ПРИЛОЖЕНИЕ П2: ПРОЕКТ QC-T001

Проектът *QC-T001* съдържа кода на предложената реализация на паралелния структурен модел на *поляризацията* на фотони. Моделираната ситуация е един от редките случаи, когато е приложимо използването на “локални скрити променливи“. Основните структурни елементи на модела са: източник на естествена светлина, три линейни поляризатора, детектор на фотони, квантови канали, класически канали. Светлинният източник излъчва кохерентен светлинен поток с равномерна поляризация във всички посоки, подобно на естествената светлина. Поляризаторите могат да се настройват.

```
/*
 * qc-T001.xc
 *
 *  Създаден на: 07.02.2013
 *       Автори: Милен Луканчевски, Бисер Николов
 *
 *  ПОЛЯРИЗАЦИЯ
 *
 *  Цел: Първоначално представяне на модел на квантови явления в
паралелна среда.
 *  Осн. идея: Заедно с представянето на предложения модел да се
илюстрира
 *  използването на «локални скрити променливи» (local hidden
variables) и
 *  на класически, предварително заложени, вероятности.
 *
 *  Моделирано явление: Поляризация на фотони.
 *
 *  Експериментална постановка (E1 - ПОЛЯРИЗАЦИЯ.docx):
 *  - източник на естествена светлина;
 *  - три линейни поляризатора;
 *  - детектор на фотони.
 *
 *  Източникът излъчва кохерентен светлинен поток с равномерна
поляризация
 *  във всички посоки (подобно на естествената светлина).
 *
 *  Поляризаторите могат да се настройват, като им се задава:
 *  - нулева поляризация (поляризаторът е абсолютно прозрачен);
 *  - хоризонтална поляризация;
 *  - вертикална поляризация;
 *  - ъглова (+) поляризация;
 *  - ъглова (-) поляризация.
 *
 *  Основни елементи на модела:
 *  - активни (задача/процес);
 *  - пасивни (канали, съобщения).
 *
 *  Източникът на светлина, поляризаторите и детекторът се
представят като процеси.
 *  Фотоните се предават като съобщения между тези процеси.
 *  Поляризацията на фотоните се представя чрез променливата на
състоянието им
 *  (локална скрита променлива, local hidden variable).
 *
 *  Източникът на светлина се управлява от RRNG/TRNG.
 *  Генераторът е базиран на един от наличните в паралелната
изпълнителна среда
 *  ентропийни източника - четири кръгови осцилатора (ROSC).
 *  При всяко обръщение към RRNG/TRNG генератора той изработва и
връща 32 bit
 *  случайна последователност. Тази случайна последователност се
```

```
изработва
 *  в 8 вътрешни обръщения към кръговите генератори с изчакване
от 50 usec
 *  за натрупване на необходимата ентропия. При всяко вътрешно
обръщение
 *  се изработва 4 bit случайна последователност.
 *  Това заема общо 400 usec (8 цикъла x 50 usec/цикъл).
 *
 *  Литература за използвания метод:
 *  [А.23, А.24]
 *  [Б.6, Б.10, Б.16]
 *
 */

//#include <xs1.h>
#include <platform.h>

//#include <stdlib.h>
//#include <string.h>
#include <limits.h>

// ТИП ПОЛЯРИЗАЦИЯ НА ИЗТОЧНИКА
#define LE_POLAR  0
// 0 - POLAR_NATURAL
// 1 - POLAR_HORIZONTAL
// 2 - POLAR_VERTICAL
// 3 - POLAR_ANGULAR_PLUS
// 4 - POLAR_ANGULAR_MINUS

const int N = 2048*4;
#define RNG_TYPE  1
// 1 - Генератор тип RNG  с примитивна функция crc32 на
изпълнителната среда;
// 2 - Генератор тип TRNG с кръговите осцилатори от
изпълнителната среда
//     поддържа се при XS1-L, но не и при XS1-G4.

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

#define STOP return

typedef enum {NOPOLAR=-1, HORIZONTAL, VERTICAL, ANGULAR_PLUS,
ANGULAR_MINUS} POLAR;
typedef enum {POS_A, POS_B, POS_C} POSITION;
typedef enum {PASS_0, PASS_1, PASS_2, PASS_3} PASS;

#define CTRL_STOP    UINT_MAX    /* Команда за край     */
