5 Driving Requirements for a Mathematical Library

This chapter describes and identifies the requirements that drive the development of a mathematical library for the evaluation of elementary mathematical functions in S/C software systems.

5.1 Numerical Computations in Spacecraft Software and in Its Development

Software running on S/C systems requires numerical computations for the following range of tasks:

- for Guidance Navigation and Control (GNC) duties in AOCS systems,
- for S/C equipment/payload control systems, and
- for S/C instrument data processing (scientific computing).

During the development phases of S/C on-board software numerical computations are also required for:

- GNC algorithm design and development,
- S/C equipment simulation-model development, and
- S/C environment simulation-model development.

The computers that perform these numerical computations on S/Cs are often based on 32 bit Scalable Processor Architectures (SPARCs) (e.g., SPARC V7 for the ERC32 and SPARC V8 for the LEON processors such as the LEON2, LEON3 and LEON4) with Institute of Electrical and Electronics Engineers (IEEE) 754 compliant Floating Point Units (FPUs) (e.g., MEIKO and GRFPU). During the development of the S/C software, x86-64 architecture processors are usually used though. For S/C software, the numerical calculation shall never bring the processor or FPU to a halt because of an error.

GTD-TR-01-BL-0001 The Basic mathematical Library (BL) shall run on SPARC V8 and x86-64 processor architectures without halting the processor.

These hardware environments, together with the wide use of the International Organization for Standardization (ISO) C programming language for S/C software development, and the availability of a to ECSS) qualified Operating System (OS) such as Real-Time Executive for Multiprocessor Systems (RTEMS), make an ISO C software library for numerical computations, based on IEEE-754 arithmetic, an appropriate candidate to pre-qualify. This pre-qualification, in compliance with the ECSS standards and to industrial standards like Motor Industry Software Reliability Asociation (MISRA), would provide the S/C software developer community with an additional *building block* to rely on, when piecing together their specific software system.

GTD-TR-01-BL-0002 The BL shall be written in ISO/IEC 9899.

GTD-TR-01-BL-0003 The BL shall be compatible with the RTEMS OS.

GTD-TR-01-BL-0004 The BL shall be IEEE 754-2008 compliant.

GTD-TR-01-BL-0005 The BL shall be ECSS category B compliant.

GTD-TR-01-BL-0006 The BL shall be MISRA-C:2012 compliant.

5.2 Development Process of On-Board Software

On-board software development and validation activities undergo several phases where different platforms are used (see [Eic09], Table 3.2):

- first AOCS algorithm development in Matlab and the testing of these algorithms on an *algorithms in the loop* Functional Validation Bench (FVB)/Functional Engineering Simulator (FES) (shown in Figure 5.1),
- validation of the on-board software, implemented in C, on a *software in the loop* Software Validation Facility (SVF) (shown in Figure 5.2), and
- hybrid validations of the on-board software with *controller in the loop* and *hardware in the loop* approaches.

