AN4230
Application note

STM32 microcontrollers random number generation validation using NIST statistical test suite

Introduction

Many standards have created requirements and references for the construction, the validation and the use of random number generators (RNG), in order to verify that the output they produce is indeed random.

The purpose of this application note is to provide guidelines to verify the randomness of the numbers generated by the random number generator peripheral embedded in a selection of STM32 microcontrollers. This verification is based on the National Institute of Standards and Technology (NIST) Statistical Test Suite (STS) SP 800-22rev1a (April 2010).

This document is structured as follows:

- A general introduction to STM32 microcontrollers random number generator (see *Section 1*)
- The NIST SP800-22b test suite (see *Section 2*)
- The steps needed to run NIST SP800-22b test and analysis (see *Section 3*)

Table 1. Applicable products

Type	Product series
Microcontrollers	STM32F2 series, STM32F4 series, STM32F7 series, STM32L0 series, STM32L4 series.

Contents

List of figures

1 STM32 microcontrollers random number generator

1.1 Introduction

Random number generators (RNG) used for cryptographic applications typically produce sequences made of random 0 and 1 bits.

There are two basic classes of random number generators:

1. Deterministic random number generator or pseudo random number generator (PRNG):
   A deterministic RNG consisting of an algorithm that produces a sequence of bits from an initial value called a seed. To ensure forward unpredictability, care must be taken in obtaining seeds. The values produced by a PRNG are completely predictable if the seed and generation algorithm are known. Since in many cases the generation algorithm is publicly available, the seed must be kept secret and generated from a TRNG.
2. Non deterministic random number generator or True random number generator (TRNG):
   A nondeterministic RNG producing randomness which depends on some unpredictable physical source (i.e. the entropy source) outside of any human control.

The RNG hardware peripheral implemented on the STM32 microcontrollers and described in *Section 1.2.1* is a true random number generator.

1.2 STM32 microcontrollers implementation description

1.2.1 True random number generator

The following table lists the STM32 microcontrollers lines that embed the RNG hardware peripheral:

Table 2. STM32 lines embedding the RNG hardware peripheral

STM32 series	STM32 lines
STM32F2	STM32F2x5 STM32F2x7
STM32F4	STM32F405/415 STM32F407/417 STM32F410 STM32F427/437 STM32F429/439 STM32F469/479
STM32F7	STM32F7x5 STM32F7x6
STM32L0	STM32L05x STM32L06x STM32L072/073
STM32L4	STM32L4x6

The True random number generator peripheral implemented on STM32 microcontrollers is based on an analog circuit. This circuit generates a continuous analog noise that will be used on the RNG processing in order to produce a 32-bit random number.

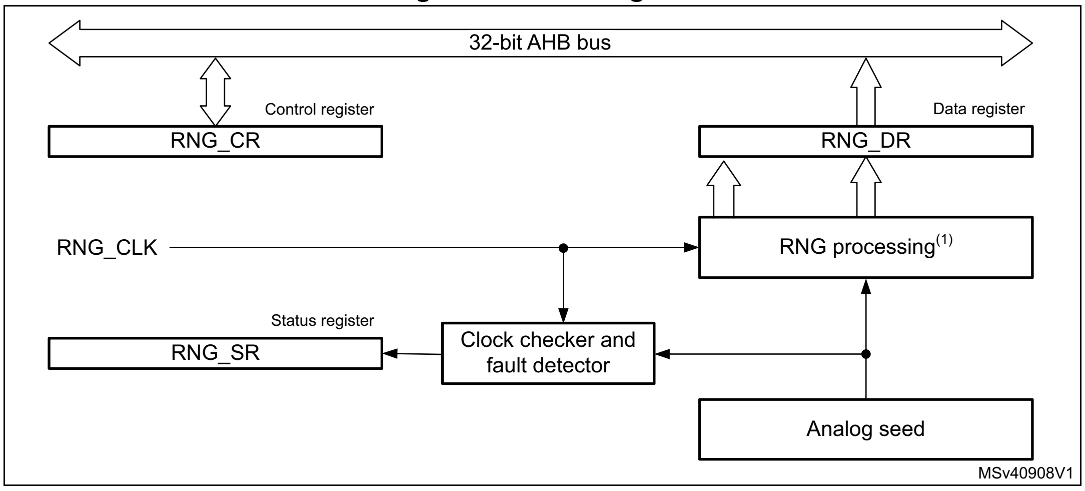
The analog circuit is made of several ring oscillators whose outputs are XORed.

The RNG processing is clocked by a dedicated clock at a constant frequency and for a subset of microcontrollers it can also be clocked with a different value of frequency.

For more details about the RNG peripherals please refer to the STM32 microcontroller reference manual.

Figure 1 shows the TRNG block diagram on STM32 microcontrollers.

Figure 1. Block diagram



2 NIST SP800-22b test suite

2.1 Introduction

The NIST SP800-22b statistical test suite has been carried out using a statistical test suite (sts) developed by the *National institute of standards and technology (*NIST) to probe the quality of random generators for cryptographic applications. A comprehensive description of the suite was presented in the paper entitled: *"A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications"*.

2.2 NIST SP800-22b test suite description

The NIST SP800-22b statistical test suite "sts-2.1.1" is a software package which was developed by the *National Institute of Standards and Technology*; it can be downloaded from the NIST web site: search for "*NIST Statistical Test Suite download*" at http://csrc.nist.gov.

The source code has been written in ANSI C. The NIST statistical test suite consists of 15 tests that were developed to test the randomness of a binary sequence. These tests focus on a variety of types of non-randomness that could exist in a sequence. From this point of view, the test suite can be classified as follows:

Frequency tests

- Frequency (Monobit) test:
  To measure the distribution of 0's and 1's in a sequence and to check if the result is similar to the one expected for a truly random sequence.
- Frequency test within a block:
  To check whether the frequency of 1's in an M-bit block is approximately M/2, as would be expected from the theory of randomness.
- Run tests:
  To assess if the expected total number of runs of 1's and 0's of various lengths is as expected for a random sequence.
- Test of the longest run of 1's in a block:
  To examine the long runs of 1's in a sequence.

Test of linearity

- Binary matrix rank test:
  To assess the distribution of the rank for 32x32 binary matrices.
- Linear complexity test:
  To determine the linear complexity of a finite sequence.

Test of correlation (by means of Fourier transform)

- Discrete Fourier transform (spectral) test:
  To assess the spectral frequency of a bit string via the spectral test based on the discrete Fourier transform. It is sensitive to the periodicity in the sequence.

Test of finding some special strings

- Non-overlapping template matching test:
  To assess the frequency of m-bit non-periodic patterns.
- Overlapping template matching test:
  To assess the frequency of m-bit periodic templates

Entropy tests

- Maurer's "Universal Statistical" test:
  To assess the compressibility of a binary sequence of L-bit blocks.
- Serial test:
  To assess the distribution of all 2^m m-bit blocks.

Note: For m = 1, the serial test is equivalent to the Frequency test of Section 2.2.

