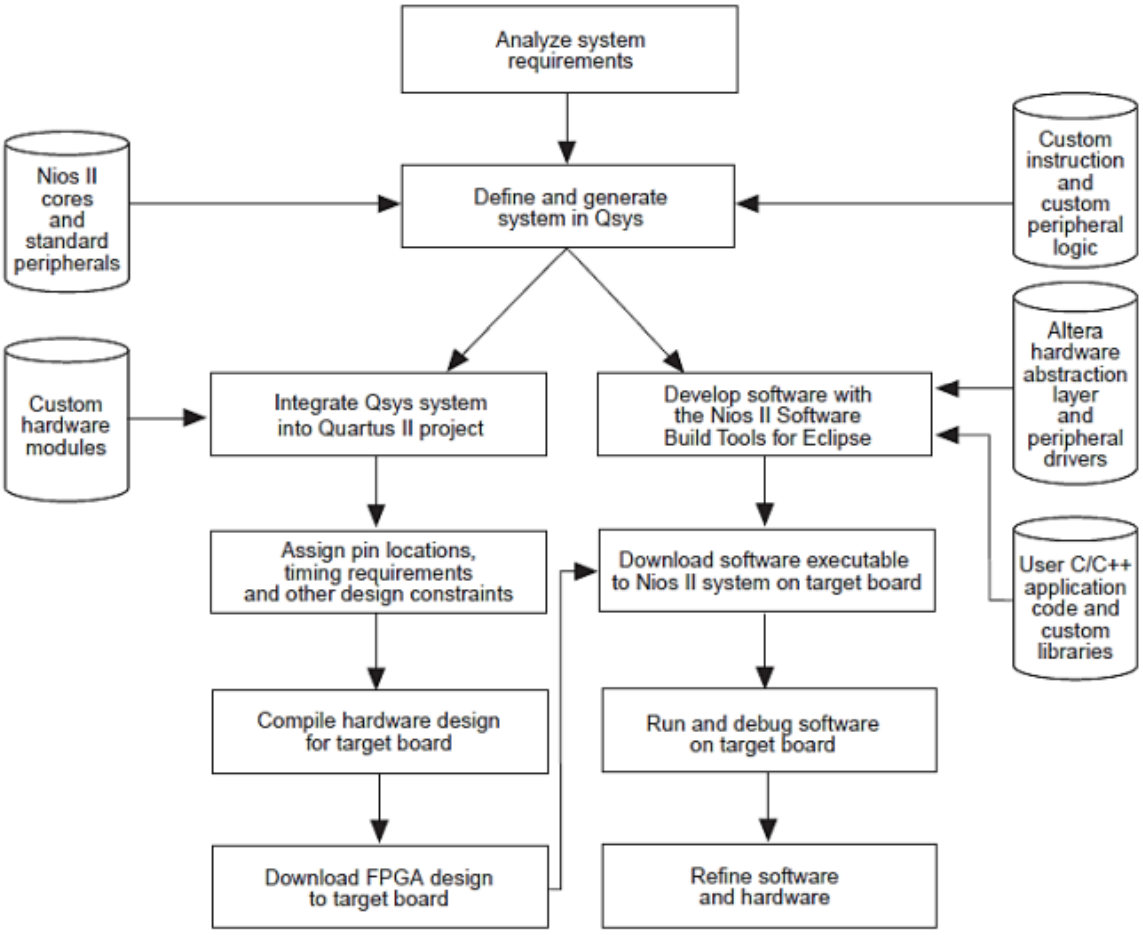
FPGA
LAB 2

اعضا: علیرضا جابری راد - مهدی رادمان - نیما سلیمی

مقدمه

روند کلی طراحی سخت افزاری/نرم افزاری سیستم مبتنی بر پردازنده Nios II:



