Conffti:
reimagining digital sound synthesis

conFFTi is a digital musical synthesizer. It takes user input from a MIDI keyboard and outputs the corresponding sound with analog wave generation. It combines the benefits of traditional analog sound generation and flexible digital reprogrammability.

Extending upon this previous project, I'm also presenting a concept exploration that I conducted individually — Synthdesk. In the following slides, I will first walk through the technical design behind conFFTi, then I will go over how the extended project came to life.

(Original project) conFFTi
ECE Capstone group project (3 members)
May 2021

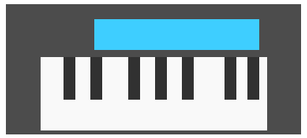
(Extended project) Synthdesk
Individual project
November 2024

What is conFFTi?

conFFTi is a digital musical synthesizer implemented on a Field Programmable Gate Array (FPGA). We used a commercial MIDI keyboard for collecting inputs. The system accepts real-time inputs from MIDI keyboard, generates digital sound signals in the FPGA, and outputs the data as audible sound through a digital analog converter (DAC).

the conFFTi system

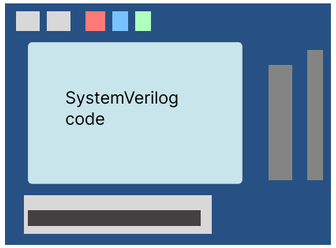
Input Detection



MIDI keyboard

MIDI breakout

Processing and Generation



FPGA

Output

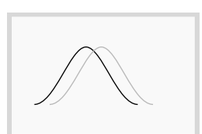
3.5mm audio jack

DAC

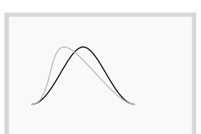
How it works

The audio processing pipeline is the heart of our system. On top of the basic oscillators that generate signals, we also added modules for aiding creative compositions. Our synthesizer is capable of producing effects such as unison detune, arpeggiation, fade in/fade out, and 8 unique timbres.

In order to modulate these effects, the FPGA takes 2 sources of user inputs: **MIDI keyboard presses**, and **operations directly on the FPGA board**. The main area of improvement for my extended concept study is to have a unified, centralized control surface.

