Arm® True Random Number Generator (TRNG)

Revision: r0p0

Characterization User Guide

Non-Confidential

Issue 02
102385_0000_02_en

Arm® True Random Number Generator (TRNG)

Characterization User Guide

Release Information

Document history

Issue	Date	Confidentiality	Change
0000-01	1 February 2021	Confidential	First release.
0000-02	25 March 2022	Non-Confidential	Second release.

Proprietary Notice

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com.

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1 Introduction

1.1 Conventions

The following subsections describe conventions used in Arm documents.

Glossary

The Arm® Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm Glossary for more information: developer.arm.com/glossary.

Typographic conventions

Convention	Use
italic	Citations.
bold	Interface elements, such as menu names. Signal names. Terms in descriptive lists, where appropriate.
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace bold`	Language keywords when used outside example code.
`monospace underline`	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`<and>`	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, **IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, and **UNPREDICTABLE**.
⚠ Caution	Recommendations. Not following these recommendations might lead to system failure or damage.
⚠ Warning	Requirements for the system. Not following these requirements might result in system failure or damage.
⚠ Danger	Requirements for the system. Not following these requirements will result in system failure or damage.
✎ Note	An important piece of information that needs your attention.

Convention	Use
Tip	A useful tip that might make it easier, better or faster to perform a task.
Remember	A reminder of something important that relates to the information you are reading.

1.2 Additional reading

This document contains information that is specific to this product. See the following documents for other relevant information:

Table 1-2: Publications

Document ID	Document name
ANSI X9.31-1988	*Public Key Cryptography Using Reversible Algorithms for the Financial Services Industry (rDSA)*
BSI AIS-31	*Functionality Classes and Evaluation Methodology for True Random Number Generators*
FIPS Publication 140-2	*Security Requirements for Cryptographic Modules*
NIST SP 800-90A	*Recommendation for Random Number Generation Using Deterministic Random Bit Generators – App C.*
NIST SP 800-90B	*Recommendation for the Entropy Sources Used for Random Bit Generation*

Note

Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.

Adobe PDF reader products can be downloaded at http://www.adobe.com

1.3 Other information

See the Arm® website for other relevant information.

- Arm® Developer.
- Arm® Documentation.
- Technical Support.
- Arm® Glossary.

2 Overview of Arm® True Random Number Generator (TRNG)

This chapter provides an overview of the Arm® True Random Number Generator (TRNG) and its characterization.

Arm® True Random Number Generator (TRNG) collects entropy from a physical entropy source, which is a component capable of generating an unpredictable or random output bit stream. The collected entropy is used to seed the cryptographic random bits generator with a secure initial state.

Note

Usually, the physical process used for collecting entropy is an inverter timing jitter that is collected from a dedicated on-chip free-running ring oscillator.

The TRNG can be used in one of two modes, each requiring a different driver:

FE TRNG

The operating mode of the Arm implementation of the FE TRNG driver is compliant with the *BSI AIS-31 Functionality Classes and Evaluation Methodology for True Random Number Generators* standard, as a true random number generator that outputs full-entropy bits at a relatively low rate.

800-90B TRNG

The operating mode of the Arm implementation of the 800-90B TRNG driver is compliant with the *NIST SP 800-90B Recommendation for the Entropy Sources Used for Random Bit Generation* standard, as a true random number generator.

2.1 TRNG characterization

Arm True Random Number Generator (TRNG) configuration parameters specify the settings of the internal ring-oscillator lengths, and the output sampling rate. The parameters are device-specific.

Each silicon process has different noise and jitter characteristics. The specific SoC layout affects these characteristics. Therefore, the TRNG behavior must be characterized on the actual silicon of the device to determine the most suitable parameters. Characterizing in this way ensures that the TRNG output has maximal entropy.

Characterization must be performed during the initial post-silicon testing of the device, or whenever substantial changes are made. For example, after changes to process or respins.

2.2 Compliance

Arm® True Random Number Generator (TRNG) complies with the following specifications:

Table 2-1: Arm® True Random Number Generator (TRNG) compliance

Document ID	Document name	Compliance
BSI AIS-31	*Functionality Classes and Evaluation Methodology for True Random Number Generators*	Compliant in an implementation using FETRNG driver with class PTG.2.
NIST SP 800-90B	*Recommendation for the Entropy Sources Used for Random Bit Generation*	Compliant with section 4.4 Approved Continuous Health Tests.

3 TRNG characterization procedure

This chapter provides the detailed Arm characterization procedure.

3.1 Characterization procedure overview

The full characterization procedure is divided into various steps.

The following table provides an overview of the steps to follow for the TRNG characterization procedure.