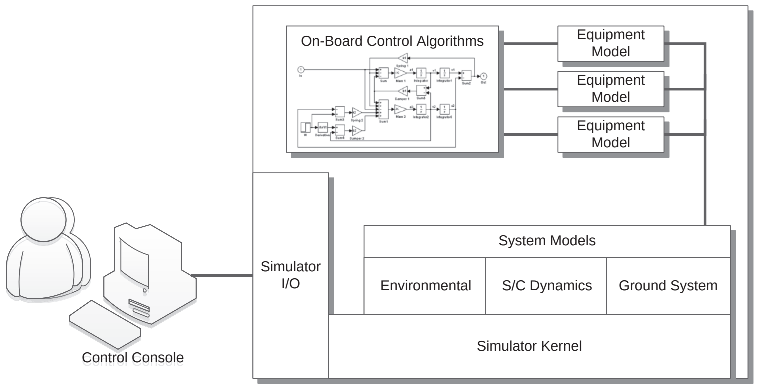

Figure 5.1: Matlab/Simulink models of the S/C algorithms running on an FVB/FES

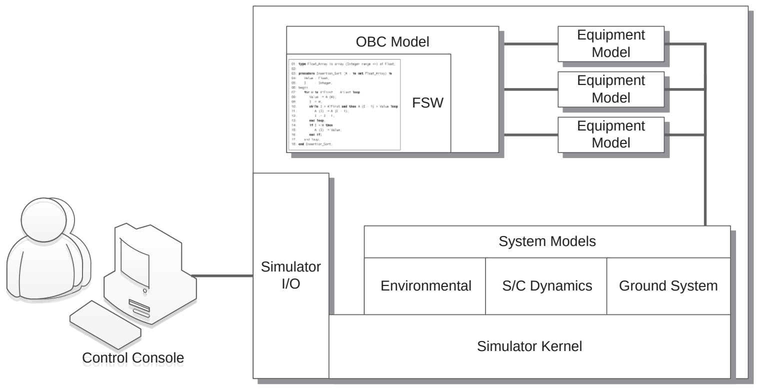

Figure 5.2: FSW running on an emulated OBC in an SVF

Furthermore, information gathered from the two Large System Integrators (LSIs) in Germany [1] on several projects[2] provides some insight into the heterogeneous environments for AOCS algorithms development (Figure 5.3).

Thus, different processor architectures are used through the development phases:

- x86-64 architecture based processors on development PCs.
- SPARC V8 architecture based processors on the target on-board architecture in SVF platforms and on the final On-Board Computer (OBC).

The development and programming environments are also diverse:

- Mathworks Matlab development environments:
  - Matlab programming
  - Simulink model development
  - Matlab programming *embedded* in Simulink models
  - Use of Simulink *accelerator mode* with C-MEX functions either manually written or auto-generated.
  - Source code auto-generation with tools like Simulink Coder.
- C programming environment, usually based on the GNU C Compiler (GCC) compiler.
- Ada programming environment.

GTD-TR-01-BL-0007 The BL shall be compatible with the GCC compiler and with Binutils.

[1] Airbus DS and OHB
[2] Meteosat Third Generation (MTG) AOCS and equipment simulation model development, AS400 satellite platform equipment simulation model development, and Sentinel 2 AOCS development.

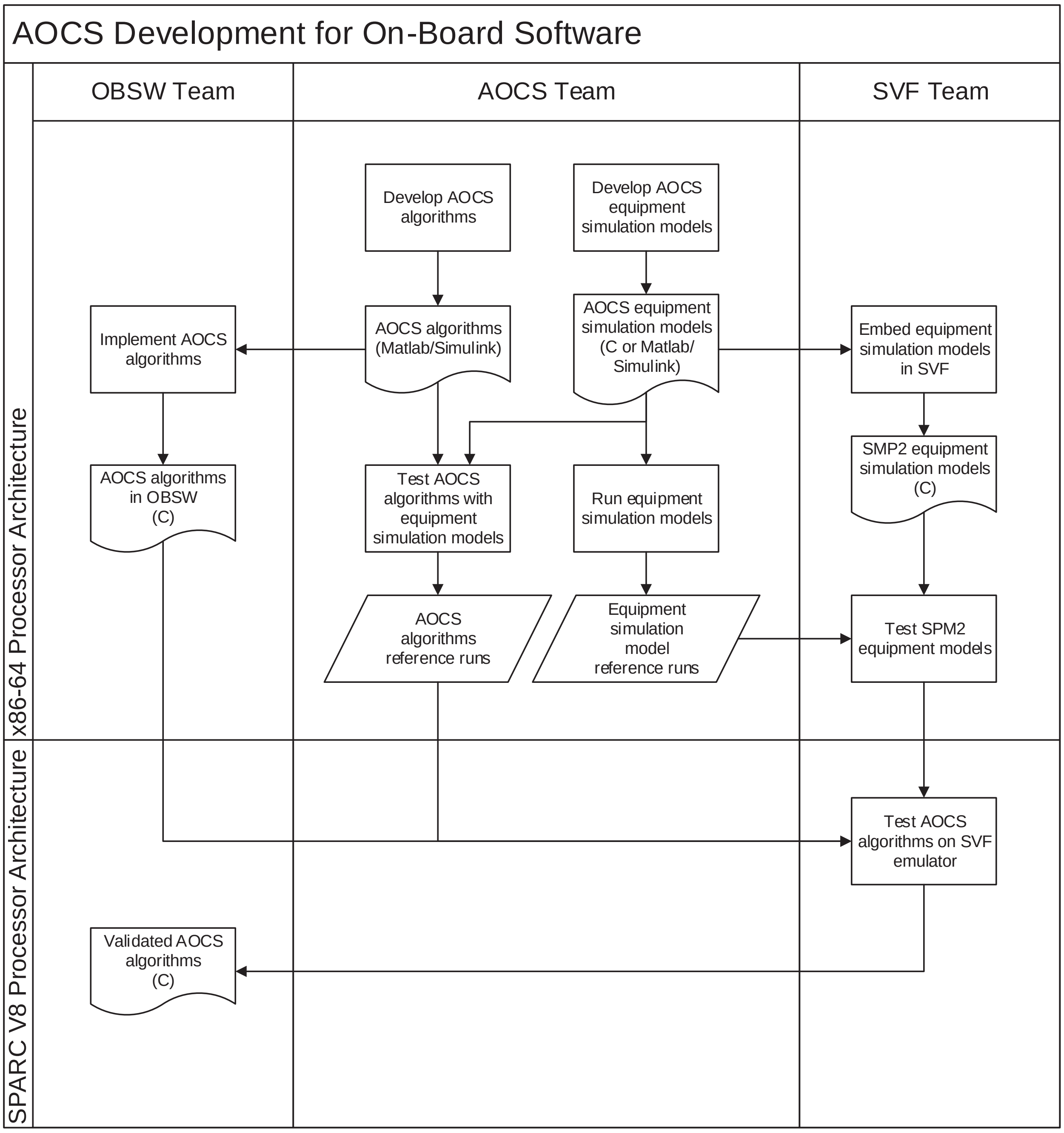
AOCS Development for On-Board Software
OBSW Team
AOCS Team
SVF Team
x86-64 Processor Architecture
SPARC V8 Processor Architecture
Develop AOCS algorithms
Develop AOCS equipment simulation models
Implement AOCS algorithms
AOCS algorithms (Matlab/Simulink)
AOCS equipment simulation models (C or Matlab/ Simulink)
Embed equipment simulation models in SVF
SMP2 equipment simulation models (C)
AOCS algorithms in OBSW (C)
Test AOCS algorithms with equipment simulation models
Run equipment simulation models
AOCS algorithms reference runs
Equipment simulation model reference runs
Test SPM2 equipment models
Test AOCS algorithms on SVF emulator
Validated AOCS algorithms (C)

