

FPGA – based Embedded System Design

LAB #2

Group Members

Mohammad Taghizadeh Givari	810198373
Zeinab Saeedi	810198411
Amin Aroufzad	810198538

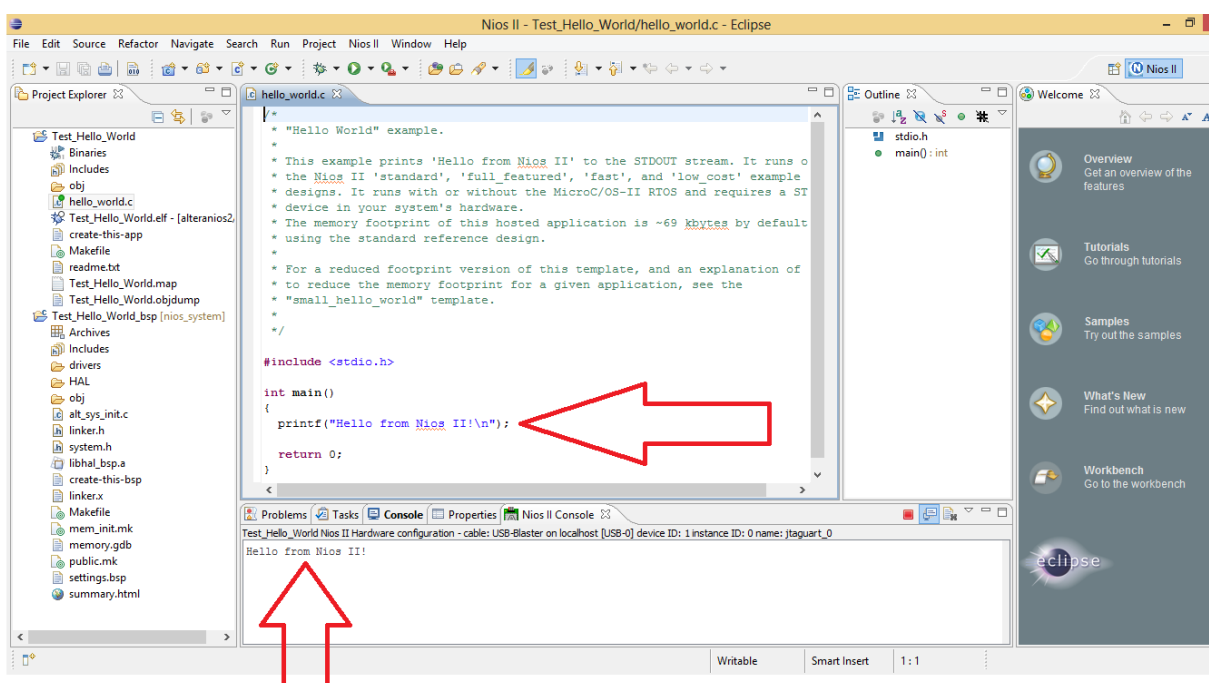
محتوا

۱. بخش ۱: آشنایی با DE2 Media Computer ۲
۲. بخش ۲: راه اندازی درایور ماوس با خروجی PS/2 ۱۹
۳. طراحی Audio Player با نمایش گرافیکی و ایجاد Echo ۳۳

بخش ۱: آشنایی با DE2 Media Computer

بخش های ۱ تا ۴:

خروجی اجرا برنامه "Hello World" بر روی برد DE2



بخش های ۵ و ۶:

خلاصه نحوه عملکرد کد های مروط به `media_interrupt_HAL`:

- ابتدا تابع `main` موجود در فایل `media_interrupt_HAL.c` اجرا می شود. این تابع دارای یک بخش اولیه است که کد های مربوط به کارهایی که فقط یک دفعه باید انجام شوند (مثل مقداردهی های اولیه، فعال کردن ماوس، پورت صدا، شناساندن وقفه ها به تابع `main` و...) در آن صورت میگیرد و یک حلقه بینهایت که کارهایی که مدام باید انجام شوند (مثل رسم مکان ماوس، بروز کردن مانیتور، رسیدگی به وقفه ها و...) در آن صورت می گیرد.

- در این تابع ابتدا متغیر های محلی و متغیر های سراری (`volatile`) تعریف می شوند. سپس وقفه های مربوط به ماوس، پورت صدا، تایمر و فشرده شدن دکمه توسط دستور `alt_irq_register` به تابع `main` شناسانده می شوند. ماوس، پورت صدا، نمایشگر کاراکتری LCD، مانیتور (`Pixel buffer`) و پورت های موازی (`parallel_ports`) توسط دستور `alt_up_...._open_dev` فعال می شوند. سپس تصویر اولیه ای که روی مانیتور قرار است نشان داده شود، ترسیم میشود. (ابتدا پیش زمینه تصویر با رنگ بنفش ترسیم شده، سپس یک مستطیل آبی وسط صفحه حاوی متن کشیده شده و در آخر عبارت متحرک (`Altera`) در موقعیت اولیه اش رسم میشود).

- سپس در یک حلقه بینهایت، ابتدا در یک حلقه صبر می کند تا همگام با تایمر شود، بعد اقداماتی که همواره باید انجام گیرد مثل نمایش موقعیت ماوس بر روی `seven segment displayer`، آپدیت کردن مانیتور (در کد آماده `Altera`، این کار معادل جابجا کردن محل نمایش عبارت `Altera` بر روی مانیتور است) و... صورت می گیرد. در این میان هرگاه وقفه ای از جانب ماوس، پورت صدا، تایمر و دکمه

صورت گیرد، اجرا تابع main متوقف شده، تابع مربوط به مدیریت وقفه صدا زده می شود:

:PS2_ISR

در این فایل ابتدا داده های ارسالی (۳ بایت) از سوی ماوس خوانده و ذخیره می شوند، سپس اگر پاسخ ماوس به دستور reset مثبت بود، دستور مرتبط با فعال کردن ماوس برای فرستادن موقعیتش، به ماوس فرستاده می شود تا از آن پس، ماوس وضعیت دکمه ها و موقعیتش را به پردازنده ارسال کند.

:audio_ISR

در این فایل، ابتدا وقفه مربوط به خواندن از پورت صدا (ضبط کردن صدا) بررسی می شود: ابتدا اولین LED سبز به منظور نشان دادن شروع عملیات ضبط کردن، روشن می شود. سپس داده دریافت شده در پورت های صدا راست و چپ، خوانده و در بافر r_buf و l_buf ذخیره می شوند. این روند تا زمانی که بافر صدا پر شود ادامه می یابد. در آخر وقفه خواندن غیر فعال می شود. سپس وقفه مربوط به نوشتن به پورت صدا (پخش کردن صدا) بررسی می شود: ابتدا دومین LED سبز به منظور نشان دادن شروع عملیات پخش کردن، روشن می شود. سپس داده ذخیره شده در بافر های صدا راست و چپ، خوانده و پخش می شوند. این روند تا زمانی که به انتهای بافر صدا برسیم ادامه می یابد. در آخر وقفه نوشتن غیرفعال می شود.

:pushbutton_ISR

در این فایل، ابتدا وضعیت کلید ها توسط alt_up_parallel_port_read_edge_capture بررسی می شود: اگر اولین کلید فشرده شده باشد، flag مربوط به ضبط صدا، فعال می شود. audio ریست شده و در آخر وقفه خواندن از پورت صدا به منظور ضبط صدا، فعال می شود. اگر دومین کلید فشرده شده باشد، flag مربوط به پخش صدا، فعال می شود. audio ریست شده و در آخر وقفه نوشتن به پورت صدا به منظور پخش صدا، فعال می شود.

:interval_timer_ISR

این فایل، هنگامی صدا زده می شود که زمان رسیدگی به وقفه از مدت زمان مجاز آن، تجاوز کند. در این حالت، با ۱ کردن flag مربوط به timeout، از اجرا شدن وقفه جلوگیری میکنیم تا پردازنده تمام مدت، درگیر یک وقفه نشود و پردازنده بتواند به امور دیگر هم رسیدگی کند.

