ECE 395
Spring 2018 - Looper Board

Napat Sutthasinwong (sutthas2)
Tony Rodriguez (arodri23)
Jacob Allen (jiallen2)

April 2018

Section: A - Lippold Haken
# Contents

1 Introduction ........................................ 4  
   1.1 Inspiration ..................................... 4  

2 Overview ........................................... 5  

3 Design ............................................. 6  
   3.1 State Machine ................................... 6  

4 Hyve Synthesizer ................................... 7  
   4.1 Schematic ...................................... 7  

5 LPC1768 ARM Cortex-M3 ................................ 8  
   5.1 General Purpose I/O ......................... 9  
   5.2 Schematic ..................................... 9  
   5.3 Universal Asynchronous Receiver Transmitter 10  
   5.4 Block Diagram ................................ 10  
   5.5 General Purpose Direct Memory Access Controller 10  
   5.6 Block Diagram ................................ 11  

6 Analog Digital Converter ............................ 11  
   6.1 Schematic ..................................... 12  

7 Digital Analog Converter ............................ 12  
   7.1 Schematic ..................................... 12  

8 Flash Memory ...................................... 13  
   8.1 In Application Programming ..................... 13  

9 I2C ................................................. 14  

10 Testing ............................................ 15  
   10.1 Issues ........................................ 15  
   10.2 Arduino ....................................... 16  

11 Conclusion ...................................... 16  
   11.1 User Instructions ......................... 16  
   11.2 Improvements ............................... 17  

12 Software Documentation ............................ 17  
   12.1 Recording .................................... 18  
   12.2 Playback ...................................... 18
1 Introduction

Have you ever heard a musician creating a song starting off with a beat and slowly adding to the piece? These songs utilize a looper pedal for creating a live accompaniment. A looper pedal allows a user to record audio input in real time and loop the performance. Our goal is to re-create the physical looper pedal on the ARM Cortex-M3 development board, which features the LPC1768 MCU (micro-controller unit).

1.1 Inspiration

We are not only intrigued with the use of a looper pedal but the design as well. Learning and knowing how the hardware and software behind the device works is something we would be very interested in as we have the capability to extend the functionality beyond the basic design.

Professional musicians such as Ed Sheeran or Reggie Watts take advantage of using a looper pedal to create astonishing compositions. From professionals we can see the potential and creative use of such a ‘simple’ device.

Here’s an example being played, URL:
https://www.youtube.com/watch?v=BdHKr9RXTc&feature=youtu.be&t=2m31s
2 Overview

As with every looper pedal, the design is based on the original functionality listed below. We separated some of these functions into different button inputs for ease of use.

- **Record:**
  The user is able to record a successful audio data used for playback and looping. This data is stored into a buffer array, sampled at 44100 Hz.

- **Playback:**
  The user can playback the audio data buffer once recorded or overdubbed.

- **Loop:**
  The user can playback the audio data buffer in an infinite loop until specified to stop. This loop is achieved by using a circular buffer technique, setting the head point to back at the start of the array.

- **Overdub:**
  The user can overdub over an existing track beyond the track length. The overdub effect takes places immediately even while overdubbing. This is achieved by setting the head point correctly while recording and a summation of the existing data and track data, continuously.

- **Clear Audio Data:**
  The user can clear any existing audio data for each track by recording again when a track is not currently looping.
3 Design

We plan to use the the Hyve synthesizer for audio data input and a few button controls for the user. One button is for recording audio while another is for clearing a track. We also have another button for changing the sampling frequency. The ADC, DAC, and GPDMA are all on chip. The flash memory is also on chip but we can extend the memory externally as well with our I2C interface. We are using a mini external speaker to output audio signals. Building this project on our development board gives us confidence to continue on our own printed circuit board.

3.1 State Machine
4 Hyve Synthesizer

Designed by Skot Weidmann, the Hyve synthesizer allows a user to control analog signals by touch sensitivity. We found the user control to be fascinating as one can create polyphonic sounds to the Nth degree. Position, direction, speed, and pressure force all contribute to the output signal. And so we decided to use the Hyve as our audio signal input into the system.

One of first tasks is to understand the electrical components that require the synthesizer to work. We needed to solder each chip listed below and tune them by keying the potentiometers accordingly. Using a function generator, the Hyve works well at a setting of 9 volts with a current limit at 0.4 amps. We used the oscilloscope to measure a clean signal. There is a 3.5 mm output jack in which we connected corresponding lines (analog signal, power, ground) to the development board. The output voltage is for this line is rated at 5 volts.

