PROGRAMME OF THE EUROPEAN UNION

GALILEO EGNOS NAVIGATION MADE IN EUROPE

EUSPA European Union Agency for the Space Programme

GALILEO NTCM G SOFTWARE PACKAGE

USER GUIDE

Document History

Version	Date	Comment
1.0	May 2022	First Issue
1.1	April 2024	Typo in Table 1 corrected

Table of Contents

List of figures

List of tables

References

ID	Title
[1]	European GNSS (Galileo) Open Service - NTCM-G Ionospheric Model Description, Issue 1.0, European Commission (EC)
[2]	MISRA C:2012 Guidelines for the use of the C language in critical systems (3rd Edition), ISBN 978-1-906400-10-1 (paperback), ISBN 978-1-906400-11-8 (PDF), MIRA Limited, Nuneaton, March 2013.

1 Background and Purpose of the document

Galileo NTCM G is an empirical model that provides a practical and a cost-effective solution for the determination of global TEC, as described in [1]. *Galileo NTCM G* is an alternative algorithm to *Galileo NeQuick G* to compute ionospheric corrections for Galileo single-frequency users.

The reduced complexity and runtime of the *Galileo NTCM G* algorithm are considered beneficial, in particular in those user-segments where the user equipment has limited resources available. This is typically the case of the receivers used in civil aviation (i.e. avionics receivers), and location-based services (e.g. smartphones, smartwatches, UAV, IoT devices).

The purpose of this document is to describe the *Galileo NTCM G* Software Package (*Galileo NTCM-G SWP*), which complements the *Galileo NTCM G* model description provided in [1].

The *Galileo NTCM-G SWP* contains the reference implementation of the core algorithm and the auxiliary functions described in [1]. The software implementation of *Galileo NTCM G* has been developed by the European Commission's Joint Research Centre (JRC) and is available for download on the European GNSS Service Centre (GSC) website[1].

2 Overview of the Software Package

The *Galileo NTCM-G SWP* aims at providing a reference implementation of the *Galileo NTCM G* algorithm and the auxiliary functions described in [1]. The target is to provide a portable and validated C/C++ version of the *Galileo NTCM G* model that can be easily deployed in different GNSS applications and simulation environments. The development approach is based on modern industrial rapid prototyping practice of Model Based Design (MBD) using the Matlab/Simulink toolchain, and includes the deployment of the generated code into the target C/C++ application (i.e. desktop application). The MBD approach allows for enhanced code readability, improved performance (i.e. memory and execution efficiency) and compatibility with common coding standard, i.e. MISRA C/C++ Coding Guidelines in this case [2]. The implementation has been restricted as far as possible to the *Galileo NTCM G* core processing, with the main goal of maximizing the reusability and modularity of each sub-function. Actually, the users can refactor or integrate the code within their own SW solutions by adding control logics, error handling and/or external pre-processing steps.

The package contains three main blocks:

- The MBD repository, which contains all the results and auxiliary information relying on the *Galileo NTCM G* routine according to MBD.
- The Source code repository, which includes the reference implementation of the *Galileo NTCM G*.
- The Application repository, which includes example of applications build on top of the source code.

[1] European GNSS Service Centre | European GNSS Service Centre (https://gsc-europa.eu)

The rest of the document is devoted to describe the content of such repositories and their usage.

3 Model Based Design – *NTCM_mbd*

3.1. Content

The MBD repository ***./NTCM_mbd*** includes all the results of the model based design and of the prototyping process. In particular, the content is packaged as follow:

- ***./NTCM_mbd/matlab***: the baseline Matlab code used for the preliminary design and validation of the kernel algorithms
- ***./NTCM_mbd/mbd_models***: the Simulink implementation of the NTCM procedure used for simulation and code generation purposes
- ***./NTCM_mbd/mbd_test***: the test patterns, functions and harnesses used to perform the verification of the MBD implementation, including code validation (i.e. MILvsSIL test)
- ***./NTCM_mbd/mbd_cfg***: the configuration files necessary for the model and code settings
- ***./NTCM_mbd/mbd_script***: the scripts and macros for the simulation and test run
- ***./NTCM_mbd/resources***: project file resources to perform the setup of the MBD environment

3.2. Matlab

The baseline Matlab code is available under the folder <***./NTCM_mbd/matlab***>. This code has been used as the reference code for the development of the Simulink MBD solution and the following generation of the C/C++ source code.

The *Galileo NTCM G* core function *NTCM_G.m,* and the functions used to derive the auxiliary parameters, strictly follow the algorithm description given in [1].

The function *runNTCM.m* can be used to execute the Step-by-step procedure described in [1] and also reported in Table 6. The input <*inputData*> shall be a matrix which columns contain the input parameters as indicated in Table 1.

For each row of the input matrix, the *runNTCM.m* function computes the vertical TEC (vTEC), the slant TEC (sTEC), and the ionospheric delay for the input frequency <*carrFreq*>*.*

Column index	Parameter name	Description
Column 1	ai0	Effective Ionisation Level 1st order parameter [sfu]
Column 2	ai1	Effective Ionisation Level 2nd order parameter [sfu/deg]
Column 3	ai2	Effective Ionisation Level 3rd order parameter[sfu/deg]
Column 4	doy	Day of Year [dimensionless]
Column 5	utc	Universal time [hours]

Column index	Parameter name	Description
Column 6	rx_lon_deg	User receiver Geodetic longitude [deg]
Column 7	rx_lat_deg	User receiver Geodetic latitude [deg]
Column 8	rx_alt_m	User receiver Geodetic height [meters]
Column 9	sv_lon_deg	Satellite Geodetic longitude [deg]
Column 10	sv_lat_deg	Satellite Geodetic latitude [deg]
Column 11	sv_alt_m	Satellite Geodetic height Range [meters]

Table 1: Input matrix format

In order to test the *Galileo NTCM G* Matlab implementation, a test script *testNTCM_MATLAB.m* is available under the folder <***./NTCM_mbd/mbd_script*****>**. When executing the script, the user is asked to provide an input test vector and the carrier frequency. Exemplary test vectors in Matlab '.mat' format are located in the folder <***./NTCM_mbd/mbd_test/TestPattern*****>**.

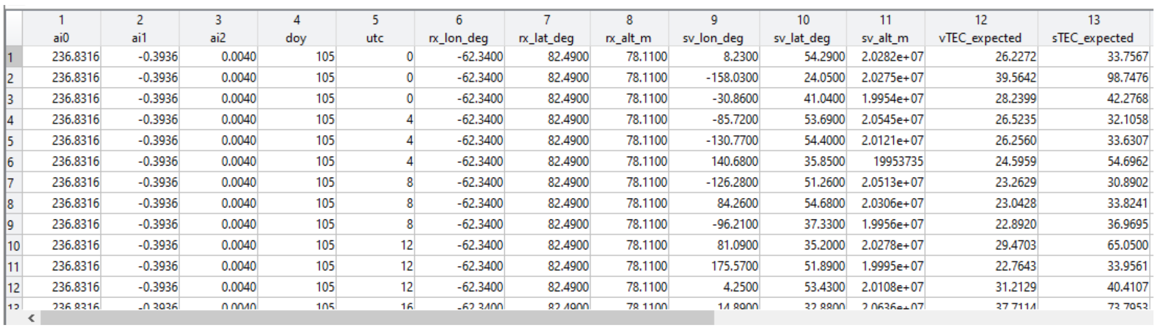
The following test vectors are available for selection:

Test vector name	Description
testVectors_RefDoc_Table_7_1_High_Solar.mat	Test vector for high solar activity as described in Section 7.1 of [1]
testVectors_RefDoc_Table_7_1_Medium_Solar.mat	Test vector for medium solar activity as described in Section 7.1 of [1]
testVectors_RefDoc_Table_7_1_Low_Solar.mat	Test vector for low solar activity as described in Section 7.1 of [1]
testVectors_Global_1_Year_High_Solar.mat	Global test vector containing equally distributed user locations for 12 days, one day at each first day of a month, and sampled at 2 hours. This test vector is for high solar activity.
testVectors_Global_1_Year_Medium_Solar.mat	Global test vector containing equally distributed user locations for 12 days, one day at each first day of a month, and sampled at 2 hours. This test vector is for medium solar activity.
testVectors_Global_1_Year_Low_Solar.mat	Global test vector containing equally distributed user locations for 12 days, one day at each first day of a month, and sampled at 2 hours. This test vector is for low solar activity.

Table 2: Matlab test vectors

The test vectors contain a Matlab Table Array data type composed of 13 columns. The first 11 columns are the one provided in Table 1, while the last two columns contain the expected values for vTEC (*vTEC_expected*) and sTEC (*sTEC_expected*).

