Digital Circuit Lab Final Project Report

B09507018 WenChieh Lo

1. Abstract

The topic of our Final Project is "Beat the Drum!", a rhythm game inspired by the music recognition version of *Taiko no Tatsujin*. At the start of the game, players can record any piece of music. The game will generate a drum score based on the rhythm of the recorded music, allowing players to play along using drums and drumsticks.

2. Onset Detection

We use **Onset Detection** to generate drum scores. In music, we can observe that when there is a sudden increase in the sound signal, it can be identified as a potential rhythm point.

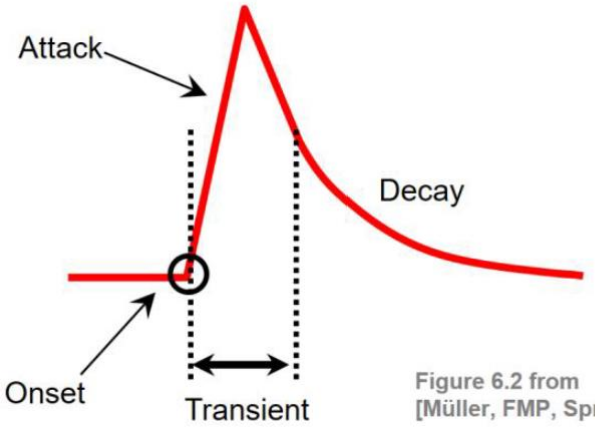
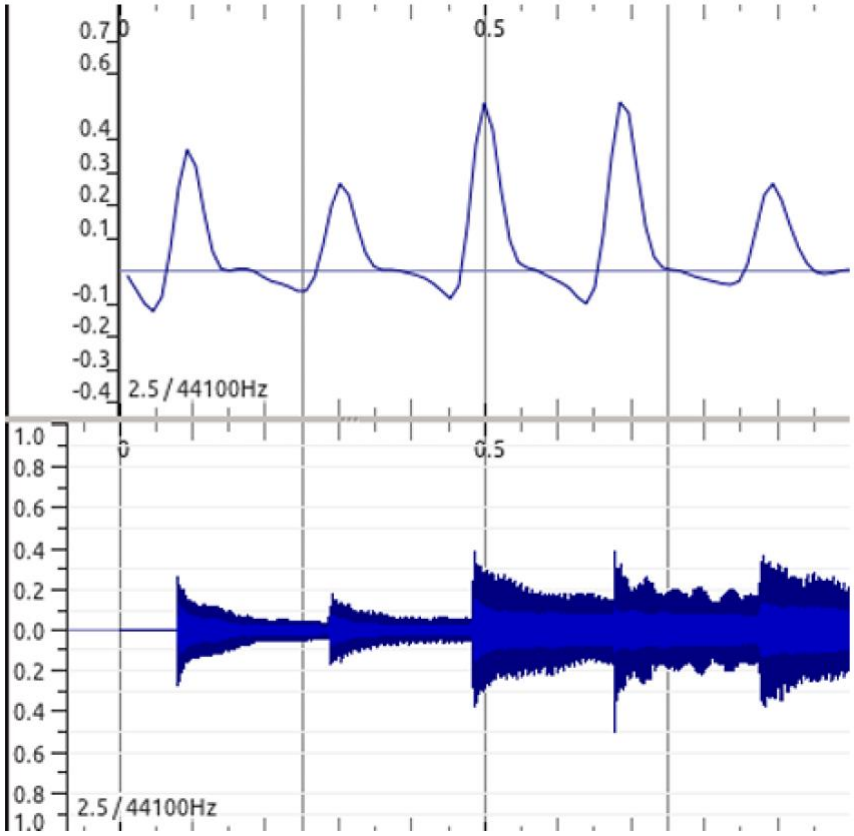
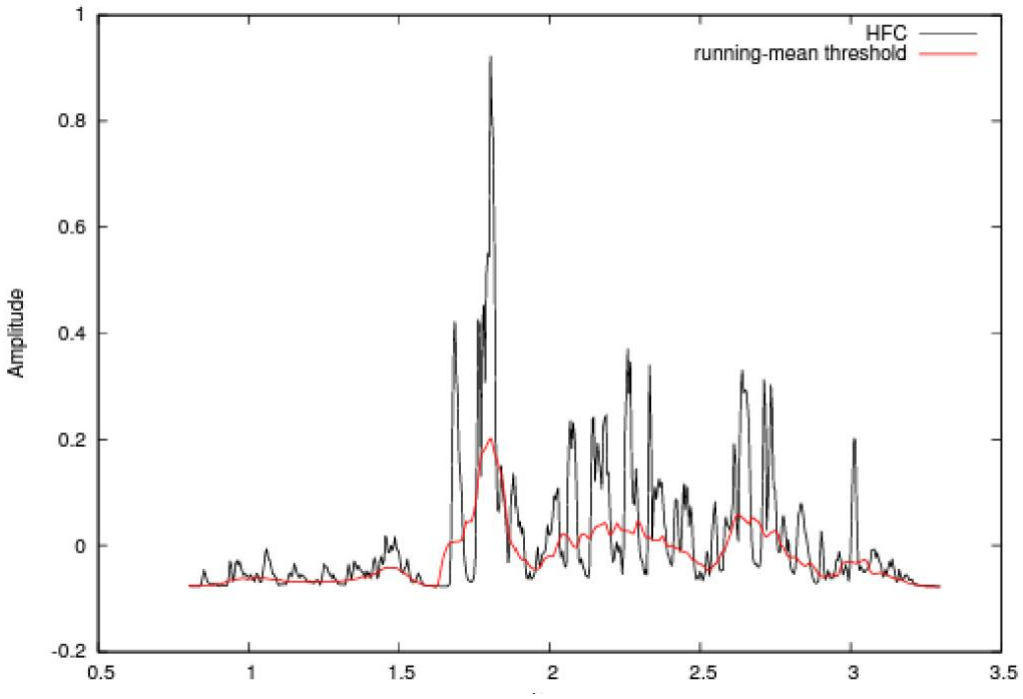


