Fantastic Tetris

Final Report

Benjie Tong(bt2414)

Weipeng Dang(wd2265)

Yanbo Zou(yz2839)

Yiran Tao(yt2487)

CSEE 4840 Embedded System Design

Spring 2016

Introduction:

The fantastic Teristic is a variation of a normal Teristic. The game will automatically generate three different blocks into the screen for the users to choose. The user can put the blocks into the grid(10*10 in dimension) in anywhere the user like. After the user put one block, the block is fixed in that position and can't be moved. After the user put all the three blocks into the grid, the game will check if the game is terminated or not. If it is not terminated, the grid will give three more blocks for the user to continue the game. The user should try to put blocks into the grid in order to form a line. If one horizontal or one vertical line is detected in the grid, that line is cleared. The game is monitored by a countdown timer, if 100 seconds passed, the game will be finished.

Hardware and Software Connections:

There are two parts for our whole system design: software and hardware. Below is an overview about all the connections:

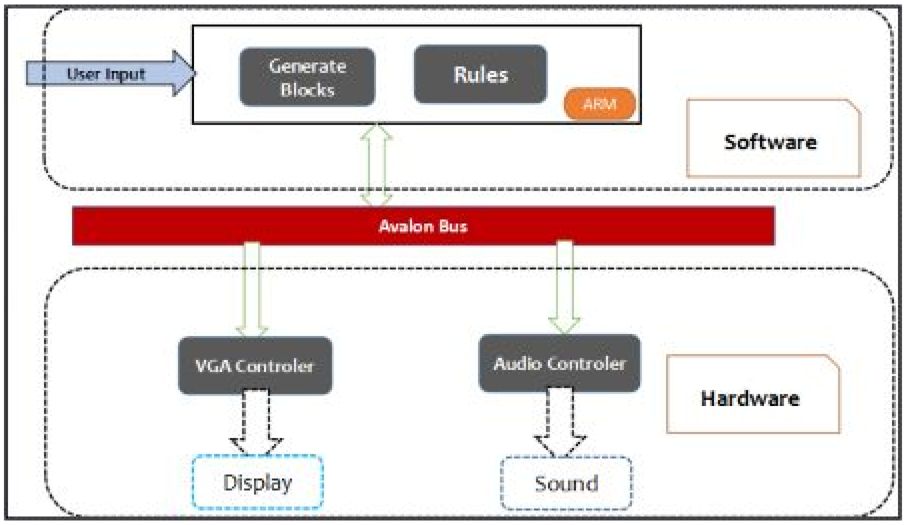
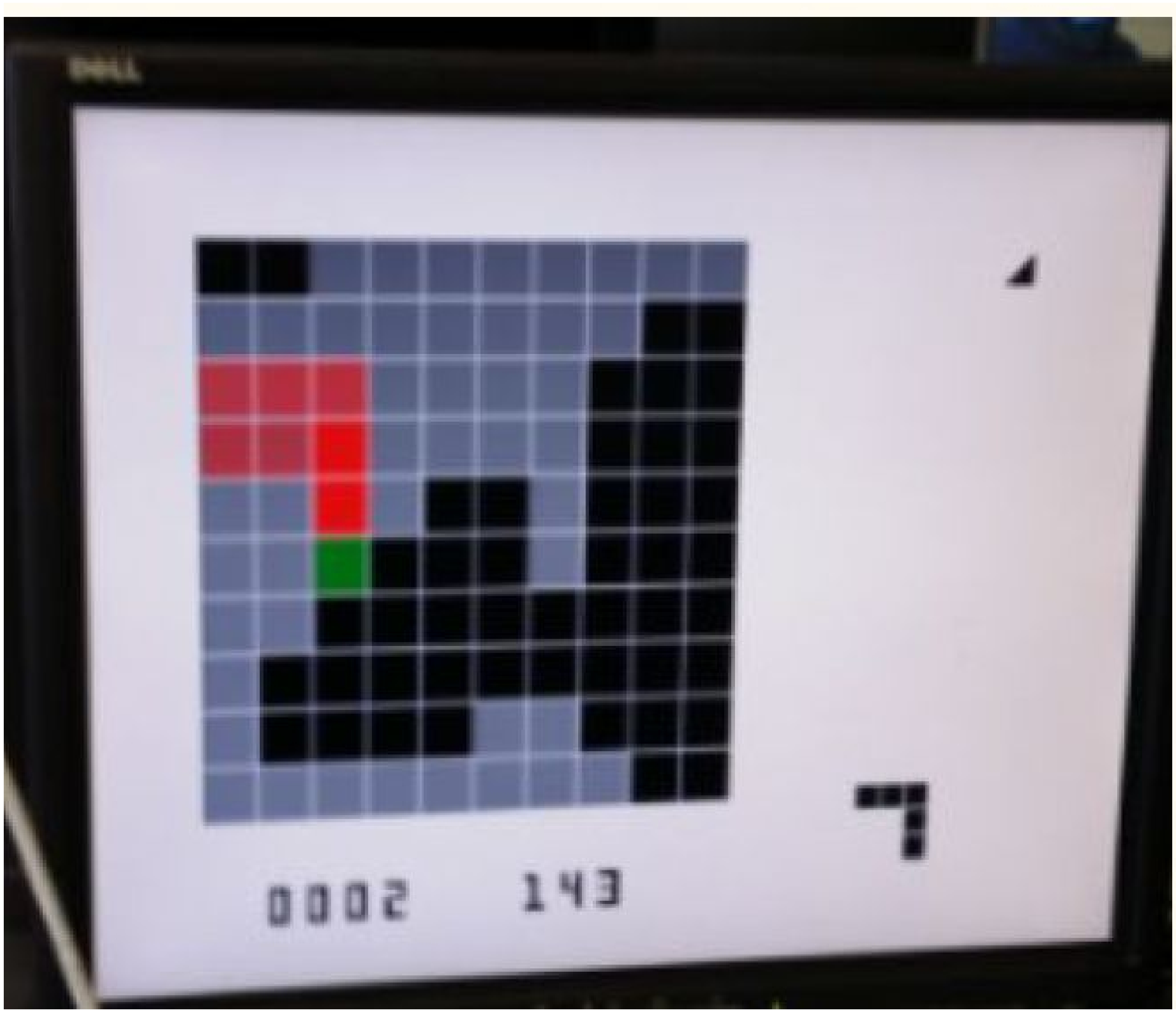


Figure 2

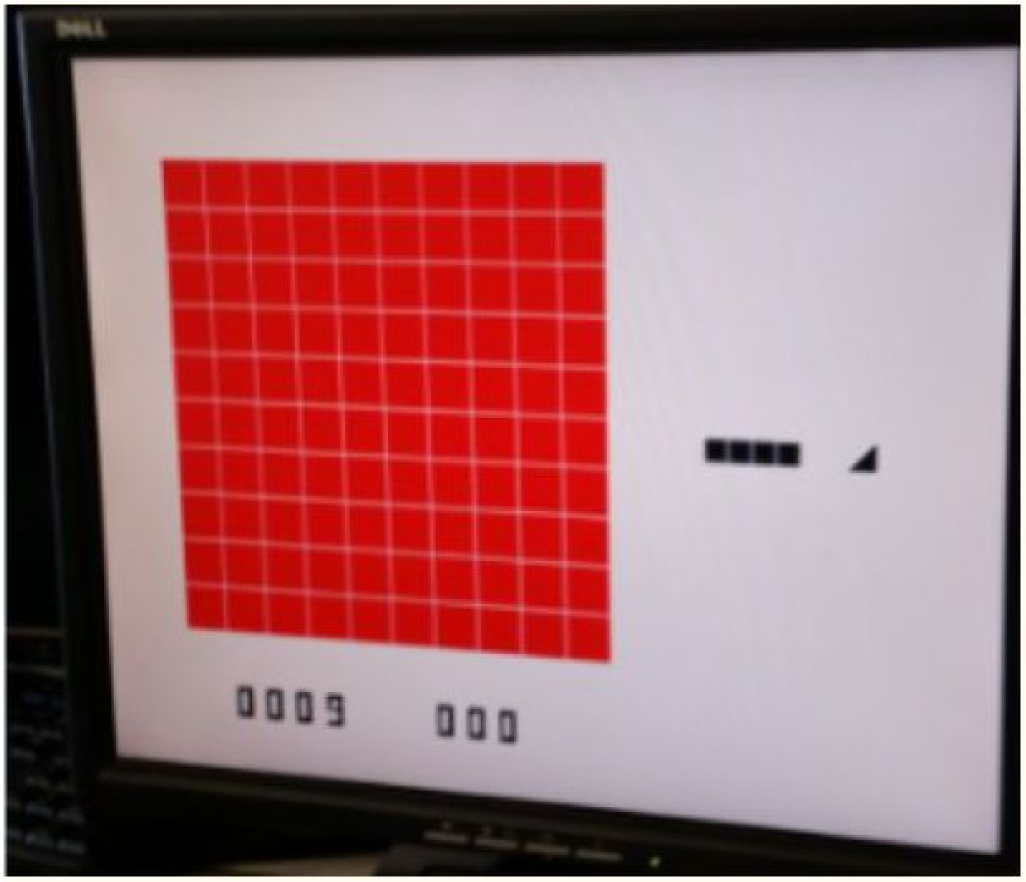
In software, the C language will be used to implement all the controls of the system. It first generates the three blocks in the screen and then keeps getting the user's inputs to control the screen display by using the VGA controller in the hardware part. The VGA controller we use is same as the lab3. The Audio Controller will be used to give sound if the game is over.

User Interface:

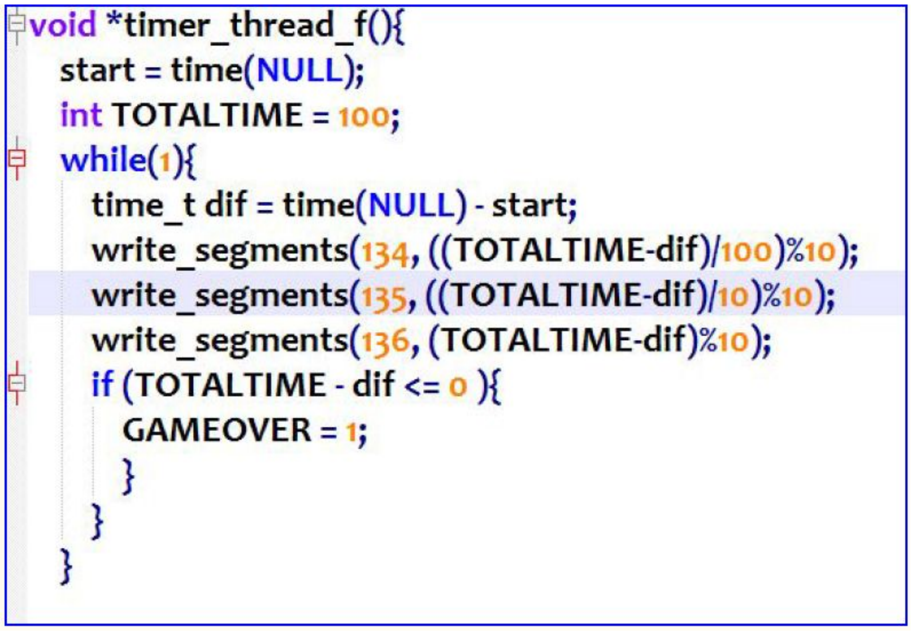


As showed in the graph, the grid is consist of 10*10 blocks. The “red” blocks are the blocks are put in the grid and can’t be moved(except one line formed and the red line is cleared). The “gray” block means that that block has not been put but can’t be put in the position because another block has already been put in the location. The “green” block has not been put and can be put in the location. The three blocks in the right of the grid are used for the user to select. After the user select and put the block into the screen, the block will disappear. If one
line is cleared, the score will be recorded at the bottom of the screen.

End Of Game:



There are two data shown in the bottom of the screen. One in the left(4 bits) is to record the score. If the user successfully clear one vertical line or a horizontal line, there will be one score stored. The three bits data in the right bottom screen is the time left. In the beginning the time is 100 seconds and it will be decreased. After 100 seconds passed, the game will terminated automatically, And the grid becomes all green, then all grey and then all red if time is zero. Also, as a bonus, if one line is cleared, we give ten seconds increment.

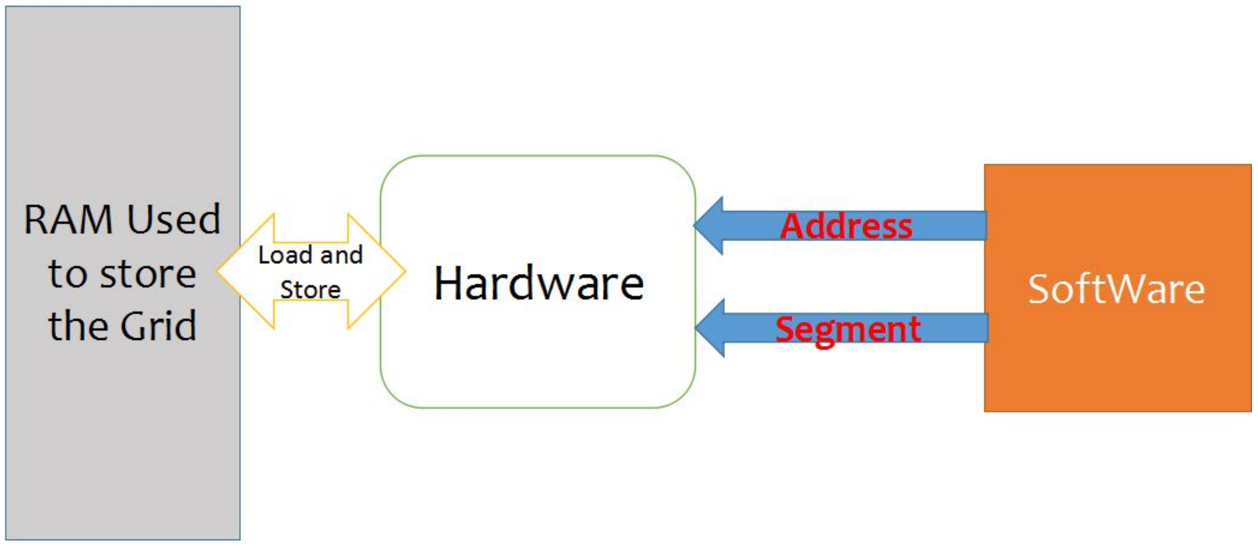


```
void *timer_thread_f(){
    start = time(NULL);
    int TOTALTIME = 100;
    while(1){
        time_t dif = time(NULL) - start;
        write_segments(134, ((TOTALTIME-dif)/100)%10);
        write_segments(135, ((TOTALTIME-dif)/10)%10);
        write_segments(136, (TOTALTIME-dif)%10);
        if (TOTALTIME - dif <= 0 ){
            GAMEOVER = 1;
        }
    }
}
```

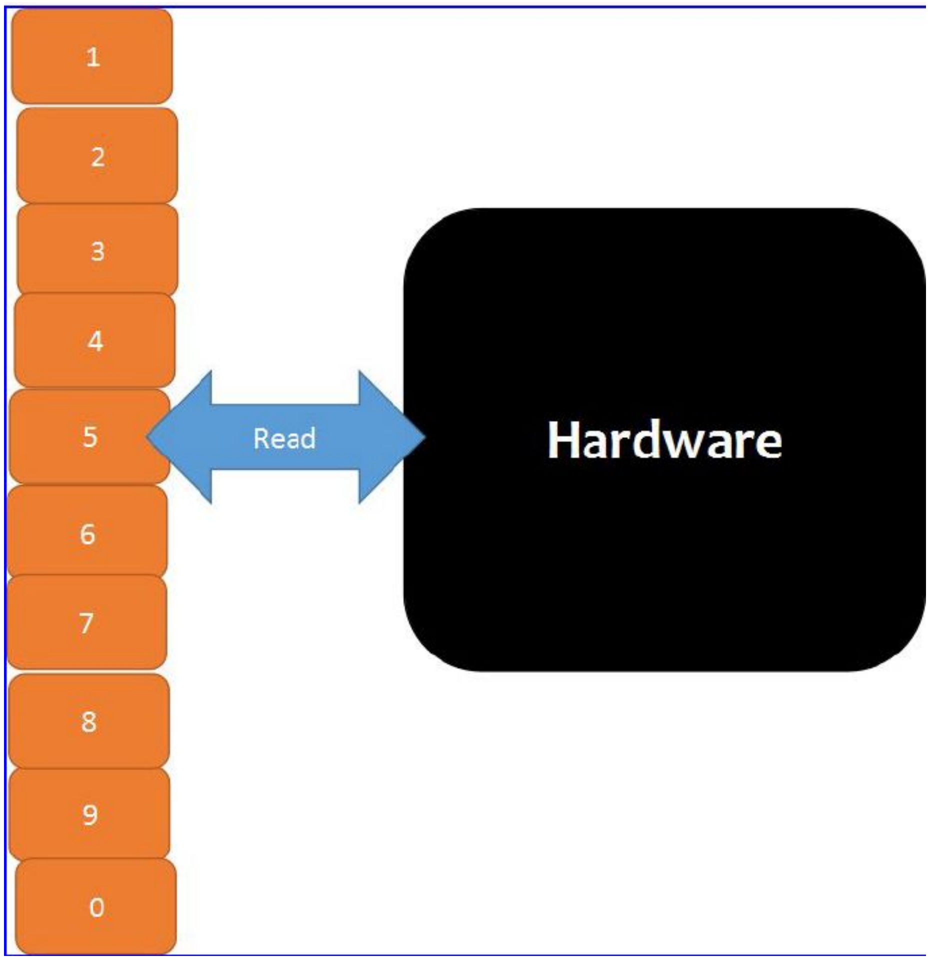
The design of the end is to use the thread method to keep record the time. The "start" is the time the game is started and we keep track the time by calculating the "dif". The time will be separated into three parts and passed into hardware by using three "write_segment" operations.

Communication:

As showed in the graph, the communication between software and hardware is using two registers to pass information of the grid. As you see, the "address" is the grid's index and the "segment" is the data in the grid. If the software gets the user's input, it may change the grid. Then the software send the segment changed in the grid to hardware by using the "address" to give the index of the grid and the "segment" is the new change of that data in the location. The hardware stores all the information of a grid by using a RAM module. It loads the grid if it needs and can change and stores the data in the RAM whenever the software indicates a change.



ROM to Integers



The ten integers are stored in the ROM and can be accessed by the hardware to displayed into the screen. We use "vcount" and "hcount" to indicate the specific location where the integer showed. The size of every 23*32 sprites.

Sprite Controller

The Sprite Controller will be used in display the "welcome" and "termination" screen. The basic connection is shown below:

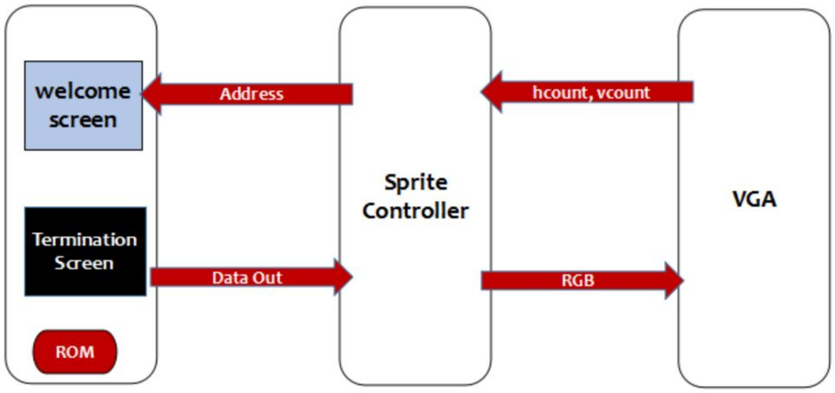


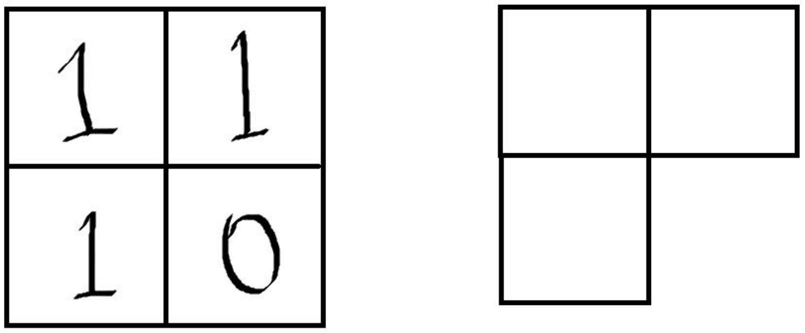
Figure 4

The two screens are stored in the ROM and the Sprite controller will give the address to the memory to get the data pixel out then it forwards to the VGA for display objective.

