DE2-115 Music Player

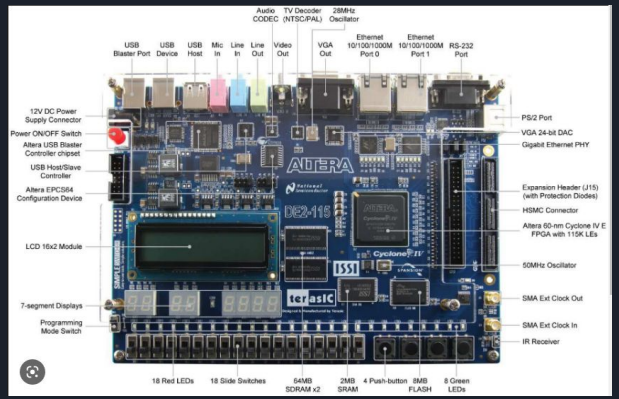
ECE 178 Embedded Systems
Group members: Puya Fard, Zoe Statzer, Jason Lawrence Alsola
Project Presentation

Introduction

Problem Statement: Creating a music player from FPGA DE2-115 NIOS 2 processor to expand our knowledge in areas of SD card, PIO, Timer, Interrupt, and Audio CODEC hardware components as well as entertainment!

Division of Tasks to approach full design:

1. Identify a block diagram with all components of input and output
2. Hardware - Create a new QSYS system required for our project
3. Software - Start coding required modules and blocks
4. Complete the system



Block Diagram and Design

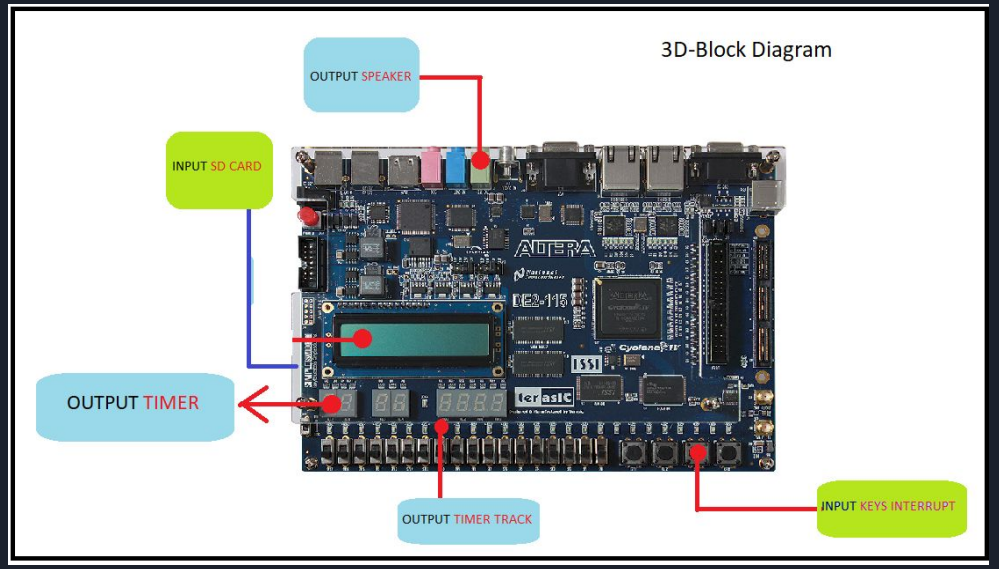
Inputs:
1. Pushbuttons for use with an ISR to skip or play/pause
2. SD card port, to read songs through FIFO buffer from memory

Outputs:
1. Hex 7-segment display for song time counter
2. LEDR display for progress bar
3. Audio via audio CODEC

Memory:
1. SD Card memory
2. SDRAM memory

Timer:
1. Will be used for counter

Interrupts:
1. Pushbutton 1, skip to next song
2. Pushbutton 2, pause or play current song
3. Pushbutton 3, skip to previous song