در آخر پس از رسیدگی به وقفه، اجرا تابع main از جایی که قبلاً متوقف شده بود، از سر گرفته میشود.

```
Nios II - C:\University_Program\NiosII_Computer_Systems\DE2\DE2_Media_Computer\Altera_original_verilog\verilog\software\test\media_interrupt_HAL.c - Eclipse
File Edit Source Refactor Navigate Search Run Project Nios II Window Help

media_interrupt_HAL.c
alt_printf ("Error: could not open character LCD device\n");
return -1;
}
else
{
    alt_printf ("Opened character LCD device\n");
    up_dev.lcd_dev = lcd_dev; // store for use by ISRs
}

/* use the HAL facility for registering interrupt service routines. */
/* Note: we are passing a pointer to up_dev to each ISR (using the context argument) as
 * a way of giving the ISR a pointer to every open device. This is useful because some of the
 * ISRs need to access more than just one device (e.g. the pushbutton ISR accesses both
 * the pushbutton device and the audio device) */
alt_irq_register (0, (void *) &up_dev, (void *) interval_timer_ISR);
alt_irq_register (1, (void *) &up_dev, (void *) pushbutton_ISR);
alt_irq_register (6, (void *) &up_dev, (void *) audio_ISR);
alt_irq_register (7, (void *) &up_dev, (void *) PS2_ISR);

/* create a messages to be displayed on the VGA and LCD displays */
char text_top_LCD[80] = "Welcome to the DE2 Media Computer...\0";
char text_top_VGA[20] = "Altera DE2\0";
char text_bottom_VGA[20] = "Media Computer\0";
char text_ALTERA[10] = "ALTERA\0";
char text_erase[10] = "    \0";

/* output text message to the LCD */
alt_up_character_lcd_set_cursor_pos (lcd_dev, 0, 0); // set LCD cursor location to top row
alt_up_character_lcd_string (lcd_dev, text_top_LCD);
alt_up_character_lcd_cursor_off (lcd_dev); // turn off the LCD cursor

/* open the pixel buffer */
pixel_buffer_dev = alt_up_pixel_buffer_dma_open_dev ("/dev/VGA_Pixel_Buffer");
if ( pixel_buffer_dev == NULL)
```

```
Nios II - C:\University_Program\NiosII_Computer_Systems\DE2\DE2_Media_Computer\Altera_original_verilog\verilog\software\test\media_interrupt_HAL.c - Eclipse
File Edit Source Refactor Navigate Search Run Project Nios II Window Help

media_interrupt_HAL.c
alt_printf ("Opened pixel buffer device\n");

/* the following variables give the size of the pixel buffer */
screen_x = 319; screen_y = 239;
color = 0x1863; // a dark grey color
alt_up_pixel_buffer_dma_draw_box (pixel_buffer_dev, 0, 0, screen_x,
    screen_y, color, 0); // fill the screen

// draw a medium-blue box in the middle of the screen, using character buffer coordinates
blue_x1 = 28; blue_x2 = 52; blue_y1 = 26; blue_y2 = 34;
// character coords * 4 since characters are 4 x 4 pixel buffer coords (8 x 8 VGA coords)
color = 0x187F; // a medium blue color
alt_up_pixel_buffer_dma_draw_box (pixel_buffer_dev, blue_x1 * 4, blue_y1 * 4, blue_x2 * 4,
    blue_y2 * 4, color, 0);

/* output text message in the middle of the VGA monitor */
char_buffer_dev = alt_up_char_buffer_open_dev ("/dev/VGA_Char_Buffer");
if ( char_buffer_dev == NULL)
    alt_printf ("Error: could not open character buffer device\n");
else
    alt_printf ("Opened character buffer device\n");

alt_up_char_buffer_string (char_buffer_dev, text_top_VGA, blue_x1 + 5, blue_y1 + 3);
alt_up_char_buffer_string (char_buffer_dev, text_bottom_VGA, blue_x1 + 5, blue_y1 + 4);

char_buffer_x = 79; char_buffer_y = 59;
ALT_x1 = 0; ALT_x2 = 5/* ALTERA = 6 chars */; ALT_y = 0; ALT_inc_x = 1; ALT_inc_y = 1;
alt_up_char_buffer_string (char_buffer_dev, text_ALTERA, ALT_x1, ALT_y);

/* this loops "bounces" the word ALTERA around on the VGA screen */
while (1)
{
    while (!timeout)
        : // wait to synchronize with timeout. which is set by the interval timer ISR
```

```
/* this loops "bounces" the word ALTERA around on the VGA screen */
while (1)
{
    while (!timeout)
    : // wait to synchronize with timeout, which is set by the interval timer ISR

    /* move the ALTERA text around on the VGA screen */
    alt_up_char_buffer_string (char_buffer_dev, text_erase, ALT_x1, ALT_y); // erase
    ALT_x1 += ALT_inc_x;
    ALT_x2 += ALT_inc_x;
    ALT_y += ALT_inc_y;

    if ( (ALT_y == char_buffer_y) || (ALT_y == 0) )
        ALT_inc_y = -(ALT_inc_y);
    if ( (ALT_x2 == char_buffer_x) || (ALT_x1 == 0) )
        ALT_inc_x = -(ALT_inc_x);

    if ( (ALT_y >= blue_y1 - 1) && (ALT_y <= blue_y2 + 1) )
    {
        if ( (ALT_x1 >= blue_x1 - 1) && (ALT_x1 <= blue_x2 + 1) ) ||
            ((ALT_x2 >= blue_x1 - 1) && (ALT_x2 <= blue_x2 + 1)) )
        {
            if ( (ALT_y == (blue_y1 - 1)) || (ALT_y == (blue_y2 + 1)) )
                ALT_inc_y = -(ALT_inc_y);
            else
                ALT_inc_x = -(ALT_inc_x);
        }
    }

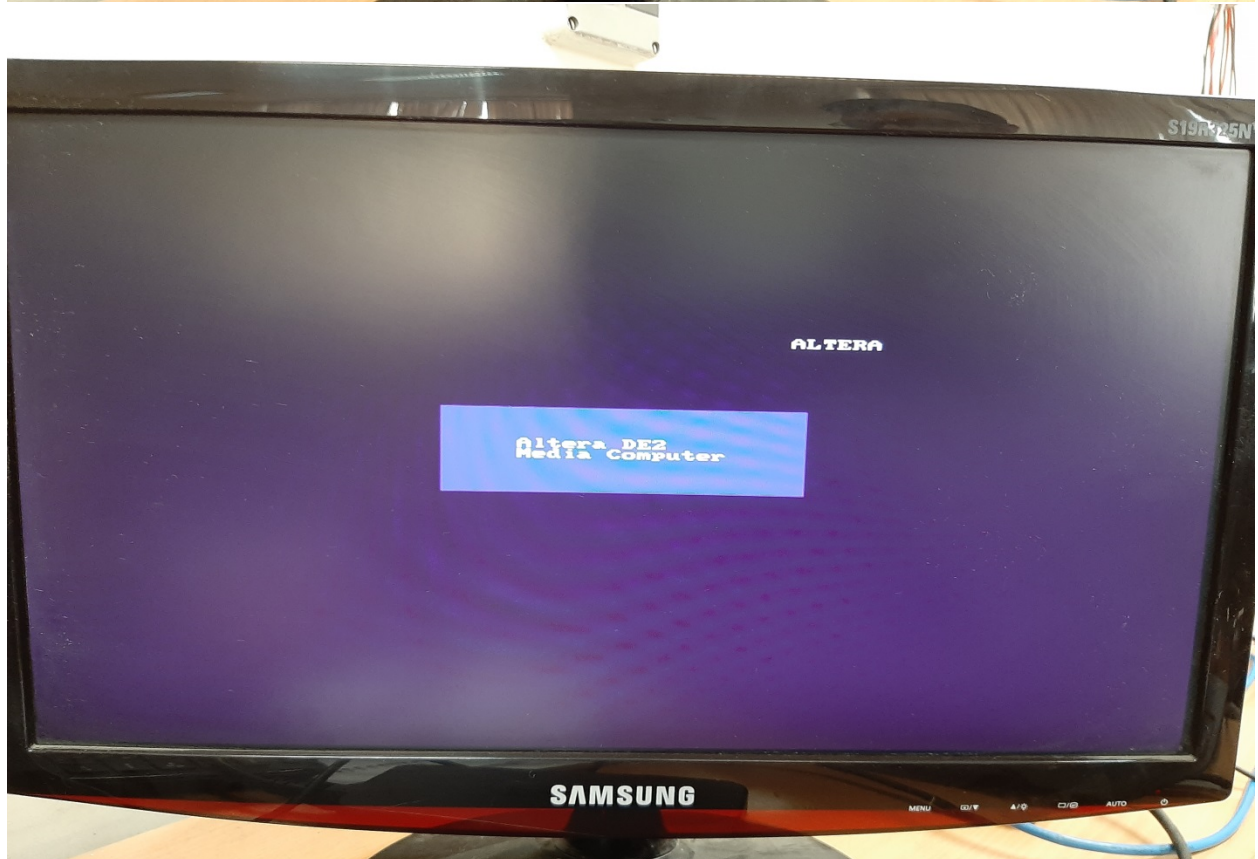
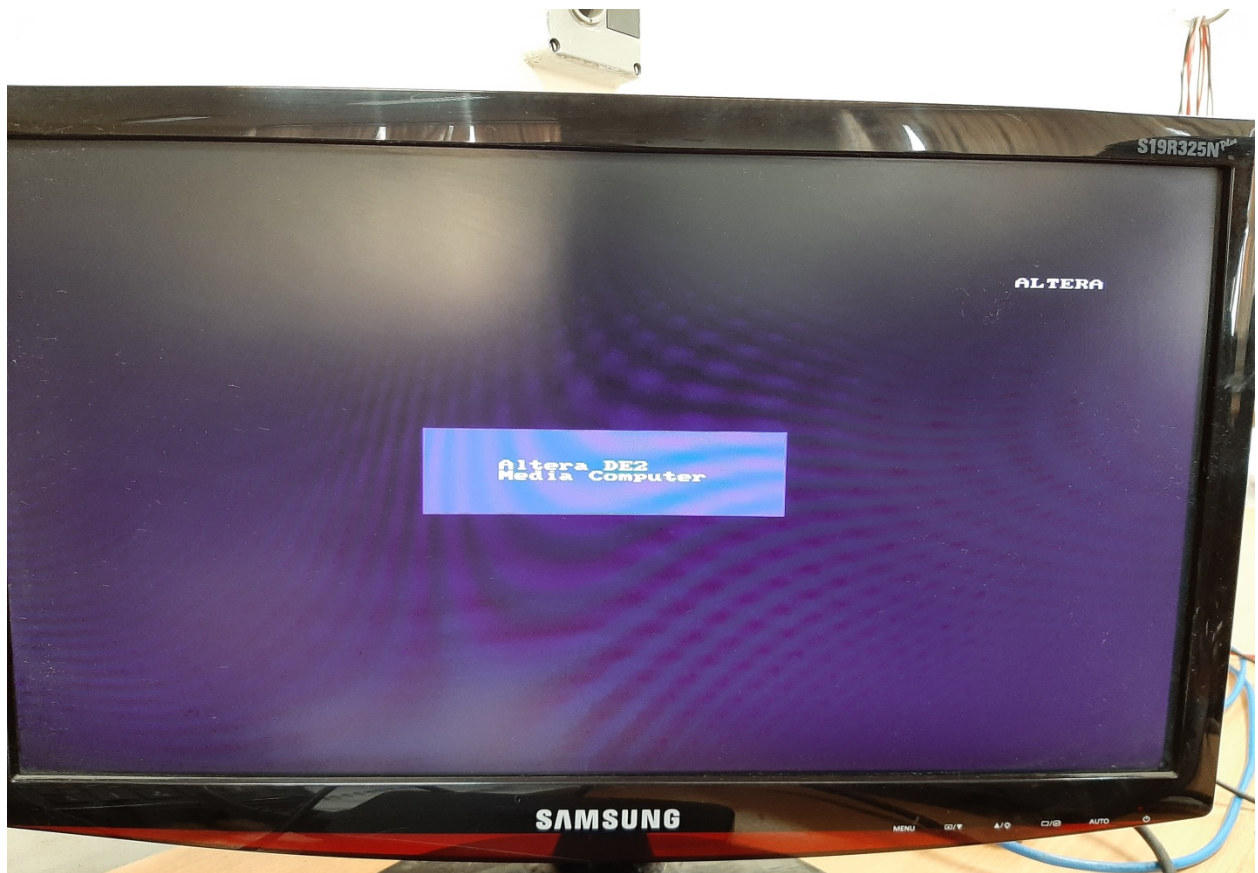
    alt_up_char_buffer_string (char_buffer_dev, text ALTERA, ALT_x1, ALT_y);

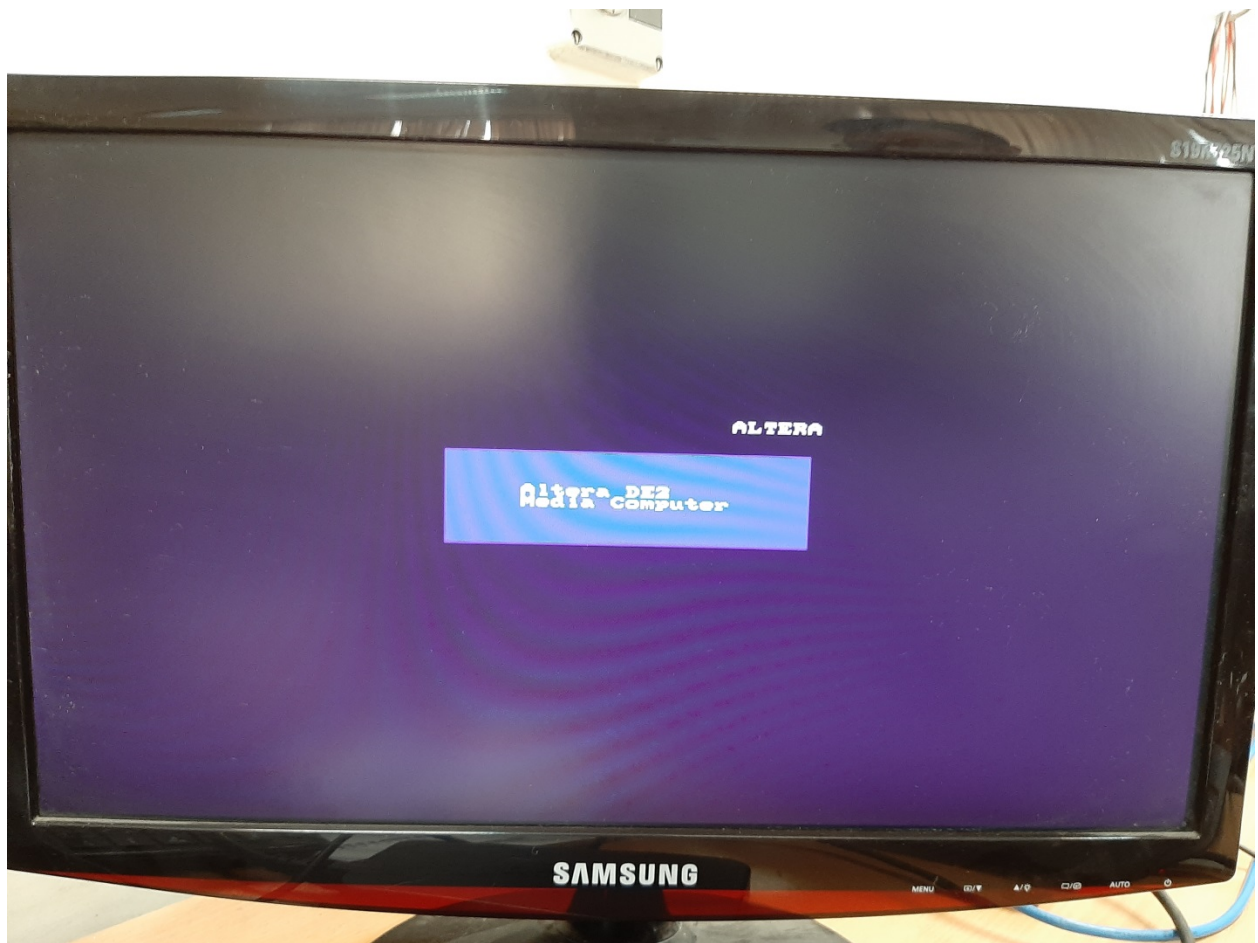
    /* also, display any PS/2 data (from its interrupt service routine) on HEX displays */
    HEX_PS2 (byte1, byte2, byte3);
    timeout = 0;
}
```

باتوجه به تصاویر فوق، در ابتدا بر روی مانیتور عبارات “Media Computer” و “Altera DE2” به صورت ثابت نمایش داده میشوند و عبارت “Altera” به صورت متحرک، مدام دور صفحه مانیتور حرکت می کند و نمایش داده می شود.