The script passes the table to the function *runNTCM.m*, reads the vTEC and sTEC output and compares them with the expected values. The script also provides some statistics for the minimum, maximum, and mean values of vTEC and sTEC both for the absolute values and the obtained residuals.



	1	2	3	4	5	6	7	8	9	10	11	12	13
	ai0	ai1	ai2	doy	utc	rx_lon_deg	rx_lat_deg	rx_alt_m	sv_lon_deg	sv_lat_deg	sv_alt_m	vTEC_expected	sTEC_expected
1	236.8316	-0.3936	0.0040	105	0	-62.3400	82.4900	78.1100	8.2300	54.2900	2.0282e+07	26.2272	33.7567
2	236.8316	-0.3936	0.0040	105	0	-62.3400	82.4900	78.1100	-158.0300	24.0500	2.0275e+07	39.5642	98.7476
3	236.8316	-0.3936	0.0040	105	0	-62.3400	82.4900	78.1100	-30.8600	41.0400	1.9954e+07	28.2399	42.2768
4	236.8316	-0.3936	0.0040	105	4	-62.3400	82.4900	78.1100	-85.7200	53.6900	2.0545e+07	26.5235	32.1058
5	236.8316	-0.3936	0.0040	105	4	-62.3400	82.4900	78.1100	-130.7700	54.4000	2.0121e+07	26.2560	33.6307
6	236.8316	-0.3936	0.0040	105	4	-62.3400	82.4900	78.1100	140.6800	35.8500	19953735	24.5959	54.6962
7	236.8316	-0.3936	0.0040	105	8	-62.3400	82.4900	78.1100	-126.2800	51.2600	2.0513e+07	23.2629	30.8902
8	236.8316	-0.3936	0.0040	105	8	-62.3400	82.4900	78.1100	84.2600	54.6800	2.0306e+07	23.0428	33.8241
9	236.8316	-0.3936	0.0040	105	8	-62.3400	82.4900	78.1100	-96.2100	37.3300	1.9956e+07	22.8920	36.9695
10	236.8316	-0.3936	0.0040	105	12	-62.3400	82.4900	78.1100	81.0900	35.2000	2.0278e+07	29.4703	65.0500
11	236.8316	-0.3936	0.0040	105	12	-62.3400	82.4900	78.1100	175.5700	51.8900	1.9995e+07	22.7643	33.9561
12	236.8316	-0.3936	0.0040	105	12	-62.3400	82.4900	78.1100	4.2500	53.4300	2.0108e+07	31.2129	40.4107
13	236.8316	-0.3936	0.0040	105	16	-62.3400	82.4900	78.1100	14.8900	32.8800	2.0636e+07	37.7114	73.7953

Figure 1: Exemplary Matlab Table data type test vector

3.3. Simulink

3.3.1. Setup Project

In order to setup the MBD solution, the user needs to open Matlab/Simulink and run *NTCM_mbd.prj* located in the ./***NTCM_mbd*** root folder. It is also suggested to close/open the project before running the different scripts (hereafter described) each time a relevant change is applied and/or an error occurs in the model. At startup, the procedure runs all the cleaning operations (i.e. workspace, file, cache, slprj, etc…) in order to work from scratch (cleaning operations can be also executed by running *mbd_startup.m*).

3.3.2. Model description

The Simulink implementation provides a useful graphical representation of the NTCM functionalities allowing to easily recognize its subcomponents and processing modules. Such components are collected into the ***./NTCM_mbd/mbd_models*** folder. Each element corresponds, through the code generation, to a specific function or branch in the final source code. The interfaces and data exchange between the main relevant functionalities are hereafter described.

- The Reference Model is the *NTCM_Procedure.slx*. This corresponds to the entry point function of the whole *Galileo NTCM G* source code. It provides a representative example of how wrapping the core algorithm and the auxiliary functions.

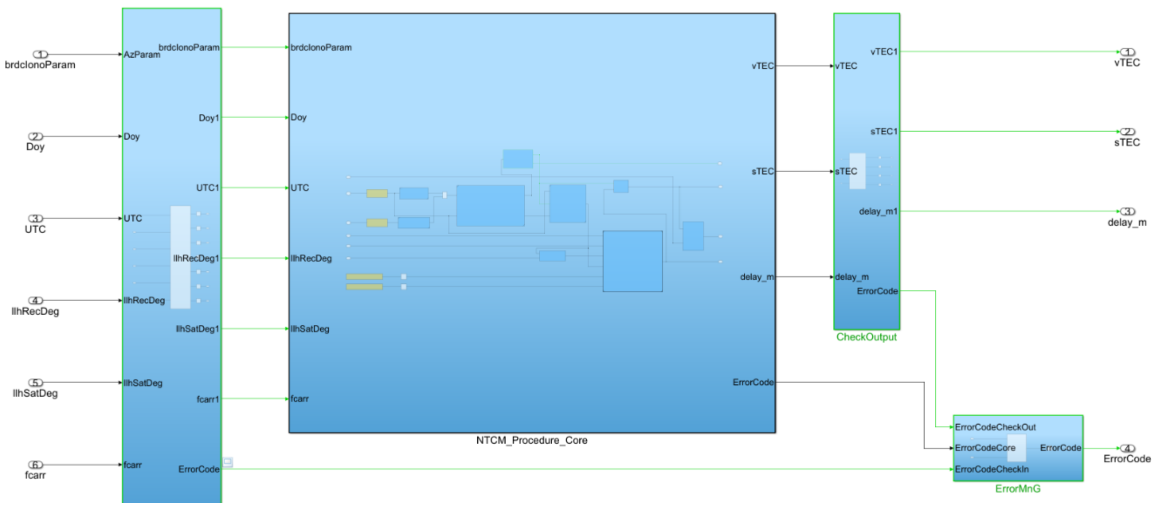


Figure 2: NTCM_Procedure.slx

In this baseline implementation, a minimal error handling of the core and I/O interface is implemented. It consists in a simple check of the variable ranges provided in Table 3 to Table 5, and forcing saturation in case the lower or upper range is exceeded.

Input	Description	lower	upper	Unit
a_{i0}	Effective Ionisation Level 1st order parameter	0	512	sfu
a_{i1}	Effective Ionisation Level 2nd order parameter	-4	4	sfu/deg
a_{i2}	Effective Ionisation Level 3rd order parameter	-0.25	0.25	sfu/deg
DoY	Day of Year (numerical value, 1st January = 1, 1st February=32, …)	1	366	dimensionless
UTC	Universal time	0	24	hours
φ_r	User receiver Geodetic latitude	-90	90	deg
λ_r	User receiver Geodetic longitude	-180	180	deg
h_r	User receiver Geodetic height	-4000	400000	meters
φ_S	Satellite Geodetic latitude	-90	90	deg
λ_S	Satellite Geodetic longitude	-180	180	deg
h_S	Satellite Geodetic height	450000	60000000	meters
f	Carrier Frequency	0	20000000	Hz

Table 3: Input parameters and ranges

Output	Description	lower	upper	Unit
vTEC	Vertical TEC	0	500	TECU
sTEC	Slant TEC	0	1500	TECU
delay	Ionospheric delay	0	10^12	meters

Table 4: Output parameters and ranges

Internal Test Points	Description	lower	upper	Unit
Elevation	Satellite elevation	0	90	deg

Table 5: Internal parameters and ranges

A green foreground in the figure will indicate hereafter model blocks that are not strictly part of the NTCM procedure in [1]. This means that the user can generally remove such functions in favour of execution time optimization or to integrate its own control strategies. Actually, such blocks can be disabled as variant subsystems by setting to zero (default is one) the Matlab workspace variable <CheckOn>.

- The *NTCM_Procedure_Core subsystem block* is the core function implementing the step-by-step procedure described in [1] and summarized in the following table.

1)	Obtain estimates of user receiver position (φ_r, λ_r, h_r) and satellite position (φ_s, λ_s, h_s)
2)	Obtain Effective Ionisation Level and broadcast coefficients (a_{i0}, a_{i1}, a_{i2})
3)	Obtain Universal Time (UT) and Day of the Year (DoY)
4)	Calculate satellite elevation (E) and azimuth (A) angles
5)	Calculate ionosphere pierce point location (φ_{ipp}, λ_{ipp}) for the user-to-satellite
6)	Call NTCM_G to calculate VTEC at pierce point location (φ_{ipp}, λ_{ipp}) and local time (LT)
7)	Calculate ionosphere Mapping Function (MF)
8)	Convert VTEC to STEC using the ionosphere mapping function.
9)	Convert STEC to time delay for the corresponding frequency

Table 6: Step-by-step procedure

The step-by-step procedure shall be repeated for each satellite-receiver line-of-sight and Galileo frequency.
Figure ***3*** and Figure ***4*** allow to identify the correspondence between the step-by-step procedure and the model blocks implementation.

