2.2. ГЕНЕРАТОРИ, БАЗИРАНИ НА ПРИМИТИВНА ФУНКЦИЯ НА ИЗПЪЛНИТЕЛНАТА СРЕДА

Решава се третата от поставените в т. 1.4 задачи: да се реализира генератор на псевдослучайни последователности чрез примитивна функция на паралелната изпълнителна среда с оглед достигането на максимална ефективност.

Формулировка на задачата:

Да се предложи реализация на генератор на псевдослучайна последователност, използващ примитивната функция *crc32()* на паралелната изпълнителна среда. Реализацията да бъде оформена в отделен проект, който включва и разработените до момента генератори.

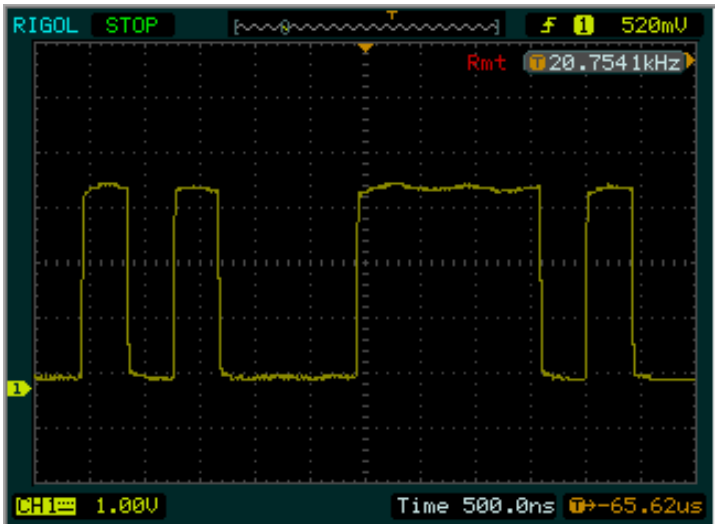
В проекта *I-T002* се използва паралелната система от т. 2.2 със *CSP* уравнение {*P* || *Q* || *L*}. Процесите *P*, *Q* и *L* имат същото предназначение.

Процесът *P* вика генераторната функция `RNG_CRC32()`, разпакетира получената *32* bit стойност и предава бит по бит по изходния си канал към *Q* крайната случайна последователност. Процесът *Q* пакетира обратно в *32* bit думи получените на входния си канал побитови последователности и ги записва в масива на извадката. Процесът *L* изпълнява помощната функция да индицира работата на *P* на светодиодната индикация.

Подобен начин на работа позволява двойна интерпретация на формираната случайна последователност – като побитова (разпакетирана) последователност и/или като последователност от *32* bit думи. При това се запазва една и съща структура на паралелната система.

Основен принцип за реализиране на дадена функция, от гледна точка на нейната ефективност, е да се използва максимално съществуващата машинна поддръжка.Това, разбира се, е валидно и в случая - колкото и да са ефективни *LFSR* генераторите, за предпочитане пред тяхната програмна реализация са наличните примитивни функции на изпълнителната среда.

В паралелната архитектура *XS1* има подобна поддръжка. Тя съдържа машинната инструкция *CRC* [37, 38, 52]. Тясната връзка



Фиг. 2.7. Част от осцилограмата на изходния сигнал на порт *oportRngBit* при *RNG* с примитивна функция на средата

между алгоритъма за изчисляване на *цикличния полиномиален код* (Cyclic Redundancy Code, *CRC*) и *LFSR* алгоритмите подсказва възможна реализация на *RNG* с инструкцията *CRC* [45].

Паралелният език *XC* дава достъп до инструкцията *CRC* чрез примитивната функция `crc32()`. Нейният прототип е

```
void crc32(unsigned &checksum, unsigned data, unsigned
poly);
```

Параметърът `checksum` задава началната стойност на контролната сума; параметърът `data` – данните, за които се изчислява контролната сума; параметърът `poly` – използвания полином. Може да се използва например полинома на *Koopman* с шестнайсетичен код *0xEB31D82E* [45].

Семантичното описание на машинната инструкция *CRC*, се дава в [52] в еквивалентен запис на *C*

```
void crc32(unsigned &checksum, unsigned data, unsigned
poly)
{
   for (int i = 0; i < 32; i++)
   {
         int xorBit = (crc & 1);
         checksum = (checksum >> 1) | ((data & 1) <<
31);
         data = data >> 1;
```

```
            if (xorBit)
                  checksum = checksum ^ poly ;
      }
   }
```

За да се генерира псевдослучайна последователност, вторият параметър `data` трябва по условие да бъде *0xFFFFFFFF*. Променливата на състоянието `uintCRC32Reg` на генераторната функция `RNG_CRC32()` има начална стойност *1*.

Самата генераторна функция `RNG_CRC32()` е капсулация на примитивната функция `crc32()`

```
UINT RNG_CRC32(UINT uintSeed, UINT uintPoly)
{
   if (uintSeed > 0)
         uintCRC32Reg = uintSeed;

   crc32(uintCRC32Reg, 0xFFFFFFFF, uintPoly);

   return uintCRC32Reg;
}
```

Този генератор се рандомизира също чрез обръщение към `Randomize()`.

На фиг. 2.7 е приведена част от осцилограмата на изходния сигнал на порт `oportRngBit` за генератора, изпълнен с разгледаната примитивна функция.

От приложените осцилограми се вижда, че периодът за изработване на *1* bit е под *500* ns, което показва ускорение около *2* пъти спрямо *LFSR* генератора с конфигурация Галоа.