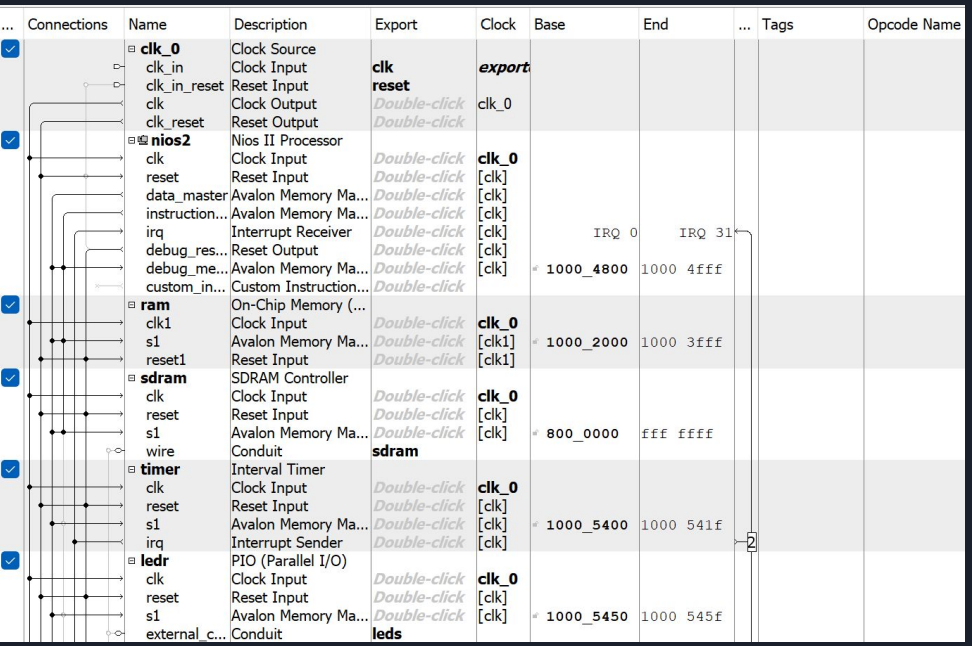


Tasks and Approach & Code logic explanation

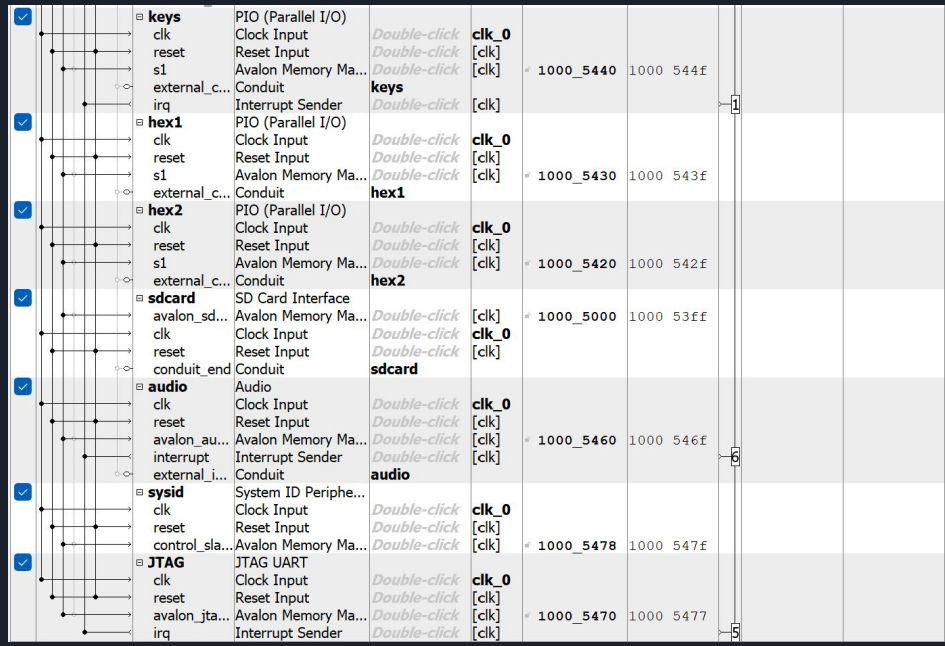
1. **Create QSYS System**

- Initialize Interval Timer
- Initialize KEYS Interrupt
  - KEYS will do PAUSE/PLAY, NEXT, PREVIOUS functions
- Program the LED Progress Bar
- Program the Song Runtime on the Seven-Segment Display
- Program and Integrate the SD Card to be read
- Program and Integrate the Audio CODEC

