TRNG9880 Random Number Processing

The TRNG9880 is a hardware SMT module for random number generation. We describe the TRNG9880 design principles, the source processing and testing of the random numbers.

Identification of Product:

This description of random number processing refer to all units that carry the updated software, with production date of 0745 and higher. Previously produced units, with lower production date, is not recommended for new designs. It is possible to update the software at TRNG98; please contact us if you have any questions. The description provided also apply to the Protego R300, if the production date restriction is met; the R300 is however no longer in production. TRNG98 no not have permission to use the "Protego" or the "R300" brand name, that is owned by Protego Information AB and it also possibly licensed www.harrysbilskrot.se

This report only apply to the random number processing; some customers have customer specific software installed that is not considered.

Web site: http://www.trng98.se
Manufacturer: Bo Dömstedt Electronics
Address: Vittnesgränden 17, SE-226 47 Lund, Sweden
Phone: +46 70 658 79 70
E-mail: TRNG98 <sales@trng98.se>
Skype: TRNG98
VAT reg number: SE631130661401
EORI: SE6311306614

Identification of the Hardware

A connection is made to the CMD serial port. At reset/power on the unit print the unit ID:

TRNG9880T-1319

Brand-name
"TRNG9880" is the brand-name identifier

Hardware revision code
T is the hardware revision code (example character):

"A" used on previously produced units. Some of these units hold the older software.
"T" (Test-Units) used on TRNG9880 produced using the older yellow can with round corners.
"H" (RoHS-units) Lead-free production.

Production year
"13" is the production year, 2013

Production week
"19" is the production week of the year.

The ID code is terminated by a CR-LF, ASCII (decimal) 13,10.
We plan to expand the ID code in future products, where the ID code may appear on two lines.

An Attempt to Define the RANDOMNESS

The first step is to understand, that randomness is not a property of a generator or a sequence. It is a property of the Observer. And different observers may be of different opinion!

Suppose that a source produce a sequence of numbers. An observer see the sequence, and try to describe it. If he cannot to this, the sequence lack a description, and we say that it is a Random Sequence. (Lars Löfgren [123,124])

We should understand that the concept of randomness is relative to the analytic capability of the observer. If we do random testing on a technical system of low complexity, such as an elevator, a simple sequence such as (select floor) 1,3,1,3, could be considered random relative to this system. An integer division module (software) for a microcontroller could in a similar way be tested using "random" numbers. If we know much on the internal structure of the system, we could select special numbers that is especially effective in testing the system.

A good source of random numbers, where any structure of any kind would be difficult to obtain, is typically needed when the observer is a human, he can profit from any small deviation, and he can gain access to large amounts of random numbers to analyse on a computer. Typical examples would be encryption products and game solutions, where players play with real money.

There is some thought, that real randomness should be of atomic decay origin. A practical problem; products that are radioactive typically have a problem at the customs. A more serious concern would be that the random properties of atomic decay have not been known for very long time, and in Physics there will be scientific progress in future. This could change the view, that we now have, that radioactive decay is "random". It is random due to that we currently lack a proper description. This can change in the future.

The Hardware Source

The unit use a Johnson noise source. The randomness originates at a high input resistance of an ordinary operational amplifier. The noise signal is amplified using both external amplifiers, and also analogue amplifies inside the microcontroller. The resulting analogue signal is export by a buffer amplifier to the ANALOG test pad, where it can be measured. If a disturbance is intentionally injected, on the power line, a correlation between the disturbance and the noise signal may be observed.

The noise signal is sampled into 8-bit bytes using a serial-to-parallel converter. The output of this register is available at the DIGTEST test pad. Note that there is an 8 bit delay between the ANALOG test pad and the DIGTEST pad.

An important property of the random number hardware, that we will need below, is a lack of memory elements and internal state. What we need is that one sample should be independent from other samples. The statistical distribution is of minor importance; but this independent property is absolutely essential.

That samples taken at different times is independent can be seen from the layout of the circuit itself, where any state or signal will be cancelled quickly. On a shorter time scale there will, however, actually be correlations, and an analogue measurement on the ANALOG test pad or the corresponding digital DIGTEST pads evidently show this. But samples taken a few bytes apart are completely independent of each other, as the circuit have forgotten what the state was. Note that a more complex analogue circuit, that may be better at any or most statistical properties, will be weaker on the independence scale, as there are more construction elements that carry state.