Figure 6.2 from
[Müller, FMP, Springer 2015]

To achieve this, we first divide the audio into small segments and perform a Fourier Transform on each segment, mapping the signal into the frequency domain. Then, we compare the energy difference of signals at the same frequency between the current and previous intervals. If the difference is positive, we identify it as an **attack** and store it; if it is negative, we consider it as **decay** and ignore it. Ultimately, we can obtain a signal, as shown in the diagram, that represents the calculated differences.



Next, we compute the average value of the preceding and following 8 windows and determine if the current window exceeds the average by several times. If it does, it indicates a sharp increase, and the window is selected as a rhythm point.



3. Fast Fourier Transform

When processing audio, the Fourier Transform is commonly used. Below is an introduction to the **Cooley-Tukey Algorithm**:

$$X[k] = \sum_{r=0}^{\frac{N}{2}-1} X[2r]W_N^{2rk} + \sum_{r=0}^{\frac{N}{2}-1} X[2r+1]W_N^{(2r+1)k}$$

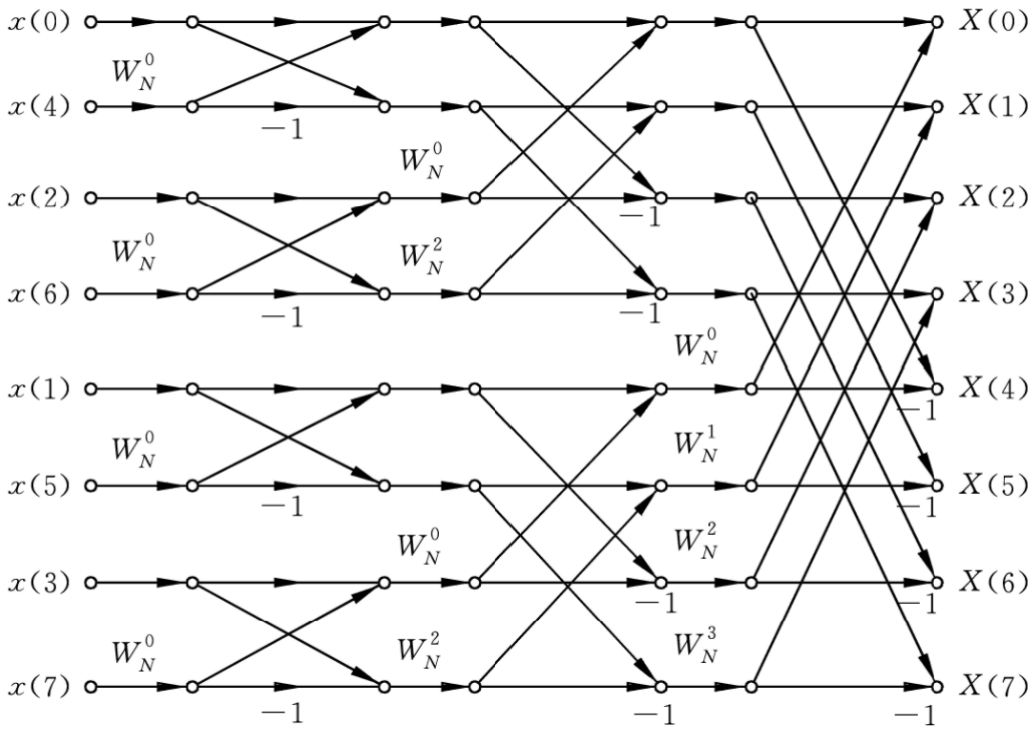
$$= \sum_{r=0}^{\frac{N}{2}-1} X[2r]W_{\frac{N}{2}}^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} X[2r+1]W_{\frac{N}{2}}^{rk}$$

$$= A[k] + W_N^k B[k] \quad , k = 0,1, \dots, \frac{N}{2} - 1$$

$$X[k + N/2] = A[k] - W_N^k B[k] \quad , k = 0,1, \dots, \frac{N}{2} - 1$$

The discrete Fourier Transform (DFT) is recursively divided into computations of size N/2, which can eventually be broken down into pairs for computation.

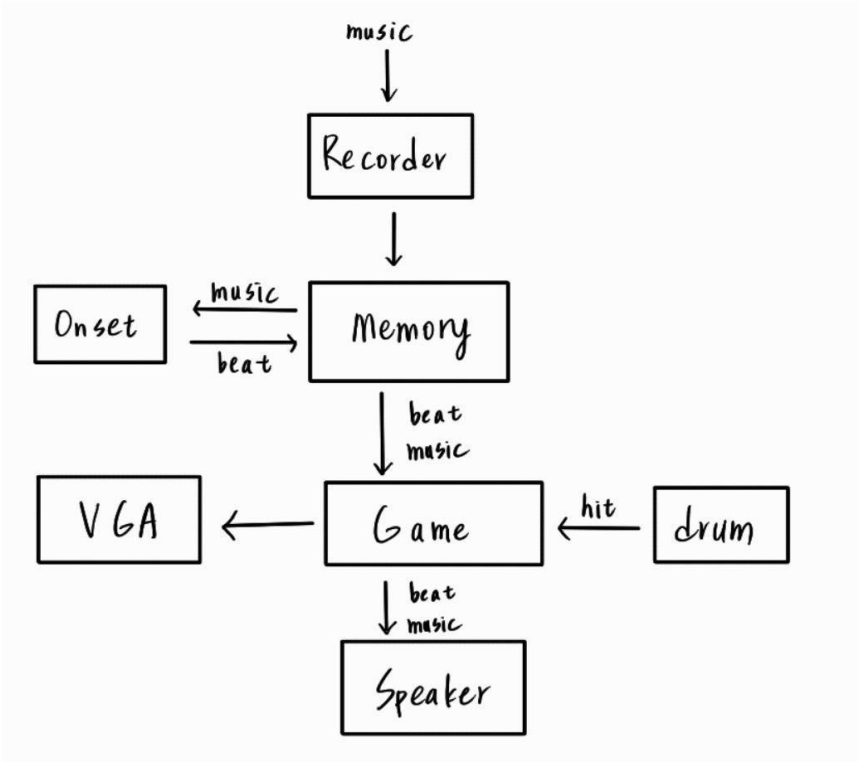
Once divided into pairs, the computation can be performed using a butterfly-like calculation method. Each stage involves N/2N/2N/2 groups of computations, with a total of $log_2 N$ stages. This process ultimately produces the correct FFT (Fast Fourier Transform) result.