System Hardware Design and Qsys



Connections	Name	Description	Export	Clock	Base	End	Tags	Opcode Name
	clk_0	Clock Source						
	clk_in	Clock Input	**clk**	***export***				
	clk_in_reset	Reset Input	**reset**					
	clk	Clock Output	*Double-click*	clk_0				
	clk_reset	Reset Output	*Double-click*					
	nios2	Nios II Processor						
	clk	Clock Input	*Double-click*	**clk_0**				
	reset	Reset Input	*Double-click*	[clk]				
	data_master	Avalon Memory Ma...	*Double-click*	[clk]				
	instruction...	Avalon Memory Ma...	*Double-click*	[clk]				
	irq	Interrupt Receiver	*Double-click*	[clk]	IRQ 0	IRQ 31		
	debug_res...	Reset Output	*Double-click*	[clk]				
	debug_me...	Avalon Memory Ma...	*Double-click*	[clk]	1000_4800	1000 4fff		
	custom_in...	Custom Instruction...	*Double-click*					
	ram	On-Chip Memory (...						
	clk1	Clock Input	*Double-click*	**clk_0**				
	s1	Avalon Memory Ma...	*Double-click*	[clk1]	1000_2000	1000 3fff		
	reset1	Reset Input	*Double-click*	[clk1]				
	sdram	SDRAM Controller						
	clk	Clock Input	*Double-click*	**clk_0**				
	reset	Reset Input	*Double-click*	[clk]				
	s1	Avalon Memory Ma...	*Double-click*	[clk]	800_0000	fff ffff		
	wire	Conduit	**sdram**					
	timer	Interval Timer						
	clk	Clock Input	*Double-click*	**clk_0**				
	reset	Reset Input	*Double-click*	[clk]				
	s1	Avalon Memory Ma...	*Double-click*	[clk]	1000_5400	1000 541f		
	irq	Interrupt Sender	*Double-click*	[clk]				
	ledr	PIO (Parallel I/O)						
	clk	Clock Input	*Double-click*	**clk_0**				
	reset	Reset Input	*Double-click*	[clk]				
	s1	Avalon Memory Ma...	*Double-click*	[clk]	1000_5450	1000 545f		
	external_c...	Conduit	**leds**					



keys	PIO (Parallel I/O)					
clk	Clock Input	*Double-click*	**clk_0**			
reset	Reset Input	*Double-click*	[clk]			
s1	Avalon Memory Ma...	*Double-click*	[clk]	1000_5440	1000 544f	
external_c...	Conduit	**keys**				
irq	Interrupt Sender	*Double-click*	[clk]			1
hex1	PIO (Parallel I/O)					
clk	Clock Input	*Double-click*	**clk_0**			
reset	Reset Input	*Double-click*	[clk]			
s1	Avalon Memory Ma...	*Double-click*	[clk]	1000_5430	1000 543f	
external_c...	Conduit	**hex1**				
hex2	PIO (Parallel I/O)					
clk	Clock Input	*Double-click*	**clk_0**			
reset	Reset Input	*Double-click*	[clk]			
s1	Avalon Memory Ma...	*Double-click*	[clk]	1000_5420	1000 542f	
external_c...	Conduit	**hex2**				
sdcard	SD Card Interface					
avalon_sd...	Avalon Memory Ma...	*Double-click*	[clk]	1000_5000	1000 53ff	
clk	Clock Input	*Double-click*	**clk_0**			
reset	Reset Input	*Double-click*	[clk]			
conduit_end	Conduit	**sdcard**				
audio	Audio					
clk	Clock Input	*Double-click*	**clk_0**			
reset	Reset Input	*Double-click*	[clk]			
avalon_au...	Avalon Memory Ma...	*Double-click*	[clk]	1000_5460	1000 546f	
interrupt	Interrupt Sender	*Double-click*	[clk]			6
external_i...	Conduit	**audio**				
sysid	System ID Periphe...					
clk	Clock Input	*Double-click*	**clk_0**			
reset	Reset Input	*Double-click*	[clk]			
control_sla...	Avalon Memory Ma...	*Double-click*	[clk]	1000_5478	1000 547f	
JTAG	JTAG UART					
clk	Clock Input	*Double-click*	**clk_0**			
reset	Reset Input	*Double-click*	[clk]			
avalon_jta...	Avalon Memory Ma...	*Double-click*	[clk]	1000_5470	1000 5477	
irq	Interrupt Sender	*Double-click*	[clk]			5

Interval TIMER initialization

Important things to keep in mind:

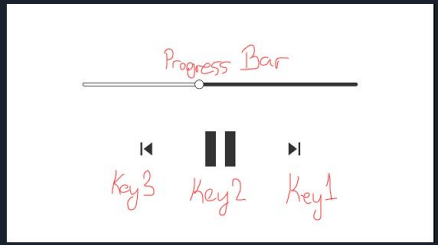
1. Set the Period correctly to 1 sec interval
2. START and STOP the timer according to the system flow
3. Interconnect the system to work with KEYS Interrupt

```
// Timer configuration //
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_BASE, 0b1000); // Initial stop
    IOWR_ALTERA_AVALON_TIMER_PERIODH(TIMER_BASE, 0x02FA); // Top half of 50,000,000
    IOWR_ALTERA_AVALON_TIMER_PERIODL(TIMER_BASE, 0xF080); // Bottom half of 50,000,000
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_BASE, 0b0110); // Start, Continuous


    // May want to export the following code to a function //
```