با فشردن کلید ۱، اولین LED سبز روشن شده و عملیات ضبط صوت بر روی پورت صدا، آغاز می شود و تا پر شدن بافر صدا، ادامه می یابد.

با فشردن کلید ۲، دومین LED سبز روشن شده و عملیات پخش صوت بر روی پورت صدا، آغاز میشود و تا پخش تمام بافر صدا، ادامه می یابد.





خواسته ها

Alt_up_dev (۱)

همان طور که مشاهده می شود

Alt_up_dev یک داده از نوع structure

است که در آن تعدادی data که از

نوع pointer هستند ، نگهداری می شود.

این Structure در فایل global.h تعریف

شده است تا در تمامی فایل ها هنگامی که

header ما include میشود بتوان از آن

استفاده کرد و به این pointer ها دسترسی داشت.

که pointer های مربوطه در عکس قابل مشاهده هستند.

این structure به توابع به عنوان input ، pass میشود که به مرابط میتوان در کد آن را مشاهده نمود و کار آن دسترسی دادن به device هایی است که قرار است از آن ها استفاده کرده و آنها را **open** کنیم.

Alt_up_parallel_port_dev → *key_dev → opens the pushbutton KEY parallel port

Alt_up_ps2_dev → ps2_dev → opens ps2 port

Alt_up_character_lcd_dev → lcd_dev → opens the 16x2 character display port

Alt_up_audio_dev → audio_dev → opens the audio port

Alt_up_char_buffer_dev → char_buffer_dev → output text message in the middle of the VGA monitor

Alt_up_pixel_buffer_dma_dev → pixel_buffer_dev → opens pixel buffer

```
18 struct alt_up_dev {
19     alt_up_parallel_port_dev *KEY_dev;
20     alt_up_parallel_port_dev *green_LEDs_dev;
21     alt_up_parallel_port_dev *red_LEDs_dev;
22     alt_up_ps2_dev *PS2_dev;
23     alt_up_character_lcd_dev *lcd_dev;
24     alt_up_audio_dev *audio_dev;
25     alt_up_char_buffer_dev *char_buffer_dev;
26     alt_up_pixel_buffer_dma_dev *pixel_buffer_dev;
27 };
```

(۲)

```
66  /* declare volatile pointer for interval timer, which does not have HAL functions */
67  volatile int * interval_timer_ptr = (int *) 0x10002000; // interval timer base address
```

پردازنده Nios II، می تواند تا ۳۲ وقفه را پشتیبانی کند. هر کدام از وقفه هایی که برای پردازنده تعریف می شود، تایمر خاص خود را دارند، یعنی مدت زمان مجاز، که هر کدام از وقفه ها نهایتاً می تواند از پردازنده استفاده کند، متفاوت است. به همین دلیل برای تایمر ۳۲ رجیستر در نظر گرفته شده است که مدت زمان مجاز هر وقفه، در این رجیسترها، مطابق با irq number متناظر آن وقفه، ذخیره می شود.

(۳-۴)

در PS2_ISR :

```
* Pushbutton - Interrupt Service Routine
*
* This routine checks which KEY has been pressed. If it is KEY1 or KEY2, it writes this
* value to the global variable key_pressed. If it is KEY3 then it loads the SW switch
* values and stores in the variable pattern
```

همان طور که در comment قابل مشاهده است ، این تابع وضعیت کلید ها را بررسی میکند و اگر کلید فشرده شده key1 یا key2 باشد ، این مقدار را در key_pressed که یک global variable است ، ذخیره میکند و اگر کلید فشرده شده key3 باشد ، توسط پروتکل نوشته شده در توابعی که در پایین آمده است مقدار sw که load شده است را ذخیره میکند.

```
void PS2_ISR(struct alt_up_dev *up_dev, unsigned int id)
```

```
(void) alt_up_ps2_write_data_byte (up_dev->PS2_dev, (unsigned char) 0xF4);
```

```
alt_up_ps2_read_data_byte (up_dev->PS2_dev, &PS2_data) == 0
```

در PUSHBUTTON_ISR :

```
* Pushbutton - Interrupt Service Routine
*
* This ISR checks which KEY has been pressed. If KEY1, then it enables audio-in
* interrupts (recording). If KEY2, it enables audio-out interrupts (playback).
```

طبق comment ها ، میتوان نتیجه گرفت که این ISR با توجه به وضعیت KEY ها ، interrupt های مربوطه فعال میکند که این interrupt ها به این صورت هستند:

KEY1 → recording interrupts

KEY2 → playback interrupts

توابع این function به صورت زیر است:

```
void pushbutton_ISR(struct alt_up_dev *up_dev, unsigned int id)
```

```
/* read the pushbutton interrupt register */
KEY_value = alt_up_parallel_port_read_edge_capture (up_dev->KEY_dev);
alt_up_parallel_port_clear_edge_capture (up_dev->KEY_dev); // Clear the interrupt
```

که در این قسمت رجیستر وقفه توسط تابع alt_up_parallel_port_read_edge_capture خوانده میشود.

```
alt_up_audio_enable_read_interrupt (audio_dev);
```

```
alt_up_audio_enable_write_interrupt (audio_dev);
```

```
alt_up_audio_dev *audio_dev;
```

```
audio_dev = up_dev->audio_dev;
```

و در این ۲ تابع هم interrupt های خواندن و نوشتن به پورت های صدا فعال میشوند.

در audio_ISR :

```
* Audio - Interrupt Service Routine
*
* This interrupt service routine records or plays back audio, depending on which type
* interrupt (read or write) is pending in the audio device.
```

همان طور که مشاهده می شود این تابع با توجه به نوع interrupt (read یا write) که داده شده است ، تصمیم گیری میکند.

```
if (alt_up_audio_read_interrupt_pending(up_dev->audio_dev)) // check for read interrupt
{
    if (alt_up_audio_write_interrupt_pending(up_dev->audio_dev)) // check for write interrupt
```

که در پایان هم با توجه به این که کدام interrupt فعال شده بوده است ، interrupt مربوطه را disable میکند:

```
alt_up_audio_disable_read_interrupt(up_dev->audio_dev);
...
alt_up_audio_disable_write_interrupt(up_dev->audio_dev);
```

نوع وقفه ها:

نرم افزاری → Audio_interrupt(Read_interrupt)

نرم افزاری → Audio_interrupt (write_interrupt)

سخت افزاری → Mouse_interrupt (PS2_ISR)

علت:

در Mouse_interrupt (PS2_ISR)، این ماوس است که با فرستادن ۳ بایت داده به پردازنده، باعث ایجاد وقفه در پردازنده می شود. از جایی که ارسال بایت از ماوس به پردازنده توسط سخت افزار ماوس و ارتباط سیمی ماوس با برد DE2 صورت می گیرد، این وقفه، از نوع سخت افزاری می باشد.

در Audio_interrupt با فشرده شده کلید، ISR مربوط به Push Button صدا زده می شود. سپس در PUSHBUTTON_ISR، با ۱ شدن flag مربوط به record و Play، ضبط و پخش صدا شروع می شود. پس در Audio_interrupt ، کد PUSHBUTTON_ISR است که وقفه را برای ضبط و پخش صدا ایجاد می کند. پس چون کد نرم افزاری، وقفه را ایجاد می کند، این وقفه، نرم افزاری می باشد.

تمامی توابع مربوطه به همراه آرگومان های آن ها و همچنین تعارف آرگومان ها در عکس ها مشخص شده اند.

```
/* use the HAL facility for registering interrupt service routines. */
/* Note: we are passing a pointer to up_dev to each ISR (using the context argument) as
 * a way of giving the ISR a pointer to every open device. This is useful because some of the
 * ISRs need to access more than just one device (e.g. the pushbutton ISR accesses both
 * the pushbutton device and the audio device) */
alt_irq_register (0, (void *) &up_dev, (void *) interval_timer_ISR);
alt_irq_register (1, (void *) &up_dev, (void *) pushbutton_ISR);
alt_irq_register (6, (void *) &up_dev, (void *) audio_ISR);
alt_irq_register (7, (void *) &up_dev, (void *) PS2_ISR);
```

برای register کردن از تابع alt_irq_register استفاده شده است که از سری توابع HAL است.

تعریف این تابع به شکل زیر است:

```
int alt_irq_register (alt_u32 id,
                     void* context,
                     alt_isr_func handler)
{
    int rc = -EINVAL;
    alt_irq_context status;

    if (id < ALT_NIRQ)
    {
        /*
         * interrupts are disabled while the handler tables are updated to ensure
         * that an interrupt doesn't occur while the tables are in an inconsistent
         * state.
         */

        status = alt_irq_disable_all ();

        alt_irq[id].handler = handler;
        alt_irq[id].context = context;

        rc = (handler) ? alt_irq_enable (id): alt_irq_disable (id);

        alt_irq_enable_all(status);
    }
    return rc;
}
```

که آرگومان های ما یک alt_u32 است که یک عدد 32b است و یک pointer که ما به آن مستقیماً آدرس محتویات up_dev را داده ایم و آخرین آرگومان آن handler آن است که با توجه به نوع handler مورد نیاز به آن داده شده است.

(۵)

```
/* globals used for audio record/playback */
extern volatile int buf_index_record, buf_index_play;
unsigned int l_buf[BUF_SIZE];           // audio buffer
unsigned int r_buf[BUF_SIZE];           // audio buffer
```

همان طور که در عکس قابل مشاهده است بافر های مورد نیاز برای audio ، ۲ buffer است که این buffer ها به صورت چپ و راست تعریف شده اند. این بافر ها به صورت static array به سبب buf_size تعریف شده اند.

همچنین دو index تعریف شده است که طول بافر را میپیماید و توسط آن به داده های موجود در این Array دسترسی پیدا میکنیم.

```
void audio_ISR(struct alt_up_dev *up_dev, unsigned int id)
{
    int num_read; int num_written;

    unsigned int fifospace;
```

در این تابع به مدیریت بافر با توجه به interrupt های آمده میپردازیم که این interrupt ها توسط structure alt_up_dev که به از جنس pointer است ، مشخص میگردند.