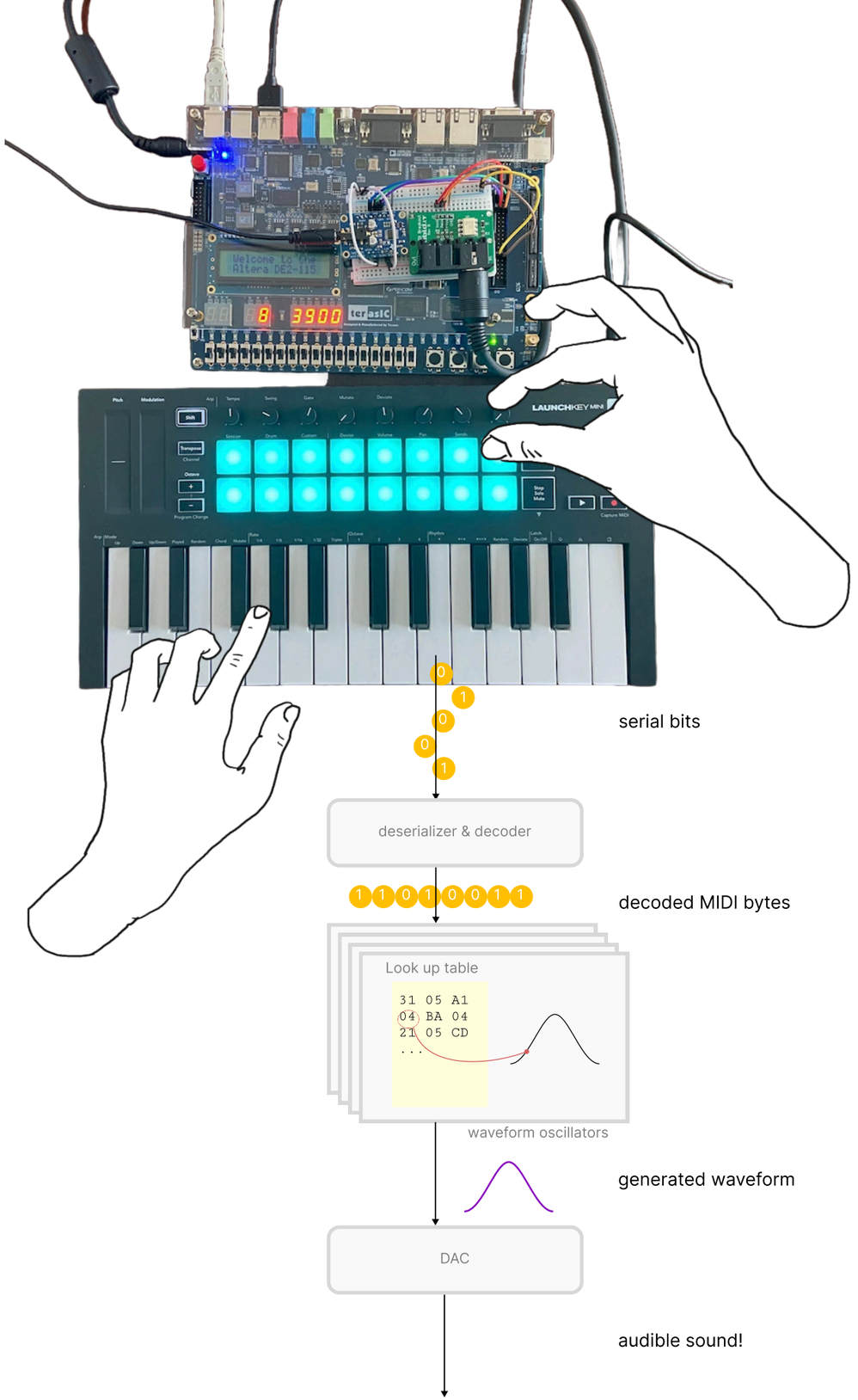


phase shifter



duty cycle modulation

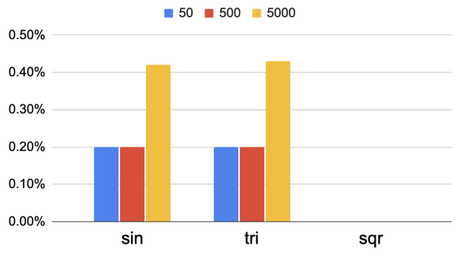
demo video:
https://www.youtube.com/watch?v=zdNPF3ZBTvo&ab_channel=MichelleChang



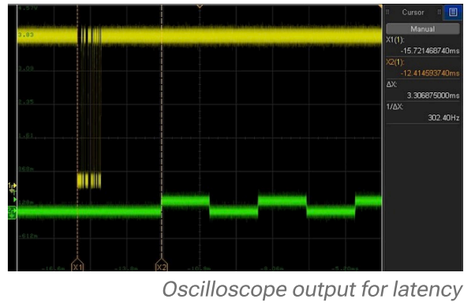
Device Verification

Before we started implementaion, we lookup common limiattions and baseline performance metrics of hardware and software synths. We lined out the following requirements, which we revisited in the end:

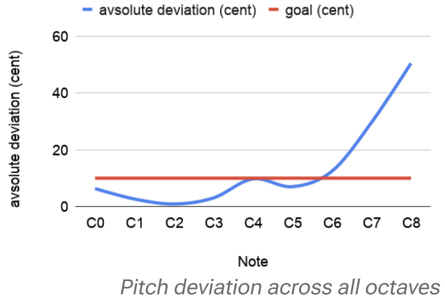
- Latency: Achieved
- Pitch Deviation: Partially achieved. Exceeded the 10 cents requirement for notes above C6.
- Signal distortion: Achieved.
- Add programmable features: Achieved.
- Industry standard output audio (44.1kHz, 2 channel, 16 bit): Achieved.