4.1 Schematic

![Schematic Diagram](image.png)
Before the LPC1768 board, we tried to build the system using the Armstrap Eagle which is another development board based on the 168 MHz Cortex-M4 ARM chip. Unfortunately, we could not seem to setup and talk to the board easily using Eclipse software. The issue seemed to be something wrong with the device driver recognition or initialization commands.

Prototyping on the LPC1768 will give us confidence in continuing the system design on our own printed circuit board. We only needed to use the ARM chip from the board to test out functionality designs through Keil software.

The chip has a 4GB address space that incorporates several distinct memory regions:

- Core: Cortex-M3 32-bit RISC
- Operating Frequency: 100MHz Max
- Operating Voltage: 2.4-3.6V (3.3V typical)
- Memory: 512kB Flash, 64kB RAM
- Communication Interfaces: UART (4), I2C (3), SPI (2), PWM (6), ADC (8), ADC (1)
- Debugging/Programming: supports SWD interfaces, supports ISP through UART
5.1 General Purpose I/O

Before converting the output analog signal from the Hyve, we wanted to test communication connection to the chip through GPIO pins. The chip provides a separate register for each pin and configuration for pin modes. These pins will be used to capture analog signal input, produce analog signal output, and simulate user control buttons.

These pins have configurable pull-up/down resistors and different modes that were useful such as open drain mode. The open drain mode was useful when simulating a pin line as a button. We noticed that when a line is connected to a pin but is open (acting like an antenna), there is noise / signal interference that affects the signal value of the pin. By pulling down the value of the pin when not connected we can mitigate this noise. All GPIOs are located on an AHB (Advanced High-performance Bus) bus for fast access by the GPDMA Controller.

5.2 Schematic

![Schematic Diagram]

5.3 Universal Asynchronous Receiver Transmitter

Testing the GPIO pins using on board LEDs was not enough for us to display data information. Using UART connection we can accurately display the data lines from GPIO pins through PuTTY software. We needed to correctly connect the transmitter and receiver lines from the UART to our JTAG (Joint Test
Action Group) board in order to talk to PuTTY. This runs at 3.3 volts.

The PuTTY settings that enables serial connection was specifying the correct serial line and setting the speed / baud rate to 115200.

5.4 Block Diagram

![Block Diagram Image]

5.5 General Purpose Direct Memory Access Controller

The DMA (Direct Memory Access) controller allows us to use memory to memory transactions. Each DMA stream provides unidirectional serial DMA transfers for a single source and destination memory region. The GPDMA has eight channels on the AHB multi-layer matrix in which we use for the GPIO, UART, ADC converter, DAC converter, and timer match signals. We needed to initialize and correctly configure this controller correctly for use.

Below are some these settings parameters needed:

- Channel Number
- Source and Destination Memory
- Transfer Size, Width and Type
6 Analog Digital Converter

After testing out general connection to the board, the next step was to try linking the analog signal output from the Hyve to the LPC1768 as an analog input to one of the pins. We used the on board 12-bit successive approximation ADC to achieve this. Using the GPDMA we are able to set up a channel for the ADC. The conversion rate is set at 200K Hz. Clocking for the ADC is provided by the APB clock. A programmable divider is included in each converter to scale this clock to the clock needed by the successive approximation process (maximum 12.4 MHz). A fully accurate conversion requires 31 of these clocks.

The AHB matrix interconnect provides a separate bus for each AHB master. Split APB bus allows for higher throughput with fewer stalls between the CPU and DMA. A single level of write buffering allows the CPU to continue without waiting for completion of APB writes if the APB was not already busy. To use this ADC properly we needed to setup the channel and pin configurations. The main while loop enables, disables, and re-setup the channel to get a new ADC input value at every loop. Pin 24 was used for analog signal input. The ADC input can multiplex among 8 pins. We also needed to disable the GPDMA interrupt.

- Measurement range VSS to VREFP (3V)
- 12-bit conversion rate up to 400 kHz
- Burst conversion mode for single or multiple inputs
7 Digital Analog Converter

The next step after ensuring the ADC was working properly is to try converting the ADC signal value back into analog signal. The development board came with a small speaker in which we can link analog values from the DAC as input and connect properly (power 5V, ground). The DAC is 10-bit with a maximum update rate of 1 Mhz. We needed to set the pin configurations properly as well.

