BILKENT UNIVERSITY

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

EEE 491
Electrical and Electronics Engineering Design I

Spoken Number Recognition Project

FINAL LAB REPORT

Prepared by:

Alkım Ege Akarsu

Halil Duyan

Ibrahim Koray Ilgaz

11.01.2023

TABLE OF CONTENTS

1 PROJECT OBJECTIVE AND SPECIFICATIONS

The objective of the project is to design and implement a spoken number recognition system. The system works by comparing the mel-frequency cepstrum coefficients of a sample recording to a database of reference coefficients and choosing the most similar one.

Specifications of applicable blocks are as follows: (block diagram can be found in section 2)

1.1 PCB Block

- Single supply operation.
- Output swings between 0V - 3.3V. (silence corresponds to 1.65V output)
- Operational amplifiers used as active components.
- Electret as microphone type.
- 2 stages: amplification and anti-aliasing filter (LPF).
- Anti-aliasing stage uses two cascaded low-pass filters with Sallen-Key topology.
- Gain from 100 Hz to 3 kHz in the -1 to 1 dB range.
- Upper corner frequency between 4 kHz and 5 kHz.
- 80 dB/decade attenuation after upper corner frequency.
- Adjustable gain and silence DC output level.
- Everything to be implemented on a PCB with a maximum area of 25cm^2.

1.2 ADC Block

- A Serial Peripheral Interface (SPI) to an analog to digital converter (ADC) board.
- ADC board input voltage values change between 0 and 3.3V.
- The SPI interface operates at 12.5MHz clock rate.
- A Decimation Module was implemented to decimate the number of samples.
- Optionally Decimator takes the average of the 32 samples or ignores the first 31 samples.
- ADC sample rate is at 500K samples per second before decimation.
- After decimation ADC produces 15625 samples per second.
- In order to fill 16K Dual Port Ram ADC operates for 1.048576 seconds.
- Dual-port RAM is included with size 16384 x 32 bits.
- In order to make the module compatible with debug RAM was created with 32 bit width.
- An optional Sound detector is added to align the speech and recording time.

1.3 CONTROL Block

- Controls the overall system
- Reads ready signals from each block
- Sends start signals to each block in response to ready signals
- Updates frame address signal of WINDOW Block
- Creates a loop which executes WINDOW, FFT, MEL Blocks 64 times

1.4 WINDOW Block

- WINDOW Block reads 512 12-bit data from the ADC Block's RAM.
- Has a ROM filled with 512 8-bit data, Hanning coefficients
- Multiplies data from ADC Block's RAM with Hanning coefficients
- Produces overlapped frames(0-512, 256-768 etc)

1.5 FFT Block

- FFT Block reads 512 20-bit data samples from the Lab-WINDOW's dual-port RAM.
- FFT IP core executes the 512 point FFT function.
- FFT IP core produces 512 output samples with 64 bits.
- First 256 outputs of the IP core are fed to the multipliers to take the magnitude.
- Results of the multipliers are summed and written in 256 x 32 bit Dual Port RAM.

1.6 MEL Block

- One frame as input from the FFT block. (256 * 32 bit)
- 16 triangular, overlapping, mel scale filter banks on data from 96 Hz to 5440 Hz.
- 16 energy levels as output to the debug block. (16 * 32 bit)

1.7 DEBUG Block

- Uses universal asynchronous receiver-transmitter (UART) protocol.
- Baud rate: 115200.
- Transfer energy levels from MEL block to PC in 8-bit chunks.

1.8 DCT Block

- Input from BASYS 3, 64 frames of data.
- Output as a 16x64 matrix, output is DCT of input data without start and stop words.

1.9 COMPARISON Block

- Input is a 16x64 matrix from the DCT block.
- Output is the most likely number that is spoken in the sample recording.

2 SYSTEM DESCRIPTION

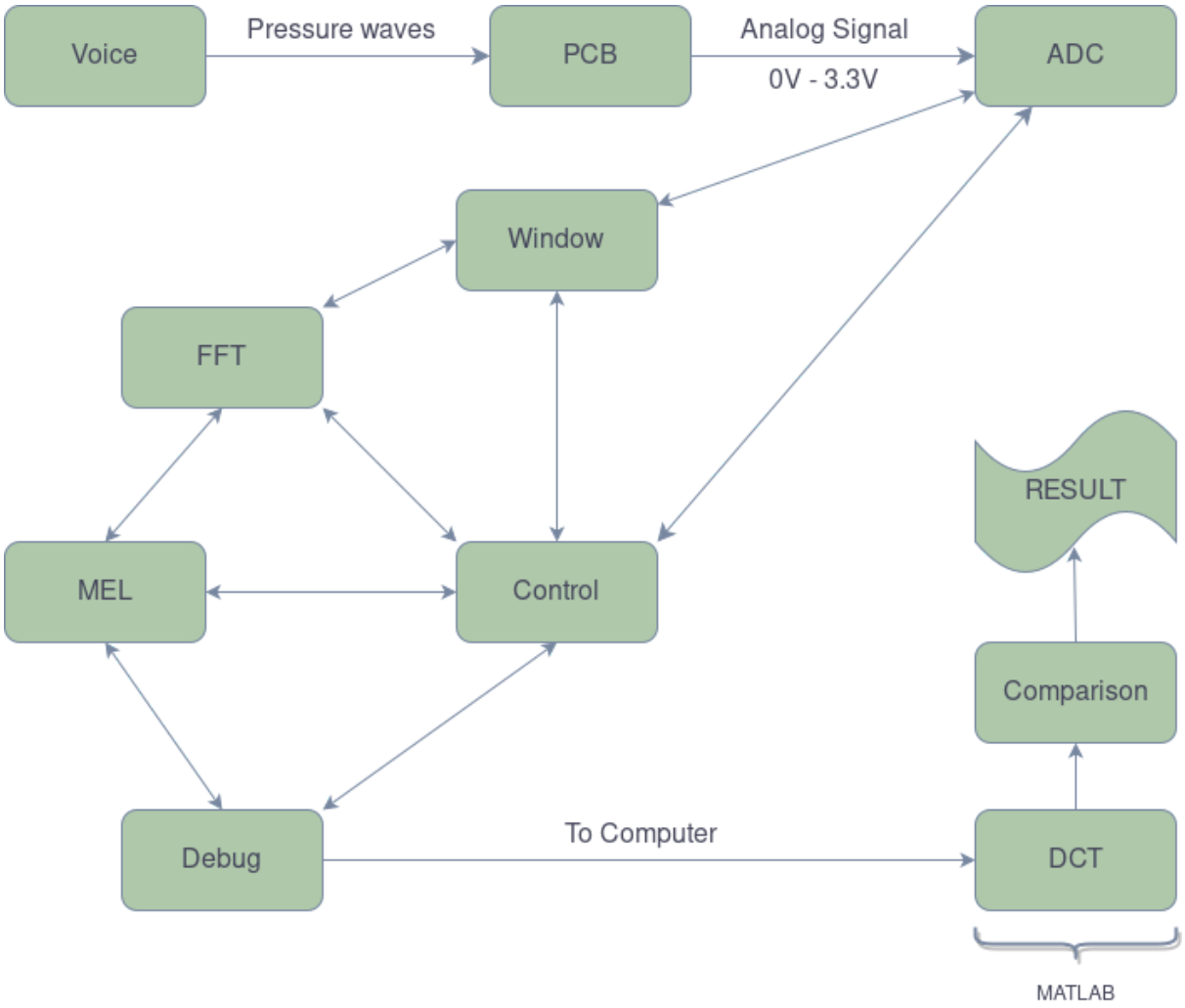

Figure 1: Block diagram of the overall system.

The overall operation of the system can be described using the block diagram in figure 1. First, the one second voice sample of the speaker is converted to an analog signal using an electret microphone and some analog circuitry. Second, the analog signal is digitized by an analog to digital converter. Then the digital signal is framed into many parts and a windowing function is used before the FFT operation. Following the windowing procedure, the FFT function is used to get the frequency spectrum of the frame. Then, the output of the FFT is passed through a mel-scale filter bank for energy extraction. After transferring the output of the filter bank from the BASYS 3 to the PC running MATLAB using the debug block, the discrete cosine transform of the energies are calculated. Finally, the obtained cepstral coefficients are compared to reference coefficients by their distance in euclidean space and the closest reference recording is selected and displayed.

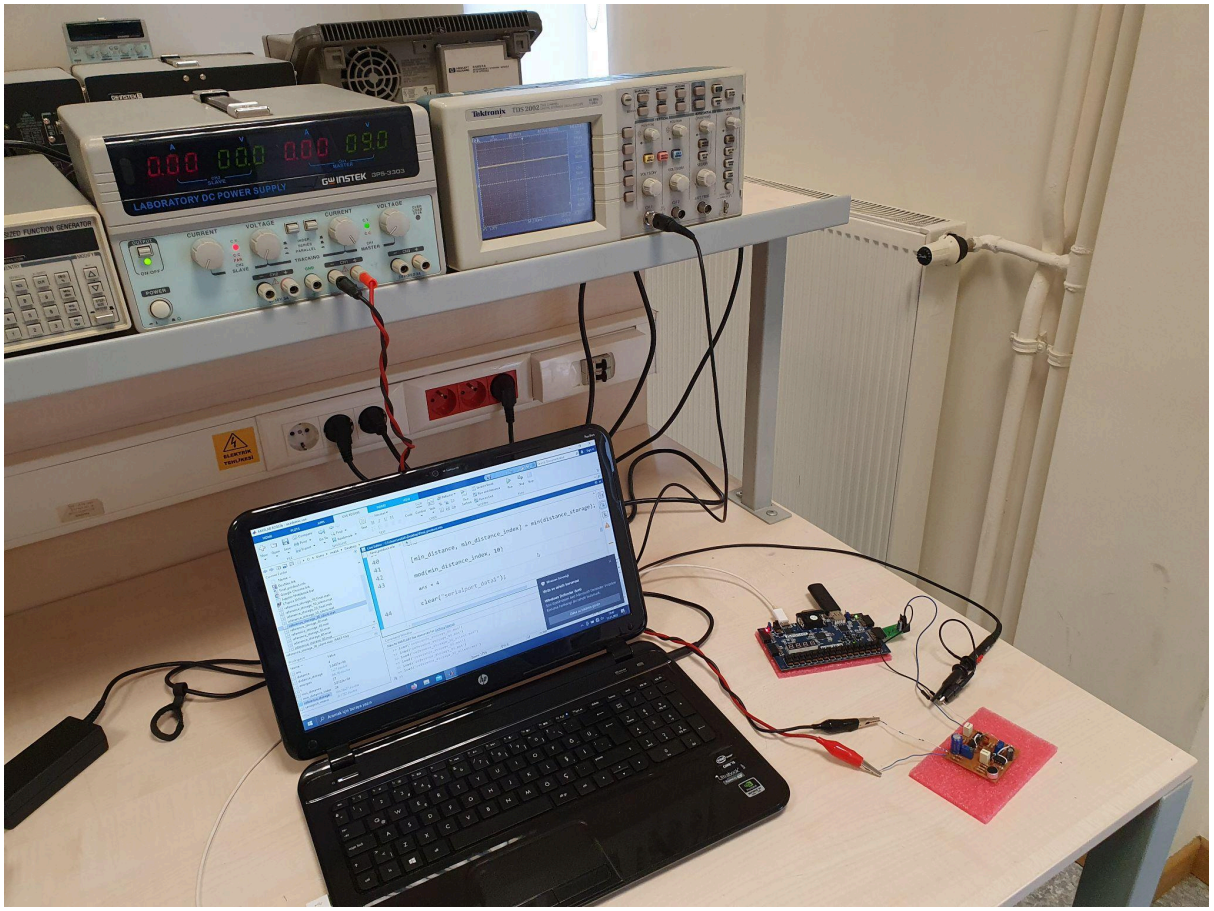

Figure 2: Photograph of the whole system.

2.1 PCB Block

This block is responsible for transforming the pressure waves resulting from human speech into analog signals that go into the ADC block. For the system to function properly, the PCB block should not change the nature of the signal when performing its function. For this reason, this block should introduce the least possible amount of noise and distortion to the signal in the frequency range of interest which is 96 Hz to 5440 Hz.

2.2 ADC Block

The ADC block is responsible for converting the analog output of the PCB into a digital signal. The voltage limits for the input signal of the ADC block are 0V and 3.3V. Therefore, for a sinusoidal function to be detected properly it needs to be biased. Offset is determined as 1.65V. In the ADC block obtained data is also decimated by 32 to produce 16384 data samples per record. It is also possible to control the time of the recording in this block with the sound detector.

2.3 CONTROL Block

This block can be called the main block of the entire project, as it controls the whole system with its inputs and outputs. It executes several commands in sequence:

- When the ready signal from the ADC block is received, the control block sends a start signal to the window block.