Signal shape distortion



Oscilloscope output for latency



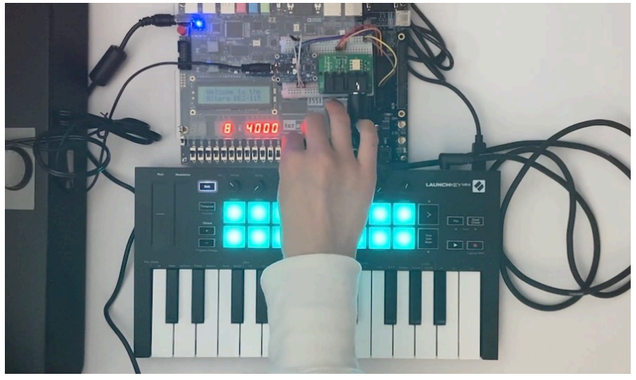
Pitch deviation across all octaves

User testing

We accomplished our vision in creating a versatile system that brings creative reprogrammabilities into traditional analog sound generation.

However, there is space for further improvement — as mentioned before, the control for some of our programmable features is directly on the FPGA.

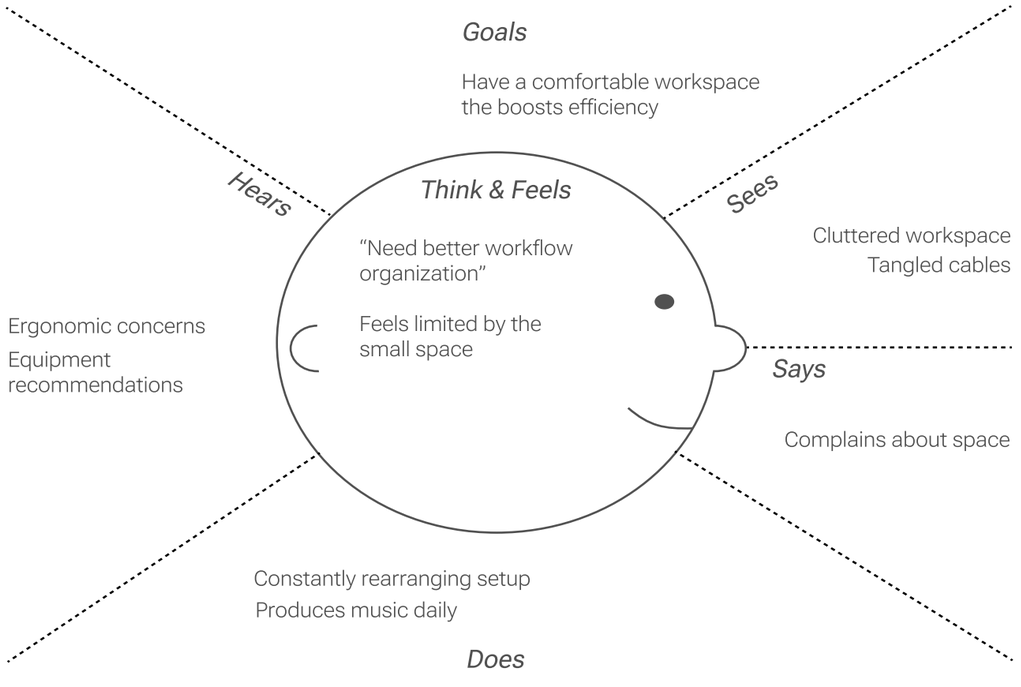
During user testing sessions, some reported that for those without prior experience with this board, our project could be more intuitive if it had a neater, more user-friendly interface.



