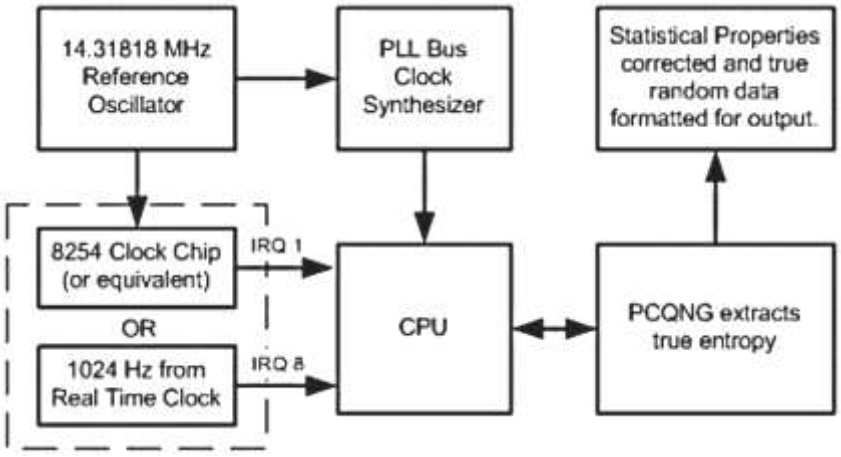
Design Principles and Testing of the PCQNG 2.0 Device

The PCQNG True Random Number Generator (TRNG) represents a leap forward in the production of high quality, True or Quasirandom (algorithmically extended) numbers for use on your Personal Computer. The PCQNG extracts the True Physical Entropy being constantly generated by hardware already present in your PC to produce a continuous, high-speed stream of high-quality random numbers containing nearly one bit of actual entropy per output bit. The generator's output is made easily available through the PCQNG software in a number of user selectable modes.

Many suggestions have been made for possible methods of obtaining entropy from personal computers. These include keyboard timing or intervals, following mouse movements and timing, and air turbulence in hard drives. There have also been many vague ideas of using the various timing signals from subsystems in the PC, such as the BIOS clock, the serial card and the real time clock. In addition to these are the progressively more restrictive ideas that require certain additional hardware to be attached to your computer. These include measuring ping response time on computers which are networked, digitizing static from a radio receiver through the sound card, and finally, even using digitized images of Lava Lamps.

All these ideas suffer from one or more limitation or theoretical flaw which have made them virtually unusable as practical sources of random numbers. The most serious defect in all the methods proposed is that there is no theoretical basis for precisely quantifying the actual entropy content of the output sequence. There is usually no acceptable method of predicting, or even measuring the statistical properties of those sequences within necessary confidence levels. The output bit rate is sometimes measured in bits per minute and some of the methods require the active, physical participation of the PC user (you) to produce any output at all.

The specific implementation of the PCQNG 2.0 will be described with reference to the following Block Diagram:



The fundamental source of entropy in the PCQNG is the same as in virtually all types of TRNG devices. These devices consist of two distinct parts: one is a fluctuating signal source having an unpredictable temporal evolution; the second is a periodic signal, causally independent of the first signal, that is used to generate a timing sequence for sampling the first signal.

The possible exception to this is a purely quantum mechanical device using superposition or entangled states and the collapse of their wave function at a detector. Another generator that is believed to be random is the purely mechanical generator, such as the type used for most lottery drawings. Mechanical generators rely solely on complex initial conditions and chaotic transformations to produce the appearance of randomness. Precise quantification of bias in such a system is probably not possible due to changes in the mechanical components over time, although there is reason enough to believe that bias will exist.

Referring to the Block Diagram of the PCQNG, a low-frequency sampling signal is derived by dividing the reference oscillator signal to ultimately produce an interrupt (IRQ1) that, in turn, causes the CPU to read a count of its core clock (Time Stamp Counter), and send the instantaneous count to the PCQNG program. This happens at a rate of about 1000Hz in a single processor and 1024Hz in a multi-processor system. Note that in a multi-processor system, the interrupt is derived from the Real-Time Clock (RTC) on IRQ8. Multi-processor systems will therefore produce a slightly higher true entropy rate than is predicted for a single-processor system. Although the CPU clock is ultimately derived from the reference clock, there are at least two levels of Phase-Locked Loop (PLL) frequency multiplication that raise the original 14.318MHz to the typical 1- 3GHz core frequency. The CPU clock will contain independent noise components that are produced by thermal and shot noise and power supply noise in each of the PLL frequency multipliers.

The reference oscillator is used to produce several clocks used by the computer: a BIOS clock (1.193182MHz), an FSB or bus clock (66.67MHz upward in integers multiples 33.33MHz) and numerous others for use on the PCI bus, ISA bus, USB, serial port, key board, etc. These are not directly used by the PCQNG.

