Step Counter Music DJ

Troy Davis

Caitlin Smart

Summary: Audio Player that updates playing song by matching beats per minute with steps per minute measured from an accelerometer.

ABSTRACT

We are using the Altera DE2 development board, which includes a Cyclone II FPGA and various other hardware and I/O interfaces, to implement an audio player that reads WAV files from an SD card. An accelerometer is used to measure an individual's step rate. The audio player chooses a song that has a beats per minute that is similar to this step rate. Steps per minute (SPM) and the currently playing track title is displayed on the LCD.

TABLE OF CONTENTS

FUNCTIONAL REQUIREMENTS

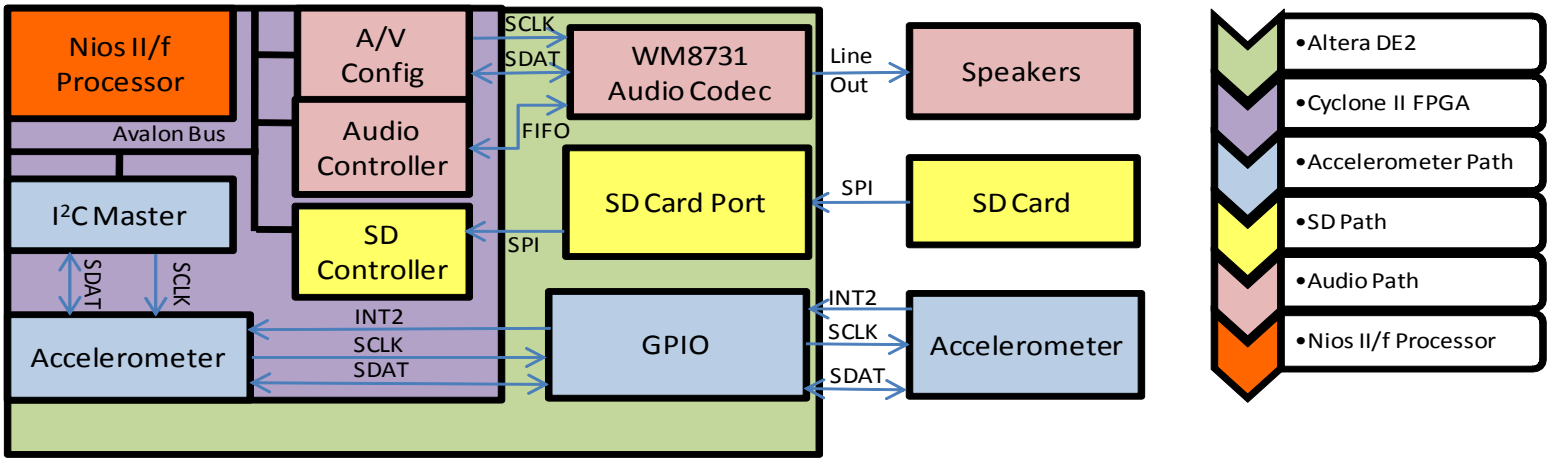
CORE REQUIREMENTS

1. Decode and play MP3 files to external speakers
2. Load MP3 files from SD card
3. Select a song based on the song's beats per minute
4. Select a song based on accelerometer input
5. Update song selection after each song ends
6. Smoothly fade into a new song
7. One momentary switch to force poll accelerometer
8. One dip switch to turn off accelerometer input
9. Standard MP3 controls (next song, previous song, etc.)
10. Display current track information on LCD
11. Volume control

