Hardware Design and Analysis

Final Project Report

Paper Tape Music Player

紙捲磁帶播放機

Team32

Contents

Abstract

The original concept comes from mimicking a *Gramophone (Fig 1.1)* using an Infrared detector, but later for the purpose of simplicity, we decide to design a new device which we named *paper tape music player*.

Figure 1.1 Gramophone

Figure 1.2 Cassette

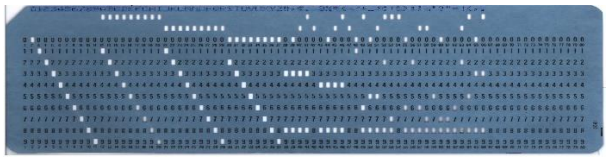


Figure 1.3 punched card

Paper tape music player (Fig 1.4) adopt the concept of a *Cassette (Fig 1.2)* and a *punched card (Fig 1.3)*, in which the paper tape will record the data of a coded music sheet, and the information on it will be scanned by an Infrared Detector while being spun by the power of a micro-motor, after the data being processed by the FPGA board, the music will be played by the audio module.

The data of a music sheet will be coded in **decimal** form and later interpreted by **binary** representation with python code on generated paper tape, we represent each object being coded a note symbol and a size of 1 byte, and each note symbol is encoded by a various number of equal black lines and white intervals respectively, in which the first 6 bits of the byte represents **pitch(音高)** and the last 2 bits represents **duration(音符持續時間)**.

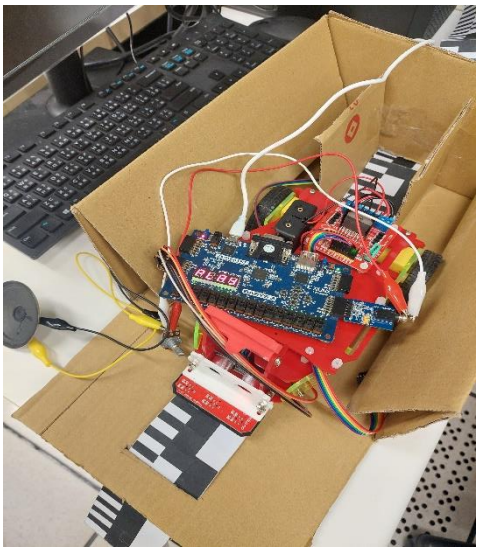


Figure 1.4 Actual implementation of our design

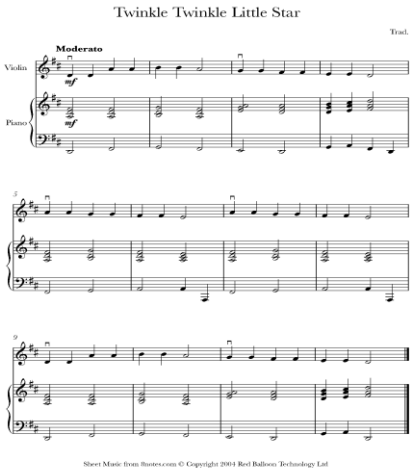


Figure 1.5 The Original Music sheet of Twinkle_Twinkle_Little_Star

We design a **table**(enclosed in the end of the file) for each note symbol and duration, and we represent each distinct pitch with **2-digit** decimal number and duration for **1-digit** decimal number, so in our coded music sheet *(Fig 1.6)* each note symbol is represented by exactly a **3-digit decimal number**, and the paper tape is generated by interpreting each 3-digits decimal number with a **1-byte** binary number by **concatenation**.

The paper tape is divided into 3 sections, in which the leftmost section represents **changes of bit** which the rightmost section represents **changes of bytes**, these two sections are used for error correcting so that the FPGA knows when it should read the new bits or write new bytes into memory. Lastly, the middle section is the **actual data** of the coded music sheet in binary representation.

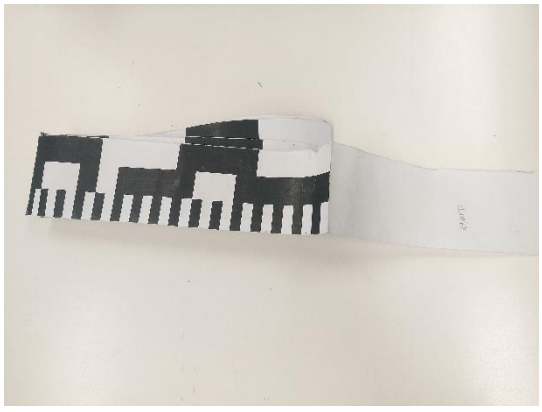


Figure 1.6 Encoded Paper Tape

```
000_150_003_150_
003_230_003_230_
003_250_003_250_
003_230_230_002_
210_003_210_003_
190_003_190_003_
170_003_170_003_
150_150_002_633_
```