Figure 5.3: Example of AOCS algorithm and equipment simulation model development for on-board software

GTD-TR-01-BL-0008 The BL shall provide best practices to develop an interface to access all its functionality from Matlab.

GTD-TR-01-BL-0009 The BL shall provide best practices to develop an interface to access all its functionality from Simulink.

GTD-TR-01-BL-0010 The BL shall provide best practices to develop C-MEX function wrappers for all its procedures.

GTD-TR-01-BL-0011 The BL shall provide best practices to develop S-function wrappers for all its procedures.

GTD-TR-01-BL-0012 The BL shall be compatible with source code auto-generated with the Simulink Coder.

GTD-TR-01-BL-0013 The BL shall provide best practices to develop an interface to access all its functionality from Ada.

The combination of different processor architectures (x86-64 and SPARC V8) and programming environments (Matlab/Simulink, C, and Ada) for algorithms and equipment simulation-model development, requires a high degree of cross-platform reproducibility of numerical computations if reliable and comparable results are to be obtained on each platform for a conclusive validation of the on-board software.

The capability to compare the numerical results obtained on x86-64 platforms running Matlab code and the numerical results obtained with the on-board software developed in C running on the SPARC V8 target architecture would provide the possibility to rely on these comparable results when searching for a software problem, making the detected problems much easier to reproduce on the development/simulation environment.

GTD-TR-01-BL-0014 The BL shall obtain numerically reproducible results on x86-64 and SPARC V8 processors.

GTD-TR-01-BLTS-0001 The BL shall be tested on a LEON2, a LEON4, and a x86-64 platform for its reproducibility.

5.3 Mathematical and Numerical Requirements for On-Board Software

Numerical computation requirements for S/C software development come from domain specific needs such as the ones described in § 5.1.

To satisfy these demands a mathematical library shall provide procedures for at least some elementary mathematical *functions*. These procedures will then be combined to construct the software modules that respond to those needs.