#define CTRL_PASS_0    PASS_0    /* Начален опит: x x x  */
#define CTRL_PASS_1    PASS_1    /* Първи   опит   A x x  */
#define CTRL_PASS_2    PASS_2    /* Втори   опит   A x C  */
```

```
#define CTRL_PASS_3    PASS_3    /* Трети   опит   A B C  */
#define CTRL_PASS_NUM   4

typedef struct
{
  UINT id;      // identifier
  POLAR pol;    // локална скрита променлива на състоянието
  BOOL rndH;    // вероятност при поляризация от тип HORIZONTAL
  BOOL rndV;    // вероятност при поляризация от тип VERTICAL
  BOOL rndA_P; // вероятност при поляризация от тип ANGULAR_PLUS
  BOOL rndA_M; // вероятност при поляризация от тип ANGULAR_MINUS
  BOOL rndX1;   // резервна вероятност
  BOOL rndX2;   // резервна вероятност
} PHOTON;

int intCounterHORIZONTAL,
    intCounterVERTICAL,
    intCounterANGULAR_PLUS,
    intCounterANGULAR_MINUS;

void taskLightEmitter(chanend chanRightCtrl, chanend chanRight);
void taskPolaroid(chanend chanLeftCtrl, chanend chanRightCtrl,
              chanend chanLeft, chanend chanRight, POSITION pos);
void taskLightDetector(chanend chanLeftCtrl, chanend chanLeft);

POLAR GetPolarByPosition(POSITION pos, PASS pass);
PHOTON CopyPhoton(POLAR pol, PHOTON src);

void Randomize(void);
UINT RNG_CRC32(UINT uintSeed, UINT uintPoly);
UINT RNG_ROSC(void);

int main(void)
{
  chan chanE2A,
       chanA2B,
       chanB2C,
       chanC2D;

  chan chanE2ACtrl,
       chanA2BCtrl,
       chanB2CCtrl,
       chanC2DCtrl;

  par
  {
    on stdcore[0]: taskLightEmitter(chanE2ACtrl, chanE2A);
    on stdcore[1]: taskPolaroid(chanE2ACtrl, chanA2BCtrl,
chanE2A, chanA2B, POS_A);
    on stdcore[2]: taskPolaroid(chanA2BCtrl, chanB2CCtrl,
chanA2B, chanB2C, POS_B);
    on stdcore[3]: taskPolaroid(chanB2CCtrl, chanC2DCtrl,
```

```
chanB2C, chanC2D, POS_C);
    on stdcore[0]: taskLightDetector(chanC2DCtrl, chanC2D);
  }

  return 0;
}

void taskLightEmitter(chanend chanRightCtrl, chanend chanRight)
{
  UINT uintID;
  UINT uintRandom;
  int intCTRL[CTRL_PASS_NUM] =
  {
    CTRL_PASS_0, CTRL_PASS_1, CTRL_PASS_2, CTRL_PASS_3
//    CTRL_PASS_3, CTRL_PASS_3, CTRL_PASS_3, CTRL_PASS_3
  };

  uintID = 0;

  intCounterHORIZONTAL    = 0;
  intCounterVERTICAL      = 0;
  intCounterANGULAR_PLUS  = 0;
  intCounterANGULAR_MINUS = 0;

  for(int i = 0; i < CTRL_PASS_NUM; i++)
  {
    chanRightCtrl <: intCTRL[i];

    for(int k = 0; k < N/4; k++)
    {
      #if (RNG_TYPE == 1)
        Randomize();
        uintRandom = RNG_CRC32(0, 0xEB31D82E);
      #elif (RNG_TYPE == 2)
        uintRandom = RNG_ROSC();
      #else
        #error INVALID RNG_TYPE
      #endif

      for(int i = 0; i < 4; i++)
      {
        PHOTON phRight;

        uintID++;

        phRight.id = uintID;

      #if (LE_POLAR == 0)
        phRight.pol  = uintRandom & 0x03;
      #elif (LE_POLAR == 1)
        phRight.pol  = HORIZONTAL;
      #elif (LE_POLAR == 2)
```

```
        phRight.pol  = VERTICAL;
      #elif (LE_POLAR == 3)
        phRight.pol  = ANGULAR_PLUS;
      #elif (LE_POLAR == 4)
        phRight.pol  = 5;
      #else
        #error INVALID LE_POLAR
      #endif

        phRight.rndH   = (uintRandom & 0x04) >> 2;
        phRight.rndV   = (uintRandom & 0x08) >> 3;
        phRight.rndA_P = (uintRandom & 0x10) >> 4;
        phRight.rndA_M = (uintRandom & 0x20) >> 5;
        phRight.rndX1  = (uintRandom & 0x40) >> 6;
        phRight.rndX2  = (uintRandom & 0x80) >> 7;

        switch(phRight.pol)
        {
          case HORIZONTAL:
            intCounterHORIZONTAL++;
            break;
          case VERTICAL:
            intCounterVERTICAL++;
            break;
          case ANGULAR_PLUS:
            intCounterANGULAR_PLUS++;
            break;
          case ANGULAR_MINUS:
            intCounterANGULAR_MINUS++;
            break;
        }

        chanRight <: phRight;

        uintRandom >>= 8;
      }
    }
  }

  chanRightCtrl <: CTRL_STOP;
  STOP;
}

void taskPolaroid(chanend chanLeftCtrl, chanend chanRightCtrl,
              chanend chanLeft, chanend chanRight, POSITION pos)
{
  UINT uintCtrl;
  PHOTON phLeft, phRight;
  POLAR pol;

  while(TRUE)
  {
```

```
select
{
  case chanLeftCtrl :> uintCtrl:
  {
    switch(uintCtrl)
    {
      case CTRL_STOP:
      {
        chanRightCtrl <: uintCtrl;
        STOP;
        break;
      }
      case CTRL_PASS_0:
      case CTRL_PASS_1:
      case CTRL_PASS_2:
      case CTRL_PASS_3:
      {
        pol = GetPolarByPosition(pos, uintCtrl);
        // Т.пр. за проверка настройката на поляризацията
        chanRightCtrl <: uintCtrl;
        break;
      }
    }
    break;
  }
  case chanLeft :> phLeft:
  {
    switch(pol)
    {
      case NOPOLAR:
      {
        phRight = CopyPhoton(phLeft.pol, phLeft);
        chanRight <: phRight;
        break;
      }
      case HORIZONTAL:
      {
        if(phLeft.pol == HORIZONTAL)
        {
          phRight = CopyPhoton(phLeft.pol, phLeft);
          chanRight <: phRight;
        }
        else if(phLeft.pol == ANGULAR_PLUS ||
                phLeft.pol == ANGULAR_MINUS)
        {
          if(phLeft.rndH == TRUE)
          {
            phRight = CopyPhoton(HORIZONTAL, phLeft);
            chanRight <: phRight;
          }
        }
        break;
```

```
}
case VERTICAL:
{
  if(phLeft.pol == VERTICAL)
  {
    phRight = CopyPhoton(phLeft.pol, phLeft);
    chanRight <: phRight;
  }
  else if(phLeft.pol == ANGULAR_PLUS ||
          phLeft.pol == ANGULAR_MINUS)
  {
    if(phLeft.rndV == TRUE)
    {
      phRight = CopyPhoton(VERTICAL, phLeft);
      chanRight <: phRight;
    }
  }
  break;
}
case ANGULAR_PLUS:
{
  if(phLeft.pol == ANGULAR_PLUS)
  {
    phRight = CopyPhoton(phLeft.pol, phLeft);
    chanRight <: phRight;
  }
  else if(phLeft.pol == HORIZONTAL ||
          phLeft.pol == VERTICAL)
  {
    if(phLeft.rndA_P == TRUE)
    {
      phRight = CopyPhoton(ANGULAR_PLUS, phLeft);
      chanRight <: phRight;
    }
  }
  break;
}
case ANGULAR_MINUS:
{
  if(phLeft.pol == ANGULAR_MINUS)
  {
    phRight = CopyPhoton(phLeft.pol, phLeft);
    chanRight <: phRight;
  }
  else if(phLeft.pol == HORIZONTAL ||
          phLeft.pol == VERTICAL)
  {
    if(phLeft.rndA_M == TRUE)
    {
      phRight = CopyPhoton(ANGULAR_MINUS, phLeft);
      chanRight <: phRight;
    }
```

```
            }
            break;
          }
        }

        break;
      }
    }
  }
}

void taskLightDetector(chanend chanLeftCtrl, chanend chanLeft)
{
  UINT uintCtrl;
  int intCounter[CTRL_PASS_NUM], intInx;
  PHOTON phLeft;

  intInx = 0;
  for(int i = 0; i < CTRL_PASS_NUM; i++)
    intCounter[i] = 0;

  while(TRUE)
  {
    select
    {
      case chanLeftCtrl :> uintCtrl:
      {
        switch(uintCtrl)
        {
          case CTRL_STOP:
          {
            // Т.пр. за проверка на резултата
            STOP;
            break;
          }
          case CTRL_PASS_0:
          case CTRL_PASS_1:
          case CTRL_PASS_2:
          case CTRL_PASS_3:
          {
            intInx = uintCtrl;
            break;
          }
        }
        break;
      }
      case chanLeft :> phLeft:
      {
