Microprocessor Systems Final Report

Weiheng Zhuang Ziyu Zhu

Table of Contents

Introduction

The final project is designed to make the microprocessor to function like a piano. With a USB keyboard input and a speaker output, the user can play the piano. This project would use a terminal output as user interface, timer to control DMA for faster data access, USB HID input and DAC output. For the piano, there would be one octave for the user to play. The corresponding keys on the keyboard are from Q to U horizontally.

Piano was chosen as the instrument because the author was familiar with the piano. Apart from that, it is very easy and straightforward to play using a keyboard. Every key can be linked into a piano note. Also, the piano note frequency was designed in a good pattern.

$$f_n = f_0 \cdot (a)^n$$

f_0 is the frequency of one fixed note which must be defined. A common choice is setting the A above middle C (A_4) at f_0 = 440 Hz.

n is the number of half steps away from the fixed note that is chosen.

$a = 2^{\frac{1}{12}}$

f_n is the frequency of the note n half steps away.

Subsystem: Audio output

Introduction

Audio formats nowadays are very broad, each of them have lots of differences on sampling, encoding, decoding, etc. For this subsystem, wav format is used as the original piano note source format.

Waveform Audio File Format is an audio file format standard, developed by IBM and Microsoft, for storing an audio bitstream on PCs. It is the main format used on Microsoft Windows systems for uncompressed audio. The usual bitstream encoding is the pulse-code modulation (PCM) format. [1]

In a PCM stream, the amplitude of the analog signal is sampled regularly at uniform intervals, and each sample is quantized to the nearest value within a range of digital steps. A PCM stream has two basic properties that determine the stream's fidelity to the original analog signal: the sampling rate, which is the number of times per second that samples are taken; and the bit depth, which determines the number of possible digital values that can be used to represent each sample. The sine wave is sampled at regular intervals, shown as vertical lines. For each sample, one of the available values is chosen. The PCM process is commonly implemented on ADC. This generates a fully discrete input signal that can be easily encoded as digital data for storage or manipulation. Then, in order to demodulate the digital signals, DAC produces a voltage or current that represents the value presented on their digital inputs. This output would then generally be filtered and amplified for use. To recover the original signal from the sampled data, a demodulator can apply the procedure of modulation in reverse. After each sampling period, the demodulator reads the next value and transitions the output signal to the new value. As a result of these transitions, the signal retains a significant amount of high-frequency energy due to imaging effects. To remove these undesirable frequencies, the demodulator passes the signal through a reconstruction filter that suppresses energy outside the expected frequency range. Thus wav was the ideal audio format for the piano sound, because DAC can output the data directly to the speaker to demodulate.[2]

[1] "WAV," *Wikipedia*, 01-Oct-2022. [Online]. Available: https://en.wikipedia.org/wiki/WAV.
[2] "Pulse-code modulation," *Wikipedia*, 27-Oct-2022. [Online]. Available: https://en.wikipedia.org/wiki/Pulse-code_modulation.

The WAVE file format is a subset of Microsoft's RIFF specification for the storage of multimedia files. A RIFF file starts out with a file header followed by a sequence of data chunks. A WAVE file is often just a RIFF file with a single "WAVE" chunk which consists of two sub-chunks -- a "fmt" chunk specifying the data format and a "data" chunk containing the actual sample data. The "fmt" chunk is a specific container format that includes a four-character tag (FourCC) and the size (number of bytes) of the chunk. The tag specifies how the data within the chunk should be interpreted, and there are several standard FourCC tags. Tags consisting of all capital letters are reserved tags. The outermost chunk of a RIFF file has a form tag; the first four bytes of chunk data are a FourCC that specify the form type and are followed by a sequence of subchunks. In the case of a WAV file, those four bytes are the FourCC. The remainder of the RIFF data is a sequence of chunks describing the audio information. The full wav format with RIFF header can be visualized by the following figure.