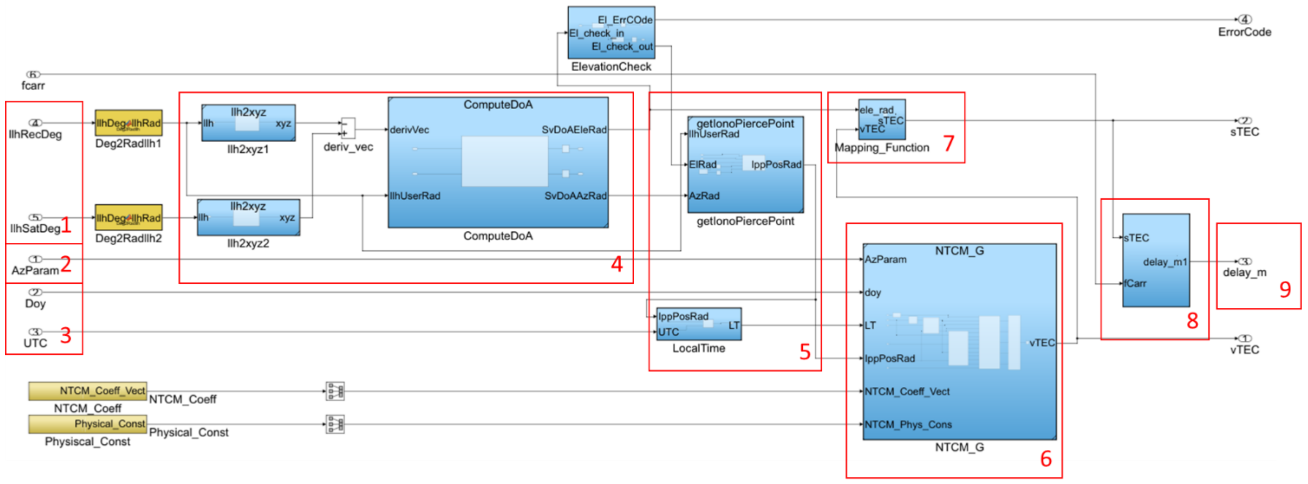


Figure 3: NTCM_Procedure_Core subsystem block

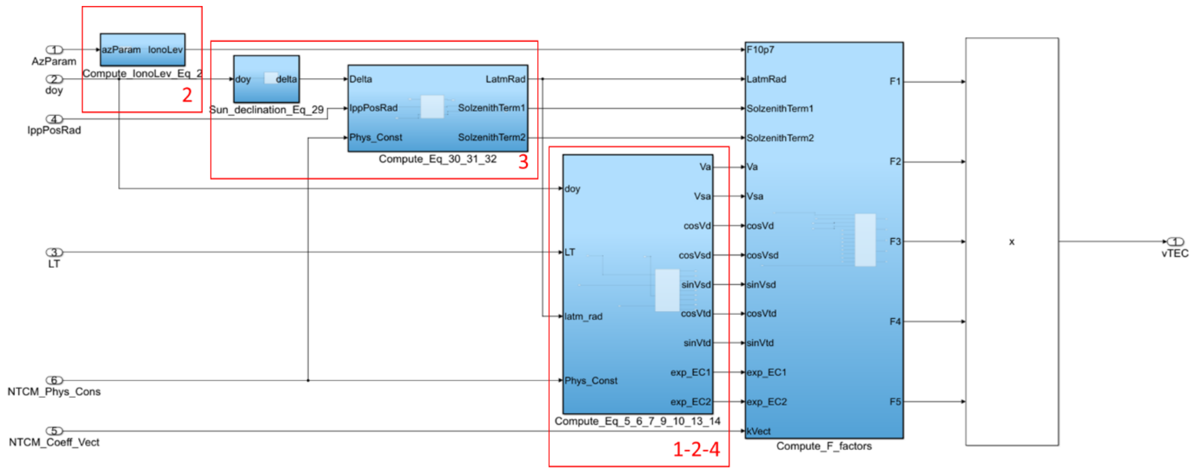
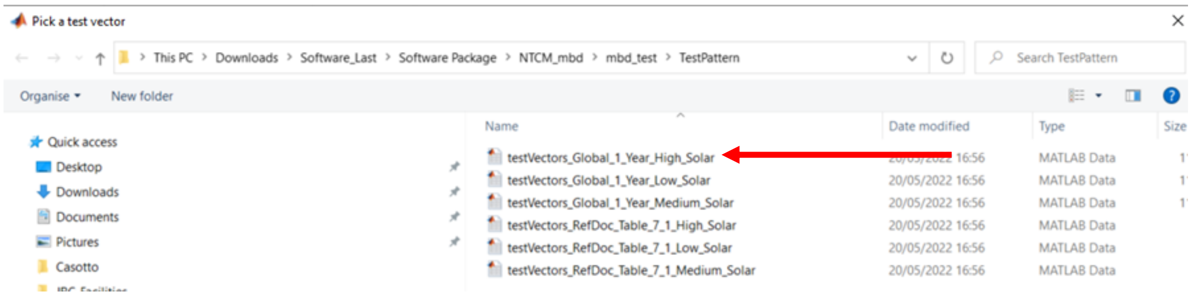


Figure 4: NTCM_G.slx

- ➢ The *NTCM_G.slx* implements the *Galileo NTCM G* model described in [1]. The approach considers five major dependencies of TEC, i.e., local time dependency (F_1), seasonal dependency (F_2), geomagnetic field dependency (F_3), equatorial anomaly dependency (F_4) and solar activity dependency (F_5). The dependencies are combined in a multiplicative way. As shown in Figure ***4***, the implementation groups the different equations described in [1] and encapsulate those sets in different sub-functions. The subsystems naming rule reflects this approach and the numbered red squares identify the link with the dependency factors (1-5).

3.3.3. Simulation and performance test

The user can run the script *PerformanceTest.m* in order to verify the model behaviour with respect to different scenarios. Several datasets are available for selection, as described in Table 2 and depicted in Figure 5.


Pick a test vector

This PC > Downloads > Software_Last > Software Package > NTCM_mbd > mbd_test > TestPattern

Search TestPattern

Organise ▾ New folder

Quick access
Desktop
Downloads
Documents
Pictures
Casotto

Name	Date modified	Type	Size
testVectors_Global_1_Year_High_Solar	16:56	MATLAB Data	1
testVectors_Global_1_Year_Low_Solar	20/05/2022 16:56	MATLAB Data	1
testVectors_Global_1_Year_Medium_Solar	20/05/2022 16:56	MATLAB Data	1
testVectors_RefDoc_Table_7_1_High_Solar	20/05/2022 16:56	MATLAB Data	
testVectors_RefDoc_Table_7_1_Low_Solar	20/05/2022 16:56	MATLAB Data	
testVectors_RefDoc_Table_7_1_Medium_Solar	20/05/2022 16:56	MATLAB Data	

Figure 5: Selection of the test pattern

After selection, the script will open the performance test harness, i.e. the top model that refer the target NTCM_Procedure.slx.