- Approximate entropy test:
  To assess the entropy of a bit string, comparing the frequency of all m-bit patterns against all (m+1)-bit patterns.

Random walk tests

- Cumulative sums (Cusums) test:
  To assess that the sum of partial sequences is not too large or too small; it is indicative of too many 0's or 1's.
- Random excursion test:
  To assess the distribution of states within a cycle of random walk.
- Random excursion variant test:
  To detect deviations from the expected number of visits to different states of the random walk.

Each of the above tests is based on a calculated test statistic value, which is a function of the testing sequence. The test statistic value is used to calculate a Pvalue where:

Pvalue is the probability that the perfect random number generator would have produced a sequence less random than the sequence that was tested.

For more details about the NIST statistical test suite, refer to the following NIST document available on the NIST web site:

"A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications" Special Publication 800-22 Revision 1a.

3 NIST SP800-22b test suite running and analyzing

3.1 Firmware description

To run the NIST statistical test suite as described in the previous section, two firmware are needed, one on the STM32 microcontroller side and one on the NIST SP800-22b test suite side.

3.1.1 On the STM32 microcontroller side

The firmware package is provided upon request, for more details please contact your local ST sales representative.

The program allows to generate random numbers using the STM32 microcontroller random number generator (RNG) peripheral and to retrieve them on a workstation for testing with the NIST statistical test suite.

Each firmware program is used to generate 10 blocks of 64,000 bytes of random number, so the output file will contain 5,120,000 random bits to be tested with the NIST statistical test.

As recommended by the NIST statistical test suite, the output file format can be:

1. a sequence of ASCII 0's and 1's if the *FILE_ASCII_FORMAT* Private define is uncommented in the main.c file
2. A binary file of random bytes if the *FILE_BINARY_FORMAT* Private define is uncommented in the main.c file.

Form more details about the program description and settings, refer to the readme file inside the firmware package.

Note: The USART configuration can be changed by the user via the SendToWorkstation() function in the main.c file.

The output values can be changed by the user by modifying the Private define in the main.c file:

```
#define NUMBER_OF_RANDOM_BITS_TO_GENERATE   512000
#define BLOCK_NUMBER 10
```

3.1.2 On the NIST SP800-22b test suite side

Downloaded on a workstation, the NIST statistical test suite package sts-2.1.1 verifies the randomness of the output file of the random number generator from STM32 microcontrollers.

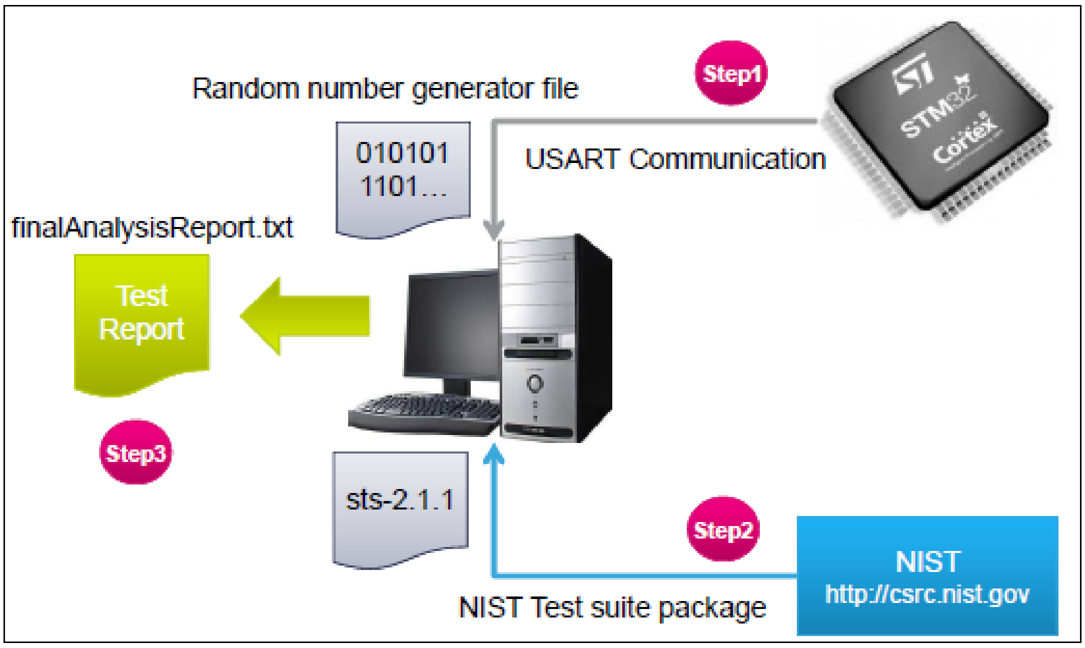
The generator file to be analyzed should be stored under the *data* folder (sts-2.1.1\data).

For more details about how the NIST statistical tests work, refer to section How to Get Started in *“A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications” Special Publication 800-22 Revision 1a*.

3.2 NIST SP800-22b test suite steps

Figure 2 describes the different steps needed to verify the randomness of an output number generated by STM32 microcontrollers using the NIST statistical test suite package sts-2.1.1.

Figure 2. Block diagram of deviation testing of a binary sequence from randomness based on NIST test suite



3.2.1 First step: random number generator

Connect the STMicroelectonics board to the workstation. Depending on the type of board, the connection is made via:

- a null-modem female/female RS232 cable
- a USB “type A to Mini-B" cable

Run STM32 microcontroller RNG generation via the UART firmware in order to generate a random number as described in the previous section. You can store data on the workstation using a terminal emulation application such as HyperTerminal (a program that comes with Windows operating system, from Windows 98).

3.2.2 Second step: NIST statistical test

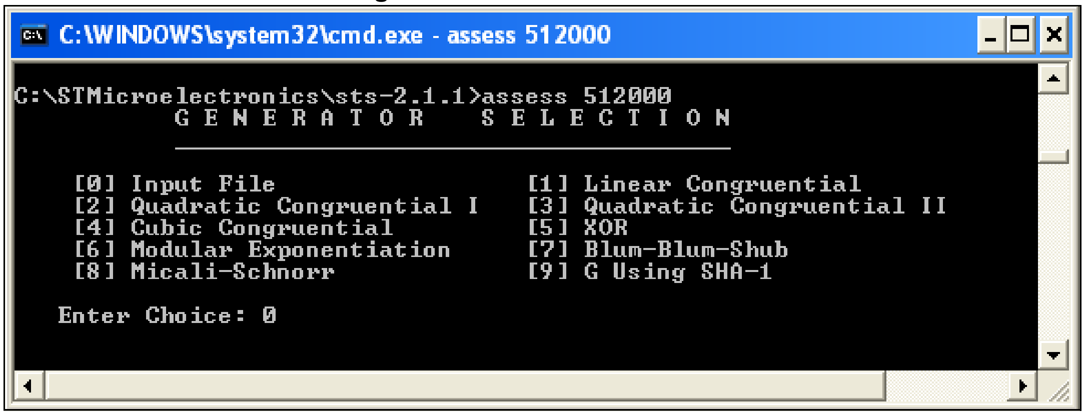
Compile sts-2.1.1 package as described in the NIST statistical test suite documentation in order to create an executable program using visual C++ compiler.