Demo: FPGA-based interface

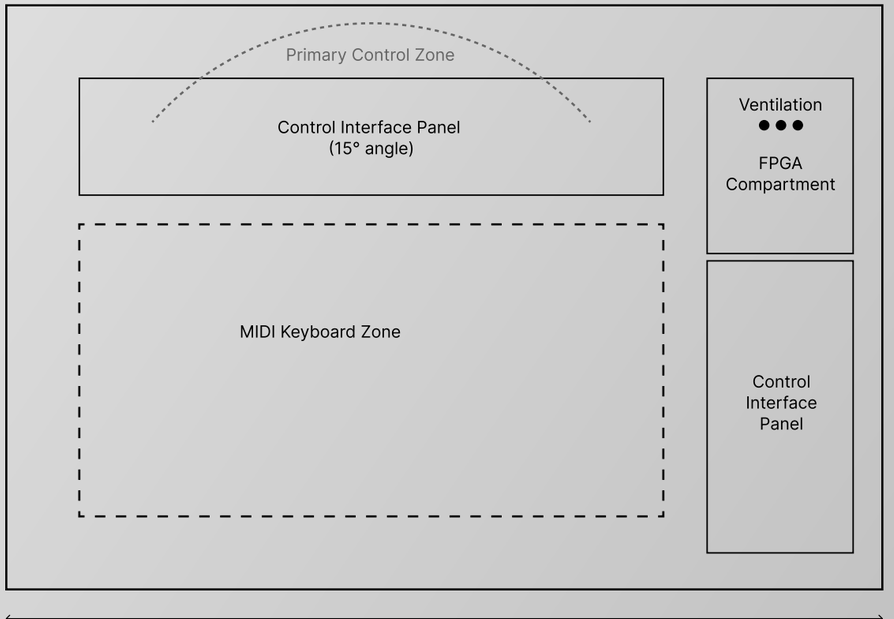
Reimagining... **SynthDesk**

While revisiting this project, I felt inspired to imagine a product that would transform our design into a cohesive, intuitive system. SynthDesk reimagines the relationship between musical workspace and instrument. It solves a critical pain point for electronic musicians: the often cluttered nature of music production workspaces.



A Concept Study: Integrating digital music production with furniture design

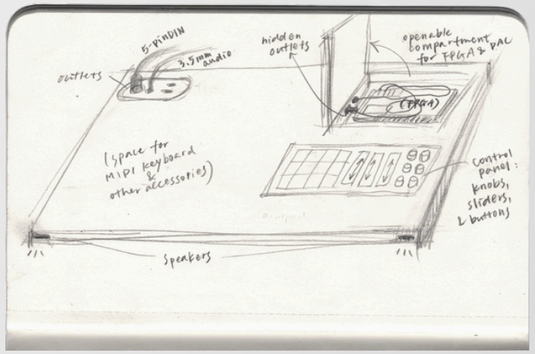
Building upon my experience with ConFFTi, this concept proposes integrating synthesizer controls directly into a desk's surface, creating a unified system that is suitable for compact workspaces. This concept exploration focuses on defining the technical specifications for hardware integration and a reliable control protocol.



Musicians can connect their instruments to the desk outlets to create a FPGA-based synthesis system, all controlled through an intuitive interface:

- Embedded controls and outlets on the desktop on a slanted control panel
- Embedded speakers
- Compartment for housing the hardware: FPGA board, MIDI breakout, and the DAC.

With the sketch, I created the rendered visual wih Midjourney.



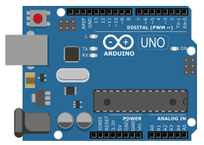
Created with Midjourney

Technical breakdown

The block diagram shows the hardware integration of how Synthdesk would connect to the existing conFFTi system.

Microcontroller (MCU)

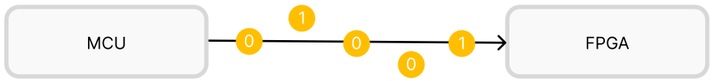
We need a microcontroller for intaking the control signals from the UI and encoding it in a predefined protocol to be sent to the FPGA. We can leverage the Arduino UNO board for its built in I2C protocol capability, adequate memory size, and flexible reprogrammability.



I2C Protocol

We need to define a message structure to encode and decode accurate information over serial data transmission.

The payload will be a fixed **6-byte message** that includes **control type, control ID, value, timestamp, and modifier**. We also need start and end markers for precise timing placement, as well as a byte for length to ensure robustness.