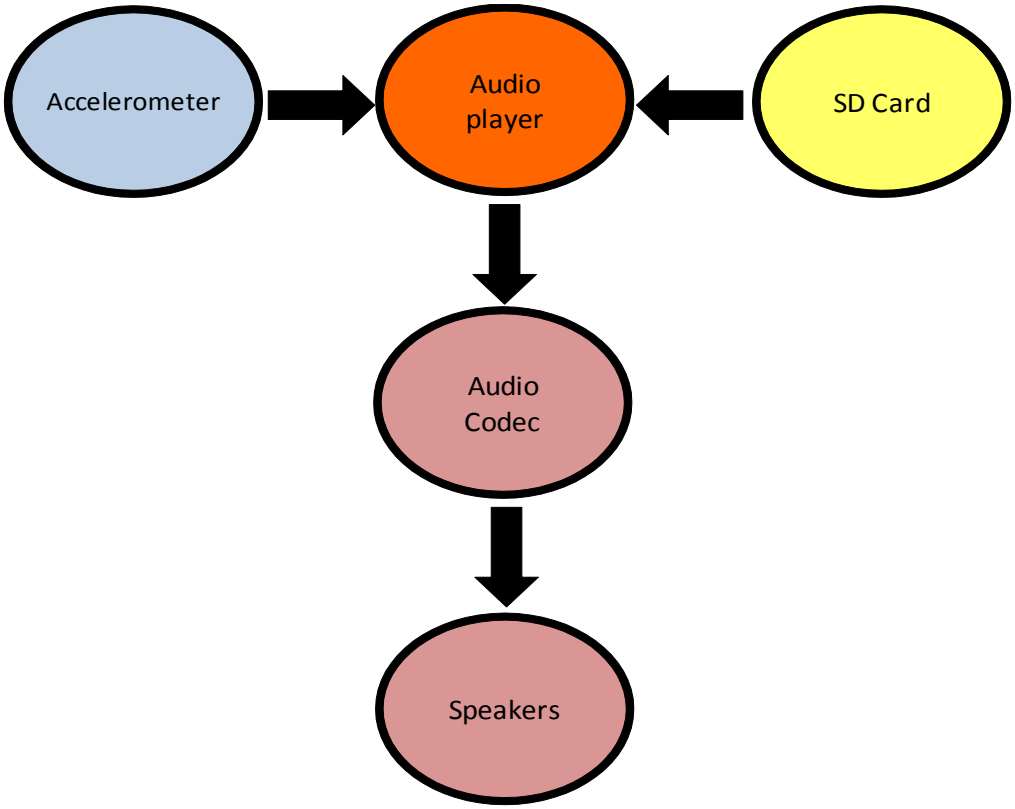
OBJECTIVES MET

1. Decoded and played WAV files to external speakers
2. Loaded WAV files from SD card
3. Selected a song based on the song's beats per minute
4. Selected a song based on accelerometer input
5. Updated song selection after each song ends
6. Did not implement fading to new song
7. Implemented momentary switch to update currently playing song based on current steps per minute
8. Did not implement dip switch to turn off accelerometer input
9. Did not implement standard MP3 controls
10. Displayed current track title on LCD
11. Did not implement volume control

DESIGN AND DESCRIPTION OF OPERATIONS



Hardware Block Diagram



Signal Flow Diagram

Our system is broken down into three main components controlled by the Nios II/f processor.

SD card: The SD card controller first checks to see if an SD card is present. The system will hold until an SD card is inserted. It then parses the SD card, looking for all WAV files stored on the card. As the files are found, the clusters associated with those songs are stored to be easily accessed later. Once the SD card is parsed, the obtained WAV files have their header information stripped. All this information is stored in the audio player structure.

Accelerometer: The accelerometer is configured to generate an interrupt on its external interrupt pin when a transient motion greater than 0.375g is observed. It should be noted that the accelerometer debounce is configured to disable interrupts for 150ms after an interrupt is detected. Our accelerometer vhdl component counts the amount of interrupts generated by the accelerometer device and makes them available as data out on the Avalon bus during a read operation. The read operation also reset the hardware counter. Using this interrupt counter, our software is able to determine the amount of events occurring per unit time. Measuring the hardware counter every half second and extrapolating this value to steps per minute leads to a large inaccuracy in the result. In order to maintain a high refresh rate while improving accuracy, a history of interrupt counter values is generated over a 25 second period. Using this history, a more accurate representation of the steps per minute can be obtained while maintaining a refresh rate of 2 Hz. In order to decrease lag time on start up, the history is initialized to 120 steps per minute. This value was recognized as being a good representation of the average steps per minute of an average person at an average pace. The average steps per minute calculated is passed to the audio player for song determination.