Noise and Disturbances

The TRNG9880 use double 100nF decoupling capacitors (older units used one) to lower external power line noise and to decouple the unit from the environment, in order to eliminate a need for external components. The noise source is further protected by an old-style resistor-capacitor low pass network. The operational amplifier have good power line noise rejection; all modern operational amplifiers have very good noise rejection. There is a low pass network in the noise amplification itself to reduce the risk of high frequency blocking, on radio frequencies, due to transmitters or similar on-board equipment.

The reference level of the analogue amplifier inside the microcontroller is calculated continuously using programmable analogue amplifiers, set to produce a DC reference level. If a low frequency disturbance enter the unit, due to a high magnetic field or an elevated power line disturbance, the reference level will swing and track the input signal, thereby cancel out the disturbance. The update can be seen as a non-random signal at about 7kHz on the ANALOG test pad.

The datasheet state protection from a 0.3V power line signal at 200Hz. This can be tested on any unit. For a higher frequency of 100Hz-1MHz the protection is set to 0.1V on the power line. At higher frequencies the low-pass filter is effective.

The unit run on an AC amplifier; the effect that happens if a disturbance enters, is that "the swing" blocks the comparator; we see a long string of "000000" followed by a section of random input, and then a long string of "111111", and random input, and then the long string of "00000000" again. This is further taken care of in the software, where an algorithm runs as follows:

A small buffer of 2 previous sampled bytes is used for comparison. If the new input byte is different compared to either of these, the sample is OK. When the third repeated byte is received the input is considered to be a non-random repetition.

For the two special bytes values of 0x00 and 0xFF, that corresponds to a blocked input of either one or zeroes, only single bytes is accepted. The first repeat byte is considered to be a non random input.

The response to the non-random input is simply to request the next input byte. This is a loop that will break on a watch-dog reset, as the watch dog counters is (intentionally) not updated in the sample analyse loops.

A serious problem with low-frequency blocking, as described above, do not influence the quality of the final random numbers produced, but as several or most samples will be discarded the output production speed is lowered.

It is OK to still use the unit; a buffer overrun error is produced if the application read to many bytes too fast.

It is possible to gain direct access to this software by issuing the **4R** test command. This command switch the input source from the random number source to the CMD input pin. A non-random input string can be input, to simulate a troublesome disturbance signal, and the module will process this pseudo-input in the normal fashion. This command is for evaluation and test purposes only.

Reset and Watch-Dog circuits

The TRNG9880 have an analogue external watch-dog circuit, that is needed for continuous operation of the unit. The circuit issue a hard reset, that is as good as a power-cycle reset. To test the circuit ground the HEARTBEAT test pad and measure the signal on RESET test pad. A high voltage indicate that the module is in reset. Output is typically a 5Hz wave.

The module also have a microcontroller internal watch dog circuit that, as usual, is implemented using digital electronics, and as a result of this, as usual, may block or lock, and thus do not guarantee a reset. The circuit is used to monitor the operation of the unit, and if there is a software problem, or some format problem in input, or other issue that may occur, run a reset from the internal software. This is implemented by stopping the external signal to the analogue watch dog circuit, that will then hard-reset the chip.

For correct operation the customer's application must accept an occasional TRNG9880 reset cycle. The reset circuit is there to help.

Testing of random numbers

Almost all published and recommended tests for random numbers have the property that they are extremely sensitive to a bit bias of 0/1; that the binary "1" may be of different probability than exactly ½. This can be observed by generating a test string of perfect statistical properties with the addition that the probability of the "1"-bit is 0.499, as an example. Check out witch tests that fail on this input.

However, for any industrial use, or use in gaming such as card shuffling, other properties are much more important: unpredictability, resistance to tampering, the possibility to verify the random number source.

Due to this much effort in the processing of the random numbers is to even out the 0/1 binary bias, as any deviation whatsoever would instantly fail ALL statistical tests. But please remember that this is a property of the TESTS, and several tests have a high accuracy requirement and are extremely sensitive.