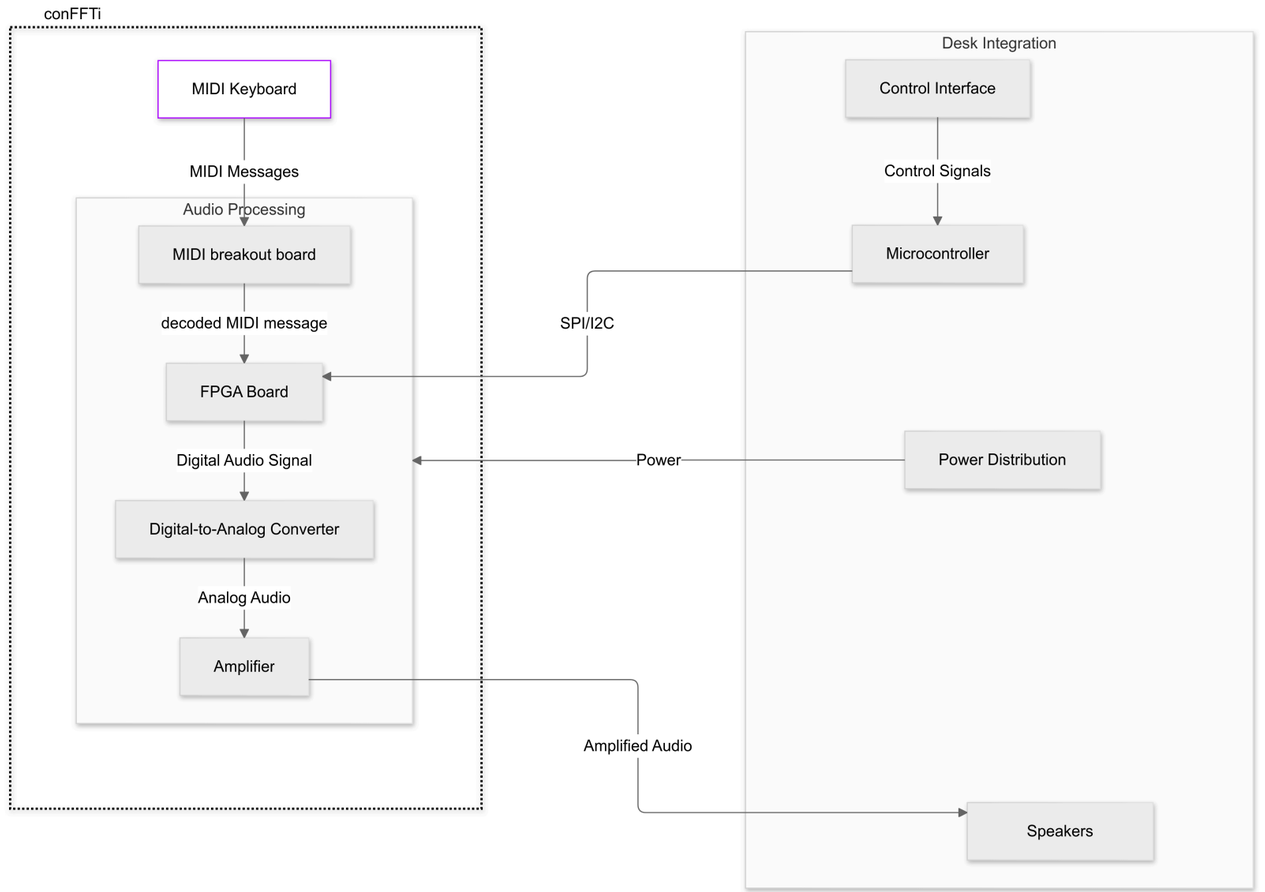
FPGA decoder

Based on the predefined protocol, the FPGA code needs to be updated with an additional deserializing decoder module for translating the messages sent from the microcontroller.

Conclusion

While currently a theoretical study, this project demonstrates the feasibility of embedding digital music interfaces into furniture, considering both the technical requirements and user experience implications.

This intersection of physical and digital design spaces represents exactly the kind of challenge I look forward to tackling through further education and hands-on experience in the program.



Block diagram

Fixed: 0x55	5 5
Usually fixed: 0x06	0 6
controlType	
controlNum	
value (high byte)	
value (low byte)	
modifiers	
timestamp	
Fixed: 0xAA	A A

I2C protocol message visualization

```
void sendControlMessage(byte controlType, byte controlNum, uint16_t value,
                        uint16_t modifiers, byte timestamp) {
  Wire.beginTransmission(I2C_FPGA_ADDRESS);

  Wire.write(START_MARKER);
  Wire.write(0x06);  // length of payload
  Wire.write(controlType);
  Wire.write(controlNum);
  Wire.write((value >> 8) & 0xFF);  // Value high byte
  Wire.write(value & 0xFF);         // Value low byte
  Wire.write((modifiers);  // Modifiers
  Wire.write(timestamp);
  Wire.write(END_MARKER);

  Wire.endTransmission();

}
```

I2C protocol Arduino code snippet