Table 3-1: Characterization high-level procedure steps

Step	Execution	Comments
Prepare characterization test program.	Prepare a characterization test program that uses the CC_TST_TRNG routine that Arm supplies.	See 3.2 Characterization test program on page 10 and 3.2.1 Characterization test conditions on page 12.
Base iteration: Find the minimal sample count.	Run the preliminary test to establish the minimal (base) value of the sample count.	See 3.3 Base iteration on page 13.
First iteration: Run first set of characterization tests.	Run a series of characterization tests under multiple test conditions.	See 3.4 First characterization iteration on page 14.
First iteration: Analyze first characterization test results.	Run the 3.5 TRNG characterization utility on page 16.	This utility outputs a set of configuration parameters (sample count values) and a list of corners to use as a basis when you run the second iteration tests.
Second iteration: Run second set of characterization tests.	1. Use CC_TST_TRNG with the configuration parameters that were received in the first iteration. 2. Run the characterization tests in worst-case conditions, as determined in the previous step.	See 3.6 Second characterization iteration on page 23.
Second iteration: Analyze second characterization test results.	Run the 3.5 TRNG characterization utility on page 16. Use this analysis to generate the mass production Arm® True Random Number Generator (TRNG) configuration parameters.	**Note:** If the results are not good enough, repeat the second iteration.
Restart tests iteration	Re-run tests using data from an entropy source that produces outputs for use in production.	See 3.7 Restart tests iteration on page 25.

3.2 Characterization test program

You must implement the characterization test program, using the CC_TST_TRNG API that Arm supplies.

The CC_TST_TRNG API is included in `<prefix>-TRNG_Characterization.c`, where <prefix> represents the product-specific part number and version. For the macros used in following sections, refer to `<prefix>-TRNG_Characterization.c`.

Note

Usually, for a memory-constrained IoT scenario, the collected raw data is redirected to a peripheral, to reduce memory usage. For a device without an available peripheral, you can utilize the max delay between blocks to design a private protocol to collect the data. The max delay between blocks follows this formula: *max_delay* = 4*(192 * *SAMPLE_CNT*/*RNG_CLK*).

Example of API

```
/*
 * Collect TRNG output for characterization
 *
 * @regBaseAddress:     The base address of TRNG registers in system memory map
 * @TRNGMode:           Base iteration    - TRNG_MODE_FAST
 *                      First iteration   - TRNG_MODE_FAST
 *                      Second iteration  - TRNG_MODE_FE compliant with BSI AIS-31
 *                                          TRNG_MODE_80090B compliant with NIST SP
 800-90B
 * @roscLength:         Ring oscillator length (0 to 3)
 * @sampleCount:        Ring oscillator sample counter value
 * @buffSize:           Total buffer size covered header, sample_bits and footer.
 *                      Must be between 52 and 2^24 bytes.
 *                      buffsize >= header(16 bytes)+sample_bits/8 (bytes)+footer(12
 bytes).
 *                      For example, if 100Mbits of data need to be collected, the
 minimal
 *                      buffSize should be 12,500,000+28=12,500,028 bytes. To be
 safe, the
 *                      buffSize can be 12,501,000 bytes.
 * @callbackFunc:       Callback function to process the real output data
 *                      This callbackFunc is called many times in CC_TST_TRNG.
 *                      First, it is called after the 16-byte header is generated.
 *                      Then, it is called repeatedly when the 24-byte EHR registers
 are full.
 *                      Finally, it is called after the 12-byte footer is generated.
 * @return              0 on success. Non-zero value on failure.
 */
int CC_TST_TRNG(    unsigned long regBaseAddress,
                    uint32_t TRNGMode,
                    uint32_t roscLength,
                    uint32_t sampleCount,
                    uint32_t buffSize,
                    callback_TRNG callbackFunc);
```

The TRNG modes are as follows:

- TRNG_MODE_FAST - This mode is the same thing as setting TRNG_MODE to 0. Use in Base iteration, First iteration, and Restart tests iteration (for 800-90B characterization).
- TRNG_MODE_FE - This mode is the same thing as setting TRNG_MODE to 1. Use in Second iteration for BSI AI-S31 characterization.

- TRNG_MODE_80090B - This mode is the same thing as setting TRNG_MODE to 2. Use in Second iteration for NIST SP 800-90B characterization.

Definition of callback_TRNG

```
/*
 * Callback function type define
 * The partner must implement this function according the system resource.
 * This function is called by CC_TST_TRNG() to process the data generated
 * or collected by CC_TST_TRNG().
 * @outputSize          The size of the output data
 * @outputBuffer        The buffer of the data
*/
typedef void (*callback_TRNG)(uint32_t outputSize, uint8_t *outputBuffer);
```

Note

Partners must implement the callback function according to the system resource.

3.2.1 Characterization test conditions

Each characterization test is executed by running the characterization test program under a combination of conditions.

These tests are described in Characterization test conditions on page 12.

Table 3-2: Characterization test conditions

Configuration variable	Operating conditions	Filename values
Ring oscillator length.	The four configurable lengths that Arm® True Random Number Generator (TRNG) allows.	R0, R1, R2, R3. R0 is the shortest length and R3 is the longest. **Note:** In some chips, the ring oscillator might not properly work when configured to the shorter lengths.
Voltage.	High, typical, low.	VH, VT, VL.
Temperature.	High, typical, low.	TH, TT, TL.
CMOS process corner.	Typical, fast/fast, fast/slow, slow/fast, slow/slow.	CT, CFF, CFS, CSF, CSS.

