Random Number Generator

Random Numbers generated by Software

R01AN4040ED0102
Rev. 1.02
November 28, 2018

1 Introduction

This application note describes how to generate a Random Number by software, taking the advantage of the imprecise internal Low-speed on-chip oscillator (15 kHz) and independent internal High-speed on-chip oscillator (32 MHz) of RL78 micro controllers.

This Application Note shall give an example how a random number may be generated by software. This Application Note references to a RL78/F14 micro controller (R5F10PPJ). In general, each other micro controller equipped with two timers, clocked by independent oscillators, can be used. In case of random number generation, a big tolerance of the oscillators is an advantage.

The algorithm and handling for its generation has to be adapted to the dedicated application and available resources.

There is no certification for that Random Number generation nor calculation of achievable entropy nor official test release for the excellence of scrambling. Nevertheless, tables are implemented which show examples of generated Random Numbers and its entropy.

1.1 Target Devices

RL78 16-bit micro-controllers

2 Table of Contents

3 Usage of Random Numbers

In some applications, growing security requirements makes it necessary to use random numbers for safe handling of data. This Application Note may help to generate an unpredictable key for encryption.
As an example, his random number can be used for data encryption, which is intend to send encrypted data between two micro controllers.

Micro controllers are designed for high precise, reproducible operation. A specific stimuli has to produce a predictable outcome. A micro controller is not designed to generate any random effect. The user expects same functionality under each condition. Differences in the power supply voltage or temperature shall have no influence to the micro controller. A precise CPU / peripheral clock is requested. Especially, when data has to be transferred to an external component e.g., via CSI, UART, LIN or CAN protocol a precise clock frequency is mandatory.

In contrary, generation of a random number needs unpredictable effects. So it is not so easy to generate real random numbers using a standard micro controller with acceptable efforts.

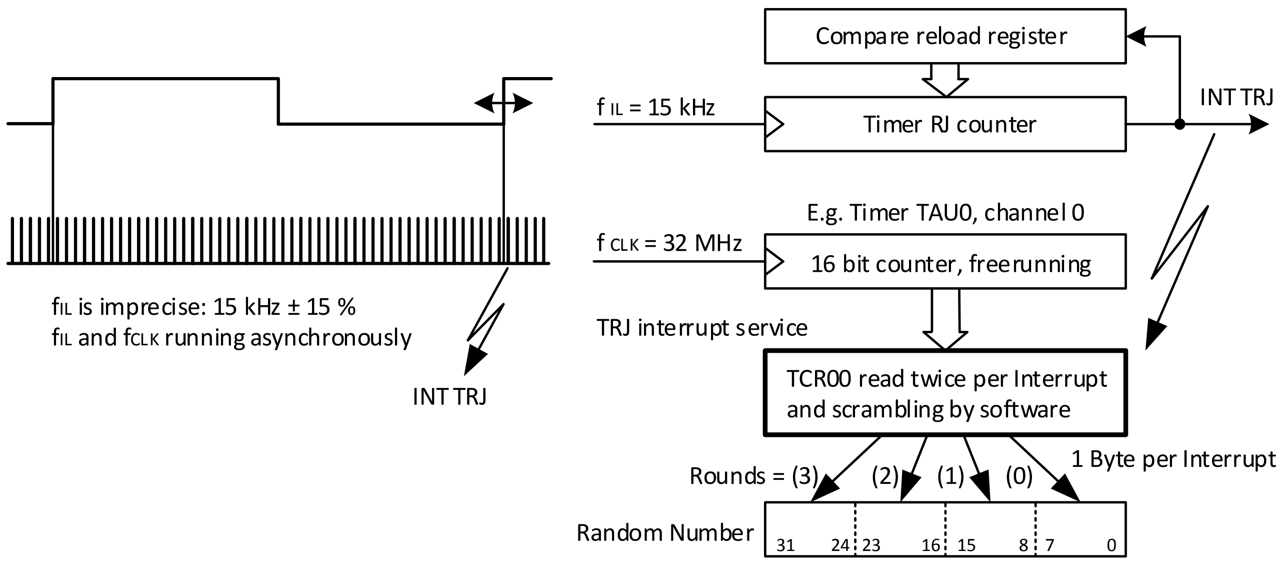
4 Random Number generation by software, using two internal clock sources

4.1 Overview of the generation of Random Numbers in this Application Note

Beside the precise internal high-speed on-chip oscillator and the possibility to connect a high precise quartz externally, RL78 products have implemented an internal Low-speed on-chip oscillator (LOCO) with a nominal frequency of 15 kHz. Its tolerance is ± 15%.

Using LOCO´s tolerance, the effect that LOCO and peripheral clock are asynchronous, several timer readouts, an undefined wait time and a special kind of scrambling of the values read from timer counters gives the possibility, to generate a random number with a good entropy.