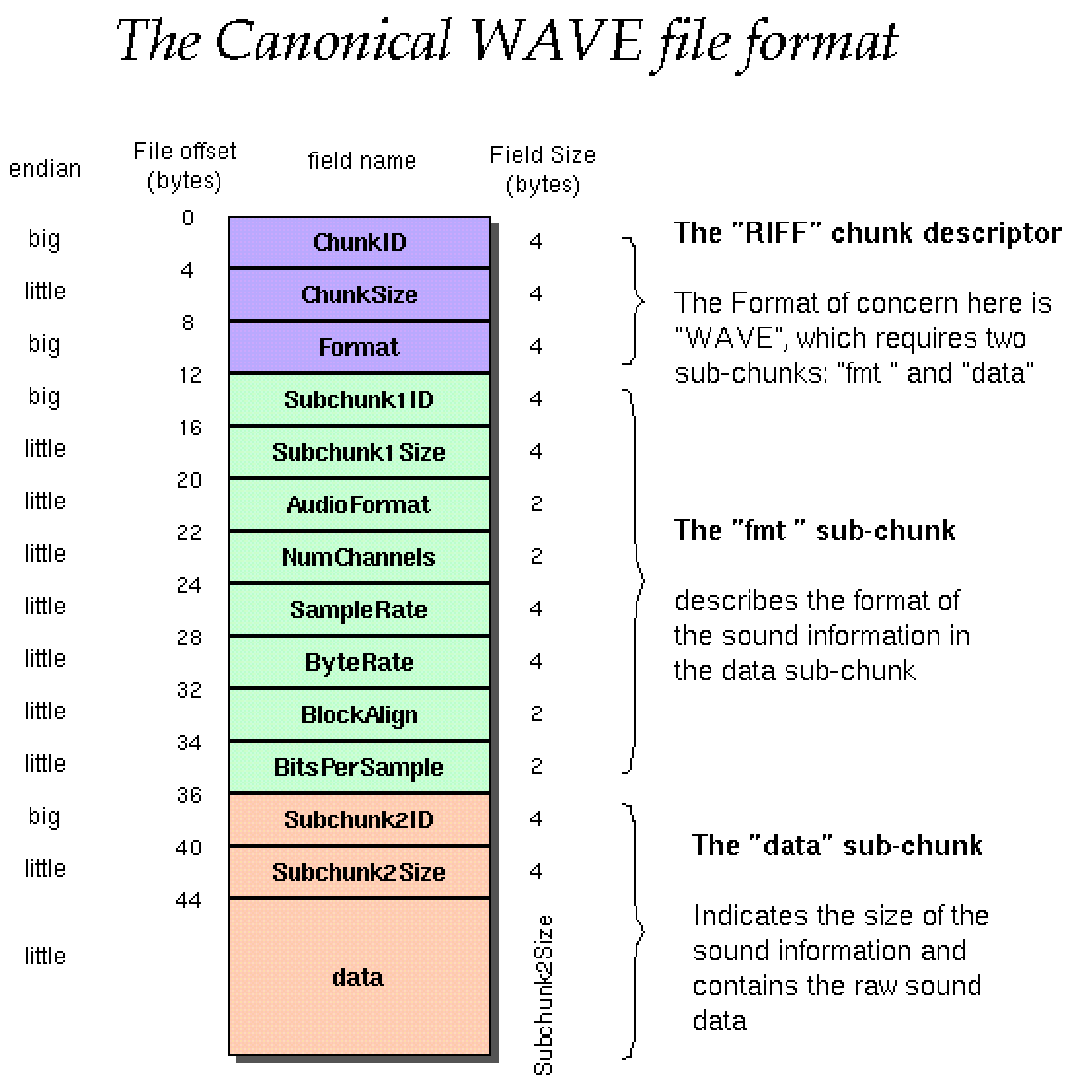


Figure 1. WAVE RIFF format

Low Level Description

For this subsystem, the first object is to obtain the piano note sound file. The file is from C4 to B5[3], one octave. Then Audacity is used to transform the file. The ideal sound file should be in wav format, signed 16-bit PCM. If using a hexadecimal file reader, such as HxD, the transformed wav file can be read with the RIFF header to check the status. For the piano note that was download and transformed, it is in PCM encoding, which is ideal; stereo channels, which is acceptable, the sound may be longer than expected compared to mono channel; the sample rate is 44.1kHz, which is a very usual rate; bits per sample is 16, just as the number that wav is transformed.

In order to process all converted wav files, a python script can be written to process them together. For one wav file, there are about 300 KB. There is no need to use all of them, and there is still empty space at the beginning and end. So in the python script, the bytes after 70000 are saved, until there are 25000 bytes. It should be noted that the wav data chunk is in little endian, but the speaker in DAC should be using big endian, so there should be a conversion for the data.

```
    with open(wavfile, 'rb') as openfileobject:
        for data in iter(partial(openfileobject.read, 2), b''):
            dataw = hex(int.from_bytes(data, byteorder='big', signed=False))
```

To input the array into the DAC with DMA, HAL_DAC_Start_DMA should be used. For the 4th argument, the data size to input into the DAC, it can be modified to change the length of the sound. For the 5th argument, because DAC only has 8B_R, 12B_L and 12B_R. Both 8B_R and 12B_R can be used.

```
    HAL_DAC_Start_DMA(&hDAC1, DAC_CHANNEL_1, (uint32_t*)dow,15000,
DAC_ALIGN_8B_R);
```