Audio Player: The audio player uses the average steps per minute to determine which song to play. It does this by mapping the average steps per minute to a range of acceptable beats per minute for the playback song. Once the acceptable range is determined, the struct initialized by the SD card is searched and a corresponding song is selected. This song is then played to completion. A

momentary switch was incorporated to allow the exiting of a song mid-playback. Once a song has finished playback, the process of song selection begins again. In this fashion, audio playback will be updated at the end of playback or when the button is pressed.

BILL OF MATERIALS

- Altera Terasic DE2 development board - $269.00USD
- 2 - Freescale Semiconductor triple axis accelerometers on a breakout board from Sparkfun (Part # SEN-10955). - 2 X $9.95 = $19.90
    - Uses I2C interface, user selectable full scales of ±2g/±4g/±8g, has current consumption of 6µA – 165µA, supply voltage of 1.95V – 3.6V, and interface voltage of 1.6V – 3.6V.
    - Supplier site: https://www.sparkfun.com/products/10955
    - Datasheet: http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Accelerometers/MMA8452Q.pdf
- Punch board to mount accelerometers - $2.00
- FAT 16 SD card - $10.00
- 40 pin ribbon cable - $15.00
- 9V DC 1.3A Power Brick - $10.00
- Wire wrap - $0.50
- Headphones - $10.00
- 40 pin header - $0.20
- 2 – 6 pin headers – 2 X $0.05 = $0.10

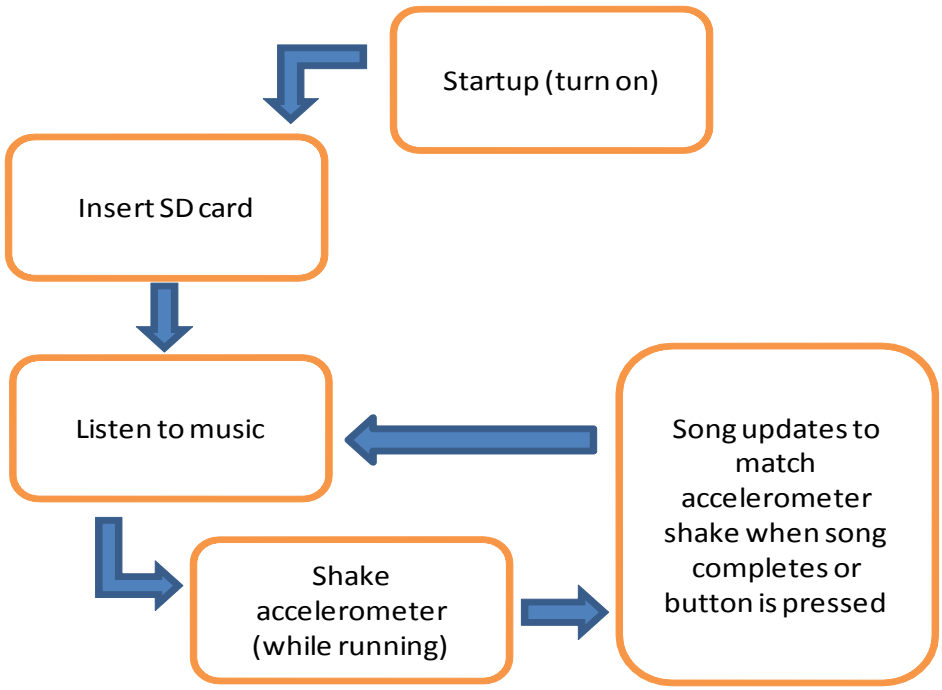
Total cost: $336.70

SOURCES OF REUSABLE DESIGN UNITS

The reusable design components that we utilized in our design were:

- I^2C master from opencores, which we modified slightly to facilitate the design of our accelerometer controller core. This core proved to be very useful in the control of the accelerometer device, but failed to provide a NAK in the event of I^2C reads, and also violated the timing requirements of the audio codec.
- Audio AV Config from Altera, through their University core program, proved to be ideal in the control of the audio codec. Register values are set before the core is instantiated on the FPGA. Therefore, volume control was not possible in our design.
- Audio DAC FIFO from Terasic provided a DAC FIFO for the audio codec. Our particular design ran the FIFO at 18.432 MHz at a sample rate of 48 kHz and a 16 bit data width. The audio DAC FIFO also made use of several peripheral cores for timing and reset control.
- A software implementation of SPI control of the SD card was found from Cornell University. Tying the SD card pins to PIO Avalon slaves allowed for software bit banging in this fashion.