        intCounter[intInx]++;
        break;
      }
    }
```

```
  }
}

POLAR GetPolarByPosition(POSITION pos, PASS pass)
{
  POLAR polarResult = NOPOLAR;

  switch(pass)
  {
    case PASS_0:
      polarResult = NOPOLAR;
      break;
    case PASS_1:
      if(pos == POS_A)
        polarResult = HORIZONTAL;
      else
        polarResult = NOPOLAR;
      break;
    case PASS_2:
      if(pos == POS_A)
        polarResult = HORIZONTAL;
      else if(pos == POS_B)
        polarResult = NOPOLAR;
      else if(pos == POS_C)
        polarResult = VERTICAL;
      break;
    case PASS_3:
      if(pos == POS_A)
        polarResult = HORIZONTAL;
      else if(pos == POS_B)
        polarResult = ANGULAR_PLUS;
      else if(pos == POS_C)
        polarResult = VERTICAL;
      break;
  }

  return polarResult;
}

PHOTON CopyPhoton(POLAR pol, PHOTON src)
{
  PHOTON dst;

  dst.id   = src.id;
  dst.pol  = pol;
  dst.rndH = src.rndH;
  dst.rndV = src.rndV;
  dst.rndA_P = src.rndA_P;
  dst.rndA_M = src.rndA_M;
  dst.rndX1 = src.rndX1;
  dst.rndX2 = src.rndX2;
```

```
  return dst;
}

/*************************************************************
 * ГЕНЕРАЦИЯ НА СЛУЧАЙНИ ПОСЛЕДОВАТЕЛНОСТИ
 *************************************************************/

UINT uintCRC32Reg = 1;  // CRC32 - глобално състояние на RNG
генератора

void Randomize(void)
{
  timer timerSeed;
  UINT uintSeed;
  timerSeed :> uintSeed;

  RNG_CRC32(uintSeed, 0xEB31D82E);
}

/*
 * ГЕНЕРАТОР НА ПСЕВДОСЛУЧАЙНА ПОСЛЕДОВАТЕЛНОСТ - RNG
 * (чрез примитивна функция на изпълнителната среда)
 */
UINT RNG_CRC32(UINT uintSeed, UINT uintPoly)
{
  if (uintSeed > 0)
    uintCRC32Reg = uintSeed;

  crc32(uintCRC32Reg, 0xFFFFFFFF, uintPoly);

  return uintCRC32Reg;
}

/*
 * ГЕНЕРАТОР НА ДЕЙСТВИТЕЛНА СЛУЧАЙНА ПОСЛЕДОВАТЕЛНОСТ RRNG/TRNG
 * (чрез наличен в изпълнителната среда източник на ентропия)
 */

// работен период на кръговите генератори
// 50 usec = 5000 x 10E-9 = 50E-6 sec
const int ROSC_PERIOD = 5000;

UINT RNG_ROSC(void)
{
  timer timerT;     // таймер за формиране на работния период
  UINT uintT;
  UINT uintResult;  // буфер на 32 bit случайна последователност
  UINT r0, r1, r2, r3;     // начална стойност на ROSC
  UINT r0a, r1a, r2a, r3a; // крайна стойност на ROSC

  uintResult = 0;
```

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

  // Цикли на обръщение към кръговите генератори ROSC
  for(int i = 0; i < 8; i++)
  {
    // Read Ring Oscilators
    r0a = getps(0x070B);
    r1a = getps(0x080B);
    r2a = getps(0x090B);
    r3a = getps(0x0A0B);

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

    // Изчакване на работния период
    timerT :> uintT;
    timerT when timerafter(uintT + ROSC_PERIOD) :> void;

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

    // Read Ring Oscilators
    r0 = getps(0x070B);
    r1 = getps(0x080B);
    r2 = getps(0x090B);
    r3 = getps(0x0A0B);

    // освобождаване на старшата тетрада на буфера
    uintResult >>= 4;

    uintResult |= (
      ((r0 - r0a) & 0x1) << 3 |  // старши бит на тетрадата
      ((r1 - r1a) & 0x1) << 2 |
      ((r2 - r2a) & 0x1) << 1 |
      ((r3 - r3a) & 0x1)         // младши бит на тетрадата
                  ) << 28;  // изместване на формираната 4 bit
                            // случайна последователност
                           // в старшата тетрада на 32 bit буфер
  }

  // Връщане на формираната 32 bit действителна сл. посл.
  return uintResult;
}
/*****************************************************************/
```