Game Logics and Software Realization

The game logic is pretty much straight forward and the software runs simulation of the game in the form of matrices, instead of virtually, then pass the matrix to hardware for display. Thus state of all variables are stored in a two dimensional array.

The 14 block shapes are declared as global variables in the form of 2 dimensional arrays and named according to the previously designed index. Formats of the arrays are as indicated below.

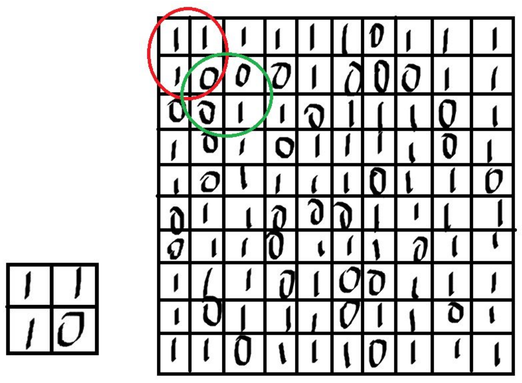


The main game matrix is initialized when the game starts. Software then generates the 3 shapes using a random algorithm to automatically select 3 numbers from 1 to 14. The time counting starts along with the generations. The main matrix stores the 10*10 grid plus another row to store important variables like a bool value for judging whether player is choosing blocks or moving blocks, the index of the 3 shapes generated, current score and the chosen block shape. There is another cache matrix to store the temporary state of the main game matrix when player is moving blocks in the game interface. This is because when the block is moved, the area it covered before the movement needs to return to the previous state. In each movement, software iterate through the relevant area on the main game interface to judge if the state of one block is unoccupied and covered by the current shape, unoccupied and uncovered, occupied but covered by the current shape, occupied but not covered. Once a 'put' command is triggered, the software will apply changes to the main game matrix if the covered area is all unoccupied, otherwise do nothing. When a 'put' command is completed successfully, that is, the chosen shape is used, the corresponding index stored in the main matrix will be changed to 0. If player decide to choose another shape instead of the current one, he or she will have to press 'esc' to return to the 'choosing shapes' state and the main game matrix will remain unchanged.

After 'put' operation is completed, software then checks if there is a row or column that can be cleared, there will be two arrays to store the corresponding row and column index and add the number of clearable rows and columns to the current score stored, then put the new value into main matrix. Each cleared row or column will then add a bonus 10 seconds to the current counting.

To decide if the game is ended or not. There is two ways, one is timed up or there is nowhere to put the blocks. After each 'put' operation, software checks in the main game matrix where the 3 shapes is stored, if not 0, initialize a two dimensional temporary array to store the shape of one of the blocks left, then try to add this array to a sub-matrix from the main game matrix, the sub-matrix will be of the same shape of the temporary array. If any of the position in the temporary array turns to 2 after adding, this sub-matrix could not hold the corresponding shape. After iterating through the whole main game interface, if there is no place that can hold the current shape, software will do this to the next shape left.

For example, to exam if the left shape can be put to the game interface, iterate from the upper left corner, that is matirx[0][0], subtract the sub-matrix from the game interface and do an adding operation, cleared there will be 2s, thus proceed to examine next sub matrix. When encounter the sub-matrix in the green circle, there will be no 2s after adding, thus there is still place to put the corresponding shape.



Either way, at the end of game, the game interface will flash three times and change color, stopping by setting all 100 blocks red then exit the progrom.

Hardware: Vision realization

The vision realization is based on lab 3, so the main file of this part are VGA_LED.vs and VGA_LED_Emulator.vs. VGA_LED.vs is the top module communicate with Avalon MM and VGA_LED_Emulator.vs is the module for the achievement of the display

In VGA_LED.vs, there are mainly two 8-bit inputs that are passed from Avalon MM: *Address* and *Writedata*. Address is used to indicate where the module should pass the data and Writedata is the data that will be passed. According to the our algorithms, if the Address is smaller than 128 (which means Address[7] = 0), the module passes both address and writedata to VGA_LED_Emulator.vs for further process; if the Address is equal or greater than 128 (which means Address[7] = 1), according to its specific value, directly store the data from Writedata to the registers that we build. For more detailedly, we use address 0-99 and their data to represent the color of the blocks in our 10*10 grids and we use 128-138 to represent the tetris select option, pointing arrow position, score, timing and ending status. These will be explained more in the following paragraph.

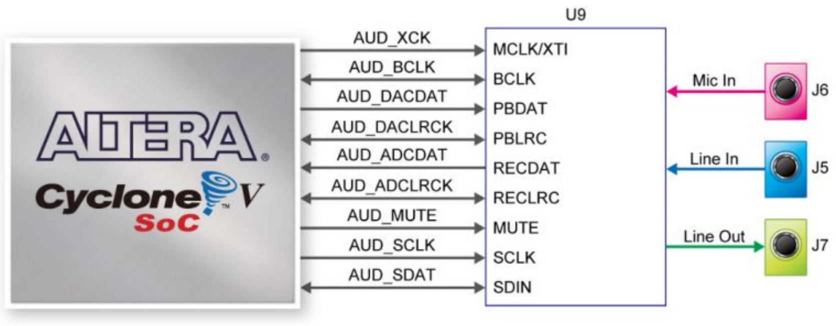
In VGA_LED.Emulator, it use the same module as lab3 which the resolution is 640 * 480. This module will receive the data passing from VGA_LED.vs and calculate the right position for the pixel color. Based on the displayment we can separate the screen to three parts: Main grid. Tetris selection and score as well as timing. For the main grid, it is a 10*10 grid and each block has 32*32=1024 pixels. The color of each block will be based on the value that stored in the RAM called grid_array according to its reading address. Every time when the software passes data to VGA_LED.vs, if it is the value for grid, they will be passed to the RAM according to their writing addresses. Since every time the whole grid data will be refreshed and passed so we do not need to worry the initialization value. The calculation of the reading address is based on the function ***read_address <= (hcount[10:6]-2) +(vcount[9:5]-2)*10,*** Another very important thing is that in order to have the color only in the range of the grid there is a needed of flag to turn on or off, which will be also used in the rest two parts. For the tetris selection part, by defining the range of hcount and vcount, we specified the whole 14 kinds of tetris in the game and according to the data passed from software we display the selected one in the area. A better improvement will be using sprite since it will save more coding lines. For the score and timing area, we used sprite and by storing the RGB data for each pixel in to the ROM we can display them in any position that we want based on the read address calculation. The read address will tell the rom which pixel RGB should be use in the moment and according to the time and score data passing from software we can successfully pass the right one to display.

Using sprite is a easy and good way to achieve displayment. And the transformation from an image to the Pixel RGB value can be achieved by using Matlab.

Audio And Sound

We use XBOX handle to control the game. Instead of using others' driver, we design our own driver.
Because there are no available sources online that can be applied directly, we tested and reverse engineered the corresponding keycode for the 'up' 'down' 'left' 'right', 'A' and 'B' keys and map it to the keyboard driver. However if we use usb-hub to connect both keyboard and xbox controller at the same time, either of the two won't work.

The audio is used for giving sound when we start the game. In the welcome screen, there should be a welcome sound which is positive and make players feel confident. During the game, the sound is changed. Then in the last part, the game over part, the sound should be changed to be a negative sound.



As showed in the figure 5, this is a possible audio device we will use: Analog Devices SSM2603 audio CODEC (Encoder/Decoder).

Code:

Project.c:

```
/* datalink rework,  date: 2016-4-26*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include "usbkeyboard.h"
/*************************************Header                                         Files
Included*********************************************/
#include "vga_led.h"//import struct, 2016-4-26
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <pthread.h>

pthread_t timer_thread;
pthread_t click_thread;
int GAMEOVER = 0;
int timer;
time_t start;
int **matrix;

/*************************************End
Headers**************************************************/
#define MATRIX_ROW 11
#define MATRIX_COL 10
#define B(x) (((x)!=0)?1:0)



/*************************CODE FOR CALLING THE INTERFACE BETWEEN HARDWARE AND
SOFTWARE*******************************/
/*segment: the data you want to write into the hardware
        index: the index of register you want to write into
*/

/*Pass into the index and get the data as the return value
unsigned int read_segments(int index){xbox
        vga_led_arg_t vla;
        vla.digit = index;
        if (ioctl(vga_led_fd, VGA_LED_READ_DIGIT, &vla)) {
    perror("ioctl(VGA_LED_READ_DIGIT) failed");
    return -1;
  }
        return vla.segment;
}
```

```
*/

int vga_led_fd;
int xboxduplicate = 0;
/***************************************************END
CODE************************************************************/

//initialize blocks
int b1[1][5] = {1, 1, 1, 1, 1};
int b2[5][1] = {1, 1, 1, 1, 1};
int b3[1][4] = {1, 1, 1, 1};
int b4[4][1] = {1, 1, 1, 1};
int b5[2][2] = {{1, 1}, {1, 1}};
int b6[3][3] = {{1, 1, 1}, {1, 0, 0}, {1, 0, 0}};
int b7[3][3] = {{1, 1, 1}, {0, 0, 1}, {0, 0, 1}};
int b8[2][1] = {1, 1};
int b9[3][1] = {1, 1, 1};
int b10[1][1] = {1};
int b11[2][2] = {{1, 1}, {1, 0}};
int b12[2][3] = {{1, 1, 0}, {0, 1, 1}};
int b13[1][3] = {1, 1, 1};
int b14[1][2] = {1, 1};

struct libusb_device_handle *keyboard=NULL, *xbox=NULL;
uint8_t endpoint_address;
uint8_t endpoint_address2;

        int ifxbox = 0;
```

```
typedef struct block{
        int height;
        int width;
        //int blockArray[];
} blockStuct;


blockStuct block1;
blockStuct block2;
blockStuct block3;
blockStuct block4;
blockStuct block5;
blockStuct block6;
blockStuct block7;
blockStuct block8;
blockStuct block9;
blockStuct block10;
blockStuct block11;
blockStuct block12;
blockStuct block13;
blockStuct block14;

int vga_led_fd;
//AVALON-MM INFO:
//                  - 16bit data link

//new struct //included in vga_led.h
/*
typedef struct {
```

```
            u16 data;
} vga_led_arg_t;
*/


//8bit address write
int A=0,B=0,Up=0,Down=0,Left=0,Right=0;
int A2=0,B2=0,Up2=0,Down2=0, Left2=0,Right2=0;
/* Write the contents of the array to the display */
void write_segments(unsigned int index, unsigned int segment)
{
  vga_led_arg_t vla;
  int i;
  if(GAMEOVER){
int j;
for(j=0; j<4;j++){
            for(i= 0;i<100;i++)
{vla.address = i;


            vla.data = j;
 ioctl(vga_led_fd, VGA_LED_WRITE_DIGIT, &vla);


}
sleep(1);
}
exit(-1);}


else{
  for (i = 0 ; i < 256 ; i++) {
            if(index==101) index = 128;
            if(index==102) index = 129;
            if(index==103) index = 130;
            if(index==107) index = 133;
            if(index==104) index = 138;
            vla.address = index;
     vla.data = segment;
     if (ioctl(vga_led_fd, VGA_LED_WRITE_DIGIT, &vla)) {
       perror("ioctl(VGA_LED_WRITE_DIGIT) failed");
       return;
     }
   }
 }
}


void sound(int type){
            write_segments(0xFF,0);
            write_segments(0xFF,type);
}
void passToHardware(int** matrix){
            perror("P2H enter");
            int row, col, index;
            int segment = 0;
            for (row = 0; row < 11; row++){
                        for(col= 0; col < 10; col++){
                                    index = (row*10) + col ;
                                    segment = matrix[row][col];
                                    write_segments(index, segment);
```

```
                }
        }
}



int generate()
{
        int i;
        i = (random() % (14-1+1)) + 1;
        return i;
}

void clearLine(int** matrix)
{
        int row;
        int col;
        int row_to_clean[10] = {11, 11, 11, 11, 11, 11, 11, 11, 11, 11};
        int col_to_clean[10] = {11, 11, 11, 11, 11, 11, 11, 11, 11, 11};
        int i, j;
        int t, k;
        int p, q, r;
        int score;
        score = 0;
        for(row = 0; row < 10; row ++){ //get rows that can be cleaned
                //for(i = 0; i<10; i++){
                        if (matrix[row][0] == 1 && matrix[row][1] == 1 && matrix[row][2] == 1 &&
matrix[row][3] == 1 && matrix[row][4] == 1 && matrix[row][5] == 1 && matrix[row][6] == 1 &&
matrix[row][7] == 1 && matrix[row][8] == 1 && matrix[row][9] == 1){
                                printf("clean line %d \n", row);
```

```
					row_to_clean[row] = row;
				}
				//else{
				//	row_to_clean[i] = 11;
				//}
			//}
		}
		for(col = 0; col < 10; col++){ //get columns that can be cleaned
			//for(j = 0; j <10; j++){
				if (matrix[0][col] == 1 && matrix[1][col] == 1 && matrix[2][col] == 1 &&
matrix[3][col] == 1 && matrix[4][col] == 1 && matrix[5][col] == 1 && matrix[6][col] == 1 &&
matrix[7][col] == 1 && matrix[8][col] == 1 && matrix[9][col] == 1){
					printf("clean line %d \n", col);
					col_to_clean[col] = col;
				}
				//else{
				//	col_to_clean[j] = 11;
				//}
			//}
		}

		for(p = 0; p <10; p ++ ){
			int temp = row_to_clean[p];
			printf("row=%d\n", temp);
			if (temp != 11){
				score = score + 1;
				for(q = 0; q < 10; q++){
					matrix[temp][q] = 0;
				}
```

```
                printf("cleared row %d \n", temp);
            }
        }

        for(k = 0; k <10; k ++ ){
            int temp2 = col_to_clean[k];
            printf("col=%d\n", temp2);
            if (temp2 != 11){
                score = score + 1;
                for(r = 0; r < 10; r++){
                    matrix[r][temp2] = 0;
                }
                printf("cleared line %d \n", temp2);
            }
        }



        matrix[10][5] = matrix[10][5] + score;
        int send_score = matrix[10][5];
        int thousand, hundred, tens, ones;
        start += 10*score;
        /*********************************************TO
    DO*********************************************/
        fprintf(stderr, "score= %d\n", send_score);
        thousand = send_score/1000;
        hundred = (send_score - 1000*thousand)/100;
        tens = (send_score - 1000*thousand - 100*hundred)/10;
        ones = (send_score - 1000*thousand - 100*hundred - 10*tens);
```

```
	int send_seg_1,  send_seg_2;

	send_seg_1 = ones + 16*tens;
	send_seg_2 = hundred + 16*thousand;

	write_segments(131, send_seg_2);
	write_segments(132, send_seg_1);

}

int checkIfGameEnd(int* b, int** matrix){
	int i;
	int end = 1;
	int m, n, temp_row, temp_col;
	int p, q;
	int temp = 0;
	int temp1[1][5];
	int temp2[5][1];
	int temp3[1][4];
	int temp4[4][1];
	int temp5[2][2];
	int temp6[3][3];
	int temp7[3][3];
	int temp8[2][1];
	int temp9[3][1];
	int temp10[1][1];
	int temp11[2][2];
	int temp12[2][3];
	int temp13[1][3];
```

```
        int temp14[1][2];
        for (i = 0; i<3; i++){
                temp = b[i];
                switch(temp){
                        case 0:
                                end = 0;
                                break;
                        case 1:
                                for (m = 0; m < 10; m++){
                                        for (n = 0; n < 10; n++){
                                                int temp_block = matrix[m][n];
                                                /*start to  extract blocks from matrix if there is an
empty bloack and start from this block there are enough blocks to match the shape,
                                                regardless if the blocks are empty or not*/
                                                if (temp_block == 0 && 10 - m > 0 && 10 - n < 6){
                                                        for(temp_row = 0; temp_row < 1; temp_row
++){
                                                                for (temp_col = 0; temp_col < 5;
temp_col ++){
                                                                        /*extract the shape of one of
the left shapes and compare*/
                                                                        temp1[temp_row][temp_col] =
matrix[m + temp_row][n + temp_col];
                                                                        for (p = 0; p < 1; p ++){
                                                                                for (q = 0; q < 5; q ++){
                                                                                        if(  b1[p][q]  ==
temp1[p][q]){
                                                                                                break;
                                                                                        }else{
```

```
                                                /*  the
game is not end if there is one area that can fit in any of the shapes*/
                                                end   =
0;
                                                break;
                                            }
                                            break;
                                        }
                                        break;
                                    }
                                    break;
                                }
                                break;
                            }
                            break;
                        }else{
                            break;
                        }
                        break;
                    }
                    break;
                }
                break;
            case 2:
                for (m = 0; m < 10; m++){
                    for (n = 0; n < 10; n++){
                        int temp_block = matrix[m][n];
                        if (temp_block == 0 && 10 - m > 4 && 10 - n > 0){
                            for(temp_row = 0; temp_row < 5; temp_row
++){
                                for  (temp_col  =  0;  temp_col  <  1;
temp_col ++){
                                    temp2[temp_row][temp_col] =
matrix[m + temp_row][n + temp_col];
                                    for (p = 0; p < 5; p ++){
                                        for (q = 0; q < 1; q ++){
                                            if(  b2[p][q]  ==
temp2[p][q]){
                                                break;
                                            }else{

                                                end   =
0;
                                            }
                                            break;
                                        }
                                        break;
                                    }
                                    break;
                                }
                                break;
                            }
                        }else{
                            break;
                        }
                        break;
                    }
                    break;
```

```
                }
                break;
            case 3:

                for (m = 0; m < 10; m++){
                    for (n = 0; n < 10; n++){
                        int temp_block = matrix[m][n];
                        if (temp_block == 0 && 10 - m > 0 && 10 - n > 3){
                            for(temp_row = 0; temp_row < 1; temp_row
++){
                                for (temp_col = 0; temp_col < 4;
temp_col ++){
                                    temp3[temp_row][temp_col] =
matrix[m + temp_row][n + temp_col];
                                    for (p = 0; p < 1; p ++){
                                        for (q = 0; q < 4; q ++){
                                            if( b3[p][q] ==
temp3[p][q]){

                                                break;
                                            }else{
                                                end =
0;

                                            }
                                            break;
                                        }
                                        break;
                                    }
                                    break;
                                }
                                break;
                            }
                            break;
                        }else{
                            break;
                        }
                        break;
                    }
                    break;
                }
                break;
            case 4:

                for (m = 0; m < 10; m++){
                    for (n = 0; n < 10; n++){
                        int temp_block = matrix[m][n];
                        if (temp_block == 0 && 10 - m > 3 && 10 - n > 0){
                            for(temp_row = 0; temp_row < 4; temp_row
++){
                                for (temp_col = 0; temp_col < 1;
temp_col ++){
                                    temp4[temp_row][temp_col] =
matrix[m + temp_row][n + temp_col];
                                    for (p = 0; p < 4; p ++){
                                        for (q = 0; q < 1; q ++){
                                            if( b4[p][q] ==
temp4[p][q]){
                                                break;
                                            }else{
                                                end    =
0;
                                            }
                                            break;
                                        }
                                        break;
                                    }
                                    break;
                                }
                                break;
                            }
                            break;
                        }else{
                            break;
                        }
                        break;
                    }
                    break;
                }
                break;
            case 5:

                for (m = 0; m < 10; m++){
                    for (n = 0; n < 10; n++){
                        int temp_block = matrix[m][n];
                        if (temp_block == 0 && 10 - m > 1 && 10 - n > 1){
                            for(temp_row = 0; temp_row < 2; temp_row
++){
                                for (temp_col = 0; temp_col < 2;
temp_col ++){
                                                            temp5[temp_row][temp_col] =
matrix[m + temp_row][n + temp_col];
                                                            for (p = 0; p < 2; p ++){
                                                                  for (q = 0; q < 2; q ++){
                                                                        if(  b5[p][q]   ==
temp5[p][q]){

                                                                              break;
                                                                        }else{
                                                                              end    =
0;

                                                                        }
                                                                        break;
                                                                  }
                                                                  break;
                                                            }
                                                            break;
                                                      }
                                                      break;
                                                }
                                                break;
                                          }else{
                                                break;
                                          }
                                          break;
                                    }
                                    break;
                              }
                              break;
```

```
                    case 6:

                        for (m = 0; m < 10; m++){
                            for (n = 0; n < 10; n++){
                                int temp_block = matrix[m][n];
                                if (temp_block == 0 && 10 - m > 2 && 10 - n > 2){
                                    for(temp_row = 0; temp_row < 3; temp_row
++){
                                        for (temp_col = 0; temp_col < 3;
temp_col ++){
                                            temp6[temp_row][temp_col] =
matrix[m + temp_row][n + temp_col];
                                            for (p = 0; p < 3; p ++){
                                                for (q = 0; q < 3; q ++){
                                                    if( b6[p][q] ==
temp6[p][q]){

                                                        break;
                                                    }else{
                                                        end =
0;

                                                    }
                                                    break;
                                                }
                                                break;
                                            }
                                            break;
                                        }
                                        break;
                                    }
```

```
                                                break;
                                        }else{
                                                break;
                                        }
                                        break;
                                }
                                break;
                        }
                        break;
                case 7:

                        for (m = 0; m < 10; m++){
                                for (n = 0; n < 10; n++){
                                        int temp_block = matrix[m][n];
                                        if (temp_block == 0 && 10 - m > 2 && 10 - n > 2){
                                                for(temp_row = 0; temp_row < 3; temp_row
++){
                                                        for (temp_col = 0; temp_col < 3;
temp_col ++){
                                                                temp7[temp_row][temp_col] =
matrix[m + temp_row][n + temp_col];
                                                                for (p = 0; p < 3; p ++){
                                                                        for (q = 0; q < 3; q ++){
                                                                                if( b7[p][q] ==
temp7[p][q]){
                                                                                        break;
                                                                                }else{
                                                                                        end =
0;
```

```
                                    }
                                    break;
                                }
                                break;
                            }
                            break;
                        }
                        break;
                    }
                    break;
                }else{
                    break;
                }
                break;
            }
            break;
        }
        break;
    case 8:

        for (m = 0; m < 10; m++){
            for (n = 0; n < 10; n++){
                int temp_block = matrix[m][n];
                if (temp_block == 0 && 10 - m > 1 && 10 - n > 0){
                    for(temp_row = 0; temp_row < 2; temp_row
++){
                        for (temp_col = 0; temp_col < 1;
temp_col ++){
                            temp8[temp_row][temp_col] =
matrix[m + temp_row][n + temp_col];
                            for (p = 0; p < 2; p ++){
                                for (q = 0; q < 1; q ++){
                                    if(  b8[p][q]  ==
temp8[p][q]){

                                                break;
                                    }else{
                                                end   =
0;

                                    }
                                    break;
                                }
                                break;
                            }
                            break;
                        }
                        break;
                    }
                    break;
                }else{
                    break;
                }
                break;
            }
            break;
        }
        break;
    case 9:
```

```
                                        for (m = 0; m < 10; m++){
                                                for (n = 0; n < 10; n++){
                                                        int temp_block = matrix[m][n];
                                                        if (temp_block == 0 && 10 - m > 2 && 10 - n > 0){
                                                                for(temp_row = 0; temp_row < 3; temp_row
++){
                                                                        for (temp_col = 0; temp_col < 1;
temp_col ++){
                                                                                temp9[temp_row][temp_col] =
matrix[m + temp_row][n + temp_col];
                                                                                for (p = 0; p < 3; p ++){
                                                                                        for (q = 0; q < 1; q ++){
                                                                                                if( b9[p][q] ==
temp9[p][q]){

                                                                                                        break;
                                                                                                }else{
                                                                                                        end =
0;

                                                                                                }
                                                                                                break;
                                                                                        }
                                                                                        break;
                                                                                }
                                                                                break;
                                                                        }
                                                                        break;
                                                                }
                                                                break;
                                                        }else{
```

```
                                    break;
                                }
                                break;
                            }
                            break;
                        }
                        break;
                    case 10:

                        for (m = 0; m < 10; m++){
                            for (n = 0; n < 10; n++){
                                int temp_block = matrix[m][n];
                                if (temp_block == 0 && 10 - m > 0 && 10 - n > 0){
                                    for(temp_row = 0; temp_row < 1; temp_row
++){
                                        for (temp_col = 0; temp_col < 1;
temp_col ++){
                                            temp10[temp_row][temp_col]
= matrix[m + temp_row][n + temp_col];
                                            for (p = 0; p < 1; p ++){
                                                for (q = 0; q < 1; q ++){
                                                    if( b10[p][q] ==
temp10[p][q]){

                                                        break;
                                                    }else{
                                                        end    =
0;

                                                    }
                                                    break;
                                                }
                                                break;
                                            }
                                            break;
                                        }
                                        break;
                                    }
                                    break;
                                }else{
                                    break;
                                }
                                break;
                            }
                            break;
                        }
                        break;
                    case 11:

                        for (m = 0; m < 10; m++){
                            for (n = 0; n < 10; n++){
                                int temp_block = matrix[m][n];
                                if (temp_block == 0 && 10 - m > 2 && 10 - n > 2){
                                    for(temp_row = 0; temp_row < 3; temp_row
++){
                                        for (temp_col = 0; temp_col < 3;
temp_col ++){
                                            temp11[temp_row][temp_col]
= matrix[m + temp_row][n + temp_col];
                                            for (p = 0; p < 3; p ++){
```

```
                                        for (q = 0; q < 3; q ++){
                                            if(  b6[p][q]   ==
temp11[p][q]){

                                                break;
                                            }else{
                                                end    =
0;

                                            }
                                            break;
                                        }
                                        break;
                                    }
                                    break;
                                }
                                break;
                            }
                            break;
                        }else{
                            break;
                        }
                        break;
                    }
                    break;
                }
                break;
            case 12:

                for (m = 0; m < 10; m++){
                    for (n = 0; n < 10; n++){
                        int temp_block = matrix[m][n];
                        if (temp_block == 0 && 10 - m > 1 && 10 - n > 2){
                            for(temp_row = 0; temp_row < 2; temp_row
++){
                                for (temp_col = 0; temp_col < 3;
temp_col ++){
                                    temp12[temp_row][temp_col]
= matrix[m + temp_row][n + temp_col];
                                    for (p = 0; p < 2; p ++){
                                        for (q = 0; q < 3; q ++){
                                            if( b12[p][q] ==
temp12[p][q]){
                                                break;
                                            }else{
                                                end =
0;
                                            }
                                            break;
                                        }
                                        break;
                                    }
                                    break;
                                }
                                break;
                            }
                            break;
                        }else{
                            break;
                        }
                        break;
                    }
                    break;
                }
                break;
            case 13:

                for (m = 0; m < 10; m++){
                    for (n = 0; n < 10; n++){
                        int temp_block = matrix[m][n];
                        if (temp_block == 0 && 10 - m > 0 && 10 - n > 2){
                            for(temp_row = 0; temp_row < 1; temp_row
++){
                                for (temp_col = 0; temp_col < 3;
temp_col ++){
                                    temp13[temp_row][temp_col]
= matrix[m + temp_row][n + temp_col];
                                    for (p = 0; p < 1; p ++){
                                        for (q = 0; q < 3; q ++){
                                            if( b13[p][q] ==
temp13[p][q]){

                                                break;
                                            }else{
                                                end =
0;

                                            }
                                            break;
                                        }
                                        break;
```

```
                                                }
                                                break;
                                            }
                                            break;
                                        }
                                        break;
                                    }else{
                                        break;
                                    }
                                    break;
                                }
                                break;
                            }
                            break;
                        case 14:

                            for (m = 0; m < 10; m++){
                                for (n = 0; n < 10; n++){
                                    int temp_block = matrix[m][n];
                                    if (temp_block == 0 && 10 - m > 0 && 10 - n > 1){
                                        for(temp_row = 0; temp_row < 1; temp_row
++){
                                            for (temp_col = 0; temp_col < 2;
temp_col ++){
                                                temp14[temp_row][temp_col]
= matrix[m + temp_row][n + temp_col];
                                                for (p = 0; p < 1; p ++){