اگر interrupt برای read باشد:

```
if (alt_up_audio_read_interrupt_pending(up_dev->audio_dev)) // check for read interrupt
{
    alt_up_parallel_port_write_data (up_dev->green_LEDs_dev, 0x1); // set LEDG[0] on

    // store data until the buffer is full
    if (buf_index_record < BUF_SIZE)
    {
        num_read = alt_up_audio_record_r (up_dev->audio_dev, &(r_buf[buf_index_record]),
            BUF_SIZE - buf_index_record);
        /* assume we can read same # words from the left and right */
        (void) alt_up_audio_record_l (up_dev->audio_dev, &(l_buf[buf_index_record]),
            num_read);
        buf_index_record += num_read;

        if (buf_index_record == BUF_SIZE)
        {
            // done recording
            alt_up_parallel_port_write_data (up_dev->green_LEDs_dev, 0); // turn off LEDG
            alt_up_audio_disable_read_interrupt(up_dev->audio_dev);
        }
    }
}
```

اگر interrupt برای read فعال شده باشد ، در گام اول LEDG[0] روشن می شود.

ابتدا به تعریف alt_up_audio_record_l ویا alt_up_audio_record_r میپردازیم که مانند هم عمل میکنند و صرفا یکی بر روی r_buff و دیگری بر روی l_buff کار خود را انجام میدهد میپردازیم.

4.3.11 alt_up_audio_record_l

Prototype: unsigned int alt_up_audio_record_l(alt_up_audio_dev *audio, unsigned int *buf, int len)
Include: <altera_up_avalon_audio.h>
Parameters: audio – the audio device structure
buf – the pointer to the allocated memory for storing audio data. Size of buf should be no smaller than len words.
len – the number of data in words to read from the input FIFO
Returns: The total number of words read.
Description: Read len words of data from left input FIFO, if the FIFO is above a threshold, and store data to where buf points.

که کار این function آن است که به تعداد len word گفته شده ، data بخواند و در buffer داده شده ، آن را ذخیره کند.

پس ما همزمان داریم تعدادی از words را در l_buff و r_buff میخوانیم و به تعداد num_read تعداد data های خوانده شده است را با index که array ما را که همان buffer است میپیمایم ، جمع میکنیم و index را update میکنیم. سپس چک میکنیم که آیا بافر ما پر شده است یا نه. در صورت پر شدن بافر read_interrupt را disable می کنیم و LEDG[0] را خاموش میکنیم.

اگر interrupt برای write باشد:

```
if (alt_up_audio_write_interrupt_pending(up_dev->audio_dev)) // check for write interrupt
{
    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);
        }
    }
}
```

در اولین گام اگر write interrupt فعال شده باشد ، LEDG[1] را روشن میکنیم.

ابتدا به تعریف alt_up_audio_play_l ویا alt_up_audio_play_r میپردازیم که مانند هم عمل میکنند و صرفاً یکی بر روی r_buff و دیگری بر روی l_buff کار خود را انجام میدهد میپردازیم.

4.3.13 alt_up_audio_play_r

Prototype: unsigned int alt_up_audio_play_r (alt_up_audio_dev *audio, unsigned int *buf, int len)
Include: <altera_up_avalon_audio.h>
Parameters: audio – the audio device structure
buf – the pointer to the data to be written. Size of buf should be no smaller than len words.
len – the number of data in words to be written into the output FIFO
Returns: The total number of data written.
Description: Write len words of data into right output FIFO, if space available in FIFO is above a threshold.

به طور خلاصه این تابع به تعداد len words که به آن داده شده است ، data را بر روی بافر نوشته و مقدار number of written data را برمی گرداند.

پس ما همزمان داریم تعدادی data بر روی l_buff و r_buff مینویسیم و سپس index ای که برای نوشتن استفاده میشود را با تعداد data های نوشته شده یعنی num_written جمع میکنیم و buf_index_record را update میکنیم .

سپس چک میکنیم که آیا بافر Recording پر شده است یا نه. اگر شده بود ، write interrupt را disable میکنیم و LEDG[1] را خاموش میکنیم.

Source:

ftp://ftp.intel.com.br/Pub/fpgaup/pub/Intel_Material/12.1/Tutorials/HAL_tutorial.pdf

بخش ۲: راه اندازی درایور ماوس با خروجی PS/2

بدست آوردن وضعیت ماوس و دکمه های آن :

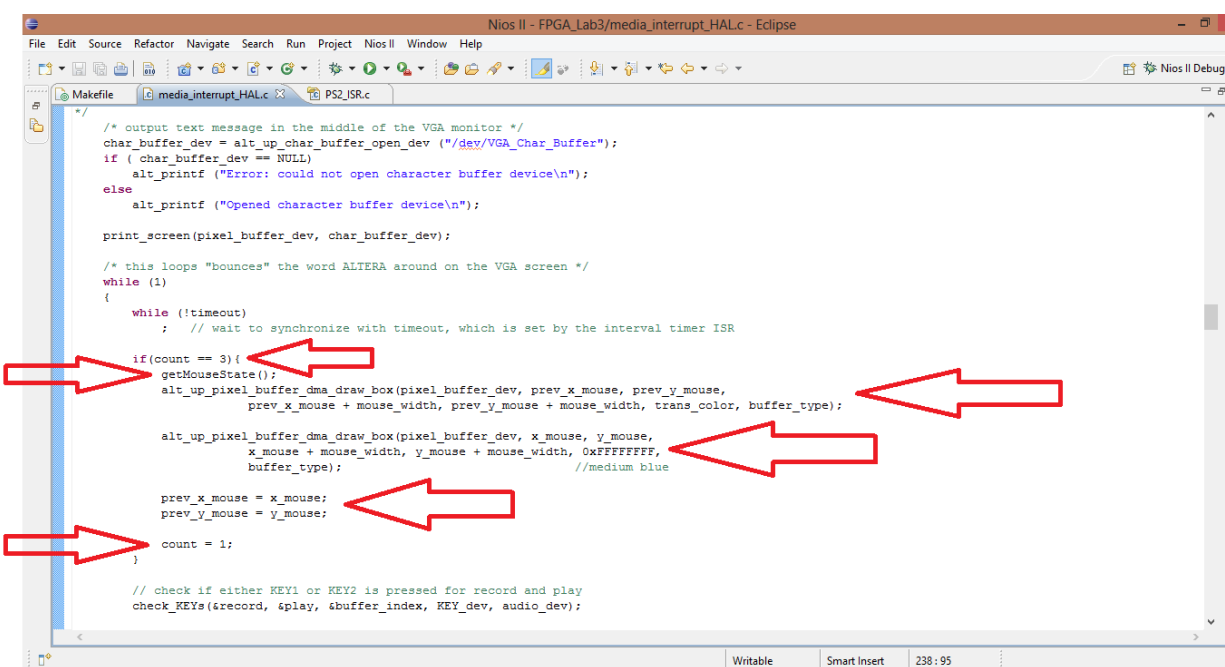
برای بدست آوردن موقعیت ماوس و وضعیت دکمه های آن، نیاز به ۳ بایت داده است، اما در هر وقفه ماوس فقط یک بایت داده ارسال می کند. به همین دلیل بعد از هر ۳ وقفه، همه ۳ بایت داده ماوس، دریافت می شود. مطابق با تصویر زیر، یک متغیر شمارنده به اسم count، تعریف می کنیم. در ابتدا مقدار آن را 0 می کنیم تا if مربوط به فعال کردن ماوس اجرا شود. بعد از آن مقدار count را 1 می کنیم تا از دوباره فعال کردن ماوس جلوگیری شود. (فعال کردن ماوس فقط یک بار و آن هم اول برنامه باید اجرا شود) هر بار که وقفه ماوس اجرا شود، مقدار count یک واحد افزایش می یابد تا مقدار آن به 3 برسد. در این حالت همه ۳ بایت داده ماوس دریافت و در متغیر های copy_byte ذخیره شدند. حال متغیر های اصلی (byte_1, byte_2, byte_3) را با مقادیر copy_byte1, copy_byte2, copy_byte3 که همان داده های ماوس هستند، مقدار دهی می کنیم.

```
void PS2_ISR(struct alt_up_dev *up_dev, unsigned int id)
{
    unsigned char PS2_data;

    /* check for PS/2 data--display on HEX displays */
    if (alt_up_ps2_read_data_byte (up_dev->PS2_dev, &PS2_data) == 0)
    {
        /* allows save the last three bytes of data */
        copy_byte1 = copy_byte2;
        copy_byte2 = copy_byte3;
        copy_byte3 = PS2_data;

        if(count > 0){
            if(count != 3){
                count++;
            }
            if(count == 3){
                byte1 = copy_byte1;
                byte2 = copy_byte2;
                byte3 = copy_byte3;
            }
        }
        if(count == 0){
            if ( (copy_byte2 == (unsigned char) 0xAA) && (copy_byte3 == (unsigned char) 0x00) ){
                // mouse inserted: initialize sending of data
                (void) alt_up_ps2_write_data_byte_with_ack (up_dev->PS2_dev, (unsigned char) 0xF4);
                count = 1;
            }
        }
    }
    return;
}
```

سپس در تابع main موجود در فایل media_interrupt_HAL.c، هرگاه که مقدار count ۳ شود، (مقادیر ۳ بایت ماوس byte_1,byte2,byte_3 معتبر باشد.)، با فراخوانی تابع getMouseState موقعیت ماوس آپدیت می شود. سپس با کشیدن یک مربع سیاه بر روی موقعیت قبلی ماوس، ماوس قبلی پاک می شود. در ادامه با رسم یک مربع سفید بر روی موقعیت جدید ماوس، ماوس در موقعیت جدیدش رسم می شود و چون هر دفعه موقعیت قبلی ماوس پاک می شود و ماوس در موقعیت جدیدش رسم می شود، ماوس در صفحه حرکت می کند. در آخر موقعیت قبلی ماوس نیز آپدیت و برابر با موقعیت فعلی ماوس می شود. مقدار count را هم ۱ می کنیم تا آپدیت شدن موقعیت ماوس و پاک کردن و رسم ماوس، پس از هر ۳ وقفه ماوس، اجرا شود تا مقادیر ۳ بایت ماوس معتبر باشد.



```

/*
 * output text message in the middle of the VGA monitor */
char_buffer_dev = alt_up_char_buffer_open_dev ("/dev/VGA_Char_Buffer");
if (char_buffer_dev == NULL)
    alt_printf ("Error: could not open character buffer device\n");
else
    alt_printf ("Opened character buffer device\n");

print_screen(pixel_buffer_dev, char_buffer_dev);

/* this loops "bounces" the word ALTERA around on the VGA screen */
while (1)
{
    while (!timeout)
        ; // wait to synchronize with timeout, which is set by the interval timer ISR

    if(count == 3){
        getMouseState();
        alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, prev_x_mouse, prev_y_mouse,
            prev_x_mouse + mouse_width, prev_y_mouse + mouse_width, trans_color, buffer_type);

        alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, x_mouse, y_mouse,
            x_mouse + mouse_width, y_mouse + mouse_width, 0xFFFFFFFF,
            buffer_type); //medium blue

        prev_x_mouse = x_mouse;
        prev_y_mouse = y_mouse;

        count = 1;
    }

    // check if either KEY1 or KEY2 is pressed for record and play
    check_KEYS(&record, &play, &buffer_index, KEY_dev, audio_dev);
}

```

دکمه چپ:

وضعیت دکمه چپ، در اولین بیت بایت اول قرار دارد. اگر باقی مانده تقسیم بایت اول را بر ۲ حساب کنیم، این بیت که نشان دهنده وضعیت کلیک شدن دکمه چپ ماوس است، بدست می آید (اگر اولین بیت ۱ باشد، بایت اول، فرد بوده و باقی مانده تقسیم بر ۲ هم ۱ می شود که برابر با این بیت هست. اگر اولین بیت ۰ باشد، بایت اول، زوج بوده و باقی مانده تقسیم آن بر ۲، ۰ می شود که برابر با این بیت است).

