Application note

# STM32 microcontroller random number generation validation using the NIST statistical test suite

## Introduction

Many standards contain requirements and references for the extraction, validation, and use of random number generators (RNGs), in order to verify that their output is indeed random.

This application note provides guidelines for verification of the randomness of numbers generated by the RNG peripheral embedded in a selection of STM32 microcontrollers (MCUs). These MCUs are listed in Table 1. The verification is based either on the NIST (national institute of standards and technology) SP 800-22rev1a (April 2010), or SP 800-90b (January 2018) statistical test suite (STS).

This document is structured as follows:

- A general introduction to the STM32 microcontroller random number generator. See Section 1 STM32 MCU RNG.
- The NIST SP800-22rev1a test suite. See Section 2 NIST SP800-22rev1a test suite.
- The steps needed to run NIST SP800-22rev1a test and analysis. See Section 3 NIST SP800-22rev1a test suite running and analyzing.
- The NIST SP800-90b test suite See Section 4 NIST SP800-90b test suite.
- The steps needed to run NIST SP800-90b test and analysis. See Section 3 NIST SP800-22rev1a test suite running and analyzing.

Table 1. **Applicable products**

| Type | | Products | |
|---|---|---|---|
| | | Can be checked with SP800-22rev1a | Can be checked with SP800-90b |
| Microcontrollers | Series | STM32F2 Series, STM32F4 Series, STM32F7 Series, STM32L0 Series, STM32L4 Series | STM32L5 Series, STM32U5 Series, STM32MP1 Series |
| | Lines | STM32H742, STM32H743/753, STM32H745/755, STM32H747/757 lines, STM32H750 Value line, STM32L4R5/S5, STM32L4R7/S7, STM32L4R9/S9 lines | STM32H7A3/7B3 line, STM32H7B0 Value line, STM32H723/733, STM32H725/735, STM32H730 Value line, STM32L4P5/Q5 line |

# 1 STM32 MCU RNG

## 1.1 RNG overview

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

There are two basic classes of random number generators:

- Deterministic RNG or pseudo RNG (PRNG)
  A deterministic RNG consists 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.
- Non-deterministic RNG or true RNG (TRNG)
  A non-deterministic RNG produces randomness that depends on some unpredictable physical source (the entropy source) outside of any human control.

The RNG hardware peripheral implemented in some STM32 MCUs is a true random number generator.

## 1.2 STM32 MCU implementation description

The table below lists the STM32 Arm® Cortex® core-based MCUs that embed the RNG peripheral.

*Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.*

Table 2. **STM32 lines embedding the RNG hardware peripheral**

| Series | STM32 lines |
|---|---|
| STM32F2 Series | STM32F2x5, STM32F2x7 |
| STM32F4 Series | STM32F405/415, STM32F407/417, STM32F410, STM32F427/437, STM32F429/439, STM32F469/479 |
| STM32F7 Series | STM32F7x5, STM32F7x6 |
| STM32L0 Series | STM32L05x, STM32L06x, STM32L072/073 |
| STM32L4 Series | STM32L4x6 |
| STM32L4+ Series | All lines |
| STM32H7 Series | STM32H723/733, STM32H725/735, STM32H730 Value line, STM32H742, STM32H743/753, STM32H745/755, STM32H747/757, STM32H750 Value line, STM32H7A3/7B3, STM32H7B0 Value line |
| STM32L5 Series | STM32L5x2 |
| STM32U5 Series | STM32U575, STM32U585 |
| STM32G0 Series | All lines |
| STM32G4 | All lines |
| STM32MP1 Series | All lines |

The true RNG implemented in the STM32 MCUs is based on an analog circuit. This circuit generates a continuous analog noise that is used in the RNG processing 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, the RNG dedicated clock can be reduced using the divider inside the RNG peripheral.

For more details about the RNG peripherals, refer to the STM32 reference manuals.

The figure below shows a simplified view of a true RNG in STM32 microcontrollers.

Figure 1. **STM32 true RNG block diagram of the products verified with the SP800-22rev1a**

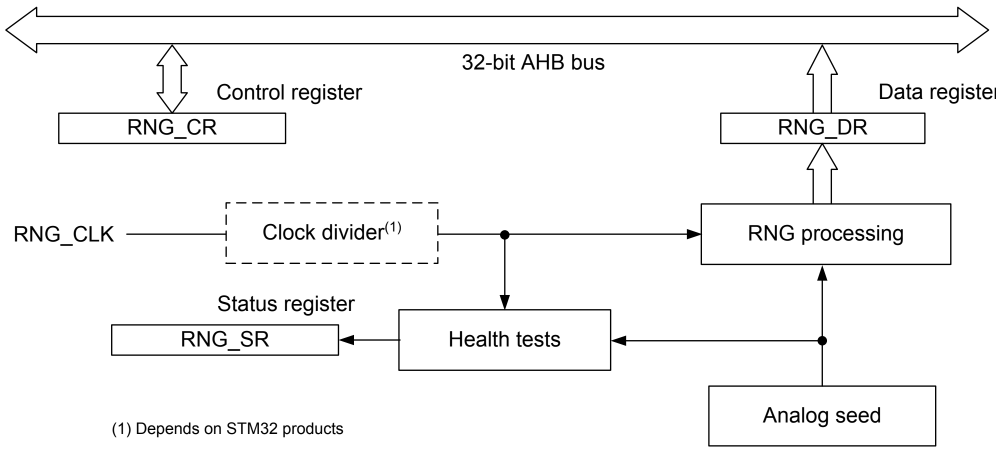
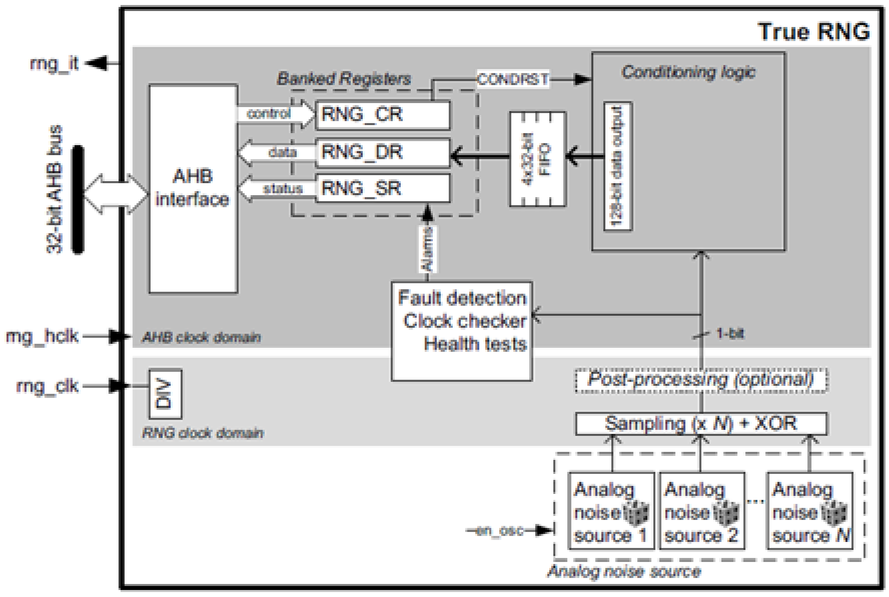


Figure 2. **STM32 true RNG block diagram of the products verified with the SP800-90b**



# 2 NIST SP800-22rev1a test suite

## 2.1 NIST SP800 22rev1a overview

The NIST SP800-22rev1a statistical test suite is used to probe the quality of RNGs for cryptographic applications. A comprehensive description of the suite is presented in the NIST document entitled *A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications*.