The DAC value was a simple as setting it from the ADC value at every main loop iteration. At this point, the system was successfully converting the audio signal both ways and the user can hear the correct frequency out the speaker live.
8 Flash Memory

After getting live audio data passing through the system working, we needed to set up a buffer to capture this data for later use. The user specifies this action through a simulated button for recording the data. Unfortunately, the default memory location when using a simple array (of size \( \frac{n}{2} \approx 60 \) about) is not enough. We could store a small buffer to test random values but it does not re-create the original signal recorded. And so we tried to learn and use flash memory instead as it provides 512 kb storage and since we are sampling at such a high rate. From calculation, we expected the flash memory to give us a few seconds (\( \tau \approx 3s \)) of recording time.

When a user program begins execution after reset, interrupt vectors are set to point to the beginning of the flash memory. Using this limited memory, we are creating our audio data buffer (value at each address). Instead of an array, we use a pointer to store and grab a value. We used in application programming to copy the data from RAM over into flash.

8.1 In Application Programming

In IAP, the application code is performing erase and write operation on the on-chip flash memory. The on-chip flash memory is not accessible during IAP operations. When the user application code starts executing, the interrupt vectors from the user flash area are active. We had to disable interrupts before making an IAP call. We also tried to ensure that user interrupt vectors are active in RAM, interrupt handlers reside in RAM.

Each sector size varies (4kb, 32kb) and is labelled by a number (1-29). The IAP write command has a lowest setting of 256 bytes to be written. This means we wanted to create a small buffer on the RAM of size 64 in order to copy over values consecutively on the addresses. That means we had to store the buffer fully during recording and only after the buffer is full do we copy it into flash, and clear the buffer for more recording values. We used a pointer to keep track of where the next memory address should be used as destination for copy.

- Prepare Sector: This command is executed before erasing and writing to flash. We also execute this command after every IAP write command during the main loop because the IAP write command causes relevant sectors to be protected again.
- Erase Sector: This command is executed before entering the main loop to clear and existing data in the corresponding flash memory.
- Write Sector This command is executed during the main loop to copy the audio buffer data on the RAM to flash when ready.

Below is a diagram of how we setup the IAP command and result calls in register addresses. The parameter sizes varies with command calls.
I2C is a protocol for peripheral communication. We wanted to implement it with the non high-speed features since it was easier to test them with the Arduino we had acting as the slave/protocol analyzer for our test purposes. This actually caused some issues down the road that prevented getting one of the final parts of I2C functioning, and we should have tried using the extra arm chip provided earlier in the semester to test this instead.

The part that had an issue come up was in performing the second half of a master initiated read of the slaves data. So looking at the diagram below that would be after the first ACK from the slave. The master would read the SDA line, and then ACK the slave in return. The issue here is the Arduino runs a lot slower than the arm chip, and it can mess up the timing so we did have clock stretching to help with this. Some specifics of the Arduino hardware made it difficult to pin-down what the issue was, but we have some predictions in the Arduino section.

Flash memory would not be enough memory for a perfect final product since it allows us to only store a few seconds of recording. In order to achieve a longer record time similar to a real looper pedal, we would need to extend the memory externally. Writing the I2C interface was a challenging process to set up this communication protocol but would allow us to talk externally and transfer data.
10 Testing

Using the user manual and schematic sheet, we carefully wired the hardware correctly. But there were times when we were unsure if an issue was hardware or software related. Noise interference was a slight problem on some parts of the breadboard and open wire. We were able to see how clean a signal was through the oscilloscope and use open drain mode on a pin if needed. Trying to see the ADC value input to the system was done through printing from the UART / JTAG into PuTTY. All testing done was incremental and making sure each component was working correctly.

Once a user records a sample, the system automatically loops using a circular buffer technique by keeping track of the head point. We tested looping a simple one frequency sound to polyphonic sounds, to multiple notes. We also added in some extra features such as changing the sampling frequency by down-sampling or up-sampling at double rate controlled by the user.

10.1 Issues

- Noise:
The looper board is able to pass audio samples from the Hyve to the speaker but you can notice very minimal noise in the background. In the first working version the noise was much more significant and we reduced this noise as much as possible. The issue lied on the software side as delays such as waiting for a channel to be set or the order of when to grab an ADC value affected the output signal.

- Prints:
Printing values into TTY was very useful in the sense that we can see some real data. But we eventually found out that trying to print every sample affects the output signal and does not actually print every sample since we are sampling at such a high frequency. The print delay is not fast enough and so to test values correctly we would store values into a buffer and print after the main loop (break on command).

- Reading Flash:
Once we were able to read and write from flash memory we noticed that the playback was speed up by a certain factor. The output signal was closely related to the original signal but off by some factor. This was due to the difference in sampling frequency and playback frequency in which we had to fix.