[3] "LuckyLux," *pixabay,* 11-Oct-2022. [Online]. Available: https://pixabay.com/sound-effects/search/luckylux/?manual_search=1&order=none

Subsystem: USB Keyboard user input

Introduction

The PS/2 interface has been gradually replaced by USB. Only a small number of desktops still provide a complete PS/2 keyboard and mouse interface. A small number of machines do not have PS/2, and most of the machines only provide a set of keyboards. And the mouse can be shared PS/2 or only available for the keyboard. Some mice and keyboards can use a converter to convert the interface from USB to PS/2, and there are also adapter cables that can be branched from USB to PS/2 interfaces for keyboards and mice.

But because USB has a more efficient and stable transmission rate, the bandwidth and rate of PS/2 are far behind that of the USB interface. So the PS/2 interface was gradually replaced by the USB interface over time.

However, because the USB interface can only support a maximum of 6 keys on the keyboard without conflict, and the PS/2 keyboard interface can support all keys at the same time without conflict. Therefore, most motherboard PS/2 keyboard communication ports are still reserved. When the PS/2 interface is connected to the keyboard device, the difference we can feel is very small compared with the USB interface. The most obvious difference is the key conflict, so the current mechanical keyboard is rarely used between the PS/2 interface. superior.

What we are going to use in this project is a USB keyboard, so we can find HID_KEYBD_Info_TypeDef in the USB driver file. This struct defines all the information we can get about keyboard input.The subsystem of the USB keyboard input needs to be able to return the required key input and all key inputs that do not conflict in the final result presentation.

Low Level Description

First download the required USB driver files and other required files from the lab website. After configuring the required files, we first need to declare a variable of USBH_HandleTypeDef data type for initialization. In the main function, we need to use the USBH_Init function to initialize our handler, and then use the USBH_RegisterClass function to declare our USB processing type. Since we use a keyboard, we use HID. Finally, use the USBH_Start function to complete the initialization.

```
    USBH_Init(&husbh, USBH_UserProcess, 0);
    USBH_RegisterClass(&husbh, USBH_HID_CLASS);
    USBH_Start(&husbh);
```

Then we need to use the USBH_HID_EventCallback function to accept the input interrupt of each USB device. First of all, we first declare two variables of HID_KEYBD_Info_TypeDef data type, one is used to accept the information value returned by the keyboard, and the other is used for subsequent conflict-free processing. Then we declare a char type variable to accept the returned characters, and then declare a HID_HandleTypeDef type variable to test the input result. First use USBH_HID_KeybdInit to determine whether the input is a keyboard.

```
if (HID_Handle->Init == USBH_HID_KeybdInit)
```

Then we call the USBH_HID_GetKeybdInfo function to capture the returned input value. It is worth mentioning here that even though uint8_t keys[6] is an array of size 6 in the struct of HID_KEYBD_Info_TypeDef, the USBH_HID_GetKeybdInfo function can only return the 0th value. This is not in line with our design, because it is impossible for the piano to only emit the 0th sound when multiple keys are pressed. This requires us to find ways to obtain other worthy methods. First of all, after testing, if multiple keys are pressed at the same time, the function can only return the 0th value, but keys[6] is filled with other pressed keys. So at this point we can write a for loop to detect the last character that is not '\0'. Because in the absence of input, keys[6] defaults to 6 '\0' internally. At this point, the variables declared before come in handy. We store the found characters in the 0th bit of the variable keys[0], and then call USBH_HID_GetASCIICode to get the characters we want.

Subsystem: other codes

This part is mainly to go through the entire code, and to supplement the overall understanding of the entire document.