3.2.2 Output-file names

Output-files are named according to a standard format.

The output file for each characterization test is named according to the Filename Values column in Characterization test conditions on page 12. Output filename format is `trng_samples_R*_S*_V*_T*_C*.bin`.

For example, for the following characterization test:

- The longest ring oscillator length.
- Sample counter value of five.
- High voltage.
- Low temperature.
- Fast or slow CMOS corner.

The filename is `trng_samples_R3_S5_VH_TL_CFS.bin`.

3.3 Base iteration

The base iteration is used to find the minimum sample counter value for which Arm® True Random Number Generator (TRNG) operates properly, under typical operating conditions.

About this task

The typical operating conditions are:

- The second-longest ring-oscillator.
- Typical voltage.
- Typical temperature.
- Typical process corner.

Procedure

1. Find the minimum sample counter by calling CC_TST_TRNG with increasing values of *`sampleCount`* (starting with 1) until it exits successfully.
    - At least 200Kbits (2.0e5 bits) must be collected for the base iteration and the first characterization iteration.
    - Set the *`TRNGMode`* to 0 for both the NIST 80-090B and FE-TRNG driver modes.

   In many systems, the test succeeds immediately (with *`sampleCount=1`*), which is an expected and even desirable result.
2. Use the minimum sample counter for the 3.4 First characterization iteration on page 14

Example 3-1: Base iteration sample code

The following is sample code for data type, macro, and callback function definition:

```
typedef unsigned char uint8_t;
   typedef unsigned int uint32_t;
   /*
   * Define the buffer size for the base iteration.
   * At least 200 Kbits(25 KBytes) must be collected
   * every time EHR registers output 192 bits (24Bytes) value.
   * Therefore, set the collected data as 25008 bytes (25008/24 = 1042).
   * The full buffer size is 25008+28(header+footer)=25036 bytes.
   */
```

```
    #define BASE_ITERATION_BUF_LEN 25036 //0x61CC
    static uint8_t data_buf[BASE_ITERATION_BUF_LEN];
    static uint32_t buffer_index = 0;

    static void callback_BaseIteration(uint32_t outputSize, uint8_t *outputBuffer)
    {
    if (NULL == outputBuffer)
    {
    printf("invalid input parameter!\r\n");
    }

    memcpy(data_buf+buffer_index, outputBuffer, outputSize);
    buffer_index += outputSize;
    }
```

The following is sample code for finding minimum sample counter value:

```
int ret = 0;
    uint32_t i = 0;
    uint32_t sampleCounter = 0;

    for (i=1; ;i++)
    {
    ret = CC_TST_TRNG(g_CcBaseAddr,    // regBaseAddress
    0,      // TRNGMode
    2,      // roscLength
    i,      // sampleCount
    BASE_ITERATION_BUF_LEN,  // buffSize
    callback_BaseIteration); // callback
    buffer_index = 0;

    if (ret == 0)
    break;
    }
    sampleCounter = i;
```

3.4 First characterization iteration

Perform this procedure for each of the characterization test conditions combinations.

Procedure

1. Set up the test operating conditions.

2. Run CC_TST_TRNG under these conditions with each of the following `sampleCount` values:
   - First set: S_{SMP1} = The minimal `sampleCount` found in 3.3 Base iteration on page 13.
   - Second set: S_{SMP2} = Ceiling(1.5 * (S_{SMP1}+1)-1).
   - Third set: S_{SMP3} = Ceiling(1.5 * (S_{SMP2}+1)-1).

   For example, if the minimum value of `sampleCount` is 8, then run CC_TST_TRNG under each of the 180 conditions with:

   - `SMP1=8` in the first set.
   - `SMP2=13` in the second set.
   - `SMP3=20` in the third set.

   Note

   Set the `TRNGMode` to 0 for both the NIST 80-090B and FE-TRNG driver modes.

   Overall, you must run a total of 540 characterization tests: (3 (sample counter values) * 4 (ring oscillator configurations) * 3 (voltages) * 3 (temperatures) * 5 (process corners).

   You must export all the data that is collected as part of the first characterization iteration from the device, using your implementation of the `callbackFunc` callback function.

   The estimated total time for the characterization procedure is between 16-18 hours: 5 chips * 3 temperature conditions * (30 minutes to set each chip in each temperature condition + [30-40] minutes for running all oscillator, voltage, and sample count combinations in this temperature).
3. Save the results of each test run in the relevant output file that is named according to 3.2.2 Output-file names on page 12.
   When the first characterization iteration is complete, you can expect to have 540 files that are named according to the different test conditions.
4. Run the 3.5 TRNG characterization utility on page 16 on the collected files that you exported from the device.
   The utility analyzes the data and returns to the following parameters.
   - A set of four sample counter values, one value for each ring oscillator length.
   - The worst-case corners (voltage/temperature/process) that you use for the 3.6 Second characterization iteration on page 23. There are several worst-case corners for each ring oscillator.

