundamentalStepSize>s
171. //!
172. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Global Variables \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/
173. **double** dTimeDiff, StartTime, ActualTime, NextTime ;
174. **LONGLONG** g\_Frequency, g\_CurrentCount, g\_LastCount ;
175. **LPCTSTR** extMode\_semaphore = "extMode\_semaphore";
176. **LONG** SemCount = 0;
177. LARGE\_INTEGER freqcounter ;
178. NTSTATUS status;
179. pSetTimerResolution setFunction;
180. pQueryTimerResolution queryFunction;
181. **ULONG** minResolution, maxResolution, actualResolution;
182. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/


186. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Sub Rate Declarations \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/
187. // Thread handles of the sub rates and semaphores for sub rate threads. The array
188. // is indexed by TID, i.e. the first one or two elements are unused.
189. %foreach i = numSampleTimes
190. %**if** i == 0 || i == 1 && tid01Eq
191. %**continue**
192. %endif
193. %assign s = sampleTime%<i>
194. %assign o = offset%<i>
195. // TID%<i>: sample time = %<s>s, offset = %<o>s
196. %endforeach

199. **struct** sub\_rate {
200. **HANDLE** **thread**;
201. **HANDLE**    sem;
202. } sub\_rate[%<numSampleTimes>];
203. //! Flag if the simulation has been terminated.
204. //!
205. **int** simulationFinished = 0;
206. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/



211. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ext mode Handle \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/
212. %**if** extMode == 1
213. // Indication that the base rate thread has started
214. **HANDLE** ext\_mode\_ready;
215. %endif
217. %endfunction
218. // End of function generateDeclarations



223. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Debug function \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/
224. %function printfunc() Output
225. //printf("%s\n", \_\_func\_\_);
226. %endfunction
227. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/


231. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* sub rates \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/
232. // Start of function to generate all sub rate functions
233. %function generateRtOneStep() Output
234. %foreach j = numSampleTimes - 1
235. %assign i = j + 1
236. %**if** i == 1 && tid01Eq
237. %**continue**
238. %endif
240. **LPCTSTR** sub\_rate\_%<i>\_semaphore = "sub\_rate\_%<i>\_semaphore";
242. **DWORD** WINAPI sub\_rate%<i>(**LPVOID** arg)
243. {
244. wprintf(L"The sub\_rate%<i>() thread ID is %u\n", GetCurrentThreadId());
245. **if** (!RealTimeThreadPriority())**return** -1;
246. sub\_rate[%<i>].sem = OpenSemaphore(SYNCHRONIZE | SEMAPHORE\_MODIFY\_STATE, FALSE, sub\_rate\_%<i>\_semaphore);
247. **while**(!simulationFinished) {
248. setFunction(maxResolution, TRUE, &actualResolution);
249. WaitForSingleObject(sub\_rate[%<i>].sem,INFINITE);    // sem\_val = 1
250. %<printfunc()>
251. %<modelName>\_step%<i>();
252. WaitForSingleObject(sub\_rate[%<i>].sem,INFINITE);    // sem\_val = 0
254. }
255. **return** NULL;
256. }
257. %endforeach
258. %endfunction
259. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/


263. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* base rate \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/
264. %% GENERATEMAIN
265. %%  This function generates code of the base rate function and main function.
266. %%
267. %function generateMain() Output
268. //!
269. //! This is the thread function of the base rate loop.
270. //! Fundamental sample time = %<fundamentalStepSize>s
271. //!
272. **DWORD** WINAPI base\_rate(**LPVOID** arg)
273. {
275. wprintf(L"The base\_rate() thread ID is %u\n", GetCurrentThreadId());
276. **if** (!RealTimeThreadPriority())**return** -1;
277. %assign nsec = FEVAL("uint32", (fundamentalStepSize - FEVAL("floor", fundamentalStepSize))\*10000000)
278. **if** (!QueryPerformanceFrequency((LARGE\_INTEGER\*)&g\_Frequency))**return** -1; // Get Processor Frequency
279. boolean\_T eventFlags[%<numSampleTimes>];             // Model has %<numSampleTimes> rates
280. int\_T taskCounter[%<numSampleTimes>] = %<FcnInitializeTaskCounter()>;
281. int\_T OverrunFlags[%<numSampleTimes>];
282. **int** step\_sem\_value ;
283. int\_T i;