DATASHEET



User Perspective Block Diagram

Measured Power:

7.79V DC

644mA DC

FPGA to board

Inputs

- SD to FPGA
  - Through an SD Card Controller (SPI)
- Accelerometer to FPGA
  - Through I^2C bus
- Dip switches/buttons/LEDs to FPGA
  - Through PIO registers

Outputs

- FPGA to speaker
  - Uses audio codec to transmit out
- FPGA to LCD

- LCD module takes in data (LCD_DATA) to display

Off-board

Inputs

- Accelerometer (power from DE2 board)
  - Through GPIO pins

Outputs

- Speaker
  - Through Line Out port

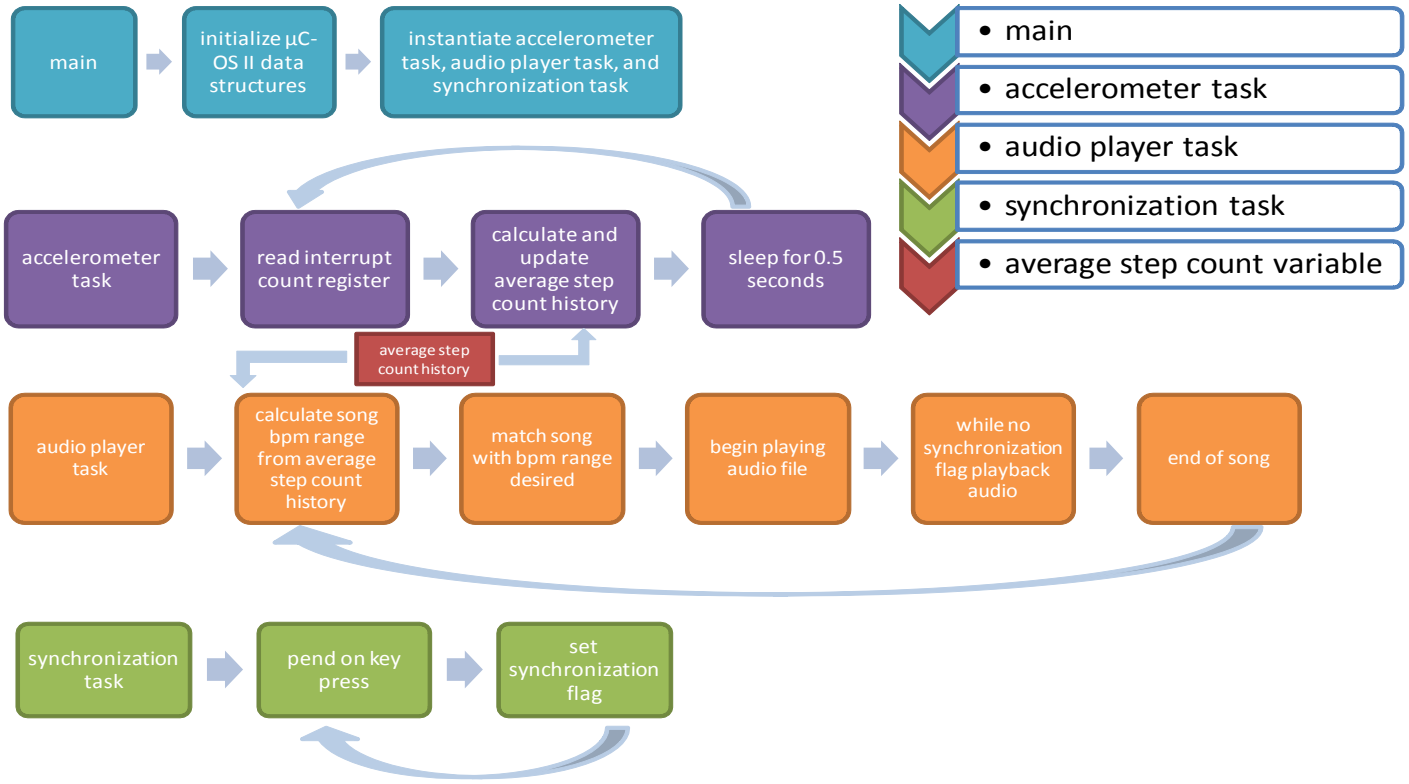
BACKGROUND READING

An article on IEEE Xplore, “Implementation of SD Card Music Player Using Altera DE2-70,” by Liang Hong-wei, Li Jian-ai, and Kan Ling-ling, discusses using our board as a music player using WAV files stored on an SD card. Even though we are using MP3s in our music player, this article has useful information about playing audio files from a FAT16 formatted SD card. They have a block diagram of their SD music player as well as their implementation in SOPC builder, which was very helpful to compare to our implementation. The link to this paper is:

Another article, “MP3 decoding on FPGA: a case study for floating point acceleration,” by Alexandros Papakonstantinou et al, discusses the acceleration of Floating Point applications, using an MP3 decoder implementation that relies on floating point math. They show that real-time processing can be done through floating point acceleration. This will likely be useful to us if we need to speed up the processing of our MP3 files for a real-time application. The link is:

Links to these articles can be found in the references section as references [6] and [7].

SOFTWARE DESIGN



Software Block DIagram

We use the MicroC/OS-II RTOS to implement four tasks: main, accelerometer, audio player, and synchronization. The main task initializes our µC-OS II data structures and instantiates the other three tasks. The accelerometer task calculates the average steps per minute (SPM) based on an average of the last 50 measurements to more accurately model SPM and mask against variances in step rate. The audio player task selects a song from the SD card within a beats per minute (BPM) range that matches the average SPM. The song is played to completion, at which point a new song is selected. The synchronization task pends on a button press. When pressed, the current song ends, forcing audio player to select a new song.

TEST PLAN