## 2.2 NIST SP800-22rev1a test suite description

The NIST SP800-22rev1a statistical test suite “sts-2.1.1” is a software package developed by NSIT that can be downloaded from the NIST web site (search for download the NIST Statistical Test Suite at csrc.nist.gov).

The source code has been written in ANSI C. The NIST statistical test suite consists of 15 tests that verify the randomness of a binary sequence. These tests focus on various types of non-randomness that can exist in a sequence.

These test 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 a M-bit block is approximately M/2, as 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
- Test of linearity
  - Binary matrix rank test

    To assess the distribution of the rank for 32 x 32 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 2m m-bit blocks.

*Note: For m = 1, the serial test is equivalent to the frequency test.*

- – *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.
  - 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 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, that is a function of the testing sequence.

The test statistic is used to calculate a **Pvalue**, which is the probability that a perfect random number generator generated 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 SP800-22rev1a 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 SP800-22rev1a test suite side.

### 3.1.1 STM32MCU side

The firmware package is provided upon request. For more details, contact the local ST sales representative.

This program allows random numbers generation, using the STM32 RNG peripheral. It also retrieves these numbers on a workstation for testing with the NIST statistical test suite.

Each firmware program is used to generate 10 64-Kbyte blocks of random numbers. The output file contains 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 one of the followings:

- a sequence of ASCII 0's and 1's if the `FILE_ASCII_FORMAT` Private define is uncommented in the `main.c` file
- 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 via the `SendToWorkstation()` function in the `main.c` file.

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

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

### 3.1.2 On the NIST SP800-22rev1a 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 STM32 RNG peripheral.

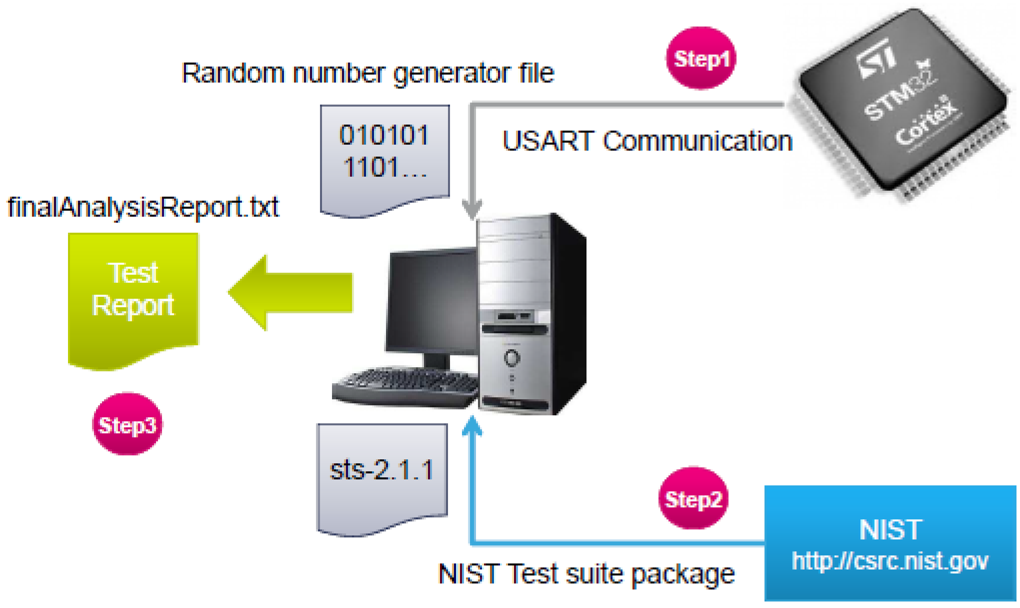
The generator file to be analyzed must be stored under the data folder `(sts-2.1.1\data)`.

For more details about how the NIST statistical tests work, refer to section ‘How to get started in the NIST document *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*.

## 3.2 NISTSP800-22rev1a test suite steps

The figure below describes the steps needed to verify the randomness of an output number generated by STM32 MCUs using the NIST statistical test suite package sts-2.1.1.

Figure 3. **Block diagram of deviation testing of a binary sequence from randomness based on NIST test suite**



### 3.2.1 Step1: random number generator

Connect the STM32 board to the workstation. Depending on the type of board, the connection is made as follows:

- via a null-modem female/female RS232 cable
- via a USB Type-A to Mini-B cable

The STM32 RNG is run via the UART firmware in order to generate a random number as described in Section 3.1.1 . Data are stored on the workstation using a terminal emulation application such as a PuTTY (free and open-source terminal emulator, serial console and network file transfer application).

### 3.2.2 Step 2: NIST statistical test

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

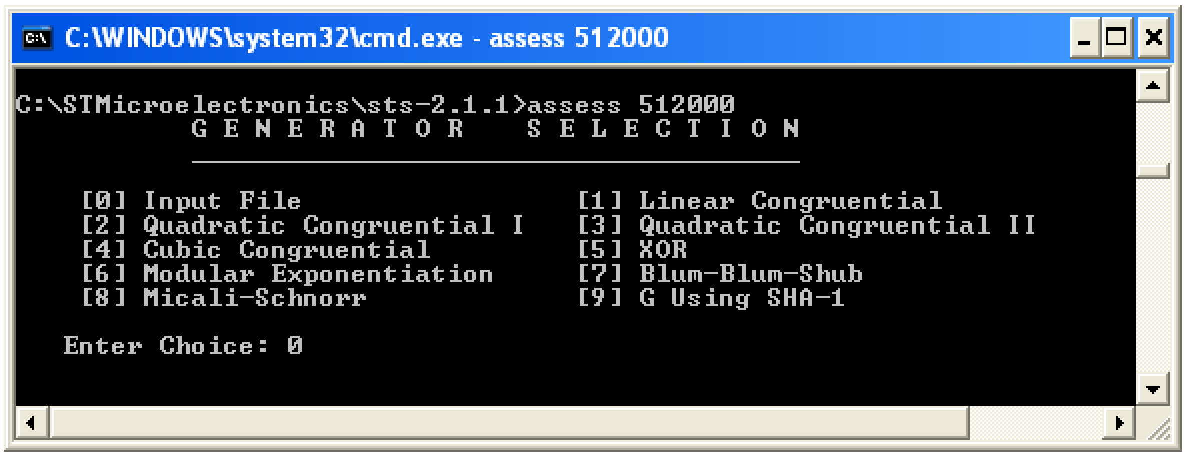
After running the NIST statistical test suite program, a series of menu prompts are displayed in order to select the data to be analyzed and the statistical tests to be applied.

