#include <stdio.h>

#include <xio.h>

#include <stdbool.h>

#include "xparameters.h"

#include "cam\_ctrl\_header.h"

#include "vmodcam\_header.h"

#define blDvmaCR 0x00000000 // Control Reg Offset

#define blDvmaFWR 0x00000004 // Frame Width Reg Offset

#define blDvmaFHR 0x00000008 // Frame Height Reg Offset

#define blDvmaFBAR 0x0000000c // Frame Base Addr Reg Offset

#define blDvmaFLSR 0x00000010 // Frame Line Stride Reg Offeset

#define blDvmaHSR 0x00000014 // H Sync Reg Offset

#define blDvmaHBPR 0x00000018 // H Back Porch Reg Offset

#define blDvmaHFPR 0x0000001c // H Front Porch Reg Offset

#define blDvmaHTR 0x00000020 // H Total Reg Offset

#define blDvmaVSR 0x00000024 // V Sync Reg Offset

#define blDvmaVBPR 0x00000028 // V Back Porch Reg Offset

#define blDvmaVFPR 0x0000002c // V Front Porch Reg Offset

#define blDvmaVTR 0x00000030 // V Total Reg Offset

bool get\_switch() {

return true;

}

int main() {

// Robot variables.

short robot\_size = 25;

short robot\_min\_x\_area = 0;

short robot\_max\_x\_area = 640;

short robot\_min\_y\_area = 0;

short robot\_max\_y\_area = 240;

short robot\_min\_x\_position = robot\_min\_x\_area + robot\_size;

short robot\_max\_x\_position = robot\_max\_x\_area - robot\_size;

short robot\_min\_y\_position = robot\_min\_y\_area + robot\_size;

short robot\_max\_y\_position = robot\_max\_y\_area - robot\_size;

short robot\_x\_velocity = 0;

short robot\_y\_velocity = 0;

short robot\_x\_position = robot\_max\_x\_area / 2;

short robot\_y\_position = robot\_max\_y\_area / 2;

// Camera variables.

bool show\_robot\_vision = true;

// Code

u32 lDvmaBaseAddress = XPAR\_DVMA\_0\_BASEADDR;

short posX = 0;

short posY = 0;

for (posX = 0; posX < 2560; posX++) {

for (posY = 0; posY < 720; posY++) {

XIo\_Out16(XPAR\_DDR2\_SDRAM\_MPMC\_BASEADDR + 2 \* (posY \* 2560 + posX),

(posX / 40) << 3);

}

}

XIo\_Out32(lDvmaBaseAddress + blDvmaHSR, 40); // hsync

XIo\_Out32(lDvmaBaseAddress + blDvmaHBPR, 260); // hbpr

XIo\_Out32(lDvmaBaseAddress + blDvmaHFPR, 1540); // hfpr

XIo\_Out32(lDvmaBaseAddress + blDvmaHTR, 1650); // htr

XIo\_Out32(lDvmaBaseAddress + blDvmaVSR, 5); // vsync

XIo\_Out32(lDvmaBaseAddress + blDvmaVBPR, 25); // vbpr

XIo\_Out32(lDvmaBaseAddress + blDvmaVFPR, 745); // vfpr

XIo\_Out32(lDvmaBaseAddress + blDvmaVTR, 750); // vtr

XIo\_Out32(lDvmaBaseAddress + blDvmaFWR, 0x00000500); // frame width

XIo\_Out32(lDvmaBaseAddress + blDvmaFHR, 0x000002D0); // frame height

XIo\_Out32(lDvmaBaseAddress + blDvmaFBAR, XPAR\_DDR2\_SDRAM\_MPMC\_BASEADDR); // frame base addr

XIo\_Out32(lDvmaBaseAddress + blDvmaFLSR, 0x00000A00); // frame line stride

XIo\_Out32(lDvmaBaseAddress + blDvmaCR, 0x00000003); // dvma enable, dfl enable

/\*

\* Camera Initialization

\*/

// CamIicCfg(XPAR\_CAM\_IIC\_0\_BASEADDR, CAM\_CFG\_640x480P);

