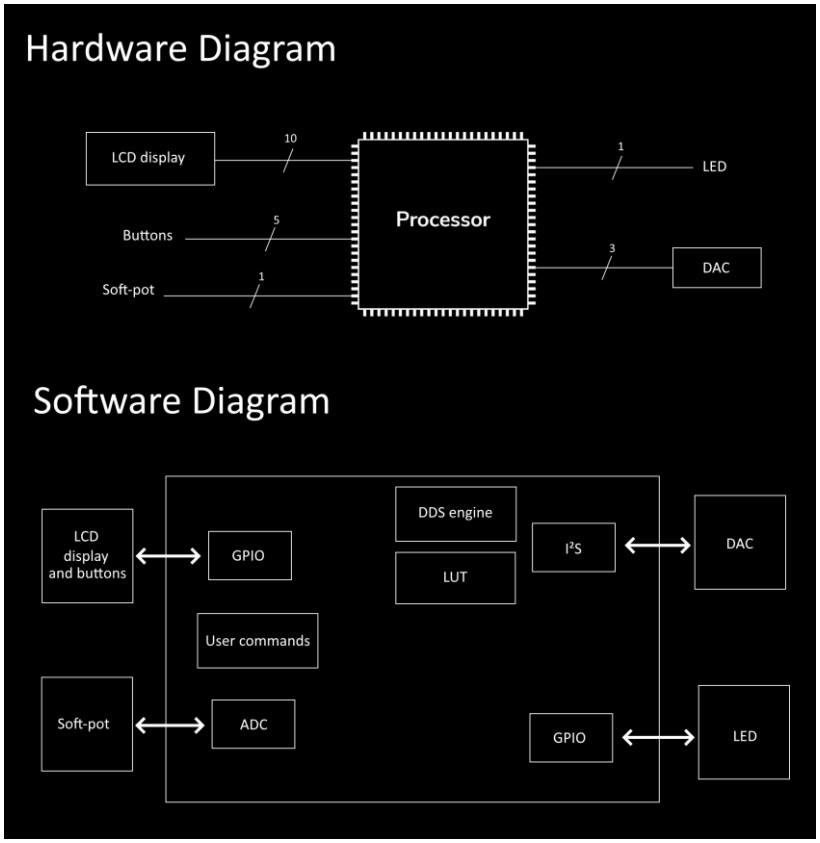
Application description:

A stereo synthesizer that generates up to 6 voices. The input is a ribbon sensor (soft-pot) that mimics a 2-octave keyboard (that can be transposed). Volume, harmonics, waveform and octaves are configured via a simple UI.

Hardware description:

The system uses a STM32H743 microcontroller, a PCM5102A for audio CODEC, a 16x2 LCD screen, 4 buttons with debouncing circuit, a LED to signal input detection, a button with interrupt routine attached to reset the DDS, a separate 5V power supply for the LCD and a ribbon sensor for input. The board has many points for signal testing.



Software description:

The software consists of two main functions: The main loop that runs asynchronously and an interrupt routine that runs at the sample rate. The main loop controls the UI, both LCD and the 5 buttons. The interrupt operates the DDS engine (direct digital synthesis). Many small functions were created to operate the peripherals and the DDS. The DDS is basically a LUT (Look up table) operator. It read the LUT value and then relocate the reading position according to the desired frequency. The system also operates harmonics, multiples of the main frequency. In order to have precise timings, the system uses a fixed point through left and right shifting of the LUT counter. It uses shift instead of multiplication in order to take less CPU processing power. In the DDS interrupt it's also applied volume to the channel and the decreasing volume for the harmonics. Since the interrupt must happen in less time that it takes to another interrupt to begin, all voices are processed in the DDS all the time, whether they are active or not.

A closer look on code:

No code was used from another source apart from the HAL.

For the ADC:

```
// measure the ADC value, condition it to return 25 values
static uint32_t adc(){
    uint32_t average = 50;
    uint32_t counter = 0;
    int32_t value = 0;

    for(counter = 0; counter < average; counter++){
        HAL_ADC_Start(&hadc1);
        HAL_ADC_PollForConversion(&hadc1, 1);
        value += HAL_ADC_GetValue(&hadc1);
        HAL_ADC_Stop(&hadc1);
    }

    value /= (average * 2420);  // value ranges from 27 to 2
    value -= 2;                 // value ranges from 25 to 0

    if(value < 0)
        value = 0;

    return value;
}
```

The function not only read the ADC multiple times and apply an average but also conditions it to be used by the main loop. It returns 0 in case of no pressure on the sensor or a value between 1 to 25, for the 24 keys of a 2-octave keyboard.