Figure 1: Principle of the Random Number generation by hard- and software

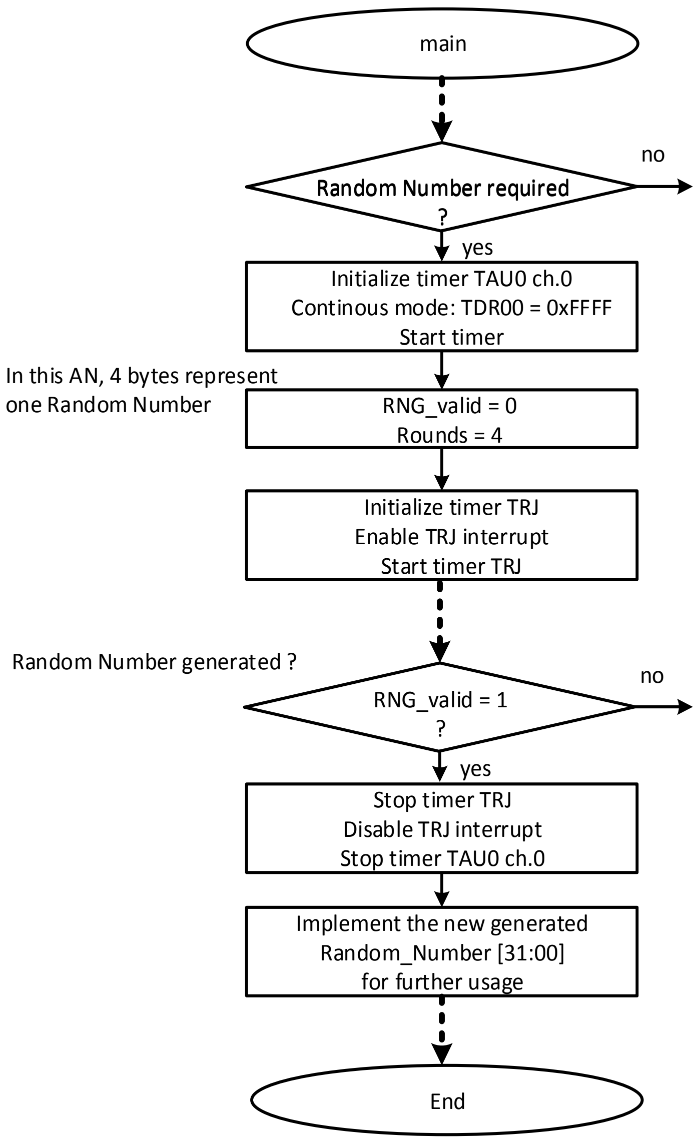

fIL is imprecise: 15 kHz ± 15 %
fIL and fCLK running asynchronously
INT TRJ
Compare reload register
f IL = 15 kHz
Timer RJ counter
INT TRJ
E.g. Timer TAU0, channel 0
f CLK = 32 MHz
16 bit counter, freerunning
TRJ interrupt service
TCR00 read twice per Interrupt and scrambling by software
Rounds = (3)
(2)
(1)
(0)
1 Byte per Interrupt
Random Number
31 24 23 16 15 8 7 0

4.2 Preparation for a Random Number generation (4 bytes assembly)

When a Random Number is required for any security action, initial settings have to be made. The timers have to be initialized, timer TRJ interrupt has to be enabled and the generation of the Random Number has to be started. The generation is driven by interrupt. When finished, the flag RNG_valid = 1 is set.

When RNG_valid = 1, a new Random Number has been generated, the used timers can be stopped and the Random Number is ready for further usage.

Figure 2: Initialization for a required Random Number

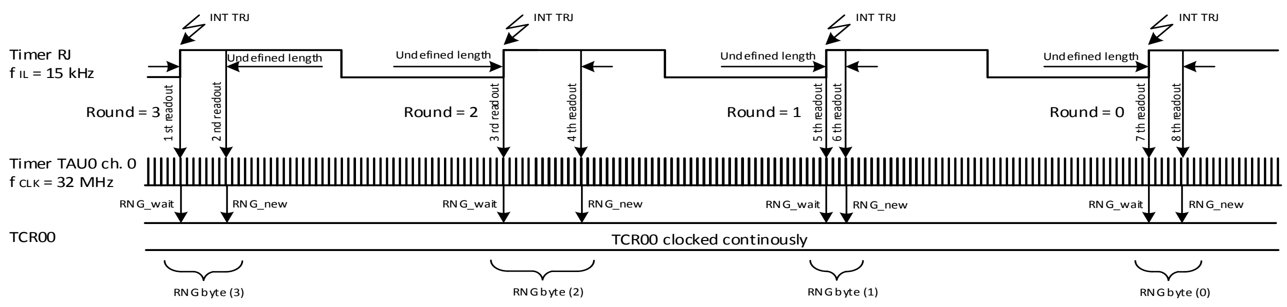


4.3 Corporation of timer TAU0 and timer TRJ (4 bytes assembly)

When a random number has to be generated, timer TRJ and timer TAU0, channel 0 are initialized and started. Timer TAU0, channel 0 is set as free running timer (TDR00 = 0xFFFF) and clocked by the precise CPU and peripheral clock fCLK.

Timer TRJ is clocked by the unprecise internal Low-speed on-chip oscillator (LOCO) clock fIL. In this example, TRJ reload register is set with value 1 to have an acceptable short time for Random Number generation. Timer TRJ interrupt is enabled.

Figure 3: Timing constellation timer TRJ and reading of TCR00



4.4 Scrambling by software within the timer TRJ interrupt service routine

Overall, the timer TRJ interrupt routine is entered four times to generate one 32 bit Random Number. Each interrupt generates ¼ part (1 Byte) of that Random Number.

When the timer TRJ interrupt occurs, its interrupt service routine is entered. The timer TAU0 channel0 (TCR00) is read first time in this interrupt service sequence. The lower 8 bits of TCR00 are stored and its value is set as a wait time of undefined length. Timer TAU0 ch.0 and timer TRJ are not stopped, they keep counting. The CPU waits up to the undefined wait time has elapsed and TCR00 is read again.

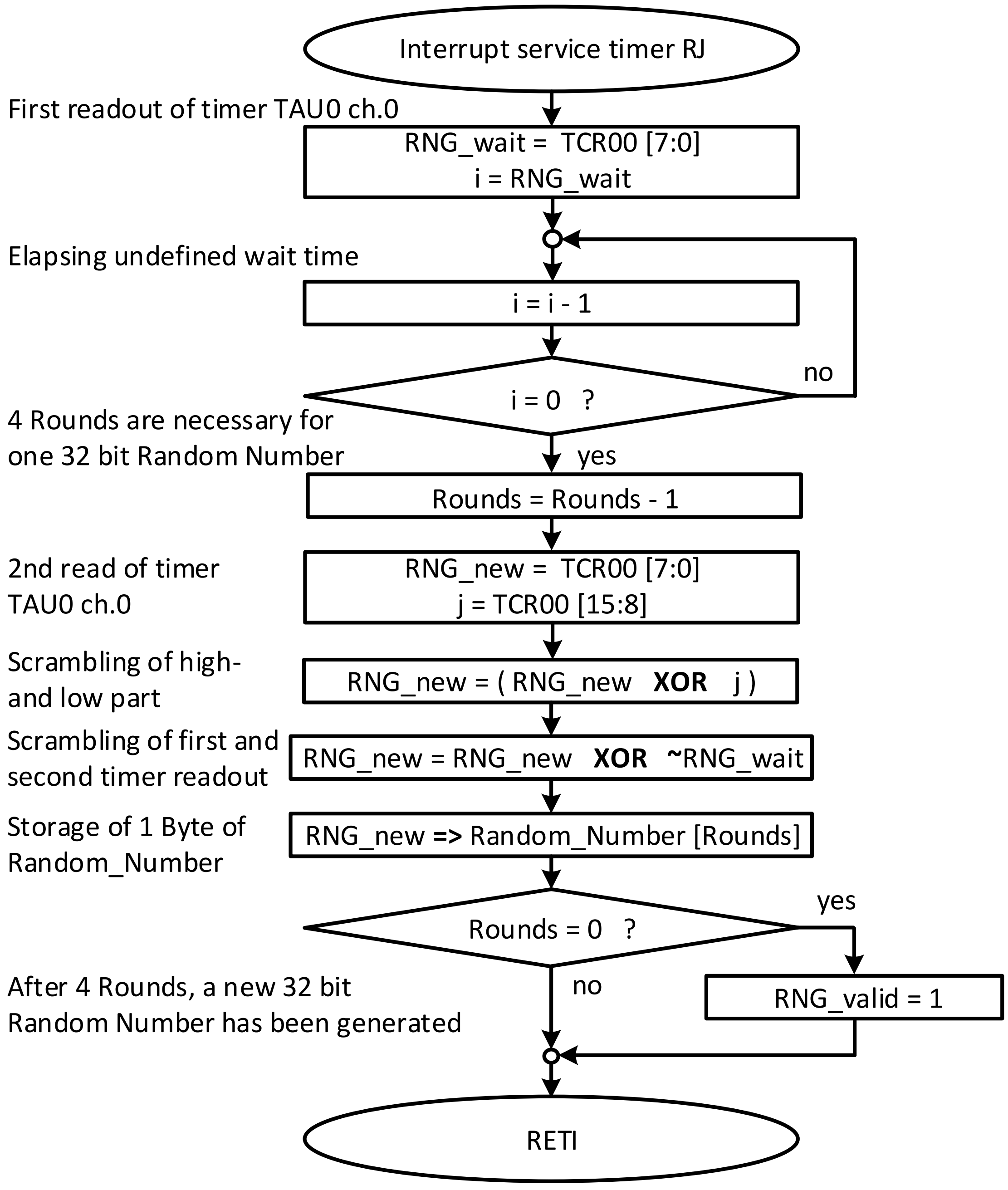
Using the two TCR00 read values within one timer TRJ interrupt service, the following scrambling is executed:

RNG_new [7:0] **=** TCR00_2nd read [7:0] **XOR** TCR00_2nd read [15:8]
XOR inverted (TCR00_1st read [7:0])

This byte RNG_new [7:0] is stored as ¼ part of the 32 bit Random Number. The location of the byte depends on the entering number of the four interrupt entries, necessary for 32 bit Random Number.

After the fourth and last interrupt service execution, a marker is set that the main routine can stop the timers and the new generated Random Number can be used for further operation.

Figure 4: Timer TRJ interrupt service: Read TCR00 twice and scrambling for one RNG byte


Interrupt service timer RJ
First readout of timer TAU0 ch.0
RNG_wait = TCR00 [7:0]
i = RNG_wait
Elapsing undefined wait time
i = i - 1
i = 0 ?
no
4 Rounds are necessary for one 32 bit Random Number
yes
Rounds = Rounds - 1
2nd read of timer TAU0 ch.0
RNG_new = TCR00 [7:0]
j = TCR00 [15:8]
Scrambling of high- and low part
RNG_new = (RNG_new XOR j)
Scrambling of first and second timer readout
RNG_new = RNG_new XOR ~RNG_wait
Storage of 1 Byte of Random_Number
RNG_new => Random_Number [Rounds]
Rounds = 0 ?
yes
no
RNG_valid = 1
After 4 Rounds, a new 32 bit Random Number has been generated
RETI

5 Resources for generation of an unpredictable Random Number

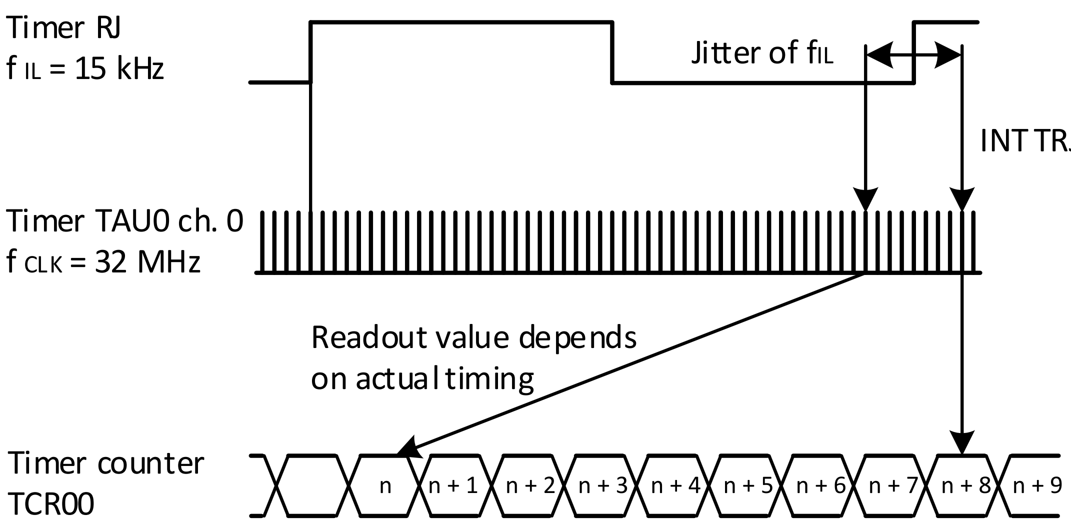
5.1 Frequency jitter improves Random Number entropy

The clock sources, used for Random Number generation have to be independent and asynchronously from each other.

In the micro controller specification, frequency tolerance is given for each one of the internal oscillators. This tolerance is valid for the whole VDD voltage range, temperature range, over the whole lifetime and depends on production window. In practice, within a stable surrounding, under constant external influences (VDD is constant, temperature is constant), within short time, the actual given frequency will be nearly constant. Means, in a constant surrounding the tolerance value, mentioned in the data sheet, has only minor influence for a variation of frequency.

During operation, each oscillator has a frequency jitter. This jitter cannot be influenced from external and is always happen. In case of Random Number generation, a frequency jitter will improve the entropy.