Hardware: Isolated component design allowed the components to be tested individually. Routing I^2C signals through GPIO pins allowed the observation of I^2C data and I^2C clock to be captured on an oscilloscope to verify signal integrity and correctness. In this fashion, both the audio codec and the accelerometer I^2C implementations were tested. The implementation for the SD card file structure used was a robust design and required little testing. However, before integration of the audio codec and audio player, we were able to decode SD card contents, building a database of audio files.

Software: Here again, our isolated component design allowed testing of the components individually. SD card testing followed a similar format as SD hardware testing. Calculating average steps per minute from the accelerometer separately from the rest of the system allowed verification of the correctness of the calculation without interference from other system tasks. Calibration of accelerometer register values was determined empirically through physical experimentation. Lastly, before system integration testing, the audio player was run without accelerometer input to ensure a high degree of audio quality was maintained and that real time playback was possible.

RESULTS OF EXPERIMENTS AND CHARACTERIZATION

The Nios II/e was incapable of providing WAV data at a rate capable of maintaining full audio FIFO buffers. This resulted in choppy, unintelligible audio playback. Upgrading to the Nios II/f processor provided the necessary increase in performance to allow sustained audio playback. At 50 MHz, audio quality was severely diminished. Several groups were able to increase audio quality by increasing system speed. 100 MHz was chosen for system clock frequency as it balanced performance and ease of implementation.

REFERENCES

[1] *Nios II Software Developer's Handbook, ver. 11.0,* Altera, San Jose, CA, 2011.

[2] *Embedded Peripherals IP User Guide, ver. 11.0,* Altera, San Jose, CA, 2011.

[3] MP3 Player. [Online]. Available: http://www.opencores.org/projects/i2c/ via www.alterawiki.com/wiki/MP3_Player

[4] Music Player. [Online]. Available: http://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2010/vs327_rw363/WAV_player/ECE%205760.htm

[5] Terasic SD Card Music Player. [Online]. Available: http://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2010/vs327_rw363/WAV_player/ECE%205760.htm

[6] L. Hong-wei, L. Jian-ai, and K. Ling-ling, "Implementation of SD Card Music Player Using Altera DE2-70," in *Multimedia and Signal Processing (CMSP), 2011 International Conference on*, vol.2, Guilin, China, 2011, pp.150-153. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5957486&contentType=Conference+Publications&searchField%3DSearch_All%26queryText%3Dsd+music+player+DE2.

