ПРИЛОЖЕНИЕ П5: ПРОЕКТ I-T005

Проектът *I-T005* е развитие на *I-T004*, като добавя контролиращия блок от фиг. 1.4. Контролът на генерираната случайна последователност се основава на монобитния тест и се извършва в реално време. Приложим е за всякакъв тип генератори. Принципът на работа е предложен и разгледан в т. 2.6.

```
/*
 * I-T005.xc
 *
 *  Създаден на: 30.05.2012
 *   Автор: Луканчевски
 *
 *   Генератори:
 *   - LFSR конфигурация на Фибоначи;
 *   - LFSR конфигурация на Галоа;
 *   - Алгоритъм-M (Кнут);
 *   - Генератор, базиран на примитивната функция crc32 на изпълнителната среда;
 *   - Генератор, базиран на наличен в изпълнителната среда ентропиен източник;
 *   - Генератор, базиран на вградения в командата за избор недетерминизъм.
 *
 *    За извеждане на генерираната последователност се използва линията:
 *    XD0[P1A0] >>> СЛУЧАЙНА ПОСЛЕДОВАТЕЛНОСТ
 *
 *    За управление на процеса taskL се използва линията:
 *    XD1[P1B0] >>> СТРОБ СИНХРО >>> XD13[P1F0]
 *
 *    Примерът цели да се реализира монобитния тест
 *    за проверка на генерираната последователност в реално време
 *
 *  Литература:
 *  xs1_en.pdf - стр.43
 *  XS1-Library(X6020B).pdf - стр.35
 */

#include <xs1.h>
#include <stdlib.h>
#include <limits.h>

#define RNG_TYPE  5
// 1 - LFSR конфигурация на Фибоначи
// 2 - LFSR конфигурация на Галоа
// 3 - Algorithm-M
// 4 - Генератор с примитивна функция crc32 на изпълнителната среда
// 5 - Генератор с кръговите осцилатори от изпълнителната среда
// 6 - Генератор, базиран на вградения в командата за избор недетерминизъм.

#define LFSR_F_VER  2
// 1 - RNG_LFSR_F() работи с фиксиран полином
// 2 - RNG_LFSR_F() работи с полином, задаван като параметър

typedef enum {FALSE=0, TRUE} BOOL;
typedef unsigned int UINT;
```

```
typedef unsigned char BYTE;

// бутон стартиране на генерацията
in port iportButStart = XS1_PORT_1K;
// бутон спиране генерацията на неограничената последователност
in port iportButStop  = XS1_PORT_1L;

// порт за извеждане на генерираната последователност
out port oportRngBit = XS1_PORT_1A;
// изходящ стробиращ сигнал за генериране на последователността
out port oportSync   = XS1_PORT_1B;
// входящ стробиращ сигнал за генериране на последователността
in port  iportSync   = XS1_PORT_1F;

// порт на светодиодната индикация
out port oportLed    = XS1_PORT_4F;
int intLed = 0;

const int M = 32;     // дължина на регистъра
#define   NW 1024    // размер на извадката в words
const int N = NW*32;  // размер на извадката в bits

#define NW_AM  8192    // размерност на масива в Алгоритъм-М
UINT uintDelay[NW_AM];   // работен масив в Алгоритъм-М

// работен период на кръговите осцилатори
const int ROSC_PERIOD = 5000;
// период на вкл/изкл на светодиода:
// (10^8, 1/sec) * (0.125, sec)
const int LED_PERIOD  = 12500000;

UINT uintShiftReg = 1;// LFSR  - глобално състояние на генератора
UINT uintCRC32Reg = 1;// CRC32 - глобално състояние на генератора
UINT uintRandomArr[NW];// масив за фиксираната последователност

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

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

typedef struct
{
  QUEUE queueRandomSeq;  // буфер за неогр. последователност

  UINT uintCount0;  // текущ брой нулеви стойности
```

```
  UINT uintCount1;  // текущ брой единични стойности
  UINT uintAvgD;  // текуща оценка на известеното мат.очакване
  HILOW64 hlChiD;  // текуща оценка на известеното 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

#if (RNG_TYPE == 6)
  void taskP(chanend chanRight, chanend chanIn1, chanend
chanIn2);
#else
  void taskP(chanend chanRight);