Figure 5: Influence of frequency jitter on timer readout

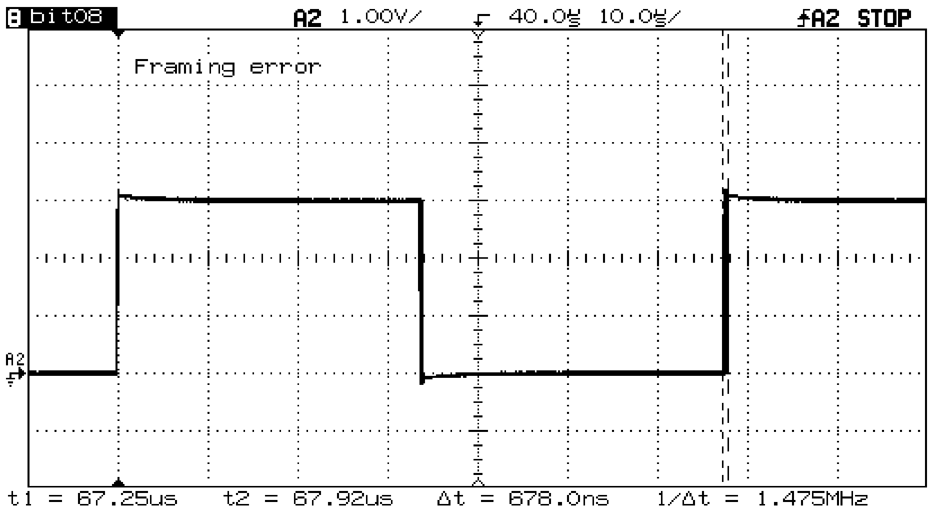


Due to the frequency jitter of the internal low-speed oscillator, the value in timer counter TCR00 differs, depending on when the next rising edge triggers the interrupt INTTRJ for reading TCR00. Due to the accidentalness of the occurrence of the rising edge, the value in TCR00 is unpredictable.

5.2 Investigation of RL78/F14 internal low-speed oscillator´s jitter range

Below, the frequency jitter has been measured at a single RL78/F14 micro controller at room temperature, supplied with VDD = 3V. This measurement shall give an impression for the influence of the jitter for such application. This measurement is no guaranteed data and might be used for reference only.

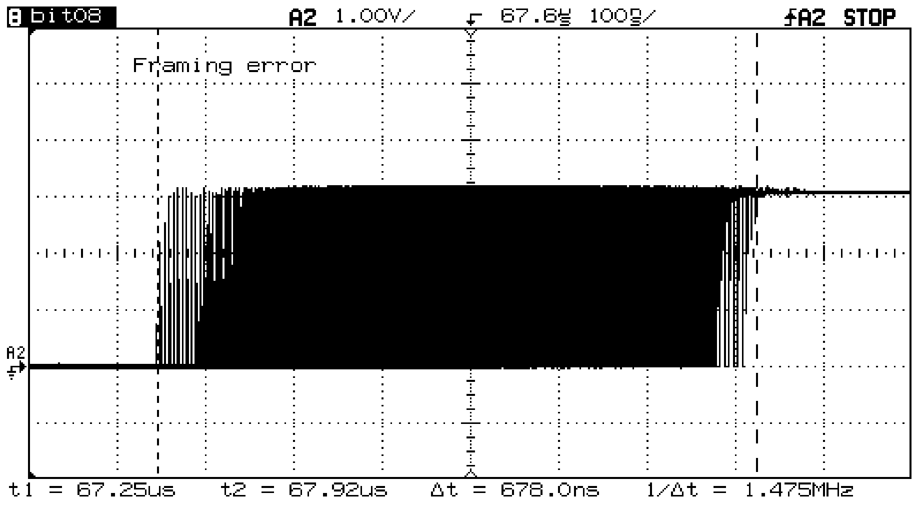
Figure 6: Measurement of a 15 kHz cycle of the internal low-speed oscillator



A2 = P140/PCLBUZ0, output of the internal Low-speed-clock fIL = 15 kHz

The first rising edge of each clock cycle is the trigger for each measurement. The second rising edge is observed for the timing tolerance of its occurrence (jitter).

Figure 7: Zoom of a bundle of second rising edges, showing the jitter range

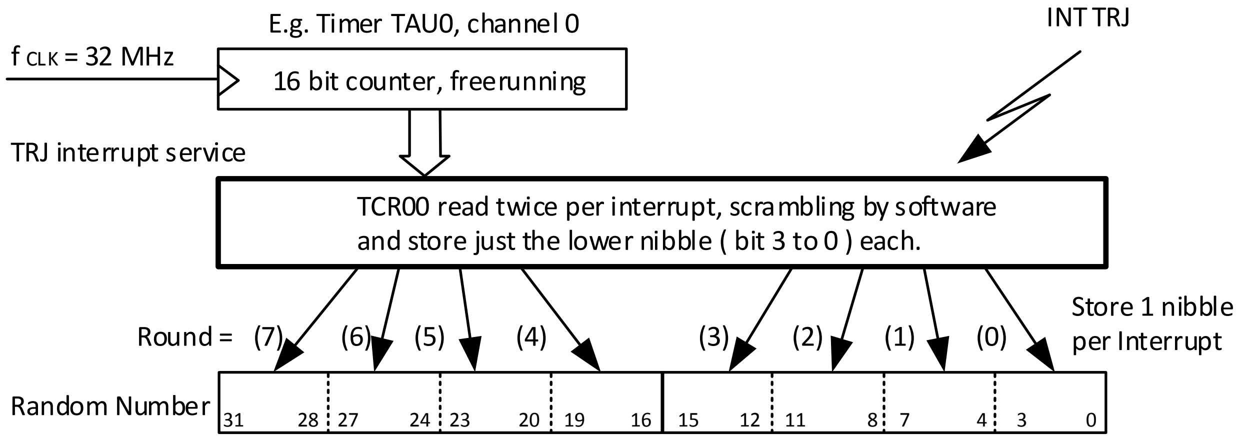


At this micro controller, a jitter range of 678 ns is measured. During 678 ns, the timer counter TCR00 is clocked 678 ns / 31 ns ≈ 22 times. Means, the content of TCR00 may differ of 22, caused by the oscillators jitter only. So, the value of the lowest 5 bits of TCR00 is unpredictable and may change from clock cycle to clock cycle.

5.3 Improved unpredictable Random Number caused by jittered values exclusively

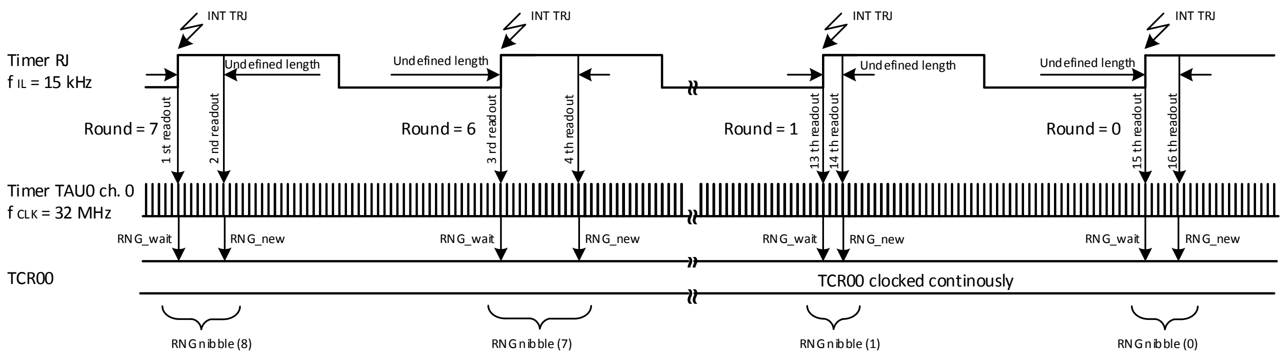
As stated above, from theoretical point of view, the lowest 5 bits of each read value is influenced by jitter. For further improvement of the Random Number entropy, the Random Number might be assembled with jittered values exclusively.