دکمه راست:

وضعیت دکمه راست، در دومین بیت بایت اول قرار دارد. اگر بایت اول را به میزان ۱ بیت، به سمت راست، شیفت دهیم، دومین بیت که نشان دهنده وضعیت دکمه راست است، در اولین بیت قرار می گیرد. سپس اگر باقی مانده تقسیم بایت اول را بر ۲ حساب کنیم، این بیت که نشان دهنده وضعیت کلیک شدن دکمه راست ماوس است، بدست می آید (اگر اولین بیت عدد شیفت یافته، ۱ باشد، عدد شیفت یافته فرد بوده و باقی مانده تقسیم آن بر ۲ هم ۱ می شود که برابر با این بیت هست. اگر اولین بیت ۰ باشد، عدد شیفت یافته، زوج بوده و باقی مانده تقسیم آن، بر ۲، ۰ می شود که برابر با این بیت است).

دکمه وسط:

وضعیت دکمه وسط، در سومین بیت بایت اول قرار دارد. اگر بایت اول را به میزان ۲ بیت، به سمت راست، شیفت دهیم، سومین بیت که نشان دهنده وضعیت دکمه وسط است، در اولین بیت قرار می گیرد. سپس اگر باقی مانده تقسیم بایت اول را بر ۲ حساب کنیم، این بیت که نشان دهنده وضعیت کلیک شدن دکمه وسط ماوس است، بدست می آید (اگر اولین بیت عدد شیفت یافته، ۱ باشد، عدد شیفت یافته فرد بوده و باقی مانده تقسیم آن بر ۲ هم ۱ می شود که برابر با این بیت هست. اگر اولین بیت ۰ باشد، عدد شیفت یافته، زوج بوده و باقی مانده تقسیم آن، بر ۲، ۰ می شود که برابر با این بیت است).

وضعیت ماوس:

برای بدست آوردن موقعیت افقی و عمودی ماوس، مطابق تصویر صفحه بعد، ابتدا علامت تغییرات (مثبت یا منفی بودن تغییرات افقی و عمودی) را مشخص میکنیم:

علامت تغییرات افقی:

علامت تغییرات افقی، در پنجمین بیت بایت اول قرار دارد. اگر بایت اول را به میزان ۴ بیت، به سمت راست، شیفت دهیم، پنجمین بیت که نشان دهنده علامت تغییرات افقی است، در اولین بیت قرار می گیرد. سپس اگر باقی مانده تقسیم بایت اول را بر ۲ حساب کنیم، این بیت که نشان دهنده علامت تغییرات افقی ماوس است، بدست می آید (اگر اولین بیت عدد شیفت یافته، ۱ باشد، عدد شیفت یافته فرد بوده و باقی مانده تقسیم آن بر ۲ هم ۱ می شود که برابر با این بیت هست. اگر اولین بیت 0 باشد، عدد شیفت یافته، زوج بوده و باقی مانده تقسیم آن، بر ۲، 0 می شود که برابر با این بیت است.)

علامت تغییرات عمودی:

علامت تغییرات عمودی، در ششمین بیت بایت اول قرار دارد. اگر بایت اول را به میزان ۵ بیت، به سمت راست، شیفت دهیم، ششمین بیت که نشان دهنده علامت تغییرات عمودی است، در اولین بیت قرار می گیرد. سپس اگر باقی مانده تقسیم بایت اول را بر ۲ حساب کنیم، این بیت که نشان دهنده علامت تغییرات عمودی ماوس است، بدست می آید (اگر اولین بیت عدد شیفت یافته، ۱ باشد، عدد شیفت یافته فرد بوده و باقی مانده تقسیم آن بر ۲ هم ۱ می شود که برابر با این بیت هست. اگر اولین بیت 0 باشد، عدد شیفت یافته زوج بوده و باقی مانده تقسیم آن، بر ۲، 0 می شود که برابر با این بیت است.)

تغییرات افقی و عمودی:

در آخر اگر علامت تغییرات مثبت بود، تغییرات افقی و عمودی موجود در بایت دوم و سوم را با مقدار افقی و عمودی قبلی ماوس جمع میکنیم در غیر این صورت تغییرات را از مقدار قبلی کم میکنیم.

```
void setMouseBounds(unsigned int x_max, unsigned int y_max)
{
    max_x_mouse = x_max;
    max_y_mouse = y_max;
}

void getMouseState()
{
    left_button_click = byte1 & 2; /* left button is in byte1[0] */
    right_button_click = (byte1 >> 1) & 2; /* right button is in byte1[0] */
    middle_button_click = (byte1 >> 2) & 2; /* middle button is in byte1[0] */

    /* x_sign is in byte1[4], 0 = + and 1 = - */
    int x_sign = ((byte1 >> 4) & 2) ? 1 : 0;
    //x_mouse = x_mouse + byte2;
    x_mouse = !x_sign ? x_mouse + ((byte2)) : x_mouse - (((byte2))); /* x_difference is in byte2 */
    x_mouse = (x_mouse >= max_x_mouse) ? max_x_mouse :
    (x_mouse <= 0) ? 0 :
    x_mouse;

    /* y_sign is in byte1[5], 0 = + and 1 = - */
    int y_sign = ((byte1 >> 5) & 2) ? 1 : 0;
    //y_mouse = y_mouse + byte3;
    y_mouse = !y_sign ? y_mouse + ((byte3)) : y_mouse - (((byte3))); /* y_difference is in byte3 */
    y_mouse = (y_mouse >= max_y_mouse) ? max_y_mouse :
    (y_mouse <= 0) ? 0 :
    y_mouse;
}

void print_screen(alt_up_pixel_buffer_dma_dev *pixel_buffer_dev, alt_up_char_buffer_dev *char_buffer_dev)
{
    int x1, x2, y1, y2;
    int color = 0x1863; // fill the screen with a dark grey color
    alt_up_pixel_buffer_dma_draw_box (pixel_buffer_dev, 0, 0, max_x_mouse,
```

:setMouseBounds

در این تابع، صرفاً متغیرهای سراسری `max_x_mouse`، `max_y_mouse` به ترتیب با ۳۱۹ و ۲۳۹ (ماکزیمم مختصات افقی و عمودی) مقداردهی می‌شوند.

```
else if (KEY_value == 0x4) // check KEY2
{
    // reset counter to start playback
    *counter = 0;
    alt_up_audio_reset_audio_core(audio_dev);
    *KEY2 = 1;
}

void setMouseBounds(unsigned int x_max, unsigned int y_max)
{
    max_x_mouse = x_max;
    max_y_mouse = y_max;
}

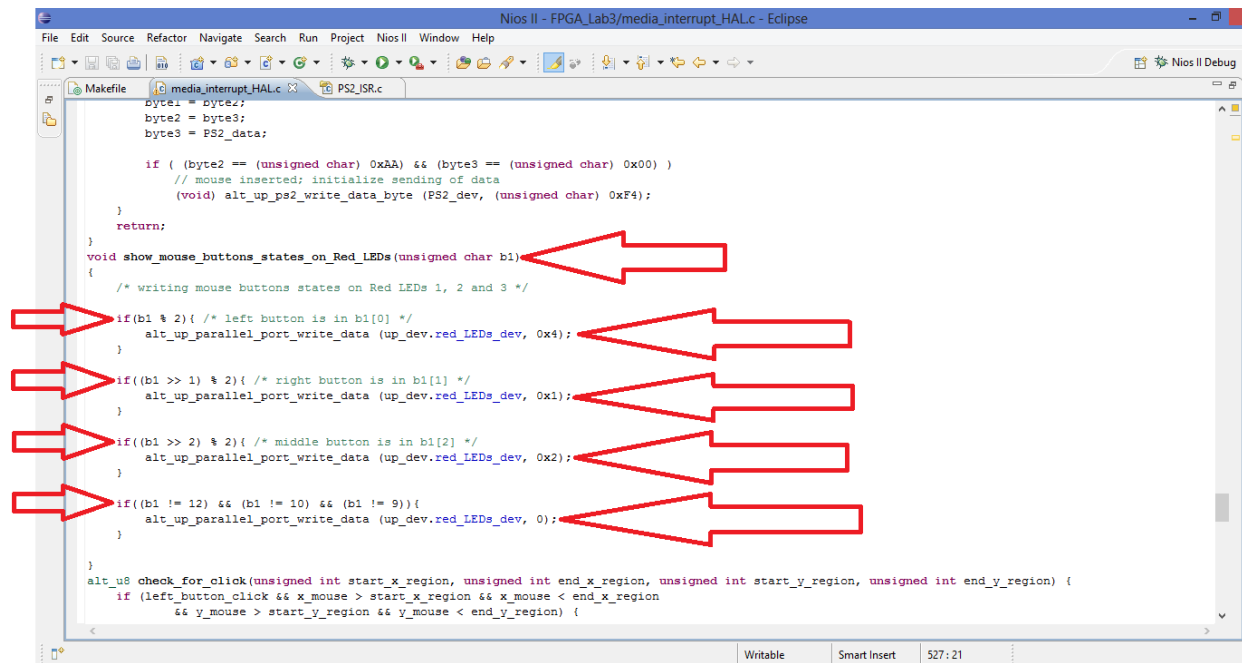
void getMouseState()
{
    left_button_click = byte1 & 2; /* left button is in byte1[0] */
    right_button_click = (byte1 >> 1) & 2; /* right button is in byte1[0] */
    middle_button_click = (byte1 >> 2) & 2; /* middle button is in byte1[0] */

    /* x_sign is in byte1[4], 0 = + and 1 = - */
    int x_sign = ((byte1 >> 4) & 2) ? 1 : 0;
    //x_mouse = x_mouse + byte2;
    x_mouse = !x_sign ? x_mouse + ((byte2)) : x_mouse - (((byte2))); /* x_difference is in byte2 */
    x_mouse = (x_mouse >= max_x_mouse) ? max_x_mouse :
    (x_mouse <= 0) ? 0 :
    x_mouse;

    /* y_sign is in byte1[5], 0 = + and 1 = - */
    int y_sign = ((byte1 >> 5) & 2) ? 1 : 0;
    //y_mouse = y_mouse + byte3;
    y_mouse = !y_sign ? y_mouse + ((byte3)) : y_mouse - (((byte3))); /* y_difference is in byte3 */
    y_mouse = (y_mouse >= max_y_mouse) ? max_y_mouse :
    (y_mouse <= 0) ? 0 :
    y_mouse;
}
```