- When the ready signal from the window block is received, the control block sends a start signal to the FFT block. In the first loop, the framing address of the window block starts at 0. After the start signal of the FFT block is sent, the framing address of the window block is increased by 256.
- When the ready signal from the FFT block is received, the control block sends a start signal to the MEL block.
- When the ready signal from the MEL block is received, the control block sends a start signal to the window block. If this loop has been completed 64 times, the start signal is given to the debug block.

If the implementations of the DCT block and compare block had been done on the BASYS3, the control block would make the loop go back to the window block after the DCT block. In the current configuration, the compare and the debug blocks are executed only once.

2.4 WINDOW Block

This block is responsible for framing the data stored in the ADC block's RAM. The address signal is taken from the control block. By incrementing that address signal and sending it to the ADC block's RAM, 512 samples are taken and multiplied with 512 Hanning Coefficients, which are stored in a ROM inside the window block. The Control Block then increases the address signal by 256, so the frames are overlapped. In the first loop, data is taken from the addresses 0 to 512. In the second loop, data is taken from the addresses 256 to 768, etc.

2.5 FFT Block

The FFT block is responsible for taking the Fast Fourier Transform of the frames created by the window block. To operate, the FFT block requires the whole sequence and then it starts to compute the FFT of the provided frame. Output of the FFT IP is 64 bits long containing both real and imaginary parts. These parts are squared and summed to obtain the magnitude. 32 bits of the resulting numbers were selected from the serially provided outputs and written into the dual port ram to be read by the MEL block

2.6 MEL Block

This block is responsible for applying the mel-scale filter bank to the data received from the FFT block. Passing the frequency data from the mel-scale filter bank results in the amount of energy contained in the frequency ranges of each of the banks. The resulting energies are normalized by the length of the filter. Finally, the common logarithm of the energies are calculated and multiplied by 10, resulting in the audio power of each filter bank. The output of this block results in N coefficients where N is the number of mel-scale filter banks.

2.7 DEBUG Block

This block is responsible for transferring the energy data obtained from the MEL block from the BASYS 3 development card to the PC running MATLAB.

2.8 DCT Block

This block evaluates the discrete cosine transform of the output of the MEL block using the "*dct*" function of MATLAB. The generation of cepstral coefficients is now complete.

2.9 COMPARISON Block

This block compares the cepstral coefficients of the sample recording to the cepstral coefficients of the reference recordings. This comparison is achieved by calculating the euclidean distance which is done with the MATLAB command "*norm*". The reference recording with the closest cepstral coefficients is chosen and displayed as the result.

3 IMPLEMENTATION AND TESTS

3.1 PCB Block

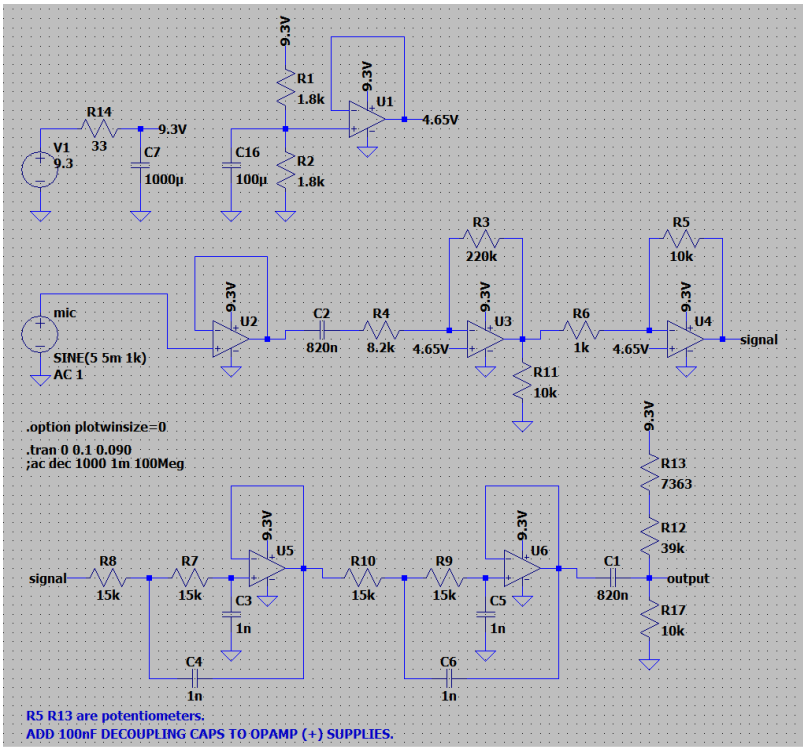

Figure 3: LTSpice [1] schematic of PCB design. The first row is the power delivery, the second row supplies the gain, and the third row is the anti-aliasing filter.

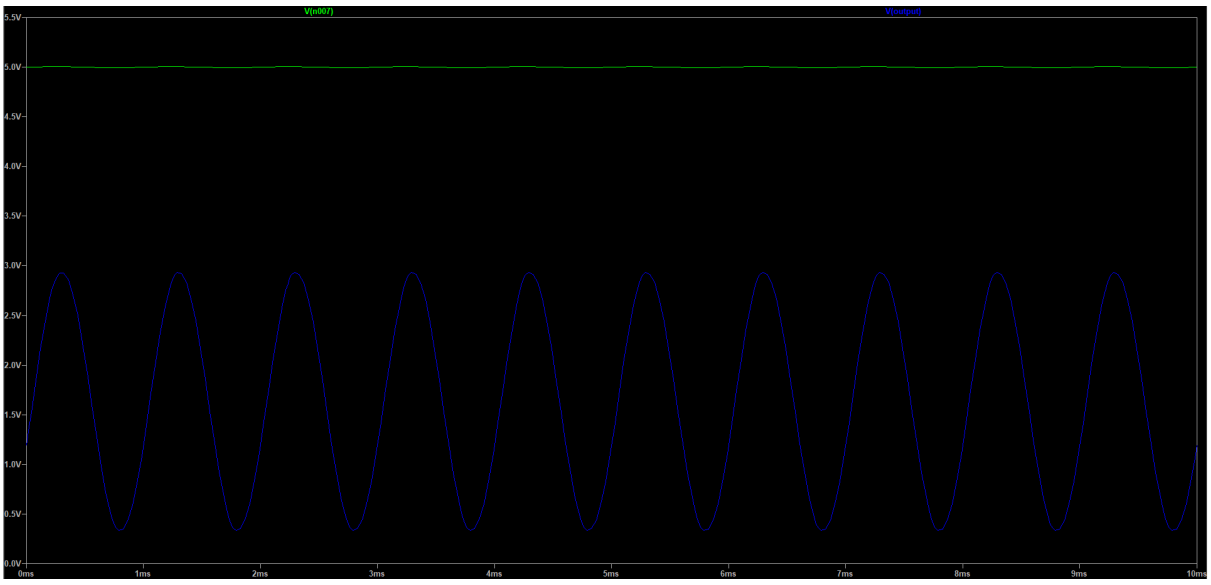

Figure 4: LTSpice transient simulation input (green, microphone) and output (blue) waveforms.

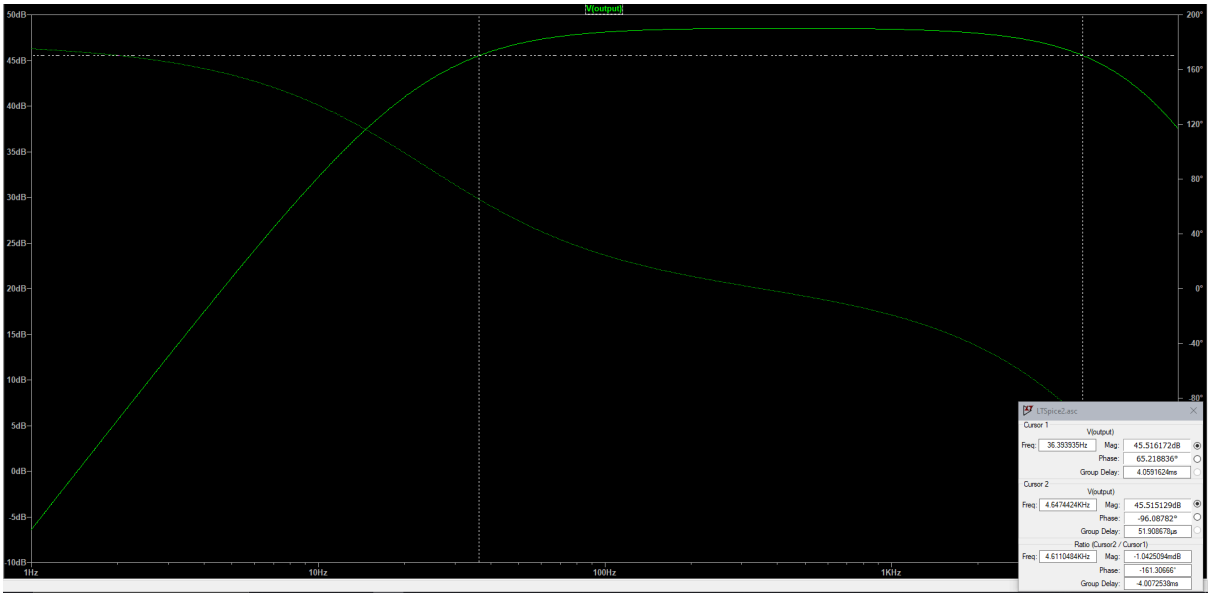

Figure 5: LTSpice AC analysis lower (36 Hz) and upper (4647 Hz) -3dB cutoff frequencies.

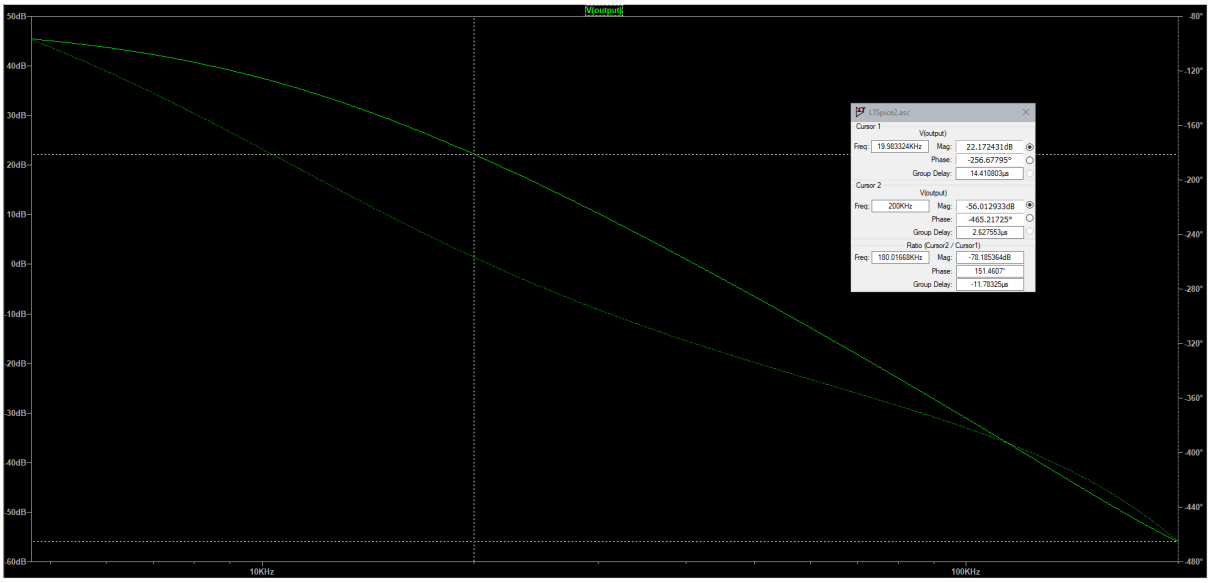

Figure 6: LTSpice AC analysis -80 dB/decade attenuation.

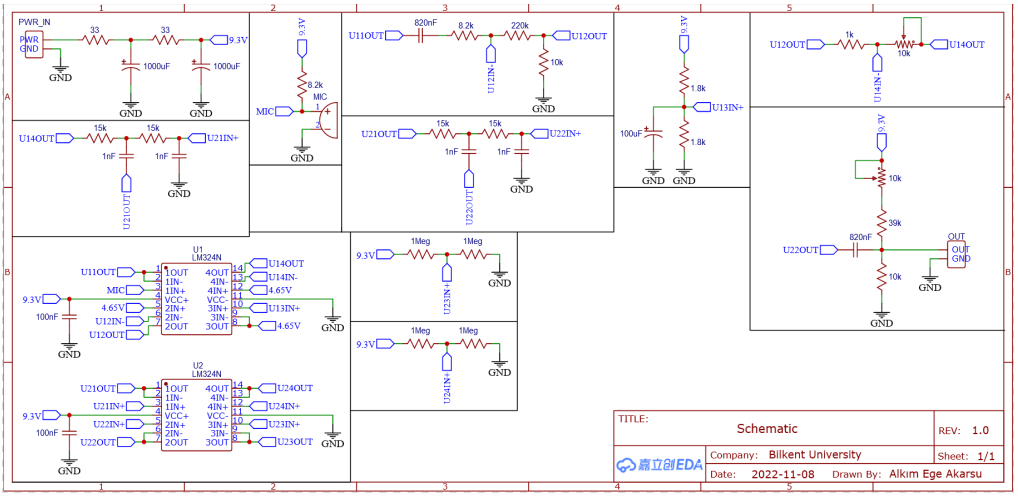

Figure 7: EasyEDA [2] circuit schematic.

