```c
/*
 * helloworld.c: simple test application
 * Currently used to test lab 3 for Space Invaders.
 * Taylor Cowley and Andrew Okazaki
 */

#include <stdio.h>
#include <stdint.h>
#include "platform.h"
#include "xparameters.h"
#include "xaxivdma.h"
#include "xio.h"
#include "time.h"
#include "unistd.h"
#include "tank.h"
#include "interface.h"
#include "aliens.h"
#include "bunkers.h"
#include "mother_ship.h"
#include "util.h"
#include "sound/xac97_l.h"
#include "sound/sound.h"
#include "xgpio.h"
#include "mb_interface.h"
#include "xintc_l.h"
#include "sound/sound.h"
#include "pit.h"



#define DEBUG

#define SCREEN_RES_X 640    // Our screen resolution is 640 * 480
#define SCREEN_RES_Y 480    // Our screen resolution is 640 * 480
#define BLACK 0x00000000    // Hex value for black
#define BLUE  0x2222FF

#define ONE_SECOND  100     // 100 ticks in a second
#define HALF_SECOND 50      // 50 ticks in half a second
#define QUARTER_SECOND 25   // 25 ticks in a quarter second
#define EIGHTH_SECOND 12    // 12 ticks in an eigth second
#define TENTH_SECOND 10     // 10 ticks in a tenth second
#define TWENTIETH_SECOND 5  // 5 ticks in a twentieth second
#define SUPER_FAST 2        // super fast

#define MOTHER_SHIP_SPEED TENTH_SECOND      // Mother ship moves slowly
#define MOTHER_SHIP_SPAWN_CONSTANT 1000     // Mother ship spawns infrequently
#define ALIEN_SHOT_SPAWN_CONSTANT 100       // Aliens shoot frequently
#define ALIEN_MOVE_SPEED HALF_SECOND        // aliens move very slowly

#define BUTTON_UP       0x4 // Constants for button masks
#define BUTTON_DOWN     0x10
#define BUTTON_LEFT     0x8
#define BUTTON_RIGHT    0x2
#define BUTTON_CENTER   0x1

// All speed modifiers for the PIT timer
#define PIT_TENTH_SPEED 10000000            // This is really slow. like paused
```

```
#define PIT_FIFTH_SPEED 5000000             //
#define PIT_THREEFOURTHS_SPEED 7500000      //
#define PIT_NORMAL_SPEED 1000000            // The speed the FIT was running at
#define PIT_1_5x_speed 750000               //
#define PIT_2x_speed 500000                 //
#define PIT_4x_speed 250000                 //
#define PIT_10x_speed 100000                // 10x speed
#define PIT_100x_speed 10000                // 100x speed
#define PIT_LUDICROUS_SPEED PIT_100x_speed  // (same as 100x speed)

#define PIT_NO_OFFSET 0                     // For writing to registers
#define PIT_CONTROL_RUN 0x00000007          // Control value to make it RUN
#define PIT_GOOD_INITIAL_VALUE 100000000    // initial value for counter


void print(char *str);          // print exists!

#define FRAME_BUFFER_0_ADDR 0xC1000000  // Starting location in DDR

//-----------------------------
void timer_interrupt_handler();             // interrupt handler for timer
void pb_interrupt_handler();                // interrupt handler for buttons
void interrupt_handler_dispatcher(void *);  // dispatch the interrupts
void init_pit();                        // Init our pit registers to good values
void change_speed_on_input();
//-----------------------------


XGpio gpLED;  // This is a handle for the LED GPIO block.
XGpio gpPB;   // This is a handle for the push-button GPIO block.
uint32_t* framePointer0 = (uint32_t*) FRAME_BUFFER_0_ADDR;
int32_t currentButtonState;     // Current button being pressed
int32_t mother_ship_points;
uint32_t cpu_usage_timer = 0;
uint32_t sound_count = 0;


void timer_interrupt_handler(){
    static uint32_t timerCount;                 // Timer for timing
    static uint32_t mother_ship_move_counter;   // Timer for mother ship

    tank_update_bullet(framePointer0);          // update all bullets
    aliens_update_bullets(framePointer0);       // update all bullets

    timerCount++;                               // Increment all counters
    mother_ship_move_counter++;
    mother_ship_points++;

    int32_t r = rand();
    if(r%ALIEN_SHOT_SPAWN_CONSTANT == 0){
        alien_missle(framePointer0);    // Make the aliens fire
    }
    if(r%MOTHER_SHIP_SPAWN_CONSTANT == 0){
        mother_ship_spawn();            // mother ship spawns!