جهت طراحی و پیاده سازی یک سیستم با استفاده از Nios II System ابتدا باید آنالیز کنیم که سیستم ما به چه چیزهایی نیاز دارد. یعنی علاوه بر پردازنده و حافظه به چه peripheral ها، سخت افزارهای تعبیه شده در هسته ی Nios II، ساختارهای شخصی سازی شده در استفاده از peripheral ها و دستورات شخصی سازی شده ای برای اجرا در پردازنده نیاز دارد و این موارد را در محیط Qsys تعریف و تولید میکنیم. سپس سخت افزارهایی که خارج از FPGA هستند را در محیط Qsys به سیستم اضافه میکنیم و به پروژه ساخته شده در Quartus II پیوست میکنیم. در نهایت پس از تعیین پین های ورودی و خروجی و زمانبندی های لازم برای اجرای سیستم، طراحی سخت افزار مذکور را در محیط Quartus II کامپایل و سنتز میکنیم و به برد FPGA مورد نظر انتقال میدهیم. بخشی از این اطلاعات به بخش طراحی نرم افزاری منتقل میشود تا بتوانیم از سخت افزارهای موجود از طریق نرم افزار بهره مند شویم. در شاخه ی راستی که در شکل مشاهده میکنید، ابتدا با استفاده از HAL ها و درایورهای Peripheral ها و برنامه نویسی به زبان C یا C++ و کمک گرفتن از کتابخانه های آماده برای برنامه نویسی به این دو زبان، بخش نرم افزاری را به طور کامل در محیط Eclipse پیاده سازی میکنیم. سپس، با وجود نرم افزار و اطلاعاتی که از بخش سخت افزاری داریم کد را کامپایل و روی برد مقصد اجرا میکنیم و میتوانیم آن را اشکال زدایی کنیم. در نهایت میتوانیم بهبودهایی را جهت عملکرد هرچه بهتر سیستم به لحاظ سخت افزاری و نرم افزاری بر روی طراحی و پیاده سازی سیستم اعمال کنیم.

FPGA
LAB 2

بخش اول: آشنایی با DE2 Media Computer و اجرای کد نمونه

- خلاصه عملکرد کدهای مربوط به media_interrupt_HAL را گزارش نمایید:

عملکرد کد این فایل بدین صورت است که دستورات مربوط به ضبط و پخش صدا را با دریافت وقفه حین فشردن کلید های فیزیکی روی برد انجام میدهد و در حین پخش یا ضبط صدا یکی از ال ای دی های روی برد روشن میماند. همچنین وضعیت موس بر روی 6 تا از 7-segment ها نمایش داده میشود. علاوه بر موارد گفته شده بر روی مانیتور و ال سی دی روی برد مواردی نمایش داده میشود که در ادامه در مورد آن ها توضیح داده میشود.

- مشخص کنید که در ابتدا بر روی مانیتور، چه عبارتی نمایش داده میشود:

در مانیتور عبارت زیر نمایش داده میشود

Altera DE2

Media Computer

و عبارت ALTERA بین دور کادر آبی رنگی که متن بالا در آن نمایش داده میشود و لبه های مانیتور جابجا میشود.

در ال سی دی روی بورد عبارت زیر نمایش داده میشود و حرکت میکند:

Welcome to the DE2 Media Computer…

- با فشردن کلید های 1 و 2 چه عملیاتی اجرا میشود:

1. با فشردن کلید 1 (KEY[1]) صوت به مدت ده ثانیه ضبط میشود و در این حین ال ای دی سبز 0 (LEDG[0]) روشن میماند. عملیات ضبط با استفاده از interrupt ها کنترل میشود.

2. با فشردن کلید 2 یک اینتراپت جهت پخش کردن فایل ضبط شده ایجاد میشود و فایل ضبط شده پخش میشود. در حین پخش شدن صوت، ال ای دی سبز 1 روشن میماند.

بخش دوم: راه اندازی درایور ماوس با خروجی PS/2

با مطالعه کد مربوط به ماوس، مشخص کنید که برای دریافت اطلاعات کامل مربوط به وضعیت فعلی ماوس (byte 1 و byte 2 و byte 3) چند بار باید تابع PS2_ISR فراخوانی شود؟ فراخوانی این تابع به چه صورت خواهد بود؟

سه بار باید فراخوانی شود زیرا این تابع ورودی را بایت به بایت دریافت میکند به همین دلیل برای دریافت سه بایت مربوط به وضعیت ماوس باید سه بار این تابع فراخوانی شود. فراخوانی این تابع به صورت وقفه (interrupt) است.