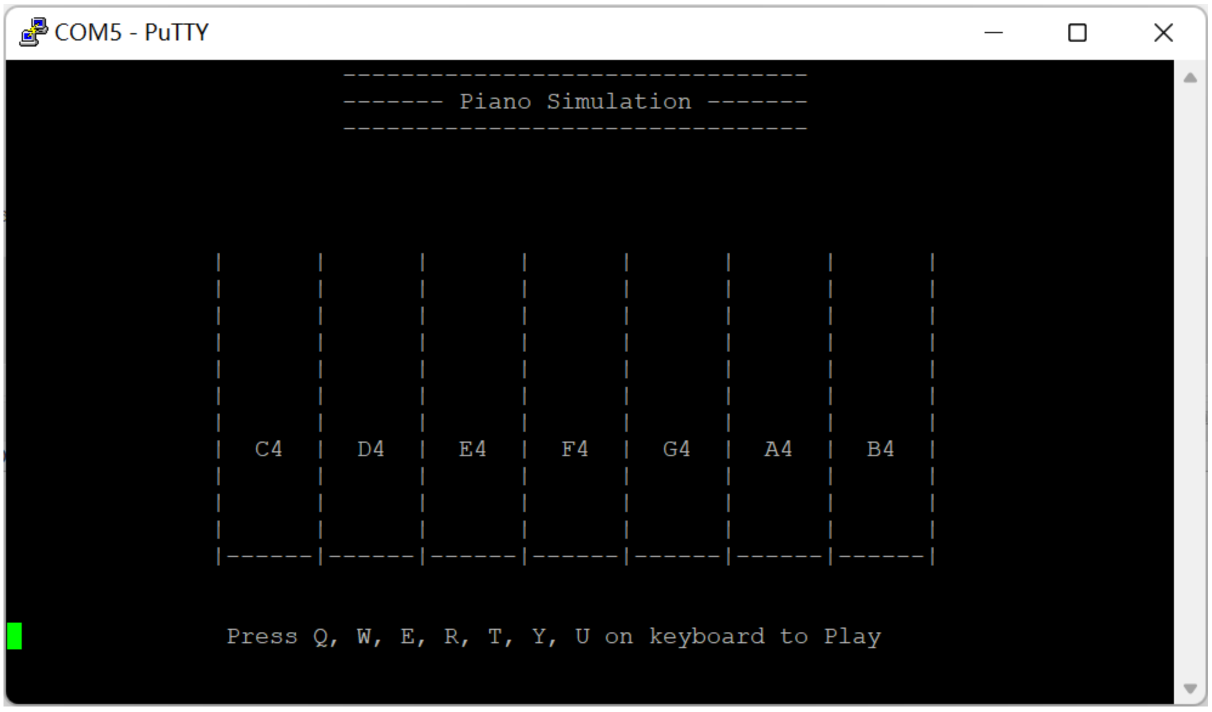
This final solution has a lot of initial setup that needs to be perfected. We need to use Timer, DAC and DMA. It's a complex linkage mechanic, but we decided to try something new. DAC will be triggered by Timer, and then the signal will be transmitted by DMA. First, in the setting of Timer, we declare a variable of TIM_MasterConfigTypeDef. Select the MasterOutputTrigger as TIM_TRGO_UPDATE, and select the MasterSlaveMode as TIM_MASTERSLAVEMODE_DISABLE. Finally, set Timer as master through HAL_TIMEx_MasterConfigSynchronization. In this way, we can control the DAC output through the Timer. Then enter the DAC settings, here we need to set DAC_Trigger to DAC_TRIGGER_T6_TRGO. The initialization setting of DMA can still be done according to lab5.

```
    TIM_MasterConfigTypeDef sMasterConfig;
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
    HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig);
```

Then, we need to make some special settings for the USBH_UserProcess function. This function is the function we bound in USBH_Init, which means that it will be called when the USB device changes. We can judge in this function that if it is detected that the keyboard is plugged in, we will start our simulation. In the case of id == HOST_USER_CLASS_ACTIVE, print the user interface in the terminal and return a parameter that can be started to main. Then we return to the main function. When the detection condition of while is true, it will start to loop the USBH_Process function, that is, to continuously judge the USB conditions and environment. Finally we return to the USB callback function. After getting the input through the keyboard, we can perform the corresponding DAC conversion through the input characters.

Results and Analysis

The screenshot demonstrates how the program works. Pressing the corresponding keys on the keyboard would generate the piano note.



```
--------------------------------
------- Piano Simulation -------
--------------------------------

|      |      |      |      |      |      |      |
|      |      |      |      |      |      |      |
|      |      |      |      |      |      |      |
|      |      |      |      |      |      |      |
|      |      |      |      |      |      |      |
|      |      |      |      |      |      |      |
|  C4  |  D4  |  E4  |  F4  |  G4  |  A4  |  B4  |
|      |      |      |      |      |      |      |
|      |      |      |      |      |      |      |
|      |      |      |      |      |      |      |
|------|------|------|------|------|------|------|

   Press Q, W, E, R, T, Y, U on keyboard to Play
```