KEYS Interrupt initialization

1. Continuously check if the corresponding KEY is pressed
   a. KEY3: skip to previous song and load it to the board to play.
   b. KEY2: play and pause the current loaded song.
   c. KEY1: skip to next song and load it to the board to play.




```
    else if (*edge_capture_ptr == 0b0100) // Assuming the Play/Pause button is KEY2
        // Add Play functionality later
    {
        if (paused == 0) // If not paused, pauses
        {
                paused = 1;

                // Stops Timer, thereby stopping normal operations but not the PC from checking Pushbuttons again
                IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_BASE, 0b1000);
                IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_BASE, 0b00);

                // !!! Insert whatever other code needs to execute and pause here !!! //

                IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0x00); // Resets the Pushbutton context; IMPORTANT
        }
        else // If paused, plays
        {
                paused = 0;

                IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_BASE, 0b0110);

                // !!! Insert whatever other code needs to execute and play here !!! //

                IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0x00); // Resets the Pushbutton context; IMPORTANT
        }
    }
else if (*edge_capture_ptr == 0b1000) // Previous song KEY3
        // Add Play functionality later
{
```

LEDR Progress bar implementation

Important things to keep in mind:

1. Do the correct partition required to divide the period of the time equally to 18 LEDs.
2. Create a function that will continuously write the correct partition required as the song progresses.
3. Make sure to reset after finishing the progress bar.

```
TOBit = IORD_ALTERA_AVALON_TIMER_STATUS(TIMER_BASE) & 0b0001;
if (TOBit == 0b0001) // Meaning a whole second has passed
{

        scrub++;
        if (scrub < totalSecs) // As long as the timer is still within the song playing
        {
                if (scrub >= partitions) // If a threshold has passed and an LED needs to light up.
                {
                                scrub = 0; // Resets scrub so as to keep partitions static
                                progress = (progress * 2) + 1; // Shifts left, then keeps the previous LEDs lit
                                IOWR_ALTERA_AVALON_PIO_DATA(LEDR_BASE, progress); // Updates LEDs
                }
                //IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_BASE, 0b00); // Resets TO bit to continue operations; IMPORTANT
        }
        else // When a song has completely elapsed
        {
                break; // Exit loop, change later
                // Will probably want to reset all the variables used after the song is finished
        }
```

HEX Display Counter

Important things to keep in mind:

1. Close and reset all HEX display before starting and set it to 0.
2. Create the corresponding masking array that will have the corresponding counter.
3. Create a loop that will continuously count++ and write to HEX1 and HEX2 display locations.

Masking array:

```
// Song Run Time Variables //
int min[]={2139062080,2139062137,2139062052,2139062064,2139062041,2139062034}; //minutes 0-5

int sec[]={1077968767,1081704319,1076133759,1076920191,1075412863,1074954111,1073905535,
        1081638783,1073774463,1074823039,2034270079,2038005631,2032435071,2033221503,
        2031714175,2031255423,2030206847,2037940095,2030075775,2031124351,608206719,
        611942271,606371711,607158143,605650815, 605192063,604143487,611876735,604012415,
        605060991,809533311,813268863,807698303,808484735,806977407,806518655,805470079,
        813203327,805339007,806387583,423657343,427392895,421822335,422608767,421101439,
        420642687,419594111,427327359,419463039,420511615,306216831,309952383,304381823,
        305168255,303660927,303202175,302153599,309886847,302022527,303071103,75530111}; //seconds 00-60
int a = 0;
int b = 0;

// Shut off 7 segment displays to start //
IOWR_ALTERA_AVALON_PIO_DATA(HEX1_BASE, 2139062143);
IOWR_ALTERA_AVALON_PIO_DATA(HEX2_BASE, 2139062143);
```

Counter code for HEX:

