COMS20600 Concurrency (2012-13) University of Bristol

Assignment 1 ("LED Ant Defender Game" 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, Mo 12th 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 ***Tue 13th Nov 2012 and 10am on Fr 20th Nov 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 Defender Game" on XMOS XC-1A board

You are given an XC code skeleton that provides you with the structure and helper routines to implement a basic game on the XC-1A board. Your task is to extend the given skeleton code to implement the following game: An "LED Ant" is represented by a position on the clock wheel of the XC-1A board. Each "LED Ant" is visualised by one active red LED on the 12-position LED clock marked with LED labels I, II, II,…, XII. No two LED Ants can have the same position on the clock. During the game, the user has to defend LED positions I, XII and XI from an LED Attacker Ant by controlling one LED Defender Ant. Once you have implemented a working, stable and well playable game, you may want to showcase your skills in programming the XC-1A by adding further concurrent game features of your choice for top marks.

Defender Ant

The user controls one "LED Ant" by pressing either **button A** (moving 1 position clockwise) or **button D** (moving 1 position anti-clockwise). The defender ant can only move to a position that is not already occupied by the attacker ant. The defender's starting position is LED XII. A sound is played when the user presses a button.

Attacker Ant

A second "LED Ant" is controlled by the system and starts at LED position VI. It then attempts moving in one direction (either clockwise or anti-clockwise). This attempt is denied if the defender ant is already located there, in this case the attacker ant changes direction. To make the game more interesting: before attempting the n^th^ move, the attacker ant will change direction if n is divisible by 31, 37 or 43. The game ends when the attacker has reached any one of the LED positions I, XII or XI.

About the Skeleton Code

Your task is to implement missing code in the processes `userAnt`, `attackerAnt` and `controller` according to the game description. The defender is controlled by a process called `userAnt`, which has channels to `buttonListener`, `visualise` and `controller`. The process `buttonListener` is implemented for you; it sends the button inputs to `userAnt`, the value 7 sent indicates button A pressed on its own and value 14 sent indicates button D pressed on its own. The `visualiser` is also implemented for you; it waits to receive the current position of the ants and instructs the board to switch LEDs accordingly.