[7] A. Papakonstantinou et al. "MP3 decoding on FPGA: a case study for floating point acceleration." [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.208.3461.

[8] *Freescale Semiconductor Datasheet Document Number: MMA8452Q, ver. 4.1,* Freescale Semiconductor, Tempe, AZ, 2011.

[9] *Wolfson Microelectronics Datasheet WM8731/WM8731L, ver. 3.4,* Wolfson Microelectronics , Edinburgh, United Kingdom, 2004.

[10] *Freescale Semiconductor Application Note Document Number: AN4071, ver. 1.0,* Freescale Semiconductor, Tempe, AZ, 2011.

APPENDICES:

QUICK START MANUAL

To assemble this project, you will need the DE2 board, speakers, a FAT 16 formatted 2 GB SD card, and the MMA8452Q accelerometer breakout board.

First, the accelerometer needs to be connected to the GPIO pins as follows:

ACCELEROMETER	GPIO
3.3V	VCC33 (pin 29) on GPIO 0 which corresponds to 3.3V
SDA	GPIO_0(3)
SCL	GPIO_0(5)
I2	GPIO_0(7)
GND	GND (pin 12 or 30) on GPIO 0 which corresponds to ground

Note the I1 pin and the SA0 pin on the bottom of the breakout board are not connected to anything.

The SD card needs to be formatted to FAT 16. The songs must be WAV files that are 16 bit, 48 kHz, and stereo. We used the website http://audio.online-convert.com/convert-to-wav to convert MP3 files to this format. The name of the WAV files must start with an integer which corresponds to the BPM of the song, followed by a space, and then the song title without spaces.

To run the volatile project, program the FPGA with Step_Music_DJ.sof which is in Step_Music_DJ.zip. In Nios II IDE, import Step_Counter_Music_DJ project located in Step_Music_DJ/software once the zip file is extracted. It is then possible to build and run on the DE2 board via JTAG connection.

To flash the non-volatile project, program the FPGA with Step_Music_DJ.pof which is in Step_Music_DJ.zip. In Nios II IDE, import Step_Counter_Music_DJ project located in Step_Music_DJ/software once the zip file is extracted. The Flash Programmer tool can then be used to flash the software to the board.

The SD card will need to be inserted and the speakers plugged in and turned on to begin the music playback. Once the project is instantiated on the board and running, either by running in volatile using the IDE or by turning on the board in flash, the music should begin to play automatically. The accelerometer needs to be shaken in the x-axis to register interrupts. Song selection will occur at end of playback or when Key 3 is pressed.