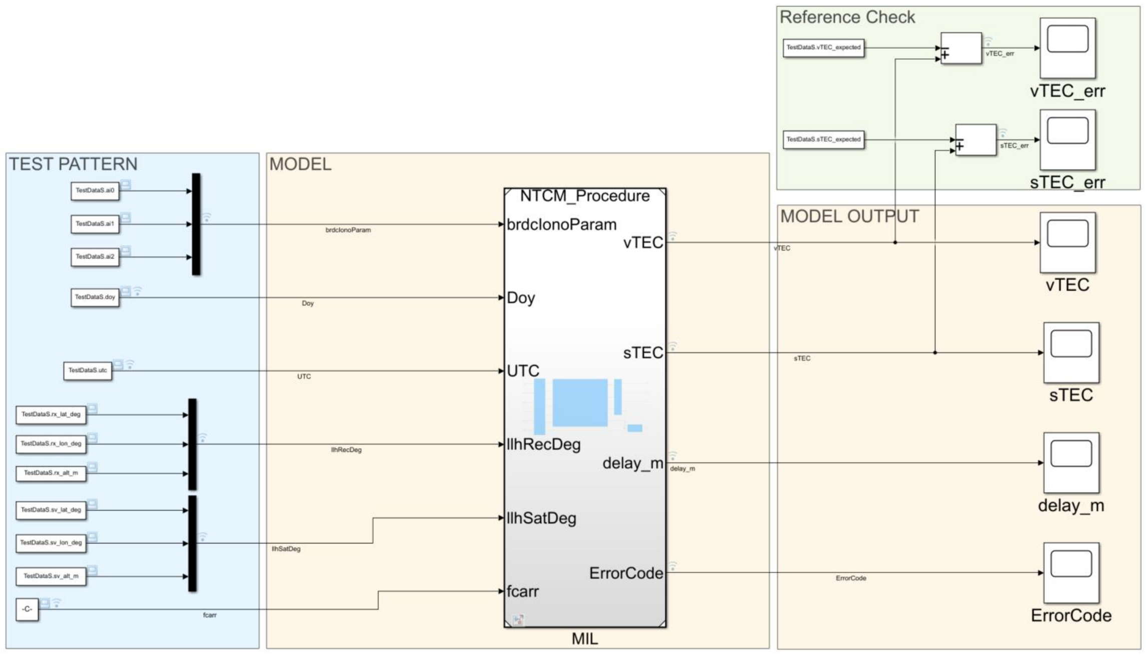


Figure 6: Performance Test Harness

This harness can be run as standard Simulink model allowing:

- to check the NTCM output for the selected input test pattern
- to check the model output with respect to the reference test vector (Reference Check in Figure 6)

All simulation data are saved in *NTCM_OUTPUT.mat*. It is worth noting that the default carrier frequency is Galileo E1 (1575.42 MHz) and the resulting ionospheric delay is given in meters. The user can compute the delay for a different frequency by changing the <*GNSScarrierFrequency*> workspace variable value.

3.3.4. Code generation and MILvsSIL

The user can run the script *MiLvsSiLTest.m* in order to experience code generation, building of the generated code, and performing the MiL(Model in the Loop) vs SiL (Software in the Loop) tests. Actually, the procedure can use as reference test pattern the same datasets considered for the performance test and listed in Table 2. After test pattern selection, the script will open the MiLvsSiL test harness.

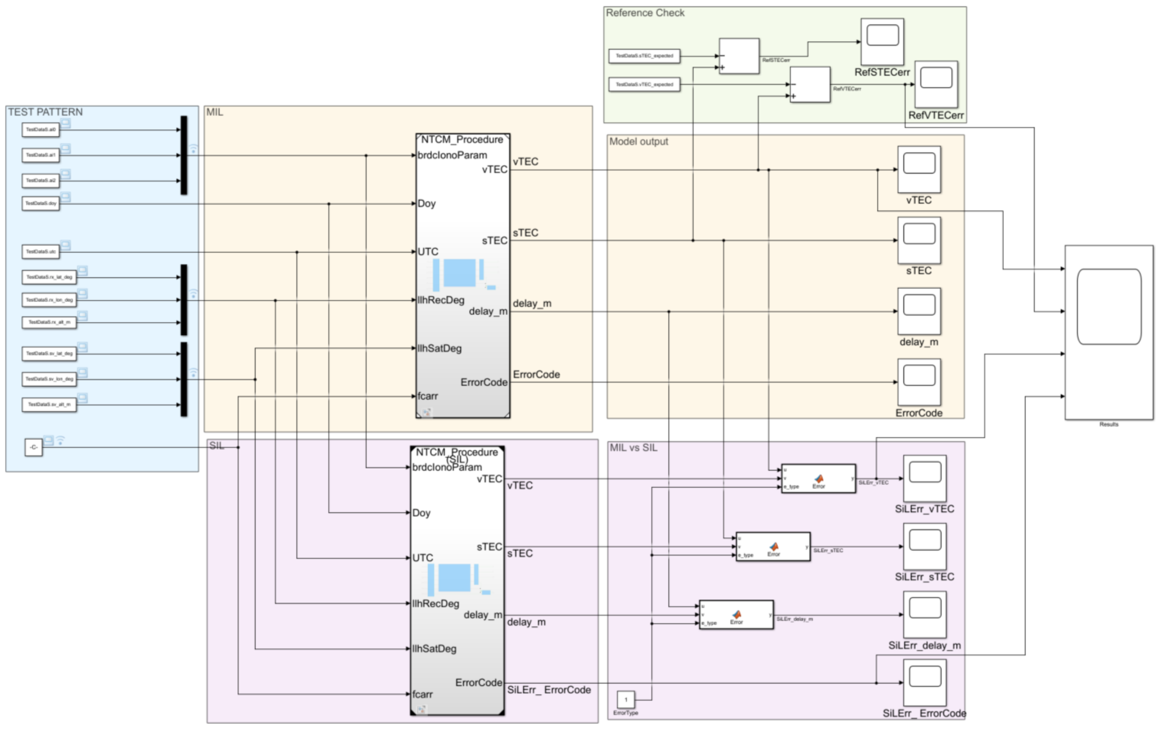


Figure 7: MiL vs SiL Test Harness

In this case, the top model refers to the target *NTCM_Procedure.slx* and to its replica set in SiL mode. In this configuration, the direct run of the Simulink model allows to:

- Generate C/C++ source code according to configuration parameter and template setting included in ./mbd_cfg
- Wrap, compile and build the generated source code within the SiL system
- Directly compare the output obtained by using the generated code with the ones generated by the reference *Galileo NTCM G* performance model (MiL system in Figure 7)

All products of the code generation are available and collected in the ***./NTCM_mbd/slprj*** folder, which is created at the time of compilation. In particular, the target source code can be found in the folder ***./NTCM_mbd/slprj/ert*** .

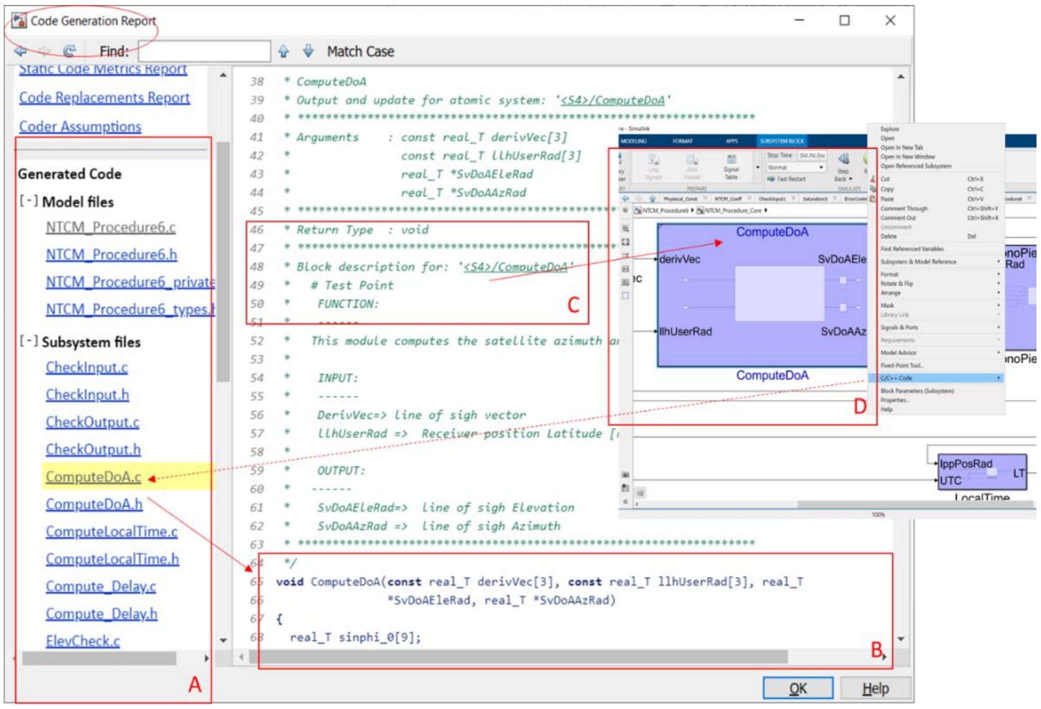
- ***Note-1****: MiLvsSiL need an Embedded Coder licence for code generation. If this capability is not available code report considered in the next section will not be available and the user can directly refer to the source code described in Sec.4.*
- ***Note-2*** *: For the sake of simplicity the MiLvsSiL does not include run time code advisor checks. The checked code, which is one that feed the final source package, can be obtained running the script CodegenTest.m and carrying out the same steps considered for the MiLvsSiL.m procedure.*
- ***Note-3****: It is worthy to remind that the SiL/PiL (.xil) files that are integral part of the code generation and SiL testing are not part of the final source code package. This test stabs (instrumentation probes) can just be neglected, since they shall be replaced at integration level by the wrapper functions requested for the deployment into the user target application.*