4. System Architecture

(a) Top

At the start of the game, the system first records audio using a **Recorder**. The recorded music data is stored in **Memory (SRAM)** and then sent to the **Onset Detection** module for processing to identify the rhythm drum score. The drum score is then stored back in Memory.



Next, both the music and drum score are sent to the **Game Controller**, which detects the drumstick's strikes and controls the corresponding VGA game visuals and speaker output to play the music.

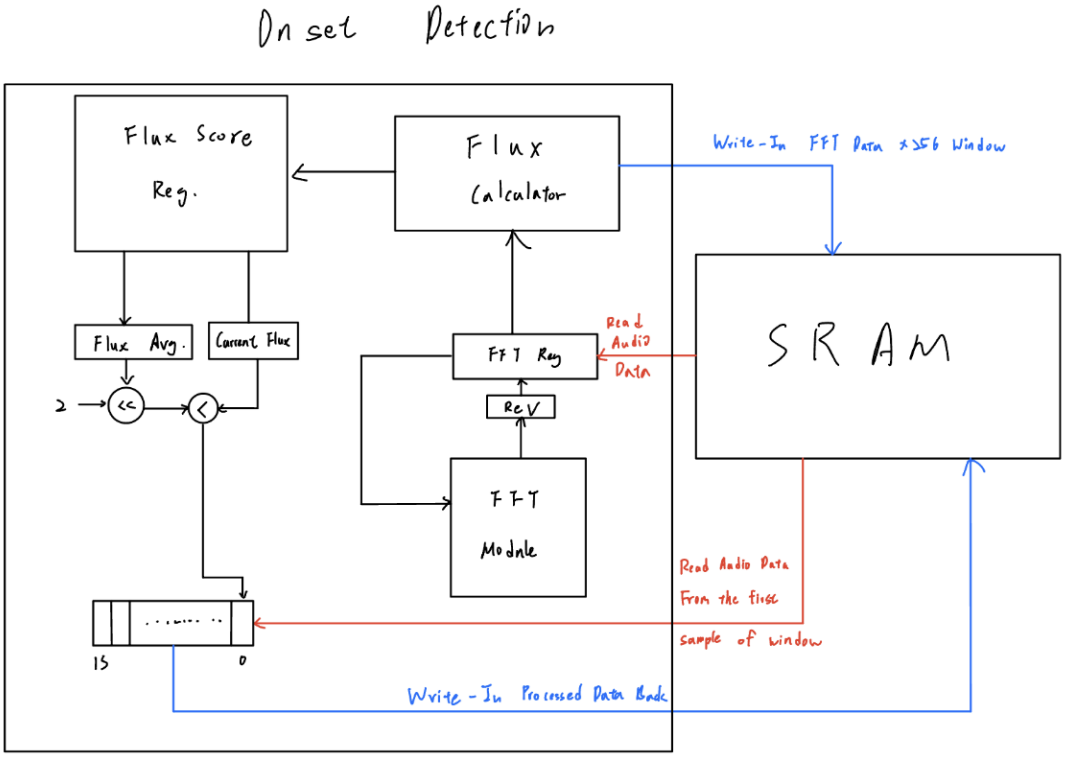
(b) Recorder、Speaker

We use a speaker and a recorder connected to the FPGA board to record and play our audio. I2C protocol were used to transmit the audio data.