Messerschmid and Fassoulas (see [MF05], "Anhang C: Formelsammlung"), and Eickhoff (see [Eic09], §6 "Numerical Foundations of System Simulation") offer information for an assessment of which elementary functions are needed for S/C software and simulator development. Together with information collected from different AOCS developments[3] the following list has been obtained[4]:

- Trigonometric functions: `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, and `atan2`
- Rounding operations: `floor`, `ceil`, `round`, and `trunc`
- Vector operations: dot product, cross product, and vector norm (`hypot` in the case of two dimensional vectors)
- Other operations: `sqrt`, `fmod`, `fabs`, and `signbit`

A basic mathematical library shall provide at least these elementary functions with the exception of the vector operations dot product, cross product, and vector norms above two dimensions, which are not provided by basic libraries like the `libm` in `Newlib` and are also not covered by the C standard. These operations can be part of a high level mathematical library, since their development poses challenges of different nature to the other elementary functions. The optimized handling of arrays and matrices, and operating on data blocks have been the drivers of the development of higher level libraries such as Basic Linear Algebra Sub-programs (BLAS) for this purpose.

The elementary functions required for scientific data processing are dependent on their specific purpose and thus, impossible to list. However, the previous list of elementary functions shall be reasonably extended for the BL to provide some more functionality for on-board scientific computations. At least the procedures `exp`, `pow`, logarithms (base 10, `log10`, and e, `log`), `modf`, `fmin` and `fmax`, `isfinite`, `isinf`, `isnan`, and `copysign` shall be included to obtain the list of procedures proposed by the Statement of Work (SoW) ([AD01]) for S/C GNC and on-board scientific computing purposes.

GTD-TR-01-BL-0015 The BL shall provide the following procedures: `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `atan2`, `floor`, `ceil`, `round`, `trunc`, `hypot`, `sqrt`, `fmod`, `remainder`, `fabs`,

[3]Specifically the AOCS Software (SW) of the the satellites MTG and Sentinel 2.

[4]To use a common language with [AD01] whenever possible we will use here the names of the procedures of `math.h` that implement the evaluations of the required elementary functions.

`signbit`, `modf`, `exp`, `pow`, `log10`, `log`, `fmin`, `fmax`, `isfinite`, `isinf`, `isnan`, and `copysign`.

GTD-TR-01-BL-0016 The BL shall be compatible to `math.h` regarding the procedures defined by ISO/IEC 9899.

GTD-TR-01-BL-0017 The BL shall provide float (32 bit floating-point) and double (64 bit floating-point) versions of the procedures.

For compatibility to existing code and to the programming experience of S/C SW developers it is important that the BL offers a similar interface to the `math.h` present in the ISO C compliant mathematical libraries widely used; not only in the exposed procedures but also in the provided constants as defined by the Portable Operating System Interface (POSIX) standard for float and double floating-point numbers.

GTD-TR-01-BL-0018 The BL shall provide the following constants as defined by the POSIX standard: `M_E`, `M_LOG2E`, `M_LOG10E`, `M_LN2`, `M_LN10`, `M_PI`, `M_PI_2`, `M_PI_4`, `M_1_PI`, `M_2_PI`, `M_2_SQRTPI`, `M_SQRT2`, `M_SQRT1_2`, MAXFLOAT, HUGE_VAL, HUGE_VALF, `INFINITY`, NAN.

The fact that operations with angles are very extended in S/C SW and that the ISO C99 standard requires the use of radians as arguments make it reasonable two include two extra functions: one for converting from degrees to radians and one for reducing the value of an angle to a range where trigonometric operations on the angle will yield the same result as on the original value.

GTD-TR-01-BL-0019 The BL shall provide a procedure to convert angles in degrees to radians.

GTD-TR-01-BL-0020 The BL shall provide a procedure to convert an angle to an equivalent one in the range $[0, 2\pi]$.

5.3.1 Numerical Reproducibility

A further numerical requirement, the reproducibility of numerical computations (as defined by Revol and Théveny in [RT13]), together with the accuracy, are sometimes surrounded in *mythical thought*[5].