    }
    if(mother_ship_move_counter >= MOTHER_SHIP_SPEED){  // MS moves
        mother_ship_move_counter = 0;
        mother_ship_move();
    }
    if(mother_ship_points > TENTH_SECOND){
        mother_ship_points = 0;         // Mother ship points will display
        mother_ship_points_blink();
    }
    if(timerCount >= HALF_SECOND ){
        timerCount = 0;
        aliens_move(framePointer0); // move the aliens
    }


    // Now to check the buttons.
    if(currentButtonState & BUTTON_LEFT){
        tank_move_left(framePointer0);      // Moving the tank left
    }
    if(currentButtonState & BUTTON_RIGHT){
        tank_move_right(framePointer0);     // Moving the tank right
    }
    if(currentButtonState & BUTTON_CENTER){
        tank_fire(framePointer0);           // Fire the tank!
    }
    if(currentButtonState & BUTTON_UP){     // Not functional yet
        sound_vol_up();
    }
    if(currentButtonState & BUTTON_DOWN){       // Not functional yet
        sound_vol_down();
    }

}

// Interrupt handler for the push buttons
void pb_interrupt_handler(){
    XGpio_InterruptGlobalDisable(&gpPB);    // Can't be interrupted by buttons

    currentButtonState = XGpio_DiscreteRead(&gpPB, 1);
    // Time to clear the interrupt and reenable GPIO interrupts
    XGpio_InterruptClear(&gpPB, 0xFFFFFFFF);
    XGpio_InterruptGlobalEnable(&gpPB);
}

// We are making sound here :)
void sound_interrupt_handler(){
// Making sound!
    sound_run();
}

// Main interrupt handler, queries interrupt controller to see what peripheral
// fired the interrupt and then dispatches the corresponding interrupt handler.
// This routine acks the interrupt at the controller level but the peripheral
// interrupt must be ack'd by the dispatched interrupt handler.
// Question: Why is timer_interrupt_handler() called after ack'ing controller
// but pb_interrupt_handler() is called before ack'ing the interrupt controller?
void interrupt_handler_dispatcher(void* ptr) {
    int intc_status = XIntc_GetIntrStatus(XPAR_INTC_0_BASEADDR);
    // Check the FIT interrupt first.
    if (intc_status & XPAR_PIT_0_MYINTERRUPT_MASK){
        XIntc_AckIntr(XPAR_INTC_0_BASEADDR, XPAR_PIT_0_MYINTERRUPT_MASK);
        timer_interrupt_handler();  // It was a timer interrupt! call that fn
    }
    // Check the push buttons.
    if (intc_status & XPAR_PUSH_BUTTONS_5BITS_IP2INTC_IRPT_MASK){
        pb_interrupt_handler();     // It was a button interrupt!
        XIntc_AckIntr(XPAR_INTC_0_BASEADDR, // Acknowledge the interrupt
                XPAR_PUSH_BUTTONS_5BITS_IP2INTC_IRPT_MASK);
    }
     // Check the sound card
    if (intc_status & XPAR_AXI_AC97_0_INTERRUPT_MASK){
    // Acknowledge that interrupt
    XIntc_AckIntr(XPAR_INTC_0_BASEADDR, XPAR_AXI_AC97_0_INTERRUPT_MASK);
    sound_interrupt_handler();  // Make sound!
    }
}



// Initializes our PIT with proper values in its registers
void init_pit(){
    // Set up our count register with a good initial value
    PIT_mWriteSlaveReg0 (XPAR_PIT_0_BASEADDR, PIT_NO_OFFSET, PIT_GOOD_INITIAL_VALUE);

    // Set up our reload register with normal value: 100x a second, same as fit
    PIT_mWriteSlaveReg1 (XPAR_PIT_0_BASEADDR, PIT_NO_OFFSET, PIT_NORMAL_SPEED); // 100x a
second

    // Put value in control register to enable interrupts, reload, and count
    PIT_mWriteSlaveReg2 (XPAR_PIT_0_BASEADDR, PIT_NO_OFFSET, PIT_CONTROL_RUN);

}

void init_interrupts(void){
    int32_t success;
    init_pit();
    print("\n\rHello . Let's have a fun \e[31m\e[1mtime \e[21m\e[0m\n\r");
    success = XGpio_Initialize(&gpPB, XPAR_PUSH_BUTTONS_5BITS_DEVICE_ID);
    // Set the push button peripheral to be inputs.
    XGpio_SetDataDirection(&gpPB, 1, 0x0000001F);
    // Enable the global GPIO interrupt for push buttons.