The attacker ant is controlled by a process `attackerAnt`, which has channels to the `visualiser` and `controller`. The `controller` process responds to "permission-to-move" requests from `attackerAnt` and `userAnt`. The process also checks if an `attackerAnt` has moved to LED positions I, XII and XI.

```
  /////////////////////////////////////////////////////////////////////////////////////////
  //
  // COMS20600 - WEEKS 3 and 4
  // ASSIGNMENT 1
  // CODE SKELETON
  // TITLE: "LED Ant Defender Game"
  //
  // - this is the first assessed piece of coursework in the unit
  // - this assignment is to be completed in pairs during week 3 and 4
  // - it is worth 10% of the unit (i.e. 20% of the course work component)
  //
  // OBJECTIVE: given a code skeleton with threads and channels setup for you,
  //            implement a basic concurrent system on the XC-1A board
  //
  // NARRATIVE: You are given an XC code skeleton that provides you with
  // the structure and helper routines to implement a basic game on the
  // XC-1A board. Your task is to extend the given skeleton code to implement
  // the following game:
  //
  // An “LED Ant” is represented by a position on the clock wheel of the
  // XC-1A board. Each “LED Ant” is visualised by one active red LED on
  // the 12-position LED clock marked with LED labels I, II, II,…, XII.
  // No two LED Ants can have the same position on the clock. During the
  // game, the user has to defend LED positions I, XII and XI from an
  // LED Attacker Ant by controlling one LED Defender Ant and blocking the
  // attacker's path.
  //
  // Defender Ant
  // The user controls one “LED Ant” by pressing either button A (moving
  // 1 position clockwise) or button D (moving 1 position anti-clockwise).
  // The defender ant can only move to a position that is not already occupied
  // by the attacker ant. The defender’s starting position is LED XII. A sound
  // is played when the user presses a button.
  //
  // Attacker Ant
  // A second “LED Ant” is controlled by the system and starts at LED position VI.
  // It then attempts moving in one direction (either clockwise or anti-clockwise).
  // This attempt is denied if the defender ant is already located there, in this
  // case the attacker ant changes direction. To make the game more interesting:
  // before attempting the nth move, the attacker ant will change direction if n is
  // divisible by 23, 37 or 41. The game ends when the attacker has reached any one
  // of the LED positions I, XII or XI.
  //
  /////////////////////////////////////////////////////////////////////////////////////////

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

out port cled0 = PORT_CLOCKLED_0;
out port cled1 = PORT_CLOCKLED_1;
out port cled2 = PORT_CLOCKLED_2;
out port cled3 = 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;

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

//DISPLAYS an LED pattern in one quadrant of the clock LEDs
int showLED(out port p, chanend fromVisualiser) {
  unsigned int lightUpPattern;
  while (1) {
        fromVisualiser :> lightUpPattern; //read LED pattern from visualiser process
    p <: lightUpPattern;              //send pattern to LEDs
  }
  return 0;
}
```

```
//PROCESS TO COORDINATE DISPLAY of LED Ants
void visualiser(chanend fromUserAnt, chanend fromAttackerAnt, chanend toQuadrant0, chanend toQuadrant1, chanend
toQuadrant2, chanend toQuadrant3) {
  unsigned int userAntToDisplay = 11;
  unsigned int attackerAntToDisplay = 5;
  int i, j;
  cledR <: 1;
  while (1) {
        select {
          case fromUserAnt :> userAntToDisplay:
          break;
          case fromAttackerAnt :> attackerAntToDisplay:
          break;
        }
        j = 16<<(userAntToDisplay%3);
        i = 16<<(attackerAntToDisplay%3);
        toQuadrant0 <: (j*(userAntToDisplay/3==0)) + (i*(attackerAntToDisplay/3==0)) ;
        toQuadrant1 <: (j*(userAntToDisplay/3==1)) + (i*(attackerAntToDisplay/3==1)) ;
        toQuadrant2 <: (j*(userAntToDisplay/3==2)) + (i*(attackerAntToDisplay/3==2)) ;
        toQuadrant3 <: (j*(userAntToDisplay/3==3)) + (i*(attackerAntToDisplay/3==3)) ;
  }
}

//PLAYS a short sound (pls use with caution and consideration to other students in the labs!)
void playSound(unsigned int wavelength, out port speaker) {
  timer  tmr;
  int t, isOn = 1;
  tmr :> t;
  for (int i=0; i<2; i++) {
    isOn = !isOn;
    t += wavelength;
    tmr when timerafter(t) :> void;
    speaker <: isOn;
  }
}

//READ BUTTONS and send to userAnt
void buttonListener(in port b, out port spkr, chanend toUserAnt) {
  int r;
  while (1) {
    b when pinsneq(15) :> r;   // check if some buttons are pressed
    playSound(2000000,spkr);   // play sound
    toUserAnt <: r;            // send button pattern to userAnt
  }
}

//WAIT function
void waitMoment() {
  timer tmr;
  int waitTime;
  waitTime += 8000000;
  tmr when timerafter(waitTime) :> void;
}



/////////////////////////////////////////////////////////////////////////////////////////