After running the NIST statistical test suite program, a series of menu prompts will be displayed in order to select the data to be analyzed and the statistical tests to be applied.

In this application note, the NIST statistical test suite is compiled under the name *assess.exe* and saved under NIST_Test_Suite_OutputExample folder. As described in the previous section, the random number is defined as *512,000 bits per block*.

The first screen appears as in *Figure 3*.

Figure 3. Main sts-2.1.1 screen

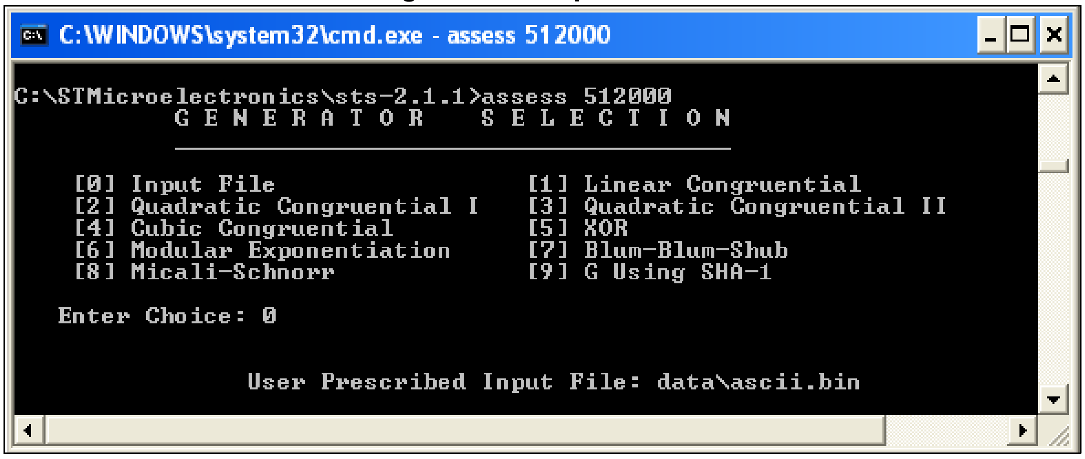


```
C:\WINDOWS\system32\cmd.exe - assess 512000

C:\STMicroelectronics\sts-2.1.1>assess 512000
           G E N E R A T O R    S E L E C T I O N
           ______________________________________

    [0] Input File                  [1] Linear Congruential
    [2] Quadratic Congruential I    [3] Quadratic Congruential II
    [4] Cubic Congruential          [5] XOR
    [6] Modular Exponentiation      [7] Blum-Blum-Shub
    [8] Micali-Schnorr              [9] G Using SHA-1

   Enter Choice: 0
```

When value 0 is entered, the program requires to enter the file name and path of the random number to be tested.

The second screen appears as in *Figure 4*.

Figure 4. File input screen



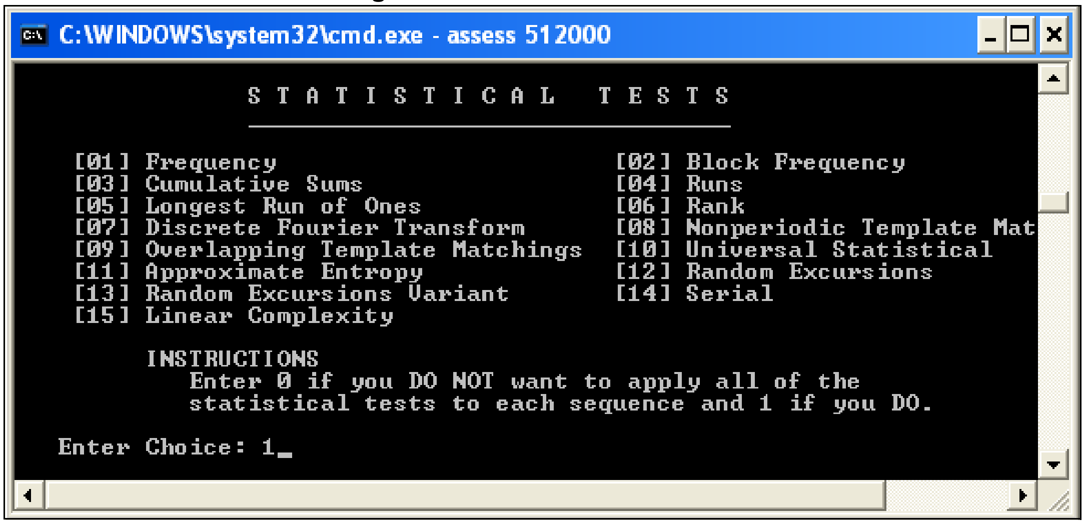
```
C:\WINDOWS\system32\cmd.exe - assess 512000

C:\STMicroelectronics\sts-2.1.1>assess 512000
           G E N E R A T O R    S E L E C T I O N
           ______________________________________

    [0] Input File                  [1] Linear Congruential
    [2] Quadratic Congruential I    [3] Quadratic Congruential II
    [4] Cubic Congruential          [5] XOR
    [6] Modular Exponentiation      [7] Blum-Blum-Shub
    [8] Micali-Schnorr              [9] G Using SHA-1

   Enter Choice: 0


                User Prescribed Input File: data\ascii.bin
```

This application note provides you with an example of two files per series, generated with STM32 microcontroller random number generator with different file formats as recommended by NIST:

1. *ascii.bin:* sequence of ASCII 0's and 1's.
2. *binary.bin*: each byte in the data file contains 8 bits of data.

Then, the NIST statistical test suite displays 15 tests that can be run via the screen in *Figure 5*.

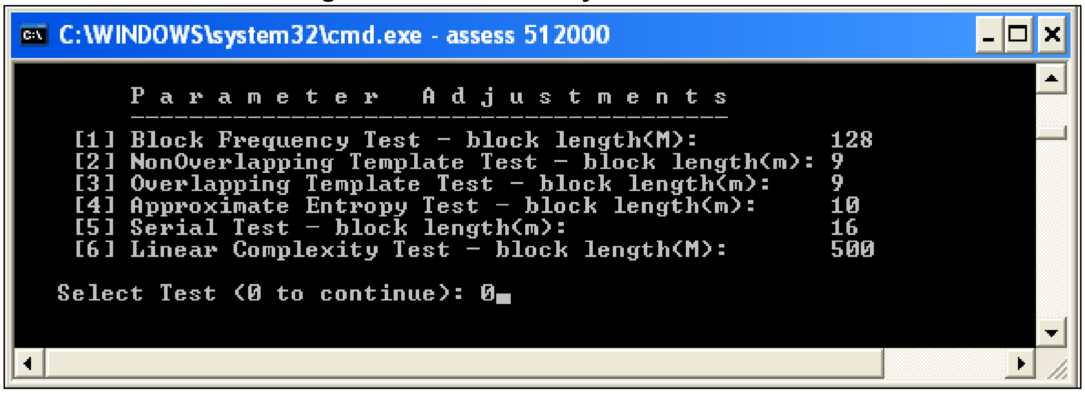
Figure 5. Statistical test screen



```
C:\WINDOWS\system32\cmd.exe - assess 512000

                S T A T I S T I C A L   T E S T S
                _________________________________

    [01] Frequency                       [02] Block Frequency
    [03] Cumulative Sums                 [04] Runs
    [05] Longest Run of Ones             [06] Rank
    [07] Discrete Fourier Transform      [08] Nonperiodic Template Mat
    [09] Overlapping Template Matchings  [10] Universal Statistical
    [11] Approximate Entropy             [12] Random Excursions
    [13] Random Excursions Variant       [14] Serial
    [15] Linear Complexity

         INSTRUCTIONS
            Enter 0 if you DO NOT want to apply all of the
            statistical tests to each sequence and 1 if you DO.

   Enter Choice: 1
```

In this case, ‘1’ has been selected to apply all of the statistical tests. *Figure 6* is displayed to change the parameter adjustments.