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

The various steps are detailed as follows:

1. The first screen that appears is shown below.

Figure 4. Main sts-2.1.1 screen



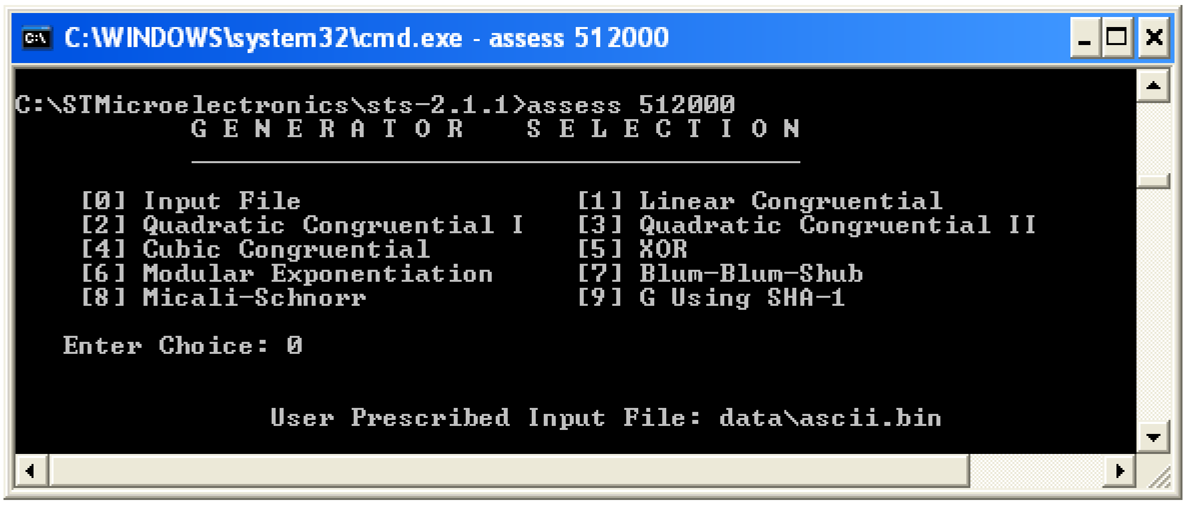
C:\WINDOWS\system32\cmd.exe - assess 512000

```
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.

2. The second screen is shown below.

Figure 5. File input screen



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

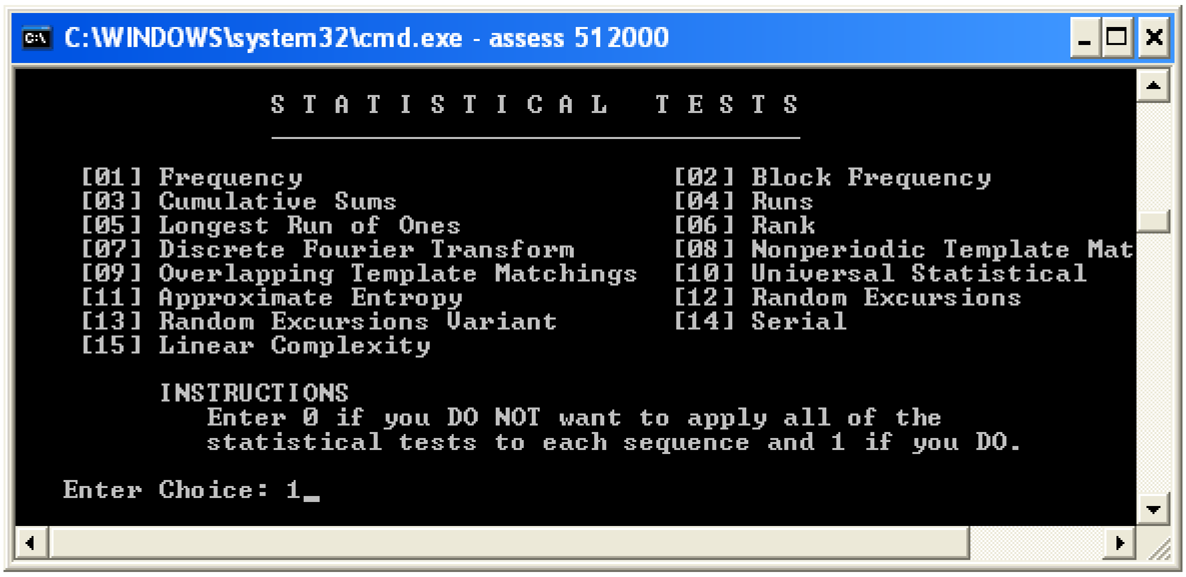
```
C:\STMicroelectronics\sts-2.1.1>assess 512000
           G E N E R A T O R    S E L E C T I O N
           ______________________________________

    [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 details an example with two files per series, generated with the STM32 RNG, with the following file formats as recommended by NIST:

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

3. The NIST statistical test suite displays 15 tests that can be run via the screen shown below.

Figure 6. Statistical test screen

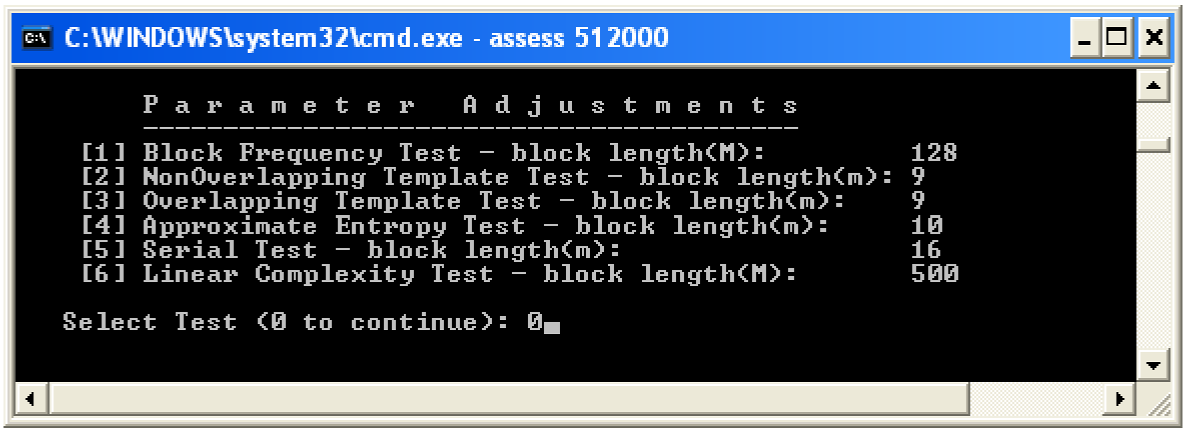


```
C:\WINDOWS\system32\cmd.exe - assess 512000

                         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.

4. The parameter adjustments ca be done in the screen shown below.

Figure 7. Parameter adjustment screen

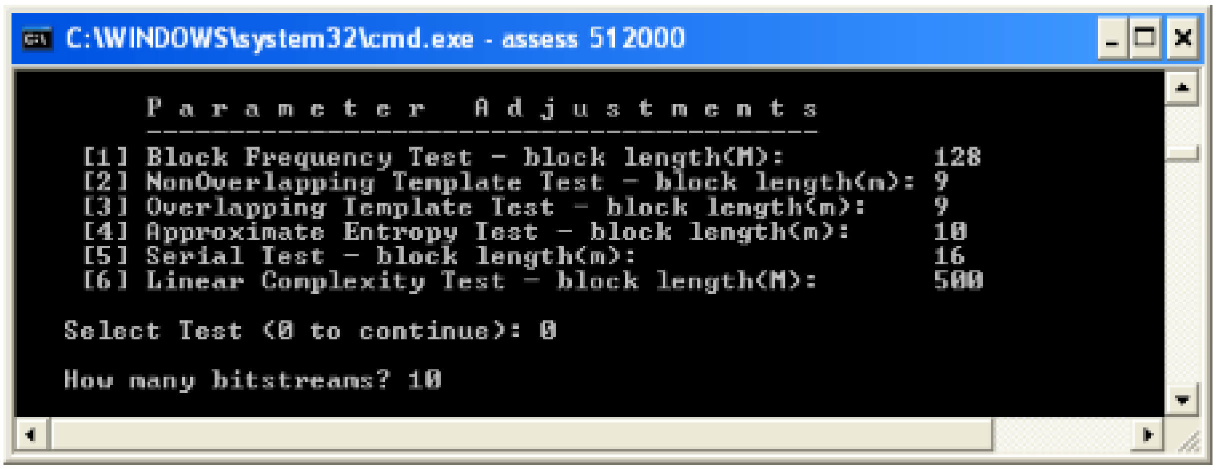


```
C:\WINDOWS\system32\cmd.exe - assess 512000

        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.

5. The user needs to provide the number of bitstreams.

Figure 8. **Bitstream input**

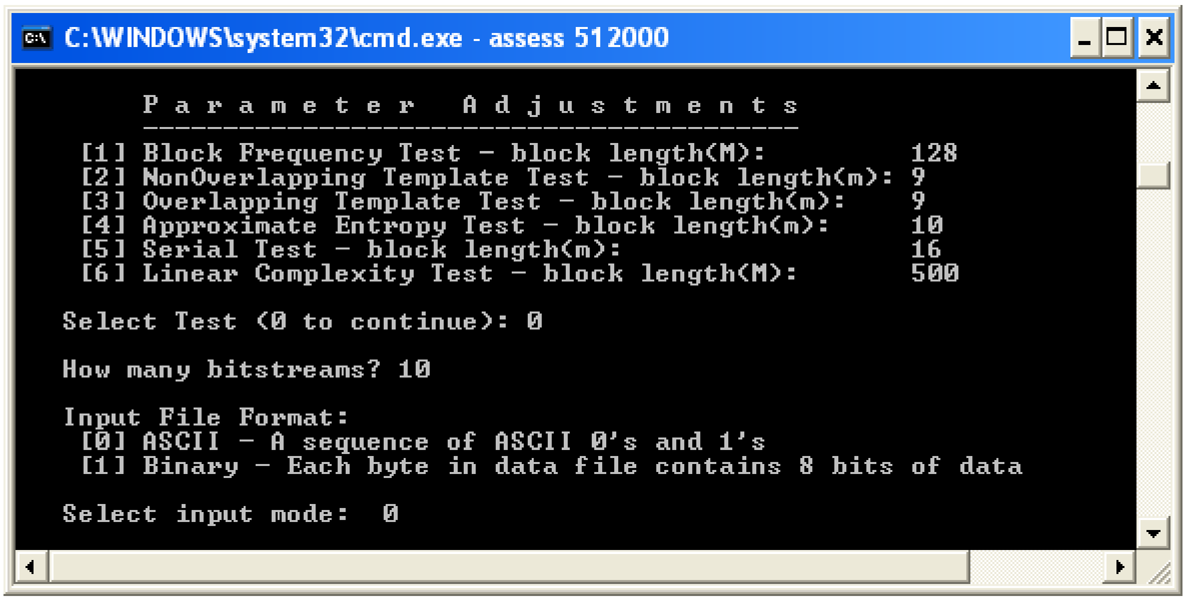


```
C:\WINDOWS\system32\cmd.exe - assess 512000

        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: 10 is entered for this example, meaning 10 blocks of 512 Kbits are selected (5,12 Mbits).

6. The user 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 9. **Input file format**

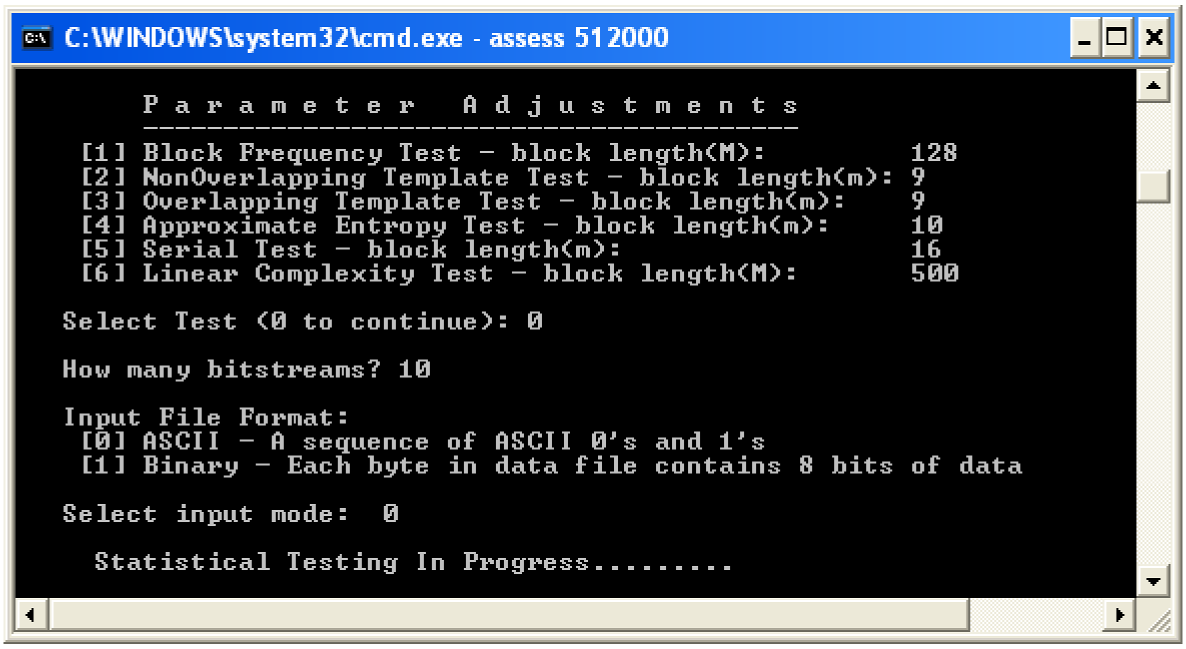


```
C:\WINDOWS\system32\cmd.exe - assess 512000

        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.