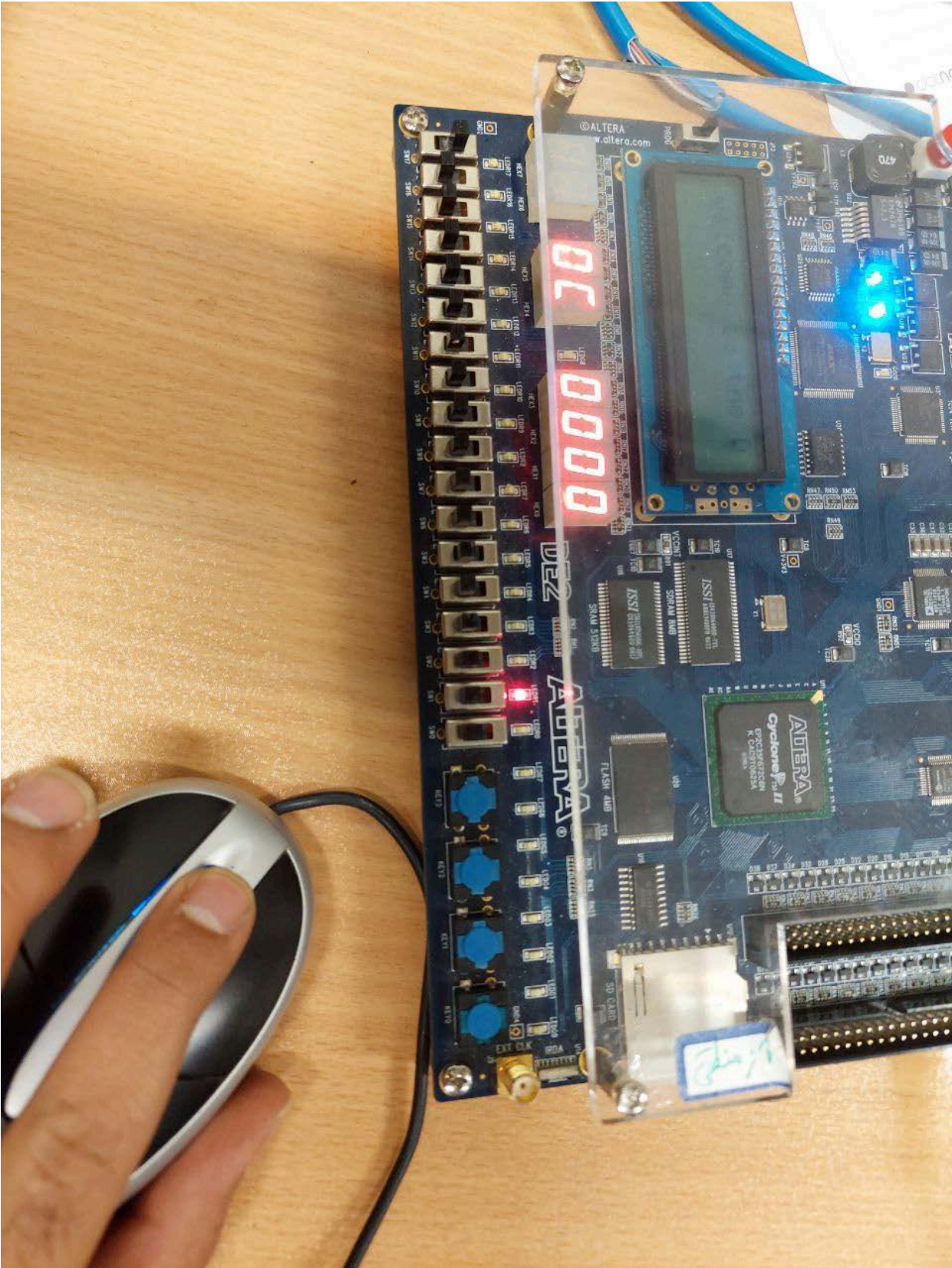
نمایش وضعیت ۳ دکمه ماوس بر روی LED های قرمز برد DE2:

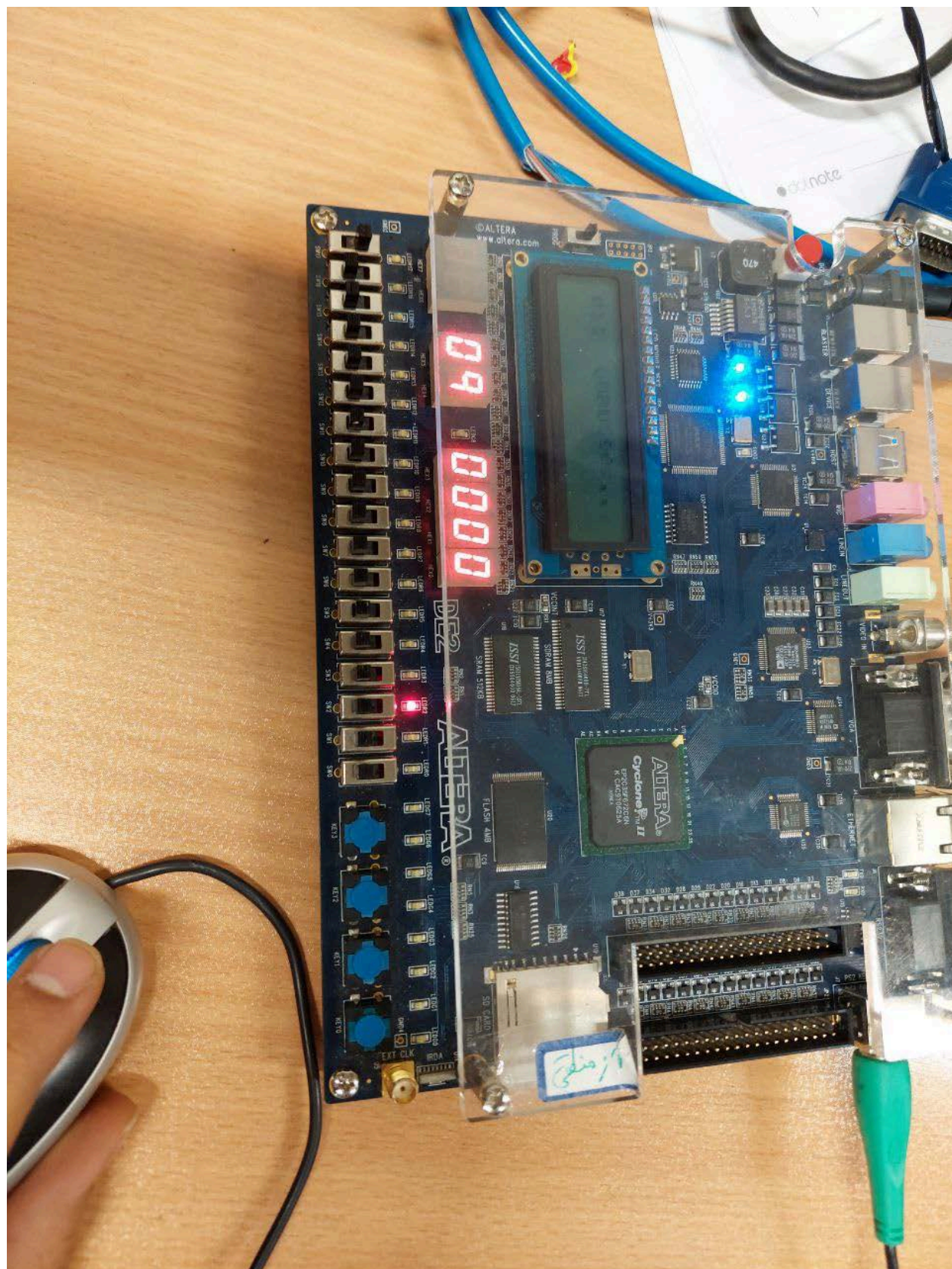
با بررسی وضعیت بیت های اول، دوم و سوم بایت اول (شیفت دادن بایت اول به میزانی که بیت اول، دوم و سوم در اولین بیت قرار گیرد، سپس باقی مانده تقسیم عدد شیفت یافته بر ۲) متناسب با اینکه کدام بیت ۱ است، یکی از LED های قرمز روشن می شود، در آخر اگر هیچ کدام از ۳ بیت، ۱ نبودند LED قرمزی که روشن شده بود، خاموش می شود.



```
void show_mouse_buttons_states_on_Red_LEDs(unsigned char b1)
{
    /* writing mouse buttons states on Red LEDs 1, 2 and 3 */
    if(b1 & 2){ /* left button is in b1[0] */
        alt_up_parallel_port_write_data (up_dev.red_LEDs_dev, 0x4);
    }
    if((b1 >> 1) & 2){ /* right button is in b1[1] */
        alt_up_parallel_port_write_data (up_dev.red_LEDs_dev, 0x1);
    }
    if((b1 >> 2) & 2){ /* middle button is in b1[2] */
        alt_up_parallel_port_write_data (up_dev.red_LEDs_dev, 0x2);
    }
    if((b1 != 12) && (b1 != 10) && (b1 != 9)){
        alt_up_parallel_port_write_data (up_dev.red_LEDs_dev, 0);
    }

    alt_u8 check_for_click(unsigned int start_x_region, unsigned int end_x_region, unsigned int start_y_region, unsigned int end_y_region) {
        if (left_button_click && x_mouse > start_x_region && x_mouse < end_x_region
            && y_mouse > start_y_region && y_mouse < end_y_region) {
```