On one side, it is often thought that the compliance to the IEEE 754 standard automatically yields reproducible results on different platforms. The reality is that this is not the purpose of the standard and that the standard leaves room for implementation dependent decisions that lead to

[5]As opposed to scientific thought.

non-reproducible results even possibly between different executions on the same platform (see [CK09]). On the other side, the assumption of unavoidable stepwise degradation of accuracy because of round-off errors in floating-point computations produces numerical requirements such as the hypothetical: "All computations shall be done in double (64 bit) precision" and "The relative error shall be smaller than $10^{-}8$"; these would require a proper numerical analysis to justify the big loss of accuracy from the first to the second requirement.

Thus, reproducibility shall be assessed by numerical analysis. This kind of analysis shows that processor and FPUs features, together with the developed SW procedures' characteristics define the reproducibility of the results. An analysis of possible SW procedures is out of the scope of this document, we will focus on the impact of FPU characteristics.

Since the reproducibility of numerical computations on different processor architectures and software environments is necessary for a consistent S/C software development, requirements and constraints in the use of FPU features will be necessary, to define execution environments that are as homogeneous as possible.

FPUs like the MEIKO, GRFPU, or the ones present in x86-64 processors have built in algorithms for some elementary mathematical functions (e.g. square root). The amount of provided functionality is different from FPU to FPU and so are the algorithms provided for a given elementary function. Thus, the use of these built in functions on different platforms will yield different numerical results. This shall be avoided by the BL.

GTD-TR-01-BL-0021 The BL shall not use FPU built in functions for elementary math function evaluation.

Another feature of FPUs, which is present on some x86-64 processors but is not present on the MEIKO and GRFPUs, is the Fused Multiply-Add (FMA) operation. Its presence or absence also results in inconsistent numerical results on different platforms, since the rounding error of an operation is not the same if the multiplication and an addition are done in a single step (i.e. with FMA) or in two (i.e., without FMA). Thus, the BL shall not rely on the presence of an FMA operation for algorithm optimization (e.g., for polynomial evaluation with Horner's method). The FMA feature is a requirement of the IEEE 754 standard in its 2008 version [AD01] but this was not the case in the 1985 original version. This is the reason that the MEIKO FPU does not contain it although being IEEE 754 standard compliant.

GTD-TR-01-BL-0022 The BL shall not rely on the presence of the FMA operation for algorithm optimization.

The treatment of denormal or subnormal floating-point numbers by the FPU poses an additional challenge to the production of reproducible results on different architectures. The ability of the FPUs to produce and consume subnormal floating-point numbers opposed to their treatment as 0, produces clearly different numerical results.

Subnormals are floating-point numbers smaller than the smallest *normal* floating-point number and bigger than 0. Normal floating-point numbers are those that have 1 as their most significant bit. This is accomplished by shifting the bits in the mantissa to the left and lowering the exponent by the same amount. Since for all normal floating-point numbers this bit is equal to

1 it is not even stored. If a floating-point number is produced that has the lowest exponent and one or more zeros as its most significant bits, this number cannot be *normalized* by shifting the mantissa left, since the exponent cannot be further lowered. These floating-point numbers are called denormals or subnormals.

These subnormal floating point numbers are defined in the IEEE 754 standard and provide at least the following features:

- They extend the floating-point range linearly between 0 and the smallest normal floating-point number.
- They permit the so called *gradual underflow*, since subnormal numbers are produced where an abrupt underflow would have occurred.

The backside for these features are some drawbacks:

- The use of subnormals produces loss of precision. There are less significant bits in the mantissa, since the leftmost bits are 0.
- The FPU performance can be up to about 100 times worse when operating with subnormals when special microcodes are used for this purpose.

Because of this reason, several FPUs do not handle subnormal floating-point numbers. Among these is the GRFPU, but not the Intel FPUs and the MEIKO FPU.

The FPUs not handling subnormal numbers either round the produced results up to the smallest floating-point number or flush the result to zero (FTZ) and set the corresponding underflow bit. When handling subnormals as input arguments, they set an FPU trap (as in the case of the GRFPU), which is handled as a fatal error by operating systems and such as RTEMS and VxWorks, and cross compilation systems such as Bare-C Cross Compilation System (BCC). To prevent this behavior, FPUs like the GRFPU must be set in a non-standard mode (non standard regarding IEEE 754) that handles the subnormals as zero. This mode can also be enforced on Intel x86-64 processors, writing to the corresponding SSE register and setting the Denormals-Are-Zero (DAZ) and Flush-To-Zero (FTZ) modes.