Figure 1.7 coded music sheet

Because the nature of the IR detection module, the white block denotes high output which is 1'b1, and a black block denotes low output which is 1'b0. Additionally, the first byte of the paper tape is 8'b0000_0000 and the last byte is 8'b1111_1111, these two values are of **special purpose** that are only used for recognizing the **start** and **end** of the data-stream for IR-scanning and music playing. Also worth mentioning is that we only processed the IR signal for each 0.02 second to **avoid disturbances**.

After the paper tape being successfully read-in by the IR module, the user can direct the FPGA device to play, pause or restart the music. In addition, clear the data previously read-in and read-in another new instance of paper tape is also possible. We use FSM to achieve such requirements, and each state will be transitioned to the next after one second if any of the button is pressed.

The FPGA can store at most **5 tracks** at a time, and we allocation each track **200 bytes** of memory space for a total of **1024 bytes** by employing **contiguous memory allocation** scheme, and we also design **memory protection mechanisms** so that each track can only access its own memory space. The user can use the switches on the FPGA to decide which memory space to store, for example: 5'b00001 represents the first track, 5'b00010 represents the second track, and so on.

The state of the *Paper tape music player* will be displayed so that the user can know the current working status of the device for convenience and debugging purposes.

What we have learned from the lab:

We eventually delivered the successful implementation in time despite late submission.

At the beginning I once have the thought of designing a video game but soon later rejected it since it will be more appropriate to design a more hardware-related project in such a course from our perspective.

Before submitting the final project proposal, we had a thorough discussion online, and we quickly came out the initial idea of the whole design.

Through the implementation, we did follow the proposal and implemented the design despite having a slightly modification on the data representation, and this enables us to actually spread the workload in a more **parallel-like fashion** to complete the project in an efficient way.

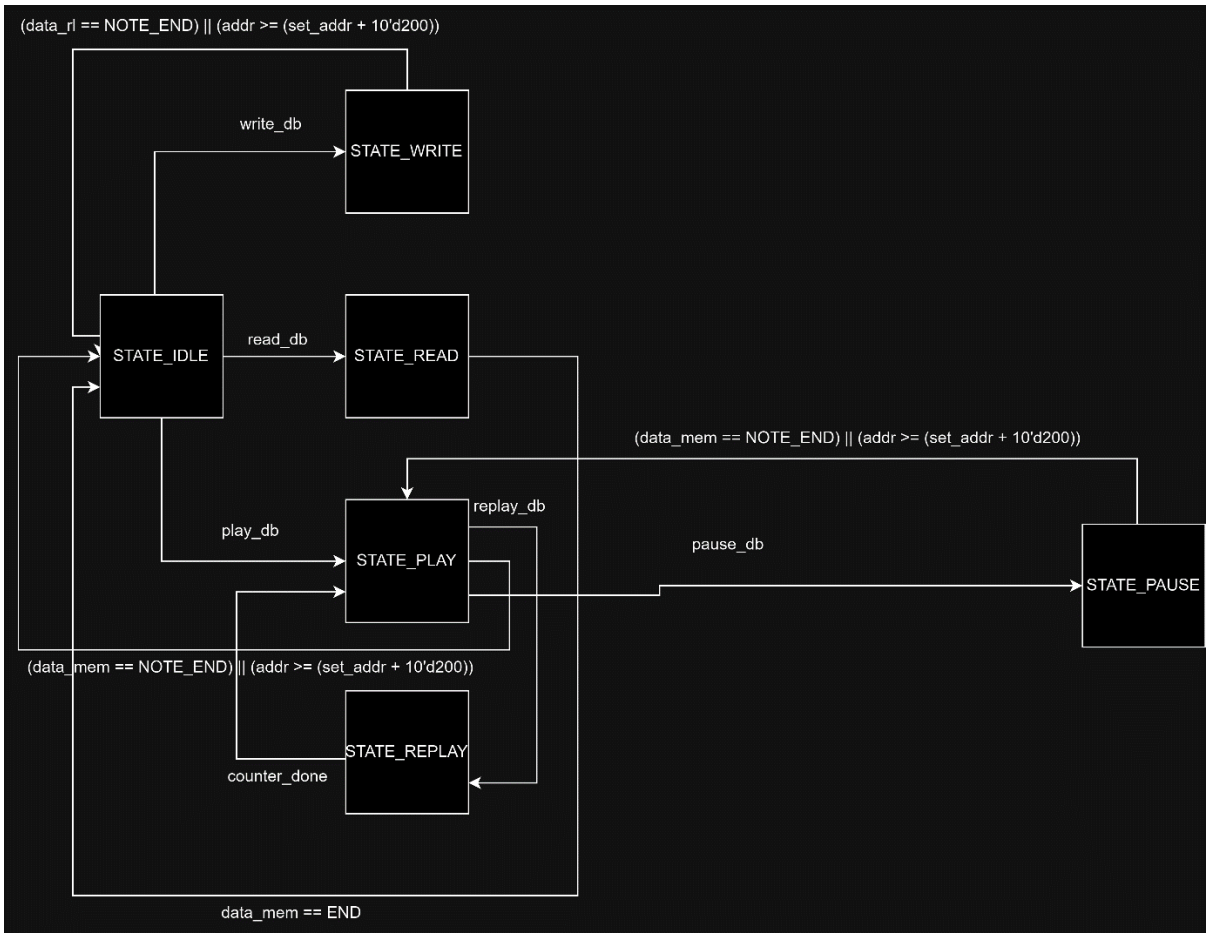
Through the implementation process we incorporated various technique such as python code for **image generation**, **data representation** and **error correcting** on the paper tape, **memory allocation** and **protection mechanisms** on the FPGA, detecting IR signal with delayed cycle etc., **workload distribution** and **project design**, and this is indeed a valuable experience since we developed the whole design concept from scratch and applying various concept onto a single device.