3.3.5. Model-Code Navigation and Report

After running the MiLvsSiL test harness, the user can access the MBD reports. In particular, at the end of the simulation two reports are automatically created:

a) The *code generation report* (see Figure 8) that provides:
   - capability to navigate the generated source code by simply clicking on the name of the function (A to B in Figure 8)
   - capability to identify the module of the Simulink model that corresponds to the source code function or branch by using the block element hyperlink (C-D in Figure 8)
   - the bidirectional traceability of the Model to the Code, allowing to go from the model block back to the code (right click on the model block, select Navigate C/C++ section)

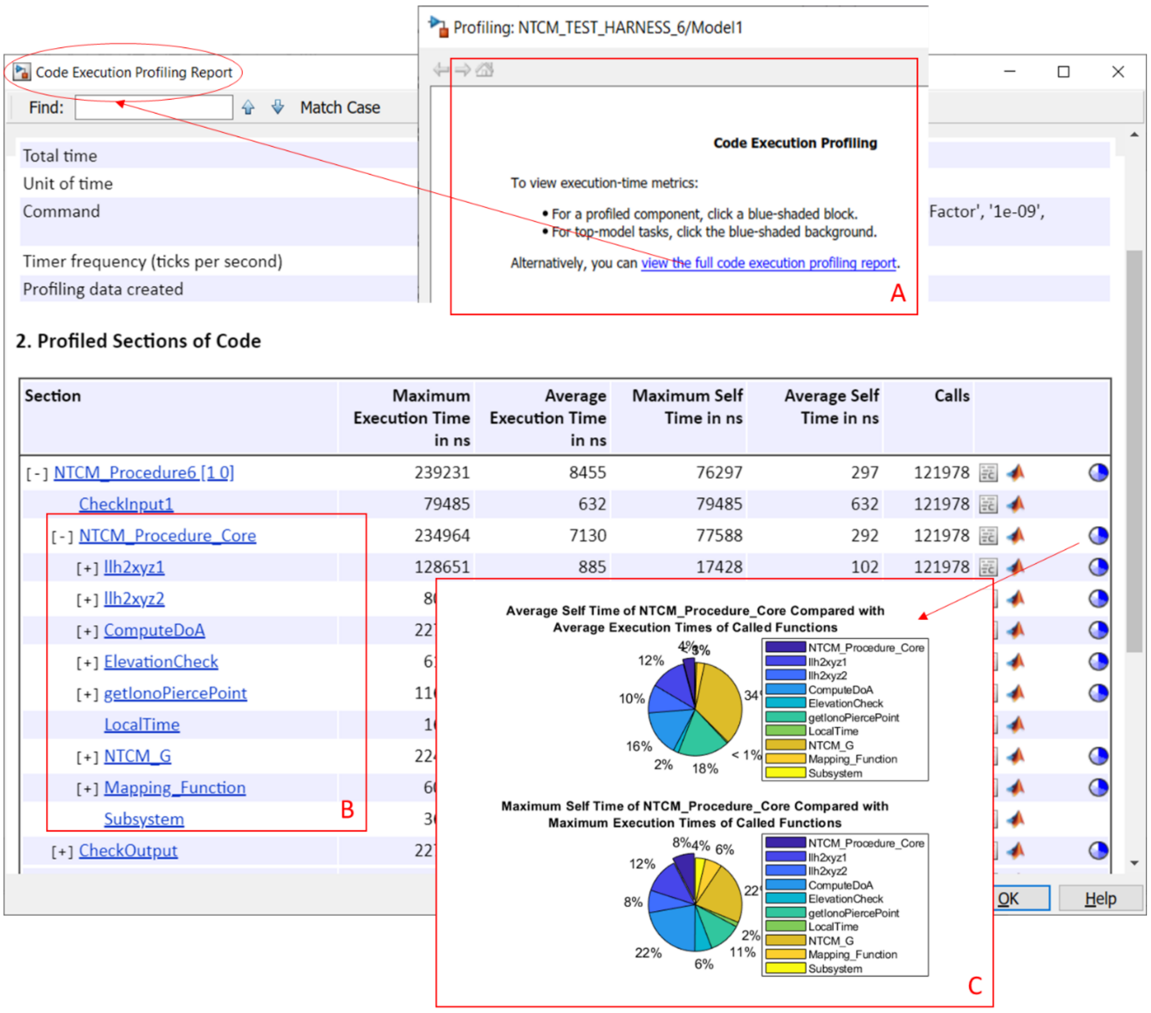
b) The *code execution profiling* report which provides metrics based on data collected from a SIL execution: execution times and CPU Utilization are calculated from data recorded by instrumentation probes added to the SIL test harness or inside the code generated for each component. To access the code execution profiling, the user can click the hyperlink in the window available after simulation (Figure 9). Just as for the code generation report, the user can navigate the profiler through the function tree and learn how each sub-function contributes to the whole computational load.



```
 * ComputeDoA
 * Output and update for atomic system: '<S4>/ComputeDoA'
 * ******************************************************************
 * Arguments    : const real_T derivVec[3]
 *                const real_T llhUserRad[3]
 *                real_T *SvDoAEleRad
 *                real_T *SvDoAAzRad
 * ****************************************
 * Return Type  : void
 * ****************************************
 * Block description for: '<S4>/ComputeDoA'
 *   # Test Point
 *    FUNCTION:
 *    ------
 *   This module computes the satellite azimuth a
 *
 *    INPUT:
 *    ------
 *    DerivVec=> line of sigh vector
 *    llhUserRad =>  Receiver position Latitude [
 *
 *    OUTPUT:
 *   ------
 *    SvDoAEleRad=>  line of sigh Elevation
 *    SvDoAAzRad =>  line of sigh Azimuth
 * ******************************************************************
 */
void ComputeDoA(const real_T derivVec[3], const real_T llhUserRad[3], real_T
                *SvDoAEleRad, real_T *SvDoAAzRad)
{
  real_T sinphi_0[9];
```

Figure 8 : Code Generation Report

The code generation report can be used for code navigation. In fact, it generates an on-line documentation browser (.html) equivalent to other code reporting solutions (e.g. *doxigen*). Actually, the code settings make the source code compatible with *doxigen* reporting format (see Sec.4).


Code Execution Profiling Report
Find:
Match Case
Profiling: NTCM_TEST_HARNESS_6/Model1
A
B
C
OK
Help
Average Self Time of NTCM_Procedure_Core Compared with Average Execution Times of Called Functions
4%
< 3%
12%
10%
16%
2%
18%
< 1%
34
Maximum Self Time of NTCM_Procedure_Core Compared with Maximum Execution Times of Called Functions
8%
4%
6%
12%
8%
22%
6%
11%
2%
22

Total time

Unit of time

Command

Factor', '1e-09',

Timer frequency (ticks per second)

Profiling data created

Code Execution Profiling

To view execution-time metrics:

- For a profiled component, click a blue-shaded block.
- For top-model tasks, click the blue-shaded background.

Alternatively, you can view the full code execution profiling report.

2. Profiled Sections of Code

Section	Maximum Execution Time in ns	Average Execution Time in ns	Maximum Self Time in ns	Average Self Time in ns	Calls
[-] NTCM_Procedure6 [1 0]	239231	8455	76297	297	121978
CheckInput1	79485	632	79485	632	121978
[-] NTCM_Procedure_Core	234964	7130	77588	292	121978
[+] llh2xyz1	128651	885	17428	102	121978
[+] llh2xyz2	8				
[+] ComputeDoA	22				
[+] ElevationCheck	6				
[+] getIonoPiercePoint	11				
LocalTime	1				
[+] NTCM_G	22				
[+] Mapping_Function	6				
Subsystem	3				
[+] CheckOutput	22				

■	NTCM_Procedure_Core
■	llh2xyz1
■	llh2xyz2
■	ComputeDoA
■	ElevationCheck
■	getIonoPiercePoint
■	LocalTime
■	NTCM_G
■	Mapping_Function
■	Subsystem

Figure 9 : Code Generation Report

4 Source Code – *NTCM_src* (C/C++)

4.1. Content

The Source Code repository ***./NTCM_src*** includes the *Galileo NTCM G* C/C++ source code and some additional files allowing to document, deploy and test it within a desktop application. In particular, the content is packaged as follow:

- ***./NTCM_src/src***: the C/C++ code corresponding to the baseline *Galileo NTCM G* implementation and including :
  - ***./NTCM_src /src/lib:*** collecting the procedure core functions
  - ***./NTCM_src/src/aux_fun:*** collecting all the external functions (i.e. file read and write) requested to feed the main function
  - ***./NTCM_src/src (root):*** including the main function *run_NTCM_main.c* and the *ConfigRun.h* used to configure the target code execution
- ***./NTCM_src/html***: containing the *doxigen* documentation files

4.2. User Guidelines

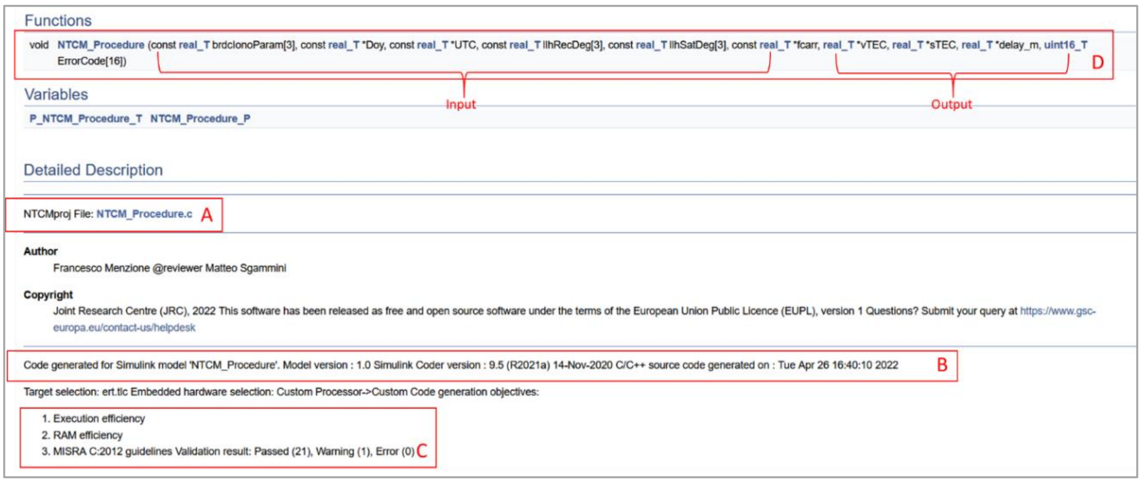
4.2.1. Code architecture and design drivers

The code can be navigated using the *doxigen* documentation by using the shortcut *Start Doxygen* shortcut available in the SW package root folder*.*

The SW architecture of the library ***./src/NTCM_lib*** is based on two following modules :

1) *NTCM_Procedure*, whose entry point is the function *NTCM_procedure.c*, which calls the other functions, encapsulated in standalone .c files to increase the modularity
2) the *NTCM_G* function, which embeds all the mathematical sub-functions

As shown in the figure below, all .c files (A) includes the code versioning (B) and the auto-coding objectives (C). The blocks and configuration parameter setting used in the *mbd* process generate all the functions following the *reusable function paradigm (*D). The input variables are passed per argument, while the output require pointers (or direct return variable). In this manner, the users can setup multiple function call in the final integration code.



```
void NTCM_Procedure (const real_T brdcIonoParam[3], const real_T *Doy, const real_T *UTC, const real_T llhRecDeg[3], const real_T llhSatDeg[3], const real_T *fcarr, real_T *vTEC, real_T *sTEC, real_T *delay_m, uint16_T
    ErrorCode[16])
```

P_NTCM_Procedure_T NTCM_Procedure_P

NTCMproj File: NTCM_Procedure.c

Author
Francesco Menzione @reviewer Matteo Sgammini

Copyright
Joint Research Centre (JRC), 2022 This software has been released as free and open source software under the terms of the European Union Public Licence (EUPL), version 1 Questions? Submit your query at https://www.gsc-europa.eu/contact-us/helpdesk

Code generated for Simulink model 'NTCM_Procedure'. Model version : 1.0 Simulink Coder version : 9.5 (R2021a) 14-Nov-2020 C/C++ source code generated on : Tue Apr 26 16:40:10 2022

Target selection: ert.tlc Embedded hardware selection: Custom Processor->Custom Code generation objectives:

1. Execution efficiency
2. RAM efficiency
3. MISRA C:2012 guidelines Validation result: Passed (21), Warning (1), Error (0)

Figure 10: Source Code abstract

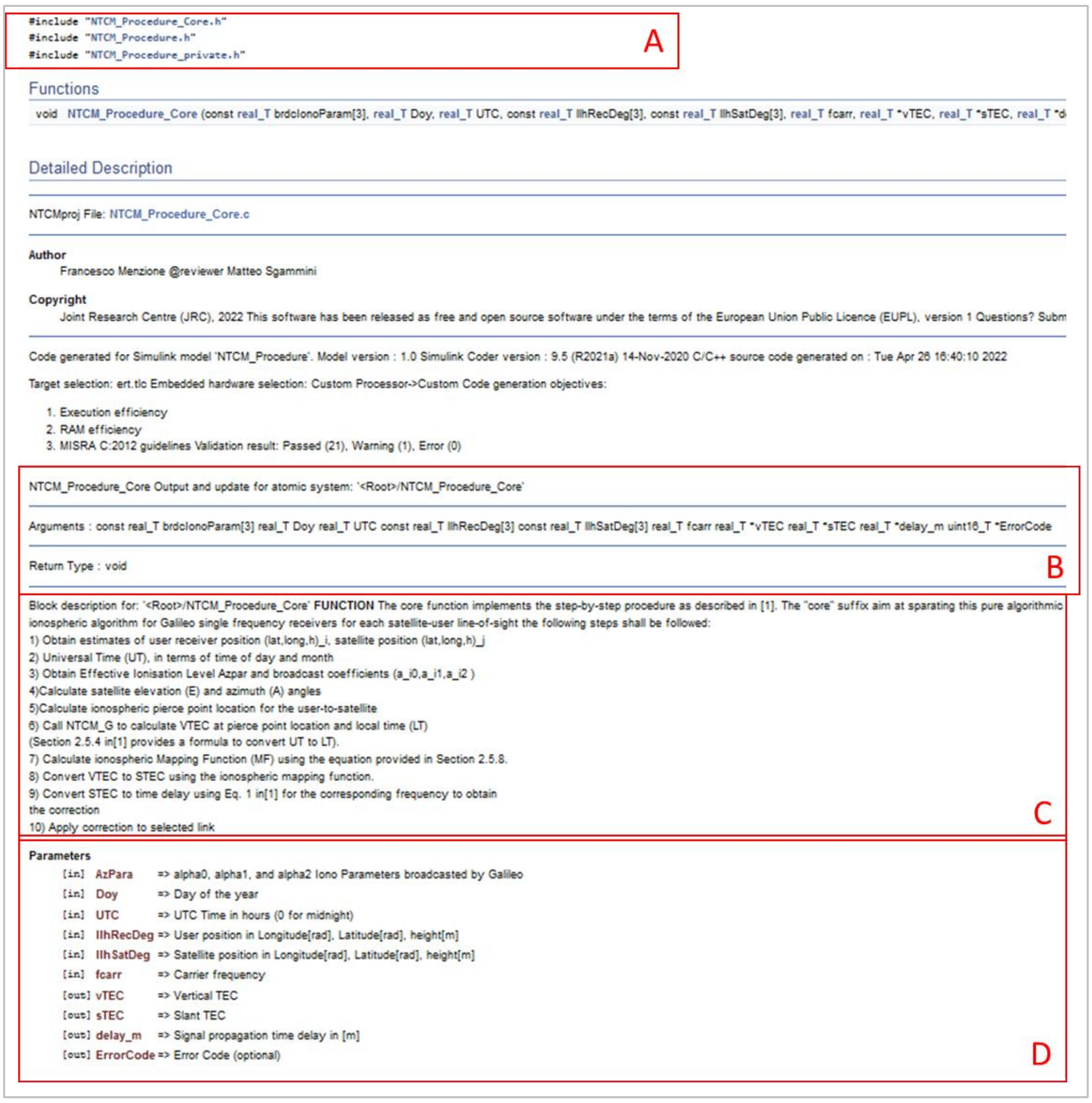
The code does not use structured data. Furthermore, some optimizations were turned off in favour of readability.

Function parameters, i.e.physical constants and *Galileo NTCM G* coefficient (as well as error codes), have been implemented as *define preprocessor variables* since they are not expected to change at procedure level. Nevertheless, it is worth to remind that both physical parameter and *Galileo NTCM G* coefficient vectors are passed to NTCM_G.c function as arguments to eventually perform sensitivity analysis on the model or time-varying approaches.