Figure 8: Random Number generated by 8 nibbles assembly (bit 3 to 0 each)


E.g. Timer TAU0, channel 0
f CLK = 32 MHz
16 bit counter, freerunning
INT TRJ
TRJ interrupt service
TCR00 read twice per interrupt, scrambling by software and store just the lower nibble (bit 3 to 0) each.
Round = (7) (6) (5) (4) (3) (2) (1) (0)
Store 1 nibble per Interrupt
Random Number
31 28 27 24 23 20 19 16 15 12 11 8 7 4 3 0

Doing so, for one 32 bit Random Number, 8 nibbles are necessary. So, the time for the generation of one Random Number needs 8 cycles of the Internal low-speed on-chip oscillators f_{IL} (15 kHz ± 15%) clock frequency.

Figure 9: Timing diagram for operation of 8 nibbles assembly



5.4 Usage of available resources in RL78/F14 within this AN

Internal low-speed on-chip oscillator f_{IL} (15 kHz ± 15%) is used as clock source for timer TRJ.

Timer TAU00 is clocked much faster (32 MHz ± 2%) than timer TRJ by an independent second clock source.

Timer TRJ and timer TAU00 are running asynchronously.

The lower 8 bits of counter TCR00 readout value is used for an undefined wait time between first and second readout.

During whole Random Number generation, timer TAU0 ch.0 is not stopped. There is no defined start value of TAU0 ch.0 in between.

For one random byte, counter TCR00 is read twice with an unpredictable wait time in between.

Several arithmetic operations are executed using first and second readout value for scrambling of one byte.

Especially the lower 8 bits of the 16-bit timer counter TCR00 are used for Random Number generation, due to the lower bits will change much more often than the higher part.

A 32 bit Random Number is assembled by generation of 4 Bytes separately, taking advantage of additional timers drift during RNG generation by chance.

The Random Number is stored in the RAM. It is possible to specify uninitialized RAM cells (__no_init) for additional arithmetic operation of the not initialized reset values with the newly generated Random Number.

This Application Note discards this option reading uninitialized RAM to avoid unnecessary RAM ECC error detection.

The timers are not blocked for this task. When a Random Number has been generated, the timers can be used for other tasks.

6 Analysis of achievable Random Numbers entropy by 4 bytes assembly

6.1 Random Numbers generated one after each other

In this example, after reset release, a sequence of 40 Random Numbers each (RNG batch) are generated using the principle and scrambling algorithm, described in this AN. The 40 Random Numbers are generated continuously, there is no reset or halt in between, the timers keep running.
In each column, in the first row above, left side, the first generated random byte is shown. Bottom line, 4th byte right is the last generated random byte of the batch of 160 bytes.
This is repeated six times, documented in the six columns below.