286. %**if** extMode == 1
287. %<SLibGenERTExtModeInit()>
288. **HANDLE** ext\_mode\_ready = OpenSemaphore(SYNCHRONIZE | SEMAPHORE\_MODIFY\_STATE, FALSE, extMode\_semaphore);
289. ReleaseSemaphore(ext\_mode\_ready,1,NULL);
290. %endif
291. QueryPerformanceCounter((LARGE\_INTEGER\*)&g\_CurrentCount);
292. // Main loop, running until all the threads are terminated
293. **while**(rtmGetErrorStatus(%<modelName>\_M) == NULL && !rtmGetStopRequested(%<modelName>\_M)) {
294. StartTime= QueryPerformanceCounter((LARGE\_INTEGER\*)&g\_CurrentCount);
295. //ActualTime = 100000000\*(double)g\_CurrentCount/(double)g\_Frequency;  // Timekeeping over QueryPerformanceCounter()
296. ActualTime = (((**double**)rdtsc()/(**double**)g\_Frequency));                 // Timekeeping over rdtsc()
297. %<printfunc()>
299. // Check subrate overrun, set rates that need to run this time step
300. %foreach i = numSampleTimes
301. %**if** i == 0 || i == 1 && tid01Eq
302. %**continue**
303. %endif
304. **if** (taskCounter[%<i>] == 0) {
305. **if** (eventFlags[%<i>]) {
306. OverrunFlags[0] = **false**;
307. OverrunFlags[%<i>] = **true**;
309. // Sampling too fast
310. rtmSetErrorStatus(%<modelName>\_M, "Overrun");
311. **return** 0;
312. }

315. eventFlags[%<i>] = **true**;
316. }
318. %assign T = FEVAL("int32", FEVAL("floor", sampleTime%<i> / fundamentalStepSize))
319. taskCounter[%<i>]++;
320. **if** (taskCounter[%<i>] == %<T>) {
321. taskCounter[%<i>]= 0;

324. }
325. %endforeach


329. // Trigger sub-rate threads
330. %foreach i = numSampleTimes
331. %**if** i == 0 || i == 1 && tid01Eq
332. %**continue**
333. %endif
334. %assign s = sampleTime%<i>
335. %assign o = offset%<i>
336. // Sampling rate %<i>, sample time = %<s>, offset = %<o>
337. **if** (eventFlags[%<i>]) {

340. eventFlags[%<i>] = FALSE;

343. **if** (ReleaseSemaphore(sub\_rate[%<i>].sem,0,&SemCount)!=0) {
344. rtmSetErrorStatus(%<modelName>\_M, "Overrun");
345. printf("Sub rate %<i> overrun, sample time=%<s>s, offset=%<o>s is too fast\n");
346. **break**;
347. }
349. **if**(SemCount !=0){
350. rtmSetErrorStatus(%<modelName>\_M, "Overrun");
351. printf("Sub rate %<i> overrun, sample time=%<s>s, offset=%<o>s is too fast\n");
352. **break**;
353. }
354. ReleaseSemaphore(sub\_rate[%<i>].sem,2,&SemCount);
355. }
356. %endforeach
358. // Execute base rate step
359. %**if** solverMode == "SingleTasking"
360. %<modelName>\_step();
361. %**else**
362. %<modelName>\_step0();
363. %endif
364. %**if** extMode == 1
365. rtExtModeCheckEndTrigger();
366. %endif

369. **do** {

372. QueryPerformanceCounter((LARGE\_INTEGER\*)&g\_LastCount);
373. //NextTime = 100000000\*((double)g\_LastCount)/(double)g\_Frequency;           // Timekeeping over QueryPerformanceCounter()
374. NextTime = (((**double**)rdtsc()/(**double**)g\_Frequency));                         // Timekeeping over rdtsc()
376. //Checking Base\_rate Overrun
377. **if** (%<nsec>-((NextTime-ActualTime)\*10000) < 0) {
378. printf("Base rate (%<fundamentalStepSize>s) overrun \n");
379. **continue**;
380. }
381. }**while** (0);
383. setFunction(maxResolution, TRUE, &actualResolution);
385. // Calculating sleeptime
386. **double** sleeptime = %<nsec> -((NextTime-ActualTime)\*10000);
387. li.QuadPart = -sleeptime;
389. // Opening Waitable timer nanosleep
390. SetWaitableTimer(timer, &li, 0, NULL, NULL, 0);
391. setFunction(maxResolution, TRUE, &actualResolution);
393. // Waiting for timer to finish
394. WaitForSingleObject(timer, INFINITE);


398. }
399. simulationFinished = 1;
400. // Final step for sub rate step functions
401. **for** (i = %<1 + tid01Eq>; i < %<numSampleTimes>; i++) {
402. ReleaseSemaphore(sub\_rate[i].sem,2,&SemCount);
404. }
405. }// end of function void \* base\_rate()
406. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/


410. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* main function \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/
411. //!
412. //! This is the main function of the model.
413. //! Multirate - Multitasking case main file
414. //!
415. int\_T main(int\_T argc, **const** char\_T \*argv[])
416. {
418. **const** **HINSTANCE** hLibrary = LoadLibrary("NTDLL.dll");
419. **if** (hLibrary == NULL)
420. {
421. printf("Failed to load NTDLL.dll (%d)\n", GetLastError());
422. **return** 1;
423. }
425. queryFunction = (pQueryTimerResolution)GetProcAddress(hLibrary, "NtQueryTimerResolution");
426. **if** (queryFunction == NULL)
427. {
428. printf("NtQueryTimerResolution is null (%d)\n", GetLastError());
429. **return** 1;
430. }
432. queryFunction(&minResolution, &maxResolution, &actualResolution);
433. printf("Win32 Timer Resolution:\n\tMinimum Value:\t%u\n\tMaximum Value:\t%u\n\tActual Value:\t%u\n\n", minResolution, maxResolution, actualResolution);
435. setFunction = (pSetTimerResolution)GetProcAddress(hLibrary, "NtSetTimerResolution");
436. **if** (setFunction == NULL)
437. {
438. printf("NtSetTimerResolution is null (%d)\n", GetLastError());
439. **return** 1;
440. }
441. printf("Setting Timer Resolution to the maximum value (%d)...\n", maxResolution);
442. status = setFunction(maxResolution, TRUE, &actualResolution);
443. **if** (status == STATUS\_SUCCESS)
444. {
445. printf("Success! (Current resolution: %d)\n\n", actualResolution);
447. }
449. **if** (status == STATUS\_TIMER\_RESOLUTION\_NOT\_SET)
450. {
451. printf("Timer not set (Return Code: %d)\n", status);
452. **return** 2;
453. }
455. **const** char\_T \*errStatus;
456. int\_T i;
458. // Some info over the generated Threads
459. wprintf(L"The main() process ID is %u\n", GetCurrentProcessId());
460. wprintf(L"The main() thread ID is %u\n", GetCurrentThreadId());
461. wprintf(L"\n");

464. **if**(!RealTimePriorityClass())**return** -1;
466. %**if** extMode == 1
467. // External mode
468. signal(SIGINT, abortHandler); // important for letting the destructor be called.
469. rtParseArgsForExtMode(argc, argv);
470. CreateSemaphore(NULL,0,1,extMode\_semaphore);
471. %**else**
472. (**void**)(argc);
473. (**void** \*)(argv);
474. %endif


478. // Initialize model
479. %<modelName>\_initialize();
480. simulationFinished = 0;
481. // Prepare task attributes \*/
483. %foreach i = numSampleTimes
484. %**if** i == 0 || i == 1 && tid01Eq
485. %**continue**
486. %endif
487. %assign s = sampleTime%<i>
488. %assign o = offset%<i>
489. // Initializing the step semaphore of the loop %<i> \*/
490. CreateSemaphore(NULL,0,2,sub\_rate\_%<i>\_semaphore);
492. // Starting loop %<i> thread for sample time = %<s>s, offset = %<o>s.
493. CreateThread(NULL,0,sub\_rate%<i>,(**LPVOID**)NULL,0,NULL);


497. %endforeach
498. // Starting the nanosleep function
499. CreateThread(NULL,0,nanosleep,(**LPVOID**)NULL,0,NULL);
500. // Starting the base rate thread
501. CreateThread(NULL,0,base\_rate,(**LPVOID**)NULL,0,NULL);

504. **if** (!RealTimeThreadPriority())**return** -1;


508. %**if** extMode == 1
509. // External mode
510. WaitForSingleObject(ext\_mode\_ready,INFINITE);
511. **while**(rtmGetErrorStatus(%<modelName>\_M) == NULL && !rtmGetStopRequested(%<modelName>\_M)) {
512. rtExtModeOneStep(%<modelName>\_M->extModeInfo, NUMST, (boolean\_T \*)&rtmGetStopRequested(RT\_MDL));
513. Sleep(%<FEVAL("uint32", fundamentalStepSize \* 1000)>);
514. }
515. %endif
517. // Wait for threads to finish
518. %foreach i = numSampleTimes
519. %**if** i == 0 || i == 1 && tid01Eq
520. %**continue**
521. %endif

524. %endforeach
526. %**if** extMode == 1
527. rtExtModeShutdown(%<numSampleTimes>);
528. %endif
529. // Terminate model
530. %<modelName>\_terminate();
531. errStatus = rtmGetErrorStatus(%<modelName>\_M);
532. **if**(errStatus != NULL && strcmp(errStatus, "Simulation finished")) {
533. %%printf("%s\n", rtmGetErrorStatus(%<modelName>\_M));
534. **if**(!strcmp(errStatus, "Overrun")) {
535. printf("ISR overrun - sampling rate too fast\n");
536. }
537. **return**(1);
538. }
539. CancelWaitableTimer(timer);    // Stop timer
540. CloseHandle(timer);             // Delete handle
541. **return** 0;
542. }
543. // Local Variables:
544. // compile-command: "make -f %<modelName>.mk"
545. // End:
546. %endfunction
547. /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/