    XGpio_InterruptGlobalEnable(&gpPB);
    // Enable all interrupts in the push button peripheral.
    XGpio_InterruptEnable(&gpPB, 0xFFFFFFFF);
    // Register the interrupt handler
    microblaze_register_handler(interrupt_handler_dispatcher, NULL);
    // And enable interrupts
     XIntc_EnableIntr(XPAR_INTC_0_BASEADDR, // interrupts to enable
    (XPAR_PIT_0_MYINTERRUPT_MASK |  // fit timer
            XPAR_PUSH_BUTTONS_5BITS_IP2INTC_IRPT_MASK | // buttons
                XPAR_AXI_AC97_0_INTERRUPT_MASK));   // sound card
    // Master the enable
    XIntc_MasterEnable(XPAR_INTC_0_BASEADDR);
    // And enable again
    microblaze_enable_interrupts();
}

// This changes the speed of the pit timer, and hence the game, based
// on a key input of 0-9
// THis is done by changing the value of the timer's reload register.
```

```
void change_speed_on_input(){
    char input = getchar();
    switch (input){ // And change the speed based on input
    case '1':
        PIT_mWriteSlaveReg1 (XPAR_PIT_0_BASEADDR, PIT_NO_OFFSET, PIT_TENTH_SPEED);  // 
10x a second
        break;
    case '2':
        PIT_mWriteSlaveReg1 (XPAR_PIT_0_BASEADDR, PIT_NO_OFFSET, PIT_FIFTH_SPEED);  // 
50x a second
        break;
    case '3':
        PIT_mWriteSlaveReg1 (XPAR_PIT_0_BASEADDR, PIT_NO_OFFSET, 
PIT_THREEFOURTHS_SPEED);    // 75x a second
        break;
    case '4':
        PIT_mWriteSlaveReg1 (XPAR_PIT_0_BASEADDR, PIT_NO_OFFSET, PIT_NORMAL_SPEED); // 1x 
speed
        break;
    case '5':
        PIT_mWriteSlaveReg1 (XPAR_PIT_0_BASEADDR, PIT_NO_OFFSET, PIT_1_5x_speed);   // 1.5
 speed
        break;
    case '6':
        PIT_mWriteSlaveReg1 (XPAR_PIT_0_BASEADDR, PIT_NO_OFFSET, PIT_2x_speed); // 2x 
speed
        break;
    case '7':
        PIT_mWriteSlaveReg1 (XPAR_PIT_0_BASEADDR, PIT_NO_OFFSET, PIT_4x_speed); // 4x 
speed
        break;
    case '8':
        PIT_mWriteSlaveReg1 (XPAR_PIT_0_BASEADDR, PIT_NO_OFFSET, PIT_10x_speed);    // 
10x speed
        break;
    case '9':
        PIT_mWriteSlaveReg1 (XPAR_PIT_0_BASEADDR, PIT_NO_OFFSET, 
PIT_LUDICROUS_SPEED);   // LUDICROUS SPEED
        break;
    case '0':
        PIT_mWriteSlaveReg1 (XPAR_PIT_0_BASEADDR, PIT_NO_OFFSET, PIT_NORMAL_SPEED); // 
100x a second NORMAL SPEED
        break;
    }
}

int main() {
    sound_init_AC_97();
    init_platform();                   // Necessary for all programs.
    init_interrupts();
    int Status;                        // Keep track of success/failure of system 
function calls.
    XAxiVdma videoDMAController;
    // There are 3 steps to initializing the vdma driver and IP.
    // Step 1: lookup the memory structure that is used to access the vdma driver.
    XAxiVdma_Config * VideoDMAConfig = XAxiVdma_LookupConfig(XPAR_AXI_VDMA_0_DEVICE_ID);
    // Step 2: Initialize the memory structure and the hardware.
    if(XST_FAILURE == XAxiVdma_CfgInitialize(&videoDMAController,
VideoDMAConfig, XPAR_AXI_VDMA_0_BASEADDR)) {
        xil_printf("VideoDMA Did not initialize.\r\n");
    }
    // Step 3: (optional) set the frame store number.
    if(XST_FAILURE ==  XAxiVdma_SetFrmStore(&videoDMAController, 2, XAXIVDMA_READ)) {
        xil_printf("Set Frame Store Failed.");
    }
    // Initialization is complete at this point.

    // Setup the frame counter. We want two read frames. We don't need any write frames
but the
    // function generates an error if you set the write frame count to 0. We set it to 2
    // but ignore it because we don't need a write channel at all.