Table 1: Six batches of 40 Random Numbers each, generated one after each other

1st RNG batch	2nd RNG batch	3rd RNG batch	4th RNG batch	5th RNG batch	6th RNG batch
56 96 DF D8	45 AA CE 1A	FE E7 72 5B	49 D7 99 14	28 8E 43 9D	7F 10 C9 49
EC 7B 10 E4	E5 8C 16 C3	99 38 71 4F	98 CB 64 D9	D0 2D 81 F6	1C 87 53 0B
9D EA 27 B2	40 98 79 10	EF 88 55 C9	77 B7 02 B9	FF 3F 79 AE	A0 AE FC AD
C4 50 E3 FC	AC 15 14 A8	AD 00 57 2E	80 D0 C9 42	77 3C FA 77	74 F4 B5 4A
10 D5 6C 93	66 32 ED D9	CB 6B F4 27	80 F3 C7 69	1C D0 99 EC	07 70 E0 D9
26 30 A6 75	A6 54 BD B4	77 20 A3 7F	21 BC 70 4D	DA 44 32 D6	A6 53 BA 9F
1A D4 29 12	E9 F8 9E 2A	74 AD 33 4D	BE 7C 29 84	04 62 92 7C	33 D3 8A F3
36 8B EA 8E	22 78 CC 04	10 DD A0 4F	74 FB B3 96	D2 4E B4 D0	A0 24 6D B4
45 7E CE 0B	42 E6 6E 3C	85 EC 15 E3	CE 18 0D 86	98 67 9C EE	1E 55 EC 78
50 EC F1 AA	65 FC 56 E1	84 2D C9 58	DB 37 96 08	1E AB 2C DA	11 38 BA 33
39 D4 BA 2A	0A B5 FE 30	C3 17 62 9B	DC 7A 07 C2	9E F9 E9 59	8C 4E C5 FB
99 E6 7A B5	ED 98 76 F6	41 04 81 5B	19 40 92 40	97 90 39 AD	25 51 F7 25
04 94 7F 12	89 40 03 A8	2E 57 B6 D9	FF 1B 8C 37	AC 07 A8 5D	45 6C A6 CA
91 3C 74 C8	24 F8 F3 54	6B 63 4E 01	13 A6 8C 90	E0 DC 30 E9	B3 41 9C C5
03 64 40 EE	F4 BD 23 29	D5 8D C3 CD	06 63 DA F5	FA 16 CF 44	73 FB BE A7
83 0A C1 FA	47 C4 1E 46	49 38 D4 0D	AC 32 F8 85	B4 3F 4C 86	0E 5B 9B 22
56 8F DC 1E	90 95 3E 43	56 AA 79 E7	59 94 E3 73	A5 2C 48 83	35 EF 8E 05
D5 34 E4 2A	86 B6 E4 23	3D 6A 0F 80	AC 31 B2 5D	EE 98 73 34	C5 62 1B D5
6F D6 75 1C	48 07 DB 04	20 61 69 84	0B C9 49 0A	B2 6B 23 76	6C 3B 79 36
90 27 44 18	27 B1 59 02	14 B0 58 FE	AF 64 39 75	B2 02 7B 79	59 9B 70 FF
F3 B4 90 27	49 83 C1 B6	4A 82 3B 8B	9D 9C 5C C5	A7 CB 20 9A	BE 71 EB DC
28 F2 79 E0	0B 8D B6 63	0A A6 F1 CE	D9 4F 88 D5	B3 9E 1C 57	49 E9 94 4D
C4 6D 06 22	B7 67 E6 73	8C D8 D4 44	69 52 2F 83	EE E6 95 3E	D9 7C BA F8
6F 03 E5 31	34 C3 6D 3A	2C F3 1E 34	7C 7F 99 71	FE FD 44 E4	AE 27 81 C2
64 56 E1 98	FF 7F 16 CF	DD 4A 03 24	34 4B BF FA	8D CB 3B 46	4B A7 CB 88
AC C2 91 34	63 1C E1 41	A0 0D 85 5C	AA 53 CF 9A	15 DB 74 09	27 48 CD 0D
B5 E1 7D 9F	A2 9B 37 60	48 C4 72 3B	0E AC DA 1A	85 2C 5C A2	20 89 9E 37
CF 91 45 1A	E3 3F B3 32	AE 47 F2 AD	C4 65 E4 54	28 9B 0B BA	52 5C E7 AE
D8 36 1B 9B	D7 B9 6C 8B	61 D5 C2 35	69 97 13 38	EC 12 D7 A0	6F E6 DE 32
5E 1A AE B1	E7 64 DB A3	EA F7 61 15	FD 49 18 23	49 F2 87 BD	EB F4 62 31
C0 6A A3 D1	E9 18 52 1A	50 97 16 2B	A7 EB 00 5F	E0 6B 9C 0D	4F 84 1E 4A
1A D0 BE 00	09 31 83 BB	C7 7C 23 F7	CA 85 17 5C	3B FB 42 FB	80 A9 EF 71
A1 40 FC 12	2D 87 4E F1	9C 1A 98 17	85 75 FC 4B	6D 49 2E 9B	B3 19 9B 4F
69 A6 0A 6F	88 68 E1 DA	40 EE 96 39	8E CB 11 0E	70 0F BD 3E	68 F4 1C 33
C6 3F 4C E0	2C B1 EE 02	72 45 07 7C	87 3F 28 83	67 FB 32 85	74 B3 FF 4F
C6 A3 3A E3	3B 04 B2 1D	1F A5 BE 30	D7 1D 70 EC	23 FC 5D 8D	46 FA A3 7C
F8 3A 8D AC	40 E1 42 17	8C A2 57 A3	34 1A BF 4B	DA DF EB 8A	FA A3 71 F5
25 00 4A FA	CB 67 E8 8C	84 71 21 7E	3C A5 8F C6	01 C6 95 0B	A7 5C 15 78
19 42 E6 6E	51 0A CF 00	A1 01 4E EC	43 99 C1 F8	BD 54 A3 69	82 C4 BD 5B
3A 84 7B C4	6E AE 67 38	46 13 D8 BE	0B 9B DE 3E	CC 70 48 E6	99 D9 7A DC

Due to one byte can represent 256 different values (0x00 – 0xFF), some byte – values are available several times, latest when 257 bytes or more are given.

A good entropy is achieved, when each value has nearly the same number of occurrence within a large number of random bytes.
In the table above, 960 random bytes are listed. An optimal theoretical distribution of 960 / 256 = 3.75 is calculated. So, when each value occurs 3 - 5 times in average, a very good distribution is given.

Table 2: Example for the distribution of a few byte - values given in Table 1 above

Byte value “00” is generated	5 times.
Byte value “01” is generated	3 times.
Byte value “02” is generated	4 times.
Byte value “03” is generated	4 times.
Byte value “04” is generated	6 times.
Byte value “05” is generated	1 times.
Byte value “06” is generated	2 times.
Byte value “07” is generated	5 times.
Byte value “08” is generated	1 times.
Byte value “09” is generated	2 times.
Byte value “10” is generated	5 times.
Byte value “11” is generated	2 times.
Byte value “12” is generated	4 times.
Byte value “13” is generated	3 times.
Byte value “14” is generated	3 times.
Byte value “15” is generated	5 times.
Byte value “16” is generated	4 times.
Byte value “17” is generated	4 times.
Byte value “18” is generated	4 times.
Byte value “19” is generated	3 times.
Byte value “20” is generated	4 times.
…	
Byte value “FF” is generated	5 times.

The distribution is in the expected range. When the number of random bytes increase, the optimal theoretical distribution for each value will become closer.

6.2 100 Random Numbers, each one generated after reset release

Hackers try to get aware of the algorithm of scrambling units. Especially the first values, generated after reset release are most interesting for them, due to the CPU registers have defined values after reset. So, the whole system has a fixed and reproducible start position. If there is a predictable algorithm behind, especially the first few Random Numbers gives hackers the chance to crack it. So, the seed is most important for real randomness.

In the table below, each of the 100 Random Numbers have been generated after reset release. After reset release, the micro controller starts as usual. The timers for random number generation gets initialized, the software for Random Number Generation is started and the first Random Number (32 bit long) is generated, shown in the table below.