//

// for (posX = 0; posX < 2560; posX++)

// {

// for (posY = 0; posY < 720; posY++)

// {

// XIo\_Out16(XPAR\_DDR2\_SDRAM\_MPMC\_BASEADDR + 2 \* (posY \* 2560 + posX), 0X0F0);

// }

// }

//

// CamCtrlInit(XPAR\_CAM\_CTRL\_0\_BASEADDR, CAM\_CFG\_640x480P, 640 \* 2);

//

// Draw the background color

for (posX = 0; posX < 2560; posX++) {

for (posY = 0; posY < 720; posY++) {

XIo\_Out16(XPAR\_DDR2\_SDRAM\_MPMC\_BASEADDR + 2 \* (posY \* 2560 + posX),

0xFFF);

}

}

short color\_square[10][10];

short i = 0;

short j = 0;

for (i = 0; i < 10; i++) {

for (j = 0; j < 10; j++) {

color\_square[i][j] = 0;

}

}

short last\_red\_row = 0;

short last\_red\_column = 0;

short count = 0;

while (true) {

count += 1;

bool new\_switch\_value = get\_switch();

if (show\_robot\_vision != new\_switch\_value) {

show\_robot\_vision = new\_switch\_value;

// Clear the top-left of the screen.

for (posX = 0; posX < 640; posX++) {

for (posY = 0; posY < 480; posY++) {

XIo\_Out16(

XPAR\_DDR2\_SDRAM\_MPMC\_BASEADDR + 2 \* (posY \* 2560 + posX),

0xFFF);

}

}

}

// Scanning through pixels

short row = 0;

short col = 0;

short pos\_x = 0;

short pos\_y = 0;

short most\_red\_row = 0;

short most\_red\_column = 0;

for (row = 0; row < 10; row++) {

for (col = 0; col < 10; col++) {

short total\_red = 0;

for (pos\_x = row \* 64; pos\_x < (row + 1) \* 64; ++pos\_x) {

for (pos\_y = col \* 48; pos\_y < (col + 1) \* 48; ++pos\_y) {

u16 color =

XIo\_In16(XPAR\_DDR2\_SDRAM\_MPMC\_BASEADDR + 2 \* (pos\_y \* 2560 + pos\_x + 640));

if (show\_robot\_vision) {

if (((color >> 8) & 0b1111) == 15) {

XIo\_Out16(

XPAR\_DDR2\_SDRAM\_MPMC\_BASEADDR + 2 \* (pos\_y \* 2560 + pos\_x),

color);

} else {

XIo\_Out16(

XPAR\_DDR2\_SDRAM\_MPMC\_BASEADDR + 2 \* (pos\_y \* 2560 + pos\_x),

0);

}

}

if (((color >> 8) & 0b1111) == 15) {

total\_red += ((color >> 8) & 0b1111);

}

}

}

color\_square[row][col] = total\_red;

}

}

// End Scanning

// finding most red

for (row = 0; row < 10; ++row) {

for (col = 0; col < 10; ++col) {

if (color\_square[row][col]

> color\_square[most\_red\_row][most\_red\_column]) {

most\_red\_row = row;

most\_red\_column = col;

}

}

} // End finding red

/\*\*\* Drawing \*\*\*/

// Stay at the point

if (!(most\_red\_row == last\_red\_row && most\_red\_column == last\_red\_column)) {

if (!show\_robot\_vision) {

// Draw our starting point in green

for (pos\_x = last\_red\_row \* 64; pos\_x < (last\_red\_row + 1) \* 64;

++pos\_x) {

for (pos\_y = last\_red\_column \* 48;

pos\_y < (last\_red\_column + 1) \* 48; ++pos\_y) {

XIo\_Out16(

XPAR\_DDR2\_SDRAM\_MPMC\_BASEADDR + 2 \* (pos\_y \* 2560 + pos\_x),

0x00F0);

}

}

// Draw our target point in red

for (pos\_x = most\_red\_row \* 64; pos\_x < (most\_red\_row + 1) \* 64;