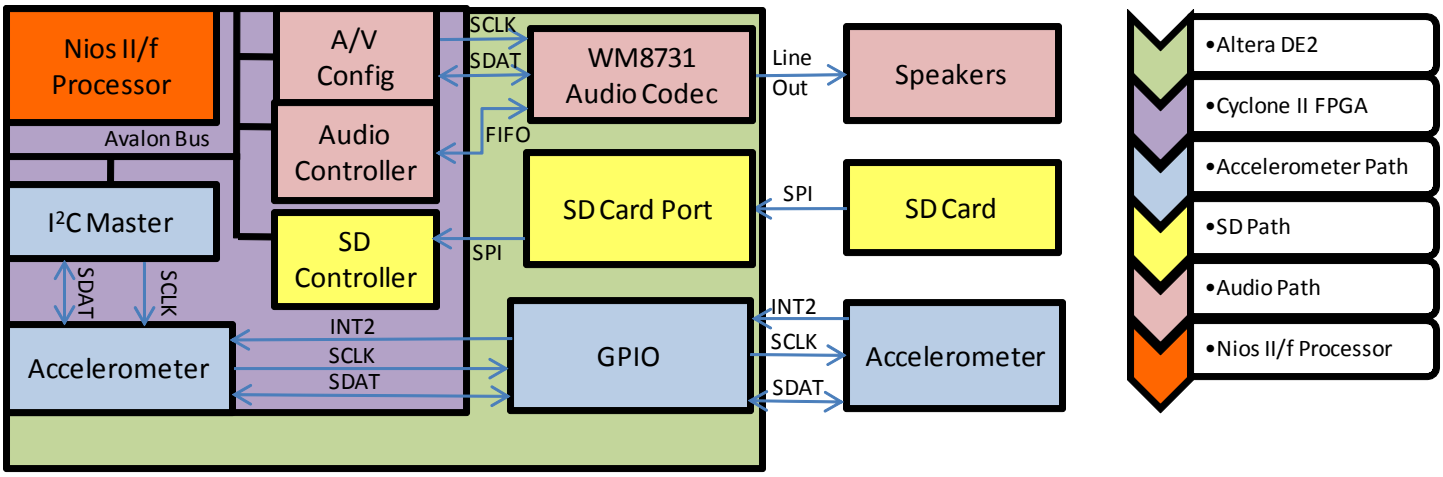
FUTURE WORK

Future work for this project could include completing the core requirements in functional requirements that we were not able to finish, such as volume control and standard MP3 player controls. Implementing MP3s would be a great addition to the project, as we were only able to implement WAV files. We were planning on using the MAD lib decoder to decode MP3s. Some extras that we thought could be added to this project were to flash LEDs to the music and implement Bluetooth speaker capabilities.

A major addition to the project would be to transfer to a small portable board, such as the Altera DE0. This would allow for a better experience as you could actually jog with the board. The DE0 also has an accelerometer built into it, so the external accelerometer would not be needed. However, this board does not have an on-board audio codec, so an external audio codec would need to be added.

Another addition could be to implement signal processing into this project. Currently, BPM is found by estimating the BPM of a song, or pulling it from the MP3 song information. Signal processing could be used to find the BPM of a song. Also, signal processing could be used to change the speed of the song to match the desired BPM. Auto tune could be included so the pitch of the song doesn't change. A possible extension of this would be to sync the music to the accelerometer, so the beat perfectly matches your step.