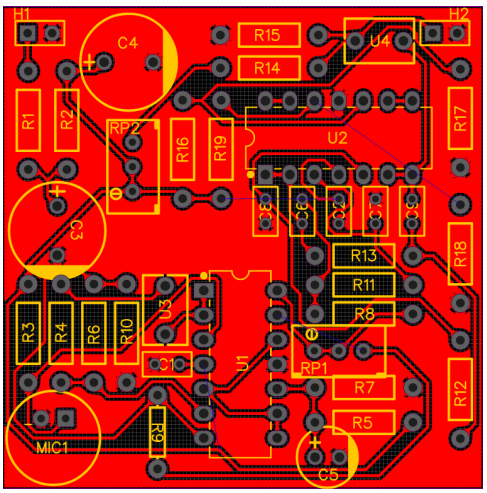

Figure 8: EasyEDA PCB design.

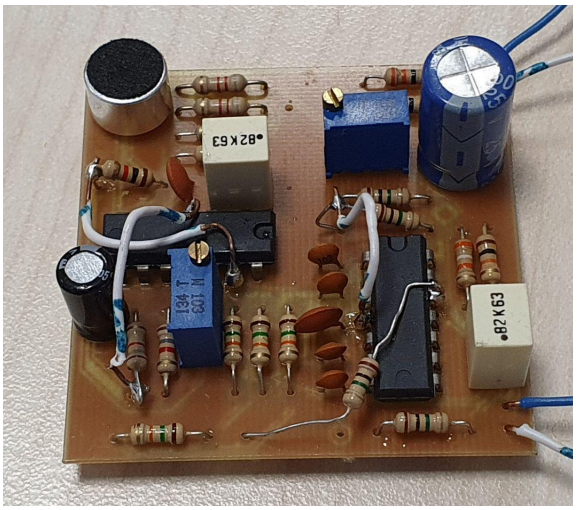

Figure 9: Picture of printed and populated PCB.

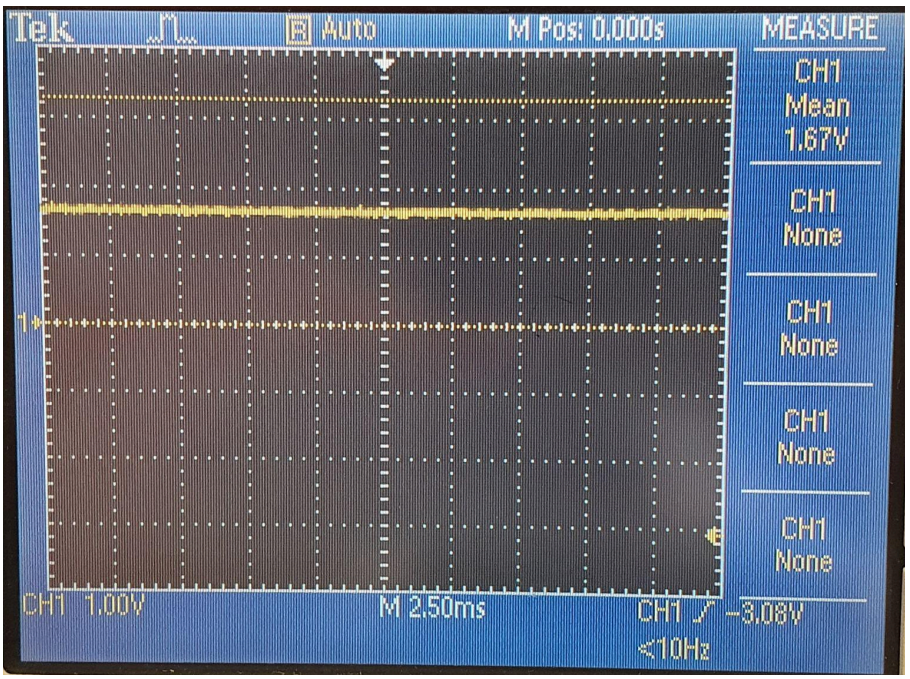

Figure 10: PCB output with no input.

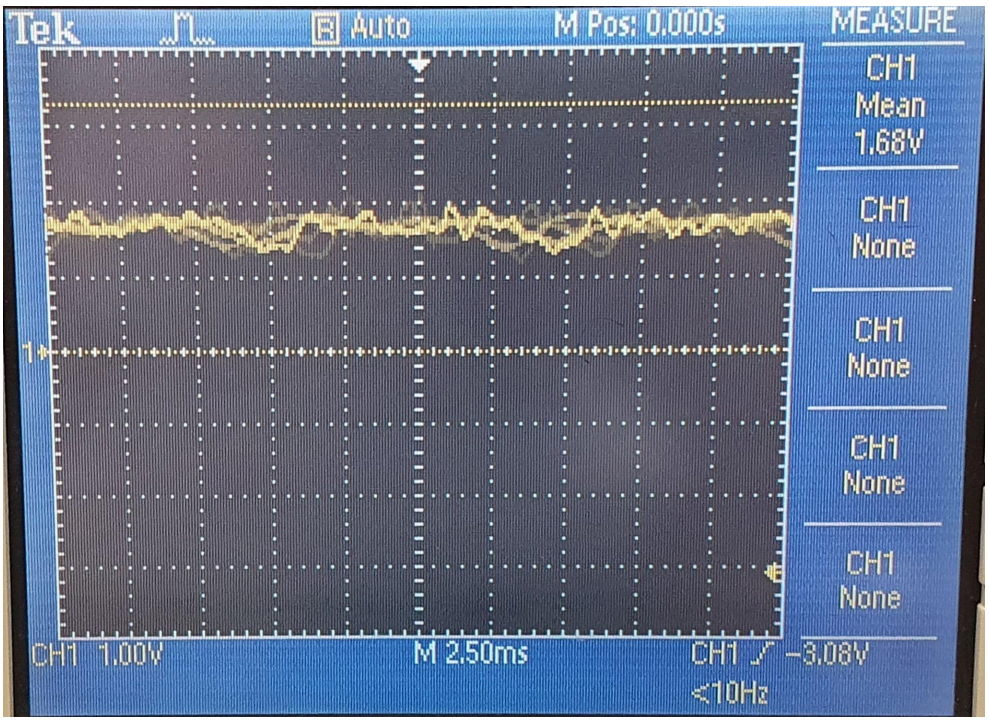

Figure 11: Example PCB output while talking.

3.2 ADC Block

For the ADC block to operate as expected there are several timing requirements that need to be satisfied. Since the ADC provides output data via a serial connection, the data needs to be read in time. The mode of the ADC is also controlled with the timing of the signals. There are two signals that are provided to the ADC module [3] from the FPGA board: the spi clock signal (spi_clk_out) and the chip select signal (chip_sel_out). The signals are aligned in order to keep the ADC module in the operating mode and read the data from the spi data output. To achieve this task, two clock counters were utilized and the smallest common period was set as the clock counter limit. This limit was found by 25 periods of the spi_clk_out signal since chip_sel_out needs to go down and up in determined rising edges. According to the specifications, one spi period is eight times of i_clock. Thus, establishing a clock counter explained as above, it became possible to adjust the falling and rising edge of the chip_sel_out signal without depending on the rising or falling edges of the spi signal. Finally, chip_sel_out and spi_clk_signals were aligned according to the datasheet provided in the references.

After reading the data serially to the FPGA board, it was decimated by 32 and written into the dual port ram. 12 bits of this data are the actual bits that are read from the spi input and the others are zero in order to make the ram compatible with the debug block. The signal to noise ratio of the resulting images were found above 45 consistently. The figures belonging to the tests of the lab ADC can be seen below.

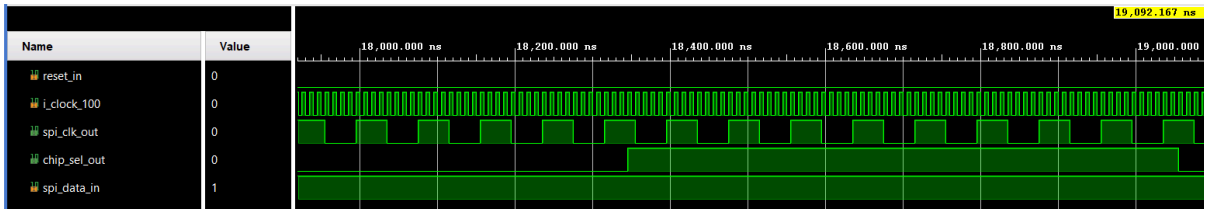

Figure 12: spi_clock and chip_sel_out signals of the ADC block.

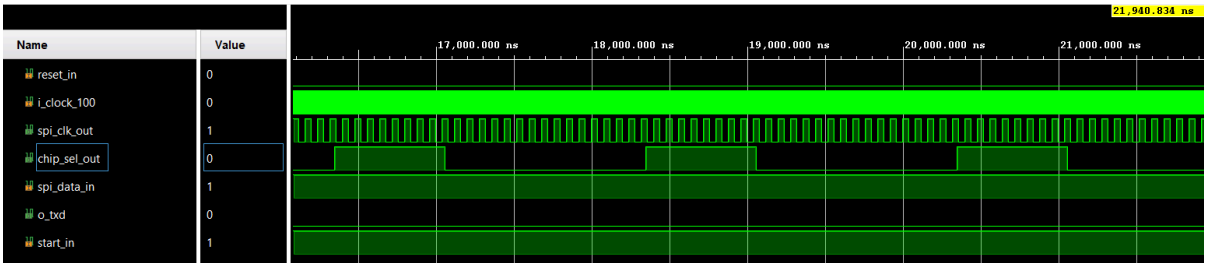

Figure 13: Timing diagram of the ADC block.

3.3 WINDOW Block

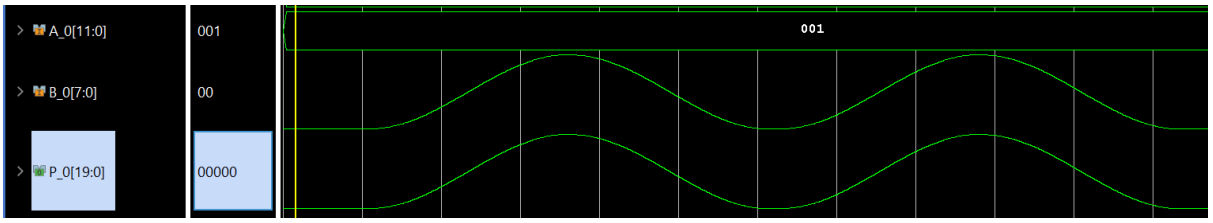

Figure 14: Multiplication result of the Window Block, where A_0 is the data from ADC Block, and B_0 is Hanning Coefficients.

3.4 FFT Block

In the FFT block the main task is to operate the given FFT IP and obtain the FFT of the provided frame. Required information related to the IP block was given in both the assignment and the datasheet provided in the references [4]. In short, the FFT IP requires three groups of signals which are configuration, s_axis_data and m_axis_data. The FFT IP flags the starting event and last word of the output which enables us to utilize these signals when loading and reading the data. In this block we slowed down the clock of the FFT IP by 10 in order to create time for the data manipulation that will be made to the received data words.

By reading the data sheet and trial-and-error, the behavior of the FFT IP was inspected and timing of the IP was understood according to the configuration in our assignment. Then, data was loaded to the IP block. When loading, all 512 words of the frame are given to the IP, however, only the first 256 words of the output were received and the others were ignored since the FFT produces a symmetric output. The output words of the FFT are split into real and imaginary parts. Then, they are squared and summed to obtain the magnitude of the word. Calculated magnitudes were written into the 256 by 32 bit dual port RAM for the MEL block. At the end of every sequence the FFT block resets itself to ensure the successful operation of the FFT IP. Resulting timing diagram and output can be seen below.