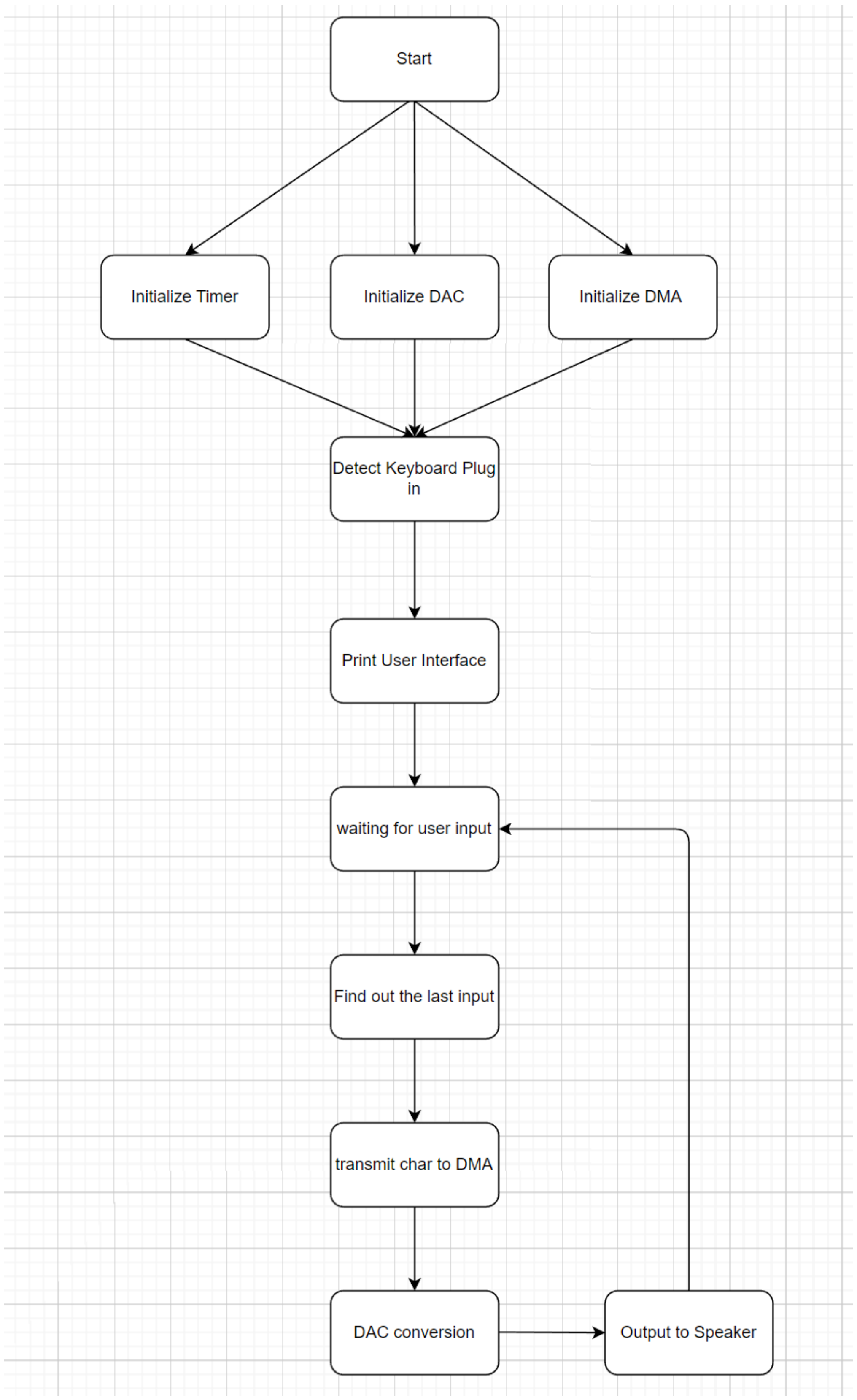
Figure 2. Piano User Interface

Conclusions

Through this final project, we have conducted a general review on what we have learned this semester. Combining escape sequence, timer interrupt, DAC, DMA and USB HID input together was interesting and complicated. We strengthened our image on those topics and also learnt a lot of new things, including the input and output of wav audio files, the input of USB keyboard and so on. Those knowledge might not be closely correlated with microprocessors, but they are still important topics to discover to accomplish our goals. In all, the outcome of this project and course should be improving our self-studying ability, hands-on experience and manual reading. We believe that in the future study and life, we can use this spirit of exploration and the courage to not give up easily when encountering difficulties in more fields.

Appendix

[1] Final project flowchart.


Start
Initialize Timer
Initialize DAC
Initialize DMA
Detect Keyboard Plug in
Print User Interface
waiting for user input
Find out the last input
transmit char to DMA
DAC conversion
Output to Speaker