Example 3-2: Sample code for the first characterization iteration

In this example, SMP1=1, SMP2=2 and SMP3=4 and callback_FirstIteration is used as the callback_BaseIteration for the base iteration.

The *printData* function used in this example is a reference. Its purpose is to output data from data_buf buffer through an I/O channel. CC_TST_TRNG collects data from the device and fills up the data_buf buffer by calling callback function callback_FirstIteration.

```
void first_iteration_test()
    {
    uint32_t TRNGMode = 0;
    uint32_t roscLength = 0;
    uint32_t sampleCounter = 1;
    int ret = 0;

    for (roscLength=0; roscLength<4; roscLength++)
    {
    sampleCounter = 1;
    ret = CC_TST_TRNG(g_CcBaseAddr, TRNGMode, roscLength, sampleCounter,
 FIRST_ITERATION_BUF_LEN, callback_FirstIteration);
    printData(ret, roscLength, sampleCounter);
    buffer_index = 0;
    memset(data_buf, 0, sizeof(data_buf));

    sampleCounter = 2;
    ret = CC_TST_TRNG(g_CcBaseAddr, TRNGMode, roscLength, sampleCounter,
 FIRST_ITERATION_BUF_LEN, callback_FirstIteration);
    printData(ret, roscLength, sampleCounter);
    buffer_index = 0;
    memset(data_buf, 0, sizeof(data_buf));

    sampleCounter = 4;
    ret = CC_TST_TRNG(g_CcBaseAddr, TRNGMode, roscLength, sampleCounter,
 FIRST_ITERATION_BUF_LEN, callback_FirstIteration);
    printData(ret, roscLength, sampleCounter);
    buffer_index = 0;
    memset(data_buf, 0, sizeof(data_buf));
    }
    }
```

3.5 TRNG characterization utility

The TRNG characterization utility allows you to analyze the first and second characterization test results independently and automatically. Therefore, there is no need to send the test results to Arm for analysis.

Before you begin

Before starting this procedure, do the following:

- Verify that one of the following operating systems is installed:
  - Ubuntu 20.04 LTS OS
  - CentOS 7.7.1908
- Verify that Python 3.7 or above is installed.

- Prepare the input files as required for each iteration in the characterization process as described in the following sections:
  - 3.4 First characterization iteration on page 14
  - 3.6 Second characterization iteration on page 23
  - 3.7 Restart tests iteration on page 25

Note

The size of each file, which is known as the *buffsize* in CC_TST_TRNG, must be equal to or greater than the following:

- First characterization iteration: 25036 bytes
- Second characterization iteration for AIS-31 mode: 12500044 bytes
- Second characterization iteration for 800-90B mode: 125044 bytes.

- Install r-base-core for Ubuntu by entering `$sudo apt install r-base-core`.
- Verify that ImageMagicK is installed. It is generally pre-installed on Ubuntu. If not, install it by entering `$sudo apt install imagemagick`.
- For Ubuntu users:
  - Verify that libdivsufsort is installed. If not, install it by entering `$sudo apt install libdivsufsort-dev`.
  - Verify that libbz2 is installed. If not, install it by entering `$sudo apt install libbz2-dev`.
- For CentOS users:
  - Build and install libdivsufsort from GitHub.
  - Install libbz2 by entering `$sudo yum install bzip2-devel`.

Note

The NIST tools have their own licensing requirements. Refer to the NIST website for the latest licensing information.

Procedure

1. Copy the entire TRNG characterization utility folder to your work area.
2. Copy the input files specified in the *Prerequisites* section to a folder in your work area.
3. Open a terminal session and navigate to the folder where the TRNG characterization utility is saved.
   You can invoke the help menu by running one the following commands:

```
./characterization.py -h
```

or

```
./characterization.py --help
```

4. Execute the utility by running the following command:

   ```
   ./characterization.py <TRNG_STEP> <DATA_FILES_LOCATION>
   ```

   a) Replace <TRNG_STEP> with one of the following options:
      - FIRST - 3.4 First characterization iteration on page 14
      - SECOND_AIS - 3.6 Second characterization iteration on page 23
      - SECOND_NIST - 3.6 Second characterization iteration on page 23
      - RESTART_NIST - 3.7 Restart tests iteration on page 25

   b) Set the <DATA_FILES_LOCATION> to a relative or an absolute path to an existing folder.

   For example, to run the first characterization iteration on files located in `my_folder`, which is located one hierarchy above the location of the utility, enter:

   ```
   ./characterization.py FIRST ../my_folder/
   ```

5. Optionally, you can specify a name for the log file.
   To do this, use one of the following flags:

   ```
   -l <FILE_name>
   ```

   or

   ```
   --log=<FILE_NAME>
   ```

   By default, the log file name is `characterization_log.log`.