(c) Onset Detection（Red: input、Blue: output）

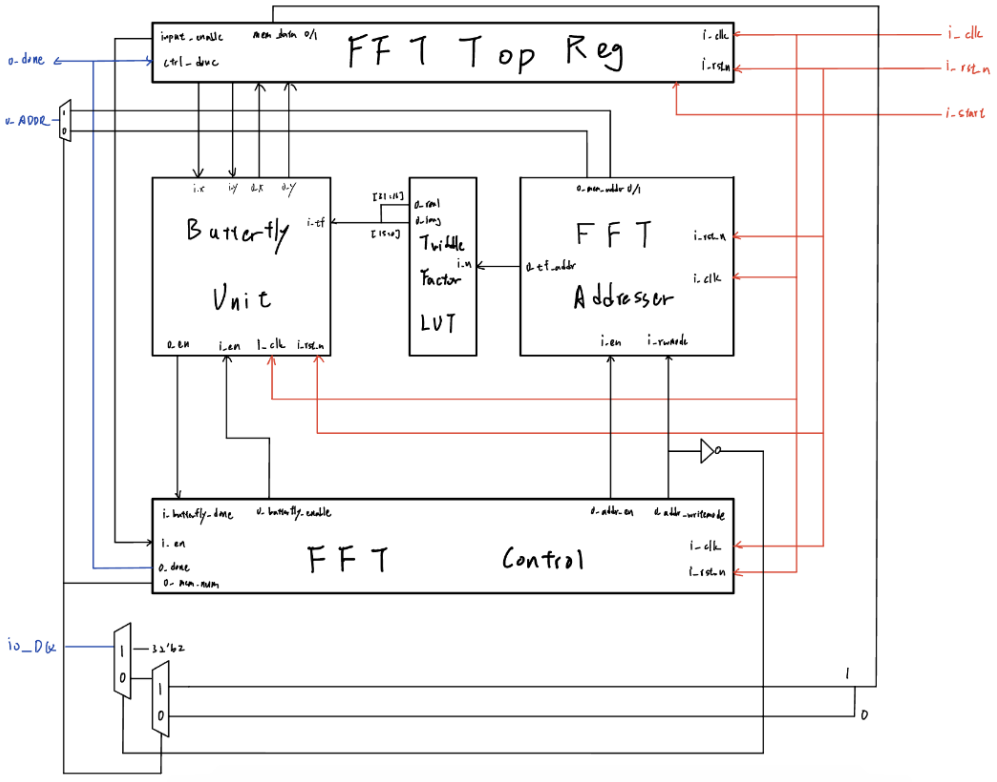
- **SRAM**: Stores music data and the FFT results of the first 256 windows.
- **FFT Module**: Performs the Fourier Transform and uses Rev to organize the computed FFT values.
- **FFT Reg**: Temporarily stores the FFT computation results.
- **Flux Calculator**: Temporarily stores the sum of FFT results across different windows.
- **Flux Score Reg**: Temporarily stores the sum of the flux values for different groups of 256 windows. It also provides the average value to compute the Moving Average. When the flux exceeds the Moving Average by several

times, it writes a beat to memory.