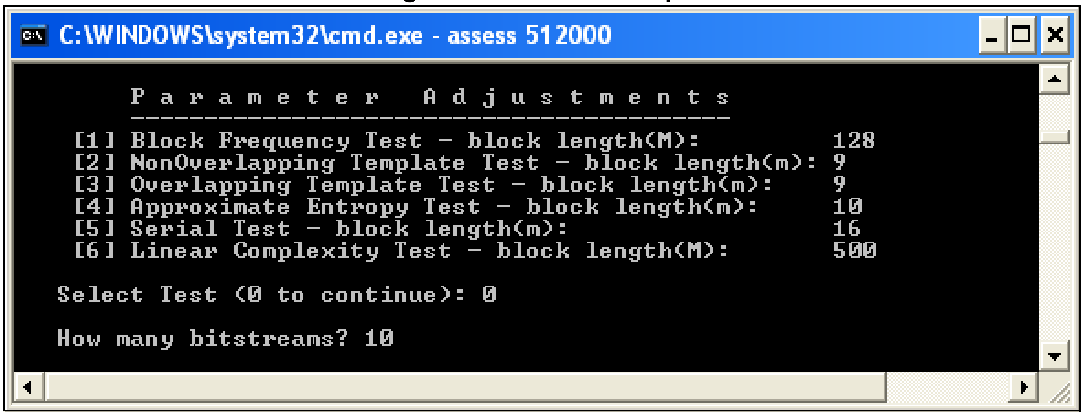
Figure 6. Parameter adjustment screen



```
C:\WINDOWS\system32\cmd.exe - assess 512000

        P a r a m e t e r   A d j u s t m e n t s
        -----------------------------------------
    [1] Block Frequency Test - block length(M):         128
    [2] NonOverlapping Template Test - block length(m): 9
    [3] Overlapping Template Test - block length(m):    9
    [4] Approximate Entropy Test - block length(m):     10
    [5] Serial Test - block length(m):                  16
    [6] Linear Complexity Test - block length(M):       500

   Select Test (0 to continue): 0
```

In this example, the default settings are kept and value ‘0’ is selected to go to the next step.

Figure 7. Bitstreams input



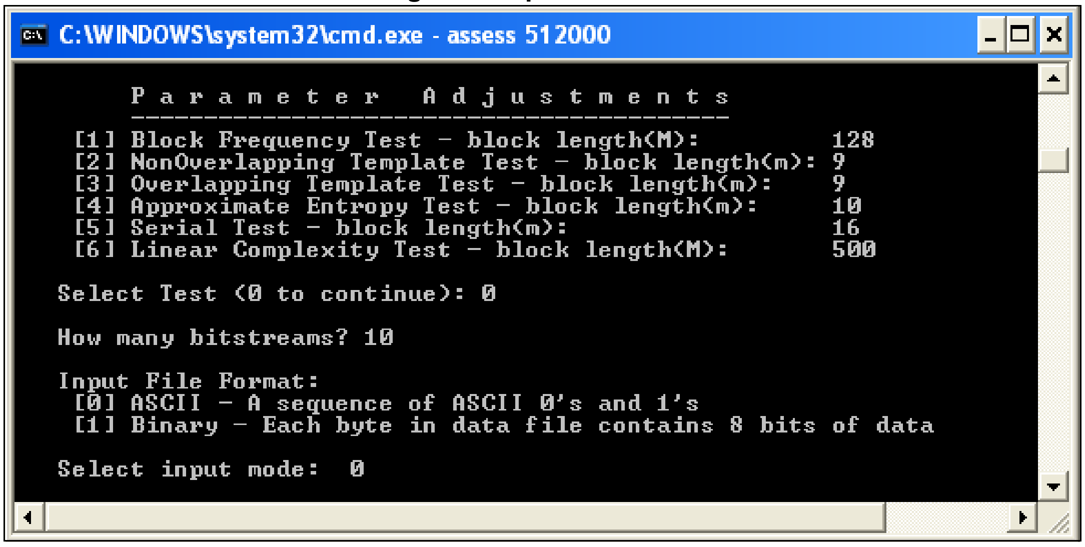
```
C:\WINDOWS\system32\cmd.exe - assess 512000

        P a r a m e t e r   A d j u s t m e n t s
        -----------------------------------------
    [1] Block Frequency Test - block length(M):         128
    [2] NonOverlapping Template Test - block length(m): 9
    [3] Overlapping Template Test - block length(m):    9
    [4] Approximate Entropy Test - block length(m):     10
    [5] Serial Test - block length(m):                  16
    [6] Linear Complexity Test - block length(M):       500

   Select Test (0 to continue): 0

   How many bitstreams? 10
```

The NIST statistical test suite requires to put the number of bitstreams; here, ‘10’ is entered.

You have selected 10 blocks of 512,000 bits which equal to 5,120,000 bits.

You must then specify whether the file consists of bits stored in ASCII format or hexadecimal strings stored in a binary format using the following screen.