- Speaker:
The mini speaker we used was not good enough to output polyphonic sounds super clear. Plugging in headphones into the 3.5mm jack on the Hyve, you can hear the distinct sounds much more clearly.
10.2 Arduino

The arduino was meant to allow for testing the code on our arm chip in a way that gave us very easy serial debugger output, and allowed us to quickly develop test cases. Due to what we wanted to use the Arduino for though this actually turned out to cost a lot of time, and in the end was part of the reason we didn’t get I2C working properly. The Arduino occasionally would break the USB spec for voltage which does not really matter for the testing purposes, but did make us worry about if other problems were caused by it. It would also not be able to properly interact with analog lines if it was not plugged into wall power which is a rather odd issue. At the end the problem was getting the slave to write to the line and keep pace with the clock. The Arduino seemed to have issues reading the masters ACK which is very odd since it was able to read its address, and any other data we sent to it. The underlying issue might have something to do with the internal resistors not being able to keep up with the clock when it has to do its own manipulation of SDA line voltage.

11 Conclusion

A real musician may not use our final product due to a unclean hardware design (musicians usually use looper pedals with feet, sturdy build), but we are content with the progress made on the functionality side. We didn’t expect it but it took many weeks for us trying to get the basics up and running. The design is far off from perfect as we have many more elements in mind to add to the system. Skot Weidmann surprised us with his appearance during demo day and was intrigued enough to post a quick demo on his instagram.

11.1 User Instructions

- Synthesize Mode:
  The user can freely play live audio sound from the Hyve to the speaker.

- Record:
  The user can capture a few seconds of recording time from the samples played.

- Looping:
  After a successful record the user can still play live audio sound on top of the recorded sound loop.

- Sampling frequency: The user is able to change the sampling frequency of the audio data playback for looping by a factor of 2 (up-sample or down-sample).

- Clear:
  The user is able to clear a track from a button.
11.2 Improvements

- **Overdubbing**: Overdubbing would complete the design of a basic looper pedal. One would have to keep track of the head point correctly while adding new samples on top of existing data. To go even further one can try to clear last overdubbed samples on an existing track by keeping track of each overdub.

- **Control Mode**: In addition to the synthesize mode, one can add a control mode and use the Hyve as a control board. In this mode the user can manipulate the track by use touching the board in a certain position, speed or direction.

- **Multi-track**: With extended memory, we can allow for more than one track at a time. Each track can be represented by a buffer loop and be manipulated separately.

- **Sampling frequency**: One can extend the sampling frequency change values from discrete to continuous by using an algorithm to keep track of the factor and touch sensitivity input.

- **Filtering**: One can add filtering in order to filter out specified frequencies from the track. This can be done depending on which filter is needed.

12 Software Documentation

All low-level C code is included in the zip file attached with this project PDF. We will not include all the code here as it seems too long. This means all setup code for GPIO and GPDMA controllers / handlers will be in the source code files. Included below is a few of the main logic for main functionalities such as recording and playback within the main while loop. The I2C code is also too long to include here as there are many parts. Please check the source code files for direct information.
12.1 Recording

```c
// Record
if (is_recording == 0) {
    // printf(RECORDING... \n);
    if (buffer_set_counter == 0) {
        // printf(SETTING BUFFER: %d \n\n, buffer_set_counter);
        buffer_loop(buffer_set_counter) = adc_value;
        // printf(SETTING BUFFER: %d\n\n, buffer_loop(buffer_set_counter), adc_value);
        buffer_set_counter++;
        //delay ms(10);
        is_empty = 0;
    }
}

// Buffer full
if (buffer_set_counter == 64 && copy_flash_counter == 976)
```

12.2 Playback

```c
if (play_back && is_recording) {
    // printf(BUFFER: %d\n\n, *check_addr + buffer_counter);
    if (buffer_counter == (64 * copy_flash_counter) - 1) { buffer_counter = 0; }

    // Downsampling
    /*
    * if (play_back_idx % 4 == 0) {
    *     dac_value = adc_value + *(check_addr + buffer_counter);
    *     buffer_counter++;
    *     play_back_idx++;
    * } else {
    *     dac_value = adc_value + *(check_addr + buffer_counter);
    *     play_back_idx++;
    * }
    */
    dac_value = adc_value + *(check_addr + buffer_counter);
    buffer_counter++;
```

url: https://www.waveshare.com/w/upload/3/3f/LPC178x7xUserManual_EN.pdf
url: https://www.waveshare.com/w/upload/3/30/LPC178x7xDatasheet_EN.pdf