6. Optionally, you can override the results of the dependency check performed by the utility before the analysis process starts.
   To do this, use one of the following flags:

   ```
   -f
   ```

   or

   ```
   --force
   ```

   The dependency checker verifies that the required tools mentioned in the *Prerequisites* section are installed. In some cases, the dependency checker cannot detect that the required tools have already been installed. This flag allows the utility to run even if the dependency checker fails.

Results

When the utility starts to operate, the following message is displayed:

```
INFO: Logging of application ### TRNG Characterization script ### started.
```

If there are no issues with the format of the input files, the following message is displayed:

```
INFO: Input file format verification step OK.
```

If there are no issues with data files conversion process, the following message displays:

```
INFO: *.bin -> *.dat conversion step completed.
INFO: *.dat -> *.param conversion step completed.
INFO: *.param -> params.dat conversion step completed.
```

Note

If there is an issue when running the utility, an error message is displayed. See 3.5.2 Characterization utility error messages on page 21 for more information.

The characterization process is complete when the following message is displayed:

```
INFO: Running characterization...
INFO: Analysis completed. The results are the following:
```

Note

The analysis process can take up to several hours.

Next steps

After the process is complete, read the output from the screen or from the log file. See 3.5.1 Characterization utility output on page 19 for more information.

3.5.1 Characterization utility output

After the TRNG characterization utility runs, the output is displayed.

Here are some example outputs for the:

- First characterization iteration
- Second NIST characterization iteration
- Second AIS characterization iteration
- Restart NIST test iteration

First characterization iteration outputs

The raw output should be similar to the following example:

```
STRNG
For R3 (and each S) the worst dt happens in
```

```
R S V T  C  start         b         t        dt         o
22032 3 4 L H FF 159000 1.0086670 0.5741914 0.1746939 -1868.344
5237  3 2 L H FS  66000 0.9953333 0.4628209 0.1481583 -1722.774
10989 3 1 L H FF  96000 1.0200000 0.6869580 0.1082101 -1414.891
* For STRNG with R3 sample every 6.8e+02 cycles
For R2 (and each S) the worst dt happens in
R S V T  C  start         b         t        dt         o
29228 2 4 L H FS 165000 0.9780000 0.7054703 0.1741946 -1930.156
4760  2 2 L H FS  21000 0.9873333 0.5865288 0.1517287 -1773.215
22511 2 1 L H FF  12000 1.0353330 0.6924616 0.1292126 -1616.153
* For STRNG with R2 sample every 6.6e+02 cycles
For R0 (and each S) the worst dt happens in
R S V T  C  start        b         t        dt         o
1156  0 4 L H FF  99000 1.027333 0.4571524 0.2965376 -2523.992
1295  0 2 L H FF 120000 1.026667 0.5301767 0.2359807 -2291.166
12051 0 1 H H SS 114000 1.054000 0.8205470 0.1905827 -1780.738
* For STRNG with R0 sample every 2.3e+02 cycles
For R1 (and each S) the worst dt happens in
R S V T  C  start         b         t        dt         o
15560 1 4 L H FS 147000 0.9706667 0.5221741 0.2324226 -2264.606
26372 1 2 L H FS 111000 0.9773333 0.4478159 0.1956571 -2062.281
23119 1 1 L H FF  54000 1.0180000 0.4234745 0.1739592 -1923.775
* For STRNG with R1 sample every 3.7e+02 cycles
* For STRNG use R0 with 2.3e+02 clocks between samples
```

The parsed output should be similar to the following example:

```
2021-01-06 10:33:56,068 - INFO: Analysis completed. The results are the following:
The Worst conditions:
For R0:
VH TH CSS
VL TH CFF
For R1:
VL TH CFF
VL TH CFS
For R2:
VL TH CFS
VL TH CFF
For R3:
VL TH CFF
VL TH CFS

The sample counter values:
For R0 use sample counter value 2.3e+02
For R1 use sample counter value 3.7e+02
For R2 use sample counter value 6.6e+02
For R3 use sample counter value 6.8e+02
```

In the example above, the sample count values are:

- R0 = 230
- R1 = 370
- R2 = 660
- R3 = 680

And the worst-case corners are:

```
For R0:
 VH TH CSS
 VL TH CFF
For R1 :
 VL TH CFF
 VL TH CFS
For R2:
 VL TH CFS
 VL TH CFF
For R3:
 VL TH CFF
 VL TH CFS
```

Second NIST iteration outputs

The log messages on success output should be similar to the following example:

```
  TIMESTAMP - INFO: Analysis completed for file nist_trng_samples_R0_S410_VT_TH_CSF.bin
TIMESTAMP - INFO: IID test PASSED for file nist_trng_samples_R0_S410_VT_TH_CSF.bin. Please take
 note of the following H_original value: 0.8854! (for all files)
TIMESTAMP - INFO: Characterization iteration SECOND_NIST completed successfully! Please continue
 with collecting data for step RESTART_NIST.
```

