2.6. ВАРИАНТ НА МОНОБИТНИЯ ТЕСТ ЗА ПРОВЕРКА НА ГЕНЕРИРАНАТА ПОСЛЕДОВАТЕЛНОСТ В РЕАЛНО ВРЕМЕ

Решава се шестата от поставените в т. 1.4 задачи: със средствата на паралелната изпълнителна среда да се реализира *монобитния* тест за проверка на генерираната последователност в реално време.

Формулировка на задачата:

Да се предложи реализация на *контролния блок* от фиг. 1.4. Реализацията трябва да отчита следните две условия:

1. Контролният блок да се основава на монобитния тест за статистическа проверка.
2. Алгоритъмът за изпълнение на теста да позволява извършването на проверката в реално време.

В литературните източници се описват множество тестове за проверка дали изработваната от даден генератор случайна последователност притежава необходимите характеристики и доколко е близка до действително-случайна последователност [3, 6, 34, 39].

Измежду най-често цитираните тестове за статистическа проверка са *монобитния* или *честотния* тест (Monobit Test, Frequency Test); *серийния* или *двубитния* тест (Serial Test, Two-bit Test); *покер* теста (Poker Test); теста за *дълги последователности от 0 или 1* (Runs Test); *автокорелационния* тест (Autocorellation Test); *спектралния* тест (Spectral Test) и много други.

Отбелязва се обаче, че не съществува отделен тест или група тестове, които могат да дадат крайна оценка за работата на *RNG*. Единствено, може да се открие определена слабост на проверявания *RNG*. Всеки подобен тест определя дали генерираната последователност притежава специфична за действително-случайните последователности характеристика и няма дефинитивен, а вероятностен характер [39].

Зададеният в условието *монобитен* тест определя дали броя

на *0*-те и *1*-те в генерираната последователност е приблизително равен, както се очаква за равномерно-разпределена случайна величина. Нека обозначим с n_0 броя на *0*-те, а с n_1 – броя на *1*-те в последователност от *N* на брой стойности. Изчислява се статистиката

$$X1 = \frac{(n_0 - n_1)^2}{N}, \qquad (2.7)$$

която е апроксимация на разпределението χ^2 със степен на свобода *1*, ако обемът *N* на извадката отговаря на условието $N \geq 10$. Обемът на извадката в нашия случай е $N = 1024 \times 32 = 32\,768$, т.е. условието за приложение на статистиката се покрива. Самата извадка се поддържа в масива `uintRandomArr[1024]`, като всеки елемент на този масив съдържа пакетирано *32* bit случайно число. За целите на монобитния тест, то се разглежда като *32* bit участък от генерираната случайна последователност от *0* и *1*.

ГЕНЕРАТОР	N	n_0	n_1	m_d	M	$n_0 - n_1$	Монобит тест		
							$Xd_1 = (n_0 - n_1)^2$	$X_1 = (n_0 - n_1)^2/N$	p
RNG_LFSR_F	32768	16323	16445	16445	0.50186	-122	14884	0.45422	0.50034
RNG_LFSR_G	32768	16442	16326	16326	0.49823	116	13456	0.41064	0.52164
RNG_ALG_M	32768	16931	15837	15837	0.48331	1094	1196836	36.52454	0.00000
RNG_CRC32	32768	16380	16388	16388	0.50012	-8	64	0.00195	0.96475
RNG_ROSC	32768	16469	16299	16299	0.49741	170	28900	0.88196	0.34767
ПЕРИОДИЧЕН	32768	16384	16384	16384	0.50000	0	0	0.00000	1.00000

Фиг. 2.19. Параметри на изчисляваната статистика

На фиг. 2.19 в табличен вид са приведени реални резултати от измерванията за шестте разработени в работата генератора. Освен n_0, n_1, *N* и X_1, са посочени и параметрите: *M* – математическото очакване, m_d – изместеното математическо очакване, X_{d1} – изместената статистика X_1 и съответната доверителна вероятност *p*, отговаряща на X_1. Вижда се, че за определянето на доверителната вероятност може да се използва и изместената статистика. Оттук се намира едно от средствата за удовлетворяване на второто условие по отношение на решението, като тежките операции с плаваща запетая се заместват с бързи целочислени операции и се избягва операцията делене. Не случайно текущата реализация на езика *XC* не поддържа аритметика с плаваща запетая [53].