GTD-TR-01-BL-0023 The BL shall have a configurable modus to handle denormal arguments as zero.

GTD-TR-01-BL-0024 The BL shall have a configurable modus to flush denormal computation results to zero.

The reproducibility of results is also affected by the compiler used and the processor's instruction pipelining. Guidelines and best practices are defined in the next chapter to address these aspects and to make numerical reproducibility, and known accuracy, plausible aspirations for S/C on-board software development.

5.4 Other Non-Functional Requirements

The most common execution environments of the BL in S/C systems will be bare-metal environments (without an OS) and environments with an POSIX-like OS like RTEMS.

In either environment On-Board Software (OBSW) developers will expect the BL will behave in accordance to POSIX as far as it concerns the use of procedures present in `math.h`. An important exception to this is the use of the *global*[6] error handling variable `errno`, which is not used in OBSW developments[7].

GTD-TR-01-BL-0025 The BL shall not use the `errno` global variable for error handling.

GTD-TR-01-BL-0026 The BL procedures shall behave as defined by the POSIX standard for *special cases* i.e. regarding arguments and outputs involving Not a Number (NaN), infinities, and ±0.

These behaviors of the procedures for *special* arguments are difficult to present in prose thus, since a clear characterization of the implemented procedures is mandatory for the BL, a tabular form will be used in the design and the user manual of the BL.

GTD-TR-01-BL-0027 The BL design shall define the behavior of the procedures in a tabular format, presenting the parameter domain decomposition and the corresponding output.

Regarding multi-threading, having thread-safe and reentrant procedures is mandatory for a numerical computation library that will run on multi-threaded and real time applications on OSs such as RTEMS.

GTD-TR-01-BL-0028 The BL procedures shall be reentrant and thread-safe.

Regarding the performance, the most important characteristic of the BL to be used for S/C SW is that the Worst Case Execution Time (WCET) of the implemented procedures shall be bounded and well defined over the complete argument domain. In addition, the memory footprint of the BL shall be kept minimal and very close to the one that the `libm` in `Newlib` has.

GTD-TR-01-BL-0029 The BL procedures shall have a bounded WCET.

GTD-TR-01-BL-0030 The BL procedures shall have a well defined WCET over the complete argument domain.

[6]The ISO C99 standard does not define `errno` to be explicitly a global variable, but just a modifiable *lvalue* that is accessible after a function call.

[7]Some reasons for this are that the thread-safety of this variable is implementation dependent and that its use was forbidden by MISRA C (MISRA C:2004 rule 20.5)

GTD-TR-01-BL-0031 The BL shall have a memory footprint at most 30% bigger than the `libm` in `Newlib`.

GTD-TR-01-BLTS-0002 The Basic mathematical Library Test Suite (BLTS) shall perform execution time tests, analyzing the minimum, maximum, average, and median execution times of the procedures.

GTD-TR-01-BLTS-0003 The BLTS shall perform memory usage tests, analyzing the memory usage of the BL procedures.

5.5 Reusing the `libm` in `Newlib` for the Mathematical Library

The `libm` included in `Newlib`, originally developed at the University of California in Berkeley in the late eighties, later reorganized into Freely Distributable Math Library (FDLIBM) by Sun Microsystems in 1993 before it was included i `Newlib`, has an important heritage of use for on-board software in the European space industry.

Its properties as a portable library and wide use in embedded systems and the European space industry, make this `libm` a good candidate to produce, by a re-engineering process, the BL with the characteristics described in this document.

GTD-TR-01-BL-0032 The BL design shall document every modification applied to the reused library.

5.6 Testing the Mathematical Library

To be able to pre-qualify the BL a test suite (the BLTS) will have to be developed and qualified also in compliance with ECSS. This BLTS will have a component running on a Windows or GNU/Linux *host* system and will control the execution of the test cases (the other component) on the target platforms.

GTD-TR-01-BLTS-0004 The BLTS shall be ECSS category D compliant.

GTD-TR-01-BLTS-0005 The execution of the BLTS shall controlled from Windows or GNU/Linux *host* systems.

Further, to test the BL it is mandatory to have a clear method to:

- produce a defined and unambiguous set of input and output values to check the accuracy of the evaluation of the mathematical functions implemented in the procedures of the BL and
- to measure the difference between the evaluation of the BL procedures and the reference output values.