                                                    for (q = 0; q < 2; q ++){
                                                        if( b14[p][q] ==
temp14[p][q]){
                                                break;
                                                        }else{
                                                end    =
0;
                                                        }
                                                        break;
                                                    }
                                                    break;
                                                }
                                                break;
                                            }
                                            break;
                                        }
                                        break;
                                    }else{
                                        break;
                                    }
                                    break;
                                }
                                break;
                            }
                            break;
                        default:
                            break;
                    }
                    break;
                }
```

```
        return end;
}

//CHECK INPUT HERE
int putBlock(int j, int* b, int** matrix, int** cache){
        int right_bound, left_bound, up_bound, low_bound, xsize, ysize;
        int temp = b[j];
        int q = j;
        int inChoose;
        struct usb_keyboard_packet packet;
        int transferred;
        char keystate[12];
        fprintf(stderr, "putting\n");
        fprintf(stderr,"%d\n", q);
        fprintf(stderr, "%d\n", temp);

        switch(temp){
                case 1:
                        right_bound = 5;
                        low_bound = 9;
                        xsize = 5;
                        ysize = 1;
                        break;
                case 2:
                        right_bound = 9;
                        low_bound = 5;
                        xsize = 1;
                        ysize = 5;
                        break;
```

```
case 3:
        right_bound = 6;
        low_bound = 9;
        xsize = 4;
        ysize = 1;
        break;
case 4:
        right_bound = 9;
        low_bound = 6;
        xsize = 1;
        ysize = 4;
        break;
case 5:
        right_bound = 8;
        low_bound = 8;
        xsize = 2;
        ysize = 2;
        break;
case 6:
        right_bound = 7;
        low_bound = 7;
        xsize = 3;
        ysize = 3;
        break;
case 7:
        right_bound = 7;
        low_bound = 7;
        xsize = 3;
        ysize = 3;
        break;
case 8:
        right_bound = 9;
        low_bound = 8;
        xsize = 1;
        ysize = 2;
        break;
case 9:
        right_bound = 9;
        low_bound = 7;
        xsize = 1;
        ysize = 3;
        break;
case 10:
        right_bound = 9;
        low_bound = 9;
        xsize = 1;
        ysize = 1;
        break;
case 11:
        right_bound = 8;
        low_bound = 8;
        xsize = 2;
        ysize = 2;
        break;
case 12:
        right_bound = 7;
        low_bound = 8;
        xsize = 3;
```

```
                    ysize = 2;
                    break;
                case 13:
                    right_bound = 7;
                    low_bound = 9;
                    xsize = 3;
                    ysize = 1;
                    break;
                case 14:
                    right_bound = 8;
                    low_bound = 9;
                    xsize = 2;
                    ysize = 1;
                    break;
                default:
                    break;
            }
            int loop_flag = 1;


            fprintf(stderr, "moving\n");
            int x = 0;
            int y = 0;
            moveBlock(temp, xsize, ysize, x, y, matrix, cache, 0);
            while(loop_flag == 1){
        uint8_t input_report[20];
        if(!ifxbox)
            libusb_interrupt_transfer(keyboard, endpoint_address,
                              (unsigned char *) &packet, sizeof(packet),
                              &transferred, 0);
```

```
else{
        //fprintf(stderr, "ifxbox: %d\n",ifxbox);
        A2=A; B2=B; Up2=Up; Down2=Down;Left2=Left; Right2=Right;
libusb_control_transfer(xbox,
LIBUSB_ENDPOINT_IN|LIBUSB_REQUEST_TYPE_CLASS|LIBUSB_RECIPIENT_INTERFACE,
                0x01, (0x01<<8)|0x00, 0, input_report, 20, 1000);
        A=input_report[3] == 16;
        B=input_report[3] == 32;
        Up=input_report[2] == 1;
        Down=input_report[2] == 2;
        Left=input_report[2] == 4;
        Right=input_report[2] == 8;
        //fprintf(stderr, "A %d, B %d\n Up %d, Down %d, Left %d, Right %d\n", A, B, Up, Down,
Left, Right);

}

int
xboxvalid=((A!=A2)||(B!=B2)||(Up!=Up2)||(Down!=Down2)||(Left!=Left2)||(Right!=Right2))&&(B2=
=0)&&(A2==0)&&(Up2==0)&&(Down2==0)&&(Left2==0)&&(Right2==0);
        if(xboxvalid) fprintf(stderr, "valid");
          if (transferred == sizeof(packet) || (ifxbox && xboxvalid)) {

                //pthread_create(&click_thread, NULL, click_thread_f, NULL);
                //sprintf(keystate, "%02x %02x %02x", packet.modifiers, packet.keycode[0],
packet.keycode[1]);
          if (packet.keycode[0] == 0x29 || B){
                sound(2);
```

```
                    matrix[10][0] = 1;
                    fprintf(stderr, ",........................................h");
                    return 0;
                    //break;
                }else if(packet.keycode[0] == 0x4F || Right){
                    //current position
                            sound(1);
                            fprintf(stderr, "right\n");
                            fprintf(stderr, "%d\n", x);
                            fprintf(stderr, "%d\n", y);

                    if (x < right_bound){
                            x = x + 1;
                            y = y;
                            moveBlock(temp, xsize, ysize, x, y, matrix, cache, 0);
                    }else{
                            x = x;
                            y = y;
                    }
                }else if(packet.keycode[0] == 0x50 || Left ){
                    //current position
sound(1);
                            fprintf(stderr, "left\n");
                            fprintf(stderr, "%d\n", x);
                            fprintf(stderr, "%d\n", y);

                    if (x > 0){
                            x = x - 1;
                            y = y;
```

```
                        moveBlock(temp, xsize, ysize, x, y, matrix, cache, 0);
                }else{
                        x = x;
                        y = y;
                }
        }else if(packet.keycode[0] == 0x51 ||Down){
                //current position
sound(1);
                        fprintf(stderr, "down\n");
                        fprintf(stderr, "%d\n", x);
                        fprintf(stderr, "%d\n", y);

                if (y < low_bound){
                        x = x;
                        y = y + 1;
                        moveBlock(temp, xsize, ysize, x, y, matrix, cache, 0);
                }else{
                        x = x;
                        y = y;
                }
        }else if(packet.keycode[0] == 0x52 ||Up){
                //current position
sound(1);
                        fprintf(stderr, "up\n");
                        fprintf(stderr, "%d\n", x);
                        fprintf(stderr, "%d\n", y);

                if (y == 0){
                        x = x;
                        y = y;
                }else{
                        x = x;
                        y = y - 1;
                        moveBlock(temp, xsize, ysize, x, y, matrix, cache, 0);
                }

            }else if(packet.keycode[0] == 0x28 || A){ // Enter
                check = 0;
                check = moveBlock(temp, xsize, ysize, x, y, matrix, cache, 1); // tell moveBlock to
apply change to cache matrix

                if(check == 1){
                        inChoose = 1;
                        //moveBlock(temp, xsize, ysize, x, y, matrix, cache, 1); // if can put, apply
change to matrix
                        loop_flag = 0;
sound(2);
                        break;
                }else if(check == 0){
sound(1);
                        inChoose = 0;// if cant put, keep in putBlock loop
                }
            }
            }
        }
        matrix[10][0] = inChoose;
        return 1;
}
```

```
int checkIfCanPut(int** cache){
        int i, j;
        int check;
        check = 1;
        int temp3;
        temp3 = 0;
        for(i = 0; i<10; i++){
                for(j = 0; j < 10; j++){
                        temp3 = cache[i][j];
                        if (temp3 == 3){
                                check = 0;
                                break;
                        }
                }
        }
        return check;
}


int moveBlock(int block, int xsize, int ysize, int x, int y, int** matrix, int** cache, int boolPut){
        int i, j;
        int m, n;
        int** temp; //temp array to store shape of chosen block
        //int** cache;
        int c, cache_row, cache_col;
        fprintf(stderr, "here2\n");
        int check;
        //initiliaze cache matrix
        cache = malloc(11 * sizeof *cache);
        for (c=0; c< 11; c++)
        {
                cache[c] = malloc(10 * sizeof(int));
        }

        fprintf(stderr, "cache initiliazing\n");

    for (cache_row = 0; cache_row < 11; cache_row++){
                        for (cache_col = 0; cache_col < 10; cache_col++){
                                cache[cache_row][cache_col] = matrix[cache_row][cache_col];
                        }
        }
        fprintf(stderr, "cache initialized from matrix\n");

        temp = malloc(ysize * sizeof *temp);
        for (i=0; i< ysize; i++)
        {
                temp[i] = malloc(xsize * sizeof(int));
        }

        fprintf(stderr, "temp array initiliazing\n");

        for (m = 0; m < ysize; m++){
                for (n = 0; n < xsize; n++){
                        //printf("x == %d", b5[0][1]);
                        temp[m][n] = 0;
        //array[i][j] = 1;
    }
}
fprintf(stderr, "temp array initiliazed \n");


int p, q;
switch(block){
    case 1:
        for (p = 0; p < ysize; p++){
            for (q = 0; q < xsize; q++){
                temp[p][q] = b1[p][q];
            }
        }
        break;
    case 2:
        for (p = 0; p < ysize; p++){
            for (q = 0; q < xsize; q++){
                temp[p][q] = b2[p][q];
            }
        }
        break;
    case 3:
        for (p = 0; p < ysize; p++){
            for (q = 0; q < xsize; q++){
                temp[p][q] = b3[p][q];
            }
        }
        break;
    case 4:
        for (p = 0; p < ysize; p++){
                for (q = 0; q < xsize; q++){
                        temp[p][q] = b4[p][q];
                }
        }
        break;
case 5:
        for (p = 0; p < ysize; p++){
                for (q = 0; q < xsize; q++){
                        temp[p][q] = b5[p][q];
                }
        }
        break;
case 6:
        for (p = 0; p < ysize; p++){
                for (q = 0; q < xsize; q++){
                        temp[p][q] = b6[p][q];
                }
        }
        break;
case 7:
        for (p = 0; p < ysize; p++){
                for (q = 0; q < xsize; q++){
                        temp[p][q] = b7[p][q];
                }
        }
        break;
case 8:
        for (p = 0; p < ysize; p++){
                for (q = 0; q < xsize; q++){
                        temp[p][q] = b8[p][q];
                }
        }
        break;
case 9:
        for (p = 0; p < ysize; p++){
                for (q = 0; q < xsize; q++){
                        temp[p][q] = b9[p][q];
                }
        }
        break;
case 10:
        for (p = 0; p < ysize; p++){
                for (q = 0; q < xsize; q++){
                        temp[p][q] = b10[p][q];
                }
        }
        break;
case 11:
        for (p = 0; p < ysize; p++){
                for (q = 0; q < xsize; q++){
                        temp[p][q] = b11[p][q];
                }
        }
        break;
case 12:
        for (p = 0; p < ysize; p++){
                for (q = 0; q < xsize; q++){
```

```
                temp[p][q] = b12[p][q];
            }
        }
        break;
    case 13:
        for (p = 0; p < ysize; p++){
            for (q = 0; q < xsize; q++){
                temp[p][q] = b13[p][q];
            }
        }
        break;
    case 14:
        for (p = 0; p < ysize; p++){
            for (q = 0; q < xsize; q++){
                temp[p][q] = b14[p][q];
            }
        }
        break;
    default:
        //printf(" y = %d\n", 0);
        break;
}


if (boolPut == 0){
    int temp_row, temp_col;
    int check = 0;
    for (temp_row = 0; temp_row < ysize; temp_row ++){
        for (temp_col = 0; temp_col < xsize; temp_col++){
```

```
                    cache[0 + y + temp_row][0 + x + temp_col] =
temp[temp_row][temp_col] + cache[0 + y + temp_row][0 + x + temp_col];
                        int judge;
                        judge = cache[0 + y + temp_row][0 + x + temp_col];
                        int block_part = temp[temp_row][temp_col];
                        if (judge == 2 && block_part == 1){
                            cache[0 + y + temp_row][0 + x + temp_col] = 3;
                        }else if (judge == 1 && block_part == 1){
                            cache[0 + y + temp_row][0 + x + temp_col] = 2;
                        }
                }
            }
            printf("checking matrix start\n");
            int m_x, m_y;
                    int test_m;
                    test_m = 0;
                    for (m_x = 0; m_x < 10; m_x ++){
                        for (m_y = 0; m_y < 10; m_y ++){
                            int test_m = cache[m_x][m_y];
                            printf("%d", test_m);
                        }
                        printf("\n");
                    }
            printf("checking matrix end\n");
            passToHardware(cache);
            return 0;

        }else if(boolPut == 1){
            int temp_row, temp_col;
```

```
		int check = 0;
		for (temp_row = 0; temp_row < ysize; temp_row ++){
			for (temp_col = 0; temp_col < xsize; temp_col++){
				cache[0 + y + temp_row][0 + x + temp_col] =
temp[temp_row][temp_col] + cache[0 + y + temp_row][0 + x + temp_col];
					int judge;
					judge = cache[0 + y + temp_row][0 + x + temp_col];
					int block_part = temp[temp_row][temp_col];
					if (judge == 2 && block_part == 1){
						cache[0 + y + temp_row][0 + x + temp_col] = 3;
					}else if (judge == 1 && block_part == 1){
						cache[0 + y + temp_row][0 + x + temp_col] = 2;
					}
			}
		}
		passToHardware(cache);
		check = checkIfCanPut(cache);


		if(check == 0){
			return 0;
		}else if(check == 1){
			int put_x, put_y;
			for( put_x = 0 ; put_x < ysize ; put_x ++){
				for( put_y = 0; put_y < xsize; put_y ++){
					matrix[0 + y + put_x][0 + x + put_y] = temp[put_x][put_y] +
matrix[0 + y + put_x][0 + x + put_y];

				}
```

```
                        }
                        passToHardware(matrix);
                        return 1;

                }

        }

}



//CHECK INPUT VALUE
int selectBlock(int length, int *b, int **cache)
{
        int t = 0;
        int select_flag = 1;
        fprintf(stderr, "sB en\n");
        struct usb_keyboard_packet packet;
        uint8_t input_report[20];
        int transferred;
        char keystate[12];
        while(select_flag == 1){
if(!ifxbox)
        libusb_interrupt_transfer(keyboard, endpoint_address,
                                (unsigned char *) &packet, sizeof(packet),
                                &transferred, 0);

else{
```

```
            //fprintf(stderr, "ifxbox: %d\n",ifxbox);
            A2=A; B2=B; Up2=Up; Down2=Down;Left2=Left; Right2=Right;
libusb_control_transfer(xbox,
LIBUSB_ENDPOINT_IN|LIBUSB_REQUEST_TYPE_CLASS|LIBUSB_RECIPIENT_INTERFACE,
                        0x01, (0x01<<8)|0x00, 0, input_report, 20, 1000);
            A=input_report[3] == 16;
            B=input_report[3] == 32;
            Up=input_report[2] == 1;
            Down=input_report[2] == 2;
            Left=input_report[2] == 4;
            Right=input_report[2] == 8;
            //fprintf(stderr, "A %d, B %d\n Up %d, Down %d, Left %d, Right %d\n", A, B, Up, Down,
Left, Right);

}

int
xboxvalid=((A!=A2)||(B!=B2)||(Up!=Up2)||(Down!=Down2)||(Left!=Left2)||(Right!=Right2))&&(B2=
=0)&&(A2==0)&&(Up2==0)&&(Down2==0)&&(Left2==0)&&(Right2==0);
            if(xboxvalid) fprintf(stderr, "valid");
            //fprintf(stderr,"%d %d", transferred, sizeof(packet));
            if (transferred == sizeof(packet) || (xboxvalid && ifxbox) ) {
                        //pthread_create(&click_thread, NULL, click_thread_f, NULL);
                        //fprintf(stderr, "%02x", packet.keycode[0]);

               if (packet.keycode[0] == 0x29 || B){
sound(2);
printf("esc pressed\n");
                                    continue;
```

```
        }else if(packet.keycode[0] == 0x51 || Down){sound(1);
                    fprintf(stderr,"down pressed\n");
            if ( t == 2){
                    t = 0;
                    //hightlight b[j]
            }else{
                    t = t + 1;
            }
            write_segments(107, t);
            //down
        }else if(packet.keycode[0] == 0x52 || Up){sound(1);
                    fprintf(stderr,"up pressed\n");
            if ( t == 0){
                    t = 2;
            }else{
                    t = t - 1;
            }
            write_segments(107, t);
            //up
        }else if(packet.keycode[0] == 0x28 || A){  //enter
                    //rintf(stderr,"enter pressed\n");
                    sound(2);
            if (b[t] == 0){
                    continue;
                    //packet.keycode[0x51];if chosen block has been used, move cursor to next
block
            }else{


                    select_flag = 0;
```

```
                        return t;
                        //select_flag = 0;
                        //break;
                    }
            }
            }
        }
}

void genNewBlock(int set_new_blocks, int* b)
{
        int i, j;
        if (set_new_blocks == 1){
                srandom(time(NULL));
                for (i=0; i< 3; i++)
                {
                        b[i] = generate();
                        //printf("%d\n", b[i]);
                }
                //checkIfFullLine() = 0;
            //set_new_blocks = 0;
        }else{
                return;
        }

}

void showBlock(int j, int** cache){
        int temp;
```

```
temp = cache[10][j+1];
switch(temp){
        case 1:
                cache[0][0] = cache[0][1] = cache[0][2] = cache[0][3] = cache[0][4] = 2;
                break;
        case 2:
                cache[0][0] = cache[1][0] = cache[2][0] = cache[3][0] = cache[4][0] = 2;
                break;
        case 3:
                cache[0][0] = cache[0][1] = cache[0][2] = cache[0][3] = 2;
                break;
        case 4:
                cache[0][0] = cache[1][0] = cache[2][0] = cache[3][0] = 2;
                break;
        case 5:
                cache[0][0] = cache[1][0] = cache[0][1] = cache[1][1] = 2;
                break;
        case 6:
                cache[0][0] = cache[0][1] = cache[0][2] = cache[1][0] = cache[2][0] = 2;
                break;
        case 7:
                cache[0][0] = cache[0][1] = cache[0][2] = cache[1][2] = cache[2][2] = 2;
                break;
        case 8:
                cache[0][0] = cache[1][0] = 2;
                break;
        case 9:
                cache[0][0] = cache[0][1] = cache[0][2] = 2;
                break;
```

```
            case 10:
                cache[1][1] = 2;
                break;
            case 11:
                cache[0][0] = cache[0][1] = cache[1][0] = 2;
                break;
            case 12:
                cache[0][0] = cache[0][1] = cache[1][1] = cache[1][2] = 2;
                break;
            case 13:
                cache[0][0] = cache[0][1] = cache[0][2] = 2;
                break;
            case 14:
                cache[0][0] = cache[0][1] = 2;
                break;
            default:
                perror("no data input");
                break;

        }

        fprintf(stderr, "showed block on main");
        //passToHardware(cache);
}


void *timer_thread_f(){
        start = time(NULL);
        int TOTALTIME = 60;
```

```
	while(1){
		time_t dif = time(NULL) - start;
		write_segments(134, ((TOTALTIME-dif)/100)%10);
		write_segments(135, ((TOTALTIME-dif)/10)%10);
		write_segments(136, (TOTALTIME-dif)%10);
		if (TOTALTIME - dif <= 0 ){
			/*

			int time_x, time_y;
			for(time_x = 0; time_x < 10; time_x ++){
				for(time_y = 0; time_y < 10; time_y ++){
					matrix[time_x][time_y] = 3;
				}
			}
			passToHardware(matrix);
			*/
			printf("eeeeeeeeeeeeeeeeeeeeend");
			//pthread_exit(NULL);
			GAMEOVER = 1;
			//sexit(-1);
			}
		}
	}

/*
void *click_thread_f(){
	int start = clock();
	int end = 0;
	//****************************send start music signal************************
```

```
    while(1){
        end = clock();
        if (end - start == 1000){
            //**************send end music signal**************
            pthread_exit(NULL);
        }
    }

}
*/
//*************************************************************************
int main(void)
{
    vga_led_arg_t vla;
    static const char filename[] = "/dev/vga_led";

    fprintf(stderr, "loading MOD");

    if ( (vga_led_fd = open(filename, O_RDWR)) == -1) {
    fprintf(stderr, "could not open %s\n", filename);
    return -1;
    }

    int *b;
    int **matrix;
    int rows = MATRIX_ROW;
    int cols = MATRIX_COL;
    int i, k, t;
    int **cache;
```

```
	struct usb_keyboard_packet packet;
	int transferred;
	char keystate[12];
	int c, cache_row, cache_col;

	matrix = malloc(rows * sizeof *matrix);
	b = (int *)malloc(3 * sizeof (int));
	cache = malloc(rows * sizeof *cache);
	for (k=0; k<1; k++)
{
  b[k] = malloc(cols * sizeof(int));
}

for (t = 0; t < 3; t ++){
	b[t] = 0;
}

for (i=0; i<rows; i++)
{
  matrix[i] = malloc(cols * sizeof(int));
}

	for (c=0; c< rows; c++)
{
  cache[c] = malloc(cols * sizeof(int));
}

int m, n;
```

```
        for (m = 0; m < MATRIX_ROW; m++){
                for (n = 0; n < MATRIX_COL; n++){
                        //printf("x == %d", b5[0][1]);
                        matrix[m][n] = 0;
                        //array[i][j] = 1;
                        //fprintf(stderr, "initlized");
                }
        }

        //initiliaze cache matrix



      for (cache_row = 0; cache_row < 11; cache_row++){
                for (cache_col = 0; cache_col < 10; cache_col++){
                        //printf("x == %d", b5[0][1]);
                        cache[cache_row][cache_col] = matrix[cache_row][cache_col];
                        //array[i][j] = 1;
                }
        }

        int inChoose;
        int end;

        block1.height = 1;
        block1.width = 5;
        //block1.blockArray = b1;

        block2.height = 5;
        block2.width = 1;
```

```
//block1.blockArray = b2;

block3.height = 1;
block3.width = 4;
//block1.blockArray = b3;

block4.height = 4;
block4.width = 1;
//block4.blockArray = b4;

block5.height = 2;
block5.width = 2;
//block1.blockArray = b5;

block6.height = 3;
block6.width = 3;
//block1.blockArray = b6;

block7.height = 1;
block7.width = 5;
//block1.blockArray = b7;

block8.height = 2;
block8.width = 1;
//block1.blockArray = b8;

block9.height = 3;
block9.width = 1;
//block1.blockArray = b9;
```

```
block10.height = 1;
block10.width = 1;
//block1.blockArray = b10;

block11.height = 2;
block11.width = 3;
//block1.blockArray = b11;

block12.height = 2;
block12.width = 3;
//block1.blockArray = b1;2

block13.height = 1;
block13.width = 3;
//block1.blockArray = b13;

block14.height = 1;
block14.width = 2;
//block1.blockArray = b14;



genNewBlock(1, b);
inChoose = 1;

/*  matrix structure

 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
```

```
0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0
state b[0] b[1] b[2] end scr   j   0   0   0

*/
matrix[10][0] = inChoose;
matrix[10][1] = b[0];
matrix[10][2] = b[1];
matrix[10][3] = b[2];
cache[10][0] = inChoose;
cache[10][1] = b[0];
cache[10][2] = b[1];
cache[10][3] = b[2];
perror("started");
passToHardware(matrix);
write_segments(131, 0);
write_segments(132, 0);
perror("p2h out");
int j;
int set_new_blocks;
int found = 0;

//printf("keyboard?????");
```

```
	if ((keyboard = openkeyboard(&endpoint_address)) == NULL){
		fprintf(stderr, "keyboard not found" );
		//exit(1);
	}else{
	found = 1;
		fprintf(stderr,"keyboard found addr = %p\n",keyboard);
	}



	if((xbox = openxbox(&endpoint_address2))==NULL){
		fprintf(stderr, "xbox not found");
	}else{
	found = 1;
		fprintf(stderr, "xbox found addr = %p\n ",xbox);
		ifxbox = 1;
	}
	if(found ==0) exit(1);




	pthread_create(&timer_thread, NULL, timer_thread_f, NULL);

fprintf(stderr, "current endpoint = %p %p", endpoint_address, endpoint_address2);

	fprintf(stderr, "current 0x82? %d", endpoint_address == 0x82);

 for (;;) {
uint8_t input_report[20];
if(!ifxbox)
```

```
        libusb_interrupt_transfer(keyboard, endpoint_address,
                                (unsigned char *) &packet, sizeof(packet),
                                &transferred, 0);

else{
        //fprintf(stderr, "ifxbox: %d\n",ifxbox);
        A2=A; B2=B; Up2=Up; Down2=Down;Left2=Left; Right2=Right;
libusb_control_transfer(xbox,
LIBUSB_ENDPOINT_IN|LIBUSB_REQUEST_TYPE_CLASS|LIBUSB_RECIPIENT_INTERFACE,
                0x01, (0x01<<8)|0x00, 0, input_report, 20, 1000);
        A=input_report[3] == 16;
        B=input_report[3] == 32;
        Up=input_report[2] == 1;
        Down=input_report[2] == 2;
        Left=input_report[2] == 4;
        Right=input_report[2] == 8;
        //fprintf(stderr, "A %d, B %d\n Up %d, Down %d, Left %d, Right %d\n", A, B, Up, Down,
Left, Right);

}

int
xboxvalid=((A!=A2)||(B!=B2)||(Up!=Up2)||(Down!=Down2)||(Left!=Left2)||(Right!=Right2))&&(B2=
=0)&&(A2==0)&&(Up2==0)&&(Down2==0)&&(Left2==0)&&(Right2==0);
        if(xboxvalid) fprintf(stderr, "valid");
        //fprintf(stderr,"%d %d", transferred, sizeof(packet));
        if (transferred == sizeof(packet) || (xboxvalid && ifxbox) ) {
                fprintf(stderr, "key press catch\n");
```

```
////////////////////////////
//Main loop
inChoose = matrix[10][0];
if (inChoose == 1){
        passToHardware(matrix);
        j = selectBlock(3, b, matrix);
        showBlock(j, cache);
        //fprintf(stderr, "sB out");
        inChoose = 0;
        matrix[10][0] = inChoose;
        cache[10][0] = inChoose;
        //showBloack(j, cache);
        //passToHardware(cache);
        matrix[10][6] = j;
        passToHardware(matrix);

}else{
        fprintf(stderr, "put Block next\n");
        fprintf(stderr,"%d\n", j);
        //putBlock(j, b, matrix, cache);
        if (putBlock(j, b, matrix, cache) == 0) continue;
        b[j] = 0;//make used blocks blank
        matrix[10][j+1] = 0;
        passToHardware(matrix);
        clearLine(matrix);
        int boolEnd;
        boolEnd = 0;
        boolEnd = matrix[10][4];
        boolEnd = checkIfGameEnd(b, matrix);
```

```
if (boolEnd == 1){
        matrix[10][4] = 1;
        passToHardware(matrix);
        fprintf(stderr,"%s\n", "end");
        break;
}else{
        matrix[10][4] = 0;
        if (b[0] == 0){
                if (b[1] == 0){
                        if (b[2] == 0){
                                set_new_blocks = 1;
                                genNewBlock(set_new_blocks, b);
                        }
                }else{
                        set_new_blocks = 0;
                }
        }else{
                set_new_blocks = 0;
        }
        //genNewBlock(set_new_blocks);
        inChoose = 1;
        matrix[10][0] = 1;
        matrix[10][1] = b[0];
        matrix[10][2] = b[1];
        matrix[10][3] = b[2];
        cache[10][0] = 1;
        cache[10][1] = b[0];
        cache[10][2] = b[1];
        cache[10][3] = b[2];
```

```
                                        passToHardware(matrix);
                              }
                    }
                    //break;



                    }
          }

EXIT: return 1;
                    //return 0;
};

/*
 * Avalon memory-mapped peripheral for the VGA LED Emulator
 *
 * Stephen A. Edwards
 * Columbia University
 */

module VGA_LED(input logic        clk,
               input logic    reset,
               input logic [7:0]  writedata,
               input logic    write,
               input                    chipselect,
               input logic [7:0]  address,

               output logic [7:0] VGA_R, VGA_G, VGA_B,
               output logic             VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n,
              output logic            VGA_SYNC_n);




         logic[7:0] first_sel,second_sel,third_sel,arrow_sel, score1, score2, array_value, time1, time2,
time3,finish;
         logic[7:0] array_address;
         logic write_en;



    VGA_LED_Emulator led_emulator(.clk50(clk), .*);

         always_ff @(posedge clk) begin
                  write_en = 0;
                  if(reset)begin
                           array_value <= 0;



                  end else if(chipselect && write) begin
                    if (address[7] == 0) begin
                                    array_value <= writedata[7:0];
                                    array_address <= address[7:0];
                                    write_en =1;
                           end
                           else
                           case(address[4:0])
                           5'd0 : first_sel <= writedata;
                           5'd1 : second_sel <= writedata;
                           5'd2 : third_sel <= writedata;
                    5'd3 : score1 <= writedata;
                    5'd4 : score2 <= writedata;
                    5'd5 : arrow_sel <= writedata;
                    5'd6 : time1 <= writedata;
                    5'd7 : time2 <= writedata;
                    5'd8 : time3 <= writedata;
                    5'd10 : finish <= writedata;
                    endcase
                end
        end

 //assign array_index = 8'd21;
        //assign array_value = 8'd1;

endmodule
```