Testing of the random number hardware

The unit include a simple information rate calculator based upon a frequency table using 5 bit input blocks. When the frequency table has been filled, an information rate figure is computed using the observed frequency table. A test based upon the information rate have the advantages that

1. it is not sensitive to the 0/1 relative frequency
2. it directly measures the quantity that we are interested in
3. it don't give false positive alarms based upon some probability calculation (like "FIPS"!), and,
4. it would catch any kind of hardware problem or error.

The unit have a **1T** test command, that give access to the raw input bit stream unprocessed. The command is implemented to write a binary output on the CMD port. For test purposes the **1X** test command is recommended, that write the output in HEX. This is better as it is more easy to detect simple serial port communication problems, that may fool the external test software.

TRNG9880 Start-Up Sequence

The error conditions that are observed externally, with optional messages on the CMD pin, are all set to TRUE/ERROR at start-up, and the corresponding tests must then clear the errors before the "OK" message is transmitted.

The output buffer is set to empty, and the most important memory locations are filled with randomness directly from the random number source. Processing then begin, but there can be no output, as all errors are set. During the processing all registers and memory naturally obtain updated and fresh randomness from the hardware. The output buffer begin to fill up. When the output buffer is full, and on condition that all errors was cleared, the external error is reset, and randomness may be delivered to the application.

The Non-Algorithmic Encryption Computation

The non-algorithmic encryption was found in 1995, with the attractive property that it push a computation, that possibly solve it, towards the set of uncomputable functions. Even if the available hardware prevent implementation of the exact theoretical concept, the general idea of using a varying function is attractive in a random number generator. The complexity of the generated function refer to a complexity of languages, and since this apply to any languages of any kind, the complexity is based upon a much more solid foundation than any specific technology, invention, or physical effect.

When we investigate the cryptographic strength of a cipher, we may think of a function that solve it. A typical question would be how difficult it would be to find the solution; how it would be implemented or how much ciphertext would be needed. We note that there is an uncountable continuum of functions, so, in principle, a solution would normally exist.

Unfortunately, only a countable subset of functions are computable, and can be implemented. We can implement a function/algorithm by a software, and there are an infinite number of softwares. They can be ordered in a numerical order of 256 1-byte softwares and then 65536 softwares of 2-byte lengths; etc, and, even if they are very very many, they are still countable, and the countable subset of functions are a zero subset of all functions in an uncountable continuum, and of no chance to cover the set in any way.

For small cipher functions that are algorithms, there are computable solutions for each and every instance or variant of the cipher. But for more arbitrary ciphers, the function that solve them is at random of all functions, and the probability that this specific function just happens to be also computable, is exactly zero, as a point (of solution) cover no area (of functions).

Uncomputable, in this setting, mean that the function is not included in the above mentioned set of countable software(s). We refer to the **"non-algorithmic encryption.pdf"** report for details; also included a valuable list of references of published literature on encryption.

Your first question would be the algorithm of the encryption; sorry, there is none; as the non-algorithmic encryption implement an interpretation and not a fixed specific algorithm. Rather, the input symbol stream is interpreted as a symbolic program of computation, and so the concept of Algorithm should be associated with the random input.

The non-algorithmic computer have a 64 byte memory divided in a high and low half; a pointer select one byte on each half as the active memory cell. There are four 8 bit registers. An "ADD"-sum of all input bytes are also available.

The implementation include a list of 64 "operations" that make a small computation using the input registers. The registers and the memory is updated. Some combination or intermediate result is used as an 8 bit output. The output is taken directly from the operations. The CPU carry flag is also assigned. Some operations assign or updates the memory pointer. A general-purpose nonlinear operation, implemented as two independent balanced 256 byte substitutions, is also available.

During operation an input byte selects one of the 0..63 operations. Each operation produce an output byte. The calculation cannot be reversed, so a cipher implementation would use an invertible mapping controlled by the output bytes. Each operation perform only a small computation and several updates is used in order to produce the final result.

After a description of all these complexities, it may come as a surprise that, in the output, structure is easy to find. This is the Trivial properties of the output, that don’t depend much on the input. The trivial properties comes from that, if the memory and the registers are all random, and we select a random operation, there may still be structure produced by some operations, and a 1/64 part of this would on average be found in the output. Adding all the operations together it would be unlikely that all possible aspects of the output to be exactly balanced.