    XAxiVdma_FrameCounter myFrameConfig;
    myFrameConfig.ReadFrameCount = 2;
    myFrameConfig.ReadDelayTimerCount = 10;
    myFrameConfig.WriteFrameCount =2;
    myFrameConfig.WriteDelayTimerCount = 10;
    Status = XAxiVdma_SetFrameCounter(&videoDMAController, &myFrameConfig);
    if (Status != XST_SUCCESS) {
        xil_printf("Set frame counter failed %d\r\n", Status);
        if(Status == XST_VDMA_MISMATCH_ERROR)
            xil_printf("DMA Mismatch Error\r\n");
    }
    // Now we tell the driver about the geometry of our frame buffer and a few other
things.
    // Our image is 480 x 640.
    XAxiVdma_DmaSetup myFrameBuffer;
    myFrameBuffer.VertSizeInput = 480;    // 480 vertical pixels.
    myFrameBuffer.HoriSizeInput = 640*4;  // 640 horizontal (32-bit pixels).
    myFrameBuffer.Stride = 640*4;         // Dont' worry about the rest of the values.
    myFrameBuffer.FrameDelay = 0;
    myFrameBuffer.EnableCircularBuf=1;
    myFrameBuffer.EnableSync = 0;
    myFrameBuffer.PointNum = 0;
    myFrameBuffer.EnableFrameCounter = 0;
    myFrameBuffer.FixedFrameStoreAddr = 0;
    if(XST_FAILURE == XAxiVdma_DmaConfig(&videoDMAController, XAXIVDMA_READ,
&myFrameBuffer)) {
        xil_printf("DMA Config Failed\r\n");
    }
    // We need to give the frame buffer pointers to the memory that it will use. This
memory
    // is where you will write your video data. The vdma IP/driver then streams it to the
HDMI
    // IP.
    myFrameBuffer.FrameStoreStartAddr[0] = FRAME_BUFFER_0_ADDR;
    myFrameBuffer.FrameStoreStartAddr[1] = FRAME_BUFFER_0_ADDR + 4*640*480;

    if(XST_FAILURE == XAxiVdma_DmaSetBufferAddr(&videoDMAController, XAXIVDMA_READ,
            myFrameBuffer.FrameStoreStartAddr)) {
        xil_printf("DMA Set Address Failed Failed\r\n");
    }
    // Print a sanity message if you get this far.
    xil_printf("Woohoo! I made it through initialization.\n\r");
    // Now, let's get ready to start displaying some stuff on the screen.
    // The variables framePointer and framePointer1 are just pointers to the base address
    // of frame 0 and frame 1.
```

```c
    uint32_t* framePointer0 = (uint32_t*) FRAME_BUFFER_0_ADDR;
    // Just paint some large red, green, blue, and white squares in different
    // positions of the image for each frame in the buffer (framePointer0 and
framePointer1).
    int row=0, col=0;
    for( row=0; row<SCREEN_RES_Y; row++) {
        for(col=0; col<SCREEN_RES_X; col++) {
            framePointer0[row*SCREEN_RES_X + col] = BLACK;
        }
    }

    bunkers_init(framePointer0);            // Init the bunkers
    tank_init();                            // initialize the tank
    tank_draw(framePointer0, false);        // draw the tank
    interface_init_board(framePointer0);    // draw the tanks at the top
    aliens_init(framePointer0);             // initialize aliens
    mother_ship_init(framePointer0);        // Init the mother ship

    // This tells the HDMI controller the resolution of your display (there must be a
better way to do this).
    XIo_Out32(XPAR_AXI_HDMI_0_BASEADDR, 640*480);

    // Start the DMA for the read channel only.
    if(XST_FAILURE == XAxiVdma_DmaStart(&videoDMAController, XAXIVDMA_READ)){
        xil_printf("DMA START FAILED\r\n");
    }
    int frameIndex = 0;
    // We have two frames, let's park on frame 0. Use frameIndex to index them.
    // Note that you have to start the DMA process before parking on a frame.

    if (XST_FAILURE == XAxiVdma_StartParking(&videoDMAController, frameIndex,
XAXIVDMA_READ)) {
        xil_printf("vdma parking failed\n\r");
    }

    // ------------------------------------------------------------
    // Required, or the whole program halts for an unidentified reason
    srand((unsigned)time( NULL ));
    // Why is this necessary?
    // --------------------------------------------------------------


    while(1){
        //      cpu_usage_timer++;
        // Now we wait for input. You can input 0-9, with varying speeds for the
        // clock depending on the number
        change_speed_on_input();
    }
    cleanup_platform();
    return 0;
}
```