VGA_LED_EMULATOR.sv:

```
/*
 * Seven-segment LED emulator
 *
 * Stephen A. Edwards, Columbia University
 */

module VGA_LED_Emulator(
 input logic        clk50, reset,
// input logic [10:0]  soft_in_x, soft_in_y,
// input reg [1:0] array [0:100];

 /*input logic [3:0] first_sel, second_sel, third_sel,  //selection input
 input logic first_used, second_used, third_used, //if used input
 input logic [1:0] arrow_sel,*/

 input logic [7:0] array_address,
 input logic [7:0] array_value,
 input logic [7:0] first_sel,second_sel,third_sel,arrow_sel, score1, score2, time1, time2, time3, finish,
 input write_en,
 output logic [7:0] VGA_R, VGA_G, VGA_B,
 output logic       VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);

/*
 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
 *
 * HCOUNT 1599 0           1279      1599 0
 *          _____________          _______
 * _________|    Video    |__________|  Video
 *
 *
 * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
 *     ____________________     ___________
 * |___|      VGA_HS        |____|
 */
   // Parameters for hcount
   parameter HACTIVE      = 11'd 1280,
         HFRONT_PORCH = 11'd 32,
         HSYNC        = 11'd 192,
         HBACK_PORCH  = 11'd 96,
         HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC + HBACK_PORCH; // 1600
```

```
// Parameters for vcount
parameter VACTIVE      = 10'd 480,
        VFRONT_PORCH = 10'd 10,
        VSYNC       = 10'd 2,
        VBACK_PORCH  = 10'd 33,
        VTOTAL      = VACTIVE + VFRONT_PORCH + VSYNC + VBACK_PORCH; // 525

logic [10:0]                    hcount; // Horizontal counter
                              // Hcount[10:1] indicates pixel column (0-639)
logic                         endOfLine;

always_ff @(posedge clk50 or posedge reset)
  if (reset)          hcount <= 0;
  else if (endOfLine) hcount <= 0;
  else                hcount <= hcount + 11'd 1;

assign endOfLine = hcount == HTOTAL - 1;

// Vertical counter
logic [9:0]                     vcount;
logic                         endOfField;

always_ff @(posedge clk50 or posedge reset)
  if (reset)          vcount <= 0;
  else if (endOfLine)
    if (endOfField)   vcount <= 0;
    else              vcount <= vcount + 10'd 1;

assign endOfField = vcount == VTOTAL - 1;
```

```
   // Horizontal sync: from 0x520 to 0x5DF (0x57F)
   // 101 0010 0000 to 101 1101 1111
   assign VGA_HS = !( (hcount[10:8] == 3'b101) & !(hcount[7:5] == 3'b111));
   assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

   assign VGA_SYNC_n = 1; // For adding sync to video signals; not used for VGA

   // Horizontal active: 0 to 1279     Vertical active: 0 to 479
   // 101 0000 0000  1280              01 1110 0000  480
   // 110 0011 1111  1599              10 0000 1100  524
   assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
                        !( vcount[9] | (vcount[8:5] == 4'b1111) );

   /* VGA_CLK is 25 MHz
    *          __    __    __
    * clk50    __|  |__|  |__|
    *
    *          _____      __
    * hcount[0]__|    |_____|
    */
   assign VGA_CLK = hcount[0]; // 25 MHz clock: pixel latched on rising edge
   //hardware and software distance
  // logic[10:0] soft_hard_x, soft_hard_y;

        //assign   soft_hard_x   =   (soft_in_x   >=   hcount)?   ((soft_in_x   -   hcount)>>1):   ((hcount   -
soft_in_x)>>1);
//        assign soft_hard_y = (soft_in_y >= vcount)? (soft_in_y - vcount): (vcount - soft_in_y);
```

```
            /////logic of select////
        logic grid; //all grid area
        assign   grid   =   ((hcount[10:6]   >=   5'd2)&(hcount[10:6]<=   5'd11))   &   ((vcount[9:0]>=
10'd64)&(vcount[9:0]<=10'd383));
        //h:000 1000 0000  v:00 0100 0000
        //  011 0000 0000    01 1000 0000


        ///////calculate read data address///////
        logic [7:0] read_address;
        logic [9:0] num_add;
        logic [9:0] time_add;
        logic [23:0] data_num0;
        logic [23:0] data_num1;
        logic [23:0] data_num2;
        logic [23:0] data_num3;
        logic [23:0] data_num4;
        logic [23:0] data_num5;
        logic [23:0] data_num6;
        logic [23:0] data_num7;
        logic [23:0] data_num8;
        logic [23:0] data_num9;



        //////color bits ////
        logic [7:0] color_data;
        logic color_on;
```

```
	////memory connection//////
	grid_array
grid_array(.clock(clk50),.data(array_value),.wraddress(array_address),.rdaddress(read_address),.q(co
lor_data),.wren(write_en));
	num0 num0 (.clock(clk50), .address(num_add), .q(data_num0));
	num1 num1 (.clock(clk50), .address(num_add), .q(data_num1));
	num2 num2 (.clock(clk50), .address(num_add), .q(data_num2));
	num3 num3 (.clock(clk50), .address(num_add), .q(data_num3));
	num4 num4 (.clock(clk50), .address(num_add), .q(data_num4));
	num5 num5 (.clock(clk50), .address(num_add), .q(data_num5));
	num6 num6 (.clock(clk50), .address(num_add), .q(data_num6));
	num7 num7 (.clock(clk50), .address(num_add), .q(data_num7));
	num8 num8 (.clock(clk50), .address(num_add), .q(data_num8));
	num9 num9 (.clock(clk50), .address(num_add), .q(data_num9));


	////grid controller///
	always_comb
		begin
		read_address <= 0;
		color_on <=0;
		if(grid) begin
			read_address <= (hcount[10:6]-2)  +(vcount[9:5]-2)*10;
			color_on <= 1;
		end

	end
	/////score controller/////
	logic[23:0] num_1000,num_100,num_10,num_1, clock_100, clock_10, clock_1;
	logic   num_1000_on,num_100_on,   num_10_on,   num_1_on,   clock_100_on,   clock_10_on,
clock_1_on;
	/*
	always_comb begin
		num_1000 <= data_num0;
		num_100 <= data_num0;
		num_10 <= data_num0;
		num_1 <= data_num0;
	end
	*/

	always_comb begin
		case (score1[7:4])
		4'h9: num_1000 <= data_num9;
		4'h8: num_1000 <= data_num8;
		4'h7: num_1000 <= data_num7;
		4'h6: num_1000 <= data_num6;
		4'h5: num_1000 <= data_num5;
		4'h4: num_1000 <= data_num4;
		4'h3: num_1000 <= data_num3;
		4'h2: num_1000 <= data_num2;
		4'h1: num_1000 <= data_num1;
		4'h0: num_1000 <= data_num0;
		default: num_1000 <= data_num0;
		endcase

		case (score1[3:0])
```

```
4'h9: num_100 <= data_num9;
4'h8: num_100 <= data_num8;
4'h7: num_100 <= data_num7;
4'h6: num_100 <= data_num6;
4'h5: num_100 <= data_num5;
4'h4: num_100 <= data_num4;
4'h3: num_100 <= data_num3;
4'h2: num_100 <= data_num2;
4'h1: num_100 <= data_num1;
4'h0: num_100 <= data_num0;
default: num_100 <= data_num0;
endcase

case (score2[7:4])
4'h9: num_10 <= data_num9;
4'h8: num_10 <= data_num8;
4'h7: num_10 <= data_num7;
4'h6: num_10 <= data_num6;
4'h5: num_10 <= data_num5;
4'h4: num_10 <= data_num4;
4'h3: num_10 <= data_num3;
4'h2: num_10 <= data_num2;
4'h1: num_10 <= data_num1;
4'h0: num_10 <= data_num0;
default: num_10 <= data_num0;
endcase

case (score2[3:0])
4'h9: num_1 <= data_num9;
```

```
            4'h8: num_1 <= data_num8;
            4'h7: num_1 <= data_num7;
            4'h6: num_1 <= data_num6;
            4'h5: num_1 <= data_num5;
            4'h4: num_1 <= data_num4;
            4'h3: num_1 <= data_num3;
            4'h2: num_1 <= data_num2;
            4'h1: num_1 <= data_num1;
            4'h0: num_1 <= data_num0;
            default: num_1 = data_num0;
            endcase
end

always_comb begin
            case (time1[3:0])
            4'h9: clock_100 <= data_num9;
            4'h8: clock_100 <= data_num8;
            4'h7: clock_100 <= data_num7;
            4'h6: clock_100 <= data_num6;
            4'h5: clock_100 <= data_num5;
            4'h4: clock_100 <= data_num4;
            4'h3: clock_100 <= data_num3;
            4'h2: clock_100 <= data_num2;
            4'h1: clock_100 <= data_num1;
            4'h0: clock_100 <= data_num0;
            default: clock_100 <= data_num0;
            endcase

            case (time2[3:0])
```

```
/*

                4'h9: clock_10 <= data_num9;
                4'h8: clock_10 <= data_num8;
                4'h7: clock_10 <= data_num7;
                4'h6: clock_10 <= data_num6;
                4'h5: clock_10 <= data_num5;
                4'h4: clock_10 <= data_num4;
                4'h3: clock_10 <= data_num3;
                4'h2: clock_10 <= data_num2;
                4'h1: clock_10 <= data_num1;
                4'h0: clock_10 <= data_num0;
                default: clock_10 = data_num0;
                endcase

                case (time3[3:0])
                4'h9: clock_1 <= data_num9;
                4'h8: clock_1 <= data_num8;
                4'h7: clock_1 <= data_num7;
                4'h6: clock_1 <= data_num6;
                4'h5: clock_1 <= data_num5;
                4'h4: clock_1 <= data_num4;
                4'h3: clock_1 <= data_num3;
                4'h2: clock_1 <= data_num2;
                4'h1: clock_1 <= data_num1;
                4'h0: clock_1 <= data_num0;
                default: clock_1 <= data_num0;
                endcase
            end

/*
```

```
        always_comb begin


                if
((hcount[10:0]>=500)&(hcount[10:0]<=545)&(vcount[9:0]>=416)&(vcount[9:0]<=447))
                        begin
                                num_add <= (hcount[10:1]-250)+(vcount[4:0])*23;
                                clock_100_on <= 1;
                                clock_10_on <= 0;
                                clock_1_on <=0;
                        end

                else    if
((hcount[10:0]>=546)&(hcount[10:0]<=591)&(vcount[9:0]>=416)&(vcount[9:0]<=447))
                        begin
                                num_add <= (hcount[10:1]-273)+(vcount[4:0])*23;
                                clock_100_on <= 0;
                                clock_10_on <= 1;
                                clock_1_on <=0;
                        end

                else    if
((hcount[10:0]>=592)&(hcount[10:0]<=637)&(vcount[9:0]>=416)&(vcount[9:0]<=447))
                        begin
                                num_add <= (hcount[10:1]-296)+(vcount[4:0])*23;
                                clock_100_on <= 0;
                                clock_10_on <= 0;

                                clock_1_on <=1;

                        end

                else

                        begin

                                clock_100_on <=0;

                                clock_10_on <=0;

                                clock_1_on <=0;

                                num_add <= 0;

                        end

                end

                        */




        always_comb begin

                        num_1000_on <= 0;

                        num_100_on <= 0;

                        num_10_on <= 0;

                        num_1_on <= 0;

                        num_add <= 0;

                        clock_100_on <=0;

                        clock_10_on <=0;

                        clock_1_on <=0;



if((hcount[10:0]>=192)&(hcount[10:0]<=237)&(vcount[9:0]>=416)&(vcount[9:0]<=447))

                        begin

                        num_add <= (hcount[10:1]-96)+(vcount[4:0])*23;
```

```
                        num_1000_on <= 1;
                        num_100_on <= 0;
                        num_10_on <= 0;
                        num_1_on <= 0;
                        end

                else                                                                    if
((hcount[10:0]>=238)&(hcount[10:0]<=283)&(vcount[9:0]>=416)&(vcount[9:0]<=447))
                        begin
                        num_add <= (hcount[10:1]-119)+(vcount[4:0])*23;
                        num_1000_on <= 0;
                        num_100_on <= 1;
                        num_10_on <= 0;
                        num_1_on <= 0;
                        end
                else                                                                    if
((hcount[10:0]>=284)&(hcount[10:0]<=329)&(vcount[9:0]>=416)&(vcount[9:0]<=447))
                        begin
                        num_add <= (hcount[10:1]-142)+(vcount[4:0])*23;
                        num_1000_on <= 0;
                        num_100_on <= 0;
                        num_10_on <= 1;
                        num_1_on <= 0;
                        end
                else                                                                    if
((hcount[10:0]>=330)&(hcount[10:0]<=375)&(vcount[9:0]>=416)&(vcount[9:0]<=447))
                        begin
                        num_add <= (hcount[10:1]- 165)+(vcount[4:0])*23;
                        num_1000_on <= 0;
                        num_100_on <= 0;
                        num_10_on <= 0;
                        num_1_on <= 1;
                        end

                else                                                              if
((hcount[10:0]>=500)&(hcount[10:0]<=545)&(vcount[9:0]>=416)&(vcount[9:0]<=447))
                        begin
                             num_add <= (hcount[10:1]-250)+(vcount[4:0])*23;
                             clock_100_on <= 1;
                             clock_10_on <= 0;
                             clock_1_on <=0;
                        end

                else     if
((hcount[10:0]>=546)&(hcount[10:0]<=591)&(vcount[9:0]>=416)&(vcount[9:0]<=447))
                        begin
                             num_add <= (hcount[10:1]-273)+(vcount[4:0])*23;
                             clock_100_on <= 0;
                             clock_10_on <= 1;
                             clock_1_on <=0;
                        end

                else     if
((hcount[10:0]>=592)&(hcount[10:0]<=637)&(vcount[9:0]>=416)&(vcount[9:0]<=447))
                        begin
                             num_add <= (hcount[10:1]-296)+(vcount[4:0])*23;
                             clock_100_on <= 0;
                             clock_10_on <= 0;
```

```
                              clock_1_on <=1;
                    end
end




logic [5:0] linex; // coordinates for drawing the black line
logic [4:0] liney;

assign linex = hcount[6:1];
assign liney = vcount[4:0];

logic line; // vertical and horizontal line

assign line = ((linex[4:0] == 5'd31) | (liney[4:0] == 5'b11111));
//assign line = (linex[5:0] == 6'b111111);

//14 difference kinds of tetris
//central for each tetris

logic one_range;
logic two_range;
logic three_range;

assign one_range = ((vcount[8:0] >= 9'd40)&(vcount[8:0] <= 9'd120));
assign two_range = ((vcount[8:0]>= 9'd200)&(vcount[8:0]<=9'd280));
assign three_range = ((vcount[8:0]>= 9'd360)&(vcount[8:0]<=9'd440));
```

```
        logic tetris_line_v;
        logic tetris_line_h;

        assign tetris_line_v = (hcount[4:0] == 5'd0);
        assign tetris_line_h = (vcount[3:0] == 4'd8);

                /*896 928 960 992 1024 1055

         _ _ _ _ _  40 (200) (360)
        |_|_|_|_|_|  56 (216) (376)
        |_|_|_|_|_|  72 (232) (392)
        |_|_|_|_|_|  88 (248) (408)
        |_|_|_|_|_|  104 (264) (424)
        |_|_|_|_|_|  120 (280) (440)

        reference of the block
*/

        //1*5
        logic one_five_x;
        logic one_five_y;

        assign one_five_x = ((hcount[10:0]>=11'd896)&(hcount[10:0]<=11'd1056));
        assign          one_five_y          =          ((vcount[8:0]>=9'd72)&(vcount[8:0]<=9'd88))          |
((vcount[8:0]>=9'd232)&(vcount[8:0]<=9'd248)) | ((vcount[8:0]>=9'd392)&(vcount[8:0]<=9'd408));

        //1*4
        logic one_four_x;
```

```
	logic one_four_y;

	assign one_four_x =((hcount[10:0]>=11'd896)&(hcount[10:0]<=11'd1024));
	assign one_four_y = ((vcount[8:0]>=9'd72)&(vcount[8:0]<=9'd88)) |
((vcount[8:0]>=9'd232)&(vcount[8:0]<=9'd248)) | ((vcount[8:0]>=9'd392)&(vcount[8:0]<=9'd408));

	//5*1
	logic five_one_x;
	logic five_one_y;

	assign five_one_x = ((hcount[10:0]>=11'd960)&(hcount[10:0]<=11'd992));
	assign five_one_y = ((vcount[8:0]>=9'd40)&(vcount[8:0]<=9'd120)) |
((vcount[8:0]>=9'd200)&(vcount[8:0]<=9'd280)) | ((vcount[8:0]>=9'd360)&(vcount[8:0]<=9'd440));

	//4*1
	logic four_one_x;
	logic four_one_y;

	assign four_one_x = ((hcount[10:0]>=11'd960)&(hcount[10:0]<=11'd992));
	assign four_one_y = ((vcount[8:0]>=9'd56)&(vcount[8:0]<=9'd120)) |
((vcount[8:0]>=9'd216)&(vcount[8:0]<=9'd280)) | ((vcount[8:0]>=9'd376)&(vcount[8:0]<=9'd440));

	//square
	logic two_two_x;
	logic two_two_y;

	assign two_two_x = ((hcount[10:0]>=11'd928)&(hcount[10:0]<=11'd992));
	assign two_two_y = ((vcount[8:0]>=9'd72)&(vcount[8:0]<=9'd104)) |
((vcount[8:0]>=9'd232)&(vcount[8:0]<=9'd264)) | ((vcount[8:0]>=9'd392)&(vcount[8:0]<=9'd424));
```

```
	// inverse_7
	logic invert_7_top;
	logic invert_7_bottom;

	assign invert_7_top = ((hcount[10:0]>=11'd928)&(hcount[10:0]<=11'd1024)&
(((vcount[8:0]>=9'd56)&(vcount[8:0]<=9'd72)) | ((vcount[8:0]>=9'd216)&(vcount[8:0]<=9'd232)) |
((vcount[8:0]>=9'd376)&(vcount[8:0]<=9'd392))));
	assign invert_7_bottom = ((hcount[10:0]>=11'd928)&(hcount[10:0]<=11'd960)&
(((vcount[8:0]>=9'd72)&(vcount[8:0]<=9'd104)) | ((vcount[8:0]>=9'd232)&(vcount[8:0]<=9'd264)) |
((vcount[8:0]>=9'd392)&(vcount[8:0]<=9'd424))));

	//seven
	logic seven_top;
	logic seven_bottom;

	assign seven_top = ((hcount[10:0]>=11'd928)&(hcount[10:0]<=11'd1024)&
(((vcount[8:0]>=9'd56)&(vcount[8:0]<=9'd72)) | ((vcount[8:0]>=9'd216)&(vcount[8:0]<=9'd232)) |
((vcount[8:0]>=9'd376)&(vcount[8:0]<=9'd392))));
	assign seven_bottom = ((hcount[10:0]>=11'd992)&(hcount[10:0]<=11'd1024)&
(((vcount[8:0]>=9'd72)&(vcount[8:0]<=9'd104)) | ((vcount[8:0]>=9'd232)&(vcount[8:0]<=9'd264)) |
((vcount[8:0]>=9'd392)&(vcount[8:0]<=9'd424))));

	//2*1
	logic two_one_x;
	logic two_one_y;

	assign two_one_x = ((hcount[10:0]>=11'd960)&(hcount[10:0]<=11'd992));
	assign two_one_y = ((vcount[8:0]>=9'd72)&(vcount[8:0]<=9'd104)) |
((vcount[8:0]>=9'd232)&(vcount[8:0]<=9'd264)) | ((vcount[8:0]>=9'd392)&(vcount[8:0]<=9'd424));

	//3*1
	logic three_one_x;
	logic three_one_y;

	assign three_one_x = ((hcount[10:0]>=11'd960)&(hcount[10:0]<=11'd992));
	assign three_one_y = ((vcount[8:0]>=9'd56)&(vcount[8:0]<=9'd104)) |
((vcount[8:0]>=9'd216)&(vcount[8:0]<=9'd264)) | ((vcount[8:0]>=9'd376)&(vcount[8:0]<=9'd424));

	//1*1
	logic one_one_x;
	logic one_one_y;

	assign one_one_x = ((hcount[10:0]>=11'd960)&(hcount[10:0]<=11'd992));
	assign one_one_y = ((vcount[8:0]>=9'd72)&(vcount[8:0]<=9'd88)) |
((vcount[8:0]>=9'd232)&(vcount[8:0]<=9'd248)) | ((vcount[8:0]>=9'd392)&(vcount[8:0]<=9'd408));

	//s
	logic s_top;
	logic s_bottom;

	assign s_top = ((hcount[10:0]>=11'd960)&(hcount[10:0]<=11'd1024)&
(((vcount[8:0]>=9'd72)&(vcount[8:0]<=9'd88)) | ((vcount[8:0]>=9'd232)&(vcount[8:0]<=9'd248)) |
((vcount[8:0]>=9'd392)&(vcount[8:0]<=9'd408))));
	assign s_bottom = ((hcount[10:0]>=11'd960)&(hcount[10:0]<=11'd992)&
(((vcount[8:0]>=9'd88)&(vcount[8:0]<=9'd104)) | ((vcount[8:0]>=9'd248)&(vcount[8:0]<=9'd264)) |
((vcount[8:0]>=9'd408)&(vcount[8:0]<=9'd424))));
```

```
	//invert_s
	logic invert_s_top;
	logic invert_s_bottom;

	assign invert_s_top = ((hcount[10:0]>=11'd928)&(hcount[10:0]<=11'd992)&
(((vcount[8:0]>=9'd72)&(vcount[8:0]<=9'd88)) | ((vcount[8:0]>=9'd232)&(vcount[8:0]<=9'd248)) |
((vcount[8:0]>=9'd392)&(vcount[8:0]<=9'd408))));
	assign invert_s_bottom = ((hcount[10:0]>=11'd960)&(hcount[10:0]<=11'd1024)&
(((vcount[8:0]>=9'd88)&(vcount[8:0]<=9'd104)) | ((vcount[8:0]>=9'd248)&(vcount[8:0]<=9'd264)) |
((vcount[8:0]>=9'd408)&(vcount[8:0]<=9'd424))));

	//1*3
	logic one_three_x;
	logic one_three_y;

	assign one_three_x =((hcount[10:0]>=11'd928)&(hcount[10:0]<=11'd1024));
	assign one_three_y = ((vcount[8:0]>=9'd72)&(vcount[8:0]<=9'd88)) |
((vcount[8:0]>=9'd232)&(vcount[8:0]<=9'd248)) | ((vcount[8:0]>=9'd392)&(vcount[8:0]<=9'd408));

	//1*2
	logic one_two_x;
	logic one_two_y;

	assign one_two_x =((hcount[10:0]>=11'd928)&(hcount[10:0]<=11'd992));
	assign one_two_y = ((vcount[8:0]>=9'd72)&(vcount[8:0]<=9'd88)) |
((vcount[8:0]>=9'd232)&(vcount[8:0]<=9'd248)) | ((vcount[8:0]>=9'd392)&(vcount[8:0]<=9'd408));

	//arrow
	logic arrow_one_x;
logic arrow_one_y;

assign arrow_one_x = ((hcount[10:0]>= 1120-(vcount-71)*2)&(hcount[10:0]<=1120));
assign arrow_one_y = ((vcount[8:0]>= 9'd72)&(vcount[8:0]<=9'd88));

logic arrow_two_x;
logic arrow_two_y;

assign arrow_two_x = ((hcount[10:0]>= 1120-(vcount-231)*2)&(hcount[10:0]<=1120));
assign arrow_two_y = ((vcount[8:0]>= 9'd232)&(vcount[8:0]<=9'd248));

logic arrow_three_x;
logic arrow_three_y;

assign arrow_three_x = ((hcount[10:0]>= 1120-(vcount-391)*2)&(hcount[10:0]<=1120));
assign arrow_three_y = ((vcount[8:0]>= 9'd392)&(vcount[8:0]<=9'd408));


////arrow controller//////
logic curr_arrow;
assign curr_arrow = arrow_sel == 8'd0 ? (arrow_one_x & arrow_one_y):
                                         arrow_sel == 8'd1 ? (arrow_two_x & arrow_two_y):
                                        (arrow_three_x & arrow_three_y);

/////type select controller////
logic cur_first;
logic cur_second;
logic cur_third;
```

```
assign cur_first = first_sel == 8'd0 ? 0:
                                                  first_sel == 8'd1 ? (one_five_x & one_five_y):
                                           first_sel == 8'd3 ? (one_four_x & one_four_y):
                                           first_sel == 8'd2 ? (five_one_x & five_one_y):
                                           first_sel == 8'd4 ? (four_one_x & four_one_y):
                                           first_sel == 8'd5 ? (two_two_x & two_two_y):
                                           first_sel == 8'd6 ? (invert_7_top | invert_7_bottom):
                                           first_sel == 8'd7 ? (seven_top | seven_bottom):
                                           first_sel == 8'd8 ? (two_one_x & two_one_y):
                                           first_sel == 8'd9 ? (three_one_x & three_one_y):
                                           first_sel == 8'd10 ? (one_one_x & one_one_y):
                                           first_sel == 8'd11 ? (s_top | s_bottom):
                                           first_sel == 8'd12 ? (invert_s_top | invert_s_bottom):
                                           first_sel == 8'd13 ? (one_three_x & one_three_y):
                                           (one_two_x & one_two_y);

assign cur_second = second_sel == 8'd0 ? 0:
                                                  second_sel == 8'd1 ? (one_five_x & one_five_y):
                                           second_sel == 8'd3 ? (one_four_x & one_four_y):
                                           second_sel == 8'd2 ? (five_one_x & five_one_y):
                                           second_sel == 8'd4 ? (four_one_x & four_one_y):
                                           second_sel == 8'd5 ? (two_two_x & two_two_y):
                                           second_sel == 8'd6 ? (invert_7_top | invert_7_bottom):
                                           second_sel == 8'd7 ? (seven_top | seven_bottom):
                                           second_sel == 8'd8 ? (two_one_x & two_one_y):
                                           second_sel == 8'd9 ? (three_one_x & three_one_y):
                                           second_sel == 8'd10 ? (one_one_x & one_one_y):
                                           second_sel == 8'd11 ? (s_top | s_bottom):
                                           second_sel == 8'd12 ? (invert_s_top | invert_s_bottom):
```

```
                                        second_sel == 8'd13 ? (one_three_x & one_three_y):
                                        (one_two_x & one_two_y);


assign cur_third = third_sel == 8'd0 ? 0:
                                            third_sel == 8'd1 ? (one_five_x & one_five_y):
                                        third_sel == 8'd3 ? (one_four_x & one_four_y):
                                        third_sel == 8'd2 ? (five_one_x & five_one_y):
                                        third_sel == 8'd4 ? (four_one_x & four_one_y):
                                        third_sel == 8'd5 ? (two_two_x & two_two_y):
                                        third_sel == 8'd6 ? (invert_7_top | invert_7_bottom):
                                        third_sel == 8'd7 ? (seven_top | seven_bottom):
                                        third_sel == 8'd8 ? (two_one_x & two_one_y):
                                        third_sel == 8'd9 ? (three_one_x & three_one_y):
                                        third_sel == 8'd10 ? (one_one_x & one_one_y):
                                        third_sel == 8'd11 ? (s_top | s_bottom):
                                        third_sel == 8'd12 ? (invert_s_top | invert_s_bottom):
                                        third_sel == 8'd13 ? (one_three_x & one_three_y):
                                        (one_two_x & one_two_y);

logic finish_on;


always_comb begin
        finish_on = 0;
        if(finish[7:0] ==8'd1)
                finish_on=1;


end
```

```
    always_comb begin

        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff}; // white

        if(finish_on)begin
            if (grid)
                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00}; //
            if(line & grid)
                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff}; // grey
            if(curr_arrow)
            {VGA_R, VGA_G, VGA_B} = {8'h20, 8'h20, 8'h20}; //grey
        end




                if (color_on) begin

                    if (color_data==1)
                        {VGA_R,  VGA_G,  VGA_B}  =  {8'h80,  8'h80,  8'h80};
//grey
                    else if (color_data==2)
                        {VGA_R,  VGA_G,  VGA_B}  =  {8'h00,  8'hff,  8'h00};  //
green
                    else if (color_data==3)
                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00}; //red
                    else
                        {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00}; //
black
                end



                if(clock_100_on)begin
                    {VGA_R, VGA_G, VGA_B} = clock_100;
                end
                else if(clock_10_on)begin
                    {VGA_R, VGA_G, VGA_B} = clock_10;
                end
                else if(clock_1_on)begin
                    {VGA_R, VGA_G, VGA_B} = clock_1;
                end





                if(num_1000_on)begin
                    {VGA_R, VGA_G, VGA_B} = num_1000;
                end
                else if(num_100_on)begin
                    {VGA_R, VGA_G, VGA_B} = num_100;
                end
                else if(num_10_on)begin
                    {VGA_R, VGA_G, VGA_B} = num_10;
```

```
            end
            else if(num_1_on)begin
                {VGA_R, VGA_G, VGA_B} = num_1;
            end



        if(line & grid)
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff}; // grey

        if (cur_first & one_range)
                if(tetris_line_h | tetris_line_v)
                    {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff}; //white
                else
                    {VGA_R,  VGA_G,  VGA_B}  =  {8'h20,  8'h20,  8'h20};
//grey



        if (cur_second & two_range)
                if(tetris_line_h | tetris_line_v)
                    {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff}; //white
                else
                    {VGA_R,  VGA_G,  VGA_B}  =  {8'h20,  8'h20,  8'h20};
//grey

        if (cur_third & three_range)
                if(tetris_line_h | tetris_line_v)
                    {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff}; //white
                else
                    {VGA_R,  VGA_G,  VGA_B}  =  {8'h20,  8'h20,  8'h20};
```

```
//grey

                              if(curr_arrow)
                                              {VGA_R, VGA_G, VGA_B} = {8'h20, 8'h20, 8'h20}; //grey
         end
endmodule
```