Figure 8. Input file format

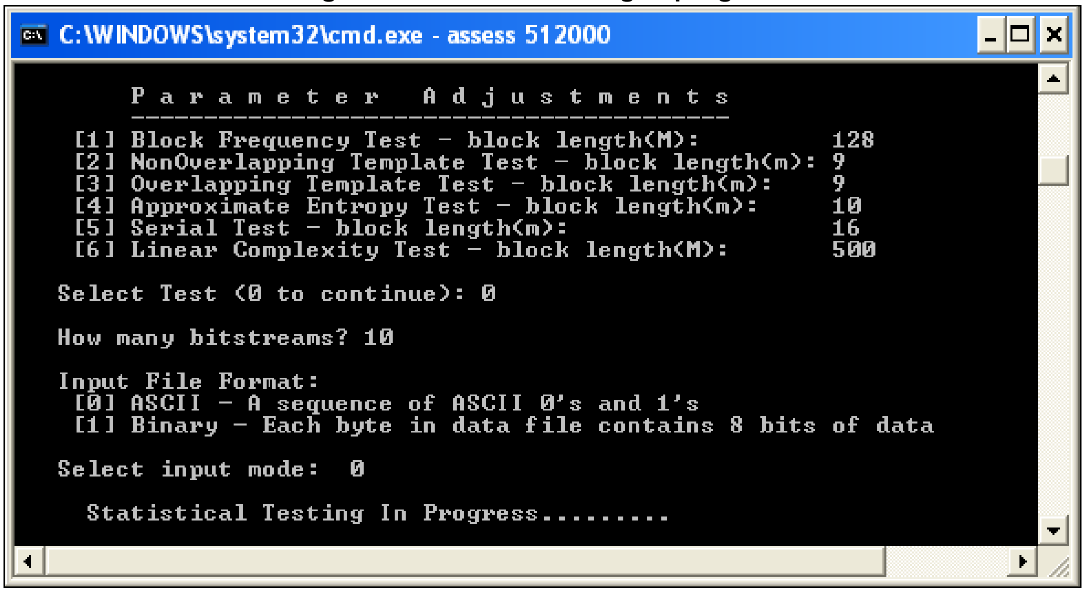


```
C:\WINDOWS\system32\cmd.exe - assess 512000

        P a r a m e t e r   A d j u s t m e n t s
        -----------------------------------------
    [1] Block Frequency Test - block length(M):         128
    [2] NonOverlapping Template Test - block length(m): 9
    [3] Overlapping Template Test - block length(m):    9
    [4] Approximate Entropy Test - block length(m):     10
    [5] Serial Test - block length(m):                  16
    [6] Linear Complexity Test - block length(M):       500

   Select Test (0 to continue): 0

   How many bitstreams? 10

   Input File Format:
    [0] ASCII - A sequence of ASCII 0's and 1's
    [1] Binary - Each byte in data file contains 8 bits of data

   Select input mode:  0
```

Value ‘0’ is selected because the file is in ASCII format.

After entering all necessary inputs, the NIST statistical test suite starts analyzing the input file.

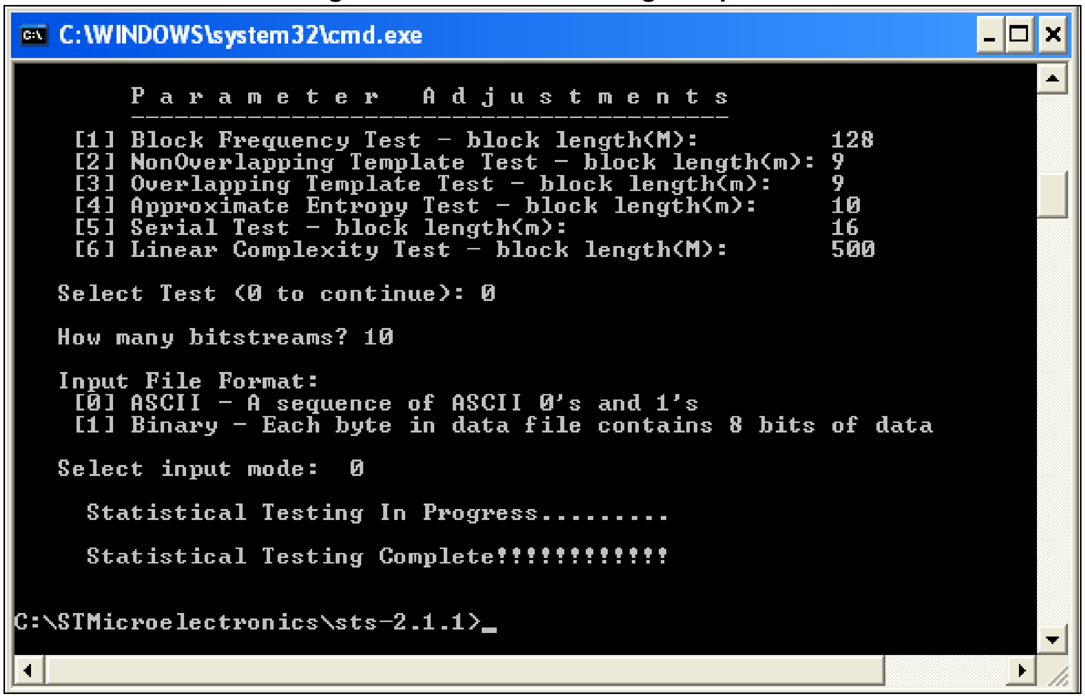
Figure 9. Statistical testing in progress



```
C:\WINDOWS\system32\cmd.exe - assess 512000

        P a r a m e t e r   A d j u s t m e n t s
        -----------------------------------------
    [1] Block Frequency Test - block length(M):         128
    [2] NonOverlapping Template Test - block length(m): 9
    [3] Overlapping Template Test - block length(m):    9
    [4] Approximate Entropy Test - block length(m):     10
    [5] Serial Test - block length(m):                  16
    [6] Linear Complexity Test - block length(M):       500

   Select Test (0 to continue): 0

   How many bitstreams? 10

   Input File Format:
    [0] ASCII - A sequence of ASCII 0's and 1's
    [1] Binary - Each byte in data file contains 8 bits of data

   Select input mode:  0

     Statistical Testing In Progress.........
```

When the testing process is complete, as shown in *Figure 10*, the statistical test results can be found in sts-2.1.1\experiments\AlgorithmTesting.

Figure 10. Statistical testing complete



C:\WINDOWS\system32\cmd.exe

```
        P a r a m e t e r   A d j u s t m e n t s
        -----------------------------------------
    [1] Block Frequency Test - block length(M):         128
    [2] NonOverlapping Template Test - block length(m): 9
    [3] Overlapping Template Test - block length(m):    9
    [4] Approximate Entropy Test - block length(m):     10
    [5] Serial Test - block length(m):                  16
    [6] Linear Complexity Test - block length(M):       500

   Select Test (0 to continue): 0

   How many bitstreams? 10

   Input File Format:
    [0] ASCII - A sequence of ASCII 0's and 1's
    [1] Binary - Each byte in data file contains 8 bits of data

   Select input mode:  0

     Statistical Testing In Progress.........

     Statistical Testing Complete!!!!!!!!!!!!

C:\STMicroelectronics\sts-2.1.1>_
```

3.2.3 Third step: tests report

The NIST statistical tests provide an analytical routine to facilitate the interpretation of the results. A file named finalAnalysisReport is generated when the statistical testing is complete and saved under sts2.1.1\experiments\AlgorithmTesting. The report contains a summary of experimental results of 15 tests, as described in *Appendix A*.

The NST statistical tests also provide a detailed report for each test, which is saved under sts-2.1.1\experiments\AlgorithmTesting\<Test suite name>.

Note: You can find two examples per series of complete NIST statistical test suite output reports under "NIST_Test_Suite_OutputExample" .

1. Example of an *Ascii_File_Format*. This example has 2 folders:
   – ***Input_File*** which contains the random number generator saved with the ascii format.
   – ***Final_Analysis_Report*** which contains the complete NIST statistical test suite output report based on this input file, the summary of experimental results and the report of each test.
2. *Example of a Binary_File_Format.* This example has 2 folders:
   – ***Input_File*** which contains the random number generator saved with the binary format.
   – ***Final_Analysis_Report*** which contains the complete NIST statistical test suite output report based on this input file, the summary of experimental results and the report of each test.