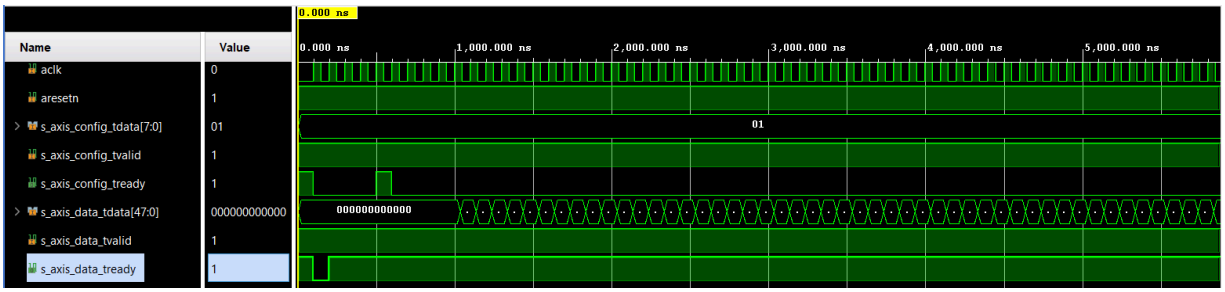

Figure 15: Timing Diagram of the FFT block

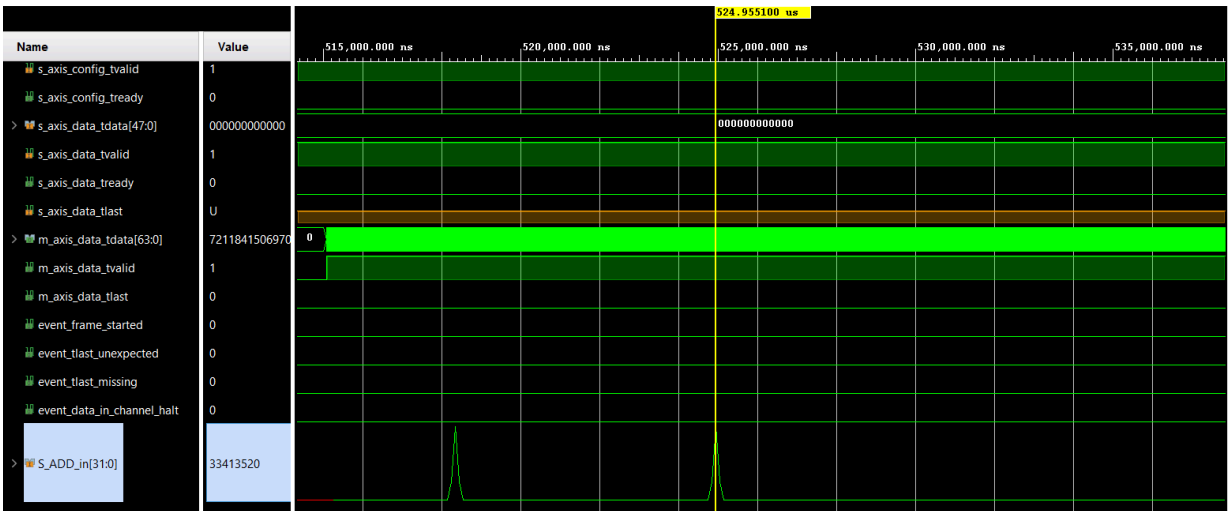

Figure 16: Output of the FFT block

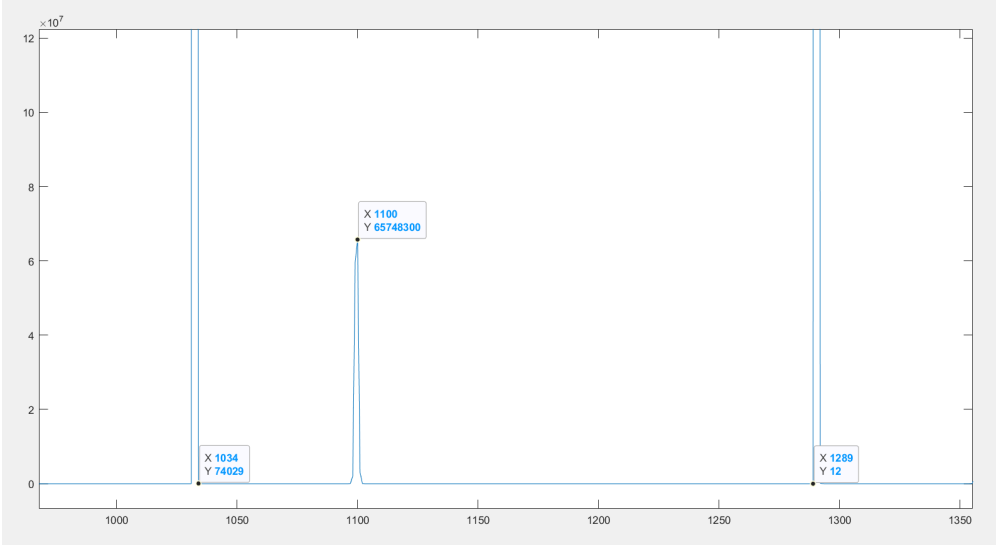

Figure 17: Output of the FFT block at 2KHz

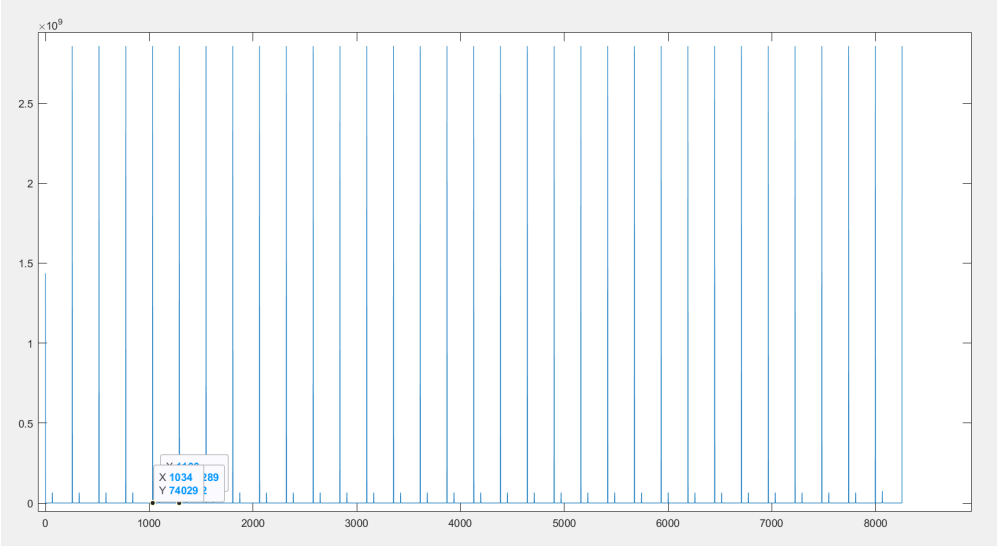

Figure 18: Output of the FFT block at 2KHz

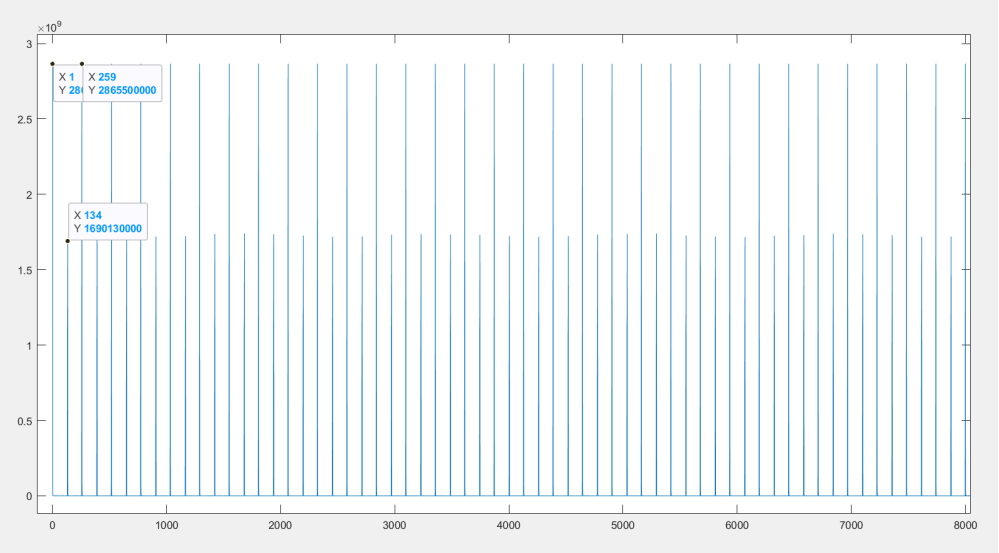

Figure 19: Output of the FFT block at 4KHz

3.5 MEL Block

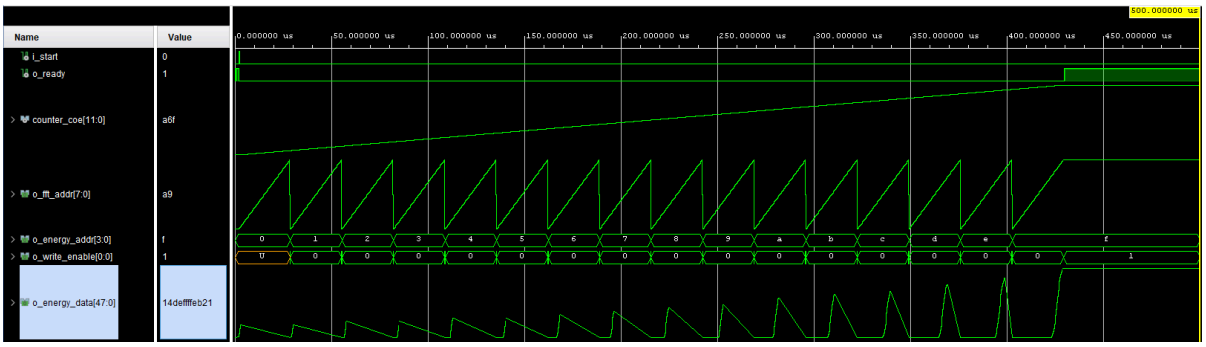

Figure 20: Vivado [5] behavioral simulation of MEL block. Every time o_write _enable becomes high, the data in o_energy_data gets written to the output ram at the address o_energy_addr. The data coming from the FFT block is assumed to be FFFF FFFF for this simulation. Each filter bank getting larger can be observed in the output.

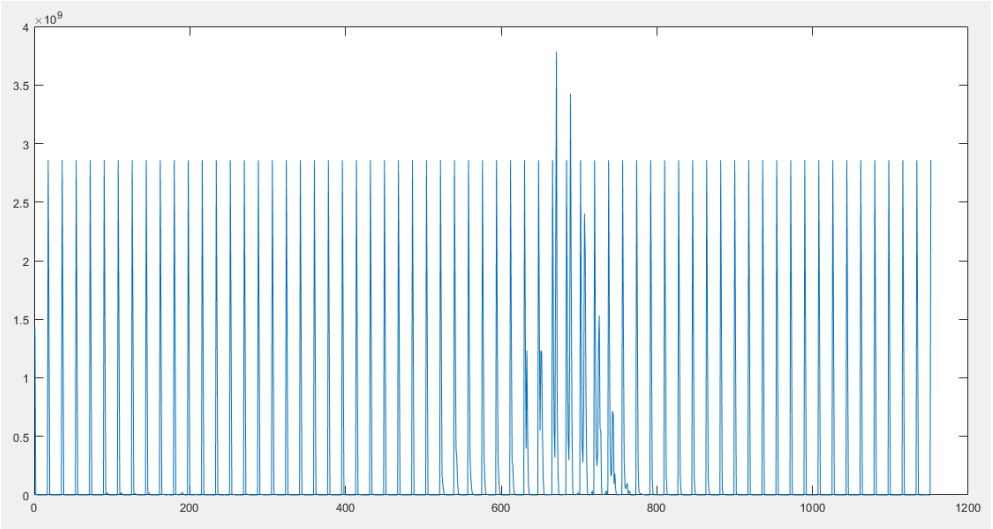

Figure 21: Plot of 64 frames of energy data transferred from BASYS 3 to MATLAB.

3.6 DEBUG Block

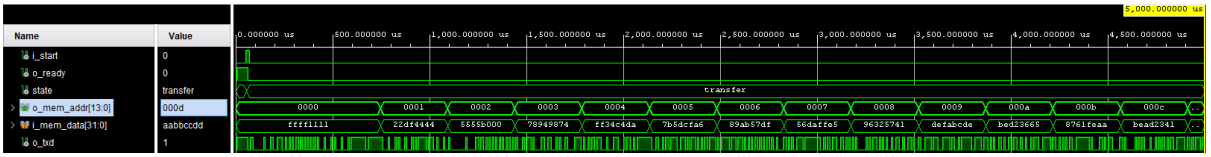

Figure 22: Vivado behavioral simulation of DEBUG block. After i_start becomes high, the transfer process gets started. Memory address is sent to the related RAM via o_mem_addr and related data is received via i_mem_data. Finally, using the o_txd pin, the data is sent in 10 bit chunks, the first bit being the start bit (low) and the last bit being the stop bit (high).