MakeFIle:

```
EXTRA_CFLAGS= -Wall -g



ifneq (${KERNELRELEASE},)

# KERNELRELEASE defined: we are being compiled as part of the Kernel
     obj-m := vga_led.o

else

# We are being compiled as a module: use the Kernel build system

         KERNEL_SOURCE := /usr/src/linux
     PWD := $(shell pwd)

default: module project

project: project.o usbkeyboard.o
         gcc -g -o project usbkeyboard.c usbkeyboard.h project.c -lusb-1.0 -lpthread
```

```
project.o: project.c usbkeyboard.h

usbkeyboard.o: usbkeyboard.c usbkeyboard.h



module:
	${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} modules

clean:
	${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} clean
	${RM} project

endif
```

Usbkeyboard.c

```
#include "usbkeyboard.h"

#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>


/* References on libusb 1.0 and the USB HID/keyboard protocol
 *
 * http://libusb.org
 * http://www.dreamincode.net/forums/topic/148707-introduction-to-using-libusb-10/
 * http://www.usb.org/developers/devclass_docs/HID1_11.pdf
 * http://www.usb.org/developers/devclass_docs/Hut1_11.pdf
 */

/*
 * Find and return a USB keyboard device or NULL if not found
 * The argument con
 *
 */
#define B(x) (((x)!=0)?1:0)
#define RETRY_MAX                     5
#define REQUEST_SENSE_LENGTH          0x12
#define INQUIRY_LENGTH                0x24
#define READ_CAPACITY_LENGTH          0x08

// HID Class-Specific Requests values. See section 7.2 of the HID specifications
#define HID_GET_REPORT                0x01
#define HID_GET_IDLE                  0x02
#define HID_GET_PROTOCOL              0x03
#define HID_SET_REPORT                0x09
#define HID_SET_IDLE                  0x0A
#define HID_SET_PROTOCOL              0x0B
#define HID_REPORT_TYPE_INPUT         0x01
#define HID_REPORT_TYPE_OUTPUT        0x02
#define HID_REPORT_TYPE_FEATURE       0x03

struct libusb_device_handle *openkeyboard(uint8_t *endpoint_address) {
  libusb_device **devs;
  struct libusb_device_handle *keyboard = NULL;
  struct libusb_device_descriptor desc;
```

```
ssize_t num_devs, d;
uint8_t i, k;

/* Start the library */
if ( libusb_init(NULL) < 0 ) {
  fprintf(stderr, "Error: libusb_init failed\n");
  exit(1);
}

/* Enumerate all the attached USB devices */
if ( (num_devs = libusb_get_device_list(NULL, &devs)) < 0 ) {
  fprintf(stderr, "Error: libusb_get_device_list failed\n");
  exit(1);
}

/* Look at each device, remembering the first HID device that speaks
   the keyboard protocol */

for (d = 0 ; d < num_devs ; d++) {
  libusb_device *dev = devs[d];
  if ( libusb_get_device_descriptor(dev, &desc) < 0 ) {
    fprintf(stderr, "Error: libusb_get_device_descriptor failed\n");
    exit(1);
  }

  if (desc.bDeviceClass == LIBUSB_CLASS_PER_INTERFACE) {
    struct libusb_config_descriptor *config;
    libusb_get_config_descriptor(dev, 0, &config);
    for (i = 0 ; i < config->bNumInterfaces ; i++)
```

```
            for ( k = 0 ; k < config->interface[i].num_altsetting ; k++ ) {
             const struct libusb_interface_descriptor *inter =
               config->interface[i].altsetting + k ;
             if ( inter->bInterfaceClass == LIBUSB_CLASS_HID &&
                  inter->bInterfaceProtocol == USB_HID_KEYBOARD_PROTOCOL) {
               int r;
               if ((r = libusb_open(dev, &keyboard)) != 0) {
                 fprintf(stderr, "Error: libusb_open failed: %d\n", r);
                 exit(1);
               }
               if (libusb_kernel_driver_active(keyboard,i))
                 libusb_detach_kernel_driver(keyboard, i);
               //libusb_set_auto_detach_kernel_driver(keyboard, i);
               if ((r = libusb_claim_interface(keyboard, i)) != 0) {
                 fprintf(stderr, "Error: libusb_claim_interface failed: %d\n", r);
                 exit(1);
               }
               *endpoint_address = inter->endpoint[0].bEndpointAddress;
               goto found;
             }
            }
   }
  }

 found:
  libusb_free_device_list(devs, 1);

  return keyboard;
}
```

```
struct libusb_device_handle *openxbox(uint8_t *endpoint_address) {
  libusb_device *devs;
  struct libusb_device_handle *xbox = libusb_open_device_with_vid_pid(NULL, 0x045E, 0x028E);

  return xbox;
}



static int display_xbox_status(libusb_device_handle *handle)
{
        int r;
        uint8_t input_report[20];
        printf("\nReading XBox Input Report...\n");
        libusb_control_transfer(handle,
LIBUSB_ENDPOINT_IN|LIBUSB_REQUEST_TYPE_CLASS|LIBUSB_RECIPIENT_INTERFACE,
                HID_GET_REPORT, (HID_REPORT_TYPE_INPUT<<8)|0x00, 0, input_report, 20,
1000);
        printf("   D-pad: %02X\n", input_report[2]&0x0F);
        printf("        Start:%d, Back:%d, Left Stick Press:%d, Right Stick Press:%d\n",
B(input_report[2]&0x10), B(input_report[2]&0x20),
                B(input_report[2]&0x40), B(input_report[2]&0x80));
        // A, B, X, Y, Black, White are pressure sensitive
        printf("   A:%d, B:%d, X:%d, Y:%d, White:%d, Black:%d\n", input_report[4], input_report[5],
                input_report[6], input_report[7], input_report[9], input_report[8]);
        printf("   Left Trigger: %d, Right Trigger: %d\n", input_report[10], input_report[11]);
        printf("   Left Analog (X,Y): (%d,%d)\n", (int16_t)((input_report[13]<<8)|input_report[12]),
                (int16_t)((input_report[15]<<8)|input_report[14]));
        printf("   Right Analog (X,Y): (%d,%d)\n", (int16_t)((input_report[17]<<8)|input_report[16]),
                (int16_t)((input_report[19]<<8)|input_report[18]));
        return 0;
}
```

Usbkeyboard.h:

```
#ifndef _USBKEYBOARD_H
#define _USBKEYBOARD_H

#include <libusb-1.0/libusb.h>




#define USB_HID_KEYBOARD_PROTOCOL 1

/* Modifier bits */
#define USB_LCTRL  (1 << 0)
#define USB_LSHIFT (1 << 1)
#define USB_LALT   (1 << 2)
#define USB_LGUI   (1 << 3)
#define USB_RCTRL  (1 << 4)
#define USB_RSHIFT (1 << 5)
#define USB_RALT   (1 << 6)
#define USB_RGUI   (1 << 7)

struct usb_keyboard_packet {
  uint8_t modifiers;
  uint8_t reserved;
  uint8_t keycode[6];
};

/* Find and open a USB keyboard device.  Argument should point to
   space to store an endpoint address.  Returns NULL if no keyboard
   device was found. */
extern struct libusb_device_handle *openkeyboard(uint8_t *);

#endif
```

Vga_led.c:

```
/*
 * Device driver for the VGA LED Emulator
 *
 * A Platform device implemented using the misc subsystem
 *
 * Stephen A. Edwards
 * Columbia University
 *
 * References:
 * Linux source: Documentation/driver-model/platform.txt
 *               drivers/misc/arm-charlcd.c
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
 *
 * "make" to build
 * insmod vga_led.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree vga_led.c
 */

#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "vga_led.h"

#define DRIVER_NAME "vga_led"

/*
 * Information about our device
 */
struct vga_led_dev {
	struct resource res; /* Resource: our registers */
	void __iomem *virtbase; /* Where registers can be accessed in memory */
//	u8 segments[VGA_LED_DIGITS];
```

```
	u8 data[0xff];
} dev;

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up

static void write_digit(int digit, u8 segments)
{
	iowrite8(segments, dev.virtbase + digit);
	dev.segments[digit] = segments;
} */



static void write_data(unsigned int address,u8 data)
{
	iowrite8(data,dev.virtbase + address);//write to sepecific address.
	dev.data[address] = data; // copy to vertbase
}

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static long vga_led_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
	vga_led_arg_t vla;
	switch (cmd) {
	case VGA_LED_WRITE_DIGIT:
		if (copy_from_user(&vla, (vga_led_arg_t *) arg,
				   sizeof(vga_led_arg_t)))
			return -EACCES;
//		if (vla.digit > 8)
//			return -EINVAL;
		write_data(vla.address, vla.data);
		break;
	default:
		return -EINVAL;
	}

	return 0;
}

/* The operations our device knows how to do */
static const struct file_operations vga_led_fops = {
	.owner		= THIS_MODULE,
	.unlocked_ioctl = vga_led_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice vga_led_misc_device = {
	.minor		= MISC_DYNAMIC_MINOR,
	.name		= DRIVER_NAME,
	.fops		= &vga_led_fops,
};
```

```
/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init vga_led_probe(struct platform_device *pdev)
{

        int i, ret;
        /* Register ourselves as a misc device: creates /dev/vga_led */
        ret = misc_register(&vga_led_misc_device);

        /* Get the address of our registers from the device tree */
        ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
        if (ret) {
                ret = -ENOENT;
                goto out_deregister;
        }

        /* Make sure we can use these registers */
        if (request_mem_region(dev.res.start, resource_size(&dev.res),
                                DRIVER_NAME) == NULL) {
                ret = -EBUSY;
                goto out_deregister;
        }

        /* Arrange access to our registers */
        dev.virtbase = of_iomap(pdev->dev.of_node, 0);
        if (dev.virtbase == NULL) {
                ret = -ENOMEM;
```

```
		goto out_release_mem_region;
	}

	/* Display a welcome message */
	for (i = 0; i <= 0xff; i++)
		write_data(i, 0x00);

	return 0;

out_release_mem_region:
	release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
	misc_deregister(&vga_led_misc_device);
	return ret;
}

/* Clean-up code: release resources */
static int vga_led_remove(struct platform_device *pdev)
{
	iounmap(dev.virtbase);
	release_mem_region(dev.res.start, resource_size(&dev.res));
	misc_deregister(&vga_led_misc_device);
	return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id vga_led_of_match[] = {
	{ .compatible = "altr,vga_led" },
        {},
};
MODULE_DEVICE_TABLE(of, vga_led_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver vga_led_driver = {
        .driver  = {
                .name   = DRIVER_NAME,
                .owner = THIS_MODULE,
                .of_match_table = of_match_ptr(vga_led_of_match),
        },
        .remove         = __exit_p(vga_led_remove),
};

/* Called when the module is loaded: set things up */
static int __init vga_led_init(void)
{
        pr_info(DRIVER_NAME ": init\n");
        return platform_driver_probe(&vga_led_driver, vga_led_probe);
}

/* Called when the module is unloaded: release resources */
static void __exit vga_led_exit(void)
{
        platform_driver_unregister(&vga_led_driver);
        pr_info(DRIVER_NAME ": exit\n");
}
```

```
module_init(vga_led_init);
module_exit(vga_led_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("VGA 7-segment LED Emulator");
```

Vga_led.h:

```
#ifndef _VGA_LED_H
#define _VGA_LED_H

#include <linux/ioctl.h>

#define VGA_LED_DIGITS 8

typedef struct {
  //unsigned char digit;    /* 0, 1, .. , VGA_LED_DIGITS - 1 */
  //unsigned char segments; /* LSB is segment a, MSB is decimal point */
  unsigned int address;
  unsigned int data;
} vga_led_arg_t;

#define VGA_LED_MAGIC 'q'

/* ioctls and their arguments */
#define VGA_LED_WRITE_DIGIT _IOW(VGA_LED_MAGIC, 1, vga_led_arg_t *)
#define VGA_LED_READ_DIGIT  _IOWR(VGA_LED_MAGIC, 2, vga_led_arg_t *)

#endif
```