++pos\_x) {

for (pos\_y = most\_red\_column \* 48;

pos\_y < (most\_red\_column + 1) \* 48; ++pos\_y) {

XIo\_Out16(

XPAR\_DDR2\_SDRAM\_MPMC\_BASEADDR + 2 \* (pos\_y \* 2560 + pos\_x),

0x0F00);

}

}

}

/\*\*\* End of Drawing \*\*\*/

/\*\*\* Drawing the irobot path \*\*\*/

for (row = 0; row < 10; row++) {

for (col = 0; col < 10; col++) {

// Draw the current block in red.

if (row == most\_red\_row && col == most\_red\_column) {

for (pos\_x = row \* 64; pos\_x < row \* 64 + 5; ++pos\_x) {

for (pos\_y = 480 + col \* 24;

pos\_y < 480 + col \* 24 + 5; ++pos\_y) {

XIo\_Out16(

XPAR\_DDR2\_SDRAM\_MPMC\_BASEADDR + 2 \* (pos\_y \* 2560 + pos\_x + 640),

0x0F00);

}

}

}

// Draw the last block in green.

else if (row == last\_red\_row && col == last\_red\_column) {

for (pos\_x = row \* 64; pos\_x < row \* 64 + 5; ++pos\_x) {

for (pos\_y = 480 + col \* 24;

pos\_y < 480 + col \* 24 + 5; ++pos\_y) {

XIo\_Out16(

XPAR\_DDR2\_SDRAM\_MPMC\_BASEADDR + 2 \* (pos\_y \* 2560 + pos\_x + 640),

0x00F0);

}

}

}

// Draw everything else in black.

else {

for (pos\_x = row \* 64; pos\_x < row \* 64 + 5; ++pos\_x) {

for (pos\_y = 480 + col \* 24;

pos\_y < 480 + col \* 24 + 5; ++pos\_y) {

XIo\_Out16(

XPAR\_DDR2\_SDRAM\_MPMC\_BASEADDR + 2 \* (pos\_y \* 2560 + pos\_x + 640),

0x0);

}

}

}

}

}

robot\_x\_velocity = most\_red\_row - last\_red\_row;

robot\_y\_velocity = most\_red\_column - last\_red\_column;

/\*\*\* End of Drawing the irobot path \*\*\*/

last\_red\_row = most\_red\_row;

last\_red\_column = most\_red\_column;

}

///////////////////////////////////////////////////////////////////////////////////////////////

// ROBOT

///////////////////////////////////////////////////////////////////////////////////////////////

// Find the robot's new position.

if (robot\_x\_position + robot\_x\_velocity >= robot\_min\_x\_position

&& robot\_x\_position + robot\_x\_velocity

<= robot\_max\_x\_position) {

robot\_x\_position += robot\_x\_velocity;

}

if (robot\_y\_position + robot\_y\_velocity >= robot\_min\_y\_position

&& robot\_y\_position + robot\_y\_velocity

<= robot\_max\_y\_position) {

robot\_y\_position += robot\_y\_velocity;

}

// Draw the robot.

// The x-coordinates of the robot playing area go from 0 to 640.

// The y-coordinates go from 480 to 720.

for (posX = 0 + robot\_min\_x\_area; posX < 0 + robot\_max\_x\_area; posX++) {

// Shift everything down by 480 and to the right by 0

// so that the robot playing area is in the bottom-left corner.

for (posY = 480 + robot\_min\_x\_area; posY < 480 + robot\_max\_x\_area;

posY++) {

if (posX >= 0 + robot\_x\_position - robot\_size

&& posX <= 0 + robot\_x\_position + robot\_size

&& posY >= 480 + robot\_y\_position - robot\_size

&& posY <= 480 + robot\_y\_position + robot\_size) {

XIo\_Out16(

XPAR\_DDR2\_SDRAM\_MPMC\_BASEADDR + 2 \* (posY \* 2560 + posX),

0xF0F);

} else {

XIo\_Out16(

XPAR\_DDR2\_SDRAM\_MPMC\_BASEADDR + 2 \* (posY \* 2560 + posX),

0xFFF);

}

}

}

}

}