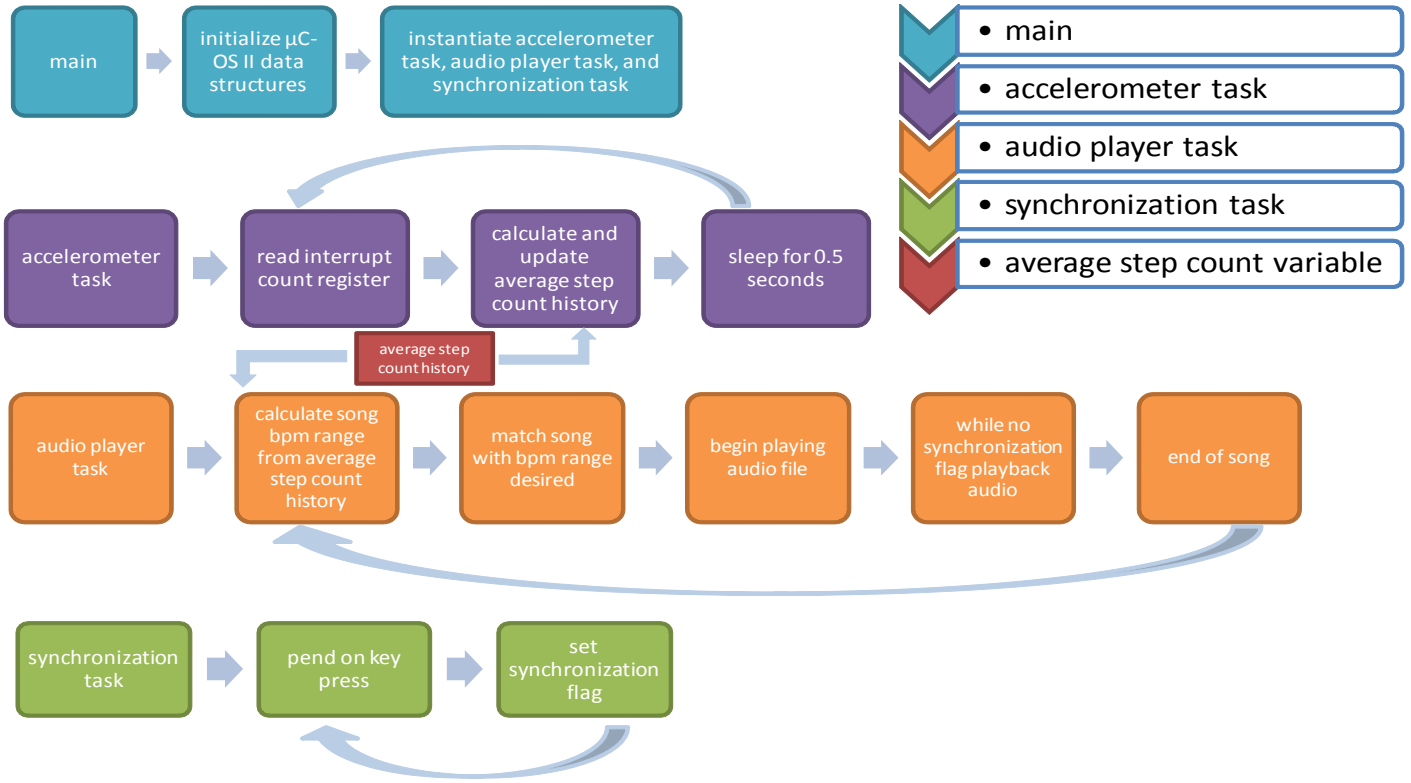
HARDWARE DOCUMENTATION



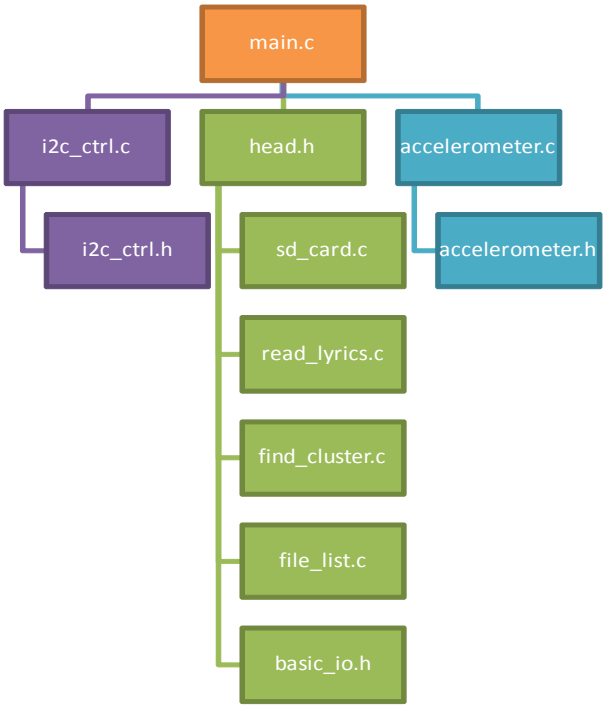
Hardware Block Diagram

SOURCE CODE

Refer to Step_Music_DJ.zip for source code.



Software Block Diagram



Software File Hierarchy

Index of Source Code Files:

- main.c (T): Main code file for project. Instantiates and runs project files and tasks.
- head.h (T): Header file for project. Contains all function declarations, variables, and defines.
- sd_card.c (T): Part of SD card library. Controls SD card functionality.
- find_cluster.c (T): Part of SD card library. Provides sd_card.c with functionality.
- file_list.c (T): Part of SD card library. Provides sd_card.c with functionality.
- read_lyrics.c (T): Part of SD card library. Provides sd_card.c with functionality.
- basic_io.h (T): Header file providing basic IO functionality.
- accelerometer.h (T): The header file for the accelerometer driver. Contains register defines and function definitions.
- accelerometer.c (T): Software driver for accelerometer. Performs register writes and reads via I^2C.
- i2c_ctrl.h (T): The header file for the I^2C master driver. Contains register defines and function definitions.
- i2c_ctrl.c (T): Software driver for I^2C master. Provides I^2C read/write functionality as well as initialization of I^2C controller.