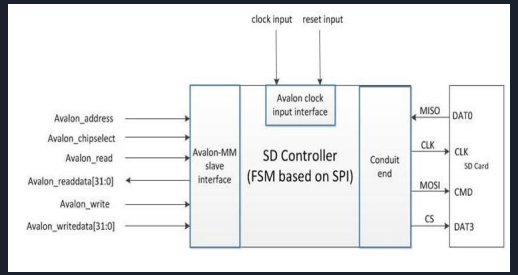
```
// Song runtime code //
if (sec[a] == 75530111) // If sec=60 increase minutes and reset seconds
{
            a=0; //reset second mask counter
            b++; //increase minute mask counter
            timertrack--;
            IOWR_ALTERA_AVALON_PIO_DATA(HEX1_BASE, sec[a]);
            IOWR_ALTERA_AVALON_PIO_DATA(HEX2_BASE, min[b]);
}
else if(timertrack > totalSecs)
{
IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_BASE, 0x8);
}
else //seconds is not yet equal to 60, continue counting
{
            IOWR_ALTERA_AVALON_PIO_DATA(HEX1_BASE, sec[a]);
            IOWR_ALTERA_AVALON_PIO_DATA(HEX2_BASE, min[b]);
            a++; //increase second mask counter
}
// end of Song runtime code //


IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_BASE, 0b00); // Resets TO bit to continue operations; IMPORTANT
timertrack++;
```

SD Card implementation

Important things to keep in mind:

1. SD card must be kept in FAT16 format to function properly.
2. SD card must be less than 4GB storage.
3. Read .wav file data from SD card which will be used to import music files.
4. Store the file in SDRAM via FIFO buffer.




```
void NewSong(void)
{
    alt_up_sd_card_dev *device = NULL;
    int connected = 0;

    device = alt_up_sd_card_open_dev(SDCARD_NAME);
    if (device != NULL)
    {
    printf("Initialized. Waiting for SD card...\n");
    while(1)
    {
     if ((connected == 0) && (alt_up_sd_card_is_Present()))
     {
         printf("Card connected.\n");
         if (alt_up_sd_card_is_FAT16())
         {
         printf("FAT16 file system detected.\n");

         printf("Looking for first file.\n");
         char * firstFile = "filenameunchanged";
         alt_up_sd_card_find_first(".", firstFile);
         printf("Volume Name: '%s'\n\n", firstFile);

         short file;
```

```
    while((file = alt_up_sd_card_find_next(firstFile)) != -1)
    {
        int contentCount = 0;
        printf("==========================\n");
        printf("Found file: '%s'\n", firstFile);

        short fileHandle = alt_up_sd_card_fopen(firstFile,false);
        printf("File handle: %i\n", fileHandle);

        printf("Contents:\n");
        short int readCharacter;

        while ((readCharacter = alt_up_sd_card_read(fileHandle)) != -1)
        {
            printf("%c", readCharacter);
            ++contentCount;
        }

        printf("\nContent size: %i", contentCount);
        printf("\n==========================\n\n");
    }
```

Audio CODEC

In this section of our project, we have a lot of problem initializing our Audio CODEC software and running it. It is still debugging and trying to figure out what the problem is.

Current code we have to implement Audio CODEC:

```
#define BUF_SIZE 80000 // about 10 seconds of buffer (@ 8K samples/sec)
int main(void)
{
alt_up_audio_dev * audio_dev;
/* used for audio record/playback */
unsigned int l_buf;
unsigned int r_buf;
// open the Audio port
audio_dev = alt_up_audio_open_dev (AUDIO_NAME);
if ( audio_dev == NULL)
alt_printf ("Error: could not open audio device \n");
else
alt_printf ("Opened audio device \n");
/* read and echo audio data */
while(1)
{
int fifospace = alt_up_audio_read_fifo_avail (audio_dev, ALT_UP_AUDIO_RIGHT);
if ( fifospace > 0 ) // check if data is available
{
// read audio buffer
alt_up_audio_read_fifo (audio_dev, &(r_buf), 1, ALT_UP_AUDIO_RIGHT);
alt_up_audio_read_fifo (audio_dev, &(l_buf), 1, ALT_UP_AUDIO_LEFT);
// write audio buffer
alt_up_audio_write_fifo (audio_dev, &(r_buf), 1, ALT_UP_AUDIO_RIGHT);
alt_up_audio_write_fifo (audio_dev, &(l_buf), 1, ALT_UP_AUDIO_LEFT);
}
}
// What if len (the 1s in the HAL functions) gets changed to 10?
}
```

Challenges and Constraints

- Audio CODEC ended up being much too difficult for us to be able to read the .wav file from the SD card and output
  - Also very little information online about the DE2-115 Wolfson WM8731 CODEC

- Interrupts with push buttons gave a lot of error since it didn't turn back to main() and continue the rest of the program

- Reading a .wav file from an SD card proved challenging as the data was not easily read in (ie. lots of random symbols, letters, and numbers)

Final Deliverable Demo