/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Copyright (C) 2023 Advanced Micro Devices, Inc. All Rights Reserved.

\* SPDX-License-Identifier: MIT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* helloworld.c: simple test application

\*

\* This application configures UART 16550 to baud rate 9600.

\* PS7 UART (Zynq) is not initialized by this application, since

\* bootrom/bsp configures it to baud rate 115200

\*

\* ------------------------------------------------

\* | UART TYPE BAUD RATE |

\* ------------------------------------------------

\* uartns550 9600

\* uartlite Configurable only in HW design

\* ps7\_uart 115200 (configured by bootrom/bsp)

\*/

#include <stdio.h>

#include "platform.h"

#include "xparameters.h"

#include "xil\_printf.h"

#include "xil\_io.h"

#include "xgpio.h"

#include "myip.h"

#include "sleep.h"

#include "xtmrctr.h"

#define NUM\_TASKS 3

#define sec 100000000

#define millis 100000

#define micros 1000

// Register assignments

uint32\_t \*reg0 = XPAR\_MYIP\_0\_S00\_AXI\_BASEADDR; // block1: brick13-0(13-0), brick27-14(27-14)

uint32\_t \*reg1 = (uint32\_t \*)0x44a00004; // block2

uint32\_t \*reg2 = (uint32\_t \*)0x44a00008; // block3

uint32\_t \*reg3 = (uint32\_t \*)0x44a0000c; // ballX(9-0), ballY(19-10)

uint32\_t \*reg4 = (uint32\_t \*)0x44a00010; // paddleX(9-0), bigPaddleEn(10)

uint32\_t \*reg5 = (uint32\_t \*)0x44a00014; // score (11-0)[BCD], lives(15-12)

uint32\_t \*reg6 = (uint32\_t \*)0x44a00018;

uint32\_t \*reg7 = (uint32\_t \*)0x44a0001c;

// Global variables

uint32\_t block1 = 0x0FFFFFFF;

uint32\_t block2 = 0x0FFFFFFF;

uint32\_t block3 = 0x0FFFFFFF;

uint16\_t ballX = 210;

uint16\_t ballY = 400;

char ballXs = 1; // ball x speed

char ballYs = -1; // ball y speed

uint16\_t paddleX = 307; // 130 to 510

uint8\_t bigPaddle = 0;

uint16\_t score = 420; // decimal val not bcd

uint8\_t lives = 9;

uint32\_t startTime = 0;

uint32\_t counter = 0;

uint8\_t b = 0;

uint8\_t btns = 0;

uint8\_t inPlay = 1;

uint8\_t btnPrev = 0;

XTmrCtr timer;

XGpio buttons;

// function decs

void drawBricks();

void drawBall();

void drawPaddle();

void drawScore();

uint32\_t time();

void demo();

int main()

{

init\_platform();

// initialize buttons

XGpio\_Initialize(&buttons, XPAR\_AXI\_GPIO\_0\_DEVICE\_ID);

XGpio\_SetDataDirection(&buttons, 1, 0xF);

// initialize timer

XTmrCtr\_Initialize(&timer, XPAR\_AXI\_TIMER\_0\_BASEADDR);

XTmrCtr\_SetResetValue(&timer,0, 1294967296);

XTmrCtr\_Start(&timer, 0);

uint32\_t tmr1 = time();

uint32\_t tmr2 = time();

while(1) {

if (time() == 0) XTmrCtr\_Reset(&timer,0);

if (time() - tmr1 > 7\*millis){ // everthing here updates every 7 ms

drawBall();

drawBricks();

drawScore();

tmr1 = time();

}

if (time() - tmr2 > 4\*millis){

drawPaddle();

tmr2 = time();

}

}

cleanup\_platform();

return 0;

}

void drawBricks(){

if (inPlay){

\*reg0 = block1;

\*reg1 = block2;

\*reg2 = block3;

}

}

void drawBall(){

// Handle all the known borders

if (ballX + ballXs <= 130) ballXs = -ballXs;

if (ballX + ballXs >= 503) ballXs = -ballXs;

if (ballY + ballYs <= 12) ballYs = -ballYs;

// check for paddle collision

if (ballY + ballYs >= 445){

if ((ballX >= paddleX - 7) && (ballX <= paddleX + 27)){ // if the ball is within left bounds

if (ballX < paddleX + 14){ // if the ball is left of the middle of the paddle

ballXs &= 0b01111111;

//ballXs = -1;

} else {

ballXs |= 0b10000000;

//ballXs = 1;

}

ballYs = -ballYs;

}

//&& (ballX >= paddleX || ballX+7 <= paddleX +27)) ballYs = -ballYs;

} else score = 0;

ballX += ballXs;

ballY += ballYs;

/\*

if (ballY < 445 && ballY > 168){

if (ballX < 503 && ballX > 130){

ballX += ballXs;

ballY += ballYs;

} else {

ballXs = - ballXs;

ballX += ballXs;

}

} else {

ballYs = -ballYs;

ballY += ballYs;