4 Conclusion

This application note describes the main guidelines and steps to verify the randomness of numbers generated by the STM32 microcontrollers random number generator peripheral using NIST statistical test suite SP800-22rev1a, April 2010.

15 tests of NIST statistical test suite are passed in accordance with the minimum pass rate for each statistical test run on each STM32 series support the RNG peripheral.

Appendix A Additional information

The results are represented as a table with p rows and q columns.

- The number of rows, p, corresponds to the number of statistical tests applied.
- The number of columns, q = 13, is distributed as follows:
  - columns 1-10 correspond to the frequency of P-values10,
  - column 11 is the P-value that arises via the application of a chi-square test11,
  - column 12 is the proportion of binary sequences that passed,
  - column 13 is the corresponding statistical test.

An example is shown in *Table 3*. For more details, refer to finalAnalysisReport file under sts-2.1.1\experiments\AlgorithmTesting).

Table 3. Example

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING
SEQUENCES
------------------------------------------------------------------------------
   generator is <data/ascii.bin>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
  0   1   2   1   2   1   1   1   0   1  0.911413    10/10     Frequency
  1   1   0   1   3   0   2   1   1   0  0.534146    10/10     BlockFrequency
  0   1   3   3   0   1   0   2   0   0  0.122325    10/10     CumulativeSums
  1   1   3   1   0   1   1   1   0   1  0.739918    10/10     CumulativeSums
  2   0   2   2   1   1   0   1   0   1  0.739918    10/10     Runs
  1   0   1   1   0   3   1   1   0   2  0.534146     9/10     LongestRun
  1   2   1   0   2   1   1   0   0   2  0.739918    10/10     Rank
  3   0   1   2   1   1   0   1   0   1  0.534146     9/10     FFT
  1   1   1   0   0   2   1   2   0   2  0.739918    10/10     NonOverlappingTemplate
  1   1   0   0   1   1   1   3   0   2  0.534146    10/10     NonOverlappingTemplate
  0   2   1   0   4   0   2   0   0   1  0.066882    10/10     NonOverlappingTemplate
  0   0   0   1   1   3   0   2   1   2  0.350485    10/10     NonOverlappingTemplate
  0   1   2   2   1   1   1   2   0   0  0.739918    10/10     NonOverlappingTemplate
  2   2   1   0   2   0   1   1   1   0  0.739918    10/10     NonOverlappingTemplate
  1   0   2   2   1   1   1   0   1   1  0.911413    10/10     NonOverlappingTemplate
  0   0   1   1   0   0   2   3   1   2  0.350485    10/10     NonOverlappingTemplate
  1  1  1  0  4  0  0  0  2  1 0.122325   10/10     NonOverlappingTemplate
  1  0  1  0  2  3  0  0  2  1 0.350485   10/10     NonOverlappingTemplate
  0  1  0  2  2  2  2  0  1  0 0.534146   10/10     NonOverlappingTemplate
  1  0  1  1  2  0  2  1  0  2 0.739918   10/10     NonOverlappingTemplate
  0  1  2  0  0  0  3  2  1  1 0.350485   10/10     NonOverlappingTemplate
  2  1  1  1  2  2  0  0  1  0 0.739918   10/10     NonOverlappingTemplate
  1  1  2  3  1  0  1  0  1  0 0.534146   10/10     NonOverlappingTemplate
  1  3  1  3  1  1  0  0  0  0 0.213309   10/10     NonOverlappingTemplate
  1  1  2  0  2  2  0  0  1  1 0.739918   10/10     NonOverlappingTemplate
  2  1  1  1  1  0  1  1  1  1 0.991468   10/10     NonOverlappingTemplate
  2  0  0  0  1  4  0  1  0  2 0.066882   10/10     NonOverlappingTemplate
  2  0  0  1  2  2  1  0  0  2 0.534146   10/10     NonOverlappingTemplate
  0  2  1  1  0  3  1  1  0  1 0.534146   10/10     NonOverlappingTemplate
  0  2  1  1  2  1  1  0  1  1 0.911413   10/10     NonOverlappingTemplate
  0  0  1  0  4  0  2  1  0  2 0.066882   10/10     NonOverlappingTemplate
  1  2  1  2  0  2  2  0  0  0 0.534146   10/10     NonOverlappingTemplate
  3  0  1  1  0  1  3  1  0  0 0.213309   10/10     NonOverlappingTemplate
  2  0  1  0  1  0  1  4  0  1 0.122325   10/10     NonOverlappingTemplate
  0  1  1  0  1  1  0  1  4  1 0.213309   10/10     NonOverlappingTemplate
  3  2  0  0  1  0  3  1  0  0 0.122325   10/10     NonOverlappingTemplate
  0  3  0  2  1  1  1  1  0  1 0.534146   10/10     NonOverlappingTemplate
  0  1  1  4  0  0  1  0  1  2 0.122325   10/10     NonOverlappingTemplate
  0  0  2  0  1  2  1  4  0  0 0.066882   10/10     NonOverlappingTemplate
  2  1  2  0  0  1  2  1  1  0 0.739918   10/10     NonOverlappingTemplate
  1  1  2  2  0  0  0  1  2  1 0.739918   10/10     NonOverlappingTemplate
  1  0  1  3  1  0  1  2  1  0 0.534146   10/10     NonOverlappingTemplate
  1  3  2  1  1  0  0  1  0  1 0.534146   10/10     NonOverlappingTemplate
  2  0  0  0  0  0  2  1  1  4 0.066882   10/10     NonOverlappingTemplate
  0  1  1  0  1  0  1  2  1  3 0.534146   10/10     NonOverlappingTemplate
  1  2  0  0  2  2  0  1  1  1 0.739918   10/10     NonOverlappingTemplate
  1  3  0  2  0  0  0  2  2  0 0.213309    9/10     NonOverlappingTemplate
  3  1  0  0  0  2  1  0  3  0 0.122325   10/10     NonOverlappingTemplate
  1  0  2  0  3  0  1  1  1  1 0.534146   10/10     NonOverlappingTemplate
  0  1  1  1  0  2  2  2  0  1 0.739918   10/10     NonOverlappingTemplate
  2  1  0  0  3  1  1  0  2  0 0.350485   10/10     NonOverlappingTemplate
  0  0  1  2  1  4  1  1  0  0 0.122325  10/10   NonOverlappingTemplate
  1  1  0  1  1  3  2  0  1  0 0.534146  10/10   NonOverlappingTemplate
  2  1  0  2  1  0  1  0  1  2 0.739918  10/10   NonOverlappingTemplate
  0  0  2  3  1  1  0  1  1  1 0.534146  10/10   NonOverlappingTemplate
  1  1  1  0  1  3  1  0  1  1 0.739918   9/10   NonOverlappingTemplate
  0  1  4  0  0  1  2  1  1  0 0.122325  10/10   NonOverlappingTemplate
  1  3  0  0  0  0  1  1  2  2 0.350485  10/10   NonOverlappingTemplate
  2  1  1  1  1  0  1  0  2  1 0.911413  10/10   NonOverlappingTemplate
  2  1  2  0  1  1  0  1  1  1 0.911413   9/10   NonOverlappingTemplate
  1  0  1  0  2  2  1  2  0  1 0.739918  10/10   NonOverlappingTemplate
  0  1  3  1  1  1  1  2  0  0 0.534146  10/10   NonOverlappingTemplate
  4  0  0  0  2  1  2  0  0  1 0.066882  10/10   NonOverlappingTemplate
  0  0  0  0  2  0  1  2  2  3 0.213309  10/10   NonOverlappingTemplate
  2  2  0  0  3  1  0  1  0  1 0.350485  10/10   NonOverlappingTemplate
  2  0  2  0  1  0  2  2  0  1 0.534146   9/10   NonOverlappingTemplate
  1  0  1  0  0  1  3  0  3  1 0.213309   9/10   NonOverlappingTemplate
  1  2  1  1  0  1  3  0  0  1 0.534146  10/10   NonOverlappingTemplate
  0  1  2  0  1  0  1  2  0  3 0.350485  10/10   NonOverlappingTemplate
  2  0  2  0  0  0  2  0  3  1 0.213309  10/10   NonOverlappingTemplate
  2  1  2  1  0  1  0  2  0  1 0.739918  10/10   NonOverlappingTemplate
  1  0  1  0  4  0  0  1  2  1 0.122325  10/10   NonOverlappingTemplate
  0  0  0  2  1  1  3  2  1  0 0.350485  10/10   NonOverlappingTemplate
  1  3  1  0  2  0  1  0  0  2 0.350485   9/10   NonOverlappingTemplate
  0  0  1  0  0  3  1  2  3  0 0.122325  10/10   NonOverlappingTemplate
  0  1  0  2  1  0  1  2  3  0 0.350485  10/10   NonOverlappingTemplate
  0  0  0  2  4  2  1  1  0  0 0.066882  10/10   NonOverlappingTemplate
  1  0  0  2  0  1  2  1  2  1 0.739918   9/10   NonOverlappingTemplate
  0  0  2  0  1  2  0  0  1  4 0.066882  10/10   NonOverlappingTemplate
  1  1  0  1  1  1  2  0  2  1 0.911413  10/10   NonOverlappingTemplate
  0  0  0  5  2  0  1  2  0  0 0.004301  10/10   NonOverlappingTemplate
  2  1  0  0  1  1  0  1  3  1 0.534146  10/10   NonOverlappingTemplate
  1  1  1  0  0  2  1  2  0  2 0.739918  10/10   NonOverlappingTemplate
  2  0  2  1  0  1  2  0  1  1 0.739918  10/10   NonOverlappingTemplate
  0  1  3  0  1  0  1  2  1  1 0.534146  10/10   NonOverlappingTemplate
  1  1  1  1  1  1  1  2  1  0 0.991468  10/10   NonOverlappingTemplate
```