نمایش وضعیت ماوس بر روی HEX Displayer های برد DE2:

```
Nios II - FPGA_Lab3/media_interrupt_HAL.c - Eclipse
File Edit Source Refactor Navigate Search Run Project Nios II Window Help

Makefile media_interrupt_HAL.c PS2_ISR.c

}

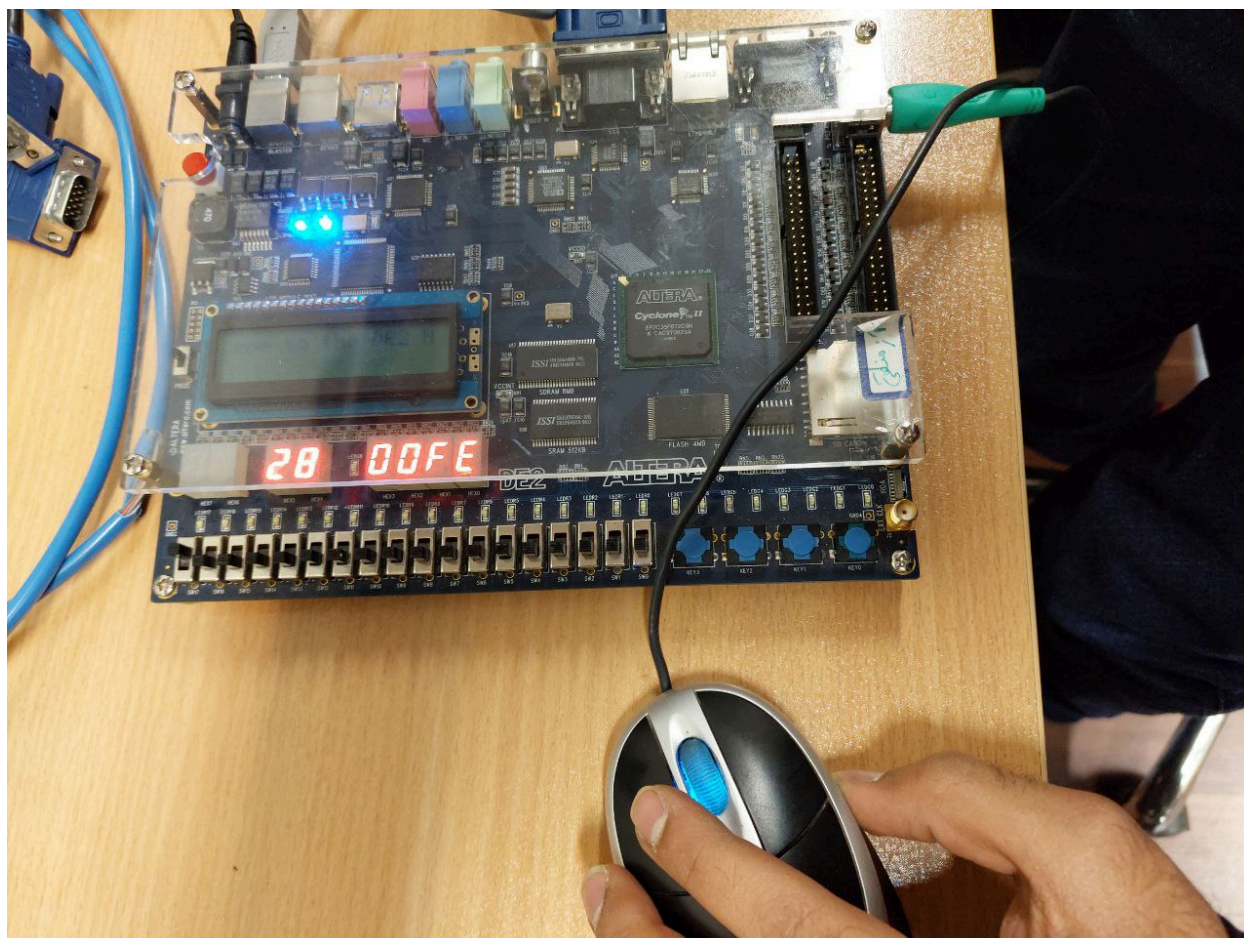
/* Subroutine to show a string of HEX data on the HEX displays
 * Note that we are using pointer accesses for the HEX displays parallel port. We could
 * also use the HAL functions for these ports instead
 */
void HEX_PS2(unsigned char b1, unsigned char b2, unsigned char b3)
{
    volatile int *HEX3_HEX0_ptr = (int *) 0x10000020;
    volatile int *HEX7_HEX4_ptr = (int *) 0x10000030;

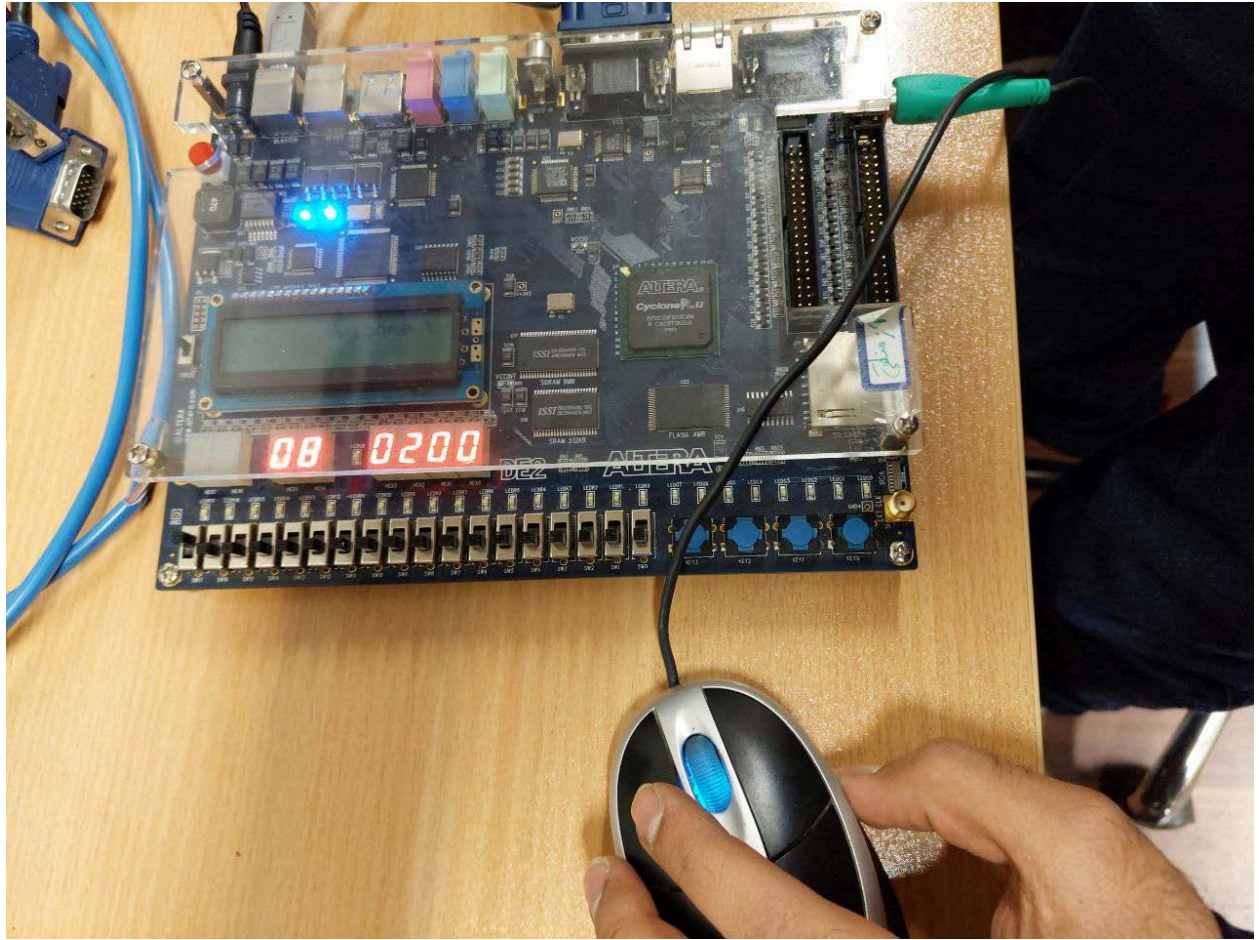
    /* SEVEN_SEGMENT_DECODE_TABLE gives the on/off settings for all segments in
     * a single 7-seg display in the DE2 Media Computer, for the hex digits 0 - F */
    unsigned char seven_seg_decode_table[] = { 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x07,
                                                0x7F, 0x67, 0x77, 0x7C, 0x39, 0x5E, 0x79, 0x71 };

    unsigned char hex_segs[] = { 0, 0, 0, 0, 0, 0, 0, 0 };
    unsigned int shift_buffer, nibble;
    unsigned char code;
    int i;

    shift_buffer = (b1 << 16) | (b2 << 8) | b3;
    for ( i = 0; i < 6; ++i )
    {
        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);
}

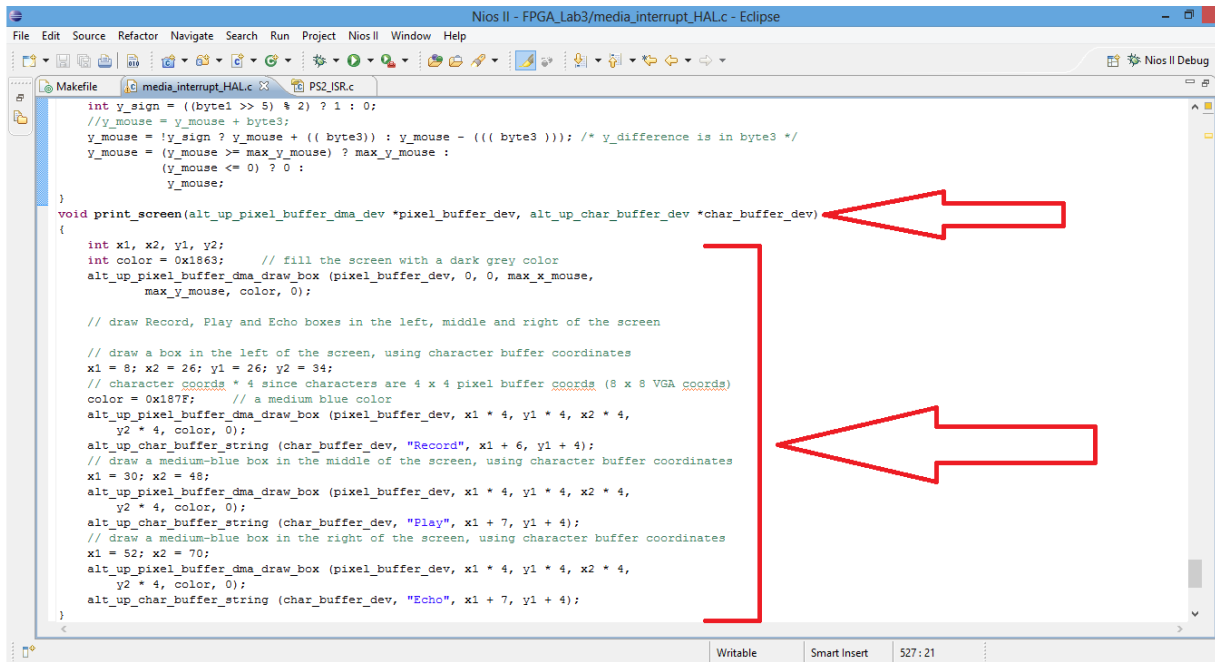
Writable Smart Insert 527:21
```





چاپ کردن صفحه نمایش بر صفحه مانیتور:

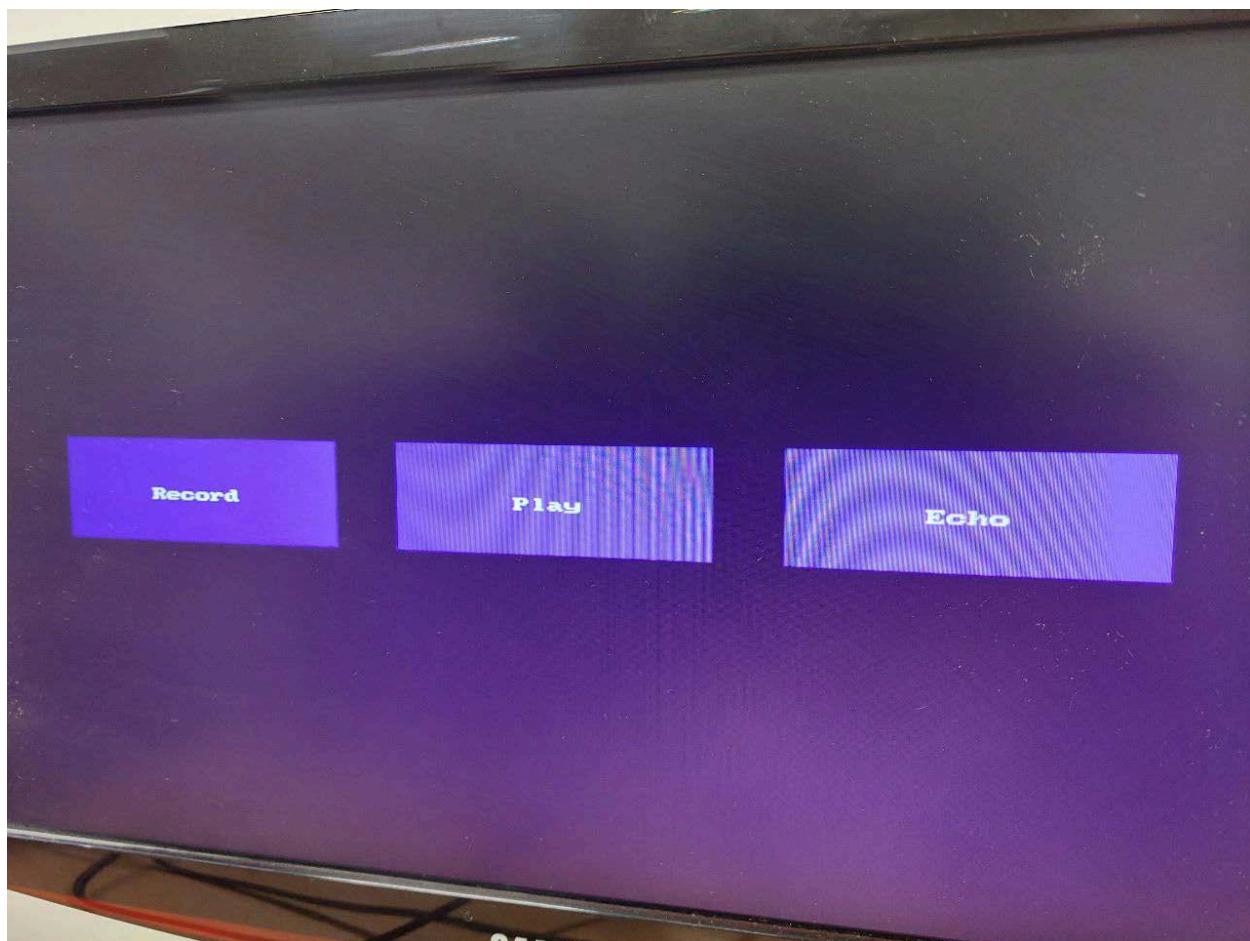
ابتدا کل صفحه را با رنگ بنفش پر می کنیم. سپس ۳ مستطیل با رنگ آبی ملایم، رسم می کنیم. در آخر در مرکز ۳ مستطیل رسم شده به ترتیب عبارت های Record, Play, Echo را چاپ می کنیم.