-林奕為

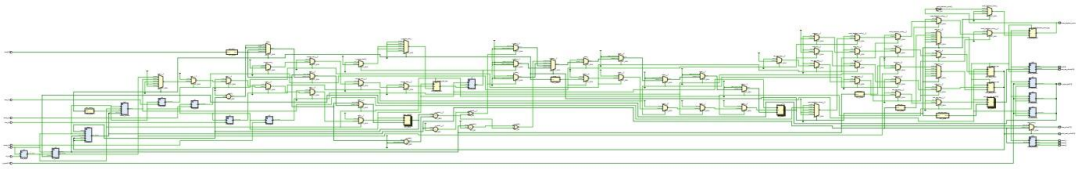
Objective:

- *Correctly plays the audio user desires.* (Y)
- *Correctly reads in the coded music sheet.* (Y)
- *Correctly generates coded paper tapes.* (Y)
- *Correctly display current state in 7-segment of FPGA.* (Y)
- *Allows resume, pause, play(restart) operations.* (Y)
- *Allows clear data, read data operations.* (Y)
- *Store multiple music tracks in FPGA memory* (Y)
- *Memory Protection Mechanism (Y)*

State Transition Diagram:



Block Diagram:



Note encoding table:

Note symbol: (first 6 bits)	
rest	6'd0
Do-L	6'd1
Do-L-up	6'd2
Re-L	6'd3
Re-L-up	6'd4
Mi-L	6'd5
Mi-L-up	6'd6
Fa-L	6'd7
Fa-L-up	6'd8
So-L	6'd9
So-L-up	6'd10
La-L	6'd11
La-L-up	6'd12
Si-L	6'd13
Si-L-up	6'd14
Do	6'd15
Do-up	6'd16
Re	6'd17
Re-up	6'd18
Mi	6'd19
Mi-up	6'd20

Fa	6'd21
Fa-up	6'd22
So	6'd23
So-up	6'd24
La	6'd25
La-up	6'd26
Si	6'd27
Si-up	6'd28
Do-H	6'd29
Do-H-up	6'd30
Re-H	6'd31
Re-H-up	6'd32
Mi-H	6'd33
Mi-H-up	6'd34
Fa-H	6'd35
Fa-H-up	6'd36
So-H	6'd37
So-H-up	6'd38
La-H	6'd39
La-H-up	6'd40
Si-H	6'd41
Si-H-up	6'd42
Do-HH	6'd43
Do-HH-up	6'd44
Re-HH	6'd45
Re-HH-up	6'd46
Mi-HH	6'd47
Mi-HH-up	6'd48
Fa-HH	6'd49
Fa-HH-up	6'd50
So-HH	6'd51
So-HH-up	6'd52
La-HH	6'd53
La-HH-up	6'd54
Si-HH	6'd55

Si-HH-up	6'd56
tempo	(last 2 bits)
1 second	2'd0
1/2 second	2'd1
1/4 second	2'd2
1/8 second	2’d3