Second AIS iteration outputs

The log messages on success output should be similar to the following example:

```
TIMESTAMP - INFO: Analysis completed for file trng_samples_R2_S680_VT_TH_CT.dat
TIMESTAMP - INFO: Tests PASSSED for file trng_samples_R2_S680_VT_TH_CT.dat.(for all files)
TIMESTAMP - INFO: Characterization iteration SECOND_AIS completed successfully! The currently
 used sample counter values can be finalized.
```

Restart NIST test iteration outputs

The log messages on success output should be similar to the following example:

```
TIMESTAMP - INFO: Analysis completed for file nist_trng_samples_R0_S410_VT_TH_CSF.bin
TIMESTAMP - INFO: Restart test PASSED for file nist_trng_samples_R0_S410_VT_TH_CSF.bin (for all
 files)
TIMESTAMP - INFO: Characterization iteration RESTART_NIST completed successfully!
```

3.5.2 Characterization utility error messages

If the TRNG characterization utility has a problem when running, one of the following messages is displayed. Take the required action to fix the problem.

Table 3-3: Dependency check errors

Error message	Action required
ERROR: R installation not found. Please install the package 'r-base-core' through your package manager.	R-base-core is not installed on this machine. Follow the instructions in *Prerequisites* section above.
ERROR: Imagemagick installation not found. Imagemagick installation not found.	Imagemagick is not installed on this machine. Follow the instructions in *Prerequisites* section above.
ERROR: libdivsufsort-dev package installation not found. libdivsufsort-dev package installation not found.	The libdivsufsort-dev package is not installed on this machine. Follow the instructions in *Prerequisites* section above.
ERROR: libbz2-dev package installation not found.	The libbz2-dev package is not installed on this machine. Follow the instructions in *Prerequisites* section above.
ERROR: 32bit libc6/glibc package installation not found.	Install the i386 version of the libc package for your system. The package name could differ for different Linux distributions/versions.

Table 3-4: Input file errors

Error message	Action required
ERROR: The number of different sample counter values across the dataset (got: x) is not equal to the expected value of 3. Please check your dataset!	There are x different sample count values mentioned in the files names. Only 3 are expected. Check the input data set.
ERROR: Error while parsing file <<trng_samples_R0_S16000_VT_TH_CSS.bin>>: roscLength flag at word offset 1 is not equal to the value given in the filename! Parsed value is: 1	Check why the ROSC length value is different between the file name (R0 in this example) and the file header (R1 in this example).
ERROR: Error while parsing file <<trng_samples_R0_S1600_VT_TH_CSS.bin>>: sampleCount value at word offset 2 is not equal to the value given in the filename! Parsed value is: 16000	Check why the Sample Count value is different between the file name (1600 in this example) and the file header (16000 in this example).
ERROR: Error while parsing file <<trng_samples_R1_S2_VT_TT_CT.bin>>: TRNGMode flag at word offset 1 is inconsistent to the value given for command line argument TRNG_STEP! Parsed value is: 2	Check why the TRNG Mode is different between the file header (1 in this example) and the command line argument (2 in this example).
ERROR: Error while parsing file <<trng_samples_R0_S16000_VT_TH_CSF.bin>>: dataset size is not large enough for the characterization iteration specified by command line argument TRNG_STEP. Parsed length of dataset is 124984 bytes.	Recollect the raw data as specified for the current iteration.
ERROR: Error while parsing file <<trng_samples_R0_S16000_VT_TH_CSS.bin>>: signature values at word offset 0 and 3 are not the same!. Values of both words must be 0xAABBCCDD! Please make sure your file is binary file formatted according to the documentation!	Recollect the raw data as specified. Check why the file header does not match the format specified in the output format in A CC_TST_TRNG output format on page 27.
ERROR: Error while parsing file <<trng_samples_R0_S16000_VT_TH_CSS.bin>>: No ending signature value found after data section.	Recollect the raw data as specified. Check why the file footer does not match the format specified in the output format in A CC_TST_TRNG output format on page 27.
ERROR: The following .bin files have unexpected filenames: R3_S1_VT_TT_CT.bin	Verify that all files names to comply with the naming conventions specified in 3.2.2 Output-file names on page 12.
ERROR: Error while parsing file <<trng_samples_R0_S2_VT_TT_CT.bin>>: error flag at bit 0 is set - Samples were lost during collection!	Some of the bits that were output from the ROSC were not saved to files. Recollect the data for the corners as specified in 3.6 Second characterization iteration on page 23. As with the first-iteration tests, it is critical that: • All output bits are collected using a single contiguous execution of the test. • All resulting bits are saved in the output file without any gaps.