4.2.2. Functions documentation

The code comments are generated according to a custom template compliant with *doxigen* format and provide the following information for each function:

- Function dependencies (A)
- Function prototype interface, i.e. arguments and return type (B)
- Function description (C)
- Function input/output parameter (D)


```
#include "NTCM_Procedure_Core.h"
#include "NTCM_Procedure.h"
#include "NTCM_Procedure_private.h"
```

Functions

void NTCM_Procedure_Core (const real_T brdcIonoParam[3], real_T Doy, real_T UTC, const real_T llhRecDeg[3], const real_T llhSatDeg[3], real_T fcarr, real_T *vTEC, real_T *sTEC, real_T *d

Detailed Description

NTCMproj File: NTCM_Procedure_Core.c

Author
Francesco Menzione @reviewer Matteo Sgammini

Copyright
Joint Research Centre (JRC), 2022 This software has been released as free and open source software under the terms of the European Union Public Licence (EUPL), version 1 Questions? Subm

Code generated for Simulink model 'NTCM_Procedure'. Model version : 1.0 Simulink Coder version : 9.5 (R2021a) 14-Nov-2020 C/C++ source code generated on : Tue Apr 26 16:40:10 2022

Target selection: ert.tlc Embedded hardware selection: Custom Processor->Custom Code generation objectives:

1. Execution efficiency
2. RAM efficiency
3. MISRA C:2012 guidelines Validation result: Passed (21), Warning (1), Error (0)

NTCM_Procedure_Core Output and update for atomic system: '<Root>/NTCM_Procedure_Core'

Arguments : const real_T brdcIonoParam[3] real_T Doy real_T UTC const real_T llhRecDeg[3] const real_T llhSatDeg[3] real_T fcarr real_T *vTEC real_T *sTEC real_T *delay_m uint16_T *ErrorCode

Return Type : void

Block description for: '<Root>/NTCM_Procedure_Core' **FUNCTION** The core function implements the step-by-step procedure as described in [1]. The "core" suffix aim at sparating this pure algorithmic ionospheric algorithm for Galileo single frequency receivers for each satellite-user line-of-sight the following steps shall be followed:
1) Obtain estimates of user receiver position (lat,long,h)_i, satellite position (lat,long,h)_j
2) Universal Time (UT), in terms of time of day and month
3) Obtain Effective Ionisation Level Azpar and broadcast coefficients (a_i0,a_i1,a_i2)
4)Calculate satellite elevation (E) and azimuth (A) angles
5)Calculate ionospheric pierce point location for the user-to-satellite
6) Call NTCM_G to calculate VTEC at pierce point location and local time (LT)
(Section 2.5.4 in[1] provides a formula to convert UT to LT).
7) Calculate ionospheric Mapping Function (MF) using the equation provided in Section 2.5.8.
8) Convert VTEC to STEC using the ionospheric mapping function.
9) Convert STEC to time delay using Eq. 1 in[1] for the corresponding frequency to obtain
the correction
10) Apply correction to selected link

Parameters

[in]	AzPara	=> alpha0, alpha1, and alpha2 Iono Parameters broadcasted by Galileo
[in]	Doy	=> Day of the year
[in]	UTC	=> UTC Time in hours (0 for midnight)
[in]	llhRecDeg	=> User position in Longitude[rad], Latitude[rad], height[m]
[in]	llhSatDeg	=> Satellite position in Longitude[rad], Latitude[rad], height[m]
[in]	fcarr	=> Carrier frequency
[out]	vTEC	=> Vertical TEC
[out]	sTEC	=> Slant TEC
[out]	delay_m	=> Signal propagation time delay in [m]
[out]	ErrorCode	=> Error Code (optional)

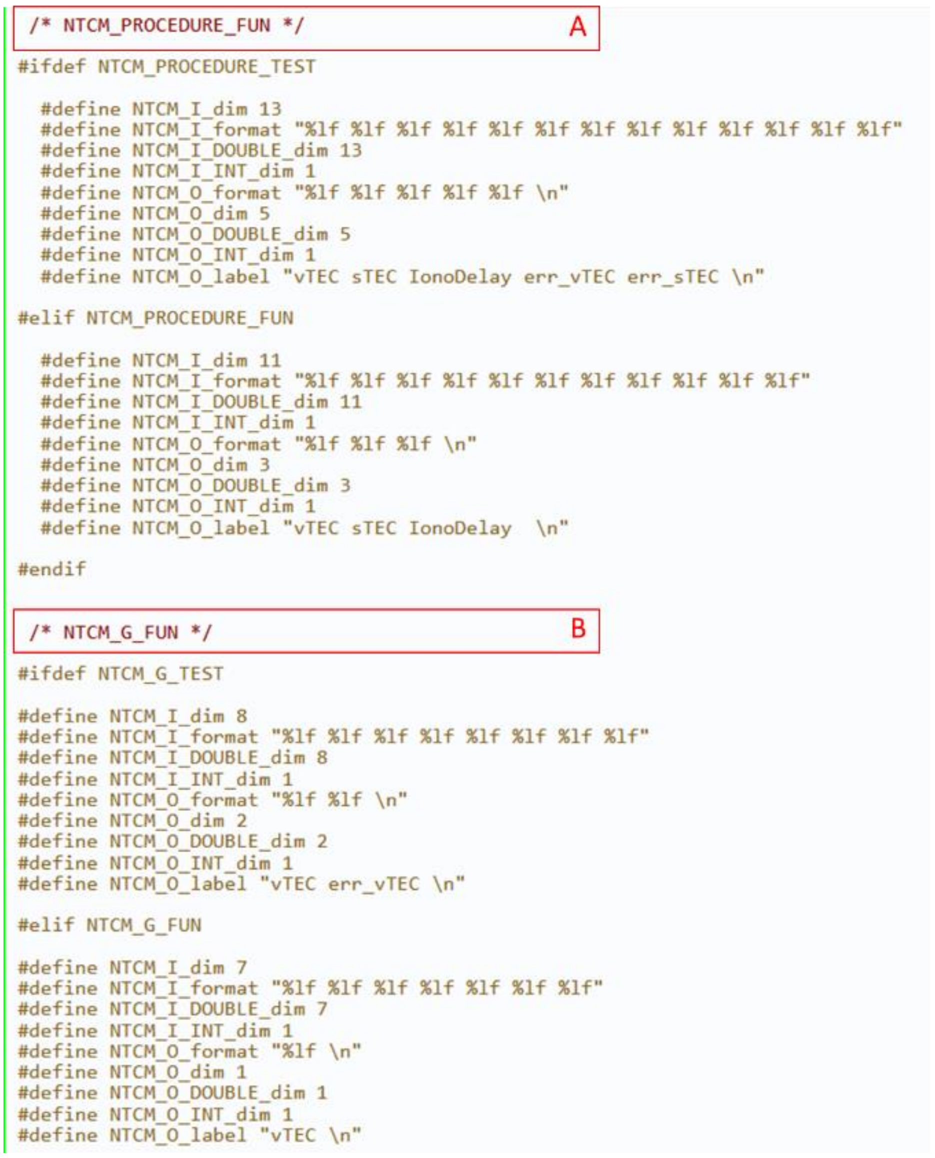
Figure 11: Function Documentation

4.2.3. *Galileo NTCM G* baseline implementation

The baseline *run_NTCM_main.c* allows to build the *Galileo NTCM G* baseline application and, as a result:

1) to read input test patterns from .txt file and run NTCM procedure (*_fun*)
2) to replicate, at source code level, the MiLvsSiL test described in Section 3.3.4 by comparing the output with the reference test vectors (*_test*)
3) to perform *_fun* and *_test* considering only the NTCM_G submodule