Table 3: 100 Random Numbers, each one generated after reset release

Random Number 1-25	Random Number 26-50	Random Number 51-75	Random Number 76-100
13 78 B6 65	CA 11 57 40	74 2D C1 76	F5 8A 26 9F
A6 CF BE 68	13 E5 74 2D	78 7B D7 C9	6B F2 F3 1C
C3 71 99 AC	34 C9 80 40	F2 AB 49 82	F6 57 F8 A8
AA 27 F5 18	35 68 D1 71	91 6E 57 D7	07 ED 66 7B
0E D3 AF 45	34 E7 F0 52	5F 14 A9 19	9F 5D B2 8F
A0 7D CE 2C	33 99 C5 6E	42 61 D3 C4	19 43 3C 81
96 D9 95 0C	4B 8C C7 E3	AD E2 29 D9	E5 1A 27 C7
FD 8E 76 80	5F F3 BA 07	E1 5D B2 8F	14 7F E0 76
01 E1 64 79	2E 56 09 85	6D B2 78 76	6E E7 BB 39
78 92 D7 CE	82 3D 9B 8C	FE 04 40 AB	42 6C DA E7
2E 50 E0 73	5B BF 20 DB	9D 39 DF D8	34 C9 B6 56
28 4D E2 76	27 B9 7B 7B	20 7D 1B 8B	9E D1 5F EC
0F 4E 98 53	19 B8 0A 25	52 F8 85 3E	37 EF C4 4F
8B 5F D3 C3	22 D2 0B 54	79 A4 F3 51	8A 23 F7 A8
FA E1 23 92	A5 3B 7E D5	18 D7 13 14	82 21 9C B7
8C 6C 33 E6	81 DC 19 2F	B2 7A C0 E6	5F 02 FE 35
86 DC 15 D5	83 C1 B6 49	78 D9 1A DA	B6 66 0F D0
03 FC F5 25	78 2B A9 05	7D 02 C0 0B	C9 B6 41 81
38 A1 85 A2	5B 91 20 35	AF 61 D3 C3	39 D5 0A 35
34 E7 84 31	34 9B E3 31	68 A6 09 5A	27 BE 72 7B
F9 F8 B3 7B	01 47 30 D9	79 81 C6 E0	2F 85 C2 67
78 2B F0 30	22 D0 6F 54	B5 05 C8 FE	CF 01 6A AF
A0 7D FB AC	81 34 4C E0	40 95 FC 15	4A 0E 15 20
64 8D D2 4A	D7 4F BE AE	96 E1 51 CC	19 B2 74 49
35 D3 08 55	9F 60 E0 D3	F0 70 8F B7	A8 62 DE CE

The table above shows an even distribution of values, a high level of entropy is given.

7 Hints for the usage of this Application Notes software package

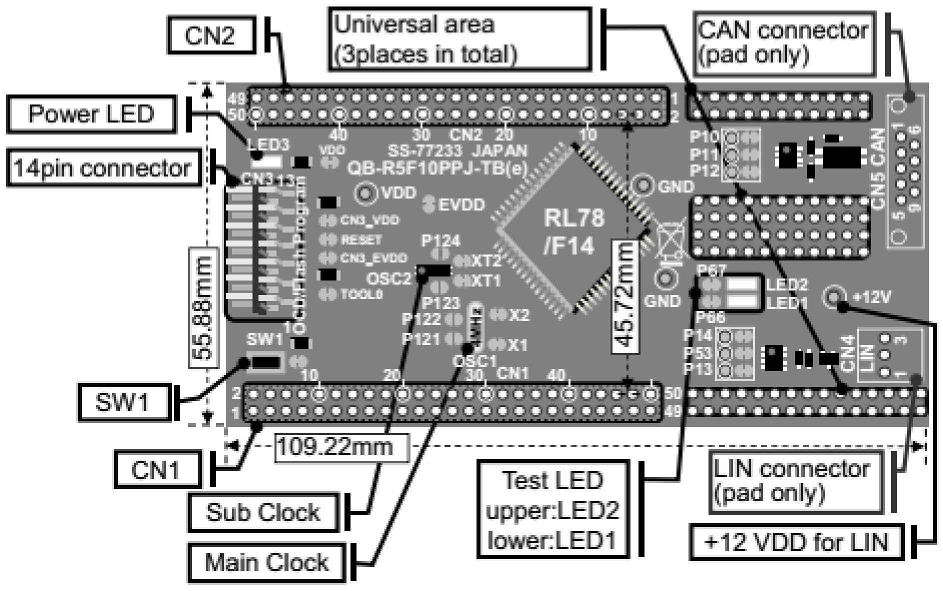
For customer´s convenience, a software package (Random-Number_AN.zip) developed, compiled and debugged using IAR Embedded Workbench V2.21.5 is available to demonstrate the functionality of this Application Note. If any kind of Random Number Generation shall be implemented in users application by software, some functions of this Application Note might be copied into user´s software flow.

Nevertheless adaption will be necessary to users application and its available resources.

7.1 Target board QB- R5F10PPJ-TB

Originally, this software is written for RL78/F14 (R5F10PPJ). The target board QB- R5F10PPJ-TB has been used for debugging. If the user will get an impression of the software and there is no hardware change at the target board, the software can run at his target board QB- R5F10PPJ-TB without any change.

Figure 10: Target board QB- R5F10PPJ-TB, dimensions and part layout



At the target board QB- R5F10PPJ-TB, two LED´s are mounted at pin P66 and P67. When these LED´s are toggling the software has generated a Random Number and the end of the software flow is reached. Toggeling P66, P67 is just for debug purpose. In a real application of random number generation it is not necessary.

If the software has to be adapted to users PCB, please check carefully, if the voltage level at the output ports P66 and P67 will not influence user´s PCB in any negative way. Otherwise, simply delete that port toggling.

7.2 Necessary controller resources used in this Application Note´s Software

Timer TAU0 channel 0, clocked as fast as possible. Two counter TCR00 readouts are used for one random byte.

Timer TRJ is clocked by the slow and unprecise internal Low-speed on-chip oscillator. Its interrupt triggers the readout of timer TAU0 channel 0 counter TCR00.

Each other resource of any other controller, which fulfills the described features in similar manner can be used to generate Random Numbers. A large tolerance of the oscillator is an advantage. Adaption of resources, timing and scrambling algorithm might be necessary.

7.3 Software for Random Number generation (4 bytes assembly)

7.3.1 Sequence to start the generation of a Random Number

```
void main(void)

{
   R_MAIN_UserInit();
   RNG_Init_and_start();          // Prepare timer TRJ and TAU0 ch.0 and start RNG generation

   while ( RNG_valid != 1)        // Marker RNG_valid = 1 shows that a new RNG is generated
   {
     asm ( "nop" );               // Wait up to a complete random number ( 32 bit ) is given
   }

// A Random Number is generated and ready for use.

    R_TAU0_Channel0_Stop();  // Stop TAU0 channel 0
    R_TMR_RJ0_Stop();        // Stop timer TRJ, disable INTTRJ0 interrupt
}
```

7.3.2 Initialisation of timer TAU0 and timer TRJ and start of Random Number generation

```
void RNG_Init_and_start (void)

{
// TAU0 ch0: "freerunning" emulation by writing 0xFFFF into its compare register

    TDR00 = 0xFFFF;               // TAU0 ch.0 clocked with 32 MHz continously running
    R_TAU0_Channel0_Start();      // Start Timer TAU0 channel 0

// In this AN, four Timer TRJ Interrupts are used to readout TCR00 and store in Random_Number [4]

    Rounds = 4;                   // Define 4 TRJ interrupts for 2 readouts of TCR00 per interrupt
    RNG_valid = 0x00;             // Mark, that still no new Random_Number is generated
    R_TMR_RJ0_Start();            // Start timer TRJ, clocked by 15 kHz +/- 15% ( fIL )
}
```