Table 3-5: Second iteration NIST mode errors

Error message	Action required
INFO: NIST tool not found at local path 'SP800-90B_EntropyAssessment'. Trying to download it from https://github.com/usnistgov/SP800-90B_EntropyAssessment/archive/v1.0.zip ERROR: Downloading NIST tool failed. Please check your network connection!	Check your internet connection. The required NIST tool is missing and should be automatically downloaded by the utility. Alternatively, you can manually download it to the folder that contains the utility by entering the following command: `./git clone https://github.com/usnistgov/SP800-90B_EntropyAssessment`

Error message	Action required
ERROR: IID test FAILED for file nist_trng_samples_R0_S1200_VT_TH_CFS.bin!	Repeat this iteration with increased sample counter values for corner cases with the same ring oscillator length (R0 in this example) as the failed tests.
ERROR: Characterization iteration SECOND_NIST completed with failed tests!	

Table 3-6: Second iteration AIS mode errors

Error message	Action required
ERROR: Error while parsing file <<trng_samples_R0_S210_VH_TL_CSF.bin>>: dataset size is not large enough for the characterization iteration specified by command line argument TRNG_STEP. Parsed length of dataset is 12499632 bytes.	Recollect the raw data as specified for the current iteration.
WARNING: Tests FAILED for file trng_samples_R0_S410_VH_TH_CFF.dat. Please see report file report_trng_samples_R0_S410_VH_TH_CFF.txt for details!	Repeat this iteration with increased sample counter values for corner cases with the same ring oscillator length (R0 and R3 in this example) as the failed tests.
WARNING: Tests FAILED for file trng_samples_R3_S760_VT_TH_CFF.dat. Please see report file report_trng_samples_R3_S760_VT_TH_CFF.txt for details!	
ERROR: Characterization iteration SECOND_AIS completed with failed tests! Please adjust sample counter values for rings R0, R3 then recollect data for the failed cases and rerun the iteration.	

3.6 Second characterization iteration

You must run a second set of characterization tests to determine the TRNG configuration parameters.

Procedure

1. Collect data for each of the worst corners identified as result of the first iteration.
   Call CC_TST_TRNG with each of the four ring oscillator lengths (each with its corresponding sample count).

   Depending on the intended TRNG driver, each call to CC_TST_TRNG must be as follows:

   - FE TRNG driver: Call CC_TST_TRNG with *`TRNGMode=1`* and collect 100Mbit (12.5MB).
     - Arm recommends that all output bits be collected using a single contiguous execution of the test if your system allows. However, if there is insufficient memory to collect all required bits in a single run, you can split each test into several runs and concatenate the data. Each run must be a multiple of 1Mbits. Examples are splitting the test into 100 runs and collecting 100 x 1Mbits, 50 runs (50 x 2Mbits), or 25 runs (25 x 4Mbits).

    - All resulting bits must be saved in the output file without any gaps.
- 800-90B TRNG driver: CC_TST_TRNG with *`TRNGMode=2`* and collect 1Mbit.
    - Due to the required size of the data, the output bits can be non-contiguous, and concatenation of several smaller sets of consecutive samples (generated using the same noise source) is allowed.
    - Smaller sets must contain at least 1000 samples.
    - The concatenated dataset must contain at least 1,000,000 samples.
    - All resulting bits must be saved in the output file without any gaps.

Note

If any bits are dropped and not captured in the output file, you must rerun the test as the statistical analysis of the output is meaningless.

The same output file naming rules apply for both drivers. For more information, see 3.2.2 Output-file names on page 12.

2. Run the 3.5 TRNG characterization utility on page 16 with the resulting output files for statistical analysis.
The returned results confirm or refute the TRNG configuration that is used for mass production. After the 3.7 Restart tests iteration on page 25, these configuration values must be updated in the relevant TRNG driver files.

Note

If there are errors, recollect the data for each failed corner with a larger sample count value. Then rerun the test and repeat until successful.

Example 3-3: Sample code of the second characterization test that is based on the FE TRNG driver

The following sample code includes the macro and callback function definitions.

```
/*
 * Define the buffer size for the second characterization iteration.
 * For example, at least 100 Mbits (12.5 MBytes) must be collected
 * every time EHR registers output 192 bits (24Bytes) value.
 * Therefore, set the collected data as 12500016 bytes (12500016/24 = 520834).
 * The full buffer size is 12500016+28(header+footer)=12500044 bytes.
 */
#define SECOND_ITERATION_BUFF_LEN      (12500044)

static void callback_SecondIteration(uint32_t outputSize, uint8_t *outputBuffer)
{
 int i = 0;
 if (NULL == outputBuffer)
 {
  printf("invalid input parameter!\r\n");
 }
 for (i=0; i<outputSize; i++)
 {
  printf("%02x", *(outputBuffer+i));
 }
}
```