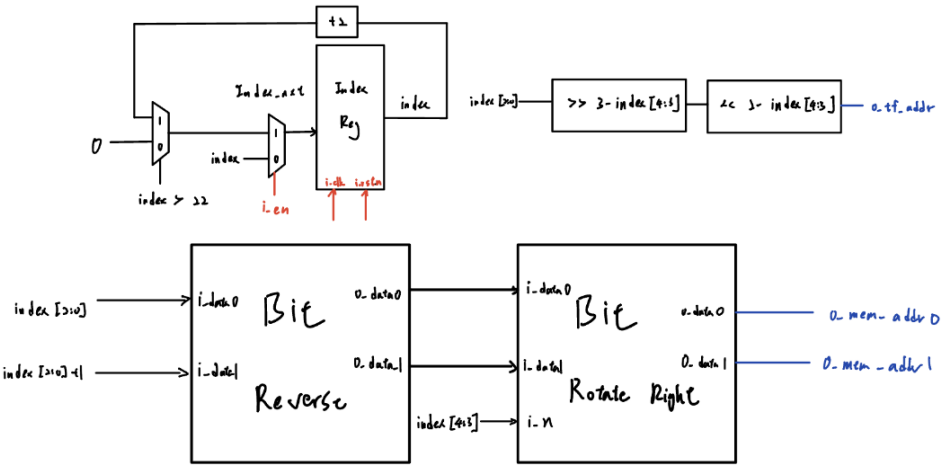


4. FFT

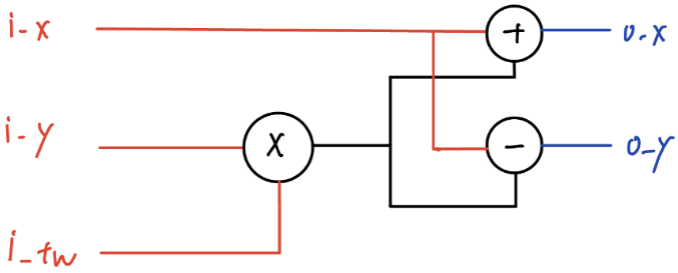
(a) Top



(b) Address

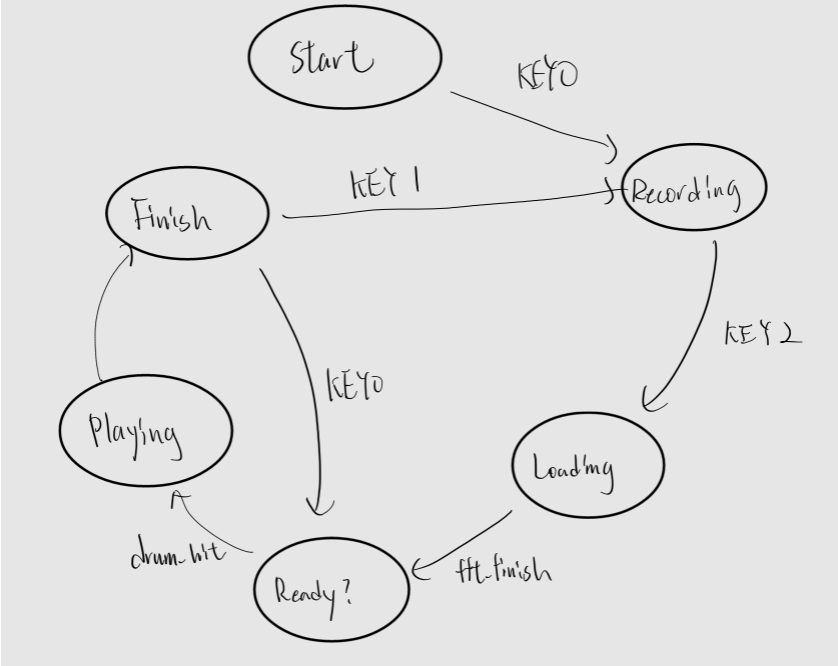


(c) Butterfly Unit



5. Hardware Scheduling

1. Top

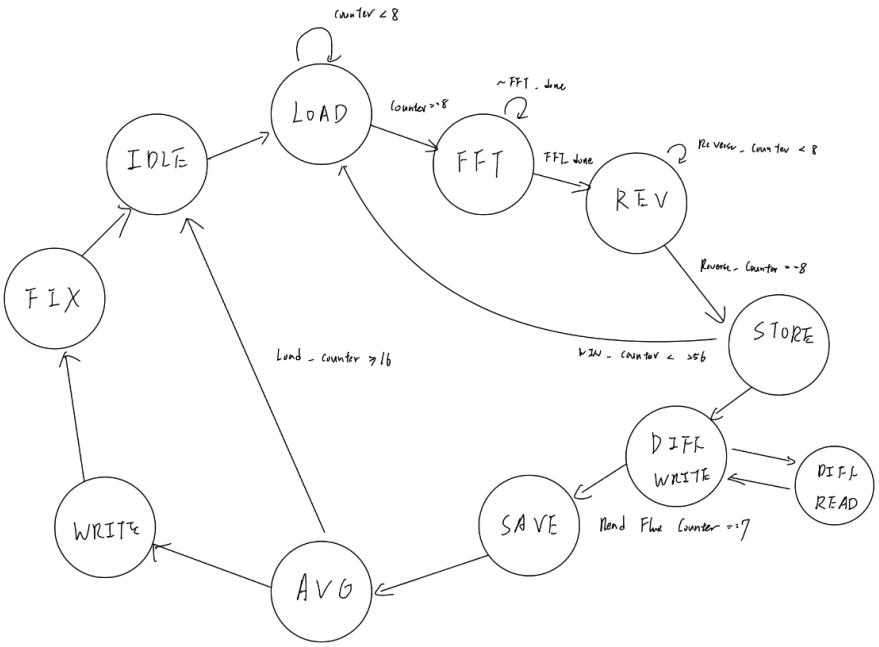


- **Start**: The initial IDLE state. Pressing reset will return the system to the Start state.
- **Recording**: After transitioning from the Start state by pressing key0, the

system enters the Recording state. Here, the Top Module controls the Audio Module to start recording, and the audio file is saved into SRAM.

- **Loading**: Once the user finishes recording, the system transitions to the Loading state. The Top Module controls the Onset Detection Module to process the audio signal and store the calculated beats into SRAM.
- **Ready**: After the Onset Detection Module completes processing, it sends a finish signal to the Top Module, transitioning the system to the Ready state. The game starts when the player hits the drum once.