//
//  RELEVANT PART OF CODE TO EXPAND FOR YOU
//
/////////////////////////////////////////////////////////////////////////////////////////


//DEFENDER PROCESS... The defender is controlled by this process userAnt,
//                    which has channels to a buttonListener, visualiser and controller
void userAnt(chanend fromButtons, chanend toVisualiser, chanend toController) {
  unsigned int userAntPosition = 11;       //the current defender position
  int buttonInput;                         //the input pattern from the buttonListener
  unsigned int attemptedAntPosition = 0;   //the next attempted defender position after considering button
  int moveForbidden;                       //the verdict of the controller if move is allowed
  toVisualiser <: userAntPosition;         //show initial position
  while (1) {
        fromButtons :> buttonInput;
        if (buttonInput == 14) attemptedAntPosition = userAntPosition + 1;
        if (buttonInput == 7)  attemptedAntPosition = userAntPosition - 1;

        ////////////////////////////////////////////////////////////
        //
        // !!! place code here for userAnt behaviour
        //
        /////////////////////////////////////////////////////////////
  }
}
```

```
//ATTACKER PROCESS... The attacker is controlled by this process attackerAnt,
//                    which has channels to the visualiser and controller
void attackerAnt(chanend toVisualiser, chanend toController) {
  int moveCounter = 0;                       //moves of attacker so far
  unsigned int attackerAntPosition = 5;      //the current attacker position
  unsigned int attemptedAntPosition;         //the next attempted  position after considering move direction
  int currentDirection = 1;                  //the current direction the attacker is moving
  int moveForbidden = 0;                     //the verdict of the controller if move is allowed
  toVisualiser <: attackerAntPosition;       //show initial position
  while (1) {
        ////////////////////////////////////////////////////////////
        //
        // !!! place your code here for attacker behaviour
        //
        /////////////////////////////////////////////////////////////
    waitMoment();
  }
}

//COLLISION DETECTOR... the controller process responds to “permission-to-move” requests
//                      from attackerAnt and userAnt. The process also checks if an attackerAnt
//                      has moved to LED positions I, XII and XI.
void controller(chanend fromAttacker, chanend fromUser) {
  unsigned int lastReportedUserAntPosition = 11;      //position last reported by userAnt
  unsigned int lastReportedAttackerAntPosition = 5;   //position last reported by attackerAnt
  unsigned int attempt = 0;
  fromUser :> attempt;                                //start game when user moves
  fromUser <: 1;                                      //forbid first move
  while (1) {
        select {
          case fromAttacker :> attempt:
                 /////////////////////////////////////////////////////////////
                 //
                 // !!! place your code here to give permission/deny attacker move or to end game
                 //
                 /////////////////////////////////////////////////////////////
                 break;
          case fromUser :> attempt:
                 /////////////////////////////////////////////////////////////
                 //
                 // !!! place your code here to give permission/deny user move
                 //
                 /////////////////////////////////////////////////////////////
            break;
                 }
  }
}

//MAIN PROCESS defining channels, orchestrating and starting the processes
int main(void) {
        chan buttonsToUserAnt,         //channel from buttonListener to userAnt
             userAntToVisualiser,      //channel from userAnt to Visualiser
             attackerAntToVisualiser,  //channel from attackerAnt to Visualiser
             attackerAntToController,  //channel from attackerAnt to Controller
             userAntToController;      //channel from userAnt to Controller
        chan quadrant0,quadrant1,quadrant2,quadrant3; //helper channels for LED visualisation

        par{
          //PROCESSES FOR YOU TO EXPAND
          on stdcore[1]: userAnt(buttonsToUserAnt,userAntToVisualiser,userAntToController);
          on stdcore[2]: attackerAnt(attackerAntToVisualiser,attackerAntToController);
          on stdcore[3]: controller(attackerAntToController, userAntToController);

          //HELPER PROCESSES
          on stdcore[0]: buttonListener(buttons, speaker,buttonsToUserAnt);
          on stdcore[0]: visualiser(userAntToVisualiser,attackerAntToVisualiser,quadrant0,quadrant1,quadrant2,quadrant3);
          on stdcore[0]: showLED(cled0,quadrant0);
          on stdcore[1]: showLED(cled1,quadrant1);
          on stdcore[2]: showLED(cled2,quadrant2);
          on stdcore[3]: showLED(cled3,quadrant3);
        }
        return 0;
}
```