خواسته 1)نمایش حالت دکمه بر روی LEDR[2..0]، وضعیت حرکت افقی ماوس در HEX[7..4] و وضعیت حرکت عمودی ماوس در HEX[3..0]:

نحوه ی اجرای این بخش را در فیلم های ارسال شده میتوانید مشاهده کنید.

نمایش وضعیت دکمه های ماوس بر روی ال ای دی های قرمز:

وضعیت فشرده بودن یا نبودن کلید های ماوس بنا بر جهتی که دارند بر روی ال ای دی های قرمز نمایش داده میشوند. یعنی چپ کلیک بر LED[2]، دکمه ی وسط بر LED[1] و راست کلیک بر روی LED[0] نمایش داده میشوند. بدین منظور با الهام گیری از کد

FPGA
LAB 2

های مربوط به ال ای دی های سبز رنگ، تغییراتی در کد اعمال شد. در بدنه struct مربوط به peripheral ها که در فایل globals.h قابل مشاهده است خط زیر اضافه شد:

```
alt_up_parallel_port_dev *red_LEDs_dev;
```

همچنین تابع زیر جهت برگرداندن وضعیت فشرده شدن دکمه های ماوس نوشته شد:

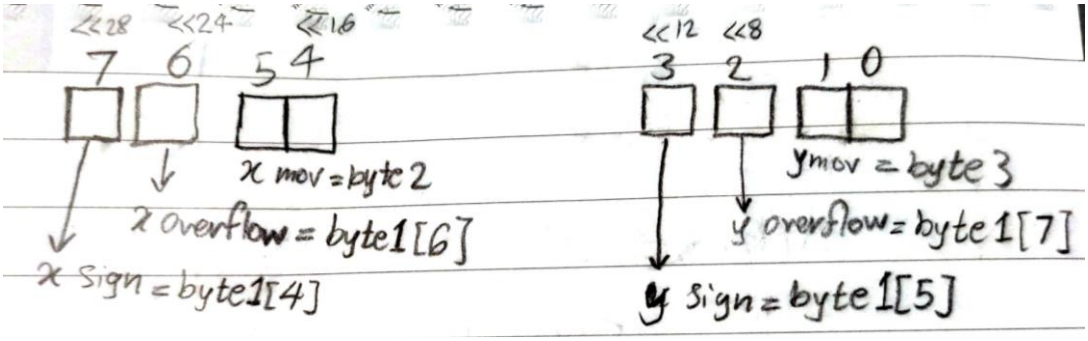
```
unsigned char give_bottons_stat(unsigned char byte1, unsigned char byte2, unsigned char byte3){
    return ((byte1%2)<<2) | (((byte1/4)%2)<<1) | ((byte1/2)%2);
}
```

و در ادامه با اضافه کردن تابع بالا به بدنه ی حلقه ی اصلی اجرای برنامه و تعیین وضعیت ال ای دی های قرمز بر مبنای مقدار برگردانده شده توسط این تابع، عملکرد مورد نظر تحقق یافت:

```
bottons_stat=give_bottons_stat(byte1,byte2,byte3);//determines status of the mouse bottons
alt_up_parallel_port_write_data (up_dev.red_LEDs_dev, bottons_stat);
```

نمایش وضعیت حرکت ماوس بر روی 7-segment ها:

بر مبنای تصویر زیر، در تابع از پیش نوشته شده ی HEX_PS2 تغییراتی اعمال شد:



تغییرات مطابق با تقسیم بندی بالا انجام شد که بخش تغییر یافته ی کد را در تصویر زیر میتوانید مشاهده کنید:

```
////////////////////////////////////////////////
unsigned char seg7,seg6,seg54,seg3,seg2,seg10;//seg54 means segment 5 and segment 4 together, for seg10 the same property holds
seg7=(b1>>4)%2;
seg6=(b1>>6)%2;
seg54=b2;
seg3=(b1>>5)%2;
seg2=(b1>>7)%2;
seg10=b3;
shift_buffer = (seg7 << 28) | (seg6 << 24) | (seg54 << 16) | (seg3 << 12) | (seg2 << 8) | seg10;
////////////////////////////////////////////////

for ( i = 0; i < 8; ++i )//end number changed from 6 to 8
{
    nibble = shift_buffer & 0x0000000F;     // character is in rightmost nibble
    code = seven_seg_decode_table[nibble];
    hex_segs[i] = code;
    shift_buffer = shift_buffer >> 4;
}
/* drive the hex displays */
*(HEX3_HEX0_ptr) = *(int *) (hex_segs);
*(HEX7_HEX4_ptr) = *(int *) (hex_segs+4);
```