Xusb.c:

```
/*
 * xusb: Generic USB test program
 * Copyright © 2009-2012 Pete Batard <pete@akeo.ie>
 * Contributions to Mass Storage by Alan Stern.
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
 */

#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>

#include <libusb-1.0/libusb.h>
```

```
#if defined(_WIN32)
#define msleep(msecs) Sleep(msecs)
#else
#include <unistd.h>
#define msleep(msecs) usleep(1000*msecs)
#endif

#if defined(_MSC_VER)
#define snprintf _snprintf
#define putenv _putenv
#endif

#if !defined(bool)
#define bool int
#endif
#if !defined(true)
#define true (1 == 1)
#endif
#if !defined(false)
#define false (!true)
#endif

// Future versions of libusb will use usb_interface instead of interface
// in libusb_config_descriptor => catter for that
#define usb_interface interface

// Global variables
static bool binary_dump = false;
static bool extra_info = false;
static bool force_device_request = false;        // For WCID descriptor queries
static const char* binary_name = NULL;

static int perr(char const *format, ...)
{
        va_list args;
        int r;

        va_start (args, format);
        r = vfprintf(stderr, format, args);
        va_end(args);

        return r;
}

#define ERR_EXIT(errcode) do { perr("   %s\n", libusb_strerror((enum libusb_error)errcode)); return
-1; } while (0)
#define CALL_CHECK(fcall) do { r=fcall; if (r < 0) ERR_EXIT(r); } while (0);
#define B(x) (((x)!=0)?1:0)
#define be_to_int32(buf) (((buf)[0]<<24)|((buf)[1]<<16)|((buf)[2]<<8)|(buf)[3])

#define RETRY_MAX                  5
#define REQUEST_SENSE_LENGTH       0x12
#define INQUIRY_LENGTH             0x24
#define READ_CAPACITY_LENGTH       0x08

// HID Class-Specific Requests values. See section 7.2 of the HID specifications
#define HID_GET_REPORT             0x01
```

```
#define HID_GET_IDLE              0x02
#define HID_GET_PROTOCOL          0x03
#define HID_SET_REPORT            0x09
#define HID_SET_IDLE              0x0A
#define HID_SET_PROTOCOL          0x0B
#define HID_REPORT_TYPE_INPUT     0x01
#define HID_REPORT_TYPE_OUTPUT    0x02
#define HID_REPORT_TYPE_FEATURE   0x03

// Mass Storage Requests values. See section 3 of the Bulk-Only Mass Storage Class specifications
#define BOMS_RESET                0xFF
#define BOMS_GET_MAX_LUN          0xFE

// Section 5.1: Command Block Wrapper (CBW)
struct command_block_wrapper {
        uint8_t dCBWSignature[4];
        uint32_t dCBWTag;
        uint32_t dCBWDataTransferLength;
        uint8_t bmCBWFlags;
        uint8_t bCBWLUN;
        uint8_t bCBWCBLength;
        uint8_t CBWCB[16];
};

// Section 5.2: Command Status Wrapper (CSW)
struct command_status_wrapper {
        uint8_t dCSWSignature[4];
        uint32_t dCSWTag;
        uint32_t dCSWDataResidue;
        uint8_t bCSWStatus;
};

static uint8_t cdb_length[256] = {
//      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
        06,06,06,06,06,06,06,06,06,06,06,06,06,06,06,06,  //  0
        06,06,06,06,06,06,06,06,06,06,06,06,06,06,06,06,  //  1
        10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,  //  2
        10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,  //  3
        10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,  //  4
        10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,  //  5
        00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,  //  6
        00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,  //  7
        16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,  //  8
        16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,  //  9
        12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,  //  A
        12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,  //  B
        00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,  //  C
        00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,  //  D
        00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,  //  E
        00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,  //  F
};

static enum test_type {
        USE_GENERIC,
        USE_PS3,
        USE_XBOX,
        USE_SCSI,
        USE_HID,
} test_mode;
static uint16_t VID, PID;

static void display_buffer_hex(unsigned char *buffer, unsigned size)
{
        unsigned i, j, k;

        for (i=0; i<size; i+=16) {
                printf("\n  %08x  ", i);
                for(j=0,k=0; k<16; j++,k++) {
                        if (i+j < size) {
                                printf("%02x", buffer[i+j]);
                        } else {
                                printf("  ");
                        }
                        printf(" ");
                }
                printf(" ");
                for(j=0,k=0; k<16; j++,k++) {
                        if (i+j < size) {
                                if ((buffer[i+j] < 32) || (buffer[i+j] > 126)) {
                                        printf(".");
                                } else {
                                        printf("%c", buffer[i+j]);
                                }
                        }
                }
        }
        printf("\n" );
```

```
}

static char* uuid_to_string(const uint8_t* uuid)
{
	static char uuid_string[40];
	if (uuid == NULL) return NULL;
	sprintf(uuid_string,
"{%02x%02x%02x%02x-%02x%02x-%02x%02x-%02x%02x-%02x%02x%02x%02x%02x%02x}
",
		uuid[0], uuid[1], uuid[2], uuid[3], uuid[4], uuid[5], uuid[6], uuid[7],
		uuid[8], uuid[9], uuid[10], uuid[11], uuid[12], uuid[13], uuid[14], uuid[15]);
	return uuid_string;
}

// The PS3 Controller is really a HID device that got its HID Report Descriptors
// removed by Sony
static int display_ps3_status(libusb_device_handle *handle)
{
	int r;
	uint8_t input_report[49];
	uint8_t master_bt_address[8];
	uint8_t device_bt_address[18];

	// Get the controller's bluetooth address of its master device
	CALL_CHECK(libusb_control_transfer(handle,
LIBUSB_ENDPOINT_IN|LIBUSB_REQUEST_TYPE_CLASS|LIBUSB_RECIPIENT_INTERFACE,
		HID_GET_REPORT, 0x03f5, 0, master_bt_address, sizeof(master_bt_address), 100));
	printf("\nMaster's bluetooth address: %02X:%02X:%02X:%02X:%02X:%02X\n",
master_bt_address[2], master_bt_address[3],
		master_bt_address[4], master_bt_address[5], master_bt_address[6],
master_bt_address[7]);

	// Get the controller's bluetooth address
	CALL_CHECK(libusb_control_transfer(handle,
LIBUSB_ENDPOINT_IN|LIBUSB_REQUEST_TYPE_CLASS|LIBUSB_RECIPIENT_INTERFACE,
		HID_GET_REPORT, 0x03f2, 0, device_bt_address, sizeof(device_bt_address), 100));
	printf("\nMaster's bluetooth address: %02X:%02X:%02X:%02X:%02X:%02X\n",
device_bt_address[4], device_bt_address[5],
		device_bt_address[6], device_bt_address[7], device_bt_address[8],
device_bt_address[9]);

	// Get the status of the controller's buttons via its HID report
	printf("\nReading PS3 Input Report...\n");
	CALL_CHECK(libusb_control_transfer(handle,
LIBUSB_ENDPOINT_IN|LIBUSB_REQUEST_TYPE_CLASS|LIBUSB_RECIPIENT_INTERFACE,
		HID_GET_REPORT, (HID_REPORT_TYPE_INPUT<<8)|0x01, 0, input_report,
sizeof(input_report), 1000));
	switch(input_report[2]){	/** Direction pad plus start, select, and joystick buttons */
		case 0x01:
			printf("\tSELECT pressed\n");
			break;
		case 0x02:
			printf("\tLEFT 3 pressed\n");
			break;
		case 0x04:
			printf("\tRIGHT 3 pressed\n");
			break;
		case 0x08:
```

```
                printf("\tSTART presed\n");
                break;
        case 0x10:
                printf("\tUP pressed\n");
                break;
        case 0x20:
                printf("\tRIGHT pressed\n");
                break;
        case 0x40:
                printf("\tDOWN pressed\n");
                break;
        case 0x80:
                printf("\tLEFT pressed\n");
                break;
}
switch(input_report[3]){        /** Shapes plus top right and left buttons */
        case 0x01:
                printf("\tLEFT 2 pressed\n");
                break;
        case 0x02:
                printf("\tRIGHT 2 pressed\n");
                break;
        case 0x04:
                printf("\tLEFT 1 pressed\n");
                break;
        case 0x08:
                printf("\tRIGHT 1 presed\n");
                break;
        case 0x10:
                printf("\tTRIANGLE pressed\n");
                break;
            case 0x20:
                printf("\tCIRCLE pressed\n");
                break;
            case 0x40:
                printf("\tCROSS pressed\n");
                break;
            case 0x80:
                printf("\tSQUARE pressed\n");
                break;
        }
        printf("\tPS button: %d\n", input_report[4]);
        printf("\tLeft Analog (X,Y): (%d,%d)\n", input_report[6], input_report[7]);
        printf("\tRight Analog (X,Y): (%d,%d)\n", input_report[8], input_report[9]);
        printf("\tL2 Value: %d\tR2 Value: %d\n", input_report[18], input_report[19]);
        printf("\tL1 Value: %d\tR1 Value: %d\n", input_report[20], input_report[21]);
        printf("\tRoll (x axis): %d Yaw (y axis): %d Pitch (z axis) %d\n",
                //(((input_report[42] + 128) % 256) - 128),
                (int8_t)(input_report[42]),
                (int8_t)(input_report[44]),
                (int8_t)(input_report[46]));
        printf("\tAcceleration: %d\n\n", (int8_t)(input_report[48]));
        return 0;
}
// The XBOX Controller is really a HID device that got its HID Report Descriptors
// removed by Microsoft.
// Input/Output reports described at http://euc.jp/periphs/xbox-controller.ja.html
static int display_xbox_status(libusb_device_handle *handle)
```

```
{
	int r;
	uint8_t input_report[20];
	printf("\nReading XBox Input Report...\n");
	CALL_CHECK(libusb_control_transfer(handle,
LIBUSB_ENDPOINT_IN|LIBUSB_REQUEST_TYPE_CLASS|LIBUSB_RECIPIENT_INTERFACE,
		HID_GET_REPORT, (HID_REPORT_TYPE_INPUT<<8)|0x00, 0, input_report, 20,
1000));
	printf("   D-pad: %02X\n", input_report[2]&0x0F);
	printf("        Start:%d, Back:%d, Left Stick Press:%d, Right Stick Press:%d\n",
B(input_report[2]&0x10), B(input_report[2]&0x20),
		B(input_report[2]&0x40), B(input_report[2]&0x80));
	// A, B, X, Y, Black, White are pressure sensitive
	printf("   A:%d, B:%d, X:%d, Y:%d, White:%d, Black:%d\n", input_report[4], input_report[5],
		input_report[6], input_report[7], input_report[9], input_report[8]);
	printf("   Left Trigger: %d, Right Trigger: %d\n", input_report[10], input_report[11]);
	printf("   Left Analog (X,Y): (%d,%d)\n", (int16_t)((input_report[13]<<8)|input_report[12]),
		(int16_t)((input_report[15]<<8)|input_report[14]));
	printf("   Right Analog (X,Y): (%d,%d)\n", (int16_t)((input_report[17]<<8)|input_report[16]),
		(int16_t)((input_report[19]<<8)|input_report[18]));
	return 0;
}

static int set_xbox_actuators(libusb_device_handle *handle, uint8_t left, uint8_t right)
{
	int r;
	uint8_t output_report[6];

	printf("\nWriting XBox Controller Output Report...\n");
	memset(output_report, 0, sizeof(output_report));
	output_report[1] = sizeof(output_report);
	output_report[3] = left;
	output_report[5] = right;

	CALL_CHECK(libusb_control_transfer(handle,
LIBUSB_ENDPOINT_OUT|LIBUSB_REQUEST_TYPE_CLASS|LIBUSB_RECIPIENT_INTERFACE,
		HID_SET_REPORT, (HID_REPORT_TYPE_OUTPUT<<8)|0x00, 0, output_report, 06,
1000));
	return 0;
}

static int send_mass_storage_command(libusb_device_handle *handle, uint8_t endpoint, uint8_t
lun,
	uint8_t *cdb, uint8_t direction, int data_length, uint32_t *ret_tag)
{
	static uint32_t tag = 1;
	uint8_t cdb_len;
	int i, r, size;
	struct command_block_wrapper cbw;

	if (cdb == NULL) {
		return -1;
	}

	if (endpoint & LIBUSB_ENDPOINT_IN) {
		perr("send_mass_storage_command: cannot send command on IN endpoint\n");
		return -1;
	}

	cdb_len = cdb_length[cdb[0]];
	if ((cdb_len == 0) || (cdb_len > sizeof(cbw.CBWCB))) {
		perr("send_mass_storage_command: don't know how to handle this command
(%02X, length %d)\n",
			cdb[0], cdb_len);
		return -1;
	}

	memset(&cbw, 0, sizeof(cbw));
	cbw.dCBWSignature[0] = 'U';
	cbw.dCBWSignature[1] = 'S';
	cbw.dCBWSignature[2] = 'B';
	cbw.dCBWSignature[3] = 'C';
	*ret_tag = tag;
	cbw.dCBWTag = tag++;
	cbw.dCBWDataTransferLength = data_length;
	cbw.bmCBWFlags = direction;
	cbw.bCBWLUN = lun;
	// Subclass is 1 or 6 => cdb_len
	cbw.bCBWCBLength = cdb_len;
	memcpy(cbw.CBWCB, cdb, cdb_len);

	i = 0;
	do {
		// The transfer length must always be exactly 31 bytes.
		r = libusb_bulk_transfer(handle, endpoint, (unsigned char*)&cbw, 31, &size, 1000);
		if (r == LIBUSB_ERROR_PIPE) {
				libusb_clear_halt(handle, endpoint);
			}
			i++;
		} while ((r == LIBUSB_ERROR_PIPE) && (i<RETRY_MAX));
		if (r != LIBUSB_SUCCESS) {
			perr("   send_mass_storage_command: %s\n", libusb_strerror((enum libusb_error)r));
			return -1;
		}

		printf("   sent %d CDB bytes\n", cdb_len);
		return 0;
}

static int get_mass_storage_status(libusb_device_handle *handle, uint8_t endpoint, uint32_t
expected_tag)
{
		int i, r, size;
		struct command_status_wrapper csw;

		// The device is allowed to STALL this transfer. If it does, you have to
		// clear the stall and try again.
		i = 0;
		do {
			r = libusb_bulk_transfer(handle, endpoint, (unsigned char*)&csw, 13, &size, 1000);
			if (r == LIBUSB_ERROR_PIPE) {
				libusb_clear_halt(handle, endpoint);
			}
			i++;
		} while ((r == LIBUSB_ERROR_PIPE) && (i<RETRY_MAX));
```

```
	if (r != LIBUSB_SUCCESS) {
		perr("   get_mass_storage_status: %s\n", libusb_strerror((enum libusb_error)r));
		return -1;
	}
	if (size != 13) {
		perr("   get_mass_storage_status: received %d bytes (expected 13)\n", size);
		return -1;
	}
	if (csw.dCSWTag != expected_tag) {
		perr("   get_mass_storage_status: mismatched tags (expected %08X, received
%08X)\n",
			expected_tag, csw.dCSWTag);
		return -1;
	}
	// For this test, we ignore the dCSWSignature check for validity...
	printf("   Mass Storage Status: %02X (%s)\n", csw.bCSWStatus,
csw.bCSWStatus?"FAILED":"Success");
	if (csw.dCSWTag != expected_tag)
		return -1;
	if (csw.bCSWStatus) {
		// REQUEST SENSE is appropriate only if bCSWStatus is 1, meaning that the
		// command failed somehow.  Larger values (2 in particular) mean that
		// the command couldn't be understood.
		if (csw.bCSWStatus == 1)
			return -2;	// request Get Sense
		else
			return -1;
	}
	// In theory we also should check dCSWDataResidue.  But lots of devices
	// set it wrongly.
	return 0;
}

static void get_sense(libusb_device_handle *handle, uint8_t endpoint_in, uint8_t endpoint_out)
{
	uint8_t cdb[16];		// SCSI Command Descriptor Block
	uint8_t sense[18];
	uint32_t expected_tag;
	int size;
	int rc;

	// Request Sense
	printf("Request Sense:\n");
	memset(sense, 0, sizeof(sense));
	memset(cdb, 0, sizeof(cdb));
	cdb[0] = 0x03;	// Request Sense
	cdb[4] = REQUEST_SENSE_LENGTH;

	send_mass_storage_command(handle, endpoint_out, 0, cdb, LIBUSB_ENDPOINT_IN,
REQUEST_SENSE_LENGTH, &expected_tag);
	rc = libusb_bulk_transfer(handle, endpoint_in, (unsigned char*)&sense,
REQUEST_SENSE_LENGTH, &size, 1000);
	if (rc < 0)
	{
		printf("libusb_bulk_transfer failed: %s\n", libusb_error_name(rc));
		return;
	}
	printf("   received %d bytes\n", size);

	if ((sense[0] != 0x70) && (sense[0] != 0x71)) {
		perr("   ERROR No sense data\n");
	} else {
		perr("   ERROR Sense: %02X %02X %02X\n", sense[2]&0x0F, sense[12], sense[13]);
	}
	// Strictly speaking, the get_mass_storage_status() call should come
	// before these perr() lines.  If the status is nonzero then we must
	// assume there's no data in the buffer.  For xusb it doesn't matter.
	get_mass_storage_status(handle, endpoint_in, expected_tag);
}

// Mass Storage device to test bulk transfers (non destructive test)
static int test_mass_storage(libusb_device_handle *handle, uint8_t endpoint_in, uint8_t
endpoint_out)
{
	int r, size;
	uint8_t lun;
	uint32_t expected_tag;
	uint32_t i, max_lba, block_size;
	double device_size;
	uint8_t cdb[16];	// SCSI Command Descriptor Block
	uint8_t buffer[64];
	char vid[9], pid[9], rev[5];
	unsigned char *data;
	FILE *fd;

	printf("Reading Max LUN:\n");
	r = libusb_control_transfer(handle,
LIBUSB_ENDPOINT_IN|LIBUSB_REQUEST_TYPE_CLASS|LIBUSB_RECIPIENT_INTERFACE,
		BOMS_GET_MAX_LUN, 0, 0, &lun, 1, 1000);
	// Some devices send a STALL instead of the actual value.
	// In such cases we should set lun to 0.
	if (r == 0) {
		lun = 0;
	} else if (r < 0) {
		perr("   Failed: %s", libusb_strerror((enum libusb_error)r));
	}
	printf("   Max LUN = %d\n", lun);

	// Send Inquiry
	printf("Sending Inquiry:\n");
	memset(buffer, 0, sizeof(buffer));
	memset(cdb, 0, sizeof(cdb));
	cdb[0] = 0x12;  // Inquiry
	cdb[4] = INQUIRY_LENGTH;

	send_mass_storage_command(handle, endpoint_out, lun, cdb, LIBUSB_ENDPOINT_IN,
INQUIRY_LENGTH, &expected_tag);
	CALL_CHECK(libusb_bulk_transfer(handle, endpoint_in, (unsigned char*)&buffer,
INQUIRY_LENGTH, &size, 1000));
	printf("   received %d bytes\n", size);
	// The following strings are not zero terminated
	for (i=0; i<8; i++) {
		vid[i] = buffer[8+i];
		pid[i] = buffer[16+i];
		rev[i/2] = buffer[32+i/2];		// instead of another loop
	}
	vid[8] = 0;
	pid[8] = 0;
	rev[4] = 0;
	printf("   VID:PID:REV \"%8s\":\"%8s\":\"%4s\"\n", vid, pid, rev);
	if (get_mass_storage_status(handle, endpoint_in, expected_tag) == -2) {
		get_sense(handle, endpoint_in, endpoint_out);
	}

	// Read capacity
	printf("Reading Capacity:\n");
	memset(buffer, 0, sizeof(buffer));
	memset(cdb, 0, sizeof(cdb));
	cdb[0] = 0x25;  // Read Capacity

	send_mass_storage_command(handle, endpoint_out, lun, cdb, LIBUSB_ENDPOINT_IN,
READ_CAPACITY_LENGTH, &expected_tag);
	CALL_CHECK(libusb_bulk_transfer(handle, endpoint_in, (unsigned char*)&buffer,
READ_CAPACITY_LENGTH, &size, 1000));
	printf("   received %d bytes\n", size);
	max_lba = be_to_int32(&buffer[0]);
	block_size = be_to_int32(&buffer[4]);
	device_size = ((double)(max_lba+1))*block_size/(1024*1024*1024);
	printf("   Max LBA: %08X, Block Size: %08X (%.2f GB)\n", max_lba, block_size, device_size);
	if (get_mass_storage_status(handle, endpoint_in, expected_tag) == -2) {
		get_sense(handle, endpoint_in, endpoint_out);
	}

	// coverity[tainted_data]
	data = (unsigned char*) calloc(1, block_size);
	if (data == NULL) {
		perr("   unable to allocate data buffer\n");
		return -1;
	}

	// Send Read
	printf("Attempting to read %d bytes:\n", block_size);
	memset(cdb, 0, sizeof(cdb));

	cdb[0] = 0x28;  // Read(10)
	cdb[8] = 0x01;  // 1 block

	send_mass_storage_command(handle,   endpoint_out,   lun,   cdb,   LIBUSB_ENDPOINT_IN,
block_size, &expected_tag);
	libusb_bulk_transfer(handle, endpoint_in, data, block_size, &size, 5000);
	printf("   READ: received %d bytes\n", size);
	if (get_mass_storage_status(handle, endpoint_in, expected_tag) == -2) {
		get_sense(handle, endpoint_in, endpoint_out);
	} else {
		display_buffer_hex(data, size);
		if ((binary_dump) && ((fd = fopen(binary_name, "w")) != NULL)) {
			if (fwrite(data, 1, (size_t)size, fd) != (unsigned int)size) {
				perr("   unable to write binary data\n");
			}
			fclose(fd);
		}
	}
	free(data);
```

```
        return 0;
}

// HID
static int get_hid_record_size(uint8_t *hid_report_descriptor, int size, int type)
{
        uint8_t i, j = 0;
        uint8_t offset;
        int record_size[3] = {0, 0, 0};
        int nb_bits = 0, nb_items = 0;
        bool found_record_marker;

        found_record_marker = false;
        for (i = hid_report_descriptor[0]+1; i < size; i += offset) {
                offset = (hid_report_descriptor[i]&0x03) + 1;
                if (offset == 4)
                        offset = 5;
                switch (hid_report_descriptor[i] & 0xFC) {
                case 0x74:      // bitsize
                        nb_bits = hid_report_descriptor[i+1];
                        break;
                case 0x94:      // count
                        nb_items = 0;
                        for (j=1; j<offset; j++) {
                                nb_items = ((uint32_t)hid_report_descriptor[i+j]) << (8*(j-1));
                        }
                        break;
                case 0x80:      // input
                        found_record_marker = true;
                        j = 0;
                        break;
                case 0x90:      // output
                        found_record_marker = true;