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

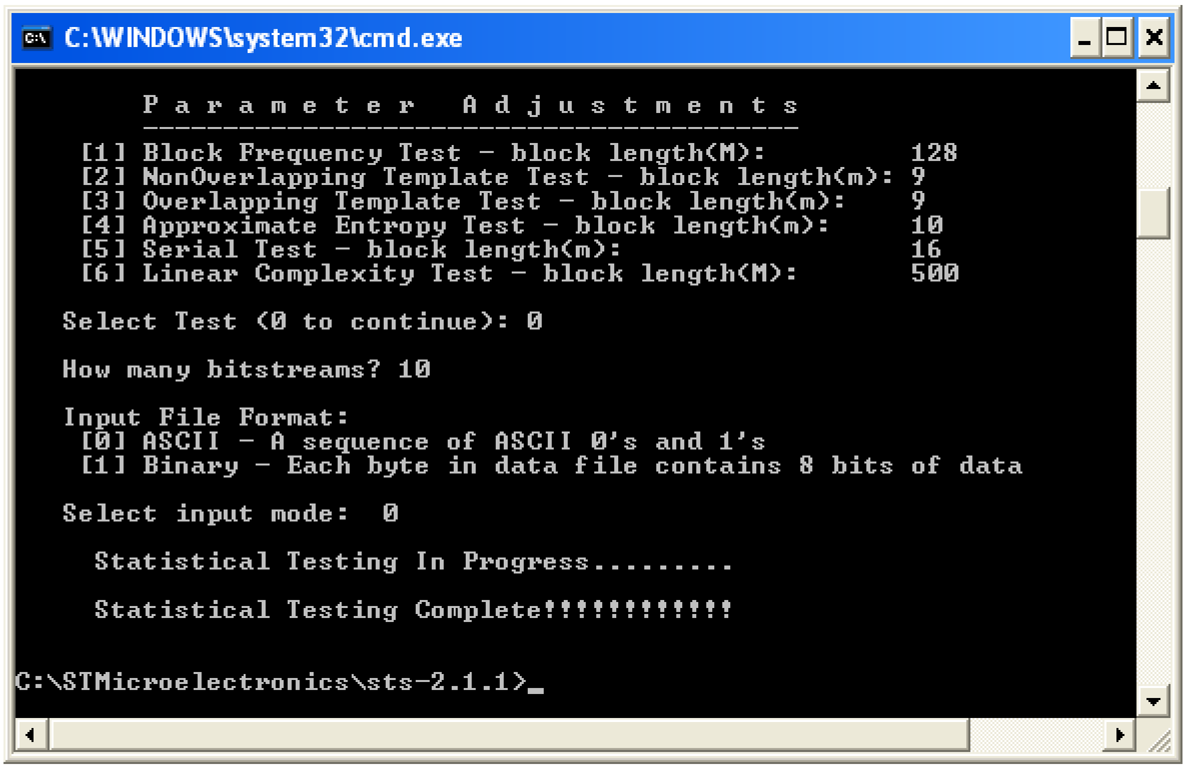
Figure 10. **Statistical testing in progress**



```
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.........
```

8. When the testing process is complete, the screen below appears.

Figure 11. **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>_
```

The statistical test results can be found in `sts-2.1.1\experiments\AlgorithmTesting`.

### 3.2.3 Step 3: test 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 (see NIST SP800-22rev1a statistical test suite).

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

The two following examples of complete NIST statistical test suite output reports are available under `NIST_Test_Suite_OutputExample`:

- example of an `Ascii_File_Format`, with two folders:
  - `Input_File`: contains the random number generator saved with the ascii format.
  - `Final_Analysis_Report`: 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.
- example of a `Binary_File_Format`, with two folders:
  - `Input_File`: contains the random number generator saved with the binary format.
  - `Final_Analysis_Report`: 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 NIST SP800-90b test suite

## 4.1 Cryptographic random bit overview

The cryptographic random bit generators (RBGs), also known as random number generators (RNGs), require a noise source that produces outputs with some level of unpredictability, expressed as min-entropy.

The specificity of the NIST SP800-90b statistical test suite is to probe the quality of random generators for cryptographic applications by its standardized means of estimating the quality of a source of entropy.

A comprehensive description of the suite is presented in the NIST document entitled *Recommendation for the Entropy Sources Used for Random Bit Generation*.

The NIST SP800-90b statistical test suite can be downloaded from the GitHub web site (https://github.com/usnistgov/SP800-90B_EntropyAssessment).

## 4.2 NIST SP800 90b test suite validation steps

The `SP800-90B_EntropyAssessment C++` package implements the min-entropy assessment methods included in *Special publication 800-90B*.

To ensure that all the requirements for the *NIST Special Publication (SP) 800-90* series recommendations are met it is necessary to validate the entropy source.

The general flow of entropy source validation testing is summarized in the steps below:

1. Data collection.
2. IID track or non IID track.
3. Initial entropy estimate.
4. Restart tests.
5. Entropy estimation for entropy sources using a conditioning component.
6. Additional noise sources.

The project is composed of two separate sections:

- -non-IID tests: provide an estimate for min-entropy, called $H_{original}$, for any data provided (the STM32 certifiable TRNG noise source is tested using non-IID tests)
- -ea-restart tests: check the relations between noise source samples generated after restarting the entropy source, and compare the results to the initial entropy estimate HI

*Note: $H_I$ = min ($H_{original}$, $H_{submitter}$) for binary sources where $H_{submitter}$ is the entropy estimate provided based on the submitter’s analysis of the noise source.*

# 5 Firmware description used for RNG NIST evaluation

## 5.1 STM32 MCU side

### 5.1.1 Data collection description

Random datasets are extracted from STM32-RNG peripheral entropy sources using the interrupt method (see Figure 12). The USART is used to send the generated raw data to the workstation to be tested with the NIST statistical test suite.

- 1,024,000 random samples are generated continuously from the RNG peripheral with the conditioning stage deactivated. Then it is shortened to keep exactly 1,000,000 samples. Sampled data are converted to binary in order to respect the format required by NIST SP800-90b tool suite. These binaries are used for the continuous tests.
- 1024 x 1000 random samples are extracted with restart at each 1000 bits extracted. They are then shortened to get 1000 x1000 samples as recommended by NIST. In restart extraction, 1000 unconditioned samples are collected directly from the noise source by disabling/enabling the RNG for each restart among the 1024 restarts required.

This random dataset can be extracted either with collecting continuous bits without disabling the RNG module or with restarting the RNG at each 1000 bits. The output file format is a sequence of hexadecimal digits in ASCII format which is converted to binary format without alteration of data using a small python script. This way, random bits are ready to be processed by NIST tests suite in *.bin dataset format.

An STM32 USART peripheral is configured to establish an USB communication between the STM32 board and a workstation.

### 5.1.2 STM32 firmware description

- CPU system modules, PLL clocks and peripherals used within the system are configured in the system initialization (see Figure 12).
- RNG Initialization function activates the RNG in the control register with the conditioning stage deactivated. A health test module is also activated by writing suitable values to the HTCR register. To be aligned with NIST recommended configuration in the reference manual. The RNG interrupt setting is also performed by this function.
- After a hardware reset, the main program waits for the fulfillment of a block by random bits before sending them to the workstation. When the number of blocks reaches its maximum value, the system execution is stopped and the 1 megabit random dataset is stored in a log file within the workstation.
- When the IE (Interrupt Enable) bit is set in the RNG control register, RNG triggers an interrupt each time new 32 random bits are available in the RNG data register (see Figure 13). A callback function is then executed (`HAL_RNG_ReadyDataCallback`), it extracts the 32 bits raw data and stores them in an STM32 internal memory. When a block of settable size is fulfilled and the maximum of block number is reached, two user flags become true accordingly (`NewBlockFlag` and `EndOfCollect`). Flags are supervised by the main function to control data transmission to the workstation.
- The activation of interrupts can trigger a second callback function (`HAL_RNG_ErrorCallback`) to run when the RNG detects seed or clock errors. A counter of errors is consequently incremented, RNG is then restarted, and the current dataset extraction is cancelled (see Figure 14.
- Note that the HTCR value is characterized for NIST certification to minimize the seed errors without causing any clock errors. In this experimentation no seed errors were detected during extraction according to this HTCR value.

- Two extraction modes are performed, with restart at each 1000 bits and continuous without restart:
  - When 1000 x 1024 restart dataset is to be extracted, the following macros definitions are considered.