3.7 DCT Block

The built-in MATLAB function “dct” is applied to all 64 frames that are received from the BASYS 3 board. This block is implemented in MATLAB so there is no simulation to be done.

3.8 COMPARISON Block

The cepstrum coefficients of the sample from the DCT block are compared with the cepstrum coefficients of reference recordings. This is accomplished using the built-in MATLAB function “norm”. This block is implemented in MATLAB so there is no simulation to be done.

4 DISCUSSION AND CONCLUSION

Many problems were encountered and solved during the implementation project. First, while all the group members had previous experience with the software used in this project, some time and practice was needed to refresh the necessary skills for the project.

In the PCB block, the specific operational amplifier used (LM324N) had some non-ideal characteristics (crossover distortion) that was solved by putting a shunt resistor to the output [6].

In the ADC block, the main problem was the timing of the signals and insufficient information on the timing of the reading instance along the clock signal. This problem was solved with the help of the instructor of the course and perfected by trial-and-error.

In the FFT block, several problems were encountered. The first one was the simulations with the IP core. It turned out that the simulation in Vivado may fail to update the changes in the code when simulation was done with an FFT IP. This problem was solved by disabling the incremental compilation in the simulation settings. The Second problem that was faced during the development of the FFT block was the behavior of the IP core. It was observed that the timing of the flag signals such as event_started or s_axis_data_tready can vary according to the timing of the start_in signal in the simulation. To solve this problem code was written in a way to be independent of the duration of the signals. The third problem that was experienced with the ip block was the speed of the clock. It was not possible to operate the IP core with 100 MHz while multiplying and summing the words that are serially provided from the ip core. Thus, the problem was solved by slowing down the clock of the FFT IP core. Last problem of the ip core was the reset signal. After coding the block, it was not possible to operate the FFT block for some time since the IP was insensitive to the inputs. While checking the configuration of the IP core, it was seen that the reset signal was active low by default. After changing the settings of the signal the problem was solved and reset signal was utilized to refresh the ip core for every sequence of frames. The system can be improved by the implementation of a more sophisticated speech recognition algorithm.

5 REFERENCES

[1] *LTSpice* XVII (2022), Analog Devices. [Online]. Available: https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html

[2] *EasyEDA* (2022), EasyEDA. [Online]. Available: https://easyeda.com

[3] *ADC121S101x Single-Channel, 0.5 to 1-Msps, 12-Bit Analog-to-Digital Converter*. https://www.ti.com/. (n.d.). Retrieved January 11, 2023, from https://www.ti.com/lit/ds/symlink/adc121s101.pdf

[4] *Fast Fourier Transform v9.1 LogiCORE IP Product Guide*. (2022, May 4). Retrieved January 11, 2023, from https://www.xilinx.com/content/dam/xilinx/support/documents/ip_documentation/xfft/v9_1/pg109-xfft.pdf

[5] *Vivado* (2022.1), Xilinx. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html

[6] "Adding a resistor to reduce crossover distortion in an LM324/LM358," *Electronics Stack Exchange*, Nov. 26, 2017. https://electronics.stackexchange.com/questions/341843/adding-a-resistor-to-reduce-crossover-distortion-in-an-lm324-lm358 (accessed Jan. 11, 2023).

6 APPENDIX

6.1 VHDL CODE

6.1.1 ADC Block

```
library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;

entity lab_adc_main is
  Port ( reset_in_adc: IN STD_LOGIC;
         start_in_adc: IN STD_LOGIC;
         i_clock_100: IN STD_LOGIC;
         spi_data_in: IN STD_LOGIC;

         spi_clk_out: OUT STD_LOGIC;
         chip_sel_out: OUT STD_LOGIC;
         ready_outadc: OUT STD_LOGIC;


         ram_addr : OUT STD_LOGIC_VECTOR (13 DOWNTO 0);
         ram_data : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)

         );

end lab_adc_main;

architecture Behavioral of lab_adc_main is

signal spi_clock_counter : integer range 0 to 4 := 0;
signal cs_clock_counter : integer range 0 to 199 := 0; -- 25 spi clock cycles
signal data_clock_counter : integer range 0 to 31 := 0;
signal data_counter : integer range 0 to 16384 := 0; -- 16385 in total because first sample is a dummy
sample and it will be discarded in the data transfer

signal spi_clock  : std_logic := '0'; -- 125 MHz
signal data_clock : std_logic := '0'; -- 15625 samples per second
signal sound_detector : std_logic := '0';
signal cs_duplicate : std_logic := '0';
signal start_signal : std_logic := '0';
signal start_ins : std_logic := '0';
signal reset_in_adcs : std_logic := '0';
signal ready_outadcs : std_logic := '1';


signal data_holder : std_logic_vector (31 downto 0) := x"00000000"; -- fsm counter ile çalýþacak,
serial signalý 12 bit dataya dönüþtürecek
```

```
signal data_keeper : std_logic_vector (31 downto 0) := x"00000000"; -- data holder her convertionda deðiþecek, keeper her 32de bir deðiþecek ve rame baðlanacak
signal ram_a_addr  : std_logic_vector (13 downto 0)  := "00000000000000"; -- ram writing addres


begin

spi_clk_out <= spi_clock;
ram_addr <= ram_a_addr;
ram_data <= data_keeper;

reset_in_adcs <= reset_in_adc;
start_ins <= start_in_adc;
ready_outadc <= ready_outadcs;


starter: process(i_clock_100)
begin
   if rising_edge(i_clock_100) then
      if data_counter = 0 then
         if start_ins = '1' and reset_in_adcs = '0' then
          ready_outadcs <= '0';
          start_signal  <= '1';

         elsif reset_in_adcs = '1' then
          ready_outadcs <= '1';
          start_signal  <= '0';

         end if;

      elsif data_counter = 16384 or reset_in_adcs = '1' then
         ready_outadcs <= '1';
         start_signal  <= '0';
      else

      end if;
   end if;


end process starter;


spiclk_signal: process(i_clock_100)
begin
   if rising_edge(i_clock_100) then
      if spi_clock_counter = 3 then
         spi_clock_counter <= 0;
       spi_clock <= not spi_clock;
```

```
      else
     spi_clock_counter <= spi_clock_counter + 1;
     end if;
  end if;
end process spiclk_signal;



dataclk_signal: process(cs_duplicate)
begin
   if rising_edge(cs_duplicate) then
      if data_clock_counter = 15 then
         data_clock_counter <= 0;
         data_clock <= not data_clock;
      else
         data_clock_counter <= data_clock_counter + 1;
      end if;
   end if;
end process dataclk_signal;

CS_signal: process(i_clock_100)
begin
   if rising_edge(i_clock_100) then
      if cs_clock_counter = 5 then
         chip_sel_out <= '0';
         cs_duplicate <= '0';
         cs_clock_counter <= cs_clock_counter + 1;
      elsif
          cs_clock_counter = 134 then
          chip_sel_out <= '1';
          cs_duplicate <= '1';
          cs_clock_counter <= cs_clock_counter + 1;
      elsif
          cs_clock_counter = 199 then
          cs_clock_counter <= 0;
      else
          cs_clock_counter <= cs_clock_counter + 1;

      end if;
   end if;
end process CS_signal;

data_hold_signal : process(i_clock_100) -- holds every converted data from ADC, in data transfer one
in every 32 will be selected and given to the ram
begin
   if rising_edge(i_clock_100) then
      if start_signal = '1' then
```

```
        if cs_clock_counter = 36  then
        data_holder(11) <= spi_data_in;
        elsif cs_clock_counter = 44  then
        data_holder(10) <= spi_data_in;
        elsif cs_clock_counter = 52  then
        data_holder(9) <= spi_data_in;
        elsif cs_clock_counter = 60  then
        data_holder(8) <= spi_data_in;
        elsif cs_clock_counter = 68  then
        data_holder(7) <= spi_data_in;
        elsif cs_clock_counter = 76  then
        data_holder(6) <= spi_data_in;
        elsif cs_clock_counter = 84  then
        data_holder(5) <= spi_data_in;
        elsif cs_clock_counter = 92  then
        data_holder(4) <= spi_data_in;
        elsif cs_clock_counter = 100  then
        data_holder(3) <= spi_data_in;
        elsif cs_clock_counter = 108  then
        data_holder(2) <= spi_data_in;
        elsif cs_clock_counter = 116  then
        data_holder(1) <= spi_data_in;
        elsif cs_clock_counter = 124  then
        data_holder(0) <= spi_data_in;
        end if;
        end if;
    end if;
end process data_hold_signal;

--sound_detector_signal : process(cs_duplicate) -- A THRESHOLD MAY BE NEEDED
####
--begin
--    if rising_edge (cs_duplicate) then
--       if data_holder < "011111110000" OR data_holder > "10000010000" then
--       sound_detector <= '1';
--       end if;

--end if;
--end process sound_detector_signal;

data_transfer : process(data_clock)
begin
if start_signal  = '1' then
   -- if sound_detector = '1' then
        if rising_edge (data_clock) then
            if data_counter = 1 then
                                        ram_a_addr <= "00000000000000";
                                        data_counter <= data_counter + 1;
```

```
                        elsif data_counter = 16384 then
                                ram_a_addr <= "00000000000000";
                           data_counter <= 0;
                        else
                                ram_a_addr <= ram_a_addr + 1; -- address update
                                data_keeper <= data_holder - 2048  ; -- data update -- data became 2s
compliment
                                data_counter <= data_counter + 1;
          end if;
       end if;
  -- end if;
end if;

end process data_transfer;


end Behavioral;
```

6.1.2 CONTROL Block

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
USE IEEE.NUMERIC_STD.ALL;

entity main is
   Port ( reset_in : in STD_LOGIC;
          clock_in : in STD_LOGIC;
          start_adc_out : out STD_LOGIC;
          ready_adc_in : in STD_LOGIC;
          frame_addr_out : out STD_LOGIC_VECTOR (13 downto 0);
          start_window_out : out STD_LOGIC;
          ready_window_in : in STD_LOGIC;
          start_fft_out : out STD_LOGIC;
          ready_fft_in : in STD_LOGIC;
          start_mel_out : out STD_LOGIC;
          ready_mel_in : in STD_LOGIC;
          start_dct_out : out STD_LOGIC;
          ready_dct_in : in STD_LOGIC;
          start_comp_out : out STD_LOGIC;
          ready_comp_in : in STD_LOGIC;
          start_debug_out : out STD_LOGIC;
          ready_debug_in : in STD_LOGIC;
          start_in : in STD_LOGIC;
          ready_out : out STD_LOGIC
          --spi_clk_out: OUT STD_LOGIC;
          --chip_sel_out: OUT STD_LOGIC;
```

```
        --spi_data_in: IN STD_LOGIC

        );
end main;

architecture Behavioral of main is

--signal loop_counter : integer range 0 to 63:=63;
signal loop_counter : UNSIGNED (5 downto 0) := "111111";
signal frame_addr_sig : STD_LOGIC_VECTOR(13 DOWNTO 0):="00000000000000";
signal start_adc_out_sig : std_logic;
signal start_window_out_sig : std_logic;
signal start_debug_out_sig : std_logic;
signal start_fft_out_sig : std_logic;
signal start_mel_out_sig : std_logic;
signal debug_start_counter : std_logic_vector(1 downto 0):="10";


type states is (idle, adc, window, fft, mel, dct, comp, debug); -- States of the FSM
signal state : states := idle; -- State signal signal


begin

frame_addr_out <= frame_addr_sig;
start_adc_out <= start_adc_out_sig;
start_window_out <= start_window_out_sig ;
start_debug_out <= start_debug_out_sig;
start_fft_out <= start_fft_out_sig;
start_mel_out <= start_mel_out_sig;

fsm: process(clock_in)
begin
if rising_edge(clock_in) then
   case state is
      when idle =>
         ready_out <= '1';
         -- Reset everything
         start_adc_out_sig <= '0';
         start_window_out_sig <= '0';
         start_fft_out_sig <= '0';
         start_mel_out_sig <= '0';
         start_dct_out <= '0';
         start_comp_out <= '0';
         start_debug_out_sig <= '0';
         loop_counter <= "111111";
         debug_start_counter <= "10";
         frame_addr_sig <= "00000000000000";
```