For the interrupt routine (only one channel):

```
// left channel
for(harmonic = 0; harmonic <= sound_config.harmonics1; harmonic++){
    counter1[harmonic] += increment1[harmonic];
    if(counter1[harmonic] >= bigmax)
        counter1[harmonic] -= bigmax;
    switch(sound_config.shape1){
    case sine:
        out_l += (uint16_t)((int16_t)LUTsine[counter1[harmonic] >> 10] * sound_config.vol1 / (16 * (harmonic + 1)));
        break;
    case square:
        out_l += (uint16_t)((int16_t)LUTsquare[counter1[harmonic] >> 10] * sound_config.vol1 / (16 * (harmonic + 1)));
        break;
    case triangle:
        out_l += (uint16_t)((int16_t)LUTtri[counter1[harmonic] >> 10] * sound_config.vol1 / (16 * (harmonic + 1)));
        break;
    case sawtooth:
        out_l += (uint16_t)((int16_t)LUTsaw[counter1[harmonic] >> 10] * sound_config.vol1 / (16 * (harmonic + 1)));
        break;
    default:
        break;
    }
}
```

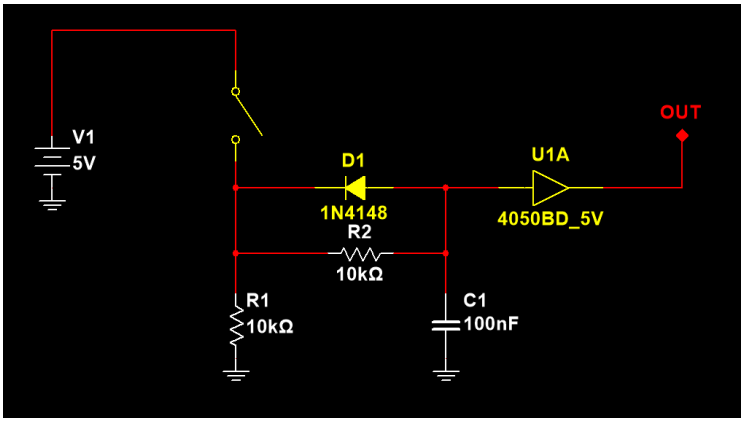
The for loop will run through all of the channel's harmonics, increasing the position in the LUT, multiplying by the channel's volume, and then dividing the result by the harmonic factor: an inverse relation with the frequency of the nth harmonic ($A = {}^{1}/_{F_{hn}}$).

What wasn't done:

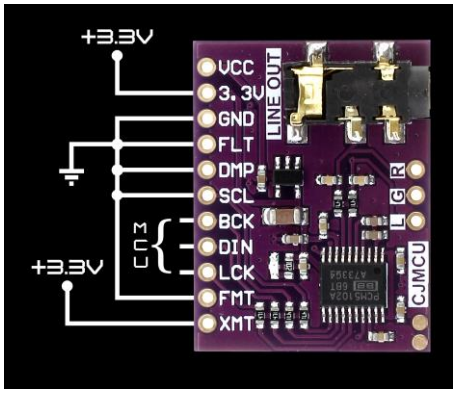
Unfortunately, I didn't count on the polling I²S function being so slow. It takes almost 15µs for it to send the complete package. As a result, I had to slow the sample frequency from 40kHz to 20kHz. With this frequency, I was able to run 6 voices in the DDS. As I wasted a lot of time trying to figure what was wrong with the I²S communication I didn't manage to make the USB port operational.

To build the program: Download the project folder and open it in the STM32Cube. The board I used was soldered and this is imperative: The PCM5102A chip is very sensitive to supply voltage fluctuations so breadboards should be avoided in order to minimize bad contact. All pins are described in the code.

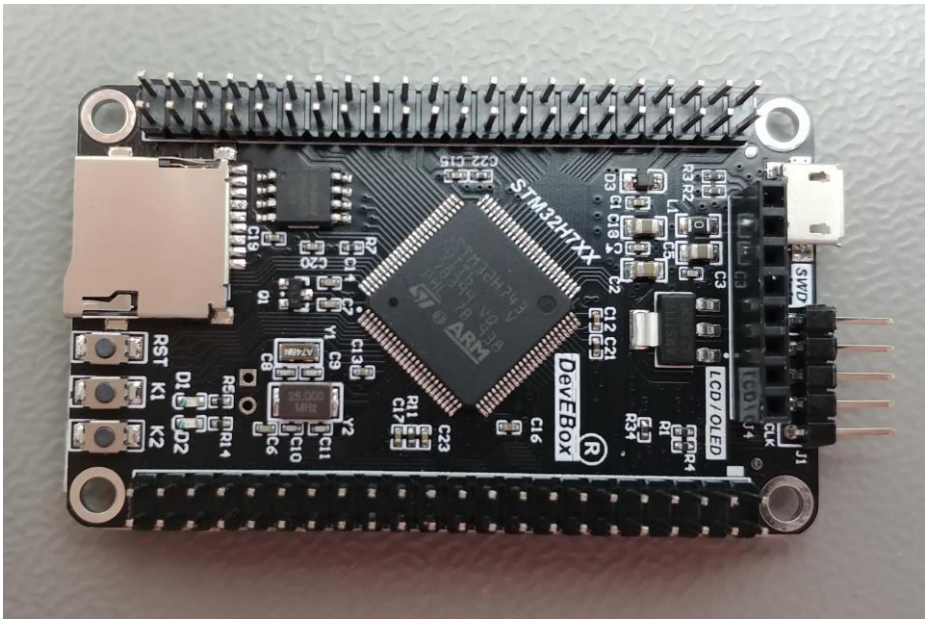
The system was debugged and tested using the LCD and an oscilloscope. Hardware testing was done using a multimeter, the oscilloscope and LED bars. The debouncing circuit is as follows:



The PCM connections are as follows:



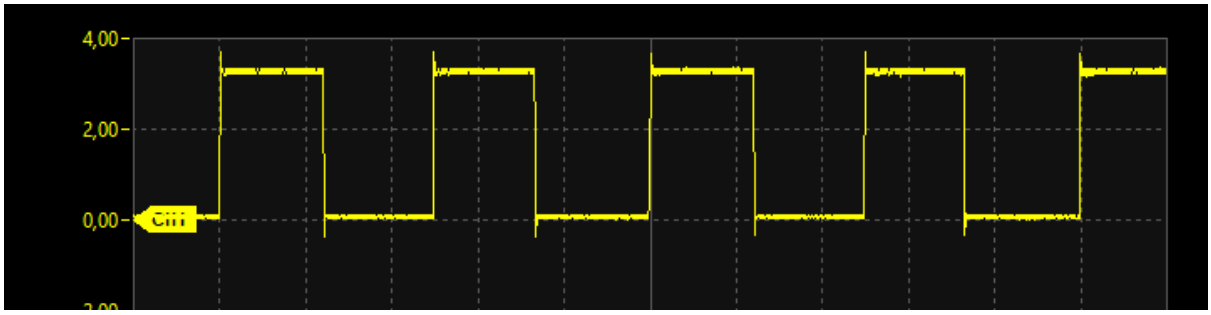
The microcontroller board used:



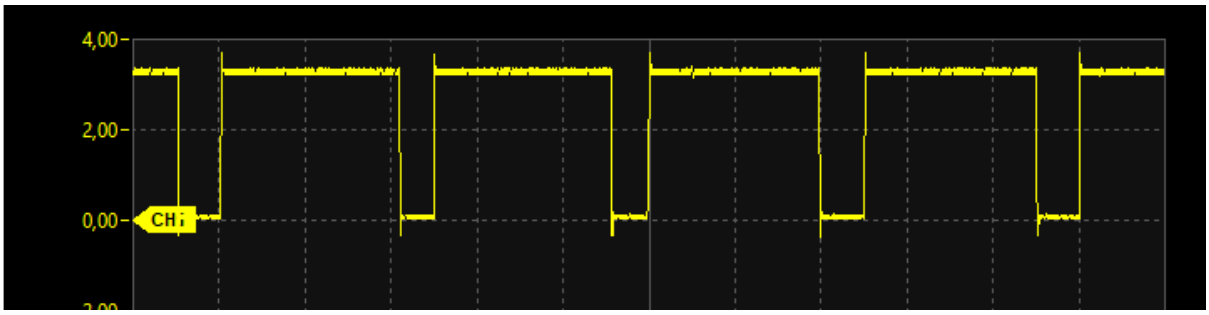
For future iterations, it I would like to implement the serial communication and also implement the DMA writing for the I²S in order to increase the sampling rate.

This was my first time programming this MCU and using the STM32Cube hence all the trouble trying to do simple tasks. Although I didn't manage to complete the project in this platform, the project itself will carry all the lessons and assets to my Bachelor's work, the final version of the synth, that uses the ESP-IDF and an ATMega2560 for the UI.

Profiling: The duration of the interrupt function is of vital importance for this project. If it locks the core to itself, the main loop won't run. Here's the interrupt duration with only two voices active:



It uses approximately 25µs of the 50µs period, ~15µs of those only to the I²S. Now with all 6 voices active the interrupt gets almost all of the core processing time:



The period on for the interrupt is now approximately 40µs, of the 50µs available.

Grading:

I only implemented a state machine for the UI so I consider this a very basic one plus there are missing aspects of the original project (+2 + 2). What was implemented in code is working perfectly (+ 2). No code was used from other sources (+ 1). I think this project is based on a really common idea but I hope to have gone beyond the basics of synth concepts (+ 2). Bonus: There is version control and system profiling (+ 3?).