- **Playing**: In the Playing state, the Top Module controls the Audio Module to play the game music. Simultaneously, it decodes the drum score stored in SRAM and controls the VGA Module to display the drum visuals.
- **Finish**: After the game ends, the player can choose to replay the same song or record a new one.

2. Onset Detection

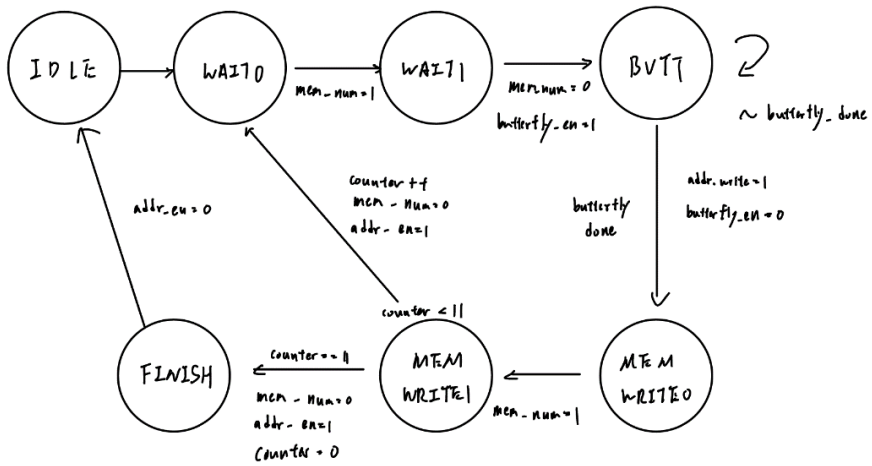


- **IDLE**: The initial state.
- **LOAD**: Reads audio data from **SRAM** into a register. Once 8 points are read, the system transitions to **FFT**.
- **FFT**: Performs the Fourier Transform on the audio data from **SRAM**. Upon receiving an **FFT_done** signal, the system moves to **REV**.
- **REV**: Processes the data in the register by performing a transformation, swapping the original data with data stored at bit-reversed addresses.
- **STORE**: Adds the data in the register to **Flux_tmp** for accumulation. If 256 windows of data have been accumulated, the system transitions to **DIFF**. If fewer than 256 windows are accumulated, the system returns to **LOAD** to

continue reading audio data from **SRAM** and calculating FFT.

