Assignment 2 (Week 6-7: "LED Particle Simulation" on XC-1A board)

- This assignment is the first assessed piece of coursework in the unit.
- It is to be completed in pairs. (report any change to the team structure to the course director BEFORE starting your assignment)
- It is worth **10% of the unit mark** (i.e. 20% of the coursework component).
- Submission: Every student is required to upload their full piece of work (incl all XC source files) as a single ***ZIP file to SAFE before 23:59:59, Wed 28th Nov 2012***. Make sure you submit it early enough (not last minute!) to avoid upload problems. (Each member of a team has to upload an identical copy of the teams work.)
- Assessment: You will present your submitted program running (using the XC-1A boards) in the labs on ***Thu 29th Nov 2012 and 10am on Tue 4th Dec 2012***. You will need to attend these lab session to get a mark. At these labs, we will ask you questions about your work and you will be able to showcase the merits of it.
- Do not attempt to plagiarise or copy code between teams etc. It is not worth it, be proud of your own work! We will ask you questions about your work in the labs - so you must understand the code your team developed in any case.

Your Task:

"LED Particle Simulation" on XMOS XC-1A board

Introduction. Concurrent simulations on parallel architectures are frequently used in science to model complex real-world phenomena, such as chemical, physical, meteorological or biological systems. In most cases, these simulation systems approximate the phenomena by a multitude of simple, similarly structured constituent threads which, when interacting, show the emerging behaviour of interest (e.g. protein structures, particle distributions, weather patterns etc).

Task Details. You are given an XC code skeleton that provides you with the most basic structure and helper routines to implement a tiny simulation on the XC-1A board. Your task is to extend the given skeleton code to implement the following simulation modelling three (or potentially more) interacting particles:

LED Particles. An "LED Particle" is represented by a position on the clock wheel of the XC-1A board, a constant velocity property and a direction of current movement (i.e. clockwise=1 or anti-clockwise=-1). There are three such LED Particles (starting at locations 0, 3 and 6 in directions -1, 1, -1), each of which is visualised by one active red LED on the 12-position LED clock. No two particles can ever occupy the same position on the clock.

Simulation Control. The simulation can be started by pressing button A, paused and restarted by pressing button B and terminated by pressing button C.

Simulation. During the simulation, the LED Particles move at their given velocity (to be implemented as a delay of a particular thread before the next move) from their current location to the neighbouring location in the current direction. In case an LED Particle encounters a neighbouring LED Particle in the location it seeks to enter, it collides and BOTH particles change their direction to move away from each other. Each LED Particle is modelled with channels (from neighbours[]) that link to the two neighbouring particles on the LED clock. The channels in the array show[] provide a link from ants to the visualiser, where visible positions can be updated by a particle sending its current position (0..11). The thread visualiser can playSound when a token >11 is received.

About the Skeleton Code Your task is to implement missing code, mainly in the threads main, particle and buttonListener to implement the simulation as described.
- only use the channels predefined, do not add extra channels to the concurrent system;
- use thread replication to create multiple instances of the particle thread;
- for system start, pause and shut down, send an appropriate token from the buttonListener to the visualiser, which in turn uses its channels to send tokens to all threads that need to act;

The showLED and playSound functions are implemented for you. The thread visualiser waits to receive the current position of particles and instructs the board to switch on/off LEDs accordingly.

Basic Extras and Excellence.

If you aim for a mark in the merit range (60-64) and above, implement a graceful shutdown of all threads (do not break out of loops here but formulate a termination condition).

If you aim for a mark in the upper merit range (65-69) and above, implement a system where a particle can provide immediate feedback to its neighbours at any time when not moving and allow the user to input particle start positions before the simulation.

For a first class mark (70-74), also argue during presentation why your system cannot produce deadlock by circular waits of particle feedback or otherwise, and implement a number of complex extra features of your choice (more than three particles, more complex physics with transferable particle impulses, differing velocities etc).