```
0  0  3  2  2  2  0  1  0  0 0.213309   10/10    NonOverlappingTemplate
2  1  2  1  1  1  0  0  1  1 0.911413   10/10    NonOverlappingTemplate
0  1  1  1  0  1  2  1  2  1 0.911413   10/10    NonOverlappingTemplate
1  0  0  2  3  2  0  0  1  1 0.350485   10/10    NonOverlappingTemplate
2  2  1  0  0  1  1  3  0  0 0.350485   10/10    NonOverlappingTemplate
2  0  2  2  0  0  0  1  0  3 0.213309   10/10    NonOverlappingTemplate
1  1  1  2  0  0  2  2  0  1 0.739918   10/10    NonOverlappingTemplate
3  2  1  0  1  0  0  0  1  2 0.350485    9/10    NonOverlappingTemplate
2  0  1  1  2  1  1  0  1  1 0.911413   10/10    NonOverlappingTemplate
0  2  0  0  1  0  1  3  2  1 0.350485   10/10    NonOverlappingTemplate
1  0  3  1  0  0  1  2  1  1 0.534146    9/10    NonOverlappingTemplate
1  2  0  2  0  4  0  0  0  1 0.066882   10/10    NonOverlappingTemplate
1  0  1  0  2  0  1  1  4  0 0.122325    9/10    NonOverlappingTemplate
1  1  1  1  0  0  3  2  1  0 0.534146   10/10    NonOverlappingTemplate
2  0  1  0  0  1  1  1  1  3 0.534146    9/10    NonOverlappingTemplate
3  0  1  1  1  1  1  1  1  0 0.739918   10/10    NonOverlappingTemplate
1  0  0  1  1  1  1  3  0  2 0.534146   10/10    NonOverlappingTemplate
3  3  1  1  0  0  0  1  1  0 0.213309   10/10    NonOverlappingTemplate
1  0  0  1  1  0  1  1  4  1 0.213309   10/10    NonOverlappingTemplate
0  2  2  3  1  1  0  1  0  0 0.350485   10/10    NonOverlappingTemplate
2  1  0  3  0  0  2  0  1  1 0.350485   10/10    NonOverlappingTemplate
1  0  0  3  0  1  1  2  0  2 0.350485    9/10    NonOverlappingTemplate
1  2  0  1  0  0  3  2  0  1 0.350485   10/10    NonOverlappingTemplate
2  1  1  1  2  0  1  1  1  0 0.911413   10/10    NonOverlappingTemplate
0  0  1  2  1  1  1  3  0  1 0.534146   10/10    NonOverlappingTemplate
0  0  2  1  3  0  3  0  0  1 0.122325   10/10    NonOverlappingTemplate
0  2  2  1  0  0  2  1  1  1 0.739918   10/10    NonOverlappingTemplate
1  2  0  2  2  1  0  0  1  1 0.739918   10/10    NonOverlappingTemplate
1  1  2  0  2  1  3  0  0  0 0.350485   10/10    NonOverlappingTemplate
2  1  0  1  1  1  3  1  0  0 0.534146    9/10    NonOverlappingTemplate
2  4  0  1  1  1  0  0  0  1 0.122325   10/10    NonOverlappingTemplate
0  0  1  0  2  2  2  2  0  1 0.534146   10/10    NonOverlappingTemplate
2  1  2  0  1  1  1  1  0  1 0.911413   10/10    NonOverlappingTemplate
1  1  1  4  1  1  1  0  0  0 0.213309   10/10    NonOverlappingTemplate
1  0  0  2  0  1  2  1  3  0 0.350485   10/10    NonOverlappingTemplate
```