- **DIFF**: Calculates the difference between **Flux_tmp** at different frequency points and the **Flux_tmp** stored in **SRAM** for the previous 256 windows. The positive differences are summed to calculate the **Flux**. The updated **Flux_tmp** is then written back to **SRAM**, replacing the previous values.
- **SAVE**: Stores the calculated **Flux** into **Flux_avg**.
- **AVG**: Checks if there are 16 sets of **Flux_score** available for calculation. If so, it evaluates whether the **Flux** from 8 windows prior exceeds the average of these 16 sets. If it does, the system identifies it as a beat (**beat = 1**) and transitions to **WRITE**. If not, or if fewer than 16 sets are available, the system returns to **IDLE** to continue processing new data.
- **WRITE**: Processes the audio data from 8×256×8 points prior and updates the **LSB** of the data to represent the beat. This updated data is then written back to the corresponding **SRAM** address.
- **FIX**: Checks whether the **WRITE** operation modified any **SRAM** addresses. If so, it adjusts the **SRAM** read address accordingly.

3. FFT



- **IDLE**: The initial state.
- **WAIT0/1**: Sequentially loads the data required for computation.
- **BUTT**: Performs butterfly unit calculations.
- **WRITE0/1**: Sequentially writes the computed data back to memory.
- **FINISH**: Signals the completion of the computation.

6. Fitter Summary

Fitter Summary	
Fitter Status	Successful - Fri Jun 16 11:30:29 2023
Quartus II 64-Bit Version	15.0.0 Build 145 04/22/2015 SJ Full Version
Revision Name	DE2_115
Top-level Entity Name	DE2_115
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	9,786 / 114,480 (9 %)
Total combinational functions	9,316 / 114,480 (8 %)
Dedicated logic registers	2,440 / 114,480 (2 %)
Total registers	2440
Total pins	480 / 529 (91 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	31 / 532 (6 %)
Total PLLs	1 / 4 (25 %)

7. Timing Analyzer

TimeQuest Timing Analyzer Summary	
Quartus II Version	Version 15.0.0 Build 145 04/22/2015 SJ Full Version
Revision Name	DE2_115
Device Family	Cyclone IV E
Device Name	EP4CE115F29C7
Timing Models	Final
Delay Model	Combined
Rise/Fall Delays	Enabled

8. Problems

(a) FFT Cannot Perform Large-Point Calculations

When using a large reg array for calculations, FPGA resources can quickly become insufficient, causing synthesis failure. To compensate for the loss in accuracy, we ultimately chose an 8-point FFT and combined it with 256 overlapping windows to simulate a sufficiently long signal for rhythm recognition.

(b) Parameters Cannot Be Fine-Tuned

Compared to the Librosa library, where the parameters have been fine-tuned to find the optimal range, when using FPGA, we must first synthesize the circuit onto the FPGA before testing the results. Therefore, many parameters such as the FFT sample points, threshold multipliers, etc., cannot be as easily fine-tuned as in software. Additionally, the digital precision in FPGA is not comparable to that in software. Thus, we can only select the best parameters available at the time for adjustment.

(c) How to Store Drum Sheet and Audio Together

We chose to modify the original 16-bit audio file to 15 bits. In other words, we discarded the least significant bit (LSB) to store the drum hits. This not only saves space but also synchronizes the drum hits with the audio timeline.

(d) Vibration Sensor is Too Sensitive

We used a digital vibration sensor, and when the drumstick hits the drumhead, it registers more than one vibration. To address this, we implemented a debouncer to limit the sensor to a maximum of three vibrations per second.

9. Reference

1. https://www.runoob.com/w3cnote/verilog-fft.html
2. https://blog.csdn.net/matrix_laboratory/article/details/53709638?spm =1001.2101.3001.6650.1&utm_medium=distribute.pc_relevant.none- task-blog-2~default~CTRLIST~Rate-1-53709638-blog-128320041.235%5Ev32%5Epc_relevant_default_base3&depth_1-utm_source=distribute.pc_relevant.none-task-blog- 2~default~CTRLIST~Rate-1-53709638-blog-128320041.235%5Ev32%5Epc_relevant_default_base3&utm_relevant_index=2&fbclid=IwAR0uEll73XXavJ7JHWMBfdN6-0faAW_DvBa5vJvyidfYi9pITjZXg9cbpcs
3. https://blog.csdn.net/chengfengwenalan/article/details/79854730