```
/* number of random bits*/
#define NUMBER_OF_RANDOM_BITS        1024
/* number of 32bits words*/
#define NUMBER_OF_RANDOM_32_BITS   NUMBER_OF_RANDOM_BITS /32
/* number of restarts */
#define BLOCK_NUMBER                 1000
```

When continuous 1,024,000 dataset bits is to be extracted, the following macros definitions are considered.

```

/* number of random bits*/
#define NUMBER_OF_RANDOM_BITS        1024*1000
/* number of 32bits words*/
#define NUMBER_OF_RANDOM_32_BITS   NUMBER_OF_RANDOM_BITS /32
/* number of restarts */
#define BLOCK_NUMBER                 1
```

*Note:* *In continuous extraction mode, the size of a block is set to 1,024,000 and the number of blocks is set to 1, so that the RNG is disabled only after the generation of the entire dataset.*

The figures below show the different function description modes.

**Figure 12. Main function in interrupt mode**

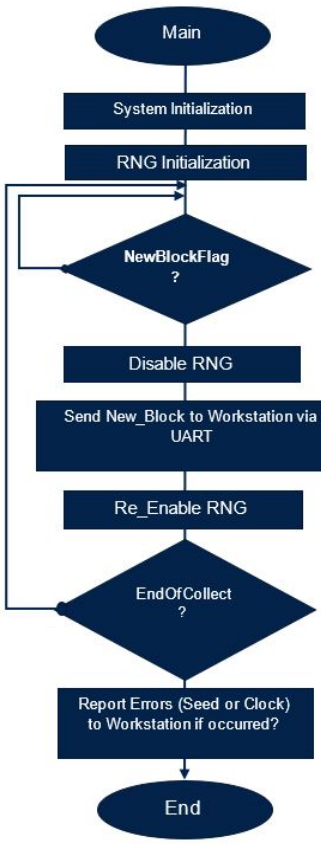


Figure 13. ISR callback function when DRDY interrupt flag is set

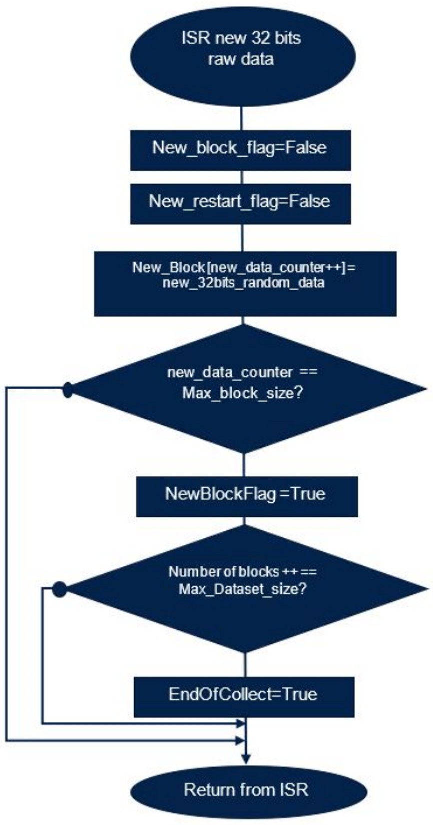
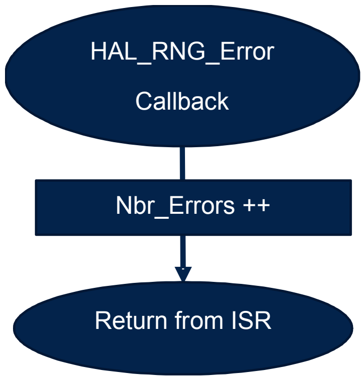

ISR new 32 bits raw data
New_block_flag=False
New_restart_flag=False
New_Block[new_data_counter++]= new_32bits_random_data
new_data_counter == Max_block_size?
NewBlockFlag =True
Number of blocks ++ == Max_Dataset_size?
EndOfCollect=True
Return from ISR

Figure 14. ISR callback function when Error-flag



### 5.1.3 NIST compliant RNG configuration

The table below summarizes the RNG configuration values to use to run the NIST SP800-90B test suite. Those configurations correspond to a RNG clock of 48 MHz.

**Table 3. RNG register values**

| Product | RNG_CR value | HTCR value |
|---|---|---|
| STM32L4P/Q | 0x00F00D40 | 0xAA74 |
| STM32L5 | 0x00F00D40 | 0xA2B3 |
| STM32U5-2M(1)(2) | 0x00F60D40 | 0x9AAE |
| STM32H7A3/7B3(1) | 0x00F00D40 | 0x72AC |
| STM32H72x/73x | 0x00F00D40 | 0xAA74 |
| STM32WLx(1) | 0x00F00D40 | 0xAA74 |

1. *A special device must be used to perform the NIST test suite*
2. *The RNG configuration values can be updated*

*Note: Entropy results can be shared with customers when required.*

## 5.2 NIST SP800-90B test suite steps

### 5.2.1 Step 1: random number generator

Connect the STM32 board to the workstation, via an USB Type-A to Micro-B cable

The STM32 RNG is run via the UART firmware in order to generate a random number as described in STM32 MCU side. Data are stored on the workstation using a terminal emulation application such as PuTTY (free and open-source terminal emulator, serial console and network file transfer application).

### 5.2.2 Step 2: NIST statistical tests

The Makefile is used to compile the program as described in the readme file mentioned in NIST SP800-90b test suite side.

For non-IID tests, the user must follow the steps detailed below:

1. Use the Makefile to compile the program: make non_iid
2. Run the program with