To obtain an accurate calculation of the total, true entropy available requires an accurate value of the total rms cycle-to-cycle clock jitter in the CPU's core clock, as well as the distribution of the jitter. This distribution closely approximates Gaussian, as obtained by direct measurement, with little loss of accuracy. The total rms jitter may be estimated by a combination of measurements and theoretical calculations, or it may be obtained by direct measurement.

The estimation of the total rms jitter in the core clock will be simplified for purposes of this description. The jitter in the low-frequency sampling signal is assumed to be zero, although this is not effectively true due to additional variance being introduced by completion of the CPU's execution of the current instruction at the time the interrupt arrives. Also, the jitter produced by the bus multiplier (core clock multiplier) will also be assumed to be zero. Again, there is some additional jitter produced in this second PLL multiplier, although it is constrained to track the average bus phase fairly closely. In addition, most newer clock synthesizers utilize spread-spectrum timing (SST) to minimize radio frequency interference (RFI). SST is a frequency modulation imposed on the bus clock with a modulation frequency of 33-50KHz and a frequency deviation of 0.5-2%. Although SST is not yet universal, current trends indicate it soon will be. Because all computers which may run the PCQNG do not use SST, this additional source of jitter is not included in the present analysis.

Finally, with these simplifications the entire source of jitter will be considered to arise in the first PLL multiplier that raises the 14.318MHz reference frequency to the bus frequency of 100-400MHz (in 2003). Specification sheets for both Cypress and Intel clock synthesizers show maximum cycle-to-cycle jitter in the bus clock to be 200-250ps rms. A maximum value specification is usually about twice the typical value specification, yielding a probable average value of 100ps rms. This cycle-to-cycle rms value must now be converted to a total rms jitter spanning the approximately 1ms sampling period. This is done by calculating the number of high-frequency cycles that span the sampling period, taking the square-root of this number and then multiplying the result by the cycle-to-cycle value. Far a comprehensive analysis of PLL jitter see *Modeling Jitter in PLL-based Frequency Synthesizers.* (Kundert, 2003)

Although the jitter is measured by reading the core clock counter, the core frequency is not actually available in any system using interrupts to read the counter. This is a result of the sequence of events caused by a hardware interrupt: First a level shift or pulse is sent to the programmable interrupt controller (PIC). The PIC checks for other pending interrupts and their priorities. Then the PIC signals the CPU that there is a pending interrupt. The CPU responds by requesting an interrupt vector from the PIC which points to the location of the appropriate interrupt service routine. The interrupt vector is sent to the CPU over the bus, and no matter what the core frequency or when the interrupt actually arrives, the CPU cannot begin the interrupt service routine (ISR) until the next cycle of the bus clock. This sequence of events produces a time quantization with an effective rate of 33.33MHz on all recent PC's.

Taking this measurement quantization into account: 33.33×10^6 times 0.001= 33.33×10^3. The square-root of this number of cycles is 182.6 times 100ps yielding a total jitter of 18.26ns rms: the cumulative jitter spanning each sampling period. This is equal to 0.609 cycles of the 33.33MHz quantization frequency and will be called the

normalized jitter. There is no difference in the resulting entropy calculation which signal is considered to contain the jitter.

Calculating the entropy is a straight-forward numerical procedure once the normalized jitter is determined. The algorithm first calculates the probability of actually finding a high value in the sampled signal where one is expected. This is most easily done by assigning the jitter to the lower frequency (LF) signal and then integrating the probability distribution function (PDF) across all time periods where the PDF overlaps the high state of the higher frequency (HF) signal. The PDF is centered around the rising edge of the LF signal, and in this case the distribution is Gaussian with standard deviation equal to the normalized rms jitter. The integration is repeated over a large number of equally spaced points exactly spanning one high state, or half cycle, of the HF signal. These results are then averaged.

This result will represent the true average if the ratio of the HF to LF frequency is an irrational number. In that case, the rising edge of the LF signal will appear uniformly in all phases of the HF signal; an assumption of the algorithm. The actual HF to LF ratio, in conjunction with the size of the normalized jitter, yields results indistinguishable from those expected from this assumption. HF to LF ratios which are integers or contain rational fractions may produce significantly different results, especially when the normalized jitter is small.

The true entropy may now be directly calculated from the probability of predicting a high state or "1", p(1), and the probability of predicting a low state or "0", which is just p(0) = 1-p(1). These two probabilities are used in Shannon's entropy equation:

$$H = -\frac{1}{Ln2}[p(1)Ln\,p(1) + p(0)Ln\,p(0)]$$

The Table 1 was calculated using various normalized rms jitter values:

JITTER	ENTROPY
.5	.99998
.4	.9991
.3	.9864
.2	.8999
.1	.6333
.05	.4014
.02	.2039
.01	.1180
.005	.0667

TABLE 1

It has been determined theoretically and empirically that when the LF signal is sampling a multi-bit binary counter, counting the cycles of the HF signal, each successively more significant bit effectively contains just one-half the normalized jitter of the previous bit. In our example, the LSB has a normalized jitter of 0.609. This would result in an entropy of almost exactly 1.000. If only the one bit were used, the rest of the entropy would be wasted. The next more significant bit would contain a normalized jitter of 0.305, representing an entropy of 0.98. Table 2 shows the entropy contained in each of the 8 lower bits:

BIT	JITTER	ENTROPY
0	.609	1.0
1	.305	.98
2	.152	.80
3	.076	.53
4	.038	.34
5	.019	.20
6	.010	.11
7	.005	.06

TABLE 2

The total true entropy available in the 8 LSB's is 4.0 bits. This result is obtained for a sample rate of 1000Hz, which yields a final entropy rate of about 4Kbits/second. Even using a more conservative jitter value of 50ps (instead of 100ps) results in a calculated entropy rate of 3Kbits/second, while actual measurements on Intel- and AMD-based PC's from 133MHz to 1.7GHz typically yield 5-6Kbits/second. The somewhat higher measured entropy is due to the additional jitter sources that were disregarded in this simplified estimate. Results from PC's with higher CPU core frequencies - above 1GHz - will generally produce progressively higher measured entropy rates; some approaching a maximum of nearly 8Kbps. Measurements of the actual (Gaussian) probability distribution function of the sampled data indicate that entropy rates above about 5Kbps are due to chaotic sources rather than true entropy. That is, jitter caused by interactions of the interrupt timing with the Windows OS, as well as with the various instruction completion times in the CPU. It is highly unlikely that the chaotic processes of one computer could be duplicated in any other computer; however, no chaotic process can be considered to produce true entropy.

It is essential to understand the theoretical source and magnitude of the entropic process used for producing any true RNG. For illustrative purposes one might consider, as has been suggested elsewhere, using the BIOS clock and the RTC as a source of random bits. The 1.19318MHz BIOS clock is derived by dividing the 14.318MHz reference clock by 12. This will result in a cycle-to-cycle jitter of 70ps x SQRT(12) = 242ps rms (based on the typical jitter of the reference oscillator). At a sampling rate of 16Hz the normalized jitter is 0.0789 and the calculated entropy is 1.31 bits per sample, yielding a total entropy rate of just 21bits/second.

The raw core clock counter values are processed by the PCQNG software to isolate and extract the bits containing the true entropy. This is simply the lower 8 bits after adjusting for the 33.33Mhz quantization. A special mode allows the raw bits to be directly accessed for confirmation of their entropy content and statistical properties; or, they may be processed by the user's preferred methods. These bits are further processed in one of two ways, depending on their intended use:

1) Eight entropy-containing bits per sample are sent bit-wise through a very powerful statistical stirring program which mixes the entropy in the output bits and corrects any statistical defect without changing the total entropy content. The output bits are then assembled into 8-bit words and sent word-wise through another stage of statistical stirring. The final output bit-rate is selectable through mode settings which allow a range of output rates from 1bit in 8 samples to 32 bits per sample. Output rates at or below the actual available entropy rate (about 4Kbps) ensure true randomness: output rates above the true entropy rate are produced by "stretching" the available entropy in the stirring functions while still maintaining virtually perfect statistical properties. Since the higher bit rates are derived from stretching a continuous stream of true random bits, there is no limiting period and a constantly changing seed, making cryptographic attack prohibitively expensive.
2) Eight entropy-containing bits per sample are sent bit-wise through a minimal statistical stirring program which mixes the entropy in the output bits and corrects statistical defects without reducing their true entropy. These eight bits are exclusive-ored to produce a single output bit at a rate of 1000bits/second. These random event generator (REG) bits are intended for special applications relating to direct mind-machine interaction (DMMI) and other parapsychological experimentation. Decades of laboratory testing

and subsequent meta-analyses indicate a small but persistent effect on the statistical properties of true random generators in response to operator intention. The statistical properties of REG-mode bits - in the absence of operator intention - are calculated by a combination of theoretical and empirical methods to deviate from perfect randomness by less than 5ppm in both 1/0 balance and autocorrelation.

TESTING THE PCQNG

The PCQNG has been tested extensively by well-known programs such as Marsaglia's DIEHARD suite of tests and most of NIST's new tests. ComScire's tests (ComScire RNGmeter), which are more stringent than either of these, have also been applied to bit counts up to 10 billion. The PCQNG has not failed any test for randomness. All other generators we have tested fail ComScire's tests, with the exception of one type of pseudorandom generator.

REG bits have been tested directly only to 100 million bits due to their much lower bit-rate. The final statistical properties of REG bits are determined by measuring the bias and autocorrelation of bits derived from smaller blocks of samples and applying known equations for calculating the resulting properties of groups of bits which are exclusive-ored.

The RNGmeter is now available as a general testing program including a detailed description of its tests.

NOTE: The PCQNG must be run on a fifth generation (Pentium class) or higher computer because the core clock counter was not available in earlier CPU's. PCQNG 2.0 will run in Windows98/ME/XP/2000/2003 (Not for WindowsNT/95).