For a top first class mark (75+), also implement further complex features and impress the marker with your knowledge and implementation, for instance, a solution that has minimal synchronization overheads, is highly scalable and allows the user to specify specific physical parameters etc...

```
/////////////////////////////////////////////////////////////////////////////////////////
//
// COMS20600 - WEEKS 6 and 7
// ASSIGNMENT 2
// CODE SKELETON
// TITLE: "LED Particle Simulation"
//
/////////////////////////////////////////////////////////////////////////////////////////


#include <stdio.h>
#include <platform.h>

out port cled[4] = {PORT_CLOCKLED_0,PORT_CLOCKLED_1,PORT_CLOCKLED_2,PORT_CLOCKLED_3};
out port cledG = PORT_CLOCKLED_SELG;
out port cledR = PORT_CLOCKLED_SELR;
in port  buttons = PORT_BUTTON;
out port speaker = PORT_SPEAKER;

#define  noParticles    3   //overall number of particles threads in the system



/////////////////////////////////////////////////////////////////////////////////////////
//
//  Helper Functions provided for you
//
/////////////////////////////////////////////////////////////////////////////////////////


//DISPLAYS an LED pattern in one quadrant of the clock LEDs
void showLED(out port p, chanend fromVisualiser) {
  unsigned int lightUpPattern;
  unsigned int running = 1;
  while (running) {
    select {
      case fromVisualiser :> lightUpPattern: //read LED pattern from visualiser process
        p <: lightUpPattern;                 //send pattern to LEDs
      break;
      default:
      break;
    }
  }
}


//PLAYS a short sound (pls use with caution and consideration to other students in the labs!)
void playSound(unsigned int wavelength, int duration, out port speaker) {
  timer  tmr;
  int t, isOn = 1;
  tmr :> t;
  for (int i=0; i<duration; i++) {
    isOn = !isOn;
    t += wavelength;
    tmr when timerafter(t) :> void;
    speaker <: isOn;
  }
}


//WAIT function
void waitMoment(uint myTime) {
  timer tmr;
  unsigned int waitTime;
  tmr :> waitTime;
  waitTime += myTime;
  tmr when timerafter(waitTime) :> void;
}
```

```
/////////////////////////////////////////////////////////////////////////////////////////
//
//  RELEVANT PART OF CODE TO EXPAND
//
/////////////////////////////////////////////////////////////////////////////////////////

//PROCESS TO COORDINATE DISPLAY of LED Particles
void visualiser(chanend toButtons, chanend show[], chanend toQuadrant[], out port speaker) {
  unsigned int display[noParticles];  //array of ant positions to be displayed, all values 0..11
  unsigned int running = 1;           //helper variable to determine system shutdown
  int j;                              //helper variable
  cledR <: 1;
  while (running) {
        for (int k=0;k<noParticles;k++) {
          select {
            case show[k] :> j:
              if (j<12) display[k] = j; else
              playSound(20000,20,speaker);
            break;
            ///////////////////////////////////////////////////////////////////////
            //
            //   ADD YOUR CODE HERE TO ACT ON BUTTON INPUT
            //
            ///////////////////////////////////////////////////////////////////////
            default:
            break;
          }
          //visualise particles
          for (int i=0;i<4;i++) {
            j = 0;
            for (int k=0;k<noParticles;k++)
              j += (16<<(display[k]%3))*(display[k]/3==i);
            toQuadrant[i] <: j;
          }
    }
  }
}

//READ BUTTONS and send commands to Visualiser
void buttonListener(in port buttons, chanend toVisualiser) {
  int buttonInput;            //button pattern currently pressed
  unsigned int running = 1;   //helper variable to determine system shutdown
  while (running) {
            buttons when pinsneq(15) :> buttonInput;
            ///////////////////////////////////////////////////////////////////////
            //
            //   ADD YOUR CODE HERE TO ACT ON BUTTON INPUT
            //
            ///////////////////////////////////////////////////////////////////////
  }
}


//PARTICLE...thread to represent a particle - to be replicated noParticle-times
void particle(chanend left, chanend right, chanend toVisualiser, int startPosition, int startDirection) {
  unsigned int moveCounter = 0;          //overall no of moves performed by particle so far
  unsigned int position = startPosition; //the current particle position
  unsigned int attemptedPosition;        //the next attempted position after considering move direction
  int currentDirection = startDirection; //the current direction the particle is moving
  int leftMoveForbidden = 0;             //the verdict of the left neighbour if move is allowed
  int rightMoveForbidden = 0;            //the verdict of the right neighbour if move is allowed
  int currentVelocity = 1;               //the current particle velocity
  ///////////////////////////////////////////////////////////////////////
  //
  //   ADD YOUR CODE HERE TO SIMULATE PARTICLE BEHAVIOUR
  //
  ///////////////////////////////////////////////////////////////////////

}
```

```
//MAIN PROCESS defining channels, orchestrating and starting the threads
int main(void) {
        chan quadrant[4];               //helper channels for LED visualisation
        chan show[noParticles];         //channels to link visualiser with particles
        chan neighbours[noParticles];   //channels to link neighbouring particles
        chan buttonToVisualiser;        //channel to link buttons and visualiser

        //MAIN PROCESS HARNESS
        par{

          //BUTTON LISTENER THREAD
          on stdcore[0]: buttonListener(buttons,buttonToVisualiser);

          ///////////////////////////////////////////////////////////////////////
          //
          //   ADD YOUR CODE HERE TO REPLICATE PARTICLE THREADS particle(…)
          //
          ///////////////////////////////////////////////////////////////////////

          //VISUALISER THREAD
          on stdcore[0]: visualiser(buttonToVisualiser,show,quadrant,speaker);

          //REPLICATION FOR THREADS PERFORMING LED VISUALISATION
          par (int k=0;k<4;k++) {
            on stdcore[k%4]: showLED(cled[k],quadrant[k]);
          }

        }
        return 0;
}
```