7.3.3 Setup of timer TAU0 ch0, generated by Applilet

```
void R_TAU0_Create(void)

{
   TAU0EN = 1U;          /* supplies input clock */
   TPS0 = _0000_TAU_CKM0_FCLK_0 | _0000_TAU_CKM1_FCLK_0 | _0000_TAU_CKM2_FCLK_0 |
            _0000_TAU_CKM3_FCLK_0;

/* Stop all channels */
   TT0 = _0001_TAU_CH0_STOP_TRG_ON | _0002_TAU_CH1_STOP_TRG_ON |
          _0004_TAU_CH2_STOP_TRG_ON | _0008_TAU_CH3_STOP_TRG_ON |
          _0010_TAU_CH4_STOP_TRG_ON | _0020_TAU_CH5_STOP_TRG_ON |
          _0040_TAU_CH6_STOP_TRG_ON | _0080_TAU_CH7_STOP_TRG_ON |
          _0200_TAU_CH1_H8_STOP_TRG_ON | _0800_TAU_CH3_H8_STOP_TRG_ON;

/* clear PWM output delay */
   PWMDLY1 = _0000_TAU_PWM_DELAY_CLEAR;

/* Mask channel 0 interrupt */
   TMMK00 = 1U;          /* disable INTTM00 interrupt */
   TMIF00 = 0U;          /* clear INTTM00 interrupt flag */

/* Interrupt setting for channel 1 to 7 is not shown in that list below. */
         …

/* Channel 0 used as interval timer */
   TMR00 = _0000_TAU_CLOCK_SELECT_CKM0 | _0000_TAU_CLOCK_MODE_CKS |
              _0000_TAU_COMBINATION_SLAVE |  _0000_TAU_TRIGGER_SOFTWARE |
              _0000_TAU_MODE_INTERVAL_TIMER | _0000_TAU_START_INT_UNUSED;
   TDR00 = _0C7F_TAU_TDR00_VALUE;
   TO0 &=   ~_0001_TAU_CH0_OUTPUT_VALUE_1;
   TOE0 &= ~_0001_TAU_CH0_OUTPUT_ENABLE;
}
```

7.3.4 Start and stop of timer TAU0 ch0, generated by Applilet

```
void R_TAU0_Channel0_Start(void)
{
   TS0 |= _0001_TAU_CH0_START_TRG_ON;
}


void R_TAU0_Channel0_Stop(void)
{
   TT0 |= _0001_TAU_CH0_STOP_TRG_ON;
}
```

7.3.5 Setup of timer TRJ, generated by Applilet

```
void R_TMR_RJ0_Create(void)

{
   TRJ0EN = 1U;                                  /* enable input clock supply */
   TRJCR0 &= (uint8_t)~_01_TMRJ_COUNT_START;    /* disable TMRJ0 operation */
   TRJMK0 = 1U;                                  /* disable INTTRJ0 interrupt */
   TRJIF0 = 0U;                                  /* clear INTTRJ0 interrupt flag */

/* Set INTTRJ0 low priority */
   TRJPR10 = 1U;
   TRJPR00 = 1U;
   TRJMR0 = _00_TMRJ_MODE_TIMER | _40_TMRJ_COUNT_SOURCE_FIL;
   TRJIOC0 = _00_TMRJ_TRJIOC_INITIAL_VALUE;
   TRJ0 = _0001_TMRJ_TRJ0_VALUE;
}
```

7.3.6 Start and stop of timer TRJ, generated by Applilet

```
void R_TMR_RJ0_Start(void)
{
   TRJIF0 = 0U;                            /* clear INTTRJ0 interrupt flag */
   TRJMK0 = 0U;                            /* enable INTTRJ0 interrupt */
   TRJCR0 |= _01_TMRJ_COUNT_START;         /* enable TMRJ operation */
}


void R_TMR_RJ0_Stop(void)
{
   TRJCR0 &= (uint8_t)~_01_TMRJ_COUNT_START;    /* disable TMRJ operation */
   TRJMK0 = 1U;                                  /* disable INTTRJ0 interrupt */
   TRJIF0 = 0U;                                  /* clear INTTRJ0 interrupt flag */
}
```

7.3.7 Timer TRJ interrupt: Reading TCR00 twice and scrambling for one RNG byte

```
#pragma vector = INTTRJ0_vect

__interrupt static void r_tmr_rj0_interrupt(void)

{
// Each time when timer TRJ generates an interrupt, counter TCR00 is read. The first value is used
// for a wait time where the CPU waits. After wait has elapsed, TCR00 is read again and these values are
// mixed up by inversion and XOR operations, like a scrambling unit by software.
// Finally one Byte is used as 1/4 part of a random number, stored in Random_Number [Rounds].
// In a loop of 4 TRJ interupts, 4 Bytes are collected to build a final 32 bit value as Random_Number.
// When the Random_Number is valid, it is marked by setting marker RNG_valid = 1 for further usage.

    RNG_wait = ( TCR00 & 0x00FF );     // The last 8 bits of TCR00 are used for wait time and TCR00_1st

// Wait time depends on randomness read value of TCR00(7:0)

    for ( i = RNG_wait; i > 0; i --);           // RNG wait time has to elapse, TAU0 ch0 keeps counting

// Scrambling of the two TAU0 ch0 readouts for one byte of the 32 - Bit Random_Number.

    Rounds --;                          // Preparation for next interrupt service and actual storage RAM address
    RNG_new = ( TCR00 & 0x00FF );     // TCR00 is read 2nd time within this interrupt service
    j = ( TCR00 >> 8 ) ;                       // High - part of counter register TCR00 in "j"

    RNG_new = RNG_new ^ j ;             // New Random value: TCR00_2nd: High-part  XOR  Low-part
    RNG_new = RNG_new ^ ~ RNG_wait ;         // RNG_new: XOR with inverted wait-time (TCR00_1st)
    Random_Number [Rounds] = (RNG_new);    // 1/4 part of the new Random_Number stored in RAM

     if  (Rounds == 0 )                        // Random_Number is completed after 4 TRJ service routines
    {
      RNG_valid = 0x01;                     // Marker setting: A Random_Number is prepared for further usage
    }
 }
```

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/contact/

Revision History

Rev.	Date	Description	
		Page	**Summary**
0.10	22.09.2017	All	Initial edition
0.11	27.10.2017	Ch. 5	Improved description for unpredictable values due to jitter.
		All	First released version
1.02	28.11.2018	Ch. 5.3	Improved unpredictable Random Number caused by jittered values exclusively

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 6.0