```
    if start_in = '1' then
      start_adc_out_sig <= '1';
      state <= adc;
    end if;
  when adc =>
    start_adc_out_sig <= '0';
    ready_out <= '0';
    if start_adc_out_sig = '0' then
      if ready_adc_in = '1' then
        state <= window;
        start_window_out_sig <= '1';
      end if;
    end if;
  when window =>
    start_window_out_sig <= '0';
    if start_window_out_sig = '0' then
      if ready_window_in = '1' then
        state <= fft;
        start_fft_out_sig <= '1';
        frame_addr_sig <= frame_addr_sig + 256;
      end if;
    end if;
  when fft =>
    start_fft_out_sig <= '0';
    if start_fft_out_sig = '0' then
      if ready_fft_in = '1' then
        state <= mel;
        start_mel_out_sig <= '1';
      end if;
    end if;
  when mel =>
    start_mel_out_sig <= '0';
    if start_mel_out_sig = '0' then
      if ready_mel_in = '1' then
        state <= dct;
        start_dct_out <= '1';
      end if;
    end if;
  when dct =>
    start_dct_out <= '0';
    if ready_dct_in = '1' then
      state <= comp;
      start_comp_out <= '1';
    end if;
  when comp =>
    start_comp_out <= '0';
    if ready_comp_in = '1' then
      state <= debug;
```

```
            start_debug_out_sig <= '1';

         end if;
      when debug =>
         start_debug_out_sig <= '0';
         if start_debug_out_sig = '0' then
            debug_start_counter <= debug_start_counter - 1;
            if debug_start_counter = "00" then
               if loop_counter = 0 and ready_debug_in = '1' then
                  state <= idle;
                  loop_counter <= "111111";
                  debug_start_counter <= "10";
               elsif loop_counter > 0 and ready_debug_in = '1' then
                  loop_counter <= loop_counter - 1;
                  start_window_out_sig <= '1';
                  state <= window;
               end if;
            end if;
         end if;

   end case;
end if;
end process;


end Behavioral;
```

6.1.3 WINDOW Block

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

entity main_window is
   Port ( reset_in : in STD_LOGIC;
          clock_in : in STD_LOGIC;
          start_in : in STD_LOGIC;
          ready_out : out STD_LOGIC:='0';
          frame_addr_in : in STD_LOGIC_VECTOR (13 downto 0);
          adc_addr_out : out STD_LOGIC_VECTOR (13 downto 0);
          rom_addr_out: out STD_LOGIC_VECTOR (8 DOWNTO 0);
          ram_addr_out: out STD_LOGIC_VECTOR (8 DOWNTO 0)
   );
end main_window;

architecture Behavioral of main_window is
```

```
signal adc_adr_out: STD_LOGIC_VECTOR (13 downto 0) := "00000000000000";
signal rom_adr_out: STD_LOGIC_VECTOR (8 DOWNTO 0) :="000000000";
signal ram_adr_out: STD_LOGIC_VECTOR (8 DOWNTO 0) :="000000000";
signal counter : STD_LOGIC:= '0' ;


type states is (idle, window); -- States of the FSM
signal state : states := idle; -- State signal signal

begin

ram_addr_out <= ram_adr_out;
rom_addr_out <= rom_adr_out;



windowing : process(clock_in)
begin



if rising_edge(clock_in) then
   case state is
      when idle =>
         ready_out <= '1';
         -- Reset everything
         adc_addr_out <= "00000000000000";
         rom_adr_out <= "000000000";
         ram_adr_out <= "000000000";
         adc_adr_out <= "00000000000000";
         if start_in = '1' then
            adc_adr_out <= frame_addr_in;
            state <= window;
         end if;
      when window =>
         adc_adr_out <= adc_adr_out + 1;
         rom_adr_out <= rom_adr_out + 1;
         ram_adr_out <= ram_adr_out + 1;
         adc_addr_out <= adc_adr_out;
         if rom_adr_out = "111111111" then
            rom_adr_out <= "000000000";
            ram_adr_out <= "000000000";
            ready_out <= '1';
            state <= idle;
         end if;
   end case;
end if;
```

```
end process;
end Behavioral;
```

6.1.4 FFT Block

```
library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_SIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;


entity FFTFSM is
  Port ( reset_in_FFT  : IN STD_LOGIC;
         start_in_FFT  : IN STD_LOGIC;
         i_clock_100   : IN STD_LOGIC;
         FFT_out_valid : IN STD_LOGIC; -- M_AXIS_DATA_tvalid e baðlanacak, outputu çekecek
         FFT_sdata_ready: IN STD_LOGIC; -- S_AXIS_DATA_tready e girecek, word count baþlatacak
         ram_data_FFTWindow : IN STD_LOGIC_VECTOR(19 DOWNTO 0); -- windowdan gelen
data
         FFT_data_in   : IN STD_LOGIC_VECTOR(63 DOWNTO 0); -- FFT den çýkan sonuç
         mul_re_P      : IN STD_LOGIC_VECTOR(59 DOWNTO 0); -- reel mul outputu
         mul_im_P      : IN STD_LOGIC_VECTOR(59 DOWNTO 0); -- im mul outputu
         S_ADD_in      : IN STD_LOGIC_VECTOR(31 DOWNTO 0) ; -- adder outputu 32 bit output
rama yazýlacak

         ram_addr_FFTWindow : OUT STD_LOGIC_VECTOR(8 DOWNTO 0); -- ram address of ram
window
         ram_addr_FFT  : OUT STD_LOGIC_VECTOR(7 DOWNTO 0); -- ram addres of ram in fft to
write 32 bit
         ram_data_FFT  : OUT STD_LOGIC_VECTOR(31 DOWNTO 0); -- 32 bit output rama
yazýlacak
         FFT_data_out  : OUT STD_LOGIC_VECTOR(47 DOWNTO 0); -- FFTye giden data
         FFT_config_out: OUT STD_LOGIC_VECTOR(7 DOWNTO 0); -- FFT config LSB 1 olacak
         mul_re_A      : OUT STD_LOGIC_VECTOR(29 DOWNTO 0); -- reel mul a ve b inputu
         mul_im_A      : OUT STD_LOGIC_VECTOR(29 DOWNTO 0);  -- im mul a ve b inputu
         FFT_data_valid: OUT STD_LOGIC; -- S_AXIS_DATA_tvalid e girecek, input verecek
         write_FFTram  : OUT STD_LOGIC_VECTOR ( 0 to 0 ):="0" ;

         FFT_clocko     : OUT STD_LOGIC := '0';
         FFT_reset      : OUT STD_LOGIC := '0';
         A_ADD_out      : OUT STD_LOGIC_VECTOR(30 DOWNTO 0); -- reel partýn karesinin msb
31 biti addera gidecek
         B_ADD_out      : OUT STD_LOGIC_VECTOR(30 DOWNTO 0); -- im partýn karesinin msb
31 biti addera gidecek
         ready_out_FFT  : OUT STD_LOGIC := '1'

         );
```

```
end FFTFSM;

architecture Behavioral of FFTFSM is

signal FFT_clock_counter : integer range 0 to 9 := 0;
signal word_counter : integer range 0 to 513 := 0;
signal mul_counter : integer range 0 to 257 := 0;
signal mul_countera : integer range 0 to 256 := 1;

signal FFT_clock  : std_logic := '0'; -- 10 MHz
signal FFT_resets  : std_logic := '0'; -- fft reset
signal start_in_FFTs   : std_logic;
signal ready_out_FFTs  : std_logic := '1' ;
signal start_load_FFTs : std_logic := '0' ;
signal reset_in_FFTs   : std_logic := '0' ;

signal FFT_out_valids  : std_logic := '0' ; -- FFtden gelen valid signal M_AXIS_DATA_tvalid e
baðlanacak, outputu çekecek
signal FFT_sdata_readys: STD_LOGIC; -- S_AXIS_DATA_tready e girecek, word count baþlatacak

signal ram_data_FFTWs      : STD_LOGIC_VECTOR (19 DOWNTO 0);
signal FFT_data_ins        : STD_LOGIC_VECTOR (63 DOWNTO 0);
signal mul_re_Ps           : STD_LOGIC_VECTOR(59 DOWNTO 0);
signal mul_im_Ps           : STD_LOGIC_VECTOR(59 DOWNTO 0);

signal ram_addr_FFTWs      : STD_LOGIC_VECTOR (8 DOWNTO 0):= "000000000"; -- out
signal ram_addr_FFTs       : STD_LOGIC_VECTOR (7 DOWNTO 0):= "00000000";
signal ram_data_FFTs       : STD_LOGIC_VECTOR (31 DOWNTO 0);
signal FFT_data_outSignal  : STD_LOGIC_VECTOR (47 DOWNTO 0):= x"000000000000";
signal FFT_config_Signal   : STD_LOGIC_VECTOR (7 DOWNTO 0);
signal mul_re_As           : STD_LOGIC_VECTOR(29 DOWNTO 0);
signal mul_im_As           : STD_LOGIC_VECTOR(29 DOWNTO 0);
signal A_ADD_outs          : STD_LOGIC_VECTOR(30 DOWNTO 0);
signal B_ADD_outs          : STD_LOGIC_VECTOR(30 DOWNTO 0);
signal A_ADD_outx          : STD_LOGIC_VECTOR(30 DOWNTO 0);
signal B_ADD_outx          : STD_LOGIC_VECTOR(30 DOWNTO 0);
signal FFT_data_valids     : std_logic := '1' ; -- S_AXIS_DATA_tvalid e girecek, input verecek
signal write_FFTrams       : STD_LOGIC_VECTOR ( 0 to 0 ):="0" ;


begin

FFT_config_Signal <= "00000001" ;
FFT_config_out <= FFT_config_Signal ;

ram_data_FFTWs <= ram_data_FFTWindow;  -- input
FFT_data_ins <= FFT_data_in;
mul_re_Ps <= mul_re_P;
```

```
mul_im_Ps <= mul_im_P;
start_in_FFTs <= start_in_FFT;
FFT_out_valids <= FFT_out_valid;
FFT_sdata_readys <= FFT_sdata_ready;
reset_in_FFTs <= reset_in_FFT;

ram_addr_FFTWindow <= ram_addr_FFTWs; -- output
ram_addr_FFT <= ram_addr_FFTs;
ram_data_FFT <= ram_data_FFTs;
FFT_data_out <= FFT_data_outSignal;

ready_out_FFT <= ready_out_FFTs;
FFT_clocko <= FFT_clock;
FFT_data_valid <= FFT_data_valids;
FFT_reset <= FFT_resets;
write_FFTram <= write_FFTrams;

FFT_data_outSignal(19 downto 0) <= ram_data_FFTWs;


mul_re_A <= mul_re_As;
mul_im_A <= mul_im_As;

A_ADD_outs <= mul_re_Ps(55 downto 25); -- Multiplierdan reelin karesinin MSB 31 bitini addera
ver
B_ADD_outs <= mul_im_Ps(55 downto 25); -- Multiplierdan imaginerin karesinin MSB 31 bitini
addera ver

A_ADD_out <= A_ADD_outx ;
B_ADD_out <= B_ADD_outx ;


FFTclk_signal: process(i_clock_100)
begin
   if rising_edge(i_clock_100) then
      if FFT_clock_counter = 4 then
        FFT_clock_counter <= FFT_clock_counter + 1;
        FFT_clock <= '0';
      elsif FFT_clock_counter = 9 then
        FFT_clock_counter <= 0 ;
        FFT_clock <= '1';
      else
        FFT_clock_counter <= FFT_clock_counter + 1;
      end if;
   end if;
end process FFTclk_signal;

starter: process(i_clock_100)
```

```
begin
   if rising_edge(i_clock_100) then -- mul counter ram adress üzerinden readyoutu bir yap tekrar
     if mul_counter = 0 then
         if start_in_FFTs = '1' and reset_in_FFTs = '0' then
         ready_out_FFTs <= '0' ;
         FFT_resets <= '1';
         elsif reset_in_FFTs = '1' then
         ready_out_FFTs <= '1' ;
         FFT_resets <= '0';
         end if;
     elsif mul_counter = 257 or reset_in_FFTs = '1' then
     ready_out_FFTs <= '1' ;
     FFT_resets <= '0';
     end if;
   end if;
end process starter;


FFTWORDCOUNTER: process (i_clock_100,FFT_clock)
begin
   if  rising_edge(FFT_clock) then
      if FFT_sdata_readys = '1' then
            if word_counter < 513 then
            word_counter <= word_counter + 1;
            else
            word_counter <= 0;
            end if;
      else
      word_counter <= 0;
        end if;
   end if;

end process FFTWORDCOUNTER;

FFTMULCOUNTER: process (i_clock_100,FFT_clock)
begin
   if  rising_edge(FFT_clock) then
      if FFT_out_valids = '1' then
         if mul_counter < 257 then
         mul_counter <= mul_counter + 1;
         ELSE
         mul_counter <= 0;
         end if;
      else
      mul_counter <= 0;
      end if;

   end if;
```

```
end process FFTMULCOUNTER;

FFTDATALOADER: process (i_clock_100)
begin
   if  rising_edge(i_clock_100) then
      if FFT_sdata_readys = '1' then
         if word_counter = 0  then
            if FFT_clock_counter = 1 then
            ram_addr_FFTWs <= ram_addr_FFTWs + 1;

            end if;
         elsif word_counter > 0 and word_counter < 512 then
            if FFT_clock_counter = 1 then
            ram_addr_FFTWs <= ram_addr_FFTWs + 1;

            end if;
         elsif word_counter = 512 then


         end if;
      else
      ram_addr_FFTWs <= "000000000" ;
      end if;
   end if;


end process FFTDATALOADER;

FFTDATARECIEVER: process (i_clock_100)
begin
   if  rising_edge(i_clock_100) then
         if FFT_out_valids = '1' then

            if mul_counter = 0  then
               if FFT_clock_counter = 9 then
               mul_re_As <= FFT_data_ins(29 downto 0);
               mul_im_As <= FFT_data_ins(61 downto 32);
               ram_addr_FFTs <= "00000000" ;
               write_FFTrams <= "1";
               end if;

            elsif mul_counter = 1 then
               if FFT_clock_counter = 1 then -- FFT OUTPUTU MULTIPLIERA VER
                  mul_re_A <= mul_re_As;
                  mul_im_A <= mul_im_As;

               elsif FFT_clock_counter = 3 then -- MULDAN ADDERA VER
```

```
      A_ADD_outx <= A_ADD_outs;
      B_ADD_outx <= B_ADD_outs;

    elsif FFT_clock_counter = 7 then -- ADDERDAN RAME YAZ
      ram_data_FFTs <= S_ADD_in;

    elsif FFT_clock_counter = 9 then -- YENÝ DATAYI ÇEK RAMI UPDATELE
      mul_re_As <= FFT_data_ins(29 downto 0);
      mul_im_As <= FFT_data_ins(61 downto 32);
      ram_addr_FFTs <= ram_addr_FFTs + 1 ;

    end if;

  elsif mul_counter > 1 and mul_counter < 256 then
    if FFT_clock_counter = 1 then
      mul_re_A <= mul_re_As;
      mul_im_A <= mul_im_As;

    elsif FFT_clock_counter = 3 then
      A_ADD_outx <= A_ADD_outs;
      B_ADD_outx <= B_ADD_outs;

    elsif FFT_clock_counter = 7 then
      ram_data_FFTs <= S_ADD_in;

    elsif FFT_clock_counter = 9 then
      mul_re_As <= FFT_data_ins(29 downto 0);
      mul_im_As <= FFT_data_ins(61 downto 32);
      ram_addr_FFTs <= ram_addr_FFTs + 1 ;

    end if;
  elsif mul_counter = 256 then
    if FFT_clock_counter = 1 then
      mul_re_A <= mul_re_As;
      mul_im_A <= mul_im_As;

    elsif FFT_clock_counter = 3 then
      A_ADD_outx <= A_ADD_outs;
      B_ADD_outx <= B_ADD_outs;

    elsif FFT_clock_counter = 7 then
      ram_data_FFTs <= S_ADD_in;

    elsif FFT_clock_counter = 9 then
      write_FFTrams <= "0";

    end if;
  elsif mul_counter = 257 then
```

```
        end if;
      end if;
   end if;


end process FFTDATARECIEVER;
end Behavioral;
```