خواسته 2) تعیین و ذخیره ی موقعیت مکانی ماوس بر روی صفحه نمایش و وضعیت فشرده شدن کلید های ماوس:

تصویر و توضیحات مربوط به تابع وضعیت سه کلید ماوس را در بخش قبلی میتوانید مشاهده کنید. این تابع که با نام give_bottons_stat معرفی شده مقدار برگشتی اش در bottons_stat ذخیره میشود.

اما به منظور تعیین موقعیت ماوس در صفحه تابع زیر نوشته شد. در این تابع به علت پرش هایی که در ماوس رخ میداد شرط حرکت با گام های کمتر از 10 و حرکت نکردن به هنگام یک شدن overflow در هر یک از جهات، اضافه شد. پس از تعیین موقعیت بعدی ماوس، زیربرنامه ای به منظور کنترل عدم تجاوز موقعیت ماوس از ابعاد صفحه اجرا میشود.

به منظور حرکت سریعتر ماوس میتوانید بخش کامنت شده را از حالت کامنت خارج کنید و با افزایش ضریب speed سرعت جابجایی ماوس را افزایش دهید. تصویر تابع معرفی شده را در زیر مشاهده میفرمایید:

```
void mouseLocation(unsigned char b1, unsigned char b2, unsigned char b3){
    short overflow_x, overflow_y, sign_x, sign_y;
    //short speed;
    overflow_x=(b1>>6)%2;
    overflow_y=(b1>>7)%2;
    sign_x=(b1>>4)%2;
    sign_y=(b1>>5)%2;

    /*speed=1;
    b2=b2*speed;
    b3=b3*speed;*/

    //horizontal location
    if(sign_x){
        if(!overflow_x && (256-b2)<10)
            mouse_x=mouse_x -(256-b2);
    }
    else{
        if(!overflow_x && b2<10)
            mouse_x=mouse_x+b2;
    }
    if(mouse_x>screen_max_x)
        mouse_x=screen_max_x;
    else if(mouse_x<0)
        mouse_x=0;

    //vertical location
    if(sign_y){
        if(!overflow_y && (256-b3)<10)
            mouse_y=mouse_y+(256-b3);
    }
    else{
        if(!overflow_y && b3<10)
            mouse_y=mouse_y-b3;
    }
    if(mouse_y>screen_max_y)
        mouse_y=screen_max_y;
    else if(mouse_y<0)
        mouse_y=0;
}
```

بخش سوم: طراحی Audio Player با نمایش گرافیکی و ایجاد Echo

خواسته 1) نمایش ماوس بر روی صفحه:

با استفاده از دو حلقه ی زیر و تابع مربوط به رسم یک پیکسل بر روی صفحه و تابع مربوط به مدیریت موقعیت ماوس که در بخش قبل مشاهده کردید، نشانگر ماوس به صورت فلش بر روی صفحه نمایش داده شد. علاوه بر آن حلقه ی دیگری به منظور پاک کردن تصویر موقعیت قبلی ماوس به کد اضافه شد:

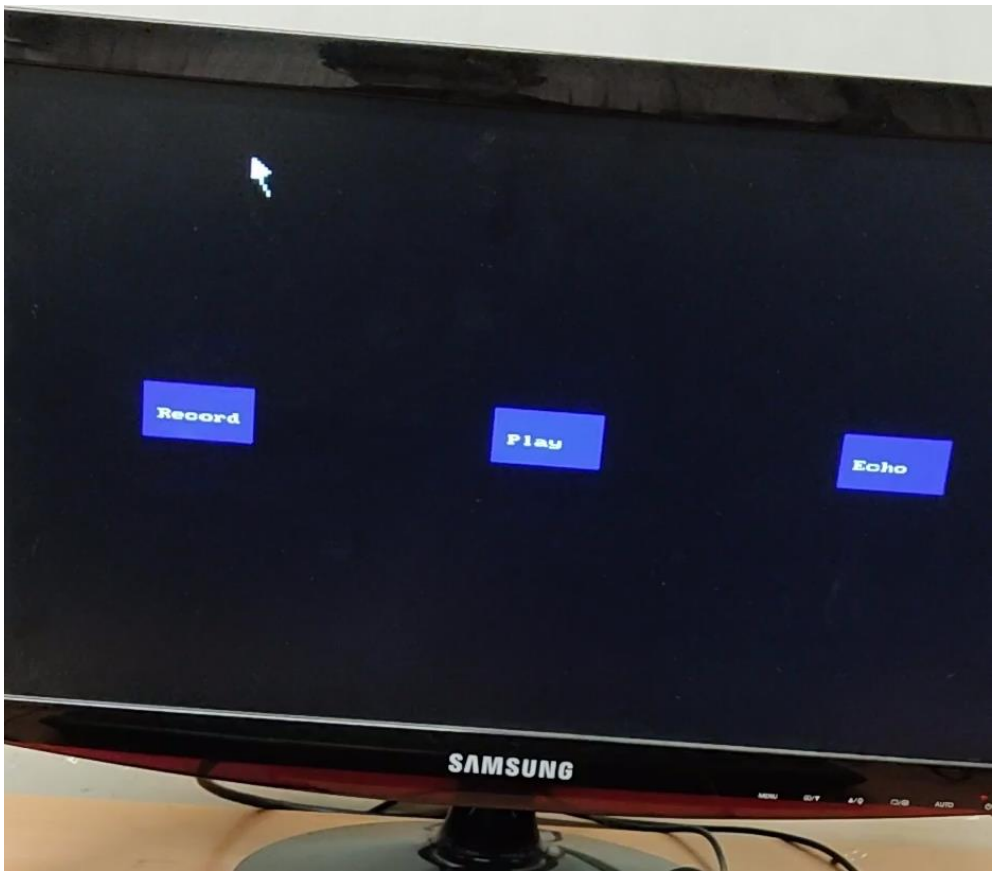
کد:

```
for(j=0;j<8;j++){//clear previous mouse cursor
    for(i=0;i<16;i++){
        if(cursor_shape[i][j]!= -1)
            alt_up_pixel_buffer_dma_draw(pixel_buffer_dev,0,mouse_x+j, mouse_y+i);
    }
}

mouseLocation(byte1,byte2,byte3);//calculates the location of the mouse cursor in the screen

for(j=0;j<8;j++){//print mouse cursor
    for(i=0;i<16;i++){
        if(cursor_shape[i][j]!= -1)
            alt_up_pixel_buffer_dma_draw(pixel_buffer_dev,cursor_shape[i][j]*(-1),mouse_x+j, mouse_y+i);
    }
}
```

در کد بالا cursor_shape همان ماتریس دو بعدی داده شده در صورت پروژه است. تمامی متغیرهای استفاده شده به صورت global در فایل global.c موجود هستند و در فایل اصلی به صورت extern استفاده شده اند.

خواسته 2) نمایش مستطیل های ضبط، پخش و اکو کردن صوت ورودی:

در تصویر زیر نمایش این سه المان را بر روی نمایشگر مشاهده میکنید:



به منظور نمایش این سه المان و تصویر پس زمینه، از کدهای قبلی موجود الهام گرفته شد و کد زیر نوشته شد:

```
/***************Boxes**************/
//Record Botton
color=0x187F;
record_x1=40; record_y1=110; record_x2=70; record_y2=130;
alt_up_pixel_buffer_dma_draw_box (pixel_buffer_dev, record_x1 , record_y1 , record_x2 ,
    record_y2, color, 0);

//Play Botton
play_x1=140; play_y1=110; play_x2=170; play_y2=130;
alt_up_pixel_buffer_dma_draw_box (pixel_buffer_dev, play_x1 , play_y1 , play_x2 ,
    play_y2, color, 0);

//Echo Botton
echo_x1=240; echo_y1=110; echo_x2=270; echo_y2=130;
alt_up_pixel_buffer_dma_draw_box (pixel_buffer_dev, echo_x1 , echo_y1 , echo_x2 ,
    echo_y2, color, 0);


/*************Texts in the boxes*************/
alt_up_char_buffer_string (char_buffer_dev, "Record\0", record_x1/4 + 1, record_y1/4 + 3);
alt_up_char_buffer_string (char_buffer_dev, "Play\0", play_x1/4 + 1, play_y1/4 + 3);
alt_up_char_buffer_string (char_buffer_dev, "Echo\0", echo_x1/4 + 1, echo_y1/4 + 3);
```