#endif
void taskQ(chanend chanLeft);
void taskL(void);
void taskS(UINT uintGen, chanend chanOut);

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

BOOL StopOnError(void);
int SetLed(int intL);

void Randomize(void);
UINT RNG_LFSR_F(UINT uintSeed, UINT uintPoly);
UINT RNG_LFSR_G(UINT uintSeed, UINT uintPoly);
UINT RNG_ALG_M(UINT uintSeed, UINT uintPoly);
UINT RNG_CRC32(UINT uintSeed, UINT uintPoly);
UINT RNG_ROSC(void);

int main(void)
{
  chan chanPC, chanSP1, chanSP2;

  oportSync <: 0;

  intLed = 0;
  oportLed <: intLed;

  Randomize();

  par
  {
#if (RNG_TYPE == 6)
    taskP(chanPC, chanSP1, chanSP2);
    taskS(0, chanSP1);
    taskS(1, chanSP2);
```

```
#else
    taskP(chanPC);
#endif
    taskQ(chanPC);
    taskL();
  }

  oportLed <: 0;

  return 0;
}

#if (RNG_TYPE == 6)
  void taskP(chanend chanRight, chanend chanIn1, chanend chanIn2)
#else
  void taskP(chanend chanRight)
#endif
{
  UINT uintMsg;
  int intButStop;

  // изчакване на старт
  iportButStart when pinseq(0) :> void;

  // ЕТАП 1: Генериране на фиксирана последователност
  for(int i = 0; i < N; i++)
  {
#if (RNG_TYPE == 1)
    uintMsg = RNG_LFSR_F(0, 0x80000057);

    chanRight <: uintMsg;
    oportSync <: 1;
#elif (RNG_TYPE == 2)
    uintMsg = RNG_LFSR_G(0, 0x80000057);

    chanRight <: uintMsg;
    oportSync <: 1;
#elif (RNG_TYPE == 3)
    uintMsg = RNG_ALG_M(0, 0x80000057);

    // разпакетиране
    for(int i = 0; i < M; i++)
    {
      chanRight <: (uintMsg & 0x1);
      uintMsg >>= 1;

      oportSync <: 1;
    }
#elif (RNG_TYPE == 4)
    uintMsg = RNG_CRC32(0, 0xEB31D82E);

    // разпакетиране
```

```
    for(int i = 0; i < M; i++)
    {
      chanRight <: (uintMsg & 0x1);
      uintMsg >>= 1;

      oportSync <: 1;
    }
#elif (RNG_TYPE == 5)
    uintMsg = RNG_ROSC();

    // разпакетиране
    for(int i = 0; i < M; i++)
    {
      chanRight <: (uintMsg & 0x1);
      uintMsg >>= 1;

      oportSync <: 1;
    }
#elif (RNG_TYPE == 6)
    select
    {
      case chanIn1 :> uintMsg:
        break;
      case chanIn2 :> uintMsg:
        break;
    }

    chanRight <: uintMsg;
    oportSync <: 1;
#else
  #error INVALID RNG_TYPE
#endif
  }

  // ЕТАП 2: Генериране на неограничена последователност
  while(TRUE)
  {
    iportButStop :> intButStop;
    if(intButStop == 0)
    {
      oportSync <: 0;

      // изчакване на повторен старт
      iportButStart when pinseq(0) :> void;
    }

#if (RNG_TYPE == 1)
    uintMsg = RNG_LFSR_F(0, 0x80000057);
    chanRight <: uintMsg;
    oportSync <: 1;
#elif (RNG_TYPE == 2)
    uintMsg = RNG_LFSR_G(0, 0x80000057);
```

```
    chanRight <: uintMsg;
    oportSync <: 1;
#elif (RNG_TYPE == 3)
    uintMsg = RNG_ALG_M(0, 0x80000057);

    // разпакетиране
    for(int i = 0; i < M; i++)
    {
      chanRight <: (uintMsg & 0x1);
      uintMsg >>= 1;

      oportSync <: 1;
    }
#elif (RNG_TYPE == 4)
    uintMsg = RNG_CRC32(0, 0xEB31D82E);

    // разпакетиране
    for(int i = 0; i < M; i++)
    {
      chanRight <: (uintMsg & 0x1);
      uintMsg >>= 1;

      oportSync <: 1;
    }
#elif (RNG_TYPE == 5)
    uintMsg = RNG_ROSC();

    // разпакетиране
    for(int i = 0; i < M; i++)
    {
      chanRight <: (uintMsg & 0x1);
      uintMsg >>= 1;

      oportSync <: 1;
    }
#elif (RNG_TYPE == 6)
    select
    {
      case chanIn1 :> uintMsg:
      {
        break;
      }
      case chanIn2 :> uintMsg:
      {
        break;
      }
    }

    chanRight <: uintMsg;
    oportSync <: 1;
#else
  #error INVALID RNG_TYPE
```

```
#endif
  }
}

void taskQ(chanend chanLeft)
{
  int i, j;
  UINT uintMsg, uintNewVal;