6.1.5 MEL Block

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;



entity MEL_logic is
   Port ( i_clk : in STD_LOGIC; -- 100MHz clock. 10n period. State change every 5n.
          i_start : in STD_LOGIC; -- Start when high.
          i_reset : in STD_LOGIC; -- Reset all registers.
          o_ready : out STD_LOGIC; -- High when ready to get data from FFT.
          o_write_enable : out STD_LOGIC_VECTOR (0 DOWNTO 0); -- High when writing energies
          o_fft_addr : out STD_LOGIC_VECTOR (7 DOWNTO 0); -- Addr to read from dual-port
RAM in Lab-FFT.     maximum value: 255
          i_fft_data : in STD_LOGIC_VECTOR (31 DOWNTO 0); -- Data from dual-port RAM in
Lab-FFT. (Frequency spectrum)
          o_energy_addr : out STD_LOGIC_VECTOR (3 DOWNTO 0); -- Addr to write to dual-port
RAM in Lab-MEL.     maximum value: 15
          o_energy_data : out STD_LOGIC_VECTOR (47 DOWNTO 0); -- Data to dual-port RAM in
Lab-MEL. (Energies)
          o_coe_addr : out STD_LOGIC_VECTOR (11 DOWNTO 0); -- Addr to read from ROM
containing filter bank coefficients.     maximum value: 2671
          i_coe_data : in STD_LOGIC_VECTOR (7 DOWNTO 0) -- Data from ROM containing filter
bank coefficients.
        );
end MEL_logic;



architecture Behavioral of MEL_logic is

-- Signals
signal slow_clk : STD_LOGIC;
signal counter_slow_clk : UNSIGNED (3 DOWNTO 0) := "0000";

type states is (idle, working); -- States of the FSM
```

```
signal state : states := idle; -- State signal initialization

signal energy_accumulator : UNSIGNED (47 DOWNTO 0) := x"000000000000"; -- Accumulate all
energy for one filter

signal counter_coe : UNSIGNED (11 DOWNTO 0) := x"000"; -- Coefficient counter     maximum
value: 2671

signal ready_sig : STD_LOGIC;

begin

-- Combinational logic
o_energy_addr <= STD_LOGIC_VECTOR(to_unsigned((to_integer(counter_coe)) / 167,
o_energy_addr'length));
o_fft_addr <= STD_LOGIC_VECTOR(to_unsigned(((to_integer(counter_coe)) mod 167) + 3,
o_fft_addr'length));
o_coe_addr <= STD_LOGIC_VECTOR(counter_coe);

slow_clk <= counter_slow_clk (1);

o_ready <= ready_sig;

o_energy_data <= STD_LOGIC_VECTOR(energy_accumulator);



process(i_clk)
begin
   if rising_edge(i_clk) then
      counter_slow_clk <= counter_slow_clk + 1;
   end if;
end process;



process(i_clk)
begin
   if rising_edge(i_clk) then
      if counter_coe = 2671 then
         ready_sig <= '1';
      else
         if i_start = '1' then
            ready_sig <= '0';
         end if;
      end if;
   end if;
end process;
```

```
process(slow_clk)
begin
if rising_edge(slow_clk) then
   -- Check reset
   if i_reset = '1' then
      state <= idle;
   end if;

   -- States and what they do
   if state = idle then
      -- Reset all registers
      energy_accumulator <= x"000000000000";
      counter_coe <= x"000";
      o_write_enable <= "0";
      -- Wait for i_start and change required signals
      if ready_sig = '0' then
         state <= working;
      end if;
   elsif state = working then
      -- Multiply one value of FFT with one value of filter coefficients
      -- Add the multiplied value to energy_accumulator
      -- Loop through 167 fft values
      -- Write energy_accumulator to o_energy_data
      -- Loop through 16 energy values
      if counter_coe = 2671 then -- No more work to do
         state <= idle;
      else
         counter_coe <= counter_coe + 1; -- Increment coe_counter by 1
         energy_accumulator <= energy_accumulator + (UNSIGNED(i_fft_data) *
UNSIGNED(i_coe_data));
         if (counter_coe + 1) mod 167 = 166 then -- Write enable high
            o_write_enable <= "1";
         end if;
         if (counter_coe + 1) mod 167 = 0 then -- Next filter
            energy_accumulator <= x"000000000000";
            o_write_enable <= "0";
         end if;
      end if;
   else -- State is not idle nor working. ERROR
      state <= idle;
   end if;
end if;

end process;
```

```
end Behavioral;
```

6.1.6 DEBUG Block

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;


entity lab_debug is
   port ( i_clk_100 : IN STD_LOGIC;
          i_start : IN STD_LOGIC;
          i_reset : IN STD_LOGIC;
          i_mem_data : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
          o_ready : OUT STD_LOGIC;
          o_mem_addr : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
          o_txd : OUT STD_LOGIC
   );
end lab_debug;


architecture Behavioral of lab_debug is

type states is (idle, transfer); -- States of the FSM
signal state : states := idle; -- State signal initialization

signal baud_clk : std_logic := '0';
signal debug_starter_s : std_logic := '0';
signal baud_counter : integer range 0 to 867 := 0;

signal start_str : std_logic_vector (31 downto 0) := x"55aacc03";
signal stop_str : std_logic_vector (31 downto 0) := x"aa5503cc";

signal addr_counter : integer range 0 to 17 := 0;
signal byte_counter : integer range 0 to 3 := 0;
signal bit_counter : integer range 0 to 9 := 0;

signal mem_addr : std_logic_vector (3 downto 0) := "0000";
signal mem_data : std_logic_vector (31 downto 0) := x"00000000";



begin

start_str <= x"55aacc03";
stop_str <= x"aa5503cc";
```

```
o_mem_addr <= mem_addr;
mem_data <= i_mem_data;

baud_signal: process(i_clk_100)
begin
   if rising_edge(i_clk_100) then
      if baud_counter = 433 then
         baud_counter <= baud_counter + 1;
         baud_clk <= '0';
      elsif baud_counter = 867 then
         baud_counter <= 0;
         baud_clk <= '1';
      else
         baud_counter <= baud_counter + 1;
      end if;
   end if;
end process baud_signal;

starter: process (i_clk_100)
begin
if rising_edge(i_clk_100) then
   if i_start = '0' and baud_counter = 0 then
      debug_starter_s <= '0';
   else
      if i_start = '1' then
         debug_starter_s <= '1';
      end if;
   end if;
end if;

end process starter;

UART_counters: process(baud_clk)
begin
          if state = transfer then
                    if rising_edge(baud_clk) then
                              if bit_counter = 9 then
                                        bit_counter <= 0;
                                        if byte_counter = 3 then
                                                  byte_counter <= 0;
                                                  if addr_counter = 17 then
                                                            addr_counter <= 0;
                                                  else
                                                            addr_counter <= addr_counter + 1;
                                                  end if;
                                        else
                                                  byte_counter <= byte_counter + 1;
                                        end if;
```

```
                        else
                                bit_counter <= bit_counter + 1;
                        end if;
                end if;
        elsif state = idle then
          addr_counter <= 0;
     byte_counter <= 0;
          bit_counter <= 0;
        end if;
end process UART_counters;

ROM_management: process(baud_clk)
begin
         if falling_edge(baud_clk) then
      if bit_counter = 9 and byte_counter = 3 and addr_counter /= 0 then
        mem_addr <= mem_addr + 1;
      elsif state = idle then
        mem_addr <= "0000";
      end if;
         end if;
end process ROM_management;

UART_tx: process(i_clk_100)
begin
   if i_reset = '1' then
      state <= idle;
   end if;

   if rising_edge(i_clk_100) then
      case state is
         when idle =>
                                o_txd <= '1';
                                o_ready <= '1';
                                -- Reset everything

           if debug_starter_s = '1' then
              state <= transfer;
              o_ready <= '0';
           end if;
         when transfer => -- Send start string, ROM, stop string
           if baud_counter = 867 then
              if addr_counter = 0 then -- Send start string
                 if bit_counter = 0 then
                    o_txd <= '0';
                 elsif bit_counter = 9 then
                    o_txd <= '1';
                 else
                    o_txd <= start_str((bit_counter - 1) + (byte_counter * 8));
```

```
                end if;
            elsif addr_counter = 17 then -- Send stop string, state to idle
                if bit_counter = 0 then
                    o_txd <= '0';
                elsif bit_counter = 9 then
                    o_txd <= '1';
                    if byte_counter = 3 then
                        state <= idle;
                    end if;
                else
                    o_txd <= stop_str((bit_counter - 1) + (byte_counter * 8));
                end if;
            else -- Send ROM data
                if bit_counter = 0 then
                    o_txd <= '0';
                elsif bit_counter = 9 then
                    o_txd <= '1';
                else
                    o_txd <= mem_data((bit_counter - 1) + (byte_counter * 8));
                end if;
            end if;
        end if;
      when others =>
        state <= idle;
    end case;
  end if;
end process UART_tx;

end Behavioral;
```

6.1.7 TOP Level

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity top is
  Port ( reset_in : in STD_LOGIC;
       clock_in : in STD_LOGIC;
       start_in : in STD_LOGIC;
       ready_out : out STD_LOGIC;
       data_out : out STD_LOGIC;
       spi_clk_out: OUT STD_LOGIC;
       chip_sel_out: OUT STD_LOGIC;
       spi_data_in: IN STD_LOGIC
       );
end top;
```