Example 3-3: Second characterization test sample code

The following sample code is based on `callback_SecondIteration`.

```
void second_iteration_test()
{
 uint32_t TRNGMode = 1;
 uint32_t roscLength = 0;
 uint32_t sampleCounter = 1;
 int ret = 0;
 uint32_t i = 0;

 for (roscLength=0; roscLength<4; roscLength++)
 {
  /*
  * The following line must be updated for your board.
  * It is based on Musca-B1 test boards.
  */
  if ((roscLength == 1) || (roscLength == 3))
   continue;
  /*
  * The sample counter values in the following switch statement must
  * be updated according the real cases. The following are the values
  * for the tested Musca-B1 board.
  */
  switch (roscLength)
  {
   case 0:
    sampleCounter = 360;
    break;
   case 1:
    sampleCounter = 420;
    break;
   case 2:
    sampleCounter = 600;
    break;
   case 3:
    sampleCounter = 620;
    break;
  }
  printf("\r\nroscLength=%d, sampleCount=%d\r\n", roscLength, sampleCounter);
  ret = CC_TST_TRNG(g_CcBaseAddr, TRNGMode, roscLength, sampleCounter,
 SECOND_ITERATION_BUFF_LEN, callback_SecondIteration);
  printf("\r\nret = %d\r\n", ret);
 }
}
```

3.7 Restart tests iteration

If you have selected the 800-90B driver, you must run a third characterization step to verify the results from the second iteration.

Before you begin

- Ensure that the sample counter values you use are from a successful second characterization iteration.

Procedure

1. Perform 1000 power cycles of the device, and in each cycle call CC_TST_TRNG with *TRNGMode* = 0. Collect 1000 bits under standard temperature, voltage, and process corner for each ring oscillator setting.
2. Concatenate the 1000 sequences one by one to form a single file for each ring oscillator.
3. Analyze the file, using the 3.5 TRNG characterization utility on page 16, to confirm or refute the TRNG configuration that was calculated in the second iteration.
   You must ensure that each file contains exactly 1,000,000 bits of collected samples. Because CC_TST_TRNG can output only multiples of 24 bytes (the length of the EHR register), this requirement is achieved by stripping the excess bits. Collect at least 1000 bits in each power cycle and keep only the first 1000 bits before concatenating the sequences together.

   The TRNG characterization utility produces an H_original value for each analyzed file in the second iteration analysis. Group these values by the analyzed ring oscillator (R0-R3) and select the smallest one from each set of values. These four values must be used as the H_I values for the corresponding four input files (R0-R3) during the restart tests iteration.

   The following snippet of code shows an example.

   ```
   TIMESTAMP - INFO .bin -> nist_.bin conversion step completed.
   Enter H_I value for file nist_trng_samples_R2_S2300_VT_TT_CT.bin
   ```

   If there are failed tests, increase the sample counter value and repeat the restart tests.

Next steps

After verification, update these configuration values in the relevant files of the TRNG driver that you are using.

Appendix A CC_TST_TRNG output format

This appendix describes the format of the CC_TST_TRNG output.

When CC_TST_TRNG is run, in addition to collecting samples, it stores some metadata in its output buffer. This buffer is described in terms of 32-bit little-endian words.

The following table lists the value of each word.

Table A-1: CC_TST_TRNG output format

Buffer offset (32-bit words)	Value
0	Signature value: `0xAABBCCDD`.
1	• Bits [23:0] - *`buffSize`*. • Bits [25:24] - *`roscLength`*. • Bits [31:30] - *`TRNGMode`*.
2	*`sampleCount`*
3	Signature value: `0xAABBCCDD`.
4..N-1	Collected samples. Each 32-bit word contains the first collected sample in bit 0, and the last collected sample in bit 31.
N	Signature value: `0xDDCCBBAA`.
N+1	Error flags • Bit [0] - Samples were lost during collection. • Bit [1] - Autocorrelation error occurred. • Bit [2] - CRNGT error that is detected and recovered. • Bit [3] - Input that is stuck at same level for 32 bits. **Note:** Bits[3:1] have no effect on the characterization analysis process.
N+2	Signature value: `0xDDCCBBAA`.

The value of "N" is derived from the buffer size, and is calculated to fit the entire data in the supplied buffer (as per the *`buffSize`* argument).

When parsing the output, it is important to test that the signature value is correct.

Appendix B Revisions

This appendix describes the technical changes between released issues of this book.

Table B-1: Issue 01

Change	Location
First release.	-

Table B-2: Differences between issues 01 and 02

Change	Location
Product renamed Arm® True Random Number Generator (TRNG).	Entire document
Clarified description of TRNG modes.	3.2 Characterization test program on page 10
Clarified information about `printData` in example of sample code for first characterization iteration.	3.4 First characterization iteration on page 14
Corrected description of second characterization iteration procedure, that each test run must be a multiple of 1Mbits.	3.6 Second characterization iteration on page 23
Fixed buffer size values in the sample code of the second characterization test.	3.6 Second characterization iteration on page 23
Clarified description of restart tests iteration. Added extra information to step 3 of procedure.	3.7 Restart tests iteration on page 25
Reorganised appendix structure. No technical changes.	A CC_TST_TRNG output format on page 27