  rtStatistics.queueRandomSeq.intX = 0;

  // ЕТАП 1
  for (i = 0; i < NW; i++)
  {
    // пакетиране
    for (j = 0; j < M; j++)
    {
      chanLeft :> uintMsg;
      oportRngBit <: uintMsg;

    // формиране на пакета в масива на фикс. последователност
      uintMsg = uintMsg << 31;
      uintRandomArr[i] = uintMsg | (uintRandomArr[i] >> 1);
    }

    // запис на пакета в цикл. буфер на неогр. последователност
    rtStatistics.queueRandomSeq.uintArr[(rtStatistics.queueRan-
domSeq.intX + i) % NW] = uintRandomArr[i];
  }

  // Обработка след формирането на фиксираната последователност
  // от NW пакета
  Statistics();
  StopOnError();

  //ЕТАП 2
  while(TRUE)
  {
    // Формиране на единичен пакет
    uintNewVal = 0;

    for (j = 0; j < M; j++)
    {
      chanLeft :> uintMsg;
      oportRngBit <: uintMsg;

      // формиране на пакета в масива
      // на фиксираната последователност
      uintNewVal = (uintMsg << 31) | (uintNewVal >> 1);
    }

    // Обработка в реално време на NW пакета
```

```
    // от неогр. последователност
    // с изместване на един отчет по оста на времето
    StatisticsInc(uintNewVal);
    StopOnError();

    // запис на пакета в цикл. буфер
    // на неограничената последователност
    // след завършване на обработката
    rtStatistics.queueRandomSeq.uintArr[
      rtStatistics.queueRandomSeq.intX] = uintNewVal;
    rtStatistics.queueRandomSeq.intX =
      (rtStatistics.queueRandomSeq.intX + 1) % NW;
  }
}

void taskL(void)
{
  timer timerT;
  int intT;

  timerT :> intT;
  intT += LED_PERIOD;

  while(TRUE)
  {
    // изчакване на вх. строб
    iportSync when pinseq(1) :> void;
    intLed = (intLed & 0xC) | SetLed(intLed);
    oportLed <: intLed;

    timerT :> intT;
    intT += LED_PERIOD;
    timerT when timerafter(intT) :> void;
  }
}

void taskS(UINT uintGen, chanend chanOut)
{
  timer timerT;
  int intT;

  while(TRUE)
  {
    chanOut <: (uintGen & 0x1);

    timerT :> intT;
    intT += 100;
    timerT when timerafter(intT) :> void;
  }
}

void Statistics(void)
```

```
{
  UINT uintRandom;

  rtStatistics.uintCount0 = 0;
  rtStatistics.uintCount1 = 0;
  rtStatistics.uintAvgD   = 0;
  rtStatistics.hlChiD.hi  = 0;
  rtStatistics.hlChiD.lo  = 0;
  rtStatistics.uintErrors = 0;

  for(int i = 0; i < NW; i++)
  {
    uintRandom = rtStatistics.queueRandomSeq.uintArr[i];

    for(int j = 0; j < M; j++)
    {
      if((uintRandom & 0x1) == 1)
        rtStatistics.uintCount1++;
      else
        rtStatistics.uintCount0++;

      uintRandom >>= 1;
    }
  }

  rtStatistics.uintAvgD = rtStatistics.uintCount1;

  {void, rtStatistics.hlChiD.lo} =
    mac((rtStatistics.uintCount0 - rtStatistics.uintCount1),
      (rtStatistics.uintCount0 - rtStatistics.uintCount1), 0, 0);
}

void StatisticsInc(UINT uintNewVal)
{
  UINT uintOldVal;

  // отстраняване на началната стойност от статистиката
  uintOldVal = rtStatistics.queueRandomSeq.uintArr[
                 rtStatistics.queueRandomSeq.intX];
  for(int j = 0; j < M; j++)
  {
    if((uintOldVal & 0x1) == 1)
      rtStatistics.uintCount1--;
    else
      rtStatistics.uintCount0--;

    uintOldVal >>= 1;
  }

  // добавяне на новата стойност към статистиката
  for(int j = 0; j < M; j++)
  {
```

```
    if((uintNewVal & 0x1) == 1)
      rtStatistics.uintCount1++;
    else
      rtStatistics.uintCount0++;

    uintNewVal >>= 1;
  }

  rtStatistics.uintAvgD = rtStatistics.uintCount1;

  {void, rtStatistics.hlChiD.lo} =
    mac((rtStatistics.uintCount0 - rtStatistics.uintCount1),
      (rtStatistics.uintCount0 - rtStatistics.uintCount1), 0, 0);