```
4  0  2  0  2  0  1  0  0  1 0.066882   10/10    NonOverlappingTemplate
0  1  2  2  0  0  0  2  2  1 0.534146   10/10    NonOverlappingTemplate
1  0  2  0  1  1  0  3  1  1 0.534146   10/10    NonOverlappingTemplate
4  1  1  0  0  0  0  1  1  2 0.122325   10/10    NonOverlappingTemplate
2  1  0  0  1  1  2  1  1  1 0.911413   10/10    NonOverlappingTemplate
0  1  1  1  0  0  1  4  1  1 0.213309   10/10    NonOverlappingTemplate
1  2  2  0  0  5  0  0  0  0 0.004301   10/10    NonOverlappingTemplate
0  2  2  2  0  2  0  1  1  0 0.534146   10/10    NonOverlappingTemplate
3  1  1  0  0  2  0  1  1  1 0.534146   10/10    NonOverlappingTemplate
2  1  0  1  1  1  0  2  1  1 0.911413   10/10    NonOverlappingTemplate
1  1  2  1  0  1  1  0  1  2 0.911413   10/10    NonOverlappingTemplate
0  1  2  3  0  2  0  2  0  0 0.213309   10/10    NonOverlappingTemplate
0  1  0  1  3  1  2  0  0  2 0.350485   10/10    NonOverlappingTemplate
3  0  0  3  1  0  1  0  1  1 0.213309    9/10    NonOverlappingTemplate
0  1  0  2  0  2  1  0  3  1 0.350485   10/10    NonOverlappingTemplate
1  1  3  2  0  1  0  1  0  1 0.534146   10/10    NonOverlappingTemplate
0  1  0  3  0  0  0  1  0  5 0.002043   10/10    NonOverlappingTemplate
2  1  0  1  0  0  3  1  1  1 0.534146   10/10    NonOverlappingTemplate
2  0  0  0  2  1  1  0  3  1 0.350485   10/10    NonOverlappingTemplate
0  2  1  1  2  2  0  0  0  2 0.534146   10/10    NonOverlappingTemplate
0  0  1  0  3  0  1  1  1  3 0.213309   10/10    NonOverlappingTemplate
1  0  2  2  0  0  0  2  3  0 0.213309   10/10    NonOverlappingTemplate
1  2  2  1  1  1  0  1  0  1 0.911413   10/10    NonOverlappingTemplate
1  1  1  1  4  0  0  1  0  1 0.213309    9/10    NonOverlappingTemplate
0  1  1  2  2  1  0  0  2  1 0.739918   10/10    NonOverlappingTemplate
2  1  0  0  2  0  1  2  1  1 0.739918   10/10    NonOverlappingTemplate
1  0  1  1  0  0  2  3  1  1 0.534146   10/10    NonOverlappingTemplate
2  1  0  0  1  2  2  0  0  2 0.534146   10/10    NonOverlappingTemplate
0  0  2  0  1  0  3  3  0  1 0.122325   10/10    NonOverlappingTemplate
1  1  1  2  0  3  1  0  0  1 0.534146   10/10    NonOverlappingTemplate
1  0  1  1  1  2  2  1  0  1 0.911413   10/10    NonOverlappingTemplate
0  1  2  0  1  1  2  0  1  2 0.739918   10/10    NonOverlappingTemplate
1  2  0  0  1  3  1  1  0  1 0.534146   10/10    NonOverlappingTemplate
2  1  1  0  1  0  0  2  0  3 0.350485   10/10    NonOverlappingTemplate
2  1  0  0  1  1  0  1  3  1 0.534146   10/10    NonOverlappingTemplate
```

```
  2   0   1   0   1   2   1   0   2   1  0.739918     10/10      OverlappingTemplate
  1   0   2   1   0   2   2   1   1   0  0.739918     10/10      Universal
  1   1   0   0   2   0   2   3   1   0  0.350485     10/10      ApproximateEntropy
  0   1   1   1   1   0   0   0   1   0     ----       5/5       RandomExcursions
  1   1   0   0   2   0   0   0   0   1     ----       5/5       RandomExcursions
  0   1   1   1   0   0   0   0   1   1     ----       5/5       RandomExcursions
  0   0   0   0   0   1   1   0   2   1     ----       5/5       RandomExcursions
  1   0   0   0   3   0   0   0   1   0     ----       5/5       RandomExcursions
  0   0   0   1   1   0   0   1   1   1     ----       5/5       RandomExcursions
  1   0   1   1   0   2   0   0   0   0     ----       5/5       RandomExcursions
  1   0   0   0   1   1   1   0   1   0     ----       5/5       RandomExcursions
  2   1   0   1   1   0   0   0   0   0     ----       5/5       RandomExcursionsVariant
  2   1   0   0   1   1   0   0   0   0     ----       5/5       RandomExcursionsVariant
  1   1   0   2   1   0   0   0   0   0     ----       5/5       RandomExcursionsVariant
  1   2   0   1   1   0   0   0   0   0     ----       5/5       RandomExcursionsVariant
  1   1   1   1   0   0   0   1   0   0     ----       5/5       RandomExcursionsVariant
  1   1   0   1   1   0   0   0   0   1     ----       5/5       RandomExcursionsVariant
  0   1   0   2   1   0   0   0   0   1     ----       5/5       RandomExcursionsVariant
  0   0   0   1   0   1   0   3   0   0     ----       5/5       RandomExcursionsVariant
  0   0   0   0   0   0   2   1   1   1     ----       5/5       RandomExcursionsVariant
  0   0   1   0   0   0   1   1   1   1     ----       5/5       RandomExcursionsVariant
  0   0   0   1   0   0   2   0   2   0     ----       5/5       RandomExcursionsVariant
  0   1   0   0   1   1   1   1   0   0     ----       5/5       RandomExcursionsVariant
  1   0   0   2   0   1   1   0   0   0     ----       5/5       RandomExcursionsVariant
  1   0   0   0   2   1   0   0   0   1     ----       5/5       RandomExcursionsVariant
  0   0   0   1   1   0   1   1   1   0     ----       5/5       RandomExcursionsVariant
  0   0   0   0   2   0   2   0   0   1     ----       5/5       RandomExcursionsVariant
  0   0   1   0   1   2   1   0   0   0     ----       5/5       RandomExcursionsVariant
  0   0   1   0   0   2   2   0   0   0     ----       5/5       RandomExcursionsVariant
  1   1   0   0   0   2   3   0   2   1  0.350485     10/10      Serial
  0   2   1   0   3   1   0   1   1   1  0.534146     10/10      Serial
  2   1   1   1   0   1   1   3   0   0  0.534146     10/10      LinearComplexity
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test, with the exception of the random excursion
(variant) test, is approximately 8 for a sample size of 10 binary sequences.
```

The minimum pass rate for the random excursion (variant) test is approximately 4 for a sample size of 5 binary sequences.

For further guidelines, construct a probability table using the MAPLE program provided in the addendum section of the documentation.

Revision history

Table 4. Document revision history

Date	Revision	Changes
13-May-2013	1	Initial release.
22-Jun-2016	2	Updated *Section : Introduction*. Updated *Section 1: STM32 microcontrollers random number generator*. Added *Figure 2: STM32 lines embedding the RNG hardware peripheral*. Updated *Figure 1: Block diagram*. Updated *Section 3.1: Firmware description*. Updated *Section 3.2.1: First step: random number generator*. Updated *Section 3.2.2: Second step: NIST statistical test*. Updated *Section 4: Conclusion*.