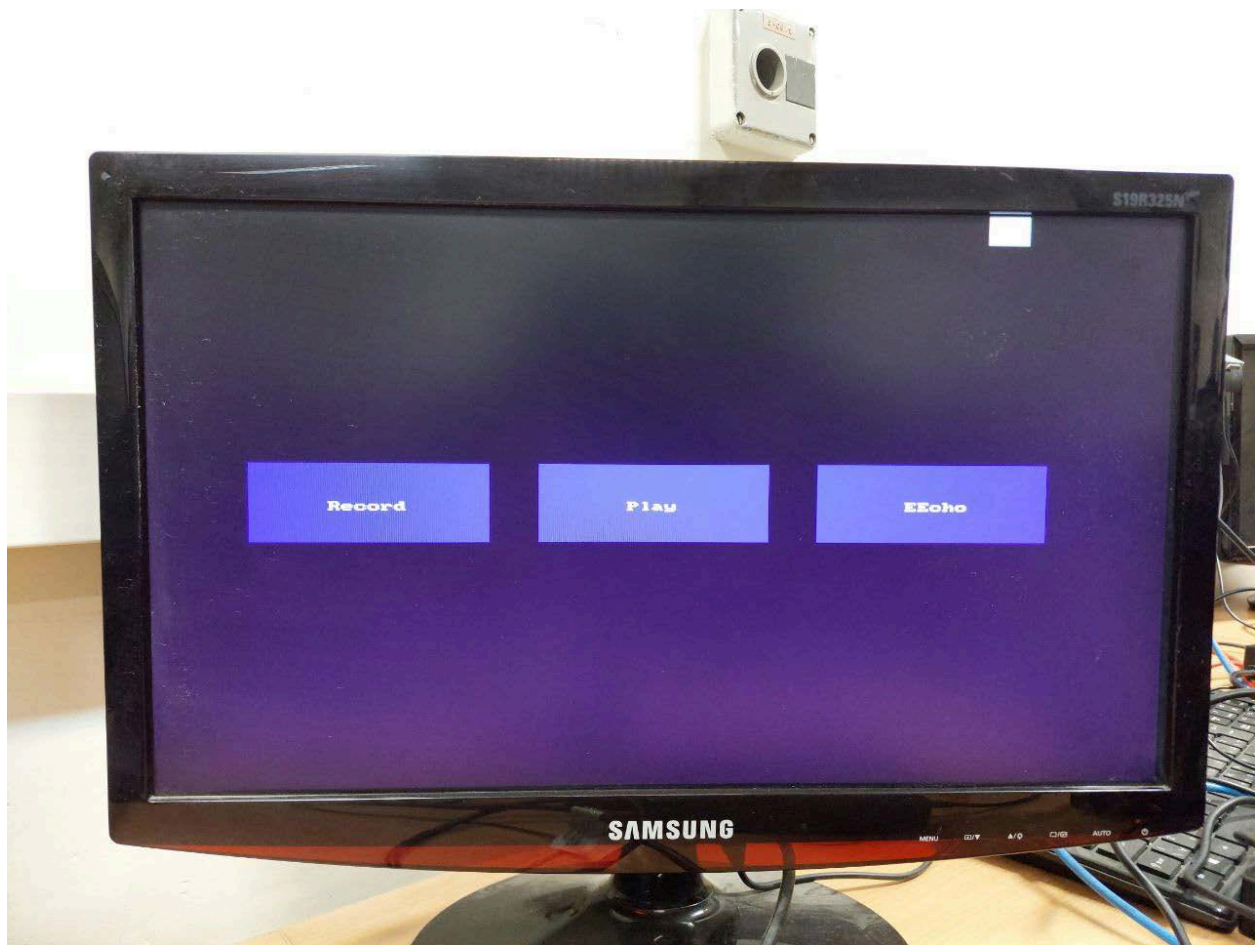


```
int y_sign = ((byte1 >> 5) % 2) ? 1 : 0;
//y_mouse = y_mouse + byte3;
y_mouse = !y_sign ? y_mouse + ((byte3)) : y_mouse - (((byte3))); /* y_difference is in byte3 */
y_mouse = (y_mouse >= max_y_mouse) ? max_y_mouse :
(y_mouse <= 0) ? 0 :
y_mouse;
}
void print_screen(alt_up_pixel_buffer_dma_dev *pixel_buffer_dev, alt_up_char_buffer_dev *char_buffer_dev)
{
    int x1, x2, y1, y2;
    int color = 0x1863; // fill the screen with a dark grey color
    alt_up_pixel_buffer_dma_draw_box (pixel_buffer_dev, 0, 0, max_x_mouse,
    max_y_mouse, color, 0);

    // draw Record, Play and Echo boxes in the left, middle and right of the screen

    // draw a box in the left of the screen, using character buffer coordinates
    x1 = 8; x2 = 26; y1 = 26; y2 = 34;
    // character coords * 4 since characters are 4 x 4 pixel buffer coords (8 x 8 VGA coords)
    color = 0x187F; // a medium blue color
    alt_up_pixel_buffer_dma_draw_box (pixel_buffer_dev, x1 * 4, y1 * 4, x2 * 4,
    y2 * 4, color, 0);
    alt_up_char_buffer_string (char_buffer_dev, "Record", x1 + 6, y1 + 4);
    // draw a medium-blue box in the middle of the screen, using character buffer coordinates
    x1 = 30; x2 = 48;
    alt_up_pixel_buffer_dma_draw_box (pixel_buffer_dev, x1 * 4, y1 * 4, x2 * 4,
    y2 * 4, color, 0);
    alt_up_char_buffer_string (char_buffer_dev, "Play", x1 + 7, y1 + 4);
    // draw a medium-blue box in the right of the screen, using character buffer coordinates
    x1 = 52; x2 = 70;
    alt_up_pixel_buffer_dma_draw_box (pixel_buffer_dev, x1 * 4, y1 * 4, x2 * 4,
    y2 * 4, color, 0);
    alt_up_char_buffer_string (char_buffer_dev, "Echo", x1 + 7, y1 + 4);
}
```



طراحی Audio Player با نمایش گرافیکی و ایجاد Echo

پس از فشردن شدن کلید ۱ بر روی برد، flag مربوط به record، ۱ شده و ISR مربوط به ضبط، اجرا می شود.

سپس اولین LED سبز به منظور نشان دادن شروع عملیات ضبط کردن، روشن می شود. سپس داده

دریافت شده در پورت های صدا راست و چپ خوانده و در بافر right_buf و left_buf ذخیره میشوند.

این روند تا زمانی که بافر صدا پر شود ادامه می یابد. در آخر flag، مربوط به ضبط، 0 شده و اجرا

وقفه مربوط به ضبط صدا، خاتمه می یابد. در انتهای این ISR، با فراخوانی تابع echo_maker، اکو یافته

صدای ضبط شده تولید می شود.

*** اجرای record:

```
if (record)
{
    printf("record is running\n");
    alt_up_parallel_port_write_data(green_LEDs_dev, 0x1); // set LEDG[0] on

    // record data until the buffer is full
    if (buffer_index < BUF_SIZE)
    {
        num_read = alt_up_audio_record_r(audio_dev, &(right_buf[buffer_index]),
                                          BUF_SIZE - buffer_index);

        (void)alt_up_audio_record_l(audio_dev, &(left_buf[buffer_index]),
                                     num_read);
        buffer_index += num_read;
        printf("num:%d\n", num_read);

        if (buffer_index == BUF_SIZE)
        {
            printf("done record\n");
            // done recording
            buffer_index = 0;
            record = 0;
            alt_up_parallel_port_write_data(green_LEDs_dev, 0x0); // set LEDG off
            echo_maker(left_buf, right_buf, echo_left_buf, echo_right_buf);
            //echo =1;
        }
    }
}
```


پس از فشردن شدن کلید ۲ بر روی برد، flag مربوط به play، ۱ شده و ISR مربوط به پخش، اجرا می شود. سپس دومین LED سبز به منظور نشان دادن شروع عملیات پخش کردن، روشن می شود. سپس داده ذخیره شده در بافر های صدا راست و چپ، خوانده و پخش می شوند. این روند تا رسیدن به انتهای بافر صدا ادامه می یابد. در آخر flag، مربوط به پخش، 0 شده و اجرا وقفه مربوط به پخش صدا، خاتمه می یابد.

**** اجرای play:**

```
else if (play)
{
    printf("play is running\n");
    alt_up_parallel_port_write_data(green_LEDs_dev, 0x2); // set LEDG[1] on

    // output data until the buffer is empty
    if (buffer_index < BUF_SIZE)
    {
        num_written = alt_up_audio_play_r(audio_dev, &(right_buf[buffer_index]),
            BUF_SIZE - buffer_index);

        (void)alt_up_audio_play_l(audio_dev, &(left_buf[buffer_index]),
            num_written);
        buffer_index += num_written;

        if (buffer_index == BUF_SIZE)
        {
            printf("done play\n");
            // done playback
            buffer_index = 0;
            play = 0;
            alt_up_parallel_port_write_data(green_LEDs_dev, 0x0); // set LEDG off
        }
    }
}
```

***اجرای echo :

تابع `echo_maker` تغییر ایجاد شده روی ورودی برای اینکه اکو شده به نظر بیاید را ایجاد میکند.

در واقع این تابع:

نمونه های قبل از ۱۰۰۰ رو تغییر نمی دهد.

نمونه های بین ۱۰۰۰ تا ۵۰۰۰ را با نصف آن نمونه (یک شیفت به راست) و یک چهارم (دو شیفت به راست) ۱۰۰۰ نمونه قبلی آن جمع می کند.

از نمونه های ۵۰۰۰ به بعد را، با نصف آن نمونه (یک شیفت به راست) و یک چهارم (دو شیفت به راست) ۵۰۰۰ نمونه قبلی آن جمع می کند.

```
void echo_maker(unsigned int left_buf[], unsigned int right_buf[], unsigned int* echo_left_buf, unsigned int* echo_right_buf) {
    int i;
    // ALPHA = 0.5 is >> 1, BETA = 0.25 is >> 2
    for (i = 0; i < BUF_SIZE; ++i) {
        if (i >= ECHO_INDEX1) {
            if (i >= ECHO_INDEX2) {
                echo_left_buf[i] = left_buf[i] >> 1 + left_buf[i - ECHO_INDEX2] >> 2;
                echo_right_buf[i] = right_buf[i] >> 1 + right_buf[i - ECHO_INDEX2] >> 2;
            }
            else {
                echo_left_buf[i] = left_buf[i] >> 1 + left_buf[i - ECHO_INDEX1] >> 2;
                echo_right_buf[i] = right_buf[i] >> 1 + right_buf[i - ECHO_INDEX1] >> 2;
            }
        }
        else {
            echo_left_buf[i] = left_buf[i];
            echo_right_buf[i] = right_buf[i];
        }
    }
}
```

اجرای آن:

```
else if (echo) {

    printf("echo is running\n");
    alt_up_parallel_port_write_data(green_LEDs_dev, 0x3); // set LEDG[2] on

    // output data until the buffer is empty
    if (buffer_index < BUF_SIZE)
    {
        num_written = alt_up_audio_play_r(audio_dev, &(echo_right_buf[buffer_index]),
            BUF_SIZE - buffer_index);

        (void)alt_up_audio_play_l(audio_dev, &(echo_left_buf[buffer_index]),
            num_written);
        buffer_index += num_written;

        if (buffer_index == BUF_SIZE)
        {
            printf("done echo\n");
            // done playback
            buffer_index = 0;
            echo = 0;
            alt_up_parallel_port_write_data(green_LEDs_dev, 0x0); // set LEDG off
        }
    }
}
```