The input values will have to be unambiguously defined, guaranteeing that the evaluation with the BL and the evaluation for reference is done on the same argument or arguments. For example, the input values shall not be defined in their decimal representation if it cannot be guaranteed that their conversion to machine representation will yield the same result[8].

GTD-TR-01-BLTS-0006 The BLTS shall have a set of unambiguous input values for testing the BL in the form of defined 32 bit and 64 bit floating-point datums.

The most extended way to present differences of accuracy between floating-point calculations is to present this difference in Units in the Last Place (ULPs). The notion of ULP is a relative error measure but it is not always defined the same way and the implementations to obtain accuracy differences in ULPs may vary. [Hig02] gives in §2.1 the definition $ulp(y) = \beta^{e-t}$[9] and states that "If x is any real number we can say that y and x agree to $|y - x|/ulp(y)$ ULPs in y". The Boost C++ library provides the `float_distance` function to obtain the "representation distance between two floating point values (ULP)", which is a different concept than the one defined by [Hig02], since it only measures the *representation distance* between two floating-point numbers (which is always an integer), while the definition by Higham measures the difference between a real number and a floating-point number in ULPs of the floating-point number (which is not an integer in the general case).

This requires a clear definition of the error measure presentation when testing the BL.

GTD-TR-01-BLTS-0007 The BLTS shall measure the differences in accuracy in ULPs between the results of the BL procedures and the reference output values.

GTD-TR-01-BLTS-0008 The BLTS shall have a definition of ULP and how two floating-point numbers differ in ULPs.

GTD-TR-01-BL-0033 The BL procedures shall obtain results with a maximum difference of 0.5 ULPs to the actual value of the mathematical function they implement.

[8]The conversion of the literal string "1.4142135623730950488016887242097" to an IEEE 754 floating point number may yield different results depending on the used algorithm [Reg].

[9]Where β is the base or radix of the floating-point system, t the precision or size of the mantissa, and e the exponent of the floating-point representation of y.

The reference output values, with which the outputs of the BL are going to be compared to, shall be values obtained with a reliable method and their accuracy and precision shall be in any case higher than the one achievable with the BL to be tested, to guarantee that the BL is tested against mathematically more correct results.

GTD-TR-01-BLTC-0009 The test cases of the BLTS shall have reference output values that have been produced with an arbitrary precision mathematical software set to a higher precision than 64 bits.

GTD-TR-01-BLTC-0010 The reliability of the arbitrary precision mathematical software used to produce the reference output values to test the BL shall be assessed and justified.

The BLTS shall produce test result reports, after running the test cases in batch or manual modus on the target platforms, that will have to be sent back to the host system controlling the test execution for their compilation into documents for pre-qualification evidence. The dependencies to run the test suite shall be minimal, to make sure the test suite can be reused on target systems where a delta qualification is required. A widely available interface for accessing and controlling the execution of SW on OBCs is GNU Debugger (GDB).

GTD-TR-01-BLTS-0011 The BLTS shall run in batch and manual modus.

GTD-TR-01-BLTS-0012 The BLTS shall have no dependencies to commercial third party software.

GTD-TR-01-BLTS-0013 The BLTS shall use GDB as the main interface to the Hardware (HW) execution platforms.

GTD-TR-01-BLTS-0014 The BLTS produce test result reports on the target platforms and get them back to the host system.

The reports produced by the BLTS shall also include structural coverage results. Providing sufficient evidence on the achieved coverage with the tests that have been run on the targets is an important measure of the quality of the tests. Statement coverage data can be obtained while running the test suite on the target, while loop and Multiple Condition Decision Coverage (MC/DC) coverage data can be obtained *off line*, using proprietary tools on the test cases set, but not while running the BLTS. Path coverage shall also be analyzed for those cases where a very high cyclomatic complexity of a procedure demands extra evidence for an adequate test coverage.

GTD-TR-01-BLTS-0015 The BLTS shall gather statement coverage data while running the test suite on target and on x86-6.

GTD-TR-01-BLTC-0016 The BLTS shall provide loop and MC/DC coverage data obtained while developing the test suite.

GTD-TR-01-BLTC-0017 The BLTS shall analyze manually the path coverage of procedures with a very high cyclomatic complexity.