От оставащите операции – броене, изваждане, умножение и сравнение, единствено операция умножение е трудоемка. Нейната сложност се оценява като $O(n^2)$, т.е. времето за умножение е от порядъка на квадрата на разрядността на двата *n*-разрядни съмножителя [3].

Архитектурата *XS1* поддържа на апаратно ниво *DSP* операцията *MAC* (Multiply And Accumulate) за целочислени операнди. Тази операция се изпълнява от специализирано устройство, включено в чипа, и трудоемкостта на целочисленото умножение се свежда до тази на простите аритметически операции – броене, събиране/изваждане [37, 52].

Операцията *MAC* е достъпна чрез едноименната функция `mac()`

```
{unsigned, unsigned} mac(unsigned a, unsigned b,
unsigned c, unsigned d);
```

която умножава двата операнда *a* и *b* от тип дума (*word*) и към получената двойна дума (*double word*) на произведението събира съставната двойна дума {*c*, *d*}. Параметрите *c* и *d* са съответно старшата и младшата дума на събираемото.

Функцията връща получената двойна дума като множествен резултат. Поддръжката на множествен резултат е особеност на *XC*, описана в документацията на езика [49, 53].

Множественият резултат, връщан от функцията `mac()`, изисква да се обяви подходящ тип, който в нашия случай е

```
typedef struct
{
   UINT hi;
   UINT lo;
} HILOW64;
```

В полето *hi* се получава старшата, а в полето *lo* – младшата дума на *64* bit произведение.

Монобитният тест е много подходящ и заради възможността да се редуцира обработката като се използва *плъзгаща се статистика*. За целта обработката се разделя на два етапа – начална (цялостна) обработка и инкрементална (частична) обработка. Тези етапи се вписват едно към едно в етапите на работа на процеса *Q*, разгледани в т. 2.1: етапа на формиране на началната ограничена последователност от *1024x32* bit, последван от етапа на формиране на неограничената последователност.

Функционалността на контролния блок е разбита в следните три функции:

```
          // обработка на цялата последователност
void Statistics(void);
          // инкрементална обработка
void StatisticsInc(UINT new);
```

```
                // проверка на доверителната вероятност
BOOL StopOnError(void);
```

Където функцията `Statistics()` формира началната статистика на база на пълния обем на извадката от случайните стойности, получени в края на първия етап; `StatisticsInc()` формира инкременталната статистика в края на всеки цикъл от втория етап; функцията `StopOnError()` взема решение за статистическата коректност на резултата.

Контролният блок използва няколко помощни типа данни, съобразени с функционалността му, променливата с обобщената статистика `rtStatistics`, контролните константи `uintChiHi` и `uintChiLo`

```
typedef struct
{
   int intX;          // индекс на първия елемент в ма-
сива
   UINT uintArr[NW]; // масив в основата на цикличния
буфер
} QUEUE;              // цикличен буфер (double ended
queue)

typedef struct
{
   QUEUE queueRandomSeq;    // буфер за неограничената
последователност

   UINT uintCount0;         // текущ брой нулеви
стойности
   UINT uintCount1;         // текущ брой единични стой-
ности
   UINT uintAvgD;           // текуща оценка на изместе-
ното мат.очакване
   HILOW64 hlChiD;          // текуща оценка на изместе-
ното Chi2 разпр.

   UINT uintErrors;         // брояч на грешките
} RTSTATISTICS;

RTSTATISTICS rtStatistics;              // текуща
статистика

const UINT uintChiHi = 217412;          // p = 0.01
const UINT uintChiLo = 4;               // p = 0.991
```

P	χ^2	χ^{d2}
0.0005	12.11567	397006
0.0010	10.82757	354798
0.0020	9.54954	312919
0.0030	8.80747	288603
0.0040	8.28382	271444
0.0050	7.87944	258193
0.0060	7.55030	247408
0.0070	7.27297	238321
0.0080	7.03347	230473
0.0090	6.82283	223570
0.010	6.63490	217412
0.020	5.41189	177337
0.030	4.70929	154314
0.040	4.21788	138212
0.050	3.84146	125877
0.060	3.53738	115913
0.070	3.28302	107578
0.080	3.06490	100431
0.090	2.87437	94187
0.100	2.70554	88655
0.110	2.55422	83697
0.120	2.41732	79211
0.130	2.29251	75121
0.140	2.17796	71367
0.150	2.07225	67904
0.160	1.97423	64691
0.170	1.88294	61700
0.180	1.79762	58905
0.190	1.71762	56283
0.200	1.64238	53817