```
./ea_non_iid [-i|-c] [-a|-t] [-v] [-l <index>,<samples> ] <file_name> [bits_per_symbol]
```

where

```
-i indicates that data is unconditioned and returns an initial entropy estimate (default).
-c indicates that data is conditioned.
-a estimates the entropy for all data in the binary file (default).
-t truncates the created bit-string representation of data to the first 1 Mbit.
-l reads (at most) data samples after indexing into the file by * bytes.
-v verbosity flag for more output (optional, can be used multiple times)bits_per_symbol:
number of bits per symbol. Each symbol is expected to fit within a single byte.
```

Example:

```
/ea_non_iid ../bin/l5.bin 1 -i -t -v
```

For restart tests, the user must follow the steps detailed below:

1. Use the Makefile to compile the program: make restart.
2. Run the program with.

```
./ ea_restart [-i|-n] [-v] <file_name> [bits_per_symbol] <H_I>
```

Where

```
[bits_per_symbol]: Must be between 1-8, inclusive.
<H_I>: Initial entropy estimate.
[-i|-n]: '-i' for IID data, '-n' for non-IID data. Non-IID is the default.
-v Optional verbosity flag for more output.
```

```
<file_name>: Must be relative path to a binary file contains the data that consists of 1000
restarts, each with 1000 samples. The data is converted to rows and columns.
```

Example:

```
./ea_restart It_res_1000x1000.bin 1 0.7695 -n -v
```

### 5.2.3 Step 3: test report

The NIST statistical tests provide an analytical routine to facilitate the interpretation of the results:

- For the non IID tests, the result for each IID test is provided and, at the end, the final min entropy.
- For the restart tests, we determine the *H*r and *H*c , the entropy estimates of the row and the column datasets, respectively. the final result of the restart tests is min (H_r, H_c, H_I).

# 6 Conclusion

This application note describes the main guidelines and steps to verify the randomness of numbers generated by the STM32 microcontrollers RNG peripheral, using either NIST statistical test suite SP800-22rev1a, April 2010 or SP800-90B, January 201b.

# Appendix A NIST SP800-22rev1a statistical test suite

The results are represented as a table with p rows and q columns, where:

- p, the number of rows, corresponds to the number of statistical tests applied
- q, the number of columns (q = 13) is distributed as follows:
  - columns 1-10 correspond to the frequency of 10 Pvalues
  - column 11 is the Pvalue 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

The example below shows the first and last part of the test results. For more details, refer to the finalAnalysisReport file under `sts-2.1.1\experiments\AlgorithmTesting`.

**Part 1**

```
------------------------------------------------------------------------------
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
```

**Part 2**

```
 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.

# Appendix B NIST SP800-90b statistical test suite

This is an example of the execution of the non-IID tests.

```
$ ./ea_non_iid ../bin/l5.bin 1 -i -t -v Opening file: '../bin/l5.bin'
Number of Binary Symbols: 1024000
Symbol alphabet consists of 2 unique symbols
Running non-IID tests...
Running Most Common Value Estimate...
MCV Estimate: mode = 530185, p-hat = 0.51775878906249995, p_u = 0.51903071907139675
Most Common Value Estimate (bit string) = 0.946108 / 1 bit(s)
Running Entropic Statistic Estimates (bit strings only)...
Collision Estimate: X-bar = 2.4954732991667945, sigma-hat = 0.49998011778222412, p =
0.55717151750062044
Collision Test Estimate (bit string) = 0.843807 / 1 bit(s)
Markov Estimate: P_0 = 0.4822412109375, P_1 = 0.51775878906249995, P_0,0 =
0.48618305677846313, P_0,1 = 0.51381694322153693, P_1,0 = 0.47857068759018079, P_1,1 =
0.52142931240981927, p_max = 6.2798397734367098e-37
Markov Test Estimate (bit string) = 0.939536 / 1 bit(s)
Compression Estimate: X-bar = 5.2113069751685934, sigma-hat = 1.0187463366852749, p =
0.035431813237513987
Compression Test Estimate (bit string) = 0.803135 / 1 bit(s)
Running Tuple Estimates... t-Tuple Estimate: t = 16, p-hat_max = 0.53187518804173173, p_u =
0.53314533218239224
LRS Estimate: u = 17, v = 38, P_{max,W} = 0.50423097749594759, p_u = 0.50550366496585808
T-Tuple Test Estimate (bit string) = 0.907399 / 1 bit(s)
LRS Test Estimate (bit string) = 0.984207 / 1 bit(s)
Running Predictor Estimates...
MultiMCW Prediction Estimate: N = 1023937, Pglobal' = 0.51634782805743162 (C = 527405)
Plocal = 0.42674646958653317 (r = 21)
Multi Most Common in Window (MultiMCW) Prediction Test Estimate (bit string) = 0.953585 / 1
bit(s)
Lag Prediction Estimate: N = 1023999, Pglobal' = 0.50526439621655594 (C = 516087) Plocal =
0.40830971576974662 (r = 20)
Lag Prediction Test Estimate (bit string) = 0.984890 / 1 bit(s)
MultiMMC Prediction Estimate: N = 1023998, Pglobal' = 0.51899462537798435 (C = 530147)
Plocal = 0.42674521449187308 (r = 21)
Multi Markov Model with Counting (MultiMMC) Prediction Test Estimate (bit string) =
0.946208 / 1 bit(s)
LZ78Y Prediction Estimate: N = 1023983, Pglobal' = 0.51899440603936897 (C = 530139) Plocal =
0.42674552311446512 (r = 21)
LZ78Y Prediction Test Estimate (bit string) = 0.946209 / 1 bit(s)
H_original: 1.000000
H_bitstring: 0.803135
min(H_original, 1 X H_bitstring): 0.803135
```