خواسته 3)در صورت کلیک روی هر یک از دکمه ها عملیات مربوطه اجرا شود:

این بخش با استفاده از دو تابع پیاده سازی میشود که command_detect بر حسب وضعیت دکمه چپ و موقعیت ماوس عملیات مربوطه را در متغیر command مینویسد و این متغیر در record_play_echo به عنوان مبنا برای اجرای دستور مورد استفاده قرار میگیرد. دستور ها هم به صورت وقفه اجرا میشوند.

```
void command_detector(unsigned char bottons_stat, alt_up_audio_dev * audio_dev){
    if((bottons_stat>>2)%2){
        if(mouse_x>=40 && mouse_x<=70 && mouse_y>=110 && mouse_y<=130)
            command="Record\0";
        else if(mouse_x>=140 && mouse_x<=170 && mouse_y>=110 && mouse_y<=130)
            command="Play\0";
        else if(mouse_x>=240 && mouse_x<=270 && mouse_y>=110 && mouse_y<=130)
            command="Echo\0";
        else
            command="          \0";
    }
    else
        command="          \0";
    return;
}
```

```
void record_play_echo(alt_up_audio_dev * audio_dev){
            if(!strcmp(command,"Record")){
                // reset the buffer index for recording
                buf_index_record = 0;
                // clear audio FIFOs
                alt_up_audio_reset_audio_core (audio_dev);
                // enable audio-in interrupts
                alt_up_audio_enable_read_interrupt (audio_dev);
            }
            else if(!strcmp(command,"Play")){
                // reset counter to start playback
                echo_en=0;
                buf_index_play = 0;
                // clear audio FIFOs
                alt_up_audio_reset_audio_core (audio_dev);
                // enable audio-out interrupts
                alt_up_audio_enable_write_interrupt (audio_dev);
            }
            else if(!strcmp(command,"Echo")){
                // reset counter to start playback
                echo_en=1;
                buf_index_play = 0;
                // clear audio FIFOs
                alt_up_audio_reset_audio_core (audio_dev);
                // enable audio-out interrupts
                alt_up_audio_enable_write_interrupt (audio_dev);
            }
}
```

کد زیر در لوپ اصلی قرار دارد:

```
command_detector(bottons_stat,audio_dev);
record_play_echo(audio_dev);
```

خواسته 4) اکو کردن:

فیلم مربوط به عملکرد این بخش به همراه فایل ها ارسال شده است. این بخش علاوه بر بخش قبل با استفاده از فشردن کلید فیزیکی چهارم رو برد هم پیاده سازی شده:

```
else if (KEY_value == 0x8){
    // reset counter to start playback
    echo_en=1;
    buf_index_play = 0;
    // clear audio FIFOs
    alt_up_audio_reset_audio_core (audio_dev);
    // enable audio-out interrupts
    alt_up_audio_enable_write_interrupt (audio_dev);
}
```

همچینین تغییرات زیر در وقفه صورت گرفته است:

```
extern volatile int echo_en;
```

```
if (alt_up_audio_write_interrupt_pending(up_dev->audio_dev))    // check for write interrupt
{
    if(echo_en){
        alt_up_parallel_port_write_data (up_dev->green_LEDs_dev, 0x4); // set LEDG[2] on

        // output data until the buffer is empty
        if (buf_index_play < BUF_SIZE)
        {
            num_written = alt_up_audio_play_r (up_dev->audio_dev, &(r_buf_echo[buf_index_play]),
            BUF_SIZE - buf_index_play);
            /* assume that we can write the same # words to the left and right */
            (void) alt_up_audio_play_l (up_dev->audio_dev, &(l_buf_echo[buf_index_play]),
                num_written);
            buf_index_play += num_written;

            if (buf_index_play == BUF_SIZE)
            {
            // done playback
                alt_up_parallel_port_write_data (up_dev->green_LEDs_dev, 0); // turn off LEDG
                alt_up_audio_disable_write_interrupt(up_dev->audio_dev);
            }
        }
    }
    else{//play the audio normally
        alt_up_parallel_port_write_data (up_dev->green_LEDs_dev, 0x2); // set LEDG[1] on

        // output data until the buffer is empty
        if (buf_index_play < BUF_SIZE)
        {
            num_written = alt_up_audio_play_r (up_dev->audio_dev, &(r_buf[buf_index_play]),
                    BUF_SIZE - buf_index_play);
            /* assume that we can write the same # words to the left and right */
            (void) alt_up_audio_play_l (up_dev->audio_dev, &(l_buf[buf_index_play]),
                    num_written);
            buf_index_play += num_written;

            if (buf_index_play == BUF_SIZE)
            {
                // done playback
                alt_up_parallel_port_write_data (up_dev->green_LEDs_dev, 0); // turn off LEDG
                alt_up_audio_disable_write_interrupt(up_dev->audio_dev);
            }
        }
    }
```