The driver program supports different configurations by setting accordingly the pre-compilation flags before compiling (see Figure 12).



```
/* NTCM_PROCEDURE_FUN */
#ifdef NTCM_PROCEDURE_TEST

  #define NTCM_I_dim 13
  #define NTCM_I_format "%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf"
  #define NTCM_I_DOUBLE_dim 13
  #define NTCM_I_INT_dim 1
  #define NTCM_O_format "%lf %lf %lf %lf %lf \n"
  #define NTCM_O_dim 5
  #define NTCM_O_DOUBLE_dim 5
  #define NTCM_O_INT_dim 1
  #define NTCM_O_label "vTEC sTEC IonoDelay err_vTEC err_sTEC \n"

#elif NTCM_PROCEDURE_FUN

  #define NTCM_I_dim 11
  #define NTCM_I_format "%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf"
  #define NTCM_I_DOUBLE_dim 11
  #define NTCM_I_INT_dim 1
  #define NTCM_O_format "%lf %lf %lf \n"
  #define NTCM_O_dim 3
  #define NTCM_O_DOUBLE_dim 3
  #define NTCM_O_INT_dim 1
  #define NTCM_O_label "vTEC sTEC IonoDelay  \n"

#endif

/* NTCM_G_FUN */
#ifdef NTCM_G_TEST

#define NTCM_I_dim 8
#define NTCM_I_format "%lf %lf %lf %lf %lf %lf %lf %lf"
#define NTCM_I_DOUBLE_dim 8
#define NTCM_I_INT_dim 1
#define NTCM_O_format "%lf %lf \n"
#define NTCM_O_dim 2
#define NTCM_O_DOUBLE_dim 2
#define NTCM_O_INT_dim 1
#define NTCM_O_label "vTEC err_vTEC \n"

#elif NTCM_G_FUN

#define NTCM_I_dim 7
#define NTCM_I_format "%lf %lf %lf %lf %lf %lf %lf"
#define NTCM_I_DOUBLE_dim 7
#define NTCM_I_INT_dim 1
#define NTCM_O_format "%lf \n"
#define NTCM_O_dim 1
#define NTCM_O_DOUBLE_dim 1
#define NTCM_O_INT_dim 1
#define NTCM_O_label "vTEC \n"
```

Figure 12: run_NTCM_main.c configuration

5 Application – *NTCM_app*

5.1. Content

The Application repository ***./NTCM_app*** includes the *Galileo NTCM G* target application, build files and scripts allowing to run the NTCM functions from shell command. In particular, the content is packaged as follow:

- ***./NTCM_app/bin***: the executable and object product of the compilation (.exe, .out, .o)
- ***./NTCM_app/build*** : VS projects used for the build
- ***./NTCM_app/scripts*** : all shell scripts wrapping the driver program for test

The following build toolchains are supported:

- Microsoft C++ Compiler (Microsoft Visual Studio 2022)
- GNU compiler gcc (GCC for Windows Subsystem MVS via WSL distribution)

For this reason, all ***NTCM_app*** files are encapsulated within two different sub-folders ***./mvs*** and ***./gcc*** according to the target OS.

5.2. User guidelines

5.2.1. Microsoft Visual Studio Build with MVS and GCC

A unified approach for the driver program has been selected by using different MVS solutions collected in:

- ***NTCM_app/build/mvs/NTCM_mvs*** ==> Microsoft C++ Compiler
- ***NTCM_app/build/gcc/NTCM_lx*** ==> GCC (via WSL) \

The user needs Microsoft Visual Studio 2022 to work with such solutions, which include different projects. The different projects share the same NTCM source code, available in *NTCM_src*, but they differ for:

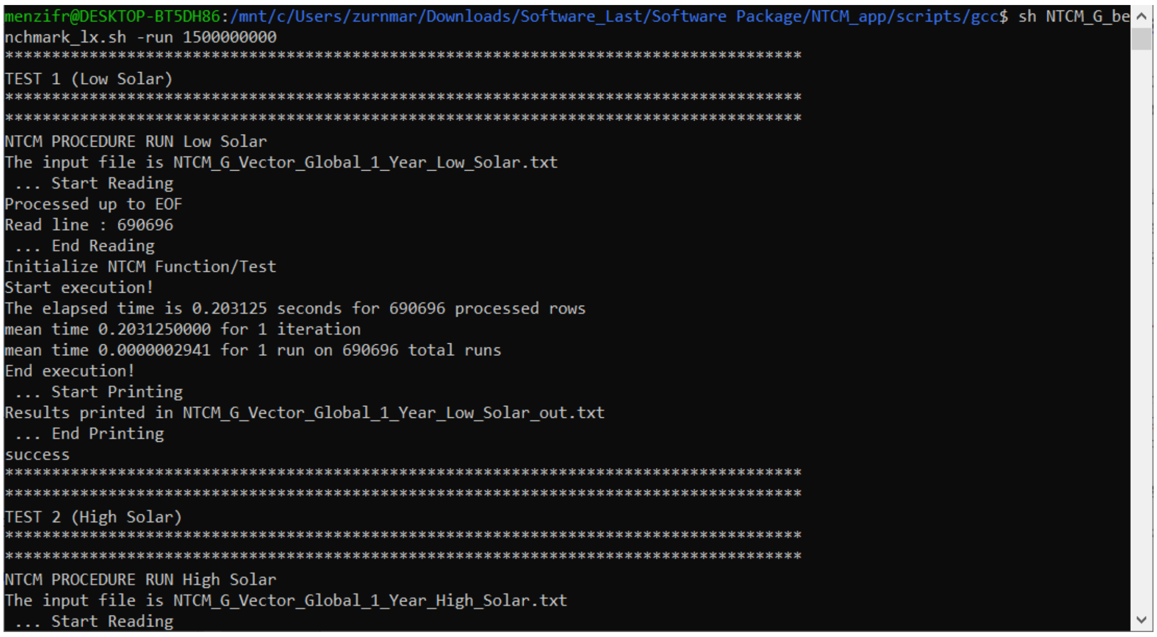
- Pre-processor options corresponding to the different testing and/or debugging objectives (see pre-processor definition in the *solution properties*)
- input passed to the driver program (see command arguments in *solution properties*)

All Input/Output files are included in the ***./NTCM_app/script/Data*** and ***./NTCM_app/script/Out.***

- ***Note-1*** *: NTCM_lx solution needs WSL distribution with UBUNTU 18 LTS. If not available the user can change the project configuration properties and point to a different client linux machine*
- ***Note-2*** *: NTCM_mvs directly refer to NTCM_app/script Data/Out content for debugging and testing purposes*
- ***Note-3*** *: NTCM_lx need to copy Data/Out content in the target folder for debugging and testing of the solution*
- ***Note-4*** *: debug/release and x64, x86 configuration can be controlled by changing configuration manager of the solution*

5.2.2. Run the target applications

The driver programs (**./bin**) are provided for exemplificative purposes and give evidence of the possibility to use the source code on a target application. The user can run them by using shell scripts included in ***./scripts***.



```
menzifr@DESKTOP-BT5DH86:/mnt/c/Users/zurnmar/Downloads/Software_Last/Software Package/NTCM_app/scripts/gcc$ sh NTCM_G_be
nchmark_lx.sh -run 1500000000
*************************************************************************************
TEST 1 (Low Solar)
*************************************************************************************
*************************************************************************************
NTCM PROCEDURE RUN Low Solar
The input file is NTCM_G_Vector_Global_1_Year_Low_Solar.txt
 ... Start Reading
Processed up to EOF
Read line : 690696
 ... End Reading
Initialize NTCM Function/Test
Start execution!
The elapsed time is 0.203125 seconds for 690696 processed rows
mean time 0.2031250000 for 1 iteration
mean time 0.0000002941 for 1 run on 690696 total runs
End execution!
 ... Start Printing
Results printed in NTCM_G_Vector_Global_1_Year_Low_Solar_out.txt
 ... End Printing
success
*************************************************************************************
*************************************************************************************
TEST 2 (High Solar)
*************************************************************************************
*************************************************************************************
NTCM PROCEDURE RUN High Solar
The input file is NTCM_G_Vector_Global_1_Year_High_Solar.txt
 ... Start Reading
```

Figure 13: Shell script run

The current set of the available scripts is listed together with their setting options:

- **NTCM Procedure function**

Command :	./run_NTCM_procedure	
Options :	-h	command description for the NTCM procedure function
	-run	direct function run (fastest I/O without check)
	-check	output comparison with reference input vector (see -h)

- **NTCM Procedure benchmark**

Command :	./NTCM_procedure_benchmark	
Options :	-h	benchmark header
	-run	run with High, Medium and Low solar activity test vectors

- **NTCM_G Model**

Command : ./run_NTCM_G

Options : -h command description for the NTCM_G Model
-run direct function run (fastest I/O without check)
-check output comparison with reference input vector (see -h)

- **NTCM_G Model benchmark**

Command :	./NTCM_G_banchmark	
Options :	-h	benchmark header
	-run	run with High, Medium and Low solar activity test vectors

- ***Note-1****: run the script according to the target OS Windows/linux*
- ***Note-2****: currently the input of the main program is externally pre-allocated in static manner and maximum 700000 lines can be processed. Next version would overcome this limit if needed (no impact on the core code functions and not strictly necessary for training usage)*

European Union Agency for the Space Programme

LINKING SPACE TO USER NEEDS

www.euspa.europa.eu

@EU4Space

@EU4Space

EUSPA

@space4eu

EUSPA