This is an example of the execution of the restart tests.

```
$ ./ea_restart It_res_1000x1000.bin 1 0.7695 -n -v
Opening file: '/home/Desktop/It_restart_1000x1000_bins/It_res_1000x1000.bin'
Loaded 1000000 samples made up of 2 distinct 1-bit-wide symbols.
H_I: 0.769500
ALPHA: 5.0251553006530614e-06, X_cutoff: 654
X_max: 592
Restart Sanity Check Passed...
Running non-IID tests...
Running Most Common Value Estimate...
Literal MCV Estimate: mode = 524815, p-hat = 0.52481500000000003, p_u = 0.52610132816195332
Most Common Value Estimate (Rows) = 0.926587 / 1 bit(s)
Literal MCV Estimate: mode = 524815, p-hat = 0.52481500000000003, p_u = 0.52610132816195332
Most Common Value Estimate (Cols) = 0.926587 / 1 bit(s)
Running Entropic Statistic Estimates (bit strings only)...
Literal Collision Estimate: X-bar = 2.4939272870560187, sigma-hat = 0.49996374423442524, p =
0.56366499259647185
Collision Test Estimate (Rows) = 0.827090 / 1 bit(s)
Literal Collision Estimate: X-bar = 2.4988330426351748, sigma-hat = 0.49999926291763369, p =
0.54001782688956723
Collision Test Estimate (Cols) = 0.888921 / 1 bit(s)
Literal Markov Estimate: P_0 = 0.52481500000000003, P_1 = 0.47518499999999997, P_0,0
= 0.5286091289311472, P_0,1 = 0.4713908710688528, P_1,0 = 0.52062569446782725, P_1,1 =
0.47937430553217275, p_max = 3.6149969304059282e-36
Markov Test Estimate (Rows) = 0.919808 / 1 bit(s)
Literal Markov Estimate: P_0 = 0.52481500000000003, P_1 = 0.47518499999999997, P_0,0 =
0.52490687194535213, P_0,1 = 0.47509312805464787, P_1,0 = 0.5247146368564598, P_1,1 =
0.4752853631435402, p_max = 1.4806522327909964e-36
Markov Test Estimate (Cols) = 0.929869 / 1 bit(s)
Literal Compression Estimate: X-bar = 5.2053159685055412, sigma-hat = 1.0214276541597365, p
= 0.040397638399976898
Compression Test Estimate (Rows) = 0.771598 / 1 bit(s)
Literal Compression Estimate: X-bar = 5.2093623833151179, sigma-hat = 1.0198576801023285, p
= 0.037170466884093645
Compression Test Estimate (Cols) = 0.791617 / 1 bit(s)
Running Tuple Estimates...
Literal t-Tuple Estimate: t = 16, p-hat_max = 0.54171776156680396, p_u = 0.54300118613085924
Literal LRS Estimate: u = 17, v = 38, p-hat = 0.50185202103932736, p_u = 0.50313992749997694
Literal t-Tuple Estimate: t = 16, p-hat_max = 0.53421514909068124, p_u = 0.535500045383837
Literal LRS Estimate: u = 17, v = 36, p-hat = 0.50196715258596614, p_u = 0.50325505791399572
T-Tuple Test Estimate (Rows) = 0.880973 / 1 bit(s)
T-Tuple Test Estimate (Cols) = 0.901041 / 1 bit(s)
LRS Test Estimate (Rows) = 0.990968 / 1 bit(s)
LRS Test Estimate (Cols) = 0.990638 / 1 bit(s)
Running Predictor Estimates...
Literal MultiMCW Prediction Estimate: N = 999937, Pglobal' = 0.52385156519694509 (C =
522532) Plocal can't affect result (r = 21)
Multi Most Common in Window (MultiMCW) Prediction Test Estimate (Rows) = 0.932770 / 1 bit(s)
Literal MultiMCW Prediction Estimate: N = 999937, Pglobal' = 0.52391856160939276 (C =
522599) Plocal can't affect result (r = 22)
Multi Most Common in Window (MultiMCW) Prediction Test Estimate (Cols) = 0.932586 / 1 bit(s)
Literal Lag Prediction Estimate: N = 999999, Pglobal' = 0.50560437226186183 (C = 504316)
Plocal can't affect result (r = 20)
Lag Prediction Test Estimate (Rows) = 0.983919 / 1 bit(s)
Literal Lag Prediction Estimate: N = 999999, Pglobal' = 0.50229941431576741 (C = 501011)
Plocal can't affect result (r = 19)
Lag Prediction Test Estimate (Cols) = 0.993381 / 1 bit(s)
Literal MultiMMC Prediction Estimate: N = 999998, Pglobal' = 0.52608038171682492 (C =
524793) Plocal can't affect result (r = 21)
Multi Markov Model with Counting (MultiMMC) Prediction Test Estimate (Rows) = 0.926645 / 1
bit(s)
```

```
Literal MultiMMC Prediction Estimate: N = 999998, Pglobal' = 0.5260743824720614 (C = 524787)
Plocal can't affect result (r = 22)
Multi Markov Model with Counting (MultiMMC) Prediction Test Estimate (Cols) = 0.926661 / 1
bit(s)
Literal LZ78Y Prediction Estimate: N = 999983, Pglobal' = 0.52609326184739169 (C = 524798)
Plocal can't affect result (r = 21)
LZ78Y Prediction Test Estimate (Rows) = 0.926610 / 1 bit(s)
Literal LZ78Y Prediction Estimate: N = 999983, Pglobal' = 0.52608626262396685 (C = 524791)
Plocal can't affect result (r = 22)
LZ78Y Prediction Test Estimate (Cols) = 0.926629 / 1 bit(s)
H_r: 0.771598
H_c: 0.791617
H_I: 0.769500
Validation Test Passed...
min(H_r, H_c, H_I): 0.769500
```

## Revision history

Table 4. **Document revision history**

| Date | Version | Changes |
|---|---|---|
| 13-May-2013 | 1 | Initial release. |
| 22-Jun-2016 | 2 | Updated:<br>• Introduction.<br>• Section 1: STM32 microcontrollers random number generator.<br>• Figure 1: Block diagram.<br>• Section 3.1: Firmware description.<br>• Section 3.2.1: First step: random number generator.<br>• Section 3.2.2: Second step: NIST statistical test.<br>• Section 4: Conclusion.<br>• Added<br>• Figure 2: STM32 lines embedding the RNG hardware peripheral. |
| 1-Oct-2019 | 3 | Updated:<br>• Introduction and Table 1: Applicable products<br>• Table 2: STM32 lines embedding the RNG hardware peripheral<br>• Figure 1: STM32 true RNG block diagram<br>• Section 6: Conclusion<br>• Appendix A: NIST SP800-22b statistical test suite<br>Added:<br>Section 4: NIST SP800-90b test suite<br>Section 5: NIST SP800-90b test suite running and analyzing<br>Appendix B: NIST SP800-90b statistical test suite |
| 10-Oct-2019 | 4 | Updated Table 2: STM32 lines embedding the RNG hardware peripheral. |
| 8-Jan-2020 | 5 | Updated:<br>• Table 1: Applicable products<br>• Table 2: STM32 lines embedding the RNG hardware peripheral<br>• Section 5.1.1: STM32 MCU side |
| 18-Aug-2020 | 6 | Updated with new products STM32H72x/73x:<br>• Table 1: Applicable products<br>• Table 2: STM32 lines embedding the RNG hardware peripheral |
| 25-Aug-2021 | 6 | Migration to DITA |
| 1-Jul-2022 | 7 | Updated:<br>• Table 1. Applicable products<br>• Section Appendix A  NIST SP800-22rev1a statistical test suite<br>• Section 4  NIST SP800-90b test suite<br>• Added Section 5  Firmware description used for RNG NIST evaluation |

## Contents

## List of tables

# List of figures