```
architecture Behavioral of top is

signal frame_sig : std_logic_vector(13 downto 0);
signal bos: std_logic;
signal dolu: std_logic :='1';
signal bos1: std_logic_vector(13 downto 0);
signal bos2: std_logic_vector(11 downto 0);
signal bos3: std_logic_vector(8 downto 0);
signal bos4: std_logic_vector(19 downto 0);
signal start_window_sig : std_logic;
signal ready_window_sig: std_logic;
signal window_ram_adr_sig : std_logic_vector(13 downto 0);
signal window_ram_data_sig : std_logic_vector(19 downto 0);
signal data_out_sig : std_logic;
signal start_debug_sig: std_logic;
signal ready_debug_sig: std_logic;
signal start_fft_sig: std_logic;
signal ready_fft_sig: std_logic;
signal start_mel_sig: std_logic;
signal ready_mel_sig: std_logic;
signal start_adc_sig: std_logic;
signal ready_adc_sig: std_logic;
signal spi_clk_out_sig : std_logic;
signal chip_sel_out_sig : std_logic;
signal spi_data_in_sig: std_logic;
signal adc_data_out_sig: std_logic_vector(31 downto 0);
signal adc_adr_in_sig: std_logic_vector(13 downto 0);
signal mel_debug_ram_addr_sig :  STD_LOGIC_VECTOR ( 3 downto 0 );
signal mel_debug_ram_data_sig :STD_LOGIC_VECTOR ( 31 downto 0 );
signal mel_fft_ram_addr_sig :  STD_LOGIC_VECTOR ( 7 downto 0 );
signal mel_fft_ram_data_sig :STD_LOGIC_VECTOR ( 31 downto 0 );

component main is
    Port ( reset_in : in STD_LOGIC;
           clock_in : in STD_LOGIC;
           start_adc_out : out STD_LOGIC;
           ready_adc_in : in STD_LOGIC;
           frame_addr_out : out STD_LOGIC_VECTOR (13 downto 0);
           start_window_out : out STD_LOGIC;
           ready_window_in : in STD_LOGIC;
           start_fft_out : out STD_LOGIC;
           ready_fft_in : in STD_LOGIC;
           start_mel_out : out STD_LOGIC;
           ready_mel_in : in STD_LOGIC;
           start_dct_out : out STD_LOGIC;
           ready_dct_in : in STD_LOGIC;
```

```
        start_comp_out : out STD_LOGIC;
        ready_comp_in : in STD_LOGIC;
        start_debug_out : out STD_LOGIC;
        ready_debug_in : in STD_LOGIC;
        start_in : in STD_LOGIC;
        ready_out : out STD_LOGIC
        --spi_clk_out: OUT STD_LOGIC;
        --chip_sel_out: OUT STD_LOGIC;
        --spi_data_in: IN STD_LOGIC

        );
end component main;

component lab_window is
   port ( reset_in : in STD_LOGIC;
        clock_in : in STD_LOGIC;
        frame_addr_in : in STD_LOGIC_VECTOR (13 downto 0);
        adc_addr_out : out STD_LOGIC_VECTOR (13 downto 0);
        adc_data_in : in STD_LOGIC_VECTOR (11 downto 0);
        mem_addr_in : in STD_LOGIC_VECTOR (8 downto 0);
        mem_data_out : out STD_LOGIC_VECTOR (19 downto 0);
        start_in : in STD_LOGIC;
        ready_out : out STD_LOGIC
   );
end component lab_window;

component lab_debug is
   port  (i_clk_100 : IN STD_LOGIC;
        i_start : IN STD_LOGIC;
        i_reset : IN STD_LOGIC;
        i_mem_data : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        o_ready : OUT STD_LOGIC;
        o_mem_addr : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
        o_txd : OUT STD_LOGIC);
end component lab_debug;

component lab_adc is
   Port (
   reset_in: IN STD_LOGIC;
   i_clock_100: IN STD_LOGIC;
   spi_clk_out: OUT STD_LOGIC;
   chip_sel_out: OUT STD_LOGIC;
   spi_data_in: IN STD_LOGIC;
   start_in: IN STD_LOGIC;
   ready_outadc: OUT STD_LOGIC;
   ram_data_out : OUT STD_LOGIC_VECTOR(31 downto 0);
   ram_addr_in : IN STD_LOGIC_VECTOR (13 DOWNTO 0)
    );
```

```
end component lab_adc;

component TOPFFT is
Port (   reset_in_FFT  : IN STD_LOGIC;
        start_in_FFT  : IN STD_LOGIC;
        i_clock_100   : IN STD_LOGIC;
        ram_data_FFTW : IN STD_LOGIC_VECTOR(19 DOWNTO 0);-- windowdan gelen data

        ramB_addr_FFT  : IN STD_LOGIC_VECTOR(7 DOWNTO 0); -- ram addres of ram in fft to
READ 32 bit
        ram_addr_FFTW  : OUT STD_LOGIC_VECTOR(8 DOWNTO 0); -- ram address of ram
window
        ramb_data_FFT  : OUT STD_LOGIC_VECTOR(31 DOWNTO 0); -- 32 bit output ramDEN
OKUNACAK
        ready_out_FFT  : OUT STD_LOGIC

 );
end component TOPFFT;

component MEL_top is
   Port ( i_clk : in STD_LOGIC; -- 100MHz clock. 10n period. State change every 5n.
          i_start : in STD_LOGIC; -- Start when high.
          i_reset : in STD_LOGIC; -- Reset all registers.
          o_ready : out STD_LOGIC; -- High when ready to get data from FFT.
          o_fft_addr : out STD_LOGIC_VECTOR (7 DOWNTO 0); -- Addr to read from dual-port
RAM in Lab-FFT.
          i_fft_data : in STD_LOGIC_VECTOR (31 DOWNTO 0); -- Data from dual-port RAM in
Lab-FFT. (Frequency spectrum)
          i_energy_addr : in STD_LOGIC_VECTOR (3 DOWNTO 0); -- Addr to read from dual-port
RAM in Lab-MEL.     maximum value: 15
          o_energy_data : out STD_LOGIC_VECTOR (31 DOWNTO 0) -- Data from dual-port RAM in
Lab-MEL. (Energies)
        );
end component;


begin
data_out <= data_out_sig;

main_logic: main
   port map(   reset_in => reset_in,
               clock_in => clock_in,
               start_adc_out => start_adc_sig,
               ready_adc_in => ready_adc_sig,
               frame_addr_out(13 DOWNTO 0) => frame_sig(13 DOWNTO 0),
               start_window_out => start_window_sig,
               ready_window_in => ready_window_sig,
               start_fft_out => start_fft_sig,
```

```
            ready_fft_in => ready_fft_sig,
            start_mel_out => start_mel_sig,
            ready_mel_in => ready_mel_sig,
            start_dct_out => bos,
            ready_dct_in => dolu,
            start_comp_out => bos,
            ready_comp_in => dolu,
            start_debug_out => start_debug_sig,
            ready_debug_in => ready_debug_sig,
            start_in => start_in,
            ready_out => ready_out
    );

component_window : lab_window
    port map (  reset_in => reset_in,
            clock_in => clock_in,
            frame_addr_in(13 downto 0) => frame_sig(13 downto 0),
            adc_addr_out(13 downto 0) => adc_adr_in_sig(13 downto 0),
            adc_data_in(11 downto 0) => adc_data_out_sig(11 downto 0),
            mem_addr_in(8 downto 0) => window_ram_adr_sig(8 downto 0),
            mem_data_out(19 downto 0) => window_ram_data_sig(19 downto 0),
            start_in => start_window_sig,
            ready_out => ready_window_sig
    );

component_debug : lab_debug
    port map (  i_clk_100 => clock_in,
            i_start => start_debug_sig,
            i_reset => reset_in,
            i_mem_data(31 DOWNTO 0) => mel_debug_ram_data_sig,
            o_ready => ready_debug_sig,
            o_mem_addr(3 DOWNTO 0) => mel_debug_ram_addr_sig,
            o_txd => data_out_sig

    );

component_adc : lab_adc
    port map (
        reset_in => reset_in,
        start_in => start_adc_sig,
        i_clock_100 => clock_in,
        spi_clk_out => spi_clk_out,
        chip_sel_out => chip_sel_out,
        ready_outadc => ready_adc_sig,
        spi_data_in => spi_data_in,
        ram_addr_in(13 DOWNTO 0) => adc_adr_in_sig(13 DOWNTO 0),
        ram_data_out(31 DOWNTO 0) => adc_data_out_sig(31 DOWNTO 0)
```

```
   );

 component_fft : TOPFFT
  Port map (
        reset_in_FFT  => reset_in,
        start_in_FFT  => start_fft_sig,
        i_clock_100   => clock_in,
        ram_data_FFTW => window_ram_data_sig(19 downto 0),

        ramB_addr_FFT  => mel_fft_ram_addr_sig,
        ram_addr_FFTW  => window_ram_adr_sig(8 downto 0),
        ramb_data_FFT  => mel_fft_ram_data_sig,
        ready_out_FFT  => ready_fft_sig

 );

component_mel : MEL_top
   Port map(
         i_clk => clock_in,
         i_start => start_mel_sig,
         i_reset => reset_in,
         o_ready => ready_mel_sig,
         o_fft_addr => mel_fft_ram_addr_sig,
         i_fft_data => mel_fft_ram_data_sig,
         i_energy_addr => mel_debug_ram_addr_sig,
         o_energy_data => mel_debug_ram_data_sig
        );
end Behavioral;
```

6.2 MATLAB CODE

6.2.1 DCT and COMPARISON Blocks

```
% ---------- CREATE REFERENCE VALUES ----------:
% Create empty reference storage
reference_storage = [];

% Convert timetable data to matrix form
serialport_matrix = cell2mat(serialport_data1.Data);

% Loop over each reference recording
for i = 1 : size(serialport_matrix, 1)

% Separate each batch (16) of energy.
% 64 rows of 16 energies.
energies = reshape(serialport_matrix(i, :), [18, 64]); % Reshape so every column is one frame
energies = transpose(energies); % Transpose so every row is one frame
energies(:, 1) = []; % Get rid of start numbers
```

```
energies(:, end) = []; % Get rid of stop numbers
% Normalize by triangular filter area.
energies(:, 1) = energies(:, 1) / 7;
energies(:, 2) = energies(:, 2) / 7;
energies(:, 3) = energies(:, 3) / 9;
energies(:, 4) = energies(:, 4) / 10;
energies(:, 5) = energies(:, 5) / 11;
energies(:, 6) = energies(:, 6) / 12;
energies(:, 7) = energies(:, 7) / 14;
energies(:, 8) = energies(:, 8) / 16;
energies(:, 9) = energies(:, 9) / 17;
energies(:, 10) = energies(:, 10) / 20;
energies(:, 11) = energies(:, 11) / 23;
energies(:, 12) = energies(:, 12) / 25;
energies(:, 13) = energies(:, 13) / 28;
energies(:, 14) = energies(:, 14) / 32;
energies(:, 15) = energies(:, 15) / 36;
energies(:, 16) = energies(:, 16) / 41;
% Take logarithm of each energy.
energies = 10*log10(energies);
% Discrete cosine transform.
energies = dct(energies, [], 2);
%energies = dct2(energies);

% Add processed features to reference storage
reference_storage = cat(3, reference_storage, energies);

end

% ---------- COMPARE BLOCK ----------:
% Separate each batch (16) of energy.
% 64 rows of 16 energies.
energies = reshape(serialport_data1, [18, 64]); % Reshape so every column is one frame
energies = transpose(energies); % Transpose so every row is one frame
energies(:, 1) = []; % Get rid of start numbers
energies(:, end) = []; % Get rid of stop numbers

% Normalize by triangular filter area.
energies(:, 1) = energies(:, 1) / 7;
energies(:, 2) = energies(:, 2) / 7;
energies(:, 3) = energies(:, 3) / 9;
energies(:, 4) = energies(:, 4) / 10;
energies(:, 5) = energies(:, 5) / 11;
energies(:, 6) = energies(:, 6) / 12;
energies(:, 7) = energies(:, 7) / 14;
energies(:, 8) = energies(:, 8) / 16;
energies(:, 9) = energies(:, 9) / 17;
energies(:, 10) = energies(:, 10) / 20;
```

```
energies(:, 11) = energies(:, 11) / 23;
energies(:, 12) = energies(:, 12) / 25;
energies(:, 13) = energies(:, 13) / 28;
energies(:, 14) = energies(:, 14) / 32;
energies(:, 15) = energies(:, 15) / 36;
energies(:, 16) = energies(:, 16) / 41;

% Take logarithm of each energy.
energies = 10*log10(energies);

% Discrete cosine transform.
energies = dct(energies, [], 2);
%energies = dct2(energies);

% Compare the feature matrices.
distance_storage = [];

for i = 1 : size(reference_storage, 3) % Get distance from every reference
distance = (norm(reference_storage(:, :, i) - energies))^(2);
distance_storage(i) = distance;
end

[min_distance, min_distance_index] = min(distance_storage);

mod(min_distance_index, 10)
```