                        j = 1;
                        break;
                case 0xb0:      // feature
                        found_record_marker = true;
                        j = 2;
                        break;
                case 0xC0:      // end of collection
                        nb_items = 0;
                        nb_bits = 0;
                        break;
                default:
                        continue;
                }
                if (found_record_marker) {
                        found_record_marker = false;
                        record_size[j] += nb_items*nb_bits;
                }
        }
        if ((type < HID_REPORT_TYPE_INPUT) || (type > HID_REPORT_TYPE_FEATURE)) {
                return 0;
        } else {
                return (record_size[type - HID_REPORT_TYPE_INPUT]+7)/8;
        }
}
```

```
static int test_hid(libusb_device_handle *handle, uint8_t endpoint_in)
{
	int r, size, descriptor_size;
	uint8_t hid_report_descriptor[256];
	uint8_t *report_buffer;
	FILE *fd;

	printf("\nReading HID Report Descriptors:\n");
	descriptor_size                    =                    libusb_control_transfer(handle,
LIBUSB_ENDPOINT_IN|LIBUSB_REQUEST_TYPE_STANDARD|LIBUSB_RECIPIENT_INTERFACE,
		LIBUSB_REQUEST_GET_DESCRIPTOR,          LIBUSB_DT_REPORT<<8,          0,
hid_report_descriptor, sizeof(hid_report_descriptor), 1000);
	if (descriptor_size < 0) {
		printf("   Failed\n");
		return -1;
	}
	display_buffer_hex(hid_report_descriptor, descriptor_size);
	if ((binary_dump) && ((fd = fopen(binary_name, "w")) != NULL)) {
		if (fwrite(hid_report_descriptor, 1, descriptor_size, fd) != descriptor_size) {
			printf("   Error writing descriptor to file\n");
		}
		fclose(fd);
	}

	size          =          get_hid_record_size(hid_report_descriptor,          descriptor_size,
HID_REPORT_TYPE_FEATURE);
	if (size <= 0) {
		printf("\nSkipping Feature Report readout (None detected)\n");
	} else {
		report_buffer = (uint8_t*) calloc(size, 1);
		if (report_buffer == NULL) {
			return -1;
		}

		printf("\nReading Feature Report (length %d)...\n", size);
		r                              =                              libusb_control_transfer(handle,
LIBUSB_ENDPOINT_IN|LIBUSB_REQUEST_TYPE_CLASS|LIBUSB_RECIPIENT_INTERFACE,
			HID_GET_REPORT,  (HID_REPORT_TYPE_FEATURE<<8)|0,  0,  report_buffer,
(uint16_t)size, 5000);
		if (r >= 0) {
			display_buffer_hex(report_buffer, size);
		} else {
			switch(r) {
			case LIBUSB_ERROR_NOT_FOUND:
				printf("   No Feature Report available for this device\n");
				break;
			case LIBUSB_ERROR_PIPE:
				printf("   Detected stall - resetting pipe...\n");
				libusb_clear_halt(handle, 0);
				break;
			default:
				printf("   Error: %s\n", libusb_strerror((enum libusb_error)r));
				break;
			}
		}
		free(report_buffer);
	}
	size = get_hid_record_size(hid_report_descriptor, descriptor_size, 
HID_REPORT_TYPE_INPUT);
	if (size <= 0) {
		printf("\nSkipping Input Report readout (None detected)\n");
	} else {
		report_buffer = (uint8_t*) calloc(size, 1);
		if (report_buffer == NULL) {
			return -1;
		}

		printf("\nReading Input Report (length %d)...\n", size);
		r = libusb_control_transfer(handle, 
LIBUSB_ENDPOINT_IN|LIBUSB_REQUEST_TYPE_CLASS|LIBUSB_RECIPIENT_INTERFACE,
			HID_GET_REPORT, (HID_REPORT_TYPE_INPUT<<8)|0x00, 0, report_buffer, 
(uint16_t)size, 5000);
		if (r >= 0) {
			display_buffer_hex(report_buffer, size);
		} else {
			switch(r) {
			case LIBUSB_ERROR_TIMEOUT:
				printf("   Timeout! Please make sure you act on the device within the 
5 seconds allocated...\n");
				break;
			case LIBUSB_ERROR_PIPE:
				printf("   Detected stall - resetting pipe...\n");
				libusb_clear_halt(handle, 0);
				break;
			default:
```

```
                        printf("   Error: %s\n", libusb_strerror((enum libusb_error)r));
                        break;
                    }
                }

                // Attempt a bulk read from endpoint 0 (this should just return a raw input report)
                printf("\nTesting interrupt read using endpoint %02X...\n", endpoint_in);
                r = libusb_interrupt_transfer(handle, endpoint_in, report_buffer, size, &size, 5000);
                if (r >= 0) {
                    display_buffer_hex(report_buffer, size);
                } else {
                    printf("   %s\n", libusb_strerror((enum libusb_error)r));
                }

                free(report_buffer);
        }
        return 0;
}

// Read the MS WinUSB Feature Descriptors, that are used on Windows 8 for automated driver
installation
static void read_ms_winsub_feature_descriptors(libusb_device_handle *handle, uint8_t bRequest,
int iface_number)
{
#define MAX_OS_FD_LENGTH 256
        int i, r;
        uint8_t os_desc[MAX_OS_FD_LENGTH];
        uint32_t length;
        void* le_type_punning_IS_fine;
```

```
	struct {
		const char* desc;
		uint8_t recipient;
		uint16_t index;
		uint16_t header_size;
	} os_fd[2] = {
		{"Extended Compat ID", LIBUSB_RECIPIENT_DEVICE, 0x0004, 0x10},
		{"Extended Properties", LIBUSB_RECIPIENT_INTERFACE, 0x0005, 0x0A}
	};

	if (iface_number < 0) return;
	// WinUSB has a limitation that forces wIndex to the interface number when issuing
	// an Interface Request. To work around that, we can force a Device Request for
	// the Extended Properties, assuming the device answers both equally.
	if (force_device_request)
		os_fd[1].recipient = LIBUSB_RECIPIENT_DEVICE;

	for (i=0; i<2; i++) {
		printf("\nReading %s OS Feature Descriptor (wIndex = 0x%04d):\n", os_fd[i].desc,
os_fd[i].index);

		// Read the header part
		r = libusb_control_transfer(handle,
(uint8_t)(LIBUSB_ENDPOINT_IN|LIBUSB_REQUEST_TYPE_VENDOR|os_fd[i].recipient),
			bRequest, (uint16_t)(((iface_number)<< 8)|0x00), os_fd[i].index, os_desc,
os_fd[i].header_size, 1000);
		if (r < os_fd[i].header_size) {
			perr("   Failed: %s", (r<0)?libusb_strerror((enum libusb_error)r):"header size
is too small");
```

```
				return;
			}
			le_type_punning_IS_fine = (void*)os_desc;
			length = *((uint32_t*)le_type_punning_IS_fine);
			if (length > MAX_OS_FD_LENGTH) {
				length = MAX_OS_FD_LENGTH;
			}

			// Read the full feature descriptor
			r                                    =                                    libusb_control_transfer(handle,
(uint8_t)(LIBUSB_ENDPOINT_IN|LIBUSB_REQUEST_TYPE_VENDOR|os_fd[i].recipient),
				bRequest,   (uint16_t)(((iface_number)<<   8)|0x00),   os_fd[i].index,   os_desc,
(uint16_t)length, 1000);
			if (r < 0) {
				perr("   Failed: %s", libusb_strerror((enum libusb_error)r));
				return;
			} else {
				display_buffer_hex(os_desc, r);
			}
		}
}



static int test_device(uint16_t vid, uint16_t pid)
{
		libusb_device_handle *handle;
		libusb_device *dev;
		uint8_t bus, port_path[8];
```

```
	struct libusb_bos_descriptor *bos_desc;
	struct libusb_config_descriptor *conf_desc;
	const struct libusb_endpoint_descriptor *endpoint;
	int i, j, k, r;
	int iface, nb_ifaces, first_iface = -1;
	struct libusb_device_descriptor dev_desc;
	const char* speed_name[5] = { "Unknown", "1.5 Mbit/s (USB LowSpeed)", "12 Mbit/s (USB
FullSpeed)",
		"480 Mbit/s (USB HighSpeed)", "5000 Mbit/s (USB SuperSpeed)"};
	char string[128];
	uint8_t string_index[3];	// indexes of the string descriptors
	uint8_t endpoint_in = 0, endpoint_out = 0;	// default IN and OUT endpoints

	printf("Opening device %04X:%04X...\n", vid, pid);
	handle = libusb_open_device_with_vid_pid(NULL, vid, pid);

	if (handle == NULL) {
		perr("  Failed.\n");
		return -1;
	}

	dev = libusb_get_device(handle);
	bus = libusb_get_bus_number(dev);
	if (extra_info) {
		r = libusb_get_port_numbers(dev, port_path, sizeof(port_path));
		if (r > 0) {
			printf("\nDevice properties:\n");
			printf("        bus number: %d\n", bus);
			printf("         port path: %d", port_path[0]);
```

```
                for (i=1; i<r; i++) {
                        printf("->%d", port_path[i]);
                }
                printf(" (from root hub)\n");
        }
        r = libusb_get_device_speed(dev);
        if ((r<0) || (r>4)) r=0;
        printf("           speed: %s\n", speed_name[r]);
    }

    printf("\nReading device descriptor:\n");
    CALL_CHECK(libusb_get_device_descriptor(dev, &dev_desc));
    printf("            length: %d\n", dev_desc.bLength);
    printf("      device class: %d\n", dev_desc.bDeviceClass);
    printf("               S/N: %d\n", dev_desc.iSerialNumber);
    printf("           VID:PID: %04X:%04X\n", dev_desc.idVendor, dev_desc.idProduct);
    printf("         bcdDevice: %04X\n", dev_desc.bcdDevice);
    printf("         iMan:iProd:iSer:   %d:%d:%d\n",   dev_desc.iManufacturer,   dev_desc.iProduct,
dev_desc.iSerialNumber);
    printf("          nb confs: %d\n", dev_desc.bNumConfigurations);
    // Copy the string descriptors for easier parsing
    string_index[0] = dev_desc.iManufacturer;
    string_index[1] = dev_desc.iProduct;
    string_index[2] = dev_desc.iSerialNumber;


    printf("\nReading first configuration descriptor:\n");
    CALL_CHECK(libusb_get_config_descriptor(dev, 0, &conf_desc));
    nb_ifaces = conf_desc->bNumInterfaces;
```

```
	printf("		nb interfaces: %d\n", nb_ifaces);
	if (nb_ifaces > 0)
		first_iface = conf_desc->usb_interface[0].altsetting[0].bInterfaceNumber;
	for (i=0; i<nb_ifaces; i++) {
		printf("		interface[%d]: id = %d\n", i,
			conf_desc->usb_interface[i].altsetting[0].bInterfaceNumber);
		for (j=0; j<conf_desc->usb_interface[i].num_altsetting; j++) {
			printf("interface[%d].altsetting[%d]: num endpoints = %d\n",
				i, j, conf_desc->usb_interface[i].altsetting[j].bNumEndpoints);
			printf("   Class.SubClass.Protocol: %02X.%02X.%02X\n",
				conf_desc->usb_interface[i].altsetting[j].bInterfaceClass,
				conf_desc->usb_interface[i].altsetting[j].bInterfaceSubClass,
				conf_desc->usb_interface[i].altsetting[j].bInterfaceProtocol);
			if	(	(conf_desc->usb_interface[i].altsetting[j].bInterfaceClass	==
LIBUSB_CLASS_MASS_STORAGE)
			  && ( (conf_desc->usb_interface[i].altsetting[j].bInterfaceSubClass == 0x01)
			  || (conf_desc->usb_interface[i].altsetting[j].bInterfaceSubClass == 0x06) )
			  && (conf_desc->usb_interface[i].altsetting[j].bInterfaceProtocol == 0x50) ) {
				// Mass storage devices that can use basic SCSI commands
				test_mode = USE_SCSI;
			}
			for (k=0; k<conf_desc->usb_interface[i].altsetting[j].bNumEndpoints; k++) {
				struct libusb_ss_endpoint_companion_descriptor *ep_comp = NULL;
				endpoint = &conf_desc->usb_interface[i].altsetting[j].endpoint[k];
				printf("				endpoint[%d].address:	%02X\n",	k,
endpoint->bEndpointAddress);
				// Use the first interrupt or bulk IN/OUT endpoints as default for
testing
				if ((endpoint->bmAttributes & LIBUSB_TRANSFER_TYPE_MASK) &
(LIBUSB_TRANSFER_TYPE_BULK | LIBUSB_TRANSFER_TYPE_INTERRUPT)) {
					if (endpoint->bEndpointAddress & LIBUSB_ENDPOINT_IN) {
						if (!endpoint_in)
							endpoint_in = endpoint->bEndpointAddress;
					} else {
						if (!endpoint_out)
							endpoint_out = endpoint->bEndpointAddress;
					}
				}
				printf("				max	packet	size:	%04X\n",
endpoint->wMaxPacketSize);
				printf("	polling interval: %02X\n", endpoint->bInterval);
				libusb_get_ss_endpoint_companion_descriptor(NULL,	endpoint,
&ep_comp);
			}
		}
	}
	libusb_free_config_descriptor(conf_desc);

	libusb_set_auto_detach_kernel_driver(handle, 1);
	for (iface = 0; iface < nb_ifaces; iface++)
	{
		printf("\nClaiming interface %d...\n", iface);
		r = libusb_claim_interface(handle, iface);
		if (r != LIBUSB_SUCCESS) {
			perr("   Failed.\n");
		}
	}
	printf("\nReading string descriptors:\n");
	for (i=0; i<3; i++) {
		if (string_index[i] == 0) {
			continue;
		}
		if (libusb_get_string_descriptor_ascii(handle, string_index[i], (unsigned char*)string, 
128) >= 0) {
			printf("   String (0x%02X): \"%s\"\n", string_index[i], string);
		}
	}
	// Read the OS String Descriptor
	if (libusb_get_string_descriptor_ascii(handle, 0xEE, (unsigned char*)string, 128) >= 0) {
		printf("   String (0x%02X): \"%s\"\n", 0xEE, string);
		// If this is a Microsoft OS String Descriptor,
		// attempt to read the WinUSB extended Feature Descriptors
		if (strncmp(string, "MSFT100", 7) == 0)
			read_ms_winsub_feature_descriptors(handle, string[7], first_iface);
	}

	switch(test_mode) {
	case USE_PS3:
		CALL_CHECK(display_ps3_status(handle));
		break;
	case USE_XBOX:
		CALL_CHECK(display_xbox_status(handle));
		CALL_CHECK(set_xbox_actuators(handle, 128, 222));
		msleep(2000);
		CALL_CHECK(set_xbox_actuators(handle, 0, 0));
		break;
	case USE_HID:
		test_hid(handle, endpoint_in);
		break;
	case USE_SCSI:
		CALL_CHECK(test_mass_storage(handle, endpoint_in, endpoint_out));
	case USE_GENERIC:
		break;
	}

	printf("\n");
	for (iface = 0; iface<nb_ifaces; iface++) {
		printf("Releasing interface %d...\n", iface);
		libusb_release_interface(handle, iface);
	}

	printf("Closing device...\n");
	libusb_close(handle);

	return 0;
}

int main(int argc, char** argv)
{
	bool show_help = false;
	bool debug_mode = false;
	const struct libusb_version* version;
	int j, r;
	size_t i, arglen;
	unsigned tmp_vid, tmp_pid;
```

```
uint16_t endian_test = 0xBE00;
char *error_lang = NULL, *old_dbg_str = NULL, str[256];

// Default to generic, expecting VID:PID
VID = 0;
PID = 0;
test_mode = USE_GENERIC;

if (((uint8_t*)&endian_test)[0] == 0xBE) {
        printf("Despite their natural superiority for end users, big endian\n"
                "CPUs are not supported with this program, sorry.\n");
        return 0;
}

if (argc >= 2) {
        for (j = 1; j<argc; j++) {
                arglen = strlen(argv[j]);
                if ( ((argv[j][0] == '-') || (argv[j][0] == '/'))
                  && (arglen >= 2) ) {
                        switch(argv[j][1]) {
                        case 'd':
                                debug_mode = true;
                                break;
                        case 'i':
                                extra_info = true;
                                break;
                        case 'w':
                                force_device_request = true;
                                break;
                        case 'b':
                                if ((j+1 >= argc) || (argv[j+1][0] == '-') || (argv[j+1][0] == '/')) {
                                        printf("   Option -b requires a file name\n");
                                        return 1;
                                }
                                binary_name = argv[++j];
                                binary_dump = true;
                                break;
                        case 'l':
                                if ((j+1 >= argc) || (argv[j+1][0] == '-') || (argv[j+1][0] == '/')) {
                                        printf("    Option -l requires an ISO 639-1 language
        parameter\n");
                                        return 1;
                                }
                                error_lang = argv[++j];
                                break;
                        case 'j':
                                // OLIMEX ARM-USB-TINY JTAG, 2 channel composite device
        - 2 interfaces
                                if (!VID && !PID) {
                                        VID = 0x15BA;
                                        PID = 0x0004;
                                }
                                break;
                        case 'k':
                                // Generic 2 GB USB Key (SCSI Transparent/Bulk Only) - 1
        interface
                                if (!VID && !PID) {
                                        VID = 0x0204;
                                        PID = 0x6025;
                                }
                                break;
                        // The following tests will force VID:PID if already provided
                        case 'p':
                                // Sony PS3 Controller - 1 interface
                                VID = 0x054C;
                                PID = 0x0268;
                                test_mode = USE_PS3;
                                break;
                        case 's':
                                // Microsoft Sidewinder Precision Pro Joystick - 1 HID
        interface
                                VID = 0x045E;
                                PID = 0x0008;
                                test_mode = USE_HID;
                                break;
                        case 'x':
                                // Microsoft XBox Controller Type S - 1 interface
                                VID = 0x045E;
                                PID = 0x028E;
                                test_mode = USE_XBOX;
                                break;
                        default:
                                show_help = true;
                                break;
                        }
                    } else {
                        for (i=0; i<arglen; i++) {
                                        if (argv[j][i] == ':')
                                            break;
                                }
                                if (i != arglen) {
                                        if (sscanf(argv[j], "%x:%x" , &tmp_vid, &tmp_pid) != 2) {
                                            printf("    Please specify VID & PID as \"vid:pid\" in
        hexadecimal format\n");
                                            return 1;
                                        }
                                        VID = (uint16_t)tmp_vid;
                                        PID = (uint16_t)tmp_pid;
                                } else {
                                        show_help = true;
                                }
                        }
                    }
                }

                if ((show_help) || (argc == 1) || (argc > 7)) {
                    printf("usage: %s [-h] [-d] [-i] [-k] [-b file] [-l lang] [-j] [-x] [-s] [-p] [-w] [vid:pid]\n",
        argv[0]);
                    printf("   -h      : display usage\n");
                    printf("   -d      : enable debug output\n");
                    printf("   -i      : print topology and speed info\n");
                    printf("   -j      : test composite FTDI based JTAG device\n");
                    printf("   -k      : test Mass Storage device\n");
                    printf("   -b file : dump Mass Storage data to file 'file'\n");
                    printf("   -p      : test Sony PS3 SixAxis controller\n");
                    printf("   -s      : test Microsoft Sidewinder Precision Pro (HID)\n");
```

```
		printf("   -x      : test Microsoft XBox Controller Type S\n");
		printf("   -l lang : language to report errors in (ISO 639-1)\n");
		printf("   -w      : force the use of device requests when querying WCID descriptors\n");
		printf("If only the vid:pid is provided, xusb attempts to run the most appropriate test\n");
		return 0;
	}

	// xusb is commonly used as a debug tool, so it's convenient to have debug output during libusb_init(),
	// but since we can't call on libusb_set_debug() before libusb_init(), we use the env variable method
	old_dbg_str = getenv("LIBUSB_DEBUG");
	if (debug_mode) {
		if (putenv("LIBUSB_DEBUG=4") != 0)	// LIBUSB_LOG_LEVEL_DEBUG
			printf("Unable to set debug level");
	}

	version = libusb_get_version();
	printf("Using libusb v%d.%d.%d.%d\n\n", version->major, version->minor, version->micro, version->nano);
	r = libusb_init(NULL);
	if (r < 0)
		return r;

	// If not set externally, and no debug option was given, use info log level
	if ((old_dbg_str == NULL) && (!debug_mode))
		libusb_set_debug(NULL, LIBUSB_LOG_LEVEL_INFO);
	if (error_lang != NULL) {
		r = libusb_setlocale(error_lang);
		if (r < 0)
			printf("Invalid or unsupported locale '%s': %s\n", error_lang, libusb_strerror((enum libusb_error)r));
	}

	test_device(VID, PID);

	libusb_exit(NULL);

	if (debug_mode) {
		snprintf(str, sizeof(str), "LIBUSB_DEBUG=%s", (old_dbg_str == NULL)?"":old_dbg_str);
		str[sizeof(str) - 1] = 0;   // Windows may not NUL terminate the string
	}

	return 0;
}
```