The SG100 TRNG Platform design described

The SG100 true random number generator consists of two parts;

- a hardware noise generator
- and a software driver.

Between the SG100 hardware and the SG100 software driver is a computer port with a port driver. This port driver could be a serial interface or a USB/serial interface. The important question here is how to cleverly split our mission into two parts: one suitable for processing in the hardware and the other suitable for the software driver; and how to overcome possible problems with the computer port in the middle. These questions affects throughput, quality, and reliability.

On the SG100 generator platform we have chosen to do as little processing as possible in hardware.

This results in less good statistical performance, when measured on the raw analog noise source hardware, but gives us several advantages. The entropy given by the noise source is though high.

If no processing of the noise takes place in hardware it is simple to calculate a figure just how well the device is operating. In "how well" the quantum noise effect functions.

The compatibility is increased. SG100 works on any serial port and on USB devices, facilitating the use of the same hardware on all platforms including embedded systems.

This is Extremely important!

You can analyze the analog characteristics of the noise source separately from the post-processing part. This is not possible with devices that stuffs the analog noise source and the post-processing part inside the same box. There is no practical possibility to know for the user if there is a functioning noise source inside in this scenario! There is no way you can statistically "test out" if there is a pseudorandom source or a true random analog source inside the box. Trust but verify.

Design Considerations for Hardware.

Resistance against Radio Frequency (RF) fields has been in the specification from the beginning of our product development cycle. No customer has to fear that the operation of the SG100 generator can be intentionally influenced from any external RF-field. This has been accomplished by using a noise generating process with a high output level. The SG100 generator has a built in RF-field filter.

The hardware outputs is an irregular square wave that is feed into the computer serial UART or serial to USB converter chip inside the EVO line of devices. This samples and converts the input stream into digital form. During this process the computer UART selects which parts of the noise stream that will be interpreted as start bits and which parts that will be ignored as stop bits. The decision of when to interpret the stream as "1" or "0" is up to the UART. You may use any bitrate below 100,000 that is accepted by the serial interface driver.

Not converting the SG100 platform output into byte-serial form makes interception of the serial stream very difficult, as it is unknown what bits are selected as startbits and as the sampling in the UART is somewhat different on all computers.

Hardware Properties

- High resistance against power fluctuations.
- Low influence of vibrations and temperature
- High resistance against external electromagnetic fields, Information feed into computer difficult to intercept.

Software Properties

- Easy to use API interface, fl Immediate action if the device fails. SDK
- Fast response to the calling process.
- Interface for multiple processes reading noise.
- No cryptographic or statistical weaknesses.
- Do not deliver low quality noise when first called or if called repeatedly.

Software Development Kit PSDK

The SDK also called PSDK separated in two parts:

Recommended
Use the source code driver for new development.

- A pre-built driver for the Windows operating system is delivered with the SDK. The API of the driver (*.h file) is included as well as compiled demo programs in C/C++. Using the provided compiled demo program you may extract noise to a named file to facilitate an easy interface with almost any statistical or security product. The driver may be linked directly into the EXE of an OEM product.

- A source code Linux/Windows driver stack is also provided. This could be used for all versions of Linux including Linux x86_64 and Windows 10, Windows 7, Windows Vista, Windows XP.

Below is the general processing steps described. It is practically the same in both versions, pre-built and source code version driver stack.

The provided SG100 driver DLL operates the hardware and checks for hardware errors.

A wide range of hardware errors can be detected and action taken accordingly. If the device fails any pending call for noise will be returned with an error status. If the user disconnects the SG100 generator a signal is given to the calling application to let the user reinstall the noise generator without any influence on the running application.

The driver do a continuous statistical check of the hardware. The driver then calculates the rate of information obtained form the SG100 generator. If the rate falls below 70% the device is considered faulty. A rate above 96% is acceptable without further action and if a rate lower than 96% is obtained this will be remedied by reading the noise stream twice, giving the driver access to the double amount of input. Reading twice, if necessary, will give us only half overall throughput.

The hardware, built according to a minimal noise-processing criteria, cannot create complex output correlations by itself. We will have a statistical bit bias from the hardware, with "1" and "0" not occurring with exactly 50%-50% distribution, and also a correlation between adjacent bits (and a correlation between the start bit and the first bit). We see that there cannot be any high complex correlations because the hardware lacks memory. After a short period of time a previous noise stream cannot influence the current noise stream.

Suppose we read four bytes and form a 32 bit integer A. Suppose we know that the current information flow is 94%. We then read a second 32-bit integer B, at some other point in time, maybe a few seconds later. We now know that A and B are independent and random, and also that we have statistical deficiencies, as indicated above. To form a 32-bit integer C with almost 100% information rate it is sufficient to add A and B using 32-bit binary add (with carry) C=A+B. We may picture this by an arrow A pointing on the circumference of a circle divided into 2A32 segments. The selection of A has some bias, making not all the 2A32 segments equally probable. If we now turn the arrow B steps forward it is visualised that if the bits of A and B has the independence property, above, the number C will have surprisingly good statistical properties. Note that 2*94%=192%; the number C, consisting of 32 bits, clearly cannot store any more information than 32 bits (100%).

In this way, using a similar but more complex "adding", the driver can guarantee that the minimum information flow is 96%. Using a SG100 developer package the reader may obtain the raw SG100 output and verify this for himself.

The software driver has a buffer of noise ready for reading to facilitate fast response when called. Currently this has been set to 32.000 bytes. The buffer also has an additional purpose, here is the pre-processed input from the driver "whitened" to enable the output stream to pass any statistical or cryptographic test. This is done using a combination of statistical and cryptographic techniques. This relieve the customer, almost regardless of application, from the need to further improve the SG100 driver output.

The software driver uses process synchronisation allowing multiple processes to read noise simultaneously. The synchronisation also blocks processes if the noise buffer becomes empty or during an initial start-up period before the noise buffer has been thoroughly "whitened". When the noise buffer becomes empty the driver reads tree times the buffer size to allow filling the buffer with a maximum information content. We have seen that the minimal information input rate is 96%. The reader should note that it is possible to read this minimum information content only if an application asks for a very long continuous string of noise, and that the 32.000 first bytes of that long string will most probably be of 100% quality. To experimentally obtain the (maximum) 4% lack of information, the output must be intentionally processed to break the cryptographic operations, which will be most difficult mainly because of the small statistical deficiency searched for. This is practically infeasible for the string lengths that the drivers has been designed for.