Software Processing, Output Buffer

The TRNG9880 have a 17 bytes random buffer, that is used to average the random data. This is based upon that a linear operation modulo a word size, if we have independent distributions, yield a more flat distribution as a result. The key-word is independent, and this can be aided by several tricks. One such trick would be to run a simple byte substitution on one input stream and add it to the other stream, that is unprocessed. The byte substitution alters the distribution making the operation more efficient.

The raw input randomness is processed in the non-algorithmic processing engine, and then added to the old value at the current buffer position (one byte). An alternate buffer position, that is not arranged in any consecutive order, is also used. A resulting processed byte is then transferred into the TRNG9880 output buffer, where the new value is added to the buffer. The output buffer, that have a length prime to the work buffer, is also used to further lower statistical errors.

We shall note that using long test sequences and good test software, the buffer length of the buffers, or at least the product of the buffer lengths, can be obtained. If a correlation test is run on the exactly same buffer update bytes, a limited and small statistical deficiency can still be measured. Due to this is the run-length of the 17 bytes buffer not fixed, 15 and 17 steps is run at random as a function of the random number

input. This "de-stabilises" any correlation test, and the problem disappears (almost).

It is my purpose to illustrate, that to reverse-engineer the output from the TRNG9880 extremely costly cryptoanalytical tools must be used.

Each output byte is a sum of

A previous byte of the output buffer
The previous byte in the work buffer
The output from the encryption engine
An alternate byte from another position in the work buffer
The current sample byte, and a sum of previously sampled bytes

All TRNG9880 output bytes are true random bytes that originates in the hardware noise amplifier. This is arranged by simply requiring that the input loops, that read in and process bytes, are run faster than the allowed extraction process, where bytes are updated into the output buffer. This is enforced by an update in the work-buffer loop, that only updates the internal structures, without sending a new byte to the output buffer. Note that if the unit is not run at absolutely 100% of its capacity, each output byte will have corresponding more input sample bytes. The unit always run at full speed internally. At start-up the unit is run a while before the output is opened, to guarantee the complete randomness of the first byte.

In a detailed analysis shall also take into consideration that the input information rate of the hardware will not be 100% * 8 bits/byte but is normally lower, like 95% of 8 bits/byte. This means that the output must run slower that 0.95*input sample frequency, and this is enforced in the module.

Further Reading

The mathematical properties of computer-Add, XOR, and multiplication was first studied in [134]. A Eurocrypt Proceedings at about 1985 made these results available for the more general research community (by re-discovery).

A simple and good statistical test software is the Crypt-X [166].

It is OK to forward more detailed questions to the author at TRNG98. The TRNG9815 contain a much more detailed analysis with test data taken from individual processing steps. Some of this material trnaslates directly to the TRNG9880, the TRNG9815 use a more complex update, though, with a separate block encryption added at the end.

An issue on how to improve randomness quality: The Application can run a circular update buffer using a buffer length not prime with any buffer in the TRNG9880, most simply a prime buffer length. Simply ADD the bytes is a good and simple update algorithm.

To protect against fraud (by us) for a game application, the following algorithm can be run:

Obtain (download) a C software that implement the old IBM DES algorithm (The AES is not recommended). Modify by scrapping the key expasion algorithm, so the key size is 16 rounds x 48 bits = 768 bits.

Generate an one-time 768 bit key and a 64 bit key. Save this secret key in the software of your application.

To circumvent any attempt of controlling the output, generate 64 bits input "message" and an 768 bits "key" from the TRNG9880. Then XOR with your fixed secret key. Then encrypt the resulting XOR message with the XOR key. The resulting 64 bits is your resulting random bytes, that cannot be controlled by any external operator.

Reference:

[123]
Löfgren, Lars
"Computability",
Systems & Control Encyclopedia,
Cambridge: Pergamon Press, 1987

[124]
Löfgren, Lars
"Autology",
Systems & Control Encyclopedia,
Cambridge: Pergamon Press, pp 326-333, 1987

[166]
"Crypt-X 98"
Information Security Research Centre,
Centre in Statistical Science and
Industrial Mathematics,
Queensland University of Technology.
http://www.isrc.qut.edu.au/resource/cryptx/

[134]
Rodin, Gunnar
"Metoder för kryptering av datorlagrade data"
(Statskontoret 1972-05-10)
Institutionen för Informatonsbeandling - ADB
Library code: KTHB1100042873 "2b1155"
Kungl. Tekniska Högskolan, Stockholm, 1972