}

BOOL StopOnError(void)
{
  BOOL boolStop = FALSE;

  if(rtStatistics.hlChiD.lo > uintChiHi ||
     rtStatistics.hlChiD.lo < uintChiLo)
    boolStop = TRUE;

  if(boolStop)
  {
    if(rtStatistics.uintErrors < UINT_MAX)
      rtStatistics.uintErrors++;
  }

  return boolStop;
}

int SetLed(int intL)
{
  switch(intL & 0x3)
  {
  case 0x0:
    return 0x1;
  case 0x1:
    return 0x3;
  case 0x3:
    return 0x2;
  case 0x2:
    return 0x0;
  default:
    return 0x0;
  }
}

void Randomize(void)
{
  timer timerSeed;
```

```
  UINT uintSeed;
  timerSeed :> uintSeed;

  // инициализация на LFSR
  RNG_LFSR_G(uintSeed, 0x80000057);

  // инициализация на Алгоритъм-М
  srand(uintSeed);
  for(int i = 0; i < NW_AM; i++)
  {
    uintDelay[i] = rand();
  }

  RNG_CRC32(uintSeed, 0xEB31D82E);
}

UINT RNG_LFSR_F(UINT uintSeed, UINT uintPoly)
{
  UINT uintMSB;
  UINT uintShiftRegCopy = uintShiftReg;

  if (uintSeed > 0)
    uintShiftReg = uintSeed;

#if (LFSR_F_VER == 1)
  uintMSB = (uintShiftReg >> 31) ^ (uintShiftReg >> 6) ^
            (uintShiftReg >> 4) ^ (uintShiftReg >> 2) ^
            (uintShiftReg >> 1) ^ uintShiftReg;
#elif (LFSR_F_VER == 2)
  uintMSB = 0;

  for(int i = 0; i < (M - 1); i++)
  {
    if((uintPoly & 0x00000001) == 1)
      uintMSB ^= uintShiftRegCopy;

    par
    {
      uintShiftRegCopy >>= 1;
      uintPoly >>= 1;
    }
  }
#else
  #error INVALID LFSR_F_VER
#endif

  uintMSB = (uintMSB & 0x00000001) << 31;
  uintShiftReg = uintMSB | (uintShiftReg >> 1);

  return uintShiftReg & 0x00000001;
}
```

```
UINT RNG_LFSR_G(UINT uintSeed, UINT uintPoly)
{
  UINT uintResult;

  if (uintSeed > 0)
    uintShiftReg = uintSeed;

  if(uintShiftReg & 0x00000001)
  {
    uintShiftReg = 0x80000000 | ((uintShiftReg ^ uintPoly) >> 1);
    uintResult = 1;
  }
  else
  {
    uintShiftReg >>= 1;
    uintResult = 0;
  }

  return uintResult;
}

UINT RNG_ALG_M(UINT uintSeed, UINT uintPoly)
{
  UINT uintInx, uintRandomBit, uintResult;

  // пакетиране
  uintInx = 0;
  for(int i = 0; i < M; i++)
  {
    uintRandomBit = RNG_LFSR_F(0, 0x80000057);
    uintRandomBit = uintRandomBit << 31;
    uintInx = uintRandomBit | (uintInx >> 1);
  }

  uintInx = uintInx % NW_AM;
  uintResult = uintDelay[uintInx];
  uintDelay[uintInx] = rand();

  return uintResult;
}

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

  crc32(uintCRC32Reg, 0xFFFFFFFF, uintPoly);

  return uintCRC32Reg;
}

UINT RNG_ROSC(void)
```

```
{
  timer timerT;
  UINT uintT, uintResult;
  UINT r0, r1, r2, r3;
  UINT r0a, r1a, r2a, r3a;

  uintResult = 0;

  for(int i = 0; i < 8; i++)
  {
    r0a = getps(XS1_L_PS_RING_OSC_DATA0);  // Register 0x070B
    r1a = getps(XS1_L_PS_RING_OSC_DATA1);  // Register 0x080B
    r2a = getps(XS1_L_PS_RING_OSC_DATA2);  // Register 0x090B
    r3a = getps(XS1_L_PS_RING_OSC_DATA3);  // Register 0x0A0B

    // Enable Ring Oscilators
    setps(XS1_L_PS_RING_OSC_CTRL, 0xF);   // Register 0x060B

    timerT :> uintT;
    timerT when timerafter(uintT + ROSC_PERIOD) :> void;

    // Disable Ring Oscilators
    setps(XS1_L_PS_RING_OSC_CTRL, 0x0);   // Register 0x060B

    r0 = getps(XS1_L_PS_RING_OSC_DATA0);  // Register 0x070B
    r1 = getps(XS1_L_PS_RING_OSC_DATA1);  // Register 0x080B
    r2 = getps(XS1_L_PS_RING_OSC_DATA2);  // Register 0x090B
    r3 = getps(XS1_L_PS_RING_OSC_DATA3);  // Register 0x0A0B

    uintResult >>= 4;
    uintResult |= (((r0 - r0a) & 0x1) << 3 |
                   ((r1 - r1a) & 0x1) << 2 |
                   ((r2 - r2a) & 0x1) << 1 |
                   ((r3 - r3a) & 0x1)) << 28;
  }

  return uintResult;
}
```