p	χ^2	χ^{d2}
0.300	1.07420	35199
0.400	0.70833	23210
0.500	0.45494	14907
0.600	0.27500	9011
0.700	0.14847	4865
0.800	0.06418	2103
0.900	0.01579	517
0.910	0.01278	419
0.920	0.01009	331
0.930	0.00772	253
0.940	0.00567	186
0.950	0.00393	129
0.960	0.00252	82
0.970	0.00141	46
0.980	0.00063	21
0.990	0.00016	5
0.991	0.00013	4
0.992	0.00010	3
0.993	0.00008	3
0.994	0.00006	2
0.995	0.00004	1
0.996	0.00003	1
0.997	0.00001	0
0.998	0.00001	0
0.999	0.00000	0
1.000	0.00000	0

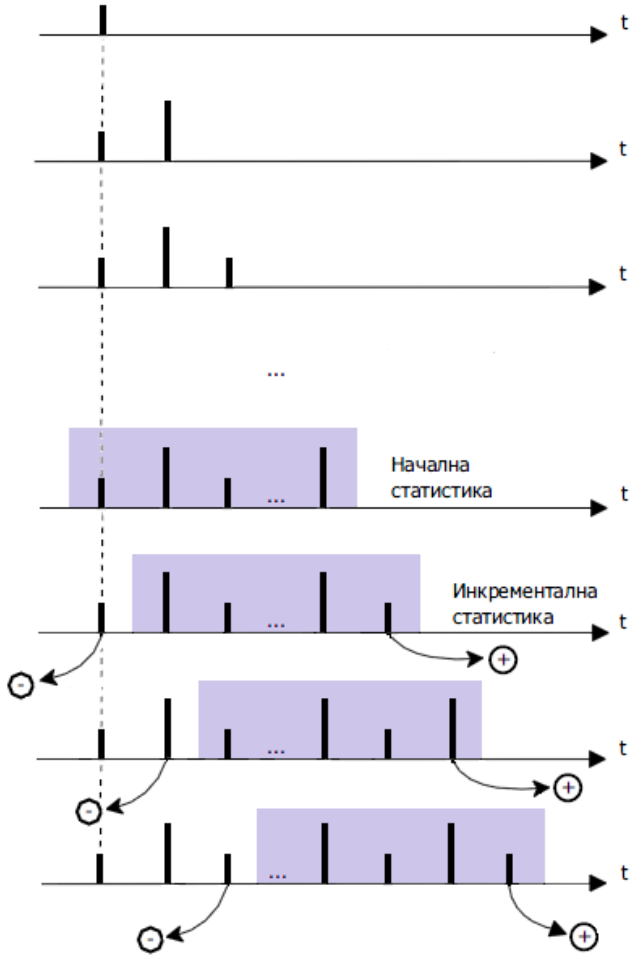
Фиг. 2.20. Таблица на разпределението χ^2 при степен на свобода 1

Променливата с обобщената статистика `rtStatistics` съдържа буфера `queueRandomSeq` с текущата извадка, броячите `uintCount0` на *0*-те и `uintCount1` на *1*-те, текущата оценка `hlChiD` на изместената статистика X_{d1} и брояча на грешките `uintErrors`.

Контролните константи `uintChiHi` и `uintChiLo` се вземат от таблицата на фиг. 2.20 и отговарят на максималната и минималната стойност на доверителната вероятност на χ^2 разпределението[1]. Те служат на функцията `StopOnError()` като критерий за грешка.

Функцията `StatisticsInc()` формира *инкременталната* (плъзгащата се) статистика в края на всеки цикъл от втория етап

1 Таблицата на χ^2 разпределението е взета от [4]. Изместените стойности χ^{d2} за работа с целочислена аритметика се получават чрез умножение по *32768* - средата на диапазона на типа *UINT*.



Фиг. 2.21. Формиране на контролната плъзгаща се статистика в реално време

на работа, който е непрекъснат. Така тя се извършва в ритъм с изработването на случайната последователност.

Инкременталната статистика е много ефективна защото не обработва всички *1024* отчета от извадката. Единствено се отстранява най-стария отчет, добавя се новия отчет и се преизчислява текущата оценка `hlChiD` на изместената статистика X_{d1}, както е илюстрирано на фиг. 2.21.

В крайна сметка се получава максимално ефективно, съобразено с предоставяните от изпълнителната среда средства решение, което има практическа приложимост.