}

\*/

\*reg3 = (ballY << 10) | (ballX & 0b1111111111); // concatenate the bits but make sure no extra

}

void drawPaddle(){

// get button input

btns = XGpio\_DiscreteRead(&buttons,1);

// Move the paddle

if ((btns == 2) & (paddleX >= 130)) paddleX--;

if ((btns == 4) & (paddleX <= 483)) paddleX++;

if ((btns == 8)&& (btnPrev == 0)){

bigPaddle = !bigPaddle;

btnPrev = btns;

}

\*reg4 = (bigPaddle << 10) | (paddleX & 0b1111111111);

}

void drawScore()

{

// scoreboard

uint16\_t bcd = 0;

uint16\_t shift = 0;

uint16\_t tmp = score;

while (tmp > 0){

char digit = tmp % 10;

bcd |= (digit << shift);

tmp /= 10;

shift += 4;

}

\*reg5 = (lives << 12) | bcd; // update lives also

}

void demo(){

drawPaddle();

// Move the blocks

// block1 = 0x0AAA9555; // alternating

if (((counter & 0x1FFFFF) == 0x1F0000) && (b == 0)){

b = 1;

}

if((b == 1) && ((counter & 0x1FFF) == 0x1000)){

if (block3 > 0){

block3 = block3 << 1;

} else if (block2 > 0){

block2 = block2 << 1;

} else if (block1 > 0){

block1 = block1 << 1;

} else {

block1 = block2 = block3 = 0x0FFFFFFF;

b = 0;

}

}

drawBricks();

// Inc the score and dec lives

if ((counter & 0x1FFF) == 0x1000){

score++;

if (score == 999) score = 0;

if (lives == 0) lives = 10;

lives--;

}

drawScore();

// Move the ball

if ((counter % 0x0400) == 0){

if (ballY < 445 && ballY > 168){

if (ballX < 503 && ballX > 130){

ballX += ballXs;

ballY += ballYs;

} else {

ballXs = - ballXs;

ballX += ballXs;

}

} else {

ballYs = -ballYs;

ballY += ballYs;

}

}

drawBall();

}

uint32\_t time(){

return XTmrCtr\_GetValue(&timer, 0);

}

#define wallR 130

#define wallL 510

#define top 12

#define paddleY 450

#define paddleW 27

#define paddleH 10

#define numBricks 84

char bigPaddleEn = 0;

unsigned int ballX, ballY; paddleX, bricks1, bricks2, bricks3, score;

char ballXs, ballYs, inPlay, lives;

unsigned int brickRow [6] = {0,0,0,0,0,0};

void gameOver();

int main(){

brickRow[0] = bricks1 & 0xFFFA000; // 1111 1111 1111 1100

brickRow[1] = bricks1 & 0x0003FFF; // 0011 1111 1111 1111

brickRow[2] = bricks2 & 0xFFFA000;

brickRow[3] = bricks2 & 0x0003FFF;

brickRow[4] = bricks3 & 0xFFFA000;

brickRow[5] = bricks3 & 0x0003FFF;

}

void drawBall(){

// update the ball coordinates

ballX += ballXs;

ballY += ballYs;

// Handle all normal space collisions

if (ballX <= wallL || ballX >= wallR) ballXs = -ballXs; // left and right walls

if (ballY <= top) ballYs = -ballYs; // top of the screen

// Handle brick collisions

if (ballY >= 100 && ballY <= 108){ // if ballY is within row 1 of bricks

// check the whole row for collisions with any brick's x-coordinate

for (char x = 0; x < 14; x++){ //check row 1

if (brickRow[0] & (1 << x) > 0){ // if brick x is unbroken

if ( (ballX >= (132 + 27\*x)) && (ballX <= (157 + 27\*x)) ) // if the ballX overlaps the brick's X

brickRow[0] &= ~(unsigned int)(1 << x); // might not work; set bit X to 0

}

}

} else if (ballY >= 110 && ballY <= 118){

}

// Handle paddle collision and miss

if (ballY >= paddleY && !bigPaddleEn){ // if ball is in the right y space

if ((ballX + 7 <= paddleX) || (ballX >= paddleX + 27)){ // if the ball did not hit the paddle

// game over; stop updating bricks, set inPlay to 0, enable bigPaddle.

if (lives - 1 == 0){

inPlay = 0;

bigPaddleEn = 1;

} else { // still alive; reset ball pos, delay, dec lives

lives--;

usleep(5000000); // 5 seconds

ballY = 210;

ballX = (time() % 400) + 130;

}

} else { // it did hit the paddle: give the ball a trajectory accordingly

ballXs = ((ballX + 3) - (paddleX + 13)) / 4;

ballYs = -ballYs;

}

} else ballYs = -ballYs;

}

void resetAll(){

bigPaddleEn = 0;

lives = 4;

score = 0;

bricks1 = 0xFFFFFFFF;

bricks2 = 0xFFFFFFFF;

bricks3 = 0xFFFFFFFF;

ballX = (time() % 400) + 130;

ballY = 210;

}