تابع مربوط به تولید اکو در بافر مخصوص به خود در بدنه ی کد وقفه ی مربوط به صدا تعبیه شده که پس از اتمام ضبط صدا، این تابع فعال میشود و بافر مربوط به صدای اکو را تولید میکند.

```
unsigned int l_buf_echo[BUF_SIZE];
unsigned int r_buf_echo[BUF_SIZE];

if(buf_index_record == BUF_SIZE)
    make_echo();
```

```
void make_echo(){
    int i;
    for(i=5000;i<BUF_SIZE;i++){
        l_buf_echo[i]=l_buf[i-5000]+l_buf[i-1000];
        r_buf_echo[i]=r_buf[i-5000]+r_buf[i-1000];
    }
    return;
}
```

خواسته پنجم) دریافت ورودی و انجام عملیات:

این قسمت به سادگی و کامنت کردن command_detect و جایگزینی آن با ;(“%s”, command)scanf قابل پیاده سازی است که در کد کامنت شده است، برای تست آن را از حالت کامنت خارج کنید.

خواسته ششم) بررسی صحت عملکرد سیستم:

صحت عملکرد سیستم را میتوانید در ویدیو های ارسال شده مشاهده کنید. همچنین در صورت تمایل میتوانید کد را ران کرده و از صحت عملکرد آن اطمینان حاصل کنید.

امتیازی

همانطور که در بالا مشاهده کردید، نشانگر ماوس در پروژه پیاده سازی شد.

-نحوه راه اندازی ماوس:

با استفاده از دو حلقه ای که در بالا توضیح داده شد نشانگر ماوس در هر اجرای حلقه نمایش داده شده و پاک میشود و مبنای رسم آن ماتریس دو بعدی ای است که در متغیر مربوطه ذخیره شده.

-ضرورت استفاده از Back Buffer:

به دلیل آنکه اگر بخواهیم از یک بافر به طور همزمان بخوانیم و بنویسیم، در بسیاری از موارد بخشی از داده هایی که باید خوانده شوند حین خوانده شدن مقدارشان تغییر میکند و مقدار درستی خوانده نمیشود، لذا لازم است ابتدا عمل خواندن انجام شود و سپس اطلاع داده شود که عمل نوشتن صورت بگیرد و برعکس و این مورد باعث کند شدن کار کردن با آن دستگاه میشود.

لذا به منظور حل این مشکل از دو بافر استفاده میکنند که در هر مرحله یک بافر نقش بافر خواندن و دیگری نقش بافر نوشتن را بر عهده میگیرد و در هر بار انجام عملیات نقش این دو عوض میشود. در این صورت عملیات نوشتن و خواندن بدون توقف به طور همزمان با هم انجام میشود و چون عملیات خواندن بدون توقف است، خروجی سریع تر و روان تری از آن سیستم دریافت میکنیم.

در برخی موارد حتی برای عملکرد بهتر از چندین بافر بهره گرفته میشود.

نکته تکمیلی:

بخش عمده ی تغییرات در فایل media_interrupt_HAL.c اعمال شده و سایر تغییرات در فایل های توضیح داده شده در گزارش اعمال شده است و میتوانید به آن ها مراجعه کنید. فایل جدیدی به کد اضافه نشده و لذا امکان جدا کردن هر بخش کد امکانپذیر نبود.

با تشکر از توجه شما